컴퓨터 구조 (CSED311)

<Lab1 - ALU and Vending Machine>

Team ID : 33
Student 1 : 곽민성 (Gwak Minseong, 20230840)
Student 2 : 김재환 (Kim Jaehwan, 20230499)

1. Introduction

이번 랩의 목표는 베릴로그를 사용하여 ALU와 Vending machine를 구현하는 것을 목표로 한다. 올바른 구현을 위해서 기초 베릴로그 문법 및 Blocking 할당과 Non-Blocking 할당, 조합 논리와 순차 논리 등에 대해서 정확히 이해하여야 한다. 이 랩을 통해서 기본적인 하드웨어 설계 및 언어에 대해서 배울 수 있다.

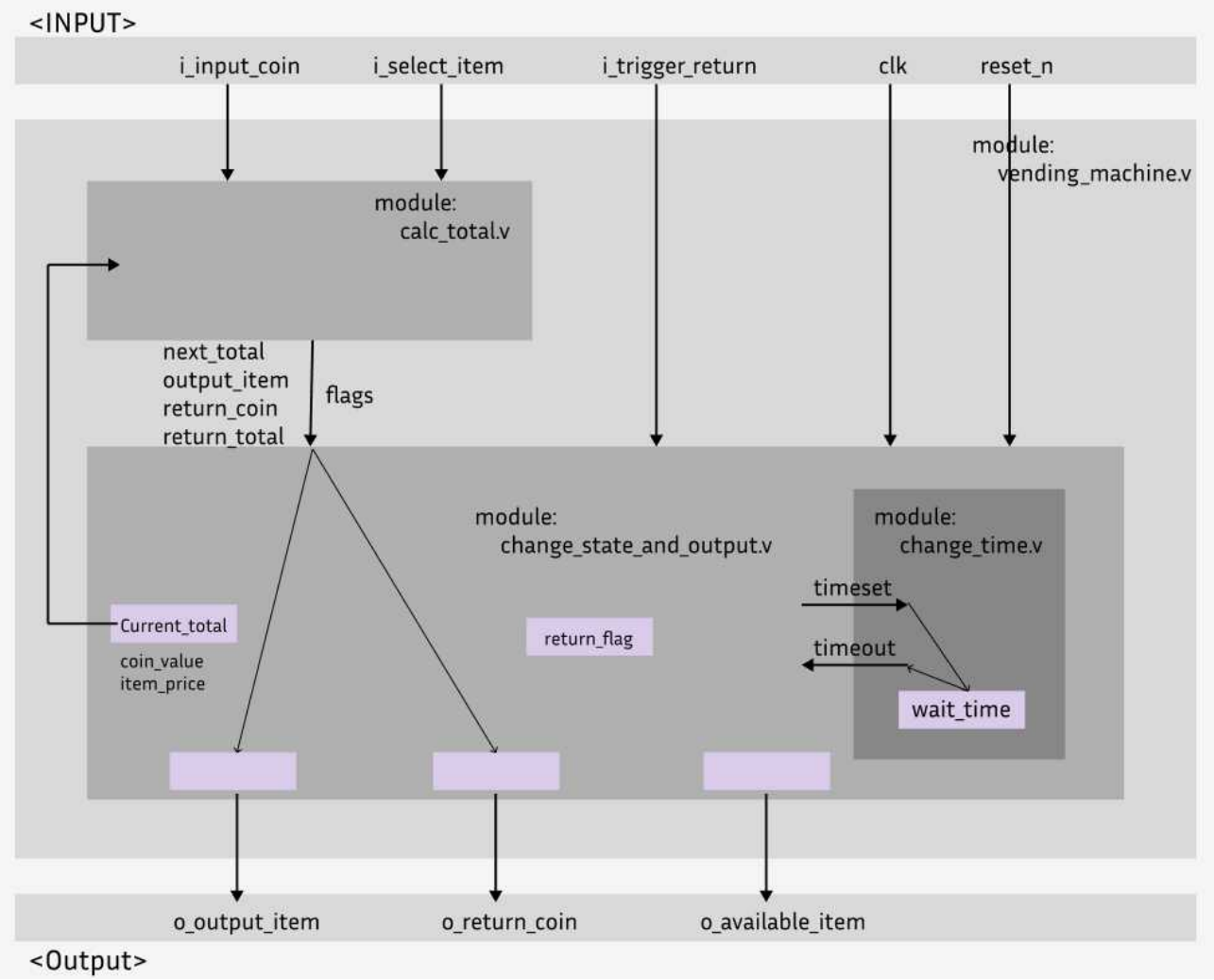
2. ALU - Implementation

ALU는 단순한 순차 논리만으로 연산 결과와 OverflowFlag를 계산하여 출력한다. 그 구현은 다음과 같다.

```
10      initial begin
11          C = 0;
12          OverflowFlag = 0;
13      end
14
15      always @(*) begin
16          OverflowFlag = 0;
17          case(FuncCode)
18          `FUNC_ADD : begin
19              C = A + B;
20              OverflowFlag = (~(A[data_width-1] ^ B[data_width-1])) & (A[data_width-1] ^ C[data_width-1]);
21          end
22          `FUNC_SUB : begin
23              C = A - B;
24              OverflowFlag = (A[data_width-1] ^ B[data_width-1]) & (A[data_width-1] ^ C[data_width-1]);
25          end
26          `FUNC_ID : C = A;
27          `FUNC_NOT : C = ~A;
28          `FUNC_AND : C = A & B;
29          `FUNC_OR : C = A | B;
30          `FUNC_NAND : C = ~(A & B);
31          `FUNC_NOR : C = ~(A | B);
32          `FUNC_XOR : C = A ^ B;
33          `FUNC_XNOR : C = ~(A ^ B);
34          `FUNC_LLS : C = A << 1;
35          `FUNC_LRS : C = A >> 1;
36          `FUNC_ALS : C = A <<< 1;
37          `FUNC_ARS : C = A >>> 1;
38          `FUNC_TCP : C = ~A + 1;
39          `FUNC_ZERO : C = 0;
40          endcase
41      end
```

3. Vending machine - Design

다음과 같은 자판기 모듈을 구성하였다.



- **vending_machine.v**

  Top level module로서, 주어진 입력을 적절히 처리하여 알맞은 출력을 제공한다.

- **calc_total.v**

  돈 및 물건 계산 처리를 담당한다. Vending_machine 모듈의 하위 모듈이며, 입력값과 change state and output 모듈로부터 여러 가지 정보를 받아 조합론적으로 여러 가지 정보를 계산한다.

- **change_state_and_output.v**

  상태 전환 및 출력을 담당한다. 입력값, calc_total 모듈로부터 계산된 정보, change_time 모듈 정보를 받아 알맞은 출력을 만드는 모듈이다.

- **change_time.v**

  시간 초과 메커니즘을 담당한다. change_state_and_output 모듈의 하위 모듈이며, timeset 신호를 받기 시작한 이후로 100 time을 세는 모듈이다. 시간이 0이 되면 timeout 신호를 출력한다.

4. Vending machine - Implementation

- **vending machine.v**

Vending machine은 탑 레벨 모듈이며, 입력 및 출력은 기존 스켈레톤 코드 그대로 사용하였다.

```
49      // Internal states. You may add your own net variables.
50      wire [`kTotalBits-1:0] current_total;
51
52      // Next internal states. You may add your own net variables.
53      wire [`kTotalBits-1:0] next_total;
54
55      wire [`kNumItems-1:0] output_item;
56      wire [`kNumCoins-1:0] return_coin;
57
58      // Variables. You may add more your own net variables.
59      wire [`kTotalBits-1:0] return_total;
60
61      wire flag_inserted;
62      wire flag_output_item;
```

주어진 코드 이외에 다음과 같은 변수를 정의하였다. current total 및 next total 변수를 통해 계산 모듈과 상태 관리 모듈 간 잔액 정보를 전달하며, 계산 모듈로부터 계산된 정보를 전달하는 와이어 변수를 선언하였다.

```
65      calc_total calc_total(
66                          .current_total(current_total),
67                          .i_input_coin(i_input_coin),
68                          .i_select_item(i_select_item),
69                          .coin_value(coin_value),
70                          .item_price(item_price),
71                          .next_total(next_total),
72                          .output_item(output_item),
73                          .return_coin(return_coin),
74                          .return_total(return_total),
75                          .flag_inserted(flag_inserted),
76                          .flag_output_item(flag_output_item));
```

```
78      change_state_and_output change_state_and_output(
79                          .clk(clk),
80                          .reset_n(reset_n),
81                          .next_total(next_total),
82                          .output_item(output_item),
83                          .return_coin(return_coin),
84                          .return_total(return_total),
85                          .flag_inserted(flag_inserted),
86                          .flag_output_item(flag_output_item),
87                          .i_trigger_return(i_trigger_return),
88                          .item_price(item_price),
89
90                          .o_output_item(o_output_item),
91                          .o_return_coin(o_return_coin),
92                          .o_available_item(o_available_item),
93                          .current_total(current_total));
```

위와 같이 두 모듈을 연결하였다.

- **calc_total.v**

calc_total은 여러 정보를 입력받아 필요한 계산을 수행하는 모듈이다.

```
6       input [`kTotalBits-1:0] current_total;
7       input [`kNumItems-1:0] i_select_item;
8       input [`kNumCoins-1:0] i_input_coin;
9       input [31:0] coin_value [`kNumCoins-1:0];
10      input [31:0] item_price [`kNumItems-1:0];
11
12      output reg [`kTotalBits-1:0] next_total;
13
14      wire [`kTotalBits-1:0] required_money;
15      assign required_money = (i_select_item[0] * item_price[0]
16                             + i_select_item[1] * item_price[1]
17                             + i_select_item[2] * item_price[2]
18                             + i_select_item[3] * item_price[3]);
19
20      output reg [`kNumItems-1:0] output_item;
21      output reg [`kNumCoins-1:0] return_coin;
22      output flag_inserted = (i_input_coin != `kNumCoins'b0);
23      output flag_output_item = ((i_select_item != `kNumItems'b0) && (current_total >= required_money));
24      output reg [`kTotalBits-1:0] return_total;
25      integer i;
```

위와 같이 입력, 출력을 구성하였다. 또한 내부적인 변수로 required_money 와이어 변수 및 반복문 순회 용도로 integer를 선언하였다.

```
27      always @(*) begin
28          output_item = `kNumItems'b0;
29          next_total = current_total
30                  + coin_value[0] * i_input_coin[0]
31                  + coin_value[1] * i_input_coin[1]
32                  + coin_value[2] * i_input_coin[2];
33
34          if(next_total >= required_money) begin
35              output_item = i_select_item;
36              next_total = next_total - required_money;
37          end
38
39          return_total = next_total;
40          return_coin = `kNumCoins'b0;
41
42          for(i = `kNumCoins-1; i >= 0; i--) begin
43              if(return_total >= coin_value[i]) begin
44                  return_coin = return_coin | (`kNumCoins'b001 << i);
45                  return_total =  return_total - coin_value[i];
46              end
47          end
48      end
```

다음 조합 논리를 통해 계산한다. 계산되는 값은 다음과 같다.

- next_total : 입력 및 선택한 물건이 자판기에서 나온 후 남은 잔액
- output_item : 자판기에서 나올 아이템
- return_coin : 자판기에서 반환될 동전
- return_total : 자판기에서 동전이 반환된 후 아직 반환되지 않고 남은 값
- flag_inserted, flag_output_item : 동전 삽입, 물건이 나왔는지 확인하는 플래그

- **change_state_and_output.v**

```
6       input clk;
7       input reset_n;
8       input [`kTotalBits-1:0] next_total;
9       input [`kNumItems-1:0] output_item;
10      input [`kNumCoins-1:0] return_coin;
11      input [`kTotalBits-1:0] return_total;
12      input flag_inserted;
13      input flag_output_item;
14      input i_trigger_return;
15      input [31:0] item_price [`kNumItems-1:0];
16
17      output reg [`kNumCoins-1:0] o_return_coin;
18      output reg [`kNumItems-1:0] o_output_item;
19      output reg [`kNumItems-1:0] o_available_item;
20      output reg [`kTotalBits-1:0] current_total;
21
22      integer i;
23
24      wire timeout;
25      wire timeset;
26      assign timeset = flag_inserted || flag_output_item;
27
28      reg flag_return;
29
30      initial begin
31          flag_return = 0;
32      end
```

위와 같이 입력, 출력을 정의하였다. 또한 반복문 순회를 위한 integer 변수와 내부 모듈 change_time에 연결하기 위한 timeout 및 timeset 와이어를 정의하였다. 또한 동전 반환 시 한 번에 반환되지 않는 문제를 해결하기 위한 플래그를 하나 추가하였다.

아래 코드는 현재 자판기에 들어간 총 잔액 및 동전 반환에 관한 순차 논리이다. 아래 코드에 대한 자세한 설명은 Discussion에서 논의한다.

```
34      always @(posedge clk) begin
35          if (!reset_n) begin
36              current_total <= 0;
37              o_output_item <= 0;
38              o_return_coin <= 0;
39              flag_return <= 0;
40          end
41          else begin
42              current_total <= next_total;
43              o_output_item <= output_item;
44
45              if(timeout) begin
46                  o_return_coin <= return_coin;
47                  current_total <= return_total;
48              end
49              else begin
50                  if(i_trigger_return) begin
51                      o_return_coin <= return_coin;
52                      flag_return <= 1;
53                  end
54                  else begin
55                      o_return_coin <= 0;
56                      if(flag_return) begin
57                          current_total <= return_total;
58                          flag_return <= 0;
59                      end
60                  end
61              end
62          end
63      end
```

아래 코드는 조합 논리를 통해 o_available_item을 계산하는 코드이다.

```
65      always @(*) begin
66          o_available_item = 4'b0000;
67          for(i = 0; i < 4; i++) begin
68              if(current_total >= item_price[i])
69                  o_available_item = o_available_item | (4'b0001 << i);
70          end
71      end
```

아래 코드는 change_state_and_output의 내부 모듈인 change_time과 연결하는 코드이다.

```
73      change_time change_time(
74                              .clk(clk),
75                              .reset_n(reset_n),
76                              .timeset(timeset),
77                              .timeout(timeout));
```

- **change_time.v**

100 단위의 시간을 계산하여, 시간 내로 입력이 더 주어지지 않을 경우 동전을 반환하도록 하는 모듈이다. 순차 논리로 동작하며, timeset 신호를 받으면 `kWaitTime만큼 켜지며 클럭당 1씩 감소하다가, 0이 되면 timeout신호를 출력한다.

```
3   module change_time(clk,reset_n,timeset,timeout);
4       input clk;
5       input reset_n;
6       input timeset;
7       output timeout;
8       reg [`kTotalBits : 0] wait_time;
9       assign timeout = (wait_time == 0);
10
11      always @(posedge clk) begin
12          if (!reset_n) wait_time <= -1;
13          else if (timeset) wait_time <= `kWaitTime;
14          else if (wait_time > 0) wait_time <= wait_time - 1;
15          else wait_time <= 0;
16      end
17  endmodule
```

5. Discussion

과제를 수행하면서 다음과 같은 점에 대하여 고민하였다.

- **각 모듈을 어떻게 설계하였는가?**

calc_total 모듈에서 입력을 조합 논리를 통해 처리하고, change_state_and_output 모듈에서 calc_total의 결과를 적용하여 출력하는 방식으로 설계하였다. 또한 wait_time을 처리하기 위한 타이머 모듈은 별도의 기능으로 분리할 수 있으므로 모듈을 분리하였다.

- **동시에 여러 물건 구매시.**

한 번에 여러 물건을 선택했을 때, 잔액이 물건 하나 구매하기에는 충분하지만 둘 모두 구매할 수 없는 경우 어떻게 대응해야할지 명확히 주어지지 않았다. 이 경우 물건을 모두 구매할 수 있는 경우만 모두 구매하고, 아니라면 아무 물건도 출력하지 않도록 구현하였다. 예를 들어 2000원이 있을 때 1000원 물건과 2000원 짜리 물건을 동시에 누른다고 하더라도, 아무것도 나오지 않는다.

- **동전반환 로직**

동전을 반환할 때 설계 논리상 100원, 500원 1000원 값을 각각 한 개 씩만 반환할 수 있다는 문제가 있었다. 이를 처리하기 위해서 새로운 작동 방식을 고심해 만들어 적용하였다.

이하 방식은 테스트벤치를 만족하면서, 논리적으로도 올바른 합리적인 작동방식이다.

- i_trigger_return 신호를 받고 나서, 신호가 0이 될 때 100원, 500원, 10000원 동전을 최대 한 개씩 반환한다. 따라서 모든 동전을 반환하기 위해서 i_trigger_return 신호를 여러번 받아야 한다.
- 100 클럭 후 timeout 발생시에는 모든 동전을 연속해서 반환한다.

6. Conclusion

모든 기능을 성공적으로 구현하여, 아래와 같이 주어진 테스트 케이스를 통과하는 모습을 확인할 수 있다.

ALU	Vending Machine
### SIMULATING ### sim_time : 0 A = 1, B = 1, FuncCode = 0, Result = 0 sim_time : 1 A = 1, B = 1, FuncCode = 0, Result = 2 Test 1 Passed sim_time : 2 A = 1, B = 1, FuncCode = 0, Result = 2 sim_time : 3 A = 1, B = 1, FuncCode = 0, Result = 2 Test 2 Passed sim_time : 4 A = 1, B = 1, FuncCode = 1, Result = 2 sim_time : 5 A = 1, B = 1, FuncCode = 1, Result = 0 Test 3 Passed sim_time : 6 A = 1, B = 1, FuncCode = 2, Result = 0 sim_time : 7 A = 1, B = 1, FuncCode = 2, Result = 1 Test 4 Passed sim_time : 8 A = 1, B = 1, FuncCode = 3, Result = 1 sim_time : 9 A = 1, B = 1, FuncCode = 3, Result = 65534 Test 5 Passed sim_time : 10 A = 11, B = 5, FuncCode = 4, Result = 65534 sim_time : 11 A = 11, B = 5, FuncCode = 4, Result = 1 Test 6 Passed sim_time : 12 A = 11, B = 5, FuncCode = 5, Result = 1 sim_time : 13 A = 11, B = 5, FuncCode = 5, Result = 15 Test 7 Passed sim_time : 14 A = 11, B = 5, FuncCode = 6, Result = 15 sim_time : 15 A = 11, B = 5, FuncCode = 6, Result = 65534 Test 8 Passed sim_time : 16 A = 11, B = 5, FuncCode = 7, Result = 65534 sim_time : 17 A = 11, B = 5, FuncCode = 7, Result = 65520 Test 9 Passed sim_time : 18 A = 11, B = 5, FuncCode = 8, Result = 65520 sim_time : 19 A = 11, B = 5, FuncCode = 8, Result = 14 Test 10 Passed sim_time : 20 A = 11, B = 5, FuncCode = 9, Result = 14 sim_time : 21 A = 11, B = 5, FuncCode = 9, Result = 65521 Test 11 Passed sim_time : 22 A = 11, B = 5, FuncCode = 10, Result = 65521 sim_time : 23 A = 11, B = 5, FuncCode = 10, Result = 22 Test 12 Passed sim_time : 24 A = 11, B = 5, FuncCode = 11, Result = 22 sim_time : 25 A = 11, B = 5, FuncCode = 11, Result = 5 Test 13 Passed sim_time : 26 A = 11, B = 5, FuncCode = 12, Result = 5 sim_time : 27 A = 11, B = 5, FuncCode = 12, Result = 22 Test 14 Passed sim_time : 28 A = 11, B = 5, FuncCode = 13, Result = 22 sim_time : 29 A = 11, B = 5, FuncCode = 13, Result = 5 Test 15 Passed sim_time : 30 A = 11, B = 5, FuncCode = 14, Result = 5 sim_time : 31 A = 11, B = 5, FuncCode = 14, Result = 65525 Test 16 Passed sim_time : 32 A = 11, B = 5, FuncCode = 15, Result = 65525 sim_time : 33 A = 11, B = 5, FuncCode = 15, Result = 0 Test 17 Passed sim_time : 34 A = 32767, B = 1, FuncCode = 0, Result = 0 sim_time : 35 A = 32767, B = 1, FuncCode = 0, Result = 32768 Test 18 Passed sim_time : 36 A = 32768, B = 1, FuncCode = 1, Result = 32768 sim_time : 37 A = 32768, B = 1, FuncCode = 1, Result = 32767 Test 19 Passed All tests passed	### SIMULATING ### initial test sim_time: 8 PASSED : o_available_item: 0, expected 0 Available item test sim_time: 10 PASSED : available item: 1, expected 0b0001 PASSED : available item: 3, expected 0b0011 PASSED : available item: 7, expected 0b0111 PASSED : available item: 15, expected 0b1111 Return_coin test sim_time: 52 PASSED : return coin: 1, expected 0b001 PASSED : return coin: 2, expected 0b010 PASSED : return coin: 4, expected 0b100 PASSED : return coin: 3, expected 0b011 PASSED : return coin: 5, expected 0b101 PASSED : return coin: 6, expected 0b110 PASSED : return coin: 7, expected 0b111 Select item test sim_time: 122 PASSED : output item: 0b110, expected 0b0110 success: 13 / 13

7. References

[1] Lecture notes.