Interfacing XC6200 To Microprocessors (TMS320C50 Example)

XAPP 064 October 9, 1996 (Version 1.1)

Application Note by Bill Wilkie

Summary

The issues involved in interfacing XC6200 family members to microprocessors are discussed. An example using the Motorola 68020 processor is described.

Xilinx Family	***Demonstrates***
XC6200	Parallel programming interface.
	Driving device control inputs from within the FPGA core array.

Overview

This note explains the key issues involved in interfacing XC6200 parts to microprocessors by looking at a specific example. The XC6200 parallel interface signals are described and related to the signals provided by a widely used microprocessor. Different ways of implementing the design are discussed, including a method which uses no glue logic between the two devices. More details on XC6200 features can be found in the Xilinx data sheet for the XC6200 family.

The sections covered in this note are:

Introduction

One of the key features of the XC6200 family is the easy access to all the device memory and array logic. The fastest way of configuring the chip is via the parallel programming interface. This allows a CPU to directly write the entire device configuration and also modify the state of registers within logic cells. Having full 32-bit access to the configuration RAM makes dynamic reconfiguration a real possibility.

The 32-bit interface also means users can read or write the state of columns of 32 cells within the array simultaneously.

Although the data bus is 32 bits wide, it is also straightforward to interface to 8 and 16-bit microprocessors.

The CPU interface is internally governed by a device control register. This determines the width of the data bus. This is initially 8 bits but can be modified to 16 or 32 bits. In the case of an 8-bit interface, only bits *D<7:0>* are of relevance. Other data bus bits will not be driven during reads.

The interface signals provided make it fairly straightforward to interface to any microprocessor. In this example a Texas Instruments TMS320C50 microprocessor is used as this is a commonly used processor in DSP applications.

XC6200 Requirements

XC6200 provides the following signals specifically for parallel access:

$\overline{CS}$

Chip Select enables the programming circuitry and initiates address decoding. When *$\overline{CS}$* is low data can be read from or written to the control memory. This signal is intended to be used in conjunction with address decoding circuitry to select one part within a larger array for programming.

D<d:0>

(*d*+1)-bit bidirectional data bus. Used for device configuration and direct cell register access.

A<a:0>

Address bus for CPU access of internal registers and configuration memory. '*a*' varies between family members.

Rd$\overline{Wr}$

When *$\overline{CS}$* is low this signal determines whether data is read from or written to the control memory. If *Rd$\overline{Wr}$* is high then a read cycle takes place. If *Rd$\overline{Wr}$* is low then a write cycle takes place.

The full 32-bit data bus is not available on all device package options due to pin limitations. The width of the address bus varies between XC6200 family members. The XC6216 is used as an example. In this case the address bus is 16 bits, *A<15:0>*. This part will take up the entire data address space of some processors which only have a 16-bit address bus. If this is not desired then paging registers can be implemented in the address decoding to allow as many peripheral devices as required.

The CPU interface is synchronized by the *GClk* signal. This clock controls all the RAM and register interface circuitry within the XC6200 device.

XC6200 write and read cycles are shown in Figures 1 and 2. The a.c. parameter numbers are the XC6200 data sheet references.

$\overline{CS}$ is normally high. The XC6200 continually samples *$\overline{CS}$* on the rising edge of *GClk*. All the other CPU interface signals are also sampled on the rising edge of *GClk*. The set up and hold times specified in the XC6200 data sheet must be met.

Once the XC6200 detects that *$\overline{CS}$* has gone low a parallel CPU cycle begins.This is time t_1 in Figures 1 and 2. The type of cycle (read or write) is determined by the value of *Rd$\overline{Wr}$* sampled at time t_1. The address bus is also sampled at this time. What happens next depends on whether it is a read or write cycle.

Write Cycle

The data bus is sampled at t_1. *Rd$\overline{Wr}$* is sampled low at t_1. After t_1 the address and data busses are ignored. The write cycle now takes place inside the XC6200 device. The cycle is split into 4 phases. During phase 1 the XC6200 decodes the address. The memory location addressed is written with the data value captured at t_1 during phase 2. By phase 4 the device is ready for another cycle to start.

If *$\overline{CS}$* is still low at time t_2 the cycle is extended. The internal write still occurs during phase 2. *$\overline{CS}$* must be sampled high before phase 4 can be entered and the cycle terminated.

Read Cycle

Rd$\overline{Wr}$ is sampled high at t_1. After t_1 the address bus is ignored. The address is decoded during phase 1 and the memory location addressed is internally read during phase 2. The data is then driven onto the data bus during phase 3 - after t_2 in Figure 2.

If *$\overline{CS}$* is sampled high at time t_2, phase 4 will be entered and the cycle will terminate. The data bus will enter a high impedance state after t_3. Another read cycle may be started by driving *$\overline{CS}$* low during phase 4. An additional clock cycle must be inserted before a write cycle can be started as the data bus cannot be driven until the read cycle data is removed from the bus, t_{CKDZ} after t_3. Thus an interface optimized for fast state reading could perform a burst of reads with only two clock cycles per read and no waiting between reads.

If the processor requires the data to be held on the data bus for longer than one clock cycle, *$\overline{CS}$* must be held low until it is safe for the data bus to enter the high impedance state. There are two possible situations here:

1) *$\overline{CS}$* is sampled low at t_2 and high at t_3. The data bus enters the high impedance state t_{CKDZ} after t_3. Another read cycle cannot begin until the next rising *GClk* edge.Thus in this case the data is still only present on the bus for one clock cycle but an extra clock cycle must be inserted between consecutive reads.

2) *$\overline{CS}$* is sampled low at t_2 and t_3. In this case the cycle is extended. It is the rising edge of *$\overline{CS}$* which

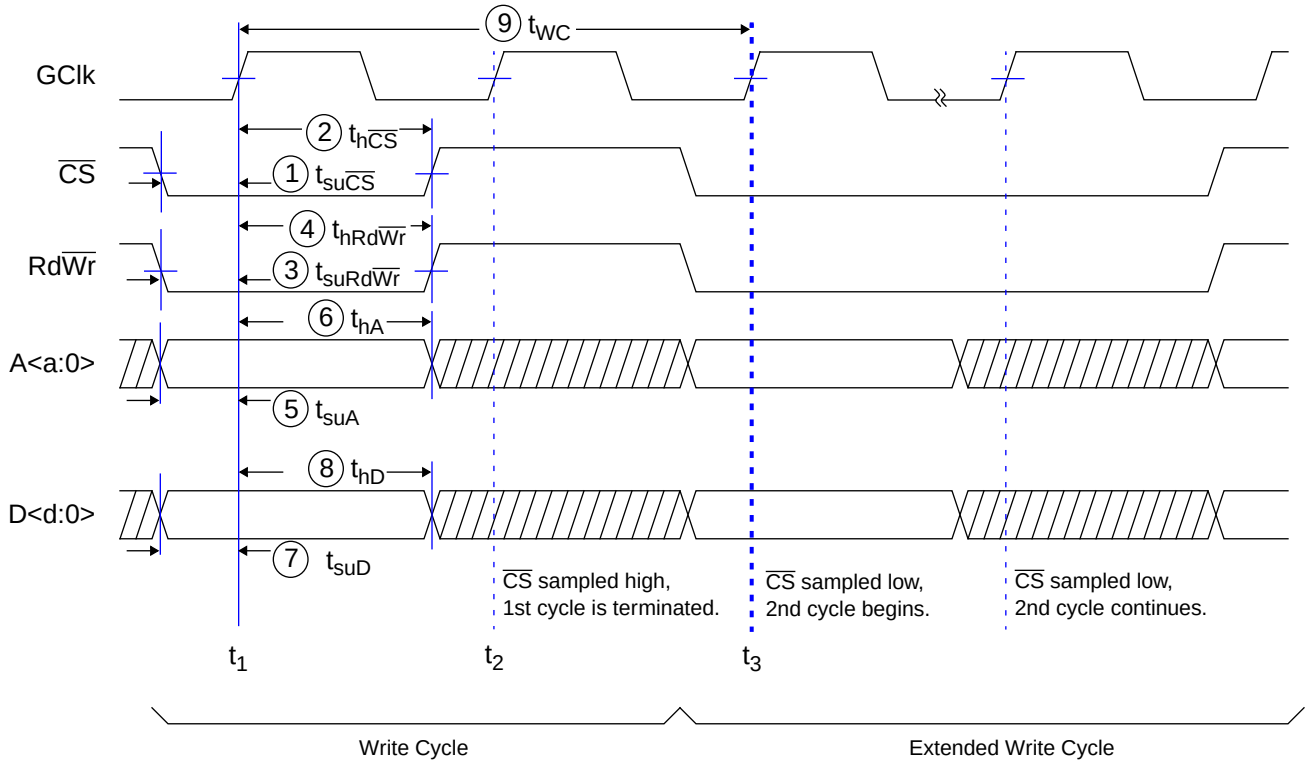

Figure 1. XC6200 Write Cycles

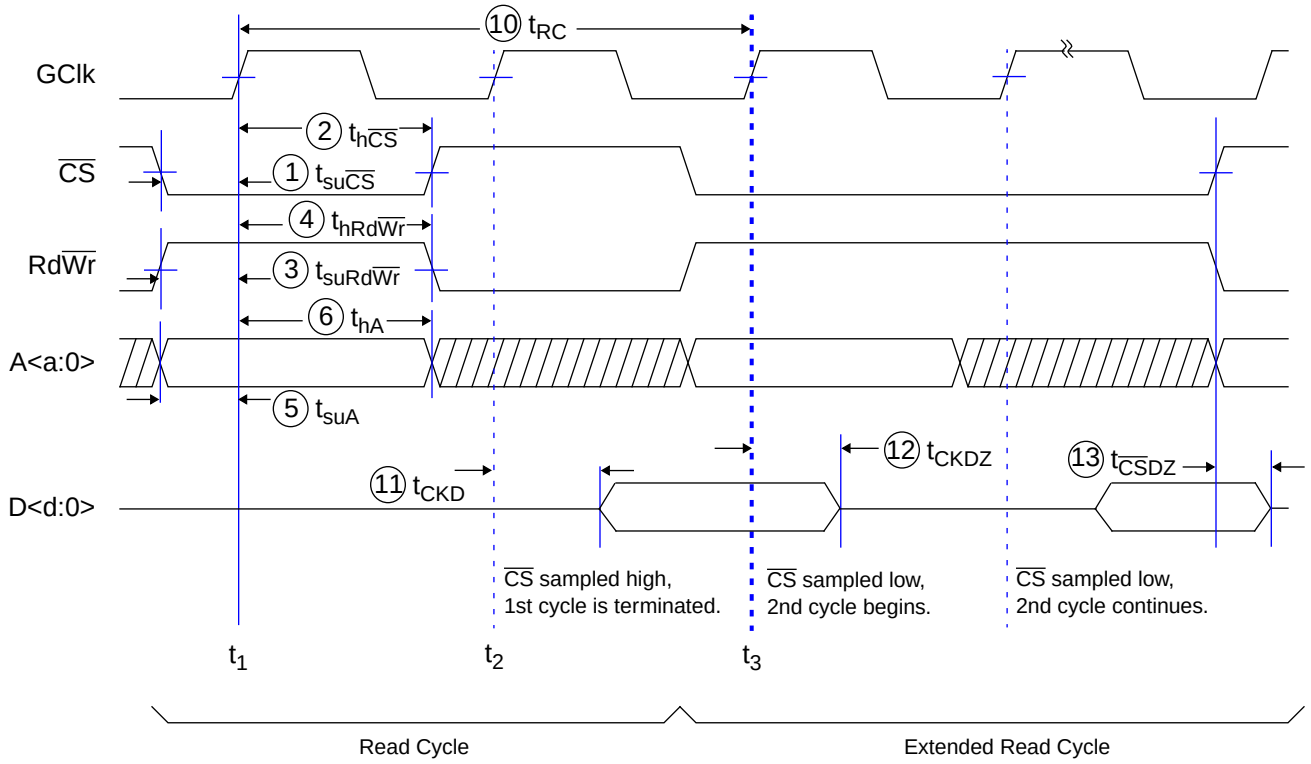

Figure 2. XC6200 Read Cycles

causes the XC6200 to switch off its data bus drivers. This will happen $t_{\overline{CS}DZ}$ after $\overline{CS}$ goes high. Note that this is slightly different from the normal short read cycle, where it is the rising edge of *GClk* at t_3 which switches off the bus.

In all cases $\overline{CS}$ must be sampled high on a rising edge of *GClk* to terminate the cycle.

TMS320C50 Requirements

The TMS320C50 is a 16-bit DSP oriented processor. It has 3 separate 16-bit address spaces. The 16-bit address bus is qualified with 3 strobe signals to distinguish program, data and I/O port accesses. The XC6216 could be mapped to the data or I/O spaces. Each address references a 16-bit location. Thus a single XC6216 would take up half of the available data or I/O address space. If more address space was required paging registers could be used.

The processor uses the following major signals to communicate with peripheral devices:

D<15:0>

16-bit bidirectional data bus.

A<15:0>

16-bit address bus output.

R/$\overline{W}$

Output which determines whether a bus cycle is a read or a write cycle. A high level indicates a read cycle. A low level indicates a write.

$\overline{WE}$

Write Enable output has suitable timing to be used as a write pulse for asynchronous devices such as RAM and latches. Data may be latched in the external device on the rising edge of $\overline{WE}$.

$\overline{PS}$,$\overline{DS}$,$\overline{IS}$

Strobe outputs which indicate the address space to which an external bus cycle is to be applied - program, data or I/O.

$\overline{Strb}$

Strobe output which indicates an external bus cycle.

$\overline{Ready}$

Input which can be used to insert wait states in CPU cycles.

ClkOut1

Master clock output. Cycles at the machine cycle rate of the CPU.

The processor also has some additional interface signals which are not used in this example.

If wait states are required they can be inserted automatically by the processor or the *Ready* signal can be used.

When the XC6200 is in 8-bit mode (as it is initially) it will use the 8 least significant bits of the data bus to transfer information (*D<7:0>*). To make the most of the processor's 16-bit data bus, the XC6200 device control register should be modified to 16-bit mode.

Example Circuit

The example shown here uses the TMS320C50 *ClkOut1* signal to clock the FPGA. One wait state is required during read cycles. If this is generated using the processor software wait state generator then no wait state circuitry is required.

A basic example circuit is shown in Figure 3. This example places the XC6216 in the processor's I/O memory space. The XC6216 takes up the lower half of the I/O address space.

A correctly timed $\overline{CS}$ pulse is generated from $\overline{IS}$ and $\overline{Strb}$. It is necessary to generate a $\overline{CS}$ pulse because during consecutive read cycles the strobe signals remain low throughout the burst. The FPGA needs to detect the high to low and low to high transitions on $\overline{CS}$ to start and terminate CPU cycles. The example circuit will generate a single clock cycle $\overline{CS}$ pulse at the start of each processor read or write to the FPGA. Depending on the propagation delays of the logic elements used, a glitch may be generated on $\overline{CS}$ at the end of the last read cycle in a sequence. This does not matter to the FPGA as $\overline{CS}$ is sampled synchronously on the rising edge of *GClk*. A more complicated pulse generation circuit could be used if this was a problem for other circuitry.

The *Ready* input is not shown as one wait state is automatically inserted by the processor. This is not necessary during write cycles but the software wait state generator generates the same number of wait states for both reads and writes. If 0-wait state write

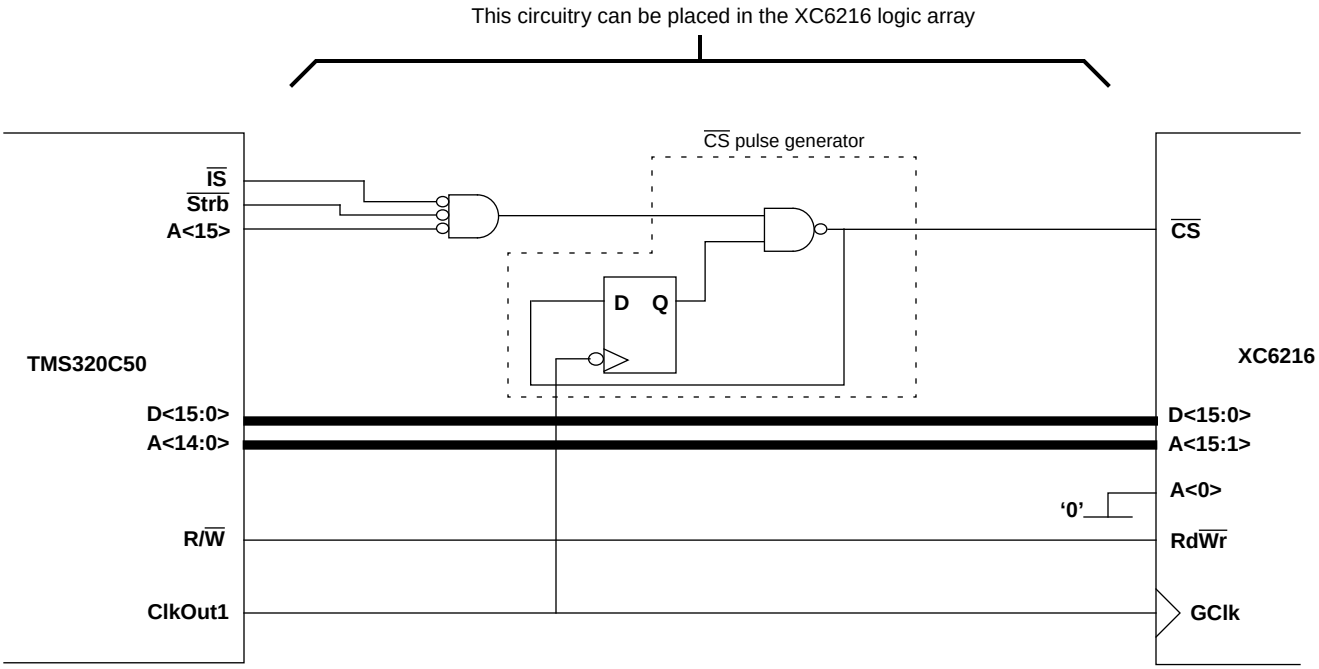

Figure 3. Basic Example Circuit

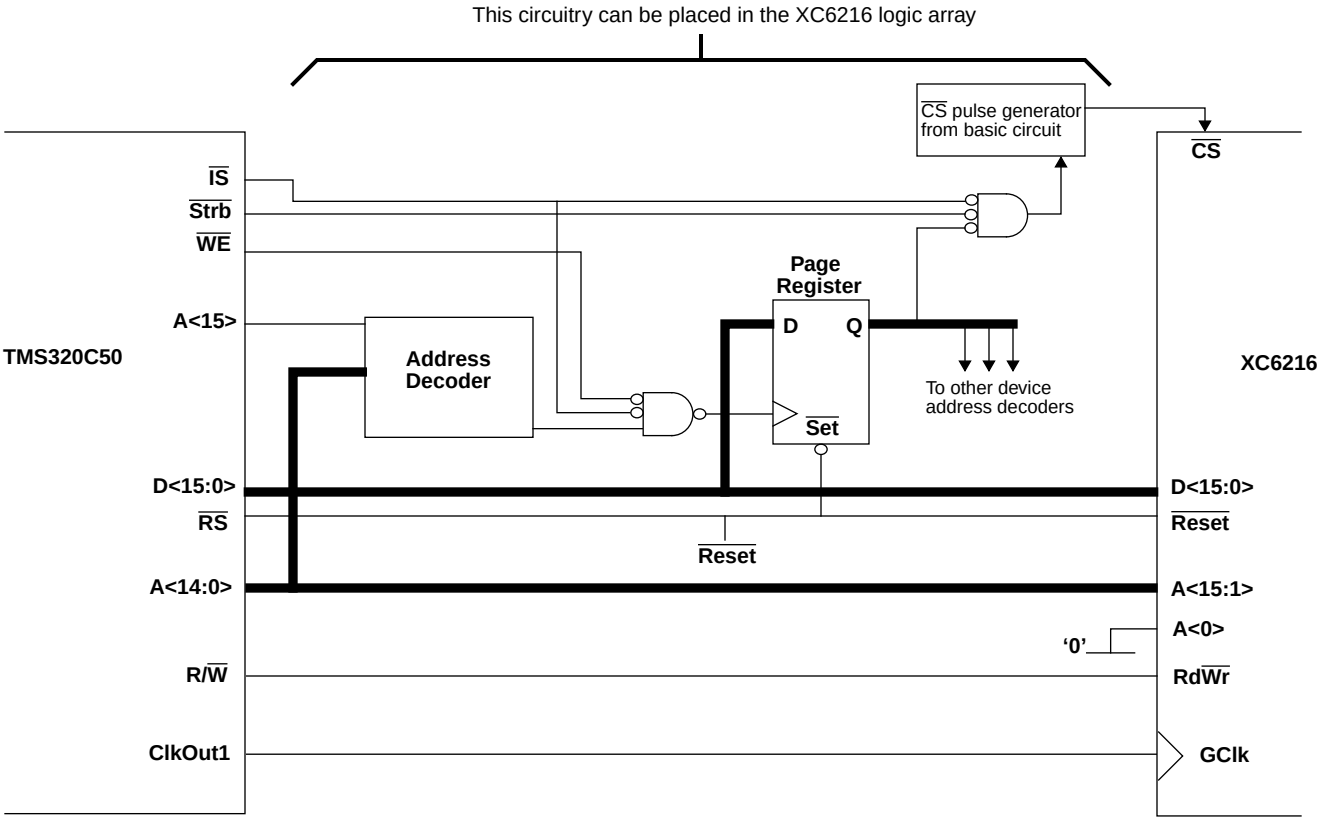

Figure 4. Example Circuit With Paging Register

cycles were required the *Ready* signal would have to be used to generate the read cycle wait state. An example circuit to do this is shown in Figure 7.

Each TMS320C50 address accesses two 8-bit XC6216 locations. Hence *A<0>* on the XC6216 is grounded. The FPGA configuration register must be set to 16-bit data bus width. The first write cycle to the configuration register will be an 8-bit cycle as far as the XC6216 is concerned, however the configuration register is located at an even address so it does not matter that *A<0>* is tied to ground.

Figure 4 shows a modified circuit with paging registers which allow more devices to be located within the available address space. In this example the paging register and the XC6216 are located in the I/O memory space. The page register is written when a processor I/O space write cycle occurs to the address decoded by the address decoder block. The complexity of the address decoder will depend on how many separate devices are to be directly written in the I/O space. The XC6216 $\overline{CS}$ can only be asserted if the paging register has been written with an appropriate value first. The paging register may be any width up to 16 bits. The output from the register could be further decoded or used directly, as in the example here.

These designs assume $\overline{CS}$ and the data bus will meet all the relevant set up and hold times. This is dependant on the gate delays and the clock speed used. If these times are not met then $\overline{CS}$ can be retimed and more wait states added.

Timing

Timing diagrams for the example circuit of Figure 7 are shown in Figures 5 and 6. It is assumed that there will be one wait state during read cycles and no wait states during write cycles.

Write Cycle

At the start of a write cycle the processor drives *A<15:0>* with the address to be written. *R/*$\overline{W}$ is driven low and the appropriate address space strobe is asserted ($\overline{IS}$, $\overline{DS}$ or $\overline{PS}$). In the example here it is assumed that the FPGA is located in the I/O address space.

$\overline{Strb}$ goes low on the falling edge of *ClkOut1*. In the example circuit this causes $\overline{CS}$ to be asserted. It is assumed that no wait states are required and *Ready* is always high during writes. If the FPGA timing requirements for the set up time of $\overline{CS}$ and the data bus cannot be met due to a very fast clock or large propagation delays in the decoding, $\overline{CS}$ would have to be retimed and a wait state added in the same way as for a read cycle.

The XC6216 samples $\overline{CS}$ low at t_1. This starts the write cycle inside the XC6216. The address and data busses and *Rd*$\overline{Wr}$ pin are also sampled at t_1.

SInce there is no wait state, the processor deasserts $\overline{Strb}$ on the next falling edge of *ClkOut1*. This causes $\overline{CS}$ to be deasserted. In the case of write cycles $\overline{Strb}$ is always deasserted between cycles. Thus the $\overline{CS}$ pulse generation circuit is not required here. However it is required for read cycles.

The XC6216 samples $\overline{CS}$ high at t_2 and terminates the cycle. The internal XC6216 write actually completes during the first clock cycle of the next CPU cycle, however this does not matter as the FPGA will be ready to sample $\overline{CS}$ again on the rising edge of *ClkOut1* and start a new cycle at time t_3 if required. The internal XC6216 write cycle takes place between t_1 and t_3 in Figure 5. Times t_1 to t_3 in Figure 5 correspond to t_1 to t_3 in Figure 1.

The processor holds the address and data busses stable during the cycle but this is not important because the XC6216 samples at t_1 and the bus values are irrelevant after this time.

The $\overline{WE}$ signal is shown as this is used as a convenient way of clocking the page register in Figure 4. Care must be taken when designing a paging scheme to ensure that a spurious write to the FPGA does not occur immediately after the page register has been activated. This will not happen in the example shown here as $\overline{Strb}$ has been deasserted by the time the paging register has been written so a spurious $\overline{CS}$ pulse will not be generated.

Read Cycle

A standard TMS320C50 read cycle takes only a single *ClkOut1* cycle. An XC6216 read cycle requires two clock cycles therefore a wait state is required.

The processor outputs the address to be read and drives *R/*$\overline{W}$ high to signify a read cycle. The appropriate address space strobe signal ($\overline{IS}$, $\overline{DS}$ or $\overline{PS}$) is also asserted. $\overline{Strb}$ is asserted on the falling *ClkOut1* edge, causing a $\overline{CS}$ pulse to be generated. The $\overline{CS}$ pulse is used to generate a zero on the processor's *Ready* input. *Ready* is sampled low by the processor at time t_1, signalling a wait state.

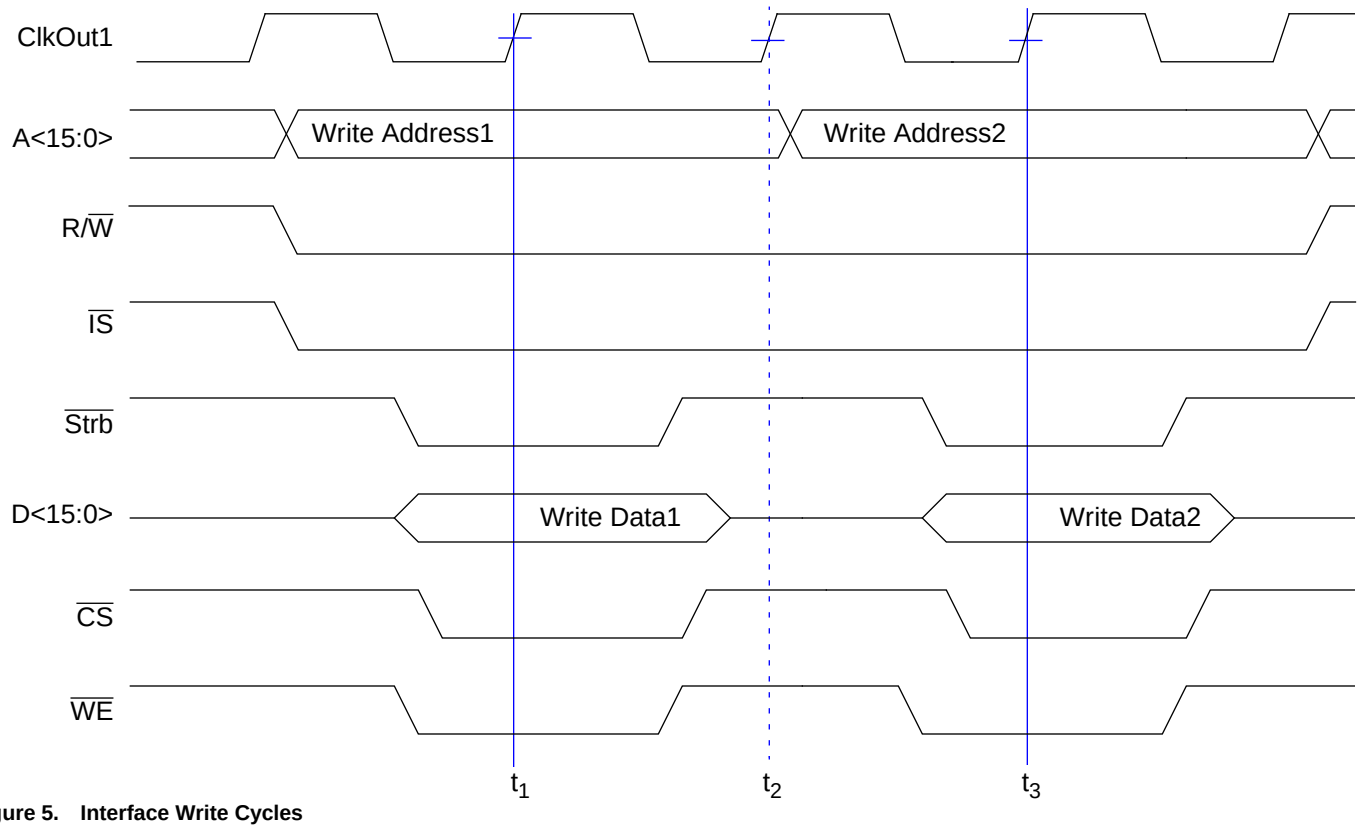

Figure 5. Interface Write Cycles

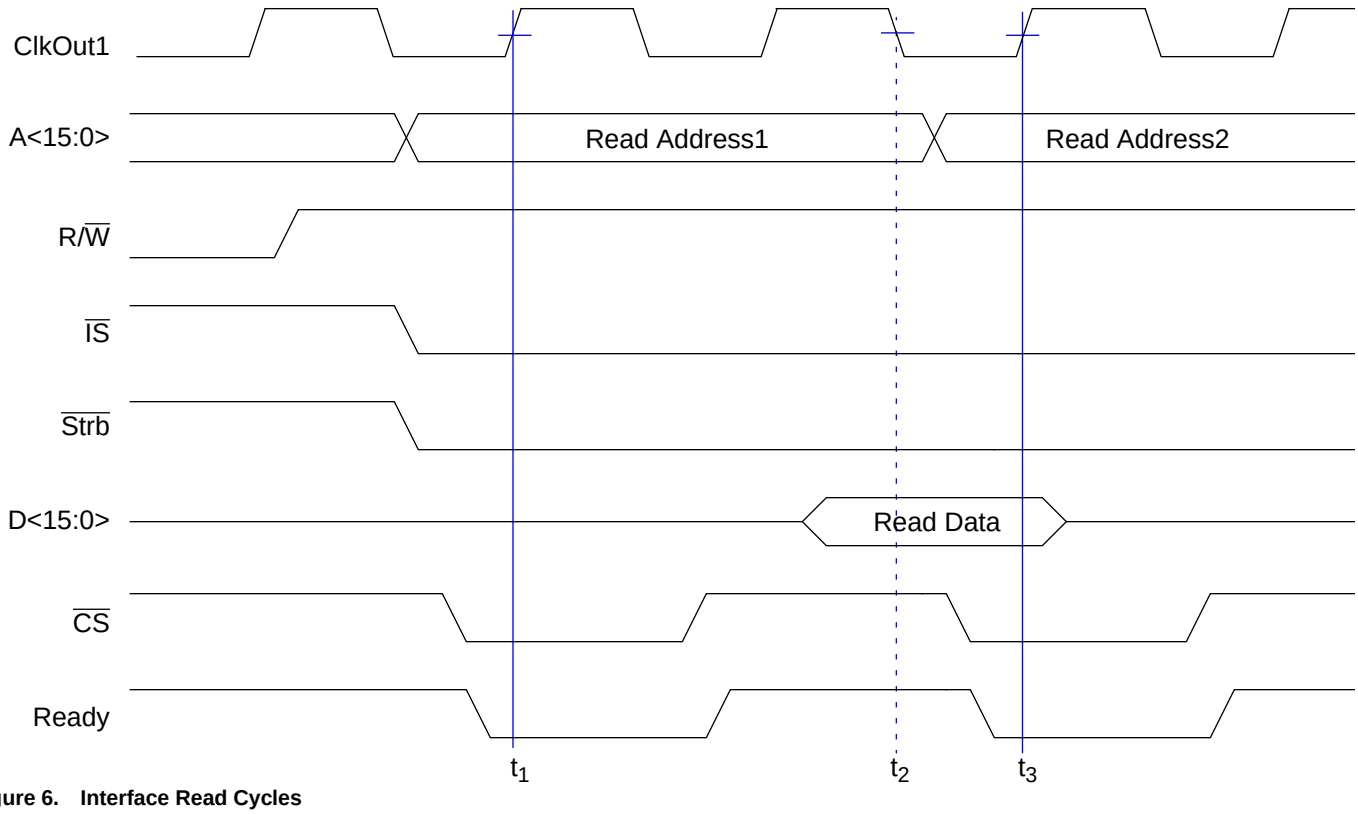

Figure 6. Interface Read Cycles

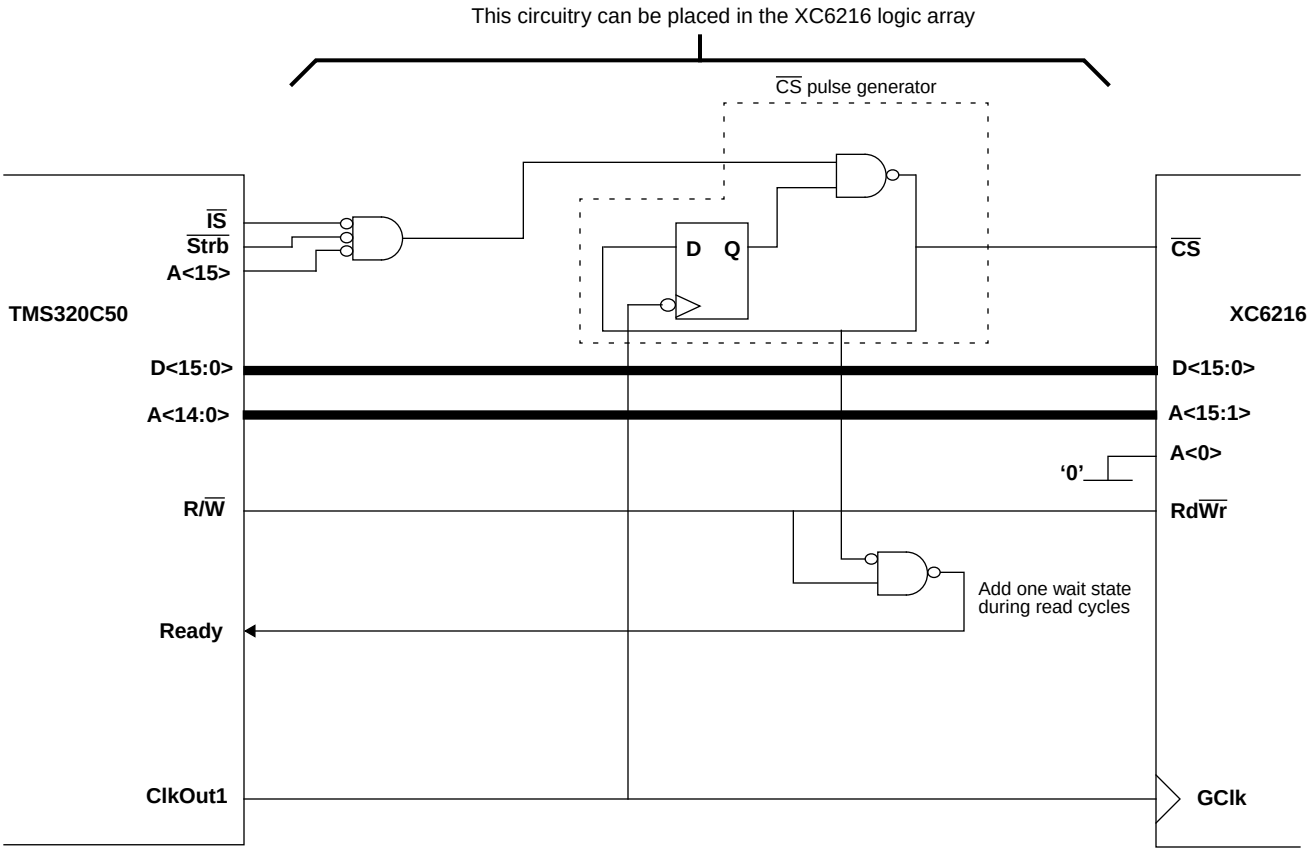

Figure 7. Example Circuit With External Wait State Generator

The XC6216 samples *CS* low at t_1. This starts the read cycle inside the XC6216. The address bus and *RdWr* pin are also sampled at t_1.

The XC6216 performs its internal read and drives the data bus for a clock cycle following the next rising edge on *ClkOut1*. The FPGA also samples *CS* high at this time and terminates the cycle.

The processor samples the data around time t_2. The internal XC6216 read cycle takes place between t_1 and t_3. Times t_1 and t_3 in Figure 6 correspond to t_1 and t_3 in Figure 2.

If there are no more read cycles the processor deasserts *Strb* at t_2. If there are more read cycles *Strb* remains asserted and another *CS* pulse is produced by the pulse generator circuit.

The processor holds the address bus stable during the cycle but this is not important because the XC6216 samples at t_1 and the bus values are irrelevant after this time.

Implementation

All the logic between the processor and the FPGA in Figures 3, 4 and 7 could be implemented in a small EPLD on the board. This may be the best option if a very fast clock is being used.

Another possibility makes use of the XC6200 family's ability to drive its own control inputs from user logic within the programmable array. This is fully described in the XC6200 family data sheet. In this case the configuration for the interface circuit is stored in a Xilinx serial PROM. On power up this is serially loaded into the FPGA. The FPGA is configured so that its *CS* input pin is driven from the output of the interface circuit within the logic array rather than from external circuitry.

Using this method, the interface circuit could easily be expanded to provide all the timing and glue logic for an entire board. Minimal circuitry would be loaded serially from the PROM to allow the microprocessor to complete the process in fast parallel mode.

Summary

XC6200 parallel interface gives fast access to internal configuration and logic state data.

Parallel interface gives user total control over all registers in logic design.

XC6200 is easily interfaced.

Interface circuitry can be implemented in XC6200 array itself, booted from PROM.

The techniques shown here can be easily adapted to any 8, 16 or 32-bit microprocessor.

Limitations And Restrictions