APPLICATION NOTE

Interfacing 68000 family peripherals to the XA

AN96098

Philips
Semiconductors

PHILIPS

Abstract

The XA is not limited to interface to 80XX compatible peripherals. By using some glue logic, or a PLD if available, the XA can be interfaced to a 68000 compatible peripheral. Using the 68000 DTACKn signal enables the use of slow 68000 peripherals without the need of an external WAIT state generator.

Purchase of Philips I^2C components conveys a license under the I^2C patent to use the com-ponents in the I^2C system, provided the system conforms to the I^2C specifications defined by Philips.

APPLICATION NOTE

Interfacing 68000 family peripherals to the XA

AN96098

Author(s):

Marco Kuystermans

System Laboratory Eindhoven,
The Netherlands

Keywords

68k XA PCF8584 WAIT

Date: 1996-10-25

Summary

Many peripherals with a 68000 compatible interface are on the market. This document shows that these 68000 peripherals can be interfaced to the XA very easily. This means that an XA user is not limited to an XA compatible peripheral in his XA design.

Several interfacing aspects are covered: normal interfacing, interrupt interfacing and bus arbitration.

This document assumes that users are familiar with the XA and its bus interface.

CONTENTS

1. HOW TO INTERFACE 68000 FAMILY PERIPHERALS TO THE XA

1.1 introduction

The Philips Semiconductors XA is a 16 bit high speed microcontroller with an 80XX compatible bus (found on e.g. an 8051 or 8048 with separate read and write strobes). This means that popular periherals with an 80XX compatible bus can be used with the XA.

This documents shows that besides 80XX peripherals also peripherals with a 68000 bus can be used. Extra advantage is that the 68000's DTACKn output is extremely suitable to generate an XA WAIT strobe. The main advantage of using this DTACKn output is that a full handshake is available (synchronous) and consequently NO external wait state generator is needed.

1.1.1 General remarks:

- Because the XA wait mechanisme is rather complex, this document is NOT intended for the starting XA user, instead some fundemental knowledge about the XA is needed.
- In this documed is assumed that the nessecary address latches are provided (2 x 74HCT573), i.e. a demultiplexed bus is used:

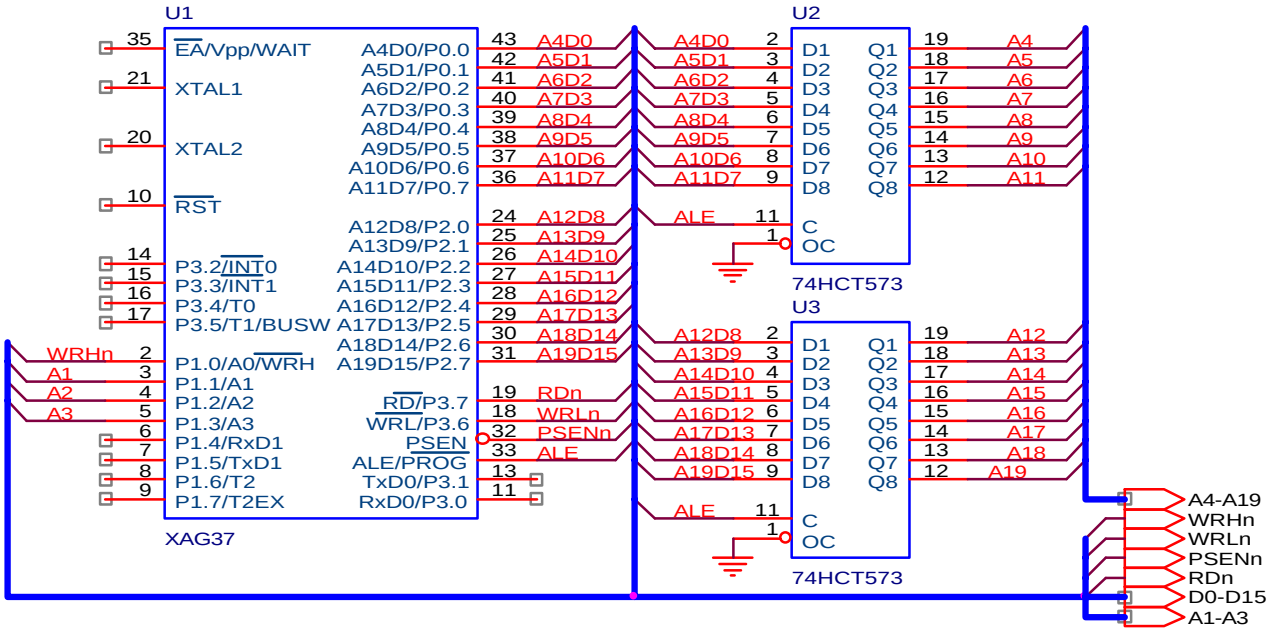

Figure 1, demultiplexed XA bus

1.2 Differences between an 80XX and 68000 bus

A 68000 bus compatible peripheral has a significantly different interface than a 80XX bus type peripheral. 68000 and 80XX bus interfaces consists of several different control signals (see Table 1):

Table 1, Microcontroller control signals

	68k compatible peripheral	*80XX compatible peripheral*	*XA compatible peripheral*
Data strobes	2 + direction: R/Wn *(UDSn + LDSn)	2: WRn + RDn	3: WRLn + WRHn + RDn
Chip select	CSn	CSn	CSn
Hand shake	DTACKn	Not available	~WAIT
Code strobe	Not available	PSENn	PSENn

Both 68000 and 80XX peripherals use data strobes to read or write data. An 80XX peripheral has separate read and write strobes (see Figure 3), a 68000 peripheral uses a data strobe (only indicating a data transfer, not the direction, see Figure 2) AND a R/Wn signal to indicate a READ (R/Wn = 1) or a WRITE (R/Wn = 0) cycle. At both the 68000 and 80XX data strobes' rising edges data will (write) or must (read) be valid.

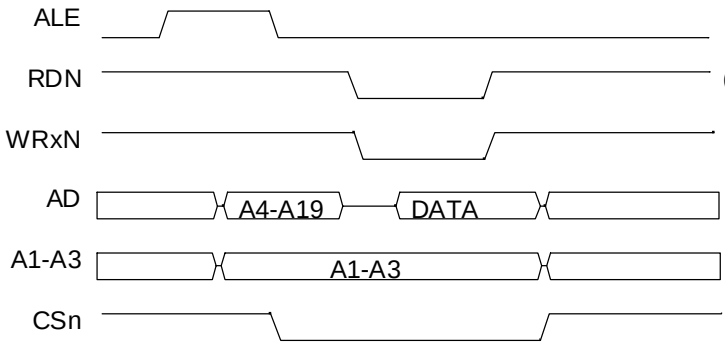

Figure 3, 80XX bus

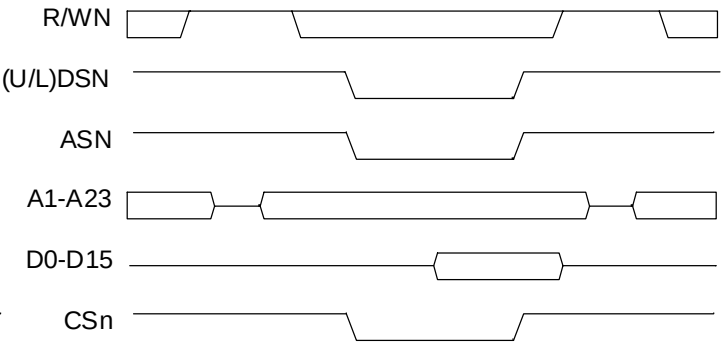

Figure 2, 68000 bus

Normally an 80XX chipselect (not) signal is generated by decoding addresses after the address latches. To avoid spikes on this signal (addresses can change during ALE = 1) the CSn signal should be gated with ALE. A 68000 chipselect can be constructed by decoding addresses and gating it with the Address Strobe (ASn) signal generated by the 68000.

Some 8 bit wide 68000 compatible peripherals do not have separate Data Strobe and Chip Select inputs. On these peripherals the Data strobe and Chip Select are multiplexed to one signal. In this document a multiplexed DSn (data strobe) and CSn (chip select strobe). will be called CSn.

Real 16 bit wide 68000 peripherals have two Data strobes; UDSn (upper data strobe) and LDSn (lower data strobe) and consequently need a Chip Select (not) input.

Both the XA and 68000 are 16 bit microcontrollers. Therefore address line 0 has no meaning and is on the 68000 decoded to UDSn (A0 = 0, little endian!) and LDSn (A0 = 1). On the XA A0 is decoded to WRLn (A0 = 0, big endian!) and WRHn (A0 = 1). On the XA no separate read low and high are available. All read cycles (except if the XA bus is configured as 8bit) are 16 bit, if only 8bits are needed the upper or lower 8 bits are discarded.

The following pictures show how to convert XA strobes to 8 bit or 16 bit 68000 peripherals.

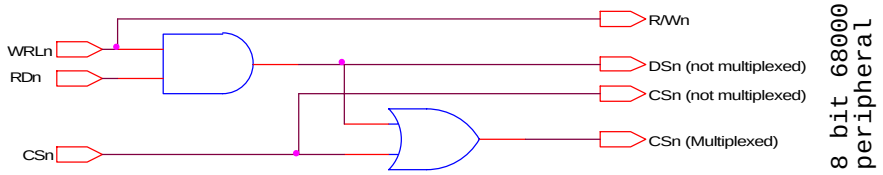

Figure 4, converting XA to 68000 strobes (8 bit)

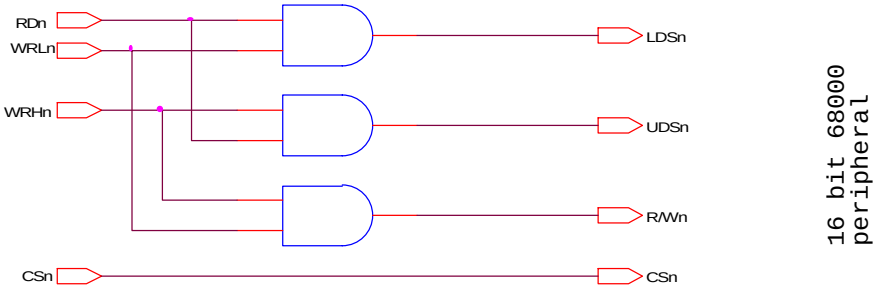

Figure 5, converting XA to 68000 strobes (16 bit)

Please be aware that a 68000 peripheral has no separate data and program memory area, instead a lineair memory space is available with mixed memory. It is up to the user to decide in which XA memory area the 68000 peripheral is located.

Because on the XA it is not possible to write to program memory, writing is always performed via the XA data write strobe (WRHn and/or WRLn). Writing is achieved through the XA's data and extra segment (DS & ES, see XA user's guide [1]). Reads can be executed in both data (RDn strobe via ES & DS) and program memory (PSENn via Program counter or Code segment).

In Figure 4 and Figure 5 RDn can be replaced by PSENn. If PSENn is used to read data from a 68000 peripheral, consequently MOVC must be used.

1.3 Using DTACKn to Generate an XA compatible WAIT signal

1.3.1 Generating 68000 bus signals

Figure 6 shows the signals from both the XA and a 68000 peripheral combined. First of all a 68000 compatible data strobe needs to be constructed composed from XA signals. The easiest way to accomplish this is to combine the RDn or WRLn/WRHn strobes with a chip select (decoded addresses, or the most simple solution only one address line). Figure 6 shows the generated 68000 CSn strobe (or LDSn in case of real 16 bit wide 68000 peripherals because in this example WRLn is used). t_2 in Figure 6 indicates the propagation delay of the decoding circuit.

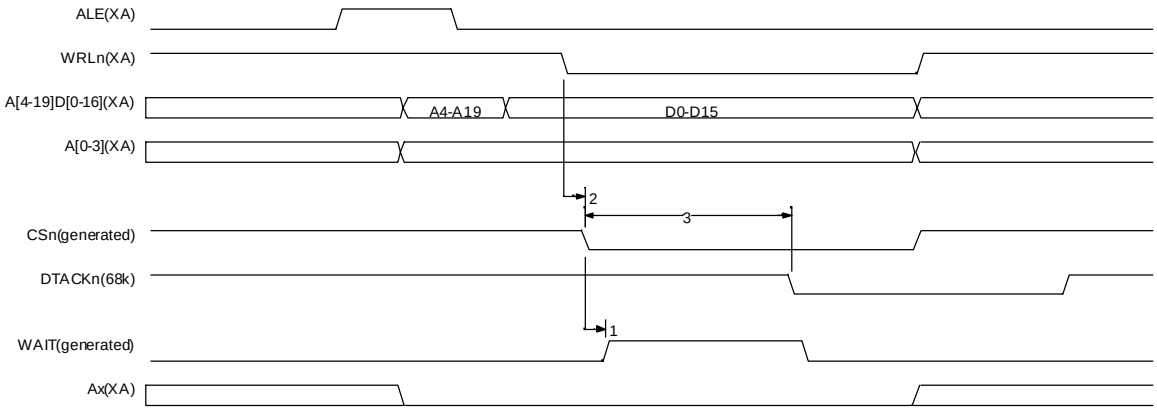

Figure 6, 68000 and XA signals combined

The XA's WRLn/WRHn signal can be used to generated the 68000's R/Wn signal. In case of a 16 bit wide 68000 peripheral BOTH WRLn and WRHn need to be monitored to generate the R/Wn signal. 68000 peripherals with an 8 bit wide data bus can be connected to either D8:15 (using only WRHn) or to D0:7 (using only WRLn).

Please note that some 68000 peripherals do not allow t_2 (R/Wn set-up to CS low) to be 0ns, for example the PCF8584 needs t_2 to be 10ns or higher. You need to use an address line to generate a R/Wn signal (see figure 1, Ax(XA)), if the decoding logic propagation delay is shorter than the required time t_2. A drawback to this solution is reading and writing is not possible on the same address.

1.3.2 68000 DTACKn signal.

A 68000 peripheral generates a DTACKn to provide a real handshake between the microcontroller and its peripheral. DTACKn indicates when the peripheral is ready to receive data (in case of a write cycle), or when the microcontroller can expect valid data on the databus, placed on the bus by a peripheral. This DTACKn can be used to generate an XA compatible WAIT signal

1.3.3 Generating the XA WAIT signal.

An XA WAIT signal must be asserted immediately after (minimal 34ns before end of strobe, i.e. the rising edge) a read (PSENn or RDn) or write (WRLn or WRHn) strobe is asserted. The XA will ONLY insert waitstates after the XA databus is stable up to the point the WAIT signal is de-asserted. A WAIT

signal has no effect if it is asserted outside a data strobe. It is however allowed to generate a WAIT signal before a data strobe is asserted, but will only be active after this strobe is asserted.

After a CSn (LDSn) strobe has been generated the device's DTACKn signal is high until the device is ready to receive. During this high period the XA must generate wait cycles, see t_3 in Figure 6. So the equation for the WAIT signal (t_3) is: /CSn * DTACKn.

There are several ways to construct a XA WAIT signal from a DTACKn signal, discrete or via a PLA. The following schematic (Figure 7) shows *a* discrete implementation for 8 bit wide peripherals.

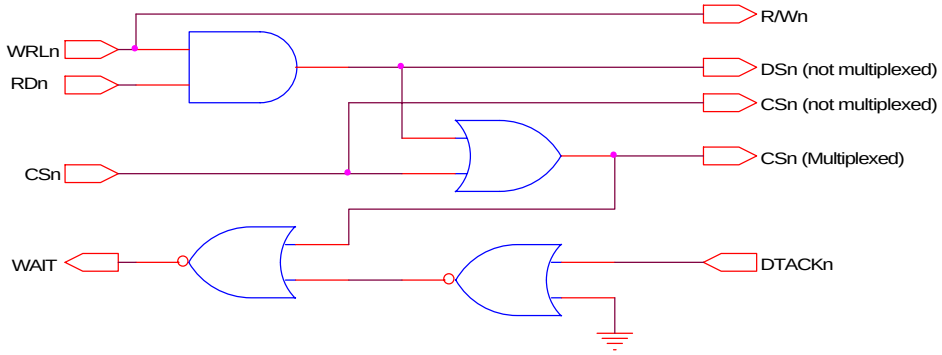

Figure 7, discrete solution (8 bit wide 68000 peripheral)

or if LDSn and UDSn are needed (16 bit peripheral):

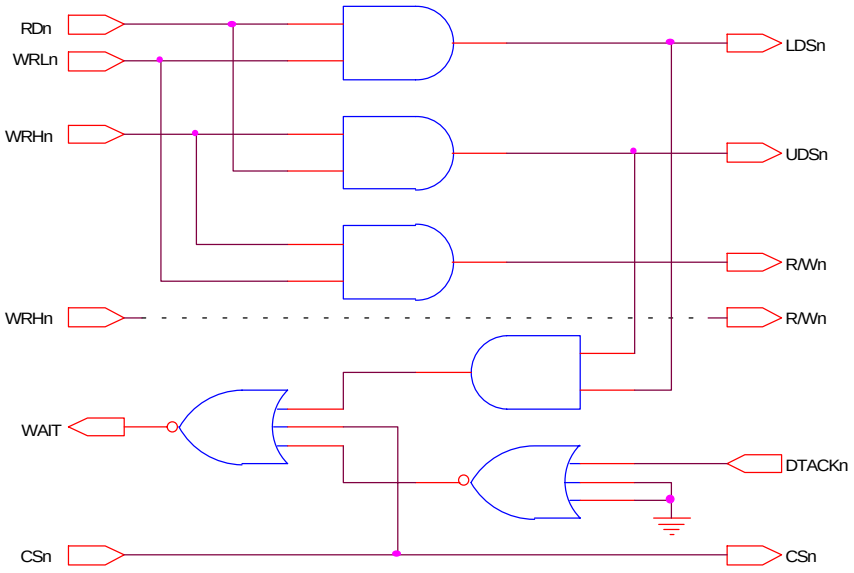

Figure 8, Discrete solution 16 bit peripheral

Table 2, Figure 8 WAIT truth table

U/LDSn	CSn	DTACKn	WAIT
0	0	*0*	*0*
0	0	*1*	*1*
0	1	*0*	*0*
0	1	*1*	*0*
1	0	*0*	*0*
1	0	*1*	*0*
1	1	*0*	*0*
1	1	*1*	*0*

Table for both UDSn and LDSn

Table 3, Figure 8 data strobe thruth table

WRLn	WRHn	RDn	LDSn	UDSn	R/Wn
0	0	0	*x*	*x*	*x*
0	0	1	*0*	*0*	*0*
0	1	0	*x*	*x*	*x*
0	1	1	*0*	*1*	*0*
1	0	0	*x*	*x*	*x*
1	0	1	*1*	*0*	*0*
1	1	0	*0*	*0*	*1*
1	1	1	*1*	*1*	*1*

x = NOT VALID

You can also use a PLA if one is present (see Figure 9):

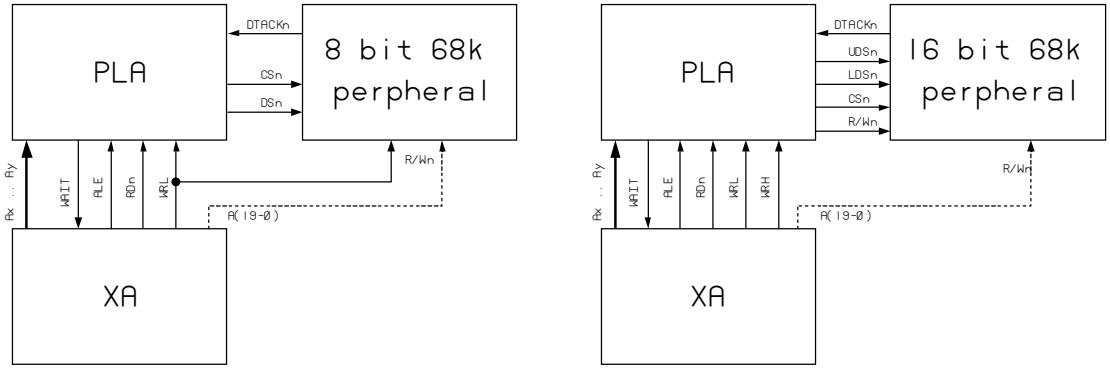

Figure 9, PLA plus XA plus 68000

The PLA must have the following equations (multiplexed DSn and CSn, e.g. SC68C562):

(1a) CSn = /((Ax *..* Ay) * /(WRLn * RDn) * /ALE)

(1b) WAIT = ... + /CSn * DTACKn

(1c) R/Wn = WRLn (R/Wn directly connected to WRLn)

The dots represent other funtions to drive the WAIT pin (see general remarks 1.3.4)

Non multiplexed 8 bit 68k peripheral (e.g. ST 68HC901):

(2a) CSn = /((Ax *..* Ay) * /ALE)

(2b) DSn = /(WRLn * RDn)

(2c) WAIT = ... + /CSn * /DSn * DTACKn

(2d) R/Wn = WRLn (R/Wn directly connected to WRLn)

In case of 16 bit wide 68000 peripherals, where separate UDSn and LDSn strobes are needed (e.g. SCC66470 video and system controller);

(3a) LDSn = /(WRLn * RDn)

(3b) UDSn = /(WRHn * RDn)

(3c) R/Wn = WRHn * WRLn

(3d) CSn = /((Ax *..* Ay) * /ALE)

(3e) WAIT = ... + /(LDSn * UDSn) * DTACKn * /CSn

The (generated) R/Wn, LDSn and UDSn signals are available for ALL 68000 peripherals. The DTACKn signal is generated by a 68000 peripheral. All 68000 DTACKn pins are open drain outputs and connected together as wired OR. The only device specific signal is CSn and therefore ALL CSn signals need to be monitored if a corresponding WAIT signal needs to be generated for that particular device:

(4a) CS1n = /((Ax *..* Ay) * /ALE)

(4b) CS2n = /((Ax *..* Ay) * /ALE)

(4c) WAIT = ... + /(LDSn * UDSn) * DTACKn * (/CS1n + /CS2n)

1.3.4 General remarks:

- The XA WAIT pin is combined with the EAn (external access) function. This pin is sampled at the rising edge of RESET, therefore in functions 1b, 2e and 3c an extra EAn term needs to be added. An example, however beyond the scope of this document, can be: EAn = ALE * WRLn * RDN * EAmode. During RESET ALL XA pins are high, this effect is used to generate EAn or not (depending on EAmode, e.g. a jumper). Of course WAIT strobes will be generated during ALE = 1, but as stated in this paragraph WAIT strobes outside a data strobe will not put the XA in WAIT mode.
- ALL chip select lines are decoded address lines ANDed with /ALE, this prevents generating spikes on the CSn lines while addresses are not stable (ALE = 1).
- Please be sure the WAIT pin is not overridden by the WAIT disable bit.

1.4 68000 Interrupt mechanism

When using a 68000 peripheral, consequently the 68000 interrupt mechanism is used. This means that the XA has to generate an IACKn to this peripheral. It is not allowed to assert the CSn and IACKn strobes at the same time, the IACKn is (so to say) also a Chip Select.

A way of generating a IACKn is decoding an interrupt vector address and combining it with a read strobe (more or less an alternative Chip Select, compare with 1a), e.g. with a PLA:

```
(5) IACKn = /((Ax * .. * Ay) * /RDn * /ALE)
```

Ax to Ay must be an other address now than the address in the previous paragraph, so CSn address for normal data accesses IACKn address

So to generate an IACKn (example via Extra Segment = ES):

```
(6a)    MOV.b       ES, #xxh            xx = (addresses A19 - A16)
(6b)    MOV.b       R2, #yyyyh          yyyy = (addresses A15 - A0)
(6c)    OR.b        SSEL, #04h          aligns ES to R2
(6d)    MOV.b       R3, [R2]            R3 holds peripheral`s vector
(6e)    JMP         [R3]
```

Generating an IACKn can also be achieved via a code read if the full code range is not used:

```
(7)   IACKn = /((Ax * .. * Ay) * /PSENn * /ALE)
```

Using PSENn to generate IACKn enables the use of the same address as the CSn address (with RDn), it is now a code space address. An interrupt acknowledge can have the following construction (example via Code Segment = CS):

```
(8a)    MOV.b       CS, #xxh            xx = (addresses A19 - A16)
(8b)    MOV.b       R2, #yyyyh          yyyy = (addresses A15 - A0)
(8c)    OR.b        SSEL, #04h          aligns CS to R2
(8d)    MOVC.b      R3, [R2]            R3 holds peripheral`s vector
(8e)    JMP         [R3]
```

Notes:

- If XA code is running internal exclusively (within the on board EPROM) and only one peripheral needs an IACKn, NO address decoding is needed and PSENn can be connected to IACKn directly.
- Please be sure xx:yyyy in formula 8a and 8b is above the XA internal memory range, in case of the P51XAG37 xx:yyyy > 0x00:7FFF

1.5 68000 bus arbitration with the XA

Missing on the XA is a bus arbitration scheme. If the XA had an ONCE mode, i.e. a pin that forces all pins to float, it would have been very easy to implement. It is however possible to construct 68000 bus arbitration with discrete components. This XA bus arbitration scheme will also utilise the XA WAIT pin. During a bus arbitration cycle a WAIT signal will halt the XA.

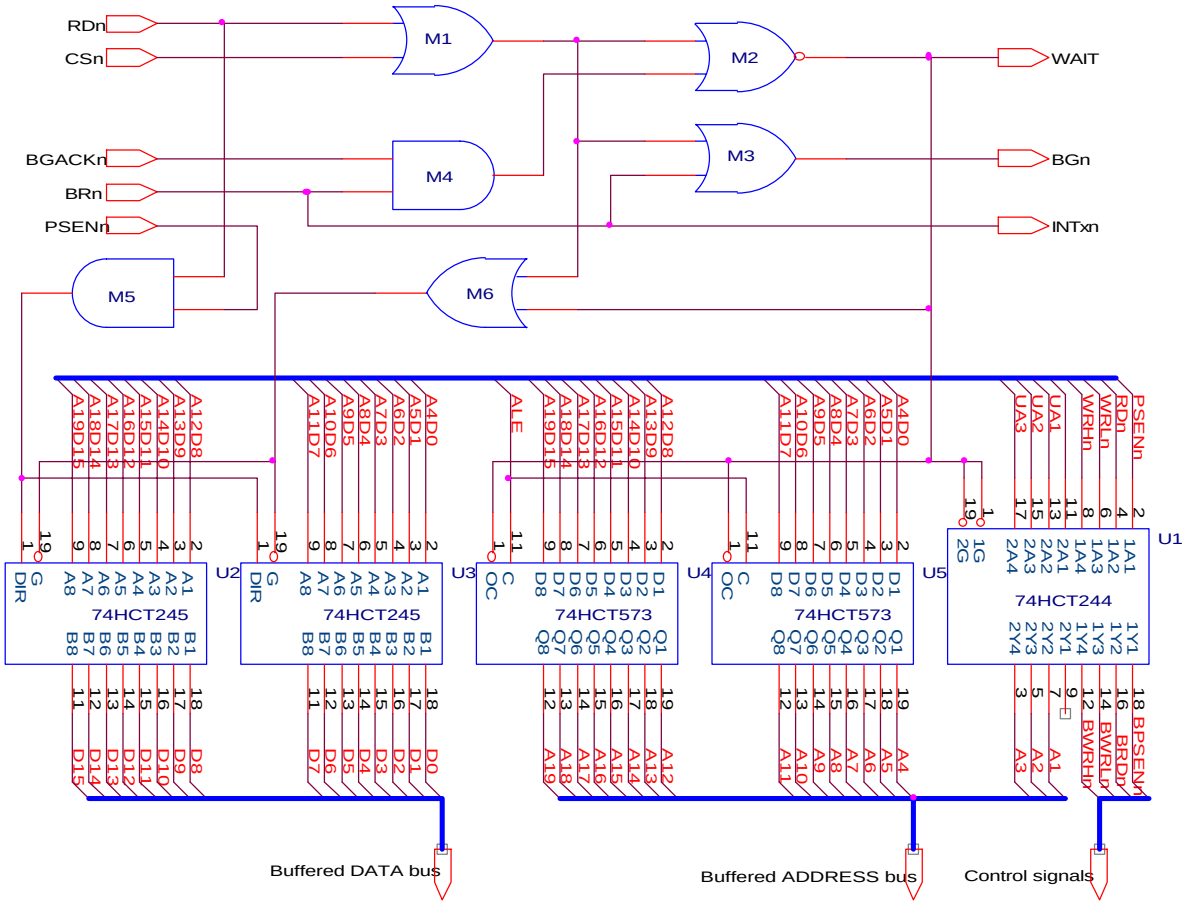

Figure 10, Bus arbitration on XA

A device that requests the bus asserts the BRn (bus request) strobe. The master device must return a BGn (Bus Grand) strobe when the master is ready. After the slave has received the BGn signal it starts the bus arbitration cycle by asserting BGACKn. During this strobe the slave device must have complete access to the (shared) memory, therefore the XA is on hold and its signals must float. As stated NO ONCE pin is available therefore the XA memory bus must be buffered.

Due to the XA multiplexed address/databus structure latches are needed to demultiplex. The latches have an Output Enable (OEn) input, connecting this input to WAIT will float the address bus when the XA is in WAIT.

The only thing missing is the non-floating databus, the not multiplexed address lines A3 to A1 and the control signals RDn, WRLn, WRHn and PSENn. The databus can be made floating by using a bi-

directional buffer. Writing or reading is indicated by the direction (DIR) signal. If DIR=1 a write is indicated else a read. A DIR signal is constructed by ANDing both write strobes (WRLn and WRHn).

A3 to A1 and the control signals only need an output buffer (with Output Enable control). Figure 10 shows a solution with discrete components.

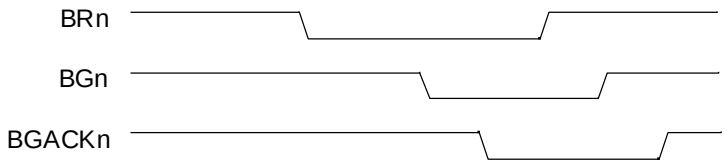

Figure 11, Bus arbitration timing diagram

The XA must generate a BGn signal. This signal is constructed by monitoring the RDn, WRLn, WRHn and PSENn strobes. If one of these strobes is asserted AND a BRn is pending a BGn is generated up to the point the slave device is negating the BRn.

Figure 11 shows that in principle BGn must be asserted longer than BRn. In the 68000 specification BGACKn asserted to BGn negated is min. 1.5 and max. 3.5 clocks. BGACKn asserted to BRn negated min. 20ns max. 1.5 clocks. This shows that BRn and BGn can be negated at the same time. Figure 10 shows that in fact BGn is negated after BRn has been negated and some propagation delay should be considered.

The WAIT signal is generated during BRn or BGACKn asserted, but only if one of the XA control strobes is asserted AND the XA reads at the bus “request enable address”. This bus request enable address is a decoded dedicated address. Just using the RDn without decoding address can cause the following problem: If during a “normal” XA read a Bus Request occurs (for the bus request signal is generated aynchronously), WAIT can be generated to late for the XA to sample. The bus however will float when both BRn and RDn strobe are generated. If during this situation the XA is not in WAIT mode, unpredictable things can occur. BRn is connected to one of the two XA external interrupts inputs (INT0n or INT1n) to be sure the XA will access external memory when a bus request is pending. The bus arbitration cycle will only continue after the XA has accessed external memory.

The glue logic (Figure 10) can be replaced by a PLA and combined with other functions. The bus arbitration WAIT signal is constructed as follows:

```
(9)   WAIT = .. + /(RDn * WRLn * WRHn * PSEN) * (Ax * .. * Ay) * /(BRn * BGACKn)
```

The Bus Grand (BGn) is assembled as follows (output to slave);

```
(10)  BGn = /(( RDn * WRLn * WRHn * PSEN) * (Ax * .. * Ay) + BRn)
```

The following instructions must be part of the interrupt routine to generate a BGn and put the XA in wait:

```
(11a)   MOV.b     ES, #xxh          xx = (addresses A19 - A16)
(11b)   MOV.b     R2, #yyyyh        yyyy = (addresses A15 - A0)
(11c)   OR.b      SSEL, #04h        aligns ES to R2
(11d)   MOV.b     R3, [R2]          read to generate BGn
(11e)   RETI                        return from interrupt
```

The XA interrupt latency determines the time it takes for the XA to generate BGn after BRn asserted.

2. EXAMPLE, INTERFACING THE PCF8584 TO THE XA

Interfacing the PCF8584 (See IC12 [2]) to microcontrollers can cause serious problems. Causes for these problems are: The Interface Mode Control, the interface the PCF8584 will run in is determined by the first write cycle to this peripheral. Secondly it is a relatively (to the XA) SLOW device. In contrast this example shows that it is in fact quite easy to interface the PCF8584 to the XA.

2.1.1 PCF8584 to XA implementation

In “80C51” mode the PCF8584 needs write and read strobes longer than 250nS, the XA can generate read strobes with a maximum length of 4 times the oscillator clock period, a XA write strobe is even shorter, its length is max. 2 clock periods. This means when the XA is running at 8MHz the write strobe will match the PCF8584 write strobe specification. Running the XA on higher frequencies than 8MHz will cause the need of an external wait state generator.

It is however possible to use the PCF8584 WITHOUT the use of an external wait state generator on every XA clock (up to the maximum XA clock). Using the PCF8584 in 68000 mode will do the trick. Using the PCF8584 in 68000 mode:

1. R/Wn asserted before CSn (t_2) needs to be 10ns or higher. This time is needed to enable the PCF8584 to decode a 80C51 or 68k bus mode (it is by the way the only 68k peripheral that does **not** allow t_2 to be 0ns). If the decoding logic is too fast an address is needed as R/Wn strobe instead of WRLn or WRHn.
2. The first PCF8584 cycle needs to be a write cycle, this cycle determines the bus mode the PCF8584. In contrast of getting the PCF8584 in 80C51 mode, it is allowed to have write or read cycles to other peripherals first. Having a read cycle from the XA first will put the PCF8584 in 80C51 mode, thus not asserting the DTACKn signal (it is now the RDn input) and therefore causing the XA to be in a endless WAIT.
3. When an address line is used to generate a R/Wn signal, reading and writing is not performed on the same address.
4. Excessive accesses to the PCF8584 will slow down the entire application, so using the PCF8584 in time critical applications in polling mode should be avoided.

The following picture shows an actual hardcopy of all signals used to connect the XA to a PCF8584. Please note that the DTACKn rising edge has the shape of an exponential function:

$$U = \overline{U} \bullet \left(1 - e^{-\frac{t}{RC}}\right)$$

this is caused by the open drain (wired OR) structure of this pin (pull up resistor R and parasitic capacitance C).

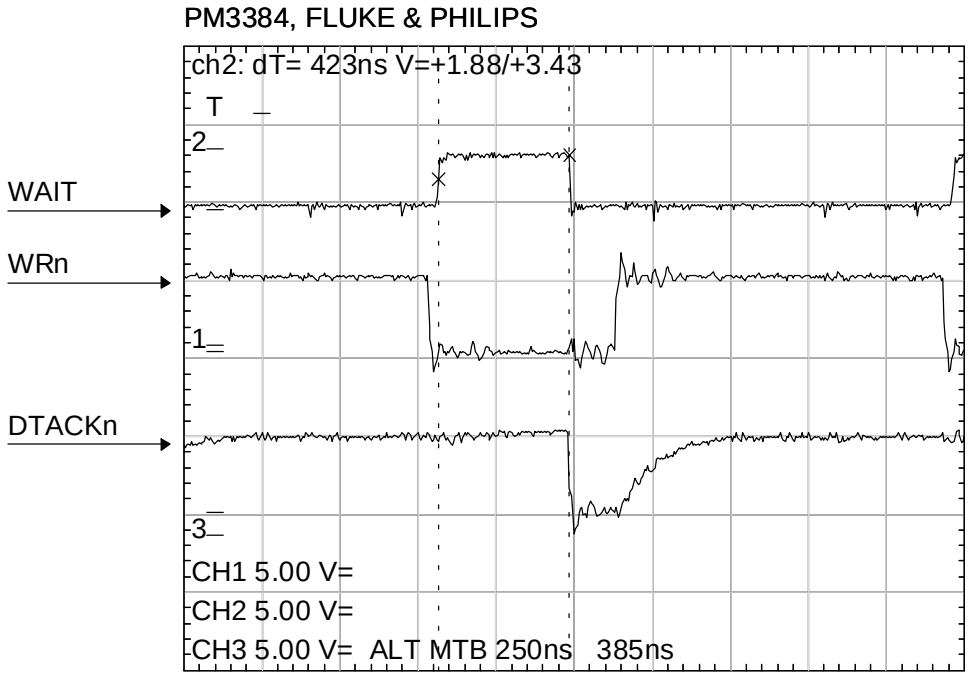

Figure 12, Scope hardcopy

2.1.2 Usage.

Please be sure the first cycle to the PCF8584 is a write cycle.

If using an address line as R/Wn strobe, writing to the PCF8584 must be performed on an other address than reading from the PCF8584 (e.g. if A15 is used):

Reading PCF8584 address 0:

```
(12a)   MOV.b     ES, #0Fh
(12b)   MOV.b     R2, #8000h          address line A15 = 1
(12c)   OR.b      SSEL, #04h          aligns ES to R2
(12d)   MOV.b     R3L, [R2]           Result is stored in R3L
```

Writing to PCF8584 address 0:

```
(13a)   MOV.b     ES, #0Fh
(13b)   MOV.b     R2, #0000h          address line A15 = 0
(13c)   OR.b      SSEL, #04h          aligns ES to R2
(13d)   MOV.b     [R2], #data8        data8 written to PCF8584
```

APPENDIX 1 REFERENCES

[2]	16-bit 80C51XA Microcontrollers -	Data Handbook IC25	PHILIPS: 9397 750 00733
[3]	I2C Peripherals	Data Handbook IC12	PHILIPS: 9397 750 00306
[4]	80C51 based 8-bit Microcontrollers	Data Handbook IC20	PHILIPS: 9397 750 00013
[5]	The 68000 microprocessor	Michalel A. Miller	ISBN: 0675 205220 02