IMPLEMENTING SPI PROTOCOL WITH ENCRYPTED DATA TRASMISSION ON FPGA

SUBMITTED TO:

Dr. P. Prithvi

Assistant Professor – Electronics and communication engineering

Dr. Vadthiya Narendar

Assistant Professor – Electronics and communication engineering

SUBMITTED BY:
Harishwar – 21ECB0B52
S. Mahathi – 21ECB0B53
Saurabh – 21ECB0B54

ABSTRACT

SPI or Serial Peripheral Interface is a communication protocol for serial transmission of data. With the other major communication protocols being I2C and UART. But SPI protocol has the advantage of being a full-duplex communication interface and can transfer data at very high rates (around tens of MHzs).

Also, now-a-days, the authenticity of communication has become a constraint of necessity, especially in the fields of signal processing, etc., where signals of small amplitudes are required to be transmitted with extreme precision. Hence, this project aims at implementing an interface between two devices (namely the master and the slave) where the data to be transferred if first encoded using the hamming code and then transmitted via the SPI protocol. The project focuses on designing the digital circuit of the master and slave device so that they can encode the data and transmit it in the full duplex mode.

TABLE OF CONTENTS

1. INTRODUCTION
2. ABOUT THE SPI PROTOCOL
3. THE HAMMING CODE
    i. ENCODER
    ii. DECODER
4. WORKING
    i. MASTER MODULE – INITIALISATION
        – DATA TRANSMISSION
        – DATA RECEPTION
    ii. SLAVE MODULE – ACTIVATION
        – DATA TRANSMISSION
        – DATA RECEPTION
        – DATA VALIDATION
5. DATA ENCODING / DECODING
6. SIMULATIONS
7. APPLICATIONS
8. FUTURE SCOPE
9. CONCLUSION

CERTIFICATE

This is to certify that the dissertation work entitled **IMPLEMENTING SPI PROTOCOL ON FPGA** is a bonafide record of project work carried out by Harishwar Sangem (21ECB0B52) , Sarvepalli Mahathi (21ECB0B53) and Saurabh (21ECB0B54)submitted to the faculty of 'Electronics and Communication Engineering Department', as a mini project in FPGA based system design lab in "Electronics and Communication Engineering" at National Institute of Technology, Warangal during the academic year (2022-2023).

Dr. NARENDAR SIR,	**Dr. PRITHVI MAM**,
ASSISTANT PROFESSOR,	ASSISTANT PROFESSOR,
DEPT.OF ECE,	DEPT.OF ECE,
NIT WARANGAL.	NIT WARANGAL.

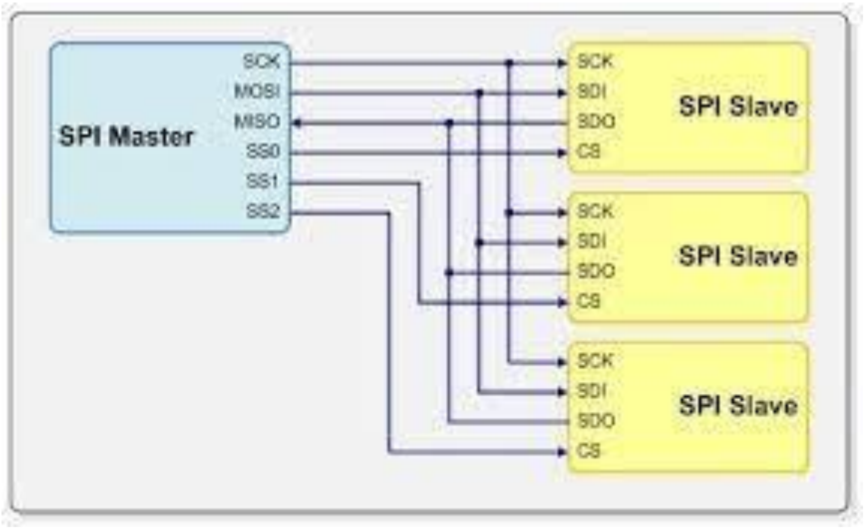
ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my faculty in-charges Smt. P. Prithvi mam, Assistant Professor, and Dr. Narendar sir, Assistant Professor, Department of Electronics and Communication Engineering, National Institute of Technology, Warangal for their constant supervision, guidance, suggestions and encouragement during this seminar.

1. INTRODUCTION

SERIAL PERIFERAL INTERFACE PROTOCOL:

The SPI protocol, as discussed prior is a serial communication protocol. It operates with a single master and multiple slaves.



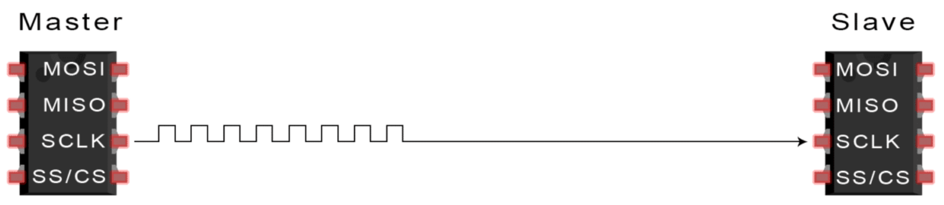
The master is the main controller of the protocol as it generated the synchronising clock pulse, based on which the devices sample or send data.

Each of the slave is connected to the master through 4 wires, which are:

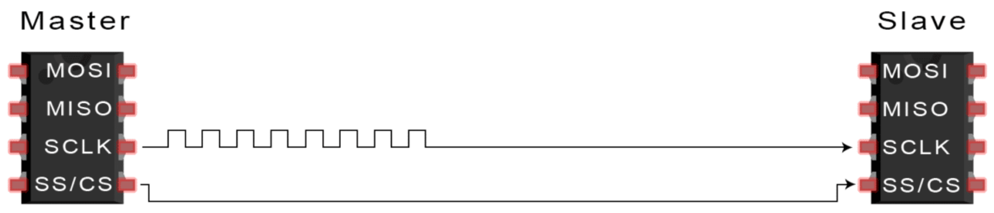
- **SCK**: This is line for the transfer of the synchronising clock pulse, based on which the devices sample or send data.
- **MOSI**: Master Out Slave In is the line through which the master sends data serially to the slave.
- **MISO**: Master In Slave Out is the line through which the slave sends data serially to the master.
- **CS**: This is the chip select or the slave select line. The master contains as many slave select lines as there are slaves as this is the medium possessed by the master to activate a particular slave.

2. ABOUT THE SPI PROTOCOL

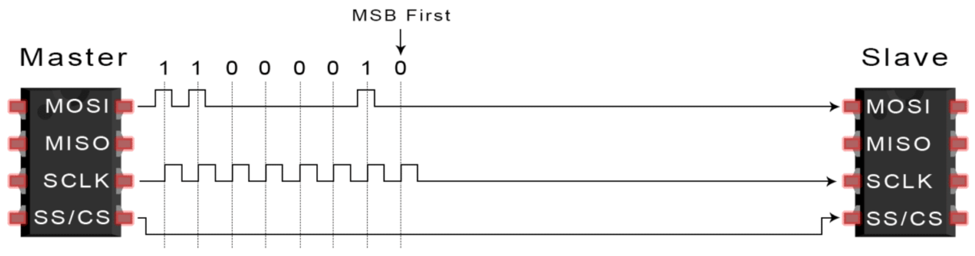
1. The SPI protocol gets initialised when the master firsts starts generating the synchronising clock pulse, this is the basis on which the whole communication occurs.



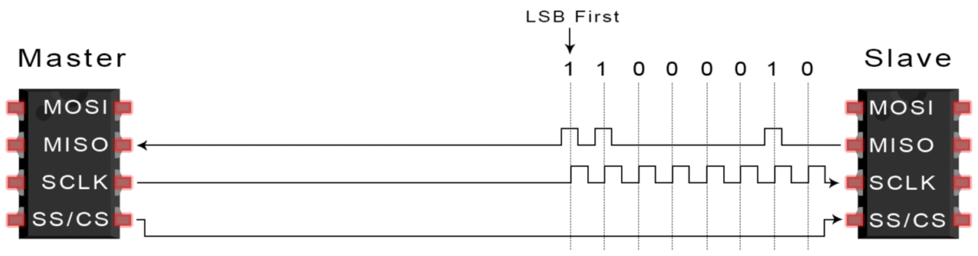
2. After initialisation, the master chooses the slave it needs to communicate with. This is done using the slave select line. The CS line to each of the slaves is HIGH by default, and once the master wants to activate any particular slave, it pulls the CS pin of that slave to LOW until the communication is finished.



3. This is followed by the master or the slave sending the encrypted data bit by bit. The slave transmits through the MISO line and the master transmits through the MOSI line.
Also, the transmission happens in the MSB first format.



4. The received data is then stored in a register.
5. Once the master feels that the communication is completed, it released the CS pin back to HIGH.



Modes of SPI Protocol :

Based on Clock Polarity CPOL and Clock Phase CPHA, there are four operating modes.

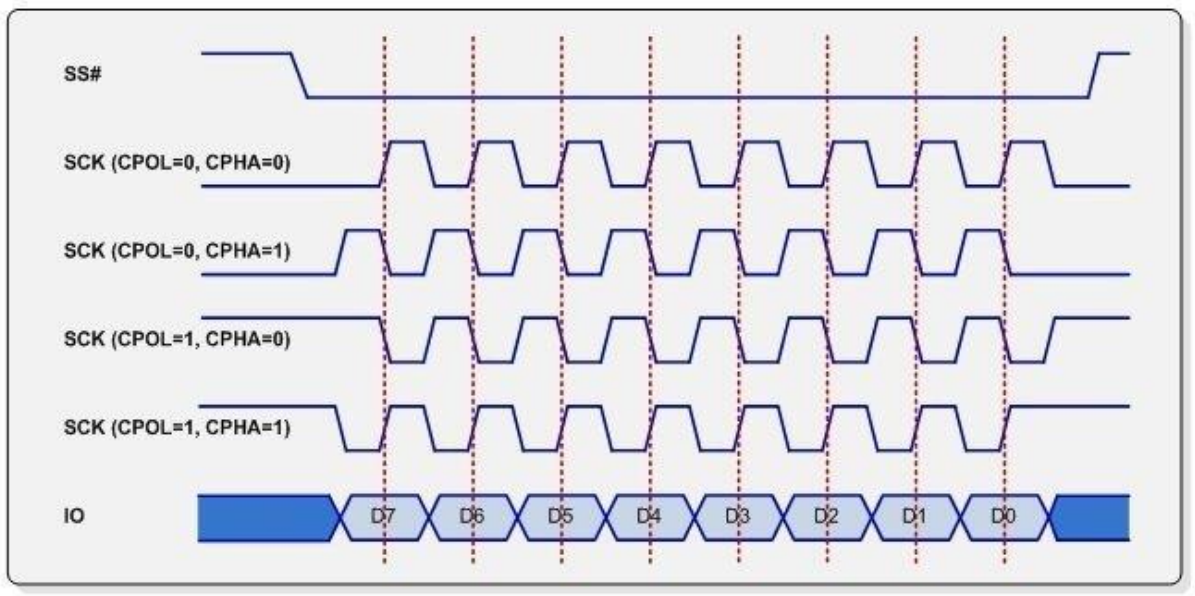
Mode 0: In this mode, the clock polarity (CPOL) is set to 0 and the clock phase (CPHA) is set to 0. This means that the data is sampled on the leading edge of the clock signal (when the clock transitions from low to high) and the data is output on the falling edge of the clock (when the clock transitions from high to low).

Mode 1: In this mode, the CPOL is set to 0 and the CPHA is set to 1. This means that the data is sampled on the trailing edge of the clock (when the clock transitions from high to low) and the data is output on the rising edge of the clock (when the clock transitions from low to high).

Mode 2: In this mode, the CPOL is set to 1 and the CPHA is set to 0. This means that the data is sampled on the leading edge of the clock and the data is output on the rising edge of the clock.

Mode 3: In this mode, the CPOL is set to 1 and the CPHA is set to 1. This means that the data is sampled on the trailing edge of the clock and the data is output on the falling edge of the clock.

The specific mode that is used depends on the requirements of the specific application and the devices that are communicating with each other.



3. THE HAMMING CODE

The hamming code is one of the primary encoder-decoder codes capable of identifying an incorrectly transmitted data byte and rectifying errors upto 1 bit.

Encoder

The Hamming encoder concatenates the given data with parity bits to generate the final code word.
If the number of bits in the data are r then number of parity bits is given by m as follows:

$$2^r \geq m + r + 1$$

Decoder

The decoder generates synchronisation bits by taking the xor of each of the parity bit with its respective message bits. And based on the value of the synchronous array, the position of the erroneous bit is identified and rectified.

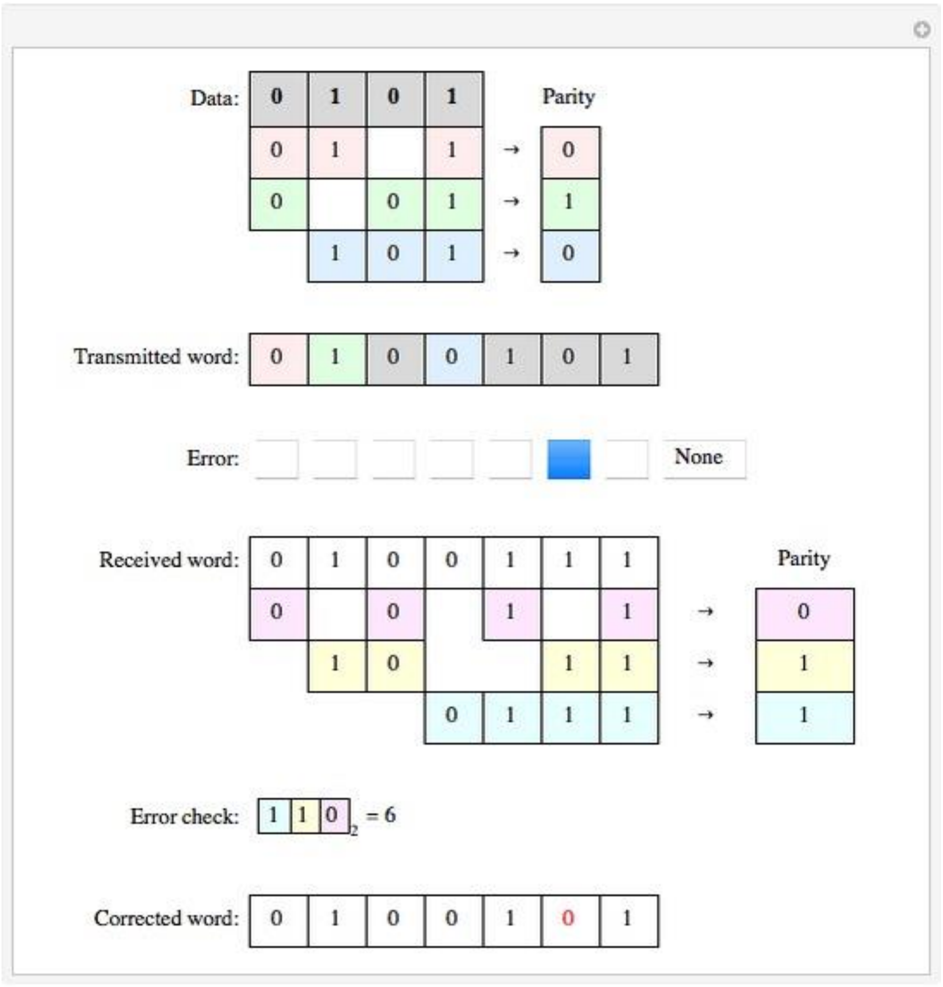


Fig. Schematic of the hamming encoder-decoder

4. WORKING

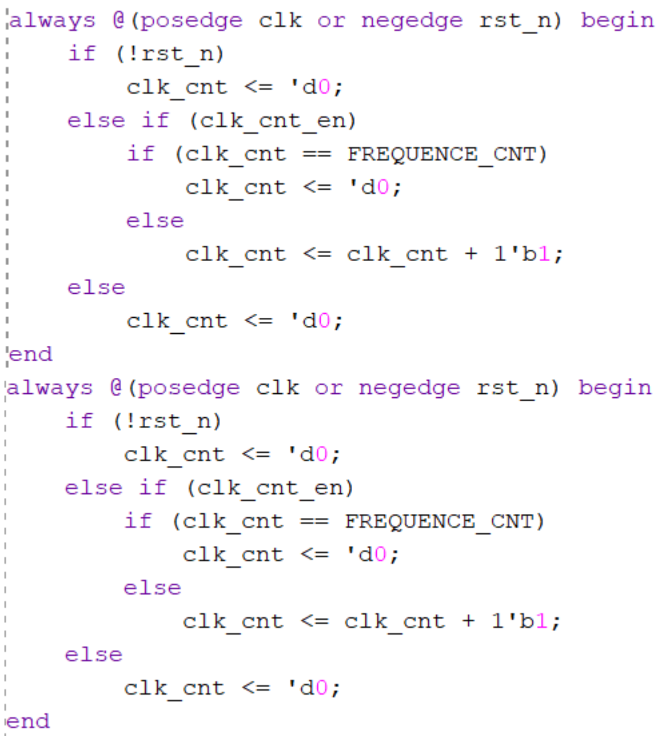
SPI PROTOCOL

The spi protocol uses three modules on the whole for implementation namely – master, slave and loopback. The loopback is an interfacing module between the manual inputs by the user, the master and the slave.

THE MASTER

Initialisation

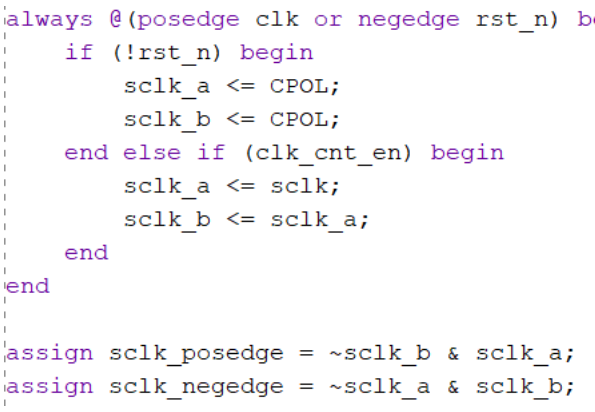
In the SPI protocol, the master controls the synchronising the clock pulse and hence controls the communication in the network. In the project, we use an external clock, controlled manually, based on which the SCK of the master is generated. The frequency of synchronisation is chosen first, based on which the frequency count parameter is given a value. This frequency count parameter controls the clocl_count_enable pin which in turn controls the sclk.



```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        clk_cnt <= 'd0;
    else if (clk_cnt_en)
        if (clk_cnt == FREQUENCE_CNT)
            clk_cnt <= 'd0;
        else
            clk_cnt <= clk_cnt + 1'b1;
    else
        clk_cnt <= 'd0;
end
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        clk_cnt <= 'd0;
    else if (clk_cnt_en)
        if (clk_cnt == FREQUENCE_CNT)
            clk_cnt <= 'd0;
        else
            clk_cnt <= clk_cnt + 1'b1;
    else
        clk_cnt <= 'd0;
end
```

Fig. Generating the sclk

As discussed earlier, the sampling or sending of the data occurs at specific clock edges, as specified by the mode. Hence the capturing of the clock edge is necessary to drive the subsequent actions.



```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        sclk_a <= CPOL;
        sclk_b <= CPOL;
    end else if (clk_cnt_en) begin
        sclk_a <= sclk;
        sclk_b <= sclk_a;
    end
end

assign sclk_posedge = ~sclk_b & sclk_a;
assign sclk_negedge = ~sclk_a & sclk_b;
```

Fig. capturing clock edges

After the initialisation condition i.e., the sclk generation, the master moves into the communication mode.

Data transmission

The data required to be transmitted by the master during the communication is specified by the user through the main module. It is initially stored in a register and is encrypted (through the hamming code) prior to the beginning of the data transmission.

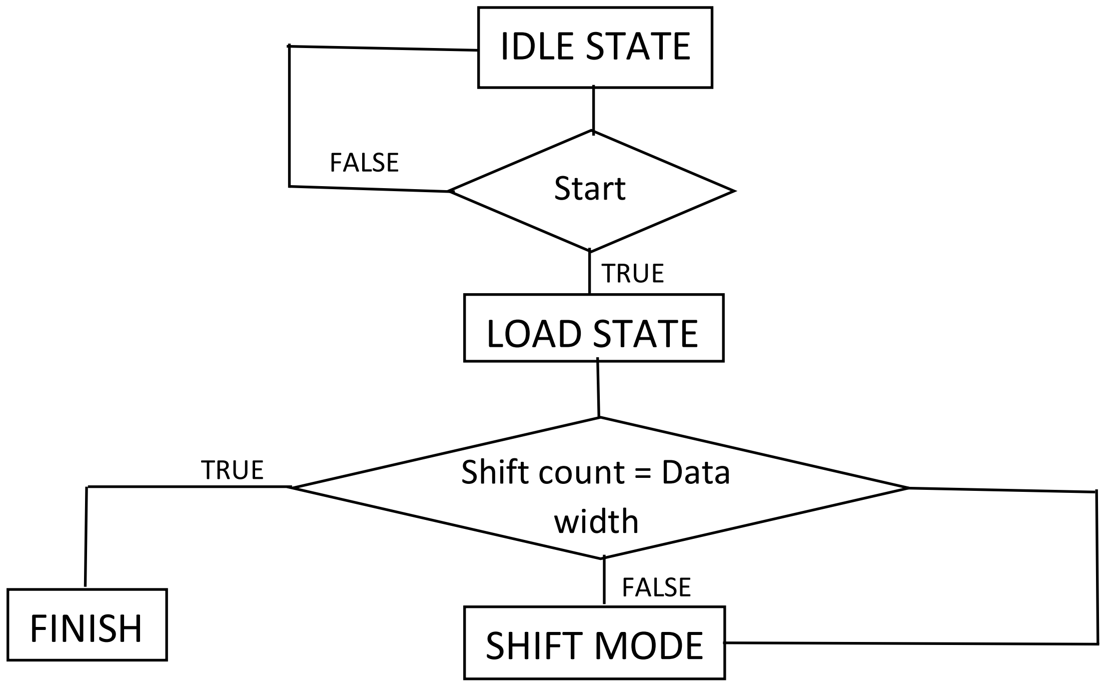


Fig. flow diagram of master in the communication mode where it transmits data to the slave

Each of the modes or states are discussed below.

IDLE state

This is the state prior to the beginning of the communication. Here the default values of all the parameters are established.

```
IDLE     : begin
    clk_cnt_en   <= 1'b0 ;
    data_reg     <= 'd0  ;
    cs_n         <= 1'b1 ;
    shift_cnt    <= 'd0  ;
    finish       <= 1'b0 ;
end
```

LOAD state

Once the start trigger from the main module is given, the master simultaneously loads the data it needs to transmits and activates the required slave select pin.

```
LOAD     : begin
    clk_cnt_en   <= 1'b1     ;
    data_reg     <= data_in  ;
    cs_n         <= 1'b0     ;
    shift_cnt    <= 'd0      ;
    finish       <= 1'b0     ;
end
```

SHIFT state

With the next clock edge (the manual clock provided through the main module), the master moves into the shift state. This is where the actual serial transmission of the data begins. The master used the MOSI line and transmits the data loaded bit by bit, with MSB first. The shift_cnt variable is used to keep track of the number of bits transferred, and once it’s value reaches the number of bits required to be transferred, the transmission stops.

Also, the transmission of bits occurs at the positive/negative edge of the sclk.

```
SHIFT    : begin
    if (shift_en) begin
        shift_cnt    <= shift_cnt + 1'b1 ;
        data_reg     <= {data_reg[DATA_WIDTH-2:0],1'b0};
    end else begin
        shift_cnt    <= shift_cnt     ;
        data_reg     <= data_reg      ;
    end
    clk_cnt_en   <= 1'b1 ;
    cs_n         <= 1'b0 ;
    finish       <= 1'b0 ;
end
```

DONE/ FINISH state

This is the state preceding the termination of the shift state, and is reached once the shift counter touches the value of the data length. Here the concerned parameters are reset to their default values, and with the next clock trigger the system reaches the IDLE state, ready for another transmission.

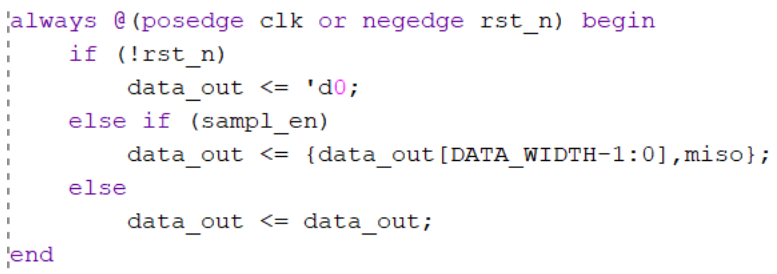
```
DONE     : begin
    clk_cnt_en   <= 1'b0 ;
    data_reg     <= 'd0  ;
    cs_n         <= 1'b1 ;
    data_reg     <= 'd0  ;
    finish       <= 1'b1 ;
end
```

Data reception

With the initialisation of communication and the activation of slave, the master gets enabled to sample the incoming data (along with the transmission of data as seen above).

When the sample_en pin is enables, the master samples the data coming in through the MISO line at the respective edge of the clock as specified by the mode (positive edge in this case).

The data thus sampled is save in a register (data_out in this case).



```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        data_out <= 'd0;
    else if (sampl_en)
        data_out <= {data_out[DATA_WIDTH-1:0],miso};
    else
        data_out <= data_out;
end
```

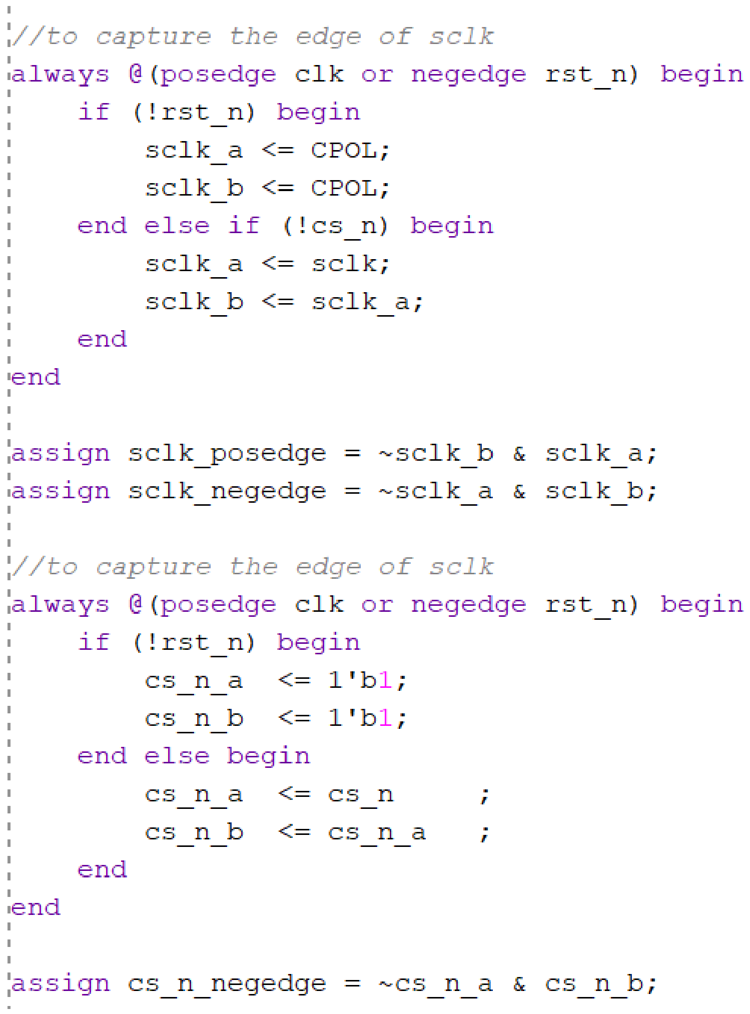
THE SLAVE

Activation (Recognising the addressing by the master)

The slave samples the CS line at every reset or clock edge, and once a negative edge at the slave select line is captured, it moves onto the further processing.

Also, the sclk is sampled to capture the edges, just as it was done in the master, so that the occurrence of the positive and the negative edges can be captured for further applications.

The data required to be transmitted by the slave during the communication is specified by the user through the main module. It is initially stored in a register and is encrypted (through the hamming code) prior to the beginning of the data transmission.



```
//to capture the edge of sclk
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        sclk_a <= CPOL;
        sclk_b <= CPOL;
    end else if (!cs_n) begin
        sclk_a <= sclk;
        sclk_b <= sclk_a;
    end
end

assign sclk_posedge = ~sclk_b & sclk_a;
assign sclk_negedge = ~sclk_a & sclk_b;

//to capture the edge of sclk
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        cs_n_a  <= 1'b1;
        cs_n_b  <= 1'b1;
    end else begin
        cs_n_a  <= cs_n     ;
        cs_n_b  <= cs_n_a   ;
    end
end

assign cs_n_negedge = ~cs_n_a & cs_n_b;
```

Fig. capturing the negative edge of CS and the edges of sclk

Once the negative edge in the CS line i.e., the addressing by the master is recognised, the slave proceeds to obliging the master by either transmitting or receiving data as per a pre-decided design.

Data transmission

In contrast to the multiple states present in the master for the transmission of data, the slave just has two states, it uses an enable signal to start or stop the data transmission instead of a counter. This simplicity in the slave architecture is a major advantage of the SPI protocol.

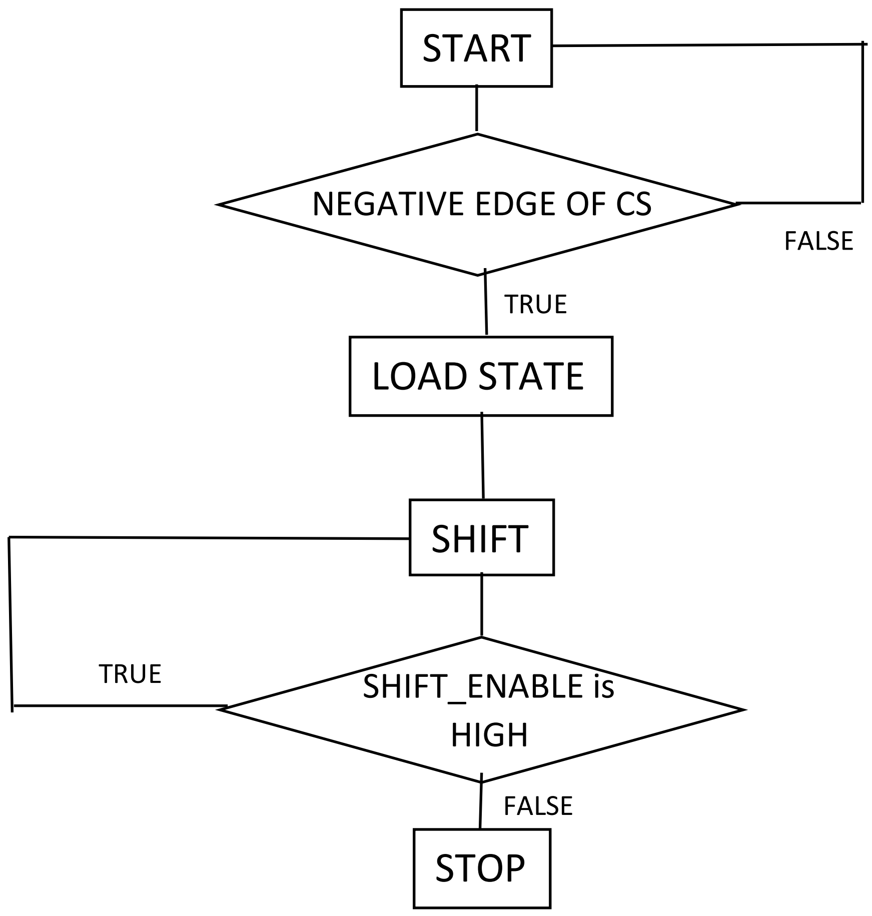
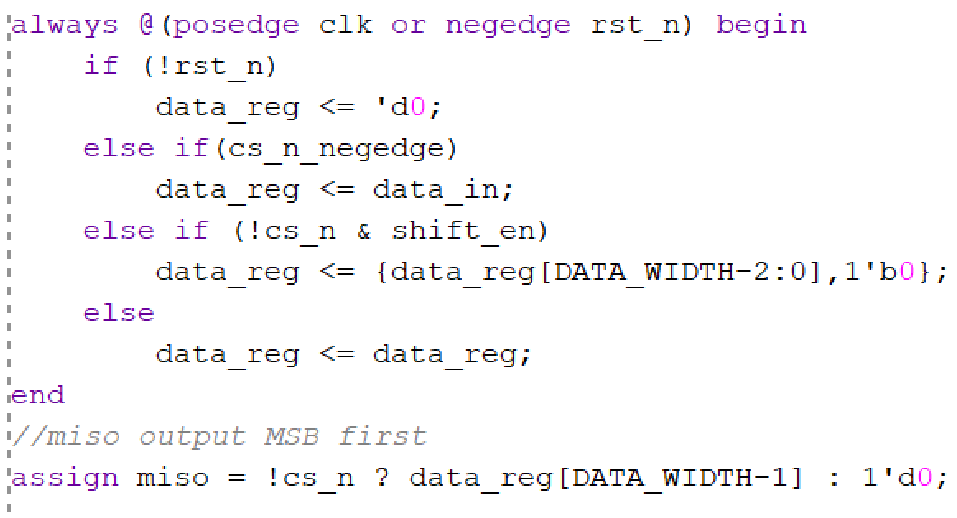


Fig. flow diagram of slave in the communication mode where it transmits data to the master

The algorithmic flow of the slave during the data transmission is as follows:

1. With the reception of a negative edge at the CS pin and the enabling of the shift pin, the encrypted data stored gets loaded into the data_in register.
2. With the next positive edge of the main clock, the slae moves onto the shift mode, wherein the serial transmission of the data takes place in the MSB first format.
3. With every positive or negative edge of the sclk (as specified by the mode) the data loaded gets shifter bit by bit.
4. The transmission carries on until the CS pin remains low and the shift enable pin remains high.

5. Once the converse of the above occurs, the slave ceases the serial transmission of data.



```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        data_reg <= 'd0;
    else if(cs_n_negedge)
        data_reg <= data_in;
    else if (!cs_n & shift_en)
        data_reg <= {data_reg[DATA_WIDTH-2:0],1'b0};
    else
        data_reg <= data_reg;
end
//miso output MSB first
assign miso = !cs_n ? data_reg[DATA_WIDTH-1] : 1'd0;
```

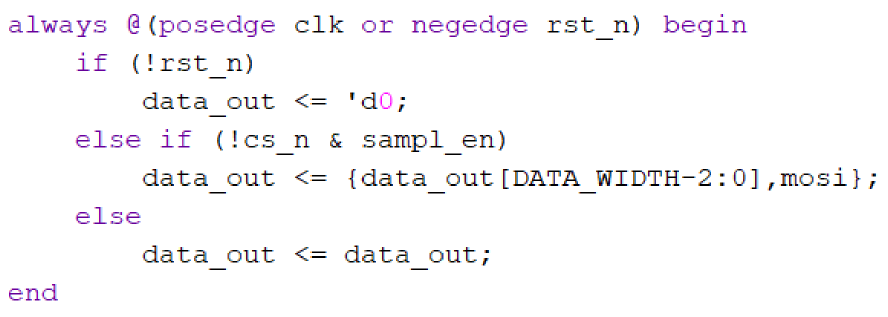
Fig. transmission of data

Data Reception

With the initialisation of communication and the activation of slave, the slave gets enabled to sample the incoming data (along with the transmission of data as seen above).

When the sample_en pin is enables, the slave samples the data coming in through the MOSI line at the respective edge of the clock as specified by the mode (positive edge in this case).

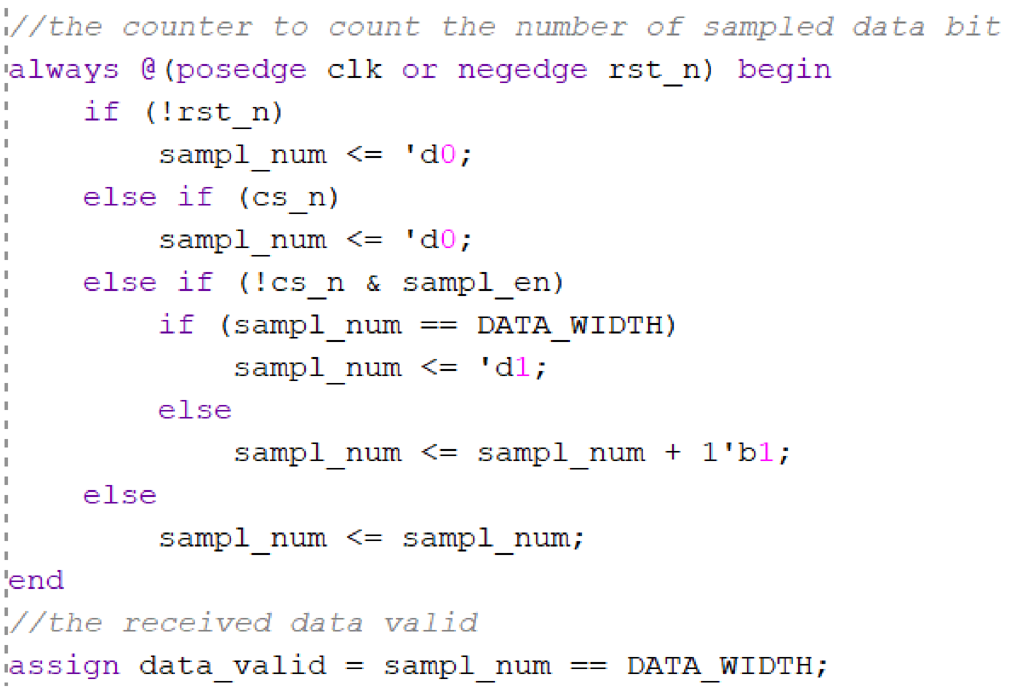
The data thus sampled is save in a register (data_out in this case).



```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        data_out <= 'd0;
    else if (!cs_n & sampl_en)
        data_out <= {data_out[DATA_WIDTH-2:0],mosi};
    else
        data_out <= data_out;
end
```

Data Validation

There is also the design to validate the data received by the slave by counting the number or data bits received and comparing it with the expected data length.



```
//the counter to count the number of sampled data bit
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        sampl_num <= 'd0;
    else if (cs_n)
        sampl_num <= 'd0;
    else if (!cs_n & sampl_en)
        if (sampl_num == DATA_WIDTH)
            sampl_num <= 'd1;
        else
            sampl_num <= sampl_num + 1'b1;
    else
        sampl_num <= sampl_num;
end
//the received data valid
assign data_valid = sampl_num == DATA_WIDTH;
```

Fig. Validating the received data

5. DATA ENCODING/DECODING: The Hamming code

Encoding

Here, the data input given into the master/slave for transmission is a 4-bit data value, encoding which produces a 7-bit data value. Here are the equations for calculating the parity bits of a (7,4) Hamming code

Encoding equation:

r1 = d1

r2 = d2

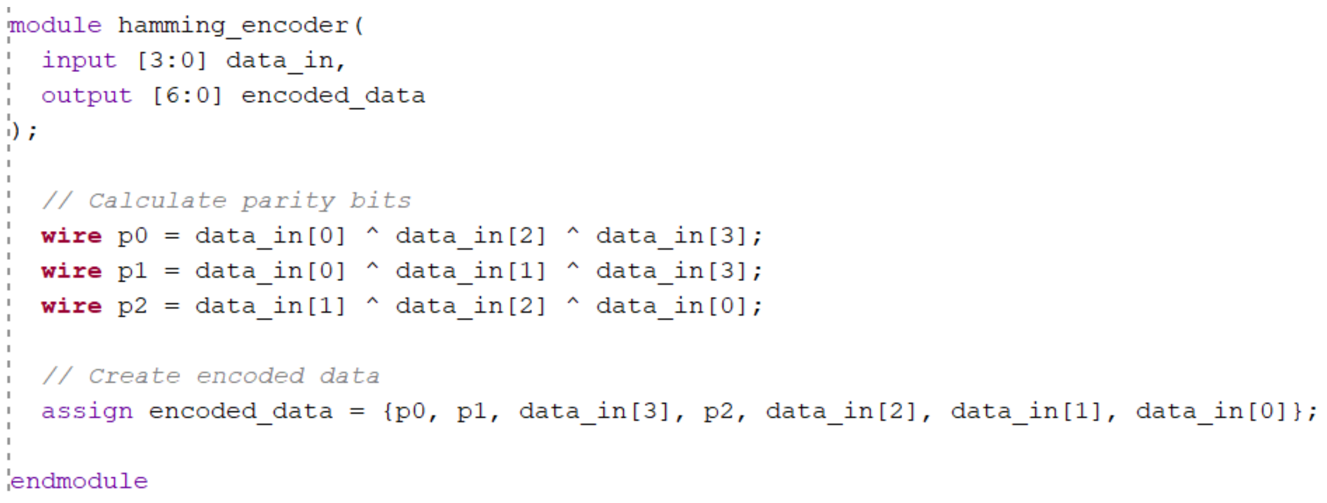
r3 = d3

r4 = d4

r5 = d1 xor d2 xor d4

r6 = d1 xor d3 xor d4

r7 = d2 xor d3 xor d4



```
module hamming_encoder(
  input [3:0] data_in,
  output [6:0] encoded_data
);

  // Calculate parity bits
  wire p0 = data_in[0] ^ data_in[2] ^ data_in[3];
  wire p1 = data_in[0] ^ data_in[1] ^ data_in[3];
  wire p2 = data_in[1] ^ data_in[2] ^ data_in[0];

  // Create encoded data
  assign encoded_data = {p0, p1, data_in[3], p2, data_in[2], data_in[1], data_in[0]};

endmodule
```

Decoder

A Hamming decoder is a circuit that takes in a received message and corrects any errors that may have occurred during transmission. The mathematical equations used in a Hamming decoder depend on the specific Hamming code being used.

The decoding process of the (7,4) Hamming code involves the following steps:

The received 7-bit code word is written out in a row, with each bit represented as either a 0 or a 1.

Parity check equations are used to determine if any errors have occurred in the received code word. The parity bits are calculated using the following equations:

P1 = D1 XOR D2 XOR D4

P2 = D1 XOR D3 XOR D4

P3 = D2 XOR D3 XOR D4

Here, XOR represents the exclusive OR operation, which returns 1 if and only if exactly one of the operands is 1.

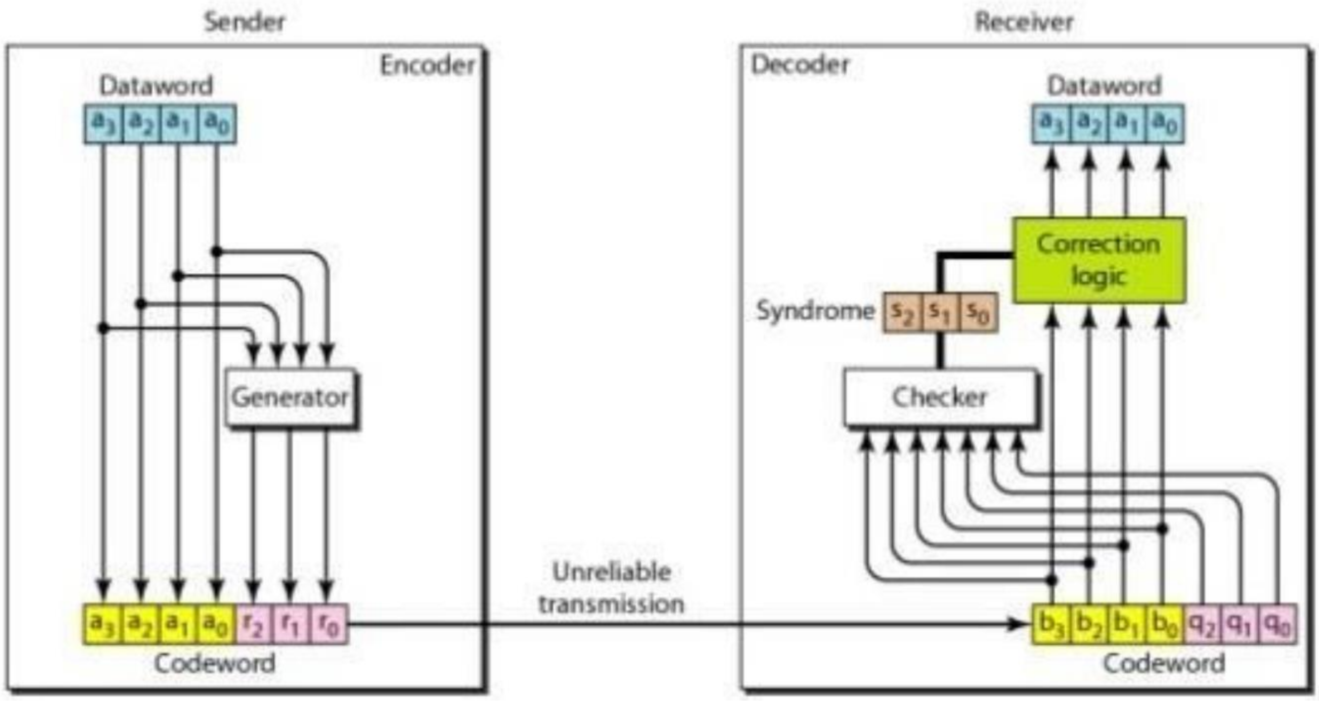
2. The calculated parity bits are compared with the corresponding received parity bits. If there is a difference between any calculated and received parity bit, this indicates that an error has occurred in the corresponding data bit.

3. The position of the erroneous bit can be identified by looking at the binary representation of the position of the incorrect parity bit. For example, if the first parity bit (P1) is incorrect, then the binary representation of the position of the incorrect parity bit is 001 (since it is the first bit). This corresponds to the position of the erroneous data bit (D1).

4. The erroneous data bit is corrected by flipping its value from 0 to 1 or from 1 to 0.

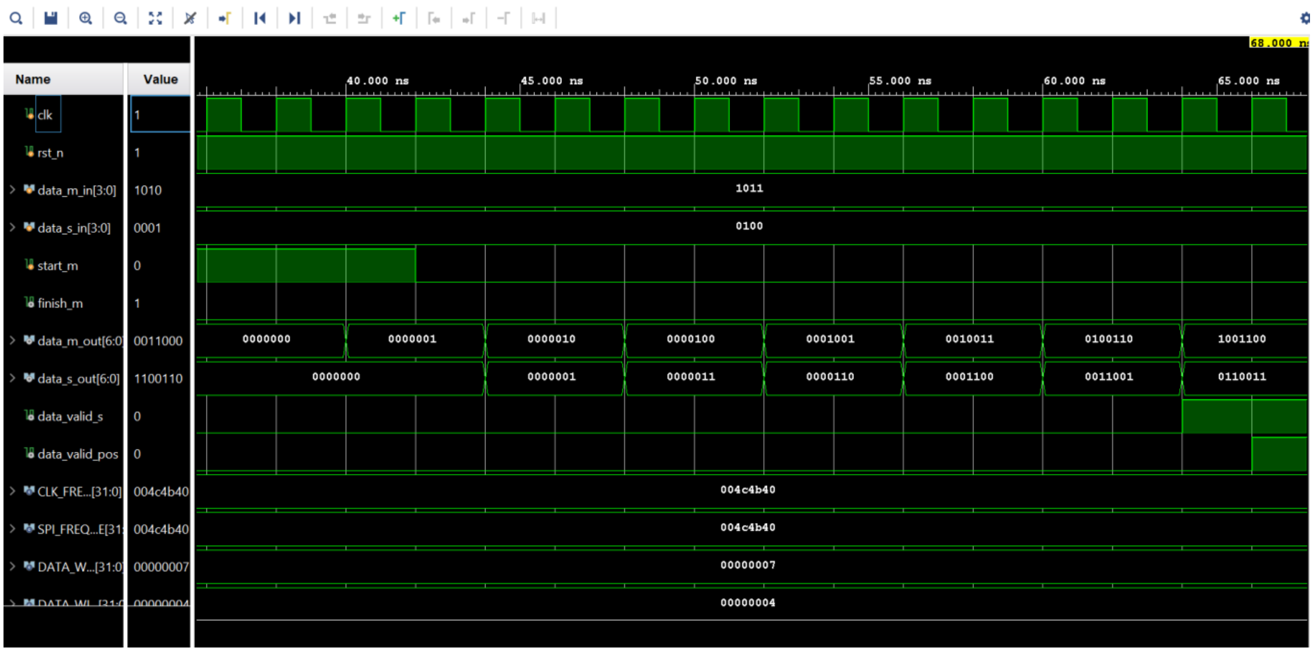
Overall, the (7,4) Hamming code is a simple but effective error-correcting code that can detect and correct a single-bit error in a block of four data bits.

```
module hamming_decoder(
  input [6:0] hamming_bits,
  output reg [3:0] decoded_bits, output reg error);
reg [6:0] hamming_bits_store;
  // Calculate syndrome bits
  reg [2:0] syndrome;
   always@(*)
  begin
  syndrome[0] = hamming_bits[0] ^ hamming_bits[2] ^ hamming_bits[4] ^ hamming_bits[6];
  syndrome[1] = hamming_bits[1] ^ hamming_bits[2] ^ hamming_bits[5] ^ hamming_bits[6];
  syndrome[2] = hamming_bits[3] ^ hamming_bits[4] ^ hamming_bits[5] ^ hamming_bits[6];
  hamming_bits_store=hamming_bits;


  // Correct bit if syndrome indicates an error

  if (syndrome != 0) begin
    error = 1;
    hamming_bits_store[syndrome-1] = ~hamming_bits_store[syndrome-1];
  end else begin
    error = 0;
  end

  // Decode data bits
  decoded_bits[0] = hamming_bits_store[2];
  decoded_bits[1] = hamming_bits_store[4];
  decoded_bits[2] = hamming_bits_store[5];
  decoded_bits[3] = hamming_bits_store[6];
end
  //return decoded_bits;
endmodule
```

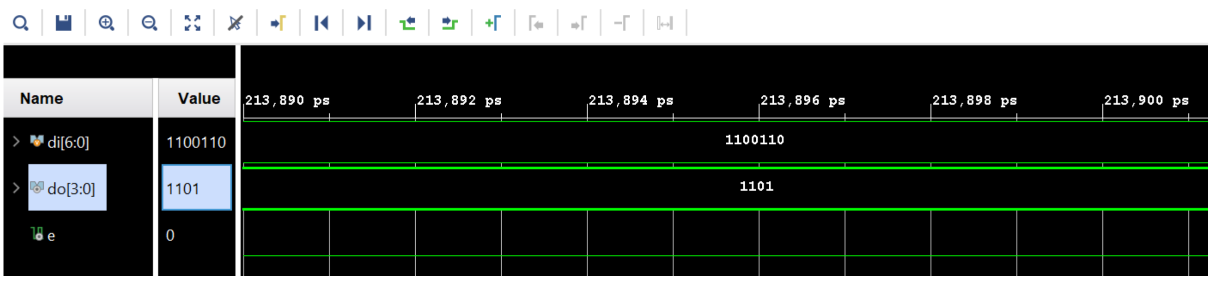


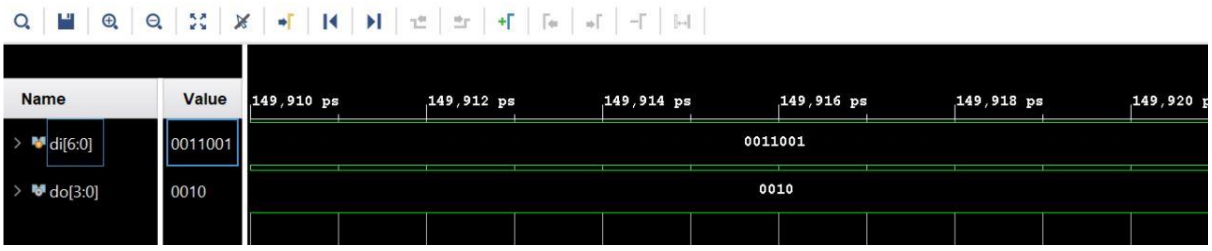
6. SIMULATIONS

The following is the simulation of the transmission of data ‘1011’ from the master and ‘0100’ from the slave.



The authenticity of the received output can be validated through the decoder module as shown below:





7. APPLICATIONS

The SPI protocol with encrypted data transmission can be used in areas where sensitive signals have to be transmitted with high accuracy, minimal hardware and high speed. Some of the application include:

Interfacing with Sensors: SPI is commonly used for interfacing with sensors such as accelerometers, gyroscopes, and temperature sensors. These sensors often require high-speed communication and low latency, making SPI an ideal choice.

Memory Devices: SPI is used for interfacing with memory devices such as EEPROM, Flash memory, and SRAM. The protocol's high-speed data transfer capabilities make it ideal for reading and writing large amounts of data.

Audio and Video Applications: SPI is used for audio and video applications, such as controlling audio codecs, LCD displays, and graphics controllers. The protocol's high-speed data transfer capabilities make it suitable for transferring large amounts of data in real-time.

Control Systems: SPI is used for control systems, such as motor control, where high-speed communication is required between the microcontroller and motor driver.

Networking and Communication: SPI is used in networking and communication applications, such as Ethernet controllers and wireless transceivers. The protocol's high-speed data transfer capabilities make it ideal for transferring large amounts of data quickly and efficiently.

8. FUTURE SCOPE

Replacing the hamming code encryption with more secure encryption algorithms like the AES.

Differentiating the slave and master modules and implementing one on each FPGA to establish practical connection between two digital devices through the SPI communication protocol.

Expanding the architecture of the master to cater for multiple slaves and multiple modes of transmission.

9. CONLUSION

The project 'Implementation of SPI protocol with encrypted data transmission' aims at designing the digital system possessed by a master and a slave present in a network connected with the SPI protocol. With the externals controls like the system clock, data for transmission and trigger to indicate the master to start the transmission being given by the user through the pins on the FPGA board.

Additionally, it adds an encryption component to the data being transmitted to ensure authenticated and secure data transmission.