Anant Singh
Advanced Hardware Design
October 14, 2021

Homework 4

Deliverables-

1. Datapath

1.1 Simple Function Datapath

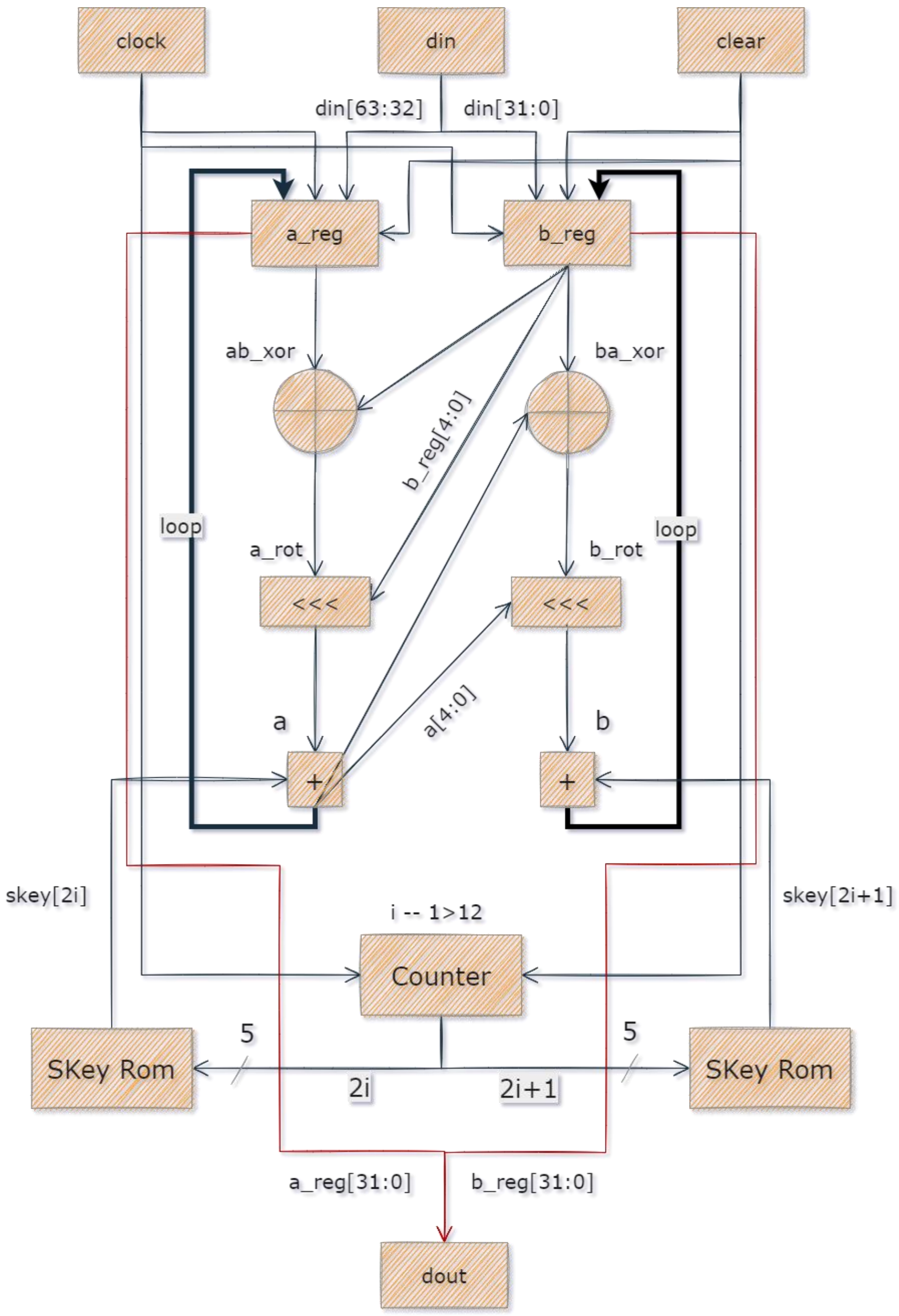

Image 1: Simple Function Datapath

1.2 Simple function inverse Datapath

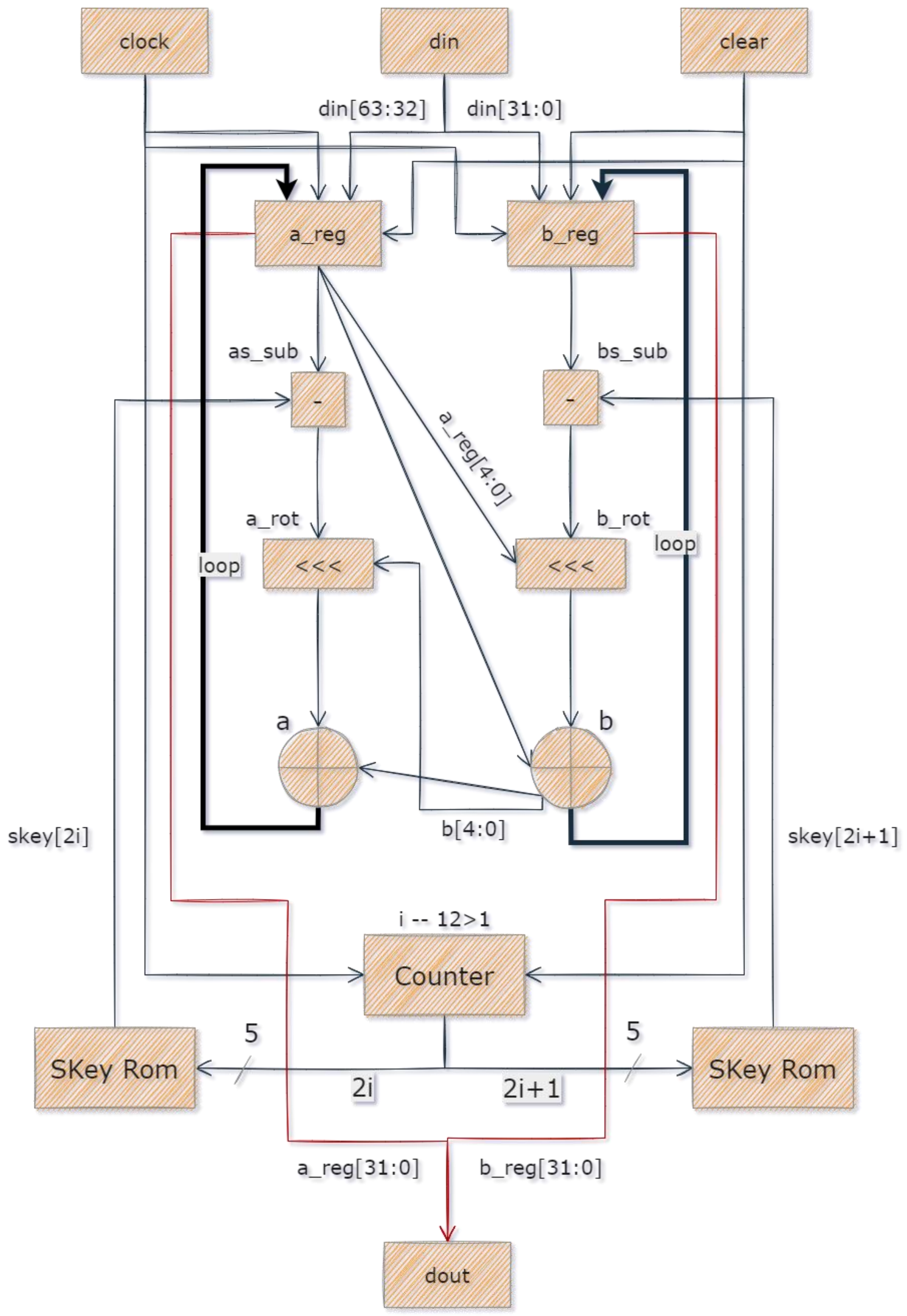

Image 2: Simple Function inverse Datapath

2. Code Description and Justification

Encoder working: The 64 bit input is divided in half and loaded into two registers a_reg and b_reg of 32bit length each. These registers are updated on each positive edge clock signal to the updated value after following the relationship described in simple function. This involves taking XOR of a_reg and b_reg followed by a b_reg[4:0] dependent left rotation. After that value of a calculated by adding key from skey rom based on the cycle (i.e 1 to 12, as there as 12 encryption cycles). The address for key is cleverly generated by concatenating value generated by counter (on each positive edge of clock signal) and 0 this equals 2i. Then process to calculate b continues in similar fashion. The main point to pay attention is that instead of using values from a_reg we use latest a from previous steps. ba_xor is calculated by XOR between b_reg and a followed by left rotation dependent on a[4:0]. Finally, we get value of b by adding output from rotation and a key from skey rom dependent on counter value i, which is skey[2i+1]. This algorithm is designed to avoid loops and depend on clock cycle and counter to implement the simple function. Once the counter reaches 12 from 1 the counting stops and the variable Encoded becomes high, which stops updating process of a_reg and b_reg which means the din is encrypted now. The d_out is concurrently assigned to concatenation of a_reg and b_reg. The clear input signal is used to reset the process (enable high) and load input value to the a_reg and b_reg. Low clear also resets the counter to 1 and encoded to 0, enabling the process again. This whole process takes 13 clock signals to complete the encryption, 1 cycle to load inputs and 12 to encrypt it. All the calculation steps are concurrent and registers and counter (upon which the output depends) are updated in a clocked manner which imitates a loop.

Decoder working: The 64 bit input is divided in half and loaded into two registers a_reg and b_reg of 32bit length each. These registers are updated on each positive edge clock signal to the updated value after following the relationship described in inverse simple function. This involves subtracting a key from skey dependent on I from counter such as skey[2i+1]. Then b_rot is calculated by right rotation dependent on a_reg[4:0]. Then b is XOR of b_rot and a_reg. Then process to calculate a is similar to calculating the b. The main point now is that we use latest b instead of old value from b_reg. as_sub is evaluated by subtracting counter output dependent skey from skey rom which is skey[2i]. Followed by a right rotation dependent on b[4:0]. Finally, we calculate a by XOR of a_rot and b together. This algorithm is designed to avoid loops and depend on clock cycle and counter to implement the simple function. Once the counter reaches 1 from 12 the counting stops and the variable Decoded becomes high, which stops updating process of a_reg and b_reg which means the din is decrypted now. The d_out is concurrently assigned to concatenation of a_reg and b_reg. The clear input signal is used to reset the process (enable high) and load input value to the a_reg and b_reg. Low clear also resets the counter to 12 and decoded to 0, enabling the process again. This whole process takes 13 clock signals to complete the decryption, 1 cycle to load inputs and 12 to decrypt it. All the calculation steps are concurrent and registers and counter (upon which the output depends) are updated in a clocked manner which imitates a loop.

Testbench working: Testbench includes 2 processes out of which on is to generate clock signal, the other process is to test the function of Encoder or Decoder. The testcases are loaded from text file using built-in library in Verilog and through imports in VHDL. The testing process includes a loop to test each input against the correct value of encryption/decryption. Each loop contains loading of Input and output from file to testing din and dout. Then clear signal is kept low for 1 ns to load values and then after 12 clock cycles output is checked against the output from file. If they come to a difference, then simulation is interrupted, and error message is displayed. If all the testcases are passed, then testbench checks for case when clear signal does not go low after 13 clock cycles, t_dout is checked against f_dout as they should be equal. Then it is checked for the case when clear signal comes to low in between of a encryption/decryption process, and checked against input and output as at this point all three should be equal i.e t_din = f_din = t_dout. If this test case is also passed then we check it for the case when clear signal is left at low (0) then t_din = f_din = t_dout should be true through out the case. If this test is also passed, then All test Passed is displayed and simulation ends.

3. Video Link

https://youtu.be/tHNV7yJFBOk