M.Jyothi, L.Ravi Chandra, M.Sahithi, S.Daya Sagar Chowdary, K.Rajasekhar, K.Purnima / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622
www.ijera.com Vol. 2, Issue 2, Mar-Apr 2012, pp.875-883

Implementation of SPI Communication Protocol for Multipurpose Applications with I2C Power and Area Reduction

M.Jyothi[#1], L.Ravi Chandra[#2], M.Sahithi[#3],S.Daya Sagar Chowdary[#4],K.Rajasekhar[#5],K.Purnima[#6]

[#1] M.Jyothi, M.Tech student, Department of ECE, K L University, Vijayawada, INDIA

[#2, 3, 4, 5,6]Department of ECE, K L University, Vijayawada, INDIA

[*]L. Ravi Chandra Department of ECE, KL University, Vijayawada, INDIA

Abstract—The main objective of this paper is to implement the SPI communication used for the Fully Configurable Freely Scalable Digital Audio System. First we used Philipses 12C (Inter IC Communication), but it was too slow, and the number of the connected devices was limited. The speed of the communication between ICs is much faster than for the Full Duplex proper of the SPI communication. We use SPI protocol because it is frequently used when few I/O lines are available, but communication between two or more devices must be fast and easy to implement. Here we mainly focus on the advantages of SPI protocol when compared with I2C protocol which are mainly used for communication purpose.

***Key words*— I2C, SPI.**

I. INTRODUCTION

Today at the low end of the communication protocols we find two world wide protocols: I2C, SPI both protocols are well suited for communications between integrated circuits for low/medium data transfer speed with on-board peripherals. The two protocols coexist in modern digital electronic systems, and they probably will continue to complete in the future, as they both I2C and SPI are actually quite complimentary for this kind of communication. SPI plays virtual role in way of communications.

II.INTER INTIGRATED CIRCUIT (I2C)

I²C (Inter-Integrated Circuit; generically referred to as "two-wire interface") is a multi-master serial single-ended computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, or cellphone or other electronics. I2C is a two-wire, bi-directional serial bus that provides a simple and efficient method of data exchange between devices. It is most suitable for applications requiring occasional communication over a short distance between many devices. The I2C standard is a true multi-master bus including collision detection and arbitration that prevents data corruption if two or more masters attempt to control the bus simultaneously.

A. Design

I²C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock (SCL), pulled up with resistors. Typical voltages used are +5 V or +3.3 V although systems with other voltages are permitted. The I²C reference design has a 7-bit address space with 16 reserved addresses, so a maximum of 112 nodes can communicate on the same bus. Common I²C bus speeds are the 100 Kbit/s standard mode and the 10 Kbit/s low-speed mode, but arbitrarily low clock frequencies are also allowed. Recent revisions of I²C can host more nodes and run at faster speeds (400 Kbit/s Fast mode, 1 Mbit/s Fast mode plus or Fm+, and 3.4 Mbit/s High Speed mode). These speeds are more widely used on embedded systems than on PCs. There are also other features, such as 16-bit addressing.

Note that the bit rates quoted are for the transactions between master and slave without clock stretching or other hardware overhead. Protocol overheads include a slave address and perhaps a register address within the slave device as well as per-byte ACK/NACK bits. So the actual transfer rate of user data is lower than those peak bit rates alone would imply.

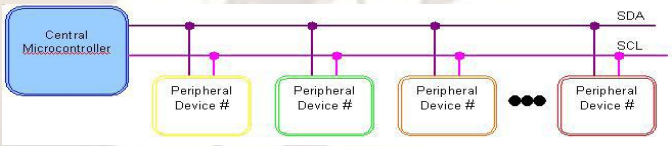

Fig: 1. I2C has two lines in total

The reference design, as mentioned above, is a bus with a clock (SCL) and data (SDA) lines with 7-bit addressing. The bus has two roles for nodes: master and slave:

- Master node — node that issues the clock and addresses slaves
- Slave node — node that receives the clock line and address.

The bus is a multi-master bus which means any number of master nodes can be present. Additionally, master and slave

roles may be changed between messages (after a STOP is sent). There are four potential modes of operation for a given bus device, although most devices only use a single role and its two modes:

- master transmit — master node is sending data to a slave
- master receive — master node is receiving data from a slave
- slave transmit — slave node is sending data to the master
- slave receive — slave node is receiving data from the master

The master is initially in master transmit mode by sending a start bit followed by the 7-bit address of the slave it wishes to communicate with, which is finally followed by a single bit representing whether it wishes to write(0) to or read(1) from the slave. If the slave exists on the bus then it will respond with an ACK bit (active low for acknowledged) for that address. The master then continues in either transmit or receive mode (according to the read/write bit it sent), and the slave continues in its complementary mode (receive or transmit, respectively). The address and the data bytes are sent most significant bit first. The start bit is indicated by a high-to-low transition of SDA with SCL high; the stop bit is indicated by a low-to-high transition of SDA with SCL high. If the master wishes to write to the slave then it repeatedly sends a byte with the slave sending an ACK bit. (In this situation, the master is in master transmit mode and the slave is in slave receive mode.) If the master wishes to read from the slave then it repeatedly receives a byte from the slave, the master sending an ACK bit after every byte but the last one. (In this situation, the master is in master receive mode and the slave is in slave transmit mode.) The master then ends transmission with a stop bit, or it may send another START bit if it wishes to retain control of the bus for another transfer (a "combined message").

B. Message protocols

I²C defines three basic types of messages, each of which begins with a START and ends with a STOP:

- Single message where a master writes data to a slave;
- Single message where a master reads data from a slave;
- Combined messages, where a master issues at least two reads and/or writes to one or more slaves.

In a combined message, each read or write begins with a START and the slave address. After the first START, these are also called repeated START bits; repeated START bits are not preceded by STOP bits, which is how slaves know the next transfer is part of the same message. Any given slave will only respond to particular messages, as defined by its product documentation. Pure I²C systems support arbitrary message structures. SMBus is restricted to nine of those structures, such as read word N and write word N, involving a single slave. PMBus extends SMBus with a Group protocol, allowing multiple such SMBus transactions to be sent in one combined message. The terminating STOP indicates when those grouped actions should take effect. For example, one PMBus operation might reconfigure three power supplies (using three different I2C slave addresses), and their new configurations would take effect at the same time: when they receive that STOP. With only a few exceptions, neither I²C nor SMBus define message semantics, such as the meaning of data bytes in messages. Message semantics are otherwise product-specific. Those exceptions include messages addressed to the I²C general call address (0x00) or to the SMBus Alert Response Address; and messages involved in the SMBus Address Resolution Protocol (ARP) for dynamic address allocation and management. In practice, most slaves adopt request/response control models, where one or more bytes following a write command are treated as a command or address. Those bytes determine how subsequent written bytes are treated and/or how the slave responds on subsequent reads. Most SMBus operations involve single byte commands.

C. I2C Protocol

The I2C bus physically consists of 2 active wires and a ground connection. The active wires, called SDA and SCL, are both bidirectional. SDA is the **S**erial Data line, and SCL is the **S**erial Clock line. Every device hooked up to the bus has its own unique address, no matter whether it is an MCU, LCD driver, memory, or ASIC. Each of these chips can act as a receiver and/or transmitter, depending on the functionality. Obviously, an LCD driver is only a receiver, while a memory or I/O chip can be both transmitter and receiver. The I2C bus is a multi-master bus. This means that more than one IC capable of initiating a data transfer can be connected to it. The I2C protocol specification states that the IC that initiates a data transfer on the bus is considered the Bus Master, which generally is a microcontroller. Consequently, at that time, all the other ICs are regarded to be Bus Slaves. First, the MCU will issue a START condition. This acts as an 'Attention' signal to all of the connected devices. All ICs on the bus will listen to the bus for incoming data. Then the MCU sends the ADDRESS of the device it wants to access, along with an indication whether the access is a Read or Write operation (Write in our example). Having received the address, all IC's will compare it with their own address. If it doesn't match, they simply wait until the bus is released by the stop condition (see below). If the address matches, however, the chip will produce a response called the ACKNOWLEDGEMENT signal. Once the MCU receives the acknowledgement, it can start transmitting or receiving DATA. In our case, the MCU will transmit data. When all is done, the MCU will issue the STOP condition. This is a signal that the bus has been released

and that the connected ICs may expect another transmission to start any moment.

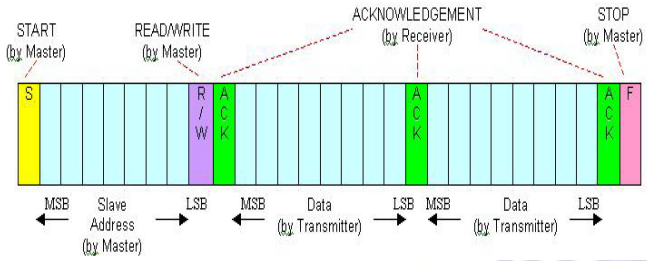

Fig: 2. I2C Communication

D. I2C Configuration

i. The Start and Stop Configuration

Start	SDA SCL	The chip issuing the Start condition first pulls the SDA (data) line low, and next pulls the SCL (clock) line low.
Stop	SDA SCL	The Bus Master first releases the SCL and then the SDA line.

Fig: 3. Start and stop with SDA and SCL

- A single message can contain multiple Start conditions. The use of this so-called "repeated start" is common in I2C.
- A Stop condition ALWAYS denotes the END of a transmission. Even if it is issued in the middle of a transaction or in the middle of a byte. It is "good behaviour" for a chip that, in this case, it disregards the information sent and resumes the "listening state", waiting for a new start condition.

ii. Transmitting a byte to a slave

Once the START condition has been sent, a byte can be transmitted by the MASTER to the SLAVE. This first byte after a start condition will identify the slave on the bus (address) and will select the mode of operation. The meaning of all following bytes depends on the slave.

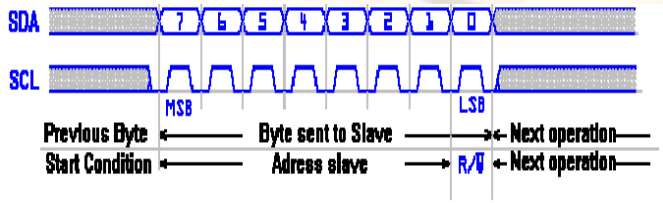

Fig: 4.SDA and SCL timing diagram

iii. Receiving a byte from a slave

Once the slave has been addressed and the slave has acknowledged this, a byte can be received from the slave if the R/W bit in the address was set to READ (set to '1'). The protocol syntax is the same as in transmitting a byte to a slave, except that now the master is not allowed to touch the SDA line. Prior to sending the 8 clock pulses needed to clock in a byte on the SCL line, the master releases the SDA line. The slave will now take control of this line. The line will then go high if it wants to transmit a '1' or, if the slave wants to send a '0', remain low.

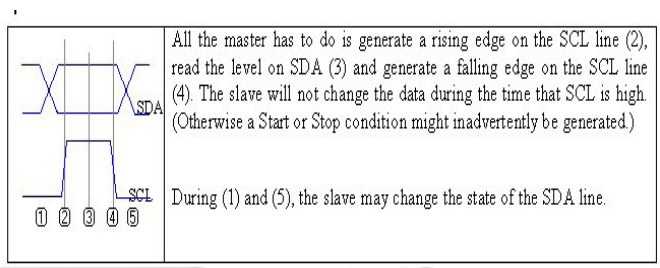

Fig: 5. Rising and falling edge of the SDA and SCL

In total, this sequence has to be performed 8 times to complete the data byte. Bytes are always transmitted MSB first.

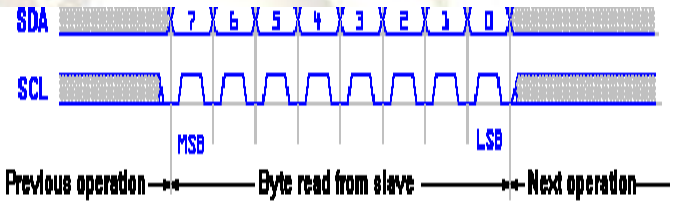

Fig: 6.Read operation of SDA and SCL

The meaning of all bytes being read depends on the slave. There is no such thing as a "universal status register". You need to consult the data sheet of the slave being addressed to know the meaning of each bit in any byte transmitted.

iv. Getting Acknowledgement from a Slave

When an address or data byte has been transmitted onto the bus, then this must be ACKNOWLEDGED by the slave(s). In case of an address: If the address matches its own, then only that slave will respond to the address with an ACK. In case of a byte transmitted to an already addressed slave, then that slave will respond with an ACK as well. The slave that is going to give an ACK pulls the SDA line low immediately after reception of the 8th bit transmitted, or, in case of an address byte, immediately after evaluation of its address. In practical applications this will not be noticeable. This means that as soon as the master pulls SCL low to complete the transmission of the bit (1), SDA will be pulled low by the slave (2). The master now issues a clock pulse on the SCL line (3). The slave will release the SDA line upon completion of this clock pulse (4).

Fig: 7. Clock polarities with SDA and SCL

II.SPI Communication

The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link standard named by Motorola that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines. Sometimes SPI is called a "four-wire" serial bus, contrasting with three-, two-, and one-wire serial buses. SPI is a general-purpose synchronous serial interface. During an SPI transfer, transmit and receive data is simultaneously shifted out and in serially. A serial clock line synchronizes the shifting and sampling of the information on two serial data lines. Motorola created the SPI port in the mid 1980's to use in their microcontroller product families. The SPI is mainly used to allow a microcontroller to communicate with peripheral devices such as E2PROMs. SPI devices communicate using a master-slave relationship. Due to its lack of built-in device addressing, SPI requires more effort and more hardware resources than I2C when more than one slave is involved. But SPI tends to be simpler and more efficient than I2C in point-to-point (single master, single slave) applications for the very same reason; the lack of device addressing means less overhead.

SPI is a serial bus standard established by Motorola and supported in silicon products from various manufacturers. SPI interfaces are available on popular communication processors and microcontrollers. It is a synchronous serial data link that operates in full duplex (signals carrying data go in both directions simultaneously). Devices communicate using a master/slave relationship, in which the master initiates the data frame. When the master generates a clock and selects a slave device, data may be transferred in either or both directions simultaneously. In fact, as far as SPI is concerned, data are always transferred in both directions. It is up to the master and slave devices to know whether a received byte is meaningful or not. So a device must discard the received byte in a "transmit only" frame or generate a dummy byte for a "receive only" frame. SPI specifies four signals: clock (SCK1); master data output, slave data input (SI1); master data input, slave data output (SO1); and chip select (CS). Figure 1 shows these signals in a single-slave configuration. SCK1 is generated by the master and input to all slaves. SI1 carries data from master to slave. SO1 carries data from slave back to master. A slave device is selected when the master asserts its CS signal.

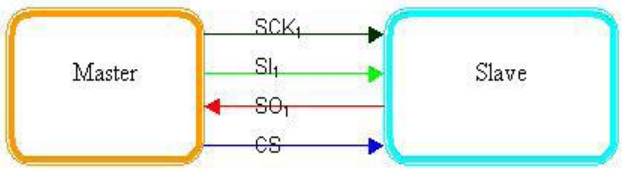

Fig: 8. Single master, single slave SPI implementation

Four logic signals are necessary to connect 2 or more devices with SPI:
- SCLK- Serial Clock (output from master)
- MOSI / SIMO - Master Out Slave In (output from master).
- MISO / SOMI - Master In Slave Out (output from slave)
- SS - Slave Select (active low, output from master). The SPI bus can operate with a single master device and with one or more slave devices. If a single slave device is used, the SS pin may be fixed to logic low if the slave permits it. Some slaves require the falling edge (high to low transition) of the chip select to initiate an action such as the Maxim MAX1242 ADC, which starts conversion on said transition. With multiple slave devices, an independent SS signal is required from the master for each slave device. Most slave devices have tri-state outputs so their MISO signal becomes high impedance ("disconnected") when the device is not selected. Devices without tri-state outputs can't share SPI bus segments with other devices; only one such slave could talk to the master, and only its chip select could be activated.

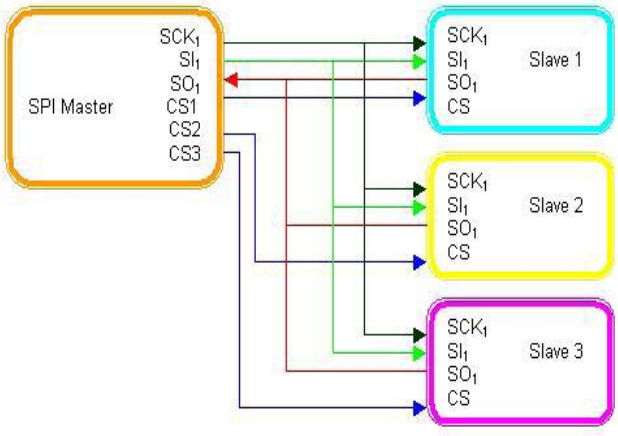

Fig. 9. Single master, multiple slave SPI implementations

Form the above diagram, we can justify that, data can be passed from one master to multiple slaves depends on activation of slave selection signal. Full duplex communication is in existence till now. The master generates slave select signals using general-purpose discrete input/output pins or other logic. This consists of old-fashioned

bit banging and can be pretty sensitive. You have to time it relative to the other signals and ensure, for example, that you don't toggle a select line in the middle of a frame. While SPI doesn't describe a specific way to implement multi-master systems, some SPI devices support additional signals that make such implementations possible. However, it's complicated and usually unnecessary, so it's not often done.

A pair of parameters called clock polarity (CPOL) and clock phase (CPHA) determines the edges of the clock signal on which the data are driven and sampled. Each of the two parameters has two possible states, which allows for four possible combinations, all of which are incompatible with one another. So a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used that are fixed in different configurations, the master will have to reconfigure itself each time it needs to communicate with a different slave. SPI does not have an acknowledgement mechanism to confirm receipt of data. In fact, without a communication protocol, the SPI master has no knowledge of whether a slave even exists. SPI also offers no flow control. If you need hardware flow control, you might need to do something outside of SPI. Slaves can be thought of as input/output devices of the master. SPI does not specify a particular higher-level protocol for master-slave dialog. In some applications, a higher-level protocol is not needed and only raw data are exchanged. An example of this is an interface to a simple codec. In other applications, a higher-level protocol, such as a command-response protocol, may be necessary. Note that the master must initiate the frames for both its command and the slave's response.

If there is only one slave device then the SS pin on the slave device can be fixed to logic low state. If there is 2 or more slave devices in the system, then an independent SS signal is required from the master device for each slave device. When the master device wants to start a communication it has to set the clocks, that is less than or equal to the slave device's maximum frequency (most commonly from 1 to a few MHz). SPI communication is a full duplex communication, the master device sends a byte to the desired slave device in the meantime it receives a byte from the slave device. Transmissions may involve any number of clock cycles. When there are no more data to be transmitted, the master device stops toggling its clock. Normally, it then deselects the slave device. Every slave device on the bus that hasn't been activated using its Slave Select line must disregard the input clock and MOSI signals, and may not drive MISO. The master device selects only one slave at a time.

A. Data and Control Lines of the SPI

The SPI requires two control lines (CS and SCK) and two data lines (SI and SO).

With CS (Chip-Select) the corresponding peripheral device is selected. This pin is mostly active-low. In the unselected state the SO lines are hi-Z and therefore inactive. The master decides with which peripheral device it wants to communicate. The clock line SCLK is brought to the device whether it is selected or not. The clock serves as synchronization of the data communication. The majority of SPI devices provide these four lines. Sometimes it happens that SDI and SDO are multiplexed, for example in the temperature sensor LM74 from National Semiconductor or that one of these lines is missing. A peripheral device which must or cannot be configured, requires no input line, only a data output. As soon as it gets selected it starts sending data. In some ADCs therefore the SDI line is missing (e.g. MCCP3001 from Microchip). There are also devices that have no data output. For example LCD controllers (e.g. COP472-3 from National Semiconductor), which can be configured, but cannot send data or status messages.

In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data. Free scale's SPI Block Guide names these two options as CPOL and CPHA respectively, and most vendors have adopted that convention.

The timing diagram is shown to the right. The timing is further described below and applies to both the master and the slave device.

-At CPOL=0 the base value of the clock is zero.

For CPHA=0, data are captured on the clock's rising edge (low to high transition) and data are propagated on a falling edge (high to low clock transition). For CPHA=1, data are captured on the clock's falling edge and data are propagated on a rising edge.

-At CPOL=1 the base value of the clock is one (inversion of CPOL=0)

For CPHA=0, data are captured on clock's falling edge and data are propagated on a rising edge. For CPHA=1, data are captured on clock's rising edge and data are propagated on a falling edge. That is, CPHA=0 means sample on the leading (first) clock edge, while CPHA=1 means sample on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling. Note that with CPHA=0, the data must be stable for a half cycle before the first clock cycle. For all CPOL and CPHA modes, the initial clock value must be stable before the chip select line goes active. Also, note that "data is read" in this document more typically means "data may be read". The MOSI and MISO signals are usually stable (at their reception points) for the half cycle until the next clock transition. SPI master and slave devices may well sample data at different points in that half cycle. This adds more flexibility to the communication channel between the master and slave.

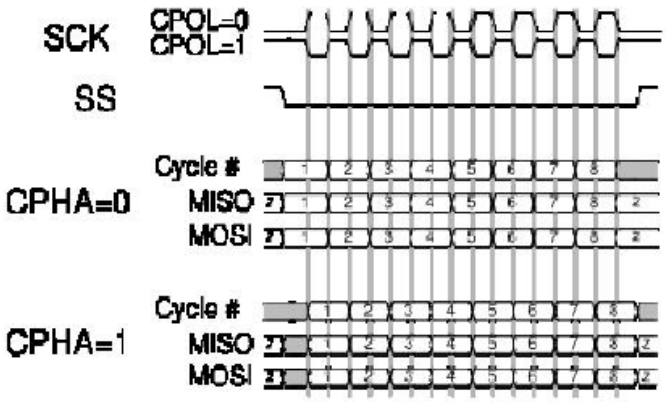

Fig.10. Clock dependencies from CPHA and CPOL

CPOL	CPHA	Active edge
0	0	Rising
0	1	Falling
1	0	Falling
1	1	Rising

Table: 1.CPOL and CPHA setup table

Some devices even have minor variances from the CPOL/CPHA modes described above. Sending data from slave to master may use the opposite clock edge as master to slave. Devices often require extra clock idle time before the first clock or after the last one, or between a command and its response. Some devices have two clocks, one to "capture" or "display" data, and another to clock it into the device. Many of these "capture clocks" run from the chip select line.

Some devices require an additional flow control signal from slave to master, indicating when data are ready. This leads to a "five wire" protocol instead of the usual four. Such a "ready" or "enable" signal is often active-low, and needs to be enabled at key points such as after commands or between words. Without such a signal, data transfer rates may need to be slowed down significantly, or protocols may need to have "dummy bytes" inserted, to accommodate the worst case for the slave response time. Examples include initiating an ADC conversion, addressing the right page of flash memory, and processing enough of a command that device firmware can load the first word of the response. (Many SPI masters don't support that signal directly, and instead rely on fixed delays.)

Many SPI chips only support messages that are multiples of 8 bits. Such chips cannot interoperate with the JTAG or SGPIO protocols, or any other protocol that requires messages that are not multiples of 8 bits.

B.SPI Configuration

Because there is no official specification, what exactly SPI is and what not, it is necessary to consult the data sheets of the Components one wants to use. Important are the permitted clock frequencies and the type of valid transitions. There are no general rules for transitions where data should be latched. Although not specified by Motorola, in practice four modes are used. These four modes are the combinations of CPOL and CPHA. In table 2, the four modes are listed.

SPI-mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Table2: SPI Modes

If the phase of the clock is zero, i.e. CPHA = 0, data is latched at the rising edge of the clock with CPOL = 0, and at the falling edge of the clock with CPOL = 1. If CPHA = 1, the polarities are reversed. CPOL = 0 means falling edge, CPOL = 1 rising edge. The micro controllers from Motorola allow the polarity and the phase of the clock to be adjusted. A positive polarity results in latching data at the rising edge of the clock. However data is put on the data line already at the falling edge in order to stabilize. Most peripherals which can only be slaves, work with this configuration. If it should become necessary to use the other polarity, transitions are reversed.

III.SPI vs. I2C

Although both SPI and I2C provide good support for communication with slow peripheral devices that are accessed intermittently, each of the way of communication have its own advantages towards each other. SPI is better suited than I2C for applications that are naturally thought of as data streams (as opposed to reading and writing addressed locations in a slave device). An example of a "stream" application is data communication between microprocessors or digital signal processors. Another is data transfer from analog-to-digital converters. SPI can also achieve significantly higher data rates than I2C which is limited to 400KHz in most cases. SPI-compatible interfaces often range into the tens of megahertz. SPI really gains efficiency in applications that take advantage of its duplex capability, such as the communication between a

"codec" (coder-decoder) and a digital signal processor, which consists of simultaneously sending samples in and out.

Due to SPI lack of built-in device addressing, it requires more effort and more hardware resources than I2C when more than one slave is involved. The disadvantage here lies that it is a three-wire interface and if you are having more than 1 device, then you have to provide each device with separate Chip Select pins (CS). But SPI tends to be simpler and more efficient than I2C in point-to-point (single master, single slave) applications for the very same reason; the lack of device addressing means less overhead. On the other hand, I2C requires only two wires to implement and has a unique address so that a master/slave relationship can be maintained compare to SPI which needed three wires to implement the addressing mode. I2C also offers better support for communication with on-board devices that are accessed on an occasional basis. I2C's competitive advantage over other low-speed short-distance communication schemes is that its cost and complexity don't scale up with the number of devices on the bus because of the generic nature of the bus interface.

Besides, the complexity of the supporting I2C software components can be significantly higher than that of several competing schemes such as SPI in a very simple configuration. With its built-in addressing scheme and straightforward means to transfer strings of bytes, I2C is an elegant, minimalist solution for modest, "inside the box" communication needs. I2C is also a true multi-master bus because it has collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer. Furthermore, I2C also preserve data integrity by filtering rejects spikes on the bus data line.

IV.COMPARISON BETWEEN I2C AND SPI

We are compared the SPI and I2C protocols. Here the SPI protocol is much faster than I2C. The area and power was reduced compare to I2C protocol. XPower and Datasheet may have some Quiescent Current differences. This is due to the fact that the quiescent numbers in XPower are based on measurements of real designs with active functional elements reflecting real world design scenarios.

Power summary:	**I(mA)**	**P(mW)**
Total estimated power consumption:		**48**
Vccint 2.50V:	**17**	**42**
Vcco33 3.30V:	**2**	**7**
Clocks:	**12**	**30**
Inputs:	**2**	**5**
Logic:	**0**	**0**
Outputs:		
Vcco33	**0**	**0**
Signals:	**0**	**0**
Quiescent Vccint 2.50V:	**3**	**7**
Quiescent Vcco33 3.30V:	**2**	**7**

Thermal summary:	
Estimated junction temperature:	**26C**
Ambient temp:	**25C**
Case temp:	**26C**
Theta J-A range:	**27 - 30C/W**

Decoupling Network Summary:	**Cap Range (uF)**	**#**
Capacitor Recommendations:		
Total for Vccint		12
	470.0-1000.0	1
	0.470- 2.200	1
	0.0470-0.2200	2
	0.0100-0.0470	3
	0.0010-0.0047	5
		Invalid Program Mode
Total for Vcco33		8
	470.0-1000.0	1
	0.0470-0.2200	1
	0.0100-0.0470	2
	0.0010-0.0047	4

I2C

XPower and Datasheet may have some Quiescent Current differences. This is due to the fact that the quiescent numbers in XPower are based on measurements of real designs with active functional elements reflecting real world design scenarios.

Power summary:	**I(mA)**	**P(mW)**
Total estimated power consumption:		**68**
Vccint 2.50V:	**15**	**38**
Vcco33 3.30V:	**2**	**7**
Inputs:	**0**	**1**
Logic:	**11**	**27**
Outputs:		

Vcco33	**0**	**0**
Signals:	**1**	**4**
Quiescent Vccint 2.50V:	**3**	**7**
Quiescent Vcco33 3.30V:	**2**	**7**

Thermal summary:		
Estimated junction temperature:	**26C**	
Ambient temp:	**25C**	
Case temp:	**26C**	
Theta J-A range:	**27 - 30C/W**	
Decoupling Network Summary:	**Cap Range (uF)**	**#**
Capacitor Recommendations:		
Total for Vccint		12
	470.0-1000.0	1
	0.470- 2.200	1
	0.0470-0.2200	2
	0.0100-0.0470	3
	0.0010-0.0047	5
Total for Vcco33		8
	470.0-1000.0	1
	0.0470-0.2200	1
	0.0100-0.0470	2
	0.0010-0.0047	4

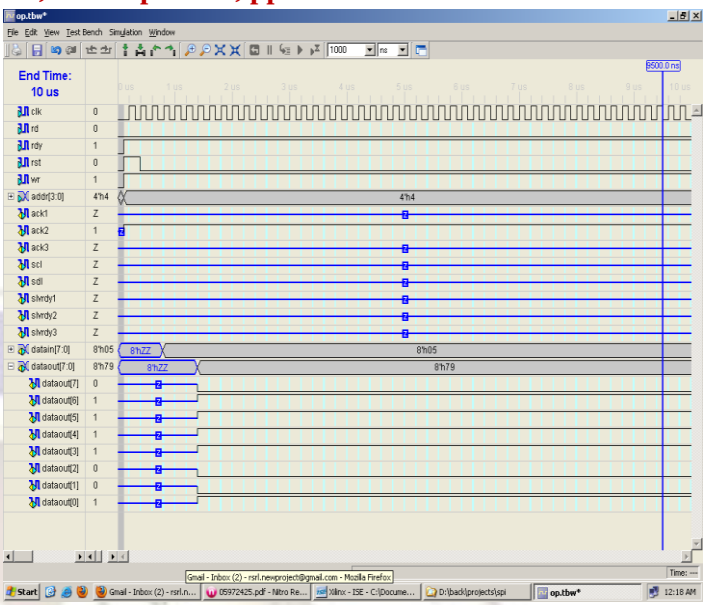

Fig: 11. Simulation results of I2C

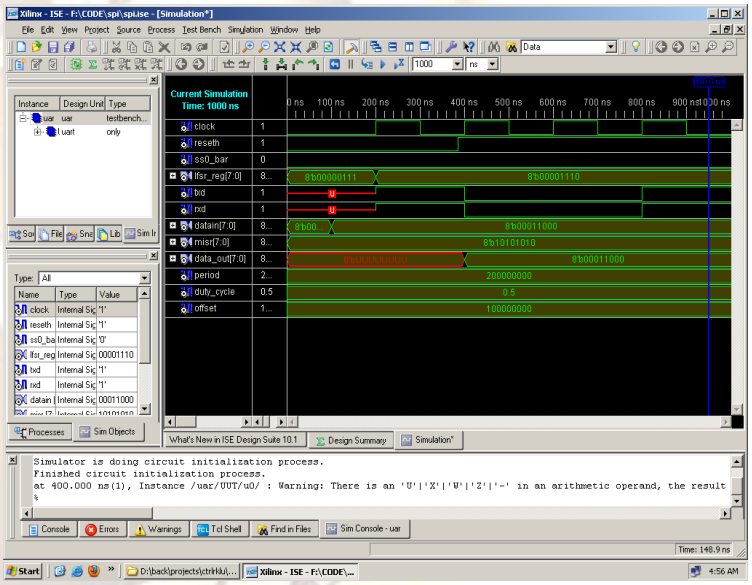

Fig.12. Simulation Results of SPI

IV. SIMULATION RESULTS

The SPI and I2C communication described above is designed using VHDL and simulated. The simulation results here shown are about the SPI protocol and I2C protocol which is designed with high speed, power and area reduction. The SPI is much faster than the I2C protocol. The SPI is a full duplex protocol and it has a high speed protocol compare to I2C.

Here Fig 11 and 12 shows the results of SPI and I2C communication protocol. Here compare the two protocols the SPI is much faster compare to I2C. The two are having their own importance in this design.

V.CONCLUSION

FINALLY IN THIS PAPER WE COMPARE SPI AND I2C PROTOCOL .WE DESIGN HIGH SPEED, POWER AND AREA REDUCTION. THE SPEED OF THE COMMUNICATION BETWEEN ICS IS MUCH FASTER THAN FOR THE FULL DUPLEX PROPER OF THE SPI COMMUNICATION. WE USE SPI PROTOCOL BECAUSE IT

IS FREQUENTLY USED WHEN FEW I/O LINES ARE AVAILABLE, BUT COMMUNICATION BETWEEN TWO OR MORE DEVICES MUST BE FAST AND EASY TO IMPLEMENT. HERE WE MAINLY FOCUS ON THE ADVANTAGES OF SPI PROTOCOL WHEN COMPARED WITH I2C PROTOCOL WHICH ARE MAINLY USED FOR COMMUNICATION PURPOSE. FURTHERMORE THESE PROTOCOLS CAN BE APPLICABLE FOR DIFFERENT APPLICATIONS LIKE SOC, CPU AND DSP PROCESSORS.

VI. REFERENCES

[1] Motorola, "MC68HC II manual".
[2] Texas Instruments, "MSP430xlxx family users guide".
[3] Texas Instruments website, www.ti.com.
[4] Peter Kaszas, Akos Szekacs, Tibor Szakall.
[5] "Audio system controlling protocol (ASCP) with AES3," unpublished.
[6] Philips's website, www.philips.com.
[7] D.J. Wheeler, R. Needham, TEA, a Tiny Encryption Algorithm, in the proceedings of FSE 1994, Lecture Notes in Computer Science, Vol 1008, pp 363-366, Leuven, Belgium, December 1994, singer- verlag.
[8] M. Matsui, Linear Cryptanalysis Method for DES Cipher, in the proceedings of Eurocrypt 1993, Lecture Notes in Computer Science, vol 765, pp 386-397, Lofthus, Norway, May 1993, Springer-Verlag.
[9] J. Daemen, V. Rijmen, the Design of Rijndael, Springer-Verlag, 2001.
[10] FIPS 197, \Advanced Encryption Standard," Federal Information Processing Stan- dard, NIST, U.S. ept. of Commerce, November 26, 2001.