Eva Suska 260618967
Marcel Morin 260605670

g54_Crazy Eights Game - Lab 5

Description

Input: 1 bit CLK, RST, player button, 6 bit player_address, 2 bit player_mode
Output:
7 bit pileFace indicating the face of the top card in the middle pile
7 bit pileSuit indicating the suit of the top card in the middle pile
6 bit NUM indicating the number of cards the dealer pile has
6 bit compNUM indicating the number of cards in the computers hand
1 bit compEmpty indicating if the computer's hand is empty (ie. computer won at the end)
6 bit playerNUM indicating the number of cards in the player's hand
1 bit playerEmpty indicating if the player's hand is empty (ie. player won at the end)
7 bit playerFace indicating the face of the card in your hand the address input is set on
7 bit playerSuit indicating the suit of the card in your hand the address input is set on

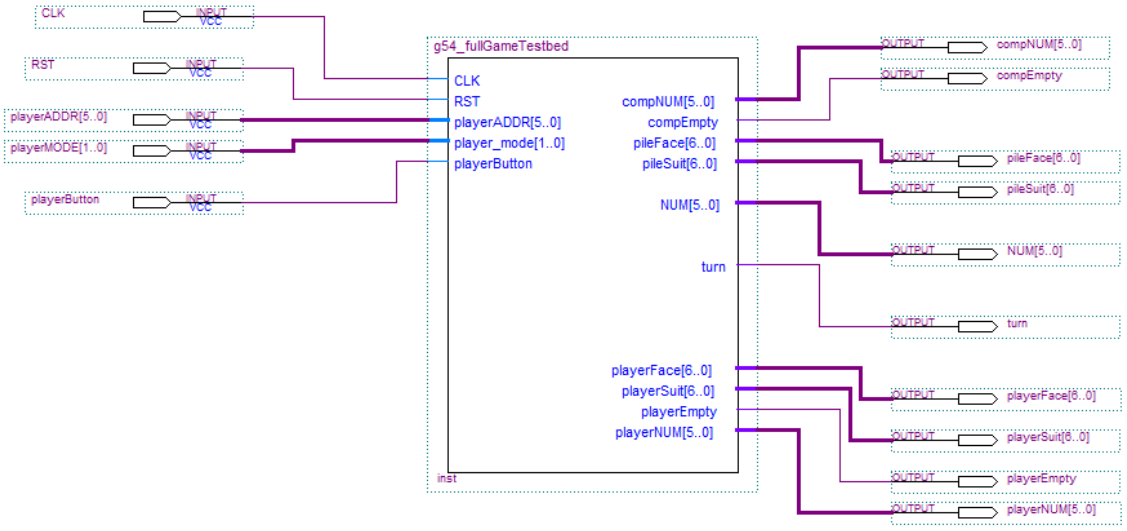

figure 1: pinout diagram of the crazy eights game

This program simulates the Crazy Eights card game on the cyclone 2 Altera board against a computer player. The goal of the game is to get rid of all your cards in your hand, first player to do this wins. This two player game deals 8 "shuffled" cards from a standard 52 card deck to each player, and then places one card on the play pile to start the game off.

The game then chooses a random player to start, this player can either place a card on the play pile (if it matches the suit or value of the card on the top of the pile or is an 8), or pick

up another card from the deck. It is then the next players turn and so on until one of the players has no more cards in their hand, this player has won.
For details of the rules of Crazy Eights game see: https://en.wikipedia.org/wiki/Crazy_Eights

---

How it works

The game is controlled by a system controller finite state machine pictured below (figure 2).

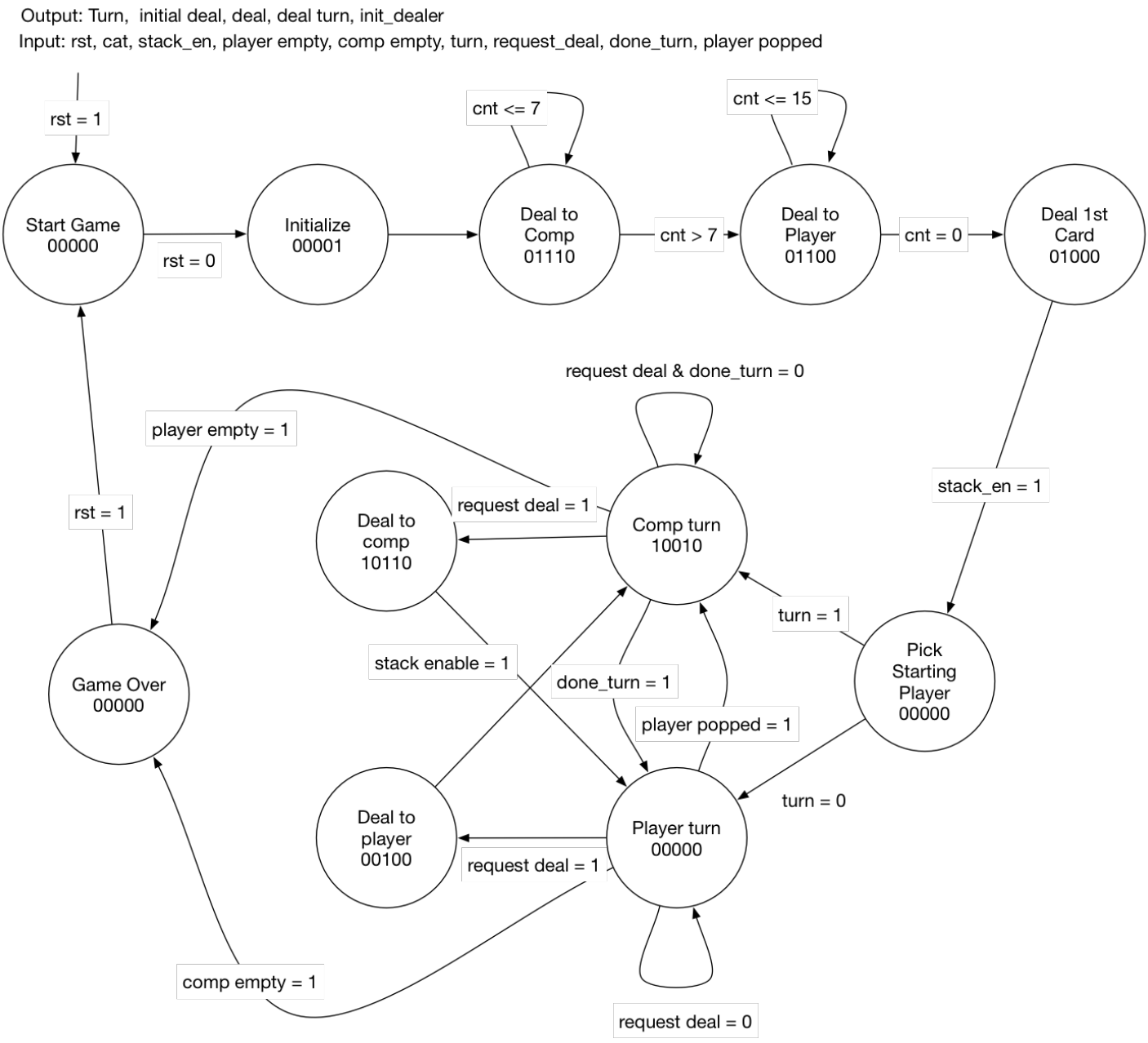

figure 2: state diagram of the system controller

Output	What it controls
turn	whose turn it is to play (1 if computer, 0 if player)
initial_deal	game is being set up, dealer is dealing out initial starting cards
deal	dealer is dealing out cards
d_turn	indicates which player the dealer should be dealing cards to (1 if computer, 0 if player)
init_dealer	initializes the dealer's stack with the full 52-standard card deck

Detailed descriptions of what happens in each state:

Start Game:
waits for reset, then goes to initialize state.

Initialize:
The high output init in this state signals to put the dealer's stack mode to init and enables the stack to fill it with all the 52 cards of the deck.

Deal to Comp:
The initial_deal output signals that the game is being set up. Deal is also high to let the dealer know we want to be dealing out cards, and d_turn is high to indicate that we want to deal to the computer.
8 random cards from the dealers stack will be placed in the computers stack. The cards are chosen by random from the Randu component, and once the counter hits 7 (count starts at 0), all the cards have been dealt, and we should now deal to the player.

Deal to Player:
The initial_deal output signals that the game is being set up. Deal is also high to let the dealer know we want to be dealing out cards, and d_turn is low to indicate that we want to deal to the player.
Like the deal to comp state, 8 random cards from the dealers stack will be placed in the player's stack, while continuing counting each card so that once the count hits 15 the 8 cards have been dealt to the player, and the count resets, bringing you to the next state, deal 1st card.

Deal 1st Card:
The initial_deal output stays high indicating that the game is still being set up.
This deals one random card to the D flip flop which represents the top card of middle play pile.

Pick starting player:
This state picks a random player (either comp or player) and brings you to that state depending on the what the turn signal generated is (1 for comp, 0 for player).

Comp turn:
Sets the output turn to 1 to indicate it is the computers turn, as well as d_turn to 1 to indicate that any card from the dealer to be dealt to the comp.

This state connects to the Computer Player FSM modulo that controls what move the computer player makes. If this FSM outputs a request_deal (if it has no card to play) this bring us to the state Deal to Comp. If the FSM outputs that it is done_turn it has already placed a card down on the pile and the state goes to the Player Turn. While request_deal and turn is 0 it stays at the state. If the player's stack is empty, that means the player has won and the state goes to Game Over.

Deal to Comp:
Turn output is high indicating it is the computers turn still, deal is high indicting that it is making a deal, and d_turn = 1 indicting it is dealing to the computer player.
This deals one random card from the dealer stack and adds it to the hand (stack) of the computer player. Once the stack_enable is high indicating the card was dealt it goes to the Player Turn state.

Player turn:
Sets the output turn to 0 to indicate it is the players turn, as well as d_turn to 0 to indicate that any card from the dealer to be dealt to the player.
Here the player can make a move. If you want to play a card on the pile, you will "pop" that card on the pile, if the rules allow it the signal player popped will be high and this will bring you to the Comp Turn state. Or you can request a card from the dealer, which enables the request deal signal and brings you to the Deal to player state. If the computers stack is empty, the computer has won and this brings you to the Game Over state.

Deal to Player:
Turn output is low indicating it is the computers turn still, deal is high indicting that it is making a deal, and d_turn = 0 indicting it is dealing to the player.
This deals one random card from the dealer stack and adds it to the hand (stack) of the player. Once the stack_enable is high indicating the card was dealt it goes to the Comp Turn state.

Game Over:
One of the players has won (one of the players stack is empty). This will be indicated on the board, and then waits for the rest signal to go high to bring it back to the Start Game state.

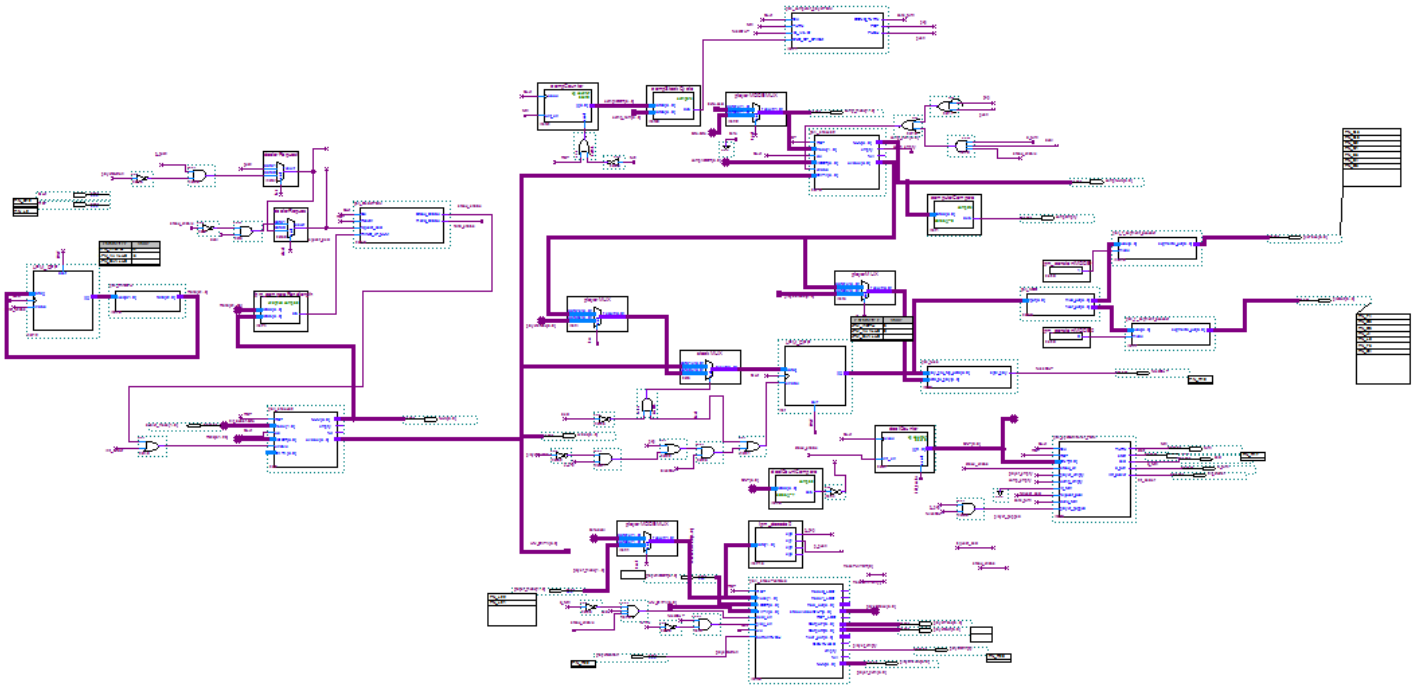

figure 3: crazy eights game circuit

Eva Suska 260618967
Marcel Morin 260605670

Entire Game: To explain how our entire game (figure 3) was set up we will split it up into schematic components.

Schematic Components:

Dealer:

The dealer circuit (figure 4) is composed of the Dealer finite state machine (see lab 4 for details), the dealer's Stack, the Random number generator to deal out a random card, and the lpm_compare to make sure that the random number generated is within the number of cards left in the stack. This is essentially the Lab 4 dealers modified to have its stack output onto a shared bus line that can be accessed by either player and the play pile (depending on who needs a card).

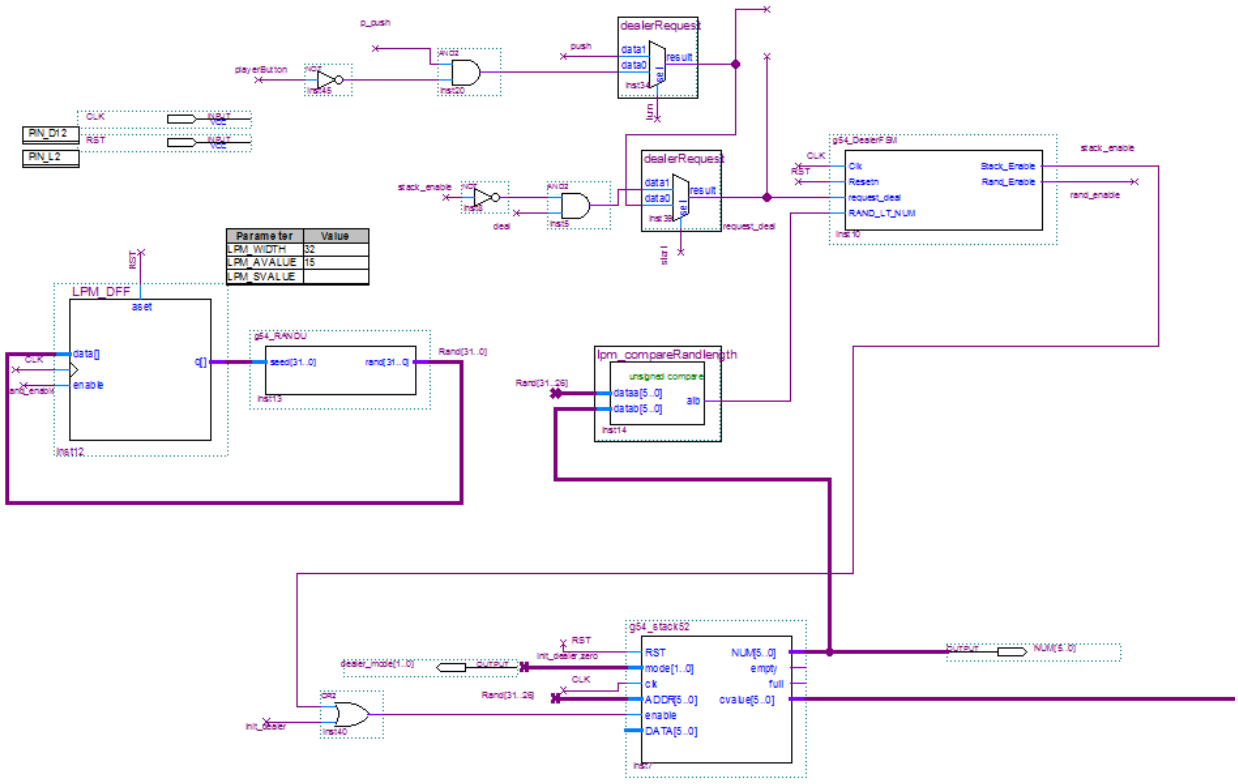

figure 4: dealer schematic

The multiplexers are to control when the dealer gives out cards. During the deal stage (at the beginning of the game) the start signal is high and therefore the multiplexer just lets the output of NOT stack_enable AND deal determine the request_deal of the dealer FSM. When the start signal is set to low (Dealer has dealt cards and place a card on top of the play pile) the signal that now determines the request_deal is dependent on whose turn it is as the turn signal controls the next multiplexer. In this case, if turn is high the computer will be the one controlling when a card is to be dealt to them. We don't need to AND it with anything else as based on how we designed our computer player FSM the push signal, which demands a card to be dealt to

them, will only go high for a couple clock cycles. When the turn is low it is the human player asking for a deal. This action is triggered through the pulse generator output, however we need to check that this is a push action and not something else, which is why we AND it with the p_push signal. This p_push signal is flagged high whenever the players mode is in push. No other control signals are needed.
Lastly, the dealer has an **init_dealer** OR'ed with the **stack_enable** of the Dealer FSM at the enable input of the stack. Naturally we want our stack to be enabled when the Dealer FSM has found a card to pop to whoever requested for a card, however we also needed to initialize the stack at the start of the game to load in all the cards. This required a signal to enable the stack as it initializes at the start of the game when the dealer_mode is put to initialize.

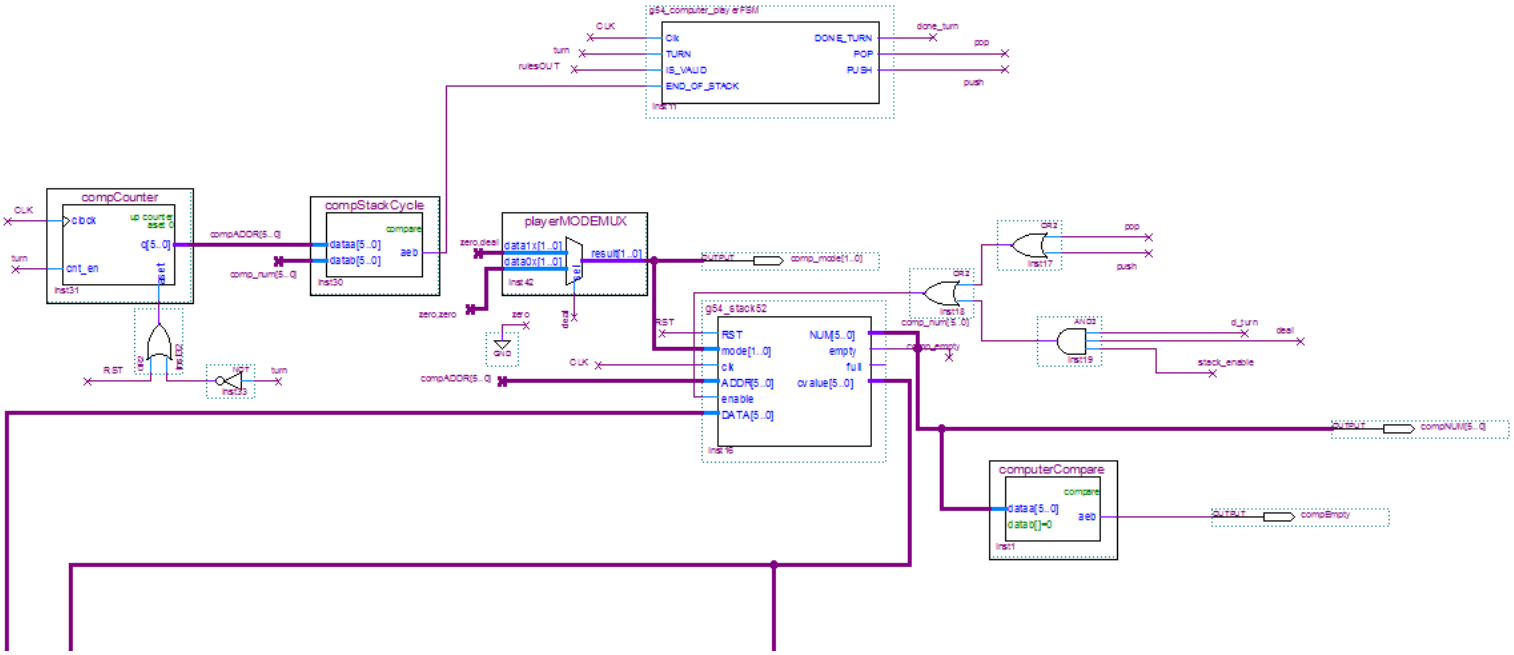

figure 5: computer schematic

Computer: The computer player (figure 5) is made up of the computer FSM, a stack with control inputs, and a counters to help the FSM cycle through the card in its stack as his hands grow or shrink in size. Linking this diagram with the previous Dealer schematic, the DATA line of the computer's stack is connected to the output line to the Dealer's stack. This dealer card line is also connect to the player's stack. We don't use a multiplexer here as the control variable that determines who intakes the dealer card in the stack is the Controller FSM which controls the enable signals of this entire system. Therefore we can put our card dealt onto this shared line and it wont get pushed into the two stacks connected to it. The enable input of our computer stack is **pop** OR **push** which is a signal outputted by the computer FSM and indicates that the stack needs to be enables to either receive or play a card. This is controlled by the turn input which is an output of the controller FSM. The other signals ANDed together are used at the start of the game when cards are being dealt. At the starting stages of the game, the **pop**, **push**, & **turn** signals are all low so we created a **d_turn** signal to set which stack gets its cards dealt to

at the beginning. When **d_turn** is high, the **deal** signal is high(meaning the dealer is in the state of dealing a card), and the **stack_enable** signal is high(signal outputs by the dealer FSM to pop a card) the comp players stack will get enabled and receive a card into its stack.
Other control signals to note are the counters set signal which resets the counter after the computer player has finished playing so that next turn it can cycle through all of its cards again, and the mode multiplexer. The multiplexer control signal is **deal** so that when the dealer is dealing a card the mode of the stack will be set to push to be able to receive a card. This may not necessarily mean that the computer player will receive a card as the enable signal still needs to be flagged high.
Lastly the computer's stack is compared to zero so that when the computer plays its last card the **compEmpty** signal is set to high and indicates that the computer has won.

Computer Player FSM:

INPUT:
TURN indicates its is the computers turn
IS_VALID if the card is valid to play from the rules
END_OF_STACK indicates it tried all of the cards in the hand

OUTPUT:
DONE_TURN = 1 when the computer has made its move
PUSH to get a new card from the dealer
POP to play a card

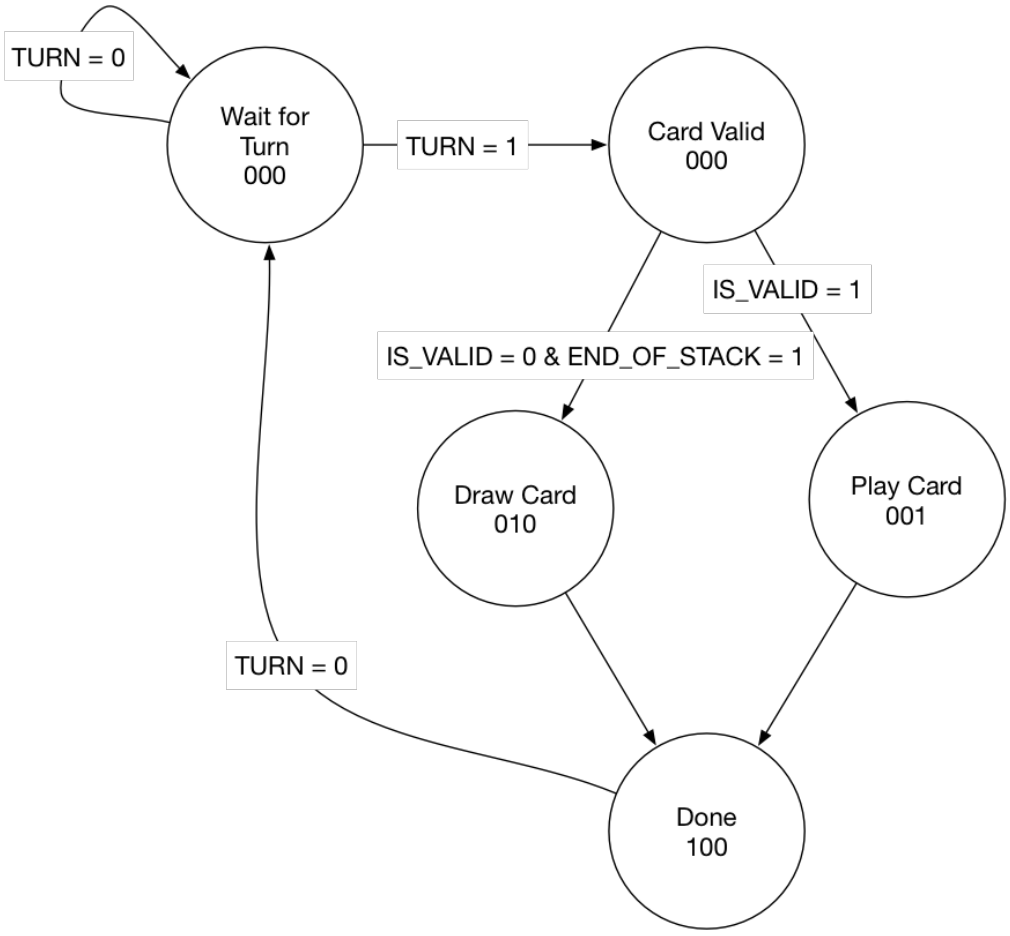

figure 6: computer player FSM state diagram

Eva Suska 260618967
Marcel Morin 260605670

Wait for turn
De-assert the DONE signal when the TURN signal indicates the Human Player's turn. Look at the TURN control signal if it is 1 then go to the state Card Valid.

Card Valid
Scan through the cards in the Computer Player's Hand (stack) one at a time. For each card, check the validity of playing it, using the rules module. If the card is valid go to play card state, if it not valid and you've reached the end of the stack go to draw card state.

Play Card
The card is valid then play it (pop it and transfer it to the play pile), and assert the DONE signal (the main controller will then change the turn indicator to point to the human player). Then go to the done state.

Draw Card
If no valid card is found, then draw a card from the Play Deck (i.e. pop the play deck and transfer (push) the card to the computer player's hand). Then go to the done state.

Done
Assert the DONE signal, and return to the wait for turn state when the TURN = 0.

Player: Our player component (figure 7) doesn't contain a FSM as you are the one deciding what actions to take and even though a correct play is available, you could possibly ask for a card to be dealt even though you have a playable card in your hand. We took the stackTestBed module from lab 3 and used that as our main block component and added more control inputs to suit our needs. The first component we added as the decoder for the mode the player was in. This would flag the corresponding **p_push** or **p_pop** signal. The multiplexer is controlled by the **deal** signal so that when the dealer is dealing a card the stack is '01' mode (push). Otherwise the player is free to pick any mode except **init**(10) which we disabled. The playerButton is inputed into a pulse generator inside the stackTestBed module. The only control signals here

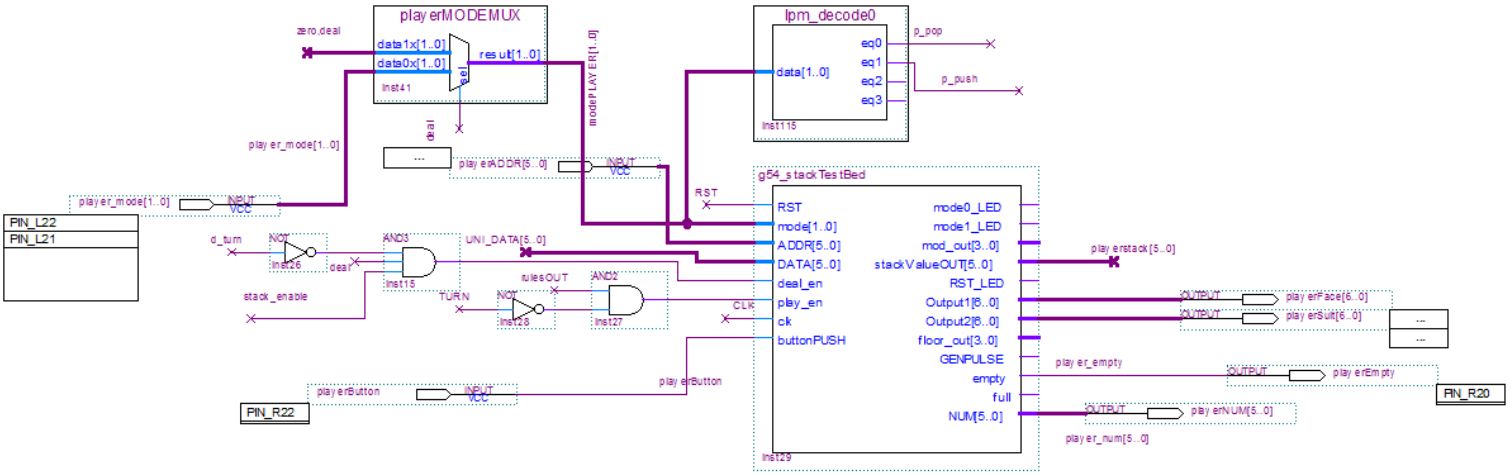

figure 7: player schematic

are the signals going into the **play_en** and **deal_en** block inputs. The play enable determines when the player is able to play a card into the play pile and is OR'ed at the input of the player's stack enable. This can only be done if **turn** is low (its the players turn), and if the **ruleOUT** is high (the play is valid). The deal_en input is also OR'ed at the player's stark enable and is used to enable the stack at the beginning of the game when the dealer is dealing out cards. If you compare this enable to the computer's stack enable you'll notice that neither of them can be enabled at the same time as the d_turn needs to be low for the player to receive cards. Just like the computer we output whenever the stack is zero to indicate that the player has won the game.

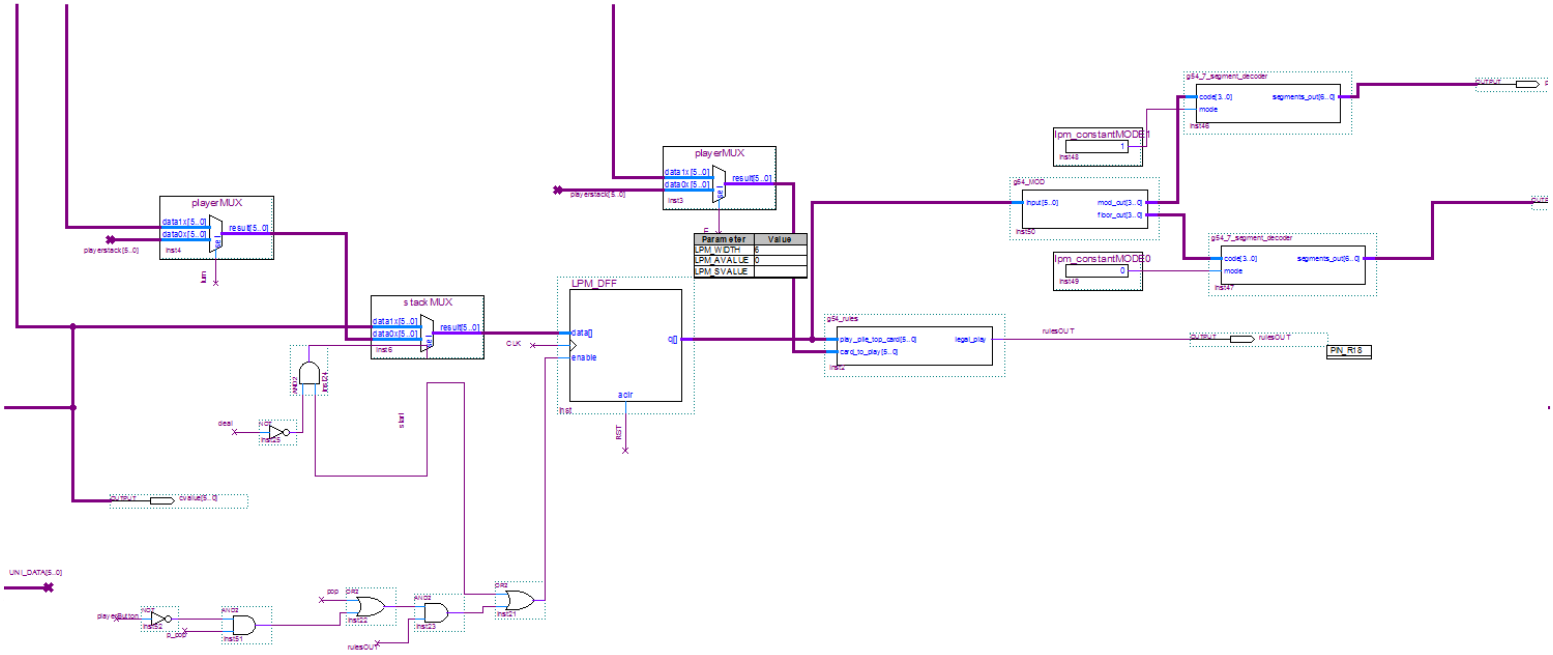

figure 8: play pile centre schematic

Play pile: The play pile (figure 8), which is a D flip flop as only one card needs to be kept for reference, is the centrepiece of this system and needs to intake a card from either the dealer (for the starting card) or either player. The stackMUX controls whether the Dealer or one of the players has access to the play pile. The playerMUX leading into the stackMUX is what controls which player gets to put a card on into the play pile. The stackMUX is controlled by the start and deal signal. This is to make sure that the dealer isn't dealing cards into the play pile and also makes it so that the dealer has access to the play pile when it is done dealing and still at the start of the game, ie. when **start** is high and **deal** is low. The flip flop itself also needs to have an enable in order to control when its can receive a card. This is either when start is high and the dealer plays a card onto the pile, or when **rulesOUT** is high and the respective player pops a card.
The output of the play pile is then put into the rules module so that players can have their selected cards compared and the play can be flagged as legal or not. The card compared to the play pile is controlled by a turn signal so the player's whose turn it is will have their card compared. We also connected the play pile card to the Mod component to display the card on our board.

For details of about the rules modulo and dealer FSM see lab 4. For details about the stack see lab 3. For details about the 7-segment decoder which displays the card values and the random number generator Randu see lab 2.

User Interface

Code on Altera Board	Representation
SW9 switch	Resets the game
SW8 - SW3 switches	binary address of cards in your hand (where SW3 LSB)
SW1 - SW0 switches	operation (move) you can make (see below for details
KEY0 button	Button to preform operations
LEDR9	ON if your turn
LEDR0	ON if your move was invalid
LEDG7	ON if computers hand is empty ie. the computer won
LEDG6	ON if your hand is empty ie. you won
HEX3	Top card on pile suit
HEX2	Top card on pile value
HEX1	Your card's suit
HEX0	Your card's value

representation of the keys on the Altera cyclone 2 board

- To start the game you must reset. This is done by switching the switch SW9 on the board up (left most switch). This will set up the game for you, deal the cards out (8 random cards per player) and place a card on the pile to start.
- It will now be your turn, this is indicated by the red LEDR9 (left most red led on the board). If the light is on it is your turn if it is off it is the computer's turn.
- The card on the top of pile will be displayed on the 7 segment LEDs. The suit of the card on the top of the play pile will be displayed on HEX3 represented as numbers 0, 1, 2, 3 for each suit (figure 9).

figure 9: display of the suits of the cards

- The face value of the card (figure 10) of the top of the pile will be displayed on the next 7 segment LED HEX2.

figure 10: display of the face of the cards

- Your hand of cards will be displayed on the next two 7 segment LEDs. The suit value displayed on HEX1 like as above and the face value displayed on HEX0.
    - You can go through your hand by changing the address of your stack. The address of your stack is displayed using the switches SW8 to SW3 as the binary address of your stack, with SW3 being the LSB (least significant bit). The switch down as 0 and the switch up as 1. ie. the first card of your hand should be displayed when all switches are down (address 0 of your stack) and to see the next card switch SW3 up (address 1) and so on.
        - Note that all of the cards in your hand will be at the beginning of your stack.
        - If your stack is empty at that address the value should be displayed as a dashed line.
- The operations or moves you can do are controlled by the right most switch SW0. The button that allows you to perform a move is KEY0.
    - To play a card (pop it onto the middle card pile), select a card from your hand using the address and if the card is able to be played according to the rules (same suit or same face value as the top card on the play pile or an 8) then make sure both the switches (SW1 and SW0) are down and then press the button KEY0. This will now place your card on the top of the pile.
        - If the card you have selected is not playable according to the rules, your card will not be placed at the top of the pile, and the red LEDR0 will be on indicating it is an invalid play as well as the red LEDR9 indicating it is still your turn will be on, and you can still make a move.
    - To request a card from the dealer (usually when you have no cards to play) switch SW0 up (make sure SW1 is down) and press the button KEY0. This will add another card to your hand from the dealer.

A complete list of the operations are below. They are encoded as a binary value with SW0 being the LSB.

Operation	Binary Value	Switches On
POP (play card)	00	
PUSH (request a card)	01	SW0
INIT (don't need to use when playing)	10	SW1

Operation	Binary Value	Switches On
NO OP (does nothing)	11	SW1 SW0

- Once you have made a move on your turn, it now goes to the computer players turn.
    - LEDR9 should be off
    - The computer will either place a card down and you will see the top card change or will request a card
- Once the red LEDR9 is off it is your turn again and you can make a move the same way as before, and repeat.
- The game will end when you or the computer has no more cards left in their hand.
    - the green LEDG7 indicates if the computers hand is empty ie. the computer won
    - the green LEDG7 indicates your hand is empty ie. you won
- To reset the game switch SW9

Testing

We began the testing by first testing the individuals modules:
Rules:
Simulated on quartus simulator. See lab 4

Dealer FSM:
Simulated on quartus simulator, created test bed, simulated test bed, tested on board. See lab 4.

Stack_52:
Created a testbed and simulated it and then tested it on the board. See Lab 3 for details.

7-segment decoder: displayed values for the HEX0 - HEX3 on the board
Tested it when testing the stack_52 testbed on the board once we initiated the stack with all the values of the 52 cards we went through to make sure

Randu: generates random numbers
We created a Test Bench file and simulated the results on model sim. This can be seen in Lab 2.

Computer Player FSM:
We created a wave file and simulated on Quartus wave simulation to check that the functionality was working as expected.

System Controller FSM:
We created a wave file and simulated on Quartus wave simulation to check that the functionality was working as expected.

Once we knew the individuals modules worked as expected, we created a test bed for the entire game and simulated it on Quartus first then tested the functionality on the Altera board.

Eva Suska 260618967
Marcel Morin 260605670

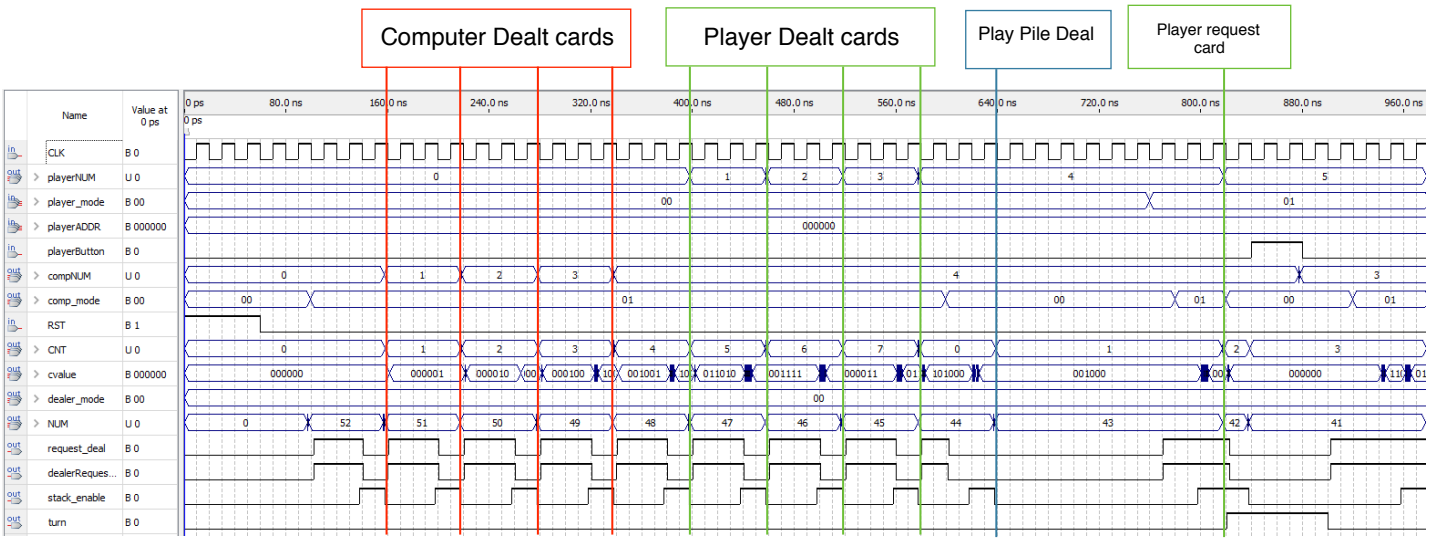

figure 11: Quartus simulation of the overall system

Overall system: We changed the cards being dealt at the beginning to 4 cards per person so that we could simulate the dealing of cards at the start of the game and also have the player make a play. In the simulation results (figure 11) we made both the computer and player play a turn. This was successful as you can see the computer ask for a card and the player also getting a card pushed. The dealer's card number goes down from 44 cards to 43 as it also has to play a card to the play pile after having dealt 4 cards to each player. The player the asks for a card immediately after getting a card pushed to the player the computer takes his turn. At this stage our system works properly and it was time to move to the entire implementation to the board.
Due to the fact that our Quartus simulation was restricted in length we had to do the rest of the testing on our physical hardware.

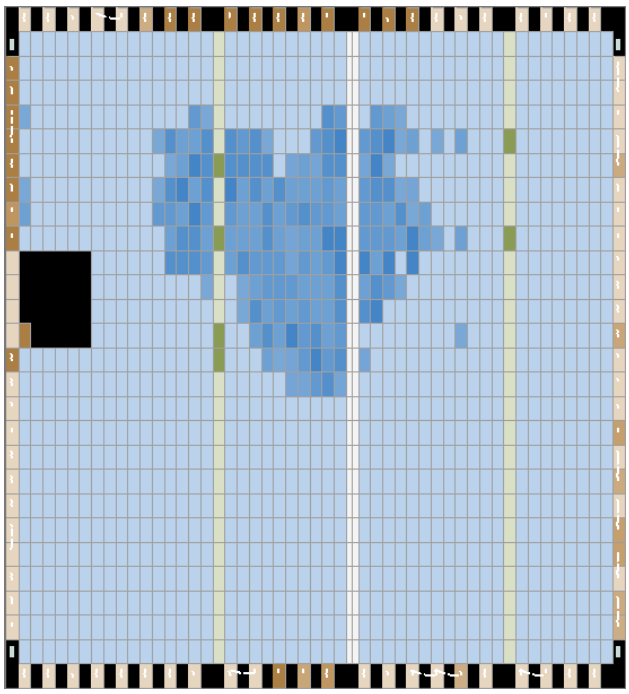

figure 12: Chip planner

Flow Summary	
Flow Status	Successful - Tue Apr 11 18:25:46 2017
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Full Version
Revision Name	g54_Lab3
Top-level Entity Name	g54_fullGameTestbed
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	2,216 / 18,752 (12 %)
Total combinational functions	1,911 / 18,752 (10 %)
Dedicated logic registers	1,031 / 18,752 (5 %)
Total registers	1031
Total pins	83 / 315 (26 %)
Total virtual pins	0
Total memory bits	9,984 / 239,616 (4 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

figure 13: Flow summary

FPGA Resource Utilization

The darker blocks in the Chip Planner (figure 12) indicate what design takes up. For our specific design we occupied less than 20% of the total board which according to our Flow Summary (figure 13) total logic elements is around the right percentage. We could have obtained an even smaller block occupation through using smaller stacks for our hands as there is no way to fill up your stack completely as a player. In the end we decided to just use our full stacks as there was plenty of space on the board for it.

Conclusion - Issues that Arose

As mentioned in the testing section, the errors that arose through this lab were noticed when testing the design on the physical board and trying to complete the game. The main error we noticed was the computer or player receiving too many of the same card value when requesting a deal. Although our deal at the start of the game worked perfectly, this mid game dealings didn't work as expected and sometimes didn't even execute when expected. A card would be dealt out and the dealer's counter would decrease, but the receiving player's stack number wouldn't increase meaning that they hadn't received the card. This is most likely due to our Controller State Machine not enabling the stack lines for long enough or at the same time. With this issue in mind we were able to show a valid player play onto the stack during our first turn and also an example of the computer requesting a card. However we weren't able to demonstrate the computer correctly playing a card onto the stack and this was were we lost points in our lab demonstration. This issue could either be from the computer not correctly cycling through and comparing his cards, or intaking multiple cards of the same value due to our controller enabling not correctly syncing with our dealer's stack enable and computer stack enable. With these issues present we also weren't able to demonstrate the end game state of our system and received partial marks for trying.

Overall our demise came from our lack of testing more components at latter stages of the game. Testing all of our components throughout the semester did enable us to have a semi-working game but further testing and tweaking is needed to ensure that later turns in our system don't fail.

Conclusion - Improvements

Our main point of improvement could be in our overarching controller design. After having finished the computer FSM we wanted to place everything together and have a big game controller to enable and control the dealer, player and computer. This caused us to need to place control enable signals and the inputs of all the stacks in our game and put down logic to make sure they were enabled when they needed to be. This started making our design overly complicated and caused us to get caught up in the logic added at all the enable signals of our system. An improvement could be to simplify our system with multiple embedded Finite State Machines. Just as the computer and Dealer state machine where controlled by the controller state machine, we could have made our controller state machine into easier subsections to understand. We could have implemented a Deal_out & Play_game state. Deal_out could have taken cared of dealing out the starting hand and played the first card to the pile while Play_game could have focused on the back and forth repetitiveness of the computer and player playing cards down. Within these components would be other state machines that would control who played & who was suppose to be dealt a card.
Our major components were all working correctly and up until this lab all of our lab demonstrations had worked as intended. This tells us that our fault was when designing our final game controller and by making it too broad. Compartmentalizing and breaking the game into smaller sections of Finite State Machines instead of a long string of states (figure 2) shows that our technique was more prone to flaws and error. Future implementations should take this into account to be able to implement the fully working game onto the FPGA boards.