XAPP353 (v1.0) February 14, 2001

# CoolRunner XPLA3 SMBus Controller Implementation


## Summary

This document details the VHDL implementation of an system Management Bus (SMBus) controller in a Xilinx CoolRunner® XPLA3™ 256-macrocell CPLD. CoolRunner CPLDs are the lowest power CPLDs available, making this the perfect target device for an SMBus controller. To obtain the VHDL code described in this document, go to section **VHDL Code Disclaimer and Download Instructions**, page 20 for instructions.


## Introduction

The SMBus is a derivative of the $I^2C$ bus and is a serial, two-wire interface used in many rechargeable (secondary) battery 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 10-100 KHz operation, each device connected to the bus is software addressable by a unique address with a simple Master/Slave protocol.

The CoolRunner SMBus Controller design contains a synchronous microcontroller (µC) interface and provides SMBus Master/Slave capability. It is intended to be used with a microcontroller (µC) or microprocessor (µP) as shown in Figure 1. The microprocessor interface used in this design is the Hitachi SH7750 but can be easily changed to accommodate other processor or controller types.

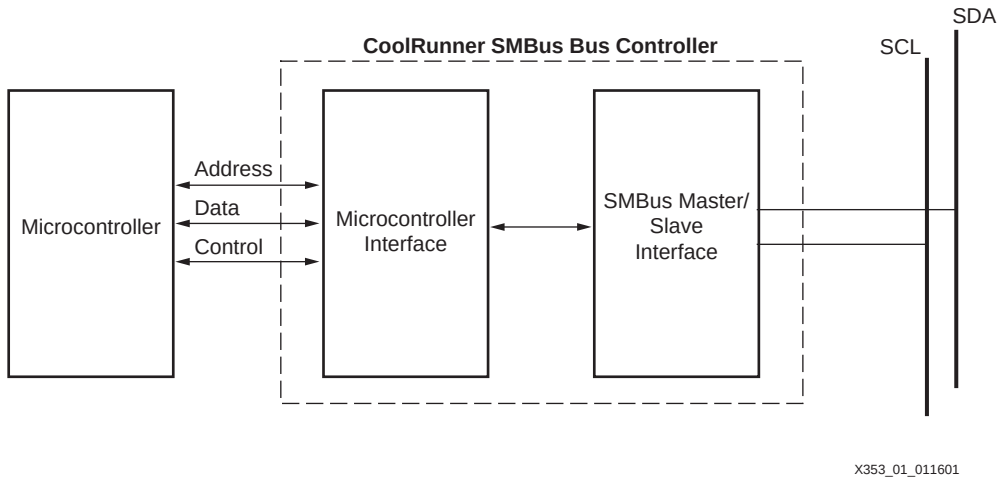


*Figure 1:* **CoolRunner SMBus Controller**

## SMBus Background

This section describes the main protocol of the SMBus. For more details and timing diagrams, please refer to the System Management Bus Specification Revision 1.1 at: http://www.smbus.org/specs/smbus110.pdf. This implementation of the SMBus controller conforms to Revision 1.1.

The SMBus consists of two wires, serial data (SDA), and serial clock (SCL), which carry information between the devices connected to the bus. There are two optional wires, SMBSUS# and SMBALERT# as described in the SMBus specification, which are not included in this implementation of the SMBus controller, but can easily be added by the user. The

SMBus specification does not limit the bus capacitance unlike the $I^2C$ specification. Instead the specification limits pull-down current to 350 µA. 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, Slaves, or Hosts. 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 SMBus protocol defines an arbitration procedure that insures that only is allowed to control he bus at any given time. The arbitration and clock synchronization procedures defined in the SMBus specification are supported by the CoolRunner SMBus Controller.

Data transfers on the SMBus 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.

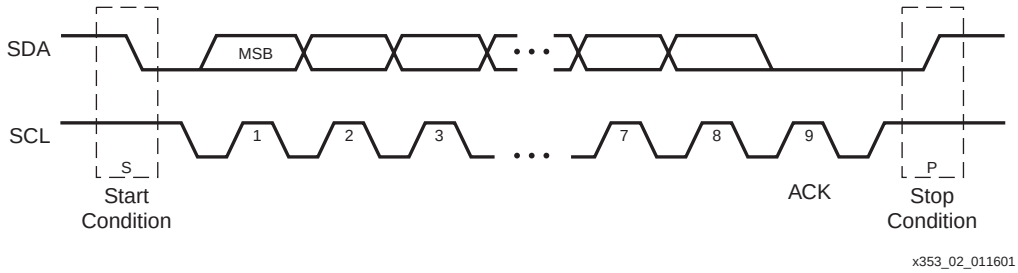


*Figure 2:* **Data Transfer on the SMBus**

Each data packet on the SMBus 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. However, the Slave may hold the SCL line Low for an extended period of time within the limits of the specification to complete a task. All Slaves are required to acknowledge their own address as a method for the Master to determine the presence of detachable devices on the bus. 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.

The SMBus specification provides an optional error checking mechanism using Packet Error Checking (PEC) which is derived from a CRC-8 polynomial. Any SMBus Master is permitted to determine if a Slave is PEC compatible. The Master will do this by acknowledging the Slave on the last bit of the transfer and supplying additional clocks to attempt to clock out the PEC data. This is done in lieu of the normally applied not-acknowledge bit at the end of a transfer as described earlier. Once the Master determines this, it is required to register the PEC capabilities of the Slave for future reference. This implementation of the SMBus controller does not support PEC, but will safely ignore an attempt by another Master to clock out the PEC.

Standard communication on the bus between a Master and a Slave is composed of four parts: START, Slave address, data transfer, and STOP. The SMBus protocol defines a data transfer format for both 7-bit and 10-bit addressing. The implementation of the SMBus controller in the Xilinx CoolRunner CPLD supports the 7-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 SMBus Controller

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

- Microcontroller interface
- Master, Slave, or Host 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
- 10-100 KHz operation

## Signal Descriptions

The I/O signals of the CoolRunner SMBus 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 SMBus Controller Signal Description**

| Name | Direction | Description |
|---|---|---|
| SDA | Bidirectional | SMBus Serial Data. |
| SCL | Bidirectional | SMBus Serial Clock. |
| ADDR_BUS[7:0] | Input | **µC Address Bus.** |
| DATA_BUS[7:0] | Bidirectional | **µC Data Bus**. |
| BS_N | Input | **Bus Cycle Start**. Active Low µC signal that indicates the start of the bus cycle and is asserted during each bus cycle. |
| CS_N | Input | **Chip Select Signal.** Active Low µC signal indicating device being selected. |
| RD_WRN | Input | **Read/Write.** "1" indicates a read, "0" indicates a write. |
| RD_N | Input | **Read.** Active Low strobe signal that indicates a read cycle. |
| WE_N | Input | **Write.** Active Low strobe signal that indicates a write cycle. |

*Table 1:* **CoolRunner SMBus Controller Signal Description *(Continued)***

| | | |
|---|---|---|
| RESET | Input | **Reset.** Active Low signal that resets the SMBus controller. |
| RDY_N | Output | **Ready.** Active Low wait state request signal. |
| IRQ | Output | **Interrupt Request.** Active Low. |
| MCF | Output | **Data Transferring Bit.** While one byte of data is being transferred over the SMBus, 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 66 KHz SCL signal assumes the frequency to be 66 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 SMBus Controller, shown in Figure 3 was broken into two major blocks, the µC interface and the SMBus interface.

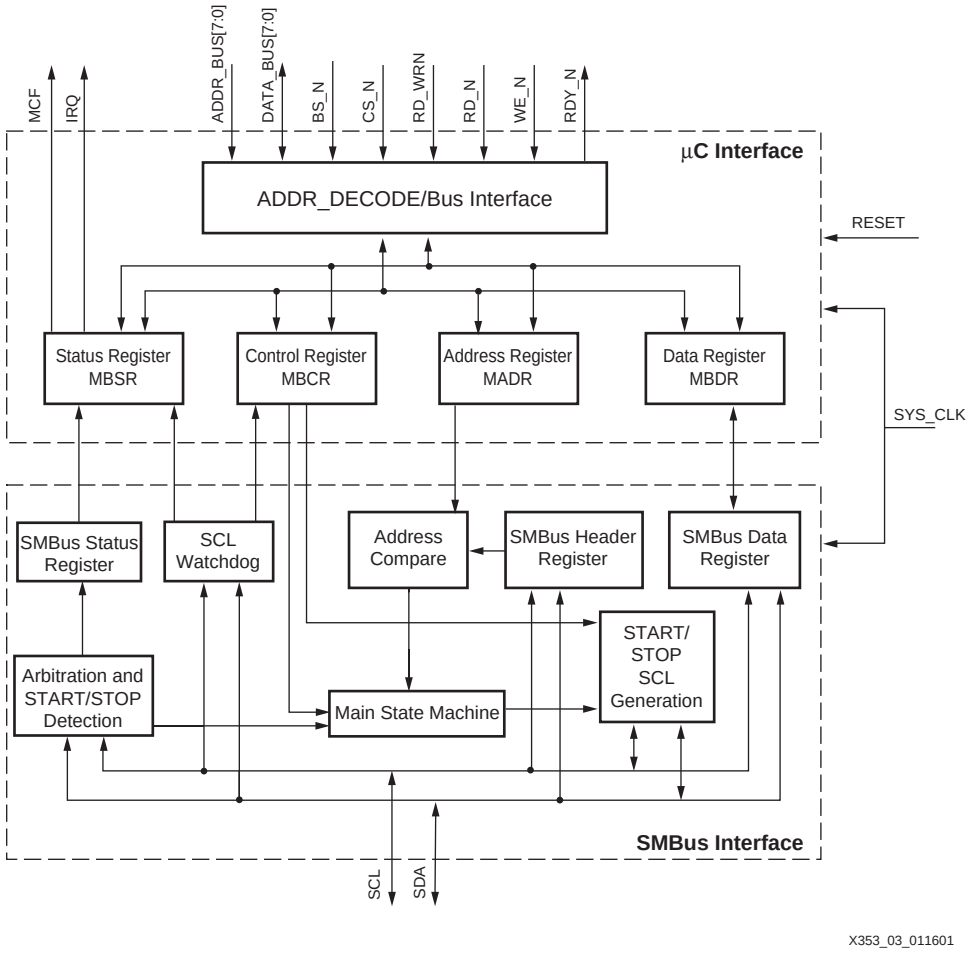


*Figure 3:* **CoolRunner SMBus Controller**

## Microcontroller Logic

The µC interface for the SMBus controller design supports a synchronous 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.

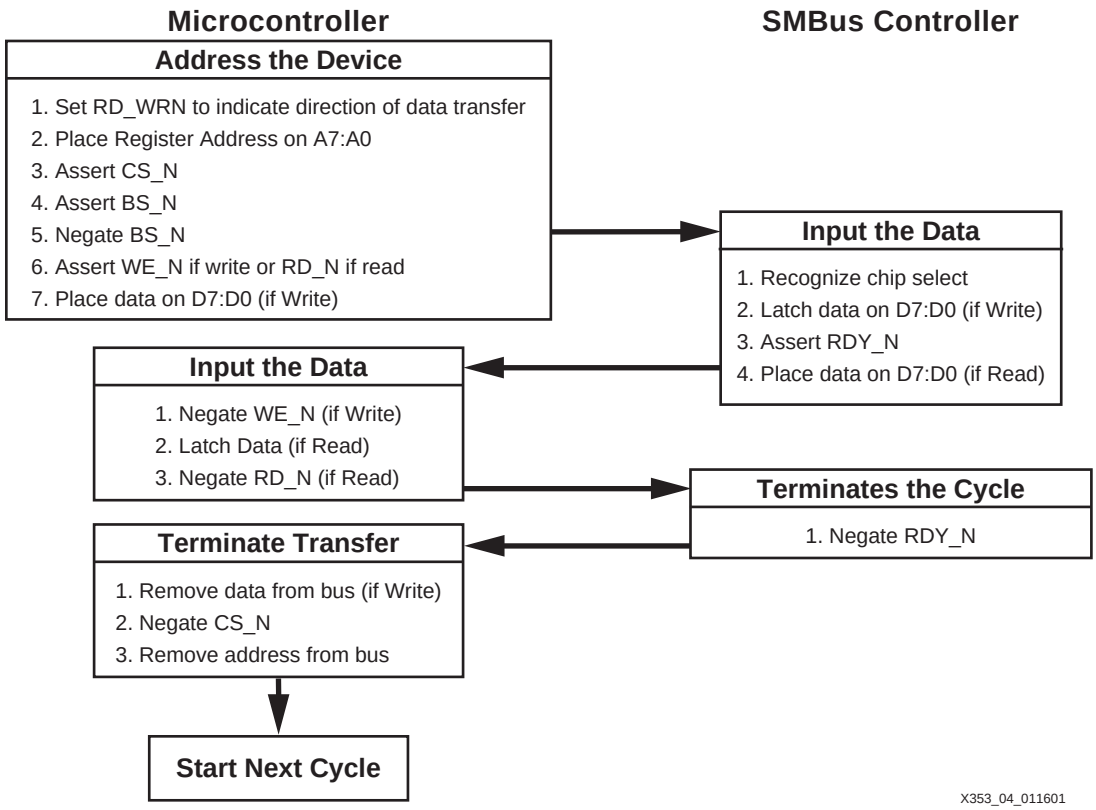


*Figure 4:* **µC Read/Write Protocol**

## Address Decode/Bus Interface Logic

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

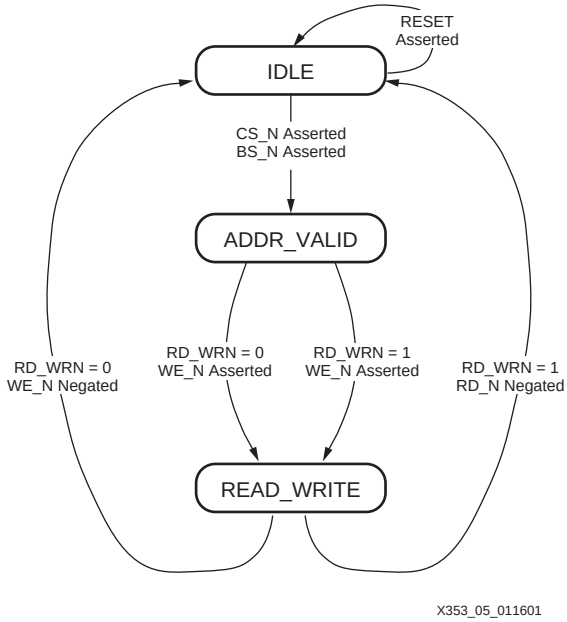


*Figure 5:* **µC Bus Interface State Machine**

In the first cycle, IDLE, the µC places the address on the address bus, sets the read/write line, RD_WRN, to the correct state, and asserts bus cycle start (BS_N) and chip select (CS_N). BS_N indicates the start of the bus cycle.

Upon the assertion of BS_N and CS_N, the CoolRunner SMBus Controller transitions to the ADDR_VALID state to decode the register address. The enables for the internal registers are set in this state. BS_N is then negated by the µC. If this is a write cycle, the µC asserts WE_N. If this is a read cycle, the µC asserts RD_N.

The CoolRunner SMBus Controller automatically progresses to the READ_WRITE state upon assertion of WE_N or RD_N. The SMBus Controller asserts RDY_N indicating to the µC that the is ready for data or has placed data on the bus. If this is a read cycle, the µC 3-states the data bus and the requested data is then placed on the bus by the SMBus Controller. If this is a write cycle, the SMBus Controller latches the data on the data bus into the addressed SMBus internal register.

The RD_WRN line is set to write and BS_N and CS_N are negated to indicate that the data transfer is complete. The negation of BS_N and CS_N causes the CoolRunner SMBus Controller to transition to the IDLE state upon which RDY_N is negated indicating that the cycle is complete.

## CoolRunner SMBus Controller Registers

The SMBus Controller is addressed by the active low chip select signal CS_N. Therefore, only the lower eight bits of the address bus are used to determine which internal register is being accessed.

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

*Table 2:* **SMBus Controller Registers**

| Address | Register | Description |
|---|---|---|
| MBASE + $8Ch | MADR | SMBus Address Register |
| MBASE + $90h | MBCR | SMBus Control Register |
| MBASE + $92h | MBSR | SMBus Status Register |
| MBASE + $94h | MBDR | SMBus Data I/O Register |

### Address Register (MADR)

This field contains the specific Slave address to be used by the SMBus 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 SMBus controller when in Slave mode. |
| 0 | - | - | Unused |

### Control Register (MBCR)

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

*Table 4:* **Control Register Bits**

| Bit Location | Name | µC Access | Description |
|---|---|---|---|
| 7 | MEN | Read/Write | **SMBus Controller Enable.** This bit must be set before any other MBCR bits have any effect<br>"1" enables the SMBus controller<br>"0" resets and disables the SMBus controller |
| 6 | MIEN | Read/Write | **Interrupt Enable.**<br>"1" enables interrupts. An interrupt occurs if MIF bit in the status register is also set<br>"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 SMBus controller generates a START condition in Master mode. When this bit is cleared, a STOP condition is generated and the SMBus 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. |

*Table 4:* **Control Register Bits *(Continued)***

| Bit Location | Name | µC Access | Description |
|---|---|---|---|
| 4 | MTX | Read/Write | **Transmit/Receive Mode Select.** This bit selects the direction of Master/Slave transfers.<br>"1" selects an SMBus Master transmit<br>"0" selects an SMBus Master 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<br>"1" - ACK bit = "1" - no acknowledge<br>"0" - ACK bit = "0" - acknowledge<br>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 SMBus 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 SMBus 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 one byte of data is being transferred, this bit is cleared. It is set by the rising edge of SCL during the acknowledge cycle of the transfer and is only High for this SCL clock period.<br>"1" transfer is complete<br>"0" transfer in progress<br>Note that in the CoolRunner SMBus 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 SMBus matches the Slave address in the MADR register, the SMBus 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 SMBus. This bit is set when a START condition is detected and cleared when a STOP condition is detected.<br>"1" indicates the bus is busy<br>"0" indicates the bus is idle |
| 4 | MAL | Read Software Clearable | **Arbitration Lost Bit.** This bit is set by hardware when arbitration for the SMBus is lost. This bit must be cleared by the mC software writing a "0" to this bit. |
| 3 | Reserved | - | - |

*Table 5:* **Status Register Bits *(Continued)***

| Bit Location | Name | µC Access | Description |
|---|---|---|---|
| 2 | SRW | Read | **Slave Read/Write Bit.** When the SMBus 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.<br>"1" indicates Master reading from Slave<br>"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.<br>"1" indicates that no acknowledge was received<br>"0" indicates that an acknowledge was received |

### Data Register (MBDR)

This register contains data to/from the SMBus. Physically, this register is implemented by two byte-wide registers at the same address, one for the SMBus transmit data and one for the SMBus received data. This eliminates any possible contention between the µC and the CoolRunner SMBus 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 SMBus, in receive mode, this register contains the data received from the SMBus. Note that in receive mode, it is assumed that the µC will be able to read this register during the next SMBus transfer. The received SMBus data is placed in this register after each complete transfer, the SMBus 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:* **SMBus Data Register Bit**

| Bit Location | Name | µC Access | Description |
|---|---|---|---|
| 7-0 | D7:D0 | Read/Write | SMBus Data |

## SMBus Interface Logic

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

### Arbitration

Arbitration of the SMBus 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 SMBus 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 SMBus Controller switches to Slave mode and resets the MSTA bit.

The CoolRunner SMBus 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 SMBus 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 SMBus Controller is the generator of these conditions.

## SCL Watchdog

This process monitors the SCL and SDA bus values to determine if either of two timing conditions have been violated:

- The SMBus specification requires that when SCL has been low for 25 ms, all devices must reset their communication. When this condition has been met, the MEN bit of the Control Register is cleared by this process which resets the SMBus when in either master or slave mode of operation. The constant BUS_LOW_TIMEOUT in the VHDL source code can be modified to compensate for system clock frequencies other than the current 66 MHz.
- The SMBus specification also requires that when SCL and SDA have been high for 50 ms the SMBus is considered free. This process will then clear the MBB bit (Bus Busy Bit) of the Status Register indicating the bus is no longer busy. The constant BUS_HIGH_TIMEOUT in the VHDL source code can be modified to compensate for system clock frequencies other than the current 66 MHz.

## Generation of SCL, SDA, START and STOP Conditions

This process generates the SCL and SDA signals output on the SMBus when in Master mode. The clock frequency of the SCL signal can be anywhere from 10 KHz to 100 KHz and is determined by dividing down the input clock. Several constants in the VHDL source code determine the timing of SCL and SDA to meet the SMBus Specification requirements. The number of input clock cycles required for generation of a 10-100 KHz SCL signal is set by the constants HIGH_CNT (High cycle of SCL) and LOW_CNT (Low cycle of SCL) and is currently calculated for a system clock of 66 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 SMBus requirements on hold time for the SDA lines after generating a START condition and after outputting data. The constant TBUF must specifies the number of system clock cycles required to meet the minimum bus free time between stop and start conditions. Similarly, the constant STOP_REP_START_SU determines the setup time for repeated start and stop conditions.

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.

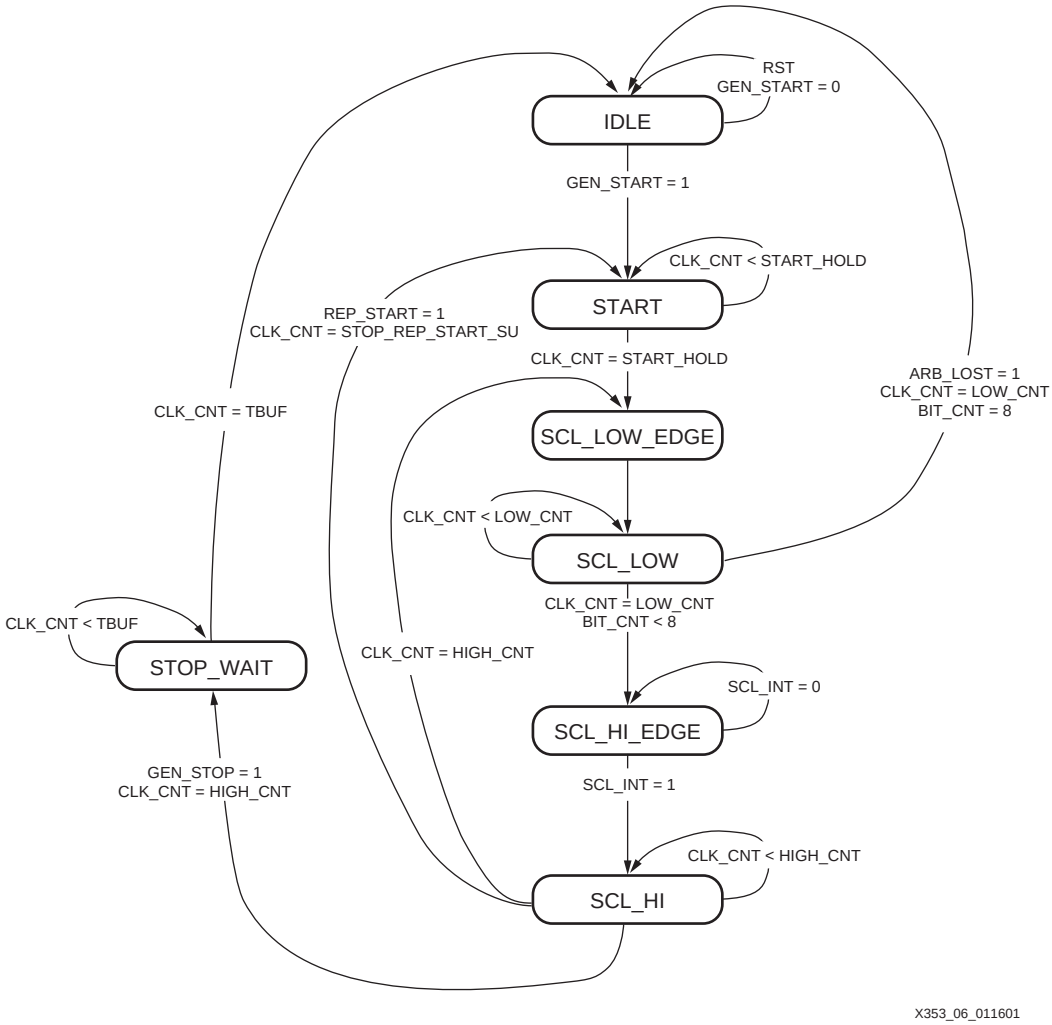


*Figure 6:* **SCL, SDA, START, and STOP Generation State Machine**

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 SMBus 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 SMBus Controller 3-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 3-state buffer for these outputs.

In the IDLE state, SCL and SDA are 3-stated, allowing any Master to control the bus. Once a request has entered to generate a start condition, the CoolRunner SMBus 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. If this had been a Repeated START condition, the RSTA bit in the Control Register is cleared.

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 SMBus 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 SCL has been High for the amount of time specified by the constant STOP_REP_START_SU. 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.

The STOP_WAIT state is used to insure that the hold time requirement after a STOP condition is met as specified by the constant TBUF.

## SMBus Interface Main State Machine

The main state machine for the SMBus 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 SMBus Controller is addressed as a Slave.

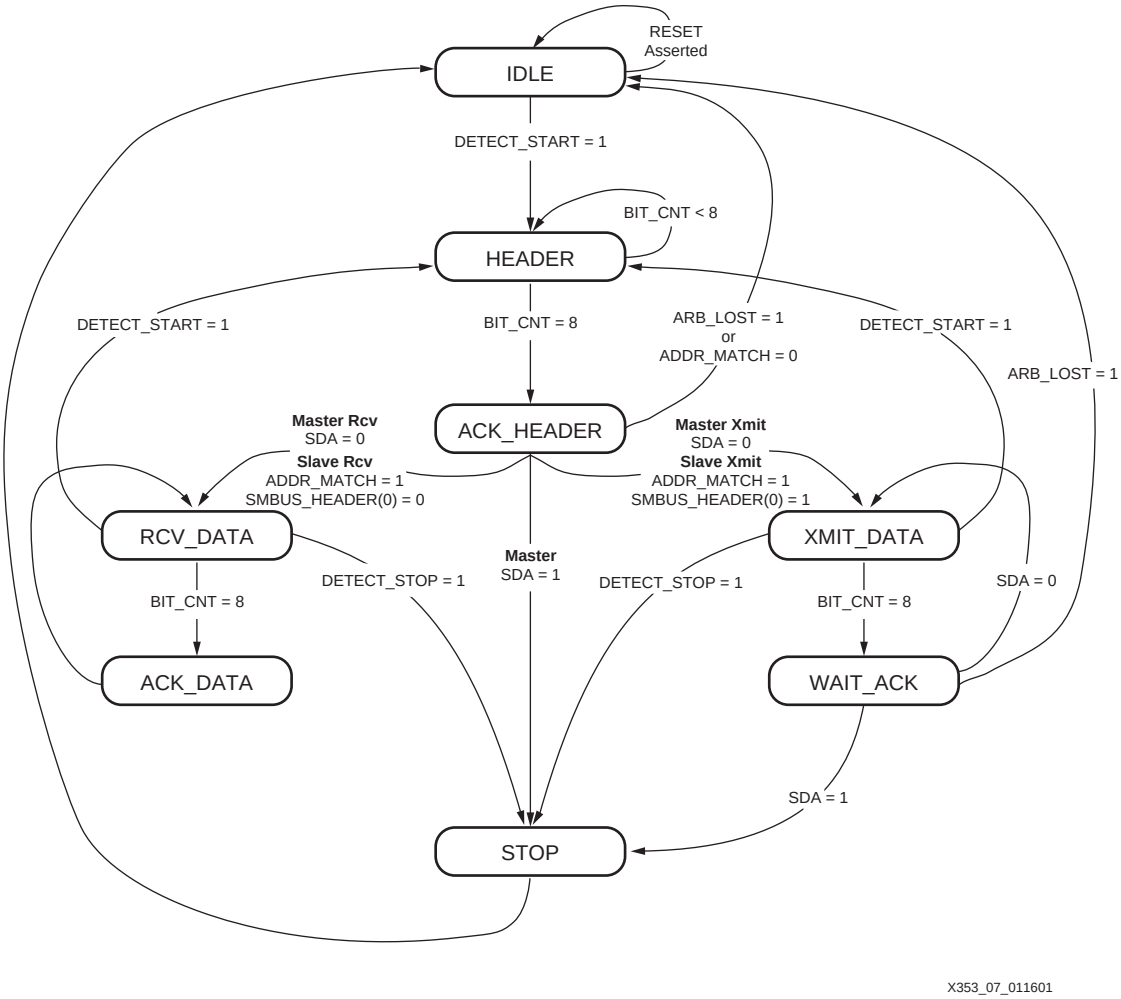


*Figure 7:* **SMBus Interface Main State Machine**

This state machine utilizes and controls a counter that counts the SMBus 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 SMBus header that has been received and another that stores the SMBus 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 SMBus controller or another Master—either source will transition the state machine to the HEADER state.

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

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

In the ACK_HEADER state, the CoolRunner SMBus design samples the SDA line if in Master mode to determine whether the addressed SMBus 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 SMBus 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 SMBus Header shift register is constantly compared with the SMBus address set in the MADR register. If these values match in the ACK_HEADER state, the CoolRunner SMBus 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 SMBus Master. Again, the LSB of the SMBus header is used to determine the direction of the data transfer and the appropriate state is chosen.

The RCV_DATA state shifts the incoming SMBus data into the SMBus 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 SMBus data register to the SDA line. When the entire byte has been output, the state machine transitions to the WAIT_ACK 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 SMBus 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 SMBus design is in Master mode. The next state is always the IDLE state and the SMBus activity is completed.

## Operational Flow Diagrams

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

## Initialization

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

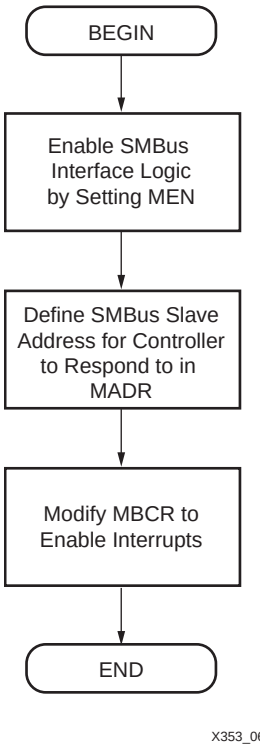


*Figure 8:* **CoolRunner SMBus Controller Initialization Flow Chart**

## Master Transmit/Receive

The flow charts for transmitting data and receiving data while SMBus 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.

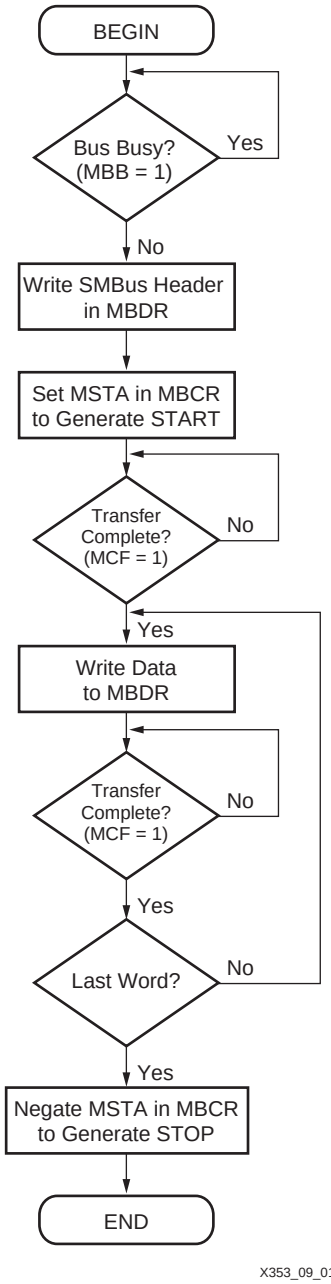


*Figure 9:* **Master Transmit Flow Chart**

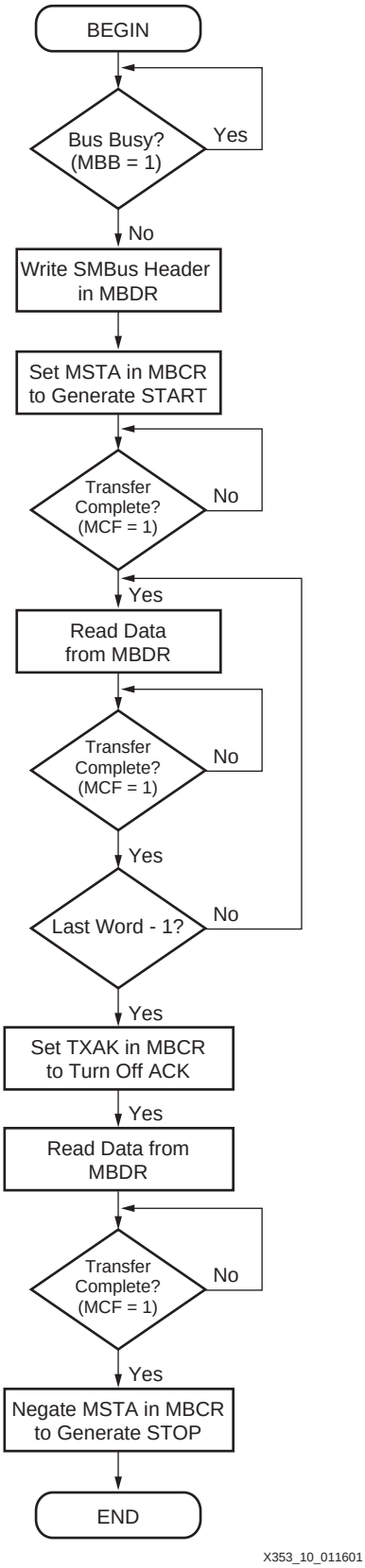


*Figure 10:* **Master Receive Flow Chart**

## 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 SMBus 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.

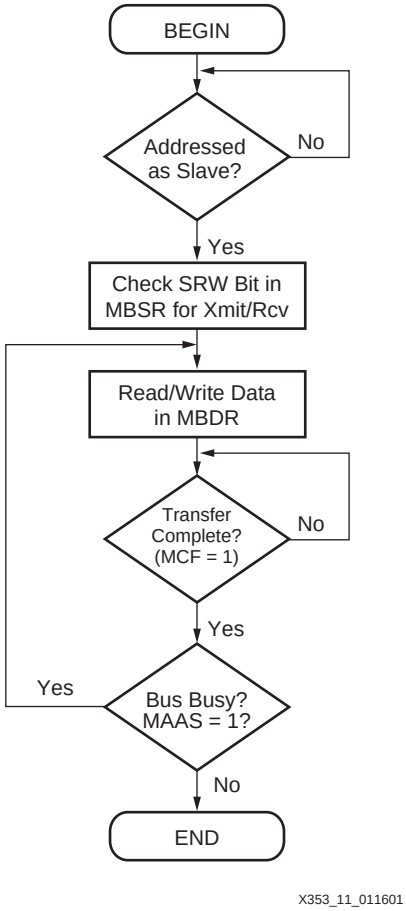


*Figure 11:* **Slave/Transmitter Flow Chart**

# CoolRunner XPLA3 CPLD Implementation

The design of the CoolRunner SMBus Controller was implemented in VHDL and targeted to a 256 macrocell CoolRunner CPLD in a 144-pin TQFP package (XCR3256XL-7TQ144C) using Xilinx Project Navigator. (Xilinx Project Navigator software is available free-of-charge from the Xilinx website: www.xilinx.com/products/software/webpowered.htm.).

**Note**:

Since the system clock frequency was 66 MHz, the speed of the design is critical and therefore, only the -7 and -10 parts can be used at this frequency. The system clock frequency can be changed by the user which may allow or prohibit the use of other speed grades.

**Note**:

The SMBus SCL line is used as a clock input into the CoolRunner SMBus Controller. If there are many loads on the SMBus, 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 SMBus system. If the rise time of the SMBus signals are greater than 100 ns, external buffers can be used between the actual SMBus connections and the CoolRunner CPLD.

The SMBus design utilization in an XPLA3 256-macrocell device is shown in Table 7. This utilization was achieved using certain fitter parameters, actual results may vary. As shown, there is plenty of room remaining in the device for the implementation of other logic in the system.

*Table 7:* **CoolRunner XPLA3 256-Macrocell Utilization**

| Resource | Available | Used | Utilization |
|---|---|---|---|
| Function Blocks | 16 | 14 | 87.50% |
| Macrocells | 256 | 178 | 69.54% |
| P-terms | 768 | 446 | 58.08% |
| Fold-back NANDs | 128 | 0 | 0.00% |
| I/O Pins | 116 | 27 | 23.28% |

### Design Verification

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. Please note that all verification of this design has been done through simulations.

## VHDL Code Disclaimer and Download Instructions

VHDL source code and test benches are available for this design. THIRD PARTIES MAY HAVE PATENTS ON THE SYSTEM MANAGEMENT BUS. BY PROVIDING THIS HDL CODE AS ONE POSSIBLE IMPLEMENTATION OF THIS STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THE PROVIDED IMPLEMENTATION OF THE SMBus IS FREE FROM ANY CLAIMS OF INFRINGEMENT BY ANY THIRD PARTY. XILINX EXPRESSLY DISCLAIMS ANY WARRANTY OR CONDITIONS, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, AND XILINX SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE, THE ADEQUACY OF THE IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OR REPRESENTATION THAT THE IMPLEMENTATION IS FREE FROM CLAIMS OF ANY THIRD PARTY. FURTHERMORE, XILINX IS PROVIDING THIS REFERENCE DESIGNS "AS IS" AS A COURTESY TO YOU.

**XAPP353** - http://www.xilinx.com/products/xaw/coolvhdlq.htm

## Conclusion

This document has detailed the design of an SMBus Controller design 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 |
|---|---|---|
| 02/14/01 | 1.0 | Initial Xilinx release. |