LAB 4 REPORT

Many-to-one linear-feedback shift register (LFSR),

One-to-many linear-feedback shift register (LFSR),

Content-addressable memory (CAM) design, Scan chain design,

Built-in self test, Mealy machine sequence detector

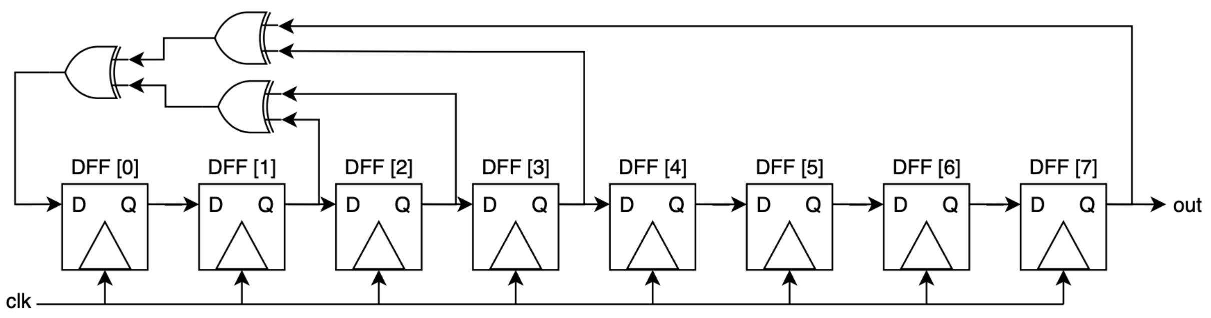
Team 37:

1. 徐美妮 Mary Madeline Nicole 109006205
2. 林之耀 Kevin Richardson Halim 109006277

I. Block diagrams and state transition diagrams

$$\begin{tikzcd}
bd \arrow[r] & 7b \arrow[r] & f6 \arrow[r] & ed \arrow[r] & db \arrow[r] & b7 \arrow[r] & 6f \arrow[r] & df \arrow[r] & be \arrow[r] & 7c \arrow[r] & \ldots
\end{tikzcd}$$

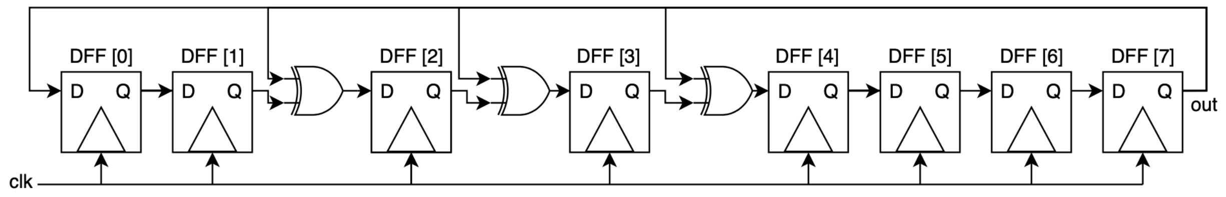
This is the state transition diagram of the DFFs in LFSR for the first ten states after rst_n is raised to 1’b1 in hexadecimal.



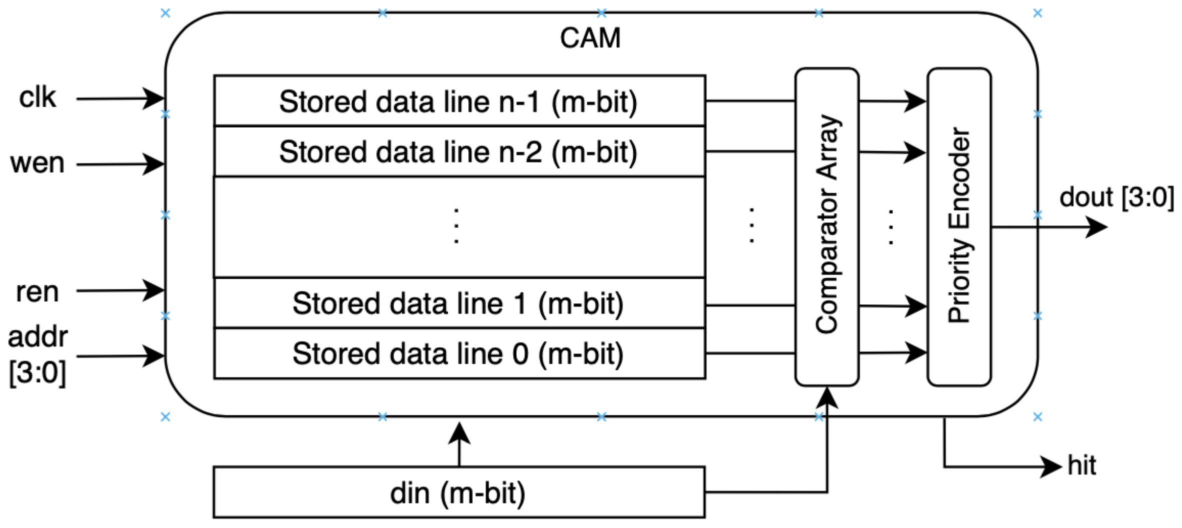
1.1 Many-to-one linear-feedback shift register

$$\begin{tikzcd}
bd \arrow[r] & 67 \arrow[r] & ce \arrow[r] & 81 \arrow[r] & 1f \arrow[r] & 3e \arrow[r] & 7c \arrow[r] & f8 \arrow[r] & ed \arrow[r] & c7 \arrow[r] & \ldots
\end{tikzcd}$$

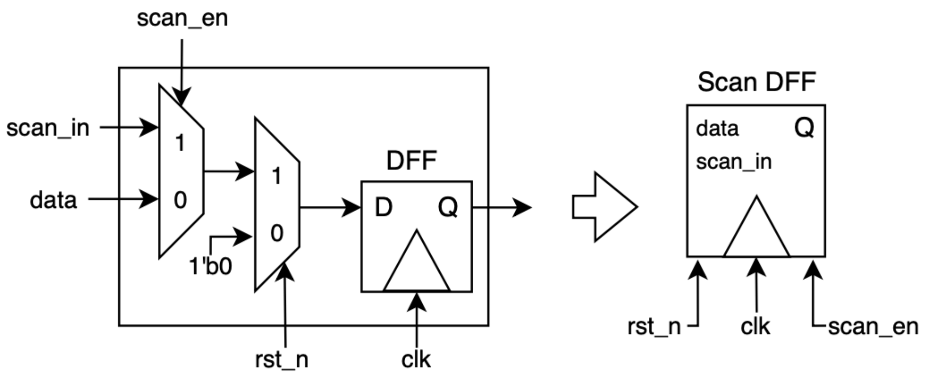
This is the state transition diagram of the DFFs in LFSR for the first ten states after rst_n is raised to 1’b1 in hexadecimal.

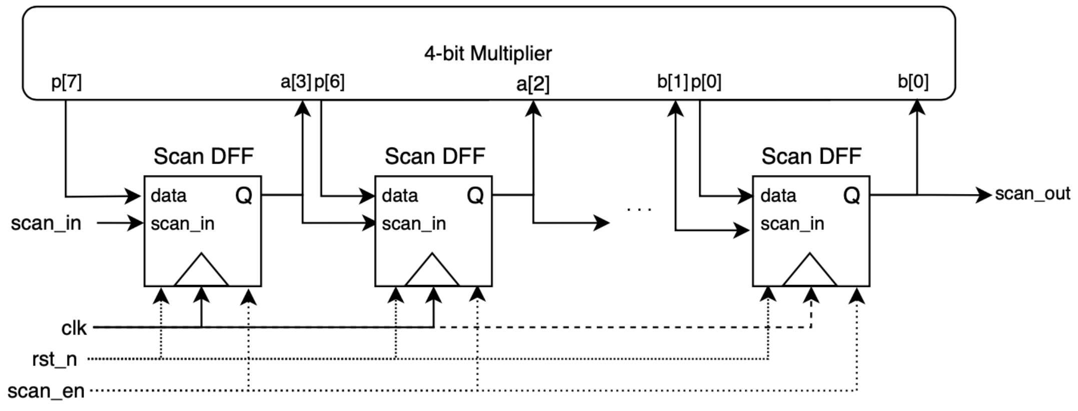


1.2 One-to-many linear-feedback shift register

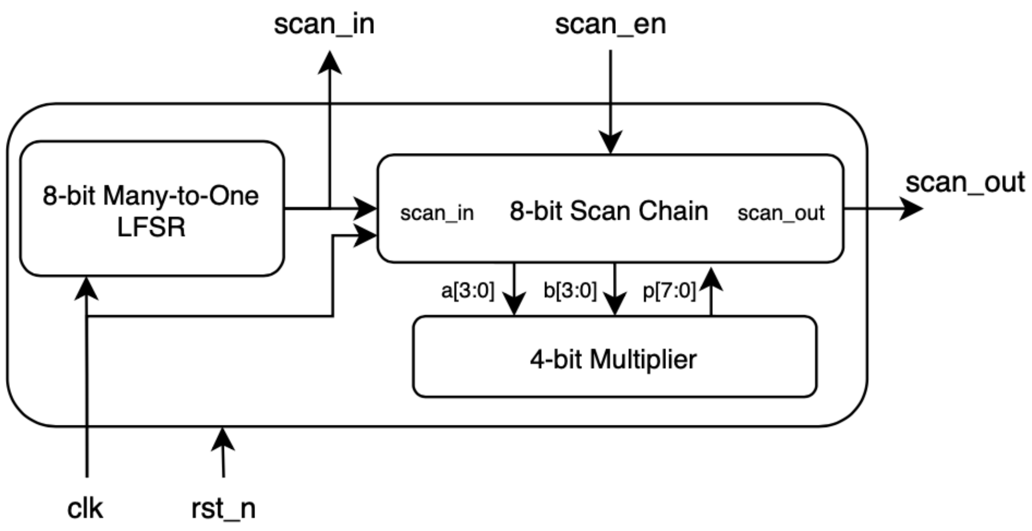

CAM
clk
wen
ren
addr [3:0]
Stored data line n-1 (m-bit)
Stored data line n-2 (m-bit)
Stored data line 1 (m-bit)
Stored data line 0 (m-bit)
Comparator Array
Priority Encoder
dout [3:0]
din (m-bit)
hit

1.3 Content-addressable memory (CAM) design

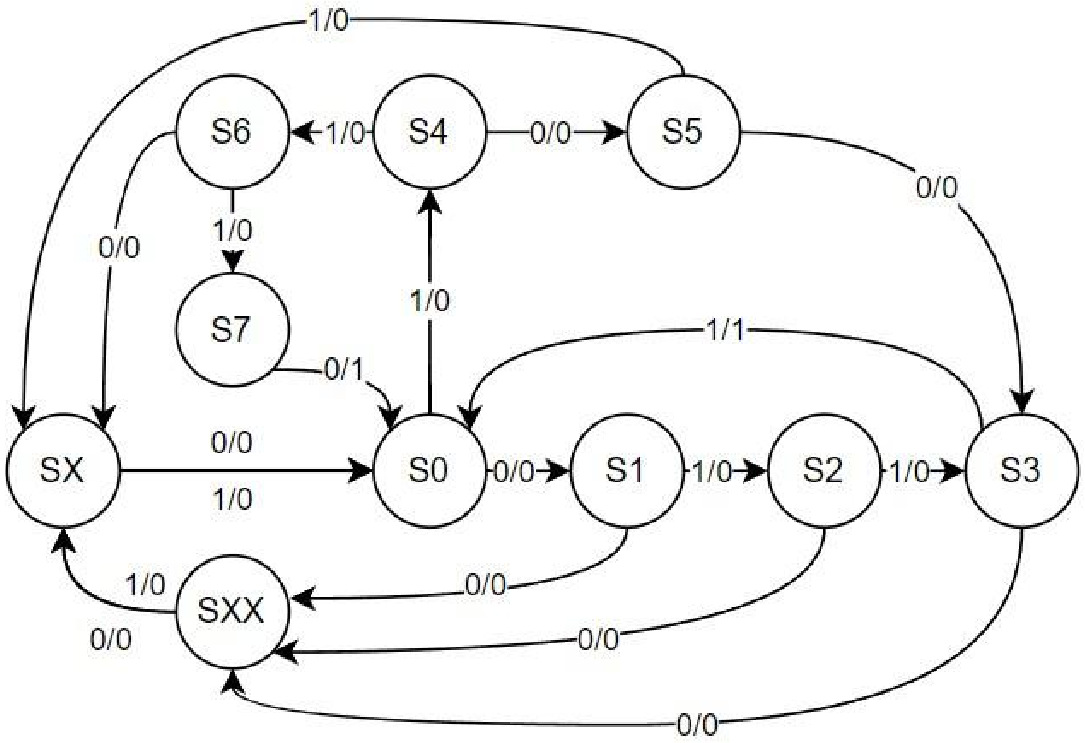


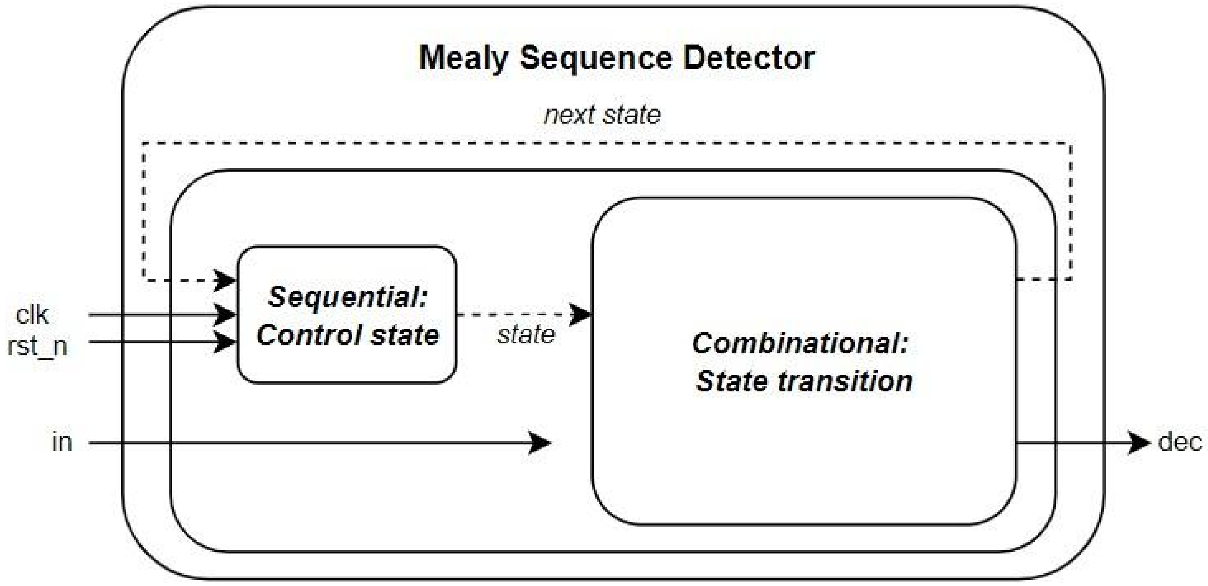


1.4 Scan chain design



1.5 Built-in self test





1.6 Mealy Machine sequence detector

II. Design Process

First, start by making the block diagrams and state transition diagrams of the design.

Second, the modules are written in verilog. We used combinational and sequential circuits to create our modules. Updates of the states are done sequentially, and the others are done combinatorially. Then we check all the spelling, syntax, and errors.

Third, the testbench is built. The testbench is very useful to check whether or not the module works as it should. The use of display and task when making the testbench makes it easier to spot mistakes at our module, by using the CAD server. We made the testbench according to the input changes of each module. Critical conditions, or edge cases were also made, relying on when the module would break.

Finally, debugging and reiterating each line to fix some bugs that were faced. Such as wrong I/O, swapped variables, wrong naming, wrong behavior, typos, etc.

Modules:

1. Many-to-one & One to many linear-feedback shift register

In a shift register, a cascade of flip flops are used, where the output of one flip-flop is connected to the input of the next. They share a single clock signal, which causes the data stored in the system to shift from one location to the next. Basically, the input bit of the register is a linear function of its previous state. The LFSR is a pseudo random output generator.

We must never clear the LFSR. If we reset the DFF's into 8'd0. It will always remain in that state and output a sequence of 0 from that point. Because when an XOR gate has zero as its input, the output will always be zero.

In many to one, values of many registers are used to determine the new value for the first. In one to many, the output of the last register is used to determine the values of many registers, therefore it is called one to many. The change in the values of each register is sequential, however the calculation of the gates are combinational.

2. Content-addressable memory (CAM) design

Content-addressable memory is a type of memory that compares the input data with the currently loaded contents of the memory block, and generates a given output which is the address of the content of the memory itself.

This module contains memory, has 5 inputs (*clk, wen, ren, addr[3:0], din[7:0])* and 2 outputs (*dout[3:0] and hit)*. The idea of the design is to store the input to the CAM everytime the write is enabled, and when the read is enabled, we search the memory if there's any matching data with the input, then it will output the address of the data in the CAM. When there are multiple matches, the largest address will be the output.

We check whether the current input is inside the memory that we currently have by using a comparator array. When read is enabled, every value in the storage module will be compared. The output of the comparator is 16 bits, where each bit is the address of the storage. If there is no matching data in the memory, the output will be zero.

Then we used priority encoder to choose the larger address among the memory addresses that has the current input.

3. Scan chain design

Scan chain is a technique used in design for testing. This module is a collection SDFF with a module that needs to be tested, and has 2 1-bit inputs (*scan_in* and *scan_en*) and 1 1-bit output (*scan_out*).

SDFF itself is a DFF modified module. Named SDFF since it's a DFF module with extra input selection of input data, the selection is either *data* or *scan_in*. Input *data* is wired with the output from the tested module) and input *scan_in* (wired with the output from SDFF before it or the input from scan chain design itself).

The scan chain design works like a shift bit, since SDFF's scan_in is wired to the SDFF's scan_out before it. The most front SDFF's scan_out would be the Scan chain's scan_out itself and the most back SDFF's scan_in would be the Scan chain's scan_in itself. The SDFF captures the value from SDFF before it when scan_en is set to be 1'b1 and the SDFF captures the value given by the module you want to test. The shift and capture happened on every positive edge clock cycle.

How to use this technique? Input the bit as many as SDFF you have (8 in this module) start from the least significant and immediately assign the *scan_en* to 1’b0 for a clock cycle and assign it back to 1’b1 to captured the value from the module you want to test (the module inside). The scan chain design will output the captured value on every positive edge clock cycle the output should be emitted from the least significant bit.

4. Built-in self test

The Built-in self test (BIST) consists of 3 inputs (*clk, rst_n, scan_en)* and 2 outputs (*scan_in, scan_out)*. This module provides its own input that wants to be tested and uses the previous module of the Many to One LFSR with a little modified output and the Scan Chain design.

The LFSR in this module generates a new pseudo random output every clock cycle. This module has been modified, so the output is 1 bit length from the most significant bit. The random generated output will be wired with *scan_in* and connected to the input on an 8-bits Scan Chain module.

The Scan Chain design module will get a random input for the number and then after receiving 8 bits (4 bits for the first number and 4 bits for the second number) for 8 positive clock cycles, the *scan_en* will be set to 1’b0 for a clock cycle and return 1’b1 after it to catch the output generated. The output then will be wired to the BIST module’s output *scan_out* for 8 positive clock cycles since the value 4-bits multiplier’s output is 8 bit number.

This module has its own input generator, therefore it’s called Built-in self test module. It has the input generator, which is LFSR and the Scan Chain design to do the input output and binded to a module that wants to be tested, which is a 4-bit Multiplier for this experiment.

5. Mealy Machine Sequence detector

The mealy machine sequence detector sets its output combinationally, because the output does not change depending on the clock cycle. We designed the mealy machine sequence detector by first determining the states per each input. States from state 0 to state 3 were predetermined on the question, so we completed the state diagram as shown above in Part I. There are a total of 10 states, from 0 to 7 and two extra states created, which were named SX and SXX, to be the dummy state of the failed sequence.

Because the sequence that we want to detect are only 0111, 1001, 1110, when the inputs are not according to the states in the middle of the operation, the next state to that current state will be either state X or state XX. In this case, state X's next state will be state 0 regardless of the input, and state XX's next state will be state X, also regardless of the input. These states, X and XX do not care what the input is, but will output 0 because when any of the previous states go to these states, it indicates that the sequence input is wrong and not according to the four bit sequence given. The dec will only output 1 at the right sequence before going back to the initial state.

Our module is divided into two, combinational and sequentially. Reset and updating the state to the next state was done sequentially, according to the positive clock trigger. While updating which state will be the next state according to the input was done combinationally.

III. List of Contributions

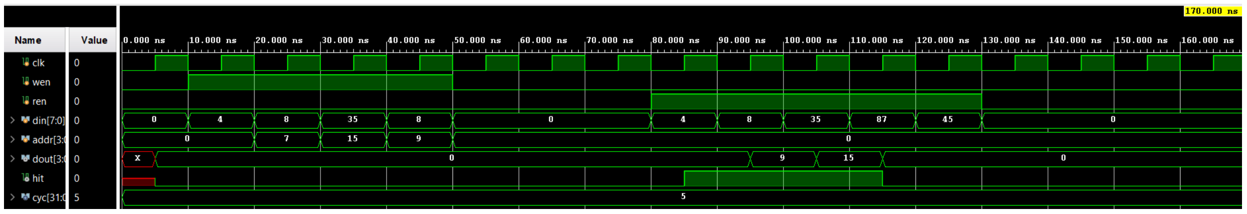
徐美妮 Mary Madeline Nicole 109006205's contributions, such as :

- Design and write module Mealy Machine sequence detector,
- Design and write testbench Mealy Machine sequence detector,
- Design and write testbench Content-addressable memory (CAM),
- Design and write testbench Scan Chain design,
- Design and write testbench Built in self test,
- Draw block diagrams and state transition diagrams,
- Simulate and synthesis modules in Vivado,
- Write the report.

林之耀 Kevin Richardson Halim 109006277's contributions, such as :

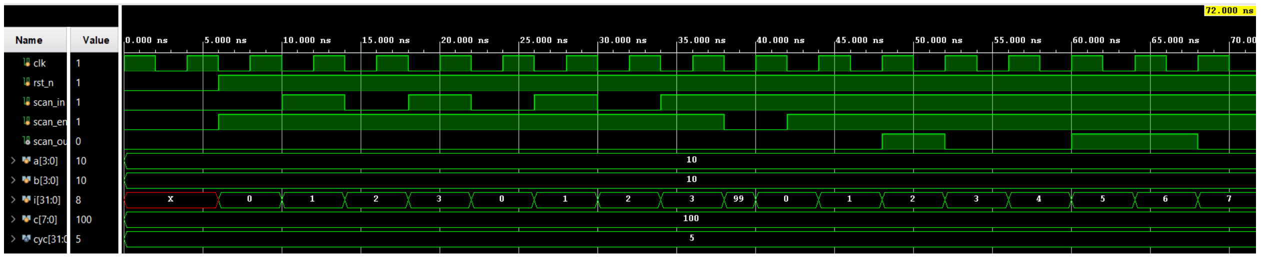
- Design and write module Content-addressable memory (CAM) ,
- Design and write module Scan Chain design,
- Design and write module Built in self test,
- Design and write testbench Content-addressable memory (CAM),
- Design and write testbench Scan Chain design,
- Design and write testbench Built in self test,
- Simulate and synthesis modules in Vivado,
- Write the report.

IV. Design Test



4.1 Content-addressable memory (CAM) design

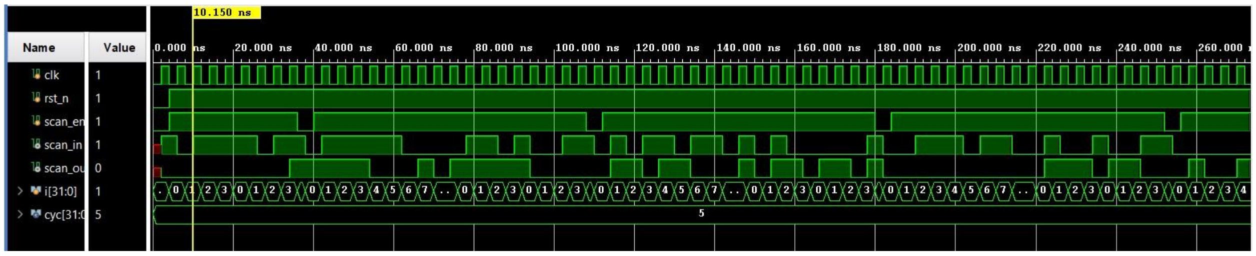
We tested this module by changing the values of ren and wen. Also playing around the values of the 8 bit input din, so that we can make sure that our program is correct. The testbench inputs values to the storage modules, where some are doubled to check whether our CAM design can output the address correctly. According to the waveform that was generated, our program is right.



4.2 Scan chain design

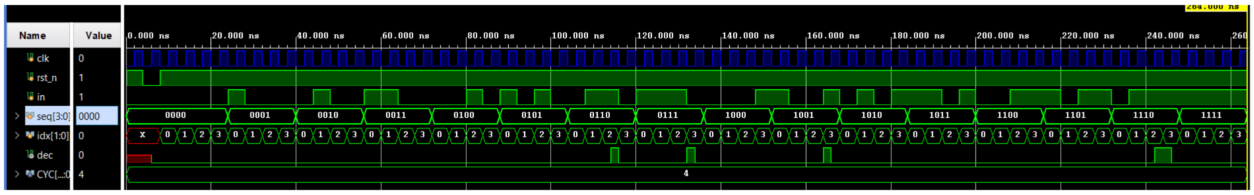
For the Scan chain design, we created a testbench to see whether the imputed value gives the expected output value. Since this experiment is tested with a 4-bits Multiplier module. The output should be exactly the amount of the first number times the second number.

We also created a task Test to write errors when run with ncverilog. We also added more displays at the initial block to easier keep track of the test cases that are running.



4.3 Built-in self test

This module can actually test itself. So the testbench we created to test this module is just updating the clock cycle, resetting the module and enabling the scan during the first negative edge trigger. The waveform above shows that our module outputs the correct answer. For this experiment we set the scan_en for after every two 4 positive clock cycles for two 4-bits input and before 8 positive clock for cycles 8-bits output.



4.4 Mealy Machine Sequence detector

We created a testbench for the mealy machine sequence detector by creating a new reg, called sequence to be the pattern that will be tested, and idx for the index of which position of the pattern will be tested now. First, we reset, then tested the module. A repeat block of (2**4) was used. First we determine the index of the sequence that wanted to be tested, because we are testing the sequence from left to right, we check the most significant bit first, to the least significant bit of the address index of the sequence. We also add the sequence pattern by 1, to be able to test all possible combinations from a 4 bit number. We also created a task Test to make debugging in ncverilog easier.

V. What we have learned from Lab 4

- ➔ Create a mealy sequence detector that can detect multiple sequences.
- ➔ We can create a Scan Chain Design to test our module with only 1 bit input and will output the result as the clock cycle positive edge triggered.
- ➔ Creating and implementing LFSR as a pseudo-random generator inside the Built-in self test.