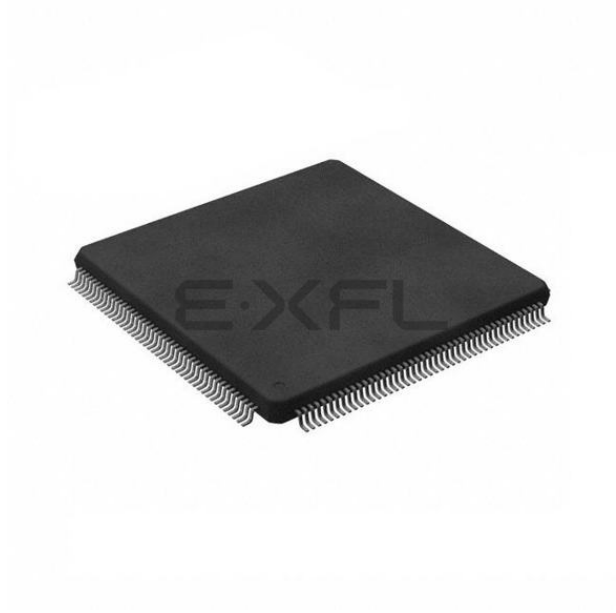


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	ARM7®
Core Size	16/32-Bit
Speed	25MHz
Connectivity	EBI/EMI, SPI, UART/USART
Peripherals	POR, WDT
Number of I/O	58
Program Memory Size	-
Program Memory Type	ROMless
EEPROM Size	-
RAM Size	2K x 8
Voltage - Supply (Vcc/Vdd)	1.8V ~ 3.6V
Data Converters	-
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	176-LQFP
Supplier Device Package	176-TQFP (24x24)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/at91m63200-25au

Pin Description

Table 2. AT91M63200 Pin Description

Module	Name	Function	Type	Active Level	Comments
EBI	A0 - A23	Address Bus	Output	–	All valid after reset
	D0 - D15	Data Bus	I/O	–	
	CS4 - CS7	Chip Select	Output	High	A23 - A20 after reset
	NCS0 - NCS3	Chip Select	Output	Low	
	NWR0	Lower Byte 0 Write Signal	Output	Low	Used in Byte Write option
	NWR1	Lower Byte 1 Write Signal	Output	Low	Used in Byte Write option
	NRD	Read Signal	Output	Low	Used in Byte Write option
	NWE	Write Enable	Output	Low	Used in Byte Select option
	NOE	Output Enable	Output	Low	Used in Byte Select option
	NUB	Upper Byte Select (16-bit SRAM)	Output	Low	Used in Byte Select option
	NLB	Lower Byte Select (16-bit SRAM)	Output	Low	Used in Byte Write option
	NWAIT	Wait Input	Input	Low	
	BMS	Boot Mode Select	Input	–	Sampled during reset
MPI	MPI_NCS	Chip Select	Input	Low	
	MPI_RNW	Read Not Write Signal	Input	–	
	MPI_BR	Bus Request from External Processor	Input	High	
	MPI_BG	Bus Grant to External Processor	Output	High	
	MPI_NOE	Output Enable	Input	Low	
	MPI_NLB	Lower Byte Select	Input	Low	
	MPI_NUB	Upper Byte Select	Input	Low	
	MPI_A1 - MPI_A9	Address Bus	Input	–	
	MPI_D0 - MPI_D15	Data Bus	I/O	–	
AIC	IRQ0 - IRQ3	External Interrupt Request	Input	–	PIO-controlled after reset
	FIQ	Fast External Interrupt Request	Input	–	PIO-controlled after reset
Timer	TCLK0 - TCLK5	Timer External Clock	Input	–	PIO-controlled after reset
	TIOA0 - TIOA5	Multi-purpose Timer I/O Pin A	I/O	–	PIO-controlled after reset
	TIOB0 - TIOB5	Multi-purpose Timer I/O Pin B	I/O	–	PIO-controlled after reset
USART	SCK0 - SCK2	External Serial Clock	I/O	–	PIO-controlled after reset
	TXD0 - TXD2	Transmit Data Output	Output	–	PIO-controlled after reset
	RXD0 - RXD2	Receive Data Input	Input	–	PIO-controlled after reset
SPI	SPCK	SPI Clock	I/O	–	PIO-controlled after reset
	MISO	Master In Slave Out	I/O	–	PIO-controlled after reset
	MOSI	Master Out Slave In	I/O	–	PIO-controlled after reset
	NSS	Slave Select	Input	Low	PIO-controlled after reset
	NPCS0 - NPCS3	Peripheral Chip Select	Output	Low	PIO-controlled after reset
PIO	PA0 - PA29	Programmable I/O Port A	I/O	–	Input after reset
	PB0 - PB27	Programmable I/O Port B	I/O	–	Input after reset
WD	NWDOVF	Watchdog Timer Overflow	Output	Low	Open drain
Clock	MCKI	Master Clock Input	Input	–	Schmitt trigger
	MCKO	Master Clock Output	Output	–	
Reset	NRST	Hardware Reset Input	Input	Low	Schmitt trigger, internal pull-up

Memory Map

Figure 3. AT91M63200 Memory Map

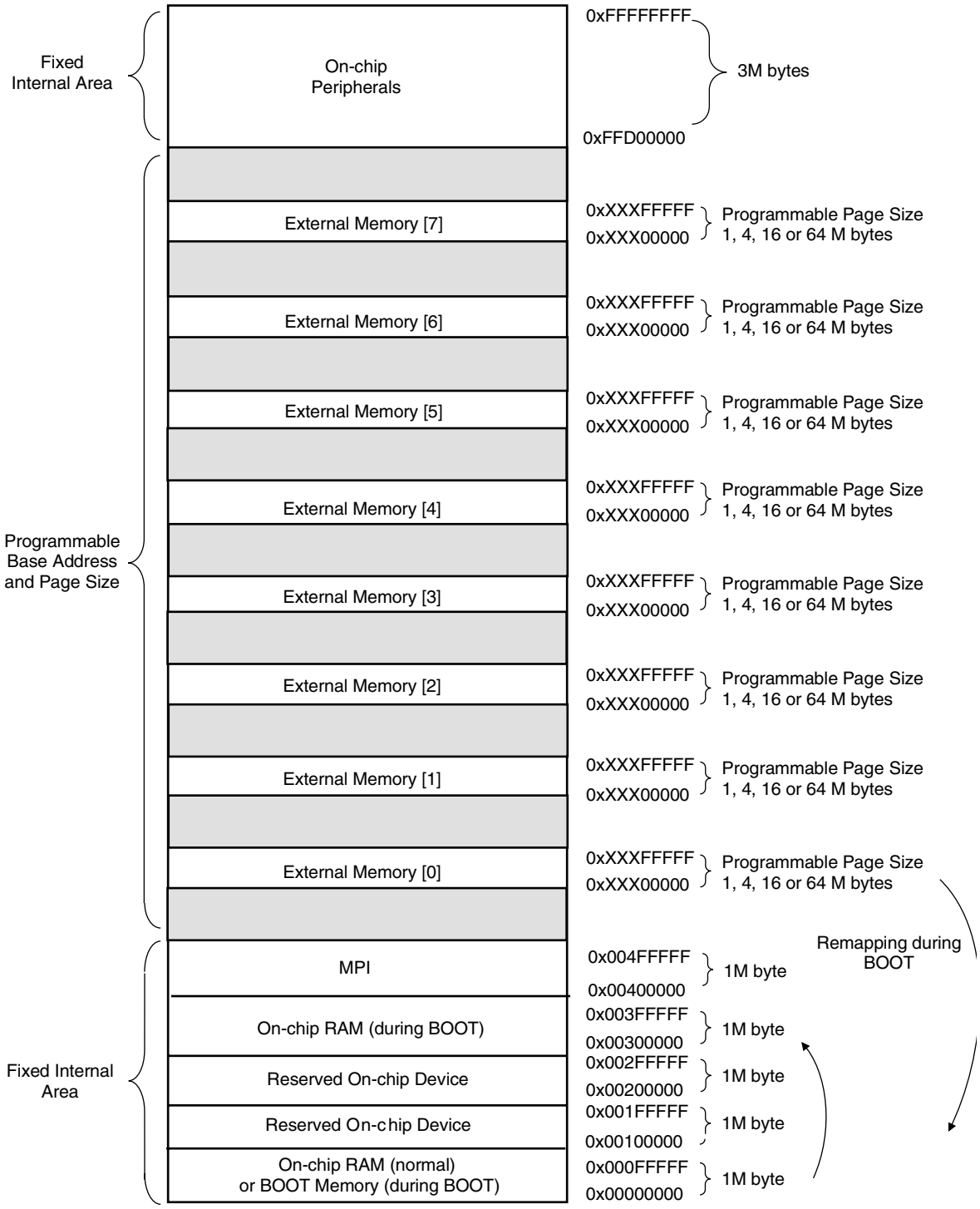
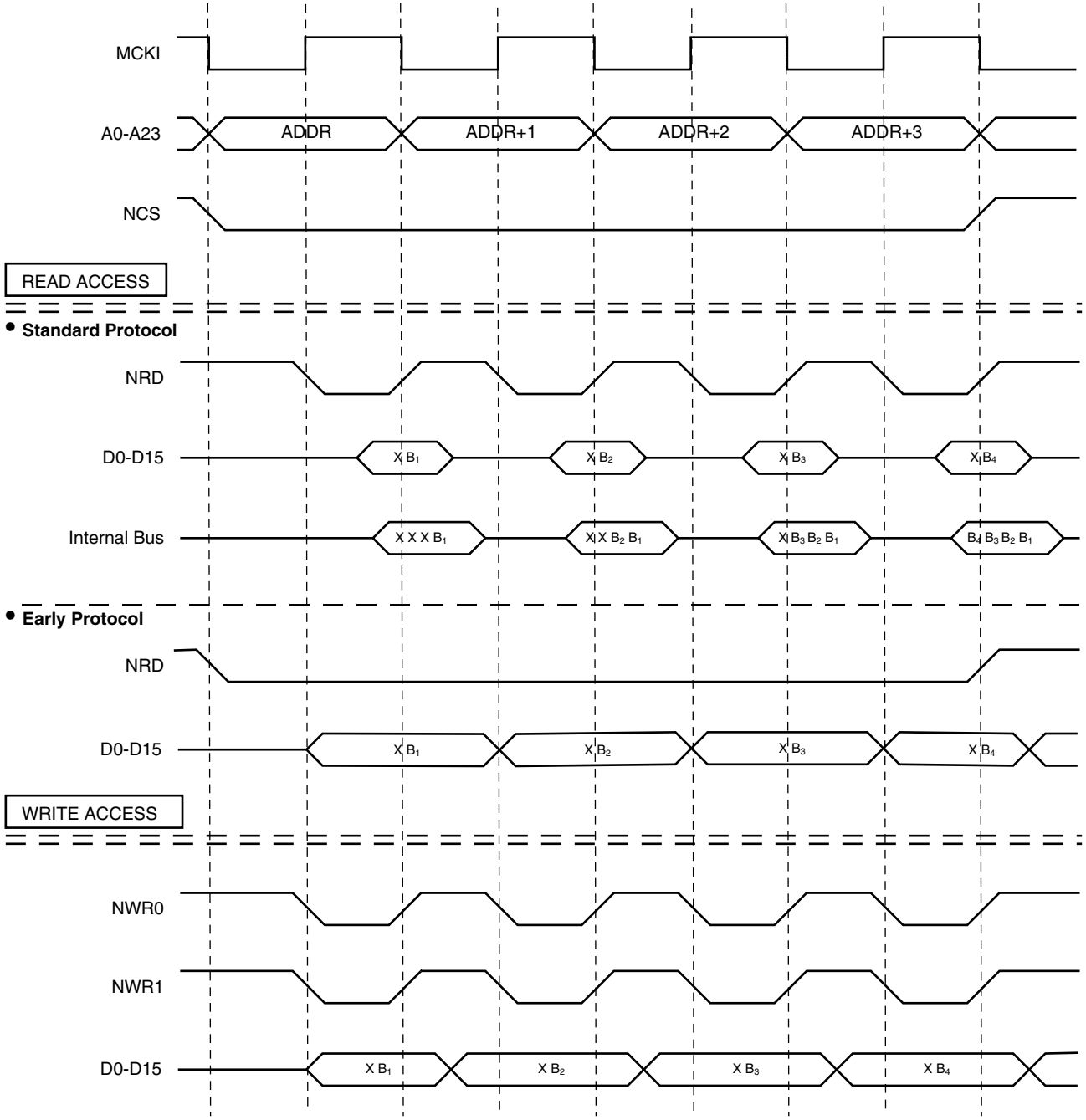


Figure 28. 0 Wait States, 8-bit Bus Width, Word Transfer



Hardware Interrupt Vectoring

The hardware interrupt vectoring reduces the number of instructions to reach the interrupt handler to only one. By storing the following instruction at address 0x00000018, the processor loads the program counter with the interrupt handler address stored in the AIC_IVR register. Execution is then vectored to the interrupt handler corresponding to the current interrupt.

```
ldrPC,[PC,# -&F20]
```

The current interrupt is the interrupt with the highest priority when the Interrupt Vector Register (AIC_IVR) is read. The value read in the AIC_IVR corresponds to the address stored in the Source Vector Register (AIC_SVR) of the current interrupt. Each interrupt source has its corresponding AIC_SVR. In order to take advantage of the hardware interrupt vectoring, it is necessary to store the address of each interrupt handler in the corresponding AIC_SVR at system initialization.

Priority Controller

The NIRQ line is controlled by an 8-level priority encoder. Each source has a programmable priority level of 7 to 0. Level 7 is the highest priority and level 0 the lowest.

When the AIC receives more than one unmasked interrupt at a time, the interrupt with the highest priority is serviced first. If both interrupts have equal priority, the interrupt with the lowest interrupt source number (see Table 7) is serviced first.

The current priority level is defined as the priority level of the current interrupt at the time the register AIC_IVR is read (the interrupt which will be serviced).

In the case when a higher priority unmasked interrupt occurs while an interrupt already exists, there are two possible outcomes depending on whether the AIC_IVR has been read.

- If the NIRQ line has been asserted but the AIC_IVR has not been read, then the processor will read the new higher priority interrupt handler address in the AIC_IVR register and the current interrupt level is updated.
- If the processor has already read the AIC_IVR, then the NIRQ line is reasserted. When the processor has authorized nested interrupts to occur and reads the AIC_IVR again, it reads the new, higher priority interrupt handler address. At the same time, the current priority value is pushed onto a first-in last-out stack and the current priority is updated to the higher priority.

When the End of Interrupt Command Register (AIC_EOICR) is written, the current interrupt level is updated with the last stored interrupt level from the stack (if any). Hence, at the end of a higher priority interrupt, the AIC returns to the previous state corresponding to the preceding lower priority interrupt which had been interrupted.

Interrupt Handling

The interrupt handler must read the AIC_IVR as soon as possible. This de-asserts the NIRQ request to the processor and clears the interrupt in case it is programmed to be edge triggered. This permits the AIC to assert the NIRQ line again when a higher priority unmasked interrupt occurs.

At the end of the interrupt service routine, the End of Interrupt Command Register (AIC_EOICR) must be written. This allows pending interrupts to be serviced.

Interrupt Masking

Each interrupt source, including FIQ, can be enabled or disabled using the command registers AIC_IECR and AIC_IDCR. The interrupt mask can be read in the read-only register AIC_IMR. A disabled interrupt does not affect the servicing of other interrupts.

Interrupt Clearing and Setting

All interrupt sources which are programmed to be edge triggered (including FIQ) can be individually set or cleared by respectively writing to the registers AIC_ISCR and AIC_ICCR. This function of the interrupt controller is available for auto-test or software debug purposes.

Fast Interrupt Request

The external FIQ line is the only source which can raise a fast interrupt request to the processor. Therefore, it has no priority controller.

The external FIQ line can be programmed to be positive- or negative-edge triggered or high- or low-level sensitive in the AIC_SMR0 register.

The fast interrupt handler address can be stored in the AIC_SVR0 register. The value written into this register is available by reading the AIC_FVR register when an FIQ interrupt is raised. By storing the following instruction at address 0x0000001C, the processor will load the program counter with the interrupt handler address stored in the AIC_FVR register.

```
ldrPC,[PC,# -&F20]
```

Alternatively, the interrupt handler can be stored starting from address 0x0000001C as described in the ARM7TDMI datasheet.

Software Interrupt

Interrupt source 1 of the advanced interrupt controller is a software interrupt. It must be programmed to be edge triggered in order to set or clear it by writing to the AIC_ISCR and AIC_ICCR.

This is totally independent of the SWI instruction of the ARM7TDMI processor.

AIC Interrupt Enable Command Register

Register Name: AIC_IECR
Access Type: Write only

31	30	29	28	27	26	25	24
IRQ0	IRQ1	IRQ2	IRQ3	---	---	---	---
23	22	21	20	19	18	17	16
---	---	---	---	---	---	---	---
15	14	13	12	11	10	9	8
---	PIOBIRQ	PIOAIRQ	WDIRQ	TC5IRQ	TC4IRQ	TC3IRQ	TC2IRQ
7	6	5	4	3	2	1	0
TC1IRQ	TC0IRQ	SPIRQ	US2IRQ	US1IRQ	US0IRQ	SWIRQ	FIQ

- **Interrupt Enable**
  0 = No effect.
  1 = Enables corresponding interrupt.

AIC Interrupt Disable Command Register

Register Name: AIC_IDCR
Access Type: Write only

31	30	29	28	27	26	25	24
IRQ0	IRQ1	IRQ2	IRQ3	---	---	---	---
23	22	21	20	19	18	17	16
---	---	---	---	---	---	---	---
15	14	13	12	11	10	9	8
---	PIOBIRQ	PIOAIRQ	WDIRQ	TC5IRQ	TC4IRQ	TC3IRQ	TC2IRQ
7	6	5	4	3	2	1	0
TC1IRQ	TC0IRQ	SPIRQ	US2IRQ	US1IRQ	US0IRQ	SWIRQ	FIQ

- **Interrupt Disable**
  0 = No effect.
  1 = Disables corresponding interrupt.

Fast Interrupt Sequence

It is assumed that:

- The Advanced Interrupt Controller has been programmed, AIC_SVR[0] is loaded with fast interrupt service routine address and the fast interrupt is enabled.
- The instruction at address 0x1C (FIQ exception vector address) is:

```
ldr pc, [pc, #-&F20]
```

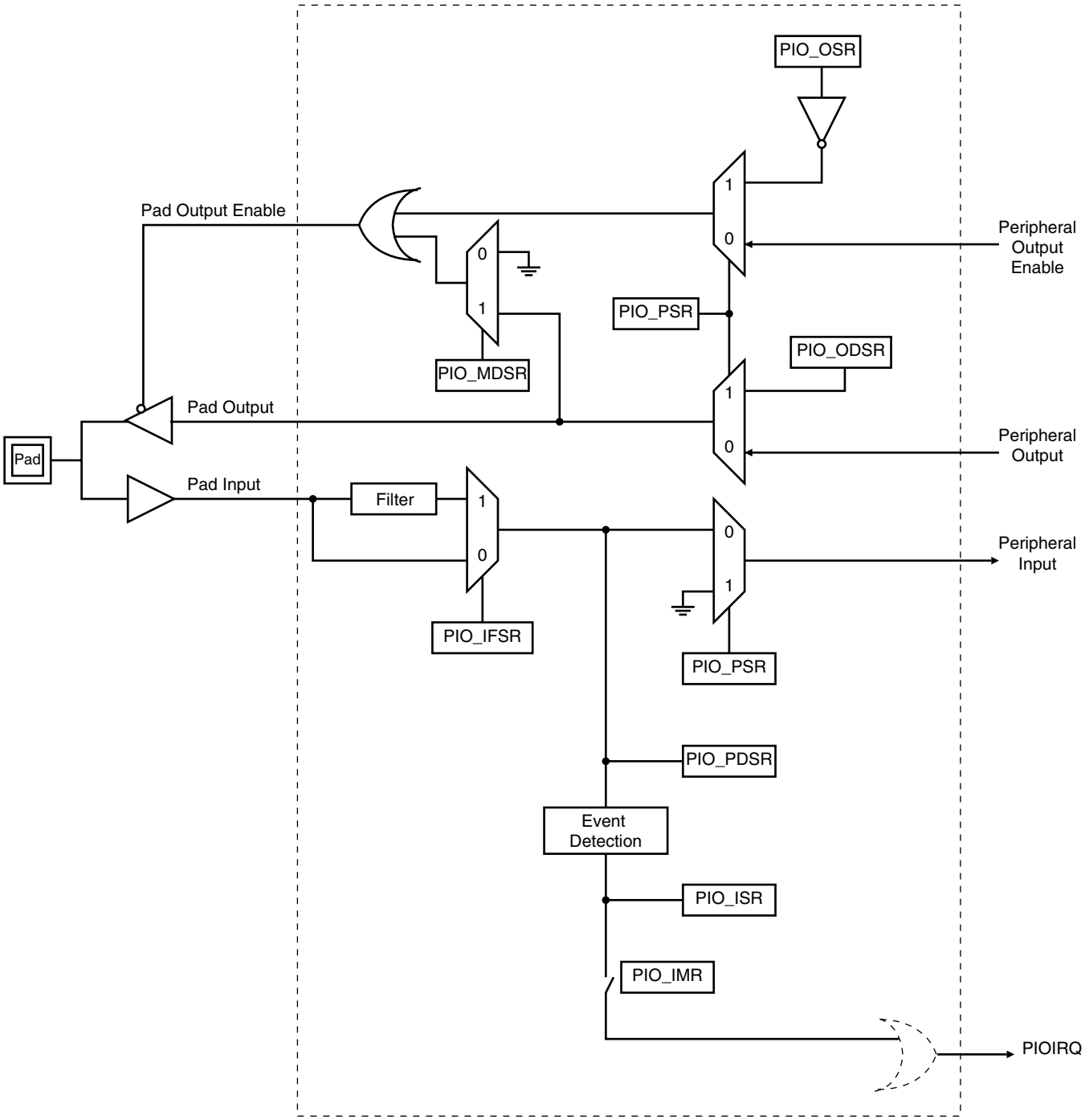
- Nested fast interrupts are not needed by the user

When NFIQ is asserted, if bit F of CPSR is 0, the sequence is:

1. The CPSR is stored in SPSR_fiq, the current value of the Program Counter is loaded in the FIQ link register (r14_fiq) and the Program Counter (r15) is loaded with 0x1C. In the following cycle, during fetch at address 0x20, the ARM core adjusts r14_fiq, decrementing it by 4.
2. The ARM core enters FIQ mode.
3. When the instruction loaded at address 0x1C is executed, the Program Counter is loaded with the value read in AIC_FVR. Reading the AIC_FVR has the effect of automatically clearing the fast interrupt (source 0 connected to the FIQ line), if it has been programmed to be edge triggered. In this case only, it de-asserts the NFIQ line on the processor.
4. The previous step has the effect of branching to the corresponding interrupt service routine. It is not necessary to save the Link Register (r14_fiq) and the SPSR (SPSR_fiq) if nested fast interrupts are not needed.
5. The Interrupt Handler can then proceed as required. It is not necessary to save registers r8 to r13 because FIQ mode has its own dedicated registers and the user r8 to r13 are banked. The other registers, r0 to r7, must be saved before being used, and restored at the end (before the next step). Note that if the fast interrupt is programmed to be level sensitive, the source of the interrupt must be cleared during this phase in order to de-assert the NFIQ line.
6. Finally, the Link Register (r14_fiq) is restored into the PC after decrementing it by 4 (with instruction sub pc, lr, #4, for example). This has the effect of returning from the interrupt to whatever was being executed before, and of loading the CPSR with the SPSR, masking or unmasking the fast interrupt depending on the state saved in the SPSR.

Note: *The F-bit in the SPSR is significant. If it is set, it indicates that the ARM core was just about to mask FIQ interrupts when the mask instruction was interrupted. Hence, when the SPSR is restored, the interrupted instruction is completed (FIQ is masked).*

Figure 38. Parallel I/O Multiplexed with a Bi-directional Signal



PIO Status Register

Register Name: PIO_PSR
Access Type: Read only
Reset Value: 0x3FFFFFFF (A)
0x0FFFFFFF (B)

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

This register indicates which pins are enabled for PIO control. This register is updated when PIO lines are enabled or disabled.

1 = PIO is active on the corresponding line (peripheral is inactive).
0 = PIO is inactive on the corresponding line (peripheral is active).

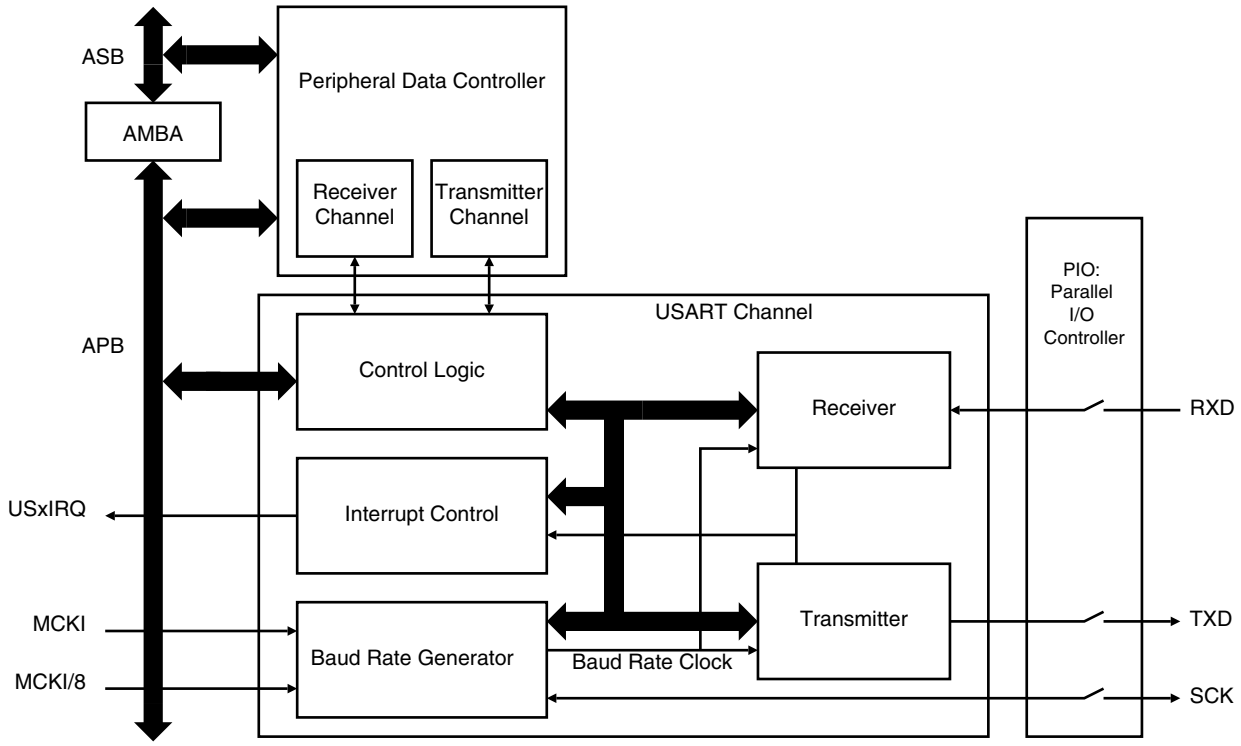
USART: Universal Synchronous/Asynchronous Receiver/Transmitter

The AT91M63200 provides three identical, full-duplex, universal synchronous/asynchronous receiver/transmitters that interface to the APB and are connected to the Peripheral Data Controller.

The main features are:

- Programmable baud rate generator
- Parity, framing and overrun error detection
- Line break generation and detection
- Automatic echo, local loopback and remote loopback channel modes
- Multi-drop mode: address detection and generation
- Interrupt generation
- Two dedicated peripheral data controller channels
- 5-, 6-, 7-, 8- and 9-bit character length

Figure 39. USART Block Diagram



Pin Description

Each USART channel has the following external signals:

Name	Description
SCK	USART serial clock can be configured as input or output: SCK is configured as input if an external clock is selected (USCLKS[1] = 1) SCK is driven as output if the external clock is disabled (USCLKS[1] = 0) and clock output is enabled (CLKO = 1)
TXD	Transmit Serial Data is an output
RXD	Receive Serial Data is an input

Note: *After a hardware reset, the USART clock is disabled by default (see "PMC: Power Management Controller" on page 139). The user must configure the Power Management Controller before any access to the user interface of the USART.*

Note: *After a hardware reset, the USART pins are deselected by default (see "PIO: Parallel I/O Controller" on page 55). The user must configure the PIO Controller before enabling the transmitter or receiver.*

If the user selects one of the internal clocks, SCK can be configured as a PIO.

Break

A break condition is a low signal level which has a duration of at least one character (including start/stop bits and parity).

Transmit Break

The transmitter generates a break condition on the TXD line when STTBRK is set in US_CR (Control Register). In this case, the character present in the Transmit Shift Register is completed before the line is held low.

To cancel a break condition on the TXD line, the STPBRK command in US_CR must be set. The USART completes a minimum break duration of one character length. The TXD line then returns to high level (idle state) for at least 12 bit periods to ensure that the end of break is correctly detected. Then the transmitter resumes normal operation.

The BREAK is managed like a character:

- The STTBRK and the STPBRK commands are performed only if the transmitter is ready (bit TXRDY = 1 in US_CSR).
- The STTBRK command blocks the transmitter holding register (bit TXRDY is cleared in US_CSR) until the break has started.
- A break is started when the Shift Register is empty (any previous character is fully transmitted). US_CSR.TXEMPTY is cleared. The break blocks the transmitter shift register until it is completed (high level for at least 12 bit periods after the STPBRK command is requested).

In order to avoid unpredictable states:

- STTBRK and STPBRK commands must not be requested at the same time.
- Once an STTBRK command is requested, further STTBRK commands are ignored until the BREAK is ended (high level for at least 12 bit periods).
- All STPBRK commands requested without a previous STTBRK command are ignored.
- A byte written into the Transmit Holding Register while a break is pending but not started (bit TXRDY = 0 in US_CSR) is ignored.
- It is *not permitted* to write new data in the Transmit Holding Register while a break is in progress (STPBRK has not been requested), even though TXRDY = 1 in US_CSR.
- A new STTBRK command *must not* be issued until an existing break has ended (TXEMPTY = 1 in US_CSR).

The standard break transmission sequence is:

1. Wait for the transmitter ready (US_CSR.TXRDY = 1).
2. Send the STTBRK command (write 0x0200 to US_CR).
3. Wait for the transmitter ready (bit TXRDY = 1 in US_CSR).
4. Send the STPBRK command (write 0x0400 to US_CR).

The next byte can then be sent:

5. Wait for the transmitter ready (bit TXRDY = 1 in US_CSR).
6. Send the next byte (write byte to US_THR).

Each of these steps can be scheduled by using the interrupt if the bit TXRDY in US_IMR is set.

For character transmission, the USART channel must be enabled before sending a break.

Receive Break

The receiver detects a break condition when all data, parity and stop bits are low. When the low stop bit is detected, the receiver asserts the RXBRK bit in US_CSR. An end-of-receive break is detected by a high level for at least 2/16 of a bit period in asynchronous operating mode or at least one sample in synchronous operating mode. RXBRK is also asserted when an end-of-break is detected.

Both the beginning and the end of a break can be detected by interrupt if the bit US_IMR.RXBRK is set.

Peripheral Data Controller

Each USART channel is closely connected to a corresponding Peripheral Data Controller channel. One is dedicated to the receiver. The other is dedicated to the transmitter.

Note: *The PDC is disabled if 9-bit character length is selected (MODE9 = 1) in US_MR.*

The PDC channel is programmed using US_TPR (Transmit Pointer) and US_TCR (Transmit Counter) for the transmitter and US_RPR (Receive Pointer) and US_RCR (Receive Counter) for the receiver. The status of the PDC is given in US_CSR by the ENDTX bit for the transmitter and by the ENDRX bit for the receiver.

The pointer registers (US_TPR and US_RPR) are used to store the address of the transmit or receive buffers. The counter registers (US_TCR and US_RCR) are used to store the size of these buffers.

The receiver data transfer is triggered by the RXRDY bit and the transmitter data transfer is triggered by TXRDY. When a transfer is performed, the counter is decremented and the pointer is incremented. When the counter reaches 0, the status bit is set (ENDRX for the receiver, ENDTX for the transmitter in US_CSR) and can be programmed to generate an interrupt. Transfers are then disabled until a new non-zero counter value is programmed.

USART Control Register

Name: US_CR
Access Type: Write only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	SENDA	STTTO	STPBRK	STTBRK	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	–	–

- **RSTRX: Reset Receiver**
  0 = No effect.
  1 = The receiver logic is reset.
- **RSTTX: Reset Transmitter**
  0 = No effect.
  1 = The transmitter logic is reset.
- **RXEN: Receiver Enable**
  0 = No effect.
  1 = The receiver is enabled if RXDIS is 0.
- **RXDIS: Receiver Disable**
  0 = No effect.
  1 = The receiver is disabled.
- **TXEN: Transmitter Enable**
  0 = No effect.
  1 = The transmitter is enabled if TXDIS is 0.
- **TXDIS: Transmitter Disable**
  0 = No effect.
  1 = The transmitter is disabled.
- **RSTSTA: Reset Status Bits**
  0 = No effect.
  1 = Resets the status bits PARE, FRAME, OVRE and RXBRK in the US_CSR.
- **STTBRK: Start Break**
  0 = No effect.
  1 = If break is not being transmitted, start transmission of a break after the characters present in US_THR and the Transmit Shift Register have been transmitted.
- **STPBRK: Stop Break**
  0 = No effect.
  1 = If a break is being transmitted, stop transmission of the break after a minimum of one character length and transmit a high level during 12 bit periods.
- **STTTO: Start Time-out**
  0 = No effect.
  1 = Start waiting for a character before clocking the time-out counter.
- **SENDA: Send Address**
  0 = No effect.
  1 = In multi-drop mode only, the next character written to the US_THR is sent with the address bit set.

USART Interrupt Disable Register

Name: US_IDR
Access Type: Write only

31	30	29	28	27	26	25	24
COMMRX	COMMTX	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

- **RXRDY: Disable RXRDY Interrupt**
  0 = No effect.
  1 = Disables RXRDY Interrupt.
- **TXRDY: Disable TXRDY Interrupt**
  0 = No effect.
  1 = Disables TXRDY Interrupt.
- **RXBRK: Disable Receiver Break Interrupt**
  0 = No effect.
  1 = Disables Receiver Break Interrupt.
- **ENDRX: Disable End of Receive Transfer Interrupt**
  0 = No effect.
  1 = Disables End of Receive Transfer Interrupt.
- **ENDTX: Disable End of Transmit Transfer Interrupt**
  0 = No effect.
  1 = Disables End of Transmit Transfer Interrupt.
- **OVRE: Disable Overrun Error Interrupt**
  0 = No effect.
  1 = Disables Overrun Error Interrupt.
- **FRAME: Disable Framing Error Interrupt**
  0 = No effect.
  1 = Disables Framing Error Interrupt.
- **PARE: Disable Parity Error Interrupt**
  0 = No effect.
  1 = Disables Parity Error Interrupt.
- **TIMEOUT: Disable Time-out Interrupt**
  0 = No effect.
  1 = Disables Receiver Time-out Interrupt.
- **TXEMPTY: Disable TXEMPTY Interrupt**
  0 = No effect.
  1 = Disables TXEMPTY Interrupt.
- **COMMTX: Disable ARM7TDMI ICE Debug Communication Channel Transmit Interrupt**
  This bit is implemented for USART0 only.
  0 = No effect.
  1 = Disables COMMTX Interrupt.
- **COMMRX: Disable ARM7TDMI ICE Debug Communication Channel Receive Interrupt**
  This bit is implemented for USART0 only.
  0 = No effect.
  1 = Disables COMMRX Interrupt.

USART Receive Pointer Register

Name: US_RPR
Access Type: Read/Write

31	30	29	28	27	26	25	24
RXPTR							
23	22	21	20	19	18	17	16
RXPTR							
15	14	13	12	11	10	9	8
RXPTR							
7	6	5	4	3	2	1	0
RXPTR							

- **RXPTR: Receive Pointer**
  RXPTR must be loaded with the address of the receive buffer.

USART Receive Counter Register

Name: US_RCR
Access Type: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
RXCTR							
7	6	5	4	3	2	1	0
RXCTR							

- **RXCTR: Receive Counter**
  RXCTR must be loaded with the size of the receive buffer.
  0: Stop peripheral data transfer dedicated to the receiver.
  1-65535: Start peripheral data transfer if RXRDY is active.

Master Mode

In master mode, the SPI controls data transfers to and from the slave(s) connected to the SPI bus. The SPI drives the chip select(s) to the slave(s) and the serial clock (SPCK). After enabling the SPI, a data transfer begins when the ARM core writes to the SP_TDR (Transmit Data Register). See Table 13.

Transmit and receive buffers maintain the data flow at a constant rate with a reduced requirement for high-priority interrupt servicing. When new data is available in the SP_TDR (Transmit Data Register) the SPI continues to transfer data. If the SP_RDR (Receive Data Register) has not been read before new data is received, the Overrun Error (OVRES) flag is set.

The delay between the activation of the chip select and the start of the data transfer (DLYBS), as well as the delay between each data transfer (DLYBCT), can be programmed for each of the four external chip selects. All data transfer characteristics including the two timing values are programmed in registers SP_CSR0 to SP_CSR3 (Chip Select Registers). See Table 13.

In master mode, the peripheral selection can be defined in two different ways:

- Fixed Peripheral Select: SPI exchanges data with only one peripheral
- Variable Peripheral Select: Data can be exchanged with more than one peripheral

Figures 47 and 48 show the operation of the SPI in master mode. For details concerning the flag and control bits in these diagrams, see the tables in the “Programmer’s Model”, starting on page 99.

Fixed Peripheral Select

This mode is ideal for transferring memory blocks without the extra overhead in the transmit data register to determine the peripheral.

Fixed Peripheral Select is activated by setting bit PS to zero in SP_MR (Mode Register). The peripheral is defined by the PCS field, also in SP_MR.

This option is only available when the SPI is programmed in master mode.

Variable Peripheral Select

Variable Peripheral Select is activated by setting bit PS to one. The PCS field in SP_TDR (Transmit Data Register) is used to select the destination peripheral. The data transfer characteristics are changed when the selected peripheral changes, according to the associated chip select register.

The PCS field in the SP_MR has no effect.

This option is only available when the SPI is programmed in master mode.

Chip Selects

The chip select lines are driven by the SPI only if it is programmed in master mode. These lines are used to select the destination peripheral. The PCSDEC field in SP_MR (Mode Register) selects 1 to 4 peripherals (PCSDEC = 0) or up to 15 peripherals (PCSDEC = 1).

If Variable Peripheral Select is active, the chip select signals are defined for each transfer in the PCS field in SP_TDR. Chip select signals can thus be defined independently for each transfer.

If Fixed Peripheral Select is active, chip select signals are defined for all transfers by the field PCS in SP_MR. If a transfer with a new peripheral is necessary, the software must wait until the current transfer is completed, then change the value of PCS in SP_MR before writing new data in SP_TDR.

The value on the NPCS pins at the end of each transfer can be read in the SP_RDR (Receive Data Register).

By default, all NPCS signals are high (equal to one) before and after each transfer.

Mode Fault Detection

A mode fault is detected when the SPI is programmed in master mode and a low level is driven by an external master on the NPCS0/NSS signal.

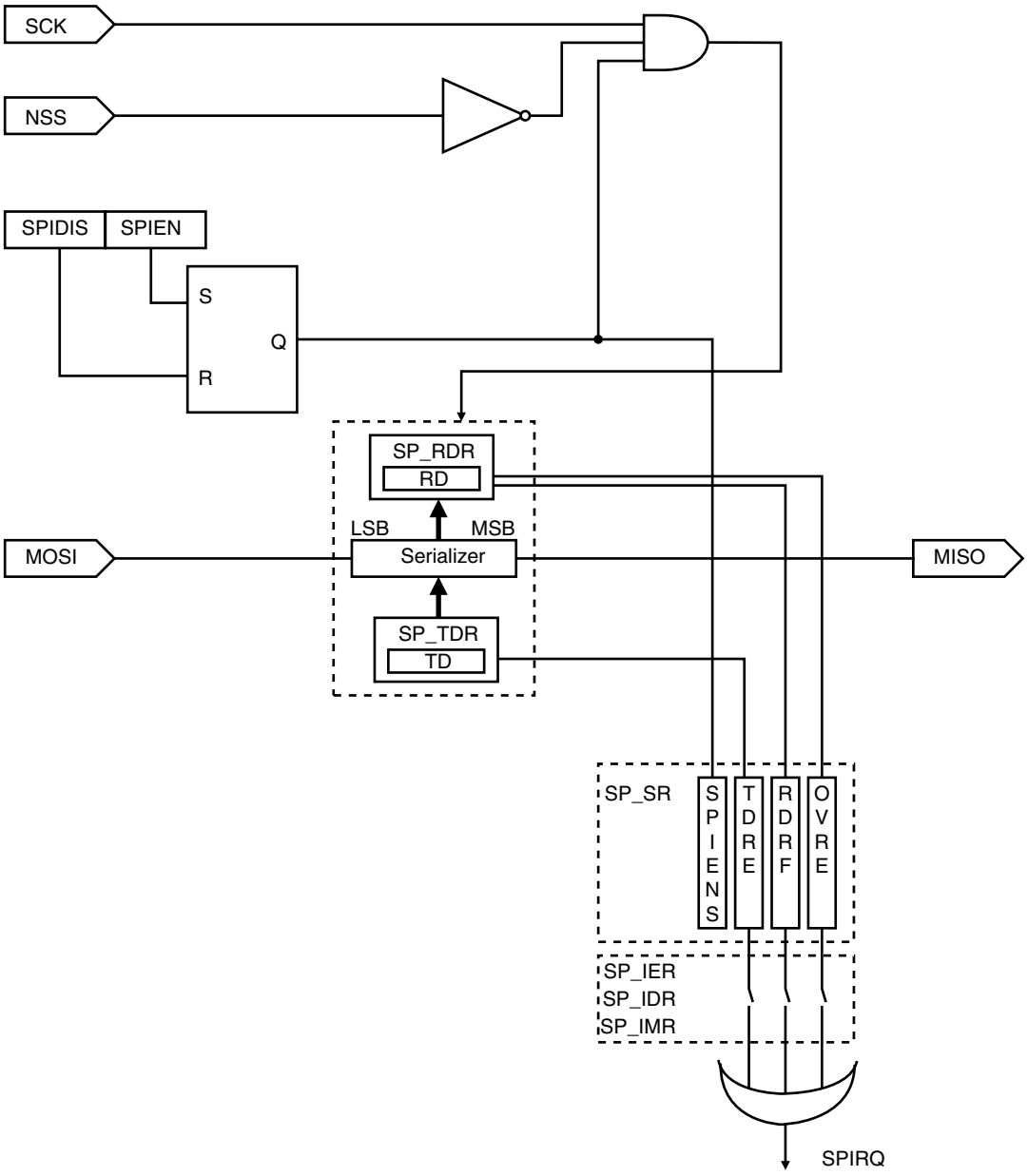
When a mode fault is detected, the MODF bit in the SP_SR is set until the SP_SR is read and the SPI is disabled until re-enabled by bit SPIEN in the SP_CR (Control Register).

Slave Mode

In slave mode, the SPI waits for NSS to go active low before receiving the serial clock from an external master.

In slave mode, CPOL, NCPHA and BITS fields of SP_CSR0 are used to define the transfer characteristics. The other chip select registers are not used in slave mode.

Figure 49. SPI in Slave Mode



SPI Status Register

Register Name: SP_SR
Access Type: Read only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	SPIENS
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
–	–	SPENDTX	SPENDRX	OVRES	MODF	TDRE	RDRF

- **RDRF: Receive Data Register Full**
  0 = No data has been received since the last read of SP_RDR.
  1 = Data has been received and the received data has been transferred from the serializer to SP_RDR since the last read of SP_RDR.
- **TDRE: Transmit Data Register Empty**
  0 = Data has been written to SP_TDR and not yet transferred to the serializer.
  1 = The last data written in the Transmit Data Register has been transferred in the serializer.
  TDRE equals zero when the SPI is disabled or at reset. The SPI enable command sets this bit to one.
- **MODF: Mode Fault Error**
  0 = No mode fault has been detected since the last read of SP_SR.
  1 = A mode fault occurred since the last read of the SP_SR.
- **OVRES: Overrun Error Status**
  0 = No overrun has been detected since the last read of SP_SR.
  1 = An overrun has occurred since the last read of SP_SR.
  An overrun occurs when SP_RDR is loaded at least twice from the serializer since the last read of the SP_RDR.
- **SPENDRX: End of Receiver Transfer**
  0 = The End of Transfer signal from the Peripheral Data Controller channel dedicated to the receiver is inactive.
  1 = The End of Transfer signal from the Peripheral Data Controller channel dedicated to the receiver is active.
- **SPENDTX: End of Transmitter Transfer**
  0 = The End of Transfer signal from the Peripheral Data Controller channel dedicated to the transmitter is inactive.
  1 = The End of Transfer signal from the Peripheral Data Controller channel dedicated to the transmitter is active.
- **SPIENS: SPI Enable Status**
  0 = SPI is disabled.
  1 = SPI is enabled.

TC: Timer/Counter

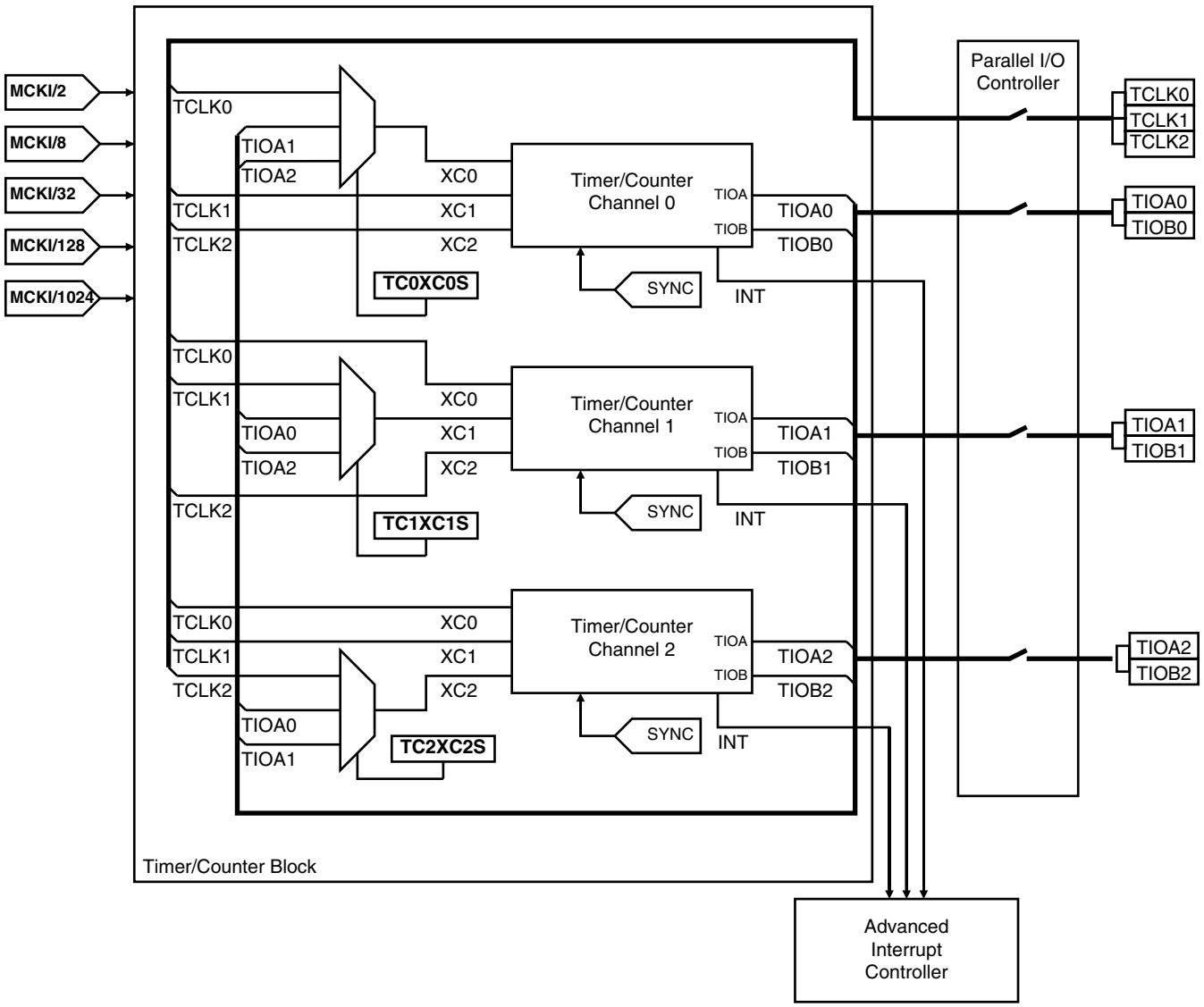
The AT91M63200 features two Timer/Counter blocks, each containing three identical 16-bit Timer/Counter channels. Each channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse width modulation.

Each Timer/Counter channel has three external clock inputs, five internal clock inputs, and two multi-purpose input/output signals which can be configured by the user. Each channel drives an internal interrupt signal which can be programmed to generate processor interrupts via the AIC (Advanced Interrupt Controller).

The Timer/Counter block has two global registers which act upon all three TC channels. The Block Control Register allows the three channels to be started simultaneously with the same instruction. The Block Mode Register defines the external clock inputs for each Timer/Counter channel, allowing them to be chained.

Each Timer/Counter block operates independently and has a complete set of block and channel registers. Since they are identical in operation, only one block is described below (see "Timer/Counter Description" on page 113). The internal configuration of a single Timer/Counter block is shown in Figure 53.

Figure 53. TC Block Diagram



TC Counter Value Register

Register Name: TC_CVR
Access Type: Read only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
CV							
7	6	5	4	3	2	1	0
CV							

- **CV: Counter Value**
  CV contains the counter value in real time.

TC Register A

Register Name: TC_RA
Access Type: Read only if WAVE = 0, Read/Write if WAVE = 1

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
RA							
7	6	5	4	3	2	1	0
RA							

- **RA: Register A**
  RA contains the Register A value in real time.

PMC System Clock Status Register

Register Name: PMC_SCSR
Access Type: Read only

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
–	–	–	–	–	–	–	CPU

- **CPU: CPU Clock Status**
  0 = CPU clock is enabled.
  1 = CPU clock is disabled.

SF: Special Function Registers

The M63X00 provides registers which implement the following special functions:

- Chip identification: a chip identifier module which enables software to recognize certain characteristics of the chip and the version number
- RESET status
- Protect mode (see “Protect Mode” on page 42)

SF User Interface

Chip ID Base Address: 0xFFF00000

Table 18. SF Memory Map

Offset	Register	Name	Access	Reset State
0x00	Chip ID Register	SF_CIDR	Read only	Hardwired
0x04	Chip ID Extension Register	SF_EXID	Read only	Hardwired
0x08	Reset Status Register	SF_RSR	Read only	See register description
0x0C	Reserved	–	–	–
0x10	Reserved	–	–	–
0x14	Reserved	–	–	–
0x18	Protect Mode Register	SF_PMR	Read/Write	0x0

Chip ID Register

Register Name: SF_CIDR
Access Type: Read only

31	30	29	28	27	26	25	24
EXT	NVPTYP			ARCH			

23	22	21	20	19	18	17	16
ARCH				VDSIZ			

15	14	13	12	11	10	9	8
NVDSIZ				NVPSIZ			

7	6	5	4	3	2	1	0
0	1	0	VERSION				

- **VERSION: Version of the chip**
  This value is incremented by one with each new version of the chip (from zero to a maximum value of 31).