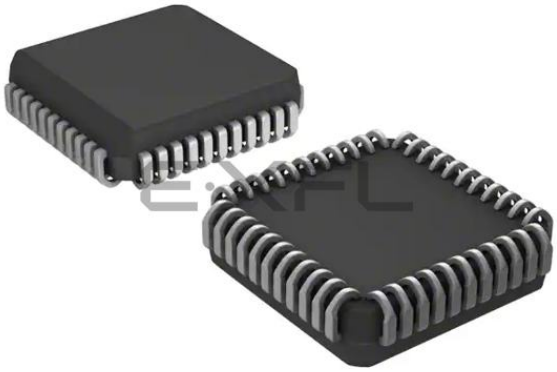


Welcome to E-XFL.COM

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-LCC (J-Lead)
Supplier Device Package	44-PLCC (16.59x16.59)
Purchase URL	https://www.e-xfl.com/product-detail/nxp-semiconductors/pxac37kfa-00-512

PIN CONFIGURATIONS

44-Pin PLCC Package

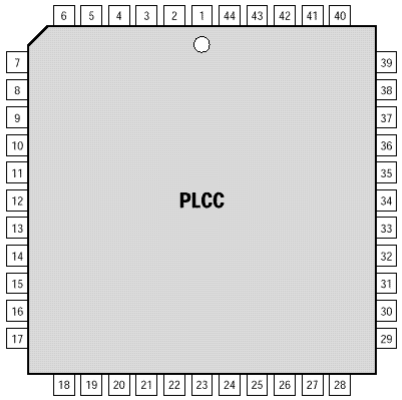


Figure 1. 44-pin PLCC package

Table 2. 44-pin PLCC package pin functions

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

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

44-pin LQFP package

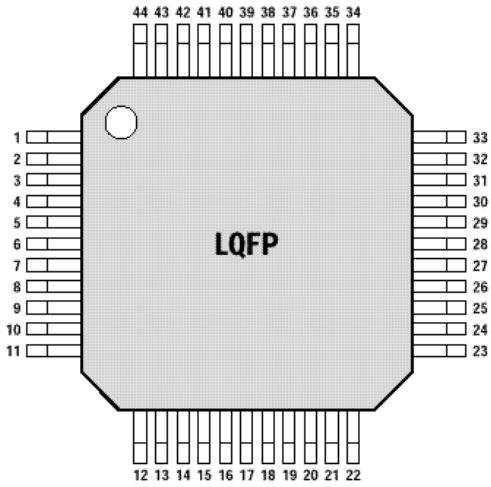


Figure 2. 44-pin PLCC package

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

BLOCK DIAGRAM

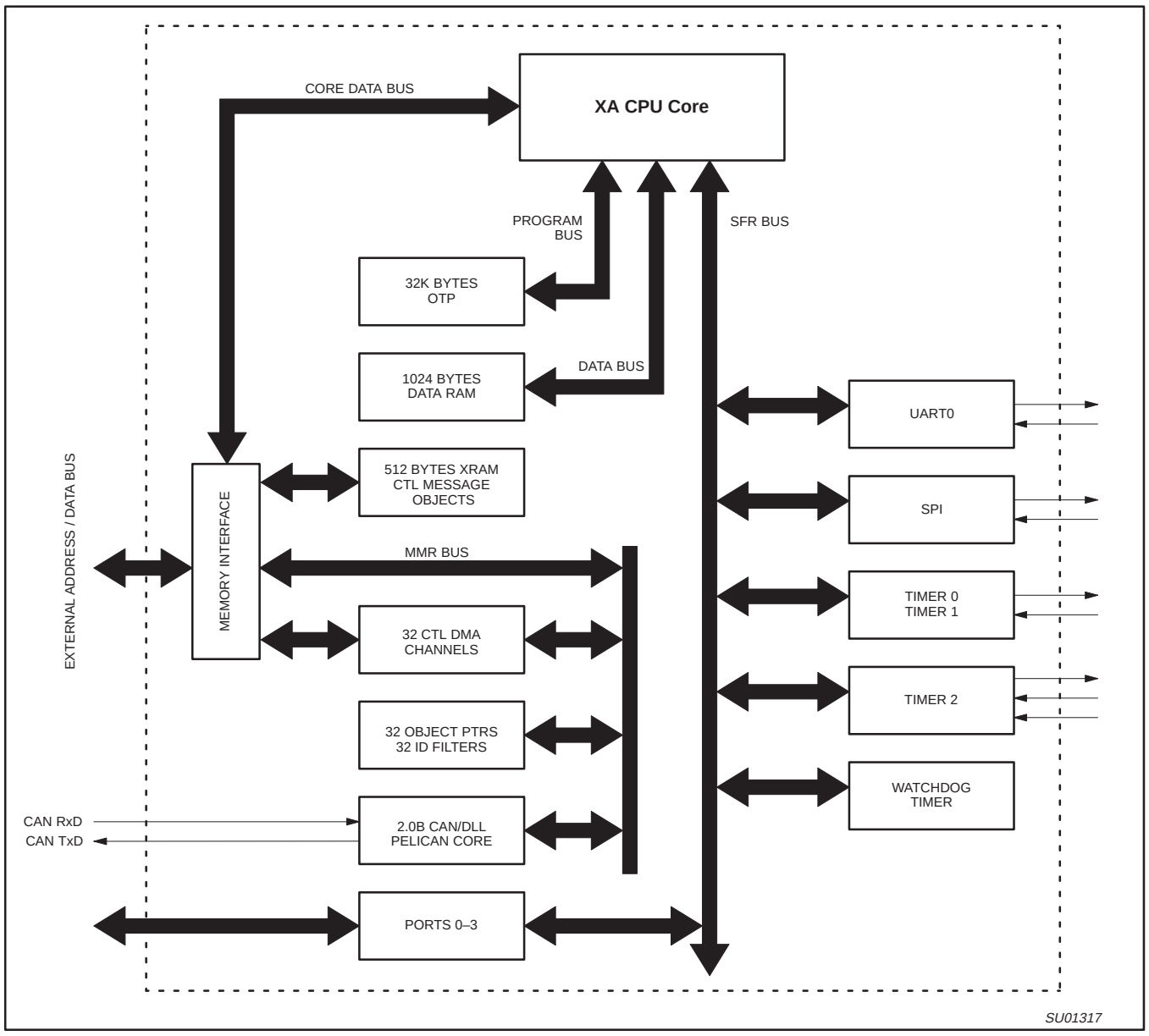


Figure 4. XA-C3 Simplified Block Diagram

PIN DESCRIPTIONS

Table 4. Pin Descriptions

MNEMONIC	PIN NUMBERS		TYPE	NAME AND FUNCTION
	PLCC	LQFP		
V_{SS}	1, 22	16, 39	I	Ground: 0V Reference.
V_{DD}	23, 44	17, 38	I	Power Supply: This is the power supply voltage for normal, Idle and Power–Down operation.
P0.0 – P0.7	43 – 36	37–30	I/O	Port 0: Port 0 is an 8–bit I/O Port with user –configurable pins. Port 0 latches have 1's written to them and are configured in the Quasi–Bidirectional mode during Reset. The operation of Port 0 pins as inputs or outputs depends upon the Port configuration selected. Each Port pin is configured independently. *Refer to the sections on I/O Port configuration and DC Electrical Characteristics for details.* **NOTE:** 2. When the External PROGRAM/DATA bus is used, Port 0 becomes the multiplexed low DATA/Instruction Byte and Address lines 4 through 11.
P1.0 – P1.7	2 – 9	40 – 44 1 – 3	I/O	Port 1: Port 1 is an 8–bit I/O Port with user –configurable pins. Port 1 latches have 1's written to them and are configured in the Quasi–Bidirectional mode during Reset. The operation of Port 1 pins as inputs or outputs depends upon the Port configuration selected. Each Port pin is configured independently. *Refer to the sections on I/O Port configuration and DC Electrical Characteristics for details.*
P1.0	2	40	O	WRH/: Address bit 0 of the External Address bus when the External DATA bus is configured for 8–bit width. When the External DATA bus is used, this pin becomes the High Byte Write Strobe (WRH).
P1.1	3	41	O	A1: Address bit 1 of the External Address bus.
P1.2	4	42	O	A2: Address bit 2 of the External Address bus.
P1.3	5	43	O	A3: Address bit 3 of the External Address bus.
P1.4	6	44	I	SPIRx: Receiver serial input of SPI.
P1.5	7	1	O	SPITx: Transmitter serial output of SPI.
P1.6	8	2	I	T2 ; SPICLK: Timer/counter 2 external clock input or Timer/counter 2 Clock–Out mode output, or SPI Clock output. **NOTES:** 3. SPICLK must be configured to idle in the logic '1' state in order to use either the T2 or P1.6 output functions, even if the SPI Port is not in use! 4. The default state from Reset of the SPICLK polarity is "inverted" which yields an SPICLK idle state of logic '1'. 5. If the SPI Clock polarity is changed by the user during SPI Port usage, it must be restored to "inverted" polarity before using either the P1.6 or Timer/counter 2 output functions.
P1.7	9	3	O	T2EX: Timer/counter 2 reload/capture/direction control.
P2.0 – P2.7	24 – 31	18 – 25	I/O	Port 2: Port 2 is an 8–bit I/O port with user–configurable pins. Port 2 latches have 1's written to them and are configured in the Quasi–Bidirectional mode during Reset. The operation of Port 2 pins as inputs or outputs depends upon the Port configuration selected. Each Port pin is configured independently. Refer to the sections on I/O port configuration and DC Electrical Characteristics for details. **NOTES:** 6. When the External 16–bit PROGRAM/DATA bus is used, Port 2 is MUXed between High (DATA/Instruction) Byte and Address lines 12 through 19.
P3.0 – P3.7	11, 13 – 19	5, 7 –12	I/O	Port 3: Port 3 is an 8–bit I/O Port with user–configurable pins. **NOTES:** 7. Port 3 latches have 1's written to them and are configured in the Quasi–Bidirectional mode during Reset. 8. The operation of Port 3 pins as inputs or outputs depends upon the Port configuration selected. 9. Each Port pin is configured independently. Refer to the sections on I/O Port configuration and DC Electrical Characteristics for details.
P3.0	11	5	I	RxD0: Receiver serial input of UART 0.
P3.1	13	7	O	TxD0: Transmitter serial output of UART 0.
P3.2	14	8	I	INT0/: External interrupt 0 input.
P3.3	15	9	I	INT1/: External interrupt 1 input.

MNEMONIC	PIN NUMBERS		TYPE	NAME AND FUNCTION
	PLCC	LQFP		
P3.4	16	10	I/O	T0: Timer 0 External count input or Timer 0 Overflow output.
P3.5	17	11	I/O	T1 : Timer 1 External count input or Timer 1 Overflow output.
P3.6	18	12	O	WRL*/*: External DATA memory Low Byte Write Strobe.
P3.7	19	13	O	RD*/*: External DATA memory Read Strobe.
RST*/*	10	4	I	RESET*/*: **NOTE:** 10. A low on this pin resets the XA–C3, causing I/O Ports and peripherals to take on their default states, and the processor to begin execution at the Address contained in the Reset Vector. Refer to the Reset section for details.
ALE ; PROG*/*	33	27	I/O	Address Latch Enable ; Program Pulse*/*: **NOTES:** 11. A high output on the ALE pin signals External circuitry to latch the address portion of the multiplexed Address/DATA bus. 12. A pulse on ALE occurs only when needed to process an External bus cycle. During EPROM programming, this pin is used as the Program pulse input.
PSEN*/*	32	26	O	Program Store Enable*/*: This is the Read Strobe for External PROGRAM memory. **NOTES:** 13. When the microcontroller accesses External PROGRAM memory, PSEN*/* is driven low in order to enable memory devices. 14. PSEN*/* is only active when External code accesses are performed.
EA*/* ; WAIT ; V_{PP}	35	29	I	External Access*/* ; WAIT ; Programming Supply Voltage: **NOTES:** 15. The EA*/* input determines whether the internal PROGRAM memory of the XA–C3 is used for code execution. 16. The EA*/* pin is latched as the (External) Reset input is released and its value applied during later execution. When latched as a 0, External PROGRAM memory is used exclusively. When latched as a 1, internal PROGRAM memory will be used up to its limit, and External PROGRAM memory is used above that point. 17. After Reset is released, this pin takes on the function of a Bus WAIT input. If WAIT is asserted High during any External bus access, that cycle will be extended until WAIT is released. 18. During EPROM programming, this pin is also the programming supply voltage input.
CAN RxD	12	6	I	CAN Receive Data input: CAN serial receiver input to the SJA1000 PeliCAN core.
CAN TxD	34	28	O	CAN Transmit Data output: CAN serial transmitter output from the SJA1000 PeliCAN core.
XTAL1	21	15	I	Crystal 1: Input to the inverting amplifier used in the oscillator circuit and input to the internal clock generator circuits.
XTAL2	20	14	O	Crystal 2: Output from the oscillator amplifier.

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

WATCHDOG TIMER

The watchdog timer subsystem protects the system from incorrect code execution by causing a system Reset when the watchdog timer underflows as a result of a failure of software to feed the timer prior to the timer reaching its terminal count. It is important to note that the watchdog timer is running after any type of Reset and must be turned off by user software if the application does not use the watchdog function.

Watchdog Function

The watchdog consists of a programmable prescaler and the main timer. The prescaler derives its clock from the TCLK source that also drives timers 0, 1, and 2. The watchdog timer subsystem consists of a programmable 13–bit prescaler, and an 8–bit main timer. The main timer is clocked (decremented) by a tap taken from one of the top 8–bits of the prescaler as shown in Figure 14.

The clock source for the prescaler is the same as TCLK (same as the clock source for the timers). Thus the main counter can be clocked as often as once every 32 TCLKs (see Table 8). The watchdog generates an underflow signal (and is autoloaded from WDL) when the watchdog is at count 0 and the clock to decrement the watchdog occurs. The watchdog is 8 bits wide and the autoload value can range from 0 to FFh. (The autoload value of 0 is permissible since the prescaler is cleared upon autoload).

This leads to the following user design equations:

$t_{MIN} = t_{OSC} \times 4 \times 32$ (W = 0, N = 4)

$t_{MAX} = t_{OSC} \times 64 \times 4096 \times 256$ (W = 255, N = 64)

$t_D = t_{OSC} \times N \times P \times (W + 1)$

where

t_{OSC} is the oscillator period

N is the selected prescaler tap value

W is the main counter autoload value

P is the prescaler value from Table 8

t_{MIN} is the minimum watchdog time–out value (when the autoload value is 0)

t_{MAX} is the maximum time–out value (when the autoload value is FFh)

t_D is the design time–out value.

The watchdog timer is not directly loadable by the user. Instead, the value to be loaded into the main timer is held in an autoload register. In order to cause the main timer to be loaded with the appropriate value, a special sequence of software action must take place. This operation is referred to as feeding the watchdog timer.

To feed the watchdog, two instructions must be sequentially executed successfully. No intervening SFR accesses are allowed, so interrupts should be disabled before feeding the watchdog. The instructions should move A5h to the WFEED1 register and then 5Ah to the WFEED2 register. If WFEED1 is correctly loaded and WFEED2 is not correctly loaded, then an immediate watchdog Reset will occur. The program sequence to feed the watchdog timer or cause new WDCON settings to take effect is as follows:

```
clr     ea              ; disable global interrupts.
mov.b   wfeed1,#A5h     ; do watchdog feed part 1
mov.b   wfeed2,#5Ah     ; do watchdog feed part 2
setb    ea              ; re–enable global interrupts.
```

This sequence assumes that the XA interrupt system is enabled and there is a possibility of an interrupt request occurring during the feed sequence. If an interrupt was allowed to be serviced and the service routine contained any SFR access, it would trigger a watchdog Reset. If it is known that no interrupt could occur during the feed sequence, the instructions to disable and re–enable interrupts may be removed.

The software must be written so that a feed operation takes place every t_D seconds from the last feed operation. Some tradeoffs may need to be made. It is not advisable to include feed operations in minor loops or in subroutines unless the feed operation is a specific subroutine.

To turn the watchdog timer completely off, the following code sequence should be used:

```
mov.b   wdcon,#0        ; set WD control register to clear
                          WDRUN.
mov.b   wfeed1,#A5h     ; do watchdog feed part 1
mov.b   wfeed2,#5Ah     ; do watchdog feed part 2
```

This sequence assumes that the watchdog timer is being turned off at the beginning of the User's initialization code and that the XA interrupt system has not yet been enabled. If the watchdog timer is to be turned off at a point when interrupts may be enabled, instructions to disable and re–enable interrupts should be added to this sequence.

Watchdog Control Register (WDCON)

The Reset values of the WDCON and WDL registers will be such that the watchdog timer has a timeout period of $4 \times 4096 \times t_{OSC}$ and the watchdog is running. WDCON can be written by software but the changes only take effect after executing a valid watchdog feed sequence.

Table 8. Prescalar Select Values in WDCON

PRE2	PRE1	PRE0	DIVISOR
0	0	0	32
0	0	1	64
0	1	0	128
0	1	1	256
1	0	0	512
1	0	1	1024
1	1	0	2048
1	1	1	4096

Watchdog Detailed Operation

When External Reset is applied, the following takes place:

- Watchdog run control bit set to ON (1).
- Autoload register WDL set to 00 (min. count).
- Watchdog time–out flag cleared.
- Prescaler is cleared.
- Prescaler tap set to the highest divide.
- Autoload takes place.

When coming out of a hardware Reset, the software should load the autoload register and then feed the watchdog (i.e., cause an autoload).
If the watchdog is running and happens to underflow at the time the External Reset is applied, the watchdog time–out flag will be cleared.

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

DC ELECTRICAL CHARACTERISTICS

Table 19. DC Electrical Characteristics

V_{DD} = 4.5V to 5.5V unless otherwise specified;

$T_{ambient}$ = 0 to +70°C for commercial, –40°C to +85°C for industrial, unless otherwise specified.

SYMBOL	PARAMETER	TEST CONDITIONS	LIMITS			UNIT
			MIN	TYP	MAX	
Supply Currents						
I_{DD}	Supply current, operating mode	32 MHz		54	80	mA
I_{ID}	Supply current, Idle mode	32 MHz		25	30	mA
I_{PD}	Power–Down mode current			5	100	µA
I_{PDI}	Power–Down mode current (–40°C to +85°C)				150	µA
Inputs						
V_{RAM}	RAM keep–alive voltage	RAM keep–alive voltage	1.5			V
V_{IL}	Input Low voltage		–0.5		$0.22V_{DD}$	V
V_{IH}	Input High voltage, except XTAL1, RST/	At 5.0V	2.2			V
V_{IH1}	Input High voltage to XTAL1, RST/	At 5.0V	$0.7V_{DD}$			V
V_{OL}	Output Low voltage all ports, ALE, PSEN/[3]	I_{OL} = 3.2mA, V_{DD} = 5.0V			0.5	V
V_{OH1}	Output High voltage all ports, ALE, PSEN/[1]	I_{OH} = –100mA, V_{DD} = 4.5V	2.4			V
V_{OH2}	Output High voltage, ports P0–3, ALE, PSEN/[2]	I_{OH} = 3.2mA, V_{DD} = 4.5V	2.4			V
C_{IO}	Input/Output pin capacitance				15	pF
I_{IL}	Logical 0 Input current, P0–3[6]	V_{IN} = 0.45V		–25	–75	µA
I_{LI}	Input Leakage current, P0–3[5]	V_{IN} = V_{IL} or V_{IH}			±10	µA
I_{TL}	Logical 1–to–0 Transition current — all ports[4]	At 5.5V			–650	µA
CAN RxD						
V_{IL}	Input Low voltage		–0.5		$0.22V_{DD}$	V
V_{IH}	Input High voltage	V_{DD} = 5.0V	2.2			V
C_I	Input pin capacitance				15	pF
I_{IL}	Logical 0 Input current	V_{IN} = 0.45V		–25	–75	µA
I_{LI}	Input Leakage current	V_{IN} = V_{IL} or V_{IH}			±10	µA
CAN TxD						
V_{OL}	Output Low voltage	I_{OL} = 3.2mA, V_{DD} = 5.0V			0.5	V
V_{OH}	Output High voltage	I_{OH} = –100mA, V_{DD} = 4.5V	2.4			V
C_O	Output capacitance				15	pF
I_{TL}	Logical 1–to–0 Transition current	V_{DD} = 5.5V			–650	µA

NOTES:

1. Ports in Quasi–Bidirectional mode with weak pull–up (applies to ALE, PSEN/ only during Reset operations).
2. Ports in Push–Pull mode, both pull–up and pull–down are assumed to be of the same strength
3. In all output modes
4. Port pins source a transition current when used in Quasi–Bidirectional mode and externally driven from 1 to 0. This current is highest when V_{IN} is approximately 2V.
5. Measured with port in high–impedance output mode.
6. Measured with port in Quasi–Bidirectional output mode.
7. Load capacitance for all outputs=80pF.
8. Under steady state (non–transient) conditions, I_{OL} must be externally limited as follows:
   Maximum I_{OL} per port pin: 15mA (*NOTE: This is 85°C specification for VDD = 5V.)
   Maximum I_{OL} per 8–bit port: 26mA
   Maximum total I_{OL} for all outputs: 71mA
   If I_{OL} exceeds the test condition, V_{OL} may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.
9. See Figures 29, 30, 32, and 33 for I_{DD} test conditions, and Figure 31 for I_{CC} vs. Frequency.

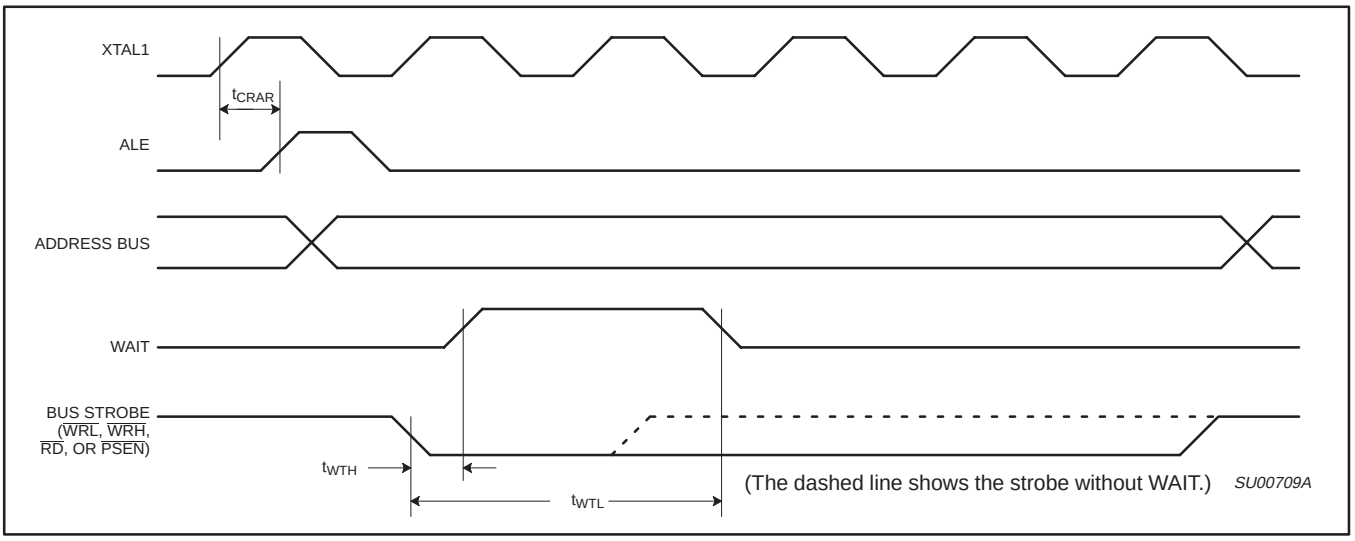


Figure 25. WAIT Signal Timing

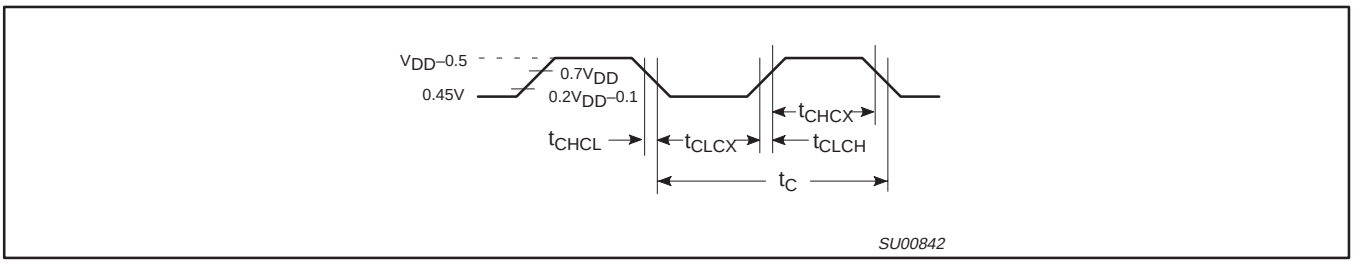


Figure 26. External Clock Drive

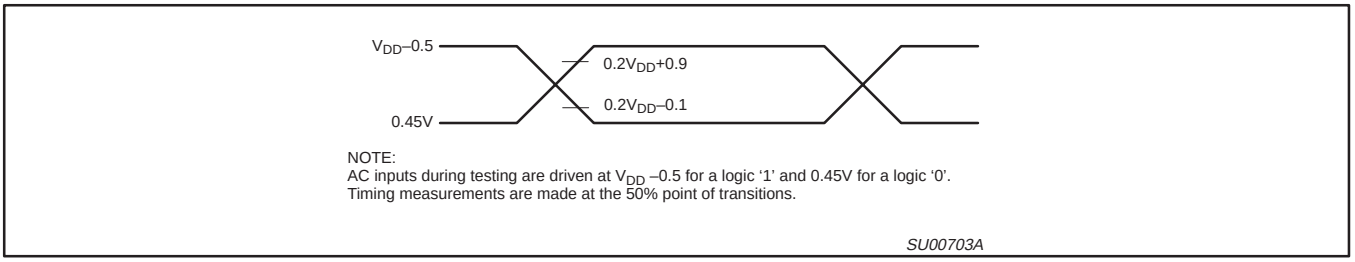


Figure 27. AC Testing Input/Output

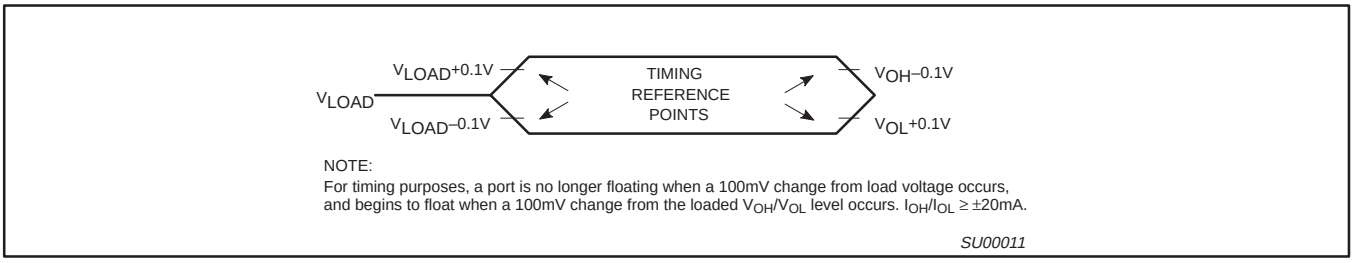


Figure 28. Float Waveform

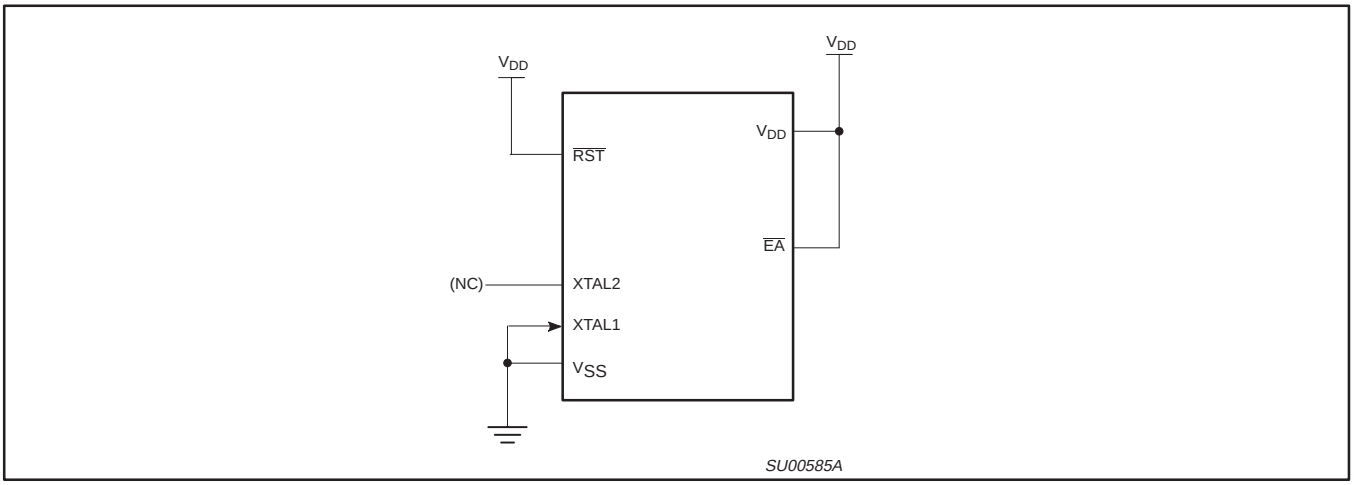


Figure 33. I_{DD} Test Condition, Power-Down Mode

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.

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.

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.

At the start of a non–Fragmented message, prior to writing any data bytes, the DMA will begin by writing 01h into the first byte of the buffer (byte 0). Once the complete frame has been stored, the DMA will write the frame information into byte 0, with bits [5] and [4] always set to '1'.

When the application wants to read from the object's buffer, it can read byte 0 to determine if the DMA is currently updating the buffer. If byte 0 contains 01h, then the buffer is currently being updated. The application should not continue to read from the buffer.

When the application starts to read from the buffer, it should set the semaphore to 10b. After reading is finished, the application should check the semaphore again. If it is still 10b, everything is OK.

If, however, the semaphore becomes 01b or 11b after the CPU access is finished, it means that either the buffer is currently being accessed by DMA or has been accessed by DMA during the time the CPU was performing reads. In either case, the CPU should wait until the semaphore bits become 11b again, and reread.

Use of the semaphore bits is not mandatory. However, their use may help to maintain data consistency.

There are no dedicated semaphore bits for use with Fragmented messages. In the case of a Fragmented message (in DeviceNet only), the DMA will write a 00h in byte 0 of the object's buffer. After the completion of a CTL message, the byte count (1 to 255) will be written to byte 0.

Avoiding Data Corruption for Transmit Message Objects
To avoid data corruption when transmitting messages, there are three possible approaches:

1. If the Message Complete interrupt is enabled for the transmit message, the User application would write to the transmit buffer after seeing the interrupt. Once the interrupt flag is set, it is known for sure that the pending message has already been transmitted.
2. Wait until OBJ_EN clears before writing to the buffer. This can be done by polling the OBJ_EN bit.
3. Clear OBJ_EN, while the object is still in pre–arbitration.

In the first two cases, the pending message will be transmitted completely before the next message gets sent. For the third case, the message will not be transmitted. Instead, a message with new content will enter pre–arbitration.
There is an additional mechanism that prevents corruption of a message that is being transmitted. If a transmission is ongoing for a Message Object, the XA-C3 hardware will prevent the User from clearing the OBJ_EN bit in the object's MnCTL register.

OSEK, DEVICENET, AND CANOPEN FRAMES OF INTEREST

OSEK ConsecutiveFrame

Data Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2 – DLC	User Data							
1	0	0	1	0	SN			

DeviceNet I/O Message

Data Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2 – DLC	User Data							
1	Fragment Type		Fragment Count					

Fragment Type = 00
- Fragment Count = 0 ... This is the First Fragment
- Fragment Count = 3F ... This is both the First and Last Fragment

Fragment Type = 01 ... Middle Fragment
Fragment Type = 10 ... Last Fragment

CANopen Download Domain Segment Request

Data Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2 – DLC	User Data							
1	ccs (User specified)			t	n (User specified)			c

c = 0 ... not last segment

c = 1 ... last segment

CANopen Auto–Acknowledge Tx Response to Download Domain Segment

Data Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2 – 8	reserved							
1	scs (User specified)			t	not used, always 0000			

CAN/CTL RELATED INTERRUPTS

The CAN/CTL module will generate five different Event interrupts to the XA core:

- Rx Message Complete
- Tx Message Complete
- Rx Buffer Full
- Message Error
- Frame Error

Rx and Tx Message Complete Interrupts

In the following discussion (and elsewhere in the document) the term "message" applies to a complete transfer of information. For single–frame messages, the "message complete" condition occurs at the end of the frame. For multi–frame (Fragmented) messages, message complete occurs after the last frame is received and stored. Since the hardware doesn't recognize or handle Fragmentation for transmit messages, the Tx message complete

Message Error Interrupt

There are two possible sources of a Message Error Interrupt: Tx Buffer Underflow, and Fragmentation Error. When either of these conditions occur for any Message Object, the Message Error Interrupt Flag (CANINTFLG[3]) will be set. In addition, the **M**essage **E**rror **I**nfo **R**egister (MEIR) will be updated to reflect the number of the object which suffered the message error, and the specific type of message error encountered (Tx Buffer Underflow or Fragmentation Error).

The MERIF interrupt flag is cleared by writing '1' to the flag's position in CANINTFLG[3].

Tx Buffer Underflow

This condition occurs when the transmit engine is "starved" due to the inability of the DMA to gain access to the bus. This interrupt condition is predominantly for system debugging. It should never occur during normal operation unless there is a serious flaw in the system (e.g., a peripheral which asserts the WAIT signal for an extended period).

Fragmentation Error

Fragmentation Error is an out–of–sequence Fragment count. For each successive frame of a Fragmented message, the new Fragment count must equal the previous count plus one. For DeviceNet, the Fragment count field is 6 bits wide, for OSEK it is 4 bits wide, and for CANopen it is merely a single, toggling bit.

If a new start–of–message indicator is received for an object, while the XA-C3 is already in the process of assembling a message for that object, the pointers for that object will be automatically reset and assembly will re–commence at the bottom of that object's message buffer. The previous, in–progress message will be overwritten, and no interrupt or error flag of any kind will be generated.

Frame Error Interrupt

There are six conditions generated from within the CAN core, any of which may cause the Frame Error Interrupt Flag (the FERIF bit in CANINTFLG[4]) to be set:

- Bus Error
- Pre–Buffer Overflow
- Arbitration Lost
- Error Warning
- Error Passive
- Bus Off
- Each condition has a corresponding status flag in the **F**rame **E**rror **St**atus **R**egister (FESTR), which will be set when that condition occurs. Each condition also has a corresponding enable bit in the **F**rame **E**rror **En**able **R**egister (FEENR). If a particular condition's enable bit is set, then when hardware sets that condition's status flag, the Frame Error Interrupt Flag will also be set. The Frame Error Interrupt Flag is cleared using a 2–step process:

1. The six individual Frame Error Status Flags in the FESTR register must first be cleared. Details on clearing these flags will be found in the following sections.
2. The FERIF bit can then be cleared by writing '1' to the flag's bit position in CANINTFLG[4].

Bus Error

When a Bus Error occurs, the BERR status flag in FESTR[3] will be set, generating a Frame Error interrupt, if enabled. The BERR status flag is cleared by executing a read of the Error Code Capture Register (ECCR).

The type and location of the error within the bit stream will be encoded and stored in the Error Code Capture register for the benefit of the User application. The ECCR register must be read by the CPU in order to be reactivated for capturing the next error code, as well as to clear the BERR status flag. Error codes in the ECCR register are interpreted as shown in Table 25. A read of the ECCR register should be executed before the Bus Error interrupt is enabled.

Table 25. Error Codes for the Error Code Capture Register (ECCR)

ECCR[7:6]	Interpretation
00	Bit Error
01	Form Error
10	Stuff Error
11	Other Error
ECCR[5]	Interpretation
0	Tx Error, error occurred during transmission
1	Rx Error, error occurred during reception
ECCR[4:0]	Interpretation
00011	Start of Frame
00010	ID28 … ID21
00110	ID20 … ID18
00100	SRR Bit
00101	IDE Bit
00111	ID17 … ID13
01111	ID12 … ID5
01110	ID4 … ID0
01100	RTR Bit
01101	Reserved Bit 1
01001	Reserved Bit 0
01011	Data Length Code
01010	Data Field
01000	CRC Sequence
11000	CRC Delimiter
11001	Acknowledge slot
11011	Acknowledge Delimiter
11010	End Of Frame
10010	Intermission (go buy popcorn)
10001	Active Error Flag
10110	Passive Error Flag
10011	Tolerate DOM bits
10111	Error Delimiter
11100	Overload Flag

Pre–Buffer Overflow

The XA-C3 stores one complete frame (which can be up to 13 bytes) in a receive "pre–buffer" while the previous frame is being processed. Even under extreme conditions, this should provide ample time for the previous frame to be written to memory by DMA. If for some reason the DMA is unable to gain access to the bus for a long period of time, the pre–buffer could overflow. In this event, the XA-C3 will stop accepting the new message. That is, once the five pre–buffer bytes are full, subsequent incoming bits will be ignored.

MERIF Message Error Interrupt Flag (cleared by writing ‘1’)

RBFIF Rx Buffer Full Interrupt Flag (cleared by writing ‘1’)

TMCIF Transmit Message Complete Interrupt Flag (should be cleared using the 2–step process described in the section entitled *Rx and Tx Message Complete Interrupts* on page 47).

RMCIF Receive Message Complete Interrupt Flag (should be cleared using the 2–step process described in the section entitled *Rx and Tx Message Complete Interrupts* on page 47

FESTR (Frame Error Status Register)

- Address: MMR base + 22Ch
- Access: Read, byte or word
- Reset Value: 00h

FESTR

7	6	5	4	3	2	1	0
–	–	PBO	ARBLST	BERR	BOFF	ERRW	ERRP

PBO Frame Error sub–type is Pre–Buffer Overflow (cleared by writing ‘1’)

ARBLST Frame Error sub–type is Arbitration Lost (cleared by reading the ALCR register)

BERR Frame Error sub–type is Bus Error (cleared by reading the ECCR register)

BOFF Frame Error sub–type is Bus Off (cleared by writing ‘1’)

ERRW Frame Error sub–type is Error Warning (cleared by writing ‘1’)

ERRP Frame Error sub–type is Error Passive (cleared by writing ‘1’)

FEENR (Frame Error Enable Register)

- Address: MMR base + 22Eh
- Access: Read, byte or word
- Reset Value: 00h

FEENR

7	6	5	4	3	2	1	0
–	–	PBOE	ARBLSTE	BERRE	BOFFE	ERRWE	ERRPE

PBOE Pre–Buffer Overflow Enable (0 = disabled, 1 = enabled)

ARBLSTE Arbitration Lost Enable (0 = disabled, 1 = enabled)

BERRE Bus Error Enable (0 = disabled, 1 = enabled)

BOFFE Bus Off Enable (0 = disabled, 1 = enabled)

ERRWE Error Warning Enable (0 = disabled, 1 = enabled)

ERRPE Error Passive Enable (0 = disabled, 1 = enabled)

MCIR (Message Complete Info Register)

- Address: MMR base + 229h
- Access: Read, byte or word
- Reset Value: 00h

MCIR

7	6	5	4	3	2	1	0
–	–	1 or More	Object Number				

1orMore 0 = No objects whose INT_EN bits are set currently have a message complete condition. 1 = One or more objects whose INT_EN bits are set currently have a message complete condition.

Object Number These 5 bits encode the lowest object number (0 – 31) of all objects whose INT_EN bits are set **AND** who currently have a message complete condition. If there are no such objects (1orMore = 0), these bits will be 00000b.

MEIR (Message Error Info Register)

- Address: MMR base + 22Ah
- Access: Read, byte or word
- Reset Value: 00h

MEIR

7	6	5	4	3	2	1	0
TBU	FRAG	RBF	Object Number				

[TBU FRAG RBF] 001 = Most recent is Rx Buffer Full interrupt.
010 = Most recent is Fragmentation Error interrupt.
100 = Most recent is Tx Buffer Underflow interrupt.

Object Number These 5 bits encode the object number (0 – 31) of the Message Object experiencing the most recent Message Error (Tx Buffer Underflow, Fragmentation Error, or Rx Buffer Full) condition. If more than one object are encountering Message Errors, only the most recent object number will be available.

MCPLH (Message Complete Status Flags High)

- Address: MMR base + 226h
- Access: Read/Clear, byte or word
- Reset Value: 0000h

CAN Interrupt SFRs

As with all XA Event interrupts, the five CAN interrupts can be independently enabled, disabled, and prioritized using the interrupt control SFRs in the XA Core (see IEH, IEL, and IPA0 – IPA7 in Table 26 on page 50 and see Table 16 on page 26). Bit positions are given below in .

Table 27. SFR Interrupt Enable/Priority Bit Positions

NOTE: ALSO SEE TABLE 25 ON PAGE 49

SFR Name		SFR Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IEH		427	EMRI	EMTI	EMER	ECER	ESPI	unused	ETI0	ERI0
IEL		426	EA	unused	EBUFF	ET2	ET1	EX1	ET0	EX0
IPA0		4A0	–	PT0			–	PX0		
IPA1		4A1	–	PT1			–	PX1		
IPA2		4A2	–	PBUFF			–	PT2		
IPA4		4A4	–	PTI0			–	PRI0		
IPA5		4A5	–	PSPI			–	unused		
IPA6		4A6	–	PMER			–	PCER		
IPA7		4A7	–	PMRI			–	PMTI		

EMRI	Rx Message Complete interrupt enable.
EMTI	Tx Message Complete interrupt enable.
EMER	Message Error interrupt enable.
ECER	Frame Error interrupt enable.
ESPI	SPI Port Interrupt enable.
ETI0, ERI0	XA-C3 Serial Port 0 interrupt enable bits.
EBUFF	Rx Buffer Full interrupt enable.
EA, ET2, ET1, EX1, ET0, EX0	XA-C3 Enable All, Timer, and External interrupt enable bits.
PX0, PT0, PX1, PT1, PT2	XA-C3 External and Timer interrupt priority fields.
PBUFF	Rx Buffer Full interrupt priority field.
PRI0, PTI0	XA-C3 Serial Port 0 interrupt priority fields.
PSPI	SPI Port interrupt priority field.
PMRI	Rx Message Complete interrupt priority field.
PMTI	Tx Message Complete interrupt priority field.
PMER	Message Error interrupt priority field.
PCER	Frame Error interrupt priority field.

POWER–DOWN AND IDLE MODE

Background: XA Power–Down and Idle modes

Power–Down mode on the XA means that the main oscillator is clamped–off and there is no chip activity of any kind. I_{dd} in this mode is on the order of a few tens of microamps. Wake–up from power–down is accomplished via a system reset or a transition on the External Interrupt 0 or 1 pins. The wake–up period is 10,000 oscillator clocks (enough for several CAN frames to be transmitted).

Idle mode on the XA means that the clocks are running but are gated–off to the processor core. Most peripherals are active, but some may be put to sleep along with the core. Wake–up from Idle mode is instantaneous, and is initiated via any interrupt. I_{dd} in Idle mode is in the range of 25–30 mA @ 32 MHz if the CAN/CTL module is deactivated, perhaps 54–80 mA @ 32 MHz if the CAN is left active. Note that putting the XA core, by itself, into Idle mode reduces power consumption by approximately 30 mA @ 32MHz.

XA-C3 Idle Mode

The default condition for the CTL/CAN module will be to stay awake in Idle mode, so that the core can “sleep” while CAN transmissions/receptions are in progress. Any interrupt (e.g., Message Complete) will wake up the core. An option will be provided to include the CAN/CTL module in Idle mode. This option will be selected in software by writing to the SLPEN bit in MMR CANCMR[3]. If the CAN ***does*** go to sleep in Idle mode, then any transition on the CAN RxD input pin will be asynchronously latched and will immediately re–enable the clocks to the CAN/CTL module so that it can begin receiving the incoming frame. There will ***not*** be any interrupt generated, however, and the processor core will remain in idle mode. The CPU will only come out of Idle mode once a complete message is received and stored and a Message–Complete interrupt is generated (unless, of course, some other system interrupt wakes it up prior to that). The CCB will generate a “ccb_idle_n” signal which will be routed to all of the other CAN/CTL blocks (including the CMI) at the top level.

XA-C3 Power–Down Mode

If a transition of the CAN RxD input occurs when the XA-C3 is in Power–Down mode, the CPU will enter Idle mode (after a **9892** clock delay), and the CCB and Message Handler circuits will be activated to receive and process the incoming frame. When either of these blocks generates an interrupt (or some other enabled interrupt occurs), only then will the CPU come out of Idle mode and begin executing code. Code execution will resume either in the interrupt service routine, if its priority is higher than current code, or with the next instruction following the Power–Down instruction. At this time the termination of the Power–Down mode is actually complete.

CAN Sleep Enable

Certain conditions must be met before the CAN/CTL module can be safely put to sleep (Idle or Power–Down). Essentially, there must be no CAN activity in progress and no interrupts pending. The CCB must generate a “sleepok” signal (SLPOK=CANSTR[2]) which indicates that these conditions are met. This signal must be used to enable the “ccb_idle_n” signal. In addition, the “sleepok” signal

SPICFG

7	6	5	4	3	2	1	0
SPSTT	SPB2	SPB1	SPB0	SPFG	Rsvd	Rsvd	SPIDL

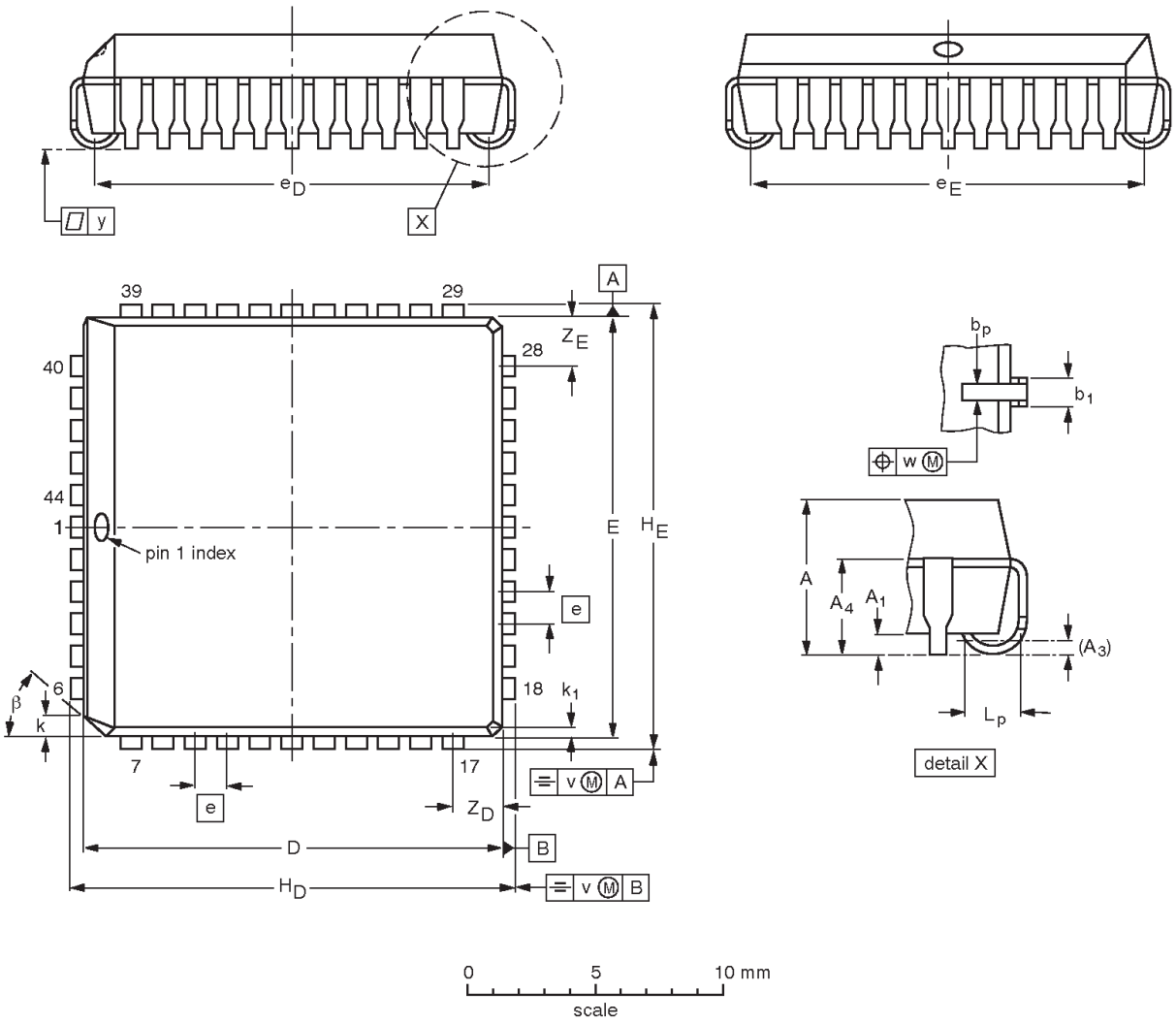
SPSTT — SPI Start
0 = Cycle finished, cleared by hardware and on reset
1 = Start

SPB2 – SPB0 — Number of SPI bits transceived = SPICFG[6:4] + 1

Rsvd — Reserved bits, write only zeros

SPIDL — SPI TxD idle state
0 = idle low
1 = idle high

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