**Department of Engineering**

**McGill University**

**ECSE 323**

**DIGITAL SYSTEM DESIGN**

**Project - 21**

**2017-12-07**

## TABLE OF CONTENTS

## INTRODUCTION

The game ***"21",*** is a comparing card game between usually several players and a dealer, where each player in turn competes against the dealer, but players do not play against each other. In this project, there is only one player playing against an automated dealer.

In this game, there is a stack 52 that represents the standard deck of 52 cards from which 2 cards are dealt to each player, i.e. the human player and the computer player. However, the dealer has one card face down as opposed to both being faced up like the player's. In this project, 2 cards are handed out to the player and 1 card to the dealer at the beginning of each round. The dealer's second card is added to the dealer's first card at the beginning of the dealer's turn. The game is won by the player who has the closest value of 21 as long as the player does not go over 21. Every *game* consists of 5 *rounds* (an *invalid play* does not count as a round). The rules which dictate the game as seen in [1, pp. 7-8] are as follows:

- *Aim of the 21 Card Game is to get 21 or as close to as possible.*
- *Number cards have their face value, jacks, kings and queens are worth 10. Ace can be either 1 or 11 and the player who holds the ace gets to choose the value of the card.*
- *The dealer and all other players have two cards. With the exception of the dealer the players have their cards face up. The dealer has one card up and one card face down.*
- *The dealer goes to each player one at a time (in this case, there will be one player). The player needs to decide if they want another card (hit) or will sit on what they have. You can have as many cards as you like as long as you don't go over 21.*
- *The dealer does this with every player. Players are not competing against each other, but against the dealer.*
- *The dealer then turns over their other card and needs to decide what to do. If the dealer has 16 or under then they must take another card.*
- *If the dealer has 21 (Ace and a ten value card) the dealer wins.*
- *If the dealer goes bust then everyone else wins.*
- *We reshuffle the deck of cards after every game.*

If both players have the same value of cards, then it is a draw, hence an *invalid play*. However, if they both have a value of '21', then the dealer wins.

## SYSTEM DESCRIPTION AND FEATURES

The system that has been implemented has the following features:

- It can reset the game at any time, i.e. all values become zero and the deck initialized.
- The game has to reset before starting the first game.
- The system can deal out random cards from the deck at initialization and whenever any of the players *hit.*
- The system displays the sum of cards of every player at the beginning of each roundas well as the card that has been 'popped' from the stack 52.
- During the human player's turn, this system displays the sum of cards the human has as well as the dealer's card that is faced up. The human can decide whether to *hit* or to *Stop.* The system then displays the card that the human player gets in case the human decides to hit.
- The system starts the round with the human player, then switches to the dealer's turn depending on whether the human player goes bust or decides to *Stop.* If the human player loses, then the system restarts the round.
- The system waits for a user input at the end of each round in order to start a new one.
- The system has the ability to acknowledge whenever a game or a round is over, and whether there is a winner or a draw.
- The system reshuffles after every game of five legal rounds, i.e. the total of wins of either the human player or dealer is equal to five.

## USER INTERFACE

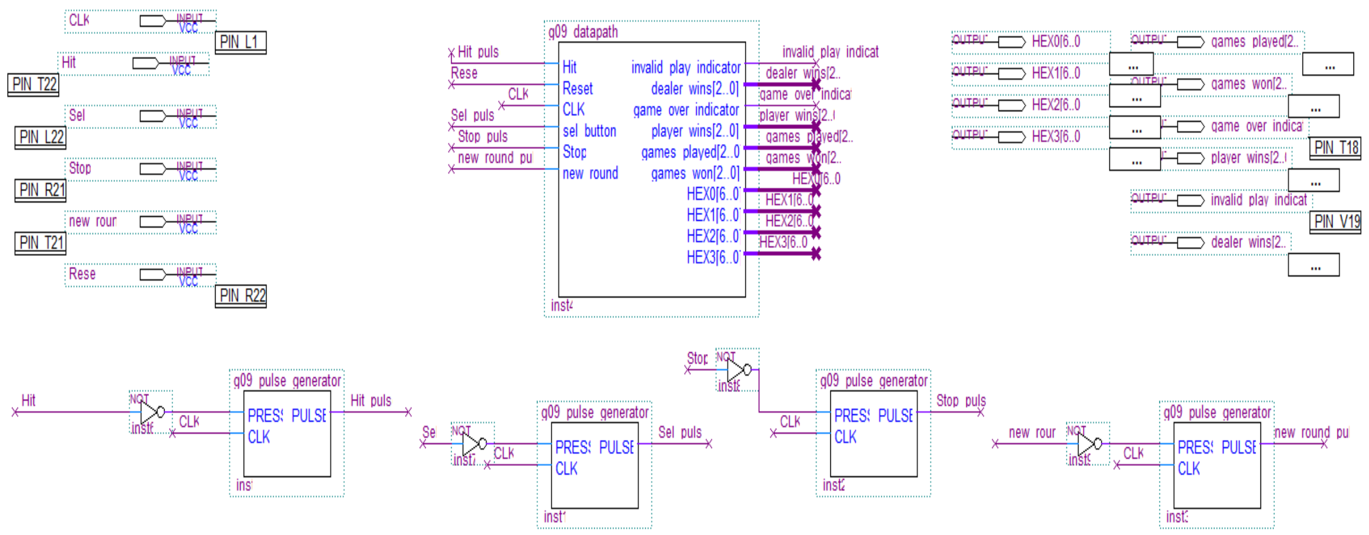


**Figure 1 - Datapth Symbol**

User Inputs:

- Reset: Pushbutton KEY0
- Hit: Pushbutton KEY1
- Stop: Pushbutton KEY2
- New round: Pushbutton KEY3
- Select Ace=11: Toggle Switch SW[0]

Other inputs (not user): CLK

Displayed Outputs:

- Sum of player card values: HEX 2,3
- Sum of dealer card values: HEX 0,1
- Card: HEX 0,1 (the use of the same 7-seg displayed is explained in the design method part)
- Player wins: LEDG[5..7]
- Dealer wins: LEDR[0..2]
- Invalid Play Indicator: LEDR[5]
- Game Over Indicator: LEDR[4]
- Games Won: LEDG[2..0]
- Games Played: LEDR[7..9]

## DESIGN METHOD

A good way to design a system is to separate control modules from the datapath modules. In our design, we use datapath/controller architecture. The datapath modules are for the purpose of processing and manipulating data while the control modules are for generating control signals to direct the operation of the datapath modules. The datapath is also supposed to send back status signals. Figure #2, from [2, pp. 4], shows how datapath/controller architecture is connected clearly.

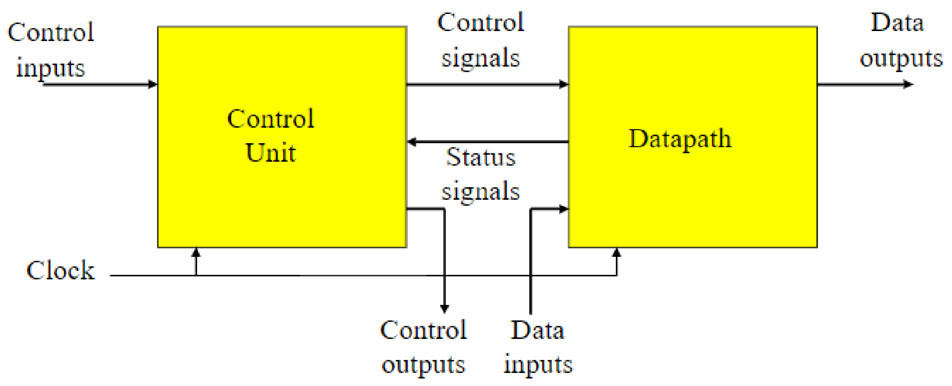


**Figure 2 - Datapath/Controller Architecture**

In this report, the explanation of the design is going to be split into the explanation of control modules then datapath modules. Figure #3 shows the full system controller; the control modules of the system.

## Full System Controller

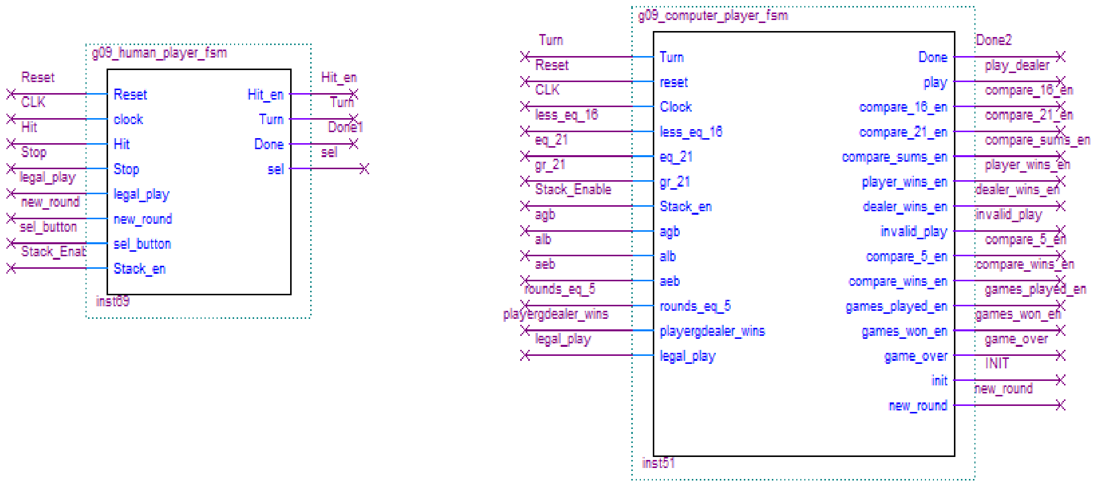


**Figure 3 - Control Modules**

The full system controller is made up of two control modules, the g09_computer_player_fsm and the g09_human_player_fsm. These two modules run concurrently and keep interacting with each other.

To design the control modules, we designed its state diagram first. While the design of this state diagram, it was advised to use as many states as needed and that was our approach to it as well. It is known that Quartus software will minimize redundant states automatically. Therefore, it is a better idea to keep it simple and clear for us and make the software do the optimization and minimization. This approach is time-saving and less prone to error. We will start by explaining the g09_human_player_fsm (referred to as HC from now on) state diagram since that is where the game starts when the human's (player) turn. An explanation of the state diagram of the g09_computer_player_fsm (referred to as CC from now on) will follow. As for the notation of the state diagrams, each circle in the diagram represent a state with the outputs asserted mentioned inside the circle. A list of zeros in the circle (00...00) or an empty circle means that none of the outputs are asserted. The input values on top of the arrows between the states represent the needed inputs to go between states in the direction specified.

## Human Controller

A copy of the state diagram of the HC is found in Figure #4. When the clock starts (system turned on), the HC is at the initial state A. It will wait for Reset to be '0' then '1' in order to be able to go to state B then X in two clock cycles. The reason for waiting for Reset to be '0' then '1' is to detect its rising edge. Moving on, both of states B and X are states that do not assert any outputs. At the next rising edge clock cycle, state C is reached where the process of giving the 2 initial cards for the player as well as the initial card for the dealer starts. The reason for giving one card for the dealer at this time as opposed to 2 has been stated before. The second card will be given to the dealer at the beginning of his turn (during CC states). So, at states C,D, & E, Hit_en = '1' to give the datapath the signal to give the cards to the dealer and the player depending on the value of Done. At state C, Done = '1' meaning the card will be given to the dealer while at state D & E, Done = '0' which means the cards will be given to the player. The states X,Y & Z that sit between B,C,D, & E are put for making Hit_en = '0' in between which is needed for the g09_computer_dealer_fsm designed in Lab 4; transitions on input signals were done on a 2-stage manner. After giving the initial cards, it becomes the player's decision to either Hit or Stop and if there's an Ace card then choose it to be either 1 or 11. If he hits (Hit = '1'), HC goes to state G from F. If the sum of his cards goes above 21 (legal_play = '0'), HC will go to the next state J, where it waits for a new_round button press. Legal_play is an output of HC and an input for the datapath to count that the dealer won this round. It is also an input for the CC where control signals are sent to check if this is the fifth round of the game or not and therefore count as needed (more details in CC explanation). Same logic is used if the player decides to Set the Ace card value to 11 or 1 (initial value). Now, if the player decides to stop, HC will go to state I, where Turn is asserted meaning that CC will move from state B to state C and it will be the dealer's turn until the end of the round. Meanwhile, HC will be at state J after I waiting for a new_round button press. Keep in mind that the dealer plays in clock cycles speed so for the user can't press the new_round button before the dealer finishing his turn.

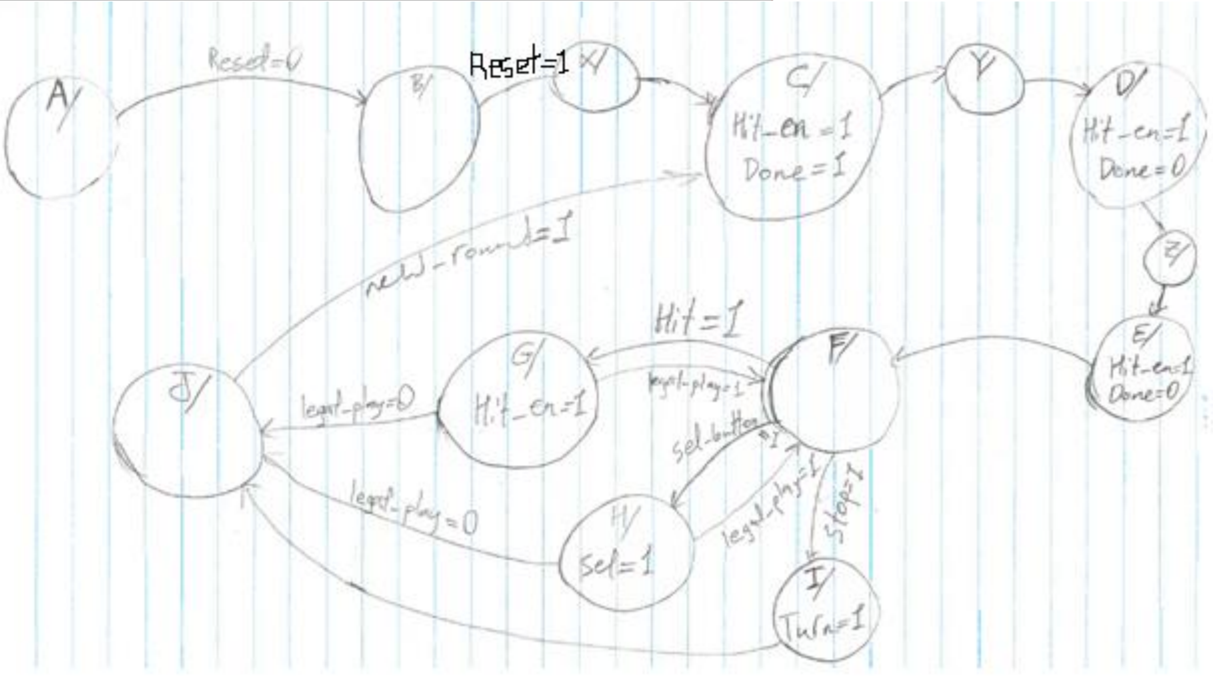


**Figure 4 - HC State Diagram**

## Computer Controller

The state diagram for the CC is displayed in Figure #5. When starting a new game (Reset = ‘1’), CC starts at the first state A. Turn is also initially ‘0’ until there’s a change of turn between player and dealer. That means that the CC will go to the next state B at the next clock cycle. After that, when Turn is asserted from the HC, the dealer’s turn starts (Done = ‘1’) by moving to state C in the CC where the dealer gets the second card that wasn’t given at the beginning of the game by asserting Play = ‘1’ (another way of showing one card of the dealer and hiding the other). After receiving that card (Stack_en = ‘1’), the next state will enable the comparison of the sum of the dealer’s cards and 16. If less or equal, dealer should get another card meaning go back to state C. If equal to 21 then enable the counter of dealer_wins to add one to the dealer_wins. If more than 21 then enable the counter player_wins to add one to the player_wins. However, if between 17 and 20, then enable the comparison between sum of the dealer’s cards and the player’s cards and determine the winner. If they are equal, then assert the invalid_play_indicator which is at state I. Other than an invalid round, at the end of each round, it should be checked if it is the fifth round and determine the winner of the game (5 rounds). These steps of determining the winner of a full game is done at states J,K,L, and M. At the case of states I and M, the invalid_play_indicator or the game_over_indicator will be displayed on the Altera board until a new_round_button is pressed. This is done using D flip_flops which will be explained in details at the explanation of the datapath modules.

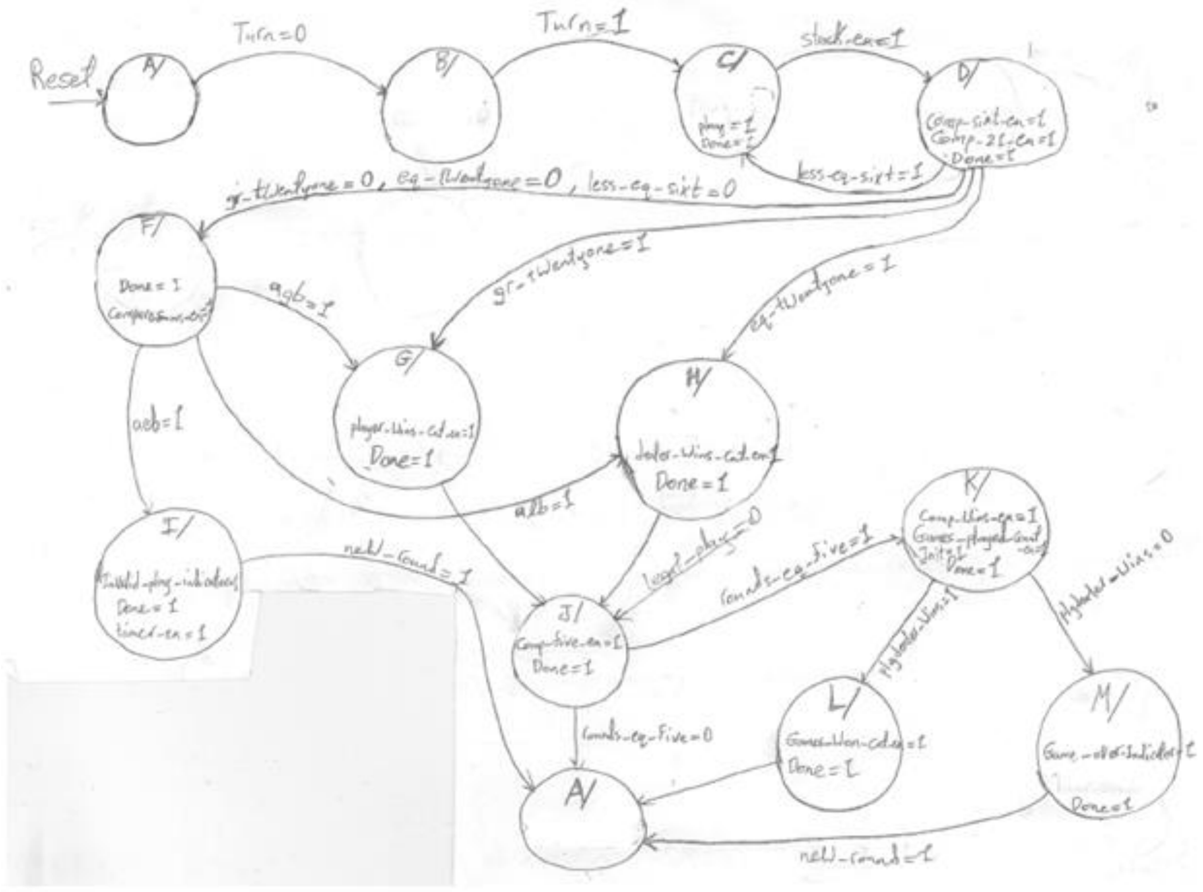


Figure 5 - CC State Diagram

## Full System Datapath

For the datapath modules, screenshots of parts of it will be explained individually. The parts are counters, comparators, rules modules (player and dealer) and the dealer testbed (that chooses the random cards of the deck).

### Counters

Starting with the counters, Figure #6 includes a screenshot of the counters found in this system. There is a total of 4 counters in the system and one adder which adds the rounds won by player and rounds won by dealer (incrementing by 1 each round). These counters are all controlled/enabled by control signals that are outputted from the CC or HC and occasionally by legal_play output of the player. So, to enable the counter of dealer_wins, one of the following options should happen: 1) either the players go bust (sum_player>21 and therefore legal_play = 0) or 2) the dealer_wins_en is asserted in CC after determining that the dealer won either by having a sum of 21 or dealer has a bigger sum than player. A multiplexer was added at the input of the enable of the counter to select the right option above at the right time. The select input of this multiplexer is an OR gate of Done1 and Done2 which are the control signals of Done from CC and HC. This counter will clear/reset its value at the end of each game or at the press or Reset (starting a new game). For the counters of player_wins, games_won and games_played, enabling them is dependent on the outputs of the CC which are players_wins_en, games_won_en, and games_played_en. These counters will also be cleared at the end of each game or when starting a new game. The adder that will output the number of rounds played will always be enabled since it's an addition of player_wins and dealer_wins and both get cleared at the end of each game. Therefore, there is no problem in having it always enabled.

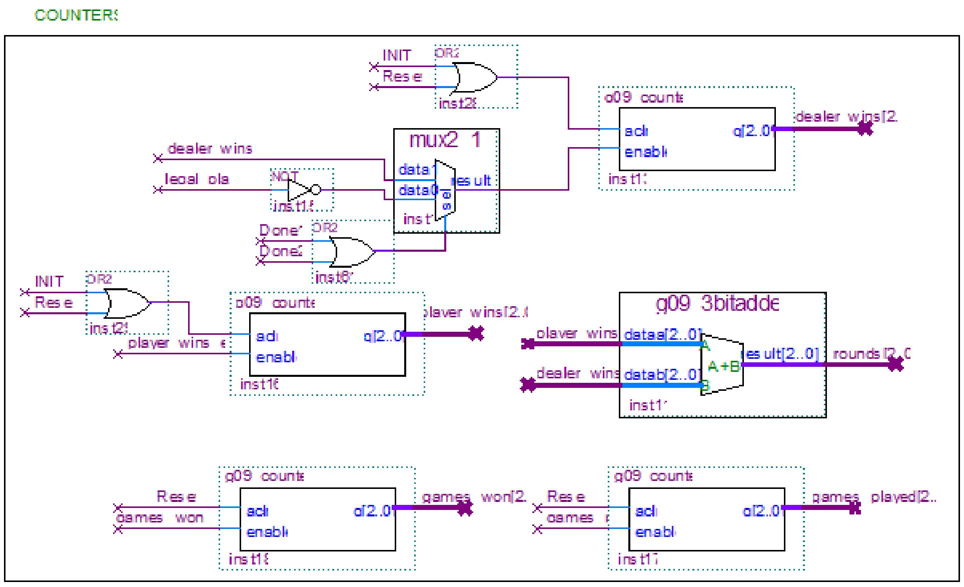


**Figure 6 – Counters**

## Comparators

A screenshot of the comparators used in this system is displayed in Figure #7. There is a total of 5 comparators in the system. All of them are being enabled by control signals coming from CC. These signals are comp_16_en, comp_21_en, comp_5_en, comp_sums_en, and comp_wins_en. All these signals are asserted at the right time during the CC and that is shown clearly in the state diagram in Figure #8 and its explanation.

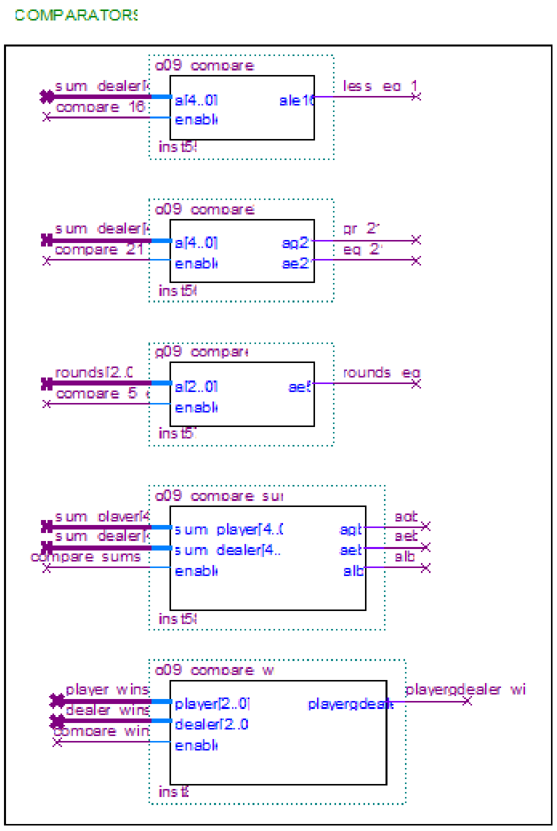


**Figure 7 – Comparators**

## 7-Segment Display

As stated in the user interface part, we decided to interchange the first two 7-segments displays between the sum of the dealer cards and the card (value and suit) when the player decides to hit. So, at the beginning of the game, before the player decides to hit (if s/he does), the sum of the dealer cards is displayed. After the player hits, that is when the card given to him starts showing on the first two 7-segments display.

To add this feature into the system, a modulo_10 circuit should be designed to split the first digit of the sum of dealer or player cards from the second bit. The card should go through a modulo 13 circuit as done in previous labs (to separate value from suit). As shown in Figure #9 and mentioned in user interface, the sum of the player's cards will have two 7-segments displayed during the whole game. Therefore, Sum_player[4..0] will go through Mod-10 and then the 7-segment decoder circuits for first digit and second digit of the sum.

However, for the sum_dealer and the card value after going through mod-10 and mod-13 respectively, multiplexers should be added to choose whether to send the suit of the card or the first digit (starting from the left) of the sum to the 7-segment decoder circuit. Same logic is used for the choice between face value of the card of the second digit on the sum. The explained above will The select input of these multiplexers will be dependent on the press of the button Hit and the start of a new turn. This is done by using a register which will be explained in more details in the next section.

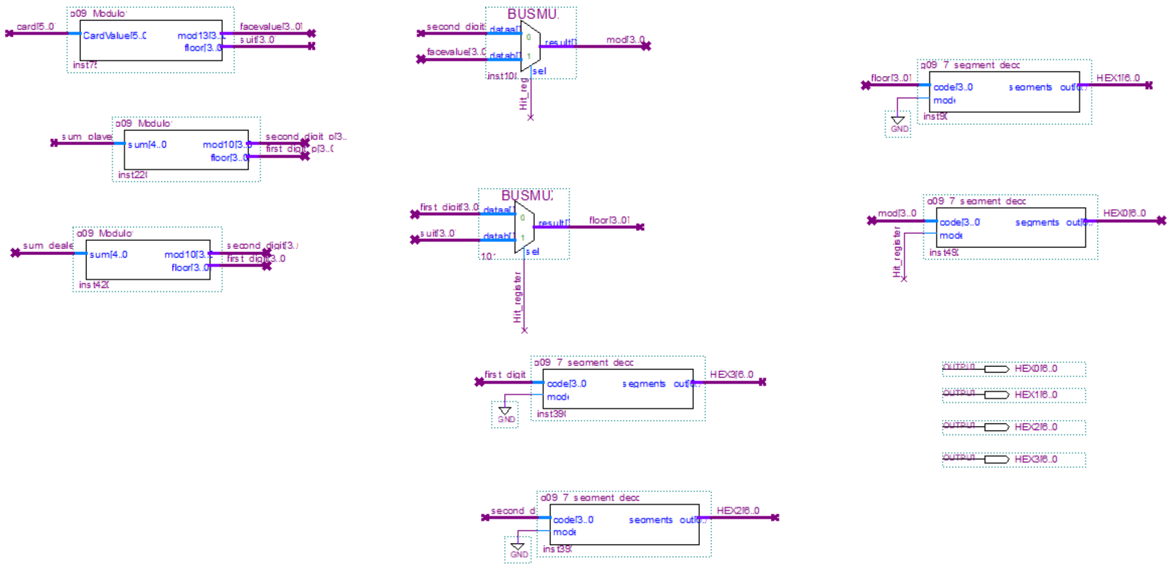


**Figure 8 - 7-segment display circuits**

## Registers

As for the registers used that keep the invalid play indicator and game over indicator showing until the next round, these are shown in Figure #10. Figure#9 also includes the register that saves the signal that selects the card value and its suit to be displayed on the 7-segement display after the player hits a new card. The input of this register is the pushbutton Hit and the register will update its value whenever the pushbutton is pressed or a new round starts (the OR gate). The register will clear its value when a new game is started (Reset = ‘1’). This will allow us to show the sum of the dealers cards before the player makes a decision of whether to hit or stop and the card given to the player after the button Hit is pressed. Similar logic has been used for the other two D Flip Flops. This time, however, the flip flop will update its value every clock cycle but will clear its value after each new round or new game. That means that the invalid play indicator or the game over indicator will stay displaying until new round button or Reset button is pressed. Refer to the state diagram of CC (Figure #5) and focus on states I and M to understand the logic explained above clearly.

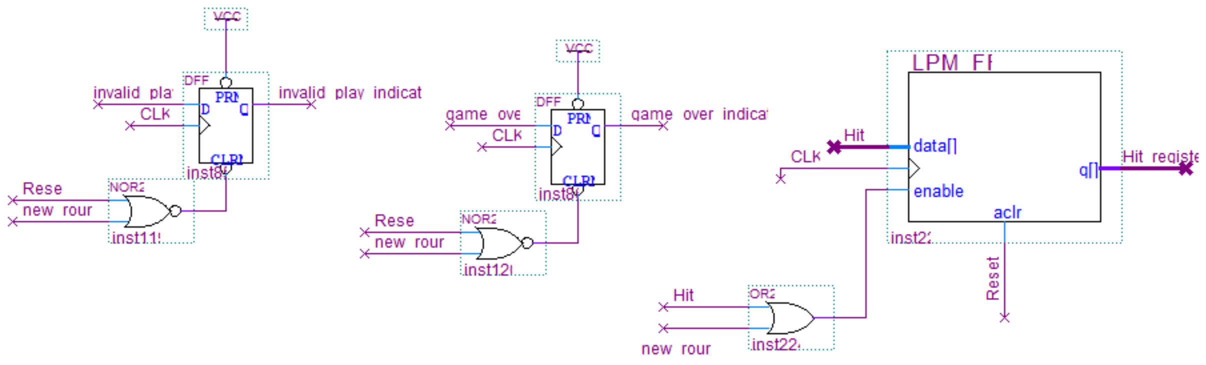


**Figure 9 – Registers used**

## Testbed and Rules(Player & Dealer)

The only part to complete the whole system is the part where new random cards are given to the player and dealer, sums of cards of dealer and of players are computed, and legal play of player is determined. This is done by having the dealer_testbed circuit from lab4 that outputs a random card from the deck of cards. This card will then be connected to the human_rules circuit or dealer_rules circuit depending on who's turn it is on getting that specified card. An explanation of both rules circuits will follow. The way of determining the right order of giving the cards (to the player or dealer) is done by using the Done1 and Done2 outputs of the HC and CC. The way it has been implemented exactly is shown in Figure 10.

As seen in that same Figure 10, the dealer's rules circuit has inputs of sum_enable, an asynchronous reset, and a clock. As the rules is synchronous, it adds only when needed, that is when Stack_en is active, i.e. a card has been popped, and sum_dealer does not constantly add at every clock cycle as it is feeded back to itself. Moreover, as the sum of the dealer cards is connected from the output to the input of the same component, then it should be synchronous, otherwise it wouldn't function.

The Human Player Rules had the same modifications as the Dealer Rules, plus a SEL enable input that selects the ACE to be 11 depending on the user input. Within the Human Rules, there is a signal Sel_en that is dependent on a clock cycle and determines whether the current card is an ACE and adds ten or nothing to the sum depending on the choice of the user. Moreover, legal_play only matters for the human player as the human player starts the game. Therefore, it becomes an input for the controllers as well as a counter. On the other hand, the legal_play of the dealer rules is unused, therefore omitted.

The destination of the outputs of both rules circuits have already been explained in the Design Method.

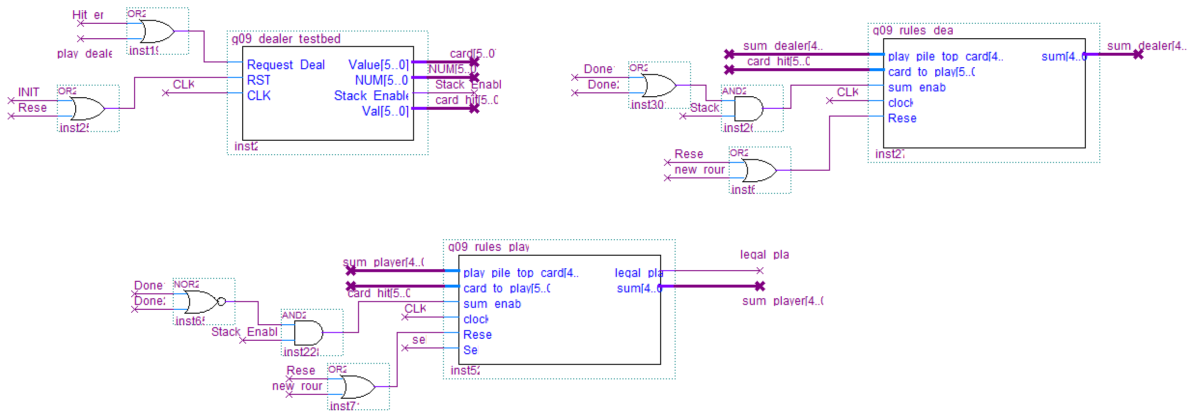


**Figure 10 - Players Rules and Dealer Testbed**

## SYSTEM TESTING

Testing of the system was performed by first testing the individual components of the datapath and then once each of them were validated, the entire system was tested. The preliminary testing of block components include the comparators and the counters, specifically g09_compare21.vwf and g09_counter.vwf as well as g09_Modulo10 that were all designed using VHDL. The other comparators such as g09_compare16 have been tested later as a part of the whole system they all follow the same logic in VHDL, and therefore it becomes redundant if they're all tested individually. Some components that were designed in previous labs were modified, and therefore tested again such as rules.

The individual components that were tested other than the preliminary and previously tested components are:

- Dealer Player Rules (g09_rules_dealer)
- Human Player Rules (g09_rules_player)
- Dealer's Controller (g09_computer_player_fsm)
- Human Controller (g09_human_player_fsm)

## Dealer Player Rules

Below is a schematic of g09_datapath_testing, which was created to test for the rules of the dealer before putting it in the whole system, that is g09_datapath. The comparator that checks whether the sum is less or equal that 16 is also tested.

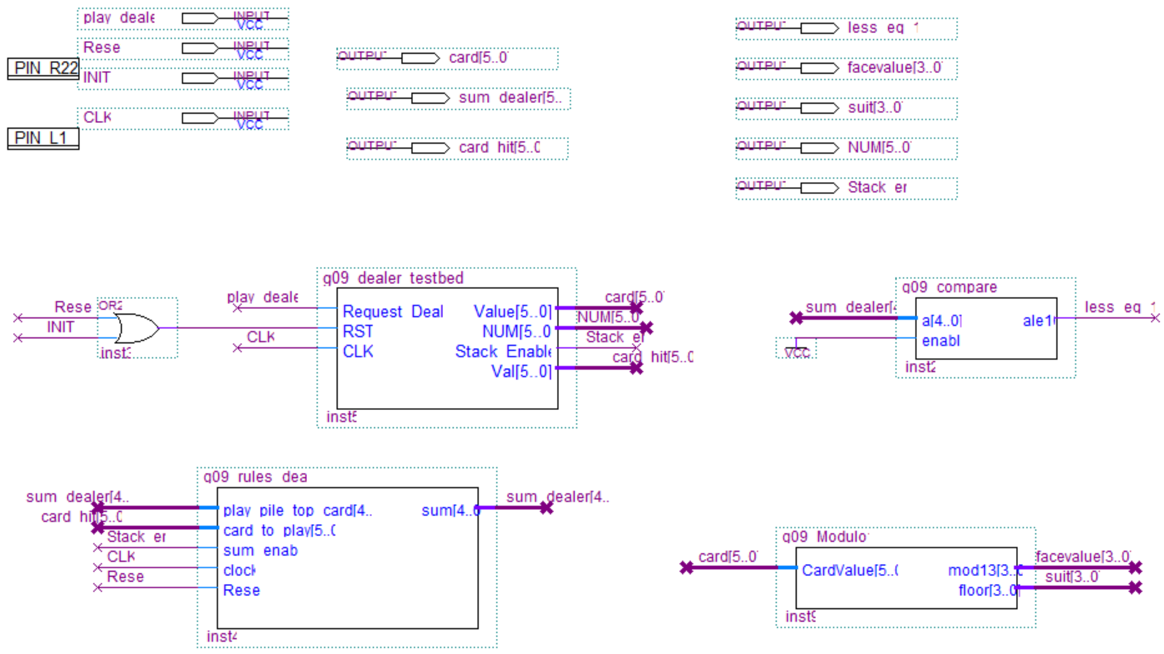


**Figure 11 - Body Schematic of Datapath Testing with Dealer's Rules and Dealer Testbed**

Below is the waveform file of the dealer's rules test.

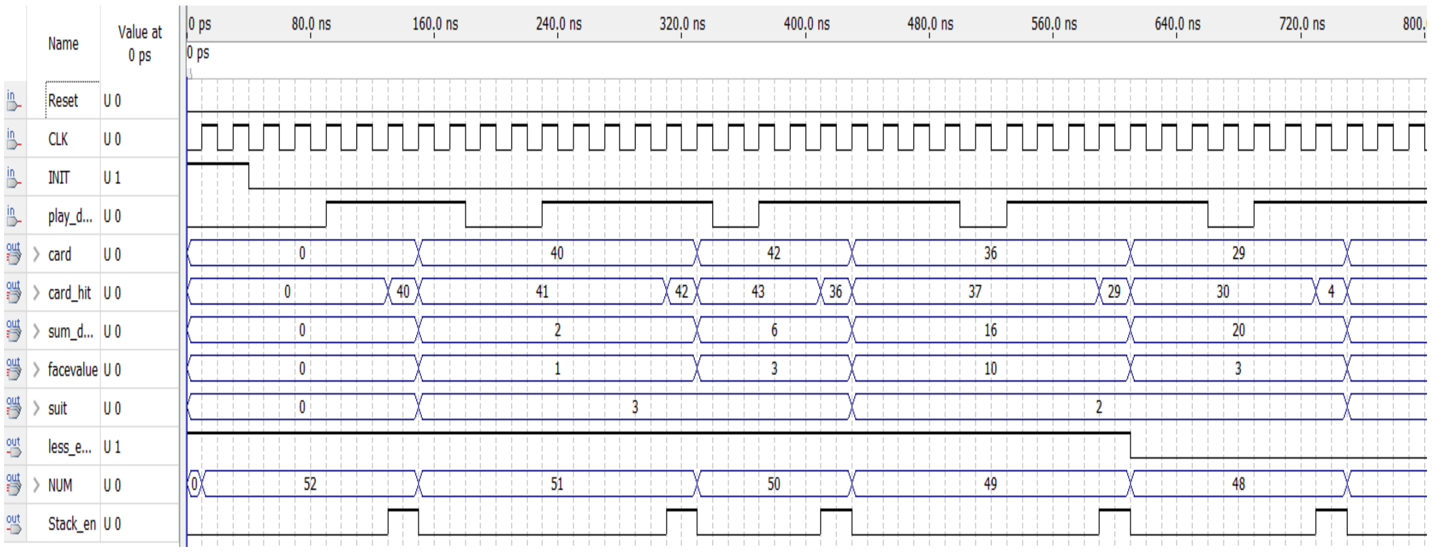


**Figure 12 – Dealer Player Rules Test**

As can be seen in Figure 12, at initialization, the number of cards is 52. Then, at every rising edge where Stack_en is active, a card is popped. The sum_dealer adds its previous value to the card_hit, that is the card that has been popped. For instance, sum_dealer equals 6 as it adds its previous value of 2 to 4 (facevalue represents facevalue_without_addition in rules so 1 has to be added to it, therefore 3 + 1). Therefore, this test shows that g09_rules_dealer is fully functional.

## Human Player Rules

Below is a test, where the Sel has been tested, as well as legal_play.

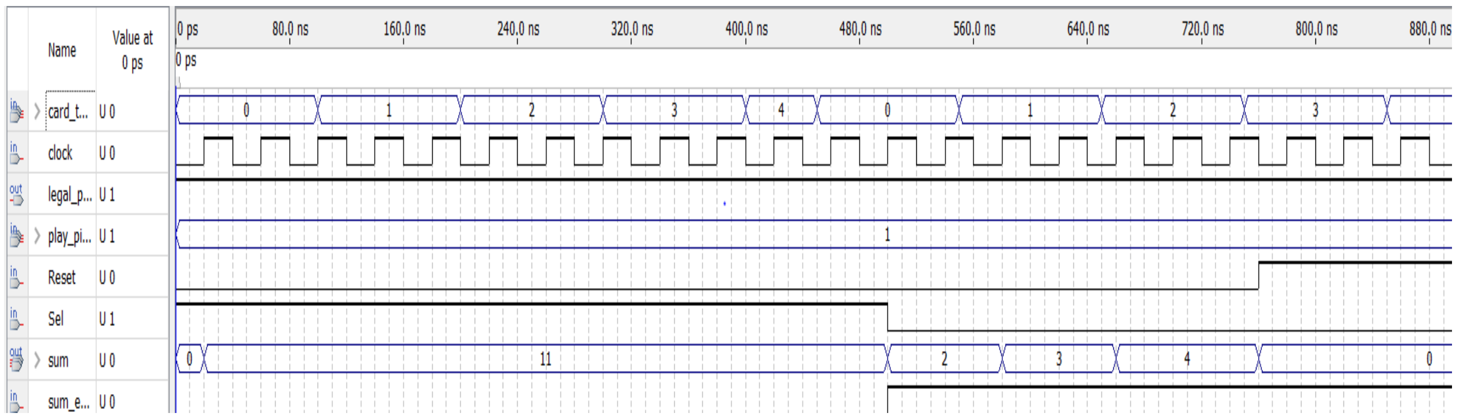


**Figure 13 - Human Player Rules Test**

In this test, when Sel is 1 and the card is ACE, that is represented by card_to_play to be 0, the sum is increased by 11 regardless of sum_enable as it is not dependent upon it. However, when Sel is 0 and the card is ACE, then the sum is only increased by 1. This test reflects exactly what is intended by this game.

## Dealer's Controller

Below is a test of the dealer's controller with random inputs, but with Reset equal to 0.

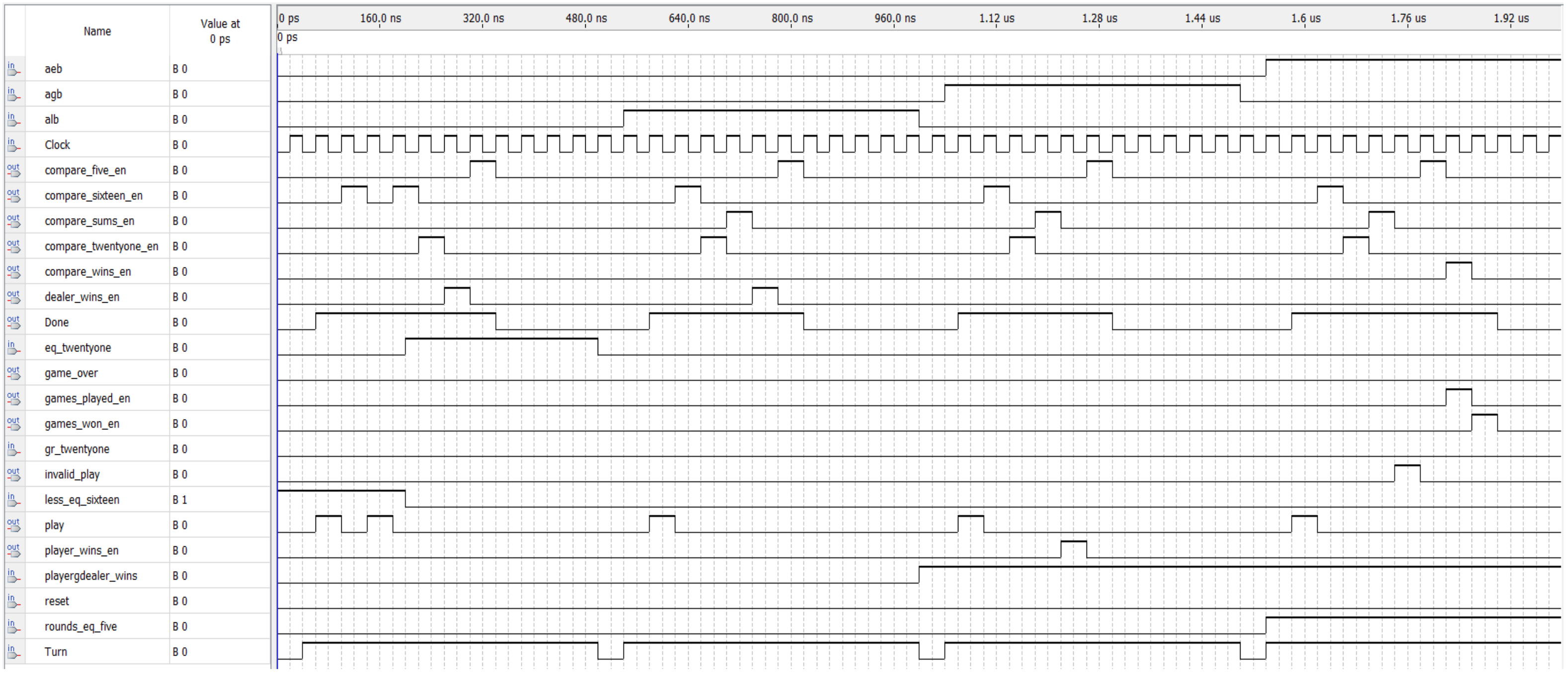


**Figure 14 - Dealer Controller Test**

As can be seen in Figure 14, the dealer's controller goes to state C after detecting Turn to be active, and play becomes active right away, which enables the dealer testbed in order to get the second card. The dealer then has to hit or play again depending on less_eq_sixteen as mentioned in state D. Moreover, dealer wins whenever eq_21 is active as shown by the ouptut dealer_wins_en.

Moreover, Done becomes 0 whenever the dealer finished its turn. As can be seen at the second time Turn becomes active, eq_21 and gr_21 are equal to 0 when compare21_en is active at state D, therefore the compare_sums_en is activated on the next state where the two players' sum are compared. The controller then decides the dealer_wins is activated due to alb is activated, that is sum_player < sum_dealer. The same logic apply to the next turns with agb and then aeb, where invalid_play becomes active.

Finally, for the last turn, games_played is actvated as rounds_eq_five is active, which is what is expected. The output games_won_en is active and game_over is low as playergdealer_wins is active which means that the player_wins rounds are greater than the dealer_wins rounds in a game of 5 rounds. The output player_wins is only active when agb is active as opposed to dealer_wins where it becomes active when alb or aeb is active, after reaching state F where compare_sums_en is active.

This test was successful as it met all our expectations.

Below is another test, but with Reset as high at a certain point.

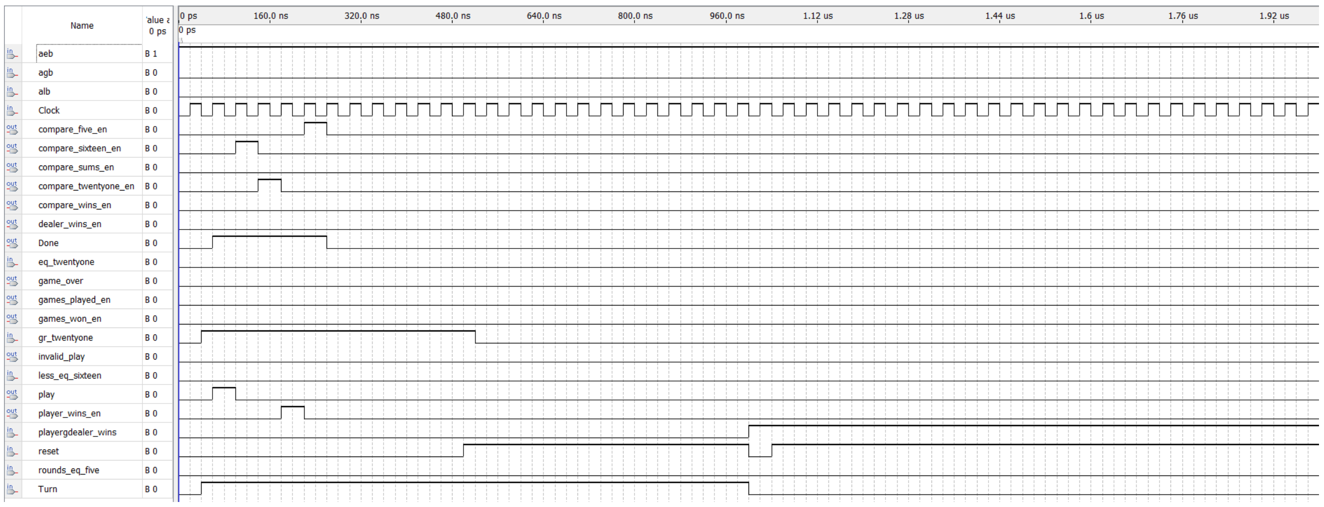


**Figure 15 - Dealer Controller Reset Test**

As expected, when Reset is high, all outputs are low.

## Human Controller

Below is a test of the human's controller.

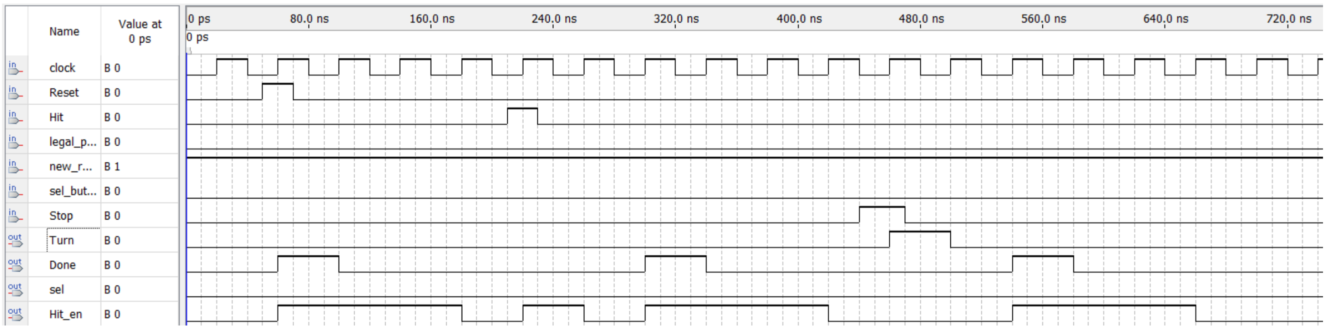


**Figure 16 - Human Controller Test**

The human controller goes to state C at the rising edge where Reset is high. The output Done becomes high whenever the human controller is at state C as expected in order to activate the dealer's rules. It goes through the initialization process where one card is given to the dealer at C and two cards to the player at every state afterwards. The dealer then goes to state F where it waits for an input to 'Hit", "Select" or "Stop". In this case, "Hit" is selected, so Hit_en becomes one. Later, at the second state F occurrence, "Stop" is chosen, which reults in Turn to become high. This test was successful and met all our expectations.

## Datapath

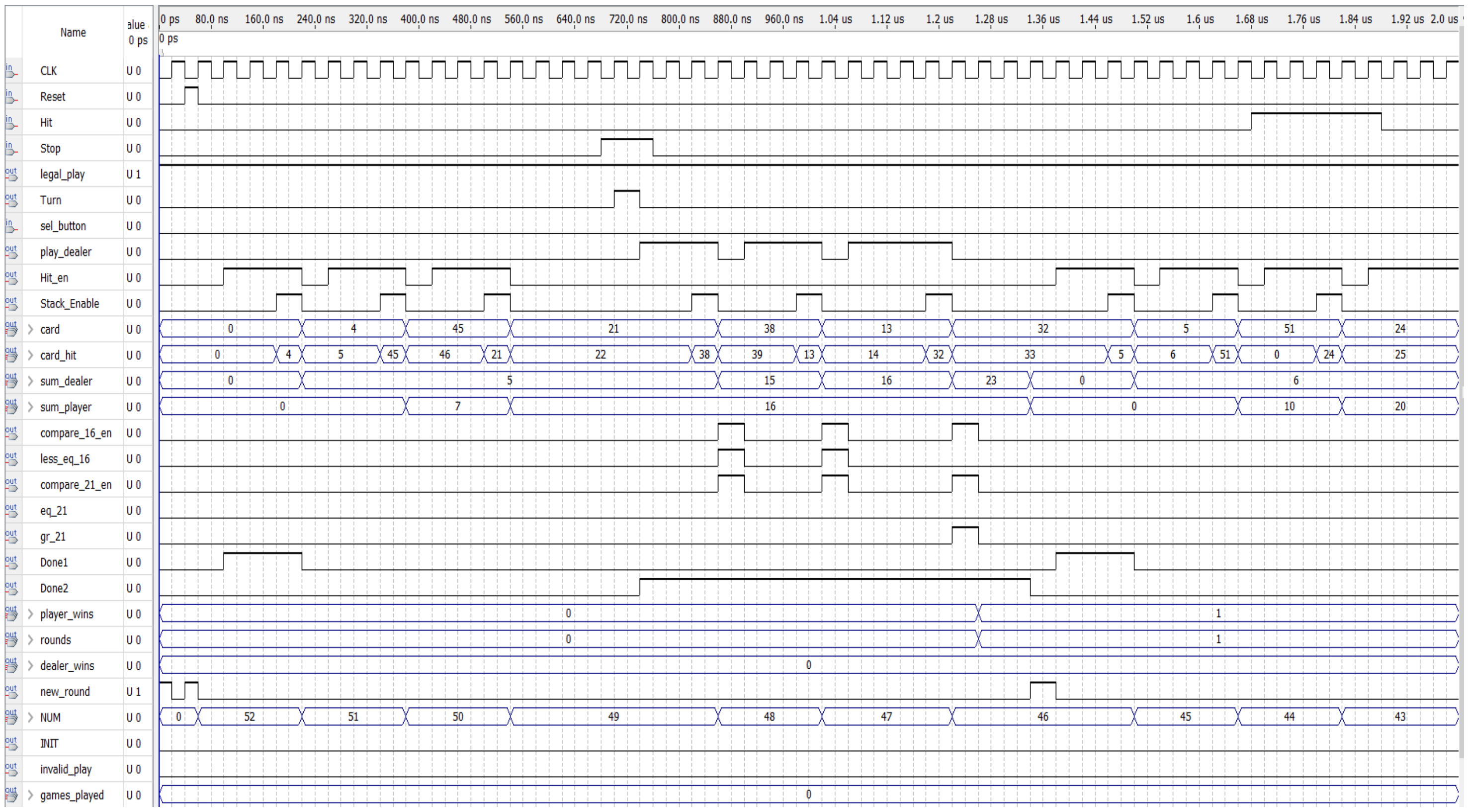


**Figure 17 - Datapth Test**

As can be seen in Figure 17, the human player starts his turn until he/she presses “Stop” to switch to the dealer’s turn. The player successfully pops cards and sums the card values whenever Stack_Enable goes high. Since the player does not hit, legal play remains high as the value is less or equal to 21. When the dealer plays, Done 2 is asserted (used along Done1 to active the dealer rules) until the end of the dealer’s turn, then new_round is asserted. In this particular test, the dealer loses as the computer gets ends up with a sum_dealer of 23, therefore player_wins is incremented by one and gr_21 is high as shown . Rounds is incremented by one at the end.

Test with "Hit":

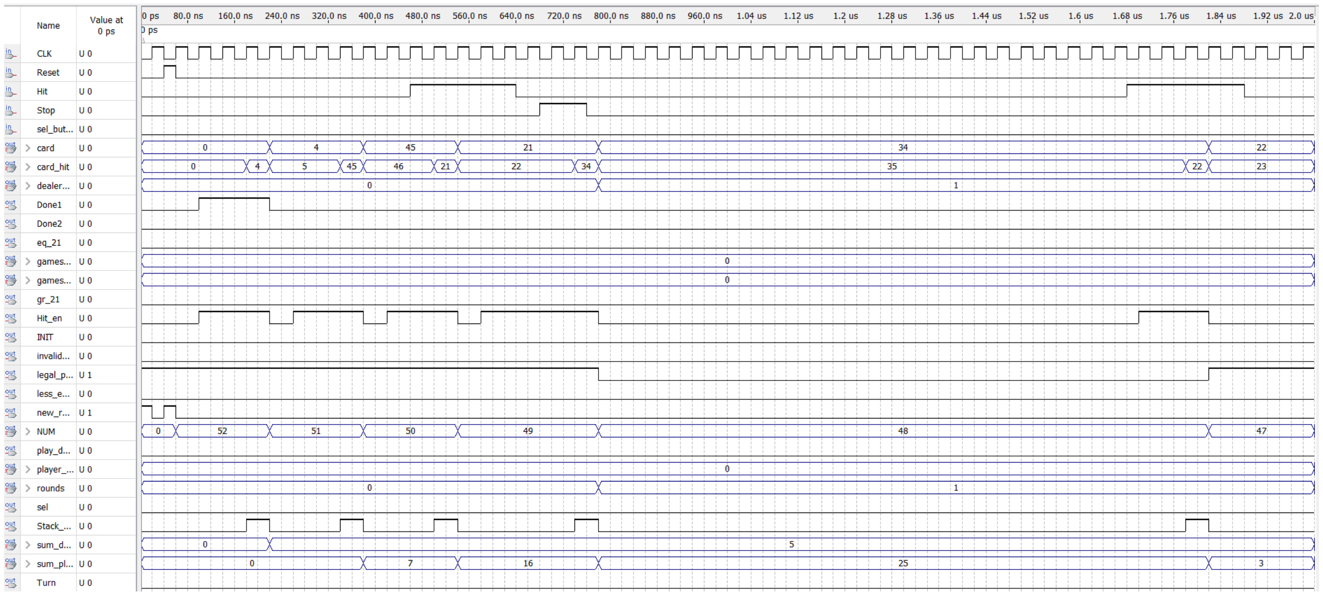


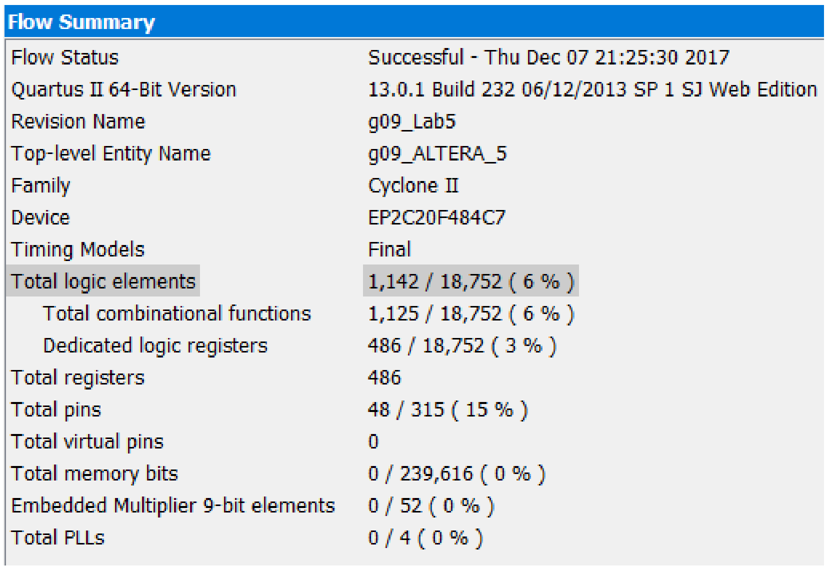
**Figure 18 - Datapath Test with "Hit"**

The human player "hit" this time and got a sum of 25, which equates to a legal play of 0, meaning dealer wins is incremented by one and Turn remains 0 as it does not switch to dealer's turn and rounds is incremented by one.

These two tests output what we expected and so the datapath works.

## Resource Utilization

A summary of the FPGA resource utilization (from the Compilation Report's Flow Summary) is shown below where the total number of logic elements used is highlighted.



| Flow Summary | |
|---|---|
| Flow Status | Successful - Thu Dec 07 21:25:30 2017 |
| Quartus II 64-Bit Version | 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition |
| Revision Name | g09_Lab5 |
| Top-level Entity Name | g09_ALTERA_5 |
| Family | Cyclone II |
| Device | EP2C20F484C7 |
| Timing Models | Final |
| Total logic elements | 1,142 / 18,752 ( 6 % ) |
| Total combinational functions | 1,125 / 18,752 ( 6 % ) |
| Dedicated logic registers | 486 / 18,752 ( 3 % ) |
| Total registers | 486 |
| Total pins | 48 / 315 ( 15 % ) |
| Total virtual pins | 0 |
| Total memory bits | 0 / 239,616 ( 0 % ) |
| Embedded Multiplier 9-bit elements | 0 / 52 ( 0 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

**Figure 19 - Full System Resource Utilization**

## CONCLUSION

### Significant Issues in the Design Process

1. Figuring out what the controller should do in each state was difficult. A simple wrong control statement in a state might have led to race conditions. For example, figuring out when to pop from the dealer testbed after the request deal has been sent out to the dealer_FSM was difficult.
2. Following on 1, the dealer_FSM waits for a low request deal to go from state A to state B, which caused problems when doing the human controller fo rthe initialization part as request deal (hit_en) is always one for state C until E. Therefore, we added empty states between them with hit_en being low X, Y, Z.
3. Having to take into consideration of so many states all together was a daunting task.

### Possible Enhancements

1. State minimization could be applied to my controller_FSM to minimize the number of states which would result in lesser combinational logic and as such lesser propagation delay.
2. The new round pushbutton could be pressed at any time during the dealer's turn which would annul its turn and start a new round. However, technically speaking, the computation of the computer player is so fast that there is no way of pressing on the new round while the dealer is playing unless it reaches state I (invalid play) or state M (game over) where we wait for the new round button press to start a new round.
3. Selecting Ace is not an option for the dealer as it would be difficult to chose what is the best choice. As for the player, selecting Ace is not an option for the first card he/she gets during initialization, therefore if the first card is Ace, then it can only be 1. Moreover, selecting Ace cannot happen during mid-hand or at anytime the human player feels that their Ace cards should have a certain value, i.e. selecting Ace can only happen at the popping stage, and stays at that value regardless of what the next card is or if the player desires to change its value later.

## REFERENCES

[1]J. Clark, *ECSE-323 Digital System Design Lab #4 – VHDL for Sequential Circuit Design.* Montreal: McGill University, Fall 2017.
[2]J. Clark, *ECSE-323 Digital System Design Datapath/Controller Lecture #1.* Montreal: McGill University, Fall 2017
[3]J. Clark, *ECSE-323 Digital System Design Lab #5 – System Integration for the Card Game.* Montreal: McGill University, Fall 2017.
[4]Altera Corp., *LPM Quick Reference Guide.* Altera Corp., December 1996.