ECSE 323 – Digital System Design

Crazy-Eights Card Game

g39\_datapath

**Description:**

The data path circuit contains all the necessary components to play the *Crazy-Eights* game on the Altera DE II board.

***Crazy-Eights game:***

The pack:

* Standard 52-card pack

Number of players:

* Human vs Machine (two players)

Objective:

* To finish the hand first.

Game play:

*Game rules:*

* All eights are wild. Thus, an eight of any suit is always legal to play on all other cards and all cards are always legal to play on an eight of any suit.
* A card can be played on another of the same suit.
* A card can be played on another of the same value.

*Setting up the game (Using the dealer’s deck):*

* Deal seven random cards to the human.
* Deal seven random cards to the computer player.
* The dealer then turns up a random card (a starter).

*The play:*

* Starting with the computer, a player may only play one playable card per turn. That is, a card that abides by the *Game rules* above.
* In the case of an absence of such a card, the player must draw one card from the dealer’s deck, passing the turn to the other player.
* The player with an empty hand first wins the game
* If the dealer hand finishes first, then it’s a draw.

**The circuit has the following inputs/outputs:**

All the inputs (except the clock) are user controlled

*resetButton:* 1-bit input to start a new game

*playButton:* 1-bit input to play the selected card from the human’s hand

*drawButton:* 1-bit input to request a card from the dealer (human player draws a card)

*Selector:* 2-bits input to scroll up/down (selects card to play)

*displaySelect:* 4-bits input to choose which value or card to view on the Altera board

*clk:* 1-bit input that controls the clock (50MHz)

All outputs are viewed on the Altera board (7-segment LEDs)

To display cards, we use the following outputs:

*suit:* 7-bit output representing the suit of a card

*face:* 7-bit output representing the face value of a card

To display number of cards or game status indicators, we use the following outputs:

*firstDigit:* 7-bit output representing the 10s of a number

*secondDigit:* 7-bit output representing the 1s of a number

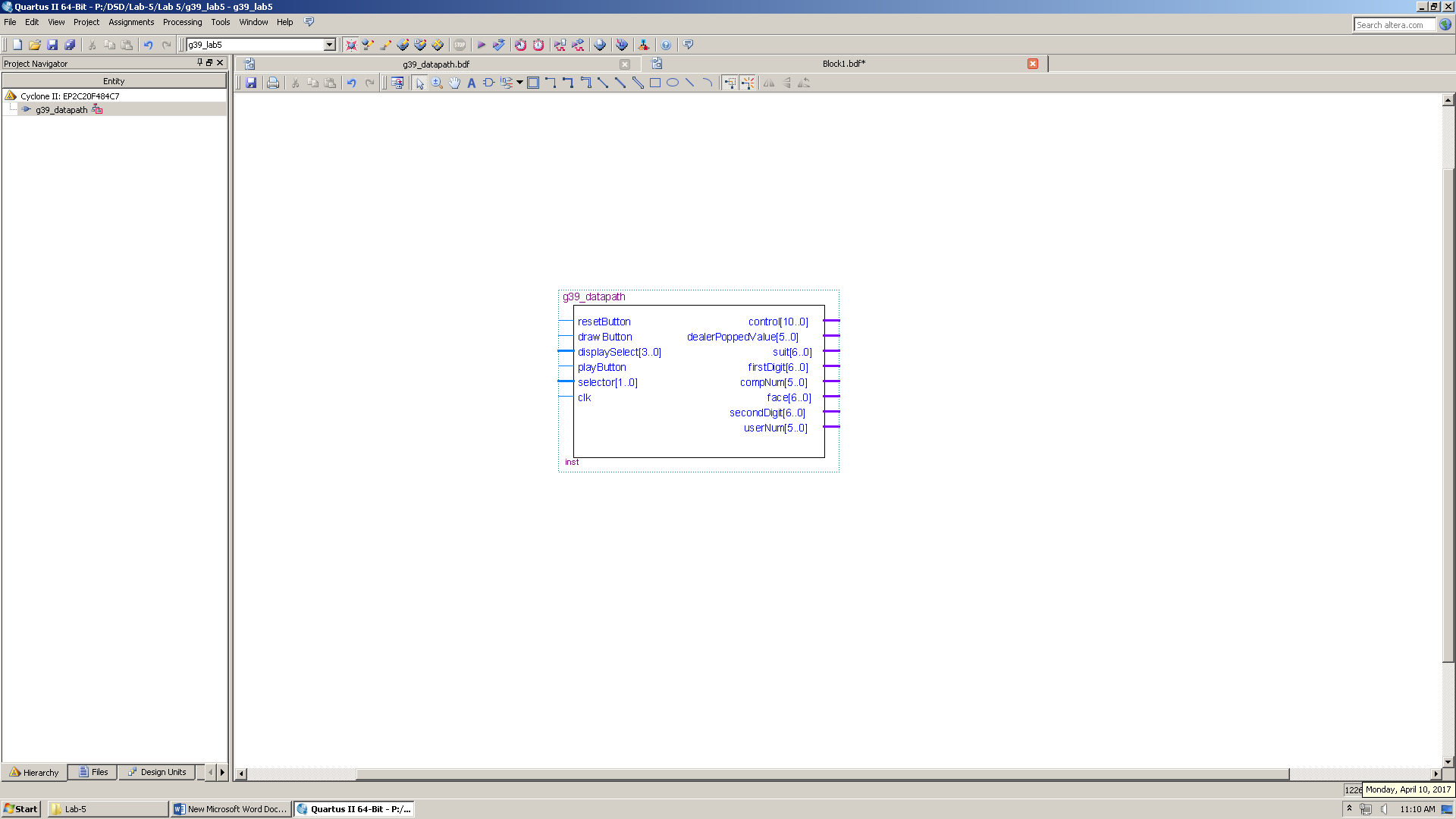


Fig. 1

**Implementation of *g39\_datapath*:**

The data path consists of a large number of sub-circuits listed below:

1. Decks and stacks

* *g39\_playPile:* one instance that represents the play pile in the game play.
* *g39\_stack26:* two instances that represent each player’s hand.
* *g39\_dealerTestbed:* one instance that represents the dealer’s deck.

1. *g39\_numCheck:* checks if a deck contains seven cards, zero cards or otherwise.
2. *g39\_computerFSM:* automates the computer’s turn.
3. *g39\_controlFSM:* a finite state machine that controls the flow of the game.
4. *g39\_UI:* user interface circuit that outputs the selected value to the display
5. *g39\_addressGenerator:* changes the address according to the user selected card.
6. *g39\_rules:* determines whether a given card is playable.
7. Reset circuitry: resets all the components once a new game is requested.

Due to the complexity of the data path, we will provide a detailed description of each of the sub-circuits above.

Sub-circuits that were used in this design were explained in previous reports, those are hyperlinked here:

* [g39\_rules](g39_rules_Report.pdf)
* [g39\_Modulo13](g39_Modulo_13_Report.pdf)
* [g39\_7\_segment\_decoder](g39_7_segment_decoder_Report.pdf)
* [g39\_singlePulse](g39_testbed_report.pdf)
* [g39\_RANDU](g39_RANDU_Report.pdf)
* [g39\_pop\_enable](g39_stack52_report.pdf)
* [g39\_dealerFSM](g39_dealer_Report.pdf)
* [g39\_lab3 (stack52)](g39_stack52_report.pdf)

***g39\_controlFSM:***

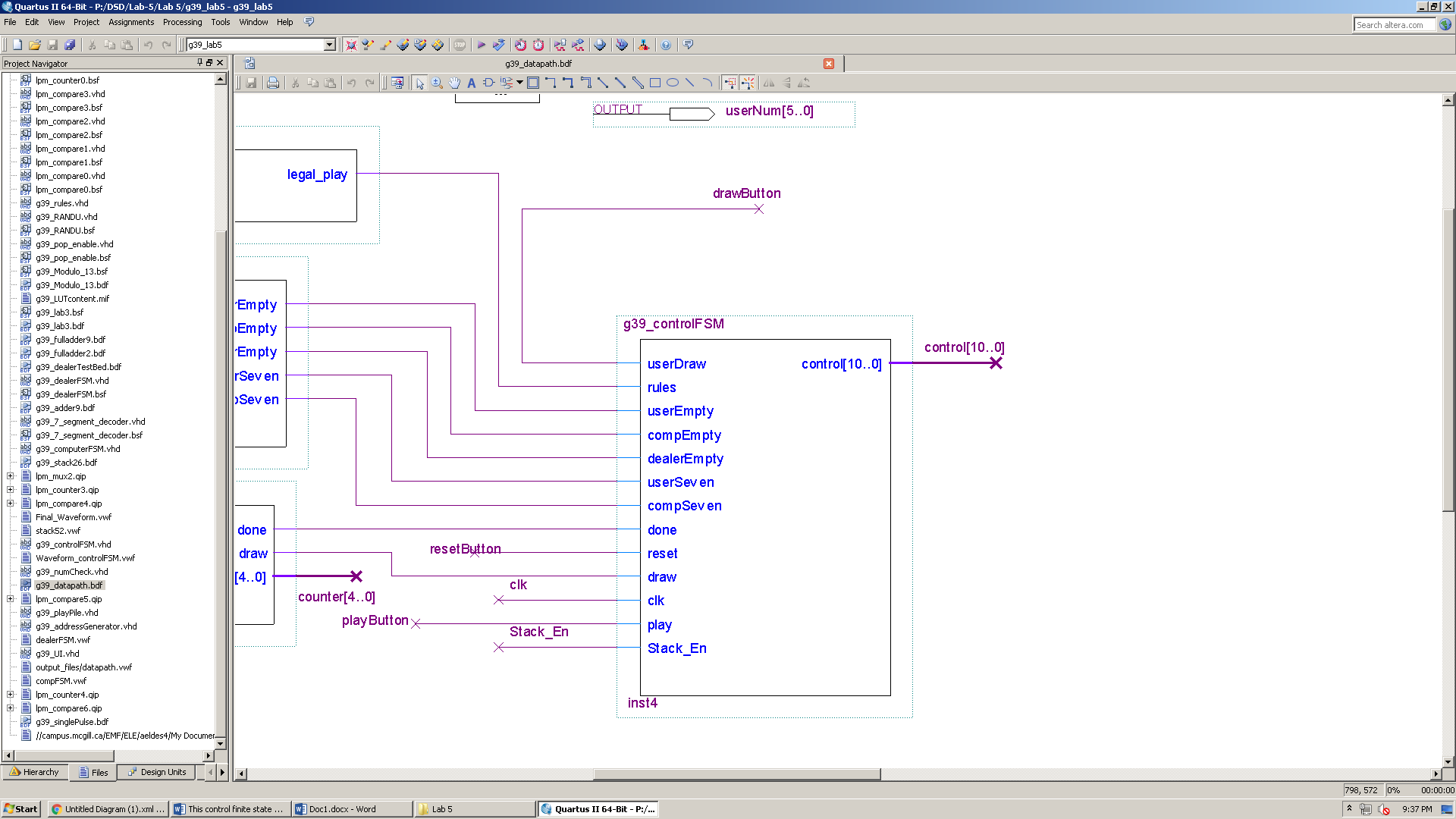
This control finite state machine controls almost every aspect of the game. Generally, the following:

1. From State A up until State H is overhead. This is where the setup of the whole game happens. The game is started by initializing the dealer’s stack. Followed by setting the mode of the dealer’s stack to pop , then dealing the card to respective players , whether it be the user or the computer, 7 times. After that a card is popped from the dealer’s stack and added to a register to hold the “top of the pile” card. What is previously explained takes place only once during a game.
2. Then comes the computer turn to play, which is automated by the *computerFSM* sub-circuit
3. Afterwards, the computer’s decision is checked, if it draws, the dealer’s stack is set to pop and the computer’s to push. If it plays, the computer’s stack is popped and the “top of the pile” card is set to popped value
4. After the computer plays/draws, a check of whether there is a winner or if the game is over takes place
5. Then comes the user’s turn. If the user decides to play, the play is checked to see whether it was valid or not. If not the user is notified and the turn returns to them. If it is valid, then the card is popped from the user’s stack and is added to the “top of the pile” register. If a user decides to draw, a card is popped from the dealer’s stack and is pushed to the user’s deck
6. After the user plays/draws, a check of whether there is a winner or if the game is over takes place
7. Step 2 to 6 are then repeated until there is a winner or the dealer is out of the cards

**The circuit has the following inputs/outputs:**

*userDraw:* 1-bit input for when the user decides to draw card (pushbutton)

*play:* 1-bit input for when the user decides to play a card (pushbutton)

*reset:* input for when the user decides to start a new game (pushbutton)

*userEmpty:* 1-bit input (Output of NumCheck sub-circuit)

*compEmpty:* 1-bit input (Output of NumCheck sub-circuit)

Fig. 2

*dealerEmpty:* 1-bit input (Output of NumCheck sub-circuit)

*userSeven:* 1-bit input (Output of NumCheck sub-circuit)

*compSeven:* 1-bit input (Output of NumCheck sub-circuit)

*done:* 1-bit input (Output of computerFSM sub-circuit)

*draw:* 1-bit input (Output of computerFSM sub-circuit)

*Stack\_En:* 1-bit input (Output of dealerFSM sub-circuit)

*clk:* 50 MGHz clock

control: 11-bit output indicating certain control variables which are explained below

*dealerToPile(10), Request\_Deal(9), turn(8), userMode(7..6), compMode(5..4), dealerMode(3..2), validPlay(1), gameOver(0)*

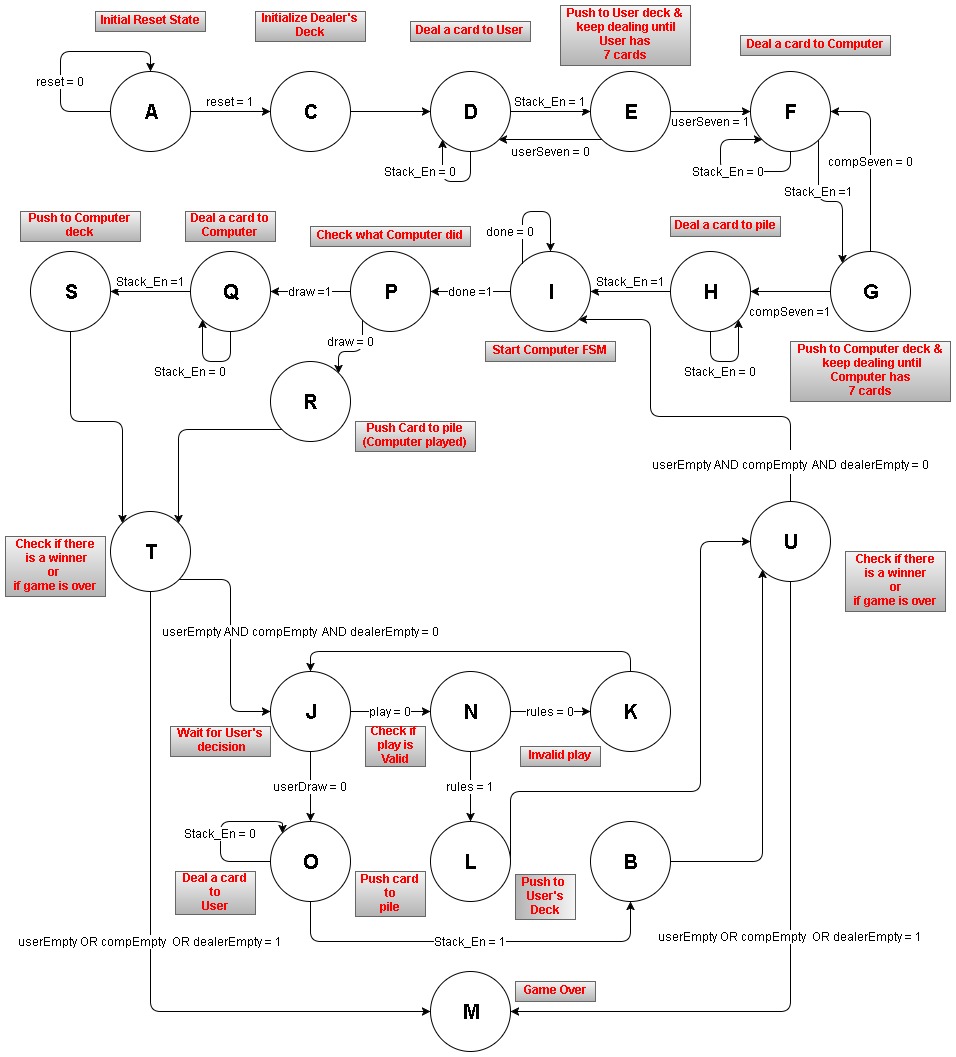


Fig. 3

Note: Whenever popping from dealer’s deck we use Stack\_En as an indicator that the popping is done, since popping in the dealerFSM takes more than 1 clock cycle to finish. This is not the case with initializing

***Stack Operations: 00 🡪 NOP, 01 🡪 PUSH, 10 🡪 INIT, 11 🡪 POP***

***A => control <= “01000000000";***

This is the initial reset state. This state basically gives outputs that do not have any effects on the datapath. For example, all the modes for the decks (stacks) are set to “00” which is NOP mode, and Request\_Deal is set to “1” since it is active low, meaning it also has no effect on the dealerFSM circuit.

***C => control <= "01000001000";***

All this state does is initialize the dealer’s deck and fill it with all 52 cards by setting control(3..2) to “10”, INIT mode, which is then connected to the dealer’s stack.

***D => control <= "00000001100";***

This state pops a card from the dealer’s deck. It does that by setting the mode of dealer’s stack to “11”. It also sets request deal, control(9), to “0” to start the dealerFSM.

***E => control <= "01001000000";***

This state pushes the popped value from the previous state to the user’s deck by setting control(7..6) to “01” which is then connected to the user’s stack.

***F => control <= "00000001100";***

This state pops a card from the dealer’s deck. It does that by setting the mode of dealer’s stack to “11”. It also sets request deal, control(9), to “0” to start the dealerFSM.

***G => control <= "01000010000";***

This state pushes the popped value from the previous state to the computer’s deck by setting control(5..4) to “01” which is then connected to the computer’s stack.

***H => control <= "10000001100";***

This state pops a card from the dealer’s deck. It does that by setting the mode of dealer’s stack to “11”. It enables the pile by setting control(10), dealerToPile, to “1”. It also sets request deal, control(9), to “0” to start the dealerFSM

***I => control <= "01100000000";***

This state changes the turn to indicate the computer’s turn, by setting control(8) to “1” which is then connected to the computerFSM

***P => control <= "01000000000";***

This is a check state that checks whether to computer played or drew a card.

***Q => control <= "00000001100";***

Reaching this state means the computer decided to draw a card. It sets request deal, control(9), to “0” and the mode of the dealer’s stack, control(3..2), to “11”.

***R => control <= "01000110000";***

Reaching this state means the computer decided to play. It sets the computer’s mode, control(5..4), to “11”, popping the card and triggering the playPile sub-circuit to register the value.

***S => control <= "01000010000";***

After the computer decides to draw a card and a card is popped from the dealer’s deck, at this state, the card is pushed to the computer’s deck. This is done by setting the computer’s stack mode, control(5..4), to “01”

***T => control <= "01000000000";***

This is a check state if a player won, or if the dealer is out of card. It determines whether or not we should proceed to the end state, state M.

***J => control <= "01000000000";***

This is a wait state. All it does is wait for the player’s decision

***N => control <= "01000000000";***

This is a check state, it can be thought of as a wait state as well. It uses the rules input to this circuit to determine which state to proceed to.

***K => control <= "01000000000";***

This state indicates an invalid play by setting validPlay, control(1), to “0”

***L => control <= "01011000010";***

This state indicates a valid play. It does so by setting control(9) to “1”. It also pops a card from the user’s deck by setting control(7..6) to “11”, triggering the playPile sub-circuit to register the value.

***O => control <= "00000001100";***

This state pops a card from the dealer’s deck. It does that by setting the mode of dealer’s stack to “11”. It also sets request deal, control(9), to “0” to start the dealerFSM.

***B => control <= "01001000000";***

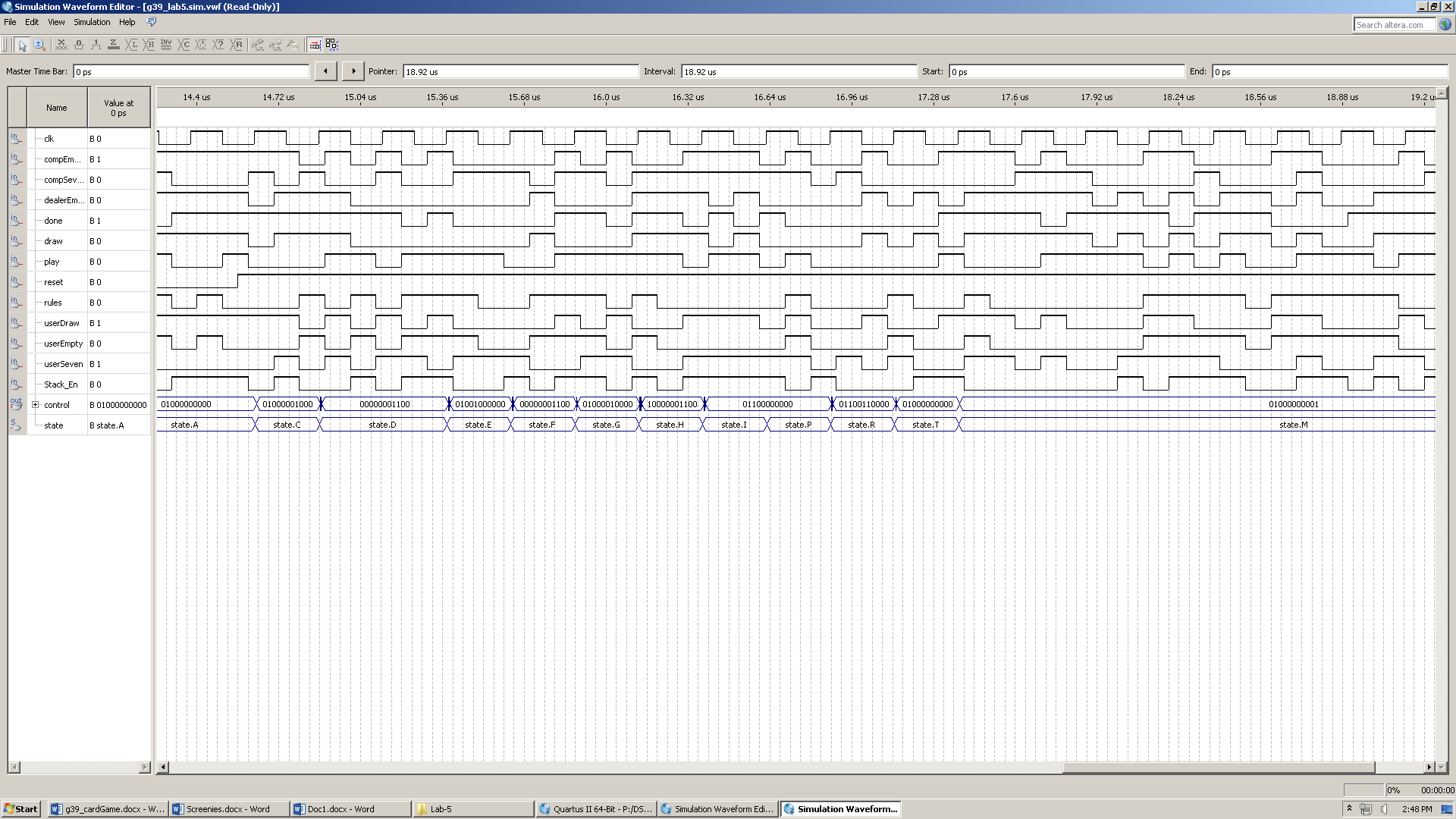
This state pushes the drawn card to the user’s deck by setting the mode of the user’s stack, control(7..6), to “01”.

***U => control <= "01000000000";***

This is a check state if a player won, or if the dealer is out of card. It determines whether or not we should proceed to the end state, state M.

***M => control <= "01000000001";***

This is the final state. This state is reached if a player wins or if the dealer is out of cards. It sets the gameOver output, control(0), to “1” to indicate that a game is over.

**Simulation:**

***g39\_addressGenerator:***

Fig. 4

This sub-circuit takes the button presses of the user and generates an address for the rest of the sub-circuits to use.

*selector*: 2-bit input, selector(1) is a toggle switch (to either go up or down), and selector(0) is a push button to perform the increment/decrement

*reset*: 1-bit input that resets the circuit

*userAddress*: 6-bit output/buffer that represents the address the user is selecting from the user’s deck/stack

*clk*: 50 MGHz clock

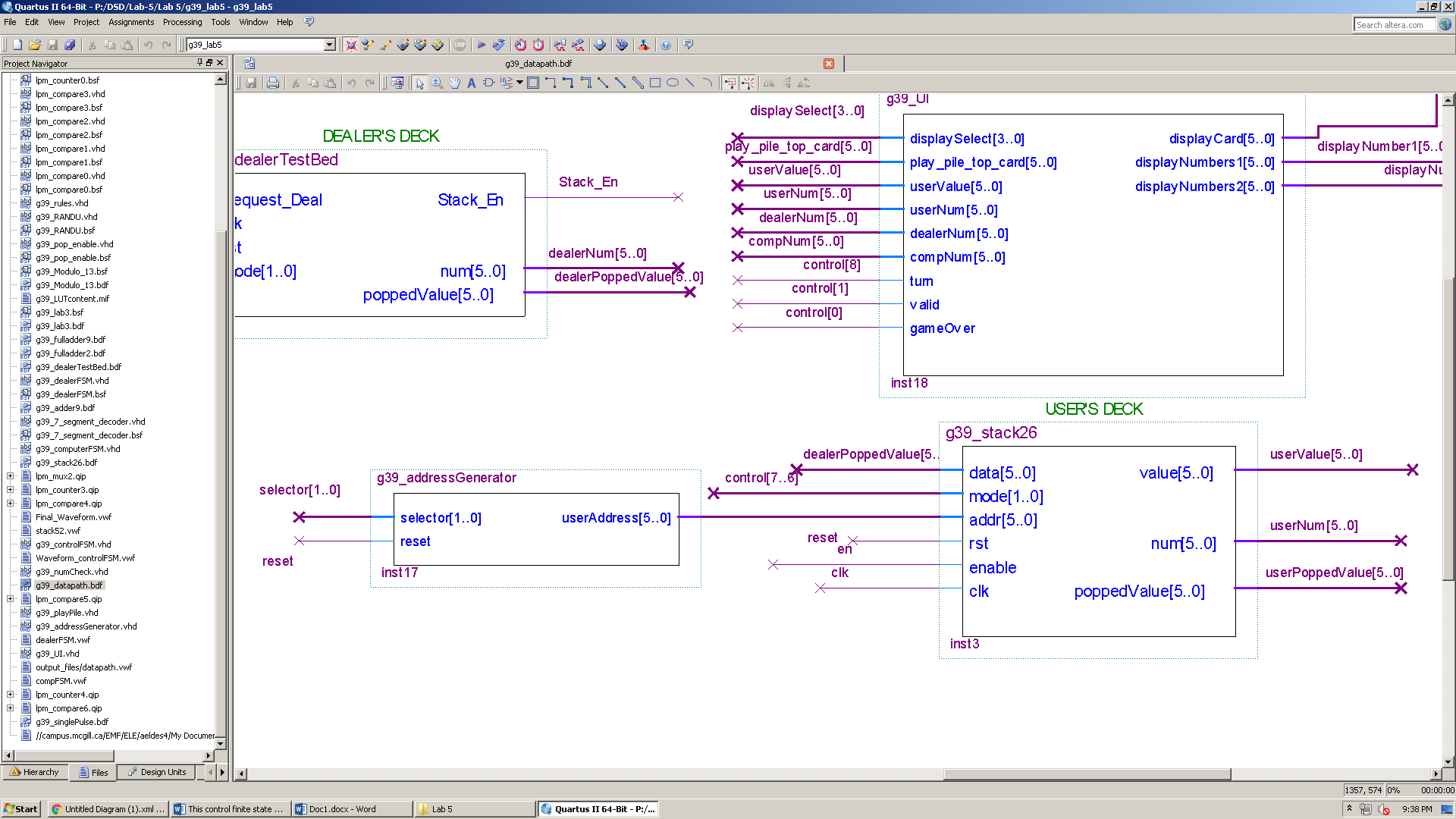


Fig. 5

There are 3 scenarios that this circuit takes care of:

1. *Selector* = “00” AND userAddress > "000000": With these conditions, the circuit decrements the userAddress
2. *Selector* = “10” AND userAddress < "011010": With these conditions, the circuit increments the userAddress
3. *Otherwise,* the circuit retains the previous the userAddress

**Simulation:**

Fig. 6

***g39\_stack26:***

In order to reduce the use of Embedded Array blocks on the Altera board and since each player can have a maximum of 26 cards, we created another stack element that is exactly the same as the one described above but with 26 elements.

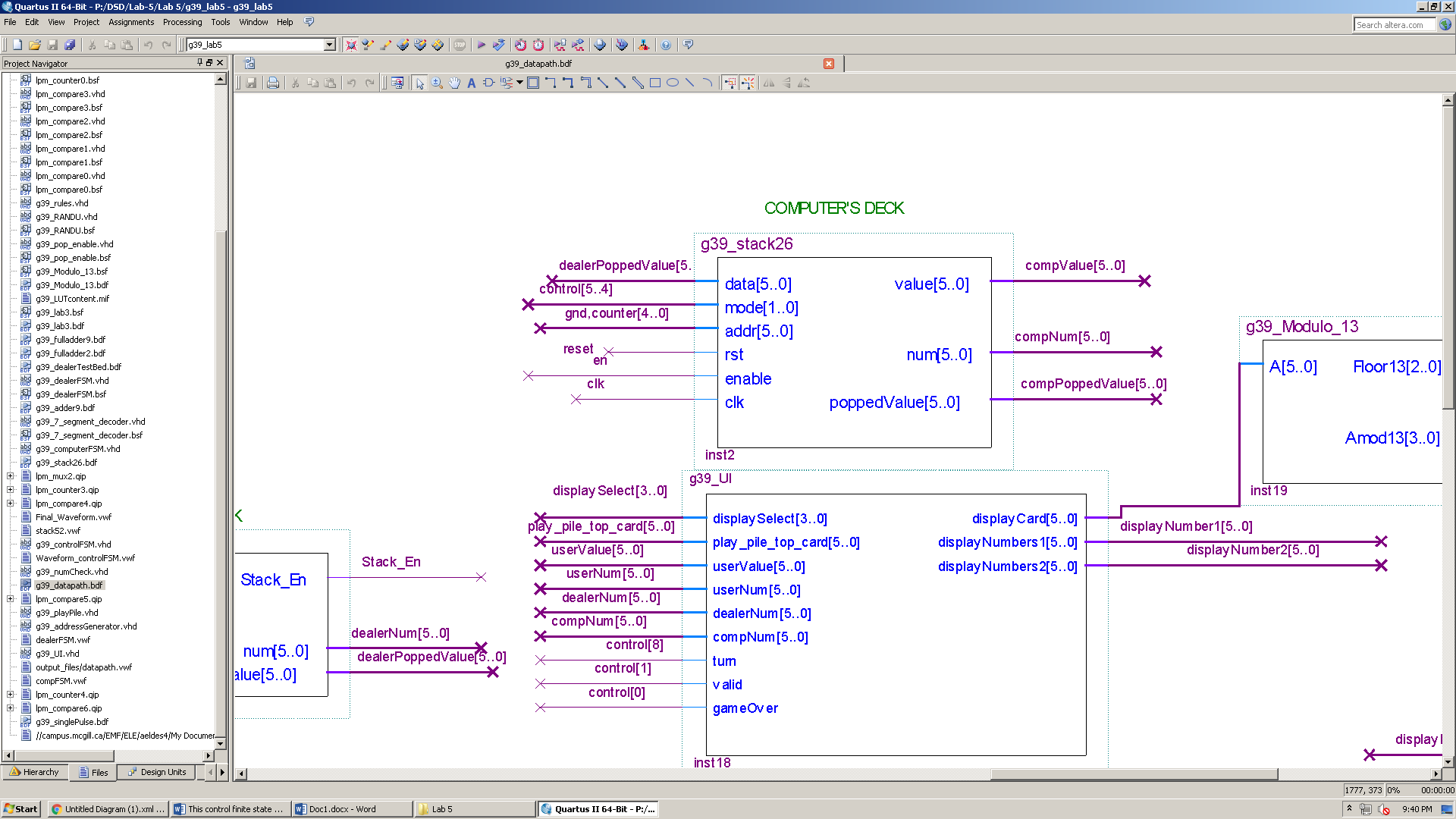


Fig. 7

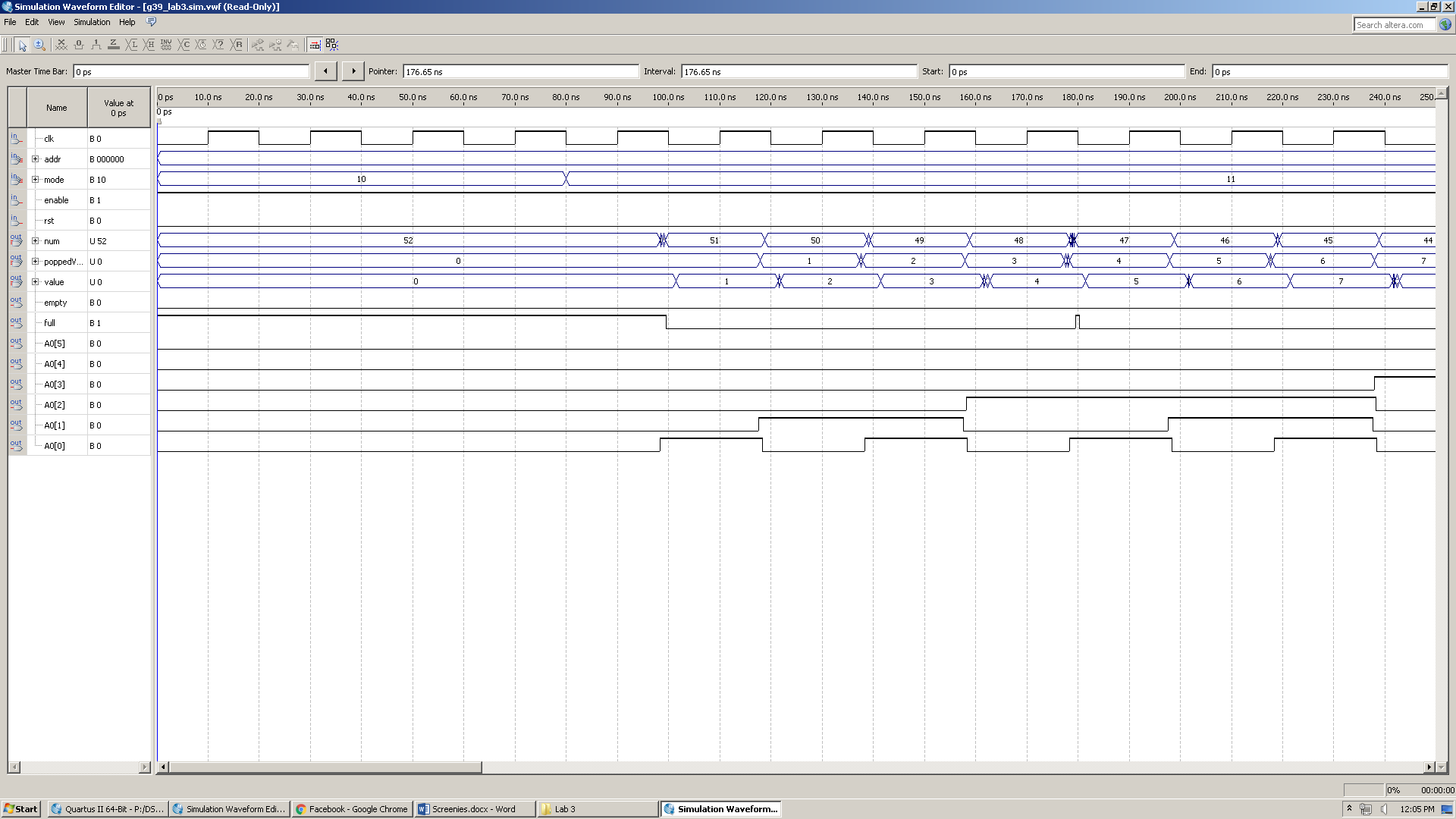
**Simulation****:**

Fig. 8

***g39\_computerFSM:***

This circuit is responsible for the computer’s turn, whether it is playing or drawing. When turn becomes “1” it is the computer’s turn, it then moves from the wait state, state A, to state B, the scanning state. Afterwards, it starts to scan the addresses by utilizing a counter, this is a loop between state B and C and it exits in only one of two cases:

1. A valid card was found, and so the FSM proceeds to state E which indicates playing a card
2. No valid card was found and the counter is equal to compNum (number of cards in the computer deck) which makes the numCheck input be “1” by the comparator, and the FSM proceeds to state D, drawing a card.

After either state, E or D, the computer proceeds to state A, the wait state, again. It waits for the user to finish their turn and for the g39\_controlFSM to signal the start of another computer turn.

**This circuit has the following inputs/outputs:**

*turn*: 1-bit input indicating the start of the computer’s turn (Output of g39\_controlFSM)

*rules*: 1-bit input indicating a valid play (Output of g39\_rules)

*numCheck*: 1-bit input indicating the computer has no valid card to play (Output of comparator, shown below)

*clk*: 50 MGHz clock

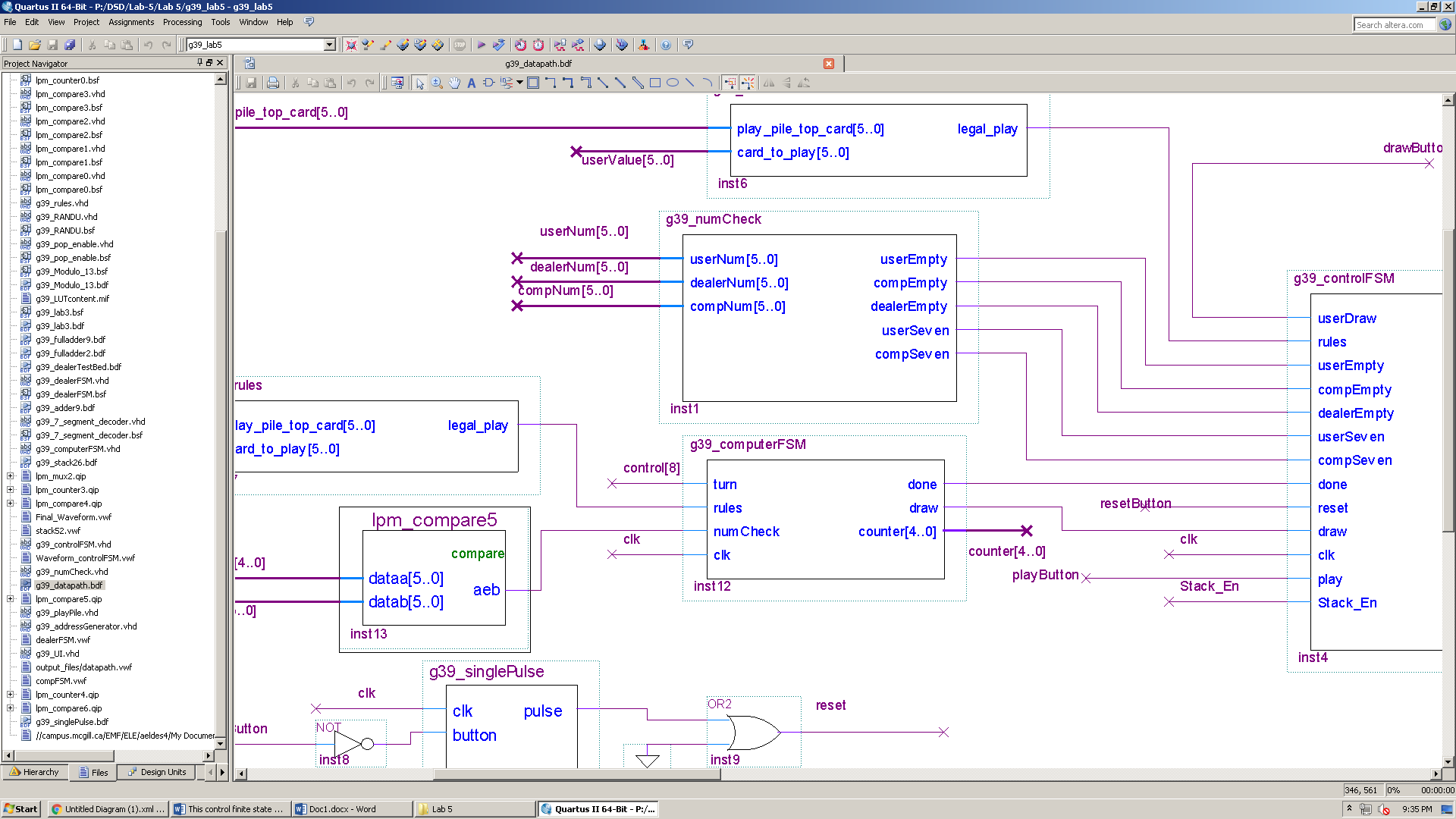
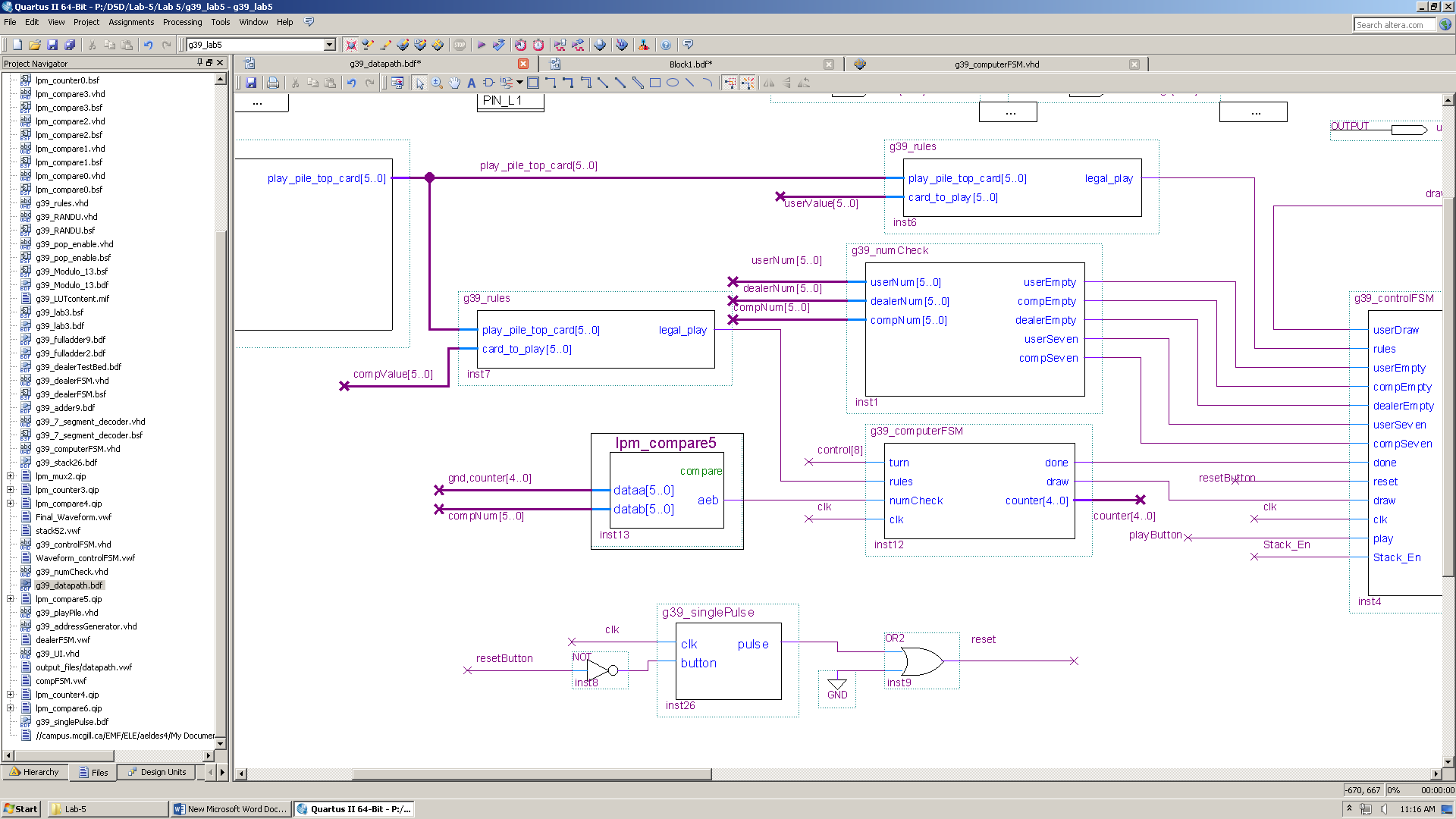
*done*: 1-bit output indicating the computer is done playing

*draw*: 1-bit output indicating the computer decided to draw

*counter*: 1-bit buffer used in the comparator to ensure all cards are scanned

Fig. 10

Fig. 9



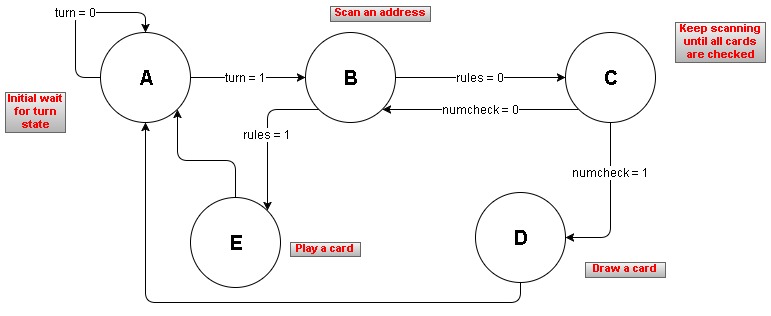


Fig. 11

Outputs: *done, draw, counter*

State A: *done = ‘0’, draw = ‘0’, counter <= "00000"*

State A is a wait state. Once turn becomes ‘1’ by the g39\_controlFSM, it moves to state B, the scanning state. It also resets the counter to 0.

State B: *done = ‘0’, draw = ‘0’, counter <= counter*

At state B, the counter is retained and a card is scanned.

State C: *done = ‘0’, draw = ‘0’, counter <= counter + 1*

At state C, the counter is incremented, and if it is not the last card in the deck (indicated by g39\_numCheck) then it moves back to state B.

State D: *done = ‘1’, draw = ‘1’, counter <= "00000"*

State D indicates a card is drawn and so sets the done and draw signals to ‘1’.

State E: *done = ‘0’, draw = ‘0’, counter <= "00000"*

State E indicates a card is played and so sets the done signal to ‘1’ and the draw signals to ‘0’.

Note: We are using a convention in which a draw signal of ‘0’ from the g39\_computerFSM, indicates the computer decided to play a card, rather than having another signal called “play”.

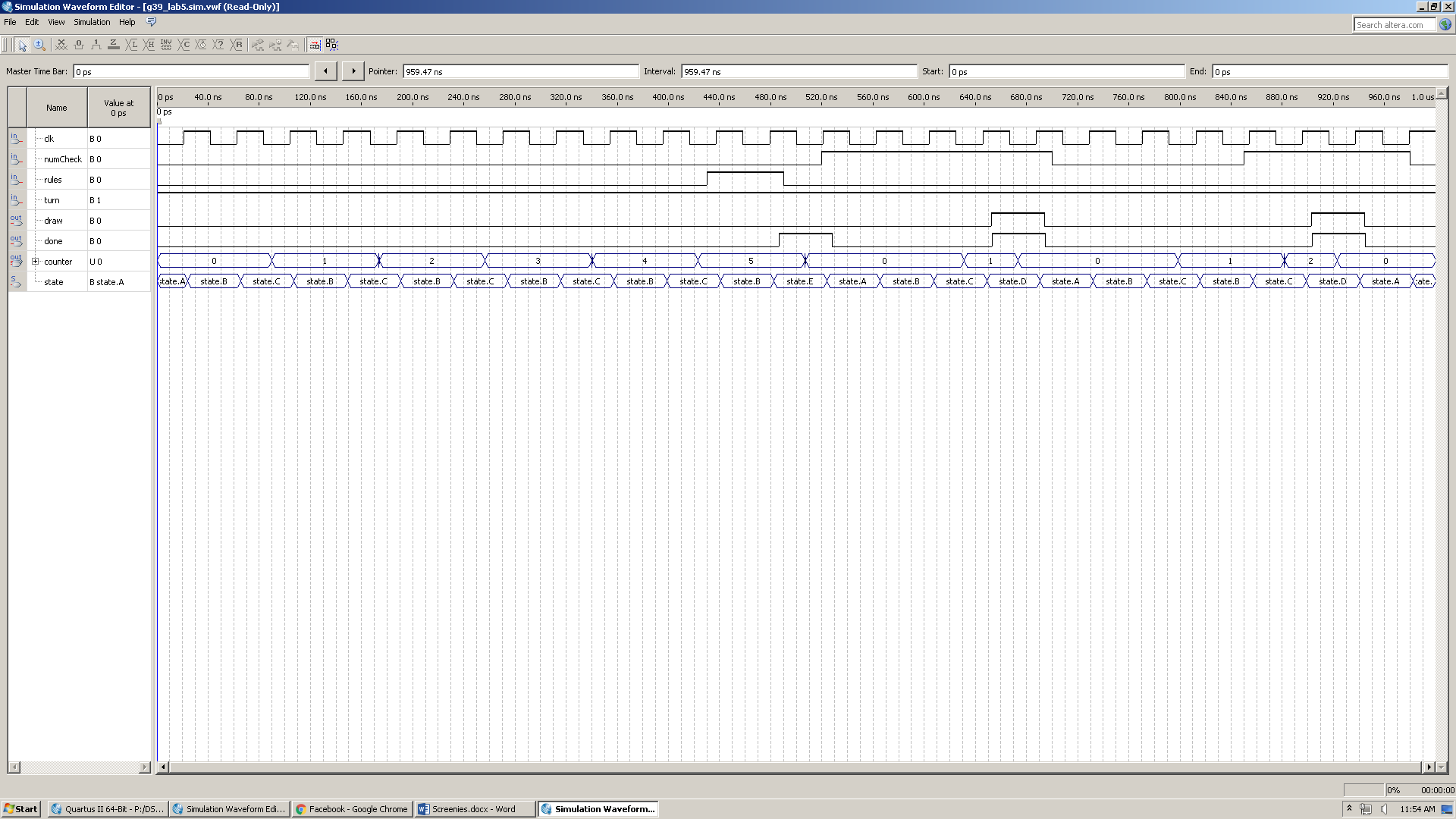
**Simulation:**

Fig. 12

***g39\_dealerTestbed:***

This circuit takes care of the dealer’s deck. It was explained in detail in g39\_dealer\_Report (done in Lab 4).

Some changed were made to the testbed and its sub-circuits in order to fit in the data path.

1. g39\_dealerFSM sub-circuit:

When the finite state machine was first implemented in Lab 4, its input was a pushbutton. This is why we needed a “buffer” state to wait until the pushbutton stabilizes. However, this is not the case in the data path. Request\_Deal input is derived by *g39\_controlFSM*. Therefore, the buffer state was removed and RAND\_Enable is asserted one clock cycle after requesting a deal.

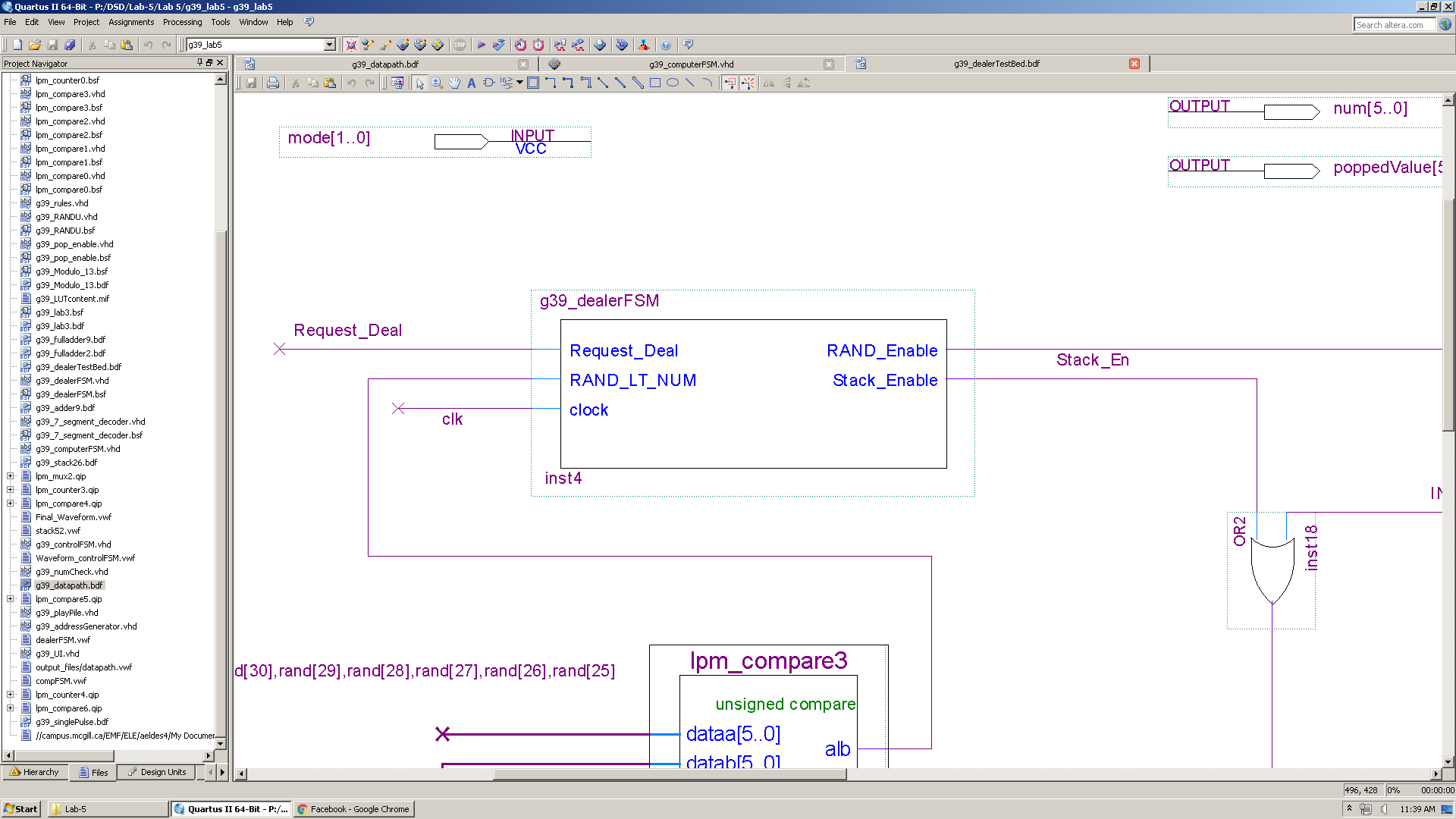
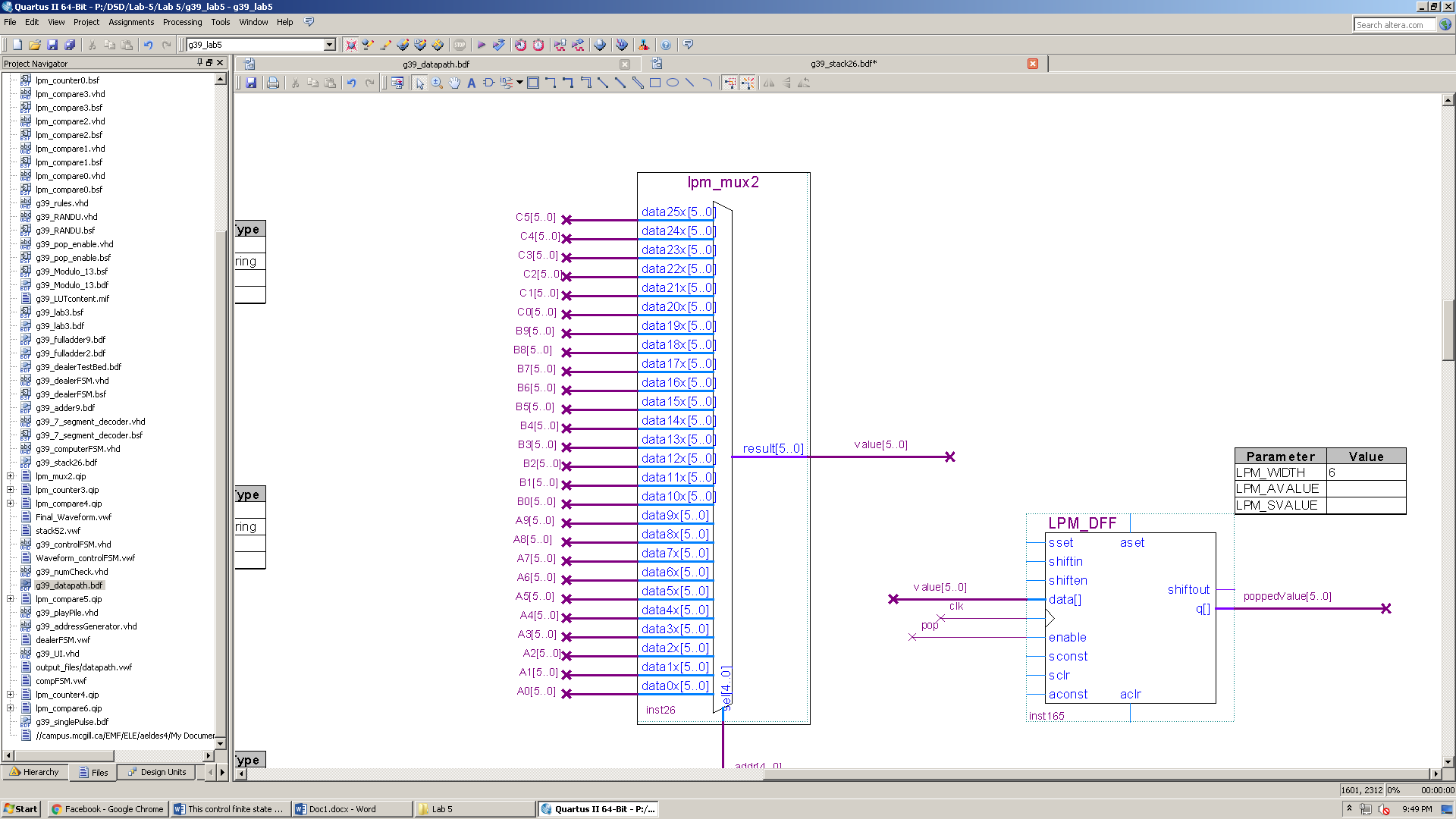


Fig. 13

1. g39\_lab3 sub-circuit (Stack52):

Stack52 was first introduced in g39\_stack52\_report (done in Lab 3). However, the way it was implemented did not allow to save a popped value. Instead, it overrides the value at the popping address with the one in the next location, and so on. Dealing a card in this situation is problematic. It is equivalent to removing a card from the dealer’s deck and dealing the next one without removing it. To solve this, we use a D-FF that’s enabled whenever we pop. This way we deal the popped value and remove it from the deck.

1. Changes in the testbed:
   * Modulo and 7-segment decoder circuits were removed since we don’t output anything directly from the dealer’s testbed.
   * Reset is now done when rst = 1

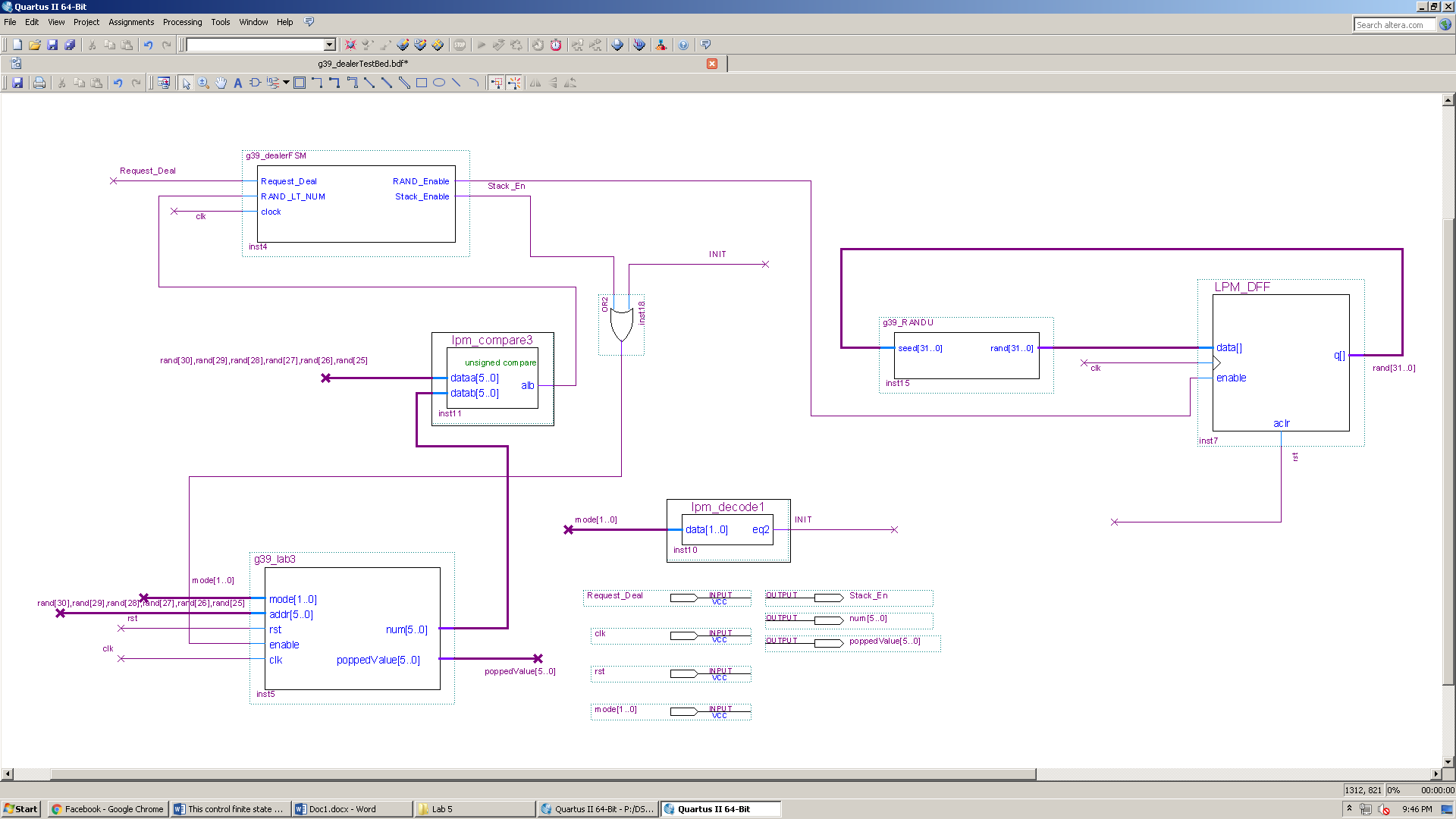


Fig. 14

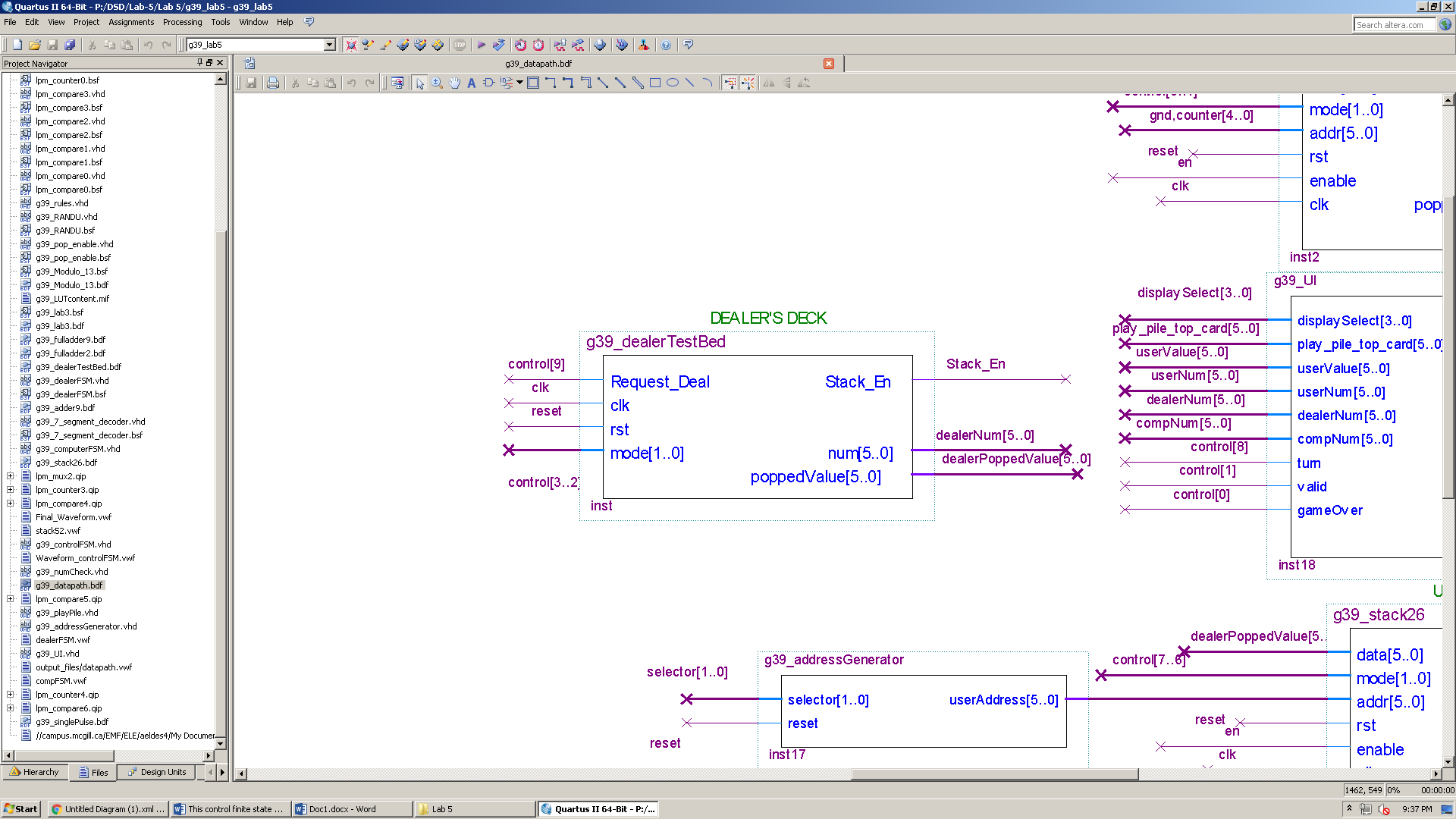


Fig. 15

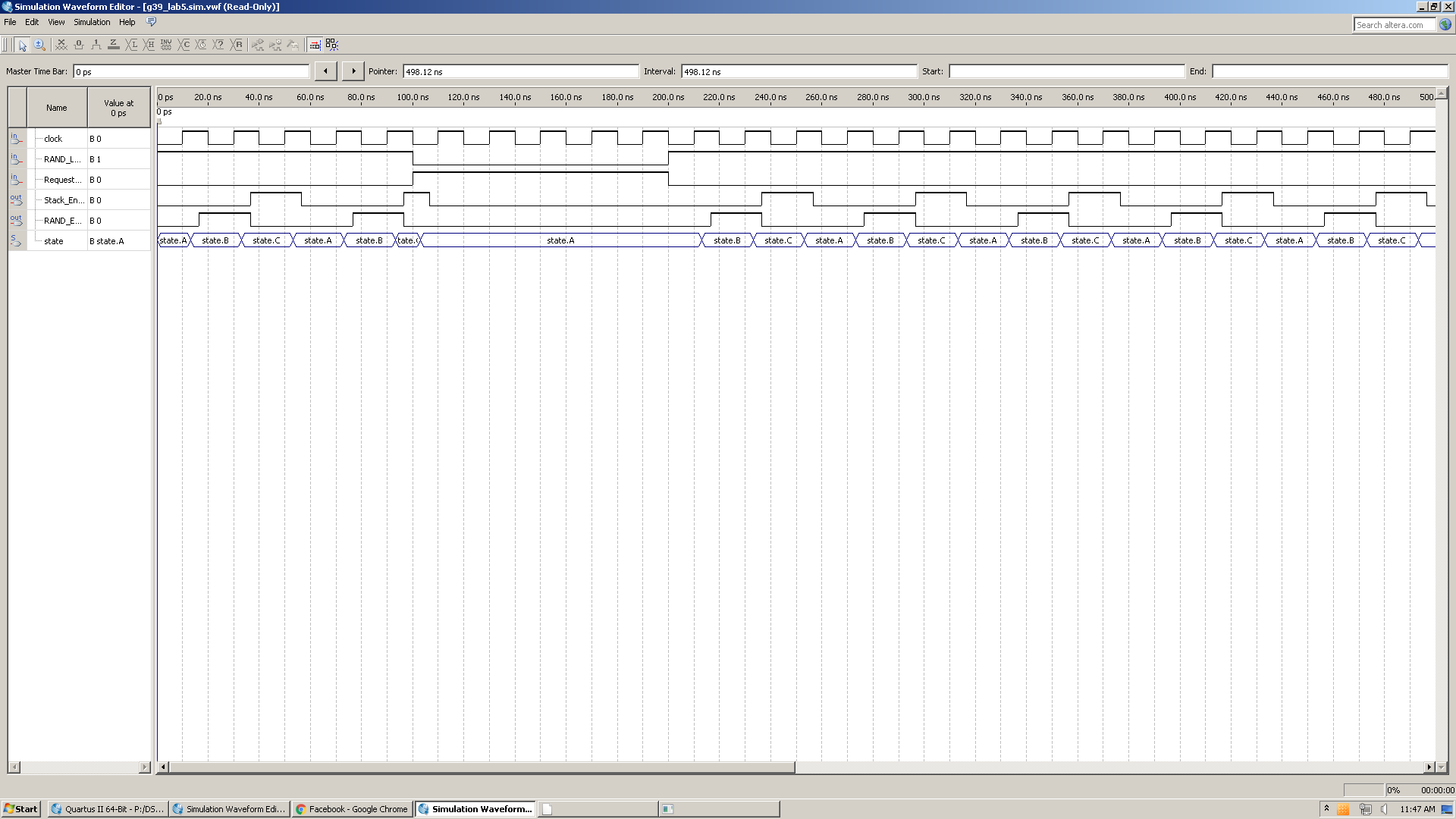
**Simulation:** 

Fig. 16

***g39\_numCheck:***

This circuit is needed for two reasons:

* Determine if a game has ended. This happens if a player finishes their hand or if the dealer has no more cards to deal.
* Determine when to stop dealing cards at the beginning of a game. This is the case when a player has seven cards in their hand.

**The circuit has the following inputs/outputs:**

*userNum, dealerNum, compNum:* 6-bit inputs, the number of cards in the respective stacks.

*userEmpty, compEmpty, dealerEmpty, userSeven, compSeven:* 1-bit outputs representing the status of the respective stacks.

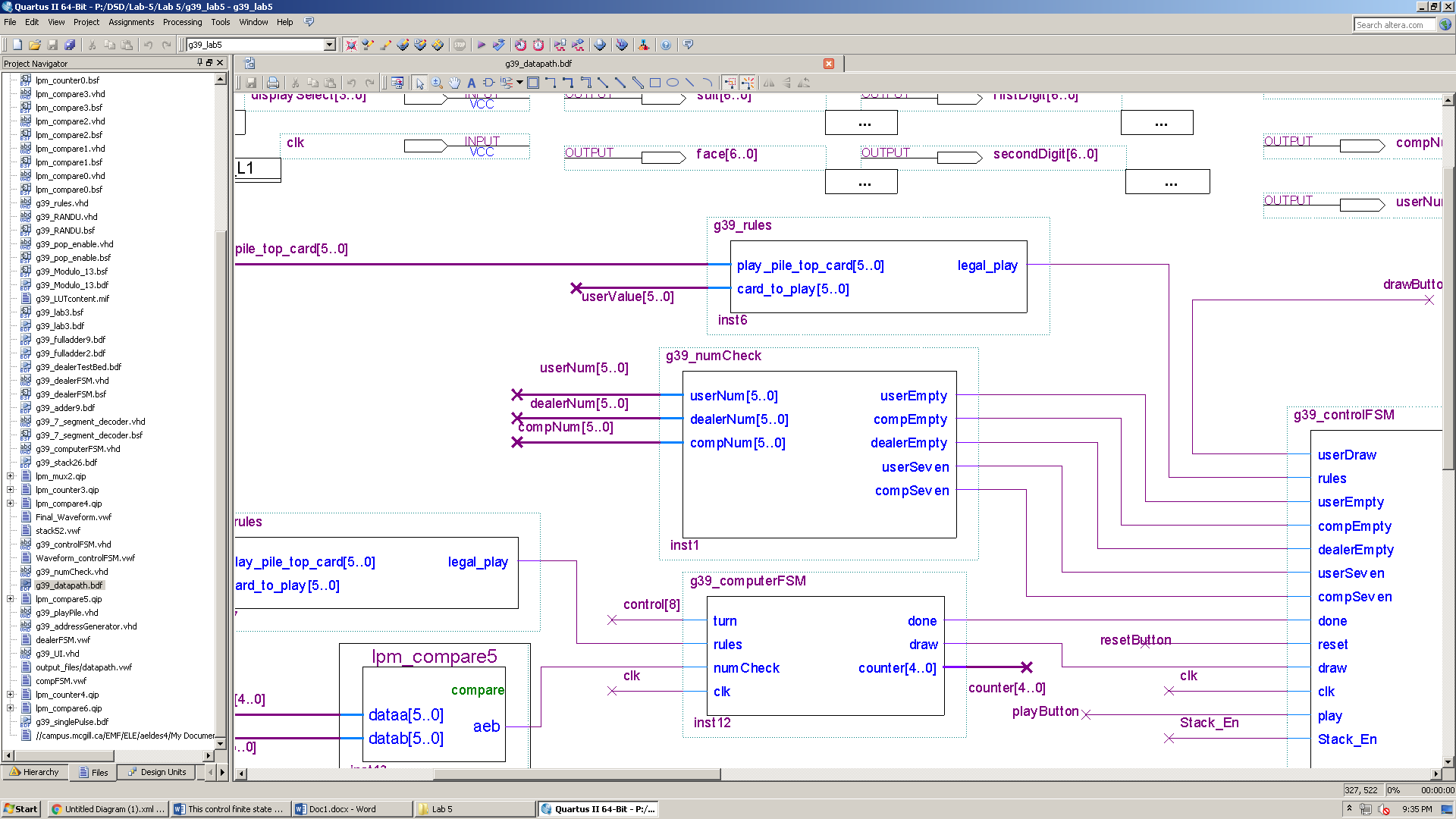


Fig. 17

The circuit takes its inputs from the players’ stack26 and from the dealerTestbed and does the following:

* userEmpty = ‘1’ if the human player has an empty hand.
* compEmpty = ‘1’ if the computer’s hand is empty.
* dealerEmpty = ‘1’ if the dealer’s deck is empty.
* userSeven = ‘1’ if the human player has seven cards.
* compSeven = ‘1’ if the computer has seven cards.

The outputs of numCheck is then fed to the controller to determine the end of a game, or the end of dealing cards.

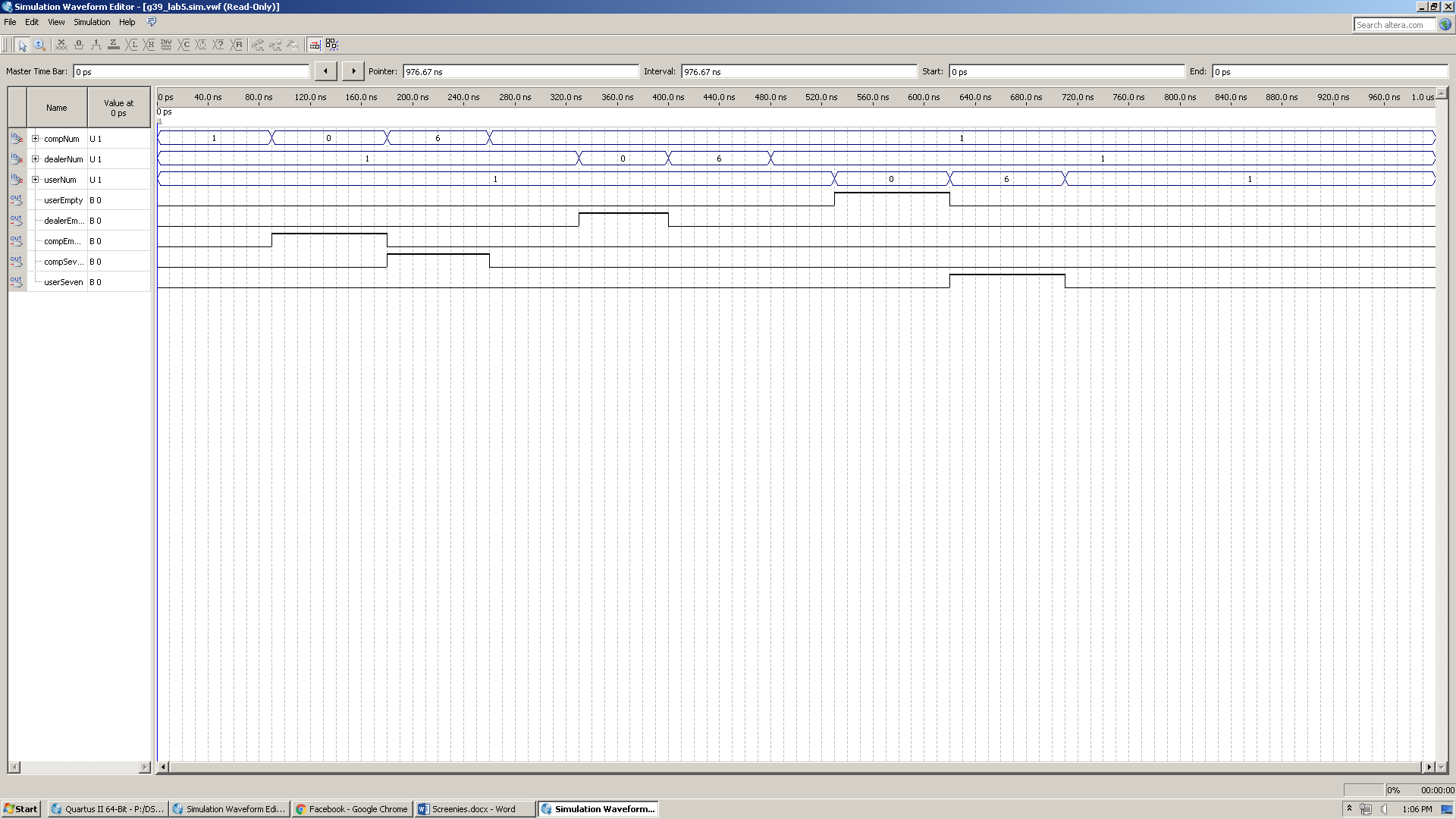
**Simulation:**

Fig. 18

***g39\_playPile:***

As mentioned in the lab manual, a stack component is not needed for the play pile since only the top card is relevant to the play. For this, we use a buffer to hold the top card. The value is changed under three conditions:

1. if a new game is requested, play pile card is reset to 0.
2. When the dealer puts a starter card at the beginning of a game, play pile updates accordingly.
3. if a player plays a valid card, play pile updates its value to that card.

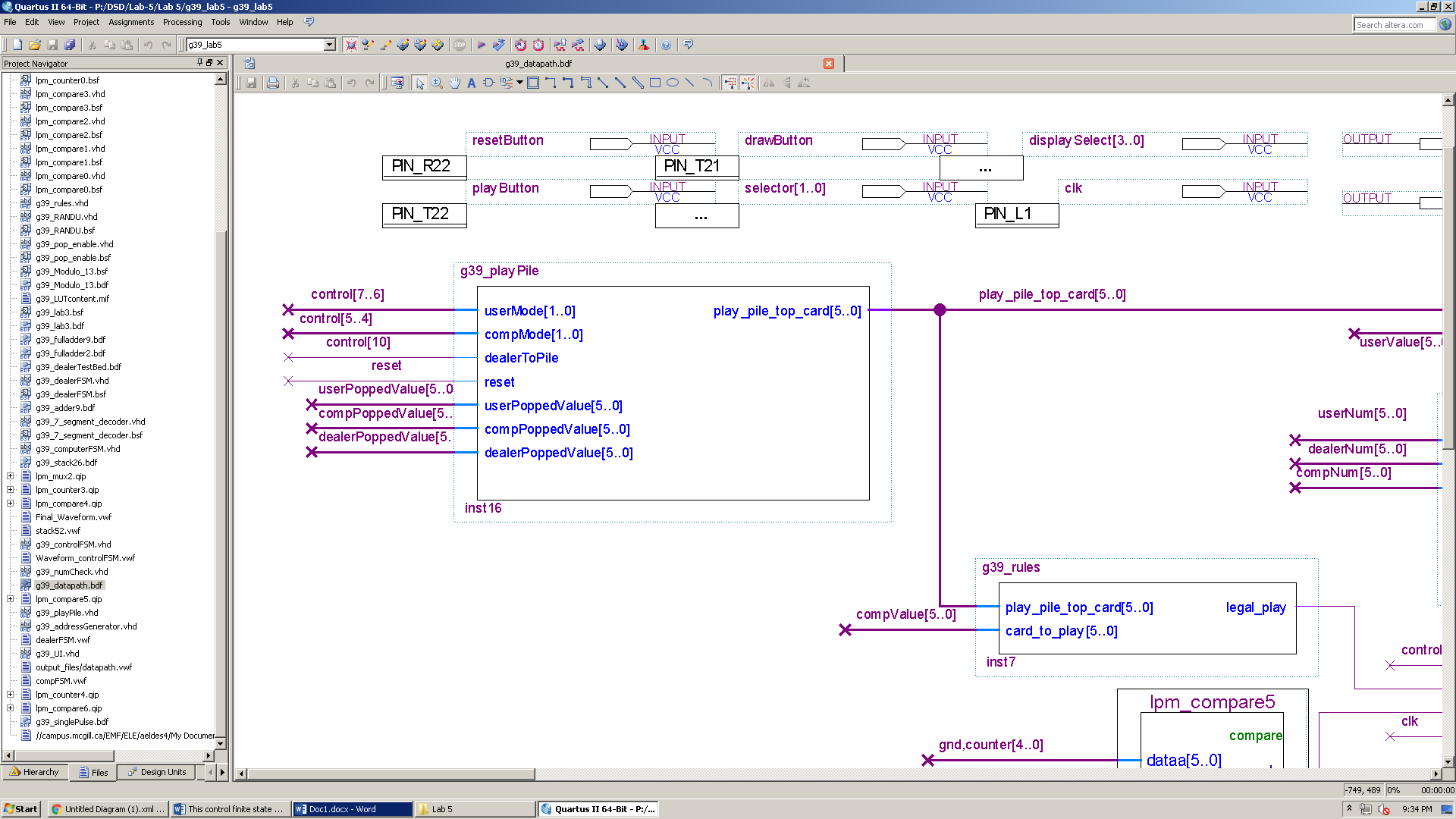
g39\_playPile was implemented in VHDL.

**The circuit has the following inputs/outputs:**

*userMode, compMode:* 2-bits inputs representing the modes of both player’s stacks (decks)

These are derived by *g39\_controlFSM*

*userPoppedValue, compPoppedValue, dealerPoppedValue*: 6-bits inputs representing the card to play (either player) or draw (from the dealer)



*dealerToPile:* 1-bit input

Fig. 19

*reset:* 1-bit input

*play\_pile\_top\_card:* 6-bit output

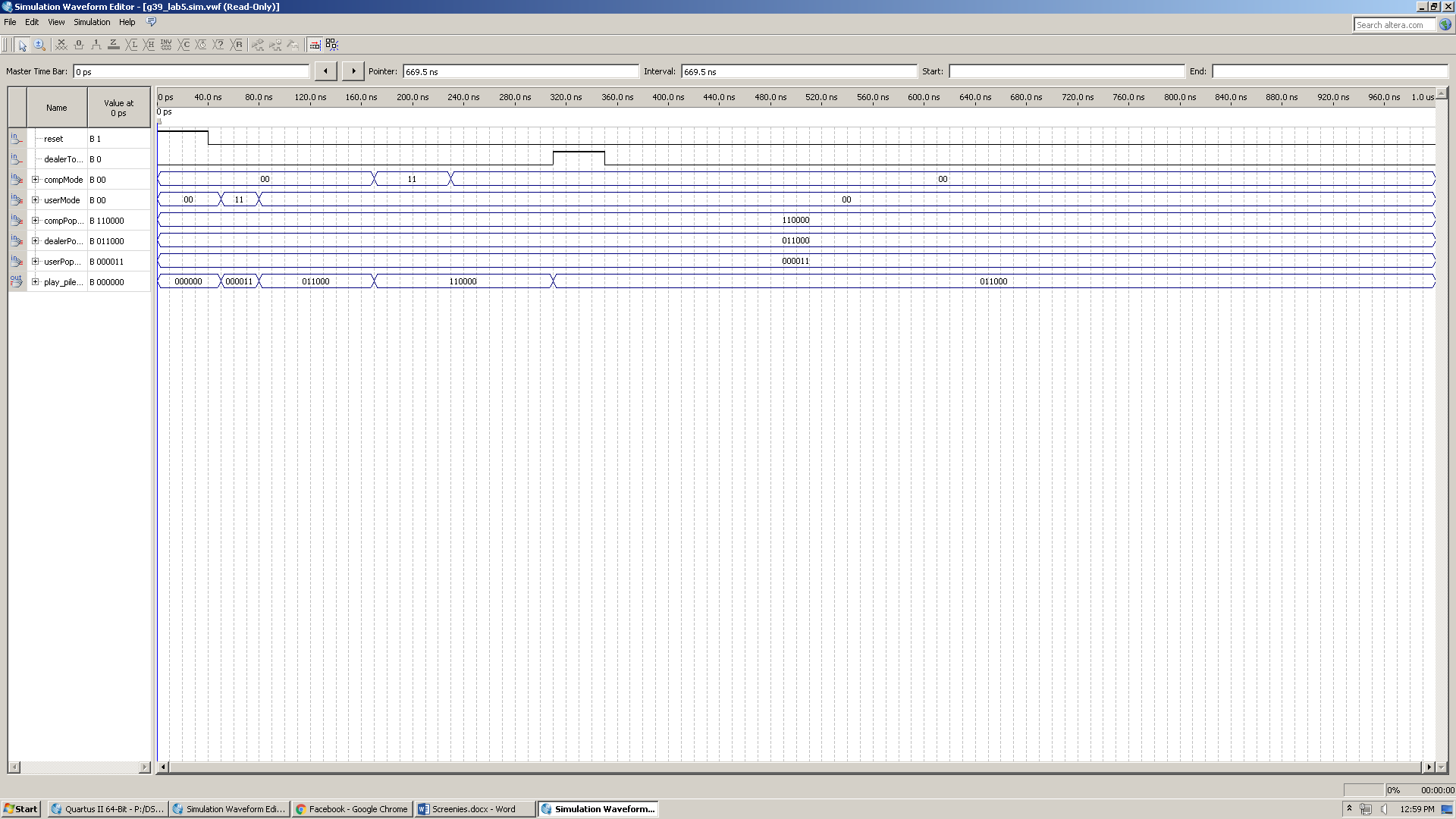
**Simulation**

Fig. 20

***g39\_UI:***

The circuit is an interface to the user. This is the only way for the user to communicate with the system and play the game.

**The circuit has the following inputs/outputs:**

*displaySelect:* 4-bits user input to select which value to display

*play\_pile\_top\_card:* 6-bits input representing the pile top card

*userValue:* 6-bits input representing the user’s card (at the user selected address using g39\_addressGenerator)

*userNum, dealerNum, compNum:* 6-bit inputs representing the number of cards in the respective stacks.

*turn, valid, gameOver:* 1-bit inputs indicating the status of the game (derived from the controller)

*displayCard:* 6-bit output representing the card to display

*displayNumbers1:* 6-bit output representing the first digit of the number/indicator to display

*displayNumbers2:* 6-bit output representing the second digit of the number/indicator to display

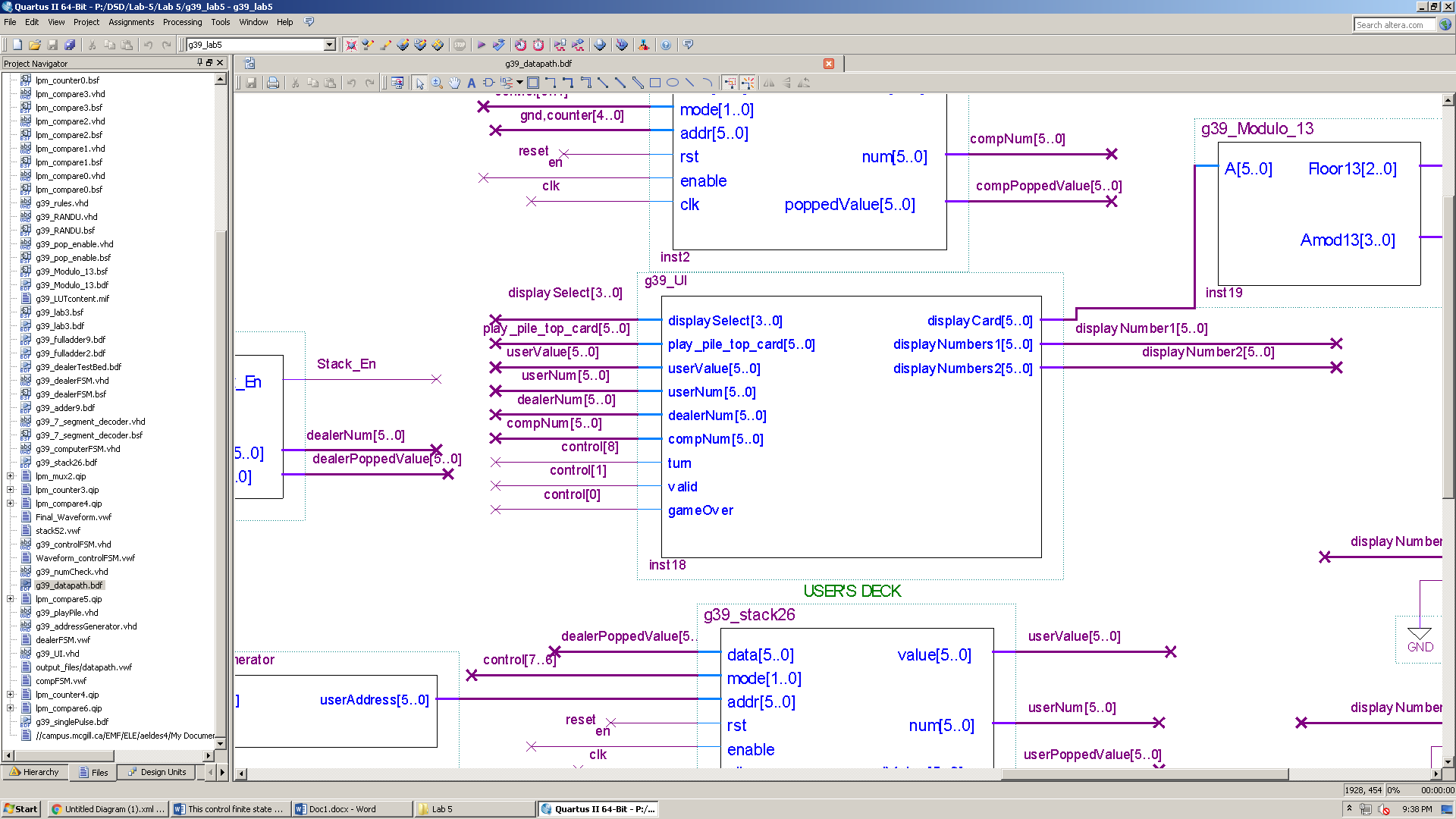


Fig. 21

Since we have 9 possible selections, we need a 4-bits *displaySelect* input (4 toggle switches on the Altera board). The UI displays either a card or a number/indicator.

* For a card to be correctly display, we need a modulo13 circuit and two 7-segment decoders one for the suit (mode 0) and one for the face\_value (mode 1).
* To display a number or an indicator (which is represented by either a 1 or a 0) we don’t need a modulo13 (which decodes the value of a card). Instead, split the number into two digits (because 7-segment decoder displays one digit at a time) by dividing by 10. The two digits are then fed to two 7-segment decoders both with mode = 0.

Therefore, we use all four 7-segment displays on the Altera board.

Some design choices were made to make it easier for the user to interpret the displayed values:

* When *displaySelect* is 0 or 1, we display cards. So, we made the two number/indicator 7-segment LEDs display zeros. This is done by making *displayNumbers1* and *displayNumbers2* = “000000”.
* For *displaySelect* is 2 to 8, the user wants to display a number or an indicator. So, we made the two card 7-segment LEDs display zeros. This is done by making *displayCard* output “001001”. When feeding “001001” to our modulo 13 its outputs are Amod13 = “1001” and floor13 = “000”. Since Amod13 is fed to a 7-segment decoder with mode 1, it outputs zeros. Similarly, the second 7-segment decoder outputs zeros since its mode is 0 and floor13 is zeros.
* When displaying an indicator (turn, valid, game over), *displayNumbers1* has no use since these are either 1 or 0 (*displayNumbers2* is sufficient). So, in order for the user to distinguish between the different indicators, *displayNumbers1* is used.
  + When displaying *trun* (*displaySelect* = “00101”), *displayNumbers1* displays A. This is done by making *displayNumbers1* = “001010”.
  + When displaying *valid* (*displaySelect* = “00110”), *displayNumbers1* displays B. This is done by making *displayNumbers1* = “001011”.
  + When displaying *gameOver* (*displaySelect* = “00111”), *displayNumbers1* displays C. This is done by making *displayNumbers1* = “001100”.
* When displaying *winner* indicator (*displaySelect* = “01000”):
  + If draw, display dd. This is done by setting both *displayNumbers1* and *displayNumbers2* to “001101”.
  + If computer won, display F1. This is done by setting *displayNumbers1* to “001111” and *displayNumbers2* to “000001”.
  + If user won, display F0. This is done by setting *displayNumbers1* to “001111” and *displayNumbers2* to “000000”.

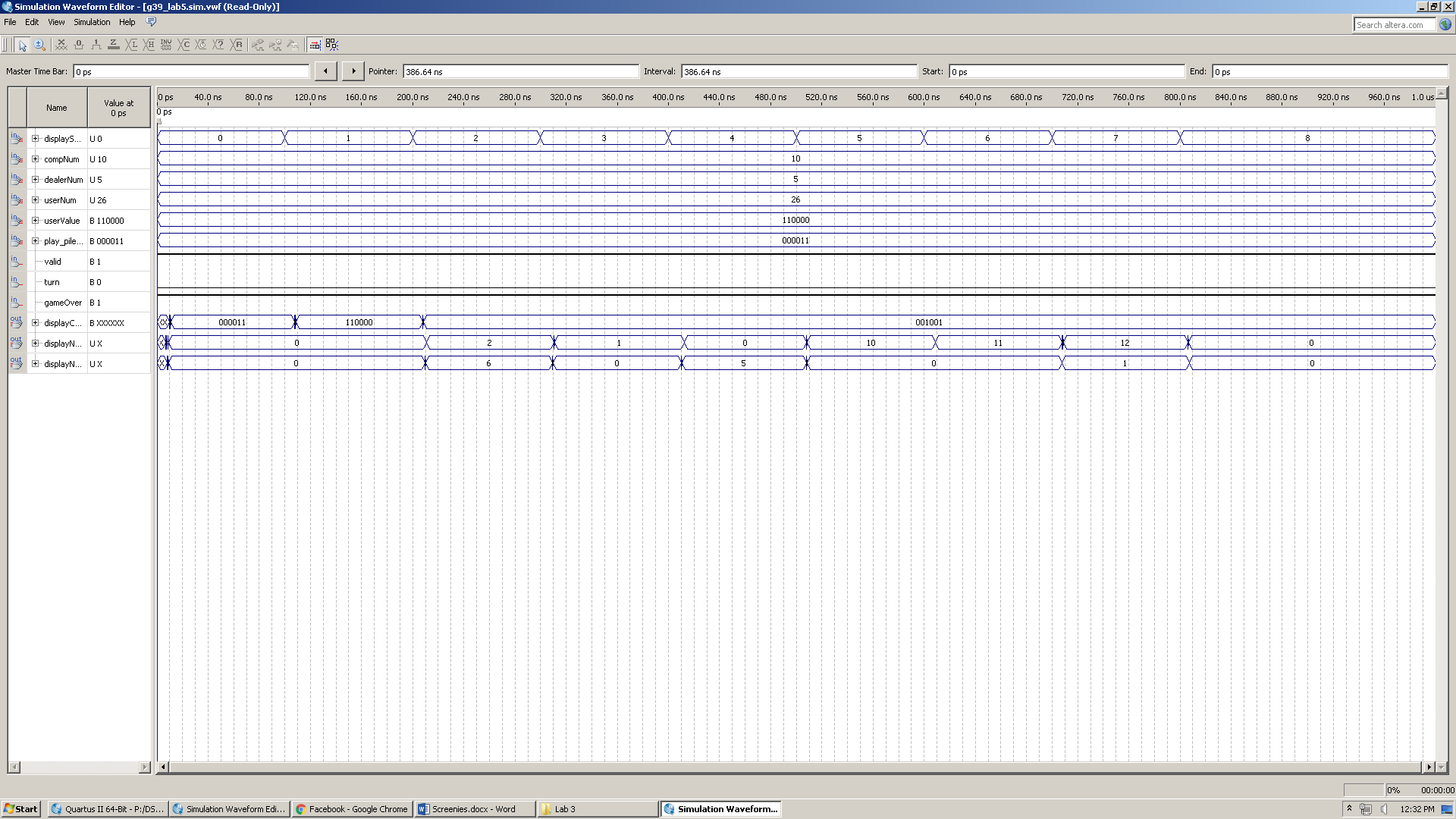
**Simulation:**

Fig. 22

***Reset Circuitry:***

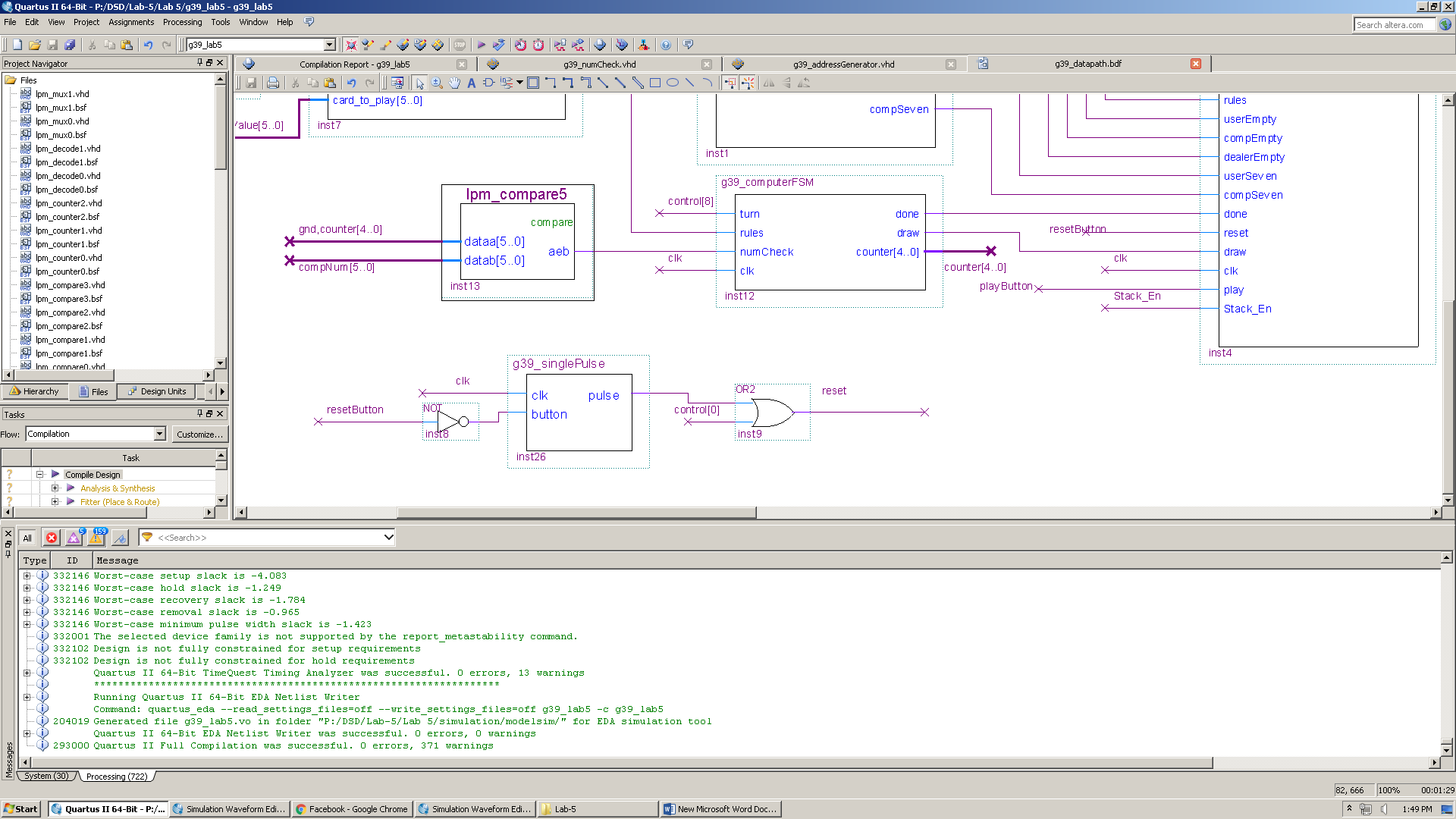
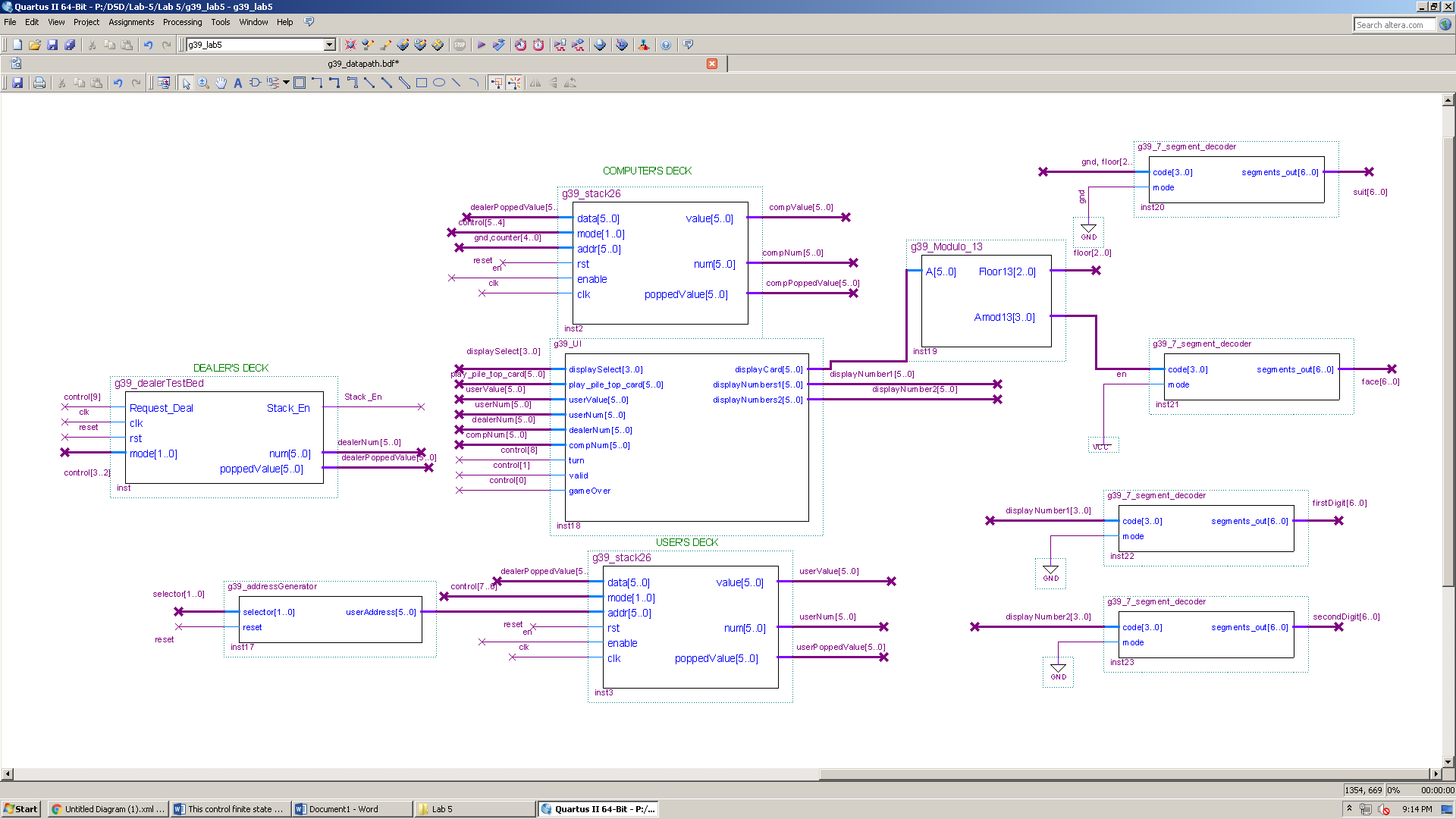
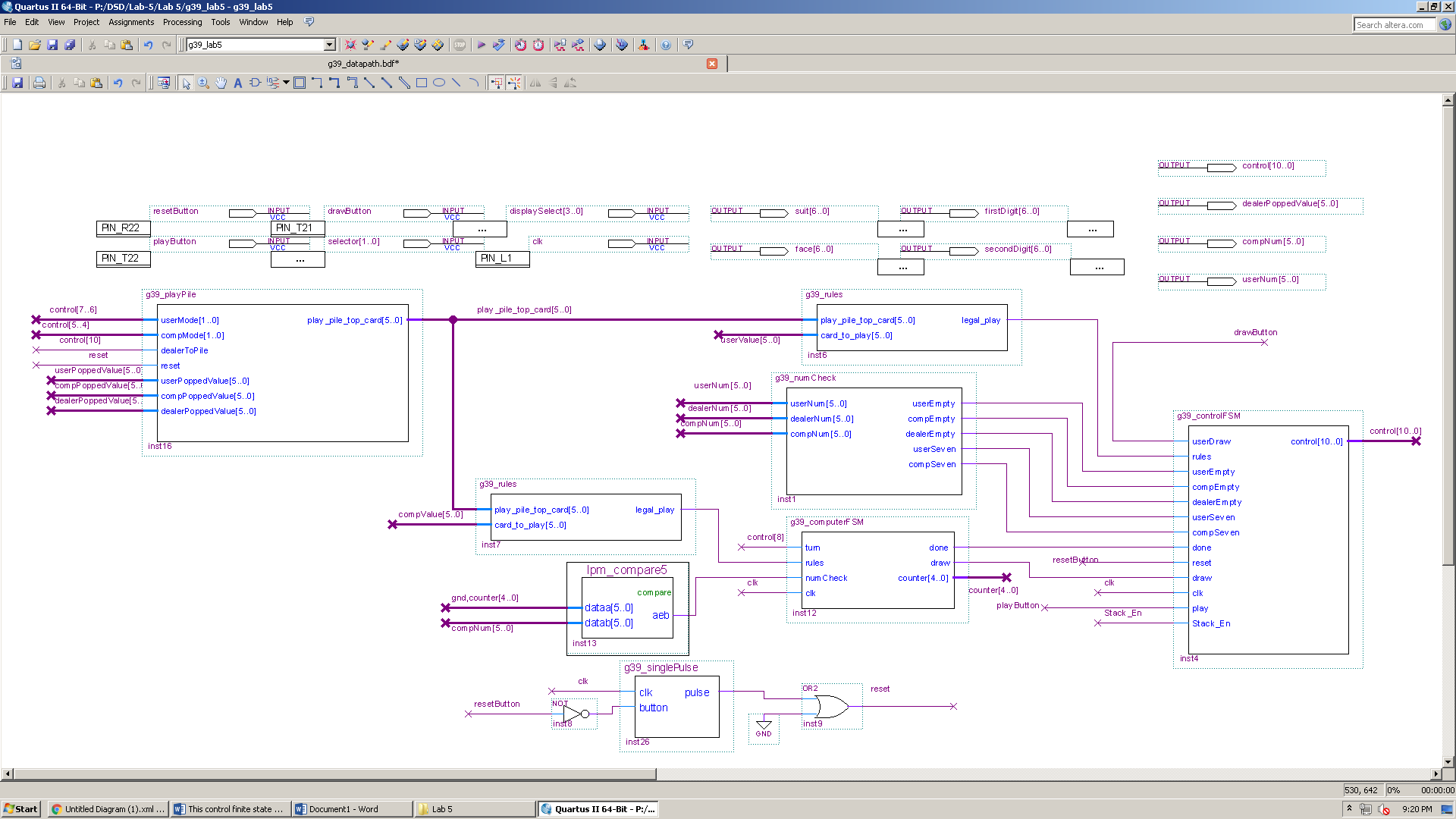


Fig. 23

When starting a game a new game, it is necessary to reset the following components:

* Both players’ hands
* Dealer’s deck
* play pile
* address generator

***The complete system data path:***



Flow

Fig. 24

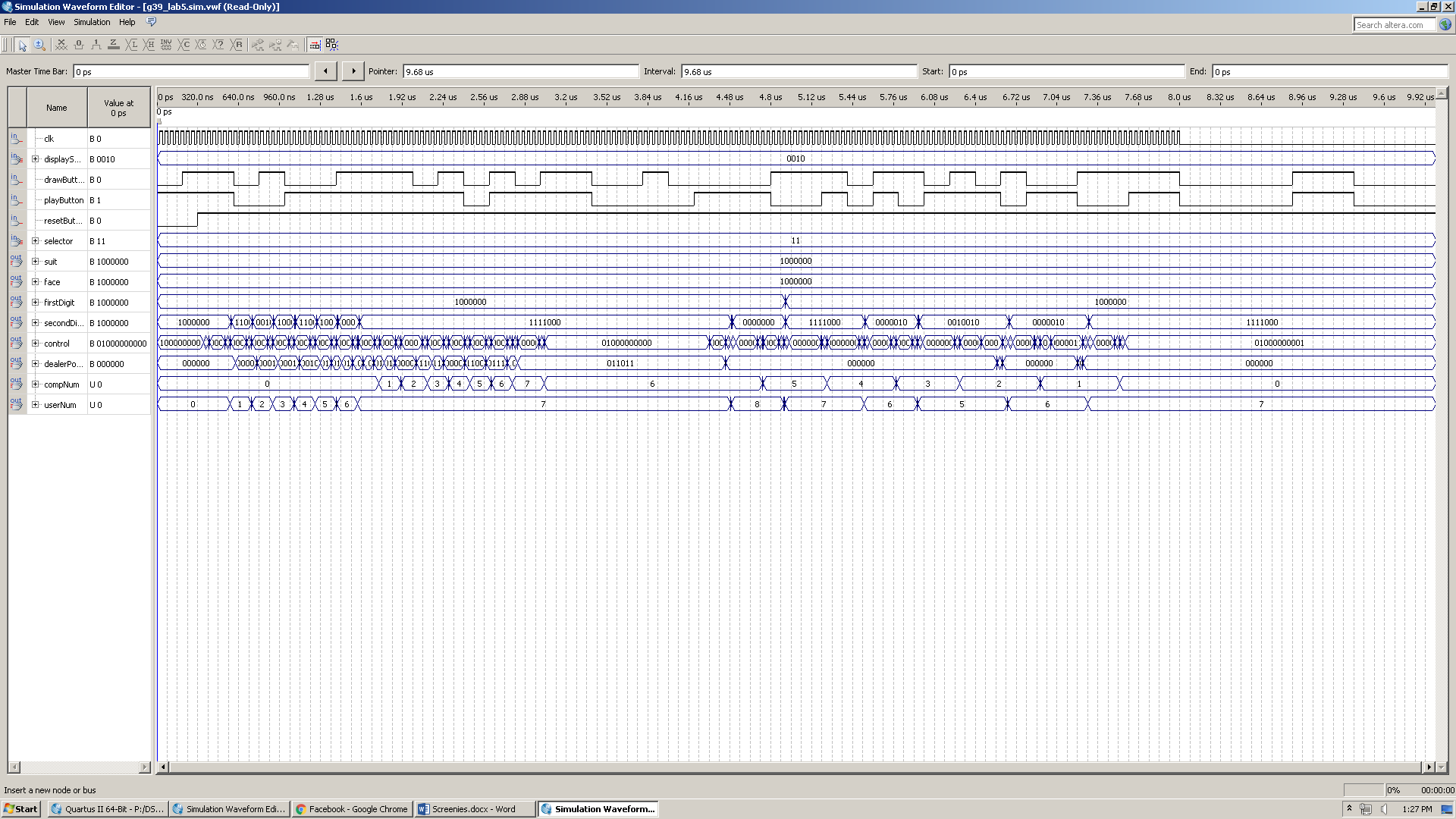
**Simulation:**

Fig. 25

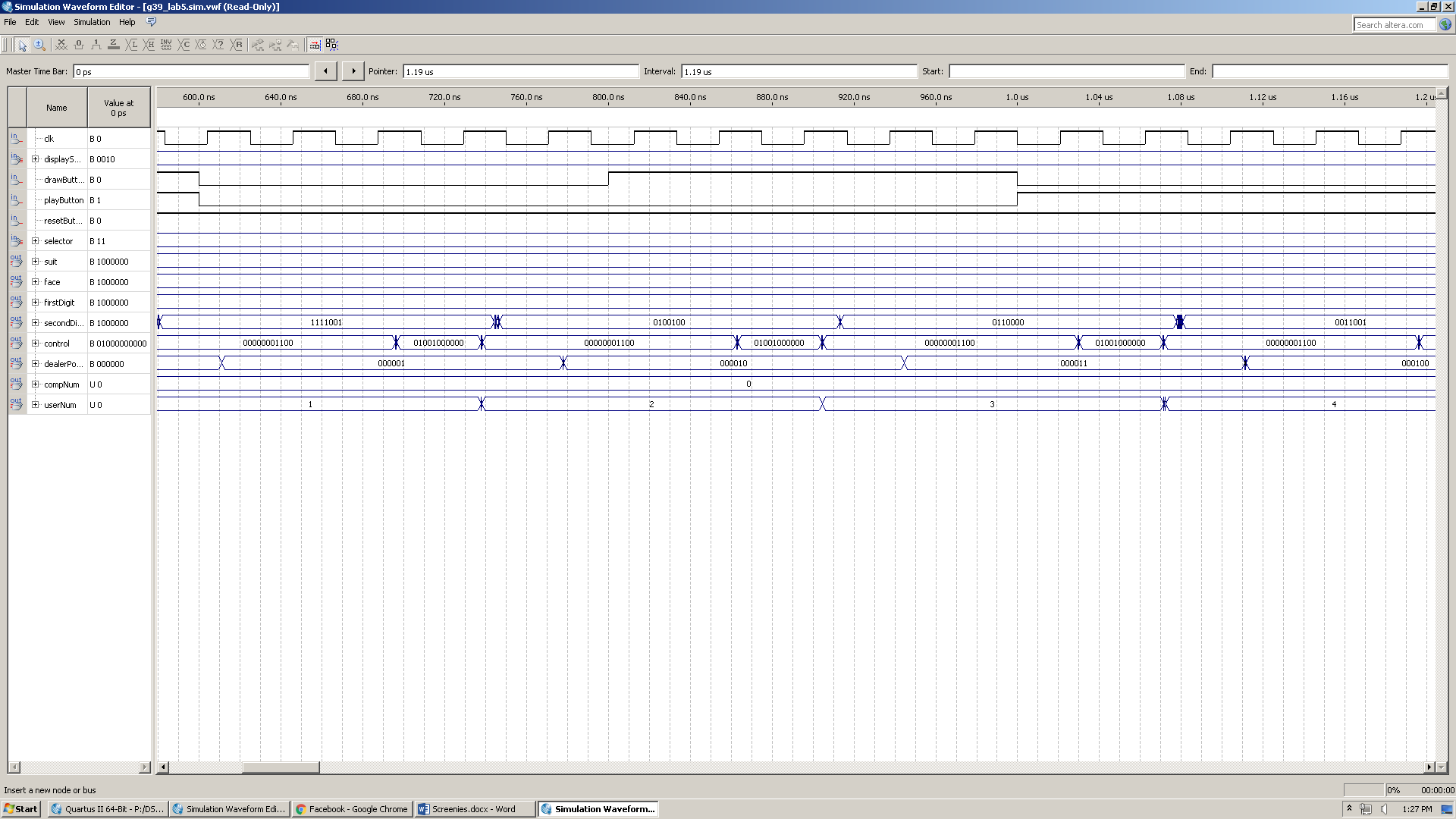
Zoomed in version:

Fig. 26

**Timing Analysis, Flow Summary, and Pin Assignments:**

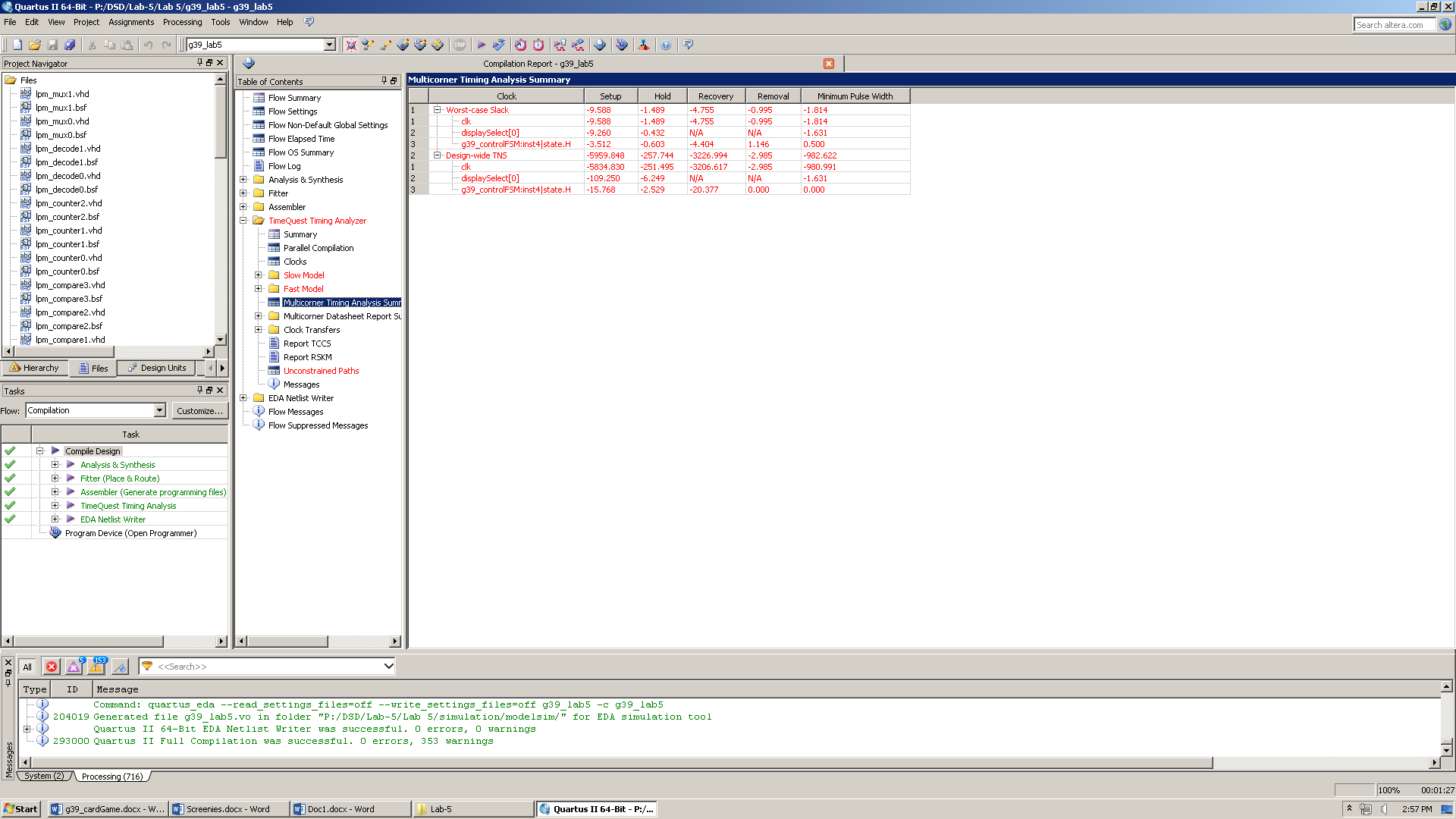


Fig. 27

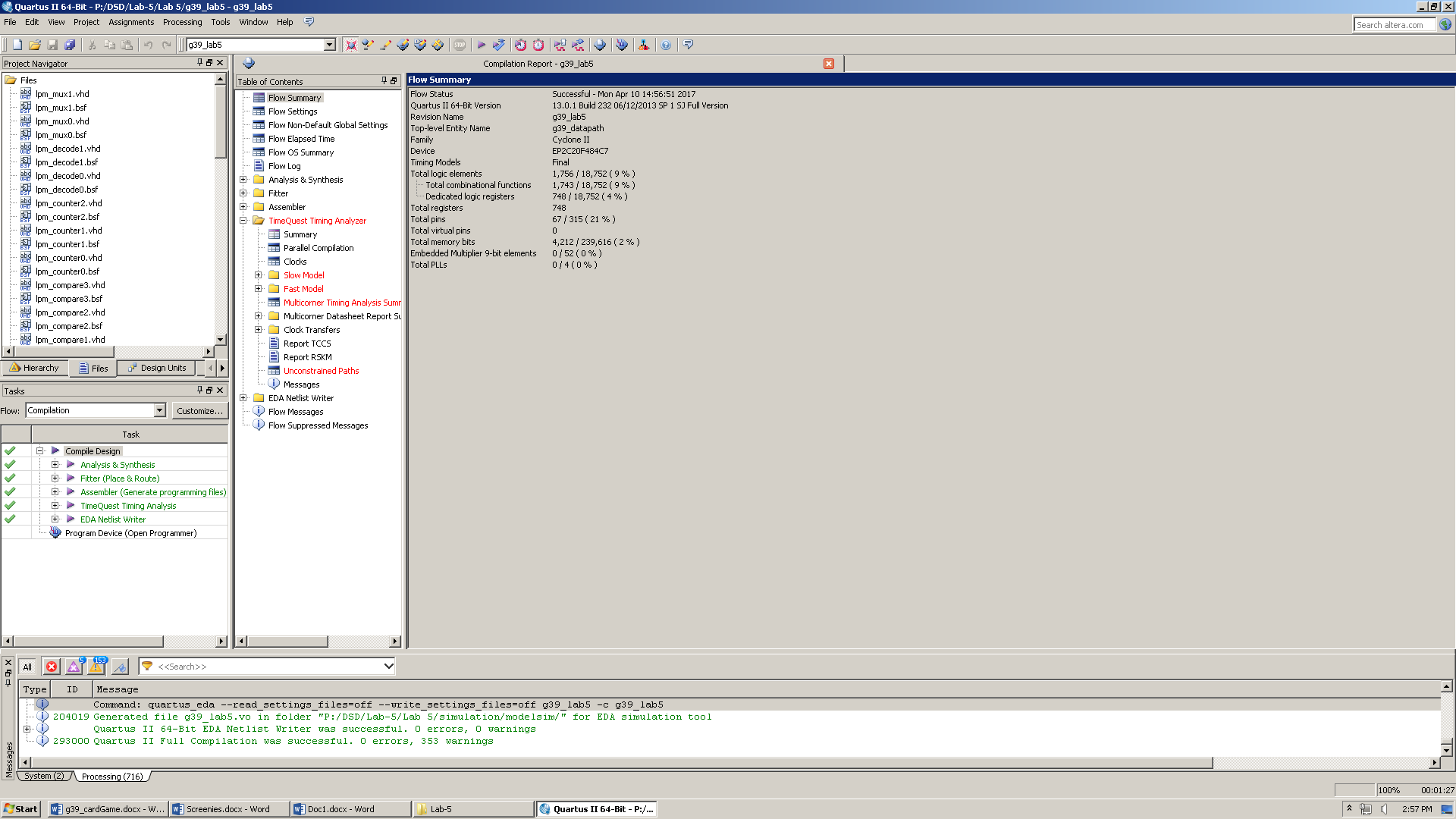


Fig. 28

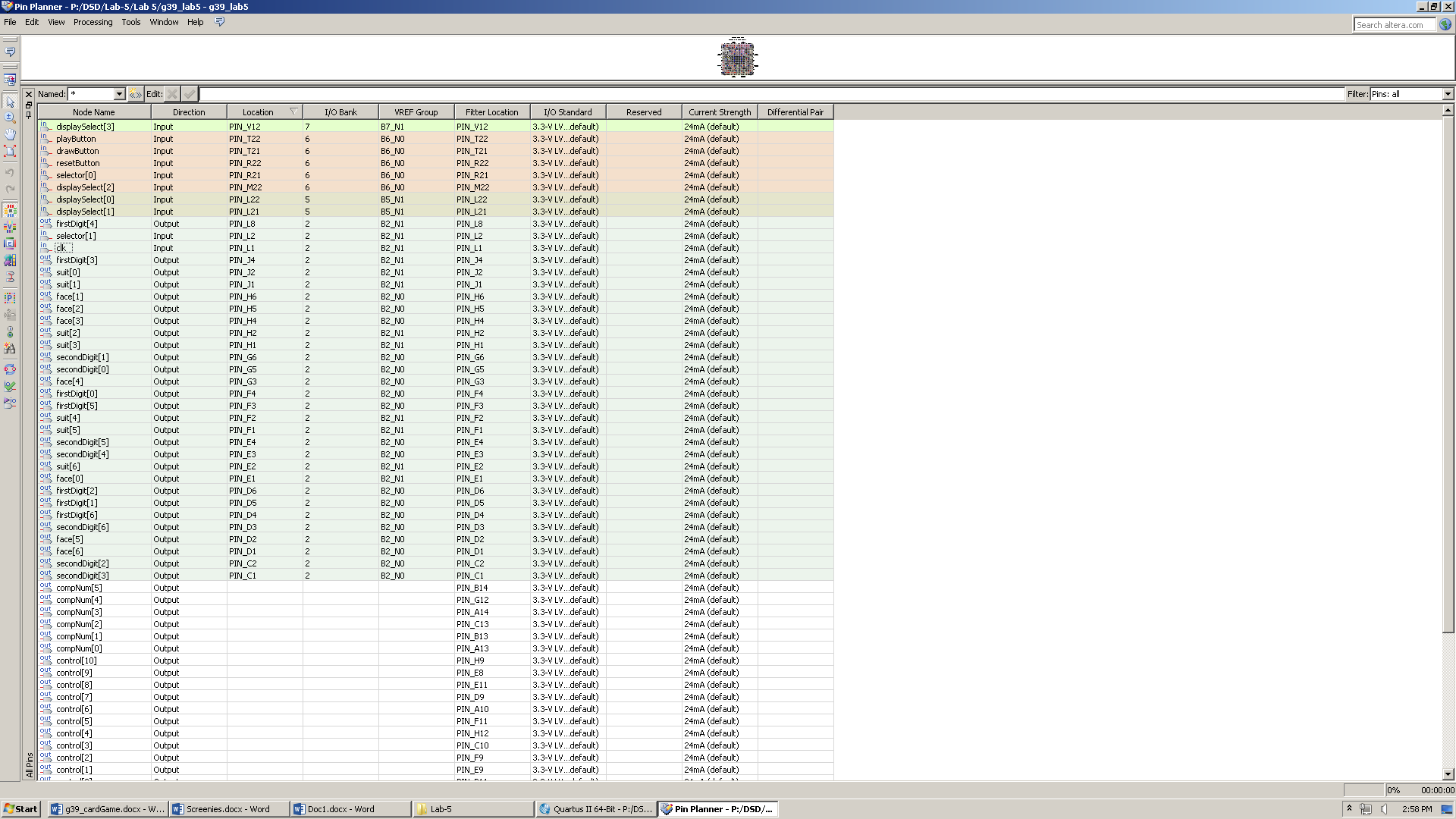


Fig. 29