Gareth Peters (260678626)
Peter Quinn (260689207)

December 7, 2017
ECSE 323

Lab 5: controller_FSM, computer_FSM, lab5_testbed

Gareth Peters (260678626)
Peter Quinn (260689207)

December 7, 2017
ECSE 323

Gareth Peters (260678626)
Peter Quinn (260689207)

December 7, 2017
ECSE 323

<u>Introduction</u>

In this lab all previously built components were integrated into one final working game of 21 [1].

The complete system required two new components to be designed, a controller FSM and a computer FSM. These two new components are then combined with other circuits from previous labs to create a final working testbed to play the game. The two new circuits are discussed below.

g07_controller_fsm asserts the control signals for the entire system. It reads the inputs from the buttons to determine the actions the player wishes to perform and enables the correct data paths in the correct order to play the game. It also determines if the computer or player has won a round and controls the signals for the computer FSM.

g07_computer_fsm represents the dealer player of the game. It decides what actions to take given information based on the sum of its hand and whether or not it is his turn or not.

The complete circuit that allows the user to play 21 is constructed in g07_lab5_testbed. The testbed circuit is then uploaded to the board to play the game using the UI of the board. The buttons and dip switches allow the user to play their turn and select the outputs they wish to see on the LED displays. The circuit was found to be fully functional and allowed the user to successfully play games of 21.

Gareth Peters (260678626)
Peter Quinn (260689207)

December 7, 2017
ECSE 323

g07_controller_FSM

Design Method

This circuit is designed to control the gameplay of a game of 21 on a Cyclone II FPGA.

Inputs

hit	Connected to a push button. Allows the user to perform the hit action during their turn. Also used to trigger transitions to the next state in certain states (see the flow chart in Fig. 1)
stay	Connected to a push button. Allows the user to perform the hit action during their turn.
d_bust	Indicates if the dealer has busted (hand exceeded 21)
p_bust	Indicates if the player has busted (hand exceeded 21)
d_done	Indicates that the dealer has its turn
reset_sig	Resets the controller to its initial state (init_rand)
clock	The synchronous clock used in the system
d_sum[5..0]	The dealer's hand
p_sum[5..0]	The player's hand
new_card[5..0]	The most recent card taken from the deck

Table 1: The inputs of the g07_controller_FSM and their descriptions

Outputs

en_p	Asserted to add a card to the player's hand
en_d	Asserted to add a card to the dealer's hand
p_won	Asserted to indicate that the player has won a round. Increments a counter
d_won	Asserted to indicate that the dealer has won a round. Increments a counter
init	Asserted when starting a new round. Resets the deck and the players hands

d_turn	Indicates when it is the dealer’s turn. Used by g07_computer_FSM
p_turn	Indicates when it is the player’s turn. Connected to an LED light to signal the player that it is their turn
rand_init	Initializes the random number generator
load	Asserted to store a new random card in a flip flop and remove it from the deck

Table 2: The outputs of the g07_controller_FSM and their descriptions

Logic

The circuit initializes the random number generator, initializes the deck (g07_stack52) with 52 cards, then deals 2 cards to each the player and the dealer, which are stored in a “hand”. If the dealer has been dealt 21, the controller moves to the dealer_won state, otherwise the player is allowed to choose whether to take another card (hit) or pass their turn (stay). These inputs are given by push buttons on the board. If the player's total exceeds 21 with the addition of a new card, the player “busts”, and the dealer wins, so the controller moves to the dealer_won state. When the player chooses to end their turn, the dealer is allowed to play (see g07_computer_fsm for details on how the dealer plays). When the dealer is finished playing, the scores of the dealer and the player are compared to determine the winner. The winner is the hand with the higher total, without exceeding 21. In the event of a tie, the dealer wins. After the winner has been determined, the controller moves into a state that asserts the signal to increment either the player or the dealer win counter. The controller then returns to reset the deck to start a new round.

Dealing cards is broken up into 2 stages in order to function properly with the surrounding circuitry. First, a random card is removed from the stack and loaded in a flip flop temporarily. The input new_card is used to recognize when a new card has been loaded by comparing consecutive values of the flip flop. The controller only goes to the next state when the new card has been properly loaded. Second, the card in the flip flop is added into the correct hand. The input dealer_sum and player_sum are used to detect when the operation has completed before moving to the next state.

Two wait states, wait_rand and reset_wait, are included. Wait_rand waits for an input from the user. This is to ensure that random number generator produces random numbers that depend on a time delay based on the user input, which will be more random than initiating everything when the game starts. By tieing the generation of random numbers to user input (which is much slower than the rate at which random numbers are generated, 20 ns), it is unpredictable where cards from the stack are going to be dealt from. Reset_wait includes a delay of 12 clock cycles, which is more than long enough to ensure that all signals have fully propagated.

The end_game state is intended to allow the player to review the final state of the round before beginning a new one by pressing the hit button. A flowchart describing the circuit can be seen in Fig. 1.

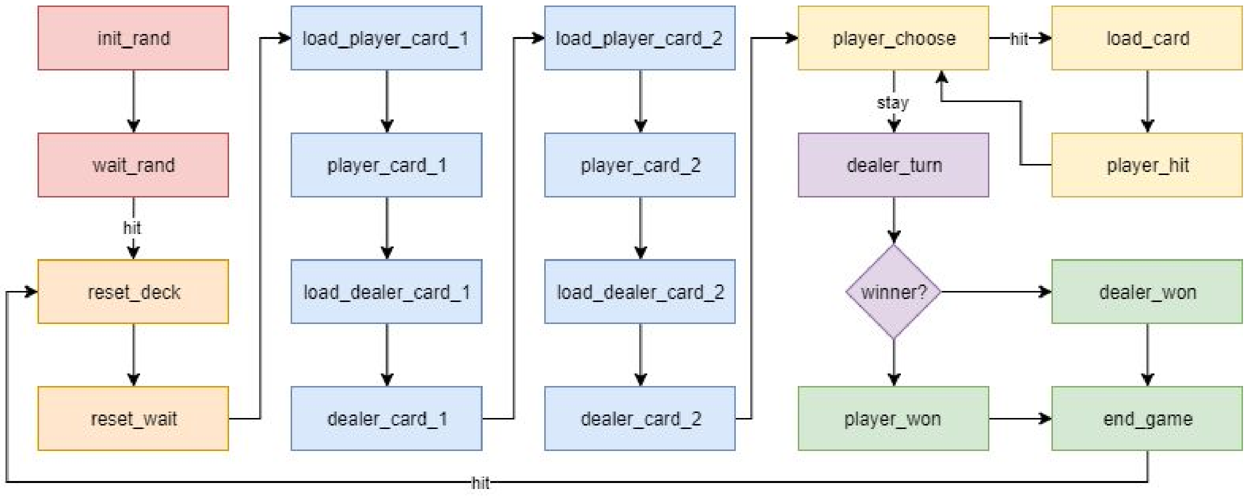

Fig. 1: Flowchart describing the logic implemented in g07_controller_FSM. Labelled arrows represents inputs that must be received to trigger the state transition. Unlabelled arrows represent automatic state changes.The boxes represent the different states. The decision box “winner?” is placed for clarity. The logic is implemented in the dealer_turn state to determine the winner after the dealer has played.

Schematic

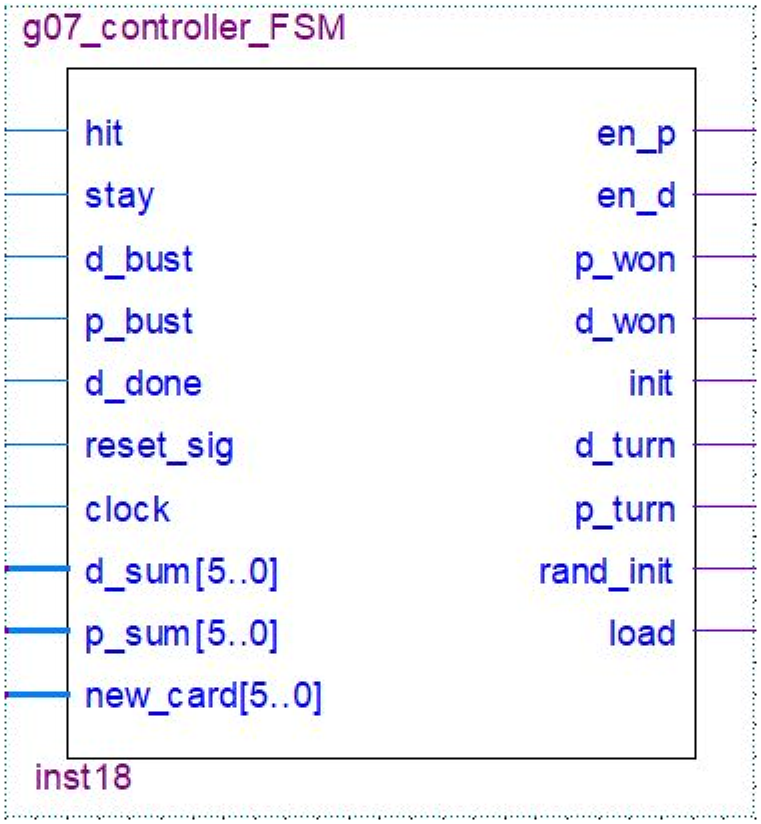

Fig. 2: Symbol of g07_controller FSM

The circuit was built by writing the code for the state machine in VHDL using the Quartus II VHDL text editor. As such, no gate level diagram is available. The VHDL code is provided. The circuit symbol with all inputs and outputs of the circuit can be viewed in Fig. 2.

Simulation/Hardware Synthesis Procedure

The circuit was tested by uploading it to a Cyclone II EP2C20F484C7 as a component in the complete circuit g07_lab5_testbed. The controller successfully initialized the deck and dealt 2 cards to the dealer and the player. If the dealer did not win with 21, the player was allowed to request new cards until they either exceeded 21 or chose to stay by pressing the appropriate buttons. The dealer then played, and a winner was determined. LED lights connected to counters incremented by the d_won and p_won outputs allowed the user to keep track of the score. The player was able to review the review the game and then start a new round by pressing the hit key. The game was played multiple times to cover all possible situations that could be encountered, including dealer or player busting, a tie, dealer winning and player winning.

Results and Discussion

The circuit functioned correctly when tested on the board. In combination with the other components in g07_lab5_testbed, it enables the user to play a game of 21.

The advantage of this circuit is that it is able to reliably produce the correct signals needed to control all the data flow in the game, while remaining relatively simple in terms of design. An advantage of designing the controller as a FSM separate from the other components of the circuit is that it allows the whole system to be designed in a more modular way. Using a modular design allows for simpler testing of individual functions, and the circuit surrounding the FSM can be modified and improved without having to change the FSM, as long as the same control signals are used.

One limitation of this circuit is that it contains a large number of states, and therefore is quite costly in terms of components. The number of states in this circuit could likely be reduced by modifying the components this circuit interacts with or introducing new components. One particular example is the card dealing mechanism. It would be beneficial to have the card dealing be a single state in the controller fsm, rather than the two seperate states of loading a new card and then assigning the card to a hand. However, modifying the surrounding components or adding a new component will introduce additional complexity and circuitry to them, which may cancel out any simplifications or reductions achieved in the reduction of the states in the controller fsm.

g07_computer_fsm

Design Method

The g07_computer_FSM circuit decides what actions the dealer takes (hit or stay). If the sum of its hand is greater than 16, the dealer stays and the round ends. As long as the dealer's sum is less than or equal to 16, the dealer will hit.

The dealer waits until it's his turn (state WAIT), which is given by a dealer_turn signal from the g07_controller_FSM circuit (See Below for description). Once the turn signal is set to High, the dealer goes into the CP_TURN state, where it checks if the sum of its hand is greater than 16. If so, the dealer goes to the STAY state and if not it goes into the LOAD_CARD state.

The LOAD_CARD state waits for a new card to be drawn, and once it has, it will go into the DRAW state. In the DRAW state, the new card will be added to the sum of the old sum of the dealer's hand through the g07_rules module exterior to this circuit. Once the sum of the hand has changed, the FSM will move back to the CP_TURN to check if the new sum is greater than 16. The computer waits for the new sum and new card to change because the wait time is variable in the full g07_lab5_testbed circuit. The STAY state ends the turn of the dealer, and sets the dealer_done signal to High. The state transition diagram for the FSM is as described in Fig. 3. The reset bit resets the FSM back to WAIT_TURN when it is set to be High.

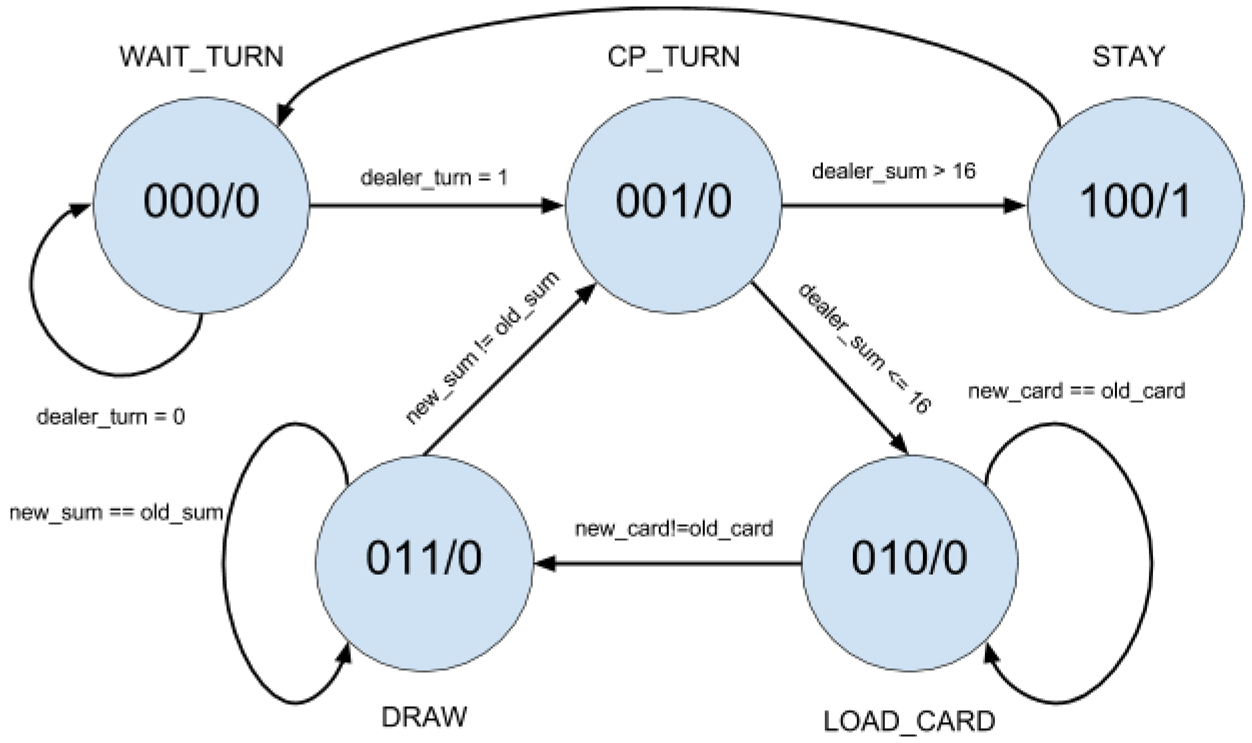

Fig. 3: State transition diagram for g07_computer_FSM

Schematic

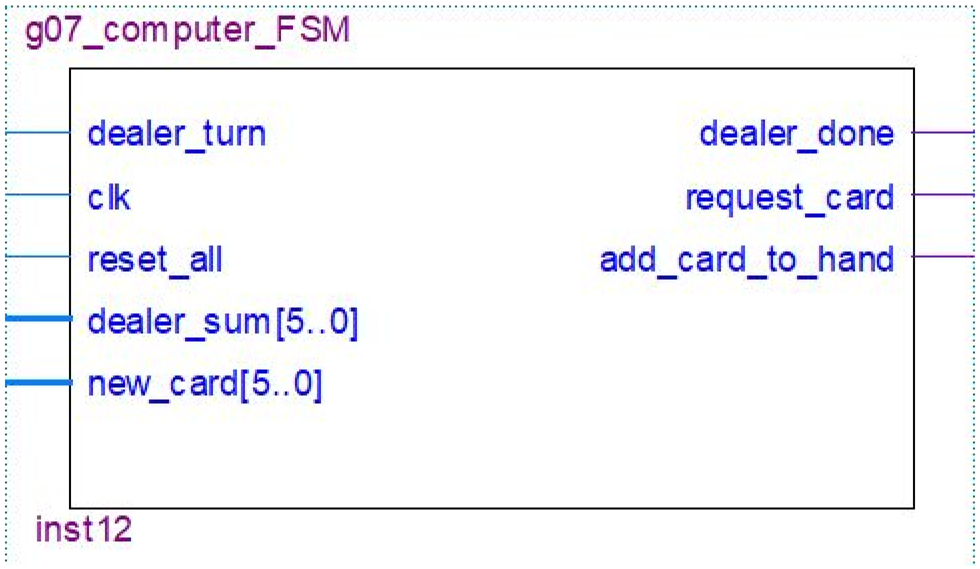

Fig 4: Circuit symbol for the g07_computer_FSM

The circuit was built by writing the code for the state machine in VHDL using the Quartus II text editor. As such, no gate level diagram is available. The VHDL code is provided in the submitted .zip folder. The circuit symbol with the inputs and outputs of the circuit can be viewed in Fig. 4.

Simulation/Hardware Synthesis Procedure

The circuit was tested with three separate timing simulations: one for when the dealer draws only once, one when the dealer draws twice and one to test the reset. Each test simulates what the g07_lab5_testbed would do when everything is connected properly as per the description above.

The first timing simulation in Fig.5 shows that the computer asserts the request_card signal as long as the new_card variable stays constant. Once it finally changes to a different card (in this case a Queen), the FSM transitions to the state of DRAW which asserts the add_card_to_hand signal which calculates the new sum of the hand. The FSM then asserts the dealer_done signal once the dealer detects that the sum changes.

The order of operations for this first test are as follows: dealer's hand has one card with a Jack (sum of 10), the dealer_turn is the asserted by what would be the g07_controller_FSM. The computer then requests a card to be loaded from the stack and then waits for the last loaded card to change to a new card. Once it does, the computer then waits for that card to be added to his hand and the new sum to be calculated. Once the sum changes, the dealer checks if he has a sum greater than 16, which he does in this case, so he then ends his turn.

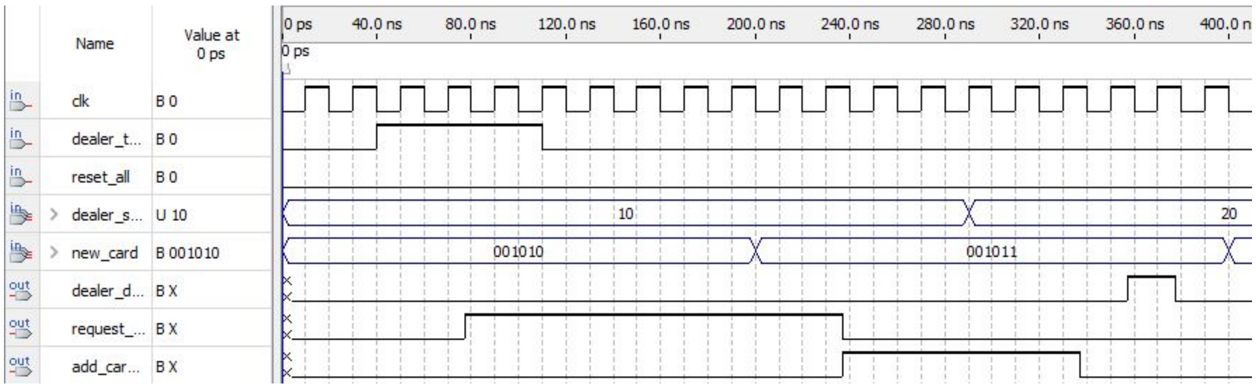

Fig. 5: Timing simulation of the g07_computer_FSM demonstrating when dealer does not bust

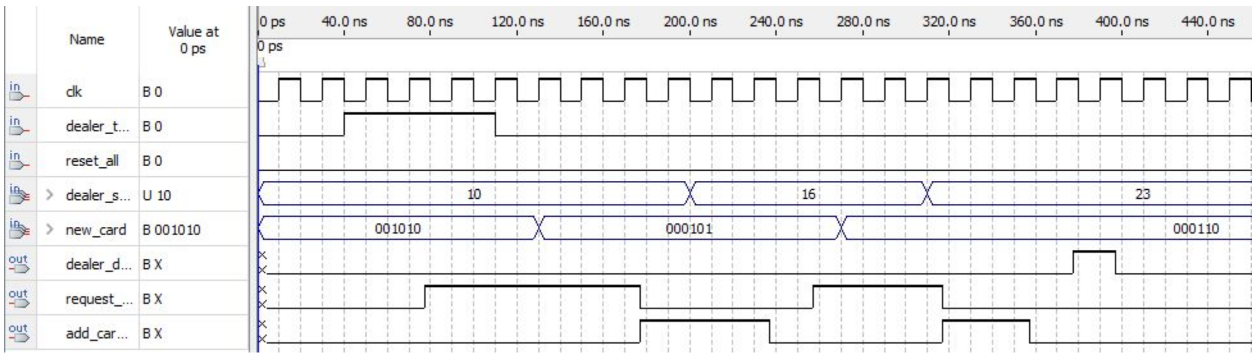

Fig. 6: Timing simulation of the g07_computer_FSM demonstrating when dealer does bust

The second timing simulation in Fig.6 demonstrates what the dealer does when the dealer goes bust. The simulation is not much different from the first test, aside from the fact that the dealer adds two extra cards to his hand since the sum of his hand is still less than or equal to 16 after the first card is added. This is to demonstrate that the computer does continuously draw cards if its hand's sum is less than or equal to 16. The sum in this test eventually reaches 23 since the computer draws a 7 with a hand sum of 16. The g07_controller_FSM would then see that the dealer is done and assert check the g07_rules module to see if the dealer has bust.

The last timing simulation shown in Fig. 7 was done to test whether or not the reset bit forces the computer FSM to return to its initial WAIT_TURN state. This uses the same timing simulation as the second test except the reset input bit is set to High before the dealer is done and another dealer_turn signal is set right after it. In the testbed, every circuit component is reset once the reset button is pressed, including the computer FSM. The controller would then deal out a new hand to the dealer that has a new hand. In this case the sum of that new hand is 16 with the last card being dealt a 6. The computer player FSM then operates as usual and asserts the dealer_done signal once the hand sum is greater than 16.

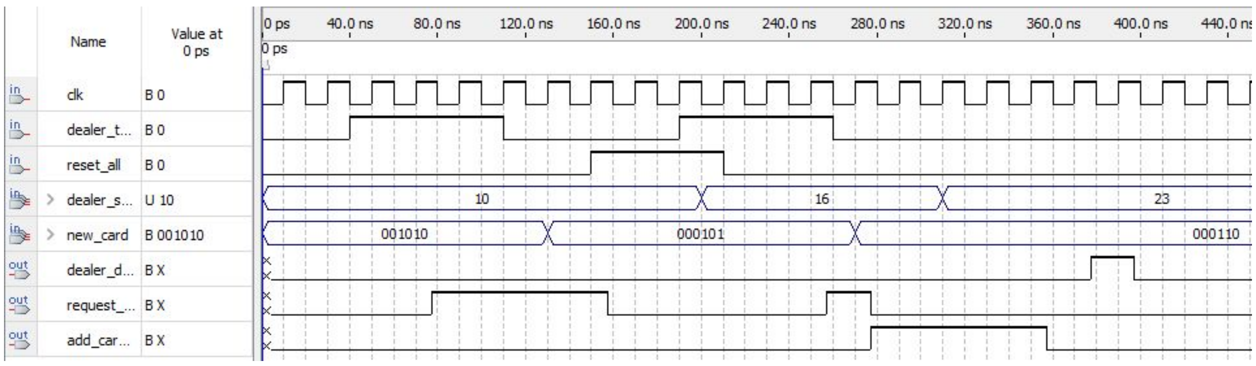

Fig. 7: Timing simulation of the g07_computer_FSM to test the reset input

Results and Discussion

The circuit send out the dealer_done signal once its hand sum is greater than 16 as it should. The computer also only acts once the dealer_turn signal is set to high and it also resets back to the WAIT_TURN state as it should.

One of the main advantages to this circuit is that most important information that it needs is calculated separate from this FSM. This makes it so that the controller can access other components of the dealer without having to go through the computer FSM to obtain that information. An example of this would be the dealer's hand sum, which is calculated and stored outside of the computer FSM circuit. If computer FSM did calculate its sum locally, the FSM would have many more states and a lot more inputs and outputs corresponding to request signals to obtain the dealer's sum, an output for the dealer's sum, etc.

A limitation that this circuit has is that it uses 3 flip flops rather than 2 because it has the extra state to load the new card. The first implementation of g07_computer_FSM contained only 4 states, where the loading of the new card and the loading of the new sum were both done at the same time, similarly to what the controller does with the player's hand. However, this caused hanging issues where during the testing of the g07_lab5_testbed board, the computer player would get stuck in between the DRAW state and the CP_TURN state. The fact that the drawing of a new card is separate into two sections causes greater delay in than what the previous implementation had, as now it waits for its new card in the LOAD_CARD state, and then in the DRAW state it waits for its new sum. If time allowed for it, a 4 state computer FSM could have been implemented to allow for only 2 flip flops in the overall circuit.

g07_lab5_testbed

Design Method

The testbed was divided into several sections each with a specific function. They are each discussed individually here.

Button Inputs

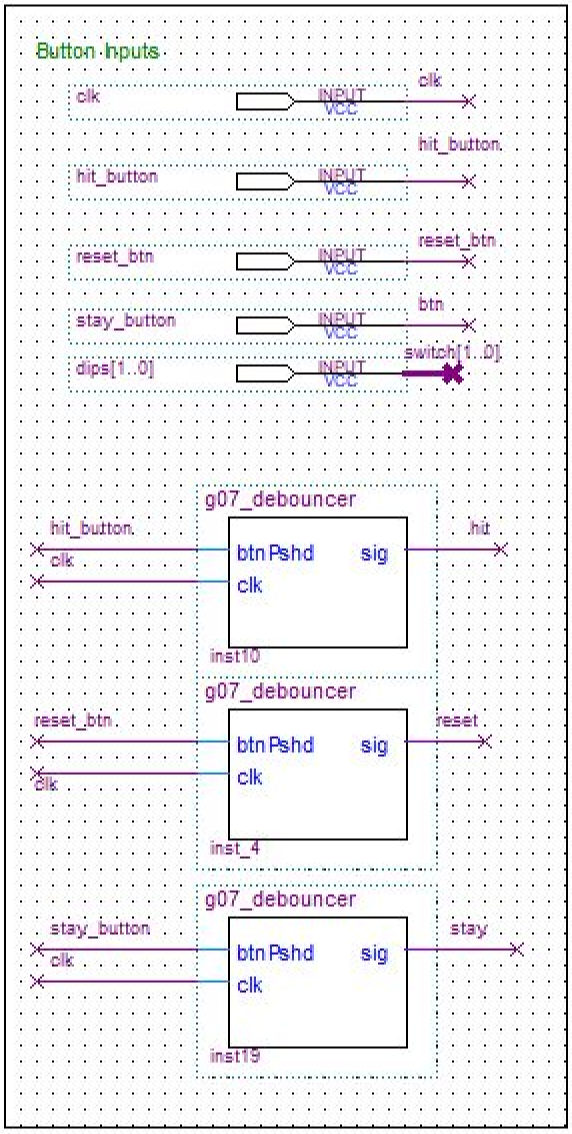

Fig 8: Button inputs sub circuit in the testbed

This subcircuit in Fig. 8 contains all the inputs to the circuit that the user can interact with, as well as the clock input. The three push buttons: hit, stay, and reset are connected to g07_debouncer units. The dips[1..0] bus is set by two dip switches on the board and allows the user to select the output they wish to see on the LED displays.

Controller FSM and Dealer Play FSM

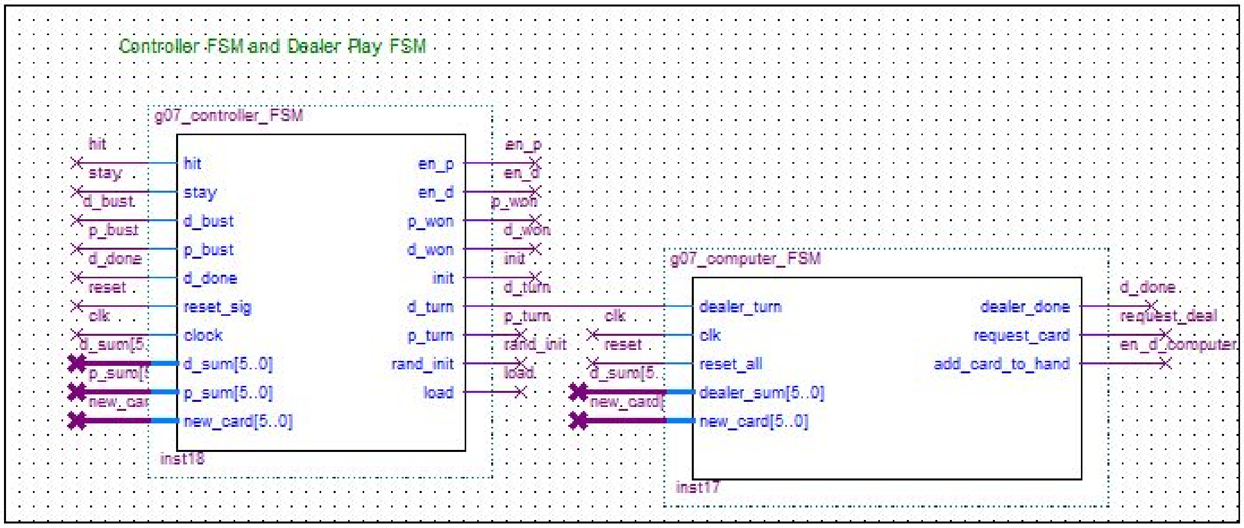

Fig. 9: g07_controller_FSM and g07_computer_FSM in the testbed

The subcircuit in Fig. 9 contains the two circuits that were designed in this lab. See the two sections above for details on how they work.

Random Number Generator

See section titled “Random Number Generator” in section g07_lab4_testbed in lab report 4.

Card Dealing

The card dealing subsection of the circuit in Fig. 10 handles the dealing of the cards out from the deck. The g07_dealer_FSM from Lab 4 (see Lab 4 report section “g07_dealer_FSM” for details) is enabled by either a request from the g07_computer_FSM to give the dealer a card or the g07_controller_FSM to deal the player a card.

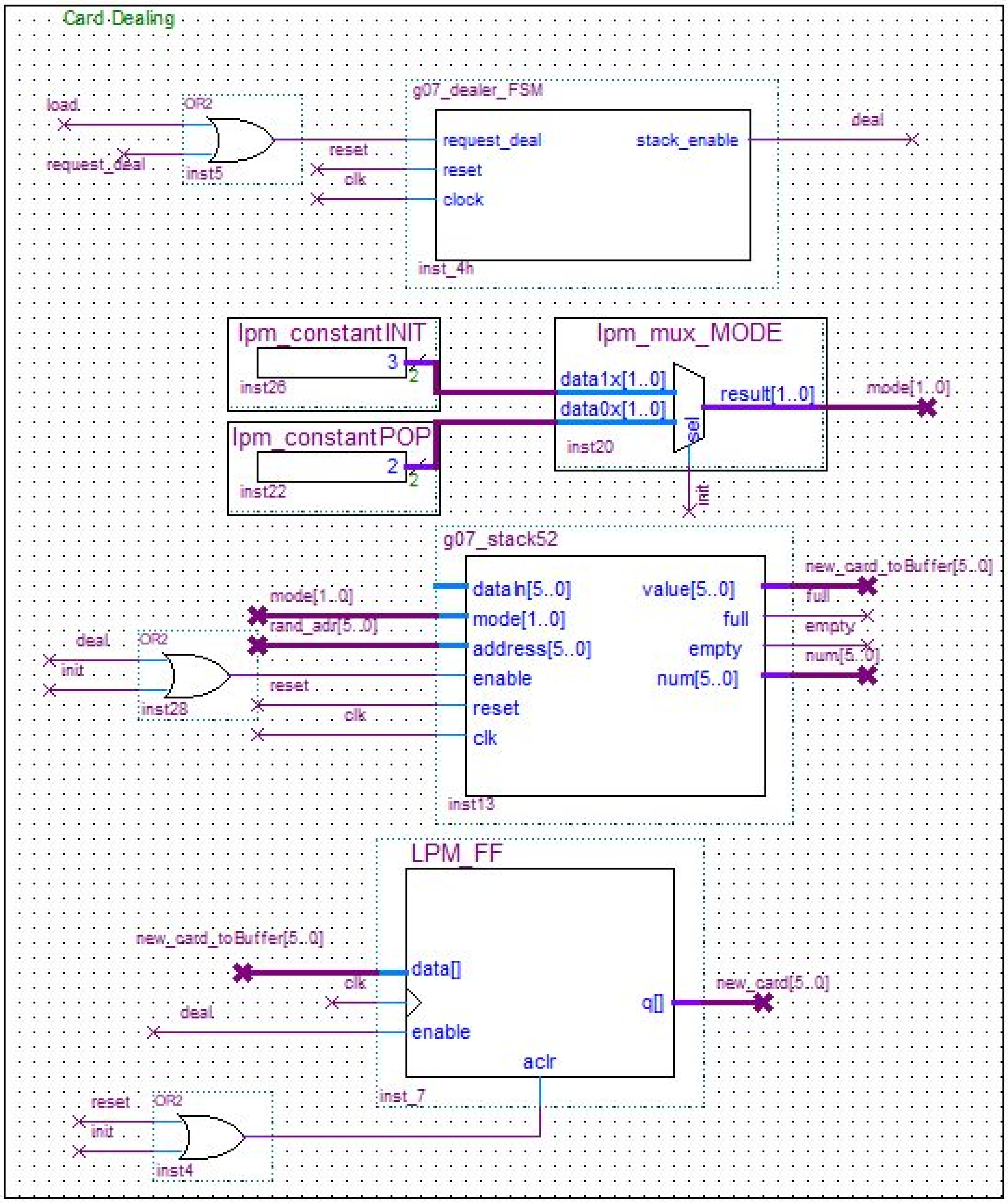

Fig. 10: Card dealing subcircuit of testbed

The g07_stack_52 circuit from Lab 3 (see Lab 3 report section “g07_stack_52” for details) represents the full card deck. An init signal is sent from the g07_controller_FSM which both enables the stack and and loads the lpm_constantINIT into the mode input. The combination of these two things initializes the stack at the beginning of each round and on reset. The enable of the stack is also controlled by the deal signal obtained from the g07_dealer_FSM, which always pops a single card off the stack from the random address rand_addr generated by g07_RANDU. The number of the cards on the stack is displayed on the board for testing purposes on one of the LED displays. The card that is popped off the stack is fed into an LPM_FF circuit which stores the newly popped card. The new card is then given as an output to be later handled by the card handler circuit and to be displayed on the LED display.

Hands and Card Adders

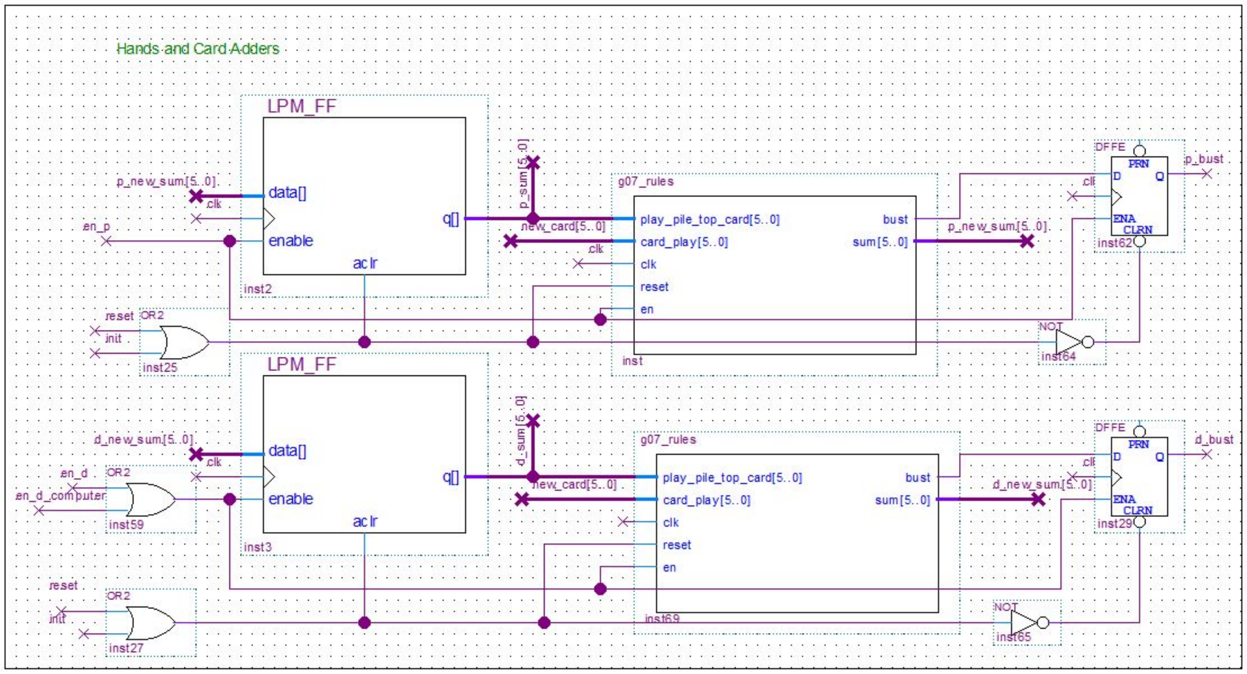

Fig. 11: Hands and Card Adders subcircuit of testbed

The subcircuit in Fig. 11 is responsible for keeping track of and updating the player’s and the dealer’s total sum. Each sum is stored in a 6 bit LPM FF.

To add a new card to the players hand, the en_p signal is asserted by the controller, which allows the g07_rules module to add a new card (card_play[5..0]) to the old sum of the player (play_pile_top_card[5..0]). After the addition of the new card, the new sum is then loaded back into the LPM FF. When the controller detects that the output of the flip flop has updated, it stops the assertion of the en_p signal. The rules module is also responsible for detecting if the player has busted and storing this information in a DFF. The FFs are cleared when the round has ended and a new round is started (init) or when the system is reset (reset).

The dealer's hand functions in the same way as the players hand, with the only difference being that the FFs and rules module for the dealer's hand can be activated either by the en_d signal from the controller FSM, or by the en_d_computer signal from the computer player FSM.

Scorekeeping

The scorekeeping section of the testbed in Fig. 12 keeps track of the amount of rounds the player and dealer have each won. It contains two lpm_counter circuits and some logic gates to decide how many wins the player or dealer have. The lpm_counters increment when their player or dealer wins a round. These enable signals come from the g07_controller_FSM once it decides who has won the round. The counters are reset when the reset button is pressed or after either the player or the dealer won 3 hands (winning the game).

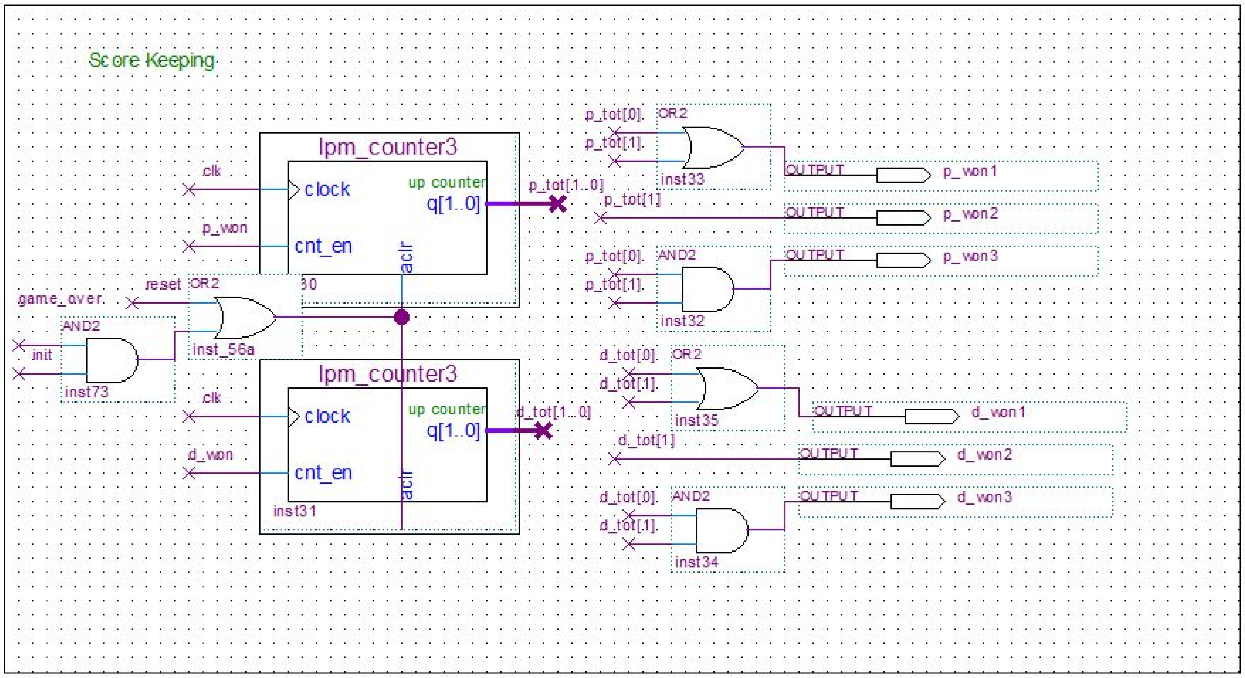

Fig. 12: Scorekeeping subcircuit of testbed

The output of the counters are given as input to a logic circuit that determines how many round wins the player/dealer has. The outputs of the logic gates are given as LED pins on the Altera board.

Determine winner and endgame

This subsection of the testbed in Fig. 13 determines whether the dealer or player has won three rounds (the game). When either of two reach three wins, the circuit goes into a waiting state where the user can check different values on the board. The user can continue to the next game by pressing the hit button (the end_game state in the controller_fsm). The p_win_game and d_win_game outputs are set to 1 when the respective player/dealer wins. The values for p_win_game , game_over and d_win_game are each held in a D flip flop until another game is initiated where they will all be reset back to zero since the lpm_counters from the previous section get reset.

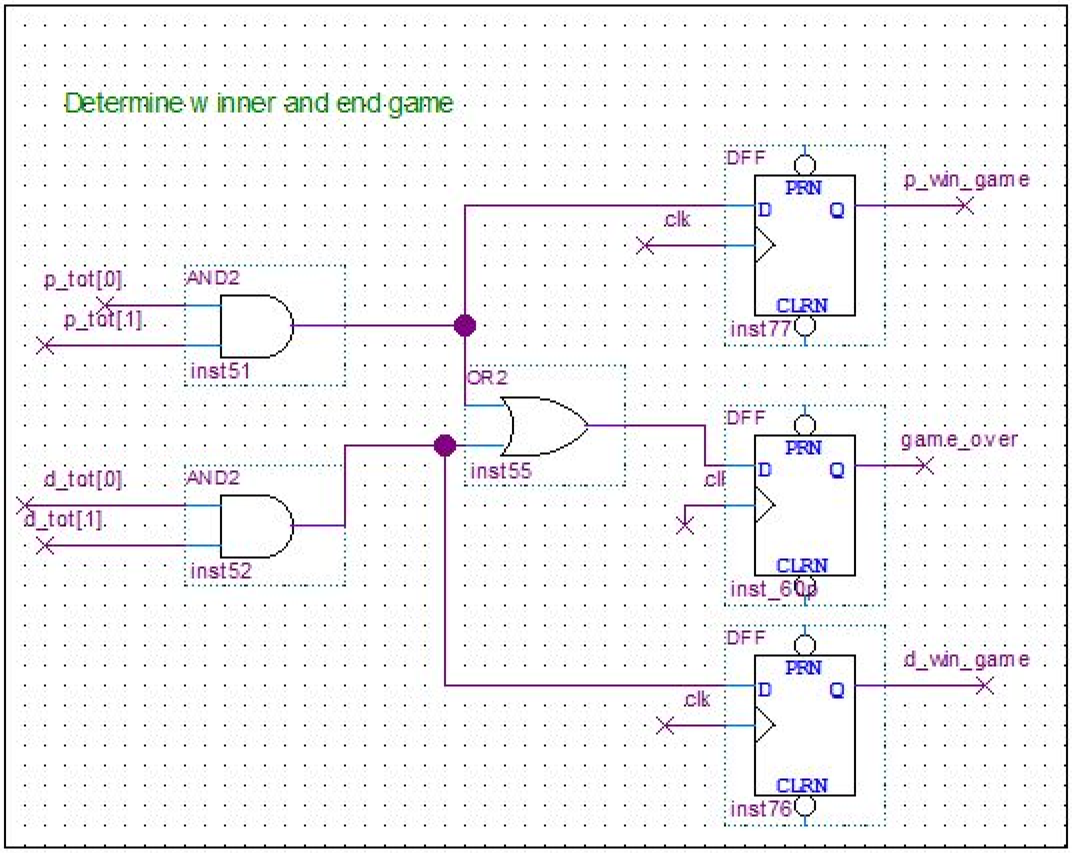

Fig. 13: Determine winner and endgame subcircuit

Decimal Number Display Number of Cards In Deck

See section titled “Number of Cards in Stack Display” in section g07_lab4_testbed in lab report 4. The output of the number of cards from this section is given as an output the LED display on the board.

Card Display

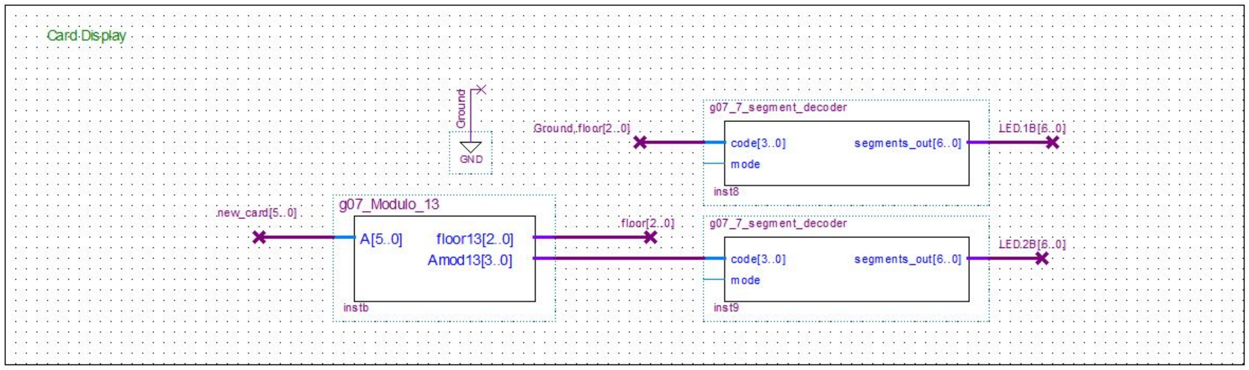

Fig. 14: Card Display subcircuit in testbed

This subcircuit in Fig. 14 takes the value stored in the LPM_FF connected to the stack. This value is the last card that has been dealt. The card value is decoded into its suit and face value using the g07_Modulo_13 circuit. The two outputs of g07_Modulo_13 are sent to g07_7_segment_decoder. The decoded values are then fed to the LED Displays subcircuit.

Decimal Number Display Hand Sums

The circuit in Fig. 15 for one of the modes of the display on the Altera board. The sum of the hands of both the dealer and the player are given as input to and lpm_divide circuit. The denominator of the lpm_divide circuits is an lpm_constant containing the number 10. The quotient then represents the 10s digit of the sum and the remainder the 1s digit.

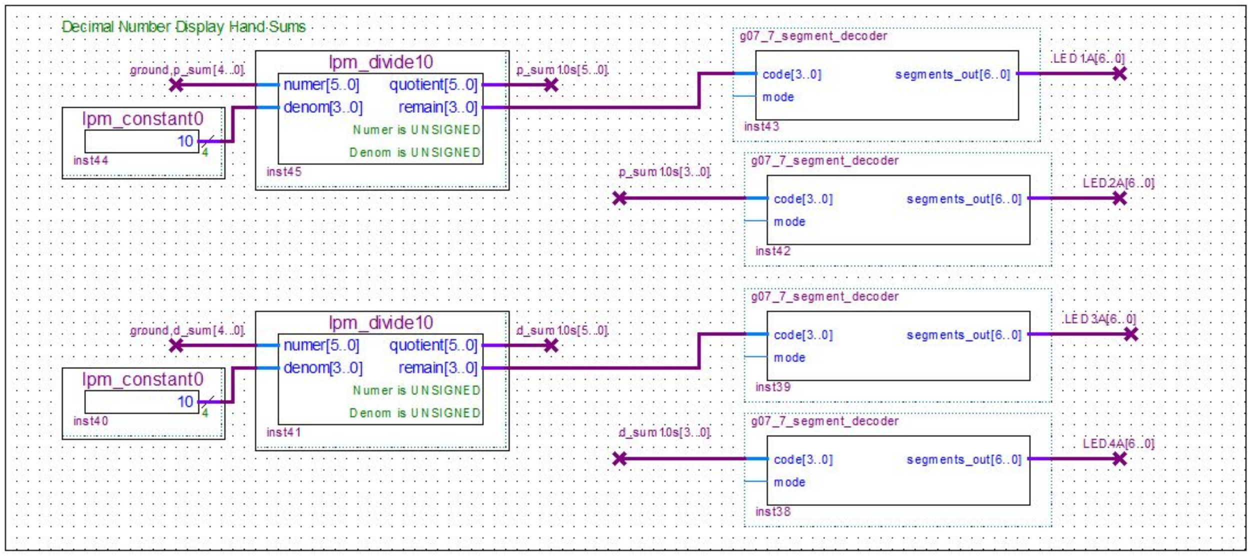

Fig 15: Decimal Number Display Hand Sums subcircuit

The quotient and remainder of each of the lpm_divide circuits are fed into g07_7_segment_decoder circuits with mode=0. The output of these decoders then give the output to the Altera board to display the sums of both hands.

<u>Games Played, Games won by Player/Dealer</u>

The Games Played circuit in Fig. 16 is responsible for incrementing and displaying the amount of games the player or dealer has won, as well as the amount of games played in total. The lpm_counter circuits are only enabled when a dealer or player has won 3 rounds, and the user continues to the next game by pressing the hit button. The lpm_counter circuits have a limit of 9. The output of the counters are each given as input to their own g07_7_segment_decoder circuit to be displayed on the board.

The outputs of the counters are also added by an lpm_add_sub circuit to obtain the total amount of games played. The total number of games player are displayed through the same method used in the previous section “Decimal Number Display Hand Sums” for the sums of the hands.

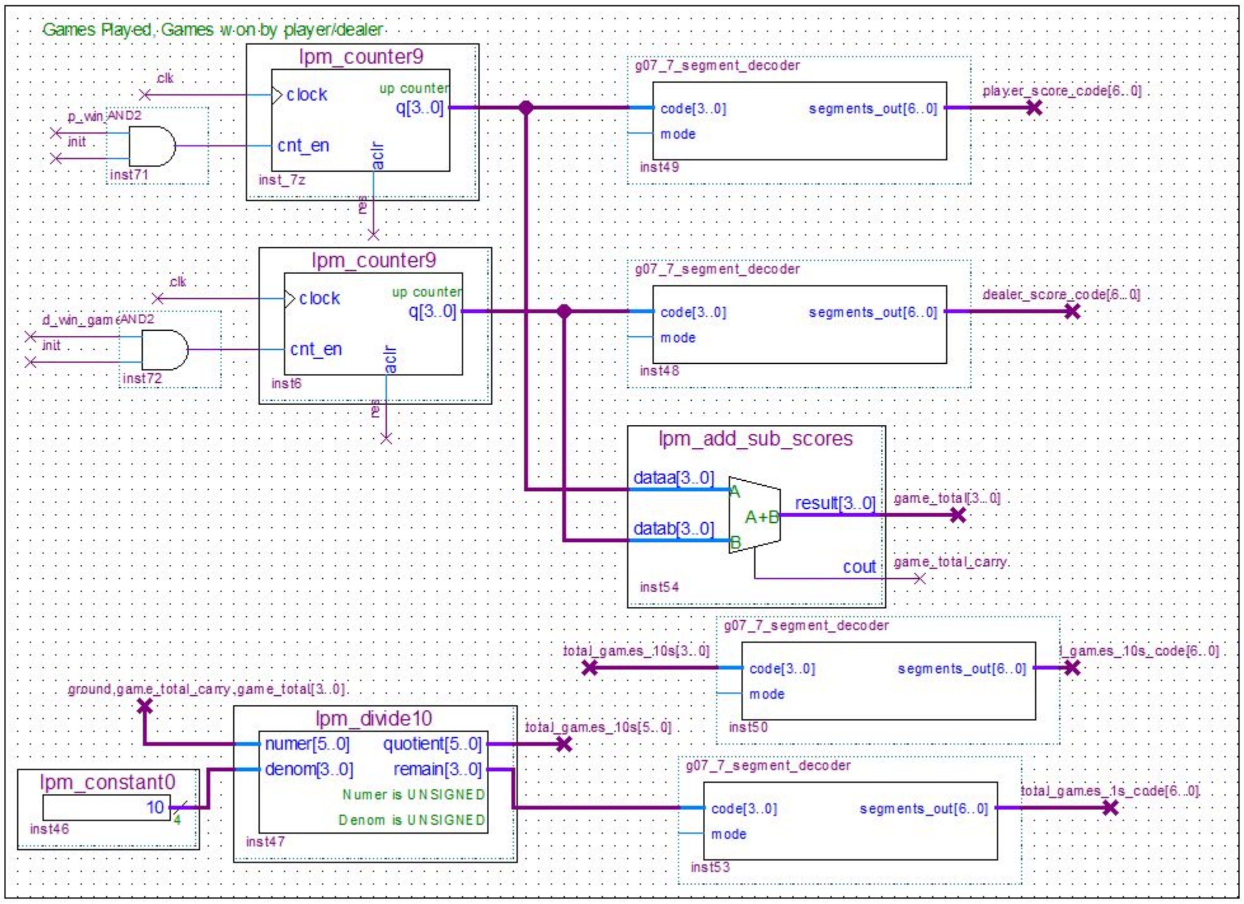

Fig. 16: Games Played, Games won by Player/Dealer subcircuit

LED Displays

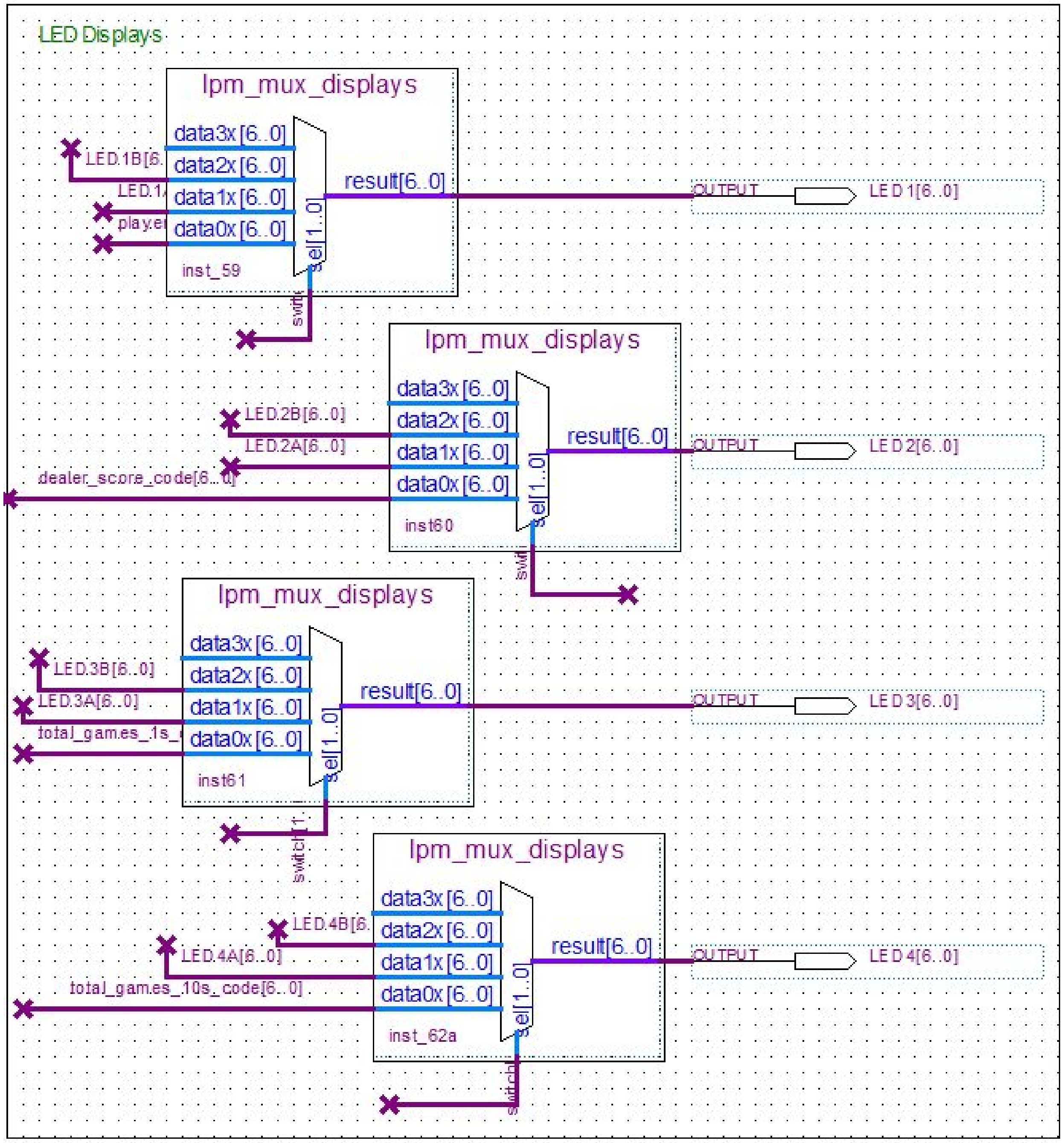

Fig. 17: The LED display subcircuit schematic

Thesection of the testbed in Fig. 17 is responsible for deciding which LED modes to display on the Altera board. The LED displays on the board can be one of three modes. See Table 3 for the different modes the display can have. The modes of the display are controlled by two dip switches on the board given in section “Button Inputs” above .

	LED4	LED3	LED2	LED1
A(00)	Total Games(10s)	Total Games (1s)	Dealer Game Wins	Player Game Wins
B(01)	Dealer Hand Sum (10s)	Dealer Hand Sum (1s)	Player Hand Sum (10s)	Player Hand Sum (1s)
C(10)	Number of cards in deck (10s)	Number of cards in deck (1s)	Newest Card (Suit)	Newest Card (Value)
D(11)	Nothing	Nothing	Nothing	Nothing

Table 3: The modes of the LED displays on the board

Additional Gameplay UI

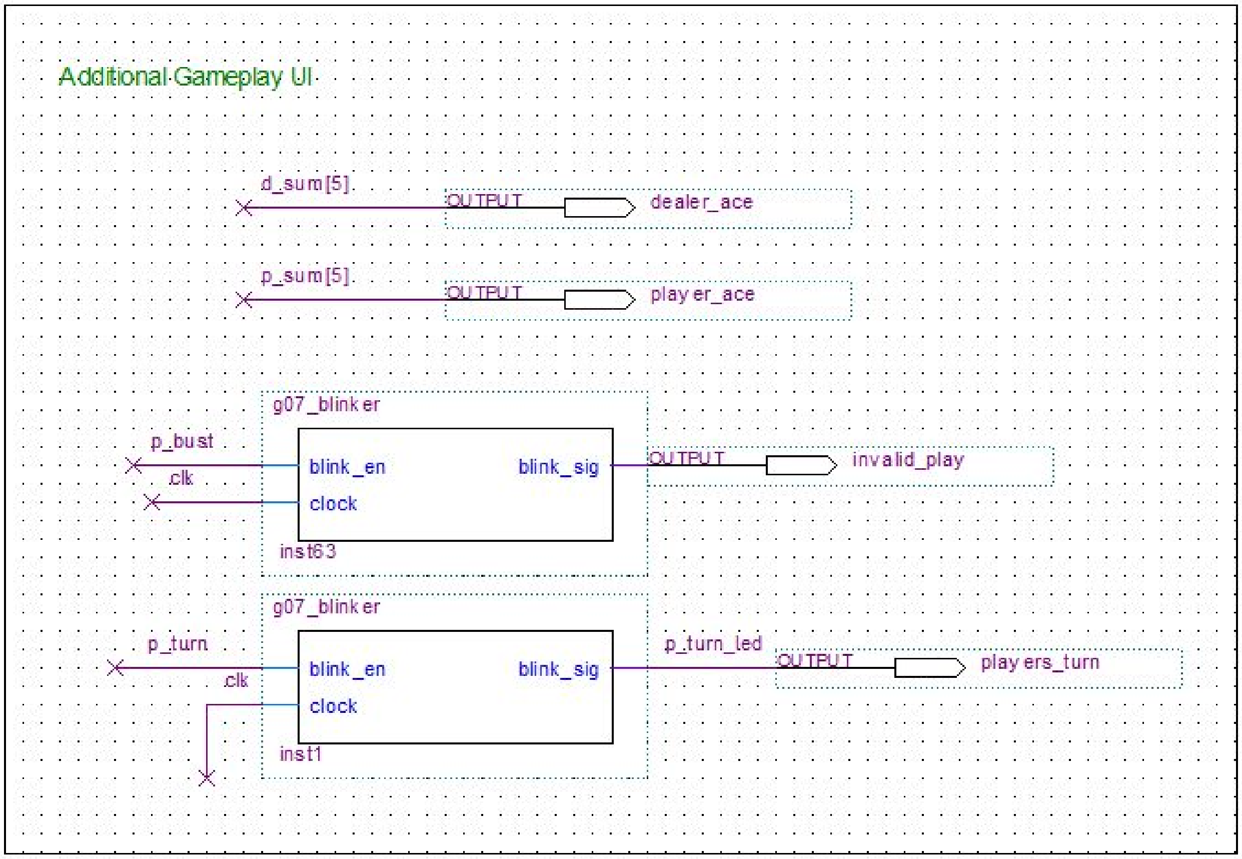

Fig. 18: Additional Gameplay UI subcircuit of the testbed

The Additional Gameplay UI section shown in Fig. 18 checks if the dealer and player have an ace that is being counted as 11 in their hand and outputs this status to an LED light. The bit representing this information is encoded in the 6th bit of the flip flop holding their hand (see Hands and Adders section).

This subcircuit also check if the player has busted. If the player has bust, a red LED light on the board is made to blink using the g07_blinker circuit. A blinking green LED light is also used to indicate to the player when it is their turn.

Schematic

See above schematics.

Simulation/Hardware Synthesis Procedure

The g07_lab5_testbed was tested on the Cyclone II EP2C20F484C7 FPGA board. Games were played several times with the dealer to test the consistency of the entire circuit. See Table 4 for the pin assignments, their descriptions, and their assigned signals.

The different components of the player's turn was tested simultaneously, whether it was the detection of an invalid play (player busted) or whether the hit and stay options performed their proper functions at all times. Both the player and dealer's hand sums were displayed on the board. The displaying of the most recent card dealt was used to determine if the circuit was summing the player's cards properly. The LED designating the signal of a high ace in the player's hand was also useful for testing proper summation. The displaying of the number of cards on stack was then used to test that hit would only pop one card.

The computer player was tested similarly to the player. The displaying of the last card dealt helped in deciding if the summation of cards was working correctly for the dealer and whether or not the dealer should have hit or stay with his previous sum before hitting his last card.

Endgame was tested indirectly through the several tests performed earlier throughout. The number of games won by player and dealer were both displayed on the board, as well as the amount of total games. The rounds won were also displayed through the use of LEDs (green for player and red for dealer) so that we could test to see if the controller was properly choosing the winner of a round.

The reset button was tested at several different sections of testing, whether it was immediately after startup or after many games were played.

These tests were each performed several times simply by playing the game on the board and observing if any errors occur throughout.

Pin Assignments

Altera Board Pins	Assigned Signal	Description
Green LEDs		
LEDG0	p_won1 (Scorekeeping)	Player won 1 round
LEDG1	p_won2 (Scorekeeping)	Player won 2 rounds
LEDG2	p_won3 (Scorekeeping)	Player won 3 rounds and the game
LEDG3	N/A	N/A
LEDG4	N/A	N/A
LEDG5	N/A	N/A
LEDG6	player_ace (Additional Gameplay UI)	Turned on when the player had an ace counted as 11 in hand
LEDG7	players_turn (Additional Gameplay UI)	Blinked to indicate it was the player’s urn
Red LEDS		
LEDR0	d_won1 (Scorekeeping)	Dealer won 1 round
LEDR1	d_won2 (Scorekeeping)	Dealer won 2 rounds
LEDR2	d_won3 (Scorekeeping)	Dealer won 3 rounds and the game
LEDR3	N/A	N/A
LEDR4	N/A	N/A
LEDR5	N/A	N/A
LEDR6	N/A	N/A
LEDR7	N/A	N/A
LEDR8	dealer_ace (Additional Gameplay UI)	Turned on when the dealer has an ace counted as 11 in hand
LEDR9	Invalid_play (Additional Gameplay UI)	Blinks when the player busts

LED Displays		
HEX0	LED1[6..0] (LED Displays)	Displays as per Table 3
HEX1	LED2[6..0] (LED Displays)	Displays as per Table 3
HEX2	LED3[6..0] (LED Displays)	Displays as per Table 3
HEX3	LED4[6..0] (LED Displays)	Displays as per Table 3
Push Buttons		
KEY0	hit_button (Button Inputs)	Used by the user to add a card to his/her hand
KEY1	stay_button (Button Inputs)	Used by the user to end his/her turn
KEY2	N/A	N/A
KEY3	reset_btn (Button Inputs)	Resets the entire board to Round 1 of Game 1
Switches		
SW0	dips[1]	Controls the displays as per Table 3
SW1	dips[0]	Controls the displays as per Table 3

Table 4: The pin assignments for testing the g07_lab5_testbed circuit. No switches other than SW0 and SW1 were used

Total logic elements	1,257 / 18,752 (7 %)
Total combinational functions	1,198 / 18,752 (6 %)
Dedicated logic registers	632 / 18,752 (3 %)
Total registers	632
Total pins	44 / 315 (14 %)
Total virtual pins	0
Total memory bits	3,328 / 239,616 (1 %)

Table 5: Flow summary of g07_lab5_testbed

Results and Discussion

Designing the system as smaller interconnected modules allows each module to be built and tested debugged separately from the rest of the system, which is a much simpler task than trying to test and debug a single large circuit. Designing the building blocks in previous labs allowed for those circuits to be tested separately to be eventually put into the overall testbed in this lab.

The advantage to this circuit is that it creates an equally fair game for the user and the dealer, as the game is always random because of the Random Number Generator section of the circuit. This way, the game cannot be abused in any way by the player. The feedback provided to the user also provides the game with some strategy for the player. Knowing that the must hit if they have less than 17 allows the player to make decisions based on whether or not he/she has an ace or what the last card dealt was. For example, if the player needs a 6 to win, and the last card dealt was a 6, the chances of getting a 6 is less than if there were 4 6’s in the deck.

One of the main disadvantages to this circuit is that it is relatively slow. It takes several clock cycles for most operations to complete. For example, popping a card off the stack takes at least 5 clock cycles to change the outputted hand sum. This is mainly due to the fact that the new card of the stack is buffered and stored for testing and displaying purposes, as well as delay from POP_EN and the flip flops inside the stack. Not buffering the new card would remove a clock cycle, but would also remove last card display for the user when playing the game. The maximum delay comes from this fact, and if the maximum delay were to be greater than 20ns, the user feedback of the new card would have had to been removed.

The longest delay propagation delay is 12.371ns which is from the input dip switches to LED4. This is less than our clock period of 20 ns, so it is acceptable.

References

[1] "21 Card Game | Planning With Kids", Planning With Kids, 2017. [Online]. Available: https://planningwithkids.com/2012/06/20/21-card-game/. [Accessed: 05- Dec- 2017].

[2] J. Clark, Lab #5 – System Integration for the Card Game. Montreal: McGill, 2017.