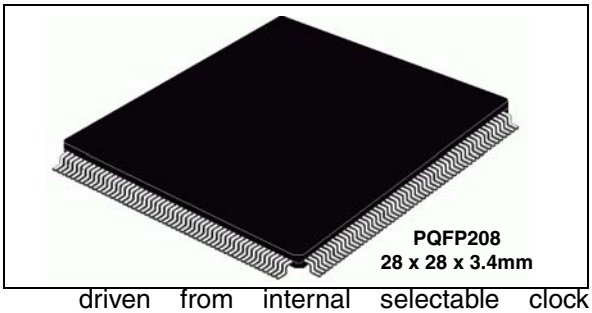
STR720

ARM720T™ 16/32-BIT MCU WITH 16K RAM, USB, CAN, 3 TIMERS, ADC, 6 COMMUNICATIONS INTERFACES

PRELIMINARY DATA



- **ARM720T MCU**
  - 32-bit RISC MCU with 3-stage pipeline.
  - Max. CPU frequency 70 MHz
  - Fully instructions compatible with the ARM7 family of processors
  - 8 KByte Instruction + Data cache, 4-way set-associative
  - Write buffer de-coupling CPU from system memory during write operations
  - MMU for virtual to physical address mapping and memory protection
- **Memories**
  - 16 KBytes Program RAM Memory
  - External SDRAM Interface for up to 128 Mbytes SDRAM
  - External Memory Interface (EMI) for up to 8 Mbytes SRAM, Flash, ROM.
  - ATAPI interface supporting PIO4 mode
- **Nested interrupt controller**
  - Fast interrupt handling with multiple vectors
  - 32 vectors with 16 IRQ priority levels
  - 2 maskable FIQ sources
- **Clock, Reset and Supply Management**
  - Internal system clocks generated by fully internal PLL
  - Power management providing different operating modes: RUN, SLOW, STOP, STANDBY.
- **34 I/O ports**
  - 34 multifunctional bidirectional I/O lines
  - 5 ports with interrupt capability
- **3 Timers**
  - Two 16-bit programmable Timer modules with prescaler (f_{APB} divided by 1 to 256) driven from internal selectable clock (oscillator or CPU clock), output compare and input capture functions.
  - 16-bit Watchdog Timer with 8-bit prescaler
- **6 Communications Interfaces**
  - USB v 2.0 Full Speed (12Mbit/s) Device Function with Suspend and Resume support
  - CAN module compliant with the CAN specification V2.0 part B (active).
  - Two High-Speed Universal Asynchronous Receiver Transmitter (UART) for full-duplex asynchronous communication.
  - Two Buffered Serial Peripheral Interfaces (BSPI) for full-duplex, synchronous, communications with external devices, master or slave operation.
- **A/D Converter**
  - Sigma Delta Analog/Digital Converter with 11.5-bit ENOB resolution supporting 4 multiplexed inputs at up to 950 Hz sampling rate.
- **Development Tools Support**
  - 5-pin JTAG port (IEEE 1149.1 Standard)

Table 1. Device Summary

Features	STR720RBQ6	STR720RXH6
Boot ROM - bytes	4K	
Program RAM - bytes	16K	
Operating Voltage	3.0 to 3.6V for I/Os and A/D, 1.8V for core and backup block	
Operating Temperature	-40 to +85°C	
Package	PQFP208 28x28x3.4	BGA (contact marketing for details)

Rev. 4.0

Table of Contents

1 INTRODUCTION

STR720 is an STMicroelectronics new generation super-integrated single-chip device.
It combines the high performance of ARM720T™ CPU microprocessor (revision 3) with high peripheral functionalities and enhanced I/O capabilities. It also provides on-chip high-speed RAM, clock generation via PLL and several embedded custom digital logics. STR720 is software compatible with the ARM processor family.

It is built using 0.18 μm HCMOS8 process, with 1.8 V internal logic voltage and 3.3 V capable I/O lines.

The ARM720T™ is a member of the Advanced RISC Machines (ARM) family of general purpose 32-bit microprocessors combining in a single chip an 8 KByte cache, an enlarged write buffer, and a Memory Management Unit (MMU).

The ARM720T™ belongs to the ARM7 family, making it software-compatible with all the ARM processors.

The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles: its instruction set and related decode mechanism are much simpler than those of micro-programmed Complex Instruction Set Computers (CISC). This simplicity results in a high instruction throughput and impressive real-time interrupt response from a small and cost-effective chip.

The on-chip mixed data and instruction cache, together with the write buffer, substantially rise the average execution speed and reduce the average amount of memory bandwidth required by the processor. This allows the external memory to support Direct Memory Access (DMA) channels with minimal performance loss.

The MMU supports a conventional two-level, page-table structure and a number of extensions that make it ideal for embedded control, UNIX, and object-oriented systems. The allocation of virtual addresses with different task IDs improve performance in task switching operations with the cache enabled. These relocated virtual addresses are monitored by the Embedded-ICE block.

The memory interface is designed to allow the performance potential to be realized without incurring high costs in the memory system. Speed-critical control signals are pipelined to allow system control functions to be implemented in standard low-power logic, and these control signals facilitate the exploitation of the fast local access modes offered by industry-standard DRAMs.

For more information on the ARM720T core please refer to the ARM720T Rev 3 Technical Reference Manual.

Related Documents:

Available from www.arm.com:

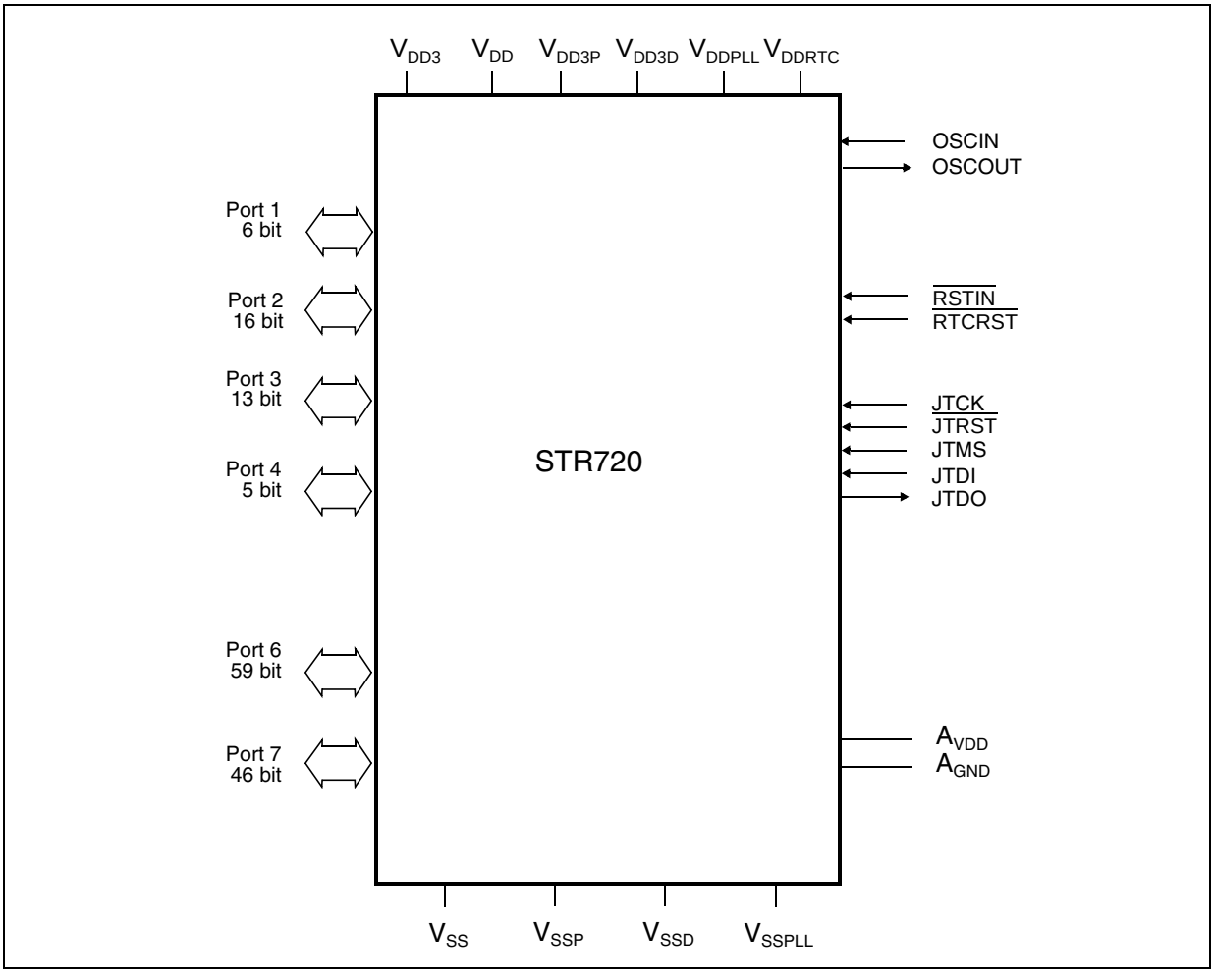
ARM720T (Rev 3) Technical Reference Manual

2 ACRONYMS

- ADC Analog to Digital Converter
- AHB Advanced High-performance Bus
- APB Advanced Peripheral Bus
- CPU Central Processing Unit
- DMA Direct Memory Access
- EIC Enhanced Interrupt Controller
- EFT Extended Function Timer
- EMI External Memory Interface
- GCR Global Configuration Register
- ICE In-Circuit Emulator
- IRQ Interrupt ReQuest
- JTAG Joint Test Access Group (IEEE 1149.1 Standard)
- PLL Phase-Locked Loop
- RAM Random Access Memory
- ROM Read-Only Memory
- RCCU Reset and Clock Control Unit
- RISC Reduced Instruction Set Computing
- SPI Serial Peripheral Interface
- BSPI Buffered SPI
- UART Universal Asynchronous Receiver Transmitter
- WDG Watch-Dog Unit
- WIU Wake-up/Interrupt Unit

Please refer to Figure 1 for an overview of the device interfaces.

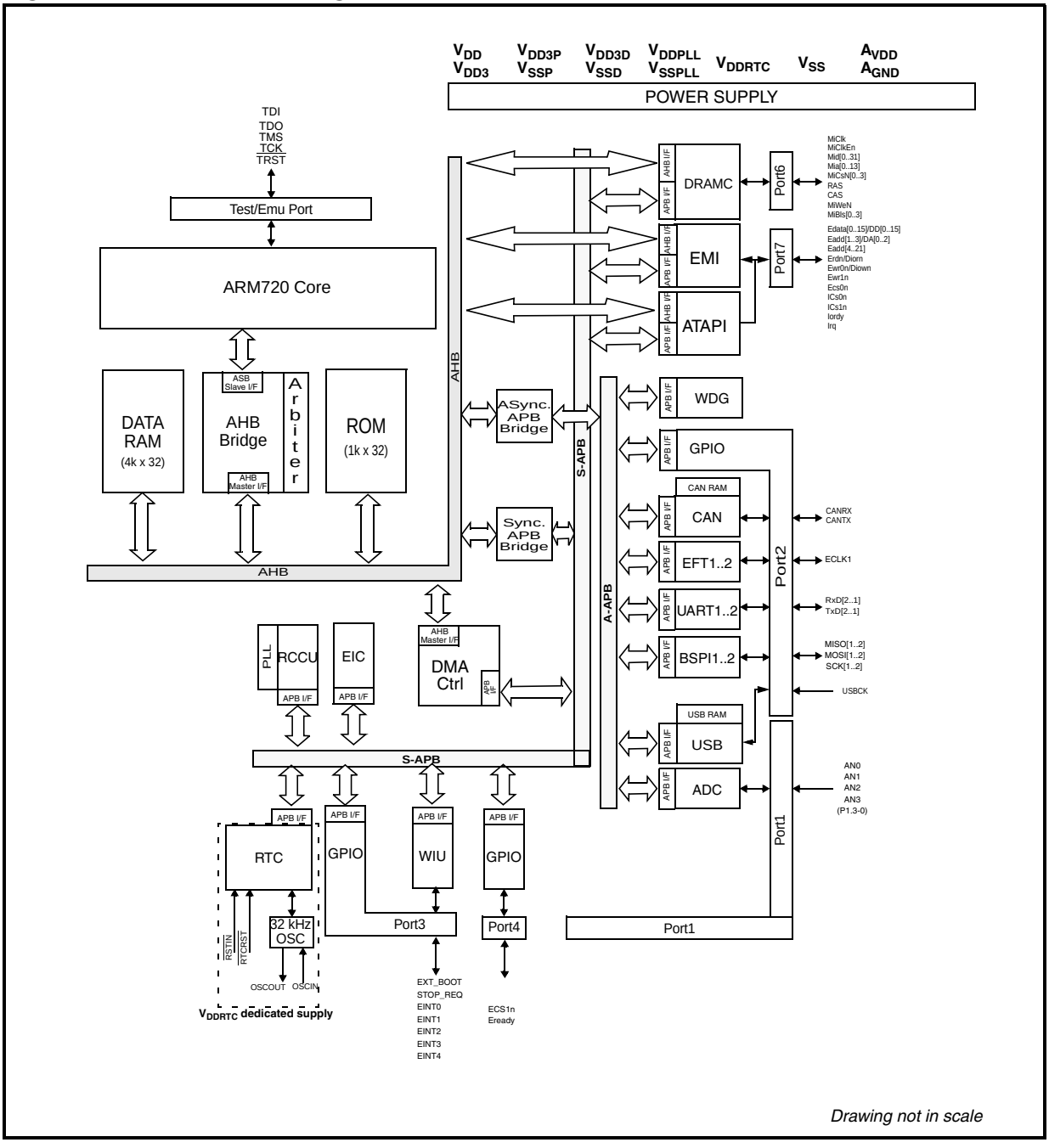
Figure 1. STR720 PQFP Package Logic Symbol



3 SYSTEM BLOCK DIAGRAM

The Figure 2: STR720 Block Diagram on page 13 gives an overview of the complete STR720 microcontroller, showing how ARM720T processor and their peripherals are interfaced.

Figure 2. STR720 Block Diagram



4 PIN DATA

In the following tables, pin information are reported, including alternate function mapping and reference to package pin out.

4.1 Power Supply pins

Table 2. Power Supply Pins

Symbol	Pin	I/O	Function
V_{DD3}	Note1	-	3.3V Digital Supply Voltage for the I/O pads.
V_{DD}	Note2	-	1.8V Digital Supply Voltage for core circuitry.
V_{DD3P}	22	-	3.3V Supply Voltage.
V_{SSP}	23	-	Ground Voltage.
	24 30	-	Reserved, must be tied to GND.
	29 31	-	Reserved, must be tied to GND.
V_{DDPLL}	121	-	1.8 V Supply Voltage for internal PLL.
V_{SSPLL}	120	-	Ground Voltage for internal PLL.
V_{DDRTC}	159	-	1.8V Digital Supply Voltage for RTC logic.
V_{SS}	Note3	-	Digital Ground.
A_{VDD}	32	-	Reference voltage for the on-chip A/D Converter (+3.0V/+3.6V).
A_{GND}	37	-	Reference ground for the on-chip A/D Converter.
	28	-	Reserved, must be tied to GND.
	26	-	Reserved, must be left floating.

Note1: 16 Pins (1,10,46,55,66,75,84,93,103,113,129,143,163,180,196) are V_{DD3} (3.3 supply).
Note2: 3 Pins (8,59,140) are V_{DD} (1.8 supply).
Note3: 19 Pins (2,9,11,47,56,60,67,76,85,94,104,114,130,141,142,162,181,197) are V_{SS} (ground).

4.2 Global Pins

In Table 3 clock, reset and configuration pins of the device are reported.

Table 3. Global Pins

Symbol	Pin	I/O	Function
OSCIN	160	I	Input of the 32 kHz oscillator amplifier and input of the internal clock generator for RTC and WDG alternate clocks.
OSCOUT	161	O	Output of the 32 kHz oscillator amplifier circuit.
$\overline{\text{RSTIN}}$(Note 1)	157	I	System RESET Input with Schmitt-Trigger characteristics. A low level at this pin resets asynchronously the STR720 device, except the RTC logic. This has to be driven low by an external pull-down resistor when V_{DDRTC} is connected and the rest of the device is not powered.

Table 3. Global Pins

Symbol	Pin	I/O	Function
$\overline{\text{RTCRST}}$(Note 1)	158	I	RTC RESET Input with Schmitt-Trigger characteristics. A low level at this pin asynchronously resets the RTC registers.

Note1: The signal applied to this pin is internally filtered by an on-chip RC filter whose width can range from 50 ns to 500 ns.

4.3 JTAG pins

In Table 4 the pins related to the JTAG interface are listed.

Table 4. JTAG/Emu Pins

Symbol	Pin	I/O	Function
JTDO	42	O	ARM720 JTAG Test Data Out. JTDO is a test data serial output signal used for test instructions and data.
JTCK	39	I	ARM720 JTAG Test Clock. JTCK is a test input used to synchronize the JTAG test logic.
JTMS	40	I	ARM720 JTAG Test Mode Select. JTMS is an input signal used to sequence the test controller's state machine. JTMS is sampled on the rising edge of JTCK.
JTDI	41	I	ARM720 JTAG Test Data In. JTDI is test data serial input used for test instructions and data. JTDI is sampled on the rising edge of JTCK.
$\overline{\text{JTRST}}$ (Note 1)	38	I	ARM720 JTAG Test Circuit Reset. $\overline{\text{JTRST}}$ is an active low Schmitt-trigger input signal used to asynchronously initialize the test controller.

Note1: The signal applied to this pin is internally filtered by an on-chip RC filter whose width can range from 50 ns to 500 ns.

4.4 Port 1 pins

Port 1 is an analog port connected to the 4 analog input channels of the on-chip A/D converter are available.

Table 5. Port 1 Pins

Symbol	Pin	I/O	Function
P1.0	33	I	*A/D*. Analog Input Channel 0 (*ANA0*).
P1.1	34	I	*A/D*. Analog Input Channel 1 (*ANA1*).
P1.2	35	I	*A/D*. Analog Input Channel 2 (*ANA2*).
P1.3	36	I	*A/D*. Analog Input Channel 3 (*ANA3*).
P1.4	27	O	Reserved.
P1.5	25	O	Reserved.

4.5 Port 2 pins

Port 2 consists of 16 bidirectional general purpose I/O pins. They are under ARM720 control through a configuration register, a data register and a direction register. These registers are read/write registers. Port lines can be individually configured as general purpose inputs, general purpose outputs or dedicated peripheral lines. The port can be read at any time: input lines return the pin level; output lines return the level of the output driver input. When written, the port stores the data in an internal register: it drives the pins only if they are configured as general purpose outputs. Port 2 outputs can be configured as push/pull or open drain drivers.

Table 6. Port 2 Pins

Symbol	Pin	I/O	Function
P2.0	122	I/O	Port 2 General Purpose Input/Output data line 0.
		I	*UART1*. Receive Data Input (*RxD1*).
P2.1	123	I/O	Port 2 General Purpose Input/Output data line 1.
		O	*UART1*. Transmit Data Output (*TxD1*).
P2.2	124	I/O	Port 2 General Purpose Input/Output data line 2.
		I	*UART2*. Receive Data Input (*RxD2*).
P2.3	125	I/O	Port 2 General Purpose Input/Output data line 3.
		O	*UART2*. Transmit Data Output (*TxD2*).
P2.4	126	I/O	Port 2 General Purpose Input/Output data line 4.
		I	*BSPI1*. Master Mode. Master Input/Slave Output line (*MISO1*).
		O	*BSPI1*. Slave Mode. Master Input/Slave Output line (*MISO1*).
P2.5	127	I/O	Port 2 General Purpose Input/Output data line 5.
		O	*BSPI1*. Master Mode. Master Output/Slave Input line (*MOSI1*).
		I	*BSPI1*. Slave Mode. Master Output/Slave Input line (*MOSI1*).
P2.6	128	I/O	Port 2 General Purpose Input/Output data line 6.
		O	*BSPI*1. Master Mode. Output serial clock (*SCK*1).
		I	*BSPI1*. Slave Mode. Input serial clock (*SCK*1).
P2.7	131	I/O	Port 2 General Purpose Input/Output data line 7.
		I	*BSPI2*. Master Mode. Master Input/Slave Output line (*MISO2*).
		O	*BSPI2*. Slave Mode. Master Input/Slave Output line (*MISO2*).
P2.8	132	I/O	Port 2 General Purpose Input/Output data line 8.
		O	*BSPI2*. Master Mode. Master Output/Slave Input line (*MOSI2*).
		I	*BSPI2*. Slave Mode. Master Output/Slave Input line (*MOSI2*).
P2.9	133	I/O	Port 2 General Purpose Input/Output data line 9.
		O	*BSPI*2. Master Mode. Output serial clock (*SCK*2).
		I	*BSPI2*. Slave Mode. Input serial clock (*SCK*2).

Table 6. Port 2 Pins (Continued)

Symbol	Pin	I/O	Function
P2.10	134	I/O	Port 2 General Purpose Input/Output data line 10.
		I	*USB*. USB interface 48 MHz clock input *(USBCK*).
P2.11	135	I/O	Port 2 General Purpose Input/Output data line 11.
		I	*CAN*. CAN module receive pin *(CANRX*).
P2.12	136	I/O	Port 2 General Purpose Input/Output data line 12.
		O	*CAN*. CAN module transmit pin *(CANTX*).
P2.13	137	I/O	Port 2 General Purpose Input/Output data line 13.
		I	*EFT1*. Timer 1 External Clock Input line *(ECLK1*).
P2.14	138	I/O	Port 2 General Purpose Input/Output data line 14.
		I/O	*USB*. USB interface D- signal *(USBDM*). The usage of this alternate function require a special programming of GPIO registers.
P2.15	139	I/O	Port 2 General Purpose Input/Output data line 15.
		I/O	*USB*. USB interface D+ signal *(USBDP*). The usage of this alternate function require a special programming of GPIO registers.

4.6 Port 3 pins

Port 3 consists of 13 bidirectional general purpose I/O pins. They are controlled through a configuration register, a data register and a direction register. These registers are read/write registers. Port lines can be individually configured as general purpose inputs, general purpose outputs or dedicated peripheral lines. The port can be read at any time: input lines return the pin level; output lines return the level of the output driver input. When written, the port stores the data in an internal register: it drives the pins only if they are configured as general purpose outputs. Port 3 outputs can be configured as push/pull or open drain drivers.

Table 7. Port 3 Pins

Symbol	Pin	I/O	Function
P3.0	144	I/O	Port 3 General Purpose Input/Output line 0.
P3.1	145	I/O	Port 3 General Purpose Input/Output line 1.
P3.2	146	I/O	Port 3 General Purpose Input/Output line 2.
P3.3	147	I/O	Port 3 General Purpose Input/Output data line 3.
P3.4	148	I/O	Port 3 General Purpose Input/Output data line 4.
P3.5	149	I/O	Port 3 General Purpose Input/Output data line 5.
P3.6	150	I/O	Port 3 General Purpose Input/Output data line 6.
		I	SYS *EXT_BOOT*. External boot mode selection. The state of this pin is sampled at reset.
P3.7	151	I/O	Port 3 General Purpose Input/Output data line 7.
		O	WIU. Stop mode request, active low ($\overline{\text{STOP\_REQ}}$).

Table 7. Port 3 Pins (Continued)

Symbol	Pin	I/O	Function
P3.8	152	I/O	Port 3 General Purpose Input/Output data line 8.
		I	External Interrupt 0 (*EINT0*).
P3.9	153	I/O	Port 3 General Purpose Input/Output data line 9.
		I	External Interrupt 1(*EINT1*).
		I	*BSPI1*. Slave Mode. Slave select line (*SS1*).
P3.10	154	I/O	Port 3 General Purpose Input/Output data line 10.
		I	External Interrupt 2 (*EINT2*).
		I	*BSPI2*. Slave Mode. Slave select line (*SS2*).
P3.11	155	I/O	Port 3 General Purpose Input/Output data line 11.
		I	External Interrupt 3 (*EINT3*).
		O	*RCCU*. PLL lock status (PLL_LOCK).
P3.12	156	I/O	Port 3 General Purpose Input/Output data line 12.
		I	External Interrupt 4 (*EINT4*).
		I	*EFT2* Timer 2 Input Capture line B *(ICAPB2*).

4.7 Port 4 pins

Port 4 consists of 5 bidirectional general purpose I/O pins. They are under ARM720 control through a configuration register, a data register and a direction register. These registers are read/write registers. Port lines can be individually configured as general purpose inputs, general purpose outputs or dedicated peripheral lines. The port can be read at any time: input lines return the pin level; output lines return the level of the output driver input. When written, the port stores the data in an internal register: it drives the pins only if they are configured as general purpose outputs. Port 4 outputs can be configured as push/pull or open drain drivers.

Table 8. Port 4 Pins

Symbol	Pin	I/O	Function
P4.0	16	I/O	Port 4 General Purpose Input/Output data line 0.
		I	*EMI* External Ready signal (*Eready*).
P4.1	15	I/O	Port 4 General Purpose Input/Output data line 1.
P4.2	14	I/O	Port 4 General Purpose Input/Output data line 2.
P4.3	13	I/O	Port 4 General Purpose Input/Output data line 3.
P4.4	12	I/O	Port 4 General Purpose Input/Output data line 4.
		O	*EMI* active low Chip Select - bank 1 (*Ecs1n*).

4.8 Main Clock input pins

Table 9. Main Clock Input Pins

Symbol	Pin)	I/O	Function
CLK	17	I-P	This pin is used as main system clock source.
	18	I-P	Reserved, must be tied to GND.
	20	I-P	Reserved, must be tied to GND.
	21	I-P	Reserved, must be tied to GND.
VREF	19	I	CLK input reference voltage

4.9 Port 6 pins

Port 6 is a configurable 8/16/32-bit interface port used by the Synchronous DRAM memory interface. DRAMC is under STR720 control through configuration registers. Port lines are permanently assigned to the relevant pins and they are listed in the following table.

Table 10. Port 6 Pins

Symbol	Pin	I/O	Function
P6.0	108	I/O	*SDRAM* Data line 0 (*Mid[0]*).
P6.1	107	I/O	*SDRAM* Data line 1 (*Mid[1]*).
P6.2	106	I/O	*SDRAM* Data line 2 (*Mid[2]*).
P6.3	105	I/O	*SDRAM* Data line 3 (*Mid[3]*).
P6.4	102	I/O	*SDRAM* Data line 4 (*Mid[4]*).
P6.5	101	I/O	*SDRAM* Data line 5 (*Mid[5]*).
P6.6	100	I/O	*SDRAM* Data line 6 (*Mid[6]*).
P6.7	99	I/O	*SDRAM* Data line 7 (*Mid[7]*).
P6.8	118	I/O	*SDRAM* Data line 8 (*Mid[8]*).
P6.9	117	I/O	*SDRAM* Data line 9 (*Mid[9]*).
P6.10	116	I/O	*SDRAM* Data line 10 (*Mid[10]*).
P6.11	115	I/O	*SDRAM* Data line 11 (*Mid[11]*).
P6.12	112	I/O	*SDRAM* Data line 12 (*Mid[12]*).
P6.13	111	I/O	*SDRAM* Data line 13 (*Mid[13]*).
P6.14	110	I/O	*SDRAM* Data line 14 (*Mid[14]*).
P6.15	109	I/O	*SDRAM* Data line 15 (*Mid[15]*).
P6.16	53	I/O	*SDRAM* Data line 16 (*Mid[16]*).
P6.17	52	I/O	*SDRAM* Data line 17 (*Mid[17]*).
P6.18	51	I/O	*SDRAM* Data line 18 (*Mid[18]*).
P6.19	50	I/O	*SDRAM* Data line 19 (*Mid[19]*).
P6.20	49	I/O	*SDRAM* Data line 20 (*Mid[20]*).

Table 10. Port 6 Pins (Continued)

Symbol	Pin	I/O	Function
P6.21	48	I/O	*SDRAM* Data line 21 (*Mid[21]*).
P6.22	45	I/O	*SDRAM* Data line 22 (*Mid[22]*).
P6.23	44	I/O	*SDRAM* Data line 23 (*Mid[23]*).
P6.24	65	I/O	*SDRAM* Data line 24 (*Mid[24]*).
P6.25	64	I/O	*SDRAM* Data line 25 (*Mid[25]*).
P6.26	63	I/O	*SDRAM* Data line 26 (*Mid[26]*).
P6.27	62	I/O	*SDRAM* Data line 27 (*Mid[27]*).
P6.28	61	I/O	*SDRAM* Data line 28 (*Mid[28]*).
P6.29	58	I/O	*SDRAM* Data line 29 (*Mid[29]*).
P6.30	57	I/O	*SDRAM* Data line 30 (*Mid[30]*).
P6.31	54	I/O	*SDRAM* Data line 31 (*Mid[31]*).
P6.32	83	O	*SDRAM* Address line 0 (*Mia[0]*).
P6.33	82	O	*SDRAM* Address line 1 (*Mia[1]*).
P6.34	81	O	*SDRAM* Address line 2 (*Mia[2]*).
P6.35	80	O	*SDRAM* Address line 3 (*Mia[3]*).
P6.36	79	O	*SDRAM* Address line 4 (*Mia[4]*).
P6.37	78	O	*SDRAM* Address line 5 (*Mia[5]*).
P6.38	77	O	*SDRAM* Address line 6 (*Mia[6]*).
P6.39	74	O	*SDRAM* Address line 7 (*Mia[7]*).
P6.40	73	O	*SDRAM* Address line 8 (*Mia[8]*).
P6.41	72	O	*SDRAM* Address line 9 (*Mia[9]*).
P6.42	86	O	*SDRAM* Address line 10 (*Mia[10]*).
P6.43	71	O	*SDRAM* Address line 11 (*Mia[11]*).
P6.44	87	O	*SDRAM* Address line 12 (*Mia[12]*).
P6.45	88	O	*SDRAM* Address line 13 (*Mia[13]*).
P6.46	N/A	N/A	*Not implemented pin.*
P6.47	69	O	*SDRAM* Memory clock signal (*MiClk*).
P6.48	70	O	*SDRAM* Memory clock enable signal (*MiClkEn*).
P6.49	89	O	*SDRAM* active low Chip Select - bank 0 (*MiCsN[0]*).
P6.50	95	O	*SDRAM* active low Chip Select - bank 1 (*MiCsN[1]*).
P6.51	97	O	*SDRAM* active low Chip Select - bank 2 (*MiCsN[2]*).
P6.52	98	O	*SDRAM* active low Chip Select - bank 3 (*MiCsN[3]*).
P6.53	90	O	*SDRAM* active low Setup Active signal (*RAS*).
P6.54	91	O	*SDRAM* active low Access Active signal (*CAS*).
P6.55	92	O	*SDRAM* active low Write Enable signal (*MiWeN*).

Table 10. Port 6 Pins (Continued)

Symbol	Pin	I/O	Function
P6.56	96	O	*SDRAM* active low Byte Lane 0 Strobe signal (*MiBls[0]*).
P6.57	119	O	*SDRAM* active low Byte Lane 1 Strobe signal (*MiBls[1]*).
P6.58	43	O	*SDRAM* active low Byte Lane 2 Strobe signal (*MiBls[2]*).
P6.59	68	O	*SDRAM* active low Byte Lane 3 Strobe signal (*MiBls[3]*).

4.10 Port 7 pins

Port 7 is a configurable 8/16-bit data port shared between the External Memory Interface for the connection of memory components - such as ROM, FLASH or SRAM devices - and the IDE interface. Both EMI and IDE are under ARM720 control through configuration registers. Shared port lines can be selected by software using the related GCR register bit.

Table 11. Port 7 Pins

Symbol	Pin	I/O	Function
P7.0	194	I/O	*EMI* Data line 0 (*Edata[0]*).
		I/O	*IDE* Data line 0 (*IDD[0]*).
P7.1	198	I/O	*EMI* Data line 1 (*Edata[1]*).
		I/O	*IDE* Data line 1 (*IDD[1]*).
P7.2	200	I/O	*EMI* Data line 2 (*Edata[2]*).
		I/O	*IDE* Data line 2 (*IDD[2]*).
P7.3	202	I/O	*EMI* Data line 3 (*Edata[3]*).
		I/O	*IDE* Data line 3 (*IDD[3]*).
P7.4	204	I/O	*EMI* Data line 4 (*Edata[4]*).
		I/O	*IDE* Data line 4 (*IDD[4]*).
P7.5	206	I/O	*EMI* Data line 5 (*Edata[5]*).
		I/O	*IDE* Data line 5 (*IDD[5]*).
P7.6	208	I/O	*EMI* Data line 6 (*Edata[6]*).
		I/O	*IDE* Data line 6 (*IDD[6]*).
P7.7	4	I/O	*EMI* Data line 7 (*Edata[7]*).
		I/O	*IDE* Data line 7 (*IDD[7]*).
P7.8	195	I/O	*EMI* Data line 8 (*Edata[8]*).
		I/O	*IDE* Data line 8 (*IDD[8]*).
P7.9	199	I/O	*EMI* Data line 9 (*Edata[9]*).
		I/O	*IDE* Data line 9 (*IDD[9]*).
P7.10	201	I/O	*EMI* Data line 10 (*Edata[10]*).
		I/O	*IDE* Data line 10 (*IDD[10]*).

Table 11. Port 7 Pins (Continued)

Symbol	Pin	I/O	Function
P7.11	203	I/O	*EMI* Data line 11 (*Edata[11]*).
		I/O	*IDE* Data line 11 (*IDD[*11*]*).
P7.12	205	I/O	*EMI* Data line 12 (*Edata[12]*).
		I/O	*IDE* Data line 12 (*IDD[*12*]*).
P7.13	207	I/O	*EMI* Data line 13 (*Edata[13]*).
		I/O	*IDE* Data line 13 (*IDD[*13*]*).
P7.14	3	I/O	*EMI* Data line 14 (*Edata[14]*).
		I/O	*IDE* Data line 14 (*IDD[*14*]*).
P7.15	5	I/O	*EMI* Data line 15 (*Edata[15]*).
		I/O	*IDE* Data line 15 (*IDD[*15*]*).
P7.16	168	O	*EMI* Address line 0 (*Eadd[0]*).
		O	*IDE* Address line 0 (*IDA[*0*]*).
P7.17	169	O	*EMI* Address line 1 (*Eadd[1]*).
		O	*IDE* Address line 1 (*IDA[*1*]*).
P7.18	170	O	*EMI* Address line 2 (*Eadd[2]*).
		O	*IDE* Address line 2 (*IDA[2]*).
P7.19	191	O	*EMI* Address line 3 (*Eadd[3]*).
P7.20	190	O	*EMI* Address line 4 (*Eadd[4]*).
P7.21	189	O	*EMI* Address line 5 (*Eadd[5]*).
P7.22	188	O	*EMI* Address line 6 (*Eadd[6]*).
P7.23	187	O	*EMI* Address line 7 (*Eadd[7]*).
P7.24	178	O	*EMI* Address line 8 (*Eadd[8]*).
P7.25	177	O	*EMI* Address line 9 (*Eadd[9]*).
P7.26	176	O	*EMI* Address line 10 (*Eadd[10]*).
P7.27	175	O	*EMI* Address line 11 (*Eadd[11]*).
P7.28	174	O	*EMI* Address line 12 (*Eadd[12]*).
P7.29	173	O	*EMI* Address line 13 (*Eadd[13]*).
P7.30	172	O	*EMI* Address line 14 (*Eadd[14*).
P7.31	171	O	*EMI* Address line 15 (*Eadd[15]*).
P7.32	7	O	*EMI* Address line 16 (*Eadd[16]*).
P7.33	186	O	*EMI* Address line 17 (*Eadd[17]*).
P7.34	185	O	*EMI* Address line 18 (*Eadd[18]*).
P7.35	179	O	*EMI* Address line 19 (*Eadd[19]*).
P7.36	182	O	*EMI* Address line 20 (*Eadd[20]*).
P7.37	184	O	*EMI* Address line 21 (*Eadd[21]*).

Table 11. Port 7 Pins (Continued)

Symbol	Pin	I/O	Function
P7.38	193	O	*EMI* active low External Read Strobe. (*Erdn*).
		O	*IDE* active low Read strobe (*IDiorn*).
P7.39	183	O	*EMI* active low External Write Strobe, bits 7:0 of external memory (*Ewr0n*).
		O	*IDE* active low Write strobe (I*Diown*).
P7.40	6	O	*EMI* active low External Write Strobe, bits 15:8 of external memory (*Ewr1n).*
P7.41	192	O	*EMI* active low Chip Select - bank 0 (*Ecs0n*).
P7.42	166	O	*IDE* active low Chip select 0 (I*Cs0n).*
P7.43	167	O	*IDE* active low Chip select 1 (I*Cs1n).*
P7.44	164	I	*IDE* Ready (*Iordy).*
P7.45	165	I	*IDE* Interrupt (*Irq).*

4.11 PQFP208 Package Pin Configuration

The package used for STR720 device is a Plastic Quad-Flat Package supporting 208 pin. For a complete description of this package mechanical characteristics See “PACKAGE

MECHANICAL DATA" on page 373. In the following drawing and table the device pin mapping is reported

Figure 3. STR720 PQFP208 pin configuration (top view)

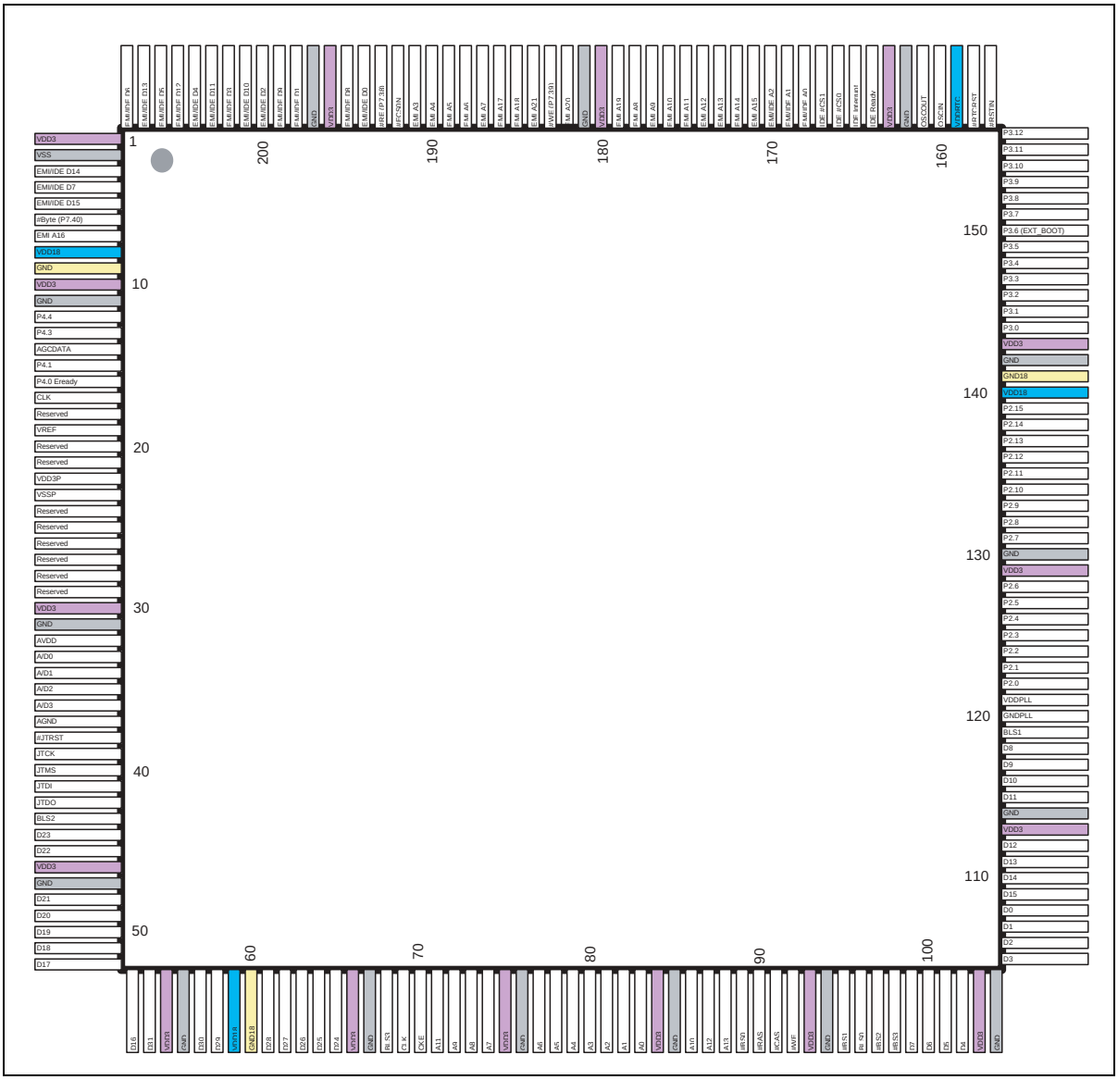


Table 12. STR720 PQFP208 pin map

Pin	Symbol	Pin	Symbol	Pin	Symbol	Pin	Symbol
1	VDD3	53	MID[16]	105	MID[3]	157	RSTIN_N
2	VSS	54	MID[31]	106	MID[2]	158	RTCRST_N
3	EDATA[14] IDD[14]	55	VDD3	107	MID[1]	159	VDDRTC
4	EDATA[7] IDD[7]	56	VSS	108	MID[0]	160	OSCIN
5	EDATA[15] IDD[15]	57	MID[30]	109	MID[15]	161	OSCOUT
6	EWR1_N	58	MID[29]	110	MID[14]	162	VSS
7	EADD[16]	59	VDD	111	MID[13]	163	VDD3
8	VDD	60	VSS	112	MID[12]	164	IORDY
9	VSS	61	MID[28]	113	VDD3	165	IRQ
10	VDD3	62	MID[27]	114	VSS	166	ICS0_N
11	VSS	63	MID[26]	115	MID[11]	167	ICS1_N
12	P4[4]	64	MID[25]	116	MID[10]	168	EADD[0] IDA[0]
13	P4[3]	65	MID[24]	117	MID[9]	169	EADD[1] IDA[1]
14	P4[2]	66	VDD3	118	MID[8]	170	EADD[2] IDA[2]
15	P4[1]	67	VSS	119	MIBLS_N[1]	171	EADD[15]
16	P4[0]	68	MIBLS_N[3]	120	VSSPLL	172	EADD[14]
17	CLK	69	MICLK	121	VDDPLL	173	EADD[13]
18	Reserved (GND)	70	MICLKEN	122	P2[0]	174	EADD[12]
19	VREF	71	MIA[11]	123	P2[1]	175	EADD[11]
20	Reserved (GND)	72	MIA[9]	124	P2[2]	176	EADD[10]
21	Reserved (GND)	73	MIA[8]	125	P2[3]	177	EADD[9]
22	VDD3P	74	MIA[7]	126	P2[4]	178	EADD[8]
23	VSSP	75	VDD3	127	P2[5]	179	EADD[19]
24	Reserved (GND)	76	VSS	128	P2[6]	180	VDD3
25	Reserved	77	MIA[6]	129	VDD3	181	VSS
26	Reserved (float)	78	MIA[5]	130	VSS	182	EADD[20]
27	Reserved	79	MIA[4]	131	P2[7]	183	EWR0_N
28	Reserved (GND)	80	MIA[3]	132	P2[8]	184	EADD[21]
29	Reserved (GND)	81	MIA[2]	133	P2[9]	185	EADD[18]
30	Reserved (GND)	82	MIA[1]	134	P2[10]	186	EADD[17]
31	Reserved (GND)	83	MIA[0]	135	P2[11]	187	EADD[7]

Table 12. STR720 PQFP208 pin map

Pin	Symbol	Pin	Symbol	Pin	Symbol	Pin	Symbol
32	AVDD	84	VDD3	136	P2[12]	188	EADD[6]
33	AN0	85	VSS	137	P2[13]	189	EADD[5]
34	AN1	86	MIA[10]	138	P2[14]	190	EADD[4]
35	AN2	87	MIA[12]	139	P2[15]	191	EADD[3]
36	AN3	88	MIA[13]	140	VDD	192	ECS0_N
37	AGND	89	MCS_N[0]	141	VSS	193	ERD_N
38	JTRST_N	90	MISA_N	142	VSS	194	EDATA[0] IDD[0]
39	JTCK	91	MIAA_N	143	VDD3	195	EDATA[8] IDD[8]
40	JTMS	92	MIWE_N	144	P3[0]	196	VDD3
41	JTDI	93	VDD3	145	P3[1]	197	VSS
42	JTDO	94	VSS	146	P3[2]	198	EDATA[1] IDD[1]
43	MIBLS_N[2]	95	MCS_N[1]	147	P3[3]	199	EDATA[9] IDD[9]
44	MID[23]	96	MIBLS_N[0]	148	P3[4]	200	EDATA[2] IDD[2]
45	MID[22]	97	MCS_N[2]	149	P3[5]	201	EDATA[10] IDD[10]
46	VDD3	98	MCS_N[3]	150	P3[6]	202	EDATA[3] IDD[3]
47	VSS	99	MID[7]	151	P3[7]	203	EDATA[11] IDD[11]
48	MID[21]	100	MID[6]	152	P3[8]	204	EDATA[4] IDD[4]
49	MID[20]	101	MID[5]	153	P3[9]	205	EDATA[12] IDD[12]
50	MID[19]	102	MID[4]	154	P3[10]	206	EDATA[5] IDD[5]
51	MID[18]	103	VDD3	155	P3[11]	207	EDATA[13] IDD[13]
52	MID[17]	104	VSS	156	P3[12]	208	EDATA[6] IDD[6]

5 ARCHITECTURE OVERVIEW

This chapter is meant to give an overview of the peripherals available inside STR720, with the dedicated STR720 customizations. STR720 peripherals can be divided in the following groups:

- **AHB peripheral set**. This set is composed of all peripherals requiring fast access and high transfer throughput and it contains program RAM memory, boot-ROM memory, DMA processor capable of transferring data between peripherals and memory, ATAPI interface for IDE device connection, DRAMC controller allowing the usage of commercial SDRAM banks as system memory, and EMI which enables a direct connection of external FLASH memories for system boot or other external memory mapped devices. Two different AHB-APB bridges enable access to the rest of STR720 peripheral set.
- **S-APB peripheral set**. This set contains those system peripherals which require to run synchronously with ARM720T core, due to performance requirements or ease of integration. It is composed of an Enhanced Interrupt Controller (EIC) and a specific Wake-Up Interrupt Unit (WIU) to extend ARM720T interrupt capabilities, a Reset-Clock Control Unit (RCCU) to select between different system clock source options and implement all the power saving modes, a Real Time Clock to keep timing during powerdown mode, a Real-Time clock generator, and configurable I/O ports giving access to a large number of configurable pins.
- **A-APB peripheral set**. This set contains most of system peripherals and it is designed to run at a lower frequency independent from the ARM720T one, thus reducing the overall power consumption. It is composed of a set of serial channels to implement different kind of user interfaces (BSPI and UART), protocol specific serial interfaces as CAN and USB, a 4-channel A/D converter suitable for control-voltage monitoring, 2 independent Enhanced Function Timers (EFT) which can be used as system scheduler, Watch DoG for system reliability (WDG), and configurable I/O port.

In the following sections a brief overview of the STR720 system is given. For a more detailed description of each peripheral, please refer to the related chapters.

5.1 Enhanced Interrupt Controller (EIC)

The ARM720T CPU provides two levels of interrupt:

- FIQ (Fast Interrupt Request) for fast, low latency interrupt handling.
- IRQ (Interrupt Request) for more general interrupts.

The Enhanced Interrupt Controller (EIC) implements the handling of multiple interrupt channels, interrupt priority and automatic vectorization. It provides:

- 32 maskable interrupt channels, mapped on ARM720T interrupt request pin IRQ
- 32 interrupt vectors
- 3 maskable interrupt channels, mapped on ARM720T fast interrupt request pin FIQ.

- 16 programmable priority levels for each interrupt channel mapped on IRQ
- hardware support for interrupt nesting (up to 16 interrupt requests can be nested), with internal HW nesting stack
- at register offset 0x18h, the start address of the ISR of the highest priority interrupt or directly the jump instruction to it (defined by the application).

The EIC performs the following operations without software intervention:

- reject/accept an interrupt request according to the related channel mask bit,
- compare all IRQ requests with the current priority level interrupt. The IRQ is asserted if the priority of the current interrupt request is higher than the stored current priority,
- load the address vector of the highest priority IRQ to the Interrupt Vector Register (offset 0x18h)
- save the previous interrupt priority in the HW priority stack whenever a new IRQ is accepted
- update the Current Interrupt Priority Register with the new priority whenever a new interrupt is accepted

5.1.1 IRQ Interrupt Vector Table

Up to 32 interrupt channels are mapped on low priority ARM720T interrupt request pin (IRQ). Hereafter the mapping of the different interrupt sources on the vector tables is described. The highest-priority interrupt request will be issued on ARM720T IRQ pin. In case of multiple interrupt requests mapped on the same interrupt vector, a polling on interrupt flags contained inside peripherals must be performed by the application code in order to establish the exact source of interrupt (see Interrupt Flags column in Table 13 on page 32).

There are 16 priority levels for IRQ mapped interrupt. The internal daisy-chain will define the priority relationship for different interrupt vectors having the same priority level: the higher is the interrupt vector number the higher is the priority in the daisy chain.

Interrupt Vector 0

- External interrupt requests.
  Four different interrupt requests are OR-ed together to generate a single interrupt request:
  - External interrupt 0 (EINT0)
  - External interrupt 1 (EINT1)
  - External interrupt 2 (EINT2)
  - External interrupt 3 (EINT3)

Interrupt Vector 1

- External interrupt request 4 (EINT4).

Interrupt Vector 2

- Wake-up Controller interrupt request
  From 0 to 11 (the number of wake-up input alternate function pins) different interrupt requests are OR-ed together to generate a single interrupt request. Refer to Wake-up Controller specifications for further details on interrupt request generation.

Interrupt Vector 3

- Extended Function Timer 1 interrupt request.
  Five different interrupt requests are OR-ed together to generate a single interrupt request:
  - Input Capture A
  - Output Compare A
  - Timer Overflow
  - Input Capture B
  - Output Compare B

Interrupt Vector 4

- Extended Function Timer 2 interrupt request.
  Three different interrupt requests are OR-ed together to generate a single interrupt request:
  - Input Capture A
  - Timer Overflow
  - Input Capture B

Interrupt Vector 5

- UART1 interrupt request
  Nine different interrupt requests are OR-ed together to generate a single interrupt request:
  - Receive buffer full
  - Transmit buffer empty
  - Transmit buffer half empty
  - Parity error
  - Frame error
  - Overrun error
  - Timeout not empty
  - Timeout idle
  - Receive buffer half full

Interrupt Vector 6

- UART2 interrupt request (see Interrupt Vector 5).

Interrupt Vector 7

- DMA controller interrupt request.

  Four different interrupt requests are OR-ed together to generate a single interrupt request:

  - Data stream 2 and 3 completed transfer interrupts
  - Data stream 2 to 3 transfer error interrupts

Interrupt Vector 8

- Reserved.

Interrupt Vector 9

- Reserved.

Interrupt Vector 10

- USB interface high priority events interrupt request.

  All isochronous and double-buffer endpoint correct transfer interrupt requests are OR-ed together to generate a single interrupt request. The number of isochronous/ double-buffered endpoints is programmable by software, so the actual number of events raising this interrupt line can vary from 0 to 7.

Interrupt Vector 11

- USB interface low priority and generic events interrupt request.

  All endpoint correct transfer interrupt requests, except isochronous and double-buffered (see Interrupt Vector 10) are OR-ed together with the generic USB events to generate a single interrupt request.

  The USB events not related to endpoint activity are:

  - Data overrun/underrun
  - USB Error
  - USB Wake-up
  - USB Suspend
  - USB Reset
  - Start of frame
  - Missing start of frame

  The number of isochronous/double-buffered endpoints is programmable by software, so the actual number of events raising this interrupt line can vary from 7 to 15.

Interrupt Vector 12

- CAN Interface interrupt request.

  Thirty-two different interrupt requests are OR-ed together to generate a single interrupt request:

  - Status interrupt
  - Message object 1 to 32 reception/transmission interrupts

Interrupt Vector 13

- Buffered SPI 1 interrupt request.

  Five different interrupt requests are OR-ed together to generate a single interrupt request:

  - Receive interrupt
  - Receive overflow
  - Transmit interrupt
  - Transmit underflow
  - Bus error

Interrupt Vector 14

- Buffered SPI 2 interrupt request (see Interrupt Vector 13).

Interrupt Vectors 15

- Primary IDE interface interrupt request.

Interrupt Vectors 16

- Real-Time Clock periodic interrupt request.

Interrupt Vector 23

- Analog to Digital Converter sample ready interrupt request.

Interrupt Vector 29

- Extended Function Timer 2 output compare A interrupt request.

Interrupt Vector 30

- Extended Function Timer 2 output compare B interrupt request.

Interrupt Vector 31

- Watchdog Timer interrupt request.

Table 13. IRQ Interrupt Vector Summary

Vector	Peripheral	Peripheral Interrupt Flags
IRQ0	**IRQ03IT**: External Interrupts EINT0-EINT3	4
IRQ1	**IRQ4IT**: External Interrupt EINT4	1
IRQ2	**WIUIT**: WIU wake-up event interrupt	11
IRQ3	**EFT1IT**: EFT1 global interrupt	5
IRQ4	**EFT2IT**: EFT2 global interrupt	3
IRQ5	**UART1IT**: UART 1 global interrupt	9
IRQ6	**UART2IT**: UART 2 global interrupt	9
IRQ7	**DMAIT**: DMA event global interrupt	4
IRQ8	**Reserved.**	
IRQ9	**Reserved.**	
IRQ10	**USBHPIT**: USB high priority event interrupt	0 - 7
IRQ11	**USBLPIT**: USB low priority event interrupt	7 - 15
IRQ12	**CANIT**: CAN module general interrupt	32
IRQ13	**BSPI1IT**: BSPI 1 global interrupt	5
IRQ14	**BSPI2IT**: BSPI 2 global interrupt	5
IRQ15	**IDEPIRQ:** IDE Primary channel interrupt	1
IRQ16	**RTCPERIT**: RTC Periodic interrupt	1
IRQ17	**Reserved.**	
IRQ18	**Reserved.**	
IRQ19	**Reserved.**	
IRQ20	**Reserved.**	
IRQ21	**Reserved.**	
IRQ22	**Reserved.**	
IRQ23	**ADCIT:** ADC sample ready interrupt	1
IRQ24	**Reserved.**	
IRQ25	**Reserved.**	
IRQ26	**Reserved.**	
IRQ27	**Reserved.**	
IRQ28	**Reserved.**	
IRQ29	**EFT2OCA:** EFT2 Output Compare A Interrupt	1
IRQ30	**EFT2OCB:** EFT2 Output Compare B Interrupt	1
IRQ31	**WDGIT**: WDG timer interrupt	1

5.1.2 FIQ Interrupt Vector Table

Three maskable interrupt sources are mapped on FIQ vectors:

- FIQ0: External interrupt channel 0
- FIQ1: Extended Function Timer 2 output compare B interrupt request
- FIQ2: AHB Error detection interrupt request

Having only one FIQ vector shared by the three interrupt sources, a polling on FIQ interrupt pending bits contained inside EIC must be performed by the application code in order to establish the exact source of interrupt, in case more than one FIQ source is enabled. Besides, having no priority mechanism, in case of contemporaneous FIQ events, the software shall manage the priority simply by polling the pending bits and managing the concurrence.

Table 14. FIQ Interrupt Vector Table

Vector	Interrupt Source
FIQ0	External Interrupt IRQ0
FIQ1	EFT2 Output Compare B Interrupt
FIQ2	AHB Error detection Interrupt

5.1.3 IRQ Interrupt Vectoring

The EIC (Enhanced Interrupt Controller) implements an interrupt structure pointing the processor at the first instruction location of the channel-specific Interrupt (IRQ) Service Routine.

IVR (Interrupt Vector Register) is the EIC's 32-bit register at address 0xFFFF_FC18 acting as pointer. It is composed by two main fields: the upper half word (16 bit) is directly programmable, while the lower half word is the mirrored entry of a register table (named SIR) indexed by the interrupt channel (see Chapter 7: ENHANCED INTERRUPT CONTROLLER (EIC) on page 51).

The absolute address 0x0000_0018 is where the ARM720T CPU jumps as consequence of an interrupt request on its IRQ pin.

The STR720 implementation is such that the ARM's 0x0000_0018 location and the EIC's IVR location are considered as distinct addresses. The EIC IVR register can contain, for example, the absolute address of the interrupt service routine or an index to a jump table; at location 0x0000_0018 the interrupt handling routine starts, using IVR as index or pointer to the actual response routine. To implement this behaviour the absolute 0x0000_0018 location must contain an instruction which allows to perform a jump to the location pointed to by the IVR register.

For instance, the absolute location 0x0000_0018 should contain the following instruction:

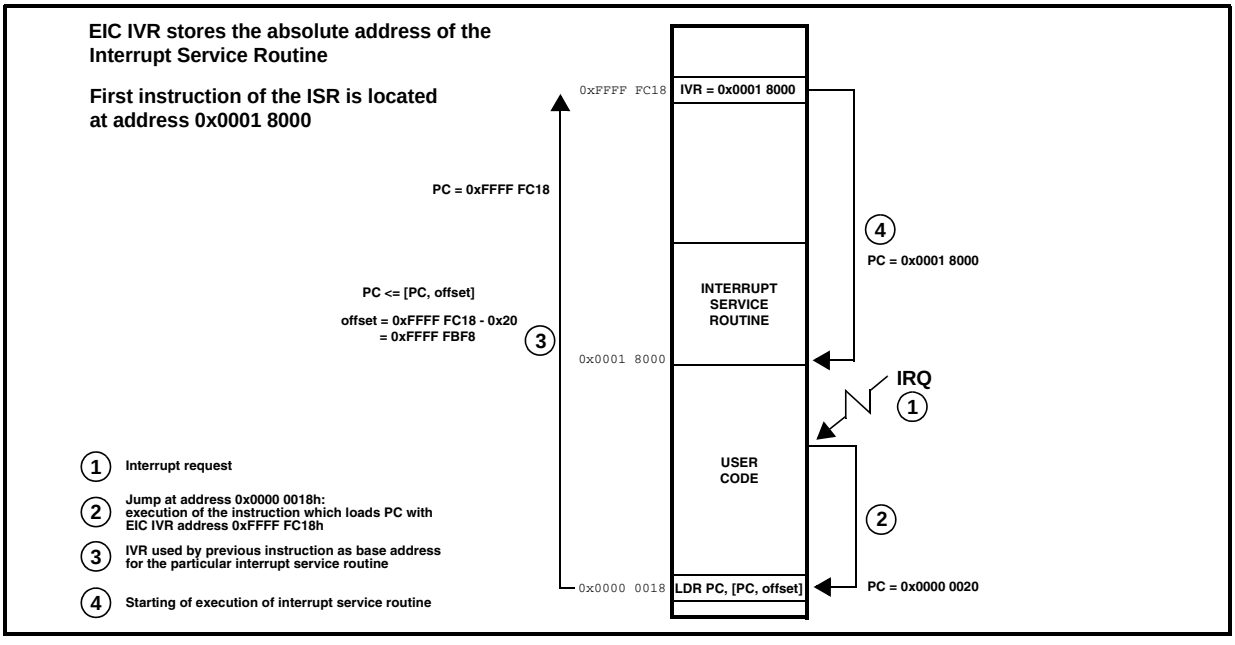
LDR PC, [PC, offset]

where "offset" is what to add to the PC to obtain the EIC IVR address. The instruction above allows the CPU to load the Program Counter register with the address kept in EIC IVR and so to jump to the location pointed to by the IVR register.

The user has to consider that when the absolute address 0x0000_0018 is being fetched, the PC is equal to 0x0000_0020 (18h + 8h). So, the offset is equal to:

offset = IVR address – 0x0000_0020 = 0xFFFF_FC18 – 0x0000_0020 = 0xFFFF_FBF8

Figure 4. IRQ Interrupt Vectorization



This mechanism allows a jump to virtually any location of the 4GB memory space. However, since the channel dependent portion of IVR are the lower 16 bits, once the base address (in the upper part) has been fixed the interrupt handler routines can only be within a 64Kbyte range from that base address.

5.2 Wake-up/Interrupt management Unit (WIU)

The Wake-up/Interrupt Management Unit can be seen as an extension of the number of external interrupt lines but it can also manage up to 16 wake-up lines capable of exiting STOP mode:

- 4 external wake-up lines all connected to the IRQ0 channel of the interrupt controller. These wake-up lines have dedicated input ports.
- 1 external wake-up line directly mapped to a dedicated channel of the interrupt controller (IRQ1). This wake-up line has a dedicated input port.
- 10 internal wake-up lines connected to the IRQ2 channel of the interrupt controller module. The ‘internal” lines are mapped on some significant STR720 signals.

The external wake-up pins (available as alternate function of general purpose I/O lines) can be programmed as plain external interrupt lines or as wake-up lines to exit STOP mode. When the external lines are defined as wake-up pins a programmable signal edge will asynchronously trigger the wake-up event, changing the status of the alternate output function STOP_REQ, associated to P3.7 and allowing the external power control to restore system main clock source. In this way STOP mode can be exited without resetting the whole system, provided that STOP_REQ is enabled.

In STR720 implementation, the 10 internal wake-up lines are associated with the input of serial communication modules and also with other relevant internal events, as defined in Table 15. In this way, if properly programmed, the system can be woken up by the detection of activity on any serial bus or when the external CLK restarts. For example, wake-up line 15 can be used to restore STR720 system when only the Real Time Clock is left running.

Table 15. Wake-up line sources

Wake-up line #	Wake-up line source
6	RCCU: Change of PLL Lock condition (**see note below**).
7	CLK clock line
8	Port2.0: UART1 Receive Data Input (RxD1)
9	Port2.2: UART2 Receive Data Input (RxD2)
10	Port2.6: BSPI1. Slave Mode. Input serial clock (SCK1).
11	Port2.9: BSPI2. Slave Mode. Input serial clock (SCK2).
12	Port2.11: CAN module receive pin (CANRX).
13	Reserved.
14	USB wake-up event: generated while exiting from suspend mode.
15	RTC related event: Alarm Interrupt.

Note Wake-up line 6 is associated to any change of PLL lock condition detected by RCCU unit but, unlike all others, this wake-up line cannot be used to restore system from STOP mode. It can be used as an interrupt extension line only.

Note Wake-up line 7, associated to CLK clock, will be always set its corresponding pending bit as long as the external clock is present.

5.3 DMA Controller (DMAC)

The DMA controller provides access to 2 data streams inside STR720 system. Since peripherals are memory mapped, data transfers from/to peripherals are managed like memory/memory data transfers.

Each data stream can be associated to one particular peripheral, which triggers a DMA request starting the data transfer between the corresponding buffers defined in the stream descriptor. Data stream 3 of DMA controller can alternatively be used as a memory/memory data transfer triggered by a software DMA request, independently from any peripheral activity. In case data stream 3 is also associated to a peripheral request, the two modes can be selected by software.

The DMA controller supports circular buffer management avoiding the generation of interrupts when the controller reach the end of the source buffer. When a stream is configured in circular buffer mode and the end of buffer is reached, DMA controller reloads the start address and continues the transfer until software notifies the end of operations.

The priority between different DMA triggering sources is defined by hardware, source 0 being the highest priority request and source 3 being the lowest one. The mapping of DMA requests

implemented in STR720 system is detailed in Table 16. Where multiple sources are available

Table 16. DMA request mapping

Stream	DMA triggering source
0	Reserved.
1	Reserved.
2	BSPI1 Receive data request
3	Memory-to-memory / BSPI1 Transmit data request / ADC

for a stream triggering event, the selection is performed by specific configuration bits located in SGCR3 register. Note that ATAPI can be programmed so to transfer its data to/from memory using the memory-to-memory transfer mode, so there is no need to generate a specific request to the DMA controller.

The DMA controller treats bytes in memory as being in Little Endian format: the lowest numbered byte in a word is considered the word least significant byte and the highest numbered byte the most significant.

5.4 DRAM controller (DRAMC)

The STR720 device provides an SDRAM which supports four external banks containing SDRAM memories, all banks being of the same DRAM type. Each bank can be enabled/ disabled through configuration registers and its size can vary between 64 KBytes and 32 MBytes. This allows the controller to address from 64 KBytes (1 bank of 64 KBytes) up to 128 MBytes (four banks of 32 MBytes). Due to pad number limitation, the maximum addressable range of 128 MBytes (32 MBytes each bank) can be reached only when 32 bit wide memories are used and column address is at least 9 bit long.

Accesses to the SDRAM memory are done using an AHB interface, while a dedicated APB interface enables the CPU to configure various parameters of SDRAMC accesses to the external memory as word and column size, data latency, setup time, idle time, bank enabling, refresh period.

When ARM720 cache is enabled, all accesses to SDRAM memory are performed as bursts of 4 transfers on AHB bus and in order to achieve the highest performance level, SDRAM interface must be aware of this peculiar access mode. Unfortunately a limitation in ARM720 core and SDRAM interface requires this specific configuration to be enforced explicitly by software. This configuration selection is performed by using CACHE_CONFIG bit of SGCR1 register, which can configure STR720 to work in “burst-access” mode and it should be used whenever cache is enabled, so to have optimal performance from the system. See Section 10.5: *Programming considerations on page 114* about SDRAM configuration, and Section 24.3: *S-GCR Block description on page 352* for further details and limitations about the usage of SDRAM “burst-access” mode.

5.5 External Memory Interface

The External Memory Interface (EMI) controls the data flow from AHB bus to an external memory components - both ROM, SRAM and PCMCIA Interface and it can access up to 8 Mbytes of memory space, possibly partitioned in 4 different memory banks.

The AHB interface manages 32 bit data that can be then packed/unpacked before being send through a 16 bit data bus. The EMI will automatically unpack the data and reformat it to the right part of the bus for write access. For read access, it will buffer the access to build the complete 32-bit data to be send to the AHB.

The length of time required to properly transfer data to external memory components is controllable by programming of internal registers dedicated to each external memory space. In case the response time of the external device is not constant, a specific alternate function is provided on P4.0 used as an external ready signal, whose transition flags the end of current access. This external ready functionality can be enabled by programming the internal EMI registers which also contain flag bits to inform the controller if a particular memory space is accessible or not. These internal registers are accessed via an APB interface to the EMI block.

The External Memory Interface shares a 46-bit port (port 7) with the IDE interface pins so that the two interfaces cannot be used at the same time. The address bus starts from bit 0 (P7.16) and ends on bit 21 (P7.37) always representing the 16-bit word location accessed by the EMI block. The selection between EMI and IDE pins can be done by setting the corresponding bit in SGCR1 register. Beside this limitation, also the number of accessible memory banks is limited to 2: bank 0 will be always accessible while bank 1 can be used only through a specific I/O programming since its chip select line is shared with another GPIO pad (P4.4 = CS1). Memory banks CS2 and CS3 are never accessible.

5.6 ATAPI IDE interface

ATA 4 Industry Standard EIDE controller. The controller allows interfacing to devices such as Hard disk drives or CD ROM to the STR720 system being capable of transferring data either under CPU control or using the DMA controller “memory-to-memory” transfer mode, thus being able to spare CPU time even if PIO modes are configured. The IDE controller supports the following features:

- Primary-only channel, supporting up to two IDE devices.
- Support for CD-ROM and tape peripherals.
- Independently programmable timing for each device.
- Programmable posted writes and read-prefetch.
- Software selectable endianness for data read from CDROM / tape peripheral.
- Support for I/O Channel Ready.
- Support for PIO modes 0, 2, 3 and 4.

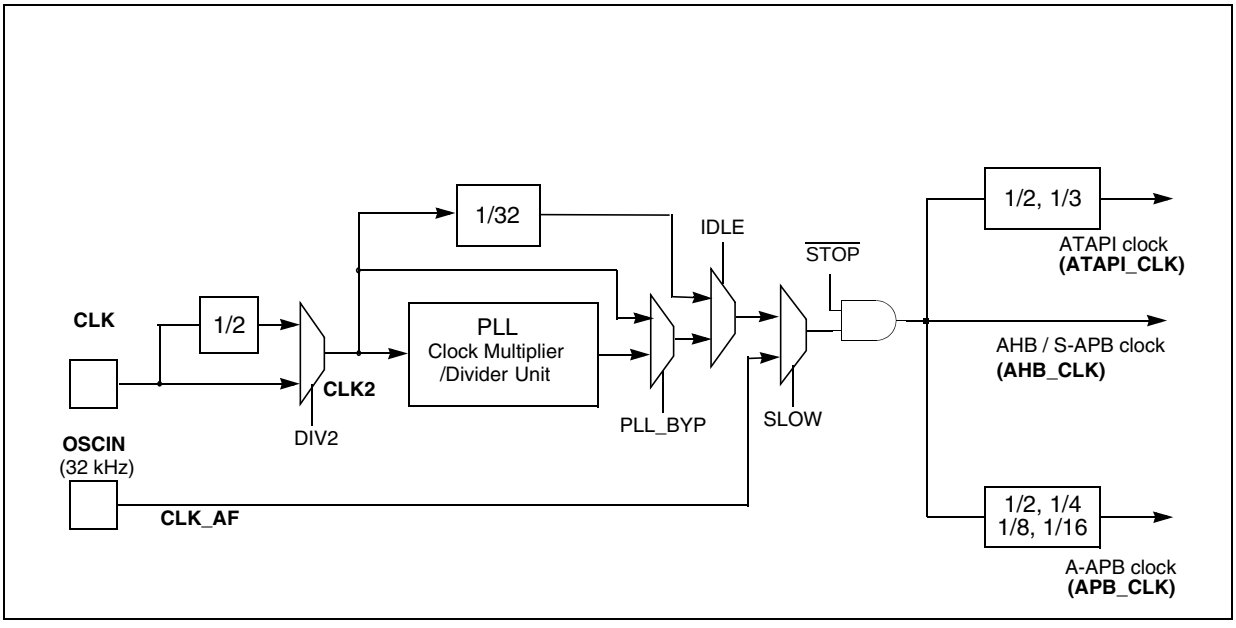
5.7 Reset and Clock Control Unit (RCCU)

Reset and Clock Control Unit (RCCU) is responsible of the control and distribution of the reset and the clocks signals of STR720.

5.7.1 Clock management

All possible operating modes (including low power ones) are managed by this peripheral by selecting the system clock source and prescaling it so to adapt its frequency to actual system requirements. A conceptual block diagram of RCCU clock management structure is reported in Figure 5 on page 38

Figure 5. RCCU Simplified Block Diagram



As the diagram reports, different clock frequencies can be used for the main system composed by the AHB and the S-APB subsystems. As an example, when CLK clock is driven by a 16 MHz signal, the following values are possible, for main system clock:

- CLK: 16 MHz
- CLK divided by 2
- CLK divided by 32 or by 64
- CLK multiplied by PLL (see RUN mode section)
- CLK_AF source: 32 kHz oscillator output.

A programmable prescaler derives A-APB subsystem clock from main clock signal, while independent and dedicated prescalers are implemented in order to generate the ATAPI clock. All serial interfaces (CAN, BSPI, UART) have configurable internal prescalers in order to generate the correct baud rates. Each peripheral inside STR720 system can be

independently stopped, freezing its clock input signal, by proper configuration registers located in the A-APB bridge, in GCR registers and in a specific Clock Gating Control block.

If application software requires it, RCCU block can issue an interrupt, through the WIU block, upon any change of PLL lock condition, so to take proper action when system clock frequency is different from what is set in RCCU configuration.

For a detailed description of STR720 operating modes, see Chapter 25: POWER REDUCTION MODES on page 364.

Note The AHB and S-APB blocks are driven by the same clock signal (AHB_CLK) .

5.7.2 RESET management

The Reset Manager resets the MCU when one of the following events occurs:

– A Hardware reset, initiated by a low level on the Reset pin;
– A Software reset, forced setting a control bit inside the RCCU
– A Watchdog end of count condition (when enabled)

The event causing the last Reset is flagged in the CLKFLAG register: the corresponding bit is set. A hardware initiated reset will leave all these bits reset.

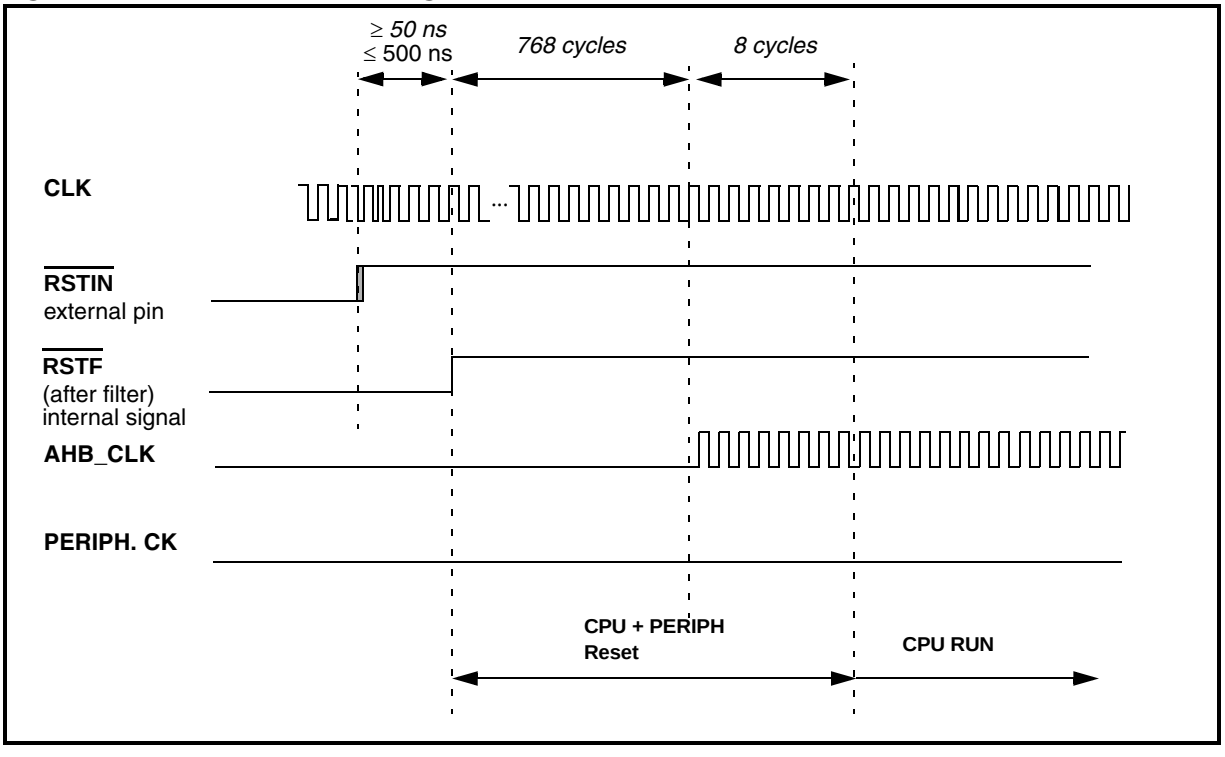
The hardware reset overrides all other conditions and forces the system to the reset state. During the Reset phase, the internal registers are set to their reset values, where these are defined, and the I/O pins configuration is restored to the condition reported in Section 5.18: *General Purpose I/O Ports on page 44* and Section 5.19: *Dedicated Pins on page 45*.

Reset from pad is asynchronous: when it is driven low for a duration longer than the on-chip RC filter width (between 50 ns and 500 ns), a Reset cycle is initiated (see Chapter 26: SYSTEM RESET on page 370). The on-chip Watchdog Timer generates a reset condition if the Watchdog mode is enabled and if the programmed period elapses without the specific code written to the appropriate register (see Chapter 20: WATCHDOG TIMER (WDG) on page 303).

When the Reset pin goes high again, the on-chip RC filter width plus 768 CLK plus 8 AHB_CLK cycles are counted before exiting the Reset state (plus possibly up to four additional CLK period, depending on the delay between the rising edge of the Reset pin and the first rising edge of CLK). As an example it corresponds to an interval between 48.25 µs and 48.5 µs with a 16 MHz input clock.

At the end of the Reset phase, the Program Counter will be set to the location specified in the Reset Vector located in the 0000 0000h location of memory.

Figure 6. Reset General Timing



When a reset is asserted, RCCU registers will be preset in the following state:

- Reference Clock (CLK clock), with no sub-division, will be used as System Clock and fed in input to the PLL;
- PLL will be switched off and bypassed with its multiplication factor preset to 1;
- Most of peripheral clocks will be stopped (see Table 83: *Peripheral clock and reset gating on page 366*)

It is up to the STR720 boot code to properly program the RCCU registers so as to use the PLL, to set different division factors for APB_CLK, to enable or disable the peripheral clocks according to application requirements.

5.8 Real Time Clock (RTC)

The real time clock is a 48 bits counter with a resolution of one period of the 32768 Hz input clock. The counter is fed by the output clock of a 32768 Hz crystal oscillator. The RTC and the oscillator are supplied by a dedicated 1.8 V stand-by power supply allowing the possibility to switch off all the device keeping the counter running.

The counter is reset by a dedicated in pin which is supplied by 1.8 V too. The system reset does not have any effect on the RTC.

The RTC generates a configurable periodic tick (see Table 13, “IRQ Interrupt Vector Summary,” on page 32 and Table 15, “Wake-up line sources,” on page 35) and a programmable alarm.

Readable/Writable registers contain the time elapsed since clock start, measured as 30.52 µs ticks on a 48 bit counter.

The counter overflow is not handled being higher than 200 years.

5.9 AHB-APB bridges

In order to reduce the power consumption of the overall core, AHB and A-APB are clocked with different frequencies: AHB with an high frequency to have high transfer performance, A-APB with a scaled down frequency in order to reduce power consumption. The AHB-APB asynchronous bridge provides a resynchronization mechanism in order to enable master of the AHB bus to access the APB peripherals. For information on the APB bridges mapping, please refer to Section 6.3: *APB Bridges Mapping on page 48*.

Since the ratio between AHB and A-APB could be also 1/16, more than 32 AHB-cycles could be necessary to access a peripheral.

Moreover, the AHB-APB asynchronous bridge also controls the clock and reset signals of the peripherals connected to A-APB subsystem. The Peripheral Clock Gating (PCG) registers allow to clock off independently any of the peripherals connected to the bridge.

For peripherals that do have both an AHB and APB interface but that do not support two independent clock domains, as External Memory Interface (EMI) and DMA Controller (DMAC), a synchronous APB bridge is also implemented (S-APB). This bridge is also used to connect those APB peripherals having specific bandwidth requirements, in order to avoid the synchronization overhead present in the Asynchronous APB bridge.

5.10 Clock Gating Control (CGC)

The Clock Gating Control block (CGC) is a bank of registers which can be used to activate clock and reset signals for most AHB and S-APB peripheral. In this way only the peripherals actually required by the application need to be clocked, while all the other ones can be kept frozen or under reset so to reduce system power consumption. This block acts as a complement to the similar feature implemented by the A-APB bridge on its peripherals.

For most of the peripherals connected to AHB or S-APB subsystems there are three bits inside CGC block: reset on/off control, normal clock on/off control and debug clock on/off; an identical situation exists for A-APB peripherals which use a similar structure found in A-APB bridge. The list of controlled peripherals can be found in Table 83: *Peripheral clock and reset gating on page 366* where also their reset status can be found.

5.11 Universal Asynchronous Receiver/Transmitter (UART)

STR720 provides 2 Universal Asynchronous Receiver Transmitter peripherals (UART) to interface to other microcontrollers, microprocessors or external peripherals through serial communication. Both UARTs support full duplex asynchronous communication with external peripherals.

Data frames can be either 8-bit long (8 data bits or 7 data bits plus an automatically generated parity bit) or 9-bit long (9 data bits or 8 data bits plus an automatically generated parity bit or 8 data bits plus a wake up bit, useful for communication in multiprocessor systems). Parity, framing, and overrun error detection can be automatically added to increase the reliability of data transfers. An internal 16-bit baud rate generator is available: the clock for both transmit and receive channels can be obtained dividing the input clock by any divisor value from 1 to 2^{16}-1.

In order to reduce the number of interrupt to the ARM7, two internal FIFO's (16 words of 9 bits each) can be enabled via software, one for transmitted data and one for received data.

5.12 Buffered Serial Peripheral Interface (BSPI)

STR720 provides two Buffered Serial Peripheral Interfaces (BSPI). The BSPI implements an industry standard serial synchronous full duplex 4-pin interface, fully compliant with Motorola SPI protocol. The BSPI can be used to communicate with peripheral devices or it can be used for inter-processor communications in a multiple-master environment.

The buffered SPI can be configured to operate either in slave or in master mode, even if slave mode is subject to some limitations since Slave Select lines are shared with external interrupt 1 and 2 lines on P3.9 and P3.10 pads. Two internal FIFO's (16 words of 16 bits each) are available, one on the receive channel and one on the transmit channel. The BSPI can be programmed to operate with words 8 and 16 bit long. The BSPI serial clock frequency can be programmed dividing the input clock by an even value in the range starting from 6 to 254.

5.13 Controller Area Network Interface (CAN)

Two independent CAN interfaces are implemented on STR720. The Controller Area Network serial bus is compliant to the CAN Protocol Version 2.0 Part A (messages with 11 bit identifiers) and B (messages with 29 bit identifiers).

5.14 Universal Serial Bus Interface (USB)

STR720 provides a full speed USB slave interface, compliant with revision 1.1 of the USB standard, giving a peak transfer rate of 12 Mbits/s. This interface shares its data lines with pins P2.14 and P2.15, while its clock source is taken from an external clock line, shared with P2.10, which must be set to 48 MHz (50% duty cycle) when the USB is required. STR720 system features built-in USB transceivers which requires two series resistors of 27 $\Omega \pm 5\%$, one for each data line, to adapt the output impedance according to USB standard specifications.

Eight Endpoints are available; each may be software-configured to be Control/Interrupt/Bulk/ Isochronous. Double-buffering is supported for Bulk and Isochronous transfers.

CRC generation/checking, NRZI encoding/decoding, bit-stuffing and packet filtering are handled by hardware; interrupt is generated at the end of each packet correctly sent/received, or on transmission errors. Packets are stored in an internal buffer memory implemented as a single port 512 x 16 bit RAM.

USB suspend/resume are fully supported, allowing to stop the system clocks while keeping the wake-up capability and be compliant with USB current consumption specifications for bus powered devices.

5.15 WatchDoG timer (WDG)

The watchdog timer represents one of the fail-safe mechanisms which have been implemented to prevent the controller from malfunctioning. It is able to generate a system reset upon expiration of a programmed period of time.

The watchdog timer can be enabled by software. When $\overline{\text{RSTIN}}$ pin is released, the Watchdog Timer is not enabled, and it must be activated by software: once enabled, it can no longer be disabled by software. A prescaler allows to widely program the WDG overflow period whose length depends upon the selected WDG clock using EE bit of WDTCR register: if APB clock is selected, writing EE to ‘0’, the period can range between 66 ns and 0.5 s (with APB clock at 30 MHz), otherwise if low frequency clock (4 kHz derived from the 32 kHz RTC clock) is selected, writing EE to ‘1’, the period can be extended between 500 µs to about 4200 s independently from system clock frequency.

To prevent a system Reset the application code shall write, with a particular sequence, the Watchdog Timer Control Register at regular intervals. At the end of a successful write operation the Watchdog Timer restarts to count from the preset value. In order to prevent a system reset, the user has to write the sequence A55A, 5AA5 in the WDTKR register of WDG peripheral.

The Watch DoG timer can also be used as a free-running timer, providing an interruption request when counting down counter reaches 0.

5.16 Extended Function Timer (EFT)

The STR720 system includes 2 Extended Function Timers (EFT). Each EFT consists of a 16-bit counter driven by a programmable prescaler.

They may be used for a variety of purposes, including pulse length measurement of up to two input signals (input capture) or generation of up to two output waveforms (output compare and PWM). Pulse lengths and waveform periods can be modulated from a very wide range using the timer prescaler.

Not all their alternate function I/O features are available: only module EFT1 external clock input function and one EFT2 output compare I/O line can be used. Regardless of the number

of available alternate function I/Os, the number of interrupt events each EFT block can generate is the same and it is indicated in Section 5.1.1: *IRQ Interrupt Vector Table on page 28*.

EFT1 external clock source can be selected by software (in place of system clock) to obtain time bases independent from the system clock frequency (usually dependent on the selected system configuration mode) or to count external events.

5.17 Σ–Δ Analog to Digital Converter (ADC)

For analog signal measurement, a 11.5-bit ENOB resolution A/D converter (ADC) with 4 multiplexed input channels. It returns the conversion result on 16 bits with a frequency of up to 950 Hz each channel. It is a second-order quad-channel Sigma-Delta converter specifically suited for low conversion rate application (512 times oversampling rate). The typical SINAD is 71 dB.

The ADC supports two different conversion modes. In the standard Auto Scan Continuous mode, the analog levels on each channel are sequentially and repeatedly sampled and converted. Alternatively, the Single Channel conversion mode, can be used where the analog level on a specified channel is repeatedly sampled and converted into a digital result without software intervention.

Each channel has its own result data register where the value obtained from the last completed conversion is stored. In both modes it is possible to associate a DMA transfer to the end of conversion interrupt without interrupting the ARM720T for the data transfer.

Only one interrupt source is implemented, called ‘End of conversion’ which is activated each time a conversion sweep is completed and the result is written on the data result registers.

5.18 General Purpose I/O Ports

The General Purpose IO (GPIO) Ports are programmable by software in several conditions: Input, Output, Alternate Function, Open Drain, Push-Pull, Bidirectional Weak and high impedance.

Three General Purpose I/O Ports are available on STR720 system: P2, composed of 16 bits, P3, composed of 13 bits, and P4, composed of 5 bits only. Although all their registers are described as 16-bit wide, the actual number of available pins corresponds to what is stated above and in Chapter 4: PIN DATA on page 14.

During and just after the reset no alternate function is active: in particular each Port bit is configured in Input mode (PC0=1, PC1=0, PC2=0). The output driver is in high impedance while the input section is active (see Chapter 23: GENERAL PURPOSE I/O PORTS on page 343).

Being each GPIO pin configured in Input during Reset phase, the IO pins are released to High Impedance condition. To avoid power consumption the user has to drive the IOs to stable

levels. After the reset phase each pin can be reconfigured by software according to the application requirements.

The input section of P2.13 is implementing a TTL Schmitt Trigger device, while all other ports are plain input buffer without hysteresis.

Table 17. GPIOs Reset configuration

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Port 2	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T
Port 3	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	I-T	N/A	N/A	N/A
Port 4	I-T	I-T	I-T	I-T	I-T	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

I-T *Input-Tristate TTL*
B-WD *Bidirectional Weak Pull-Down*

Summarizing, hereinafter the reset values of all Port Configuration and Data registers are reported.

Table 18. Port registers: Reset configuration

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Port 2	**PC02**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	PC12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	PC22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Port 3	**PC03**	1	1	1	1	1	1	1	1	1	1	1	1	1	N/A	N/A	N/A
	PC13	0	0	0	0	0	0	0	0	0	0	0	0	0	N/A	N/A	N/A
	PC23	0	0	0	0	0	0	0	0	0	0	0	0	0	N/A	N/A	N/A
Port 4	**PC04**	1	1	1	1	1	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	PC14	0	0	0	0	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	PC24	0	0	0	0	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

5.19 Dedicated Pins

The status under and after reset of the dedicated ports and pins is hereafter reported.

Table 19. Dedicated Pins Reset status

Pin (Note 1)	Status under Reset	Value under Reset
OSCIN	Input	-
OSCOUT	Output	not OSCIN
$\overline{\text{RSTIN}}$	Input	Z
$\overline{\text{RTCRST}}$	Input	Z
TCLK	Input	Z
$\overline{\text{TRST}}$	Input	Z
TMS	Input	Z

Table 19. Dedicated Pins Reset status

Pin (Note 1)	Status under Reset	Value under Reset
TDI	Input	Z
TDO	Output Tristate	Z
Port1[0:3]	Analog Input	-
Port1[6:7]	High impedance	Z
Port6[0:31]	Input	Z
Port6[32:45]	Output	0
Port6[47]	Output	0
Port6[48]	Output	1
Port6[49:59]	Output	1
Port6[53:59]	Output	1
Port7[0:15]	Input	Z
Port7[16:37]	Output	0
Port7[38:43]	Output	1
Port7[44:45]	Input	Z

Note1: to avoid power consumption the user has to drive the input pins to stable values.

JTAG input pins require special attention since they do not implement any internal resistor: external devices should be connected to guarantee proper levels and to avoid possible spurious consumption of the input logic and/or undesired malfunctioning of the JTAG module.

When the system is put into STOP mode, all the pins maintains the status previously programmed while entering in STANDBY mode all pins are restored to their reset state.

5.20 Miscellanea Registers

The Miscellanea Registers module gathers all the registers that describe the current configuration of STR720 system. They can be accessed through the APB interface and allow a software configuration of some of the main features of the device. The complete list of configurable features can be found in Chapter 24: MISCELLANEA REGISTERS (GCR - CGC - AHB_ERR) on page 351. The most relevant features which can be configured are given below.

- Boot re-mapping (BRM) specifies where to find the boot program code. By default, it maps the boot-ROM or EMI area to address 0x0000_0000 after reset.
- DMA Stream selection bit (ST3_SEL) configure which peripherals will have access to DMA request line 3.
- Port7 Access Selection (P7AS) which enables to specify if Port7 is used for EMI or ATAPI-IDE interface.
- Address of last AHB transaction which resulted in a ERROR response (useful during application debug).

6 MEMORY ORGANIZATION

The memory space of the STR720 device is configured in a Von Neumann architecture. Code memory, data memory, registers and I/O ports are organized within the same linear address space of 4 GBytes.

The entire memory space is sub-divided in 8 main blocks (512 MBytes each), selected by the most significant bits of the AHB address bus.

There are three external blocks in the STR720 memory map: the SDRAM area, the ATAPI-IDE area and the EMI area, all using little endian format.

In the following sub-sections the relevant memory areas are briefly described.

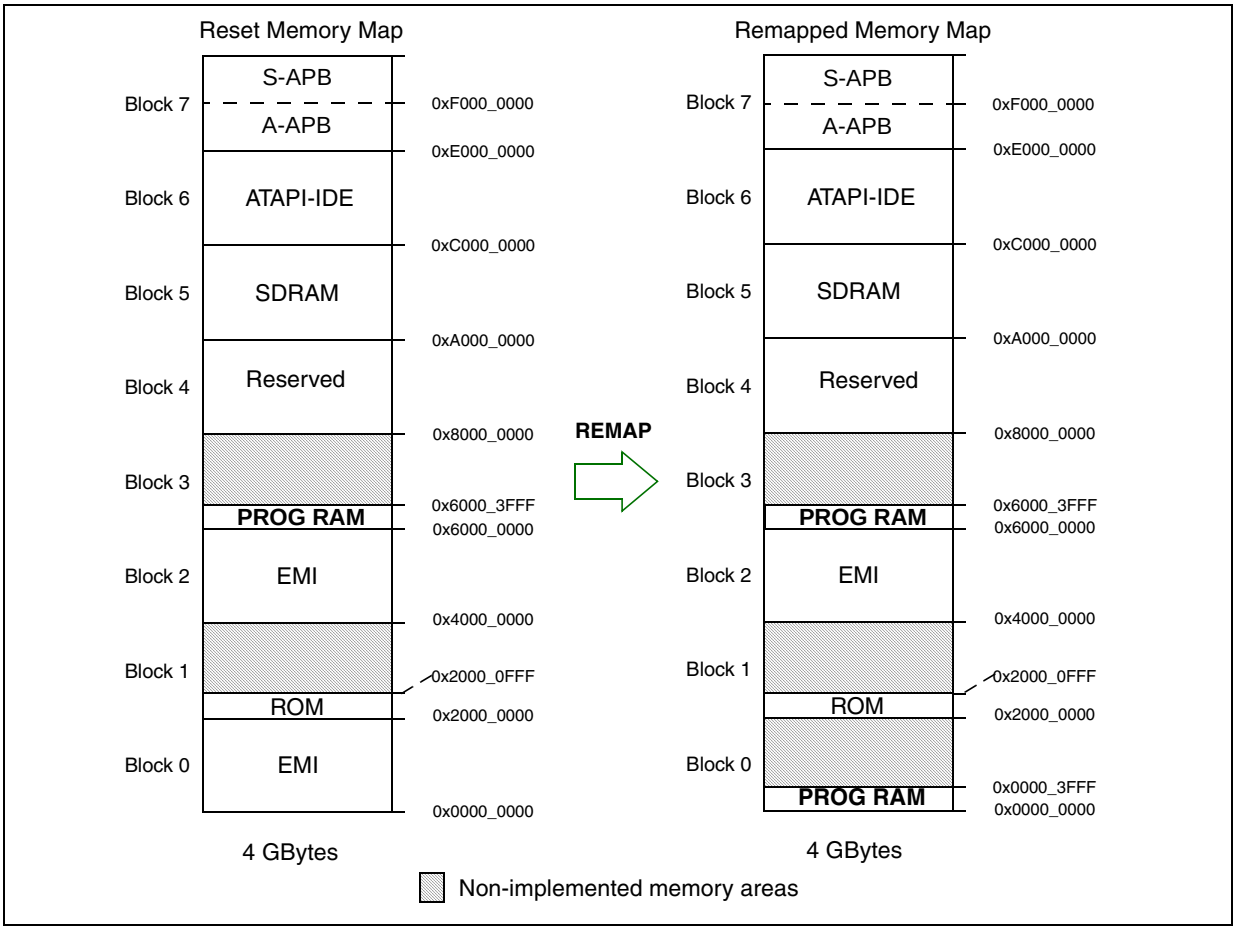
6.1 Program memory

STR720 implements a 16 KBytes program RAM block, located in the address range from 0x6000_0000 to 0x6000_3FFF and organized as a 4K of 32-bit wide memory.

Immediately after system reset, program memory is accessible only at the above mentioned address range within Block 3, allowing its initialization during the boot procedure when the code is fetched from EMI area (external-boot mode). At the end of boot sequence a specific GCR register is written to modify the memory map (“Boot re-map” operation). After the “Boot re-map” operation, Program Memory will be available also from address 0x0000_0000 (Block 0) to allow application code execution.

Program Memory can be used for both code (instructions) and data (constants, tables, etc.) storage. Code fetches are always made on even byte addresses. Word data accesses can be made to even or odd byte addresses.

6.2 Memory map



6.3 APB Bridges Mapping

APB bridges allow the connection between the high performance AHB bus, where ARM720, DMA, memory block and memory interfaces are located, to the peripherals residing on the APB bus. Two separated APB subsystems have been implemented on STR720 device: Asynchronous APB subsystem and Synchronous APB subsystem. They are briefly described in the following sections, along with the related peripheral address mapping.

6.3.1 Asynchronous APB sub-system (A-APB)

In order to reduce the power consumption of the overall device, the Asynchronous APB sub-system, in short A-APB, can be clocked at a frequency different from the one used for ARM720T core. A specific asynchronous AHB-APB bridge provides a re-synchronization mechanism in order to enable the master of AHB bus to access the APB peripherals although residing on a different clock domain.

The A-APB memory space decoding is provided by the A-APB bridge which assigns a 4 KBytes memory area to each peripheral. Table 20 reports the A-APB memory mapping.

Table 20. A-APB sub-system memory map

Peripheral Name	Start Address	End Address	Data Size	Peripheral Register Map
Asynchronous AHB-APB Bridge (A3BRG)	0xE000_0000	0xE000_0FFF	32	Section 15.4.1
A-APB Global Control Register (A-GCR)	0xE000_1000	0xE000_1FFF	16	Section 24.5.1
General Purpose I/O 2 (GPIO2)	0xE000_2000	0xE000_2FFF	16	Section 23.4.1
Buffered Serial Port Interface 1 (BSPI1)	0xE000_3000	0xE000_3FFF	16	Section 17.4.1
Buffered Serial Port Interface 2 (BSPI2)	0xE000_4000	0xE000_4FFF	16	Section 17.4.1
Universal Async. Receiver/Transm. 1 (UART1)	0xE000_5000	0xE000_5FFF	16	Section 16.4.1
Universal Async. Receiver/Transm. 2 (UART2)	0xE000_6000	0xE000_6FFF	16	Section 16.4.1
Extended Function Timer 1 (EFT1)	0xE000_7000	0xE000_7FFF	16	Section 21.4.1
Extended Function Timer 2 (EFT2)	0xE000_8000	0xE000_8FFF	16	Section 21.4.1
Analog Digital Converter (ADC)	0xE000_9000	0xE000_9FFF	16	Section 22.4.1
Controller Area Network Interface (CAN)	0xE000_A000	0xE000_AFFF	16	Section 18.7.11
Universal Serial Bus Interface (USB)	0xE000_B000	0xE000_BFFF	16	Section 19.6.4
Watch-DoG (WDG)	0xE000_C000	0xE000_CFFF	16	Section 20.4.1

6.3.2 Synchronous APB sub-system (S-APB)

Some peripherals require to be tightly coupled with ARM720 core for performance reasons and some others have both AHB and APB ports, typically separating data from and control information. In both cases it is required to have an APB sub-system which runs synchronously with the AHB clock frequency. A plain AHB-APB bridge is used to connect these synchronous APB peripherals to the system. The S-APB memory space decoding is provided by the S-APB bridge which assigns a 1 KByte memory area to each peripheral. Table 21 reports the S-APB memory mapping:

Table 21. S-APB sub-system memory map

Peripheral name	Start Address	End Address	Data Size	Peripheral Register Map
DRAM controller (DRAMC)	0xF000_0000	0xF000_03FF	16	Section 10.4.1
External Memory Interface (EMI)	0xF000_0400	0xF000_07FF	16	Section 11.4.1
Direct Memory Access Controller (DMAC)	0xF000_0800	0xF000_0BFF	16	Section 9.4.1
S-APB Global Control Register (S-GCR)	0xF000_0C00	0xF000_0FFF	16	Section 24.5.1
Reset and Clock Control Unit (RCCU)	0xF000_1000	0xF000_13FF	16	Section 13.4.1
General Purpose I/O 3 (GPIO3)	0xF000_1400	0xF000_17FF	16	Section 23.4.1
General Purpose I/O 4 (GPIO4)	0xF000_1800	0xF000_1BFF	16	Section 23.4.1
Reserved	0xF000_1C00	0xF000_1FFF	32	
Reserved	0xF000_2000	0xF000_23FF	32	
Wake-up/Interrupt Unit (WIU)	0xF000_2400	0xF000_27FF	16	Section 8.5
Real Time Clock (RTC)	0xF000_2800	0xF000_2BFF	16	Section 14.4.1
Clock gating control block (CGC)	0xF000_2C00	0xF000_2FFF	16	Section 24.5.1
AHB Error decoder (AERR)	0xF000_3000	0xF000_33FF	32	Section 24.5.1
Reserved	0xF000_3400	0xF000_37FF	-	
Reserved	0xF000_3800	0xF000_3BFF	-	
Enhanced Interrupt Controller (EIC)	0xF000_3C00	0xF000_3FFF	32	Section 7.4.1

In order to provide an effective usage of EIC vectorized interrupt features, S-APB sub-system is mapped on multiple address ranges: the first one starts from 0xF000_0000 and ends to 0xF000_3FFF then the first replica begins, using the addresses from 0xF000_4000 to 0xF000_7FFF and so on up to the last address range duplicate which extends from 0xFFFF_C000 to 0xFFFF_FFFF. Within this last S-APB address range image the EIC registers appears starting from 0xFFFF_FC00, allowing to reach IVR register at absolute address 0xFFFF_FC18, inside the range supported by the indirect jump instruction stored at IRQ vector location (0x0000_0018).

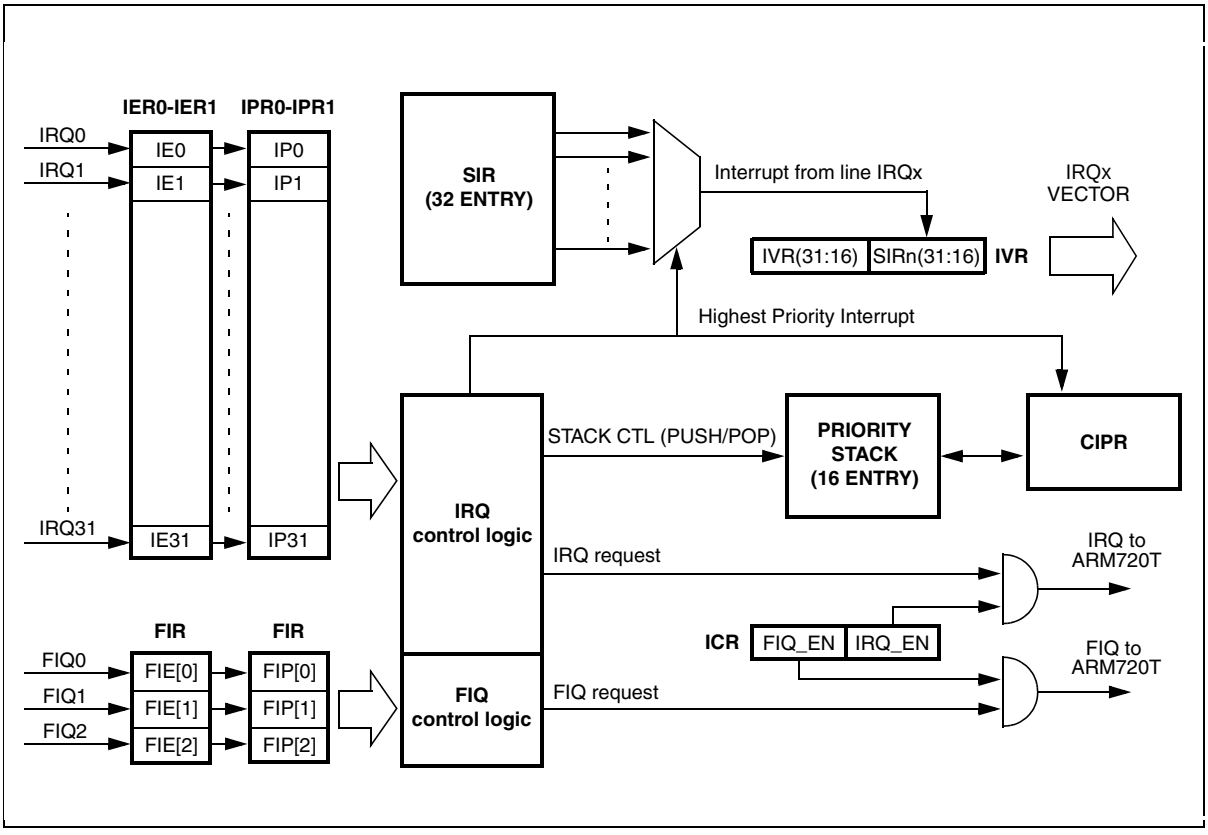
7 ENHANCED INTERRUPT CONTROLLER (EIC)

7.1 Introduction

The ARM720T CPU provides two levels of interrupt, FIQ (Fast Interrupt Request) for fast, low latency interrupt handling and IRQ (Interrupt Request) for more general interrupts.

Hardware handling of multiple interrupt channels, interrupt priority and automatic vectorization require therefore a separate Enhanced Interrupt Controller (EIC).

Figure 7. EIC block diagram



7.2 Main Features

- 32 maskable interrupt channels, mapped on ARM's interrupt request pin IRQ
- 16 programmable priority levels for each interrupt channels mapped on IRQ
- hardware support for interrupt nesting (16 levels)
- 3 maskable interrupt channels, mapped on ARM's interrupt request pin FIQ, with neither priority nor vectorization

7.3 Functional Description

The EIC (Enhanced Interrupt Controller) implements an interrupt structure pointing the processor at the first instruction location of the channel-specific Interrupt (IRQ) Service Routine.

IVR (Interrupt Vector Register) is the EIC's 32-bit register at address offset + 18h acting as pointer. It is composed by two main fields: the upper half word (16 bit) is directly programmable, while the lower half word is the mirrored entry of a register table (named SIR) indexed by the interrupt channel. The upper part is generally meant to contain jump opcode or base jump address depending on the application. The lower part supplies different jump offsets as a function of the interrupt channel.

The FIQ interrupt channels have not been provided with any vectoring nand/nor priority mechanism.

The EIC supports a fully programmable interrupt priority structure. Sixteen priority levels are available to define the channel priority relationships. Each channel has a 4-bit field, that defines its priority level in the range 0 - 15.

The on-chip peripheral channels and the external interrupt sources can be programmed within these 16 priority levels, level 15 indicates the highest priority, whereas level 0 the lowest.

If several units are located at the same priority level, an internal daisy chain, fixed for each device and depending on which interrupt input line the device is connected to, defines the priority relationship within that level. The higher is the input line index, the higher is the priority in the daisy chain. The user should keep this in mind avoiding that an interrupt, with a lower rank in the internal daisy chain, is never executed because it is at the same priority level of another interrupt channel with higher occurrence rate.

Example:

Supposing the following hardware configuration is implemented:

Module A: Interrupt channel 2
Module B: Interrupt channel 6

If both the priority levels have been set to 5, and both modules generate simultaneously an interrupt request to the EIC, the request from Module B is served first.

7.3.1 Priority Level Arbitration

The SIPL (Source Interrupt Priority Level) field in the SIRs (Source Interrupt Register) provides the priority level of each of the 32 interrupt sources.

Every clock cycle all the pending interrupts (stored in the IPR register) and their priorities are evaluated, and a winner channel (if any) is elected.

An interrupt requesting channel, to be the winner, must:

- be enabled (IER - Interrupt Enable Register)
- have the highest priority level related to the current interrupt requests, and higher than the current one (CIPR - Current Interrupt Priority Register)
- have the highest position in the chain of the pending interrupts with the same SIPL value

The CIPR provides the priority of the interrupt (IRQ type) currently serviced. CIPR is set to 0 (lowest priority) upon reset and can be modified during program execution, updating the register with a value that MUST be greater than, or equal to, the initial priority of the IRQ currently serviced (if it isn't, the write operation has no effect).

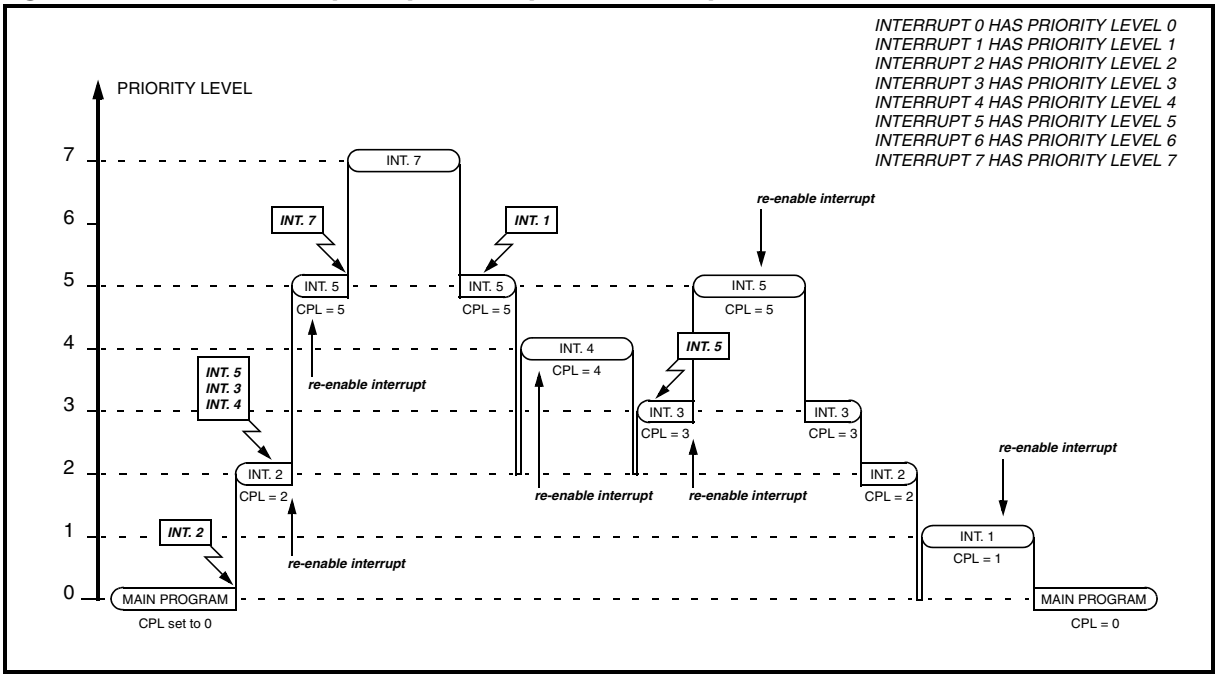
For safe operations it's better to disable the IRQ, before attempting to modify the CIPR register, to avoid dangerous race conditions.

The interrupt arbitration is stopped as soon as an interrupt is accepted by the processor (i.e. the IVR - Interrupt Vector Register is read and the interrupt service routine starts), and the CIPR and CICR registers are updated (see Figure 8: INTx represents a generic interrupt request of priority x).

One clock later, after the priority stack update, the arbitration starts once again.

Before exiting the interrupt service, the CPU has to clear the related interrupt pending bit in the EIC, to communicate the end of interrupt; this is possible by writing a '1' to the proper IPR bit.

Figure 8. Nested interrupt request sequence example



The IPR is a read/clear register, so writing a ‘0’ has no effect, while writing a ‘1’ reset the related bit. Therefore refrain from using read-modify instructions to avoid corruption of the IPR status.

Dynamic priority level modification: the main program and routines can be specifically prioritized. Since CIPR is a read/write register, it is possible to modify dynamically the current priority value during program execution. This means that a critical section can have a higher priority with respect to other interrupt requests. A subsequent lowering of the priority is allowed only down to the initial level of interrupt routine. Such initial priority level has been, of course, conveniently stored inside the EIC hardware stack.

Note that it is likewise possible to even manipulate the Main Program execution priority.

Maximum depth of nesting: no more than 15 routines can be nested (one for each priority level, from 1 to 15). If an interrupt routine at level N is being serviced, no other Interrupts located at level N can interrupt it. This implies a maximum number of 16 nested levels including the main program priority level.

Priority level 0: Interrupt requests at level 0, even if enabled, cannot be acknowledged, as their priority cannot be strictly higher than the CIPR value. The pending bit in the IPR (Interrupt Pending Register) is anyway set. Such a feature comes useful in a totally interrupt polled environment.

The whole interrupting process is constituted by the following subsequent major steps:

- Every clock cycle the IPR (Interrupt Pending Register) is updated and the pending interrupts are arbitrated to finally activate the IRQ/FIQ request to the CPU.
- Once the interrupt request has been accepted by the CPU, the program execution jumps at the first instruction of the related interrupt channel software routine (see Section 7.5 for more details).

The enhanced interrupt controller will automatically:

- Reject/accept an interrupt request according to the channel enable bit (IER - Interrupt Enable Register).
- Compare all interrupt requests (pending or new) with the current priority interrupt level. If the priority of the interrupt requests is higher than the current priority, an interrupt IRQ is asserted.
- The register 0x18 is loaded with the corresponding address vector.
- When an interrupt request is accepted, the previous interrupt priority and channel ID (content of CIPR and CICR registers) are saved by the EIC in the EIC's hardware stack.
- The current priority register (CIPR) and the current channel register (CICR) are automatically updated with the accepted interrupt priority and channel ID.
- When the pending bit in the IPR is cleared, the end of interrupt condition is detected, and the EIC restores in the CIPR and the CICR the previous interrupt priority and channel ID saved in the hardware stack.

The EIC allows to mask all the interrupt sources by resetting the enable bits in the ICR (Interrupt Control Register).

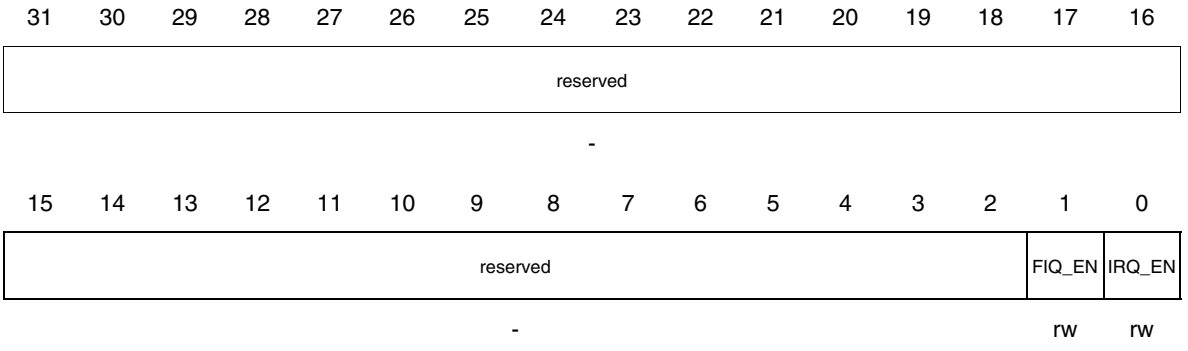
7.4 Register Description

Reading from the “Reserved” field in any register will return ‘0’ as result. Attempt to write in the same field has no effect.

Interrupt Control Register (ICR)

Address Offset: 00h

Reset value: 0000 0000h



Bit 31:2 = *Reserved.*

Bit 1= **FIQ_EN:** *Global FIQ output ENable bit*
This bit enables FIQ output request to the CPU.
1: Enhanced Interrupt Controller FIQ output request to CPU is enabled
0: Enhanced Interrupt Controller FIQ output request to CPU is disabled, even if the EIC logic has detected valid and enabled fast interrupt requests at its inputs

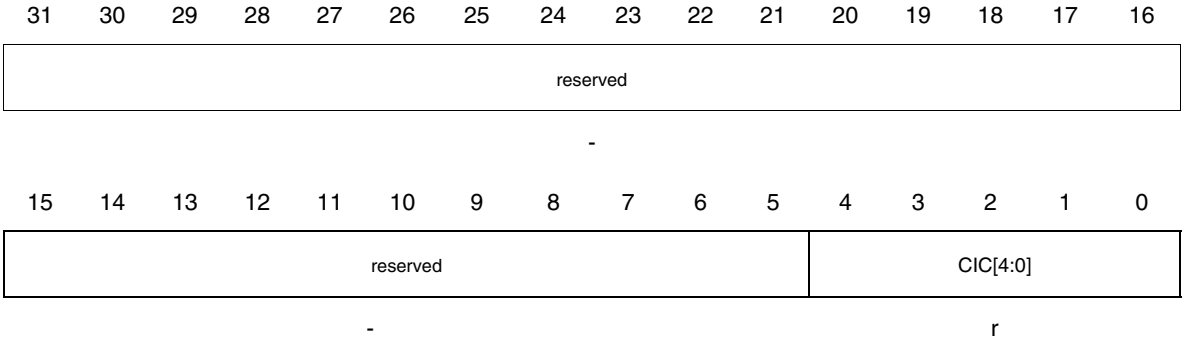
Bit 0 = **IRQ_EN:** *Global IRQ output ENable bit*
This bit enables IRQ output request to the CPU.
1: Enhanced Interrupt Controller IRQ output request to CPU is enabled
0: Enhanced Interrupt Controller IRQ output request to CPU is disabled, even if the EIC logic has detected valid and enabled interrupt requests at its inputs

The ICR register is an enable register at the EIC IRQ/FIQ outputs level: it has no influence on the EIC internal logic behaviour.

Current Interrupt Channel Register (CICR)

Address Offset: 04h

Reset value: 0000 0000h



The CIC Register reports the number of the interrupt channel currently serviced. There are 32 possible channel IDs (0 to 31), so the significant register bits are only five (4 down to 0).

After reset, the CIC value is set to '0' and is updated by the EIC logic only after the processor has started servicing a valid IRQ interrupt request.

To make this happen, some EIC registers must be set as in the following:

- ICR IRQ_EN =1
- IER not all 0 (at least one interrupt channel must be enabled)
- among the interrupt channels enabled by the IER registers, at least one must have the SIPL field, of the related SIR register, not set to 0, because the EIC generates a processor interrupt request (asserting the nIRQ line) **ONLY IF** it detects an enabled interrupt request whose **priority value is bigger than the CIPR (Current Interrupt Priority Register) value.**

When the nIRQ line is set, the processor will read the EIC IVR (Interrupt Vector Register), and this read operation will advise the EIC logic that the ISR (Interrupt Service Routine) has been initiated and the CICR register can be properly updated.

The CICR value can't be modified by the CPU (read only register).

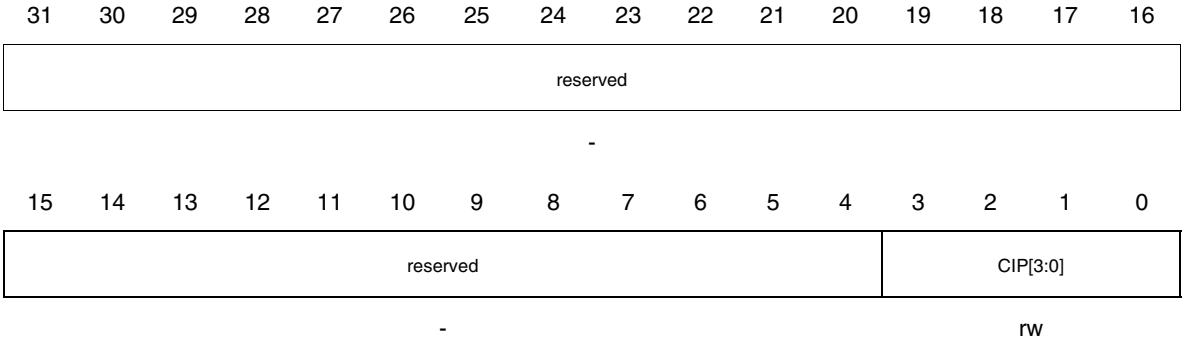
Bit 31:5 = *Reserved.*

Bit 4:0 = **CIC[4:0]:** *Current Interrupt Channel*.
Number of the interrupt whose service routine is currently in **execution** phase.

Current Interrupt Priority Register (CIPR)

Address Offset: 08h

Reset value: 0000 0000h



The CIP Register reports the priority value of the interrupt currently serviced. There are 16 possible priority values (0 to 15), so the significant register bits are only four (3 down to 0). After reset, the CIP value is set to '0' and is updated by the EIC logic only after the processor has started servicing a valid IRQ interrupt request.
To make this happen, some EIC registers must be set as in the following:

- ICR IRQ_EN =1 (to have the nIRQ signal set)
- IER not all 0 (at least one interrupt channel must be enabled)
- among the interrupt channels enabled by the IER register, at least one must have the SIPL field, of the related SIR register, not set to 0, because the EIC generates a processor interrupt request (asserting the IRQ line) **ONLY IF** it detects an enabled interrupt request whose **priority value is bigger than the CIPR (Current Interrupt Priority Register) value.**

When the IRQ line is set, the processor will read the EIC IVR (Interrupt Vector Register), and this read operation will advise the EIC logic that the ISR (Interrupt Service Routine) has been initiated and the CIPR register can be properly updated.
The CIPR value can be modified by the processor, to promote a running ISR to a higher level: the EIC logic will allow a write to the CIP field of any value bigger, or equal, than the priority value associated to the interrupt channel currently serviced.
E.g.: suppose the IRQ signal is set because of an enabled interrupt request on channel #4, whose priority value is 7 (i.e. SIPL of SIR7 is 7); after the processor read of IVR register, the EIC will load the CIP field with 7. Until the interrupt service procedure will be completed, writes of values 7 up to 15 will be allowed, while attempts to modify the CIP content with priority lower than 7 will have no effect.

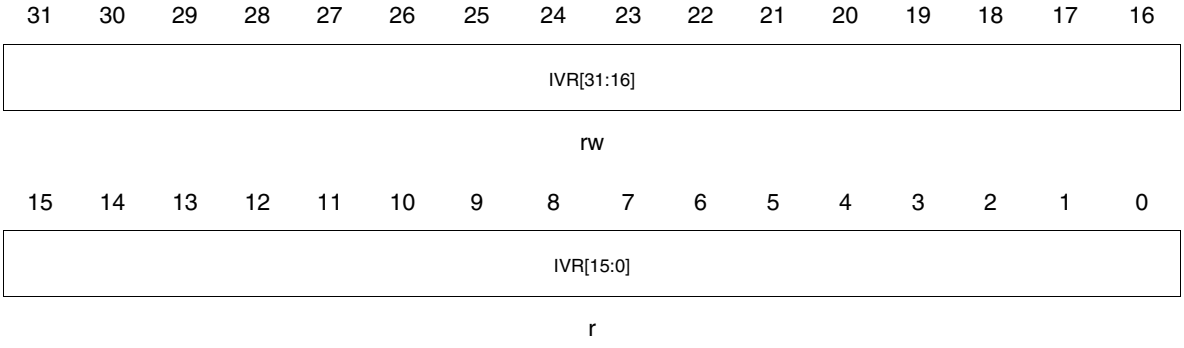
Bit 31:4 = *Reserved.*

Bit 3:0 = **CIP[3:0]:** *Current Interrupt Priority.*
Priority value of the interrupt whose service routine is currently in execution phase.

Interrupt Vector Register (IVR)

Address Offset: 18h

Reset value: 0000 0000h



The IVR is the EIC register the processor has to read after detecting the nIRQ signal assertion.

The IVR read operation tells the EIC logic that the interrupt service routine (ISR), related to the pending request, has been initiated.

This means that:

- the IRQ signal can be de-asserted
- the CIPR and CICR can be updated
- no interrupt requests, whose priority is lower, or equal, than the current one can be processed

Bit 31:16 = **IVR(31:16)**: *Interrupt Vector (High portion).*
its content does not depend on the interrupt to be serviced, but has to be programmed by the user (see Note) at initialization time and it's common to all the interrupt channels. It is Read/ Write.

Bit 15:0 = **IV(15:0)**: *Interrupt Vector (Low portion).*
its content depends on the interrupt to be serviced (i.e. the one, in the enabled bunch, with the highest priority), and it's a copy of the SIV (Source Interrupt Vector) value of the SIR (Source Interrupt Register) related to the channel to be serviced. It is Read only.

Note The EIC logic does not care about the IVR content: from the controller point of view it's a simple concatenation of two 16-bit fields

- **IVR = IVR(31:16) & SIRn(31:16)**

What has to be written both in the IVR(31:16) and in all the SIRn(31:16) is mostly implementation dependent, but two main settings can be recognized:

- IVR(31:0) contains a 32 bit address

- IVR(31:0) contains an instruction

In the first case (address), IVR(31:16) will have to be loaded with the higher part of the address pointing to the memory location where the interrupt service routines begin; the single SIRn(31:16) will have to contain the lower 16 bits (offset) of the memory address related to the channel specific ISR.

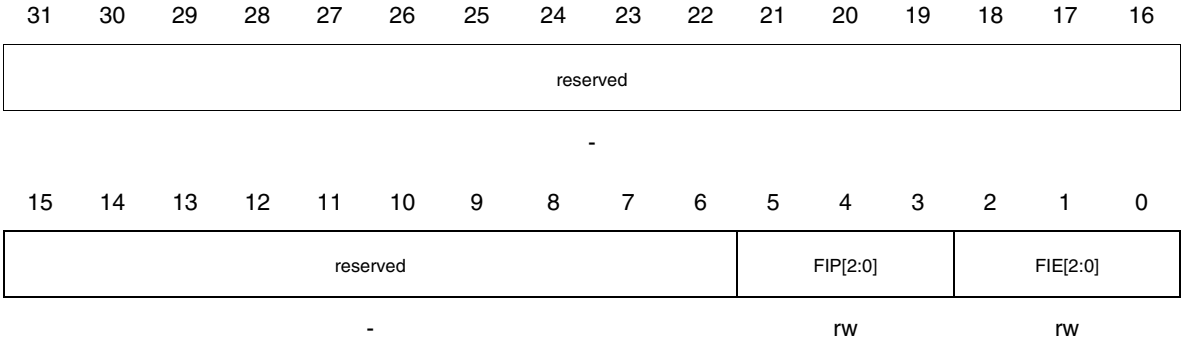
If required and supported by the system/processor configuration, it can also be convenient to initialize IVR(31:16) and SIRn(31:16) to provide an instruction opcode (typically a jump) on IVR register read.

In this situation, IVR(31:16) has to contain the higher part of the instruction to be passed, while SIRn(31:16) will be loaded with the channel depending parameters (usually jump offset).

Fast Interrupt Register (FIR)

Address Offset: 1Ch

Reset value: 0000 0000h



In order for the controller to react to the 3 fast-interrupt (FIQ) channels the enable bits from 2 down to 0 must be set to 1. The field 5 down to 3 keeps the information on the interrupt request of the specific channel.

Bit 31:6 = *Reserved*.

Bit 5 = **FIP[2]:** *Channel 2 Fast Interrupt Pending Bit*

Bit 4 = **FIP[1]:** *Channel 1 Fast Interrupt Pending Bit*

Bit 3 = **FIP[0]:** *Channel 0 Fast Interrupt Pending Bit*

These bits are set by hardware by a Fast interrupt request on the corresponding channel. These bits are cleared only by software, i.e. writing a '0' has no effect, whereas writing a '1' clears the bit (forces it to '0').
1: Fast Interrupt pending
0: No Fast interrupt pending

Bit 2 = **FIE[2]:** *FIQ Channel 2 Interrupt Enable bit*

Bit 1 = **FIE[1]:** *FIQ Channel 1 Interrupt Enable bit*

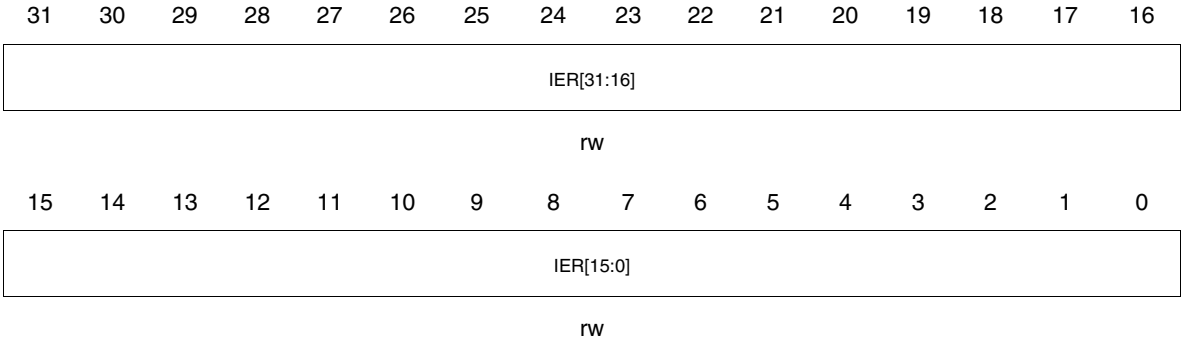
Bit 0 = **FIE[0]:** *FIQ Channel 0 Interrupt Enable bit*

In order to have the controller responding to a request on a specific channel, the corresponding bit in the FIR register must be set to 1.
1: Fast Interrupt request issued on the specific channel is enabled
0: Fast Interrupt request issued on the specific channel is disabled

Interrupt Enable Register 0 (IER0)

Address Offset: 20h

Reset value: 0000 0000h



Bit 31 = **IER[31]:** *Channel 31 Interrupt Enable bit*

Bit 30 = **IER[30]:** *Channel 30 Interrupt Enable bit*

...

Bit 1 = **IER[1]:** *Channel 1 Interrupt Enable bit*

Bit 0 = **IER[0]:** *Channel 0 Interrupt Enable bit*

The IER0 is a 32 bit register: it provides an enable bit for each of the 32 EIC interrupt input channels.

In order to enable the interrupt response to a specific interrupt input channel the corresponding bit in the IER0 register must be set to ‘1’.

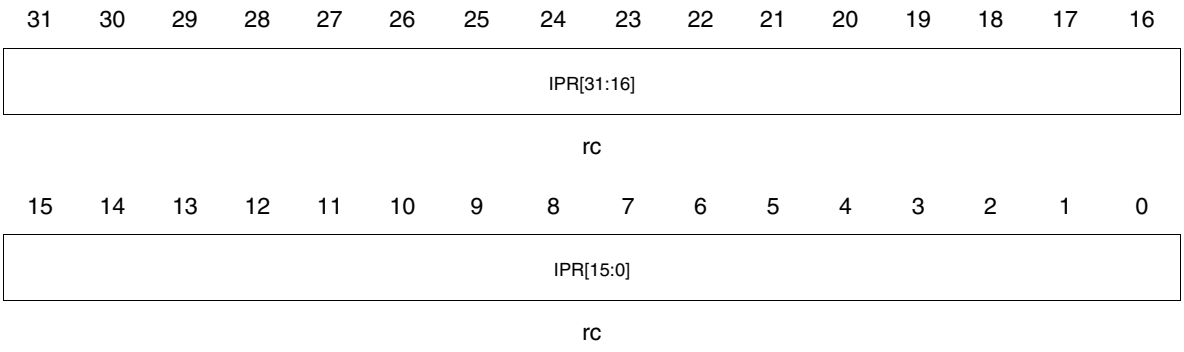
A ‘0’ value makes the EIC mask the related interrupt channel and the interrupt pending bit, in the IPR register, is never set.

1: Input channel enabled
0: Input channel disabled

Interrupt Pending Register 0 (IPR0)

Address Offset: 40h

Reset value: 0000 0000h



Bit 31= **IPR[31]:** *Channel 31 Interrupt Pending bit*

Bit 30 = **IPR[30]:** *Channel 30 Interrupt Pending bit*

...

Bit 1 = **IPR[1]:** *Channel 1 Interrupt Pending bit*

Bit 0 = **IPR[0]:** *Channel 0 Interrupt Pending bit*

The IPR0 is a 32 bit register: it provides a pending bit for each of the 32 EIC interrupt input channels.

This is where actually the information about the channel interrupt status is kept: if the corresponding bit in the enable register IER0 has been set, the IPR0 bit set high (active) means that the related channel has asserted an interrupt request that has not been serviced yet.

The bits are Read/Clear, i.e. writing a ‘0’ has no effect, whereas writing a ‘1’ clears the bit (forces it to ‘0’).

1: Interrupt pending
0: No interrupt pending

Note *Before exiting the ISR (Interrupt Service Routine), the software must be sure to have cleared the EIC IPR0 bit related to the executed routine: this bit clear will be read by the EIC logic as End of Interrupt (EOI) sequence, and will allow the interrupt stack pop and new interrupts processing .*

Remark:

The Interrupt Pending bits must be handled with care because the EIC State Machine, and its internal priority hardware stack, could be forced to a not recoverable condition if unexpected pending bit clear operations are performed.

Example:

1. suppose that one or more interrupt channels are enabled, with a priority higher than zero; as soon as an interrupt request arises, the EIC FSM processes the new input and asserts the nIRQ signal. If, before reading the IVR (0x18) register, for any reason, the processor clears the pending bits, the nIRQ signal will remain asserted the same, even if no more interrupts are pending.

The only way to reset the nIRQ line logic is to read the IVR (0x18) register.

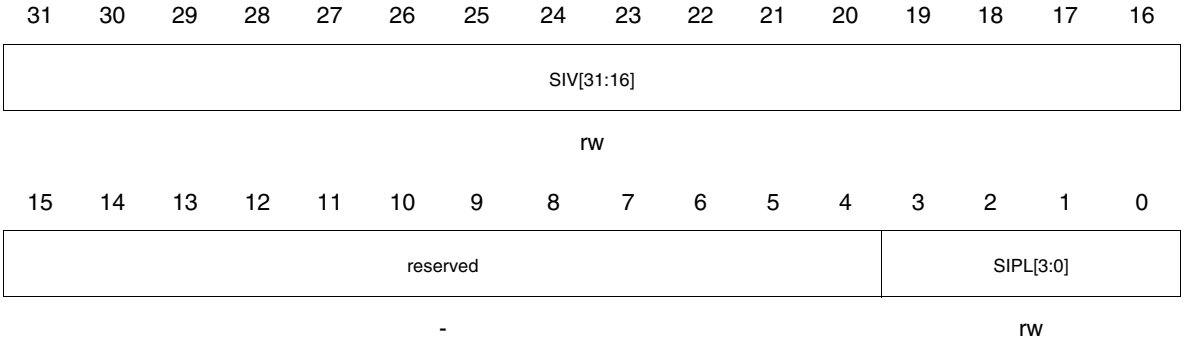
2. suppose that one or more interrupt channels are enabled, with a priority higher than zero; as soon as an interrupt request arises, the EIC FSM processes the new input and asserts the nIRQ signal. If, after reading the IVR (0x18) register, for any reason, the processor clears the pending bit related to the serviced channel, before completing the Interrupt Service Routine, the EIC logic will detect an End Of Interrupt command, will send a pop request to the priority stack and a new interrupt, even of lower priority, will be processed.

To close an interrupt handling section End Of Interrupt (EOI), the interrupt pending clear operation must be performed, at the end of the related Interrupt Service Routine, on the pending bit related to the serviced channel; on the other hand, as soon as the pending bit of the serviced channel is cleared (even by mistake) by the software, the EOI sequence is entered by the EIC logic.

Source Interrupt Registers - Channel n (SIRn)

Address Offset: 60h to DCh

Reset value: 0000 0000h



There are 32 different SIRn registers, as many as the input interrupt channels are.

Bit 31:16 = **SIV[31:16]:** *Source Interrupt Vector for interrupt channel n (n = 0... 31).*
This field contains the interrupt channel depending part of the interrupt vector, that will be provided to the processor when the Interrupt Vector Register (Address 0x18h) is read.

Depending on what the processor expects (32 bit address or instruction, see IVR description), the SIV will have to be loaded with the interrupt channel ISR address offset or with the lower part (including the jump offset) of the first ISR instruction.

Bit 15:4: *Reserved*.

Bit 3:0 = **SIPL[3:0]:** *Source Interrupt Priority Level for interrupt channel n (n = 0... 31).*
These 4 bits allow to associate the interrupt channel to a priority value between 0 and 15. The reset value is 0.

Note *To be processed by the EIC priority logic, an interrupt channel must have a priority level higher than the current interrupt priority (CIP); the lowest value CIP can have is 0, so all the interrupt sources that have a priority level equal to 0 will never generate an IRQ request, even if properly enabled.*

7.4.1 Register map

Refer to Table 21 on page 50 for the base address.

Table 22. EIC register map

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**ICR**	reserved																														FIQ_EN	IRQ_EN
4	**CICR**	reserved																											CIC[4:0]				
8	**CIPR**	reserved																											CIP[3:0]				
18	**IVR**	Jump Instruction Opcode or Jump Base Address																Jump Offset															
1C	**FIR**	reserved																										FIP [2:0]			FIE [1:0]		
20	**IER0**	IER[31:0]																															
40	**IPR0**	IPR[31:0]																															
60	**SIR0**	SIV0[31:16]																reserved													SIPL0[3:0]			
64	**SIR1**	SIV1[31:16]																reserved													SIPL1[3:0]			
68	**SIR2**	SIV2[31:16]																reserved													SIPL2[3:0]			
6C	**SIR3**	SIV3[31:16]																reserved													SIPL3[3:0]			
70	**SIR4**	SIV4[31:16]																reserved													SIPL4[3:0]			
74	**SIR5**	SIV5[31:16]																reserved													SIPL5[3:0]			
78	**SIR6**	SIV6[31:16]																reserved													SIPL6[3:0]			
7C	**SIR7**	SIV7[31:16]																reserved													SIPL7[3:0]			
80	**SIR8**	SIV8[31:16]																reserved													SIPL8[3:0]			
84	**SIR9**	SIV9[31:16]																reserved													SIPL9[3:0]			
88	**SIR10**	SIV10[31:16]																reserved													SIPL10[3:0]			
8C	**SIR11**	SIV11[31:16]																reserved													SIPL11[3:0]			
90	**SIR12**	SIV12[31:16]																reserved													SIPL12[3:0]			
94	**SIR13**	SIV13[31:16]																reserved													SIPL13[3:0]			
98	**SIR14**	SIV14[31:16]																reserved													SIPL14[3:0]			
9C	**SIR15**	SIV15[31:16]																reserved													SIPL15[3:0]			
A0	**SIR16**	SIV16[31:16]																reserved													SIPL16[3:0]			
A4	**SIR17**	SIV17[31:16]																reserved													SIPL17[3:0]			
A8	**SIR18**	SIV18[31:16]																reserved													SIPL18[3:0]			
AC	**SIR19**	SIV19[31:16]																reserved													SIPL19[3:0]			
B0	**SIR20**	SIV20[31:16]																reserved													SIPL20[3:0]			
B4	**SIR21**	SIV21[31:16]																reserved													SIPL21[3:0]			
B8	**SIR22**	SIV22[31:16]																reserved													SIPL22[3:0]			
BC	**SIR23**	SIV23[31:16]																reserved													SIPL23[3:0]			
C0	**SIR24**	SIV24[31:16]																reserved													SIPL24[3:0]			
C4	**SIR25**	SIV25[31:16]																reserved													SIPL25[3:0]			
C8	**SIR26**	SIV26[31:16]																reserved													SIPL26[3:0]			
CC	**SIR27**	SIV27[31:16]																reserved													SIPL27[3:0]			
D0	**SIR28**	SIV28[31:16]																reserved													SIPL28[3:0]			
D4	**SIR29**	SIV29[31:16]																reserved													SIPL29[3:0]			
D8	**SIR30**	SIV30[31:16]																reserved													SIPL30[3:0]			
DC	**SIR31**	SIV31[31:16]																reserved													SIPL31[3:0]			

7.5 Programming considerations

Here are a few guideline steps on how to program the EIC's registers in order to get the controller up and running quickly. In the following, it is assumed to deal with standard interrupts and that the user wants, for example, to detect an interrupt on channel #22, in which the priority has been assigned to be 5.

First of all, it is necessary to assign the priority and the jump address data related to the interrupt channel #22. Therefore:

- set in field SIPL of SIR22 the binary "0101", i.e. priority of 5 (it must be not 0 to have an IRQ to be generated).

Two registers are involved to supply the channel interrupt vector data to the EIC controller (IVR[31:16] and SIR22[31:16]:

- write in SIR22[31:16], i.e. in the upper part of the SIR register related to channel #22, the memory address offset (or the jump offset) where the Interrupt Service Routine, related to interrupt channel #7, starts.
- insert the base jump address (or the jump opcode) in the most significant half of the IVR register, i.e. IVR[31:16].

Finally, response to the interrupt must be enabled both at the global level and at the single interrupt channel level. To do so these steps shall be followed:

- set the IRQ_EN bit of ICR to 1
- set bit # 22 of IER0 to 1

As far as the FIQ interrupts are concerned, since those interrupts do not have any vectorization nor priority, only the first two steps above are involved. Supposing the user wants to enable FIQ channel #1:

- set the FIQ_EN bit of ICR to 1.
- set bit #1 of FIE in FIR register to 1.

7.6 Application note

Every Interrupt Service Routine (ISR) should be composed by the following blocks of code.

- A **Header routine** to enter ISR. It must be:

```
1) STMFD sp!,{r5,lr} The workspace r5 plus the current return address lr_irq
                     is pushed into the system stack.
2) MRS r5,spsr       Save the spsr into r5
3) STMFD sp!,{r5}    Save r5
4) MSR cpsr_c,#0x1F  Reenable IRQ, go into system mode
5) STMFD sp!,{lr}    Save lr_sys for the system mode
```

Note r5 is a generic register among those available, namely r0 up to r12. Since there is no way to save SPSR directly into the ARM's stack, the operation is executed in two steps using r5 as transition help register.

- The ISR **Body routines**.
- A **Footer routine** to exit ISR. It must be:

```
1) LDMFD sp!,{lr}     Restore lr_sys for the system mode
2) MSR cpsr_c,#0xD2   Disable IRQ, move to IRQ mode
3) Clear pending bit in EIC (using the proper IPRx)
4) LDMFD sp!,{r5}     Restore r5
5) MSR spsr,r5        Restore Status register spsr
6) LDMFD sp!,{r5,lr}  Restore status lr_irq and workspace
7) SUBS pc,lr,#4      Return from IRQ and interrupt reenabled
```

In the following two paragraphs some comments on the code lines introduced above and some hints useful when realizing subroutines to be invoked by an ISR are reported.

7.6.1 Avoiding LR_sys and r5 registers content loss

A first example refers to a LR_sys content loss problem: it is assumed that an ISR without instruction 5) in the header routine (and consequently without instruction 1) in the footer routine) has just started; the following happens:

- Instruction 4) is executed (so system mode is entered)
- A subroutine is called with a BL instruction and LR_sys now contains the return address A: the first subroutine instruction should store register LR_sys into the stack (the address of this instruction is called B)
- A higher priority interrupt starts before previous operation could be executed
- New ISR stores address B into LR_irq and enters system mode
- A new subroutine is called with a BL instruction: LR_sys is loaded with the new return address C (this overwrites the previous value A!) which is now stored into the stack.
- The highest priority ISR ends and address B is restored: now LR_sys value can be put into the stack but its value has changed to address C (instead of A).

The work-around to avoid such a dangerous situation is to insert line 5) at the end of header routine and consequently line 1) at the beginning of footer routine.

Similar reasons could lead register **r5** to be corrupted. To fix this problem, lines 3) in header and 4) in footer should be added.

7.6.2 Hints about subroutines used inside ISRs

A case in which a subroutine is called by an ISR is hereafter considered.

Supposing that such a kind of procedure starts with an instruction like:

```
STMFD SP!, { ... , LR }
```

probably it will end with:

```
LDMFD SP!, { ... , LR }
MOV   PC, LR
```

If a higher priority IRQ occurs between the last two instructions, and the new ISR calls in turn another subroutine, LR content will be lost: so, when the last IRQ ends, the previously interrupted subroutine will not return to the correct address.

To avoid this, the previous two instructions shall be replaced with the unique instruction:

```
LDMFD SP!, { ... , PC }
```

which automatically moves stored link register directly into the program counter, making the subroutine correctly return.

7.7 Interrupt latency

As soon as an interrupt request is generated (either from external interrupt source or from an on-chip peripheral), the request itself must go through three different stages before the interrupt handler routine can start. The interrupt latency can be seen as the sum of three different contributions:

- Latency due to the synchronisation of the input stage. This logic can be present (e.g. synchronization stage on external interrupts input lines) or not (e.g. on-chip interrupt request), depending on the interrupt source. Either zero or two clock cycles delay are related to this stage.
- Latency due to the EIC itself. Two clock cycles are related to this stage.
- Latency due to the ARM720T Release 3 interrupt handling logic (refer to the documentation available on www.arm.com).

Table 23. Interrupt latency (in clock cycles)

	SYNCH. STAGE		EIC STAGE
	min	max	
FIQ	0	2	2
IRQ			

8 WAKE-UP INTERRUPT UNIT (WIU)

8.1 Introduction

The Wake-up/Interrupt Management Unit supports up of 16 external wake-up/interrupt lines. These 16 Wake-up pins (usually alternate function of general purpose I/O lines) can be programmed as external interrupt lines or as wake-up lines, capable of restoring the normal operations from those low power modes where system clocks is stopped.

In STR720 implementation, the 16 interrupt lines are associated to the different IRQ channels in the following way:

- 4 external wake-up lines all connected to the IRQ0 channel of the interrupt controller. These wake-up lines have dedicated input ports.
- 1 external wake-up line directly mapped to a dedicated channel of the interrupt controller (IRQ1). This wake-up line has a dedicated input port.
- 10 internal wake-up lines also connected to the IRQ2 channel of the interrupt controller module. The ‘internal” lines are mapped on some significant STR720 signals.

The mapping of wake-up/interrupt events on available WIU lines can be found in Section 5.2: *Wake-up/Interrupt management Unit (WIU) on page 34*, where also possible limitation of usage can be found.

For each wake-up/interrupt line a pending bit is available, so a 16-bit register contains the interrupt pending bits of the 16 external lines (WUPR).

Interrupt pending bits are set by hardware or software on occurrence of the trigger event and are reset by software.

A 16-bit register (WUMR) is used to mask the interrupt or wake-up requests coming from the 16 wake-up/interrupt lines: resetting the bits of this mask register prevents the interrupt or wake-up requests, generated on the corresponding lines.

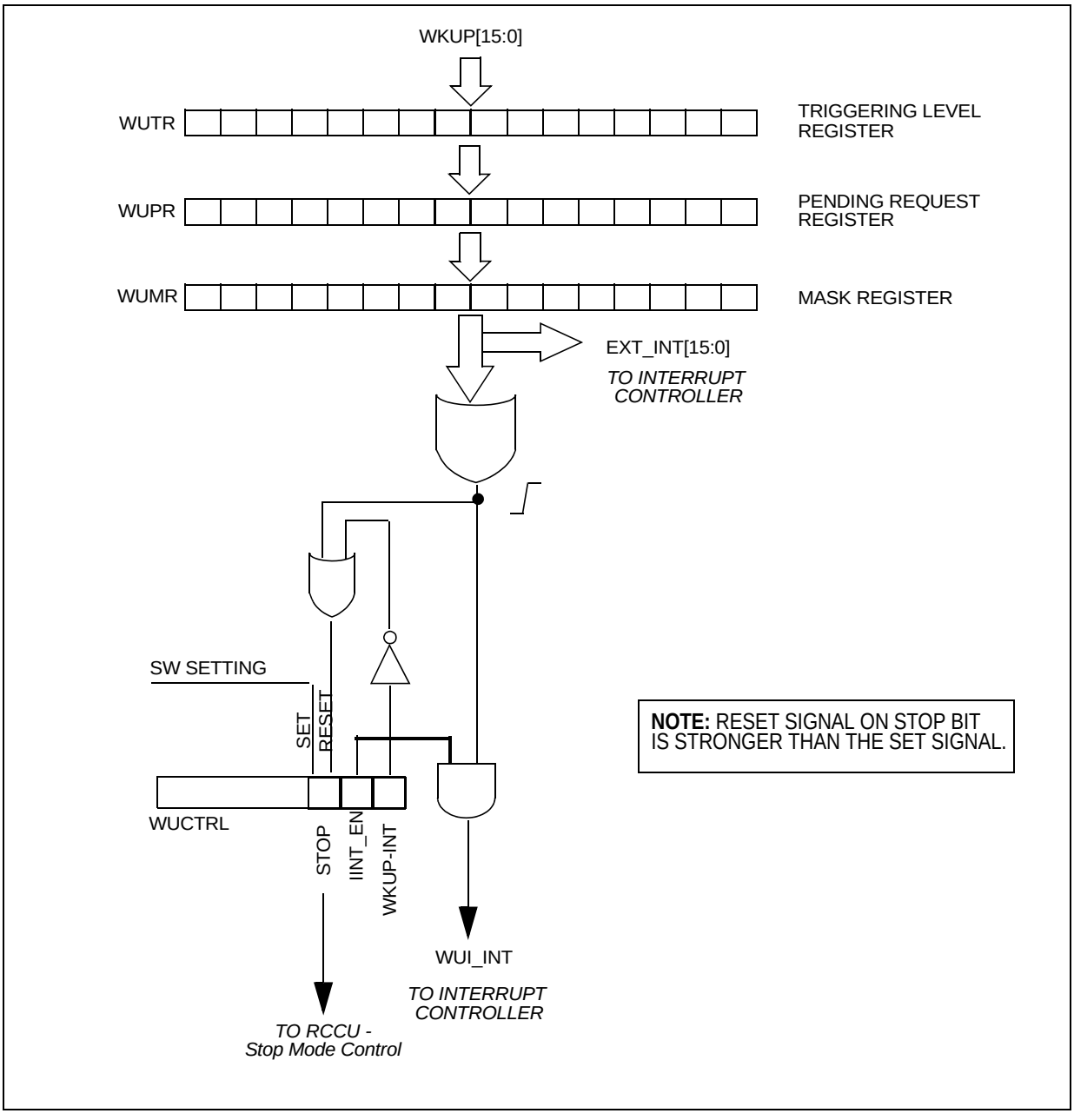
In order to give more flexibility to the user for the application, a 16-bit register (WUTR) allows to program the triggering on rising or falling edge of the external wake-up pins.

8.2 Main Features

- Supports 16 external wake-up or interrupt lines.
- Wake-Up lines can be used to wake-up the system from stopped clock conditions.
- Programmable selection of wake-up or interrupt.
- Programmable wake-up/interrupt trigger edge polarity.
- All Wake-Up Lines maskable.

8.3 Functional Description

Figure 9. Wake-Up/Interrupt Lines Management Unit Block Diagram



8.3.1 Interrupt Mode Selection.

To configure the 16 lines as interrupt sources, use the following procedure:

- Configure the mask bits of the 16 wake-up lines (WUMR);
- Configure the triggering edge register of the wake-up lines (WUTR);
- Set properly the enable and mask bits corresponding to the line of interrupt controller tied to Wake-up Unit: so an interrupt coming from one of the 16 lines can be correctly acknowledged;
- Reset the WKUP-INT bit in the WUCTRL register to disable Wake-up Mode (default value is 0);
- Set the INT_EN bit in the WUCTRL register to enable the 16 wake-up lines as external interrupt source lines.

8.3.2 Wake-up Mode Selection

To configure the 16 lines as wake-up sources, use the following procedure:

- Set the mask bits of the 16 wake-up lines (WUMR).
- Configure the triggering edge register of the wake-up lines (WUTR).
- Set the enable and the mask bits corresponding to the line of Interrupt Controller tied to Wake-up Unit (as for Interrupt Mode selection), only if an interrupt routine is to be executed after a wake-up event. Otherwise, if the wake-up event only restarts the execution of the code from where it was stopped, the interrupt channel shall be masked.
- Set the WKUP-INT bit in the WUCTRL register to select Wake-up Mode.
- Set the INT_EN bit in the WUCTRL register to enable the 16 wake-up lines as external interrupt source lines. This is not mandatory if the wake-up event does not require an interrupt response.
- Write the sequence 1,0,1 to the STOP bit of WUCTRL. This is the STOP bit setting sequence. Pay attention that the three write operations are effective even though not executed in a strict sequence (intermediate instructions are allowed, except for write instructions to the registers of Wake-Up). If the sequence is not completed within 64 clock periods (system clock), a TIMEOUT expires and reset the sequence. In this case the software shall re-enter the sequence (1, 0, 1). To reset the sequence it is sufficient to write twice a logic ‘0’ to the STOP bit of WUCTRL register (corresponding anyway to a bad sequence).

To detect if STOP Mode was entered or not, immediately after the last STOP bit setting of the sequence, poll the STOP bit itself (WUCTRL register).

8.3.3 STOP Mode Entering Conditions

Assuming the device is in Run mode, during the STOP bit setting sequence the following case may occur:

Case 1: A wake-up event on the external wake-up lines occurs during the STOP bit setting sequence

There are two possible cases:

1 Interrupt requests to the CPU are disabled: in this case the device will not enter STOP mode, no interrupt service routine will be executed and the program execution continues from the instruction following the STOP bit setting sequence. The status of STOP bit will be again:
STOP = 0
The application can determine why the device did not enter STOP mode by polling the pending bits of the external lines (at least one must be at 1).

2 Interrupt requests to CPU are enabled: in this case the device will not enter STOP mode and the interrupt service routine will be executed. The status of STOP bit will be again:
STOP = 0
The interrupt service routine can determine why the device did not enter STOP mode by polling the pending bits of external lines (at least one must be at 1).

Case 2: wrong STOP bit setting sequence

This can happen if an Interrupt request is acknowledged during the STOP bit setting sequence. In this case polling on STOP bit (WUCTRL register) will give:
STOP = 0

This means that device did not enter STOP mode due to a bad STOP bit setting sequence: the user shall retry the entire sequence.

Case 3: correct STOP bit setting sequence

In this case the device enters STOP mode.

The WKUP-INT bit can be used by an interrupt routine to detect and to distinguish events coming from Interrupt Mode or from Wake-up Mode, allowing the code to execute different procedures.

To exit STOP mode, it is sufficient that one of the 16 wake-up lines (not masked) generates an event: the clock restarts after the delay needed for the oscillator to restart.

Note: After exiting from STOP Mode, the software can successfully reset the pending bits (edge sensitive), even though the corresponding wake-up line is still active (high or low, depending on the Trigger Event register programming); the user must poll the external pin status to detect and distinguish a short event from a long one (for example a different key pushing time).

8.4 Register description

Wake-up Control Register (WUCTRL)

Address Offset: 00h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved													STOP	INT_ EN	WKUP INT
-													rw	rw	rw

Bit 15:3 = Reserved. These bits must be always written to 0.

Bit 2 = **STOP:** *Stop bit.*
To enter STOP Mode, write the sequence 1,0,1 to this bit with three write operations**.** When a correct sequence is recognized, the STOP bit is set and RCCU puts the device in STOP Mode (see Section 13: *RESET AND CLOCK CONTROL UNIT (RCCU) on page 138*). If the setting sequence isn't recognized within 64 clock periods, a TIMEOUT counter expires and reset the sequence state machine, so the device doesn't enter in STOP Mode and the STOP bit is reset by hardware. The software sequence succeeds only if the following conditions are true:

- the WKUP-INT bit is 1,
- all unmasked pending bits are reset,
- at least one mask bit is equal to 1 (at least one external wake-up line is not masked).

Otherwise the device cannot enter STOP mode, and the program continues the execution of the code: the STOP bit remains cleared.
The bit is reset by hardware if, during STOP mode, a wake-up interrupt comes from any of the unmasked wake-up lines. Otherwise the STOP bit is at 1 in the following cases (See "Wake-up Mode Selection" on page 72. for details):

- After the first write instruction of the sequence (a 1 is written to the STOP bit)
- At the end of a successful sequence (i.e. after the third write instruction of the sequence)

WARNING: If Interrupt requests are acknowledged during the sequence, the system will not enter STOP mode (since the sequence is not completed): at the end of the interrupt service routine, it is suggested to reset the sequence state machine. To reset the sequence it is necessary to write twice a logic '0' to the STOP bit of WUCTRL register (corresponding anyway to a bad sequence); on the contrary the incomplete sequence is waiting for the completion and only the TIMEOUT counter will reset the state machine. The user must re-enter the sequence to set the STOP bit.

WARNING: Whenever a STOP request is issued to the system, a few clock cycles are needed to enter STOP mode. Hence the execution of the instruction following the STOP bit setting

sequence might start before entering STOP mode (consider the ARM7 three-stage pipeline). In order to avoid to execute any valid instruction after a correct STOP bit setting sequence and before entering the STOP mode, it is mandatory to execute a dummy set of few instructions after the STOP bit setting sequence. **In particular at least six dummy instructions** (e.g. *MOV R1, R1*) shall be added after the third valid writing operation in STOP bit. Again, if exiting from STOP mode an interrupt routine shall be serviced, another set of dummy instructions shall be added, to take into account of the latency period: this is evaluated in at **least other three dummy instructions.** This to consider that when STOP mode entered, the pipeline content is frozen as well, and when the system restarts the first executed instruction was fetched and decoded before entering the STOP mode itself.

Bit 1 = **INT_EN:** *Interrupt Channel Enable.*
This bit is set and cleared by software.
0: Global mask for the 16 interrupt lines
1: The 16 external wake-up lines enabled as interrupt sources
WARNING: In order to avoid spurious interrupt requests on the Wake-up unit associated channel in the interrupt controller, due to change of interrupt source, it is suggested to reset the mask bit inside the interrupt controller before programming INT_EN bit as needed.

bit 0 = **WKUP-INT:** *Wake-up Interrupt.*
This bit is set and cleared by software.
0: The 16 external wake-up lines can be used to generate interrupt requests
1: The 16 external wake-up lines to work as wakeup sources for exiting from STOP mode

Wake-up Mask Register (WUMR)

Address Offset: 04h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WUM15	WUM14	WUM13	WUM12	WUM11	WUM10	WUM9	WUM8	WUM7	WUM6	WUM5	WUM4	WUM3	WUM2	WUM1	WUM0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:0 = **WUM[15:0]**: *Wake-Up Mask bits*
If WUMx is set, an interrupt and/or a wake-up event (depending on INT_EN and WKUP-INT bits) are generated if the corresponding WUPx pending bit is set. More precisely, if WUMx=1 and WUPx=1 then:

- If INT_EN=1 and WKUP-INT=1 then an interrupt and a wake-up event are generated.
- If INT_EN=1 and WKUP-INT=0 only an interrupt is generated.
- If INT_EN=0 and WKUP-INT=1 only a wake-up event is generated.
- If INT_EN=0 and WKUP-INT=0 neither interrupts nor wake-up events are generated.

If WUMx is reset, no wake-up events can be generated.

Wake-up Trigger Register (WUTR)

Address Offset: 08h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WUT15	WUT14	WUT13	WUT12	WUT11	WUT10	WUT9	WUT8	WUT7	WUT6	WUT5	WUT4	WUT3	WUT2	WUT1	WUT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:0 = **WUT[15:0]**: *Wake-Up Trigger Polarity bits*
The WUTx bits can be set and cleared by software to select the triggering edge.
0: The corresponding WUPx pending bit will be set upon the falling edge of the input wake-up line.
1: The corresponding WUPx pending bit will be set upon the rising edge of the input wake-up line.

WARNING

1 As the external wake-up lines are edge triggered, no glitches must be generated on these lines.

2 If either a rising or a falling edge on external wake-up lines occurs during writing of WUTR register, the pending bit will not be set.

Wake-up Pending Register (WUPR)

Address Offset: 0Ch

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WUP15	WUP14	WUP13	WUP12	WUP11	WUP10	WUP9	WUP8	WUP7	WUP6	WUP5	WUP4	WUP3	WUP2	WUP1	WUP0
rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc	rc

Bit 15:0 = **WUP[15:0]**: *Wake-Up Pending bits*
The WUPx bits are Read/Clear, they are set by hardware on occurrence of the trigger event. They can be reset by software writing a ‘1’; writing a ‘0’ is ignored.
0: No Wake-Up trigger event occurred
1: Wake-Up Trigger event occurred

8.5 Register map

Table 24. WIU Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**WUCTRL**	Reserved													STOP	INT -EN	WKUP -INT
4	**WUMR**	WUMR[15:0]															
8	**WUTR**	WUTR[15:0]															
C	**WUPR**	WUPR[15:0]															

Refer to Table 21 on page 50 for the base address.

8.6 Programming considerations

The following paragraphs give some guidelines for the design of the application program.

8.6.1 Procedure for Entering/Exiting STOP mode

1 Program the polarity of the trigger event of external wake-up lines by writing register WUTR.

2 Check that at least one mask bit (register WUMR) is equal to 1 (at least one external wake-up line is not masked).

3 Reset at least the unmasked pending bits: this allows to generate a rising edge on the Interrupt channel WUI_INT when the trigger event occurs (an interrupt is recognized when a rising edge occurs).

4 Select the interrupt source of the channel (see description of INT_EN bit in the WUCTRL register) and set the WKUP-INT bit.

Note *If clock line is chosen as wake-up event, it should be considered that as long as clock source is active the corresponding pending bit will be seen at '1'.*

Note *The change of PLL lock condition **cannot** be used as wake-up event. Selecting this input as the only wake-up source will stuck the system so as only a reset event can restore normal operations.*

5 To generate an interrupt on the associated channel, set the related enable, mask and priority bits in the interrupt controller.

6 Reset the STOP bit in register WUCTRL.

7 To enter STOP mode, write the sequence 1, 0, 1 on the STOP bit in the WUCTRL register with three write operations. To reset the sequence it is sufficient to write twice a logic '0' to the STOP bit of WUCTRL register (corresponding anyway to a bad sequence). If the sequence is not completed within 64 clock periods (system clock), a TIMEOUT expires and reset the sequence. In this case the software shall re-enter the sequence (1, 0,1).

8 The code to be executed just after the STOP sequence must check the STOP bit status to determine if the device entered STOP mode or not (See "Wake-up Mode Selection" on page 72. for details). If the device did not enter in STOP mode it is necessary to re-loop the procedure from the beginning, otherwise the procedure continues from next point (see also section 25.4 on page 368).

9 Poll the wake-up pending bits to determine which wake-up line caused the exit from STOP mode.

10 Clear the wake-up pending bit that was set.

8.6.2 Simultaneous Setting of Pending Bits

It is possible that several simultaneous wake-up event set different pending bits. In order to accept subsequent events on external wake-up/interrupt lines, once the first interrupt routine has been executed, it is necessary to clear at least one pending bit (the corresponding pending bit in WUPRx register): this operation allows to generate a rising edge on the internal line (if there is at least one more pending bit set and not masked) and so to set the interrupt controller pending bit again. A further interrupt on the same channel of the interrupt controller will be serviced depending on the status of its mask bit. Two possible situations may arise:

1 The user chooses to reset all pending bits: no further interrupt requests will be generated on channel. In this case the user has to:

- Reset the interrupt controller mask bit (to avoid generating a spurious interrupt request during the next reset operation on the WUPR register)
- Reset WUPR register.

2 The user chooses to keep at least one pending bit active: at least one additional interrupt request will be generated on the same interrupt controller channel. In this case the user has to reset the desired pending bits. This operation will generate a rising edge on the interrupt controller channel and the corresponding pending bit will be set again. An interrupt on this channel will be serviced depending on the status of corresponding mask bit.

8.6.3 Dealing with level-active signals as interrupt lines

It must be noticed that using signals generated for level sensitive interrupt lines through the WIU which supports edge-sensitive inputs only, has the consequence of requiring a special handling during the interrupt response routine in order to avoid missing some events. Actually it is better to clear the WIU pending bit as first action, so to be able to detect any subsequent edge of the level-sensitive line which could be otherwise skipped if another event occurs while still processing a previously issued one, leaving the WIU line insensitive to any further event occurring on that line. This situation can happen with external interrupt lines, for example, and it is peculiar of lock/unlock events detection. As an example the safest sequence to be followed in the case of a lock/unlock event is reported below:

1 Clear WIU pending bit corresponding to the used line (WUPR bit 6).

2 Clear EIC pending bit corresponding to WIU (IPR0 bit 2, if used).

3 Clear both RCCU pending bits, once the nature of the interrupting event has been detected and taken proper care of (LOCK_I/ULOCK_I bits of CLKFLAG).

9 DMA CONTROLLER (DMAC)

9.1 Introduction

The DMA controller provides access to 2 data streams inside STR720 system. Since peripherals are memory mapped, data transfers from/to peripherals are managed like memory/memory data transfers.

Each data stream can be associated to one particular peripheral, which triggers a DMA request starting the data transfer between the corresponding buffers defined in the stream descriptor. Data stream 3 of DMA controller can alternatively be used as a memory/memory data transfer triggered by a software DMA request, independently from any peripheral activity. In case data stream 3 is also associated to a peripheral request, the two modes can be selected by software.

The DMA controller supports circular buffer management avoiding the generation of interrupts when the controller reach the end of the source buffer. When a stream is configured in circular buffer mode and the end of buffer is reached, DMA controller reloads the start address and continues the transfer until software notifies the end of operations.

The priority between different DMA triggering sources is defined by hardware, source 2 being the highest priority request and source 3 being the lowest one.

9.2 Main Features

- 2 independently configurable streams.
- Independent source and destination word size, supporting packing and unpacking.
- 16 word deep FIFO.
- Support programmable burst size (1, 4, 8, 16 words).
- Support circular buffer management.

9.3 Functional Description

The DMA Controller is a single channel DMA Controller capable of servicing up to 2 data streams. To reduce the gate count only one FIFO has been implemented which is shared among the 2 data streams via the use of priority selection logic as outlined in Table 25 on page 82.

Table 25. DMA Request Priority

DMA Request	Priority
External2	3
External3 - Internal0/1	4 (Lowest)

DMA Request3 is multiplexed with 2 internal request lines which are selected when a memory to memory transfer is required. The DMAC block diagram is shown in Figure 10

The DMA Controller can be used for a memory to memory transfer, a peripheral to memory transfer or a memory to peripheral transfer.

A DMA data transfer consists of a sequence of DMA data burst transfers. There are two types of burst transfers, the first one is from the source address to the DMA Controller and the second one is from the DMA Controller to the destination address. Each data burst transfer is characterized by the burst length (1, 4, 8, or 16 words) and by the word width (1 byte, 2 bytes or 4 bytes). The DMA data transfer is complete when the programmed total number of bytes has been transferred from the source address to the destination address (Terminal Count register going to zero).

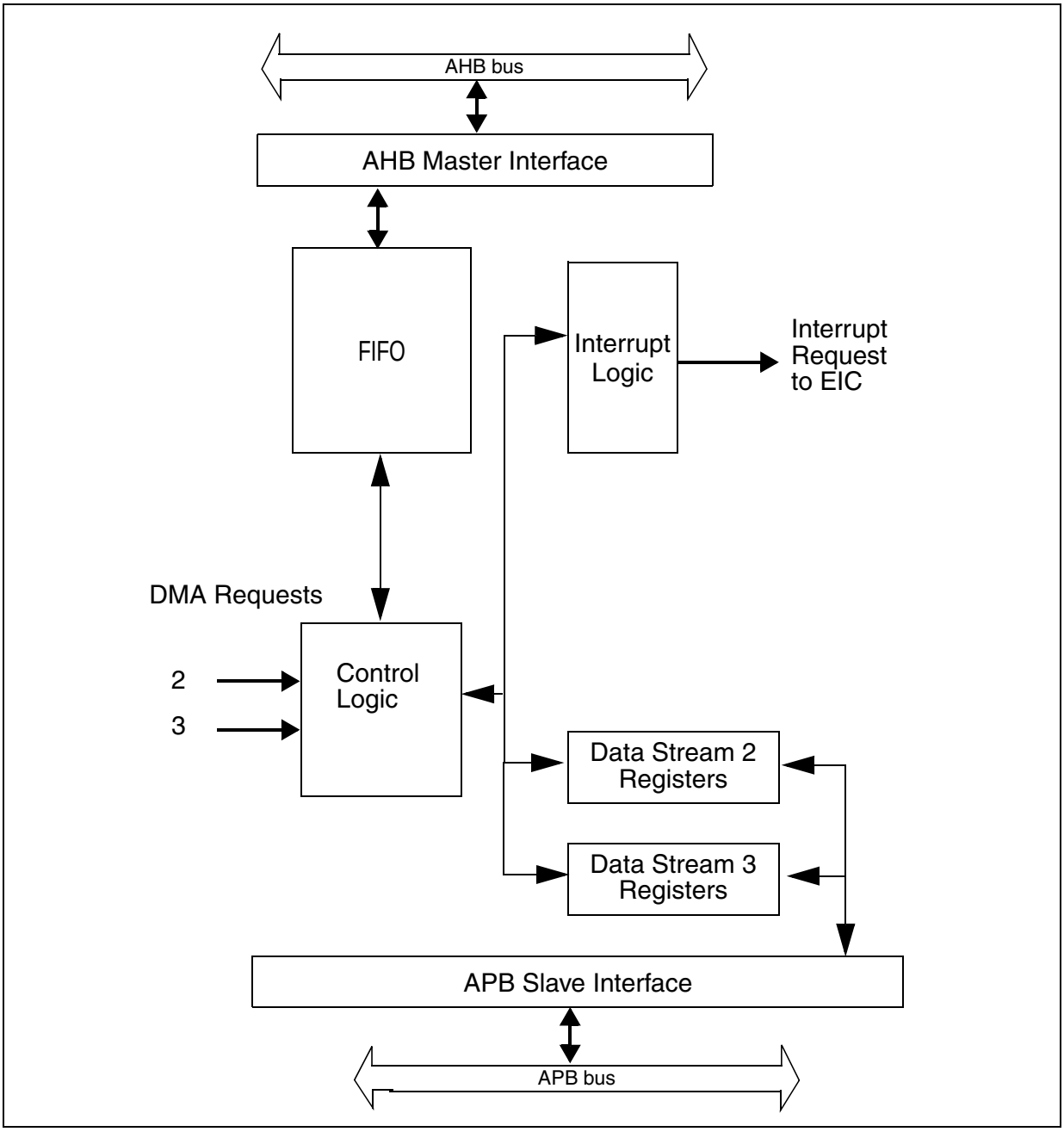
The Control Logic is used to arbiter between the data streams. It selects the request from the highest priority active data stream, passing the data from the chosen stream to and from the FIFO as required. Each stream has a set of data stream registers which are used by the Control Logic to determine source and destination addresses and the amount and format of data to be transferred. They also provide various other control and status information.

The DMA data stream registers are all 16 bit wide and are accessed via the APB Bus. The control logic can read all of these registers and can write some of them.

There is a single dedicated interrupt line from the DMA Controller to the EIC. It is driven by 4 internal interrupt flags (two per data stream) which are ored together. They work as follows:

Once a transfer for a specific stream is complete (this condition being detected by its Terminal Count register going to zero) the interrupt flag for the data stream is set. The interrupt logic then generates an interrupt to the EIC telling it that a request for one of the data streams has been completed. At the same time this data stream is disabled (i.e. the DMA Controller clears the enable bit of its control register). The status register must be read to determine which data stream caused the interrupt to be raised. The DMA Controller can now safely reconfigure this data stream (if required) for future transfers. A data stream can be re-enabled via a write to the enable bit of the corresponding Control Register.

Figure 10. DMA Controller block diagram



The DMA FIFO block consists of a 16 32-bit words deep FIFO plus Data Pack and Data Unpack units. The purpose of the FIFO block is to accommodate bus latency and burst length, and to perform any packing or unpacking operations on the data that may be necessary to accommodate different data-out/data-in width ratios.

DMA Request 3 is multiplexed with 2 internal request signals which can be used for a memory to memory data transfer. To allow their use, the ‘Mem2Mem’ bit in the control register must be

set. This tells the DMA Controller that data stream 3 is now configured for an internal, rather than an external, transfer request. An internal data transfer consists of two phases: phase 0 (where internal request 0 is asserted) is used for the memory to FIFO data transfer and phase 1 (where internal request 1 is asserted) is used for the FIFO to memory data transfer. An internal transfer will begin once the 'Mem2Mem' bit in the control register is set and data stream 3 is enabled, provided there are no pending requests for any of the other data streams, which all have higher priority. If there are pending requests, then these will be serviced first. When no other requests are pending, phase 0 will start as soon as the channel FIFO can accept the next data burst from the source address. The data packet size is defined in the data stream configuration registers. Phase 1 is asserted if the channel FIFO contains enough data for a next data burst to be sent to the destination address.

Phase 1 will be also started in order to flush the FIFO contents if any of the following conditions occur:

- If the terminal count reaches zero this indicates that the FIFO has received the total number of bytes to be transferred to the destination address. Since this number may be not a integer multiple of the selected burst size, bytes remaining in the FIFO after terminal count reaches zero must be flushed to the destination address.
- If stream 3 is disabled (via a write to the enable bit of the control register for this data stream).

9.3.1 Circular mode operations

In circular buffer mode, the DMA controller reloads the start address when the word counter (Terminal Count register) reaches the end of count and continues the transfer until application software sets the LAST bit, writing into DMALUBuff register the buffer location where the last data to be transferred is located. The stream configured in circular mode is controlled by a CIRCULAR flag in DMACtrl register ('0'=normal mode, '1'=circular mode), a LAST flag in DMALast register ('0'=infinite mode, '1'=last buffer sweep), and a register DMALUBuff (read and write).

Regarding the usage of circular buffer mode when circular buffer is the source of transfer, application software should operate in the following way:

1. Set DMA stream configuration registers writing CIRCULAR bit to'1' and LAST bit to '0'.
2. Proceed feeding the circular buffer using an index to keep trace of last buffer location used.
3. When the end of transfer condition occurs, write DMALUBuff with the value of the index and set LAST bit to '1'.
4. The DMA interrupt line will be activated as soon as DMALUBuff location has been correctly transferred.

In case circular buffer is the destination of transfer, application should proceed in a similar way:

1. Set DMA stream configuration registers writing CIRCULAR bit to'1' and LAST bit to '0'.
2. Proceed fetching the circular buffer using an index to keep trace of last buffer location used.
3. When the end of transfer condition occurs, stop DMA operation writing DMA_EN to '0'.
4. The last buffer location to be used will be indicated by DESTCurrent register pair.

Circular buffer mode has the following limitations:

- When buffer size is not a multiple of the configured burst size. This is due to the fact that FIFO is always flushed when the end of buffer is reached, and the resulting runt burst would not be followed by a transfer moving the remaining locations, located at the beginning of circular buffer, to complete the programmed burst size.
- DMALUBuff value, when used, must always be aligned to a buffer location which is a multiple of the configured burst size and its distance from the current value of TC (buffer location currently under use) must always be greater than a full burst. This is to prevent setting of last used buffer location to a place where DMA is already passed or it is about to pass during the current transfer completion, since DMA controller keeps operating in parallel to CPU program execution.
- When memory to memory data transfer is configured on stream 3.

9.4 Register description

The DMA registers are accessed via the APB bus and the register data path is 16 bit wide.

Source base address low X (DMASourceLoX) (X=2,3)

Address Offset: 80h - C0h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMASourceLoX															
rw															

DMASourceLoX contains the low base address for stream X source DMA buffer.

Bit 15:0 = **DMASourceLoX[15:0]**

Source base address high X (DMASourceHiX) (X=2,3)

Address Offset: 84h - C4h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMASourceHiX															
rw															

DMASourceHiX contains the high base address for stream X source DMA transfer.

Bit 15:0 = **DMASourceHiX[15:0]**

Destination base address low X (DMADestLoX) (X=2,3)

Address Offset: 88h - C8h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMADestLoX															
rw															

DMADestLoX contains the low base address for stream X destination DMA buffer.

Bit 15:0 = **DMADestLoX[15:0]**

Destination base address high X (DMADestHiX) (X=2,3)

Address Offset: 8Ch - CCh

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMADestHiX															
rw															

DMADestHiX contains the high base address for stream X destination DMA buffer.

Bit 15:0 = **DMADestHiX[15:0]**

Maximum count register X (DMAMaxX) (X=2,3)

Address Offset: 90h - D0h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMAMaxX															
rw															

This register is programmed with stream X maximum data unit count, defining the buffer size. The data unit is equal to the DMA data with (byte, half-word or word) configured for the source side, if Dir bit of DMACtrlX register is 0, or for the destination side, if Dir bit of DMACtrlX register is 1. Upon enabling DMA Controller, the content of the Maximum Count Register is loaded in the Terminal Count Register.

Bit 15:0 = **DMAMax0[15:0]**

Control register 2 (DMACtrl2)

Address Offset: 94h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved		Dir	reserved			Circular	DeSize		SoBurst		SoSize		DeInc	SoInc	Enable
-		rw	-			rw	rw		rw		rw		rw	rw	rw

DMACtrl2 register is used to configure stream 2 operations.

Bit 0 = **Enable:** *DMA enable*
0: DMA disable
1: DMA enable

Bit 1 = **SoInc:** *Increment Current Source Register*
This bit is used to enable the Current Source Register after each source to DMA data transfer.
0: Current Source Register unchanged
1: Current Source Register incremented

Bit 2 = **DeInc:** *Increment Current Destination Register*
This bit is used to enable the Current Destination Register after each DMA to destination data transfer.
0: Current Destination Register unchanged
1: Current Destination Register incremented

Bit 4:3 = **SoSize:** *Source to DMA data width*
These bits are used to select the data width for source to DMA data transfer.
00: 1 byte
01: 1 half-word
10: 1 word
11: reserved

Bit 6:5 = **SoBurst:** *DMA peripheral burst size*
These bits are used to define the number of words in the peripheral burst. When the peripheral is the source (Dir = 0), SoSize words are read in to the FIFO before writing FIFO contents to destination. When the peripheral is the destination (Dir = 1), the DMA interface will automatically read the correct number of source words to compile an SoBurst of the DeSize data.
00: Single
01: 4 incrementing
10: 8 incrementing
11: 16 incrementing

Bit 8:7 = **DeSize:** *DMA to destination data width*
These bits are used to select the data width for DMA to destination data transfer.
00: 1 byte
01: 1 half-word
10: 1 word
11: reserved

Bit 9 = **Circular:** *Circular mode*
This bit is used to enable DMA to operates in the circular buffer mode.
0: normal buffer mode
1: circular buffer mode

Bit 12:10 = Reserved. They must be always written to ‘0’.

Bit 13 = **Dir:** *Direction transfer*
This bit defines which field between DeSize and SoSize is used to compute burst and terminal count word unit; since the side of DMA transfer requiring a burst is usually the peripheral one this bit can be considered indicating the direction of the transfer, specifying if the peripheral is acting as source or destination.
0: SoSize defines the word unit for burst and terminal count (peripheral is the source).
1: DeSize defines the word unit for burst and terminal count (peripheral is the destination).

Bit 15:14 = Reserved. They must be always written to ‘0’.

Control register 3 (DMACtrl3)

Address Offset: D4h

Reset value: 0000h

15 14	13	12	11	10	9	8 7	6 5	4 3	2	1	0
reserved	Dir	reserved	Mem2Mem	Res.	Circular	DeSize	SoBurst	SoSize	DeInc	SoInc	Enable
-	rw	-	rw	-	rw	rw	rw	rw	rw	rw	rw

DMACtrl3 register is used to configure stream 3 operations.

Bit 0 = **Enable:** *DMA enable*
0: DMA disable
1: DMA enable

Bit 1 = **SoInc:** *Increment Current Source Register*
This bit is used to enable the Current Source Register after each source to DMA data transfer.
0: Current Source Register unchanged
1: Current Source Register incremented

Bit 2 = **DeInc:** *Increment Current Destination Register*
This bit is used to enable the Current Destination Register after each DMA to destination data transfer.
0: Current Destination Register unchanged
1: Current Destination Register incremented

Bit 4:3 = **SoSize:** *Source to DMA data width*
These bits are used to select the data width for source to DMA data transfer.
00: 1 byte
01: 1 half-word
10: 1 word
11: reserved

Bit 6:5 = **SoBurst:** *DMA peripheral burst size*
These bits are used to define the number of words in the peripheral burst. When the peripheral is the source (Dir = 0), SoSize words are read in to the FIFO before writing FIFO contents to destination. When the peripheral is the destination (Dir = 1), the DMA interface will automatically read the correct number of source words to compile an SoBurst of the DeSize data.
00: Single
01: 4 incrementing
10: 8 incrementing
11: 16 incrementing

Bit 8:7 = **DeSize:** *DMA to destination data width*
These bits are used to select the data width for DMA to destination data transfer.
00: 1 byte
01: 1 half-word
10: 1 word
11: reserved

Bit 9 = **Circular:** *Circular mode*
This bit is used to enable DMA to operates in the circular buffer mode.
0: normal buffer mode
1: circular buffer mode

Bit 10 = Reserved. It must be always written to '0'.

Bit 11 = **Mem2Mem:** *Selects memory to memory transfer*
This configure stream 3 to operate a memory to memory transfer. When Mem2Mem is set, the DMA will disregard the DMA request connected to stream 3, and transfer data from source to destination as fast as possible until MaxCnt expires.
0: Stream3 not configured for mem to mem transfer
1: Stream3 configured for mem to mem transfer
When the stream 3 is configured as a memory-memory transfer, SoBurst relates to the source side burst length, regardless of the value written in Dir bit (see below).

Bit 12 = Reserved. It must be always written to '0'.

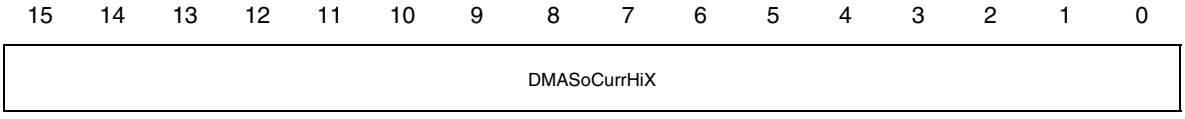
Bit 13 = **Dir:** *Direction transfer*
This bit defines which field between DeSize and SoSize is used to compute burst and terminal count word unit; since the side of DMA transfer requiring a burst is usually the peripheral one this bit can be considered indicating the direction of the transfer, specifying if the peripheral is acting as source or destination. The value written in this bit is ignored when the channel is configured to perform a memory-to-memory transfer (Mem2Mem bit written to '1').
0: SoSize defines the word unit for burst and terminal count (peripheral is the source).
1: DeSize defines the word unit for burst and terminal count (peripheral is the destination).

Bit 15:14 = Reserved. They must be always written to '0'.

Current Source address high X (DMASoCurrHiX) (X=2,3)

Address Offset: 98h - D8h

Reset value: 0000h



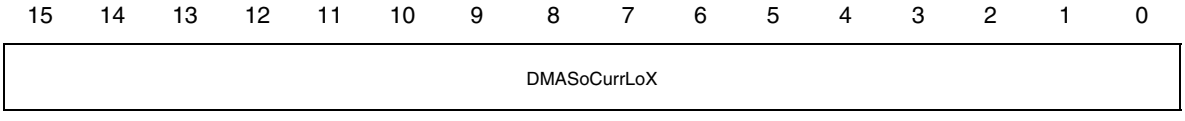
DMASoCurrHiX holds the current value of the high source address pointer related to stream X. This register is read only.

Bit 15:0 = **DMASoCurrHiX[15:0]**

Current Source address low X (DMASoCurrLoX) (X=2,3)

Address Offset: 9Ch - DCh

Reset value: 0000h



DMASoCurrLoX register holds the current value of the low source address pointer related to stream X. This register is read only.

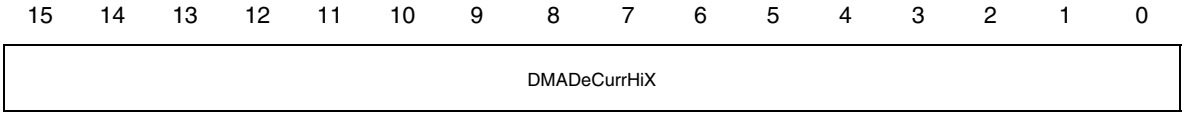
Bit 15:0 = **DMASoCurrLoX[15:0]**

The value in the registers (DMASoCurrLo and DMASoCurrHi) is used as an AHB address in a source to DMA data transfer over the AHB bus. If the SoInc bit in the Control Register is set to ‘1’, the value in the Current Source Registers will be incremented as data are transferred from a source to the DMA. The value will be incremented at the end of the address phase of the AHB bus transfer by the transferred size value. If SoInc bit is ‘0’, the Current Source Register will hold a same value during the whole DMA data transfer.

Current Destination address high X (DMADeCurrHiX) (X=2,3)

Address Offset: A0h - E0h

Reset value: 0000h



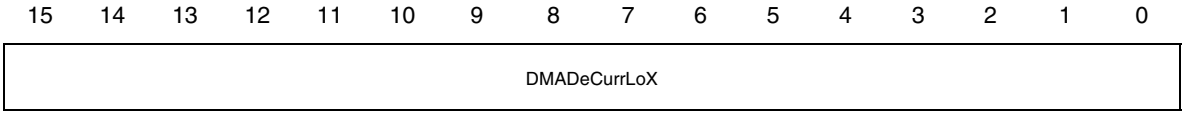
DMADeCurrHiX holds the current value of the high destination address pointer related to stream X. This register is read only.

Bit 15:0 = **DMADeCurrHiX[15:0]**

Current Destination address low X (DMADeCurrLoX) (X=2,3)

Address Offset: 24h - 64h - A4h - E4h

Reset value: 0000h



DMADeCurrLoX holds the current value of the low destination address pointer related to stream X. This register is read only.

Bit 15:0 = **DMADeCurrLoX[15:0]**

The value in the registers (DMADeCurrLo and DMADeCurrHi) is used as an AHB address in a DMA to destination data transfer over the AHB bus. If the DeInc bit in the Control Register is set to ‘1’, the value in the Current Destination Registers will be incremented as data are transferred from DMA to destination. The value will be incremented at the end of the address phase of the AHB bus transfer by the transferred size value. If DeInc bit is ‘0’, the Current Destination Register will hold a same value during the whole DMA data transfer.

Terminal Counter Register X (DMATCntX) (X=2,3)

Address Offset: A8h - E8h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMATCntX															
r															

DMATCntX contains the number of data units remaining in the current DMA transfer. The data unit is equal to the DMA data with (byte, half-word or word) configured for the source side, if Dir bit of DMACtrlX register is 0, or for the destination side, if Dir bit of DMACtrlX register is 1. The register value is decremented every time data is transferred to the DMA FIFO. When the Terminal Count reaches zero, the FIFO content is transferred to the Destination and the DMA transfer is finished. This is a read only register.

Bit 15:0 = **DMATCntX[15:0]**

Note *DMATCntX register can be used also in Circular buffer mode, with the exception of the last buffer sweep. Once LAST flag is set, the value of DMATCntX register becomes not meaningful and should be ignored.*

Last Used Buffer location X (DMALUBuffX) (X=2,3)

Address Offset: ACh - ECh

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMALUBuff															
rw															

DMALUBuffX is used in circular buffer mode during last buffer sweep, and it contains the circular buffer position where the last data to be used by stream X is located. The first buffer location is indicated writing 0x0000 into this register, the second with 0x0001 and so on, up to the last location which is indicated setting this register with (DMAMaxX - 1). For a description of this register usage, see also *Circular mode operations on page 84*

Bit 15:0 = **DMALUBuffX[15:0]**

Interrupt Mask Register (DMAMask)

Address Offset: F0h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								SEM3	SEM2	res.	res.	SIM3	SIM2	res.	res.
-								rw	rw	rw	rw	rw	rw	rw	rw

The DMA Mask Register is user to select the status flag that can generate an interrupt.

Bits 1:0 = Reserved. They must always be written to 0.

Bit 2 = **SIM2:** *Stream 2 Interrupt Mask*
This bit controls the generation of DMA interrupts triggered by stream 2 transfer end events.
1: Stream 2 transfer end interrupt is enabled
0: Stream 2 transfer end interrupt is masked

Bit 3 = **SIM3:** *Stream 3 Interrupt Mask*
This bit controls the generation of DMA interrupts triggered by stream 3 transfer end events.
1: Stream 3 transfer end interrupt is enabled
0: Stream 3 transfer end interrupt is masked

Bits 5:4 = Reserved. They must always be written to 0.

Bit 6 = **SEM2:** *Stream 2 Error Mask*
This bit controls the generation of DMA interrupts triggered by stream 2 transfer errors events.
1: Stream 2 transfer error interrupt is enabled
0: Stream 2 transfer error interrupt is masked

Bit 7 = **SEM3:** *Stream 3 Error Mask*
This bit controls the generation of DMA interrupts triggered by stream 3 transfer errors events.
1: Stream 3 transfer error interrupt is enabled
0: Stream 3 transfer error interrupt is masked

Bit 15:8 = Reserved. They must be always written to ‘0’

Interrupt Clear Register (DMAClr)

Address Offset: F4h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								SEC3	SEC2	res.	res.	SIC3	SIC2	rese.	res.
-								w	w	w	w	w	w	w	w

The DMA Clear Register is used to clear the status flags. This is a write-only register and it will return 0000h every time it is read.

Bits 1:0 = Reserved. They must always be written to 0.

Bit 2 = **SIC2:** *Stream 2 Interrupt Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 2 transfer end event.
1: Clear INT2 flag in DMAStatus register
0: No effect

Bit 3 = **SIC3:** *Stream 3 Interrupt Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 3 transfer end event.
1: Clear INT3 flag in DMAStatus register
0: No effect

Bits 5:4 = Reserved. They must always be written to 0.

Bit 6 = **SEC2:** *Stream 2 Error Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 2 transfer error event.
1: Clear ERR2 flag in DMAStatus register
0: No effect

Bit 7 = **SEC3:** *Stream 3 Error Clear*
This bit allows clearing the pending interrupt flag corresponding to stream 3 transfer error event.
1: Clear ERR3 flag in DMAStatus register
0: No effect

Bit 15:8 = Reserved. They must be always written to ‘0’

Interrupt Status Register (DMAStatus)

Address Offset: F8h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				ACT3	ACT2	res.	res.d	ERR3	ERR2	res.	res.	INT3	INT2	res.	res.
-				r	r	r	r	r	r	r	r	r	r	r	r

DMAStatus provides status information regarding the DMA Controller. This is a read-only register.

Bits 1:0 = Reserved. They must always be written to 0.

Bit 2 = **INT2:** *Data stream 2 interrupt flag*
When a transfer end event occurs on stream 2, this bit will be set to ‘1’ and if SIM2 bit of DMAMask has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in SIC2 bit of DMAClear register.

Bit 3 = **INT3:** *Data stream 3 interrupt flag*
When a transfer end event occurs on stream 3, this bit will be set to ‘1’ and if SIM3 bit of DMAMask has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in SIC3 bit of DMAClear register.

Bits 5:4 = Reserved. They must always be written to 0.

Bit 6 = **ERR2:** *Data stream 2 error flag*
When a transfer error event occurs on stream 2, this bit will be set to ‘1’ and if SEM2 bit of DMAMask has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in SEC2 bit of DMAClear register.

Bit 7 = **ERR3:** *Data stream 3 error flag*
When a transfer error event occurs on stream 3, this bit will be set to ‘1’ and if SEM3 bit of DMAMask has also set to ‘1’ by software then a DMA interrupt request will be generated. This flag is cleared by writing ‘1’ in SEC3 bit of DMAClear register.

Bits 9:8 = Reserved. They must always be written to 0.

Bit 10 = **ACT2:** *Data stream 2 status*
1: Data stream 2 is active.
0: Data stream 2 is not active

Bit 11 = **ACT3:** *Data stream 3 status*
1: Data stream 3 is active.
0: Data stream 3 is not active

Bit 15:12 = Reserved. They must be always written to ‘0’

Last Flag Register (DMALast)

Address Offset: FCh

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												LAST3	LAST2	Res.	Res.
-												rw	rw	rw	rw

DMALast controls the activation of last buffer sweep mode for the streams configured in circular buffer mode. See also *Circular mode operations on page 84*.

Bits 1:0 = Reserved. They must always be written to 0.

Bit 2 = **LAST2:** *LAST buffer sweep stream 2*
This bit is used to notify DMA that last circular buffer sweep started. If this bit is set while stream 2 is configured in circular mode, the corresponding data stream interrupt flag will be set when DMA uses the circular buffer location contained in DMALUBuff2 register.
0: Continuous circular buffer mode
1: Last circular buffer sweep started.

Bit 3 = **LAST3:** *LAST buffer sweep stream 3*
This bit is used to notify DMA that last circular buffer sweep started. If this bit is set while stream 3 is configured in circular mode, the corresponding data stream interrupt flag will be set when DMA uses the circular buffer location contained in DMALUBuff3 register.
0: Continuous circular buffer mode
1: Last circular buffer sweep started.

Bit 15:4 = Reserved. They must be always written to ‘0’

9.4.1 Register map

The following table summarizes the registers implemented in the DMA macrocell

Table 26. DMA Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
80	**DMA SourceLo2**	DMASoLo2															
84	**DMA SourceHi2**	DMASoHi2															
88	**DMA DestLo2**	DMADeLo2															
8C	**DMA DestHi2**	DMADeHi2															
90	**DMAMax2**	DMAMax2															
94	**DMACtrl2**	reserved		Dir	reserved			Circular	DeSize		SoBurst		SoSize		DeInc	SoInc	Enable
98	**DMA SoCurrHi2**	DMASoCurrHi2															
9C	**DMA SoCurrLo2**	DMASoCurrLo2															
A0	**DMA DeCurrHi2**	DMADeCurrHi2															
A4	**DMA DeCurrLo2**	DMADeCurrLo2															
A8	**DMATCnt2**	DMATCnt2															
AC	**DMA LUBuff2**	DMALUBuff2															
B0	**-**	Reserved															
C0	**DMA SourceLo3**	DMASoLo3															
C4	**DMA SourceHi3**	DMASoHi3															
C8	**DMA DestLo3**	DMADeLo3															
CC	**DMA DestHi3**	DMADeHi3															
D0	**DMAMax3**	DMAMax3															
D4	**DMACtrl3**	reserved		Dir	res.	Mem2 Mem	res	Circular	DeSize		SoBurst		SoSize		DeInc	SoInc	Enable
D8	**DMA SoCurrHi3**	DMASoCurrHi3															
DC	**DMA SoCurrLo3**	DMASoCurrLo3															

Table 26. DMA Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E0	**DMA DeCurrHi3**	DMADeCurrHi3															
E4	**DMA DeCurrLo3**	DMADeCurrLo3															
E8	**DMATCnt3**	DMATCnt3															
EC	**DMA LUBuff3**	DMALUBuff3															
F0	**DMAMask**	reserved								SEM3	SEM2	res.	res.	SIM3	SIM2	res.	res.
F4	**DMAClr**	reserved								SEC3	SEC2	res.	res.	SIC3	SIC2	res.	res.
F8	**DMA Status**	reserved				Act3	Act2	res.	res.	Err3	Err2	res.	res.	Int3	Int2	res.	res.
FC	**DMA Last**	reserved												LAST3	LAST2	res.	res.

Refer to Table 21 on page 50 for the base address.

10 DRAM CONTROLLER (DRAMC)

10.1 Introduction

The DRAM Controller block is an AHB slave used to provide an interface between the system bus and external memory devices. The memory clock frequency is the same as the system bus clock frequency.

The controller supports four external banks, containing either SDRAM or EDO memories; all banks must be of the same memory type. Each bank can be independently configured and enabled/disabled by means of internal configuration registers.

All the internal registers are accessed via an Advanced Peripheral Bus interface (APB) to the DRAM Controller.

A refresh timer is provided to issue refresh commands to the memory; such refresh counter uses the CLK input signal prescaled by 16 in order to trigger the refresh events.

10.2 Main Features

- 8, 16 or 32 bits for the memory data bus width.
- 4 external memory banks.
- It is possible to use less than four banks, but banks must be contiguous.
- Each bank can span from 64 kBytes up to 32 MBytes in 64 kBytes increments.
- 128 MBytes of total addressable external memory space, when all banks have the maximum size.
- Little endian operations supported.

10.3 Functional Description

10.3.1 AHB Interface

The DRAM Controller is a configurable peripheral devoted to manage an external memory component. The AHB Interface decomposes the system bus transfers into memory accesses supported by the selected memory bank and performs the data transfers to/from the external chip. At the end of each access it will execute DRAM precharge, i.e. the SDRAM banks are not kept open.

It supports up to 4 memory banks, where a bank is one single memory chip or more chips that share the same addresses and chip select. If a disabled memory bank is accessed, the Controller will return the ERROR response, otherwise it will return the OKAY response.

External memory transfers are configurable to be 8,16 or 32-bit wide.

Memory Address remapping

The AHB Interface remaps AHB address into DRAM row and column addresses; the 8, 9 and 10 bits wide column addresses are supported.

Table 27 shows how AHB address bits are mapped to MIA bits as row or column address. Note that in the “Column” line bit 10 is always zero, while **ba** means: the same bit value used in “Row” line. (This because bits [15:10] are never considered when used as column address).

Table 27. DRAM Address bits versus AHB address bits

Memory Data Width	Address	Column Address Width	Memory Interface Address bits															
			15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
8 bits	Column		**ba**	**ba**	**ba**	**ba**	**ba**	**0**	9	8	7	6	5	4	3	2	1	0
	Row	9	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9
		10	x	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
16 bits	Column		**ba**	**ba**	**ba**	**ba**	**ba**	**0**	10	9	8	7	6	5	4	3	2	1
	Row	8	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9
		9	x	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
		10	x	x	24	23	22	21	20	19	18	17	16	15	14	13	12	11
32 bits	Column		**ba**	**ba**	**ba**	**ba**	**ba**	**0**	11	10	9	8	7	6	5	4	3	2
	Row	8	x	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
		9	x	x	24	23	22	21	20	19	18	17	16	15	14	13	12	11
		10	x	x	x	24	23	22	21	20	19	18	17	16	15	14	13	12

The DRAM Controller reads the 27 less significant bits of the AHB address bus; they are decomposed into:

- Byte lanes MIBLSout[3:0], that are used to select which byte to read/write if the DRAM data bus is larger than the data to read/write (for instance, when writing one byte into a DRAM with a 32 bits data bus). They are decoded from HADDR[1:0] taking into account AHB data width and DRAM data width.
- Column address, as specified in Table 27, line “Column”. 8, 9 or 10 bits are really used by the memory.
- Row address, the remaining AHB address bits excluded the last two ones, HADDR[26:25].
- Chip selects MICSout[3:0], to select one of four external bank to use. This is determined by the value of HADDR[26:16] and the size of each external bank.

10.3.2 APB Interface

The APB Interface contains the DRAM Controller registers and is used only to access these registers.

The length of time required to properly transfer data from/to external memory is controlled by an internal Bank Configuration Register which is dedicated to each bank. The data fields of this register determine the number of memory clock cycles used to access that bank; the access time also depends on the bus width and the latency of the selected device. A flag bit in the Memory Configuration Register (one register for all banks) informs the controller if a particular bank is accessible or not.

To allow the STR720 to directly configure the SDRAM devices, the SDRAM Configuration Registers are provided. When they are written via the system bus, the DRAM Controller will drive memory bus output signals with the values from the SDRAM Configuration Registers. This means that the content of the selected register will be outputted on the memory bus and the corresponding memory bank chip select will be asserted, for one clock period.

10.3.3 Refresh Timer

A refresh timer is provided to issue refresh commands to SDRAM and EDO devices. Refresh commands are issued to all banks simultaneously.

When a refresh is requested, the DRAMC waits for any active memory transfer to complete before activating the refresh.
Any system bus memory transfers started while the refresh is in progress are stalled by the Controller.

The refresh counter uses an enable signal, synchronous with the rising edge of the system clock, which is generated by prescaling CLK input signal by 16.
This signal is used to ensure that the refresh counter operates at the same frequency regardless of the current system clock configuration depending on PLL and RCCU settings.

If the PWRSAVE bit in the Memory Configuration Register is set to ‘1’ , the refresh logic will set external memories in the power save self-refresh mode (as long as the bit remain high).

10.4 Register description

The following paragraphs give detailed descriptions and operations of each of the bits in the DRAMC registers.

Bank 1 Configuration Register (MB1Config)

Address Offset: 00h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				DEVWID[1:0]		DATALAT[1:0]		SETUPTIME[2:0]			IDLETIME[2:0]			SDRAMCOL[1:0]	
				rw		rw		rw			rw			rw	

The Bank 1 Configuration Register (MB1Config) is a 16-bit read/write control register used to configure specific parameters of external region 1 (Bank 1).
The MB1Config control bits are described below.

Bit 15:12 reserved (should be written as zero and ignored on read).

Bit 11:10 **DEVWID[1:0]**: *Device Width*.
This field defines the data width of the external memory device 0.
= 0b00. Byte (8 bit)
= 0b01. Half Word (16 bit)
= 0b10. Word (32 bit)
= 0b11. Reserved.

Bit 9:8 **DATALAT[1:0]**: *Data Latency*.
This field defines the number of memory clock cycles between the start of a memory read access and the first valid data. This parameter is usually indicated in SDRAM datasheets as CAS latency. The DATALAT value is valid between 0 and 3.

Bit 7:5 **SETUPTIME[2:0]**: *Setup Time.*
This field defines the number of memory clock cycles the memory drivers spends in a wait state before accessing the external memory. This parameter is usually indicated in SDRAM datasheets as RAS-CAS delay. The SETUPTIME value is valid between 0 and 7.

Note SETUPTIME field should be set using the RAS-CAS delay value found on SDRAM datasheet minus 1. As an example to get a RAS-CAS delay of 2 clock cycles, SETUPTIME should be configured to 1.

Bit 4:2 **IDLETIME[2:0]**: *Idle Time*.
This field defines the minimum time the memory driver must spend in the IDLE state following memory accesses (the value defines the number of memory clock cycles). This parameter is usually indicated in SDRAM datasheets as precharge delay. The IDLETIME value is valid between 0 and 7.

Bit 1:0 **SDRAMCOL[1:0]**: *SDRAM Column Width*
This field specifies the width of the SDRAM column address .
= 0b00. 8 bits
= 0b01. 9 bits
= 0b10. 10 bits
= 0b11. Reserved.

Bank 2 Configuration Register (MB2Config)

Address Offset: 04h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				DEVWID[1:0]		DATALAT[1:0]		SETUPTIME[2:0]			IDLETIME[2:0]			SDRAMCOL[1:0]	
				rw		rw		rw			rw			rw	

The Bank 2 Configuration Register (MB2Config) is a 16-bit read/write control register used to configure specific parameters of external region 2 (Bank 2).
The MB2Config control bits are the same as MB1Config register.

Bank 3 Configuration Register (MB3Config)

Address Offset: 08h

Reset value: 0000h

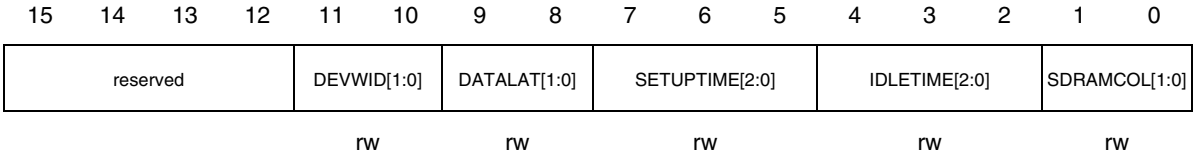
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				DEVWID[1:0]		DATALAT[1:0]		SETUPTIME[2:0]			IDLETIME[2:0]			SDRAMCOL[1:0]	
				rw		rw		rw			rw			rw	

The Bank 3 Configuration Register (MB3Config) is a 16-bit read/write control register used to configure specific parameters of external region 3 (Bank 3).
The MB3Config control bits are the same as MB1Config register.

Bank 4 Configuration Register (MB4Config)

Address Offset: 0Ch

Reset value: 0000h

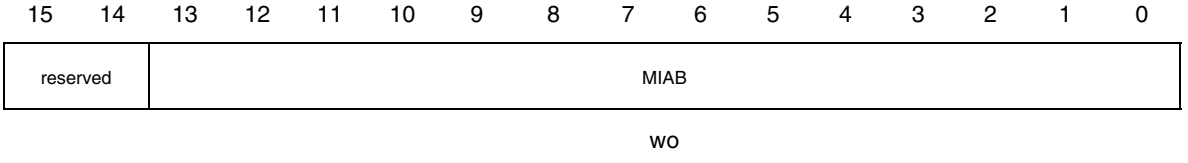


The Bank 4 Configuration Register (MB4Config) is a 16-bit read/write control register used to configure specific parameters of external region 4 (Bank 4).
The MB4Config control bits are the same as MB1Config register.

Bank 1 SDRAM Configuration Register Low (SDRAM1ConfigLo)

Address Offset: 10h

Reset value: 0000h



The Bank 1 SDRAM Configuration Register Low (SDRAM1ConfigLo) is a 16-bit write only control register used to configure the SDRAM device of external region 1 (Bank 1).
The SDRAM1ConfigLo control bits are described below.

Bit 15:14 reserved (should be written as zero).

Bit 13:0 **MIAB[13:0]**: *Memory Interface Address Bus.*
This field defines the data to be written in the SDRAM internal configuration register by directly driving the SDRAM address bus lines. They will be asserted on the SDRAM address bus with the value written in SDRAM1ConfigLo register for one clock period.

Bank 1 SDRAM Configuration Register High (SDRAM1ConfigHi)

Address Offset: 14h

Reset value: 0000h

The Bank 1 SDRAM Configuration Register High (SDRAM1ConfigHi) is a 16-bit write only control register used to configure the SDRAM device of external region 1 (Bank 1). The SDRAM1ConfigHi control bits are described below.

Bit 15:3 reserved (should be written as zero).

Bit 2 **MIWE**: *Memory Interface Write Enable.*
This bit allows to control directly the value on the SDRAM write enable line. When it is written to '1' the corresponding memory pin will be asserted to '0' for one clock period. This is a write-ony bit.

Bit 1 **MIAA**: *Memory Interface Access Active (nCAS).*
This bit allows to control directly the value on the SDRAM column address strobe line. When it is written to '1' the corresponding memory pin will be asserted to '0' for one clock period. This is a write-ony bit.

Bit 0 **MISA**: *Memory Interface Setup Active (nRAS).*
This bit allows to control directly the value on the SDRAM row address strobe line. When it is written to '1' the corresponding memory pin will be asserted to '0' for one clock period. This is a write-ony bit.

Bank 2 SDRAM Configuration Register Low (SDRAM2ConfigLo)

Address Offset: 18h

Reset value: 0000h

The Bank 2 SDRAM Configuration Register Low (SDRAM2ConfigLo) is a 16-bit write only control register used to configure the SDRAM device of external region 2 (Bank 2).
The SDRAM2ConfigLo control bits are the same as SDRAM1ConfigLo register.

Bank 2 SDRAM Configuration Register High (SDRAM2ConfigHi)

Address Offset: 1Ch

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved													MIWE	MIAA	MISA
													wo	wo	wo

The Bank 2 SDRAM Configuration Register High (SDRAM2ConfigHi) is a 16-bit write only control register used to configure the SDRAM device of external region 2 (Bank 2).
The SDRAM2ConfigHi control bits are the same as SDRAM1ConfigHi register.

Bank 3 SDRAM Configuration Register Low (SDRAM3ConfigLo)

Address Offset: 20h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved		MIAB													
		wo													

The Bank 3 SDRAM Configuration Register Low (SDRAM3ConfigLo) is a 16-bit write only control register used to configure the SDRAM device of external region 3 (Bank 3).
The SDRAM3ConfigLo control bits are the same as SDRAM1ConfigLo register.

Bank 3 SDRAM Configuration Register High (SDRAM3ConfigHi)

Address Offset: 24h

Reset value: 0000h

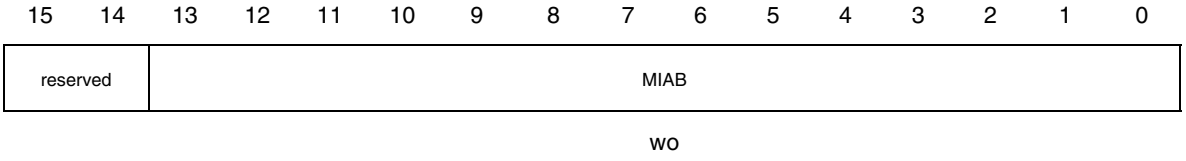
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved													MIWE	MIAA	MISA
													wo	wo	wo

The Bank 3 SDRAM Configuration Register High (SDRAM3ConfigHi) is a 16-bit write only control register used to configure the SDRAM device of external region 3 (Bank 3).
The SDRAM3ConfigHi control bits are the same as SDRAM1ConfigHi register.

Bank 4 SDRAM Configuration Register Low (SDRAM4ConfigLo)

Address Offset: 28h

Reset value: 0000h



The Bank 4 SDRAM Configuration Register Low (SDRAM4ConfigLo) is a 16-bit write only control register used to configure the SDRAM device of external region 4 (Bank 4).
The SDRAM4ConfigLo control bits are the same as SDRAM1ConfigLo register.

Bank 4 SDRAM Configuration Register High (SDRAM4ConfigHi)

Address Offset: 2Ch

Reset value: 0000h

The Bank 4 SDRAM Configuration Register High (SDRAM4ConfigHi) is a 16-bit write only control register used to configure the SDRAM device of external region 4 (Bank 4).
The SDRAM4ConfigHi control bits are the same as SDRAM1ConfigHi register.

Memory Configuration Register (MemConfig)

Address Offset: 30h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved		PWRSAVE	TYPE	B4EN	B3EN	B2EN	B1EN	REFR[7:0]							
		rw	rw	rw	rw	rw	rw	rw							

The Memory Configuration Register (MemConfig) is a 16-bit read/write control register used to configure parameters that are the same for all banks.
The MemConfig control bits are described below.

Bit 15:14 reserved (should be written as zero and ignored on read).

Bit 13 **PWRSAVE**: *Power Save Mode*.
This bit selects whether to enter the power save self-refresh mode.
= 1. The next refresh cycle will set the memory devices in the self-refresh mode.
= 0. Next refresh cycle the memories will exit the self-refresh mode.

Bit 12 **TYPE**: *Memory Type*.
This bit selects the type of the external memory.
= 1. SDRAM.
= 0. EDO.

Bit 11 **B4EN**: *Bank 4 Enable.*
This bit enables Bank 4 of memory.
= 1. Bank enabled.
= 0. Bank disabled.

Bit 10 **B3EN**: *Bank 3 Enable.*
This bit enables Bank 3 of memory.
= 1. Bank enabled.
= 0. Bank disabled.

Bit 9 **B2EN**: *Bank 2 Enable.*
This bit enables Bank 2 of memory.
= 1. Bank enabled.
= 0. Bank disabled.

Bit 8 **B1EN**: *Bank 1 Enable.*
This bit enables Bank 1 of memory.
= 1. Bank enabled.
= 0. Bank disabled.

Bit 7:0 **REFR[7:0]**: *Refresh Period*.
This field is used to determine the refresh period. The refresh period can be set in

steps of 16 times the CLK period (T_{CLK}). The REFR value is valid between 0 and 255. For an example of the correspondence between REFR setting and refresh period see Table 28 Refresh Period..

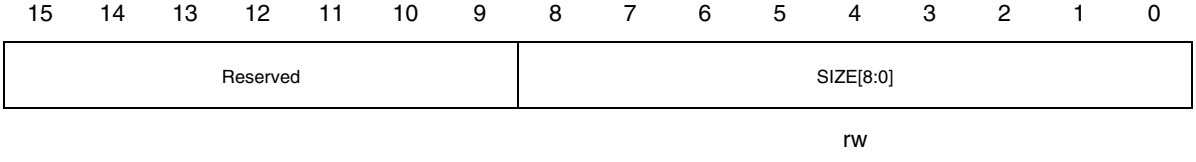
Table 28. Refresh Period

REFR	Refresh Period	Refresh period with f_{CLK} = 48 MHz
00000000	Refresh is disabled	Refresh is disabled
00000001	16 * T_{CLK}	333.33 ns
00000010	32 * T_{CLK}	666.66 ns
00000011	48 * T_{CLK}	1 µs
...		
11111111	4080 * T_{CLK}	85 µs

Size of bank 1 Register (Bank1Size)

Address Offset: 34h

Reset value: 0000h



The Size of bank 1 Register (Bank1Size) is a 16-bit read/write control register used to specify the size of external region 1 (Bank 1).
The Bank1Size control bits are described below.

Bit 15:9 Reserved (should be written as zero and ignored on read).

Bit 8:0 **SIZE[8:0]**: *Bank Size*.
This field defines the size of the external memory device 1 in 64 kBytes steps.The SIZE value is valid between 0 and 511, where (SIZE+1) represents the size of the bank in 64Kbytes steps. See also Table 29 .

Table 29. Size field and the corresponding actual size

SIZE field	Bank actual size
0.0000.0000	1 x 64kBytes = 64kBytes
0.0000.0001	2 x 64kBytes = 128kBytes
0.0000.0010	3 x 64kBytes = 192kBytes
...	
0.0000.1111	16 x 64kBytes = 1MBytes

Table 29. Size field and the corresponding actual size

SIZE field	Bank actual size
...	
0.0001.1111	32 x 64kBytes = 2MBytes
...	
0.0011.1111	64 x 64kBytes = 4MBytes
...	
0.0111.1111	128 x 64kBytes = 8MBytes
...	
0.1111.1111	256 x 64kBytes = 16MBytes
...	
1.1111.1111	512 x 64kBytes = 32MBytes

Size of bank 2 Register (Bank2Size)

Address Offset: 38h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved							SIZE[8:0]								
							rw								

The Size of bank 2 Register (Bank2Size) is a 16-bit read/write control register used to specify the size of external region 2 (Bank 2).
The Bank2Size control bits are the same as Bank1Size register.

Size of bank 3 Register (Bank3Size)

Address Offset: 3Ch

Reset value: 0000h

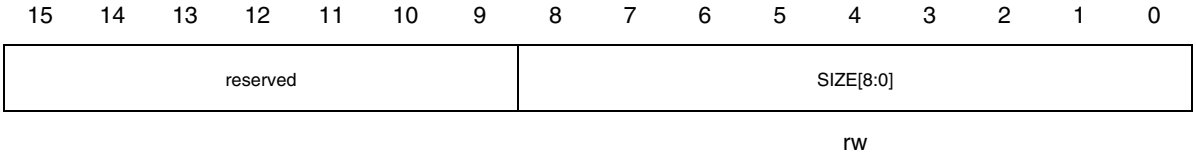
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved							SIZE[8:0]								
							rw								

The Size of bank 3 Register (Bank3Size) is a 16-bit read/write control register used to specify the size of external region 3 (Bank 3).
The Bank3Size control bits are the same as Bank1Size register.

Size of bank 4 Register (Bank4Size)

Address Offset: 40h

Reset value: 0000h



The Size of bank 4 Register (Bank4Size) is a 16-bit read/write control register used to specify the size of external region 4 (Bank 4).
The Bank4Size control bits are the same as Bank1Size register.

10.4.1 Register map

In the Table 30 an overview of the DRAMC registers is reported.

Table 30. DRAMC Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**MB1Config**	reserved				DEVWID[1:0]		DATALAT[1:0]		SETUPTIME[2:0]			IDLETIME[2:0]			SDRAM-COL[1:0]	
4	**MB2Config**	reserved				DEVWID[1:0]		DATALAT[1:0]		SETUPTIME[2:0]			IDLETIME[2:0]			SDRAM-COL[1:0]	
8	**MB3Config**	reserved				DEVWID[1:0]		DATALAT[1:0]		SETUPTIME[2:0]			IDLETIME[2:0]			SDRAM-COL[1:0]	
C	**MB4Config**	reserved				DEVWID[1:0]		DATALAT[1:0]		SETUPTIME[2:0]			IDLETIME[2:0]			SDRAM-COL[1:0]	
10	**SDRAM1ConfigLo**	reserved		MIAB													
14	**SDRAM1ConfigHi**	reserved													MIWE	MIAA	MISA
18	**SDRAM2ConfigLo**	reserved		MIAB													
1C	**SDRAM2ConfigHi**	reserved													MIWE	MIAA	MISA
20	**SDRAM3ConfigLo**	reserved		MIAB													
24	**SDRAM3ConfigHi**	reserved													MIWE	MIAA	MISA
28	**SDRAMC4onfigLo**	reserved		MIAB													
2C	**SDRAM4ConfigHi**	reserved													MIWE	MIAA	MISA
30	**MemConfig**	reserved		PWR-SAVE	TYPE	B3EN	B2EN	B1EN	B0EN	REFR[7:0]							
34	**Bank1Size**	reserved								SIZE[8:0}							
38	**Bank2Size**	reserved								SIZE[8:0}							
3C	**Bank3Size**	reserved								SIZE[8:0}							
40	**Bank4Size**	reserved								SIZE[8:0}							

Refer to Table 21 on page 50 for the base address.

10.5 Programming considerations

DRAMC programming and usage is accomplished via the internal registers described in previous paragraphs.

At reset time all the Controller registers will be reset to 0. It is up to the user the correct programming of the four Bank Size registers and Bank enable bits in the MemConfig register, in order to properly describe the physical devices connected to the DRAM Controller.
Note that banks are contiguous, even if it is possible to use less than four banks: if an AHB transfer is accessing a disabled bank, the DRAM Controller will return the error response to the AHB master (ARM720T CPU or DMA controller).

After power-up the CPU must configure each SDRAM device, i.e. perform the precharge-refresh-mode register set procedure as specified in the SDRAM device data sheet. This is accomplished with the correct values in the 8 SDRAMxConfigLo,Hi registers. A write access to the high registers will start the SDRAM configuration cycle, during which the value written to the register will be asserted on the memory bus for one clock period.

When ARM720T caches are enabled, all accesses to SDRAM memory are performed as bursts of 4 transfers and in order to achieve the highest performance level, SDRAM interface must be aware of this particular access mode. Unfortunately a limitation in ARM720T core and SDRAM interface requires this specific configuration to be enforced explicitly by software. This configuration selection is performed by using CACHE_CONFIG bit of SGCR1 register, which can configure STR720 to work in “burst-access” mode and it should be used whenever cache is enabled, so to have optimal performance from the system.

In “burst-access” mode all transfers issued by ARM720T core are identified as 16-byte fixed length burst, possibly early terminated when the actual burst length is shorter than 16, as it normally happens on cache line refills. In this way any sequence of transfers generated by a cached ARM720T core fetching its code from a cached region can be legally completed by SDRAM interface, including multiple register load/store sequences.

When ARM720T is used with its cache disabled or it is fetching its code from a non-cacheable SDRAM region, this bit should remain cleared, so to avoid incurring in a system hang, when long sequences of internal instructions generates a sequence of consecutive accesses longer than a 16-byte burst. In case system hangs for this particular configuration mistake, application program execution can be restored only with a system reset.

11 EXTERNAL MEMORY INTERFACE (EMI)

11.1 Introduction

The External Memory Interface is a configurable interface designed to control the data flow from an internal Advanced High-performance Bus (AHB) to external memory components as ROM and SRAM.

The total external memory space of 8 MByte can be subdivided in 2 regions, corresponding to the 2 available chip select signals (CS0..CS1). External memory transfers are always 16 bit wide, the width of the transfers from the AHB bus to the EMI peripheral, in accordance with to the AHB specifications, are controlled by the HSIZE signal (8,16 or 32-bit). Note that the external address bus always refers to 16-bit location addresses, thus bit position 0 on this bus corresponds to bit 1 on the internal AHB address bus.

The length of time required to properly transfer data from/to external memory region is controlled by programming of internal registers dedicated to each external memory space which determines the number of wait states used to access that region. Read/Write cycle termination can be also configured to be controlled by the “External Ready” signal (Eready). Each internal register also contains a flag bit which informs the controller if a particular memory space is accessible or not.

All the internal configuration registers are accessed via an Advanced Peripheral Bus interface (APB) to the EMI block.

11.2 Main Features

- 8 MByte of total addressable external memory space.
- Up to 2 external regions/chip selects (Banks).
- Bank 0 can span from 64 kBytes up to 8 MBytes in 64 kBytes increments.
- Bank 1 can span from 64 kBytes up to 8 MBytes in 64 kBytes increments.
- Up to 7 configurable wait states for each external memory region.
- Read/Write cycle termination via “External Ready” (Eready) signal (configurable).

Note Only chip select 0 signal is always available on STR720 device, CS1 being shared with different alternate functions or general purpose I/Os.

11.3 Functional Description

The External Memory Interface is a configurable peripheral aimed to ease the connection of external memory components. EMI memory map is detailed in Table 31. Each memory address region (Bank), with the exception of Bank 0, has a configurable base address (with a granularity of 2k in the 8M total address space) and a configurable size. For bank 1 base address is selected via a dedicated 16-bit register (BBASE1) that defines the upper 16 bits of the range over which the relevant chip select will be active.

Table 31. EMI Address Map

Bank Start Address	Description	Size	Bus Width
0b_0	Bank 0 (CS0)	64k - 8M	16
0b_BBASE1[11:0] | 000_0000_0000	Bank 1 (CS1)	64k - 8M	16

For bank 1 the user has the freedom of independently choosing bank start address and bank size. Since the physical address going to the external memory (EAdd[22:1]) is simply obtained by dropping bits from 31 to 23 of the AHB address, it may happen that the lowest address of the bank does not map to the lowest address of the external memory. This does not mean that the one-to-one correspondence between an address and a physical location in the memory is lost: it means only that lowest address of the bank (as defined by bbase) will map somewhere in the middle of the memory and that, after reaching the top of the physical memory, the address will wrap around to zero.

In the special case in which the bank start address is programmed to be a multiple of the bank length, the usual result of having the lowest address of the bank mapped to the lowest address of the external memory is obtained. Therefore, to get this usual mapping, a 64 kbyte memory should be only mapped at 64k boundaries, a 128 kbyte memory should be only mapped at 128k boundaries and so on. It is up to the user the consistent programming of the “base” and “length” registers.

11.3.1 EMI Programmable Timings

Each memory bank of the EMI can have a programmable number of wait states (up to 7) added to any read or write cycle: this is accomplished via the C_LENGTH bitfield of the BCONx register (x=0,..,1).

This C_SETUP programing will translate in different cycle lengths, depending on the Read or Write type of operation and depending on the bank on which the operation is performed.

For write cycles which translate in a single external bus operation (e.g. a 16-bit write), the total length (measured in EMI internal clock units) that a single access will require (equal to the length over which the Chip Select will be active) will depend on the memory bank the cycle is executed on. It can be calculated through the formula:

	Setup	Write Enable Time	Hold
CS Length (Write, Bank 0/1) =	0	+ (C_LENGTH+1)	+ 1

This means that for memory banks 0 and 1 setup and hold times (measured in EMI internal clock units) are fixed.

Table 32 resumes the write cycle timing configurations.

Table 32. EMI Write Timings

Parameter	Field	Bank 0,1
Setup Time	n.a.	0
WE Time	C_LENGTH	“000”:”111”=1:8 AHB_CLK
Hold Time	n.a.	1
	CS Length	2:9 AHB_CLK

For read cycles which translate in a single external bus operation (e.g. a 16-bit read), the total length (measured in EMI internal clock units) that a single access will require (equal to the length over which the Chip Select will be active) will also depend on the memory bank the cycle is executed on. Its total length can be calculated through the formula:

	Setup	Read Enable Time	Hold
CS Length (Read, Bank 0/1) =	0	+ (C_LENGTH+2)	+ 0

This means that for memory banks 0 and 1 Setup and Hold times (measured in EMI internal clock units) are fixed to zero.

Table 33 resumes the read cycle timing configurations.

Table 33. EMI Read Timings

Parameter	Field	Bank 0,1
Setup Time	n.a.	0
RE Time	C_LENGTH	“000”:”111”=2:9 AHB_CLK
Hold Time	n.a.	0
	CS Length	2:9 AHB_CLK

32-bit read or write cycles are translated in 2 external bus cycles so the total external bus cycle duration can be obtained through the following formula

Total CS Length (Read/Write Bank x) = (Length of the single cyle)*(2).

However it must be noticed that in case of a 32-bit read cycle, resulting in two external bus accesses, the sampling points used to read data in are not placed symmetrically in the two halves of the access, as it is illustrated by the following formulas:

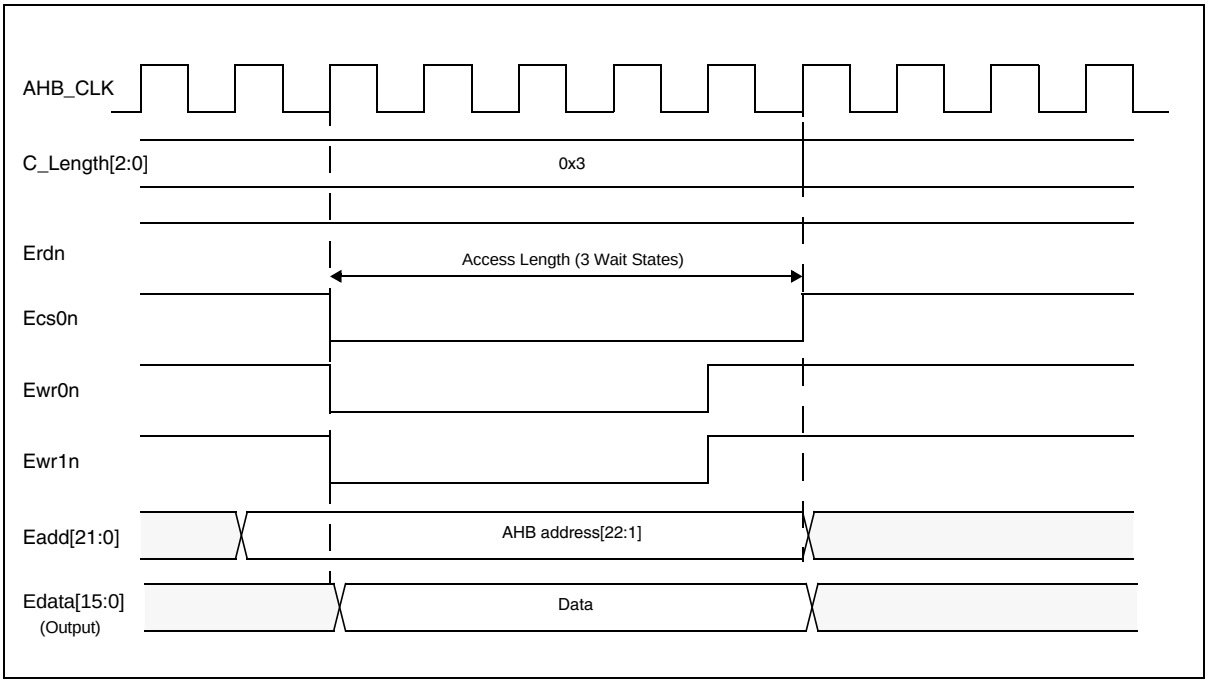
	Sampling point from valid address
Low word sampling point (Read, Bank 0/1) =	C_LENGTH+2
High word sampling point (Read, Bank 0/1 =	C_LENGTH+1

For a timing diagram example see also Section 11.3.3: *Read Access Examples on page 118*.

11.3.2 Write Access Examples

In the Figure 11 shown below, a 16-bit write is being performed. The external bus width is also 16 bits. As can be seen from the diagram, all 2 external write strobes are asserted for the duration of the write cycle. This is a one-cycle write and takes 5 AHB_CLK cycles to complete (C_LENGTH = 3). The 1 LSB of the external address is not modified since it is a 16-bit external bus.

Figure 11. 16-bit write cycle

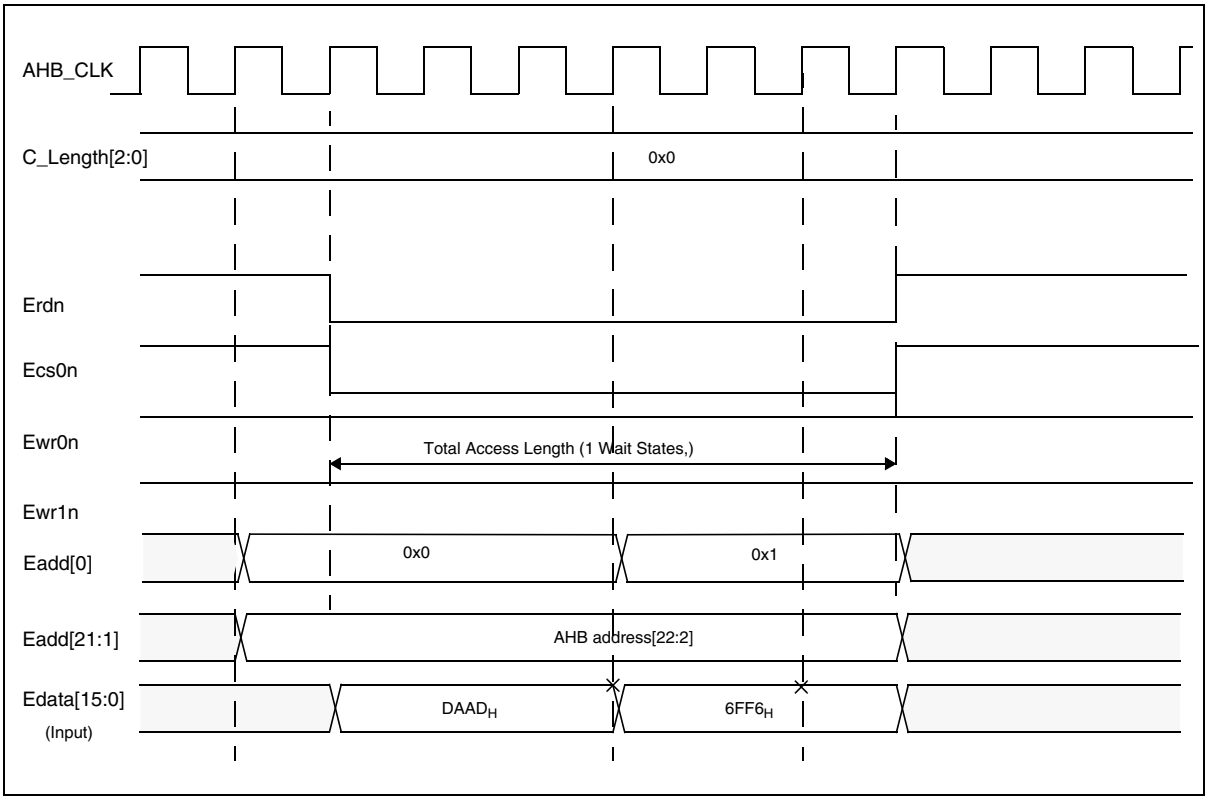


11.3.3 Read Access Examples

The diagram in Figure 12 shows a basic READ operation. In the case of a READ only 1 external read strobe is required (ERDN). In this example, a 32-bit read is being performed on the AHB but since the external bus size is 16 bits, it is necessary for the EMI peripheral to perform 2 successive read operations. The results from the 1st read operation are latched

internally in the EMI block (In this case - “DAAD” - i.e 16 LSBs of EDATA) so that the correct data (i.e “6FF6DAAD”) is output on the AHB Data Bus (“HRDATA”).

Figure 12. 32-bit Read cycle on 16-bit memory bus



For the 1st read operation, EAdd[1] is assigned “0”, for the second this is changed in “1”. Since C_LENGTH = 1, total length of access is 6 AHB_CLK cycles, first word is sampled 3 AHB_CLK cycles after chip select activation and second word is sample 2 AHB_CLK cycles after valid address (see formulas in Section 11.3.1: *EMI Programmable Timings on page 116*). In this example the minimum time for a read cycle is 2 AHB_CLK cycles.

Note *It must be noted that when a 32-bit access cycle is requested, Ecsxn line remains active low across the two 16-bit cycles automatically generated by EMI where only bit 0 of Eadd bus changes. As a consequence EMI should be used in this configuration only with external memory devices that do not latch the address bus on the Ecsxn falling edge, but instead change their output bus data directly following the address bus change while Ecsxn is kept low. This capability is often referred to as “Array data reading” in memory device datasheets.*

11.4 Register description

The following paragraphs give detailed descriptions and operations of each of the bits in the EMI registers.

Bank 0 Configuration Register (BCON0)

Address Offset: 00h

Reset value: 7F3Dh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	B_LENGTH[6:0]							EREN	ISYN0	BE	C_LENGTH[2:0]			Res.	Res.
-	rw							rw	rw	rw	rw			-	-

The Bank 0 Configuration Register (BCON0) is a 16-bit read/write control registers used to configure the operation of external region 0 (Bank 0). The BCON0 control bits are described below.

Bit 15 Reserved. This bit must be always written to ‘0’.

Bit 14:8 **B_LENGTH[6:0]**: *Bank 0 Length*.
The B_LENGTH field selects the extension of the address region 0 in 64 Kbyte increments. The following figures are expressed in bytes.
= 0x00. 64K
= 0x01. 128K
= 0x02. 192K
= 0x03. 256K
...
= 0x7F. 8192K (8M)

Bit 7 **EREN**: *External Ready Enable*.
This bit selects whether read/write cycle termination is controlled by the “Eready” input signal.
= 0. Read/Write cycle termination is not controlled by “Eready” signal.
= 1. Read/Write cycle termination is controlled by “Eready” signal.

Bit 6 **ISYN0**: *Internal Synchronization Bank 0.*
This bit selects whether the Eready signals, if used, has to be internally synchronized or not.
= 0. No internal synchronization. It provides the fastest bus cycles but requires setup and hold times to be met with respect to the EMI internal clock.
= 1. Internal synchronization. Less restrictive, but requires additional wait states caused by the internal synchronization.

Bit 5 **BE**: *Bank 0 Enable*.
BE bit enables Bank 0 of memory.
= 1. Bank enabled.
= 0. Bank disabled.

Bit 4:2 **C_LENGTH[2:0]**: *Cycle Length*.
The C_LENGTH field selects the number of wait states that will be inserted in any read/write cycle performed in memory Bank 0. The total CS length of any read or write cycle will be equal to C_LENGTH+2 periods of the EMI internal clock.
= 0x0. 0 wait states.
= 0x1. 1 wait state.
= 0x2. 2 wait states.
= 0x3. 3 wait states.
....
= 0x7. 7 wait states.

Bit 1 Reserved. This bit must be always written to '0'.

Bit 0 Reserved. **This bit must be always written to '1'.**

Note When asynchronous ready operation is configured (EREN = 1 and ISYN0 = 0) the number of wait states configured in C_LENGTH field must be greater than 4.

Bank 1 Configuration Register (BCON1)

Address Offset: 04h

Reset value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	B_LENGTH[6:0]							EREN	ISYN1	BE	C_LENGTH[2:0]			Res.	Res.
-	rw							rw	rw	rw	rw			-	-

Note *According to I/O configuration, memory bank 1 could not be accessible. In that case this register must be considered reserved and it must be always written to 0001h.*

The Bank 1 Configuration Register (BCON1) is a 16-bit read/write control registers used to configure the operation of external region 1 (Bank 1). The BCON1 control bits are described below:

Bit 15 Reserved. This bit must be always written to '0'.

Bit 14:8 **B_LENGTH[6:0]**: *Bank 1 Length*.
The B_LENGTH field selects the extension of the address region 1 in 64 Kbyte increments. The following figures are expressed in bytes.
= 0x00. 64k
= 0x01. 128k
= 0x02. 192k
= 0x03. 256k
...
= 0x7F. 8192k (8M)

Bit 7 **EREN**: *External Ready Enable*.
This bit selects whether read/write cycle termination is controlled by the "Eready" input signal.
= 0. Read/Write cycle termination is not controlled by "Eready" signal.
= 1. Read/Write cycle termination is controlled by "Eready" signal.

Bit 6 **ISYN1**: *Internal Synchronization Bank 1.*
This bit selects whether the Eready signals, if used, has to be internally synchronized or not.
= 0. No internal synchronization. It provides the fastest bus cycles but requires setup and hold times to be met with respect to the EMI internal clock.
= 1. Internal synchronization. Less restrictive, but requires additional wait states caused by the internal synchronization.

Bit 5 **BE**: *Bank 1 Enable*.
BE bit enables Bank 1 of memory.
= 1. Bank enabled.
= 0. Bank disabled.

Bit 4:2 **C_LENGTH[2:0]**: *Cycle Length*.
The C_LENGTH field selects the number of wait states that will be inserted in any read/write cycle performed in memory Bank 1. The total CS length of any read or write cycle will be equal to C_LENGTH+2 periods of the EMI internal clock.
= 0x0. 0 wait states.
= 0x1. 1 wait state.
= 0x2. 2 wait states.
= 0x3. 3 wait states.
....
= 0x7. 7 wait states.

Bit 1 Reserved. This bit must be always written to '0'.

Bit 0 Reserved. **This bit must be always written to '1'.**

Note When asynchronous ready operation is configured (EREN = 1 and ISYN0 = 0) the number of wait states configured in C_LENGTH field must be greater than 4.

Bank 1 Base Address Register (BBASE1)

Address Offset: 10h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				B_BASE1[11:0]											
-				rw											

Note *According to I/O configuration, memory bank 1 could not be accessible. In that case this register must be considered reserved and it must be always written to 0000h.*

The Bank 1 Base Address Register (BBASE1) is a 16 bit read/write register. It defines the location of the Bank 1 memory region accordingly to the expression: Start Address = 0b_BBASE1[11:0] | 000_0000_0000, where bits 22 down to 11 of the address are determined by the content of the BBASE1 register and remaining bits 10 down to 0 are set to zero. This allows the configuration of the bank start address with a granularity of 2^{11} (2k) bytes in the total EMI addressable space of 2^{23} (8M) bytes.

Bit 15:12 Reserved. These bits must be always written to '0'.

Bit 11:0 **BBASE1[11:0]**. *Bank 1 Start Address Register*.
The 12 bits of the register are the 12 MSB bits of the region start address.

11.4.1 Register map

In the Table 34 an overview of the EMI registers is reported.

Table 34. EMI Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**BCON0**	Res.	B_LENGTH[6:0]							EREN	ISYN0	BE	C_LENGTH[2:0]			Res.	Res1
4	**BCON1**	Res.	B_LENGTH[6:0]							EREN	ISYN1	BE	C_LENGTH[2:0]			Res.	Res1
8	-	Reserved (always write 0001h)															
C	-	Reserved (always write 0001h)															
10	**BBASE1**	Reserved				B_BASE1[11:0]											

Refer to Table 21 on page 50 for the base address.

11.5 Programming considerations

EMI programming and usage is accomplished via the configuration, status and data registers described in previous paragraphs.

At reset time all the EMI registers will be reset to 0x0001 or 0x0000 with the exception of the Bank 0 Configuration Register (BCON0) which, out of reset, will be in the following condition:

- BCON0[15:8]: *Bank Length* (B_LENGTH0) = 0x7F (8M bytes).
- BCON0[7]: *External Ready Enable* (EREN) = 0. External Ready not used.
- BCON0[6]: *Internal Synchronization* (ISYN0) = 0. No internal synchronization of Eready.
- BCON0[5]: *Bank Enable* (BE) = 1. Bank 0 is enabled.
- BCON[4:2]: *Cycle Length* (C_LENGTH) = 0x7. Maximum number of wait states (7).

A special care must be taken on bit 0 of each BCONx register, which must be always written at ‘1’ to assure a correct functionality of the EMI cell. This is true also for the reserved locations at the offset 08h and 0Ch respectively.

The EREN (*External Ready Enable*) bit in BBCON registers should be set only if the device connected externally does have an access time that is variable. If set, ONLY at the end of the programmed number of wait states, the status of the Eready signal will be checked: if it is low a further wait state will be inserted and the process will continue until Eready is sampled high. If EREN is cleared, the status of Eready will be ignored an the read/write cycles will always be completed in the programmed number of wait states.

Since the usage of Eready implies the sampling of an external signal, the user has two options, selected by the status of the ISYN bit. If ISYN=0, this provides the fastest bus cycle but requires that setup and hold times are met with respect to internal peripheral clock. If ISYN=1, Eready is internally synchronized by additional logic, but this will cause additional wait states determined by the internal synchronization process. When ISYN is 0, Eready is not internally resampled so to be sure that it is correctly detected by EMI logic, the number of configured wait states cannot be lower than 4.

The start address and extension of both address regions can be configured via configuration registers. However is up to the user to program them consistently, avoiding overlapping in areas where more than one chip select could be active. A simple mechanism is anyway implemented to avoid possible damages to the external components deriving from this (unwanted) condition: a priority does exist such that no more than one chip select can be active at any time. The priority scheme is as follows:

- CS0 (Highest Priority)
- CS1 (Lowest Priority)

12 ATAPI-IDE INTERFACE

12.1 Introduction

The ATAPI-IDE controller allows the STR720 to interface to up to 2 devices such as Hard Disk or CD-ROM drives. Communication over the IDE bus is performed in PIO mode only. IDE communication in DMA or UDMA modes is not supported. The STR720 initiates communication with the IDE device by writing IDE commands directly in the IDE registers. To save CPU time, the STR720 DMA controller may be used in “memory-to-memory” transfer mode to access the IDE registers.

12.2 Main Features

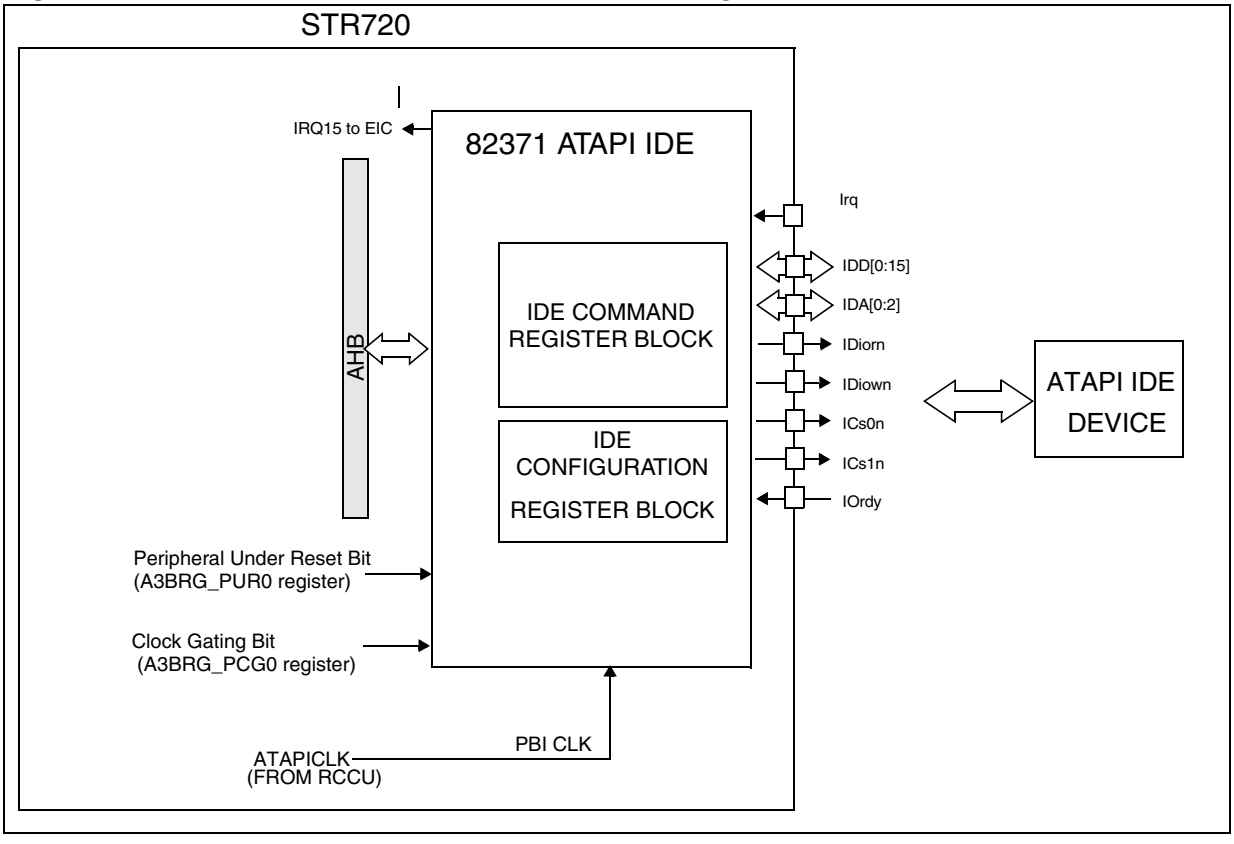
The ATAPI-IDE controller supports the following features:

- Primary-only channel, supporting up to two IDE devices
- Support for CD-ROM and tape peripherals
- Independently programmable timing for each device
- Programmable posted writes and read-prefetch
- Software selectable endianness for data read from CDROM / tape peripheral
- Support for I/O Channel Ready
- Support for PIO modes 0, 2, 3 and 4
- Variable IDE:AHB clock ratio

12.3 Functional Description

The ATAPI-IDE interface allows the application to communicate with an external ATAPI-IDE device via PBI. The 82371-type IDE is connected to the AHB bus as a slave. Operation of the interface is independent of the AHB_CLK:PBI ratio, typical ratios will be 2:1 or 3:1.

Figure 13. ATAPI-IDE Interface overview Block Diagram



AHB Wait states are inserted until the ATAPI-IDE interface has responded to the transaction. The number of wait states will vary depending on the clock ratio and on the stage of the PBI cycle in which the AHB transaction is initiated. Typical values will be 3 or 4 wait states for a ratio of 2:1, and 5 or 6 wait states for a divide ratio of 3:1.

The above assumes that the PBI does not insert any wait states. For each cycle the PBI de-asserts the RDY signal, the AHB will have additional wait states equivalent to the divide ratio inserted. i.e. 2 for 2:1, 3 for 3:1 etc.

12.4 Programming Considerations

This section gives a brief overview of the interaction between the ATAPI-IDE controller and the STR720.

12.4.1 Initialization

Due to the fact that this ATAPI-IDE interface is synchronously reset, all its output signals will remain undefined until a valid clock signal is supplied, regardless of the state of its reset input. As a consequence, when initializing the system to use the ATAPI-IDE interface, the following procedure should be followed:

1. Remove IDE specific reset (controlled by the A3BRG_PUR0 register).
2. Enable AHB clock to the ATAPI IDE Interface (using the A3BRG_PCG0 register).
3. Enable IDE related interrupt, if required (IRQ 15). Refer to the Enhanced Interrupt Controller Chapter).
4. Configure the I/O Port Registers
5. Configure Port 7 for IDE via the P7AS bit in the Miscellanea Global configuration Register
6. Set up IDE controller by programming its configuration registers located from offset 100h to 144h. Refer to Section 12.5.2.1 .

12.4.2 Basic Read Transfer

1. When a read access to the external IDE device is required, the related IDE Command registers should be programmed accordingly, for example by writing the physical address of the device location to be accessed (sector, cylinder, head) and starting the IDE device seek operation.
2. The IDE device flags any relevant event on its side by a rising edge on the interrupt line (P7.45 pin). This will generate an IRQ15 interrupt request to the core if the corresponding interrupt mask bit is enabled. Refer to the EIC register description.
3. In the interrupt service routine the application checks the IDE Status Register to determine which event triggered the interrupt request.
4. If the access to the requested location IDE reported as successful, the data can be fetched through the IDE data register.
   If DMA is used, the application then requests the DMA controller to perform a “memory-to-memory” transfer from the IDE data register (offset 010h) to the buffer in system memory where data is to be stored.
5. The end of transfer will be flagged by the DMA interrupt, notifying that the programmed number of words has been stored in the memory buffer.

12.5 Register map

The register map is split into two sections. One for IDE Command registers and the other for IDE Configuration registers. Bit 8 of the location address is used to decode which bank of registers is being accessed, as it is indicated by the address offset reported in the following tables. All addresses specified for the two register blocks are expressed as offsets with respect to the ATAPI-IDE register base address located at 0xC000_0000.

12.5.1 IDE Command Block Register Set

Addr. Offset	Register Name	7	6	5	4	3	2	1	0
000h-00fh	Reserved								
Primary IDE Port Command/Data									
010h	**Data**	DATA (rw)							
011h	**Error/Features**	ERROR (r) / FEATURES (w)							
012h	**Sector Count**	SECTOR COUNT (rw)							
013h	**Sector Number**	SECTOR NUMBER LBA [7:0] (rw)							
014h	**Cylinder Low**	CYLINDER LOW LBA [15:8] (rw)							
015h	**Cylinder High**	CYLINDER HIGH LBA [23:16] (rw)							
016h	**Device Head**	DEVICE / HEAD LBA [27:24] (rw)							
017h	**Status/ Command**	STATUS (r) / COMMAND (w)							
018h - 025h	Reserved								
Primary IDE Port Control/Status									
026h	**Alternate Status/ Device Control**	ALTERNATE STATUS (r) / DEVICE CONTROL (w)							

12.5.1.1 IDE Command Register Access

Although the IDE Command registers are mapped on byte locations in the STR720 memory map they do not behave in the same way as other memory locations. Accessing a register causes the IDE interface to fetch the selected register from the external IDE device.

The IDE Command registers are accessed individually. This means that it is not possible to access more than one register with a single instruction, even using a 16 or 32- bit instruction.
A word access to Cylinder low, will not access Cylinder high etc.

The IDE data register is different from the other registers in the command register block. Although the data register is mapped on a single byte location, it can be accessed using 16 or 32-bit instructions. A 32-word access results in two 16-bit accesses to the IDE device to complete the word on the AHB bus. A half word access results in one access to the device returning 16 bits of Data.

12.5.2 IDE Configuration Register Set

Addr. Offset	Register Name	7	6	5	4	3	2	1	0
100h 101h	**VENID (Vendor ID)**	Manufacturer's Identification Number *(r)*							
102h 103h	**DEVID (Device ID)**	Device Identification Number *(r)*							
104h	**XCMD (Command)**	Reserved							ATAPI Enable *(rw)*
105h		Reserved							
106h	**XSTS (Status)**	Reserved							
107h		Reserved		Master Target Abort Status (MTAS) *(rc)*	Receive Target Abort (RTA) *(rc)*	Signaled Target Abort Status (STAS) *(rc)*	Devsel Timing Status (DEVTS) *(r)*		Reserved
108h	**REVID (Revision ID)**	Revision Identification Number *(r)*							
109h	**(CCP) Class Code Programming**	Class Code Programming *(r)*							
10Ah	**CCC (Class Code)**	Sub Class 7:0 *(r)*							
10Bh		Base Class 15:8 *(r)*							
10Ch 10Dh	Reserved								
10Eh	**HTYPE (Header Type)**	Defines format of configuration registers as type 0 and single function *(r)*							
10Fh - 13Eh	Reserved								
140h	**PIDETIM (Primary IDE timing)**	Drive 1 Timing Select (TIM1) *(rw)*	Drive 1 Enable Prefetch and Posting (PP1EN) *(rw)*	Drive 1 Enable IOrdy Sampling (RDY1EN) *(rw)*	Drive 1 Fast Timing Bank Select (FTB1) *(rw)*	Drive 0 Timing Select (TIM0) *(rw)*	Drive 0 Enable Prefetch and Posting (PP0EN) *(rw)*	Drive 0 Enable IOrdy Sampling (RDY0EN) *(rw)*	Drive 0 Fast Timing Bank Select (FTB0) *(rw)*
141h		IDE Decode Enabled (DCDEN) *(rw)*	Slave IDE Timing Enable (STEN) *(rw)*	IOrdy Sample Point (ISPP) *(rw)*		Reserved		Recovery Time (RCTP) *(rw)*	
142h 143h	Reserved								
144h	**SLIDETIM (Slave IDE timing)**	Reserved				Primary Drive 1 IOrdy Sample Point (ISPS) *(rw)*		Primary Drive 1 Recovery Time (RCTS) *(rw)*	

Note Access to the IDE Control Registers is not supported. These registers are used to control access to the Master interface of the on-chip IDE controller, which is not supported by this interface.

Note The Master Latency Timer, bus Master interface address, Ultra DMA Control and Ultra DMA Timing Registers in the IDE Configuration block should not be used. These are omitted from the memory map above, flagging their location as reserved.

12.5.2.1 IDE Configuration Register Descriptions

Vendor ID (VENID)

Address Offset: 100h

Reset value: 8086h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Manufacturer's Identification Number [15:0]															
r															

Bits 15:0 = *Manufacturer's Identification Number*

Device ID (DEVID)

Address Offset: 102h

Reset value: 7111h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Device Identification Number [15:0]															
r															

Bits 15:0 = *Device Identification Number*

Command (XCMD)

Address Offset: 104h

Reset value: 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															ATAPI Enable
-															rw

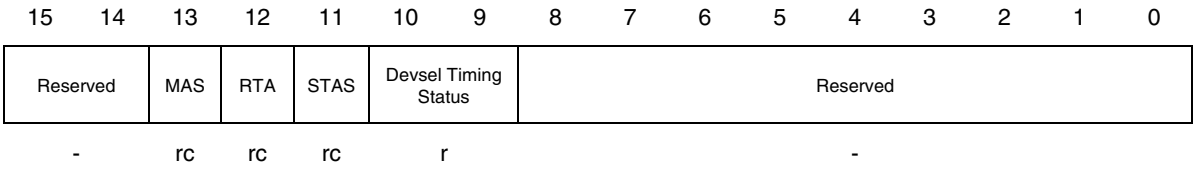
Bit 0 = *ATAPI Enable.*
This must remain set to '1'

Bits 15:1 = Reserved. Must be kept at reset value.

Status (XSTS)

Address Offset: 106h

Reset value:



Bits 8:0 = Reserved. Must be kept at reset value.

Bits 10:9 = **DEVTS**: *Devsel Timing Status.*
01 indicates medium speed timing.

Bit 11 = **STAS**: *Signaled Target Abort Status.*
Set when the ATAPI-IDE interface signals a transaction abort (ABORT signal)

Bit 12 = **RTA**: *Receive Target Abort.*
Set when the ATAPI-IDE interface receives a target abort (MTABORT signal).

Bit 13 = **MTAS**: *Master Abort Status*.
Set when the ATAPI-IDE interface generates a master abort.

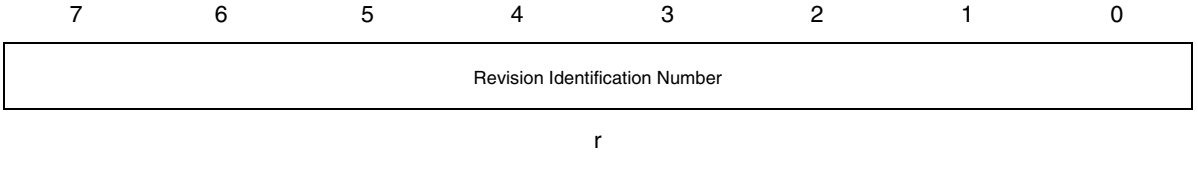
Bits 15:14 = Reserved. Must be kept at reset value.

Note Bits 13, 12 and 11 are Read and Clear (rc). The bit may be read, the read having no effect on the register value. Writing "0" to the bit has no effect; writing "1" will clear the bit.

Revision ID (REVID)

Address Offset: 108h

Reset value: 01h



Bits 7:0 = *Revision Identification Number*

Class Code Programming (CCP)

Address Offset: 109h

Reset value: 80h

7	6	5	4	3	2	1	0
Class Code Programming							
r							

Bits 7:0 = *Class Code Programming*

Class Code (CCC)

Address Offset: 10Ah

Reset value: 0101h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Base Class [15:8]								Sub Class [7:0]							
r								r							

Bits 7:0 = *Class Code: Sub Class.*

Bits 15:8 = *Class Code: Base Class.*

Header Type (HTYPE)

Address Offset: 10Eh

Reset value: 0h

7	6	5	4	3	2	1	0
Configuration Registers Format							
r							

Bits 7:0 = *Configuration Registers Format.*
Defines the format of configuration registers as Type 0 and single-function.

Primary IDE Timing (PIDETIM)

Address Offset: 140h

Reset value: 00h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DCDEN	STEN	ISPP		Reserved		RCTP		TIM1	PP1EN	RDY1EN	FTB1	TIM0	PP0EN	RDY0EN	FTB0
rw	rw	rw		-		rw		rw	rw	rw	rw	rw	rw	rw	rw

Bit 0 = **FTB0**: *Drive 0 Fast Timing Bank Select.*
0: Use 16-bit compatible timings to the Data Port for Drive 0
1: Use timing for Drive 0 defined by bit 3 (TIM0)

Bit 1 =**RDY0EN**: *Drive 0 IOrdy Sampling Enable*
0: Disable IOrdy Sampling for Drive 0
1: Enable IOrdy Sampling for Drive 0

Bit 2 = **PP0EN**: *Drive 0 Prefetch and Posting Enable*
0: Disable Prefetch and Posting for Drive 0
1: Enable Prefetch and Posting for Drive 0

Bit 3 = **TIM0**: *Drive 0 Timing Select*
0: PIO transfers use fast timing for Drive 0
1: PIO transfers use Mode 0 compatible timing for Drive 0

Bit 4 = **FTB1**: *Drive 1 Fast Timing Bank Select.*
0: Use 16-bit compatible timings to the Data Port for Drive 1
1: Use timing for Drive 1 defined by bit 7 (TIM1)

Bit 5 = **RDY1EN**: *Drive 1 IOrdy Sampling Enable*
0: Disable IOrdy Sampling for Drive 1
1: Enable IOrdy Sampling for Drive 1

Bit 6 = **PP1EN**: *Drive 1 Prefetch and Posting Enable*
0: Disable Prefetch and Posting for Drive 1
1: Enable Prefetch and Posting for Drive 1

Bit 7 = **TIM1:** *Drive 1 Timing Select*
0: PIO transfers use fast timing for Drive 1
1: PIO transfers use Mode 0 compatible timing for Drive 1

Bits 9:8 = **RCTP**: *Recovery Time.*
Defines the minimum number of clock periods inactive between successive cycles: 00=4, 01=3,10=2, 11=1

Bits 11:10 = Reserved. Must be kept at reset value.

Bits 13:12 = **ISPP**: *IOrdy Sample Point.*
Defines how many clock periods before Iordry is sampled, 00=5, 01=4, 10=3, 11=2

Bit 14 = **STEN**: *Slave IDE Timing Enable*
0: Disable Slave IDE Timing. RCTP and ISPP configurations apply to both Drives 0/1
1: Enable Slave IDE Timing. RCTP and ISPP configurations apply to just Drive 0. To configure Drive 1 RCTP and ISPP settings, use the SLIDETIM register.

Bit 15 = **DCDEN**: *IDE Decode Enable*
0: Disable IDE Decode
1: Enable IDE Decode

Slave IDE Timing (SLIDETIM)

Address Offset: 144h

Reset value: 00h

7	6	5	4	3	2	1	0
Reserved				ISPS		RCTS	
-				rw		rw	

Bits 1:0 = **RCTS**: *Primary Drive 1 Recovery Time.*
Defines the minimum number of clock periods NIRD/NIWR are inactive between successive cycles: 00=4, 01=3, 10=2, 11=1. Note that bit 14 (STEN) of PIDETIM must be set to 1 for these settings to take effect.

Bits 3:2 = **ISPS**: *Primary Drive 1 IOrdy Sample Point.*
Defines how many clock periods after NIRD/NIWR is asserted on the IDE before IOrdy is sampled: 00=5, 01=4, 10=3, 11=2. Note that bit 14 (STEN) of PIDETIM must be set to 1 for these settings to take effect.

Bits 7:4 = Reserved. Must be kept at reset value.

12.6 Timing

12.6.1 IDE PIO Read/Write Cycles

Figure 14. PIO Read Cycle

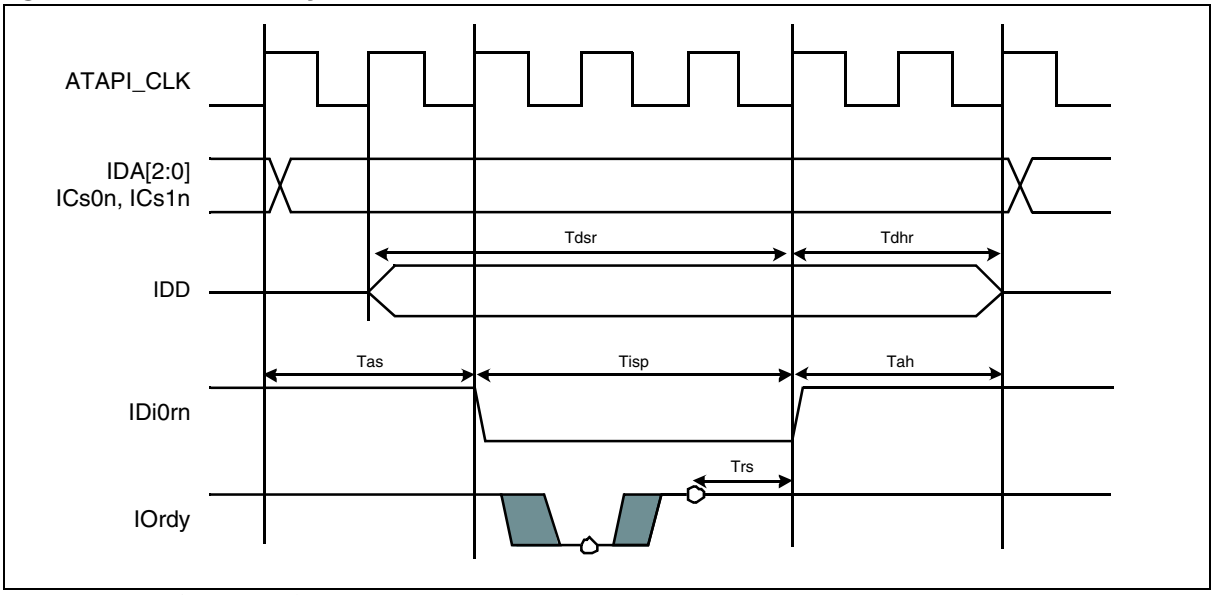


Figure 15. PIO Write Cycle

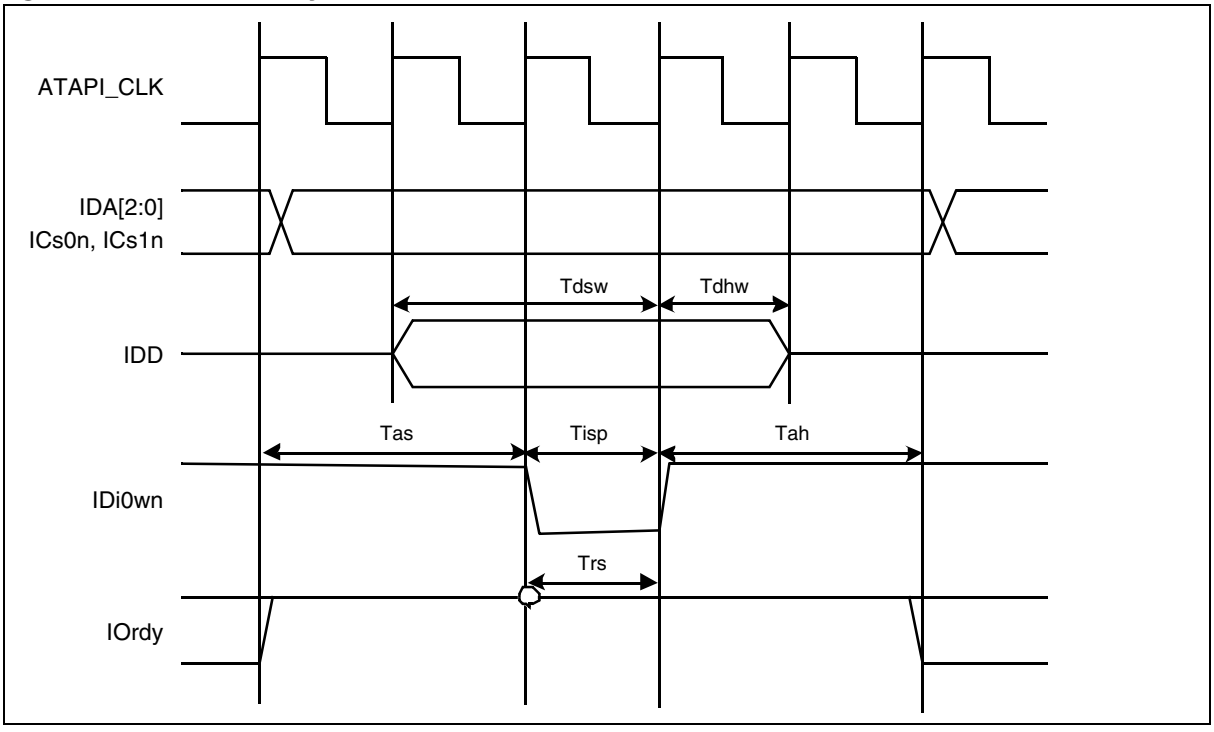


Table 35. Timing descriptions for PIO Read/Write Cycles

SYMBOL	DESCRIPTION	MIN	MAX	UNIT
Trct	Minimum Period between successive IDi0rn / IDi0wn strobes	1	14	T_{ATAPI_CLK}
Tdsr	Data setup to end of IDi0rn	10		ns
Tdhr	Data hold after IDi0rn	5		
Tdsw	Data setup to end of IDi0wn	3		T_{ATAPI_CLK}
Tdhw	Data hold after IDi0wn	1		
Tas	Address setup to start of IDi0rn / IDi0wn	2	4	
Tah	Address hold from end of IDi0rn / IDi0wn	2		
Tisp	IDi0rn / IDi0wn strobe width	3	11	
Trs	IOrdy sample active to IDi0rn / IDi0wn end	1	2	

13 RESET AND CLOCK CONTROL UNIT (RCCU)

13.1 Introduction

Reset and Clock Control Unit (RCCU) is responsible of the control and distribution of the reset and the clock signals of the STR720.

13.2 Main Features

- Three reset inputs: Watchdog reset, Hardware reset and Software reset
- Two clock inputs: CLK and CLK_AF
- Three clock outputs: AHB clock (AHB_CLK), APB clock (APB_CLK) and ATAPI clock (ATAPI_CLK)
- PLL control including:
  - PLL operation mode (POFF output) and multiplication/division (MX and DX Output) control
  - PLL reference (PLLREFCLK) clock selectable between CLK and CLK/2
- Four clock domains:
  - AHB clock can be:
    - CLK clock,
    - CLK clock divided by 2, 32, 64
    - CLK (or CLK/2) clock multiplied by the PLL
    - External low frequency clock (CLK_AF, when available)
  - APB clock can be:
    - AHB clock divided by 2, 4, 8, 16
  - ATAPI clock can be:
    - AHB clock divided by 2
    - AHB clock divided by 3
  - PLLCKREF can be:
    - CLK clock
    - CLK clock divided by 2
- Four different operating modes:
  - Run mode
  - Idle mode
  - Slow mode
  - Stop mode
- Low frequency clock (CLK_AF) selection for low power consumption modes.

13.3 Functional Description

The Reset and Clock Control Unit fully manages all the aspects related to reset assertion, reset release, clock generation and configuration.

TheAPB clock domain is fully asynchronous with respect to the other clock domains. There is a fixed timing relationship between AHB clock domain and ATAPI clock domain (ATAPI clock is aligned on AHB falling edge) thus simplifying the logic interface between these 2 domains.

13.3.1 Reset Management

The Reset Manager generates a reset when one of the following events occurs:

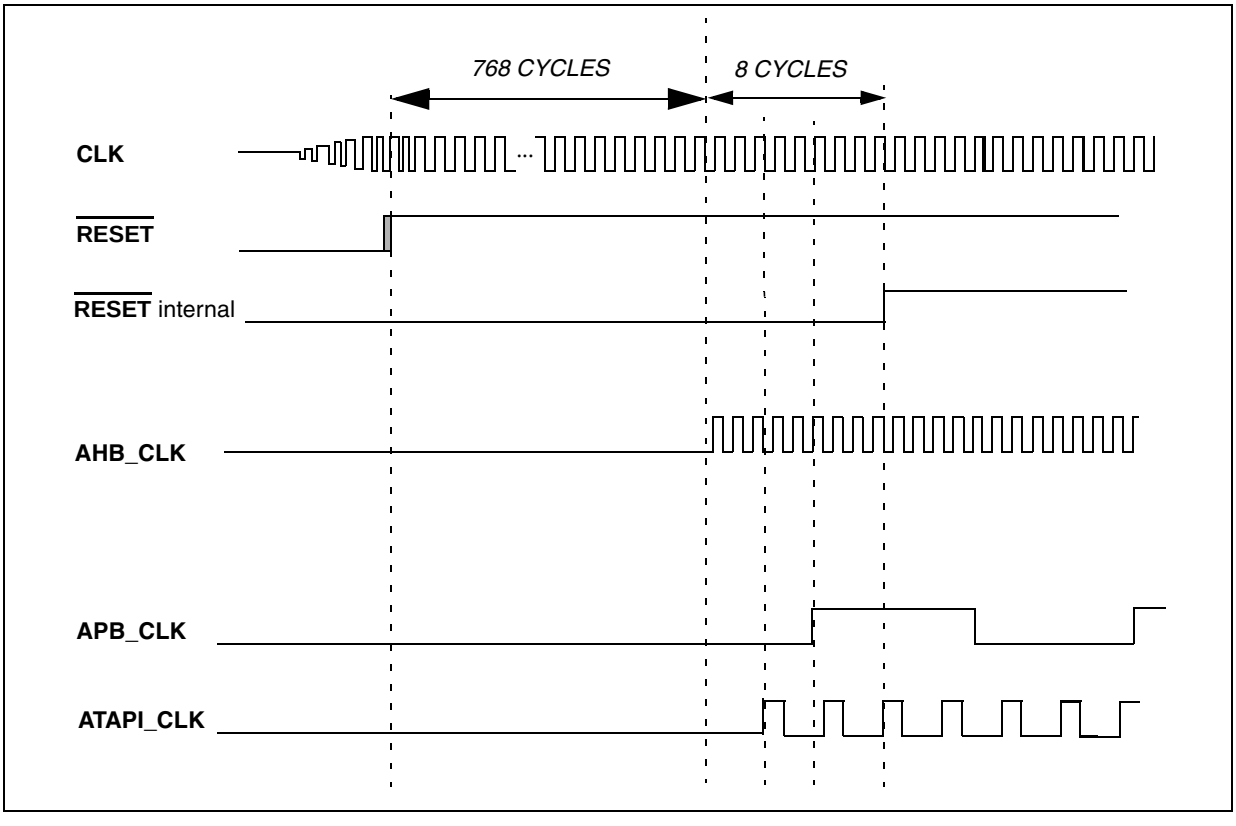
A Hardware reset, initiated by a low level on the Reset pin

A Software reset, forced setting a control bit inside the RCCU

A Watchdog reset

HW reset and Watchdog reset are asynchronous: as soon as one of these Reset inputs is driven low, a Reset cycle is initiated (see Figure 16 on page 139). When the Reset input goes high again or a Software reset is asserted, 768 CLK plus 8 AHB_CLK cycles are counted before exiting the reset state. (this signal is not used in STR720 device)

Figure 16. Reset timing



When $\overline{\text{RESET}}$ internal is deasserted, the RCCU enters Run mode of operation (see below). The event causing the last Reset is flagged in the CLKFLAG register: the corresponding bit is set. A hardware initiated reset will leave all these bits reset. The hardware reset overrides all other conditions and forces the system to the reset state. During the Reset phase, the internal registers are set to their reset values, where these are defined.

Note *On STR720 device, hardware reset pin has to be kept low for a duration longer than the on-chip RC filter width (between 50 ns and 500 ns) in order to start the reset sequence described above.*

13.3.2 PLL Management

After reset the PLL is not enabled and it must be turned on and enabled by the software.
The CLK signal drives a programmable divide-by-two circuit. If the DIV2 control bit in CLK_ FLAG register is set, PLLREFCLK, is equal to CLK divided by two; if DIV2 is reset, PLLREFCLK is identical to CLK. The divide- by- two circuit ensures also a 50% duty cycle signal. When the PLL is active, it multiplies PLLREFCLK by a factor depending on the status of the MX[5:0] bits in PLLCONF register. The multiplied clock is then divided, inside the PLL, by a factor determined by the status of the DX[1:0] bits in PLLCONF register;

The PLL contains a frequency comparator between PLLREFCLK and the PLL clock output that verifies if the PLL clock has stabilized (locked status). The bit LOCK in PLLCONF register becomes 1 when this condition occurs and maintains this value as long as the PLL is locked, going back to 0 if for some reason (noise on input clock line, switch off of PLL, stop and restart of PLLREFCLK and so on) looses the programmed frequency in which it was locked. It is possible to select the PLL clock as system clock only when the LOCK bit is ’1’ (the selection is done by clearing PLLBYP bit).

PLL LOCK status changing is monitored by two different interrupt pending bits:

- LOCK_I monitors PLL LOCK (0 to 1 transition of LOCK bit)
- ULOCK_I monitors PLL UNLOCK (1 to 0 transition of LOCK bit)

System clock switching activity is monitored by DIV2_F, PLLBYP_F, IDLE_F, SLOW_F flags inside CLKFLAG register.

In order to activate PLL to generate system clock, the following procedure is recommended:

1. Set multiplication/division factors in PLLCONF register, while PLL is still switched off.
2. Switch on PLL by resetting PLLOFF bit in PLLCONF register.
3. Wait for LOCK bit in PLL CONF register to become ‘1’ (or use related interrupt event).
4. Switch system clock to PLL output, by resetting PLLBYP bit in PLLCONF register.

Alternatively, if exact system clock frequency is not required during system set-up phase, PLLBYP bit can be reset also immediately after resetting PLLOFF bit in PLLCONF register (but not in the same instruction). In this way system clock is switched immediately to PLL output, even if it has not yet settled to its final value.

If LOCK bit return to'0' and AUTOBYP_EN bit is set to '1', the system clock switches back to PLLREFCLK disregarding PLLBYP bit value. Otherwise (i.e. AUTOBYP_EN = '0') the software has to handle the system clock switching, by monitoring PLL lock and by writing a proper bit in CLKCTL register. AUTOBYP feature can be used to avoid polling LOCK bit (or enabling the related interrupt) during system start-up phase to automatically switch to PLL clock once LOCK condition has been achieved.

Note Care must be taken in using AUTOBYP feature during normal system operation, since the sudden change of system clock frequency can lead to serial interfaces problem. It is recommended to use this feature with particular care and keeping LOCK related interrupts active, so to be notified about the clock change event.

When PLL block needs to be switched off, for example to reduce current consumption upon entering a power save mode, a special sequence is required to avoid blocking the system:

1. Set PLLBYP bit at '1' in PLLCONF register, so to switch back system clock to PLLREF-CLK.
2. Wait for PLLBYP_F bit in CLKFLAG register to be read back as '1', to confirm that system clock switch is completed.
3. Set PLLOFF bit at '1' in PLLCONF register to safely switch off PLL block.

In case PLLOFF is set to '1' without observing the sequence above, system clock stops being generated and the device hangs up to the next hardware reset event.

Whenever PLL multiplication/division factor needs to be changed, a switch-off/switch-on cycle must be performed, following the procedures detailed in the paragraphs above.

WARNING A wrong clock selection may cause the system to stop indefinitely. In this case the only way to start again is a hardware reset.

13.3.3 Mode of Operation

Four modes of operation are provided

13.3.3.1 Run mode

Run mode is entered when Software reset or a Hardware reset input is asserted (low) or a Watchdog/External reset event occurs, and the entire system asynchronously enter the reset state. RCCU reset exit status is the following:

PLL is OFF, bypassed and MX = "000111" and DX = "11"

CLK is selected as AHB clock

CLK/16 is selected as APB clock

CLK/2 is selected as ATAPI clock

13.3.3.2 Idle mode

If SLOW bit in CLKCTL register is ’0’, then a low frequency internal clock, (PLLREFCLK/32), could be used, setting IDLE bit to ’1’, as source clock for AHB clock and APB, ATAPI prescalers reference clock. IDLE mode can be used when a low power consumption is needed, but no CLK_AF is available as external low frequency clock source.

13.3.3.3 SLOW mode

When active (SLOW bit = ’1’) the external CLK_AF low frequency clock source is selected as system clock. (refer also to Idle mode)

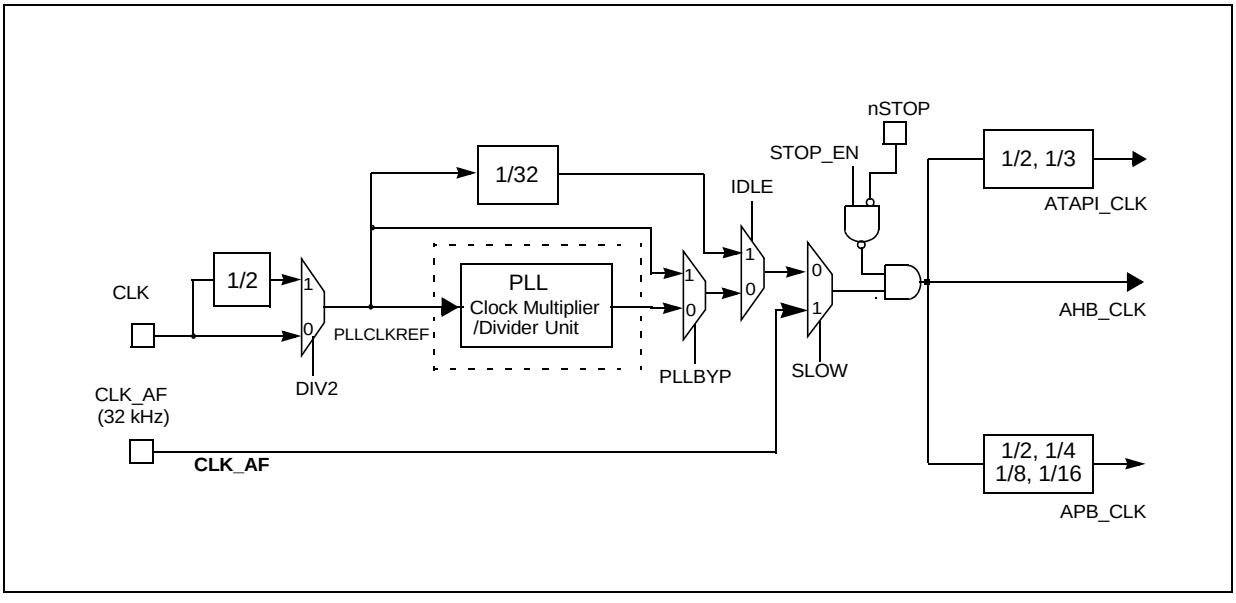
13.3.3.4 STOP mode

A STOP mode is provided, where all clocks are stopped, while power supply is maintained to the whole chip. STOP mode is entered only if STOP_EN bit inside MSKCTL register has been written to ’1’. In this case STOP input driven active (high) causes the RCCU to stretch all system clocks.

Normal operation may be resumed by an external event, connected to STOP input (refer to WIU specification). After wake-up, operations resume from previous state without any loss of information.

WARNING: All low power modes are completely under software control via the proper configuration of the RCCU registers.

Figure 17. Clock Control unit programming



13.4 Register description

PLL CONFiguration register (PLLCONF)

Address Offset: 00h

Reset value: 03C7h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOCK	AUTOBYP_EN	Reserved				PLLOFF	PLLBYP	DX[1:0]		MX[5:0]					
r	rw	-				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This register is write protected and can be unlocked using REG_PROT bit in SYSPROT register.

Bit 15 = **LOCK**: *PLL locked in*
This bit is read only.
0: The PLL is not locked or turned OFF and cannot be selected as system clock source
1: The PLL is locked

Bit 14 = **AUTOBYP_EN**: *auto-bypass enable*
This bit enables the auto-bypass mode. In auto-bypass mode when the LOCK bit return to '0' the system clock switches back to PLLCLKREF even if the PLLBYP bit is '0'.
0: auto-bypass mode off
1: auto-bypass mode on

Bit 13:10 = Reserved. Give '0' when read and must be written to '0'.

Bit 9 = **PLLOFF:** *PLL switch OFF*
0: PLL is switched on
1: PLL is switched off

Bit 8 = **PLLBYP:** *PLL bypass bit (see also AUTOBYP_EN bit)*
0: PLL provides the system clock
1: PLL is bypassed and PLLCLKREF is used as system clock

Note *If PLLBYP changes from '1' to '0', but PLLOFF bit is '1', no clock switch occurs since PLL is switched on again and a stable clock is available, but PLL may be not yet locked.*

Bit 7:6 = **DX[1:0]:** *PLL output clock divider factor.*
This bit field selects the divider factor used on the PLL output clock. Refer to Table 36 for PLL divider setting.

Table 36. PLL divider settings

DX1	DX0	CK
0	0	PLLCLKREF/1
0	1	PLLCLKREF/2

Table 36. PLL divider settings

1	0	PLLCLKREF/4
1	1	PLLCLKREF/8

Bit 5:0 = **MX[5:0]:** *PLL Multiplication Factor.*
This bit field selects the multiplier factor used to generate PLL output clock from its input reference clock. Refer to Table 37 for multiplier settings

Table 37. PLL multiplication factors

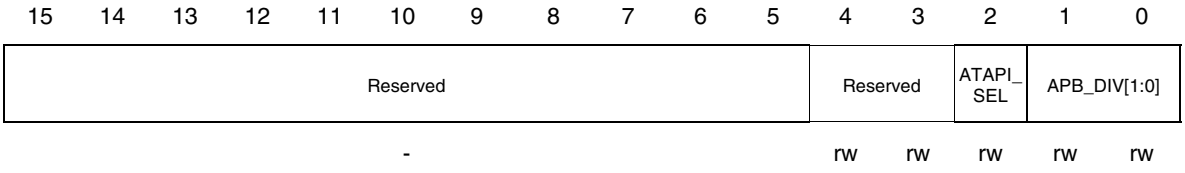
MX5	MX4	MX3	MX2	MX1	MX0	Multiply factor
0 to 7						N/A
0	0	0	1	1	1	8
0	0	1	0	0	0	9
0	0	1	0	0	1	10
...	...	...	...	...	...	...
0	0	1	1	1	1	16
0	1	0	0	0	0	17
...	...	...	...	...	...	...
0	1	1	1	1	1	32
1	0	0	0	0	0	33
...	...	...	...	...	...	...
1	1	1	1	1	1	64

WARNING Not all clock selections are possible and setting the system clock to any unsupported frequency may cause the system to malfunction. In this case the only way to start again is a hardware reset. For the allowed values, refer to section 25 on page 364.

DIViders CONFiguration register (DIVCONF)

Address Offset: 04h

Reset value: 001Fh



This register is write protected and can be unlocked using REG_PROT bit in SYSPROT register.

Bit 15:5 = Reserved. Give ‘0’ when read and must be written to ‘0’.

Bit 4:3 = Reserved.

Bit 2 = **ATAPI_SEL**: *ATAPI clock select*
0: select AHB_CLK/2 as ATAPI clock
1: select AHB_CLK/3 as ATAPI clock

Bit 1:0 = **APB_DIV[1:0]**: *APB output clock divider factor.*
This bit field configures the A-APB clock generation. Refer to Table 38 for A-APB divider settings.

Table 38. APB divider settings

APB_DIV1	APB_DIV0	APB_CK
0	0	AHB_CLK/2
0	1	AHB_CLK/4
1	0	AHB_CLK/8
1	1	AHB_CLK/16

CLocK FLAG register (CLKFLAG)

Address Offset: 08h

Reset value: 0020h after Hardware reset

Reset value: 0021h after Software reset

Reset value: 0022h after Watchdog reset

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOCK_I	ULOCK_I	Reserved							DIV2_F	PLLBYP_F	IDLE_F	SLOW_F	Res.	WDGRST_F	SWRST_F
rc	rc	-							r	r	r	r	-	r	r

Bit 15 = **LOCK_I**: *lock changing interrupt bit*
This bit indicates a 0 to 1 transition on the PLL LOCK signal. Only RCCU block can set it to '1' when PLL reaches the lock condition. Application software writing '1' on this bit will clear its value; writing '0' has no effect.
0: No lock interrupt request is provided
1: lock interrupt request is active

Bit 14 = **ULOCK_I**: un*lock changing interrupt bit*
This bit indicates a 1 to 0 transition on the PLL LOCK signal. Only RCCU block can set it to '1' when PLL loses the lock condition. Application software writing '1' on this bit will clear its value; writing '0' has no effect.
0: No unlock interrupt request is provided
1: unlock interrupt request is active

Bit 13:7 = Reserved. Give '0' when read and must be written to '0'.

Bit 6 = **DIV2_F**: *DIV2 mux status flag*
This bit is read only. It indicates the actual status of the PLLCLKREF clock
0: indicates that PLLCLKREF is CLK
1: indicates that PLLCLKREF is CLK/2

Bit 5 = **PLLBYP_F**: *PLL mux status flag*
This bit is read only. It indicates that the PLLCLK clock is actually selected.
0: indicates that PLLCLK is selected
1: indicates that PLLCLK is not selected

Bit 4 = **IDLE_F**: *IDLE mux status flag*
This bit is read only. It indicates that the IDLE clock (PLLCLKREF/32) is actually selected.
0: indicates that IDLE clock is not selected
1: indicates that IDLE clock is selected

Bit 3 = **SLOW_F**: *SLOW mux status flag*
This bit is read only. It indicates that the CLK_AF clock is actually selected.
0: indicates that CLK_AF clock is not selected
1: indicates that CLK_AF clock is selected

Bit 2 = Reserved. Give ‘0’ when read and must be written to ‘0’.

Bit 1 = **WDGRST_F**: *watchdog reset status flag*
This bit is read only.
0: No Watchdog reset occurred.
1: Watchdog reset occurred.

Bit 0 = **SWRST_F**: *software reset status flag*
This bit is read only.
0: No software reset occurred.
1: Software reset occurred.

CLocK ConTroL register (CLKCTL)

Address Offset: 0Ch

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												SWRST	DIV2	IDLE	SLOW
-												rw	rw	rw	rw

This register is write protected and can be unlocked using REG_PROT bit in SYSPROT register.

Bit 15:4 = Reserved. Give ‘0’ when read and must be written to ‘0’.

Bit 3 = **SWRST**: *software reset bit*
0: No software reset occurs.
1: Software reset occurs.

Bit 2 = **DIV2**: *DIV2 select*
0: Select CLK as PLLCLKREF
1: Select CLK/2 as PLLCLKREF

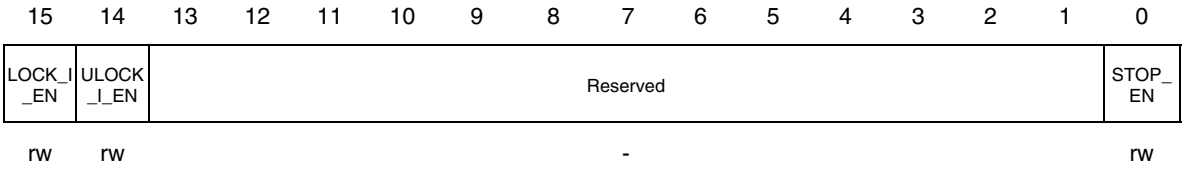
Bit 1 = **IDLE**: *IDLE mode select bit*
0: IDLE mode not selected
1: Select IDLE mode

Bit 0 = **SLOW**: *Slow mode select bit*
0: SLOW mode not selected
1: Select SLOW mode

MaSK ConTroL register (MSKCTL)

Address Offset: 10h

Reset value: 0000h



Bit 15 = **LOCK_I_EN**: *Lock interrupt masking bit*
0: No interrupt requests on PLL LOCK are provided by RCCU
1: The RCCU can request a LOCK interrupt

Bit 14 = **ULOCK_I_EN**: Un*Lock interrupt masking bit*
0: No interrupt requests on PLL UnLOCK are provided by RCCU
1: The RCCU can request a UnLOCK interrupt

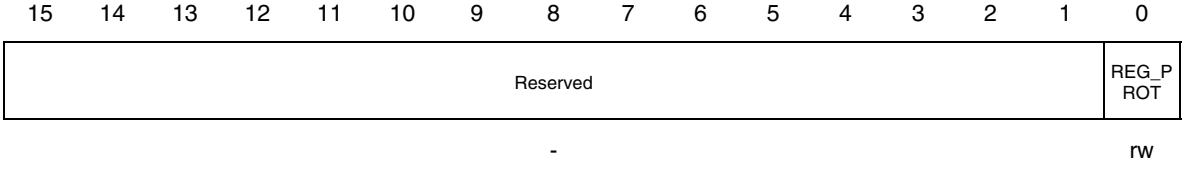
Bit 13:1 = Reserved. Give ‘0’ when read and must be written to ‘0’.

Bit 0 = **STOP_EN**: *STOP mode enable bit*
0: STOP mode is not enabled.
1: STOP mode enabled.

SYStem PROTection register (SYSPROT)

Address Offset: 14h

Reset value: 0000h



Bit 15:1 = Reserved. Give ‘0’ when read and must be written to ‘0’.

Bit 0 = **REG_PROT**: *Register write protection bit*
This bit is set and cleared by software. It is cleared by hardware after every write access to PLLCONF, DIVCONF and CLKCTL registers. This bit enables the PLLCONF, DIVCONF and CLKCTL registers writing.
0: write access to PLLCONF, DIVCONF and CLKCTL registers is disabled
1: write access to PLLCONF, DIVCONF and CLKCTL registers is enabled

13.4.1 Register map

Table 39. RCCU Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00	**PLLCONF**	LOCK	AUTOBYP_EN	Reserved				PLLOFF	PLLB-YP	DX[1:0]		MX[5:0]					
04	**DIVCONF**	Reserved											Reserved		ATAPI_SEL	APB_DIV[1:0]	
08	**CLKFLAG**	LOCK_I	ULOCK_I	Reserved							DIV2_F	PLLBYP_F	IDLE_F	SLOW_F	Res.	WDGRST_F	SWRST_F
0C	**CLKCTL**	Reserved												SWRST	DIV2	IDLE	SLOW
10	**MSKCTL**	LOCK_I_EN	ULOCK_I_EN	Reserved													STOP_EN
14	**SYSPROT**	Reserved															REG_PROT

Refer to Table 21 on page 50 for the base address.

14 REAL TIME CLOCK (RTC)

14.1 Introduction

The RTC provides a set of continuously running counters which can be used, with suitable software, to provide a clock-calendar function. The counters values can be written to set the current time/date of the system.

The RTC includes an APB slave interface, to provide access by word to internal 48-bit registers. this interface is properly disconnected by the APB bus when the main power supply is removed.

14.2 Main Features

- 48-bit programmable counter for long term measurement
- External clock input, CLK_AF, defining RTC ticks (must be at least 4 times slower than APB clock) driven by two different sources:
  - 32 kHz crystal oscillator connected to OSCIN and OSCOUT
  - External clock generator connected to OSCIN
- Separate power supply
- 2 dedicated maskable interrupt lines:
  - alarm interrupt, to generate a software programmable alarm interrupt (up to 1 s)
  - periodic interrupt, to generate a software programmable periodic interrupt (up to 32 s)

14.3 Functional Description

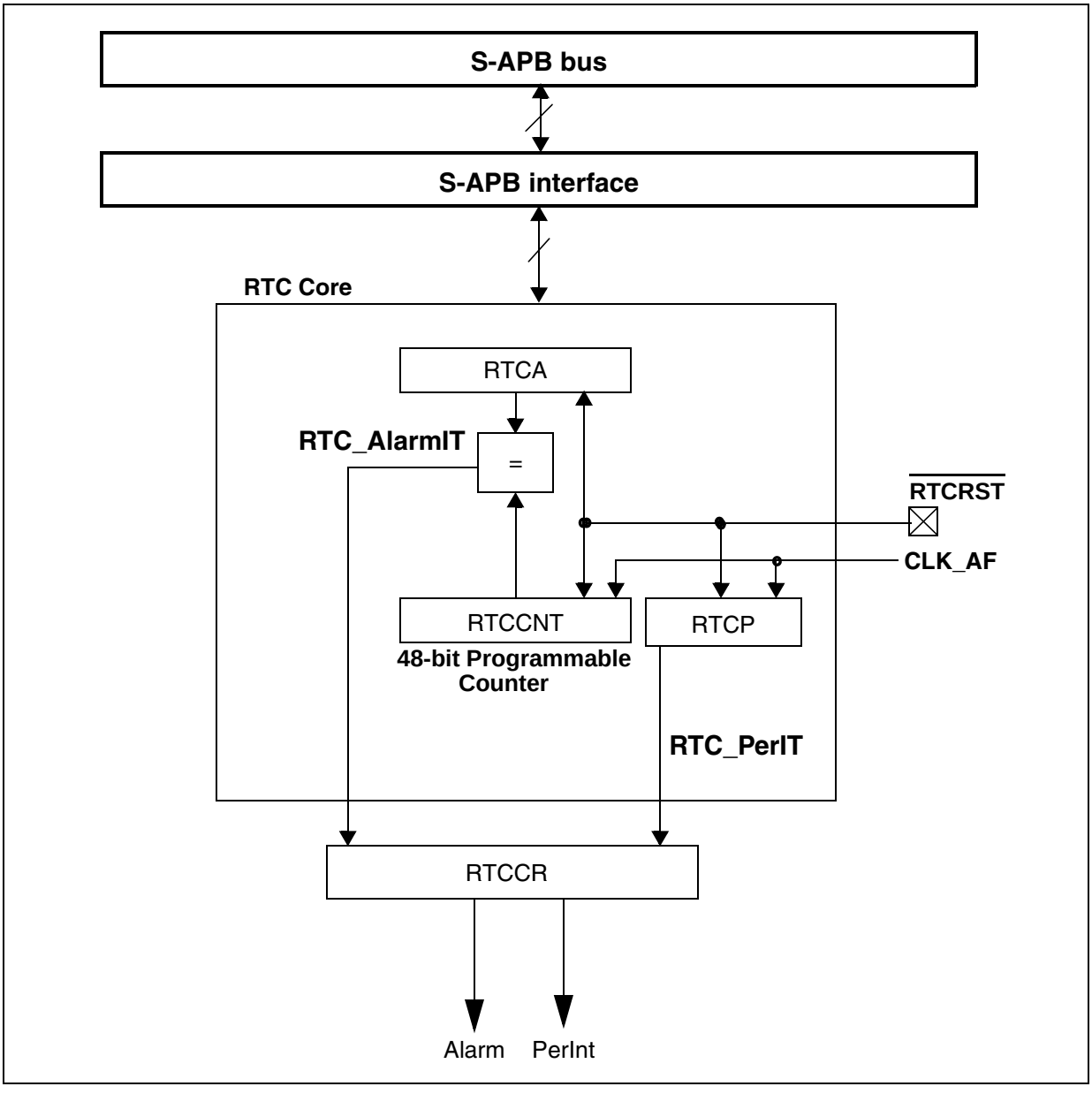
14.3.1 Overview

The RTC consists of two main units (see Figure 18 on page 151), the first one (S-APB Interface) is used to interface with the APB bus. This unit also contains a set of 16-bit registers, synchronous to the S-APB clock and accessible from the APB bus in read or write mode (for more details refer to Register description section). The APB interface is clocked by S-APB clock in order to interface with APB bus.

The other unit (RTC Core) consists of three programmable registers. The first is made of a 48-bit programmable counter that may be initialized to the current system time. The system time is incremented at CLK_AF rate and compared with a programmable date (stored in the second 48-bit register, RTCA) in order to generate an alarm via an interrupt (Alarm), if enabled in the RTCCR control register. The third register is the 20-bit RTCP register that is

loaded with the value of the desired period to create a periodic interrupt of a programmable value.

Figure 18. RTC simplified block diagram



14.3.2 Reset procedure

All system registers are asynchronously reset by either an external Reset event ($\overline{\text{RSTIN}}$ device input pin, forced low) or an internal Reset event (WDG or Software Reset), except RTCA, RTCP, RTCCNT registers. To reset them, the $\overline{\text{RTCRST}}$ device input pin has to be forced low: RTCA, RTCP and RTCCNT registers are reset to FFFF_FFFF_FFFFh, 0_8000h and 0000_0000_0000h respectively.

14.3.3 Free-running mode

After reset the peripheral will enter in free-running mode. In this operating mode the RTC programmable counter starts counting. Interrupt flags are activated too but, since interrupt signals are masked, there is no interrupt generation. Interrupt signals must be enabled by setting the appropriate bits in RTCCR register. In order to avoid spurious interrupt generation it is recommended to clear old interrupt requests before enabling them.

14.3.4 Configuration mode

To write to the RTCCNT, RTCP and RTCA registers, the peripheral must enter Configuration mode. This is obtained writing CNF bit at ‘1’.

In addition, the writing operation is only enabled if the previous writing operation is finished. To enable software to detect this situation the status bit RTOFF is provided in the RTCCR control register to indicate that an update of the registers is in progress. A new value must not be written to the RTC counters until the status bit value is ‘0’ since as long as RTOFF bit is ‘0’ all attempts to write any RTC register will fail.

A special case of configuration mode is entered also as soon as RTCCR register is written with CNF at ‘0’. In this case RTOFF becomes ‘0’ immediately and it remains as such for 1 or 2 CLK_AF cycles, preventing any subsequent modification until its normal state is restored. During this special configuration mode, only RTCCR contents are updated, while RTCCNT, RTCP and RTCA registers remain unaffected.

14.4 Register description

The RTC registers can only be accessed by word. The reserved bits can not be written and they are always read at ‘0’, unless otherwise specified.

RTCCR: RTC Control Register

Address Offset: 00h

Reset value: 0020h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					PEREN	AEN	Res.	Reserved		RTOFF	CNF	Res.	PERIR	AIR	Res.
-					rw	rw	-	-		r	rw	-	rc	rc	-

The functions of the RTC are controlled by this control register. It is not possible to write this register while the peripheral is completing a previous write operation (flagged by RTOFF=0 (see “Configuration mode” on page 152).

Bit 15:11 = Reserved. These bits must always be written to ‘0’.

Bit 10 = **PEREN**: *PERiodic interrupt ENable*
0: Periodic interrupt is masked.
1: Periodic interrupt is enabled.

Bit 9 = **AEN**: *Alarm interrupt ENable*
0: Alarm interrupt is masked.
1: Alarm interrupt is enabled.

Bit 8 = Reserved. This bits must always be written to ‘0’.

The above listed bits are used to mask interrupt requests. Note that at reset all interrupts are disabled, so it is possible to write the RTC registers so that no interrupt requests are pending after initialization.

Bits 7:6 = Reserved. These bits must always be written to ‘0’.

Bit 5 = **RTOFF**: *RTc operation OFF*
With this bit RTC reports the status of the last write operation performed on its registers, indicating if it has been completed or not. If its value is ‘0’ then it is not possible to write any RTC register. This bit can only be read by software.
0: Last write operation on RTC registers is still active.
1: Last write operation on RTC registers terminated.

Note A read operation on RTC registers with RTOFF=0 returns anyway their current value.

Bit 4 = **CNF**: *CoNfiguration Flag*
This bit must be set at ‘1’ by software in order to enter the configuration mode thus allowing to write new values in RTCCNT, RTCA, RTCP registers. The writing operation is really executed when the CNF bit after being set, is reset again by software.
0: Exit configuration mode (start update of RTC registers).
1: Enter configuration mode.

Bit 3 = Reserved. This bits must always be written to ‘0’, **but it will be read back as ‘1’**.

Bit 2 = **PERIR**: *PERiodic Interrupt Request*
This bit stores the status of periodic interrupt request signal (*RTC_PerIT*) generated by the 20 bit programmable counter when the threshold set in RTCP register is reached. When this bit is at '1', the corresponding interrupt will be generated only if PEREN bit is set to '1'. PERIR bit can be set at '1' only by hardware and can be cleared only by software, while writing '1' will left it unchanged.
0: Periodic interrupt condition not met.
1: Periodic interrupt request pending.

Bit 1 = **AIR**: *Alarm Interrupt Request*
This bit contains the status of periodic interrupt request signal (*RTC_AlarmIT*) generated by the 48 bit programmable counter when the threshold set in RTCA register is reached. When this bit is at '1', the corresponding interrupt will be generated only if AEN bit is set to '1'. AIR bit can be set at '1' only by hardware and can be cleared only by software, while writing '1' will left it unchanged.
0: Alarm interrupt condition not met.
1: Alarm interrupt request pending.

Note The alarm event generated during the standby mode (and so with the device kept under Reset by external $\overline{RSTIN}$ forced low) does not set the AIR bit.

Note Any alarm event generated during STOP mode occurs after 1s regardless of the value programmed in the RTCA register.

Bit 0 = Reserved. This bits must always be written to '0', **but it will be read back as '1'**.

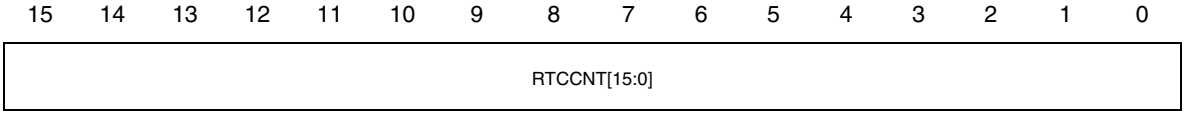
Any interrupt request remains pending until the appropriate RTCCR request bit is reset by software, notifying that the interrupt request has been granted.

RTCCNT: RTC Counter registers

The RTC counter has three 16 bit programmable counters; their count rate is based on the *CLK_AF* time reference. RTCCNT registers keep the counting value of these counters. They are write protected by bit RTOFF of RTCCR register, write operation is allowed if RTOFF value is '1'. A write operation on any of the three registers (CNT_H, CNT_M or CNT_L) loads directly the corresponding programmable counter. When reading, the current value in the counter (system date) is returned. The counters keep on running while the external clock oscillator is working even if the system is powered down.

RTCCNT_L

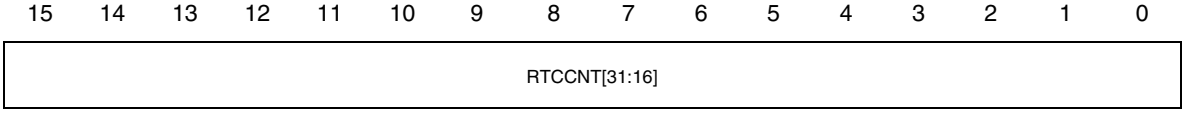
Address Offset: 04h
Reset value: 0000h



Bit 15:0 = **RTCCNT[15:0]:** *RTC CouNTer Low*
Reading RTCCNT_L register, the current value of the lower part of RTC Counter register is returned. To write this register it is required to enter into configuration mode (see RTOFF bit of RTCCR register).

RTCCNT_M

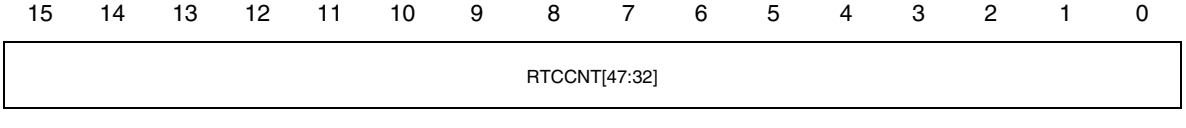
Address Offset: 08h
Reset value: 0000h



Bit 15:0 = **RTCCNT[31:16]:** *RTC CouNTer Middle*
Reading RTCCNT_M register, the current value of the middle part of RTC Counter register is returned. To write this register it is required to enter into configuration mode (see RTOFF bit of RTCCR register).

RTCCNT_H

Address Offset: 0Ch
Reset value: 0000h



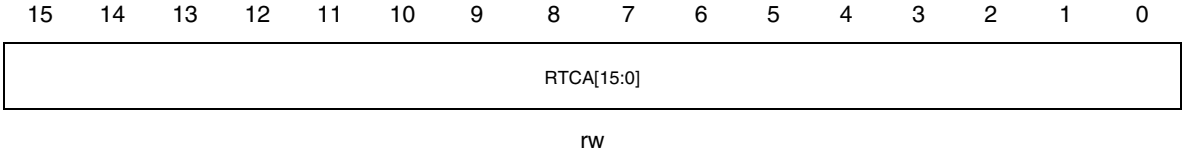
Bit 15:0 = **RTCCNT[47:32]:** *RTC CouNTer High*
Reading RTCCNT_H register, the current value of the high part of RTC Counter register is returned. To write this register it is required to enter into configuration mode (see RTOFF bit of RTCCR register).

RTCA: RTC Alarm registers

When the programmable counter RTCCNT reaches the 48 bit value stored into RTCA registers an alarm is triggered and the interrupt request RTC_alarmIT is generated. This register is write protected by bit RTOFF of RTCCR register, write operation is allowed if RTOFF value is '1'.

RTCA_L

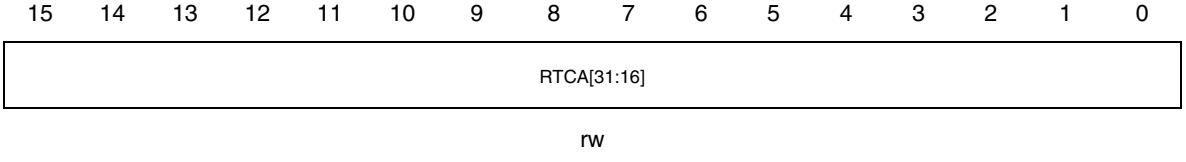
Address Offset: 10h
Reset value: FFFFh



Bit 15:0 = **RTCA[15:0]:** *RTC Alarm Low*
Reading RTCA_L register, the lower part of alarm time is returned. To write this register it is required to enter into configuration mode (see RTOFF bit of RTCCR register).

RTCA_M

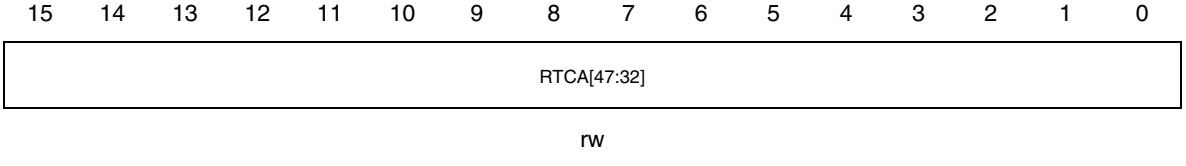
Address Offset: 14h
Reset value: FFFFh



Bit 15:0 = **RTCA[31:16]:** *RTC Alarm Middle*
Reading RTCA_M register, the middle part of alarm time is returned. To write this register it is required to enter into configuration mode (see RTOFF bit of RTCCR register).

RTCA_H

Address Offset: 18h
Reset value: FFFFh



Bit 15:0 = **RTCA[47:32]:** *RTC Alarm High*
Reading RTCA_H register, the high part of alarm time is returned. To write this register it is required to enter into configuration mode (see RTOFF bit of RTCCR register).

RTCP: RTC Periodic value registers

These registers keep the period counting value of the RTC periodic tick. It is write protected by bit RTOFF of RTCCR register, write operation is allowed if RTOFF value is '1'. They are used to strobe a periodic interrupt signal, *RTC_PerIT*, each time the counter counts down from the RTCP value. When it reaches this threshold it resets itself and begins again. This signal is independent of the other system interrupts in terms of alignment. The reset value sets the register signal period to 1 second. When reading, the current value in the counter elapsing the period of the programmable tick is returned.

Note If the RTCP registers are loaded with the value 00000000h then the periodic interrupt will always be high until the register is loaded with a different periodic value.

RTCP_L

Address Offset: 1Ch
Reset value: 8000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTCP[15:0]															
rw															

Bit 15:0 = **RTCP[15:0]:** *RTC Periodic Low*
Reading RTCP_L register, the low part of current RTCP counter value is returned. To write this register it is required to enter into configuration mode (see RTOFF bit of RTCCR register).

RTCP_H

Address Offset: 20h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												RTCP[19:16]			
-												rw			

Bit 15:4 = Reserved. These bits must always be written to '0'.

Bit 3:0 = **RTCP[19:16]:** *RTC Periodic High*
Reading RTCP_H register, the high part of current RTCP counter value is returned. To write this register it is required to enter into configuration mode (see RTOFF bit of RTCCR register).

14.4.1 Register map

RTC registers are mapped as 16 bit addressable registers as described in the table below:

Table 40. RTC Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	RTCCR	Reserved				GEN	PEREN	AEN	Res..	Reserved		RT OFF	CNF	GIR	PERIR	AIR	Res..
4	RTCCNT_L	CNT_L															
8	RTCCNT_M	CNT_M															
C	RTCCNT_H	CNTH															
10	RTCA_L	ALARM_L															
14	RTCA_M	ALARM_M															
18	RTCA_H	ALARM_H															
1C	RTCP_L	PERIOD_L															
20	RTCP_H	Reserved												PERIOD_H			

Refer to Table 21 on page 50 for the base address.

14.5 Programming considerations

RTC configuration procedure can be described by the following steps:

1. Reading RTOFF until its value is ‘1.
2. Writing CNF control bit at ‘1’.
3. Writing one or more RTC registers.
4. Writing CNF control bit at ‘0’, along with any other required RTCCR change.
5. Read back updated register for confirmation.

The writing operation actually takes effect only when CNF bit is written back to ‘0’ and it needs at least 2 cycles of CLK_AF to be terminated. During those 2 CLK_AF cycles, all RTC registers are updated simultaneously, so the sequence used to write them during the configuration phase is not relevant. There are no particular constraints about the total length of configuration procedure, as the actual update of the registers happens only when CNF bit is reset to ‘0’.

Note The configuration procedure described above should be executed by a routine located in internal RAM so to avoid any possible interaction between RTC register update and the board noise induced by external memory access, especially when the code is executed from external SDRAM memory. RTC register update sequence is quite sensitive to noise induced on the clock oscillator, and unexpected reset of RTOFF bit could occur if such interaction happens. In this case resetting RTC block by using $\overline{RTCRST}$ pin will restore normal RTC operations.

It is important to consider that once CNF bit is reset to ‘0’ and the actual update begins, none of the RTC registers can be modified, RTCCR included, until the update terminates (RTOFF becomes ‘1’ again). Since this period of time is quite long in terms of CPU clock periods (it is

at least one CLK_AF period, i.e. about 30 μs using 32 kHz crystal oscillator), it is important to consider it when RTC register update occurs within an interrupt response routine, since also interrupt pending bits, located in RTCCR, can be cleared only when RTOFF is '1'. As a consequence it is recommended to clear IRQ pending bits before or during the same configuration procedure used to update other RTC registers.

15 ASYNCHRONOUS AHB-APB BRIDGE (A3BRG)

15.1 Introduction

In order to reduce power consumption, it is possible to use different clock domains depending on the performances of each block. Typically, the core will run with a fast clock, whereas a slower clock can be sufficient for most peripherals. Nevertheless, in low power mode, the core can also be clocked by a slower clock in order just to manage with peripherals requests. The asynchronous AHB/APB bridge enables to connect completely asynchronous AHB with APB buses. It implements a synchronization stage that enables to re-synchronize the AHB with the APB.

The AHB/APB clocks ratio could be typically around 1/10. As a consequence, an APB read access will need more than 20 AHB cycle to be done.

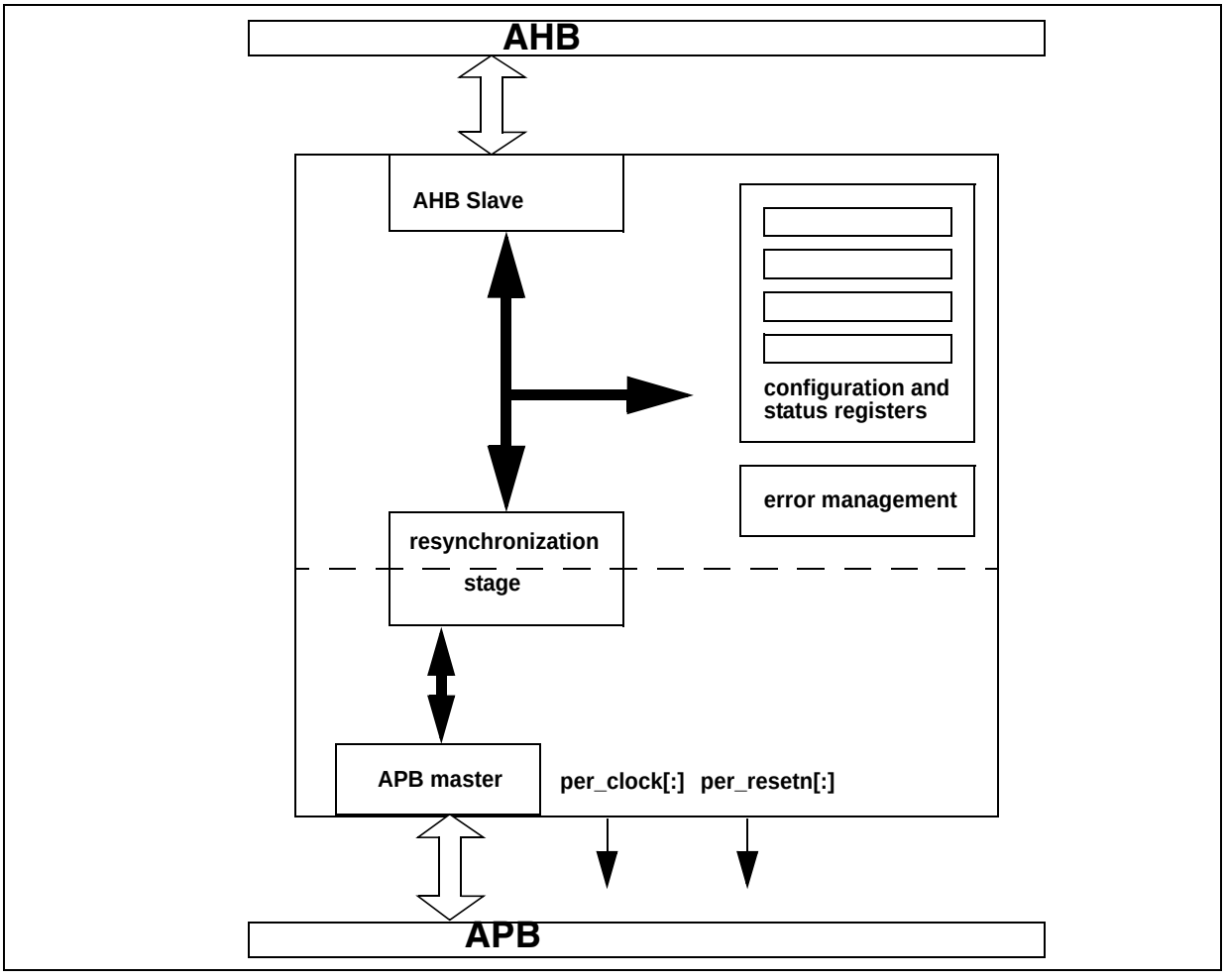
15.2 Main Features

- asynchronous AHB/APB clock domains
- access to up to 13 peripherals with 4 kBytes memory space each.
- Programmable registers for:
  - AHB/APB bridge status
  - transaction configuration
  - peripherals clock gating

15.3 Functional Description

The asynchronous AHB/APB bridge functionality can be illustrated by Figure 19 on page 161}

Figure 19. AHB/APB bridge block diagram



This AHB/APB bridge is capable of addressing 13 peripherals. Each peripheral is connected through an APB slave port to the APB bus and data size can be different from one peripheral to the other. The APB memory mapping provides 4 kBytes memory space for each peripheral.

The Bridge is able to detect some errors while accessing to a peripheral:

- an out of memory condition on the APB bus, when the address does not refer to a valid peripheral. This could arise if the set of all peripheral's memory spaces does not cover the whole AHB mapping space
- an access on the APB bus to a peripheral whose clock has been switched-off
- an access on the APB bus to a peripheral which is currently under reset.

If one of the first 3 conditions occurs, the bridge does not start an APB transaction and reports on the AHB bus an ERROR condition (if enabled). The type of error will be saved in the Bridge Status Register and the address which has generated the error condition will be reported in

the Peripheral Address Error Register. This should allow the user to retrieve the event which generated the error.

15.3.1 Peripherals clock gating

The AHB/APB bridge enables to configure the peripherals' clocks to be switched-on/off. The 32-bits Peripheral Clock Gating registers (PCGn) enables to switch-off/on any of the 13 peripherals directly. This enables to centralize the management of all the clock on this APB bus.

When a peripheral's clock is switched-off, any access to this peripheral will return an error on the AHB bus. The Bus Status Register will contain the error status, the Peripheral Address Error Register will provide the address who generated the error.

In order to help debugging real-time application, register EMU_PCGn has been introduced. The status of this register is used to force the gating of the corresponding clock line when the system enter into debug mode, based upon the following definition:
0 -> Peripheral clock switched off in debug mode.
1 -> Peripheral clock status depends on PCGn bit also in debug mode.

15.3.2 Peripherals reset control

The AHB/APB bridge has a unique reset pin that enables resetting the AHB/APB bridge as well as all peripherals. As the reset removal is aligned with AHB clock, the AHB/APB bridge provides a re-synchronization mechanism to generate the peripherals reset. The reset is asserted asynchronously and then released synchronously with the APB clock.

Moreover, the AHB/APB bridge enables to configure the peripherals' reset in order to reset independently any of the peripherals. The 32-bits Peripheral Reset Control registers (PRCn) enables to reset any of the 13 peripherals directly. This allows to centralize the management of all the reset on this APB bus.

When a peripheral is under reset, any access to this peripheral will return an error on the AHB bus. The Bus Status Register will contain the error status, the Peripheral Address Error Register will provide the address which generated the error.

15.4 Register description

The AHB/APB bridge provides a set of registers to configure peripherals clock gating, access control and monitor the AHB/APB bridge status. These registers are mapped as peripheral #0 in the AHB/APB bridge mapping.

Bridge Status Register (BSR)

Address Offset: 00h

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved							PUR	PCO	Reserved		OUTM	reserved			ERROR
r	r	r	r	r	r	r	rc	rc	-		rc	-			rc

This register stores the status of the AHB/APB allowing the application software to determine the reason of the last failed peripheral access which resulted in an ERROR response to the corresponding AHB transaction.

Bits 31:9 = Reserved. These bits should be always written to and read back as ‘0’.

Bit 8 = **PUR**: Peripheral Under Reset.
An access to a peripheral under reset has been attempted.

Bit 7 = **PCO**: Peripheral Clock Off.
An access to a not-clocked peripheral has been attempted.

Bit 6:5 = Reserved. These bits should be always written to and read back as ‘0’.

Bit 4 = **OUTM**: Out of memory.
An access out of memory has been attempted.

Bits 3:1 = Reserved. These bits must be always written to and read back as ‘0’.

Bit 0 = **ERROR**: Error.
a previous access has been aborted because it generates an error. The type of error is reported on bit 4 to 8.

Peripherals Address Error Register (PAER)

Address Offset: 08h

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved							nRW	Peripheral Address (23:16)							
-							r	r	r	r	r	r	r	r	r

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Peripheral Address (15:0)															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bit 31:25 = Reserved. These bits must be always written to and read back as ‘0’.

Bit 24= **nRW**: access type.
It returns the type of access that generate the error conditions: read(0) or write(1)

Bit 23:0 = **Peripheral Address**.
The Bridge is able to detect some errors while accessing to a peripherals. If an error occurs, the bridge ends the APB transaction and reports on the AHB bus an ERROR condition (if enabled). The address of slave generating the error condition is reported in the PERIPHERAL ADDRESS field. Depending of the size of the AHB addressing space, the MSB can be filled with 0.

Peripheral Clock Gating register 0 (PCG0)

Address Offset: 60h

Reset value: 0000 1003h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			1	P(xx)CO										1	1
-			r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r

Bits 31:13 = Reserved, always read as 0.

Bits 12:0 = **PxxCO:** Peripheral xx Clock Off with 0≤xx<12
When set, the PxxCO bit specifies the peripheral xx is clocked.
0 : Peripheral xx is gated off.
1 : Peripheral xx is clocked.

Note *Bits 0, 1 and 12 correspond to the AHB/APB bridge, the A-APB Global Control Register and the watchdog (WDG) respectively and are forced to 1 by hardware.*

Peripheral Under Reset register 0 (PUR0)

Address Offset: 80h

Reset value: 0000 1003h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			1	P(xx)CO										1	1
-			r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r

Bits 31:13 = Reserved, always read as 0.

Bit 12:0 = **PxxUR:** Peripheral xx Under Reset with 0≤xx<12

When set, the PxxUR bit specifies the peripheral xx is not under reset.

0 : Peripheral xx reset line is active.

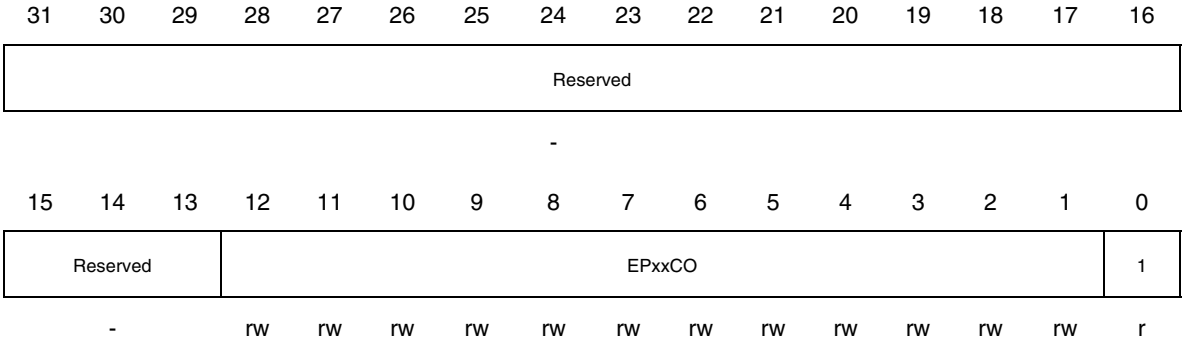
1 : Peripheral xx reset line is not active.

Note Bits 0, 1 and 12 correspond to the AHB/APB bridge, the A-APB Global Control Register and the watchdog (WDG) respectively and are forced to 1 by hardware.

Emulator Peripheral Clock Gating register 0 (EMU_PCG0)

Address Offset: A0h

Reset value: FFFF FFFFh



This register configures the status of clock gating for each of the peripherals controlled by the bridge, while the application program execution is stopped due to a debug request. Each peripheral can be configured to stop working as the user code reaches a breakpoint.

Bits 31:13 = Reserved, always read as 0.

Bits 12:0 = **EPxxCO:** Emulator Peripheral xx Clock Off with 0≤xx<12
0: Peripheral xx clock is gated off at a debug request, regardless of the state of the corresponding PxxCO bit in PCG0 register.
1: Peripheral xx clock gating status is unaffected by debug requests, and it is always determined by its corresponding PxxCO bit in PCG0 register.

Table 41 shows how the status of peripheral clock depends on PCG, EMU_PCG and ARM720 debug-mode:

Table 41. Clock Gating

PCG0	EMU_PCG0	ARM720 debug	Periph. clock
1	1	0	on
1	1	1	on
0	1	0	off
0	1	1	off
0	0	0	off
0	0	1	off
1	0	0	on
1	0	1	off

Note Bit EPCG0[0] corresponds to the AHB/APB bridge itself and is forced to 1 by hardware.

15.4.1 Register map

Table 42. AHB/APB bridge Register Map

Addr. Offset	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00	**BSR**	Reserved																							PUR	PCO	Reserved		OUTM	reserved			ERROR
04		Reserved																															
08	**PAER**	reserved							nRW	Peripheral Address (23:0)																							
0C - 5C		Reserved																															
60	**PCG0**	Reserved																						1	PxxCO							1	1
64 - 7C		Reserved																															
80	**PUR0**																							1	PxxUR							1	1
84 - 9C		Reserved																															
A0	**EMU_PCG0**	Reserved																							EPxxCO								1

Refer to Table 20 on page 49 for the base address.

16 UART

16.1 Introduction

The UART interface, also referred to as the Asynchronous Serial Controller (ASC), provides serial communication between STR720 and other microcontrollers, microprocessors or external peripherals.

The ASC supports full-duplex asynchronous communication. Eight or nine bit data transfer, parity generation, and the number of stop bits are programmable. Parity, framing, and overrun error detection are provided to increase the reliability of data transfers. Transmission and reception of data can simply be double-buffered, or 16-deep fifos may be used. For multiprocessor communications, a mechanism to distinguish the address from the data bytes is included. Testing is supported by a loop-back option. A 16-bit baud rate generator provides the ASC with a separate serial clock signal.

16.2 Main Features

- Full-duplex asynchronous communication
- Two internal FIFOs (16 words deep) for transmit and receive data
- 16-bit baud rate generator
- Data frames both 8 and 9 bit long
- Parity bit (even or odd) and stop bit
- Multiprocessor communication support

16.3 Functional Description

The ASC supports full-duplex asynchronous communication, where both the transmitter and the receiver use the same data frame format and the same baud rate. Data is transmitted on the **TXD** pin and received on the **RXD** pin.

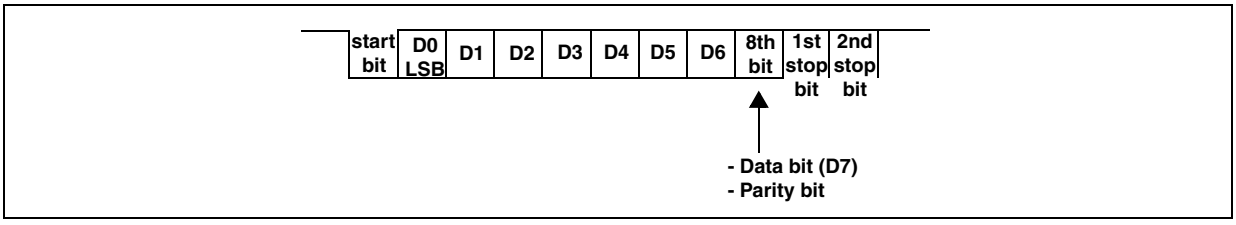
16.3.1 Data frames

Eight-bit data frames (see Figure 20) either consist of:

- eight data bits **D0-7** (by setting the **Mode** bit field to 001);
- seven data bits **D0-6** plus an automatically generated parity bit (by setting the **Mode** bit field to 011).

Parity may be odd or even, depending on the **ParityOdd** bit in the **ASCControl** register. An even parity bit will be set, if the modulo-2-sum of the seven data bits is 1. An odd parity bit will be cleared in this case.}

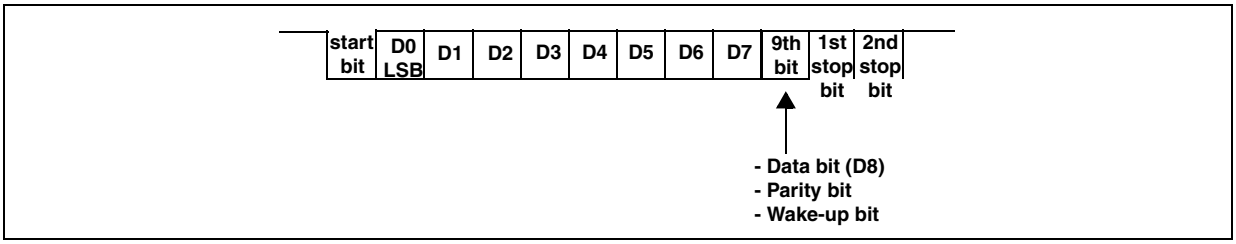
Figure 20. 8-bit data frames



Nine bit data frames (see Figure 21) either consist of:

- nine data bits **D0-8** (by setting the **Mode** bit field to 100);
- eight data bits **D0-7** plus an automatically generated parity bit (by setting the **Mode** bit field to 111);
- eight data bits **D0-7** plus a wake-up bit (by setting the **Mode** bit field to 101).

Figure 21. 9-bit data frames



Parity may be odd or even, depending on the **ParityOdd** bit in the **ASCControl** register. An even parity bit will be set, if the modulo-2-sum of the eight data bits is 1. An odd parity bit will be cleared in this case.

In wake-up mode, received frames are only transferred to the receive buffer register if the ninth bit (the wake-up bit) is 1. If this bit is 0, no receive interrupt request will be activated and no data will be transferred.

This feature may be used to control communication in multi-processor systems. When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the additional ninth bit is a 1 for an address byte and a 0 for a data byte, so no slave will be interrupted by a data byte. An address byte will interrupt all slaves (operating in 8-bit data + wake-up bit mode), so each slave can examine the 8 least significant bits (LSBs) of the received character (the address). The addressed slave will switch to 9-bit data mode, which enables it to receive the data bytes that will be coming (with the wake-up bit cleared). The slaves that are not being addressed remain in 8-bit data + wake-up bit mode, ignoring the following data bytes.

16.3.2 Transmission

Values to be transmitted are written to the transmit fifo, txfifo, by writing to **ASCTxBuffe**r. The txfifo is implemented as a 16 deep array of 9 bit vectors.

If the fifos are enabled (the **ASCControl(FifoEnable)** is set), the txfifo is considered full **(ASCStatus(TxFull)** is set) when it contains 16 characters. Further writes to **ASCTxBuffer** in this situation will fail to overwrite the most recent entry in the txfifo. If the fifos are disabled, the txfifo is considered full **(ASCStatus(TxFull)** is set) when it contains 1 character, and a write to **ASCTxBuffer** in this situation will overwrite the contents.

If the fifos are enabled, **ASCStatus(TxHalfEmpty)** is set when the txfifo contains 8 or fewer characters. If the fifos are disabled, it's set when the txfifo is empty.

Writing anything to **ASCTxReset** empties the txfifo.

Values are shifted out of the bottom of the txfifo into a 9-bit txshift register in order to be transmitted. If the transmitter is idle (the txshift register is empty) and something is written to the **ASCTx-Buffer** so that the txfifo becomes non-empty, the txshift register is immediately loaded from the txfifo and transmission of the data in the txshift register begins at the next baud rate tick.

At the time the transmitter is just about to transmit the stop bits, then if the txfifo is non-empty, the txshift register will be immediately loaded from the txfifo, and transmission of this new data will begin as soon as the current stop bit period is over (i.e. the next start bit will be transmitted immediately following the current stop bit period). Thus back-to-back transmission of data can take place. If instead the txfifo is empty at this point, then the txshift register will become empty. **ASC-Status(TxEmpty)** indicates whether the txshift register is empty.

After changing the fifoenable bit, it is important to reset the fifo to empty (by writing to the **ASCTxReset** register), since the state of the fifo pointer may be garbage.

The loop-back option (selected by the **ASCControl(LoopBack)** bit) internally connects the output of the transmitter shift register to the input of the receiver shift register. This may be used to test serial communication routines at an early stage without having to provide an external network.

16.3.3 Reception

Reception is initiated by a falling edge on the data input pin **(RXD)**, provided that the **ASCControl(Run)** and **ASCControl(RxEnable)** bits are set. The **RXD** pin is sampled at 16 times the rate of the selected baud rate. A majority decision of the first, second and third samples of the start bit determines the effective bit value. This avoids erroneous results that may be caused by noise.

If the detected value is not a 0 when the start bit is sampled, the receive circuit is reset and waits for the next falling edge transition at the **RXD** pin. If the start bit is valid, the receive circuit continues sampling and shifts the incoming data frame into the receive shift register. For subsequent data and parity bits, the majority decision of the seventh, eighth and ninth samples in each bit time is used to determine the effective bit value.

Note If reception is initiated when data input pin (RXD) is being stretched at '0' a frame error indication is reported, since reception stage samples the initial value as a falling edge.

For 0.5 stop bits, the majority decision of the third, fourth, and fifth samples during the stop bit is used to determine the effective stop bit value.

For 1 and 2 stop bits, the majority decision of the seventh, eighth, and ninth samples during the stop bits is used to determine the effective stop bit values.

For 1.5 stop bits, the majority decision of the fifteenth, sixteenth, and seventeenth samples during the stop bits is used to determine the effective stop bit value.

The effective values received on the **RXD** pin are shifted into a 10-bit rxshift register.

The receive fifo, rxfifo, is implemented as a 16 deep array of 10-bit vectors (each 9 down to 0). If the rxfifo is empty, **ASCstatus(RxBufNotEmpty)** is set to '0'. If the rxfifo is not empty, a read from **ASCRx-Buffer** will get the oldest entry in the rxfifo. If fifos are disabled, the rxfifo is considered full when it contains one character. **ASCStatus(RxHalfFull)** is set when the rxfifo contains more than 8 characters. Writing anything to **ASCRxReset** empties the rxfifo.

As soon as the effective value of the last stop bit has been determined, the content of the rxshift register is transferred to the rxfifo (unless we're in wake-up mode, in which case this happens only if the wake-up bit, bit8, is a '1'). The receive circuit then waits for the next start bit (falling edge transition) at the **RXD** pin.

ASCStatus(OverrunError) is set when the rxfifo is full and a character is loaded from the rxshift register into the rxfifo. It is cleared when the **ASCRxBuffer** register is read.

The most significant bit of each rxfifo entry (rxfifo[x][9]) records whether or not there was a frame error when that entry was received (i.e. one of the effective stop bit values was '0'). **ASCStatus(FrameError)** is set when at least one of the valid entries in the rxfifo has its MSB set.

If the mode is one where a parity bit is expected, then the bit rxfifo[x][8] (if 8 bit data + parity mode is selected) or the bit rxfifo[x][7] (if 7 bit data + parity mode is selected) records whether there was a parity error when that entry was received. Note, it does not contain the parity bit that was received. **ASCStatus(ParityError)** is set when at least one of the valid entries in the rxfifo has bit 8 set (if 8 bit data + parity mode is selected) or bit 7 set (if 7 bit data + parity mode is selected).

After changing the fifoenable bit, it is important to reset the fifo to empty (by writing to the **ASCRxReset** register), since the state of the fifo pointers may be garbage.

Reception is stopped by clearing the **ASCControl(RxEnable)** bit. A currently received frame is completed including the generation of the receive status flags. Start bits that follow this frame will not be recognized.

16.3.4 Timeout mechanism

The ASC contains an 8-bit timeout counter. This reloads from **ASCTimeout** whenever one or more of the following is true

- **ASCRxBuffer** is read
- The ASCstarts to receive a character
- **ASCTimeout** is written to

If none of these conditions hold, the counter decrements towards 0 at every baud rate tick.

ASCStatus(TimeoutNotEmpty) is ’1’ exactly whenever the rxfifo is not empty and the timeout counter is zero.

ASCStatus(TimeoutIdle) is ‘1’ exactly whenever the rxfifo is empty and the timeout counter is zero.

The effect of this is that whenever the rxfifo has got something in it, the timeout counter will decrement until something happens to the rxfifo. If nothing happens, and the timeout counter reaches zero, the **ASCStatus(TimeoutNotEmpty)** flag will be set.

When the software has emptied the rxfifo, the timeout counter will reset and start decrementing. If no more characters arrive, when the counter reaches zero the **ASCStatus(TimeoutIdle)** flag will be set.

16.3.5 Baud rate generation

The baud rate generator provides a clock at 16 times the baud rate, called the oversampling clock. This clock only ticks if **ASCControl(Run)** is set to'1'. Setting this bit to 0 will immediately freeze the state of the ASCs transmitter and receiver. This should only be done when the ASC is idle.

The baud rate and the required reload value for a given baud rate can be determined by the following formulae:

Baudrate = f_{APB} / (16 * <ASCBaudRate>)

<ASCBaudRate> = f_{APB} / (16 * Baudrate)

where: <ASCBaudRate> represents the content of the **ASCBaudRate** register, taken as unsigned 16-bit integer, and f_{APB} is the clock frequency of the APB sub-system to which UART block is connected.

The following tables list various commonly used baud rates together with the required reload values and the deviation errors for various different APB clock frequencies..

Table 43. Baud rates with f_{APB} = 16 MHz

Baud rate	Reload value (exact)	Reload value (integer)	Reload value (hex)	Deviation error
625K	1.6	2	0002	20%
38.4K	26.042	26	001A	0.160%
19.2K	52.083	52	0034	0.160%
9600	104.167	104	0068	0.160%
4800	208.333	208	00D0	0.160%
2400	416.667	417	01A1	0.080%
1200	833.333	833	0341	0.040%
600	1666.667	1667	0683	0.020%
300	3333.333	3333	0D05	0.010%
75	13333.333	13333	3415	0.003%

Table 44. Baud rates with f_{APB} = 20 MHz

Baud rate	Reload value (exact)	Reload value (integer)	Reload value (hex)	Deviation error
625K	2	2	0002	0%
38.4K	32.552	33	0021	1.358%
19.2K	65.104	65	0041	0.160%
9600	130.208	130	0082	0.160%
4800	260.417	260	0104	0.160%
2400	520.833	521	0209	0.032%
1200	1041.667	1042	0412	0.032%
600	2083.333	2083	0823	0.016%

Table 44. Baud rates with f_{APB} = 20 MHz

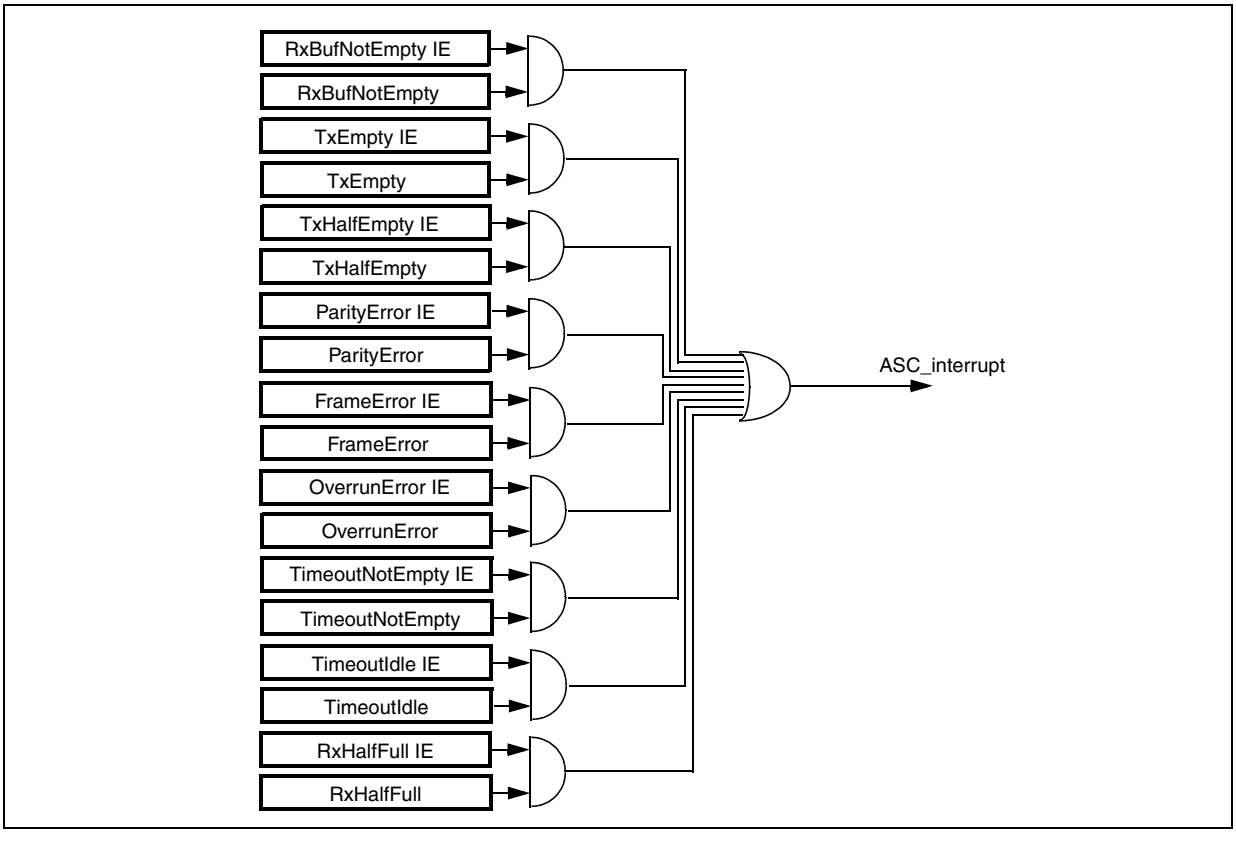
Baud rate	Reload value (exact)	Reload value (integer)	Reload value (hex)	Deviation error
300	4166.667	4167	1047	0.008%
75	16666.667	16667	411B	0.002%

16.3.6 Interrupt control

The ASC has a single interrupt coming out of it, called **ASC_interrupt**. The status bits in the **ASC-Status** register determine the cause of the interrupt. **ASC_interrupt** will go high when a status bit is 1 (high) and the corresponding bit in the **ASCIntEnable** register is 1 (see Figure 22).

Note: The ASCStatus register is read only. The Status bits can only be cleared by operating on the FIFOs. The RxFIFO and TxFIFO can be reset by writing to the ASCRxReset and ASCTxReset registers.

Figure 22. UART interrupt request



16.3.7 Using the ASC interrupts when fifos are disabled

When fifos are disabled, the ASC provides three interrupt requests to control data exchange via the serial channel:

- **TxHalfEmpty** is activated when data is moved from **ASCTxBuffer** to the txshift register.
- **TxEmpty** is activated before the stop bit is transmitted.
- **RxBufNotEmpty** is activated when the received frame is moved to **ASCRxBuffe**r.

For single transfers it is sufficient to use the transmitter interrupt **(TxEmpty)**, which indicates that the previously loaded data has been transmitted, except for the stop bit.

For multiple back-to-back transfers using **TxEmpty** would leave just one stop bit time for the handler to respond to the interrupt and initiate another transmission. Using the transmit buffer interrupt **(TxHalfEmpty)** to reload transmit data allows the time to transmit a complete frame for the service routine, as **ASCTxBuffer** may be reloaded while the previous data is still being transmitted.

TxHalfEmpty is an early trigger for the reload routine, while **TxEmpty** indicates the completed transmission of the data field of the frame. Therefore, software using handshake should rely on **TxEmpty** at the end of a data block to make sure that all data has really been transmitted.

16.3.8 Using the ASC interrupts when fifos are enabled

To transmit a large number of characters back to back, the driver routine would write 16 characters to **ASCTxBuffer**, then every time a **TxHalfEmpty** interrupt fired, it would write 8 more. When it had nothing more to send, a **TxEmpty** interrupt would tell it when everything has been transmitted.

When receiving, the driver could use **RxBufNotEmpty** to interrupt every time a character came in. Alternatively, if data is coming in back-to-back, it could use **RxHalfFull** to interrupt it when there was at least 8 characters in the rxfifo to read. It would have as long as it takes to receive 8 characters to respond to this interrupt before data would overrun. If less than eight character streamed in, and no more were received for at least a timeout period, the driver could be woken up by one of the two timeout interrupts, **TimeoutNotEmpty** or **TimeoutIdle**.

16.4 Register description

ASCBaudRate

Address Offset: 00h

Reset value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ASCBaudRate															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The **ASCBaudRate** register is the dual-function baud rate generator/reload register.

A read from this register returns the content of the timer, writing to it updates the reload register.

An auto-reload of the timer with the content of the reload register is performed each time the **ASCBaudRate** register is written to. However, if the **Run** bit of the **ASCControl** register is 0 at the time the write operation to the **ASCBaudRate** register is performed, the timer will not be reloaded until the first CPU clock cycle after the **Run** bit is 1.

Bit 15:0 = **ASCBaudRate**
Write function: 16-bit reload value
Read function: 16-bit count value

ASCTxBuffer

Address Offset: 04h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED							TX[8]	TX[7]	TX[6]	TX[5]	TX[4]	TX[3]	TX[2]	TX[1]	TX[0]
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Writing to the transmit buffer register starts data transmission.

Bit 15:9 = *Reserved.*
Write 0

Bit 8 = **TX[8]**: *Transmit buffer data D8.*
Transmit buffer data D8, or parity bit, or wake-up bit or undefined - dependent on the operating mode (the setting of the **Mode** field in **ASCControl** register).

*Note If the **Mode** field selects an 8 bit frame then this bit should be written as 0.*

*Note If the **Mode** field selects a frame with parity bit, then the TX[8] bit will contain the parity bit (automatically generated by the UART). A user writing at '0' or '1' of such a bit will have no effect on the transmitted frame.*

Bit 7 = **TX[7]**: *Transmit buffer data D7*.
Transmit buffer data D7 or parity bit - dependent on the operating mode (the setting of the **Mode** field in **ASCControl** register).

*Note If the **Mode** field selects a frame with parity bit, then the TX[7] bit will contain the parity bit (automatically generated by the UART). A user writing at '0' or '1' of such a bit will have no effect on the transmitted frame.*

Bit 6:0 = **TX[6:0]**: *Transmit buffer data D(6:0)*

ASCRxBuffer

Address Offset: 08h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED						RX[9]	RX[8]	RX[7]	RX[6]	RX[5]	RX[4]	RX[3]	RX[2]	RX[1]	RX[0]
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

The received data and, if provided by the selected operating mode, the received parity bit can be read from the receive buffer register.

Bit 15:10 = *Reserved*.
Will read back 0

Bit 9 = **RX[9]**: *Frame error*.
If set, it indicates a frame error occurred on data stored in RX[8:0] (i.e. one of the effective stop bit values was '0' when the data was received).

Bit 8 = **RX[8]**: *Receive buffer data D8.*
Receive buffer data D8, or parity error, or wake-up bit - dependent on the operating mode (the setting of the **Mode** field in the **ASCControl** register).

Note *If the **Mode** field selects a 7- or 8-bit frame then this bit is undefined. Software should ignore this bit when reading 7- or 8-bit frames.*

Bit 7 = **RX[7]**: *Receive buffer data D7.*
Receive buffer data D7, or parity error - dependent on the operating mode (the setting of the **Mode** field in the **ASCControl** register).

Bit 6:0 = **RX[6:0]**: *Receive buffer data D(6:0).*

ASCControl

Address Offset: 0Ch

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED					FifoEnable	RESERVED	RxEnable	Run	LoopBack	ParityOdd	Stop Bits		Mode		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This register controls the operating mode of the ASC and contains control bits for mode and error check selection, and status flags for error identification.

*Note Programming the mode control field **(Mode)** to one of the reserved combinations may result in unpredictable behavior.*

*Note Serial data transmission or reception is only possible when the baud rate generator run bit **(Run)** is set to 1. When the **Run** bit is set to 0, **TXD** will be 1. Setting the **Run** bit to 0 will immediately freeze the state of the transmitter and receiver. This should only be done when the ASC is idle.*

Bit 15:11= *Reserved*.
Write 0 will read back 0

Bit 10 = **FifoEnable**: *FIFO Enable*
0: FIFO mode disabled
1: FIFO mode enabled

Bit 9 = *Reserved*.
Write 0 will read back 0

Bit 8 = **RxEnable:** *Receiver Enable*
0: Receiver disabled
1: Receiver enabled

Bit 7 = **Run:** *Baudrate generator Run bit*
0: Baud rate generator disabled (ASC inactive)
1: Baud rate generator enabled

Bit 6 = **LoopBack:** *LoopBack mode enable*
0: Standard transmit/receive mode
1: Loopback mode enabled

Note This bit may be modified only when the ASC is inactive.

Bit 5 = **ParityOdd:** *Parity selection*
0: Even parity (parity bit set on odd number of ‘1’s in data)
1: Odd parity (parity bit set on even number of ‘1’s in data)

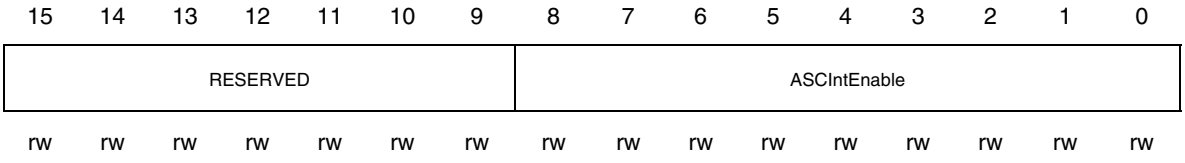
Bit 4:3 = **Stop Bits:** *Number of stop bits selection*
These bits select the number of stop bits
00: 0.5 stop bits
01: 1 stop bit
10: 1.5 stop bits
11: 2 stop bits

Bit 2:0 = **Mode:** *ASC Mode control*
000: reserved
001: 8 bit data
010: reserved
011: 7 bit data + parity
100: 9 bit data
101: 8 bit data + wake up bit
110: reserved
111: 8 bit data + parity

ASCIntEnable

Address Offset: 10h

Reset value 0000h



The **ASCIntEnable** register enables a source of interrupt.

Interrupts will occur when a status bit in the **ASCStatus** register is 1, and the corresponding bit in the **ASCIntEnable** register is 1.

Bit 15:9 = *Reserved*.
Write 0, will read back 0.

Bit 8 = **ASCIntEnable[8]= RxHalfFullIE**: *Receiver buffer Half Full Interrupt Enable*
0: interrupt disabled.
1: interrupt enabled.

Bit 7 = **ASCIntEnable[7]=TimeoutIdleIE**: *Timeout Idle Interrupt Enable*
0: interrupt disabled.
1: interrupt enabled.

Bit 6 = **ASCIntEnable[6]=TimeoutNotEmpty IE**: *Timeout Not Empty Interrupt Enable*
0: interrupt disabled.
1: interrupt enabled.

Bit 5 = **ASCIntEnable[5]=OverrunErrorIE**: *Overrun Error Interrupt Enable*
0: interrupt disabled.
1: interrupt enabled.

Bit 4 = **ASCIntEnable[4]=FrameErrorIE**: *Framing Error Interrupt Enable*
0: interrupt disabled.
1: interrupt enabled.

Bit 3 = **ASCIntEnable[3]=ParityErrorIE:** *Parity Error Interrupt Enable*
0: interrupt disabled.
1: interrupt enabled.

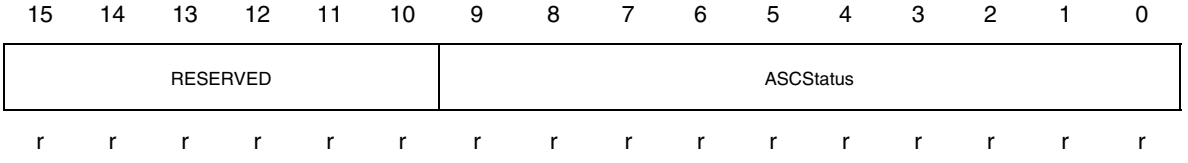
Bit 2 = **ASCIntEnable[2]=TxHalfEmptyIE:** *Transmitter buffer Half Empty Interrupt Enable*
0: interrupt disabled.
1: interrupt enabled.

Bit 1 = **ASCIntEnable[1]=TxEmptyIE:** *Transmitter Empty Interrupt Enable*
0: interrupt disabled.
1: interrupt enabled.

Bit 0 = **ASCIntEnable[0]=RxBufNotEmptyIE:** *Receiver Buffer Not Empty Interrupt Enable*
0: interrupt disabled.
1: interrupt enabled.

ASCStatus

Address Offset: 14h
Reset value: 0006h



The **ASCStatus** register determines the cause of an interrupt.

Bit 15:10 = *Reserved*.
Read back 0.

Bit 9 = **ASCStatus[9]:** *TxFull.*
Set when the txfifo contains 16 characters.

Bit 8 = **ASCStatus[8]:** *RxHalfFull*.
Set when the rxfifo contains at least 8 characters.

Bit 7 = **ASCStatus[7]:** *TimeoutIdle.*
Set when there's a timeout and the rxfifo is empty

Bit 6 = **ASCStatus[6]:** *TimeoutNotEmpty*.
Set when there's a timeout and the rxfifo is not empty

Bit 5 = **ASCStatus[5]:** *OverrunError*.
Set when data is received and the rxfifo is full.

Bit 4 = **ASCStatus[4]:** *FrameError*.
Set when the rxfifo contains something received with a frame error

Bit 3 = **ASCStatus[3]:** *ParityError.*
Set when the rxfifo contains something received with a parity error

Bit 2 = **ASCStatus[2]:** *TxHalfEmpty.*
Set when txfifo at least half empty

Bit 1 = **ASCStatus[1]:** *TxEmpty*.
Set when transmit shift register is empty

Bit 0 = **ASCStatus[0]:** *RxBufNotEmpty.*
Set when rxfifo not empty (rxfifo contains at least one entry)

ASCTimeout

Address Offset: 1Ch

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED								ASCTimeout							
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This register is to have a timeout system to be sure that not too much time pass between two successive received characters.

Bit 15:8 = *Reserved*.
Write 0, will read back 0.

Bit 7:0 = **ASCTimeout:** *Timeout.*
Timeout period in baud rate ticks.

ASCTxReset

Address Offset: 20h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

A write to this register empties the txfifo.

ASCRxReset

Address Offset: 24h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED															
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

A write to this register empties the rxfifo.

16.4.1 Register map

The following table summarizes the registers implemented in the UART macrocell.

Table 45. UART Peripheral Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**ASC Baud Rate**	ASCBaudRate															
4	**ASC TxBuffer**	RESERVED							ASCTxBuffer								
8	**ASC RxBuffer**	RESERVED							ASCRxBuffer								
C	**ASC Control**	RESERVED					FifoEn-able	RE-SERVED	RxEn-able	Run	Loop-Back	Parity-Odd	Stop Bits		Mode		
10	**ASC IntEnable**	RESERVED							ASCIntEnable								
14	**ASC Status**	RESERVED							ASCStatus								
1C	**ASC Timeout**	RESERVED								ASCTimeout							
20	**ASC TxReset**	ASCTxReset															
24	**ASC RxReset**	ASCRxReset															

Refer to Table 20 on page 49 for the base address.

17 BUFFERED SPI (BSPI)

17.1 Introduction

The BSPI block is a standard 4-pin Serial Peripheral Interface for serial control communication. It interfaces on one side to the SPI bus and on the other has a standard register data and interrupt interface.

The BSPI contains two 16-word x 16-bit FIFO's one for receive and the other for transmit. The BSPI can directly operate with words 8 and 16 bit long and can generates interrupts or DMA requests separately for receive and transmit events.

17.2 Main Features

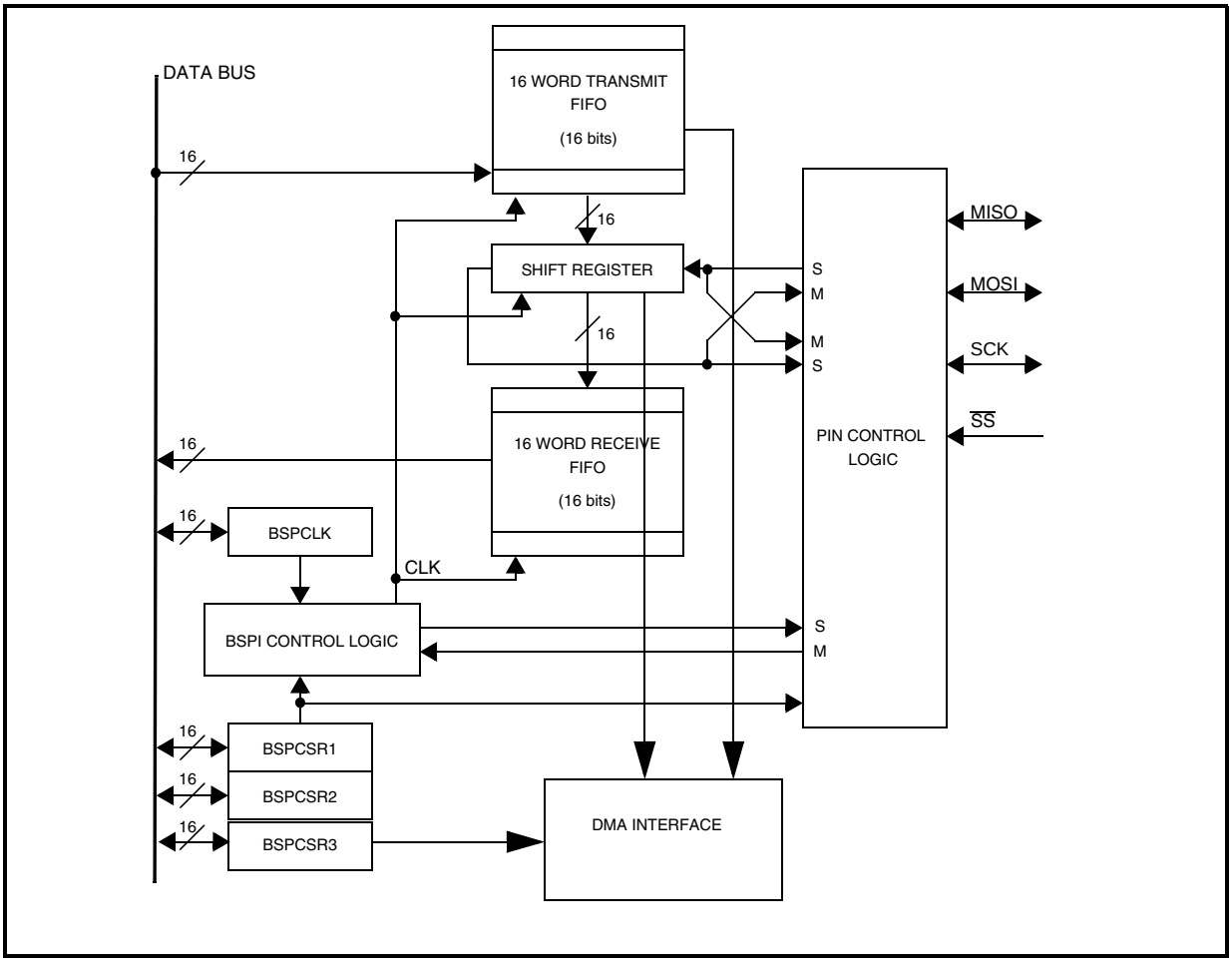
- Programmable depth receive FIFO.
- Maximum 16 word Receive FIFO.
- Programmable depth transmit FIFO.
- Maximum 16 word Transmit FIFO.
- Master and Slave modes supported.
- Internal clock prescaler.
- Programmable DMA interface

17.3 Functional Description

The processor views the BSPI as a memory mapped peripheral, which may be used by standard polling, interrupt programming techniques or DMA controlled access. Memory-mapping means processor communication can be achieved using standard instructions and addressing modes.

When an SPI transfer occurs data is transmitted and received simultaneously. A serial clock line synchronizes shifting and sampling of the information on the two serial data lines. A slave select line allows individual selection of a slave device. The central elements in the BSPI system are the 16-bit shift register and the read data buffer which is 16 words x 16-bit.A BSPI-DMA interface is also present to allow for data to be transferred to/from memory using the DMA.A block diagram of the BSPI is shown in Figure 23 on page 187.

Figure 23. BSPI Block Diagram



17.3.1 BSPI Pin Description

The BSPI is a four wire, bi-directional bus. The data path is determined by the mode of operation selected. A master and a slave mode are provided together with the associated pad control signals to control pad direction. These pins are described in Table 46 on page 187.

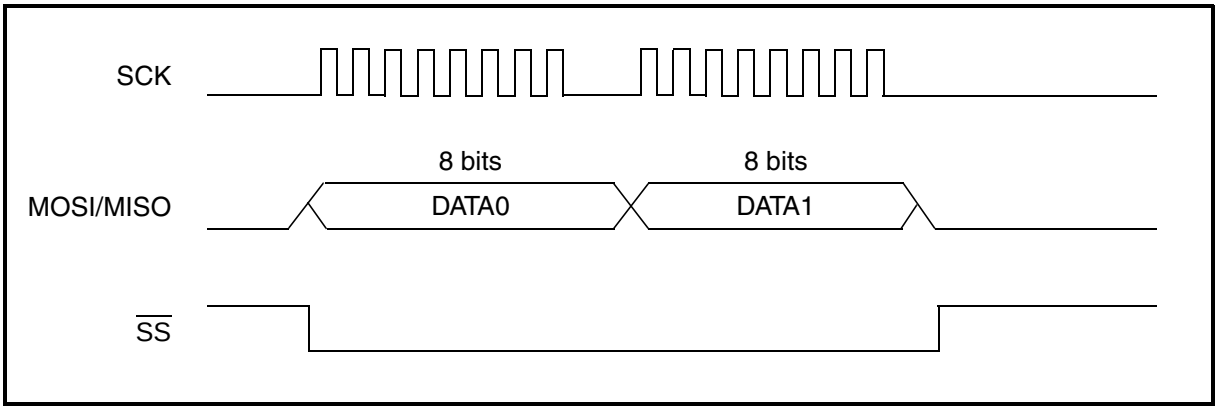
Table 46. BSPI pins

Pin Name	Description
SCK	The bit clock for all data transfers. When the BSPI is a master the SCK is output from the chip. When configured as a slave the SCK is input from the external source.
MISO	Master Input/Slave Output serial data line.
MOSI	Master Output/Slave Input serial data line.
$\overline{SS}$	Slave Select. The $\overline{SS}$ input pin is used to select a slave device. Must be pulled low after the SCK is stable and held low for the duration of the data transfer. The $\overline{SS}$ on the master must be deasserted high.This signal can be masked when in master mode-see register description of CSR Reg3 bit 0

17.3.2 BSPI Operation

During a BSPI transfer (see Figure 24 on page 188), data is shifted out and shifted in (transmitted and received) simultaneously. The SCK line synchronizes the shifting and sampling of the information. It is an output when the BSPI is configured as a master and an input when the BSPI is configured as a slave. Selection of an individual slave BSPI device is performed on the slave select line and slave devices that are not selected do not interfere with the BSPI buses.

Figure 24. BSPI Bus Transfer



The CPOL (clock polarity) and CPHA (clock phase) bits of the BSPCSR1 are used to select any of the four combinations of serial clock (see Figure 25 on page 189, Figure 26 on page 189, Figure 27 on page 190 and Figure 28 on page 190). These bits must be the same for both the master and slave BSPI devices. The clock polarity bit selects either an active high or active low clock but does not affect transfer format. The clock phase bit selects the transfer format.

There is a 16-bit shift register which interfaces directly to the BSPI bus lines. As transmit data goes out from the register, received data fills the register.

Note When BSPI cell is configured in Slave mode, SCLK_IN clock must be divided by a factor of 8 or more compared with the configured peripheral clock (APB clock).

In case a clock polarity change is required when BSPI macrocell is configured in master mode, the following sequence should be followed in order to avoid spurious clock edges generation:

1. Write ‘0’ into both MSTR and BSPE bit in BSPCSR1 register, so to safely disable BSPI macrocell.
2. Change CPOL and CPHA bits in BSPCSR1 register according to the required polarity setting.
3. Write ‘1’ into BSPE bit of BSPCSR1 register to enable again BSPI macrocell.
4. Write ‘1’ into MSTR bit of BSPCSR1 register to configure the master mode again.

It must be noticed that the above sequence should be executed as reported, without merging in a single instruction any of the steps.

Figure 25. BSPI Clocking Scheme (CPOL=0, CPHA=0)

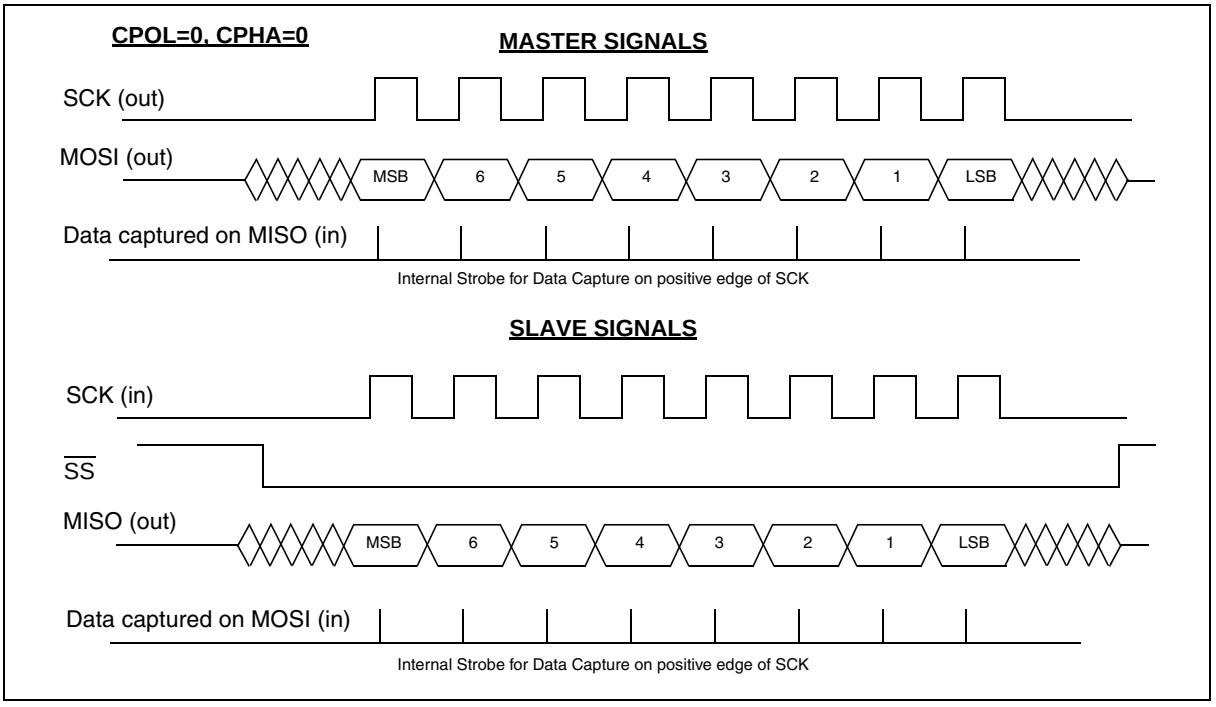


Figure 26. BSPI Clocking Scheme (CPOL=0, CPHA=1)

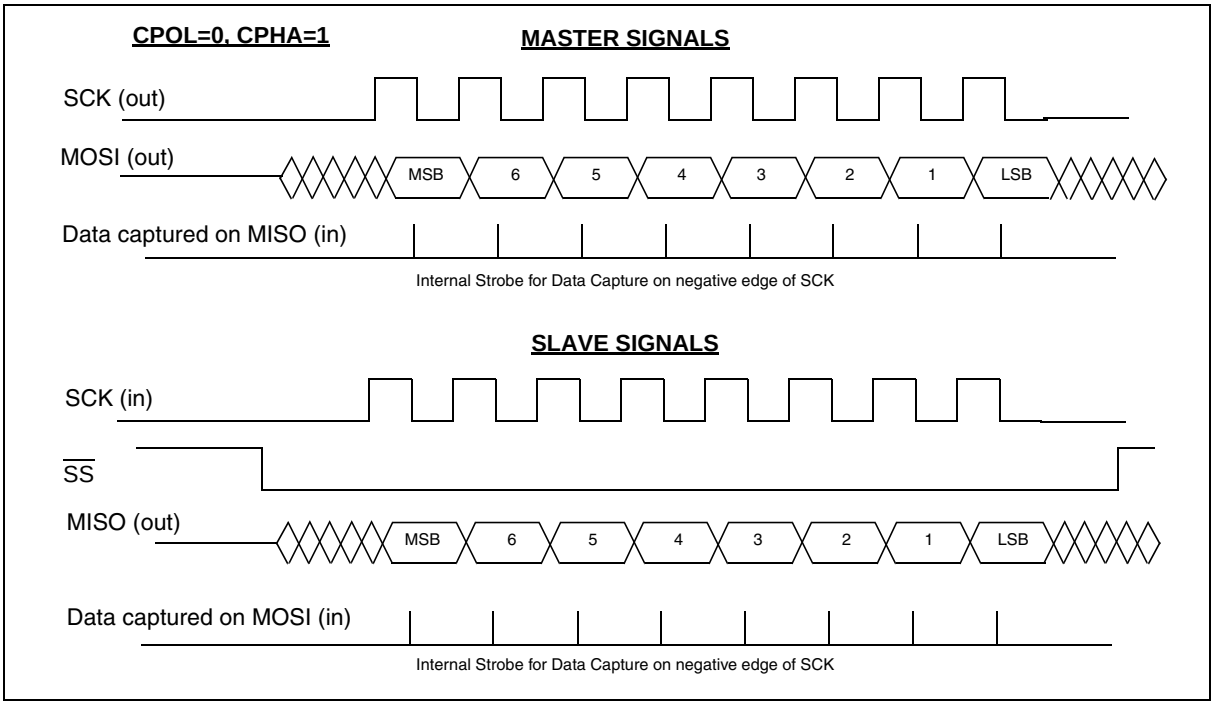


Figure 27. BSPI Clocking Scheme (CPOL=1, CPHA=0)

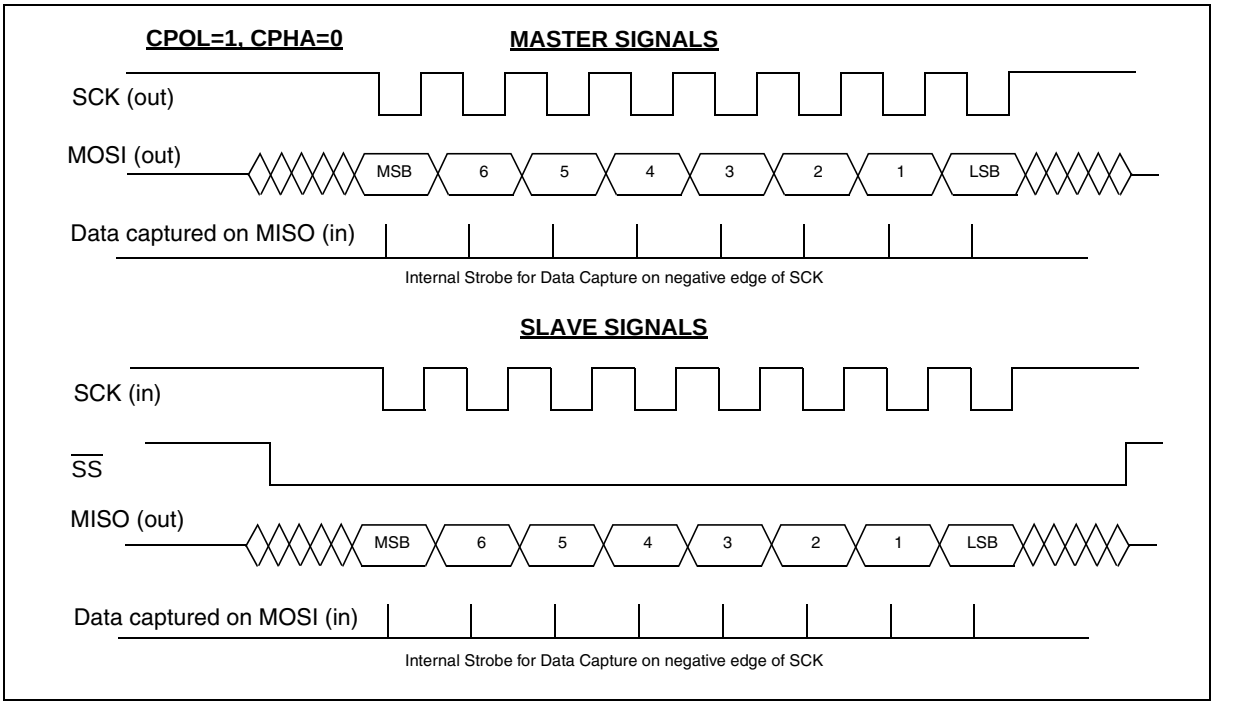
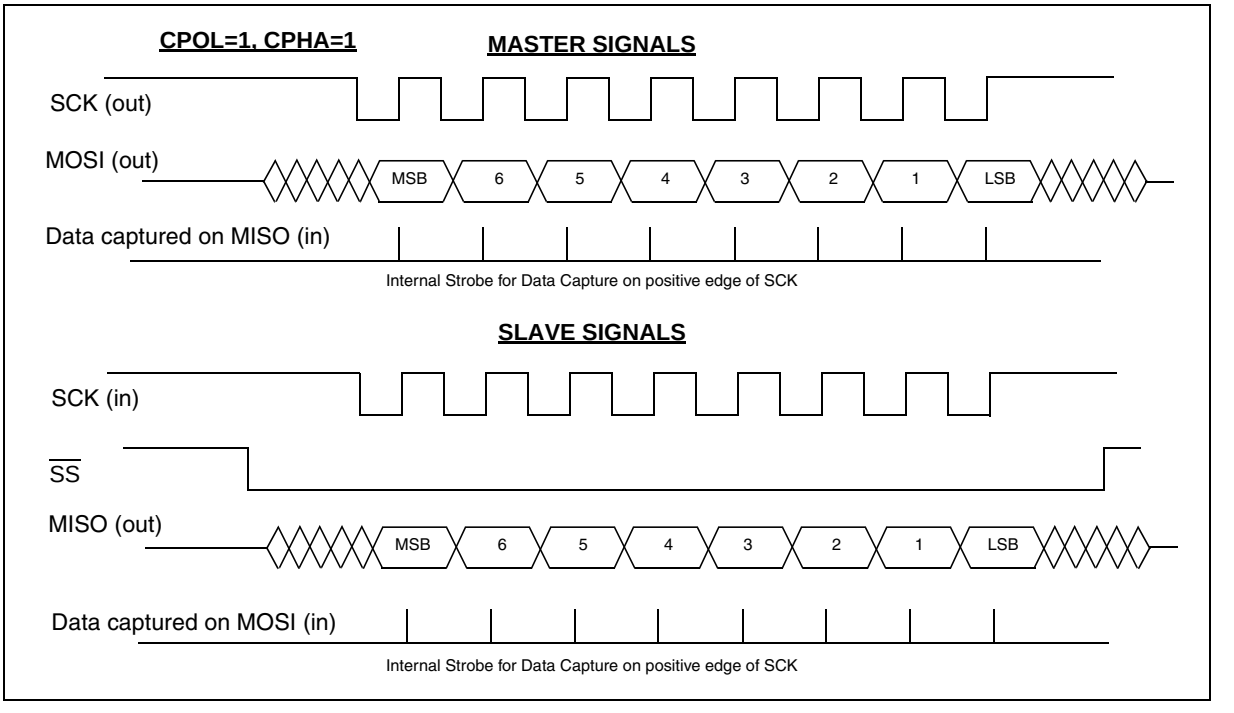


Figure 28. BSPI Clocking Scheme (CPOL=1, CPHA=1)



17.3.3 Transmit FIFO

The transmit FIFO consists of a 16 by 16 bit register bank which can operate in 8/16 bit modes as configured by the word length (WL[1:0]) control bits of BSPCSR1. Data is left justified indicating that only the most significant portion of the word is transmitted if using 8 bit mode. After a transmission is completed the next data word is loaded from the transmit FIFO.

The user can set the depth of the FIFO from the default one location up to a maximum of sixteen locations. This can be set dynamically but will only take effect after the completion of the current transmission. Status flags report if the FIFO is full (TFF), the FIFO is not empty (TFNE), the FIFO is empty (TFE) and the transmit buffer has under flown (TUFL), this last one being generated only when macrocell is configured in master mode. The transmit interrupt enable (TIE[1:0]) control bits of BSPCSR2 determine the source of the transmit interrupt. If the interrupt source is enabled then an active high interrupt will be asserted to the processor.

If BSPI is configured in master mode and the TUFL flag is asserted then a subsequent write to the transmit FIFO will clear the flag. If interrupts are enabled the interrupt will be de-asserted. Regardless of the selected BSPI mode, TFF and TFNE flags are updated at the end of the processor write cycle and at the end of each transmission.

Note Data should be written in the fifo only if the macro is enabled (see BSPI System Enable bit of BSPI Control Register). If one data word is written in the trasmit fifo before enabling the BSPI no data will be transmitted.

Note When it is necessary to be notified about the end of last word transmission, TUFL event can be used for the purpose, provided BSPI is configured in master mode. In case this notification is required also when BSPI is configured as a slave, the end of transmission cannot be detected and a suitable wait loop must be implemented.

17.3.4 Receive FIFO

The BSPI Receive FIFO is a 16 word by 16-bit FIFO used to buffer the data words received from the BSPI bus.

The FIFO can operate in 8-bit and 16-bit modes as configured by the WL[1:0] bits of the BSPCSR1 register. Irrelevant of the word depth in the FIFO, if operating in 8-bit mode, the data will occupy both the Most Significant and the Least Significant Bytes of each location of the FIFO (data can be used either as left or right justified).

The receive FIFO enable bits RFE[3:0] declare how many words deep the FIFO is for all transfers. The FIFO defaults to one word deep. Whenever there is at least one block of data in the FIFO the RFNE bit is set in the BSPCSR2 register, i.e there is data in at least one location. The RFF flag does not get set until all locations of the FIFO contain data, i.e. RFF is set when the depth of FIFO is filled and nothing has been read out.

If the FIFO is one word deep then the RFNE and RFF flags are set once data is written to it. When the data is read then both flags are cleared. A write to and a read from the FIFO can happen independent of each once RFF is not set, if RFF is set a read must occur before the next write or an overflow (ROFL) will occur.

17.3.5 Start-up Status

If the BSPI is to operate in Master mode, it must first be enabled, then the MSTR bit must be set high. The TFE flag will be set, signalling that the Transmit FIFO is empty, if the TIE is set, a TFE interrupt will be generated. The data to be transferred must be written to the Transmit Data Register, the TFE interrupt will be cleared and then the BSPI clock will be generated according to the value of the BSPCLK register. The Transfer of data then begins. A second TFE interrupt occurs so that the peripheral has a full data transfer time to request the data before the next transfer is to begin.

If the BSPI is to operate in slave mode, once again the device must be enabled. The $\overline{SS}$ line must only be asserted low after the SCK from the master is stable. The TFE flag will be set, depending on the BSPI being enabled, signalling that the Transmit Data register is empty and will be cleared by a write to the Transmit Data register. The second TFE interrupt occurs to request data for the following transfer.

17.3.6 Clocking problems and clearing of the shift-register

Should a problem arise on the clock which results in a misalignment of data in the shift register of the BSPI, it may be cleared by disabling the BSPI enable. This has the effect of setting the TFE which requests data to be written to the Transmit Register for the next transfer. Clearing the BSPI enable will also reset the counter of bits received. The next block of data received will be written to the next location in the FIFO continuing on from the last good transfer. If the FIFO was just one word deep it will be written to the only location available.

17.3.7 Interrupt control

The BSPI generates one interrupt based upon the status bits monitoring the transmit and receive logic. The interrupt is acknowledged or cleared by subsequent read or write operations which remove the error or status update condition. It is the responsibility of the programmer to ascertain the source of the interrupt and then remove the error condition or alter the state of the BSPI. In the case of multiple errors the interrupt will remain active until all interrupt sources have been cleared.

For example, in the case of TFE, whenever the last word has been transferred to the transmit buffer, the TFE flag is asserted. If interrupts are enabled then an interrupt will be asserted to the processor. To clear the interrupt the user must write at least one data word into the FIFO, or disable the interrupts if this condition is valid.

17.3.8 DMA Interface

The DMA interface is a feature of the BSPI that allows data to be transferred to or from system memory using a DMA controller instead of main CPU. Data can be transferred in single data accesses or in burst mode, the amount of words being selectable by the programmer, thus making efficient more use of system bus. The BSPI DMA interface has the following features:

1) The DMA interface can be totally disabled using a bit in BSPCSR3 register.

2) User programmable burst size: 1, 4, 8 or 16 words can be transferred at a time.

3) Two request lines to the DMA are generated: one dedicated to the BSPI operating in transmit mode and another dedicated to the BSPI operating in receive mode. These signals are generated in independently from each other inside the BSPI.

4) Received 8-bit data, being sent from the BSPI to the memory, is arranged in a format which is compatible both with little and big endian convention, replicating the received data on both high and low byte in BSPRXR register.

The DMA interface makes use of pointers inside both the transmit and receive fifo (these being independent of one another) in order for it to decide when a DMA request to transmit or receive data can be made. The choice of burst size should match the one configured in the corresponding DMA channel.

Note *There is a restriction on the burst capability of DMA interface during reception. The total number of words to be transferred to system memory via DMA must be an integer multiple of the programmed Burst Length size, since DMA interface is not capable of handling incomplete received burst transfers. For example if Burst Length size is set to 4 and at the end of received data transfer BSPI FIFO has only 3 spaces, no request would be issued. On the contrary, with an even divide, the last chunk of received data will always be equal to the programmed size of the burst length.*

17.4 Register description

BSPI Control/Status Register 1 (BSPCSR1)

Address Offset: 08h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RFE[3:0]				WL[1:0]		CPHA	CPOL	BEIE	Reserved		REIE	RIE[1:0]		MSTR	BSPE
rw				rw		rw	rw	rw	-		rw	rw		rw	rw

Bits 15:12 = **RFE[3:0]:** *Receive FIFO Enable.*
The receive FIFO can be programmed to operate with a word depth up to 16. The receive FIFO enable bits declare how many words deep the FIFO is for all transfers. The FIFO defaults to one word deep, i.e. similar to a single data register. Table below shows how the FIFO is controlled.

RFE[3:0]	Depth of FIFO
0000	1st word enabled
0001	1st & 2nd words enabled
0010	1-3 words enabled
0011	1-4 words enabled
0100	1-5 words enabled
0101	1-6 words enabled
0110	1-7 words enabled
0111	1-8 words enabled
1000	1-9 words enabled
1001	1-10 words enabled
1010	1-11 words enabled
1011	1-12 words enabled
1100	1-13 words enabled
1101	1-14 words enabled
1110	1-15 words enabled
1111	1-16 words enabled

Bit 11:10 = **WL[1:0]**: *Word Length.*
These two bits configure the word length operation of the Receive FIFO and transmit data registers as shown below:

WL[1:0]	Word Length
00	8-bit
01	16-bit
10	Reserved
11	Reserved

Bit 9 **= CPHA**: *Clock Phase Select.*
Used with the CPOL bit to define the master-slave clock relationship. When CPHA=0, as soon as the $\overline{SS}$ goes low the first data sample is captured on the first edge of SCK. When CPHA=1, the data is captured on the second edge.

Bit 8 = **CPOL***: Clock Polarity Select.*
When this bit is cleared and data is not being transferred, a stable low value is present on the SCK pin. If the bit is set the SCK pin will idle high. This bit is used with the CPHA bit to define the master-slave clock relationship.
0 = Active high clocks selected; SCK idles low.
1 = Active low clocks selected; SCK idles high.

Bit 7 = **BEIE**: *Bus Error Interrupt Enable.*
When this bit is set to '1', an interrupt will be asserted to the processor whenever a Bus Error condition occurs.

Bit 6 = *Reserved.*
To be left at 0 level (reset status).

Bit 5 = *Reserved.*
To be left at 0 level (reset status).

Bit 4 = **REIE**: *Receive Error Interrupt Enable*.
When this bit is set to '1' and the Receiver Overflow error condition occurs, a Receive Error Interrupt will be asserted to the processor.

Bits 3:2 **= RIE[1:0]**: *BSPI Receive Interrupt Enables.*
The RIE1:0 bits are interrupt enables which configure when the processor will be interrupted on received data. The following configurations are possible.

RIE1	RIE0	Interrupted on
0	0	Disabled
0	1	Receive FIFO Not Empty
1	0	Reserved
1	1	Receive FIFO Full

Bit 1 **= MSTR**: *Master/Slave Select.*
1: BSPI is configured as a master
0: BSPI is configured as a slave

Bit 0 **= BSPE**: *BSPI System Enable.*
1: BSPI system is enabled
0: BSPI system is disabled

Note The peripheral should be enabled before selecting the interrupts. In this way the application software can avoid unexpected behaviours of interrupt request signal.

BSPI Control/Status Register 2 (BSPCSR2)

Address Offset: 0Ch

Reset value: 0040h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TIE[1:0]		TFE[3:0]				TFNE	TFF	TUFL	TFE	ROFL	RFF	RFNE	BERR	Res.	DFIFO
rw		rw				r	r	r	r	r	r	r	r	-	w

Bits 15:14 **= TIE[1:0]**: *BSPI Transmit Interrupt Enable.*
These bits control the source of the transmit interrupt.

TIE1	TIE0	Interrupted on
0	0	Disabled
0	1	Transmit FIFO Empty
1	0	Transmit underflow (only in Master mode)
1	1	Transmit FIFO Full

Bits 13:10 = **TFE[3:0]**: *Transmit FIFO Enable.*
These bits control the depth of the transmit FIFO. The table below indicates all valid settings.

TFE[3:0]	Depth of FIFO
0000	1st word enabled
0001	1st & 2nd words enabled
0010	1-3 words enabled
0011	1-4 words enabled
0100	1-5 words enabled
0101	1-6 words enabled
0110	1-7 words enabled
0111	1-8 words enabled
1000	1-9 words enabled
1001	1-10 words enabled
1010	1-11words enabled
1011	1-12words enabled
1100	1-13words enabled
1101	1-14words enabled
1110	1-15words enabled
1111	1-16words enabled

Bit 9 = **TFNE**: *Transmit FIFO Not Empty.*
This bit is set whenever the FIFO contains at least one data word.

Bit 8 = **TFF**: *Transmit FIFO Full.*
TFF is set whenever the number of words written to the transmit FIFO is equal to the number

of FIFO locations enabled by TFE[3:0]. The flag is set immediately after the data write is complete.

Bit 7 = **TUFL**: *Transmit Underflow.*
This status bit gets set only when BSPI is configured in master mode, when the TFE bit is set and, by the time the Transmit Data Register contents are to be transferred to the shift register for the next transmission, the processor has not yet put the data for transmission into the Transmit Data Register.
TUFL is set on the first edge of the clock when CPHA = 1 and when CPHA = 0 on the assertion of $\overline{SS}$. If TIE[1:0] bits are set to "10" then, when TUFL gets set, an interrupt will be asserted to the processor.

Note From an application point of view, it is important to be aware that the first word available after an underflow event has occurred should be ignored, as this data was loaded into the shift register before the underflow condition was flagged.

Bit 6 = **TFE**: *Transmit FIFO Empty.*
This bit gets set whenever the Transmit FIFO has transferred its last data word to the transmit buffer. If interrupts are enabled then an interrupt will be asserted whenever the last word has been transferred to the transmit buffer.

Bit 5 = **ROFL**: *Receiver Overflow.*
This bit gets set if the Receive FIFO is full and has not been read by the processor by the time another received word arrives. If the REIE bit is set then, when this bit gets set an interrupt will be asserted to the processor. This bit is cleared when a read takes place of the CSR register and the FIFO.

Bit 4 = **RFF**: *Receive FIFO Full.*
This status bit indicates that the number of FIFO locations, as defined by the RFE[3:0] bits, are all full, i.e. if the FIFO is 4 deep then all data has been received to all four locations.
If the RIE[1:0] bits are configured as '11' then, when this status bit gets set, an interrupt will be asserted to the processor. This bit is cleared when at least one data word has been read.

Bit 3 = **RFNE**: *Receive FIFO Not Empty.*
This status bit indicates that there is data in the Receive FIFO. It is set whenever there is at least one block of data in the FIFO i.e. for 8-bit mode 8 bits and for 16-bit mode 16 bits. If the RIE[1:0] bits are configured to '01' then whenever this bit gets set an interrupt will be asserted to the processor. This bit is cleared when all valid data has been read out of the FIFO.

Bit 2 = **BERR**: *Bus Error.*
This status bit indicates that a Bus Error condition has occurred, i.e. that more than one device has acted as a Master simultaneously on the BSPI bus. A Bus Error condition is defined as a condition where the Slave Select line goes active low when the module is configured as a Master, provided that MASK_SS bit in BSPCSR3 register is not set. This indicates contention in that more than one node on the BSPI bus is attempting to function as a Master. This bit is cleared when the Slave Select line is deasserted, MASK_SS bit is set or Slave mode is selected.

Bit 1 = *Reserved.*
To be left at 0 level (reset status).

Bit 0 = **DFIFO**: *Disable for the FIFO.*
When this bit is enabled, the FIFO pointers are all reset to zero, the RFE bits are set to zero and therefore the BSPI is set to one location. The data within the FIFO is lost. This bit is reset to zero after a clock cycle.

BSPI Control/Status Register 3 (BSPCSR3)

Address Offset: 14h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								RREQ_EN	TREQ_EN	RBURST_LEN [1:0]		TBURST_LEN [1:0]		DMA_EN	MASK_SS
-								rw	rw	rw		rw		rw	rw

This register is used to control the BSPI DMA interface. When the BSPI is receiving data, it will be the source of DMA transfer and system memory will act as destination. Conversely, when the BSPI is in transmit mode, it is sending data to an external source and it will act as DMA destination while system memory will be the source.

Bits 15:8 = Reserved. These bits must be always written to ‘0’.

Bit 7 = **RREQ_EN**: *Receive REQuest ENable*
This is the enable bit for the reception DMA request and it flags the DMA controller that an amount of data corresponding at least to the configured reception burst length is available in the BSPI receive FIFO to be transferred.
0 = Receive DMA requests are disabled.
1 = Receive DMA requests are enabled.

Bit 6 = **TREQ_EN**: *Transmit REQuest ENable*
This is the enable bit for the transmission DMA request and it flags the DMA controller that in the BSPI transmit FIFO there is an amount of free locations corresponding at least to the configured transmission burst length.
0 = Transmit DMA requests are disabled.
1 = Transmit DMA requests are enabled.

Bits 5:4 = **RBURST_LEN[1:0]**: *Receive BURST LENgth*
These bits configure the burst length when the BSPI receives data. This programmable length is used to set the number of data words the DMA controller is expected to retrieve upon a receive request; this value is used in the DMA interface to determine when the BSPI is ready to send a data burst to the DMA. A word can be 16 bit or 8 bit wide according to the configuration set in WL bits inside BSPCSR1 register.

RBURST_ LEN[1:0]	BURST LENGTH
00	1 word transferred
01	4 words transferred
10	8 words transferred
11	16 words transferred

Note The value set in RBURST_LEN field should match with the configured burst length set in the corresponding DMA channel (SoBurst field in DMA control register).

Bits 3:2 = **TBURST_LEN [1:0]**: *Transmit BURST LENgth*
These bits configure the burst length when the BSPI transmits data. This programmable length is used to set the number of data words the DMA controller is expected to send upon a transmit request; this value is used in the DMA interface to determine when the BSPI is ready to receive a data burst from the DMA. A word can be 16 bit or 8 bit wide according to the configuration set in WL bits inside BSPCSR1 register.

TBURST_ LEN[1:0]	BURST LENGTH
00	1 word transferred
01	4 words transferred
10	8 words transferred
11	16 words transferred

Note The value set in TBURST_LEN field should match with the configured burst length set in the corresponding DMA channel (SoBurst field in DMA control register).

Bit 1 = **DMA_EN**: *DMA interface ENable*
This bit is a general enable switch for the DMA interface. When BSPI DMA interface is disabled no request line will be activated and data transfer can occur through interrupt notification only.
0 = DMA interface is disabled.
1 = DMA interface is enabled.

Bit 0 = **MASK_SS**: *MASK Slave Select*
This bit can be used to mask the status of Slave Select pin when BSPI is in master mode and the pad corresponding to SS pin is not available. When this bit is set to ‘1’, the Bus Error interrupt condition cannot be detected since internally the related signal is always considered high regardless from the actual pad status.
0 = Slave Select pin is used.
1 = Slave Select pin is masked.

BSPI Master Clock Divider Register (BSPCLK)

Address Offset: 10h

Reset value: 0006h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								DIV[7:0]							
-								rw							

Bits 15:8 = *Reserved*.
To be left at 0 level (reset status).

Bits 7:0 = **DIV[7:0]**: *Divide factor bits.*
These bits are used to control the frequency of the BSPI serial clock with relation to the device clock APB_CLK. In master mode this number must be an even number greater than five, i.e. six is the lowest divide factor. In slave mode this number must be an even number greater than seven, i.e. eight is the lowest divide factor.
These bits must be set before the BSPE or MSTR bits, i.e. before the BSPI is configured into master mode.

BSPI Transmit Register (BSPTXR)

Address Offset: 04h

Reset value: n/a

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16 bit Transmission	TX[15:0]															
8 bit Transmission	TX[7:0]								Not Used							
	w															

Bits 15:0 = **TX[15:0]**: *Transmit data.*
This register is used to write data for transmission into the BSPI. If the FIFO is enabled then data written to this register will be transferred to the FIFO before transmission. If the FIFO is disabled then the register contents are transferred directly to the shift register for transmission. In sixteen bit mode all of the register bits are used. In eight bit mode only the upper eight bits of the register are used while lower eight bits are ignored. In both case the data is left justified, i.e. Bit[15] = MSB, Bit[0] / Bit[8] = LSB depending on the operating mode.

BSPI Receive Register (BSPRXR)

Address Offset: 00h

Reset value: 0000h

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16 bit Reception	RX[15:0]															
8 bit Reception	RX[7:0]								RX[7:0]							
	r															

Bits 15:0 = **RX[15:0]**: *Received data*.
This register contains the data received from the BSPI bus. If the FIFO is disabled then the data from the shift register is placed into the receive register directly. If the FIFO is enabled then the received data is transferred into the FIFO. In sixteen bit mode all the register bits are utilized. In eight bit reception mode the received data is replicated on the upper and lower eight bits of the register so to support both little and big endian memory systems.

17.4.1 Register map

A summary overview of the BSPI registers is given in the following table.

Table 47. BSPI Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**BSPRXR**	RX[15:0] (*)															
04	**BSPTXR**	TX[15:0] (*)															
08	**BSPCSR1**	RFE[3:0]				WL[1:0]		CPHA	CPOL	BEIE	Reserved		REIE	RIE[1:0]		MSTR	BSPE
0C	**BSPCSR2**	TIE[1:0]		TFE[3:0]				TFNE	TFF	TUFL	TFE	ROFL	RFF	RFNE	BERR	Res.	DFIFO
10	**BSPCLK**	Reserved								DIV[7:0]							
14	**BSPCSR3**	Reserved								RREQ_EN	TREQ_EN	RBURST_LEN[1:0]		TBURST_LEN [1:0]		DMA_EN	MASK_SS

(*)Data is justified depending on transmission mode, Bit 15 = MSB.

Refer to Table 20 on page 49 for the base address.

18 CONTROLLER AREA NETWORK (CAN)

18.1 Introduction

The CAN Peripheral consists of the CAN Core, Message RAM, Message Handler, Control Registers and Module Interface (Refer to Figure 29).

The CAN Core performs communication according to the CAN protocol version 2.0 part A and B. The bit rate can be programmed to values up to 1MBit/s. For the connection to the physical layer, additional transceiver hardware is required.

For communication on a CAN network, individual Message Objects are configured. The Message Objects and Identifier Masks for acceptance filtering of received messages are stored in the Message RAM.

All functions concerning the handling of messages are implemented in the Message Handler. These functions include acceptance filtering, the transfer of messages between the CAN Core and the Message RAM, and the handling of transmission requests as well as the generation of the module interrupt.

The register set of the CAN Peripheral can be accessed directly by the CPU through the module interface. These registers are used to control/configure the CAN Core and the Message Handler and to access the Message RAM.

18.2 Main Features

- Supports CAN protocol version 2.0 part A and B.
- Bit rates up to 1 MBit/s.
- 32 Message Objects.
- Each Message Object has its own identifier mask.
- Programmable FIFO mode (concatenation of Message Objects).
- Maskable interrupt.
- Disabled Automatic Re-transmission mode for Time Triggered CAN applications.
- Programmable loop-back mode for self-test operation.
- 8-bit non-multiplex Motorola HC08 compatible module interface.
- Two 16-bit module interfaces to the AMBA APB bus.

18.3 Block Diagram

The CAN Peripheral interfaces with the AMBA APB bus. Figure 29 shows the block diagram of the CAN Peripheral.

CAN Core

CAN Protocol Controller and Rx/Tx Shift Register for serial/parallel conversion of messages.

Message RAM

Stores Message Objects and Identifier Masks.

Registers

All registers used to control and to configure the CAN Peripheral.

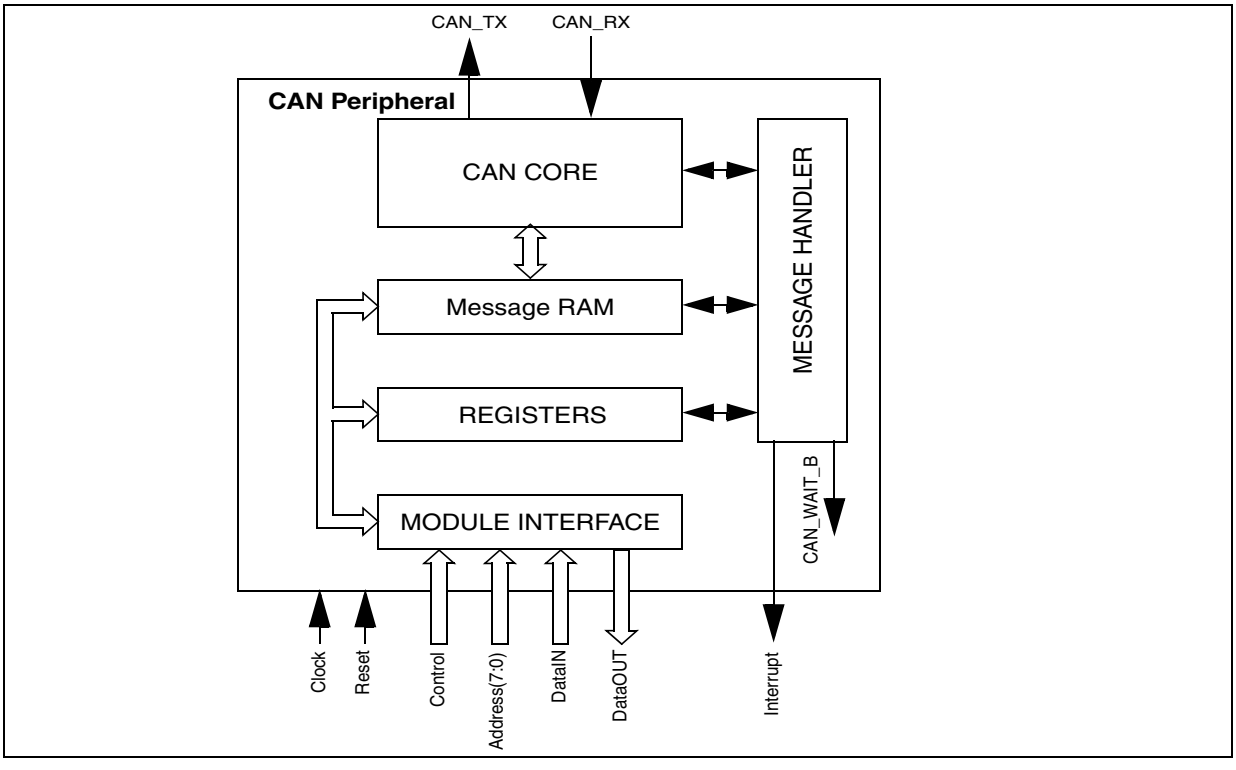
Message Handler

State Machine that controls the data transfer between the Rx/Tx Shift Register of the CAN Core and the Message RAM as well as the generation of interrupts as programmed in the Control and Configuration Registers.

Module Interface

CAN Peripheral interfaces to the AMBA APB 16-bit bus from ARM.

Figure 29. Block Diagram of the CAN Peripheral



18.4 Functional Description

18.4.1 Software Initialization

The software initialization is started by setting the Init bit in the CAN Control Register, either by a software or a hardware reset, or by going to Bus_Off state.

While the Init bit is set, all message transfers to and from the CAN bus are stopped and the status of the CAN_TX output pin is recessive (HIGH). The Error Management Logic (EML) counters are unchanged. Setting the Init bit does not change any configuration register.

To initialize the CAN Controller, software has to set up the Bit Timing Register and each Message Object. If a Message Object is not required, the corresponding MsgVal bit should be cleared. Otherwise, the entire Message Object has to be initialized.

Access to the Bit Timing Register and to the BAud Rate Prescaler (BRP) Extension Register for configuring bit timing is enabled when the Init and Configuration Change Enable *(*CCE) bits in the CAN Control Register are both set.

Resetting the Init bit (by CPU only) finishes the software initialization. Later, the Bit Stream Processor (BSP) (see Section 18.7.10: *Configuring the Bit Timing on page 249*) synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (≡ Bus Idle) before it can take part in bus activities and start the message transfer.

The initialization of the Message Objects is independent of Init and can be done on the fly, but the Message Objects should all be configured to particular identifiers or set to not valid before the BSP starts the message transfer.

To change the configuration of a Message Object during normal operation, software has to start by resetting the corresponding MsgVal bit. When the configuration is completed, MsgVal is set again.

18.4.2 CAN Message Transfer

Once the CAN Peripheral is initialized and Init bit is cleared, the CAN Peripheral Core synchronizes itself to the CAN bus and starts the message transfer.

Received messages are stored in their appropriate Message Objects if they pass the Message Handler's acceptance filtering. The whole message including all arbitration bits, DLC and eight data bytes are stored in the Message Object. If the Identifier Mask is used, the arbitration bits which are masked to "don't care" may be overwritten in the Message Object.

Software can read or write each message any time through the Interface Registers and the Message Handler guarantees data consistency in case of concurrent accesses.

Messages to be transmitted are updated by the application software. If a permanent Message Object (arbitration and control bits are set during configuration) exists for the message, only the data bytes are updated and the TxRqst bit with NewDat bit are set to start the

transmission. If several transmit messages are assigned to the same Message Object (when the number of Message Objects is not sufficient), the whole Message Object has to be configured before the transmission of this message is requested.

The transmission of any number of Message Objects may be requested at the same time. Message objects are transmitted subsequently according to their internal priority. Messages may be updated or set to not valid any time, even when their requested transmission is still pending. The old data will be discarded when a message is updated before its pending transmission has started.

Depending on the configuration of the Message Object, the transmission of a message may be requested autonomously by the reception of a remote frame with a matching identifier.

18.4.3 Disabled Automatic Re-Transmission Mode

In accordance with the CAN Specification (see ISO11898, 6.3.3 Recovery Management), the CAN Peripheral provides means for automatic re-transmission of frames that have lost arbitration or have been disturbed by errors during transmission. The frame transmission service will not be confirmed to the user before the transmission is successfully completed. This means that, by default, automatic retransmission is enabled. It can be disabled to enable the CAN Peripheral to work within a Time Triggered CAN (TTCAN, see ISO11898-1) environment.

Disabled Automatic Retransmission mode is enabled by setting the Disable Automatic Retransmission (DAR) bit in the CAN Control Register. In this operation mode, the programmer has to consider the different behaviour of bits TxRqst and NewDat in the Control Registers of the Message Buffers:

- When a transmission starts, bit TxRqst of the respective Message Buffer is cleared, while bit NewDat remains set.
- When the transmission completed successfully, bit NewDat is cleared.
- When a transmission fails (lost arbitration or error), bit NewDat remains set.

To restart the transmission, the CPU should set the bit TxRqst again.

18.4.4 Test Mode

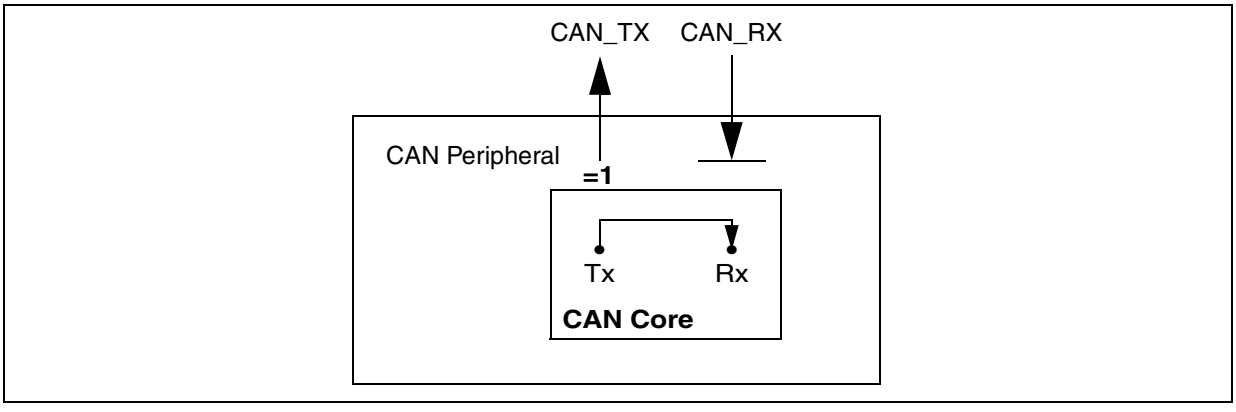
Test Mode is entered by setting the Test bit in the CAN Control Register. In Test Mode, bits Tx1, Tx0, LBack, Silent and Basic in the Test Register are writeable. Bit Rx monitors the state of the CAN_RX pin and therefore is only readable. All Test Register functions are disabled when the Test bit is cleared.

18.4.5 Silent Mode

The CAN Core can be set in Silent Mode by programming the Silent bit in the Test Register to one.

In Silent Mode, the CAN Peripheral is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the CAN Core is required to send a dominant bit (ACK bit, Error Frames), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. The Silent Mode can be used to analyse the traffic on a CAN bus without affecting it by the transmission of *dominant* bits. Figure 30 shows the connection of signals CAN_TX and CAN_RX to the CAN Core in Silent Mode.

Figure 30. CAN Core in Silent Mode

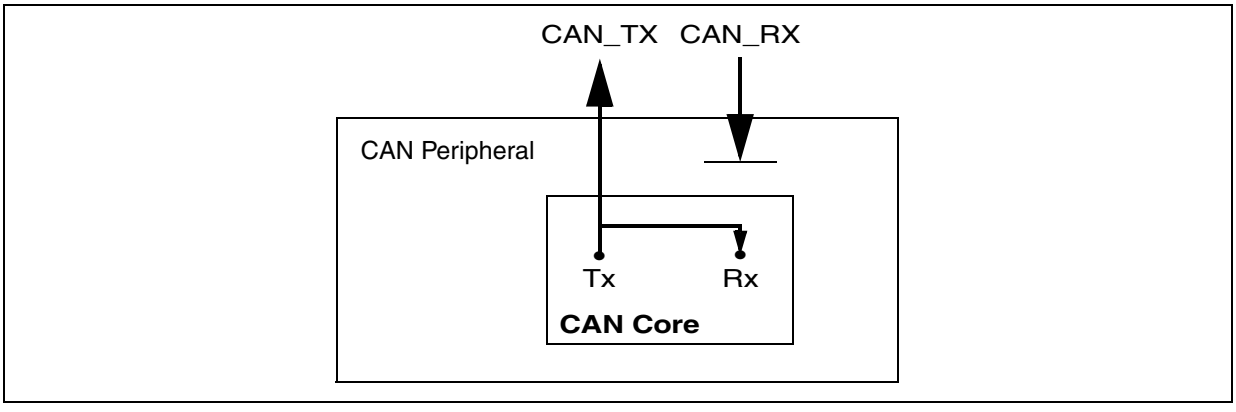


In ISO 11898-1, Silent Mode is called Bus Monitoring Mode.

18.4.6 Loop Back Mode

The CAN Core can be set in Loop Back Mode by programming the Test Register bit LBack to one. In Loop Back Mode, the CAN Core treats its own transmitted messages as received messages and stores them in a Receive Buffer (if they pass acceptance filtering). Figure 31 shows the connection of signals, CAN_TX and CAN_RX, to the CAN Core in Loop Back Mode.

Figure 31. CAN Core in Loop Back Mode

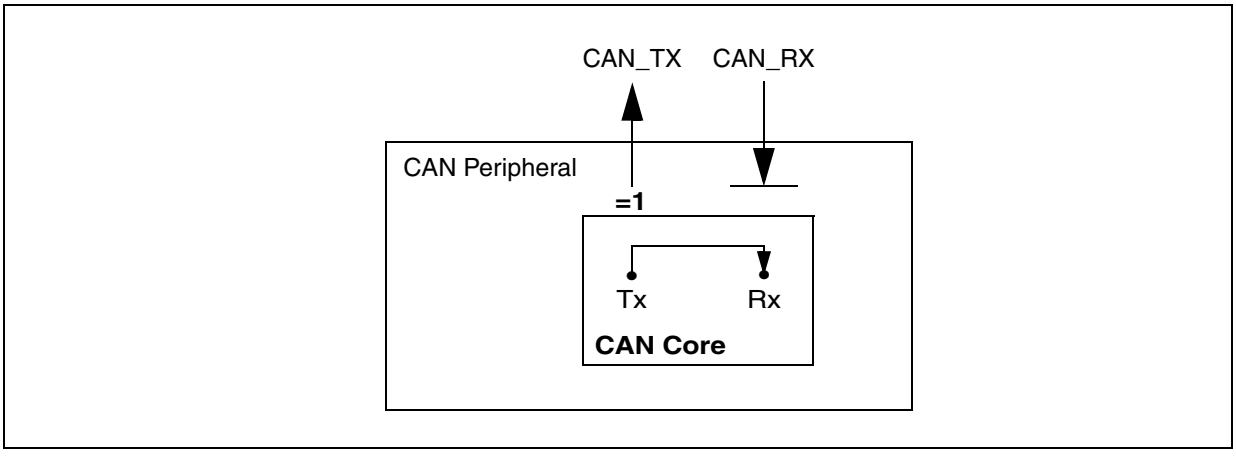


This mode is provided for self-test functions. To be independent from external stimulation, the CAN Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/ remote frame) in Loop Back Mode. In this mode, the CAN Core performs an internal feedback from its Tx output to its Rx input. The actual value of the CAN_RX input pin is disregarded by the CAN Core. The transmitted messages can be monitored on the CAN_TX pin.

18.4.7 Loop Back Combined with Silent Mode

It is also possible to combine Loop Back Mode and Silent Mode by programming bits LBack and Silent to one at the same time. This mode can be used for a “Hot Selftest”, which means that CAN Peripheral can be tested without affecting a running CAN system connected to the CAN_TX and CAN_RX pins. In this mode, the CAN_RX pin is disconnected from the CAN Core and the CAN_TX pin is held recessive. Figure 32 shows the connection of signals CAN_TX and CAN_RX to the CAN Core in case of the combination of Loop Back Mode with Silent Mode.

Figure 32. CAN Core in Loop Back Mode Combined with Silent Mode



18.4.8 Basic Mode

The CAN Core can be set in Basic Mode by programming the Test Register bit Basic to one. In this mode, the CAN Peripheral runs without the Message RAM.

The IF1 Registers are used as Transmit Buffer. The transmission of the contents of the IF1 Registers are requested by writing the Busy bit of the IF1 Command Request Register to one. The IF1 Registers are locked while the Busy bit is set. The Busy bit indicates that the transmission is pending.

As soon the CAN bus is idle, the IF1 Registers are loaded into the shift register of the CAN Core and the transmission is started. When the transmission has been completed, the Busy bit is reset and the locked IF1 Registers are released.

A pending transmission can be aborted at any time by resetting the Busy bit in the IF1 Command Request Register while the IF1 Registers are locked. If the CPU has reset the Busy bit, a possible retransmission in case of lost arbitration or in case of an error is disabled.

The IF2 Registers are used as a Receive Buffer. After the reception of a message the contents of the shift register is stored into the IF2 Registers, without any acceptance filtering.

Additionally, the actual contents of the shift register can be monitored during the message transfer. Each time a read Message Object is initiated by writing the Busy bit of the IF2 Command Request Register to one, the contents of the shift register are stored in the IF2 Registers.

In Basic Mode, the evaluation of all Message Object related control and status bits and the control bits of the IFn Command Mask Registers are turned off. The message number of the Command request registers is not evaluated. The NewDat and MsgLst bits in the IF2 Message Control Register retain their function, DLC3-0 indicate the received DLC, and the other control bits are read as ‘0’.

18.4.9 Software Control of CAN_TX Pin

Four output functions are available for the CAN transmit pin, CAN_TX. In addition to its default function (serial data output), the CAN transmit pin can drive the CAN Sample Point signal to monitor CAN_Core’s bit timing and it can drive constant dominant or recessive values. The latter two functions, combined with the readable CAN receive pin CAN_RX, can be used to check the physical layer of the CAN bus.

The output mode for the CAN_TX pin is selected by programming the Tx1 and Tx0 bits of the CAN Test Register.

The three test functions of the CAN_TX pin interfere with all CAN protocol functions. CAN_TX must be left in its default function when CAN message transfer or any of the test modes (Loop Back Mode, Silent Mode, or Basic Mode) are selected.

18.5 Register Description

The CAN Peripheral allocates an address space of 256 bytes. The registers are organized as 16-bit registers.

The two sets of interface registers (IF1 and IF2) control the CPU access to the Message RAM. They buffer the data to be transferred to and from the RAM, avoiding conflicts between CPU accesses and message reception/transmission.

In this section, the following abbreviations are used:

read/write (rw)	The software can read and write to these bits.
read-only (r)	The software can only read these bits.
write-only (w)	The software should only write to these bits.

The CAN registers are listed in Table 48.

Table 48. CAN Registers

Register Name	Address Offset	Reset Value
CAN Control Register (CAN_CR)	00h	0001h
Status Register (CAN_SR)	04h	0000h
Error Counter (CAN_ERR)	08h	0000h
Bit Timing Register (CAN_BTR)	0Ch	2301h
Test Register (CAN_TESTR)	14h	0000 0000 R000 0000 b
BRP Extension Register (CAN_BRPR)	18h	0000h
IFn Command Request Registers (CAN_IFn_CRR)	20h (CAN_IF1_CRR), 80h (CAN_IF2_CRR)	
IFn Command Mask Registers (CAN_IFn_CMR)	24h (CAN_IF1_CMR), 84h (CAN_IF2_CMR)	0000h
IFn Mask 1 Register (CAN_IFn_M1R)	28h (CAN_IF1_M1R), 88h (CAN_IF2_M1R)	FFFFh
IFn Mask 2 Register (CAN_IFn_M2R)	2Ch (CAN_IF1_M2R), 8Ch (CAN_IF2_M2R)	FFFFh
IFn Message Arbitration 1 Register (CAN_IFn_A1R)	30h (CAN_IF1_A1R), 90h (CAN_IF2_A1R)	0000h
IFn Message Arbitration 2 Register (CAN_IFn_A2R)	34h (CAN_IF1_A2R), 94h (CAN_IF2_A2R)	0000h
IFn Message Control Registers (CAN_IFn_MCR)	38h (CAN_IF1_MCR), 98h (CAN_IF2_MCR)	0000h
IFn Data A/B Registers (CAN_IFn_DAnR and CAN_IFn_DBnR)		
Interrupt Identifier Register (CAN_IDR)	10h	0000h
Transmission Request Registers 1 & 2 (CAN_TxRnR)	100h (CAN_TxR1R), 104h (CAN_TxR2R)	0000 0000h
New Data Registers 1 & 2 (CAN_NDnR)	120h (CAN_ND1R), 124h (CAN_ND2R)	0000 0000h

Table 48. CAN Registers

Register Name	Address Offset	Reset Value
Interrupt Pending Registers 1 & 2 (CAN_IPnR)	140h (CAN_IP1R), 144h (CAN_IP2R)	0000 0000h
Message Valid Registers 1 & 2 (CAN_MVnR)	160h (CAN_MV1R), 164h (CAN_MV2R)	0000 0000h

18.5.1 CAN Interface Reset State

After the hardware reset, the CAN Peripheral registers hold the reset values given in the register descriptions below.

Additionally the *busoff* state is reset and the output CAN_TX is set to recessive (HIGH). The value 0x0001 (Init = '1') in the CAN Control Register enables the software initialization. The CAN Peripheral does not influence the CAN bus until the CPU resets the Init bit to '0'.

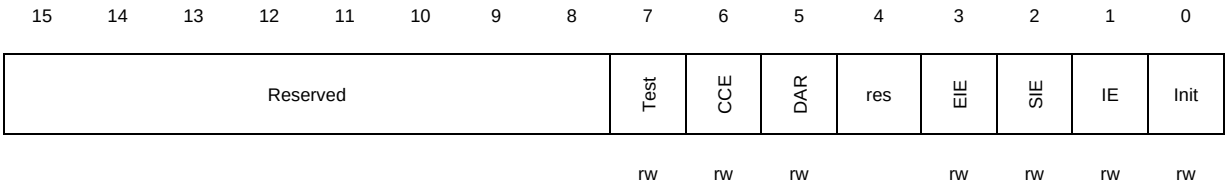
The data stored in the Message RAM is not affected by a hardware reset. After powering on, the contents of the Message RAM are undefined.

18.5.2 CAN Protocol Related Registers

These registers are related to the CAN protocol controller in the CAN Core. They control the operating modes and the configuration of the CAN bit timing and provide status information.

CAN Control Register (CAN_CR)

Address Offset: 00h
Reset value: 0001h



Bits 15:8 *Reserved*

These are reserved bits. These bits are always read as ‘0’ and must always be written with ‘0’.

Bit 7 Test: *Test Mode Enable*
0: Normal Operation.
1: Test Mode.

Bit 6 CCE: *Configuration Change Enable*
0: No write access to the Bit Timing Register.
1: Write access to the Bit Timing Register allowed (while bit Init=1).

Bit 5 DAR: *Disable Automatic Re-transmission*
0: Automatic Retransmission of disturbed messages enabled.
1: Automatic Retransmission disabled.

Bit 4 *Reserved*

This is a reserved bit. This bit is always read as ‘0’ and must always be written with ‘0’.

Bit 3 EIE: *Error Interrupt Enable*
0: Disabled - No Error Status Interrupt will be generated.
1: Enabled - A change in the bits BOff or EWarn in the Status Register will generate an interrupt.

Bit 2 SIE: *Status Change Interrupt Enable*
0: Disabled - No Status Change Interrupt will be generated.
1: Enabled - An interrupt will be generated when a message transfer is successfully completed or a CAN bus error is detected.

Bit 1 IE: *Module Interrupt Enable*
0: Disabled.
1: Enabled.

Bit 0 Init *Initialization*
0: Normal Operation.
1: Initialization is started.

*Note The busoff recovery sequence (see CAN Specification Rev. 2.0) cannot be shortened by setting or resetting the Init bit. If the device goes in the busoff state, it will set Init of its own accord, stopping all bus activities. Once Init has been cleared by the CPU, the device will then wait for 129 occurrences of Bus Idle (129 * 11 consecutive recessive bits) before resuming normal operations. At the end of the busoff recovery sequence, the Error Management Counters will be reset.*

During the waiting time after resetting Init, each time a sequence of 11 recessive bits has been monitored, a Bit0Error code is written to the Status Register, enabling the CPU to readily check up whether the CAN bus is stuck at dominant or continuously disturbed and to monitor the proceeding of the busoff recovery sequence.

Status Register (CAN_SR)

Address Offset: 04h
Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								BOff	EWarn	EPass	RxOk	TxOk	LEC		
								r	r	r	rw	rw	rw		

Bit 15:8 *Reserved*

These are reserved bits. These bits are always read as ‘0’ and must always be written with ‘0’.

Bit 7 BOff: *Busoff Status*
0: The CAN module is not in busoff state.
1: The CAN module is in busoff state.

Bit 6 EWarn: *Warning Status*
0: Both error counters are below the error warning limit of 96.
1: At least one of the error counters in the EML has reached the error warning limit of 96.

Bit 5 EPass: *Error Passive*
0: The CAN Core is error active.
1: The CAN Core is in the error passive state as defined in the CAN Specification.

Bit 4 RxOk: *Received a Message Successfully*
0: No message has been successfully received since this bit was last reset by the CPU. This bit is never reset by the CAN Core.
1: A message has been successfully received since this bit was last reset by the CPU (independent of the result of acceptance filtering).

Bit 3 TxOk: *Transmitted a Message Successfully*
0: Since this bit was reset by the CPU, no message has been successfully transmitted. This bit is never reset by the CAN Core.
1: Since this bit was last reset by the CPU, a message has been successfully (error free and acknowledged by at least one other node) transmitted.

Bits 2:0 LEC[2:0]: *Last Error Code (Type of the last error to occur on the CAN bus)*
The LEC field holds a code, which indicates the type of the last error to occur on the CAN bus. This field will be cleared to '0' when a message has been transferred (reception or transmission) without error. The unused code '7' may be written by the CPU to check for updates. Table 49 describes the error codes.

Table 49. Error Codes

Error Code	Meaning
0	No Error
1	Stuff Error: More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.
2	Form Error: A fixed format part of a received frame has the wrong format.
3	AckError: The message this CAN Core transmitted was not acknowledged by another node.
4	Bit1Error: During the transmission of a message (with the exception of the arbitration field), the device wanted to send a recessive level (bit of logical value '1'), but the monitored bus value was dominant.
5	Bit0Error: During the transmission of a message (or acknowledge bit, or active error flag, or overload flag), though the device wanted to send a dominant level (data or identifier bit logical value '0'), but the monitored Bus value was recessive. During busoff recovery, this status is set each time a sequence of 11 recessive bits has been monitored. This enables the CPU to monitor the proceedings of the busoff recovery sequence (indicating the bus is not stuck at *dominant* or continuously disturbed).
6	CRCError: The CRC check sum was incorrect in the message received, the CRC received for an incoming message does not match with the calculated CRC for the received data.
7	Unused: When the LEC shows the value '7', no CAN bus event was detected since the CPU wrote this value to the LEC.

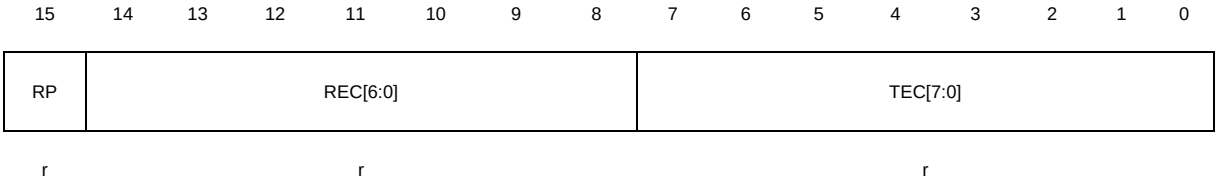
Status Interrupts

A Status Interrupt is generated by bits BOff and EWarn (Error Interrupt) or by RxOk, TxOk, and LEC (Status Change Interrupt) assuming that the corresponding enable bits in the CAN Control Register are set. A change of bit EPass or a write to RxOk, TxOk, or LEC will never generate a Status Interrupt.

Reading the Status Register will clear the Status Interrupt value (8000h) in the Interrupt Register, if it is pending.

Error Counter (CAN_ERR)

Address Offset: 08h
Reset value: 0000h



Bit 15 RP: *Receive Error Passive*
0: The Receive Error Counter is below the error passive level.
1: The Receive Error Counter has reached the error passive level as defined in the CAN Specification.

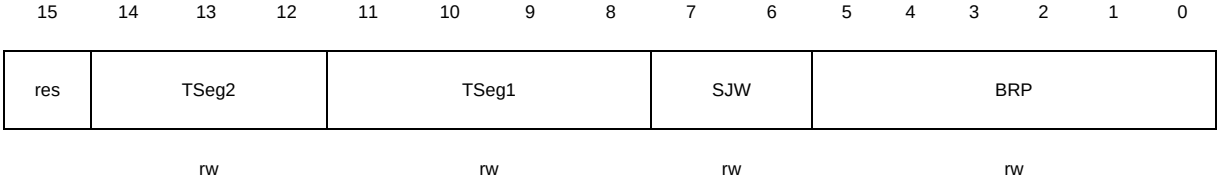
Bits 14:8 REC[6:0]: *Receive Error Counter*
Actual state of the Receive Error Counter. Values between 0 and 127.

Bits 7:0 TEC[7:0]: *Transmit Error Counter*
Actual state of the Transmit Error Counter. Values between 0 and 255.

Bit Timing Register (CAN_BTR)

Address Offset: 0Ch

Reset value: 2301h



Bit 15 *Reserved*

This is a reserved bit. This bit is always read as '0' and must always be written with '0'.

Bits 14:12 TSeg2: *Time segment after sample point*
0x0-0x7: Valid values for TSeg2 are [0 … 7]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Bits 11:8 TSeg1: *Time segment before the sample point minus Sync_Seg*
0x01-0x0F: valid values for TSeg1 are [1 … 15]. The actual interpretation by the hardware of this value is such that one more than the value programmed is used.

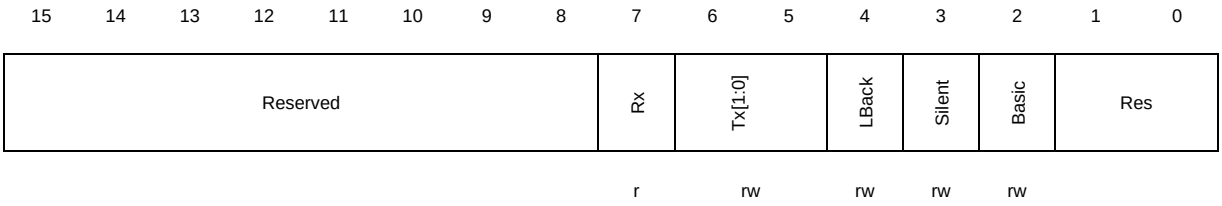
Bits 7:6 SJW*:* (*Re)Synchronization Jump Width*
0x0-0x3: Valid programmed values are [0 … 3]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Bits 5:0 BRP: *Baud Rate Prescaler*
0x01-0x3F: The value by which the oscillator frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this quanta. Valid values for the Baud Rate Prescaler are [0 … 63]. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Note With a module clock APB_CLK of 8 MHz, the reset value of 0x2301 configures the CAN Peripheral for a bit rate of 500 kBit/s. The registers are only writable if bits CCE and Init in the CAN Control Register are set.

Test Register (CAN_TESTR)

Address Offset: 14h
Reset value: 0000 0000 R000 0000 b (R:current value of RX pin)



Bits 15:8 Reserved
These are reserved bits. These bits are always read as '0' and must always be written with '0'.

Bit 7 Rx: *Current value of* CAN_RX Pin
0: The CAN bus is dominant (CAN_RX = '0').
1: The CAN bus is recessive (CAN_RX = '1').

Bit 6:5 Tx[1:0]: CAN_TX pin control
00: Reset value, CAN_TX is controlled by the CAN Core
01: Sample Point can be monitored at CAN_TX pin
10: CAN_TX pin drives a dominant ('0') value.
11: CAN_TX pin drives a recessive ('1') value.

Bit 4 LBack: *Loop Back Mode*
0: Loop Back Mode is disabled.
1: Loop Back Mode is enabled.

Bit 3 Silent: *Silent Mode*
0: Normal operation.
1: The module is in Silent Mode.

Bit 2 Basic: *Basic Mode*
0: Basic Mode disabled.
1: IF1 Registers used as Tx Buffer, IF2 Registers used as Rx Buffer.

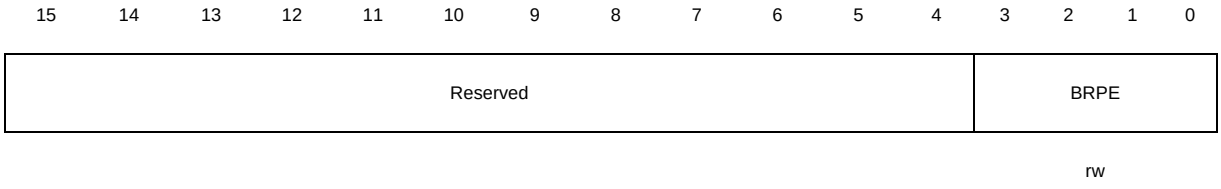
Bits 1:0 Reserved.
These are reserved bits. These bits are always read as '0' and must always be written with '0'.

Write access to the Test Register is enabled by setting the Test bit in the CAN Control Register. The different test functions may be combined, but Tx1-0 ≠ "00" disturbs message transfer.

BRP Extension Register (CAN_BRPR)

Address Offset: 18h

Reset value: 0000h



Bits 15:4 *Reserved*

These are reserved bits. These bits are always read as ‘0’ and must always be written with ‘0’.

Bits 3:0 BRPE: *Baud Rate Prescaler Extension*

0x00-0x0F: By programming BRPE, the Baud Rate Prescaler can be extended to values up to 1023. The actual interpretation by the hardware is that one more than the value programmed by BRPE (MSBs) and BRP (LSBs) is used.

18.5.3 Message Interface Register Sets

There are two sets of Interface Registers, which are used to control the CPU access to the Message RAM. The Interface Registers avoid conflict between the CPU access to the Message RAM and CAN message reception and transmission by buffering the data to be transferred. A complete Message Object (see Section 18.5.3.2) or parts of the Message Object may be transferred between the Message RAM and the IF*n* Message Buffer registers (see Section 18.5.3.1) in one single transfer.

The function of the two interface register sets is identical except for the Basic test mode. They can be used the way one set of registers is used for data transfer to the Message RAM while the other set of registers is used for the data transfer from the Message RAM, allowing both processes to be interrupted by each other. Table 50 IF1 and IF2 Message Interface Register Set on page 219 provides an overview of the two Interface Register sets.

Each set of Interface Registers consists of Message Buffer Registers controlled by their own Command Registers. The Command Mask Register specifies the direction of the data transfer and which parts of a Message Object will be transferred. The Command Request Register is used to select a Message Object in the Message RAM as target or source for the transfer and to start the action specified in the Command Mask Register.

Table 50. IF1 and IF2 Message Interface Register Set

Address	IF1 Register Set	Address	IF2 Register Set
CAN Base + 0x20	IF1 Command Request	CAN Base + 0x80	IF2 Command Request
CAN Base + 0x24	IF1 Command Mask	CAN Base + 0x84	IF2 Command Mask
CAN Base + 0x28	IF1 Mask 1	CAN Base + 0x88	IF2 Mask 1
CAN Base + 0x2C	IF1 Mask 2	CAN Base + 0x8C	IF2 Mask 2
CAN Base + 0x30	IF1 Arbitration 1	CAN Base + 0x90	IF2 Arbitration 1
CAN Base + 0x34	IF1 Arbitration 2	CAN Base + 0x94	IF2 Arbitration 2
CAN Base + 0x38	IF1 Message Control	CAN Base + 0x98	IF2 Message Control
CAN Base + 0x3C	IF1 Data A 1	CAN Base + 0x9C	IF2 Data A 1
CAN Base + 0x40	IF1 Data A 2	CAN Base + 0xA0	IF2 Data A 2
CAN Base + 0x44	IF1 Data B 1	CAN Base + 0xA4	IF2 Data B 1
CAN Base + 0x48	IF1 Data B 2	CAN Base + 0xA8	IF2 Data B 2

IF*n* Command Request Registers (CAN_IF*n*_CRR)

Address offset: 20h (CAN_IF1_CRR), 80h (CAN_IF2_CRR)
Reset Value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Busy	Reserved									Message Number					
r										rw					

A message transfer is started as soon as the application software has written the message number to the Command Request Register. With this write operation, the Busy bit is automatically set to notify the CPU that a transfer is in progress. After a waiting time of 3 to 6 APB_CLK periods, the transfer between the Interface Register and the Message RAM is completed. The Busy bit is cleared.

Bit 15 Busy: *Busy Flag*
0: Read/write action has finished.
1: Writing to the IF*n* Command Request Register is in progress.

This bit can only be read by the software.

Bits 14:6 Reserved

These are reserved bits. These bits are always read as ‘0’ and must always be written with ‘0’.

Bits 5:0 Message Number
0x01-0x20: Valid Message Number, the Message Object in the Message RAM is selected for data transfer.
0x00: Not a valid Message Number, interpreted as *0x20*.
0x21-0x3F: Not a valid Message Number, interpreted as *0x01-0x1F*.

Note When a Message Number that is not valid is written into the Command Request Register, the Message Number will be transformed into a valid value and that Message Object will be transferred.

IF*n* Command Mask Registers (CAN_IF*n*_CMR)

Address offset: 24h (CAN_IF1_CMR), 84h (CAN_IF2_CMR)
Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/ NewDat	Data A	Data B
								rw	rw	rw	rw	rw	rw	rw	rw

The control bits of the IF*n* Command Mask Register specify the transfer direction and select which of the IF*n* Message Buffer Registers are source or target of the data transfer.

Bits 15:8 *Reserved*

These are reserved bits. These bits are always read as ‘0’ and must always be written with ‘0’.

Bit 7 WR/RD: Write / Read
0: Read: Transfer data from the Message Object addressed by the Command Request Register into the selected Message Buffer Registers.
1: Write: Transfer data from the selected Message Buffer Registers to the Message Object addressed by the Command Request Register.

Bits 6:0 These bits of IFn Command Mask Register have different functions depending on the transfer direction:

Direction = Write

Bit 6 = Mask Access Mask Bits
0: Mask bits unchanged.
1: transfer Identifier Mask + MDir + MXtd to Message Object.

Bit 5 = Arb Access Arbitration Bits
0: Arbitration bits unchanged.
1: Transfer Identifier + Dir + Xtd + MsgVal to Message Object.

Bit 4 = Control Access Control Bits
0: Control Bits unchanged.
1: Transfer Control Bits to Message Object.

Bit 3 = ClrIntPnd Clear Interrupt Pending Bit
When writing to a Message Object, this bit is ignored.

Bit 2 = TxRqst/NewDat Access Transmission Request Bit
0: TxRqst bit unchanged.
1: Set TxRqst bit.

If a transmission is requested by programming bit TxRqst/NewDat in the IFn Command Mask Register, bit TxRqst in the IFn Message Control Register will be ignored.

Bit 1 = Data A Access Data Bytes 3:0
0: Data Bytes 3:0 unchanged.
1: Transfer Data Bytes 3:0 to Message Object.

Bit 0 = Data B Access Data Bytes 7:4
0: Data Bytes 7:4 unchanged.
1: Transfer Data Bytes 7:4 to Message Object.

Bits 6:0 Direction = Read

Bit 6 = Mask: *Access Mask Bits*
0: Mask bits unchanged.
1: Transfer Identifier Mask + MDir + MXtd to IF*n* Message Buffer Register.

Bit 5 = Arb: *Access Arbitration Bits*
0: Arbitration bits unchanged.
1: Transfer Identifier + Dir + Xtd + MsgVal to IF*n* Message Buffer Register.

Bit 4 = Control: *Access Control Bits*
0: Control Bits unchanged.
1: Transfer Control Bits to IF*n* Message Buffer Register.

Bit 3 = ClrIntPnd: *Clear Interrupt Pending Bit*
0: IntPnd bit remains unchanged.
1: Clear IntPnd bit in the Message Object.

Bit 2 = TxRqst/NewDat: *Access Transmission Request Bit*
0: NewDat bit remains unchanged.
1: Clear NewDat bit in the Message Object.

Note *A read access to a Message Object can be combined with the reset of the control bits IntPnd and NewDat. The values of these bits transferred to the IFn Message Control Register always reflect the status before resetting these bits.*

Bit 1 = Data A Access Data Bytes 3:0
0: Data Bytes 3:0 unchanged.
1: Transfer Data Bytes 3:0 to IF*n* Message Buffer Register.

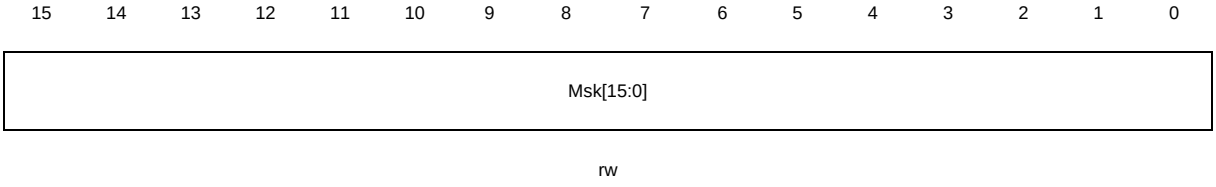
Bit 0 = Data B Access Data Bytes 7:4
0: Data Bytes 7:4 unchanged.
1: Transfer Data Bytes 7:4 to IF*n* Message Buffer Register.

18.5.3.1 IF*n* Message Buffer Registers

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM. The function of the Message Objects bits is described in Section 18.5.3.2: *Message Object in the Message Memory on page 227*.

IF*n* Mask 1 Register (CAN_IF*n*_M1R)

Address offset: 28h (CAN_IF1_M1R), 88h (CAN_IF2_M1R)
Reset Value: FFFFh



The function of the Msk bits is described in Section 18.5.3.2: *Message Object in the Message Memory on page 227*.

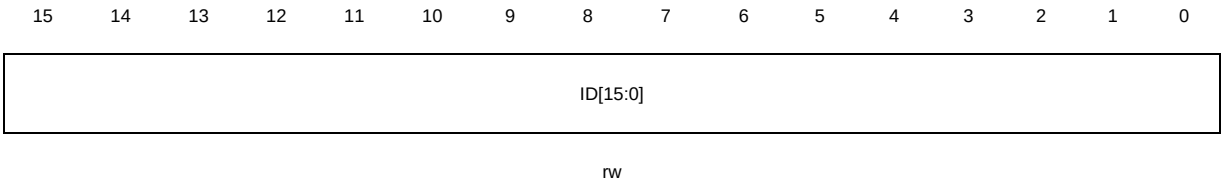
IF*n* Mask 2 Register (CAN_IF*n*_M2R)

Address offset: 2Ch (CAN_IF1_M2R), 8Ch (CAN_IF2_M2R)
Reset Value: FFFFh

The function of the Message Objects bits is described in the Section 18.5.3.2: *Message Object in the Message Memory on page 227*.

IF*n* Message Arbitration 1 Register (CAN_IF*n*_A1R)

Address offset: 30h (CAN_IF1_A1R), 90h (CAN_IF2_A1R)
Reset Value: 0000h



The function of the Message Objects bits is described in the Section 18.5.3.2: *Message Object in the Message Memory on page 227*.

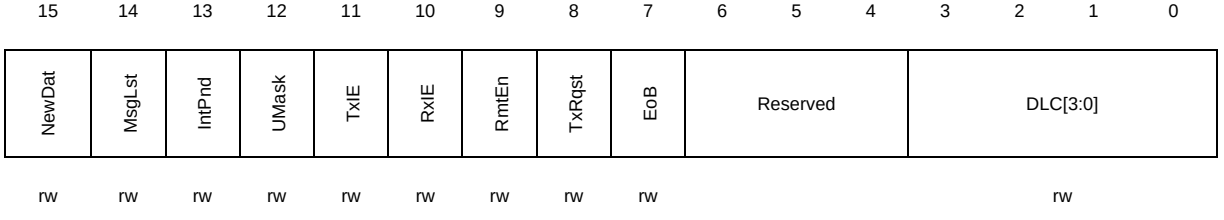
IF*n* Message Arbitration 2 Register (CAN_IF*n*_A2R)

Address offset: 34h (CAN_IF1_A2R), 94h (CAN_IF2_A2R)
Reset Value: 0000h

The function of the Message Objects bits is described in the Section 18.5.3.2: *Message Object in the Message Memory on page 227*.

IF*n* Message Control Registers (CAN_IF*n*_MCR)

Address offset: 38h (CAN_IF1_MCR), 98h (CAN_IF2_MCR)
Reset Value: 0000h



The function of the Message Objects bits is described in the Section 18.5.3.2: *Message Object in the Message Memory on page 227*.

IF*n* Data A/B Registers (CAN_IF*n*_DA*n*R and CAN_IF*n*_DB*n*R)

The data bytes of CAN messages are stored in the IF*n* Message Buffer Registers in the following order:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IF1 Message Data A1 (address 0x3C)	Data(1)								Data(0)							
IF1 Message Data A2 (address 0x40)	Data(3)								Data(2)							
IF1 Message Data B1 (address 0x44)	Data(5)								Data(4)							
IF1 Message Data B2 (address 0x48)	Data(7)								Data(6)							
IF2 Message Data A1 (address 0x9C)	Data(1)								Data(0)							
IF2 Message Data A2 (address 0xA0)	Data(3)								Data(2)							
IF2 Message Data B1 (address 0xA4)	Data(5)								Data(4)							
IF2 Message Data B2 (address 0xA8)	Data(7)								Data(6)							
	rw								rw							

In a CAN Data Frame, Data(0) is the first, Data(7) is the last byte to be transmitted or received. In CAN's serial bit stream, the MSB of each byte will be transmitted first.

18.5.3.2 Message Object in the Message Memory

There are 32 Message Objects in the Message RAM. To avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission, the CPU cannot directly access the Message Objects, these accesses are handled through the IF*n* Interface Registers.

Table 51 provides an overview of the structures of a Message Object.

Table 51. Structure of a Message Object in the Message Memory

Message Object												
UMask	Msk 28-0	MXtd	MDir	EoB	NewDat		MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
MsgVal	ID28-0	Xtd	Dir	DLC 3-0	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7

The Arbitration Registers ID28-0, Xtd, and Dir are used to define the identifier and type of outgoing messages and are used (together with the mask registers Msk28-0, MXtd, and MDir) for acceptance filtering of incoming messages. A received message is stored in the valid Message Object with matching identifier and direction set to receive (Data Frame) or transmit (Remote Frame). Extended frames can be stored only in Message Objects with Xtd set, standard frames in Message Objects with Xtd clear. If a received message (Data Frame or Remote Frame) matches more than one valid Message Object, it is stored into that with the lowest message number. For details see Section "Acceptance Filtering of Received Messages".

MsgVal — Message Valid

1: The Message Object is configured and should be considered by the Message Handler.

0: The Message Object is ignored by the Message Handler.

Note *The application software must reset the MsgVal bit of all unused Messages Objects during the initialization before it resets bit Init in the CAN Control Register. This bit must also be reset before the identifier Id28-0, the control bits Xtd, Dir, or the Data Length Code DLC3-0 are modified, or if the Messages Object is no longer required.*

UMask — Use Acceptance Mask

1: Use Mask (Msk28-0, MXtd, and MDir) for acceptance filtering.

0: Mask ignored.

Note *If the UMask bit is set to one, the Message Object's mask bits have to be programmed during initialization of the Message Object before MsgVal is set to one.*

ID28-0 — Message Identifier

ID28 - ID0, 29-bit Identifier ("Extended Frame")

ID28 - ID18, 11-bit Identifier ("Standard Frame")

Msk28-0 — Identifier Mask

1: The corresponding identifier bit is used for acceptance filtering.

0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.

Xtd — Extended Identifier

1: The 29-bit ("extended") Identifier will be used for this Message Object.

0: The 11-bit ("standard") Identifier will be used for this Message Object.

MXtd — Mask Extended Identifier

1: The extended identifier bit (IDE) is used for acceptance filtering.

0: The extended identifier bit (IDE) has no effect on the acceptance filtering.

Note *When 11-bit ("standard") Identifiers are used for a Message Object, the identifiers of received Data Frames are written into bits ID28 to ID18. For acceptance filtering, only these bits together with mask bits Msk28 to Msk18 are considered.*

Dir — Message Direction

1: Direction = transmit: On TxRqst, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the TxRqst bit of this Message Object is set (if RmtEn = one).

0: Direction = *receive*: On TxRqst, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object.

MDir — Mask Message Direction

1: The message direction bit (Dir) is used for acceptance filtering.

0: The message direction bit (Dir) has no effect on the acceptance filtering.

EoB — End of Buffer

1: Single Message Object or last Message Object of a FIFO Buffer.

0: Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer.

Note *This bit is used to concatenate two or more Message Objects (up to 32) to build a FIFO Buffer. For single Message Objects (not belonging to a FIFO Buffer), this bit must always be set to one. For details on the concatenation of Message Objects see Section* "Configuring a FIFO Buffer"*.*

NewDat — New Data

1: The Message Handler or the application software has written new data into the data portion of this Message Object.

0: No new data has been written into the data portion of this Message Object by the Message Handler since last time this flag was cleared by the application software.

MsgLst — Message Lost (only valid for Message Objects with direction = *receive*)

1: The Message Handler stored a new message into this object when NewDat was still set, the CPU has lost a message.

0: No message lost since last time this bit was reset by the CPU.

RxIE — Receive Interrupt Enable

1: IntPnd will be set after a successful reception of a frame.

0: IntPnd will be left unchanged after a successful reception of a frame.

TxIE — Transmit Interrupt Enable

1: IntPnd will be set after a successful transmission of a frame.

0: IntPnd will be left unchanged after the successful transmission of a frame.

IntPnd — Interrupt Pending

1: This message object is the source of an interrupt. The Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

0:This message object is not the source of an interrupt.

RmtEn — Remote Enable

1: At the reception of a Remote Frame, TxRqst is set.

0: At the reception of a Remote Frame, TxRqst is left unchanged.

TxRqst — Transmit Request

1: The transmission of this Message Object is requested and is not yet done.

0: This Message Object is not waiting for transmission.

DLC3-0 Data Length Code

0-8: Data Frame has 0-8 data bytes.

9-15: Data Frame has 8 data bytes

Note *The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the Message Handler stores a data frame, it will write the DLC to the value given by the received message.*

Data 0: 1st data byte of a CAN Data Frame

Data 1: 2nd data byte of a CAN Data Frame

Data 2: 3rd data byte of a CAN Data Frame

Data 3: 4th data byte of a CAN Data Frame

Data 4: 5th data byte of a CAN Data Frame

Data 5: 6th data byte of a CAN Data Frame

Data 6: 7th data byte of a CAN Data Frame

Data 7 : 8th data byte of a CAN Data Frame

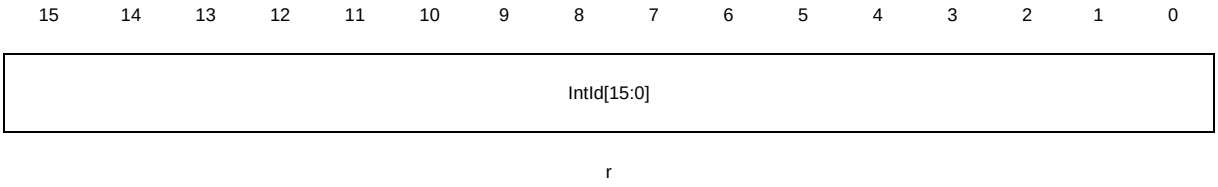
Note *The Data 0 Byte is the first data byte shifted into the shift register of the CAN Core during a reception while the Data 7 byte is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by unspecified values.*

18.5.4 Message Handler Registers

All Message Handler registers are read-only. Their contents, TxRqst, NewDat, IntPnd, and MsgVal bits of each Message Object and the Interrupt Identifier is status information provided by the Message Handler FSM.

Interrupt Identifier Register (CAN_IDR)

Address Offset: 10h
Reset value: 0000h



Bits 15:0 IntId15:0 Interrupt Identifier (Table 52 indicates the source of the interrupt)

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the application software has cleared it. If IntId is different from 0x0000 and IE is set, the IRQ interrupt signal to the EIC is active. The interrupt remains active until IntId is back to value 0x0000 (the cause of the interrupt is reset) or until IE is reset.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object' s interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object's IntPnd bit. The Status Interrupt is cleared by reading the Status Register.

Table 52. Source of Interrupts

0x0000	No Interrupt is Pending
0x0001-0x0020	Number of Message Object which caused the interrupt.
0x0001-0x0020	Number of Message Object which caused the interrupt.
0x0021-0x7FFF	unused
0x8000	Status Interrupt
0x8001-0xFFFF	unused

Transmission Request Registers 1 & 2 (CAN_TxR*n*R)

Address Offset: 100h (CAN_TxR1R), 104h (CAN_TxR2R)
Reset Value: 0000 0000h

31	30	29	28	27	26	25	24	**23**	22	21	20	19	18	17	16
TxRqst[32:17]															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TxRqst[16:1]															
r															

These registers hold the TxRqst bits of the 32 Message Objects. By reading the TxRqst bits, the CPU can check which Message Object in a Transmission Request is pending. The TxRqst bit of a specific Message Object can be set/reset by the application software through the IFn Message Interface Registers or by the Message Handler after reception of a Remote Frame or after a successful transmission.

Bits 31:16 TxRqst32-17 Transmission Request Bits (of all Message Objects)
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

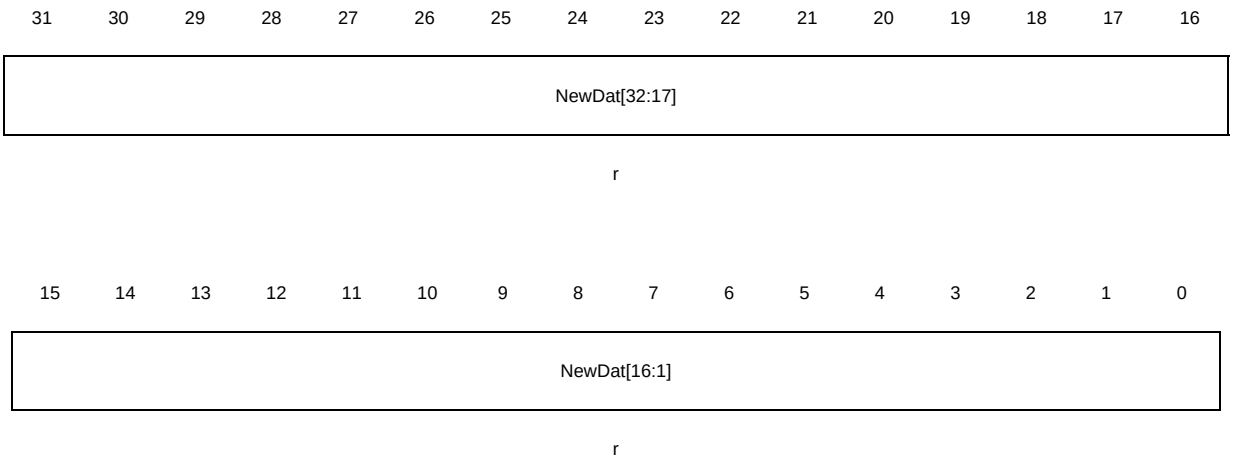
These bits are read only.

Bits 15:0 TxRqst16-1 Transmission Request Bits (of all Message Objects)
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet done.

These bits are read only.

New Data Registers 1 & 2 (CAN_ND*n*R)

Address Offset: 120h (CAN_ND1R), 124h (CAN_ND2R)
Reset Value: 0000 0000h



These registers hold the NewDat bits of the 32 Message Objects. By reading out the NewDat bits, the CPU can check for which Message Object the data portion was updated. The NewDat bit of a specific Message Object can be set/reset by the CPU through the IFn Message Interface Registers or by the Message Handler after reception of a Data Frame or after a successful transmission.

Bits 31:16 NewDat32-17 New Data Bits (of all Message Objects)
0: No new data has been written into the data portion of this Message Object by the Message Handler since the last time this flag was cleared by the application software.
1: The Message Handler or the application software has written new data into the data portion of this Message Object.

Bits 15:0 NewDat16-1 New Data Bits (of all Message Objects)
0: No new data has been written into the data portion of this Message Object by the Message Handler since the last time this flag was cleared by the application software.
1: The Message Handler or the application software has written new data into the data portion of this Message Object.

Interrupt Pending Registers 1 & 2 (CAN_IP*n*R)

Address Offset: 140h (CAN_IP1R), 144h (CAN_IP2R)
Reset Value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IntPnd[32:17]															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IntPnd[16:1]															
r															

These registers contain the IntPnd bits of the 32 Message Objects. By reading the IntPnd bits, the CPU can check for which Message Object an interrupt is pending. The IntPnd bit of a specific Message Object can be set/reset by the application software through the IFn Message Interface Registers or by the Message Handler after reception or after a successful transmission of a frame. This will also affect the value of IntId in the Interrupt Register.

Bits 31:16 IntPnd32-17 *Interrupt Pending Bits (of all Message Objects)*
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt.

Bits 15:0 IntPnd16-1 *Interrupt Pending Bits (of all Message Objects)*
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt.

Message Valid Registers 1 & 2 (CAN_MV*n*R)

Address Offset: 160h (CAN_MV1R), 164h (CAN_MV2R)
Reset Value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MsgVal[32:17]															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MsgVal[16:1]															
r															

These registers hold the MsgVal bits of the 32 Message Objects. By reading the MsgVal bits, the application software can check which Message Object is valid. The MsgVal bit of a specific Message Object can be set/reset by the application software via the IFn Message Interface Registers.

Bits 31:16 MsgVal32-17 Message Valid Bits (of all Message Objects)
0: This Message Object is ignored by the Message Handler.
1: This Message Object is configured and should be considered by the Message Handler.

Bits 15:0 MsgVal16-1 Message Valid Bits (of all Message Objects)
0: This Message Object is ignored by the Message Handler.
1: This Message Object is configured and should be considered by the Message Handler.

18.6 Register Map

Table 53. CAN Register Map

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	CAN_CR	Reserved								Test	CCE	DAR	res	EIE	SIE	IE	Init
04h	CAN_SR	Reserved								BOff	EWarn	EPass	RxOk	TxOk	LEC		
08h	CAN_ERR	RP	REC6-0							TEC7-0							
0Ch	CAN_BTR	res	TSeg2			TSeg1				SJW		BRP					
10h	CAN_IDR	IntId15-8								IntId7-0							
14h	CAN_TESTR	Reserved								Rx	Tx1	Tx0	LBack	Silent	Basic	Reserved	
18h	CAN_BRPR	Reserved												BRPE			
20h	CAN_IF1_CRR	Busy	Reserved							Message Number							
24h	CAN_IF1_CMR	Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/	Data A	Data B
28h	CAN_IF1_M1R	Msk15-0															
2Ch	CAN_IF1_M2R	MXtd	MDir	res	Msk28-16												
30h	CAN_IF1_A1R	ID15-0															
34h	CAN_IF1_A2R	MsgVal	Xtd	Dir	ID28-16												

Table 53. CAN Register Map

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
38h	CAN_IF1_MCR	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC3-0			
3Ch	CAN_IF1_DA1R	Data(1)								Data(0)							
40h	CAN_IF1_DA2R	Data(3)								Data(2)							
44h	CAN_IF1_DB1R	Data(5)								Data(4)							
48h	CAN_IF1_DB2R	Data(7)								Data(6)							
80h	CAN_IF2_CRR	Busy	Reserved								Message Number						
84h	CAN_IF2_CMR	Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/	Data A	Data B
88h	CAN_IF2_M1R	Msk15-0															
8Ch	CAN_IF2_M2R	MXtd	MDir	res	Msk28-16												
90h	CAN_IF2_A1R	ID15-0															
94h	CAN_IF2_A2R	MsgVal	Xtd	Dir	ID28-16												
98h	CAN_IF2_MCR	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC3-0			
9Ch	CAN_IF2_DA1R	Data(1)								Data(0)							

Table 53. CAN Register Map

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A0h	CAN_IF2_DA2R	Data(3)								Data(2)							
A4h	CAN_IF2_DB1R	Data(5)								Data(4)							
A8h	CAN_IF2_DB2R	Data(7)								Data(6)							
100h	CAN_TxR1R	TxRqst16-1															
104h	CAN_TxR2R	TxRqst32-17															
120h	CAN_ND1R	NewDat16-1															
124h	CAN_ND2R	NewDat32-17															
140h	CAN_IP1R	IntPnd16-1															
144h	CAN_IP2R	IntPnd32-17															
160h	CAN_MV1R	MsgVal16-1															
164h	CAN_MV2R	MsgVal32-17															

Note Reserved bits are read as 0' except for IFn Mask 2 Register where they are read as '1'.

18.7 CAN Communications

18.7.1 Managing Message Objects

The configuration of the Message Objects in the Message RAM (with the exception of the bits MsgVal, NewDat, IntPnd, and TxRqst) will not be affected by resetting the chip. All the Message Objects must be initialized by the application software or they must be “not valid” (MsgVal = ‘0’) and the bit timing must be configured before the application software clears the Init bit in the CAN Control Register.

The configuration of a Message Object is done by programming Mask, Arbitration, Control and Data fields of one of the two interface registers to the desired values. By writing to the corresponding IFn Command Request Register, the IFn Message Buffer Registers are loaded into the addressed Message Object in the Message RAM.

When the Init bit in the CAN Control Register is cleared, the CAN Protocol Controller state machine of the CAN_Core and state machine of the Message Handler control the internal data flow of the CAN Peripheral. Received messages that pass the acceptance filtering are stored in the Message RAM, messages with pending transmission request are loaded into the CAN_Core’s Shift Register and are transmitted through the CAN bus.

The application software reads received messages and updates messages to be transmitted through the IF*n* Interface Registers. Depending on the configuration, the CPU is interrupted on certain CAN message and CAN error events.

18.7.2 Message Handler State Machine

The Message Handler controls the data transfer between the Rx/Tx Shift Register of the CAN Core, the Message RAM and the IF*n* Registers.

The Message Handler FSM controls the following functions:

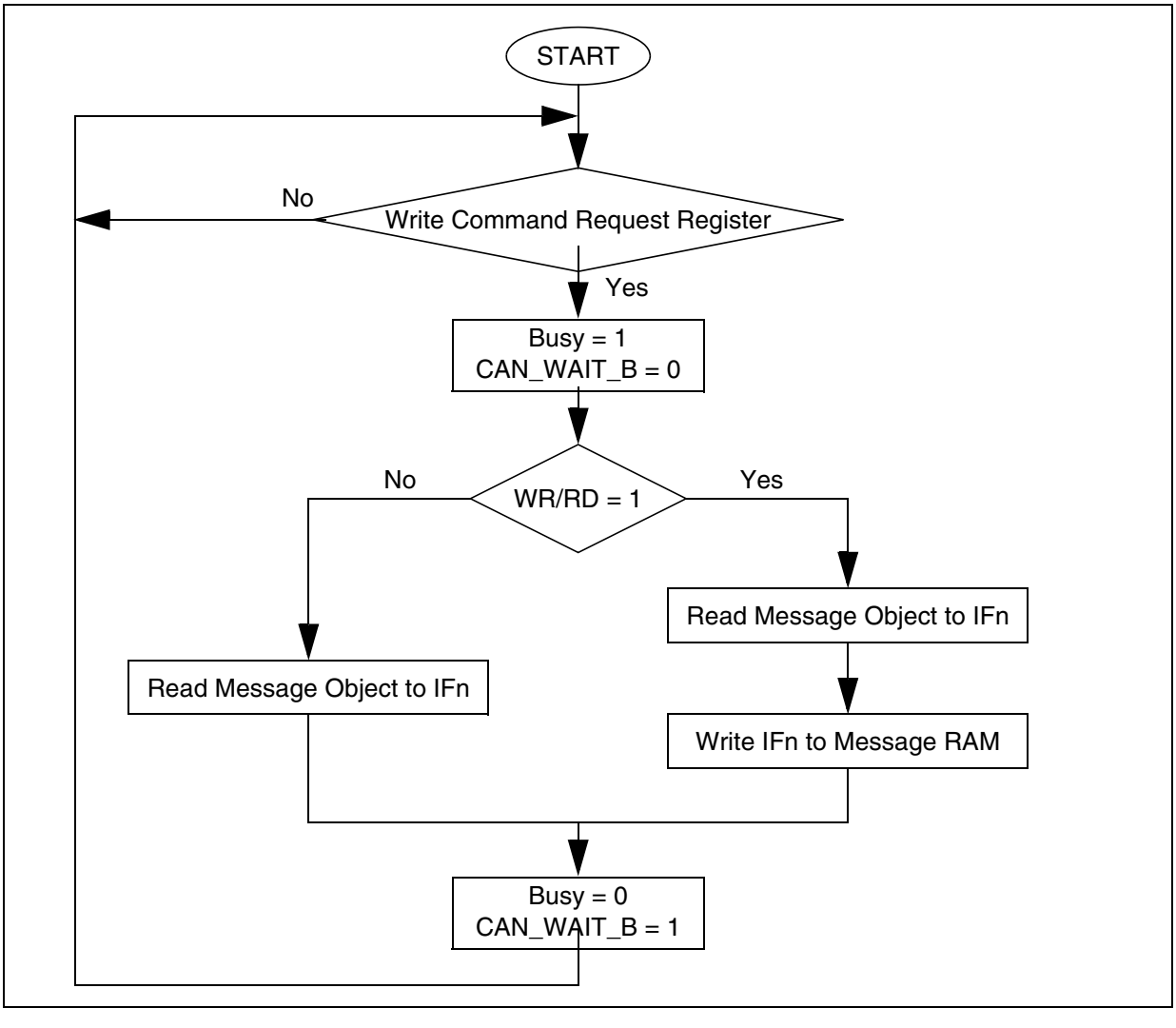
- Data Transfer from IF*n* Registers to the Message RAM
- Data Transfer from Message RAM to the IF*n* Registers
- Data Transfer from Shift Register to the Message RAM
- Data Transfer from Message RAM to Shift Register
- Data Transfer from Shift Register to the Acceptance Filtering unit
- Scanning of Message RAM for a matching Message Object
- Handling of TxRqst flags
- Handling of interrupts.

18.7.2.1 Data Transfer from/to Message RAM

When the CPU initiates a data transfer between the IF*n* Registers and Message RAM, the Message Handler sets the Busy bit in the respective Command Request Register (CAN_IFn_CRR). After the transfer has completed, the Busy bit is again cleared (see Figure 33).

The respective Command Mask Register specifies whether a complete Message Object or only parts of it will be transferred. Due to the structure of the Message RAM, it is not possible to write single bits/bytes of one Message Object. It is always necessary to write a complete Message Object into the Message RAM. Therefore, the data transfer from the IF*n* Registers to the Message RAM requires a read-modify-write cycle. First, those parts of the Message Object that are not to be changed are read from the Message RAM and then the complete contents of the Message Buffer Registers are written into the Message Object.

Figure 33. Data transfer between IF*n* Registers and Message RAM



After a partial write of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will set the actual contents of the selected Message Object.

After a partial read of a Message Object, the Message Buffer Registers that are not selected in the Command Mask Register will be left unchanged.

18.7.2.2 Message Transmission

If the shift register of the CAN Core cell is ready for loading and if there is no data transfer between the IFn Registers and Message RAM, the MsgVal bits in the Message Valid Register and TxRqst bits in the Transmission Request Register are evaluated. The valid Message Object with the highest priority pending transmission request is loaded into the shift register by the Message Handler and the transmission is started. The NewDat bit of the Message Object is reset.

After a successful transmission and also if no new data was written to the Message Object (NewDat = '0') since the start of the transmission, the TxRqst bit of the Message Control register (CAN_IFn_MCR) will be reset. If TxIE bit of the Message Control register (CAN_IFn_MCR) is set, IntPnd bit of the Interrupt Identifier register (CAN_IDR) will be set after a successful transmission. If the CAN Peripheral has lost the arbitration or if an error occurred during the transmission, the message will be retransmitted as soon as the CAN bus is free again. Meanwhile, if the transmission of a message with higher priority has been requested, the messages will be transmitted in the order of their priority.

18.7.2.3 Acceptance Filtering of Received Messages

When the arbitration and control field (Identifier + IDE + RTR + DLC) of an incoming message is completely shifted into the Rx/Tx Shift Register of the CAN Core, the Message Handler FSM starts the scanning of the Message RAM for a matching valid Message Object.

To scan the Message RAM for a matching Message Object, the Acceptance Filtering unit is loaded with the arbitration bits from the CAN Core shift register. The arbitration and mask fields (including MsgVal, UMask, NewDat, and EoB) of Message Object 1 are then loaded into the Acceptance Filtering unit and compared with the arbitration field from the shift register. This is repeated with each following Message Object until a matching Message Object is found or until the end of the Message RAM is reached.

If a match occurs, the scan is stopped and the Message Handler FSM proceeds depending on the type of frame (Data Frame or Remote Frame) received.

Reception of Data Frame

The Message Handler FSM stores the message from the CAN Core shift register into the respective Message Object in the Message RAM. Not only the data bytes, but all arbitration bits and the Data Length Code are stored in the corresponding Message Object. This is done to keep the data bytes connected with the identifier even if arbitration mask registers are used.

The NewDat bit is set to indicate that new data (not yet seen by the CPU) has been received. The application software should reset NewDat bit when the Message Object has been read. If at the time of reception, the NewDat bit was already set, MsgLst is set to indicate that the previous data (supposedly not seen by the CPU) is lost. If the RxIE bit is set, the IntPnd bit is set, causing the Interrupt Register to point to this Message Object.

The TxRqst bit of this Message Object is reset to prevent the transmission of a Remote Frame, while the requested Data Frame has just been received.

Reception of Remote Frame

When a Remote Frame is received, three different configurations of the matching Message Object have to be considered:

1) **Dir** = '1' (direction = *transmit*), **RmtEn** = '1', **UMask** = '1' or'0'
At the reception of a matching Remote Frame, the TxRqst bit of this Message Object is set. The rest of the Message Object remains unchanged.

2) **Dir** = '1' (direction = *transmit*), **RmtEn** = '0', **UMask** ='0'
At the reception of a matching Remote Frame, the TxRqst bit of this Message Object remains unchanged; the Remote Frame is ignored.

3) **Dir** = '1' (direction = *transmit*), **RmtEn** = '0', **UMask** ='1'
At the reception of a matching Remote Frame, the TxRqst bit of this Message Object is reset. The arbitration and control field (Identifier + IDE + RTR + DLC) from the shift register is stored in the Message Object of the Message RAM and the NewDat bit of this Message Object is set. The data field of the Message Object remains unchanged; the Remote Frame is treated similar to a received Data Frame.

18.7.2.4 Receive/Transmit Priority

The receive/transmit priority for the Message Objects is attached to the message number. Message Object 1 has the highest priority, while Message Object 32 has the lowest priority. If more than one transmission request is pending, they are serviced due to the priority of the corresponding Message Object.

18.7.3 Configuring a Transmit Object

Table 54 shows how a Transmit Object should be initialized.

Table 54. Initialization of a Transmit Object

MsgVal	Arb	Data	Mask	EoB	Dir	NewDat	MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
1	appl.	appl.	appl.	1	1	0	0	0	appl.	0	appl.	0

The Arbitration Register values (ID28-0 and Xtd bit) are provided by the application. They define the identifier and type of the outgoing message. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to ID28 - ID18. The ID17 - ID0 can then be disregarded.

If the TxIE bit is set, the IntPnd bit will be set after a successful transmission of the Message Object.

If the RmtEn bit is set, a matching received Remote Frame will cause the TxRqst bit to be set; the Remote Frame will autonomously be answered by a Data Frame.

The Data Register values (DLC3-0, Data0-7) are provided by the application, TxRqst and RmtEn may not be set before the data is valid.

The Mask Registers (Msk28-0, UMask, MXtd, and MDir bits) may be used (UMask=’1’) to allow groups of Remote Frames with similar identifiers to set the TxRqst bit. The Dir bit should not be masked.

18.7.4 Updating a Transmit Object

The CPU may update the data bytes of a Transmit Object any time through the IFn Interface registers, neither MsgVal nor TxRqst have to be reset before the update. Even if only a part of the data bytes are to be updated, all four bytes of the corresponding IFn Data A Register or IFn Data B Register have to be valid before the contents of that register are transferred to the Message Object. Either the CPU has to write all four bytes into the IFn Data Register or the Message Object is transferred to the IFn Data Register before the CPU writes the new data bytes.

When only the (eight) data bytes are updated, first 0x0087 is written to the Command Mask Register and then the number of the Message Object is written to the Command Request Register, concurrently updating the data bytes and setting TxRqst.

To prevent the reset of TxRqst at the end of a transmission that may already be in progress while the data is updated, NewDat has to be set together with TxRqst. For details see Section 18.7.2.2: *Message Transmission on page 242*.

When NewDat is set together with TxRqst, NewDat will be reset as soon as the new transmission has started.

18.7.5 Configuring a Receive Object

Table 55 shows how a Receive Object should be initialized.

Table 55. Initialization of a Receive Object

MsgVal	Arb	Data	Mask	EoB	Dir	NewDat	MsgLst	RxIE	TxIE	IntPnd	RmtEn	TxRqst
1	appl.	appl.	appl.	1	0	0	0	appl.	0	0	0	0

The Arbitration Registers values (ID28-0 and Xtd bit) are provided by the application. They define the identifier and type of accepted received messages. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to ID28 - ID18. Then ID17 - ID0 can be disregarded. When a Data Frame with an 11-bit Identifier is received, ID17 - ID0 will be set to ‘0’.

If the RxIE bit is set, the IntPnd bit will be set when a received Data Frame is accepted and stored in the Message Object.

The Data Length Code (DLC3-0) is provided by the application. When the Message Handler stores a Data Frame in the Message Object, it will store the received Data Length Code and eight data bytes. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by unspecified values.

The Mask Registers (Msk28-0, UMask, MXtd, and MDir bits) may be used (UMask=’1’) to allow groups of Data Frames with similar identifiers to be accepted. The Dir bit should not be masked in typical applications.

18.7.6 Handling Received Messages

The CPU may read a received message any time via the IF*n* Interface registers. The data consistency is guaranteed by the Message Handler state machine.

Typically, the CPU will write first 0x007F to the Command Mask Register and then the number of the Message Object to the Command Request Register. This combination will transfer the whole received message from the Message RAM into the Message Buffer Register. Additionally, the bits NewDat and IntPnd are cleared in the Message RAM (not in the Message Buffer).

If the Message Object uses masks for acceptance filtering, the arbitration bits shows which of the matching messages have been received.

The actual value of NewDat shows whether a new message has been received since the last time this Message Object was read. The actual value of MsgLst shows whether more than one message has been received since the last time this Message Object was read. MsgLst will not be automatically reset.

By means of a Remote Frame, the CPU may request another CAN node to provide new data for a receive object. Setting the TxRqst bit of a receive object will cause the transmission of a Remote Frame with the receive object's identifier. This Remote Frame triggers the other CAN node to start the transmission of the matching Data Frame. If the matching Data Frame is received before the Remote Frame could be transmitted, the TxRqst bit is automatically reset.

18.7.7 Configuring a FIFO Buffer

With the exception of the EoB bit, the configuration of Receive Objects belonging to a FIFO Buffer is the same as the configuration of a (single) Receive Object, see Section 18.7.5: *Configuring a Receive Object on page 245*.

To concatenate two or more Message Objects into a FIFO Buffer, the identifiers and masks (if used) of these Message Objects have to be programmed to matching values. Due to the implicit priority of the Message Objects, the Message Object with the lowest number will be the first Message Object of the FIFO Buffer. The EoB bit of all Message Objects of a FIFO Buffer except the last have to be programmed to zero. The EoB bits of the last Message Object of a FIFO Buffer is set to one, configuring it as the End of the Block.

18.7.8 Receiving Messages with FIFO Buffers

Received messages with identifiers matching to a FIFO Buffer are stored in a Message Object of this FIFO Buffer starting with the Message Object with the lowest message number.

When a message is stored in a Message Object of a FIFO Buffer, the NewDat bit of this Message Object is set. By setting NewDat while EoB is zero, the Message Object is locked for further write access by the Message Handler until the application software has written the NewDat bit back to zero.

Messages are stored into a FIFO Buffer until the last Message Object of this FIFO Buffer is reached. If none of the preceding Message Objects is released by writing NewDat to zero, all further messages for this FIFO Buffer will be written into the last Message Object of the FIFO Buffer and therefore overwrite previous messages.

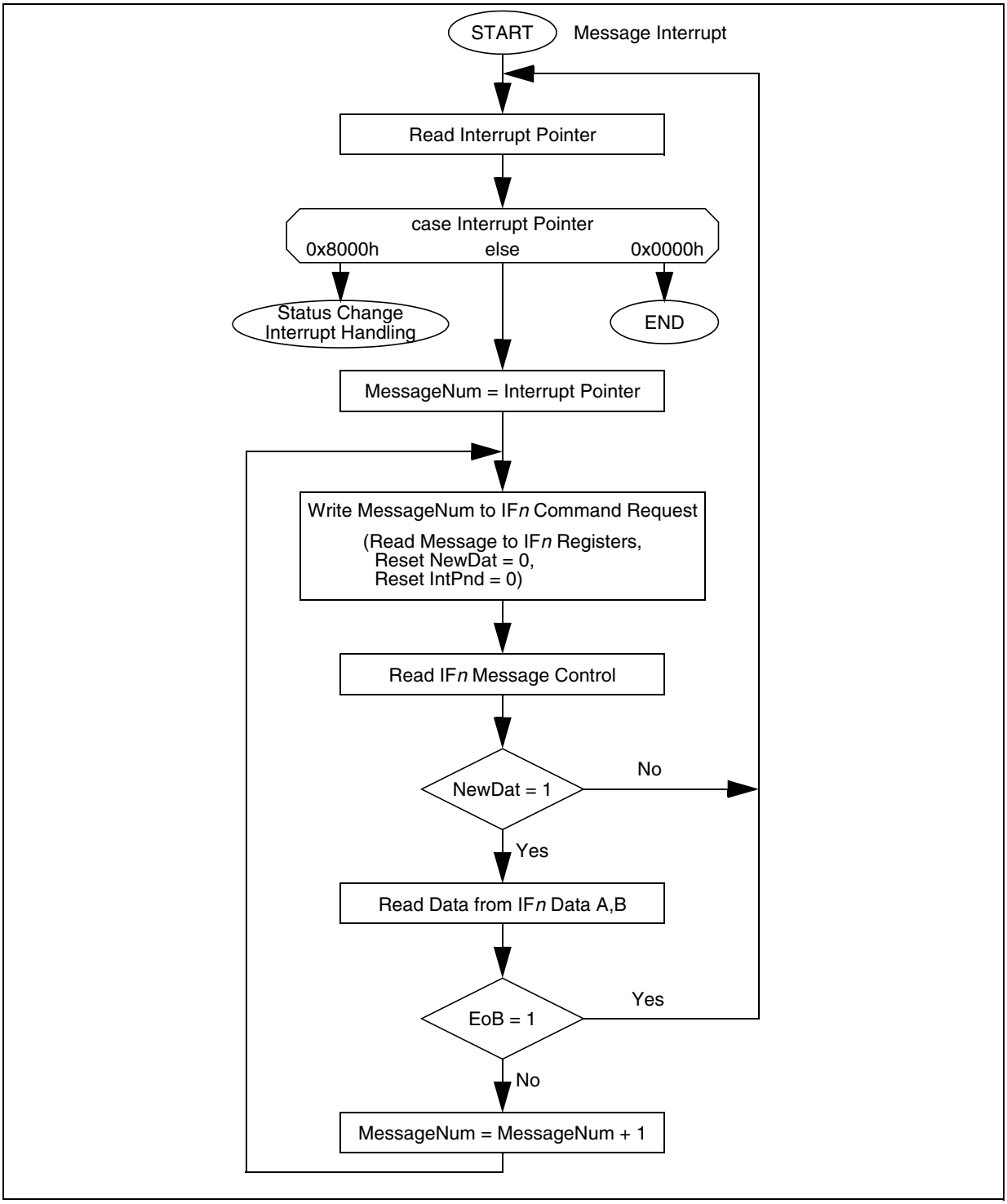
18.7.8.1 Reading from a FIFO Buffer

When the CPU transfers the contents of a Message Object to the IFn Message Buffer register by writing its number to the IFn Command Request Register, the corresponding Command Mask Register should be programmed in such a way that bits NewDat and IntPnd are reset to zero (TxRqst/NewDat = '1' and ClrIntPnd = '1'). The values of these bits in the Message Control Register always reflect the status before resetting the bits.

To assure the correct function of a FIFO Buffer, the CPU should read the Message Objects starting at the FIFO Object with the lowest message number.

Figure 34 shows how a set of Message Objects which are concatenated to a FIFO Buffer can be handled by the CPU.

Figure 34. CPU Handling of a FIFO Buffer



18.7.9 Handling Interrupts

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority, disregarding their chronological order. An interrupt remains pending until the application software has cleared it.

The Status Interrupt has the highest priority. Among the message interrupts, interrupt priority of the Message Object decreases with increasing message number.

A message interrupt is cleared by clearing the IntPnd bit of the Message Object. The Status Interrupt is cleared by reading the Status Register.

The interrupt identifier, IntId, in the Interrupt Register, indicates the cause of the interrupt. When no interrupt is pending, the register will hold the value zero. If the value of the Interrupt Register is different from zero, then there is an interrupt pending and, if IE is set, the IRQ interrupt signal to the EIC is active. The interrupt remains active until the Interrupt Register is back to value zero (the cause of the interrupt is reset) or until IE is reset.

The value 0x8000 indicates that an interrupt is pending because the CAN Core has updated (not necessarily changed) the Status Register (Error Interrupt or Status Interrupt). This interrupt has the highest priority. The CPU can update (reset) the status bits RxOk, TxOk and LEC, but a write access of the CPU to the Status Register can never generate or reset an interrupt.

All other values indicate that the source of the interrupt is one of the Message Objects. IntId points to the pending message interrupt with the highest interrupt priority.

The CPU controls whether a change of the Status Register may cause an interrupt (bits EIE and SIE in the CAN Control Register) and whether the interrupt line becomes active when the Interrupt Register is different from zero (bit IE in the CAN Control Register). The Interrupt Register will be updated even when IE is reset.

The CPU has two possibilities to follow the source of a message interrupt. First, it can follow the IntId in the Interrupt Register and second it can poll the Interrupt Pending Register (see See “Interrupt Pending Registers 1 & 2 (CAN_IPnR)” on page 235.).

An interrupt service routine that is reading the message that is the source of the interrupt may read the message and reset the Message Object’s IntPnd at the same time (bit ClrIntPnd in the Command Mask Register). When IntPnd is cleared, the Interrupt Register will point to the next Message Object with a pending interrupt.

18.7.10 Configuring the Bit Timing

Even if minor errors in the configuration of the CAN bit timing do not result in immediate failure, the performance of a CAN network can be reduced significantly.

In many cases, the CAN bit synchronization will amend a faulty configuration of the CAN bit timing to such a degree that only occasionally an error frame is generated. However, in the case of arbitration, when two or more CAN nodes simultaneously try to transmit a frame, a misplaced sample point may cause one of the transmitters to become error passive.

The analysis of such sporadic errors requires a detailed knowledge of the CAN bit synchronization inside a CAN node and interaction of the CAN nodes on the CAN bus.

18.7.10.1 Bit Time and Bit Rate

CAN supports bit rates in the range of lower than 1 kBit/s up to 1000 kBit/s. Each member of the CAN network has its own clock generator, usually a quartz oscillator. The timing parameter of the bit time (i.e. the reciprocal of the bit rate) can be configured individually for each CAN node, creating a common bit rate even though the oscillator periods of the CAN nodes (f_{osc}) may be different.

The frequencies of these oscillators are not absolutely stable, small variations are caused by changes in temperature or voltage and by deteriorating components. As long as the variations remain inside a specific oscillator tolerance range (df), the CAN nodes are able to compensate for the different bit rates by re-synchronizing to the bit stream.

According to the CAN specification, the bit time is divided into four segments (see Figure 35). The Synchronization Segment, the Propagation Time Segment, the Phase Buffer Segment 1 and the Phase Buffer Segment 2. Each segment consists of a specific, programmable number of time quanta (see Table 56). The length of the time quantum ($t_{q)}$, which is the basic time unit of the bit time, is defined by the CAN controller's system clock f_{APB} and the BRP bit of the Bit Timing Register (CAN_BTR): t_q = BRP / f_{APB}.

The Synchronization Segment, Sync_Seg, is that part of the bit time where edges of the CAN bus level are expected to occur. The distance between an edge, that occurs outside of Sync_Seg, and the Sync_Seg is called the phase error of that edge. The Propagation Time Segment, Prop_Seg, is intended to compensate for the physical delay times within the CAN network. The Phase Buffer Segments Phase_Seg1 and Phase_Seg2 surround the Sample Point. The (Re-)Synchronization Jump Width (SJW) defines how far a re-synchronization may move the Sample Point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.

Figure 35. Bit Timing

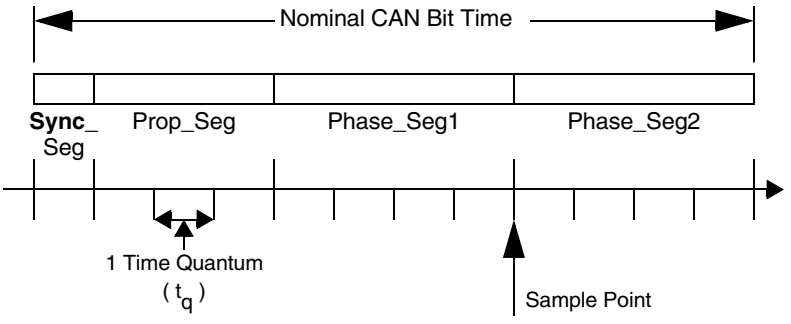


Table 56. CAN Bit Time Parameters

Parameter	Range	Remark
BRP	[1 .. 32]	defines the length of the time quantum t_q
Sync_Seg	1 t_q	fixed length, synchronization of bus input to system clock
Prop_Seg	[1.. 8] t_q	compensates for the physical delay times
Phase_Seg1	[1..8] t_q	may be lengthened temporarily by synchronization
Phase_Seg2	[1.. 8] t_q	may be shortened temporarily by synchronization
SJW	[1 .. 4] t_q	may not be longer than either Phase Buffer Segment
This table describes the minimum programmable ranges required by the CAN protocol		

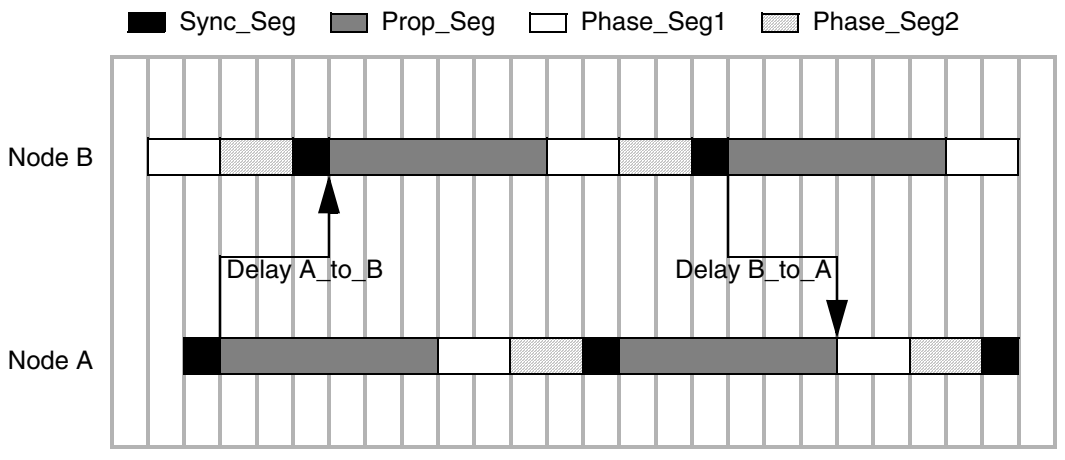
A given bit rate may be met by different bit time configurations, but for the proper function of the CAN network the physical delay times and the oscillator's tolerance range have to be considered.

18.7.10.2 Propagation Time Segment

This part of the bit time is used to compensate physical delay times within the network. These delay times consist of the signal propagation time on the bus and the internal delay time of the CAN nodes.

Any CAN node synchronized to the bit stream on the CAN bus will be out of phase with the transmitter of that bit stream, caused by the signal propagation time between the two nodes. The CAN protocol's non-destructive bitwise arbitration and the dominant acknowledge bit provided by receivers of CAN messages requires that a CAN node transmitting a bit stream must also be able to receive dominant bits transmitted by other CAN nodes that are synchronized to that bit stream. The example in Figure 36 shows the phase shift and propagation times between two CAN nodes.

Figure 36. Propagation Time Segment



Delay A_to_B >= node output delay(A) + bus line delay(A→B) + node input delay(B)

Prop_Seg >= Delay A_to_B + Delay B_to_A

Prop_Seg >= 2 • [max(node output delay+ bus line delay + node input delay)]

In this example, both nodes A and B are transmitters, performing an arbitration for the CAN bus. Node A has sent its Start of Frame bit less than one bit time earlier than node B, therefore node B has synchronized itself to the received edge from recessive to dominant. Since node B has received this edge delay (A_to_B) after it has been transmitted, B's bit timing segments are shifted with respect to A. Node B sends an identifier with higher priority and so it will win the arbitration at a specific identifier bit when it transmits a dominant bit while node A transmits a recessive bit. The dominant bit transmitted by node B will arrive at node A after the delay (B_to_A).

Due to oscillator tolerances, the actual position of node A's Sample Point can be anywhere inside the nominal range of node A's Phase Buffer Segments, so the bit transmitted by node B

must arrive at node A before the start of Phase_Seg1. This condition defines the length of Prop_Seg.

If the edge from recessive to dominant transmitted by node B arrives at node A after the start of Phase_Seg1, it can happen that node A samples a recessive bit instead of a dominant bit, resulting in a bit error and the destruction of the current frame by an error flag.

The error occurs only when two nodes arbitrate for the CAN bus that have oscillators of opposite ends of the tolerance range and that are separated by a long bus line. This is an example of a minor error in the bit timing configuration (Prop_Seg to short) that causes sporadic bus errors.

Some CAN implementations provide an optional 3 Sample Mode but the CAN Peripheral does not. In this mode, the CAN bus input signal passes a digital low-pass filter, using three samples and a majority logic to determine the valid bit value. This results in an additional input delay of 1 t_q, requiring a longer Prop_Seg.

18.7.10.3 Phase Buffer Segments and Synchronization

The Phase Buffer Segments (Phase_Seg1 and Phase_Seg2) and the Synchronization Jump Width (SJW) are used to compensate for the oscillator tolerance. The Phase Buffer Segments may be lengthened or shortened by synchronization.

Synchronizations occur on edges from recessive to dominant, their purpose is to control the distance between edges and Sample Points.

Edges are detected by sampling the actual bus level in each time quantum and comparing it with the bus level at the previous Sample Point. A synchronization may be done only if a recessive bit was sampled at the previous Sample Point and if the bus level at the actual time quantum is dominant.

An edge is synchronous if it occurs inside of Sync_Seg, otherwise the distance between edge and the end of Sync_Seg is the edge phase error, measured in time quanta. If the edge occurs before Sync_Seg, the phase error is negative, else it is positive.

Two types of synchronization exist, Hard Synchronization and Re-synchronization.

A Hard Synchronization is done once at the start of a frame and inside a frame only when Re-synchronizations occur.

- Hard Synchronization
  After a hard synchronization, the bit time is restarted with the end of Sync_Seg, regardless of the edge phase error. Thus hard synchronization forces the edge, which has caused the hard synchronization to lie within the synchronization segment of the restarted bit time.
- Bit Re-synchronization
  Re-synchronization leads to a shortening or lengthening of the bit time such that the position of the sample point is shifted with regard to the edge.

When the phase error of the edge which causes Re-synchronization is positive, Phase_Seg1 is lengthened. If the magnitude of the phase error is less than SJW, Phase_Seg1 is lengthened by the magnitude of the phase error, else it is lengthened by SJW.

When the phase error of the edge, which causes Re-synchronization is negative, Phase_Seg2 is shortened. If the magnitude of the phase error is less than SJW, Phase_Seg2 is shortened by the magnitude of the phase error, else it is shortened by SJW.

When the magnitude of the phase error of the edge is less than or equal to the programmed value of SJW, the results of Hard Synchronization and Re-synchronization are the same. If the magnitude of the phase error is larger than SJW, the Re-synchronization cannot compensate the phase error completely, an error (phase error - SJW) remains.

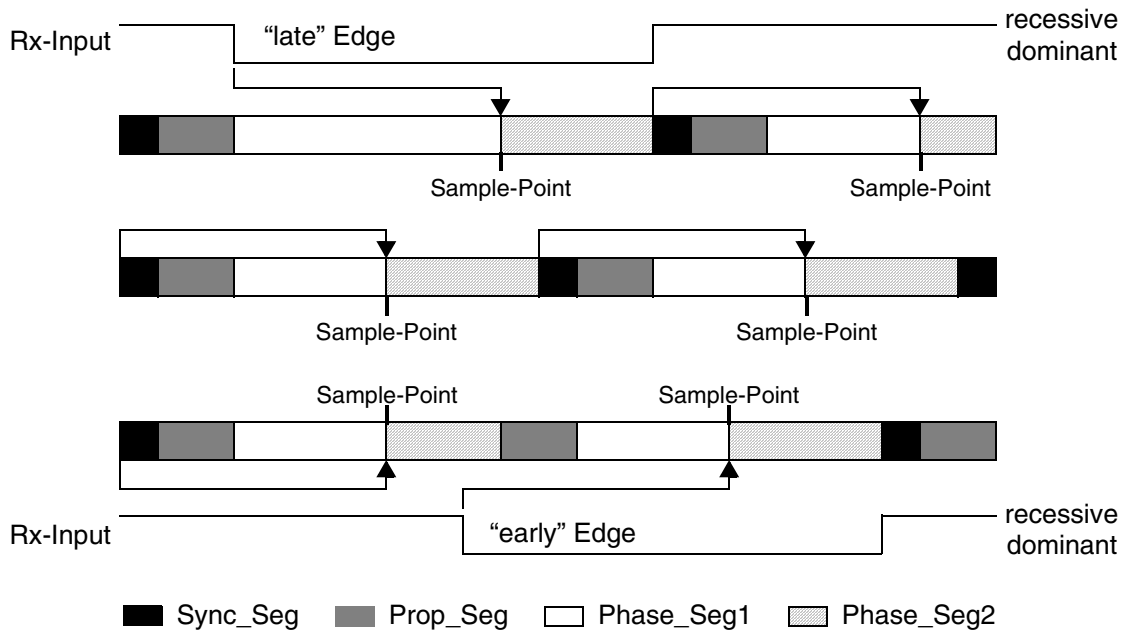
Only one synchronization may be done between two Sample Points. The Synchronizations maintain a minimum distance between edges and Sample Points, giving the bus level time to stabilize and filtering out spikes that are shorter than (Prop_Seg + Phase_Seg1).

Apart from noise spikes, most synchronizations are caused by arbitration. All nodes synchronize “hard” on the edge transmitted by the “leading” transceiver that started transmitting first, but due to propagation delay times, they cannot become ideally synchronized. The “leading” transmitter does not necessarily win the arbitration, therefore the receivers have to synchronize themselves to different transmitters that subsequently “take the lead” and that are differently synchronized to the previously “leading” transmitter. The same happens at the acknowledge field, where the transmitter and some of the receivers will have to synchronize to that receiver that “takes the lead” in the transmission of the dominant acknowledge bit.

Synchronizations after the end of the arbitration will be caused by oscillator tolerance, when the differences in the oscillator’s clock periods of transmitter and receivers sum up during the time between synchronizations (at most ten bits). These summarized differences may not be longer than the SJW, limiting the oscillator’s tolerance range.

The examples in Figure 37 show how the Phase Buffer Segments are used to compensate for phase errors. There are three drawings of each two consecutive bit timings. The upper drawing shows the synchronization on a “late” edge, the lower drawing shows the synchronization on an “early” edge, and the middle drawing is the reference without synchronization.

Figure 37. Synchronization on “late” and “early” Edges



In the first example, an edge from recessive to dominant occurs at the end of Prop_Seg. The edge is “late” since it occurs after the Sync_Seg. Reacting to the “late” edge, Phase_Seg1 is lengthened so that the distance from the edge to the Sample Point is the same as it would have been from the Sync_Seg to the Sample Point if no edge had occurred. The phase error of this “late” edge is less than SJW, so it is fully compensated and the edge from dominant to recessive at the end of the bit, which is one nominal bit time long, occurs in the Sync_Seg.

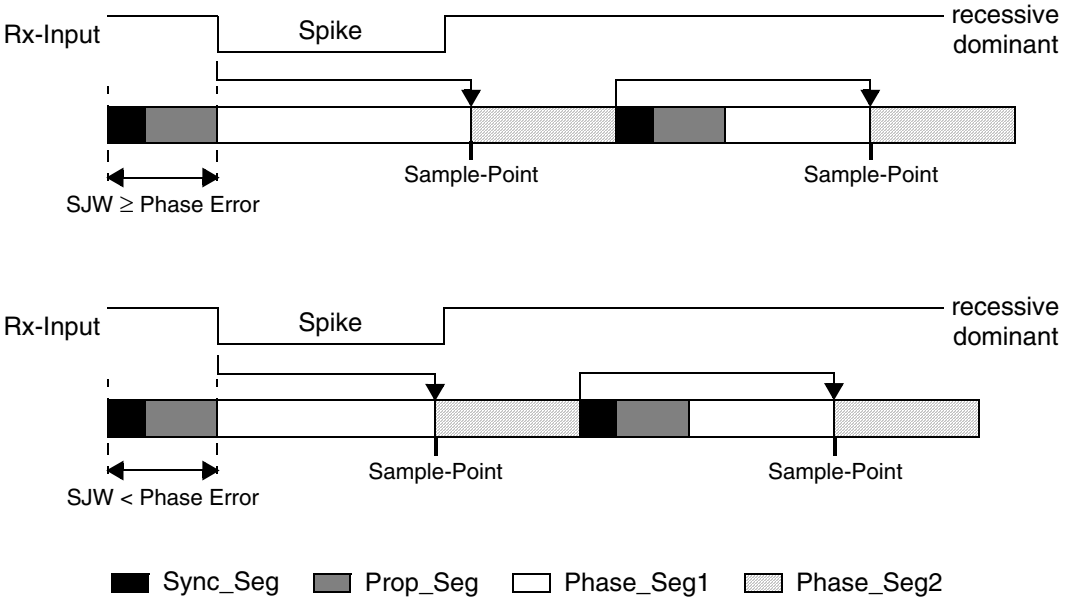
In the second example, an edge from recessive to dominant occurs during Phase_Seg2. The edge is “early” since it occurs before a Sync_Seg. Reacting to the “early” edge, Phase_Seg2 is shortened and Sync_Seg is omitted, so that the distance from the edge to the Sample Point is the same as it would have been from an Sync_Seg to the Sample Point if no edge had occurred. As in the previous example, the magnitude of phase error of this “early” edge’s is less than SJW, so it is fully compensated.

The Phase Buffer Segments are lengthened or shortened temporarily only; at the next bit time, the segments return to their nominal programmed values.

In these examples, the bit timing is seen from the point of view of the CAN state machine, where the bit time starts and ends at the Sample Points. The state machine omits Sync_Seg when synchronizing on an “early” edge, because it cannot subsequently redefine that time quantum of Phase_Seg2 where the edge occurs to be the Sync_Seg.

The examples in Figure 38 show how short dominant noise spikes are filtered by synchronizations. In both examples the spike starts at the end of Prop_Seg and has the length of “Prop_Seg + Phase_Seg1”.

Figure 38. Filtering of Short Dominant Spikes



In the first example, the Synchronization Jump Width is greater than or equal to the phase error of the spike’s edge from recessive to dominant. Therefore the Sample Point is shifted after the end of the spike; a recessive bus level is sampled.

In the second example, SJW is shorter than the phase error, so the Sample Point cannot be shifted far enough; the dominant spike is sampled as actual bus level.

18.7.10.4 Oscillator Tolerance Range

The oscillator tolerance range was increased when the CAN protocol was developed from version 1.1 to version 1.2 (version 1.0 was never implemented in silicon). The option to synchronize on edges from dominant to recessive became obsolete, only edges from recessive to dominant are considered for synchronization. The only CAN controllers to implement protocol version 1.1 have been Intel 82526 and Philips 82C200, both are superseded by successor products. The protocol update to version 2.0 (A and B) had no influence on the oscillator tolerance.

The tolerance range df for an oscillator frequency f_{osc} around the nominal frequency f_{nom} is:
$(1 - df) \bullet f_{nom} \leq f_{osc} \leq (1 + df) \bullet f_{nom}$

It depends on the proportions of Phase_Seg1, Phase_Seg2, SJW, and the bit time. The maximum tolerance df is the defined by two conditions (both shall be met):

$$\text{I:} \quad df \leq \frac{min(\text{Phase\_Seg1}, \text{Phase\_Seg2})}{2 \cdot (13 \cdot \text{bit\_time} - \text{Phase\_Seg2})}$$

$$\text{II:} \quad df \leq \frac{\text{SJW}}{20 \cdot \text{bit\_time}}$$

It has to be considered that SJW may not be larger than the smaller of the Phase Buffer Segments and that the Propagation Time Segment limits that part of the bit time that may be used for the Phase Buffer Segments.

The combination Prop_Seg = 1 and Phase_Seg1 = Phase_Seg2 = SJW = 4 allows the largest possible oscillator tolerance of 1.58%. This combination with a Propagation Time Segment of only 10% of the bit time is not suitable for short bit times; it can be used for bit rates of up to 125 kBit/s (bit time = 8 μs) with a bus length of 40 m.

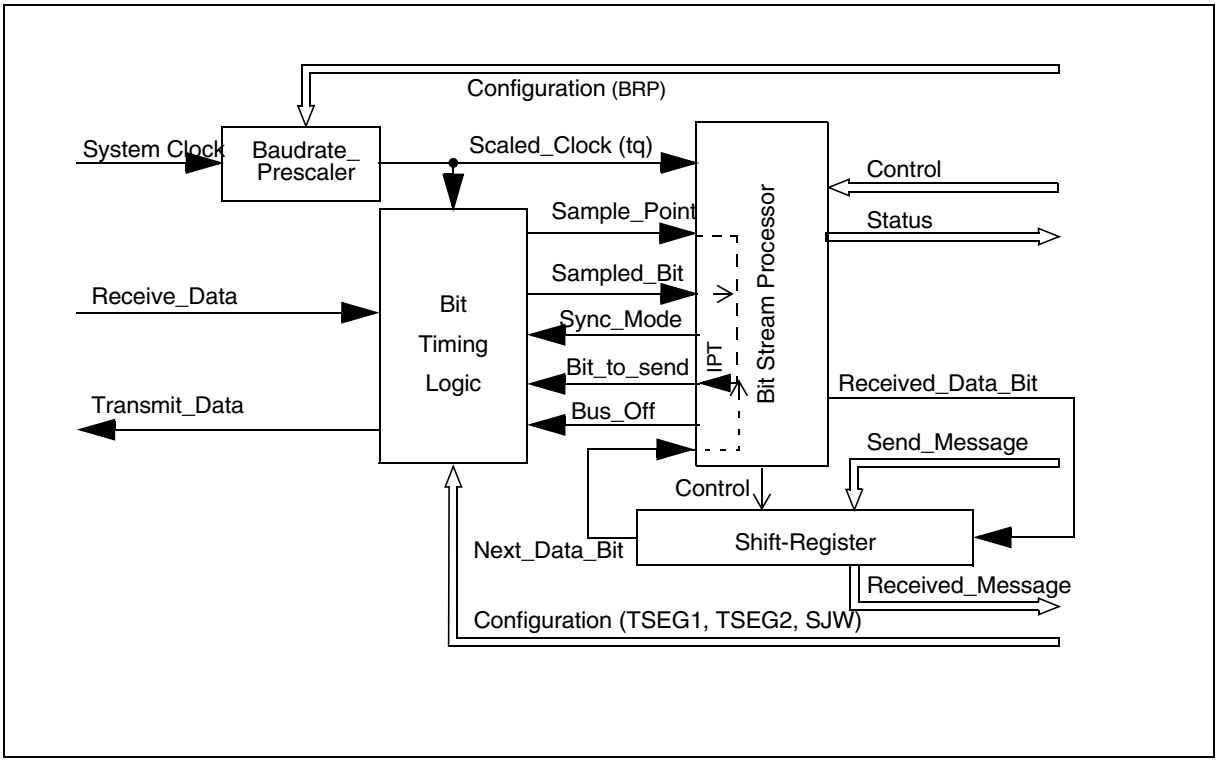
18.7.10.5 Configuring the CAN Protocol Controller

In most CAN implementations and also in the CAN Peripheral, the bit timing configuration is programmed in two register bytes. The sum of Prop_Seg and Phase_Seg1 (as TSEG1) is combined with Phase_Seg2 (as TSEG2) in one byte, SJW and BRP are combined in the other byte (see Figure 39 on page 257).

In these bit timing registers, the four components TSEG1, TSEG2, SJW, and BRP have to be programmed to a numerical value that is one less than its functional value. Therefore, instead of values in the range of [1..n], values in the range of [0..n-1] are programmed. That way, e.g. SJW (functional range of [1..4]) is represented by only two bits.

Therefore the length of the bit time is (programmed values) [TSEG1 + TSEG2 + 3] t_q or (functional values) [Sync_Seg + Prop_Seg + Phase_Seg1 + Phase_Seg2] t_q.

Figure 39. Structure of the CAN Core's CAN Protocol Controller



The data in the bit timing registers is the configuration input of the CAN protocol controller. The Baud Rate Prescaler (configured by BRP) defines the length of the time quantum, the basic time unit of the bit time; the Bit Timing Logic (configured by TSEG1, TSEG2, and SJW) defines the number of time quanta in the bit time.

The processing of the bit time, the calculation of the position of the Sample Point, and occasional synchronizations are controlled by the BTL state machine, which is evaluated once

each time quantum. The rest of the CAN protocol controller, the BSP state machine is evaluated once each bit time, at the Sample Point.

The Shift Register sends the messages serially and receives the messages parallely. Its loading and shifting is controlled by the BSP.

The BSP translates messages into frames and vice versa. It generates and discards the enclosing fixed format bits, inserts and extracts stuff bits, calculates and checks the CRC code, performs the error management, and decides which type of synchronization is to be used. It is evaluated at the Sample Point and processes the sampled bus input bit. The time that is needed to calculate the next bit to be sent after the Sample point(e.g. data bit, CRC bit, stuff bit, error flag, or idle) is called the Information Processing Time (IPT).

The IPT is application specific but may not be longer than 2 t_q; the IPT for the CAN Peripheral is 0 t_q. Its length is the lower limit of the programmed length of Phase_Seg2. In case of a synchronization, Phase_Seg2 may be shortened to a value less than IPT, which does not affect bus timing.

18.7.10.6 Calculating Bit Timing Parameters

Usually, the calculation of the bit timing configuration starts with a desired bit rate or bit time. The resulting bit time (1/bit rate) must be an integer multiple of the system clock period.

The bit time may consist of 4 to 25 time quanta, the length of the time quantum t_q is defined by the Baud Rate Prescaler with t_q = (Baud Rate Prescaler)/f_{sys}. Several combinations may lead to the desired bit time, allowing iterations of the following steps.

First part of the bit time to be defined is the Prop_Seg. Its length depends on the delay times measured in the system. A maximum bus length as well as a maximum node delay has to be defined for expandible CAN bus systems. The resulting time for Prop_Seg is converted into time quanta (rounded up to the nearest integer multiple of t_q).

The Sync_Seg is 1 t_q long (fixed), leaving (bit time – Prop_Seg – 1) t_q for the two Phase Buffer Segments. If the number of remaining t_q is even, the Phase Buffer Segments have the same length, Phase_Seg2 = Phase_Seg1, else Phase_Seg2 = Phase_Seg1 + 1.

The minimum nominal length of Phase_Seg2 has to be regarded as well. Phase_Seg2 may not be shorter than the IPT of the CAN controller, which, depending on the actual implementation, is in the range of [0..2] t_q.

The length of the Synchronization Jump Width is set to its maximum value, which is the minimum of 4 and Phase_Seg1.

The oscillator tolerance range necessary for the resulting configuration is calculated by the formulas given in Section 18.7.10.4: *Oscillator Tolerance Range on page 256*

If more than one configuration is possible, that configuration allowing the highest oscillator tolerance range should be chosen.

CAN nodes with different system clocks require different configurations to come to the same bit rate. The calculation of the propagation time in the CAN network, based on the nodes with the longest delay times, is done once for the whole network.

The oscillator tolerance range of the CAN systems is limited by that node with the lowest tolerance range.

The calculation may show that bus length or bit rate have to be decreased or that the stability of the oscillator frequency has to be increased in order to find a protocol compliant configuration of the CAN bit timing.

The resulting configuration is written into the Bit Timing Register:

(Phase_Seg2-1)&(Phase_Seg1+Prop_Seg-1)&

(SynchronisationJumpWidth-1)&(Prescaler-1)

Example for Bit Timing at High Baudrate

In this example, the frequency of APB_CLK is 10 MHz, BRP is 0, the bit rate is 1 MBit/s.

t_q	100	ns	$= t_{APB\_CLK}$
delay of bus driver	50	ns	
delay of receiver circuit	30	ns	
delay of bus line (40m)	220	ns	
t_{Prop}	600	ns	$= 6 \bullet t_q$
t_{SJW}	100	ns	$= 1 \bullet t_q$
t_{TSeg1}	700	ns	$= t_{Prop} + t_{SJW}$
t_{TSeg2}	200	ns	$=$ Information Processing Time $+ 1 \bullet t_q$
$t_{Sync\text{-}Seg}$	100	ns	$= 1 \bullet t_q$
bit time	1000	ns	$= t_{Sync\text{-}Seg} + t_{TSeg1} + t_{TSeg2}$
tolerance for APB_CLK	0.39	%	$= \frac{min(PB1, PB2)}{2x(13x(\text{bit time} - PB2))}$
			$= \frac{0.1\mu s}{2x(13x(1\mu s - 0.2\mu s))}$

In this example, the concatenated bit time parameters are $(2\text{-}1)_3\&(7\text{-}1)_4\&(1\text{-}1)_2\&(1\text{-}1)_6$, the Bit Timing Register is programmed to= 0x1600.

Example for Bit Timing at Low Baudrate

In this example, the frequency of APB_CLK is 2 MHz, BRP is 1, the bit rate is 100 KBit/s.

t_q	1	µs	$= 2 \bullet t_{APB\_CLK}$
delay of bus driver	200	ns	
delay of receiver circuit	80	ns	
delay of bus line (40m)	220	ns	
t_{Prop}	1	µs	$= 1 \bullet t_q$
t_{SJW}	4	µs	$= 4 \bullet t_q$
t_{TSeg1}	5	µs	$= t_{Prop} + t_{SJW}$
t_{TSeg2}	4	µs	= Information Processing Time + $3 \bullet t_q$
$t_{Sync\text{-}Seg}$	1	µs	$= 1 \bullet t_q$
bit time	10	µs	$= t_{Sync\text{-}Seg} + t_{TSeg1} + t_{TSeg2}$
tolerance for APB_CLK	1.58	%	$= \frac{min(PB1, PB2)}{2x(13x(\text{bit time} - PB2))}$
			$= \frac{4\mu s}{2x(13x(10\mu s - 4\mu s))}$

In this example, the concatenated bit time parameters are $(4\text{-}1)_3\&(5\text{-}1)_4\&(4\text{-}1)_2\&(2\text{-}1)_6$, the Bit Timing Register is programmed to= 0x34C1.

18.7.11 Register Map

A summary of the CAN Peripheral registers is given in the following table.

Refer to Table 20 on page 49 for the base address..

Table 57. CAN Register Map

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00h	CAN_CR	Reserved								Test	CCE	DAR	res	EIE	SIE	IE	Init
04h	CAN_SR	Reserved								BOff	EWarn	EPass	RxOk	TxOk	LEC		
08h	CAN_ERR	RP	REC6-0							TEC7-0							
0Ch	CAN_BTR	res	TSeg2			TSeg1				SJW		BRP					
10h	CAN_IDR	IntId15-8								IntId7-0							
14h	CAN_TESTR	Reserved								Rx	Tx1	Tx0	LBack	Silent	Basic	Reserved	
18h	CAN_BRPR	Reserved												BRPE			
20h	CAN_IF1_CRR	Busy	Reserved							Message Number							
24h	CAN_IF1_CMR	Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/	Data A	Data B
28h	CAN_IF1_M1R	Msk15-0															
2Ch	CAN_IF1_M2R	MXtd	MDir	res	Msk28-16												
30h	CAN_IF1_A1R	ID15-0															
34h	CAN_IF1_A2R	Msg Val	Xtd	Dir	ID28-16												
38h	CAN_IF1_MCR	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC3-0			
3Ch	CAN_IF1_DA1R	Data(1)								Data(0)							
40h	CAN_IF1_DA2R	Data(3)								Data(2)							
44h	CAN_IF1_DB1R	Data(5)								Data(4)							
48h	CAN_IF1_DB2R	Data(7)								Data(6)							
80h	CAN_IF2_CRR	Busy	Reserved							Message Number							

Table 57. CAN Register Map

Addr offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
84h	CAN_IF2_CMR	Reserved								WR/RD	Mask	Arb	Control	ClrIntPnd	TxRqst/	Data A	Data B
88h	CAN_IF2_M1R	Msk15-0															
8Ch	CAN_IF2_M2R	MXtd	MDir	res	Msk28-16												
90h	CAN_IF2_A1R	ID15-0															
94h	CAN_IF2_A2R	Msg Val	Xtd	Dir	ID28-16												
98h	CAN_IF2_MCR	NewDat	MsgLst	IntPnd	UMask	TxIE	RxIE	RmtEn	TxRqst	EoB	Reserved			DLC3-0			
9Ch	CAN_IF2_DA1R	Data(1)								Data(0)							
A0h	CAN_IF2_DA2R	Data(3)								Data(2)							
A4h	CAN_IF2_DB1R	Data(5)								Data(4)							
A8h	CAN_IF2_DB2R	Data(7)								Data(6)							
100h	CAN_TxR1R	TxRqst16-1															
104h	CAN_TxR2R	TxRqst32-17															
120h	CAN_ND1R	NewDat16-1															
124h	CAN_ND2R	NewDat32-17															
140h	CAN_IP1R	IntPnd16-1															
144h	CAN_IP2R	IntPnd32-17															
160h	CAN_MV1R	MsgVal16-1															
164h	CAN_MV2R	MsgVal32-17															

19 USB SLAVE INTERFACE (USB)

19.1 Introduction

The USB Peripheral implements an interface between a full-speed USB 2.0 bus and the APB bus.

USB suspend/resume are supported which allows to stop the device clocks for low power consumption.

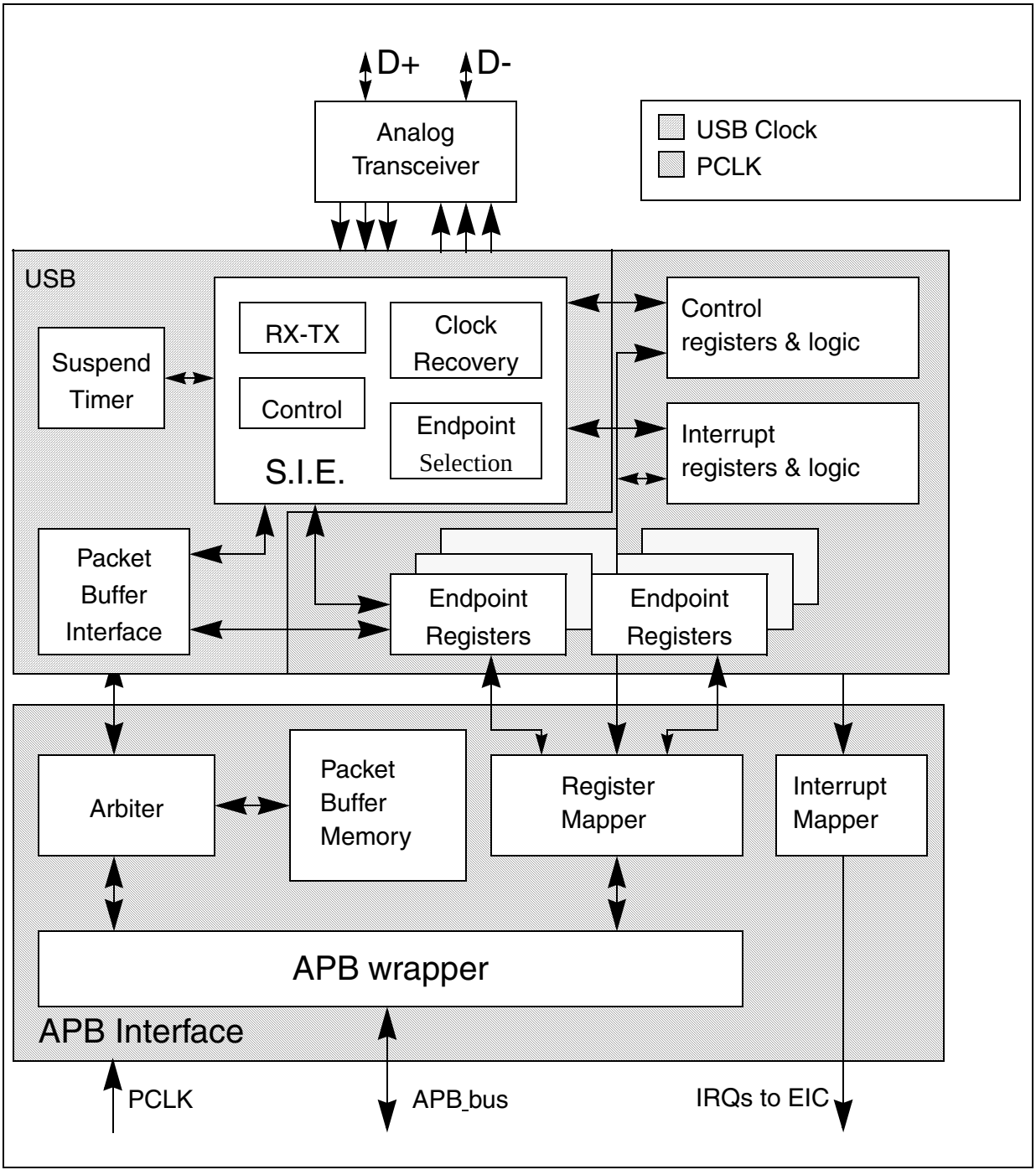
19.2 Main Features

- Configurable number of endpoints from 1 to 8.
- Cyclic Redundancy Check (CRC) generation/checking, Non-Return-to-Zero Inverted (NRZI) encoding/decoding and bit-stuffing.
- Isochronous transfers support.
- Double-buffered bulk endpoint support.
- USB Suspend/Resume operations.
- Frame locked clock pulse generation.

19.3 Block Diagram

Figure 40 shows the block diagram of the USB Peripheral.

Figure 40. USB Peripheral Block Diagram



19.4 Functional Description

The USB Peripheral provides an USB compliant connection between the host PC and the function implemented by the microcontroller. Data transfer between the host PC and the system memory occurs through a dedicated packet buffer memory accessed directly by the USB Peripheral. The size of this dedicated buffer memory must be according to the number of endpoints used and the maximum packet size. In this implementation, the dedicated memory of 512 Byte and up to 8 endpoints can be used. The USB Peripheral interfaces with the USB host, detecting token packets, handling data transmission/reception, and processing handshake packets as required by the USB standard. Transaction formatting is performed by the hardware, including CRC generation and checking.

Each endpoint is associated with a buffer description block indicating where the endpoint related memory area is located, how large it is or how many bytes must be transmitted. When a token for a valid function/endpoint pair is recognized by the USB Peripheral, the related data transfer (if required and if the endpoint is configured) takes place. The data buffered by the USB Peripheral is loaded in an internal 16 bit register and memory access to the dedicated buffer is performed. When all the data has been transferred, if needed, the proper handshake packet over the USB is generated or expected according to the direction of the transfer.

At the end of the transaction, an endpoint-specific interrupt is generated, reading status registers and/or using different interrupt response routines. The microcontroller can determine:

- which endpoint has to be served
- which type of transaction took place, if errors occurred (bit stuffing, format, CRC, protocol, missing ACK, over/underrun, etc).

Two interrupt lines are generated by the USB Peripheral : one IRQ collecting high priority endpoint interrupts (isochronous and double-buffered bulk) and another IRQ collecting all other interrupt sources (check the IRQ interrupt vector table for detailed interrupt source mapping).

Special support is offered to Isochronous transfers and high throughput bulk transfers, implementing a double buffer usage, which allows to always have an available buffer for the USB Peripheral while the microcontroller uses the other one.

The unit can be placed in low-power mode (SUSPEND mode), by writing in the control register, whenever required. At this time, all static power dissipation is avoided, and the USB clock can be slowed down or stopped. The detection of activity at the USB inputs, while in low-power mode, wakes the device up asynchronously. A special interrupt source can be connected directly to a wake-up line to allow the system to immediately restart the normal clock generation and/or support direct clock start/stop.

19.4.1 Description of USB Blocks

The USB Peripheral implements all the features related to USB interfacing, which include the following blocks:

- Serial Interface Engine (SIE): The functions of this block include: synchronization pattern recognition, bit-stuffing, CRC generation and checking, PID verification/generation, and handshake evaluation. It must interface with the USB transceivers and uses the virtual buffers provided by the packet buffer interface for local data storage,. This unit also generates signals according to USB Peripheral events, such as Start of Frame (SOF), USB_Reset, Data errors etc. and to Endpoint related events like end of transmission or correct reception of a packet; these signals are then used to generate interrupts.
- Suspend Timer: This block generates the frame locked clock pulse for any external device requiring Start-of-Frame synchronization and it detects a global suspend (from the host) when no traffic has been received for 3 mS.
- Packet Buffer Interface: This block manages the local memory implementing a set of buffers in a flexible way, both for transmission and reception. It can choose the proper buffer according to requests coming from the SIE and locate them in the memory addresses pointed by the Endpoint registers. It increments the address after each exchanged word until the end of packet, keeping track of the number of exchanged bytes and preventing the buffer to overrun the maximum capacity.
- Endpoint-Related Registers: Each endpoint has an associated register containing the endpoint type and its current status. For mono-directional/single-buffer endpoints, a single register can be used to implement two distinct endpoints. In this implementation, the number of registers is 8, allowing up to 8 double-buffer endpoints or up to 16 mono-directional/single-buffer ones in any combination.
- Control Registers: These are the registers containing information about the status of the whole USB Peripheral and used to force some USB events, such as resume and power-down.
- Interrupt Registers: These contain the Interrupt masks and a record of the events. They can be used to inquire an interrupt reason, the interrupt status or to clear the status of a pending interrupt.

The USB Peripheral is connected to the APB bus through an APB interface, containing the following blocks:

- Packet Memory: This is the local memory that physically contains the Packet Buffers. It can be used by the Packet Buffer interface, which creates the data structure and can be accessed directly by the application software. The size of the Packet Memory is 512 Bytes, structured as 256 words by 16 bits.
- Arbiter: This block accepts memory requests coming from the APB bus and from the USB interface. It resolves the conflicts by giving priority to APB accesses, while always reserving half of the memory bandwidth to complete all USB transfers. This time-duplex

scheme implements a virtual dual-port RAM that allows memory access, while an USB transaction is happening. Multi-word APB transfers of any length are also allowed by this scheme.

- Register Mapper: This block collects the various byte-wide and bit-wide registers of the USB Peripheral in a structured 16-bit wide word set addressed by the APB.
- Interrupt Mapper: This block is used to select how the possible USB events can generate interrupts and map them to IRQ lines of the EIC.
- APB Wrapper: This provides an interface to the APB for the memory and register. It also maps the whole USB Peripheral in the APB address space.

19.5 Programming Considerations

In the following sections, the expected interactions between the USB Peripheral and the application program are described, in order to ease application software development.

19.5.1 Generic USB Device Programming

This part describes the main tasks required of the application software in order to obtain USB compliant behaviour. The actions related to the most general USB events are taken into account and paragraphs are dedicated to the special cases of double-buffered endpoints and Isochronous transfers. Apart from system reset, action is always initiated by the USB Peripheral, driven by one of the USB events described below.

19.5.2 System and Power-On Reset

Upon system and power-on reset, the first operation the application software should perform is to provide all required clock signals to the USB Peripheral and subsequently de-assert its reset signal so to be able to access its registers. The whole initialization sequence is hereafter described.

As a first step application software needs to activate register macrocell clock and de-assert macrocell specific reset signal using related control bits provided by device clock management logic.

After that the analog part of the device related to the USB transceiver must be switched on using the PDWN bit in CNTR register which requires a special handling. This bit is intended to switch on the internal voltage references supplying the port transceiver . Since this circuits have a defined start-up time, during which the behaviour of USB transceiver is not defined, it is necessary to wait this time, after having set the PDWN bit in CNTR register, then the reset condition on the USB part can be removed (clearing of FRES bit in CNTR register) and the ISTR register can be cleared, removing any spurious pending interrupt, before enabling any other macrocell operation.

As a last step the USB specific 48 MHz clock needs to be activated, using the related control bits provided by device clock management logic, where applicable.

At system reset, the microcontroller must initialize all required registers and the packet buffer description table, to make the USB Peripheral able to properly generate interrupts and data transfers. All registers not specific to any endpoint must be initialized according to the needs of application software (choice of enabled interrupts, chosen address of packet buffers, etc.). Then the process continues as for the USB reset case (see further paragraph).

19.5.2.1 USB Reset (RESET Interrupt)

When this event occurs, the USB Peripheral is put in the same conditions it is left by the system reset after the initialization described in the previous paragraph: the USB_DADDR register is reset, and communication is disabled in all endpoint registers (the USB Peripheral will not respond to any packet). As a response to the USB reset event, the USB function must be enabled, having as USB address 0, implementing only the default control endpoint (endpoint address is 0 too). This is accomplished by setting the Enable Function (EF) bit of the USB_DADDR register and initializing the EP0R register and its related packet buffers accordingly. During USB enumeration process, the host assigns a unique address to this device, which must be written in the ADD[6:0] bits of the USB_DADDR register, and configures any other necessary endpoint.
When a RESET interrupt is received, the application software is responsible to enable again the default endpoint of USB function 0 within 10mS from the end of reset sequence which triggered the interrupt.

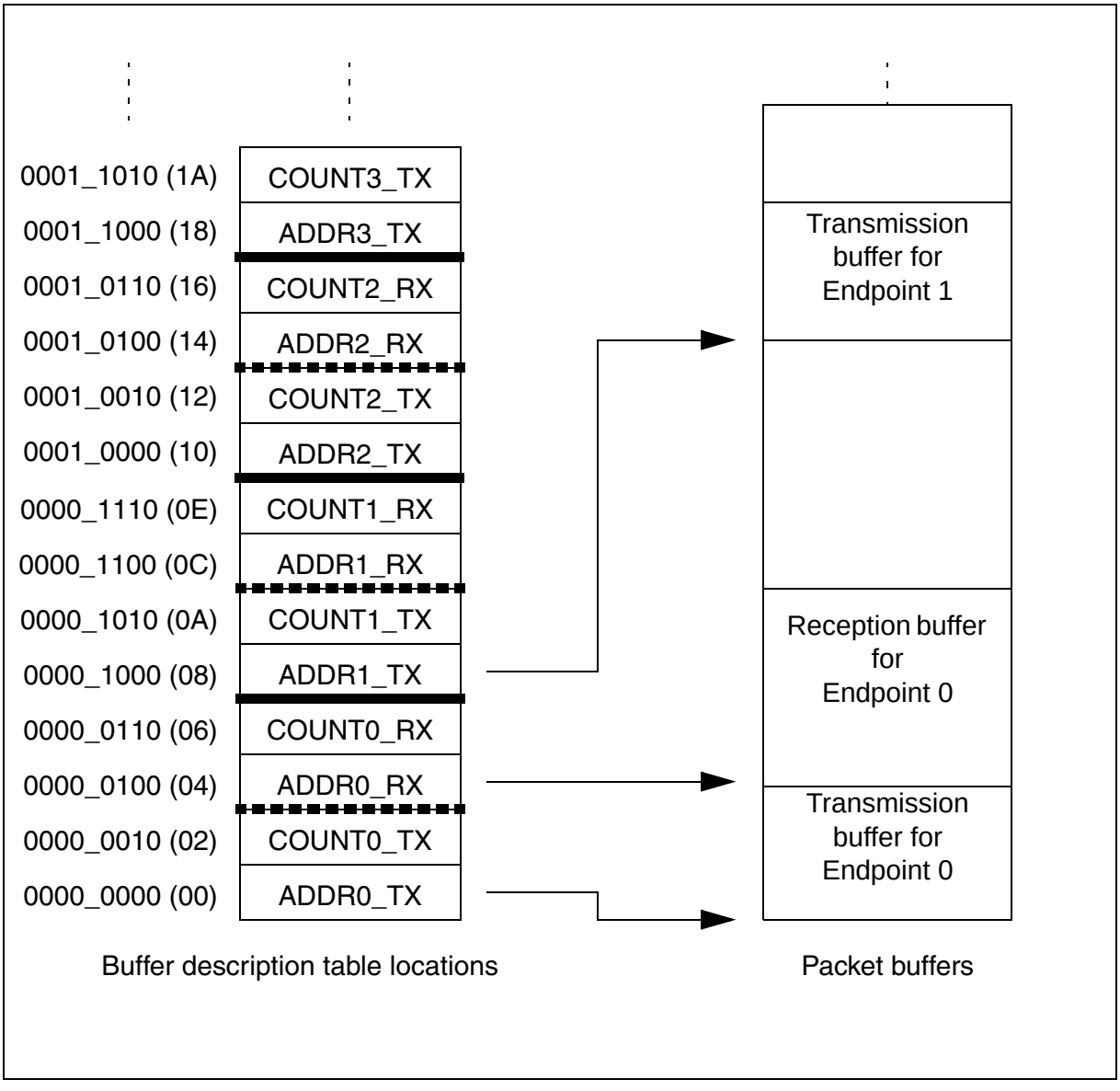
19.5.2.2 Structure and Usage of Packet Buffers

Each bidirectional endpoint may receive or transmit data from/to the host. The received data is stored in a dedicated memory buffer reserved for that endpoint, while another memory buffer contains the data to be transmitted by the endpoint. Access to this memory is performed by the packet buffer interface block, which delivers a memory access request and waits for its acknowledgement. Since the packet buffer memory has to be accessed by the microcontroller also, an arbitration logic takes care of the access conflicts, using half APB cycle for microcontroller access and the remaining half for the USB Peripheral access. In this way, both the agents can operate as if the packet memory is a dual-port RAM, without being aware of any conflict even when the microcontroller is performing back-to-back accesses. The USB Peripheral logic uses a dedicated clock. The frequency of this dedicated clock is fixed by the requirements of the USB standard at 48 MHz, and this can be different from the clock used for the interface to the APB bus. Different clock configurations are possible where the APB clock frequency can be higher or lower than the USB Peripheral one. However, due to USB data rate and packet memory interface requirements, the APB clock frequency must be greater than 8 MHz to avoid data overrun/underrun problems.

Each endpoint is associated with two packet buffers (usually one for transmission and the other one for reception). The size of the buffer can be upto 512 words each. Buffers can be placed anywhere inside the packet memory because their location and size is specified in a buffer description table, which is also located in the packet memory at the address indicated by the USB_BTABLE register. Each table entry is associated to an endpoint register and it is composed of four 16-bit words so that table start address must always be aligned to an 8-byte

boundary (the lowest three bits of USB_BTABLE register are always “000”). Buffer descriptor table entries are described in the Section “Buffer Descriptor Table”. If an endpoint is unidirectional and it is neither an Isochronous nor a double-buffered bulk, only one packet buffer is required (the one related to the supported transfer direction). Other table locations related to unsupported transfer directions or unused endpoints, are available to the user. isochronous and double-buffered bulk endpoints have special handling of packet buffers (Refer to “Isochronous Transfers” and “Double-Buffered Endpoints” respectively). The relationship between buffer description table entries and packet buffer areas is depicted in Figure 41.

Figure 41. Packet Buffer Areas and Buffer Description Table Locations



Each packet buffer is used either during reception or transmission starting from the bottom. The USB Peripheral will never change the contents of memory locations adjacent to the allocated memory buffers; if a packet bigger than the allocated buffer length is received (buffer overrun condition) the data will be copied to the memory only up to the last available location.

19.5.2.3 Endpoint Initialization

The first step to initialize an endpoint is to write appropriate values to the ADDRn_TX/ ADDRn_RX registers so that the USB Peripheral finds the data to be transmitted already available and the data to be received can be buffered. The EP_TYPE bits in the USB_EPnR register must be set according to the endpoint type, eventually using the EP_KIND bit to enable any special required feature. On the transmit side, the endpoint must be enabled using the STAT_TX bits in the USB_EPnR register and COUNTn_TX must be initialized. For reception, STAT_RX bits must be set to enable reception and COUNTn_RX must be written with the allocated buffer size using the BL_SIZE and NUM_BLOCK fields. Unidirectional endpoints, except Isochronous and double-buffered bulk endpoints, need to initialize only bits and registers related to the supported direction. Once the transmission and/or reception are enabled, register USB_EPnR and locations ADDRn_TX/ADDRn_RX, COUNTn_TX/ COUNTn_RX (respectively), should not be modified by the application software, as the hardware can change their value on the fly. When the data transfer operation is completed, notified by a CTR interrupt event, they can be accessed again to re-enable a new operation.

19.5.2.4 IN Packets (Data Transmission)

When receiving an IN token packet, if the received address matches a configured and valid endpoint one, the USB Peripheral accesses the contents of ADDRn_TX and COUNTn_TX locations inside buffer descriptor table entry related to the addressed endpoint. The content of these locations is stored in its internal 16 bit registers ADDR and COUNT (not accessible by software). The packet memory is accessed again to read the first word to be transmitted (Refer to Section “Structure and Usage of Packet Buffers”) and starts sending a DATA0 or DATA1 PID according to USB_EPnR bit DTOG_TX. When the PID is completed, the first byte from the word, read from buffer memory, is loaded into the output shift register to be transmitted on the USB bus. After the last data byte is transmitted, the computed CRC is sent. If the addressed endpoint is not valid, a NAK or STALL handshake packet is sent instead of the data packet, according to STAT_TX bits in the USB_EPnR register.

The ADDR internal register is used as a pointer to the current buffer memory location while COUNT is used to count the number of remaining bytes to be transmitted. Each word read from the packet buffer memory is transmitted over the USB bus starting from the least significant byte. Transmission buffer memory is read starting from the address pointed by ADDRn_TX for COUNTn_TX/2 words. If a transmitted packet is composed of an odd number of bytes, only the lower half of the last word accessed will be used.

On receiving the ACK receipt by the host, the USB_EPnR register is updated in the following way: DTOG_TX bit is toggled, the endpoint is made invalid by setting STAT_TX=10 (NAK) and bit CTR_TX is set. The application software must first identify the endpoint, which is

requesting microcontroller attention by examining the EP_ID and DIR bits in the USB_ISTR register. Servicing of the CTR_TX event starts clearing the interrupt bit; the application software then prepares another buffer full of data to be sent, updates the COUNTn_TX table location with the number of byte to be transmitted during the next transfer, and finally sets STAT_TX to '11' (VALID) to re-enable transmissions. While the STAT_TX bits are equal to '10' (NAK), any IN request addressed to that endpoint is NAKed, indicating a flow control condition: the USB host will retry the transaction until it succeeds. It is mandatory to execute the sequence of operations in the above mentioned order to avoid losing the notification of a second IN transaction addressed to the same endpoint immediately following the one which triggered the CTR interrupt.

19.5.2.5 OUT and SETUP Packets (Data Reception)

These two tokens are handled by the USB Peripheral more or less in the same way; the differences in the handling of SETUP packets are detailed in the following paragraph about control transfers. When receiving an OUT/SETUP PID, if the address matches a valid endpoint, the USB Peripheral accesses the contents of the ADDRn_RX and COUNTn_RX locations inside the buffer descriptor table entry related to the addressed endpoint. The content of the ADDRn_RX is stored directly in its internal register ADDR. While COUNT is now reset and the values of BL_SIZE and NUM_BLOCK bit field**s,** which are read within COUNTn_RX content are used to initialize BUF_COUNT, an internal 16 bit counter, which is used to check the buffer overrun condition (all these internal registers are not accessible by software). Data bytes subsequently received by the USB Peripheral are packed in words (the first byte received is stored as least significant byte) and then transferred to the packet buffer starting from the address contained in the internal ADDR register while BUF_COUNT is decremented and COUNT is incremented at each byte transfer. When the end of DATA packet is detected, the correctness of the received CRC is tested and only if no errors occurred during the reception, an ACK handshake packet is sent back to the transmitting host. In case of wrong CRC or other kinds of errors (bit-stuff violations, frame errors, etc.), data bytes are anyways copied in the packet memory buffer, at least until the error detection point, but ACK packet is not sent and the ERR bit in USB_ISTR register is set. However, there is usually no software action required in this case: the USB Peripheral recovers from reception errors and remains ready for the next transaction to come. If the addressed endpoint is not valid, a NAK or STALL handshake packet is sent instead of the ACK, according to bits STAT_RX in the USB_EPnR register and no data is written in the reception memory buffers.

Reception memory buffer locations are written starting from the address contained in the ADDRn_RX for a number of bytes corresponding to the received data packet length, CRC included (i.e. data payload length + 2), or up to the last allocated memory location, as defined by BL_SIZE and NUM_BLOCK, whichever comes first. In this way, the USB Peripheral never writes beyond the end of the allocated reception memory buffer area. If the length of the data packet payload (actual number of bytes used by the application) is greater than the allocated buffer, the USB Peripheral detects a buffer overrun condition. in this case, a STALL handshake is sent instead of the usual ACK to notify the problem to the host, no interrupt is generated and the transaction is considered failed.

When the transaction is completed correctly, by sending the ACK handshake packet, the internal COUNT register is copied back in the COUNTn_RX location inside the buffer description table entry, leaving unaffected BL_SIZE and NUM_BLOCK fields, which normally do not require to be re-written, and the USB_EPnR register is updated in the following way: DTOG_RX bit is toggled, the endpoint is made invalid by setting STAT_RX = '10' (NAK) and bit CTR_RX is set. If the transaction has failed due to errors or buffer overrun condition, none of the previously listed actions take place. The application software must first identify the endpoint, which is requesting microcontroller attention by examining the EP_ID and DIR bits in the USB_ISTR register. The CTR_RX event is serviced by first determining the transaction type (SETUP bit in the USB_EPnR register); the application software must clear the interrupt flag bit and get the number of received bytes reading the COUNTn_RX location inside the buffer description table entry related to the endpoint being processed. After the received data is processed, the application software should set the STAT_RX bits to '11' (Valid) in the USB_EPnR, enabling further transactions. While the STAT_RX bits are equal to '10' (NAK), any OUT request addressed to that endpoint is NAKed, indicating a flow control condition: the USB host will retry the transaction until it succeeds. It is mandatory to execute the sequence of operations in the above mentioned order to avoid losing the notification of a second OUT transaction addressed to the same endpoint following immediately the one which triggered the CTR interrupt.

19.5.2.6 Control Transfers

Control transfers are made of a SETUP transaction, followed by zero or more data stages, all of the same direction, followed by a status stage (a zero-byte transfer in the opposite direction). SETUP transactions are handled by control endpoints only and are very similar to OUT ones (data reception) except that the values of DTOG_TX and DTOG_RX bits of the addressed endpoint registers are set to 1 and 0 respectively, to initialize the control transfer, and both STAT_TX and STAT_RX are set to '10' (NAK) to let software decide if subsequent transactions must be IN or OUT depending on the SETUP contents. A control endpoint must check SETUP bit in the USB_EPnR register at each CTR_RX event to distinguish normal OUT transactions from SETUP ones. A USB device can determine the number and direction of data stages by interpreting the data transferred in the SETUP stage, and is required to STALL the transaction in the case of errors. To do so, at all data stages before the last, the unused direction should be set to STALL, so that, if the host reverses the transfer direction too soon, it gets a STALL as a status stage. While enabling the last data stage, the opposite direction should be set to NAK, so that, if the host reverses the transfer direction (to perform the status stage) immediately, it is kept waiting for the completion of the control operation. If the control operation completes successfully, the software will change NAK to VALID, otherwise to STALL. At the same time, if the status stage will be an OUT, the STATUS_OUT (EP_KIND in the USB_EPnR register) bit should be set, so that an error is generated if a status transaction is performed with not-zero data. When the status transaction is serviced, the application clears the STATUS_OUT bit and sets STAT_RX to VALID (to accept a new command) and STAT_TX to NAK (to delay a possible status stage immediately following the next setup).

Since the USB specification states that a SETUP packet cannot be answered with a handshake different from ACK, eventually aborting a previously issued command to start the new one, the USB logic doesn't allow a control endpoint to answer with a NAK or STALL packet to a SETUP token received from the host.

When the STAT_RX bits are set to '01' (STALL) or '10' (NAK) and a SETUP token is received, the USB accepts the data, performing the required data transfers and sends back an ACK handshake. If that endpoint has a previously issued CTR_RX request not yet acknowledged by the application (i.e. CTR_RX bit is still set from a previously completed reception), the USB discards the SETUP transaction and does not answer with any handshake packet regardless of its state, simulating a reception error and forcing the host to send the SETUP token again. This is done to avoid losing the notification of a SETUP transaction addressed to the same endpoint immediately following the transaction, which triggered the CTR_RX interrupt.

19.5.3 Double-Buffered Endpoints

All different endpoint types defined by the USB standard represent different traffic models, and describe the typical requirements of different kind of data transfer operations. When large portions of data are to be transferred between the host PC and the USB function, the bulk endpoint type is the most suited model. This is because the host schedules bulk transactions so as to fill all the available bandwidth in the frame, maximizing the actual transfer rate as long as the USB function is ready to handle a bulk transaction addressed to it. If the USB function is still busy with the previous transaction when the next one arrives, it will answer with a NAK handshake and the host PC will issue the same transaction again until the USB function is ready to handle it, reducing the actual transfer rate due to the bandwidth occupied by re-transmissions. For this reason, a dedicated feature called 'double-buffering' can be used with bulk endpoints.

When 'double-buffering' is activated, data toggle sequencing is used to select, which buffer is to be used by the USB Peripheral to perform the required data transfers, using both 'transmission' and 'reception' packet memory areas to manage buffer swapping on each successful transaction in order to always have a complete buffer to be used by the application, while the USB Peripheral fills the other one. For example, during an OUT transaction directed to a 'reception' double-buffered bulk endpoint, while one buffer is being filled with new data coming from the USB host, the other one is available for the microcontroller software usage (the same would happen with a 'transmission' double-buffered bulk endpoint and an IN transaction).

Since the swapped buffer management requires the usage of all 4 buffer description table locations hosting the address pointer and the length of the allocated memory buffers, the USB_EPnR registers used to implement double-buffered bulk endpoints are forced to be used as uni-directional ones. Therefore, only one STAT bit pair must be set at a value different from '00' (Disabled): STAT_RX if the double-buffered bulk endpoint is enabled for reception, STAT_TX if the double-buffered bulk endpoint is enabled for transmission. In case it is required to have double-buffered bulk endpoints enabled both for reception and transmission, two USB_EPnR registers must be used.

To exploit the double-buffering feature and reach the highest possible transfer rate, the endpoint flow control structure, described in previous chapters, has to be modified, in order to switch the endpoint status to NAK only when a buffer conflict occurs between the USB Peripheral and application software, instead of doing it at the end of each successful transaction. The memory buffer which is currently being used by the USB Peripheral is defined by the DTOG bit related to the endpoint direction: DTOG_RX (bit 14 of USB_EPnR register) for 'reception' double-buffered bulk endpoints or DTOG_TX (bit 6 of USB_EPnR register) for 'transmission' double-buffered bulk endpoints. To implement the new flow control scheme, the USB Peripheral should know which packet buffer is currently in use by the application software, so to be aware of any conflict. Since in the USB_EPnR register, there are two DTOG bits but only one is used by USB Peripheral for data and buffer sequencing (due to the uni-directional constraint required by double-buffering feature) the other one can be used by the application software to show which buffer it is currently using. This new buffer flag is called SW_BUF. In the following table the correspondence between USB_EPnR register bits and DTOG/SW_BUF definition is explained, for the cases of 'transmission' and 'reception' double-buffered bulk endpoints.

Table 58. Double-Buffering Buffer Flag Definition

Buffer flag	'Transmission' endpoint	'Reception' endpoint
DTOG	DTOG_TX (USB_EPnR bit 6)	DTOG_RX (USB_EPnR bit 14)
SW_BUF	USB_EPnR bit 14	USB_EPnR bit 6

The memory buffer which is currently being used by the USB Peripheral is defined by DTOG buffer flag, while the buffer currently in use by application software is identified by SW_BUF buffer flag. The relationship between the buffer flag value and the used packet buffer is the same in both cases, and it is listed in the following table.

Table 59. Double-Buffering Memory Buffers Usage

DTOG or SW_BUF bit value	Packet buffer used by USB Peripheral (DTOG) or application software (SW_BUF)
0	ADDRn_TX / COUNTn_TX buffer description table locations.
1	ADDRn_RX / COUNTn_RX buffer description table locations.

Double-buffering feature for a bulk endpoint is activated by:

- writing EP_TYPE bit field at '00' in its USB_EPnR register, to define the endpoint as a bulk, and
- setting EP_KIND bit at '1' (DBL_BUF), in the same register.

The application software is responsible for DTOG and SW_BUF bits initialization according to the first buffer to be used; this has to be done considering the special toggle-only property that these two bits have. The end of the first transaction occurring after having set DBL_BUF, triggers the special flow control of double-buffered bulk endpoints, which is used for all other transactions addressed to this endpoint until DBL_BUF remain set. At the end of each transaction the CTR_RX or CTR_TX bit of the addressed endpoint USB_EPnR register is set, depending on the enabled direction. At the same time, the affected DTOG bit in the USB_EPnR register is hardware toggled making the USB Peripheral buffer swapping completely software independent. Unlike common transactions, and the first one after DBL_BUF setting, STAT bit pair is not affected by the transaction termination and its value remains '11' (Valid). However, as the token packet of a new transaction is received, the actual endpoint status will be masked as '10' (NAK) when a buffer conflict between the USB Peripheral and the application software is detected (this condition is identified by DTOG and SW_BUF having the same value). The application software responds to the CTR event notification by clearing the interrupt flag and starting any required handling of the completed transaction. When the application packet buffer usage is over, the software toggles the SW_BUF bit, writing '1' to it, to notify the USB Peripheral about the availability of that buffer. In this way, the number of NAKed transactions is limited only by the application elaboration time of a transaction data: if the elaboration time is shorter than the time required to complete a transaction on the USB bus, no re-transmissions due to flow control will take place and the actual transfer rate will be limited only by the host PC.

The application software can always override the special flow control implemented for double-buffered bulk endpoints, writing an explicit status different from '11' (Valid) into the STAT bit pair of the related USB_EPnR register. In this case, the USB Peripheral will always use the programmed endpoint status, regardless of the buffer usage condition.

19.5.4 Isochronous Transfers

The USB standard supports full speed peripherals requiring a fixed and accurate data production/consume frequency, defining this kind of traffic as 'Isochronous'. Typical examples of this data are: audio samples, compressed video streams, and in general any sort of sampled data having strict requirements for the accuracy of delivered frequency. When an endpoint is defined to be 'isochronous' during the enumeration phase, the host allocates in the frame the required bandwidth and delivers exactly one IN or OUT packet each frame, depending on endpoint direction. To limit the bandwidth requirements, no re-transmission of failed transactions is possible for Isochronous traffic; this leads to the fact that an isochronous transaction does not have a handshake phase and no ACK packet is expected or sent after the data packet. For the same reason, Isochronous transfers do not support data toggle sequencing and always use DATA0 PID to start any data packet.

The Isochronous behaviour for an endpoint is selected by setting the EP_TYPE bits at '10' in its USB_EPnR register; since there is no handshake phase the only legal values for the STAT_RX/STAT_TX bit pairs are '00' (Disabled) and '11' (Valid), any other value will produce results not compliant to USB standard. Isochronous endpoints implement double-buffering to ease application software development, using both 'transmission' and 'reception' packet memory areas to manage buffer swapping on each successful transaction in order to have always a complete buffer to be used by the application, while the USB Peripheral fills the other.

The memory buffer which is currently used by the USB Peripheral is defined by the DTOG bit related to the endpoint direction (DTOG_RX for 'reception' isochronous endpoints, DTOG_TX for 'transmission' isochronous endpoints, both in the related USB_EPnR register) according to Table 60.

Table 60. Isochronous Memory Buffers Usage

DTOG bit value	DMA buffer used by USB Peripheral	DMA buffer used by application software
0	ADDRn_TX / COUNTn_TX buffer description table locations.	ADDRn_RX / COUNTn_RX buffer description table locations.
1	ADDRn_RX / COUNTn_RX buffer description table locations.	ADDRn_TX / COUNTn_TX buffer description table locations.

As it happens with double-buffered bulk endpoints, the USB_EPnR registers used to implement Isochronous endpoints are forced to be used as uni-directional ones. In case it is required to have Isochronous endpoints enabled both for reception and transmission, two USB_EPnR registers must be used.

The application software is responsible for the DTOG bit initialization according to the first buffer to be used; this has to be done considering the special toggle-only property that these

two bits have. At the end of each transaction, the CTR_RX or CTR_TX bit of the addressed endpoint USB_EPnR register is set, depending on the enabled direction. At the same time, the affected DTOG bit in the USB_EPnR register is hardware toggled making buffer swapping completely software independent. STAT bit pair is not affected by transaction completion; since no flow control is possible for Isochronous transfers due to the lack of handshake phase, the endpoint remains always '11' (Valid). CRC errors or buffer-overrun conditions occurring during Isochronous OUT transfers are anyway considered as correct transactions and they always trigger an CTR_RX event. However, CRC errors will anyway set the ERR bit in the USB_ISTR register to notify the software of the possible data corruption.

19.5.5 Suspend/Resume Events

The USB standard defines a special peripheral state, called SUSPEND, in which the average current drawn from the USB bus must not be greater than 500 μA. This requirement is of fundamental importance for bus-powered devices, while self-powered devices are not required to comply to this strict power consumption constraint. In suspend mode, the host PC sends the notification to not send any traffic on the USB bus for more than 3mS: since a SOF packet must be sent every mS during normal operations, the USB Peripheral detects the lack of 3 consecutive SOF packets as a suspend request from the host PC and set the SUSP bit to '1' in USB_ISTR register, causing an interrupt if enabled. Once the device is suspended, its normal operation can be restored by a so called RESUME sequence, which can be started from the host PC or directly from the peripheral itself, but it is always terminated by the host PC. The suspended USB Peripheral must be anyway able to detect a RESET sequence, reacting to this event as a normal USB reset event.

The actual procedure used to suspend the USB peripheral is device dependent since according to the device composition, different actions may be required to reduce the total consumption.

A brief description of a typical suspend procedure is provided below, focused on the USB-related aspects of the application software routine responding to the SUSP notification of the USB Peripheral:

1. Set the FSUSP bit in the USB_CNTR register to 1. This action activates the suspend mode within the USB Peripheral. As soon as the suspend mode is activated, the check on SOF reception is disabled to avoid any further SUSP interrupts being issued while the USB is suspended.
2. Remove or reduce any static power consumption in blocks different from the USB Peripheral.
3. Set LP_MODE bit in USB_CNTR register to 1 to remove static power consumption in the analog USB transceivers but keeping them able to detect resume activity.
4. Optionally turn off external oscillator and device PLL to stop any activity inside the device.

When an USB event occurs while the device is in SUSPEND mode, the RESUME procedure must be invoked to restore nominal clocks and regain normal USB behaviour. Particular care must be taken to insure that this process does not take more than 10mS when the wakening

event is an USB reset sequence (See “Universal Serial Bus Specification” for more details). The start of a resume or reset sequence, while the USB Peripheral is suspended, clears the LP_MODE bit in USB_CNTR register asynchronously. Even if this event can trigger an WKUP interrupt if enabled, the use of an interrupt response routine must be carefully evaluated because of the long latency due to system clock restart; to have the shorter latency before re-activating the nominal clock it is suggested to put the resume procedure just after the end of the suspend one, so its code is immediately executed as soon as the system clock restarts. To prevent ESD discharges or any other kind of noise from waking-up the system (the exit from suspend mode is an asynchronous event), a suitable analog filter on data line status is activated during suspend; the filter width is about 70nS.

The following is a list of actions a resume procedure should address:

1. Optionally turn on external oscillator and device PLL.
2. Clear FSUSP bit of USB_CNTR register.
3. If the resume triggering event has to be identified, bits RXDP and RXDM in the USB_FNR register can be used according to Table 61, which also lists the intended software action in all the cases. If required, the end of resume or reset sequence can be detected monitoring the status of the above mentioned bits by checking when they reach the “10” configuration, which represent the Idle bus state; moreover at the end of a reset sequence the RESET bit in USB_ISTR register is set to 1, issuing an interrupt if enabled, which should be handled as usual.

Table 61. Resume Event Detection

[RXDP,RXDM] Status	Wake-up event	Required resume software action
“00”	Root reset	None
“10”	None (noise on bus)	Go back in Suspend mode
“01”	Root resume	None
“11”	Not Allowed (noise on bus)	Go back in Suspend mode

A device may require to exit from suspend mode as an answer to particular events not directly related to the USB protocol (e.g. a mouse movement wakes up the whole system). In this case, the resume sequence can be started by setting the RESUME bit in the USB_CNTR register to ‘1’ and resetting it to 0 after an interval between 1mS and 15mS (this interval can be timed using ESOF interrupts, occurring with a 1mS period when the system clock is running at nominal frequency). Once the RESUME bit is clear, the resume sequence will be completed by the host PC and its end can be monitored again using the RXDP and RXDM bits in the USB_FNR register.

Note *The RESUME bit must be anyway used only after the USB Peripheral has been put in suspend mode, setting the FSUSP bit in USB_CNTR register to 1.*

19.6 Register Description

The USB Peripheral registers can be divided into the following groups:

- Common Registers: Interrupt and Control registers
- Endpoint Registers: Endpoint configuration and status
- Buffer Descriptor Table: Location of packet memory used to locate data buffers

All register addresses are expressed as offsets with respect to the USB Peripheral registers base address 0xC000 8000, except the buffer descriptor table locations, which starts at the address specified by the USB_BTABLE register. Due to the common limitation of APB bridges on word addressability, all register addresses are aligned to 32-bit word boundaries although they are 16-bit wide. The same address alignment is used to access packet buffer memory locations, which are located starting from 0xC000 8800. In this section, the following abbreviations are used:

read/write (rw)	The software can read and write to these bits.
read-only (r)	The software can only read these bits.
write-only (w)	The software can only write to these bits.
Read-clear (rc)	The software can only read or clear this bit.
Toggle (t)	The software can only toggle this bit by writing ‘1’. Writing ‘0’ has no effect.

19.6.1 Common Registers

These registers affect the general behaviour of the USB Peripheral defining operating mode, interrupt handling, device address and giving access to the current frame number updated by the host PC.

USB Control Register (USB_CNTR)

Address Offset: 40h

Reset Value: 0000 0000 0000 0011 (0003h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTRM	DOVRM	ERRM	WKUPM	SUSPM	RESETM	SOFM	ESOFM	Reserved			RESUME	FSUSP	LP MODE	PDWN	FRES
rw	rw	rw	rw	rw	rw	rw	rw				rw	rw	rw	rw	rw

Bit 15 CTRM: *Correct Transfer Interrupt Mask*
0: Correct Transfer (CTR) Interrupt disabled.
1: CTR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 14 DOVRM: *DMA over / underrun Interrupt Mask*
0: DOVR Interrupt disabled.
1: DOVR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 13 ERRM: *Error Interrupt Mask*
0: ERR Interrupt disabled.
1: ERR Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 12 WKUPM: *Wake-up Interrupt Mask*
0: WKUP Interrupt disabled.
1: WKUP Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 11 SUSPM: *Suspend mode Interrupt Mask*
0: Suspend Mode Request (SUSP) Interrupt disabled.
1: SUSP Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 10 RESETM: *USB Reset Interrupt Mask*
0: RESET Interrupt disabled.
1: RESET Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 9 SOFM: *Start Of Frame Interrupt Mask*
0: SOF Interrupt disabled.
1: SOF Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bit 8 ESOFM: *Expected Start Of Frame Interrupt Mask*
0: Expected Start of Frame (ESOF) Interrupt disabled.
1: ESOF Interrupt enabled, an interrupt request is generated when the corresponding bit in the USB_ISTR register is set.

Bits 7:5 Reserved
These are reserved bits. These bits are always read as '0' and must always be written with '0'.

Bit 4 RESUME: *Resume request*
The microcontroller can set this bit to send a Resume signal to the host. It must be activated, according to USB specifications, for no less than 1mS and no more than 15mS after which the Host PC is ready to drive the resume sequence up to its end.

Bit 3 FSUSP: *Force suspend*
Software must set this bit when the SUSP interrupt is received, which is issued when no traffic is received by the USB Peripheral for 3 mS.
0: No effect.
1: Enter suspend mode. Clocks and static power dissipation in the analog transceiver are left unaffected. If suspend power consumption is a requirement (bus-powered device), the application software should set the LP_MODE bit after FSUSP as explained below.

Bit 2 LP_MODE: *Low-power mode*
This mode is used when the suspend-mode power constraints require that all static power dissipation is avoided, except the one required to supply the external pull-up resistor. This condition should be entered when the application is ready to stop all system clocks, or reduce their frequency in order to meet the power consumption requirements of the USB suspend condition. The USB activity during the suspend mode (WKUP event) asynchronously resets this bit (it can also be reset by software).
0: No Low Power Mode.
1: Enter Low Power mode.

Bit 1 PDWN: *Power down*

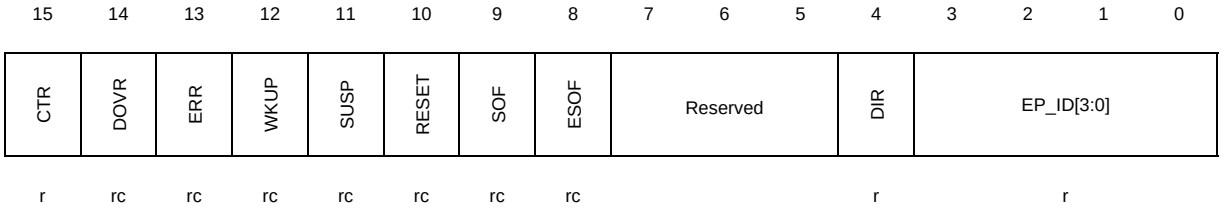
This bit is used to completely switch off all USB-related analog parts if it is required to completely disable the USB Peripheral for any reason. When this bit is set, the USB Peripheral is disconnected from the transceivers and it cannot be used.
0: Exit Power Down.
1: Enter Power down mode.

Bit 0 FRES: *Force USB Reset*
0: Clear USB reset.
1: Force a reset of the USB Peripheral, exactly like a RESET signalling on the USB. The USB Peripheral is held in RESET state until software clears this bit. A “USB-RESET” interrupt is generated, if enabled.

USB Interrupt Status Register (USB_ISTR)

Address Offset: 44h

Reset Value: 0000 0000 0000 0000 (0000h)



This register contains the status of all the interrupt sources allowing application software to determine, which events caused an interrupt request.

The upper part of this register contains single bits, each of them representing a specific event. These bits are set by the hardware when the related event occurs; if the corresponding bit in the USB_CNTR register is set, a generic interrupt request is generated. The interrupt routine, examining each bit, will perform all necessary actions, and finally it will clear the serviced bits. If any of them is not cleared, the interrupt is considered to be still pending, and the interrupt line will be kept high again. If several bits are set simultaneously, only a single interrupt will be generated.

Endpoint transaction completion can be handled in a different way to reduce interrupt response latency. The CTR bit is set by the hardware as soon as an endpoint successfully completes a transaction, generating a generic interrupt request if the corresponding bit in USB_CNTR is set. An endpoint dedicated interrupt condition is activated independently from the CTRM bit in the USB_CNTR register. Both interrupt conditions remain active until software clears the pending bit in the corresponding USB_EPnR register (the CTR bit is actually a read only bit). The USB Peripheral has two interrupt request lines:

- Higher priority USB IRQ: The pending requests for endpoints, which have transactions with a higher priority (isochronous and double-buffered bulk) and they cannot be masked.
- Lower priority USB IRQ: All other interrupt conditions, which can either be non-maskable pending requests related to the lower priority transactions and all other maskable events flagged by the USB_ISTR high bytes.

For endpoint-related interrupts, the software can use the Direction of Transaction (DIR) and EP_ID read-only bits to identify, which endpoint made the last interrupt request and called the corresponding interrupt service routine.

The user can choose the relative priority of simultaneously pending USB_ISTR events by specifying the order in which software checks USB_ISTR bits in an interrupt service routine.

Only the bits related to events, which are serviced, are cleared. At the end of the service routine, another interrupt will be requested, to service the remaining conditions.

To avoid spurious clearing of some bits, it is recommended to clear them with a load instruction where all bits which must not be altered are written with 1, and all bits to be cleared are written with '0' (these bits can only be cleared by software). Read-modify-write cycles should be avoided because between the read and the write operations another bit could be set by the hardware and the next write will clear it before the microprocessor has the time to serve the event.

The following describes each bit in detail:

Bit 15 CTR: *Correct Transfer*
This bit is set by the hardware to indicate that an endpoint has successfully completed a transaction; using DIR and EP_ID bits software can determine which endpoint requested the interrupt. This bit is read-only.

Bit 14 DOVR: *DMA over / underrun*
This bit is set if the microcontroller has not been able to respond in time to an USB memory request. The USB Peripheral handles this event in the following way: During reception an ACK handshake packet is not sent, during transmission a bit-stuff error is forced on the transmitted stream; in both cases the host will retry the transaction. The DOVR interrupt should never occur during normal operations. Since the failed transaction is retried by the host, the application software has the chance to speed-up device operations during this interrupt handling, to be ready for the next transaction retry; however this does not happen during Isochronous transfers (no isochronous transaction is anyway retried) leading to a loss of data in this case. This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 13 ERR: *Error*

This flag is set whenever one of the errors listed below has occurred:
NANS: No ANSwer. The timeout for a host response has expired.
CRC: Cyclic Redundancy Check error. One of the received CRCs, either in the token or in the data, was wrong.
BST: Bit Stuffing error. A bit stuffing error was detected anywhere in the PID, data, and/or CRC.
FVIO: Framing format Violation. A non-standard frame was received (EOP not in the right place, wrong token sequence, etc.).
The USB software can usually ignore errors, since the USB Peripheral and the PC host manage retransmission in case of errors in a fully transparent way. This interrupt can be useful during the software development phase, or to monitor the quality of transmission over the USB bus, to flag possible problems to the user (e.g. loose connector, too noisy environment, broken conductor in the USB cable and so on). This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 12 WKUP: *Wake up*

This bit is set to 1 by the hardware when, during suspend mode, activity is detected that wakes up the USB Peripheral. This event asynchronously clears the LP_MODE bit in the CTLR register and activates the USB_WAKEUP line, which can be used to notify the rest of the device (e.g. wake-up unit) about the start of the resume process. This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 11 SUSP *Suspend mode request*

This bit is set by the hardware when no traffic has been received for 3mS, indicating a suspend mode request from the USB bus. The suspend condition check is enabled immediately after any USB reset and it is disabled by the hardware when the suspend mode is active (FSUSP=1) until the end of resume sequence. This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 10 RESET: *USB RESET request*

Set when the USB Peripheral detects an active USB RESET signal at its inputs. The USB Peripheral, in response to a RESET, just resets its internal protocol state machine, generating an interrupt if RESETM enable bit in the USB_CNTR register is set. Reception and transmission are disabled until the RESET bit is cleared. All configuration registers do not reset: the microcontroller must explicitly clear these registers (this is to ensure that the RESET interrupt can be safely delivered, and any transaction immediately followed by a RESET can be completed). The function address and endpoint registers are reset by an USB reset event.
This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 9 SOF: *Start Of Frame*
This bit signals the beginning of a new USB frame and it is set when a SOF packet arrives through the USB bus. The interrupt service routine may monitor the SOF events to have a 1mS synchronization event to the USB host and to safely read the USB_FNR register which is updated at the SOF packet reception (this could be useful for isochronous applications). This bit is read/write but only '0' can be written and writing '1' has no effect.

Bit 8 ESOF: *Expected Start Of Frame*
This bit is set by the hardware when an SOF packet is expected but not received. The host sends an SOF packet each mS, but if the hub does not receive it properly, the Suspend Timer issues this interrupt. If three consecutive ESOF interrupts are generated (i.e. three SOF packets are lost) without any traffic occurring in between, a SUSP interrupt is generated. This bit is set even when the missing SOF packets occur while the Suspend Timer is not yet locked. This bit is read/write but only '0' can be written and writing '1' has no effect.

Bits 7:5 Reserved.
These are reserved bits. These bits are always read as '0' and must always be written with '0'.

Bit 4 DIR: *Direction of transaction*.
This bit is written by the hardware according to the direction of the successful transaction, which generated the interrupt request.
If DIR bit=0, CTR_TX bit is set in the USB_EPnR register related to the interrupting endpoint. The interrupting transaction is of IN type (data transmitted by the USB Peripheral to the host PC).
If DIR bit=1, CTR_RX bit or both CTR_TX/CTR_RX are set in the USB_EPnR register related to the interrupting endpoint. The interrupting transaction is of OUT type (data received by the USB Peripheral from the host PC) or two pending transactions are waiting to be processed.
This information can be used by the application software to access the USB_EPnR bits related to the triggering transaction since it represents the direction having the interrupt pending. This bit is read-only.

Bits 3:0 EP_ID[3:0]: *Endpoint Identifier*.
These bits are written by the hardware according to the endpoint number, which generated the interrupt request. If several endpoint transactions are pending, the hardware writes the endpoint identifier related to the endpoint having the highest priority defined in the following way: Two endpoint sets are defined, in order of priority: Isochronous and double-buffered bulk endpoints are considered first and then the other endpoints are examined. If more than one endpoint from the same set is requesting an interrupt, the EP_ID bits in USB_ISTR register are assigned according to the lowest requesting endpoint register, EP0R having the highest priority followed by EP1R and so on. The application software can assign a register to each endpoint according to this priority scheme, so as to order the concurring endpoint requests in a suitable way. These bits are read only.

USB Frame Number Register (USB_FNR)

Address Offset: 48h

Reset Value: 0000 0xxx xxxx xxxx (0xxxh)

Bit 15 RXDP *Receive Data + Line Status*
This bit can be used to observe the status of received data plus upstream port data line. It can be used during end-of-suspend routines to help determining the wake-up event.

Bit 14 RXDM. *Receive Data - Line Status*
This bit can be used to observe the status of received data minus upstream port data line. It can be used during end-of-suspend routines to help determining the wake-up event.

Bit 13 LCK: *Locked*
This bit is set by the hardware when at least two consecutive SOF packets have been received after the end of an USB reset condition or after the end of an USB resume sequence. Once locked, the frame timer remains in this state until an USB reset or USB suspend event occurs.

Bits12:11 LSOF[1:0]: *Lost SOF*
These bits are written by the hardware when an ESOF interrupt is generated, counting the number of consecutive SOF packets lost. At the reception of an SOF packet, these bits are cleared.

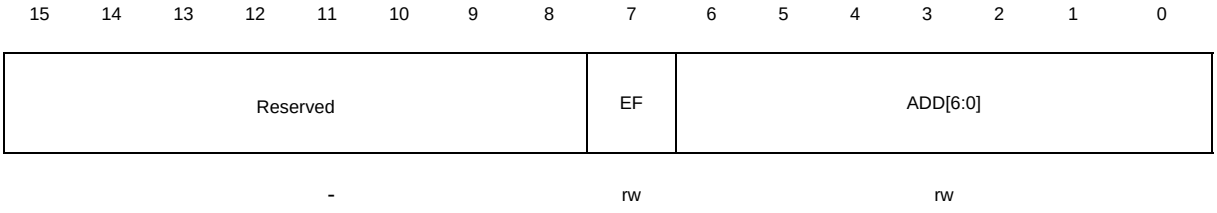
Bits 10:0 FN[10:0]**:** *Frame Number*

This bit field contains the 11-bits frame number contained in the last received SOF packet. The frame number is incremented for every frame sent by the host and it is useful for Isochronous transfers. This bit field is updated on the generation of an SOF interrupt.

USB Device Address (USB_DADDR)

Address Offset: 4Ch

Reset Value: 0000 0000 0000 0000 (0000h)



This register is also reset when a USB reset is received from the USB bus or forced through bit FRES in the USB_CNTR register.

Bits 15:8 Reserved

These are reserved bits. These bits are always read as ‘0’ and must always be written with ‘0’.

Bit 7 EF: *Enable Function*

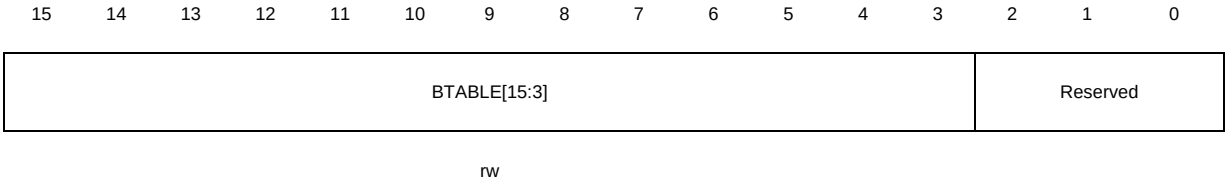
This bit is set by the software to enable the USB device. The address of this device is contained in the following ADD[6:0] bits. If this bit is at ‘0’ no transactions are handled, irrespective of the settings of USB_EPnR registers.

Bits 6:0 ADD[6:0]: *Device Address*
These bits contain the USB function address assigned by the host PC during the enumeration process. Both this field and the Endpoint Address (EA) field in the associated USB_EPnR register must match with the information contained in a USB token in order to handle a transaction to the required endpoint.

Buffer Table Address (USB_BTABLE)

Address Offset: 50h

Reset Value: 0000 0000 0000 0000 (0000h)



Bits 15:3 BTABLE[15:3]: *Buffer Table*.
These bits contain the start address of the buffer allocation table inside the dedicated packet memory. This table describes each endpoint buffer location and size and it must be aligned to an 8 byte boundary (the 3 least significant bits are always ‘0’). At the beginning of every transaction addressed to this device, the USP peripheral reads the element of this table related to the addressed endpoint, to get its buffer start location and the buffer size (Refer to Section “Structure and Usage of Packet Buffers”).

Bits 2:0 Reserved.

These are reserved bits. These bits are always read as ‘0’ and must always be written with ‘0’.

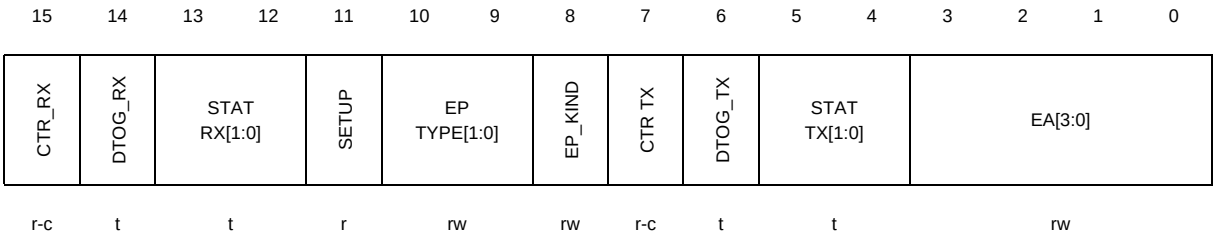
19.6.2 Endpoint-Specific Registers

The number of these registers varies according to the number of endpoints that the USB Peripheral is designed to handle. The USB Peripheral supports up to 8 endpoints. Each USB device must support a control endpoint whose address (EA bits) must be set to 0. The USB Peripheral behaves in an undefined way if multiple endpoints are enabled having the same endpoint number value. For each endpoint, an USB_EPnR register is available to store the endpoint specific information.

USB Endpoint n Register (USB_EPnR)

Address Offset: 00h to 2Ch

Reset value: 0000 0000 0000 0000b (0000h)



They are also reset when an USB reset is received from the USB bus or forced through bit FRES in the CTLR register, except the CTR_RX and CTR_TX bits, which are kept unchanged to avoid missing a correct packet notification immediately followed by an USB reset event. Each endpoint has its USB_EPnR register where *n* is the endpoint identifier.

Read-modify-write cycles on these registers should be avoided because between the read and the write operations some bits could be set by the hardware and the next write would modify them before the microprocessor has the time to detect the change. For this purpose, all bits affected by this problem have an ‘invariant’ value that must be used whenever their modification is not required. It is recommended to modify these registers with a load instruction where all the bits, which can be modified only by the hardware, are written with their ‘invariant’ value.

Bit 15 CTR_RX: *Correct Transfer for reception*
This bit is set by the hardware when an OUT/SETUP transaction is successfully completed on this endpoint; the software can only clear this bit. If the CTRM bit in USB_CNTR register is set accordingly, a generic interrupt condition is generated together with the endpoint related interrupt condition, which is always activated. The type of occurred transaction, OUT or SETUP, can be determined from the SETUP bit described below.

A transaction ended with a NAK or STALL handshake does not set this bit, since no data is actually transferred, as in the case of protocol errors or data toggle mismatches.

This bit is read/write but only '0' can be written, writing 1 has no effect.

Bit 14 DTOG_RX: *Data Toggle, for reception transfers*
If the endpoint is not Isochronous, this bit contains the expected value of the data toggle bit (0=DATA0, 1=DATA1) for the next data packet to be received. Hardware toggles this bit, when the ACK handshake is sent to the USB host, following a data packet reception having a matching data PID value; if the endpoint is defined as a control one, hardware clears this bit at the reception of a SETUP PID addressed to this endpoint.
If the endpoint is using the double-buffering feature this bit is used to support packet buffer swapping too (Refer to Section "Double-Buffered Endpoints").
If the endpoint is Isochronous, this bit is used only to support packet buffer swapping since no data toggling is used for this sort of endpoints and only DATA0 packet are transmitted (Refer to Section "Isochronous Transfers"). Hardware toggles this bit just after the end of data packet reception, since no handshake is used for isochronous transfers.
This bit can also be toggled by the software to initialize its value (mandatory when the endpoint is not a control one) or to force specific data toggle/packet buffer usage. When the application software writes '0', the value of DTOG_RX remains unchanged, while writing '1' makes the bit value toggle. This bit is read/write but it can be only toggled by writing 1.

Bits 13:12 STAT_RX [1:0] *Status bits, for reception transfers*

These bits contain information about the endpoint status, which are listed in Table 62, “Reception Status Encoding,” on page 296.These bits can be toggled by software to initialize their value. When the application software writes ‘0’, the value remains unchanged, while writing ‘1’ makes the bit value toggle. Hardware sets the STAT_RX bits to NAK when a correct transfer has occurred (CTR_RX=1) corresponding to a OUT or SETUP (control only) transaction addressed to this endpoint, so the software has the time to elaborate the received data before it acknowledge a new transaction

Double-buffered bulk endpoints implement a special transaction flow control, which control the status based upon buffer availability condition (Refer to Section “Double-Buffered Endpoints”).

If the endpoint is defined as Isochronous, its status can be only “VALID” or “DISABLED”, so that the hardware cannot change the status of the endpoint after a successful transaction. If the software sets the STAT_RX bits to ‘STALL’ or ‘NAK’ for an Isochronous endpoint, the USB Peripheral behaviour is not defined. These bits are read/write but they can be only toggled by writing ‘1’.

Bit 11 SETUP: *Setup transaction completed*

This bit is read-only and it is set by the hardware when the last completed transaction is a SETUP. This bit changes its value only for control endpoints. It must be examined, in the case of a successful receive transaction (CTR_RX event), to determine the type of transaction occurred. To protect the interrupt service routine from the changes in SETUP bits due to next incoming tokens, this bit is kept frozen while CTR_RX bit is at 1; its state changes when CTR_RX is at 0. This bit is read-only.

Bits 10:9 EP_TYPE[1:0]: *Endpoint type*

These bits configure the behaviour of this endpoint as described in Table 63, “Endpoint Type Encoding,” on page 296. Endpoint 0 must always be a control endpoint and each USB function must have at least one control endpoint which has address 0, but there may be other control endpoints if required. Only control endpoints handle SETUP transactions, which are ignored by endpoints of other kinds. SETUP transactions cannot be answered with NAK or STALL. If a control endpoint is defined as NAK, the USB Peripheral will not answer, simulating a receive error, in the receive direction when a SETUP transaction is received. If the control endpoint is defined as STALL in the receive direction, then the SETUP packet will be accepted anyway, transferring data and issuing the CTR interrupt. The reception of OUT transactions is handled in the normal way, even if the endpoint is a control one.

Bulk and interrupt endpoints have very similar behaviour and they differ only in the special feature available using the EP_KIND configuration bit.

The usage of Isochronous endpoints is explained in “Isochronous Transfers”

Bit 8 EP_KIND: *Endpoint Kind*

The meaning of this bit depends on the endpoint type configured by the EP_TYPE bits. Table 64 summarizes the different meanings.

DBL_BUF: This bit is set by the software to enable the double-buffering feature for this bulk endpoint. The usage of double-buffered bulk endpoints is explained in "Double-Buffered Endpoints".

STATUS_OUT: This bit is set by the software to indicate that a status out transaction is expected: in this case all OUT transactions containing more than zero data bytes are answered 'STALL' instead of 'ACK'. This bit may be used to improve the robustness of the application to protocol errors during control transfers and its usage is intended for control endpoints only. When STATUS_OUT is reset, OUT transactions can have any number of bytes, as required.

Bit 7 CTR_TX: *Correct Transfer for transmission*

This bit is set by the hardware when an IN transaction is successfully completed on this endpoint; the software can only clear this bit. If the CTRM bit in the USB_CNTR register is set accordingly, a generic interrupt condition is generated together with the endpoint related interrupt condition, which is always activated.

Note A transaction ended with a NAK or STALL handshake does not set this bit, since no data is actually transferred, as in the case of protocol errors or data toggle mismatches.

This bit is read/write but only '0' can be written.

Bit 6 DTOG_TX: *Data Toggle, for transmission transfers*
If the endpoint is non-isochronous, this bit contains the required value of the data toggle bit (0=DATA0, 1=DATA1) for the next data packet to be transmitted. Hardware toggles this bit when the ACK handshake is received from the USB host, following a data packet transmission. If the endpoint is defined as a control one, hardware sets this bit to 1 at the reception of a SETUP PID addressed to this endpoint.
If the endpoint is using the double buffer feature, this bit is used to support packet buffer swapping too (Refer to Section "Double-Buffered Endpoints")
If the endpoint is Isochronous, this bit is used to support packet buffer swapping since no data toggling is used for this sort of endpoints and only DATA0 packet are transmitted (Refer to section "Isochronous Transfers"). Hardware toggles this bit just after the end of data packet transmission, since no handshake is used for Isochronous transfers.
This bit can also be toggled by the software to initialize its value (mandatory when the endpoint is not a control one) or to force a specific data toggle/ packet buffer usage. When the application software writes '0', the value of DTOG_TX remains unchanged, while writing '1' makes the bit value toggle. This bit is read/write but it can only be toggled by writing 1.

Bit 5:4 STAT_TX [1:0] *Status bits, for transmission transfers*
These bits contain the information about the endpoint status, listed in Table 65. These bits can be toggled by the software to initialize their value. When the application software writes '0', the value remains unchanged, while writing '1' makes the bit value toggle. Hardware sets the STAT_TX bits to NAK, when a correct transfer has occurred (CTR_TX=1) corresponding to a IN or SETUP (control only) transaction addressed to this endpoint. It then waits for the software to prepare the next set of data to be transmitted.
Double-buffered bulk endpoints implement a special transaction flow control, which controls the status based on buffer availability condition (Refer to Section "Double-Buffered Endpoints").
If the endpoint is defined as Isochronous, its status can only be "VALID" or "DISABLED". Therefore, the hardware cannot change the status of the endpoint after a successful transaction. If the software sets the STAT_TX bits to 'STALL' or 'NAK' for an Isochronous endpoint, the USB Peripheral behaviour is not defined. These bits are read/write but they can be only toggled by writing '1'.

Bit 3:0 EA[3:0] *Endpoint Address.*
Software must write in this field the 4-bit address used to identify the transactions directed to this endpoint. A value must be written before enabling the corresponding endpoint.

Table 62. Reception Status Encoding

STAT_RX[1:0]	Meaning
00	**DISABLED:** all reception requests addressed to this endpoint are ignored.
01	**STALL**: the endpoint is stalled and all reception requests result in a STALL handshake.
10	**NAK**: the endpoint is naked and all reception requests result in a NAK handshake.
11	**VALID**: this endpoint is enabled for reception.

Table 63. Endpoint Type Encoding

EP_TYPE[1:0]	Meaning
00	BULK
01	CONTROL
10	ISO
11	INTERRUPT

Table 64. Endpoint Kind Meaning

EP_TYPE[1:0]		EP_KIND Meaning
00	BULK	DBL_BUF
01	CONTROL	STATUS_OUT
10	ISO	Not used
11	INTERRUPT	Not used

Table 65. Transmission Status Encoding

STAT_TX[1:0]	Meaning
00	**DISABLED:** all transmission requests addressed to this endpoint are ignored.
01	**STALL**: the endpoint is stalled and all transmission requests result in a STALL handshake.
10	**NAK**: the endpoint is naked and all transmission requests result in a NAK handshake.
11	**VALID**: this endpoint is enabled for transmission.

19.6.3 Buffer Descriptor Table

Although this table is located inside packet buffer memory, its entries can be considered as additional registers used to configure the location and size of packet buffers used to exchange data between USB and the STR720. Due to the common APB bridge limitation on word addressability, all packet memory locations are accessed by the APB using 32-bit aligned addresses, instead of the actual memory location addresses utilized by the USB Peripheral for the USB_BTABLE register and buffer description table locations. In the following pages two location addresses are reported: the one to be used by application software while accessing the packet memory, and the local one relative to USB Peripheral access. To obtain the correct STR720 memory address value to be used in the application software while accessing the packet memory, the actual memory location address must be multiplied by two. The first packet memory location is located at 0xC000 8800.

The buffer description table entry associated with the USB_EPnR registers is described below. A thorough explanation of packet buffers and buffer descriptor table usage can be found in the Section "Structure and Usage of Packet Buffers".

Transmission Buffer Address n (USB_ADDRn_TX)

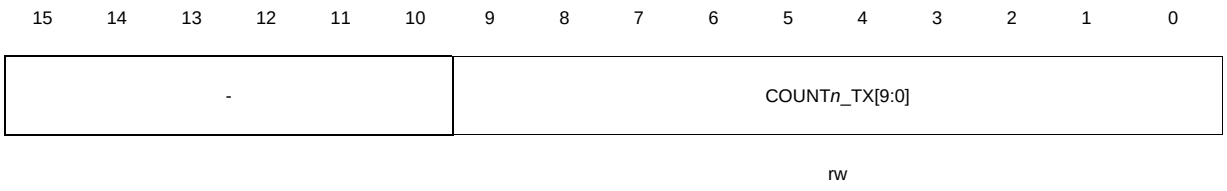
Address Offset: [USB_BTABLE] + *n**16 USB local Address: [USB_BTABLE] + *n**8

Bits 15:1 ADDRn_TX[15:1]. *Transmission Buffer Address*
These bits point to the starting address of the packet buffer containing data to be transmitted by the endpoint associated with the USB_EPnR register at the next IN token addressed to it.

Bit 0 Must always be written as '0' since packet memory is word-wide and all packet buffers must be word-aligned.

Transmission Byte Count n (USB_COUNTn_TX)

Address Offset: [USB_BTABLE] + *n**16 + 4 USB local Address: [USB_BTABLE] + *n**8 + 2

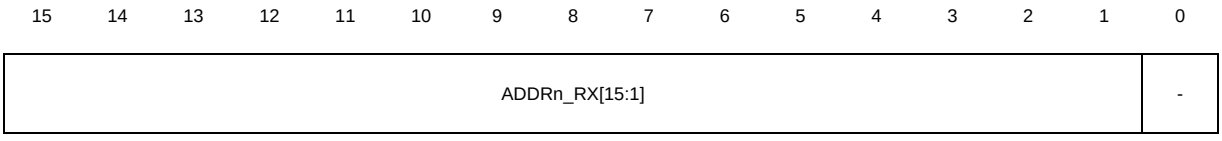


Bits 15:10 These bits are not used since packet size is limited by USB specifications to 1023 bytes. Their value is not considered by the USB Peripheral.

Bits 9:0 COUNTn_TX[9:0]. *Transmission Byte Count*
These bits contain the number of bytes to be transmitted by the endpoint associated with the USB_EPnR register at the next IN token addressed to it.

Reception Buffer Address n (USB_ADDRn_RX)

Address Offset: [USB_BTABLE] + *n**16 + 8 USB local Address: [USB_BTABLE] + *n**8 + 4



Bits 15:1 ADDRn_RX[15:1]. *Reception Buffer Address*
These bits point to the starting address of the packet buffer, which will contain the data received by the endpoint associated with the USB_EPnR register at the next OUT/SETUP token addressed to it.

Bit 0 This bit must always be written as '0' since packet memory is word-wide and all packet buffers must be word-aligned.

Reception Byte Count n (USB_COUNTn_RX)

Address Offset: [USB_BTABLE] + *n**16 + 12 USB local Address: [USB_BTABLE] + *n**8 + 6

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BLSIZE	NUM_BLOCK[4:0]					COUNT*n*_RX[9:0]									
rw	rw					r									

This table location is used to store two different values, both required during packet reception. The most significant bits contains the definition of allocated buffer size, to allow buffer overflow detection, while the least significant part of this location is written back by the USB Peripheral at the end of reception to give the actual number of received bytes. Due to the restrictions on the number of available bits, buffer size is represented using the number of allocated memory blocks, where block size can be selected to choose the trade-off between fine-granularity/ small-buffer and coarse-granularity/large-buffer. The size of allocated buffer is a part of the endpoint descriptor and it is normally defined during the enumeration process according to its maxPacketSize parameter value (See “Universal Serial Bus Specification”).

Bit 15 BL_SIZE: *BLock SIZE.*

This bit selects the size of memory block used to define the allocated buffer area.

- If BL_SIZE=0, the memory block is 2 `byte` large, which is the minimum block allowed in a word-wide memory. With this block size the allocated buffer size ranges from 2 to 62 `bytes`.
- If BL_SIZE=1, the memory block is 32 `byte` large, which allows to reach the maximum packet length defined by USB specifications. With this block size the allocated buffer size ranges from 32 to 1024 bytes, which is the longest packet size allowed by USB standard specifications.

Bits 14:10 NUM_BLOCK[4:0]: *Number of blocks*.

These bits define the number of memory blocks allocated to this packet buffer. The actual amount of allocated memory depends on the BL_SIZE value as illustrated in Table 66.

Bits 9:0 COUNTn_RX[9:0]. These bits contain the number of bytes received by the endpoint associated with the USB_EPnR register during the last OUT/ SETUP transaction addressed to it.

Table 66. Definition of Allocated Buffer Memory

Value of NUM_BLOCK[4:0]	Memory allocated when BL_SIZE=0	Memory allocated when BL_SIZE=1
0 (‘00000’)	Not allowed	32 bytes
1 (‘00001’)	2 bytes	64 bytes
2 (‘00010’)	4 bytes	96 bytes
3 (‘00011’)	6 bytes	128 bytes
...	...	...
15 (‘01111’)	30 bytes	512 bytes
16 (‘10000’)	32 bytes	544 bytes
17 (‘10001’)	34 bytes	576 bytes
18 (‘10010’)	36 bytes	608 bytes
...	...	...
30 (‘11110’)	60 bytes	992 bytes
31 (‘11111’)	62 bytes	1024 bytes

19.6.4 Register Map

A summary of the USB Peripheral registers is given in the following table.

Refer to Table 20 on page 49 for the base address..

Table 67. USB Peripheral Register Page Mapping

Offset	Register	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	USB_EP0R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x04	USB_EP1R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x08	USB_EP2R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x0C	USB_EP3R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x10	USB_EP4R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x14	USB_EP5R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x18	USB_EP6R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x1C	USB_EP7R	CTR_RX	DTOG_RX	STAT RX[1:0]		SETUP	EP TYPE[1:0]		EP_KIND	CTR TX	DTOG_TX	STAT TX[1:0]		EA[3:0]			
0x40	USB_CNTR	CTRM	DOVRM	ERRM	WKUPM	SUSPM	RESETM	SOFM	ESOFM	Reserved			RESUME	FSUSP	LP	PDWN	FRES
0x44	USB_ISTR	CTR	DOVR	ERR	WKUP	SUSP	RESET	SOF	ESOF	Reserved			DIR	EP_ID[3:0]			
0x48	USB_FNR	RXDP	RXDM	LCK	LSOF[1:0]		FN[10:0]										
0x4C	USB_DADDR	Reserved								EF	ADD[6:0]						
0x50	USB_BTABLE	BTABLE[15:3]													Reserved		

20 WATCHDOG TIMER (WDG)

20.1 Introduction

The Watchdog Timer peripheral can be used as free-running timer or as Watchdog to resolve STR720 microcontroller malfunctions due to hardware or software failures.

20.2 Main Features

- 16-bit down Counter
- 8-bit clock Prescaler
- Safe Reload Sequence
- Free-running Timer mode
- End of Counting interrupt generation
- Second clock source (4 kHz derived from 32 kHz RTC clock

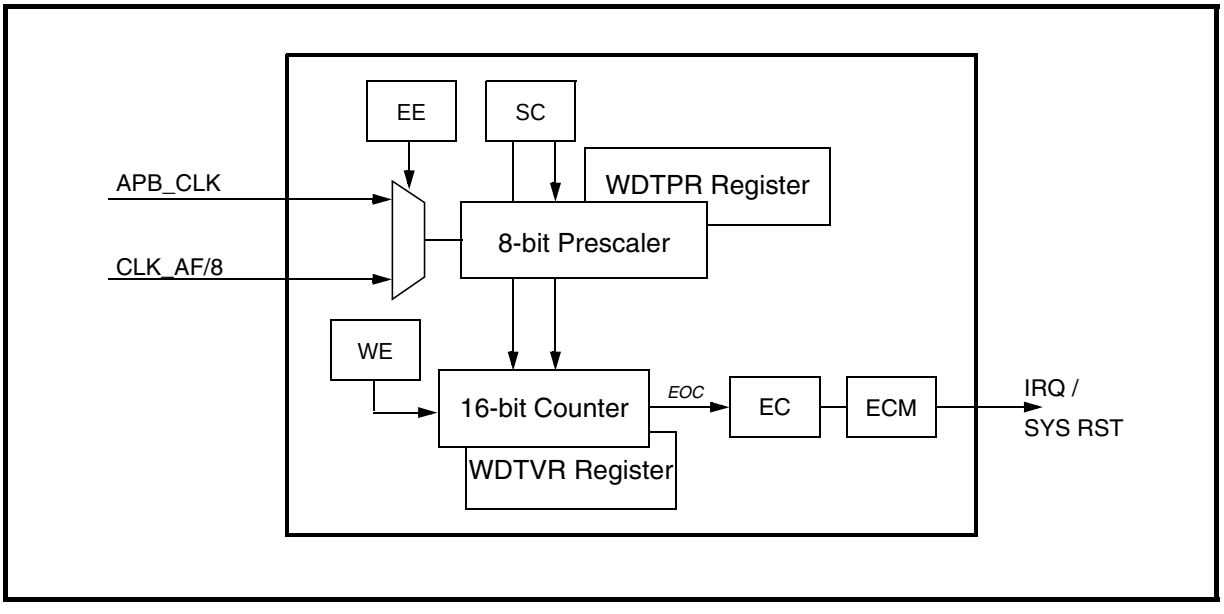
20.3 Functional Description

Figure 42 shows the functional blocks of the Watchdog Timer module. The module can work as Watchdog or as Free-running Timer. In both working modes the 16-bit Counter value can be accessed through a reading of the WDTCNT register.

20.3.1 Free-running Timer mode

If the WE bit of WDTCR register is not written to '1' by software, the peripheral enter in Free-running Timer mode. When in this operating mode as the SC bit of WDTCR register is written to '1' the WDTVR value is loaded in the Counter and the Counter starts counting down.

Figure 42. Watchdog Timer Functional Block



When it reaches the end of count value (0000h) an End of Count interrupt is generated (EC) and the WDTVR value is automatically re-loaded. The Counter runs until the SC bit is cleared. Setting SC bit again, both the Counter and the Prescaler are re-loaded with the values contained in registers WDTVR and WDTPR respectively, so it does not restart from where it was eventually stopped, but from a defined situation without need to reset and re-program the module. It is clear that, on the other hand, it is not possible to change on fly the prescaling factor since it will only effect the counter after a restart command (SC bit setting which generates a re-load operation).

The clock input signal can be either the APB_CLK or an alternate clock signal, having at least a period four times longer than the period of the APB clock. This can allow for example to generate time basis independent from the APB clock which could dynamically change according to the system mode setting (run mode, low power mode, etc.). This alternate clock signal is derived from the RTC clock and is equal to CLK_AF divided by 8.

20.3.2 Watchdog mode

If WE bit of WDTCR register is written to ‘1’ by software, the peripheral enter in Watchdog mode. This operating mode can not be changed by software (the SC bit has no effect and WE bit cannot be cleared).
As the peripheral enters in this operating mode, the WDTVR value is loaded in the Counter and the Counter starts counting down. When it reaches the end of count value (0000h) a system reset signal is generated (SYS RST).

If the sequence of two consecutive values A55Ah, 5AA5h is written in the WDTKR register see Section 20.4, the WDTVR value is re-loaded in the Counter, so the end of count can be avoided.

20.4 Register description

The Watchdog Timer registers can not be accessed by byte.
The reserved bits can not be written and they are always read at ‘0’.

WDT Control Register (WDTCR)

Address Offset: 00h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved													EE	SC	WE
-													rw	rw	rw

Bit 15:3 = *Reserved.*

Bit 2 = **EE**: *EXT_CK Enable bit.*
1: EXT_CK signal, (CLK_AF/8 is used as counting clock, its period must be at least 4 times the APB_CLK period.
0: APB_CLK is used as counting clock.
This bit can be written as long as Watchdog mode is not entered (WE bit = 0). Once WE bit is set to ‘1’, the value of EE cannot be changed anymore.

Bit 1 = **SC**: *Start Counting bit.*
1: the Prescaler loads the Prescaler pre-load value (WDTPR), the Counter loads the Timer pre-load value (WDTVR) and starts counting
0: the Counter is stopped. To restart it, SC setting will generate a re-loading of the Prescaler pre-load value and Timer pre-load value. These functionalities are permitted only in Timer Mode (WE bit = 0).

Bit 0 = **WE**: *Watchdog Enable bit.*
1: Watchdog Mode is enabled
0: Timer Mode is enabled
This bit can’t be reset by software.
If the external WDGEN signal is high the WE bit is written to ‘1’ by hardware as soon the reset has elapsed else can write to ‘1’ by software but cannot be cleared.
When WE bit is high, SC bit has no effect.

WDT Prescaler Register (WDTPR)

Address Offset: 04h

Reset value: 00FFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
-								rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:8 = *Reserved.*

Bit 7:0 = **PR[7:0]**: *Prescaler value.*
The clock to Timer Counter is divided by PR[7:0]+1.
This value takes effect when Watchdog mode is enabled (WE bit is put to '1') or the re-load sequence occurs or the Counter starts (SC) bit is put to '1' in Timer mode.

WDT pre-load Value Register (WDTVR)

Address Offset: 08h

Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TV15	TV14	TV13	TV12	TV11	TV10	TV9	TV8	TV7	TV6	TV5	TV4	TV3	TV2	TV1	TV0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:0 = **TV[15:0]**: *Timer Pre-load Value*
This value is loaded in the Timer Counter when it starts counting or a re-load sequence occurs or an End of Count is reached. The time (µs) need to reach the end of count is given by:

$$(PR[7:0]+1)*(TV[15:0]+1)*T_{CK}/1000\ (\mu s)$$

where T_{CK} is the Watchdog clock period measured in ns.
I.e. if F_{CK} = 20MHz the default timeout set after the system reset is 256*65536*50/1000 = 838861µs.

WDT Counter Register (WDTCNT)

Address Offset: 0Ch

Reset value: FFFFh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT15	CNT14	CNT13	CNT12	CNT11	CNT10	CNT9	CNT8	CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bit 15:0 = **CNT[15:0]:** *Timer Counter Value.*
The current counting value of the 16-bit Counter is available reading this register.

WDT Status Register (WDTSR)

Address Offset: 10h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved															EC
-															rc

Bit 15:1 = *Reserved*.

Bit 0 = **EC**: *End of Count pending bit.*
1: the End of Count has occurred
0: no End of Count has occurred
In Watchdog Mode (WE = 1) this bit has no effect.
This bit can be set only by hardware and must be reset by software.

WDT Mask Register (WDTMR)

Address Offset: 14h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved															ECM
-															rw

Bit15:1 = *Reserved*.

Bit 0 = **ECM**: *End of Count Mask bit.*
1: End of Count interrupt request is enabled
0: End of Count interrupt request is disabled

WDT Key Register (WDTKR)

Address Offset: 18h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
K15	K14	K13	K12	K11	K10	K9	K8	K7	K6	K5	K4	K3	K2	K1	K0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:0 = **K[15:0]:** *Key Value.*
When Watchdog Mode is enabled, writing in this register the two consecutive values (A55Ah, 5AA5h) the Counter is initialized to TV[15:0] value and the Prescaler value in WTDPR register take effect. Any number of instructions can be executed between the two writes.
If Watchdog Mode is disabled (WE = 0) a writing in this register has no effect.
The reading value of this register is 0000h.

20.4.1 Register Map

A summary of the WDG registers is given in the following table.

Table 68. Watchdog Timer Peripheral Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**WDTCR**	reserved													EE	SC	WE
4	**WDTPR**	reserved								PR(7:0)							
8	**WDTVR**	TV(15:0)															
C	**WDTCNT**	CNT(15:0)															
10	**WDTSR**	reserved															EC
14	**WDTMR**	reserved															MEC
18	**WDTKR**	K[15:0]															

Refer to Table 20 on page 49 for the base address.

21 EXTENDED FUNCTION TIMER (EFT)

21.1 Introduction

The timer consists of a 16-bit counter driven by a programmable prescaler.
It may be used for a variety of purposes, including pulse length measurement of up to two input signals (input capture) or generation of up to two output waveforms (output compare and PWM).
Pulse lengths and waveform periods can be modulated from a very wide range using the timer prescaler.

Note *The two EFT instances available in STR720 device do not implement all the features described in this chapter. Please refer to* Section 5.16: *Extended Function Timer (EFT) on page 43*.

21.2 Main Features

- Programmable prescaler: f_{APB} divided from 1 to 256, Prescaler register (0 to 255) value +1.
- Overflow status flag and maskable interrupts
- External clock input (must be at least 4 times slower than the CPU clock speed) with the choice of active edge
- Output compare functions with
  - 2 dedicated 16-bit registers
  - 2 dedicated programmable signals
  - 2 dedicated status flags
  - 2 dedicated interrupt flags.
- Input capture functions with
  - 2 dedicated 16-bit registers
  - 2 dedicated active edge selection signals
  - 2 dedicated status flags
  - 2 dedicated interrupt flags.
- Pulse width modulation mode (PWM)
- One pulse mode (OPM)
- PWM input mode
- Timer global interrupt (5 internally ORed or separated sources, depending on device)
  - ICIA: Timer Input capture A interrupt
  - ICIB: Timer Input capture B interrupt
  - OCIA: Timer Output compare A interrupt
  - OCIB: Timer Output compare B interrupt
  - TOI: Timer Overflow interrupt.

The Block Diagram is shown in Figure 43.

21.3 Functional Description

21.3.1 Counter

The principal block of the Programmable Timer is a 16-bit counter and its associated 16-bit registers.

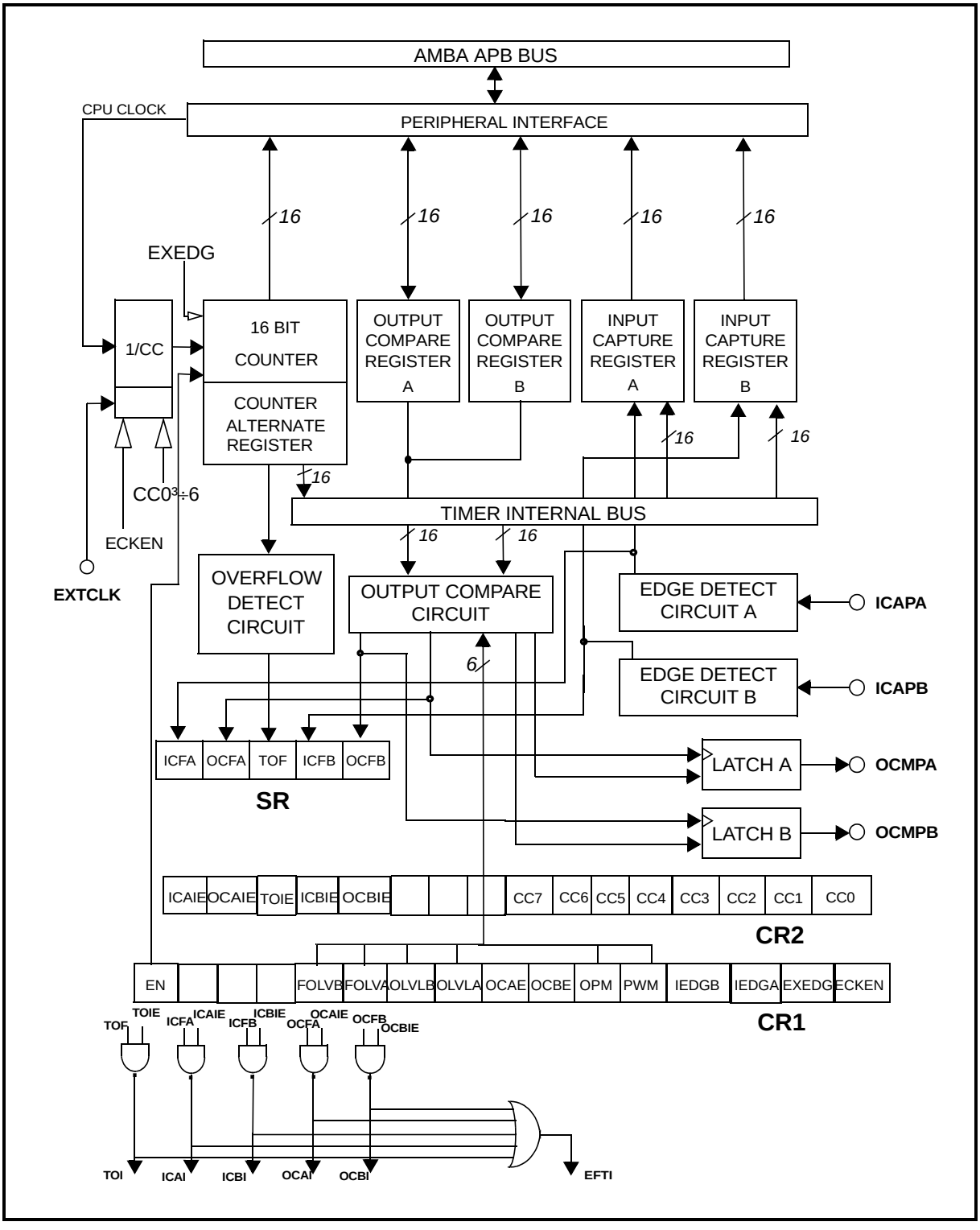
Writing in the Counter Register (CNTR) resets the counter to the FFFCh value.

The timer clock source can be either internal or external selecting ECKEN bit of CR1 register. When ECKEN = 0, the frequency depends on the prescaler division bits (CC7-CC0) of the CR2 register.

An overflow occurs when the counter rolls over from FFFFh to 0000h then the TOF bit of the SR register is set. An interrupt is generated if TOIE bit of the CR2 register is set; if this condition is false, the interrupt remains pending to be issued as soon as it becomes true.

Clearing the overflow interrupt request is done by a write access to the SR register while the TOF bit is set with the data bus 13-bit at ‘0’, while all the other bits shall be written to ‘1’ (the SR register is clear only, so writing a ‘1’ in a bit has no effect: this makes possible to clear a pending bit without risking to clear a new coming interrupt request from another source).

Figure 43. Timer Block Diagram



21.3.2 External Clock

The external clock (where available) is selected if ECKEN = 1 in CR1 register.

The status of the EXEDG bit determines the type of level transition on the external clock pin EXTCLK that will trigger the counter.

The counter is synchronized with the rising edge of the internal clock coming from the APB block.

At least four rising edges of the APB_CLK must occur between two consecutive active edges of the external clock; thus the external clock frequency must be less than a quarter of the APB clock frequency.

Figure 44. Counter Timing Diagram, internal clock divided by 2

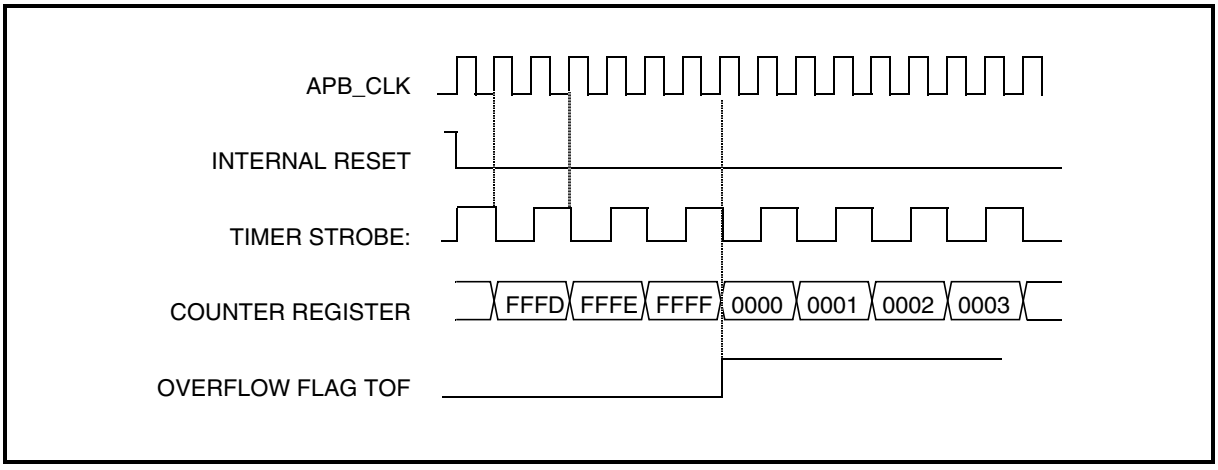


Figure 45. Counter Timing Diagram, internal clock divided by 4

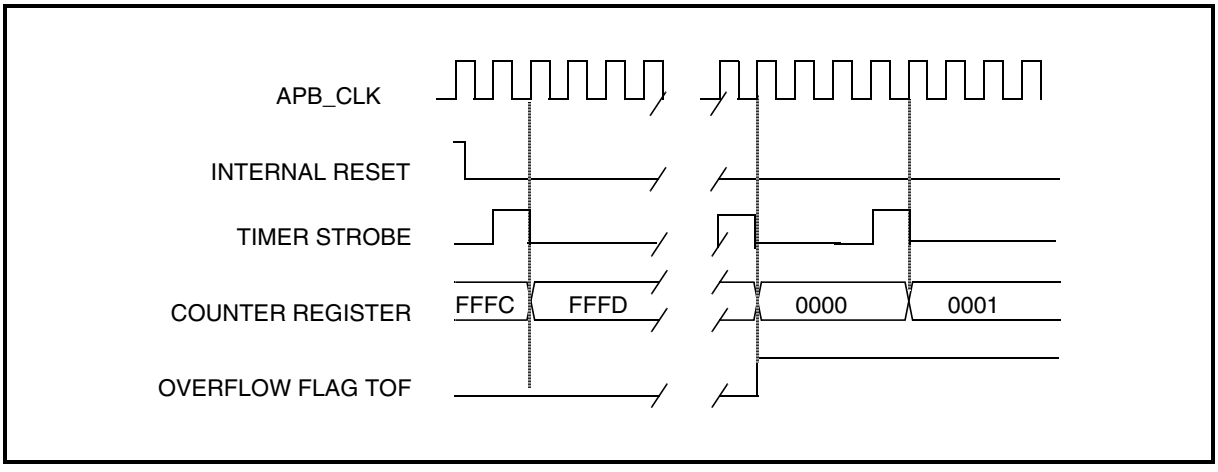
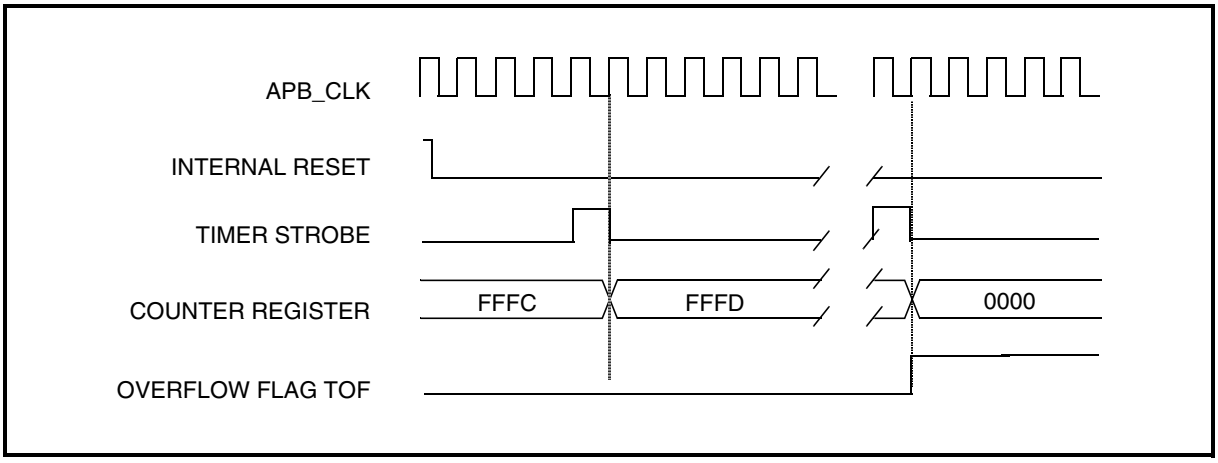


Figure 46. Counter Timing Diagram, internal clock divided by n



According to particular device implementation, the external clock can be available on a general purpose I/O pin as alternate function, this allows the Timer to count events independently by the system clock (which could be prescaled or multiplied according to the different run and low-power modes), generating regular time basis.

21.3.3 Input Capture

In this section, the index "*i*", may be A or B.

The two input capture 16-bit registers (ICAR and ICBR) are used to latch the value of the counter after a transition detected by the ICAP*i* pin (see Figure 47).

IC*i*R register are read-only registers.

The active transition is software programmable through the IEDG*i* bit of the Control Register (CR1).

Timing resolution is one/two count of the counter: (f_{APB}/(CC7÷CC0+1)).

21.3.3.1 Procedure

To use the input capture function select the following in the CR1 and CR2 registers:

– Select the timer clock source (ECKEN).
– Select the timer clock division factor (CC7÷CC0) if internal clock is used.
– Select the edge of the active transition on the ICAPA pin with the IEDGA bit, if ICAPA is active.
– Select the edge of the active transition on the ICAPB pin with the IEDGB bit, if ICAPB is active.
– Set ICAIE (or ICBIE) when ICAPA (or ICAPB) is active, to generate an interrupt after an input capture.

When an input capture occurs:

– ICF*i* bit is set.
– The IC*i*R register contains the value of the counter on the active transition on the ICAP*i* pin (see Figure 48).
– A timer interrupt is generated if ICAIE is set (if only ICAPA is active) or ICBIE is set (if only ICAPB is active); otherwise, the interrupt remains pending until concerned enable bits are set.

Clearing the Input Capture interrupt request is done by:

1. A write access to the SR register while the ICF*i* bit is cleared, 15-bit at '0' for ICAPA and 12-bit at '0' for ICAPB.

Figure 47. Input Capture Block Diagram

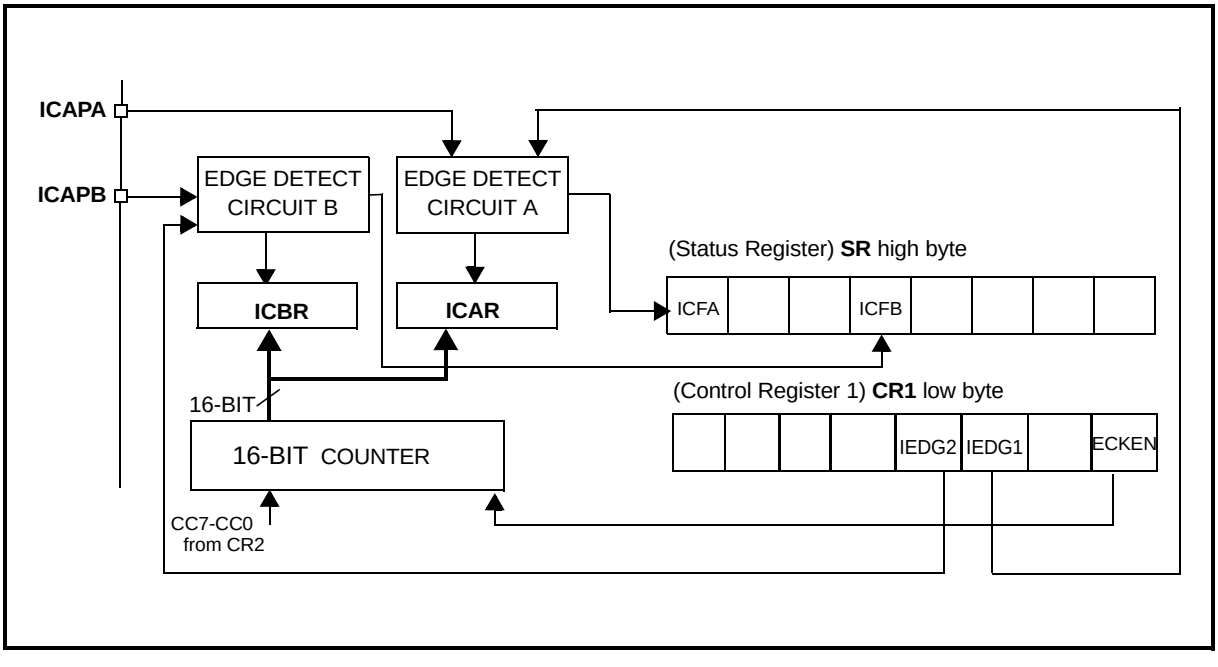
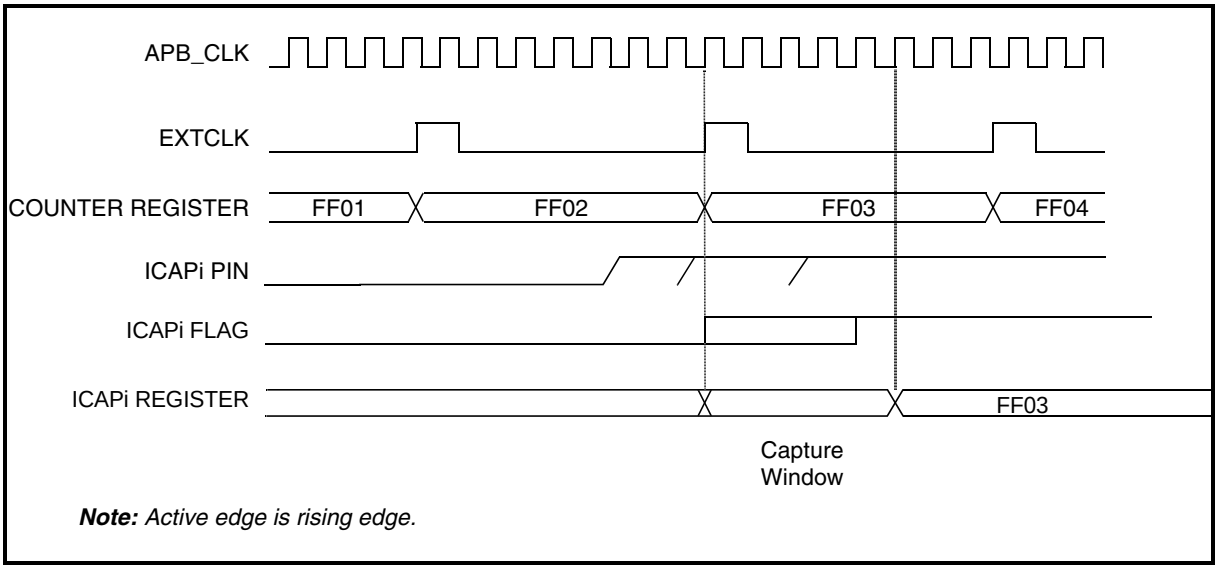


Figure 48. Input Capture Timing Diagram



Note: *Active edge is rising edge.*

21.3.4 Output Compare

In this section, the index "*i*", may be A or B.

This function can be used to control an output waveform or indicating when a period of time has elapsed.

When a match is found between the Output Compare register and the counter, the output compare function:

– Assigns pins with a programmable value if the OC*i*E bit is set
– Sets a flag in the status register
– Generates an interrupt if enabled

Two 16-bit registers Output Compare Register A (OCAR) and Output Compare Register B (OCBR) contain the value to be compared to the counter each timer clock cycle.

These registers are readable and writable and are not affected by the timer hardware. A reset event changes the OC*i*R value to 8000h.

Timing resolution is one count of the counter: (f_{APB}/(CC7÷CC0+1)).

21.3.4.1 Procedure

To use the output compare function, select the following in the CR1/CR2 registers:

– Set the OC*i*E bit if an output is needed then the OCMP*i* pin is dedicated to the output compare *i* function.
– Select the timer clock (ECKGEN) and the prescaler division factor (CC7÷CC0).

Select the following in the CR1/CR2 registers:

– Select the OLVL*i* bit to applied to the OCMP*i* pins after the match occurs.
– Set OCAIE (OCBIE) if only compare A (compare B) needs to generate an interrupt.

When match is found:

– OCF*i* bit is set.
– The OCMP*i* pin takes OLVL*i* bit value (OCMP*i* pin latch is forced low during reset and stays low until valid compares change it to OLVL*i* level).
– A timer interrupt is generated if the OCAIE (or OCBIE) bit in CR2 register is set, the OCAR (or OCBR) matches the timer counter (i.e. OCFA or OCFB is set).

Clearing the output compare interrupt request is done by a write access to the SR register while the OCF*i* bit is cleared, 14-bit at ‘0’ for OCAR and 12-bit at ‘0’ for OCBR.

If the OC*i*E bit is not set, the OCMP*i* pin is at ‘0’ and the OLVL*i* bit will not appear when match is found.

The value in the 16-bit OC*i*R register and the OLVL*i* bit should be changed after each successful comparison in order to control an output waveform or establish a new elapsed timeout.

The OC*i*R register value required for a specific timing application can be calculated using the following formula:

$$\Delta \text{ OC}i\text{R} = \frac{\Delta t * f_{APB}}{(CC7 \div CC0+1)}$$

Where:

Δt = Desired output compare period (in seconds)
f_{APB} = APB clock frequency
CC7÷CC0 = Timer clock prescaler

Figure 49. Output Compare Block Diagram

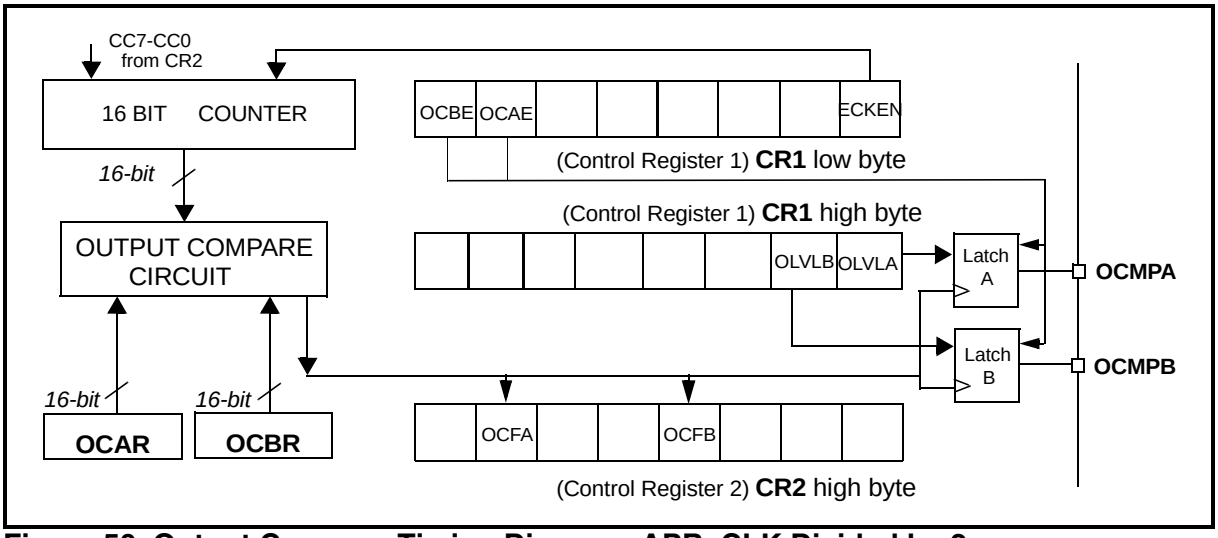
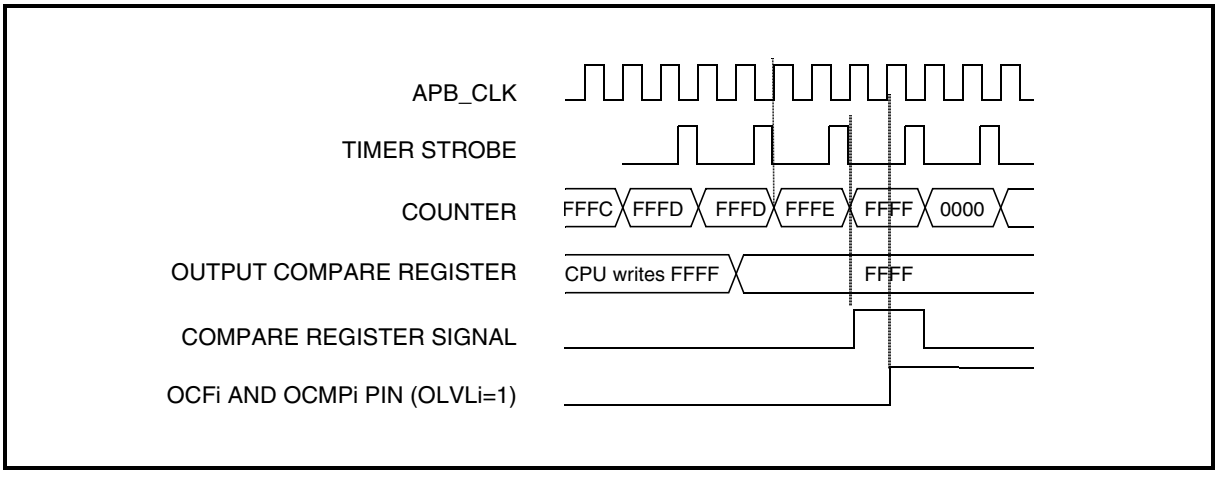


Figure 50. Output Compare Timing Diagram, APB_CLK Divided by 2



21.3.5 Forced Compare Mode

In this section the index "*i*" may represent A or B.

Bits 11:8 of CR1 register and bits 7:0 of CR2 are used (Refer to Section 21.4 for detailed Register Description).

When the FOLVA bit is set, the OLVLA bit is copied to the OCMPA pin if PWM and OPM are both cleared. When FOLVB bit is set, the OLVLB bit is copied to the OCMPB pin.

The OLVL*i* bit has to be toggled in order to toggle the OCMP*i* pin when it is enabled (OC*i*E bit=1).

Note

– When FOLV*i* is set, no interrupt request is generated.
– Nevertheless the OCF*i* bit can be set if OC*i*R = Counter, an interrupt can be generated if enabled.
– Input capture function works in Forced Compare mode.

21.3.6 One Pulse Mode

One Pulse mode enables the generation of a pulse when an external event occurs. This mode is selected via the OPM bit in the CR1 register.

The one pulse mode uses the Input Capture A function (trigger event) and the Output Compare A function.

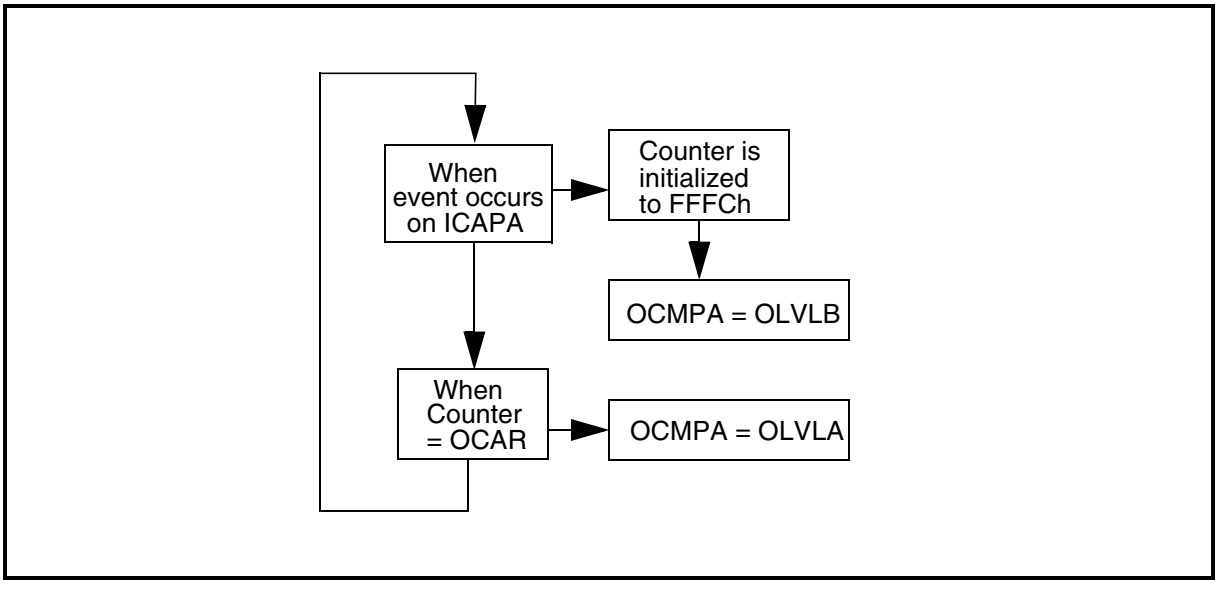
21.3.6.1 Procedure

To use one pulse mode, select the following in the CR1 register:

– Using the OLVLA bit, select the level to be applied to the OCMPA pin after the pulse.
– Using the OLVLB bit, select the level to be applied to the OCMPA pin during the pulse.
– Select the edge of the active transition on the ICAPA pin with the IEDGA bit.
– Set the OCAE bit, the OCMPA pin is then dedicated to the Output Compare A function.
– Set the OPM bit.
– Select the timer clock (ECKGEN) and the prescaler division factor (CC7-CC0).

Load the OCAR register with the value corresponding to the length of the pulse (see the formula in next Section 21.3.7.1).

Figure 51. One Pulse Mode Cycle

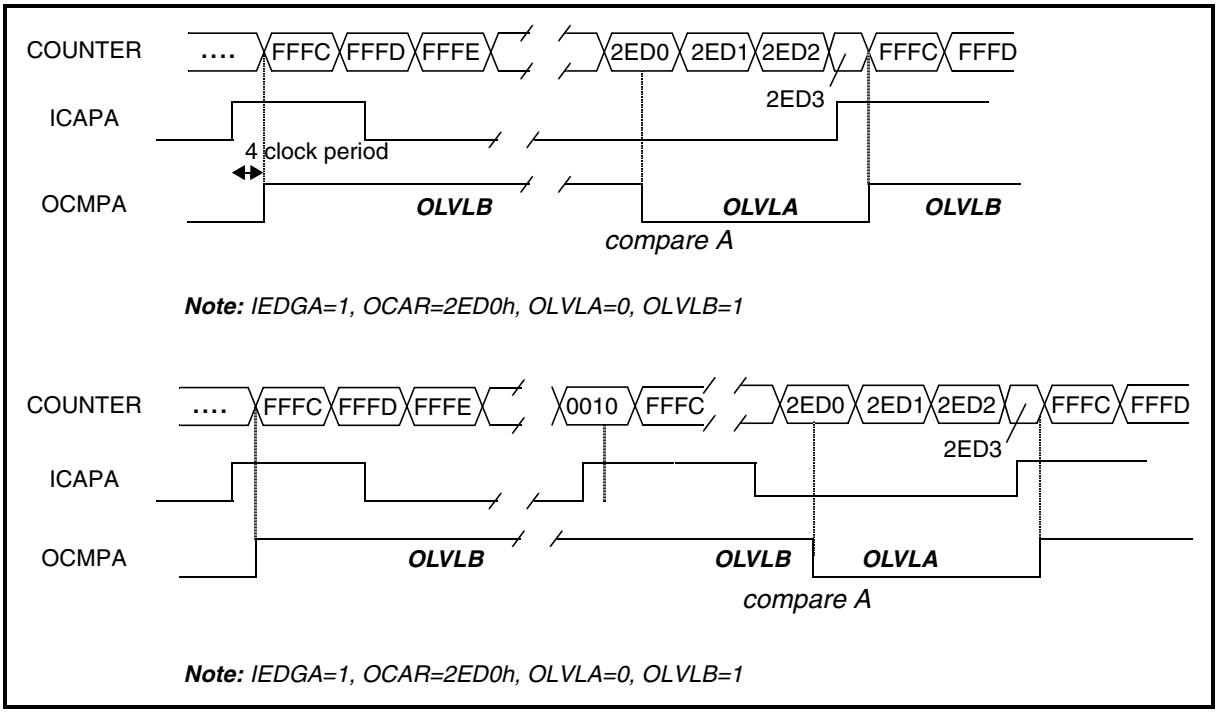


Then, on a valid event on the ICAPA pin, the counter is initialized to FFFCh and OLVLB bit is loaded on the OCMPA pin after four clock period. When the value of the counter is equal to the value of the contents of the OCAR register, the OLVLA bit is output on the OCMPA pin (See Figure 52).

Note

– The OCFA bit cannot be set by hardware in one pulse mode but the OCFB bit can generate an Output Compare interrupt.
– The ICFA bit is set when an active edge occurs and can generate an interrupt if the ICAIE bit is set. The ICAR register will have the value FFFCh.
– When the Pulse Width Modulation (PWM) and One Pulse Mode (OPM) bits are both set with FOLVA= 1, the OPM mode is the only active one, otherwise the PWM mode is the only active one.
– Forced Compare B mode works in OPM
– Input Capture B function works in OPM
– When OCAR = FFFBh in OPM, then a pulse of width FFFFh is generated
– If event occurs on ICAPA again before the Counter reaches the value of OCAR, then the Counter will be reset again and the pulse generated might be longer than expected as in Figure 52.
– If a write operation is performed on the counter register before the Counter reaches the value of OCAR, then the Counter will be reset again and the pulse generated might be longer than expected.
– If a write operation is performed on the counter register after the Counter reaches the value of OCAR, then there will have no effect on the waveform.

Figure 52. One Pulse Mode Timing



21.3.7 Pulse Width Modulation Mode

Pulse Width Modulation mode enables the generation of a signal with a frequency and pulse length determined by the value of the OCAR and OCBR registers.

The pulse width modulation mode uses the complete Output Compare A function plus the OCBR register.

21.3.7.1 Procedure

To use pulse width modulation mode select the following in the CR1 register:

– Using the OLVLA bit, select the level to be applied to the OCMPA pin after a successful comparison with OCAR register.
– Using the OLVLB bit, select the level to be applied to the OCMPA pin after a successful comparison with OCBR register.
– Set OCAE bit: the OCMPA pin is then dedicated to the output compare A function.
– Set the PWM bit.
– Select the timer clock (ECKGEN) and the prescaler division factor (CC7-CC0).

Load the OCBR register with the value corresponding to the period of the signal.

Load the OCAR register with the value corresponding to the length of the pulse if (OLVLA=0 and OLVLB=1).

If OLVLA=1 and OLVLB=0 the length of the pulse is the difference between the OCBR and OCAR registers.

The OC*i*R register value required for a specific timing application can be calculated using the following formula:

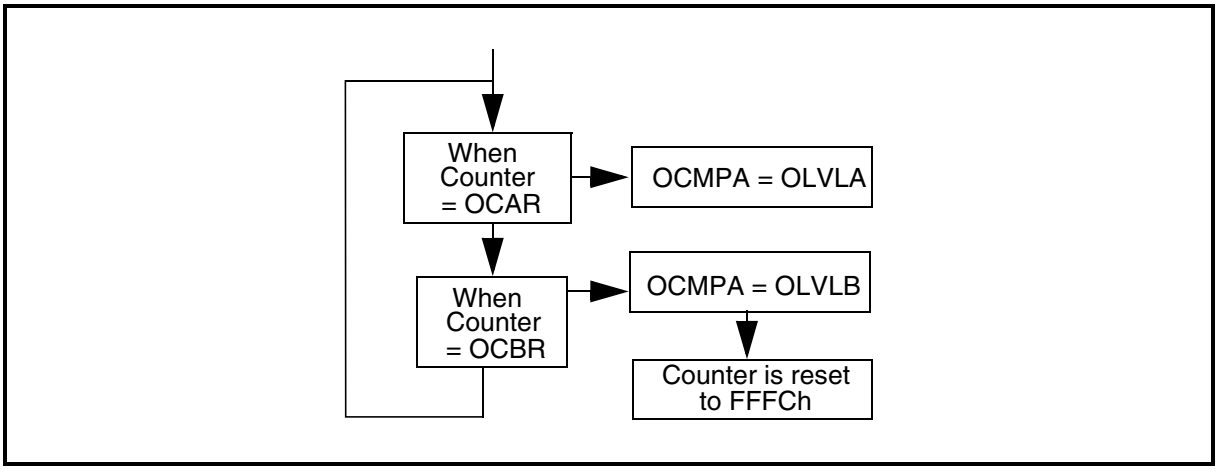
$$OCiR\ Value = \frac{t * f_{APB}}{t_{PRESC}} - 5$$

Where:

t = Desired output compare period (seconds)
f_{APB} = APB clock frequency (Hertz)
t_{PRESC} = Timer clock prescaler ((CC7..CC0)+1)

The Output Compare B event causes the counter to be initialized to FFFCh (See Figure 54).

Figure 53. Pulse Width Modulation Mode Cycle



Note

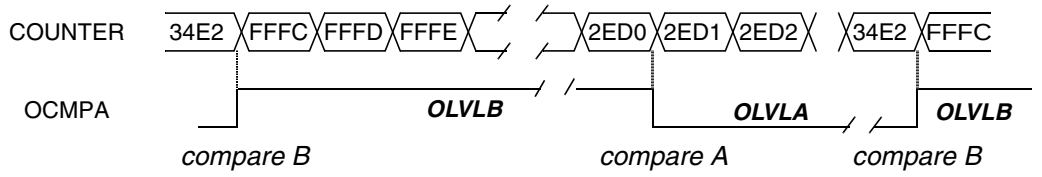
– The OCFA bit cannot be set by hardware in PWM mode, but OCFB is set every time counter matches OCBR.
– The Input Capture function is available in PWM mode.
– When Counter = OCBR, then OCFB bit will be set. This can generate an interrupt if OCBIE is set. This interrupt will help any application where pulse-width or period needs to be changed interactively.
– When the Pulse Width Modulation (PWM) and One Pulse Mode (OPM) bits are both set with FOLVA = 0, the PWM mode is the only active one, otherwise the OPM mode is the only active one.
– The value loaded in OCBR **must always be greater than** that in OCAR to produce meaningful waveforms. Note that 0000h is considered to be greater than FFFCh or FFFDh or FFFEh or FFFFh.
– When OCAR > OCBR, no waveform will be generated.

– When OCBR = OCAR, a **square** waveform with 50% duty cycle will be generated as in Figure 54.
– When OCBR and OCAR are loaded with FFFC (the counter reset value) then a square waveform will be generated & the counter will remain stuck at FFFC. The period will be calculated using the following formula:

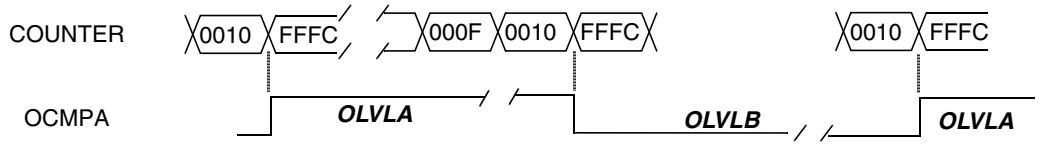
$$Period = t_{CPU} \cdot (PRESC + 1) \cdot (OCBR + 1)$$

– When OCAR is loaded with FFFC (the counter reset value) then the waveform will be generated as in Figure 54.
– When FOLVA bit is set and PWM bit is set, then PWM mode is the active one. But if FOLVB bit is set then the OLVLB bit will appear on OCMPB (when OCBE bit = 1).
– When a write is performed on CNTR register in PWM mode, then the Counter will be reset and the pulse-width/period of the waveform generated may not be as desired.

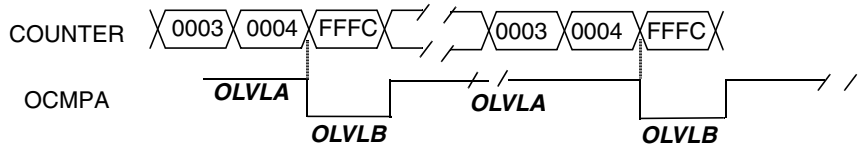
Figure 54. Pulse Width Modulation Mode Timing



Note: *OCAR = 2ED0h, OCBR = 34E2, OLVLA = 0, OLVLB = 1*



Note: *OCAR = OCBR = 0010h, OLVLA = 1, OLVLB = 0*



Note: *OCAR = FFFCh, OCBR = 0004h, OLVLA = 1, OLVLB = 0*

21.3.8 Pulse Width Modulation Input

The PWM Input functionality enables the measurement of the period and the pulse width of an external waveform. The initial edge is programmable.

It uses the two Input Capture registers and the Input signal of the Input Capture A module.

21.3.8.1 Procedure

The CR2 register must be programmed as needed for Interrupts and DMAs. To use pulse width modulation mode select the following in the CR1 register:

– set the PWMI bit
– Select the first edge in IEDGA
– Select the second edge IEDGB as the negated of IEDGA
– Program the clock source and prescaler as needed
– Enable the counter setting the EN bit.

To have a coherent measure the interrupt/DMA should be linked to the Input Capture A Interrupt, reading in ICAR the period value and in ICBR the pulse width.

To obtain the time values:

$$\text{Period} = \frac{ICAR * f_{APB}}{t_{PRESC}}$$

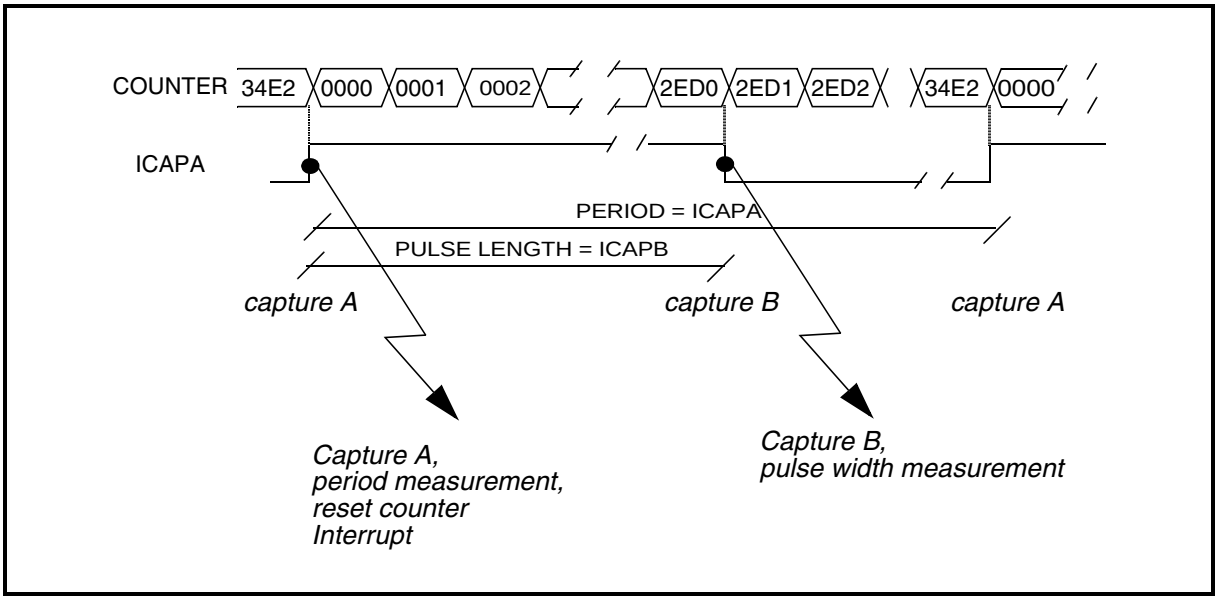
$$\text{Pulse} = \frac{ICBR * f_{APB}}{t_{PRESC}}$$

Where:

f_{APB} = APB clock frequency
t_{PRESC} = Timer clock prescaler: (CC7..CC0)+1

The Input Capture A event causes the counter to be initialized to 0000h, allowing a new measure to start. The first Input Capture on ICAPA do not generate the corresponding interrupt request.

Figure 55. Pulse Width Modulation Input Mode Timing



Note All formulas in this chapter assume that APB_CLK is the clock driving the EFT. If External clock (F_{EXT}) is selected, replace (F_{APB} / T_{PRESC}) by F_{EXT}., where T_{PRESC} = (CC7..CC0 + 1).

21.4 Register Description

Each Timer is associated with two control and one status registers, and with six pairs of data registers (16-bit values) relating to the two input captures, the two output compares, the counter. Every register can have only an access by 16 bits, that means is not possible to read or write only a byte.

Input Capture A Register (ICAR)

Address Offset: 00h

Reset value: xxxxh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

This is a 16-bit read only register that contains the counter value transferred by the Input Capture A event.

Input Capture B Register (ICBR)

Address Offset: 04h

Reset value: xxxxh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

This is a 16-bit read only register that contains the counter value transferred by the Input Capture B event.

Output Compare A Register (OCAR)

Address Offset: 08h

Reset value: 8000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This is a 16-bit register that contains the value to be compared to the CNTR register and signalled on OCMPA output.

Output Compare B Register (OCBR)

Address Offset: 0Ch

Reset value: 8000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

This is a 16-bit register that contains the value to be compared to the CNTR register and signalled on OCMPB output.

Counter Register (CNTR)

Address Offset: 10h

Reset value: FFFCh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MSB															LSB
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

This is a 16-bit register that contains the counter value.

Control Register 1 (CR1)

Address Offset: 14h

Reset value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN	PWMI	Reserved		FOLVB	FOLVA	OLVLB	OLVLA	OCBE	OCAE	OPM	PWM	IEDGB	IEDGA	EXEDG	ECKEN
rw	rw	-		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 = **EN**: *Timer Count Enable*
0: Timer counter is stopped.
1: Timer counter is enabled.

Bit 14 = **PWMI**: *Pulse Width Modulation Input*
0: PWM Input is not active.
1: PWM Input is active.

Bit 13:12 = Reserved. These bits must be always written to 0.

Bit 11 = **FOLVB**: *Forced Output Compare B*
0: No effect.
1: Forces OLVLB to be copied to the OCMPB pin.

Bit 10 = **FOLVA**: *Forced Output Compare A*
0: No effect.
1: Forces OLVLA to be copied to the OCMPA pin.

Bit 9 = **OLVLB**: *Output Level B*
This bit is copied to the OCMPB pin whenever a successful comparison occurs with the OCBR register and OCBE is set in the CR2 register. This value is copied to the OCMPA pin in One Pulse Mode and Pulse Width Modulation mode.

Bit 8= **OLVLA**: *Output Level A*
The OLVLA bit is copied to the OCMPA pin whenever a successful comparison occurs with the OCAR register and the OCAE bit is set in the CR2 register.

Bit 7= **OCBE**: *Output Compare B Enable*
0: Output Compare B function is enabled, but the OCMPB pin is a general I/O.
1: Output Compare B function is enabled, the OCMPB pin is dedicated to the Output Compare B capability of the timer.

Bit 6= **OCAE**: *Output Compare A Enable*
0: Output Compare A function is enabled, but the OCMPA pin is a general I/O.
1: Output Compare A function is enabled, the OCMPA pin is dedicated to the Output Compare A capability of the timer.

Bit 5 = **OPM**: *One Pulse Mode*
0: One Pulse Mode is not active.
1: One Pulse Mode is active, the ICAPA pin can be used to trigger one pulse on the OCMPA pin; the active transition is given by the IEDGA bit. The length of the generated pulse depends on the contents of the OCAR register.

Bit 4 = **PWM**: *Pulse Width Modulation*
0: PWM mode is not active.
1: PWM mode is active, the OCMPA pin outputs a programmable cyclic signal; the length of the pulse depends on the value of OCAR register; the period depends on the value of OCBR register.

Bit 3 = **IEDGB**: *Input Edge B*
This bit determines which type of level transition on the ICAPB pin will trigger the capture.
0: A falling edge triggers the capture.
1: A rising edge triggers the capture.

Bit 2 = **IEDGA**: *Input Edge A*
This bit determines which type of level transition on the ICAPA pin will trigger the capture.
0: A falling edge triggers the capture.
1: A rising edge triggers the capture.

Bit 1 = **EXEDG**: *External Clock Edge*
This bit determines which type of level transition on the external clock pin (or internal signal) EXTCLK will trigger the counter.
0: A falling edge triggers the counter.
1: A rising edge triggers the counter.

Bit 0 = **ECKEN**: *External Clock Enable*
0: Internal clock, divided by prescaler division factor, is used to feed timer clock.
1: External source is used for timer clock.

Control Register 2 (CR2)

Address Offset: 18h

Reset value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICAIE	OCAIE	TOE	ICBIE	OCBIE	reserved			CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0
rw	rw	rw	rw	rw	-			rw	rw	rw	rw	rw	rw	rw	rw

Bit 15 = **ICAIE**: *Input Capture A Interrupt Enable*
0: No interrupt on input capture A.
1: Generate interrupt if ICFA flag is set.

Bit 14 = **OCAIE**: *Output Compare A Interrupt Enable*
0: No interrupt on OCFA set.
1: Generate interrupt if OCFA flag is set.

Bit 13 = **TOIE**: *Timer Overflow Interrupt Enable*
0: Interrupt is inhibited.
1: A timer interrupt is enabled whenever the TOF bit of the SR register is set.

Bit 12= **ICBIE**: *Input Capture B Interrupt Enable*
0: No interrupt on input capture B.
1: Generate interrupt if ICFB flag is set.

Bit 11 = **OCBIE**: *Output Compare B Interrupt Enable*
0: No interrupt on OCFB set.
1: Generate interrupt if OCFB flag is set.

Bit 10:8 = Reserved. These bits must be always written to 0.

Bit 7:0 = **CC7-CC0**: *Prescaler division factor*
This 7-bit string is the factor used by the prescaler to divide the internal clock. Timer clock will be equal to f_{APB} / CC7÷CC0.

Status Register (SR)

Address Offset: 1Ch

Reset value: 0001h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ICFA	OCFA	TOF	ICFB	OCFB	reserved										
rc	rc	rc	rc	rc	-										

Bit 15= **ICFA**: *Input Capture Flag A*
0: No input capture (reset value).
1: An input capture has occurred. To clear this bit, write the SR register, with a '0' on the bit 15 (and '1' in all the other bit, just to avoid an unwanted clearing of another pending bit).

Bit 14= **OCFA**: *Output Compare Flag A*
0: No match (reset value).
1: The content of the counter has matched the content of the OCAR register. This bit is not set in the PWM mode even if counter matches OCAR. To clear this bit, write the SR register, with a '0' on the bit 14 (and '1' in all the other bit, just to avoid an unwanted clearing of another pending bit).

Bit 13= **TOF**: *Timer Overflow*
0: No timer overflow (reset value).
1:The counter rolled over from FFFFh to 0000h. To clear this bit, write the SR register, with a '0' on the bit 13 (and '1' in all the other bit, just to avoid an unwanted clearing of another pending bit).

Bit 12= **ICFB**: *Input Capture Flag B*
0: No input capture (reset value).
1: An input capture has occurred.To clear this bit, write the SR register, with a '0' on the bit 12 (and '1' in all the other bit, just to avoid an unwanted clearing of another pending bit).

Bit 11= **OCFB**: *Output Compare Flag B*
0: No match (reset value).
1: The content of the counter has matched the content of the OCBR register. It is set in PWM mode too. To clear this bit, write the SR register, with a '0' on the bit 11 (and '1' in all the other bit, just to avoid an unwanted clearing of another pending bit).

21.4.1 Register Map

A summary of the EFT registers is given in the following table.

Table 69. Extended Function Timer Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**ICAR**	Input Capture A															
4	**ICBR**	Input Capture B															
8	**OCAR**	Output Compare A															
C	**OCBR**	Output Compare B															
10	**CNTR**	Counter Value															
14	**CR1**	EN	PWMI	reserved		FOLVB	FOLVA	OLVLB	OLVLA	OCBE	OCAE	OPM	PWM	IEDGB	IEDGA	EXEDG	ECKEN
18	**CR2**	ICAIE	OCAIE	TOE	ICBIE	OCBIE	reserved			CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0
1C	**SR**	ICFA	OCFA	TOF	ICFB	OCFB	reserved										

Refer to Table 20 on page 49 for the base address.

22 Σ–Δ ANALOG/DIGITAL CONVERTER (ADC)

22.1 Introduction

The ADC is used in STR720 to measure signal strength and other slowly-changing signals. Four input channels are supported, which can be converted in single channel or round robin mode.

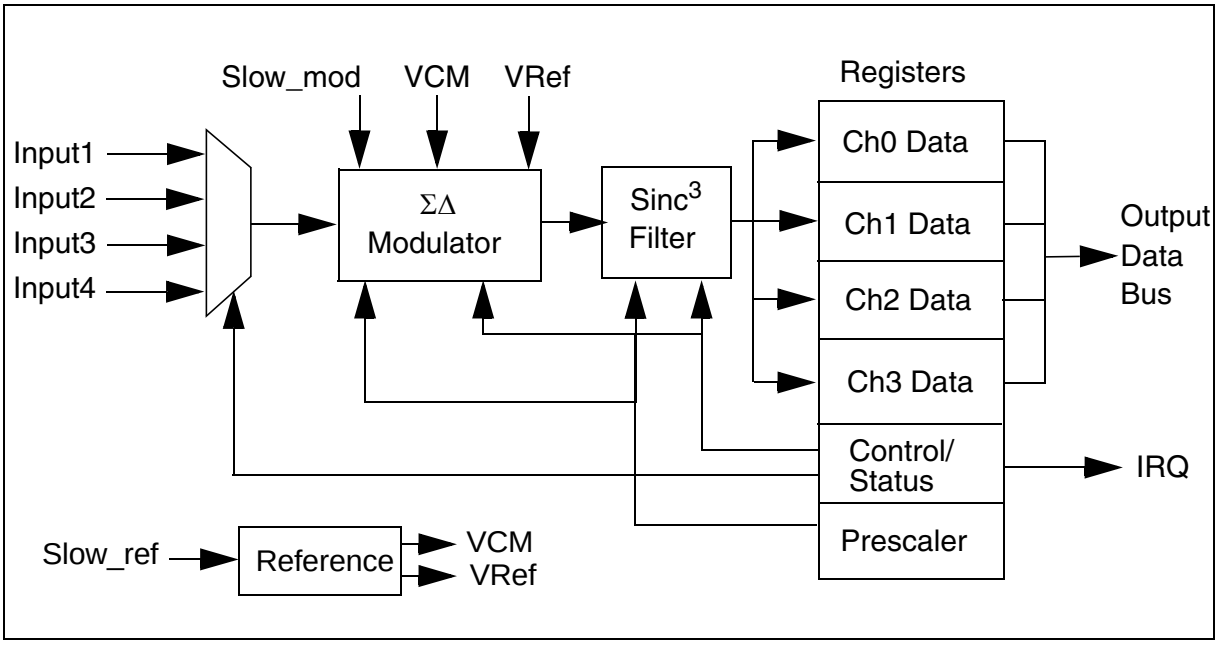
22.2 Main Features

- 11.5 bits ENOB resolution (Effective Number Of Bits)
- 0 to 2.5V input range
- 4 input channels
- Oversampling Clock (F_{MOD}) 4 MHz max.
- Input Bandwidth 977 Hz max.
- Input Sampling Frequency (F_s) 1953 Hz max.
- Conversion time (Ts) = 1/ (F_{MOD}/512)

22.3 Functional Description

The ADC consists of a four-channel single-bit Sigma-Delta modulator with a 512-sample $Sinc^3$ digital filter and a bandgap voltage reference. A block diagram of the converter is shown below in Figure 56.

Figure 56. Sigma-Delta Block Diagram



22.3.1 Normal (Round-Robin) Operation of ADC

In its normal mode of operation, the converter samples each input channel for 512 cycles of the over-sampling clock. In the first clock cycle, the $\Sigma-\Delta$ modulator is reset and the digital filter cleared. The remaining 1-bit samples from the modulator are filtered by the $Sinc^3$ filter and a 16 bit output sample supplied to the relevant data register after 512 clock cycles, the period over which the $Sinc^3$ filter has filled up and settled down. The channel select will then switch to the next input channel, the reset will again be asserted on the first clock cycle, and the filter will again fill up over 512 cycles to produce a sample. This process will be repeated for each of the channels continually in a round-robin fashion.

22.3.2 Single-Channel Operation

When sampling a single channel, that channel alone will be selected as input to the analog signal to the sigma-delta modulator. The functionality of the converter will remain the same as above in that the converter will be reset every 512 cycles, once a valid sample is produced. However, to maintain the same output frequency of the converter, only one of these samples will be taken out of every four, thus a valid sample for the channel will be produced every 2048 oversampling clock cycles, as in normal operations.

22.3.3 Low-rate operating mode

In case the required sample rate is particularly low, a special decimation mode can be used to overcome the limitation of prescaler configurations. This mode is selected by setting **DEC** bit of **ADCSR** register. In this way the oversampling factor becomes 5120 instead of 512 and a valid sample for a given channel will be produced each 20480 oversampling clock cycles, instead of 2048.

22.3.4 Interrupt and DMA Requests

An active-high interrupt/DMA request flag will be set depending on the mode of operation.
In round-robin channel selection mode, the interrupt flag will be set when all data available (**DA**) flags in the control register are set, and all the interrupt enables (**IE**) are also set. In this way the interrupt/DMA request (depending on which function is currently enabled) will be activated when all channels have been converted and their values are stored in the corresponding data registers. The interrupt/DMA request is deasserted when all data registers are read, this being the typical response of a DMA controller, or when “0” is written in all **DA** bits of **ADCCSR** register, this last case being typical of an interrupt response routine. It must be noticed that in order to work properly in this configuration all interrupt enable bits need to be set.
In single channel mode, the interrupt flag will be set when the data available flag and the interrupt flag for the selected channel are set. The interrupt request is deasserted by reading the data register associated to the selected channel (DMA response) or writing “0” in the corresponding **DA** bit of **ADCCSR** register (interrupt response). In order to have a proper single-channel interrupt/DMA request, besides selecting the single-channel mode and configuring which channel is to be converted using AXT and A[1:0] bits in ADCCSR register, the corresponding interrupt enable bit needs to be set as well.

22.3.5 Clock Timing

The sigma-delta modulator must run at a clock frequency (F_{Mod}, oversampling rate) not greater than 4 MHz. The registers in this converter are clocked by the APB clock. Double clocked synchronization for data crossing clock boundaries avoids any metastability issue. It is up to the user to correctly program the prescaler, to generate the correct oversampling frequency based on the APB frequency:

$$Presc = \frac{fAPB}{Fs \cdot 512 \cdot 8}$$

where:
Presc = Value written in prescaler register (**ADCCPR**)
fAPB = APB bus clock frequency
Fs = sampling frequency

In case the low-rate operation mode is selected, the formula above becomes:

$$Presc = \frac{fAPB}{Fs \cdot 5120 \cdot 8}$$

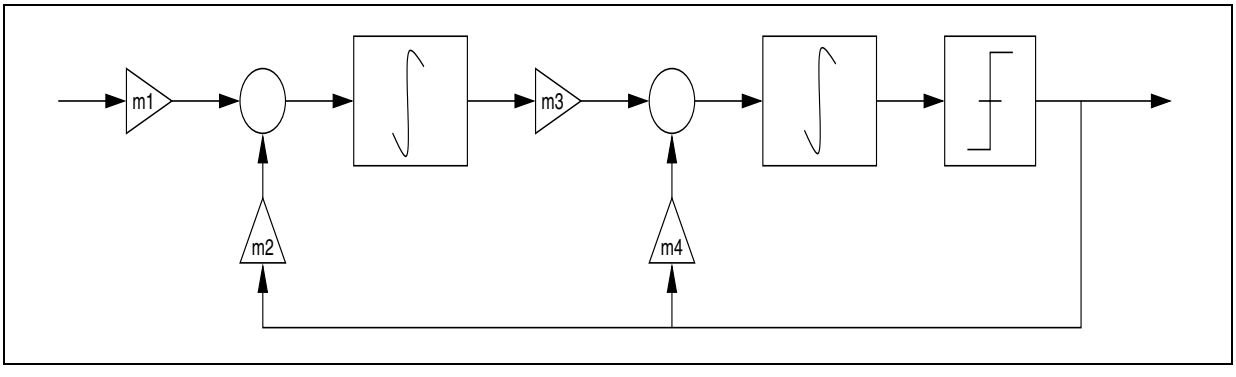
As an example, if APB clock frequency is 33 MHz and required sampling frequency is 100 Hz, **ADCCPR** register should be configured by writing 80 into it using the normal decimation mode. The same prescaler value, using low-rate decimation mode, would produce a sampling rate of 10 Hz.

Note *If the prescaler is set to generate a sampling frequency greater than specified, conversion performance is not guaranteed.*

22.3.6 Σ–Δ Modulator

The Σ–Δ modulator used is a single-bit design, second-order feedback architecture modulator including two integrators, two summing junctions and a comparator. The gains m1–m4 are set by capacitor ratios in the integrators.

Figure 57. Modulator architecture



22.3.7 The Sinc^3 Decimation Filter

The single-bit bitstream in output from the Σ–Δ modulator is fed to a Sinc^3 digital filter which filters the modulator bitstream output and decimates the sample rate to F_s. The Sinc^3 Z-domain transfer function $H(z)$ is:

$$H(z) = \frac{1}{N^3} \times \left[\frac{1 - z^{-N}}{1 - z^{-1}}\right]^3$$

with N = 171 in the case of STR720 device. $H(z)$ can be also conveniently represented as the product of three functions of the kind:

$$H(z) = H_1 \times H_2 \times H_3$$

where $H_n(z)$ {n=0,1, 2} is given by:

$$H_n(z) = \frac{1}{N} \times \left[\frac{1 - z^{-N}}{1 - z^{-1}}\right] = \frac{1}{N} \times \sum_{n=0}^{N-1} z^{-n} = (1 + z^{-1} + z^{-2} + \ldots + z^{-(N-1)})$$

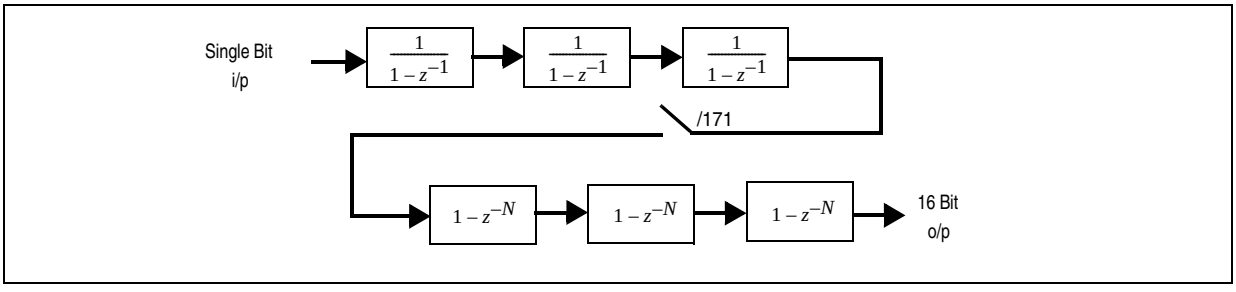
$H(z)$ is therefore the product of three low-pass filters $H_n(z)$ of 171 taps each and with $N = 171$ zeros evenly spread from 0 to F_{Mod} (modulator sampling frequency) in the frequency domain. Since, in the discrete time domain, the impulse response $h(n)$ inverse z-transform of $H(z)$ is given by the convolution:

$$h(n) = (h_1 \otimes h_2 \otimes h_3)$$

the total number of taps of $h(n)$ is $3N - 2 = 511$.

The normal implementation of this filter is shown below in Figure 3. The transfer function is decomposed into IIR and FIR sections — the IIR section (consisting of 3 integrators) runs at the full oversampling clock rate while the FIR section (with 3 differentiators) runs at the decimated rate.

Figure 58. Sinc^3 Digital Filter



The frequency response of the Sinc^3 filter is shown in Figure 59 and Figure 60 below.

Figure 59. Sinc^3 Frequency Response

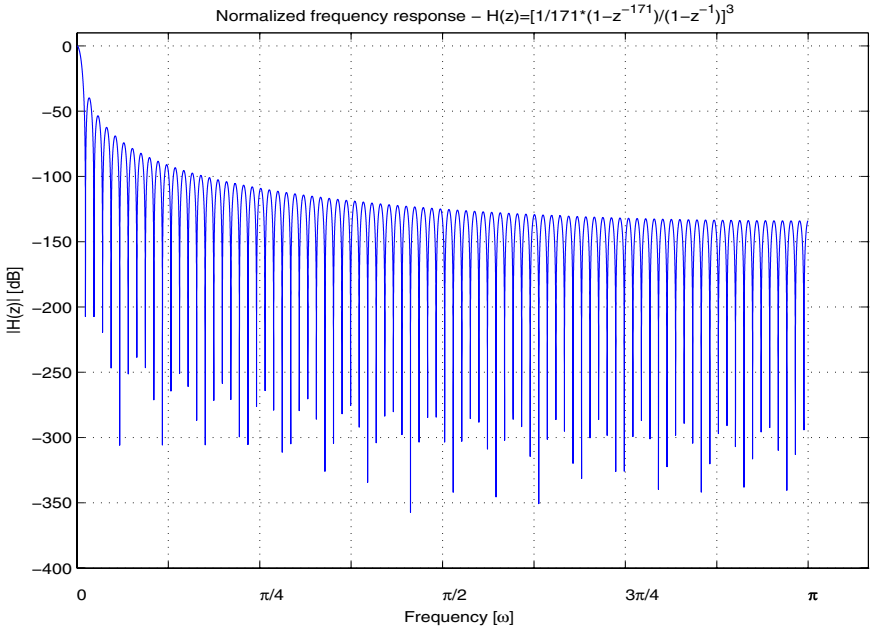
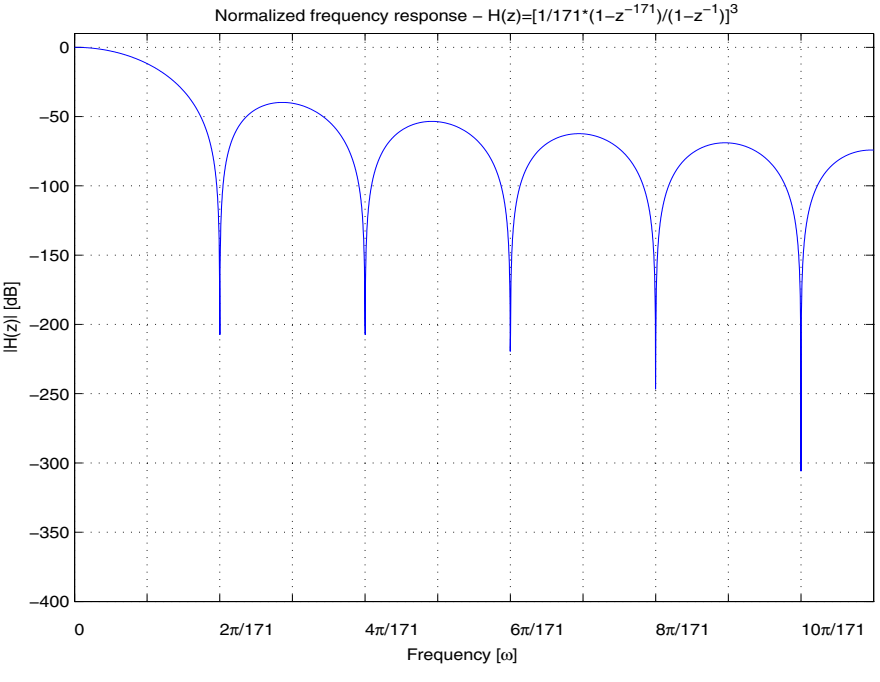


Figure 60. Sinc^3 Frequency Response - Detail



The word lengths needed in each integrator and differentiator are given by the formula *b=3log2(N) + 1*. This gives a maximum word length of 21 bits for the internal registers. As the output is only 16 bits, the LSBs are discarded.

Note Only the 12 most significant bits out of this 16 bits are guaranteed to be accurate according to the device specification, even if all 16 bits can be read.

Since the first zero of the decimation filter is located at

$$\omega = \frac{2\pi}{171}$$

while the first replica of the spectrum of the modulator after decimation is located at:

$$\omega = \frac{2\pi}{2048}$$

care must be taken to ensure that the Nyquist criterion is respected to avoid alias, i.e.:

$$F_s \geq F_{Nyq} = 2F_{Max}$$

The description above refers to the operations in normal decimation mode. When low-rate decimation mode is selected, the registers in the decimated section will be updated when the count gets to 1700 instead of 170 and the world length of the registers will be 33 bits instead of 23, the final value containing bits 33:18 of the final differentiator. The reset of the filter will occur every 5120 cycles instead of every 512.

22.3.8 Bandgap Reference

An on-chip bandgap reference generates a 1.22 V reference. This is used to generate two voltages used by the modulator — V_{CM} & V_{REF}. V_{CM} is designed to be 1.25 V, the midpoint of the converter's voltage range and V_{REF} is the feedback reference, 1.74 V. As the bandgap reference is not trimmed, absolute values of V_{CM} & V_{REF} could be inaccurate by up to± 5%. This will lead to gain and offset errors in the converter which can be calibrated out digitally if necessary.

To calibrate the converter it is necessary to input the minimum and maximum inputs supported — 0 V and 2.5 V. The digital output for a 0 V input is the offset and the gain of the converter is given by:

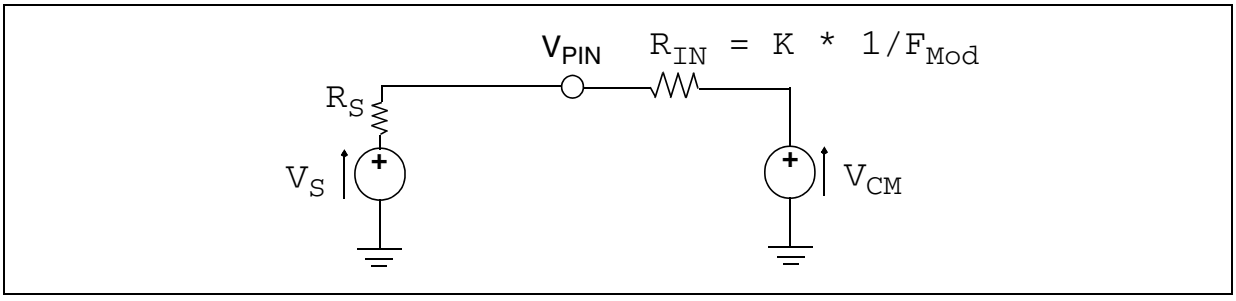
$$G = \frac{(2.5Voutput - 0Voutput)}{2.5}$$

The offset and gain correction factors can be stored digitally and applied to all outputs from the converter, for all four input channels.

22.3.9 ADC Input Equivalent Circuit

The input equivalent circuit, due to the switching at F_{Mod} rate of the input capacitance where the charge taken from the input signal to be measured is stored, can be represented as follows:

Figure 61. ADC Input Equivalent Circuit



where V_S is the voltage under measurement, R_S is the output resistance of the source, V_{PIN} the voltage the will actually converted and R_{IN} the input equivalent resistance of the ADC. R_{IN} is inversely proportional to the modulator oversampling clock and the constant K is equal to 540 [kΩ][MHz] +/- 20%.

22.3.10 ADC Output Coding

The Σ–Δ converter produces a digital sample of each analog input channel every 512 oversampling clocks (5120 when low-rate mode is selected). The digital samples in output from the Sinc^3 digital filter are stored in the four **ADCDATAn** registers as 16-bit samples of which only the first 12 most significant bits are meaningful. The converted value stored in ADCDATA[n] is a signed two's complement value and proportional to the difference (V_{IN}-V_{CM}), being ideally 0 if the input voltage were $V_{IN} = V_{CM}$. Since the gain and offset errors previously described, before actually using the result of the conversion calibration must be performed.

Note *The analog input voltage should not exceed twice the Center Voltage of the Σ-Δ Modulator (2 * V_{CM}) otherwise converter performances cannot be guaranteed. Remember that the V_{CM} Voltage has an accuracy of +/- 5% which imposes a calibration of the converter.*

22.3.11 Power Saving Features

The analog circuitry of the ADC block is switched off when bit “ADC_OFF” in the **AGCR** register is reset (see Section 24.2: *A-GCR Block description on page 351*), allowing the power consumption via AV_{DD} / AV_{SS} pins to be minimized (< 1 μA). Due to the fact that the analog section of ADC is used also to generate reference currents for the proper operation of clock input pads, it is recommended to disable the ADC by software only after having switched to a low power mode that is not using clock input pads, as SLOW or STOP mode (see Section 25: *POWER REDUCTION MODES on page 364*).

The digital section of ADC block can be stopped by using the ordinary clock gating features, provided on the device.

Note that on power-up, or after switching on “ADC_OFF” bit, the common-mode feedback circuit can take about 30 oversampling clock cycles to drive the common-mode voltage to its correct value. For this reason, the first sample should be discarded after activating ADC block.

22.4 Register description

ADC Control/Status Register (ADCCSR)

Address Offset: 20h

Reset Value: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DEC	OR	Res	IE[3:0]				Res.	AXT	A[1:0]		DA[3:0]			
-	w	rc	-	rw				-	rw	rw		rc			

This register controls the operating mode of the ADC, sets the interrupt enables, contains status flags for the availability of data and error flags in the event of data being overwritten before being read.

Bit 15 = Reserved. This bit should be written as ‘0’ and will be read always at ‘0’.

Bit 14 = **DEC**: *DECimation factor*
This bit selects the decimation mode between normal mode and low-rate mode, where same prescaler value can be used to reach very low sampling rates.
0: Normal mode, decimation by 512.
1: Low-rate mode decimation by 5120.

Bit 13 = **OR**: *OverRun*
This read-clear bit is used to notify application software that data on one of the channels has been overwritten before being read.
0: Normal operation. No overrun has occurred.
1: Overrun event occurred. This bit is set by hardware as soon as an overrun condition is detected and must be cleared by software by explicitly writing it to “0”. Writing “1” into this bit has no effect.

Bit 12 = Reserved. This bit should be written as ‘0’ and will be read always at ‘0’.

Bits 11:8 = **IE[3:0]**: *Interrupt Enable*
This set of bits allows to enable interrupt requests independently for each of the ADC channels, where bit IE[n] corresponds to ADC channel n.
0: Channel n interrupt disabled.
1: Channel n interrupt enabled.

Bit 7 = Reserved. This bit should be written as ‘0’ and will be read always at ‘0’.

Bit 6 = **AXT**: *Addressing eXTernal enable*
This bit allows to enable the single-channel operation, configuring the ADC to convert repeatedly the channel identified by A[1:0] bits of this register.
0: Round-robin addressing enabled.
1: Single-channel addressing enabled.

Bits 5:4 = **A[1:0]**: *channel Address*
These bits select the external channel to be sampled when external addressing is enabled.

Bits 3:0 = **DA[3:0]**: *Data Available*
This read-clear set of bits allows to determine on which channel data register a new sample is ready to be read, where bit DA[n] corresponds to ADC channel n. They are set by hardware as soon as a new sample on the corresponding channel is available and they can be cleared explicitly by writing them to “0” or indirectly when the corresponding data register is read. Writing “1” into these bits has no effect.
0: No sample is available on the corresponding channel.
1: New sample available on the corresponding channel.
These bits also act as interrupt flags for the corresponding channels.

ADC Clock Prescaler Register (ADCCPR)

Address Offset: 30h

Reset Value: 0006h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved									PRESC[6:0]						
-									w						

Bits 15:7 = Reserved. These bits should be always written as ‘0’.

Bits 6:0 = **PRESC[6:0]**: *Prescaler value*
The 7 bit binary value specified on the clock prescaler register determines the factor by which the ADC input clock will be divided down in order to produce the oversampling clock of the sigma-delta modulator, the actual factor being twice the PRESC register value as illustrated in Table 70 on page 341. The value placed in this register must subsequently generate an oversampling clock frequency not greater than 4 MHz from the master clock applied to the ADC. These bits can only be written by software, any read operation on them returns 0h.

Table 70. ADC Prescaler setting

Setting	Divide Factor
0	Not available (ADC frozen)
1	Not available (ADC frozen)
2	4
3	6
4	8
..	..
126	252
127	254

ADC Data Register n, n = 0 .. 3 (ADCDATAn)

Address Offsets: 00h, 08h, 10h, 18h

Reset Values: 0000h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r															

Four data registers, one for each of the analogue input channels, are available. The 12 most significant bits will contain the result of the conversion, while the least significant bits of each register should be ignored. Values reported in data registers are expressed in 2's complement notation. The data registers will be filled in numerical sequence in the round-robin channel mode. In single channel mode, only the selected channel will be updated.

Bit 15:0 = **DATA[15:0]:** *DATA sample*
This read-only register contains the last sampled value on the corresponding channel.

22.4.1 Register map

Table 71. ADC Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00	**ADCDATA0**	DATA[15:0]															
08	**ADCDATA1**	DATA[15:0]															
10	**ADCDATA2**	DATA[15:0]															
18	**ADCDATA3**	DATA[15:0]															
20	**ADCCSR**	Reserved		OR	Res.	IE[3:0]				Res.	AXT	A[1:0]		DA[3:0]			
28	**Reserved**																
30	**ADCCPR**	Reserved								PRESC[7:0]							

Refer to Table 20 on page 49 for the base address.

23 GENERAL PURPOSE I/O PORTS

23.1 Introduction

The General Purpose IO Port are programmable by software in several conditions: input, output, Alternate Function, open drain, push-pull, weak push-pull and high impedance. Each Port is configured in Input (PC0=0, PC1=1, PC2=0, see Table 72 on page 344) during the Reset phase.

Warning: being each GPIO pin configured in Input during Reset phase, the IO pins are released to High Impedance condition. To avoid power consumption the user has to drive the IOs to stable levels.

23.2 Main Features

- Data Input /Output
- Alternate Function
- CMOS Input
- PUSH-PULL Output
- Open Drain Output
- Weak PUSH-PULL Output
- Full software programmability

23.3 Functional Description

The General Purpose IO Port has three configuration registers (PC0,PC1,PC2) and one IO Data register (PD).

All the allowed Port configurations, set by programming of the configuration registers are summarized in Table 72 on page 344, where the index (n) is a generic IO bit.

A writing access to the IO Data register always loads the data in the Output Latch.The Output Latch holds the data to be sent out while the Input Latch captures the data present on the IO pin.

A reading access to the IO Data register can read the Input Latch or the Output Latch according to the Port configuration (see later).

The Figure 62 on page 344 shows the basic structure of the General Purpose IO Port bit.

Figure 62. Basic Structure of an I/O Port Bit

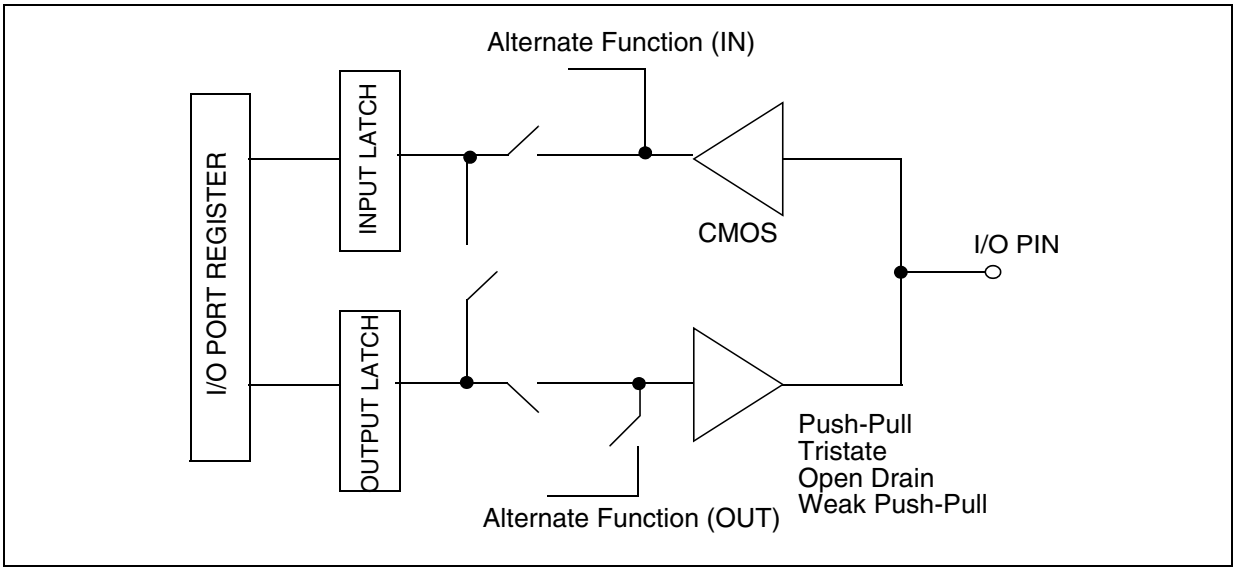


Table 72. Port Bit Configuration Table

PC0(n)	0	1	0	1	0	1	0	1
PC1(n)	0	0	1	1	0	0	1	1
PC2(n)	0	0	0	0	1	1	1	1
P(n) Configuration	RE-SERVED	IN		INOUT	OUT	OUT	AF	AF
P(n) Output	TRI	TRI		WP	OD	PP	OD	PP
P(n) Input	-	CMOS		CMOS	CMOS	CMOS	CMOS	CMOS

Notes:

Hi: High impedance
AIN: Analog Input
IN: Input
OU: Output
INOUT: Bidirectional
AF: Alternate Function

OD: Open Drain
PP: Push-Pull
WP: Weak Push-Pull
TRI: Tristate
CMOS: CMOS Standard Input(*)

(*) refer to the device electrical characteristics

The configuration PC0(n)-PC1(n)-PC2(n)=000 is Reserved and the user should not use it. In this configuration the correspondent pin enters the High Impedance state.

The configuration PC0(n)PC1(n)PC2(n)=100 and 010 configure the correspondent pin in the same state: Input, Tristate, CMOS.

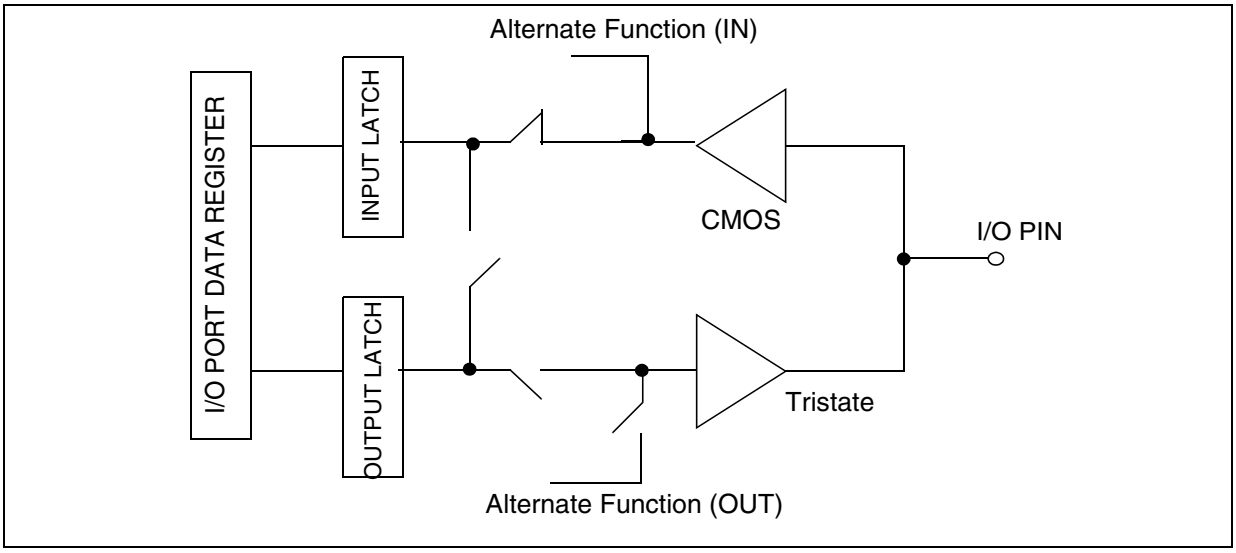
23.3.1 Input Configuration

When the IO Port is programmed as Input:

- The Output Buffer is forced tristate
- The data present on the IO pin is sampled into the Input Latch every clock cycle
- A reading access to the Data register gets the value in the Input Latch.

The Figure 63 on page 345 shows the Input Configuration of the IO Port bit.

Figure 63. Input Configuration



23.3.2 Bidirectional Configuration

When the IO Port is programmed as Bidirectional:

- The Output Buffer is turned on in Weak Pull configuration (Port2)
- The Output Buffer is turned on in Weak Push configuration (Port3, Port4)
- A reading access to the IO Data register gets the Input Latch value.

The Figure 64 on page 346 and Figure 65 on page 346 show the Bidirectional Configuration of the IO Port.

Figure 64. Bidirectional Configuration (Port2)

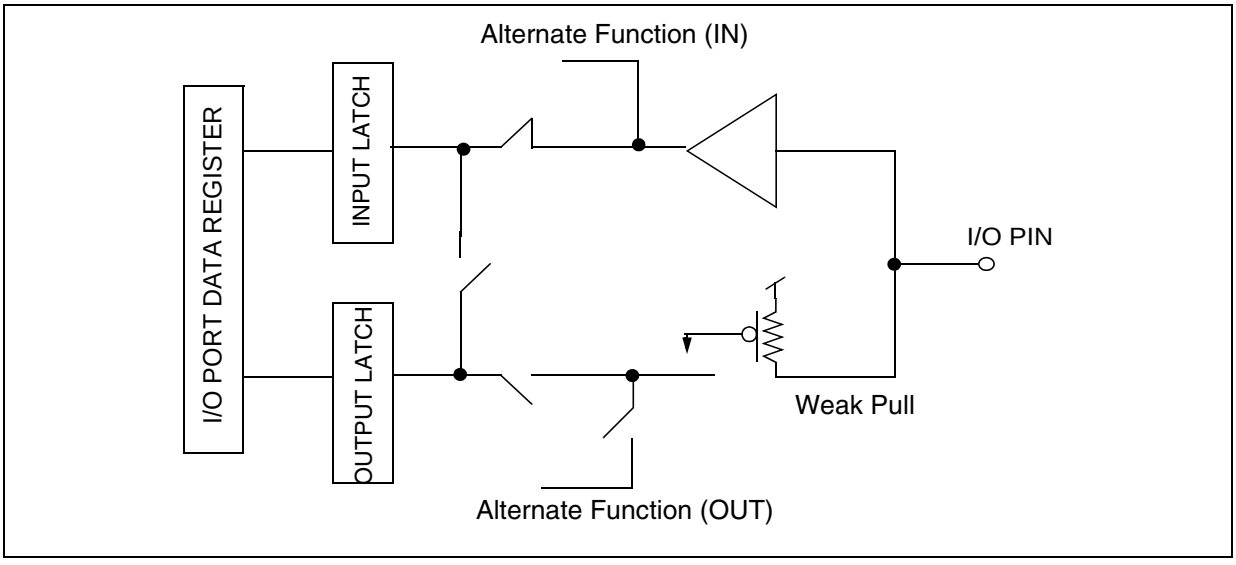
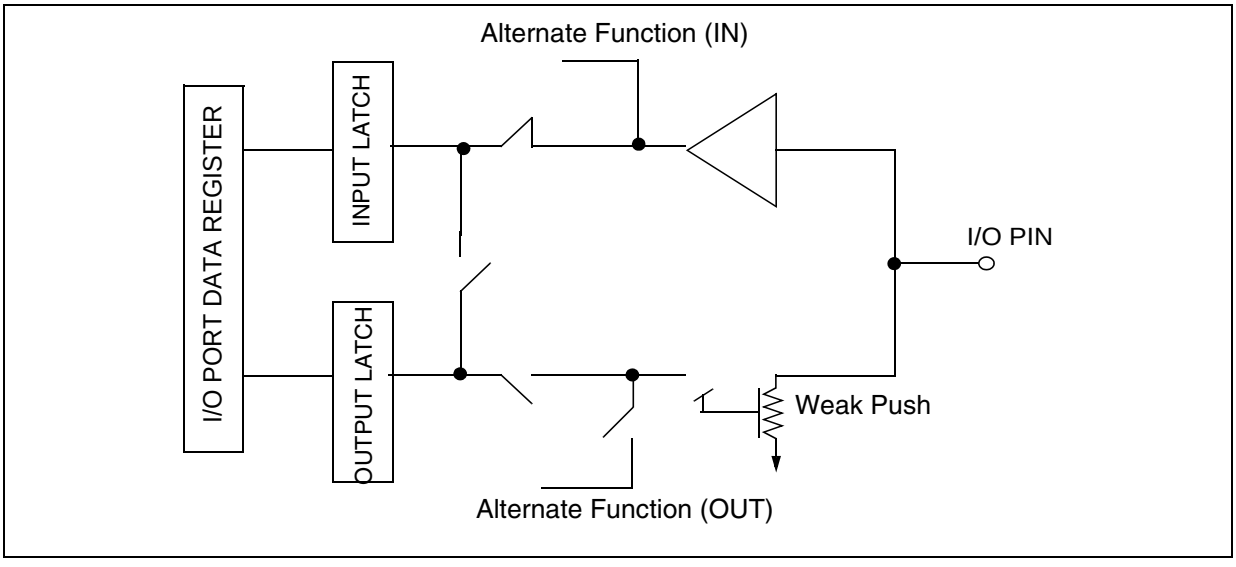


Figure 65. Bidirectional Configuration (Port3, Port4)



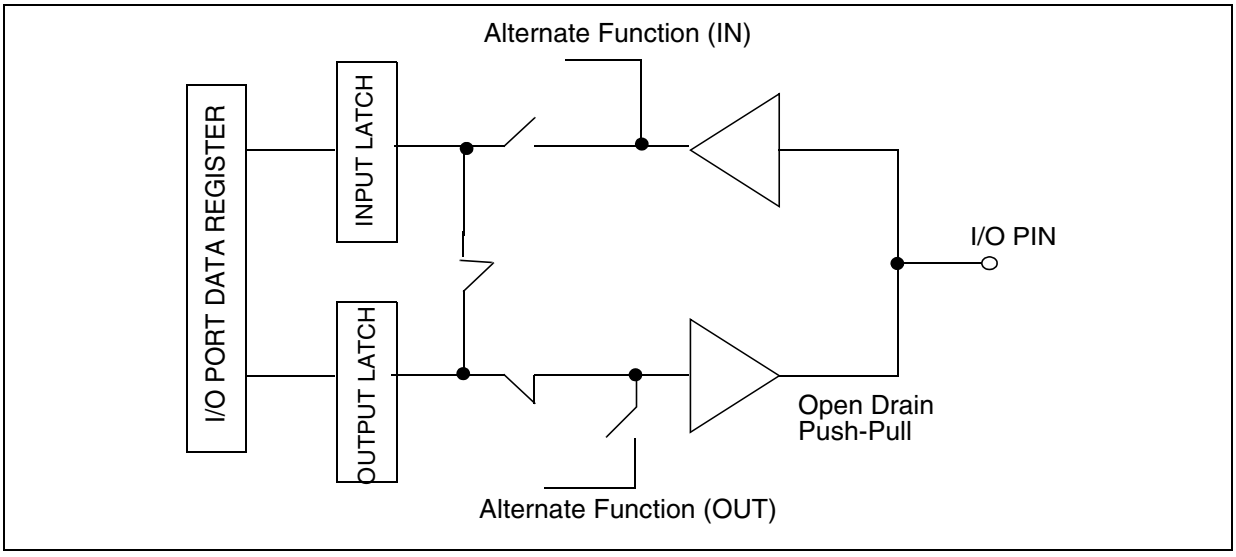
23.3.3 Output Configuration

When the IO Port is programmed as Output:

- The Output Buffer is turned on in Open Drain or Push-Pull configuration
- The data in the Output Latch drives the IO pin
- A reading access to the IO Data register gets the Output Latch value.

The Figure 66 on page 347 shows the Output Configuration of the IO Port bit.

Figure 66. Output Configuration



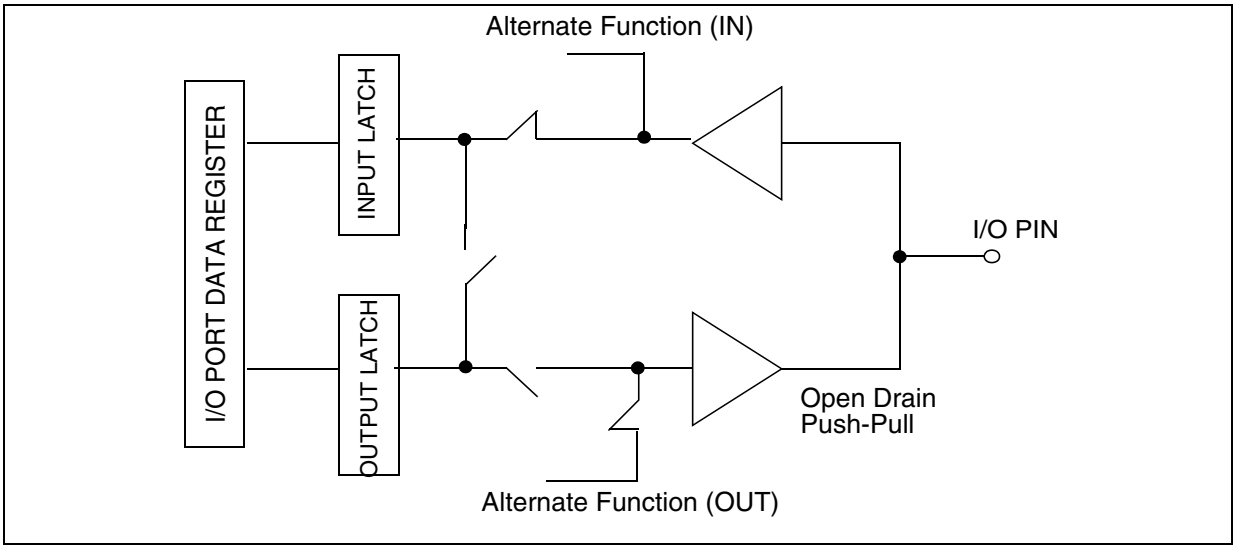
23.3.4 Alternate Function Configuration

When the IO Port is programmed as Alternate Function:

- The Output Buffer is turned on in Open Drain or Push-Pull configuration
- The Output Buffer is driven by the signal coming from the peripheral (alternate function out)
- The data present on the IO pin is sampled into the Input Latch every clock cycle
- A reading access to the Data register gets the value in the Input Latch.

The Figure 67 on page 348 shows the Alternate Function Configuration of the IO Port bit.

Figure 67. Alternate Function Configuration



23.4 Register description

The IOPORT registers can not be accessed by byte.

Port Configuration Register0 (PC0)

Address Offset: 00h

Reset value: (*)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C015	C014	C013	C012	C011	C010	C09	C08	C07	C06	C05	C04	C03	C02	C01	C00
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:0 = **C0[15:0]:** *Port Configuration bits*
See Table 72 on page 344 to configure the IO Port.

Port Configuration Register1 (PC1)

Address Offset: 04h

Reset value: (*)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C115	C114	C113	C112	C111	C110	C19	C18	C17	C16	C15	C14	C13	C12	C11	C10
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:0 = **C1[15:0]:** *Port Configuration bits*
See Table 72 on page 344 to configure the IO Port.

Port Configuration Register2 (PC2)

Address Offset: 08h

Reset value: (*)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C215	C214	C213	C212	C211	C210	C29	C28	C27	C26	C25	C24	C23	C22	C21	C20
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:0 = **C2[15:0]:** *Port Configuration bits*
See Table 72 on page 344 to configure the IO Port.

IO Data Register (PD)

Address Offset: 0Ch

Reset value: (*)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 15:0 = **D[15:0]:** *IO Data bits*
A writing access to this register always writes the data in the Output Latch.
A reading access reads the data from the Input Latch in Input and Alternate function configurations or from the Output Latch in Output and High impedance configurations.

(*) Refer to the device IO pin list

23.4.1 Register map

The following table summarizes the registers implemented in the IO port macrocell.

Table 73. IO-port Register Map

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	**PC0**	C0[15:0]															
4	**PC1**	C1[15:0]															
8	**PC2**	C2[15:0]															
C	**PD**	D[15:0]															

Refer to Table 20 on page 49 and Table 21 on page 50 for the base addresses.

24 MISCELLANEA REGISTERS (GCR - CGC - AHB_ERR)

24.1 Introduction

This section collects the registers setting all the aspects of STR720 which can be configured by software. Hereafter the specification of A-GCR, S-GCR and CGC blocks can be found, allowing application software to select between different functional options, to check the status of relevant internal configuration signals and to control clock and reset lines of most of internal peripherals. At the end of this section, the specification of the AHB error detection block is reported.

24.2 A-GCR Block description

This block contains the configuration settings related to the peripherals connected to the A-APB subsystem.

A-APB Global Configuration Register 1 (AGCR1)

Address: 0xE000_1000

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved													ADC_ STOP	Res.	ADC_ OFF
-													rw	-	rw

Bit 15:3 = Reserved. These bits must be always written to 0.

Bit 2 = **ADC_STOP**: *ADC STOP mode*
This bit can temporarily stop sampling operation of the Analog to Digital Converter block without affecting the programming of its configuration registers.
0: ADC is operating normally.
1: ADC operations are stopped.

Bit 1 = Reserved. This bit must be always written to 0.

Bit 0 = **ADC_OFF**: *ADC analog section OFF*
This bit controls the activation of the whole analog section contained in the ADC module, where also internal current references used for clock pad are generated. When this bit is set to '1' all the analog circuitry in the ADC module is switched off.
0: ADC analog section is switched on.
1: ADC analog section is switched off.

Note *Be careful on setting ADC_OFF bit since it switches off the current references used by clock pad and in this condition no clock signal can be supplied to the device through CLK pad. If this configuration is required for extreme power reduction purposes, STR720 must be put in SLOW mode before setting ADC_OFF bit, otherwise the system will stop responding up to the next hardware reset event.*

24.3 S-GCR Block description

This block contains the configuration settings related to the peripherals connected to the S-APB subsystem and to the whole device.

S-APB Global Configuration Register 1 (SGCR1)

Address: 0xF000_0C00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										CACHE_CONF	Reserved		EIC_SRES	P7AS	BRM
-										rw	-		rw	rw	rw

This register collects all software controllable configuration bits.
Bit 15:6 = Reserved. These bits must be always written to 0.

Bit 5 = **CACHE_CONF**: *CACHE CONFiguration selection*
This bit can configure SDRAM interface to work in “burst-access” mode and must be used whenever cache is enabled, so to have optimal performance from the system.
0: Normal access mode, to be used when ARM720T cache is disabled.
1: Burst-access mode, to be used when ARM720T cache is enabled.

Note *For further details and limitation of “burst-access” mode usage see* Section 10.5: *Programming considerations on page 114, in DRAM controller chapter.*

Bit 4:3 = Reserved. These bits must be always written to 0.

Bit 2 = **EIC_SRES**: *EIC Soft RESet*
This bit triggers a synchronous reset of the internal sequential logic of Enhanced Interrupt Controller, leaving priority registers unaffected. It can be useful to recover some tricky situations occurring inside the EIC state machines without requiring full reconfiguration of EIC registers (see Section 7.6: *Application note on page 67*).
0: EIC is operating normally.
1: EIC logic is kept under synchronous reset.

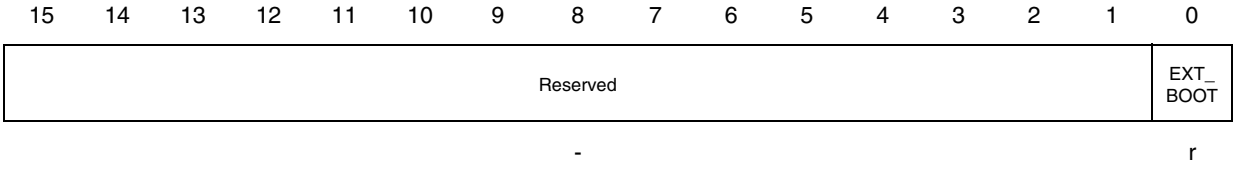
Bit 1 = **P7AS**: *Port7 Access Selection*
This bit selects which interface is using port 7 pins which can be shared between EMI and IDE, for example to access external FLASH banks during system boot and then switch to CD-ROM access to complete system start-up.
0: EMI interface uses port 7 pins.
1: IDE interface uses port 7 pins.

Bit 0 = **BRM**: *Boot ReMap*
This bit selects which memory area is mapped in the address range from 0x0000_0000 to 0x1FFF_FFFF (Block 0) and it is used to conclude the boot phase when internal RAM has been loaded with its required contents
0: Reserved ROM or EMI area is mapped on Block 0 (see EXT_BOOT bit in SGCR2 register).
1: PROG_RAM area is mapped on Block 0.

S-APB Global Configuration Register 2 (SGCR2)

Address: 0xF000_0C04

Reset value: 0x00XX



This register collects internal configuration signals status so that application software can check them, if required. This is a read-only register.

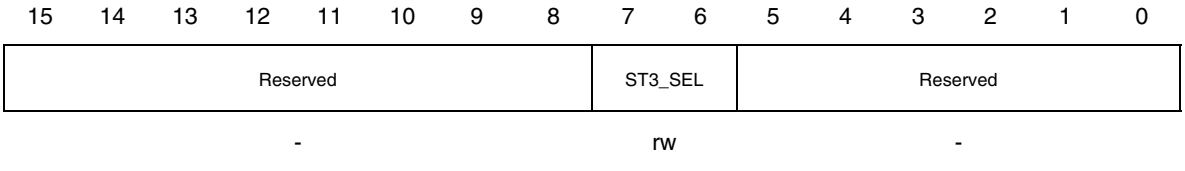
Bit 15:1 = Reserved. These bits will be always read back as ‘0’.

Bit 0 = **EXT_BOOT**: *EXTernal BOOT*
This bit reports the status of P3.8 pin at $\overline{\text{RSTIN}}$ rising edge, which is used to configure the boot mode of the device, selecting if reset vector is fetched from EMI address space. After reset rising edge P3.8 can change its value, according to application requirements, but boot mode cannot be changed unless a new hardware reset is applied to the device.
0: Reserved ROM mode.
1: STR720 boots from external EMI area.

S-APB Global Configuration Register 3 (SGCR3)

Address: 0xF000_0C08

Reset value: 0x0000



This register configures the DMA request assignment to the available DMA streams.

Bit 15:8 = Reserved. These bits must be always written to 0.

Bit 7:6 = **ST3_SEL[1:0]:** *DMA STream3 SELection*
This bit field selects which DMA-capable peripheral is associated to DMA Stream 3 request:
00: BSPI1 transmit DMA request
01: ADC sample ready request.
10: Reserved. This value should never be used.
11: Reserved. This value should never be used.

Bit 5:0= Reserved. These bits must be always written to 0.

24.4 CGC Block description

The CGC block enables to configure the AHB and S-APB peripheral clock and reset lines so as to individually control each of them. The specific clock signals used by IDE and USB can be controlled using dedicated registers implemented in these cell. A set 16-bit registers is used to keep the clock gating and reset line state configuration for each of the controlled peripherals.

CGC Peripheral Clock Gating register 1 (CGC-PCG1)

Address: 0xF000_2C00

Reset value: 0x0023

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				P11CO	P10CO	Reserved		P7CO	P6CO	P5CO	Res.	P3CO	P2CO	P1CO	P0CO
-				rw	rw	-		rw	rw	rw	-	rw	rw	rw	rw

This register configures clock gating for each of the controlled AHB/S-APB peripherals. Each peripheral can be used only when its corresponding gating bit is cleared.

Bits 15:12, 9:8, 4 = Reserved. These bits must be always written to 0.

Bits 11:10, 7:5, 3:0 = **PxCO:** *Peripheral x Clock Off*
0: Peripheral x clock is gated off and the block is not clocked.
1: Peripheral x is clocked.

The following table lists the correspondence between each controllable peripheral and the CGC-PCG1 register bits along with their reset value.

Table 74. CGC-PCG1 bit assignment

CGC-PCG1 Bit	Affected peripheral	Reset status
P0CO	EMI	On
P1CO	DRAMC	On
P2CO	DMAC	Off
P3CO	ATAPI	Off
P5CO	EIC	On
P6CO	GPIO-P3	Off
P7CO	GPIO-P4	Off
P10CO	WIU	Off
P11CO	RTC	Off

CGC Peripheral Under Reset register 1 (CGC-PUR1)

Address: 0xF000_2C04

Reset value: 0x0023

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				P11UR	P10UR	Reserved		P7UR	P6UR	P5UR	REs.	P3UR	P2UR	P1UR	P0UR
-				rw	rw	-		rw	rw	rw	-	rw	rw	rw	rw

This register configures the reset line status for each of the controlled AHB/S-APB peripherals. Each peripheral can be used only when its corresponding reset line is not active.

Bits 15:12, 9:8, 4 = Reserved. These bits must be always written to 0.

Bits 11:10, 7:5, 3:0 = **PxUR:** *Peripheral x Under Reset*
0: Peripheral x is under reset.
1: Peripheral x is operating normally.

The following table lists the correspondence between each controllable peripheral and the CGC-PUR1 register bits along with their reset value.

Table 75. CGC-PUR1 bit assignment

CGC-PUR1 Bit	Affected peripheral	Reset status
P0UR	EMI	Operative
P1UR	DRAMC	Operative
P2UR	DMAC	Under reset
P3UR	ATAPI	Under reset
P5UR	EIC	Operative
P6UR	GPIO-P3	Under reset
P7UR	GPIO-P4	Under reset
P10UR	WIU	Under reset
P11UR	RTC	Under reset

CGC Emulation register 1 (CGC-EMU1)

Address: 0xF000_2C08

Reset value: 0x0FFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				P11EM	P10EM	Reserved		P7EM	P6EM	P5EM	Res.	P3EM	P2EM	P1EM	P0EM
-				rw	rw	-		rw	rw	rw	-	rw	rw	rw	rw

This register configures the status of clock gating for each of the controlled AHB/S-APB peripherals while the application program execution is stopped due to a debug request. Each peripheral can be configured to stop working as the user code reaches a breakpoint.

Bits 15:12, 9:8, 4 = Reserved. These bits must be always written to 0.

Bits 11:10, 7:5, 3:0 = **PxEM:** *Peripheral x clock during EMulation*
0: Peripheral x clock is gated off at a debug request, regardless from the state of the corresponding PxCO bit.
1: Peripheral x clock gating status is unaffected by debug requests, and it is always determined by its corresponding PxCO bit.

The following table lists the correspondence between each controllable peripheral and the CGC-EMU1 register bits along with their reset value.

Table 76. CGC-EMU1 bit assignment

CGC-EMU 1 Bit	Affected peripheral	Reset status
P0EM	EMI	Not affected by debug requests.
P1EM	DRAMC	Not affected by debug requests.
P2EM	DMAC	Not affected by debug requests.
P3EM	ATAPI	Not affected by debug requests.
P5EM	EIC	Not affected by debug requests.
P6EM	GPIO-P3	Not affected by debug requests.
P7EM	GPIO-P4	Not affected by debug requests.
P10EM	WIU	Not affected by debug requests.
P11EM	RTC	Not affected by debug requests.

CGC Peripheral Clock Gating register 2 (CGC-PCG2)

Address: 0xF000_2C0C

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved													D2CO	D1CO	Res.
-													rw	rw	-

This register configures clock gating for each specific clock signal used by some blocks requiring dedicated clocks for their operations. Each dedicated clock signal is delivered to the related peripheral only when its corresponding gating bit is cleared.

Bits 15:3 = Reserved. These bits must be always written to 0.

Bits 2:1 = **DxCO:** *Dedicated peripheral x Clock Off*
0: Peripheral x dedicated clock is gated off and the block kernel is not clocked.
1: Peripheral x dedicated clock is delivered.

Bit 0 = Reserved. This bit must be always written to 0.

The following table lists the correspondence between each dedicated peripheral clock and the CGC-PCG2 register bits along with their reset value.

Table 77. CGC-PCG2 bit assignment

CGC-PCG2 Bit	Affected peripheral	Reset status
D1CO	ATAPI-Kernel	Off
D2CO	USB-Kernel	Off

CGC Peripheral Under Reset register 2 (CGC-PUR2)

Address: 0xF000_2C10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved														D1UR	Res.
-														rw	-

This register configures the reset line status for each of the peripheral kernels requiring a dedicated clock. Each peripheral can be used only when its corresponding reset line is not active.

Bit 15:2 = Reserved. These bits must be always written to 0.

Bit 1 = **D1UR:** *Dedicated peripheral 1 Under Reset*
0: Peripheral 1 is under reset.
1: Peripheral 1 is operating normally.

Bit 0 = Reserved. This bit must be always written to 0.

The following table lists the correspondence between each controlled peripheral and the CGC-PUR2 register bits along with their reset value.

Table 78. CGC-PUR2 bit assignment

CGC-PUR2 Bit	Affected peripheral	Reset status
D1UR	ATAPI-Kernel	Under reset

CGC Emulation register 2 (CGC-EMU2)

Address: 0xF000_2C14

Reset value: 0x001F

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved													D2EM	D1EM	Res.
-													rw	rw	-

This register configures the status of clock gating for each of the peripheral kernel requiring a dedicated clock, while the application program execution is stopped due to a debug request. Each peripheral can be configured to stop working as the user code reaches a breakpoint.

Bits 15:3 = Reserved. These bits must be always written to 0.

Bits 2:1 = **DxEM:** *Dedicated peripheral x clock during EMulation*
0: Peripheral x dedicated clock is gated off at a debug request, regardless from the state of the corresponding DxCO bit.
1: Peripheral x dedicated clock gating status is unaffected by debug requests, and it is always determined by its corresponding DxCO bit.

Bit 0 = Reserved. This bit must be always written to 0.

The following table lists the correspondence between each controllable peripheral and the CGC-EMU2 register bits along with their reset value..

Table 79. CGC-EMU2 bit assignment

CGC-EMU2 Bit	Affected peripheral	Reset status
D1EM	ATAPI-Kernel	Not affected by debug requests.
D2EM	USB-Kernel	Not affected by debug requests.

24.5 AHB Error detection block description

Due to the specific implementation of ARM720T AHB interface, any AHB transfer resulting in an error response from the addressed slave, is not detected directly by the core. Detecting these events is particularly helpful during application software debug, when due to an incorrect configuration some inconsistent memory access can be attempted. In order to detected these events a specific block is implemented, which monitors AHB activity and generates a FIQ request to the core, storing the value of the address which caused the error response to be triggered. Here are the few registers by which this block operation can be configured and used.

AHB Error Pending (AERR_P)

Address: 0xF000_3000

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															AE_P
-															rc

This register stores the pending bit which is set by hardware at the occurrence of an AHB error. The register is read-clear in the sense that application software can only write ‘0’ into this bit, to acknowledge the detected error and re-enable AHB bus monitoring. Writing ‘1’ into this register has no effect.

Bits 31:1 = Reserved. These bits must be always written to 0.

Bit 0 = **AE_P:** *AHB Error Pending*
0: No AHB error detected.
1: An AHB transfer resulted into an error response.

AHB Error Mask (AERR_M)

Address: 0xF000_3004

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
-															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															AE_M
-															rw

This register allows application software to mask the FIQ interrupt generation upon the detection of any AHB error, when such feature is not required. The block will still detect faulty transfers on the system bus, just the interrupt generation is disabled. This register is read-write.

Bits 31:1 = Reserved. These bits must be always written to 0.

Bit 0 = **AE_M:** *AHB Error Mask*
0: AHB error interrupts masked.
1: AHB error interrupts enabled.

AHB Error Faulty Address (AERR_FA)

Address: 0xF000_3008

Reset value: 0x0000_0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AERR_FA[31:0]															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AERR_FA[31:0]															
r															

This register stores the address of the location whose access caused the AHB error response to be generated. When an AHB error is detected, the address of the faulty location is saved into this register and the AE_P bit in AERR_P register is set, causing a FIQ interrupt request to be issued if AE_M bit in AERR_M register is set as well. As long as the AE_P bit is not clear by application software, this register will keep the stored address even if further AHB error response occur. This is a read-only register.

Bits 31:0 = **AERR_FA:** *AHB Error Faulty Address*

24.5.1 Register map

In the following tables a summary of all system configuration register and AHB error detection block is reported.

Table 80. Configuration Register Map

Addr.	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E000 1000	**AGCR1**	Reserved													ADC_ STOP	Res.	ADC_ OFF
F000 0C00	**SGCR1**	Reserved										CACHE _CONF	Reserved		EIC_ SRES	P7AS	BRM
F000 0C04	**SGCR2**	Reserved															EXT_ BOOT
F000 0C08	**SGCR3**	Reserved								ST3_SEL		Reserved					
F000 2C00	**CGC-PGC1**	Reserved				P11CO	P10CO	Reserved		P7CO	P6CO	P5CO	Res.	P3CO	P2CO	P1CO	P0CO
F000 2C04	**CGC-PUR1**	Reserved				P11UR	P10UR	Reserved		P7UR	P6UR	P5UR	Res.	P3UR	P2UR	P1UR	P0UR
F000 2C08	**CGC-EMU1**	Reserved				P11EM	P10EM	Reserved		P7EM	P6EM	P5EM	Res.	P3EM	P2EM	P1EM	P0EM
F000 2C0C	**CGC-PGC2**	Reserved													D2CO	D1CO	Res.
F000 2C10	**CGC-PUR2**	Reserved														D1UR	Res.
F000 2C14	**CGC-EMU2**	Reserved													D2EM	D1EM	Res.

Table 81. AHB Error detector Register Map

Addr.	Register Name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F000 3000	**AERR_P**	Reserved																															AE _P
F000 3004	**AERR_M**	Reserved																															AE _M
F000 3008	**AERR_FA**	AERR_FA[31:0]																															

Refer also to Table 20 on page 49 and Table 21 on page 50.

25 POWER REDUCTION MODES

The device contains a centralized clock management block (RCCU) that provides the controlling software with a means to dynamically disable/enable the clocks to all the major blocks in the design. The power consumption of the device may thus be tailored to the required mode of operation.

STR720 operating modes are completely under software control via the proper configuration of the RCCU registers. This chapter describes each of them in detail.

25.1 RUN Mode

In this operating mode STR720 can be used at its maximum performance level, since main system clock can be programmed to reach the highest recommended frequency, either using directly CLK input or using the on-chip PLL. In the case in which on-chip PLL is used, the following constraints must be taken into account.

In the first place it must be considered that the range of frequencies in which PLL is able to lock goes from 9 MHz to 27 MHz. When CLK input is driven with a signal whose frequency is above 27 MHz, DIV2 multiplexer must be always enabled for the on-chip PLL to operate correctly.

Besides this, another constraint comes from the maximum limit of the internal PLL VCO frequency, which is required to be less than 1080 MHz. The value of VCO oscillation frequency depends on the multiplication/division factors set in RCCU configuration registers (M and D respectively) and the frequency of reference clock (CLK or CLK divided by 2 if DIV2 is enabled). The following formulas describe this relationship.

$$f_{VCO} = 2 \cdot f_{REF} \cdot M \qquad f_{OUT} = \frac{f_{VCO}}{2 \cdot D}$$

Depending on PLL configuration. the possible system frequencies generated in RUN mode can be summarized by the following table, calculated for a generic 16 MHz clock signal supplied to CLK input. In composing the table, PLL configuration leading to the smaller current consumption have been chosen.

Table 82. RUN mode system frequencies (CLK=16 MHz)

PLL Multiplier / Divider	AHB / S-APB [MHz]	A-APB min (f/16)	A-APB max (f/2)	ATAPI
PLL_OFF/BYP	16.00	1.00	8.00	8.00
PLL_BYP	16.00	1.00	8.00	8.00
10 / 4	40.00	2.50	20.00	20.00
21 / 8	42.00	2.63	21.00	21.00
11 / 4	44.00	2.75	22.00	22.00
23 / 8	46.00	2.88	23.00	23.00
...	...	...	...	...

Table 82. RUN mode system frequencies (CLK=16 MHz)

PLL Multiplier / Divider	AHB / S-APB [MHz]	A-APB min (f/16)	A-APB max (f/2)	ATAPI
29 / 8	58.00	3.63	29.00	29.00
15 / 4	60.00	3.75	30.00	30.00
31 / 8	62.00	3.88	31.00	31.00
8 / 2	64.00	4.00	32.00	32.00
33 / 8	66.00	4.13	33.00	33.00
17 / 4	68.00	4.25	34.00	22.67
35 / 8	70.00	4.38	35.00	23.33

After reset the RCCU is configured in RUN mode with PLL switched off, PLL_BYP enabled and DIV2 disabled, so that the main system frequency at start-up is equal to CLK input. To select the actual operating system frequency, the DIV2 multiplexer should be enabled first if the input clock frequency requires it, then PLL should be switched on after having programmed its multiplying factor. The signal generated by PLL can be selected as clock source disabling the PLL_BYP multiplexer but this operation should be performed only when PLL is stable (about 200 µs). A LOCK status bit can be used to detect PLL status and switch to its clock only when lock condition is reached. Any change of PLL lock condition is registered in dedicated interrupt pending flags. If application requires to take immediate actions when PLL lock condition is lost or achieved, an automatic PLL bypass can be enforced each time the lock condition changes, switching to reference clock as soon as lock condition is lost and switching back to PLL clock when lock condition is regained. This mechanism can be enabled by setting the AUTOBYP_EN flag.

The PLL should be switched-off before changing the multiply factor, using the reference clock during this transitory phase.

For more detail about PLL management see Section 13.3.2: *PLL Management on page 140*.

During RUN mode, most of STR720 peripherals clocks can be independently switched on and off using PCG registers inside A-APB bridge, CGC block registers and a few SGCR1 register bits. In this way only the peripherals whose functionalities are required by the system need to be clocked, so the actual power consumption of the whole STR720 device can be adapted by software, according to application requirements and the exact current consumption is heavily dependent on which blocks are active and for how long. At system reset all of controlled peripherals are kept under reset with their clock signals switched off, except EIC, EMI and DRAMC which are required for system start-up. It is up to application software to enable the required peripherals. The following table summarizes the peripherals whose clock and reset lines can be controlled:

Table 83. Peripheral clock and reset gating

Block	Clock & reset gating	Reset status	Block	Clock & reset gating	Reset status
ARM720	*Not Available*	On	DMAC	CGC: P2CO, P2UR	Off
			Data RAM	*Not Available*	On
IDE block	CGC: D1CO, D1UR	Off	IDE interface	CGC: P3CO, P3UR	Off
SDRAMC	CGC: P1CO, P1UR	On	EMI	CGC: P0CO, P0UR	On
S-APB bridge	*Not Available*	On	A-APB bridge	*Not Available*	On
AGCR	*Not Available*	On	SGCR	*Not Available*	On
RCCU	*Not Available*	On	CGC	*Not Available*	On
WIU	CGC: P10CO, P10UR	Off	EIC	CGC: P5CO, P5UR	On
GPIO - P3	CGC: P6CO, P6UR	Off	GPIO - P4	CGC: P7CO, P7UR	Off
USB	A-APB: P11CO, P11UR CGC: D2CO	Off	UART1	A-APB: P5CO, P5UR	Off
CAN	A-APB: P10CO, P10UR	Off	UART2	A-APB: P6CO, P6UR	Off
EFT1	A-APB: P7CO, P7UR	Off	BSPI1	A-APB: P3CO, P3UR	Off
EFT2	A-APB: P8CO, P8UR	Off	BSPI2	A-APB: P4CO, P4UR	Off
WDG	*Not Available*	On	GPIO - P2	A-APB: P2CO, P2UR	Off
ADC	A-APB: P7CO, P7UR	Off	ADC analog	AGCR1: ADC_MDON	Off
RTC-APB	CGC: P11CO, P11UR	Off			

25.2 IDLE Mode

In this operating mode, main system clock is derived from CLK input dividing it by 32. In this way the whole system can still evolve, although at a very low rate, but power consumption will be greatly reduced. Any setting for A-APB and ATAPI clock generation remains unchanged even if proper operation of the peripheral subsystem cannot be maintained due to the reduced operating frequency.

Application software can switch off any peripheral not required in this operating mode as much as disabling the PLL (see Section 13.3.2: *PLL Management on page 140*) and any other analog part whose functionality is not required. If for example an external clock of 16 MHz is applied to CLK input, activating this operating mode results in a system frequency (AHB/S-APB clock) being either 250 kHz or 500 kHz, depending if DIV2 multiplexer is enabled or not.

No particular sequence must be used to enter or exit IDLE mode since interrupt detection is always active; only the external SDRAM banks should be put in self-refresh mode before entering IDLE mode, otherwise their contents will be lost due to the very long refresh interval. Once in self-refresh mode, SDRAMC clock can be switched off safely.

25.3 SLOW Mode

In this operating mode, main system clock is taken directly from OSCIN pad, where the 32 kHz oscillator used to supply RTC is connected. In this way system operations are virtually frozen due to the extremely low rate system frequency.

As it happens in IDLE mode, any setting for A-APB and ATAPI clocks is not affected by entering in SLOW mode, so application software is responsible for switching off any part of the system which is not strictly required, especially the analog parts: PLL, ADC and also clock input receivers since main clock signal detection is not required in this mode, being based on the 32 kHz oscillator.

Note *Before entering SLOW mode the A-APB clock frequency should be set to be AHB clock divided by 2 if the watchdog is configured to use the external low frequency clock source due to internal synchronization requirements of this clock signal which is derived from the same 32 kHz oscillator.*

It must be noticed that when main clock is switched off also at the application board level, particular care must be observed when generating internal reset events (watchdog or software triggered) since the absence of that external clock prevents the correct synchronization of the internal reset signal deassertion. As a consequence when main external clock is switched off on the board, watchdog should be disabled or used as a plain timer and no software reset should be issued, since in both cases the exit from these reset events would not be properly synchronized. In case a software reset is required, main board clock should be restarted before generating it.

No other particular care must be used to enter or exit SLOW mode, although the same consideration mentioned in IDLE mode section about external SDRAM banks and DRAMC should be applied as well.

25.4 STOP Mode

When it is required that system configuration is preserved without feeding any clock to the system, the STOP mode functionality can be used, where only RTC section remains clocked and the rest of the system is completely frozen, thus reducing its consumption to the leakage current only. Upon entering STOP mode, a specific sequence must be executed by application software in order to leave the system in a state where consumption is minimum but it is still possible to wake the system up without resetting it and thus altering previous data and configuration. The code to enter STOP mode should be written according to the following rough sequence:

1. Switch off operation of all analog sections.
2. Enter SLOW mode. As a consequence system clock is now 32 kHz.
3. Switch off PLL to remove its power consumption (see Section 13.3.2: *PLL Management on page 140*).
4. Disable STOP mode, clearing bit STOP_EN in RCCU MSKCTL register, so that system remains alive even after WIU block rises STOP mode request.
5. Mask wake-up request from CLK clock line, clearing bit 7 in WIU WUMR register. This is to be sure that the CLK clock, which is still active at this time, cannot exit immediately from STOP mode.
6. Issue the STOP sequence described in WIU section (See “WAKE-UP INTERRUPT UNIT (WIU)” on page 70.). After this sequence is completed, the STOP_REQ signal is activated so RCCU will be ready to stop main clock signal. External devices can be notified that STR720 is about to enter STOP mode by using the alternate function associated to P3.7 pad, so that they can react accordingly, for example switching off CLK clock source.
7. If application does not require to use the activity on CLK clock line as wake-up event, STOP mode could be entered now, jumping to step 9.
   Otherwise it is necessary to poll bit 7 in WIU WUPR register, each time clearing the bit, until it is read as ‘0’, meaning that no more activity is present on CLK clock line.
8. The wake-up request corresponding to CLK clock line can now be unmasked, setting to ‘1’ bit 7 in WIU WUMR register.
9. STOP mode can now be enabled, setting bit STOP_EN in RCCU MSKCTL register. From now on main system clock is frozen.

Note *The change of PLL lock condition, associated to interrupt/wake-up line number 6, **cannot** be used as wake-up event. Selecting this input as the only wake-up source will stuck the system so as only a reset event can restore normal operations.*

WARNING: Whenever a STOP request is issued to the system, a few clock cycles are needed to actually enter STOP mode. Hence the execution of the instruction following the setting of STOP_EN bit in RCCU might start before entering STOP mode (consider the ARM7 three-stage pipeline as well). In order to avoid to execute any valid instruction after a correct STOP sequence and before entering the STOP mode, it is mandatory to execute a dummy set of few instructions after the setting of STOP_EN bit. **In particular at least six dummy instructions** (e.g. *MOV R1, R1*) shall be added after the end of STOP sequence. Again, if

exiting from STOP mode an interrupt routine shall be serviced, another set of dummy instructions shall be added, to take into account of the latency period: this is evaluated in at **least other three dummy instructions.** This to consider that when STOP mode entered, the pipeline content is frozen as well, and when the system restarts the first executed instruction was fetched and decoded before entering the STOP mode itself.

Exiting STOP mode can be triggered by two different sets of events: internal events generated on some STR720 peripheral pins (as UART reception line or USB wake-up signalling) or external events restoring CLK clock line. All these events can be detected by WIU which will deactivate the STOP_REQ signal as a response, thus removing the STOP condition from RCCU block. A guideline to write the code required to exit STOP mode can be the following:

1. Upon wake-up event, WIU removes STOP_REQ signal notifying to the external devices that STR720 is about to exit STOP mode. If the traced event is triggered by STR720 peripherals, this signal will be used by a possible external power manager to restore CLK clock otherwise it will have been CLK clock activity, restarted autonomously by the external power manager, which triggered STR720 wake-up.
2. STOP_REQ deactivation restarts RCCU operations, which awakes system clock in SLOW mode.
3. Application software should now determine which wake-up source triggered the exit from STOP mode, in order to take any consequent action. If CLK clock activity is not there (see point 7 of previous sequence), a system dependent time interval must be waited so to allow a stable CLK clock to be present on STR720 pads.
4. RCCU can now be switched back in RUN mode, the PLL enabled again and all the rest of system operation restored.

25.5 STANDBY Mode

The minimum level of power consumption can be reached using STANDBY mode, where not only main system clocks are stopped but also main power supply is removed from STR720 device, except the RTC section with its 32 kHz oscillator, so that time can be still traced while most of the system is kept under reset condition. This extreme power saving mode can be exited re-initializing the whole device using $\overline{\text{RSTIN}}$ pad, thus losing any previous state and configuration while the real time clock counter is not affected.

Any action required by the system to enter safely STANDBY mode has to be performed by application software, following a request arrived from an external power manager and acknowledging the enter in STANDBY mode after having saved any required state information into an external storage area (for example an external FLASH device). A pair of GPIO pins can be used to implement this protocol. As a response to STR720 entering in STANDBY mode, the external power manager should activate $\overline{\text{RSTIN}}$ pad and then remove V_{DD} supply, leaving only V_{DDRTC} active.

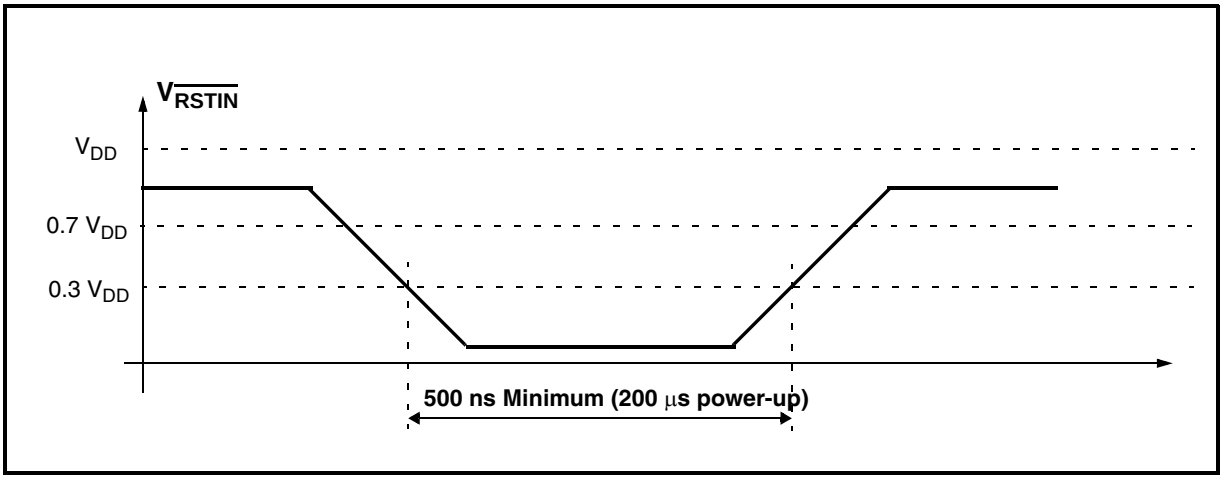
When the external power manager requires STR720 to exit STANDBY mode, it should restore V_{DD} supply to its nominal value always keeping $\overline{\text{RSTIN}}$ pad driven low so that a safe hardware reset sequence is triggered, and only after V_{DD} supply is stable to its nominal value, $\overline{\text{RSTIN}}$ pad should be raised again. STR720 will start a reset sequence, eventually restoring its previous state using the informations saved into the external storage area, and system operation can restart.

26 SYSTEM RESET

26.1 RESET Input Pin

This pin is used to force a system reset which initializes the device in a predefined state. Activating this pin does not affect in any way the operation of either Real-Time-Clock or on-chip emulation and debug sections, which are independently initialized by dedicated pins. To improve the noise immunity of the device, the RESET input pin ($\overline{RSTIN}$) has a Schmitt trigger input circuit with hysteresis and an on-chip RC filter is implemented. The filter is sized to filter out all spikes shorter than 50 ns. On the other hand, it is recommended to provide valid pulse on $\overline{RSTIN}$ with a duration of at least 500 ns to be sure that the asynchronous pulse is properly latched by system. This means that all the pulses whose length is between 50 ns and 500 ns can either be filtered or recognized as valid, depending on the operating conditions and process variations. As a recommendation, the power-up $\overline{RSTIN}$ pulse should last for about 200 µs in order to be sure that the internal current references used to supply the clock input pad are already settled to their nominal value.

Figure 68. Recommended Signal to be applied on $\overline{RSTIN}$ pin



26.2 RTCRST Input Pin

This pin is used to initialize Real-Time-Clock section independently from the rest of the system. To improve the noise immunity of this critical input pin, $\overline{RTCRST}$ has a Schmitt trigger input circuit with hysteresis and an on-chip RC filter is implemented, similar to the one described for RESET input pin. The filter is sized to filter out all spikes shorter than 50 ns and minimum $\overline{RTCRST}$ pulse duration is recommended to be longer than 500 ns. This means that all the pulses whose length is between 50 ns and 500 ns can either be filtered or recognized as valid, depending on the operating conditions and process variations. As a recommendation, the power-up $\overline{RTCRST}$ pulse should last for about 1 s in order to be sure that 32 kHz oscillator is has completed its start-up phase and settled to its nominal oscillation frequency and amplitude.

26.3 JTRST Input Pin

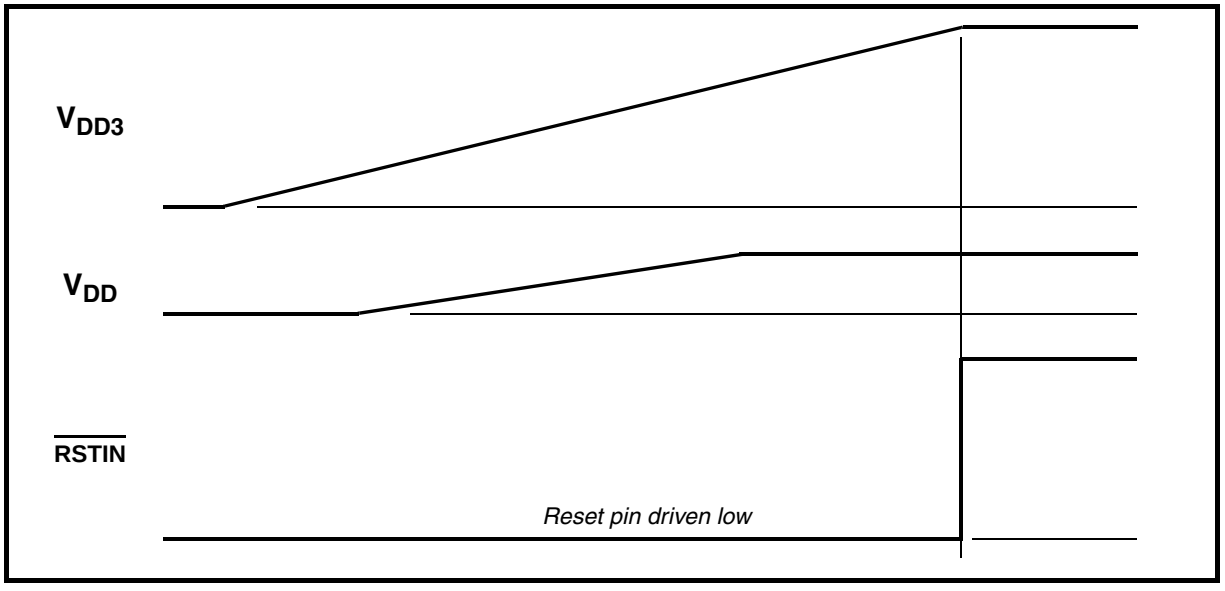
This pin is used to initialize on-chip emulation and debug logic independently from the rest of the system. To improve the noise immunity of this critical input pin, $\overline{\text{JTRST}}$ has a Schmitt trigger input circuit with hysteresis and an on-chip RC filter is implemented, similar to the one described for RESET input pin. The filter is sized to filter out all spikes shorter than 50 ns and minimum $\overline{\text{JTRST}}$ pulse duration is recommended to be longer than 500 ns. This means that all the pulses whose length is between 50 ns and 500 ns can either be filtered or recognized as valid, depending on the operating conditions and process variations.

26.4 Power-on/off and Stand-by entry/exit

At power-on or Stand-by exit, it is recommended to held low $\overline{\text{RSTIN}}$ pin to guarantee a proper initialization of STR720 system and avoid malfunctions of the RTC logic. In particular, when V_{DD} and V_{DD3} are still too low to allow the internal logic to work properly, the system must be mantained under reset status, forcing a HW Reset through the $\overline{\text{RSTIN}}$ pin. In this way the system sees a HW Reset and the flag register (CLK_FLAG in RCCU module) assumes the corresponding configuration (all Reset flags cleared).

To avoid unpredictable current injection and power consumption, it is recommended to have the V_{DD3} value always greater than V_{DD} during the power-on phase, as illustrated in next Figure 69 on page 371.

Figure 69. Power-on supply sequence

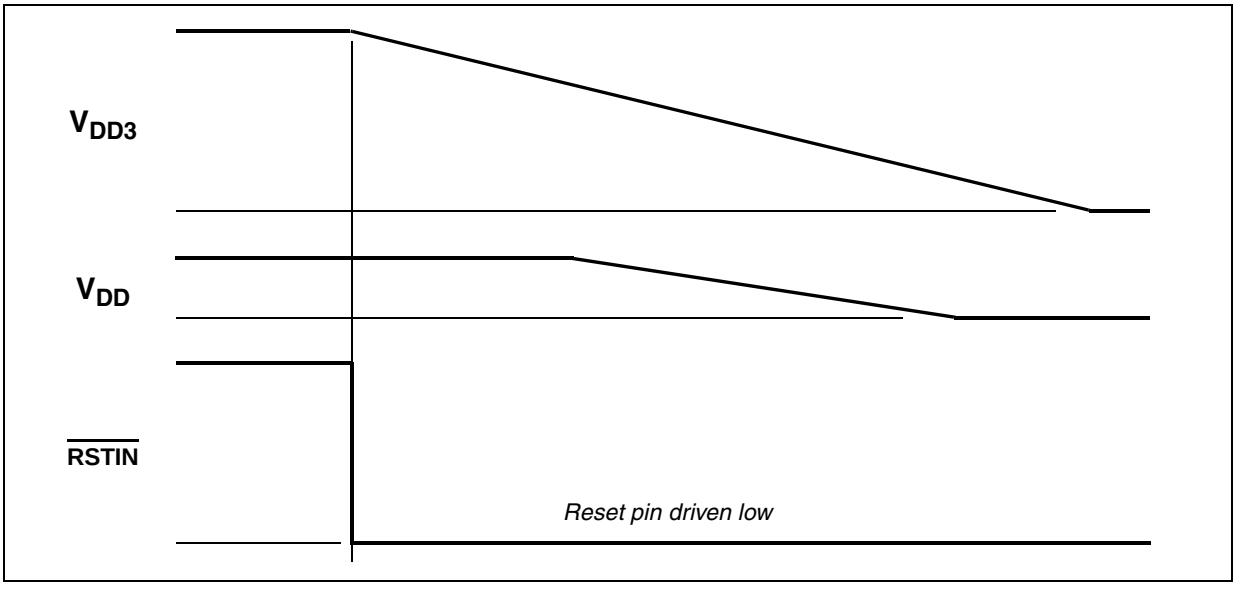


The distinction between power-on reset and hardware system reset consists in the fact that typical hardware reset sequences do not affect Real-Time-Clock section, leaving RTC running while rest of the system is restarted. This happens also when system enters and exit stand-by mode. On the contrary a power-on reset will see both system and RTC section being initialized. In order to distinguish between a power-on reset and a hardware system reset (or

exit from stand-by mode) a check on RTC Alarm registers can be done (see RTCA registers in Chapter 14: REAL TIME CLOCK (RTC) on page 150). These registers are reset to all-1 state, representing a time in the far future which is likely to be useless for any application using the alarm feature. Hence application software can use these registers as a flag to detect RTC reset events, thus distinguishing between a plain system reset (RTCA registers keep their value) and a power-on reset (RTCA registers are initialized).

Similarly during power-off sequence or Stand-by entry, $\overline{\text{RSTIN}}$ pin should be driven low before switching off power supply, as depicted in next Figure 70 on page 372.

Figure 70. Power-off supply sequence



27 PACKAGE MECHANICAL DATA

Figure 71. PQFP208 Package Outline

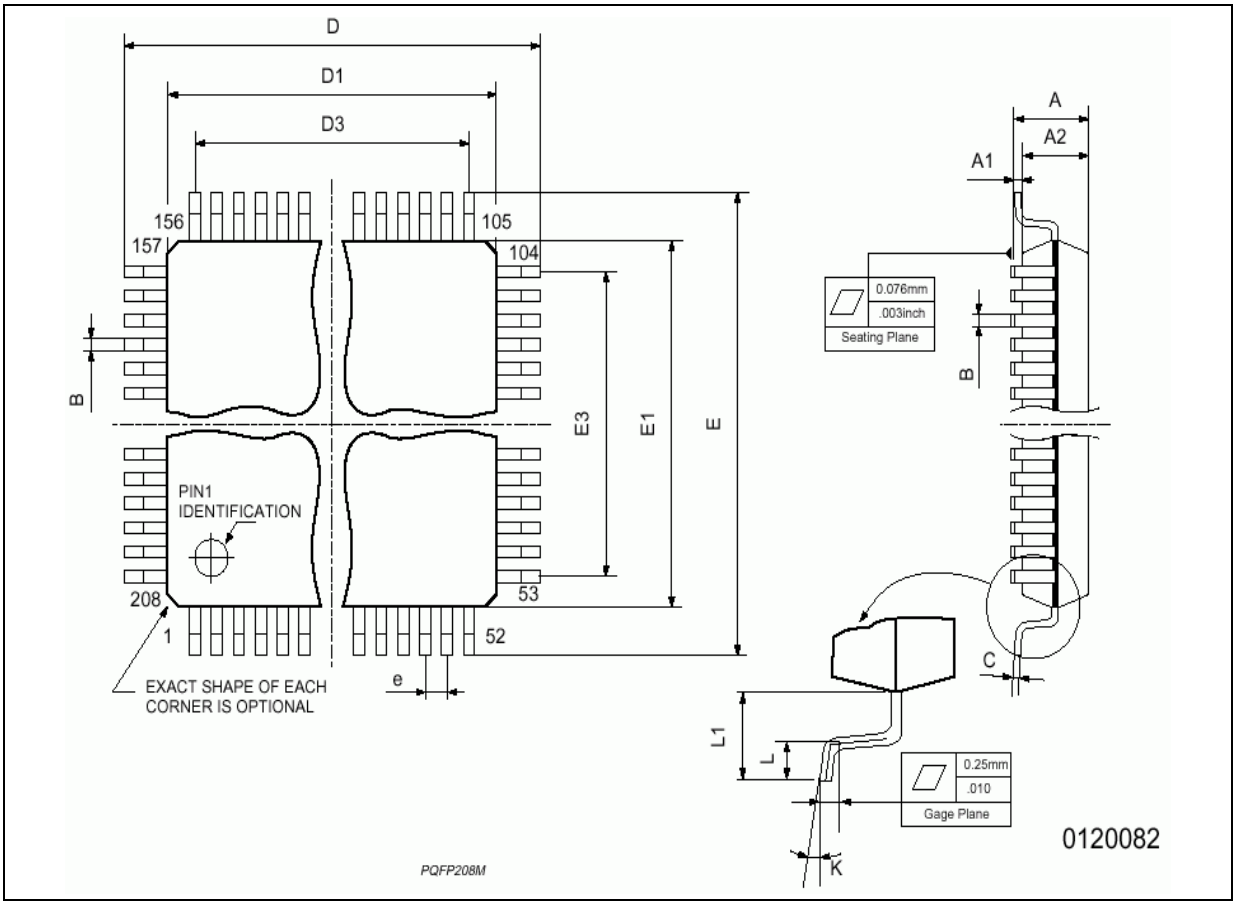


Table 84. PQFP208 Package Mechanical Data

Dim	mm			inches			Dim	mm			inches		
	min	typ	max	min	typ	max		min	typ	max	min	typ	max
A			4.10			0.161	**D3**		25.50			1.004	
A1	0.25			0.010			**e**		0.50			0.020	
A2	3.40	3.20	3.60	0.134	0.126	0.142	**E**		30.60			1.205	
B	0.17		0.27	0.007		0.011	**E1**		28.00			1.102	
C	0.09		0.20	0.003		0.008	**E3**		25.50			1.004	
D		30.60			1.205		**L**	0.45	0.60	0.75	0.018	0.024	0.029
D1		28.00			1.102		**L1**		1.30			0.051	
K	0°(min), 3.5°(typ), 7°(max)												

28 ELECTRICAL CHARACTERISTICS

28.1 Parameter Conditions

Unless otherwise specified, all voltages are referred to V_{SS}.

28.1.1 Minimum and Maximum values

Unless otherwise specified the minimum and maximum values are guaranteed in the worst conditions of ambient temperature, supply voltage and frequencies by tests in production on 100% of the devices with an ambient temperature at T_A = 25°C and T_A = 85°C.

Data based on characterization results, design simulation and/or technology characteristics are indicated in the table footnotes and are not tested in production. Based on characterization, the minimum and maximum values refer to sample tests and represent the mean value plus or minus three times the standard deviation (mean $\pm$ 3σ).

28.1.2 Typical values

Unless otherwise specified, typical data are based on T_A = 25°C, V_{DD3} = 3.3 V, V_{DD} = 1.8 V. They are given only as design guidelines and are not tested.

Typical ADC accuracy values are determined by characterization of a batch of samples from a standard diffusion lot over the full temperature range, where 95% of the devices have an error less than or equal to the value indicated (mean $\pm$ 2σ).

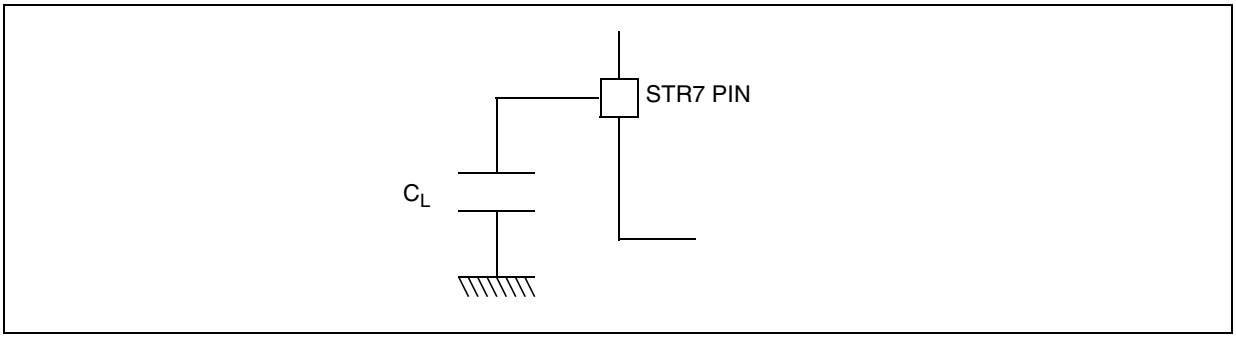
28.1.3 Typical curves

Unless otherwise specified, all typical curves are given only as design guidelines and are not tested.

28.1.4 Loading capacitor

The loading conditions used for pin parameter measurement are shown in Figure 72.

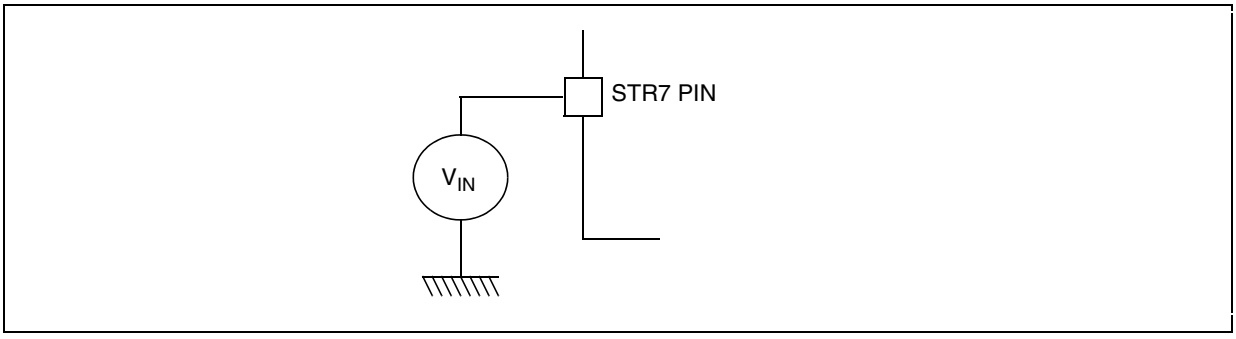
Figure 72. Pin loading conditions



28.1.5 Pin input voltage

The input voltage measurement on a pin of the device is described in Figure 73.

Figure 73. Pin input voltage



28.2 Absolute maximum ratings

This product contains devices to protect the inputs against damage due to high static voltages, however it is advisable to take normal precautions to avoid application of any voltage higher than the specified maximum rated voltages.

For proper operation it is recommended that V_{IN} and V_{OUT} be higher than V_{SS} and lower than V_{DD3}. Reliability is enhanced if unused inputs are connected to an appropriate logic voltage level (V_{DD}, V_{DD3} or V_{SS}).

Table 85. Voltage characteristics

Symbol	Parameter	Value		Unit
		Min	Max	
V_{DD3}	Voltage on V_{DD3} pins with respect to ground (V_{SS})	-0.3	+3.6	V
V_{DD}	Voltage on V_{DD} pins with respect to ground (V_{SS})	-0.3	+2.0	V
V_{DDRTC}	Voltage on V_{DDRTC} pin with respect to ground (V_{SS})	-0.3	+2.0	V
V_{DDPLL}	Voltage on V_{DDPLL} pin with respect to ground (V_{SS})	-0.3	+2.0	V
V_{SSPLL}	Voltage on V_{SSPLL} pin with respect to ground (V_{SS})	V_{SS}	V_{SS}	V
V_{DD3P}	Voltage on V_{DD3P} pin with respect to ground (V_{SS})	-0.3	+3.6	V
V_{SSP}	Voltage on V_{SSP} pin with respect to ground (V_{SS})	V_{SS}	V_{SS}	V
A_{VDD}	Voltage on A_{VDD} pin with respect to ground (V_{SS})	-0.3	+3.6	V
A_{GND}	Voltage on A_{GND} pin with respect to ground (V_{SS})	V_{SS}	V_{SS}	V
V_{IN}	Voltage on any pin with respect to ground (V_{SS})	-0.3	V_{DD3}+0.3	V
I_{OV}	Input current on any pin during overload condition	-10	+10	mA
I_{TOV}	Absolute sum of all input currents during overload condition		| 75 |	mA
T_{ST}	Storage temperature	-55	+150	°C
ESD	ESD Susceptibility (Human Body Model)		2000	V

Stresses exceeding above listed recommended “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the

device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability. During overload conditions ($V_{IN}>V_{DD}$ or $V_{IN}<V_{SS}$) the voltage on pins with respect to ground (V_{SS}) must not exceed the values obtained by the Absolute Maximum Ratings.

During Power-on and Power-off transients (including Standby entering/exiting phases), the relationships between voltages applied to the device and the main V_{DD}-V_{DD3} shall be always respected. In particular power-on and power-off of A_{VDD} shall be coherent with V_{DD} transient, in order to avoid undesired current injection through the on-chip protection diodes.

Table 86. Current characteristics

Symbol	Parameter	Value		Unit
		Min	Max	
I_{OV}	Input current on any pin during overload condition	-10	+10	mA
I_{TOV}	Absolute sum of all input currents during overload condition	-	| 75 |	mA

Table 87. Thermal characteristics

Symbol	Parameter	Value		Unit
		Min	Max	
T_J	Junction temperature under bias	–40	+105	°C
T_{ST}	Storage temperature	-55	+150	°C

28.3 Electrical sensitivity

Based on two different tests (ESD and LU) using specific measurement methods, the product is stressed in order to determine its performance in terms of electrical sensitivity.

28.3.1 Electro-Static Discharge (ESD)

Electro-Static Discharges (a positive then a negative pulse separated by 1 second) are applied to the pins of each sample according to each pin combination. Three models can be simulated: Human Body Model, Machine Model and Charge Device Model.

Table 88. ESD maximum ratings

Symbol	Ratings	Conditions	Max. value [1)]	Unit
VESD(HBM)	Electro-static discharge voltage (Human Body Model)	TA=+25°C	2000	V
VESD(MM)	Electro-static discharge voltage (Machine Model)	TA=+25°C	200	V
VESD(CDM)	Electro-static discharge voltage (Charge Device Model)	TA=+25°C	499	V

1. Data based on production characterization results, not tested in production.

28.3.2 Static Latch-Up (LU)

A supply overvoltage (applied to each power supply pin) and a current injection (applied to each input, output and configurable I/O pin) are performed on each sample. After the test DC parametric and functional testing is performed. This test conforms to the EIA/JESD 78 IC latch-up standard.

Table 89. Electrical sensitivity

Symbol	Parameter	Conditions	JEDEC Level
LU	Static latch-up class	TA=+25°C	A

28.4 Operating conditions

The following values are recommended to meet device functional specification.

Table 90. Recommended Operating Conditions

Symbol	Parameter	Value		Unit
		Min.	Max	
V_{DD3}	3.3V Operating Digital Supply Voltage for the I/O pads.	3.0	3.6	V
V_{DD}	1.8V Operating Digital Supply Voltage for core circuitry.	1.6	2.0	V
V_{DDRTC}	1.8V Operating Digital Supply Voltage for RTC logic.	1.6	V_{DD}	V
V_{DD3P}	3.3V Operating Supply Voltage for CLK input buffer. 1)	3.0	3.6	V
V_{DDPLL}	1.8 V Operating Supply Voltage for internal PLL. 2)	1.6	2.0	V
A_{VDD}	Operating Analog Supply Voltage for A/D Converter. 3)	V_{DD3} - 0.3	V_{DD3} - 0.3	V
f_{SYS}	Operating main clock frequency (core, AHB, S-APB)	-	70	MHz
f_{APB}	Operating peripheral clock frequency (A-APB)	-	35	MHz
T_A	Ambient temperature under bias	–40	+85	°C

1. For details on operating conditions related to CLK input buffer refer to section 28.7 on page 380.
2. For details on operating conditions related to PLL refer to section 28.10 on page 387.
3. For details on operating conditions related to A/D converter refer to section 28.11 on page 389.

28.5 Thermal characteristics

The average chip-junction temperature, T_J, in degrees Celsius, may be calculated using the following equation:

$T_J = T_A + (P_D \times \Theta_{JA})$ (1)

Where:

– T_A is the Ambient Temperature in °C,
– Θ_{JA} is the Package Junction-to-Ambient Thermal Resistance, in °C/W,
– P_D is the sum of P_{INT} and $P_{I/O}$ ($P_D = P_{INT} + P_{PORT}$),
– P_{INT} is the product of I_{DD} and V_{DD}, expressed in Watt. This is the Chip Internal Power,
– P_{PORT} represents the Power Dissipation on Input and Output Pins; User Determined.

P_{PORT} is going to be significant when the device is configured to drive continuously external memories and/or modules, otherwise $P_{PORT} < P_{INT}$ and it might be neglected.

An approximate relationship between P_D and T_J (if P_{PORT} is neglected) is given by:

$P_D = K / (T_J + 273°C)$ (2)

Therefore (solving equations 1 and 2):

$K = P_D \times (T_A + 273°C) + \Theta_{JA} \times P_D^2$ (3)

Where:

– K is a constant for the particular part, which may be determined from equation (3) by measuring P_D (at equilibrium) for a known T_A. Using this value of K, the values of P_D and T_J may be obtained by solving equations (1) and (2) iteratively for any value of T_A.

Table 91. Thermal Characteristics

Symbol	Description	Value		Unit
		Min	Max	
Θ_{JA}	Thermal Resistance Junction-Ambient PQFP 208 - 28 x 28 x 3.4mm / 0.5 mm pitch	-	48	°C/W
P_D	Power dissipation(1) PQFP208 - 28 x 28 x 3.4mm / 0.5 mm pitch	-	-	mW
T_J	Junction temperature(2)	-	-	°C

1. The power dissipation is obtained from the formula $P_D=P_{INT}+P_{PORT}$ where:
   P_{INT} is the chip internal power (I_{DD}xV_{DD})
   P_{PORT} is the port power dissipation determined by the user.
2. The average chip-junction temperature can be obtained from the formula $T_J = T_A + P_D \times \Theta_{JA}$.

28.6 Supply Current Characteristics

Measurements are done in the following conditions:

– Program executed from internal RAM, CPU running with RAM access.
– In RUN mode measurement all internal peripherals are clocked but not used. A-APB clock runs at half AHB clock frequency. In all other mode measurements all peripherals are switched off.
– All I/O pins in input mode with a static value at V_{DD3} or V_{SS} (no load).
– PLL switched off (for PLL current characteristics see section 28.10 on page 387).

This implies that I/O related current is not considered in these figures.

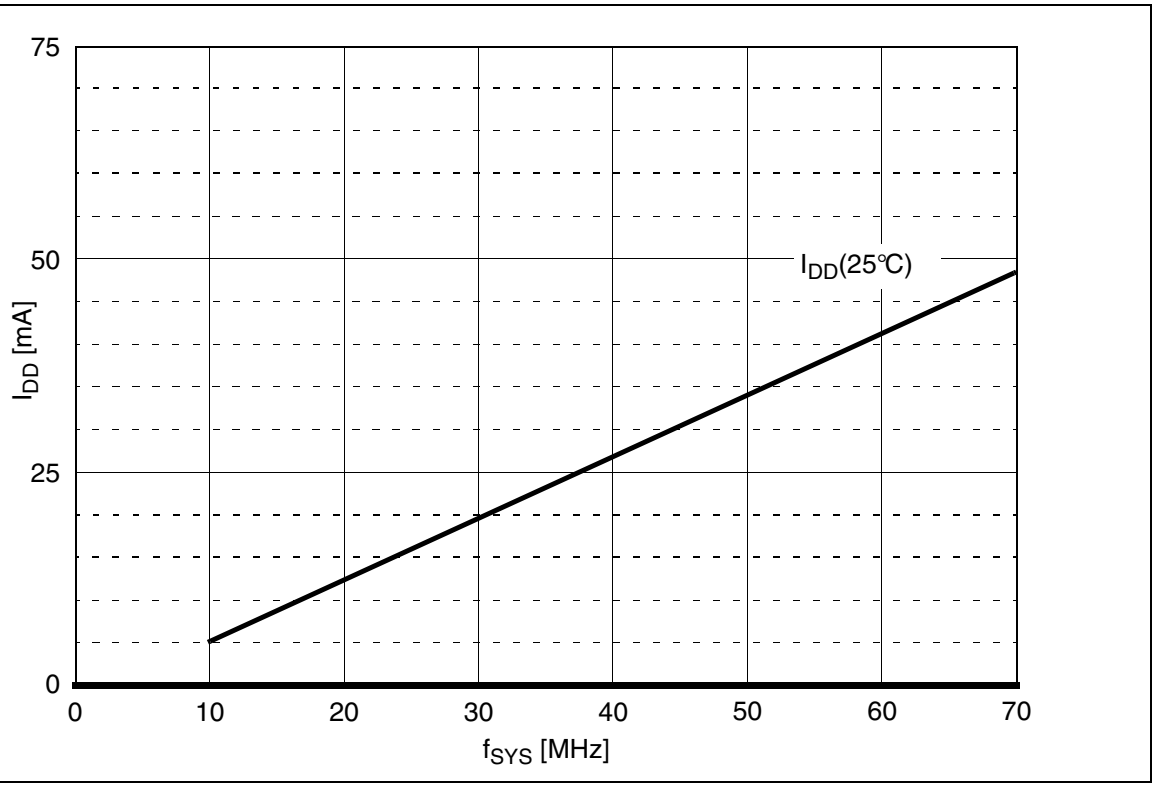
Table 92. Current consumption characteristics

Symbol	Parameter	Test Conditions(1)	Value			Unit
			Min	Typ	Max	
I_{DD_RUN}	V_{DD} RUN Mode current	@70 MHz System Clock	-	48	-	mA
I_{DD_IDLE}	V_{DD} IDLE Mode current	@1.6 MHz System Clock	-	1.25	-	mA
I_{DD_SLP}	V_{DD} SLOW Mode current	@32 kHz System Clock	-	17	-	µA
I_{DD_STP}	V_{DD} STOP Mode current	@0 MHz System Clock	-	16	-	µA
I_{DDRTC}	V_{DDRTC} all modes current	@32 kHz RTC Clock	-	3	-	µA
I_{STB}	STANDBY Mode current	0 V for the system and IO power supply, 1.8 V for RTC running on 32 kHz oscillator	-	3.5	-	µA
$I_{DD}(f)$	V_{DD} current function(1)	System clock frequency f between 10 and 25 MHz.	-	0.706 f_{SYS} - 2.229	-	mA

1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
2. Data based on design characterization performed on sample devices, not tested in production.

Following Figure 74 illustrates the dependency between system clock frequency in MHz and V_{DD} current consumption in mA, measured in typical conditions and in the indicated range.

Figure 74. Supply current versus system frequency



28.7 Clock and Timing Characteristics

28.7.1 Internal clock characteristics

Application software controls STR720 internal clocks by configuring RCCU registers and setting PLL related parameters (see Chapter 13: RESET AND CLOCK CONTROL UNIT (RCCU) on page 138). The values hereafter reported defines the limit for internal frequencies that must be respected by clock configuration.

Table 93. Internal clock limits

Symbol	Parameter	Conditions	Value			Unit
			Min	Typ	Max	
f_{CPU}	CPU, AHB and S-APB clock frequency		-	50	70	MHz
f_{APB}	A-APB clock frequency		f_{CPU} / 16	-	f_{CPU} / 2	MHz

28.7.2 CLK pad characteristics (External Clock Source)

These pads are used to provide clock signal to STR720 system from an external clock source. Since their electrical characteristics are different from normal I/O pads they are hereafter reported in a specific table.

Table 94. DC CLK pad characteristics

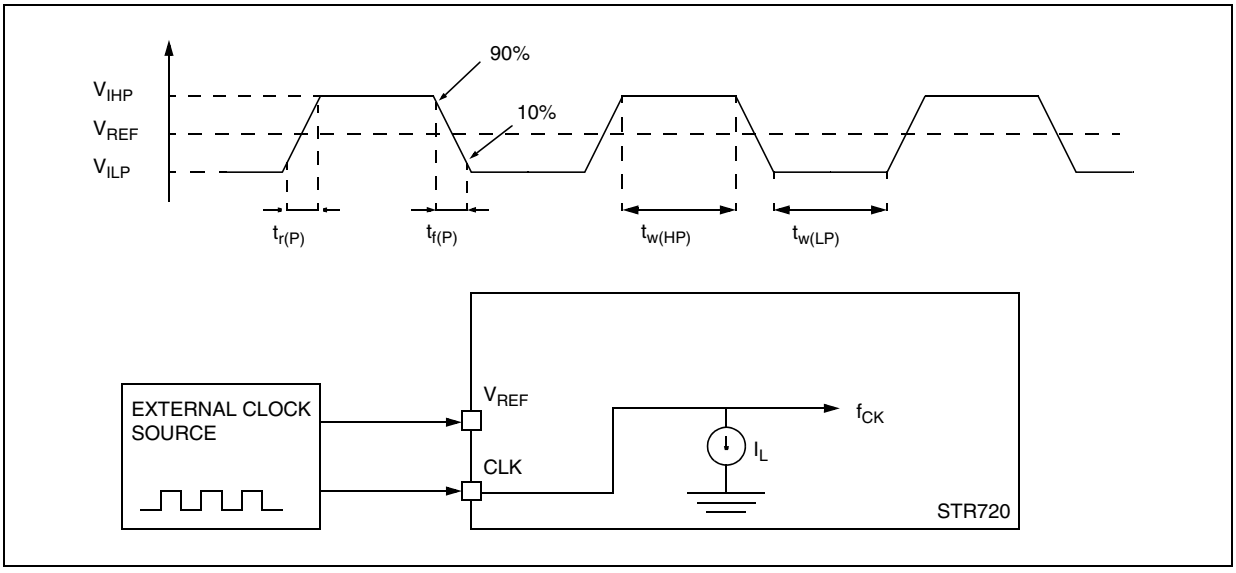
Symbol	Parameter	Test Conditions(1)	Value			Unit
			Min	Typ	Max	
V_{DD3P}	Supply Voltage for CLK input buffer.		3.14	3.30	3.47	V
V_{REF}	Reference Voltage for CLK input signal (2)	$V_{REF} = (V_{DD3P} - 1) * 0.9$ ±± 60 mV	1.87	2.1	2.28	V
V_{REF_R}	Recommended reference Voltage for CLK input signal		1.6	-	2.0	V
V_{IHP} (CLK)	Main CLK Clock (CLK) Input high level (2)	See Figure 75	V_{REF} + 0.2	-	-	V
V_{ILP} (CLK)	Main CLK Clock (CLK) Input low level (2)	See Figure 75	-	-	V_{REF} - 0.2	V

Table 95. AC CLK pad characteristics

Symbol	Parameter	Conditions	Value			Unit
			Min	Typ	Max	
$t_{w(HP)}$ $t_{w(LP)}$	Main CLK Clock (CLK) high or low time (2)	See Figure 75	14.29	-	-	ns
$t_{r(P)}$ $t_{f(P)}$	Main CLK Clock (CLK) rise or fall time (2)	See Figure 75	-	-	7	ns
I_L	CLK Input leakage current	$V_{SS} \le V_{IN} \le V_{DD3P}$	-	-	±1	µA
I_{DD3P_R}	V_{DD3P} RUN/IDLE mode current		-	2.75	-	mA
I_{DD3P_S}	V_{DD3P} SLOW/STOP mode current		-	-	10	nA

1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
2. Data based on design guidelines and validation, not tested in production.

Figure 75. Typical Application with External Clock Source



28.7.3 PLL Characteristics

STR720 implements a PLL which combines different levels of frequency dividers with a Voltage Controlled Oscillator (VCO) working as frequency multiplier. On chip PLL is used to generate system clock from an external reference clock signal. Its characteristics in term of generated clock accuracy depends heavily on the input signal and power supply quality.

In the following table, jitter is intended as the difference of the T_{MAX} and T_{MIN}, where T_{MAX} is maximum time period of the PLL output clock and T_{MIN} is the minimum time period of the PLL output clock. The value is expressed as ratio with respect to the average/nominal time period of PLL output clock.

Jitter at the PLL output can be due to the many different reasons, the major contributors being noise on input reference clock and noise on PLL supply (internally or externally generated).

PLL acts like a low pass filter for any jitter in the input clock. Input Clock jitter with the frequencies within the PLL loop bandwidth is passed to the PLL output and higher frequency jitter (frequency > PLL bandwidth) is attenuated @20dB/decade. This is the reason why, depending on input clock noise spectral characteristics, jitter performance can be better when PLL output is used as system clock than when PLL is bypassed.

Digital supply noise adds deterministic components to the PLL output jitter, independent on multiplication factor. Its effects is strongly reduced thanks to particular care used in the physical implementation and integration of the PLL module inside the device. Particular care is thus recommended in board design for what concerns V_{DDPLL} and clock routing.

Table 96. PLL Characteristics

Symbol	Parameter	Test Conditions(1)	Value			Unit
			Min	Typ	Max	
V_{DDPLL}	Supply Voltage for PLL		1.6	1.80	2.0	V
f_{REF}	PLL Reference clock frequency (2)		9	-	27	MHz
f_{VCO}	PLL VCO clock frequency(2)		-	-	1080	MHz
t_{LOCK}	PLL lock-in time		80	116	±200	µs
T_{JITTER}	Peak-to-peak PLL jitter value(3)	$(T_{MAX}-T_{MIN}) / T_{OUT}$ V_{REF} = 1.64 V ΔV_{CLK} = 400 mV f_{CLK} = 49.1 MHz DIV2 enabled,M=11, D=4.	-	2.9	3.5	%
σ_{JITTER}	Standard deviation PLL jitter value(3)	$\sigma(T) / T_{OUT}$ same as above.	-	0.5	0.7	%
I_{DDPLL_R}	V_{DDPLL} RUN mode current	Same as above	-	1.3	1.4	mA
I_{DDPLL_O}	V_{DDPLL} off current		-	-	20	nA

1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
2. Data based on design guidelines and validation, not tested in production.
3. Data based on design characterization performed on sample devices, not tested in production.

28.7.4 32 kHz Real-Time Clock Oscillator

The RTC block can be driven by a crystal oscillator. In the application, the crystal and the load capacitors have to be placed as close as possible to the oscillator pins in order to minimize output distortion and start-up stabilization time. Refer to the crystal manufacturer for more details (frequency, package, accuracy, etc.). A recommended configuration is shown on Figure 76 and detailed on Table 97 whose values have been checked using standard crystals.

Figure 76. Crystal Oscillator typical application

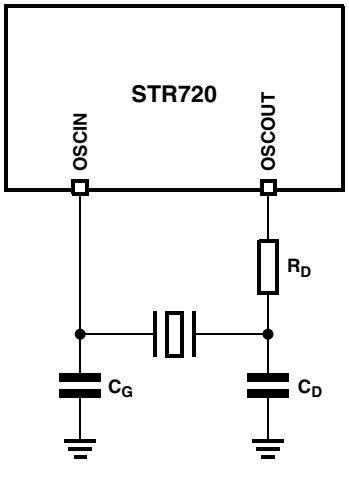


Table 97. Crystal oscillator recommanded characteristics

Symbol	Parameter	Conditions	Value			Unit
			Min	Typ	Max	
f_{OSCIN}	Crystal Frequency		-	32.678	-	kHz
f_{TOL}	Crystal frequency tolerance		-30	-	+30	ppm
R_D	Serial drain resistor		-	220	-	kΩ
C_G, C_D	Recommended load capacitors versus crystal load capacitance	R_D = 220 kΩ C_L = 9 pF Including board-stray capacitances	-	15	-	pF
		R_D = 220 kΩ C_L = 12.5 pF Including board-stray capacitances	-	22	-	pF

The electrical characteristics of STR720 oscillator are reported in Table 98.

Table 98. Oscillator electrical characteristics

Symbol	Parameter	Test Conditions(1)	Value			Unit
			Min	Typ	Max	
V_{DDRTC}	Supply Voltage for PLL		1.71	1.80	1.89	V
g_m	Oscillator Transconductance		14	30	44	µA/V
t_{SETUP}	Oscillator Start-up Time (2)	Stable V_{DD}	400	500	1000	ms
V_q	Operating point voltage		-	956	-	mV
V_{OSC}	Output voltage amplitude(2)	C_L = 12 pF, R_D = 220 kΩ	-	788	-	mV(rms)
R_{OSC}	Oscillation allowance(2)	C_L = 12 pF, R_D = 220 kΩ	500	-	-	kΩ
f_{OFFSET}	Static frequency deviation (ΔF/F)(2)	C_L = 12 pF, R_D = 220 kΩ	-	+4.0	-	ppm

1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
2. Data based on design characterization done with AEL Crystal (N. 4065), not tested in production

28.8 AC and DC characteristics

The parameters listed in the following tables represent the electrical characteristics of STR720 general purpose I/O pins and external memory interfaces (EMI and SDRAM). On Table 99 the static characteristics are reported.

Table 99. DC Electrical Characteristics

Symbol	Parameter	Test Conditions[1]	Value			Unit
			Min.	Typ	Max	
V_{IH}	Input high level TTL P2(15,14,12:0), P3(12:0), P4(4:0), P6(31:0), P7(45,44,15:0), JTDI, JTCK, JTMS		2.0	-	-	V
	Input high level TTL Schmitt Trigger (P2.13)		2.0	-	-	V
V_{IL}	Input low level TTL P2(15,14,12:0), P3(12:0), P4(4:0), P6(31:0), P7(45,44,15:0), JTDI, JTCK, JTMS		-	-	0.8	V
	Input low level TTL Schmitt Trigger(P2.13)		-	-	0.8	V
V_{HYS}	Input hysteresis TTL Schmitt trigger (P2.13)		0.4	-	-	V
V_{OH}	Output High Level Standard P2(15:0), P3(12:0), P4(3:0)	Push Pull, I_{OH} = 2 mA	V_{DD3} - 0.8	-	-	V
	Output High Level: P4(4),P7(43:0), JTDO, JTCK	Push Pull, I_{OH} = 4 mA	V_{DD3} - 0.8	-	-	V
	Output High Level: P6(59:47,45:0)	Push Pull, I_{OH} = 8 mA	V_{DD3} - 0.8	-	-	V
V_{OL}	Output Low Level Standard P2(15:0), P3(12:0), P4(3:0)	Push Pull, I_{OL}= 2 mA	-	-	0.4	V
	Output Low Level: P4(4),P7(43:0), JTDO, JTCK	Push Pull, I_{OL}= 4 mA	-	-	0.4	V
	Output Low Level: P6(59:47,45:0)	Push Pull, I_{OL}= 8 mA	-	-	0.4	V
R_{WPU}	Weak pull-up resistor P2(15:0)		-	15	-	kΩ
R_{WPD}	Weak pull-down resistor P3(12:0), P4(3:0)		-	15	-	kΩ
I_{LEAK}	Input leakage current		-	-	1	µA

1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.

In the following table the dynamic characteristics of general purpose I/O pins and external memory interfaces (EMI and SDRAM are reported). For a description of external memory

interface timing characteristics, see Section 28.12: *External Memory Bus Timing on page 393*

Table 100. AC Electrical Characteristics

Symbol	Parameter	Test Conditions(1)	Value			Unit
			Min.	Typ	Max	
T_{TR}	Output transition time (10%-90% and 90%-10%) P2(15:0), P3(12:0), P4(3:0)(2)	C_L = 50pF	-	-	5	ns
	Output transition time (10%-90% and 90%-10%) P4(4),P7(43:0), JTDO, JTCK(2)	C_L = 50pF	-	-	2.5	ns
	Output transition time (10%-90% and 90%-10%) P6(59:47,45:0)(2)	C_L = 30pF	-	-	1	ns

1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
2. Data based on design guidelines and validation, not tested in production.

28.9 Asynchronous Reset Input Characteristics

All reset input pins implement a low-pass glitch filter which improves their noise immunity and prevents system from malfunctioning as it would happen if they got activated inappropriately. The electrical characteristics of these special pins, both static and dynamic, are reported in Table 101. It must be noted that input characteristics of $\overline{\text{RSTIN}}$ ad $\overline{\text{RTCRST}}$ are different from standard I/Os, due to the fact that they are supplied at 1.8 V by V_{DDRTC} and they have to tolerate 3.3 V input signals even when no such voltage is applied to STR720 device (STANDBY mode).

Table 101. Reset pads input filter characteristics

Symbol	Parameter	Test Conditions(1)	Value			Unit
			Min	Typ	Max	
V_{IL}	Input low level TTL Schmitt trigger ($\overline{\text{JTRST}}$)		-	-	0.8	V
	Input low level Schmitt trigger ($\overline{\text{RSTIN}}$, $\overline{\text{RTCRST}}$)		0.43	0.5	0.65	V
V_{IH}	Input high level TTL Schmitt trigger ($\overline{\text{JTRST}}$)		2.0	-	-	V
	Input high level Schmitt trigger ($\overline{\text{RSTIN}}$, $\overline{\text{RTCRST}}$)		0.72	0.9	0.98	V
V_{HYS}	Input hysteresis TTL Schmitt trigger ($\overline{\text{JTRST}}$)		0.4	-	-	V
	Input hysteresis Schmitt trigger ($\overline{\text{RSTIN}}$, $\overline{\text{RTCRST}}$)		-	-	-	V
t_{FRS}	Input filtered pulse $\overline{\text{JTRST}}$, $\overline{\text{RSTIN}}$, $\overline{\text{RTCRST}}$		-	-	50	ns
t_{NFR}	Input not filtered pulse $\overline{\text{JTRST}}$, $\overline{\text{RSTIN}}$, $\overline{\text{RTCRST}}$		500	-	-	ns

1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.

28.10 USB - Universal Bus Interface

In order to implement a correct USB interface, a few external components are required to be compliant to the protocol electrical characteristics. A possible application scheme is depicted in Figure 77 and detailed in Table 102.

Figure 77. USB typical application

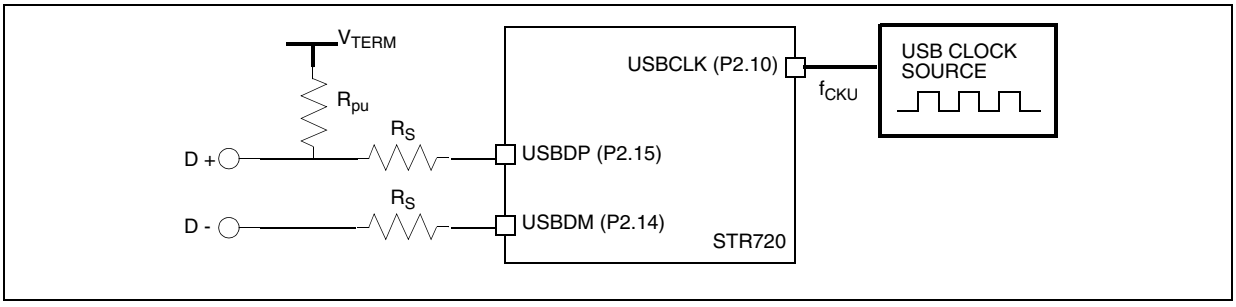


Table 102. USB typical application recommendations

Symbol	Parameter	Conditions	Value			Unit
			Min	Typ	Max	
R_S	Series resistor	Accuracy: ±5%	25.65	27	28.35	Ω
R_{pu}	Line pull-up resistor	Accuracy: ±5%	1.425	1.500	1.575	kΩ
V_{TERM}	Line termination voltage		3.0	3.3	3.6	V
V_{IHU}	USB clock signal input high level	Same as P2.10 GPIO	2	-	-	V
V_{ILU}	USB clock signal input low level	Same as P2.10 GPIO	-	-	0.8	V
f_{CKU}	USB clock signal frequency	Accuracy: ± 0.25%	47.88	48.00	48.12	MHz

Hereafter the electrical characteristics of STR720 USB interface are reported. These values are measured with a specific setup following USB standard recommendations and summed up in the “Test Condition” column of the table, where R_S is the series resistor value and C_L the capacitive load to GND connected to each USB pad.

Table 103. USB DC Characteristics

Symbol	Parameter	Test Conditions(1)	Value			Unit
			Min	Typ	Max	
V_{IL}	Single-ended receiver low level input voltage		-	-	0.8	V
V_{IH}	Single-ended receiver high level input voltage		2.0	-	-	V
V_{HYST}	Single-ended receiver hysteresis		-	400	0	mV
V_{DI}	Differential input sensitivity	|USBDP - USBDM|	200	-	-	mV
V_{CM}	Differential common mode range(2)	Includes VDI range	0.8	-	2.5	V

Table 103. USB DC Characteristics

Symbol	Parameter	Test Conditions[1]	Value			Unit
			Min	Typ	Max	
V_{OL}	Low level output voltage	Measured with pull-up of 1.425 kΩ to 3.6 V	0.0	-	0.3	V
V_{OH}	High level output voltage	Measured with pull-down of 14.25 kΩ to GND	2.8	-	3.6	V
V_{CRS}	Output signal crossover voltage	Excluding the first transition from Idle state	1.3	-	2.0	V

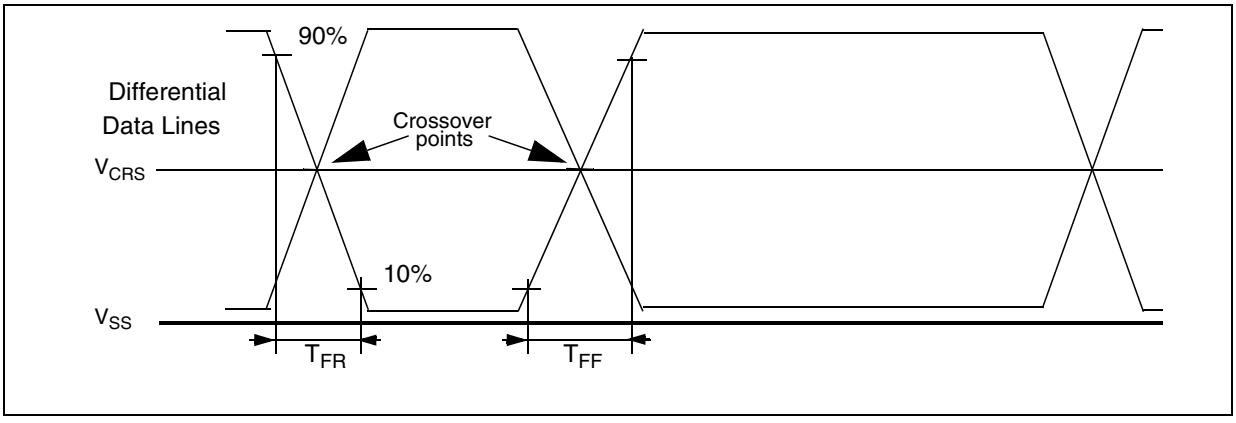
1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
2. Data based on design guidelines and validation, not tested in production.

Table 104. USB Full-speed AC Characteristics

Symbol	Parameter	Test Conditions[1]	Value			Unit
			Min	Typ	Max	
T_{FR}	Rise Time	R_S = 27 Ω, C_L = 50 pF. See Figure 78.	4	-	20	ns
T_{FF}	Fall Time	R_S = 27 Ω, C_L = 50 pF. See Figure 78.	4	-	20	ns
T_{FRFM}	Differential rise and fall time matching	(T_{FR} / T_{FF}) excluding the first transition from Idle state	90	-	111.11	%
Z_{DRV}	Driver output resistance[2]	R_S = 27 Ω.	28	-	44	Ω

1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
2. Data based on design guidelines and validation, not tested in production.

Figure 78. USB Data signal rise and fall time



28.11 S-D ADC characteristics

The static electrical characteristics of A/D converter are reported in Table 105. In the subsequent sections, parameters related to input circuit and A/D converter performance are listed.

Table 105. ADC static characteristics

Symbol	Parameter	Test Conditions[1]	Value			Unit
			Min.	Typ	Max	
A_{VDD}	Analog supply voltage for ADC	With reference to A_{VSS} pin	3.0	3.30	3.6	V
A_{IDD}	A_{VDD} operating mode current		-	1.8	-	mA
A_{IDD_O}	A_{VDD} off current	ADC off but CLK pad active[2]	-	-	-	mA
A_{IDD_S}	A_{VDD} standby/stop current	ADC off and CLK pad not used[2]	-	-	1	µA
V_{AIN}	Analog input voltage	See note (3)	0	-	2.5	V
I_{ALEAK}	Analog input leakage current		-	-	200	nA

1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
2. As long as CLK pad is used to provide system clock, either directly or thorugh PLL, a consumption will be observed on A_{VDD} pad, due to CLK pad reference current generators. Only when STR720 is put in SLOW mode (32 kHz) or STOP mode (no clock) CLK pad reference current generators can be switched off and the minimum A_{VDD} consumption can be achieved.
3. V_{AIN} may exceed A_{GND} or A_{VDD} up to the absolute maximum ratings. However, the conversion result in these cases will be meaningless. Data based on design guidelines, not tested in production.

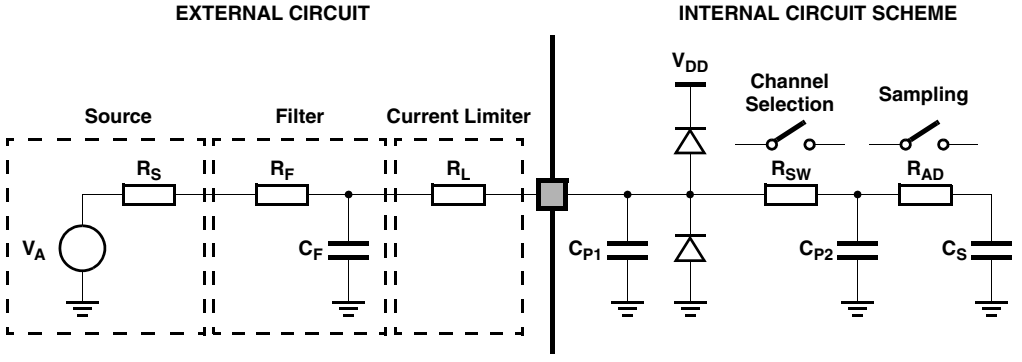
28.11.1 ADC analog input pins

To improve the accuracy of the A/D converter, the first recommendation is to properly connect decoupling capacitors on A_{VDD} pin, so to reduce as much as possible the amount of noise introduced through the analog supply voltage pin. Subsequently it is definitively necessary for analog input pins to have low AC impedance. Connecting a capacitor with good high frequency characteristics at the input pin of the device, can be effective: the capacitor should be as large as possible, ideally infinite. This capacitor contributes to attenuate noise present on the input pin; moreover, it sources charge during the sampling phase, when the analog signal source is a high-impedance source.

On each analog input pin an anti-aliasing filter is required so to avoid that components beyond the sampling frequency are folded back into the actual signal bandwidth. Due to the structure of Sigma-Delta ADC, these components cannot be removed by software post-processing of the acquired samples, so it is of fundamental importance to remove them before feeding the signal to the analog input pin. An anti-aliasing filter can be obtained in general by using a series resistance with a capacitor on the input pin (simple RC Filter), though other more complex solutions with better band rejection characteristics can be used. The RC filtering may be limited according to the value of source impedance of the transducer or circuit supplying the analog signal to be measured.

Here you can find an equivalent circuit of ADC input pin modeling the behaviour during the sampling phase.

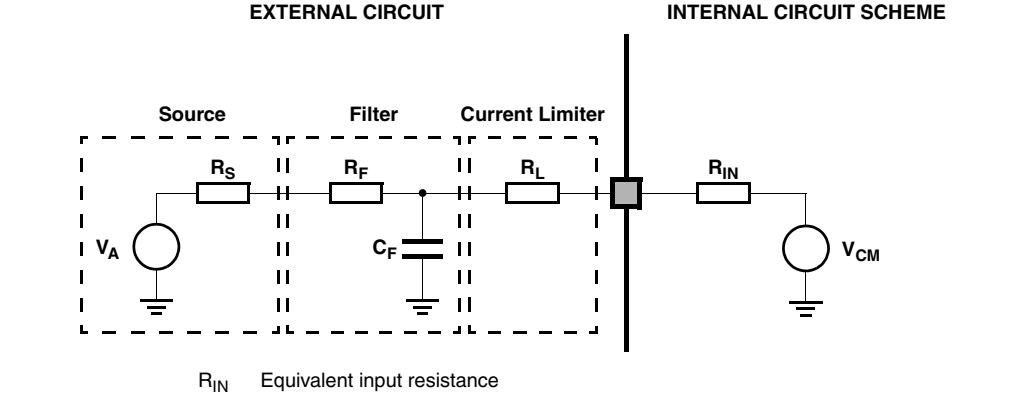
Figure 79. ADC input pin equivalent circuit during sampling



R_S	Source Impedance
R_F	Filter Resistance
C_F	Filter Capacitance
R_L	Current Limiter Resistance
R_{SW}	Channel Selection Switch Impedance
R_{AD}	Sampling Switch Impedance
C_P	Pin Capacitance (two contributions, C_{P1} and C_{P2})
C_S	Sampling Capacitance

Just after the sampling phase is over, the charge on pin capacitors has to be restored by the signal voltage source. This can be modeled as a current requested by ADC input pin depending on the frequency with which the pin is sampled by ADC (which is not the signal sampling frequency due to the oversampling architecture of Sigma-Delta converters). Equivalent input circuit modeling this aspect can be depicted as illustrated in Figure 80.

Figure 80. ADC input pin equivalent circuit



R_{IN}	Equivalent input resistance
V_{CM}	Common mode voltage

Where R_{IN} can be defined by the following formula.

$$R_{IN} = K \cdot \frac{1}{F_{MOD}}$$

All parameters required to describe ADC input pads are reported in the following table.

Table 106. ADC Input Characteristics

Symbol	Parameter	Test Conditions(1)	Value			Unit
			Min.	Typ	Max	
C_{IN}	Analog input capacitance(2)	See Figure 79. $C_{IN} = C_{P1}+C_{P2}+C_S$	-	-	5	pF
F_{MOD}	Modulator frequency(3) (oversampling clock)		-	-	4	MHz
K	Equivalent analog input resistor constant(4)	See Figure 80. In single channel mode value must be divided by 4.	1728	2160	2592	kΩ·MHz
R_{IN}	Equivalent analog input resistance(4)	See Figure 80. F_{MOD} = 4 MHz, round-robin mode.	-	540	-	kΩ
V_{CM}	Common mode voltage(2)		1.19	1.25	1.31	V
f_{AIN}	Analog input bandwidth(2) (F_{MOD}/4096)	F_{MOD} = 4 MHz	-	-	976	Hz
t_s	Conversion time	F_{MOD} = 4 MHz	128			ns

1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
2. Data based on design guidelines, not tested in production.
3. Data based on design characterization performed on sample devices, not tested in production.
4. Data based on design guidelines and validation, not tested in production.

28.11.2 ADC performance

In the following sections the performance figures used to represent ADC characteristics are reported in Table 107 and briefly explained in the subsequent sections.

Table 107. ADC Performance

Symbol	Parameter	Test Conditions(1), (2)	Value			Unit
			Min.	Typ	Max	
SINAD	Signal/Noise and distortion		68	71	-	dB
ENOB	Effective number of bits (Resolution)		11	11.5	-	bit
INL	Integral non-linearity(3)	Measured on 11 bit samples.	-	0.5	-	LSB
DNL	Differential non-linearity(3)	Measured on 11 bit samples.	-	0.5	-	LSB
THD	Total harmonic distortion		71	79	-	dB
PBR	Pass-band ripple(4)		-	-	0.1	dB

1. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
2. Full range 80 Hz sine wave input, F_{MOD} = 2 MHz
3. Data based on design guidelines and validation, not tested in production.
4. Data based on design guidelines, not tested in production.

28.11.2.1 Signal to Noise ratio and distortion (SINAD)

The acquired samples are processed to calculate their FFT, after being windowed using a four-term Blackman-Harris function so to avoid spread of the fundamental harmonic due to the non-periodic captured components. On the FFT further processing is performed so to calculate the ratio between the power of input sine wave fundamental component and the sum of the noise power plus the power of all harmonic components except the fundamental. This parameter can be also referred to as THD+N.

28.11.2.2 Effective number of bits (ENOB)

This paramer is calculated based upon the measured SINAD value, as the number of bits of an ideal converter having an a signal-to-noise ratio equal to the SINAD of the real ADC under measurement. The ideal signal-to-noise ratio can be expressed as:

$$SNR_{ideal} = 1.76 + 6.02 \cdot N_{bits}$$

thus the effective number of bits can be determined using the following formula:

$$ENOB = \frac{SINAD - 1.76}{6.02}$$

28.11.2.3 Integral non-linearity (INL)

This parameter expresses the deviation between the actual A/D convertion characteristics and the ideal one by evaluating the distance between the center of the actual convertion step and the center of the bisector line, which is obtained by drawing a straight line from 1/2 LSB before the first step of real characteristic to 1/2 LSB after its last step.

28.11.2.4 Differential non-linearity (DNL)

This parameter again expresses the deviation between actual and ideal A/D convertion characteristics. This time the size of actual step with respect to the ideal one (1 ideal LSB) is compared.

28.11.2.5 Total harmonic distortion (THD)

The FFT computed on the acquired samples is processed so to calculated the ratio between the power of input sine wave fundamental component and the power of all other harmonic components.

28.11.2.6 Pass-band ripple (PBR)

This parameter defines the characteristics of A/D converter transfer function from analog input pin to the generated sample values.

28.12 External Memory Bus Timing

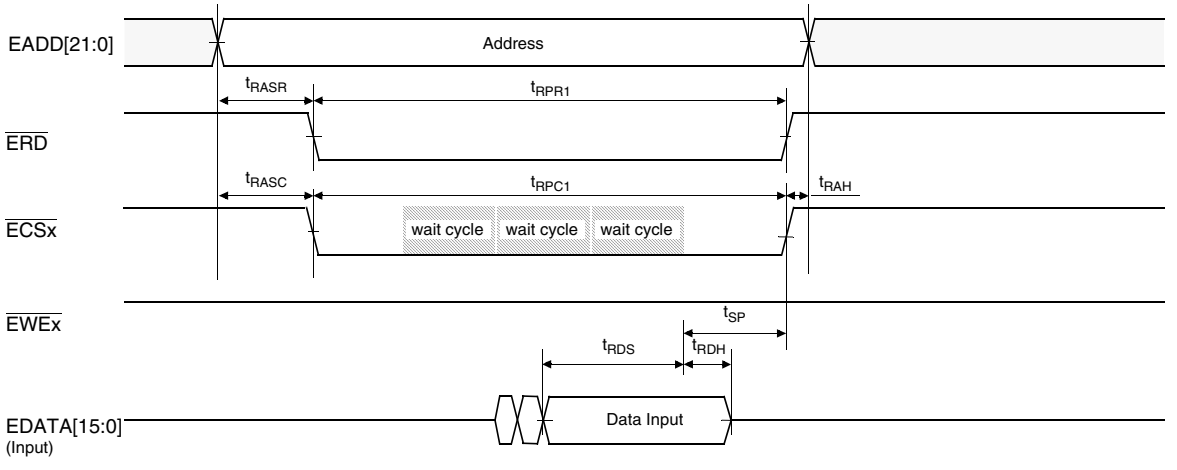
The signal composing the external memory interface are collected on Port 7 and as such all their DC electrical characteristics are reported in Table 99. The following tables and figures report the timing characteristics of read and write cycles.

Table 108. EMI Read cycle timing characteristics

Symbol	Parameter[1]	Test Conditions[2], [3]	Value[4]			Unit
			Min	Typ	Max	
t_{RPR1}	Read pulse time for $\overline{ERD}$ in 8/16-bit word access	Depends on EMI configuration.[5]	$2T_C$ - x		$9T_C$ + x	ns
t_{RPC1}	Read pulse time for $\overline{ECSx}$ in 8/16-bit word access	Depends on EMI configuration.[5]	$2T_C$ - x		$9T_C$ + x	ns
t_{RPR2}	Read pulse time for $\overline{ERD}$ in 32-bit word access	Depends on EMI configuration.[5]	$4T_C$ - x		$18T_C$ + x	ns
t_{RPC2}	Read pulse time for $\overline{ECSx}$ in 32-bit word access	Depends on EMI configuration.[5]	$4T_C$ - x		$18T_C$ + x	ns
t_{RSP}	Read sampling point		T_C - x		T_C + x	ns
t_{RDS}	Read data setup time					ns
t_{RDH}	Read data hold time					ns
t_{RASR}	Read address setup time to $\overline{ERD}$		T_C - x		T_C + x	ns
t_{RASC}	Read address setup time to $\overline{ECSx}$		T_C - x		T_C + x	ns
t_{RAH}	Read address hold time					ns

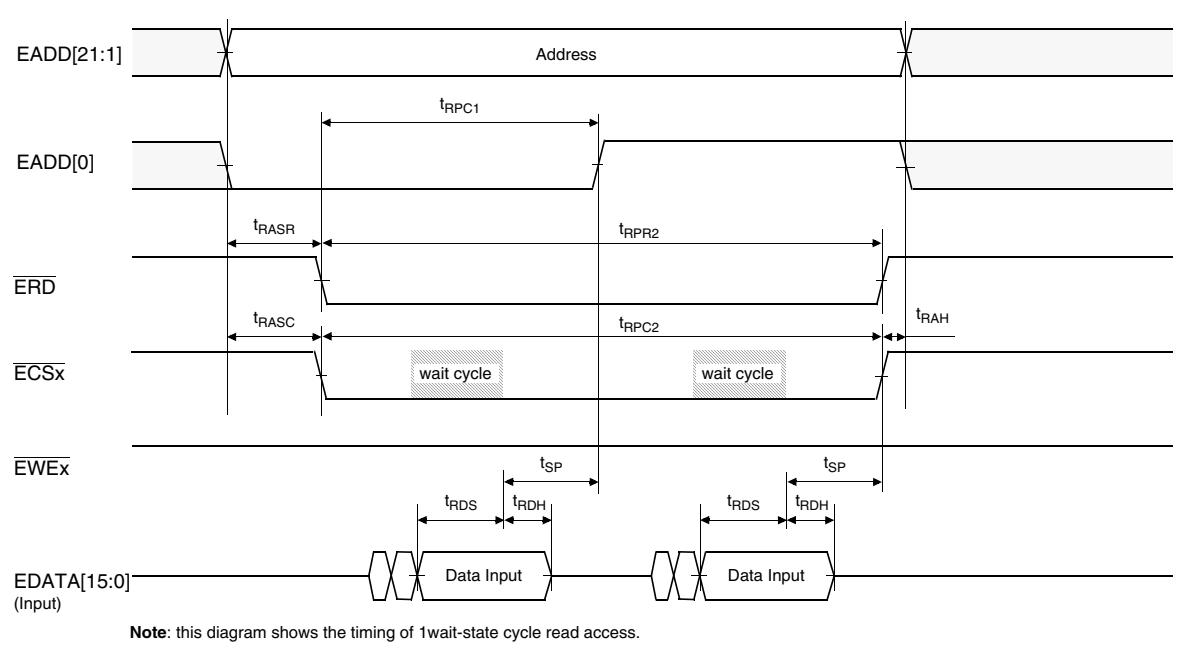
1. See Figure 81 and Figure 82 for a definition of these parameters.
2. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
3. Data based on design guidelines, not tested in production.
4. T_C is CPU clock period expressed in ns. For example if CPU clock is set to run at 50 MHz, T_C = 20 ns.
5. Actual duration is configurable. See Chapter 11: EXTERNAL MEMORY INTERFACE (EMI) on page 115.

Figure 81. EMI Read cycle timing (8-bit or 16-bit word)



Note: this diagram shows the timing of 3 wait-state cycle read access.

Figure 82. EMI Read cycle timing (32-bit word)



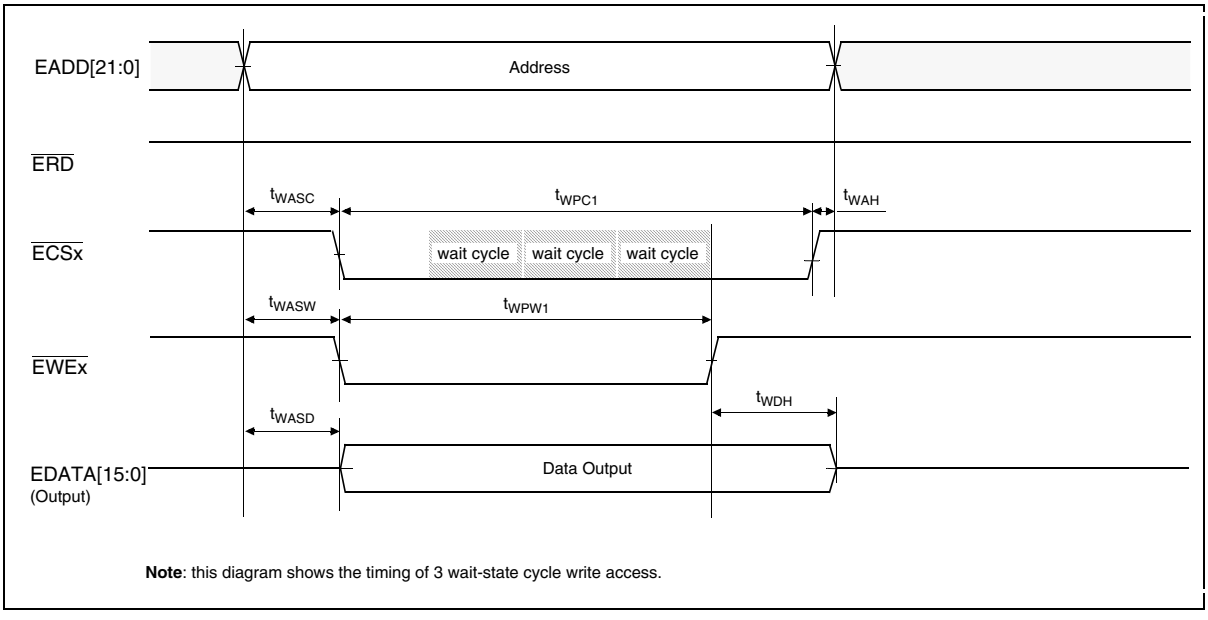
Note: this diagram shows the timing of 1wait-state cycle read access.

Table 109. EMI Write cycle timing characteristics

Symbol	Parameter[1]	Test Conditions[2], [3]	Value[4]			Unit
			Min	Typ	Max	
t_{WPC1}	Write pulse time for $\overline{ECSx}$ in 8/16-bit word access	Depends on EMI configuration.[5]	$2T_C$ - x		$9T_C$ + x	ns
t_{WPW1}	Write pulse time for $\overline{EWEx}$ in 8/16-bit word access	Depends on EMI configuration.[5]	T_C - x		$8T_C$ + x	ns
t_{WPC2}	Write pulse time for $\overline{ECSx}$ in 32-bit word access	Depends on EMI configuration.[5]	$4T_C$ - x		$18T_C$ + x	ns
t_{WDH}	Write Data Hold Time		T_C - x		T_C +x	ns
t_{WASC}	Write address setup time to $\overline{ECS}$		T_C - x		T_C +x	ns
t_{WASW}	Write address setup time to $\overline{EWEx}$		T_C - x		T_C +x	ns
t_{WASD}	Write address setup time to EDATA		T_C - x		T_C +x	ns
t_{WWT}	WEn Turnaround Time		T_C - x		T_C +x	ns
t_{WAH}	Write Address Hold Time					ns

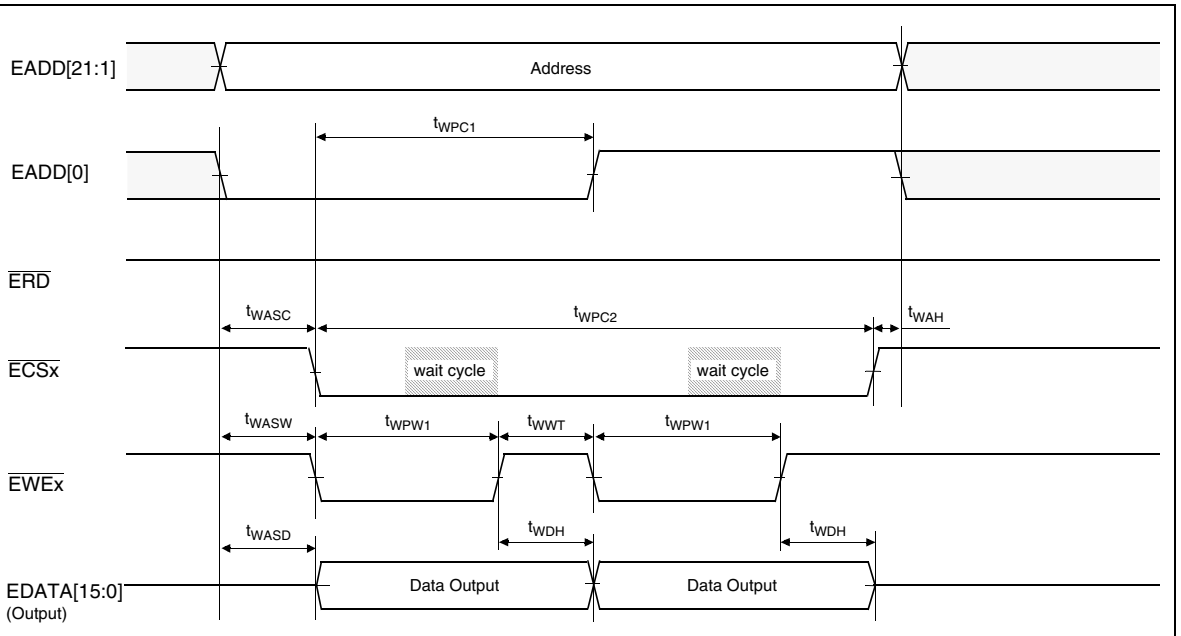
1. See Figure 83 and Figure 84 for a definition of these parameters.
2. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
3. Data based on design guidelines, not tested in production.
4. T_C is CPU clock period expressed in ns. For example if CPU clock is set to run at 50 MHz, T_C = 20 ns.
5. Actual duration is configurable. See Chapter 11: EXTERNAL MEMORY INTERFACE (EMI) on page 115.

Figure 83. EMI Write cycle timing (8-bit or 16-bit word)



Note: this diagram shows the timing of 3 wait-state cycle write access.

Figure 84. EMI Write cycle timing (32-bit word)



Note: this diagram shows the timing of 1 wait-state cycle write access.

28.13 SDRAM Interface Timing

The signals composing SDRAM interface are collected on Port 6 and as such all their DC electrical characteristics are reported in Table 99. The following tables and figures report the timing characteristics of read and write cycles, both as single and burst accesses.

Table 110. SDRAM Read cycle timing characteristics

Symbol	Parameter(1)	Test Conditions(2)	Value(4)			Unit
			Min	Typ	Max	
t_{SCH}	SDRAM clock high pulse time		-	$0.5T_C$	-	ns
t_{SCL}	SDRAM clock low pulse time		-	$0.5T_C$	-	ns
t_{SCAD}	SDRAM clock high to address valid		-	-	-	ns
t_{SCCS}	SDRAM clock high to chip select		-	-	-	ns
t_{SCRA}	SDRAM clock high to $\overline{RAS}$		-	-	-	ns
t_{SCCA}	SDRAM clock high to $\overline{CAS}$		-	-	-	ns
t_{SCWE}	SDRAM clock high to $\overline{WE}$		-	-	-	ns
t_{SDS}	SDRAM Data read setup(3)		-	-	-	ns
t_{SDH}	SDRAM Data read hold(3)		-	-	-	ns
t_{SCDA}	SDRAM clock high to data valid		-	-	-	ns
t_{SCBL}	SDRAM clock high to $\overline{BLS}$		-	-	-	ns
t_{SCCKE}	SDRAM clock high to CKEN		-	-	-	ns
t_{SRACA}	SDRAM $\overline{RAS}$ to $\overline{CAS}$ delay(3)	Depends on SDRAM configuration.(5)	T_C	-	$8T_C$	ns

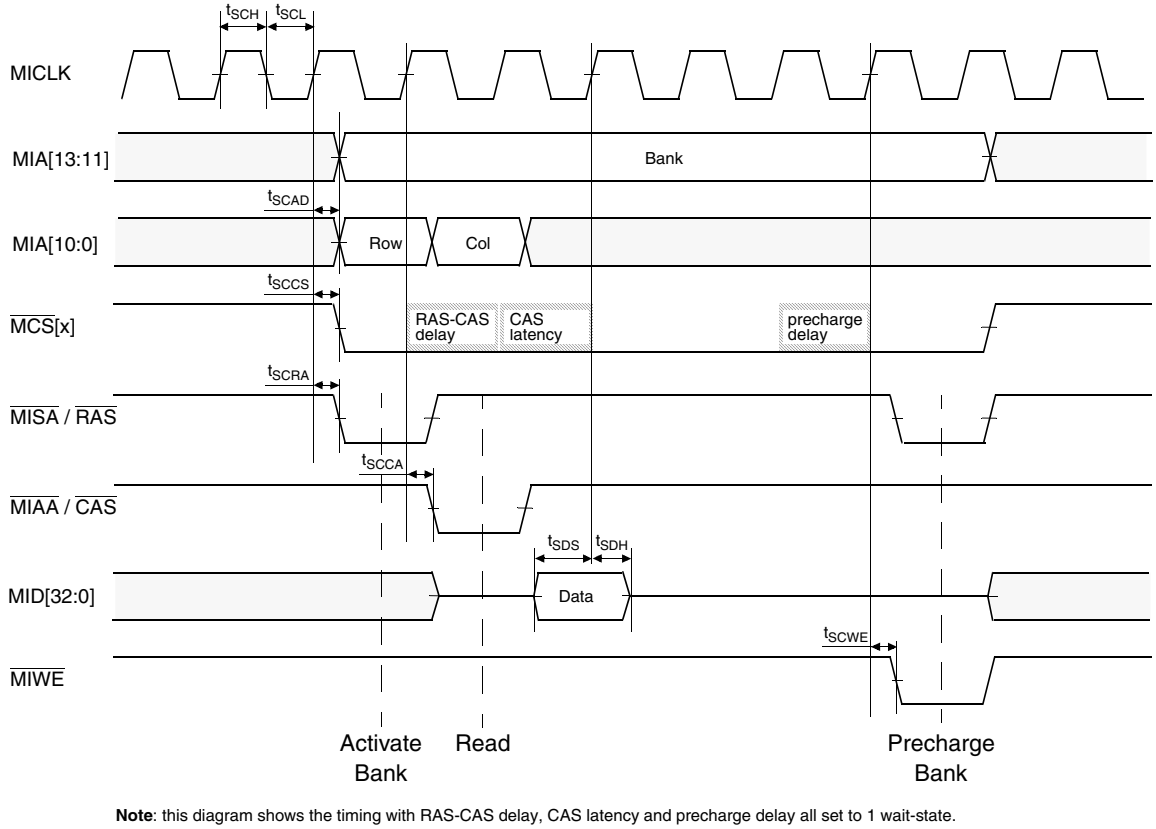
Table 110. SDRAM Read cycle timing characteristics

Symbol	Parameter[1]	Test Conditions[2]	Value[4]			Unit
			Min	Typ	Max	
t_{SCADA}	SDRAM $\overline{CAS}$ to data valid delay (CAS latency)[3]	Depends on SDRAM configuration.[5]	T_C	-	$3T_C$	ns
tSPRD	SDRAM precharge delay[3]	Depends on SDRAM configuration.[5]	0	-	$7T_C$	ns

1. See Figure 85, Figure 86 and Figure 87 for a definition of these parameters.
2. V_{DD}(all) = 1.80 V ± 10%, V_{DD3}(all) = 3.30 V ± 10%, T_A = -40 / +85 °C unless otherwise specified.
3. Data based on design guidelines, not tested in production.
4. T_C is CPU clock period expressed in ns. For example if CPU clock is set to run at 50 MHz, T_C = 20 ns.
5. Actual duration is configurable. See Chapter 10: DRAM CONTROLLER (DRAMC) on page 101.

In the following figures the timing diagrams are represented, where also the extent of the software configurable access timing is depicted.

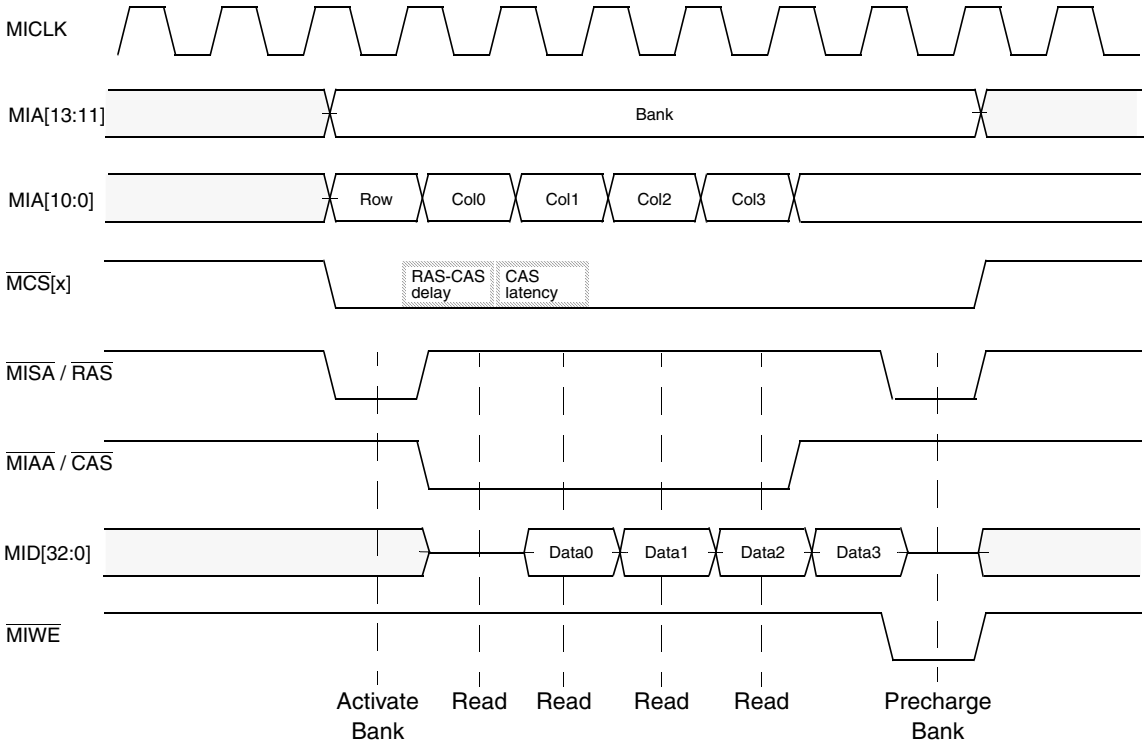
Figure 85. SDRAM Single Read Access



Note: this diagram shows the timing with RAS-CAS delay, CAS latency and precharge delay all set to 1 wait-state.

In figure 86 a sample of 4-beat burst read access is depicted, to show how access gets more efficient in the case of a burst read. The timing parameters of burst read access are the same of single read access.

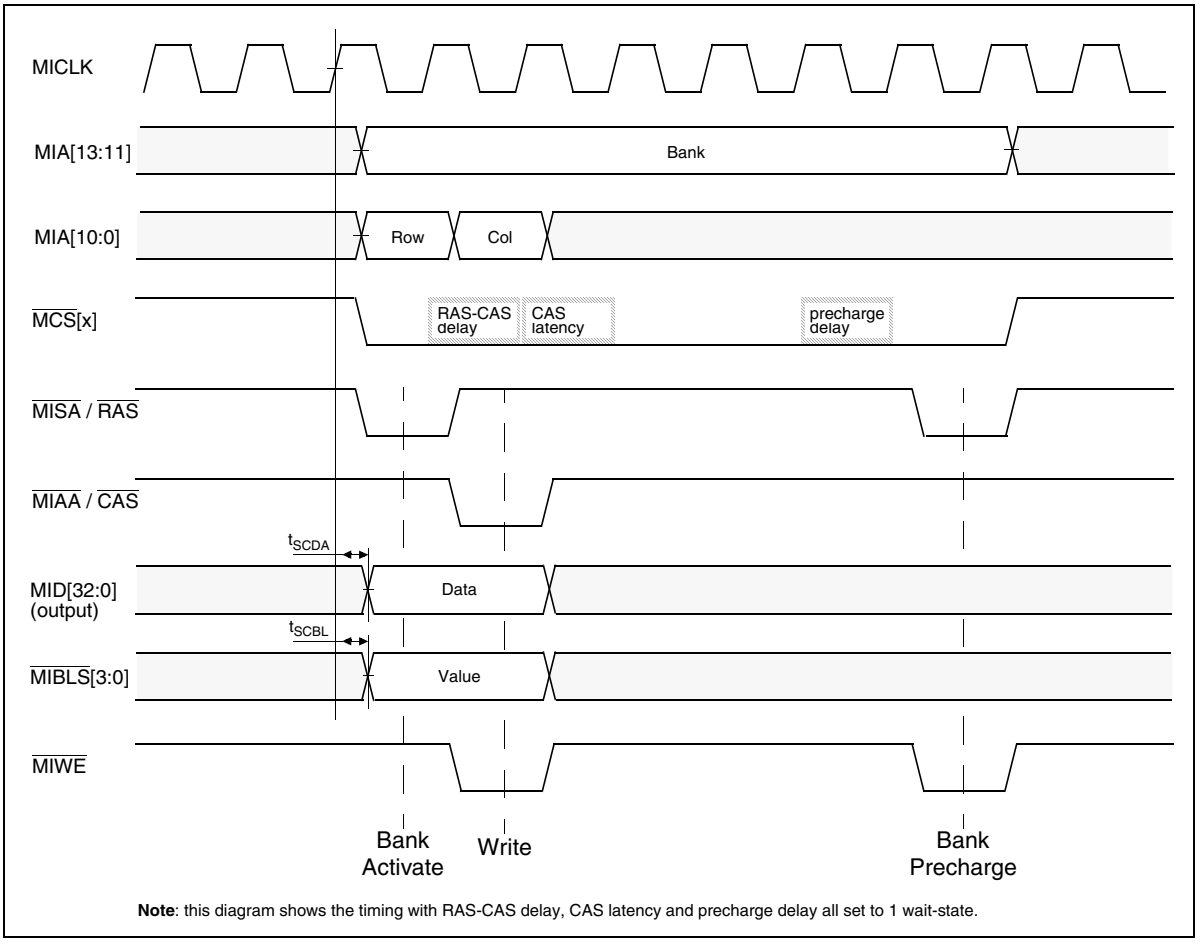
Figure 86. SDRAM Burst Read Access



Note: this diagram shows the timing with RAS-CAS delay, CAS latency set to 1 wait-state and no precharge delay.

The next two figures represent the write access timing, here considered in the single write case since burst write is performed using the same timing characteristics, and the entry in self-refresh (power-save) mode where MICKEN signal is used.

Figure 87. SDRAM Single Write Access



Note: this diagram shows the timing with RAS-CAS delay, CAS latency and precharge delay all set to 1 wait-state.

29 REVISION HISTORY

Date	Version	Comments
30 Jul 03	1.2	Preliminary distribution
22 Mar 04	2	Initial release
05 Apr 04	3	Modifed "Reset Management" on page 139
15 Apr 04	3.1	Corrected PDF links in table of contents
21 Dec 04	4.0	Revision number incremented from 3.1 to 4.0 due to Internal Document Management System change Modified page layout Updated Device Summary on page 1 Removed Section 4.12 CPG1 Package Pin configuration and Section 5.20 Embedded Trace Macrocell Renamed SLEEP mode to SLOW mode Updated ADC, CAN, USB, RTC, EFT, DMAC and ATAPI sections

Notes: