Gareth Peters (260678626)
Peter Quinn (260689207)

October 27, 2017
ECSE 323

Lab 2: RANDU, POP_EN, 7 Seg Display

g07_RANDU

Description

This circuit is intended to generate pseudo random 32-bit numbers. The circuit employs IBMs RANDU function algorithm [1]. This algorithm was chosen because it is easy to implement. A disadvantage of it is that the numbers are not extremely random, however this fact is used to test the circuit later. The randomness of the RANDU algorithm is sufficient for our needs.

The RANDU algorithm takes a 32-bit input value called seed, and applies the function R=mod(a*seed+b,c) with the values a=65539, b=0, and c=2^31. This has the effect of multiplying the input seed value by 65539, then taking the modulo 2^31 of the product. The result of the modulo is the random number. Subsequent random numbers can then be generated by feeding the output back into the input.

This algorithm is implemented in the following way.

1. To multiply the input by 65539 a shift and add approach is used, as 65539 can be written as $2^{16}+2^{1}+2^{0}$
    a. seed<<16 (seed shifted left 16 bits) and seed<<1 are added using a 32-bit adder
    b. seed is then added to this sum
2. To take the modulo 2^31, a bit wise approach is used because 2^31 is a power of 2
    a. the MSB (the 32nd bit) is replaced with 0 which removes all multiples of 2^31

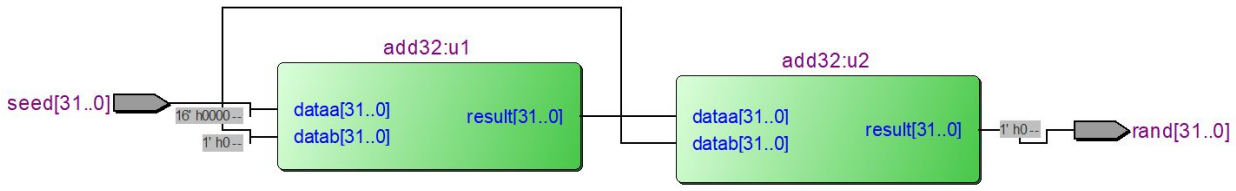

Fig.1: Flowchart describing logic used in RANDU circuit

The 32-bit adders in this circuit were implemented using the LPM library included in Quartus [2].

This circuit was built using the VHDL code editor in Quartus.

Schematic

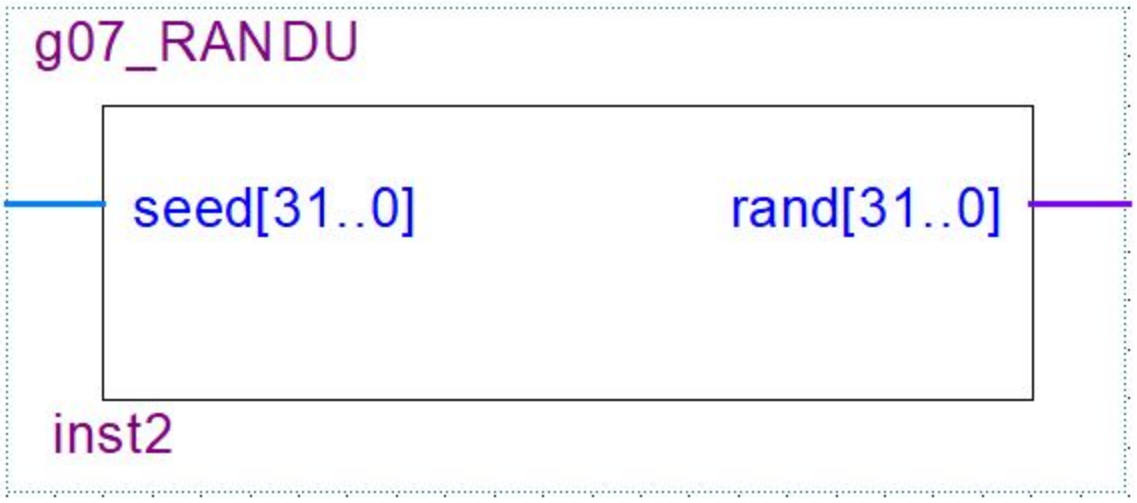

Fig.2: Symbol of g07_RANDU

The gate level diagram of this circuit is not available since it was created using VHDL code.

Testing and Simulation

The output of the circuit produces seemingly random values.
The following function was used to check if the implementation of RANDU was correct.

$$mod(R_n - 6\,R_{n-1} + 9\,R_{n-2}, 2^{31}) = 0$$

An initial seed value of 1 was used (R_{n-2}), which produced $R_{n-1} = 65539$. This value was fed back into the RANDU function and produced $R_n = 393225$. When entered into the above function, it is found that the function indeeds evaluates to 0.

This circuit was checked only using a functional simulation, which does not take into account timing issues that could arise from propagation and gate delays.

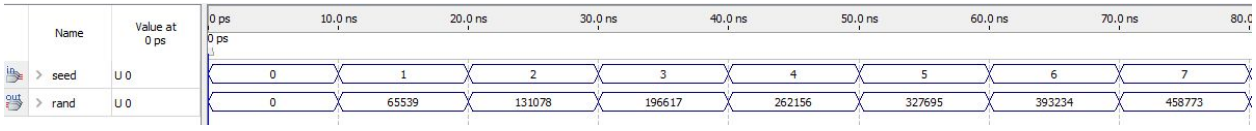

Fig. 3: Output of a waveform test file showing the random outputs for a range of seed values

g07_pop_enable

Description

This circuit is intended to take a 6-bit input N and will generate a 52-bit output, P_EN. This is intended to represent the location of a card in a hand of a card game.

The output bits are set such that for a given value of input N, P_EN[0] (the LSB of P-EN) to P_EN[N-1] are set to 0, and P_EN[N] to P_EN[51] are 1. For values of N greater than 51, the output bits are all 0. We include a clk input which takes the clock as input. This is to trigger the lpm_rom to switch values on rising edge.

This circuit was implemented using a ROM lookup table (LUT). The ROM that was used was lpm_rom which was created using the MegaWizard in Quartus [3]. A .mif (Memory Initialization File) file was used to specify the contents of the LUT. This has the advantage of making it easy to modify the contents of the ROM if needed.

The circuit was built using VHDL code.

Schematic

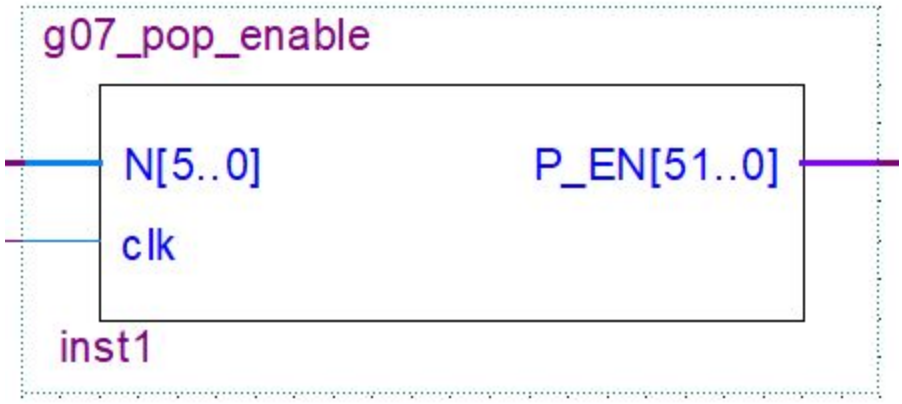

Fig. 3: Symbol of g07_pop_enable

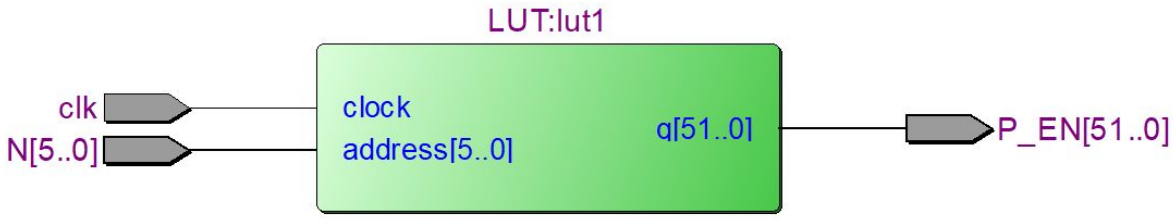

Fig. 4: Internal diagram the g07_pop_enable symbol. Note that it is very simple since it just implements lpm_rom

Testing and Simulation

Since the LPM ROM function requires a clock, for testing proposes, a clock signal was created simply as an input that oscillated between 0 and 1 at a frequency 5 times faster than the addresses were changed. This frequency was chosen because it is fast enough to ensure that the output for a given address has stabilized before changing to a new address.

It was observed that for a given address, the circuit selected the correct output from the LUT. The output achieved the correct value at the first rising edge of the clock signal following the input of an address to the circuit.

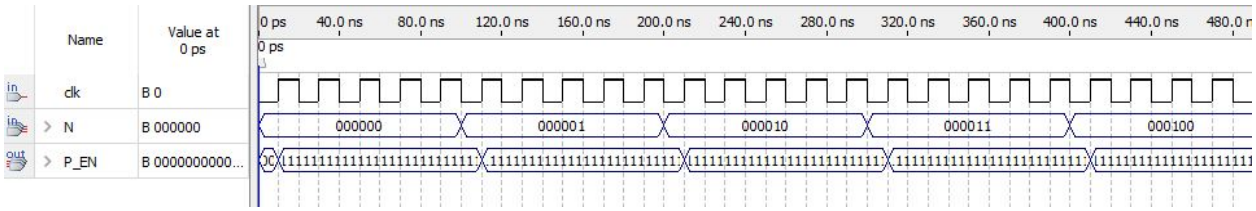

Fig. 5: Simulation output showing the response of the output to the clock signal and the changing address

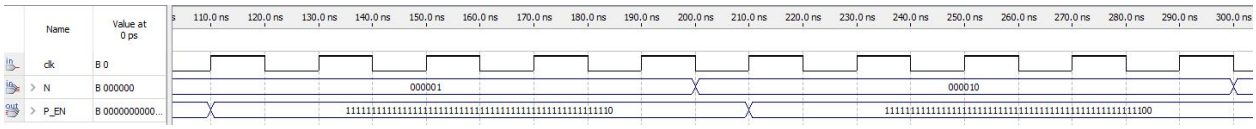

Fig. 6: Simulation showing correct output for sample input addresses

This circuit was checked only using a functional simulation, which does not take into account timing issues that could arise from signal propagation and gate delays, apart from the clocked ROM. Because the circuit depends on a clock signal, the output has a delay depending on the speed of the clock.

g07_7_segment_decoder

Description

The 7 segment decoder is meant to create 7 output bits for a 7 segment display given two inputs, “mode” and “code”. The 7 bits output bits are set based on active-low. Code is a 4 bit number, which is used to decide which segments to light up. Mode is 1 bit, which decides one of two methods to interpret the code input. The list of outputs for each of the possible inputs is shown in Fig. 8 [4].

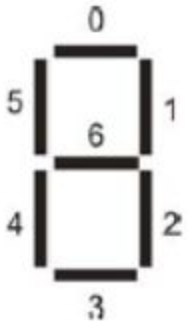

Fig. 7: The segments that the 7 output bits control (active-low)

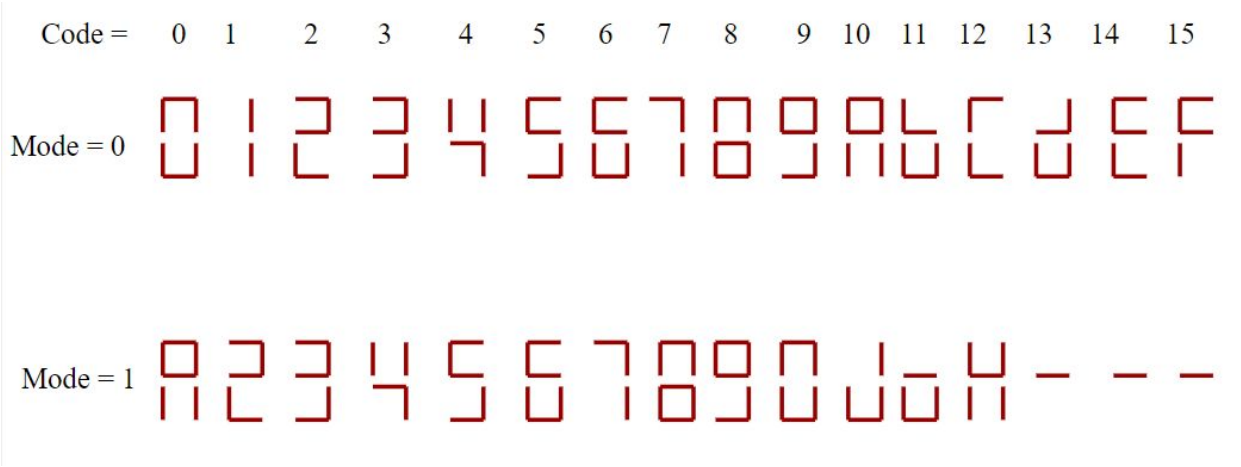

Fig. 8: Each segment display for each input.

Having mode = 0 gives a hexadecimal display and mode = 1 gives card symbols. Since there are only 13 possible card values, values 13,14 and 15 are given ‘-’ as a value. This circuit is more easily implemented in VHDL compared to block schematic since each case can be separated using ‘when’.

Schematic

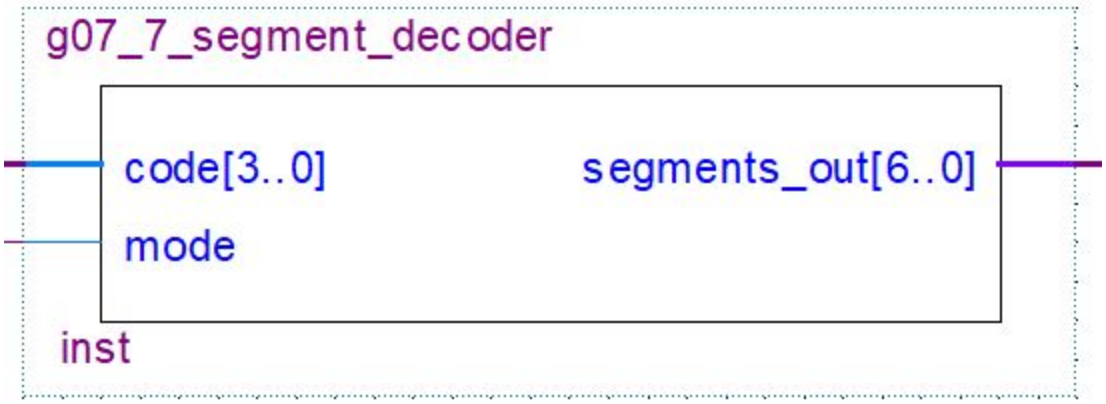

Fig. 9: Symbol of g07_7_segment_decoder

Testing and Simulation

It can be observed that every possible input has it's correct output given the LED design in Fig. 8 and the fact that the LED display has an active-low configuration. It can also be observed that changing the mode bit changes the way 'code' is deciphered as it should.

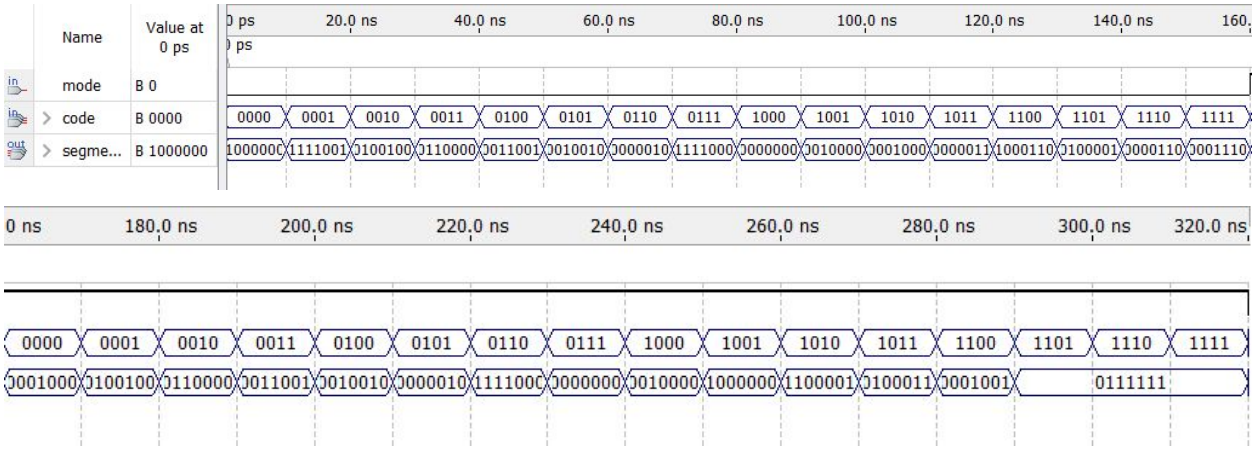

Fig. 10: Functional waveform of the 7 segment decoder circuit

This circuit was checked only using a functional simulation, which does not take into account timing issues that could arise from propagation and gate delays. There were difficulties in testing as every input had to be compared with the binary representation of the output. The Cyclone boards themselves would have been a good way to test the circuit because the boards have the LED display on them to see if the outputs were correct.

References

[1] System/360 Scientific Subroutine Package, Version III, Programmer's Manual. IBM, White Plains, New York, 1968, p. 77

[2] 2017. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/ug/ug_lpm_alt_mfug.pdf. [Accessed: 20- Oct- 2017].

[3] 2017. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_ram_rom.pdf. [Accessed: 20- Oct- 2017].

[4] CLARK, J.
Lab #3–Sequential Circuit Design
In-text: (Clark, 2017)
Your Bibliography: Clark, J. (2017). Lab #3–Sequential Circuit Design. Montreal: McGill.