Implementatio of SPI and UART protocols

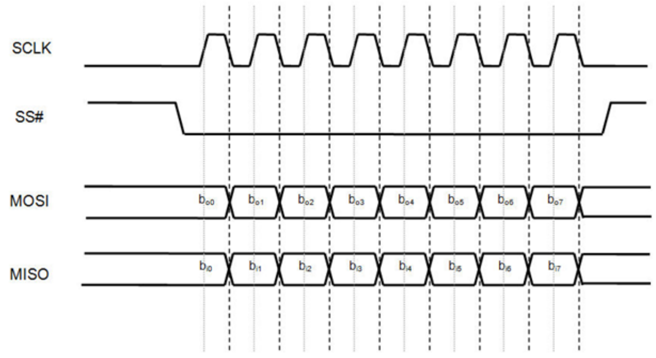
S Sai Ashish, PNVSSK Havish

***Abstract*— This is a brief report on the implementation of SPI and UART communication protocols.**

I. SPI Protocol

A. Outline of Protocol

- SPI stands for serial peripheral interface.
- Some times SPI protocol is also called a 4-wire protocol. It requires four wires MOSI, MISO, SS, and SCLK.
- The master first configures the clock using a frequency. The master then selects the particular slave device(using SS) for communication by pulling the chip select button. That particular device is selected and starts the communication starts between master and that particular slave. The master select only one slave at a time.
- The master sends out data on the MOSI line and listens to data from slave on the MISO line.
- It is full duplex communication protocol. Not limited to 8 bit words in the case of bit transferring.


SCLK
SS#
MOSI
MISO
b$_{o0}$ b$_{o1}$ b$_{o2}$ b$_{o3}$ b$_{o4}$ b$_{o5}$ b$_{o6}$ b$_{o7}$
b$_{i0}$ b$_{i1}$ b$_{i2}$ b$_{i3}$ b$_{i4}$ b$_{i5}$ b$_{i6}$ b$_{i7}$

B. Our implementation

- We used the icoboard(FPGA) as master and two arduinos as slaves.
- In our implementation we sent data from master to slaves (MOSI active; MISO inactive). In the similar way we can modify the code to send data from slave to Master.
- The clock frequency was set to 2MHz
- We sent an 8 bit integer from the master to the slaves by selecting one slave at a time.

C. Code

The following is the master code(written in verilog)

```
module example(input wire scl,
output reg[0:0] mosi,
output reg ss1, output reg ss2);
reg[20:0] delay;
reg[27:0] counter=28'd0;
parameter DIVISOR = 28'd50;
reg[0:15] xx = 16'b1100011100101001;
reg[20:0] delay1 = 21'b011110100001001000000;
reg[4:0] A ;
reg[3:0] B;
reg[1:0] flag;
initial begin
A = 0;
ss1 = 1;
ss2 = 1;
delay=0;
flag = 0;
B = 0;
end
always@(posedge scl) begin
        if(A<=7) begin
                if(ss1!=0)ss1 = 0;
                if(ss2!=1)ss2 = 1;
                mosi = xx[A];
                A = A + 1;
        end
        else if(A>=8 && A<=15) begin
                if(ss1!=1) ss1 = 1;
                if(ss2!=0) ss2 = 0;
                mosi = xx[A];
                A = A + 1;
        end
        else begin
                if(A==16) begin
                ss1 = 1;
                ss2 = 1;
                end
                if(delay==delay1)
                begin
                        A = 0;
                        delay = 0;
                end
                delay = delay + 1;
        end
end
endmodule
```

The slave(arduino) code is as follows:

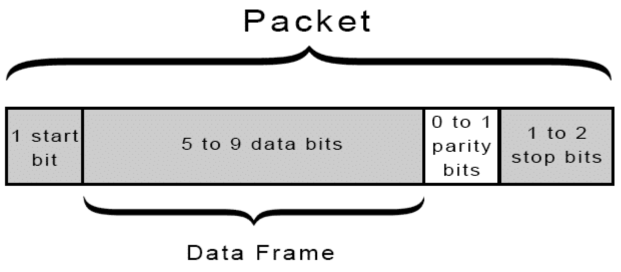
```
#include<SPI.h>

volatile boolean received;
volatile byte Slavereceived,Slavesend;
volatile char recvd;
int x;
void setup()

{
  Serial.begin(115200);

  pinMode(MISO,OUTPUT);
  pinMode(SS,INPUT);
  SPCR |= _BV(SPE);
  received = false;
  SPI.setClockDivider(SPI_CLOCK_DIV8);
  // clock is divided and set to 2 MHz
  digitalWrite(SS,HIGH);
  //Interrupt ON is set for SPI
  SPI.attachInterrupt();
  SPI.setDataMode(SPI_MODE1);
  SPI.setBitOrder(MSBFIRST);
}

//Interrupt routine function
ISR (SPI_STC_vect)
{
  Slavereceived = SPDR;
  recvd = SPDR;
  // Value received from master is stored
  received = true;
//Sets received as True
}

void loop()
{
  if(received)
   {
     // print the received value
     Serial.println(Slavereceived);
     received = false;

}
}
```

- The code basically sends the first 8 bits of a 16 bit number to slave 1 and the next 8 bits to slave 2.
- Once all the 16 bits are sent out, there will be a delay of 1 second. This is to make sure that the data bits do not get overlapped.

II. UART PROTOCOL

A. *Outline of Protocol*

- UART or Universal Asynchronous Receiver Transmitter is a serial communication device that performs parallel to serial data conversion at the transmitter side and serial to parallel data conversion at the receiver side.
- It is universal because the parameters like transfer speed, data speed, etc. are configurable.
- There is no clock used to synchronize the data transmission between sender and the receiver, rather there is something known as baud rate. Baud Rate is measured in bits per second.
- This is the serial communication technique used in serial data transmission in Arduino and various other microcontrollers to various sensors and vice-a-versa.
- **Start Bit**: Start bit is a synchronisation bit that is added before the actual data. Start bit marks the beginning of the data packet. In order to start the data transfer, the transmitting UART pulls the data line from high voltage level to low voltage level (from 1 to 0). After this 8 bits of data is sent from transmitter to receiver.
- **Stop Bit**: The Stop Bit, as the name suggests, marks the end of the data packet. In order to end the transmission, the UART maintains the data line at high voltage (1).



B. *Our implementation*

- The icoboard(fpga) acts as transmitter and the arduino acts as a receiver.
- We implemented this as a simple 4 state FSM which transitions betwen the states IDLE, START, DATA and STOP state. The states are self explanatory.
- The first part of the code sets the required baud rate for communication.
- We sent a ascii byte which represents an alphabet from FPGA to arduino.

C. *Code*

```
module transmitter(
                    input wire clk_50m,
                    output reg tx,
                    output wire tx_busy);
reg clken;
reg [7:0] din;
initial begin
        tx = 1'b1;
        din = 8'b01000001;
end
parameter TX_ACC_MAX = 100000000 / 115200;
parameter TX_ACC_WIDTH = $clog2(TX_ACC_MAX);
reg [TX_ACC_WIDTH - 1:0] tx_acc = 0;
```

```
assign clken = (tx_acc == 9'd0);

always @(posedge clk_50m) begin
if (tx_acc ==
    TX_ACC_MAX[TX_ACC_WIDTH - 1:0])
        tx_acc <= 0;
else
        tx_acc <= tx_acc + 9'b1;
end
parameter STATE_IDLE = 2'b00;
parameter STATE_START = 2'b01;
parameter STATE_DATA = 2'b10;
parameter STATE_STOP = 2'b11;
reg [7:0] data = 8'h00;
reg [2:0] bitpos = 3'h0;
reg [1:0] state = STATE_IDLE;
reg wr_en = 1;
always @(posedge clk_50m) begin
case (state)
STATE_IDLE: begin
if (wr_en) begin
    state <=STATE_START;
    data <= din;
    bitpos <= 3'h0;
    wr_en = 0;
        end
end
STATE_START: begin
if (clken) begin
        tx <= 1'b0;
        state <= STATE_DATA;
end
end
STATE_DATA: begin
if (clken) begin
        if (bitpos == 3'h7)
                state <= STATE_STOP;
        else
                bitpos <= bitpos + 3'h1;
                tx <= data[bitpos];
end
end
STATE_STOP: begin
if (clken) begin
        tx <= 1'b1;
        wr_en = 1;
        state <= STATE_IDLE;
end
end
default: begin
        tx <= 1'b1;
        state <= STATE_IDLE;
end
endcase
end
assign tx_busy = (state != STATE_IDLE);
endmodule
```

The arduino(receiver) code is as follows:

```
void setup()
{
  Serial.begin(115200);
}
void loop()
{
  if(Serial.available()>0){
  byte a = Serial.read();
  char c = a;
  Serial.println(c);

  }
  else{
    Serial.println("failed");
    }
//  delay(1000);
}
```

- As we can see the positive edge clock where state transition occurs is nominal and the data transfer is independent of the clock. This is the advantage of an asynchronous clock.
- The arduino acts as a receiver and it listens through the serial port.

III. REFERENCES

- http://www.circuitbasics.com/basics-uart-communication/
- http://www.circuitbasics.com/basics-of-the-spi-communication-protocol/
- Link for codes: github.com/havish99