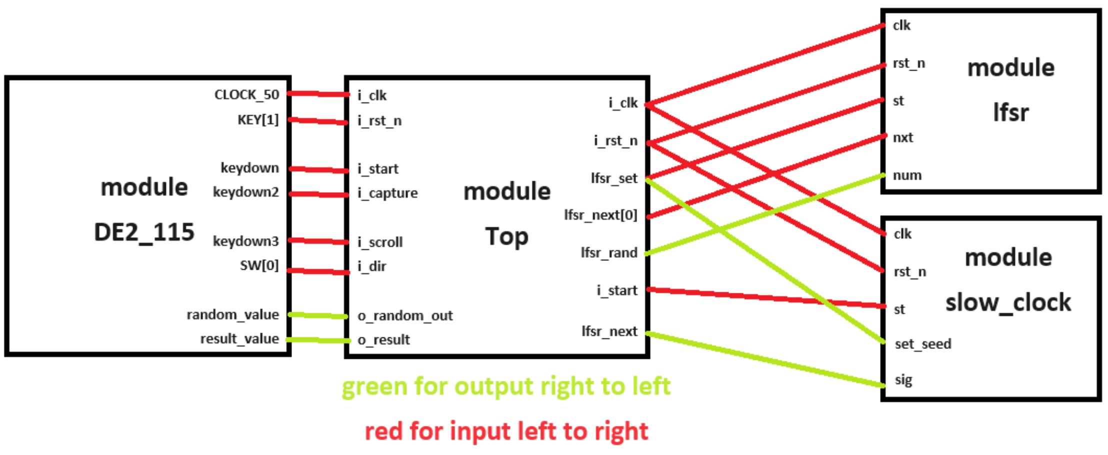
Team09_Lab1_Report

File Structure:

- team09_lab1/team09_lab1_report : This file contains information about the source code in the directory and the instructions of lab1.
- team09_lab1/src/Top.sv : We have implemented all the requirements of lab1 in this file.
- team09_lab1/src/DE2_115 : For the files in this folder, we have only modified DE2_115.sv to implement the bonus of lab1.

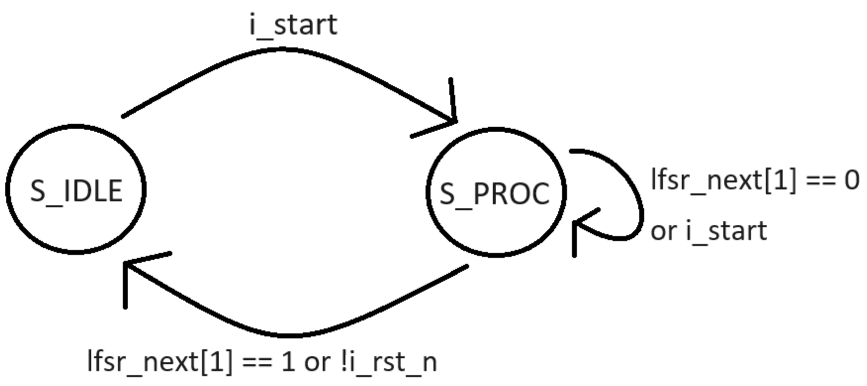
System Architecture:



- Module *DE2_115* handles all IO from/to the FPGA.
  - Pass the clock (50MHz) and the state of Switch 0 to *Top*.
  - Pass the debounced release signal of buttons 0, 2 and 3 to *Top*.
  - Pass the random value and saved values from *Top* to 7-segment decoder.
- Module *Top* handles managing the submodules of the algorithm.
  - Start *slow_clock* when start is pressed.
  - Pass the control signal from *slow_clock* to *lfsr*.
  - Pass the random value and the saved past values to *DE2_115*.
- Module *slow_clock* handles the timing for setting *lfsr* seed and generating the next number.
- Module *lfsr* handles the algorithm of generating the next pseudo-random number.

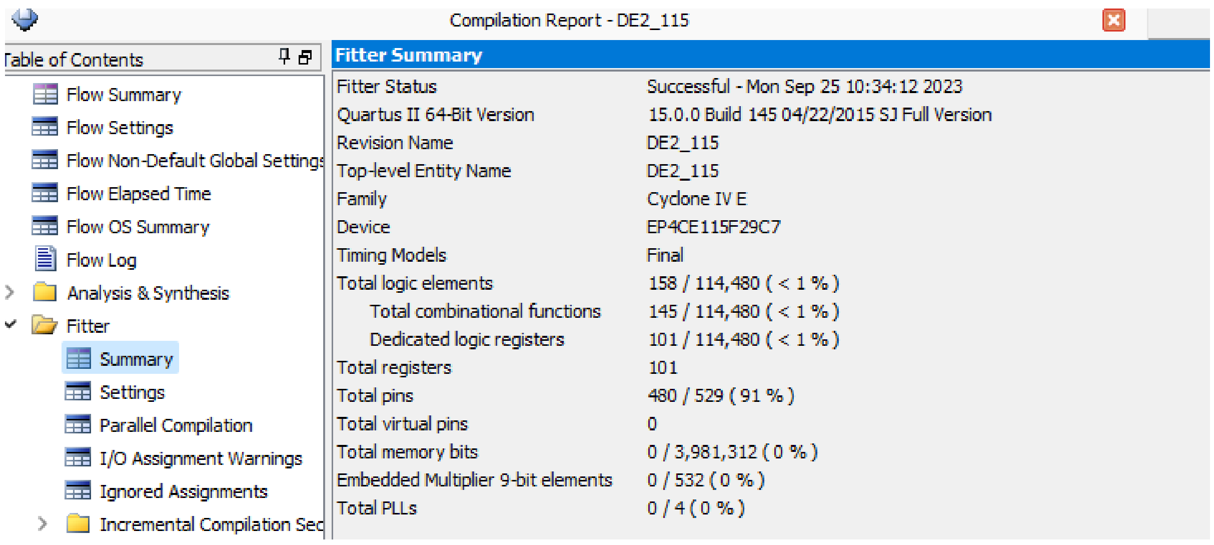
Hardware Scheduling:

- Module *Top*:



    - When *i_capture* is high, save current value from *lfsr* to internal 4x 4-bit number register *results*.
        - The oldest value is erased when a new value is saved.
        - This implements the function of saving the generated value while it is still running, and when it has finished running.
    - When *i_scroll* is high, increment/decrement *i* (2-bit MUX selection index, default 0) value by 1 depending on *i_dir*. Wrap around when overflow occurs.
        - *i_dir* high: increment; *i_dir* low: decrement.
    - Send *results[i]* to *DE2_115* as *result_value*.
- Module *slow_clock*:
    - When *st* is high and the module is idling, set state to running and send *set_seed* signal to the generator.
    - While running, increment *count_c* until it hits the current *count_lim*.
    - When hitting *count_lim* for the 0~6-th times, send a next signal to the generator.
    - When hitting *count_lim* for the 7-th time, send a next signal to the generator, set state to idling, and send a finished signal to *Top*.
- Module *lfsr*:
    - Implement a linear feedback shift register as a pseudo-random number generator.
    - Always increment the internal *seed* counter every clock cycle.
    - When *st* is high, push *seed* to the shift register *rand_c*.
        - This guarantees the seed to be different when the button is pressed almost every time as the *seed* counter increments 50 million times every second.
    - Generate the next number in sequence *rand_n* by right-shifting *rand_c* 1 bit, and setting the lowest bit to be the xor of *rand_c[15]*, *rand_c[13]*, *rand_c[12]*, and *rand_c[10]*.
    - When *st* is low and *nxt* is high, push *rand_n* to *rand_c*.
    - Pass the lowest 3 bits of *rand_c* as the generated pseudo-random number to *Top*.

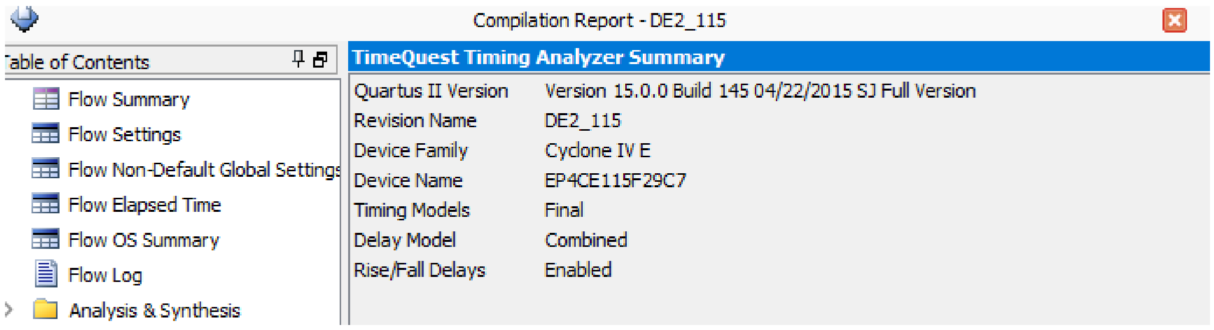
Fitter Summary:



Fitter Summary

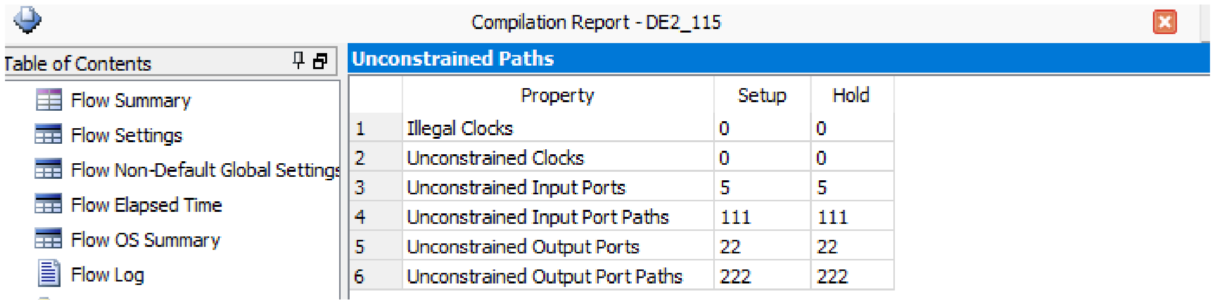
Fitter Status	Successful - Mon Sep 25 10:34:12 2023
Quartus II 64-Bit Version	15.0.0 Build 145 04/22/2015 SJ Full Version
Revision Name	DE2_115
Top-level Entity Name	DE2_115
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	158 / 114,480 (< 1 %)
Total combinational functions	145 / 114,480 (< 1 %)
Dedicated logic registers	101 / 114,480 (< 1 %)
Total registers	101
Total pins	480 / 529 (91 %)
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 %)

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



Unconstrained Paths

	Property	Setup	Hold
1	Illegal Clocks	0	0
2	Unconstrained Clocks	0	0
3	Unconstrained Input Ports	5	5
4	Unconstrained Input Port Paths	111	111
5	Unconstrained Output Ports	22	22
6	Unconstrained Output Port Paths	222	222

Problem & Solution:

1. Error (10174): system function "$random" is not supported for synthesis.
    a. 有$的語句都不能合成跑模擬，只能自己寫一個偽隨機算法
2. Error (10166): always_comb construct does not infer purely combinational logic.
    a. 有一些分支沒有指派到output
3. Error (10028): Can't resolve multiple constant drivers for net
    a. 不要在兩個 always block 內 drive 同一個 register/wire，應該寫在一個always block內作邏輯處理
4. 隨機數變化太快，無法分辨
    a. 在 slow_clock 內用 counter，clock 變化達到一定數字時再觸發 LFSR

Conclusion & Suggestion:

Having finished lab1, we have become more acquainted with writing Verilog. We have also learnt the importance of naming variables in a readable manner; otherwise it would be difficult for co-workers and ourselves to debug the program.
During the process, we have realized that even by truncating a 16-bit value generated by the LFSR, the quality of the generated number is still quite bad, since for any given 4-bit value, we know explicitly the first 3-bit of the next generated value. We may try to use other pseudo-random generators if we desire one with better statistical properties, but the LFSR is sufficient as a roll-call system for this project.