Design and Implementation of I2C protocol

Bindu P¹, Jayachandra², Pooja Ullegaddi³, Srushty Prasad⁴, Mrs. Kiran Bailey⁵

¹,²,³,⁴,⁵ Dept of ECE, B.M.S College of Engineering

Abstract—The I2C protocol is a serial communication protocol for any interface that supports only two wires to connect devices like micro controllers and other I/O peripherals. It communicates with the microcontroller through ABP interface. This paper is aimed at the design and implementation of such protocol controller in slave mode. The design itself is a part of IP ‘Multi serial communication controller’. The design is following the standards set by Philips semiconductor for its I2C controller IC PCF9564. This design sets a way for the slow devices to communicate with fast devices connected on the BUS.

Index Terms— I2C , SDA ,SCL ,Interrupt , Time out ,ACK, NACK

I. INTRODUCTION

I2C (Inter-Integrated Circuit) is a multi-master, multi-slave, single-ended, serial computer bus invented by Philips Semiconductor (now NXP Semiconductors). It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers. Each device connected to the bus is software addressable by a unique address with a simple Master/Slave protocol. Fast devices are able to communicate with slow devices. Data is transferred between a Master and a Slave synchronously to SCL on the SDA line on a byte-by-byte basis. Each data byte is 8 bits long. There is one SCL clock pulse for each data bit with the MSB being transmitted first. An acknowledge bit follows each transferred byte. Each bit is sampled during the high period of SCL; therefore, the SDA line may be changed only during the low period of SCL and must be held stable during the high period of SCL. A transition on the SDA line while SCL is high is interpreted as a command. The command may be a START or a STOP.
II. BLOCK DIAGRAM

The main blocks in the design are:

- Configuration Registers and Register access control
  The configuration registers form the backbone of the system accommodating all the registers associated with the block. The access control involves the logic to access the registers based on address bus, read/write signal transitions.

- Interrupt logic
  Interrupt logic generates the interrupt based on the status codes loaded into one of the registers in the register block i.e. status register. The CPU disables the interrupt as soon as it is serviced.

- Clock divider and Selector
  This generates I2C clock with respect to which all the internal operations are carried out and also different SCL frequencies. One of the SCL frequencies is selected depending on the speed requirement.

- Start Stop detector
  It very important to detect commands like START and STOP on SDA and SCL lines when the controller is in slave mode. These commands are detected in this block to initiate and terminate the communications respectively.
Bus status logic
The I2C communication may be encountered with three types of errors: illegal start–stop, SDA struck low, SCL struck low. All these errors are detected and reported in bus status logic.

III. IMPLEMENTATION

1. Configuration Registers and Register Access Control

The controller contains four registers which are used to configure the operation of the device as well as to send and receive serial data. The registers are selected by setting pins A0 and A1 to the appropriate logic levels before a read or write operation is executed. It is summarized in the table shown below.

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>REGISTER NAME</th>
<th>REGISTER FUNCTION</th>
<th>READ/WRITE</th>
<th>DEFAULT VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>I2CSIA</td>
<td>STATUS REGISTER</td>
<td>READ ONLY</td>
<td>0x0B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>I2CTO</td>
<td>TIME-OUT REGISTER</td>
<td>WRITE ONLY</td>
<td>0xFF</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>I2CDAT</td>
<td>DATA REGISTER</td>
<td>READ/WRITE</td>
<td>0x00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>I2CADDR</td>
<td>OWN ADDRESS REGISTER</td>
<td>READ/WRITE</td>
<td>0x00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>I2CCON</td>
<td>CONTROL REGISTER</td>
<td>READ/WRITE</td>
<td>0x00</td>
</tr>
</tbody>
</table>

The following block diagram depicts the input and outputs of register access control block. The REG_ENABLE enables one of the registers depending on A0 and A1.

The types of configuration registers available in this controller are:

- Time out Register, I2CTO
  The time-out register is used to determine the maximum time that SCL is allowed to be LOW before the I2C state machine is reset. When the I2C interface is operating, I2CTO is loaded in the time-out Counter at every SCL transition.
The most significant bit of this register (TE) is used as a time-out enable/disable. A “1” will enable the time-out function. The time-out period = (I2CTO [6:0] + 1) *113.7 µs. The time-out value may vary and is an approximate value. The time out register is used when the Serial Input Output, in the master mode, wants to send a START condition and the SCL line is held LOW by some other device and when in the master mode, the time-out feature starts every time the SCL goes LOW. In slave mode, if the SDA or SCL held low for more than the timeout period bus error is generated.

- Control Register, I2CCON
  The APB interface can write to this 8-bit register. Two bits are affected by the SIO hardware: the SI bit is set when a serial interrupt is requested, and the STO bit is cleared when a STOP condition is present on the I2C-bus. A write to the control Register clears the SI bit and causes the Serial Interrupt line to be de-asserted and the next clock pulse on the SCL line to be generated. Since none of the registers should be written to via the parallel interface once the Serial Interrupt line has been de-asserted, all the other registers that need to be modified should be written to before the content of the control register is modified.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>E</td>
<td>S</td>
<td>I</td>
<td>C</td>
<td>R</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

SCL Clock Frequencies:

<table>
<thead>
<tr>
<th>CR2</th>
<th>CR1</th>
<th>CR0</th>
<th>SERIAL CLOCK FREQUENCY (kHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>330</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>288</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>217</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>148</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>66</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>59</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>44</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>36</td>
</tr>
</tbody>
</table>

- The Data Register, I2CDAT
  I2CDAT contains a byte of serial data to be transmitted or a byte which has just been received. In master mode, this includes the slave address that the master wants to send out on the I2C-bus, with the most significant bit of the slave address in the SD7 bit position and the Read/Write bit in the SD0 bit position. The CPU can read from and write to this 8-bit register while it is not in the process of shifting a byte. This occurs when SIO is in a defined state and the serial interrupt flag is set. Data in I2CDAT remains stable as long as SI is set. Whenever the SIO generates an interrupt, the I2CDAT registers contain the data byte that was just transferred on the I2C-bus. The I2CDAT register will capture the serial address as data when addressed via the serial bus. Also, the data register will continue to capture data from the serial bus during 38H so the I2CDAT register will need to be reloaded when the bus becomes free.
SD7 –SD0 are Eight bits to be transmitted or just received. A logic 1 in I2CDAT corresponds to a HIGH level on the I2C-bus, and a logic 0 corresponds to a LOW level on the bus.

- **The Address Register, I2CADR**
  I2CADR is not affected by the SIO hardware. contents of this register are irrelevant when SIO is in a master mode. In the slave modes, the seven most significant bits must be loaded with the microcontroller’s own slave address.

The most significant bit corresponds to the first bit received from the I2C-bus after a start condition. A logic 1 in I2CADR corresponds to a HIGH level on the I2C-bus, and a logic 0 corresponds to a LOW level on the bus. The least significant bit is not used but should be programmed with a ‘0’.

- **The Status Register, I2CSTA**
  It is an 8-bit read-only register. The three least significant bits are always zero. The five most significant bits contain the status code. There are 25 possible status codes. When I2CSTA contains F8H, no relevant state information is available and no serial interrupt is requested. All other I2CSTA values correspond to defined SIO states. When each of these states is entered, a serial interrupt is requested (SI=’1’). The 25 different codes are shown in the table above.

Implementation Logic of Register Access Control:

With respect to the functional table of this block, the logic has been derived which includes AND, OR and NOT gates which select the required reg_enable bit to select a particular register with respective read or write operations.

2. **Interrupt Control**
SI = “1”: When the SI flag is set, then, if the ENSIO bit is also set, a serial interrupt is requested. SI is set by hardware when one of 24 of the 25 possible SIO states is entered. The only state that does not cause SI to be set is state F8H, which indicates that no relevant state information is available. This interrupt is very useful in slowing down the fast master which is normally operating at higher clock frequency. The block diagram for the same is as shown below.

Implementation Logic:

The status register bits decide the logic of the interrupt control. The first 3 LSB’s of the register are set to 0. The logic is given by the equations:

\[ w1 = \neg(sta7) \land (sta6) \land (sta5) \land (sta4) \land (sta3) \]
\[ w2 = (sta7) \land \neg(sta6) \land \neg(sta5) \land (sta4) \land (sta3) \]
\[ w3 = (sta7) \land (sta6) \land \neg(sta5) \land (sta4) \land \neg(sta3) \]
\[ w4 = (sta7) \land (sta6) \land (sta5) \land \neg(sta4) \land \neg(sta3) \]
\[ w5 = (sta7) \land (sta6) \land (sta5) \land \neg(sta4) \land (sta3) \]
\[ w6 = (sta7) \land (sta6) \land (sta5) \land (sta4) \land (sta3) \]
3. Clock Divider and Selector

The clock divider block divides the processor clock frequency into a frequency that is closer to the I2C communication frequency. Different I2C frequencies are generated using the bits CR1, CR2 and CR3 of the control register.

The clock frequencies only take the HIGH and LOW times into consideration. The rise and fall time will cause the actual measured frequency to be lower than expected.

![Clock Divider and Selector Diagram]

Implementation Logic

![Implementation Logic Diagram]

The clock coming as an input from the interface is nearly 667MHz from which 9MHz i2c_clk is generated for the entire I2C module to function. The generator, generates 8 different frequencies for the clock selector to choose depending on the application. The 3 bits of the control register decide which frequency is selected at what time.

4. Start Stop Detector:

When a master on the I2C bus wants to communicate with a slave, it first has to take control of the bus. This is only possible when the bus is idle, i.e. both the SDA and SCL lines are high. The master has to create a START condition to signal other devices on the I2C bus that it will take control. To create a START condition, the clock line SCL remains high, while the master changes the SDA data line to the low condition. The end of a communication session is signaled by a STOP condition. The STOP condition is generated by changing the SDA data line to high while the clock line SCL is high.
Implementation Logic:

To be able to detect the START event, the following flip flops work in accordance. This generates a pulse lasting from the START event itself to the next rise of SCL (covering exactly one falling edge) as shown. The STOP detector is similar except for the SDA clock input. It generates a pulse from the STOP event to the next rise of SCL, again covering exactly one fall of SCL in the next transaction.

5. Bus Status Logic
This block checks if the bus is held busy by the I2C controller or any other external device and indicates that no data transfer can take place at that moment. The error detected in the bus when there is a wrong start or stop condition is also identified.

Implementation logic

Slave Receiver Logic

In a given application, the I2C block may operate as a master, a slave, or both. In the slave mode, the I2C hardware looks for its own slave address and the general call address. If one of these addresses is detected, an interrupt is requested. If the processor wishes to become the bus master, the hardware waits until the bus is free before the master mode is entered so that a possible slave operation is not interrupted. If bus arbitration is lost in the master mode, the I2C block switches to the slave mode immediately and can detect its own slave address in the same serial transfer.

- Master Transmitter mode
- Master Receiver mode
- Slave Receiver mode
- Slave Transmitter mode

In the slave receiver mode, data bytes are received from a master transmitter. To initialize the slave receiver mode, user writes into the Slave Address Register (I2ADR) and the I2C Control Register.
The enable bit must be set to 1 to enable the I2C function. AA bit must be set to 1 to acknowledge its own slave address or the general call address. The STA, STO and SI bits are set to 0. After address register and control register are initialized, the I2C interface waits until it is addressed by its own address or general address followed by the data direction bit. If the direction bit is 0 (W), it enters slave receiver mode. If the direction bit is 1 (R), it enters slave transmitter mode. After the address and direction bit have been received, the SI bit is set and a valid status code can be read from the Status Register.

Implementation Logic
SCL Enable and SDA Enable Blocks:

The SCL Enable Logic is a tri-state buffer with signals such as scl_enable, scl, scl_out and scl_in. A tri-state buffer is a device that allows you to control when an output signal makes it to the bus. When the tri-state buffer's control bit is active, the input of the device makes it to the output. This is when the "valve" is open. When it's not active, the output of the device is, which is high-impedance or, equivalently, nothing. This is when the "valve" is closed, and no electrical signal is allowed to pass to the output.

The condition when scl_enable is high, whatever maybe the logic at scl_out, it will propagate to scl, this does not happen if the enable signal is held low. In any condition, the logic at scl moves to the destination block via scl_in. The logic for SDA Enable remains the same as SCL Enable block provided that sda_in, sda, sda_out and sda_enable are replaced by scl_in, scl, scl_out and scl_enable respectively.

IV. FLOWCHART
V. RESULTS

The platform used is GPL Cver which is an open source Verilog simulator and the waveforms are observed in GTKWave. The following is the simulation output for each sub-module.

1. Register Access Control

![Simulation Output](image-url)
2. Clock Divider and selector

![Clock Divider and selector diagram]

3. Start stop detector

![Start stop detector diagram]

4. Bus status logic

![Bus status logic diagram]
5. Interrupt logic

6. SDA Enable logic

7. SCL Enable Logic
8. Slave Receiver

VI. CONCLUSION

I2C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. I2C bus is popular because it is simple to use, there can be more than one master, only upper bus speed is defined and only two wires with pull-up resistors are needed to connect almost unlimited number of I2C devices. I2C can use even slower microcontrollers with general-purpose I/O pins since they only need to generate correct Start and Stop conditions in addition to functions for reading and writing a byte.

I2C bus is used by many integrated circuits and is simple to implement. Any microcontroller can communicate with I2C devices even if it has no special I2C interface. I2C specifications are flexible - I2C bus can communicate with slow devices and can also use high speed modes to transfer large amounts of data. Because of many advantages, I2C bus will remain as one of the most popular serial interfaces to connect integrated circuits on the board.

VII. ACKNOWLEDGEMENTS

We express profound gratitude to respected principal Dr. K. Mallikharjuna Babu, BMS College of Engineering for providing a congenial environment to work in. Our sincere gratitude to Dr. D. Seshachalam, Head of the Department, Electronics and Communication Engineering for encouraging and providing this opportunity to carry out the project in the department.

We would like to thank our guide Dr. Kiran Bailey, Assistant Professor, Department of ECE who helped us in all the ways to carry out the project work. She stood beside and guided us in every step.

We would also like to convey our heartfelt thanks to Mr. Sunil Matange from IESA who has extended his helping hands throughout and given us the technical clarity that we needed.

We would also like to thank Mr. Harish V Mekali, Assistant Professor, Department of ECE, for the support he has extended in coordinating, formation of assessment rubrics, generating timely notifications for submission and driving all online activities related to project works.

REFERENCES

[2] Implementation of i2c master bus controller on FPGA, Bollam Eswari, VLSI design group, National Institute of Electronics and Information Technology, Kerala, India N. Ponnagah; K. Preethi; S. G. Sreejeesh, April 2013, IEEE.


[10] http://www.i2c.info/i2c•bus•specification/


[12] https://learn.sparkfun.com/tutorials/i2c
