2024학년도 컴퓨터구조 Lab Assignment #1
ALU, Vending Machine 설계

김도영, 선민수

2024년 3월 9일

1 개관

이 과제에서는 하드웨어 기술 언어인 Verilog를 이용하여 ALU (Arithmetic Logic Unit)를 설계해 보고, 더 나아가 자판기의 동작을 제어하는 동기회로를 설계해 보는 것을 목적으로 한다.

1.1 ALU

ALU 모듈의 기능을 간략히 설명하면 다음과 같다.

- 입력 포트를 통해 두 입력값과 `FuncCode`를 입력받는다.
- 입력받은 `FuncCode`에 따라 해당하는 동작을 수행 후, 이를 출력 포트, `C`와 `OverflowFlag`로 출력한다.
  - 이때, `FuncCode`가 덧셈, 혹은 뺄셈에 해당하는 `FuncCode`라면 오버플로우 발생 시 그 여부를 `OverflowFlag`로 출력한다.

1.2 자판기

자판기(`vending_machine`) 모듈의 기능을 간략히 설명하면 다음과 같다.

- 입력 포트를 통해 투입된 동전을 나타내는 `i_input_coin`, 선택된 상품을 나타내는 `i_select_item`, 잔돈 반환 버튼을 나타내는 `i_trigger_return`을 입력받는다.
- 현재까지 투입된 동전의 총 액수에 따라, 구입 가능한 상품을 나타내는 `o_available_item`을 설정한다
  - 만약 동전이 투입되었다면, 대기 시간을 재설정한다.
  - 만약 동전이 투입되지 않았다면 대기 시간을 1 줄인다.
- 사용자가 선택한 상품, `i_select_item`에 따라 상품을 반환하거나 하지 않는다.
  - 만약 `i_select_item`으로 입력된 상품이 구매 가능하다면, `o_output_item`을 통해 해당 상품을 반환한다.
  - 만약 `i_select_item`으로 입력된 상품이 구매 불가능하다면, 상품을 반환하지 않는다.
- 만약 대기 시간이 0이 되거나 잔돈 반환 버튼이 눌렸다면, 잔돈을 반환한다.

2 설계

ALU, 자판기 모듈의 설계는 다음과 같다.

2.1 ALU

먼저, 초기 상태를 지정하는 Verilog의 `initial` 블록에서는 출력값인 `C`와 `OverflowFlag`를 `0`로 설정해 주었다. 이후, ALU의 실제 동작을 기술하는 `always` 블록에서는 `switch`문을 이용하여 `FuncCode`에 따라 해당하는 동작을 하도록 설계하였다. 이때, `always`문은 클럭 등의 신호의 edge에 반응하는 것이 아닌, 입력 `A`, `B`, `FuncCode`에 따라 변화하므로 ALU는 순차 논리 회로가 아닌 조합 논리 회로로 구성되었다.

```
...
`FUNC_ADD : begin
    C = A + B;
    OverflowFlag = {A[data_width - 1], A} + {B[data_width - 1], B} !== {C[data_width - 1], C};
end
`FUNC_SUB : begin
    C = A - B;
    OverflowFlag = {A[data_width - 1], A} - {B[data_width - 1], B} !== {C[data_width - 1], C};
end
...
```

정수 덧셈의 구현

이때, `FuncCode` `0000`과 `0001`에 해당하는 부호 있는 정수형의 덧셈과 뺄셈 연산에서는 정수 자료형의 최대 표현 범위로 인해 발생하는 오버플로우 문제가 발생할 수 있다. 따라서, 이러한 연산을 수행할때는, 입력값을 1bit씩, `A`와 `B`의 MSB와 같은 값으로 늘린 값들을 연산한 값과 실제 연산한 값이 같은지를 확인하여 만약 이 두 값이 같지 않다면 `OverFlowFlag`를 1로 설정하여 오버플로우 감지 문제를 해결하였다.

```
...
`FUNC_ARS : begin
    C = {A[data_width - 1], {A >>> 1}[data_width - 2: 0]};
    OverflowFlag = 0;
end
...
```

산술 우측 시프트 연산의 구현

또한, Verilog에서는 부호 여부를 따로 지정해주지 않으면 입력인 `A`와 `B`가 기본적으로 부호가 없는 것으로 간주된다. 때문에, 산술 우측 시프트 연산을 수행할 때 일반적으로 예상하는 것 처럼 `A`의 MSB가 연장되는 것이 아닌, `0`으로 연장되는 문제가 있다. 이 문제를 해결하기 위해, 산술 우측 시프트 연산을 수행하는 `FuncCode` `1101`에 해당하는 부분을 구현하며 MSB는 `A`의 MSB로 그대로 채우고, 그 하위 비트는 `A >>> 1`의 결과값으로 채우도록 구현하였다.

2.2 자판기

자판기 기능을 하는 모듈을 설계하기 위해, vending_machine 모듈을 세 개의 모듈, 즉 vm_merchant, vm_state, vm_timer 모듈로 나누어 설계하였다.

vm_merchant 모듈은 자판기의 핵심적인 기능을 담당하는 조합 논리회로로, 사용자가 현재 투입한 동전의 개수를 나타내는 i_input_coin, 사용자가 선택한 상품을 나타내는 i_select_item, 현재 자판기 안에 남아있는 잔액을 나타내는 current_total, 그리고 현재 남은 시간을 나타내는 wait_time을 입력으로 받는다. 이러한 입력을 통해, vm_merchant 모듈은 다음 상태에서 판매 가능한 상품을 나타내는 o_available_item, 다음 상태에서 반환해야 할 상품을 나타내는 o_output_item, 다음 상태에서 자판기 안에 남아있는 잔액을 나타내는 current_total_nxt, 그리고 다음 상태에서 반환해야 할 잔돈을 나타내는 o_return_coin을 출력으로 내보낸다.

vm_state 모듈은 클럭에 맞춰 현재 자판기의 잔액을 업데이트하는 순차 논리 회로로, 클럭, 리셋 신호와 다음 잔액을 나타내는 current_total_nxt를 받아 출력인 current_total을 새로운 잔액으로 업데이트한다.

vm_timer 모듈은 클럭에 맞춰 대기 시간을 업데이트하는 순차 논리 회로로, 클럭, 리셋 신호와 i_trigger_return, i_input_coin, i_select_item, o_available_item을 받아 다음 대기 시간인 wait_time을 업데이트한다.

3 구현

ALU의 구현에 있어서는 **설계**에서 충분히 설명하였으므로 생략하고, **구현**에서는 자판기 회로의 구현, 그 중에서도 자판기 회로의 가장 핵심적인 역할을 담당하는 vm_merchant 모듈과 vm_timer 모듈에 집중하여 설명한다.

3.1 vm_merchant 모듈

vm_merchant 모듈은 현재 입력에 따라 출력을 내보내는 조합 논리 회로로 구성되어 있다. 먼저, i_input_total의 변화에 따라, 사용자가 투입한 동전 값어치의 총량을 계산한다. 이 계산은 i_input_coin의 각 비트를 검사하여, 만약 해당 비트의 값이 1일 경우 해당하는 동전의 값어치를 input_total에 더하여 이루어진다.

```
// First set 'coins to return' and 'total values of returned coin' to 0.
o_return_coin = 0;
o_return_total = 0;
...
// Iterate o_return_coin from MSB to LSB.
for(i = `kNumCoins - 1; i >= 0 && current_total - return_total != 0 ; i = i - 1) begin
    // If the remaining balance is same or greater than the value of the coin of current bit,
    if(current_total - return_total >= coin_value[i]) begin
        // ...set the bit to 1 and add the value into total values of returned coin.
        o_return_coin[i] = 1;
        return_total = return_total + coin_value[i];
    end
end
...
```

o_return_coin 계산 논리의 구현

이 모듈에서는 또한 현재 자판기에 남아있는 잔액인 current_total의 변화에 따라 반환해야 할 잔돈의 양 또한 계산한다. 이는 o_return_coin의 각 비트에 해당하는 동전의 값어치를 MSB부터 순회하여, 남아있는 잔액의 양이 해당 동전의 값어치보다 클 경우 o_return_coin의 해당 비트를 1로 설정함으로써 이루어진다.

current_total, i_select_item에 따른 o_available_item, o_output_item의 계산도 해당 모듈에서 이루어진다. 먼저, o_available_item은 각 비트를 순회하며 해당 비트의 상품 값어치가 현재 잔액보다 클 경우, o_available_item[i]를 1로 설정하여 계산된다. 이후, 만약 해당 비트의 상품을 사용자가 선택하였다면, 다시 말해 i_select_item[i]가 1이라면, o_output_item의 해당 비트를 1로 설정하여 해당 상품을 반환함을 표시한다. 또한, 반환된 상품 값어치의 총 액수를 표시하는 output_total에 해당 상품의 액수를 더한다.

마지막으로, 다음 상태에서 자판기에 남아있는 잔액을 나타내는 current_total_nxt를 현재 잔액에 투입한 값을 더하고, 이 값에서 반환된 상품 값어치의 총합과 반환된 잔돈의 총액을 뺀 값으로 수정한다.

3.2 vm_timer 모듈

vm_timer 모듈에서는 사용자의 입력인 i_input_coin과 i_select_item, 그리고 현재 구입 가능한 상품인 o_available_item에 따라 대기 시간을 업데이트한다. 먼저, 리셋 신호에 따라 대기 시간을 0으로 업데이트한다.

이후에는 사용자의 입력에 따라 wait_time이 업데이트되는 값이 달라진다. 먼저, 만약 사용자가 동전을 투입했거나 (즉, i_input_coin이 0이 아니거나) 사용자가 선택한 상품이 구입 가능한 경우 초기값으로 대기 시간을 업데이트한다. 이후 사용자가 동전을 투입하지 않았거나 선택한 상품이 구입 불가능한 경우 초기값에서 한 클럭 사이클마다 대기 시간을 1씩 줄여 나간다.

```
...
// When the user pressed return trigger button, the wait_time is not immediately set to 0.
else if(!trigger_flag && i_trigger_return) begin
    // The module rather waits until 3 clock cycles pass.
    // The trigger_flag stores whether the vending machine is triggered return and waiting for 3 cycles.
    trigger_flag <= 1;
    wait_time <= 3;
end
...
else if(wait_time != 0) // Decreases time of wating by 1.
    wait_time <= wait_time - 1;
else if(trigger_flag && wait_time == 0) // If the wait of 3 cycles ended, reset the flag into 0.
    trigger_flag <= 0;
...
```

3 사이클 대기 논리의 구현

이때 만약 사용자가 잔돈 반환 버튼을 눌렀을 경우 i_input_coin, i_select_item에 상관없이 wait_time을 3으로 설정하고 trigger_flag를 1로 설정한다. trigger_flag는 i_trigger_return의 입력 이후 3 클럭 사이클을 기다린 후 잔돈 반환이 이루어져야 하는 테스트벤치의 조건을 만족하기 위해, '현재 잔돈 반환 버튼이 눌린 후 대기중인지'를 저장하기 위한 플래그 변수이다. 이후 3 클럭 사이클의 대기 시간이 끝나면 trigger_flag는 다시 0로 설정된다.

4 논의

4.1 자판기 회로는 Moore Machine인가 Mealy Machine인가?

유한 상태 기계의 종류에는 Moore Machine과 Mealy Machine이 있다. 두 종류 모두 조합 논리 회로와 순차 논리 회로가 결합되어, 입력에 따라 상태가 변화하고 출력을 내는, 유한한 상태를 가진 기계라는 점에서는 동일하지만 Moore Machine은 출력이 오로지 현재 상태에 의해서만 결정되고, Mealy Machine은 현재 상태 뿐만 아니라 입력에 의해서도 출력이 변화할 수 있는 기계라는 점에서 다르다. 즉, 만약 Moore Machine을 상태 다이어그램으로 그린다면 각 상태에 해당하는 노드 하나마다 출력이 하나로 정해져 있겠지만, Mealy Machine의 경우 노드와 노드를 잇는 간선마다 입력과 그에 따른 출력이 명시되어야 할 것이다.

그렇다면 과제에서 구현한 자판기 회로는 Moore Machine인가 Mealy Machine인가? 구현된 자판기 회로를 살펴보면, `o_available_item`과 `o_output_item`, `o_return_coin` 모두 내부 상태라고 할 수 있는 현재 잔액에 의해서만 출력이 변화하는 것이 아닌, 입력에 따라 출력이 변화하는 구조를 가지고 있다. 예를 들어, `o_output_item`의 경우 현재 잔액뿐만 아니라 사용자의 상품 선택 입력, 즉 `i_select_item`에 따라서도 출력이 변화하는 것을 볼 수 있다.

또한, 구현된 자판기 회로는 Mealy Machine의 중요한 특징인 입력이 조합 회로를 거쳐 출력으로 나타날 수 있기 때문에, 비동기적으로 출력이 변화할 수 있다는 특징 또한 가지고 있다. 따라서, 과제에서 구현된 자판기 회로는 Mealy Machine이며, 출력이 내부 상태에 의해서만 나타나지 않고 외부 입력에 의해서도 변화할 수 있다.

4.2 자판기 회로에서 조합 논리 회로와 순차 논리 회로의 구현

그렇다면, Mealy Machine인 자판기 회로를 어떻게 조합 논리 회로와 순차 논리 회로로 분리하여 구현할 수 있을까? Mealy Machine은 입력이 조합 논리 회로를 거쳐 순차 논리 회로에 상태가 저장되고, 저장된 상태와 입력을 통해 출력을 계산하는 구조를 가지고 있다. 즉, Mealy Machine을 구현하기 위해서는 현재 상태를 저장하는 부분을 순차 논리 회로로 구현하고, 입력을 통해 다음 상태를 계산하는 부분과 현재 상태와 입력을 이용해 출력을 계산하는 부분을 조합 논리 회로로 구현하여 이 둘을 결합하여야 한다.

구현된 자판기 회로에서 현재 상태라고 할 수 있는 부분은 대기 시간을 나타내는 `wait_time`과 현재 자판기에 남아있는 잔액을 나타내는 `current_total` 부분이다. 따라서, 구현된 자판기 회로에서는 이 둘을 업데이트하기 위해 클럭에 동기화되는 순차 논리 회로인 `always @(posedge clk) begin ... end` 블럭을 이용하여 구현하였다.

또한, 입력과 현재 상태를 기반으로 출력과 다음 상태를 계산하는 부분은 Mealy Machine에서 조합 논리로 구현된다. 따라서, 자판기 회로에서 입력에 해당하는 `i_select_item`과 `i_input_coin`과 현재 상태에 해당하는 `wait_time`과 `current_total`을 기반으로 출력과 다음 상태를 계산하는 부분은 `vm_merchant` 모듈의 `always @(*) begin ... end` 블럭을 이용하여, 입력이 변화할 때마다 비동기적으로 출력이 변화하는 조합 논리 회로로 구현하였다.

5 결론

이번 과제에서는 ALU와 자판기 회로를 레지스터 신호전달 수준(Register Transfer Level; RTL)에서 하드웨어 기술 언어인 Verilog를 이용하여 구현해 보았다. 먼저 `FuncCode`에 따라 두 입력값에 연산을 적용한 결과를 출력하는 간단한 조합 논리 회로인 ALU를 설계해 보았다. 이후에는 출력할 값을 계산하는 조합 논리 회로와 계산된 값을 기반으로 레지스터의 값을 업데이트하는 순차 논리 회로를 결합하여 사용자 입력에 따라 다양한 결과를 내놓는 자판기 회로를 설계해 보았다.

이번 과제를 통해, 하드웨어의 작동을 행동 수준에서 기술하는 RTL 프로그래밍의 개념과 실제 RTL 프로그래밍 시 논리 회로의 설계 방식에 대해 익숙해질 수 있었으며, RTL 프로그래밍은 C와 같은 더 고수준의 프로그래밍 언어와는 다르게 하드웨어의 작동에 대한 정확한 예측과 그를 고려한 설계가 필요함을 알 수 있었다.