Abbas Yadollahi – 260680343
He Qian Wang – 260688073

November 10, 2017
Team A6

Lab 3 Report: gA6 Stack 52

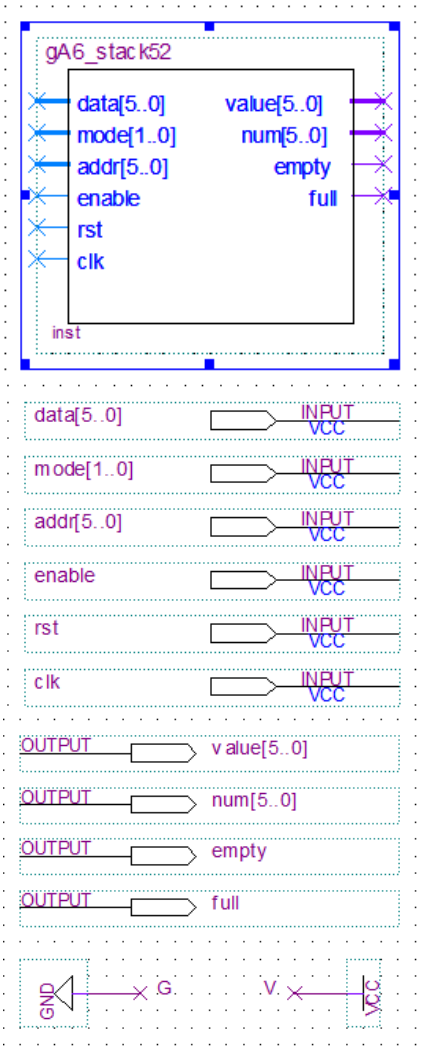
PART 1: 52-BIT STACK

Objective

The first part of the lab required us to build a 52-bit stack circuit that could represent a stack of 52 cards. Contrary to traditional stacks, however, our circuit will have the option to pop elements from anywhere within the stack by using the address of that card. The push function is still the same; the element to be pushed will still be added to the top of the stack. Once the digital circuit was completed, our final goal was to make the entire program work on our FPGA board.

Schematics

52-Bit Stack



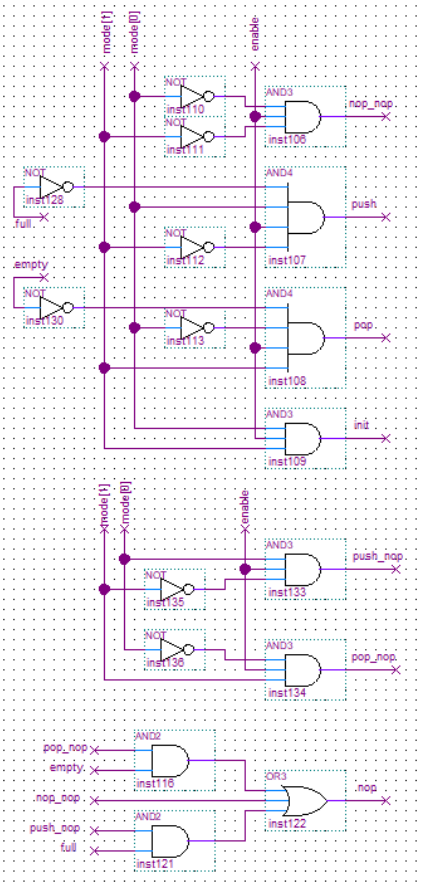
Our stack52 circuit has a total of 6 inputs and 4 outputs.

The following are the 6 inputs:

- data – 6-bit
  - Contains the value of the card that is to be pushed on the stack.
- mode – 2-bit
  - Controls the current mode of operation of the circuit.
- addr – 6-bit
  - Contains the address of the value we want to view or the address of the element to be removed during a POP operation.
- enable – 1-bit
  - Enables the operation specified by the mode input when high.
- rst – 1-bit
  - Enables the reset operation when high.
- clk – 1-bit
  - Our circuit operates on a 20ns clock period.

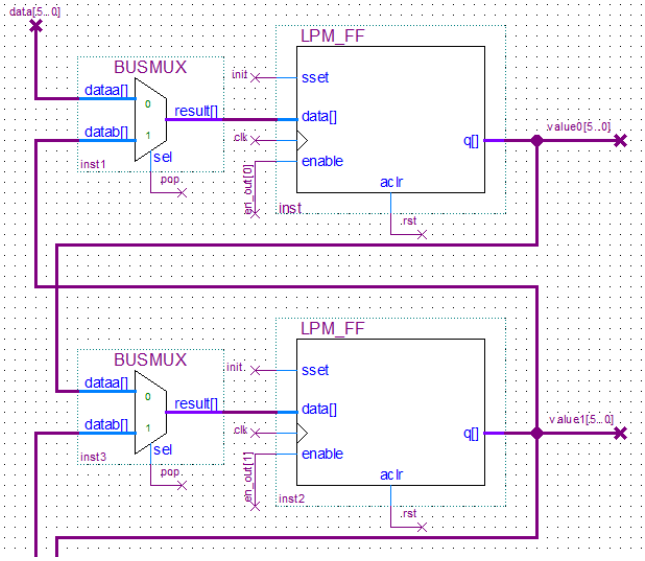
The following are the 4 outputs:

- value – 6-bit
  - Contains the value on the stack pointed to by the *addr* input.
- num – 6-bit
  - Represents the number of elements currently present in the stack.
- empty – 1-bit
  - Indicates if the stack is empty (no elements).
- full – 1-bit
  - Indicates if the stack is full (52 elements).



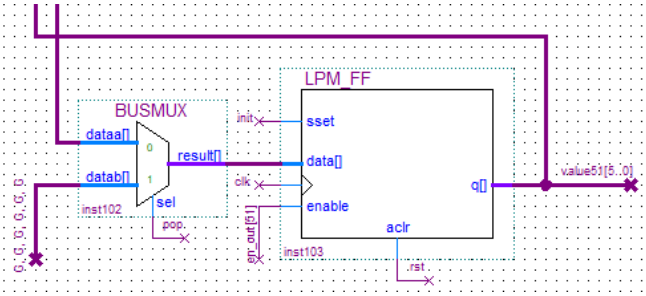
The stack52 circuit has 4 possible modes of operation: NOP, PUSH, POP and INIT. Only a single operation can be active at a given time and will only operate when *enable* = 1.

When NOP mode is on (*mode* = 00), the circuit keeps the current state. The stack52 will not change any values state regardless of what inputs are given, except for the *rst* input which will still operate when high.
When PUSH mode is on (*mode* = 01), the output will depend on other inputs. If *full* = 0, it will push the value of the *data* input onto the top of the stack and push every other element up the chain. The value stored in *num* will increase by 1. If *full* = 1, the circuit will operate as if it was in NOP mode because the stack cannot take in another element.
When POP mode is on (*mode* = 10), the output will depend on other inputs. If *empty* = 0, it will remove the element in the stack pointed to by the *addr* input. The elements stored in the stack slots from 0 up to *addr*-1 will remain unchanged, whereas the elements in *addr*+1 to 52 will be shifted down one slot to fill up the hole left by the popped element.
When INIT mode is on (*mode* = 11), we initialize all the 52 slots of the stack by replacing the currently stored element with a value representing its location in the stack. In other words, the slot 0 will contain the value 0, the slot 1 will contain value 1, and so forth until the last slot 51 contains the value 51. Initializing the stack will also fill it with elements, thus the output *full* = 1.

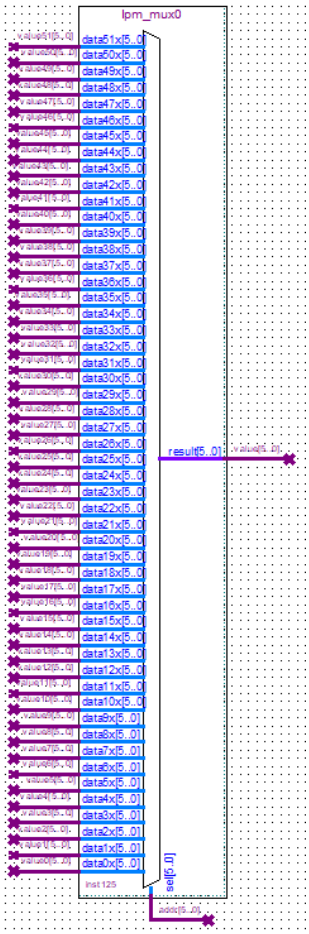


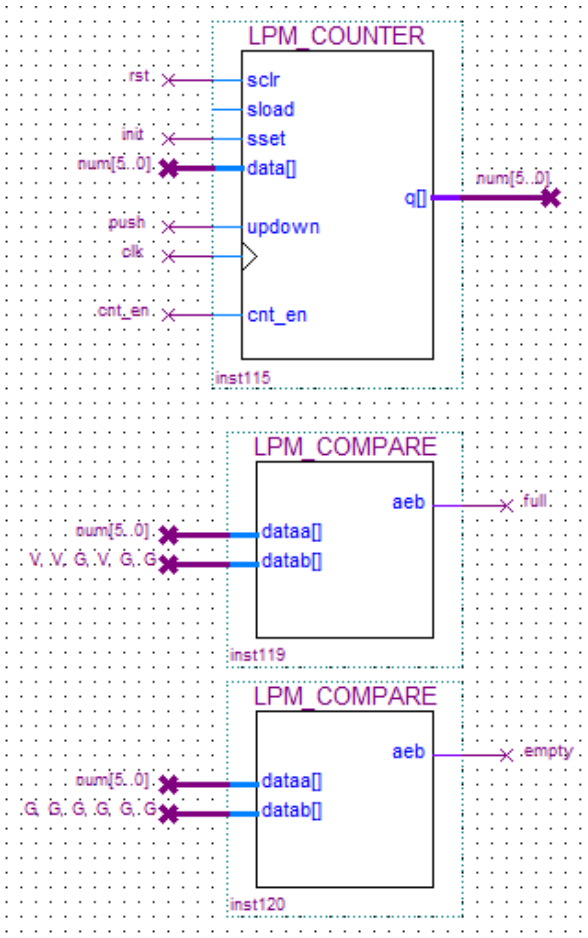
To build the stack52 digital circuit, we used 52 instances of both LPM flipflops and 2-1 mux. The flipflops act as registers which serve to memorize which values are stored at the address. Each flip-flop's *data[]* input is selected by the mux. The inputs feeding into the mux, except for the first (slot 0) and last flipflop (slot 51), are the values of the flipflop one slot lower for 0 and the value of the flipflop one slot higher for 1.

The value of the input *pop* will drive the state of the mux. If *pop* = 0, we are pushing, thus the value from one slot lower is being fed into the flipflop (to shift up). If *pop* = 1, the value from one slot higher is being fed into the flipflop (to shift down).



The exception concerning the first flipflop is that when *pop* = 0, the data to be pushed onto the flipflop is the value contained in the *data* input.
The exception concerning the last flipflop is that when *pop* = 1, there is no flipflop to feed data into the mux, so we hardcoded the value to be fed into the flipflop to 0. As a result, when we pop from the stack, there will still be a null value to fill in the empty slot.

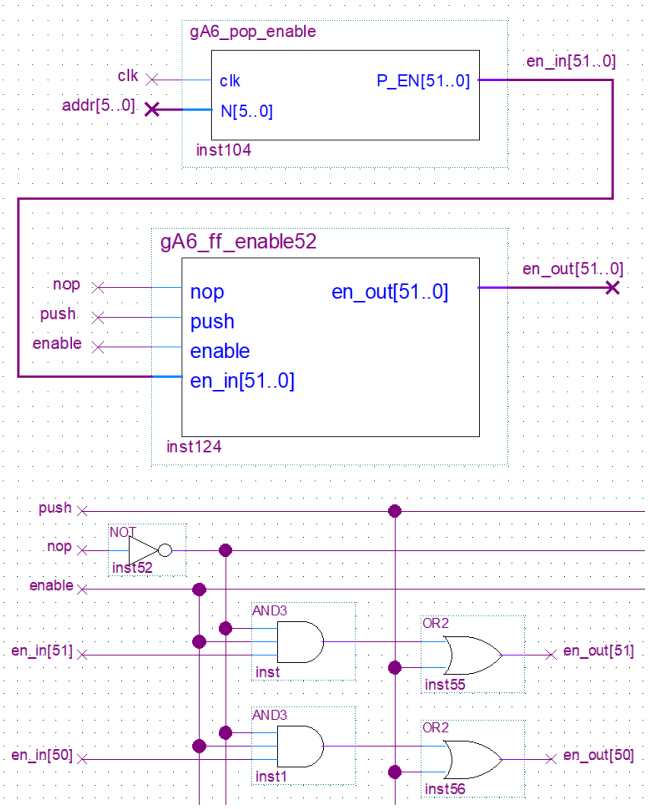




Each flipflop has a total of 5 inputs, one of them being the clock input *clk*, and 1 output. When INIT mode is on and *enable* = 1, the input of the flipflop *sset* = 1, which will automatically set the value of the flipflop to the value set by the parameter LPM_SVALUE. In our circuit, the value of LPM_SVALUE for each flipflop is equal to its location in the stack (0 to 51). When the *rst* input is high, the input of the flipflop *aclr* = 1, which will bypass the clock, being asynchronous, and automatically reset every value to 0. When *enable* = 1, the value of the input *data[]* is the new card to be stored inside the flipflop. Otherwise, the flipflop's state remains the same. The output *q[]* of represents the value that was previously stored in the flipflop (when *enable* = 1) or the value currently held by the flipflop (when *enable* = 0).
The outputs of the flipflops, *value0* to *value51*, are all connected to a 52-6 mux that selects which value to output based on the value of the input *addr*. The output of the mux, which is the output *value* of the stack52 circuit, is the value in the stack pointed by *addr*.

To keep track of the number of cards in the stack, we use an LPM counter which can increment or decrement the count. The counter has a total of 6 inputs and 1 output, with the inputs *sclr*, *sset* and the clock operating in the same manner as those of the flipflop. If *rst* = 1, the counter will be set to 0. If *init* = 1, the counter will be set to 52. The input *data[]* is the output *q[]* of the counter since it reuses the value of *num* to know where the count is currently at. The input *updown* uses the value of *push* to increment when we're in PUSH mode and decrement when we're in POP mode. In the situation where we're in NOP mode, we set *cnt_en* = 0 using a small circuit to prevent the counter from counting down. The output *num* is fed back into the counter and is also the output of the stack52 circuit.

To know whether the stack is empty or full, we use two LPM compare. We compare the value of *num* to 0 and to 52 and set the outputs *empty* and *full* according to the output of the comparator.



The last part of our stack52 circuit is used to control which flipflops are enabled during operations. We are using the 52-bit pop_enable circuit and the file gA6_popup_rom.mif from Lab 2. The input *N* of the pop_enable circuit uses the *addr* input and outputs the corresponding 52-bit number *P_EN* into the input *en_in* of the ff_enable circuit.

The ff_enable is a 52-bit circuit used to determine which flipflop is enabled during operations using the values of *nop*, *push*, *enable* and *en_out*.
If the input *push* = 1, then all flipflops will be enabled. When NOP mode is on, it will force all gates to output 0 since none of the flipflops should be operating. The input *enable* must also be high for any of the flipflops to operate. The last input is *en_in* which depends on the value of the *addr* used in the pop_enable circuit. During a POP operation, we can choose which of the 52 flipflops are enabled using the 52-bit number *en_in*. Each bit of the final 52-bit output *en_out* of the ff_enable circuit will correspond to one of the flipflop's *enable* input.

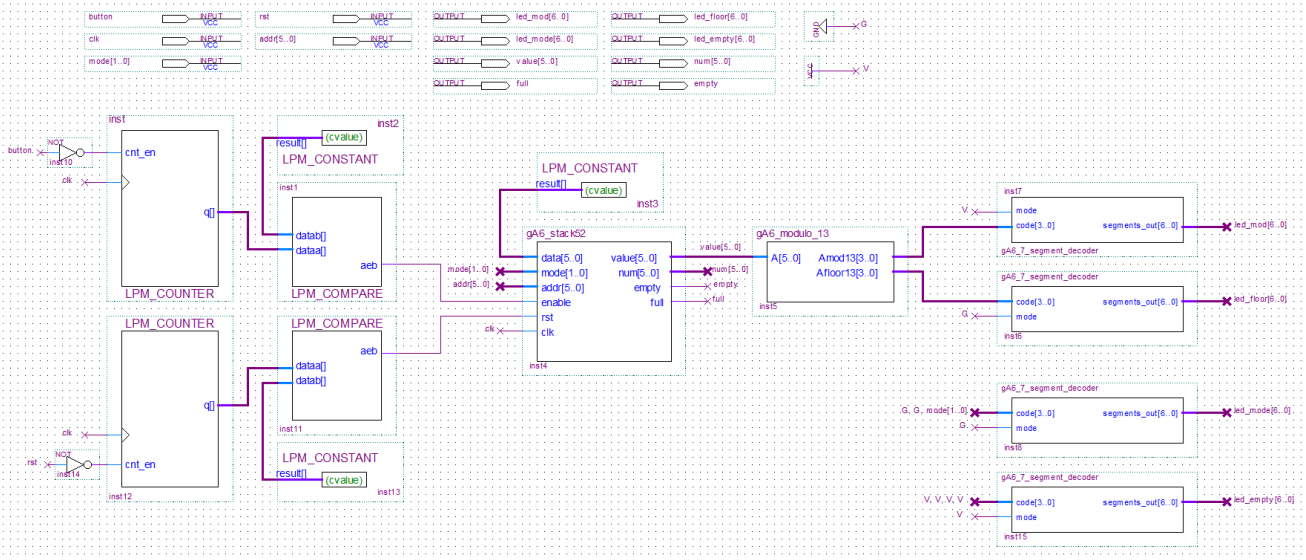
Remarks

There are a few limitations concerning the stack52 circuit.
Firstly, the stack52 is unable to partially initialize a deck. For example, we can't set it up so that only half the stack is filled with cards. When we want to reset, the circuit can either have 0 or 52 cards. In the event where we need only a certain number of cards, we would need to manually pop every element in the stack that we don't need.
Secondly, the stack52 circuit can only hold 52 elements. The issue here is that we have a 6-bit input *addr* meaning we can point to 64 different locations in the stack, even though there can only be a maximum of 52 elements. The circuit outputs *value* = 0 when pointing to non-existent stack locations, but issues could arise from this limitation.

PART 2: 52-BIT TESTBED

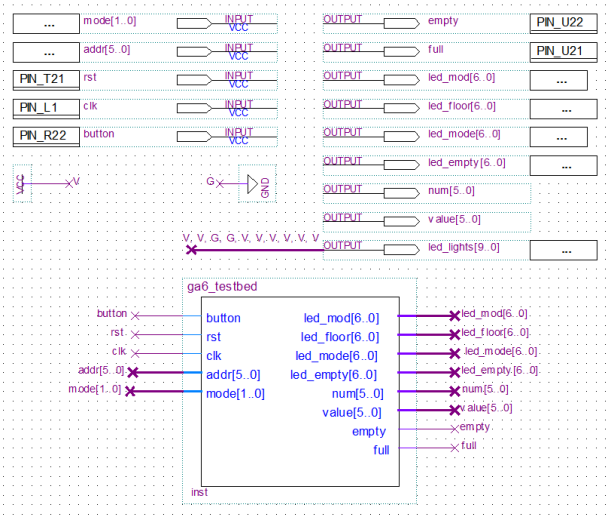
Objective

The stack52 circuit will be connected to a testbed circuit which will allow us to visualize the program on the Cyclone II's 7-segment LED display. Below is the schematic of the testbed circuit. The circuit will be uploaded to the FPGA board and rigorously tested for errors so we can evaluate the performance of the module.



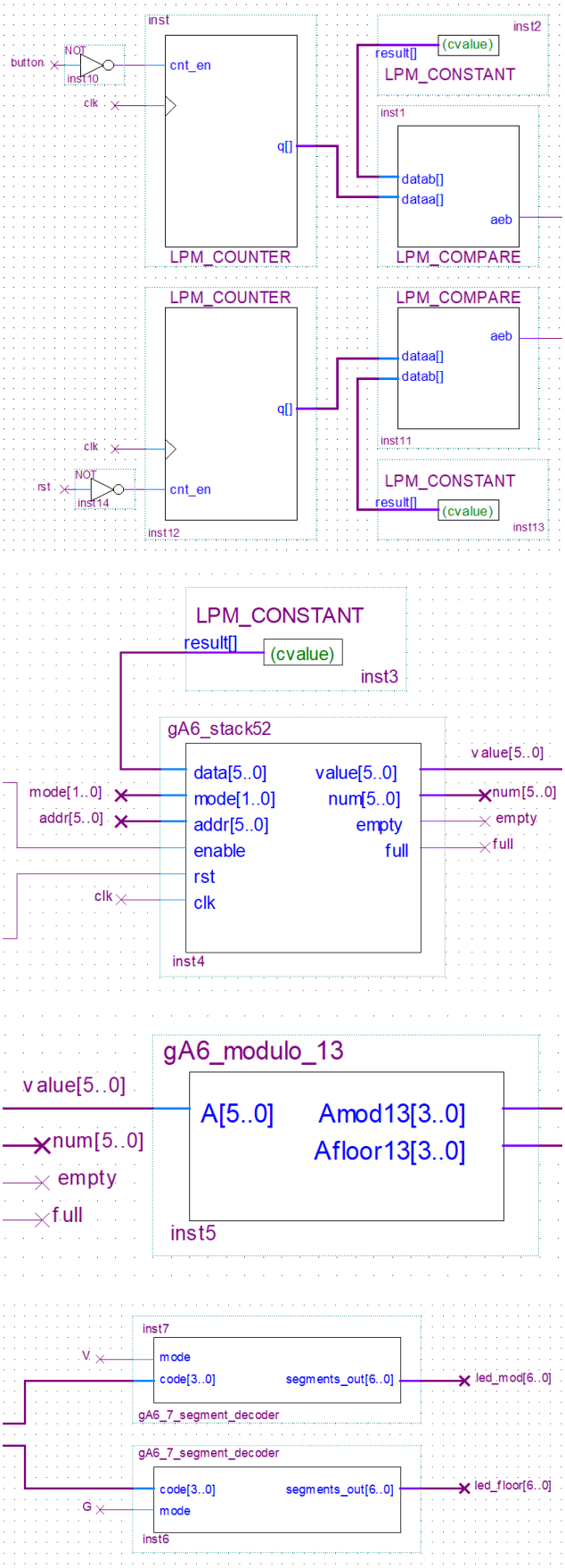
Schematics

52-Bit Testbed



The testbed circuit has a total of 5 inputs and 8 outputs. The 1-bit input button activates the circuit, it acts as enable. The 1-bit input rst activates the reset function of the stack52 circuit. The 6-bit addr input is used to point to a location on the stack. The 2-bit input mode is used to switch between the 4 operations. The clock input clk operates on a 20ns period.

The 7-bit inputs led_mod, led_floor, led_mode and led_empty are the outputs used for the 7-segment LED display of the board. The 6-bit output num indicates how many elements are currently in the stack. The 6-bit output value contains the value on the stack pointed to by the addr input. The 1-bit outputs empty and full indicate whether the stack is empty or full.



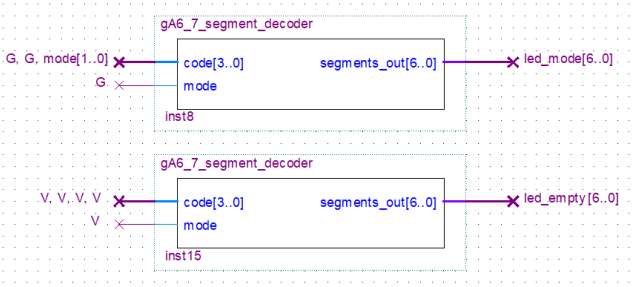
In the first section of the testbed circuit, we built a pulse generator using 2 instances of each LPM counters, LPM comparators and LPM constants. They each count up to 15 000 000 cycles before letting the *button* and *rst* signals through to the circuit. This is to make up for the jittery signals of the FPGA buttons. Through testing, the delay of 15 000 000 cycles was the best fit to prevent any glitches; the proposed delay of only 2 000 000 cycles wasn't high enough to prevent all the debouncing of the buttons.

The inputs button and rst are put through a NOT gate since the buttons are active low, meaning they are high when depressed. When the signal passes through the counters, they are fed into the stack52 circuit.

The outputs of the stack52 circuit depend on the values of the inputs and can be read in more detail in Part 1: 52-Bit Stack. The values of the inputs *mode* and *addr* are assigned according to the corresponding switches on the board. For testing purposes, we use an LPM constant of value 3 to be fed into the *data* input of the stack52. The clock input *clk* is the same as the clock used on the counter and they are all synchronized.

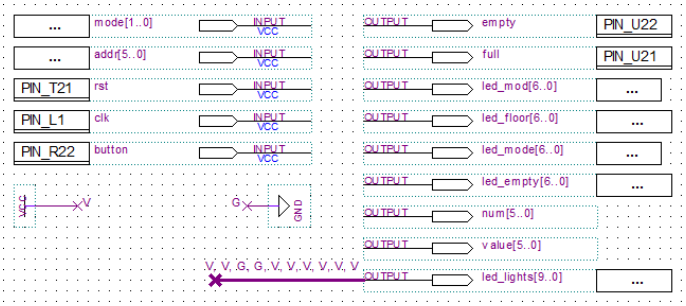
We use the 6-bit output *value* of the stack52 circuit and feed it into the modulo_13 circuit. The two 4-bit outputs *Amod13* and *Afloor13* of the modulo_13 circuit give us the values of $A\ modulo\ 13$ and $floor\left(\frac{A}{13}\right)$, respectively, of the 6-bit input *A*.
The output *Amod13* can hold a value between 0 and 12 and it represents the value of the card.
The output *Afloor13* can hold a value between 0 and 3 and it represents the suit of the card.

Each output of the modulo_13 circuit is connected to a 7_segment_decoder circuit which will take the output's value and produce a number readable by the LED display.
For the *Amod13* output, we set the input *mode* = 1 to display the value of the cards from A to K in the 6-bit output *led_mod*.
For the *Afloor13* output, we set the input *mode* = 0 to display the suit of the cards from 0 to 3 in the 6-bit output *led_floor*.

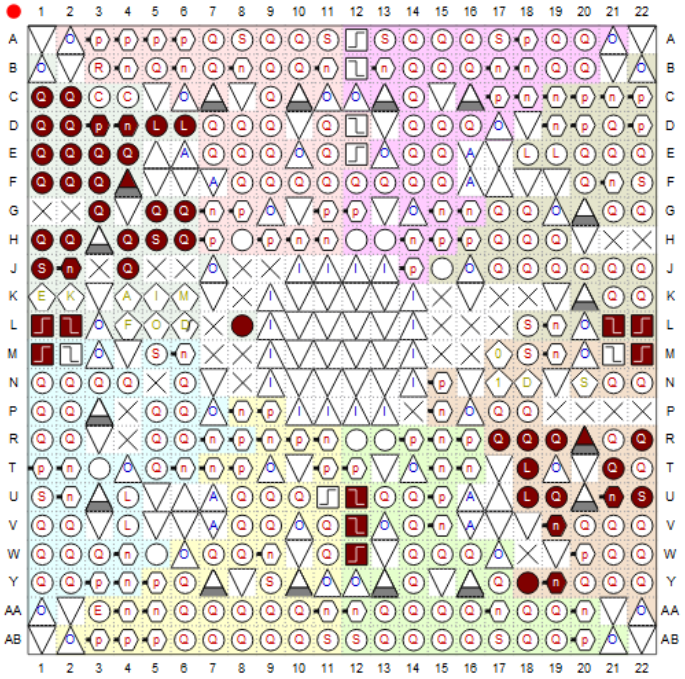


We decided it would help to show more information on the LED display, so we fed the input *mode* into a 7_segment_decoder and output as *led_mode* to show what mode of operation we're currently in. To separate the mode and the card shown on the LED display of the board, we added a hardcoded dash symbol using another 7_segment_decoder circuit and stored the output in *led_empty*.

Pin Planner



We used the pin planner to map most of the inputs and outputs of our testbed circuit to the FPGA board's buttons, switches and LEDs.
Button: button, rst
Switch: mode, addr
Red LED: led_lights
Green LED: empty, full
Display: led_mod, led_floor, led_mode, led_empty



The table on the right indicates the pin placement of every bit that is mapped to the FPGA board. All the inputs and outputs can be controlled or shown using the board.

Inputs				
button	rst	clk	mode[1..0]	
0	0	0	0	1
R22	T21	L1	M1	L2

addr[5..0]					
0	1	2	3	4	5
L22	L21	M22	V12	W12	U12

Outputs	
empty	full
0	0
U22	U21

led_mod[6..0]						
0	1	2	3	4	5	6
J2	J1	H2	H1	F2	F1	E2

led_floor[6..0]						
0	1	2	3	4	5	6
E1	H6	H5	H4	G3	D2	D1

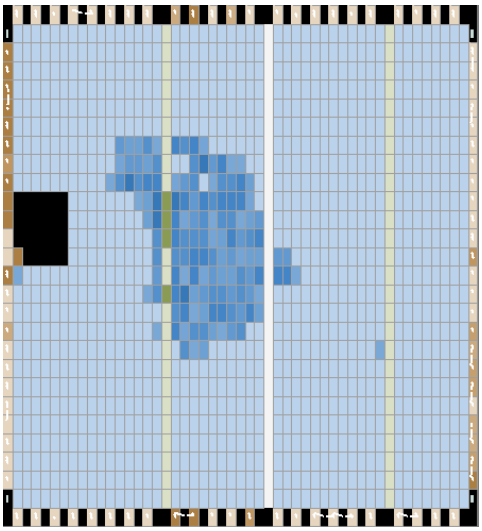
led_mode[6..0]						
0	1	2	3	4	5	6
F4	D5	D6	J4	L8	F3	D4

led_empty[6..0]						
0	1	2	3	4	5	6
G5	G6	C2	C1	E3	E4	D3

led_lights[9..0]				
0	1	2	3	4
R20	R19	U19	Y19	T18
5	6	7	8	9
V19	Y18	U18	R18	R17

Analysis

Chip Planner



As we can see on the floorplan, the mapping of the circuit is compactly fitted onto the FPGA board. The amount of spaced used (dark blue) is quite low compared to the capacity of the board. The block utilization is very low, even though we are using over 500 elements in our circuit. We managed to use approximately 10% of the board's blocks.

Compilation Analysis

	Fmax	Restricted Fmax	Clock Name	Note
1	105.75 MHz	105.75 MHz	clk	

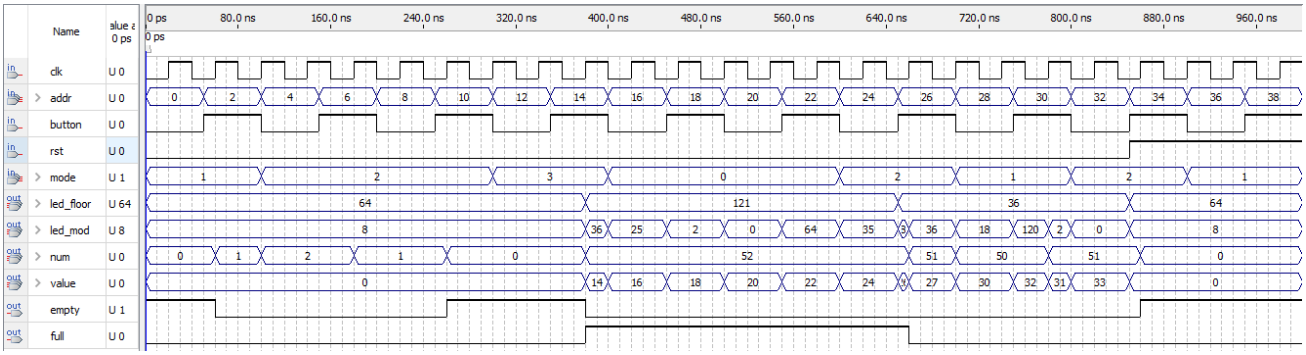
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	2,026 / 18,752 (11 %)
Total combinational functions	1,207 / 18,752 (6 %)
Dedicated logic registers	1,206 / 18,752 (6 %)
Total registers	1206
Total pins	63 / 315 (20 %)
Total virtual pins	0
Total memory bits	9,088 / 239,616 (4 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

The maximum frequency of our digital circuit if $F_{max} = 105.75\ MHz$, which is quick enough considering that our circuit runs at 50 Mhz. After comparing our circuit to those of other colleagues, we notice that our maximum frequency is quite average.

The total propagation delay of our circuit can be calculated using the inverse of the maximum frequency. Thus, we find that $t_p = \frac{1}{F_{max}} = 9.456\ ns$.

Simulations

Waveform

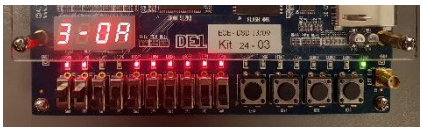


To test the logic of our digital circuit, we simulated all the different modes of operation. We set the clock to a period of 40ns (25MHz). The button is set to a clock with a period of 100ns to simulate the circuit during both enabled and disabled situations. We used a random number generator for the *mode* input to test NOP, PUSH,

POP and INIT at random moments. The input *addr* starts from 0 and increments by 2 every 50ns to test different locations in the stack. Towards the end of the simulation, we set *rst* to high to verify that the stack clears out. When analyzing the waveform, we can confirm the circuit functions properly since all outputs checkout.

SignalTap

It's impossible to test every single situation using waveforms, so we also tested the circuit using SignalTap. The following images demonstrate the main tests we went through.



Name	16	18	20	22	24	26	28	30	32
addr									0
mode									0
button									
rst									
led_floor									64
led_mod									8
led_mode									64
num									0
value									0
empty									
full									

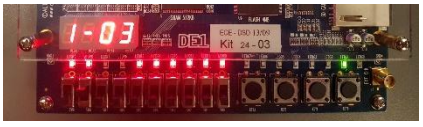
Name	16	18	20	22	24	26	28	30	32
addr									0
mode									3
button									
rst									
led_floor									64
led_mod									8
led_mode									48
num									52
value									0
empty									
full									

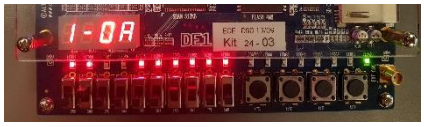
Name	16	18	20	22	24	26	28	30	32
addr									0
mode									2
button									
rst									
led_floor									64
led_mod									36
led_mode									36
num									51
value									1
empty									
full									

The first set of images shows the program at its initial state when it is first started. As we can see, the stack is empty.

The second set shows the testbed after pressing the button in INIT mode, hence *full* being high and *num* = 52.

The third set, we pressed the button in POP mode at the address 0, which is why the value at the top of the stack (*addr* = 0) is now 2. The output *num* also decremented. We notice that neither *empty* or *full* are high.





Name	16	18	20	22	24	26	28	30	32
addr									0
mode									1
button									
rst									
led_floor									64
led_mod									48
led_mode									121
num									52
value									2
empty									
full									

Name	96	98	100	102	104
addr					0
mode					1
button					
rst					
led_floor					64
led_mod					8
led_mode					121
num					0
value					0
empty					
full					

In this set, we pushed a value onto the stack. As we can see, the input *mode* = 1 and the new *value* = 2. Since we only popped one card before, we now have a full stack.

In this set, we pressed the reset button to clear the entire stack. As we can see, the input *rst* is low (active low button), *empty* is high and *num* = 0.

Remarks

The main limitation associated with our program concerns the FPGA board. Our digital circuit has many inputs and outputs, although the board only has one LED display and very limited other ways to display or control information on the board. As a result, we must neglect a lot of the valuable information.
Another issue, that also concerns the lack of inputs on the board, is that we cannot choose what value we want to push onto the stack. As discussed earlier, we use a constant value of 3 for the *data* input of the stack52 circuit since we don't have any more switches that can be used on the FPGA board.

REFERENCES

Altera. *LPM Quick Reference Guide*, December 1996
Altera. *Development and Education Board*, 2012
Clark, J. *Lab #3 – Sequential Circuit Design*, Fall 2017, McGill University