9/30/2024

**Lab 03 Report**

Ryan Beck

Jared Bronson

**Objectives**

Implement a pseudo-random number generator on the DE-10 Lite development board. The random number generator should be implemented with an appropriately sized LFSR. Taps in LSFR should be chosen to maximize the number sequence. The two on-board buttons will be used to generate and reset. When generate is pressed, a two-digit hexadecimal number, from 00-FF, will appear on two of the seven-segment displays. Pressing reset will revert the sequence back to the seed value.

**Procedures**

Using the system builder, create a Quartus Project File to include the on-board clock, seven-segment displays, and push buttons. Open the Verilog file created by the system builder and create a VHDL file with the same name. In the VHDL file, create a random number generator, rng, entity with ports for the on-board clock, seven-segment displays, and push buttons using the same names given in the Verilog file. Only use the 10 MHz clock for this design. In the generic, create a 16 bit logic vector for a seed value, and create a 16 bit hexadecimal value to populate the vector. For this design, the hexadecimal value A58B was used. Next, create architecture for the design. Initialize two 16-bit vector signals to perform operations on the LFSR. Also, initialize any other needed signals for the design. Create a lookup table with the needed values to display all 16 hexadecimal values on the seven-segment display. Begin a process sensitive to the clock and push buttons. In this process create if statements to for KEY(0) press, KEY(1) press, and a start state before either button is pushed. Make an else statement for an idle state to continuously update the LFSR to assist randomness. In the KEY(0) press statement, reset the LFSR to the seed value and display it on the display. In the KEY(1) press statement, display the LFSR that is continuously being updated. In the start state, display the original seed value. Create another process sensitive to clock, and turn off the unused seven-segment displays.

Next, create a testbench with the same generic and ports as the rng. Define signals for the clock, buttons, and seven-segment displays as well as a constant for clock period. Start a process to vary the clock between one and zero every half clock period. Start another process to test the behavior of the design for the button pushes. Simulate the design using the testbench. If the simulated behavior matches what is expected, compile and implement the design on the development board.

**Results**

The rng file, shown in Figures, successfully implements a random number generator. The design successfully resets to an original seed value by pushing reset, and generates random two digit hexadecimal numbers. The function of the file was verified with a testbench file used to run simulations. Both the testbench and the simulations can be found in Figures.

**Figures**

**A screenshot of a computer

Description automatically generated**

Figure 1: Counting sequence, incremented on 'tick' variable

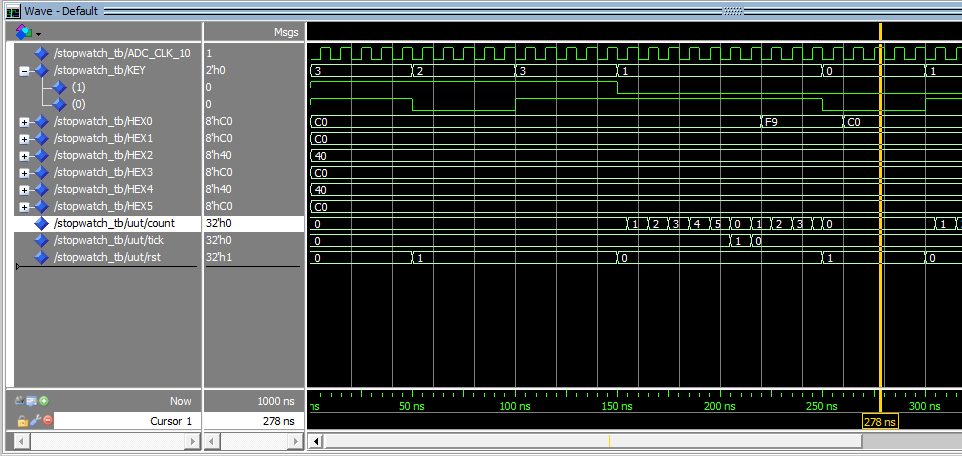


Figure 2: Reset behavior

A screenshot of a computer

Description automatically generated

Figure 3: Timer incrementing effect

A screenshot of a computer program

Description automatically generated

Figure 4: RNG.vhd file Pt. 1

A computer screen shot of a program code

Description automatically generated

Figure 5: RNG.vhd Pt. 2

A screenshot of a computer program

Description automatically generated

Figure 6: RNG\_TB.vhd Pt. 1

A screenshot of a computer

Description automatically generated

Figure 7: RNG\_TB.vhd file Pt. 2

**Conclusion**

In conclusion, we were able to implement a pseudo-random number generator on the development board. One push button resets the LFSR to the seed value, and the other button displays a random hexadecimal number, 00-FF. We did this by constantly updating the LFSR each clock cycle to improve randomness and displaying it only when generate is pushed. We configured the seven-segment display using a lookup table. Values in the lookup table were used to turn off segments corresponding the appropriate hexadecimal value. Simulations were performed using a testbench to show proper function of the design before final implementation and testing on the development board.