Group# 39 Saleh Bakhit (260632353) Adham El-Khouly (260601197)

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 (shown in Fig. 1):
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

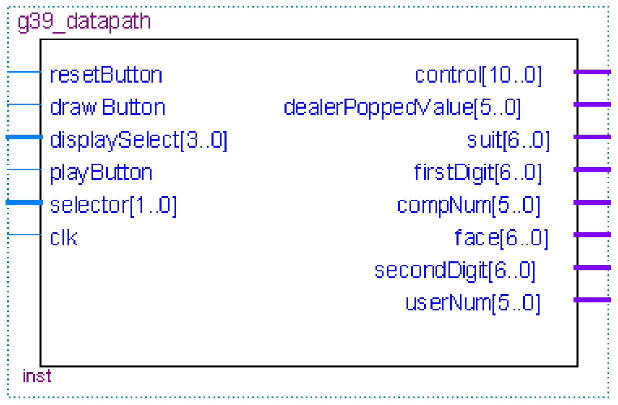


Fig. 1: Block Diagram of the Data Path

Implementation of *g39_datapath*:

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

I. 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.

II. *g39_numCheck:* checks if a deck contains seven cards, zero cards or otherwise.
III. *g39_computerFSM:* automates the computer's turn.
IV. *g39_controlFSM:* a finite state machine that controls the flow of the game.
V. *g39_UI:* user interface circuit that outputs the selected value to the display
VI. *g39_addressGenerator:* changes the address according to the user selected card.
VII. *g39_rules:* determines whether a given card is playable.
VIII. 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_Modulo13
- g39_7_segment_decoder
- g39_singlePulse
- g39_RANDU
- g39_pop_enable
- g39_dealerFSM
- g39_lab3 (stack52)

g39_controlFSM:

This control finite state machine controls almost every aspect of the game. Generally, the following (for a detailed look, consult Fig. 3):

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 (shown in Fig. 2):

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)
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

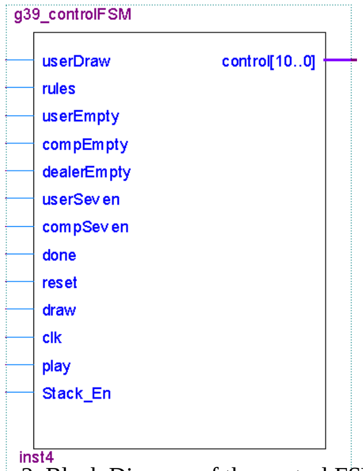


Fig. 2: Block Diagram of the control FSM

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)

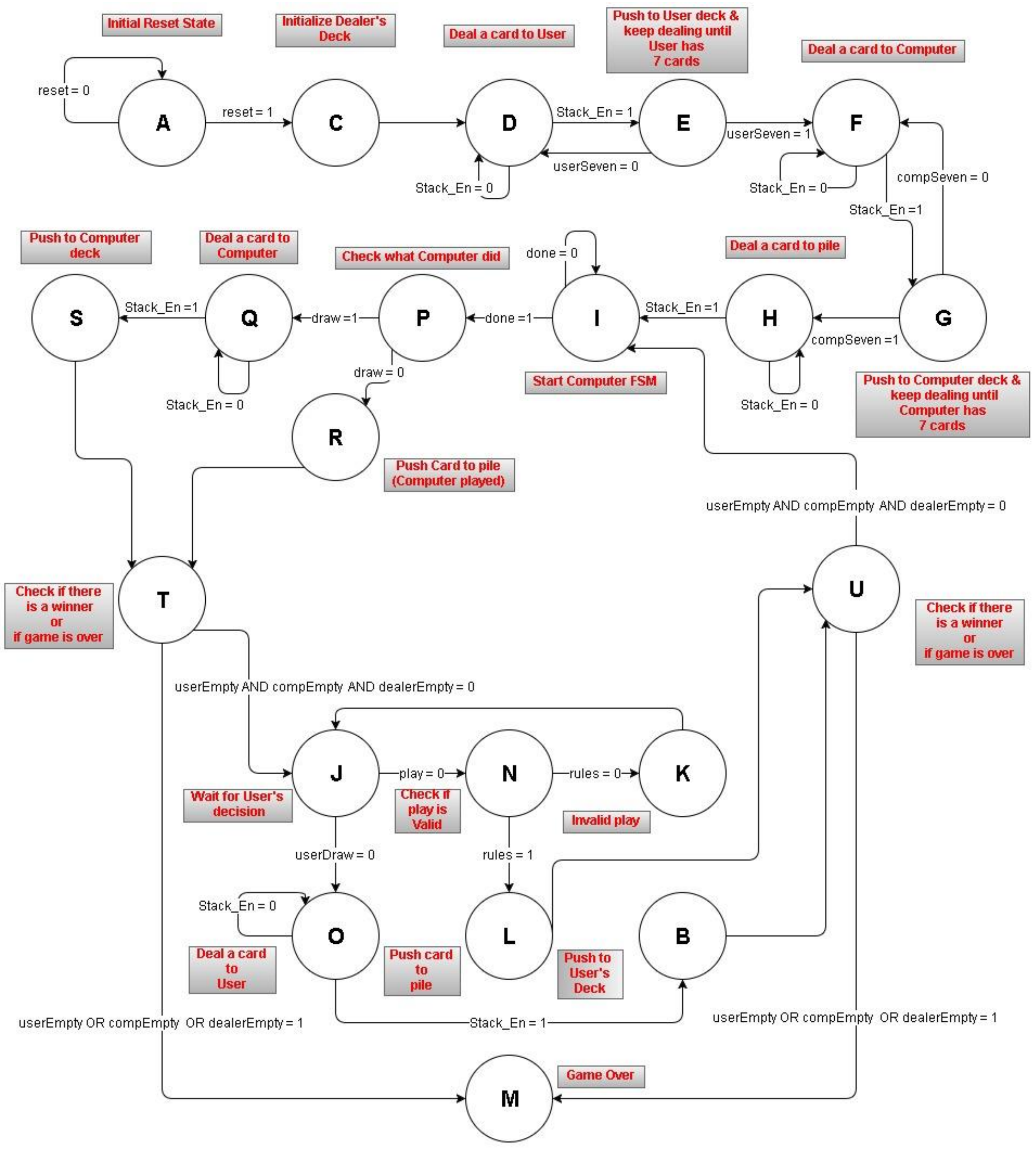


Fig. 3: State Diagram of the control FSM

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:

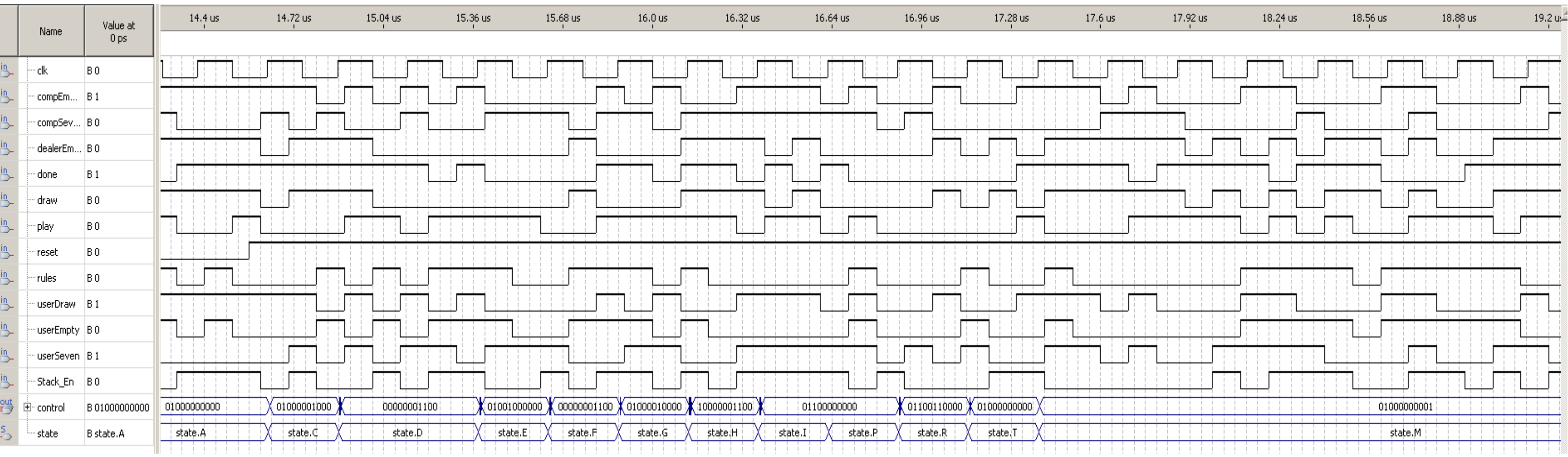


Fig. 4: simulation of the control FSM

Fig. 4 show the simulation of the circuit. It is apparent that the output reflects the logic used in designing the FSM.

g39_addressGenerator:

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

The circuit has the following inputs/outputs (shown in Fig. 5):

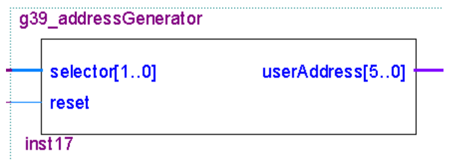
selector: 2-bit input, selector(1) is a toggle switch (to either go up or down), and selector(0) is a push button to perform the increment/decrement

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: Block Diagram of the address generator



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

These are reflected in Fig. 6.

Simulation:

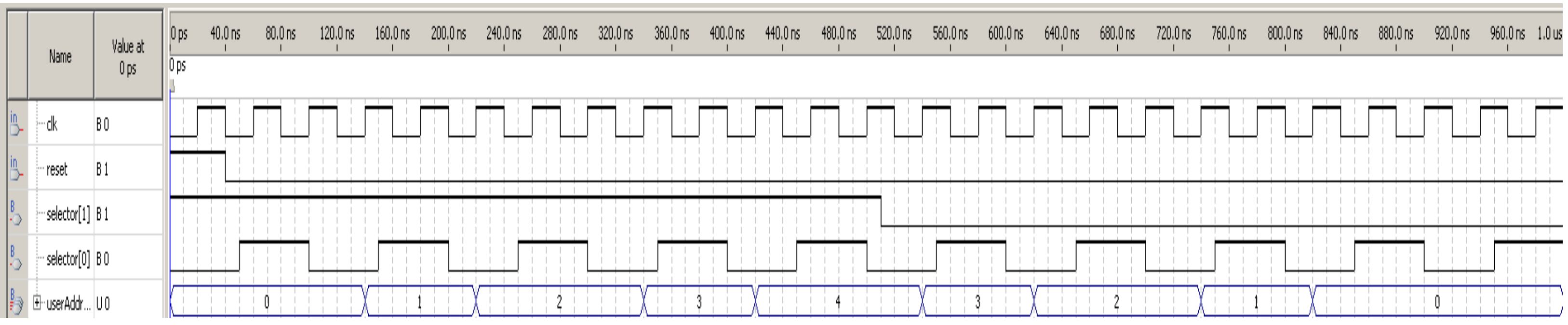
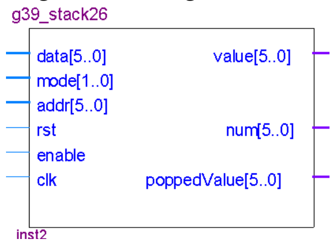


Fig. 6: simulation of the address generator

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 shows the inputs/outputs of the circuit

Fig. 7: Block Diagram of stack26



Simulation:

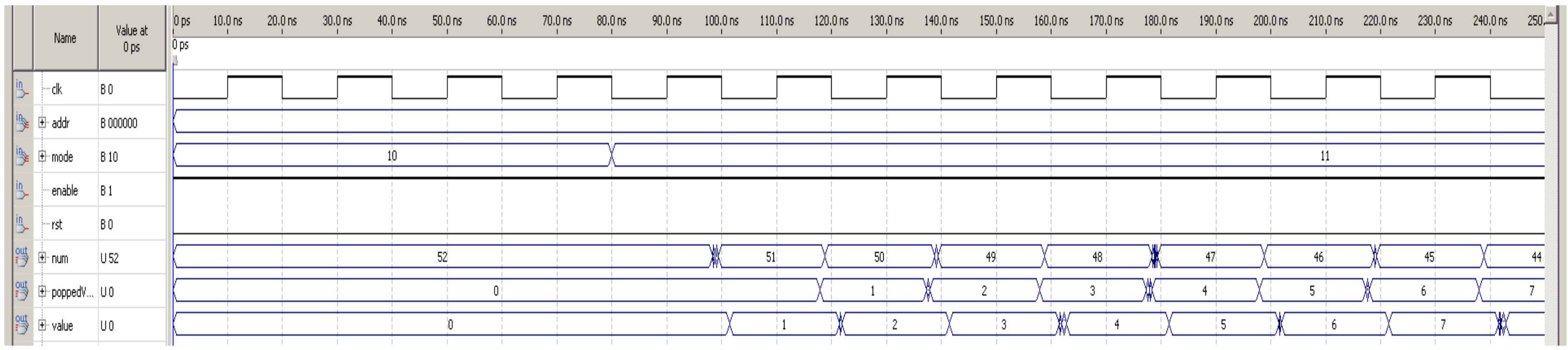


Fig. 8: simulation of stack26

Fig. 8 reflects the changes explained in ***g39_dealerTestbed*** section. Namely, poppedValue can now be used.

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 can be seen in Fig. 11.

This circuit has the following inputs/outputs (shown in Fig. 10):

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 in Fig. 9)

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. 9: Block Diagram of the comparator used

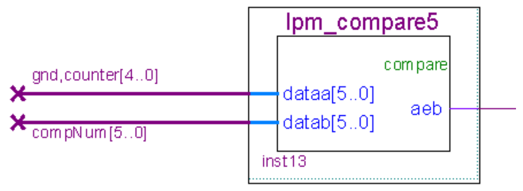
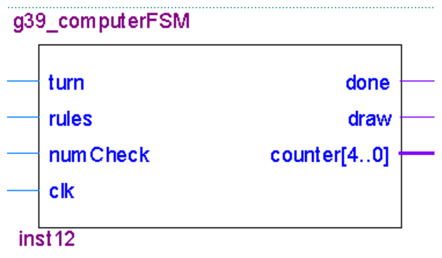


Fig. 10: Block Diagram of the computer FSM



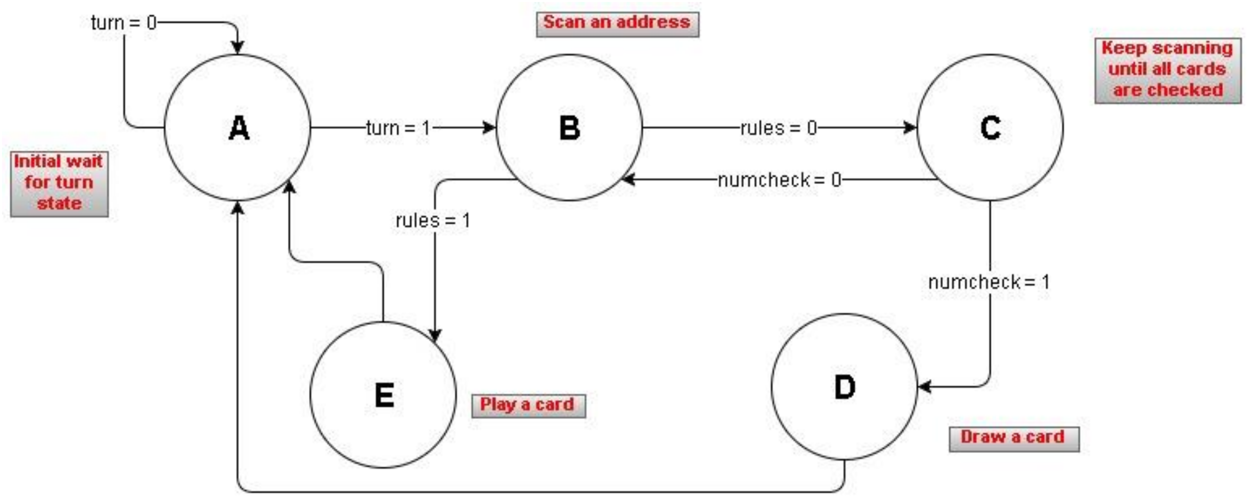


Fig. 11: State Diagram of the computer FSM

Outputs: *done, draw, counter*. It can be seen the simulation in Fig. 12 reflects the logic below.
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:

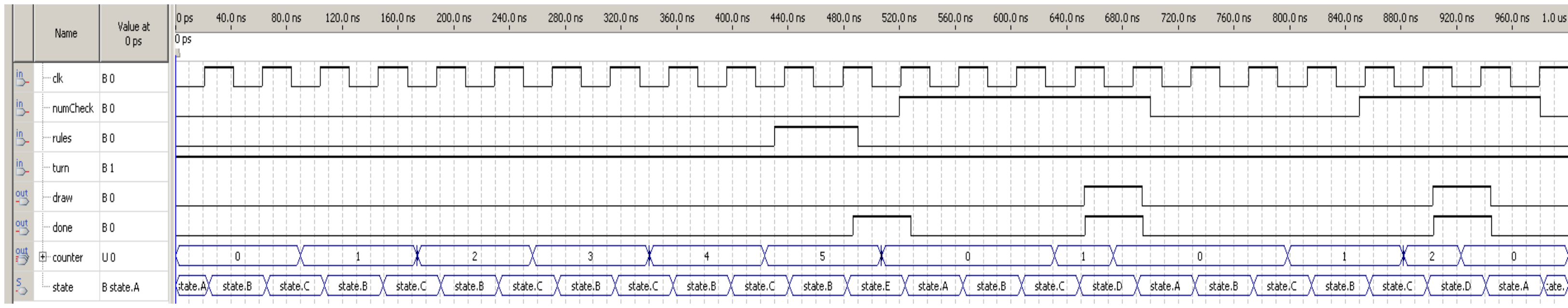


Fig. 12: simulation of control FSM

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 shows the updated dealerFSM.

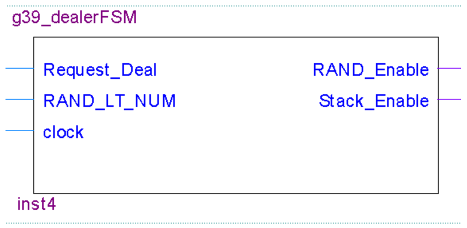


Fig. 13.1: Block Diagram of the dealer FSM

2. 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, shown in Fig. 13.2, that's enabled whenever we pop. This way we deal the popped value and remove it from the deck.

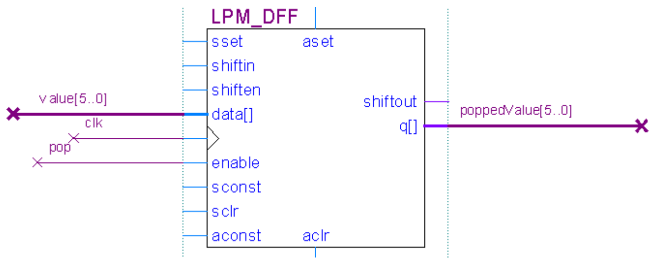
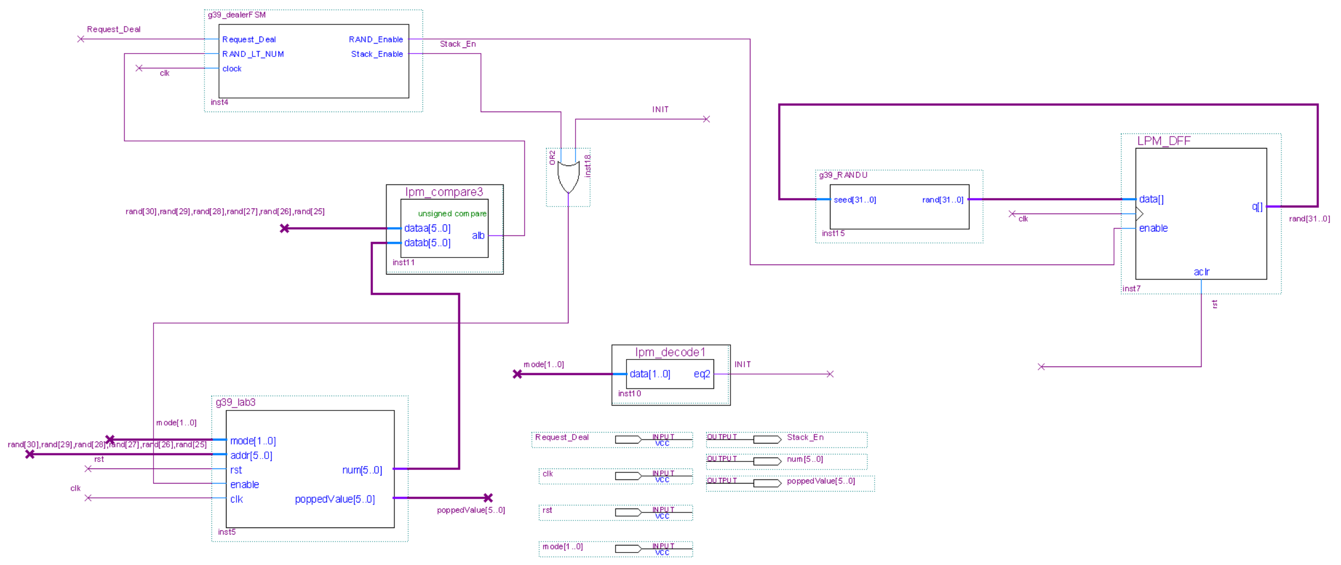


Fig. 13.2: Block Diagram of the dealer D flip flop used

3. Changes in the testbed (Current testbed is shown in Fig. 14 and Fig. 15):
   - 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

For testing purposes, simulations were made to ensure the functionality of the circuit. See Fig. 16

Fig. 14: Schematic of the dealer testbed



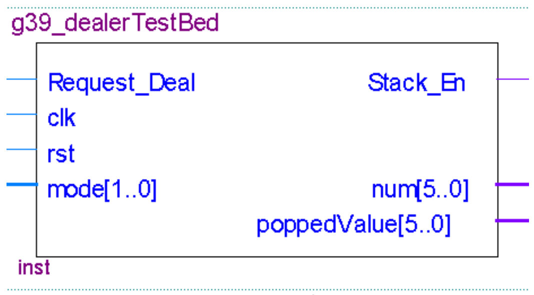


Fig. 15: Block Diagram of the dealer testbed

Simulation:

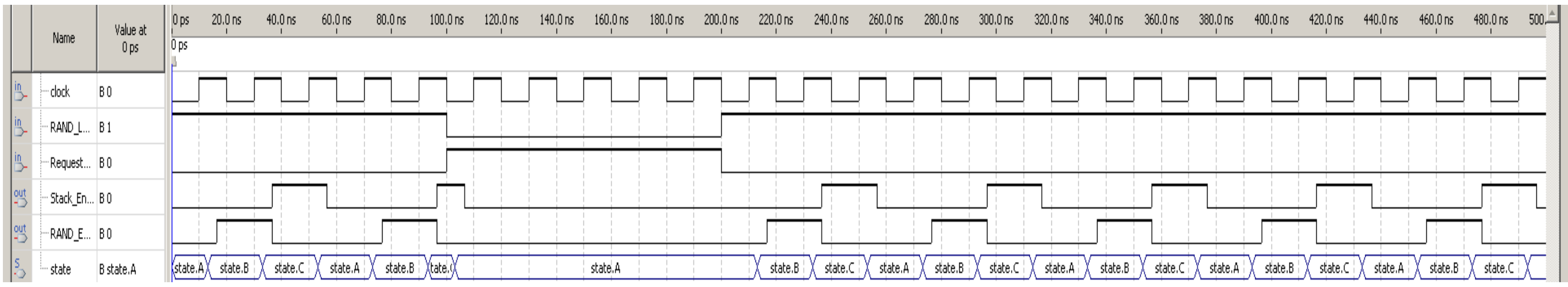


Fig. 16: simulation of the dealer testbed

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 (shown in Fig. 17):

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.

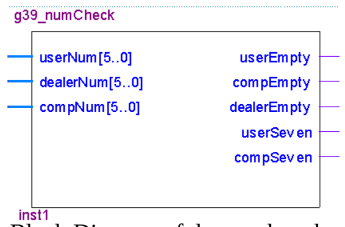


Fig. 17: Block Diagram of the number checking circuitry

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. A simulation, seen in Fig. 18, was carried out to test the circuit.

<u>Simulation:</u>

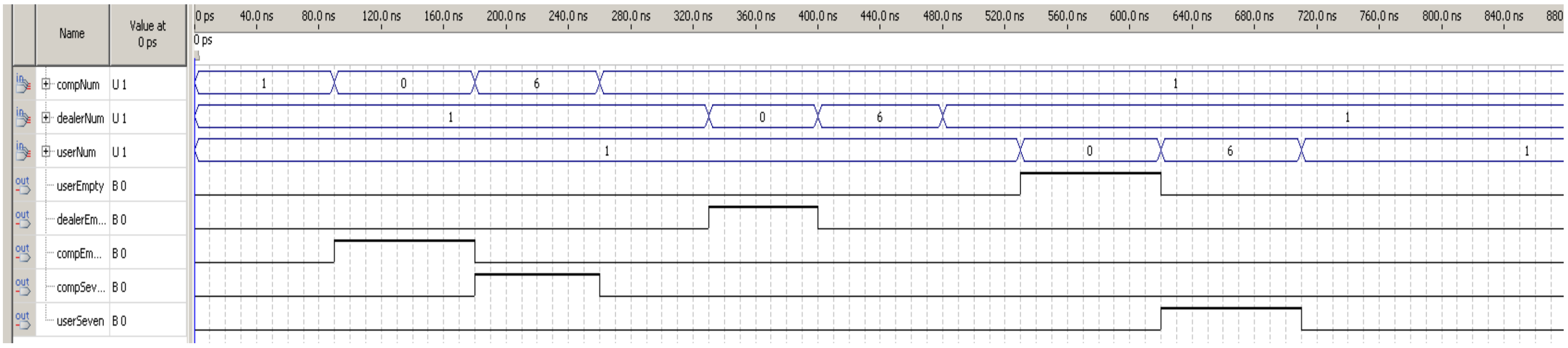


Fig. 18: simulation of the number checking circuitr

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.

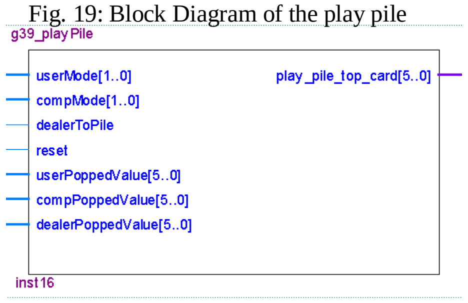
g39_playPile was implemented in VHDL. The above logic can be seen in Fig. 20.

The circuit has the following inputs/outputs (shown in Fig. 19):

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)

Fig. 19: Block Diagram of the play pile



dealerToPile: 1-bit input

reset: 1-bit input

play_pile_top_card: 6-bit output

__Simulation__

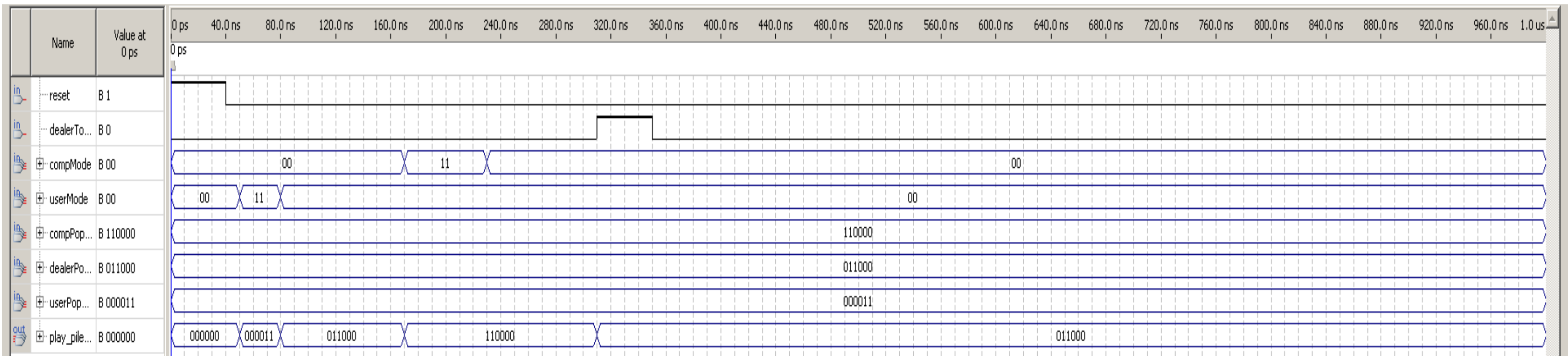


Fig. 20: simulation of the play pile

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 (shown in Fig. 21):

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

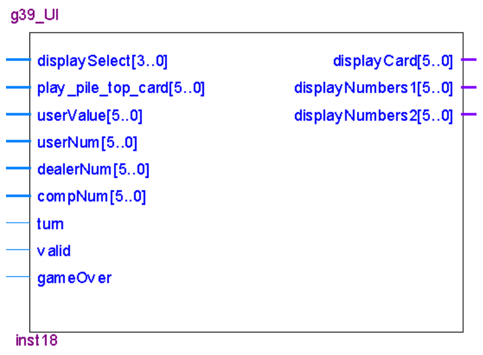


Fig. 21: Block Diagram of the user interface

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".

A simulation that matches the above logic is seen in Fig. 22.

Simulation:

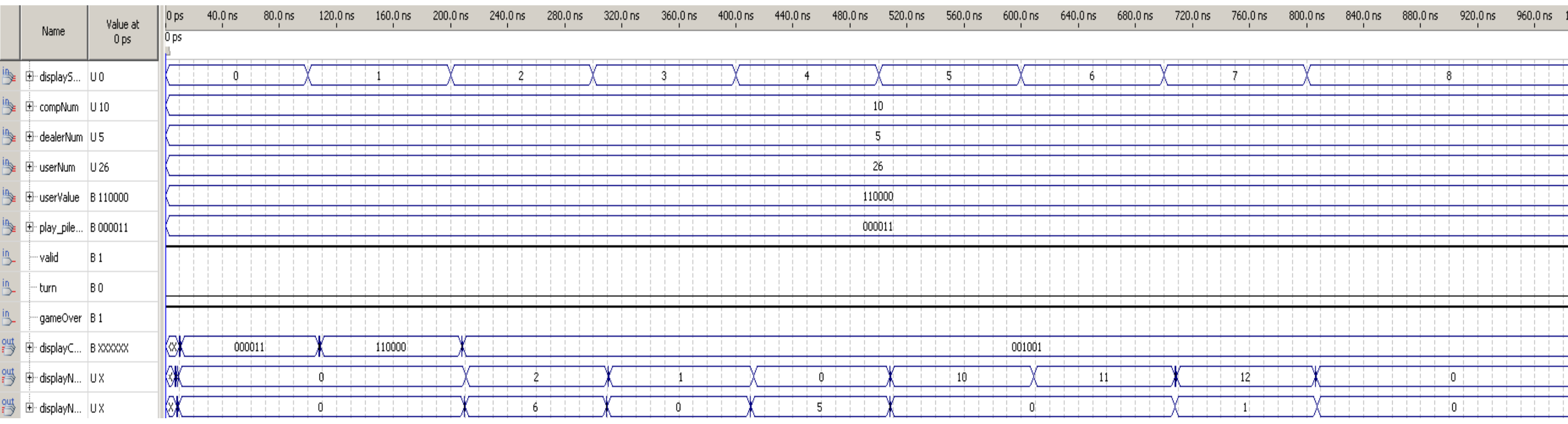


Fig. 22: simulation of the user interface

Reset Circuitry:

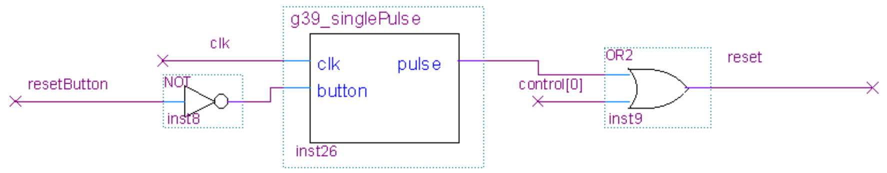


Fig. 23: Schematic of reset circuitry

When starting a new game, the reset circuitry seen in Fig. 23 is used to reset the following components:

- Both players' hands
- Dealer's deck
- play pile
- address generator

The complete system data path:

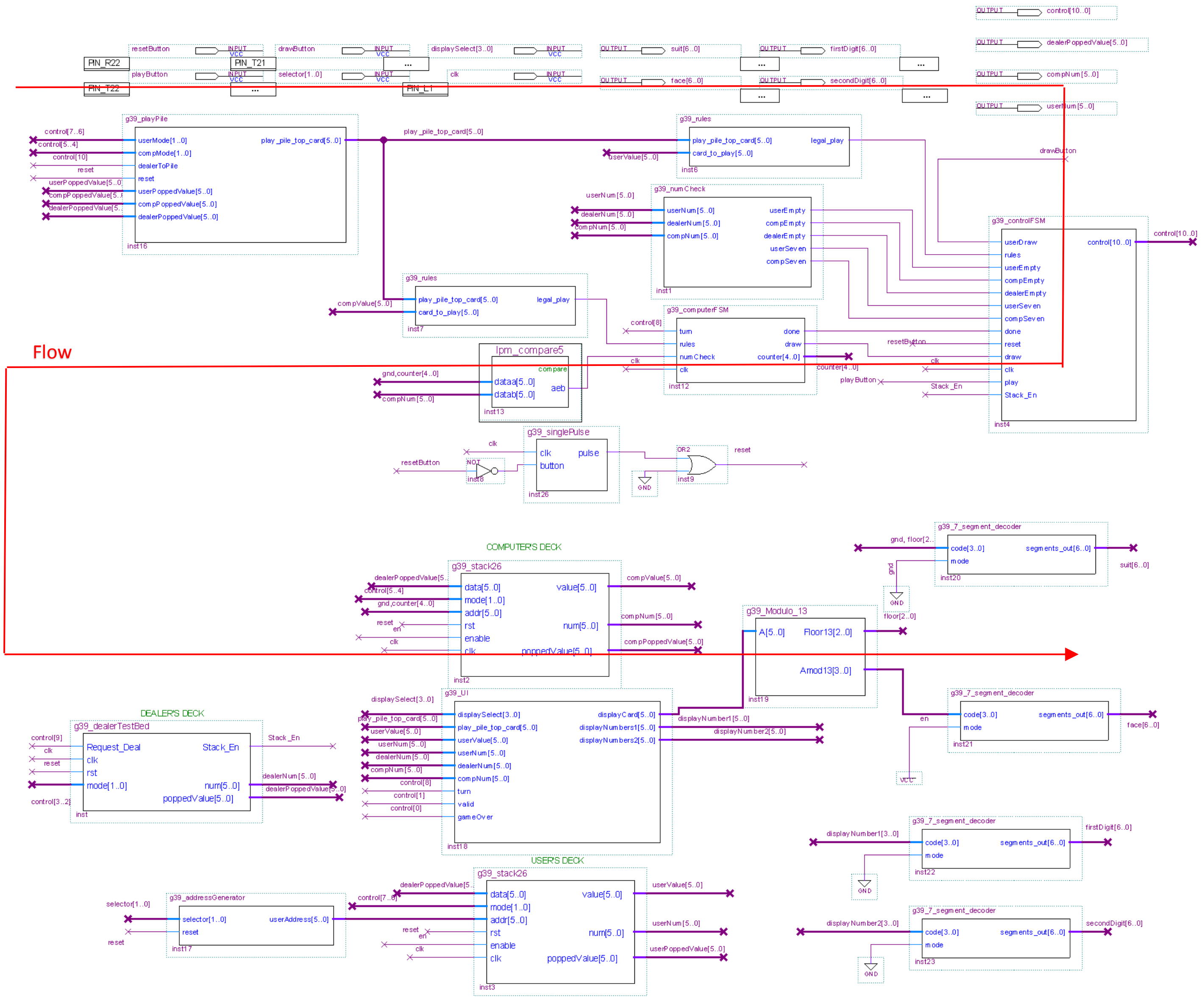


Fig. 24: Schematic of the data path

Simulation:

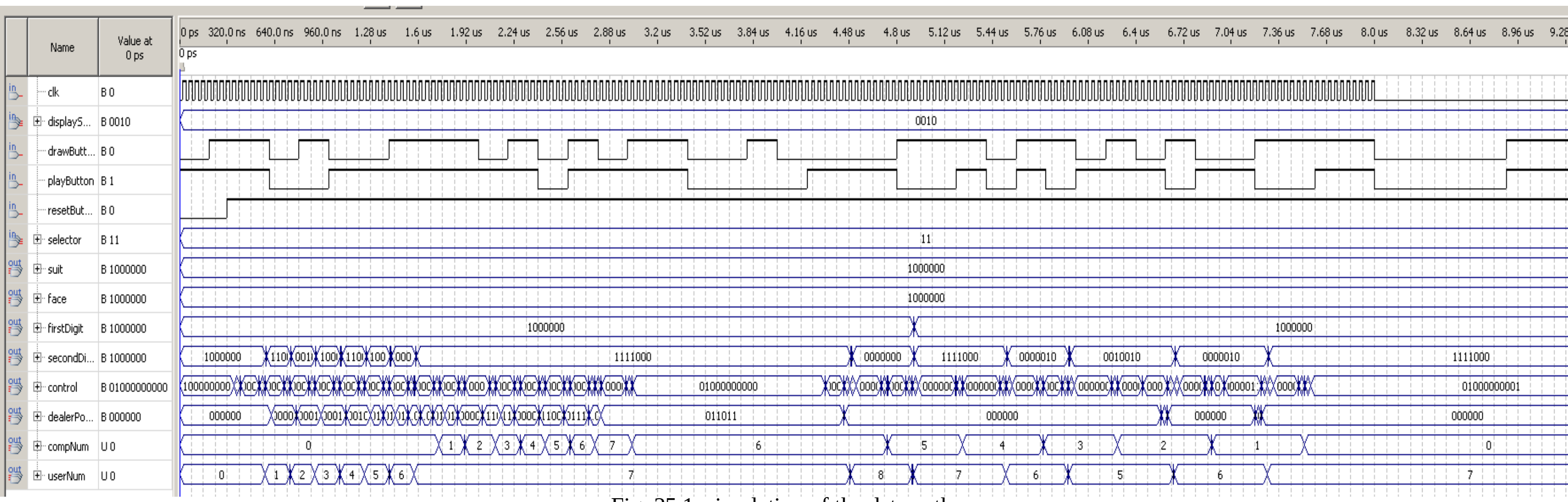


Fig. 25.1: simulation of the data path

Zoomed in version:

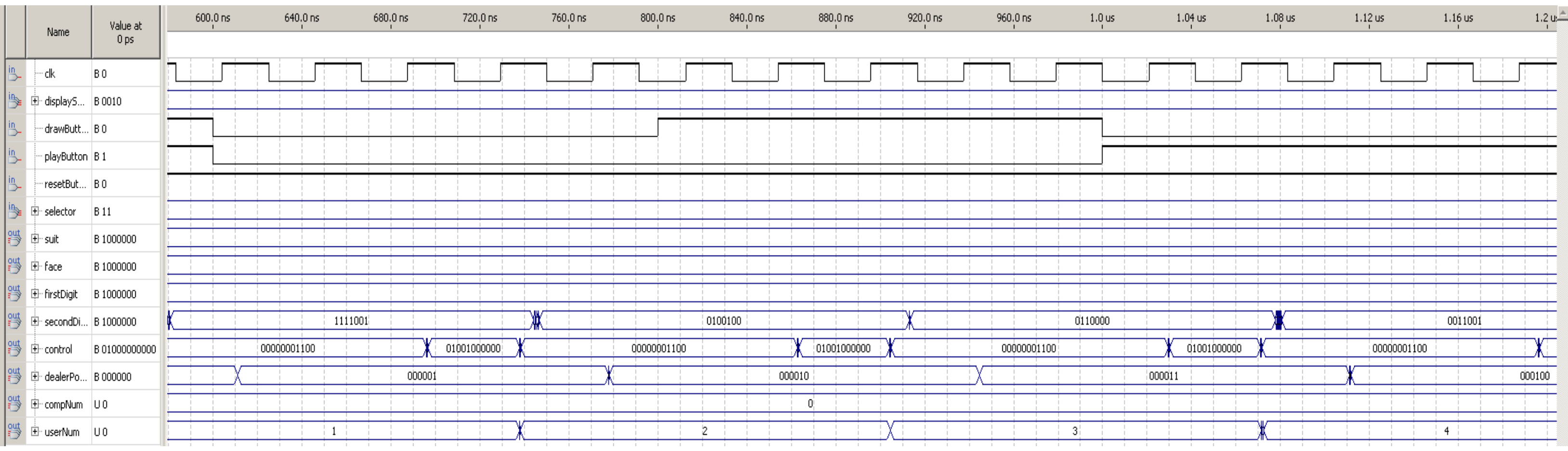
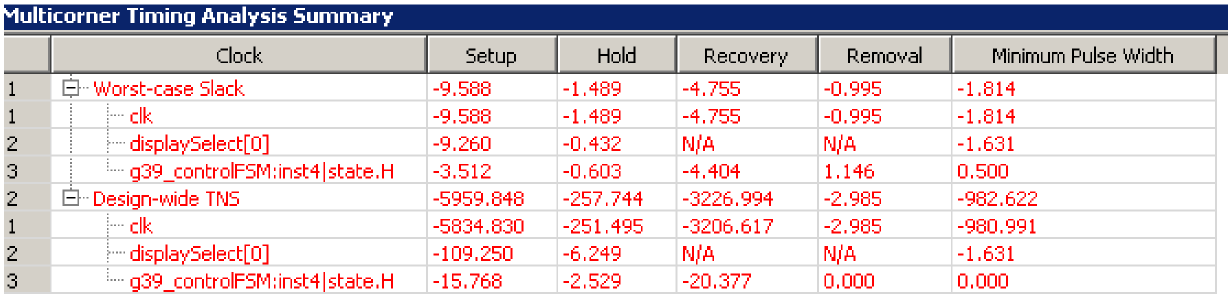


Fig. 25.2: simulation of the data path

Conclusion:

Given the time we had, we did not do as advised and used one Pop_Enable circuit for all 3 stacks. If we had done so we could've used the resources mentioned in Fig. 27 a lot more efficiently. We also did not use a single pulse generator in our user card selector pushbutton. Our selector lines could have been toggle switches instead of a pushbutton, that way we save on resources, since we would not need single pulse generators and would also avoid any rapid flipping through the cards. We had a lot of trouble managing the reset signal to make sure all the finite state machines reset properly, and thus, have edited some state machines from previous labs to match our final design. However, it all worked properly in the end. An extension to the game would be to add more players, this would be very simple, since the computerFSM can be used for all *n* computer players and the user would still play the same exact way.

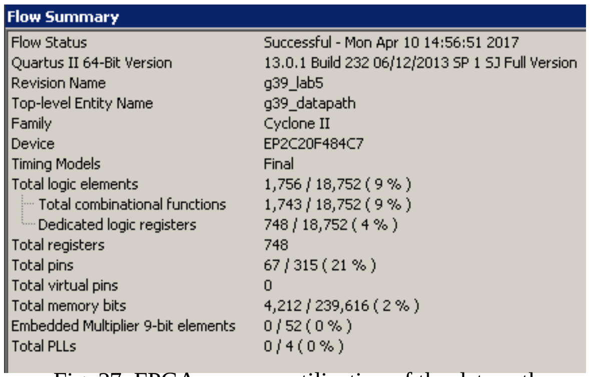
Timing Analysis, Flow Summary, and Pin Assignments:



Multicorner Timing Analysis Summary

	Clock	Setup	Hold	Recovery	Removal	Minimum Pulse Width
1	Worst-case Slack	-9.588	-1.489	-4.755	-0.995	-1.814
1	clk	-9.588	-1.489	-4.755	-0.995	-1.814
2	displaySelect[0]	-9.260	-0.432	N/A	N/A	-1.631
3	g39_controlFSM:inst4|state.H	-3.512	-0.603	-4.404	1.146	0.500
2	Design-wide TNS	-5959.848	-257.744	-3226.994	-2.985	-982.622
1	clk	-5834.830	-251.495	-3206.617	-2.985	-980.991
2	displaySelect[0]	-109.250	-6.249	N/A	N/A	-1.631
3	g39_controlFSM:inst4|state.H	-15.768	-2.529	-20.377	0.000	0.000

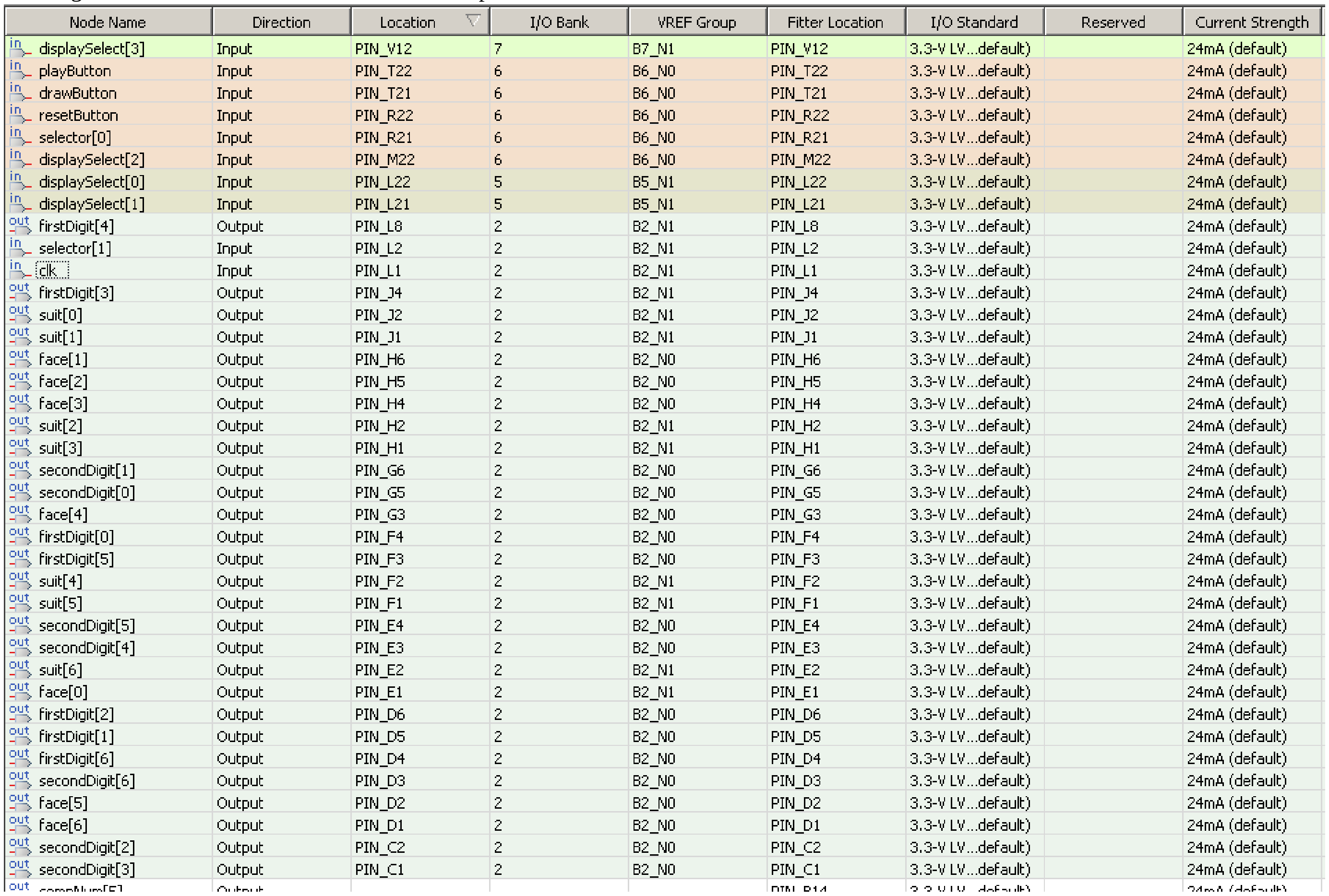
Fig. 26: Timing performance of the data path



Flow Summary

Flow Status	Successful - Mon Apr 10 14:56:51 2017
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Full Version
Revision Name	g39_lab5
Top-level Entity Name	g39_datapath
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	1,756 / 18,752 (9 %)
Total combinational functions	1,743 / 18,752 (9 %)
Dedicated logic registers	748 / 18,752 (4 %)
Total registers	748
Total pins	67 / 315 (21 %)
Total virtual pins	0
Total memory bits	4,212 / 239,616 (2 %)
Embedded Multiplier 9-bit elements	0 / 52 (0 %)
Total PLLs	0 / 4 (0 %)

Fig. 27: FPGA recourse utilization of the data path



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength
displaySelect[3]	Input	PIN_V12	7	B7_N1	PIN_V12	3.3-V LV...default)		24mA (default)
playButton	Input	PIN_T22	6	B6_N0	PIN_T22	3.3-V LV...default)		24mA (default)
drawButton	Input	PIN_T21	6	B6_N0	PIN_T21	3.3-V LV...default)		24mA (default)
resetButton	Input	PIN_R22	6	B6_N0	PIN_R22	3.3-V LV...default)		24mA (default)
selector[0]	Input	PIN_R21	6	B6_N0	PIN_R21	3.3-V LV...default)		24mA (default)
displaySelect[2]	Input	PIN_M22	6	B6_N0	PIN_M22	3.3-V LV...default)		24mA (default)
displaySelect[0]	Input	PIN_L22	5	B5_N1	PIN_L22	3.3-V LV...default)		24mA (default)
displaySelect[1]	Input	PIN_L21	5	B5_N1	PIN_L21	3.3-V LV...default)		24mA (default)
firstDigit[4]	Output	PIN_L8	2	B2_N1	PIN_L8	3.3-V LV...default)		24mA (default)
selector[1]	Input	PIN_L2	2	B2_N1	PIN_L2	3.3-V LV...default)		24mA (default)
clk	Input	PIN_L1	2	B2_N1	PIN_L1	3.3-V LV...default)		24mA (default)
firstDigit[3]	Output	PIN_J4	2	B2_N1	PIN_J4	3.3-V LV...default)		24mA (default)
suit[0]	Output	PIN_J2	2	B2_N1	PIN_J2	3.3-V LV...default)		24mA (default)
suit[1]	Output	PIN_J1	2	B2_N1	PIN_J1	3.3-V LV...default)		24mA (default)
face[1]	Output	PIN_H6	2	B2_N0	PIN_H6	3.3-V LV...default)		24mA (default)
face[2]	Output	PIN_H5	2	B2_N0	PIN_H5	3.3-V LV...default)		24mA (default)
face[3]	Output	PIN_H4	2	B2_N0	PIN_H4	3.3-V LV...default)		24mA (default)
suit[2]	Output	PIN_H2	2	B2_N1	PIN_H2	3.3-V LV...default)		24mA (default)
suit[3]	Output	PIN_H1	2	B2_N1	PIN_H1	3.3-V LV...default)		24mA (default)
secondDigit[1]	Output	PIN_G6	2	B2_N0	PIN_G6	3.3-V LV...default)		24mA (default)
secondDigit[0]	Output	PIN_G5	2	B2_N0	PIN_G5	3.3-V LV...default)		24mA (default)
face[4]	Output	PIN_G3	2	B2_N0	PIN_G3	3.3-V LV...default)		24mA (default)
firstDigit[0]	Output	PIN_F4	2	B2_N0	PIN_F4	3.3-V LV...default)		24mA (default)
firstDigit[5]	Output	PIN_F3	2	B2_N0	PIN_F3	3.3-V LV...default)		24mA (default)
suit[4]	Output	PIN_F2	2	B2_N1	PIN_F2	3.3-V LV...default)		24mA (default)
suit[5]	Output	PIN_F1	2	B2_N1	PIN_F1	3.3-V LV...default)		24mA (default)
secondDigit[5]	Output	PIN_E4	2	B2_N0	PIN_E4	3.3-V LV...default)		24mA (default)
secondDigit[4]	Output	PIN_E3	2	B2_N0	PIN_E3	3.3-V LV...default)		24mA (default)
suit[6]	Output	PIN_E2	2	B2_N1	PIN_E2	3.3-V LV...default)		24mA (default)
face[0]	Output	PIN_E1	2	B2_N1	PIN_E1	3.3-V LV...default)		24mA (default)
firstDigit[2]	Output	PIN_D6	2	B2_N0	PIN_D6	3.3-V LV...default)		24mA (default)
firstDigit[1]	Output	PIN_D5	2	B2_N0	PIN_D5	3.3-V LV...default)		24mA (default)
firstDigit[6]	Output	PIN_D4	2	B2_N0	PIN_D4	3.3-V LV...default)		24mA (default)
secondDigit[6]	Output	PIN_D3	2	B2_N0	PIN_D3	3.3-V LV...default)		24mA (default)
face[5]	Output	PIN_D2	2	B2_N0	PIN_D2	3.3-V LV...default)		24mA (default)
face[6]	Output	PIN_D1	2	B2_N0	PIN_D1	3.3-V LV...default)		24mA (default)
secondDigit[2]	Output	PIN_C2	2	B2_N0	PIN_C2	3.3-V LV...default)		24mA (default)
secondDigit[3]	Output	PIN_C1	2	B2_N0	PIN_C1	3.3-V LV...default)		24mA (default)
compNum[5]	Output				PIN_B14	3.3-V LV...default)		24mA (default)

Fig. 28: Pin assignments of the data path