DATA SHEET

XA-C3

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

Preliminary specification
Supersedes data of 1999 Dec 20

2000 Jan 25

Philips
Semiconductors

PHILIPS

LIST OF FIGURES

LIST OF TABLES

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

ORDERING INFORMATION

Table 1. Ordering Information

XA–C3 Type & Part Number	Temperature Range (degrees C)	Package Description	Operating Frequency (MHz)	Drawing Number
OTP				
PXAC37KBBD	0 to +70	Low Profile PQFP [LQFP44]	32	SOT389–1
PXAC37KBA	0 to +70	PLCC [PLCC44]	32	SOT187–2
PXAC37KFBD	–40 to +85	Low Profile PQFP [LQFP44]	32	SOT389–1
PXAC37KFA	–40 to +85	PLCC [PLCC44]	32	SOT187–2

PIN CONFIGURATIONS

44-Pin PLCC Package

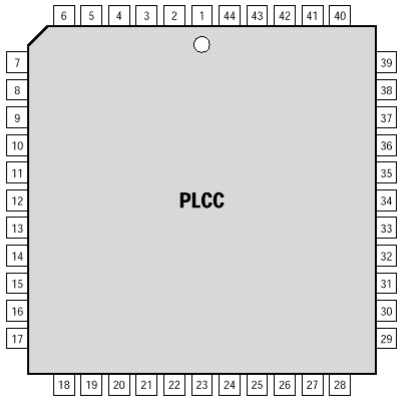


Figure 1. 44-pin PLCC package

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

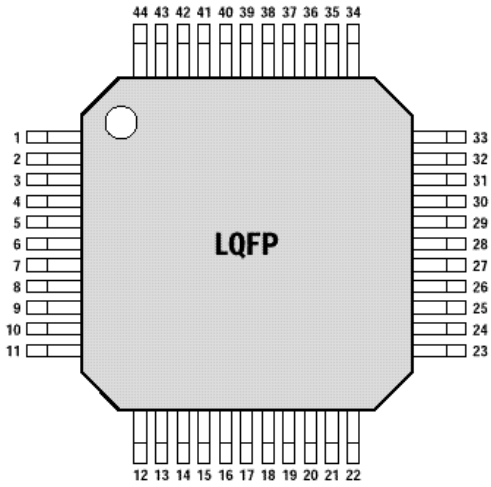


Figure 2. 44-pin PLCC package

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

LOGIC SYMBOL

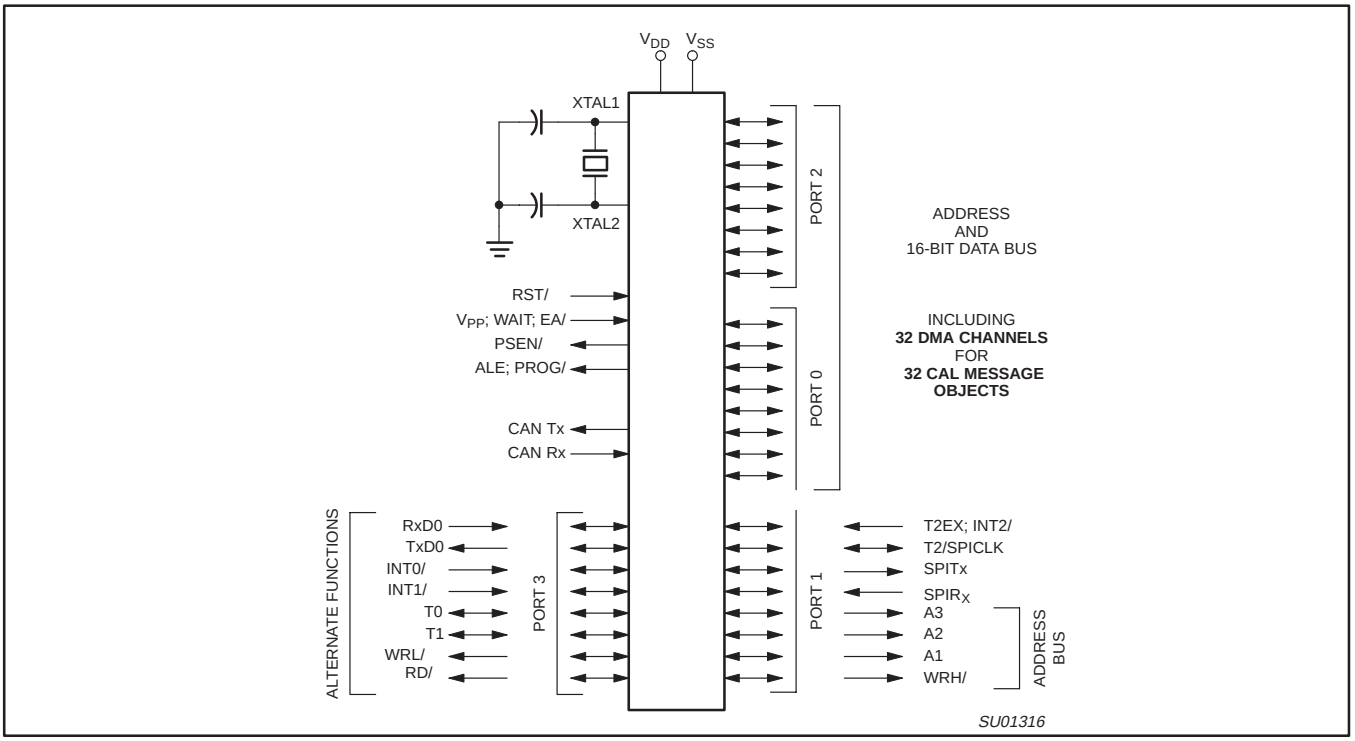


Figure 3. Logic Symbol

BLOCK DIAGRAM

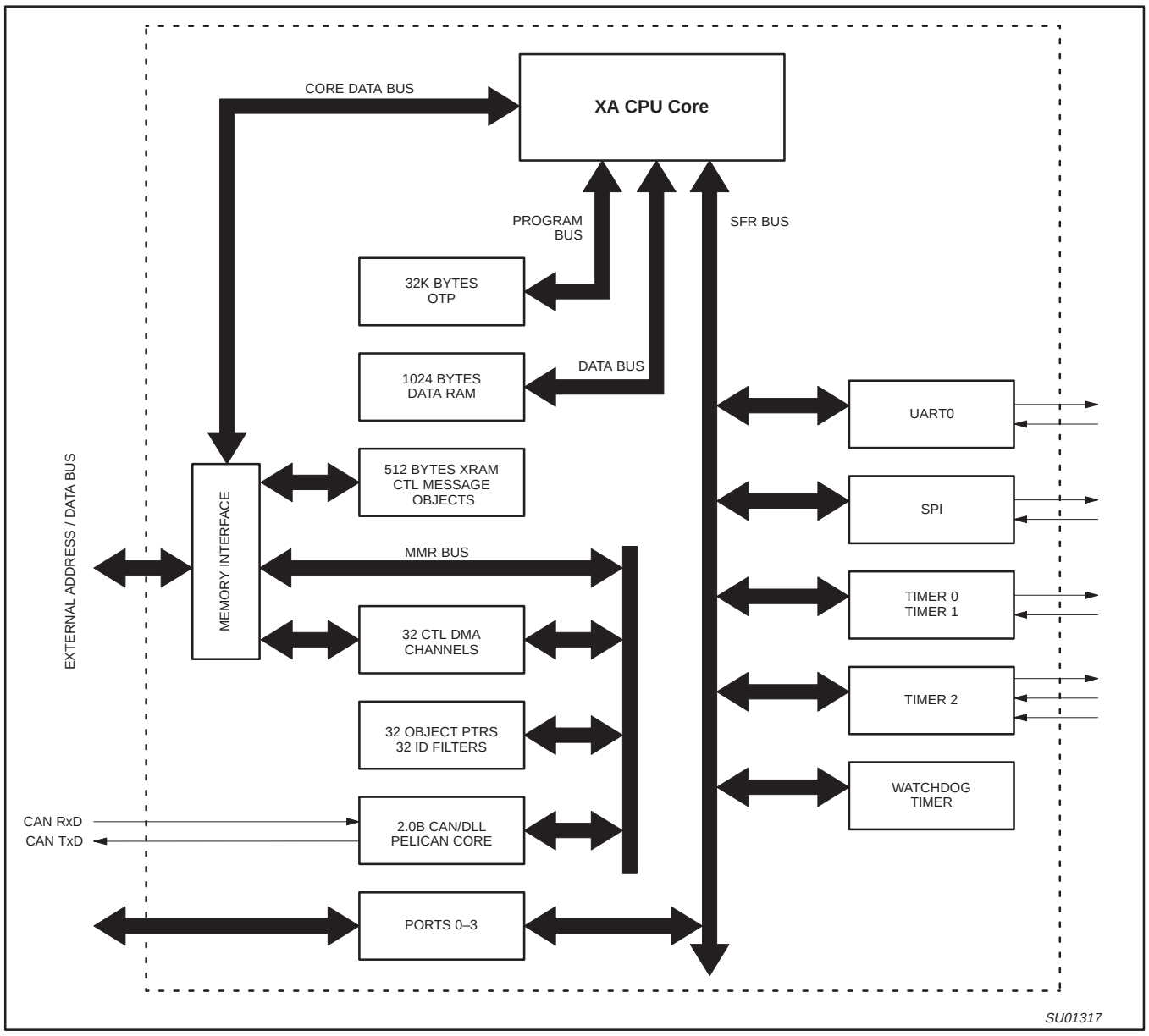


Figure 4. XA-C3 Simplified Block Diagram

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.

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

NAME	DESCRIPTION	SFR ADDRESS	BIT FUNCTIONS AND BIT ADDRESSES								RESET VALUE
			7	6	5	4	3	2	1	0	
			30F	30E	30D	30C	30B	30A	309	308	
S0STAT*	Serial port 0 extended status	421h	–	–	–	–	FE0	BR0	OE0	STINT0	00h
S0BUF	Serial port 0 buffer register	460h									xxh
S0ADDR	Serial port 0 address register	461h									00h
S0ADEN	Serial port 0 address enable register	462h									00h
SCR	System configuration register	440h	–	–	–	–	PT1	PT0	CM	PZ	00h
			21F	21E	21D	21C	21B	21A	219	218	
SSEL*	Segment selection register	403h	ESWEN	R6SEG	R5SEG	R4SEG	R3SEG	R2SEG	R1SEG	R0SEG	00h
SWE	Software Interrupt Enable	47Ah	–	SWE7	SWE6	SWE5	SWE4	SWE3	SWE2	SWE1	00h
			357	356	355	354	353	352	351	350	
SWR*	Software Interrupt Request	42Ah	–	SWR7	SWR6	SWR5	SWR4	SWR3	SWR2	SWR1	00h
			2C7	2C6	2C5	2C4	2C3	2C2	2C1	2C0	
T2CON*	Timer 2 control register	418h	TF2	EXF2	RCLK0	TCLK0	EXEN2	TR2	C2 or T2*I*	CP or RL2*I*	00h
			2CF	2CE	2CD	2CC	2CB	2CA	2C9	2C8	
T2MOD*	Timer 2 mode control	419h	–	–	–	–	–	–	T2OE	DCEN	00h
TH2	Timer 2 high byte	459h									00h
TL2	Timer 2 low byte	458h									00h
T2CAPH	Timer 2 capture register, high byte	45Bh									00h
T2CAPL	Timer 2 capture register, low byte	45Ah									00h
			287	286	285	284	283	282	281	280	
TCON*	Timer 0 and 1 control register	410h	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	00h
TH0	Timer 0 high byte	451h									00h
TH1	Timer 1 high byte	453h									00h
TL0	Timer 0 low byte	450h									00h
TL1	Timer 1 low byte	452h									00h
TMOD	Timer 0 and 1 mode control	45Ch	GATE1	C1 or T1*I*	M1	M0	GATE0	C0 or T0*I*	M1	M0	00h
			28F	28E	28D	28C	28B	28A	289	288	
TSTAT*	Timer 0 and 1 extended status	411h	–	–	–	–	–	T1OE	–	T0OE	00h
			2FF	2FE	2FD	2FC	2FB	2FA	2F9	2F8	
WDCON*	Watchdog control register	41Fh	PRE2	PRE1	PRE0	–	–	WDRUN	WDTOF	–	Note 6
WDL	Watchdog timer reload	45Fh									00h
WFEED1	Watchdog feed 1	45Dh									xxh
WFEED2	Watchdog feed 2	45Eh									xxh

NOTES:
1. Users should never write to the BCR register.
2. Users must ALWAYS INITIALIZE (Write) 00h to this register.
3. Port configurations default to Quasi–Bidirectional when the XA begins execution from Internal code memory after Reset, based on the condition found on the EA*I* pin. Thus, all PnCFGA registers will contain FFh and PnCFGB registers will contain 00h. When the XA begins execution using External code memory, the default configuration for pins that are associated with the External bus will be Push–Pull. The PnCFGA and PnCFGB register contents will reflect this difference.
4. SFR is loaded from the Reset vector.
5. All bits except F1, F0, and P are loaded from the Reset vector. Those bits are all 0.
6. The WDCON Reset value is E6h for a Watchdog Reset, E4h for all other Reset causes. The Watchdog is always turned ON as one consequence of RST*I*. Therefore, the user should turn OFF the Watchdog if immediate Watchdog operation is not desired: See the Watchdog Timer section in this Data Sheet for a recommended code example.

GENERAL NOTES:
– SFRs marked with an asterisk (*) are bit–addressable.
– The XA–C3 implements an 8–bit SFR bus, as stated in Chapter 8 of the *XA User Guide.* All SFR accesses must be 8–bit operations. Attempts to write 16 bits to an SFR will actually write only the lower 8 bits. Sixteen–bit SFR reads will return undefined data in the upper byte.
– Unimplemented bits in SFRs (indicated by "–"} are unknown at all times. Ones should not be written to these bits since they may be used for other purposes in future XA derivatives. In general, the Reset value shown for these unimplemented bits is 00h.
– The XA guards writes to all SFR bits that can be modified by hardware, including all SFR resident interrupt flags, as well as the WDTOF bit in WDCON. This mechanism, called Read–Modify–Write Lockout, prevents loss of an interrupt (or other status) flag if a bit is written to directly by hardware between the read and write of an instruction that performs a read–modify–write operation.

MEMORY-MAPPED REGISTERS

Table 6. Memory-Mapped Registers

Name	Description	Address Offset	Operation	ACCESS	Reset Value
MESSAGE OBJECT REGISTERS (n = 0 – 31)					
MnMIDH	Message n Match ID High	$000n_4n_3n_2n_1n_00000b$ (n0h)	R/W	Word only	xxxxh
MnMIDL	Message n Match ID Low	$000n_4n_3n_2n_1n_00010b$ (n2h)	R/W	Word only	x…x00b
MnMSKH	Message n Mask High	$000n_4n_3n_2n_1n_00100b$ (n4h)	R/W	Word only	xxxxh
MnMSKL	Message n Mask Low	$000n_4n_3n_2n_1n_00110b$ (n6h)	R/W	Word only	x…x000b
MnCTL	Message n Control	$000n_4n_3n_2n_1n_01000b$ (n8h)	R/W	Byte	00000xxxb
MnBLR	Message n Buffer Location	$000n_4n_3n_2n_1n_01010b$ (nAh)	R/W	Word only	xxxxh
MnBSZ	Message n Buffer Size	$000n_4n_3n_2n_1n_01100b$ (nCh)	R/W	Byte	00000xxxb
MnFCR	Message n Fragmentation Count	$000n_4n_3n_2n_1n_01110b$ (nEh)	R/W	Byte	00xxxxxxb
CAN/CTL INTERRUPT COMPLETE (CIC) REGISTERS					
MCPLH	Message Complete Status Flags High	226h	RC	Word	0000h
MCPLL	Message Complete Status Flags Low	224h	RC	Word	0000h
CANINTFLG	CAN Interrupt Flag Register	228h	RC	Byte	00h
MCIR	Message Complete Information	229h	RO	Byte	00h
MEIR	Message Error Information	22Ah	RO	Byte	00h
FEENR	Frame Error Enable	22Eh	R/W	Byte	00h
FESTR	Frame Error Status	22Ch	RC	Byte	00h
SPI REGISTERS					
SPICFG	SPI Configuration	260h	R/W	Byte	00h
SPIDATA	SPI Data	262h	R/W	Byte	00h
SPICS	SPI Control and Status	263h	R/W	Byte	00h
CAN CORE BLOCK (CCB) REGISTERS					
CANCMR	CAN Core Command	270h	R/W*	Byte	01h (Note 1)
CANSTR	CAN Core Status	271h	RO	Byte	00h
CANBTR	CAN Core Bus Timing	272h	R/W*	Word	0000h
TxERC	Tx Error Counter	274h	R/W*	Byte	00h
RxERC	Rx Error Counter	275h	R/W*	Byte	00h
EWLR	Error Warning Limit	276h	R/W	Byte	96h
ECCR	Error Code Capture	278h	RO	Byte	00h
ALCR	Arbitration Lost Capture	27Ah	RO	Byte	00h
GCTL	Global Control	27Eh	R/W	Byte	00h
MEMORY INTERFACE (MIF) REGISTERS					
MIFBTRH	MIF Bus Timing Register High	293h	R/W	Byte	FFh
MIFBTRL	MIF Bus Timing Register Low	292h	R/W	Byte	EFh
MBXSR	Message Buffer and XRAM Segment Register	291h	R/W	Byte	FFh
XRAMB	XRAM Base Address	290h	R/W	Byte	FEh

Possible Operations: R/W = Read & Write, RO = Read Only, RC = Read then Clear via a service routine, W* = Writable only while the CAN Core is in Reset mode, x = Undefined after Reset

NOTE:
1. SLPEN (Sleep Enable), CANCMR[3], is writable only when the CAN Core is in Normal mode.

XA-C3 TIMER/COUNTERS

The XA has two standard 16–bit enhanced Timer/Counters: Timer 0 and Timer 1. Additionally, it has a third 16–bit Up/Down timer/counter, T2. A central timing generator in the XA core provides the time–base for all XA Timers and Counters. The timer/event counters can perform the following functions:

– Measure time intervals and pulse duration

– Count External events

– Generate interrupt requests

– Generate PWM or timed output waveforms

All timer/counters (Timer 0, Timer 1 and Timer 2) can be independently programmed to operate either as timers or event counters. Timer 0 and Timer 1 are selectable via TMOD[6] and TMOD[2], respectively. Timer 2 is selectable via T2CON[1]. All timers may be dynamically read during program execution. All timers count up unless otherwise stated.

When running in timer mode (as opposed to counter mode) the base clock rate of all timers, including the Watchdog timer, is user–programmable. The clock driving the timers is called TCLK and is determined by the setting of two bits (PT1, PT0) (SCR[3:2]) in the System Configuration Register – See Table 5. The frequency of TCLK may be selected to be the oscillator input divided by 4 (f_C/4), the oscillator input divided by 16 (f_C/16), or the oscillator input divided by 64 (f_C/64). This gives a range of possibilities for the XA timer functions, including baud rate generation and Timer 2 capture. Note: This single SCR rate setting applies to all timers.

When timers T0, T1, or T2 are used in the counter mode, the timers will increment whenever a falling edge (high–to–low transition) is detected on an External clock pin. These inputs are sampled once every two oscillator cycles, so it can take as many as four oscillator cycles to detect a transition. Thus, the maximum count rate that can be supported is f_C /4. In general, the duty cycle of the timer clock inputs is not important. However, any high or low state on the timer clock input pins must be present for two oscillator cycles before it is guaranteed to be “seen” by the timer logic.

Timer 0 and Timer 1

These two Timer/Counters have four operating modes, which are selected by bit–pairs (M1, M0) in the TMOD register. Timer modes 1, 2, and 3 in XA are kept identical to the 80C51 timer modes for code compatibility. Only the mode 0 is replaced in the XA by a more powerful 16–bit auto–reload mode. This gives the XA timers a much larger range when used as time bases.
The recommended M1, M0 settings for the different modes are shown in Figure 6.

SCR Address:440
Not Bit Addressable
Reset Value: 00H

MSB							LSB
—	—	—	—	PT1	PT0	CM	PZ

PT1	PT0	OPERATING
		Prescaler selection.
0	0	Osc/4
0	1	Osc/16
1	0	Osc/64
1	1	Reserved
CM		Compatibility Mode allows the XA to execute most translated 80C51 code on the XA. The XA register file must copy the 80C51 mapping to data memory and mimic the 80C51 indirect addressing scheme.
PZ		Page Zero mode forces all program and data addresses to 16-bits only. This saves stack space and speeds up execution but limits memory access to 64k.

SU00589

Figure 5. System Configuration Register (SCR)

TMOD Address:45C
Not Bit Addressable
Reset Value: 00H

MSB							LSB
GATE	C1 or T1/	M1	M0	GATE	C0 or T0/	M1	M0
TIMER 1				TIMER 0			

	GATE	Gating control when set. Timer/Counter “n” is enabled only while “$\overline{\text{INTn}}$” pin is high and “TRn” control bit is set. When cleared Timer “n” is enabled whenever “TRn” control bit is set.
	C/$\overline{\text{T}}$	Timer or Counter Selector cleared for Timer operation (input from internal system clock.) Set for Counter operation (input from “Tn” input pin).

M1	M0	OPERATING
0	0	16-bit auto-reload timer/counter
0	1	16-bit non-auto-reload timer/counter
1	0	8-bit auto-reload timer/counter
1	1	Dual 8-bit timer mode (timer 0 only)

SU01325

Figure 6. Timer/Counter Mode Control (TMOD) Register

New Enhanced Mode 0

For timers T0 or T1 the 13–bit count mode on the 80C51 (current Mode 0) has been replaced in the XA with a 16–bit auto–reload mode. Four additional 8–bit data registers (two per timer: RTHn and RTLn) are created to hold the auto–reload values. In this mode, the TH overflow will set the TF flag in the TCON register (see Figure 7) and cause both the TL and TH counters to be loaded from the RTL and RTH registers respectively.

These new SFRs will also be used to hold the TL reload data in the 8–bit auto–reload mode (Mode 2) instead of TH.

The overflow rate for Timer 0 or Timer 1 in Mode 0 may be calculated as follows:

$$Timer\_Rate = f_{osc} / (N * (65536 - Timer\_Reload\_Value))$$

where N = the TCLK prescaler value: 4 (default), 16, or 64.

Mode 1

Mode 1 is the 16–bit non–auto reload mode.

Mode 2

Mode 2 configures the Timer register as an 8–bit Counter (TLn) with automatic reload. Overflow from TLn not only sets TFn, but also reloads TLn with the contents of RTLn, which is preset by software. The reload leaves THn unchanged.

Mode 2 operation is the same for Timer/Counter 0.

The overflow rate for Timer 0 or Timer 1 in Mode 2 may be calculated as follows:

$$Timer\_Rate = f_{osc} / (N * (256 - Timer\_Reload\_Value))$$

where N = the TCLK prescaler value: 4, 16, or 64.

Mode 3

Timer 1 in Mode 3 simply holds its count. The effect is the same as setting TR1 = 0.

Timer 0 in Mode 3 establishes TL0 and TH0 as two separate counters. TL0 uses the Timer 0 control bits: C0 ; T0/, GATE0, TR0, INT0/ and TF0. TH0 is locked into a timer function and takes over the use of TR1 and TF1 from Timer 1. Thus, TH0 now controls the “Timer 1” interrupt.

Mode 3 is provided for applications requiring an extra 8–bit timer. When Timer 0 is in Mode 3, Timer 1 can be turned on and off by switching it out of and into its own Mode 3, or can still be used by the serial port as a baud rate generator, or in fact, in any application not requiring an interrupt.

TCON Address:410
Bit Addressable
Reset Value: 00H

MSB							LSB
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

BIT	SYMBOL	FUNCTION
TCON.7	TF1	Timer 1 overflow flag. Set by hardware on Timer/Counter overflow. This flag will not be set if T1OE (TSTAT.2) is set. Cleared by hardware when processor vectors to interrupt routine, or by clearing the bit in software.
TCON.6	TR1	Timer 1 Run control bit. Set/cleared by software to turn Timer/Counter 1 on/off.
TCON.5	TF0	Timer 0 overflow flag. Set by hardware on Timer/Counter overflow. This flag will not be set if T0OE (TSTAT.0) is set. Cleared by hardware when processor vectors to interrupt routine, or by clearing the bit in software.
TCON.4	TR0	Timer 0 Run control bit. Set/cleared by software to turn Timer/Counter 0 on/off.
TCON.3	IE1	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
TCON.2	IT1	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.
TCON.1	IE0	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
TCON.0	IT0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

SU00604C

Figure 7. Timer/Counter Control (TCON) Register

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

timer register is loaded with FFFF hex. The underflow also sets the TF2 flag, which can generate an interrupt if enabled.

The External flag EXF2 bit toggles when Timer 2 underflows or overflows. This EXF2 bit can be used as a 17th bit of resolution, if needed. the EXF2 bit flag does not generate an interrupt in this mode. As the baud rate generator, timer T2 is incremented by TCLK.

Baud Rate Generator Mode
By setting the TCLK0 and/or RCLK0 in T2CON, Timer 2 can be chosen as the baud rate generator for the Transmitter and/or Receiver sides of UART–0.

Programmable Clock-Out

A 50% duty cycle clock can be programmed to come out on P1.6. This pin, besides being a regular I/O pin, has two alternate functions. Either it can be programmed to input the External clock for Timer/Counter 2 or to output a 50% duty cycle clock.

To configure the Timer/Counter 2 as a clock generator, bit {C2 or T2*I*} (T2CON[1]) must be cleared and bit T2OE (T2MOD[1]) must be set. Bit TR2 (T2CON[2]) also must be set to start the timer.

The Clock–Out frequency depends on the oscillator frequency and the reload value of Timer 2 capture registers (TCAP2H, TCAP2L) as shown in this equation:

$$\frac{\text{TCLK}}{2 \times (65536 - \text{TCAP2H, TCAP2L})}$$

In the Clock–Out mode Timer 2 roll–overs will not generate an interrupt. This is similar to when it is used as a baud–rate generator. It is possible to use Timer 2 as a baud–rate generator and a clock generator simultaneously. Note, however, that the baud–rate will be 1/8 of the Clock–Out frequency.

Table 7. Timer 2 Operating Modes

Bits of Special Function Registers				
TR2 T2CON[2]	**CP or RL2/ T2CON[0]**	**RCLK0 or TCLK0 T2CON[5] or T2CON[4]**	**DCEN T2MOD[0]**	**MODE**
0	X	X	X	Timer off (stopped)
1	0	0	0	16–bit auto–reload, counting up
1	0	0	1	16–bit auto–reload, counting up or down depending on T2EX pin
1	1	0	X	16–bit capture
1	X	1	X	Baud rate generator

TSTAT Address:411
Bit Addressable
Reset Value: 00H

MSB							LSB
—	—	—	—	—	T1OE	—	T0OE

BIT	SYMBOL	FUNCTION
TSTAT.2	T1OE	When 0, this bit allows the T1 pin to clock Timer 1 when in the counter mode. When 1, T1 acts as an output and toggles at every Timer 1 overflow.
TSTAT.0	T0OE	When 0, this bit allows the T0 pin to clock Timer 0 when in the counter mode. When 1, T0 acts as an output and toggles at every Timer 0 overflow.

SU00612B

Figure 9. Timer 0 and 1 Extended Status (TSTAT)

T2MOD Address:419
Bit Addressable
Reset Value: 00H

MSB							LSB
—	—	RCLK1	TCLK1	—	—	T2OE	DCEN

BIT	SYMBOL	FUNCTION
T2MOD.5	RCLK1	Receive Clock Flag.
T2MOD.4	TCLK1	Transmit Clock Flag. RCLK1 and TCLK1 are used to select Timer 2 overflow rate as a clock source for UART1 instead of Timer T1.
T2MOD.1	T2OE	When 0, this bit allows the T2 pin to clock Timer 2 when in the counter mode. When 1, T2 acts as an output and toggles at every Timer 2 overflow.
T2MOD.0	DCEN	Controls count direction for Timer 2 in autoreload mode. DCEN=0 counter set to count up only DCEN=1 counter set to count up or down, depending on T2EX (see text).

SU00610B

Figure 10. Timer 2 Mode Control (T2MOD)

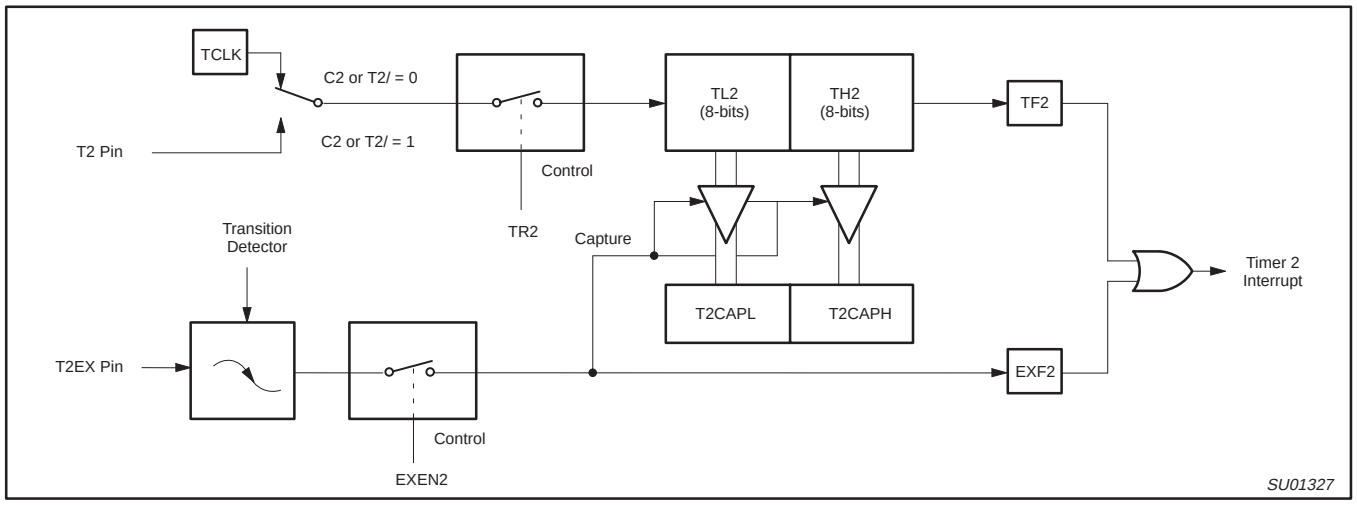


Figure 11. Timer 2 in Capture Mode

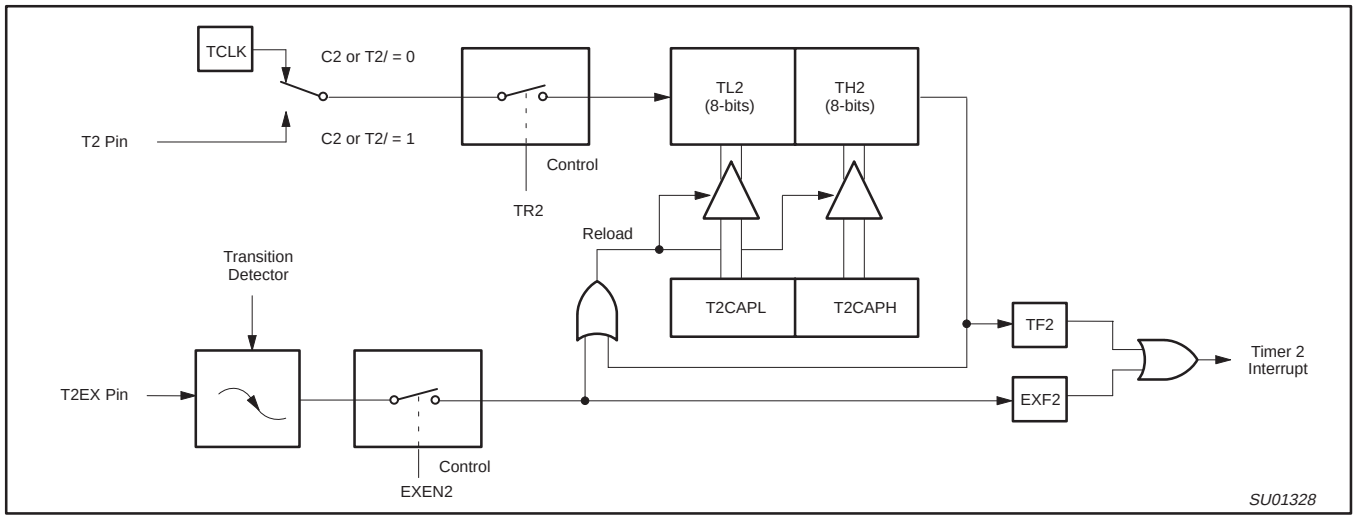


Figure 12. Timer 2 in Auto-Reload Mode (DCEN = 0)

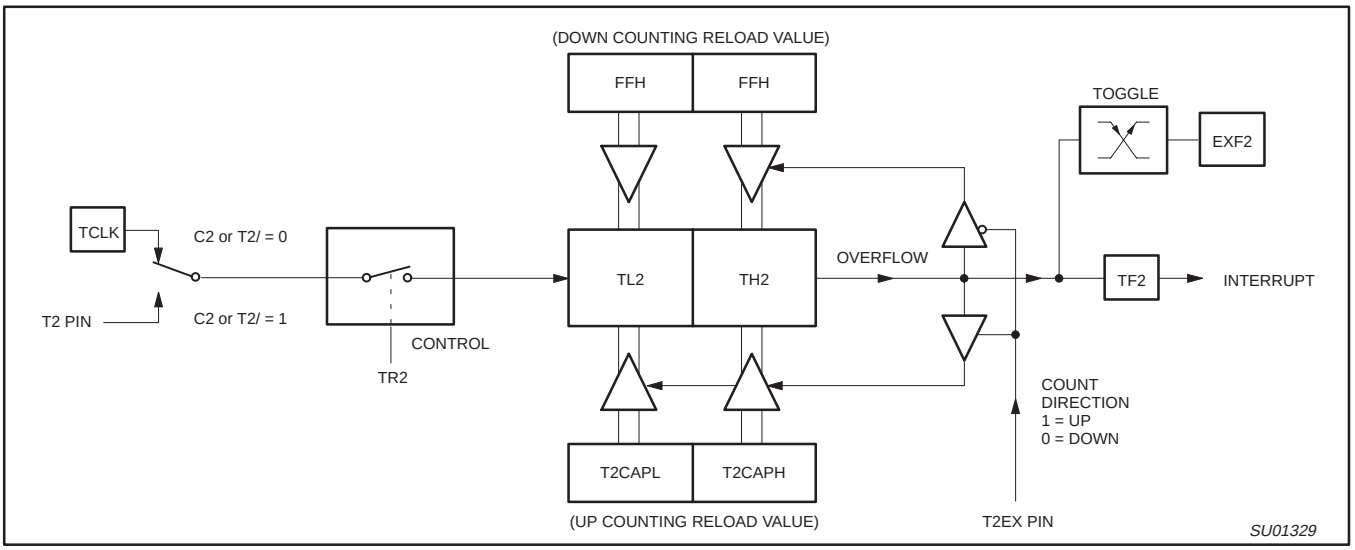


Figure 13. Timer 2 Auto Reload Mode (DCEN = 1)

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.

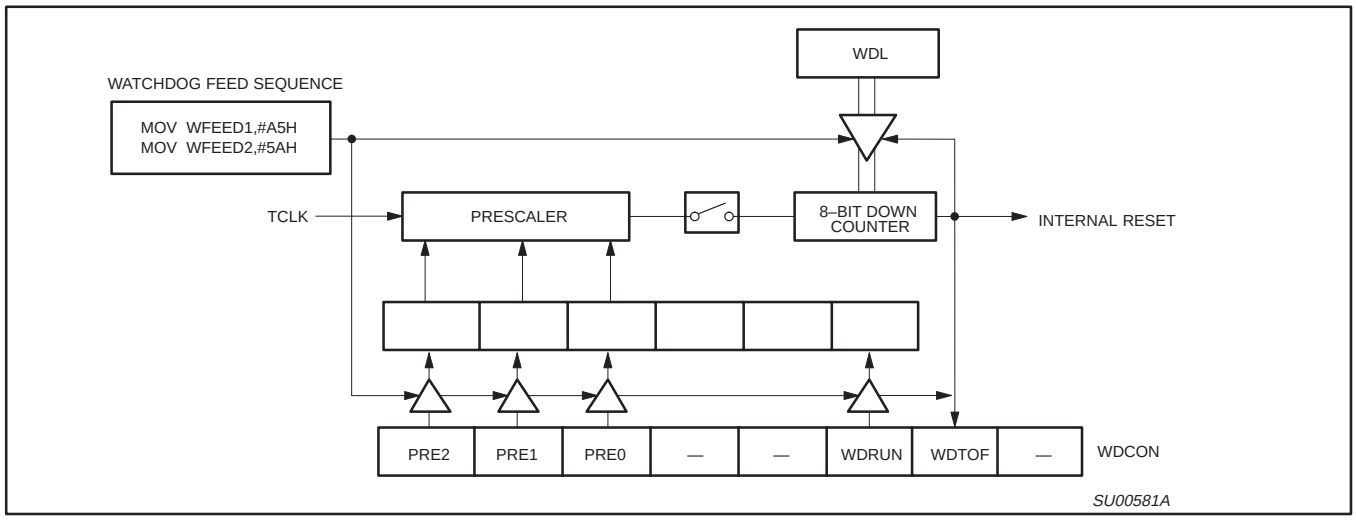


Figure 14. Watchdog Timer in XA-C3

When the watchdog underflows, the following action takes place (see Figure 14):

- Autoload takes place.
- Watchdog time–out flag is set
- Watchdog run bit unchanged.
- Autoload (WDL) register unchanged.
- Prescaler tap unchanged.
- All other device action same as External Reset.

Note that if the watchdog underflows, the Program counter will be loaded from the Reset vector as in the case of an internal Reset. The watchdog time–out flag can be examined to determine if the watchdog has caused the Reset condition. The watchdog time–out flag bit can be cleared by software.

WDCON Register Bit Definitions

WDCON[7]	PRE2	Prescaler Select 2, Reset to 1
WDCON[6]	PRE1	Prescaler Select 1, Reset to 1
WDCON[5]	PRE0	Prescaler Select 0, Reset to 1
WDCON[2]	WDRUN	Watchdog Run Control bit, Reset to 1
WDCON[1]	WDTOF	Timeout flag

UART

The XA–C3 includes 1 UART port (UART–0) that is compatible with the enhanced UART used on the 8xC51FB. Baud rate selection is somewhat different due to the clocking scheme used for the XA timers.

Four other enhancements have been made to UART operation: First, there are separate interrupt vectors for UART transmit and receive functions. Second, the UART–0 transmitter has been double–buffered, allowing packed transmission of data with no gaps between bytes and less critical interrupt service routine timing. Third, a break detect function has been added to UART–0. This operates independently of the UART and provides a start–of–break status bit that the User program may use to test BR0 (S0STAT[2]). Fourth, an Overrun Error flag has been added to detect missed characters in the received data stream.

The UART baud rate is determined by either a fixed division of the oscillator (in UART–0 Modes 0 and 2) or by the Timer 1 or Timer 2 overflow rate (in UART–0 Modes 1 and 3).

Timer 1 defaults to clock UART–0. Timer 2 can clock UART–0 through T2CON via bits RCLK0 (T2CON[5]) and/or TCLK0 (T2CON[4]).

The serial port receive and transmit registers are both accessed at Special Function Register S0BUF. Writing to S0BUF loads the transmit register, and reading S0BUF accesses the physically separate receive register.

The serial port can operate in 4 modes:

Mode 0: Serial I/O expansion mode. Serial data enters and exits through RxD. TxD outputs the shift clock. 8 bits are transmitted/received (LSB first). (The baud rate is fixed at 1/16 the oscillator frequency.)

Mode 1: Standard 8–bit UART mode. 10 bits are transmitted (through TxD) or received (through RxD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into bit RB8_0 (S0CON[2]). The baud rate is variable via Timer 1 or Timer 2 overflow rates.

Mode 2: Fixed rate 9–bit UART mode. 11 bits are transmitted (through TxD) or received (through RxD): start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On Transmit, the 9th data bit TB8_0 (S0CON[3]) can be assigned the value of 0 or 1. Or, for example, the parity bit (P, in the PSW) could be moved into TB8_0. On receive, the 9th data bit goes into bit RB8_0, while the stop bit is ignored. The baud rate is programmable to 1/32 of the oscillator frequency.

Mode 3: Standard 9–bit UART mode. 11 bits are transmitted (through TxD) or received (through RxD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). In fact, Mode 3 is the same as Mode 2 in all respects except baud rate. The baud rate in Mode 3 is variable via Timer 1 or Timer 2 overflow rates.

In all four modes, transmission is initiated by any instruction that uses S0BUF as a destination register. Reception is initiated in Mode 0 by the condition RI_0 (S0CON[0]) = 0 AND REN_0 (S0CON[4]) =

1. Reception is initiated in Mode 1, 2, or 3 by the incoming start bit if REN_0 = 1.

Serial Port Control Register

The serial port control and status register is the Special Function Register S0CON, shown in Figure 16. This register contains not only the mode selection bits, but also the 9th data bit for transmit and receive TB8_0 (S0CON[3]) and RB8_0 (S0CON[2]), and the serial port interrupt bits Transmit Interrupt flag TI_0 (S0CON[1]) and Receive Interrupt flag RI_0 (S0CON[0]) .

Transmit Interrupt Flag

In order to allow easy use of the double–buffered UART–0 transmitter feature, the TI_0 flag is set by the UART–0 hardware under two conditions. The first condition is the completion of any byte transmission. This occurs at the end of the stop bit in modes 1, 2, or 3, or at the end of the eighth data bit in mode 0. The second condition is when S0BUF is written while the UART–0 transmitter is idle.

Generally, UART transmitters generate one interrupt per byte transmitted. However, UART–0 generates one additional interrupt (as defined by the stated conditions for setting the TI_0 flag). This additional interrupt does not occur if double–buffering is bypassed as explained below. Note: If character–oriented transmission is used (not block–transmission of characters), there could be a second interrupt for each character transmitted, depending on the timing of the writes to S0BUF. For this reason, it is generally better to bypass double–buffering when UART–0 is used in character–oriented mode. This is also true if UART–0 is polled rather than interrupt–driven. The interrupt occurs at the end of the last byte transmitted when the UART becomes idle. Among other things, this allows a program to determine when a message has been transmitted completely. The interrupt service routine should handle this additional interrupt.

The recommended way to use transmit double–buffering in an application program is to have the UART interrupt service routine handle a single byte for each interrupt occurrence. Thus, the program will not require any special considerations for double–buffering. Transmitted bytes will then be tightly packed with no intervening gaps. Note: Be aware that higher priority interrupts may cause delays in servicing a transmitter interrupt, and this would defeat double–buffering.

9-Bit Mode

Because the ninth data bit TB8_0 (S0CON[3]) is not double–buffered, you must insure S0CON[3] contains the intended ninth data bit whenever it is transmitted. Alternatively, to synchronize the ninth data bit with the rest of the data stream, you could bypass double–buffering.

Bypassing Double-Buffering

The UART transmitter may be used as if it is single–buffered. The recommended UART transmitter interrupt service routine (ISR) technique to bypass double–buffering first clears the TI_0 flag (S0CON[1]) upon entry into the ISR, as in standard practice. This clears the interrupt that activated the ISR. Secondly, the TI_0 flag is cleared immediately following each write to S0BUF. This clears the interrupt flag that would otherwise direct the program to write to the second transmitter buffer. If there is any possibility that a higher priority interrupt might become active between the write to S0BUF and the clearing of the TI_0 flag, the interrupt system may have to be temporarily disabled during that sequence by clearing, then setting the EA bit (IEL[7]).

CLOCKING SCHEME AND BAUD RATE GENERATION

Clock Rates for all UART Modes

For UART Modes 0 and 2 the UART clock rate is determined by a fixed division of the oscillator clock. For Modes 1 and 3 the UART clock rate is determined by the overflow rates of either T1 or T2.

Baud Rates for UART Modes 0 and 2

In UART Mode 0, the baud rate is fixed at $f_{osc}/16$. In Mode 2, however, it is fixed rate at $f_{osc}/32$.

Baud Rate Calculations for UART Modes 0 and 2

Baud Rate for UART Mode 0:

Baud_Rate = $f_{osc}/16$

Baud Rate for UART Mode 2:

Baud_Rate = $f_{osc}/32$

Baud Rates for UART Modes 1 and 3

Table 9 shows the relationship of TCLK to pre–scalar settings for all Timers T0, T1, and T2.

Table 9. TCLK Frequencies

Pre–scalar Value	PT1 ; SCR[3]	PT0 ; SCR[2]	TCLK
4	0	0	$f_{osc}/4$
16	0	1	$f_{osc}/16$
64	1	0	$f_{osc}/64$
–	1	1	reserved

Thus, when Timers T0, T1, and T2 are used to establish the baud rate for Baud Clock, the maximum speed of timers/(Baud Clock) is $f_{osc}/4$ (since the minimum pre–scalar value N is equal to 4). Consequently, the maximum Baud_Rate equals Timer_Rate (timer overflow) divided by 16, i.e., $f_{osc}/64$.

Baud Rate Calculations for UART Modes 1 and 3

Baud Rate calculations for UART Mode 1 and 3:

Baud_Rate = Timer_Rate/16

Timer_Rate = $f_{osc}/(N \times (Timer\_Range - Timer\_Reload\_Value))$

where N = the TCLK prescaler value (4, 16, or 64).
and Timer_Range = 256 for Timer 1 in Mode 2.
and Timer_Range = 65536 for Timer 1 in Mode 0 and Timer 2 in count–up mode.

The timer reload value may be calculated as follows:

Timer_Reload_Value = Timer_Range – (f_{osc}/(Baud_Rate*N*16))

NOTES:

1. The maximum baud rate for UART–0 in Mode 1 or 3 is $f_{osc}/64$.
2. The lowest possible baud rate (for a given oscillator frequency and N value) may be found by using a timer reload value of 0.

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

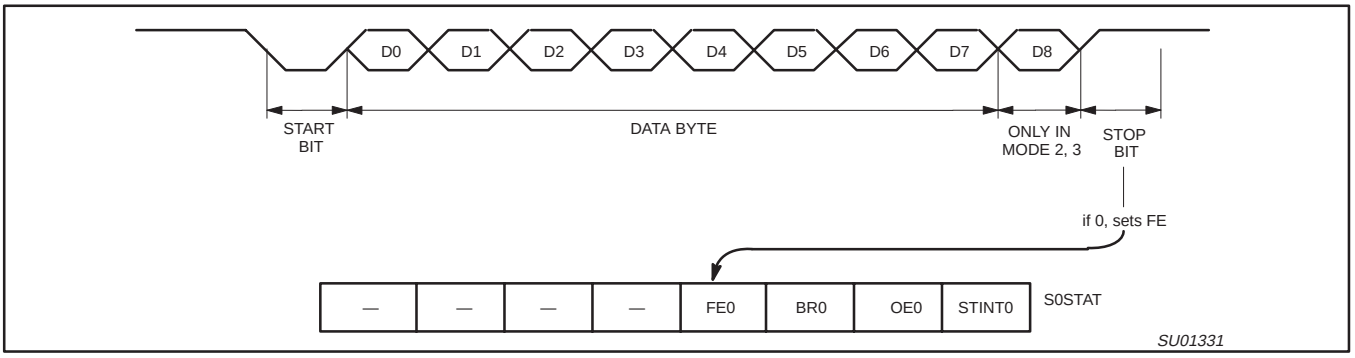


Figure 17. UART Framing Error Detection

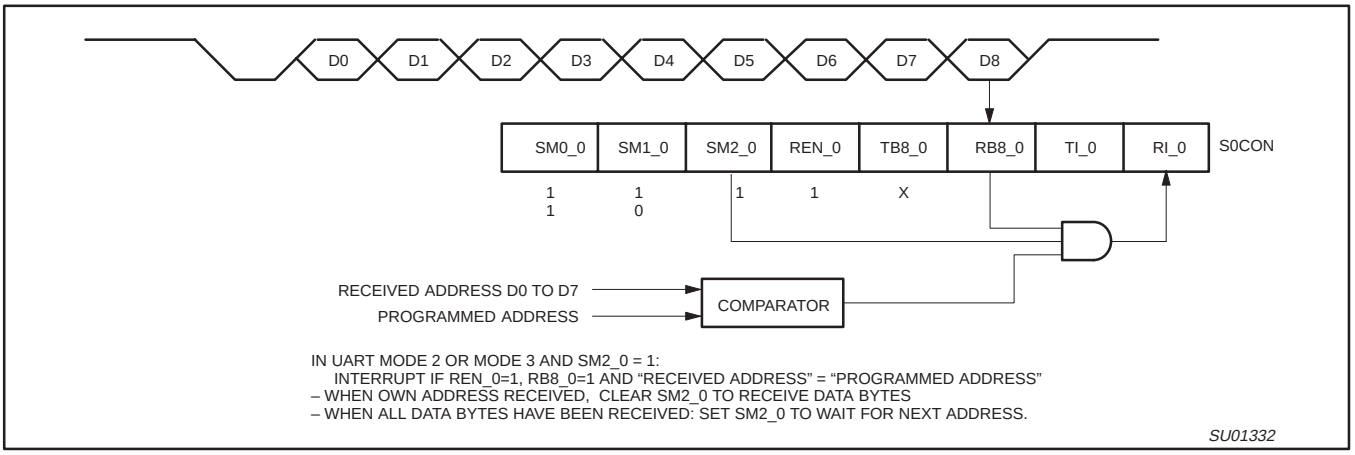


Figure 18. UART Multiprocessor Communication, Automatic Address Recognition

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.

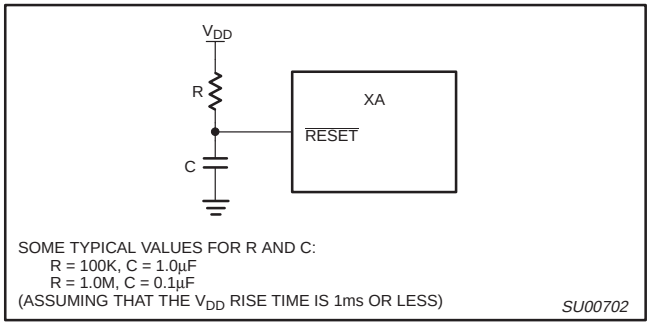


Figure 19. Recommended Reset Circuit

RST/Pin Properties and Requirements

- Active LOW for improved noise immunity
- Schmitt Trigger with Threshold = 0.7 Vdd
- RST/ must be low for the longer of 10 µs or 10 clocks
- If EA/ = 1, all Port pins are set to Quasi–Bidirectional mode
- If EA/ = 0, all External Bus pins are set to Push–Pull mode

Power-On Reset

- Must be > 10 msec to allow the on–chip oscillator to stabilize

Other Reset Effects

- Register File is zeroed except [R7] USP/SSP is set to 100h
- Internal DATA RAM is not affected
- All maskable interrupts are disabled
- DS, ES, CS, SSEL, PZ, CM, PT0 and PT1 are zeroed
- The Watchdog Timer is turned ON

Reset Timing

The EA/ pin is sampled on the rising edge of the Reset (RST/) pulse. The result of this sampling determines whether the device is to begin execution from internal or External PROGRAM memory. Specifically, if EA/ is pulled high, the XA starts in Single–Chip mode.

Lastly, after RST/ is released, the {WAIT ; V_{pp} ; EA/} pin becomes a bus WAIT signal for External bus transactions.

P3.5 is weakly pulled high whenever RST/ is asserted. Given EA/ is used at RESET to request code starts from External memory, this weak pull up assures the PXAC3 will set–up a 16 bit External bus. Thus, if External code operation is desired, the User must NEVER put a LOW on P3.5 during RESET.

Note: EA/ must be held for eight equivalent oscillator clock periods after RST/ is deasserted (i.e., after RST/ returns to ONE) to guarantee that the desired EA/ value is latched correctly.

The relationship of EA/ timing with respect to both RST/ and ALE signals is shown in Figure 20.

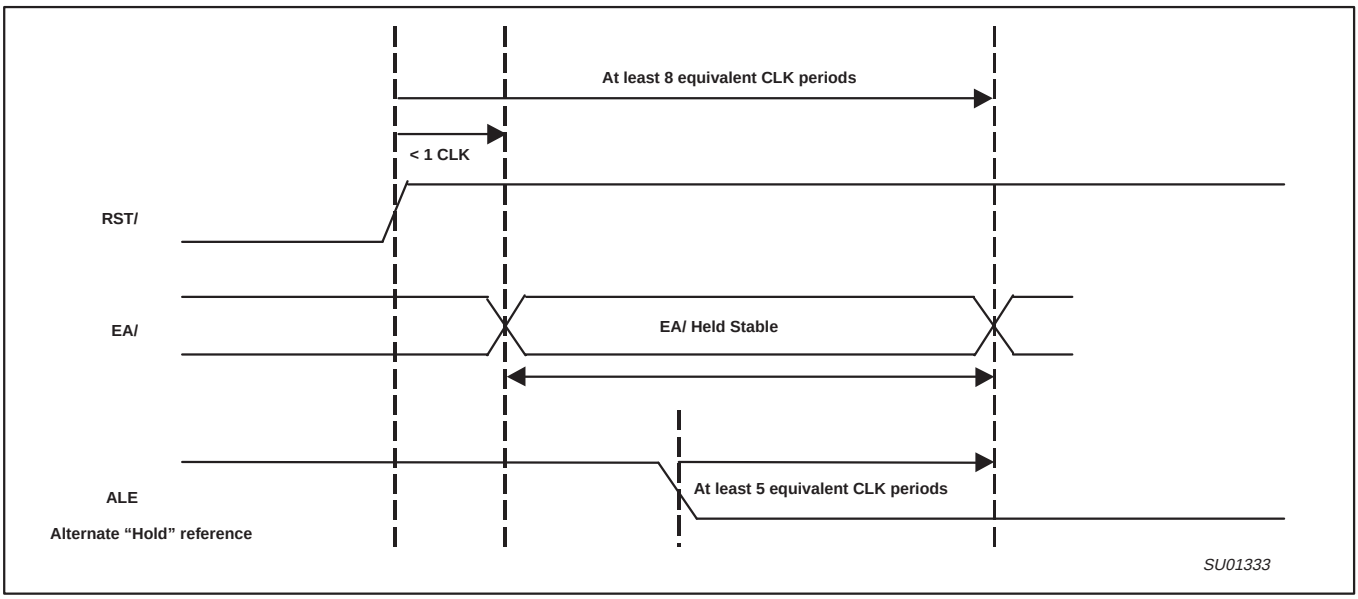


Figure 20. EA/ Timing Diagram

Power Reduction Modes

The XA–C3 supports Idle and Power–Down modes of power reduction. The Idle mode leaves some peripherals running to allow them to wake up the processor when an interrupt is generated. The Power–Down mode stops the oscillator in order to minimize power. The processor can be made to exit Power–Down mode via Reset or one of the External interrupt inputs. In order to use an External interrupt to re–activate the XA while in Power–Down mode, the External interrupt must be enabled and be configured to level–sensitive mode. In Power–Down mode, the power supply voltage may be reduced to the RAM keep–alive voltage (2V), retaining the RAM, register, and SFR values at the point where the Power–Down mode was entered.

Interrupts

Interrupt Types

There are four types of interrupts:

- **Event Interrupts** – service peripherals such as UARTs and timers.
- **Software Interrupts** – demote the priority level of a running Event Interrupt below the lowest Event priority level (i.e., 9), thereby permitting lower priority Event Interrupts to run.
- **Trap Interrupts** –accomplish multi–tasking services, such as RTOS, via non–maskable interrupts.

- **Exception Interrupts** – process non–maskable events, such as Reset, Stack Overflow, and Divide–by–zero.

The XA–C3 supports 42 vectored interrupts. These include 13 maskable Event Interrupts, 7 Software Interrupts, 16 Trap interrupts, and 6 Exception Interrupts. The number of Event Interrupts is related to the number of on–chip peripherals. The XA–C3 supports 13 maskable Event Interrupts. However, Software, Trap, and Exception Interrupts are standardized within the XA core. For core details refer to the *XA User Guide.*

Interrupt Structures

Four tables provide details of the XA-C3 Interrupt structure.

- Table 14 defines the sixteen interrupt priority levels
- Table 15 describes the Exception and Trap Interrupts
- Table 16 explains the Event Interrupts
- Table 17 lists the Software Interrupts

Event Interrupt Handling

If a higher priority Event occurs while a lower priority Event is being serviced, the higher priority Event takes over.

When Events of different priorities occur simultaneously, the highest priority Event is serviced first.

When Events of equal priority occur simultaneously, Arbitration Ranking determines which Event is serviced first. See Table 15 and Table 16.

Interrupt Priority Details

Each Event interrupt has 8 priority levels. Event interrupts may be individually masked by bits in SFR Registers IEL and IEH (see Table 5). Event interrupts can also be globally disabled via the EA bit (IEL[7]).

Using 3–bit sub–groups, Interrupt Priority Assignment (IPA) registers (IPA0, IPA1, IPA2, IPA4, IPA5, IPA6, and IPA7) assign 1 of 8 priority levels per Event Interrupt. A zero value assigns interrupt priority 0, in effect disabling an interrupt. The remaining seven priority levels are defined in Table 14.

Table 14. Interrupt Priority Levels

Priority Level	Type of Interrupt
15	Event Interrupt
14	Event Interrupt
13	Event Interrupt
12	Event Interrupt
11	Event Interrupt
10	Event Interrupt
9	Event Interrupt
8	
7	Software Interrupt
6	Software Interrupt
5	Software Interrupt
4	Software Interrupt
3	Software Interrupt
2	Software Interrupt
1	Software Interrupt
0	Interrupt Disable

NOTE:
1. Details of the priority scheme may be found in the XA User Guide.

Table 15. Exception and Trap Interrupt Vectors

DESCRIPTION	VECTOR ADDRESS	ARBITRATION RANKING
Reset (h/w, watchdog, s/w)	0000 – 0003	0 (High)
Breakpoint (h/w trap 1)	0004 – 0007	1
Trace (h/w trap 2)	0008 – 000B	1
Stack Overflow (h/w trap 3)	000C – 000F	1
Divide by 0 (h/w trap 4)	0010 – 0013	1
User RETI (h/w trap 5)	0014 – 0017	1
TRAP 0– 15 (software)	0040 – 007F	1

Table 16. Event Interrupt Vectors

DESCRIPTION	FLAG BIT	VECTOR ADDRESS	ENABLE BIT	INTERRUPT PRIORITY	ARBITRATION RANKING
External interrupt 0	IE0 ; TCON[1]	0080–0083	EX0 ; IEL[0]	PX0 ; IPA0[2:0]	2
Timer 0 interrupt	TF0 ; TCON[5]	0084–0087	ET0 ; IEL[1]	PT0 ; IPA0[6:4]	3
External interrupt 1	IE1 ; TCON[3]	0088–008B	EX1 ; IEL[2]	PX1 ; IPA1[2:0]	4
Timer 1 interrupt	TF1 ; TCON[7]	008C–008F	ET1 ; IEL[3]	PT1 ; IPA1[6:4]	5
Timer 2 interrupt	TF2 ; T2CON[7] or T2EX [P1.7][1] or EXF2 ; T2CON[6]	0090–0093	ET2 ; IEL[4]	PT2 ; IPA2[2:0]	6
(CAN) Rx buffer full	CANINTFLG[2]	0094–0097	EBUFF ; IEL[5]	PBUFF ; IPA2[6:4]	7
Serial port 0 Rx	RI_0 ; S0CON[0]	00A0–00A3	ERI0 ; IEH[0]	PRI0 ; IPA4[2:0]	10
Serial port 0 Tx	TI_0 ; S0CON[1]	00A4–00A7	ETI0 ; IEH[1]	PTI0 ; IPA4[6:4]	11
SPI Interrupt	SPFG ; SPICS[3]	00AC–00AF	ESPI ; IEH[3]	PSPI ; IPA5[6:4]	13
(CAN) Frame Error	CANINTFLG[4]	00B0–00B3	ECER ; IEH[4]	PCER ; IPA6[2:0]	14
(CAN) Message Error	CANINTFLG[3]	00B4–00B7	EMER ; IEH[5]	PMER ; IPA6[6:4]	15
(CAN) Tx message complete	CANINTFLG[1]	00B8–00BB	EMTI ; IEH[6]	PMTI ; IPA7[2:0]	16
(CAN) Rx message complete	CANINTFLG[0]	00BC–00BF	EMRI ; IEH[7]	PMRI ; IPA7[6:4]	17

NOTE:
1. When Timer 2 is used as a baud rate generator, pin T2EX [P1.7] acts as an additional External interrupt.

Table 17. Software Interrupt Vectors

DESCRIPTION	REQUEST BIT	VECTOR ADDRESS	ENABLE BIT	INTERRUPT PRIORITY
Software interrupt 7	SWR7 ; SWR[6]	0118–011B	SWE7 ; SWE[6]	fixed at 7 (highest priority)
Software interrupt 6	SWR6 ; SWR[5]	0114–0117	SWE6 ; SWE[5]	fixed at 6
Software interrupt 5	SWR5 ; SWR[4]	0110–0113	SWE5 ; SWE[4]	fixed at 5
Software interrupt 4	SWR4 ; SWR[3]	010C–010F	SWE4 ; SWE[3]	fixed at 4
Software interrupt 3	SWR3 ; SWR[2]	0108–010B	SWE3 ; SWE[2]	fixed at 3
Software interrupt 2	SWR2 ; SWR[1]	0104–0107	SWE2 ; SWE[1]	fixed at 2
Software interrupt 1	SWR1 ; SWR[0]	0100–0103	SWE1 ; SWE[0]	fixed at 1 (lowest priority)

ABSOLUTE MAXIMUM RATINGS

Table 18. Absolute Maximum Ratings

PARAMETER	RATING	UNIT
Operating temperature under bias	–55 to +125	°C
Storage temperature range	–65 to +150	°C
Voltage on EA*I* ; V_{PP} pin to V_{SS}	0 to +13.0	V
Voltage on any other pin to V_{SS}	–0.5 to V_{DD}+0.5V	V
Maximum I_{OL} per I/O pin	15	mA
Power dissipation (based on package heat transfer limitations, not device power consumption)	1.5	W

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.

AC ELECTRICAL CHARACTERISTICS

Table 20. AC Electrical Characteristics

V_{DD} = 4.5V to 5.5V; T_{amb} = 0 to +70°C for commercial, –40°C to +85°C for industrial.

SYMBOL	Figure	PARAMETER	VARIABLE CLOCK		UNIT
			MIN	MAX	
External Clock					
f_C		Oscillator frequency	0	32	MHz
t_C	22	Clock period and CPU timing cycle	$1/f_C$		ns
t_{CHCX}	22	Clock high time	$t_C * 0.5$		ns
t_{CLCX}	22	Clock low time	$t_C * 0.4$		ns
t_{CLCH}	22	Clock rise time		5	ns
t_{CHCL}	22	Clock fall time		5	ns
Address Cycle					
t_{CRAR}	21	Delay from clock rising edge to ALE rising edge	10	46	ns
t_{LHLL}	16	ALE pulse width (programmable)	$(V1 * t_C) – 6$		ns
t_{AVLL}	16	Address valid to ALE de–asserted (set–up)	$(V1 * t_C) – 12$		ns
t_{LLAX}	16	Address hold after ALE de–asserted	$(t_C/2) – 10$		ns
Code Read Cycle					
t_{PLPH}	16	PSEN/ pulse width	$(V2 * t_C) – 10$		ns
t_{LLPL}	16	ALE de–asserted to PSEN/ asserted	$(t_C/2) – 7$		ns
t_{AVIVA}	16	Address valid to instruction valid, ALE cycle (access time)		$(V3 * t_C) – 36$	ns
t_{AVIVB}	17	Address valid to instruction valid, non–ALE cycle (access time)		$(V4 * t_C) – 29$	ns
t_{PLIV}	16	PSEN/ asserted to instruction valid (enable time)		$(V2 * t_C) – 29$	ns
t_{PXIX}	16	Instruction hold after PSEN/ de–asserted	0		ns
t_{PXIZ}	16	Bus 3–State after PSEN/ de–asserted (disable time)		$t_C – 8$	ns
t_{IXUA}	16	Hold time of unlatched part of address after instruction latched	0		ns
Data Read Cycle					
t_{RLRH}	18	RD/ pulse width	$(V7 * t_C) – 10$		ns
t_{LLRL}	18	ALE de–asserted to RD/ asserted	$(t_C/2) – 7$		ns
t_{AVDVA}	18	Address valid to data input valid, ALE cycle (access time)		$(V6 * t_C) – 36$	ns
t_{AVDVB}	19	Address valid to data input valid, non–ALE cycle (access time)		$(V5 * t_C) – 29$	ns
t_{RLDV}	18	RD/ low to valid data in, enable time		$(V7 * t_C) – 29$	ns
t_{RHDX}	18	Data hold time after RD/ de–asserted	0		ns
t_{RHDZ}	18	Bus 3–State after RD/ de–asserted (disable time)		$t_C – 8$	ns
t_{DXUA}	18	Hold time of unlatched part of address after data latched	0		ns
Data Write Cycle					
t_{WLWH}	20	WR/ pulse width	$(V8 * t_C) – 10$		ns
t_{LLWL}	20	ALE falling edge to WR/ asserted	$(V12 * t_C) – 10$		ns
t_{QVWX}	20	Data valid before WR/ asserted (data setup time)	$(V13 * t_C) – 22$		ns
t_{WHQX}	20	Data hold time after WR/ de–asserted (Note 6)	$(V11 * t_C) – 5$		ns
t_{AVWL}	20	Address valid to WR/ asserted (address setup time) (Note 5)	$(V9 * t_C) – 22$		ns
t_{UAWH}	20	Hold time of unlatched part of address after WR/ is de–asserted	$(V11 * t_C) – 7$		ns
WAIT Input					
t_{WTH}	21	WAIT stable after bus strobe (RD/, WR/, or PSEN/) asserted		$(V10 * t_C) – 30$	ns
t_{WTL}	21	WAIT hold after bus strobe (RD/, WR/, or PSEN/) assertion	$(V10 * t_C) – 5$		ns

NOTES:

1. Load capacitance for all outputs = 80pF.
2. Variables V1 through V13 reflect programmable bus timing, which is programmed via the Bus Timing registers (BTRH and BTRL).
   Refer to the XA User Guide for details of the bus timing settings.
   V1) This variable represents the programmed width of the ALE pulse as determined by the ALEW bit in the BTRL register. V1 = 0.5 if the ALEW bit = 0, and 1.5 if the ALEW bit = 1.
   V2) This variable represents the programmed width of the PSEN/ pulse as determined by the CR1 and CR0 bits or the CRA1, CRA0, and ALEW bits in the BTRL register.
   – For a bus cycle with **no** ALE, V2 = 1 if CR1/0 = 00, 2 if CR1/0 = 01, 3 if CR1/0 = 10, and 4 if CR1/0 = 11. Note that during burst mode code fetches, PSEN/ does not exhibit transitions at the boundaries of bus cycles. V2 still applies for the purpose of determining peripheral timing requirements.
   – For a bus cycle **with** an ALE, V2 = the total bus cycle duration (2 if CRA1/0 = 00, 3 if CRA1/0 = 01, 4 if CRA1/0 = 10, and 5 if CRA1/0 = 11) minus the number of clocks used by ALE (V1 + 0.5).
   Example: If CRA1/0 = 10 and ALEW = 1, the V2 = 4 – (1.5 + 0.5) = 2.
   V3) This variable represents the programmed length of an entire code read cycle **with** ALE. This time is determined by the CRA1 and CRA0 bits in the BTRL register. V3 = the total bus cycle duration (2 if CRA1/0 = 00, 3 if CRA1/0 = 01, 4 if CRA1/0 = 10, and 5 if CRA1/0 = 11).

V4) This variable represents the programmed length of an entire code read cycle with **no** ALE. This time is determined by the CR1 and CR0 bits in the BTRL register. V4 = 1 if CR1/0 = 00, 2 if CR1/0 = 01, 3 if CR1/0 = 10, and 4 if CR1/0 = 11.

V5) This variable represents the programmed length of an entire data read cycle with **no** ALE. this time is determined by the DR1 and DR0 bits in the BTRH register. V5 = 1 if DR1/0 = 00, 2 if DR1/0 = 01, 3 if DR1/0 = 10, and 4 if DR1/0 = 11.

V6) This variable represents the programmed length of an entire data read cycle **with** ALE. The time is determined by the DRA1 and DRA0 bits in the BTRH register. V6 = the total bus cycle duration (2 if DRA1/0 = 00, 3 if DRA1/0 = 01, 4 if DRA1/0 = 10, and 5 if DRA1/0 = 11).

V7) This variable represents the programmed width of the RD*/* pulse as determined by the DR1 and DR0 bits or the DRA1, DRA0 in the BTRH register, and the ALEW bit in the BTRL register.

– For a bus cycle with **no** ALE, V7 = 1 if DR1/0 = 00, 2 if DR1/0 = 01, 3 if DR1/0 = 10, and 4 if DR1/0 = 11.

– For a bus cycle **with** an ALE, V7 = the total bus cycle duration (2 if DRA1/0 = 00, 3 if DRA1/0 = 01, 4 if DRA1/0 = 10, and 5 if DRA1/0 = 11) minus the number of clocks used by ALE (V1 + 0.5).
Example: If DRA1/0 = 00 and ALEW = 0, then V7 = 2 – (0.5 + 0.5) = 1.

V8) This variable represents the programmed width of the WRL*/* and/or WRH*/* pulse as determined by the WM1 bit in the BTRL register. V8 1 if WM1 = 0, and 2 if WM1 = 1.

V9) This variable represents the programmed address setup time for a write as determined by the data write cycle duration (defined by DW1 and DW0 or the DWA1 and DWA0 bits in the BTRH register), the WM0 bit in the BTRL register, and the value of V8.

– For a bus cycle **with** an ALE, V9 = the total bus write cycle duration (2 if DWA1/0 = 00, 3 if DWA1/0 = 01, 4 if DWA1/0 = 10, and
5 if DWA1/0 = 11) minus the number of clocks used by the WRL*/* and/or WRH*/* pulse (V8), minus the number of clocks used by data hold time (0 if WM0 = 0 and 1 if WM0 = 1).
Example: If DWA1/0 = 10, WM0 = 1, and WM1 = 1, then V9 = 4 – 1 – 2 = 1.

– For a bus cycle with **no** ALE, V9 = the total bus cycle duration (2 if DW1/0 = 00, 3 if DW1/0 = 01, 4 if DW1/0 = 10, and 5 if DW1/0 = 11) minus the number of clocks used by the WRL*/* and/or WRH*/* pulse (V8), minus the number of clocks used by data hold time (0 if WM0 = 0 and 1 if WM0 = 1).
Example: If DW1/0 = 11, WM0 = 1, and WM1 = 0, then V9 = 5 – 1 – 1 = 3.

V10) This variable represents the length of a bus strobe for calculation of WAIT setup and hold times. The strobe may be RD*/* (for data read cycles), WRL*/* and/or WRH*/* (for data write cycles), or PSEN*/* (for code read cycles), depending on the type of bus cycle being widened by WAIT. V10 = V2 for WAIT associated with a code read cycle using PSEN*/*. V10 = V8 for a data write cycle using WRL*/* and/or WRH*/*. V10 = V7–1 for a data read cycle using RD*/*. This means that a single clock data read cycle cannot be stretched using WAIT.
If WAIT is used to vary the duration of data read cycles, the RD*/* strobe width must be set to be at least two clocks in duration. Also see Note 4.

V11) This variable represents the programmed write hold time as determined by the WM0 bit in the BTRL register.
V11 = 0 if the WM0 bit = 0, and 1 if the WM0 bit = 1.

V12) This variable represents the programmed period between the end of the ALE pulse and the beginning of the WRL*/* and/or WRH*/* pulse as determined by the data write cycle duration (defined by the DWA1 and DWA0 bits in the BTRH register), the WM0 bit in the BTRL register, and the values of V1 and V8. V12 = the total bus cycle duration (2 if DWA1/0 = 00, 3 if DWA1/0 = 01, 4 if DWA1/0 = 10, and 5 if DWA1/0 = 11) minus the number of clocks used by the WRL*/* and/or WRH*/* pulse (V8), minus the number of clocks used by data hold time (0 if WM0 = 0 and 1 if WM0 = 1), minus the width of the ALE pulse (V1).
Example: If DWA1/0 = 11, WM0 = 1, WM1 = 0, and ALEW = 1, then V12 = 5 – 1 – 1 – 1.5 = 1.5.

V13) This variable represents the programmed data setup time for a write as determined by the data write cycle duration (defined by DW1 and DW0 or the DWA1 and DWA0 bits in the BTRH register), the WM0 bit in the BTRL register, and the values of V1 and V8.

– For a bus cycle **with** an ALE, V13 = the total bus cycle duration (2 if DWA1/0 = 00, 3 if DWA1/0 = 01, 4 if DWA1/0 = 10, and 5 if DWA1/0 = 11) minus the number of clocks used by the WRL*/* and/or WRH*/* pulse (V8), minus the number of clocks used by data hold time (0 if WM0 = 0 and 1 if WM0 = 1), minus the number of clocks used by ALE (V1 + 0.5).
Example: If DWA1/0 = 11, WM0 = 1, WM1 = 1, and ALEW = 0, then V13 = 5 – 1 – 2 – 1 = 1.

– For a bus cycle with **no** ALE, V13 = the total bus cycle duration (2 if DW1/0 = 00, 3 if DW1/0 = 01, 4 if DW1/0 = 10, and 5 if DW1/0 = 11) minus the number of clocks used by the WRL*/* and/or WRH*/* pulse (V8), minus the number of clocks used by data hold time (0 if WM0 = 0 and 1 if WM0 = 1).
Example: If DW1/0 = 01, WM0 = 1, and WM1 = 0, then V13 = 3 – 1 – 1 = 1.

3. Not all combinations of bus timing configuration values result in valid bus cycles. *Refer to the XA User Guide section on the External Bus for details.*
4. When code is being fetched for execution on the External bus, a burst–mode fetch is used that does not have PSEN*/* edges in every fetch cycle. Thus, if WAIT is used to delay code fetch cycles, a change in the low–order address lines must be detected to locate the beginning of a cycle. This would be A3 A1 while using an External 16 bit bus.
5. This parameter is provided for peripherals that have the data clocked in on the falling edge of the WR*/* strobe. This is not usually the case, and in most applications this parameter is not used.
6. Please note that the XA–C3 requires that extended data bus hold time (WM0 = 1) to be used with External bus write cycles.

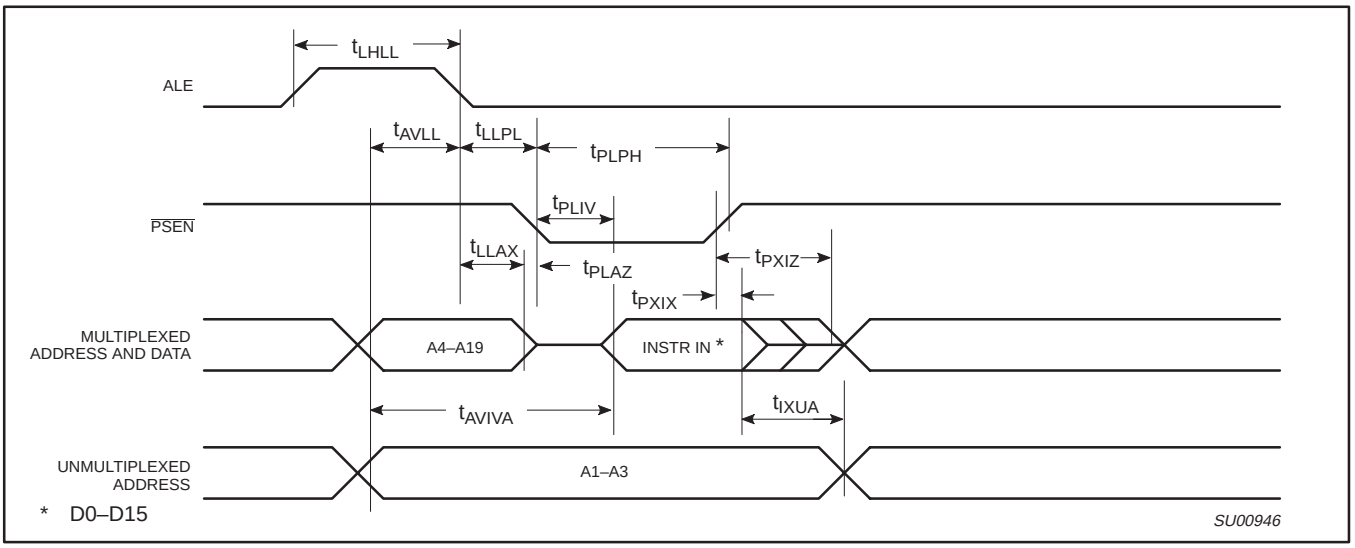


Figure 21. External PROGRAM Memory Read Cycle (ALE Cycle)

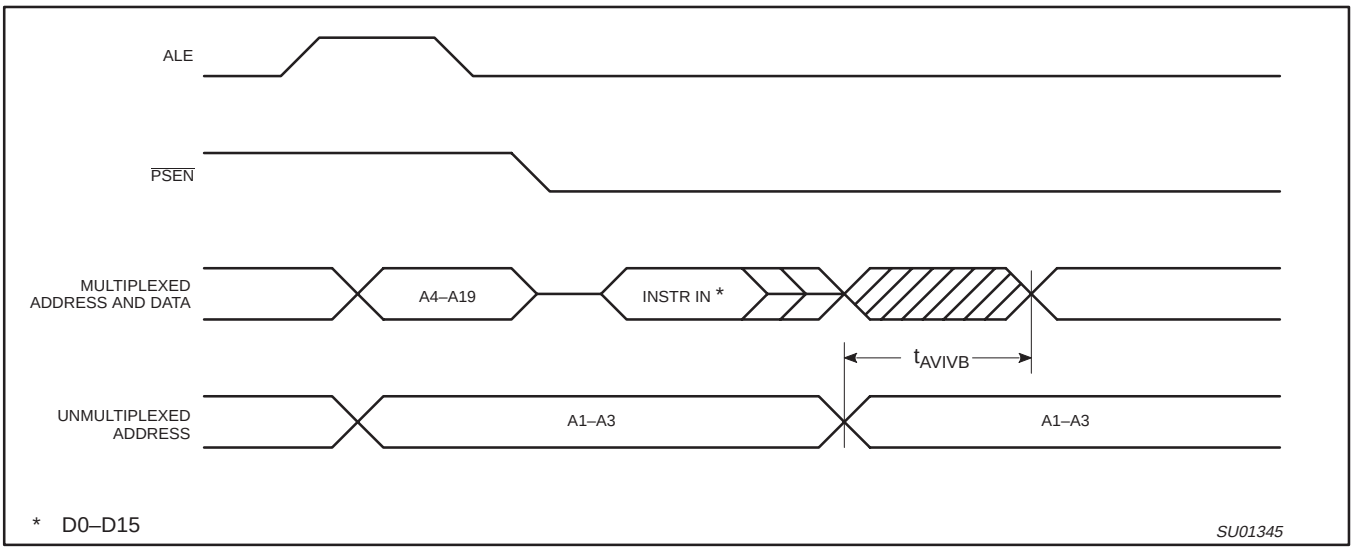


Figure 22. External PROGRAM Memory Read Cycle (Non-ALE Cycle)

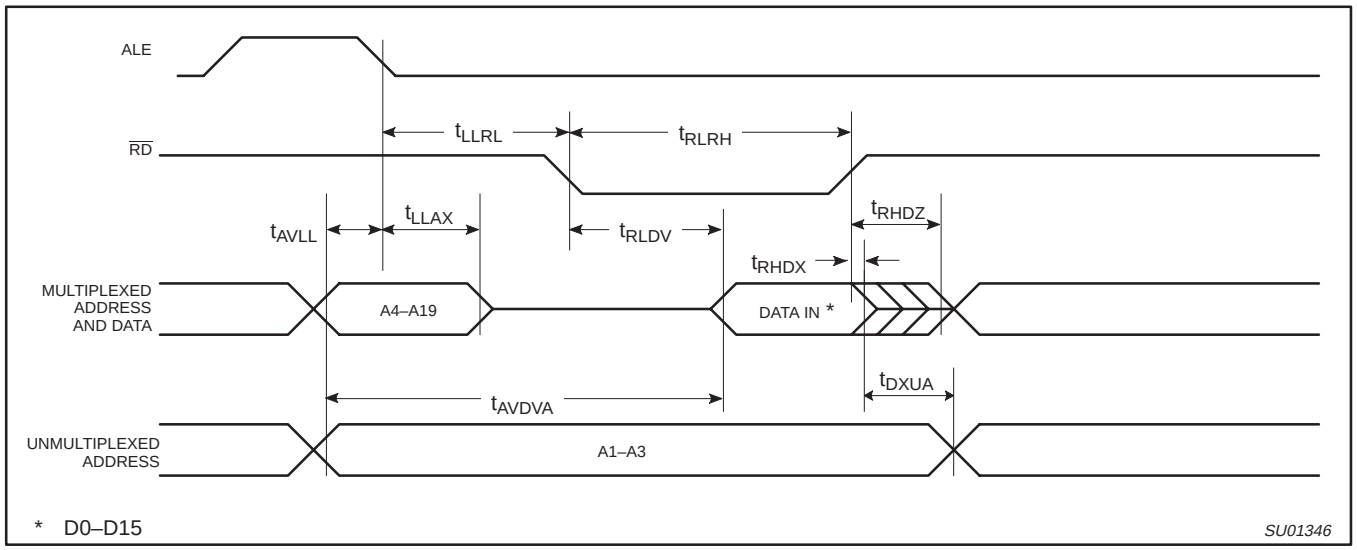


Figure 23. External DATA Memory Read Cycle (ALE Cycle)

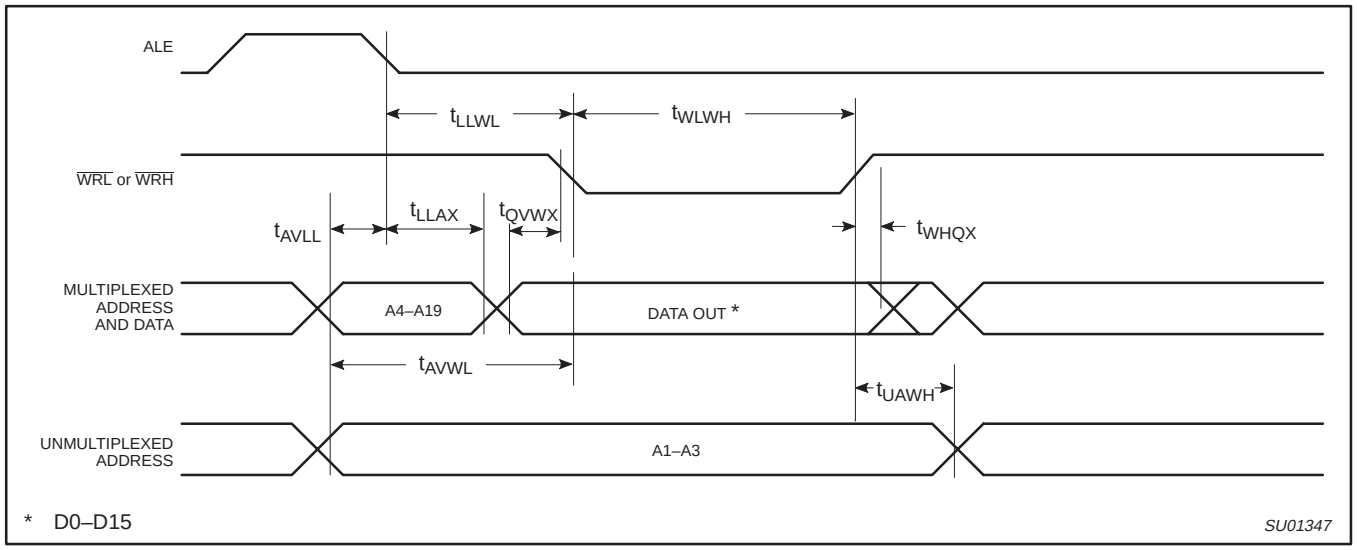


Figure 24. External DATA Memory Write Cycle

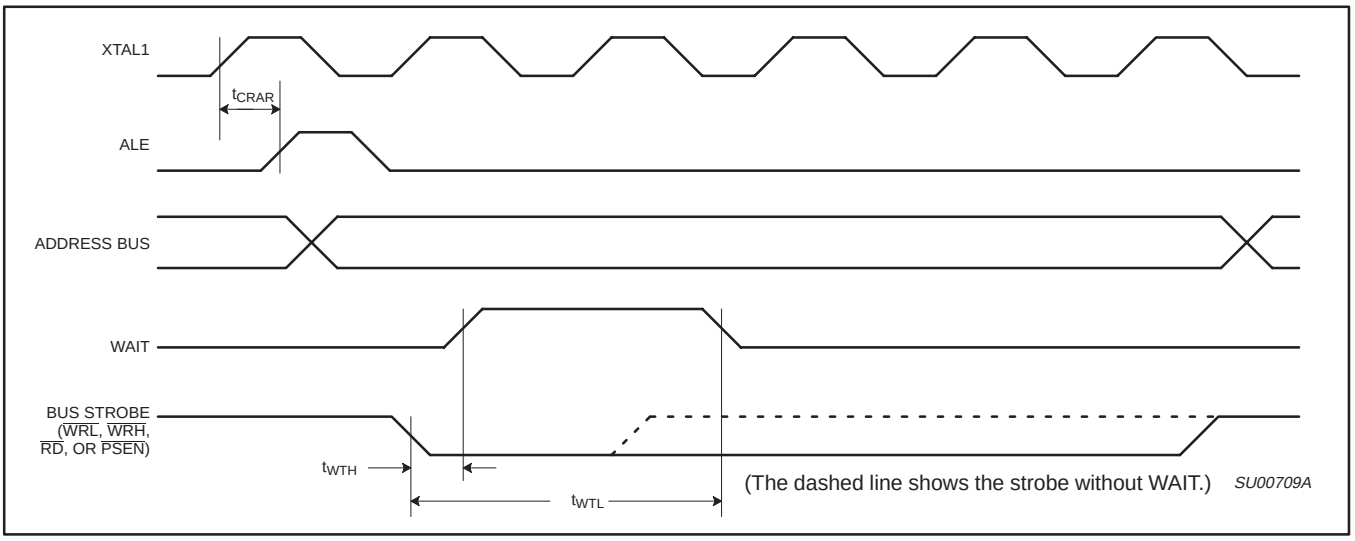


Figure 25. WAIT Signal Timing

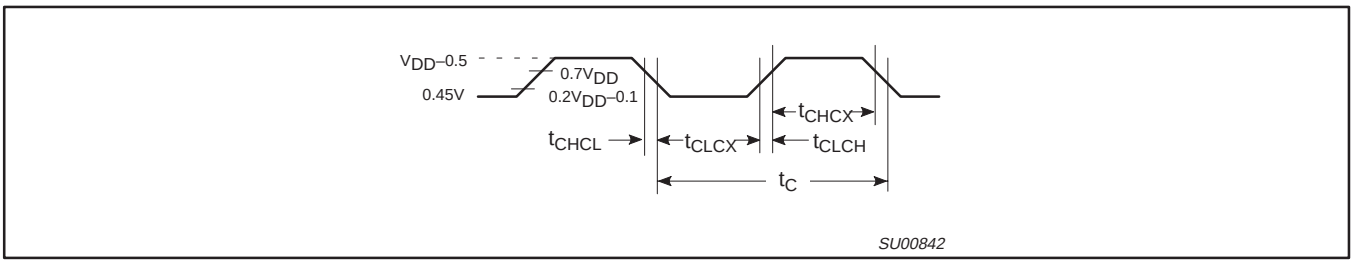


Figure 26. External Clock Drive

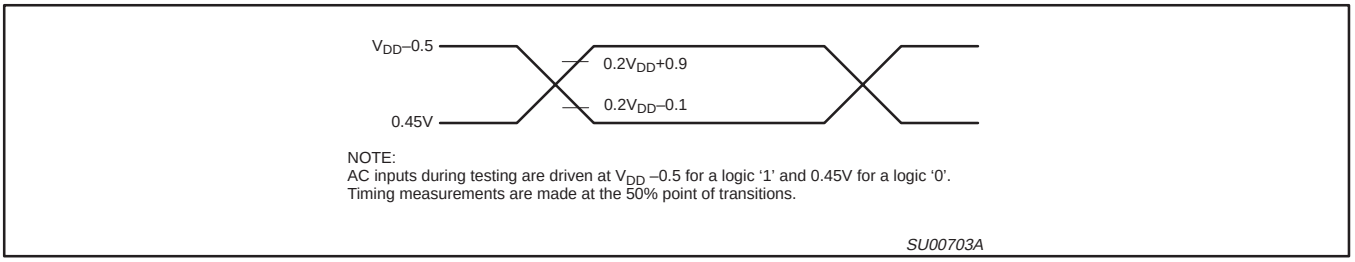


Figure 27. AC Testing Input/Output

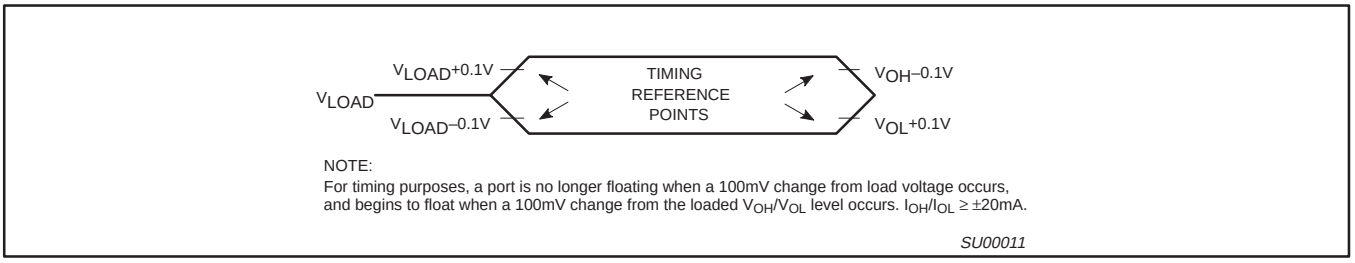


Figure 28. Float Waveform

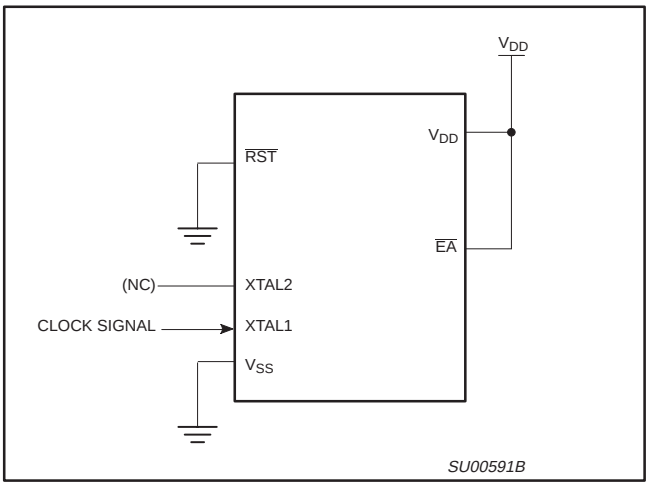


Figure 29. I_{DD} Test Condition, Active Mode

Note: All other pins are disconnected

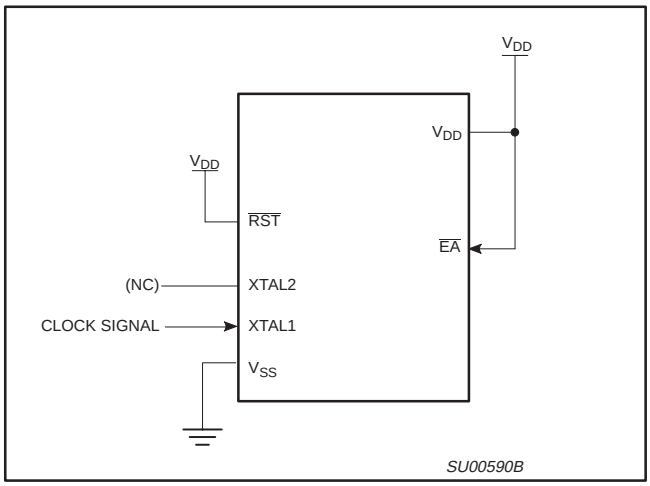


Figure 30. I_{DD} Test Condition, Idle Mode

Note: All other pins are disconnected

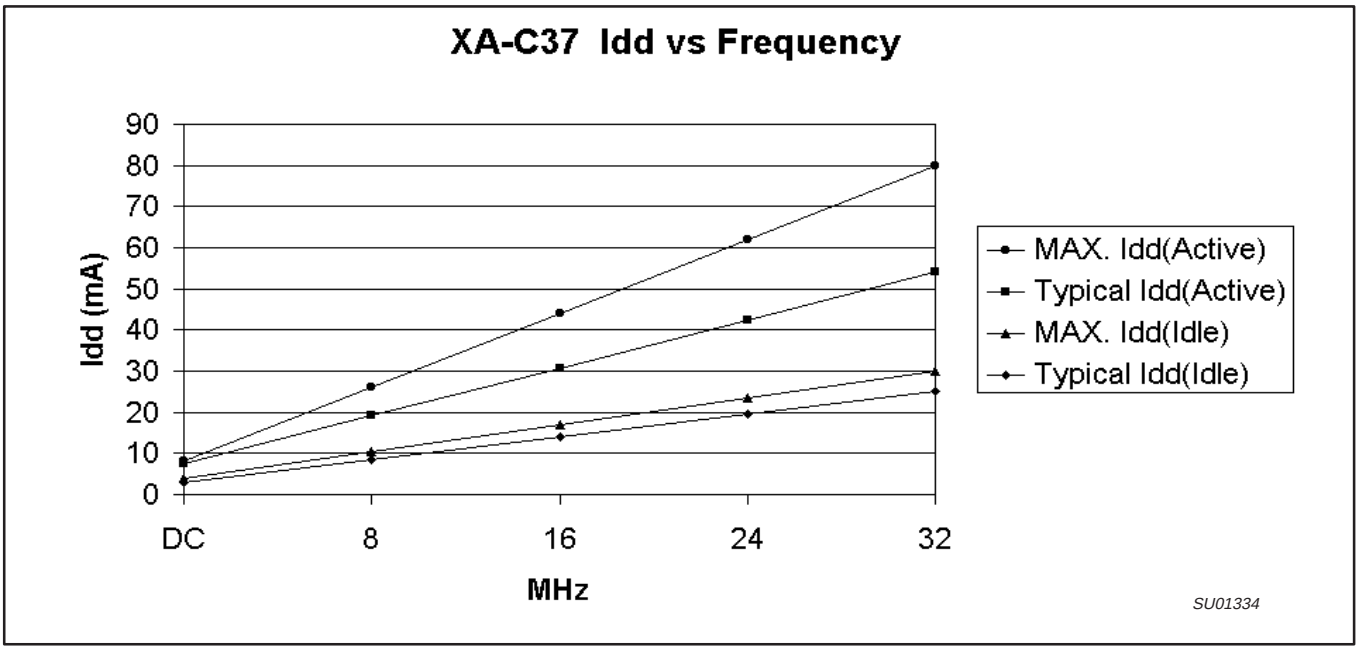


Figure 31. I_{DD} vs. Frequency at V_{DD} = 5.0V

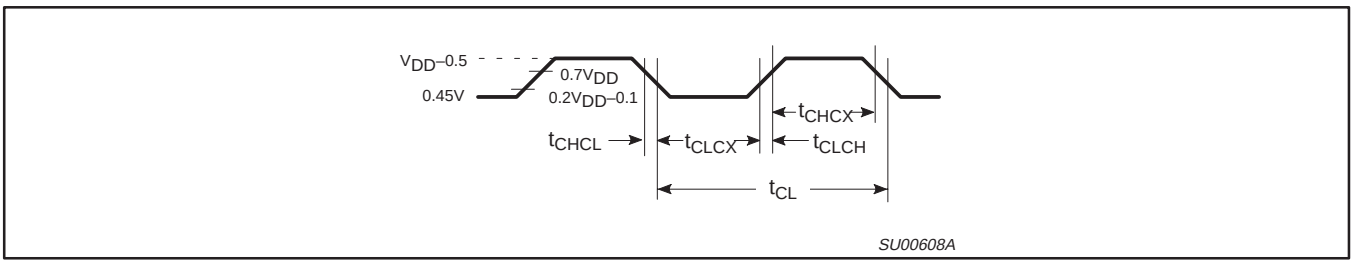


Figure 32. Clock Signal Waveform for I_{DD} Tests in Active and Idle Modes

Note: $t_{CLCH} = t_{CHCL} = 5$ ns

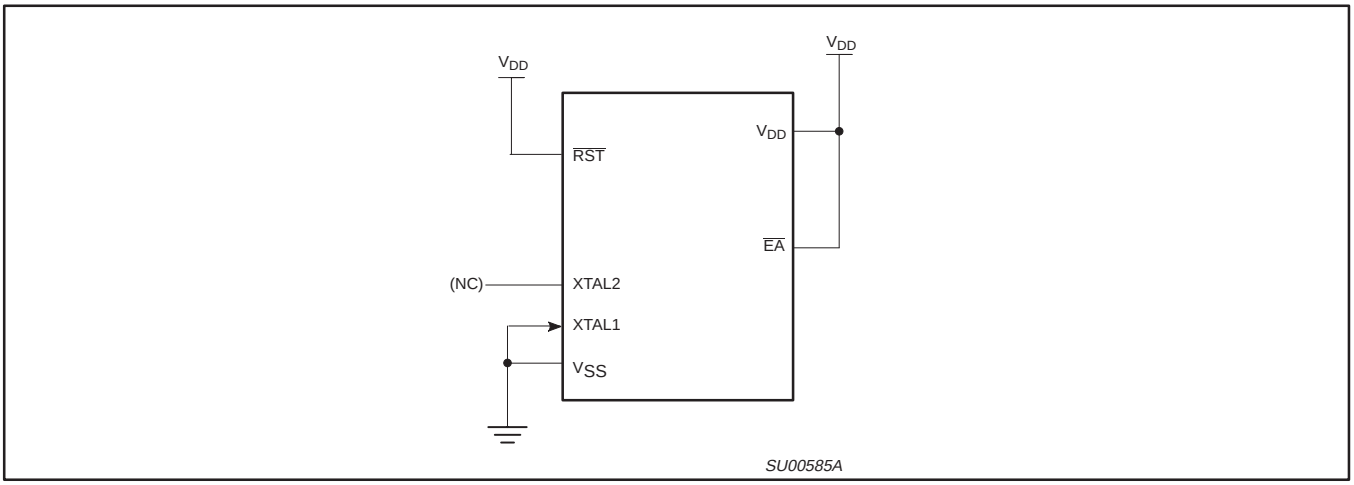


Figure 33. I_{DD} Test Condition, Power-Down Mode

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.

XA-C3 OVERVIEW

Introduction

The XA-C3 is a member of the Philips XA (eXtended Architecture) family of high performance 16–bit single–chip microcontrollers. Combined in the XA-C3 are an array of standard microcontroller peripherals, a powerful CAN 2.0A/B controller, and a unique CAN "Message Management" engine which provides integrated hardware support for most **CAN T**ransport **L**ayer **(CTL**) protocols.

Integrated into the XA-C3 microcontroller is the CAN Controller Core from the award–winning[1] Philips SJA1000 **C**AN (2.0A/B) **D**ata **L**ink **L**ayer (**CDLL**) device. Since 1986, CAN Users have developed high–level **CAN T**ransport **L**ayers. The XA-C3 implements many such **CTL** concepts *in hardware,* including automatic assembly of multi–frame Fragmented messages. In fact, the XA-C3 is the first chip with hardware **CTL** support. The CAN module embedded in the XA-C3 provides far greater CAN functionality and power than any existing CAN product*, including the SJA 1000 itself*.

CTL protocols such as Device Net, CANopen and OSEK deliver long Messages distributed over many CAN Frames (see Figure 34).

This method is called Fragmented (or, in European terminology, Segmented) messaging. The individual frames, forming a complete CTL message, are interleaved on the CAN bus together with frames belonging to other (unrelated) CTL/CAN messages. The XA-C3 transparently re–assembles up to 32 concurrent, interleaved CTL Messages in *hard*ware as directed by a new, powerful ID Screener technology with 32 Screeners and 32 DMA channels. An on–chip, 512–byte, CTL/CAN Message Buffer RAM provides single–chip storage for Receive and Transmit. This Buffer RAM is easily extended (off–chip) to accommodate up to 32 messages of 256 bytes each.

The XA-C3 provides these powerful CAN 2.0A/B and **CTL** features while maintaining **pin and function compatibility** with the present XA-G3; the new CAN Rx/Tx pins have been assigned to XA-G3 no–connects. Thus, today's XA-G3 based products can incorporate **CTL**/CAN in new designs. XA-G3 software is preserved while XA-C3 features immediately upgrade present XA-G3 board layouts to **CTL**/CAN. Additionally, the FullCAN (CAN) features of the XA-C3 can be used independently of **CTL**.

[1] **C**AN **i**n **A**utomation 1997

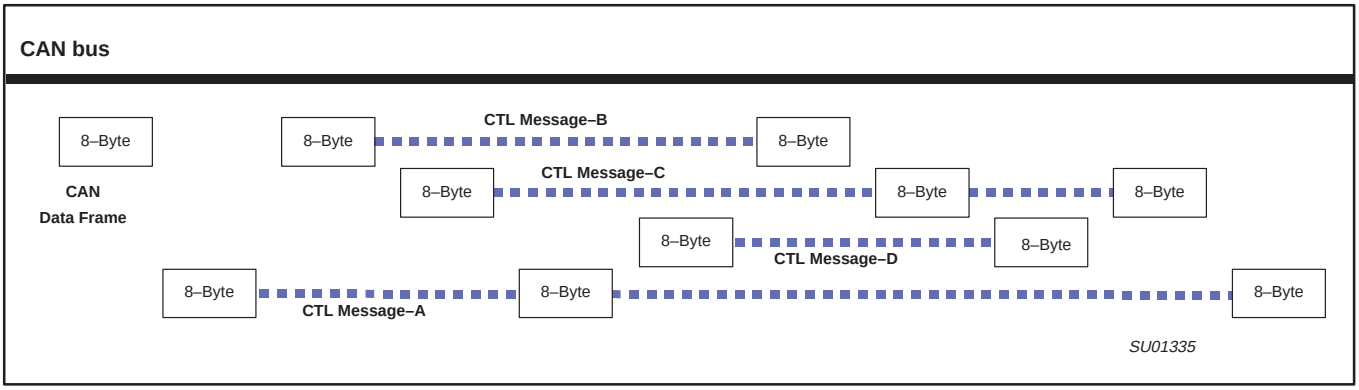


Figure 34. Interleaved CAN Data Frames

Definition of Terms

Standard and Extended CAN Frames
See Figure 35.

Acceptance Filtering
The process a CAN device implements (usually) in hardware to determine if a CAN frame should be accepted or ignored and, if accepted, to store that frame in a pre–assigned Message Object.

Message Object
A Receive RAM Buffer of pre–specified size (up to 256 bytes for CTL messages) and associated with a particular Acceptance Filter or, a Transmit RAM Buffer which the User preloads with all necessary data to transmit a complete CAN Data Frame.

CAN Arbitration ID
An 11–bit (Standard CAN 2.0A Frame) or 29–bit (Extended CAN 2.0B Frame) *identifier* field placed in the CAN Frame Header. This ID field is used to arbitrate Frame access to the CAN bus. Also used in **Acceptance Filtering** for CAN Frame reception and Transmit Pre–Arbitration.

Screener ID
A 30–bit field extracted from the incoming message which is then used in **acceptance filtering**. The screener ID includes the **CAN Arbitration ID** and the IDE bit, and can include up to 2 Data Bytes. These 30 extracted bits are the information qualified by **Acceptance Filtering**.

Match ID
A 30–bit field pre–specified by the User to which the incoming **Screener I**D is compared. Individual **Match Id**s for each of the 32 objects are programmed by the User into designated memory mapped registers.

Mask
A 29–bit field pre–specified by the User which can override (**Mask**) a **Match ID** comparison at any particular bit (or, combination of bits) in an **Acceptance Filter**. Individual **Mask**s, one for each **Message Object**, are programmed by the User in designated **MMR**s. Individual **Mask** patterns assure that *single* Receive Objects can Screen for multiple *acknowledged* **CTL**/CAN Frames and thus minimize the *number* of Receive Objects that must be dedicated to such *lower priority* Frames. This ability to **Mask** individual **Message Objects** is an important *new* **CTL** feature.

CTL
CAN Transport Layer. A generic term for any high–level protocol, which extends the capabilities of CAN while employing the CAN physical layer, CAN frame format and, adheres to the CAN

specification. Among other things, CAN Transport Layers permit transmission of Messages which exceed the 8 byte Data limit inherent to CAN Frames.

Fragmented Message
A lengthy message (in excess of 8 bytes) divided into data packets and transmitted using a sequence of individual CAN Frames. The specific ways that sequences of CAN Frames construct these lengthy messages is defined within the context of a specific **CAN T**ransport **L**ayer. The XA-C3 automatically re–assembles these packets into the original, lengthy message in hardware and reports (via interrupt) when the completed message is available as an associated **Receive Message Object**.

Message Buffer
A block of locations in XA Data memory where incoming (received) messages are stored or where outgoing (transmit) messages are staged.

MMR
Memory Mapped Register. An on–chip command/control/status register whose address is mapped into XA Data memory space and is accessed as Data memory by the XA processor.

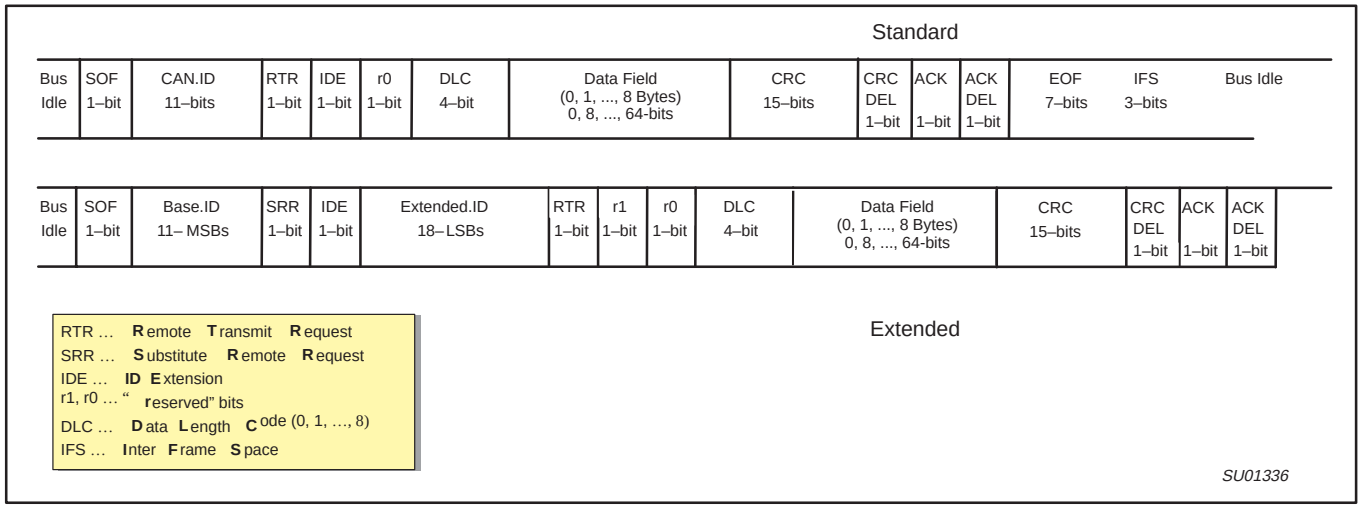


Figure 35. CAN Frame Formats

CTL/CAN Functionality of the XA-C3

Message Objects / Message Management

- The XA-C3 allows the User to define up to 32 separate CTL/CAN Message Objects.
- Any of these 32 objects can be designated as either a Receive or Transmit objects.
- Any/all of the (up to 32) Receive Objects may be enabled to hardware assemble multi frame “Fragmented” messages. For Receive Objects so enabled, **CTL**/CAN hardware interrupts the XA-C3 only at the *completion* of a multi–frame message which is assembled in a contiguous fashion and stored in the Receive message buffer associated with that object. At any given time, XA-C3 may actively assemble (up to) 32 interleaved **CTL** messages.
- Receive objects may also be used as circular CAN Frame buffers, to store up to 28 CAN frames of 8 data bytes each, between CPU interrupts.
- Receive Objects, *not* enabled to hardware–assemble messages, treat CAN2.0A/B Frames as complete (single–frame) messages and are thus *backward* compatible with today’s FullCAN Message Objects that store *single* CAN frames.
- XA-C3 supports most **CTL**/CAN protocols, i.e., Device.Net, CANopen and OSEK.
- Generally, *hardware* “Message–Management” on XA-C3 reduces the **CTL** instruction bandwidth of today’s **CTL** message processing from 80% to as low as 10%.

Acceptance Filtering
The XA-C3 provides extensive ID Screening/Filtering for 32 Message Objects. Each object has a full 30 bits of filter Match over the **CTL**/CAN ID Fields *as–well–as* 29 bits of Mask … *per object.* That is, *any* combination of (up to) 30 bits in the ID Fields may be Masked out (*“don’t care”*) and/or Matched on an *object–by–object basis*.

Message Storage
Each of the 32 Message Objects has its own designated message buffer space within the Data memory space addressed by the XA processor. The size of each message buffer is independently User specified up to a max of 256 bytes/object. **CTL** messages that exceed the 256 byte/object limit can be accommodated with simple software intervention.

The XA-C3 includes a 512 byte, on–board Message Buffer RAM where some (or all) of the 32 (Rx/Tx) message buffers may reside. Message Buffer RAM can be mapped anywhere in the 16 MByte Data memory space addressed by the XA and can be extended off–chip to a maximum of 8 KBytes. This off–chip expansion ability can accommodate up to thirty–two, 256–byte message buffers.

Transmit Pre–Arbitration
Two (2) options are available to pre–arbitrate among pending (currently enabled) transmit objects. A *default* option selects and transmits the object of highest–priority CAN arbitration ID (the *same* criteria that arbitrates access to CAN bus itself). Transmit object *number* serves as a secondary tie–breaker for *queued* transmit

objects having the *same* ID. An *alternate* option bases transmit pre–arbitration *exclusively* on transmit object *number,* i.e., independent of arbitration ID.

Remote Frame Handling
The XA-C3 supports Remote CAN Frames.

MEMORY MAPS

Data Memory Space

1K byte of internal data memory (Scratch Pad) populates the very bottom of data memory space, in Segment 0 by definition. The Memory Mapped Registers and the on–chip XRAM can also be mapped into Segment 0 (as shown in Figure 36), or into any other segment.

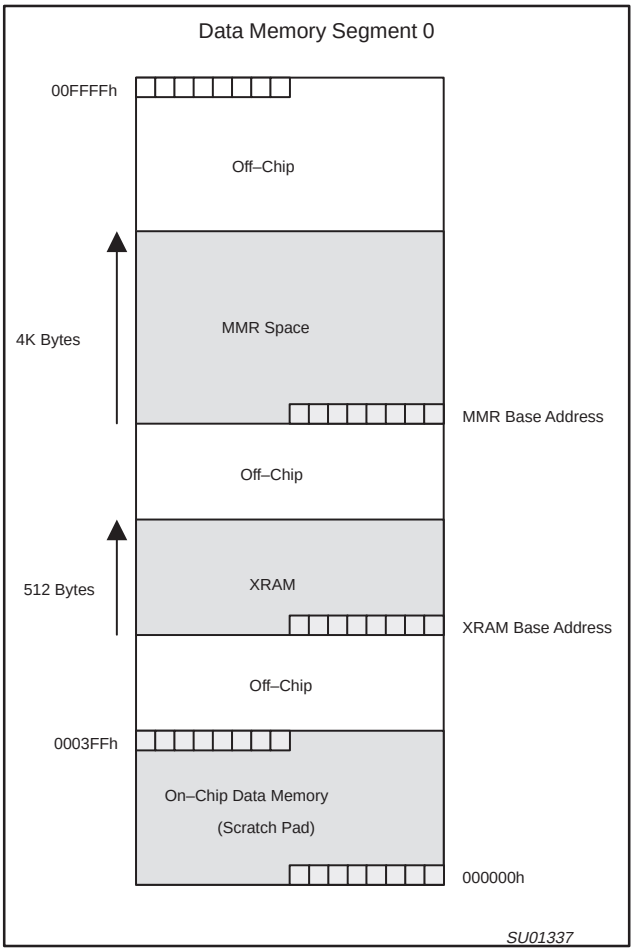


Figure 36. MMRs and XRAM mapped into Segment 00h.

Code Memory Space

32K Bytes of Internal Code Memory populate addresses 000000h – 007FFFh of code memory space. As shown in Figure 37, code memory can be extended off–chip, if desired, starting at address 008000h. The code memory address space extends to 0FFFFFh.

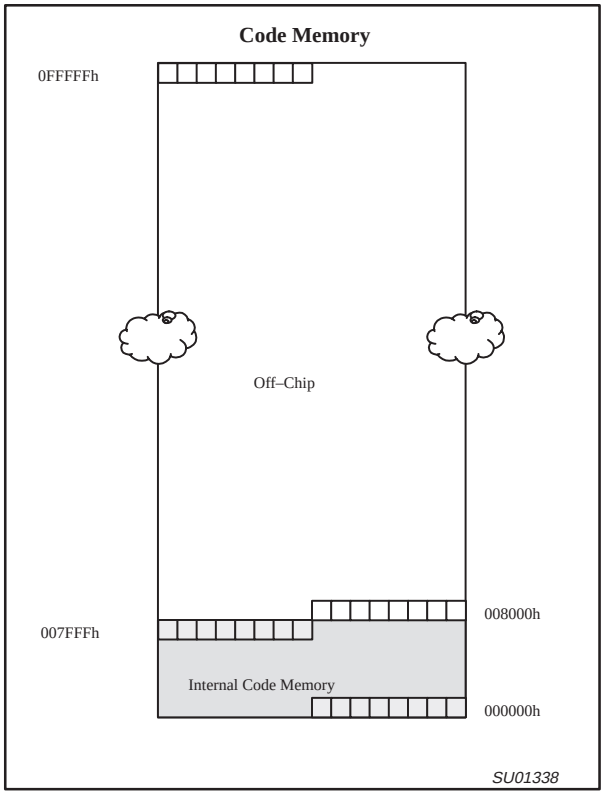


Figure 37. External Code Memory starts at 008000h.

CAN CORE BLOCK (CCB)

CAN Bus Timing

CAN System Clock
The CCB has a programmable internal system clock, whose period is denoted by tSCL. The CAN System Clock is derived from the XA Oscillator Clock based on the following expression:

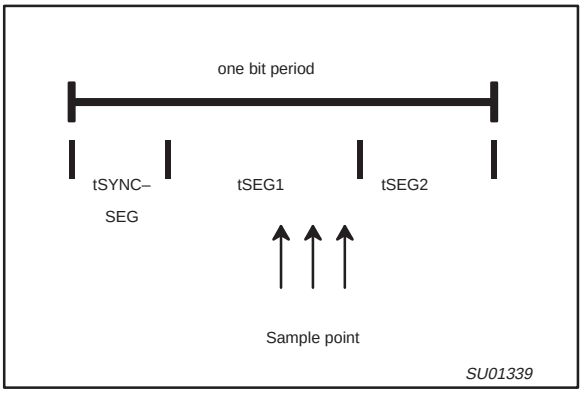
- tSCL =2 ∗ tCLK ∗ (32 ∗ BRP.5 + 16 ∗ BRP.4 + 8 ∗ BRP.3 + 4 ∗ BRP.2 + 2 ∗ BRP.1 + BRP.0 + 1)

where tCLK is the period of the XA Oscillator Clock, and BRP.5 – BRP.0 are bits in the MMR **CAN B**us **T**iming **R**egister (CANBTR). The length of a bit period in a CAN Frame is expressed in terms of number of CAN System Clocks.

Samples Per Bit
The number of samples per bit is determined by the value of the SAM bit in CANBTR.

- SAM = 0 The bus is sampled once per bit (as shown below)
- SAM = 1 The bus is sampled three times per bit (as shown below)

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.

OBJECT N MATCH ID FIELD (MNMIDH AND MNMIDL)

Mid28 – Mid18	Mid17 – Mid10	Mid9 – Mid2	Mid1	Mid0	MIDE

OBJECT N MASK FIELD (MNMSKH AND MNMSKL)

Msk28 – Msk18	Msk17 – Msk10	Msk9 – Msk2	Msk1	Msk0

SCREENER ID FIELD (ASSEMBLED FROM INCOMING BIT–STREAM)

CAN ID.28 – CAN ID.18	Data Byte 1 [7:0]	Data Byte 2 [7: 0]	x	x	IDE

NOTE:
1. For a Standard CAN Frame Message Object, only 27 bits plus IDE (11 bits of CAN ID + 2x8 bits + IDE) from the incoming message are routed to the acceptance filter. The User is therefore required to set the Msk1 and Msk0 bits in the Mask field for that object (i.e., “don’t care”). The IDE bit is not Maskable.

In many applications based on Standard CAN frames, either Data Byte 1, Data Byte 2, or both do not participate in Acceptance Filtering. Therefore, the User is required to Mask out the unused Data Byte(s).

Screener ID Field for Extended CAN Frame
The following table shows how the Screener ID field is assembled from the incoming bits of an Extended CAN Frame, and how it is compared to the Match ID and Mask fields of Object n. **Note:** The IDE bit is not Maskable.

OBJECT N MATCH ID FIELD (MNMIDH AND MNMIDL)

Mid28 – Mid18	Mid17 – Mid10	Mid9 – Mid2	Mid1	Mid0	MIDE

OBJECT N MASK FIELD (MNMSKH AND MNMSKL)

Msk28 – Msk18	Msk17 – Msk10	Msk9 – Msk2	Msk1	Msk0

SCREENER ID FIELD (ASSEMBLED FROM INCOMING BIT–STREAM)

CAN ID.28 – CAN ID.0	IDE

MnMIDH: Message n Match ID High Word
- Address: MMR base + n0h
- Access: Read, write. Word access only.
- Reset value: xxxxh

MNMIDH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mid28	Mid27	Mid26	Mid25	Mid24	Mid23	Mid22	Mid21	Mid20	Mid19	Mid18	Mid17	Mid16	Mid15	Mid14	Mid13

MnMIDL: Message n Match ID Low Word
- Address: MMR base + n2h
- Access: Read, write. Word access only.
- Reset value: xxxxxxxxxxxxxx00b (unused bits are always read as ‘0’)

MNMIDL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mid12	Mid11	Mid10	Mid9	Mid8	Mid7	Mid6	Mid5	Mid4	Mid3	Mid2	Mid1	Mid0	MIDE	–	–

MnMSKH: Message n Mask High Word
- Address: MMR base + n4h
- Access: Read, write. Word access only.
- Reset value: xxxxh

MNMSKH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Msk28	Msk27	Msk26	Msk25	Msk24	Msk23	Msk22	Msk21	Msk20	Msk19	Msk18	Msk17	Msk16	Msk15	Msk14	Msk13

NOTE:
1. Note: For transmit objects, the frame information is programmed in this register.

MnMSKL: Message n Mask Low Word
- Address: MMR base + n6h
- Access: Read, write. Word access only.
- Reset value: xxxxxxxxxxxxx000b (unused bits are always read as ‘0’)

MNMSKL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Msk12	Msk11	Msk10	Msk9	Msk8	Msk7	Msk6	Msk5	Msk4	Msk3	Msk2	Msk1	Msk0	–	–	–

MnCTL: Message n Control Register

- Address: MMR base + n8h
- Access: Read, write. Byte or word access.
- Reset value: 00000xxxb (unused bits are always read as '0')

MNCTL

7	6	5	4	3	2	1	0
–	–	–	OBJ_EN	INT_EN	Tx/Rx	FRAG	RTR_EN

OBJ_EN — Object Enable. Enables the Message Object for receive or transmit. 0 = disabled, 1 = enabled.

INT_EN — Message–Complete Interrupt Enable. Specifies whether or not a Tx or Rx Message–Complete for this object will cause the object's Message–Complete Interrupt to be generated. 0 = disabled, 1 = enabled.

Tx/Rx — Transmit or Receive. Specifies whether this is a transmit or receive Message Object. 0 = transmit object, 1 = receive object.

FRAG — Fragmented Message Enable. Only relevant for receive Message Objects. Enables automatic assembly of Fragmented Rx messages. If disabled, messages received by this object are assumed to be single–frame, or will be assembled by User software. 0 = disabled, 1 = enabled. **Note:** Masking of the CAN Identifier field by User software, for the purpose of Message Object grouping, is disallowed for objects using hardware Fragmentation assembly. However, Masking of unused bit positions in the "screener", such is mandatory in all cases.

RTR_EN — Enable Request To Transmit. 0 = the object is not enabled for RTR handling, 1 = the object is enabled for RTR handling. See section entitled *RTR Handling*, page 46.

Message Storage

When an incoming message frame has passed acceptance filtering, it will be automatically stored in data memory via DMA. Each message will be stored in its corresponding buffer area. On setup, the User is responsible for assigning a unique buffer location for each Message Object. This is specified in the object's MnBLR register. The User is also required to set up the size of each buffer in the MnBSZ register.

The XA-C3 provides a total of 512 bytes of on–chip message buffer RAM (XRAM) which may contain part or all of the CAN/CTL (transmit & receive) message buffer space. See Section entitled *On-Chip Message Buffer RAM (XRAM)* on page 55 for details.

Note: The following discussion concerning message buffer registers applies to transmit message retrieval as well as receive message storage.

MBXSR (applies to all objects)

- Address: MMR base + 291h
- Access: Read, write, byte or word
- Reset value: FFh

MBXSR

7	6	5	4	3	2	1	0
a23 – a16 of all message buffer (and XRAM) base addresses							

All 32 message buffers must reside within the same 64K memory page. This page is specified by the contents of the MBXSR (Message Buffer and XRAM Segment Register) register. Also, the 512 byte on–chip message buffer RAM (XRAM) is always positioned within that same 64K page pointed to by MBXSR.

Note: The XA-C3 brings out only 20 address lines to package pins. It can, therefore, only address 1MByte of off–chip data memory (a maximum of sixteen 64K segments). As a result, for the XA-C3, the four most significant bits of the MBSXR register must be set to '0000' if External RAM is to be used for any portion of the message buffer space.

MnBSZ: Message n Buffer Size Register

- Address: MMR base + nCh
- Access: Read–modify–write, byte or word access.
- Reset value: 00000xxxb

MNBSZ

7	6	5	4	3	2	1	0
–	–	–	–	–	BSZ.2	BSZ.1	BSZ.0

The size of an object's message buffer is specified with the 3–bit field MnBSZ[2:0] as shown in Table 23.

Table 23. Allowable Message Buffer Sizes

BSZ.2	BSZ.1	BSZ.0	Buffer Size
0	0	0	2 Bytes
0	0	1	4 Bytes
0	1	0	8 Bytes
0	1	1	16 Bytes
1	0	0	32 Bytes
1	0	1	64 Bytes
1	1	0	128 Bytes
1	1	1	256 Bytes

The User should bear in mind that only data bytes and (for Rx only) a single byte of frame or byte–count information is stored in the message buffer. Space does not need to be allocated for headers, Fragmentation information, etc. See the Rx memory buffer images below.

MnBLR: Message n Buffer Location Register

- Address: MMR base + nAh
- Access: Read, write. Word access only.
- Reset Value: xxxxh

MNBLR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
a15 – a0 of object n message buffer base address															

The Buffer Location Register holds the least significant 16 bits of the object's message buffer base address. The upper 8 bits of the 24–bit address, for *all* Message Objects, are specified by the contents of MBSR. Thus, the message buffers for all Message Objects must reside within the same 64Kbyte segment.

For any message buffer which is to be mapped into the on–chip message buffer RAM (XRAM), MnBLR bits [15:9] must match XRAMBASE bits [15:9].

Important constraints:

- 256–byte buffers *must* be located at a 256–byte boundary (MnBLR[7:0] = 00000000b)
- 128–byte buffers *must* be located at a 128–byte boundary (MnBLR[6:0] = 0000000b)
- 2–byte buffers *must* be located at a 2–byte boundary (MnBLR[0] = 0)

Note: Message buffer logical address spaces must always adhere to the above constraints. However, there are at least two cases in which the User must initialize the MnBLR register such that it does *not* point to the actual base location of the logical buffer space when reception begins. For details, please see sections entitled *Fragmented Messages in OSEK* on page 44 and *Fragmented Messages in CANopen* on page 44.

Message Assembly

The DMA will transfer the accepted message from the pre–buffer to the message buffer area one word at a time, starting from the address pointed to by [MBXSR][MnBLR]. Every time DMA transfers a byte or word, it has to request the bus. Once granted, it will write data from the 13 byte receive pre–buffer to memory. The DMA will keep requesting the bus, writing message data sequentially to the memory until the whole frame is transferred. When DMA has successfully transferred data from an incoming CAN message to memory, the contents of the receive buffer will depend on whether the message was non–Fragmented (single frame) or Fragmented.

Non–Fragmented Message Assembly

Since Masking is permitted on the 11– or 29–bit CAN Identifier for Message Objects with FRAG = 0, the complete CAN ID for the incoming message is written into the MnMIDH and MnMIDL registers when the DMA has completed. This will permit the User application to see the exact CAN identifier which resulted in the match.

As a result of the above mechanism, the contents of MnMIDH and MnMIDL can change every time an incoming frame is accepted. Since the incoming frame has to pass the Match before it can be accepted, only the bits that are Masked out will change. Therefore, the criteria for Match and Mask will not change as a result of an accepted incoming frame (see Figure 38).

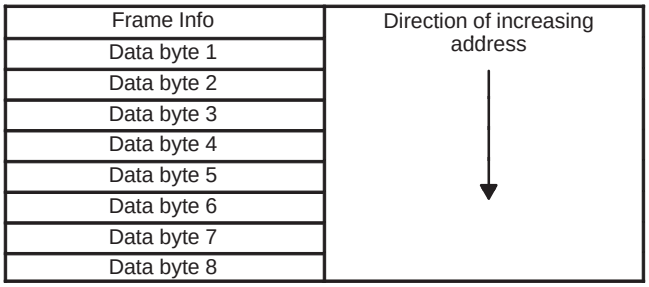


Figure 38. Memory Image for Non–Fragmented Messages

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.

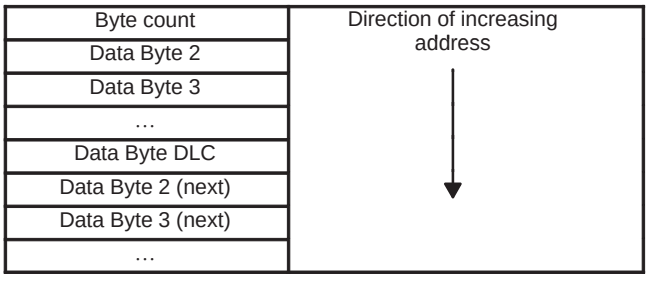


Figure 40. Memory Image for Fragmented CTL Messages (FRAG = 1 and Prtcl1 Prtcl0 ≠ 00)

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.

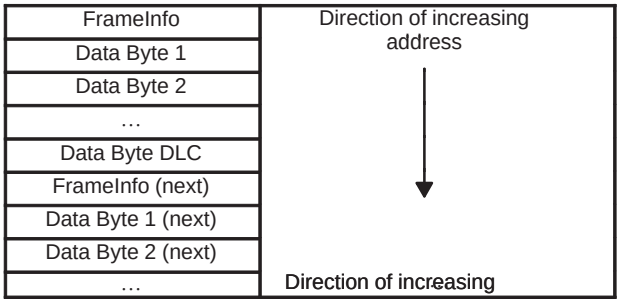


Figure 41. Memory Image for CAN Frame Buffering (FRAG = 1 and Prtcl1 Prtcl0 = 00)

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

DMA, and then interrupt the CPU that a complete message has been received.

Since Data Byte 1 of each frame contains the Fragmentation information, it will never be stored in the CTL message buffer, thus each frame will have up to seven bytes of data stored. After the entire message is received, the message buffer will contain all of the actual informational data bytes received (exclusive of Fragmentation information bytes) plus the Byte Count at location 00 which will contain the total number of informational data bytes stored.

Fragmentation Error

By looking at the Fragmentation information, the message handler can determine the first frame , the middle frames, the end frame of the message, and each sequence number. In the case of CANopen, there is no sequence number but rather a one bit field that toggles each frame. If a Fragmentation error occurs, the message handler will reset the byte count, address pointer, and generate an interrupt to the CPU. At this point the CTL message buffer is determined to be invalid.

Fragmentation checking is disabled for all objects when CAN is the system protocol (Prtcl[1:0] = 00).

Fragmentation error occurs only one way:

1. When the message handler receives a frame where the sequence number is NOT one greater than that of the previous frame. Or in the case of CANopen, the toggle bit has not toggled.

Fragmented Messages in OSEK

There are several important items that must be kept in mind with regard to hardware assembly of Fragmented OSEK messages. For a complete discussion, please see the XA-C3 User Manual. These items are summarized below:

- The OSEK FirstFrame cannot be treated as part of the Fragmented message, but must be handled as a completely separate, single–frame, non–Fragmented message. However, the FirstFrame *may* contain the first several bytes of User–data.
- For the object receiving the forthcoming message Fragments, the MnFCR register must be initialized by the User to point at an address *other* than the buffer base location. This can be byte offset ‘1’ or some other, more strategically chosen location. Since there will be no FirstFrame received for this object, there will be no write of 00h to the buffer base location, by DMA, at the beginning of the message.
- The Fragment Count Register (MnFCR) of the object receiving the message Fragments must be initialized by the User before enabling the object for receive. The initial value written to MnFCR must be identical to the SequenceNumber of the first ConsecutiveFrame that arrives (typically 0h).
- There is no “Last Frame” encoding for OSEK. Therefore, there will be neither an Rx Message Complete Status Flag, nor an interrupt, nor a Byte Count write associated with Rx Message Complete, at the conclusion of a Fragmented message. However, by carefully choosing the initial value for the MnBLR register, the User can arrange to get an Rx Buffer Full interrupt, and the associated Rx Buffer Full Byte Count write, instead.

Fragmented Messages in CANopen

In a CANopen system, the software will need to write to the object‘s Fragment Count Register (MnFCR) to initialize the toggle bit prior to receiving the first frame of any new message which requires hardware Fragmentation assembly. This bit will have to be initialized to the same state that will be received in the 1st packet (typically 0). This bit will need to be initialized each time a new channel is established, even if none of the other parameters change (e.g., Match, Mask, buffer location, buffer size, etc.).

Since the hardware cannot detect a message start, there can be no semaphore write to the bottom of the buffer space at the start of a new Fragmented message (for a discussion of the semaphore, see the section entitled *Using the Semaphore Bits, SEM1 and SEM0* on page 46. This location must still be left free for the hardware to write the byte count into at the end of the message. This means that for CANopen ***Fragmented*** messages (only Fragmented) **the software must initialize the address pointer to location ‘1’ of the designated receive buffer, not location ‘0’ as it does in DeviceNet.** It also implies, of course, that the software loses the ability to check the semaphore to determine if message reconstruction is currently in progress.

Essentially, the hardware will treat the first frame of a multi–frame CANopen message exactly the same as intermediate frames.

Auto–Acknowledge in CANopen

A Fragmented (Segmented) CANopen message may need to be acknowledged on a frame by frame basis. The XA-C3 provides hardware support for this process, with no CPU intervention. Of course the User may elect not to auto–acknowledge, or to implement the acknowledge function in software.

Suppose Message Object n (n = 0…31) is enabled for receive, with the FRAG bit set. If the high level protocol is CANopen, as selected in the GCTL register, then the following steps must be taken to ensure that CANopen frames are automatically acknowledged:

- Set the AUTO_ACK bit in GCTL.
- Set up a transmit object sequential to the CANopen receive object, i.e., the object number set to be n+1. Set the FRAG bit for this object.
- It is important NOT to set the OBJ_EN bit for the transmit message.

With the above setup, the XA-C3 will automatically generate a transmit frame upon successful reception of a CANopen frame. The User must setup the screener ID for the Tx frame in the M_{n+1}MIDH and M_{n+1}MIDL registers, the RTR bit in M_{n+1}CNTL[0], and the DLC in M_{n+1}MSKH[3:0]. The User must also store the proper “Acknowledge Byte”, as defined by the protocol specification, in byte offset 0 of the Tx object’s message buffer. Bit position [4] is a don’t care, because the XA-C3 will automatically insert the toggle bit value from the incoming frame into the toggle bit position of the outgoing auto–acknowledge frame. The format for storing the Acknowledge Byte is shown below in Table 24 (subject to change without notice by the CiA).

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.

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

condition will **always** be generated at the end of each successfully transmitted frame.

There is a control bit associated with each Message Object indicating whether a message complete condition should generate an interrupt, or just set a “message complete status flag” (for polling) without generating an interrupt. This is the INT_EN bit in the object’s MnCTL register, MnCTL[3].

There are two 16–bit MMRs, MCPLH and MCPLL, which contain the message complete status flags for all 32 objects. When a message complete (Tx or Rx) condition is detected for a particular Message Object, the corresponding bit in the MCPLH or MCPLL register will be set. This will occur regardless of whether the INT_EN bit is set for that object (in MnCTL[3]), or whether message complete status flags have already been set for any other objects.

In addition to these 32 message complete status flags, there is a Tx Message Complete Interrupt Flag and an Rx Message Complete Interrupt Flag (CANINTFLG[1] and CANINTFLG[0] respectively), which will generate the actual Event interrupt requests to the XA core. When an end of message occurs**,** at the same moment that the message complete status flag is set, the appropriate Tx or Rx Message Complete Interrupt flip–flop will also be set provided that INT_EN = 1 for the object, and the interrupt is not already set and pending.

The message complete interrupt flags should always be cleared using the 2–step process outlined below:

1. Message Complete Status Flags for all interrupt enabled objects of that type (Tx or Rx) should first be cleared by writing ‘1’ to their bit positions.
2. The Message Complete Interrupt Flag itself can now be cleared by writing ‘1’ to its bit position.

Warning: Message Complete Interrupt Flags may be cleared *before* all Message Complete Status Flags for interrupt enabled objects of that type (Rx or Tx) are removed. However, the interrupt flag will not be reset to ‘1’ by hardware, unless a new message complete condition occurs for some other interrupt enabled object. Therefore, it is strongly recommended that Message Complete Interrupt Flags be cleared only after removing all Message Complete Status Flags for interrupt enabled objects of the same type, and at the end of the interrupt service routine.

The newest addition is the **M**essage **C**omplete **I**nfo **R**egister (MCIR). MCIR[4:0] will encode the lowest object number of all objects whose INT_EN bits are set **AND** who currently have a message complete condition (objects whose message complete status flags are set). A ‘1’ in bit 5 means that *one or more* objects whose INT_EN bits are set have a message complete condition. A ‘0’ in bit 5 means that *no* objects whose INT_EN bits are set have a message complete condition. Bits 6 and 7 are unused.

Rx Buffer Full Interrupt

As successive frames of a Fragmented message are transferred by DMA into an object’s message buffer, it is possible to reach the end of the designated buffer space before the complete message has been received. When this occurs, it is necessary for the processor to intervene in order that the remainder of the message be stored without any loss of data.

If the system protocol is DeviceNet, CANopen, or OSEK, then a message buffer is considered full when the number of bytes remaining in the buffer space, *at the end of a complete frame*, is less than seven.

If the system protocol is CAN, i.e., [Prtcl1 Prtcl0] = 00, then 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.

This condition could occur if the application has underestimated the message size, or deliberately established a small buffer to conserve memory. The condition will always occur with messages containing more than 255 data bytes (excluding Fragmentation information bytes), since the maximum message buffer size is 256 bytes.

The following discussion only applies to frames which are ***not*** the last frame of a message (which also, necessarily, excludes non–Fragmented, single–frame messages). After DMA of the last data byte of the frame is completed, a check will be performed to determine if the current byte count is less than 7 bytes from the end of the assigned message buffer. If it is, then there is the potential for the next frame to overrun the buffer. We will consider this “less–than–seven–bytes–remaining” situation to be a buffer–full condition. When this condition is detected, the following will occur:

- The ***current*** byte count will be written into buffer location ‘0’ except in CAN systems. If [Prtcl1 Prtcl0] = 00, no byte count will be written.
- The address pointer will be initialized to location ‘1’
- The Rx Buffer Full interrupt will be generated

As subsequent frames are received, the data bytes will be stored, beginning at location ‘1’. The semaphore byte will ***not*** be written to again, since message assembly is still in progress. Once the end–of–message is finally received, the DMP will respond as usual, writing the byte–count to location ‘0’ and setting the Rx Message Complete Interrupt Flag. **Note that the byte count will now reflect the number of bytes received since the buffer wrapped around, *not* the total number of bytes in the message**. Software will have to calculate the difference.

The software has two choices as to how to respond to this Rx Buffer Full interrupt:

1. Read the contents of the buffer, thereby freeing up space in the buffer for any remaining frames.
2. Reposition the buffer by modifying the address pointer. Note: The least significant bit of the address pointer will already be set to ‘1’, and must remain so. The bottom location must be reserved for the byte–count which will be written at the end of the message.

If option 1 is selected, the software will retrieve the current byte count from the bottom of the buffer. It will then retrieve the designated number of data bytes from the buffer. Subsequent data received will be loaded into the buffer, beginning at location ‘1’. When the end–of–message occurs, the byte–count stored in location ‘0’ will indicate how many new bytes have been received which must now be retrieved.

For option 2, subsequent bytes will actually be written into a different buffer space, elsewhere in memory. The processor can wait until the entire message is received before retrieving any data. At that time, the ‘0’ location of the 1st buffer will indicate how many bytes are stored there, and likewise for the second buffer (or third or so on). Note that option 2 is far more efficient and can be implemented with very few instructions.

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.

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)

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

MCPLH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Obj31	Obj30	Obj29	Obj28	Obj27	Obj26	Obj25	Obj24	Obj23	Obj22	Obj21	Obj20	Obj19	Obj18	Obj17	Obj16

MCPLL (Message Complete Status Flags Low)
- Address: MMR base + 224h
- Access: Read/Clear, byte or word
- Reset Value: 0000h

MCPLL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Obj15	Obj14	Obj13	Obj12	Obj11	Obj10	Obj9	Obj8	Obj7	Obj6	Obj5	Obj4	Obj3	Obj2	Obj1	Obj0

TxERC (Tx Error Counter)
- Address: MMR base + 274h
- Access: Read, write, R/M/W, byte or word
- Reset Value: 00h

TXERC

7	6	5	4	3	2	1	0
TC_7	TC_6	TC_5	TC_4	TC_3	TC_2	TC_1	TC_0

The Tx Error Counter can only be written to when the CAN Core is in Reset mode. Hardware will preset the register to 128 when a Bus–Off condition occurs. See the section entitled *Bus Off* on page 50 for details.

RxERC (Rx Error Counter)
- Address: MMR base + 275h
- Access: Read, write, R/M/W, byte or word
- Reset Value: 00h

RXERC

7	6	5	4	3	2	1	0
RC_7	RC_6	RC_5	RC_4	RC_3	RC_2	RC_1	RC_0

The Rx Error Counter can only be written to when the CAN Core is in Reset mode. When a Bus–Off condition occurs, this register is cleared to 00h.

EWLR (Error Warning Limit Register)
- Address: MMR base + 276h
- Access: Read, write, R/M/W, byte or word
- Reset Value: 96h

EWLR

7	6	5	4	3	2	1	0
EWL_7	EWL_6	EWL_5	EWL_4	EWL_3	EWL_2	EWL_1	EWL_0

ECCR (Error Code Capture Register)
- Address: MMR base + 278h
- Access: Read, write, R/M/W, byte or word
- Reset Value: 00h

ECCR

7	6	5	4	3	2	1	0
EC1	EC0	State					

The Error Code Capture Register contains detailed information about the most recent Bus Error. See Table 25 for details. The register must be read in order to be re–enabled for capturing the next error code, as well as to clear the BERR status flag. This register should be read before enabling the Bus Error interrupt.

ALCR (Arbitration Lost Capture Register)
- Address: MMR base + 27Ah
- Access: Read, write, R/M/W, byte or word
- Reset Value: 00h

ALCR

7	6	5	4	3	2	1	0
–	–	–	Bit Number				

The ALCR latches the bit number in the CAN Identifier where the most recent Arbitration Lost occurred. See Table 26 for details. The register must be read in order to be reenabled for capturing the next arbitration lost code, as well as to clear the ARBLST status flag.

This register should be read before enabling the Arbitration Lost interrupt.

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

must be readable by the processor as an MMR. If the processor is about to put the part into power–down mode, it must read this bit first to determine if it is safe to do so. There is no need for the processor to read this bit prior to entering idle mode. The core is free to go into idle mode whenever it chooses. The CAN/CTL module will follow if and when it is ready. All of the logic required to implement everything discussed in this section will be in the CCB.

MEMORY INTERFACE UNIT

General Description

The XA-C3 memory interface (MIF) unit provides interfaces to generic memory devices such as SRAM, flash, and EPROM. The timing of memory cycles, including different strobe widths, is programmable by software.

MIF arbitrates between memory accesses from the XA core and from the DMA unit associated with the CAN/CTL function. It also provides access to the on–chip Memory Mapped Registers (MMRs) and the on–chip message buffer RAM (XRAM).

Summary of features

- Supports generic memory including SRAM, flash, and EPROM.
- Programmable timing.
- Supports wait states.
- Static 16-bit bus sizing.
- Arbitrates between CPU and DMA access.
- Relocatable Memory Mapped Register (MMR) access for CAN/CTL related configuration and data.

Memory Mapped Registers (MMRs)

The XA-C3 has several hundred bytes of memory mapped control/status registers (MMRs). These registers are mapped to the main data memory space. A 4KByte space is reserved from the data memory space for memory mapped registers (MMRs).

The base address of the MMR space is programmed by software. It can be placed anywhere within the entire 16 MByte data memory space supported by the XA architecture, other than at the very bottom of memory (address 000000h) where it would conflict with the on–chip DATA RAM (Scratch Pad). The 4K MMR space will always start at a 4K boundary.

The base address of the MMR space is determined by the contents of Special Function Registers MRBL and MRBH, as shown in Table 6 on page 11. Any address asserted by the XA whose twelve most significant bits match the concatenation MRBH[7:0] MRBL[7:4] will be automatically routed to the on–chip MMR bus.

The reset values for MRBH and MRBL are 0Fh and F0h respectively. Therefore, after a reset the MMR space is mapped to the uppermost 4K bytes of Data Segment 0Fh, but access to MMRs is *disabled*. The first 512 Bytes (offset 000h – 1FFh) of MMR Space are the Message Object Registers (eight per Message Object) for objects n = 0 – 31, as shown in Figure

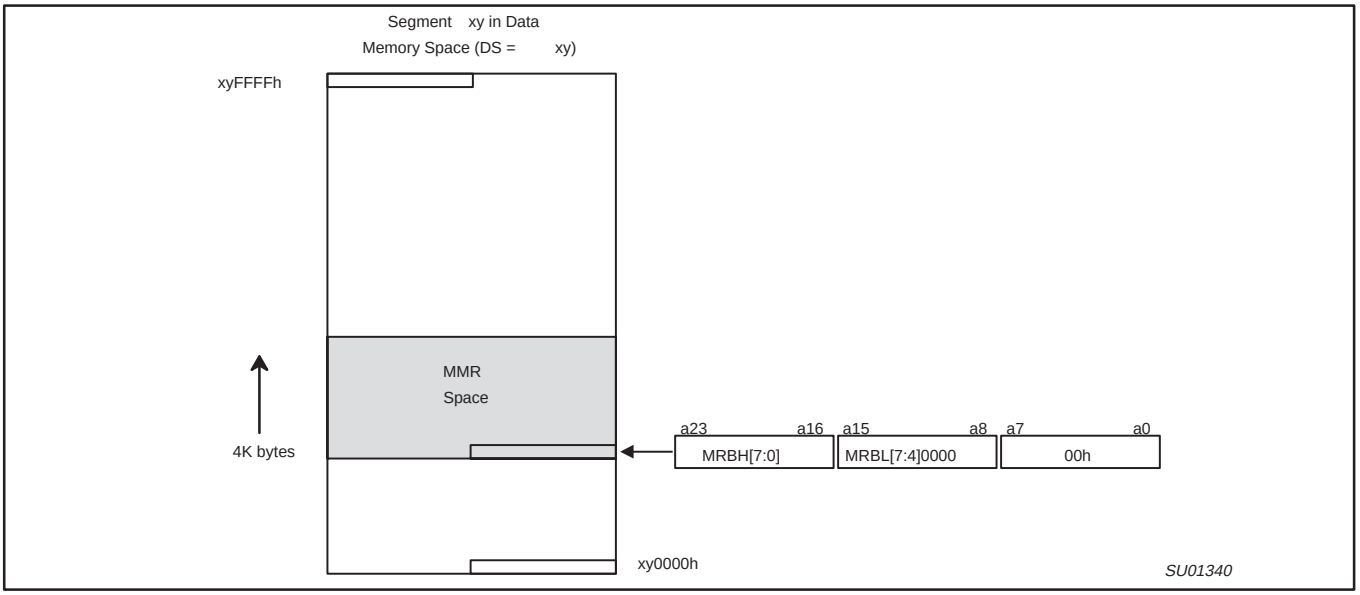


Figure 43. Formation of the MMR Base Address

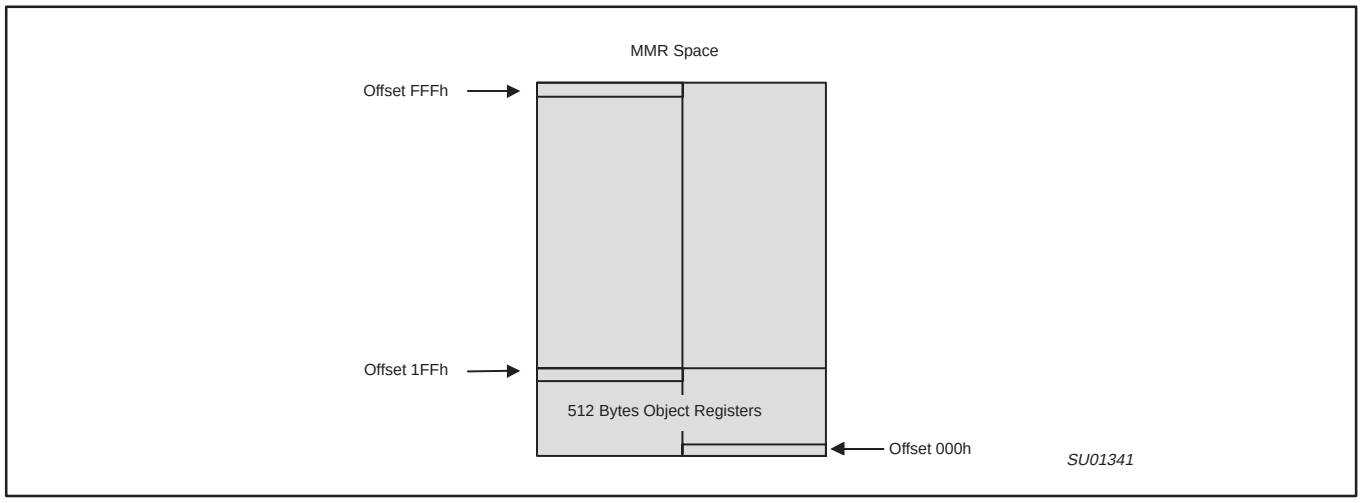


Figure 44. Detail of MMR space showing block of Message Object Registers

Special Function Register MRBH

- Address: SFR 497h
- Reset Value: 0Fh

MRBH

7	6	5	4	3	2	1	0
a23 – a16 of MMR Base Address							

Special Function Register MRBL

- Address: SFR 496h
- Reset Value: F0h

MRBL

7	6	5	4	3	2	1	0
a15 – a12 of MMR Base Address				–	–	–	MRBE

MRBE MRBE is the global enable bit for MMRs. On reset, MRBE is cleared to 0.

0 = MMRs disabled

1 = MMRs enabled

On–Chip Message Buffer RAM (XRAM)

The XA-C3 has a 512–byte on–chip message buffer RAM (XRAM) which may contain part or all of the CAN/CTL (transmit & receive objects) message buffers. This block of memory can be accessed as regular data memory. The logic address of the XRAM is programmed by software, and must start at a 512–Byte boundary.

The base address of the XRAM is determined by the contents of Memory Mapped Registers MBXSR and XRAMB as shown in and . Any address asserted by the XA core (or the DMA) whose fifteen most significant bits match the concatenation MBXSR[7:0]XRAMB[7:1] will be automatically routed to the XRAM. On reset, the XRAM is disabled. *Note: The XRAM should not be confused with the 1K Byte "scratch–pad" DATA RAM which is also provided on–chip.*

Since the uppermost 8 bits of all message buffer addresses are formed by the contents of the MBXSR register, the XRAM and all 32 message buffers must reside in the same 64K byte data memory segment. Since the XA-C3 only provides address lines A1 – A19 for accessing External memory, all External memory addresses must be within the lowest 1M byte of address space. Therefore, if there is External memory in the system into which *any* of the 32 message buffers will be mapped, then *all* 32 message buffers *and* the XRAM must also be mapped entirely into that same 64K byte segment, which must be below the 1M byte address limit.

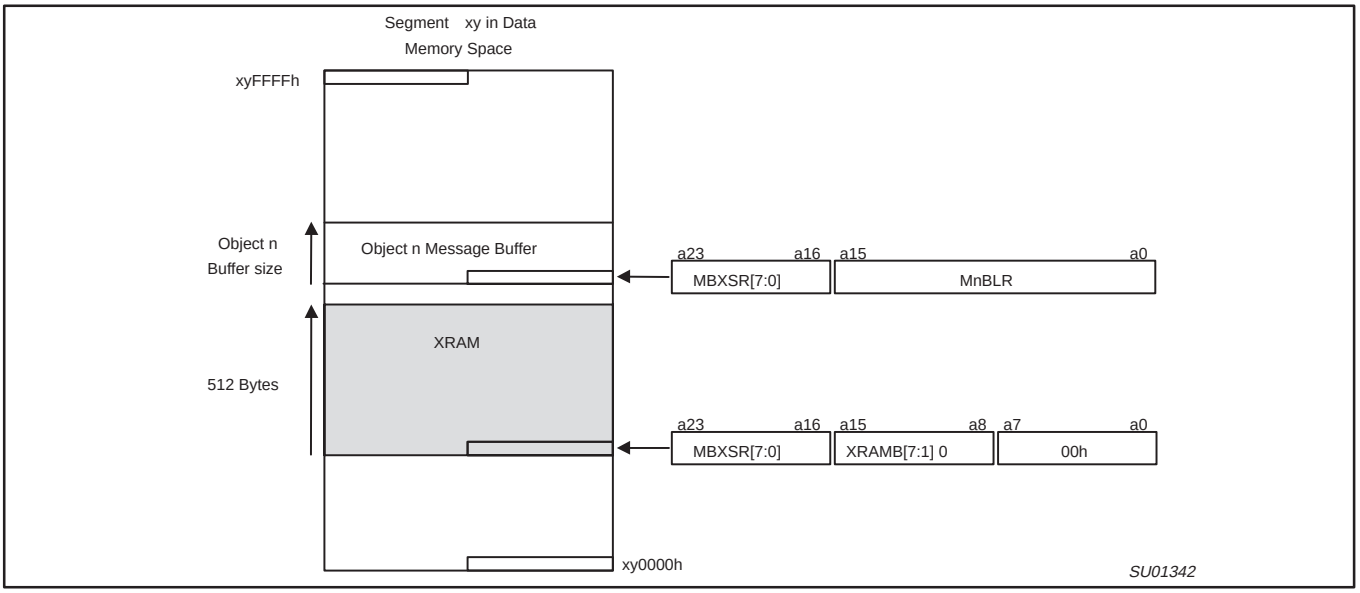


Figure 45. Formation of the XRAM base address, with object n message buffer mapped to off–chip data memory.

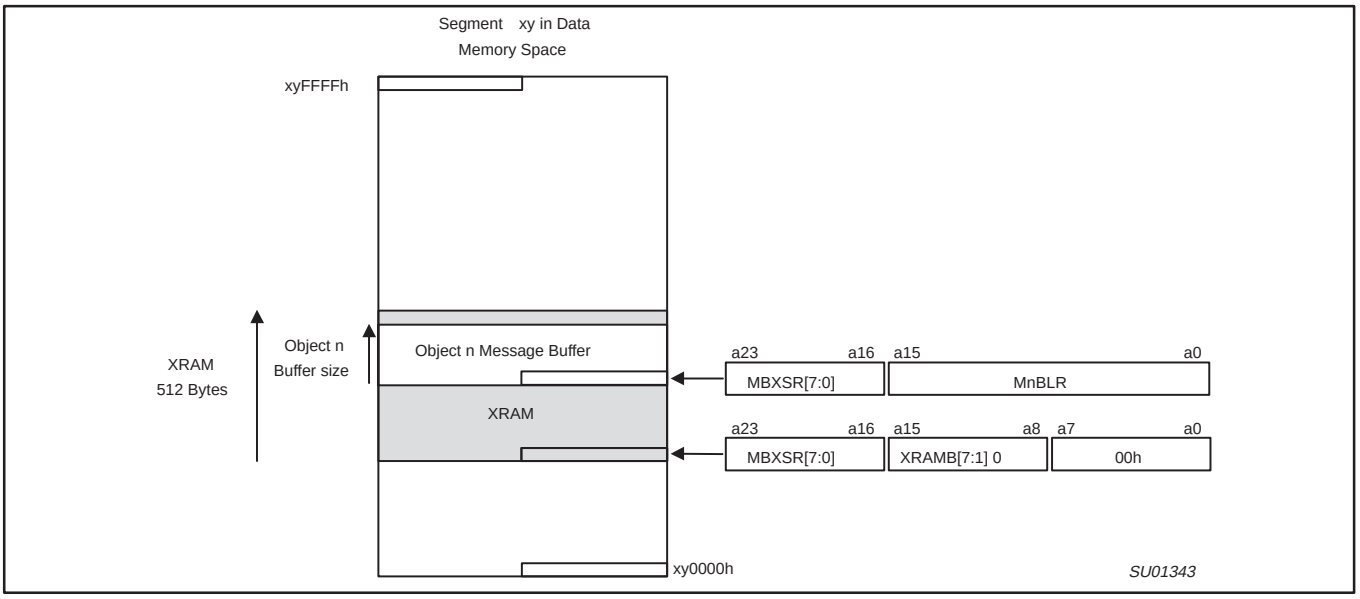


Figure 46. Object n Message Buffer mapped into the on–chip XRAM.

MBXSR (Message Buffer and XRAM Segment Register)

- Address: MMR Base + 291h
- Access: Read, write.
- Reset value: FFh

MBXSR

7	6	5	4	3	2	1	0
a23 – a16 of XRAM (and all message buffers) Base Address							

XRAMB (XRAM Base Address)

- Address: MMR Base + 290h
- Access: Read, write
- Reset value: FEh

XRAMB

7	6	5	4	3	2	1	0
a15 – a9 of XRAM Base Address							XRE

XRE — XRAM Enable bit, resets to '0'.
0 = XRAM disabled
1 = XRAM enabled

MIF Control and Configuration Registers

MIFCNTL (SFR)
- Address: SFR 495h

MIFCNTL

7	6	5	4	3	2	1	0
–	–	–	WAITD	BUSD	–	–	–

WAITD — Wait Disable
0 = Wail enabled
1 = Wait disabled

BUSD — External Access Disable
0 = enable
1 = disable

MIFBTRL (Memory Interface Bus Timing Register Low, MMR)
- Address: MMR base + 292h
- Access: Read, write, byte or word
- Reset value: EFh

MIFBTRL

7	6	5	4	3	2	1	0
WM1	WM0	ALEW	–	CR1	CR0	CRA1	CRA0

MIFBTRH (Memory Interface Bus Timing Register High, MMR)
- Address: MMR base + 294h
- Access: Read, write, byte or word
- Reset value: FFh

MIFBTRH

7	6	5	4	3	2	1	0
DW1	DW0	DWA1	DWA0	DR1	DR0	DRA1	DRA0

Note: The two MMRs MIFBTRL and MIFBTRH are not to be confused with the two SFRs BTRL and BTRH, which control the operation of the BIU, not the MIF. In order for the MIF to function properly, the contents of BTRL and BTRH have to be set at a fixed configuration on reset, by User application software, similar to the treatment for the XA-SCC MIF.

Bus Arbitration

Bus arbitration is done on an "alternate" policy. After a DMA bus access, the CPU will get the bus if requested. After a CPU bus access, the DMA will get the bus if requested. A burst access from the CPU cannot be interrupted by a DMA bus access.

SPI Port

The on–chip SPI Port uses the following Memory Mapped Registers:

SPICFG (MMR)
- Address: MMR base + 260h
- Access: Read, write, byte or word
- Reset value: 00h

SPICFG

7	6	5	4	3	2	1	0
SPCP	Rsvd	Rsvd	Rsvd	SPC3	SPC2	SPC1	SPC0

SPCP — SPICLK Polarity
0 = inverted SPICLK
1 = normal SPICLK

Rsvd — Reserved bits, only write zeros.

SPC3 – SPC0 — SPICLK timing

SPICLK = (CClk) / 4 (SPICFG[3:0] + 1)

SPIDATA (MMR)
- Address: MMR base + 262h
- Access: Read, write, byte or word
- Reset value: 00h

SPIDATA

7	6	5	4	3	2	1	0
Data							

SPICS (MMR)
- Address: MMR base + 263h
- Access: Read, write, byte or word
- Reset value: 00h

SPICFG

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

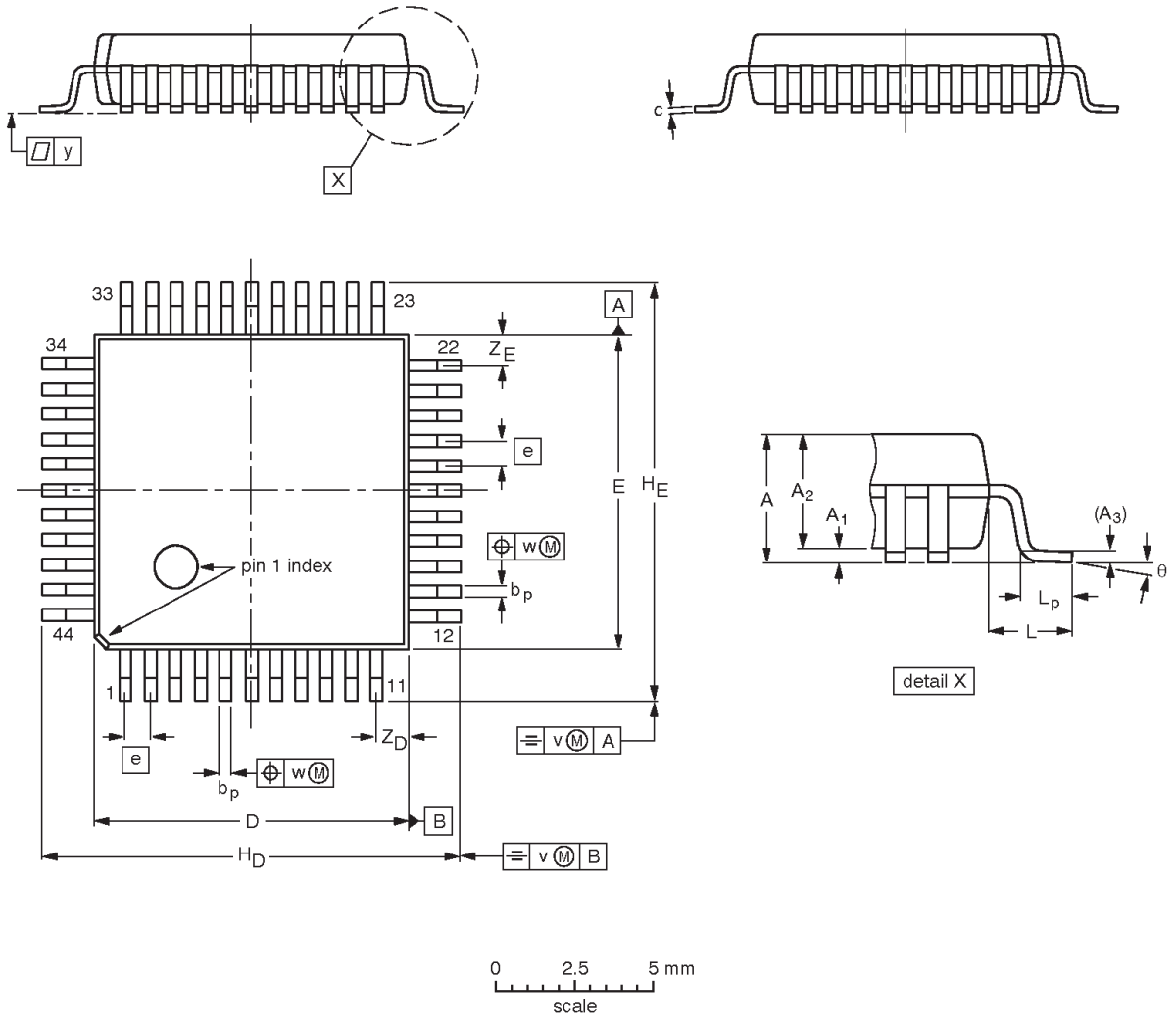
SPSTT — SPI Start
0 = Cycle finished, cleared by hardware and on reset
1 = Start

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

LQFP44: plastic low profile quad flat package; 44 leads; body 10 x 10 x 1.4 mm SOT389-1



DIMENSIONS (mm are the original dimensions)

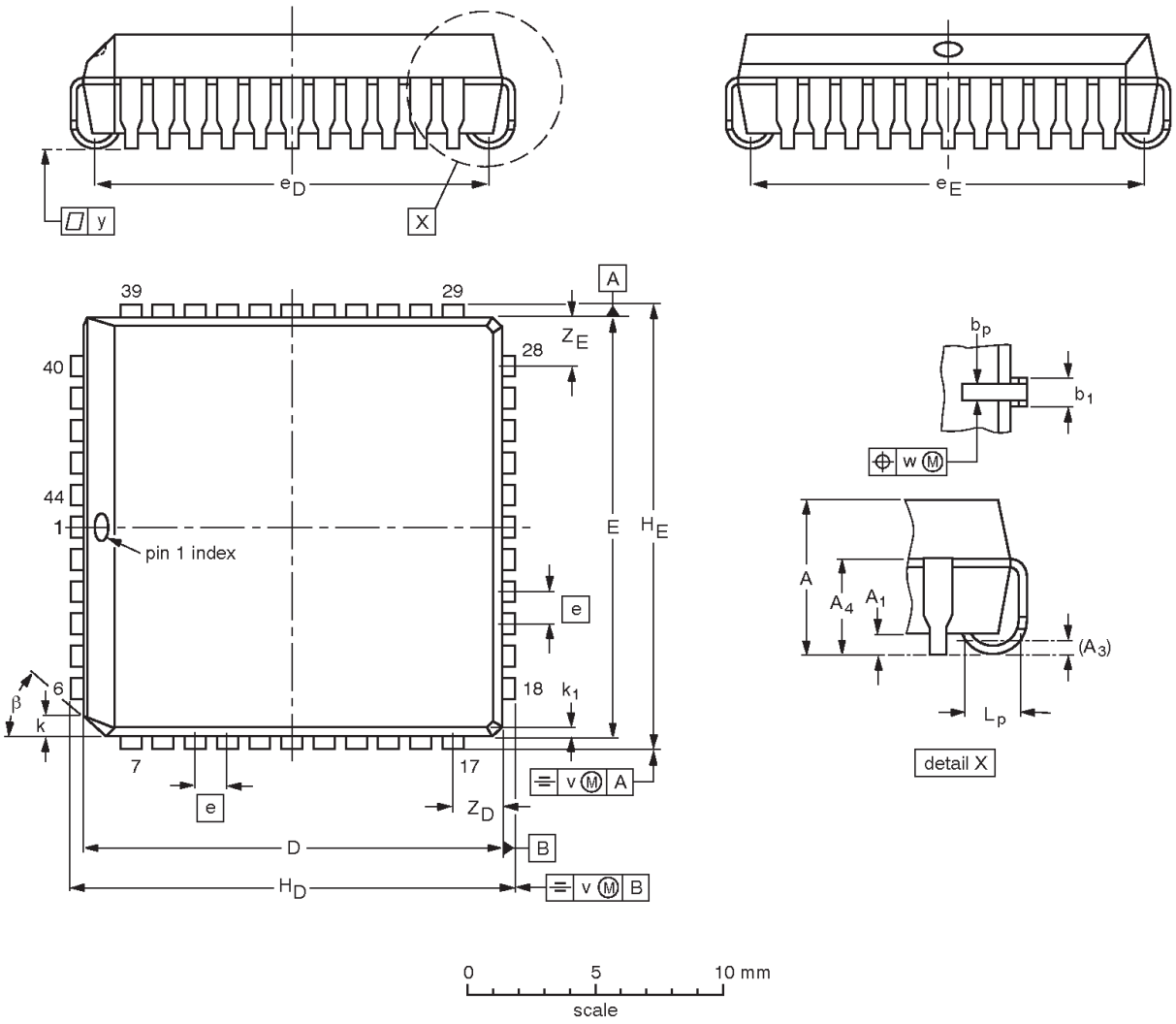
UNIT	A max.	A_1	A_2	A_3	b_p	c	$D^{(1)}$	$E^{(1)}$	e	H_D	H_E	L	L_p	v	w	y	$Z_D^{(1)}$	$Z_E^{(1)}$	θ
mm	1.60	0.15 0.05	1.45 1.35	0.25	0.45 0.30	0.20 0.12	10.10 9.90	10.10 9.90	0.80	12.15 11.85	12.15 11.85	1.0	0.75 0.45	0.20	0.20	0.10	1.14 0.85	1.14 0.85	7^o 0^o

Note

1. Plastic or metal protrusions of 0.25 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	EIAJ			
SOT389-1						~~95-12-19~~ 97-08-04

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

Data sheet status

Data sheet status	Product status	Definition [1]
Objective specification	Development	This data sheet contains the design target or goal specifications for product development. Specification may change in any manner without notice.
Preliminary specification	Qualification	This data sheet contains preliminary data, and supplementary data will be published at a later date. Philips Semiconductors reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.
Product specification	Production	This data sheet contains final specifications. Philips Semiconductors reserves the right to make changes at any time without notice in order to improve design and supply the best possible product.

[1] Please consult the most recently issued datasheet before initiating or completing a design.

Definitions

Short-form specification — The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information see the relevant data sheet or data handbook.

Limiting values definition — Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Disclaimers

Life support — These products are not designed for use in life support appliances, devices or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Philips Semiconductors
811 East Arques Avenue
P.O. Box 3409
Sunnyvale, California 94088–3409
Telephone 800-234-7381

Date of release: 01-00

Document order number: 9397 750 06805

Let's make things better.

Philips Semiconductors

PHILIPS