General description of the FSM

The purpose of the FSM is to create a gambling game and to keep track of a player's bets and funds. The FSM can be applied to operate slot machines in casinos. Firstly the player will dispense funds into their account. The player will then choose the amount of credits they want to bet. If there is enough funds for the player to bet, the game will start and the wheels will start spinning. The player will then operate a button to stop each wheel, and if all three wheels are the same number, the player will win credits based on their bet. The winnings are determined by a dice roll where the player will earn the amount of credits betted multiplied by the amount on the die. If the player fails to match up the wheels, they lose the amount of credits they betted.

Modules in the FSM project

The individual modules used in the project are vDFFRL, MUX2, MUX3, RNG, gamblinggame, and gamblingame_tb as seen in the appendix.

vDFFRL

Takes n as a parameter for bitlength. Verilog D Flip Flop with Reset and Load. A flip flop that stores the input value as an output on the rising edge of the clock only when load is asserted. Resets to 0 when reset is asserted on the rising edge of the clock.

MUX2

Takes n as a parameter for bitlength. Contains two options as input and a one bit select the output one of the two inputs.

MUX3

Takes n as a parameter for bitlength. Contains three options as input and a two bit select the output one of the three inputs.

RNG

Random number generator. Continuously increments a number from a minimum value to a maximum value, outputting a pseudorandom number between the min and max inputs.

gamblinggame

Contains the datapath and FSM for the slot machine. Uses flip flops, multiplexers, and other elements to store and structure the data, while an FSM controls the data flow in the ‘always’ blocks. The game is explained in section 1 while the schematics are shown graphically in the later sections.

The top level module. Input ‘clk’ serves as the clock to update registers. Input ‘rst’ resets all registers and brings the FSM to the initial state. Input ‘fundsadded’ specifies the amount of credits to the machine and ‘fundsbutton’ adds the funds. Input ‘bet’ specifies the bet amount. Input ‘startbutton’ starts the game and ‘wheelbutton’ stops the wheels. The output ‘displayfunds’ shows the amount of credits a player has, ‘displaybet’ shows the amount the player bets. Output ‘displaywheel0’, ‘displaywheel1’, ‘displaywheel2’, ‘displaydice’ display the wheel and dice values.

The FSM has flags from the datapath ‘win’ when all the wheels are the same value, ‘stopdispense’ when enough credits are given when won, and ‘enoughfunds’ when checks if there are enough credits to bet and start the game, ‘donespins’ checks whether three spins were done. ‘loadfunds’ and ‘selfunds’ update a player’s funds. ‘loadbet’ locks in the bet value once the game starts. ‘loadwheel’ and ‘loaddice’ lock in the wheel and dice values. ‘incrementspin’ keeps control of the amount of spins. ‘startdispense’ controls the counter that counts how many times credits are dispensed. ‘rstspins, rstwheels, rstdcounter’ resets the wheels, dice, and counters without resetting player funds.

The FSM has 11 states. ‘PLACEBET’ will be the default state and will load the bet register, and reset the wheels, dice, and counters. ’ADDFUNDS’ will add a specified amount

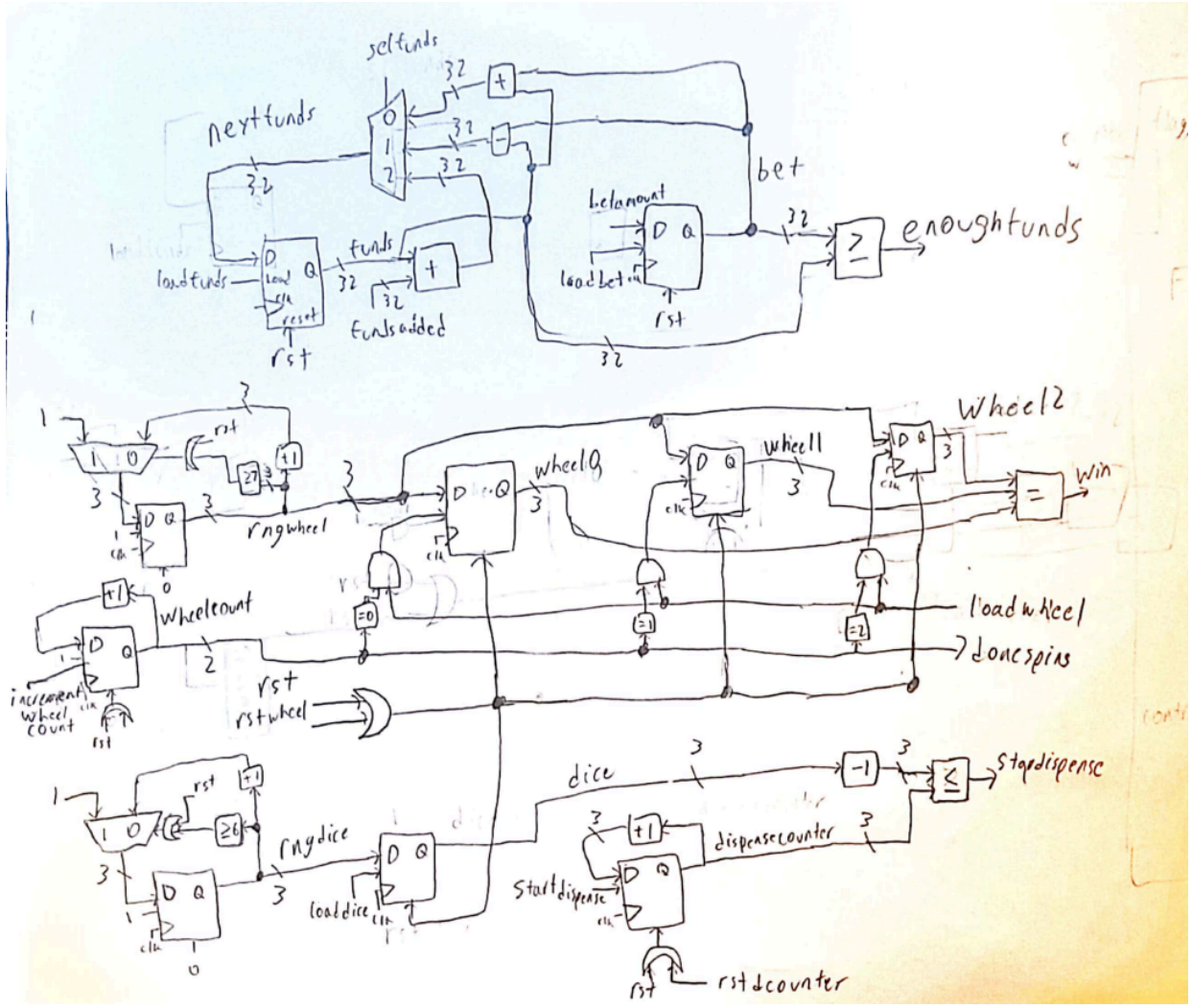
of credits into the funds register. ‘WAITRELEASEFUNDS’ will be an idle state which will check for the funds button to be released before continuing. ‘SPINWHEEL’ will lock in the bet and unlock the wheel corresponding with the amount of spins recorded in the spin counter register. ‘STOPWHEEL’ will lock the wheel register with its corresponding spin counter and will check if the wheel button is released, it will leave the loop once 3 spins are done. ‘INCREMENTSPINCOUNTER’ will check the win and will lead to a losing path or a winning path. ‘DECREMENTCREDITS’ will reduce the credits in the player’s funds if a loss is detected. ‘ROLLDICE’ will unlock the dice register. ‘STOPDICE’ will lock the dice register. ‘WINCREDITS’ will add the betted amount to funds and will loop based on the dice value since multiplication is not synthesizable.

gamblinggame_tb

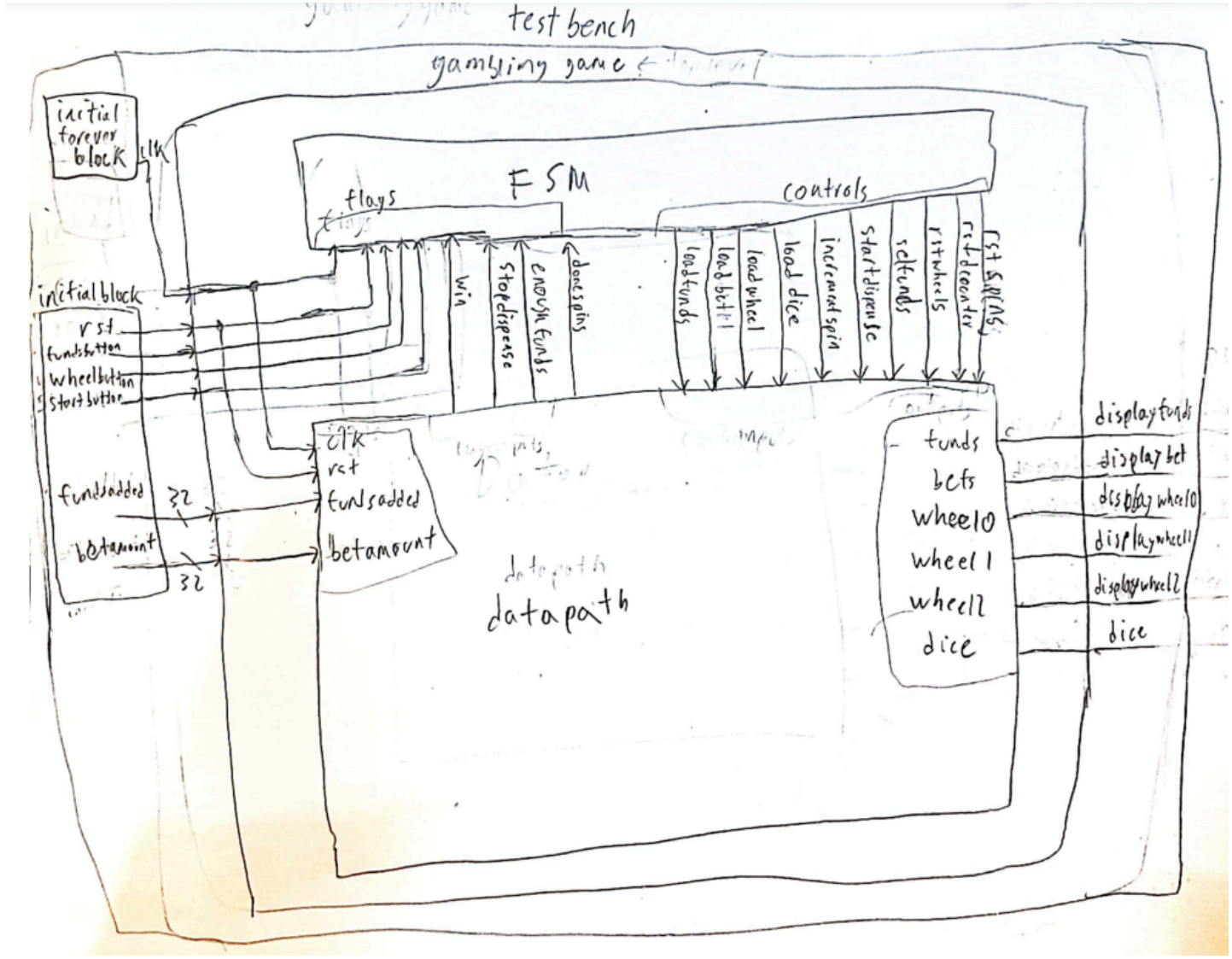
The testbench module is used to connect the inputs and outputs to logic elements to test the ‘black box’. The testbench itself has no inputs or outputs. Every input and output of the top level module is connected to a logic element in the testbench module. The testbench simulates the player changing values and pressing the buttons to operate the ‘slot machine’ implemented. The clock is generated by an initial forever block. The inputs are generated in the initial block. The testbench simulates the player adding funds, making a bet, starting the game with enough credits, a losing situation, starting the game with not enough credits, winning the game, and checks the resetting of counters once the game is finished.

Block diagrams of the datapath, top level datapath connections to FSM, and testbench connections to top level

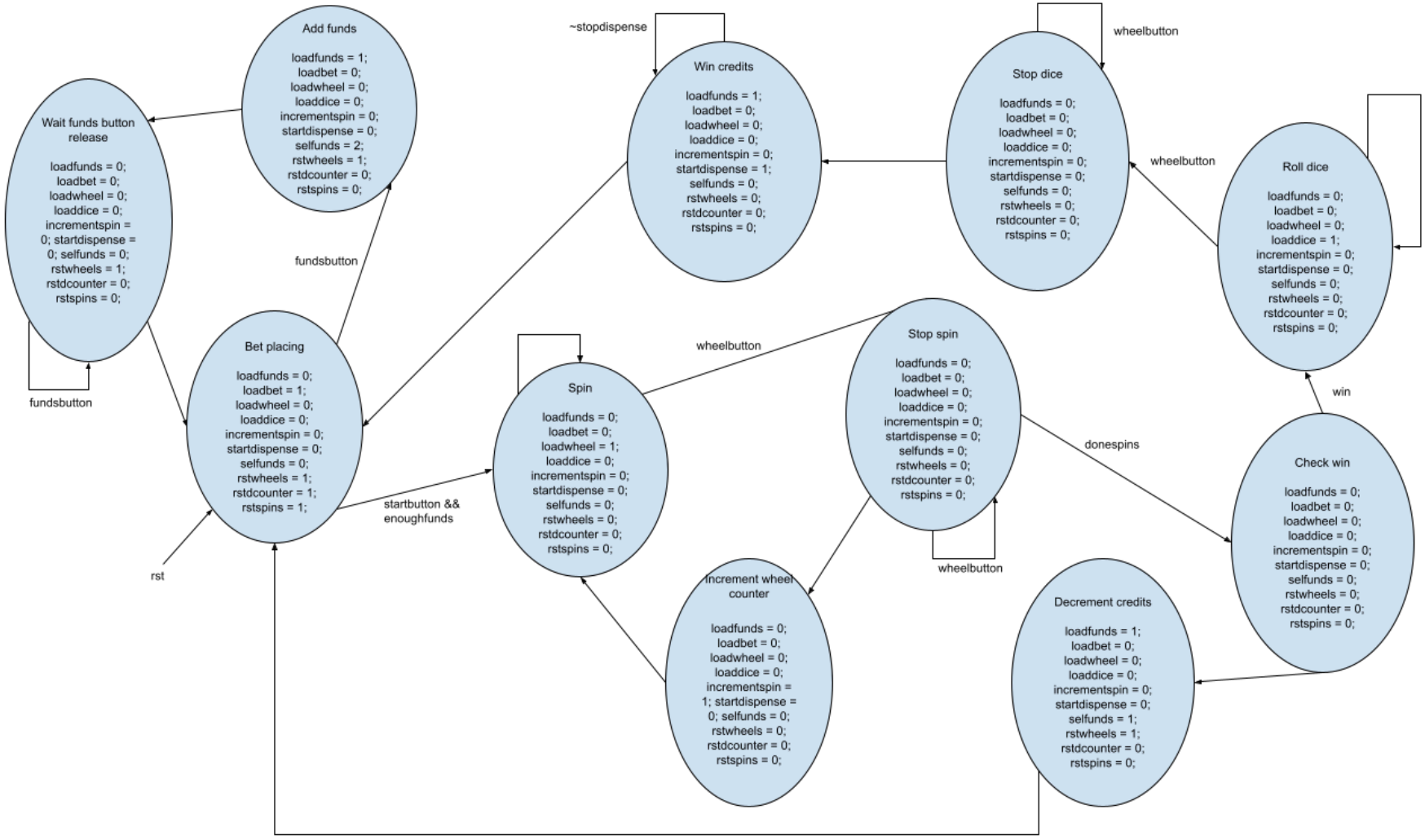
<u>The schematic of the datapath</u>



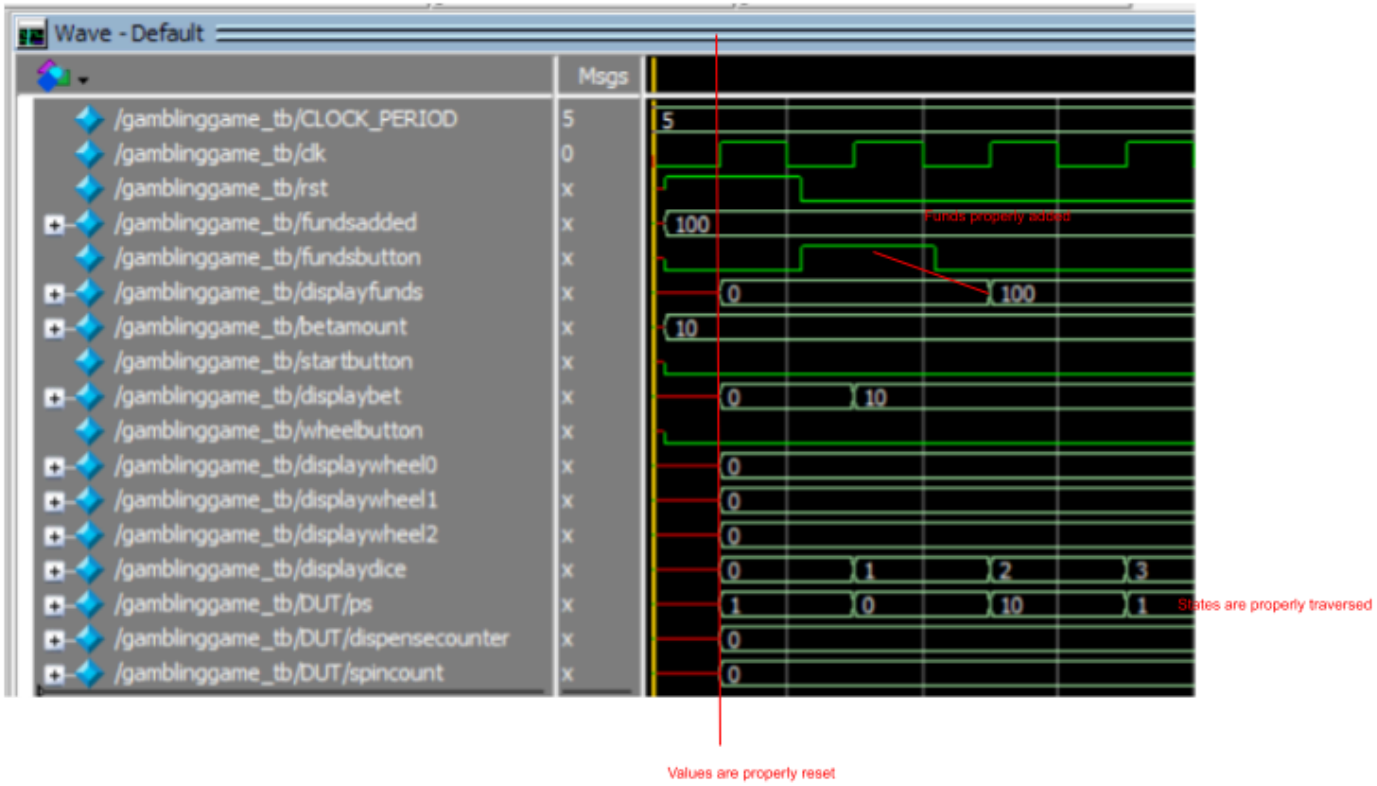
Schematic of the FSM connection to the datapath in the top level module, and the testbench connections



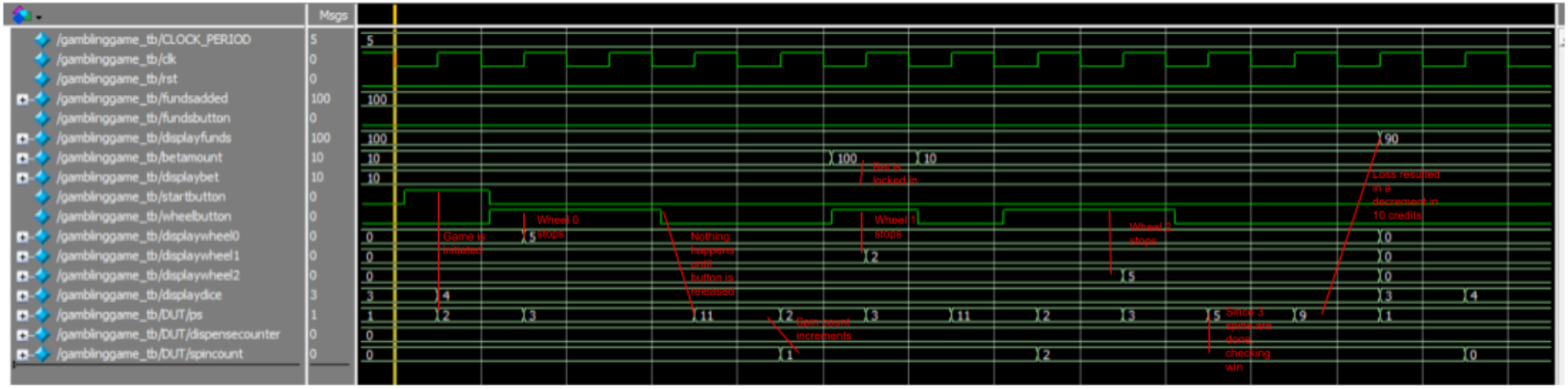
State diagram of the FSM with data flow



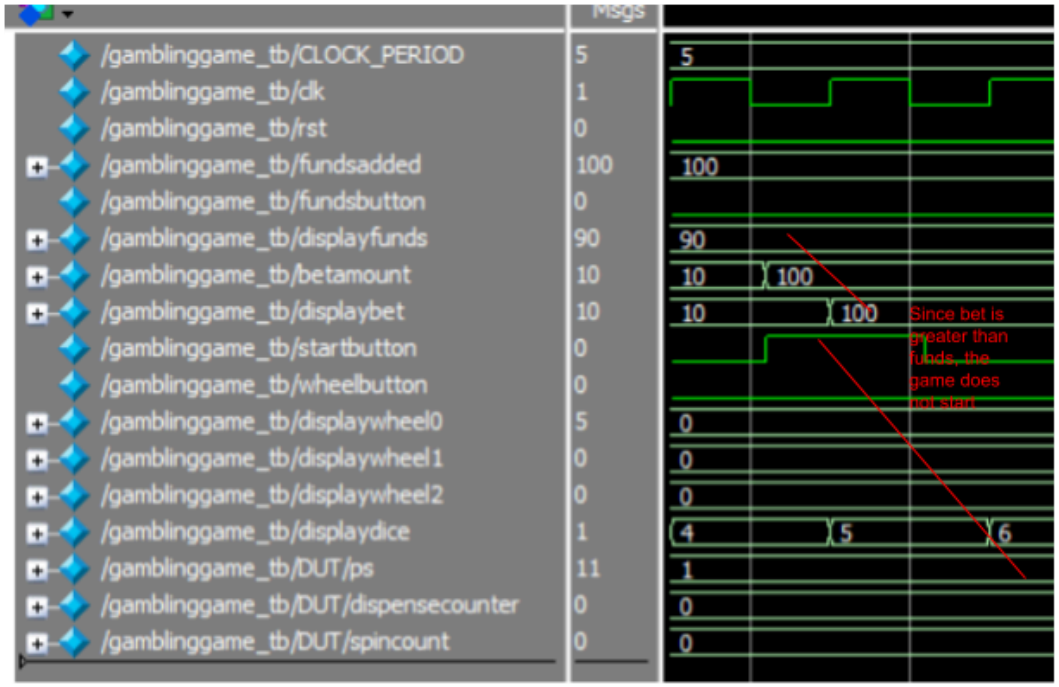
Results

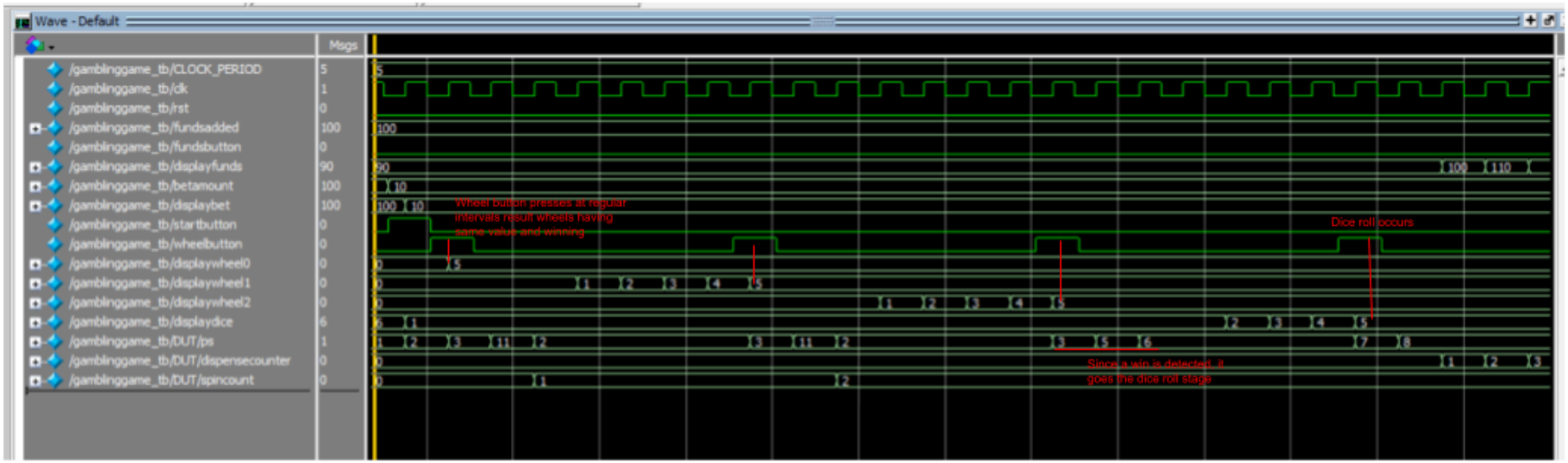


In this section of the waveform, ‘fundsadded’ is set to 100 and when the funds button is pressed, it goes from bet placing state to the 2 states that adds funds and checks if the button is released. ‘displayfunds’ is set to 100 and the state returns bet placing state. The bet amount is 10 in this state.



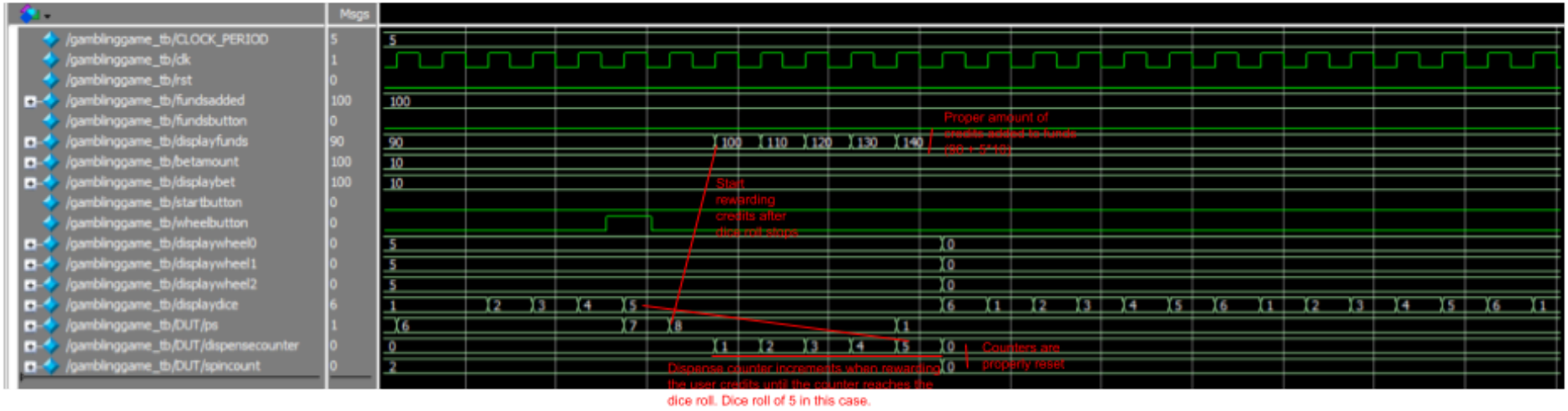
In this section of the waveform, whenever the wheel button is pressed, the wheels obtain the values 5, 4, 7. The FSM goes through the spin state to release button state, and to the increment counter state when the button is released. The amount of spins increment after each press and goes to the check win state once all three wheels are set. Since it is not a win, the state goes to decrement credits and the player loses 10 credits.


Msgs
/gamblinggame_tb/CLOCK_PERIOD
/gamblinggame_tb/clk
/gamblinggame_tb/rst
/gamblinggame_tb/fundsadded
/gamblinggame_tb/fundsbutton
/gamblinggame_tb/displayfunds
/gamblinggame_tb/betamount
/gamblinggame_tb/displaybet
/gamblinggame_tb/startbutton
/gamblinggame_tb/wheelbutton
/gamblinggame_tb/displaywheel0
/gamblinggame_tb/displaywheel1
/gamblinggame_tb/displaywheel2
/gamblinggame_tb/displaydice
/gamblinggame_tb/DUT/ps
/gamblinggame_tb/DUT/dispensecounter
/gamblinggame_tb/DUT/spincount
5
1
0
100
0
90
10
10
0
0
5
0
0
0
1
11
0
0
5
100
90
10
100
10
100
Since bet is greater than funds, the game does not start
0
0
0
4
5
6
1
0
0



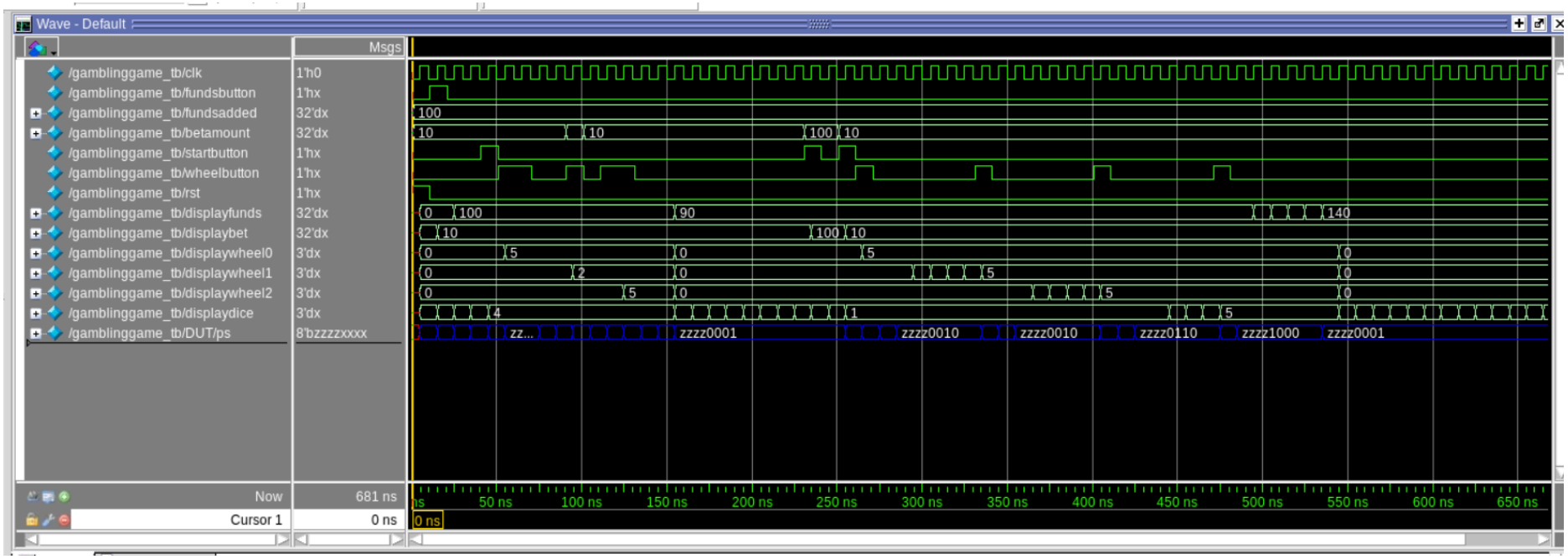
The game does not start when the start button is pressed because bet is set to 100 but only 90 credits in funds.

Since the wheels have the same value of 5, the game detects a win and a dice roll occurs. After pressing the stop wheel button, the dice stops.



Player starts winning credits after the dice roll. Since dice roll is 5 and bet is 10, the player is expecting to win 50 credits. The states go from stop dice to dispense credits to place bets. The counters and wheels are properly reseted.

Synthesized output



The same outputs are observed. 100 credits are added when ‘fundsbutton’ is pressed. Wheels stop when ‘wheelbutton’ is pressed. 10 credits are lost when the wheels stop at 5,2,5. 10 credits are won 5 times when wheels are 5,5,5 and dice roll is 5. The present state has tristate buffers, but does not appear to cause any issues with the outputs. My assumption is that I have used an 8 bit number for the present state logic, but only 4 bits are needed as there are only 12 states.

RTL compiler report

```
============================================================
  Generated by:           Encounter(R) RTL Compiler RC14.13 - v14.10-s027_1
  Generated on:           Oct 05 2023  06:47:33 pm
  Module:                 gamblinggame
  Technology library:     NanGate_15nm_OCL revision 1.0
  Operating conditions:   worst_low (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

      Instance        Cells  Cell Area  Net Area  Total Area  Wireload
-------------------------------------------------------------------------
gamblinggame            748        359         0         359    <none> (D)
  credits                99         57         0          57    <none> (D)
  betting                99         57         0          57    <none> (D)
  sub_29_41              96         46         0          46    <none> (D)
  add_29_52              33         37         0          37    <none> (D)
  add_29_30              33         37         0          37    <none> (D)
  gte_30_33              95         24         0          24    <none> (D)
  fundsmux               97         24         0          24    <none> (D)
  generatewheel          12          7         0           7    <none> (D)
    rnggendice            3          4         0           4    <none> (D)
    rngthresholddice      5          1         0           1    <none> (D)
  generatedice           14          6         0           6    <none> (D)
    rnggendice            3          4         0           4    <none> (D)
    rngthresholddice      5          1         0           1    <none> (D)
  wheel2ff               12          6         0           6    <none> (D)
  wheel1ff               12          6         0           6    <none> (D)
  wheel0ff               12          6         0           6    <none> (D)
  diceff                 12          6         0           6    <none> (D)
  dispensecountff        10          6         0           6    <none> (D)
  spincountff             7          4         0           4    <none> (D)

 (D) = wireload is default in technology library
```

Layout