XILINX ®

CoolRunner XPLA3 CPLD 8051 Microcontroller Interface

XAPP349 (v1.2) January 15, 2003

Summary

This document details the VHDL implementation of an 8051 microcontroller interface in a Xilinx CoolRunner™ XPLA3 CPLD. CoolRunner XPLA3 CPLDs are the lowest power CPLDs available, making these CPLDs the perfect interface devices for many of today's popular microcontrollers. To obtain the VHDL code described in this document, go to section **VHDL Code Download and Disclaimer**, page 10 for instructions. For implementation with a CoolRunner-II™ CPLD, refer to XAPP393 found on **http://www.xilinx.com**.

Introduction

An interface to the 8051 microcontroller has been implemented in a CoolRunner XPLA3 CPLD. This design consists of a state machine that interprets the 8051 bus cycles to read and write data to a set of registers called a register file. A high-level block diagram is shown in Figure 1. Communication between the 8051 microcontroller and the application logic is accomplished through a register file. The 8051 microcontroller writes data to the register file to configure and control the application logic. The application logic writes status information and service requests to the microcontroller through the register file. Data transfer is also done through registers. Flags can be designed that denote when registers are empty and/or full depending on the application. The number of registers and their bit definitions are defined in general terms and should be customized for the application.
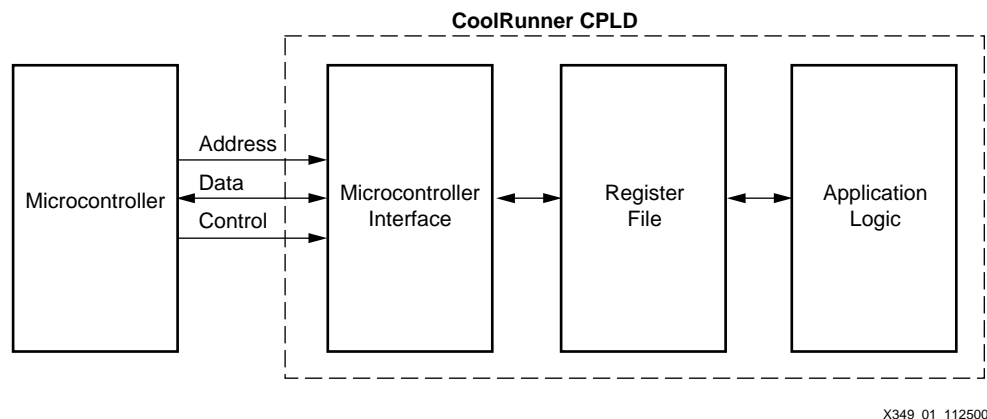

Figure 1: **CoolRunner CPLD 8051 Microcontroller Interface**
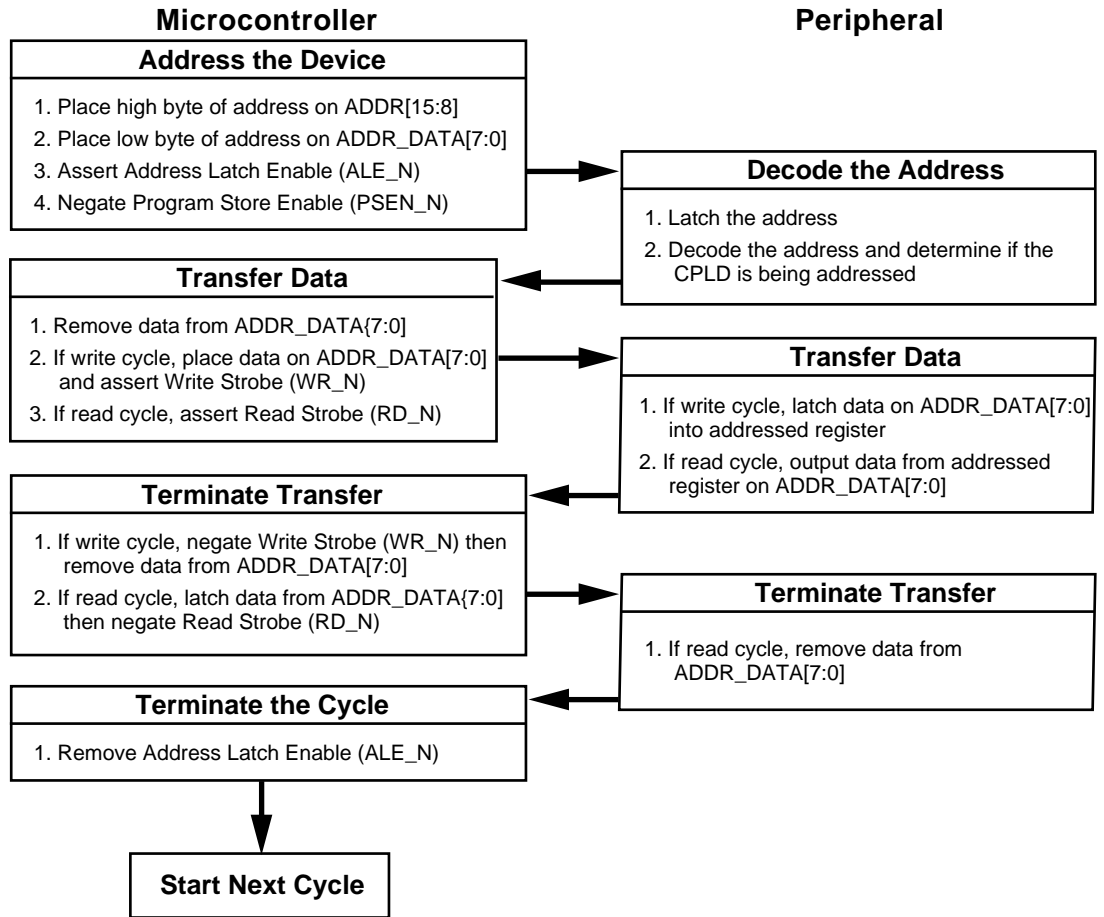
8051 Microcontroller Bus Protocol

This section describes the main protocol of the 8051 microcontroller bus. For more details and specific timing parameters, please refer to the supplier's datasheet for the 8051 microcontroller chosen for the system.

The 8051 executes synchronous bus cycles, so there is no handshaking between the peripheral and the microcontroller. The 8051 microcontroller utilizes a multiplexed address/data bus through PORT 0. The falling edge of Address Latch Enable (ALE_N) is used

to capture the lower address byte from PORT0 so that the port can then be used for data transfer. Program Store Enable (PSEN_N) is only asserted for accesses to read-only program memory and thus remains negated during external memory read and write cycles. Figure 2 shows a flow chart of the operations that occur when the 8051 accesses a peripheral.

Microcontroller   **Peripheral**

Address the Device

1. Place high byte of address on ADDR[15:8]
2. Place low byte of address on ADDR_DATA[7:0]
3. Assert Address Latch Enable (ALE_N)
4. Negate Program Store Enable (PSEN_N)

Decode the Address

1. Latch the address
2. Decode the address and determine if the CPLD is being addressed

Transfer Data

1. Remove data from ADDR_DATA{7:0]
2. If write cycle, place data on ADDR_DATA[7:0] and assert Write Strobe (WR_N)
3. If read cycle, assert Read Strobe (RD_N)

Transfer Data

1. If write cycle, latch data on ADDR_DATA[7:0] into addressed register
2. If read cycle, output data from addressed register on ADDR_DATA[7:0]

Terminate Transfer

1. If write cycle, negate Write Strobe (WR_N) then remove data from ADDR_DATA[7:0]
2. If read cycle, latch data from ADDR_DATA{7:0] then negate Read Strobe (RD_N)

Terminate Transfer

1. If read cycle, remove data from ADDR_DATA[7:0]

Terminate the Cycle

1. Remove Address Latch Enable (ALE_N)

Start Next Cycle

X349_02_112500

Figure 2: **8051 Bus Protocol for External Memory Accesses**

Bus Cycle Timing Diagrams

No numerical values for the parameters shown in these timing diagrams are provided. The data sheet for the 8051 μC used in the system must be consulted for these parameters.

External Memory Read Cycle

For an external memory read cycle (Figure 3), PORT0 contains the address of the target peripheral T_{AVLL} ns before ALE_N asserts. The address is held on PORT0 for T_{LLAX} ns and is then 3-stated T_{RLAZ} ns before RD_N asserts. Once RD_N asserts, the addressed peripheral can drive the requested data onto the bus. This data must be valid T_{RLDV} ns after RD_N

asserts. The microcontroller negates RD_N after it has retrieved the data from the bus and ends the cycle by negating ALE_N.
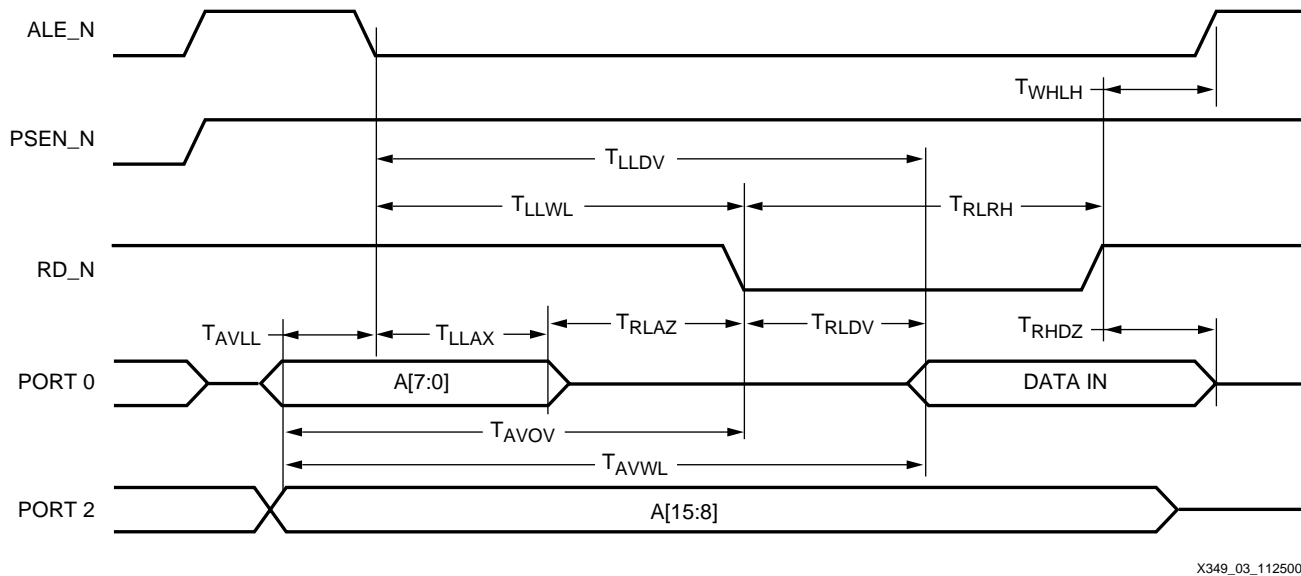

X349_03_112500

Figure 3: **8051 Microcontroller External Data Memory Read Cycle**

External Memory Write Cycle

External memory write cycles (Figure 4) are quite similar to external memory read cycles with one exception: instead of 3-stating PORT 0 after the address has been latched, the data to be written to the addressed device is placed on PORT 0 T_{QVWX} ns before WR_N asserts. WR_N then asserts and is held for T_{WLWH} ns. After WR_N negates, the data is held on PORT 0 T_{WHDX} ns and the cycle ends with the negation of ALE_N.
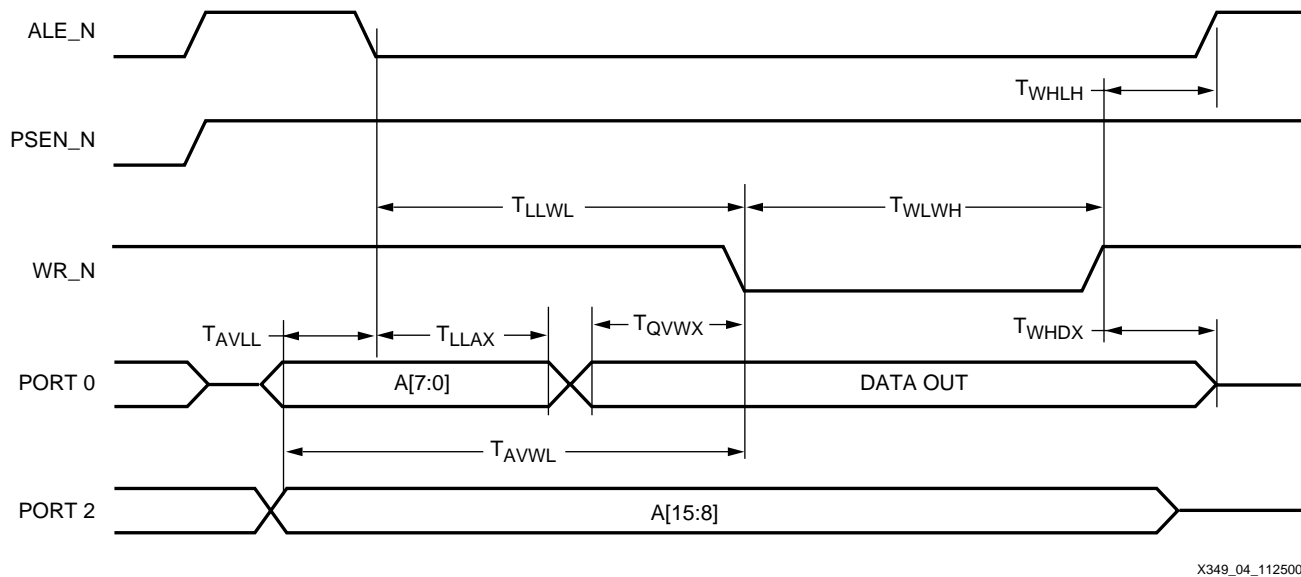

X349_04_112500

Figure 4: **8051 Microcontroller External Data Memory Write Cycle**
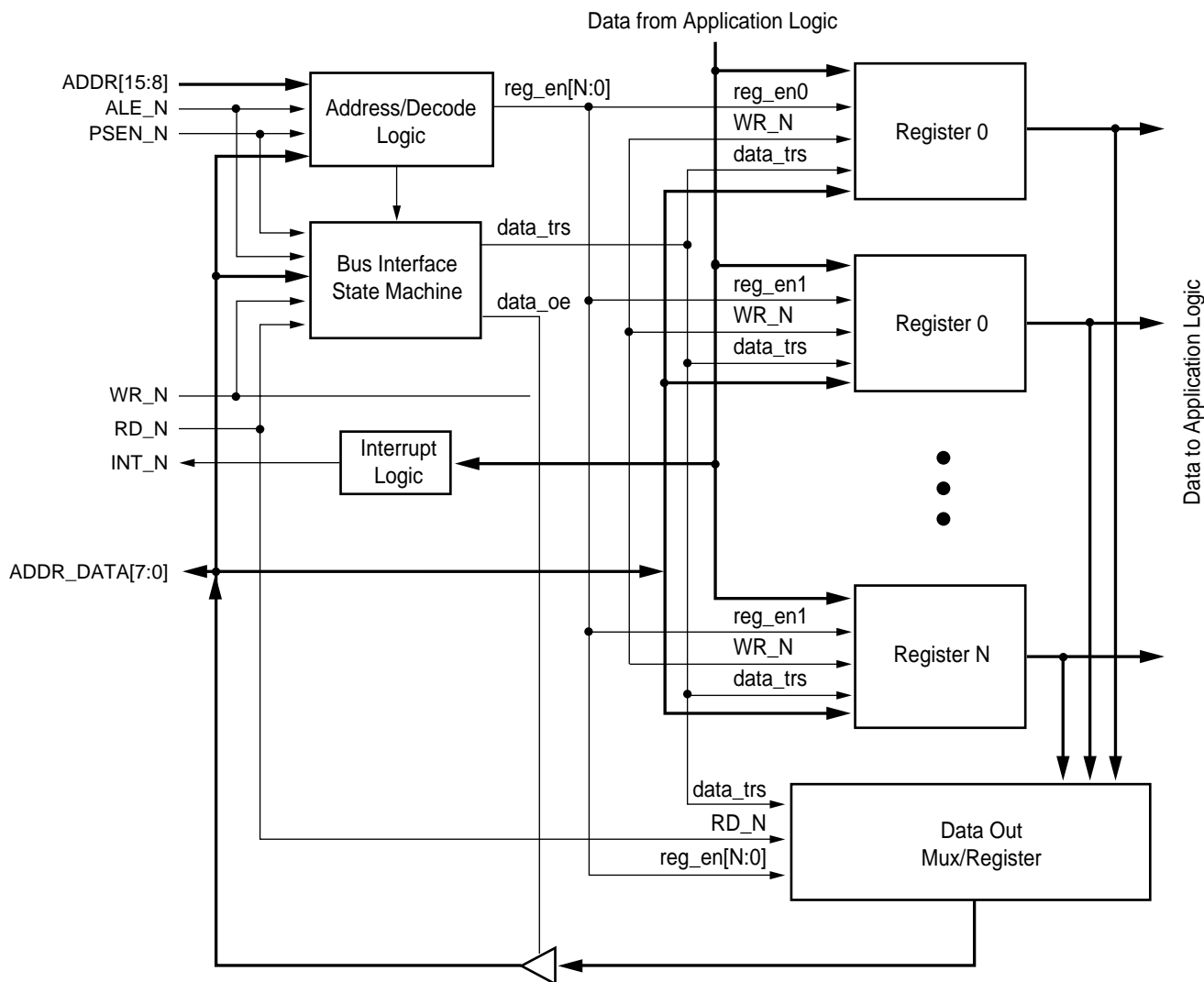
CoolRunner CPLD 8051 Microcontroller Interface Signals

The I/O signals of the CoolRunner CPLD 8051 microcontroller interface are described in Table 1. Pin numbers have not been assigned to this design, this can be done to meet the system requirements of the designer.

Table 1: **CoolRunner CPLD 8051 Interface Signal Description**

Name	Direction	Description
ADDR[15:8]	Input	µC Address Bus. High byte address bus.
ADDR_DATA[7:0]	Bidirectional	**µC Multiplexed Address/Data Bus**.
ALE_N	Input	**Address Latch Enable**. Active Low µC control signal indicating that the data present on the multiplexed address/data bus is a valid address.
PSEN_N	Input	**Program Store Enable.** Active Low µC control signal indicating that the current bus cycle is an access to the external program memory.
RD_N	Input	**Read Strobe.** Active Low µC control signal indicating that the current bus cycle is a read cycle.
WR_N	Input	**Write Strobe.** Active Low µC control signal indicating that the current bus cycle is a write cycle.
INT_N	Output	**Interrupt Request.** Active Low signal to generate an interrupt to the µC. This signal is asserted based on application logic.
CLK	Input	**Clock.** This clock is input from the system.
RESET	Input	**Reset.** Active High reset from the system. When asserted, all logic in the CoolRunner CPLD is reset.

CoolRunner CPLD Block Diagram

The block diagram of the CoolRunner CPLD 8051 Interface consists of address decode logic, a bus interface state machine, interrupt logic, user definable registers, and a data out multiplexor to supply the requested data during a read cycle as shown in Figure 5.

Figure 5: **CoolRunner CPLD 8051 Microcontroller Interface Block Diagram**

Address Decode Logic

The CoolRunner CPLD 8051 Microcontroller Interface logic decodes the upper address byte present on the ADDR bus as the device address. This address is compared to the constant BASE_ADDR in the VHDL code to determine whether the device is being addressed or not. If the upper address byte is equal to BASE_ADDR, the ADDR_MATCH signal is asserted.

The address decode logic also contains the registers clocked by ALE_N to capture the lower address bits from the multiplexed ADDR_DATA bus. This lower address byte represents the addresses of registers in the register file. This allows for the possibility of 256 registers in the register file if even and odd addresses are allowed, and 128 registers if only even addresses are allowed. The current implementation in the VHDL code instantiates four registers on even addresses.

The lower address byte is compared to register address constants set in the VHDL code. If there is a match, the register enable for the register being addressed is asserted.

The address decoding implemented in the VHDL code must be modified to fit the addressing scheme of the system. The address bits that represent the device address and the address bits that represent the register address within the device must also be adjusted for the system requirements. These parameters can be modified by changing the constants shown in Table 2 in the VHDL code.

Table 2: **Address Decode Parameters**

Constant	Description	Default Value
BASE_ADDR	Device Address	00\h
DEVICE_ADDR_BITS	Number of address bits used to decode device address	8
REG_ADDR_BITS	Number of address bits used to decode register addresses	8

Bus Interface State Machine

The 8051 μC bus protocol is implemented in the state machine shown in Figure 6.
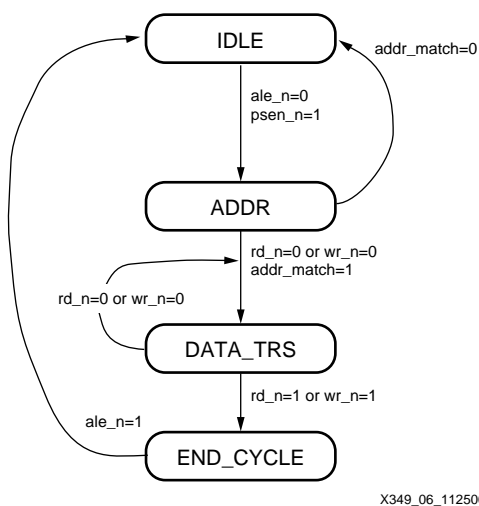

X349_06_112500

Figure 6: μ**C Bus Interface State Machine**

In the first cycle, the μC places the address on the address bus and asserts address latch enable (ALE_N). ALE_N indicates that the data on the multiplexed address/data bus is a valid address and that the address on ADDR[15:0] is also valid.

Upon the assertion of ALE_N, the state machine transitions to the ADDR_DECODE state to wait for the ADDRESS/DECODE logic to decode the address and determine if it is the device being addressed. If this is a write cycle, the μC removes the address from the multiplexed address/data bus and places the data to be written onto these signals. The write strobe (WR_N) is then asserted. If this is a read cycle, the μC 3-states the multiplexed address/data bus and asserts the read strobe (RD_N) indicating that the CoolRunner CPLD can place data from the addressed register on the data bus.

If the CoolRunner CPLD is being addressed and either RD_N or WR_N are asserted, the state machine progresses to the DATA_TRS state. If this is a read cycle, the requested data is placed on the bus and if this is a write cycle, the data from the data bus is latched in the addressed register.

The μC latches the data present on the bus if this is a read cycle and then negates the read strobe (RD_N). If this is a write cycle, the μC removes data from the bus and then negates the write strobe (WR_N). The negation of either RD_N or WR_N causes the state machine to

www.xilinx.com
1-800-255-7778

progress to the END_CYCLE state. The CoolRunner CPLD will 3-state the multiplexed address/data bus in this state, removing the data if it is a read cycle.

At this point, the µC ends the cycle by negating address latch enable (ALE_N), which causes the CoolRunner CPLD to return to the IDLE state.

Register File

The Register File consists of a bank of registers used for communication between the 8051 µC and the application logic. These registers can be defined as needed by the application. For example, most applications require a control register which sets up certain required parameters such as interrupt enables, start/stop, clock divisors, clock phases and polarities, and/or master/slave operation. This register and its address are then defined in the VHDL code and the 8051 µC reads and writes to this register to configure the application.

Likewise, many applications are required to provide status information back to the 8051 µC such as a data error, data operation complete, data available, transmit buffer is empty, etc. A status register and its address are then defined in the VHDL code and the 8051 µC reads and possibly writes to this register (to reset status conditions) to determine the application's progress and status.

The registers currently implemented in the CoolRunner 8051 Microcontroller Interface are described in the Table 3. The user will have to examine the VHDL code to create the registers required by the application and define the addressing scheme required by the system.The register definition will determine if the input to the register comes from the 8051 µC data bus or the application logic. Once the user defines the registers required by the application, the VHDL code available with this application note will need to be modified to support the correct register implementation. The constants in the VHDL code representing the register addresses will also need to be modified.

Table 3: **Register File Addresses**

Address	Register	VHDL Constant	Description
BASE + $80\h	STATUS_REG	STATUS_ADDR	**Example Status Register**
BASE + $82\h	CTRL_REG	CTRL_ADDR	**Example Control Register**
BASE + $84\h	DATAIN_REG	DATAIN_ADDR	**Example Data Input Register**
BASE + $86\h	DATAOUT_REG	DATAOUT_ADDR	**Example Data Output Register**

Example Status Register

This register is an example status register containing the status of a typical application. This status register is read-only with the exception of certain bits which are software clearable as described in Table 4. Note that for software testability, it may be determined that all bits of the

status register are writable, however, the VHDL code available with this application note does not support this.

Table 4: **Example Status Register Bits**

Bit Location	Name	μC Access	Description
7	DONE	Read	**Done Bit.** Data operation has completed. "1" operation is complete "0" operation in progress
6	ERROR	Read Software Clearable	**Error Bit.** An error requiring 8051 μC interaction has occurred. This bit will cause an interrupt to the 8051 μC if interrupts have been enabled in the control register. This bit is reset when the 8051 μC writes a 0 to this bit.
5	INTR	Read Software Clearable	**Interrupt Bit.** This bit is asserted (active high) when an interrupt is pending which causes a processor interrupt request if interrupts are enabled (INT_EN is set).This bit is reset when the 8051 μC writes a 0 to this bit.
4	NEED_DATA	Read	**Need Data Bit.** This bit is set when the data input register is empty. It is cleared when the μC writes data into the data input register. An interrupt will be asserted to the μC when this bit is set if interrupts are enabled (INT_EN is set).
3	DATA_RDY	Read	**Data Ready Bit**. This bit is set whenever the data output register is full. It is cleared when the μC reads from the data output register. An interrupt will be asserted to the μC when this bit is set if interrupts are enabled.
2-0	Unused		**Unused Bits.** These bits will read as "0" when the status register is read.

Example Control Register

This register is an example control register providing the bits to configure and control the operation of a typical application. All bits in this register can be written to and read from the 8051 μC as shown in Table 5.

Table 5: **Control Register Bits**

Bit Location	Name	μC Access	Description
7	APP_EN	Read/Write	Application Enable. This bit enables the user application logic and must be set before any other bits in the control register have any effect "1" enables the user application "0" resets and disables the user application
6	INT_EN	Read/Write	Interrupt Enable. "1" enables interrupts. An interrupt occurs if the INT_N bit in the status register is also set "0" disables interrupts but does not clear the cause of any currently pending interrupts
5	START	Read/Write	Application Start. When the μC changes this bit from "0" to "1", the application logic begins its operation on the data provided in the DATA_IN register.
4-0	CTRL_BITS	Read/Write	Control Bits. These bits need to be defined by the user application.

Example Data Input Register

This register is an example data input register containing data from the 8051 μC that the application performs some operation on, such as a transmission over the SPI bus. This data register is both readable and writable and is described in Table 6.

Table 6: **Example Data Input Register**

Bit Location	Name	μC Access	Description
7-0	DATA_IN	Read/Write	**Data Input.** Data byte from 8051 μC to perform the application's operation on.

Example Data Output Register

This register is an example data output register containing the results of the data operation. This data register is read only and is described in Table 7. Note that for software testability, it may be determined that all bits of the data output register are writable, however, the VHDL code available with this application note does not support this.

Table 7: **Example Data Output Register**

Bit Location	Name	μC Access	Description
7-0	DATA_OUT	Read Only	**Data Output.** Resulting data byte from application's operation.

Application Logic

The CoolRunner CPLD 8051 Microcontroller Interface is intended to be used as a front end to additional application logic. The preceding paragraphs have described the code available to implement this interface; however, modifications to the address decode logic, device and register addresses, and register definitions will be required to suit the application.

VHDL Testbench and Functional Simulation

A VHDL testbench has been developed that verifies the CoolRunner 8051 Microcontroller Interface through various read and write bus cycles. This testbench contains a process that emulates the bus cycles of the 8051 µC. Constants are provided at the top of the testbench file to set up the base address of the CoolRunner CPLD and all of the registers contained within the device. These constants should be modified to match the addressing scheme of the designer's system.

The testbench also contains constant data arrays for specification of the data to be input to the application logic and for the expected data output from the application logic. The user needs to modify the constants in these arrays and the size of these arrays to match the bus cycles desired in simulation.

The testbench contains dummy processes to emulate application logic. These processes assert the NEED_DATA and DATA_RDY flags as well as provide the load signal and data for the output data registers. These processes should be removed from the testbench when the application logic is available.

The ModelSim command file, *func_sim.do,* can be used to open the correct waveform window and run the simulation.

CoolRunner XPLA3 CPLD Implementation

The CoolRunner 8051 Microcontroller Interface utilizes only 57 macrocells and has been targeted to a XPLA3 64 macrocell device. The speed grade chosen is dependent on the system clock frequencies and should be analyzed by the designer to determine which speed grade is required. Once application logic has been added to the design, the user should target a CoolRunner CPLD with enough macrocells to accommodate the 8051 Microcontroller Interface described in this application note and the required application logic.

Post-fit Timing Simulation

The Xilinx Project Navigator software package outputs a timing VHDL model of the fitted design. This post-fit VHDL was simulated with the original VHDL test benches to insure design functionality using ModelTech Xilinx Edition (MXE). Please note that all verification of this design has been done through simulations.

The user of this design is strongly encouraged to thoroughly inspect the timing report for this design to insure that the design meets the timing specification of the system. The user is also strongly encouraged to perform post-fit timing simulations as well. The ModelSim command file, *post_sim.do*, can be used to open the correct waveform window and run the simulation.

VHDL Code Download and Disclaimer

All VHDL source code, VHDL testbenches, and software files associated with this design are available. Note that this reference design is intended to be used as a front-end interface to the user's application logic. It is not a standalone design. Therefore, modifications to the VHDL code will be required to support the addressing scheme, register structures, interrupt logic, and process flow of the application. This VHDL code is provided as an example that can be considered a starting point for your application.

THE DESIGN IS PROVIDED TO YOU "AS IS". XILINX MAKES AND YOU RECEIVE NO WARRANTIES OR CONDITIONs, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, AND XILINX SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. This design should be used only as an example design, not as a fully functional core. XILINX does not warrant the performance, functionality, or operation of this Design will meet your requirements, or that the operation of the Design will be uninterrupted or error free, or that defects in the Design will be

XAPP349 - **http://www.xilinx.com/products/xaw/coolvhdlq.htm**

Conclusion

This document has detailed the design of a 8051 microcontroller interface for a CoolRunner XPLA3 CPLD. Though the design has been extensively verified in simulations, Xilinx assumes no responsibility for the accuracy or the functionality of this design.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/07/00	1.0	Initial Xilinx release.
10/01/02	1.1	Minor revisions.
01/15/03	1.2	Minor revisions.