**EE312 Lab Report – Lab 2. Vending Machine**

**20160030 고은석, 20160680 추헌호**

## 1. Introduction

본 과제에서는 Verilog HDL을 통해 간단한 가상의 자판기를 구현한다. 이 자판기에는 초기에 4종류의 물건이 각각 10개씩, 3종류의 동전이 각각 5개씩 적재되어 있다. 사용자는 자판기에 동전을 넣거나, 받을 물건을 선택하거나, 넣은 돈을 되돌려내게 하는 작업을 수행할 수 있다. 자판기는 조합회로와 순차회로로 구성되며, clk 신호가 0에서 1로 rise할 때마다 각 state를 업데이트한다. 본 과제를 통해 Lab. 1에서 경험했던 Verilog HDL에 더 익숙해지고, 또 clk 신호를 받는 기본적인 형태의 순차회로를 구현함으로써 FSM의 사용에 대한 이해도를 높일 수 있다.

## 2. Design

Vending machine의 구현에는 하나의 모듈이 사용된다. 단, Lab 1. ALU보다 조금 더 복잡한 형태로, 하나의 모듈 안에서 3개의 always block이 사용된다. 모듈의 입출력은 Figure 1과 같다. 이 때 output인 stopwatch와 current_total은 실제 자판기의 output이라기보다는, testbench에서 파형을 확인하기 위해 output으로 설정한 것이다. 사실 이 둘은 states로 설명하는 것이 더 적합하다.

각 state가 입력과 기존 state에 의해 어떻게 변하는지는 Table 1에 정리하였다. 단, Figure 1에 명시한 것처럼 일부 states는 update된 값이 clk 신호가 0에서 1로 올라가는 순간에 실제로 반영된다. 이 때, 기본 코드에서 주어진 state 중 사용하지 않은 것도 있고, 새롭게 추가한 것도 있다. 세부적인 설명은 3. Implemenation에 기술되어 있다.

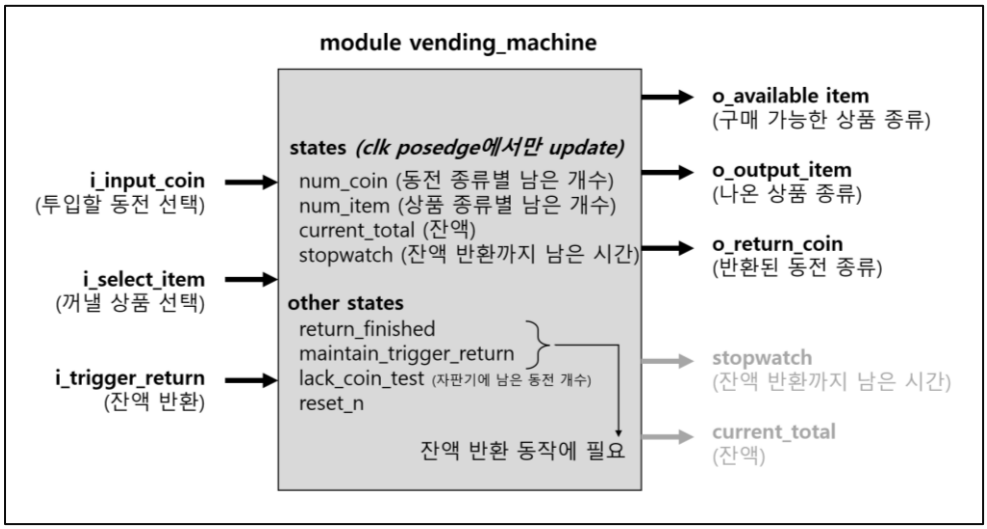


**Figure 1. I/O and states of vending_machine module**

| state | 초기 상태<br>(reset_n = 0) | 설명 |
|---|---|---|
| num_coin | ['d5, 'd5, 'd5] | i_input_coin[i]==1이면 num_coin[1]에 1을 더한다. |
| num_item | ['d10, 'd10, 'd10, 'd10] | i_select_item[i]==1이면 num_item[i]에서 1을 뺀다. |
| current_total | 0 | i_input_coin[i]==1이면 i_input_coin[i]*kkCoinValue[i]을 더한다. |
| stopwatch | `kWaitTime ('d10) | clk cycle 1번마다 1씩 빼고, 0이 되면 일단 그대로 유지하면서 잔액 반환을 실행한다. 사용자가 특정 행동을 하거나 반환이 끝나면 다시 초기값으로 돌아온다. |
| return_finished | 0 | 잔액 반환에 사용되는 state로, 반환이 끝난 순간에 1이 되며 reset_n==0이거나 동전을 다시 넣으면 0으로 돌아온다. |
| maintain_trigger_return | 0 | 잔액 반환에 사용되는 state로, i_trigger_input==1이면 1이 됐다가 반환이 끝나면 0으로 돌아온다. |
| lack_coin_test | x | o_available_item을 결정하기 위해 필요하다.<br>초기값은 어떤 상품을 구매했다고 "가정"했을 때 남은 잔액이다. |

**Table 1. Description of each states in vending_machine module**

## 3. Implementation

1) 동전 투입과 구매 가능한 상품 확인

동전 종류에는 100원, 500원, 1000원짜리가 있으며, 한 번에 하나의 동전을 투입할 수 있다. 투입된 동전의 종류에 따라 해당하는 num_coin element의 값이 올라간다. 또한 동시에 current_total에 투입한 동전 액수의 총합이 누적된다.

어떤 종류의 상품이 구매 가능한지는 두 단계를 통해 검증한다. 첫번째로 상품이 남아 있어야 하고, 또 투입한 금액의 잔액이 상품 가격 이상이어야 한다. ((current_total>=kkItemPrice[i]) && (num_items[i]>0)) 두번째로, 특정 종류의 상품을 구입한 후 거스름돈을 반환한다고 가정했을 때 자판기에 적재된 동전만으로 그것이 가능해야 한다. 이것을 테스트하는데 쓰이는 state가 lack_coin_test로, 자판기에 적재된 동전을 최대한으로 활용해서 거스름돈을 반환하는 가상의 테스트를 진행한 후 lack_coin_test가 0이 되면 성공이라고 판단하고 상품 판매가 가능하다고 표시한다.

2) 상품 선택

상품 종류에는 400원짜리, 500원짜리, 1000원짜리, 2000원짜리가 있으며 한 번에 하나의 상품을 골라 구매할 수 있다. 선택한 상품의 종류에 따라 해당하는 num_item element의 값이 차감된다. 또한 동시에 current_total에서 상품 가격만큼의 값이 빠진다.

3) 동전 반환 (Return)

동전이 반환되는 조건은 두 가지다. 첫째는 아무 동작도 시행하지 않은 채로 시간이

`kWaitTime만큼 흘렀을 때, 둘째는 i_trigger_return을 1로 설정했을 때이다. (후자는 실제 자판기에서라면 반환 레버를 돌렸을 때라고 볼 수 있다.) i_trigger_return이 1로 rise되면 이 값은 return이 끝날 때까지 maintain_trigger_return에서 유지된다. 즉 maintain_trigger_return은 반환 작업의 시작을 알린다. return이 끝나면 return_finished가 1로 설정된다. 즉 return_finished는 반환 작업의 끝을 알린다. 이 사이에서는 반복적으로 동전의 반환이 실행된다.

## 4. Evaluation

작성한 코드의 평가는 Operation Module이 작성되어 있는 vending_machine.v 파일을 포함하는 testbench 파일인 vending_machine_tb.v 파일을 ModelSim 프로그램을 통해서 시뮬레이션하는 방법으로 진행한다.

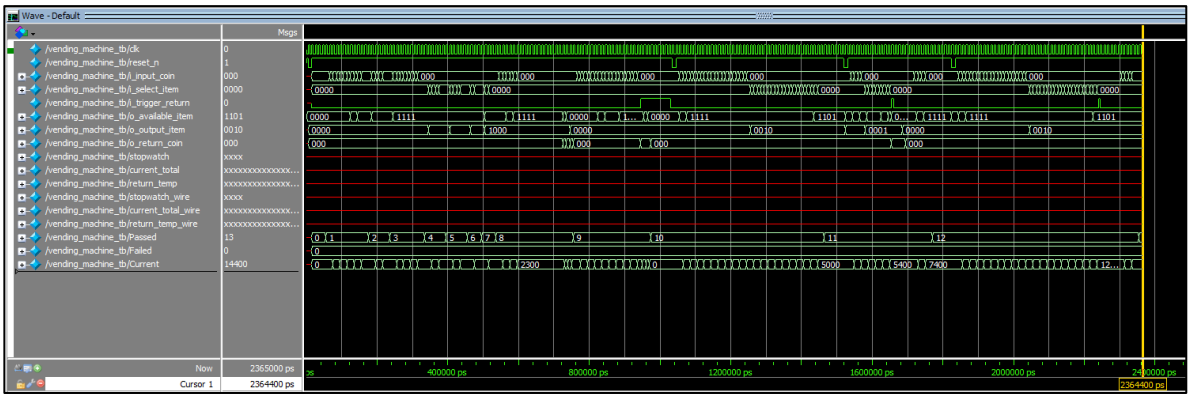


**Figure 2. Waveforms of outputs shown at Modelsim**

```
VSIM 3> run -all
# TEST                        InitialTest :
# PASSED
# TEST                   Insert100CoinTest :
# PASSED
# TEST                   Insert500CoinTest :
# PASSED
# TEST                  Insert1000CoinTest :
# PASSED
# TEST                   Select1stItemTest :
# PASSED
# TEST                   Select2ndItemTest :
# PASSED
# TEST                   Select3rdItemTest :
# PASSED
# TEST                   Select4thItemTest :
# PASSED
# TEST                      WaitReturnTest :
# PASSED
# TEST                   TriggerReturnTest :
# PASSED
# TEST                        ItemTestTest :
# PASSED
# TEST                        ItemTestTest :
# PASSED
# TEST                        ItemTestTest :
# PASSED
# Passed = 13, Failed = 0
```

**Figure 3. Results of tests shown at Modelsim**

Simulation 결과는 위와 같았고 ModelSim 프로그램의 transcript에서 볼 수 있듯이 총 13개의 testcase 중 Passed가 13개, Failed가 0개였다. 즉 성공적으로 모든 testcase를 통과했다. 이 때 마지막 3개의 테스트 이름이 같은데, 이것은 단순히 testbench 코드상의 오타로 여겨져 수정하지 않았다.

## 5. Discussion

본 과제의 개선사항으로 제안할 수 있는 것은 testbench 코드의 수정이다. 주어진 코드를 그대로 사용할 경우 Modelsim 시뮬레이션 결과에서 stopwatch_wire, current_total_wire의 파형을 확인할 수 없다. 값이 항상 x로 나타나기 때문이다. (temp_return_wire도 같은 경우이지만 우리가 사용하지 않았으므로 논외로 한다.)

일반적으로 testbench의 output은 wire이다. 그런데 상술한 3종의 경우에는 단순히 wire를 만들고 이를 vending_machine.v의 output reg들과 연결시키는 것이 아니고, 테스트벤치 내에서 자체적으로 같은 이름의 새로운 reg를 생성한 후 사용하는 방식을 채택한다. 그래서 실제로 Figure 4와 같이 일반적인 규칙에 따르도록 수정된 코드를 사용하면 모든 요소의 파형을 성공적으로 확인할 수 있다. (단, 본 과제의 4. Evaluation에서는 testbench를 임의로 수정하지 않고 주어진 코드를 그대로 사용하였기 때문에 Figure 2에서는 상술한 3종의 파형을 확인할 수 없다.)

given testbench

```
36  //Internal signals declarations:
37  reg clk;
38  reg reset_n;
39  reg [2:0]i_input_coin;
40  reg [3:0]i_select_item;
41  reg i_trigger_return;
42  wire [3:0]o_available_item;
43  wire [3:0]o_output_item;
44  wire [2:0]o_return_coin;
45
46  reg [3:0]stopwatch;
47  reg [`kTotalBits-1:0] current_total;
48  reg [`kTotalBits-1:0] return_temp;
49
50  wire [3:0]stopwatch_wire = stopwatch;
51  wire [`kTotalBits-1:0] current_total_wire = current_total;
52  wire [`kTotalBits-1:0] return_temp_wire = return_temp;
```

modified testbench

```
36  //Internal signals declarations:
37  reg clk;
38  reg reset_n;
39  reg [2:0]i_input_coin;
40  reg [3:0]i_select_item;
41  reg i_trigger_return;
42  wire [3:0]o_available_item;
43  wire [3:0]o_output_item;
44  wire [2:0]o_return_coin;
45  /*
46  reg [3:0]stopwatch;
47  reg [`kTotalBits-1:0] current_total;
48  reg [`kTotalBits-1:0] return_temp;
49  */
50  wire [3:0]stopwatch_wire; // = stopwatch;
51  wire [`kTotalBits-1:0] current_total_wire; // = current_total;
52  wire [`kTotalBits-1:0] return_temp_wire; // = return_temp;
```

**Figure 4. Suggestion for modification of testbench code**

## 6. Conclusion

작성한 vending_machine.v를 이용하여 확인해본 결과 실패한 testcase가 없었기에 vending_machine.v가 적절히 작성되었다고 판단했다. 본 과제를 통해 Verilog HDL에 더 익숙해졌고, 조합회로 뿐 아니라 순차회로를 어떻게 구현할 수 있는지에 대해 더 깊게 이해할 수 있었다.