Implementing an I²C Bus Controller in a CoolRunner™ CPLD

XAPP315 (v1.0) October 25, 1999 Application Note

Summary

This document details the VHDL implementation of an I²C controller in a Xilinx CoolRunner™ 128 macrocell CPLD. CoolRunner CPLDs are the lowest power CPLDs available and thus are the perfect target device for an I²C controller. The VHDL code described in this document can be obtained by contacting Xilinx Technical Support.

Introduction

The I²C bus is a popular serial, two-wire interface used in many systems because of its low overhead. The two-wire interface minimizes interconnections so ICs have fewer pins, and the number of traces required on printed circuit boards is reduced. Capable of 100 KHz operation, each device connected to the bus is software addressable by a unique address with a simple Master/Slave protocol.

The CoolRunner I²C Controller design contains an asynchronous microcontroller (µC) interface and provides I²C Master/Slave capability. It is intended to be used with a microcontroller (µC) or microprocessor (µP) as shown in Figure 1.

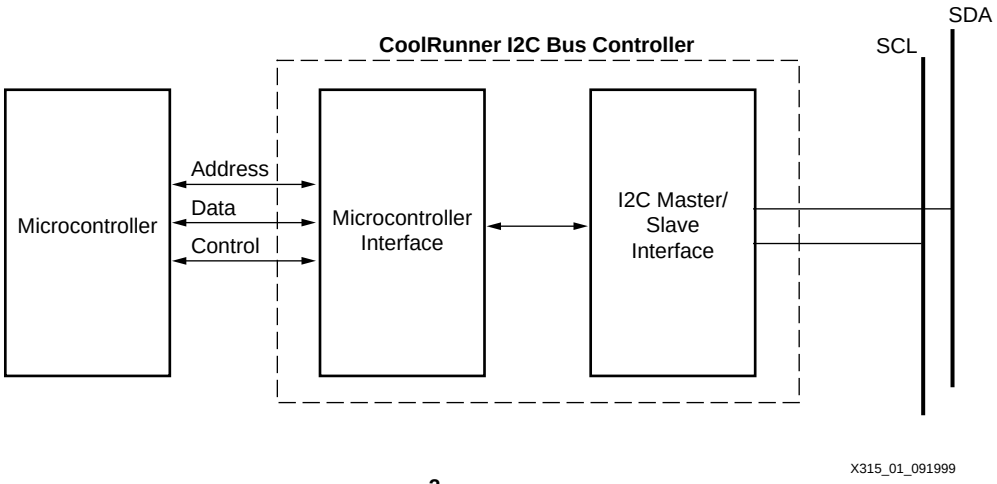

Figure 1: CoolRunner I²C Bus Controller

I²C Background

This section will describe the main protocol of the I²C bus. For more details and timing diagrams, please refer to the I²C specification.

The I²C bus consists of two wires, serial data (SDA) and serial clock (SCL), which carry information between the devices connected to the bus. The number of devices connected to the same bus is limited only by a maximum bus capacitance of 400 pF. Both the SDA and SCL lines are bidirectional lines, connected to a positive supply voltage via a pull-up resistor. When the bus is free, both lines are High. The output stages of devices connected to the bus must have an open-drain or open-collector in order to perform the wired-AND function.

Each device on the bus has a unique address and can operate as either a transmitter or receiver. In addition, devices can also be configured as Masters or Slaves. A Master is the device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. Any other device that is being addressed is considered a Slave. The I²C protocol

defines an arbitration procedure that insures that if more than one Master simultaneously tries to control the bus, only one is allowed to do so and the message is not corrupted. The arbitration and clock synchronization procedures defined in the I2C specification are supported by the CoolRunner I2C Controller.

Data transfers on the I2C bus are initiated with a START condition and are terminated with a STOP condition. Normal data on the SDA line must be stable during the High period of the clock. The High or Low state of the data line can only change when SCL is Low. The START condition is a unique case and is defined by a High-to-Low transition on the SDA line while SCL is High. Likewise, the STOP condition is a unique case and is defined by a Low-to-High transition on the SDA line while SCL is High. The definitions of data, START, and STOP insure that the START and STOP conditions will never be confused as data. This is shown in Figure 2.

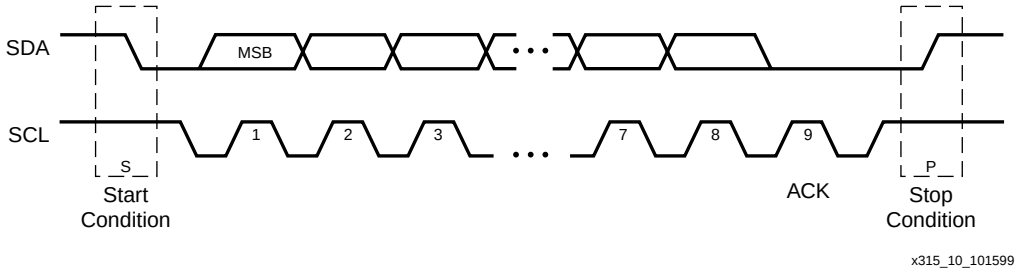

Figure 2: Data Transfer on the I2C Bus

Each data packet on the I2C bus consists of eight bits of data followed by an acknowledge bit so one complete data byte transfer requires nine clock pulses. Data is transferred with the most significant bit first (MSB). The transmitter releases the SDA line during the acknowledge bit and the receiver of the data transfer must drive the SDA line low during the acknowledge bit to acknowledge receipt of the data. If a Slave-receiver does not drive the SDA line Low during the acknowledge bit, this indicates that the Slave-receiver was unable to accept the data and the Master can then generate a STOP condition to abort the transfer. If the Master-receiver does not generate an acknowledge, this indicates to the Slave-transmitter that this byte was the last byte of the transfer.

Standard communication on the bus between a Master and a Slave is composed of four parts: START, Slave address, data transfer, and STOP. The I2C protocol defines a data transfer format for both 7-bit and 10-bit addressing. The implementation of the I2C controller in the Xilinx CoolRunner CPLD supports the seven-bit address format. After the START condition, a Slave address is sent. This address is seven bits long followed by an eighth-bit which is the read/write bit. A "1" indicates a request for data (read) and a "0" indicates a data transmission (write). Only the Slave with the calling address that matches the address transmitted by the Master responds by sending back an acknowledge bit by pulling the SDA line Low on the ninth clock.

Once successful Slave addressing is achieved, the data transfer can proceed byte-by-byte as specified by the read/write bit. The Master can terminate the communication by generating a STOP signal to free the bus. However, the Master may generate a START signal without generating a STOP signal first. This is called a repeated START.

CoolRunner I2C Controller

The CoolRunner CPLD implementation of the I2C Controller supports the following features:

- Microcontroller interface
- Master or Slave operation
- Multi-master operation
- Software selectable acknowledge bit

- Arbitration lost interrupt with automatic mode switching from Master to Slave
- Calling address identification interrupt with automatic mode switching from Master to Slave
- START and STOP signal generation/detection
- Repeated START signal generation
- Acknowledge bit generation/detection
- Bus busy detection
- 100 KHz operation

Signal Descriptions

The I/O signals of the CoolRunner I2C controller 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 I2C Controller Signal Description

Name	Direction	Description
SDA	Bidirectional	**I2C Serial Data.**
SCL	Bidirectional	**I2C Serial Clock.**
ADDR_BUS[23:0]	Input	**µC Address Bus.**
DATA_BUS[7:0]	Bidirectional	**µC Data Bus**.
AS	Input	**Address Strobe**. Active Low µC handshake signal indicating that the address present on the address bus is valid.
DS	Input	**Data Strobe.** Active Low µC handshake signal indicating that the data present on the data bus is valid or that the µC is no longer driving the data bus and the I2C Controller can place data on the data bus.
R_W	Input	**Read/Write.** "1" indicates a read, "0" indicates a write.
DTACK	Output	**Data Transfer Acknowledge.** Active Low µC handshake signal indicating that the I2C Controller has placed valid data on the data bus for a read cycle or that the I2C Controller has received the data on the bus for a write cycle.
IRQ	Output	**Interrupt Request.** Active Low.
MCF	Output	**Data Transferring Bit.** While one byte of data is being transferred, this bit is cleared. It is set by the falling edge of the ninth clock of a byte transfer. This bit is used to signal the completion of a byte transfer to the µC.
CLK	Input	**Clock.** This clock is input from the system. The constants used in generating a 100 KHz SCL signal assumes the frequency to be 1.832 MHz. Different clock frequencies can be used, but the constants in the VHDL source code must be recalculated.

Block Diagram

The block diagram of the CoolRunner I2C Controller, shown in Figure 3 was broken into two major blocks, the µC interface and the I2C interface.

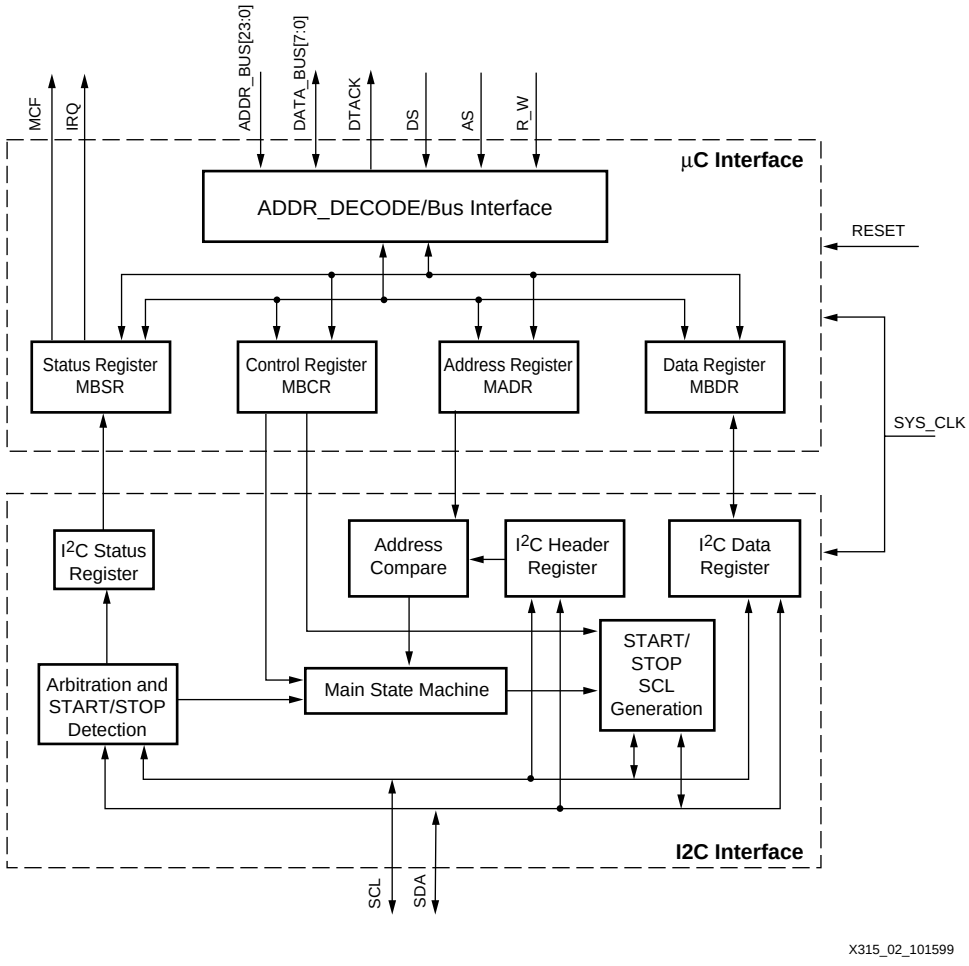

Figure 3: CoolRunner I2C Controller

µC Interface Logic

The µC interface for the I2C controller design supports an asynchronous byte-wide bus protocol. This protocol is the method in which the µC reads and writes the registers in the design and is shown in Figure 4.

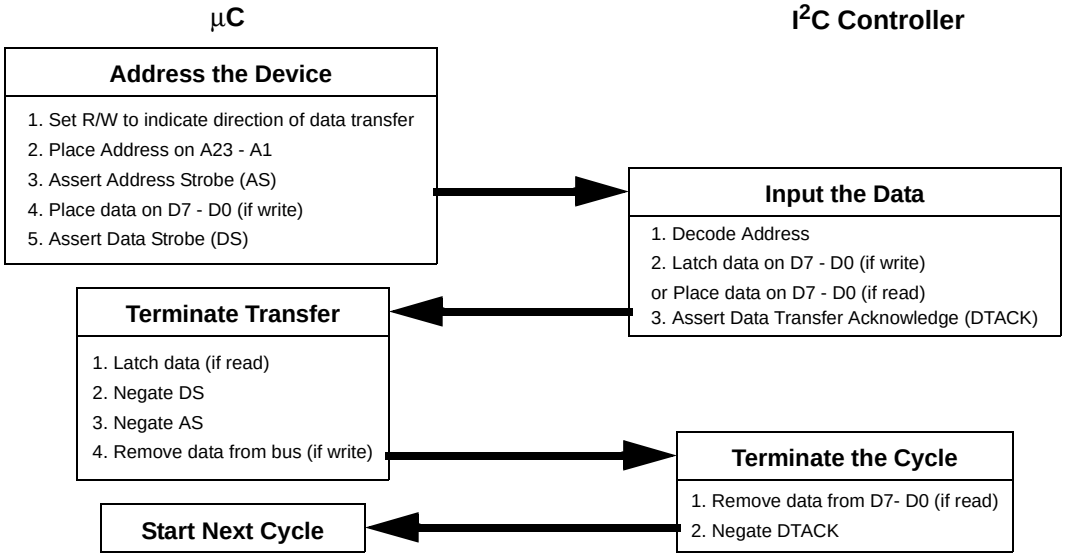

Figure 4: µC Read/Write Protocol

Address Decode/Bus Interface Logic

The µC bus protocol is implemented in the CoolRunner I2C Controller in the state machine shown in Figure 5.

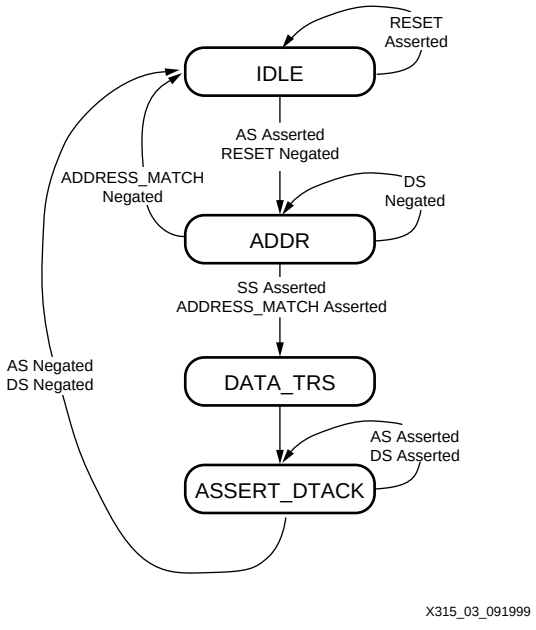

Figure 5: µC Bus Interface State Machine

In the first cycle, the µC places the address on the address bus, sets the read/write line to the correct state, and asserts address strobe (AS) and data strobe (DS). Address strobe indicates that the address present on the address bus is valid. If this is a write cycle, the µC also places the data on the data bus and DS indicates that valid data is present on the data bus. If this is a read cycle, the µC tri-states the data bus and DS indicates that the CoolRunner I2C Controller can place data on the data bus.

Upon the assertion of AS, the CoolRunner I2C Controller transitions to the ADDR state to decode the address and determine if it is the device being addressed. The enables for the internal registers are set in this state. If the CoolRunner I2C Controller is being addressed and DS is asserted, the CoolRunner I2C controller 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 CoolRunner I2C Controller automatically progresses to the ASSERT_DTACK state and asserts DTACK indicating that the data requested is ready if a read cycle or that the data has been received if a write cycle.

Upon the assertion of DTACK, the µC either removes data from the bus if this is a write cycle, or latches the data present on the bus if this is a read cycle. The read/write line is set to read and AS and DS are negated to indicate that the data transfer is complete. The negation of AS and DS causes the CoolRunner I2C Controller to negate DTACK and transition to the IDLE state.

CoolRunner I2C Controller Registers

The base address used for address decoding is set in the VHDL code via the constant BASE_ADDRESS. The base address is the upper 16 bits of the address bus. The lower address bits determine which register is being accessed.

The registers supported in the CoolRunner I2C Controller are described in the Table 2. The µC interface logic of the CoolRunner I2C Controller handles the reading and writing of these registers by the µC and supplies and/or retrieves these bits to/from the I2C interface logic.

Table 2: I2C Controller Registers

Address	Register	Description
MBASE + $141	MADR	I2C Address Register
MBASE + $145	MBCR	I2C Control Register
MBASE + $147	MBSR	I2C Status Register
MBASE + $149	MBDR	I2C Data I/O Register

Address Register (MADR)

This field contains the specific Slave address to be used by the I2C Controller. This register is read/write. (Table 3)

Table 3: Address Register Bits

Bit Location	Name	µC Access	Description
7-1	Slave Address	Read/Write	Address used by the I2C controller when in Slave mode.
0			Unused

Control Register (MBCR)

This register contains the bits to configure the I2C controller. (Table 4)

Table 4: Control Register Bits

Bit Location	Name	µC Access	Description
7	MEN	Read/Write	**I2C Controller Enable.** This bit must be set before any other MBCR bits have any effect - "1" enables the I2C controller - "0" resets and disables the I2C controller
6	MIEN	Read/Write	**Interrupt Enable.** - "1" enables interrupts. An interrupt occurs if MIF bit in the status register is also set - "0" disable interrupts but does not clear any currently pending interrupts
5	MSTA	Read/Write	**Master/Slave Mode Select.** When the µC changes this bit from "0" to "1", the I2C controller generates a START condition in Master mode. When this bit is cleared, a STOP condition is generated and the I2C controller switches to Slave mode. If this bit is cleared, however, because arbitration for the bus has been lost, a STOP condition is not generated.
4	MTX	Read/Write	**Transmit/Receive Mode Select.** This bit selects the direction of Master/Slave transfers. - "1" selects an I2C transfer - "0" selects an I2C receive
3	TXAK	Read/Write	**Transmit Acknowledge Enable.** This bit specifies the value driven onto the SDA line during acknowledge cycles for both Master and Slave receivers - "1" - ACK bit = "1" - no acknowledge - "0" - ACK bit = "0" - acknowledge Since Master receivers indicate the end of data reception by not acknowledging the last byte of the transfer, this bit is the means for the µC to end a Master receiver transfer.
2	RSTA	Read/Write	**Repeated Start.** Writing a "1" to this bit generates a repeated START condition on the bus if the I2C controller is the current bus Master. This bit is always read as "0". Attempting a repeated START at the wrong time if the bus is owned by another Master results in a loss of arbitration.
1 - 0	Reserved		

Status Register (MBSR)

This register contains the status of the I2C controller. This status register is read-only with the exception of the MIF and MAL bits, which are software clearable. All bits are cleared upon reset except the MCF and RXAK bits. (Table 5)

Table 5: Status Register Bits

Bit Location	Name	µC Access	Description
7	MCF	Read	Data Transferring Bit. While on byte of data is being transferred, this bit is cleared. It is set by the falling edge of the ninth clock of a byte transfer. - "1" transfer is complete - "0" transfer in progress Note that in the CoolRunner I2C controller, this bit is also an output pin so that a register read cycle is not required to determine that a transfer is complete.
6	MAAS	Read	Addressed as Slave Bit. When the address on the I2C bus matches the Slave address in the MADR register, the I2C controller is being addressed as a Slave and switches to Slave mode.
5	MBB	Read	Bus Busy Bit. this bit indicates the status of the I2C bus. This bit is set when a START condition is detected and cleared when a STOP condition is detected. - "1" indicates the bus is busy - "0" indicates the bus is idle
4	MAL	Read Software Clearable	Arbitration Lost Bit. This bit is set by hardware when arbitration for the I2C bus is lost. This bit must be cleared by the µC software writing a "0" to this bit.
3	Reserved		
2	SRW	Read	When the I2C controller has been addressed as a Slave (MAAS is set), this bit indicates the value of the read/write bit sent by the Master. This bit is only valid when a complete transfer has occurred and no other transfers have been initiated. - "1" indicates Master reading from Slave - "0" indicates Master writing to Slave
1	MIF	Read Software Clearable	Interrupt Bit. This bit is set when an interrupt is pending, which causes a processor interrupt request if MIEN is set. This bit must be cleared by the µC software writing a "0" to this bit in the interrupt service routine.
0	RXAK	Read	Received Acknowledge Bit. This bit reflects the value of the SDA signal during the acknowledge cycle of the transfer. - "1" indicates that no acknowledge was received - "0" indicates that an acknowledge was received

Data Register (MBDR)

This register contains data to/from the I2C bus. Physically, this register is implemented by two byte-wide registers at the same address, one for the I2C transmit data and one for the I2C received data. This eliminates any possible contention between the µC and the CoolRunner I2C Controller. Since these registers are at the same address they appear as the same register to the µC and will continue to be described as such. In transmit mode, data written into this register is output on the I2C bus, in receive mode, this register contains the data received from the I2C bus. Note that in receive mode, it is assumed that the µC will be able to read this register during the next I2C transfer. The received I2C data is placed in this register after each complete transfer, the I2C interface logic does not wait for an indication from the µC that this register has been read before proceeding with the next transfer. (Table 6)

Table 6: I2C Data Register Bit

Bit Location	Name	µC Access	Description
7 - 0	D7 - D0	Read/Write	I2C Data

I2C Interface Logic

The I2C bus interface logic consists of several different processes as seen in Figure 3. Control bits from the µC interface registers determine the behavior of these processes.

Arbitration

Arbitration of the I2C bus is lost in the following circumstances:

- The SDA signal is sampled as a "0" when the Master outputs a "1" during an address or data transmit cycle
- The SDA signal is sampled as a "0" when the Master outputs a "1" during the acknowledge bit of a data receive cycle
- A start cycle is attempted when the bus is busy
- A repeated start cycle is requested in Slave mode
- A STOP condition is detected when the Master did not request it

If the CoolRunner I2C Controller is in Master mode, the outgoing SDA signal is compared with the incoming SDA signal to determine if control of the bus has been lost. The SDA signal is checked only when SCL is High during all cycles of the data transfer except for acknowledge cycles to insure that START and STOP conditions are not generated at the wrong time. If the outgoing SDA signal and the incoming SDA signals differ, then arbitration is lost and the MAL bit is set. At this point, the CoolRunner I2C Controller switches to Slave mode and resets the MSTA bit.

The CoolRunner I2C design will not generate a START condition while the bus is busy, however, the MAL bit will be set if the µC requests a START or repeated START while the bus is busy. The MAL bit is also set if a STOP condition is detected when this Master did not generate it.

If arbitration is lost during a byte transfer, SCL continues to be generated until the byte transfer is complete.

START/STOP Detection

This process monitors the SDA and SCL signals on the I2C bus for START and STOP conditions. When a START condition is detected, the Bus Busy bit is set. This bit stays set until a STOP condition is detected. The signals, DETECT_START and DETECT_STOP are generated by this process for use by other processes in the logic. Note that this logic detects the START and STOP conditions even when the CoolRunner I2C Controller is the generator of these conditions.

Generation of SCL, SDA, START and STOP Conditions

This process generates the SCL and SDA signals output on the I2C bus when in Master mode. The clock frequency of the SCL signal is ~100 KHz and is determined by dividing down the input clock. The number of input clock cycles required for generation of a 100 KHz SCL signal is set by the constant CNT_100 KHZ and is currently calculated for a system clock of 1.832 MHz. This constant can easily be modified by a designer based on the clock available in the target system. Likewise, the constants START_HOLD and DATA_HOLD contain the number of system clock cycles required to meet the I2C requirements on hold time for the SDA lines after generating a START condition and after outputting data.

The state machine that generates SCL and SDA when in Master mode is shown in Figure 6. Note that SCL and SDA are held at the default levels if the bus is busy. This state machine generates the controls for the system clock counter.

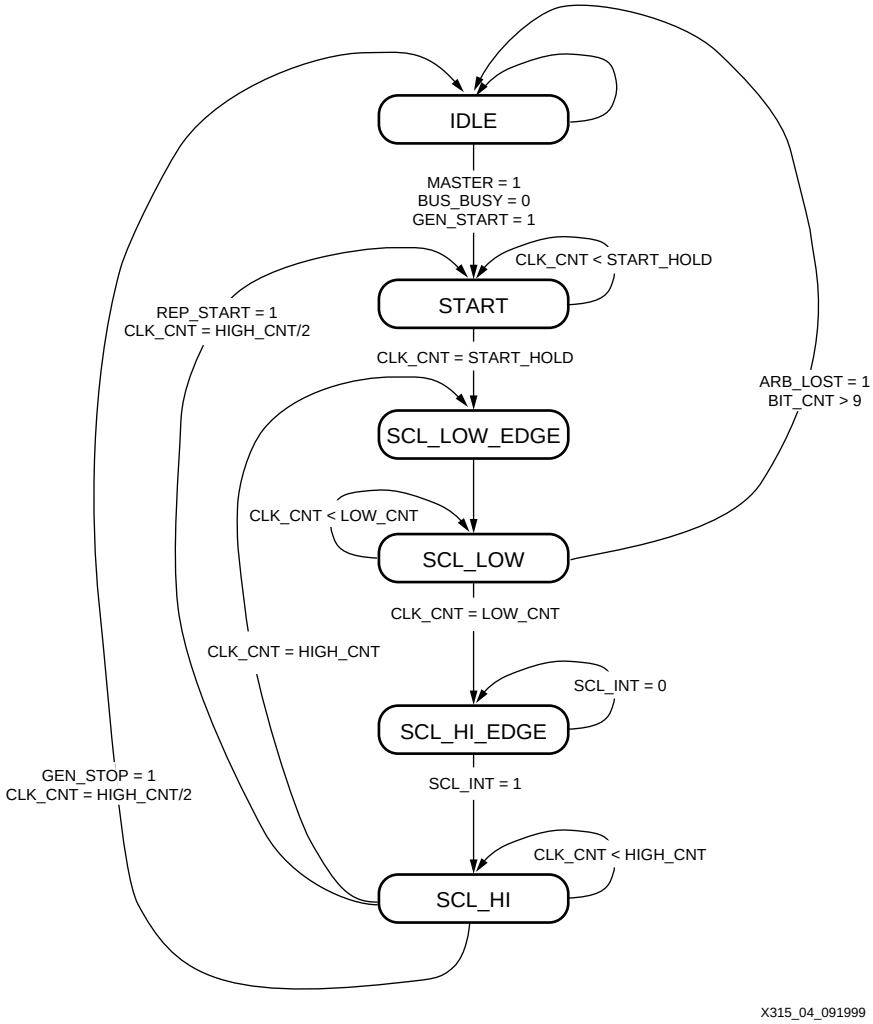

Figure 6: SCL, SDA, START, and STOP Generation State Machine

The internal SDA signal output from this design is either the SDA signal generated by this state machine for START and STOP conditions or the data from the MBDR register when the CoolRunner I2C Controller is in transmit mode. Note that both SCL and SDA are open-collector outputs, therefore, they are only driven to a "0". When a "1" is to be output on these signals, the CoolRunner I2C Controller tri-states their output buffers. The logic in the design will set internal SDA and SCL signals to "1" or "0". These internal signals actually control the output enable of the tri-state buffer for these outputs.

In the IDLE state, SCL and SDA are tri-stated, allowing any Master to control the bus. Once a request has entered to generate a start condition, the CoolRunner I2C Controller is in Master mode, and the bus is not busy, the state machine transitions to the START state.

The START state holds SCL High, but drives SDA Low to generate a START condition. The system clock counter is started and the state machine stays in this state until the required hold time is met. At this point, the next state is SCL_LOW_EDGE.

The SCL_LOW_EDGE state simply creates a falling edge on SCL and resets the system clock counter. On the next clock edge, the state machine moves to state SCL_LOW. In this state, the SCL line is held Low and the system clock counter begins counting. If the REP_START signal is asserted then the SDA signal will be set High, if the GEN_STOP signal is asserted, SDA will be set Low.

When the SCL low time has been reached, the state machine will transition to the IDLE state if arbitration has been lost and the byte transfer is complete to insure that SCL continues until the end of the transfer. Otherwise the next state is the SCL_HI_EDGE state.

The SCL_HI_EDGE state generates a rising edge on SCL by setting SCL to "1". Note, however, that the state machine will not transition to the SCL_HI state until the sampled SCL signal is also High to obey the clock synchronization protocol of the I2C specification. Clock synchronization is performed using the wired-AND connection of the SCL line. The SCL line will be held Low by the device with the longest low period. Devices with shorter low periods enter a high wait state until all devices have released the SCL line and it goes High. Therefore the SCL_HI_EDGE state operates as the high wait state as the SCL clock is synchronized.

The SCL_HI state then starts the system clock counter to count the high time for the SCL signal. If a repeated START or a STOP condition has been requested, the state machine will transition to the appropriate state after half of the SCL high time so that the SDA line can transition as required. If neither of these conditions has been requested, then the state machine transitions to the SCL_LOW_EDGE state when the SCL high time has been achieved.

I2C Interface Main State Machine

The main state machine for the I2C Interface logic is shown in Figure 7. This state machine is the same for both Slave and Master modes. In each state, the mode is checked to determine the proper output values and next state conditions. This allows for immediate switching from Master to Slave mode if arbitration is lost or if the CoolRunner I2C Controller is addressed as a Slave.

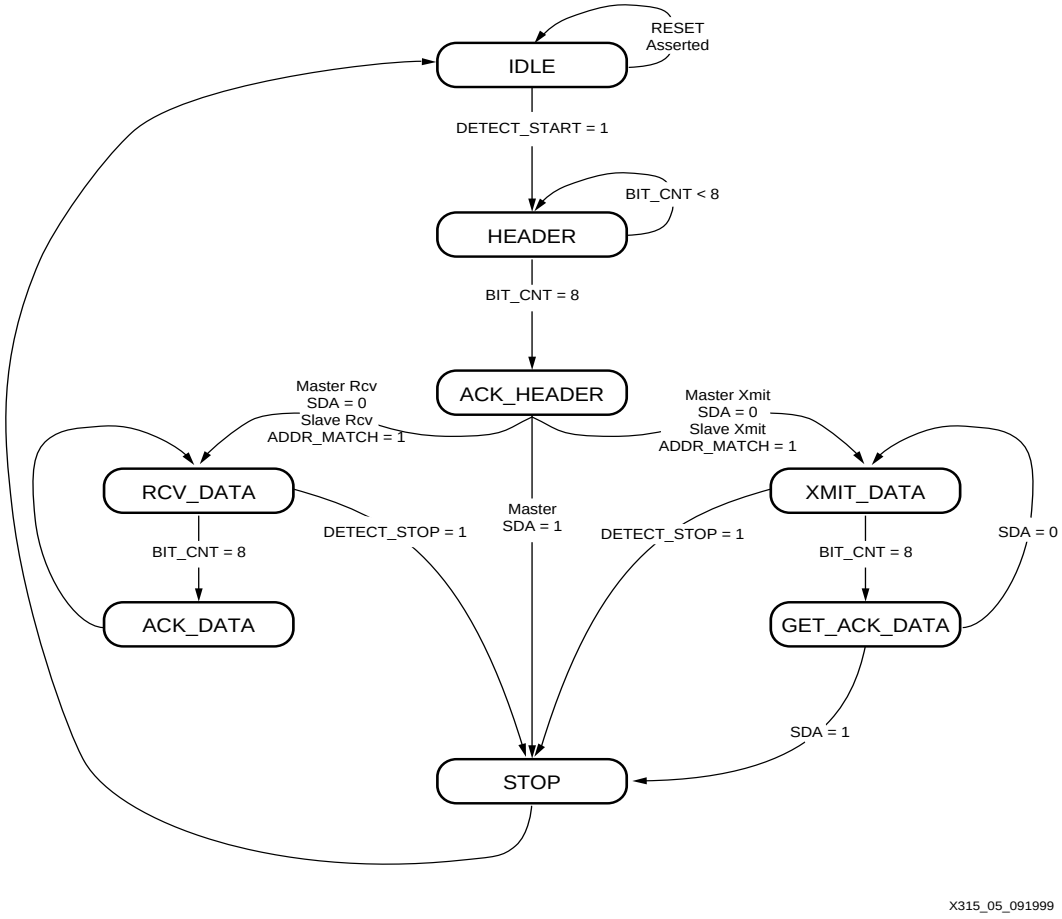

Figure 7: I2C Interface Main State Machine

This state machine utilizes and controls a counter that counts the I2C bits that have been received. This count is stored in the signal BIT_CNT. This state machine also controls two shift registers, one that stores the I2C header that has been received and another that stores the I2C data that has been received or is to be transmitted.

Note: This state machine and the associated counters and shift registers are clocked on the falling edge of the incoming SCL clock. If the load is heavy on the SCL line, the rise time of the SCL signal may be very slow which can cause susceptibility to noise for some systems. This can be particularly dangerous on a clock signal. The designer is strongly encouraged to investigate the signal integrity of the SCL line and if necessary, use external buffers for the SCL signal.

When a START signal has been detected, the state machine transitions from the IDLE state to the HEADER state. The START signal detection circuit monitors the incoming SDA and SCL lines to detect the START condition. The START condition can be generated by the CoolRunner I2C controller or another Master - either source will transition the state machine to the HEADER state.

The HEADER state is the state where the I2C header is transmitted on the I2C bus from the MBDR register if in Master mode. In this state, the incoming I2C data is captured in the I2C Header shift register. In Master mode, the I2C Header shift register will contain the data that

was just transmitted by this design. When all eight bits of the I2C header have been shifted in, the state machine transitions to the ACK_HEADER state.

In the ACK_HEADER state, the CoolRunner I2C design samples the SDA line if in Master mode to determine whether the addressed I2C Slave acknowledged the header. If the addressed Slave does not acknowledge the header, the state machine will transition to the STOP state which signals the SCL/START/STOP generator to generate a STOP. If the addressed Slave has acknowledged the address, then the LSB of the I2C header is used to determine if this is a transmit or receive operation and the state machine transitions to the appropriate state to either receive data, RCV_DATA, or to transmit data, XMIT_DATA.

The I2C Header shift register is constantly compared with the I2C address set in the MADR register. If these values match in the ACK_HEADER state, the CoolRunner I2C Controller has been addressed as a Slave and the mode immediately switches to Slave mode. The MAAS bit is then set in the MBSR status register. The SDA line will be driven as set in the TXAK register to acknowledge the header to the current I2C bus Master. Again, the LSB of the I2C header is used to determine the direction of the data transfer and the appropriate state is chosen.

The RCV_DATA state shifts the incoming I2C data into the I2C shift register for transfer to the µC. When the whole data byte has been received, the state machine transitions to the ACK_DATA state and the value of the TXAK register is output on the SDA line to acknowledge the data transfer. Note that in Master mode, the indication that the Slave has transmitted the required number of data bytes is to not acknowledge the last byte of data. The µC must negate the TXAK bit to prohibit the ACK of the last data byte. The state machine exits this pair of states when a STOP condition has been detected, otherwise, the transition between these two states continues. In Master mode, the µC requests a STOP condition by negating the MSTA bit.

The XMIT_DATA state shifts the data from the I2C data register to the SDA line. When the entire byte has been output, the state machine transitions to the GET_ACK_DATA state. If an acknowledge is received, the state machine goes back to the XMIT_DATA to transmit the next byte of data. This pattern continues until either a STOP condition is detected, or an acknowledge is not received for a data byte.

Note that the data transfer states of this state machine assume that the µC can keep up with the rate at which data is received or transmitted. If interrupts are enabled, an interrupt is generated at the completion of each byte transfer. The MCF bit is set as well providing the same indication. Data is transferred to/from the I2C data register to/from the µC data register during the acknowledge cycle of the data transfer. The state machine does not wait for an indication that the µC has read the received data or that new data has been written for transmission. The designer should be aware of the effective data rate of the µC to insure that this is not an issue.

The STOP state signals the SCL/START/STOP generator to generate a STOP condition if the CoolRunner I2C design is in Master mode. The next state is always the IDLE state and the I2C activity is completed.

Operational Flow Diagrams

The flow of the interface between the µC and the CoolRunner I2C Controller is detailed in the following flow charts. These flow charts are meant to be a guide for utilizing the CoolRunner I2C Controller in a µC system.

Initialization

Before the CoolRunner I2C Controller can be utilized, certain bits and registers must be initialized as shown in Figure 8.

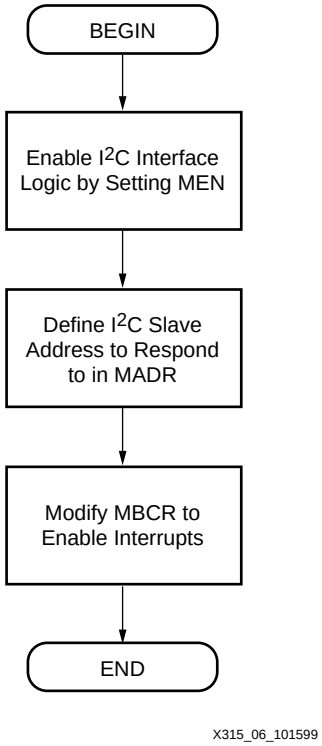

Figure 8: CoolRunner I2C Controller Initialization Flow Chart

Master Transmit/Receive

The flow charts for transmitting data and receiving data while I2C bus Master are shown in Figure 9 and Figure 10. The major difference between transmitting and receiving is the additional step in the Master Receive flow chart of turning off the acknowledge bit on the second to last data word.

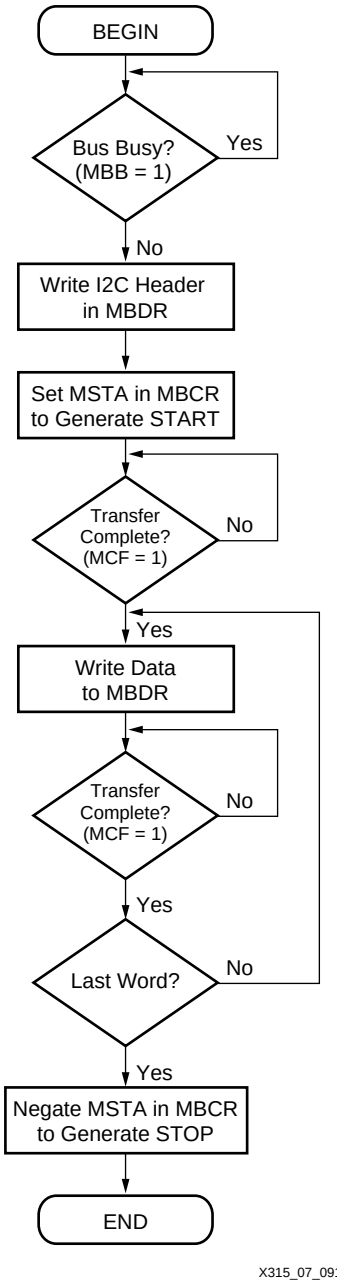

Figure 9: Master Transmit Flow Chart

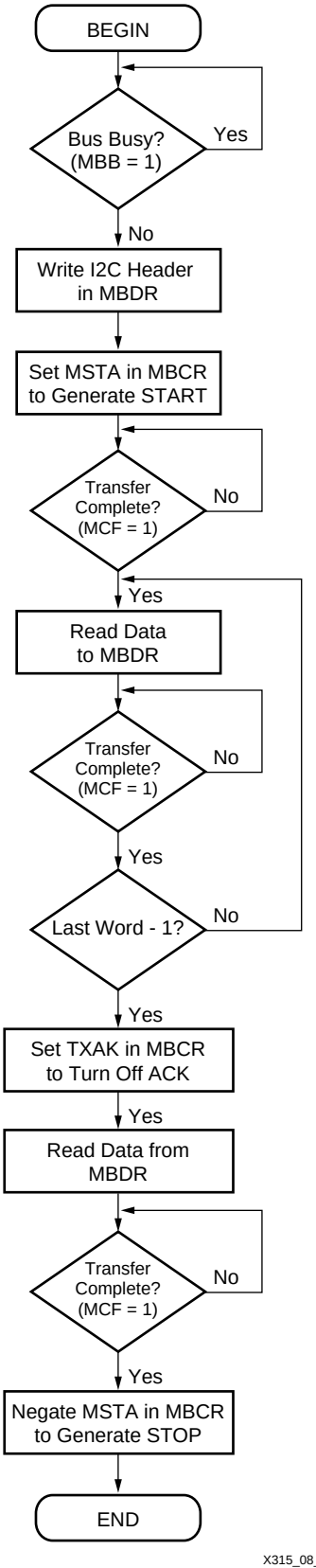

Figure 10: Master Receive Flow Chart

Slave Flow Chart

The flow chart for receiving or transmitting data in Slave mode is shown in Figure 11. If in receive mode, the first read from the MBDR register is a dummy read because data has not yet been received. Since the CoolRunner I2C Controller is in Slave mode, the only way to know that the transaction is complete is to check that the bus is busy and that the Addressed as Slave bit is still set.

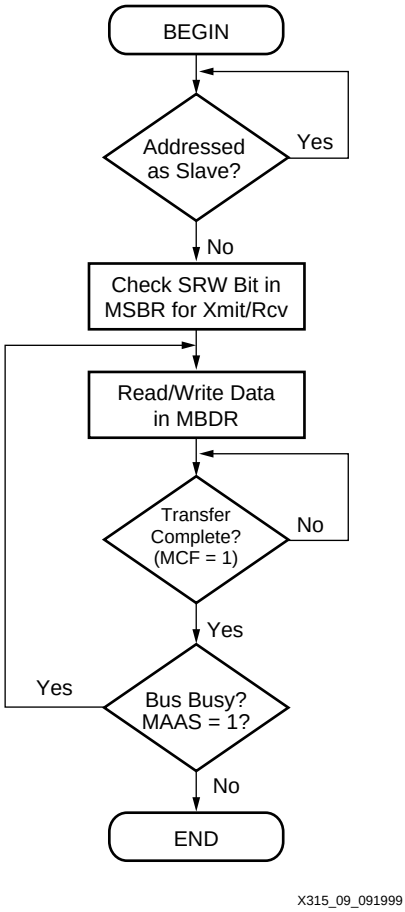

Figure 11: Slave/Transmitter Flow Chart

CoolRunner Implementation

The design of the CoolRunner I2C Controller was implemented in VHDL and synthesized using Exemplar Galileo V4.2.2. The EDIF file generated by Exemplar was imported into XPLA Professional for compilation and fitting into a CoolRunner CPLD. The design was targeted to a 3V, 128 macrocell, enhanced clocking CoolRunner CPLD in a 100-pin TQFP package (XCR3128A-7VQC).

Notes:1. Since the system clock frequency was 1.832 MHz, the speed of the design was not critical and any speed grade part could have been used.

2. The I2C SCL line is used as a clock input into the CoolRunner I2C Controller. If there are many loads on the I2C bus, the rise time of the SCL line can be quite slow. The CoolRunner CPLD for this design requires a rise time no greater than 100 nS, therefore, the designer is strongly encouraged to examine the characteristics of the SCL signal in the I2C system. If the rise time of the I2C signals are greater than 100 ns, external buffers can be used between the actual I2C bus connections and the CoolRunner CPLD.

The XPLA Professional compiler and fitter properties were set as shown in Figure 12. Pin assignments were not set on the CoolRunner I2C Controller design prior to compiling and fitting the design.

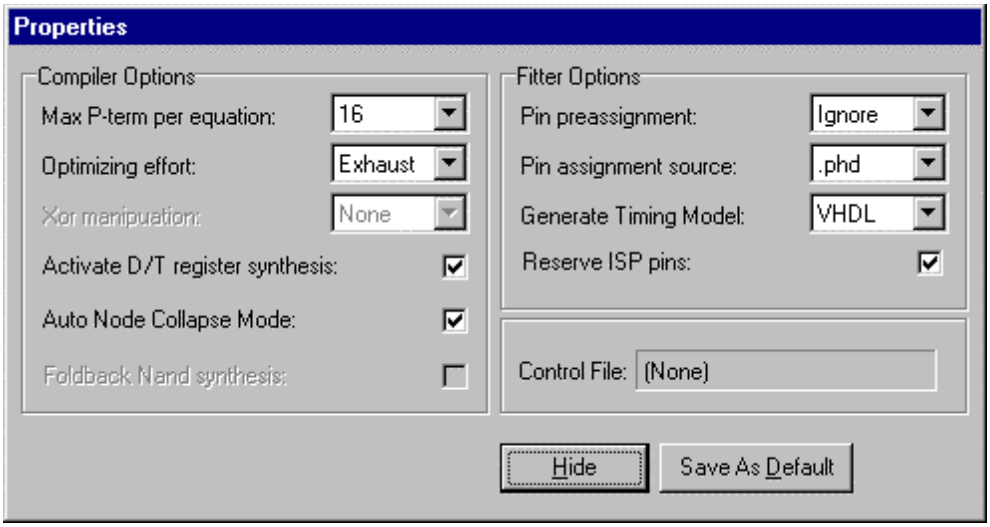

Figure 12: XPLA Professional Compiler/Fitter Properties

Due to the complexity of the CoolRunner I2C design, the fan-in parameters for the compiler (synthesis) and fitter had to be adjusted as shown in Figure 13. Adjusting the Synthesis fan-in instructs the compiler as to how many signals are allowed as fan-in to each equation. Reducing this number to 16 forced the compiler to create some internal nodes and therefore reduced the total fan-in required for each logic block. The Fitter fan-in was increased to utilize additional hardware routing resources to each logic block present in a CoolRunner CPLD. This menu can be opened by typing Ctrl-Alt-Z while in the XPLA Professional GUI.

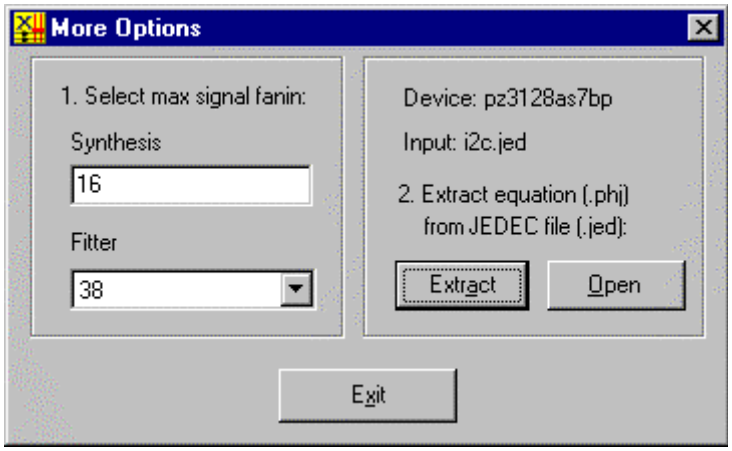

Figure 13: Adjusting Compiler and Fitter Fan-in

Design Verification

The XPLA Professional 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. Also, the CoolRunner I2C Controller design was simulated with an independently generated VHDL model of an I2C Slave design to verify that the interface specifications were implemented correctly. Please note that all verification of this design has been done through simulations.

Revision History

Date	Version #	Revision
10.225.99	1.0	Initial Xilinx release.