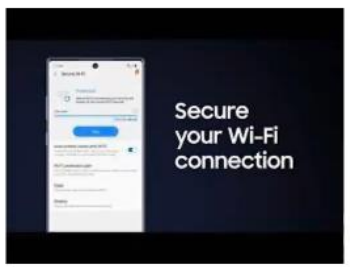
LAB2: COMPLEX COMBINATIONAL LOGIC AND DEBUGGING: HARDWARE-BASED DATA ENCRYPTION STANDARD (DES)

Nate Lannan
Electrical and Computer Engineering Department
Stillwater, OK 74078 USA

Objective

- Cryptography and especially hardware is an important element for the future.
- Future engineers can gain fantastic jobs if they understand what security is and how it works.
- I believe this laboratory will give you that opportunity and give you a fantastic opportunity to do great things!
- You can do this!!! And you are the best!!!
  - Remember that I care about your success, and I want you to succeed – so do not feel abandoned and come see me if you need help!
  - I am here anytime but its easier if you get started early!
  - Repeated : do not wait to complete this lab as you will have a very hard time completing it if you wait.

Why?



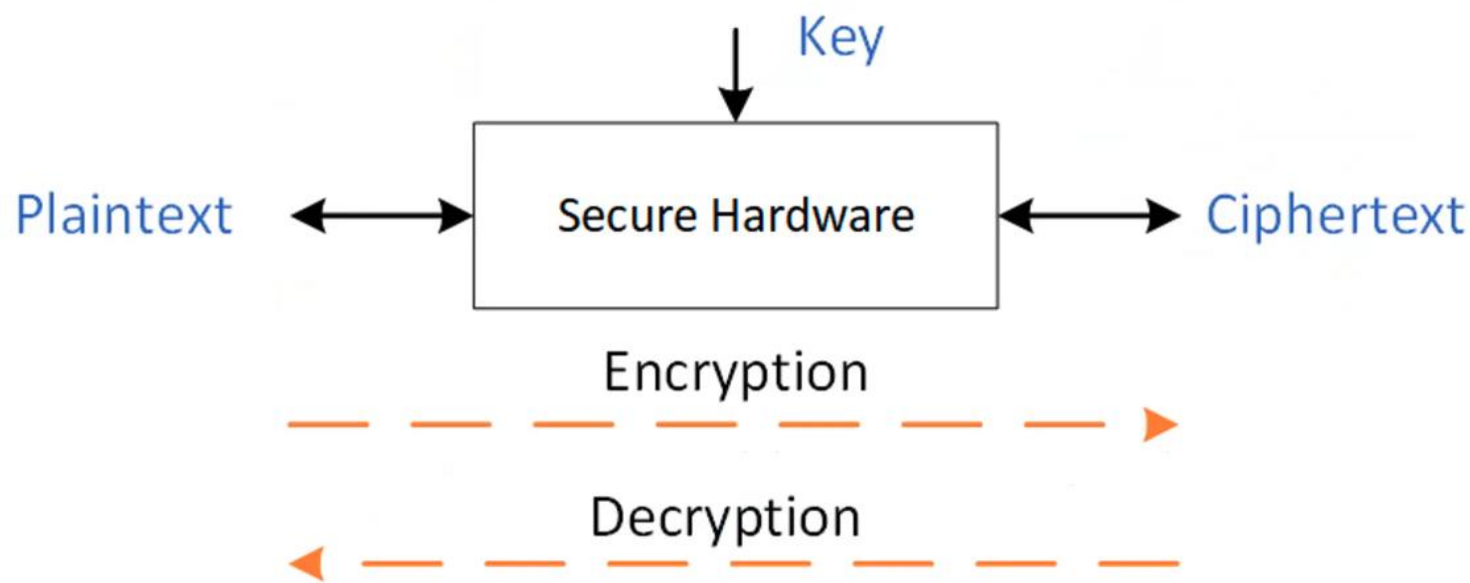
Cryptography in Hardware!!!

Cyber is a Great Career!

- Some figures I have seen indicate that 75 million jobs and 47% of jobs are vulnerable to automation – security will be important for jobs!
- The global blockchain market is estimated to grow from USD 3.0 billion in 2020 to USD 39.7 billion in 2025! [marketsandmarkets.com]
- Security hardware makes sense as its fast and easy to deploy for any user.

Cryptography

- Cryptography is quite easy if you think about it as a password and access.
- There are fancy diagrams available, but I made one that I think will help.



https://en.wikipedia.org/wiki/Caesar_cipher

Caesar Cipher

- A Caesar cipher or *shift* cipher is one of the simplest and most widely known encryption techniques.
- It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.

Plain	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W

[Wikipedia]

```
Plaintext:  THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD
```

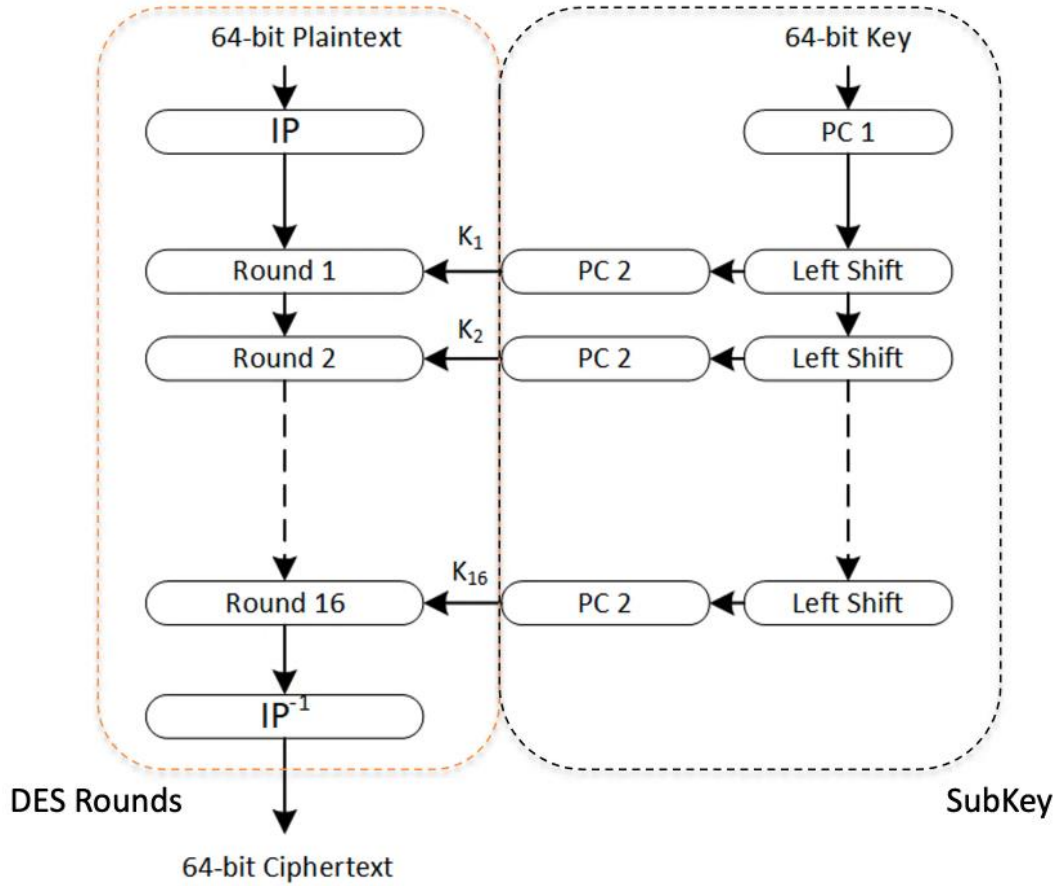
$$E_n(x) = (x + n)\ mod\ 26$$
$$D_n(x) = (x - n)\ mod\ 26$$

- Cryptography is used to help hide messages from an adversary and has been used for a long time and shifting is key to most modern-day protection mechanisms.
- Digital logic is now becoming a better way of securing messages as its faster, more efficient and secure.

DES

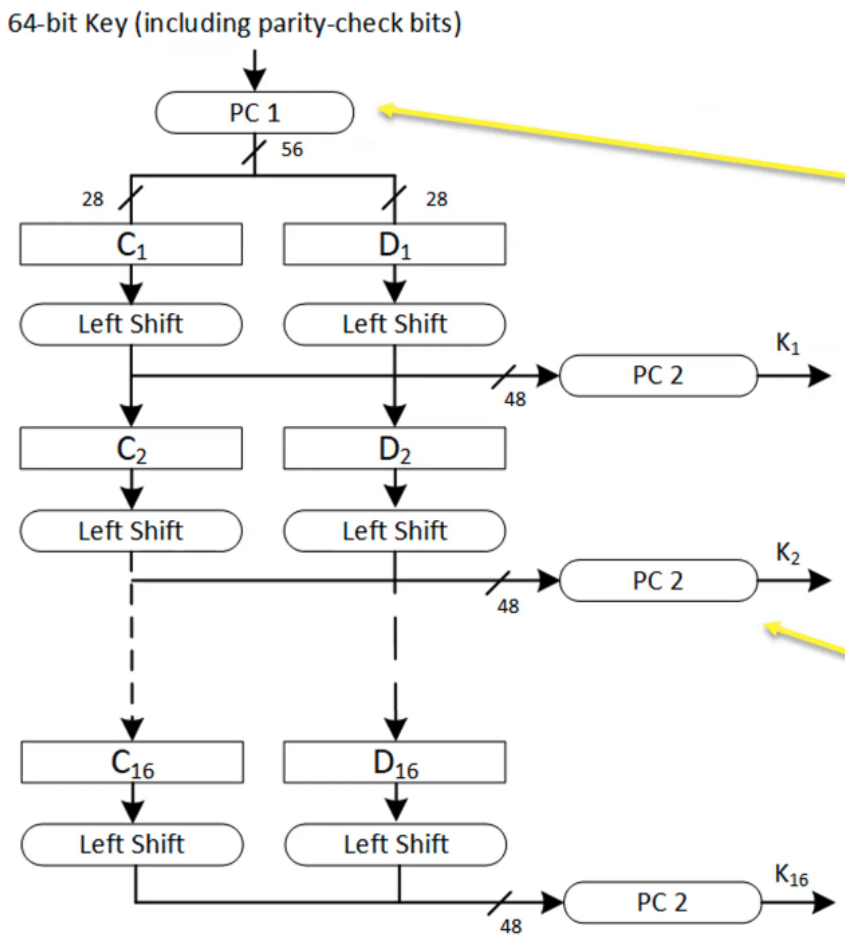
- The Data Encryption Standard or DES is a symmetric-key algorithm for encryption of data.
- The key length is 56 bits and is an early form for security in hardware as it was straightforward and easy to use.
  - Unfortunately, the relatively short key size made it easy to break so its no longer used (it was cracked in 1999 via brute force in less than 24 hours).
  - But the basics of DES are still used in something called AES that is commonly used today.
- DES was invented in the 1970s for IBM based on earlier designs by Horst Feistel.
  - The algorithm was later submitted to the National Security Agency (NSA) for use and standardized by the Federal Information Processing Standard (FIPS) in 1977.
  - Triple DES or (3DES) is still used today in some applications.
- Despite DES losing the lofty position of being the go-to data encryption standard algorithm, it is still worth learning.

Data Encryption Standard (DES)



- DES has 16 rounds denoted by the Feistel block
- It uses 16 generated SubKeys that are produced from the center block
- The architecture uses symmetric-key encryption in that it can be encrypted or decrypted based on its direction.
- It has two main blocks : DES block and SubKey generation.

SubKey Generation



Permutation Blocks

Left						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
Right						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Table 3: Permutation Choice 1 (PC-1) Function[1]

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Table 4: Permutation Choice 2 (PC-2) Bit Function

…more detail in Lab 2 document

SubKey Generation

- Key generation is the easiest portion of DES
  - It basically does two items : Permutations and Rotation,
  - PXX is sometimes called a permutation or switching of order
- Rotation?
  - A rotation is the same as a shift but specialized (sometimes called circular shift)
  - The rotation works on two separate groups of 32-bits (i.e., breaks the 64-bits into two segments of 32-bits each).
  - It also has left circular shifts by 1 or 2 based on the round.

Iter #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Left Shift	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Be careful: Positions by Security Engineers

- Security is still an ongoing area and there are many who are in this area: mathematicians, engineers, computer scientists, physicists, …
- I personally feel Electrical and Computer Engineering people understand the issue on where positions happen, but many who are not part of our community seem to not understand yet.
- I have given you an example to help by showing the IP (Initial Permutation) and IP^{-1} (Final Permutation) stages update the position (please follow along).

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Table 5: Initial Permutation (IP) Function

```
// Initial Permutation
module IP (inp_block, out_block);

   input logic [63:0]  inp_block;
   output logic [63:0] out_block;

   assign out_block[63] = inp_block[64-58];
   assign out_block[62] = inp_block[64-50];
   assign out_block[61] = inp_block[64-42];
   assign out_block[60] = inp_block[64-34];
   assign out_block[59] = inp_block[64-26];
  ...
```

Bit Swizzling in SystemVerilog (SV)

- Rotation in SV is easier with bit swizzling
- Bit swizzling has two functions in SV
  - Concatenation (putting two vectors together)
  - Replication (repeating items within a vector or bit)
- It is probably one of the coolest and best features of SV!
  - It is often under utilized – use in this lab!!!
  - It will save you an enormous amount of time!

```
assign y = {a[2:1], {3{b[0]}}, a[0], 6'b100_010};
// if y is a 12-bit signal, the above statement produces:
y = a[2] a[1] b[0] b[0] b[0] a[0] 1 0 0 0 1 0
// underscores (_) are used for formatting only to make
// it easier to read. SystemVerilog ignores them.
```

Rotation by 1 or 2

- Rotations are just circular shifts where anything that is shifted gets placed back in the registers.
- Use `assign` statements in SV to help with bit swizzling; however, its only 1 or 2 based on round.
- Good calculator available : https://onlinetoolz.net/bitshift#base=16&value=B8&bits=8&steps=2&dir=l&type=circ&allsteps=1

	A	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
	0xB8	1	0	1	1	1	0	0	0
	B	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
ROL1	0x71	0	1	1	1	0	0	0	1
	C	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
ROL2	0xE2	1	1	1	0	0	0	1	0

8-bit Example

```
assign C = {A[5:0], A[7:6]}
```

56-bit Key!

- Your DES key is is 56-bits with 8-bits of parity.
- Error detection is an important part consideration of data transmission
  - Parity checking
  - Redundancy checking
- Parity involves performing a basic calculation of the number of digital 0's and 1's making up a transmission unit
  - Parity calculated on even or odd number of 1's but for DES it is odd parity.
    - Example: `1010_101` → `1010_1011`
  - Parity bit is set per frame (byte or character)
  - Parity checking is found mostly in Asynchronous communication
- The parity within the key is computed simply by XOR'ing of the previous 7 bits.

K_{63}	K_{62}	K_{61}	K_{60}	K_{59}	K_{58}	K_{57}	K_{56}	...	K_7	K_6	K_5	K_4	K_3	K_2	K_1	K_0
X	X	X	X	X	X	X	P_7	...	Y	Y	Y	Y	Y	Y	Y	P_0

Encrypt/Decrypt Breakdown

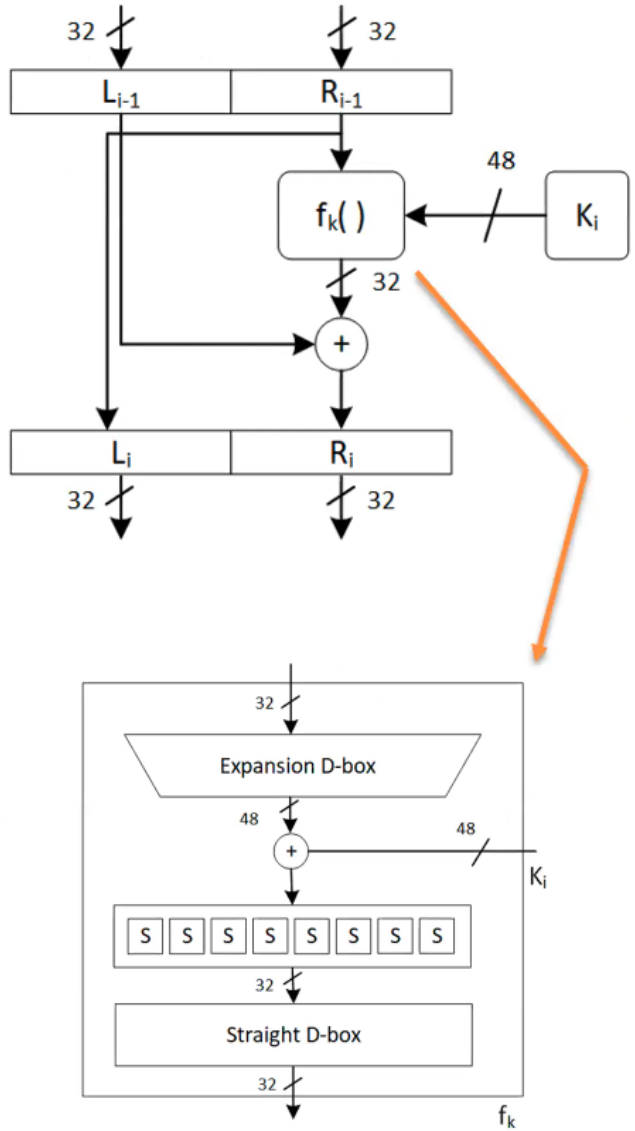
- Read Lab 2 for more details (this is broken down in detail)
- It basically does 3 functions:
  - Initial Permutation (IP)
  - Feistel Block (f_K)
  - Final Permutation (IP^{-1})
- Use the position information discussed on the previous slide!
- Some of these blocks are repeated and are called rounds in crypto-based hardware
  - DES has 16 rounds (just instantiate each round)
  - They are based on each 16 SubKeys (K_1 through K_{16}) that are generated during SubKey generation.

Feistel Block

- The key generation is easiest and probably should be done first to make sure its working.
- The main part of the encryption and decryption is done based on something called the Feistel block.
- This block was introduced by Horst Feistel at IBM in his earlier work on cryptography and its an important part of many cryptography algorithms.
- It is important to understand that I am not asking you to understand how each block works – just implement it.
  - There are some great books and websites that can help you understand more about cryptography for those that are interested.
  - See the references in the repo as well as on the Lab 2 document for more information.

Feistel Block (f_K)

- Most cryptography-based algorithms use rotation and exclusive OR (XOR) to help with modular arithmetic
- The Feistel Block uses something called S-boxes and Permutations (EP and SF)
- DES uses permutations that are called substitutions
  - These are called substitution boxes and basically are matrices that allow the symmetric-key algorithms to perform substitution.
  - Some S-boxes are generated dynamically – ours are just tables that are looked up.
- Use the provided stubbed SV (`des_stub.sv`) to help you get started.
  - https://en.wikipedia.org/wiki/Method_stub



S-box Implementation

- Again, I give you each of the eight S-boxes.
- You should split 48 bits of input into each S-box (i.e., 6-bits) each.
- Each S-box outputs 4-bits as an output for a total of 32 bits

```
logic [47:0]        sbox_pre;
assign sb8in = sbox_pre[5:0];
assign sb7in = sbox_pre[11:6];
assign sb6in = sbox_pre[17:12];
assign sb5in = sbox_pre[23:18];
assign sb4in = sbox_pre[29:24];
assign sb3in = sbox_pre[35:30];
assign sb2in = sbox_pre[41:36];
assign sb1in = sbox_pre[47:42];
```

Implementation

- You will implement DES in SystemVerilog.
- **All blocks are done with combinational logic *only*.**
- Verification will be discussed shortly but use your testbenches from Lab0 and Lab1 to help you with the testing!
  - I have also given you another testbench that actually reads values from a file with 4 sample keys, plaintext, and ciphertext.
  - There's also a small Java program (`genDESkey.java`) I wrote that generates random keys with parity in case anyone needs some to test.
- <u>You should have an input that indicates either encryption or decryption!</u>
- After you get your design working, please implement on the DSDB board and use the 7-segment display to display the plaintext and ciphertext and verify it is working correctly.

7-segment Display

- Uh-oh!
- You only have four (4) 7-segment displays, but you have 64-bits of key, plaintext, and ciphertext.
- You will have to figure a way to display that on the four 7-segment displays.
- It is not hard if you think about it a little but use the switches and push buttons to help you with this.
  - Hint: muxes are your friend!
- You should also display on the LEDs if odd parity is not correct on the 64-bit key.

Summary of DES

Algorithm 1 SubKey Schedule

Require: *key*
Ensure: `key[63:0]` with odd parity
  $\{C[0], D[0]\} = PC1(key)$
  for $1 \leq i \leq 16$ **do**
    $C[i] = LS[i](C[i-1])$
    $D[i] = LS[i](D[i-1])$
    $K[i] = PC2(\{C[i], D[i]\})$
  end for

- Encryption or Decryption is done with the same block because it is a symmetric algorithm.
- The only difference is the order of keys!
- K1 through K16 for encryption or K16 through 1 for decryption.

Algorithm 2 Encipherment

Require: plain block, `K[1]` through `K[16]`
  $\{L[0], R[0]\} = IP(\text{plain block})$
  for $1 \leq i \leq 16$ **do**
    $L[i] = R[i-1]$
    $R[i] = L[i-1] \oplus f_k(\{R[i-1], K[i]\})$
  end for
  cipher block $= FP(\{R[16], L[16]\})$

Algorithm 3 Decipherment

Require: cipher block, `K[1]` through `K[16]`
  $\{R[16], L[16]\} = IP(\text{cipher block})$
  for $1 \leq i \leq 16$ **do**
    $R[i-i] = L[i-1]$
    $L[i] = R[i-1] \oplus f_k(\{L[i-1], K[i]\})$
  end for
  plain block $= FP(\{L[0], R[0]\})$

Verification

- Verification is key to digital systems and especially digital logic.
- Remember that verification is the process of seeing something backwards in the flow (e.g., Is my output the correct output from this block?)
- To help with verification we will use a Java program I developed (`DES.java`).

```
Original plain Text: 2579DB866C0F528C
Key: 433E4529462A4A62

Encryption:

After initial permutation: 5646B9278C13B66C
After splitting: L0=5646B927 R0=8C13B66C

Round 1  8C13B66C F3EFC169 208066A253BA
Round 2  F3EFC169 25DAF255 C0B6508F6DC2
Round 3  25DAF255 1890CFBF 44D6422CC355
Round 4  1890CFBF AFB98FA0 62D142D3C4C6
Round 5  AFB98FA0 8F76DBD7 28C143CC8789
Round 6  8F76DBD7 C176D0E5 21411B9A764D
Round 7  C176D0E5 C7401A8C 2501917AD3A0
Round 8  C7401A8C B748825A 170891906D2B
Round 9  B748825A 61239171 084949255DD5
Round 10  61239171 FE28B577 01690D8B80F3
Round 11  FE28B577 CDB650DE 012D81C7CF05
Round 12  CDB650DE 8B8270E5 512CA11A07DC
Round 13  8B8270E5 DDDBEE19 D1A480D9D185
Round 14  DDDBEE19 5F82D63F 5086864266A9
Round 15  5F82D63F B35B4964 709006FA390D
Round 16  B35B4964 850AC7BE C03E202F8437

Cipher Text: ECB54739A1832EC5
```

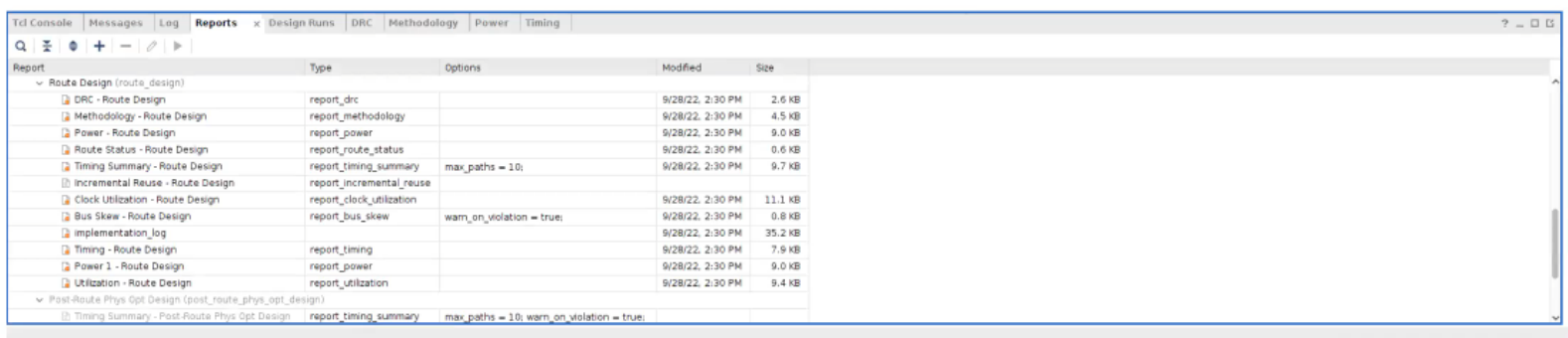
- You can use the Java program to help you with each block and what each output should be based on whether you are performing encryption or decryption
- I made a `Makefile` that you can use to compile the tool and use it.
- Please install Java JDK if you want to run on your laptop or desktop (It should be installed in ENDV 360)

Debugging with ModelSim

- You will have to use ModelSim to help you debug things.
- Use the Java program to verify what works and see if you can figure out what you did wrong.
  - Many times, it is a typo and easy to fix.
  - But this requires good understanding of the methodology for verification.
- Let's show a demo (the demo is in the repository, as well).

PPA

- We learned that Power, Performance and Area are values we utilize to determine how well our design works.
  - Power : (energy/power)
  - Performance : (delay/time)
  - Area (space)
- We will analyze this more in depth compared to Lab0/Lab1
  - First run through an implementation
  - Make sure your design works
  - Add `report_timing` and `report_power` to your reports through Implementation
    - This will require you to re-run the Implementation phase



Extra Credit

- Today, employers want to see what you did during your undergraduate degree in terms of topics and assignments.
- Document your labs, including something like this one, to impress future employers as it will help you get coveted jobs and internships.
- The lab we are doing this week is not easy and people will be impressed.
- However, caveat emptor (buyer beware) – no one wants to hear you didn't do anything impressive in engineering school.
  - Impress people with added features.
  - Build a website (check out github.io) – its free and easy!
  - Build a video showing what you did!