HO CHI MINH UNIVERSITY OF TECHNOLOGY

ELECTRICAL ELECTRONIC ENGINEERING

Electronic Department

----------o0o----------

LOGIC DESIGN / LOGIC SYNTHESIS

LAB 1

INSTRUCTORS: Tran Hoang Linh

Nguyen Tuan Hung

STUDENT : Ha Gia Huy _ 2051117

HO CHI MINH CITY, FEBRUARY, 2024

Table of Content

FIGURE

1. Overview

The purpose of this project is to practice that has accumulated on my SystemVerilog skills. More specifically, I am provided some modules and my task is to write a downstream module that captures the calculated sum and holds it for display while the next sum is being calculated. All it does is wait for the done signal and then loads the calculated sum into its own register. The captured value is then displayed on the two hex displays. Moreover, I must write the p1 module to be the top module of design, connecting everything and synthesize on Quartus. The topology should be similar to the picture below.

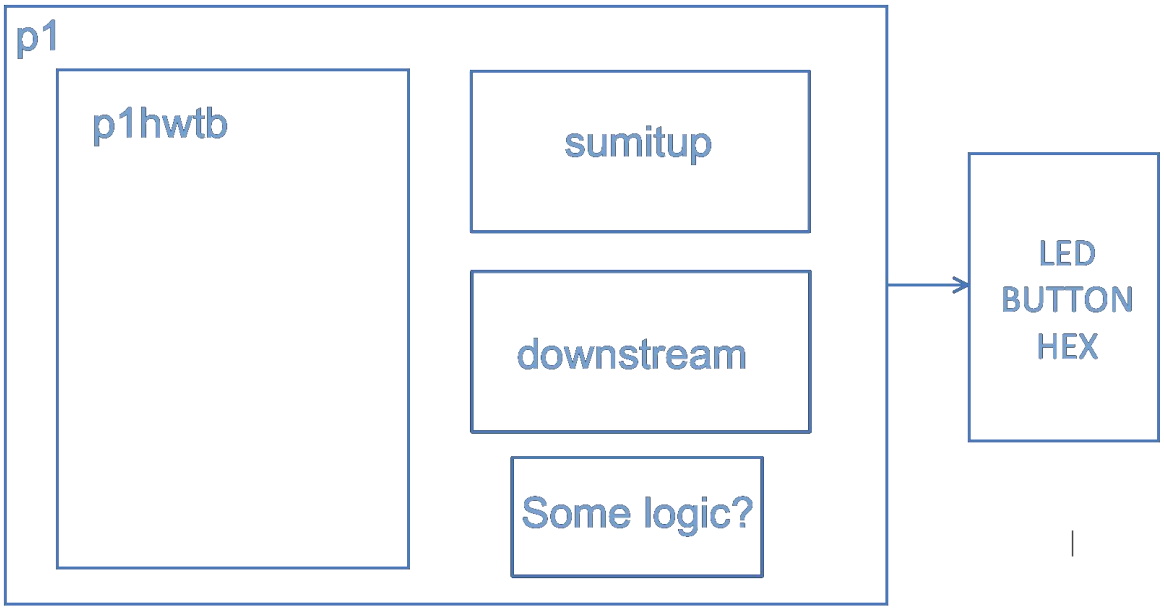


Figure 1. Topology

2. Design Strategy

The primary purpose of the sumitup module is to accumulate a series of input numbers. It's a straightforward task where the go_1 signal signifies the start of input values. With each clock cycle, a new value is presented at the input. However, the appearance of a zero on the inA input indicates the end of the series, triggering an immediate assertion of the done signal. The sumitup module, fully specified in the sumitup.sv file provided, requires no alterations. Clearly, there's more to this project beyond this module.

A downstream module responsible for capturing the computed sum and retaining it for display while the subsequent sum is being calculated. Upon detection of the done signal, this module loads the calculated sum into its register. Subsequently, the captured value is showcased on the two hex displays.

The provided hardware testbench generates a series of random values and transmits them to your sumitup module, presuming correct wiring. Upon completion, it exhibits what it interprets as the 8-bit sum of the input series in HEX3 and HEX2 (the leftmost digits). Calculated sum should be showcased in HEX1 and HEX0. The objective is to have the same two 2-digit hexadecimal numbers displayed in the topmost (left) and bottommost (right) hex displays if interfacing and wiring have been executed accurately. Additionally, upon successful match, the testbench will emit a signal (connected to LEDR0) to illuminate the LED. Combinational logic is essential for driving the displays, hinting at the need for a bcdtohex module to facilitate this task.

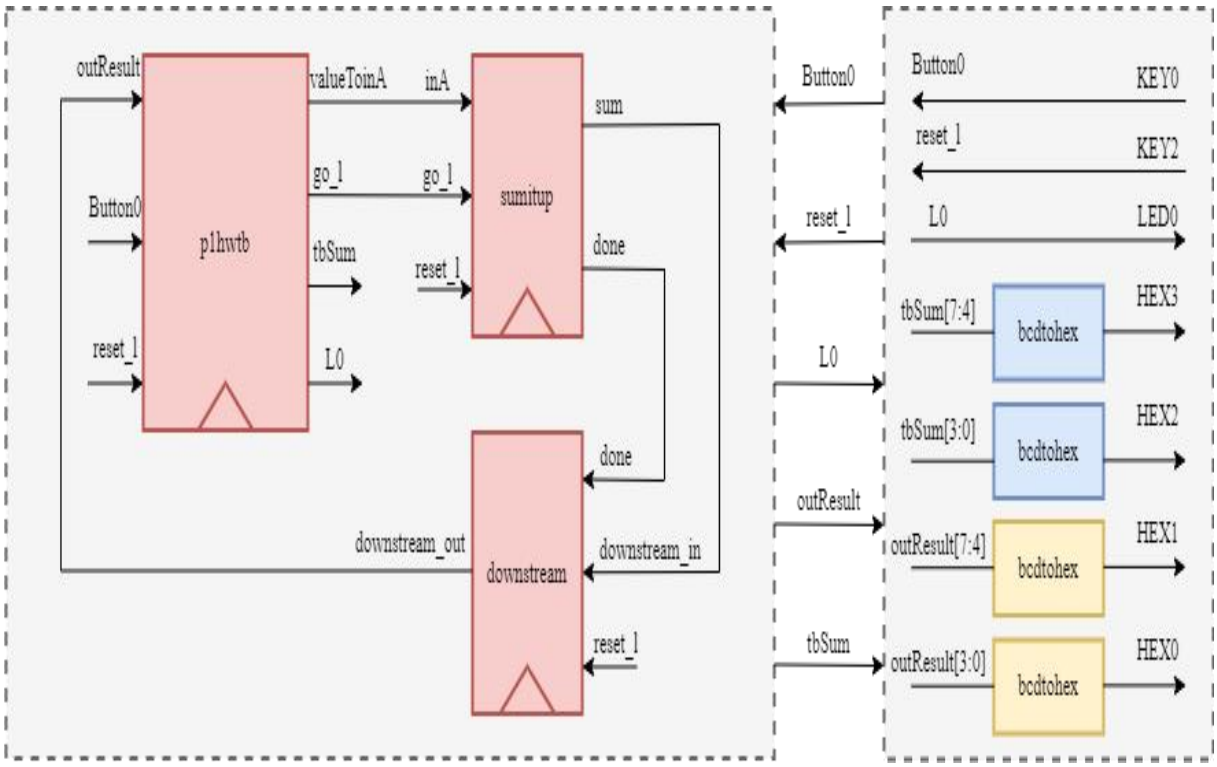


Figure 2. Block Diagram

3. Testbench Operation

Two buttons on the board serve to control the testbench's operation. KEY2 acts as a reset, causing the display to show zeros, turning off LEDR0, and resetting the Finite State Machine (FSM). Upon board reset, all displays should indicate zeros. Holding down KEY0 initiates the operation, prompting the hardware testbench to transmit a series of numbers to your sumitup thread at a rapid 50 MHz pace. The upper hex displays (HEX3 and HEX2) showcase the sum calculated by the testbench, while the lower hex displays (HEX1 and HEX0) exhibit the result of my code's computation (the value captured by the downstream module). If these two values match, LEDR0 illuminates. Releasing KEY0 resets the testbench's hex displays to zero, while my code continues to display the calculated sum in the lower digits. The system awaits the subsequent press of KEY0 to initiate a new series of number transmission and display. Activating KEY2 performs a reset, ensuring all displays show zeros. Notably, the buttons operate on an active-low basis, meaning they produce a logic 0 when pressed.

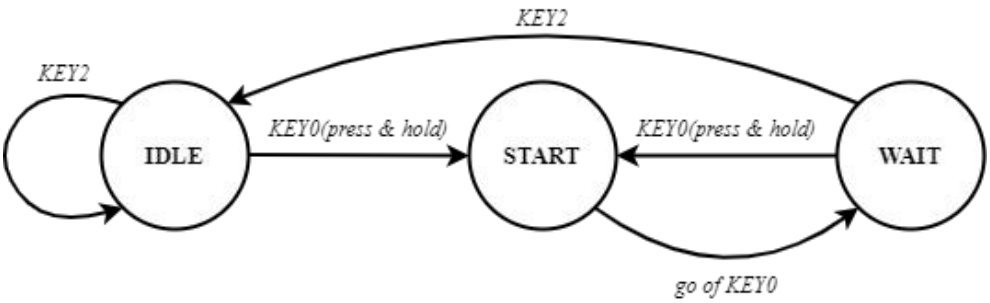
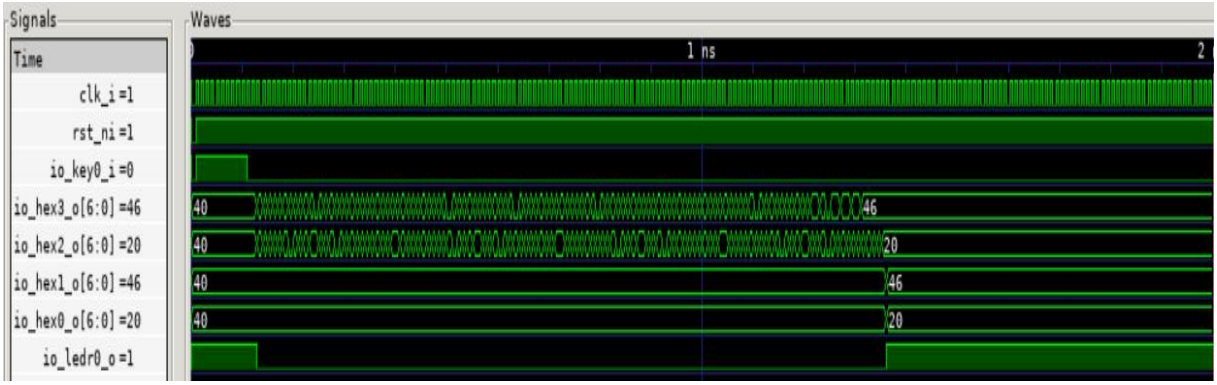


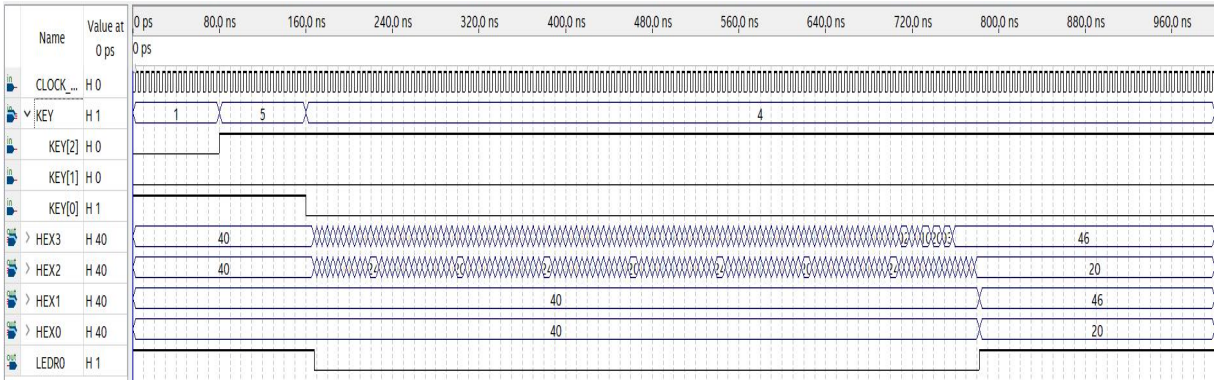
Figure 3. Finite State Machine

4. Simulation

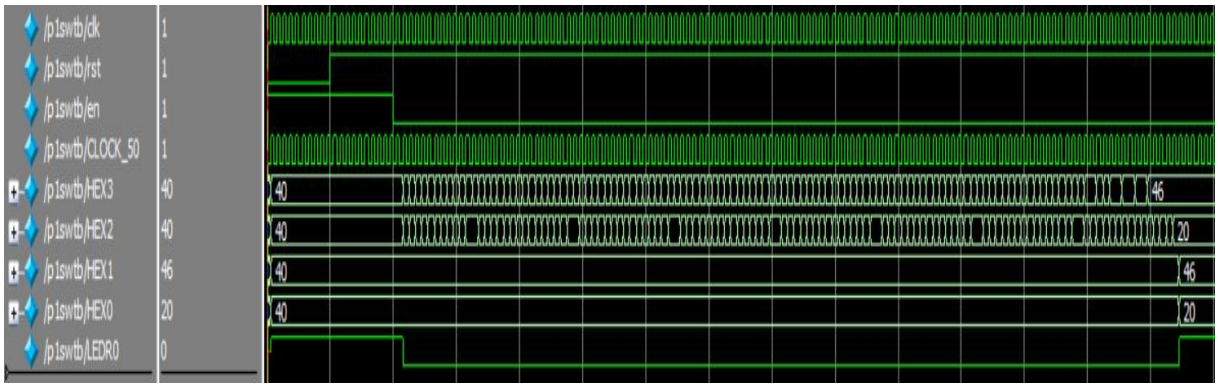
❖ VERILATOR simulator (*using tb_top.cpp in Appendix*):



❖ QUARTUS simulator (*config signals by hand*):



❖ MODELSIM simulator (*using p1swtb.sv*):




```
# Start testbench
# LED0 = x
# LED0 = 1
# LED0 = 0
# LED0 = 1
# Testbench finished, Check the output on Waveform
# If it is good, you should see at least two times that LED0 = 1
# 1 is when the testbench begins, 2 is when the testbench ends
# LED0 = 0
# ** Note: $finish    : D:/Lab/Lab1/qts/../p1swtb.sv(44)
#    Time: 11200 ps  Iteration: 0  Instance: /p1swtb
```

=> The simulation results of the three simulators are the same. Leading to the conclusion that my design worked correctly in simulate environments. Because LEDR0 active two times. Once when initializing and once after the testbench ends.

5. Result

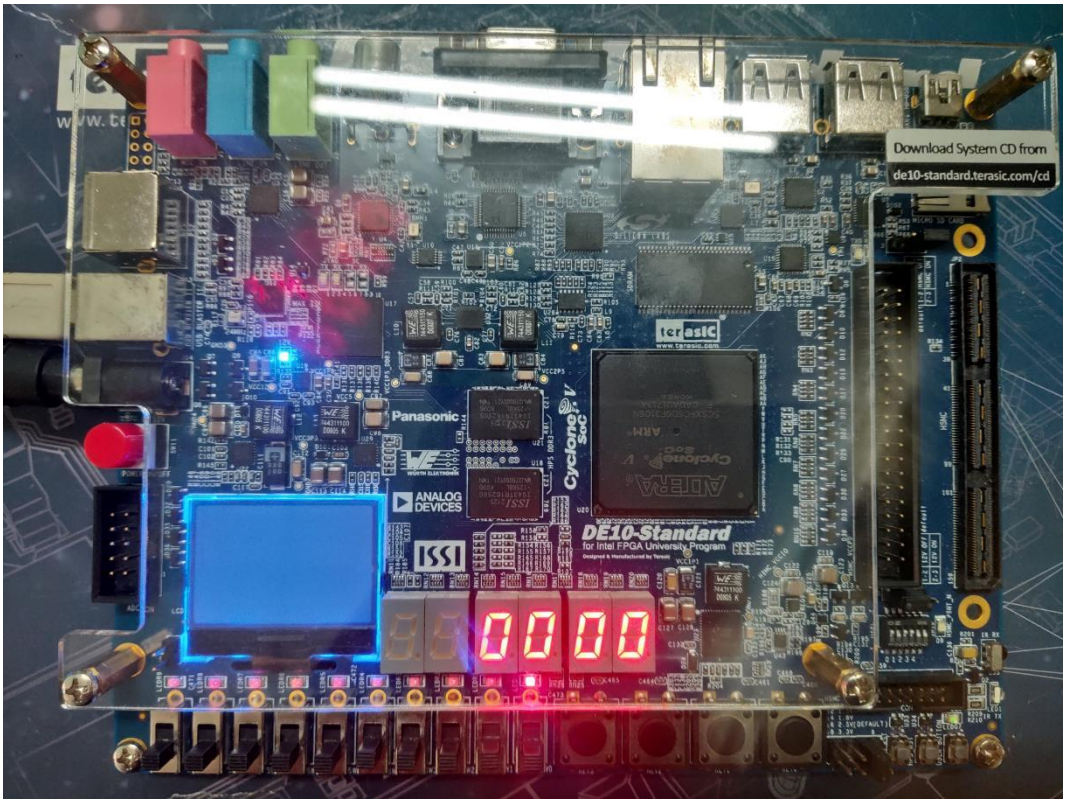


Figure 4. Initial Operation

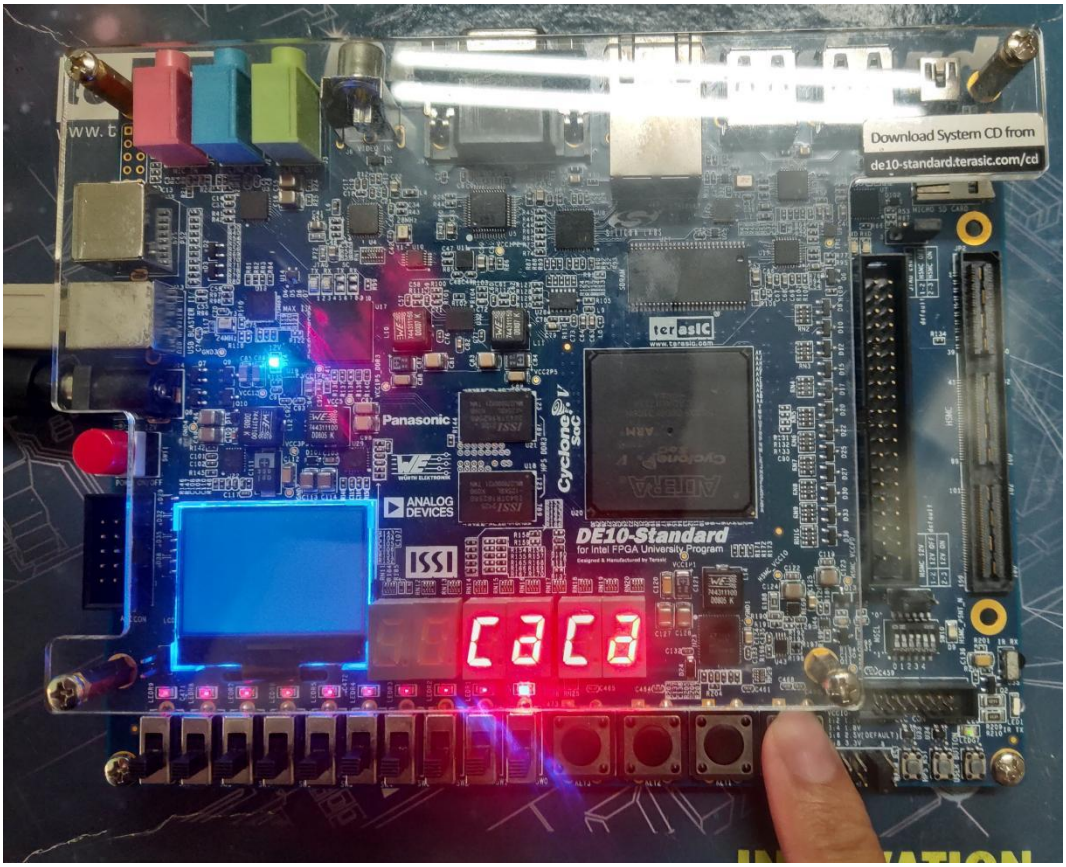


Figure 5. Start Operation

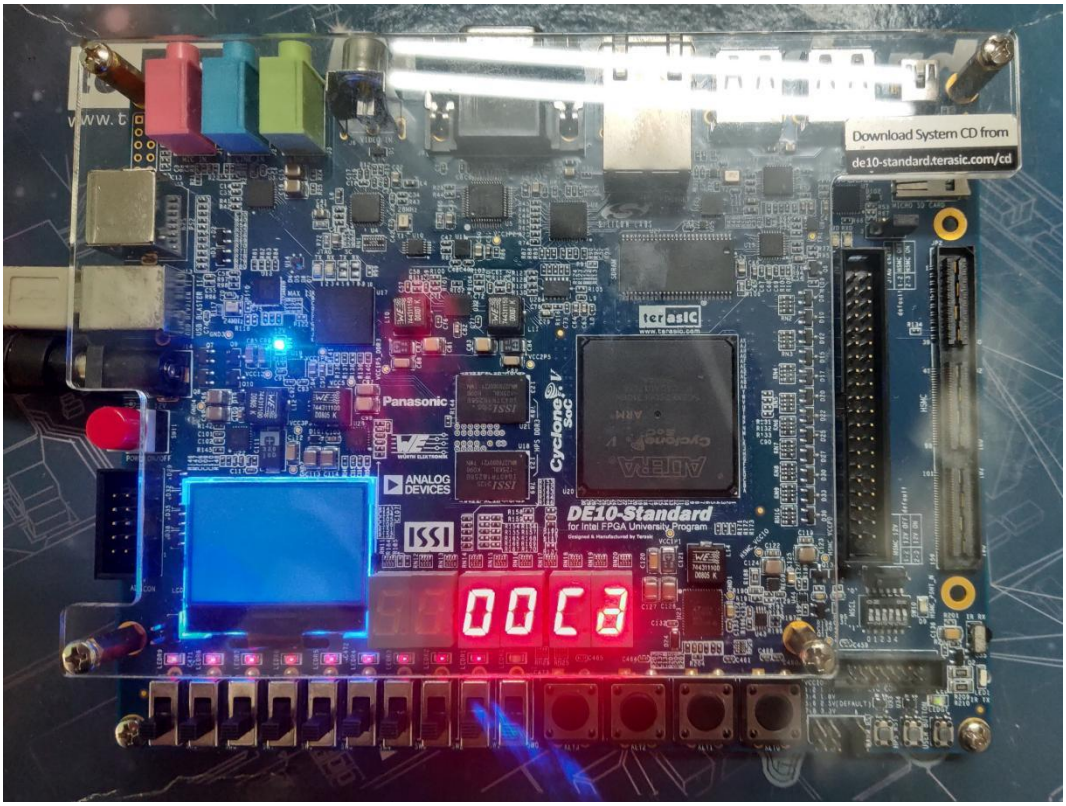


Figure 6. Wait Operation

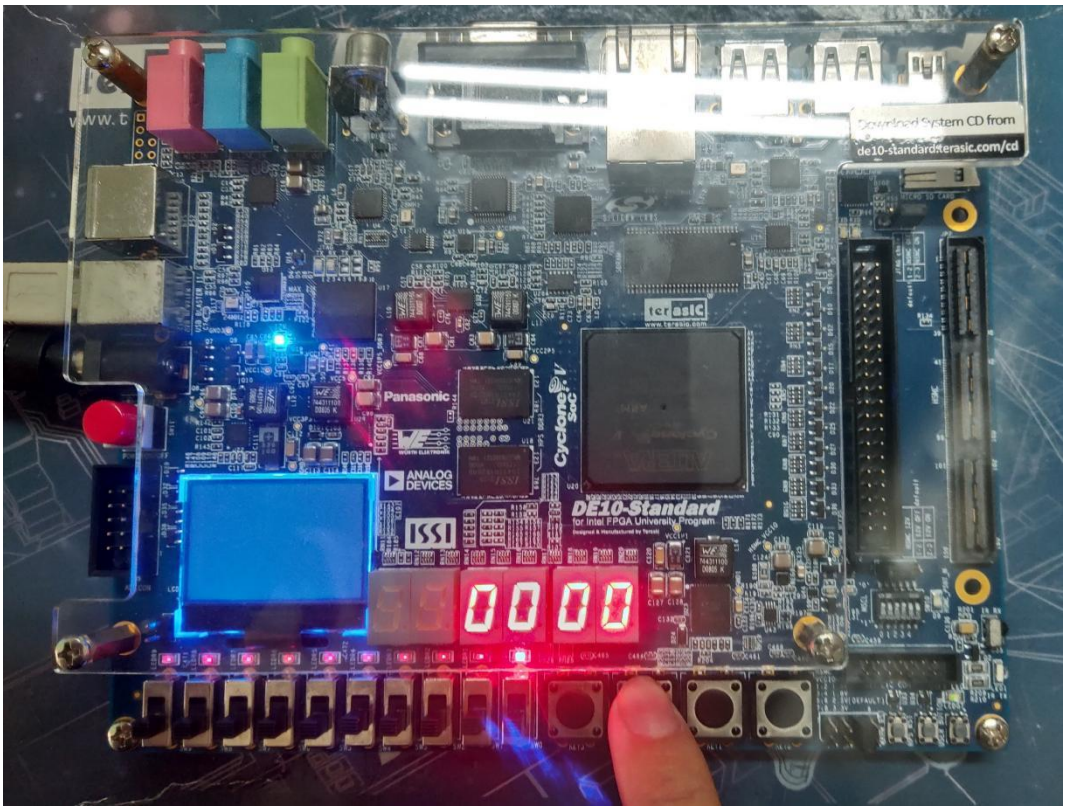


Figure 7. Reset Operation

6. Conclusion

In conclusion, this project concentrate on connecting modules and a little bit coding skill. After designing, I use VERILATOR to simulate my design with *tb_top.cpp* (in the Appendix below). Additionally, Quartus simulates all signals manually during configuration. Furthermore, using *p1swtb.sv* (provided by Teacher) to checking again on Modelsim. After using three simulators with 3 three configurations to achieve the same result (at least twice LEDR0 = 1: one is when the testbench starts and one is when the testbench ends). Consequently, my design works correctly.