USB Data Encryptor/Decryptor

Design Review Document

Team Members: Thomas Golden, Kendrick Lau, Jonathan Reitz, Neil Adi

Class: ECE337

Lab Date & Time: Tuesday 2:30-5:20 (Lab Section 2)

Submission Date: 03/31/2018

1.0 Executive Summary

With society self-activating more towards total technological immersion, the amount of data that a single person generates on any given day is growing exponentially. With this, data security and encryption/decryption are now invaluable to the everyday user as the means in which data can be stored efficiently and securely is becoming more constrained. Various data generating interactions such as creating and storing PIN numbers, registering an individual to a specific bank account, or transferring files from one user to another over a number of mediums incite the need for a feasible means of ensuring that this data won't be susceptible to malicious intent while also ensuring that the way in which this data is stored doesn't take up an exorbitant amount of space. That is to say, with all of this in mind, society's need for a way in which to encrypt and decrypt this stored data without the use of an external device is a concept that has become more important and necessary than it ever has been before. Ideally, this implementation of a fast, electrically efficient, and compact data encryption and decryption device will aid in saving this company a lot of money as less data storage space will be needed to withhold the same amount of information.

Our team of dedicated programmers and designers are proposing a USB data encryptor and decryptor to perform the necessary functions outlined above. As mentioned previously, our design aims to perform all of the necessary implementations without the use of any additional or external devices aside from the USB device itself which not only makes it more cost and space efficient to this company, but also more applicable to the many data storage needs that it undoubtedly maintains. Additionally, our design is unique such that no internet connection is necessary for this device to perform the operations that it does which not only gives it a greater sense of security, but also a more cost effective approach to solving the problem at hand. Encryption and decryption, as a whole, relies heavily on an algorithmic process that manipulates data similarly and repeatedly for each and every bit of text that requires storage in the system. Because of this, hardware designed specifically to repeat this task thousands and thousands of times makes the most sense in terms of implementation as a System-on-Chip design since the dedicated hardware can perform all of the necessary functions quickly, accurately, and with little variation in its actual process. To accurately design this device to encrypt, store, or decrypt information, various resources such as the algorithmic decoding formulaic documentation of the data, the Verilog HDL mapped simulation verification of the data, and the test bench substantiation of the finished design are all necessary to ensure that the device is created, and created successfully. The remainder of this proposal will go into more detail of the architecture of the device itself, the algorithms and design specifications with which we intend to immerse our design, and the specific pin-outs that we aim to employ in our device so that this company can visualize our idea as we see it and retain as much confidence in our ambitions as we do. Additionally, we have included some waveform diagrams as well as an overarching step by step process of the standard operation of the device in order to help you get a more in depth sense of what to expect from our design.

2.0 Design Specification

2.1 System Usage

2.1.1 System Usage Diagram

Figure 1. System Usage Diagram for USB 3-DES Encryptor/Decryptor

An example system in which the USB Encryptor/Decryptor would be used is shown above in Figure 1. The device is designed to be connected to the USB computer port. To use the device, a user would simply connect the device via a USB connection and wait for the process to be finished before removing the computer from the encryptor. While the USB device is being encrypted, inside, the encryptor is serially receiving data 8 bytes at a time, storing them locally, encrypting them using the 3-DES algorithm, and writing them back to the user's device. This process will then repeat for every packet of data on the user's device and will terminate when there is no more data on the device left to be read.

2.1.2 Implemented Standards and Algorithms Overview

- USB Version 1.1 Standard [1]
    - High speed mode at 12Mbits/s
    - NRZI encoding scheme for twisted pair data lines
- 3-DES Encryption Algorithm [2]
    - Three unique 64 bit keys are used to encrypt/decrypt the data
    - Unique keys will provide 112 bits of effective security
    - Same hardware is used for encryption and decryption
    - Will process 8 bytes of data at one time

2.2 Design Pinout

Signal	Direction (IN/OUT/BI)	Number of Bits	Description
CLK	IN	1	The system clock. **(120MHz)**
nRst	IN	1	This is an asynchronous, active-low system reset. When asserted (logic '0'), all registers/flip-flops in the device must reset to their initial values.
*Vbus	BI	1	The power input. **(5v)** *Not applicable
*Ground	BI	1	The ground. *Not applicable
Data+	BI	1	USB Protocol Data+ line encoded with NRZI
Data-	BI	1	USB Protocol Data- line encoded with NRZI
Encrypt	IN	1	Indicates to the ASIC whether incoming data is to be encrypted (high) or decrypted (low)

Figure 2. Pinout for input/output signals

2.3 Operational Characteristics

2.3.1 Waveform Characteristics

2.3.1.1 Typical USB Start of Packet with Data

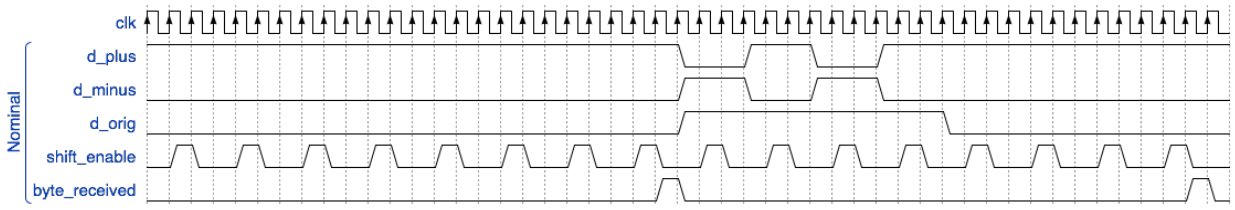

Figure 3. SYNC byte (8'b10000000) followed by data packet (8'b11110000)

USB Bus Typical Start and Packet Usage

On initialization, a SYNC byte (8'b10000000) needs to be sent to the USB bus before the actual packet byte. We shift the LSB in with NRZI coding. When d_plus changes from 0→1 or 1→0, the decoded data returns a 1, and when the d_plus keeps the current input 0→0 or 1→1, the decoded data returns a 0.

2.3.1.2 Typical USB End of Packet

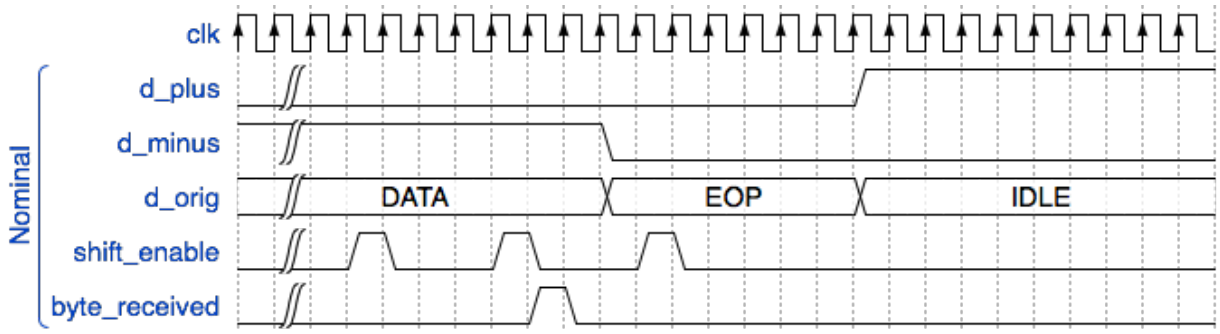

Figure 4. End of packet typical usage

USB Bus Typical End of Packet Usage

To signal the end of the packet, d_plus and d_minus both go to low. This signal can only come after a sequence of 8 bytes have been registered, i.e. this design does not support the processing of an incomplete packet of data. After any full packet, the data line can be idled and the USB bus can await the next SYNC byte and data packet.

2.3.3 USB Packet Types

Token Packets

Token packets synchronize the host and receiver to inform what type of directional information is going to be sent. There are three types:

1) In – Indicates that the USB device wants to read data
2) Out – Indicates that the USB device wants to write data

A token packet must follow the format:

SYNC	PID	EOP

Data Packets

Data packets transmit the required data from host to receiver. The maximum data payload size must be sent in multiples of bytes with the maximum being the following:

1) Full-Speed has a maximum data payload size of 1023 bytes.

A data packet must follow the format:

SYNC	PID	Data	EOP

Handshake Packets

Handshake packets lets the USB device know the status of the last packet and if it is in a good receive state:

1) ACK – Acknowledgment that the packet has been successfully received

2) NAK – Reports that the device cannot send or receive data temporarily. This packet can be used in interrupt transactions to tell the host that there is no data to send
3) STALL – The device is in a state that requires the host to intervene
A handshake packet must follow the format:

SYNC	PID	EOP

2.3.4 Packet Fields

SYNC field – Used to synchronize the clock of the receiver and transmitter. At low speed, the sync field is 8 bits, where at full speed, it is 32 bits long.

PID field – This field specifies what type of packet is being sent to the receiver. The following table details the possible Packet IDs:

Group	PID Value	Packet Identifier
Token	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET (No Response Yet)
Special	1100	PREamble
	1100	ERR
	1000	Split
	0100	Ping

Figure 5. Packet ID Lookup Table[1]

The Packet ID is 4 bits, but to ensure accuracy, the 4 bits are complimented and added to the end of the original 4 bits. PID field should have the following structure:

PID[0]	PID[1]	PID[2]	PID[3]	$PID^C[0]$	$PID^C[1]$	$PID^C[2]$	$PID^C[3]$

2.3.2 Encryption/Decryption Operation

Normal Usage of Encryption/Decryption Design:

1) Clear Register of Data to be Encrypted
2) Fill Register containing 64 bits at a time to be Encrypted/Decrypted
3) Using encryption keys, fill output buffer with Encrypted/Decrypted data
4) Once Output Buffer is full, put on USB Data Bus
5) Clear all Registers and Output Buffer

2.3.3 3-DES algorithm

The 3-DES encryption is an industry wide used block encryption algorithm. In order to circumvent the possibility of brute force attacks, each block is encrypted three times using three separate keys: K1, K2, K3. Each block will contain 64 bits to be encrypted by these three keys. The first step is to encrypt this block using K1, the second step is to then decrypt the output of first step using K2, and the third and final step is to encrypt the output of step 2 using K3. The input to the first step will be plain text, and the output to step 3 will be ciphertext. In order to decrypt this data, you will do these steps in reverse order.

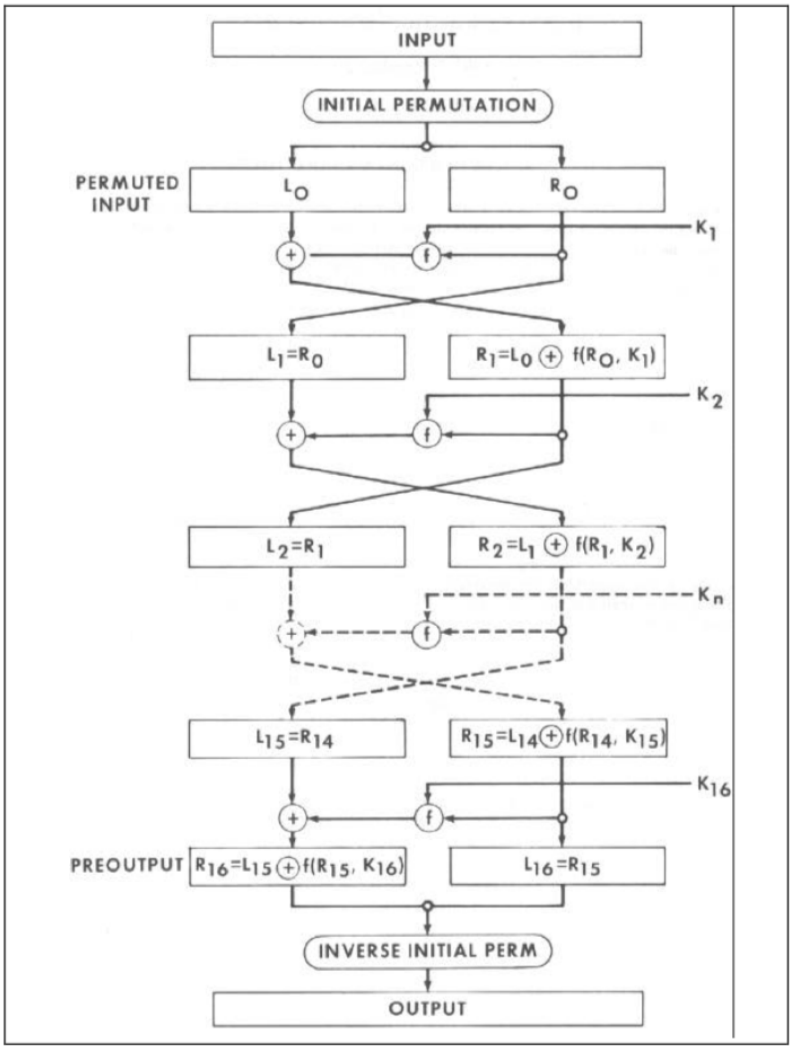

Figure 6. DES Encryption Overview Flow for a Single Key

Above is the overall flow of a round of DES encryption. To summarize, an input 64 bit plaintext block is run through an initial permutation. It is then broken into two 32B words, where the right word is extended to 48B's using an expansion permutation, where it is then XOR'd with a 48B subkey derived using a key schedule. The block is then divided into 8 6B pieces which are processed by S-boxes which replaces 6 input bits with 4 output bits using a lookup table. The

final 32B are then rearranged again according to a fixed permutation. This encoded right word is added to the left word, and it becomes the new right word for the next iteration, with the old right word becoming the new left word for the next iteration. This process is repeated 15 additional times with different subkeys until the final 64B are put into the inverse of the initial permutation and the result is the output. [3]

2.4 Requirements

The implementation of a USB Encryptor/Decryptor requires several design choices to be made based on the desires of the end user. The target user is someone who desires to securely encrypt and decrypt plaintext data on a handheld USB 1.1 storage device. Given that USB 1.1's Full Speed data transfer rate is 12 Mbit/s, our device's clock rate must be a minimum of 96 MHz in order to have an adequate sampling rate of 8 clock cycles per data bit for the incoming data. However, our desired clock rate will be to sample 10 times for each data bit, thus at 120 MHz, to allow for more leeway on the timing of our samples of the incoming data. Additionally, this will provide 10 clock cycles in between bytes to properly check for errors and store the bytes for encryption/decryption. Per our initial estimations of our critical paths, the critical path time is ~7ns, which is within our clock period of ~8.3ns. Since there is a lot of leeway in terms of clock speed for the design, we will optimize our device for area instead. The desire for a small size stems from the device's intended portability, as it is much more useful if a device can be encrypted anywhere easily. As such, our initial area goal is 3mm x 3mm, with the potential to grow as wide as a USB port (~12mm). Our estimation for our total area is ~8.083 mm^2, which is within our current dimension goals. Our goal for the number of pins for this device is what is described above, 7 pins. 4 pins (Data +, Data -, Vbus, and Ground) will be provided via the USB we will attach to our device, with the other 3 pins being provided by the system using our device.

The device will be designed to handle a USB protocol that transfers a maximum packet size of 8 bytes at a time, in order to keep the design size at a minimum. This would require 64B of storage of the plaintext. Additionally each new key is 64B long, which results in an additional need for 192B of storage. A temporary storage space of 8B will also be used for each incoming or departing byte in our device. The plan is currently to use flip-flops for the storage device, as it is simple to implement and allows us to focus more on the correct implementation of the algorithm and USB transfers.

2.5 Design Implementation

2.5.1 Design Architecture

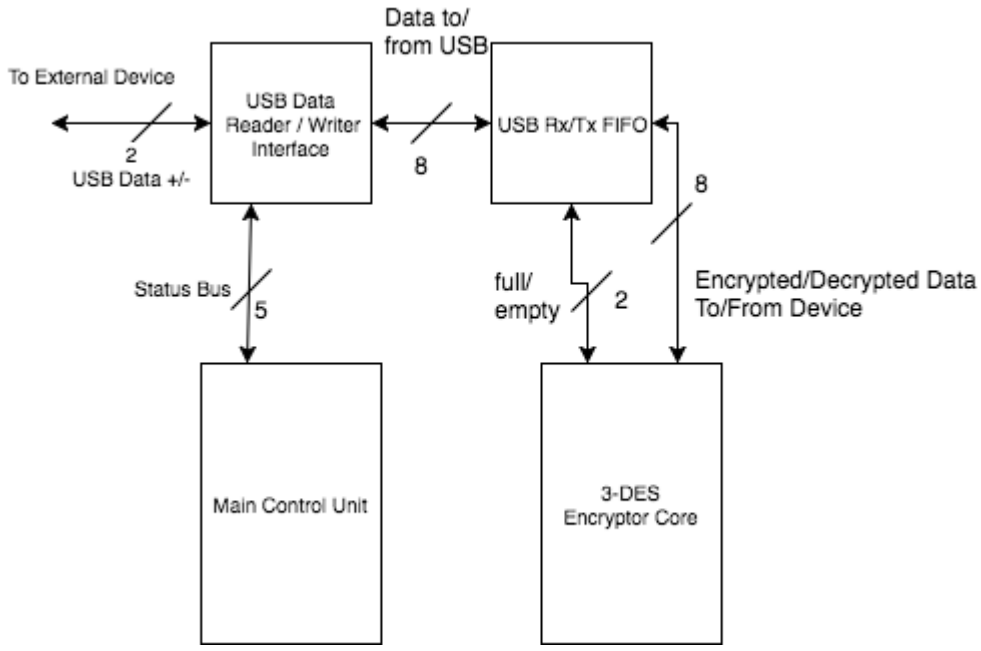

Figure 7. System Usage Diagram for USB Encryption ASIC (clock and reset not shown, but go to each block)

The intended implementation of this device is shown above in Figure 5. The USB Data Read/Write Interface will coordinate and translate the packets of data received from the external USB device into several registers that will be manipulated by the device, including a "status bus" which will indicate the type of packet that was sent, and the data bus, which will contain the raw plaintext loaded from the device that is scheduled to be encrypted/decrypted. This data bus will be fed into a FIFO, that the encryptor/decryptor core will read from after the USB is done sending an 8-byte packet of data. The Encryptor/Decryptor Core houses the main computational logic for implementing the 3-DES encryption and decryption on the USB data. The Main Control Unit will interpret the status bus from the Read/Write Interface and instruct the Encryptor/Decryptor Core on when new data is available and what computations to perform on it. It will be implemented as a Finite State Machine and will utilize various counters to keep track of where the device is in the process of encryption/decryption.

2.5.1.1 Design Area Summary

Name of Block	**Estimated Area (μm^2)**
USB Data Interface	2,284,200
USB Rx/Tx FIFO	157,350

Main Control Unit	30,600
Encryptor/Decryptor Core	1,387,500
Total Chip Area (I/O Pads included)	8,083,236

Figure 8. Major Functional Block Area Estimates

As evidenced above in the area estimation table, the USB Data Interface has by far the largest projected area of the four main functional blocks. This is largely due to the fact that it contains both a USB transmitter and receiver. The dual process of transmitting and receiving data in our design requires this area overhead, and efforts will be made to optimize this block's area as much as possible through both design optimizations as well as synthesis optimizations. Our design area estimates also put us under our 3mm x 3mm goal for our ASIC.

2.5.2 Functional Block Diagram

2.5.2.1 USB Data Interface

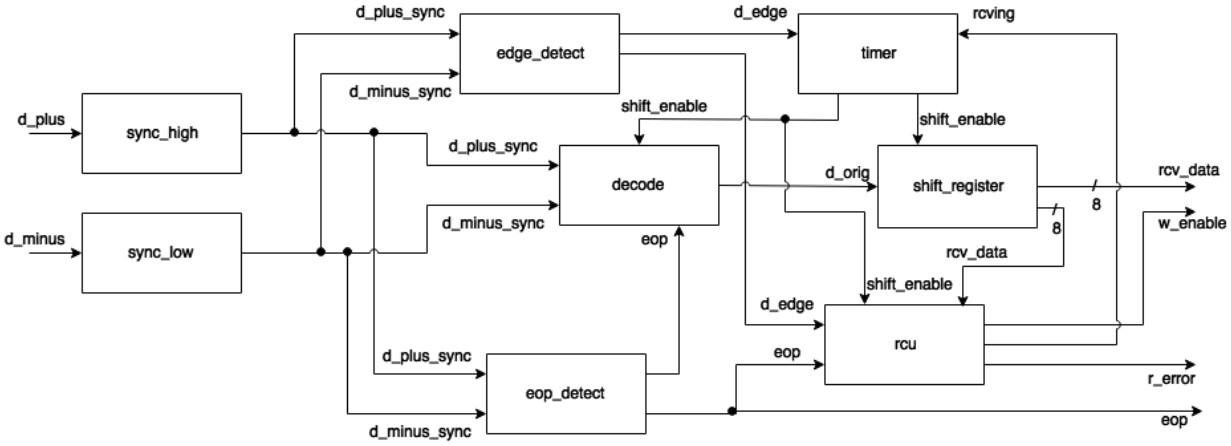

Figure 9. USB Receiver Functional Block Diagram

Many of the receiver blocks used in lab 6 can be modified to implement a higher level of the USB 1.1 protocol. The most significant features are to use d_plus and d_minus to confirm nominal data transfer, and RCU modification to communicate with the MCU block.

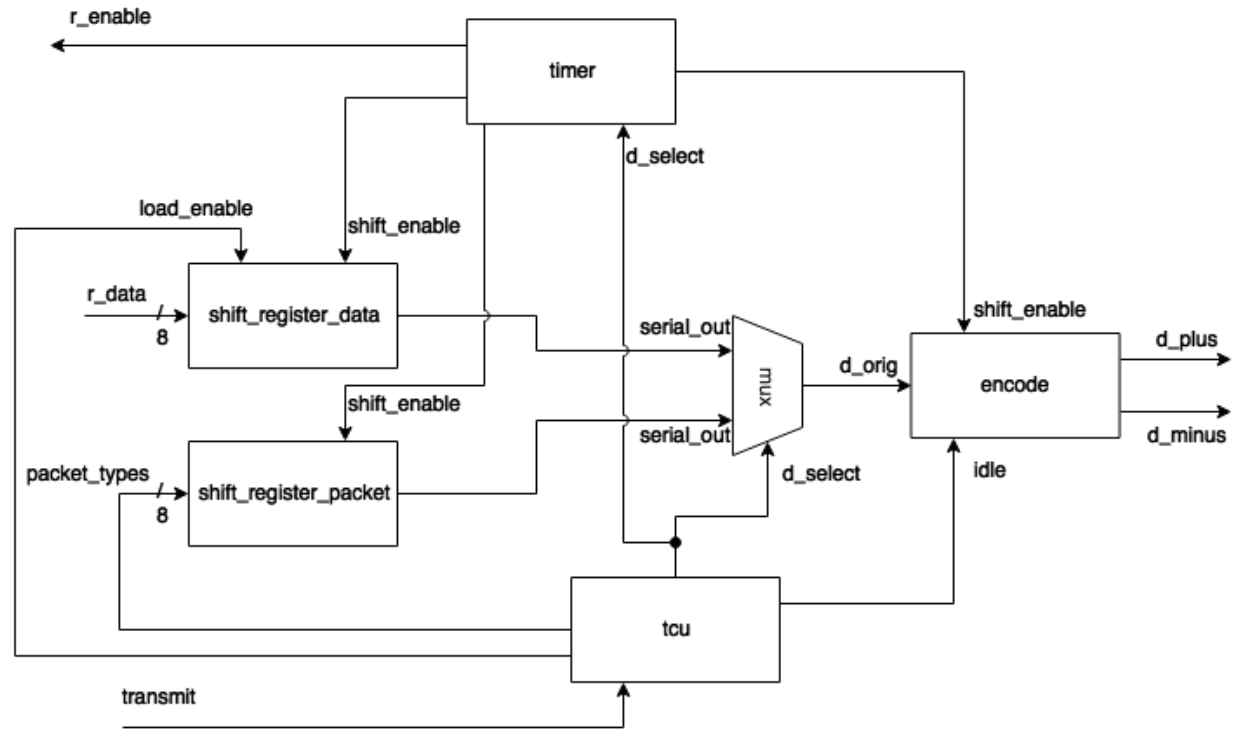

Figure 10. USB Transmitter Functional Block Diagram

The transmitter will handle packet structure and packet ID via the TCU, taking in an instruction set from the MCU. It will encode the data to NRZI format and output on the same bi-directional d_plus and d_minus wires.

2.5.2.2 USB Receiver/Transmitter Functional Blocks

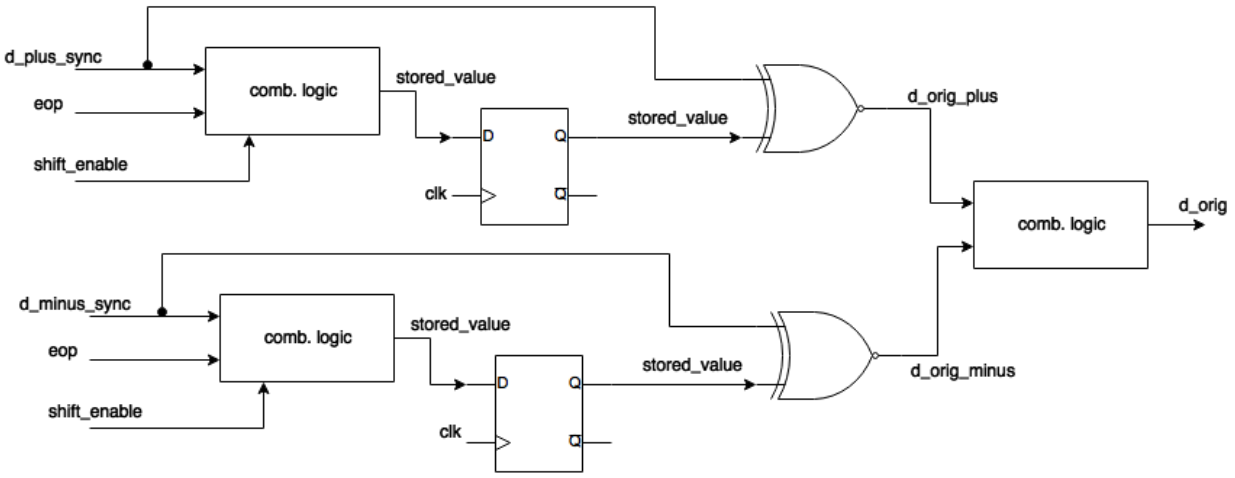

Figure 11. Decoder for USB Receiver

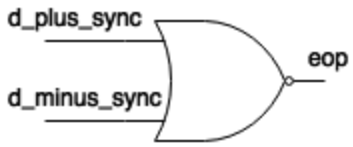

Figure 12. Block for EOP detection

The receiver will decode the NRZI encoded data on the d_plus and d_minus wires. It will also verify that the two wires are complements of each other. EOP will be determined by EOP combinational block.

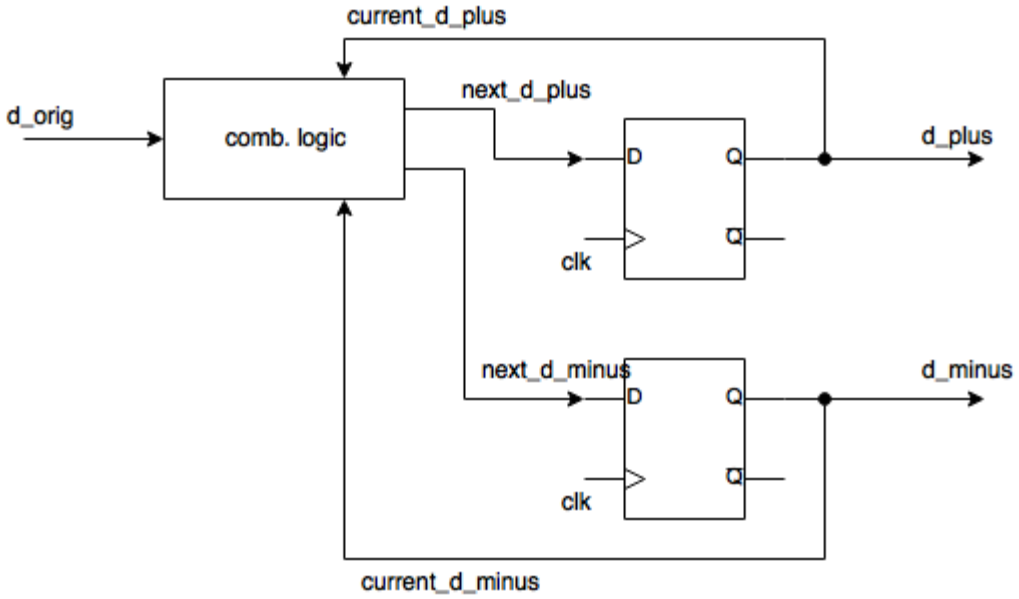

Figure 13. Encoder logic for USB transmitter

The encoder for the transmitter is made of combinational logic. As the encode block receives the data to transmit from the shift registers, it will encode and send it out to the registers. We need to know what was on the previous data line so that we are able to encode NRZI format properly.

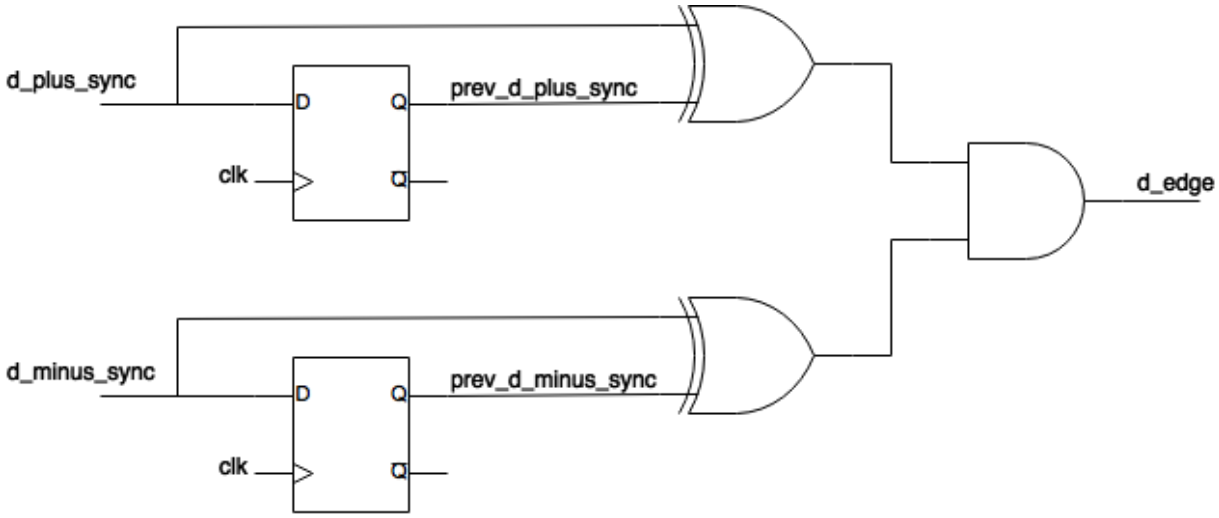

Figure 14. Block for edge detection

Edge detection for the receiver will take in both lines of data to affirm the complements.

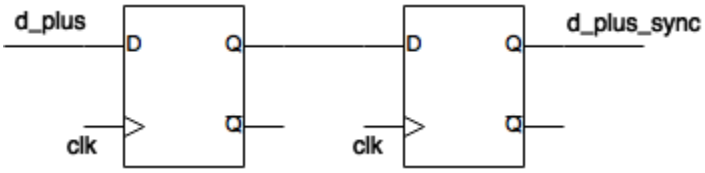

Figure 15. Synchronizer for USB receiver

The data needs to be synced with the system clock so that it can be used by the functional blocks.

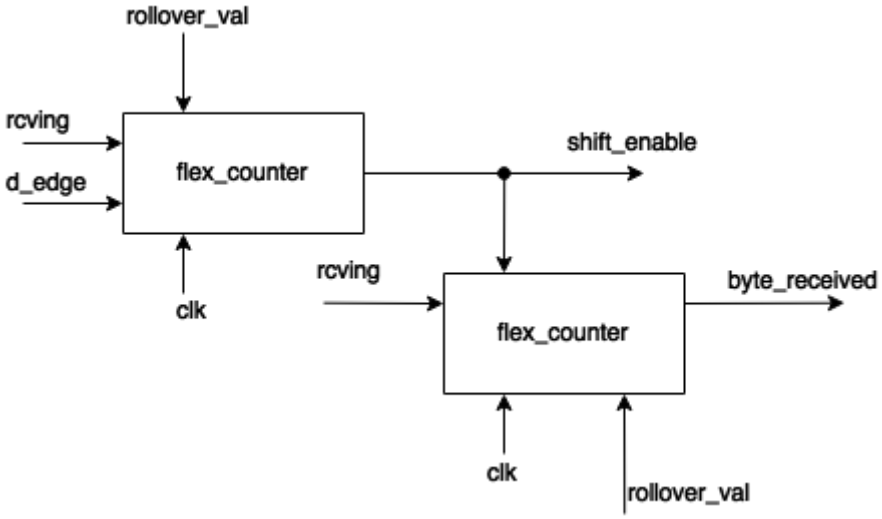

Figure 16. Timer block for receiver/transmitter

The block to transmit and receive the bytes are running on the same clock, since we need to be transmitting data at the USB full speed rate. Shift_enable should provide the timing to the shift registers so that reading and writing data can be timed correctly.

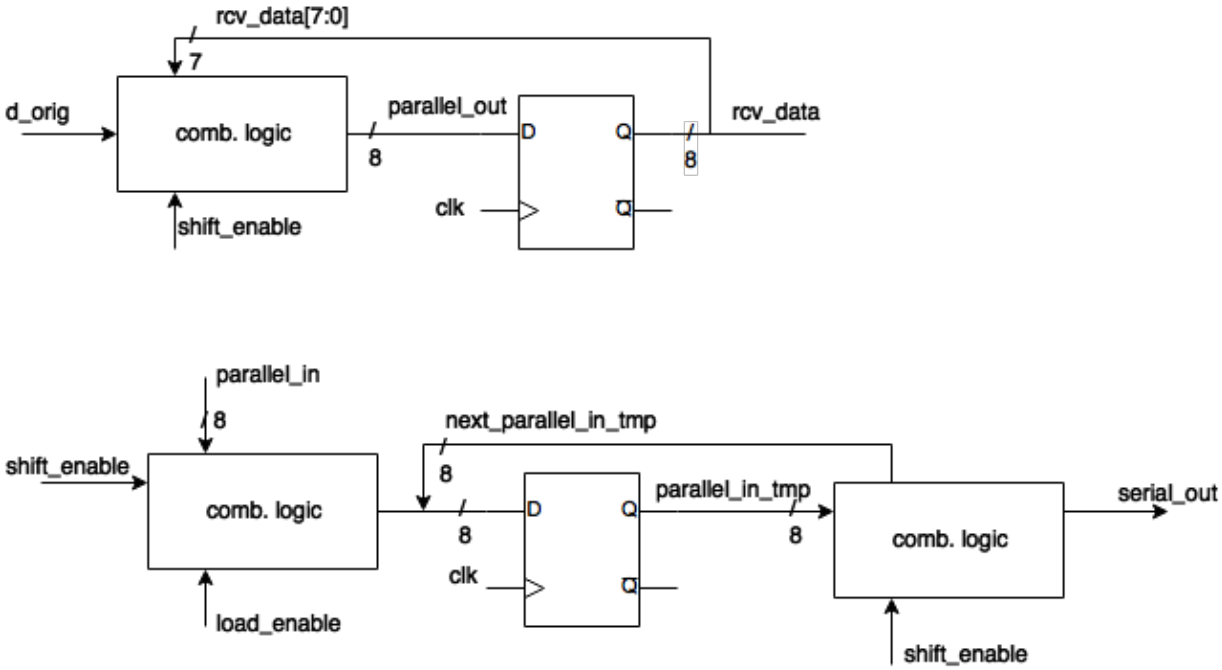

Figure 17. Shift registers for receiver and transmitter

The shift register for the receiver is a serial-to-parallel shifter. This allows for the data to be read via the data lines and put into a form for the data to be loaded into the FIFO. On the transmitter side, the shifter needs to take the parallel FIFO data and organize it so that the data can be transmitted on the data lines.

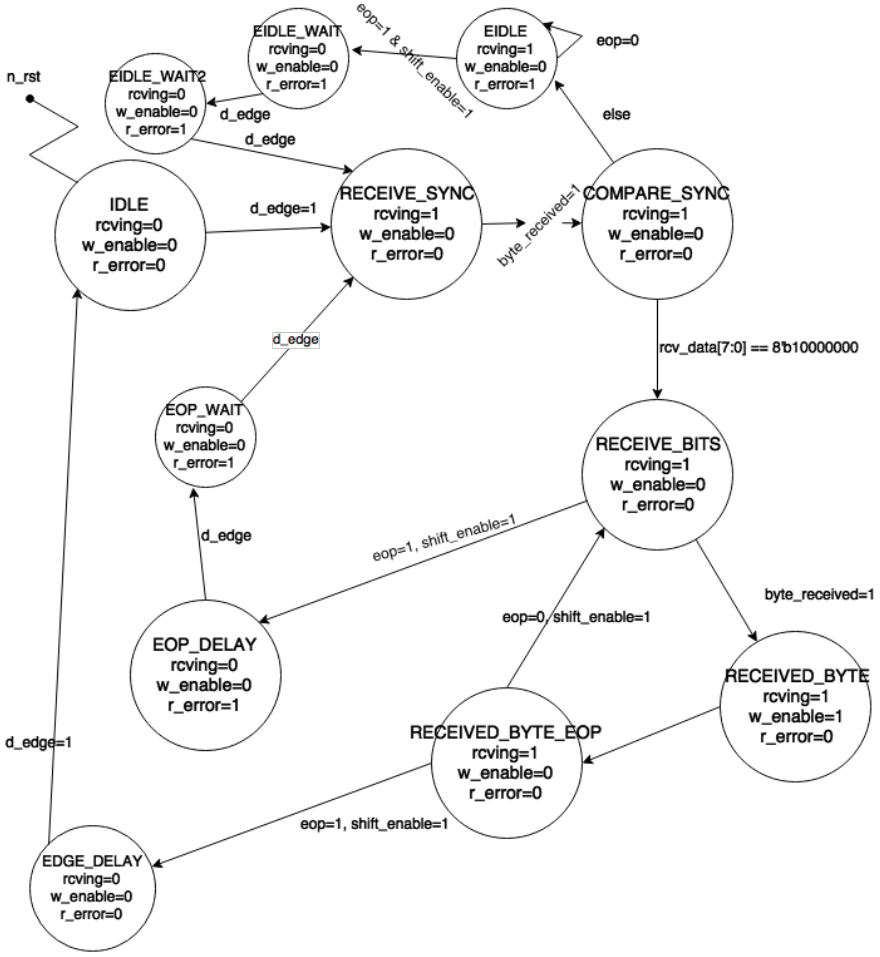

Figure 18. State transition diagram for the RCU

Most of the instruction set will be coming from the MCU, but for the receiver to make decisions faster, we are implementing next-state logic for the RCU. The transmitter will take in an instruction from the MCU to decide when to transmit the packet through.

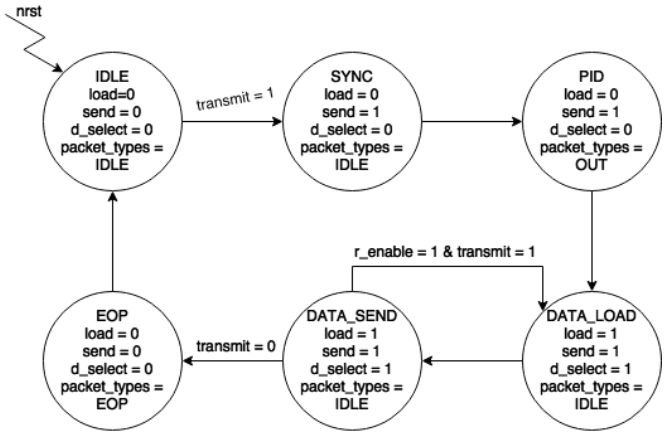

Figure 19. State transition diagram for the TCU

2.5.2.3 Area estimation of USB Transmitter/Receiver

Name of Block	Area Estimation
USB Transmitter Block Logic	385,500
USB Transmitter Registers	494,400
USB Receiver Block Logic	631,500
USB Receiver Registers	772,800

Figure 20. Area estimations for transmitter and receiver blocks

Our area estimations come from the innovus layout of Lab 6, for the USB receiver, and Lab 7, for the USB transmitter.

2.5.2.4 USB Tx/Rx FIFO

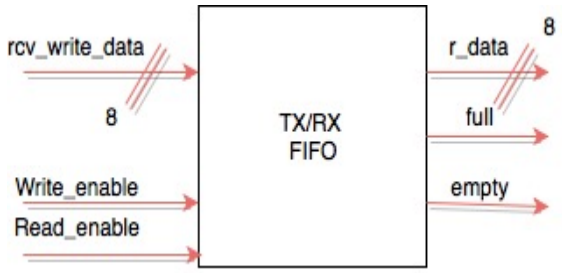

Figure 20. Top-level FIFO block

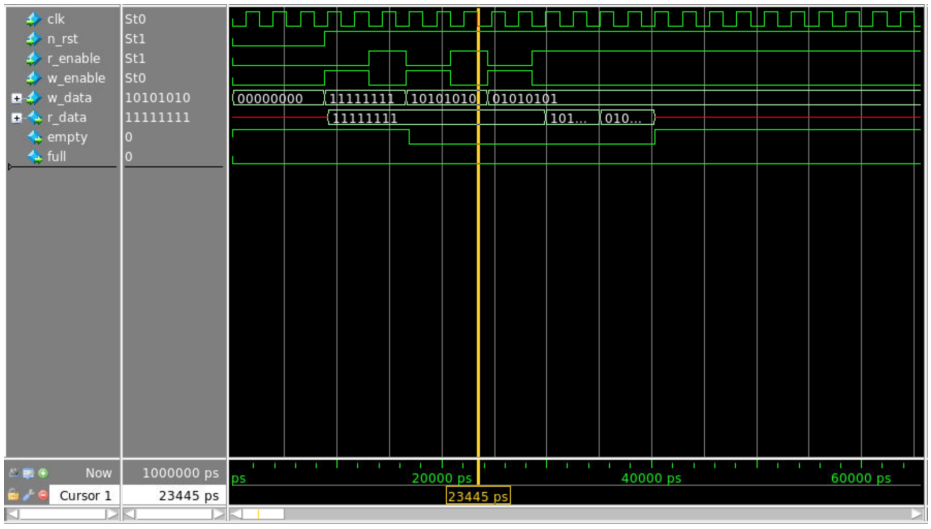

Figure 21. Nominal waveform of FIFO

The USB TX/RX FIFO will act similarly as it did in lab 6 for the USB peripheral receiver. Using the FIFO that has already been used, it will make our job of managing the packets easier while the microcontroller has a chance to read the data.

Below is an overall area estimation table for each of the blocks in the Top Level Diagram.

Name of Block	Area Estimation
FIFO R_data 8 Bytes	153,600
FIFO Full, Empty	3,750
Total USB Tx/Rx FIFO	157,350

Figure 22. Area estimation for FIFO block

A breakdown of the area for the block is provided in the block's corresponding sub-section. Provided is also the RTL diagram for the block.

64 Bit FIFO

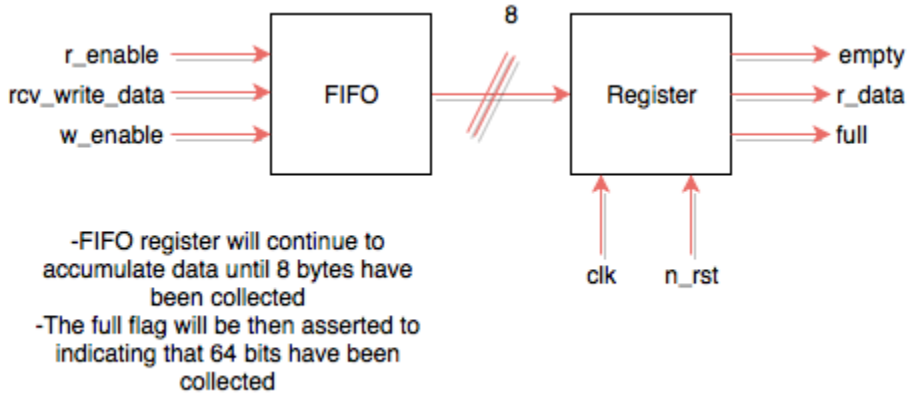

Figure 23. Top level diagram for 64 bit FIFO

The FIFO will act as an accumulator to hold 64 Bits at a time. Because the encryptor will encrypt data at 64 bits at a time and the slower USB transmits data 8 bytes at a time, it made sense to accumulate 64 bits worth of data. This leads to our area estimate for the FIFO to hold a total of 8 bytes or 64 bits.

2.5.3 Main Control Unit

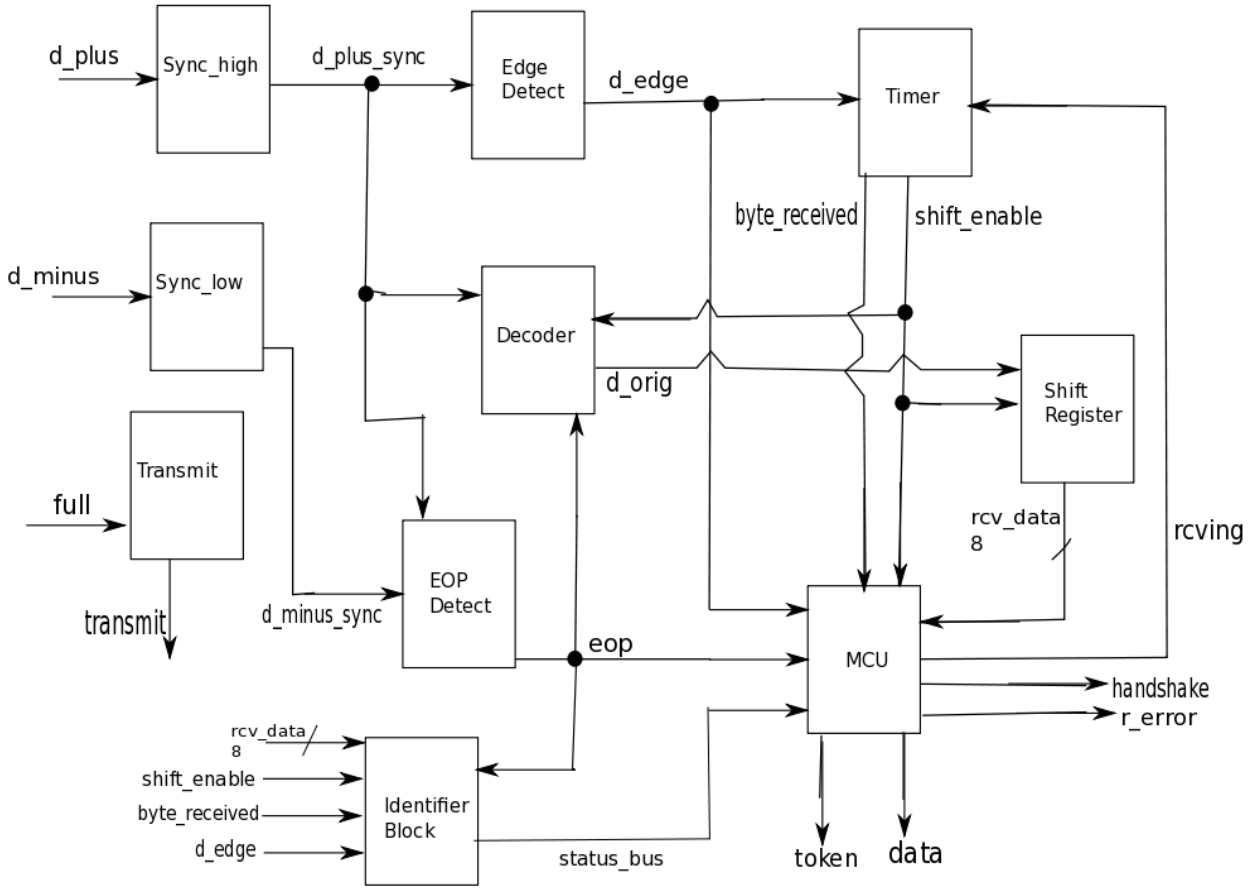

Figure 24. Top Level Block Diagram from the Main Control Unit

The Main Control Unit will function as the brain of the overall design that informs the rest of the functional blocks as to what operation should be executed based on the incoming data. The state diagram will trigger a series of outputs that inform the other modules whether or not the packet received corresponds to a token packet, a data packet, or a handshake packet, while also triggering various output registers that correspond to whether or not an error occurred while receiving data and also whether or not the system is currently receiving incoming data or not. Using additional wire inputs including 'd_edge', 'byte_received', 'eop', and 'shift_enable' from supplemental modules that detail the behavior of the incoming data, the Main Control Unit can encompass the total performance of the incoming data and send appropriate signals to the rest of the functional blocks of how best to proceed.

The Main Control Unit diagram utilizes many of the same signals as the USB Receiver/Transmitter block and as such, many of the same RTL diagrams shown previously are applicable to the blocks shown above in the MCU top level diagram. Firstly, the incoming d_plus and d_minus inputs to the module will have to be synchronized to the system clock so that the values can be utilized appropriately by the various sublevel blocks of the design through the setup shown in Figure 15.

These new synchronized input wires will have to be sent through the eop detector in order for the system to know when the end of a packet has been received and there is no more data being inputted into the system. The eop signal is invaluable to the design and is used in numerous state transition diagrams as well as the decoder functional block and is shown in Figure 12. Additionally, in order to be able to use the d_plus_sync and d_minus_sync values in a meaningful way, the inputs must be passed into a decoder block so that the data can be decoded appropriately as shown in Figure 11. This d_orig value will then be passed serially into the shift register in order to populate the 8-bit rcv_data variable correctly and obtain the first of many bytes that are going to be processed by the system design through a setup similar to that of Figure 17.

As the d_orig bit is sent from the decoder, the shift register will process and store each bit and encode them into the rcv_data variable in such a way that the least significant bit is sent first and the most significant bit is sent last. This will allow the rest of the design to observe the values in the 8-bit wire and determine whether or not the data is being sent properly and what kind of data is being sent at the time. The timing of this process is controlled by the timer functional block that utilizes variables from the MCU as well as the edge detector block as shown in Figure 14. Following a layout similar to that of Figure 16, the timer module is invaluable to the functionality of the MCU as the byte_received and shift_enable signals are referenced and used heavily in the state transitions.

As the receiver/transmitter functional block takes in a transmit signal from the MCU design, a simple register as shown below in Figure 25 is needed so that the transmitter can know when the full 8 bytes have been loaded in and when it can correctly start to transmit the data that is to be encrypted/decrypted. The ‘full’ signal will come directly from the FIFO as a means of determining when the FIFO is full, or when 8 bytes have been loaded into the system. The data flow of this process is shown in Figure 25 below.

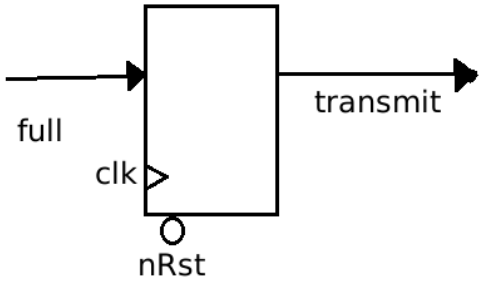

Figure 25. RTL Diagram for Transmit Functional Block

As mentioned previously, one of the main purposes of the MCU is to inform the rest of the system as to what kind of data is being received and what operation should be performed based on the type of data that is being sent. The design will need to be able to handle the processing of various data packet types and ensure that the operation being performed by the encryptor/decryptor block is appropriate for the situation. This logic is handled through the use of two different Moore-state type

machines shown above in Figure 24 as the Identifier Block and the MCU block. Using numerous signals mentioned previously out of the transmitter/receiver functional block, the functionality and flow of the main points of operation of the MCU functional block are shown in the state transition diagrams below.

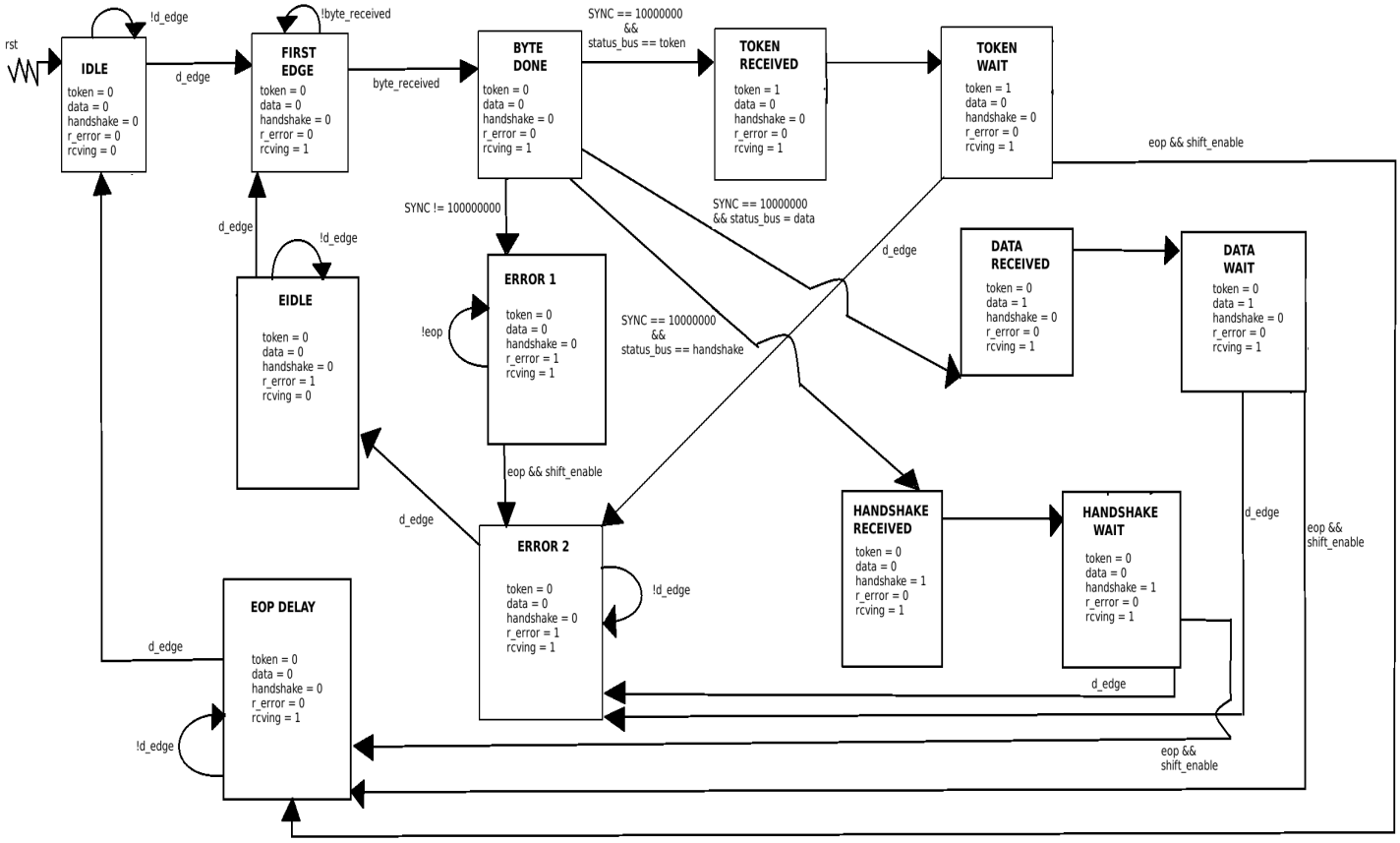

Figure 26. State Transition Diagram for MCU Block

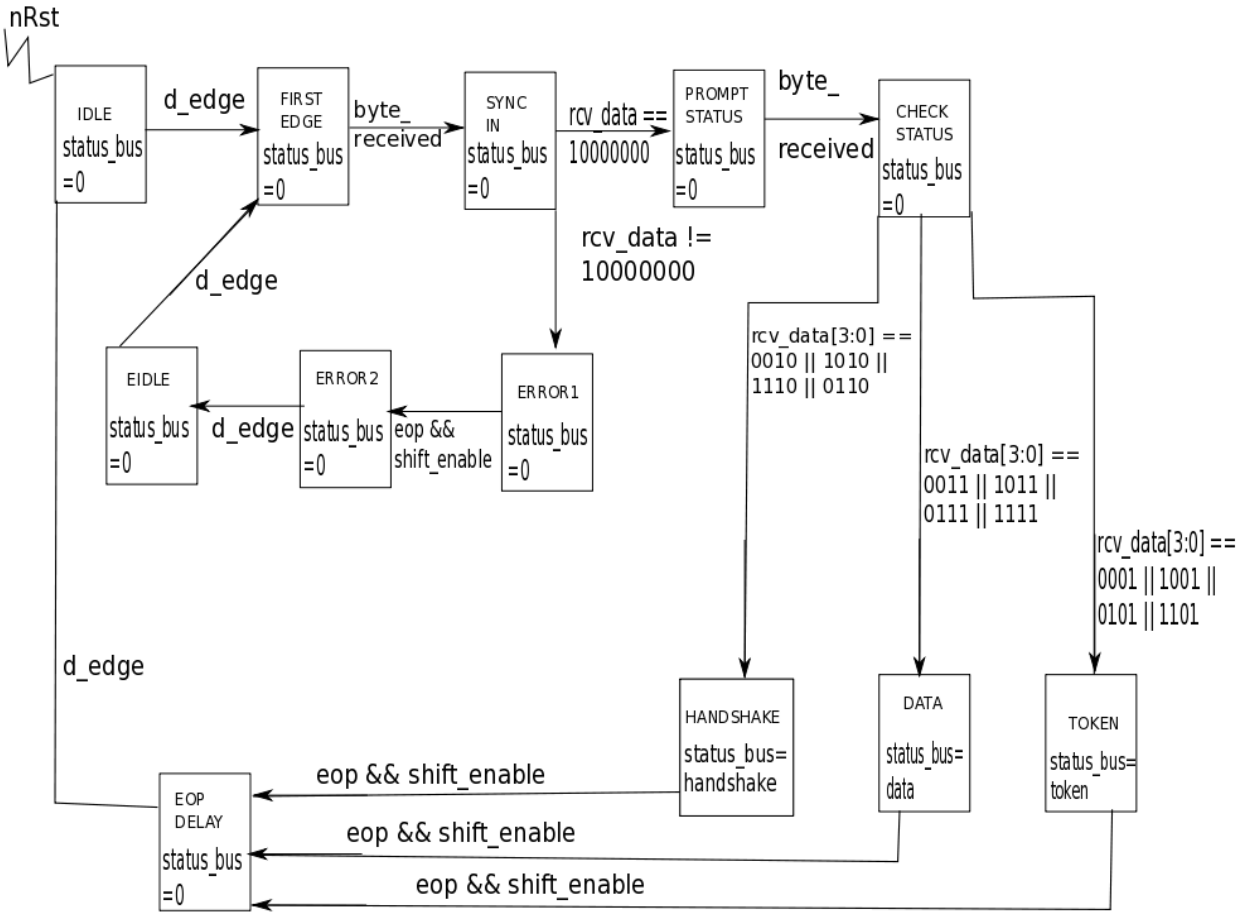

Figure 27. State Transition Diagram for Identifier Block

Name of Block	Area Estimation
MCU Registers	420,000
MCU Block Logic	461,250
Total MCU Area	881,250

Figure 28. Area estimation for Main Control Unit

Using the layouts of labs 6 and 7 as a reference, the total expected area usage is shown above and demonstrated through each subsection of the top level functional block diagram. The total area usage for this module is reduced greatly as many of the signals shown in the top level diagram can be pulled and utilized directly from the transmitter/receiver module of the design.

2.5.2.4 3-DES Encryptor/Decryptor Core

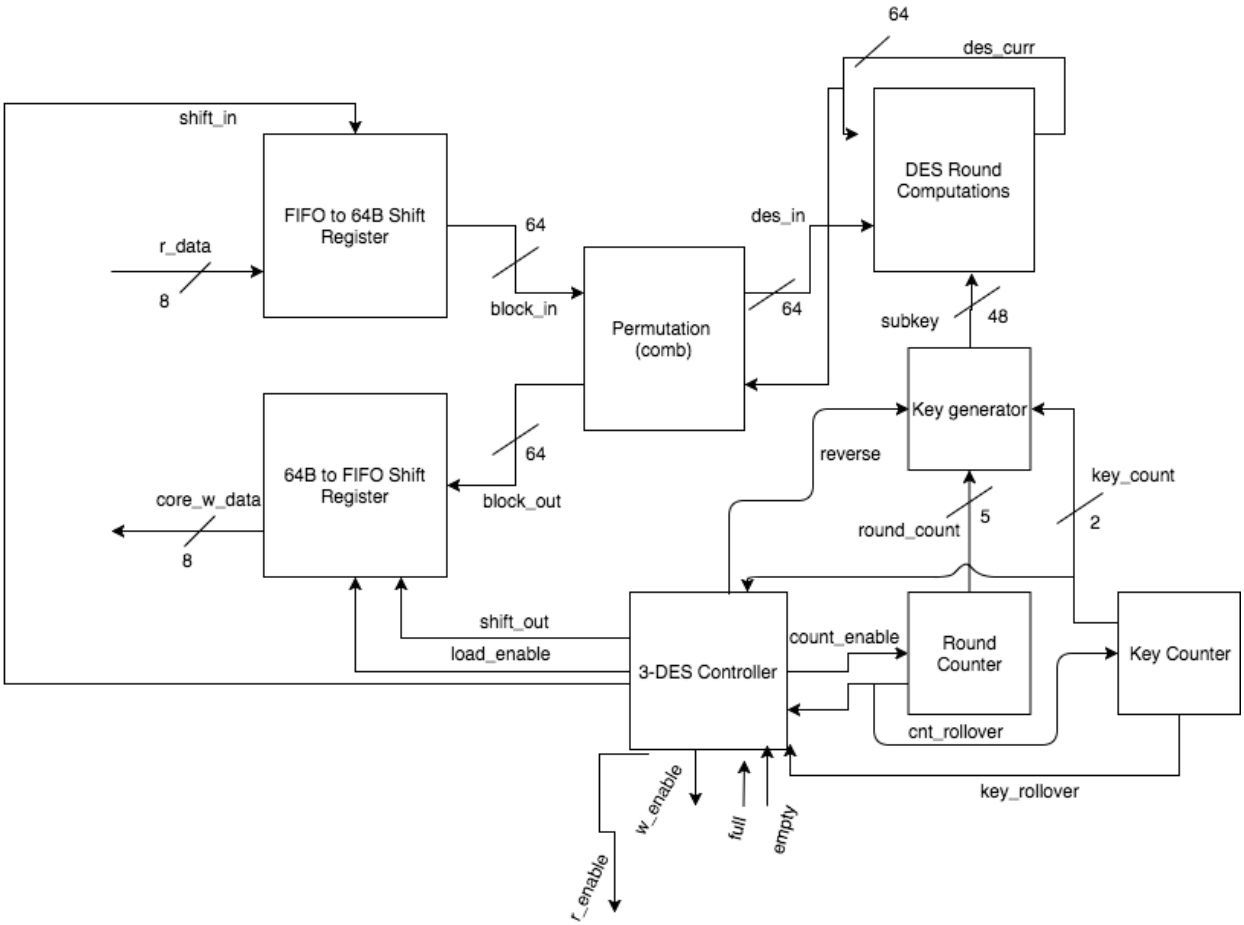

Figure 29. Top Level Encryptor/Decryptor Core Diagram

The 3-DES Encryptor/Decryptor Core is made up of several constituent blocks as shown above. Each of these blocks will be explained in detail in the following sections, but first, an overview of the program flow will be provided along with a State Transition Diagram for the 3-DES Controller Unit. When a Data Packet has been received by the USB interface, it will load 8-bytes of data into the FIFO that will be ready to be encrypted or decrypted. The FIFO will assert a full signal when all 8 bytes have been placed into the the queue. When that signal is read by the controller, the FIFO to 64B shift register will begin shifting in the data and arranging it as a 64B block to be used in the 3-DES algorithm. After the block is shifted in, the block is permuted using a set permutation by the Permutation block, and then fed into the DES Round Computation Block. A set of 3 unique keys are used for the encryption algorithm, and if the data is to be decrypted, the original keys are substituted with their reverse. The controller then enables the round counter to increment, and for each count from 1 to 16, a round of the DES algorithm is performed with a 48B subkey created by the key generator. When the round counter reaches 16, it sets its rollover flag, and the key counter increments. The key counter takes values of 0,1 or 2, and tells the key generator which key to make the subkey out of. Each value of the key counter utilizes 16 subkeys, for a total of 48 rounds through the DES Round Computations block. After the key counter asserts key-rollover, the Controller indicates the Permutation block to invert its original permutation and take in the result from

des-curr. This value becomes block-out, and becomes shifted out via the 64B to FIFO shift register until the FIFO indicates that all values are on it via the full flag.

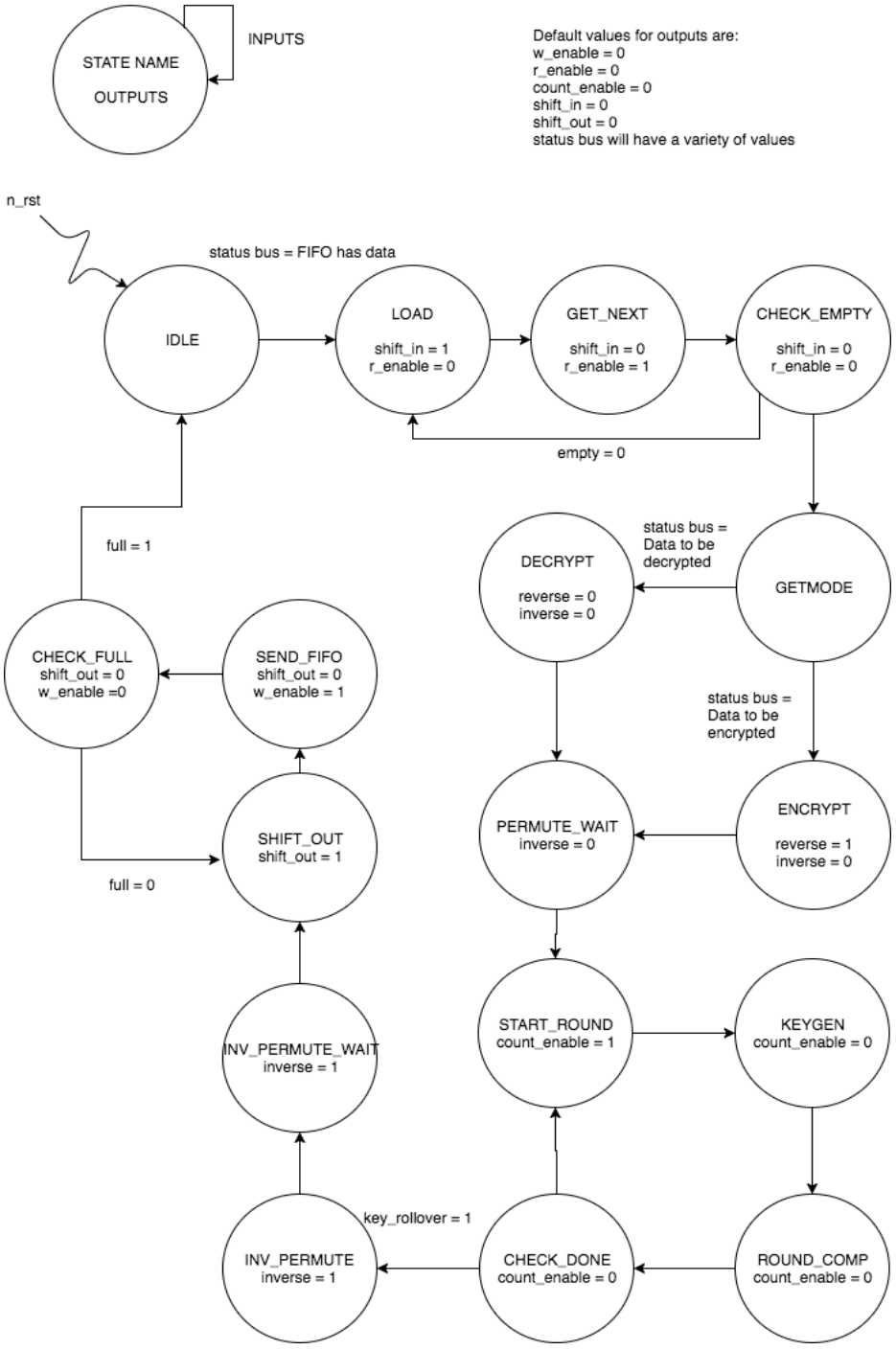

Figure 30. Encryptor/Decryptor Core Controller State Transition Diagram

Below is an overall area estimation table for each of the blocks in the previous Top Level Core Diagram.

Name of Block	Area Estimation
FIFO to 64B shift register	153,600
64B to FIFO shift register	153,600
DES Round Computations	602,250
3-DES Controller	37,000
Key Counter	12,300
Round Counter	30,750
Key Generator	408,000
Total Encryptor/Decryptor Core	1,387,500

Figure 31. Area estimation block for encryptor/decryptor core

A breakdown of the area for each block will be provided in that block's corresponding sub-section. The individual blocks along with their timing, RTL diagrams, and area estimates are provided below for better understanding:

FIFO to 64B Shift Register

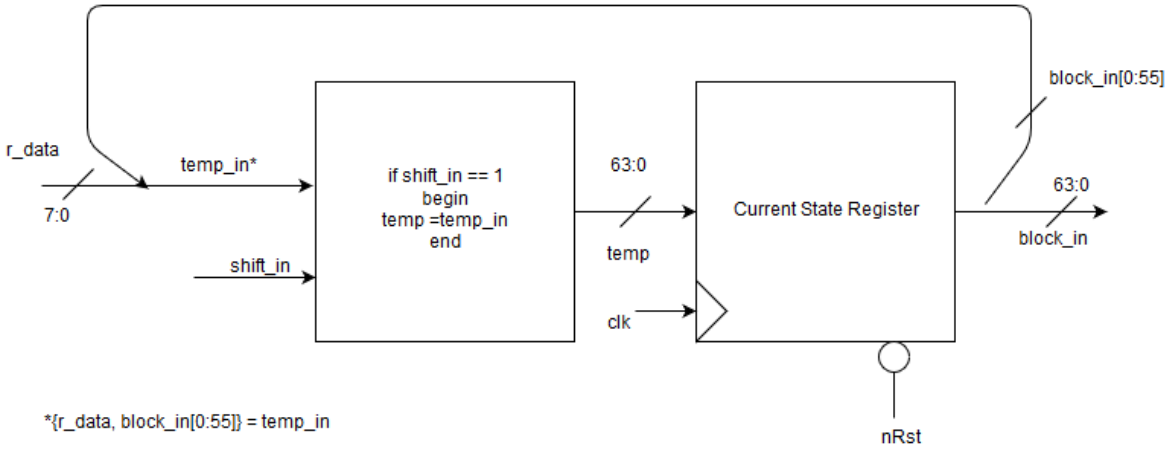

Figure 32. FIFO to 64B RTL Diagram

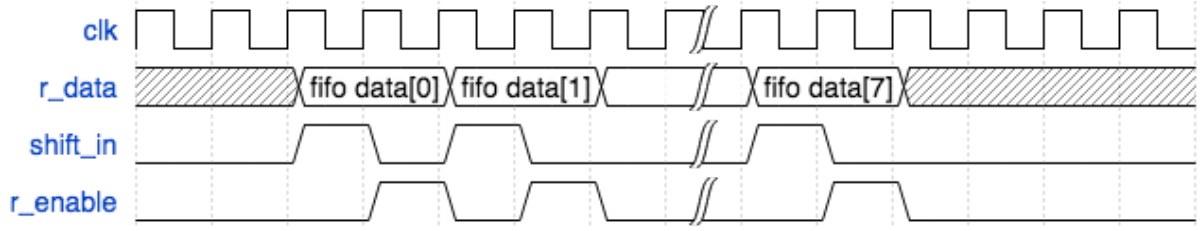

Figure 33. FIFO to 64B Timing Diagram

As is seen in the above waveform diagram and RTL diagram, this shift register will read in data from the FIFO in 8 bit segments, and the controller will prompt the FIFO for a new value after the value is shifted in. The signals shift_in and r_enable will be controlled by the core's controller and no longer be asserted

after 8 bytes are shifted in. The area for this block is estimated to be 153,600 μm^2 and this comes from the 64 registers with reset that are necessary to implement a shift register, as well as the 8 muxes necessary to implement the shifting.

64B to FIFO Shift Register

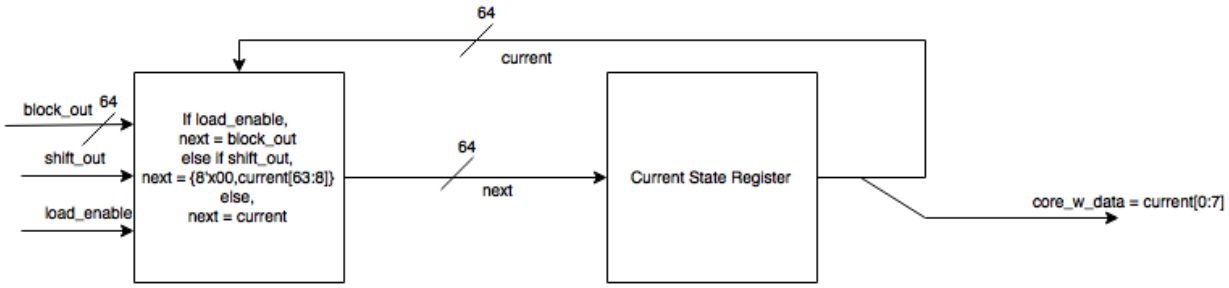

Figure 34. 64B to FIFO RTL Diagram

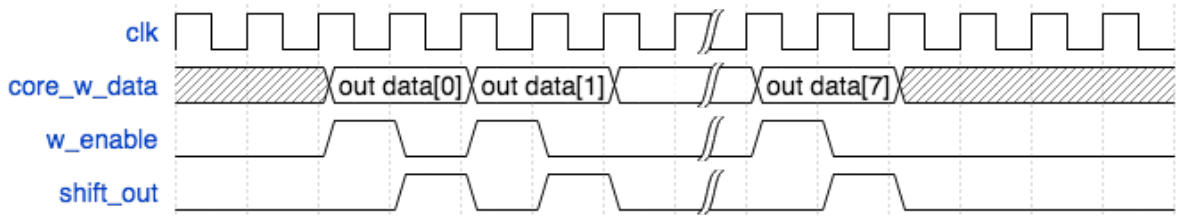

Figure 35. Waveform of nominal 64B to FIFO shift register

The above Shift register is similar to the previous one, except that it shifts a 64B value from the rest of the core, via block_out and the load_enable signal, to the FIFO in 8 byte segments. Core_w_data holds the 8B data segment to be sent to the FIFO from the shift register. It is committed to the register with the w_enable signal from the controller, and the 8B value on the register to be output is changed via an assertion of the shift_out signal from the controller. The area for this block is 153,600 μm^2. This estimate comes from the 64 registers with reset necessary to hold the data, and the muxes necessary to facilitate the shifting and loading of the data.

Permutation

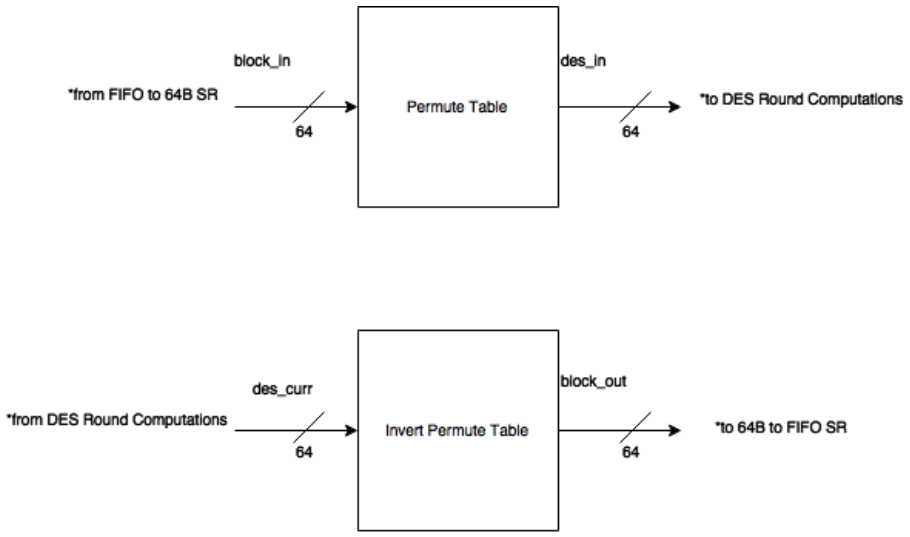

Figure 36. Permutation RTL Diagram

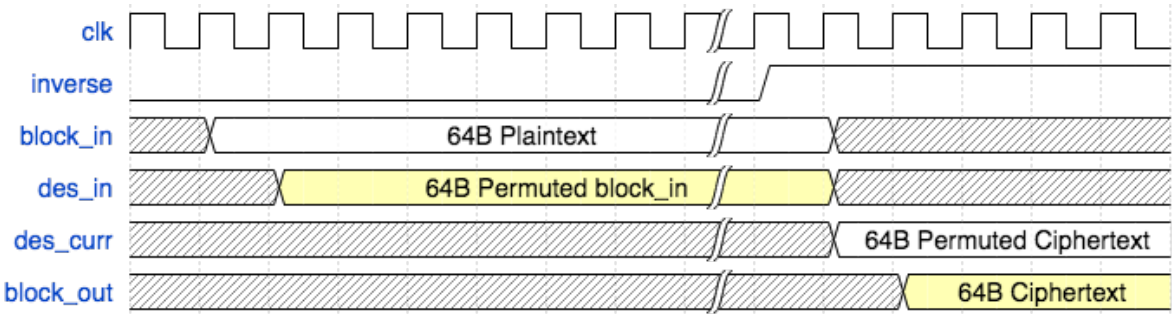

Figure 37. Waveform of nominal permutation

The Permutation block generates a fixed permutation of the 64B data on the block_in bus and outputs it on the des_in bus to be fed into the DES computation. At the end of the computation, the data on des_curr is fed into the inverse of the initial permutation and output onto block_out to be fed into the 64B to FIFO Shift Register. This block actually has no added area, as it is just a fixed rearranging of wires with no registers or combinational logic.

DES Round Computations

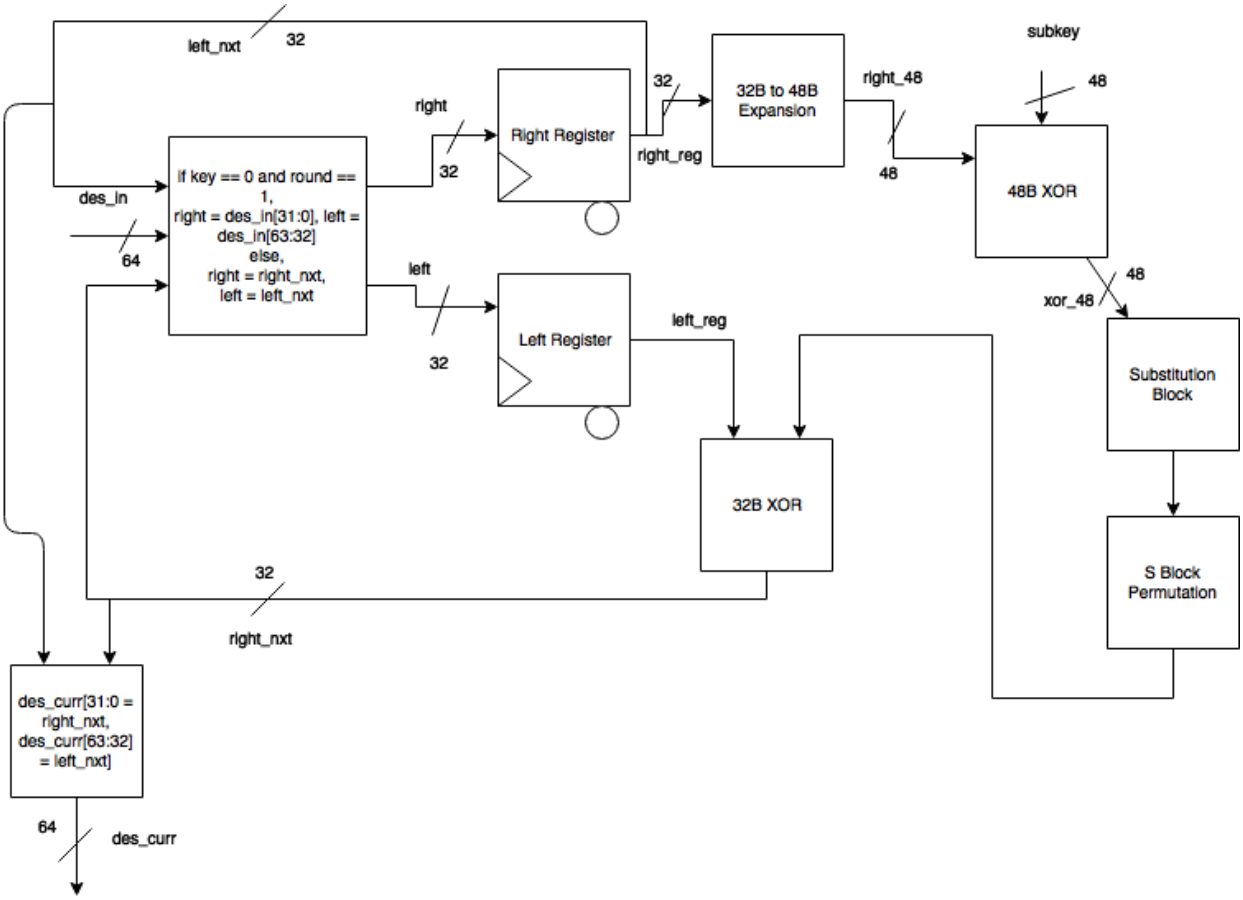

Figure 38. DESRound Computations RTL Diagram

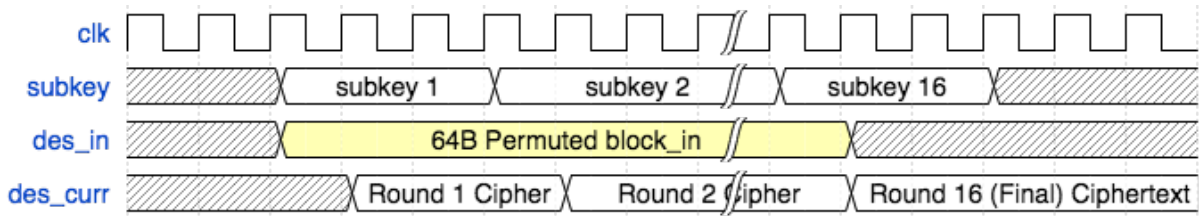

Figure 39. Nominal waveform of DESRound Computations

The DES Round Computations block takes in data from des_in and performs the rounds of the DES algorithm with the subkeys provided from the Key Generator. After 16 rounds, the key will be changed, and after 3 keys are used, the correct encrypted or decrypted value will be on des_curr, and will then be fed into the inverse permutation. The total area for the block is estimated to be 602,250 μm^2. This estimation comes from the sum of all the combinational and sequential logic necessary in this circuit. There are 64 registers with reset needed to store the right and left 32B parts during each round. The block right before the registers requires 2 32B muxes to choose between des_in and left_nxt and right_nxt. The 32B to 48B expansion requires no extra logic, as it is just the duplication of 16 fixed selected bits. The 48B XOR requires 48 2B XOR gates to implement. The Substitution Block requires 64B of SRAM for each 6B to 4B substitution and there are 8 substitutions, so that is 512B of SRAM. Each substitution also

requires 63 muxes, so that is 504 muxes total. Additionally there is a 32B XOR as well, which requires an additional 32 XOR gates. This is what makes up the total block area for this block.

Key Generator

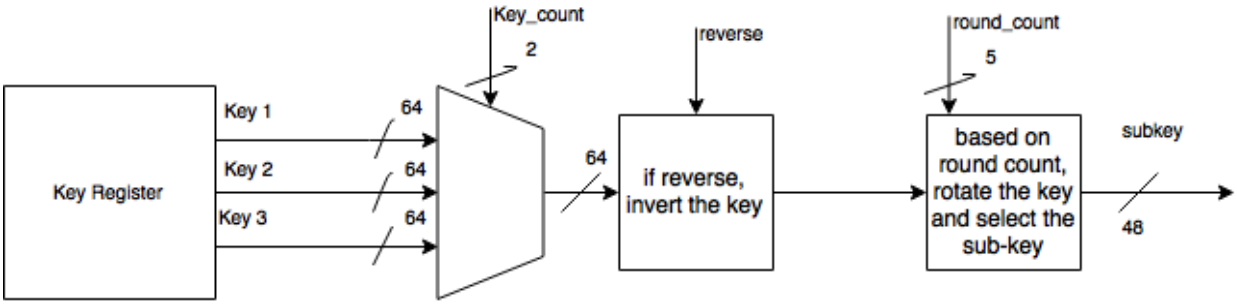

Figure 40. Key Generator RTL Diagram

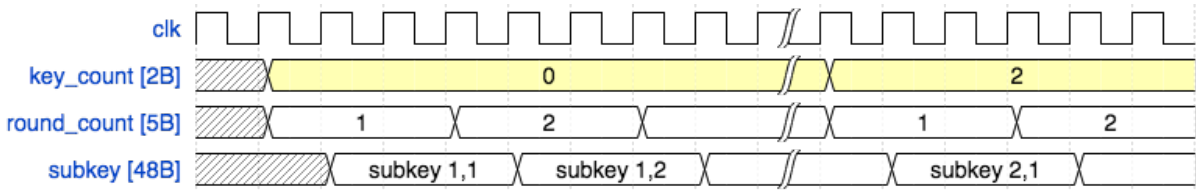

Figure 41. Nominal waveform for key generator

The Key Generator will generate the subkeys according to a specific pattern laid out in the algorithm. Additionally, if the core is set to decrypt something, the keys will be inverted and then subkeys will be generated from the inverted keys, and using these keys with the same hardware will decrypt the data. The key-count and round_count come from the two attached counters, and instruct the key generator block which key to use and which subkey to create from that key. The total area estimated for this block is 408,000 μm^2. This area comes from the need to store 192B of the keys in registers without a reset, a 3-way 64B mux, a 2 way 64B mux to invert the key potentially, and a 2 or 1 bit barrel rotator to select the sub key and rotate it for each specific round.

Round Counter and Key Counter

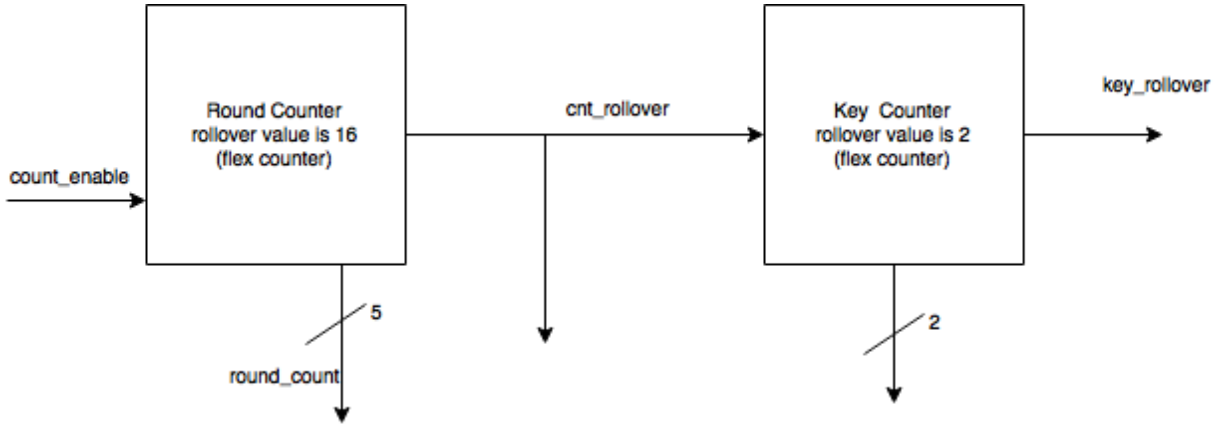

Figure 42. Round Counter and Key Counter RTL Diagram

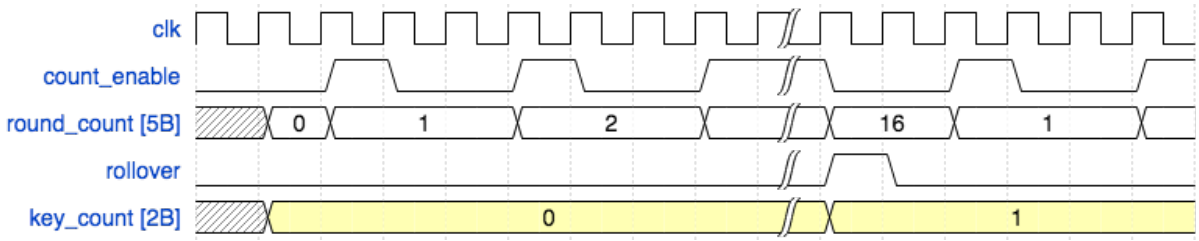

Figure 43. Nominal waveform of round counter and key counter

The two counters in the core function to count the number of rounds that the DES algorithm has run through, while also instructing the Key Generator which subkey to use for the computation round. The round counter counts from 1 to 16 and is triggered by the count enable signal from the core controller. The key_count counts from 0 to 2 and it indicates which key to use to generate the subkeys. It also indicates via the key_rollover value (not pictured) when the last encryption/decryption cycle is done and the data is ready to be inverse-permuted before transmission. The total area for the Round Counter is 30,750 μm^2 , and the total area for the Key Counter is 12,300 μm^2. The area for the Round Counter comes from the 5B necessary to hold the current count in 5 registers with reset, as well as the logic to implement a 5B adder to increment the count depending on the enable signal. The area for the Key Counter comes from the 2 registers with reset necessary to hold the current key count up to key 3, and the logic for a 2B adder to increment the count.

2.5.3 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)
DES Round Current Register	0.1	Permutation, 96 bit xor, substitution block	2.4	DES Round Current Register	0.2	2.7	8.3
USB Transmitter	0.1	Next State Logic	6.7	FIFO Register	0.2	7	8.3
USB Receiver	0.1	Next State Logic	2.39	RCVING Register	0.2	2.69	8.3
Key Generator Key Register	0.1	Rotational Logic	1.4	Sub Key Register	0.2	1.7	8.3
Status Bus Register	0.1	Next State Logic	2.9	Status Bus Register	0.2	3.2	8.3

Figure 44. Timing analysis chart for critical paths

2.5.3.1 DES Round Computations

The estimated Total Path Delay for a round of DES computation starts with the propagation delay from the left and right registers. The 2 xor blocks make for a total of 2 levels of xor gates to go through, adding ~0.4ns of delay. The substitution blocks require 6 levels of muxes for the data in the SRAM to go through and stabilize in the look up tables, which adds another ~1.2ns. The delay for all of the combinational blocks comes to ~2.4ns after multiplying the gate delay by 1.5 to account for wiring delay as well. This is added to the setup time of ~0.2ns for the left and right registers to total ~2.7ns of delay for a round of DES Computation.

2.5.3.2 USB Transmitter

Using Lab 7's synthesis files of the USB Transmitter, we were able to estimate a critical path of 7 ns which is below our target clock period of 8.3 ns. This path goes through the next state logic which is estimated to take 6.7 ns and ending with the FIFO register which will take approximately 0.2 ns. Using the longest start to end path, we are cautiously estimating that the critical path will be around 7 ns which is still below our target.

2.5.3.3 USB Receiver

Using Lab 6's synthesis files of the USB Receiver, we were able to estimate a critical path of 2.69 ns which is well below our target clock period of 8.3 ns. This path goes through the next state logic which is estimated to take 2.39 ns and ending with the receiving register which will take approximately 0.2 ns. Using the longest start to end path, we are cautiously estimating that the critical path will be around 2.7 ns which is still below our target.

2.5.3.4 Key Generator & Key Register

The 0.1 ns propagation delay is from where the key is being stored which will then have its bits rotated either 1 or 2 places via a barrel shifter. This will then be saved in a sub key register as the next subkey to be used. The barrel shifting is estimated to take 1.4 ns which is rotational logic or more plainly combinational logic. The barrel shifting uses 2 of mux, and a 2 bit and 1 bit shifter in order to shift our key with no bits lost. This sub-block is estimated to take 1.7 ns which falls below our target clock period of 8.3 ns.

2.5.3.5 Mode of Operation Bus Register

When processing the status bus register value, the first delay involves the 0.1ns propagation delay of actually storing the value of the variable. Using the synthesis files from Lab 6 as a reference, the path from the output of the register to the output of the next state logic block is estimated to be about 2.9ns which, in terms of the MCU functionality alone, is relatively low compared to the target system clock. Lastly, a time of 0.2ns must also be added onto this path to account for the setup time of the status bus register. When summed to a total, the critical path of the MCU system design is about 3.2ns which is indeed less than the target system clock and thus will not cause any timing hazards in the system.

2.6 Project Timeline and Division of Tasks

Week of March 19th - Design Review Preparation
Week of March 26th - Write Verilog Code for Each Block of the Design and Give Design Review
Week of April 2nd - Write Verilog Code for Each Block of the Design (cont.)
Week of April 9th - Interconnect Functional Blocks to Allow Interactivity with Each Module
Week of April 16th - Write Verilog Code for a Comprehensive Test Bench of the Design
Week of April 23rd - Present Technical Functionality of the Total Design
Week of April 30th - Give Final Presentation of Overall Functionality and Design of Project

Main Control Unit - Thomas Golden
USB Tx/Rx FIFO - Neil Adi
3-DES Encryptor/Decryptor Core - Jon Reitz
USB Data Reader/Writer Interface - Kendrick Lau
Comprehensive Test Bench for Encryption - Thomas Golden, Neil Adi
Comprehensive Test Bench for Decryption - Jon Reitz, Kendrick Lau

2.7 Success Criteria

2.7.1 Fixed Criteria

1. (2 points) 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. (4 points) Entire design synthesizes completely, without any inferred latches, timing arcs, and, sensitivity list warnings.
3. (2 points) 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.
4. (2 points) A complete IC layout is produced that passes all geometry and connectivity checks.
5. (2 points) The entire design complies with targets for area, pin count, throughput (if applicable), and clock rate. The final targets for these parameters will be determined by course staff based on your design review. Failure to reach any of the targets will result a score of 1 out of 2 provided that you are within 50% on area, 10% on pin count, and 25% on throughput. Doing worse in any category will result in a score of 0 out of 2.

2.7.2 Design Specific Success Criteria

1. (2 points) Demonstrate by simulation of Verilog test benches that the complete design is able to successfully encrypt file.
2. (2 points) Demonstrate by simulation of Verilog test benches that the complete design is able to successfully decrypt file.
3. (1 points) Demonstrate by simulation of Verilog test benches that the complete design is able to successfully transfer encrypted or decrypted files back to original device.
4. (2 points) Demonstrate by simulation of Verilog test benches that the design can successfully receive USB data.
5. (1 points) Demonstrate by simulation of Verilog test benches that the design can successfully transmit USB data.

References Cited

[1] "USB in a NutShell - Chapter 1 - Introduction", Beyondlogic.org, 2018. [Online]. Available: http://www.beyondlogic.org/usbnutshell/usb1.shtml. [Accessed: 11- Feb- 2018].

[2]"Triple DES", www.tutorialspoint.com, 2018. [Online]. Available: https://www.tutorialspoint.com/cryptography/triple_des.htm. [Accessed: 11- Feb- 2018].

[3] *DATA ENCRYPTION STANDARD (DES)*. Gaithersburg, MD: U.S. DEPARTMENT OF COMMERCE, 2018, pp. 1-22.