Bittar, Catherine (260687735) 8 December 2017

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

ECSE 323: Digital System Design

Lab #5: Card Game System Integration

# 

Table of Contents

[Introduction 2](#_Toc500327314)

[Deal\_FSM 3](#_Toc500327315)

[Circuit Function 3](#_Toc500327316)

[Circuit Design 4](#_Toc500327317)

[Start 5](#_Toc500327318)

[Initial card deal 5](#_Toc500327319)

[Player play 6](#_Toc500327320)

[Dealer play 6](#_Toc500327321)

[Winner determination 7](#_Toc500327322)

[Schematic 7](#_Toc500327323)

[Discussion 7](#_Toc500327324)

[g21\_test\_bed 8](#_Toc500327325)

[Circuit Function 8](#_Toc500327326)

[Circuit Design 9](#_Toc500327327)

[Single Pulse Generator 9](#_Toc500327328)

[Stack 10](#_Toc500327329)

[Output Assignments 10](#_Toc500327330)

[Schematic 10](#_Toc500327331)

[Simulation and Discussion 10](#_Toc500327332)

[Conclusion 12](#_Toc500327333)

[References 13](#_Toc500327334)

List of Figures

[Figure 1 g21\_stack52 functional block diagram 7](#_Toc498085474)

[Figure 2 g21\_stack52 opcode selection circuit schematic 7](#_Toc498085475)

[Figure 3 Shift register: BUSMUX and LPM\_FF pair 8](#_Toc498085476)

[Figure 4 INIT mode simulation results 8](#_Toc498085477)

[Figure 5 POP mode at ADDR=35 simulation results after INIT operation 9](#_Toc498085478)

[Figure 6 PUSH mode simulation results 9](#_Toc498085479)

[Figure 7 NOP mode simulation results after PUSH operation 9](#_Toc498085480)

[Figure 8 RST operation simulation results 10](#_Toc498085481)

[Figure 9 INIT mode with single clock cycle enable signal 10](#_Toc498085482)

[Figure 10 Single pulse generator circuit schematic 12](#_Toc498085483)

[Figure 11 g21\_test\_bed circuit schematic 12](#_Toc498085484)

[Figure 12 g21\_test\_bed simulation results 13](#_Toc498085485)

[Figure 13 g21\_test\_bed SignalTap simulation results 13](#_Toc498085486)

[Figure 14 Compilation report flow summary for g21\_test\_bed circuit 14](#_Toc498085487)

# 

# Introduction

The purpose of this lab was to complete the design of a Blackjack card game by integrating various components which were designed during the last four labs. This includes the stack, the rules modules, as well as smaller components like the 7-segment decoder. This left two components to design: the controller FSM, and the data path which includes the user interface. Both components were successfully tested directly on the Altera DE1 board.

The design was completed using the FPGA design software Quartus. This design was part of the project file *g21\_lab5.QPF*.

# Deal\_FSM

## **Circuit Function**

The Deal\_FSM circuit is a finite state machine with 14 inputs and 12 outputs defined below. The inputs are used to determine the next state, while the outputs are used as control signals for the rest of the circuit.

Table Deal\_FSM inputs and outputs

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Name** | **Length (bit)** | **Purpose** |
| Input | Request\_Deal | 1 | Button press – indicates player wants another card. |
| Input | Stop | 1 | Button press – indicates player has finished playing. |
| Input | INIT | 1 | Button press – indicates start of a new hand. |
| Input | reset | 1 | Button press – indicates player want to reset game. |
| Input | Clock | 1 | Clock signal |
| Input | Dealer\_legal | 1 | True if current dealer total is less than 21. |
| Input | Player\_legal | 1 | True if current player total is less than 21. |
| Input | Dealer\_total | 5 | Current dealer sum. |
| Input | Player\_total | 5 | Current player sum. |
| Input | Dealer\_high | 1 | True when dealer sum is greater than player sum. |
| Input | Player\_high | 1 | True when player sum is greater than dealer sum. |
| Input | Game\_over | 1 | True if game is over (one player has won 3 out of 5 games). |
| Input | score\_dealer | 2 | Number of hands won by dealer in current game. |
| Input | score\_player | 2 | Number of hands won by player in current game. |
| Output | rand\_seed | 1 | Starts the random address generator. |
| Output | Clear\_count | 1 | Clears the counters for the number of hands won. |
| Output | Dealer\_Enable | 1 | Enables card to be dealt to the dealer. |
| Output | Player\_Enable | 1 | Enables card to be dealt to the player. |
| Output | player\_turn | 1 | Lights an LED to indicate player turn. |
| Output | Compare\_enable | 1 | Enables the comparison of the dealer and player sums. |
| Output | Count\_dealer | 1 | Increment the hands won by dealer in current game. |
| Output | Count\_player | 1 | Increment the hands won by player in current game. |
| Output | Count\_gd | 1 | Increment games won by dealer. |
| Output | Count\_gp | 1 | Increment games won by player. |

The circuit has twenty states which enable and disable different output signals that control the data path circuit. Please see the design file *Deal\_FSM.vhd* for the details of the state transitions conditions.

## **Circuit Design**

The circuit is described in VHDL design file *Deal\_FSM.vhd*. Although the instructions [1] recommended the use of a second FSM to control the computer player, a design decision was made to integrate both in one FSM. This had the advantage of simplifying its implementation. This FSM can be broken up into five distinct operations, although each of these is composed of several states: start of hand, initial card deal, player play, dealer play, and winner determination.

### Start

The start operation initializes the circuit and waits for the user to start the game. During this operation, the random address generator is started with a seed, and the deck of cards is initialized. This operation also handles the total game count. Three states make up the start operation.

Table Start operation states and signal assigments

|  |  |  |
| --- | --- | --- |
| State | Function | Signal Assignments |
| rand\_seed | Initialize the random address generator | Rand\_seed <= ‘1’; |
| waiting | Wait for user to press start button. Reset signals to ‘0’. | Dealer\_Enable <= '0';  Player\_Enable <= '0';  count\_dealer <= '0';  count\_player <= '0';  Compare\_enable <= '0'; player\_turn <= '0'; count\_gp <= '0';  count\_gd <= '0';  rand\_seed <= '0'; |
| ready | Check if game is over and increment dealer or player game totals. | if(game\_over = ‘1’ and player\_score = ‘11’) then count\_gp <= ‘1’;  if(game\_over = ‘1’ and dealer\_score = ‘11’) then count\_gd <= ‘1’; |

### Initial card deal

This operation consists of dealing out two cards to both the dealer and the player and checking if either has already reached 21. If either player reaches 21, the hand terminates. Note that the dealer’s first comparison occurs before the players, as the dealer wins if there is ever a tie. Six states make up the initial card deal operation.

Table Initial card deal operation states and signal assigments

|  |  |  |
| --- | --- | --- |
| State | Function | Signal Assignments |
| dealer\_card1 | Deal first card to dealer. Clears hand totals if game was over. Disables game count signals. | Dealer\_enable <= ‘1’;  count\_gp <= ‘0’;  count\_gd <= ‘0’;  if(game\_over = ‘1’) then Clear\_count <= ‘1’; |
| player\_card1 | Deal first card to player. Disables clear hand signal. | Player\_enable <= ‘1’;  Dealer\_enable <= ‘0’;  Clear\_count <= ‘0’; |
| dealer\_card2 | Deal second card to dealer. | Dealer\_enable <= ‘1’;  Player\_enable <= ‘0’; |
| player\_card2 | Deal second card to player. | Player\_enable <= ‘1’;  Dealer\_enable <= ‘0’; |
| dealer\_fc | Verify if dealer has reached 21. | Player\_enable <= ‘0’;  Dealer\_enable <= ‘0’; |
| player\_fc | Verify if player has reached 21. | Player\_enable <= ‘0’;  Dealer\_enable <= ‘0’; |

### Player play

This operation allows the player to request additional cards. If the player reaches 21, he wins. If the player exceeds 21, the dealer wins. Four states make up the player play operation.

Table Player play operation states and signal assigments

|  |  |  |
| --- | --- | --- |
| State | Function | Signal Assignments |
| player\_wait | Wait for player to ask for a card or indicate he is finished. | player\_turn <= ‘1’; |
| player\_play | Enable card deal to the player. | Player\_enable <= ‘1’;  player\_turn <= ‘0’; |
| player\_en | Disable the card deal so only one card is dealt. | Player\_enable <= ‘0’; |
| player\_test | Verify if player has reached or exceeded 21. | Player\_enable <= ‘0’; |

### Dealer play

The play of the dealer replaces the Computer\_FSM specified by the instructions [1]. The operation requests cards until the dealer total has exceeded 16. Four states make up the dealer play operation.

Table Dealer play operation states and signal assigments

|  |  |  |
| --- | --- | --- |
| State | Function | Signal Assignments |
| dealer\_wait | Check if dealer has exceeded 16 or reached 21. | player\_turn <= ‘0’; |
| dealer\_play | Enable card deal to the dealer. | Dealer\_enable <= ‘1’; |
| dealer\_en | Disable the card deal so only one card is dealt. | Dealer\_enable <= ‘0’; |
| dealer\_test | Verify is dealer has exceeded 21. | Dealer\_enable <= ‘0’; |

### Winner determination

The winner is determined by comparing the totals of the dealer and player if neither has gone bust or reached 21. Note that the dealer breaks any ties. This operation handles the hands won count. Three states make up the winner determination operation.

Table Winner determination operation states and signal assigments

|  |  |  |
| --- | --- | --- |
| State | Function | Signal Assignments |
| compare | Enable external comparator module. | Compare\_enable <= ‘1’; |
| dealer\_win | Increment dealer hands won count. | count\_dealer <= ‘1’; |
| player\_win | Increment player hands won count. | count\_player <= ‘1’; |

## **Schematic**

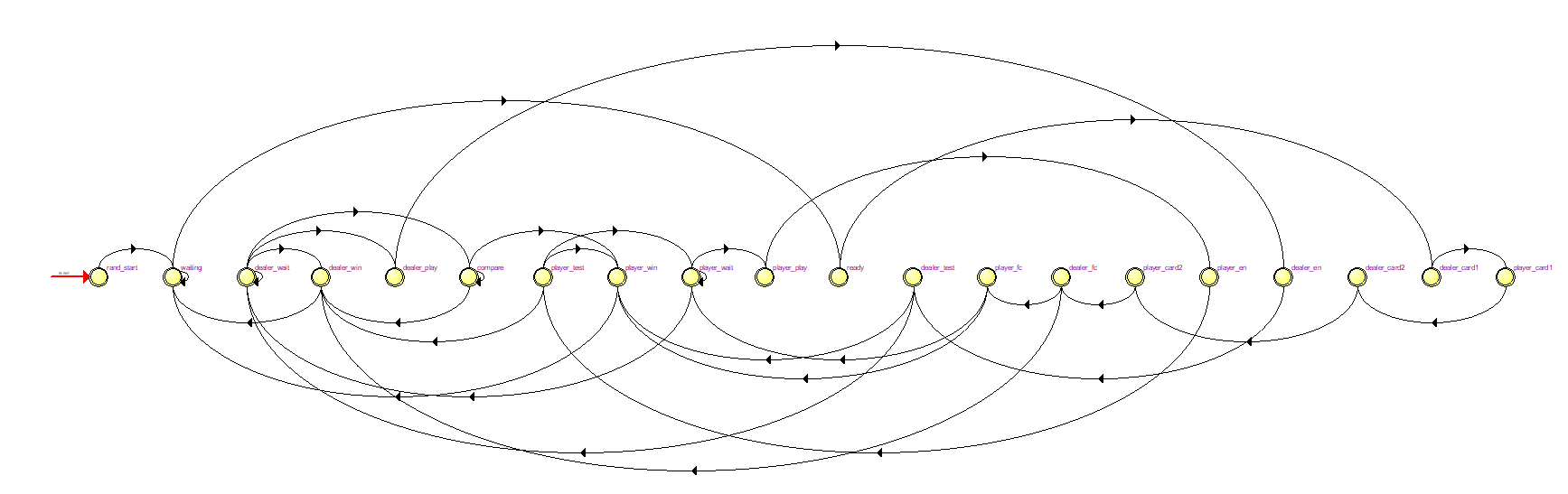


Figure Deal\_FSM generated State Machine diagram

## **Discussion**

No simulations were completed for the FSM because of the sheer quantity of states. It was found to be simpler to integrate the FSM into the test-bed and test it directly on the Altera DE1 board.

The most important design decision made here was to integrate the Computer\_FSM as a part of this controller. This had the advantage of simplifying the implementation of FSMs in the overall circuit. With a single controller, the control signals would all come from the same module as opposed to two different ones, as would be the case with a separate FSM for the computer player. The limitation lies in further implementations of the FSM. As the dealer’s play is effectively hard-coded to be controlled by the computer, it would be far more complex to exchange the role of the computer. This would useful if the game were expanded to more players and the computer would play for a different player. However, because this was the final lab of the current project, it was decided that this simplification of the controllers was acceptable.

# 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 DE1 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 10 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.

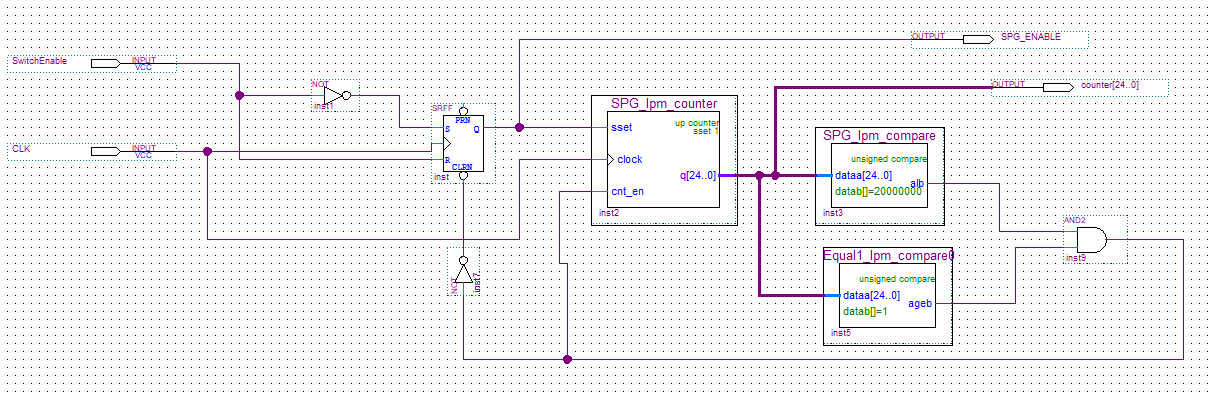


Figure 8 Single pulse generator circuit schematic

### 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**

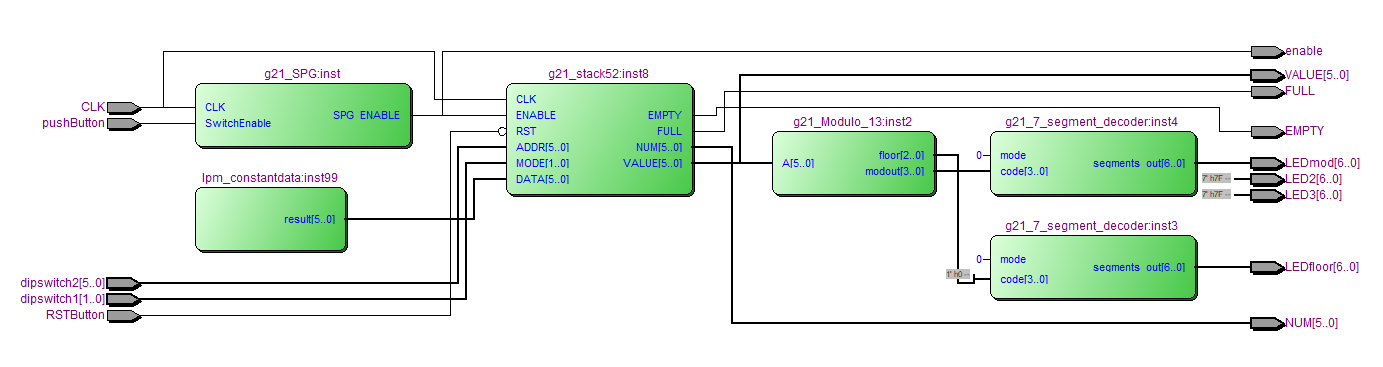


Figure 9 g21\_test\_bed circuit schematic

\*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 12 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:

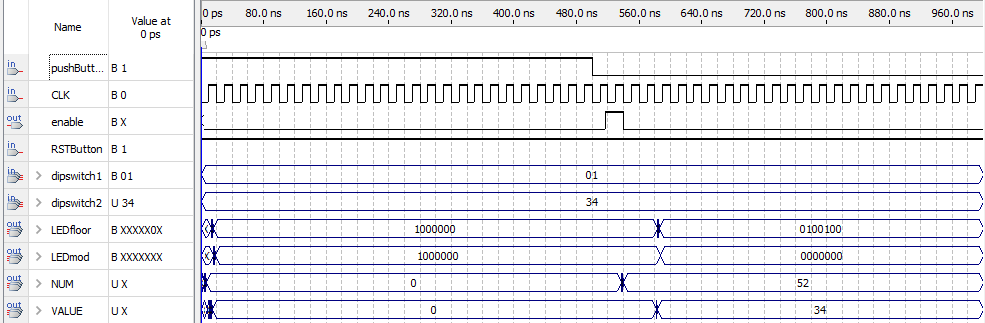


Figure 10 g21\_test\_bed simulation results

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 MHz.

A SignalTap simulation wad also conducted to ensure that the circuit was functional on the DE1 board and not only when doing regular timing simulations. Here is a snapshot of the SignalTap results when ADDR (dipswitch2) = 4 and MODE (dipswitch1) = INIT:

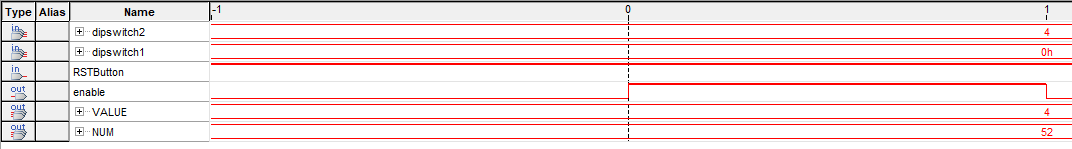


Figure 11 g21\_test\_bed SignalTap simulation results

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:

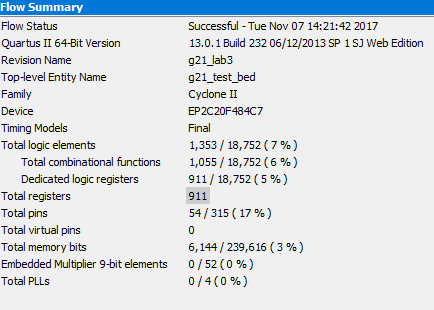


Figure 12 Compilation report flow summary for g21\_test\_bed circuit

# 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 MHz (which is fine since the DE1 board operates at 50 MHz) to account for the propagations delays.

# References

[1] Prof. J. Clark, Lab Instructions, “ECSE-323 Digital Systems Design: Lab #5 – System Integration for the Card Game”, Department of Electrical and Computer Engineering, McGill University, Nov. 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].