Project 1 - SystemVerilog FSM
A simplified credit card payment controller

September 23, 2019

Arthur Hsueh
21582168
UBC - ELEC 402

Contents

List of Figures

1 FSM General Description

My finite state machine (FSM) is a simplified application of a credit card payment controller. It accepts payment in the form of VISA, MasterCard or AMEX, and includes basic operations that one would see when paying with a credit card.

The state machine moves through the states of card association selection, confirmation of payment, pin processing and processing the actual payment. It also includes state transitions when confirmation of payment, pin processing and actual payment processing fails. As a more unique feature, I added in a fail counter for the pin processing, allowing the state machine to abort processing if the user enters their pin incorrectly too many times.

The FSM assumes a more modular design, and functions as a controller to the whole system. The unique inputs of the FSM are bits that trigger transitions to new states, and the outputs of the FSM are bits that function as enables to other modules.

2 FSM Design

The SystemVerilog code for the credit_card_payment_fsm is found in the file credit_card_payment_fsm.sv.

The inputs of the FSM are as follows:

clk	Bit width = 1. The basic clock to drive the always_ff block of the FSM
reset	Bit width = 1. Synchronous reset for the FSM. We don't an asynchronous reset to prevent payment errors from abrupt resets.
process_init	Bit width = 1. Triggers the start of the payment process, and initiates movement through the FSM.
visa_choice_in	Bit width = 1. An enable indicating the user has chosen to use VISA as payment. Only one of the choice bits should bet set at a time.
mastercard_choice_in	Bit width = 1. An enable indicating the user has chosen to use MasterCard as payment. Only one of the choice bits should bet set at a time.
amex_choice_in	Bit width = 1. An enable indicating the user has chosen to use AMEX as payment. Only one of the choice bits should bet set at a time.
pymt_amt_conf	Bit width = 1. A flag indicating that the user has accepted the amount of their purchase
pymt_amt_denied	Bit width = 1. A flag indicating that the user has denied the amount of their purchase
pin_fail	Bit width = 1. A flag indicating the user has entered their pin incorrectly
pin_fail	Bit width = 1. A flag indicating the user has entered their pin correctly
transaction_fail	Bit width = 1. A flag indicating the credit card failed to process
transaction_success	Bit width = 1. A flag indicating the credit card was successfully processed

The outputs of the FSM are as follows:

light_bit	Bit width = 1. Turns a general light on or off, indicating whether or not the process (or FSM) is running. Logic 0 indicates the light is off, an logic 1 indicates the light is on.
card_choice	Bit width = 2. Indicates to other modules the choice of card. 2'b00 represents no choice, 2'b01 represents VISA, 2'b10 represents MasterCard and 2'b11 represents AMEX.
pymt_amt_print	Bit width = 1. Initiates the external processes of printing the payment amount to the user
pin_process_init	Bit width = 1. Initiates the external processes of checking the user's pin
pymt_process_init	Bit width = 1. Initiates the external handling processes of the credit card payment
process_abort	Bit width = 1. Sends a stop signal to all other modules that depend on the FSM

The states of the FSM are all 12 bits wide. There are a total of 14 states, which requires only 4 state bits, but I am using a glitch free method taught in CPEN 311, where the outputs are driven directly by state bits. So, there are an additional 7 bits to drive outputs and 1 additional bit as a placeholder for future additional bits. The states are as follows:

idle	The process is idle and not processing any payments. The next state is triggered by the 'process_init' enable. Here, the light is turned off
init_process	The payment process has been initiated, and the light is turned on. The next state is triggered by the next clock
wait_credit_choice	The process waits for the user's choice in credit card. There are 3 possible next states, which are triggered by the 'choice' bits inputs.
choice_visa	The FSM has taken in a high 'visa_choice_in' bit, meaning the user has chosen VISA as payment. The 'card_choice' bits are outputted accordingly. The next state is triggered by the next clock.
choice_mastercard	The FSM has taken in a high 'mastercard_choice_in' bit, meaning the user has chosen MasterCard as payment. The 'card_choice' bits are outputted accordingly. The next state is triggered by the next clock.
choice_amex	The FSM has taken in a high 'amex_choice_in' bit, meaning the user has chosen AMEX as payment. The 'card_choice' bits are outputted accordingly. The next state is triggered by the next clock.
init_amount_confirm	The 'pymt_amt_print' bit is outputted logic 1 to initiate the process of informing the user of the amount they are charged. The next state is triggered by the next clock.

wait_amount_confirm Waits for confirmation from user. If the amount is confirmed ('pymt _amt_conf is logic 1), the next states of pin input are processed. If the amount is denied ('pymt _amt_denied is logic 1), the payment fails, and the FSM moves to a failed finish state.

init_pin_process The 'pin_process_init' bit is outputted logic 1 to initiate the process of handling the user's pin. The next state is triggered by the next clock.

wait_pin_process Waits for the user to input their pin. If the pin is input correctly ('pin_success' is logic 1), the FSM moves to the payment handling states. If the pin is input incorrectly ('pin_fail' is logic 1), an internal counter, 'fail_counter' is incremented and the pin process is re-initiated. The fail counter allows the user to incorrectly input their pin 3 times before the payment process fails.

init_payment_handle The 'pymt_process_init' is outputted logic 1 to initiate the actual payment handling of the credit card. The next state is triggered by the next clock.

wait_payment_handle Waits for confirmation of credit card handling. If the transaction fails ('transaction_fail' is logic 1), the FSM moves to a failed finish state. If the transaction is successful ('transaction_success' is logic 1), the FSM moves to successful finish state.

payment_success This is this successful finish state. This state can only be reached if all previous processes have successfully completed. The fail counter is reset to zero, and the FSM moves to the 'idle' state.

payment_fail This is the failed finish state. This state is a separate finish state, because the 'process_abort' bit is outputted logic one to notify all other processes to stop. This is needed because this state can be reached from multiple parts of the payment process. The fail counter is reset to zero, and the FSM moves to the 'idle' state.

3 FSM Testbench Design

The testbench declares the same inputs and outputs as the fsm as logic variables, each with an appended '_tb' for distinction, and are attached to an instantiation of the FSM with the name device-under-test (dut). The instantiation is shown below.

```
credit_card_payment_fsm dut(
    .clk                  (clk_tb),
    .reset                (reset_tb),
    .process_init         (process_init_tb),
    .visa_choice_in       (visa_choice_in_tb),
    .mastercard_choice_in (mastercard_choice_in_tb),
    .amex_choice_in       (amex_choice_in_tb),
    .pymt_amt_conf        (pymt_amt_conf_tb),
    .pymt_amt_denied      (pymt_amt_denied_tb),
    .pin_fail             (pin_fail_tb),
    .pin_success          (pin_success_tb),
    .transaction_fail     (transaction_fail_tb),
    .transaction_success  (transaction_success_tb),
    .light_bit            (light_bit_tb),
    .card_choice          (card_choice_tb),
    .pymt_amt_print       (pymt_amt_print_tb),
    .pin_process_init     (pin_process_init_tb),
    .pymt_process_init    (pymt_process_init_tb),
    .process_abort        (process_abort_tb)
);
```

An always block is set up to simulate a clock for the FSM, and its toggles the 'clk_tb' wire every 10 units of time. The testbench tests 3 different cases of traversing the FSM. The outputs of the FSM are enable bits so verification is assessed by comparing the output of the dut, which has been fed simulated inputs, to expected values. Two important variables within the dut are the 'fail_counter' and 'state' variables, so those are also tested in the testbench. Wire values are verified using the 'assert' keyword in the testbench.

```
assert (light_bit_tb === 1'b1) else $error("light_bit assert fail");
```

Details of the testbench and its simulated waveform will be discussed in section 7 - **FSM Simulation Waveform**

4 FSM Block Diagram

Below is the block diagram for the state machine.

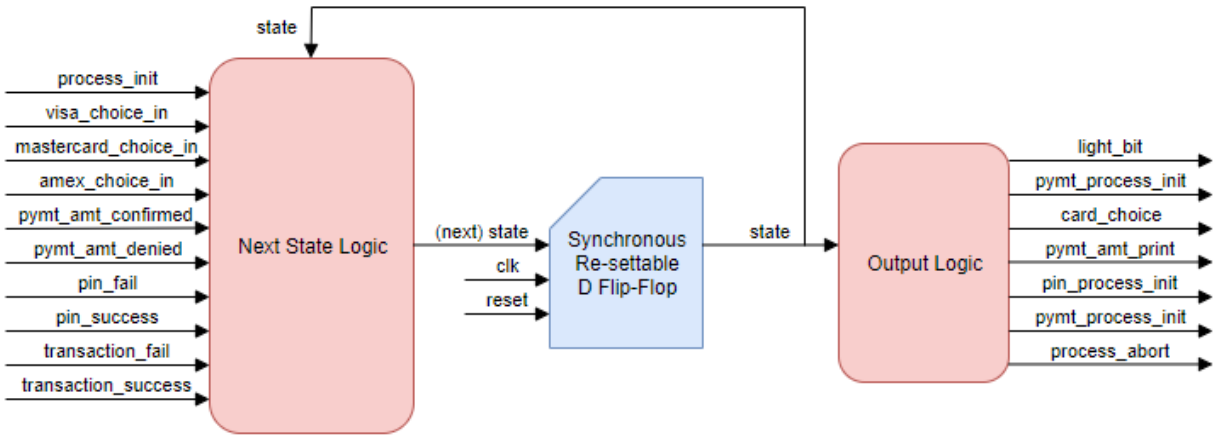

Figure 1: The block diagram for credit_card_payment_fsm

5 FSM and Testbench Block Diagram

Below is the block diagram of the FSM connected to the testbench.

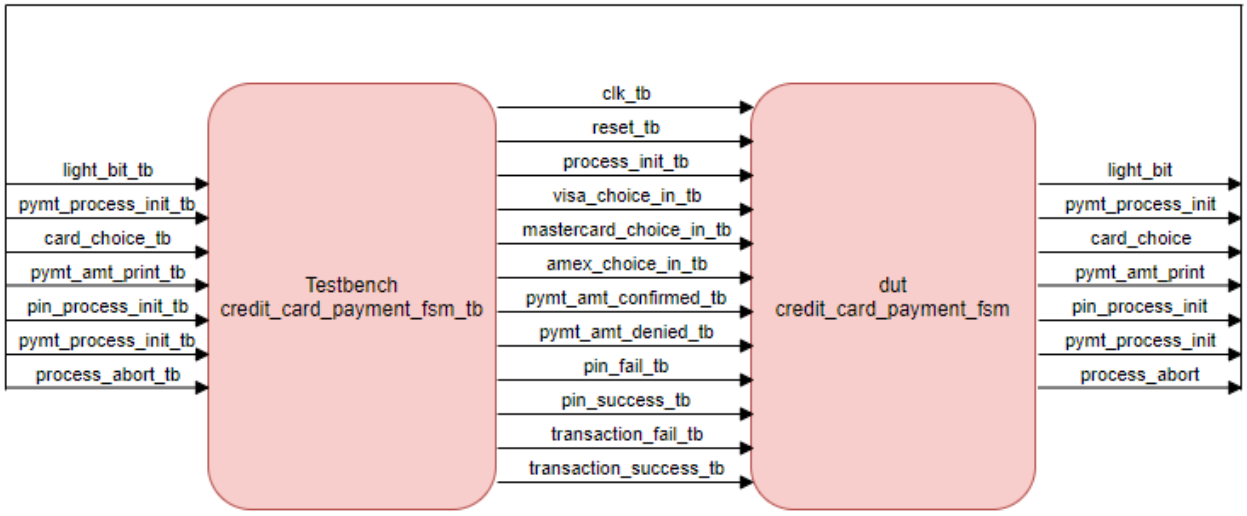

Figure 2: The block diagram for credit_card_payment_fsm_tb

6 FSM state diagram

Below is the state diagram of the FSM.

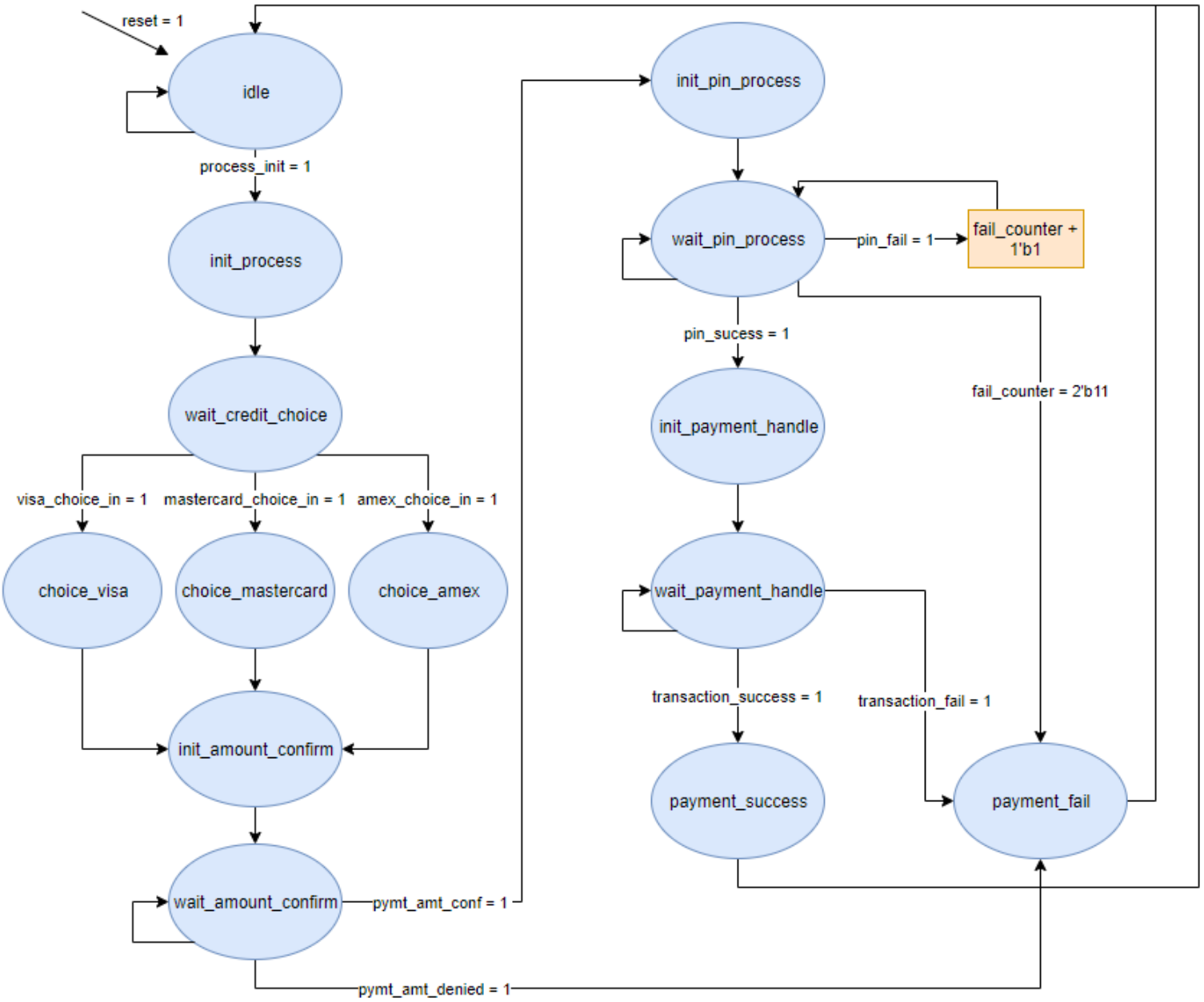

Figure 3: State diagram of the FSM

7 FSM Simulation Waveform

Below is an image of of the waveform after running the testbench. All logic buses declared in the testbench are added to the waveform. The **fail_counter** and **state** buses within the 'dut' module are added to verify the state variable and the fail counter for the pin.

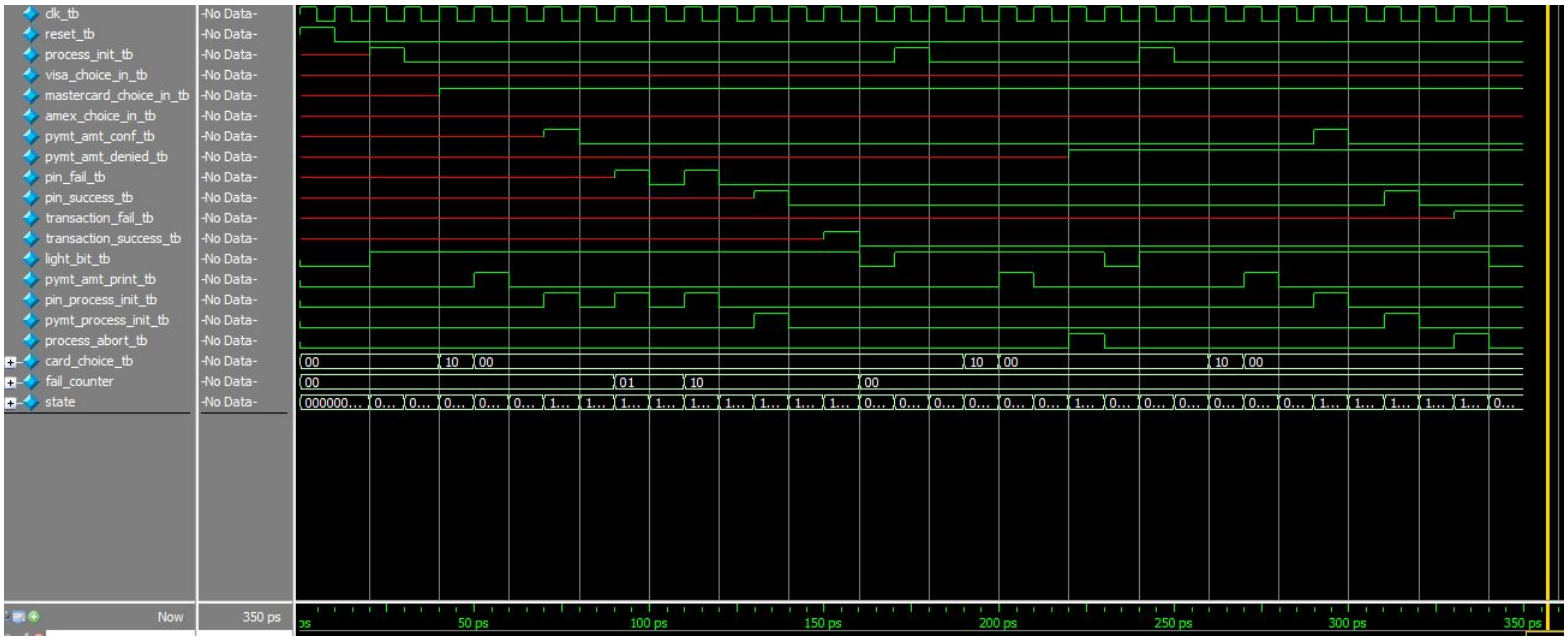

Figure 4: Testbenched waveform

As mentioned previously, the testbench tests three different cases of traversing the FSM. The first case tests a successful payment, which is also a successful traversal of the FSM, but with 2 failed pin entries. The second case tests a refused payment, in which the user denies the payment they are presented. The third case tests a failed transaction processing, in which the user accepts the payment and enters their pin correctly, but their card is denied. The three cases are different from each other, but are all in the same testbench, and always begin in the 'idle' state.

7.1 First FSM Loop

Figure 5: Waveform of the First FSM Loop

The simulation of the first FSM loop is within the time range of 0 to 160 picoseconds on the waveform, shown above. This loop simulates the FSM in the following manner:

1. The idle state is initiated by asserting 'reset_tb' for one clock cycle and de-asserting for another cycle.

2. On the next clock, 'process_init_tb' is asserted for one cycle, moving the FSM into the 'init_process' state. Another clock moves the FSM into the 'wait_credit_choice_state'. To verify correct FSM state movement, the 'light_bit_tb' is checked for output 1'b1.

3. The input 'mastercard_choice_in_tb' is asserted to simulate a user choosing to pay with MasterCard, and the FSM moves to the 'choice_mastercard' state. To verify correct FSM state movement, the output 'card_choice_tb' is checked for output 2'b10.

4. On the next clock, the FSM moves into the 'init_amount_confirm' state, to initialize printing of the payment amount to the user. The verify the correct FSM state movement, the output 'pymt_amt_print_tb' is checked for output 1'b1. The next clock moves the FSM into the 'wait_amount_confirm' state.

5. On the next clock, 'pymt_amt_conf_tb' is asserted for one clock cycle, simulating user confirmation of the payment amount. This moves the FSM into the 'init_pin_process' state. To verify correct FSM state movement, the output 'pin_process_init_tb' is checked for output 1'b1. The next clock moves the FSM into the 'wait_pin_input' state.

6. The 'pin_fail_tb' asserted for 1 clock cycle to simulate a user incorrectly inputting their pin. This is verified on the next clock by checking the internal variable 'dut.fail_counter'

for a value of 2'b01. This simulation of a failed pin input is repeated again in the next two clock cycles, and the 'dut.fail_counter' variable is checked again for an incremented value of 2'b10.

7. On the next clock, the input 'pin_success_tb' is asserted for one clock cycle, simulating correct user pin input. The FSM moves to the 'init_payment_handle' state on the next clock, initiating the external payment handling. To verify correct FSM movement, the output 'pymt_process_init_tb' is checked for output 1'b1.

8. On the next clock, the FSM moves to the 'wait_payment_handle' state. The input 'transaction_success_tb' is then asserted for one clock cycle, moving the FSM into the 'payment_success' state. This simulates a successful overall transaction.

9. On the next clock, the FSM returns to the 'idle' state. To verify correct FSM movement, the internal variable 'dut.fail_counter' is checked for a reset value of 2'b00.

7.2 Second FSM Loop

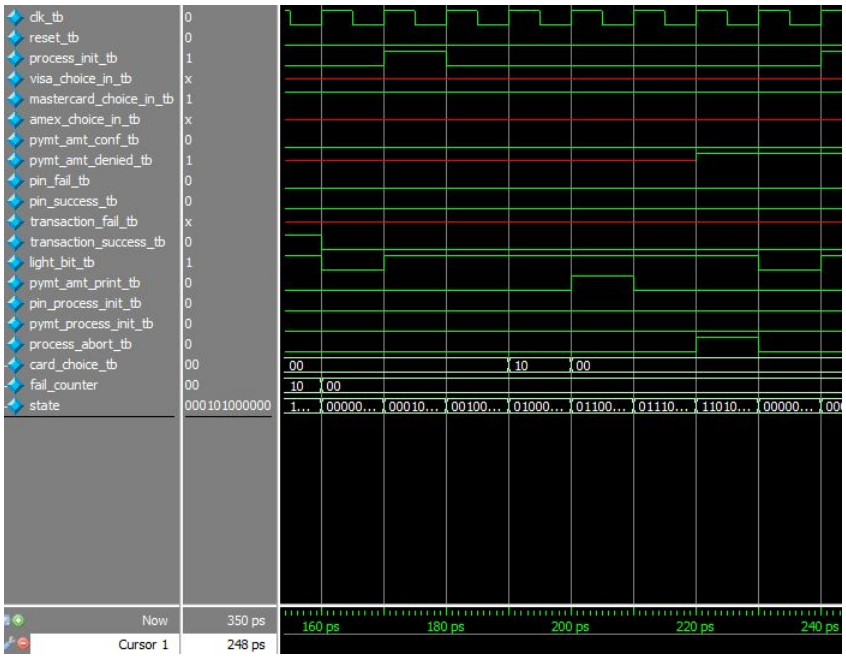

Figure 6: Waveform of the Second FSM Loop

The simulation of the second FSM loop is within the time range of 160 to 240 picoseconds on the waveform, shown above. This loop simulates the FSM in the following manner:

1. The first portion of the loop is identical to processes 2 to 4 of the first loop, stated in the previous subsection. The main test of this loop is for user denial of payment. After the previous steps, the FSM is in the 'wait_amount_confirm' state which waits for user conformation of the payment amount.

2. The input 'pymt_amt_denied_tb' is asserted for 1 clock cycle, simulating user denial of payment. This moves the FSM into the 'payment_fail' state. To verify correct state movement, the output 'pin_process_init_tb' is checked for an output value of 1'b0 (which would be 1'b1 if the user had confirmed the payment amount and the FSM moved to the next state).

3. On the next clock cycle, the FSM returns to the 'idle' state. Unlike the first FSM loop, where the FSM movement could be verified with the 'dut.fail_counter' reset, the counter value did not change whatsoever this loop. Instead FSM movement is verified by checking the internal variable 'dut.state' for a correct value of 12'b000000000000.

7.3 Third FSM Loop

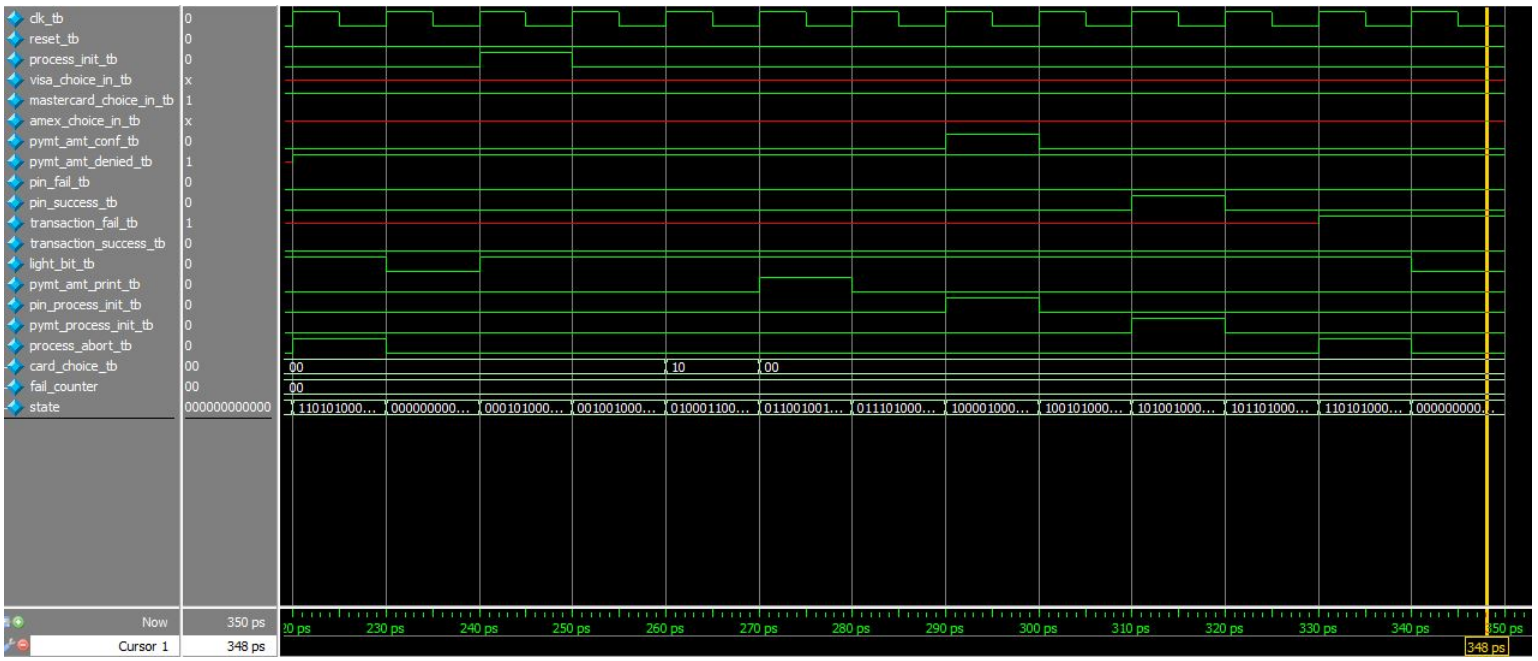

Figure 7: Waveform of the Third FSM Loop

The simulation of the second FSM loop is within the time range of 240 to 350 picoseconds on the waveform, shown above. This loop simulates the FSM in the following manner:

1. The first portion of the loop is identical to processes 2 to 7 of the first loop, stated in the first loop subsection. The one difference is that there are no pin fails, and instead simulates the user entering their pin correctly on the first try. The main test of this loop is for a failed transaction process. After the previous steps, the FSM is in the 'wait_payment_handle' state which waits for user conformation of the payment amount.

2. The input 'transaction_fail_tb' is assert for 1 clock cycle, simulating a failed credit card processing for the transaction. This moves the FSM into the 'payment_fail' state. To verify correct state movement, the output 'process_abort_tb' is checked for an output value of 1'b1.

3. On the next clock cycle, the FSM returns to the 'idle' state. Unlike the first FSM loop, where the FSM movement could be verified with the 'dut.fail_counter' reset, the counter value did not change whatsoever this loop. Instead FSM movement is verified by checking the internal variable 'dut.state' for a correct value of 12'b000000000000.