Triple DES Encryption and Decryption

Final Report

ECE 337

Seth Bontrager
Anthony Kang
Eric Murphy
Isaac Sheeley

Thursday (11:30 - 2:20)

Due: 05/07/15

TA: Utpal Mahanta

Executive Summary

Our team has designed and simulated a hardware implementation of the Triple DES (Data Encryption Standard) algorithm. Sending and receiving of encrypted data is an integral part of nearly all forms and commerce and internet communications. Our design of Triple DES is important because many financial institutions still use the encryption standard, therefore developing a more efficient or faster implementation could be very beneficial. Our design is unique in the sense that we have both optimized for speed of encryption and decryptions, while reducing the size by creating a custom hybrid pipeline that takes advantage of the clock cycles it takes to offload the data from a AHB-lite bus, which we will be interfacing to. The 3DES algorithm is appropriate for ASIC because the creators of the DES algorithm developed it with a hardware implementation in mind, and triple DES is simple 3 instantiations of regular DES. This was done because regular DES was too susceptible to brute force attacks, whereas it is still computationally infeasible to break Triple DES by brute force methods. Because of this fact, Triple DES can successfully be implemented in a hardware design to allow for the encryption and decryption of data without having to consume resources that a software based implementation would otherwise devour, and provide a very safe and secure means of data encryption. The remainder of this proposal goes over the details of implementing Triple DES, such as the design specifications, a system usage diagram that shows a very high-level block diagram of the overall algorithm along with a diagram for one round of processing. It also details the operation characteristics that describes the functions that will be performed by our chip, along with the specifications of the hybrid pipelining technique, and identifies the general type of commercial part to which our final chip would be connected.

Design Specifications

Interfacing Specifications

Figure 1: System Level Diagram

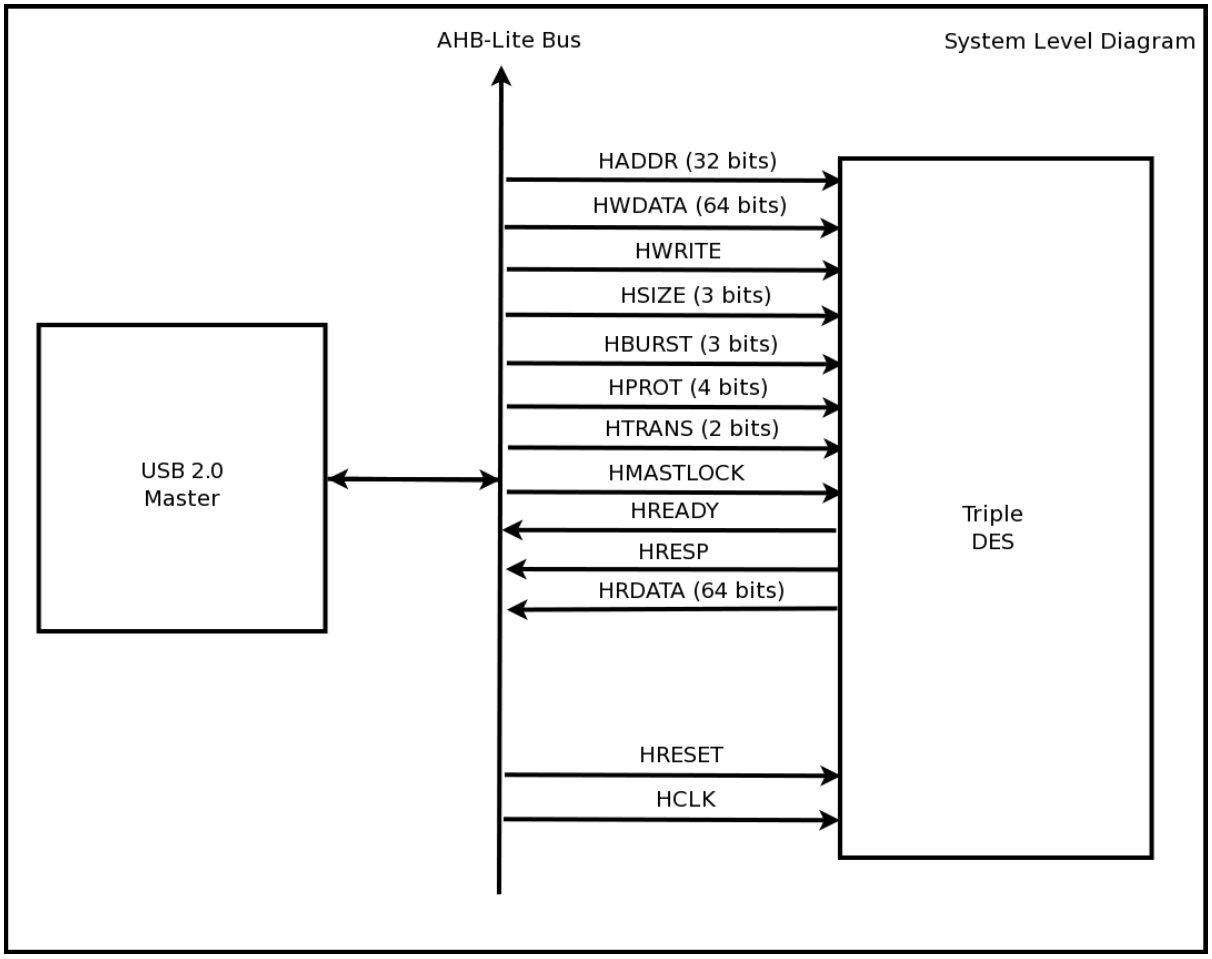


Figure 1 shows our System Level Diagram. All of the signals coming in to, and leaving, our design are signals that are part of the AHB-Lite specification. It also shows a USB 2.0 Master connected to the bus. This master is used to initiate transfers and pull encrypted / decrypted data from the bus. Other peripherals could connect to this bus as long as they follow the AHB-Lite specification and the protocols defined by our Triple DES for sending and receiving data.

Implemented Standards and Protocols:

- Data Encryption Standard (DES)
- AMBA 3 AHB-Lite Protocol

Required Interface Standards and Protocols:

- AMBA 3 AHB-Lite Protocol

    - Incorporates many of the simplifications used by the Cortex M0 (Data buses are 64 bits as opposed to 32 bits though)
    - HSIZE - must be 3’b011
    - HPROT - must be 4’b0001
    - HBURST - must be 3’b000
    - HTRANS - must be 2’b00 or 2’b10 (Since there is no burst mode)
    - HCLK - must be 5 ns or greater (Critical path is currently 4.76 ns)
    - Data must be send in the following order: encryption type, key 1, key 2, key 3, data (In order to make sure the design functions properly)
    - Data is available every eighth clock cycle after it is initially ready
    - When sending and receiving data in an eight clock cycle span on time, data should be send on the first clock cycle, followed by six clock cycles of waiting, followed by data being read on the final clock cycle
- DES
    - The initial and final permutations at the beginning and end of rounds 1 and 16, respectively, have been omitted
    - Takes 16 total rounds to process the data (Two sets of 8 rounds, where each set cannot be disturbed for 8 clock cycles, lest the data be unknown)

Design Features:

- Encrypts (or decrypts) a chunk of data through DES three times
- Uses a six stage pipeline (each stage requiring eight clock cycles to complete)
- All legitimate data will continue to be encrypted after the enable signal has been driven low
- Two slave devices for AHB-Lite are used to properly encompass all of the 32-bit address space
- Interfaces with an AHB-Lite Bus to provide for possible bus interfacing with various devices
- Operates at a speed that meets (and even exceeds) the USB 2.0 specification

Operational Characteristics

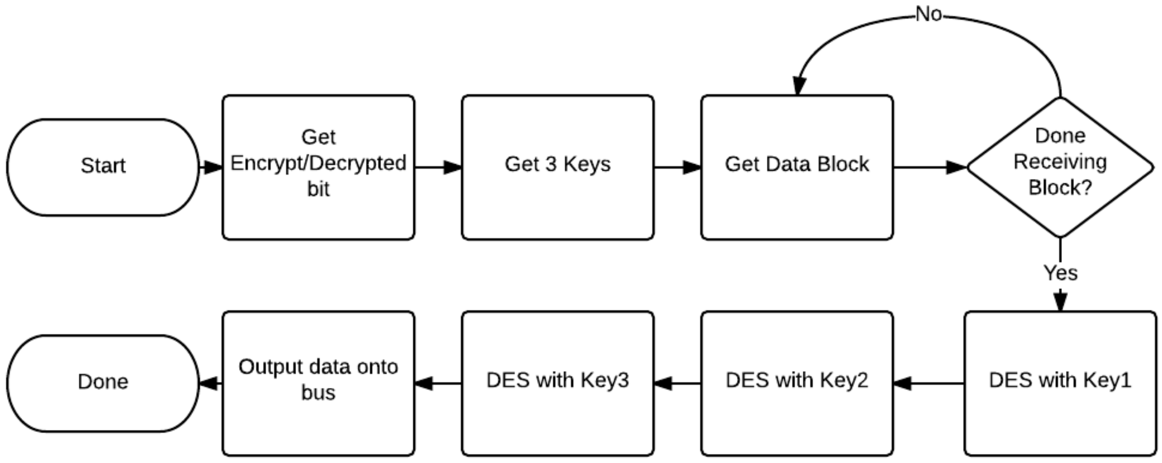
High Level Block: System.sv

This block encompasses the inputs, outputs, and the functionality of our design.

Port Descriptions:

Signal Name	Type In/Out/Bidir.	Number of bits	Description
HCLK	Input	1	Bus clock signal
HMASTLOCK	Input	1	Used by the master to signal a locked transaction
HRESET	Input	1	Bus reset signal
HWRITE	Input	1	Used by the master to signal a read or write
HTRANS	Input	2	Used by the master to classify the type of transfer
HBURST	Input	3	Used by the master to define the burst type
HSIZE	Input	3	Used by the master to indicate the size of the transfer
HPROT	Input	4	Used by the master to implement bus protections
HADDR	Input	32	System bus address
HWDATA	Input	64	Write data bus
HREADY	Output	1	Used by the slave to signal a complete transaction
HRESP	Output	1	Used by the slave to indicate the transfer status
HRDATA	Output	64	Read data bus

Sequence of Operations:



After data is sent to the triple DES block and the enable signal is sent as well, the pipeline will begin to fill up. After 52 clock cycles, the pipeline is full, a new 64 bit block of either encrypted or decrypted data is available every 8 cycles and the processing done signal is asserted, allowing it to be output on the AHB-Lite bus. This pipeline will continue until the enable signal is set low, and 52 clock cycles afterwards, the last 64 bit block of processed data will be available.

The above sequence abstracts away some of the work done by the AHB-Lite Slave. When data is being sent to our chip, the AHB-Lite Slave is picking up signals from the AHB-Lite Bus. These signals determine whether data should be read or written, along with addresses that determine where data needs to be stored or sent. Initially, the encryption type (encryption or decryption), the keys (three in total), and the first chunk of data are send on the bus. Once these values are picked up, they are sent to the first DES block. Once this block has finished processing the first chunk of data, another chunk of data is picked up from the bus. This operation continues until data has been successfully processed. Once data is ready to be sent out, the master signals that it is going to grab the data, and the slave prepares the read bus accordingly. The master later sends the data block to the slave (at the same rate as before so that data can be seamlessly processed). This operation continues until the master quits sending the slave data. Once this stage is reached, data is output on the bus until all the data has been processed. After all the data has been processed, the slave idles and wait for the master to restart transaction (exactly how the master started the transactions in the beginning of this paragraph).

Requirements

The use of Triple DES in encrypted network communication requires high speed to minimize the time it takes to encrypt the information and then decrypt it at the other end. The speed at which information can be sent is very critical in many industries today, and the methods used to encrypt the information must be fast in order to not create a bottleneck in the transmission of information. The primary optimization objective of our project will be to maximize speed. We will optimize the design for speed by implementing certain functional blocks at a gate level and pipelining several steps in the algorithm so the next block is being encrypted directly behind the previous one. Optimizing our design for area will be a secondary objective. Our target for the area was initially, 1.5mm x 1.5mm, however after changing our I/O protocol, we have increased it to be 4.0 mm x 4.0 mm. The pin count for our design will be 180 pins. Since data is going to be input and output in a parallel manner on the AHB-Lite, more pins are needed than in other kinds of designs, such as those utilizing something simple like a single pin SPI protocol which we originally intended to use.

Final Design

Design Architecture

Figure 2: Overall System Architecture

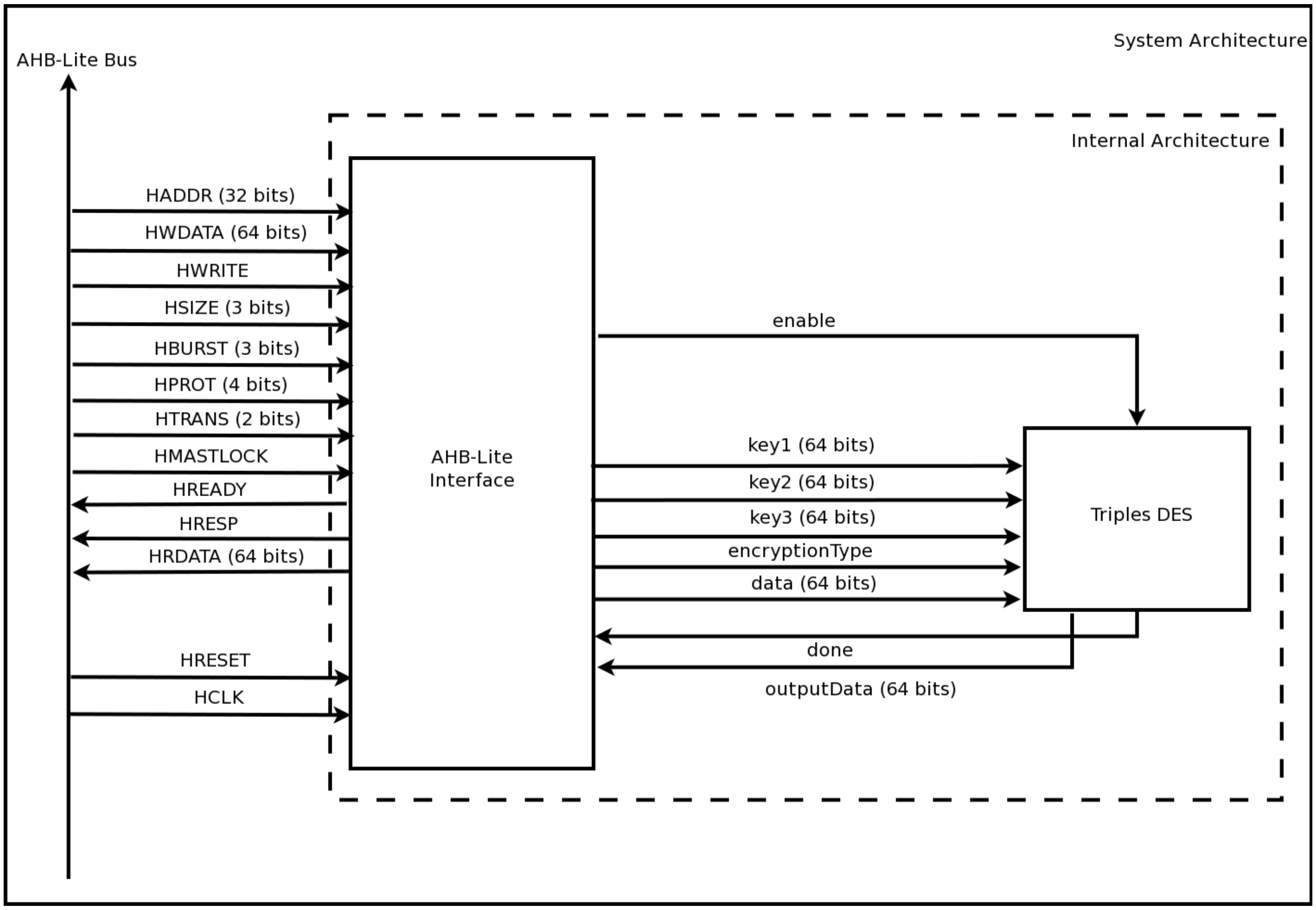
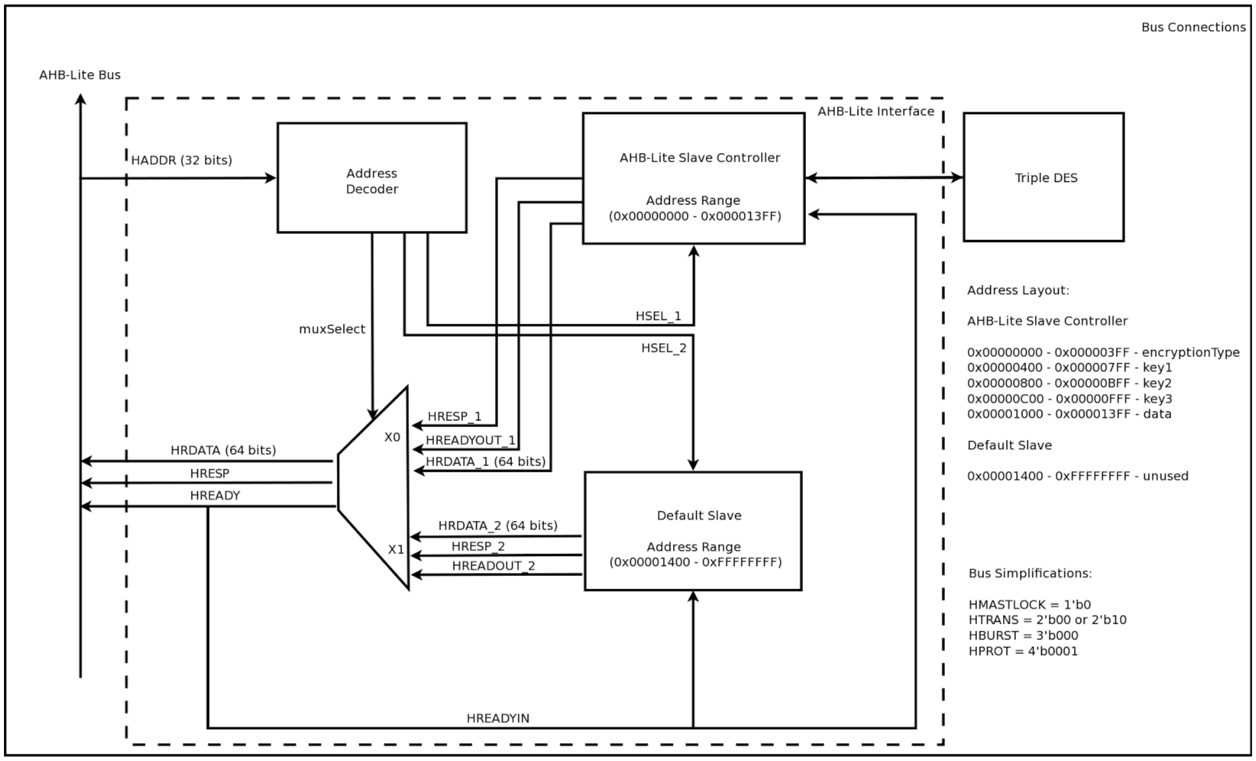


Figure 2 shows an overall system architecture. The two main blocks, the Triple DES block, and the AHB-Lite Interface, are shown, along with all of the important signals between the two. The keys and data that are collected by the Interface from the AHB-Lite Bus are sent to the Triple DES block, along with an enable signal and the type of operation (Encryption / Decryption). Inside the AHB-Lite Interface are many components, including slaves, a decoder, a multiplexer, and a memory element (Flip-Flop). The Triple DES block consists of 3 DES blocks that are each used to calculate a signal iteration of DES.

Functional Block Diagrams

Figure 3: Bus Connections



Address Layout:

AHB-Lite Slave Controller

0x00000000 - 0x000003FF - encryptionType
0x00000400 - 0x000007FF - key1
0x00000800 - 0x00000BFF - key2
0x00000C00 - 0x00000FFF - key3
0x00001000 - 0x000013FF - data

Default Slave

0x00001400 - 0xFFFFFFFF - unused

Bus Simplifications:

HMASTLOCK = 1'b0
HTRANS = 2'b00 or 2'b10
HBURST = 3'b000
HPROT = 4'b0001

Figure 3 shows a more detailed view of the AHB-Lite Bus Interface. As shown, the AHB-Lite Bus Interface is made up of a decoder, a multiplexer, and two slaves. The AHB-Lite Slave Controller is used to communicate with our Triple DES block, whereas the Default Slave is used to take of transactions that occur in address ranges outside of the scope of the AHB-Lite Slave Controller. The Address Decoder is used to determine which slave is being selected, and select the correct signals to output from the multiplexer. The multiplexer is used to output the signals that are currently being used. On the left, there is an address layout for our system. Addresses ranging from 0x000000 to 0x000013FF reference components in our AHB-Lite Slave Controller, which in turn are used in our Triple DES block. Addresses ranging from 0x00001400 to 0xFFFFFFFF are unused, and therefore are sent to the Default Slave. Specific ranges for the AHB-Lite Slave Controller, pointing to certain values used in our design, are given on the left-hand side as well.

Figure 4: System Connections between AHB-Lite Interface and the DES Blocks

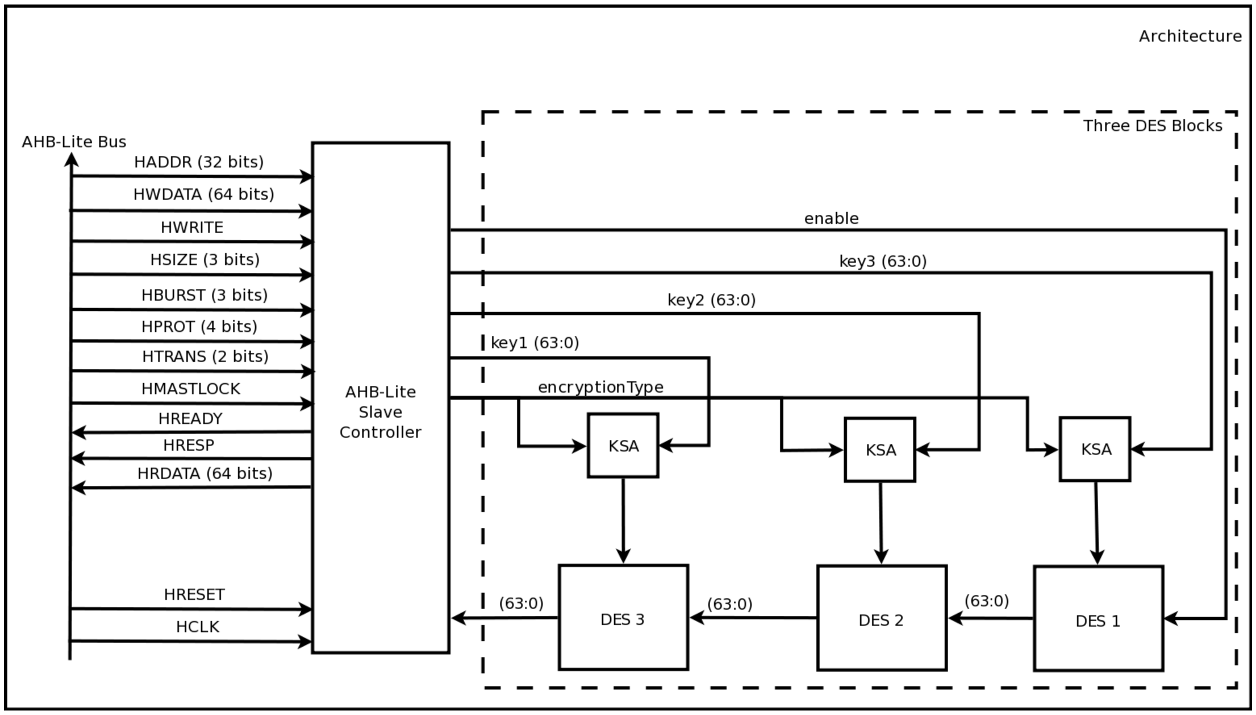


Figure 4 shows the connections between our AHB-Lite Interface and our Triple DES design. Since Figure 3 showed the specific connections for our AHB-Lite Interface, it has now been generalized as a single block. All of the values that were addressed in the AHB-Lite Slave Controller in Figure 3 are shown, along with a few other signals to show state, and output data. The Interface sends all of the necessary information to encrypt / decrypt the data, along with an enable signal, to the three DES blocks. The Key Scheduling Algorithm (KSA) is also shown for each DES Block. Each DES block is made up of two blocks that are used in a combined sequential-pipelined fashion for a tradeoff between space and speed.

Figure 5: Single DES Block Expanded (Pipelined Operation Shown)

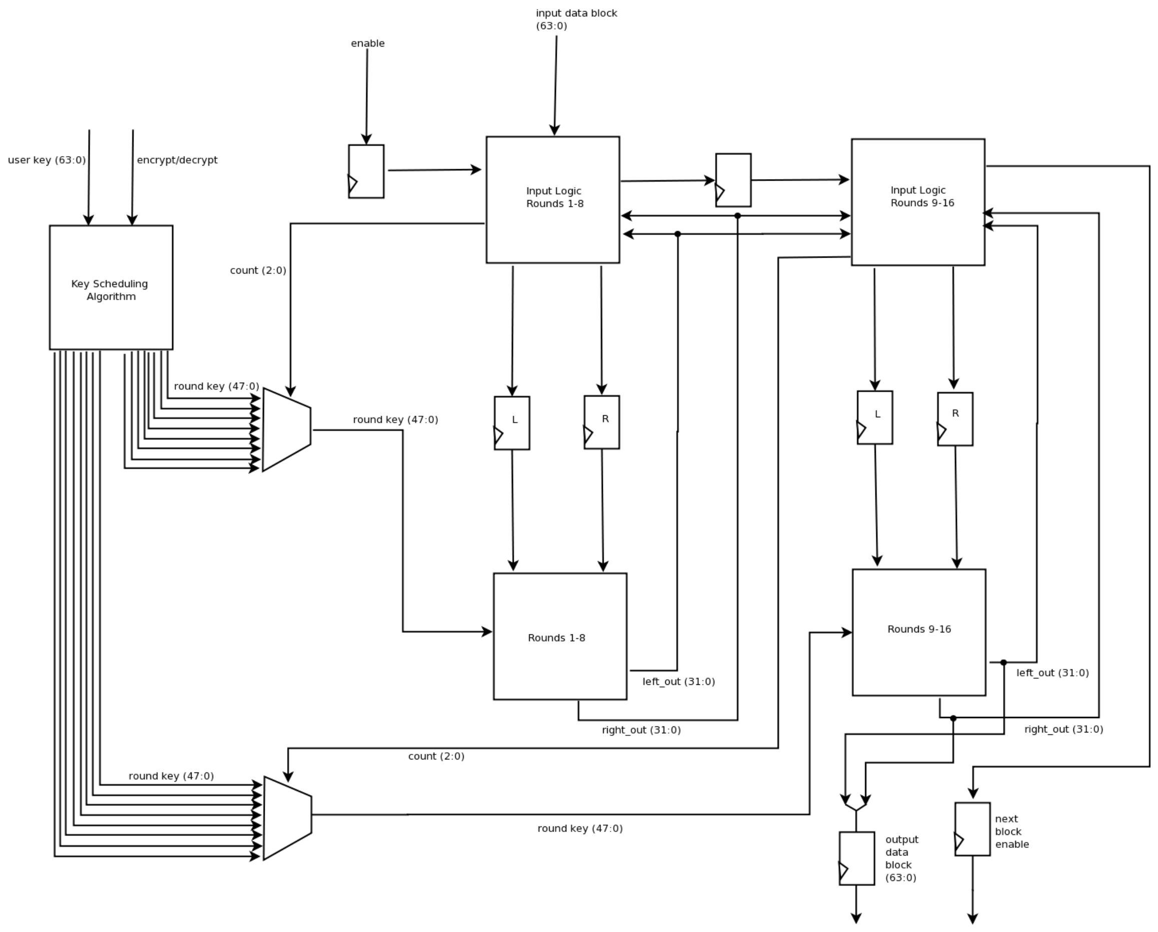


Figure 5 illustrates a single DES block. It also shows how the pipelining works. The block receives the input key, data, enable, and encrypt or decrypt signals. After then enable has been asserted, data will be processed in 64 bit chunks, spending 8 cycles to process rounds 1-8 with purely combinational logic. After 8 cycles, data is passed to rounds 9-16 on the right, and a new 64 bit data block is fed into rounds 1-8. After the next 8 rounds, rounds 9-16 feeds its output to either the next DES block or the I/O controller, and receives its next block from rounds 1-8. This continues on until the enable signal is no longer asserted. As discussed, once our pipeline is completely full, it will output 64 bits of data every 8 cycles to the I/O controller.

Figure 6: One Round of DES

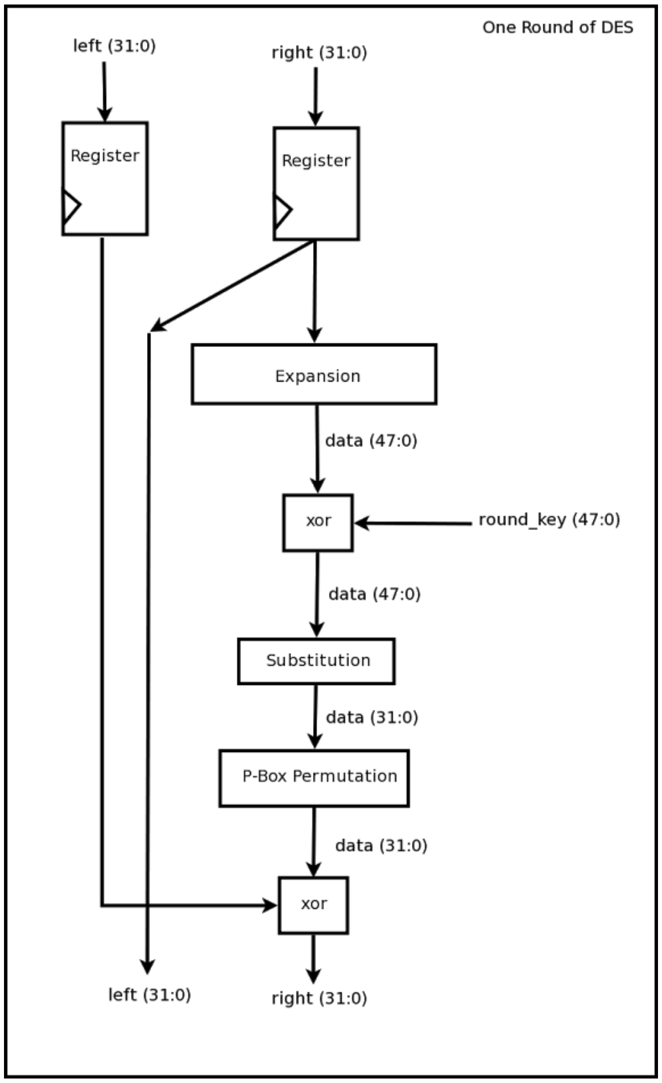


Figure 6 represents one round of DES. There are 16 total rounds of implementation of DES, and each round is accompanied by a round key. Each box in the round represents a specific function that manipulates the bits of the data.

Figure 7: Key Scheduling Algorithm

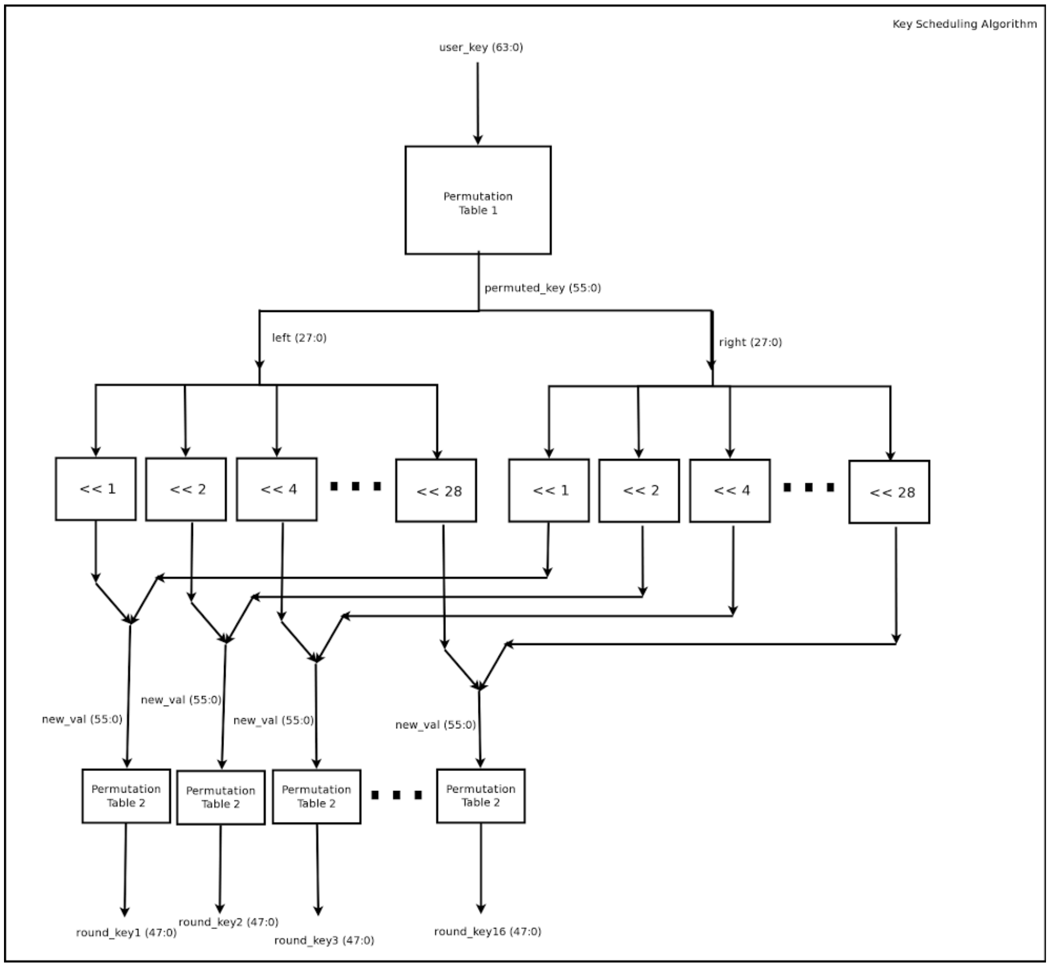


Figure 7 illustrates the Key Scheduling Algorithm. A 64 bit user key is fed into the system where it is then permuted, split into left and right halves, shifted a specific amount of times, and finally permuted again and joined. This process continues until all 16 round keys are generated.

Figure 8: Expansion Permutation

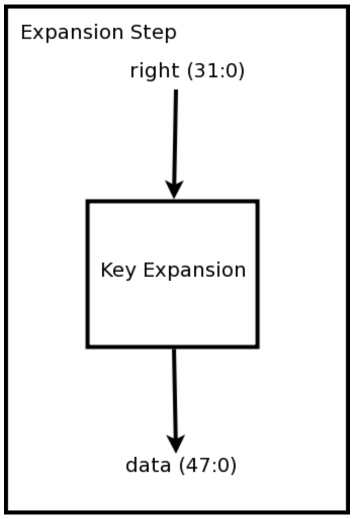


Figure 8 shows a 32 bit block of data being put into an expansion step that splits the data into 4 bit nibbles and prepends and appends bits to expand it to a 48 bit output block

Figure 9: Substitution Step

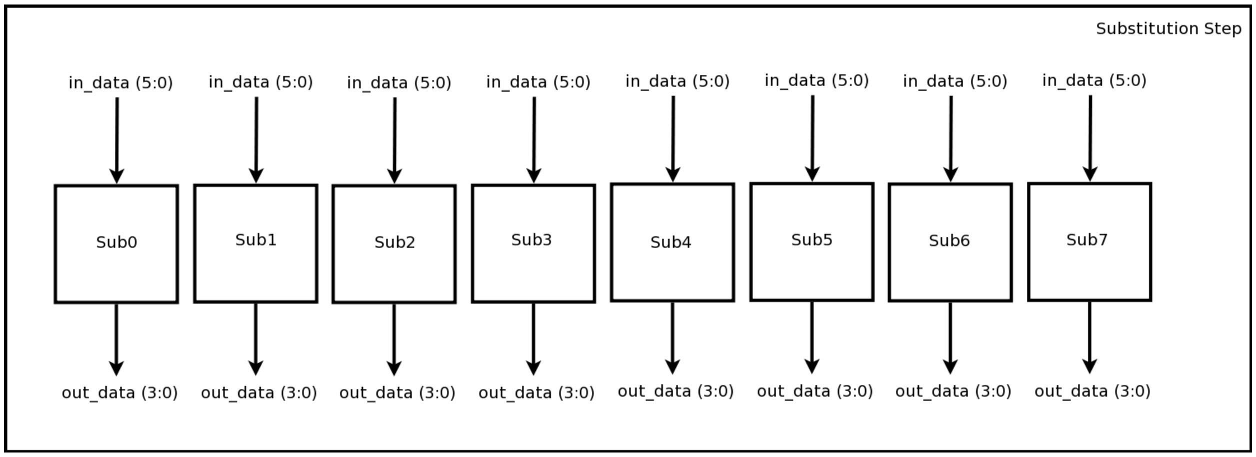


Figure 9 shows the Substitution Step in DES. In the Substitution step, the 48-bit input word is divided into eight 6-bit words and each of those words is fed into a separate S-box. Each of these S-boxes produces a 4-bit output. In each of the S-boxes, there is a 4 x 16 lookup table for an output 4-bit word. The first and last bit of the 6-bit input word are decoded into one of 4 rows and the middle 4 bits decoded into one of the 16 columns for the table lookup.

Standards and Protocols

There were many standards and protocols that our design dealt with. Our design was aimed to be used to interface to a USB 2.0 master. We also implemented the AHB-Lite Protocol, which would be used to interface our design to a master device (in our case a USB 2.0 interface). Lastly, we encrypted and decrypted our data using the Data Encryption Standard (DES).

Our design is intended to be used with a USB 2.0 interface. This interface was chosen because we wanted our design to use an easily available interface. USB 2.0 was specifically chosen because our design only operated at a certain speed, and USB 2.0 looked to be the fastest that our design was capable of. With our design sending our 64 bits of data every 8 clock cycles (each clock cycle having a period of 5 ns), we were sending out data that exceeded the USB 2.0 specification.

For our design, the AHB-Lite Protocol was used. This specific bus protocol was selected because it allows for extremely fast transactions and larger width buses than other protocols, like APB. We also chose AHB-Lite since there are many devices that directly interface with an AHB-Lite bus, so it allows our design to be used with devices other than a USB 2.0 controller (which is what our design is being constructed around). For our AHB-Lite implementation, we followed some of the simplifications taken by ARM's Cortex M0. Since we knew nothing of the AHB-Lite Protocol, we believed that using a simplified subset of AHB-Lite would be a better choice since we would be able to focus on some of the key features of AHB-Lite (like the pipelined addressing) and forgot about some of the more complicated features that weren't useful, or necessary, for our design. Our AHB-Lite Slave did not implement burst transactions (HBURST = 3'b000), it also did not have any transaction protection (HPROT = 4'h1 was used as our protection value). In addition two these two simplifications, our transaction values were also simplified (HTRANS = 2'b00 or 2'b10 since BUSY and SEQUENTIAL transaction are only possible in burst mode). The master was not allowed to lock transactions also (so HMASTLOCK = 1'b0 was the idle master lock signal). Lastly, our HRDTA and HWDATA buses were at a constant size of 64 bits (HSIZE = 3'b011 was set to show this). This bus value differs from the Cortex M0. Whereas we use a 64 bit bus length or reading and writing data, the Cortex M0 uses 32 bit buses. In our AHB-Lite Slave design, there were two slaves that could be addressed. Our first slave was tied to our Triple DES design, while our second slave was a default that slave that was used for addresses that fell out of the range of our design. Addresses that ranged from 0x00000000 to 0x000003FF corresponded to the encryptionType for our design. Addresses ranging from 0x00000400 to 0x000007FF to key1, 0x00000800 to 0x00000BFF to key2, 0x00000C00 to 0x00000FFF to key3, and finally, 0x00001000 to 0x000013FF to the data. These address ranges were chosen because the AHB-Lite Specification (as stated in the AMBA 3 AHB-Lite Protocol v1.0 Specification, in our appendices) requires that each address range must be a minimum of 1K (1024 addresses). All of the values in our design were contained within a range of addresses that was exactly 1K. For the remainder of the addresses (0x00001400 to 0xFFFFFFFF), the Default Slave picked up the signals and output the corresponding signals (as according to the specification). When transactions to the Default Slave were legitimate, HWDATA (or HRDATA, depending on the transaction) would be set to all zeros. To divert these signals to the correct slave, we used an address decoder that selects the corresponding slave that it being talked to. The slaves use this information to send and receive data to / from the AHB-Lite Bus. In addition to this decoder, a multiplexer is used to select the correct set of signals to output on to the bus. Since both of the designs could be potentially selected, both designs send their signals to a multiplexer, which, in turn, sends out the signals that are currently being selected (or were selected in the case of an incomplete transaction). This multiplexer sends out the data, the response, and the ready signal sent by the slave. This ready signal that is send out of the multiplexer (HREADY) is then send to the AHB-Lite Bus and back to each of the slaves to signify that a transaction has been completed.

DES (or in our case, Triple DES) is an encryption standard that was very popular in the 20th century. DES was designed with hardware in mind, and as such we felt like it'd be a great standard to implement in Verilog, a hardware description language. We chose to implement Triple DES since DES is an outdated specification and Triple DES is still in some in many applications. With Triple DES, we followed all of the important aspects of encryption and decryption (all steps can be found in the documentation in the appendices), including the full feistel structure, except for the initial and final permutations. The initial and final permutations and simple bit rearrangements that take place at the beginning and end of the first and final round of DES; respectively. We did not include these permutations because we believe that they don't add much to the encryption (or decryption) process. Since bit are simply being rearranged, little to security is added. Also, we programmed a Triple DES application as part of our Computer Security class where the initial and final permutations weren't included. Since one of our design specific success criteria was to make the output of the script that we had written from our Computer Security class, we weren't able to use these permutations without having different outputs from our script. Due to these many reasons, we did not include these permutations. Lastly, our DES encryption used the Electronic Codebook (ECB) block cipher mode for encryption. The reason we chose this method was that it allowed for the easiest Triple DES implementation. With other chaining modes, complexity would be increased since previously encrypted (or decrypted) values would have to be stored in order to be chained to the new values. This added to the complexity of our design, while also requiring additional area to store values to be used later.

For our design, we took advantage of pipelined operations in our Triple DES encryption. These pipelined operations allowed for multiple rounds of DES to be performed simultaneously. There are a total of 16 rounds in a single DES encryption, meaning that our design had a total of 48 rounds (16 rounds x 3 DES blocks). With these 48 total rounds, there was a possibility for our design to implement a 48 stage pipeline. Whereas a 48 stage pipeline would allow for computation of large amount of data (and increased speed), we opted for fewer stages in our pipeline to reduce the area our chip would take up. Our group implemented a 6 stage pipeline, with each stage encompassing 8 rounds of DES (2 stages of the pipeline per DES block). This pipeline allowed for computation of multiple chunks of data while also keeping the area at a reasonable level.

Timing and Area Budgets

	Critical path constraint from proposal success criteria	Critical path constraint from design budget	Critical path constraint from final synthesis report	Critical path reported by Encounter
Delay (ns)	NA	1.945	4.76	4.76
Start/End	NA	Round n Register to Round n + 1 Register	T0/A0/EncryptionType_reg to T0/T0/DUT_DES3 right_reg_9_16_reg[10]	NA
Comments	In our proposal we did not include critical paths because we were still in the early stages of developing our design and did not know what to aim for	In this stage, we had started developing our pipeline and estimated based on using APB as our new bus protocol instead of SPI.	In this stage, the critical path went up again because there we estimated wrong in the budget and certain operations took longer than expected	Critical path delay did not change from the synthesis report

	Area targets from proposal success criteria	Total area from design budget	Area estimates from final synthesis report	Area reported by Encounter
Area (mm^2)	2.25	21.6	4.81	16.1
Comments	Originally we were planning on using SPI and not pipelining. This led to a very small estimate because we have very few instantiations and very few pins	Area increased here because we started a pipeline technique and changed to using an APB bus protocol instead of SPI	This is significantly smaller because the report does not take the pads into consideration. And since we are using an I/O protocol that is largely parallel, we will have a lot of pads	Here it goes up from the synthesis report and down from the design budget. This is because it does consider that pads, but is still better from our design budget because were were able to optimize the area quite a bit.

Verification

AHB-Lite Interface: Top level block that deals with sending and receiving of data from the bus. This module will contain the IO controllers from our design (Controller 1 and Controller 2). Data is sent in 32-bit chunks, with the keys, data, and encryption type being sent in the initial transaction.

Triple DES: Top level block that deals with the encryption and decryption of our data. This block will take care of the key generation (Keys will be input in the beginning of a transaction), DES blocks (Three total for Triple DES), and pipelining.

Correct Chip Response to Encryption & Decryption Requests

- Shown in Demo: Yes
- DSSC(s) Proved: 3, 4, 5
- Highest Level of Design Module(s) involved:
  - Total Design/Chip
- Test bench Expectations/Requirements:
  - Have temporary data registers to capture encryption results for reuse in decryption request
  - Use random number generation (i.e. Have Professor Johnson choose a random number) for encryption / decryption
  - Emulate encryption request AHB-Lite bus transaction
  - Emulate decryption request AHB-Lite bus transaction
  - Capture the generated keys and compare them to known output from python script
- No pre/post processing is needed
- Use python script to verify results
- Main Verification Test Steps:
  1. Verify that when encryption_type = 1, the encryption process begins
  2. Verify that when encryption_type = 0, the decryption process begin
  3. Verify that when encryption_type = 1, the correct keyset is generated
  4. Verify that when encryption_type = 0, the correct keyset is generated

Correctness of 3DES Encryption (Top-level)

- Shown in Demo: Yes
- DSSC(s) Proved: 1, 2, 3
- Highest Level of Design Module(s) involved:
  - Total Design/Chip
- Test bench Expectations/Requirements:
  - Have several test cases that encrypt a variable number of data blocks
  - Encrypt with several different keys in different test cases
  - Output should be verified using known working python script
- Use handmade Python script as a reference for 3DES correctness
- No pre/post processing is needed

- Main Verification Test Steps:
    1. Use testbench to capture encrypted output
    2. Compare encryption output with output from known correct software implementation

Correctness of 3DES Decryption (Top-level)

- Shown in Demo: Yes
- DSSC(s) Proved: 1, 2, 4
- Highest Level of Design Module(s) involved:
    - Total Design/Chip
- Test bench Expectations/Requirements:
    - Have several test cases that decrypt a variable number of data blocks
    - Decrypt with several different keys in different test cases
    - Output should be verified using known working python script
- Use handmade Python script as a reference for 3DES correctness
- Main Verification Test Steps:
    1. Use testbench to capture decryption output
    2. Compare decryption output with output from known correct software implementation

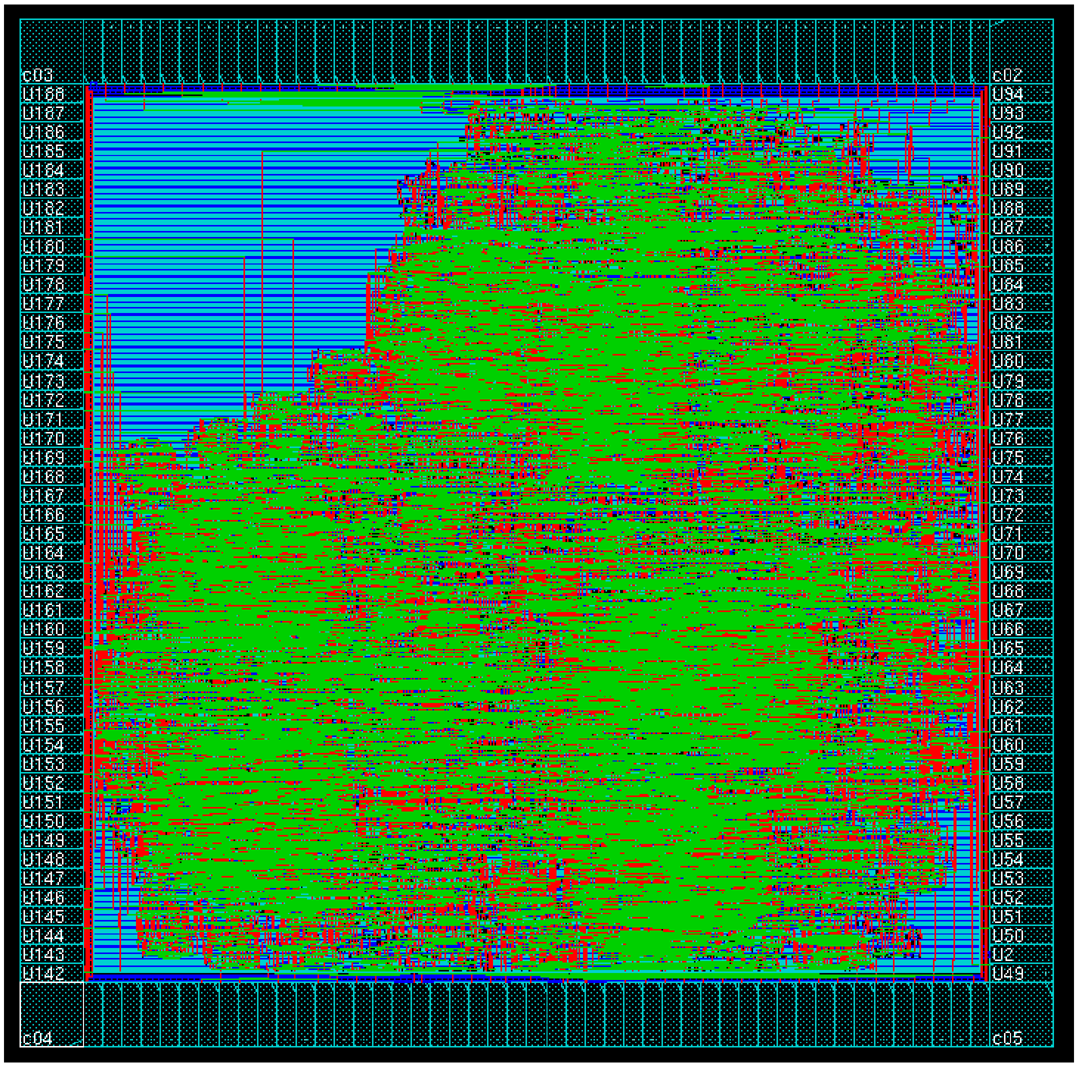
AMBA AHB-Lite Protocol Interfacing (Top-level)

- Shown in Demo: Yes
- DSSC(s) Proved: 5
- Highest Level of Design Module(s) involved:
    - Total Design/Chip
- Test bench Expectations/Requirements:
    - Have test vector of samples of each type of bus transaction and correct chip response information
    - Chip response will checked against standards
- No pre/post processing is needed
- Main Verification Test Steps:
    1. Simulate bus transaction sample
    2. Check chip response against correct response values
    3. Repeat steps 1 and 2 for all the different transaction types

Layout

Aspect Ratio	.995241727507
Dimensions	4.12965 mm x 4.1100 mm
Area	16.9728615 mm2
Row Utilization	Core Utilization: .9996 Cell Utilization: .9996
Number of Gates	13878 (1010 FFs)
Connectivity Checks	Passed, 0 Violations and 0 Warnings
Geometry Checks	Passed, 0 Violations and 0 Warnings

Figure 10: Layout of our Crip Design from Encounter (With No I/O Gaps)



Results

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 pts) **(COMPLETE)**

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

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 pts) **(COMPLETE)**

4. A complete IC layout is produced that passes all geometry and connectivity checks (2 pts) **(COMPLETE)**

5. 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. (2 pts)
    a. Area: 3.5mm x 3.5mm
    b. Pin Count: 115 (112 for the System Architecture + power, ground, clock) ******(Pin count has changed to 180 due to a change in protocol used, we originally estimated needing 181 for using AHB-Lite but it turned out we only needed 180)
    c. Clock Frequency: 200 MHz

    (COMPLETE)

Design Specific Success Criteria

1. Demonstrate by utilizing a known, working python script that the output of the design both encrypts and decrypts according to the 3DES algorithm (1 pt) **(COMPLETE)**

2. Demonstrate by simulation of verilog test benches that the complete design is able to utilize pipelining (1 pt) **(COMPLETE)**

3. Demonstrate by simulation of verilog test benches that the complete design is able to successfully implement 3DES encryption (2 pts) **(COMPLETE)**

4. Demonstrate by simulation of verilog test benches that the complete design is able to successfully implement 3DES decryption (2 pts) **(COMPLETE)**

5. Demonstrate by simulation of verilog test benches that the complete design is able to implement an IO controller for the AHB-lite (2 pts) **(COMPLETE)**

Appendix A - Data Sheets and Guide to Design Data

Account and directory where all of the files are located:	mg112/ece337/Project
Top level structural VERILOG code	source/System.sv
AHB-Lite Slave	source/AHBBLiteSlaveControler.sv
DES Block	source/DES_block.sv
DES Round	source/DES_round_wrapper.sv
Decoder	source/Decoder.sv
Default Slave	source/DefaultSlave.sv
Permutation Box	source/PBox.sv
Expansion Box	source/expansion.sv
Flex Counter	source/flex_counter.sv
Multiplexer	source/Multiplexer.sv
Round Key	source/roundkey.sv
Substitution Box 1	source/s_box1.sv
Substitution Box 2	source/s_box2.sv
Substitution Box 3	source/s_box3.sv
Substitution Box 4	source/s_box4.sv
Substitution Box 5	source/s_box5.sv
Substitution Box 6	source/s_box6.sv
Substitution Box 7	source/s_box7.sv
Substitution Box 8	source/s_box8.sv
Substitution Box Wrapper	source/s_box_wrapper.sv
Bus Connections	source/TopLevel.sv

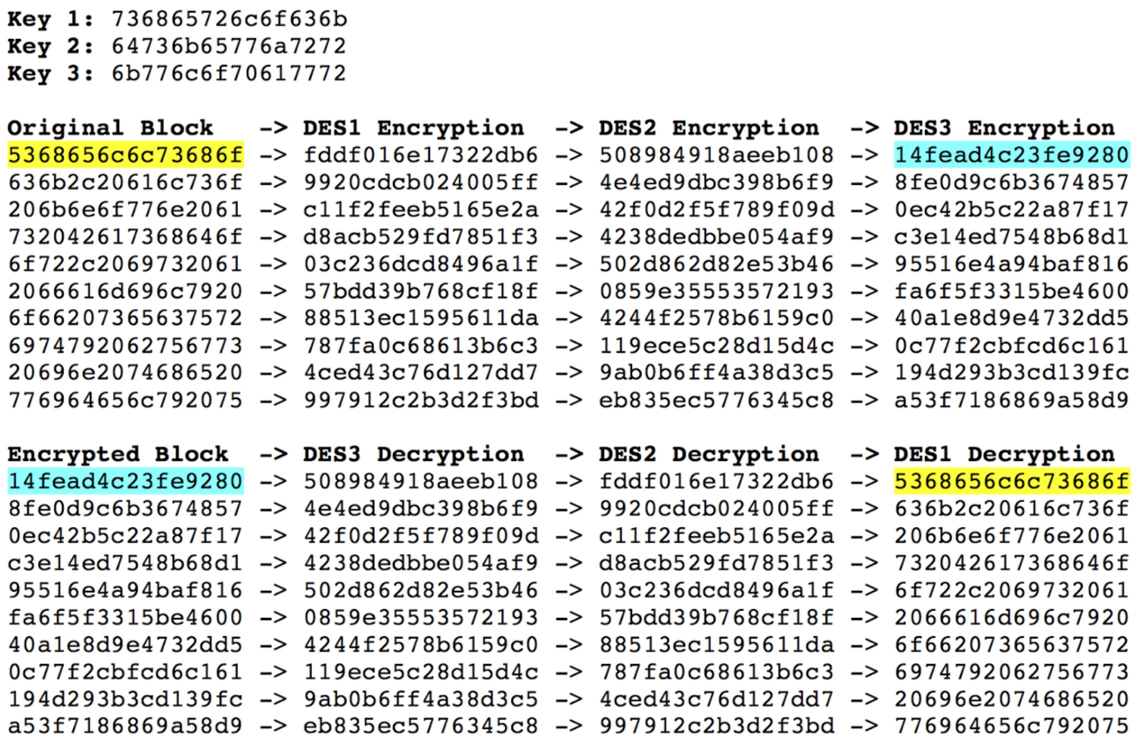
Triple DES block	source/triple_DES_block.sv
Test Benches	
AHB-Lite Slave Controller test	source/tb_AHBLiteSlaveController.sv
DES_block test	source/tb_DES_block.sv
Decoder test	source/tb_Decoder.sv
Default Slave test	source/tb_DefaultSlave.sv
Expansion test	source/tb_expansion.sv
Multiplexer test	source/tb_Multiplexer.sv
Permutation Box test	source/tb_PBox.sv
Roundkey test	source/tb_roundkey.sv
Substitution Box 1 test	source/tb_s_box1.sv
Bus Connections test	source/tb_System.sv
Triple DES block test	source/tb_triple_DES_block.sv
Python Scripts for testing	
Single DES Round Test	python_scripts/single_des_round.py
Triple DES Test	python_scripts/Triple_DES.py
Encounter files	
Encounter config file	encounter.conf
Encounter pad file	encounter.io
Encounter tcl file	encounter.tcl
Documentation	
AHB-Lite Specification	docs/AHB-Lite-Specification.pdf

AHB-Lite Slave Design	docs/AHB-Slave-Design.pdf
DES Documentation	docs/DES.pdf
Block Diagrams	docs/Project.dia
Various Pictures	docs/*.png
Project Related Documents	
Design Budget	docs/Design_Budgeting_Form.xlsx
Design Review	docs/DesignReview.pptx
Project Proposal	docs/ProjectProposal.{docx, pdf}
Report	
Mapped Report	reports/System.rep

Appendix B - Simulation Results

To ensure that the encryption and decryption of the triple DES was correct, a python script was created to generate the correct outputs that would be used to check against the output of the triple DES block. The output of this script is shown below with each 64 bit block represented in hexadecimal. The original plaintext block (yellow) goes through three sets of DES encryption to create the ciphertext block (blue). For DES1 key 1 is used, DES2 key 2, and DES3 key 3. This ciphertext block is used as input for decryption, which goes through three sets of DES decryption to produce the original plaintext block. The keys are reversed for decryption, so key 3 is used for DES3 decryption, key 2 for DES2, and key 1 for DES1.

Figure 11: Expected Outputs for Encryption and Decryption (With Provided Keys)



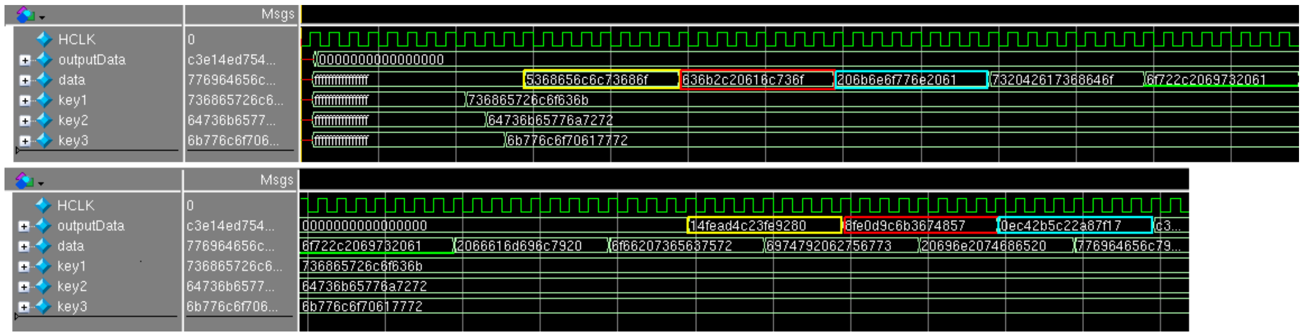
```
Key 1: 736865726c6f636b
Key 2: 64736b65776a7272
Key 3: 6b776c6f70617772

Original Block   -> DES1 Encryption  -> DES2 Encryption  -> DES3 Encryption
5368656c6c73686f -> fddf016e17322db6 -> 508984918aeeb108 -> 14fead4c23fe9280
636b2c20616c736f -> 9920cdcb024005ff -> 4e4ed9dbc398b6f9 -> 8fe0d9c6b3674857
206b6e6f776e2061 -> c11f2feeb5165e2a -> 42f0d2f5f789f09d -> 0ec42b5c22a87f17
732042617368646f -> d8acb529fd7851f3 -> 4238dedbbe054af9 -> c3e14ed7548b68d1
6f722c2069732061 -> 03c236dcd8496a1f -> 502d862d82e53b46 -> 95516e4a94baf816
2066616d696c7920 -> 57bdd39b768cf18f -> 0859e35553572193 -> fa6f5f3315be4600
6f66207365637572 -> 88513ec1595611da -> 4244f2578b6159c0 -> 40a1e8d9e4732dd5
6974792062756773 -> 787fa0c68613b6c3 -> 119ece5c28d15d4c -> 0c77f2cbfcd6c161
20696e2074686520 -> 4ced43c76d127dd7 -> 9ab0b6ff4a38d3c5 -> 194d293b3cd139fc
776964656c792075 -> 997912c2b3d2f3bd -> eb835ec5776345c8 -> a53f7186869a58d9

Encrypted Block  -> DES3 Decryption  -> DES2 Decryption  -> DES1 Decryption
14fead4c23fe9280 -> 508984918aeeb108 -> fddf016e17322db6 -> 5368656c6c73686f
8fe0d9c6b3674857 -> 4e4ed9dbc398b6f9 -> 9920cdcb024005ff -> 636b2c20616c736f
0ec42b5c22a87f17 -> 42f0d2f5f789f09d -> c11f2feeb5165e2a -> 206b6e6f776e2061
c3e14ed7548b68d1 -> 4238dedbbe054af9 -> d8acb529fd7851f3 -> 732042617368646f
95516e4a94baf816 -> 502d862d82e53b46 -> 03c236dcd8496a1f -> 6f722c2069732061
fa6f5f3315be4600 -> 0859e35553572193 -> 57bdd39b768cf18f -> 2066616d696c7920
40a1e8d9e4732dd5 -> 4244f2578b6159c0 -> 88513ec1595611da -> 6f66207365637572
0c77f2cbfcd6c161 -> 119ece5c28d15d4c -> 787fa0c68613b6c3 -> 6974792062756773
194d293b3cd139fc -> 9ab0b6ff4a38d3c5 -> 4ced43c76d127dd7 -> 20696e2074686520
a53f7186869a58d9 -> eb835ec5776345c8 -> 997912c2b3d2f3bd -> 776964656c792075
```

The following waveform shows the input blocks in ‘data’ being encrypted to produce ‘outputData’. The text below the waveform shows that the actual encryption output matches the expected output from the script.

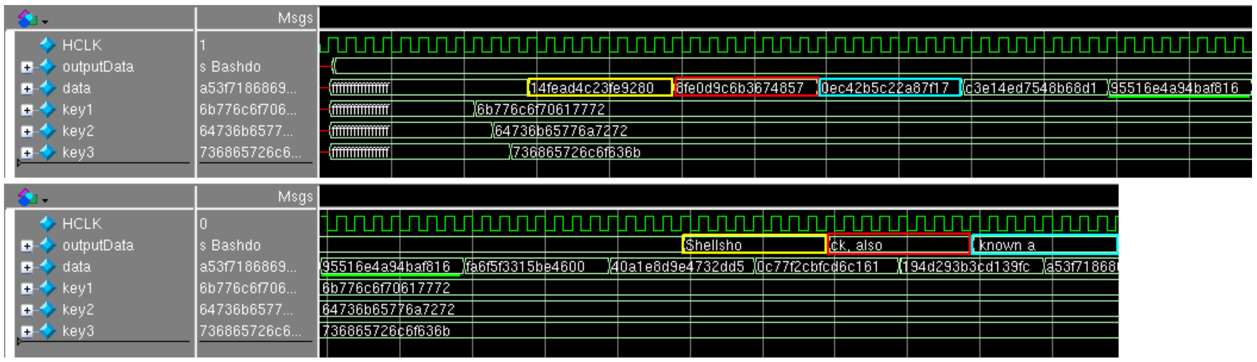
Figure 12: Expected Output for Encryption



```
Original Block   -> DES1 Encryption  -> DES2 Encryption  -> DES3 Encryption
5368656c6c73686f -> fddf016e17322db6 -> 508984918aeeb108 -> 14fead4c23fe9280
636b2c20616c736f -> 9920cdcb024005ff -> 4e4ed9dbc398b6f9 -> 8fe0d9c6b3674857
206b6e6f776e2061 -> c11f2feeb5165e2a -> 42f0d2f5f789f09d -> 0ec42b5c22a87f17
```

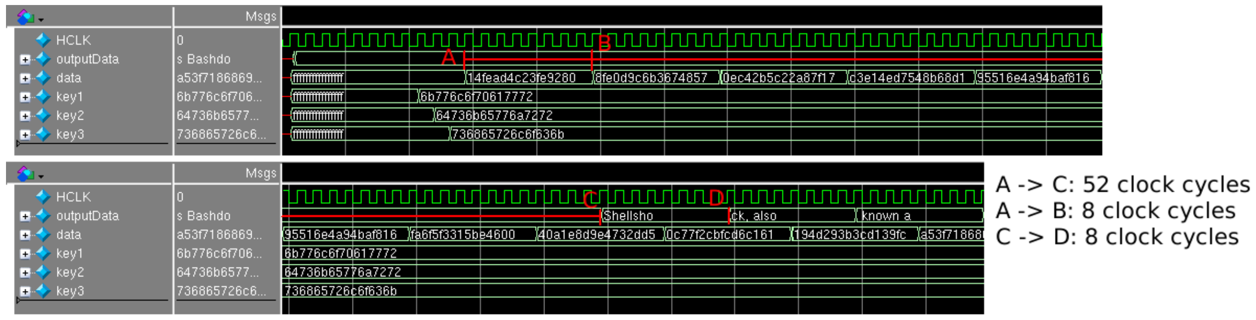
To test decryption the output of the encryption done above was used as input for decryption. The encrypted input ‘data’ and the corresponding decrypted output ‘outputData’ are in that same color boxes. The output is represented in ASCII characters to show that decryption produces readable output that matches the message used as input for encryption.

Figure 13: Expected Output for Decryption



The following waveform shows the pipelining of the triple DES block. The first block is inputted at point A and it takes 52 clock cycles for the output to be available at point C. This is how long encryption or decryption takes. At point B, 8 clock cycles after point A, the second block is inputted. 52 clock cycles later the output is available at point D, 8 clock cycles after point C. New blocks are inputted every 8 clock cycles, and once the first output is available new output blocks are available every 8 clock cycles.

Figure 14: Pipelined Operations in our Design

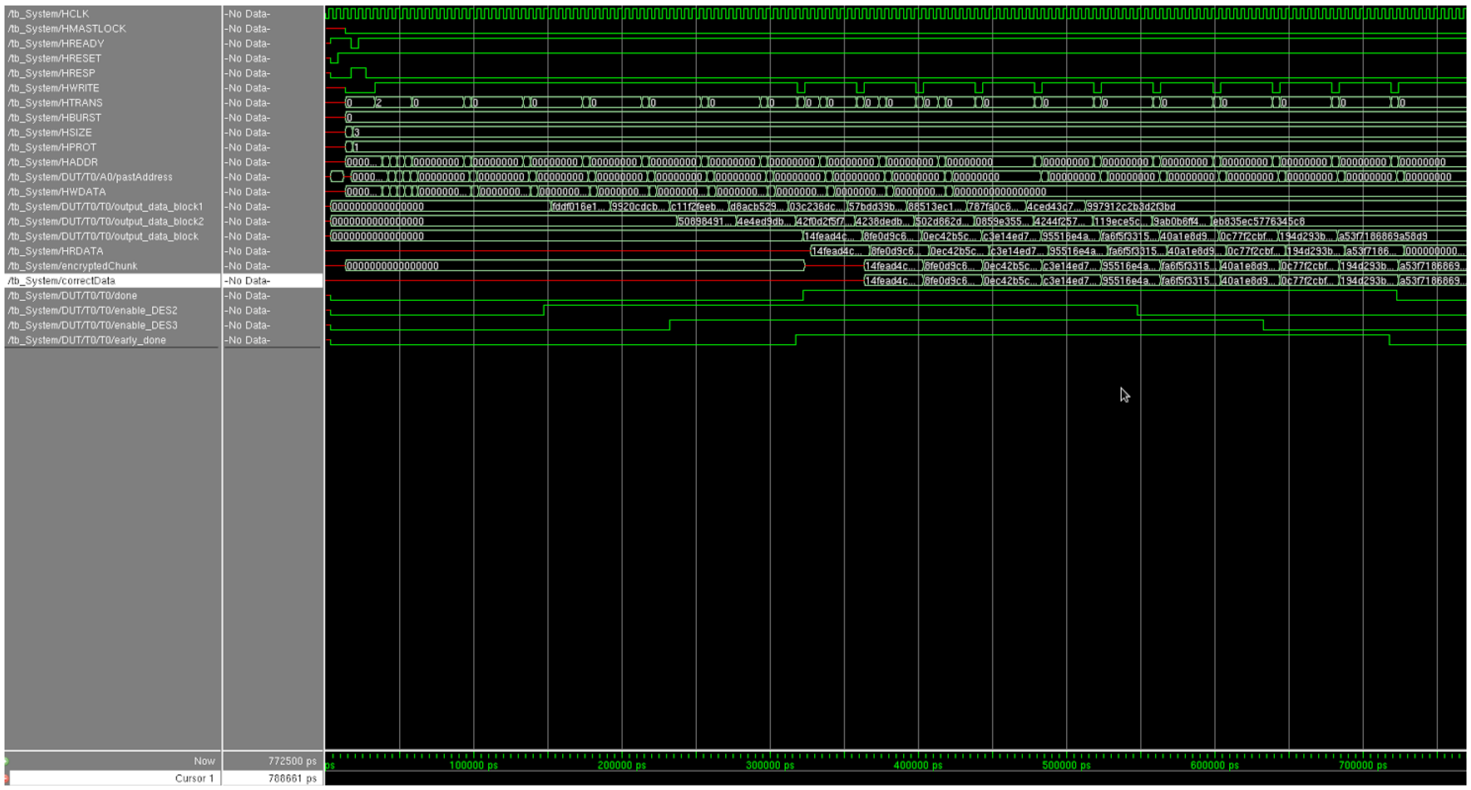


Pictured below, the operations of the AHB-Lite Bus can be seen. The values on HADDR are shown to be send to pastAddress on the rising edge of the next clock cycle. This demonstrates the pipelined nature of the addresses on the AHB-Lite Bus. Also, the addresses that are being referenced (which can be better examined using Figure 3), are being sent the values that are on the HWDATA bus. These values are being sent because the values on the HTRANS line (2’b10) signifies a transfer (A Nonsequential transfer to by specific). The default values, as specified in the Standards and Protocols section, are shown to be set on the bus. These values, along with correct address values, show steady HREADY and HRESET signals. HRESP being set to 1’b0 means that no errors are present. The HREADY is constantly 1’b1 because our device only needs one clock cycle to grab the values, never requiring HREADY to be deasserted.

Figure 15: AHB-Lite Bus Interface Interactions

Pictured below is an encryption operation (With bus activity along with pipelining). This picture shows the pipelining that occurs in our design. Each DES block takes 16 cycles to complete. When each DES block ends, the next block (or the bus), grabs the values to continue computation on, or in the case of the bus, output. Each value is grabbed 17 cycles after it is first started (16 cycles for the pipeline, and one cycle to grab from the previous round). In this picture each pipeline stage and grab can be seen. Also, the final grab from HRDATA to encryptedChunk (a testbench variable used to show output) can be seen. This grab happens on the final clock cycle of the 8 clock cycles that the value is ensured valid. All of the bus values, meanwhile, can be seen to be correct. The transaction type changes from idle to nonsequential when sending and receiving data, and the addresses are picked up in a pipelined fashion.

Figure 16: Sample Chip Operation (Many Features Shown)


/tb_System/HCLK
/tb_System/HMASTLOCK
/tb_System/HREADY
/tb_System/HRESET
/tb_System/HRESP
/tb_System/HWRITE
/tb_System/HTRANS
/tb_System/HBURST
/tb_System/HSIZE
/tb_System/HPROT
/tb_System/HADDR
/tb_System/DUT/T0/A0/pastAddress
/tb_System/HWDATA
/tb_System/DUT/T0/T0/output_data_block1
/tb_System/DUT/T0/T0/output_data_block2
/tb_System/DUT/T0/T0/output_data_block
/tb_System/HRDATA
/tb_System/encryptedChunk
/tb_System/correctData
/tb_System/DUT/T0/T0/done
/tb_System/DUT/T0/T0/enable_DES2
/tb_System/DUT/T0/T0/enable_DES3
/tb_System/DUT/T0/T0/early_done
-No Data-
Now 772500 ps
Cursor 1 788661 ps
100000 ps
200000 ps
300000 ps
400000 ps
500000 ps
600000 ps
700000 ps