Hardware Design

**Lab 3 Report**

Sequential Circuits

**Team 01**

112062122 謝佳晉 112062144 范升維

Table of Contents:

[Advanced Q1. 4-bit Ping-Pong Counter 1](#_Toc179474120)

[Advanced Q2. First-In First Out (FIFO) Queue 4](#_Toc179474121)

[Advanced Q3. Multi-Bank Memory 6](#_Toc179474122)

[Advanced Q4. Round-Robin FIFO Arbiter 9](#_Toc179474123)

[Advanced Q5. Exhaustive testbench design 10](#_Toc179474124)

[FPGA: 4‑bit Parameterized Ping‑Pong Counter 10](#_Toc179474125)

[Testbenches 11](#_Toc179474126)

[A. 4-bit Ping-Pong Counter 11](#_Toc179474127)

[B. First-In First Out (FIFO) Queue 11](#_Toc179474128)

[C. Multi-Bank Memory 11](#_Toc179474129)

[D. Round-Robin FIFO Arbiter 11](#_Toc179474130)

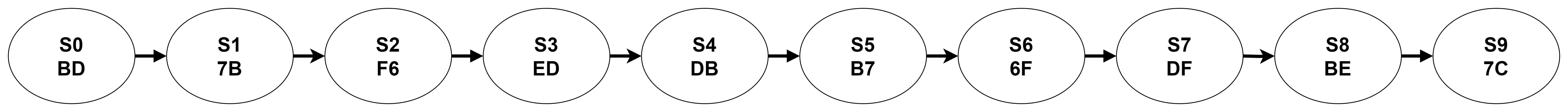
[E. 4-bit Parameterized Ping-Pong Counter 11](#_Toc179474131)

[What have we learned from Lab 3? 11](#_Toc179474132)

[Contributions 11](#_Toc179474133)

# Basic Q3. Many-to-one LFSR

1. State transition diagram:

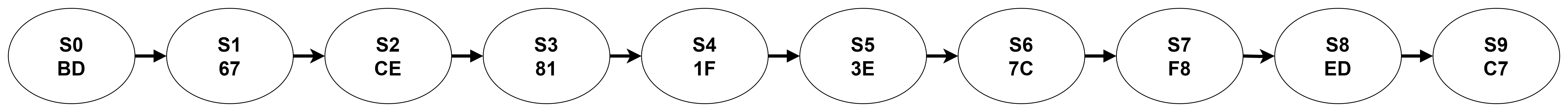


1. What happens if we reset the DFFs to 8’d0?

If we reset the DFFs to 8’d0, the LFSR will keep its state forever. Because there is no 1’b1 in the original state, any of the XOR is impossible to output 1’b1, so the next state will be the same as the original state. In consequence, the LFSR will keep on shifting, but the states will all be the same, as it’s stopped.

# Basic Q4. One-to-many LFSR

1. State transition diagram:



1. What happens if we reset the DFFs to 8’d0?

Just like the previous problem, because there is no 1’b1 in the LFSR and we only use XOR operations in the LFSR, the input of all the DFFs will keep being 1’b0. Therefore, the LFSR will also keep on shifting, but the states will all be the same, as it’s stopped, until we change the reset value.

# Advanced Q1.

# Advanced Q2.

# Advanced Q3. Built In Self Test (BIST)

In Advanced Question 3, we are required to construct a BIST module. In the module, we combine the Many To One LFSR in basic question 3 and Scan Chain Design in the previous question together.

一張含有 文字, 螢幕擷取畫面, 設計 的圖片

自動產生的描述

▲ Figure 3.1: DFF\_w\_rst

一張含有 文字, 螢幕擷取畫面, 字型, 設計 的圖片

自動產生的描述

▲ Figure 3.2: 8bit Many To One LFSR

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

▲ Figure 3.3: BIST

* 3 inputs: *clk, rst\_n, scan\_en*
* 2 output: *scan\_in, scan\_out*

We first modify the original DFF by adding a MUX to it in order to reset the whole LFSR’s value to 8’b10111101. when rst\_n is 0. We call the modified DFFs DFF\_w\_rst(Figure3.1).

After we have DFF\_w\_rst, we can use them in the LFSR(Figure 3.2) to have the original DFFs in them take different values according to rst\_n’s value. For the leftmost(1st) DFF\_w\_rst’s input value when rst\_n = 1, we do several XOR operations on the 2nd, 3rd, 4th, 8th DFF\_w\_rsts’ output values to meet the demand in spec. For the scan\_in signal, it will take in the 8th DFF\_w\_rst’s output value directly.

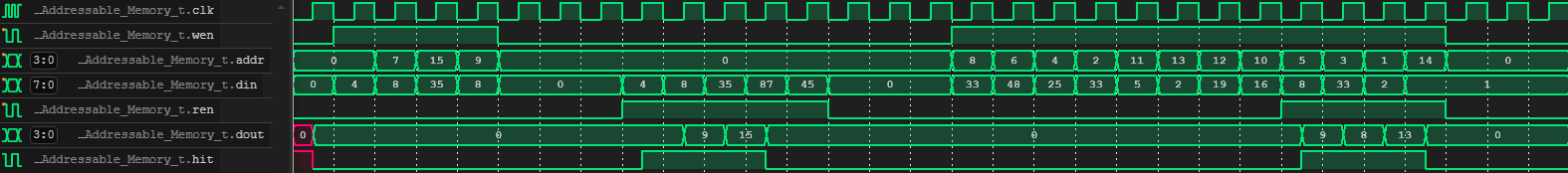
For module Scan Chain Design in BIST(Figure 3.3), it will take on the values of rst\_n, clk, scan\_en, and scan\_in output by LFSR and output scan\_out’s value after the internal operations it have done.

# Advanced Q4.

# FPGA: BIST

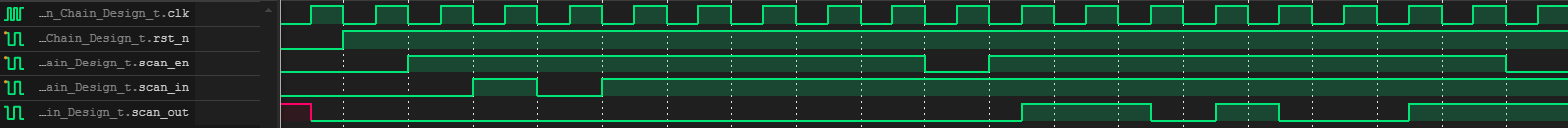
# Testbenches

## A. Content Addressable Memory (CAM)

In this testbench, we combine the waveform on the spec and some special cases:

1. same data in different stored data lines
2. data matching fails
3. ren and wen both positive simultaneously

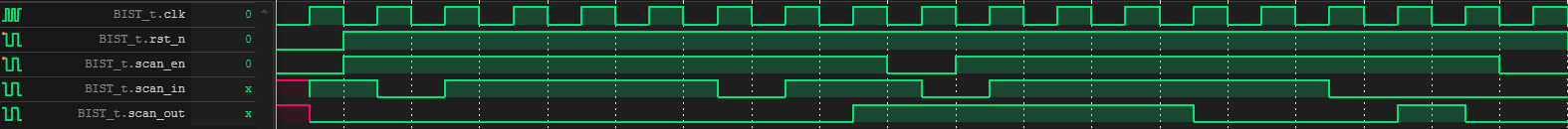
## B. Scan Chain Design



We have the same waveform as that in the spec.

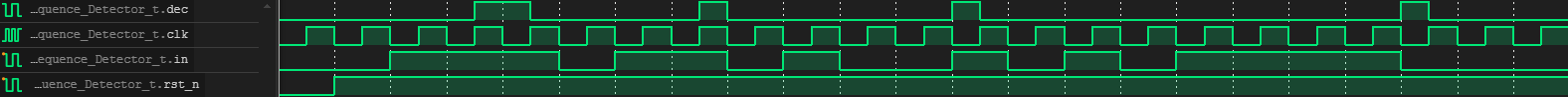
a = 4’b1111 = 4’d15, b = 4’b1010 = 4’d10 => p = a\*b = 8’b10010110 = 8’d150

## C. Built In Self Test (BIST)



In this testbench, we modify the rst\_n and scan\_en according to the working pattern of Scan Chain Design(reset->scan in->capture->scan out). We can see that the sequence of scan\_in output is 10111101, which is the same as the reversed reset value of LFSR.

## D. Mealy machine sequence detector



In this testbench, we reference the waveform on the spec and add the sequence, 1110, which is not included in the spec waveform to test if all the three sequence can be deteced.

# What have we learned from Lab 4?

# In Lab 4, we delved into the topic of "finite state machines." Due to the negative experience from the previous lab, this time we planned the circuit in advance before writing the code. We divided the modules and then assembled them. We found that this approach not only reduced the chances of making mistakes and improved our efficiency in coding and circuit design, but it also gave us a better understanding of how the entire circuit operates. We didn’t need to rack our brains trying to figure out the circuit design after finishing the code. Through this lab, we gained a deeper understanding of Moore and Mealy machines, became more proficient in writing sequential circuits, data flow modeling, and behavioral modeling, and applied these skills to state transitions of Moore and Mealy machines.

# Contributions

謝佳晉：

wrote Verilog modules: Q1, Q2, Q4

wrote testbenches: Q1, Q2, Q4, Q5

performed simulation: Q1, Q2, Q4, Q5

drew diagram: basic Q1, Q2

wrote report: basic Q3, Q5, tbQ1, tbQ2, tbQ4

made FPGA demonstration

范升維：

wrote Verilog modules: Q1, Q2, Q3, Q4

wrote testbench: Q3

performed simulation: Q1, Q2, Q3, Q4 simulation on Vivado

drew diagram: ALL (basic Q1, Q1, Q2, Q3, Q4)

wrote report: basic Q1, basic Q3, Q1, Q2, Q3, Q4, tbQ3, what we learned

organized whole report

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | b-Q1 | b-Q3 | Q1 | Q2 | Q3 | Q4 | Q5 | FPGA | What we learned |
| wrote Verilog modules |  |  |  |  |  |  |  |  |  |
| wrote testbenches |  |  |  |  |  |  |  |  |  |
| performed simulation |  |  |  |  |  |  |  |  |  |
| drew diagram |  |  |  |  |  |  |  |  |  |
| wrote report |  |  |  |  |  |  |  |  |  |
| wrote report tb |  |  |  |  |  |  |  |  |  |

|  |
| --- |
| Both |
| 謝佳晉 |
| 范升維 |