



NOTE: The XC4000E variant of the XC4000 family has improved RAM capability including fully-synchronous RAM timing plus dual-port RAM. See the XC4000E data sheet for additional details.

Using the XC4000 RAM Capability

XAPP 031.000

Application Note By ROMAN IWANCZUK

Summary

The XC4000 family of LCA devices permits CLB look-up tables to be configured as user RAM. This Application Note provides background information for users of the feature, and points out the need for carefully designed control logic. The Application Note, *High-Speed RAM Design in XC4000* (XAPP 042, page 8-139), shows a rugged, simple and elegant solution to this problem.

Xilinx Family

XC4000

Demonstrates

XC4000 RAM Capability

Introduction

LCA devices emulate logic using a look-up-table-based architecture. The look-up tables are implemented in static RAM that is written during configuration and read during operation. Unlike previous LCA families, the XC4000 family also permits the RAM to be written during operation. Using this feature, internal RAM can be included in user designs.

The RAM function in the XC4000 permits a significant increase in the system functionality that can be implemented in an FPGA. This includes traditional RAM-based logic such as FIFOs, LIFOs, register files, as well as novel applications such as the RAM-based shift register described in this application note. Interfacing to the RAM is not particularly difficult, but it requires an understanding of the issues involved.

With ~10 ns cycle time, the XC4000 RAM is much faster than the memories with which most designers are familiar; most discrete SRAMs have cycle times of 55 ns or longer. Consequently, the design of XC4000 control circuitry is more critical. Many factors, such as interconnect delays, that can usually be ignored in discrete RAM designs cannot be ignored in an LCA RAM design. Using the XC4000 RAM is like using very fast discrete SRAMs (<25 ns cycle time), where similar factors must be considered.

Figure 1 shows the address, data and control signals available on the XC4000 RAM compared to those of a discrete SRAM. Notice that the output of XC4000 RAM is permanently active, since it does not have a Chip Enable control. If a 3-state output is required the Data-Out signal can be connected to a TBUF input, as described later. Another difference is that the Write Enable on the XC4000 RAM is active High, while it is typically active-Low on discrete SRAMs. Some functional differences also exist, and are described later in this section.

A further point to note is the granularity of the XC4000 RAM. The example shown is a 16 x 1 memory, the smallest XC4000 RAM primitive. A similar 32 x 1 primitive exists; both these primitives can be combined to provide larger memories. In contrast, the smallest monolithic SRAM used in today's designs is generally 4K bits.

XC4000 RAMs consume CLB resources that could otherwise be used to implement logic, and large RAMs may restrict the amount of logic that can be included. Table 1 shows the resources used by each of the RAM primitives, and how many of each could be implemented if various members of the XC4000 family were entirely devoted to RAM. As may be seen, the total amount of memory that can be implemented in an entire XC4010 is only 12,800 bits, making it a very inefficient replacement for large SRAMs

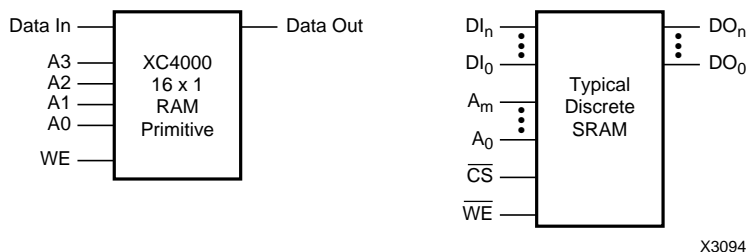


Figure 1. XC4000 RAM and Discrete SRAM Connections

Table 1. Trade-off Between RAM Primitives and Logic.

Note: If all CLBs are used as RAM there are none available for logic implementation

RAM Module	Equivalent Logic	Maximum Number of RAM Modules		
		XC4003	XC4005	XC4010
16 x 1	4-input Function Generator (F or G)	200	392	800
32 x 1	Two 4-input Function Generators and One 3-input Function Generator (F+G+H)	100	196	400

The XC4000 RAM is intended for use in small, fast RAMs in applications like FIFO buffers, scratch-pad memories and register files. For applications that require larger RAMs, it is generally more cost effective to use an external monolithic SRAM connected to the XC4000. This would, however, increase the number of I/O pins needed on the FPGA, and potentially decrease the speed of the design due to the off-chip memory accesses.

Figure 2 shows the read-cycle timing of the XC4000 RAM compared to that of a discrete SRAM. For the comparison, the SRAM is executing an address-controlled read cycle, where the Chip Select signal is permanently asserted, since the XC4000 RAM primitives do not have Chip Select control. The Write Enable signal is not shown in these diagrams, and must remain inactive during a read cycle in both cases. The diagrams are not drawn to scale to permit the relative shapes of the waveforms to be compared more easily.

The diagrams are very similar. The only difference is that on the discrete SRAM, the output data cannot change for

a period, t_{OH} , after an address change, while it can change immediately in XC4000 RAM. This parameter is not specified explicitly, but the output of any CLB must be considered invalid immediately following an input change. This is not a problem in most designs.

The corresponding write-cycle comparison is shown in Figure 3. To match the XC4000 RAM, the SRAM timing is for a Write-Enable-controlled write cycle, where the Chip Select signal is permanently asserted. Again, the diagrams are not drawn to scale so that the relative shapes of the waveforms can be compared easily.

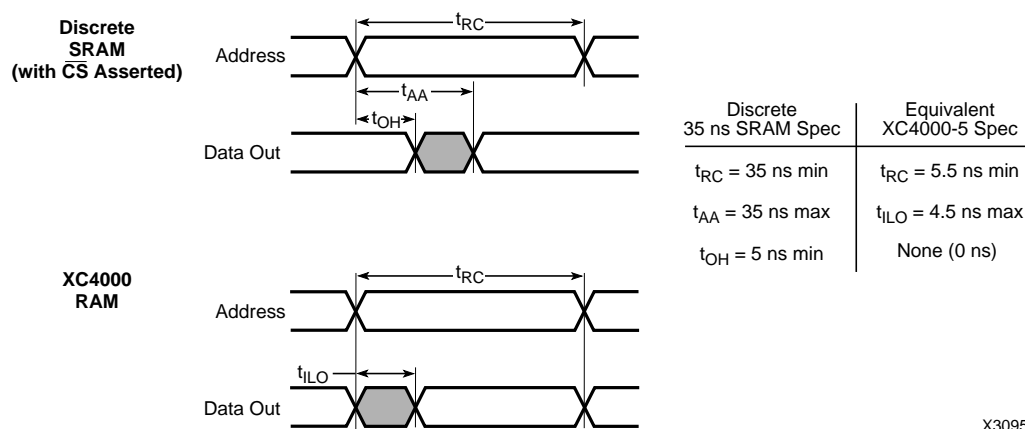
The primary difference between the discrete SRAM timing and the XC4000 RAM timing is the address hold parameter (t_{WR} on the SRAM, t_{AH} on the XC4000). In the XC4000, the address **must** remain stable for 2 ns after the Write Enable signal has been removed. This difference significantly impacts the design of the control logic, as will be discussed later.

While the Data Out signals are not shown in these diagrams, these, too, are different. In most discrete SRAMs, the Data Out signal is high impedance during the Write-Enable-controlled write cycle. In the XC4000 RAM, however, the data output has no high-impedance state and, therefore, remains active.

The write cycle starts by reading the existing data in the location addressed, and then, after WE is asserted, changes to reflect the new data. For the exact timing data output signal, please refer to the timing diagram "Read during Write" in the XC4000 data sheet.

Potential Control Logic Problems

As in any XC4000 design, the primary concern in a RAM design must be to meet the worst-case timing requirements described in the data sheet. Failure to do so can result in a design that appears to work perfectly correctly

**Figure 2. XC4000 RAM and Discrete SRAM Read Cycles**

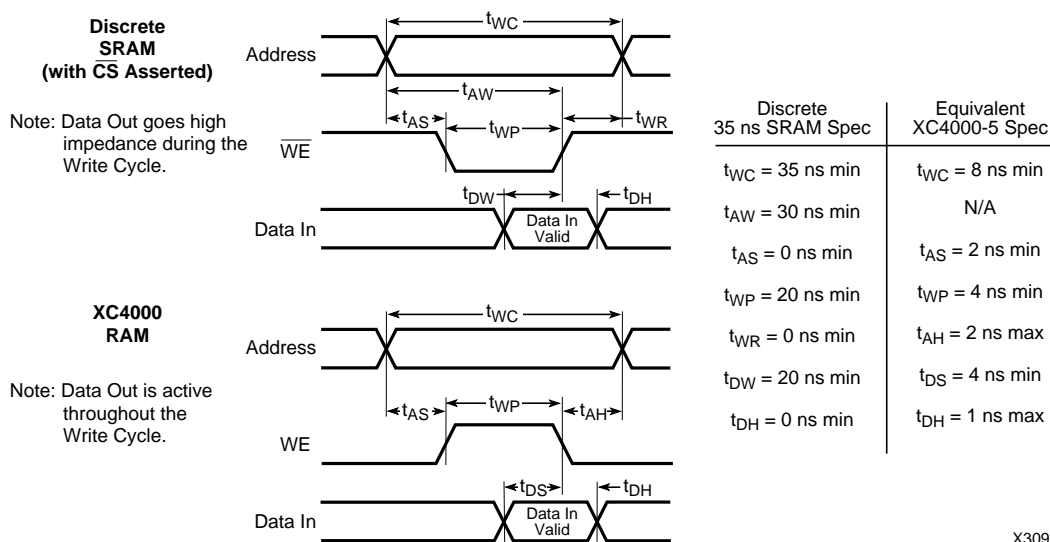


Figure 3. XC4000 RAM and Discrete SRAM Write Cycles

on the bench, but fails at a temperature extreme, at a V_{CC} extreme, or with a device from a different production lot.

A second area of concern is signal glitches, which must be avoided at all costs. Two types of glitches can cause problems in any SRAM-based design: glitches on the WE line and glitches on the address lines while WE is asserted. As has been stated earlier, the XC4000 RAM is extremely fast, and even glitches that do not meet the minimum specification for guaranteed operation can disrupt the contents of the RAM. The areas of primary concern are shown in Figure 4.

Figure 5 shows an example of a glitch-generating control circuit that might be used to generate the WE pulse in a FIFO. Notice in the timing diagram that the WE pulse is generated perfectly when Q0 and Q1 are both High. The glitch can occur as Q0 changes from 1 to 0 and Q1 “simultaneously” changes from 0 to 1; if Q1 changes before Q0, there is a momentary state that meets the requirements of the AND gate to generate WE.

This circuit might be adequate in a discrete RAM design. By judicious choice of components, the minimum timing specifications of the counter and the AND gate could be matched to ensure that glitches do not occur. Such

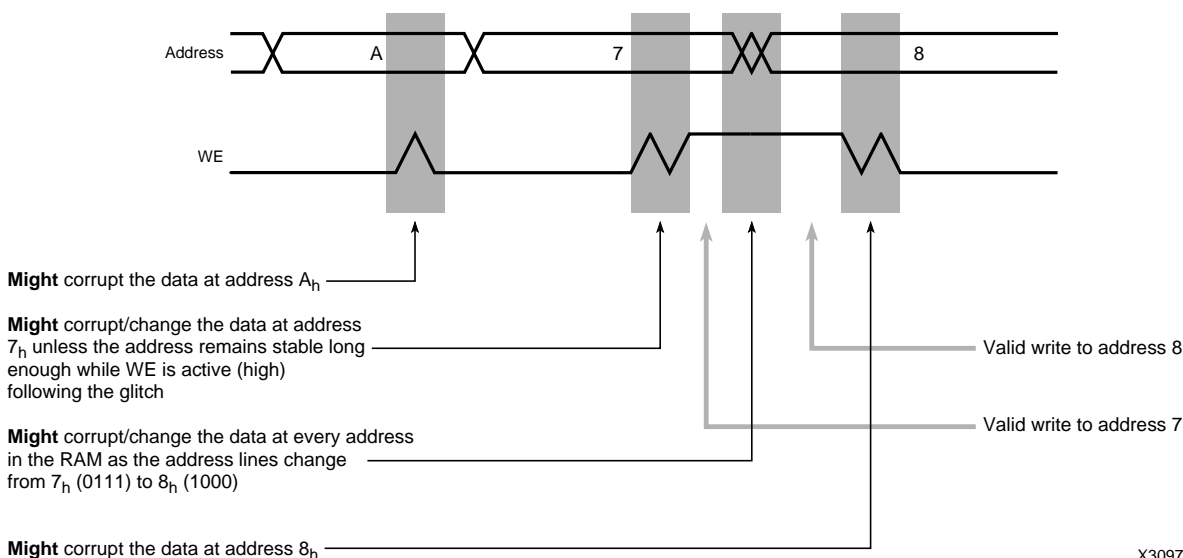


Figure 4. Typical Glitches That May Cause an XC4000 RAM Design to Fail

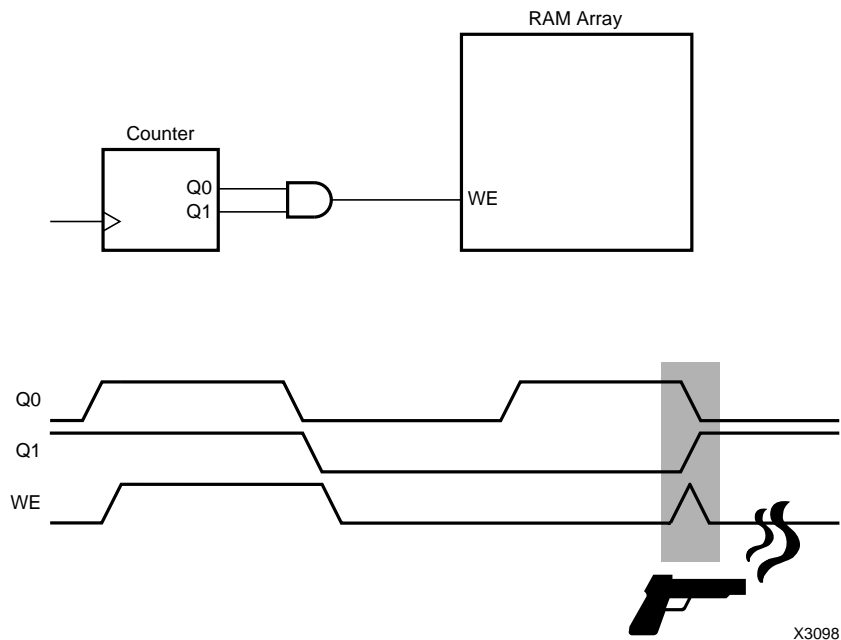


Figure 5. Example of a Marginal WE Generation Circuit

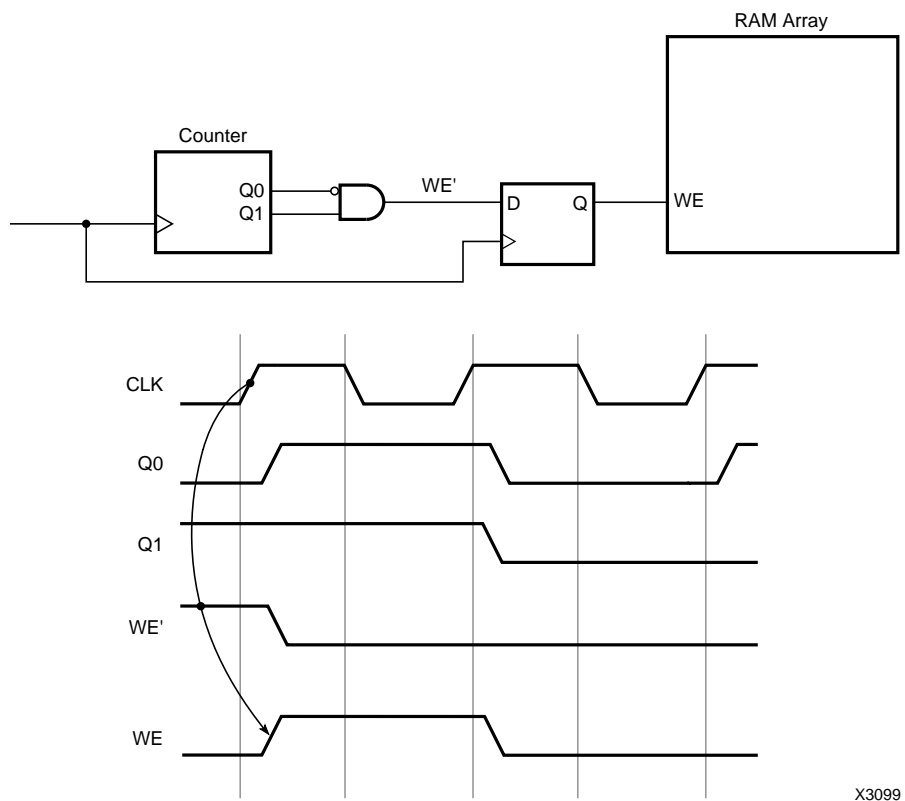


Figure 6. Example of a Glitch-Free WE Generation Circuit

matching is not possible in an FPGA environment, where the possibility of glitches is increased by the high speed of the logic functions and the relatively long routing delays. Figure 6 shows a better, glitch-free circuit.

In general, some valid techniques used in a discrete design can create marginal designs in the high-speed LCA environment. Avoid asynchronous circuits like the plague. With a little thought, most things that are be done asynchronously can be better done synchronously. If necessary, use small Gray-code or Johnson counters that can be decoded in a hazard-free manner. In a Xilinx FPGA, such counters are as easy to implement as binary counters.

Routing Delay Issues

FPGA routing delays can cause a circuit that works at speed on paper not to operate under worst-case conditions. In this situation, worst-case conditions must be interpreted as slow operation, fast operation, or any combination of these that causes a malfunction. The following issues should be considered.

- The WE signal is skewed in time by the routing delay introduced by its net. Make sure that the circuitry used to control the address and data signals takes this into account. The t_{AH} and t_{DH} requirements must not be violated.
- Compared to small RAM arrays, large RAM arrays have higher fan-out address lines with longer routing delays. Consequently, for a given speed, the address-generation circuits have less time in which to operate. Generally, large, fast RAM arrays require more ingenious control circuitry, and may necessitate partial duplication

of the address circuitry to drive separate segments of the RAM array.

- RAM modules which need to run at speed benefit greatly from manual placement. It pays to create a trial design that only implements the RAM and its control logic. This small design can be quickly placed and routed, and then optimized in the XACT Design Editor (XDE). The optimized placement can be incorporated into the main design using location constraints. Alternatively, the RAM portion can be converted into a hard macro, thus preserving its relative placement.

Creating a RAM Array

The XC4000 RAM is accessed as 16 x 1 and 32 x 1 primitives. In RAM applications requiring less than 16 words, 16 x 1 modules must be used with any unused addresses tied to ground or V_{CC} . 16 x n and 32 x n arrays can easily be created by connecting several of these primitives in parallel with common address signals.

For depths greater than 32 words, a RAM array must be constructed as shown in Figure 7. In this example, two 32 x 1 primitives are combined to implement a 64 x 1 RAM. The most significant address bit is used to select between the primitives, while the remaining address bits are common to both. During a read cycle, selection between the primitives involves multiplexing the output data. For a write cycle, the data is common to both primitives, and the WE pulse is gated to enter the data into only one.

TBUFs could be used to create the output multiplexer. However, at least half of a horizontal Longline would be

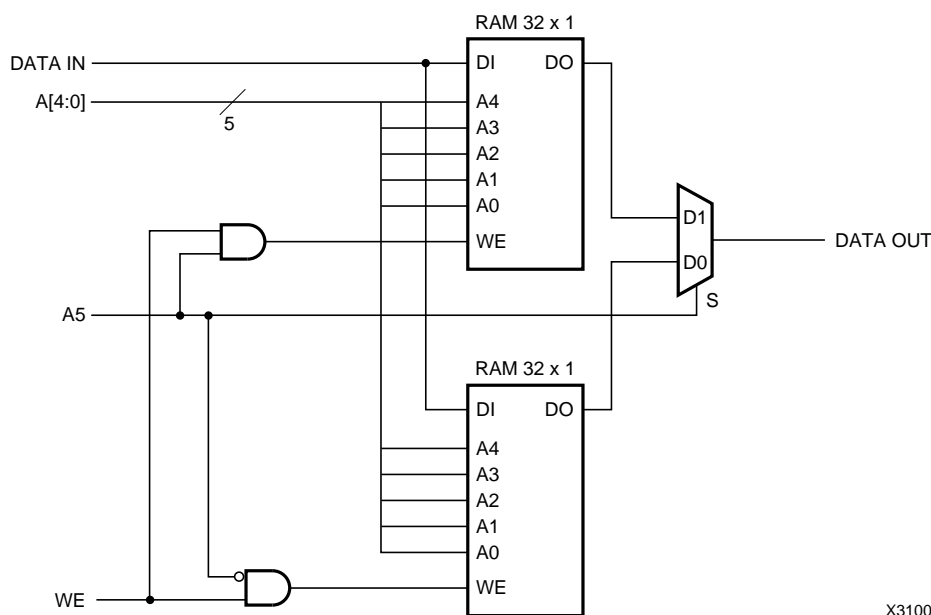


Figure 7. 64 x 1 RAM Array

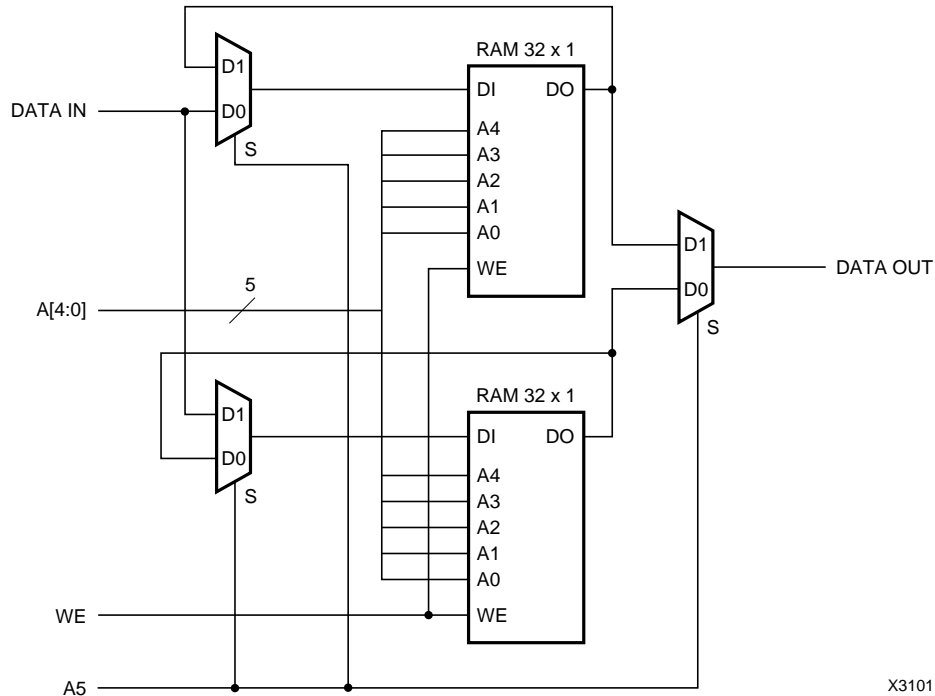


Figure 8. Alternative 64 x 1 RAM Array

consumed for each bit of RAM width, and TBUFs are slower than small logic-based multiplexers. Consequently, the use of TBUFs is only recommended for very deep RAM arrays.

Gating the WE pulse increases the delay in the WE path. This delay is not usually a problem in slower RAM applications; but, as the write-cycle time decreases, the additional delay can become unacceptable.

Figure 8 shows an alternative technique. While new data is being written into the selected primitive, the existing data is re-written into the non-selected RAM primitive. This technique introduces additional delay into the data input path, but maintains the minimum delay in the WE path, which is often the critical path. The circuit choice depends on the timing requirements of the specific system.

These expansion techniques are directly analogous to depth expansion in discrete RAMs. The only differences are the explicit output multiplexer, which would be implemented using 3-state busses in the discrete case, and the Write Enable gating, which is integrated into discrete RAMs.

Emulating SRAM with Bidirectional Data Pins

Some commercially available discrete SRAMs have a single Data Input/Output pin. This type of SRAM can be emulated in the XC4000 using the circuits shown in Figure 9. In Figure 9a, the multiplexing is performed using IOB elements; the signals inside the FPGA are unidirectional.

In Figure 9b, the bidirectional data line is extended into the FPGA and the RAM uses TBUFs to drive the data line. This circuit is appropriate where multiple data sources are required to read/drive the data line at different times.

Note that in Xilinx FPGAs, the 3-state buffers (TBUF, OBUFT) have enable signals that are active-High 3-state controls, i.e., when a logic 1 is applied, the output of the buffer is high impedance, and when a logic 0 is applied, the output is active. The T pins can be viewed as active-Low Output Enables.

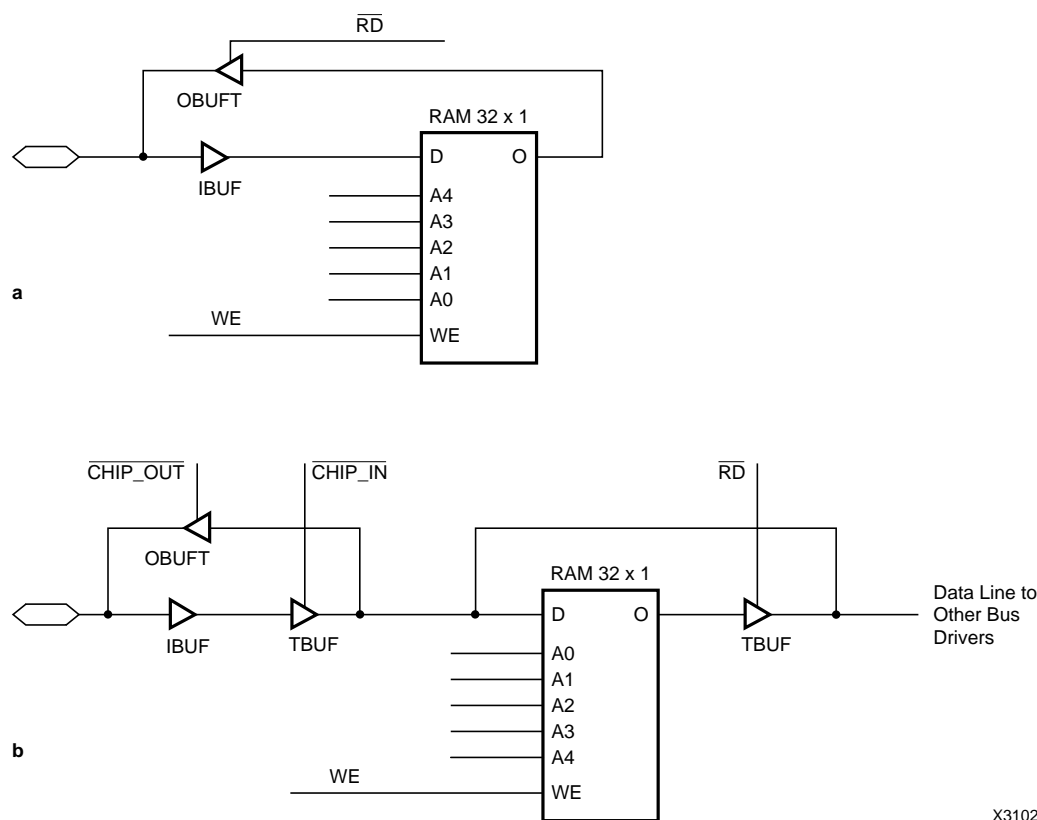
Recommended Control Logic Schemes

There are many ways to generate the WE signal for the XC4000 RAM. The choice is design dependent, and a major factor is whether the design is synchronous or asynchronous. In an asynchronous design, the WE pulse is generated from a signal originating outside the FPGA that may be gated with internally generated signals. In a synchronous design, the WE pulse is generated by logic that is completely within the FPGA.

Asynchronous Control Logic

In the asynchronous case, each design will be different, and depend on the external signals that are available. Consequently, it is impossible to make firm recommendations. However, the following discussion should illustrate some basic techniques.

Asynchronous designs generally take the form shown in Figure 10. External signals from an interface, usually a



X3102

Figure 9. Methods of Emulating a RAM that has Bidirectional Data Pins

microprocessor bus or a system backplane, are used to generate the address, control and data to the RAM. Typically, the designer is required to combine input signals to control the RAM. While bus transfers are often fast, the read cycle is usually not a problem; it is the write cycle that is difficult.

The biggest problems facing the designer are the following.

- How to create a WE signal that, at the same time, is compatible with the data and address timing of the system bus and meets the set-up and hold-time requirements of the RAM.
- How to create such a WE signal with no glitches.

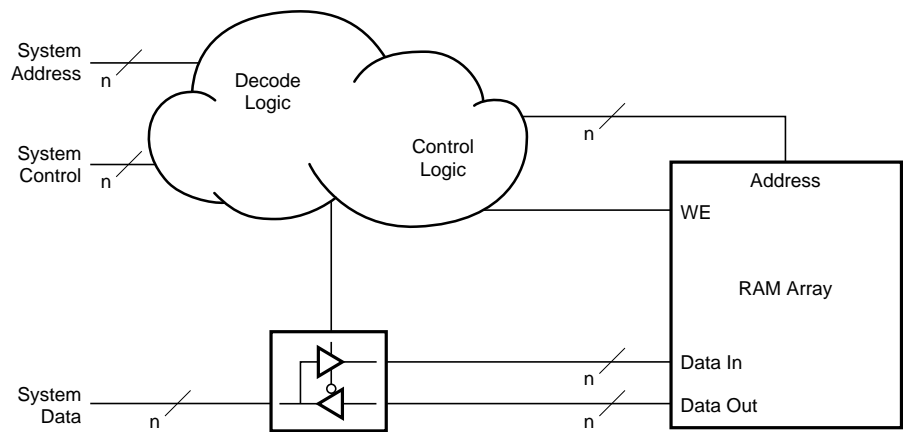
Solving these problems requires creativity. The following example describes how to solve a typical problem.

Figure 11 shows the system timing for the read and write cycles of a typical microprocessor. As mentioned previously, the design of the read-cycle control logic is rarely a problem, since the necessary interface signals are usually present early in the cycle. All that needs to be generated is a subset of the address for the RAM, and an enable signal to the output drivers. This can be done using the circuit shown in Figure 12.

The XC4000 wide decoders can be used to generate an address valid signal that can be gated with other interface control signals. The resulting signal indicates whether the RAM is being addressed during the current cycle. If the current bus cycle is a read, this signal should be registered by a flip-flop on the rising edge of T2. The resulting Qualified Read signal is used to enable the output buffers. The portion of the microprocessor address routed to the RAM depends on the size of the RAM.

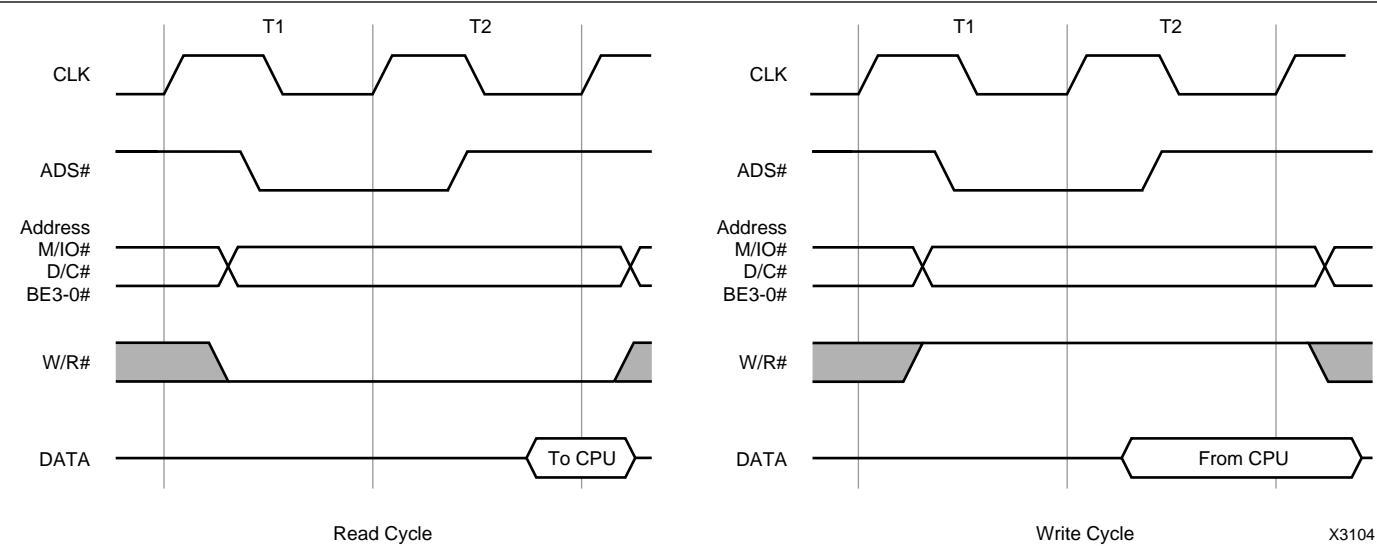
The write-cycle timing can be generated similarly to the read-cycle timing, except that the flip-flop generating the Qualified Write signal would have its CE pin connected directly to the W/R# signal. This is shown in Figure 13, which also shows the timing of the Qualified Write signal.

The write-cycle timing is more difficult than the read-cycle timing, because both the address and data hold times must be met, even with worst-case timing. It may be necessary to register the address or data to extend the time during which they are stable, Figure 14. The falling edge of the ADS# signal is used to register the address lines driving the RAM. Note that the address lines used in the control logic gating should not be registered. This would make it difficult to meet the set-up times of the flip-flops that generate the Qualified Read and Qualified Write signals.



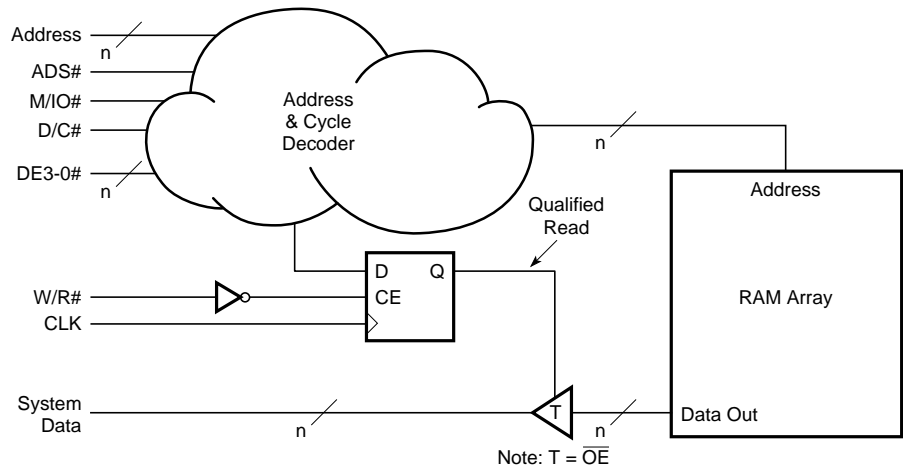
X3103

Figure 10. General Form of an Asynchronous RAM Interface



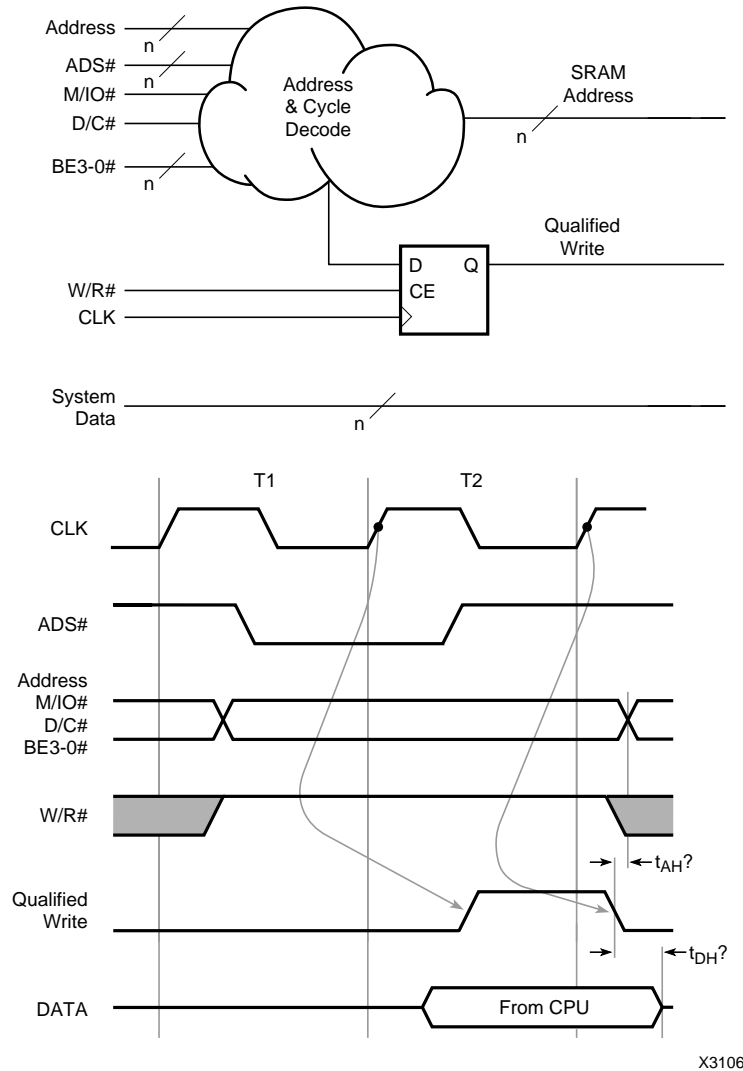
X3104

Figure 11. Typical Microprocessor Read and Write Cycles



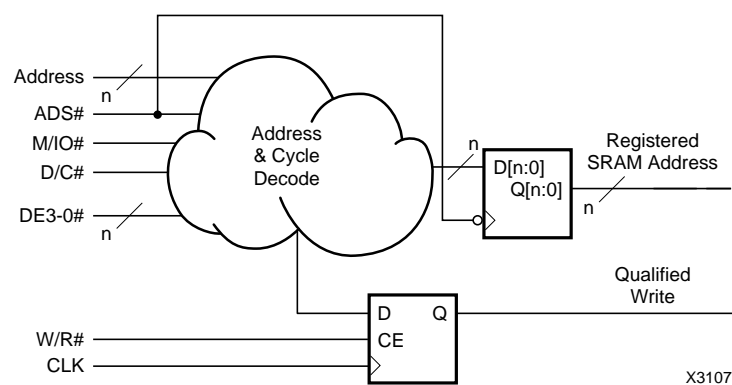
X3105

Figure 12. Implementation of the Read Cycle Logic for an Asynchronous Interface



X3106

Figure 13. Write Cycle Control Logic and Timing



X3107

Figure 14. Modified Write Cycle Logic

Figure 15 shows a 64-bit shift register implemented using RAM. A flip-flop-based 64-bit shift register would use all the flip-flops in 32 CLBs; the RAM-based version can be implemented in only 9 CLBs, a considerable saving of resources. Essentially, the shift register is implemented as a simple circular FIFO that is 1-bit wide and 64-bits

The core of this design is a small sequencer that includes the circuit shown in Figure 16. This circuit, when triggered, generates a sequence of four glitch-free pulses corresponding to four successive half periods of the clock, Figure 17. These pulses are used to control the sequence of events required for a shift cycle. The complete waveform diagram is shown in Figure 18.

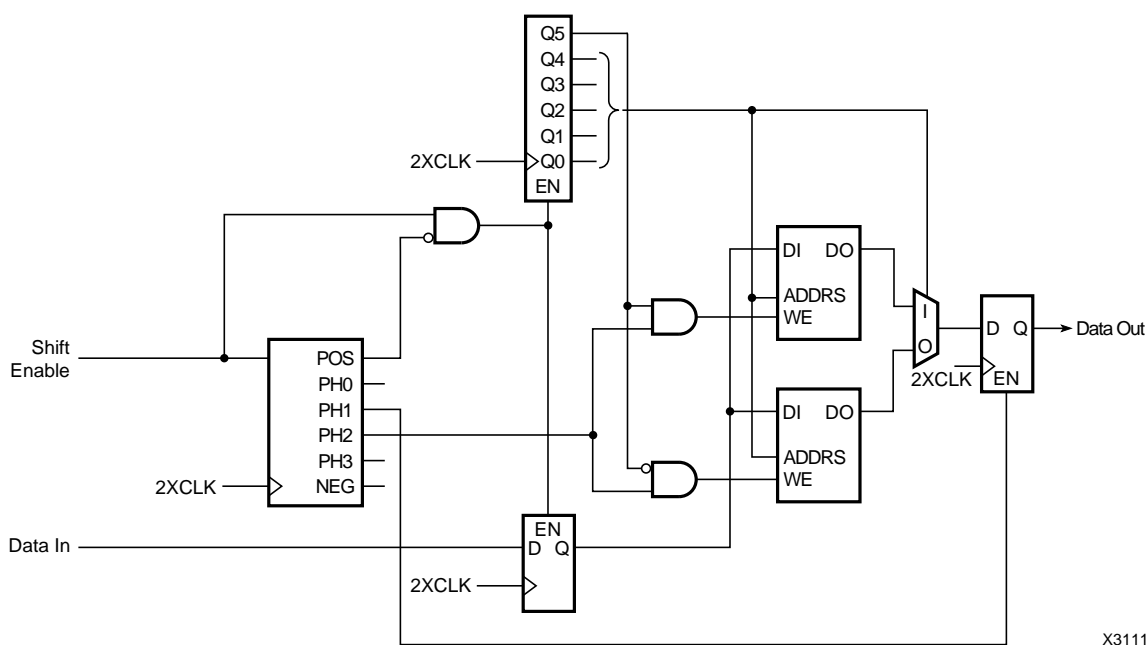


Figure 15. 64-Bit RAM-Based Shift Register

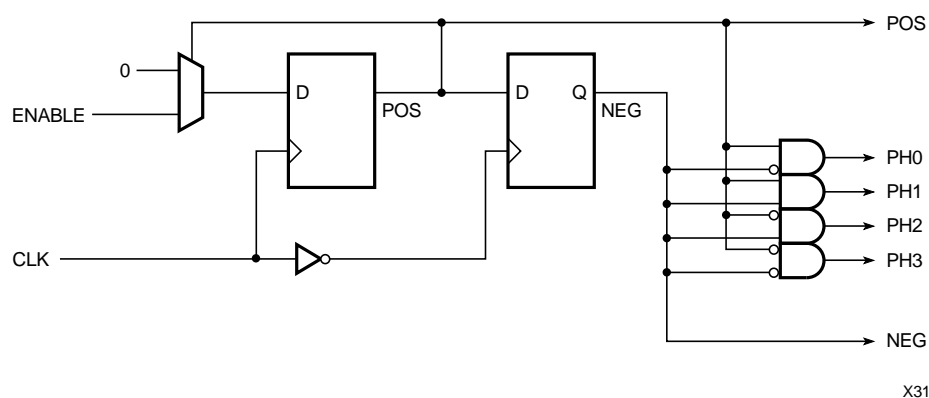
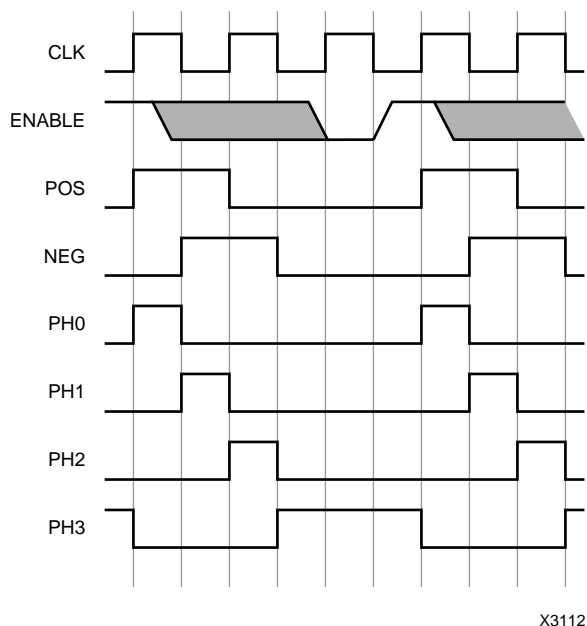


Figure 16. Glitch-Free Sequencer

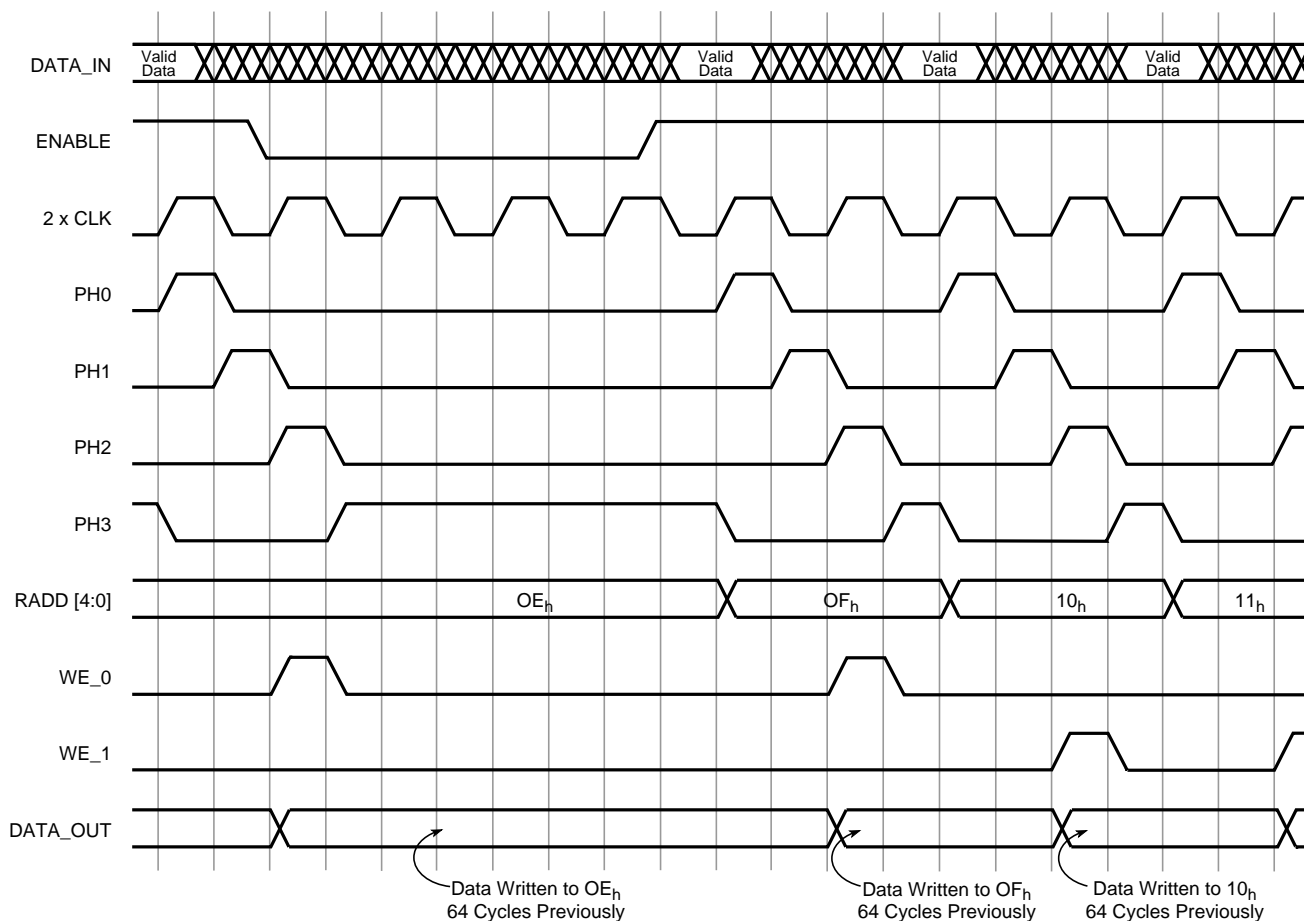


X3112

Figure 17. Waveforms for the Glitch-Free Sequencer

The important events are as follows.

1. A shift cycle is initiated on the rising edge of the 2 X CLK by asserting Enable High. At this time, the pulse sequencer is triggered, the input data is captured into a register and the address counter is incremented. This action may occur on any rising clock edge, but is ignored on the rising edge immediately following a trigger.
2. The data written to the RAM 64 clocks previously is read, and is captured into an output register on the rising clock edge that initiates PH2. Both data and address have had a full 2XCLK period to set up. The 0 ns hold time requirement of the CLB is guaranteed, since the data is stable until the WE pulse.
3. New data is entered into the RAM by the WE pulse, which is PH2 delayed by logic and routing.
4. Address and data cannot change until the end of PH3. At least half a period of the 2XCLK is available for to remove of WE and satisfy the address and data hold-time requirements.



X3108

Figure 18. Waveform for Several Shift Cycles of the 64-Bit Shift Register

As can be seen, the pulse circuit allows the orderly sequencing of the write cycle spacing out the events so that timing requirements can be satisfied. This type of sequencing is the preferred technique in synchronous RAM applications. Its advantage is that it is bulletproof; its disadvantage is that it requires a clock that is twice as fast as the cycle time.

The clock does not necessarily need a 50% duty cycle. In the shift-register example, the only duty-cycle restrictions are that the clock High time must generate an adequate WE pulse, and the clock Low time must allow the WE pulse to be removed with sufficient margin to meet the necessary hold times. Within these restrictions, an asymmetrical clock might even be beneficial, providing faster operation.

The Last Resort.

This last solution to the problem is not a nice one, but it works – most of the time. While its operation is not guaranteed by device characterization, the solution almost invariably works at room temperature, with nominal power supplies on typical parts. However, the probability of failure increases as the restrictions are relaxed.

The use of this method in a production design is particularly risky. While it will probably work reliably, occasional failures must be expected due to parts that are close to their specification limit. Additionally, to avoid field failures, every unit should be tested over the full range of temperature and voltage that it is expected to encounter.

Contrary to the advice given earlier, this solution uses an asynchronous circuit to generate a WE pulse, Figure 19.

In previous sections, this circuit would have been referred to as a glitch generator, but here it is a pulse generator; that is why it is the last resort!

Using this circuit, the only signal that is needed to perform a write to the RAM is a 1x clock at the RAM cycle rate. The leading edge of this clock sets the data and address, while the trailing edge triggers the WE pulse. The restrictions on the clock are that the address and data must set up during the first half of the clock. The second half of the clock must guarantee the WE pulse time to complete, *at the RAM*, with adequate margin to meet the address and data hold-time requirements.

The pulse-generator circuit is a self-resetting flip-flop. The worst-case loop time is >17 ns on an XC4000-5 device ($2 \times t_{ILO} + t_{RIO} + \text{Routing}$). On the same device, the WE pulse requirement of the RAM is 4 ns minimum. Within a single FPGA, the speed of different logic resources tracks reasonably well (to within 70%). Consequently, the worst-case scenario is the WE pulse width decreasing to 12 ns, while the RAM continues to require a 4 ns pulse. In a faster device, with higher V_{CC} or at a lower temperature, the width of the WE pulse will decrease; but so will the WE requirement of the RAM. As a result, the pulse width should never fail to satisfy the WE requirement.

For more reliable timing, this circuit could be converted to a hard macro in a single CLB. It could then be instantiated in the design as required.

Please see the Application Note, *High-Speed RAM Design in XC4000* (XAPP 042, page 8-139), for a rugged, simple and elegant high-speed RAM design.

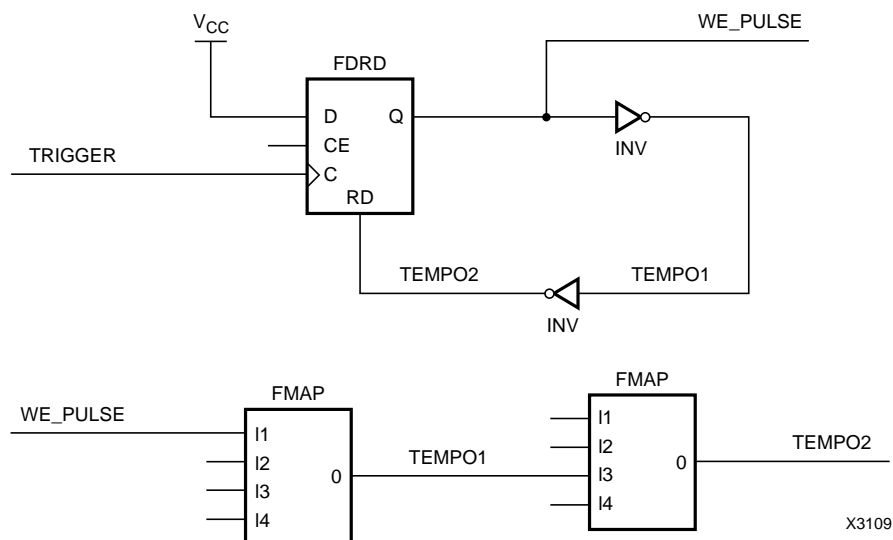


Figure 19. Pulse Generator