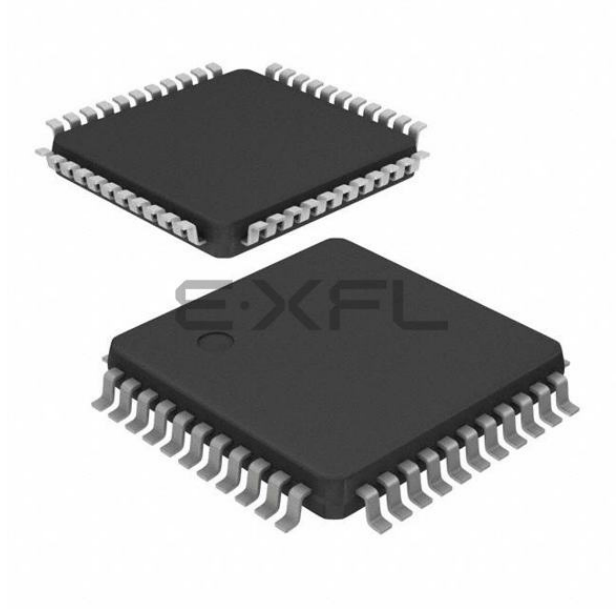


Welcome to E-XFL.COM

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	XA
Core Size	16-Bit
Speed	32MHz
Connectivity	CANbus, EBI/EMI, SPI, UART/USART
Peripherals	DMA, POR, PWM, WDT
Number of I/O	32
Program Memory Size	32KB (32K x 8)
Program Memory Type	OTP
EEPROM Size	-
RAM Size	1K x 8
Voltage - Supply (Vcc/Vdd)	4.5V ~ 5.5V
Data Converters	-
Oscillator Type	External
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	44-LQFP
Supplier Device Package	44-LQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/pxac37kfbd-00-157

XA 16-bit microcontroller family
32K/1024 OTP CAN transport layer controller
1 UART, 1 SPI Port, CAN 2.0B, 32 CAN ID filters, transport layer co-processor

XA-C3

GENERAL DESCRIPTION

The XA–C3 is a member of the Philips XA (eXtended Architecture) family of high–performance 16–bit single–chip microcontrollers. The XA–C3 combines an array of standard peripherals together with a PeliCAN CAN 2.0B engine and unique ”Message Management” hardware to provide integrated support for most CAN Transport Layer (CTL) protocols such as DeviceNet, CANopen and OSEK. For additional details, refer to the *XA-C3 Overview* on page 35.

The XA architecture supports:

- Easy 16-bit migration from the 80C51 architecture.
- 16–bit fully static CPU with 24–bit addressed PROGRAM and DATA spaces.
- Twenty–one 16–bit CPU core registers capable of all arithmetic and logic operations while serving as memory pointers.
- An enhanced orthogonal instruction set tailored for high–level support of the C language.
- Multi–tasking and direct real–time executive support.
- Low–power operation intrinsic to the XA architecture includes Power–Down and Idle modes.

FEATURES IN COMMON WITH XA-G3

- Pin–compatibility (CAN RxD and CAN TxD use the XA-G3 NC pins).
- 32K bytes of on–chip EPROM PROGRAM memory (see Table 1).
- 44–pin PLCC (Figure 1 and Table 2) and 44–pin LQFP (Figure 2 and Table 3) packages.
- Commercial (0 to 70^{o}C) and Industrial (–40 to 85^{o}C) ranges.
- Supports off–chip addressing of PROGRAM and DATA memory up to 1 megabyte each (20 address lines).
- Three standard counter/timers (T0, T1, and T2) with enhancements such as Auto Reload for PWM outputs.
- UART–0 with enhancements such as separate Rx and Tx interrupts, Break Detection, and Automatic Address Recognition.
- Watchdog with a secure WFEED1 / WFEED2 sequence.
- Four 8–bit I/O ports with 4 programmable output configurations per pin.

XA-C3 SPECIFIC FEATURES

- 32 MHz operating frequency at 4.5 to 5.5V operation.
- One Serial Port Interface (SPI)
- 1024 bytes of on–chip DATA RAM.
- 42 vectored interrupts. These include 13 maskable Events, 7 Software interrupts, 6 Exceptions, 16 software Traps, segmented DATA memory, multiple User stacks, and banked registers to support rapid context switching.
- External interfacing via a 16–bit DATA bus width.

XA-C3 CAN AND CTL FEATURES

- A PeliCAN CAN 2.0B engine from the SJA1000 Stand–alone CAN controller which supports 11– and 29–bit IDentifiers and the maximum CAN data rate (1 Mbps) and CAN Diagnostics.
- Hardware “Message Management” support for all major CTL protocols: DeviceNet, CANopen, OSEK.
- Automatic (hardware) assembly of Fragmented Messages via a Transport Layer Co-Processor. Concurrent assembly of up to 32 separate interleaved Fragmented Messages
- 32 CAN Transport Layer (CTL) Message Objects are modelled as a FullCAN Object Superset.
- 32 separate filters/screeners (one per Message Object), each allowing a 30–bit ID Match and full 29–bit Mask (i.e., each filter/screener represents a unique Group address).
- Each Message Object can be configured as Receive or Transmit.
- A separate message buffer is associated with each CTL Message Object. 32 message buffers are located in XRAM and managed by 32 DMA channels. Message buffer size for each Message Object is independently configurable in length (from 2 to 256 bytes).
- For single–chip systems there is a 512–byte (on–chip) XRAM message buffer, independent of the 1K on–chip DATA RAM, which is extendable (off–chip) to 8K bytes (i.e., 32 Message Objects that can be up to 256 bytes each).

LOGIC SYMBOL AND BLOCK DIAGRAM

Refer to Figure 3 for the logic symbol for the XA-C3 and to Figure 4 for a simplified block diagram representation.

UPGRADING XA-G3 DESIGNS TO CAN

- XA-G3 NC pins are XA-C3 CAN RxD and CAN TxD pins.
- XA-G3 UART–1 is replaced by a Serial Port Interface (SPI)
- XA-C3 software must never write to the BCR register
- XA-C3 software must initialize BTRH and BTRL with 00h

44-pin LQFP package

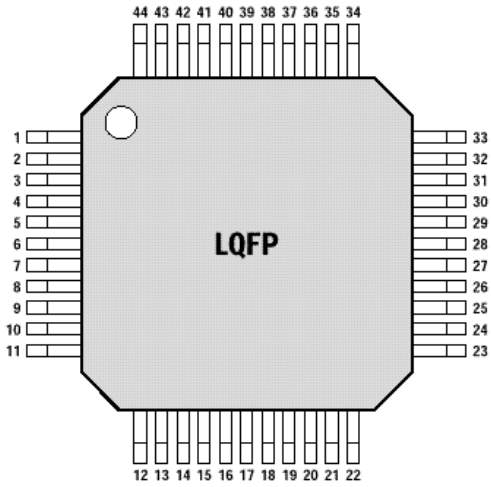


Figure 2. 44-pin PLCC package

Table 3. 44-pin LQFP package pin functions

Pin	Function (see Note)	Pin	Function (see Note)
1	P1.5 ; SPITx	23	P2.5 ; A17D13
2	P1.6 ; T2 ; SPICLK	4	P2.6 ; A18D14
3	P1.7 ; T2EX	25	P2.7 ; A19D15
4	RST/	26	PSEN/
5	P3.0 ; RxD0	27	ALE ; PROG/
6	CAN RxD	28	CAN TxD
7	P3.1 ; TxD0	29	EA/ ; Vpp ; WAIT
8	P3.2 ; INT0/	30	P0.7 ; A11D7
9	P3.3 ; INT1/	31	P0.6 ; A10D6
10	P3.4 ; T0	32	P0.5 ; A9D5
11	P3.5 ; T1	33	P0.4 ; A8D4
12	P3.6 ; WRL/	34	P0.3 ; A7D3
13	P3.7 ; RD/	35	P0.2 ; A6D2
14	XTAL2	36	P0.1 ; A5D1
15	XTAL1	37	P0.0 ; A4D0
16	VSS	38	VDD
17	VDD	39	VSS
18	P2.0 ; A12D8	40	P1.0 ; WRH/
19	P2.1 ; A13D9	41	P1.1 ; A1
20	P2.2 ; A14D10	42	P1.2 ; A2
21	P2.3 ; A15D11	43	P1.3 ; A3
22	P2.4 ; A16D12	44	P1.4 ; SPIRx

NOTE:
1. All active–low signals are indicated by a "**/**" symbol

SPECIAL FUNCTION REGISTERS

Table 5. Special Function Registers

NAME	DESCRIPTION	SFR ADDRESS	BIT FUNCTIONS AND BIT ADDRESSES								RESET VALUE
			7	6	5	4	3	2	1	0	
BCR	Bus Configuration Register	46Ah	–	–	–	WAITD	BUSD	–	–	–	07h (Note 1)
BTRH	Bus Timing Register High	469h	DW1	DW0	DWA1	DWA0	DR1	DR0	DRA1	DRA0	FFh (Note 2)
BTRL	Bus Timing Register Low	468h	WM1	WM0	ALEW	–	CR1	CR0	CRA1	CRA0	EFh (Note 2)
MIFCNTL	MIF Control Register	495h	–	–	–	WDSBL	BUSD	–	–	–	
MRBL	MMR Base address Low	496h	MA15	MA14	MA13	MA12	–	–	–	MRBE	F0h
MRBH	MMR Base address High	497h	MA23	MA22	MA21	MA20	MA19	MA18	MA17	MA16	0Fh
DS	Data Segment	441h									00h
ES	Extra Segment	442h									00h
CS	Code Segment	443h									00h
			33F	33E	33D	33C	33B	33A	339	338	
IEH*	Interrupt Enable High	427h	EMRI	EMTI	EMER	ECER	ESPI	–	ETI0	ERI0	00h
			337	336	335	334	333	332	331	330	
IEL*	Interrupt Enable Low	426h	EA	–	EBUFF	ET2	ET1	EX1	ET0	EX0	00h
IPA0	Interrupt Priority Assignment 0	4A0h	–	PT0			–	PX0			00h
IPA1	Interrupt Priority Assignment 1	4A1h	–	PT1			–	PX1			00h
IPA2	Interrupt Priority Assignment 2	4A2h	–	PBUFF			–	PT2			00h
IPA4	Interrupt Priority Assignment 4	4A4h	–	PTI0			–	PRI0			00h
IPA5	Interrupt Priority Assignment 5	4A5h	–	PSPI			–	–			00h
IPA6	Interrupt Priority Assignment 6	4A6h	–	PMER			–	PCER			00h
IPA7	Interrupt Priority Assignment 7	4A7h	–	PMRI			–	PMTI			00h
			387	386	385	384	383	382	381	380	
P0*	Port 0	430h	A11D7	A10D6	A9D5	A8D4	A7D3	A6D2	A5D1	A4D0	FFh
			38F	38E	38D	38C	38B	38A	389	388	
P1*	Port 1	431h	T2EX	T2 ; SPICLK	SPITx	SPIRx	A3	A2	A1	WRH/	FFh
			397	396	395	394	393	392	391	390	
P2*	Port 2	432h	A19D15	A18D14	A17D13	A16D12	A15D11	A14D10	A13D9	A12D8	FFh
			39F	39E	39D	39C	39B	39A	399	398	
P3*	Port 3	433h	RD/	WRL/	T1	T0	INT1/	INT0/	TxD0	RxD0	FFh
P0CFGA	Port 0 Configuration A	470h									Note 3
P1CFGA	Port 1 Configuration A	471h									Note 3
P2CFGA	Port 2 Configuration A	472h									Note 3
P3CFGA	Port 3 Configuration A	473h									Note 3
P0CFGB	Port 0 Configuration B	4F0h									Note 3
P1CFGB	Port 1 Configuration B	4F1h									Note 3
P2CFGB	Port 2 Configuration B	4F2h									Note 3
P3CFGB	Port 3 Configuration B	4F3h									Note 3
			227	226	225	224	223	222	221	220	
PCON*	Power Control Reg	404h	–	–	–	–	–	–	PD	IDL	00h
			20F	20E	20D	20C	20B	20A	209	208	
PSWH*	Program Status Word High	401h	SM	TM	RS1	RS0	IM3	IM2	IM1	IM0	Note 4
			207	206	205	204	203	202	201	200	
PSWL*	Program Status Word Low	400h	C	AC	–	–	–	V	N	Z	Note 4
			217	216	215	214	213	212	211	210	
PSW51*	80C51–compatible PSW	402h	C	AC	F0	RS1	RS0	V	F1	P	Note 5
RTH0	Timer 0 extended reload, high byte	455h									00h
RTH1	Timer 1 extended reload, high byte	457h									00h
RTL0	Timer 0 extended reload, low byte	454h									00h
RTL1	Timer 1 extended reload, low byte	456h									00h
			307	306	305	304	303	302	301	300	
S0CON*	Serial port 0 control register	420h	SM0_0	SM1_0	SM2_0	REN_0	TB8_0	RB8_0	TI_0	RI_0	00h

T2CON Address:418
Bit Addressable
Reset Value: 00H

MSB							LSB
TF2	EXF2	RCLK0	TCLK0	EXEN2	TR2	C2 or T2/	CP or RL2/

BIT	SYMBOL	FUNCTION
T2CON.7	TF2	Timer 2 overflow flag. Set by hardware on Timer/Counter overflow. Must be cleared by software. TF2 will not be set when RCLK0, RCLK1, TCLK0, TCLK1 or T2OE=1.
T2CON.6	EXF2	Timer 2 external flag is set when a capture or reload occurs due to a negative transition on T2EX (and EXEN2 is set). This flag will cause a Timer 2 interrupt when this interrupt is enabled. EXF2 is cleared by software.
T2CON.5	RCLK0	Receive Clock Flag.
T2CON.4	TCLK0	Transmit Clock Flag. RCLK0 and TCLK0 are used to select Timer 2 overflow rate as a clock source for UART0 instead of Timer T1.
T2CON.3	EXEN2	Timer 2 external enable bit allows a capture or reload to occur due to a negative transition on T2EX.
T2CON.2	TR2	Start=1/Stop=0 control for Timer 2.
T2CON.1	C2 or T2/	Timer or counter select. 0=Internal timer 1=External event counter (falling edge triggered)
T2CON.0	CP or RL2/	Capture/Reload flag. If CP/RL2 & EXEN2=1 captures will occur on negative transitions of T2EX. If CP/RL2=0, EXEN2=1 auto reloads occur with either Timer 2 overflows or negative transitions at T2EX. If RCLK or TCLK=1 the timer is set to auto reload on Timer 2 overflow, this bit has no effect.

SU001326

Figure 8. Timer/Counter 2 Control (T2CON) Register

New Timer-Overflow Toggle Output

In the XA, the timer module now has two outputs, which toggle on overflow from the individual timers. The same device pins that are used for the T0 and T1 count inputs are also used for the new overflow outputs. An SFR bit (TnOE in the TSTAT register – see Figure 9 — is associated with each counter and indicates whether Port–SFR data or the overflow signal is output to the pin. These outputs could be used in applications for generating variable duty cycle PWM outputs (changing the auto–reload register values). Also, variable frequency (f_{osc}/8 to f_{osc}/8,388,608) outputs could be achieved by adjusting the prescaler along with the auto–reload register values.

Timer T2

Timer 2 in the XA is a 16–bit Timer/Counter which can operate as either a timer or as an event counter. This is selected by {C2 or T2/} (T2CON[1]) (see Figure 8). Upon timer T2 overflow/underflow, the TF2 flag is set, which may be used to generate an interrupt. It can be operated in one of three operating modes: auto–reload (up or down counting), capture, or as the baud rate generator (for the UART via SFRs T2CON and T2MOD – see Figure 10.
These modes are shown in Table 7.

Capture Mode

In the capture mode there are two options which are selected by bit EXEN2 (T2CON[3]). If EXEN2 = 0, then timer 2 is a 16–bit timer or counter, which upon overflowing sets bit TF2 (T2CON[7]), the timer 2 overflow bit. This will cause an interrupt when the timer 2 interrupt is enabled.

If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1–to–0 transition at External input T2EX causes the current value in the Timer 2 registers, TL2 and TH2, to be captured into registers RCAP2L and RCAP2H, respectively. In addition, the transition at T2EX causes bit EXF2 (T2CON[6]) to be set. This will cause an interrupt in the same fashion as TF2 when the Timer 2 interrupt is enabled. The capture mode is illustrated in Figure 11.

Auto-Reload Mode (Up or Down Counter)

In the auto–reload mode, the timer registers are loaded with the 16–bit value in T2CAPH and T2CAPL when the count overflows. T2CAPH and T2CAPL are initialized by software. If the EXEN2 bit (T2CON[3]) is set, the timer registers will also be reloaded and the EXF2 flag T2CON[6] set when a 1–to–0 transition occurs at input T2EX. The auto–reload mode is shown in Figure 12.

In this mode, Timer 2 can be configured to count up or down. This is done by setting or clearing the DCEN (Down Counter Enable) bit T2MOD[0] (see Table 7). The T2EX pin then controls the count direction. When T2EX is high, the count is in the up direction, when T2EX is low, the count is in the down direction.

Figure 12 shows Timer 2, which will count up automatically, since DCEN = 0. In this mode there are two options selected by bit EXEN2 in the T2CON register. If EXEN2 bit = 0, then Timer 2 counts up to FFFFh and sets the TF2 (Overflow Flag) bit T2CON[7] upon overflow. This causes the Timer 2 registers to be reloaded with the 16–bit value in T2CAPL and T2CAPH, whose values are preset by software. If EXEN2 bit T2CON[3] = 1, a 16–bit reload can be triggered either by an overflow or by a 1–to–0 transition at input T2EX. This transition also sets the EXF2 bit. If enabled, either TF2 bit or EXF2 bit can generate the Timer 2 interrupt.

In Figure 13 where the DCEN bit = 1; this enables the Timer 2 to count up or down. In this mode, the logic level of T2EX pin controls the direction of count. When a logic ‘1’ is applied at pin T2EX, the Timer 2 will count up. The Timer 2 will overflow at FFFFh and set the TF2 bit flag, which can then generate an interrupt if enabled. This timer overflow also causes the 16–bit value in T2CAPL and T2CAPH to be reloaded into timer registers TL2 and TH2, respectively.

A logic ‘0’ at pin T2EX causes Timer 2 to count down. When counting down, the timer value is compared to the 16–bit value contained in T2CAPH and T2CAPL. When the value is equal, the

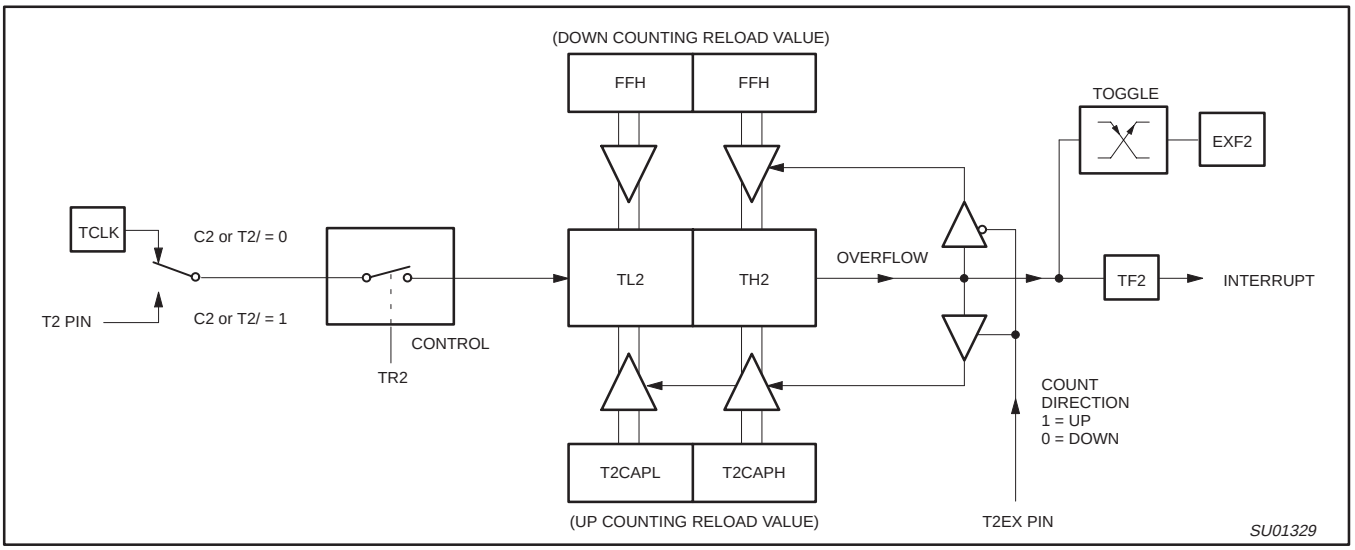


Figure 13. Timer 2 Auto Reload Mode (DCEN = 1)

3. The timer reload value may never be larger than the timer range.
4. If a timer reload value calculation gives a negative or fractional result, the baud rate requested is not possible at the given oscillator frequency and N value.

Using Timer 2 to Generate Baud Rates

Timer T2 is a 16–bit up/down counter. As a baud rate generator, Timer 2 is selected as a clock source for UART–0 transmitter and/or receiver by setting TCLK0 and/or RCLK0 in T2CON (see Table 10).

As the baud rate generator, T2 is incremented as f_{OSC}/N where N = 4, 16, or 64 depending on TCLK as programmed in SCR bits PT1 (SCR[3]) and PTO (SCR[2]). See Table 11).

NOTE: Pin T2EX [P1.7] acts as an additional External interrupt "INT2/" whenever Timer T2 is used as a baud rate generator.

Table 10. T2CON Settings

T2CON 0x418		T2CON[5]	T2CON[4]	
		RCLK0	TCLK0	

Table 11. Prescaler Select for Timer Clock

SCR 0x440		SCR[3]	SCR[2]	
		PT1	PT0	

S0STAT Address: S0STAT 421
Bit Addressable
Reset Value: 00H

MSB							LSB
—	—	—	—	FE0	BR0	OE0	STINT0

BIT	SYMBOL	FUNCTION
S0STAT.3	FE0	Framing Error flag is set when the receiver fails to see a valid STOP bit at the end of the frame. Cleared by software.
S0STAT.2	BR0	Break Detect flag is set if a character is received with all bits (including STOP bit) being logic '0'. Thus it gives a "Start of Break Detect" on bit 8 for Mode 1 and bit 9 for Modes 2 and 3. The break detect feature operates independently of the UARTs and provides the START of Break Detect status bit that a user program may poll. Cleared by software.
S0STAT.1	OE0	Overrun Error flag is set if a new character is received in the receiver buffer while it is still full (before the software has read the previous character from the buffer), i.e., when bit 8 of a new byte is received while RI_0 in S0CON is still set. Cleared by software.
S0STAT.0	STINT0	This flag must be set to enable any of the above status flags to generate a receive interrupt (RI_0). The only way it can be cleared is by a software write to this register.

SU01315

Figure 15. Serial Port Extended Status (S0STAT) Register

Note: See also Figure 17 regarding Framing Error flag.

UART Interrupt Scheme

There are separate interrupt vectors for UART–0 transmit and receive functions (see Table 12 below).

Table 12. Vector Locations for UART in XA

Vector Address	Interrupt Source	Arbitration
00A0h – 00A3h	UART 0 Receiver	10
00A4h – 00A7h	UART 0 Transmitter	11

NOTE:
The transmit and receive vectors could contain the same ISR address to work like an 8051 interrupt scheme.

Multiprocessor Communications

Modes 2 and 3 have a special provision for multiprocessor communications. In these modes, 9 data bits are received. The 9th one goes into bit RB_8 (S0CON[2]). Then comes a stop bit. UART–0 can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB_8 = 1. This feature is enabled by setting bit SM2_0 (S0CON[5]). A way to use this feature in multiprocessor systems is as follows:

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 9th bit is 1 in an address byte and 0 in a data byte. With SM2_0 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2_0 bit and prepare to receive the data bytes that will be coming. The slaves that weren't being addressed leave their SM2_0 bits set and go on about their business, ignoring the incoming data bytes.

SM2_0 has no effect in UART Mode 0, and in UART Mode 1 can be used to check the validity of the stop bit although this is better done with the Framing Error flag (FE0) {S0STAT[3]}. In a Mode 1 reception, if SM2_0 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

Error Handling, Status Flags and Break Detect
UART–0 has the four error flags as described in Figure 15.

Automatic Address Recognition
Automatic Address Recognition is a feature which allows UART–0 to recognize certain addresses in the serial bit stream by using hardware to make the comparisons. This feature saves a great deal of software overhead by eliminating the need for the software to examine every serial address which passes by the serial port. This feature is enabled by setting the SM2_0 bit. In the 9–bit UART Modes (Mode 2 and Mode 3) the Receive Interrupt flag (RI_0) (S0CON[0]) will be automatically set when the received byte contains either the "Given" address or the "Broadcast" address. The 9–bit mode requires that the 9th information bit is a 1 to indicate that the received information is an address and not data. Automatic address recognition is shown in Figure 16.

Using the Automatic Address Recognition feature allows a master to selectively communicate with one or more slaves by invoking the Given slave address or addresses. All of the slaves may be contacted by using the Broadcast address. Two special Function Registers are used to define the slave's address, S0ADDR, and the address mask, S0ADEN. S0ADEN is used to define which bits in the S0ADDR are to be used and which bits are "don't care". The S0ADEN mask can be logically ANDed with the S0ADDR to create the "Given" address which the master will use for addressing each of the slaves. Use of the Given address allows multiple slaves to be recognized while excluding others. The following examples will help to show the versatility of this scheme:

Slave 0	S0ADDR =	1100 0000
	S0ADEN =	1111 1101
	Given =	1100 00X0
Slave 1	S0ADDR =	1100 0000
	S0ADEN =	1111 1110
	Given =	1100 000X

In the above example S0ADDR is the same and the S0ADEN data is used to differentiate between the two slaves. Slave 0 requires a 0 in bit 0 and it ignores bit 1. Slave 1 requires a 0 in bit 1 and bit 0 is ignored. A unique address for Slave 0 would be 1100 0010 since slave 1 requires a 0 in bit 1. A unique address for slave 1 would be 1100 0001 since a 1 in bit 0 will exclude slave 0. Both slaves can be selected at the same time by an address which has bit 0 = 0 (for slave 0) and bit 1 = 0 (for slave 1). Thus, both could be addressed with 1100 0000.

In a more complex system the following could be used to select slaves 1 and 2 while excluding slave 0:

Slave 0	S0ADDR =	1100 0000
	S0ADEN =	1111 1001
	Given =	1100 0XX0
Slave 1	S0ADDR =	1110 0000
	S0ADEN =	1111 1010
	Given =	1110 0X0X
Slave 2	S0ADDR =	1110 0000
	S0ADEN =	1111 1100
	Given =	1110 00XX

In the above example the differentiation among the 3 slaves is in the lower 3 address bits. Slave 0 requires that bit 0 = 0 and it can be uniquely addressed by 1110 0110. Slave 1 requires that bit 1 = 0 and it can be uniquely addressed by 1110 and 0101. Slave 2 requires that bit 2 = 0 and its unique address is 1110 0011. To select Slaves 0 and 1 and exclude Slave 2 use address 1110 0100, since it is necessary to make bit 2 = 1 to exclude slave 2.

The Broadcast Address for each slave is created by taking the logical OR of S0ADDR and S0ADEN. Zeros in this result are treated as don't–cares. In most cases, interpreting the don't–cares as ones, the broadcast address will be FF hexadecimal.

Upon Reset, S0ADDR and S0ADEN are loaded with 0s. This produces a given address of all "don't cares" as well as a Broadcast address of all "don't cares". This effectively disables the Automatic Addressing mode and allows the microcontroller to use standard UART drivers which do not make use of this feature.

S0CON Address: S0CON 420

Bit Addressable
Reset Value: 00H

MSB							LSB
SM0_0	SM1_0	SM2_0	REN_0	TB8_0	RB8_0	TI_0	RI_0

Where SM0_0, SM1_0 specify the serial port mode, as follows:

SM0_0	SM1_0	Mode	Description	Baud Rate
0	0	0	shift register	$f_{OSC}/16$
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	$f_{OSC}/32$
1	1	3	9-bit UART	variable

BIT	SYMBOL	FUNCTION
S0CON.5	SM2_0	Enables the multiprocessor communication feature in Modes 2 and 3. In Mode 2 or 3, if SM2_0 is set to 1, then RI_0 will not be activated if the received 9th data bit (RB8_0) is 0. In Mode 1, if SM2_0=1 then RI_0 will not be activated if a valid stop bit was not received. In Mode 0, SM2_0 should be 0.
S0CON.4	REN_0	Enables serial reception. Set by software to enable reception. Clear by software to disable reception.
S0CON.3	TB8_0	The 9th data bit that will be transmitted in Modes 2 and 3. Set or clear by software as desired. The TB8_0 bit is not double buffered. See text for details.
S0CON.2	RB8_0	In Modes 2 and 3, is the 9th data bit that was received. In Mode 1, if SM2_0=0, RB8_0 is the stop bit that was received. In Mode 0, RB8_0 is not used.
S0CON.1	TI_0	Transmit interrupt flag. Set when another byte may be written to the UART transmitter. See text for details. Must be cleared by software.
S0CON.0	RI_0	Receive interrupt flag. Set by hardware at the end of the 8th bit time in Mode 0, or at the end of the stop bit time in the other modes (except see SM2_0). Must be cleared by software.

SU01330

Figure 16. Serial Port Control (S0CON) Register

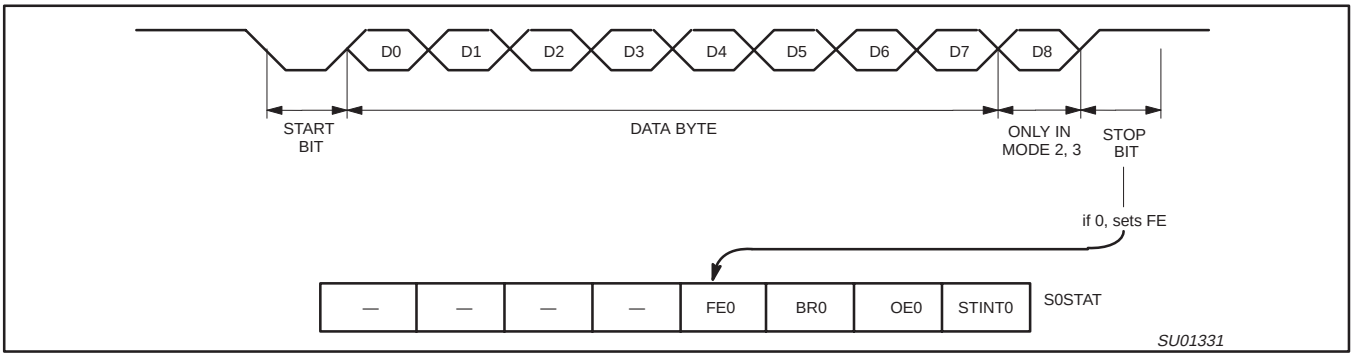


Figure 17. UART Framing Error Detection

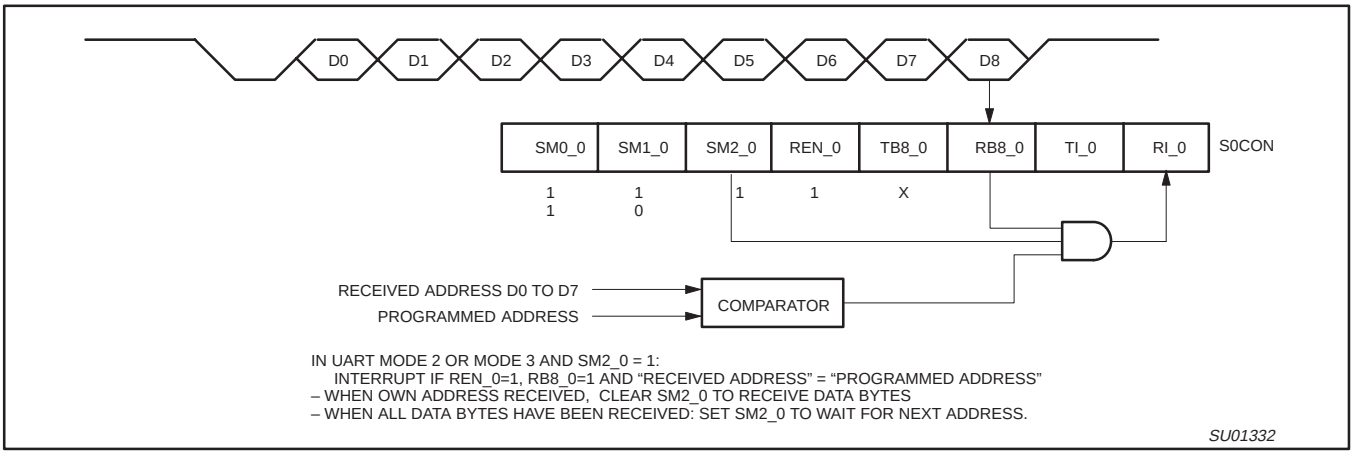


Figure 18. UART Multiprocessor Communication, Automatic Address Recognition

INPUT/OUTPUT PORT PIN CONFIGURATION

Each I/O port pin can be user–configured to one of four modes: Quasi–Bidirectional (essentially the same as standard 80C51 family I/O ports), Open–Drain, Push–Pull, and Off (High Impedance). After Reset, the default configuration is Quasi–Bidirectional.

I/O port pin configurations are determined by the settings in port configuration SFRs. There are two SFRs for each port, called PnCFGA and PnCFGB, where “n” is the port number. One bit in each of the two SFRs relates to the setting for the corresponding port pin, allowing any combination of the four modes to be mixed on any port pins. For instance, the mode of port 1 pin 3 (P1.3) is controlled by setting bit 3 (P1CFGA[3] and P1CFGB[3]).

Table 13 shows the configuration register settings for the four port pin modes. The DC electrical characteristics of each mode may be found in Table 19.

Table 13. Port Configuration Register Settings

PnCFGB	PnCFGA	Port Pin Mode
0	0	Open–Drain
0	1	Quasi–Bidirectional
1	0	Off (High Impedance)
1	1	Push–Pull

Note: Mode changes may cause glitches to occur during transitions. When modifying both registers, WRITE instructions should be carried out consecutively.

EXTERNAL BUS

If off chip code is selected (through the use of the EA/ pin), initial code fetches will be done within a full 20–bit address space. The External PROGRAM/DATA bus provides 16 bit width in a 20–bit ADDRESS space.

RESET

Refer to Figure 19 for a recommended Reset circuit example.

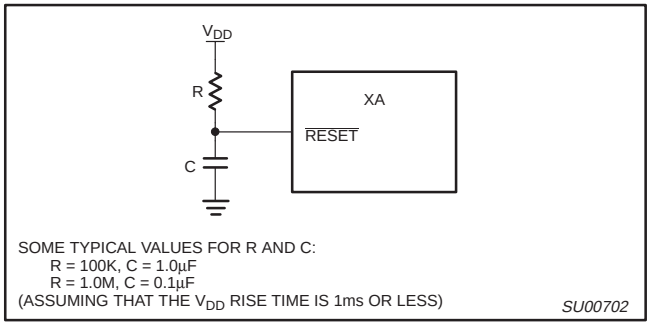


Figure 19. Recommended Reset Circuit

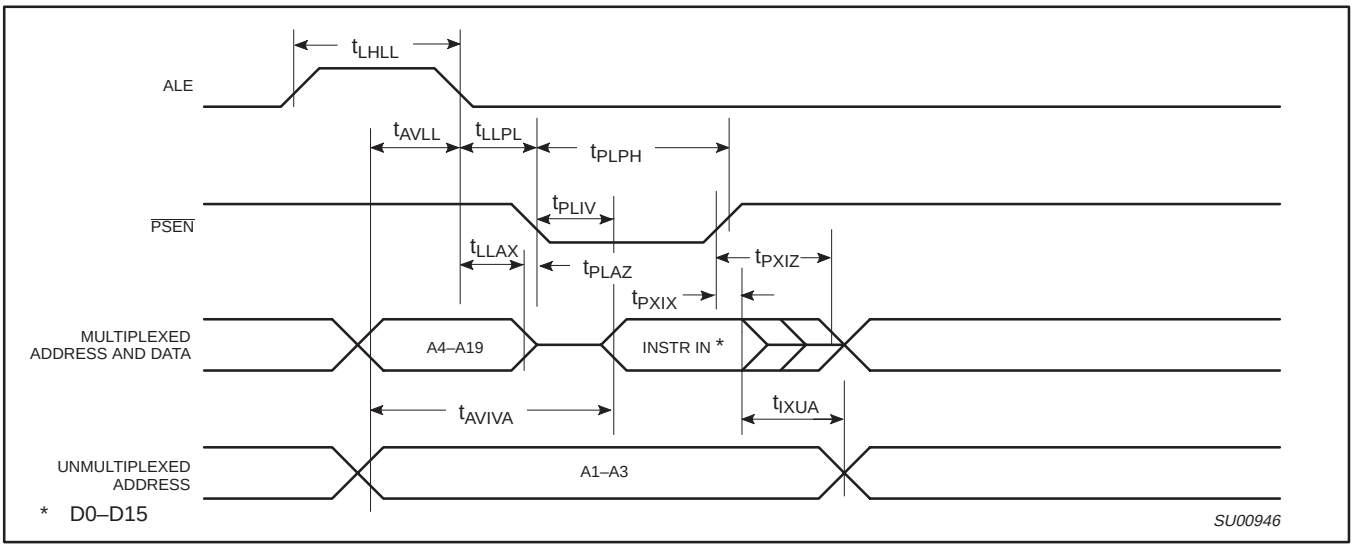


Figure 21. External PROGRAM Memory Read Cycle (ALE Cycle)

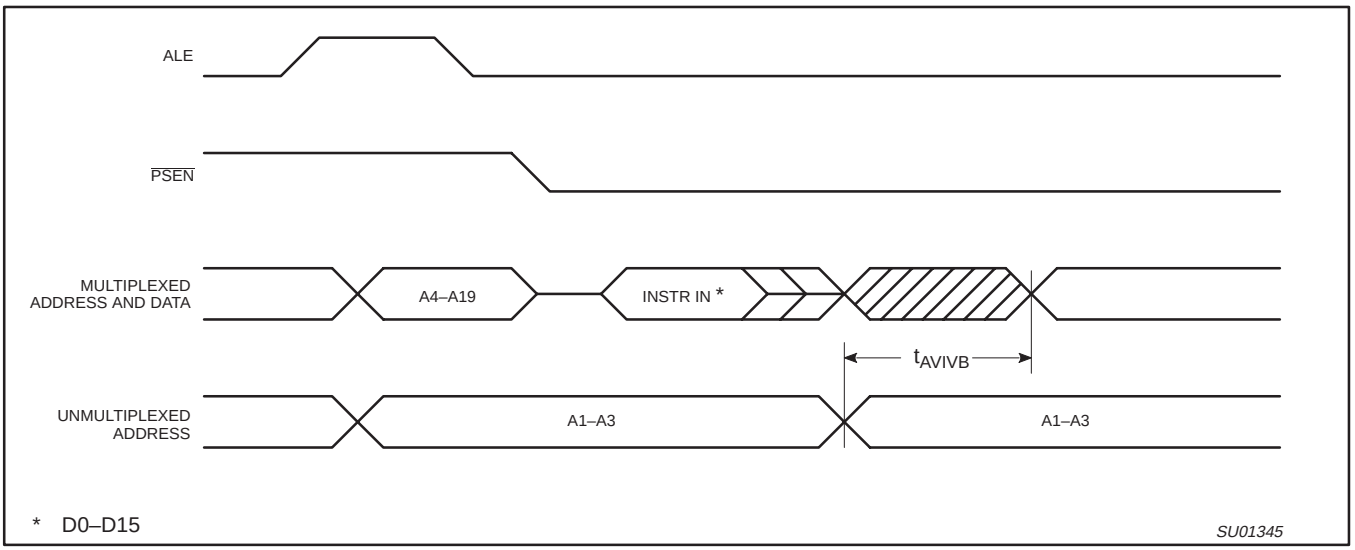


Figure 22. External PROGRAM Memory Read Cycle (Non-ALE Cycle)

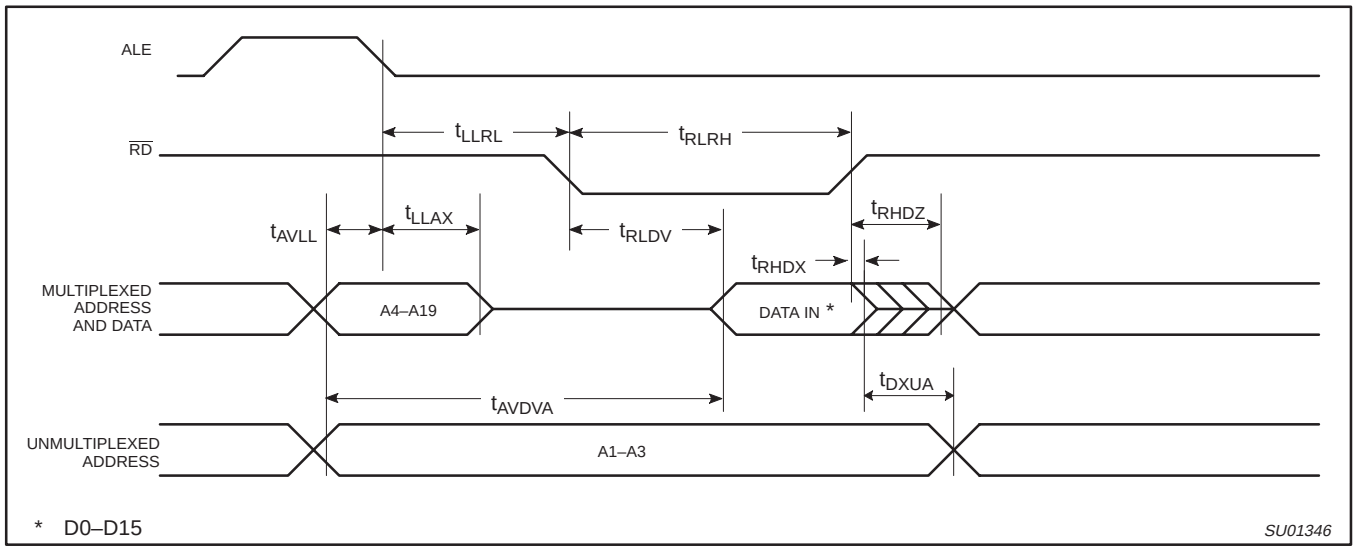


Figure 23. External DATA Memory Read Cycle (ALE Cycle)

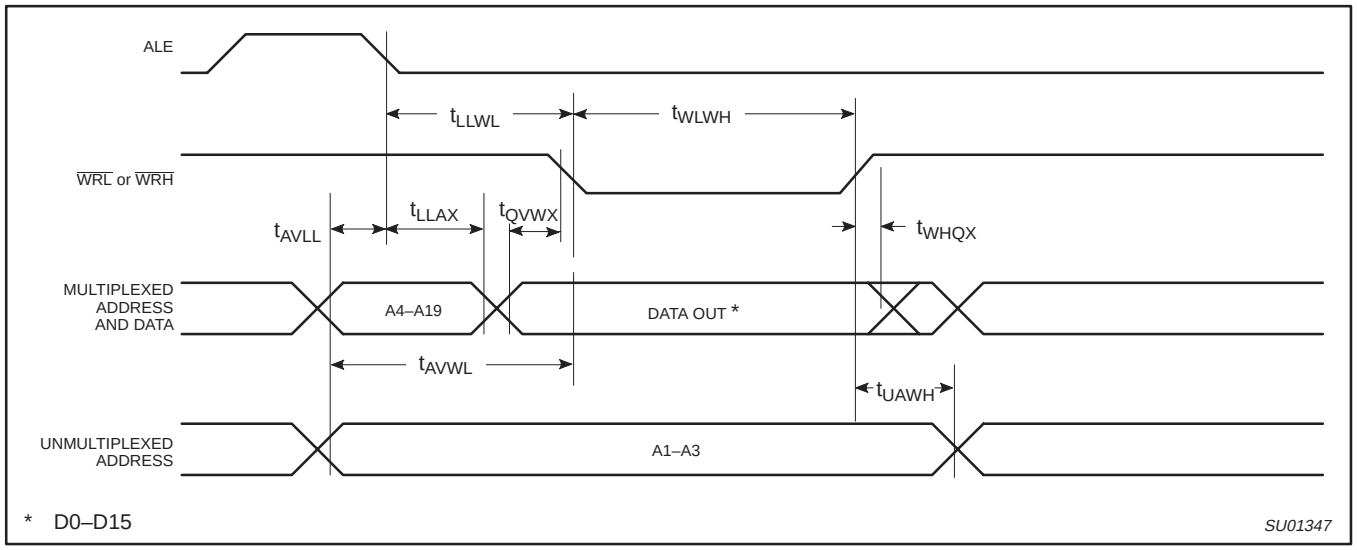


Figure 24. External DATA Memory Write Cycle

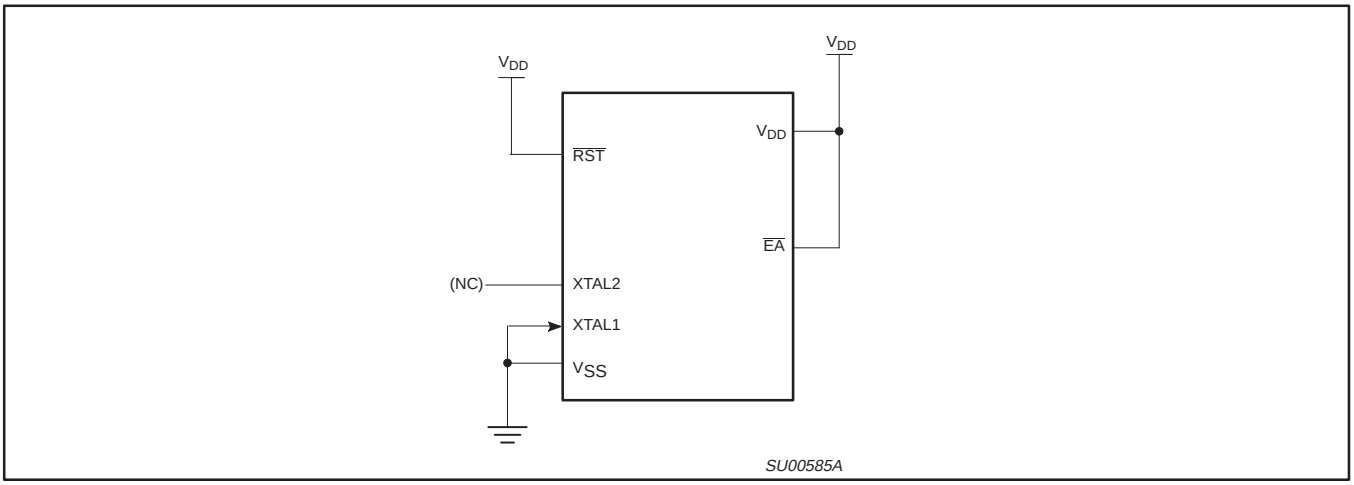


Figure 33. I_{DD} Test Condition, Power-Down Mode

Note: All other pins are disconnected. V_{DD}=2V to 5.5V

EPROM CHARACTERISTICS

The XA–C37 is programmed by using a modified Improved Quick–Pulse Programming™ algorithm. This algorithm is essentially the same as that used by the later 80C51 family EPROM parts. However, different pins are used for many programming functions.

Detailed EPROM programming information may be obtained from the internet at www.philipsmcu.com/ftp.html.

The XA–C3 contains three signature bytes that can be read and used by an EPROM programming system to identify the device. The signature bytes identify the device as an XA–Gx manufactured by Philips.

Security Bits

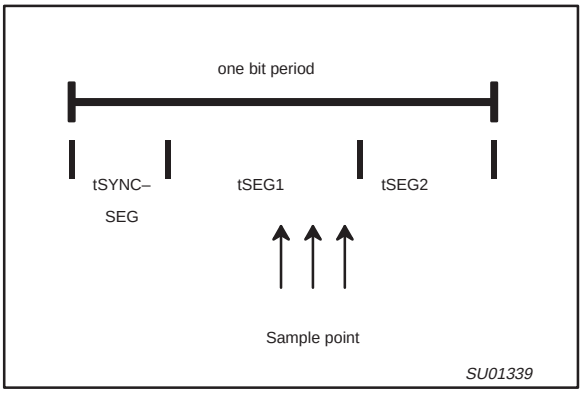
With none of the security bits programmed the code in the PROGRAM memory can be verified. When only security bit 1 (see Table 21) is programmed, MOVC instructions executed from External PROGRAM memory are disabled from fetching code bytes from the internal memory. All further programming of the EPROM is disabled. When, in addition to the above, security bits 1 and 2 are programmed, verify mode is disabled. When all three security bits are programmed, all of the conditions above apply and all External PROGRAM memory execution is disabled. (See Table 21).

Table 21. PROGRAM Security Bits

PROGRAM LOCK BITS				
	SB1	SB2	SB3	PROTECTION DESCRIPTION
1	U	U	U	No PROGRAM Security features enabled.
2	P	U	U	MOVC instructions executed from External PROGRAM memory are disabled from fetching code bytes from internal memory and further programming of the EPROM is disabled.
3	P	P	U	Same as 2, also verify is disabled.
4	P	P	P	Same as 3, External execution is disabled. Internal DATA RAM is not accessible.

NOTES:
1. P – programmed. U – unprogrammed.
2. Any other combination of the security bits is not defined.

Location of Sample Point
The location of the sample point within a bit period is determined according to the following:



- tSYNCSEG = tSCL
- tSEG1 = tSCL ∗ (8 ∗ tSEG1.3 + 4 ∗ tSEG1.2 + 2 ∗ tSEG1.1 + tSEG1.0 + 1)
- tSEG2 = tSCL ∗ (4 ∗ tSEG2.2 + 2 ∗ tSEG2.1 + tSEG2.0 + 1)

where tSEG1.3 – tSEG1.0 and tSEG2.2 – tSEG2.0 are bits in CANBTR.

Synchronization Jump Width
To compensate for phase shifts between clock oscillators of different bus controllers, any bus controller must re–synchronize on any relevant signal edge of the current transmission. The Synchronization Jump Width defines the maximum number of CAN System Clock cycles that a bit period may be shortened or lengthened by one re–synchronization, and is given by the following expression:

- tSJW = tSCL ∗ (2 ∗ SJW.1 + SJW.0 + 1)

where SJW.1 and SJW.0 are bits in CANBTR.

CANBTR: CAN Bus Timing Register
- Address: MMR base + 272h
- Access: Read, Write during reset mode only. Word access only.
- Reset value: 0000h

CANBTR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SAM	TSEG2.2	TSEG2.1	TSEG2.0	TSEG1.3	TSEG1.2	TSEG1.1	TSEG1.0	SJW.1	SJW.0	BRP.5	BRP.4	BRP.3	BRP.2	BRP.1	BRP.0

CAN Command and Status Registers

Two Modes in CAN Core Operation
The CCB has two different modes of operation: Reset mode, and Operation mode. On hardware reset, the CAN core is in Reset mode, and the RR bit of CANCMR (**CAN C**o**m**mand **R**egister) will be set. The User application would usually set up registers, etc., then put the CCB into Operation mode by clearing the RR bit.

While in Operation mode, the following conditions will cause the RR bit to be set, putting the CCB back into Reset mode:

- Tx Buffer Underflow (TBUF)
- Bus Off
- Hardware reset
- Test mode (Refer to XA-C3 User Guide, Sections 2.2.2.1 and 2.7.1.2)

CANCMR: CAN Command Register
- Address: MMR base + 270h
- Access: Read/Write, no R/M/W, Byte or Word Access. Hardware can set bit 0.
- Reset value: 01h

CANCMR

7	6	5	4	3	2	1	0
RXP	ST	LO	Reserved	SLPEN	OC1	Reserved	RR

RXP	Rx Polarity, writable during reset mode only. 0 = non–inverted, 1 = inverted.
ST	Self test, disable TxACK
LO	Listen only
Reserved	Reserved bit.
SLPEN	CTL will go back to idle if no interrupt is generated.
OC1	Output control for Tx pad. 0 = Push–Pull, 1 = Open Drain
Reserved	Reserved bit
RR	Reset Request.

CANSTR: CAN Status Register
- Address: MMR base + 271h
- Access: Read only, no write, no R/M/W. Byte access OK. Hardware can set or clear bits 7 – 2.
- Reset value: 00h

CANSTR

7	6	5	4	3	2	1	0
BS	EP	EW	TS	RS	SLPOK	–	–

BS	Bus status
EP	Error passive
EW	Error warning
TS	Transmit status
RS	Receive status
SLPOK	CAN status: no CAN bus activity and no pending core interrupts

CAN/CTL MESSAGE HANDLER

Message Objects

The XA-C3 supports 32 independent Message Objects, each of which can be either a transmit or a receive object. A receive object can be associated either with a unique CAN ID, or with a set of CAN IDs which share certain ID bit fields.

Each Message Object has access to its own block of data memory space, which is known as the object's message buffer. Both the size and base address of an object's message buffer is programmable. However, all message buffers must reside in the same 64Kbyte segment of data memory, as the contents of a single register (MBXSR…Message Buffer and XRAM Segment Register) are used to form the most significant byte of all 24–bit message buffer addresses.

Each Message Object is associated with a set of eight MMRs dedicated to that object. Some of these registers function differently for Tx than they do for Rx objects. The names of the eight MMRs are

1. MnMIDH – Message n Match ID High
2. MnMIDL – Message n Match ID Low
3. MnMSKH – Message n Mask High
4. MnMSKL – Message n Mask Low
5. MnCTL – Message n Control
6. MnBLR – Message n Buffer Location Register
7. MnBSZ – Message n Buffer Size
8. MnFCR – Message n Fragment Count Register

where n ranges from 0 to 31. In general, setting up a Message Object involves configuring some or all of its eight MMRs. Additionally, there are several MMRs whose bits control global parameters that apply to all objects. Table 22 summarizes the eight Message Object MMRs and their functions for receive and transmit objects. Details can be found in the sections that follow.

Table 22. Message Object Register Functions for Tx and Rx

Message Object Register (n = 0 – 31)	Rx Function	Tx Function	Address Offset
MnMIDH	Match ID* [28:13]	CAN ID [28:13]	n0h
MnMIDL	Match ID* [12:0][IDE][–][–]	CAN ID [12:0][IDE][–][–]	n2h
MnMSKH	Mask [28:13]	DLC	n4h
MnMSKL	Mask [12:0][–][–][–]	Not used	n6h
MnCTL	Control	Control	n8h
MnBLR	Buffer base address [a15:a0]	Buffer base address [a15:a0]	nAh
MnBSZ	Buffer size	Buffer size	nCh
MnFCR	Fragmentation count**	Not used	nEh
* After reception, the actual incoming Screener ID (without regard to Mask bits) will be stored by hardware in MnMIDH and MnMIDL for the benefit of the User application.			
** Typically written to only by hardware. Exceptions are the CANopen and OSEK protocols in which the User application must also initialize this register.			

Receive Message Objects and the Receive Process

During reception, the XA-C3 will store the incoming message in a temporary (13–byte) buffer. Once it is determined that a complete, error–free CAN frame has been successfully received, the XA-C3 will initiate the acceptance filtering ("Mask and Match") process. If acceptance filtering produces a Match with an enabled receive object's Match ID, the message is stored by the DMA engine in that object's message buffer.

Acceptance Filtering

The XA-C3 will sequentially compare the 30–bit Screener ID extracted from the incoming frame to the corresponding Match ID values specified in the MnMIDH and MnMIDL registers for all currently enabled receive objects. Any of the bits which are Masked will be excluded from this comparison. Masking is accomplished on an object–by–object basis by writing a logic '1' in the desired bit position(s) in the appropriate MnMSKH or MnMSKL register.
Any screener ID bits which are not intended to participate in acceptance filtering for a particular object *must* be Masked by the User (e.g., ID bits 0 & 1 for a Standard CAN frame, and possibly one or both data bytes).
If the acceptance filter determines that there is a Match between the incoming frame and any enabled receive object, the contents of the frame will be stored, via DMA, into the designated message buffer space associated with that object. If there is a Match to more than one Message Object, the frame will be considered to have matched the one with the lowest object number.
To summarize, Acceptance Filtering proceeds as follows:

- The "Screener ID" field is extracted from the incoming CAN Frame. The Screener ID field is assembled differently for Standard and Extended CAN Frames.
- The assembled Screener ID field is compared to the Match ID fields of all enabled receive Message Objects.
- Any bits which an object has Masked (by having '1' bits in its Mask field) are not included in the comparison. That is, if there is a '1' in some bit position of an object's Mask field, the corresponding bit in the object's Match ID field becomes a don't care (i.e., *always* yields a Match with the Screener ID).
- If filtering in this manner produces a Match, the frame will be stored via the DMA engine in that object's message buffer. If there is a Match with more than one object, the frame will be considered to have matched the one with the lowest object number.

Screener ID Field for Standard CAN Frame

The following table shows how the Screener ID field is assembled from the incoming bits of a Standard CAN Frame, and how it is compared to the Match ID and Mask fields of Object n.

The Frame Info byte contains the following bits:

FRAME INFO

7	6	5	4	3	2	1	0
IDE	RTR	SEM1	SEM0	DLC.3	DLC.2	DLC.1	DLC.0

The actual incoming Screener ID which caused the Match can be retrieved from the MnMIDH and MnMIDL registers as shown in Figure 39.

MNMIDH

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

MNMIDL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3	ID.2	ID.1	ID.0	IDE	–	–

Figure 39. Retrieving the Screener ID for an Extended CAN Frame

Fragmented Message Assembly

Masking of the 11/29 bit CAN Identifier field by User software (but only the *actual* bits of the Identifier itself!) is disallowed for any Message Object which employs auto–Fragmentation assembly. The identifier which resulted in the Match is, therefore, known unambiguously and is not included in the receive buffer. If the software needs access to this information, it can retrieve it from the appropriate MnMIDH and MnMIDL registers.

As subsequent frames of a Fragmented message are received, the new data bytes are appended to the end of the previously received packets. This process continues until a complete multi–frame message has been received and stored.

If an object is enabled with FRAG = 1, under protocols DeviceNet, CANopen, and OSEK (Prtcl1 Prtcl0 ≠ 00), the first CAN frame data byte is used to encode Fragmentation information only. That byte will not be stored in the buffer area. The storage will start with the second data byte (Data Byte 2) and proceed to the end of the frame. See Figure 40.

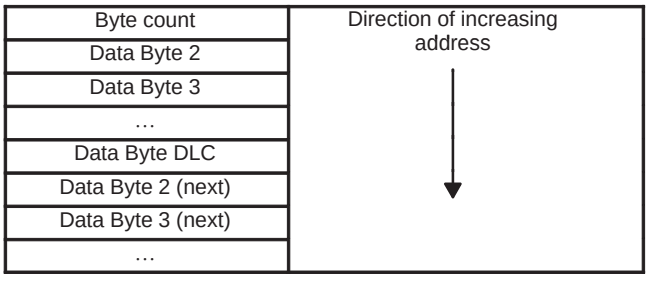


Figure 40. Memory Image for Fragmented CTL Messages (FRAG = 1 and Prtcl1 Prtcl0 ≠ 00)

If an object is enabled with FRAG = 1, with CAN as the system protocol (Prtcl1 Prtcl0 = 00), then CAN frames are stored sequentially in that object's message buffer using the format shown in . Also, if [Prtcl1 Prtcl0] = 00, Rx Buffer Full is defined as "less than 9 bytes remaining" after storage of a complete CAN frame. When the DMA pointer wraps around, it will be reset to offset '1' in the buffer, not offset '0', and there will be no Byte Count written.

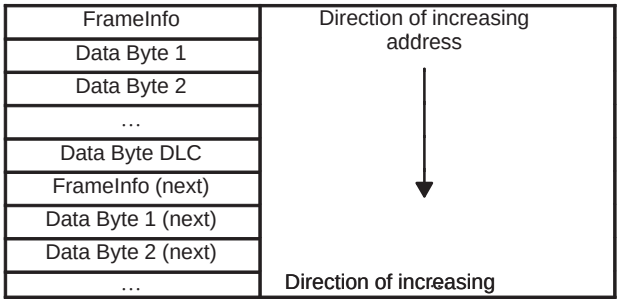


Figure 41. Memory Image for CAN Frame Buffering (FRAG = 1 and Prtcl1 Prtcl0 = 00)

During buffer access, the DMA will generate addresses automatically starting from the base location of the buffer. If the DMA has reached the top of the buffer, but the message has not been completely transferred to memory yet, the DMA will wrap around by generating addresses starting from the bottom of the buffer again. Some time before this happens, a warning interrupt will be generated so that the User application can take the necessary action to prevent data loss.

The top location of the buffer is determined by the size of the buffer as specified in MnBSZ.

The XA-C3 automatically receives, checks and reassembles up to 32 Fragmented messages automatically. When the FRAG bit is set on a particular message, the message handler hardware will use the Fragmentation information contained in Data Byte 1 of each frame.

To enable automatic Fragmented message handling for a certain Message Object, the User is responsible for setting the FRAG bit in the object's MnCTL register.

The message handler will keep track of the current address location and the number of bytes of each CTL message as it is being assembled in the designated message buffer location. After an "End of Message" is decoded, the message handler will finish moving the complete message and the byte count into the message buffer via

Table 24. Format for storing the CANopen Acknowledge byte

	7	6	5	4	3	2	1	0
Byte offset 0	scs			t = d.c.	X	X	X	X
Byte offset 1	Not used in the protocol							
Byte offset 2								
Byte offset 3								
Byte offset 4								
Byte offset 5								
Byte offset 6								
Byte offset 7								

MnFCR: Message n Fragmentation Count Register

- Address: MMR base + nEh
- Access: Read, write. Byte or word access.
- Reset Value: 00xxxxxxb (unused bits are always read as '0')

MNFCR

7	6	5	4	3	2	1	0
–	–	count					

An object's Fragmentation Count Register need not be configured by the User in DeviceNet systems. However, in CANopen and OSEK systems, the User must initialize this register.

GCTL: Global Control Byte (applies to all objects)

- Address: MMR base + 27Eh
- Access: Read, write, R/M/W, byte or word
- Reset Value: 00h

GCTL

7	6	5	4	3	2	1	0
–	–	–	–	Auto_Ack	Pre_Arb	Prtcl1	Prtcl0

Auto_Ack	Enables automatic acknowledge for CANopen. 0 = disable, 1 = enable.
Pre_Arb	Establishes the transmit pre–arbitration scheme. 0 = Pre–arbitration based on CAN ID, object number is secondary tie–breaker. 1 = Pre–arbitration based on object number only.
[Prtcl1 Prtcl0]	Indicates CTL protocol of the system (if any). 00 = CAN 01 = DeviceNet 10 = CANopen 11 = OSEK

Transmit Message Objects and the Transmit Process

In order to transmit a message, the XA application program needs to first assemble the complete message and store it in the message buffer area for that Message Object (the address of the message buffer would have been previously programmed into the object's MnBLR register). The header (CAN ID and Frame Information) must be written to the object's MnMIDH, MnMIDL, and MnMSKH registers as appropriate.

When the above is done, the Application is ready to transmit the message. To initiate a transmission, the object enable bit (OBJ_EN) must be set (except when transmitting an Auto–Acknowledge frame in CANopen). This will allow this ready–to–transmit message to participate in the pre–arbitration process.

If more than one message is ready to be transmitted. A so–called pre–arbitration process will be performed to determine which Message Object will be selected for transmission. There are two pre–arbitration policies which the User can choose between by setting or clearing the Pre_Arb bit in the GCTL register.

After a Tx Message Complete, the Tx Pre–Arbitration process is "reset", and begins again. Also, if the winning Message Object subsequently loses arbitration on the CAN bus, the Tx Pre–Arbitration process gets reset and begins again.

If there is only one transmit message whose OBJ_EN bit is set, it will be selected regardless of the pre–arbitration policy.

Pre–Arbitration Based on Priority (default mode)
This mode is selected by writing '0' to the Pre_Arb bit in GCTL[2].

The filter state machine goes through all transmit Message Objects for which the OBJ_EN bit is set. The message with the highest priority **as defined by the CAN arbitration ID field** will be selected for transmission. If more than one pending transmit message share the same CAN identifier, then secondary priority will be based on XA-C3 Message Object numbers, with the lowest numbered object winning access.

The winning message will then be output onto the CAN bus where it will compete for access with other transmitting nodes.

Pre–Arbitration Based on Object Number
As an alternative, the User may select to base pre–arbitration on Message Object number alone. This mode is selected by writing '1' to the Pre_Arb bit in GCTL[2].

The pre–arbitration state machine will go through the Message Objects sequentially, starting with object number 0, and select the first encountered transmit Message Object, with OBJ_EN set to '1', for transmission. In other words, the order in which the messages objects are examined in the pre–arbitration process is by increasing object number n, where n = 0…31. Each time pre–arbitration begins, the enabled message with the lowest object number will be selected

for transmission, regardless of the priority level represented by its CAN identifier.

Message Retrieval
Once a Message Object is selected for transmission, the DMA will begin retrieving the data from the message buffer area in memory and transferring the data to the CAN core block for transmission.

The same DMA engine and address pointer logic is used for message retrieval of transmit messages as for message storage of receive messages. Message buffer location and size information is specified in the same way. Please refer to the section entitled *Message Storage* on page 41 for a complete description.

When a message is retrieved, it will be written to the CCB sequentially. During this process, the DMA will keep requesting the bus, reading from memory and writing to the CCB.

To prepare a message for transmission, the User application is required to put the message in the appropriate object's message buffer area in the format shown below:

Data Byte 0	Direction of increasing address
Data Byte 1	
Data Byte 2	
Data Byte 3	
Data Byte 4	
Data Byte 5	
Data Byte 6	
Data Byte 7	

Please observe that the CAN identifier field and frame info must *not* be included in the transmit buffer. The transmit logic retrieves this information from the appropriate MnMIDH, MnMIDL, and MnMSKH registers. The format for storing the frame information in the MnMSKH register is shown in Figure 42.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	x	x	DLC.3	DLS.2	DLC.1	DLC.0

Figure 42. Format for Storing the Tx Frame Info in MnMSKH

Transmission of Fragmented Messages
The XA-C3 does not handle the transmission of Fragmented messages in hardware. It is the User's responsibility to write each frame of a Fragmented message to the transmit buffer, enable the object for transmission, and wait for a completion before writing the next frame to the message buffer. The User application must therefore transmit multiple frames one by one until the whole message is transmitted.
However, by using multiple Tx objects whose object numbers increase sequentially, and whose CAN IDs have been configured identically, several frames of a Fragmented message can be queued–up and enabled, and will be transmitted in order.

RTR Handling

This section describes how to receive or transmit Remote Transmit Request (RTR) frames.

Receiving an RTR Frame
1. The software must setup an Rx object with the RTR bit in MnCTL[0] set to '1'.
2. An RTR frame is received when the CAN ID Matches that of the enabled receive object whose RTR bit set to '1'.
3. If interrupt is enabled for that Message Object, an interrupt will be generated upon the RTR message reception.
4. The software would usually have a transmit object available with the same ID. Upon receiving an RTR frame, the software should update the data for the corresponding transmit object and send it out.

Transmitting an RTR Frame
1. The software must setup a Tx object with the RTR bit in MnCTL[0] set to '1'.
2. The software sets the object enable bit (OBJ_EN) which will enable the object to participate in pre–arbitration.
3. After the object wins pre–arbitration, an RTR frame will be sent out with a '1' in the RTR bit position**.**
4. At the end of a successful RTR transmission, the OBJ_EN bit will be cleared. An interrupt could be generated if it is enabled.
5. It is possible for an incoming message, with CAN ID Matching that of the transmitting RTR object, to arrive while the transmitting RTR object is in pre–arbitration, or even during transmission. In this case, the OBJ_EN bit of the transmitting RTR object will be cleared to '0', but no interrupt will be generated.

Data integrity issues

The data stored in the message buffer area can be accessed both by the CPU and by the DMA engine. Measures have been taken to ensure that the application does not read data from an object as it is being updated by the DMA. This is especially important if receive interrupts have been disabled or have not been responded to before a new message could have arrived. The general principle is,
- When DMA is accessing the buffer, the CPU should NOT attempt to read from and write to the buffer.
- When CPU is accessing the buffer, the DMA is still allowed to access the buffer. When this happens the CPU should be able to detect and abandon the data read.

Using the Semaphore Bits, SEM1 and SEM0
A three–state semaphore is used to signal whether a given buffer is:
1. Ready for CPU to read
2. Being accessed by DMA (therefore not ready for CPU read)
3. Being read by CPU

The semaphore is encoded by two semaphore bits, SEM1 and SEM0, which are in bit positions [5] and [4] of the Frame Info byte, the first byte of the receive buffer.

If the Receive Pre–Buffer overflows, the PBO status flag in FESTR[5] will be set, generating a Frame Error interrupt, if enabled. The PBO status flag is cleared by writing ‘1’ to the flag’s bit position.

Since this error will be generated *before* any acceptance filtering has been performed, there will be no Message Object number associated with the error (hence its inclusion under the category of frame error). Note that the new message being ignored may be intended for some other device on the CAN bus. This error should never occur unless there is a serious system–design problem (e.g., an off–chip device grabs the bus and fails to de–assert “WAIT” for an extended period).

Arbitration Lost
During transmission, arbitration on the CAN bus can be lost to a competing device with a higher priority CAN Identifier. In this case, the ARBLST status flag in FESTR[4] will be set, generating a Frame Error interrupt if enabled. The ARBLST status flag is cleared by executing a read of the Arbitration Lost Capture Register.

The bit position in the CAN Identifier at which arbitration was lost will be encoded and stored in the Arbitration Lost Capture Register (ALCR) for the benefit of the User application. The ALCR must be read by the CPU in order to be reactivated for capturing the next arbitration lost code, as well as to clear the ARBLST status flag. The bit position in the CAN ID is encoded and stored in the 5–bit field ALCR[4:0]. ALCR[7:5] are reserved, and are always read as zeros. The 5–bit number latched into ALCR is interpreted according to Table 26.

Table 26. Arbitration Lost Codes

ALCR[4:0]	Interpretation
0	Arbitration lost in ID28
1	Arbitration lost in ID27
2	Arbitration lost in ID26
...	...
10	Arbitration lost in ID18
11	Arbitration lost in SRR bit
12	Arbitration lost in IDE bit
13	Arbitration lost in ID17 (Extended Frame only)
...	...
30	Arbitration lost in ID0 (Extended Frame only)
31	Arbitration lost in RTR bit (Extended Frame only)

Error Warning
The EW bit in CANSTR[5] reports the error status of the core, with regard to the Error Warning Limit defined by the User. If EW is ‘0’, then both the Tx and Rx Error Counters contain values less than that stored in the Error Warning Limit Register. If either counter reaches or exceeds the value stored in the EWLR register, then the EW bit will be set to ‘1’. Subsequently if both counters decrement below the value stored in the EWLR register, the EW bit will be cleared to ‘0’.

The ERRW status flag in FESTR[1] will be set each time the EW bit in CANSTR[5] changes state, generating a Frame Error interrupt, if enabled. That is, both the 0–to–1 and the 1–to–0 transitions of the EW bit will cause the ERRW status flag to be set. The ERRW status flag is cleared by writing ‘1’ to the flag’s bit position.

Error Passive
The EP bit in CANSTR[6] reflects the Error Passive status of the core. If either the Tx or Rx Error Counter equals or exceeds the predefined value 128d, the EP bit will be set to ‘1’. Subsequently, if both counters decrement below 128d, the EP bit will be cleared to ‘0’.

Both 0–to–1 and 1–to–0 transitions of the EP bit will cause the ERRP status flag to be set, generating a Frame Error interrupt if enabled. The ERRP status flag is cleared by writing ‘1’ to the flag’s bit position in FESTR[0].

Bus Off
The BS (Bus Status) bit in CANSTR[7] reflects the Bus–On and Bus–Off status of the core. BS = 0 means the CAN core is currently involved in bus activity (Bus–On), while BS = 1 means it is not (Bus–Off).

When the Transmit Error Counter exceeds the predefined value 255d, the BS bit is set to ‘1’ (Bus–Off). In addition, the RR bit is set to ‘1’ (putting the CAN Core into Reset mode), and the BOFF status flag is set, generating a Frame Error interrupt if enabled. The Transmit Error Counter is preset to 127d, and the Receive Error Counter is cleared to 00h. The CAN Core will remain in this state until it is returned to Normal mode by clearing the RR bit.

Once the RR bit is cleared, the Tx Error Counter will decrement once for each occurrence of the Bus–Free signal (11 consecutive recessive bits). After 128 occurrences of Bus–Free, the BS bit is cleared (Bus–On). Again, the BOFF status flag is set (generating another Frame Error interrupt if enabled). At this point, both the Tx and Rx Error counters will contain the value 00h. At any time during the Bus–Off condition (BS = 1), the CPU can determine the progress of the Bus–Off recovery by reading the contents of the Tx Error Counter.

During Bus–Off, a return to Bus–On can be expedited under software control. If BS = 1, writing a value between 0 and 254 to the Tx Error Counter and then clearing the RR bit will cause the BS bit to be cleared after only 1 occurrence of the Bus–Free signal. As in the case above, on the 1–to–0 transition of the BS bit, the BOFF status flag will be set, generating another Frame Error interrupt if enabled.

The CPU can also initiate a Bus–Off condition, if the CAN Core is first put into Reset mode by setting RR = 1. Next, the value 255 is written to the Tx Error Counter, and the RR bit is cleared. With the core back in Normal mode, the Tx Error Counter contents are interpreted, and the Bus–Off condition proceeds as described above, exactly as if it had been caused by bus errors.

Note that the Tx Error Counter can only be written to when the CAN Core is in Reset mode, and that *both* 0–to–1 and 1–to–0 transitions of the BS bit will cause the BOFF status flag to be set, generating Frame Error interrupts if enabled.

CAN Interrupt Registers

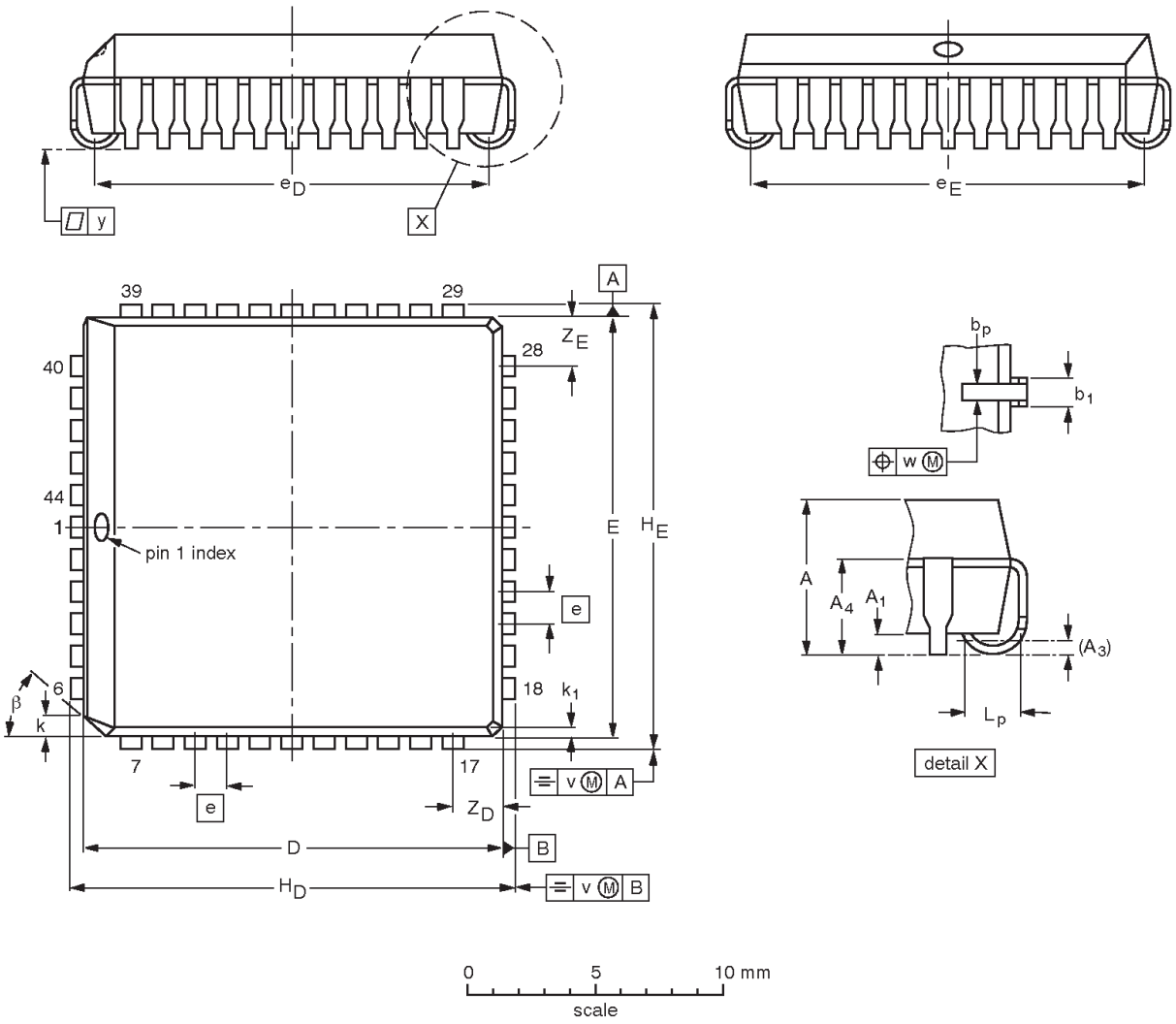
CANINTFLG (CAN Interrupt Flag Register)
- Address: MMR base + 228h
- Access: Read/Clear, byte or word
- Reset Value: 00h

CANINTFLG

7	6	5	4	3	2	1	0
–	–	–	FERIF	MERIF	RBFIF	TMCIF	RMCIF

FERIF — Frame Error Interrupt Flag (this bit is Read–Only, and must be cleared in FESTR)

PLCC44: plastic leaded chip carrier; 44 leads SOT187-2



DIMENSIONS (millimetre dimensions are derived from the original inch dimensions)

UNIT	A	A_1 min.	A_3	A_4 max.	b_p	b_1	$D^{(1)}$	$E^{(1)}$	e	e_D	e_E	H_D	H_E	k	k_1 max.	L_p	v	w	y	$Z_D^{(1)}$ max.	$Z_E^{(1)}$ max.	β
mm	4.57 4.19	0.51	0.25	3.05	0.53 0.33	0.81 0.66	16.66 16.51	16.66 16.51	1.27	16.00 14.99	16.00 14.99	17.65 17.40	17.65 17.40	1.22 1.07	0.51	1.44 1.02	0.18	0.18	0.10	2.16	2.16	45°
inches	0.180 0.165	0.020	0.01	0.12	0.021 0.013	0.032 0.026	0.656 0.650	0.656 0.650	0.05	0.630 0.590	0.630 0.590	0.695 0.685	0.695 0.685	0.048 0.042	0.020	0.057 0.040	0.007	0.007	0.004	0.085	0.085	

Note

1. Plastic or metal protrusions of 0.01 inches maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	EIAJ			
SOT187-2	112E10	MO-047AC				~~95-02-25~~ 97-12-16