ISSN:0975-9646

Implementation of Low Complex and High Secured SPI Communication System for Multipurpose Applications

M.Jyothi, L.Ravi Chandra, M.Sahithi,A.Jhansi Rani,J.Poornima,N.Naga Sudha
Department of ECE, K L University
Vijayawada, INDIA

***Abstract*—The main objective of this project is to implement a full duplex SPI (serial peripheral interfacing system) with low complexity and high security algorithms. High security is achieved by encoding the data with S.E.A (scalable encryption algorithms) along with that, error checking capability is also provided by adding even parity.**

***Key words*— SPI, SEA, error checking.**

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. But no security is provided and no error checking is provided for that communication protocol. We need such protocol for efficient and secured 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.

II.SPI COMMUNICATION

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.

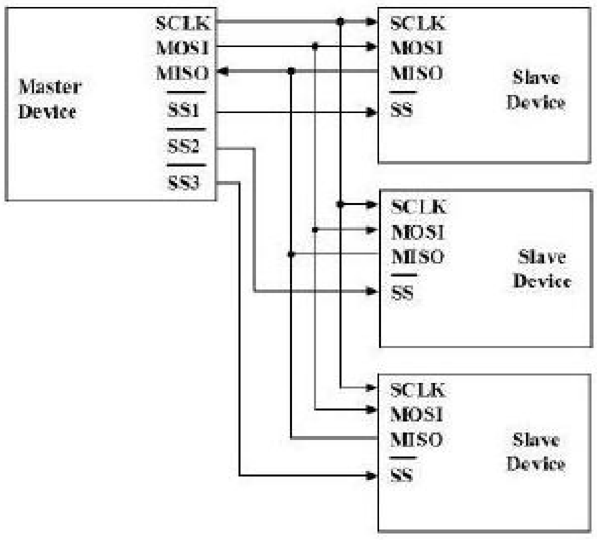


Fig. 1. The SPI Bus system with 1 master device and with 3 slave devices

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**.**

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.

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.

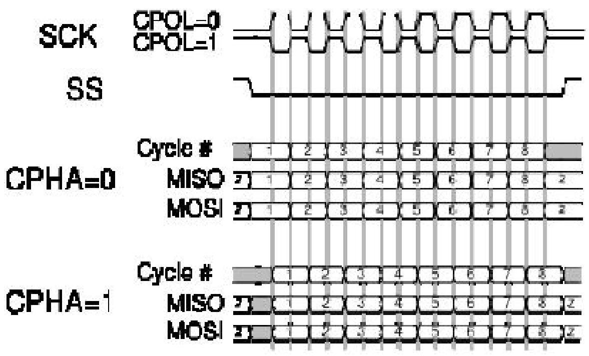
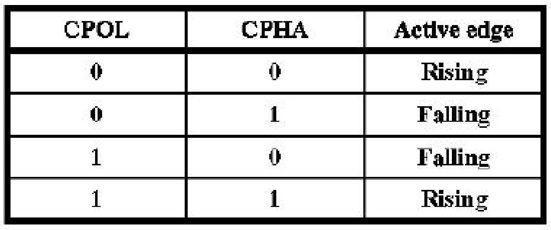


Fig.2. Clock dependencies from CPHA and CPOL



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

Fig. 3. 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.

III.PROPOSED MODEL

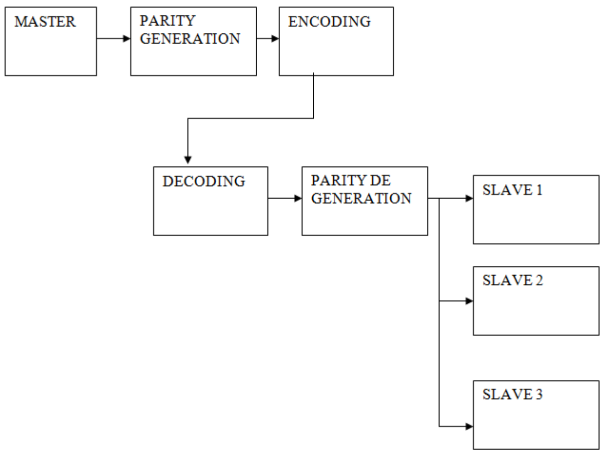


Fig.4.Low Complex and High Secure SPI Communication

In the above block diagram, Full-duplex communication is established between single master and multi slave devices with more security. For security purpose, S.E.A (Scalable Encryption algorithm) is utilized. Error checking is also main criteria while sending data from one point to another. Here, even parity generations are used for error checking purpose.

A. Error detection

In information theory and coding theory with applications in computer science and telecommunication, error detection and correction or error control are techniques that enable reliable delivery of digital data over unreliable communication channels. Many communication channels are subject to channel noise, and thus errors may be introduced during transmission from the source to a receiver. Error detection techniques allow detecting such errors, while error correction enables reconstruction of the original data. The general definitions of the terms are as follows:
Error detection is the detection of errors caused by noise or other impairments during transmission from the

transmitter to the receiver. Error correction is the detection of errors and reconstruction of the original, error-free data.

Error correction may generally be realized in two different ways:

Automatic Repeat Request (ARQ) (sometimes also referred to as backward error correction): This is an error control technique whereby an error detection scheme is combined with requests for retransmission of erroneous data. Every block of data received is checked using the error detection code used, and if the check fails, retransmission of the data is requested this may be done repeatedly, until the data can be verified.

Forward Error Correction (FEC): The sender encodes the data using an error-correcting code (ECC) prior to transmission. The additional information (redundancy) added by the code is used by the receiver to recover the original data. In general, the reconstructed data is what is deemed the "most likely" original data.

B. Error Detection Schemes

Error detection is most commonly realized using a suitable hash function (or checksum algorithm). A hash function adds a fixed-length tag to a message, which enables receivers to verify the delivered message by recomputing the tag and comparing it with the one provided. There exists a vast variety of different hash function designs. However, some are of particularly widespread use because of either their simplicity or their suitability for detecting certain kinds of errors (e.g., the cyclic redundancy check's performance in detecting burst errors.

Random-error-correcting codes based on minimum distance coding can provide a suitable alternative to hash functions when a strict guarantee on the minimum number of errors to be detected is desired. Repetition codes, described below, are special cases of error-correcting codes: although rather inefficient, they find applications for both error correction and detection due to their simplicity.

C. Repetition codes

A repetition code is a coding scheme that repeats the bits across a channel to achieve error-free communication. Given a stream of data to be transmitted, the data is divided into blocks of bits. Each block is transmitted some predetermined number of times. For example, to send the bit pattern "1011", the four-bit block can be repeated three times, thus producing "1011 1011 1011". However, if this twelve-bit pattern was received as "1010 1011 1011" – where the first block is unlike the other two – it can be determined that an error has occurred.

Repetition codes are not very efficient, and can be susceptible to problems if the error occurs in exactly the same place for each group (e.g., "1010 1010 1010" in the previous example would be detected as correct). The advantage of repetition codes is that they are extremely simple, and are in fact used in some transmissions of numbers stations**.**

D. Parity Generator

A parity bit is a bit that is added to ensure that the number of bits with the value one in a set of bits is even or odd. Parity bits are used as the simplest form of error detecting code.

There are two variants of parity bits: even parity bit and odd parity bit. When using even parity, the parity bit is set to 0 if the number of ones in a given set of bits (not including the parity bit) is even, making the entire set of bits (including the parity bit) even. When using odd parity, the parity bit is set to 0 if the number of ones in a given set of bits (not including the parity bit) is odd, keeping the entire set of bits (including the parity bit) odd. Even parity is a special case of a cyclic redundancy check (CRC), where the 1-bit CRC is generated by the polynomial x+1. If the parity bit is present but not used, it may be referred to as mark parity (when the parity bit is always 1) or space parity (the bit is always 0).

7 bits of data (number of 1s)	8 bits including parity	
	Even	odd
0000000 (0)	00000000	10000000
1010001 (3)	11010001	01010001
1101001 (4)	01101001	11101001
1111111 (7)	11111111	01111111

Parity bits are extra signals which are added to a data word to enable error checking. There are two types of Parity even and odd. An even parity generator will produce a logic 1 at its output if the data word contains an odd number of ones. If the data word contains an even number of one's then the output of the parity generator will be low. By concatenating the Parity bit to the data word, a word will be formed which always has an even number of one's i.e. has even parity.

E. Encryption

Encryption is the conversion of data into a form, called a cipher text that cannot be easily understood by unauthorized people. Decryption is the process of converting encrypted data back into its original form, so it can be understood. The use of encryption/decryption is as old as the art of communication. In wartime, a cipher, often incorrectly called a code, can be employed to keep the enemy from obtaining the contents of transmissions. (Technically, a code is a means of representing a signal without the intent of keeping it secret; examples are Morse code and ASCII.) Simple ciphers include the substitution of letters for numbers, the rotation of letters in the alphabet, and the "scrambling" of voice signals by inverting the sideband frequencies. More complex ciphers work according to sophisticated computer algorithms that rearrange the data bits in digital signals. In order to easily recover the contents of an encrypted signal, the correct decryption key is required. The key is an algorithm that undoes the work of the encryption algorithm. Alternatively, a computer can be used in an attempt to break the cipher. The more complex the encryption algorithm, the more difficult it becomes to eavesdrop on the communications without access to the key.

Encryption/decryption is especially important in wireless communications. This is because wireless circuits are easier to tap than their hard-wired counterparts. Nevertheless, encryption/decryption is a good idea when carrying out any kind of sensitive transaction, such as a credit card purchase online, or the discussion of a company secret between different departments in the organization. The stronger the cipher that is, the harder it is for unauthorized people to break it the better, in general. However, as the strength of encryption/decryption increases, so does the cost.

F. Decryption

Encryption is the initial message prepared by the sender is written as plaintext, which the sender converts into cipher text before the message is transmitted. The process of converting plaintext into cipher text is called encryption. The encryption process requires an encryption algorithm and a key. The process of recovering plaintext from cipher text is called decryption.

In classical cryptography, the key is exchanged secretly between sender and receiver over secured communication, or through a trusted intermediary. The accepted view among professional cryptographers it that the encryption algorithm should be published, whereas the key must be kept secret. The purpose of publishing the encryption algorithm is to place it before the academic cryptography community, which will discover its flaws. Better that the flaws in the encryption algorithm be first discovered in academia than when the message is secretly decoded by the attacker. Sample encryption calculation is the both the initial plaintext and the resulting cipher text may contain words or numbers or both, but is ultimately convertible into a sequence of numerals, which can be processed by computer and distributed through public communications, including the internet. For simplicity of discussion, we can speak of an initial plaintext expressed as a sequence of decimal numerals. For example, let the letters of the alphabet be represented as two-digit numbers from a=00 to z=25 (ignore blank-spaces for now). Then the plaintext for the quick brown fox becomes numeralized as 19070416200802100117142213051423, as follows:

```
 Thequickbrownfox
 t h e q u i c k b r o w n f o x
19 07 04 16 20 08 02 10 01 17 14 22 13 05 14 23
```

analogously, we may form a simple key consisting, say, of the consecutive letters of the alphabet: abcdefghijklmnopqrstuvwxyzabcd....

```
 Abcdefghijklmnopqrs
 a b c d e f g h i j k l m n o p q r s t u....
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17
18 19 20....
```

A simple encryption algorithm might consist of adding the plaintext to the encryption-key, using modulo-26 arithmetic. That is, if the sum of any two numbers obtained by ordinary addition is 26 or greater, then you subtract 26 from the ordinary sum to obtain the modulo-26 sum. Thus, 05+12=17 by both ordinary and modulo-26 arithmetic, but 15+12=27 by ordinary arithmetic but 15+12=01 by modulo-26 arithmetic. Hence, the cipher text for thequickbrownfox is 19080619241308170901240725180212, as follows:

```
    19 07 04 16 20 08 02 10 01 17 14 22 13 05 14 23
(+) 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
(modulo-26)
____________________________________________________
    19 08 06 19 24 13 08 17 09 01 24 07 25 18 02 12
```

The cipher text may then be decrypted by the receiver, using the decryption-key azyxwvutsrqponmlkjihgfedcbazyx... and modulo-26 arithmetic, as follows:

```
    19 08 06 19 24 13 08 17 09 01 24 07 25 18 02 12
(+) 00 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11
(modulo-26)
____________________________________________________
    19 07 04 16 20 08 02 10 01 17 14 22 13 05 14 23
```

It is entirely reverse process to the encryption. Inputs to the decryption block are received data along with key. Key should be same as in tx block. Then only cipher text will be converted in to original plane text.

G. Parity Degeneration

Error detection using parity compensation in binary coded decimal (BCD) and densely packed decimal (DPD) conversions, including a computer program product having a tangible storage medium readable by a processing circuit and storing instructions for execution by the processing circuit for performing a method. The method includes receiving formatted decimal data in a first format, the formatted decimal data consisting of a DPD format data or a BCD format data. One or more first parity bits are generated by converting the received data into a second format of the formatted decimal data, and by determining the parity of the data in the second format. One or more second parity bits are generated directly from the received data. An error flag is set to indicate an error in the data in the second format in response to the first parity bits not being equal to the second parity bits.

H. Parity check

This is the simplest scheme of error detection. In this scheme a single parity bit is introduced. This parity bit is appended at the end of the block of data in such a way that the block of data contains either even (even parity) or odd (odd parity) number of ones. for example in case of even parity consider a block of data to be sent is 10100100 this is a seven bit data a new parity bit will be appended at the end so that the number of one’s in the data become even 101001001. Now this data is sent across the transmission channel and if suppose error occurs during the transmission and the third bit in the block become zero (100001001) now this will be received by the receiver and will detect an error. However if two or more bits are inverted in such a way that the number of one’s remain even (e.g. 11001001) then the error will remain undetected. same is the case with odd parity in which a parity bit is introduced in such a way the resulting block of data contain odd numbers of ones. Typically, even parity is used for synchronous transmission and odd parity for asynchronous transmission. This method in not foolproof, noise impulses are often long and destroy more than one bit of data.

The parity bit is only an error detection code. The concept of parity bit has been later on developed and error detection and correction code has been developed using more than one parity bits. One such code is hamming error correcting code.

Hamming error-correcting code: this code was devised by Richard hamming at bell laboratories. Let's understand this codes with the help of Venn diagrams, let's consider 4 bit data. Figure below shows the Venn diagrams with filled in data bits, which are filled in the intersecting inner compartments. the next step is to fill in the parity bits for these four data bit the principle here is that we add the parity bits such that the total number of l's in each circle is even (even parity) please note that each of the circle have even number of l's.

After the data transfer, let us say, we encounter a situation where one of the data bit is changed from 1 to 0. Thus, an error has occurred. How will this error be detected and rectified?

The parity bit of the two circles are indicating error of one bit, since two circles are indicating errors, therefore, the error lies at the intersection of these two circle. So, we have not only recognized the error but also its source. Thus, in this case by changing the bit in error, from 0 to 1 we can easily rectify the error. Now let us discuss a scheme for error correction and detection of single bit errors in 8 bit words. The first question in this respect is: what should be the length of the code? The error detection is done by comparing the two input bit error detection and correction codes fed to the comparison logic bit by bit. Let us have a comparison logic which produce a 0 if the compared bits are same or else it produce a1. Therefore, if similar position bits are same then we get 0 at that bit position, but if they are different, that is this bit position may point to some error, then this particular bit position will be marked as 1. This way a match word called syndrome word is constructed. This syndrome word is i bit long, therefore, can represent 2i values or combinations. for example, a 4 bit syndrome word can represent 24=16 values which range from 0 to 15 as:

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111
1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111

the value 0000 or 0 represent no error while the other values i.e..2i-1 (for 4 bits 24-1=15 that is from 1 to15) represent an error condition.

The next step in this connection will be to arrange the 8 bit word and its 4 bit error correction code in such a way that a particular value of the syndrome word specifies an error in a unique bit (which may be data or error detection code'). The following arrangement of the (N+i) bits is suggested.

Bit positions	12	11	10	9	8	7	6	5	4	3	2	1
Data bits	8	7	6	5		4	3	2		1		
Correction bits					8				4		2	1

Data bits and bit positions.

The above arrangement is derived on the basis that: Syndrome Word zero implies no error. If syndrome word contains only one bit as 1 then it should be inferred that error has occurred only in the parity bits, therefore, no correction is needed in data. But how can we implement it? This can be implemented easily by assigning the check bits as 1st, 2nd, 4th, and 8th bit position. In case more than one bit in syndrome word are set as 1 then the numerical value of the syndrome word should determine the bit position which is in error. The arrangement shown in figure above has an added advantage that is each data bit position can be calculated as a function of correction bit positions. Please note, in case any one of the correction bit has changed during data transmission, that implies any one of the 1st or 2nd or 4th or 8th bit position data have altered, therefore, the syndrome bit will be 0001 if the data at first bit position has changed, 0010 if 2nd bit position has changed; or 0100 if data at 4th bit position has changed, or 1000 if data at 8th bit position has changed. Thus, the proposed bit arrangement scheme of figure above satisfies the second assumption for the bit arrangement scheme. The next assumption in this regard is the value of syndrome word should indicate the bit position which is in error, that is, if there is error in bit position 3 it should change correction bits of bit position 1 and 2 and so on. Let us discuss how this can achieve.

For example The SEC code for 8 bit word is of 4 bits

Check bit l =Even parity of(1,1,1,1,1)=1
Check bit 2 =Even parity of(1,0,1,0,1)=1
Check bit 3 =Even parity of(1,0,1,0)=0
Check bit 4 =Even parity of(1,0,1,0)=0

Therefore, the 12 bit word to be transmitted is

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Data Bits	8	7	6	5		4	3	2		1		
Check Bits					4				3		2	1
Data to be transmitted	0	1	0	1	0	1	0	1	0	1	1	1
Data Received	0	1	0	0	0	1	0	1	0	1	1	1

Error in the 5th data bit
Calculation of check bits of data received:

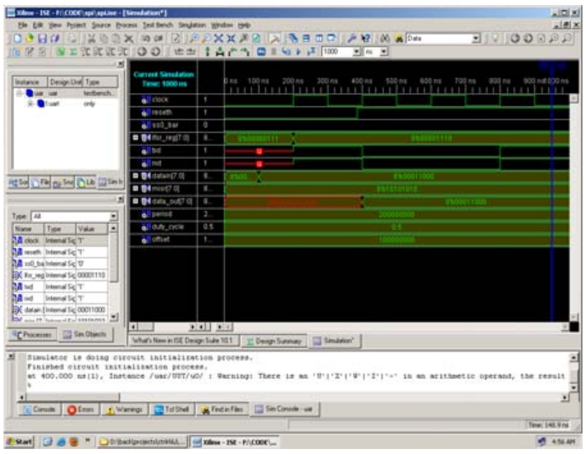
Check bit 1 = Even parity of(l,1,1,0,1,)=0
Check bit 2 = Even parity of(1,0,1,0,1,)=1
Check bit 3 = Even parity of(1,0,1,0)=0
Check bit 4 = Even parity of(0,0,1,0)=1

Syndrome word =compare the received check bits to calculated checks bits Of received data
= 0 0 1 1 Received check bits
1 0 1 0 calculated check bits
1 0 0 1

Please note for syndrome word calculation if two check bits are same then the respective bit in syndrome word will be 0 if the two check bits are same then the bit in syndrome word will be1. Thus, syndrome word = 1001, which implies that 9th bit position in the received 12 bit information is in error. The 9th bit position corresponds to 5th data bit. Change this bit to 1 if it is 0 or 0 if it is 1. Since in the received data it is 0, therefore, change it to1. Hence, the data was (excluding check bits) received as 01001011. The corrected data is 01011011 the corrected data is same as the transmitted data.

IV. SIMULATION RESULTS

The SPI communication described above is designed using VHDL and simulated. The simulation results here shown are about the slave modules of the protocol which is designed with S.E.A (scalable encryption algorithms) along with that, error checking capability is also provided by adding even parity.



```
Simulator is doing circuit initialization process.
Finished circuit initialization process.
at 400.000 ns(1), Instance /uar/UUT/u0/ : Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result
```

Fig.5. Simulation Results of USART

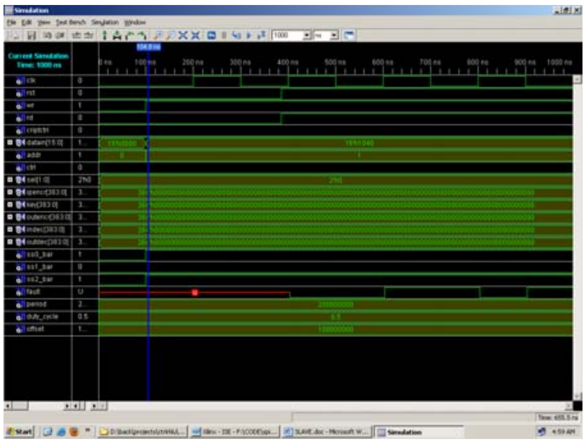


Fig.6. Simulation Results for LFSR

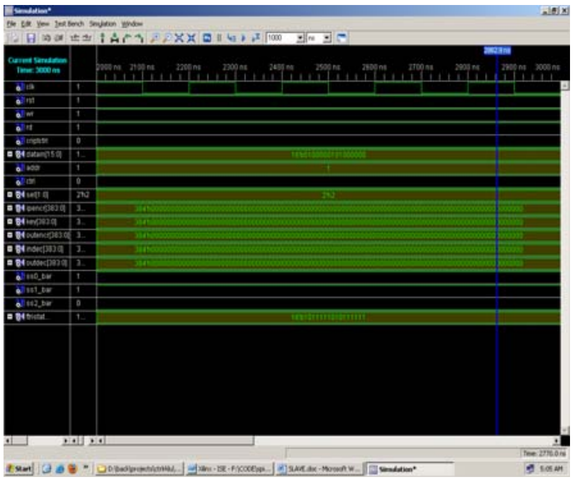


Fig.7. Simulation Results for Tristate Buffers

Here Fig 5, 6 and 7 shows the results of slaves in SPI communication protocol. Here the three slaves are USART, LFSR and tristate buffers. The three are having there own importance in this design.

V.CONCLUSION

Finally in this paper we design high speed and secured SPI Communication Protocol with Scalable Encryption Algorithm (SEA), along with error detection with even parity. Furthermore this Protocol can be applicable for different applications like SOC, CPU and DSP Processors.

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, Springer- 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. Dept. of Commerce, November 26, 2001.