**Lab1 – Vending Machine**

1. **Introduction**

The aim of Lab1 is to design a Vending Machine as a Finite State Machine (FSM). The input and output signals and their descriptions are of the following:

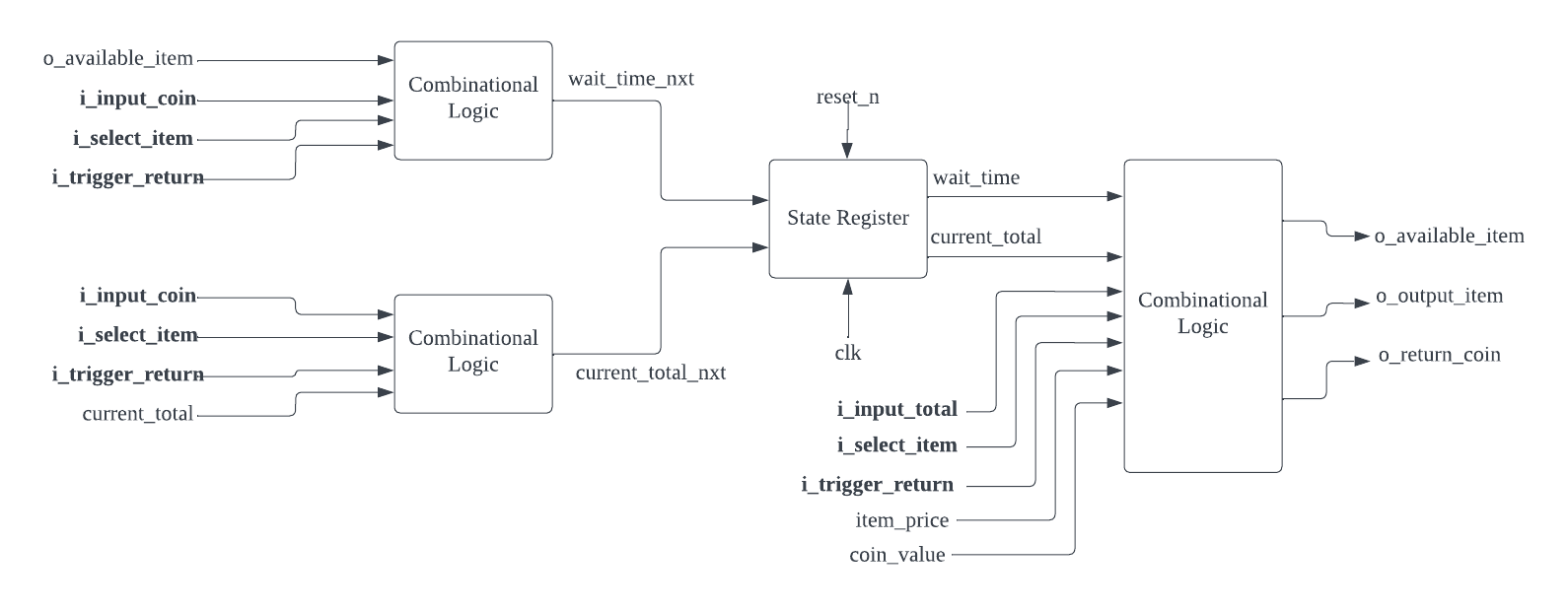
|  |  |
| --- | --- |
| i\_input\_coin | This is an array of wires length equal to the number of coins. In our case, this was of length 3 as there were 100KRW, 500KRW, and 1000KRW coins. When a coin is given to the machine, the corresponding wire in the array becomes HIGH for a specified duration of delay and goes back to LOW. |
| i\_select\_item | This is an array of wires length equal to the number of items. In our case, this was of length 4. |
| i\_return\_trigger | This wire signals the returning of all coins withheld in the Vending Machine. |
| clk | This is a clock signal that drives the state registers. |
| reset\_n | This reset signal resets the vending machine to the initial state – that is, resetting the wait\_time and current\_total. |

|  |  |
| --- | --- |
| o\_return\_coin | This is an array of wires that outputs the returned coins. |
| o\_available\_item | This is an array of wires that outputs the currently available items – that is, items whose prices are lower than current\_total |
| o\_output\_item | This is an array of wires that outputs the retrieved items. |

Our group’s design uses both wait\_time and current\_total as variables that denote the current state of the system. They are therefore updated synchronously by the posedge clk signal. There are, however, practically an infinite number of states as there is no upper bound to the amount of money that can be given to the system; consequently, drawing a Finite State Diagram is infeasible. The following sections aims to illustrate some of the design principals used for our system.

1. **Design**

Our vending machine is a **Mealy Machine** as the input signals are connected to both next state combinational logic block as well as the output combinational logic block. The figure below gives a high-level view of how wires the mapped to different modules in our design.



*Illustration of our Vending Machine design*

1. **Rules for wait\_time**

The initial value of wait\_time is defined in vending\_machine\_def.v as a constant kWaitTime. The default action for wait\_time is to decrement its value by 1 every clock cycle. And the value is reset to its initial value when 1) a user inserts a new coin, 2) selects an available item, or 3) i\_trigger\_return is turned on.

1. **Rules for current\_total\_nxt**

The internal variable current\_total and current\_total\_nxt store the total money the vending machine currently has – that is, **current\_total\_nxt = current\_total + input\_total – output\_total – return\_total**. The Next State combinational logic block takes care of updating the current\_total\_nxt.

1. **Implementation**

There are a number of modules that constitute the vending machine in junction. The below is a list of used modules and their descriptions.

|  |  |
| --- | --- |
| vending\_machine.v | This is the top module where all wires, ports, and submodules are declared. |
| vending\_machine\_def.v | This is where #define constants are declared. |
| check\_time\_and\_coin.v | This submodule implements the rules descripted in the aforementioned section **rules for wait\_time** and act as the state register for wait\_time. |
| calculate\_current\_state.v | This submodule implements the combinational logic for current\_total\_nxt, and the output signals (o\_return\_coin, o\_output\_item, o\_available\_item) |
| change\_state.v | This submodule implements the state register for current\_total. |

The following snippets of code illustrate the output combinational logic.­

|  |
| --- |
|  |
| *Combinational Logic of o\_available\_item, o\_output\_item* |

We can imagine that the values of o\_available\_item depends both on the number of coins given to the system, and the types of items taken from the system. Therefore, the sensitivity list @(i\_input\_coin, i\_select\_item) ensures that the above block of logic gets trigger every time there is a change in either of the signals.

|  |
| --- |
|  |
| *Combinational Logic of o\_return\_coin* |

The above block of logic gets triggered at every wait\_time and updates the o\_return\_coin if 1) the wait\_time is below or equal to zero, or 2) i\_trigger\_return is HIGH. An important design choice to note here is the way in which our vending machine returns multiple coins. Suppose the vending machine had to return in total 2500KRW. It would do so over multiple consecutive wait\_time by first returning 1000KRW, then 1000KRW and 500KRW.

And the below snippet illustrates the calculation logic for current\_total\_nxt.

|  |
| --- |
|  |
| *Combinational logic that calculates current\_total\_nxt* |

An important thing to note here is the fact that there are no concurrent activations – that is, there will not no i\_input\_coin as well as i\_select\_item at the same time. Therefore, our design updates the current\_total\_nxt separately depending on 1) new coins are given to the system, 2) an item is selected, 3) the money is returned, 4) nothing happened.

1. **Discussion and Conclusion**

The implementation of vending machine was in hindsight simple. There was, however, a steep initial learning curve as there were a number of rules and logics to understand. Data flow modeling and behavior modeling are also quite different from the gate-level modeling that I am used to from Digital Systems and Design. Once we laid out all the ground works, though, the implementation was fairly simple.