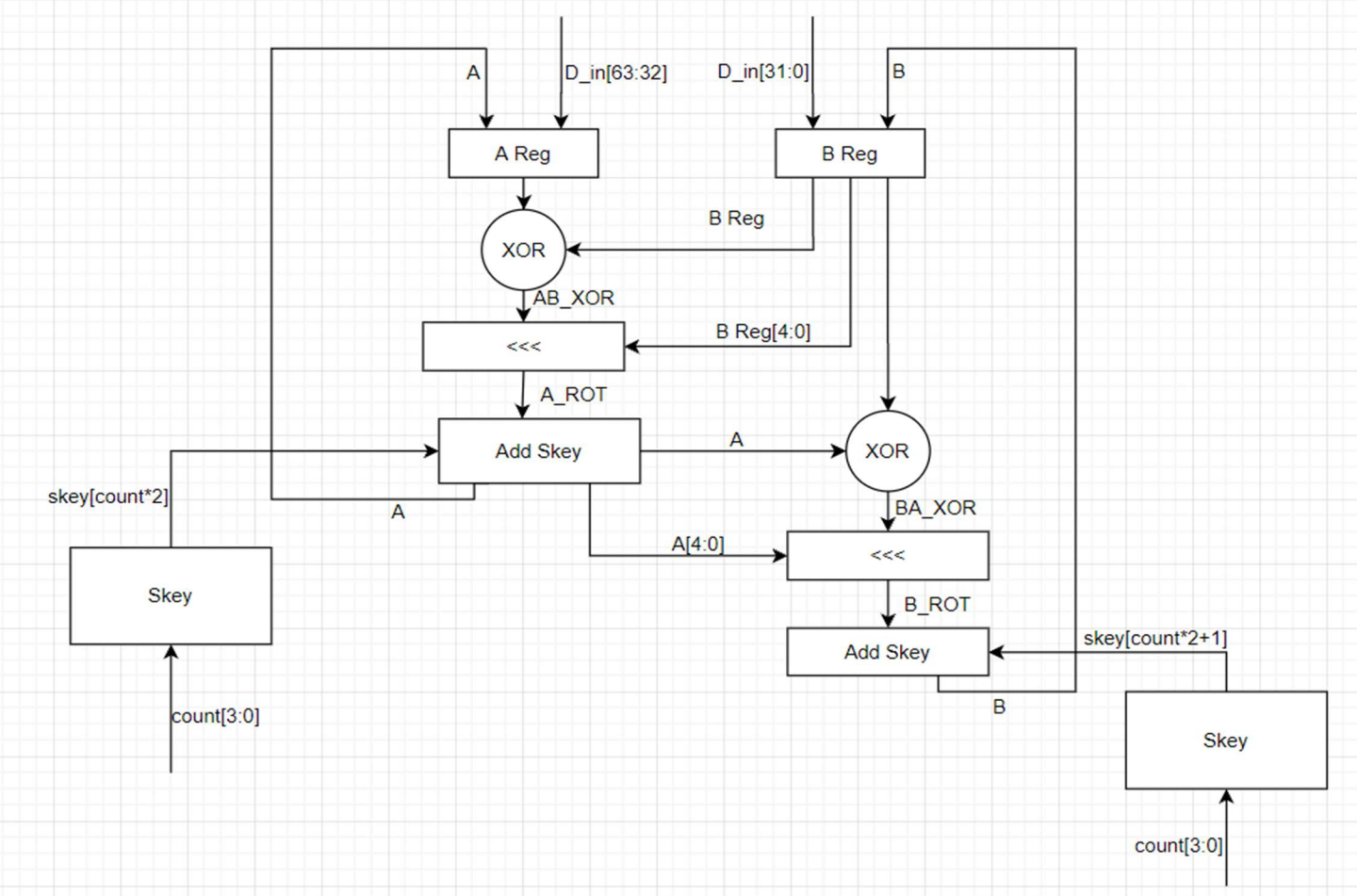
Vinay Kumar Hanumanthappa
N18468999 | vh2115@nyu.edu
HW 4 RC5 Simple Function – AHD

Video Link: https://youtu.be/swmTgY4zTAI

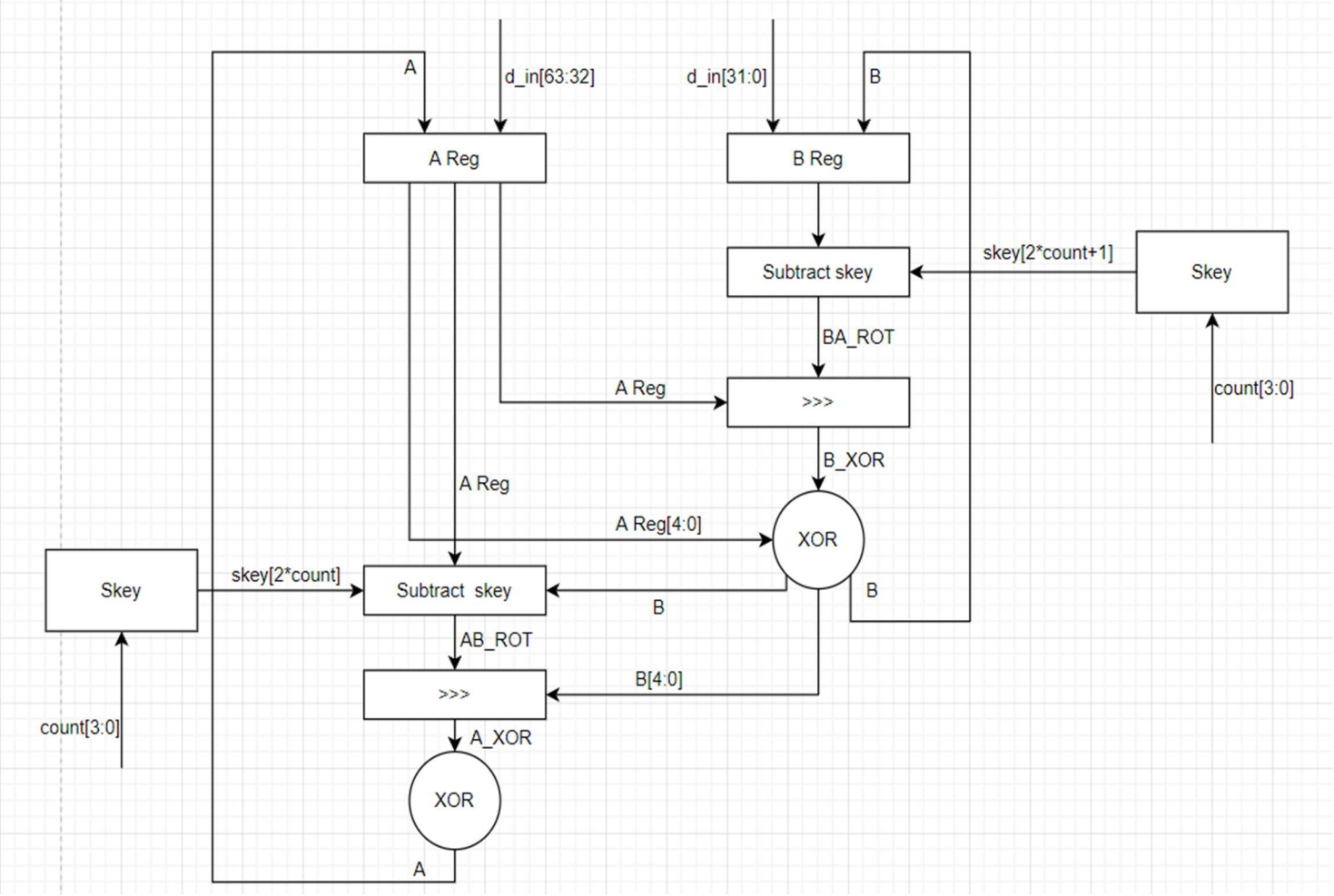
Below are the Datapath for Encode and Decode process using in RC5 system.
clr(code)⇔rst(explanation), I have excluded rst from the data path to keep it simple from both encode and decode.

Encoder:



In our Encode design we are adding 24 keys via complex function to 64-bit data (16-digit hex number) to encode it.
The encode operation is as follows.
We first split the input data into 2 halves and upper half is loaded into A register and Lower half is fed to B register.
once the input data is split, we perform XOR operation between the lower half and upper half of the data followed by left rotation of the XOR result by the number of bits, which is evaluated from last 5 bits of lower half of the data.
Here the order is very import as we need to do the exact opposite in Decode, the lower half is still plain old data.
The result of this operation is performed and stored in upper half.
Once the operation is completed, we add the result of rotation with a key from skey, first key placed in even position starting from 2, position 1 and 0 are left empty and not used, as it would be used to store secret key in full RC5 encryption and we are not using one now.
After this the output of the addition is fed back , as upper half(A Register) of 1st round of encryption, as the new input.
Similarly, these operations are done on lower half as well but with the encrypted data of upper half.
Here, for lower half we use skey in odd positions starting from 2, i.e., starts from 3.
once we are exhausted with all the keys i.e., 12 rounds of encryption we concatenate the encrypted Upper and Lower half to form the final encrypted data. Again, the concatenation is not showed in Data path to keep it simple.

Decoder:



In our Decode design we are performing the opposite function in reverse direction except the XOR function as the same would give the inverse of the operation.

We first split the encoded data into 2 halves and upper half is loaded into A register and Lower half is fed to B register.

we subtract the lower half of encrypted data with a key from skey, last key placed in odd position starting from 25 down to 2, i.e., key in 25^{th} position, if we have to relate it with encoder it is the final key added to encrypted data of B in encoder and hence its subtraction is our first operation.

Once the key is removed we perform right rotate operation(to offset the left rotate we performed in encode) by number bits that is evaluated from last 5 bits of upper half of the data.

once the rotation is completed, we perform XOR operation between the rotated result of lower half of the data and upper half of the encrypted data. Here the upper half is still the final encrypted data.

After this the output of the XOR operation is fed back , as lower half(B Register) of 1^{st} round of decryption, as the new input.

Similarly, these operations are done on upper half as well but with the 1^{st} round pf decrypted data of lower half.

Here, for upper half we use skey in even positions starting from 25 down to 2, i.e., 24^{th} position key.

once we are exhausted with all the keys i.e., 12 rounds of decryption we concatenate the decrypted Upper and Lower half to form the final Decrypted data/message. Again, the concatenation is not showed in Data path to keep it simple.

I have few assumptions/defaults taken from ppt., reset is asynchronous reset and Active low i.e., it resets a register and b register along with count to default value input (63:0), input (31:0) and 1 for up counter/12 for down counter when it is '0'.

I have kept ppt convention for naming i.e., rst=clr along with other naming same as ppt both in encode and decode.

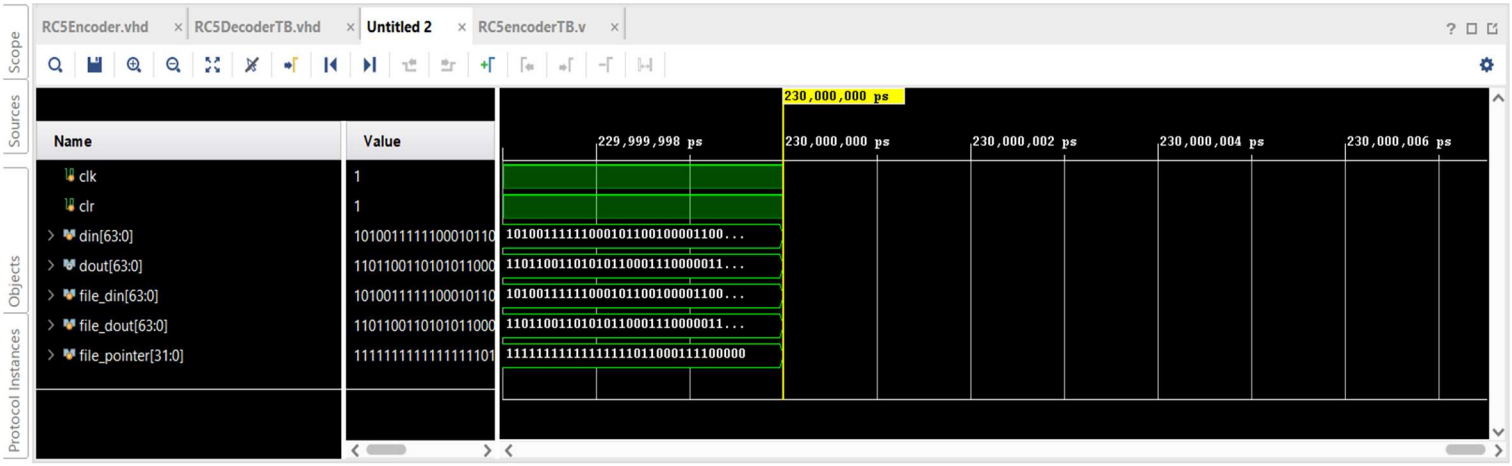
For testbench, the clock, reset and wait before the next input, to test 100 plaintext/cipher test, is adjusted in such a way that the next input is fed using asynchronous reset once the encode/decode of previous input is completed.

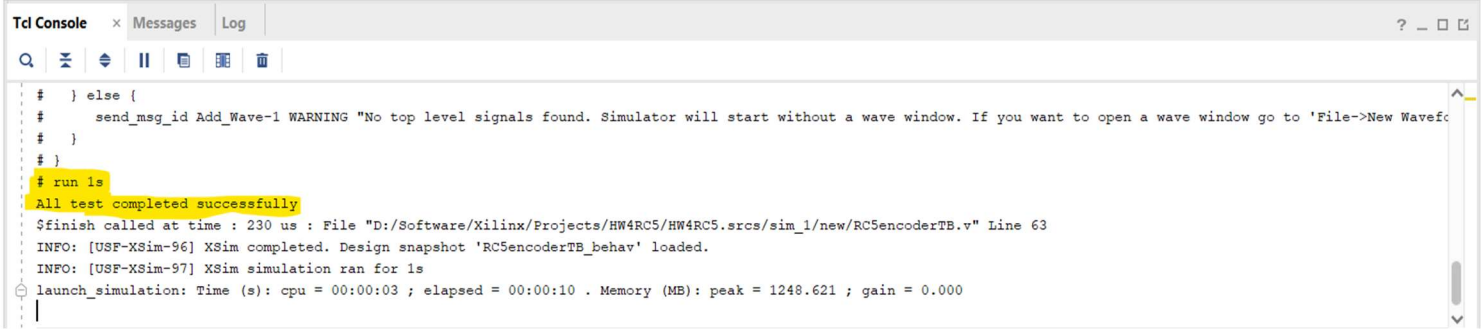
Coming to design the input data is split into 2 half, upper half assigned to A register and lower half assigned to B register. The process of encryption/decryption involves using these 2 halves to perform complex operation on one another along with addition of keys from rom, each encoding cycle uses 2 keys odd key to encode/decode B register, even key to encode/decode B register. We have 12 cycles of encryption since there are total of 24 keys.
I have generated around 200 random 16-digit hex number online, encrypted them using given python script, which is used in testbench to compare the encryption(message.mem) and decryption(encoded_messaage.mem) separately. Changed the simulation properties so that the process run for more time than the defaulted 1000ns [set to 1s] which can test all scenario and give me the final output.

Encoder is in VHDL and its Testbench is in Verilog.
Decoder is in Verilog and its Testbench is in VHDL.

I have defined clr(rst) as a separate clock signal, I could have set that in data load as well (commented this part of code and it works! just need to uncomment it and comment always process of clr(rst)].
If we write separate process for clr(rst) the delay will be twice as that of clr cycle in data assignment block, to offset the timing delay, as both data assignment block and clr block run in combination.

Encoder test Bench:





Decoder Test Bench: