APPLICATION NOTE

Using the XA EAn/WAIT pin

AN96075

Let's make things better.

Philips Semiconductors

PHILIPS

Using the XA EAn/WAIT pin

Application Note
AN96075

Abstract

The XA is the high speed 16 bit successor of the 80C51. To be able to use relatively slow (to the XA) 80C51 peripherals a wait state generator is needed. The wait pin on the XA is multiplexed with the EAn function and therefore some precautions has to be taken. The behaviour of this pin is different than on the 80C51, and users must be careful how to use this pin.

APPLICATION NOTE

Using the XA EAn/WAIT pin

AN96075

Author(s):

Marco Kuystermans
Systems Laboratory Eindhoven,
The Netherlands

Keywords

WAIT, EAn, burstmode, clocksource

Date: 1997-03-17

Summary

To limit the number of pins on the XA some pins have multiple functions. WAIT/EAn is one of them. During RESET the EAn pin determines the memory configuration of the XA. If the XA runs the WAIT function takes over functionality. To use both functions extra hardware is needed. This document describes how to construct this hardware.

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

CONTENTS

Table of Figures

1. THE XA EAN/WAIT PIN

1.1 Introduction

The EAn (External Access) pin is used to configure the XA's memory configuration during reset. After reset this PIN becomes a high active WAIT pin. The XA will run in external memory only mode when EAn is held low DURING reset. When EAn is held high during reset, the XA will run in on-chip or mixed memory mode.
During RESET all I/O pins are pulled high, however it depends on the logical level on the EAn pin if it is a weak or a "strong" (low impedance) pull-up. The status of the EAn pin is immediately reflected into the type of pull up, even during reset.

Unlike the 80C51 [2] the XA EAn pin cannot **always** be connected *directly* to Vdd or GND. If EAn is connected to Vdd WAIT states will be inserted as soon as the XA accesses the external bus (but the external bus only, because the WAIT pin only applies to the external bus).
The following table gives an overview of all possible WAIT/EAn combinations.

Table 1, Possible memory / WAIT combinations

	External only	**Mixed mode**	**Internal only**
WAIT needed	Connect EAn to WAIT generator	EAn circuit + WAIT generator (1.3,1.4)	NOT APPLICABLE
No WAIT needed	Connect EAn to GND	only EAn circuit needed (1.3)	Connect EAn to Vdd

In the following situations NO extra logic is needed:

- If the XA must run in external mode only and NO wait states need to be generated; the EAn/WAIT pin can be directly connected to ground.
- When in external memory mode wait states are needed, the EAn/WAIT pin can be connected to the WAIT state generator directly without extra logic. During reset ALL data strobes are high and consequently NO wait signal will be generated (be absolutely sure that WAIT cycles are generated only as a result of a data strobe, see 1.4)
- In case the XA will run internal only, EAn can be connected to Vdd directly because no wait states can be generated in this memory mode. Please be absolutely sure the XA is not accessing external memory in this mode. The XA will enter an infinitive wait as soon as external memory is accessed. Setting the WAITD (found in BCR, sfr address 0x46A, no bit address available [1]) bit can prevent this behaviour. This bit disables the WAIT function completely.

In all other cases extra logic is needed described in the following sections.

1.2 EAn/WAIT pin design considerations.

Before the XA WAIT pin can be used, be aware of the following things:
The EAn/WAIT input pin has NO Smitt-trigger, this means that the rising and falling edges of WAIT must be steep. If NOT, due to oscillation, the XA will generate unpredictable events, for example the XA can generate a exceptions. This restriction rules out a wired logic solution, where both signals are connected via an open drain to the EAn/WAIT pin. The slopes of this type of signals is not steep enough to be used with the XA EAn/WAIT pin.

A WAIT signal needs a holdtime and must be generated within a specified time, see Figure 1.

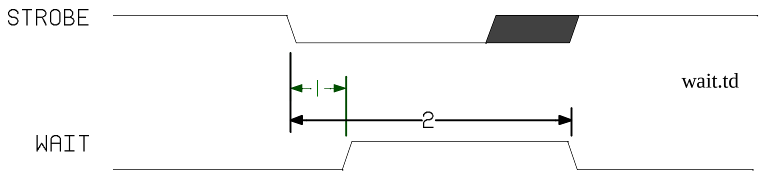

Figure 1, WAIT hold time (2) and WAIT asserted after Strobe asserted (1)

Because XA data strobes (code read and data read/writes) lengths are individually programmable, WAIT must be generated correspondingly [1]. It is impossible to generate WAIT states when the XA bus controller is programmed as 1 clock cycle data reads without ALE. For most XA data cycles WAIT needs to be generated at least 1 clock (plus 34ns) before the end of the strobe (decision point). For code reads and wait cycles with strobes of one clock in length, WAIT may be generated as late as 34ns before the end of the strobe (no extra clock).

It is allowed to generate a wait strobe before a data strobe is asserted. The following situation however should be avoided:

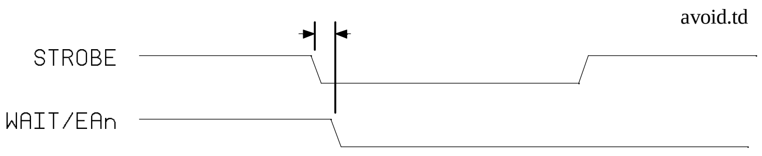

Figure 2, Too short WAIT hold time

Because of the combined WAIT and EAn “feature” this situation can occur when a application initialises (see 1.3), as a result the XA can lock up.

1.3 Generating the EAn signal.

The XA samples the EAn pin at the rising edge of RESETn, however a hold time is needed (see Figure 1). This hold time must be at least three clock cycles long. Therefore the EAn generating logic

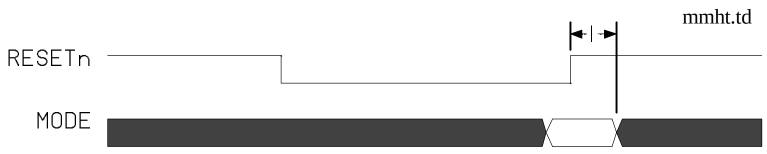

Figure 3, Memory mode hold time

needs some delay. At 20MHz the hold time is 150ns, so just simply gating this signal with RESETn will not do the job.

Absolute condition for proper operation of all EAn networks is that a “clean” RESETn is supplied to the XA. The XA RESETn input is Smitt-triggered, so a normal RC network can be used to generate the appropriate reset signal. However if a RESETn is constructed via an RC network and additional logic, for example reset is generated by an other device, it is important to use Smitt-triggered logic. If not, the additional logic can oscillate at the rising edge of RESETn. The XA does not tolerate this oscillation on the RESETn pin, and can lock-up or initialises in the wrong memory mode.

1.3.1 Generating EAn using an RC network.

The easiest way of generating an appropriate EAn signal is using a simple RC network (Figure 4). The RC loop is triggered at RESETn, holding the EAn signal for a certain period of time determined by the RC time. It is very important to use Smitt-triggered logic due to the slow slopes.

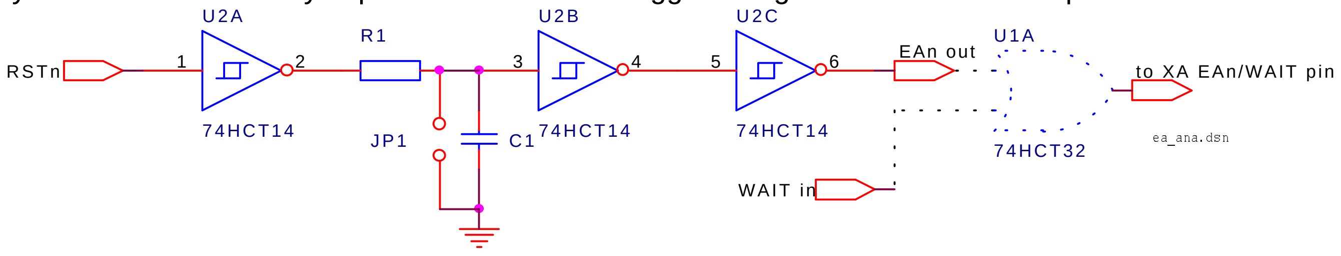

Figure 4, Generating EAn

The RC circuit can be calculated as follows:

$$Uth = Uo \cdot e^{-\frac{t}{RC}} \Rightarrow C = -\frac{t}{R \cdot \ln\left(\frac{Uth}{Uo}\right)}$$

U_{th} = HCT14 Input threshold voltage [3].
U_o = HCT14 output voltage (5V) [3].

When R_{in} >> R (e.g. 1k) and XA running @ 20MHz (t_c = 50ns), it will take a minimum of 7 clock cycles before the first ALE is generated after RESET:

$$C = -\frac{350ns}{10^3 \cdot \ln\left(\frac{0.9}{5}\right)} \Rightarrow C \approx 200pF$$

The minimum of 7 clocks applies when the XA is running external only (EAn = 0). Using the EAn circuit will force the XA to boot internally (EAn = 1). Thus it will take longer before the first external cycle is generated and consequently the value of the capacitor may be higher. Again, as stated in section 1.1, the EAn circuit is not needed if absolutely NO external cycle will be generated (i.e. the XA is executing internal only).

The jumper in 1.3.1Figure 4 has been added to enable the user to choose between the two memory modes. In case WAIT states are needed, the circuit can be combined with a WAIT state generator via the OR gate. If there is no need for generating wait cycles, [EAn out] can be connected directly to the XA EAn/WAIT pin (see Table 1).
The main advantage of this circuit is that it is very simple and exists of only one type of logic (74HCT14 [3]). A disadvantage however is that this circuit can never be implemented in programmable logic because a PLD is normally not equipped with Smitt-triggered inputs [5]. Therefore extra logic is needed something that you try to prevent with programmable logic.
In the next sections however alternative (C)PLD compatible solutions are presented.

1.3.2 Generating EAn using XA reset behaviour

During RESET all XA control, address and data pins are pulled high. Figure 6 shows that after reset it takes some time before ALE will become low for the first time. This behaviour can be used to

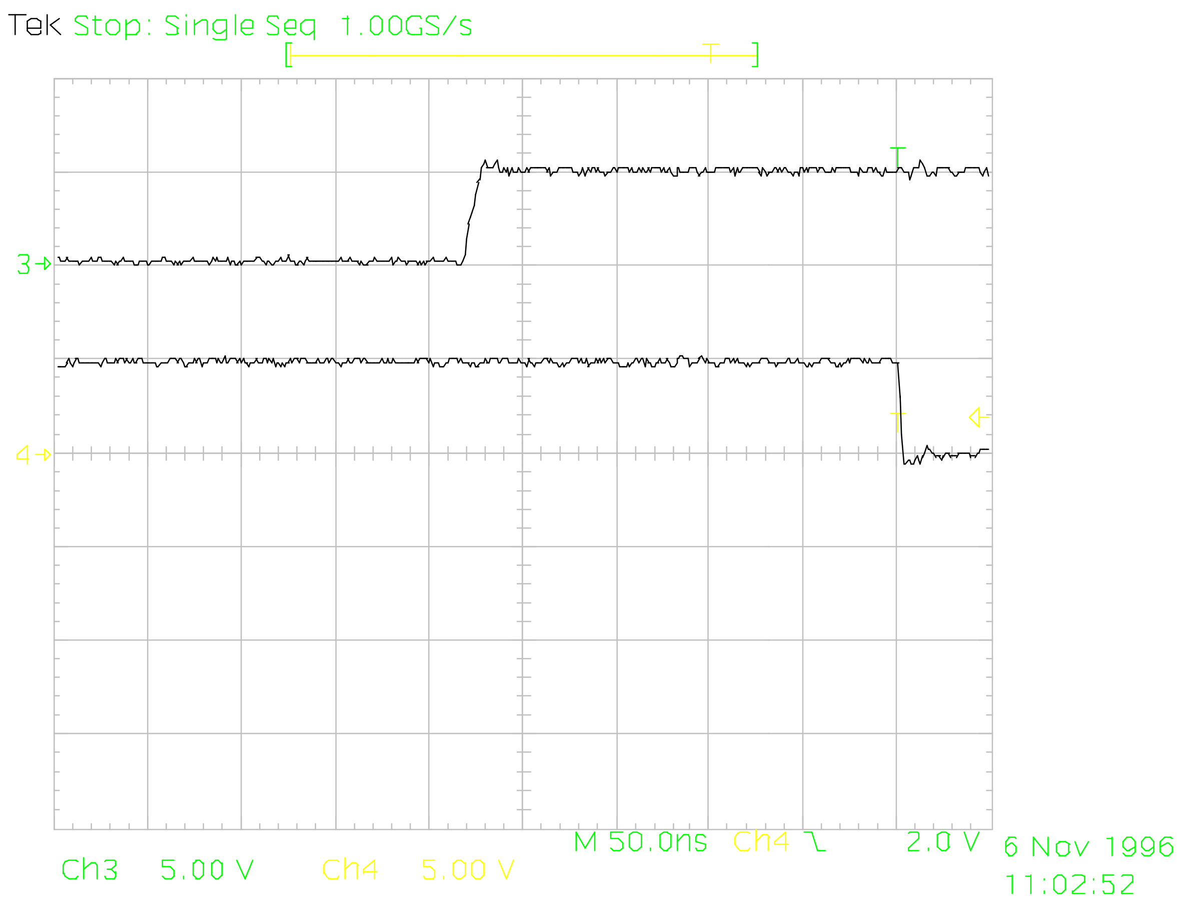

Figure 6, RESETn and XA control line relationship, 3 = RSTn / 4 = ALE

determine whether or not the XA is in reset state or not. This behaviour is based on the fact that an external cycle will be performed, if not the XA runs fully internal and the EAn circuit is not needed. Consequently EAn/WAIT can be connected to Vdd because WAIT only applies for the external bus (see 1.1).
The ALE strobe can be used in combination with for example the PSENn strobe. Besides during reset, ALE and PSENn are also high at the same time during an ALE cycle (i.e. address is available

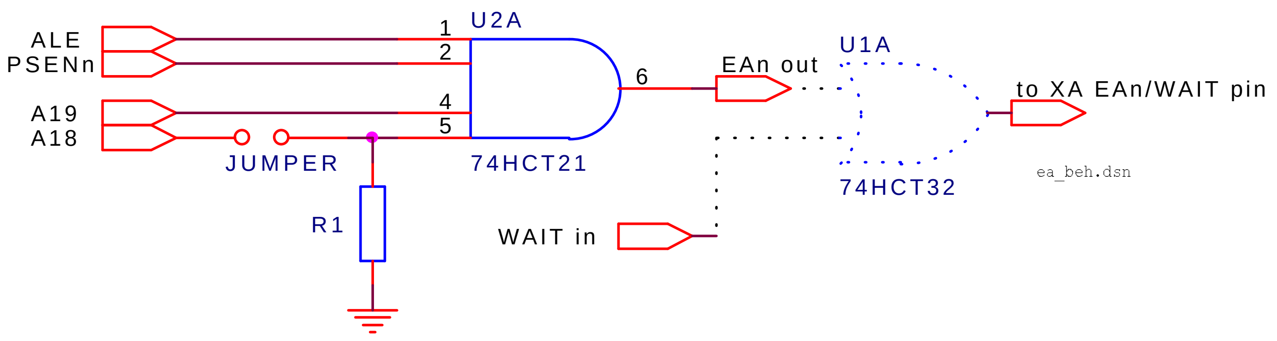

Figure 5, EAn generator using XA RESET state

on the address/data bus), consequently during this period also WAIT will be high. It is however allowed to generate a WAIT strobe outside a data strobe, but in this case NO wait cycles will be inserted.
To improve EMC behaviour it is desired to suppress these spurious WAIT strobes. This can be achieved by ANDing ALE and PSENn with a couple address lines, e.g. A19 and A18. Of course if the same address is used to decode chip selects, again this spurious wait will be generated (but now less frequent). Figure 5 shows an implementation, more addresses can be decoded if a larger AND gate is used.
If the Jumper is applied the XA will run on-chip (EAn high during reset), and consequently will cause the I/O pins to be configured as quasi bi-directional. A weak internal pull-up will make the I/O pins high. Therefore R1 in Figure 5 must have a relatively high value, so R1 > 10k.
A CPLD description of this circuit is relatively easy [4]:

```
WAIT = ALE & PSEN & A19 & ... & Ax + EAnIN;
```

1.3.3 Generating EAn using a D flip-flop

The final and most secure option is using a D flip-flop with asynchronous PRESET. During reset (RSTn = 0) the D flip-flop is PRESET, the flip-flop is reset on the rising edge of the first ALE (data = 0 clocked in), ALE is used as clock. The flip-flop will stay set as long as no exteral cycle is performed. This means if the flip-flop has never been reset, no external cycle has been performed (the XA runs fully internal), consequently the EAn circuit is not needed. In this case EAn/WAIT can be connected to Vdd because WAIT only applies for the external bus (see 1.1).

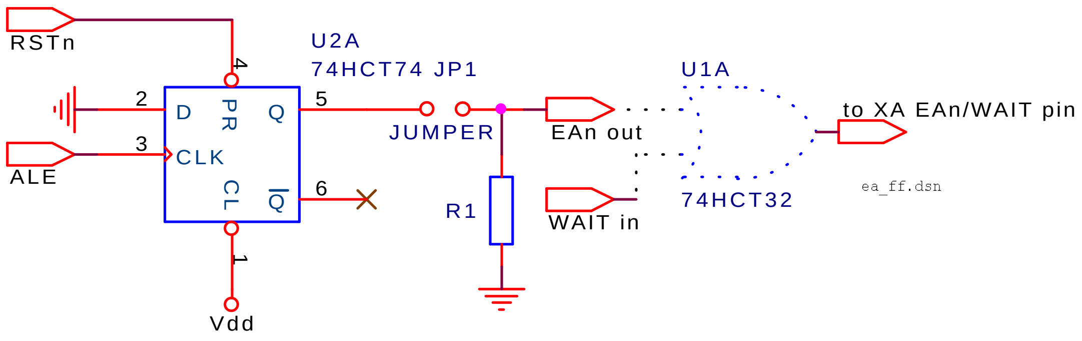

Figure 7, Generating EAn using D-flip-flop

Jumper J1 is used to determine between internal/mixed or external memory mode. When jumper J1 is not applied, resistor R1 will pull down the XA EAn/WAIT pin or, if combined with a wait state generator, one of the OR gate (U1A) inputs. This means that if jumper J1 is applied the XA EAn/WAIT pin will be “0” at RESET, i.e. full external.
Jumper J1 and resistor R1 may both be removed if only on-chip memory mode is needed and no selection between internal/external is required.
As described previously, the above described circuit can be discarded if the XA must run external only. If NO wait states are needed, the EAn pin can be connected directly to GND without extra logic. However if wait states are needed, EAn/WAIT can be connected to the WAIT state generator without extra logic.
It is NOT important to buffer the RSTn signal with a Smitt-trigger because once the flip-flop has been set, it will stay set, up to the rising edge of ALE.

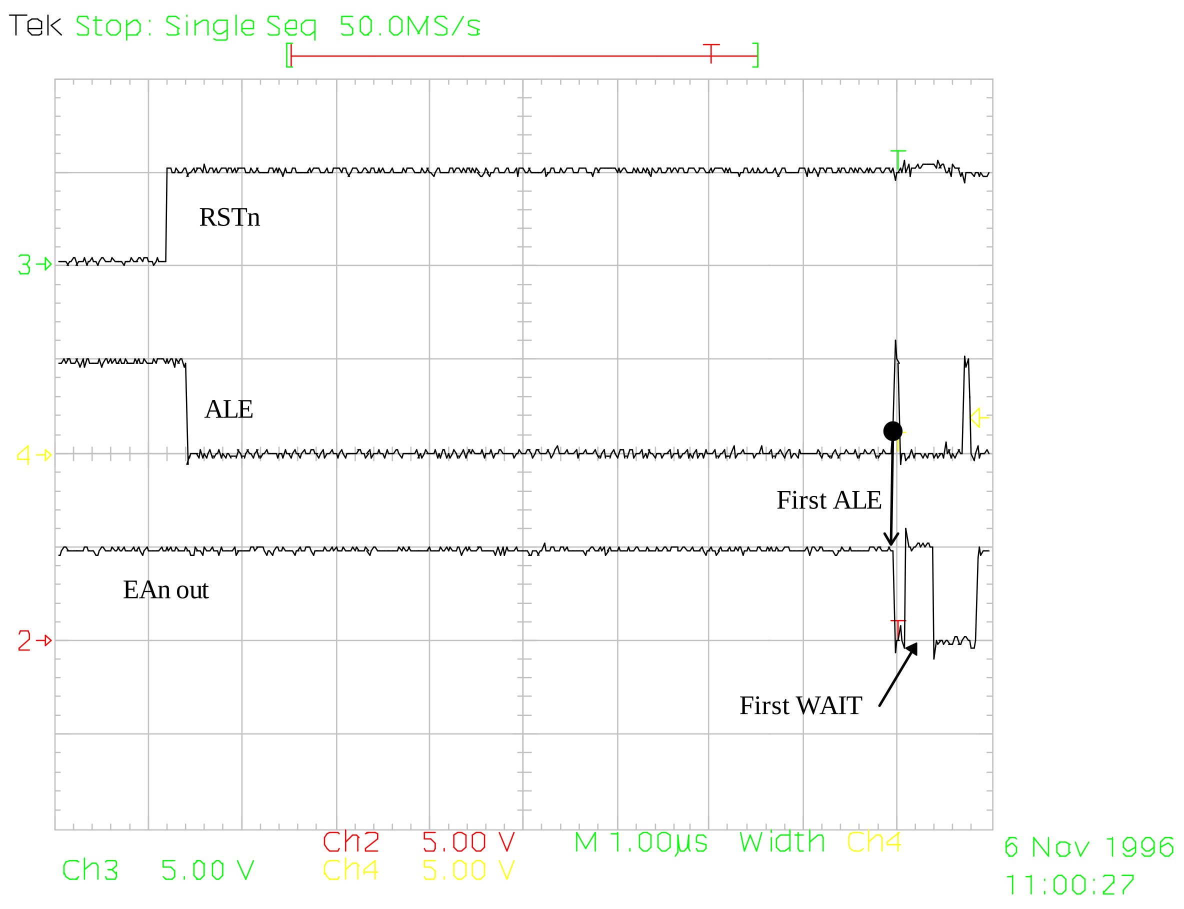

Figure 8, Generating EAn using RSTn and ALE

Figure 8 shows how EAn sets when RSTn is low and resets at the ALE rising edge. In this example it takes more than 7 µs before the first ALE strobe (= first external access, XA is running internal because EAn is high) is generated (at 20MHz). The reset falling edge is not displayed, because the reset strobe has a length of several milli seconds. Due to the resolution of the digital storage scope displaying both edges of reset will prevent the sampling of the first ALE and will consequently not be displayed.

Many designs use programmable logic to connect a microcontroller to peripherals and generate chip selects. The schematic shown in Figure 7 can also be implemented in a PLD. The following XPLA designer equations show it is very easy to implement the above described D flip-flop solution in a (C)PLD:

```
ALE_CLK     pin 4;            /* use clock pin, see PZ5032 datasheet */
ea          node istype 'reg';

equations
ea.pr       = !RSTN;
ea.clk      = ALE_CLK;
ea.d        = 0;
```

1.3.4 EAn/WAIT lock up

As mentioned in section 1.2 some EAn/WAIT situations can lock-up the XA. In this section an example will show how an EAn/WAIT signal can lock-up the XA. Figure 9 shows a problem causing design and should therefore not be used with any XA project. The transparent latch (U2A & U2B) is SET during RESETn and is reset at the first external access (datastrobes WRHn, WRLn, PSEN or RDn). The problem is EAn/WAIT is still high when the first data strobe is generated. This situation is similar as displayed in Figure 2 (section 1.2); the WAIT hold time is too short and can lock-up the XA. It depends on the propagation delay of the circuit how long it takes before EAn/WAIT is negated after the first data strobe is generated. When using HCT logic the propagation delay for the NAND gate (Figure 9) is 9ns [3].

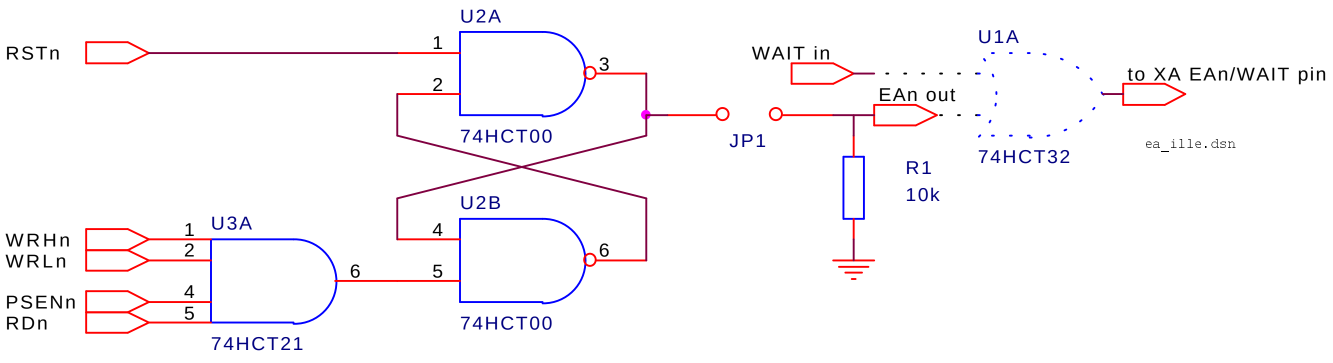

Figure 9, problem generating EAn solution

1.4 Generating WAIT

1.4.1 The XA WAIT pin

With the WAIT pin it is possible to stretch an external XA bus cycle. However the WAIT pin has no influence on code running from on-chip memory. When WAIT is asserted wait states will be inserted as soon as the XA accesses the external bus (that is asserting RDn, PSENn, WRLn and/or WRHn). The WAIT pin is therefore a part of the bus controller and not the core. When wait states are inserted the bus controller will halt the core up to the point WAIT will be negated again. In the mean time also NO internal activity takes place, because WAIT will only be granted in sequence. If not, the following erroneously situation can occur; The XA is executing code internally, at certain point data is needed from the external databus, e.g. to make a go no-go decision. If during this external access a wait signal is generated the external bus will be stretched, thus postponing the data fetch. Wrong decisions can be made if the XA continues to execute the internal code without waiting for the necessary external data.

It is possible to overrule the WAIT pin by setting bit WAITD in the BCR (sfr address 0x46A [1]) register.

The XA WAIT pin is not bi-directional and therefore the XA cannot halt external peripherals like on the 68000. It is however possible to use the 68000's DTACKn with the XA, please refer to application note AN96098 [7].

1.4.2 Illegal WAIT configuration

The XA itself can never generate a WAIT strobe. When one ore more XA data strobes are connected via an invertor to the XA WAIT input (Figure 10), the XA will latch-up as soon as it tries to

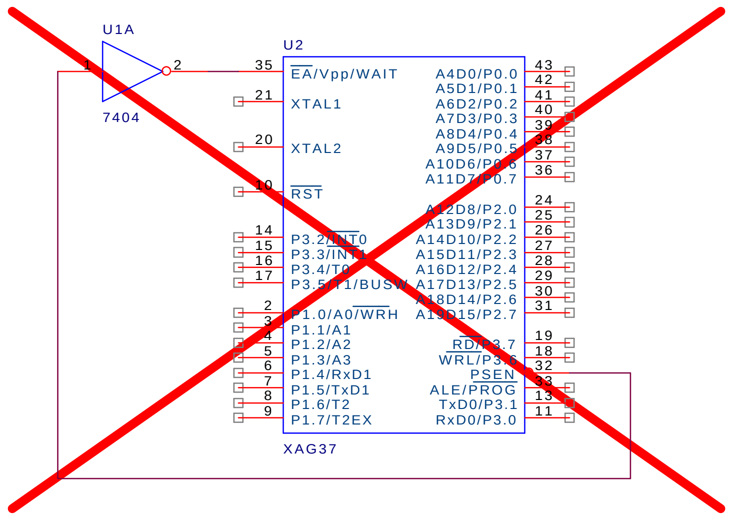

Figure 10, Illegal WAIT construction

access external memory. This is caused because when a WAIT is asserted during a data strobe, the strobe will be stretched as long as this WAIT signal is asserted. This mechanism however can never negate the WAIT signal again, because it is stretched by the same signal. This is the reason why an external WAIT state generator needs to be build.

1.4.3 Burst and non burst wait state generators

XA code and data reads can be performed during a so called burst mode. In this mode one of the 3 (4 for 8 bit data bus) non-multiplexed LSB address lines can change **without** asserting ALE and **without** negating PSENn or RDn (see Figure 11). This burst mode is transparent and cannot be controlled by the user and/or programmer.

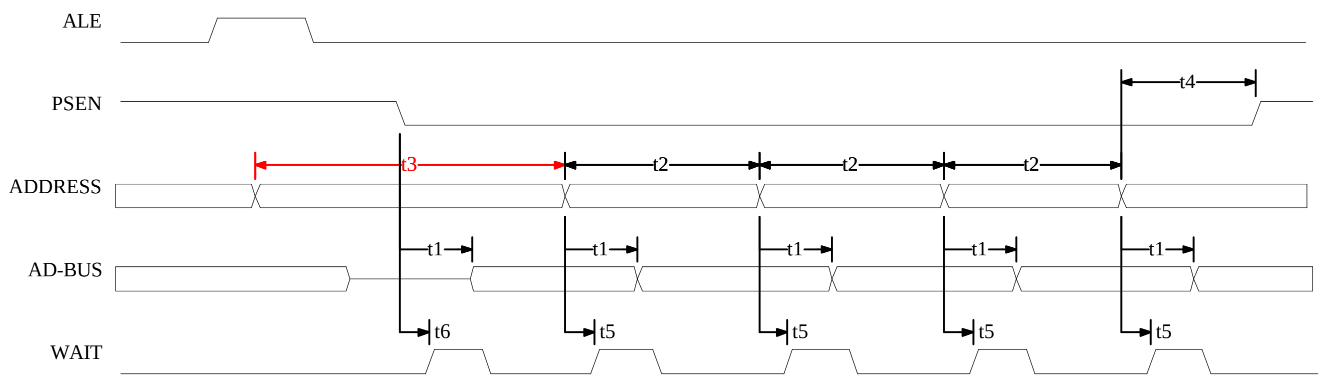

Figure 11, burst (code) read

Non bust mode wait state generators can therefore only be used in situations where no burst is generated by the XA, i.e. 16 bit write cycles. Or if the connected peripheral is only accessed one byte or word at the time. In all other cases a burst mode wait state generator is needed.

1.4.4 Generating NON burst mode WAIT states

Generating non burst WAIT using one shot

Several XA wait state implementations are available, the easiest solution is using a one shot generator (Figure 12). Looking at the schematic you can see that the WAIT signal is only generated AFTER a data strobe (i.e. RDN/WRL/WRH/PSEN) is asserted. The data strobe will also trigger the one shot generator, generating a pulse with a length determined by the RC time. By ORing the strobe with a CSn signal you can select which device needs WAIT states.

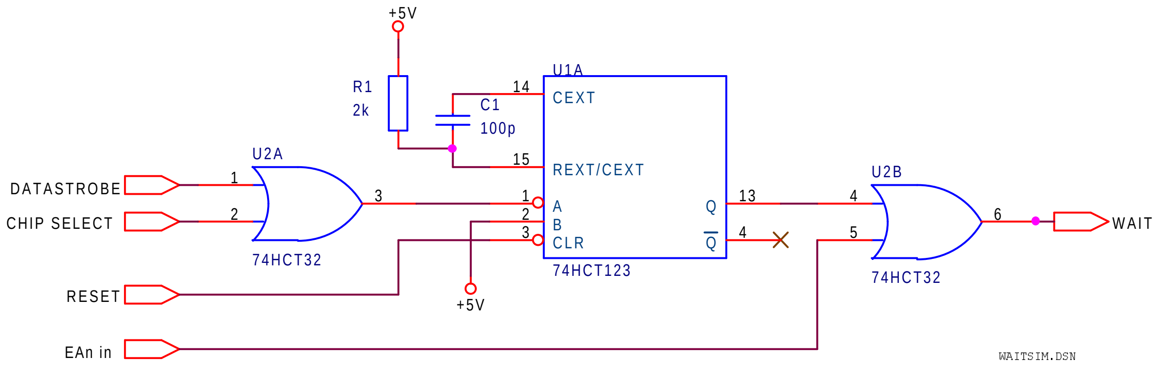

Figure 12: Wait state generator using one shot

The WAIT duration is always a whole number of clocks, therefore the RC time is not very critical except at turn over stages. The number of wait states can be made selectable when R1 is replaced by an adjustable resistor.

Generating non burst WAIT using shift register

A shift register offers a more reliable way of generating wait states. Figure 13 shows you *a* shift register implementation. This circuit only generates a WAIT signal during a datastrobe. The shift

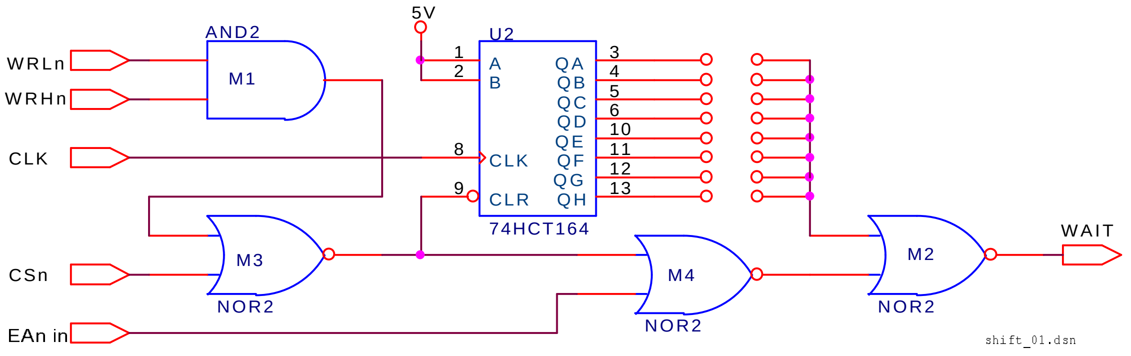

Figure 13: Wait state generator using shift register

register master reset is released when one of the data strobes is asserted AND Chip Select is low. The chip select input has bee included to make it possible to insert WAIT states for particular devices. The WAIT output will immediately become high and ones (A and B input are one) are shifted into the shift register after a data strobe is asserted.
Counting will continue until the selected (by for example a jumper) shift register output becomes high. Consequently the WAIT signal will be negated and the XA will continue by negating its strobe resulting in clearing the shift register.
To save logic in contrast with all other WAIT/EAn designs in this design for “EAn in” a NOR is used instead of an OR.
The circuit above serves as an example, more solutions are of course available [6].

1.4.5 Burst mode wait state generation

The XA burst mode can cause several problems: firstly NO WAITs are inserted when one of the address lines changes, consequently the burst cycle is too fast for the connected peripheral.
Secondly, depending how the wait state generator is constructed, waits can be inserted endlessly because the hardware expects a datastrobe.
To solve this problem besides the PSENn and RDn strobes also the address lines A1-A3 (in 8 bit databus mode A0-A3) must be monitored. So if PSENn is asserted AND the WAIT cycle has ended , new wait states need to be inserted if one of the non multiplexed addresslines change

```
PSENn H->L                      => WAIT = 1
WAIT H->L & PSEN = L            => monitor A0-A3
If A0-A3 H<->L & PSENn = L      => WAIT = 1
```

Burst mode wait state generator using discrete logic

Figure 14 shows the same shift register used in Figure 13. The magnitude comparator U1 (74HCT85) is the heart of this design. If no data strobe is generated (i.e. RDn = PSENn = WRLn = WRHn = 1) the comparator's A=B output will be zero, because the 74HCT85 A3 input is low and the

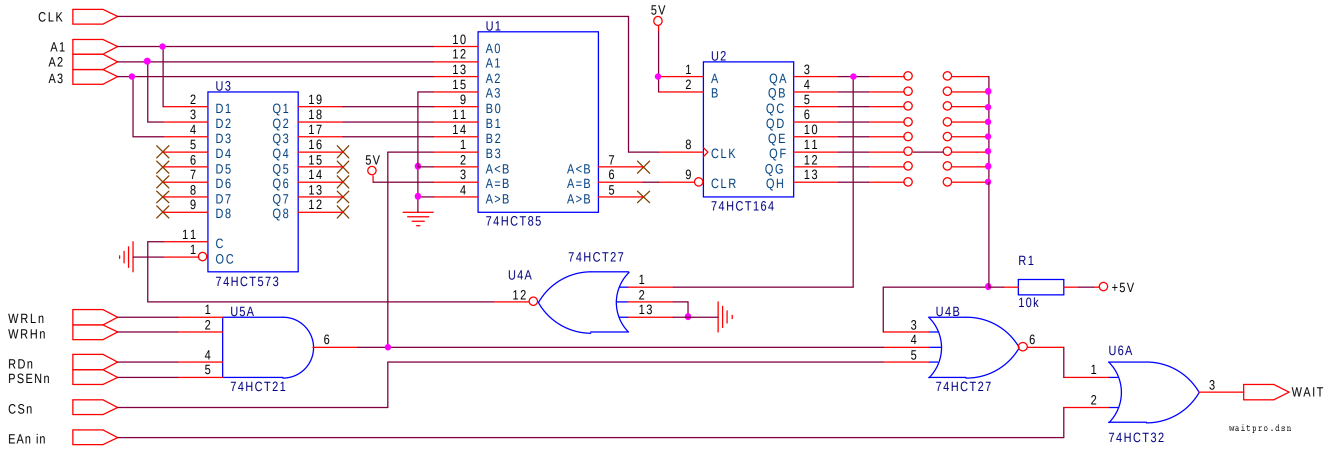

Figure 14, wait state generator with address monitor

B3 input is high. During this state the shift register is a-synchronously cleared, consequently ALL 74HCT164 outputs (QA to QH) are low. When QA is zero the 74HCT573 latch enable input (C) is high and therefore transparent for address lines A1 to A3. If one or two (both the WRLn and WRHn can be asserted simultaneously) strobes are asserted, the 74HCT85's A=B output will become HIGH because now both A3 and B3 are low. The shift register will start to shift in ones and QA will be high after one clock cycle. When QA is high the 74HCT573 latch enable will be low (C=0) and the current address is latched in. A wait signal will be generated as soon as **both** CSn and one or more XA data strobes are low up to the point the selected (by jumper) will be become high.
When during a PSENn or RDn strobe (no burst is generated in case of a 16bit write cycle) one or more addresses change (A1..A3), the comparator's A=B output will be low again and resets the shift register. Consequently QA will also be low and the latch (74HCT573) will be transparent again. Immediately the A=B output will become high and the shift register starts to shift. Because ALL shift register outputs are low after the 74HCT164 is cleared, a NEW wait cycle will be generated.

Burst mode wait state generator using CPLD

The circuit described in the previous section can also be constructed by using a Philips Semiconductors CPLD (e.g. PZ5032 [5]). The XPLA [4] module displayed below is based on Figure 14, except the shift register has been replaced by a counter. This results in less nodes, three in stead of eight.

```
Module  BURSTMODE_WAITSTATE

/***************************************************************/

/* this waitstate generator has a FIXED number of wait states */

/* Please change {number_of_wait} by required value           */
```

```
/**************************************************************/

Title   'Wait state generator for the XA'


CLOCK                   pin;                        /* clock signal from XA oscillator */

RST                     node;                       /* internal counter reset */

PSEN                    pin;                        /* program data strobe input */

WAIT                    pin istype 'reg_d';         /* WAIT signal output */

/* Alternative see section 1.4.6:  WAIT   pin; */

WRL                     pin;                        /* write low strobe input */

WRH                     pin;                        /* write high strobe input */

RDN                     pin;                        /* data read strobe input */

CSN                     pin;                        /* chip select input */

EAN_IN          pin;                                /* XA memory mode pin input */

wait_is_true    node;

/* Alternative see section 1.4.6:  wait_is_true    node istype 'reg_d'; */


bit3..bit0              node istype 'reg';          /* Up counter register bits */

count = [bit3..bit0];                               /* Up counter register */


a2..a0          pin;                                /* address sense pins */

compa = [a2..a0];                                   /* address sense register */


le                      node;                       /* latch enable, used internal only*/

latch3..latch1          node istype 'com';          /* latched in address */

latchinout = [latch3..latch1];                      /* address latch register */

/* if XA is used in 8 bit mode please add latch0 */

/*************************************************************/

equations


le = (count == 0);                                  /* latch is open when count is zero */

latchinout = (le & compa) # (latchinout & !le);     /* latch in current a3-a1*/
```

```
RST = (PSEN & RDN & WRL & WRH) # (compa != latchinout);

                                                                                                   /* do not
count when no strobe is asserted */

count.CLK = CLOCK;

count.AR = RST;

count.d = count.q + (count < 7);                    /* do not count more than 7 */


/* Alternative, Add this line. see section 1.4.6: wait_is_true.CLK = CLOCK; */

wait_is_true = (count < {number_of_wait}) & (!CSN);


WAIT.CLK = CLOCK;                          /* Alternative see section 1.4.6 : remove this line */

WAIT = ((!PSEN # !WRL # !RDN # !WRH) & wait_is_true) # EAN_IN;

end;


=========================================================================
```

On the next page a simulation can be found, the {number_of_wait} is 6.

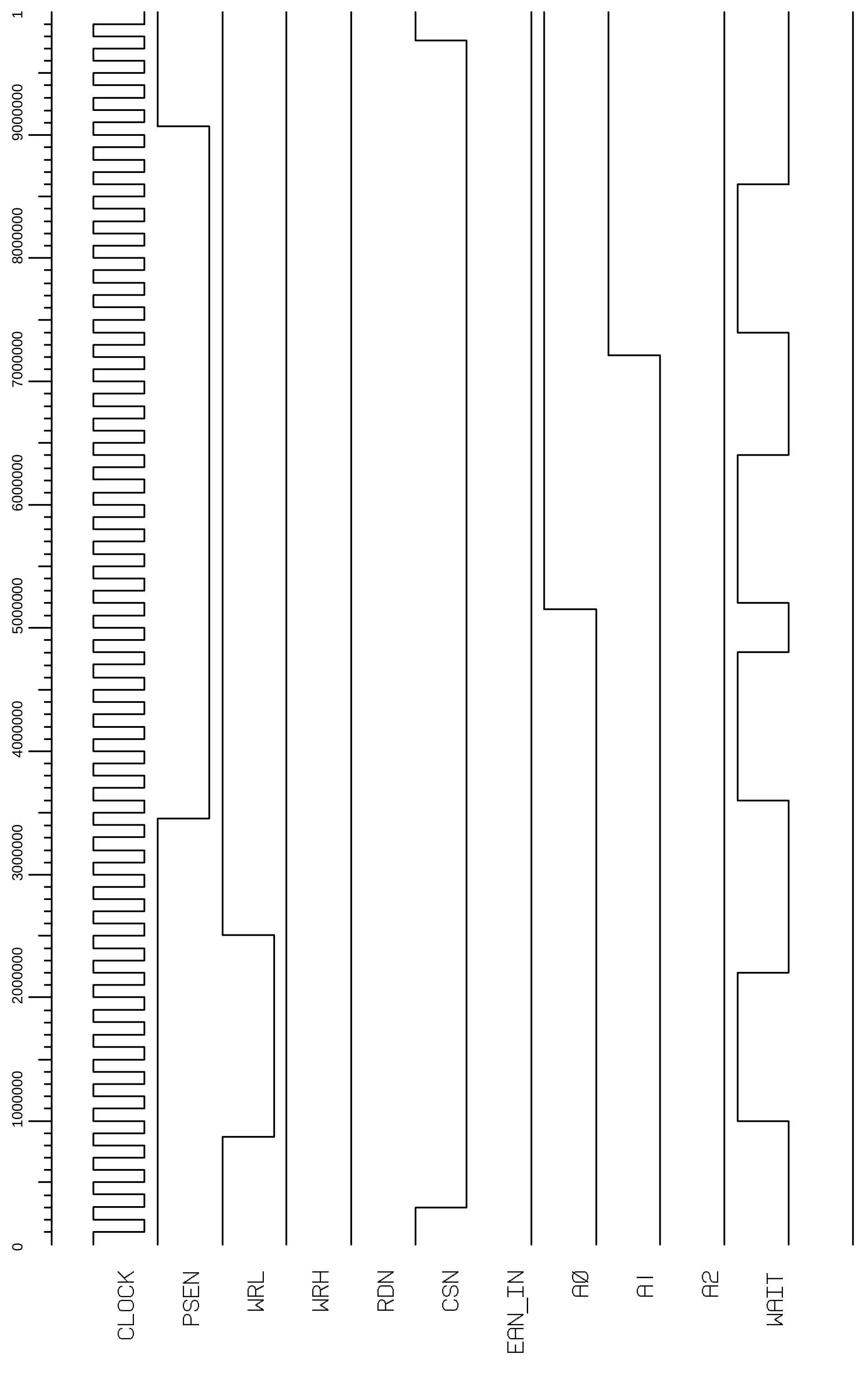

Figure 15, CPLD simulated with number of waitstate = 6

The following oscilloscope picture shows that when one or more addresses change, and PSENn or RDn is low, a WAIT will be generated. In the picture 1 represents the data strobe, 3 and 4 are A1 and A2, and 2 is WAIT

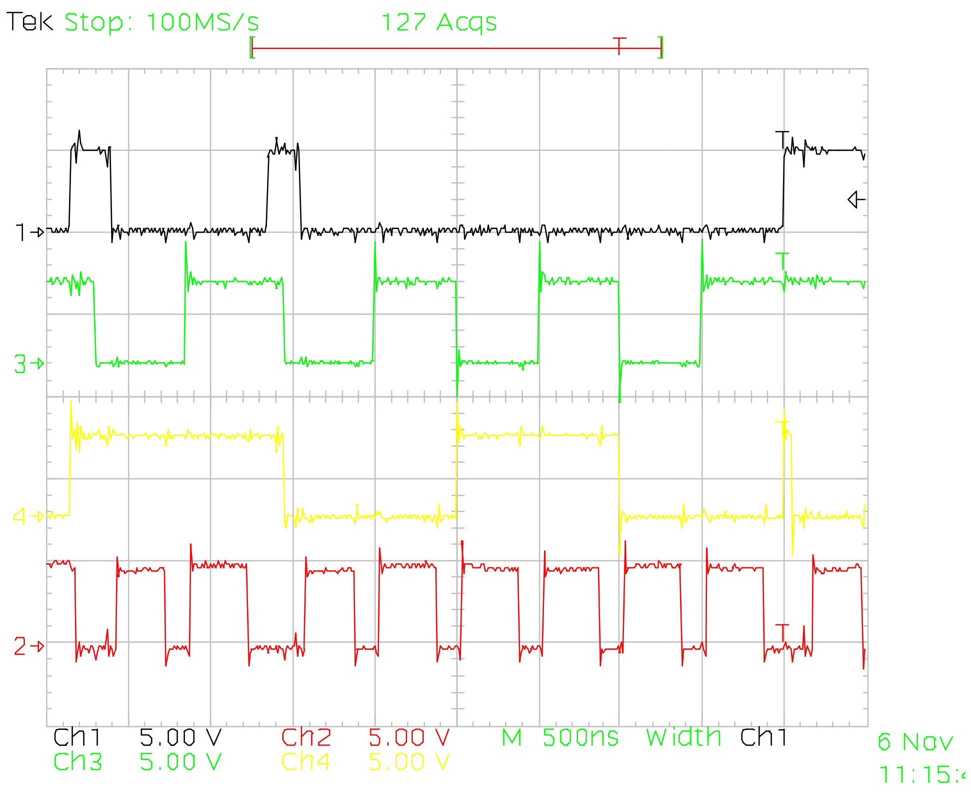

Figure 16, Oscilloscope picture with 1 = PSENn, 3 = A1, 4 = A2 and 2 = WAIT

out. Please notice the abandoned cycle on the right hand side of the picture. When a cycle is abandoned the data strobe will be negated again. The WAIT circuit does NOT generate a WAIT when none of the data strobes is asserted.

Combining one shot generator with burst detector

As an alternative for the burst mode wait state generator with shift register a burst mode wait generator with a one shot generator is displayed. In fact Figure 17 is a combination of Figure 12 and Figure 14 where the shift register is replaced by the one shot generator. This needs some minor changes in the burst mode detector because the one shot can not generate a short pulse to open the input latch. In this design the opening and closing of the latch is achieved by an OR gate which is connected to the comparator's A > B and A < B output. This way an A ≠ B signal is constructed, and is high when no strobe (PSENn, RDn, WRHn or WRLn) is generated because input A3 is low and input B3 is high. When A ≠ B is high the input latch will be transparent.

When a datastrobe is generated both inputs A3 and B3 are low and consequently A = B will become high (and of course A ≠ B low) resulting in closing the address input latch. The one shot generator will be triggered on the positive edge of A = B and thus generating a WAIT.

After WAIT is negated (determined by the one shot generator RC time), the XA will continue running. In a burst mode cycle the data strobe will not be negated, instead one or more of the addresses (A3:1) will alter. The address change will make A ≠ B high again and consequently the latch will be opened triggering the one shot generator. When the latch is transparent the new address will also be available on the comparator's B2:0 input and thus making A ≠ B low, resulting in closing the latch.

At the end of the cycle when the data strobe is negated, A ≠ B will be high and the latch transparent.

In this design the chip select input is connected to the comparator's A > B input. If chip select is not asserted (i.e. CSn = 1), the A = B output will be low and therefore NO waitcycles will be generated.

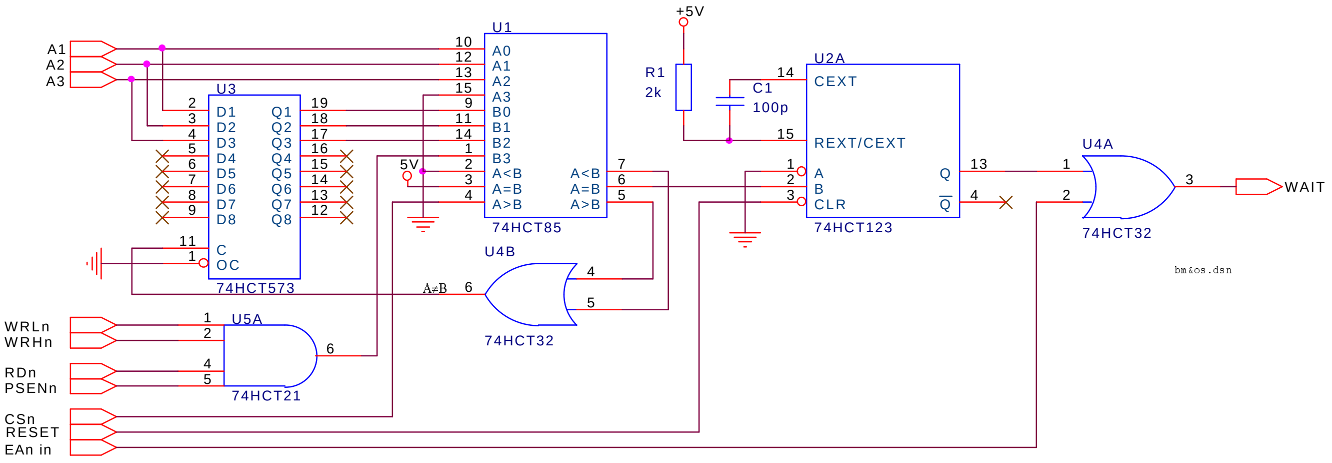

Figure 17, burst mode wait state generator with one shot

1.4.6 Known problems and solutions

The CPLD counter used in section 1.4.5 has one drawback, it can generate spikes. For example the equation *count < 7* generates a spike at the 4th clock. Going from 3 to 4 :

$\underline{0}11(3) \rightarrow 1\underline{11}(\text{spike } 7) \rightarrow 100(4)$

To prevent this problem WAIT has been constructed around a D flip-flop (see XPLA listing section 1.4.5 [4,5]) . It is however NOT guaranteed that this construction always will function. The XA and CPLD are both clocked on the same source, in some situations it can take up to a clock period before a WAIT is generated. An XA data strobe is generated at the rising edge of CLOCK (although this cannot be guaranteed), just the next rising edge will clock the connected logic resulting in WAIT being asserted. This delay can be a problem, for example when the XA write strobe length has been programmed as one clock cycle. In that case the wait signal must be generated at least at $t_x = 1^* t_c$ - 34ns. Where t_c is the length of the current strobe, this means that wait must be generated 34ns before the end of the strobe.
When the (CPLD [4,5]) logic generates a wait after one clock (minus 17ns, i.e. internal XA delay between rising edge input clock and falling edge strobe) it will be too late and WAIT will not be

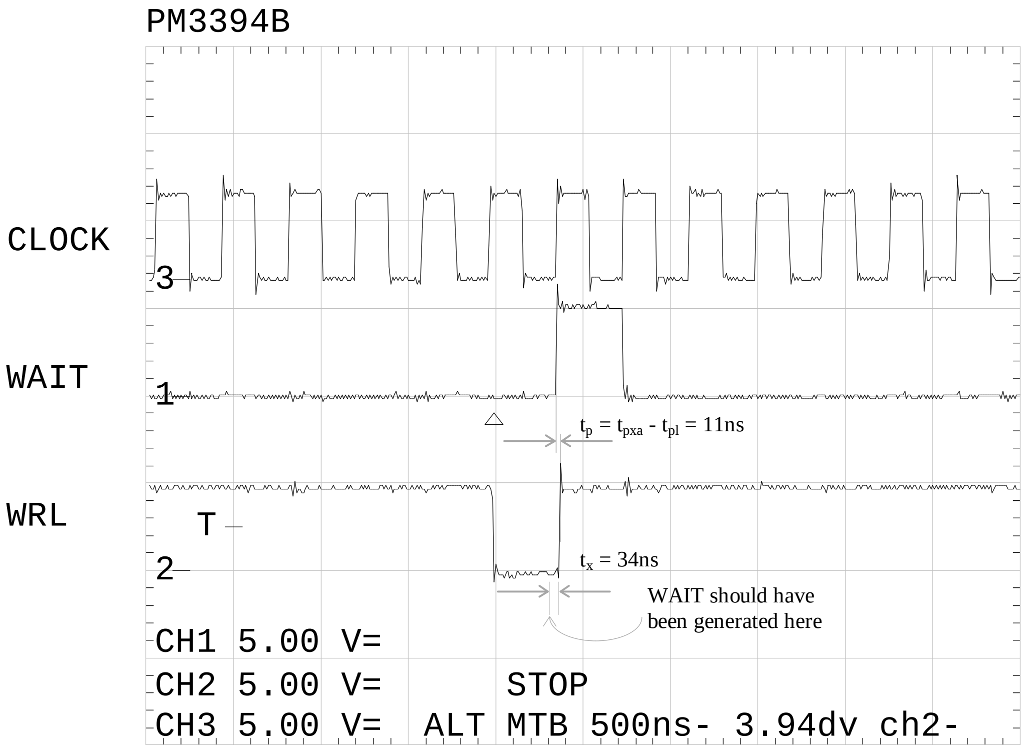

Figure 18, WAIT generated too late

sampled. See Figure 18 for an oscilloscope picture, this picture shows that the wait signal is 23ns late:
$t_l = 34 - (t_{pxa} - t_{pl}) \Rightarrow t_l = 23$ns ($t_l$ = time late, t_{pxa} propagation delay XA, t_{pl} propagation delay logic) This situation occurs when BTRL (Bus Timing Register Low) contains 0x6F [1].

Please note: although the number_of_wait in this design was 6, WAIT is only high for ONE clock period. This is caused because the counter is cleared in case NO strobe is asserted.

This problem can be solved in two ways, the first method is using !CLOCK in stead of CLOCK. Now WAIT will be asserted at the falling edge in stead of the rising edge, resulting in a delay of half a clock cycle plus logic (invertor) propagation delay.
Also this solution can provoke problems, caused by the internal XA delay between clock and the data strobe falling edge (t_{pxa}). This delay is constant (at least not depending on the clock frequency) and is around 17ns. The wait strobe must be generated after an XA data strobe is asserted, otherwise an extra clock cycle will be inserted before WAIT will be generated. So:

$0.5 * t_c = 17ns - 6ns \implies t_c = 22ns$

Resulting in a maximum frequency of 45MHz, the maximum operating frequency of the XA is currently 30MHz, therefore this solution can be used without restrictions. Please notice, the logic propagation delay can compensate t_{pxa}.

The second solution can be found in the CPLD equations. In stead of using clocked logic, combinatorial logic can be used instead to generate WAIT. The XPLA source need to be adapted (see comment in CPLD source,

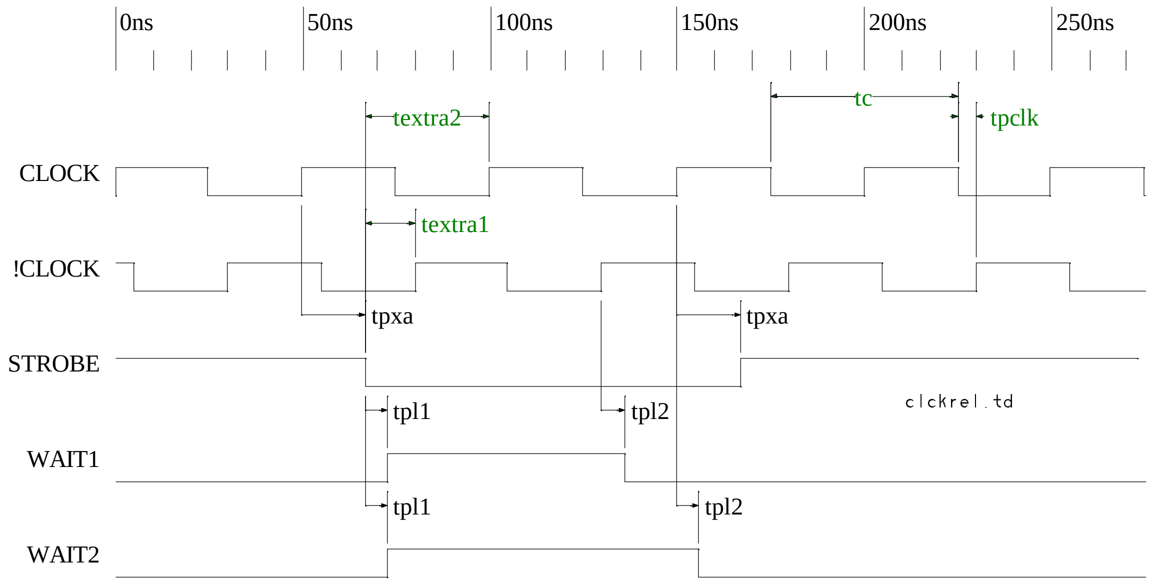

Figure 19, relation between strobe/wait/clock

section 1.4.5) to realise this implementation.

In the new equations the WAIT signal is generated immediate (minus propagation delay logic) after a data strobe is asserted. Because the relationship between the supplied clock and the XA core clock will never be 0ns, the length of the WAIT strobe is not a exact number of clocks. In stead it can be up to one clock longer than defined. In case of using CLOCK:

$t_{extra2} = t_c - t_{pxa}$

In case of using the inverted CLOCK:

$t_{extra1} = 0.5 * t_c - (t_{pxa} - t_{pclk})$

if $t_c < (t_{pxa} - t_{pclk})$ then:

t_{extra2} 1.5 * $t_c - (t_{pxa} - t_{pclk})$

t_{extra}	= time making wait strobe longer
t_c	= clock period
t_{pxa}	= time difference between rising edge clock and falling edge strobe
t_{pclk}	= time difference between CLOCK and !CLOCK
t_{pl1}& t_{pl2}	= logic propagation delay

Because t_{pl1} and t_{pl2} are both propagation delays caused by the same logic, that cancels both signals in the wait strobe length formula.

t_{pxa} and t_{pclk} are both measured and serve as an indication, the exact value can vary (influenced by temperature and supply voltage).

The next chapter (2) describes how to supply a clock source.

2. CLOCK SOURCE

Some designs in the previous sections need a CLOCK. Unfortunately the XA does not have a CLOCK output, so a core CLOCK reference is not available. Of course the XA XTAL2 output can be used, but this signal and the core clock are not in phase.

Several ways are available to offer a CLOCK source to external devices. The most common way is using a normal crystal and supplying CLOCK by XTAL2 (see Figure 20). Please be aware that extracting the CLOCK signal from this pin will raise the capacitive load (input capacitance CPLD is 8pF) on the XTAL2 pin, this can be corrected by lowering the capacitor (C2) normally found on the XTAL2 pin.

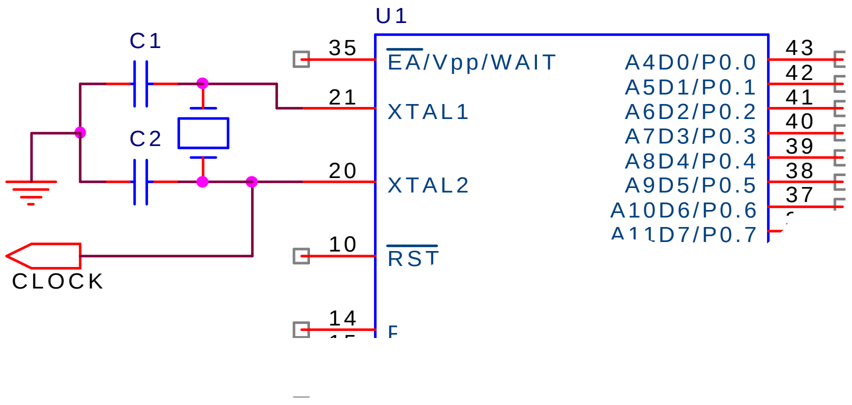

Figure 20, Clock source using XTAL

A better clock is supplied when using an XO (crystal oscillator) device. Advantage of these type of oscillator is the square wave output. An external CLOCK can be supplied in two ways, connecting the XO output to both the XA XTAL1 input and CPLD or other logic (Figure 21).

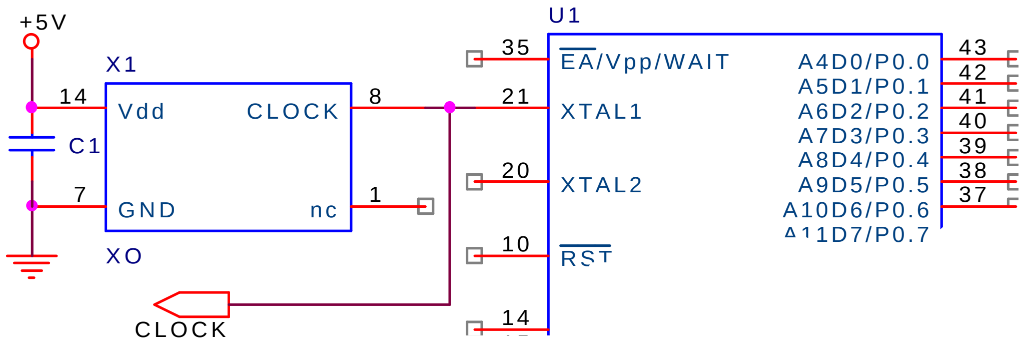

Figure 21, Clock source using XO

The other way is less conventional, the XO output is connected to the XA XTAL1 input, the XA XTAL2 output, normally dangling when using an external clock source, is connected to the CPLD or other logic. For both configurations, crystal or XO, XTAL2 is used to supply CLOCK, so in designs where crystals can be exchanged with XO's and vice-versa no jumpers are needed. Secondly the XTAL2 output is inverted, so the XA itself can provide a !CLOCK source (see section 1.4.6). XTAL2 can be used where a !CLOCK signal is needed, and therefore logic can be saved.

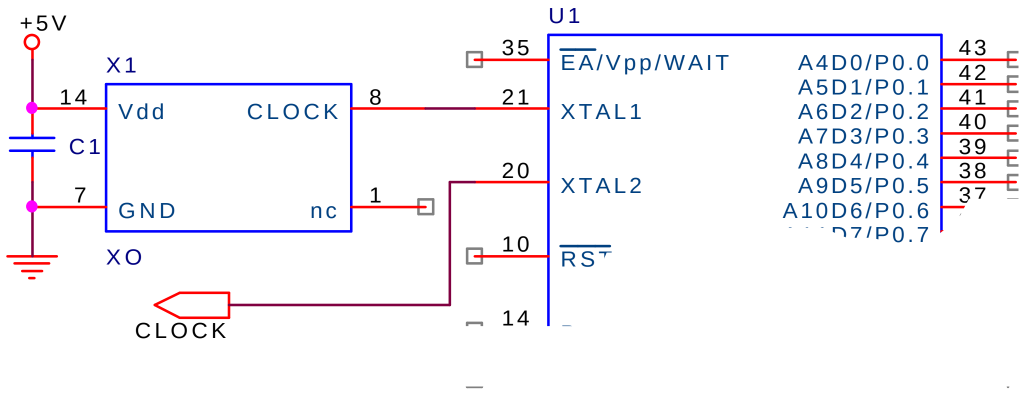

Figure 22, Clock source using XO and XTAL2 output

The CLOCK lines should be as short as possible to improve EMC behaviour and supplying a clean CLOCK to the peripherals.

REFERENCES

[1] 16-bit 80C51XA Microcontrollers - Data Handbook IC25 PHILIPS: 9397 750 00733

[2] 80C51 based 8-bit Microcontrollers
Data Handbook IC20
PHILIPS: 9397 750 00013

[3] High speed CMOS Logic Family Data Handbook IC06 PHILIPS: 9397 750 00454

[4] XPLA Designer version 2.1 User's Manual

[5] DATA SHEET PZ5032 (CPLD)

[6] Electronic Circuits, design and applications U. Tietsche, Ch. Schenk Springer-verlag ISBN 3-540-50608-X

[7] Application note AN96098 Interfacing 68000 family peripherals to the XA M. Kuystermans, PS-SLE 1996.