Name: Chris Baker
UID: 105.180.929

CS M152: Project 5

Parking Meter

Intro

A Finite State Machine can be used to provide the structure for many different electrical systems. As implied, there is a limited number of states and depending on the machine the outputs of a state could either depend on the current state or both the current state and the input. A Moore machine represents the former while a Mealy machine represents the latter FSM.

An example provided to us in the last lab was the turnstile Mealy Machine which transitioned between the locked and unlocked state based on inputs and the current state. Like a real-world turnstile a coin inserted into the machine would unlock it and once the turnstile was reset or pushed one rotation to allow someone through, then it would go back to the locked position.

Our lab is focused on designing a vending machine FSM and implementing the Verilog code to make it perform as our design intends it to behave. The inputs will contain add1, add2, add3, add4, rst1, rst2, rst, and clk. The expected outputs are val1, val2, val3, val4, a1, a2, a3, a4, and led_seg.

A big portion of this lab is built around the Seven Segment Display on the Nexus Board. This display uses a common anode to store individual 4 anode signals that determine if one of the 4 seven segments will display a number. There are also cathodes CA to CG that individually connect among the seven segments to determine which leds should be lit (segments).

In the initial state we are expected to flash 0000 with a period of 1 second and a 50 percent duty cycle. This means our display should start off showing 0000 for .5 seconds and then not showing it for .5 seconds. If we select any of the rst options (rst, r16, r150) then we will set the time value to either 0 , 16, or 150, but displayed in the 4 seven segments as decimal values. If we do not reset, then we can use the add buttons to add 60, 120, 180, or 300s to our timer. We can increase to the max value of 9999 seconds where we will be forced to have a ceiling of 9999 whenever we try to pass that value. Every second we are also decrementing our counter so we will eventually fall in time without using the add button. Upon reaching less then 180 seconds we will start to flash every other second displaying even values until we get to 0000 and the initial state.

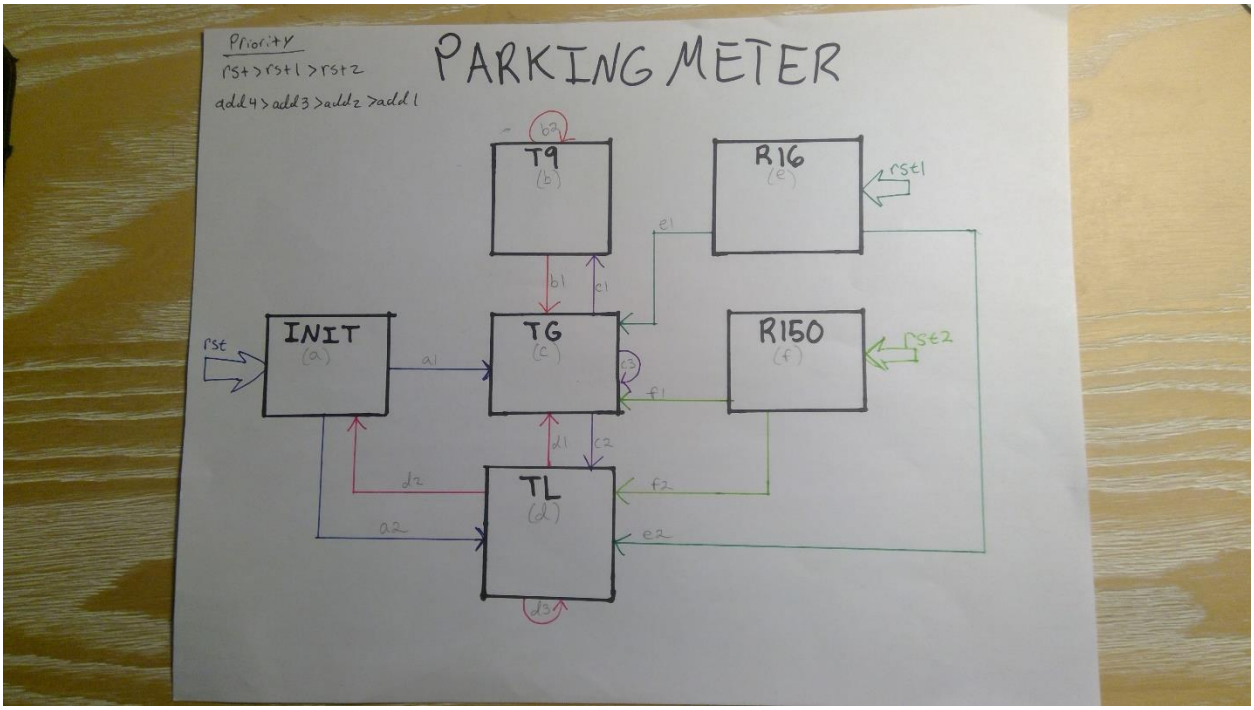
Again, any of these states can be interrupted by a reset signal from any of our three reset states. All state transitions will occur on the next posedge of the clock and several clocks will be created to give us a 10ms clock, .5s clock, 1s clock and 2s clock to use in determining values at various times.

Design Description

FSM Diagram: Parking Meter

Name: Chris Baker

UID: 105.180.929



1. FSM Notes/Legend:

- Refer to the table below for conditions organized by letter and number
- The **Block Arrows for INIT, R16, R150** show that **any state** can have a reset (Rt) **signal of 1 and move to the state**.
- **Priority: INIT>R16>R150**
    - Since we expect to move to only 1 state and the RST value has the biggest possible displacement in terms of seconds from any time value. Personal design choice.
- **Priority: add4>add3>add2>add1**
    - Since we expect no multiple add pushes, but the highest value denomination is set up to be the most meaningful. Personal design choice.
- **INIT** is the initial state

2. Table Notes:

- The variable name it is high or 1
- ! indicates it is low or 0
- Some names/conditions are abbreviated
- Addx means any of the add buttons.
- Value represents the corresponding time value that would be added for the addx button that was pushed : Remember, there is a priority : add4 over add3 over… add1
- Timer is a counter that holds the time
- **Only on the posedge of 1s clock does timer decrement**. I did not show that in my simplified table below, but at every 1s the posedge of clock takes timer = timer – 1. Then the code can perform the correct logic to move to the next state at a posedge of 10ms. So this can occur very quickly without user noticing and makes this timing a negligible factor.

Name: Chris Baker
UID: 105.180.929

- **Resets are already taken into consideration for every state as first line**. So else statements imply there is no reset and may imply that no adds or other different values depending on previous conditions.
- **Differ time expressions shown below**: **9999** refers to the 4 decimal value representation. 180s refers to the 3 decimal digits added to timer.

Current	Conditions number	Conditions	Next State/Output
ANY STATE	NA: Applies to all	Rst	INIT/timer=0
		Rst1	R16/timer=16s
		Rst2	R150/timer=150s
INIT	A2	Add1	TL/timer=60s
	A2	Add2	TL/timer=120s
	A1	Add3	TG/timer=180s
	A1	Add4	TG/timer=300s
TL	D1	(addx) & timer + value > 179	TG/timer=timer+value
	D2	Timer == 0 &!(addx)	INIT/timer = 0
	D3	Else	TL/timer = timer
TG	C1	Timer + addx > 9999	T9/ timer = 9999s
	C2	Timer = 179 & !(addx)	TL/timer = timer
	C3	Else	TG/timer=timer
T9	B1	!(addx)	TG/timer = timer
	B2	Else	T9/ timer = 9999s
R16	E2	!rst1	TL/timer = 16s
R150	F2	!rst2	TL/timer = 150s
	F1 E2	If the values are greater then 180	TG/timer = value of timer plus add

3. Explanation:

The FSM functions by starting with INIT as the initial state. From here we can move to any rst state (INIT, R16,R150) with the appropriate reset signal. For any of the states in my FSM; selecting a reset will cause a transition to those respective states at the next posedge of the clock. Further the resets set values to either 0000s, 16s, or 150s. Else we have the option to select an add button and proceed to either the TG (Greater or equal to) or TL (Less than 180) state. Once we are in the TL or TG state, we will look for add and reset signals. If the add signals are selected, then we will add the corresponding time value to our timer variable. At the same moment we are checking with a 1s period clock to perform decrements on our timer every second. Our main module uses many variables and counters to create a 10ms clock, .5s clock, 1s clock, and 2s clock. We feed the time values into our seven-segment display module which then determines based on an internal clock when to change mini states within the module to decide the anode values. Further, we use a function to implement a mux decoder that will choose the led segments to be lit up depending on the 4-time values we passed into the submodule. The submodule returns the anode values and the decoded segment at that time. Should we ever hit 9999 or greater than we will transition to the 9999 state briefly. If no add button is pushed, we will head immediately back to TG state where we can wait to decrement and check for add signals to

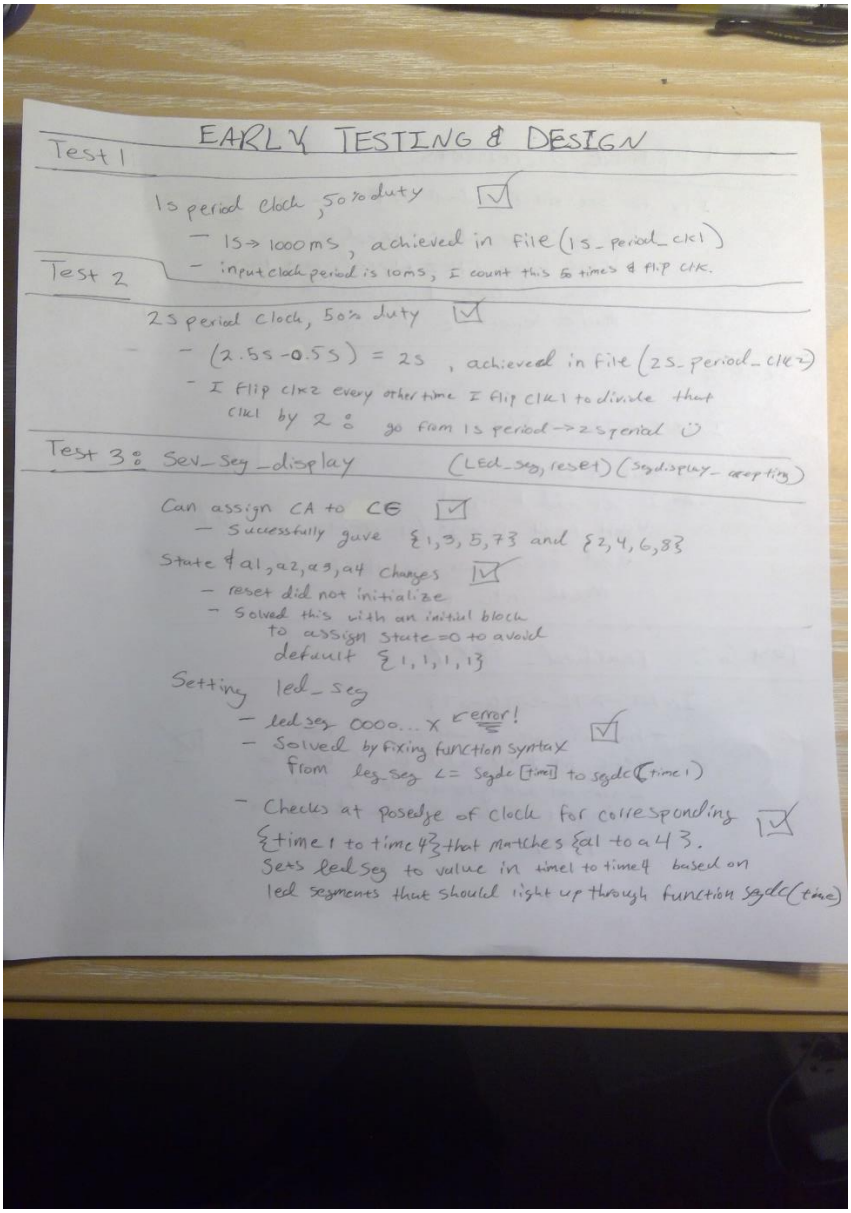
see if we need to return to 9999 for a brief moment. 9999 as well as the reset states can loop if fed the signal that keep triggering them (timer + addx > 9999s or keeping a rst signal high). Eventually though without inputs we count down and can head to the TG state. When we reach the TL state, we start to flash the time value on even numbers and display nothing on odd values. So, the time is showing every other second and giving the user a warning. Eventually we will reach the initial state where we flash the value every .5 seconds displaying 0000s.

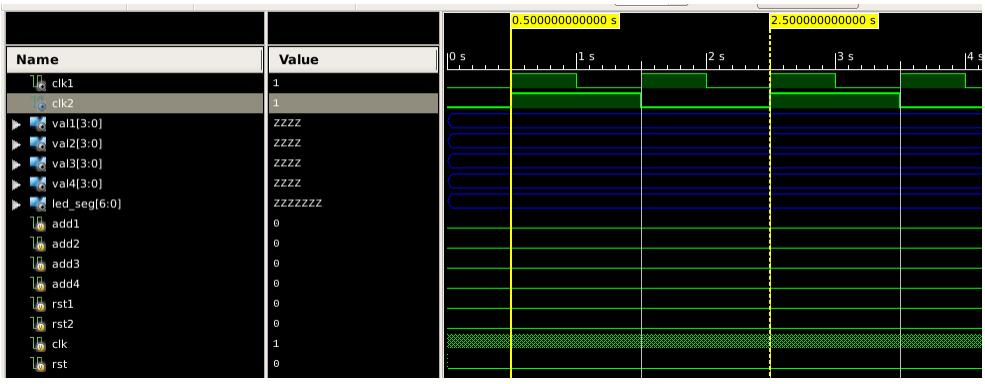
4. Methodology/Early Testing to achieve design:

Below is some early testing methodology that helped me largely go from skeleton code to achieving everything above. I started with an idea or goal and then tested what occurred or if I achieved it (checkmarks). I also wrote down things I learned, questions, or reasoning.

Name: Chris Baker

UID: 105.180.929



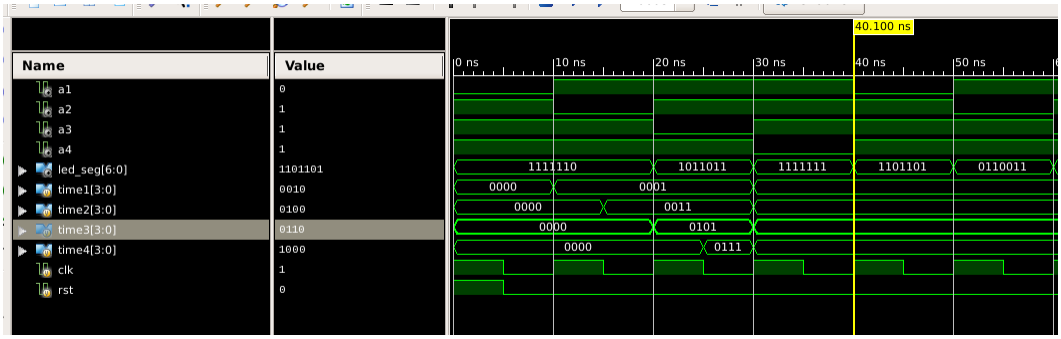


Name: Chris Baker

UID: 105.180.929

Achieved 1s and 2s period clocks

- Beginning step to make clk1 and clk2

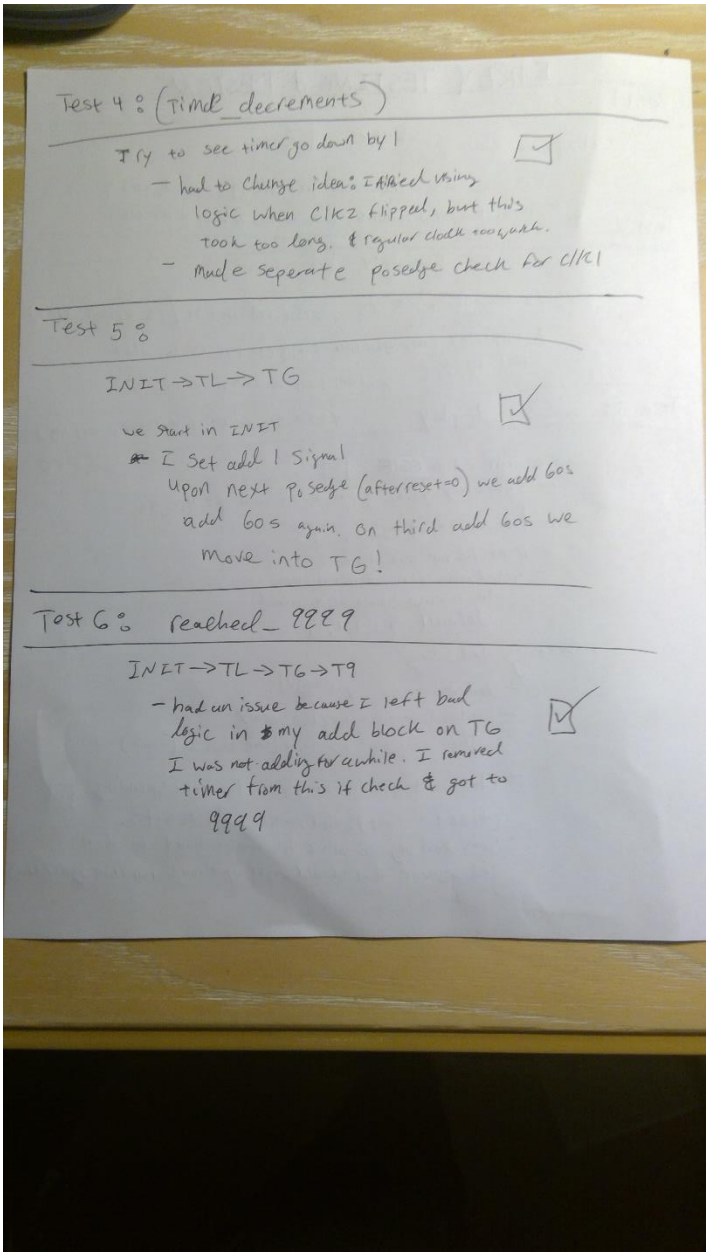


Achieved led_seg getting set correctly

- Make sure our sev seg display is taking the time values from main module.
- Also shows that led_seg is being set to certain values depending upon which leds should be lit. I started with a high signal setting , but in my TA discussion they suggested we try a low signal setting so I believe I flipped all the led seg values since then.

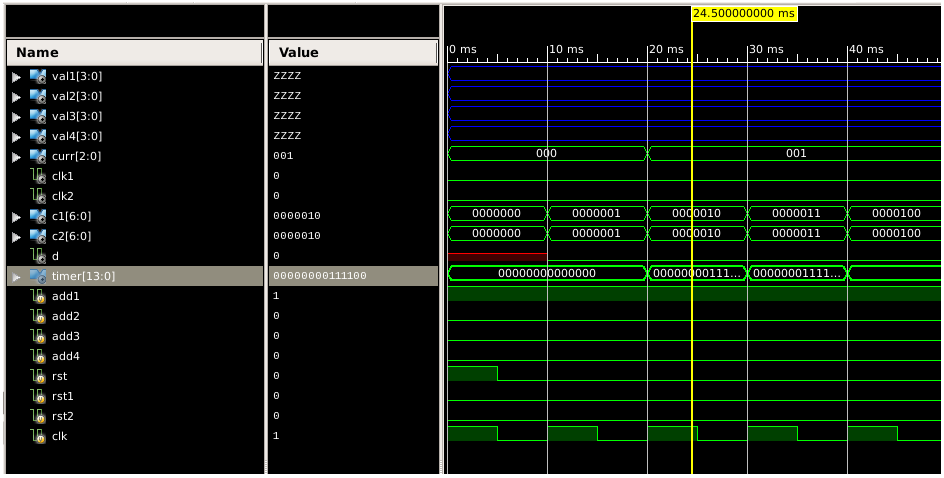
Name: Chris Baker

UID: 105.180.929



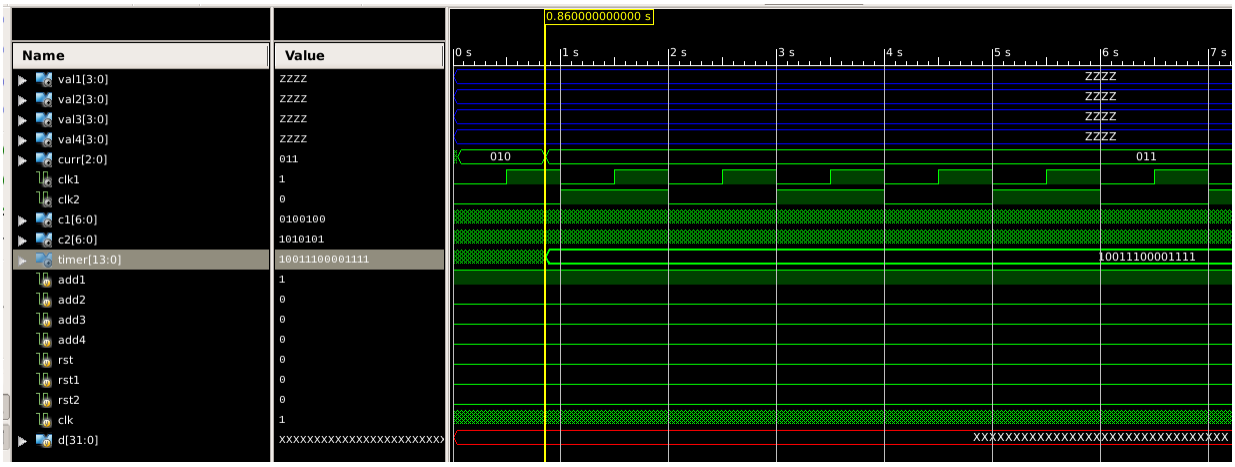
Name: Chris Baker

UID: 105.180.929



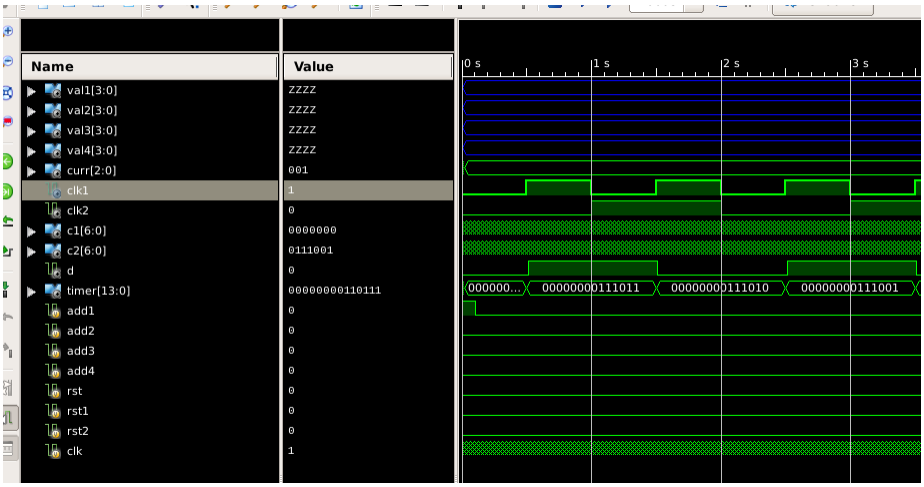
Went from Initial(000) -> TL(001) -> TG(010) states

- Simple test to make sure we are moving states correctly
- We also understand we move at posedge of clock.



Reached state T9(011)

- This demonstrated we could move from Initial to TL to TG to T9. Also passing consecutive add1 values into t9 resulted in staying at t9 and our timer value being set to a binary version of 9999s.

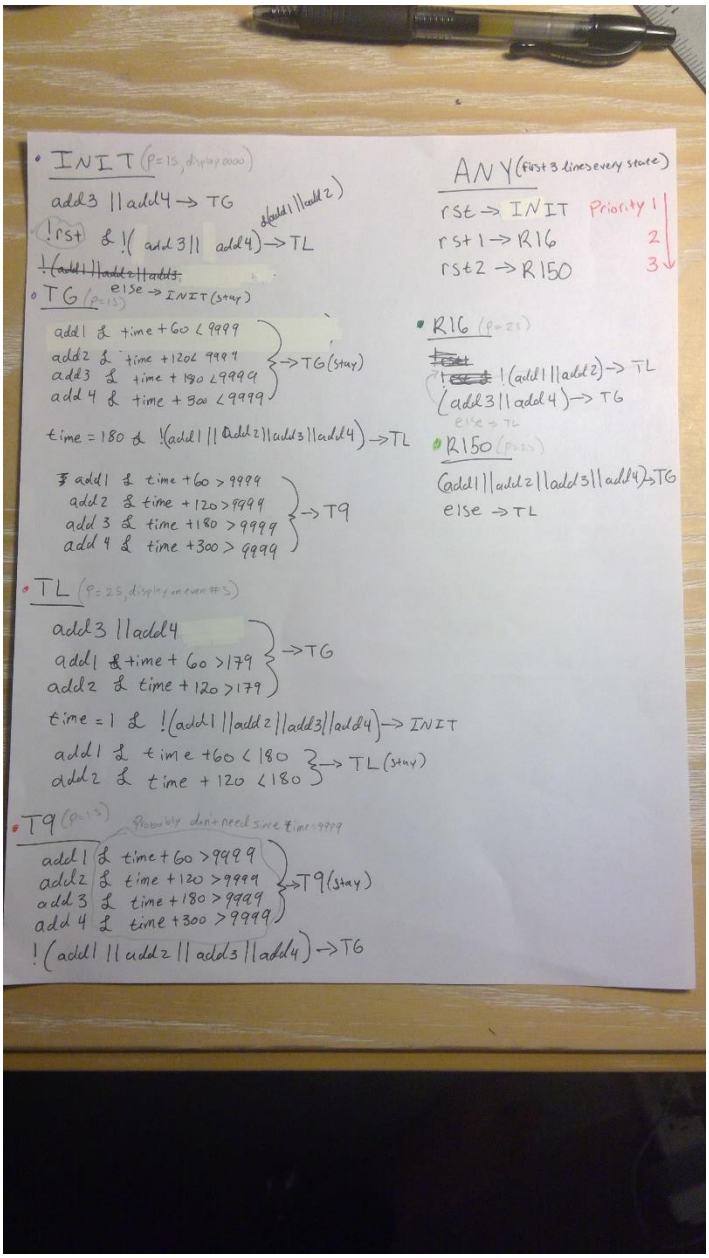


Name: Chris Baker

UID: 105.180.929

Decrementing

- Shows timer is dropping by 1

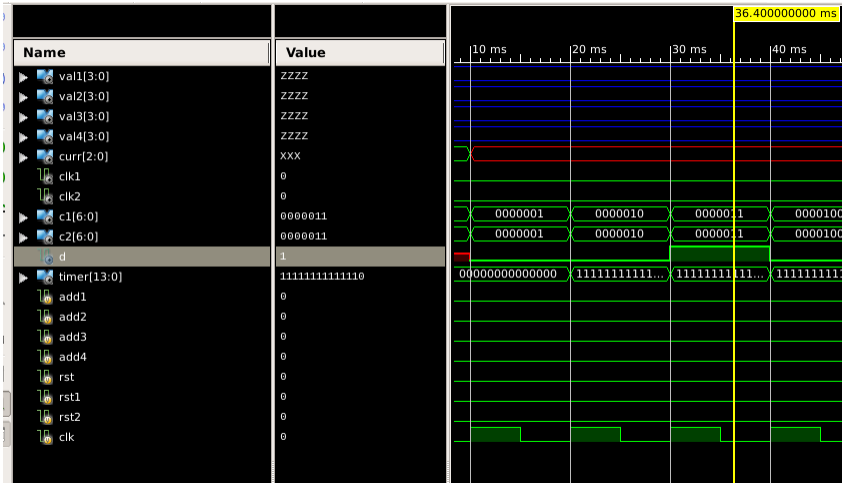


Some original notes and conditions on when to move. The table presented at the beginning is more recent and implements the logic cleaner. My code includes some redundant safety conditionals that are a little extra.

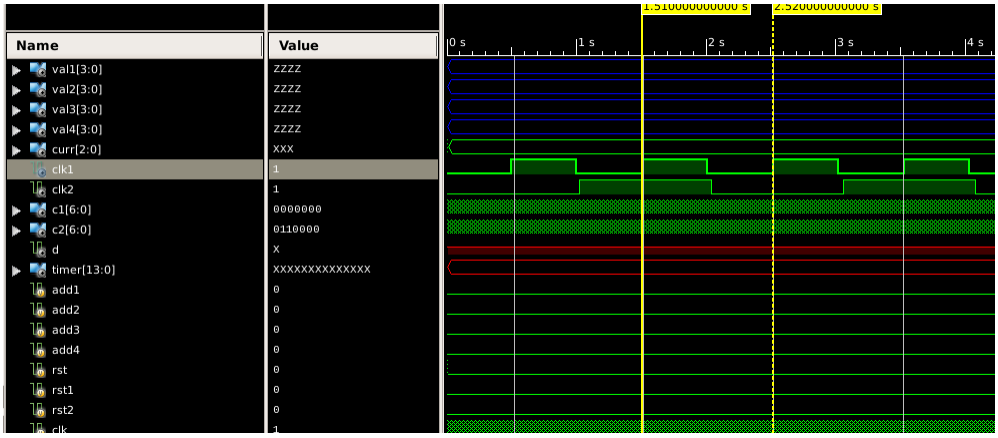
5. Errors:

Name: Chris Baker

UID: 105.180.929



One error I made above was not accounting for the decrement originally. So I had an underflow issue until I handled that by adding a logic check before decrementing. Now I do not decrement if at 0.



I realized that setting values simultaneously results in a race condition so I was getting incorrect values from undefined behaviors and this affected my clocks originally.

Schematics

I am connected a timer variable to multiple blocks to implement logic, but that has resulted in a failed schematic. I really need a passing grade on this lab to pass this class, so please do not dock me too harshly on this, or on other parts of the lab. I apologize, but the deadline to turn in is nigh and I can not fix this now.

I presume that a decoder would be used to represent the mux and function I used in the sev seg display. This could decode the values in the time variables and translate those to another mux which would then select the correct value. After that various combinational logic and many flip flops would appear in my rtl to perform counting and determine flips and clock edges. I presume the diagram would be very messy and with the timer variable being used in many places (probably due to me implementing a real time virtual timer variable instead of using test bench timer values or other simpler design?) that the diagram would be far too connected and complex.

Name: Chris Baker
UID: 105.180.929

However, we would see many muxes, some adders for counting and adding time in 60, 120, 180, and 300 increments, a decoder, and several flip flops.

Simulations and more Tests

I made several tests along the way, but **for grading purposes I recommend you see the highlighted bold test cases below and choose the ones you find most interesting**. This is because I included more tests then the recommended 1 working test and at least 4 for 80%. I tried to include a couple unique ones and several that are interesting with highlights. **Errors section included above as I checked testing and methodology for design.**

FULL SIMULATION COVERING MULTIPLE TESTS ASIDE FROM PREVIOUS TESTS(READ BOLD, unbolded text are details)

1. **LED SEG SET TO 11…1 on 500ms flips.**
2. **2. INITIAL STATE->TL with add1.**

The above photo tested the starting initial state (000). We also were able to count the 1s and 2s clocks correctly. Notice that at 520ms after our initial delay the led_set 7 bit was also set to 11..11 . This is because when we are under 180s we are in the TL(001) state and will check the

Name: Chris Baker

UID: 105.180.929

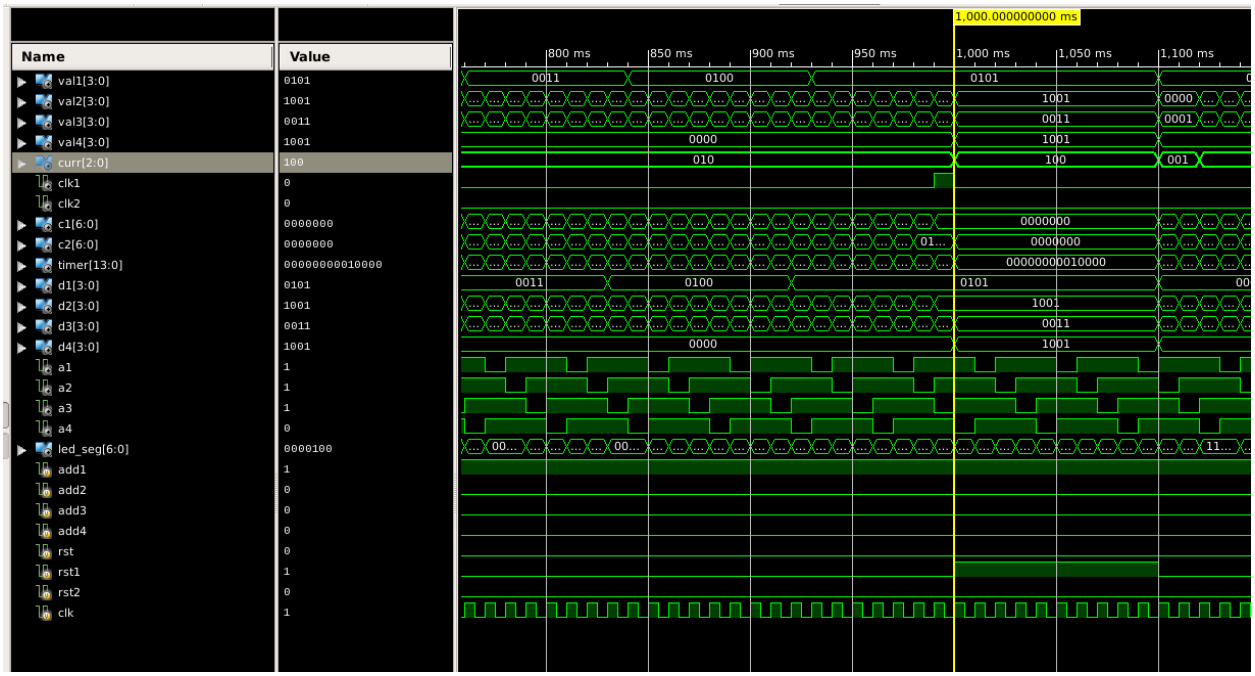
main clock for if 500ms have passed then flip a signal to either display the time or not in this state.

3. **INIT successfully resets to 000 state**
4. **Add1 works in adding 60s (other add buttons follow similar design and work for adding 120, 180, and 300 , they just transition differently : Example, Init state + add4 would make a automatic transition to TG skipping TL)**
5. **RST successfully resets to 000 state and output**
6. **TL->TG : No longer need to flash led_seg, it lives and changes freely and quickly**

At 500ms the rst signal goes low and allows us to move from INIT to TL. Once in TL we can move to TG after we used add1 3 times (=180s) this occurs at 520ms . I used 180s as a cutoff point for TL(001) to TG (010) transition .

Name: Chris Baker

UID: 105.180.929



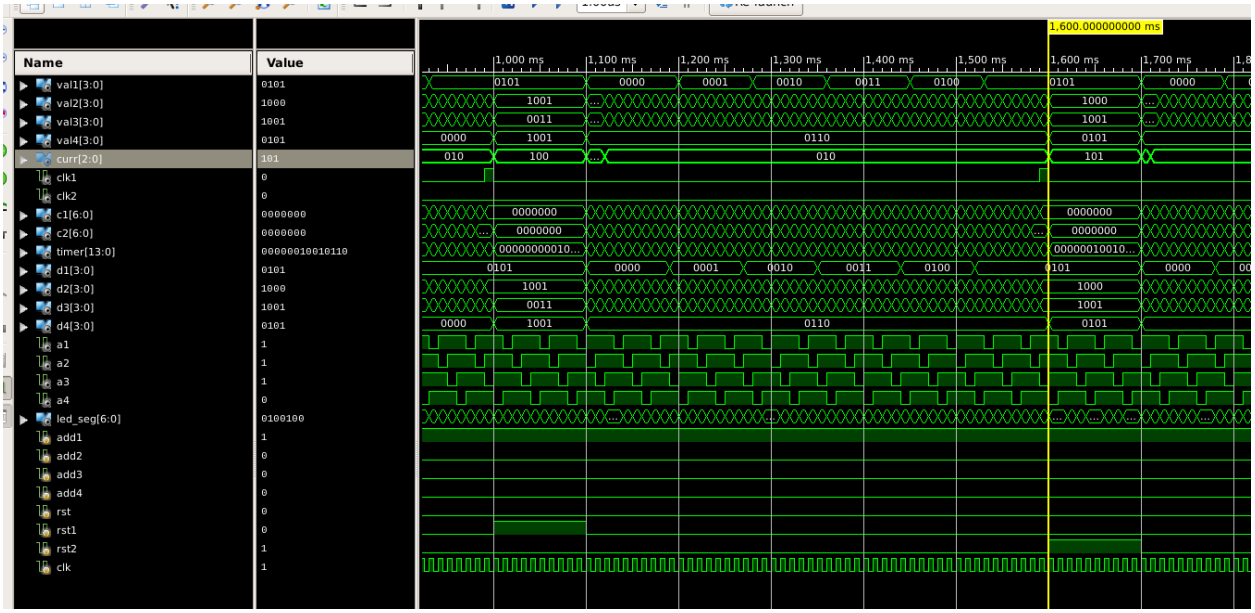
7. TG->R16 : R16 sets timer to 16

8. R16 ->TL->TG

R16(100) is then set and we move into that state at 1000ms. This causes the timer to be set to 16 shown in the timer slot following the yellow vertical line. This shows R16 works for transitioning when the rst1 signal is set. We go back to TL since our value is less then 180. At 1120ms we are back in TG since we add 60s 3 times exceeding the value of 180. Then we proceed to test R150 at 1600ms.

Name: Chris Baker

UID: 105.180.929



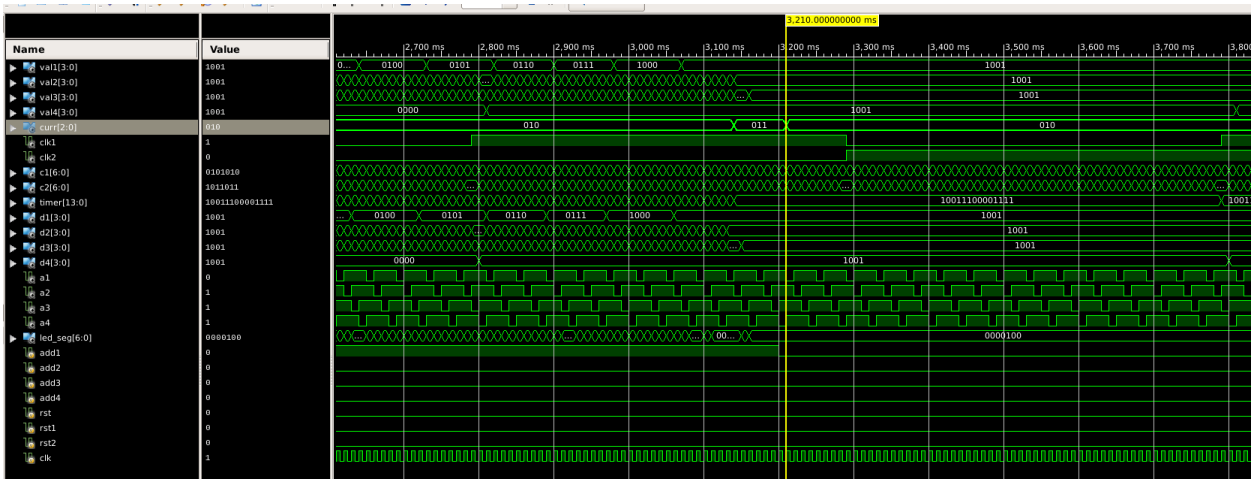
9. **R150 sets 150**
10. **R150 rst2 signal causes this transition.**
11. **Can move R150 -> TL or TG : TL without any reset or add input, TG without reset and one kind of add input**

Shown above we reached R150 at 1600ms with the signal going high. It is important to note that we also stay here while the signal is high. This happens for all reset signals. They will stay when high in that state. RST>RST1>RST2 in terms of priority also. At 1700ms we change back to TG after adding 60s to our timer immediately (I opted to allow an add to happen without a reset signal being on to allow a transition to desired TL or TG state) once exceeding the 180 number again.

Name: Chris Baker

UID: 105.180.929

12. LOOP Behavior resets and T9



Test moving to 9999 state, staying there while adding, and moving out when no add signal, and values are 9 inside the state (d and val).

At 2200 ms we stay in Init for 100ms until reset goes low showing the loop behavior of reset states and init and T9. At 2300 ms we go to TL, then to TG, and at 3140ms we are in T9 and stay in T9 until all add signals go low. Then we transition to TG upon a signal going low. The whole time we are also decrementing by 1 every 1 second clock period. Notice that led_seg also is stuck in the 9999 state with the d values each showing the value inside is 9 for each.

Name: Chris Baker
UID: 105.180.929

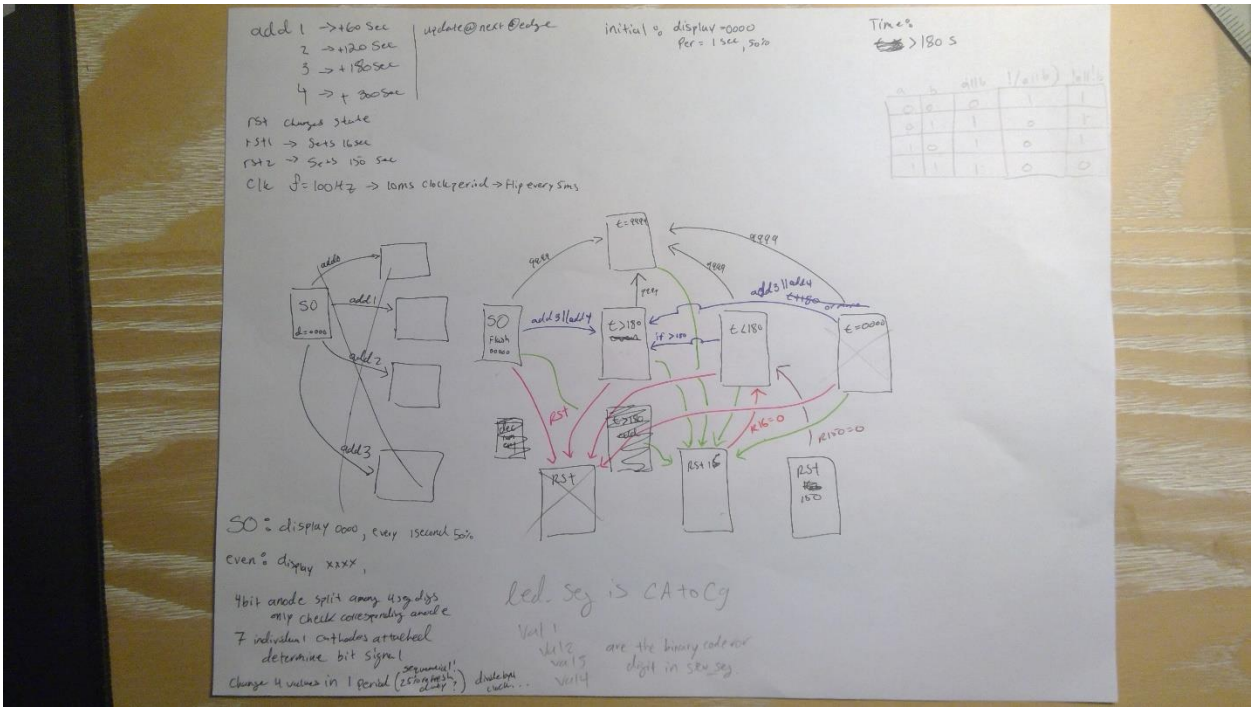
Please refer to earlier tests for more beginning tests that contributed to these results or demonstrate correctness.

Conclusion

Essentially this FSM Vending Machine performs multiple functions that resemble a real parking meter. It can add coin denomination values and flash accordingly to provide the user a warning. In implementing this machine, we created a state machine that could take the inputs of different coins and increment a time based on that. Then we applied several reset transition states for the user to move accordingly to setting values to 16, 160, or 0000. Along the way we also put in different flash logic in the seg module to simulate 1second flashing and .5 second flashing to warn the user when time was getting low, The reset states, 9999 state, and init state allow a worker to reset the machine and can keep looping that reset until that signal goes low.

]

In this lab the difficulties I encountered was trying to create the FSM and the code on my own. I originally designed this FSM.



But I quickly found out that setting an extra t=0000 state would be more work and another rst state would be redundant since the lab manual said to us the init state as a reset. My biggest issue was connecting ports correctly since I included so many ports and to also make sure that the various always blocks were not conflicting with each other. Several times I introduced race conditions and had to check and see if I tried assigning a register variable twice within the same block, but not exclusive to one

surrounding code block. This project was more time consuming than I anticipated, but it was also very interesting and tough to figure out. All in all it made it stressful and rewarding. I also made sure to test everything along the way because I originally scrapped my first plan based on that earlier fsm. Then I proceeded to test things one at a time as shown in my early testing above.