Interfacing XC6200 To Microprocessors (MC68020 Example)

XAPP 063 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 Motorola MC68020 is used as this is a well known and mature device.

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

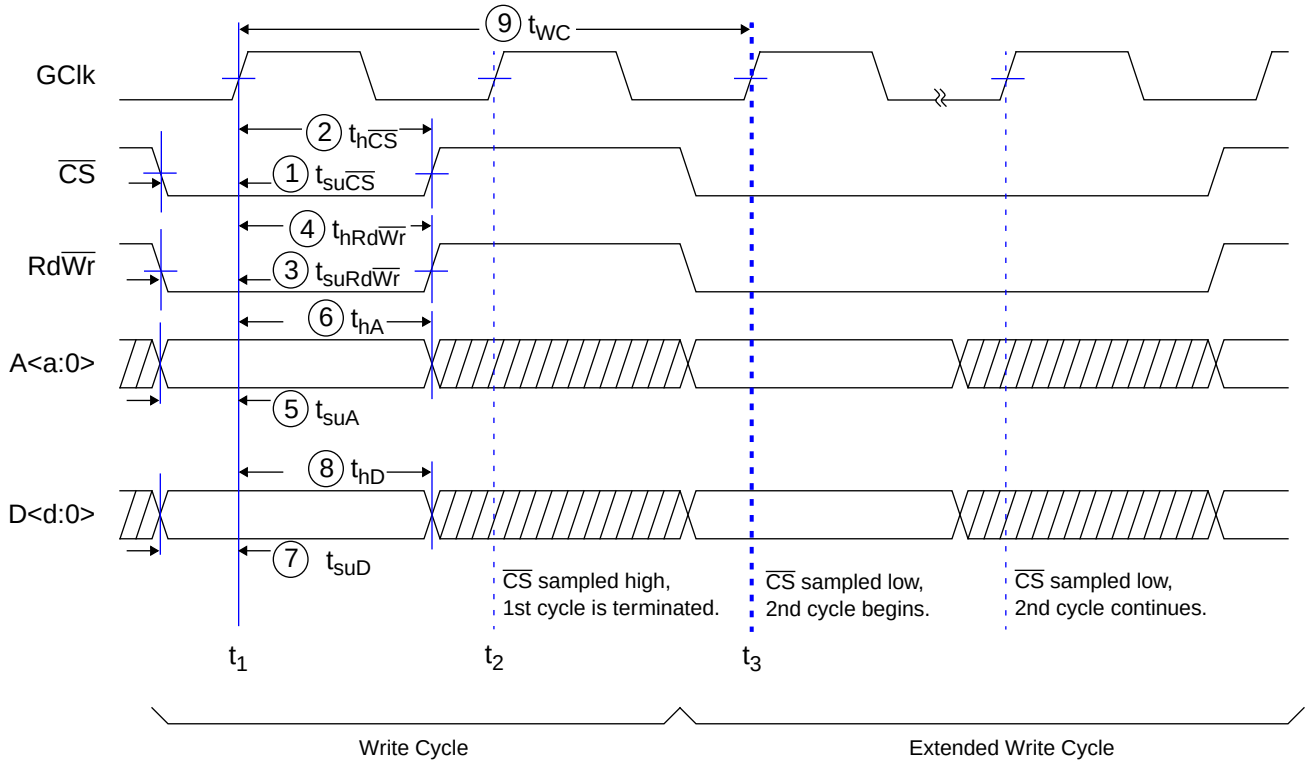

Figure 1. XC6200 Write Cycles

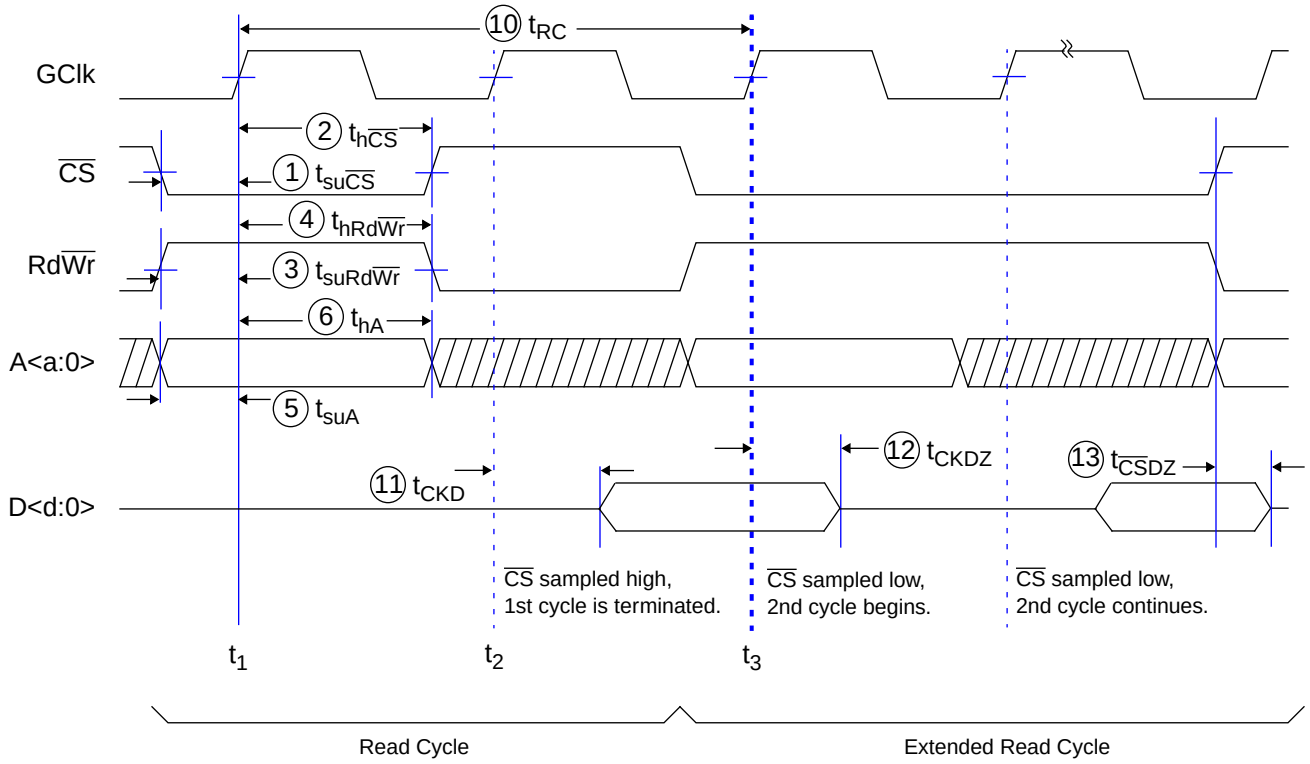

Figure 2. XC6200 Read Cycles

causes the XC6200 to switch off its data bus drivers. This will happen $t_{\overline{CS}DZ}$ after *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 *CS* must be sampled high on a rising edge of *GClk* to terminate the cycle.

MC68020 Requirements

The MC68020 can make full use of the XC6200 32-bit parallel interface as it also possesses a 32-bit data bus. It also possesses a 32-bit address bus which means that the XC6216 16-bit address bus can easily be accommodated.

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

D<31:0>

32-bit bidirectional data bus.

A<31:0>

32-bit address bus output.

FC<2:0>

Function Code outputs which further qualifies the address bus output. e.g. indicates user data space, program space, etc. These codes have to be decoded just as if they were additional address bus bits.

R/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.

AS

Address Strobe output which indicates that a valid address is present on the bus during read and write cycles.

DS

Data Strobe output which indicates that valid data is present on the bus during a write cycle. Indicates that valid data should be placed on the bus during a read cycle.

DSACK<1:0>

Data Transfer and Size Acknowledge inputs. The bus cycle does not terminate until these signals have been asserted. The 2-bit binary code informs the processor of the addressed peripheral's bus width.

The processor also has some additional interface signals which are not used in this example. The processor is capable of dynamic bus sizing. Here it is assumed that all transfers will be 32-bit. 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>*). If a byte instruction is executed the MC68020 expects to see 8-bit data on *D<31:24>*. Therefore only long word data transfers should be made, even when the XC6200 is not in 32-bit mode itself. The example here always returns *DSACK* codes for a 32-bit bus.

Example Circuit

A simple example circuit is shown in Figure 3. This example uses *DS* to generate a correctly timed *CS* signal for the XC6200 part. *AS* is used to generate *DSACK* signals to give 0-wait state operation. The exact decoding of the upper address bits will depend on how many other devices are to be addressed by the processor and where the XC6200 part is to appear in the processor's memory. The output of the address decoder must go high when the processor is addressing the XC6200 part. Address bits *A<1:0>* are connected, however, as only long word transfers should occur, these could be tied to ground on the FPGA. This example places the XC6216 in the user data memory space, *FC<2:0>* = 001.

The processor *SIZ<1:0>* signals are not used in this example as, for reasons described above, it is assumed only long word accesses will be made to the FPGA. Similarly a *DSACK<1:0>* code of 00 is always returned to signify a 32-bit port. If other devices are present on the board then the *DSACK* signals from these devices must be OR'ed together.

This design assumes *CS*, *DSACK* 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 *CS* can be retimed and *DSACK* delayed to introduce wait states.

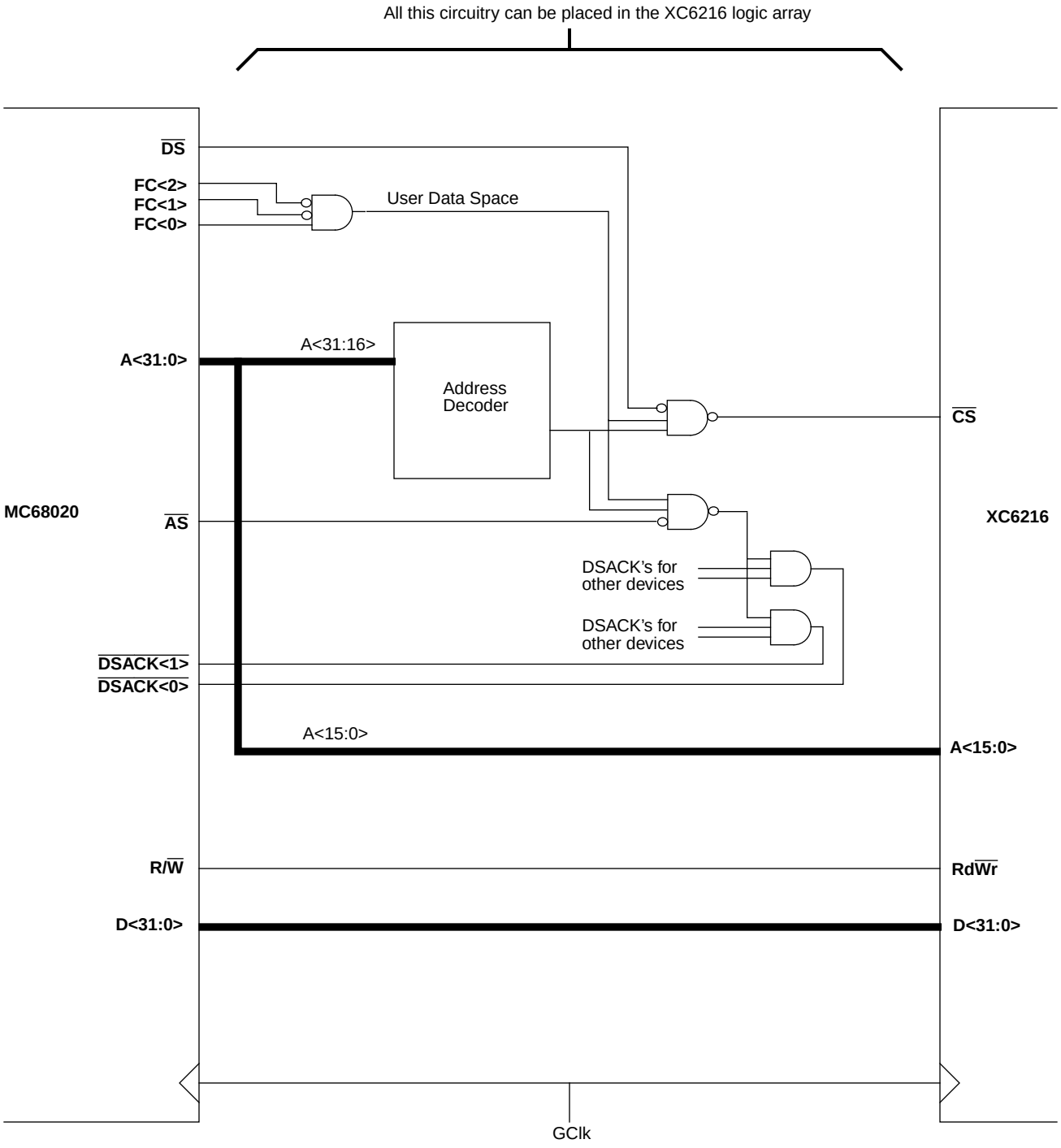

Figure 3. Example Circuit

Timing

Timing diagrams for the example are shown in Figures 4 and 5. The cycles are subdivided into half clock cycle states (S0, S1, etc).

Write Cycle

During S0 the processor outputs the address to be written and also the function codes for the address space (*FC<2:0>*). *R/W* is also driven low at this time to signify a write cycle. If the address is decoded as a XC6216 location then the outputs of the address decoder and the gate decoding *FC<2:0>* will become active.

$\overline{AS}$ is asserted during S1. This signals a valid address and generates the $\overline{DSACK}$ signals for the processor. The $\overline{DSACK}$ signals can be generated here if no wait states are required.

During S2 the data to be written is placed on the data bus. The processor samples $\overline{DSACK}$ at t_1. $\overline{DS}$ is asserted during S3 to indicate valid data on the bus. This also enables the gate driving the XC6216 $\overline{CS}$ input.

The XC6216 samples $\overline{CS}$ low at t_2. This starts the write cycle inside the XC6216. The address and data busses and *Rd*$\overline{Wr}$ pin are also sampled at t_2.

The processor deasserts $\overline{AS}$ and $\overline{DS}$ during S5, causing $\overline{CS}$ to be deasserted. The XC6216 samples $\overline{CS}$ high at t_3 and terminates the cycle. The internal XC6216 write actually completes during S0 and S1 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 S1 and start a new cycle if required. The internal XC6216 write cycle takes place between S4 and S1 of the next processor cycle. S4 corresponds to phase 1 in Figure 1. S1 of the next cycle corresponds to phase 4.

The processor holds the address and data busses stable during S5 but this is not important because the XC6216 samples at t_2 and the bus values are irrelevant after this time.

A new cycle may begin immediately after S5.

Read Cycle

During S0 the processor outputs the address to be read and also the function codes for the address space (*FC<2:0>*). *R/W* is also driven high at this time to signify a read cycle. If the address is decoded as a XC6216 location then the outputs of the address decoder and the gate decoding *FC<2:0>* will become active.

$\overline{AS}$ and $\overline{DS}$ are both asserted during S1. $\overline{AS}$ signals a valid address and generates the $\overline{DSACK}$ signals for the processor. $\overline{DSACK}$ is sampled by the processor at t_2. As with the write cycle, this assumes no wait states. $\overline{DS}$ signals that data should be placed on the data bus. This causes $\overline{CS}$ to be asserted.

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

The XC6216 performs its internal read and drives the data bus during S4. The processor samples the data on the falling edge of S4. The internal XC6216 read cycle takes place between S2 and S5. S2 corresponds to phase 1 in Figure 2. S5 corresponds to phase 4.

The processor deasserts $\overline{AS}$ and $\overline{DS}$ during S5, causing $\overline{CS}$ to be deasserted. The XC6216 samples $\overline{CS}$ high at t_3 and terminates the cycle.

The processor holds the address bus stable during S5 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 Figure 3 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 $\overline{CS}$ 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.

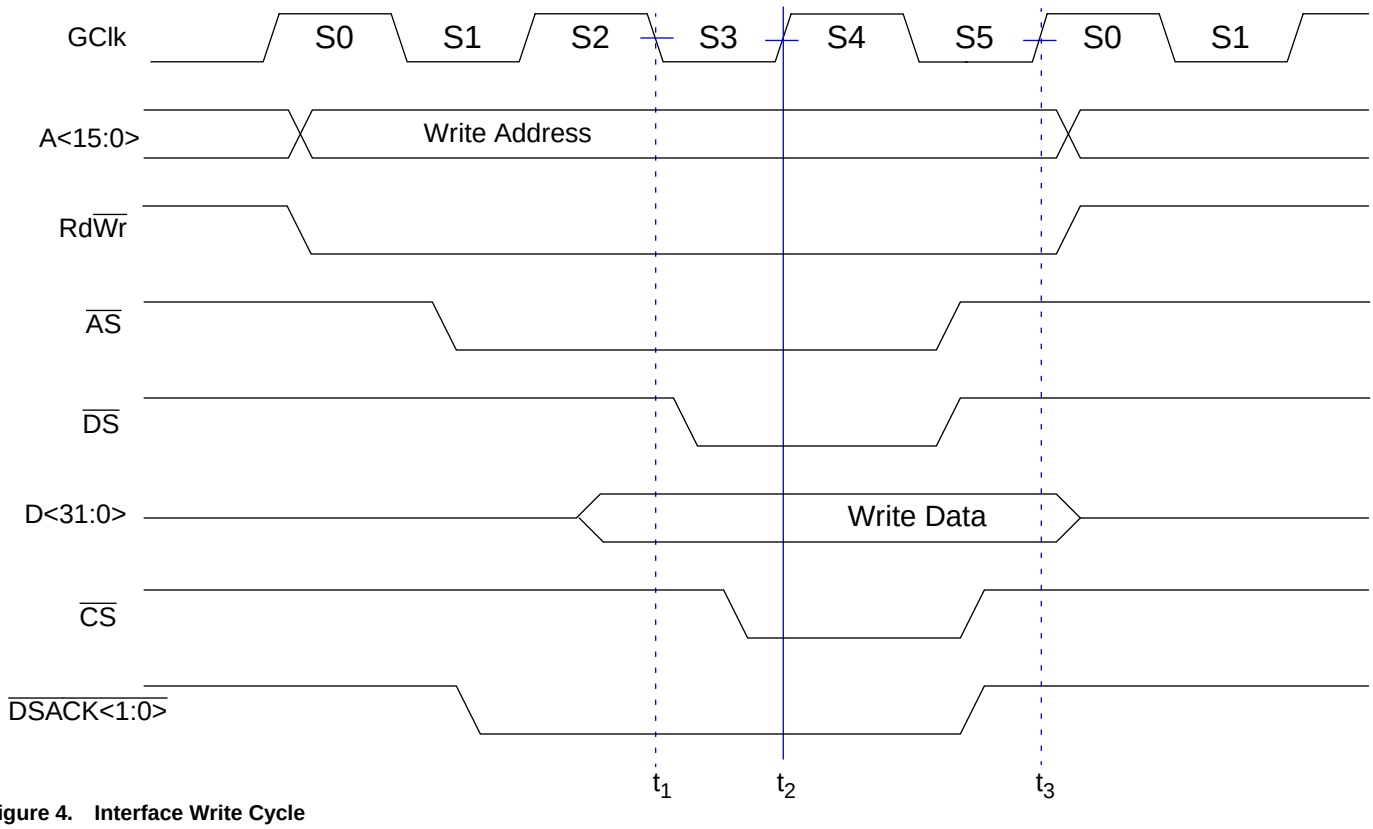

Figure 4. Interface Write Cycle

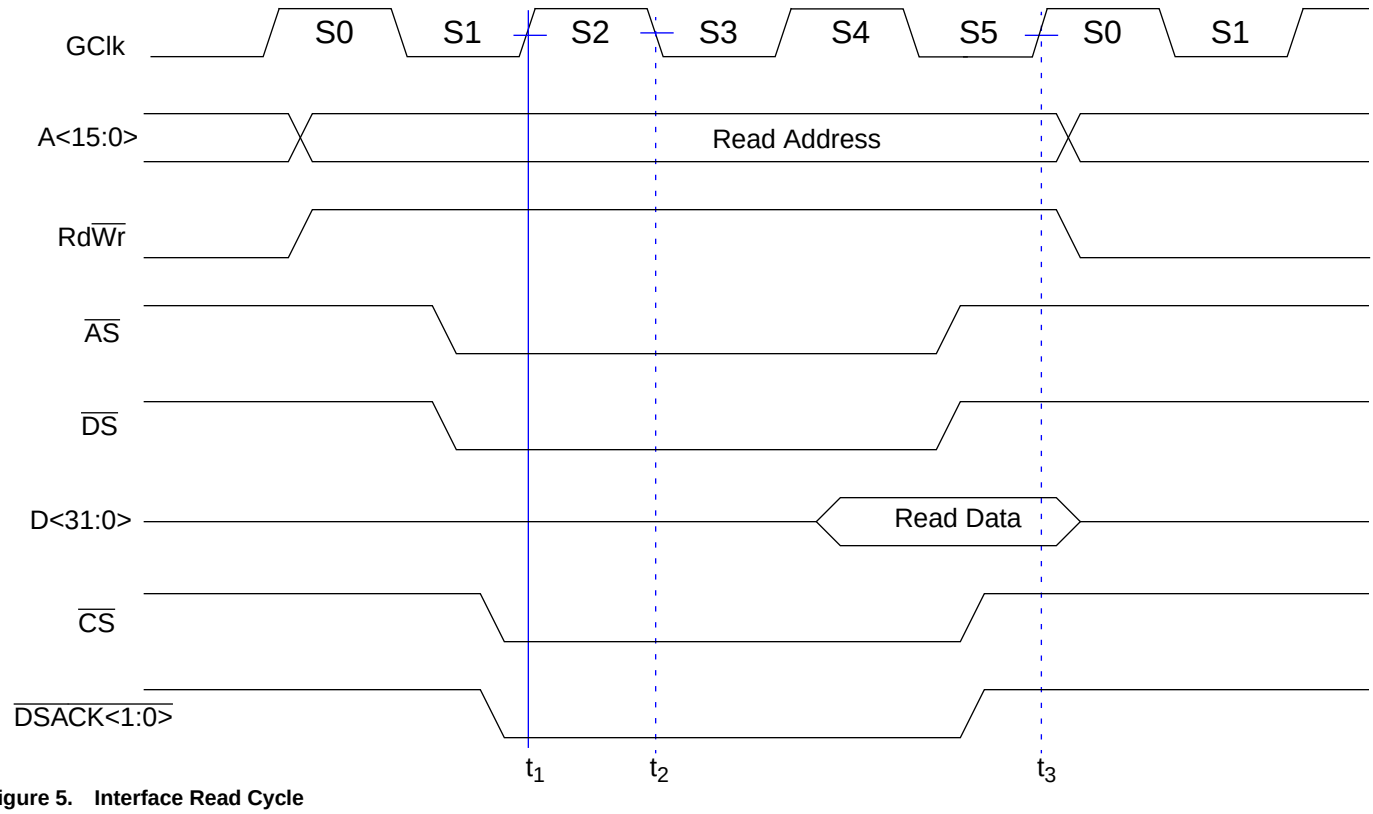

Figure 5. Interface Read Cycle

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

Warning: THIS IS AN UNTESTED DESIGN.

Xilinx, Inc. does not make any representation or warranty regarding this design or any item based on this design. Xilinx disclaims all express and implied warranties, including but not limited to the implied fitness of this design for a particular purpose and freedom from infringement. Without limiting the generality of the foregoing, Xilinx does not make any warranty of any kind that any item developed based on this design, or any portion of it, will not infringe any copyright, patent, trade secret or other intellectual property right of any person or entity in any country. It is the responsibility of the user to seek licenses for such intellectual property right where applicable. Xilinx shall not be liable for any damages arising out of or in connection with the use of the design including liability for lost profit, business interruption, or any other damages whatsoever.