CSED311: Lab 1 (due Mar. 3)

김태연(20220140), 손량(20220323)

Last compiled on: Sunday 10th March, 2024, 16:44

1 Introduction

베릴로그 RTL 설계 방법을 익히기 위해 간단한 combinational circuit과 sequential circuit인 ALU와 vending machine을 구현해 본다.

2 Design

Vending machine은 크게 다음과 같은 모듈으로 나누어 설계하였다.

2.1 Coin Storage

이름에서 유추할 수 있듯, 사용자가 넣은 동전을 관리하는 모듈이다. 이 모듈은 다음 입력을 처리할 수 있도록 설계하였다.

- 사용자가 집어넣은 동전에 대한 입력
- 사용자가 구입한 물품의 가격
- 동전 반환 버튼 입력, 시간 초과에 해당하는 입력

Coin storage의 경우 사용자 입력에 대해 바로 출력이 일어날 필요가 없으므로 Moore machine 형태로 설계하였다. 사용자의 입력만 갖고 다음 상태를 계산하기 위해, 상태를 저장하는 레지스터에 대해 다음 상태를 담는 레지스터를 하나 더 추가해 구현하였다. 이 모듈에서 사용하는 레지스터는 다음 정보를 저장한다.

- 현재 저장하고 있는 동전의 총 합
- 반환할 동전의 조합
- 반환 버튼을 누르고 실제 동전 반환이 일어날 때까지 남은 시간

이러한 정보를 저장하는 것에서, coin storage가 가지는 상태는 다음과 같은 4-튜플으로 나타낼 수 있다.

$$(C, R, T, S) \in \{0,1\}^{32} \times \{0,1\}^{\texttt{kNumCoins}} \times \{0,1\}^{2} \times \{0,1\}$$

여기서 각 변수에 대한 설명은 다음과 같다.

- C – 현재 저장 중인 동전의 총 합

- R – 반환할 동전의 조합
- T – 반환 버튼을 누르고 실제 반환이 일어날 때까지 남은 시간
- S – 반환 버튼을 누르고 기다리는 중인지의 여부

2.2 Timekeeper

일정 시간 이상 동전을 투입하지 않거나, 조작을 하지 않은 경우 현재 투입되었던 동전을 반환하고 초기 상태로 되돌아가야 한다. 만약 해당 기능을 메인 모듈의 내부에서 구현할 경우, 각 clock tick 마다 현재 동작을 확인해야만 하기 때문에 별도의 모듈로 구현하였다. 해당 모듈 내에서 사용하는 레지스터는 타이머의 남은 시간과 타이머의 갱신 여부를 저장한다. 타이머를 갱신하지 않아야 하는 경우는 다음과 같다.

- 사용자 입력 이후 타이머가 0에 도달한 경우
- 사용자가 명시적으로 반환 버튼을 누른 경우

해당 경우 모두 사용자가 입력한 동전을 모두 반환하고, 타이머를 갱신하지 않는 상태로 되돌아간다. 이는 해당 모듈의 output register인 `timeout`에 신호를 출력함으로써 구현하였다. 해당 register는 Coin storage 모듈의 input으로 들어가 반환 동작을 하게 한다. Timekeeper 모듈의 경우 Mealy machine으로 구현하였다.

2.3 계산을 위한 combinational logic

동전 조합과 액수 사이의 변환에 대한 계산을 모두 sequential logic 내부에서 계산한다면 sequential logic의 코드가 복잡해진다. 이러한 상황을 피하기 위해 간단한 계산을 수행하기 위한 combinational logic은 별도의 모듈으로 구현하도록 설계하였다. 크게 다음과 같은 combinational logic 모듈을 설계하여 사용하였다.

- 현재 자판기에 저장된 액수로 구매할 수 있는 물품 표시
- 사용자의 물품 선택을 입력받아 사용되는 액수를 계산
- 동전 조합을 입력받아 총 합을 계산
- 주어진 액수를 반환할 수 있는 동전의 조합을 계산

위에서 나열한 모듈 중 마지막 모듈의 경우에는 출력 조건 상, 각 동전 종류는 한 사이클에 최대 한 개만 반환할 수 있으므로 필요하다면 sequential logic에서 여러 번 입력 값을 바꾸어 가면서 출력 값을 이용하는 것을 가정하였다.

3 Implementation

각 베릴로그 파일에 대한 설명은 다음과 같다.

3.1 `coin_storage.v` – Coin storage 모듈 구현

Coin storage 모듈을 구현해 놓은 파일이다. 모듈의 입력은 다음과 같다.

- `clk` – clock signal
- `reset_n` – reset signal
- `coin_value_table` – 비트 위치 → 동전 액수 매핑
- `coin_input` – 넣은 동전의 조합
- `buy_price_input` – 구입한 물품의 가격
- `trigger_return` – 동전 반환 버튼
- `timeout` – 대기 시간 종료 여부

Moore machine 형태로 구현하기 때문에, 베릴로그 코드는 combinational logic과 sequential logic, 크게 두 부분으로 나뉜다. combinational logic의 부분에서는 모듈의 입력에서 다음 상태를 계산한다. 설계한 상태를 저장하는 레지스터는 다음과 같이 사용하였다.

- `coin_value` – C에 해당
- `return_coins` – R에 해당
- `remaining_time` – T에 해당
- `timer_started` – S에 해당

베릴로그 구현에서 레지스터와 wire의 mutlidriven 상황 등을 피하기 위하여 `always @(*)` 부분에서는 combinational circuit을 구현하여 다음 상태에 레지스터에 저장할 값인 `coin_value_next`, `return_coins_next`, `remaining_time_next`, `timer_started_next`을 계산하고, `always @(posedge clk)` 부분에서는 sequential circuit을 구현하여 `coin_value`, `return_coins`, `remaining_time`, `timer_started` 레지스터를 업데이트하도록 코딩하였다.

최종적으로, 모듈의 출력은 다음과 같다.

- `coin_value` – 현재 저장된 동전의 액수
- `return_coins` – 반환할 동전에 대한 출력

해당 파일에서는 `coin_value_calculator.v` 및 `return_value_calculator.v` 에서 정의된 모듈을 사용한다. 반환할 동전을 결정하고, 해당 모듈의 입력으로 주어지는 `trigger_return` 신호에 따라 `coin_value_calculator` 및 `return_value_calculator`를 통해 계산된 동전 조합을 매 clock cycle마다 하나씩 동전을 반환하도록 코드를 구성했다.

3.2 `timekeeper.v` – Timekeeper 모듈 구현

Timekeeper 모듈을 구현해 놓은 파일이다. 모듈의 입력은 다음과 같다.

- `clk` – clock signal
- `reset_n` – reset signal
- `current_total` – 현재 동전 액수의 합

- `coin_input` – 넣은 동전의 조합
- `selection_input` – 구입한 물품 입력
- `trigger_return` – 동전 반환 버튼

상태를 저장하는 레지스터는 다음과 같다.

- `remaining_time` – 남은 시간
- `timer_started` – 카운트다운의 시작 여부

Mealy machine 형태로 구현하기 때문에 베릴로그 코드는 sequential logic에 대한 부분만 있다. 동전 반환 버튼을 눌렀을 때나 10사이클의 시간이 모두 지나갔을 경우에는 출력 `timeout` 핀으로 시간이 끝났음을 신호로 보내고, 카운트다운을 리셋하도록 하였다. 동전이나 물품 선택 등 사용자의 입력이 들어올 경우 카운트다운을 시작하도록 하였다.

매 clock cycle마다 남은 시간을 감소시키고, 사용자의 선택이 있을 때 마다 시간을 초기화하도록 설계하였다. 반환은 사용자가 명시적으로 반환 버튼을 누르거나 시간이 종료되는 경우 반환 신호를 출력한다.

3.3 `item_display.v` – 구매할 수 있는 물품 표시

현재 잔액과 상품의 값을 비교하여 각 상품의 구매 가능 여부를 반환한다. 다양한 `` `kNumItems `` 값에 대응할 수 있도록 `always @(*)` 내부에서 `for`문을 사용하여 구현하였다.

3.4 `item_selector.v` – 사용자의 물품 선택 처리

사용자의 선택으로부터 구매가 가능한지 판별하며, 한번에 여러 개의 구매 버튼을 눌러도 반복문 내에서 모든 상품에 대해 구매 여부를 검사하여 최종적으로 구매 가격의 합계와 구매 물품의 종류를 반환한다. 이 모듈 역시 `item_display` 모듈과 마찬가지로 다양한 `` `kNumItems `` 값에 대응할 수 있도록 `always @(*)` 내부에서 `for`문을 사용하여 구현하였다.

3.5 `coin_value_calculator.v` – 동전 조합에 따른 액수 계산

한 clock cycle마다 동전의 조합 형태로 입력이 들어올 때, 동전의 액수와 동전의 조합을 입력으로 받아서 조합에 해당하는 액수를 반환하는 모듈을 설계하였다. 이 모듈 역시 `item_display` 모듈과 마찬가지로 다양한 `` `kNumItems `` 값에 대응할 수 있도록 `always @(*)` 내부에서 `for`문을 사용하여 구현하였다.

3.6 `return_value_calculator.v` – 반환할 수 있는 동전 조합 계산

입력되는 동전과 유사하게, 매 clock cycle마다 반환 가능한 동전 조합을 계산한다. 모든 동전의 반환을 위해 여러 cycle이 소요될 수 있으며, 반복문에서 각 반복마다 작은 금액의 동전부터 채워나가는 방법으로 반환을 진행한다.

3.7 `vending_machine.v` – Vending machine 구현

Vending machine 구현의 top-level에 해당하는 모듈이다. `coin_storage`, `item_display`, `item_selector`, `timekeeper` 모듈을 연결하여 입력에 대해 출력을 수행하도록 하였다.

4 Discussion

과제에서 고려할 필요가 없었기에 간단한 구현을 위해 반환 버튼을 누르는 동시에 동전을 투입하는 등 race condition이 일어날 만한 상황에 대한 처리는 구현하지 않았다. race condition이 발생할 때의 처리를 미리 정의해 놓은 후에 이를 구현하여 설계를 개선해 보는 것도 유익할 것이라고 생각한다.

5 Conclusion

주어진 테스트벤치의 실행 결과 다음과 같이 모든 테스트를 통과함을 알 수 있었다.

```
### SIMULATING ###
 initial test
PASSED : available item: 0, expected 0

 Insert 100 Coin test
PASSED : available item: 1, expected 1
PASSED : available item: 3, expected 3

 Insert 500 Coin test
PASSED : available item: 3, expected 3
PASSED : available item: 7, expected 7
PASSED : available item: 15, expected 15

 Insert 1000 Coin test
PASSED : available item: 7, expected 7
PASSED : available item: 15, expected 15

 Select 1st Item test
PASSED : available item: 15, expected 15
PASSED : available item: 15, expected 15
PASSED : available item: 7, expected 7
PASSED : available item: 3, expected 3
PASSED : available item: 0, expected 0

 Select 2nd Item test
PASSED : available item: 15, expected 15
PASSED : available item: 7, expected 7
PASSED : available item: 3, expected 3
PASSED : available item: 0, expected 0

 Select 3rd Item test
PASSED : available item: 15, expected 15
PASSED : available item: 7, expected 7
PASSED : available item: 1, expected 1

 Select 4th Item test
PASSED : available item: 7, expected 7
PASSED : available item: 1, expected 1
```

```
 Wait Return test
PASSED : wait 10 cycle
PASSED : return 2800

 Trigger Return test
PASSED : return 4800
TEST END
SUCCESS : 25 / 25
```