



Welcome to [E-XFL.COM](https://www.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	55MHz
Connectivity	CANbus, Ethernet, I²C, SPI, SSC, UART/USART, USB
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	62
Program Memory Size	256KB (256K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	64K x 8
Voltage - Supply (Vcc/Vdd)	1.65V ~ 1.95V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	100-LQFP
Supplier Device Package	100-LQFP (14x14)
Purchase URL	<a href="https://www.e-xfl.com/product-detail/microchip-technology/at91sam7x256-au">https://www.e-xfl.com/product-detail/microchip-technology/at91sam7x256-au</a>

## 8.4 Memory Mapping

### 8.4.1 Internal SRAM

- The SAM7X512 embeds a high-speed 128-Kbyte SRAM bank.
- The SAM7X256 embeds a high-speed 64-Kbyte SRAM bank.
- The SAM7X128 embeds a high-speed 32-Kbyte SRAM bank.

After reset and until the Remap Command is performed, the SRAM is only accessible at address 0x0020 0000. After Remap, the SRAM also becomes available at address 0x0.

### 8.4.2 Internal ROM

The SAM7X512/256/128 embeds an Internal ROM. At any time, the ROM is mapped at address 0x30 0000. The ROM contains the FFPI and the SAM-BA program.

### 8.4.3 Internal Flash

- The SAM7X512 features two banks (dual plane) of 256 Kbytes of Flash.
- The SAM7X256 features one bank (single plane) of 256 Kbytes of Flash.
- The SAM7X128 features one bank (single plane) of 128 Kbytes of Flash.

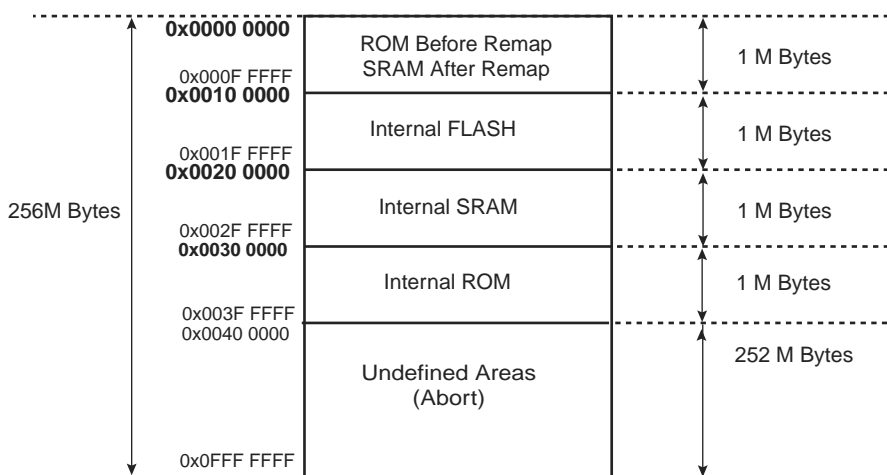
At any time, the Flash is mapped to address 0x0010 0000. It is also accessible at address 0x0 after the reset, if GPNVM bit 2 is set and before the Remap Command.

A general purpose NVM (GPNVM) bit is used to boot either on the ROM (default) or from the Flash.

This GPNVM bit can be cleared or set respectively through the commands “Clear General-purpose NVM Bit” and “Set General-purpose NVM Bit” of the EFC User Interface.

Setting the GPNVM Bit 2 selects the boot from the Flash. Asserting ERASE clears the GPNVM Bit 2 and thus selects the boot from the ROM by default.

**Figure 8-2. Internal Memory Mapping with GPNVM Bit 2 = 0 (default)**

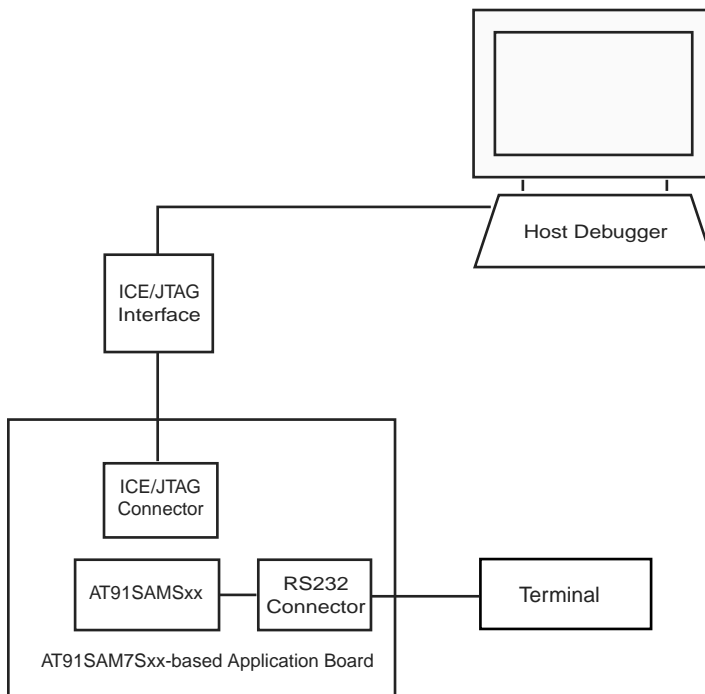


## 12.3 Application Examples

### 12.3.1 Debug Environment

Figure 12-2 shows a complete debug environment example. The ICE/JTAG interface is used for standard debugging functions, such as downloading code and single-stepping through the program.

**Figure 12-2. Application Debug Environment Example**



Each SAM7X input/output pin corresponds to a 3-bit register in the BSR. The OUTPUT bit contains data that can be forced on the pad. The INPUT bit facilitates the observability of data applied to the pad. The CONTROL bit selects the direction of the pad.

**Table 12-2. SAM7X JTAG Boundary Scan Register**

Bit Number	Pin Name	Pin Type	Associated BSR Cells
187	PA30/IRQ0/PCK2	IN/OUT	INPUT
186			OUTPUT
185			CONTROL
184	PA0/RXD0	IN/OUT	INPUT
183			OUTPUT
182			CONTROL
181	PA1/TXD0	IN/OUT	INPUT
180			OUTPUT
179			CONTROL
178	PA3/RTS0/SPI1_NPCS2	IN/OUT	INPUT
177			OUTPUT
176			CONTROL
175	PA2/SCK0/SPI1_NPCS1	IN/OUT	INPUT
174			OUTPUT
173			CONTROL
172	PA4/CTS0/SPI1_NPCS3	IN/OUT	INPUT
171			OUTPUT
170			CONTROL
169	PA5/RXD1	IN/OUT	INPUT
168			OUTPUT
167			CONTROL
166	PA6/TXD1	IN/OUT	CONTROL
165			INPUT
164			OUTPUT
163	PA7/SCK1/SPI0_NPCS1	IN/OUT	CONTROL
162			INPUT
161			OUTPUT
160	ERASE	IN	INPUT
159	PB27/TIOA2/PWM0/AD0	IN/OUT	INPUT
158			OUTPUT
157			CONTROL
156	PB28/TIOB2/PWM1/AD1	IN/OUT	INPUT
155			OUTPUT
154			CONTROL

### 16.4.3 Watchdog Timer Status Register

Register Name: WDT\_SR

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
–	–	–	–	–	–	WDERR	WDUNF

- **WDUNF: Watchdog Underflow**

0: No Watchdog underflow occurred since the last read of WDT\_SR.

1: At least one Watchdog underflow occurred since the last read of WDT\_SR.

- **WDERR: Watchdog Error**

0: No Watchdog error occurred since the last read of WDT\_SR.

1: At least one Watchdog error occurred since the last read of WDT\_SR.

### 23.7.5 Protect Mode

The Protect Mode permits reading the Interrupt Vector Register without performing the associated automatic operations. This is necessary when working with a debug system. When a debugger, working either with a Debug Monitor or the ARM processor's ICE, stops the applications and updates the opened windows, it might read the AIC User Interface and thus the IVR. This has undesirable consequences:

- If an enabled interrupt with a higher priority than the current one is pending, it is stacked.
- If there is no enabled pending interrupt, the spurious vector is returned.

In either case, an End of Interrupt command is necessary to acknowledge and to restore the context of the AIC. This operation is generally not performed by the debug system as the debug system would become strongly intrusive and cause the application to enter an undesired state.

This is avoided by using the Protect Mode. Writing PROT in AIC\_DCR (Debug Control Register) at 0x1 enables the Protect Mode.

When the Protect Mode is enabled, the AIC performs interrupt stacking only when a write access is performed on the AIC\_IVR. Therefore, the Interrupt Service Routines must write (arbitrary data) to the AIC\_IVR just after reading it. The new context of the AIC, including the value of the Interrupt Status Register (AIC\_ISR), is updated with the current interrupt only when AIC\_IVR is written.

An AIC\_IVR read on its own (e.g., by a debugger), modifies neither the AIC context nor the AIC\_ISR. Extra AIC\_IVR reads perform the same operations. However, it is recommended to not stop the processor between the read and the write of AIC\_IVR of the interrupt service routine to make sure the debugger does not modify the AIC context.

To summarize, in normal operating mode, the read of AIC\_IVR performs the following operations within the AIC:

1. Calculates active interrupt (higher than current or spurious).
2. Determines and returns the vector of the active interrupt.
3. Memorizes the interrupt.
4. Pushes the current priority level onto the internal stack.
5. Acknowledges the interrupt.

However, while the Protect Mode is activated, only operations 1 to 3 are performed when AIC\_IVR is read. Operations 4 and 5 are only performed by the AIC when AIC\_IVR is written.

Software that has been written and debugged using the Protect Mode runs correctly in Normal Mode without modification. However, in Normal Mode the AIC\_IVR write has no effect and can be removed to optimize the code.

### 23.7.6 Spurious Interrupt

The Advanced Interrupt Controller features protection against spurious interrupts. A spurious interrupt is defined as being the assertion of an interrupt source long enough for the AIC to assert the nIRQ, but no longer present when AIC\_IVR is read. This is most prone to occur when:

- An external interrupt source is programmed in level-sensitive mode and an active level occurs for only a short time.
- An internal interrupt source is programmed in level sensitive and the output signal of the corresponding embedded peripheral is activated for a short time. (As in the case for the Watchdog.)
- An interrupt occurs just a few cycles before the software begins to mask it, thus resulting in a pulse on the interrupt source.

The AIC detects a spurious interrupt at the time the AIC\_IVR is read while no enabled interrupt source is pending. When this happens, the AIC returns the value stored by the programmer in AIC\_SPU (Spurious Vector Register). The programmer must store the address of a spurious interrupt handler in AIC\_SPU as part of the application, to enable as fast as possible return to the normal execution flow. This handler writes in AIC\_EOICR and performs a return from interrupt.

### 23.8.16 AIC Spurious Interrupt Vector Register

**Register Name:** AIC\_SPU  
**Access Type:** Read-write  
**Reset Value:** 0

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

- **SIVR: Spurious Interrupt Vector Register**

The user may store the address of a spurious interrupt handler in this register. The written value is returned in AIC\_IVR in case of a spurious interrupt and in AIC\_FVR in case of a spurious fast interrupt.

### 23.8.17 AIC Debug Control Register

**Register Name:** AIC\_DEBUG  
**Access Type:** Read-write  
**Reset Value:** 0

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
–	–	–	–	–	–	GMSK	PROT

- **PROT: Protection Mode**

0 = The Protection Mode is disabled.

1 = The Protection Mode is enabled.

- **GMSK: General Mask**

0 = The nIRQ and nFIQ lines are normally controlled by the AIC.

1 = The nIRQ and nFIQ lines are tied to their inactive state.

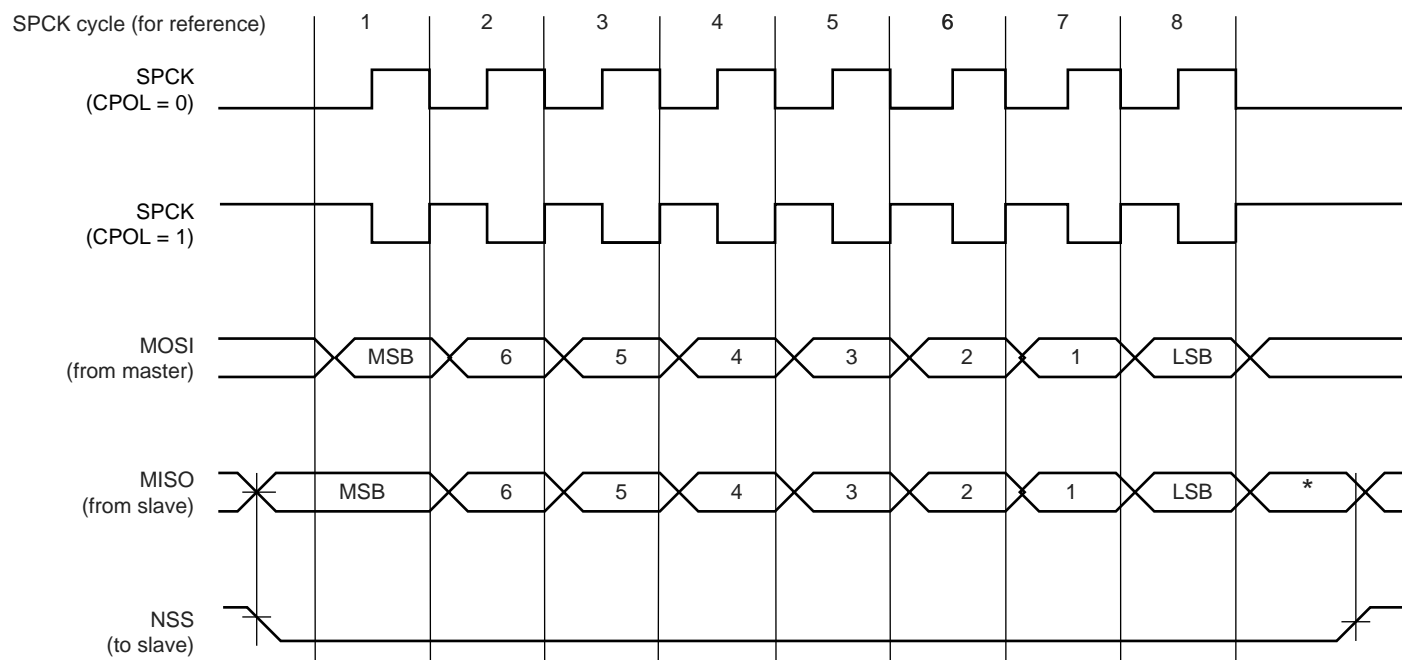
## 26.5 Debug Unit (DBGU) User Interface

Table 26-2. Register Mapping

Offset	Register	Name	Access	Reset
0x0000	Control Register	DBGU_CR	Write-only	–
0x0004	Mode Register	DBGU_MR	Read-write	0x0
0x0008	Interrupt Enable Register	DBGU_IER	Write-only	–
0x000C	Interrupt Disable Register	DBGU_IDR	Write-only	–
0x0010	Interrupt Mask Register	DBGU_IMR	Read-only	0x0
0x0014	Status Register	DBGU_SR	Read-only	–
0x0018	Receive Holding Register	DBGU_RHR	Read-only	0x0
0x001C	Transmit Holding Register	DBGU_THR	Write-only	–
0x0020	Baud Rate Generator Register	DBGU_BRGR	Read-write	0x0
0x0024 - 0x003C	Reserved	–	–	–
0x0040	Chip ID Register	DBGU_CIDR	Read-only	–
0x0044	Chip ID Extension Register	DBGU_EXID	Read-only	–
0x0048	Force NTRST Register	DBGU_FNR	Read-write	0x0
0x004C - 0x00FC	Reserved	–	–	–
0x0100 - 0x0124	PDC Area	–	–	–

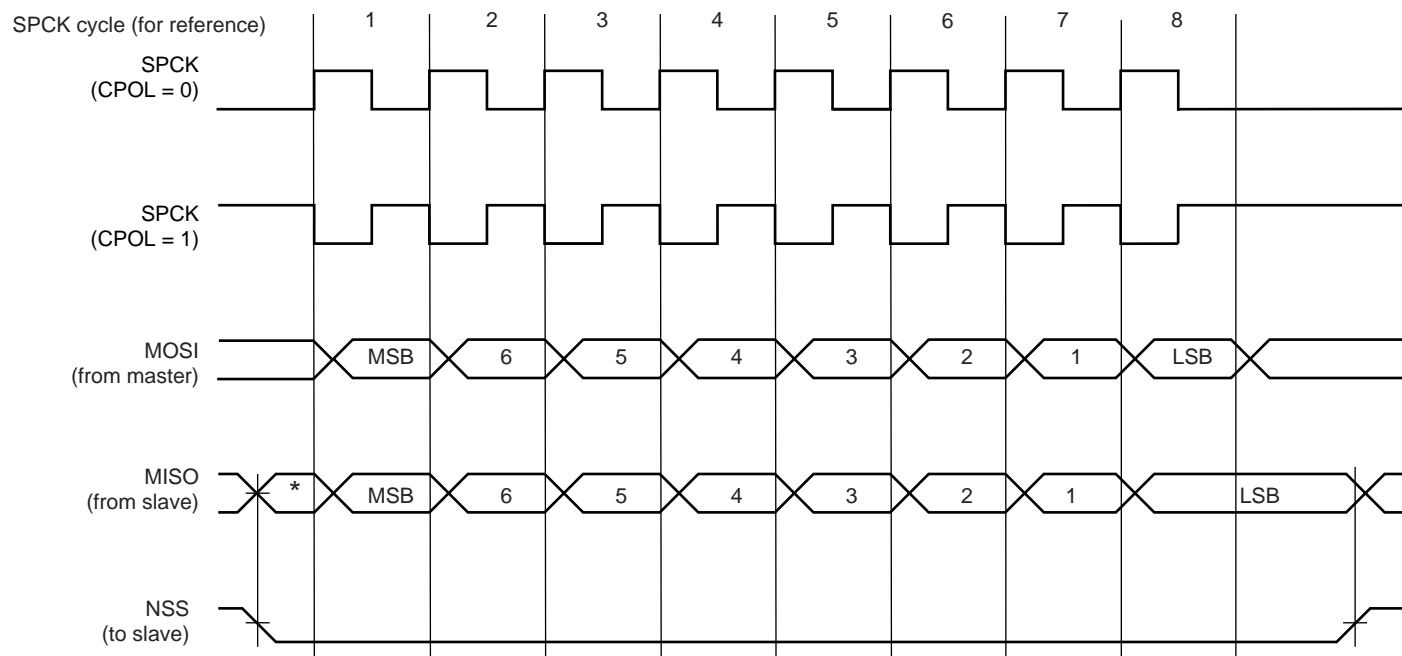


**Figure 28-3. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)**



\* Not defined, but normally MSB of previous character received.

**Figure 28-4. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)**



\* Not defined but normally LSB of previous character transmitted.

BITS	Bits Per Transfer
1101	Reserved
1110	Reserved
1111	Reserved

- **SCBR: Serial Clock Baud Rate**

In Master Mode, the SPI Interface uses a modulus counter to derive the SPCK baud rate from the Master Clock MCK. The Baud rate is selected by writing a value from 1 to 255 in the SCBR field. The following equations determine the SPCK baud rate:

$$\text{SPCK Baudrate} = \frac{MCK}{SCBR}$$

Programming the SCBR field at 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

- **DLYBS: Delay Before SPCK**

This field defines the delay from NPCS valid to the first valid SPCK transition.

When DLYBS equals zero, the NPCS valid to SPCK transition is 1/2 the SPCK clock period.

Otherwise, the following equations determine the delay:

$$\text{Delay Before SPCK} = \frac{DLYBS}{MCK}$$

- **DLYBCT: Delay Between Consecutive Transfers**

This field defines the delay between two consecutive transfers with the same peripheral without removing the chip select. The delay is always inserted after each transfer and before removing the chip select if needed.

When DLYBCT equals zero, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers.

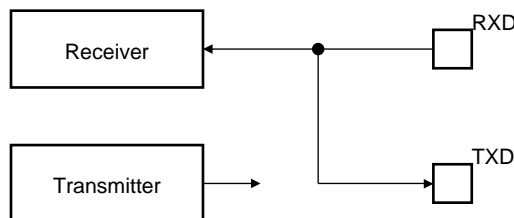
Otherwise, the following equation determines the delay:

$$\text{Delay Between Consecutive Transfers} = \frac{32 \times DLYBCT}{MCK}$$

### 30.6.8.2 Automatic Echo Mode

Automatic echo mode allows bit-by-bit retransmission. When a bit is received on the RXD pin, it is sent to the TXD pin, as shown in Figure 30-29. Programming the transmitter has no effect on the TXD pin. The RXD pin is still connected to the receiver input, thus the receiver remains active.

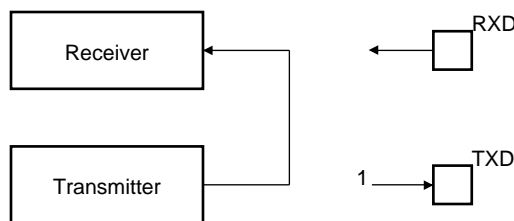
**Figure 30-29. Automatic Echo Mode Configuration**



### 30.6.8.3 Local Loopback Mode

Local loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in Figure 30-30. The TXD and RXD pins are not used. The RXD pin has no effect on the receiver and the TXD pin is continuously driven high, as in idle state.

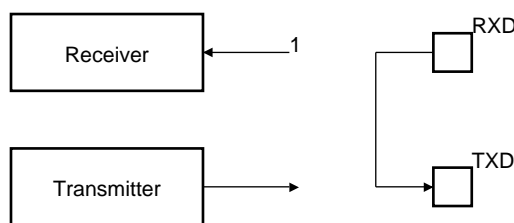
**Figure 30-30. Local Loopback Mode Configuration**



### 30.6.8.4 Remote Loopback Mode

Remote loopback mode directly connects the RXD pin to the TXD pin, as shown in Figure 30-31. The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

**Figure 30-31. Remote Loopback Mode Configuration**



### 30.7.2 USART Mode Register

**Name:** US\_MR

**Access Type:** Read-write

31	30	29	28	27	26	25	24
–	–	–	FILTER	–	MAX_ITERATION		
23	22	21	20	19	18	17	16
–	–	DSNACK	INACK	OVER	CLKO	MODE9	MSBF
15	14	13	12	11	10	9	8
CHMODE		NBSTOP		PAR			SYNC
7	6	5	4	3	2	1	0
CHRL		USCLKS		USART_MODE			

#### • USART\_MODE

USART_MODE				Mode of the USART
0	0	0	0	Normal
0	0	0	1	RS485
0	0	1	0	Hardware Handshaking
0	0	1	1	Modem
0	1	0	0	IS07816 Protocol: T = 0
0	1	0	1	Reserved
0	1	1	0	IS07816 Protocol: T = 1
0	1	1	1	Reserved
1	0	0	0	IrDA
1	1	x	x	Reserved

#### • USCLKS: Clock Selection

USCLKS		Selected Clock
0	0	MCK
0	1	MCK/DIV (DIV = 8)
1	0	Reserved
1	1	SCK

#### • CHRL: Character Length.

CHRL		Character Length
0	0	5 bits

### 31.6.7 Data Format

The data framing format of both the transmitter and the receiver are programmable through the Transmitter Frame Mode Register (SSC\_TFMR) and the Receiver Frame Mode Register (SSC\_RFMR). In either case, the user can independently select:

- the event that starts the data transfer (START)
- the delay in number of bit periods between the start event and the first data bit (STTDLY)
- the length of the data (DATLEN)
- the number of data to be transferred for each start event (DATNB).
- the length of synchronization transferred for each start event (FSLEN)
- the bit sense: most or lowest significant bit first (MSBF).

Additionally, the transmitter can be used to transfer synchronization and select the level driven on the TD pin while not in data transfer operation. This is done respectively by the Frame Sync Data Enable (FSDEN) and by the Data Default Value (DATDEF) bits in SSC\_TFMR.

## 32.5 Functional Description

### 32.5.1 TC Description

The three channels of the Timer Counter are independent and identical in operation. The registers for channel programming are listed in Table 32-4 on page 389.

### 32.5.2 16-bit Counter

Each channel is organized around a 16-bit counter. The value of the counter is incremented at each positive edge of the selected clock. When the counter has reached the value 0xFFFF and passes to 0x0000, an overflow occurs and the COVFS bit in TC\_SR (Status Register) is set.

The current value of the counter is accessible in real time by reading the Counter Value Register, TC\_CV. The counter can be reset by a trigger. In this case, the counter value passes to 0x0000 on the next valid edge of the selected clock.

### 32.5.3 Clock Selection

At block level, input clock signals of each channel can either be connected to the external inputs TCLK0, TCLK1 or TCLK2, or be connected to the internal I/O signals TIOA0, TIOA1 or TIOA2 for chaining by programming the TC\_BMR (Block Mode). See Figure 32-2 on page 378.

Each channel can independently select an internal or external clock source for its counter:

- Internal clock signals: TIMER\_CLOCK1, TIMER\_CLOCK2, TIMER\_CLOCK3, TIMER\_CLOCK4, TIMER\_CLOCK5
- External clock signals: XC0, XC1 or XC2

This selection is made by the TCCLKS bits in the TC Channel Mode Register.

The selected clock can be inverted with the CLKI bit in TC\_CMR. This allows counting on the opposite edges of the clock.

The burst function allows the clock to be validated when an external signal is high. The BURST parameter in the Mode Register defines this signal (none, XC0, XC1, XC2). See Figure 32-3 on page 378

**Note:** In all cases, if an external clock is used, the duration of each of its levels must be longer than the master clock period. The external clock frequency must be at least 2.5 times lower than the master clock

32.6.8 TC Register B

Register Name: TC\_RB [x=0..2]  
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
RB							
7	6	5	4	3	2	1	0
RB							

- **RB: Register B**  
RB contains the Register B value in real time.

### 33.6.8 PWM Interrupt Status Register

**Register Name:** PWM\_ISR

**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
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID**

0 = No new channel period has been achieved since the last read of the PWM\_ISR register.

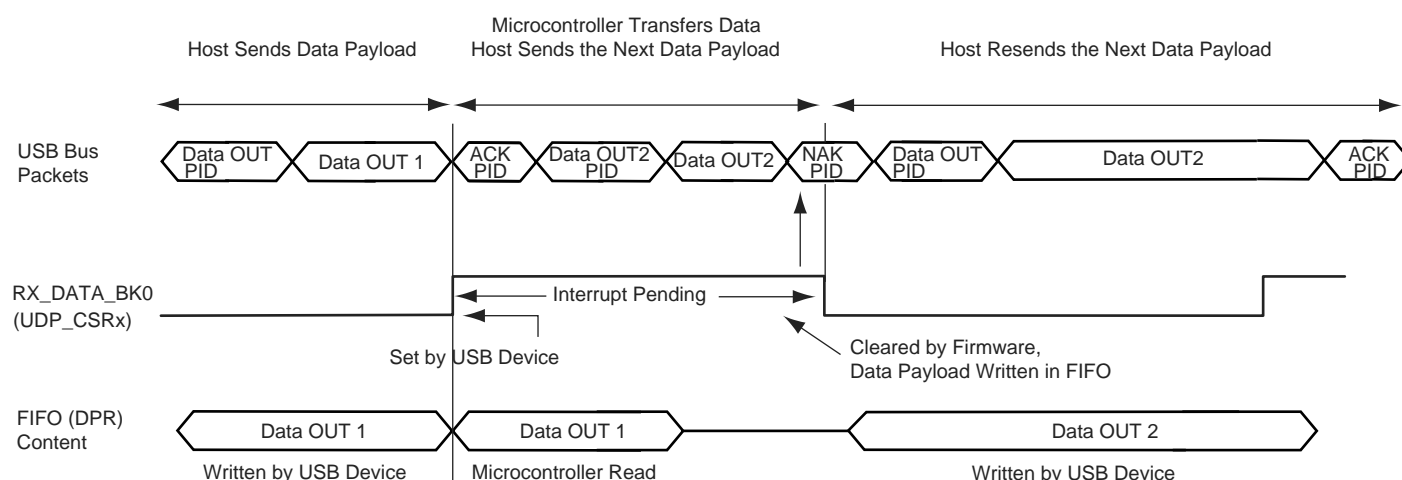
1 = At least one new channel period has been achieved since the last read of the PWM\_ISR register.

Note: Reading PWM\_ISR automatically clears CHIDx flags.



1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. While the FIFO associated to this endpoint is being used by the microcontroller, a NAK PID is returned to the host. Once the FIFO is available, data are written to the FIFO by the USB device and an ACK is automatically carried out to the host.
3. The microcontroller is notified that the USB device has received a data payload polling RX\_DATA\_BK0 in the endpoint's UDP\_CSRx register. An interrupt is pending for this endpoint while RX\_DATA\_BK0 is set.
4. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's UDP\_CSRx register.
5. The microcontroller carries out data received from the endpoint's memory to its memory. Data received is available by reading the endpoint's UDP\_FDRx register.
6. The microcontroller notifies the USB device that it has finished the transfer by clearing RX\_DATA\_BK0 in the endpoint's UDP\_CSRx register.
7. A new Data OUT packet can be accepted by the USB device.

**Figure 34-9. Data OUT Transfer for Non Ping-pong Endpoints**



An interrupt is pending while the flag RX\_DATA\_BK0 is set. Memory transfer between the USB device, the FIFO and microcontroller memory can not be done after RX\_DATA\_BK0 has been cleared. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the current Data OUT packet in the FIFO.

#### *Using Endpoints With Ping-pong Attributes*

During isochronous transfer, using an endpoint with ping-pong attributes is obligatory. To be able to guarantee a constant bandwidth, the microcontroller must read the previous data payload sent by the host, while the current data payload is received by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

$$T_{phs1} + T_{phs2} = 12 T_{csc}$$

Because this number is even, we choose  $T_{phs2} = T_{phs1}$  (else we would choose  $T_{phs2} = T_{phs1} + T_{csc}$ )

$$T_{phs1} = T_{phs2} = (12/2) T_{csc} = 6 T_{csc}$$

$$\Rightarrow PHASE1 = PHASE2 = T_{phs1}/T_{csc} - 1 = 5$$

The resynchronization jump width must be comprised between 1  $T_{csc}$  and the minimum of 4  $T_{csc}$  and  $T_{phs1}$ . We choose its maximum value:

$$T_{sjw} = \text{Min}(4 T_{csc}, T_{phs1}) = 4 T_{csc}$$

$$\Rightarrow SJW = T_{sjw}/T_{csc} - 1 = 3$$

Finally:  $CAN\_BR = 0x00053255$

#### 36.6.4.2 CAN Bus Synchronization

Two types of synchronization are distinguished: “hard synchronization” at the start of a frame and “resynchronization” inside a frame. After a hard synchronization, the bit time is restarted with the end of the SYNC\_SEG segment, regardless of the phase error. Resynchronization causes a reduction or increase in the bit time so that the position of the sample point is shifted with respect to the detected edge.

The effect of resynchronization is the same as that of hard synchronization when the magnitude of the phase error of the edge causing the resynchronization is less than or equal to the programmed value of the resynchronization jump width ( $t_{sjw}$ ).

When the magnitude of the phase error is larger than the resynchronization jump width and

- the phase error is positive, then PHASE\_SEG1 is lengthened by an amount equal to the resynchronization jump width.
- the phase error is negative, then PHASE\_SEG2 is shortened by an amount equal to the resynchronization jump width.

36.8.7 CAN Timer Register

Name: CAN\_TIM

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
TIMER15	TIMER14	TIMER13	TIMER12	TIMER11	TIMER10	TIMER9	TIMER8
7	6	5	4	3	2	1	0
TIMER7	TIMER6	TIMER5	TIMER4	TIMER3	TIMER2	TIMER1	TIMER0

• **TIMERx: Timer**

This field represents the internal CAN controller 16-bit timer value.

- **MACR: Abort Request for Mailbox x**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Cancels transfer request if the message has not been transmitted to the CAN transceiver.
Consumer	Cancels the current transfer before the remote frame has been sent.
Producer	Cancels the current transfer. The next remote frame will not be serviced.

It is possible to set MACR field for several mailboxes in the same time, setting several bits to the CAN\_ACR register.

- **MTCR: Mailbox Transfer Command**

Mailbox Object Type	Description
Receive	Allows the reception of the next message.
Receive with overwrite	Triggers a new reception.
Transmit	Sends data prepared in the mailbox as soon as possible.
Consumer	Sends a remote transmission frame.
Producer	Sends data prepared in the mailbox after receiving a remote frame from a Consumer.

This flag clears the MRDY and MABT flags in the CAN\_MS Rx register.

When several mailboxes are requested to be transmitted simultaneously, they are transmitted in turn. The mailbox with the highest priority is serviced first. If several mailboxes have the same priority, the mailbox with the lowest number is serviced first (i.e., MBx0 will be serviced before MBx 15 if they have the same priority).

It is possible to set MTCR for several mailboxes at the same time by writing to the CAN\_TCR register.

### 37.5.26.11 Transmit Underrun Errors Register

**Register Name:** EMAC\_TUND

**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
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
TUND							

- **TUND: Transmit Underruns**

An 8-bit register counting the number of frames not transmitted due to a transmit DMA underrun. If this register is incremented, then no other statistics register is incremented.

### 37.5.26.12 Carrier Sense Errors Register

**Register Name:** EMAC\_CSE

**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
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
CSE							

- **CSE: Carrier Sense Errors**

An 8-bit register counting the number of frames transmitted where carrier sense was not seen during transmission or where carrier sense was deasserted after being asserted in a transmit frame without collision (no underrun). Only incremented in half-duplex mode. The only effect of a carrier sense error is to increment this register. The behavior of the other statistics registers is unaffected by the detection of a carrier sense error.