Experiment 3 - Function Generator

Shahriar Attar,

*810100186*,

attarshahriar@gmail.com

Elahe K. Nadrabadi,

*810100132*,

elahekhodavrdi@gmail.com

***Abstract*— This document is a report for experiment #3 of Digital Logic Design Laboratory at ECE department, University of Tehran. The purpose of this experiment is to design an AFG (Arbitrary Function Generator) to generate different waveforms with different frequencies and amplitudes.**

***Keywords***— **Function Generator, Waveform generator, DAC, PWM, Frequency Selector, Amplitude Selector, FPGA Implementation, Altera Cyclone II**

1. Introduction

This experiment is to design an AFG to generate waves that may not be available in real-world signals or we want to replicate them. The generated signals can be used to verify electronic theories, test devices and so forth. We will generate all those waveforms and select which one to show based on the function selector and then we can manipulate it using frequency selector and amplitude selector and then convert digital to analog using PWM, finally we will check our design when we implement it using Quartus.

1. Waveform generator

// ITS DONE TILL HERE DONT TOUCH IT TILL

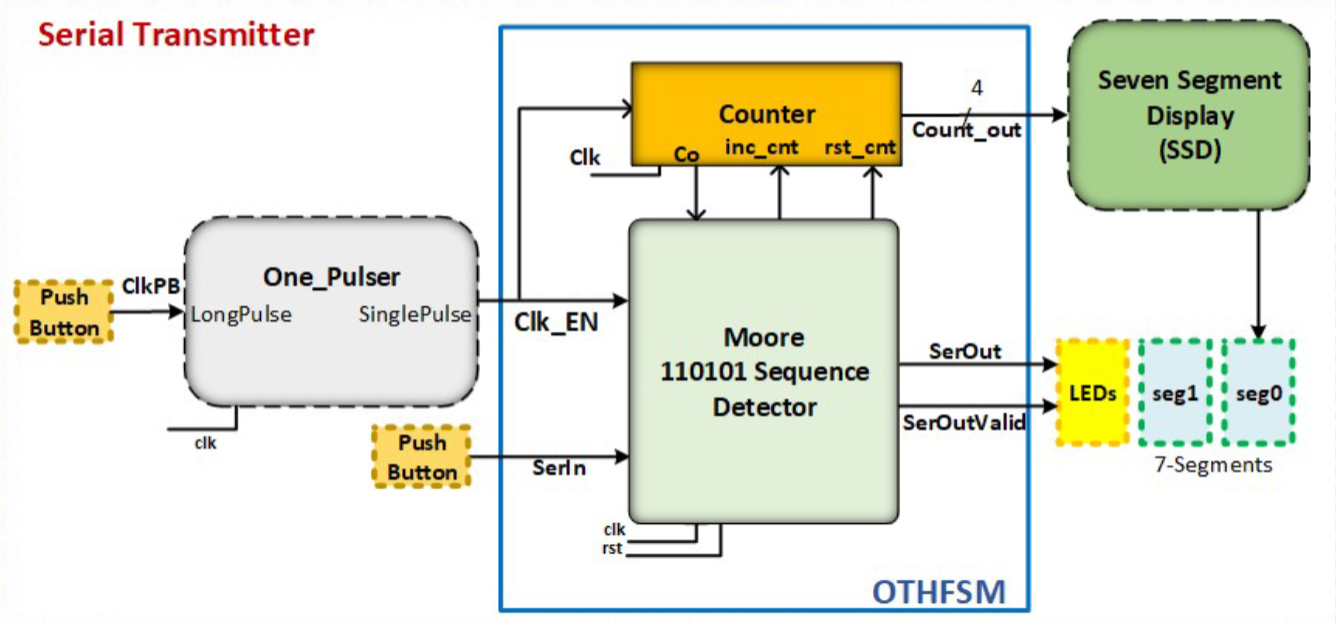


Fig. 1 overall design of serial transmitter

1. *One-Pulser*

This module provides a *Clk\_EN* input for the counter and the sequence detector part. This common input *Clk\_EN* is used for controlling the clock when the circuit is implemented on an FPGA board. The one-pulser connects to a push-button on your board (*ClkPB*) and when pressed it creates a single pulse that is synchronized with the system clock. Its state diagram is shown in Fig. 2. The main idea behind this module is that since humans are not as fast to change input in every clock toggle (since they usually have a duration of about few nanoseconds) we need a module to determine when we want to enable the clock, and since we don’t want to interfere with timing, it has to be synchronized with the system clock. In Fig. 2 the *lp* represents the *ClkPB* input in overall design, and the sp is the *Clk\_EN* output which we will assert once.

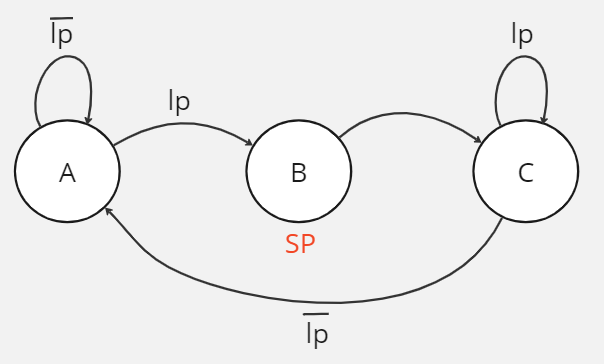


Fig. 2 finite state machine diagram of one-pulser

1. *Verilog Description:* : First we implement the module in Verilog. You can see the code in Fig. 3.

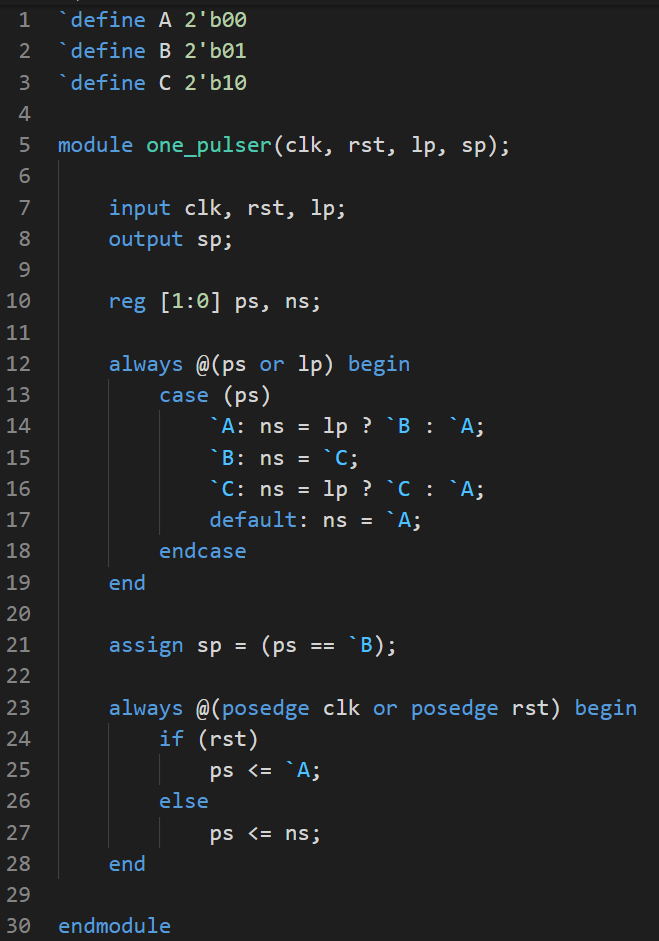


Fig. 3 verilog implementation of one-pulser

1. *Testing the design*: Then we wrote a test bench to check our module. testbench code is shown in Fig. 4 and ModelSim output is shown in Fig. 5.

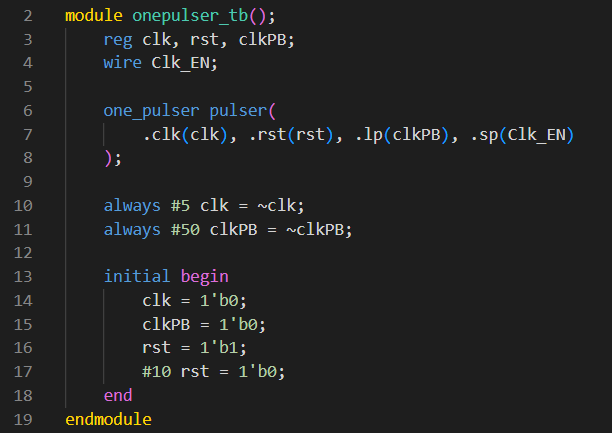


Fig. 4 testbench code for one-pulser module

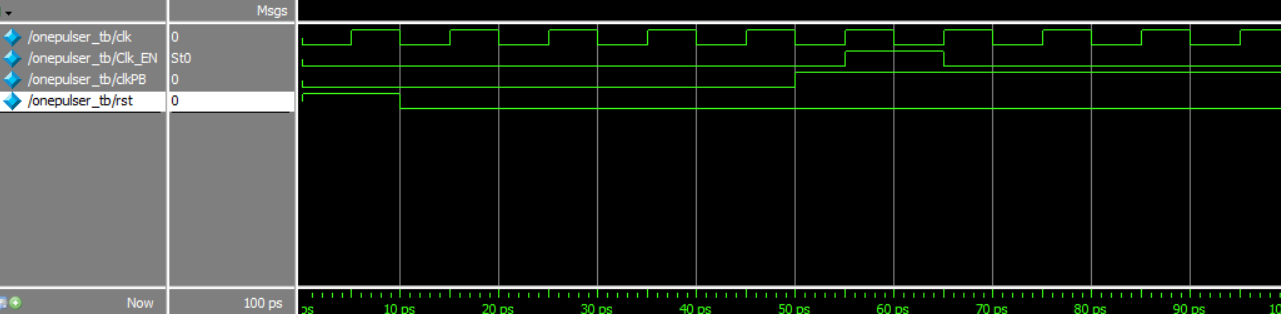


Fig. 5 ModelSim output for one-pulser testbench

1. *Orthogonal Finite State Machine*

It consists of a Moore state machine and a counter. The Moore state machine is a 110101 detector whose state diagram can be seen in Fig. 6, and the counter is 4-bit but the initial value for it is 6 so instead of 16 it counts 10 times and then the Co signal is issued. When *Clk\_EN* is pushed the Moore sequence detector checks its input and decides which state to go to. The sequence detector waits for the sequence of 110101 on its *serIn* input and when this is received, the *serOutValid* output becomes one and we transmit every input on *serIn* for the next 10 consecutive to the *serOut*, during which *serOutValid* should remain one. Note that when *serOutValid* is zero we put z on *serOut* to avoid any misunderstanding.

1. *State diagram*: You can see the state diagram of 110101 sequence Moore machine. All transitions are based on *serIn* other than the ones explicitly written.

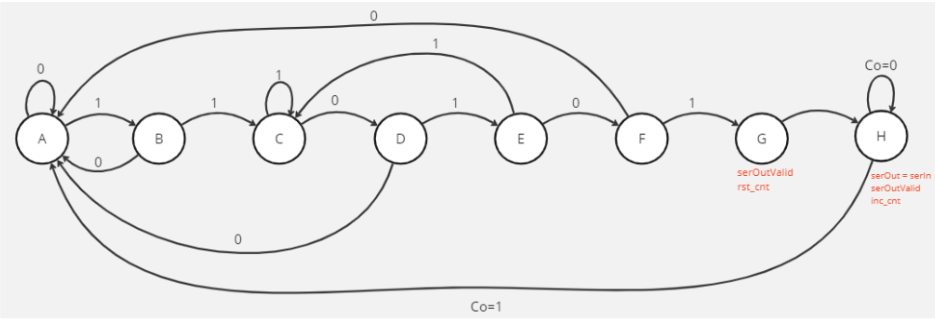


Fig .6 state diagram of 110101 sequence Moore machine

1. *Write top-level module for OTHFSM*: It consists of two main parts that are detector and counter, that each Verilog description is shown in Fig. 7 and Fig. 8 respectively. The top-level module implementation is also available on Fig. 9.

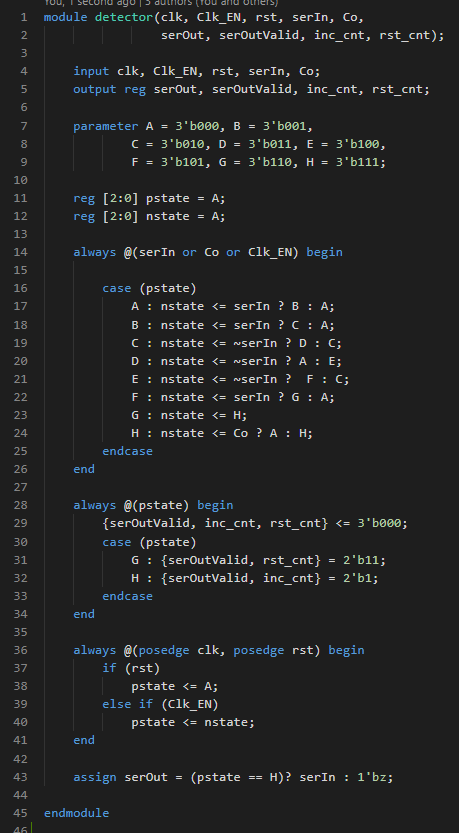


Fig.7 sequence detector implementation in Verilog

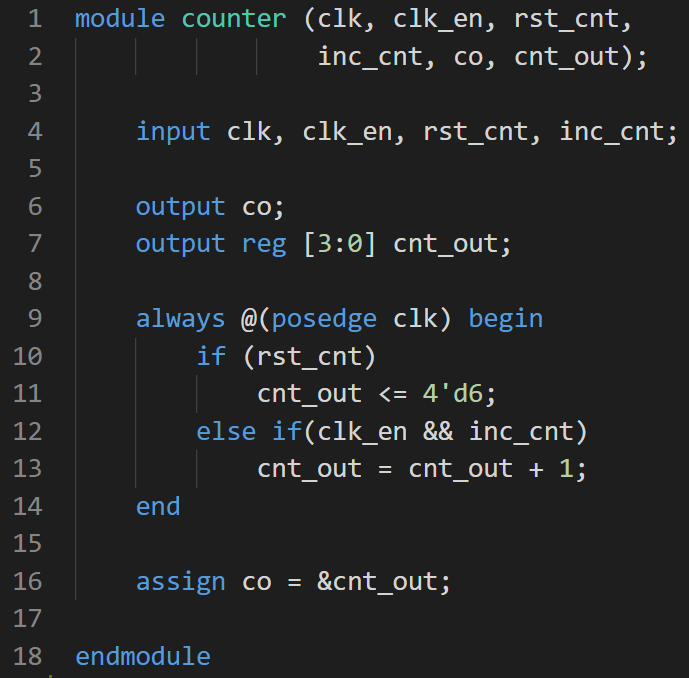


Fig. 8 counter implementation in Verilog

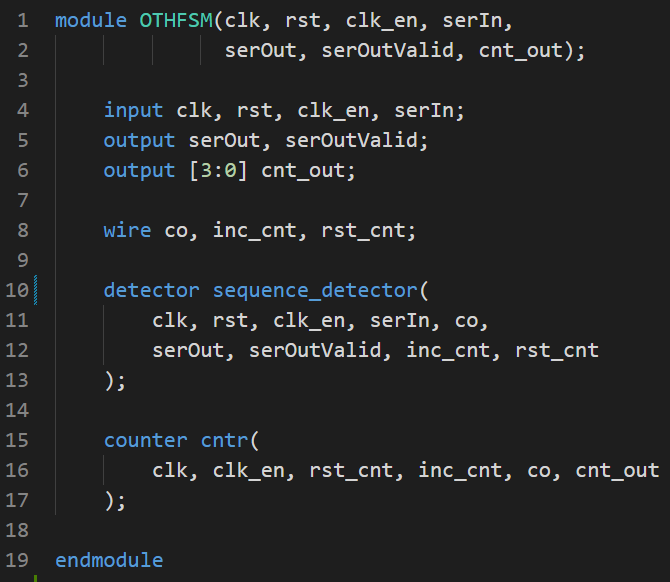


Fig. 9 top-level module in Verilog

1. *Testing the design:* The designed testbench as depicted in Fig. 11 will show the correctness of the design. ModelSim output is demonstrated in Fig. 10.

As you can see when we have detected the sequence for 10 clock cycles *serOutValid* has become one and we transmit data from *serIn* to *serOut.*

After that the *serOutValid* has gone back to zero and the value on *serOut* has been changed to the high-impedance (i.e. z value).

During each clock, counter will increase one time till we reach 15 in which our *Co* will become one as it is the result of logical-and of all bits of *cnt\_out*.

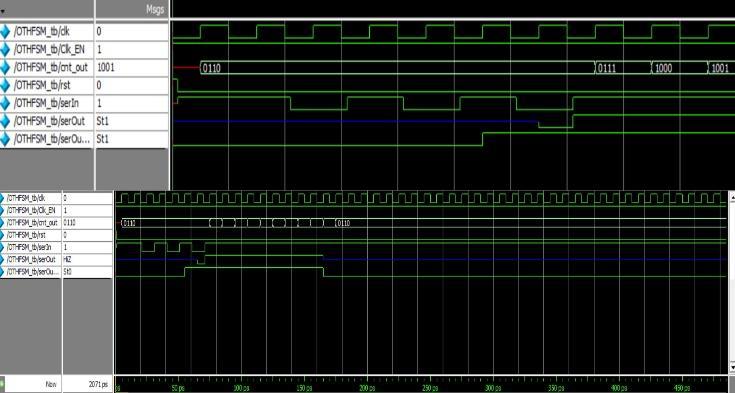


Fig. 10 ModelSim output for OTHFSM testbench

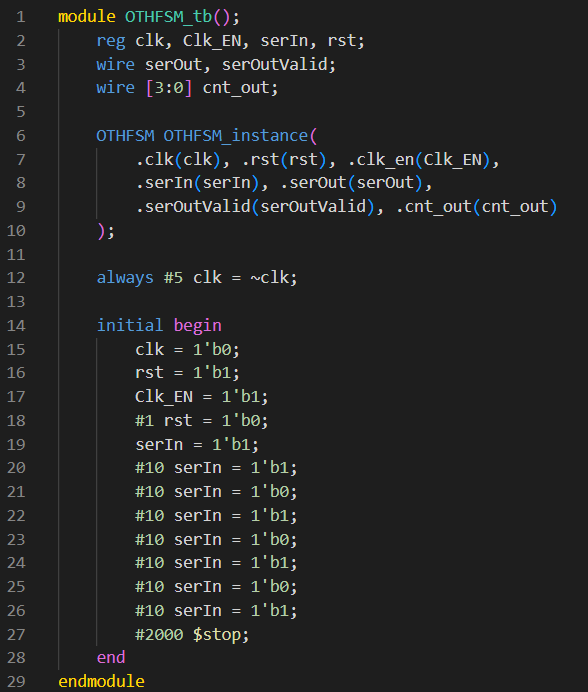


Fig. 11 OTHFSM testbench written in Verilog

1. *Seven Segment Display*

It’s a simple module used for displaying the counter output on the seven segments of your FPGA board. Each seven segment receives a 4-bit input and displays the HEX value on its 7-bit output. Verilog code is attached in Fig. 12.

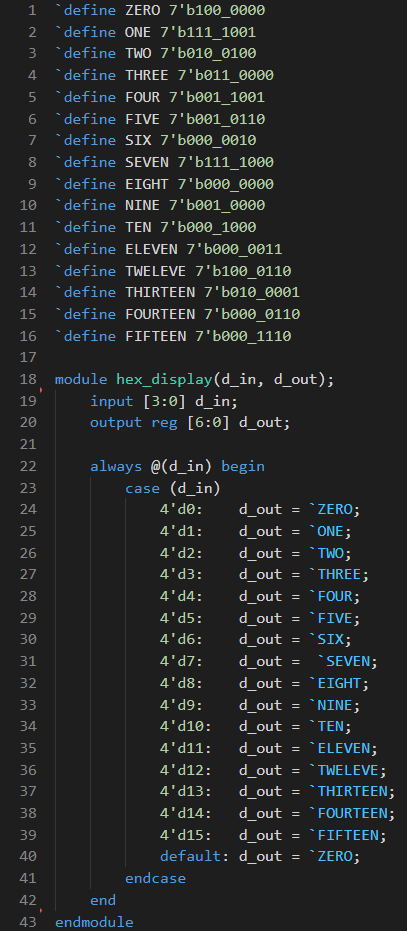


Fig. 12 seven segment display code in Verilog

1. Design Synthesis and FPGA Programming
2. *Serial Transmitter Implementation:* In this part we will use DE1 development board for our design implementation. The top-level Verilog module of the whole serial transmitter is shown in Fig. 13. Then you can follow the step by step guide to synthesise the design.

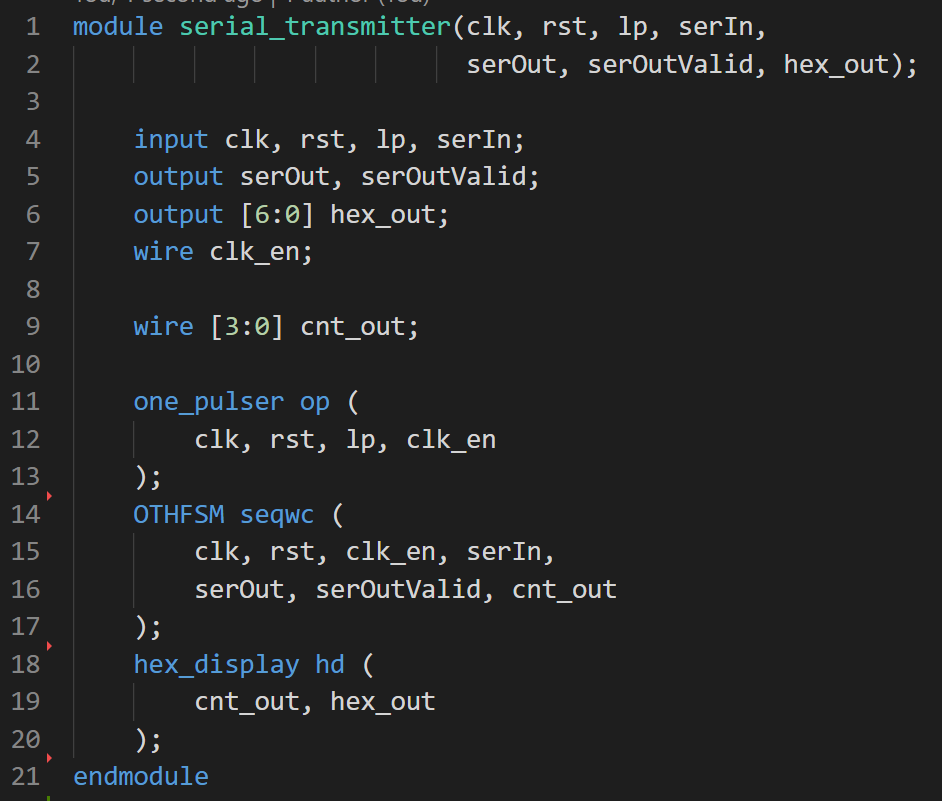


Fig. 13 top-level module implementation in Verilog language

1. *Make a Quartus Project.*
2. *Add the top-level Verilog codes to your project.*
3. *Connect the main FPGA clock to the clock input of your circuit:* for this you can check the DE1 manual and choose your own desired clock but we chose 50MHz that according to the manual is PIN\_L1.
4. *Connect a push-button to the serIn of the serial transmitter circuit, another push-button to the input of the one-pulser circuit:* you can use any of the following list for inputs: [PIN\_L22, PIN\_L21, PIN\_M22, PIN\_V12, PIN\_W12, PIN\_U12, PIN\_U11, PIN\_M2, PIN\_M1, PIN\_L2]. But obviously the buttons will differ when you use different inputs. The given list is the order from right to left. The general design of DE1 development board is shown below in Fig. 14 and Fig. 15.

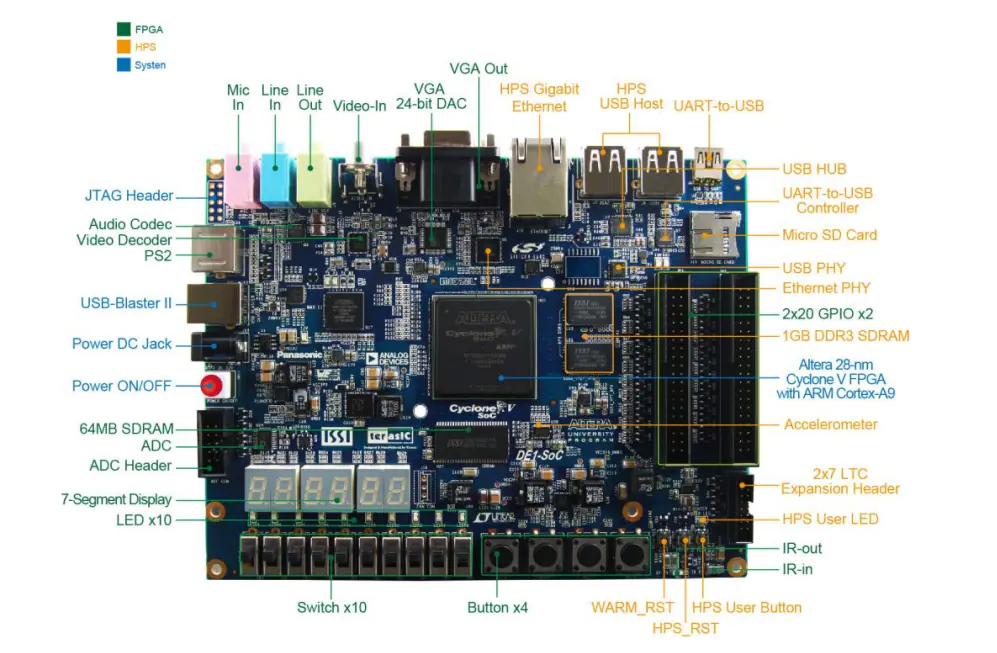


Fig. 14 general design of DE1 development board

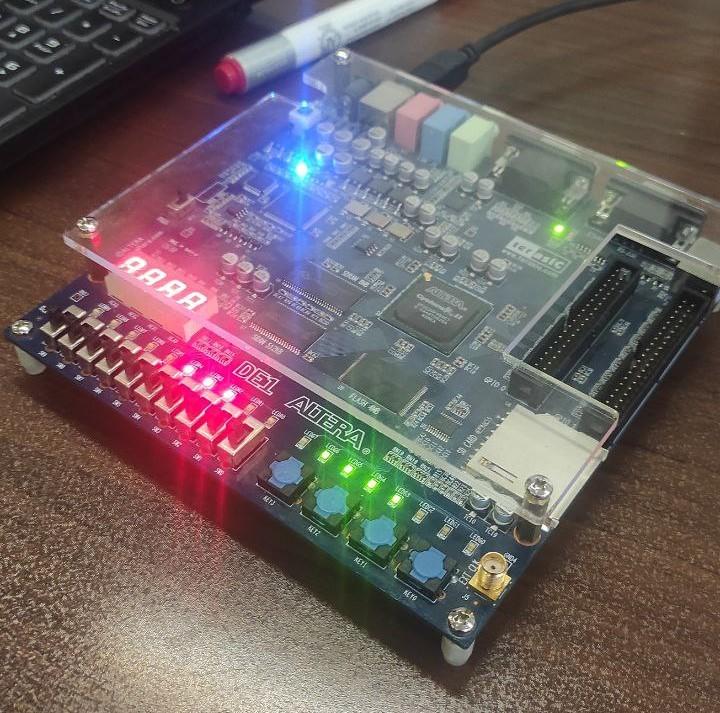


Fig. 15 general design of DE1 development board

1. *Perform the synthesis of your design.*
2. *Program the Cyclone II device.*
3. *For the output display, use an output LED for displaying the serOutValid and one LED for serOut. Use one seven segment for displaying the counter output:* You have two types of LED, namely RED and GREEN. And also you can arbitrarily choose one of the seven segments for displaying counter output in hexadecimal.
4. *Applying a serial input*: push the push-button corresponding to the serIn according to the value you want on the input, and then press the ClkPB push button once.
5. Conclusions

The general ideas discussed in this experiment were:

1. Finite State Machines and Orthogonal FSMs
2. One-Pulser
3. Seven Segment Display
4. Design Synthesis and FPGA Programming

Acknowledgment

This report was prepared by Elahe K. Nadrabadi and Shahriar Attar for 3,4 sessions of DLD Laboratory at ECE department in 1401-1402 spring semester.

References

# Introduction to the finite state machines (FSM) [Online]. Available: <https://www.inf.ed.ac.uk/teaching/courses/inf1/cl/notes/Comp1.pdf>

1. Altera DE1 board [Online]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=83