Bittar, Catherine (260687735) 10 November 2017

Sorto-Ventura, Kevin-Rafael (260692767) McGill University

ECSE 323: Digital System Design

Lab #3: g21\_stack52, g21\_test\_bed

# 

Table of Contents

[Introduction 2](#_Toc498028767)

[g21\_stack52 2](#_Toc498028768)

[Circuit Function 2](#_Toc498028769)

[Circuit Design 3](#_Toc498028770)

[Registers 3](#_Toc498028771)

[Counter 4](#_Toc498028772)

[Mode Selection 4](#_Toc498028773)

[Schematic 6](#_Toc498028774)

[Simulation and Discussion 7](#_Toc498028775)

[g21\_test\_bed 9](#_Toc498028776)

[Circuit Function 9](#_Toc498028777)

[Circuit Design 10](#_Toc498028778)

[Single Pulse Generator 10](#_Toc498028779)

[Stack 11](#_Toc498028780)

[Output Assignments 11](#_Toc498028781)

[Schematic 11](#_Toc498028782)

[Simulation and Discussion 12](#_Toc498028783)

[Conclusion 13](#_Toc498028784)

[References 14](#_Toc498028785)

List of Figures

[Figure 1 g21\_stack52 functional block diagram 6](#_Toc498031678)

[Figure 2 g21\_stack52 opcode selection circuit schematic 7](#_Toc498031679)

[Figure 3 Shift register: BUSMUX and LPM\_FF pair 7](#_Toc498031680)

[Figure 4 INIT mode simulation results 8](#_Toc498031681)

[Figure 5 POP mode at ADDR=35 simulation results after INIT operation 8](#_Toc498031682)

[Figure 6 PUSH mode simulation results 8](#_Toc498031683)

[Figure 7 NOP mode simulation results after PUSH operation 9](#_Toc498031684)

[Figure 8 RST operation simulation results 9](#_Toc498031685)

[Figure 9 INIT mode with single clock cycle enable signal 9](#_Toc498031686)

[Figure 10 Single pulse generator circuit schematic 11](#_Toc498031687)

[Figure 11 g21\_test\_bed circuit schematic 12](#_Toc498031688)

[Figure 12 g21\_test\_bed simulation results 13](#_Toc498031689)

[Figure 13 Compilation report flow summary for g21\_test\_bed circuit 13](#_Toc498031690)

# 

# Introduction

The purpose of this lab was to design a 52 element stack, and implement it on the Altera DE1 board. The stack has four modes of operation: *init*, *pop*, *push*, and *no operation*. It also has an asynchronous reset function. A test bed was designed using a single pulse generator in order to implement the 52-element stack on the Altera board. The SignalTap II logic analyzer and timing simulations were used to validate the circuit. Note that this circuit will become an important building block for the term project which involves playing a card game.

The design was completed using the FPGA design software Quartus. It was also used to compute simulations of the designed circuit. This design was part of the project file *g21\_lab3.QPF*.

# g21\_stack52

## **Circuit Function**

The g21\_stack52 circuit has 6 inputs and 4 outputs.

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Name** | **Length (bit)** | **Purpose** |
| Input | DATA | 6 | Data to be pushed |
| Input | MODE | 2 | Control |
| Input | ADDR | 6 | Location to be viewed or popped |
| Input | ENABLE | 1 | Enable operations |
| Input | RST | 1 | Asynchronous reset |
| Input | CLK | 1 | Clock signal |
| Output | VALUE | 6 | Value of stack at location specified by ADDR |
| Output | EMPTY | 1 | Indicate empty stack |
| Output | FULL | 1 | Indicate full stack |
| Output | NUM | 6 | Number of stack elements |

The circuit has four modes of operation, and an asynchronous reset function.

|  |  |  |
| --- | --- | --- |
| **Operation** | **Control** | **Function** |
| NOP | MODE = 00 | No operation |
| INIT | MODE = 01 | Initializes registers to 0,1,2,...51 |
| POP | MODE = 10 | Returns element at location ADDR, removes it, and shifts register values at greater indexes up. |
| PUSH | MODE = 11 | Places DATA element in 0th register and shifts all registers down by one. |
| RESET | RST = 1 | Resets all values of registers to 0. |

## **Circuit Design**

The circuit is described in the schematic block diagram design file *g21\_stack52.bdf*. The design was implemented in four parts: the registers, the counter, the mode selection, and the reset function.

### Registers

The registers are composed of D flip-flops from the LPM library. The lpm\_ff components were selected as they have several functions which can be used to implement multiple operations [2]. The INIT operation can be completed with the *sset* function with sets registers to pre-assigned values [2]. Both the PUSH and POP operations are performed with the *data\_load* and *enable* inputs. The *enable* input is especially useful for the pop operation which only requires the shifting of a certain portion of the stack. Finally, the asynchronous *aclr* function allows the implementationof the reset function.

The capacity to shift values between registers is essential to the function of the stack. A BUSMUX multiplexer is used to link adjoining registers. A control signal is used to select whether values are to be shifted up or down. The output of the multiplexer is fed into the *data\_load* input of the lpm\_ff.

### Counter

The counter is used to keep track of the number of registers which are in use. The lpm\_counter component, from the LPM library, was selected as it offers the option of setting a predetermined value, with *sset*, as well as counting up or down, with *count\_en* and *updown* [2]. The EMPTY and FULL signals are determined by comparing the number of registers in use NUM with 0 and 52.

### Mode Selection

Each selected operation requires the setting of several signals to control the registers and the counter. For example, the POP operations requires a signal to push the data up, to enable only certain registers, to set the counter to count down, and to enable the counter. To simplify the assignment of such signals, four different opcodes were created, corresponding to each mode of operation, in the form of constants, with the lpm\_constant module [3]. The opcode is a 10 bit value defined below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Signal** | **Opcode index** | **Value** | | | |
| **NOP** | **INIT** | **PUSH** | **POP** |
| data\_select | 9 | 0 | 0 | 0 | 1 |
| sset | 8 | 0 | 1 | 0 | 0 |
| count\_en | 7 | 0 | 0 | 1 | 1 |
| updown | 6 | 0 | 0 | 1 | 0 |
| ADDR | 5..0 | 110100 (52) | 000000 | 000000 | 000000 |

The signals functions are defined below.

|  |  |
| --- | --- |
| **Signal** | **Function** |
|
| data\_select | Select data to be pushed up or down into the registers during the POP or PUSH operations. |
| sset | Set the registers and the counter to their predetermined values for the INIT operation. |
| count\_en | Enables counting for the POP or PUSH operations. |
| updown | Determines if the counter counts up or down. |
| ADDR | Used in conjunction with the g21\_pop\_enable circuit to determine the enable signal. |

The g21\_pop\_enable circuit was designed in lab 2 and is used to assign the enable signal for every operation. The use of the same components for each mode of operation facilitates timing considerations which are quite relevant as discussed further. For the modes which require all the registers be enabled, INIT and PUSH, the address passed to the circuit is 000000 which returns all 52 signals high. For the no operation mode, none of the registers are enabled by passing the address 110100 (52) to the pop enable circuit which returns all 52 signals low. Finally, the address specified by the ADDR input replaces the opcode in its place for the pop operation.

As mentioned above, the use of the g21\_pop\_enable circuit to produce the array of enable signals for every operation facilitates timing considerations. The g21\_pop\_enable circuit employs ROM and it clock enabled. This causes the enable circuit to have an additional delay of 1 clock cycle. This becomes increasingly relevant when the enable signal is a single clock-cycle in length. In particular, it was found that the INIT operation was not occurring despite it being the selected mode, because the *sset* input was activated before the *enable* input was active as it did not clock dependant. The one clock-cycle delay meant that the INIT operation could not take place. To solve this issue, extra flip-flops were implemented with the opcode elements which interacted with the registers. The flip-flops, being clock-enabled as well, solved this issue. Simulations showed that the enable signal was actually 2 clock-cycles long. Consequently, two flip-flops are used to delay the rest of the opcode signals.

Depending on the selected mode, several conditions need to be checked within the circuit before executed the operation. For example, a PUSH operation can only take place if the MODE control bits are set to 10, the stack if not full, and the enable signal is on. A series of multiplexers are used to check conditions for each operation to take place. When a condition is not met, the no operation opcode is passed through.

## **Schematic**

Since the schematic of the stack is very large and complex, it is not shown in its entirety in this report. Rather, a block diagram is shown, as well as the mode selection circuit and an instance of the BUSMUX and lpm\_ff pair which constitutes a shift register.

For a view of the full schematic diagram, please refer to design file *g21\_stack52.bdf*.

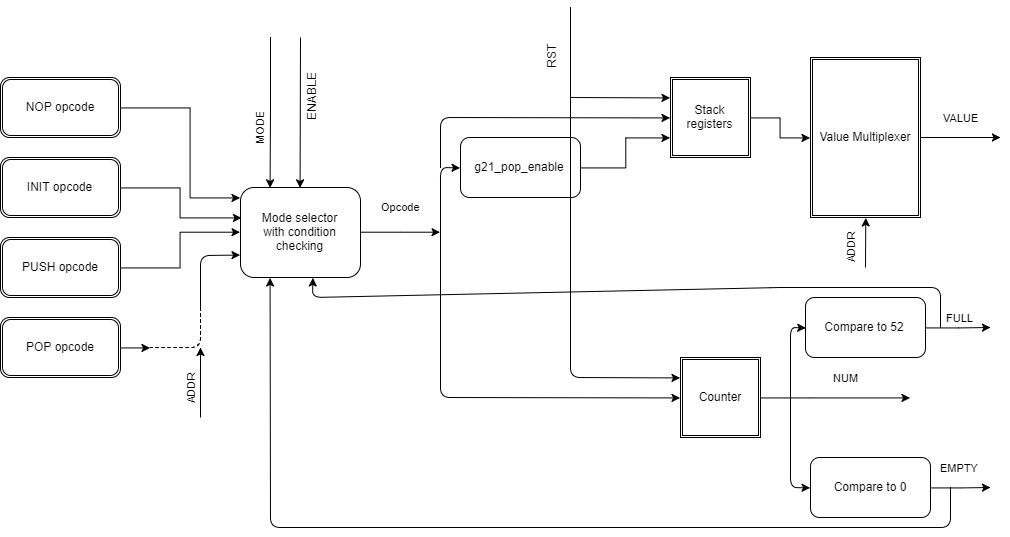


Figure 1 g21\_stack52 functional block diagram

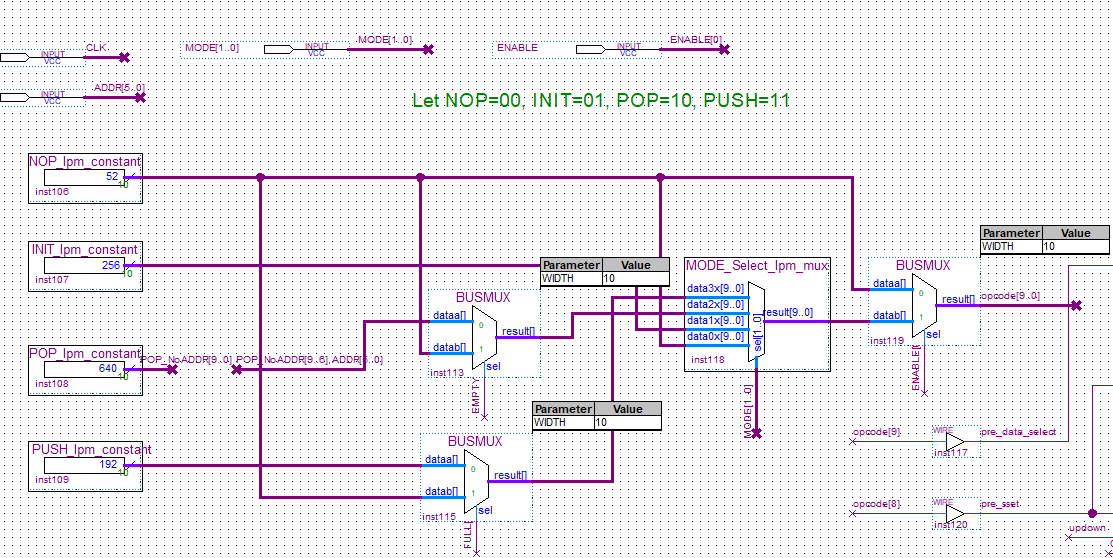


Figure 2 g21\_stack52 opcode selection circuit schematic

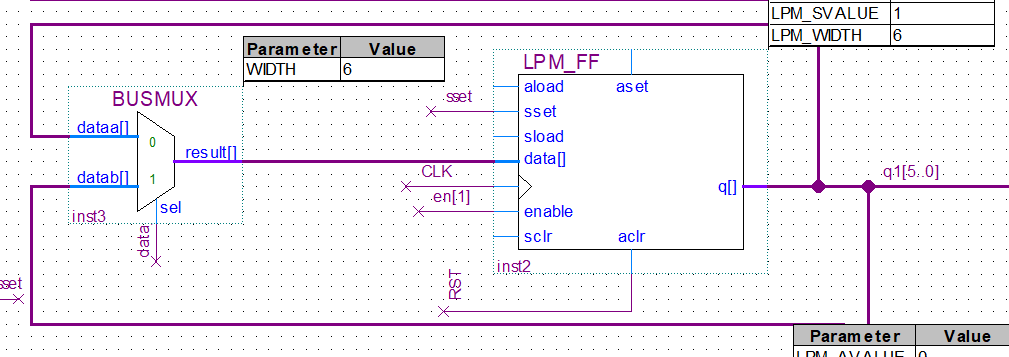


Figure 3 Shift register: BUSMUX and LPM\_FF pair

## **Simulation and Discussion**

Timing simulations were done on the circuit to ensure it was functioning correctly. Tests were done using the simulation file *g21\_stack52.vwf* on all four modes of operation plus the reset function. It should be noted that simulations using a constant enable signal can be misleading as it can mask timing issues as it was our case. It is important in simulations, to use waveforms which are representative of the physical signals which will be passed to the signal. As it is seen in the g21\_test\_bed circuit, g21\_stack52 circuit is to be used with enable signals which lasted a single clock cycle. This such simulation it is important to simulate it as such as well. This can be seen in Figure 6 INIT mode with single clock cycle enable signal. It should be noted that the maximum propagation delay is 19.104 nanoseconds. This indicates the maximum operational frequency of the clock is 52 GHz. Below are snapshots of the simulation results for INIT, POP, PUSH, NOP and RST:

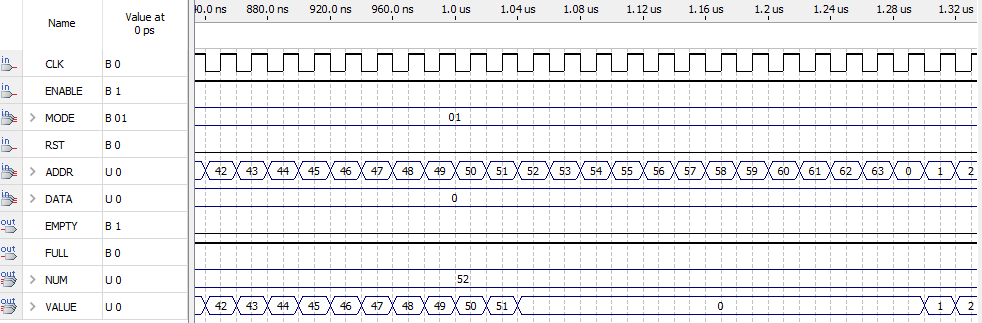


Figure 4 INIT mode simulation results

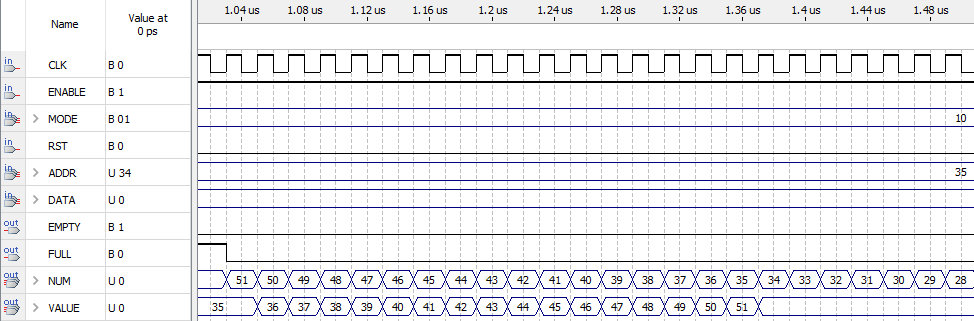


Figure 5 POP mode at ADDR=35 simulation results after INIT operation

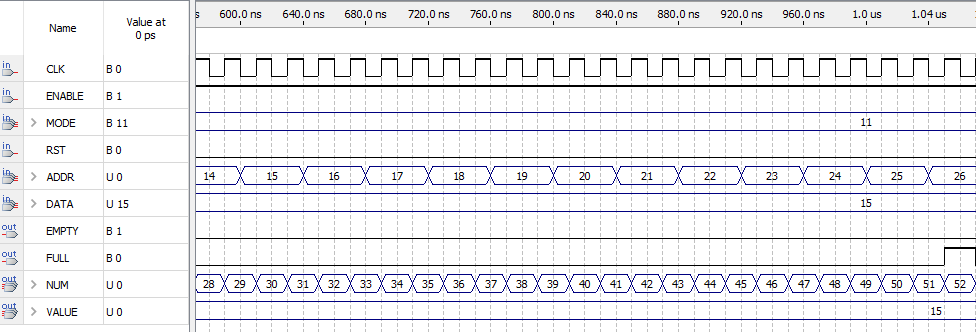


Figure 6 PUSH mode simulation results

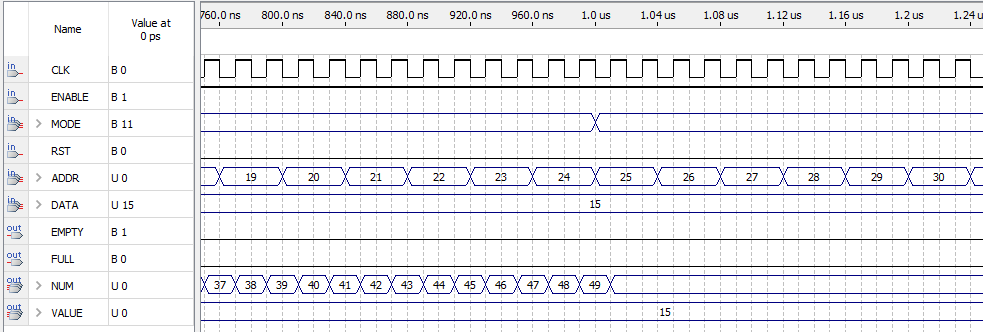


Figure 7 NOP mode simulation results after PUSH operation

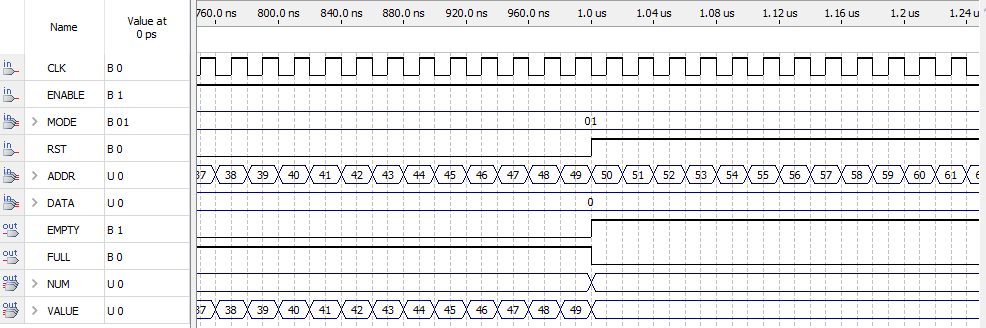


Figure 8 RST operation simulation results

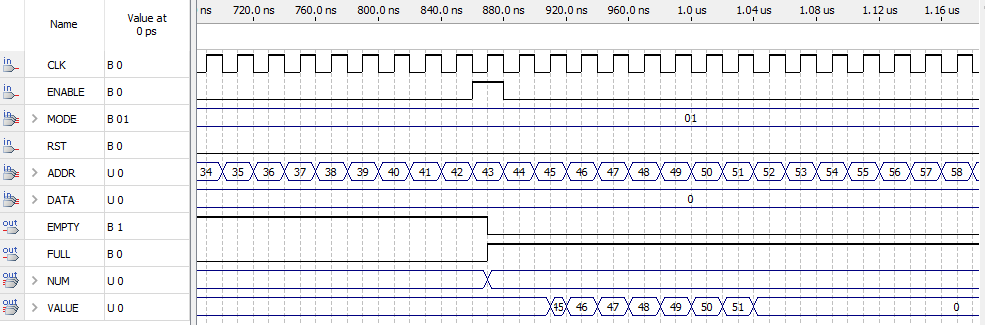


Figure 9 INIT mode with single clock cycle enable signal

# g21\_test\_bed

## **Circuit Function**

The test-bed circuit has 5 inputs and 6 outputs. It is used to implement the g21\_stack52 circuit on the physical Altera DE board.

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Name** | **Length (bit)** | **Purpose** |
| Input | CLK | 1 | Clock Signal |
| Input | pushButton | 1 | Input mapped to a button on the board and triggers a single enable pulse |
| Input | RSTButton | 1 | Input mapped to a button on the board and triggers the reset function |
| Input | dipswitch2 | 6 | Input mapped to 6 switches on the board and defines the value of ADDR |
| Input | dipswitch1 | 2 | Input mapped to 2 switches on the board and defines the value of MODE |
| Output | LEDmod | 7 | 7-segment code for mod13 of VALUE |
| Output | LEDfloor | 7 | 7-segment code for floor of VALUE |
| Output | FULL\* | 1 | Output mapped to a green LED on the board and triggers when the stack is full |
| Output | EMPTY\* | 1 | Output mapped to a red LED on the board and triggers when the stack is empty |

\*Note that the FULL and EMPTY outputs were not required in the lab instructions[1] but they were added to make testing on the board easier.

## **Circuit Design**

The g21\_test\_bed circuit is used to implement the g21\_stack52 circuit on the physical board. To do so, the circuit is comprised of three parts: receive input from user, process information, show result to user. First, the circuit receives information from the user by assigning inputs to appropriate pins on the board. Additionally, a single pulse generator circuit is implemented with the enable signal as to ensure the inherent mechanical “bouncing” of the switch does not cause create noise in the enable signal. Second, the circuit performs its operation based on the input with the g21\_stack52 circuit. Finally, the output is shown the user by lighting the correct LEDs available on the board.

### Single Pulse Generator

The single pulse generator functions as a gate for the enable signal. The enable signal can only be high for one clock cycle every 20 ms. This prevents several pulses going through the circuit which could cause the circuit to push a value 2-3 times instead of once. As seen in the Figure 7 Single pulse generator circuit schematic, the circuit is implemented with a counter. Once the enable signal is high, the counting begins. While the value of the counter is between 1 and 20000000, the register which holds the enable signal is cleared and the counting is enabled. Once the required time has elapsed, another enable pulse may be passed to the rest of the circuit.

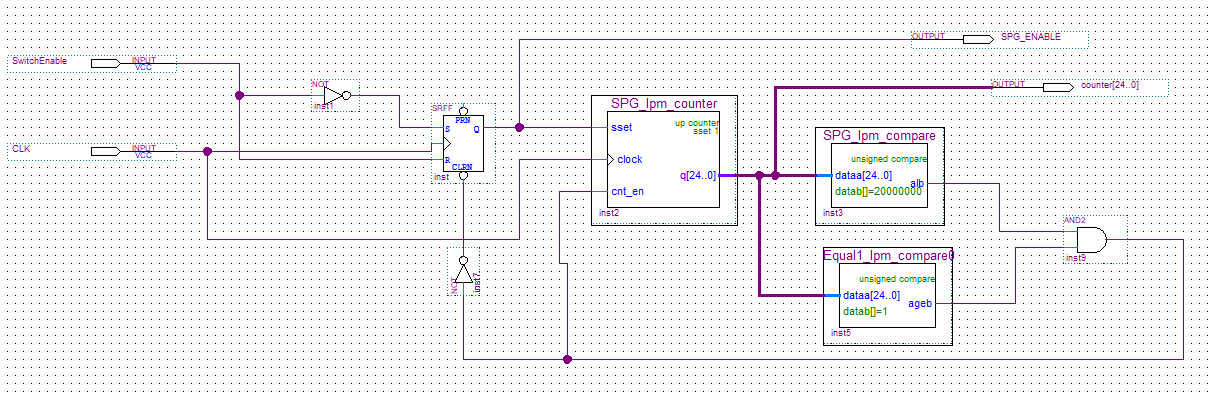


Figure 10 Single pulse generator circuit schematic

### Stack

This is an instance of the stack52 circuit that was described in the **g21\_stack52** section of this report.

### Output Assignments

The outputs are shown to the user by lighting different combinations of LEDs. The FULL and EMPTY outputs are displayed by lighting a green and a red LED. The VALUE is more complicated to display as it is a 6-bit integer. Circuits which were developed in earlier labs are used.

The g21\_Modulo13 circuit from Lab #1 is used to separate the VALUE into the mod13 and its floor. The user can recover the VALUE by multiplying floor by 13 and adding mod13. Each of these values is presented on a 7 segments LED display. The g21\_7\_segment\_decoder from Lab #2 is used to properly map values to the correct pin assignments.

## **Schematic**

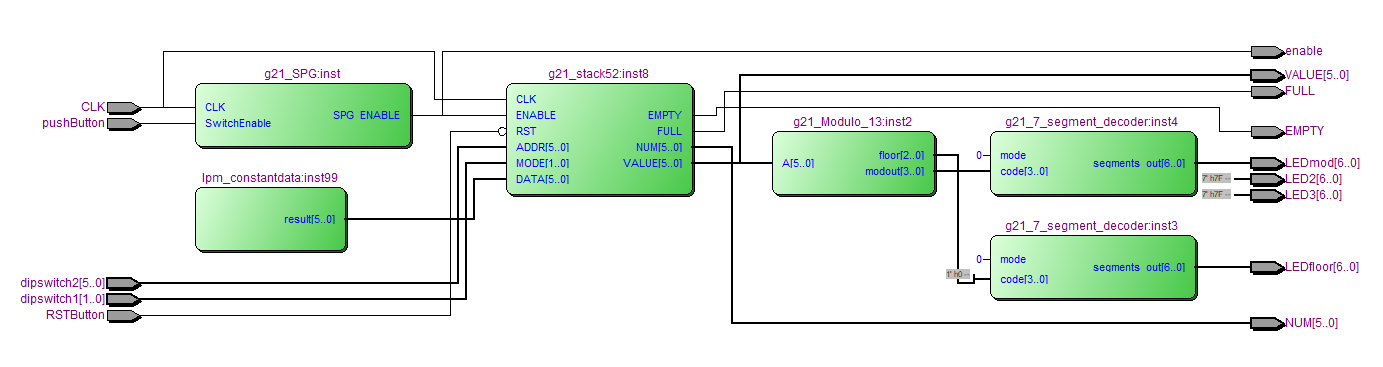


Figure 11 g21\_test\_bed circuit schematic

\*Note that the outputs enable, VALUE and NUM are not described in the Circuit Function because they were only used for debugging purposes. Similarly, LED2 and LED3 are not actual outputs, they simply set the two unused 7-segment displays to be off.

## **Simulation and Discussion**

Timing simulations were done on the circuit to ensure it was functioning correctly. The results of the simulation can be found in Figure 9 g21\_test\_bed simulation results. Tests were done using the simulation file *g21\_.test\_bed\_tim.vwf* on all four modes of operation plus the reset function. Below is a snapshot of the simulation results for INIT. Other modes are not shown as they were already tested for stack52 and the goal of this simulation is to show that the test-bed works as a whole:

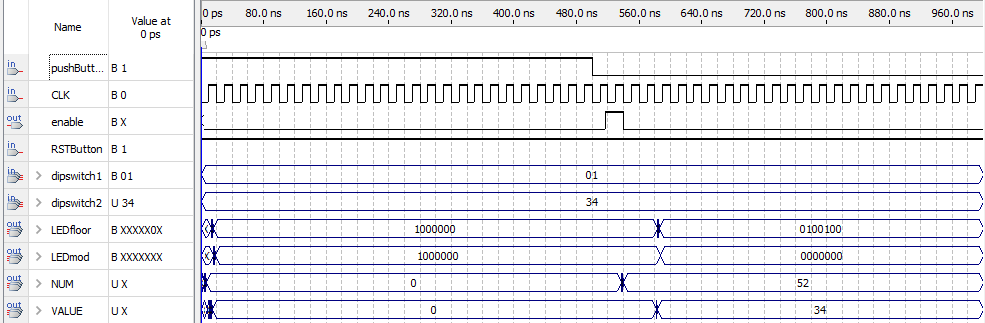


Figure 12 g21\_test\_bed simulation results

From the TimeQuest timing analyzer, it was determined that the longest propagation delay in the circuit was 25.839ns. This means the minimum clock frequency should allow for this delay. The associated maximum frequency is 38.7 GHz.

Furthermore, a summary of resource utilization was extracted from the compilation flow summary. From it, information can be analyzed about the number of logic elements registers and pins that are used by the circuit on the DE1 board:

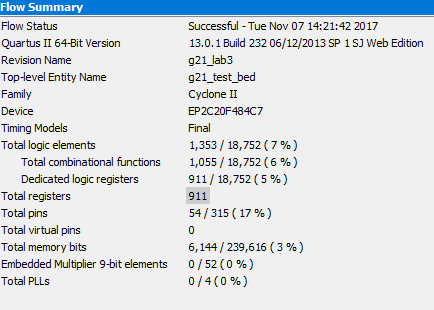


Figure 13 Compilation report flow summary for g21\_test\_bed circuit

# Conclusion

The g21\_stack52 circuit was designed and implemented on the Cyclone II Altera DE board by using a test bed which implemented a single pulse generator. It is important to consider the application of a circuit when simulating it to ensure it functions correctly, as seen with the stack circuit timing issues. Circuits which use the stack component should have a clock frequency inferior to 52 GHz to account for the propagations delays.

# 

# References

[1] Prof. J. Clark, Lab Instructions, “ECSE-323 Digital Systems Design: Lab #3 - Sequential Circuit Design”, Department of Electrical and Computer Engineering, McGill University, Oct. 2017.

[2] *LPM Quick Reference Guide*. Altera Corporation, 1996, pp. 26-28.

[3] "Opcode definition by The Linux Information Project (LINFO)", *Linfo.org*, 2017. [Online]. Available: http://www.linfo.org/opcode.html. [Accessed: 03- Nov- 2017].