TRANSMISSION OF BULK DATA OVER USB TO SD CARD
Final Report

Submission Date: 5/4/2017
ECE 337:Thursday 2:30 PM Lab
Ta: Kyle Ziga

PARTNER NAMES:

Apoorv Vidhu Sharma

Pujitha Desiraju

Kunwar Digraj Singh Jain

Pushkal Vaid

Executive Summary:

Data is essential in every aspect of industry solutions and technological advances. It is estimated that approximately 2.5 Quintillion bytes of data is created everyday and a substantial amount is transferred every second from one device to another. It is as important to be able to have portable data and a mechanism to transfer blocks of data in large quantities over reliable transmissions to a safe storage device. This gives rise to the usage of the Universal Serial Bus to transfer bulk data to and from an SD Card for safe storage.

This simple transmission protocol will enable company employees to carry around huge chunks of data which would enable them to perform tasks anywhere and at any time.

The proposed project will be designed as a peripheral to computers. The design works bidirectionally. The user is able to extract data from an SD Card as well write to it. The design will have dedicated hardware for data which is sent/received over a usb and finally written to/read from an SD card. This will prevent unnecessary memory and processor transactions. This also reduces the processing time since tasks can be completed parallely. Hence aSIC seems the best option to support the proposed design.

Successful design of the proposed accelerator will require the following resources:

- Universal Serial Bus 1.1 Specification Documents
- SD Card interfacing documentation
- Reference Standard Cell Technology Library for Final Design Layout Verification
- Verilog HDL Simulation and Design Synthesis Tool Chain

The following document contents will describe:

- Intended usage expectations and constraints for the design
- Intended system usage and design architecture models to depict flow of control and data through the design.

Design Specifications:

1. **System Usage**

- **System Usage Diagram:**

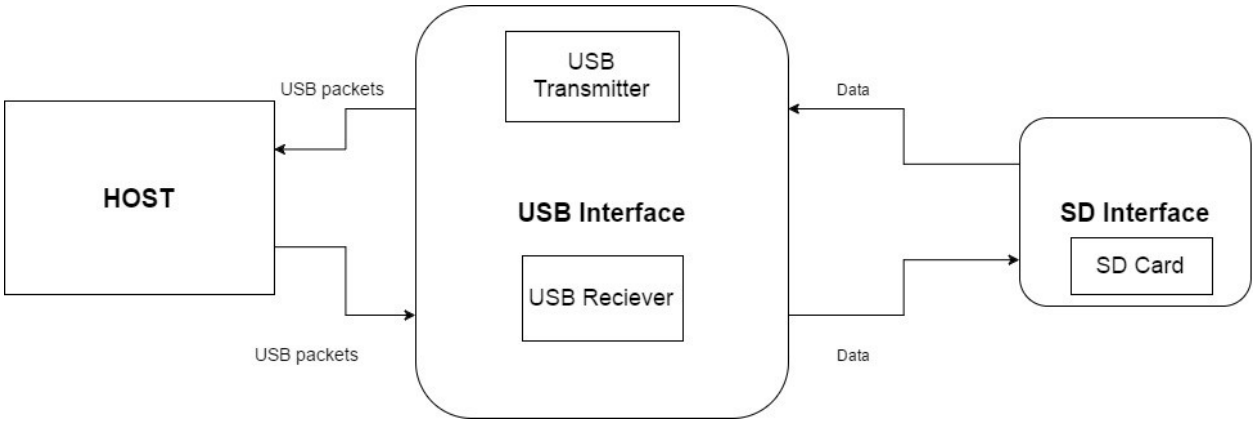


Figure 1:Example System Usage Diagram for SD Card Interface - USB Interface

An example system illustrating the intended use of the transmission can be seen in the diagram depicted above (Figure 1). In the system there are 3 main blocks , with the project mainly residing the USB and SD interface blocks. The first block , host will send D+ and D- lines to the USB interface(USB receiver). Once the receiver has received the data, it will send a handshake signal (aCK/NaCK) back to the host via the Transmitter which is received by the host in the form of D+ and D- lines , signalling that the packet of data has been successfully/unsuccessfully comprehended.The transmitter also sends Data packets to the host when signalled to read data from the SD Card. Depending on the packet being sent by the host,(IN/OUT/DaTa packet) , the receiver asserts signals to alert the USB interface of the next steps in communicating with the SD Card interface if and when it should. The data is then stored in the FIFOs within the USB interface and sent to the SD Card or read from the SD card.

The rate of transfer of data for USB 1.1 is approximately 12 Mbps and therefore the SD Card whose data transmission rate can go upto 80 Mbps has been chosen to be at approximately 12 Mbps.

The chosen model for our design is the TS2GSDC, SD Card

- **Implemented Standard(s) and Algorithms(s)**

- Protocols
    - USB (1.1) :
        - Can be run on several devices
        - Bulk data transfer protocol implemented
        - High Speed (12Mbps)
        - Traditional serial ports at such high speeds are subject to data loss and missed interrupts however USB has none of these problems
        - Packet starts with a SYNC field, defined as “10000000” is a byte long
        - The PID field follows(totally a byte with the first 4 bits depicting the PID and the last 4 bits as its complement), which states the nature of the received packet.
        - Packet specific information(data/device address/endpoint numbers) is sent after the PID field
        - The CRC field then follows which can be either 5 bits or 16 bits depending on the nature of the packet. The CRC is generated and checked for the packet-specific data only,since the PID is checked independently.
        - The packet then contains the EOP(End of Packet) which is 3 data bit durations long , followed by a bit period of IDLE state
        - Bytes of data are broken up and LSB is sent first and MSB is sent last

    - SD Card
        - Extremely portable
        - SPI mode
        - Compatible to with 12 Mbps
        - SD Cards run at 0 - 25 MHz (Default speed)
        - 48 bit commands with 8 bit reponses
        - Data is read in block with size 8 bytes (default size - 512 bytes)

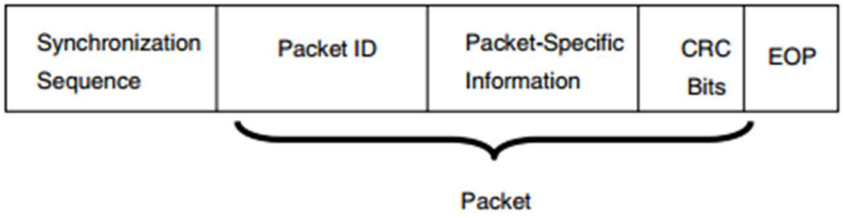
- **Design PinOut:**

1. USB Module:

Table 1: Pin Description for USB Module

Pin one side	Signal	Pin Description
1	USB Vcc (Vbus)	Power Pin
2	USB Data -	2
3	USB Data +	3
4	GND	Ground Pin

USB transmission format:



SD Card Module:

Table 3: SanDisk 32GB Ultra Class 10 SDHC UHS-I Memory Card modes of operation

Pin #	SD 4-bit Mode		SD 1-bit Mode		SPI Mode	
1	CD/DaTa [3]	Data Line 3	N/C	Not Used	CS	Card Select
2	CMD	Command Line	CMD	Command Line	DI	Data Input
3	VSS1	Ground	VSS1	Ground	VSS1	Ground
4	VDD	Supply Voltage	VDD	Supply	VDD	Supply

				Voltage		Voltage
5	CLK	Clock	CLK	Clock	SCLK	Clock
6	Vss2	Ground	Vss2	Ground	Vss2	Ground
7	DaT[0]	Data Line 0	DaTa	Data Line	DO	Data Output
8	DaT[1]	Data Line 1 / Interrupt	IRQ	Interrupt	IRQ	Interrupt
9	DaT[2]	Data Line 2 /Read Wait	RW	Read Wait	NC	Not Used

Operational Characteristics

USB 1.1 Operations:

1. Overall Design

Our project involves the bulk data transfer protocol of the USB and involves the following specifications:

The host initially sends the USB module, D+ and D- lines which are then decoded using NRZi encoding.USB serial data is NRZI (Nonreturn to Zero, Inverted) decoded before being transferred via the USB cables using the differential signaling. Decoding and differential signaling are used to help ensure data integrity and eliminate noise problems , without requiring a separate clock signal to be delivered with the data.

Data is then driven onto the USB cable(USB Receiver Module) by the differential driver. The receiver amplifies the incoming differential data and delivers the NRZI data to the decoder. Transition in the NRZI data stream represents 0s while no transition represents 1s. The NRZI decoder must maintain the synchronization with the incoming data stream to correctly sample the data. The NRZI data stream must be sampled within a data window to detect whether a transition has occurred since the previous bit time. The decoder samples the data stream during each bit time to check for transitions. The shift enables are provided by the timer module. Once the data is decoded , it is then passed through the Bit Destuffer module to eliminate stuffed 0's after 6 1's. The data is then passed to the Receiver control unit to be comprehended. at the same time , when the receiver decided that it is a data packet, data is parallelly sent to the CRC Checker to know if it is legitimate and a status is sent back to the controller.

after comprehending the type of packet received , the controller signals the BD Controller to take further actions.

1. The BD Controller after receiving an OUT packet activates the transmitter to send an aCK signal, after checking that the receiver received an aCK packet the receiver now knows to expect the data packet.after which the controller signals the transmitter to send an ACK packet back.

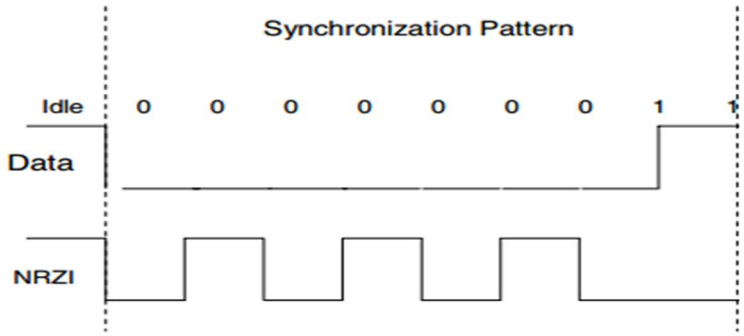
2. The BD Controller after receiving an IN packet activates the transmitter to send an aCK signal, after checking that the transmitter transmitted an aCK packet the BD controller signals the transmitter to send a data packet immediately following which it alerts the receiver to accept an aCK packet.

Within the USB transmitter , the controller decided the sequence of steps once the BD Controller lets it know the type of packet to be sent out. If a handshake packet is to be sent , the transmitter loads the SYNC/PID byte in the PTS register which shifts out bits per bit period as sampled by the timer module. If the transmitter has to send out a data packet, the data is loaded from the FIFO and is sent to the PTS register . The CRC generator parallelly accepts data and generates the CRC sequence which it then sends to the PTS to load into the register.

Data then passes through the Bit Stuffer which checks for 6 consecutive 1's and then pauses the timer module or indirectly the shifting of the bits to shift out a 0. The bits are then passed through the NRZI decoder, after which the D+ and D- lines are sent back to the host.

Transactions typically consist of three phases, or packets, as illustrated in the figure below, However, a transaction may consist of one, two, or three phases depending on the type:

1. **Synchronization Sequence**: The synchronization sequence consists of eight bits starting with seven consecutive logic 0s and ending with logic 1. The synchronization sequence also alerts USB receivers that a packet is being sent, which will immediately follow the 8-bit synchronization sequence. The USB receiver must detect the logic state of each bit value within the packet by sampling the data lines at the correct point during each bit time.



2. **Packet Identifier**:
   - **Token packets** are sent at the beginning of a USB transaction to define the transfer type. (E.g. transfer to or from a USB device)

     1. IN packet : Sent by the host to the USB Receiver signalling that the host wants to read data from the SD Card.
     2. OUT packet: Sent by the host to the USB Receiver signalling that the host wants to write data to the SD Card.
   - **Data packets** follow token packets during transactions that require data payloads be transferred to or from USB devices.

   - **Handshake packets** are typically returned from the receiving agent back to the sender, thus providing feedback relative to the success or failure of the transaction.These packets are sent by the USB transmitter :
     1. aCK - acknowledgment that the packet has been successfully received.
     2. NaK - Reports that the device temporary cannot send or received data.

3. **Packet Specific Information**: Each packet contains information that is related to the job it performs. The information may consist of a USB device address, a frame number, data to be transferred to or from the USB device, etc.

4. **End of Packet**: The end of each packet is signaled by the sending agent by driving both differential data lines low for two bit times followed by an idle for 1-bit time. The agent receiving the packet

The format of the IN packet is : SYNC,PID,DEVICE aDDR,ENP,CRC,EOP
The format of the OUT packet is : SYNC,PID,DEVICE aDDR,ENP,CRC,EOP
The format of the DaTa packet is : SYNC,PID,DaTa,CRC,EOP
The format of the HaNDSHaKE packet is : SYNC,PID,EOP

- **USB 1.1 Transmitter:**

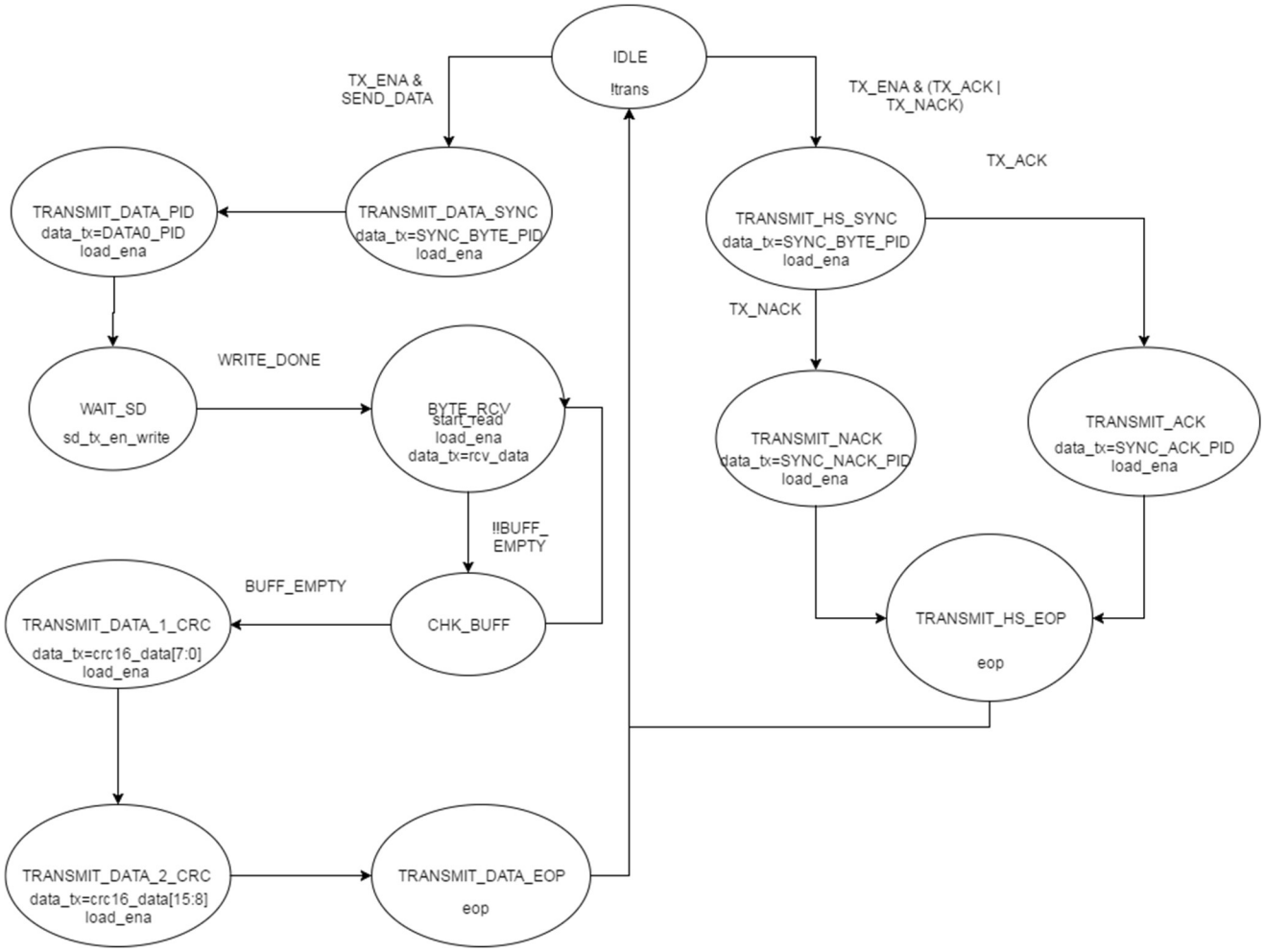


Figure 2:Transition Diagram of USB Transmitter

In the above state transition diagram , when the tx_ena signal is asserted , the transmitter is ready to perform a function and asserts a load_sync to transmit a data sync packet as is common with all the packets the transmitter transmits. The load_sync signals the PTS Shift register to load a sync byte. It then asserts load signals which denote the appropriate PID based on the inputs from the Bulk Data Controller , signalling it to either send a Data packet or a Handshake packet(ack or Nack) packet. If the transmitter has to transmit a data packet , it first signals the SD Card to write to the buffer and once the SD Card signals back that it has finished writing to the buffer, the transmitter enables the FIFO Data buffer to shift out a byte and load that data byte to the PTS Shift Register. after every load enable , the

transmitter enables the Timer block to start counting bit cycles which last for 8 system clock cycles and also count the number of bits received and assert byte_rcvd when a byte was received by the encoder. It also makes sure to clear the Timer module once byte_rcvd is asserted.The signal buff_empty is asserted by the FIFO module when it has transmitted out all the 8 bytes within the FIFO. after transmitting the data packet , the transmitter then asserts a signal to transmit the 16 bit CRC code and then asserts an eop signal to signal the NRZi encoder to pull the D+ and D- lines low,

If on the other hand, if the transmitter had to assert an aCK or NaCK packet , as signalled by the inputs tx_ack or tx_nack , the respective signals are asserted for the SYNC and PID bytes to be transmitted and the eop signal is asserted to signal the end of the packet.

Inputs for the USB module:

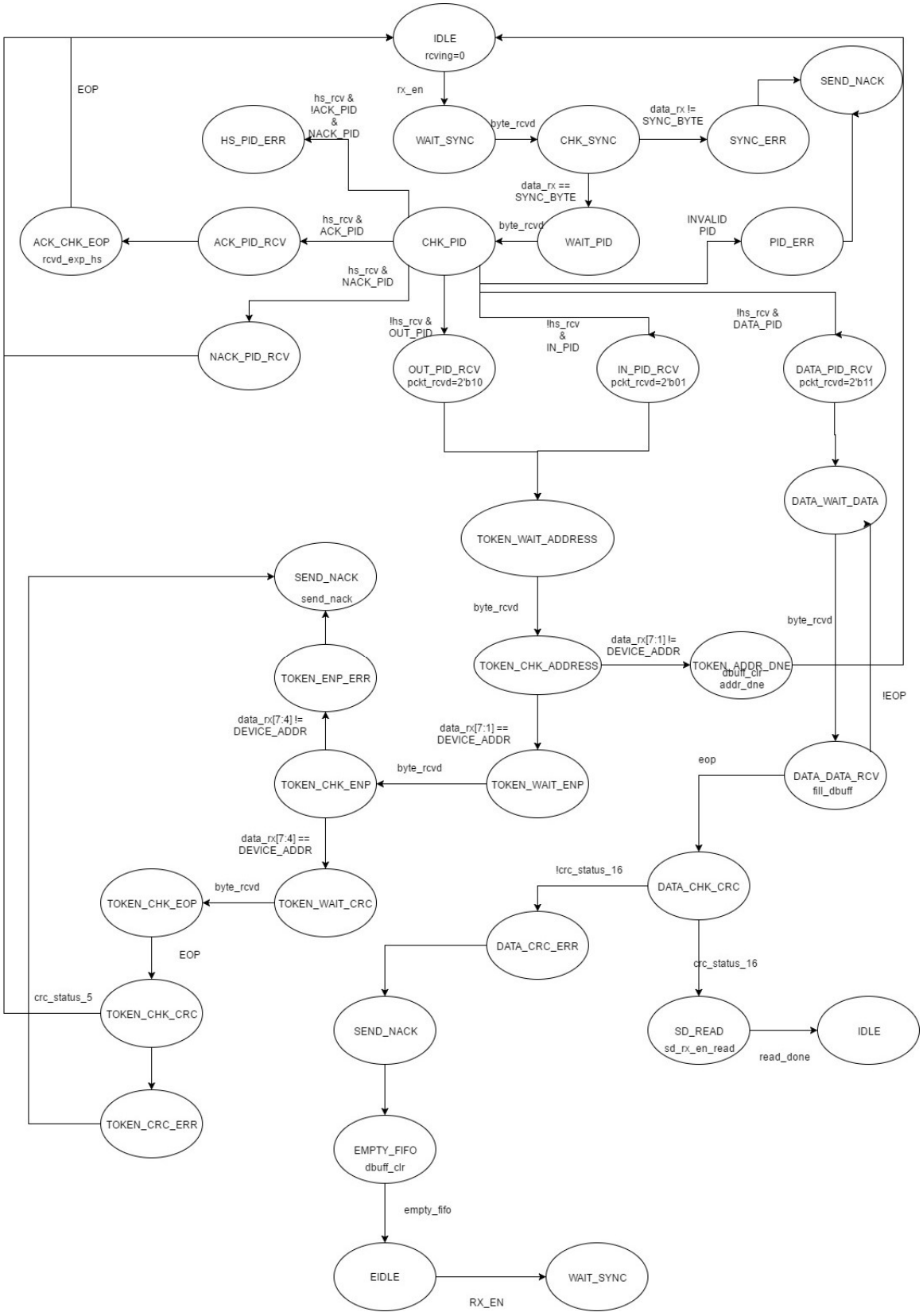
- Clk : This clock is used for transmission of data on USB bus. all transitions occurs at the positive edge of this clock.
- Nrst : This is the incoming signal. When it goes low, it resets the device transmitter. all intermediate signals take their default values.
- Send_data: This signal signals the transmitter to send a data packet by reading from the SD Card buffer and sending data back to the host.
- Tx_ena : This is the input signal when high , the device transmitter goes into transmit state and starts transmitting packets.
- Buff_empty : Input from the timer indicating that the SD Fifo is empty
- Send_data: If high then sending packet is data
- Tx_nack:This is the input signal when high , informs the transmitter to transmit a Nack packet
- Tx_ack:This is the input signal when high , informs the transmitter to transmit a ack packet
- Write_done: Input from the SD Card to let the transmitter know that the SD Card has written to the buffer

Outputs of the USB Module

- eop:Signal the NRZi encoder to transmit an EOP

- Start_read : Signals the Timer module to start counting the bit cycles and also the number of bits shifted out
- Clear_64: Clears the byte counter after the rollover value of 64 is reached.
- Trans: asserted when the receiver is active
- Sd_tx_en_write: Signals the SD Card to write into the SD Buffer as the transmitter is expecting to read data.
- Load_ena : Signals to the PTS Shift Register to load in data from the SD Buffer
- Load_sync:Signals to the PTS Shift Register to load the SYNC byte
- Load_data_pid: Signals to the PTS Shift Register to load the DaTa0 PID
- Load_data_1_crc : Signals to the PTS Shift Register to load the first byte of the CRC field of the DaTa packet CRC.
- Load_data_2_crc : Signals to the PTS Shift Register to load the second byte of the CRC field of the DaTa packet CRC.
- Load_ack: Signals to the PTS Shift Register to load the aCK PID
- Load_nack:Signals to the PTS Shift Register to load the NaCK PID

- **USB 1.1 Receiver:**


IDLE
rcving=0
EOP
rx_en
hs_rcv &
!ACK_PID
&
NACK_PID
HS_PID_ERR
WAIT_SYNC
byte_rcvd
CHK_SYNC
data_rx !=
SYNC_BYTE
SYNC_ERR
SEND_NACK
data_rx ==
SYNC_BYTE
ACK_CHK_EOP
rcvd_exp_hs
ACK_PID_RCV
hs_rcv &
ACK_PID
CHK_PID
byte_rcvd
WAIT_PID
INVALID
PID
PID_ERR
hs_rcv &
NACK_PID
!hs_rcv &
OUT_PID
!hs_rcv
&
IN_PID
!hs_rcv &
DATA_PID
NACK_PID_RCV
OUT_PID_RCV
pckt_rcvd=2'b10
IN_PID_RCV
pckt_rcvd=2'b01
DATA_PID_RCV
pckt_rcvd=2'b11
DATA_WAIT_DATA
TOKEN_WAIT_ADDRESS
SEND_NACK
send_nack
byte_rcvd
byte_rcvd
data_rx[7:1] !=
DEVICE_ADDR
TOKEN_CHK_ADDRESS
TOKEN_ADDR_DNE
dbuff_clr
addr_dne
!EOP
TOKEN_ENP_ERR
data_rx[7:4] !=
DEVICE_ADDR
data_rx[7:1] ==
DEVICE_ADDR
eop
TOKEN_CHK_ENP
byte_rcvd
TOKEN_WAIT_ENP
DATA_DATA_RCV
fill_dbuff
data_rx[7:4] ==
DEVICE_ADDR
!crc_status_16
DATA_CHK_CRC
TOKEN_CHK_EOP
byte_rcvd
TOKEN_WAIT_CRC
DATA_CRC_ERR
EOP
crc_status_16
crc_status_5
TOKEN_CHK_CRC
SEND_NACK
SD_READ
sd_rx_en_read
IDLE
read_done
TOKEN_CRC_ERR
EMPTY_FIFO
dbuff_clr
empty_fifo
EIDLE
WAIT_SYNC
RX_EN

The rx_en signals the receiver to become active. The controller first checks if the byte it first receives is a SYNC byte as is common with all the packets it receives. If it isnt a SYNC byte , it moves to a SYNC ERROR state.If it was a SYNC byte, the controller then checks the PID of the received byte. Depending on the value, it comprehends if it is a IN , OUT, DaTa ,aCK or NaCK packet. It then sets pckt_rcvd value as the respective packet value. IN as 10 , OUT as 01 DaTa as 11 , aCK as 100. However if it is comprehended as a NaCK packet , the receiver immediately asserts tx_nack and mentions to the BD Controller that it has to signal the transmitter to send a nack. after checking PID, according to the packet it detected , it then checks the device address and endpoint numbers for a TOKEN packet or data and CRC for DaTa packet . If signalled that the CRCstatus is 1 , it means that the packet specific information is good and proceeds to check if eop is 1 and then goes back into IDLE , after asserting either tx_ack if the pckt_rcvs is not 00 indicating that the packet received is wrong or is not in the ERR state otherwise it would again assert send_nack. also, the controller has multiple error states , including that of PID,SYNC,device address, ENP , DaTa suggesting that the packet received is not in the right format. also if hs_rcv is asserted, the receiver knows to expect a ack and if it received another packet instead, the controller again asserts send_nack.

Inputs :

- Clk : This clock is used for transmission of data on USB bus. all transitions occurs at the positive edge of this clock.
- Nrst : This is the incoming signal. When it goes low, it resets the device transmitter. all intermediate signals take their default values.
- Rx_en: Signal determines if the receiver is active
- EOP: Signal when asserted , means that the packet has now been completely received.
- Byte_rcvd: Signal shows that a byte has been received by the controller
- Crc_status_5: Signal denotes that the token packet specific information is correct
- crc_status_16:Signal denotes that the data within the data packet is correct
- Hs_rcv: Tells the receiver that the incoming packet should be a handshake packet
- Fifo_empty: Signal denotes that the fifo has now become empty after data has been extracted from it.
- Fifo_full: The receiver pushes information to the fifo until this signal is asserted

- Read_done: SD card sends the signal when it has finished readings from the FIFO so that the receiver moves to the next state after writing to SD Card

Outputs :

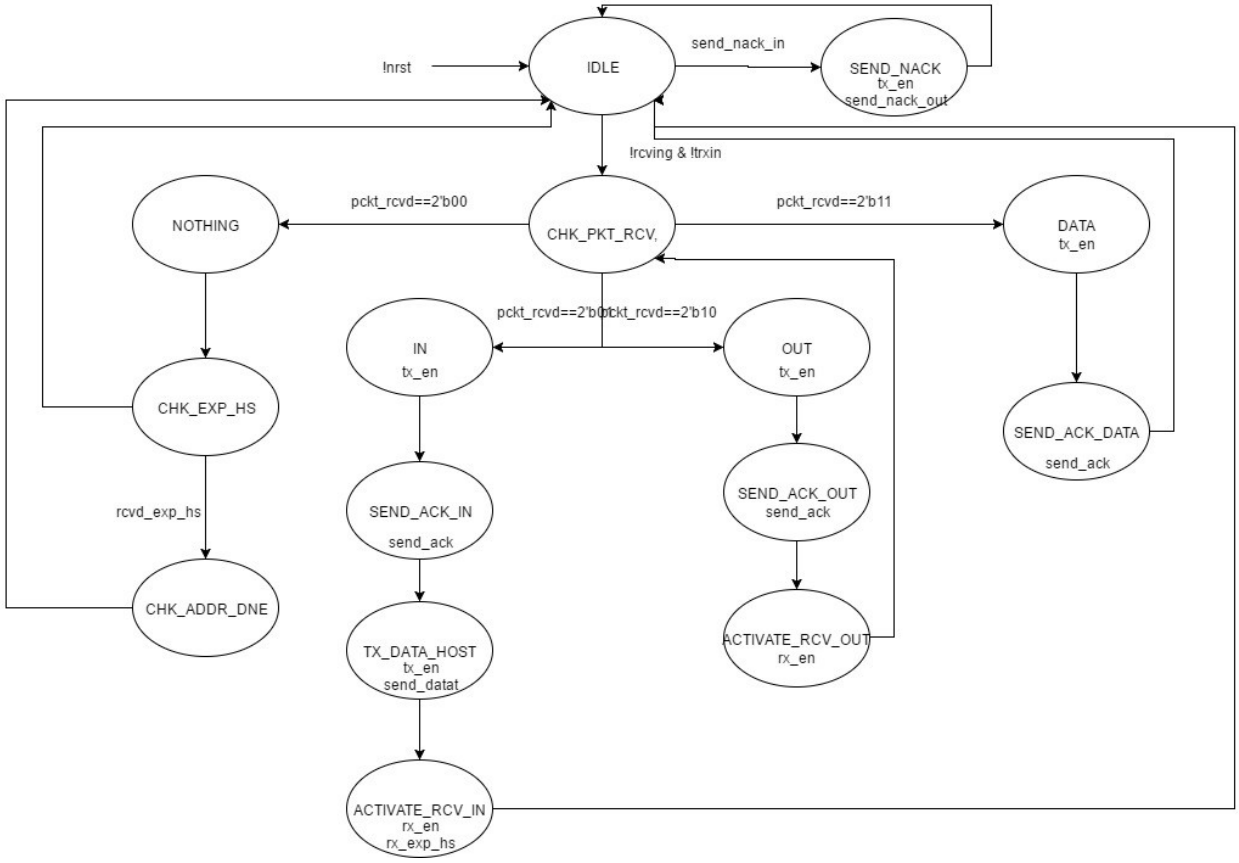
- ack_packet_rcv: Signal asserted when receiver accepts an aCK packet
- Data_rx**:** Data input to the RX Controller
- Rcving: Signal asserted by the controller when it is active.
- Dbuff_clr:Signal asserted to clear the FIFO when wrong data was received by the controller
- Rcvd_exp_hs: asserted when receiver received the expected handshake
- Send_nack: asserted to let the BD Controller know that it has to activate the transmitter to send a NaCK packet
- addr_dne: asserted when the device address received is invalid
- Sd_rx_en_read: Signal asserted to let the SD Card know that it has to start reading
- Pckt_rcvd: Value to let the BD Controller know the type of packet received

TIMING DIAGRAM

The above timing diagrams show the USB receiver. The first diagram shows the receiver receiving out, data,in ,ack and error packet.
The SYNC packet is shown above. Once byte_rcvd is asserted, the state changes from WAIT_SYNC to CHK_SYNC. If 10000000 is detected, the state then continues to check for the next field, WAIT_PID.
The following diagrams show the error checking of token packets.
The above timing diagrams show the USB receiver. The first diagram shows the receiver receiving out, data,in ,ack and error packet.
The SYNC packet is shown above. Once byte_rcvd is asserted, the state changes from WAIT_SYNC to CHK_SYNC. If 10000000 is detected, the state then continues to check for the next field, WAIT_PID.
The following diagrams show the error checking of token packets.

The WAIT PID waits for data PID to be received after byte_rcvd and then checks pid to decide if it’s a data packet and if it is , it moves to the DATA_PID_RCV.Once it receives the PID, it checks the data field until eop is asserted and then checks crc. If everything is correct, it then asserts fill dbuff and populates the fifo

- **BD Controller:**



The controller to control both the USB Receiver and USB Transmitter. This is the only mode of communication between the receiver and transmitter. When the rx_ena and tx_ena is off suggesting that both the transmitter and receiver are inactive. The controller decides which packet has to be transmitted when it receives the type of packet received from the USB receiver.

Inputs to the BD Controller:

1. N_rst: This is the incoming signal. When it goes low, it resets the device transmitter. all intermediate signals take their default values.
2. Rcving: Signal to denote that the receiver is active
3. Txing: Signal to denote that the transmitter is active
4. Rcvd_exp_hs: Signal asserted when the receiver detected the incoming handshake packet
5. Send_nack_in: Signal to denote that the IN Packet was not received properly
6. addr_dne: Signal to denote that the token packet has not been received properly
7. Tx_complete: Signal to denote that the transmission of the intended packet is complete
8. Pckt_rcvd: 3 bit value to denote the type of packet received

Outputs to the BD Controller:

9. Rx_en: Signal to activate receiver
10. Tx_en: Signal to activate transmitter
11. Send_data: asserts signal to ask the transmitter to send a data packet.
12. Send_ack: Signal the transmitter to send an aCK packet
13. Send_nack_out :Signal the transmitter to send a NaCK packet in response to a faulty OUT packet
14. Rx_exp_hs: Signal to let the transmitter know to expect a handshake packet
15. ack_packet_rcv: Signal to assert that the receiver has to receive an aCK packet

In the above timing diagram, when rcving the BD controller is shown to be inactive. When rcving goes low , the BD Controller becomes active and detects that a packet has been recieved and that it was an IN packet and therefore asserts send_ack to make the transmitter send an ack in response to the packet received.

References Cited:

"Universal Serial Bus Specification". N.p., 2017. Web. 10 Mar. 2017.

Signal Name	PID	Signal Data Format	Description
Token	Out IN	SYNC PID : 8 bits aDDR : 7 bits ENDP : 4 bits CRC5 : 5 bits EOP	First packet. Used to identify the purpose of the packet from setup, IN and OUT. Identifies the endpoint of the data.
Data	Dataa0	SYNC PID : 8 bits DaTa CRC16 : 16 bits EOP	Used to transfer DaTa over the different data lines.
Handshake	aCK NaK	SYNC PID:8 bits EOP	Determines the status of the device.

Table 2: Signal Description

2. SD Card Interface

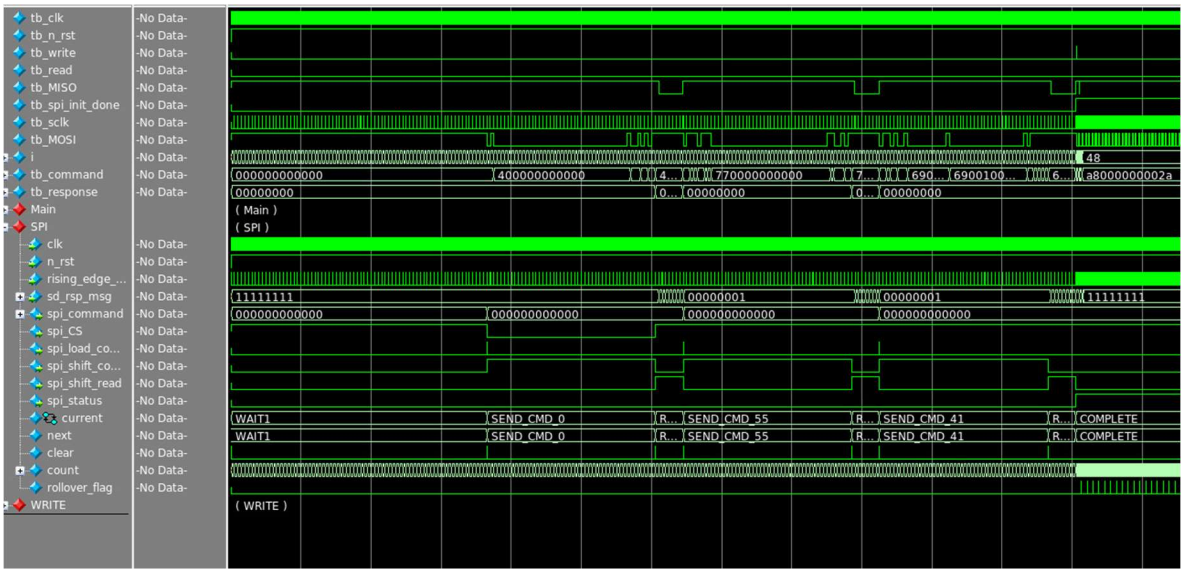


Figure (1): Timing Diagram for SPI initialization

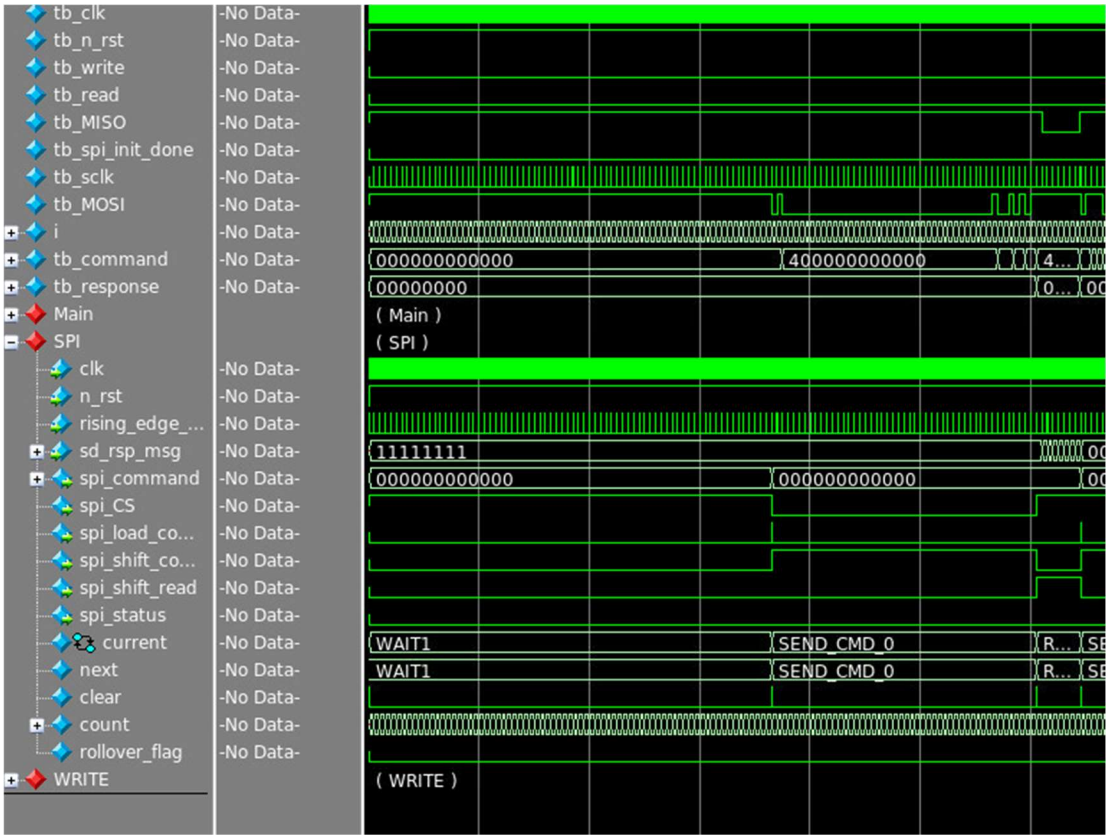


Figure (2): Timing Diagram Send Command Zero

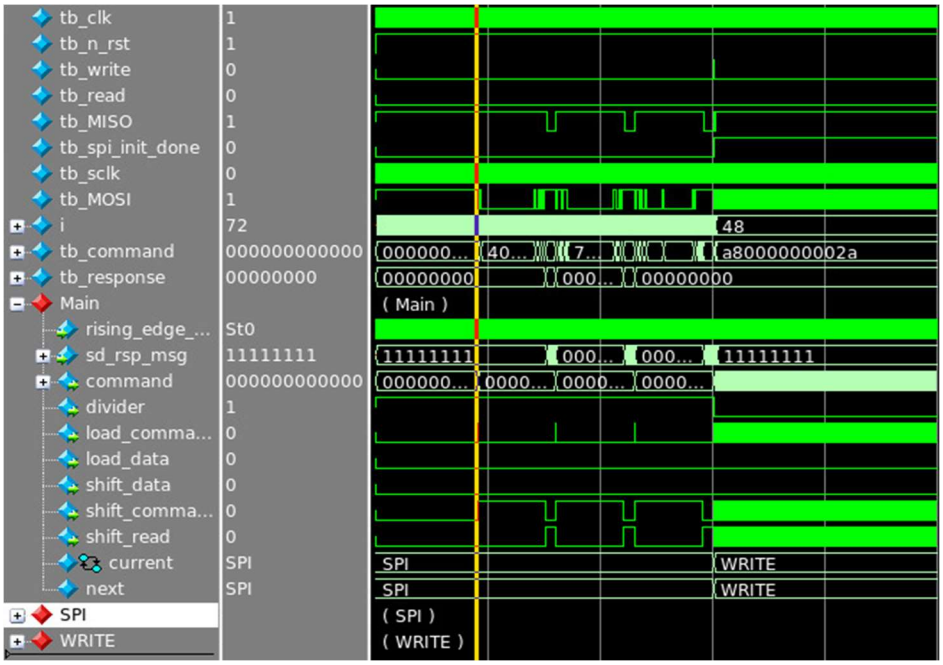


Figure (3) : Change in State of the Main Controller

The operation of the sd card depends on the state of the main controller. The state determines whether the sd is initialised to its spi mode, data is written to or read from the sd card or whether it is idle. Once the state has been determined not to be idle the sub controllers spi initialization, write and read are used to complete the sequence based on the state.

The spi initialize is called upon the initial powering up of the system and will not be called again unless the system is reset. The write is in control of writing a single block of data which is of size 8 bytes to the sd card while the read is in control of reading a single block of data of size 8 bytes from the sd card.

In Figure (1), we can see the waveform for the completed spi initialization sequence. It starts when the reset signal is asserted on startup. The sclk is set to 400 Khz while the system clock runs at 96Mhz.It then has a wait state where it needs to wait for 74 sclk cycles at 400 Khz which can be seen in the diagram where while the system clock has run several cycles over due to the faster clock rate the interface wait for 74 falling edges before moving on to the next state. It also shows the process of sending a command and upon receiving the appropriate response proceeding to the next one until the sequence is complete.

In Figure (2), we can see the interface send command zero to the SD card. The transmission of the command can be observed on the MOSI line. The response of the command can be viewed in Figure (1), where the MISO valued is 0 for 7 cycles followed by 1 for a single cycle which is the correct response.

In Figure (3), we can see the initial state of the main controller to be SPI and the moment it it complete the assertion of the spi_init_done signal.The assertion of write signal input then causes the state to change to the write state starting a new sequence which is handled by the write controller. In Figure (1), it can also be seen the immediate increase in sclk to 24Mhz when the initialization is complete.

The diagrams show us the the driving of the sd card on the sclk and the transmission of data to and from the SD card on the MOSI and MISO lines respectively. It all shows how the sub-controllers are controlled by the main FSM.

Requirements for Design:

When implementing a USB to SD card transfer module there are a number of design choices that we have to consider.

There are several constraints that we need to consider for this project. USB 1.1 devices operate at either low speed (1.5 Mb/s) or full speed(12 Mb/s). Since the bulk data transfer operates at 12 Mb/s we chose to minimize the area as the speed of the design is already fixed.

Since our system clock (96 MHz) runs at 8 times the speed of the USB input (12 MHz), the data must only be shifted once for every 8 system clock cycles. It is important to capture the data as near as possible to the middle of the incoming bit. Another design constraint that is introduced is the speed of the SD card.To write into a SD Card , its speed is 96 Mb/s. The chosen SD card will typically operate at 400 KHz.

To minimize difference in data transfer rates between modules , data buffers are introduced in the design. In the preliminary design we have decided to introduce a 8 byte buffer.

The clock that will be required for the data transfer will initially operate at 96 MHz. a clock divider will be used to divide to clock to 12 MHz, in order to ease the interfacing. after synthesizing the chip our size of the chip came to be 1.86 mm x 1.86 mm.

The table below summarizes the budget area analysis.

The two largest blocks in the design are CRC16 generation and CRC 16 checker blocks. The main reason the CRC generation block is large because the output of the block is a 16 bit register. Same goes for the CRC 16 checker. From the table it can be clearly seen that the estimated area for the USB is 3.16 mm^2.

Design Implementation

1) Design architecture:

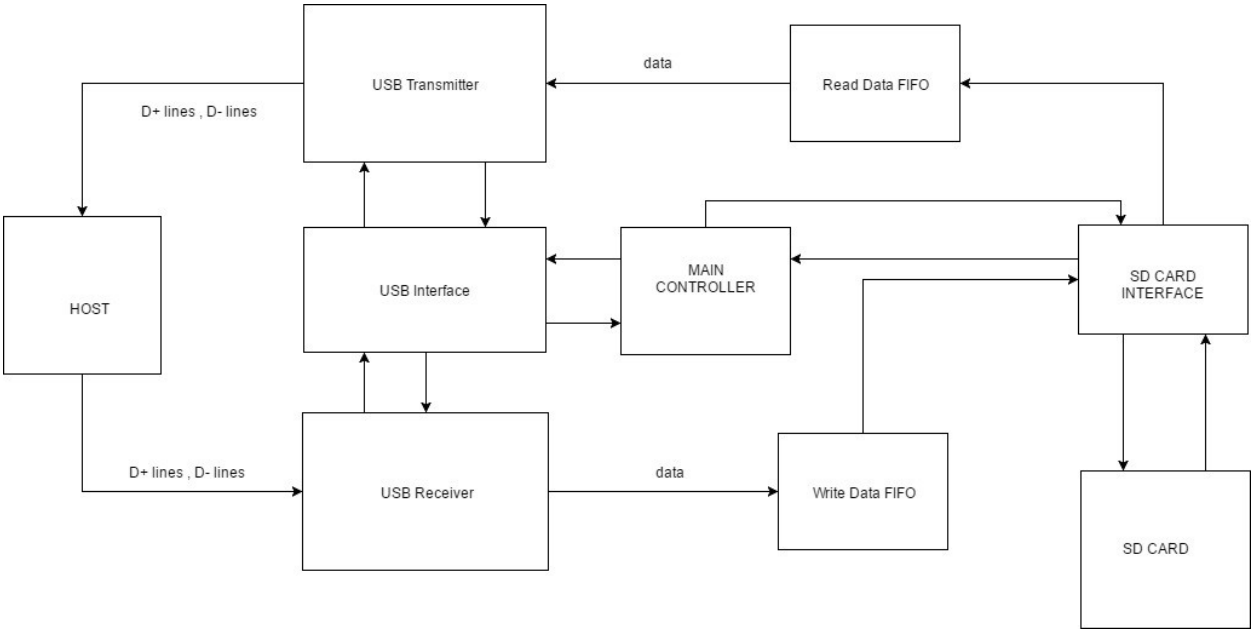


Figure 7: Encryptor-Decryptor with SD card storage architecture Diagram

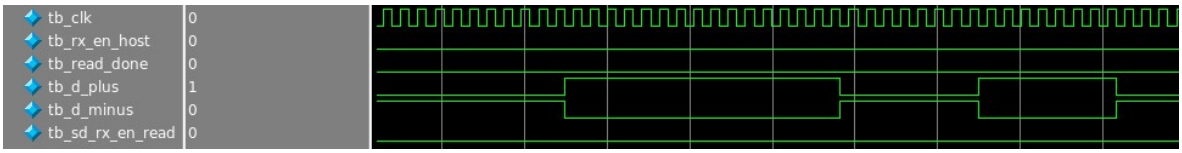
- **Working:**

The figure above illustrates the system architecture. There are 5 major blocks in the system that will be implemented.

Outputs of the Host:

1. Dplus
2. Dminus
3. Clk
4. Nrst
5. Rx_en_host

The host sends D+ , D- lines to the USB Receiver to indicate the type of Bulk Data sequence to follow. The clk is the system clock which runs at 96Mhz. The Rx_en_host is asserted by the host to activate the receiver to accept the incoming packet. It is always only asserted at the beginning of the bulk data transfer sequence.



Once the USB receiver comprehends the incoming packet as either Token or data packet , the USB interface decides the next step in the sequence as either sending a handshake or a data packet. If any of the steps include sending or receiving a data packet, the SD Card interface is enabled by the USB interface to read from or write to the external data FIFOs.

Inputs to the host:

1. Dplus
2. Dminus

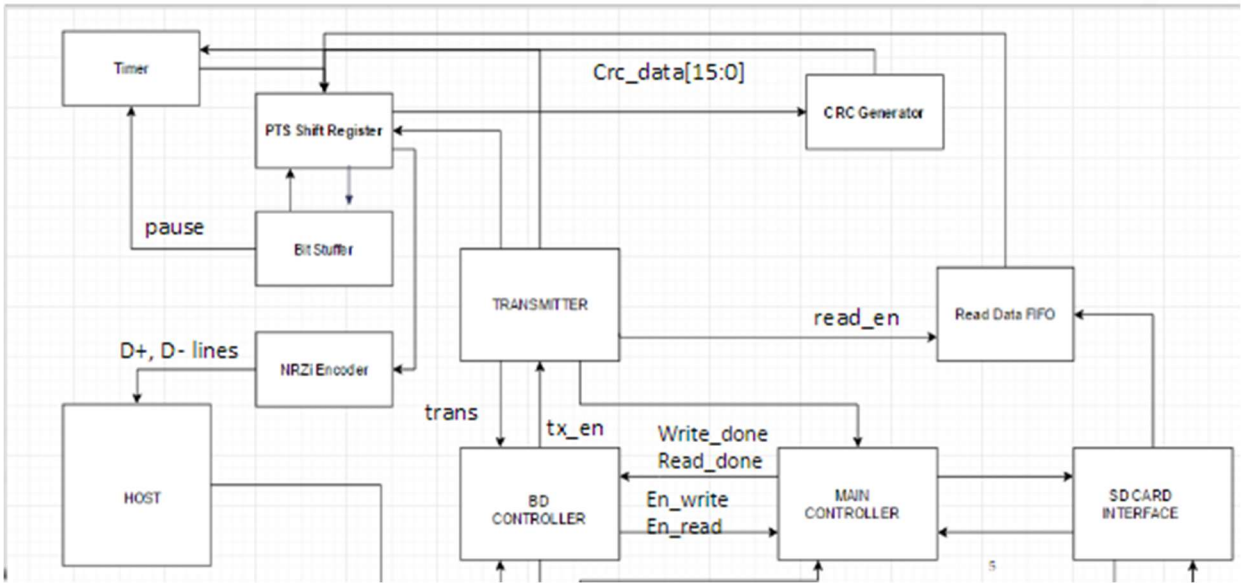
after the USB interface has received a signal to either send data or a handshake signal , the USB transmitter gets activated and outputs D+ and D- lines back to the host containing the appropriate packet information.

- **Area Estimation Table:**

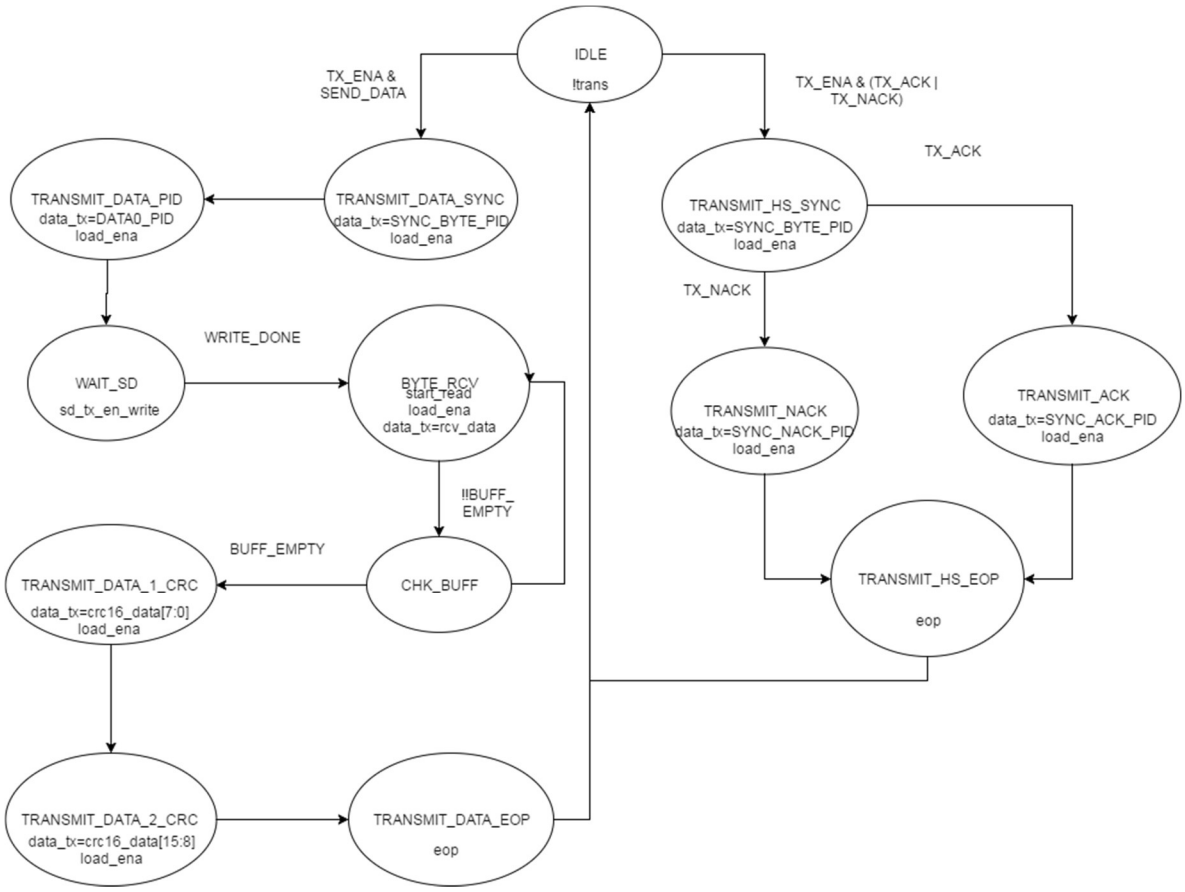
Name of Block	Area
Transmitter State Machine	20,600
Transmitter NRZi Encoder	53,600
Transmitter Bit Stuffer	3,000
Receiver Destuffer	14,400
Receiver Decode Block	9,200
CRC 16 Generator	68,200
CRC 5 Checker	24,000
CRC 16 Checker	55,200
Receiver Timer	16,000
Receiver RCU	19,600
Transmitter Timer	10000
PTS Shift Register	43600
STP Shift Register	31600
Total Core Area	**369,000**
Chip Area Calculations (units in um^2)	

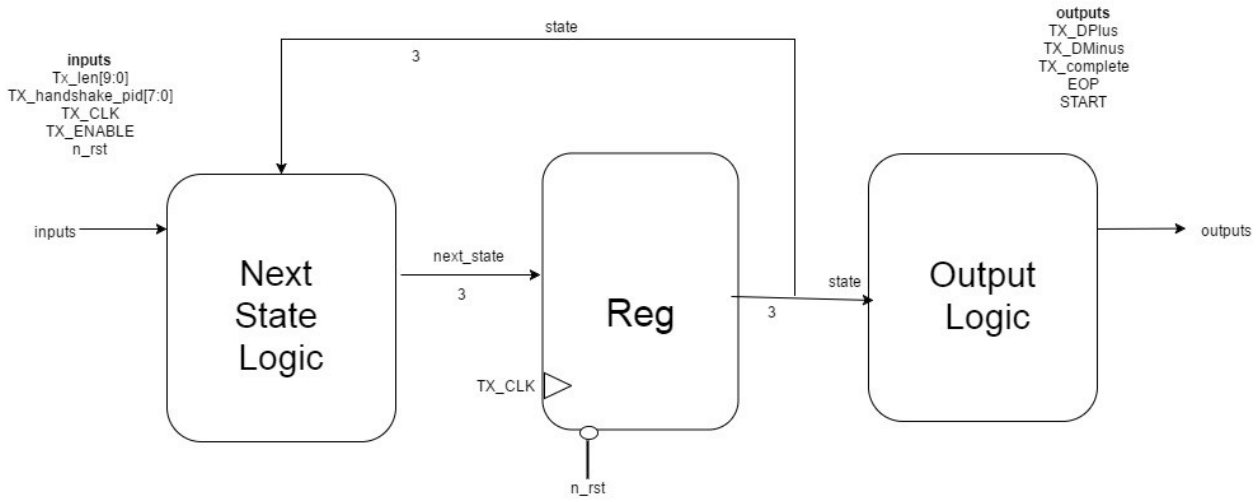
Functional Block Diagrams:

1) USB (Universal Serial Bus) 1.1 Transmitter



- Transmitter State Machine:





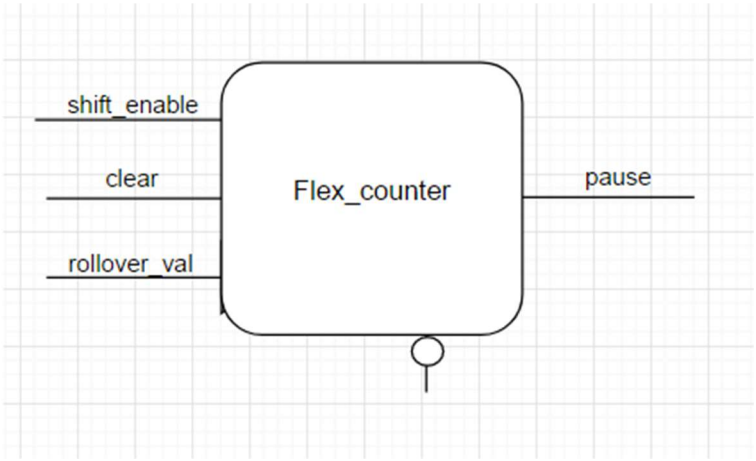
The TCU(Transmitter Control Unit) is in charge of the operation of the entire transmitter.There are 6 states overall that determine the assertion of different signals including EOP and STaRT.It is a state

machine which determines the next state logic and thereby determines and asserts signals to carry forward the operation of the USM transmitter. On reset , it also resets the entire module to its default values.

Transmitter State Machine: Next State Logic	Combinational	6	6,000	
Transmitter State Machine: Register	Register	3	9,600	Assume a max of 8 states
Transmitter State Machine: Output Logic	Combinational	5	5,000	

The total area for the transmitter state machine came out to be 0.0206 mm^2. This is because there are a 2 combinational blocks (6 bit inputs and 5 bit outputs) and 1 register block.

- **Bit Stuffer:**



In order to ensure adequate signal transitions, bit stuffing is employed by the USB transmitter while sending packets back to the host. A zero is inserted after every six consecutive ones in the data stream before the data is NRZI encoded, to force a transition in the NRZI data stream. This block adds the stuff bit(0) if six consecutive ones are found. It asserts a pause signal to stop the counter from counting up and alerts the Encoder to output a 0 bit.

Inputs:

1. Clk
2. N_rst
3. D_orig
4. Flag_8: bit cycle flag

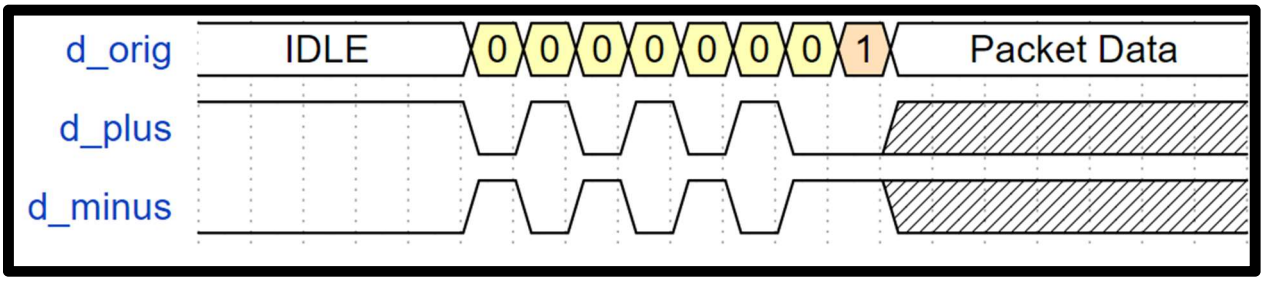
Outputs:

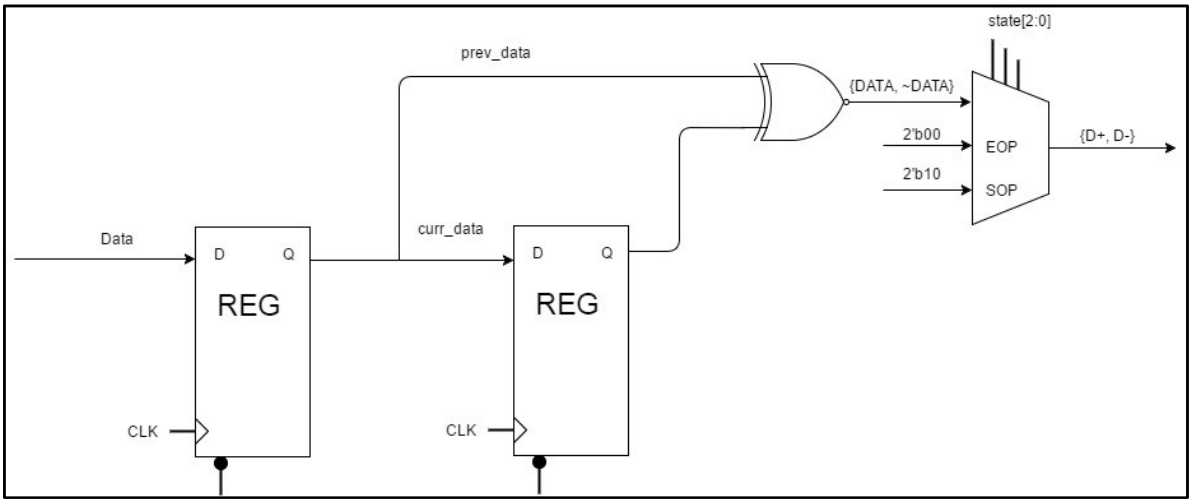
1. Pause: Signal to pause the timer and alert NRZi encoder to output 0

Transmitter Bit Stuffer: Flex Counter	Combinational	3	3,000	

The area for the bit stuffer is 0.003 mm^2. The area for this extremely small because this module instantiates the flex counter module, with a 3 bit count value.

- **NRZi Encoder**





The USB employs NRZI data encoding for transmission of packets. In NRZI encoding, logic 1 is represented by no change in level and logic 0 is represented by a change in level. The input to the block is d_orig, the non-encoded data from the bits stuffer and the flag signifying bit cycles . When flag_8 signal is asserted by the timer suggesting that a new bit is available to compare (each bit arrives at 8 system clock cycles), the NRZI encoder compares the stored value and the current to detect the output value. When an eop signal is asserted the block knows to pull the D+ and D- minus line slow to denote an end of packet.

Inputs:

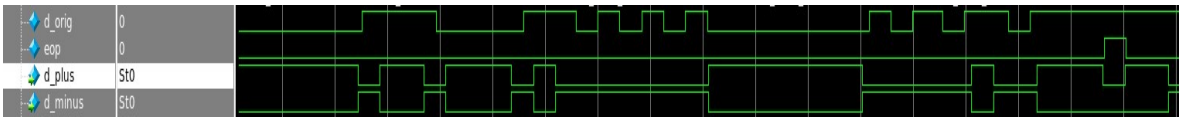
1. clk
2. N_rst
3. Pause
4. D_orig
5. Flag_8
6. eop,

Outputs:

1. d_plus
2. d_minus

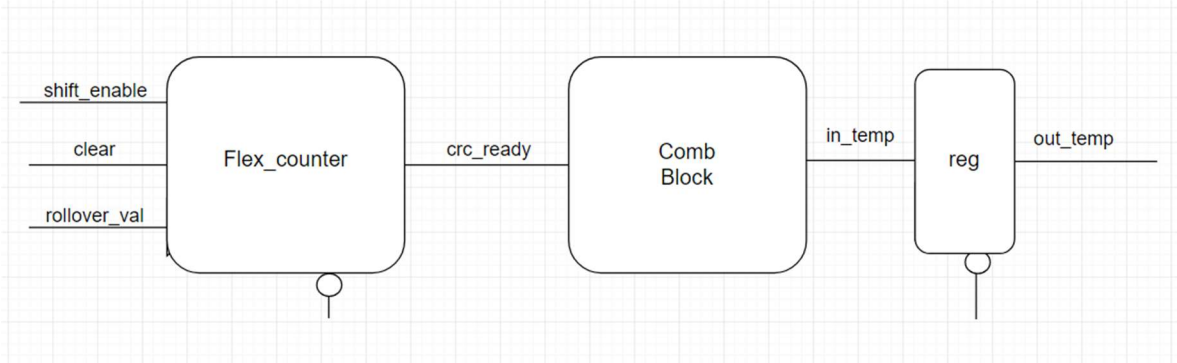
Transmitter NRZi Encoder: Reg 1	Register	8	8,000	
Transmitter NRZi Encoder: Reg 2	Register	8	25,600	
Transmitter NRZi Encoder: XOR Gate	Combinational	8	8,000	
Transmitter NRZi Encoder: Mux	Combinational	12	12,000	2 bit wide, 3 way mux (6 gates /

The are for the encoder is 0.053 mm^2. The two registers had 8 bits inputs, as did the input to the XOR gate.



When there is a change in d orig , the d plus lines are pulled high else pulled low.If an eop is detected , the Dplus and Dminus lines are pulled low.

- **CRC Generator**



The USB uses the Cyclic Redundancy Checksums (CRC) to protect all non-PID fields in token and data packets from errors during transmission. This block will generate these crc bits and send them to the PTS Shift Register to transmit to the host.. The input to this block is the data and CRC fields of the data packet. Since the CRC bits are required only for token packets , the generator is required to generate only 16 bit CRC values.

Inputs :

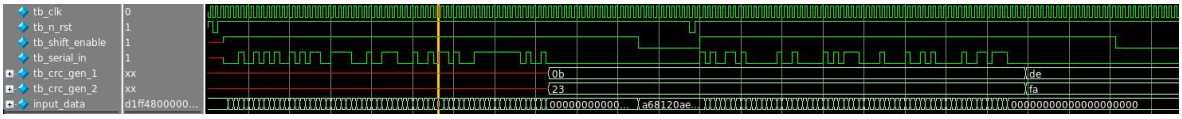
1. Clk
2. N_rst
3. Shift_enable
4. Serial_in
5. clear

Outputs:

1. Crc_gen_1
2. crc_gen_2

CRC 16 Generator: Flex Counter	Combinational	7	7,000	
CRC 16 Generator: Mux 1	Combinational	6	6,000	
CRC 16 Generator: Mux 2	Combinational	2	2,000	
CRC 16 Generator: Mux 3	Combinational	2	2,000	
CRC 16 Generator: Reg	Register	16	51,200	

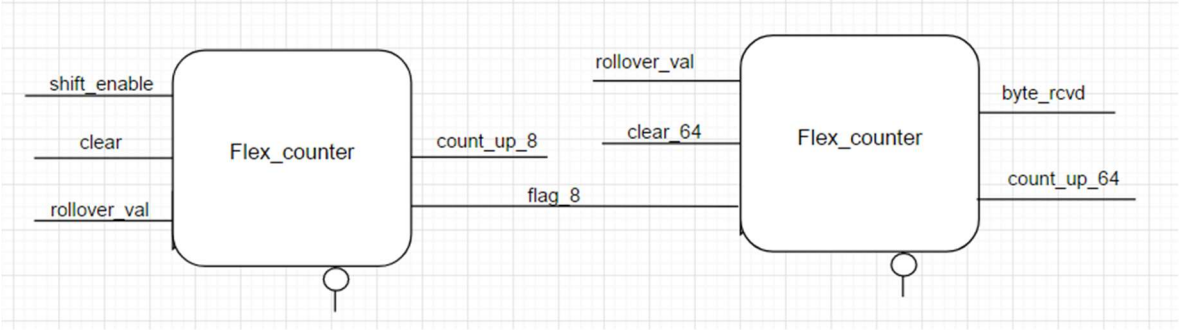
The total area for this block is 0.0682 mm^2. The flex counter is called with a 7 bit count out value. There are a total of 3 mux's in the block. The first is a 3 way mux with 2 bit input. The other two are are a 2 way mux with 1 bit input. The register has a 16 bit output.



In the above timing diagram, the CRC code is generated after the 63 cycle

- **Timer_TX**

The timer_tx is responsible for asserting flags to signify a bit period cycle which counts every 8 system cycles to perform operations on bits. Flag_8 denotes the bit cycle flag.To denote a byte has been received , byte_rcvd is asserted and is used by different modules.



Inputs

- Clk
- N_rst
- Pause
- Count_up
- clear_64

Outputs:

- Shift_en
- Byte_rcvd
- flag_8

Transmitter Timer: Flex Counter 1	Combinational	4	4000	
Transmitter Timer: Flex Counter 2	Combinational	4	4,000	
Transmitter Timer: Mux	Combinational	2	2,000	

The area for this block is 0.01 mm^2. There are 2 flex counters in this modles with a 4 bit count value. Additionally, there is also a 2 way mux with a 1 bit output.

- Read Data FIFO:

This external FIFO stores data from the SD Card and is read from by the USB transmitter when the host sends an IN packet. The transmitter controller asserts the read_enable signal from the FIFO until the fifo_empty signal is asserted to read 8 bytes of data stored in the FIFO. The SD Card asserts the write_enable signal when it is signalled by the USB to write into the FIFO , by sending it write data which is a byte at a time.
Inputs:

1. Clk
2. N_rst
3. Read_enable
4. Write_enable
5. write_data[7:0]

Outputs:

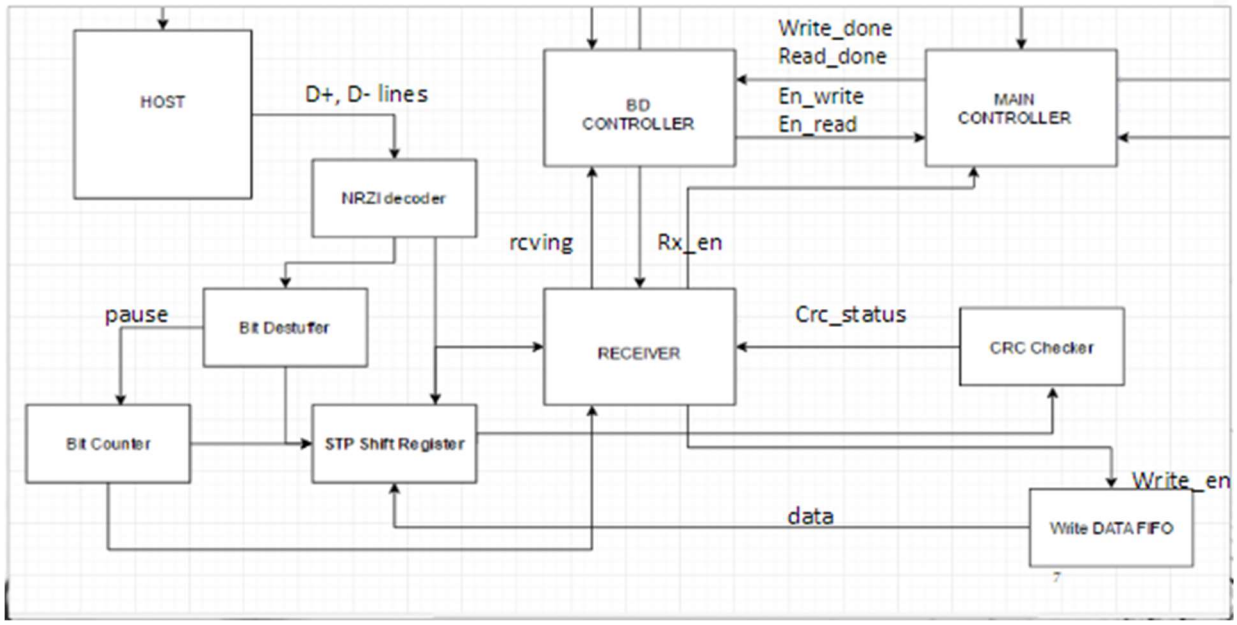
1. Read_data
2. output reg
3. Fifo_empty
4. fifo_full

PTS Shift Resgiter: Mux 1	Combinational	6	6000	
PTS Shift Resgiter: Mux 2	Combinational	4	4000	
PTS Shift Resgiter: Mux 3	Combinational	2	2000	
PTS Shift Resgiter: Reg	Register	8	25600	
PTS Shift Resgiter: Mux 4	Combinational	4	4000	
PTS Shift Resgiter: Mux 5	Combinational	2	2000	

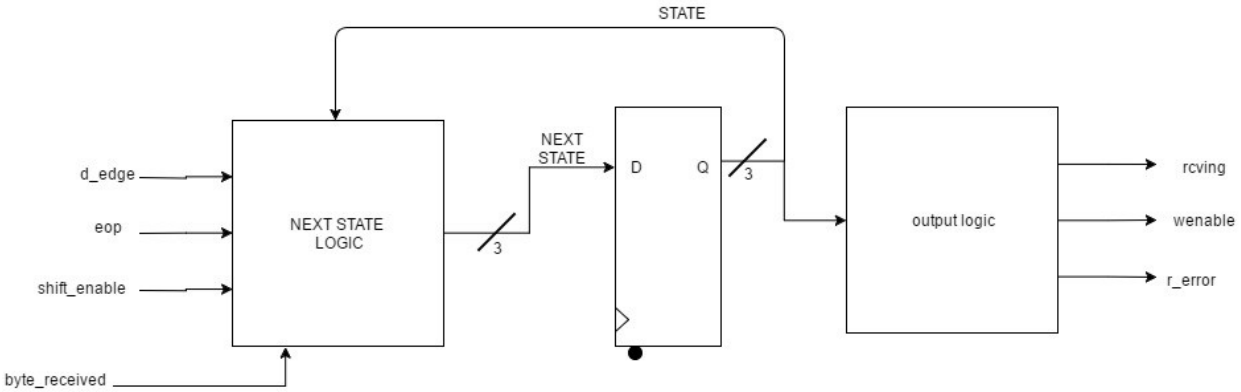
The total area for this block is 0.0436 mm^2. There a total a mux's in the block. The first is a 3 way mux with a 2 bit select. Mux 2 and 4 are a 2 way mux with a 2 bit output. Mux 3 and 5 are a 2 way mux with a 1 way out. Lastly, there is also a 8 bit register.

2) **USB (Universal Serial Bus) 1.1 Receiver**

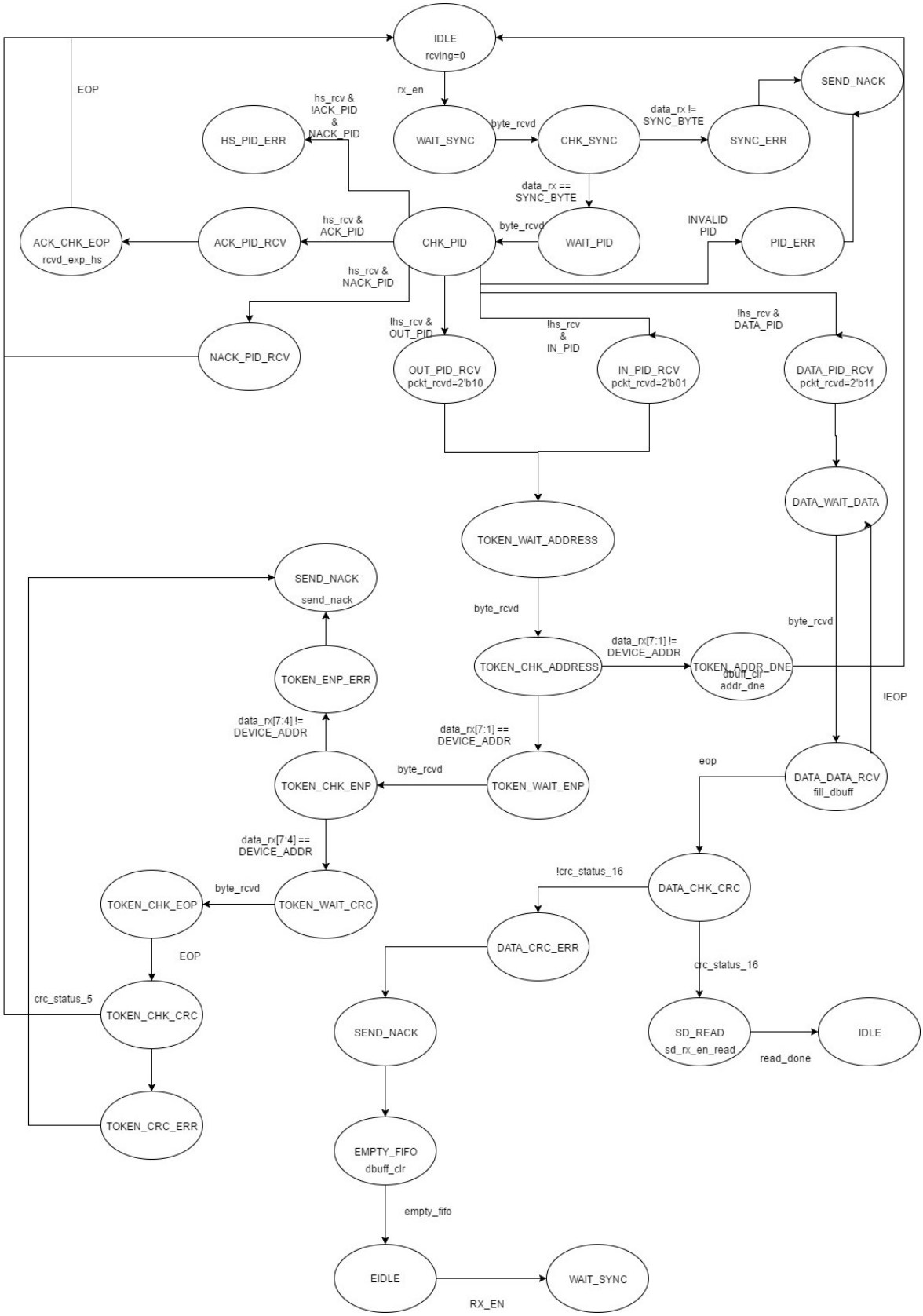
USB Receiver Functional block:



- RCU



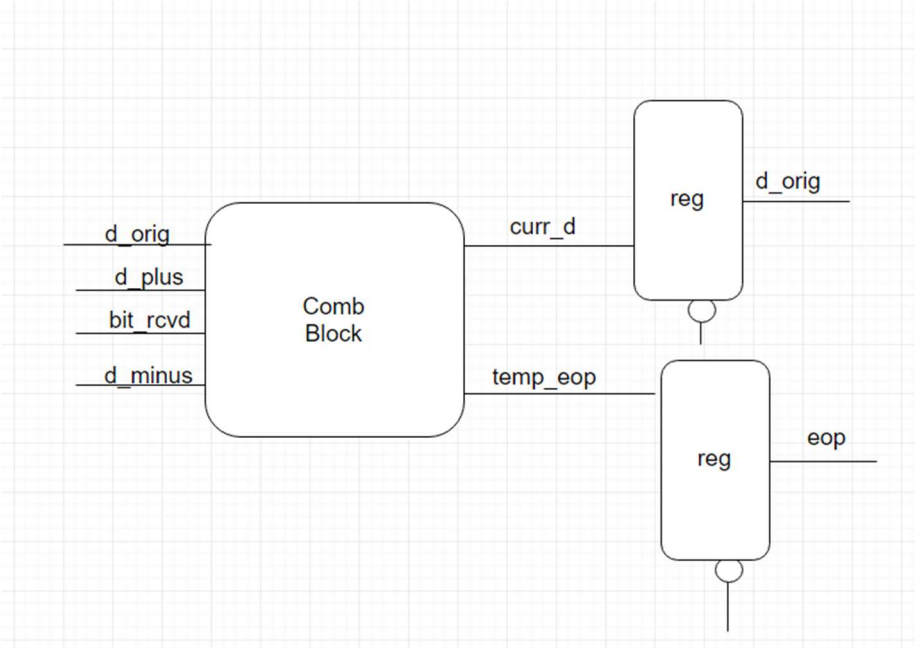
The RCU(Receiver Control Unit) is in charge of the operation of the entire receiver.There are 8 states overall that determine the assertion of different signals.It is a state machine which determines the next state logic and thereby determines and asserts signals to carry forward the operation of the USB receiver. On reset , it also resets the entire module to its default values.


IDLE
rcving=0
EOP
rx_en
hs_rcv &
!ACK_PID
&
NACK_PID
HS_PID_ERR
WAIT_SYNC
byte_rcvd
CHK_SYNC
data_rx !=
SYNC_BYTE
SYNC_ERR
SEND_NACK
data_rx ==
SYNC_BYTE
ACK_CHK_EOP
rcvd_exp_hs
ACK_PID_RCV
hs_rcv &
ACK_PID
CHK_PID
byte_rcvd
WAIT_PID
INVALID
PID
PID_ERR
hs_rcv &
NACK_PID
!hs_rcv &
OUT_PID
!hs_rcv
&
IN_PID
!hs_rcv &
DATA_PID
NACK_PID_RCV
OUT_PID_RCV
pckt_rcvd=2'b10
IN_PID_RCV
pckt_rcvd=2'b01
DATA_PID_RCV
pckt_rcvd=2'b11
DATA_WAIT_DATA
TOKEN_WAIT_ADDRESS
SEND_NACK
send_nack
byte_rcvd
byte_rcvd
data_rx[7:1] !=
DEVICE_ADDR
TOKEN_CHK_ADDRESS
TOKEN_ADDR_DNE
dbuff_clr
addr_dne
!EOP
TOKEN_ENP_ERR
data_rx[7:4] !=
DEVICE_ADDR
data_rx[7:1] ==
DEVICE_ADDR
TOKEN_CHK_ENP
byte_rcvd
TOKEN_WAIT_ENP
eop
DATA_DATA_RCV
fill_dbuff
data_rx[7:4] ==
DEVICE_ADDR
!crc_status_16
DATA_CHK_CRC
TOKEN_CHK_EOP
byte_rcvd
TOKEN_WAIT_CRC
DATA_CRC_ERR
EOP
crc_status_16
crc_status_5
TOKEN_CHK_CRC
SEND_NACK
SD_READ
sd_rx_en_read
IDLE
read_done
TOKEN_CRC_ERR
EMPTY_FIFO
dbuff_clr
empty_fifo
EIDLE
WAIT_SYNC
RX_EN

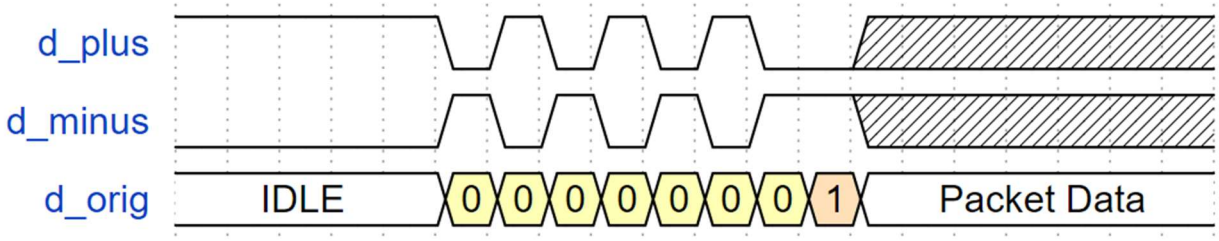
Receiver RCU: Next State Logic	Combinational	6	6,000
Receiver RCU: Reg	Register	3	9,600
Receiver RCU: Output Logic	Combinational	4	4,000

The total area for this block is 0.0196 mm^2. There a two combinational blocks (6 input bits and 4 output bits). Additionally, there is also a register to store a 3 bit state.

- NRZi Decoder.



Since the USB receives data which is NRZi encoded, to perform operations on the data, it first has to be decoded. In NRZI encoding, logic 1 is represented by no change in level and logic 0 is represented by a change in level. The input from the host is D+ and D- and when the bit_rcvd is asserted, the current and stored value are computed to output the decoded bits. When the module detects that both the D+ and D- lines are pulled low, it asserts the eop signal to indicate the end of packet.



Inputs:

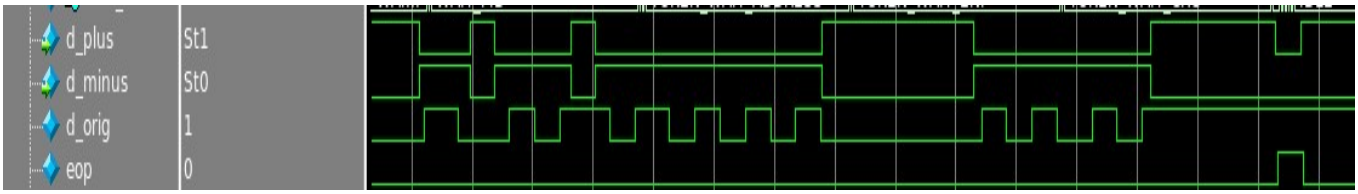
1. Clk
2. N_rst
3. D_plus
4. D_minus
5. bit_rcvd,

Outputs:

1. D_orig
2. eop

Receiver Decode Block: Mux 1	Combinational	2	2,000
Receiver Decode Block: Mux 2	Combinational	2	2,000
Receiver Decode Block: Reg	Register	1	3,200
Receiver Decode Block: NAND Gate	Combinational	1	1,000
Receiver Decode Block: XNOR Gate	Combinational	1	1,000

The total area for this block is 0.0092 mm^2. There are 2 mux's in this block. Both are a 2 2 way mux and 1 bit wide. Additionally there are 2 combination gates which are 1 bit wide. Lastly, there is also a 1 bit wide register to store the d_orig value.



The above timing waveform shows that when eop is asserted the Dplus and Dminus lines are pulled low . When there is a 1 in the dplus , there is transition in the d orig and if a 1 is detected , the dorig remains same as shown above.

- Timer_rx

The purpose of the block is to generate the byte_rcvd signal, which will allow the shift registers to shift in the next byte of data. The data must only be transmitted once for every 8 clk cycles, since the system clk is 96MHz and USB runs at 12MHz. The input is the pause signal which pauses the timer when the bit destuffer is active.

Inputs:

1. Clk,
2. N_rst
3. Cnt_up
4. Clear
5. Pause
6. d_plus

Outputs:

1. Byte_rcvd
2. bit_rcvd

Receiver Timer: Flex Counter 1	Combinational	8	8,000
Receiver Timer: Flex Counter 2	Combinational	8	8,000

The total area for this block is 0.016 mm^2. This block instantiates the flex counter module which requires 8 bits per instantiation.

- **Bit Destuffer:**

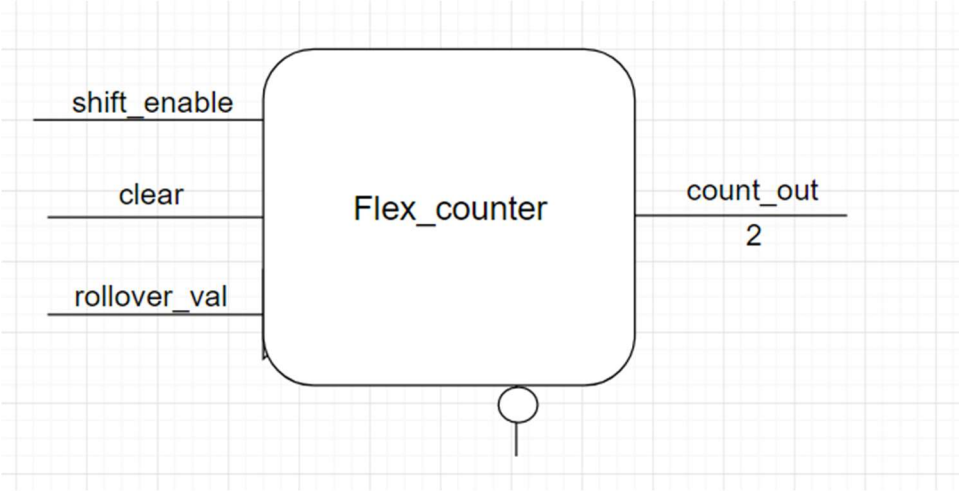
In order to ensure adequate signal transitions, bit stuffing is employed by the transmitting device when sending a packet on USB. A zero is inserted after every six consecutive ones in the data stream before the data is NRZI encoded, to force a transition in the NRZI data stream. This block detects whether the seventh (stuffed) bit is 0 after detecting 6 1's in the NRZi decoded data and doesn't allow that 0 to enter the shift register. The input to this block is the bit stream and the output is a signal which goes low when there are six consecutive ones.

Inputs:

1. Clk
2. N_rst
3. Bit_rcvd
4. d_orig,

Outputs:

1. pause



Receiver Destuffer: Reg 1	Register	2	6,400
Receiver Destuffer: Mux 1	Combinational	4	4,000
Receiver Destufferr: Mux 2	Combinational	2	2,000
Receiver Destuffer: Mux 3	Combinational	2	2,000

The area for this block is 0.0144 mm^2. The block contains 3 mux's of which 2 are 2 way, 1 bit wide and the other is a 2 bit wide 2 way mux. Moreover there is also a 2 bit register.

- **STP Shift Register:**

This is the 8-bit shift register. The input to this block is a shift enable from the timer which is asserted at every bit cycle to shift in a bit from the fifo and also pause signal suggests that the bit destuffer has detected a 0 after 6 1's so the STP should not be shifting a bit.
If the signal is high then at the positive edge it shifts the bit stream to right.

Inputs:
1. clk
2. N_rst
3. Shift_enable
4. Pause
5. serial_in

Outputs:
1. parallel_out

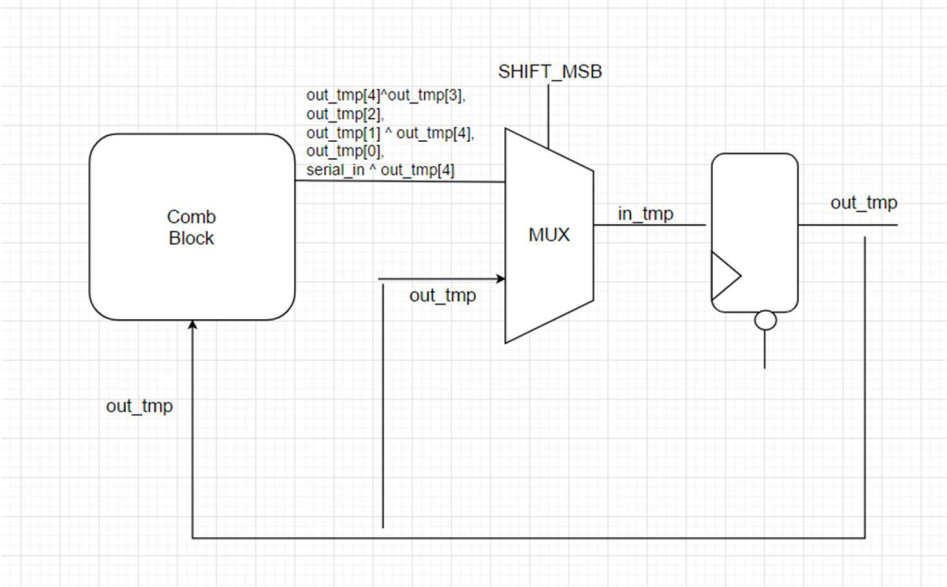
STP Shift Register: Mux 1	Combinational	4	4000
STP Shift Register: Mux 2	Combinational	2	2000
STP Shift Register: Reg	Register	8	25600

The total area for this block is 0.0316. The block contains one 2 way 2 bit wide mux and one 2 way 1 bit wide mux. Also, there is a register to store an 8 bit value.

- **CRC Checkers:**

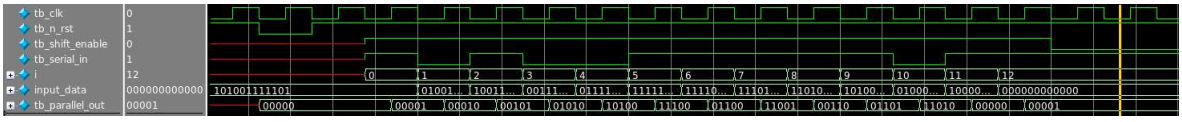
The CRC checker is used to detect if the received packet contains the right data.This is a shift register which accepts bits and passes through This is used with the token and data packets and is passed through the following equations for 5 bit CRC for token and 16 bit checker for data packets.

CRC 5 Checker



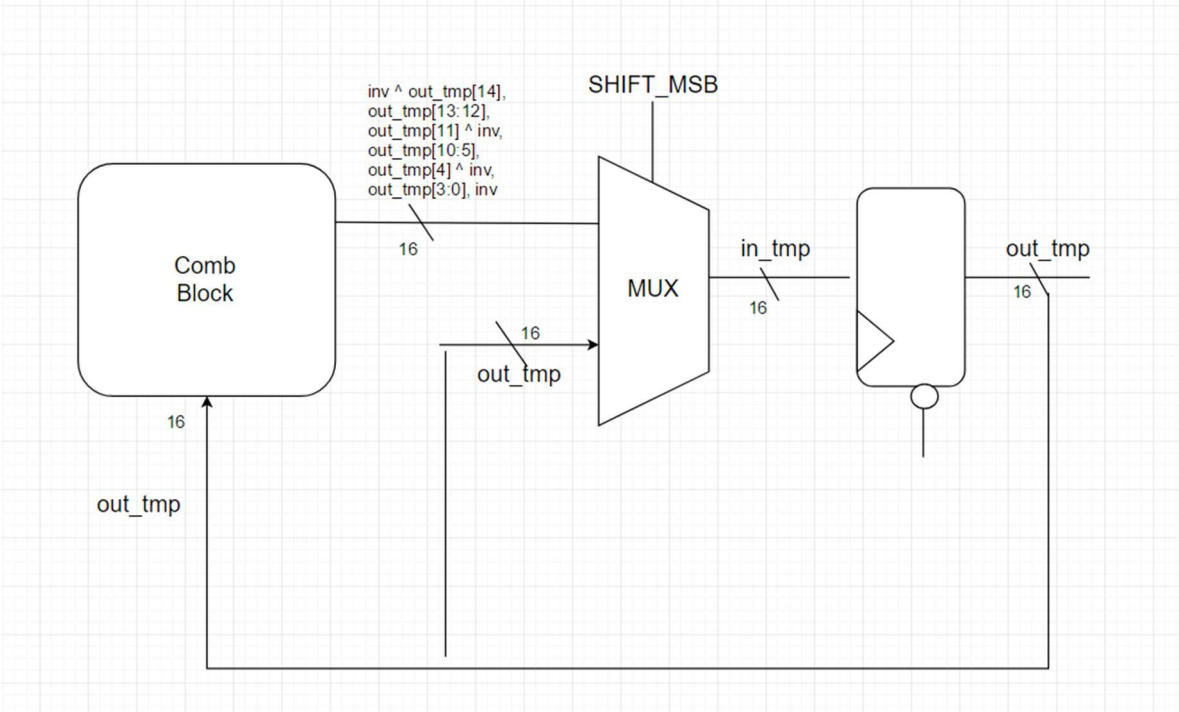
CRC 5 Checker: Mux 1	Combinational	6	6,000
CRC 5 Checker: Mux 2	Combinational	2	2,000
CRC 5 Checker: Reg	Register	5	16,000

The area for this block is 0.024 mm^2. This block contain a 3 way 2 bit wide mux and a 2 way 1 bit wide mux. The block also contains a 5 bit register.



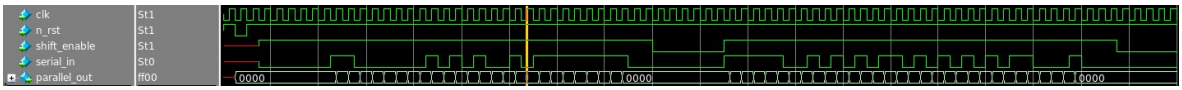
The bits are input and after 13 cycles

CRC 16 Checker



CRC 16 Checker: Mux 1	Combinational	2	2,000
CRC 16 Checker: Mux 2	Combinational	2	2,000
CRC 16 Checker: Reg	Register	16	51,200

The area of this bock is 0.055 mm^2. This block contains 2 mux's which are 2 way and 1 bit wide. Moreover the block also contains a 16 bit register.



In the above timing diagram , after 24 cycles , we get an output of all 0's when passed through the equation.

3) **SD Card Interface**

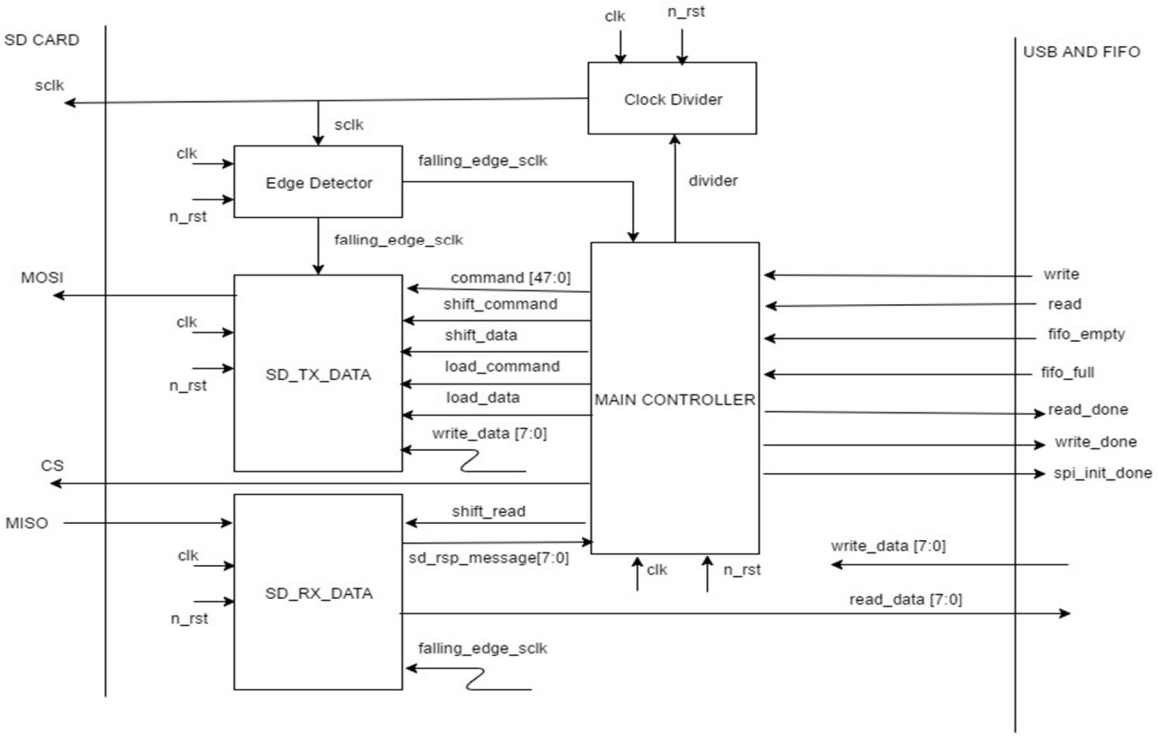


Figure SD Card Interface Design

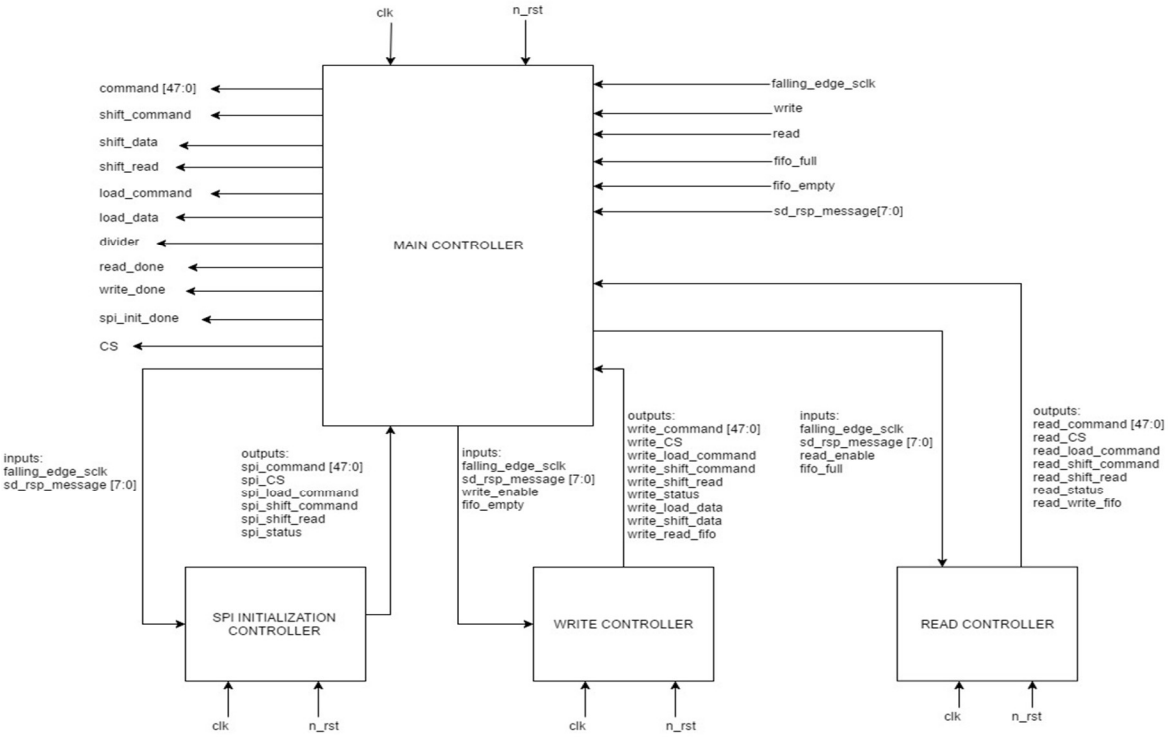


Figure SD Card Main Controller Design

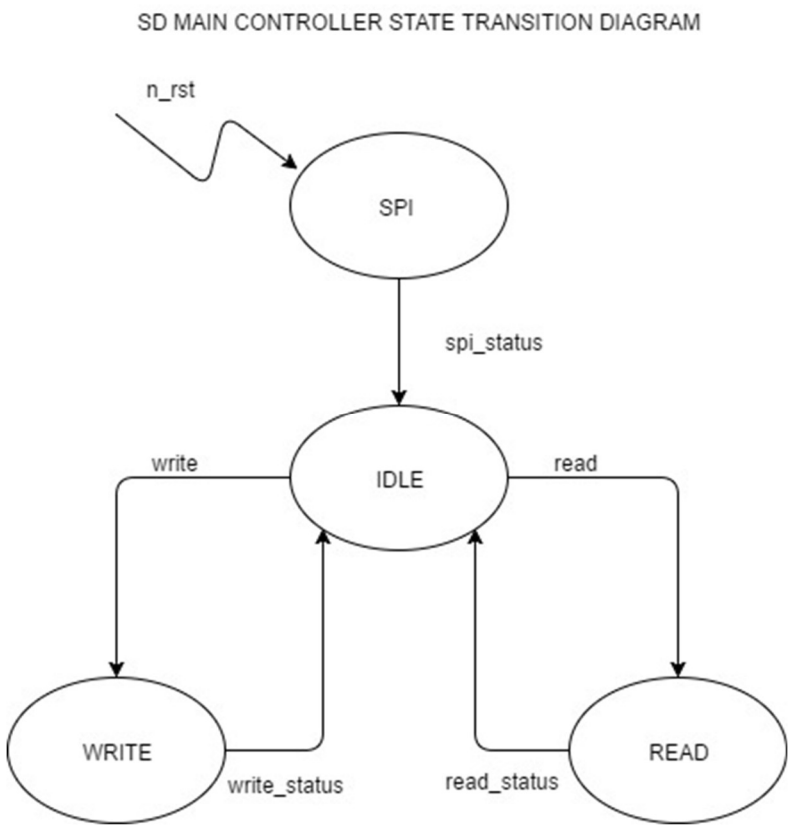
The sd card interface has 3 major functions to initialize the sd card to function in the spi mode, read and write to the card. Using the sd card in spi mode means the MOSI line is used to send data to the card and the MISO line is used to receive data. The sclk line is the clock supplied by the interface which drives the sd card (slave). The read and write capability is a single block of size 64 bits or 8 bytes at a time.

The sd card interface is the part of the design between the usb, data buffers and the sd card. The sd card interface design is shown in the figure above. The blocks that are a part of the sd card interface are Clock Divider, Main Controller, SD_TX_DATA, SD_RX_DATA and the falling edge detector. The Main controller further contains an FSM that chooses between the SPI initialization, read and write controllers and then routes the signals based on the sequence being used. This is shown in the figure above.

The FSM in the main controller sets the main signals with the signal incoming from the sub-controllers depending on the state. For example if it is in the SPI state the command is set equal to the spi_command signal. Essentially making the FSM in the main controller a sort of multiplexer with signals being set based on the state of the main controller.

Main Controller

As show in the state diagram above when the design is powered on or reset the controller initializes communication mode between the interface and the sd card to be SPI. Once it has done so it moves on to an idle state where its stays indefinitely until a read or a write signal is sent. Once a read or write signal is asserted it moves on to the sequence and does not move on until it has successfully completed. The states SPI, write and read are used to route the signals from the sub-controllers to the outputs of the main controller as shown in the figure of the main controller design. On the successful completion of a sequence it asserts a signal to the design to show that it is idle and can start a new sequence.



SD Main Controller: Register	Register	2	6400
SD Main Controller: Next State Logic	Combinational	5	5000
SD Main Controller: Output Logic	Combinational	8	8000

SPI Initialization Controller

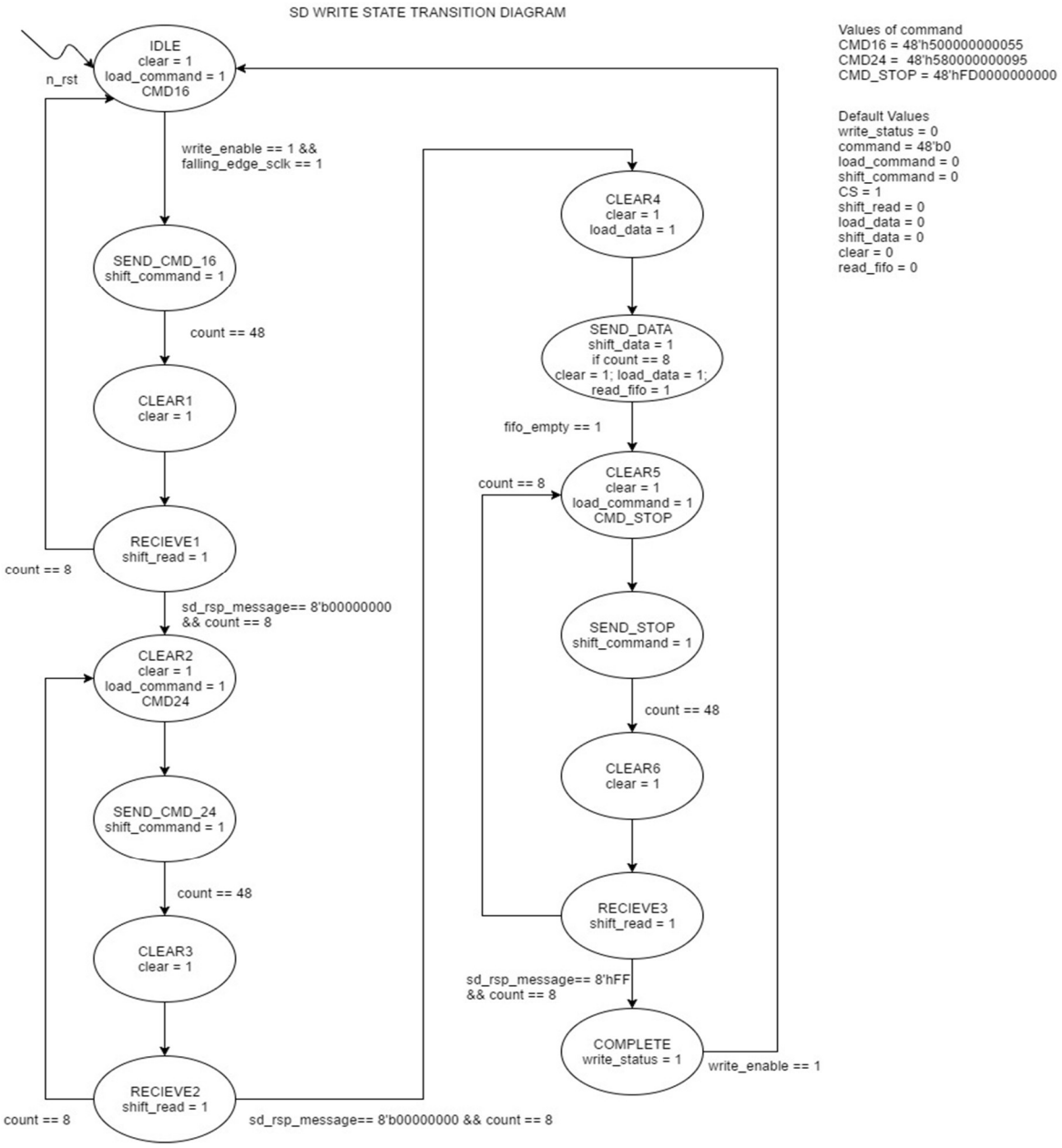
The SD initialization controller is a sub controller which is used by the main controller to complete the spi initialization sequence of the SD card. In this mode the sclk is set to be 400 Khz until the initialization is complete and spi_status is asserted. The commands for the spi initialization are sent by this controller to the main controller and by extension to the sd card. The responses are then verified and if it does not match the command is sent repeatedly until the desired response is received. An unique aspect of this controller is that it does not change states after the completion of the sequence and can only be called again if the system is reset.

SD SPI Initialization Controller:	Register	4	12800

Register			
SD SPI Initialization Controller: Next State Logic	Combinational	2	2000
SD SPI Initialization Controller: Output Logic	Combinational	6	6000

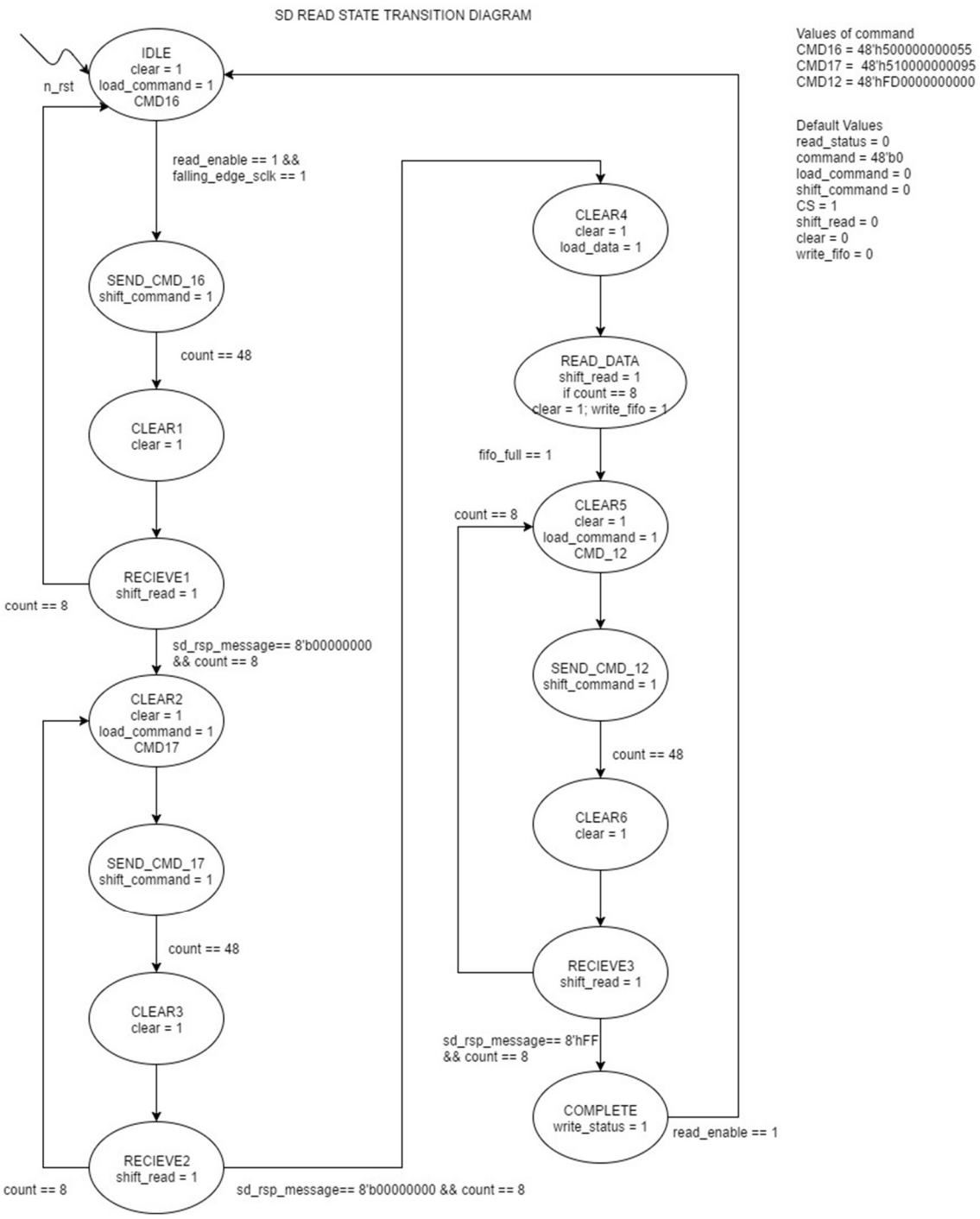
Write Controller

The SD write controller is a sub controller which is used by the main controller to complete the write sequence of the SD card. The commands for the write sequence are sent by this controller to the main controller and by extension to the sd card. The responses are then verified and if it does not match the command is sent repeatedly until the desired response is received. The writing of data the sd card is also controlled by this block. The controller is capable of performing a single block write of size 64 bits. Upon the completion of this block the fifo which holds data to be written into the SD card is emptied and the write sequence is done following which the sd card moves to the idle mode awaiting its next task.



SD Write Controller: Register	Register	4	12800
SD Write Controller: Next State Logic	Combinational	4	4000
SD Write Controller: Output Logic	Combinational	9	9000

Read Controller

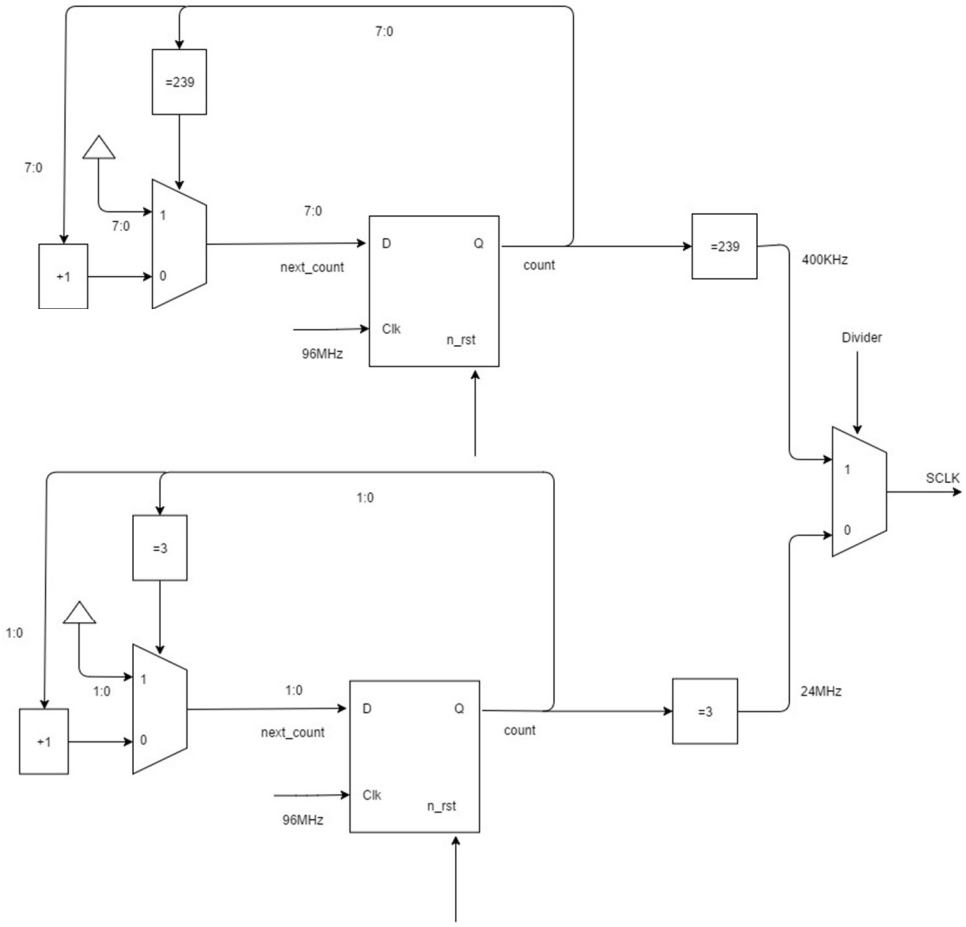


The SD read controller is a sub controller which is used by the main controller to complete the read sequence of the SD card. The commands for the read sequence are sent by this controller to the main controller and by extension to the sd card. The responses are then verified and if it does not match the command is sent repeatedly until the desired response is received. The reading of data the sd card is also

controlled by this block an important part is whether the register in the rx block holds data that is garbage, a command or data to be written to the fifo. The controller is capable of performing a single block read of size 64 bits. Upon the completion of this block the fifo which holds data to be received from the SD card is emptied and the read sequence is done following which the sd card moves to the idle mode awaiting its next task.

SD Read Controller: Register	Register	4	12800
SD Read Controller: Next State Logic	Combinational	4	4000
SD Read Controller: Output Logic	Combinational	7	7000

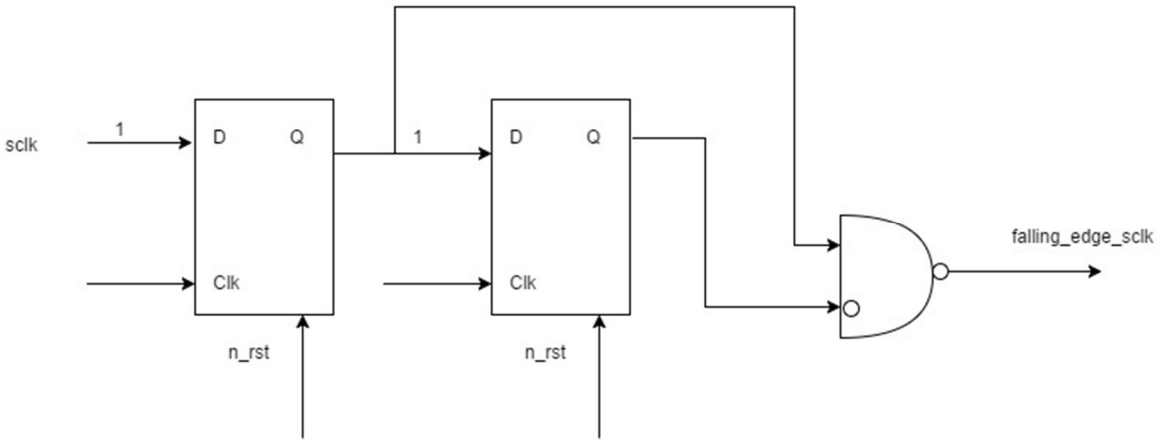
Clock Divider



The clock divider is used to divide the system clock into speeds which are required by the SD card. The initialization sequence of the sd card requires the sclk to run at 400 Khz while the read and write require to run it at 24 Mhz. Further the sclk is also passed on to the edge detector to find its falling edge which is used to send and receive data at the required timing.

Clock Divider Register: Mux 1	Combinational	2	2000	1 bit wide, 2 way mux
Clock Divider Register: Reg1	Register	3	9600	
Clock Divider Register: Reg2	Register	8	25600	

Edge Detector



The edge detector find the falling edge of the sclk. The edge found is used to transmit and receive data from the MOSI and MISO lines at the edges. It is also used to count number of cycles that have occurred in the SD card at the sclk speed which is used to move between states in the sub-controllers.

Edge Dectector Register: Reg	Register	2	6400
Edge Dectector : NAND gate	Combinational	1	1000

SD_TX_DATA

The block is a parallel to serial shift register which makes use of the flex parallel to serial register to shift out the data on the MOSI line. There are two instantiations of the flex shift register one for the 48 bit command and one for the 8 bit data packets. The load and shift command for the two being separate but the shifting of data taking place on the falling edge of the sclk.

SD_TX_DATA Register: Mux 1	Combinational	2	2000	1 bit wide, 2 way mux

SD_RX_DATA

The block is a serial to parallel shift register which makes use of the flex serial to parallel register to shift out the data on the MISO line. There is one instantiation of the flex register for both the 8 bit response message from the sd card as well as the 8 bit data packet. The shift command for both are the same and the shifts takes place on the falling edge of the sclk. The data is stored in a common 8 bit register and depending on the need is used to check for the response message or loaded into the fifo.

SD_RX_DATA Register: Reg	Register	8	25600

Design Timing analysis

Starting Component	Propagation Delay (ns)	Combinational Logic	Propagation Delay (ns)	Ending Component	Setup Time or Propagation Delay (ns)	Total Path Delay (ns)	Target Clock Period (ns)
SD Card Controller - Inputs	0	Next State Logic	5	Reg	1	6	10
Transmitter CU - Inputs	0	Next State Logic	5	Reg	1	6	10
Receiver CU - Inputs	5	-	-	Reg	1	6	10
Receiver Timer - Inputs	0	Flex Counter	10	byte_recieved	0	10	10

Table: Per path timing estimation table

Timing assumptions

tp FF	1 ns
tsetup FF	1 ns
tp gate	2 ns
tp Next State	5 ns
tp Output Logic	3 ns
wire	0 ns

Table: Timing values assumptions

The system clock that we use for our design is 96MHz. This corresponds to a period of 10ns, which is also the target clock period.

Receiver Timer:

By analyzing the receiver timer RTL, there is only one possible path:
Inputs → Flex Counter 1 → Flex Counter 2 → Outputs

assuming that the flex counter takes 5ns of delay we see that the total delay through the path will be 10ns. This exactly the same as the target clock period, which means that the data is arriving just in time.

SD Card:

From this RTL we can see that there are several paths through this RTL. From the assumptions table above, the next state block has the largest delay. Therefore one possible path that can be constructed is as follows:
Inputs → Next State Logic → Register

The delay in the next state logic block is 5ns and the register setup time is 1ns. This come to be a total delay of 6ns. another possible path could be as follows:
Register → Next State Logic → Register
although, the has the same path delay as the path mentioned above

Transmitter CU:

The longest path through this RTL is through the next state logic block. The path is as follows:
Inputs → Next State Logic → Register

The delay of the next state logic block is 5ns and setup time of the register is 1ns. This comes out to have a total delay of 6ns.

Receiver Control Unit:
The longest path through this RTL is through the next state logic block. The path is as follows:
Inputs → Next State Logic → Register

The delay of the next state logic block is 5ns and setup time of the register is 1ns. This comes out to have a total delay of 6ns.

Success Criteria:

1. **Fixed Criteria**:
   1. Test benches exist for all top-level components and the entire design. The test benches for the entire design can be demonstrated or documented to cover all of the functional requirements given in the design specific success criteria.(2)

      Testbenches exist for all relevant files

   2. Entire design synthesizes completely, without any inferred latches, timing arcs, and, sensitivity list warnings.(4)

      Synthesizes without errors

   3. Source and mapped version of the complete design behave the same for all test cases. The mapped version simulates without timing errors except at time zero.(2)

      Source and Mapped behave the same

   4. a complete IC layout is produced that passes all geometry and connectivity checks.(2)

      Layout produced

   5.The entire design complies with targets for area, pin count, throughput , and clock rate.

      Design compiles

2. **Design Specific Success Criteria**(8 points)

1. Demonstrate through the use of Verilog test bench that the complete design is able to correctly service USB host managed block IO read and write requests to/from the attached SD card, via two subsequent Bulk Transfer Sequences with the first for the request's command information and the second for the corresponding Block Data. (1pt)
   [Not demonstrated]

2. Demonstrate through the use of a Verilog test bench that the SD card interface module is able to correctly Initialize the SD Card. (1pt) **[Demonstrated]**
3. Demonstrate through the use of a Verilog test bench that the SD card interface module is able to correctly read and write blocks of data to the SD Card. (1pt) **[Partially Demonstrated]**
4. Demonstrate through the use of a Verilog test bench that the design's USB 1.1 Slave interface portion of the design is able to correctly handle Bulk Transport Sequences. (1pt) **[Demonstrated]**
5. Demonstrate through the use of a Verilog test bench that the design's USB 1.1 Receiver module is able to correctly handle reception of the packet types that must be received by a USB 1.1 slave device in order to support Bulk Transport Protocol. (2pt) **[Partially Demonstrated]**
6. Demonstrate through the use of a Verilog test bench that the design's USB 1.1 Transmitter module is able to correctly handle sending of the packet types that must be sent by a USB 1.1 slave device in order to support Bulk Transport Protocol. (2pt) **[Partially Demonstrated]**

Design Verification:

1. USB Design:

- DSSC Proved:1, 4,5,6
- Highest level of Design Module involved: USB Design (Receiver and Transmitter)
- Test bench expectations/requirements:
    - Generate D+ and D- lines to be transmitted to the USB Receiver.
    - Compare received D+ and D- lines with the packets which were transmitted by the transmitter based on the signals received by the receiver from the BD Controller and host.
- No pre/post processing is needed
- Main Verification Test Steps:
    - Receiver

        - Generate D+ D- minus which properly decode and represent SYNC byte, IN PID, OUT PID. DATA PID, device address, CRC, EOP and payload.
        - Check if the receiver comprehends the right packet type and asserts respective signals through the BD controller and sends type of packet received to the BD Controller..
        - Check if the receiver is capable of identifying 6 1's in a row and eliminate the 0 following it via the bit destuffer module.
        - Check if the receiver is capable of identifying the payload and writing it to the correct FIFO.
        - Check if the receiver can handle errors and flush the FIFO if an error occurs.
        - Check if the receiver has a bit rate of 12 Mbps.
    - Transmitter
        - Check if the transmitter is capable of generating the D+ and D-minus based on the data that is to be transmitted.
        - Check if the transmitter can read from the FIFO and generate CRC based on the data.
        - Check if the transmitter can stuff a 0 bit when 6 1's are encountered in a row. (Bit Stuffer module)
        - Check if the transmitter can transmit the proper packet based on the signals received by the BD Controller.
    - BD Controller
        - Check if the BD Controller enables proper communication between the receiver and transmitter.
        - Check if the BD Controller enables the RX and TX properly.
    - FIFOs
        - Check if the FIFOs write and read proper.

2. CRC Generator and Checker:
- DSSC Proved:5,6
- Highest level of Design Module involved: USB Design (Receiver and Transmitter)
- Test bench expectations/requirements:
    - Generate 0's when the checker is being tested
    - Generate the correct checksum when generating CRC
- No pre/post processing is needed

- Main Verification Test Steps:
    - CRC 5 and 16 Checker
        - Input any value connected with the correct checksum
        - Generate the checksum for the number from an online source
        - Check if the all 0's are output after 5 bits (5 bit checker) and 16 bits (16 bit checker)
    - CRC 16 Generator
        - Provide a value padded with 16 0's at the end of the number
        - Produce the checksum for the associated number
        - Check if the expected checksum is produced as the output

3. SD Interface:
- DSSC Proved: 2,3
- Highest level of Design Module involved: SD Card
- Test bench expectations/requirements:
    - Initialize the SD Card to SPI Mode
    - Read/Write to a designated address in the SD Card
- No pre/post processing is needed
- Main Verification Test Steps:
    - Check if the commands and responses are as per the standard to initialize the SD card to SPI mode (verified in demo)
    - Transmit/Receive data to the SD card buffer (partially verified in demo)

Actual Division of Tasks:

Kunwar Digraj Singh Jain
- Made RTL's for block diagrams
- Design Verification (Top level and USB test benches)
- Coded Receiver files
- Mapped and Layout for USB

Pujitha Desiraju
- Made RTL's for block diagrams and documentation
- USB design and implementation
- Assisted with coding in USB sub module files

Apoorv Sharma
- Made RTL's for block diagrams
- Design Area and Cost Analysis (Estimates)

- Coded Transmitter files

Pushkal Vaid

- RTL's for block Diagrams
- Coded SD card design and testbench files
- SD Card Design and Documentation

Appendix:

Account and directory where all USB files are located: **mg134**/ece337/Project_337

1. USB

- **Design Verilog Code Modules**
  - Top level structural Verilog code: source/USB.sv
  - Top level for receiver: source/receiverUSB.sv
  - RX Controller: source/usb_rx_usb.sv
  - TX Controller: source/tcu.sv
  - NRZI Encoder: source/nrzi_encode.sv
  - NRZI Decoder: source/nrzi_decode.sv
  - RX Timer: source/counter.sv
  - TX Timer: source/timer_tx.sv
  - Read FIFO: source/readFIFO.sv
  - Write FIFO: source/writeFIFO.sv
  - BD Controller: source/bd_controller.sv
  - Bit Stuffer: source/bit_stuffer.sv
  - Bit De-stuffer: source/bif_destuff.sv
  - STP: source/serial_2_parallel.sv
  - PTS: source/parallel_2_serial.sv
  - CRC 16 Generator : source/crc_16_gen.sv
  - CRC 16 Checker : source/crc_16_checker.sv
  - CRC 5 Checker : source/crc_5_checker.sv
- **Test Benches**
  - Top level USB: source/tb_USB.sv
  - Top level receiver: source/tb_receiverUSB.sv
  - CRC 16 Generator : tb_crc_16_checker.sv
  - CRC 16 Generator : tb_crc_16_gen.sv
  - CRC 5 Checker : tb_crc_5_checker.sv
- **Mapped File**
  - Top level USB: mapped/USB.v
- **Waves**
  - Top level USB: waves_USB.do
- **Diagrams**
  - BD Controller: Diagrams/BD Controller.jpg
  - CRC 5 Checker: Diagrams/CRC5_Checker.png
  - NRZI Decode: Diagrams/nrzi_decode.png
  - RX Controller: Diagrams/RX_Controller.jpg

    - TX Controller: Diagrams/TX_Controller.png
    - Stuffer: Diagrams/Stuffer.png
    - System Usage: Diagrams/System Usage Diagram_FINAL.jpg
    - System Architecture: Diagrams/SysysemArchitectire.jpg
    - TX Timer: Diagrams/Tx_timer.png
- **Data Sheets**
    - Area and Time Budgeting : Documents/design_budgeting_form.xlsx
    - Final Presentation: Documents/FinalPresentation.pptx
- **Timing Report**
    - Timing Report: timingReports/USB_postRoute.slk
- **Synthesis Reports**
    - Pads File: SynthesisFiles/innovus_pads.io
    - Synthesis: SynthesisFiles/full_run_pads.tcl

3. SD Card

Account and directory where all SD card files are located: **mg137**/ece337/Project/

- **Design Verilog Code Modules**
    - Clock Divider : sd_clock_divider.sv
    - Main Controller : sd_main_controller.sv
    - SPI controller : sd_initialization_controller.sv
    - Read Controller : sd_read_controller.sv
    - Write Controller : sd_write_controller.sv
    - SD_RX_DATA : sd_rx_data.sv
    - SD_TX_DATA : sd_tx_data.sv
    - Edge Detector : sd_rising_edge.sv
    - Top Level : sd_interface_top_level.sv
- **Test Benches**
    - Top Level Test Bench : tb_sd_interface_top_level.sv

References Cited

1. “SD Specifications Part E1 SDIO Simplified Specification.” SD Association, San Ramon, 03-Apr-2006.
2. J. Valvano and A. Gerstlauer, “EE445M/EE360L.6 Embedded and Real-Time Systems/ Real-Time Operating Systems Lecture 8: Secure Digital Card, DMA, Filesystems,” 28-Feb-2017.
3. “Design and Implementation of SD Host Controller IP Core,” Design And Reuse. [Online]. Available: https://www.design-reuse.com/articles/32530/design-and-implementation-of-sd-host-controller-ip-core.html. [Accessed: 03-May-2017].
4. “Secure Digital (SD) Card Spec and Info,” Secure Digital Card Info. [Online]. Available: http://www.chlazza.net/sdcardinfo.html. [Accessed: 03-May-2017].
5. S. Guna, “A synthesizable Verilog Model of the Serial Protocol engine for USB 1.1 device,” 2007. [Online]. Available:
6. M.C.Johnson,Ed.,”ECE 337 Lab 6: USB Peripheral Receiver Mini Design Project”
7. "Online CRC Calculation". Ghsi.de. N.p., 2017. Web. 4 May 2017.