I2C Bus Controler Implementation

Dimitar Dimitrov, Anna Andonova, Nikita Dimitrov, Petar Gugutkov

Abstract - **In this paper is discussed VHDL implementation of an I2C controller in a CPLD. 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 between IC so they have fewer pins, and the number of traces required on printed circuit boards is reduced. Capable of 100KHz in normal mode and 400KHz in fast mode, each device connected to the bus is software addressable by a unique address with a simple Master/Slave protocol.**

I. INTRODUCTION

The Inter Integration Circuit (I2C) bus is a popular serial, two-wire interface used in many systems because of its low overhead. All I2C-bus compatible devices incorporate an on-chip interface, which allows them to communicate directly with each other via the I2C-bus. This design concept solves the many interfacing problems encountered when designing digital control circuits.

Here are some of the features of the I2C-bus: Only two bus lines are required: a serial data line (SDA) and a serial clock line (SCL); Each device connected to the bus is software addressable by a unique address and simple master/ slave relationships exist at all times; Masters can operate as master-transmitters or as master-receivers; It's a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer; Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100 kbit/s in the standard mode or up to 400 kbit/s in the fast mode ;On-chip filtering rejects spikes on the bus data line to preserve data integrity ; The number of ICs that can be connected to the same bus is limited only by a maximum bus capacitance of 400 pF.

I2C-bus compatible ICs allow a system design to rapidly progress directly from a functional block diagram to a prototype. Moreover, since they 'clip' directly onto the I2C-bus without any additional external interfacing, they allow a prototype system to be modified or upgraded simply by 'clipping' or 'unclipping' ICs to or from the bus.

Some of the IC, mainly DSP and microcontrollers don't have incorporated an I2C driver, and can't take advantage of the features of the I2C bus during the communication with the other ICs. There is a two way to organize this communication. The first is to write a software driver, but this will take a lot of processor's resources. The second is to use some of the interfaces of the processor and adapt it to I2C standard.

Authors are with the Department of Microelectronics, Technical University of Sofia, FETT, Sofia, 1797, Bulgaria
E-mail: ava@ecad.vmei.acad.bg; mihotron@mail.bg

II. I2C BACKGROUND

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. 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 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.

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 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.

III. 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

A. Signal Description

The I/O signals of the 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.

B. Block Diagram

The block diagram of the I2C Controller, shown in Figure 1 was broken into two major blocks, the µC interface and the I2C interface.

C. Microcontroller interface logic

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 3-states the data bus and DS indicates that the I2C Controller can place data on the data bus. Upon the assertion of AS, the 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 I2C Controller is being addressed and DS is asserted, the 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.

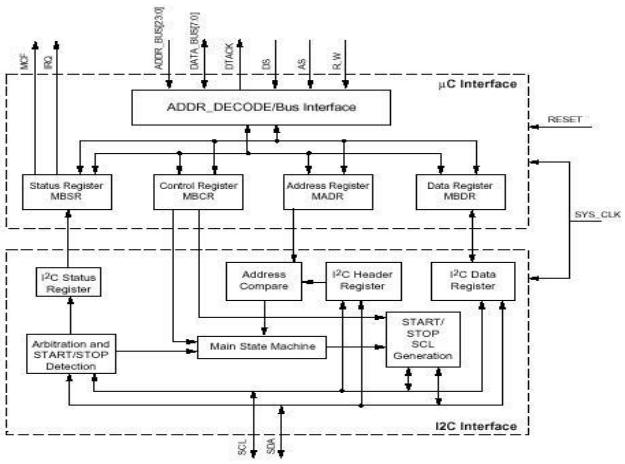

Fig. 1 Block Diagram of I2C controller

Name	**Direction**	**Description**
SDA	Bidirectional	**I2C Serial Data.**
SCL	Bidirectional	**I2C Serial Clock.**
ADDR_BUS[23:0]	Input	**Address Bus.**
DATA_BUS[7:0]	Bidirectional	**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 handshake signal indicating that the data present on the data bus is valid or that the microcontroller 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.

Tab.1 Signal Description of I2C controller

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 I2C Controller to negate DTACK and transition to the IDLE state.

D. I2C interface logic

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

D.1. 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 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. At this point, the I2C Controller switches to Slave mode. The I2C design will not generate a START condition while the bus is busy. If arbitration is lost during a byte transfer, SCL continues to be generated until the byte transfer is complete.

D.2. 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 I2C Controller is the generator of these conditions

D.3. 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 4 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. 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.

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 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. The STOP_WAIT state is used to insure that the hold time requirement after a STOP condition is met.

E. I2C interface main state machine

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 I2C Controller is addressed as a Slave. 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. 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 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 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, or to transmit data. The I2C Header shift register is constantly compared with the I2C address set in the MYADR register. If these values match in the ACK_HEADER state, the I2C Controller has been addressed as a Slave and the mode immediately switches to Slave mode. 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. 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 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 I2C data register to/from the µC data register during the acknowledge cycle of the data transfer.. The STOP state signals the SCL/START/STOP generator to generate a STOP condition if the I2C design is in Master mode. The next state is always the IDLE state and the I2C activity is completed.

IV. CONCLUSION

This I2C controller contains a micocontroller interface and provides I2C Master/Slave capability. It is intended to be used with microcontroller, microprocessor or DSP,and permit communicaton bitween them and integration cirquits to be done with standard READ/WRITE operation.

REFERENCES

[1] Philips Semiconductor:" I2C Specification"
[2] www.xilinx.com