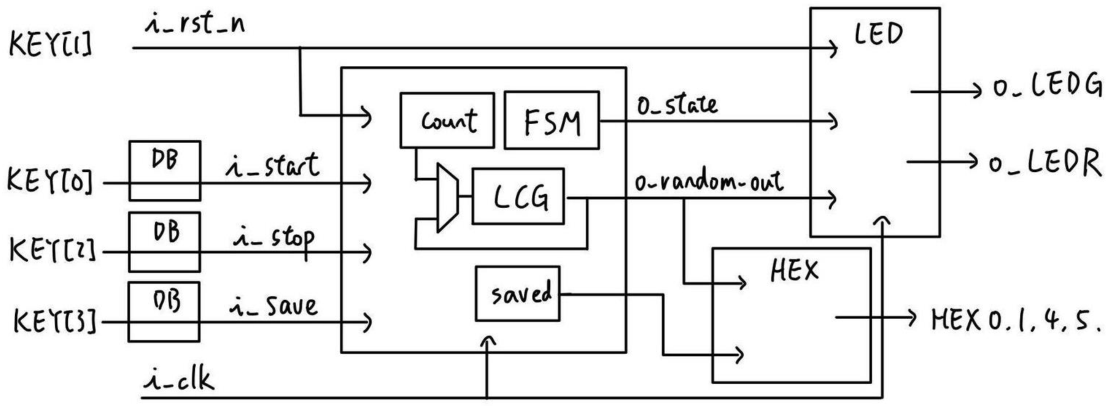
Lab 1 Report (Group 7)

I. Introduction

In this lab, we implement a simple random number generator (RNG) with additional features, such as using the LED to indicate the state and the number, and an additional 7-segment HEX decoder to load and store values.

II. System Architechture

A block diagram of our design is presented below:



We use the following modules:

A. Debounce

For the user controlled input, we need to add the debounce structure in order to stabilize the input signal to the main module.

B. LCG

This module takes an input x and outputs the number $11x + 29$ (mod 16), which can output a randomized sequence if we feed the output back to the input. Note that for the starting number, we'll take the number from the counter, which corresponds to the waiting time at the beginning that can be viewed as a random number at the beginning.

C. Top

This is the main structure of our design. We implement the finite state machine, counter and a register used to store and load specific value.

D. LED

This is a module used to control the LED lights according to the number on the screen and the state of the main module. For the red LEDs, in IDLE state, there are always two consecutive lights turned on and gradually shifted left. In the other states, 4 consecutive LEDs are the binary representation of o_random_out. For the green ones, different patterns of the 8 lights represent the current state of the FSM.

For example, OOXXXXXX means state = FAST, XXOOXXXX means state = MEDIUM…… Details are omitted here, and further information can be checked in the code.

E. *HEX*

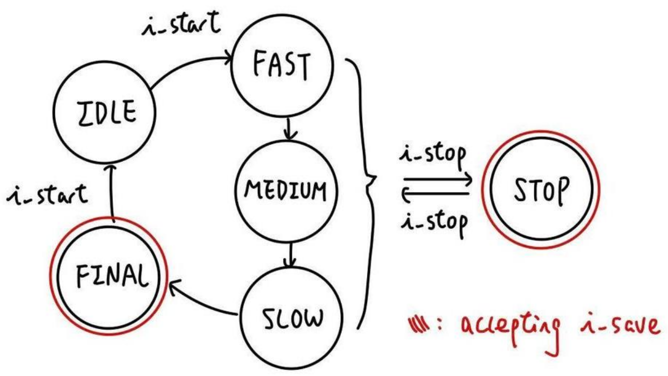
This is a module used to control the display of the screen, since we want to use another screen to display the stored value in addition to the random value. We modify the given module in order to extend the functionality, such that for some particular input, the hex decoder will display nothing by output 7'b1111111.

III. File Structure

File structure under the team07_lab01 directory:

```
├─team07_lab01_report.pdf
└─src
    ├─Top.sv
    └─DE2_115
        ├─DE2_115.sv
        ├─Debounce.sv
        ├─LED.sv
        └─SevenHexDecoder.sv
```

IV. State Control



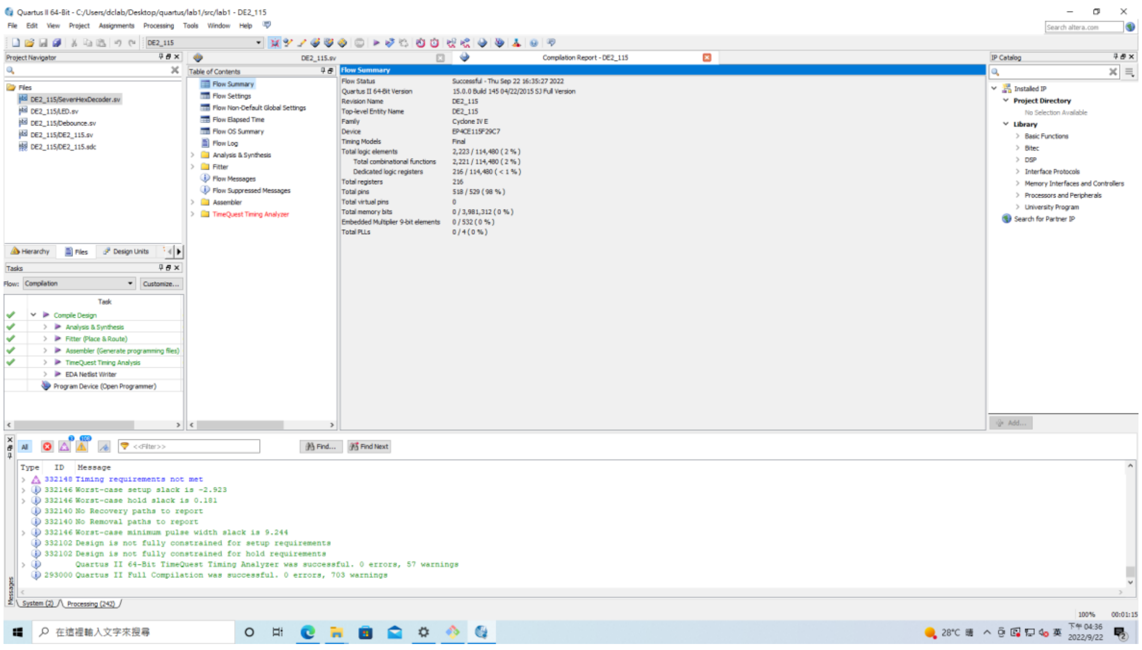
For the design, we have four user-controlled inputs, which correspond to four buttons on FPGA, respectively. Each of the input can trigger the state machine to another state:

- *i_rst_n*:
  This is used to reset everything, including transitioning back to IDLE state.
- *i_start*:
  This is used to start generating the random number if it's in IDLE state. After the process is finished, the machine will end up in the FINAL state. If it is triggered again, the state will be back to IDLE state.

- *i_stop*:
  This is used to stop the RNG during the process. If it is triggered again, the process continues from where it is halted.
- *i_save*:
  This is used to store the value to the register if the machine is in the STOP or FINAL state. Also, it can load the value from the register if the machine is in the IDLE state.

V. SCREENSHOTS

A. *Fitter Summary*

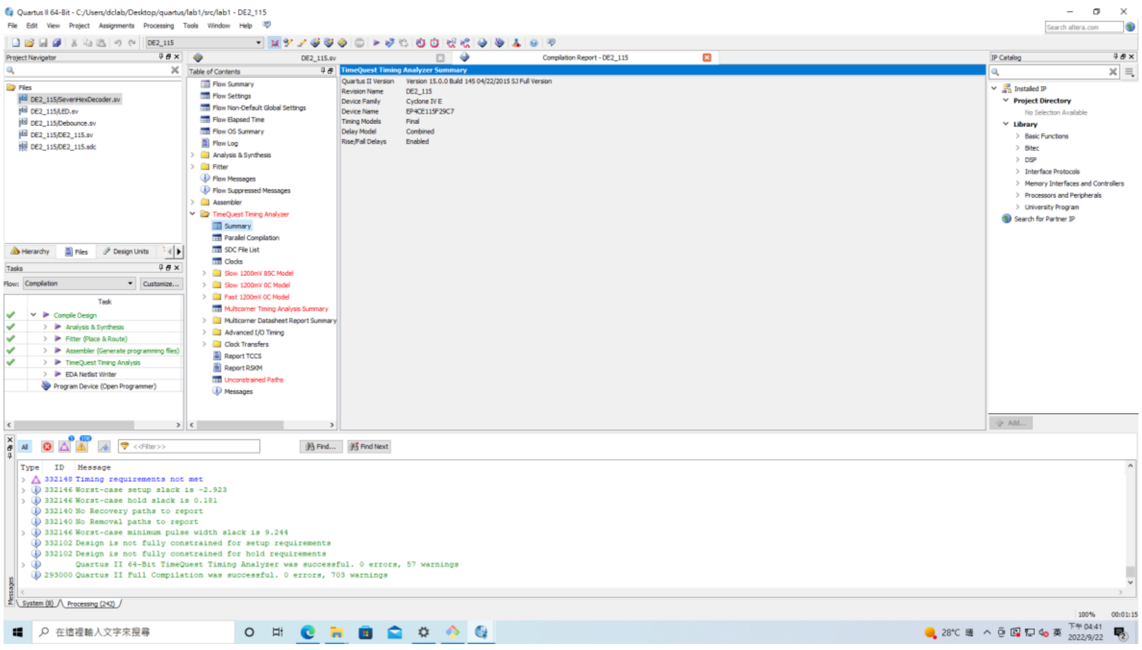


Flow Summary

Flow Status	Successful - Thu Sep 22 16:35:27 2022
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Full Version
Revision Name	DE2_115
Top-level Entity Name	DE2_115
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	2,223 / 114,480 (2 %)
Total combinational functions	2,221 / 114,480 (2 %)
Dedicated logic registers	216 / 114,480 (< 1 %)
Total registers	216
Total pins	518 / 529 (98 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

```
Type  ID     Message
      332148 Timing requirements not met
      332146 Worst-case setup slack is -2.923
      332146 Worst-case hold slack is 0.181
      332140 No Recovery paths to report
      332140 No Removal paths to report
      332146 Worst-case minimum pulse width slack is 9.244
      332102 Design is not fully constrained for setup requirements
      332102 Design is not fully constrained for hold requirements
             Quartus II 64-Bit TimeQuest Timing Analyzer was successful. 0 errors, 57 warnings
      293000 Quartus II Full Compilation was successful. 0 errors, 703 warnings
```

B. *Timing Analyzer*



TimeQuest Timing Analyzer Summary

Quartus II Version	Version 15.0.0 Build 145 04/22/2015 SJ Full Version
Revision Name	DE2_115
Device Family	Cyclone IV E
Device Name	EP4CE115F29C7
Timing Models	Final
Delay Model	Combined
Rise/Fall Delays	Enabled

```
Type  ID     Message
      332148 Timing requirements not met
      332146 Worst-case setup slack is -2.923
      332146 Worst-case hold slack is 0.181
      332140 No Recovery paths to report
      332140 No Removal paths to report
      332146 Worst-case minimum pulse width slack is 9.244
      332102 Design is not fully constrained for setup requirements
      332102 Design is not fully constrained for hold requirements
             Quartus II 64-Bit TimeQuest Timing Analyzer was successful. 0 errors, 57 warnings
      293000 Quartus II Full Compilation was successful. 0 errors, 703 warnings
```

VI. DISCUSSIONS

- Computer in the lab malfunctions all the time…
- We write a LOT of non-synthesizable codes, which need to be modified again…
- We use "Linear congruential generator" to generate the random number. However at first we poorly choose "a" and "b" , so we generate repeated output. We found that random output is sensitive to the choice of "a" and "b", and we finally choose a = 11, b= 29 to achieve random output.
- It is hard to spread the workload clearly to everyone, so most of time we write the code together in the classroom instead of doing our own parts at home. But we think we still need to know how to cooperate more efficiently like split the workload clearly in the future.