디지털시스템 - Verilog를 이용한 Vending Machine 프로젝트

목차

1) 설계 내용
    1. 설계 목표
    2. 요구 사항
    3. 입출력 정의
    4. 상태 정의
2) 상태표
    1. c_beverage_state, n_beverage_state 상태표
    2. c_change_state, n_change_state 상태표
    3. c_state, n_state 상태표
3) 결과 분석
    1. 동시 입력의 경우
    2. 초과금액이 발생한 경우
    3. 자판기의 돈이 음료의 가격(10,000원)보다 적을 경우
    4. 잔돈 반환의 경우

1) 설계 내용

1. 설계 목표

천원과 오천원을 받아 만원짜리 음료수를 뽑을 수 있는 자판기 설계(잔돈 반환 기능이 있는) 하는 것을 목표로 한다.

2. 요구 사항

1. 여러 입력이 들어오는 경우 아무런 동작을 수행하지 않는다.

   잔돈 반환과 동전 입력이 동시에 입력으로 들어온 경우, 음료 호출과 잔돈 반환이 동시에 입력으로 들어온 경우, 음료 호출과 동전 입력이 동시에 입력으로 들어온 경우, 음료 호출과 잔돈 반환과 동전 입력이 모두 동시에 입력으로 들어온 경우 입력을 무시한다.

2. 자판기에 20,000원을 초과해서 넣을 수 없다. 만약 자판기에 20,000원을 초과하여 돈이 들어온다면 20,000원을 초과하여 오르지 않고 초과된 돈의 양만큼 잔돈으로 반환된다.
3. 자판기의 돈이 음료의 가격(10,000원)보다 적을 경우 뽑을 수 없다.
4. 잔돈을 반환할 때 5,000원이 반환이 가능하면 5,000원을 반환하고, 불가능하다면 1,000원을 반환한다.
5. 모든 출력은 1 cycle 지연되어 출력된다.
6. 모든 출력은 상태에 맞게 한 주기만 출력되어야 한다. (돈이 충분한 상황에서 잔돈 반환을 한 번만 수행했는데 계속 잔돈 반환 출력이 올라가 있는 경우는 틀린 경우)

3. 입출력 정의

INPUT

coin_in(2bit)	동전 입력을 나타내는 것으로 **2'b00** 의 경우 동전 입력이 없음을 나타내고 **2'b01**의 경우 천원 입력을 나타내고 **2'b10**의 경우 오천원 입력을 나타낸다. **2'b11**이 입력으로 오는 경우는 없다고 가정한다.
beverage_take	음료를 뽑는 버튼으로 **1'b0**의 경우 명령 없음을 나타내고 **1'b1**의 경우 음료 뽑기 명령을 나타낸다.
change_take	잔돈 반환 버튼으로

	1'b0의 경우 명령 없음을 나타내고 **1'b1**의 경우 잔돈 반환 명령을 나타낸다
clk	clock의 의미로 본 프로젝트에서 clock의 주기는 **10ns**이다.
rstn	상태 초기화를 위한 입력이다.

OUTPUT

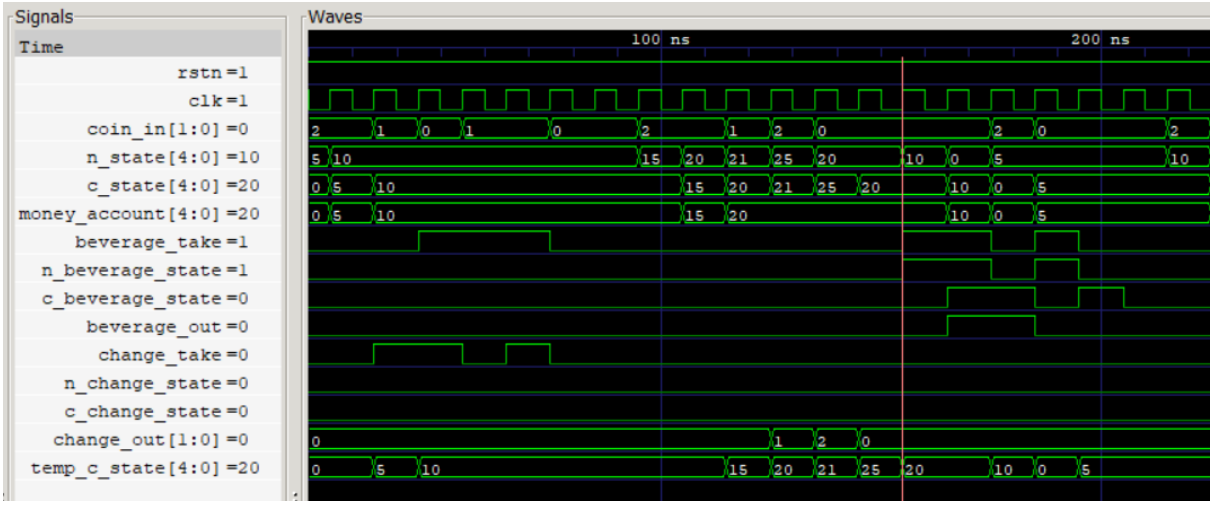
money_account(5bit)	현재 넣은 돈의 상태를 나타내는 것으로 예를 들어 **5'b00111 = 7(10진수)**는 7000원을, **5'b01000 = 8(10진수)**는 8000원을 나타낸다.
beverage_out	음료가 나오는지 나오지 않는지를 나타내는 것으로 **1'b0**의 경우 음료가 나오지 않은 것을 나타내고 **1'b1**의 경우 음료가 나왔음을 나타낸다.
change_out(2bit)	잔돈이 반환이 되었는지 나타내는 것으로 **2'b00**의 경우 잔돈이 반환되지 않은 것을, **2'b01**의 경우 천원으로 반환된 것을, **2'b10**의 경우 오천원으로 반환된 것을 나타낸다. **2'b11**인 출력의 경우는 없다.

4. 상태 정의

상태	
c_beverage_state	음료 버튼을 누른 상태(**take = 1'b0**), 누르지 않은 상태(**not take = 1'b1**)을 나타낸다.

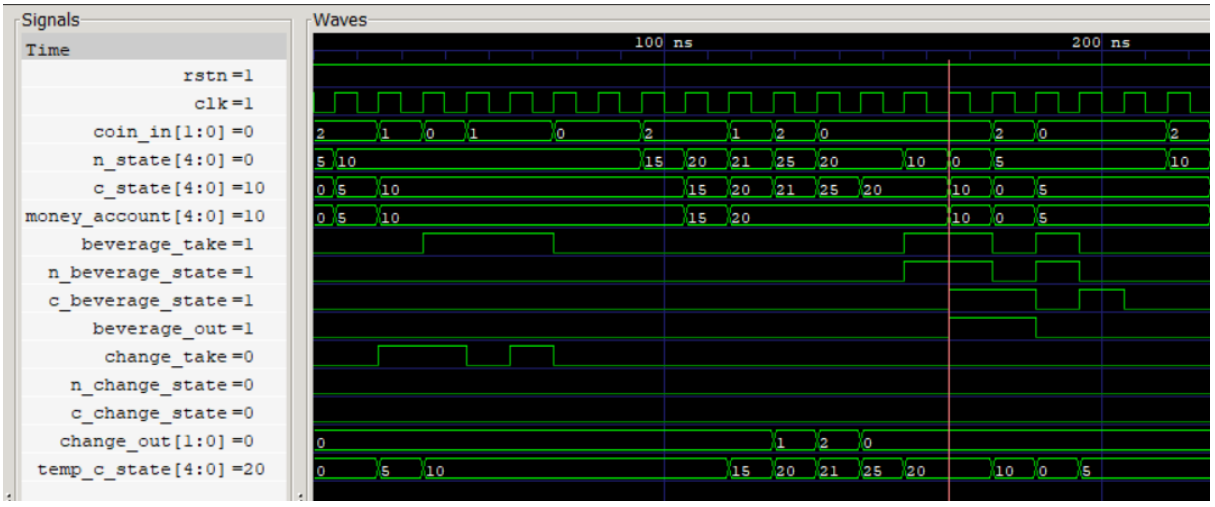
상태	
n_beverage_state	beverage_take의 입력에 따라 다음 c_beverage_state의 상태를 결정한다.
c_change_state	잔돈 반환 버튼을 누른 상태(**take = 1'b0**), 누르지 않은 상태(**not take = 1'b1**)을 나타낸다.
n_change_state	change_take의 입력에 따라 다음 c_change_state의 상태를 결정한다.
c_state(5bit)	0원부터 2만원(**S0 ~ S20**) 그리고 2만원을 초과할 경우의 상태(**S21 ~ S25**)를 나타낸다.
n_state(5bit)	coin_in / beverage_take / change_take의 입력에 따라 다음 상태를 결정한다.
temp_c_state	올바른 beverage_out과 change_out을 출력하기 위한 c_state 상태를 저장한다.

- 상태 temp_c_state에 대한 설명

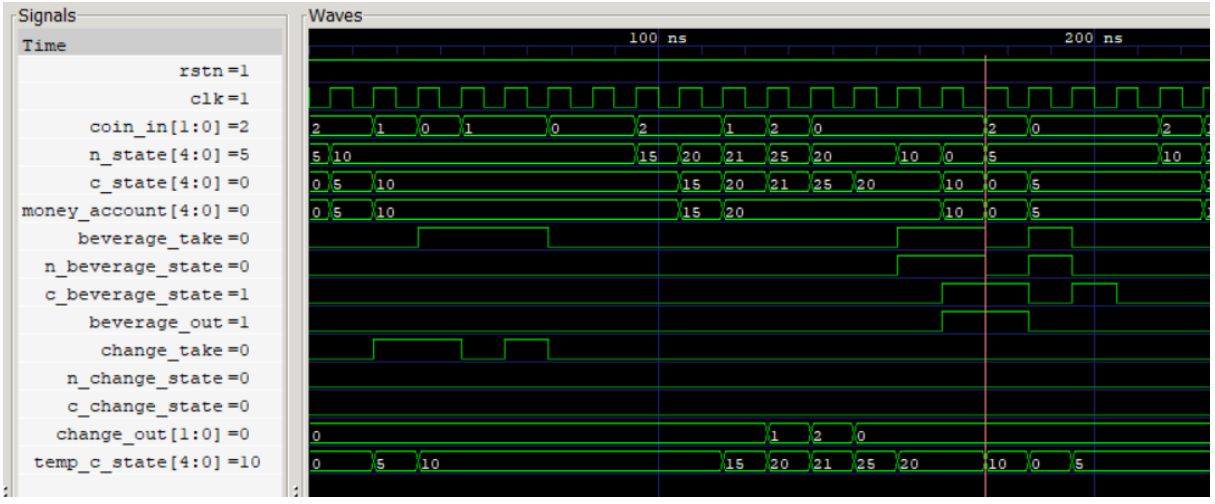


만약에 c_state를 보고 출력을 결정한다고 가정해보자.

다음 빨간 선 부분을 보면, 자판기 안에(money_account) 20,000원이 있는 상태에서 음료 호출(beverage_take)를 하였다. 그러면 c_state의 다음 상태를 나타내는 n_state는 10진수로 10이 될 것이고, 다음 posedge 에서 c_state는 10진수로 10이 될 것이다.



다음 빨간 선 부분을 보면, c_state 상태가 10진수로 10이기 때문에 음료를 반환(beverage_out)한다. 그리고 자판기 안에(money_account) 10,000원이 있는 상태에서 음료 호출(beverage_take)를 하였다. 그러면 c_state의 다음 상태를 나타내는 n_state는 10진수로 0이 될 것이고, 다음 posedge에서 c_state는 10진수로 0이 될 것이다.



다음 빨간 선 부분을 보면, 위 그림은 temp_c_state의 상태를 보고 음료 반환(beverage_out)을 결정하였기 때문에 정상적으로 출력되었지만, c_state 상태를 보고 출력을 결정한다고 가정하면 c_state의 상태가 10진수로 0이기 때문에 음료 호출(beverage_out)은 0이 되어야 한다.

하지만 우리는 자판기 안에 20,000원이 있는 상태에서 음료 호출을 2번 한 것이기 때문에 음료 반환(beverage_out) 또한 2번 실행되어야 한다.

이러한 이유로 c_state 상태를 1 cycle 지연시켜 저장하는 temp_c_state 상태 변수를 활용하는 것이다.

2) 상태표

1. c_beverage_state, n_beverage_state 상태표

temp_c_state 상태가 S0 ~ S9인 경우

현재 상태 (c_beverage_state)	음료 호출 (beverage_take)	다음 상태 (n_beverage_state)	출력 (beverage_out)
not_take	1	take	0
not_take	0	not_take	0
take	1	take	1 (if temp_c_state ≥ S10)
take	0	not_take	0

2. c_change_state, n_change_state 상태표

temp_c_state 상태가 S0인 경우

현재 상태 (c_change_state)	잔돈 반환 (change_take)	temp_c_state	다음 상태 (n_change_state)
take	-	if(temp_c_state ≠ s0)	take
not_take	1	-	take
not_take	0	-	not_take
take	1	-	take
take	0	-	not_take

현재 상태 (c_change_state)	c_state	temp_c_state	출력 (change_out)
not_take	if(c_state = S25)	-	10
not_take	if(S21 ≤ c_state ≤	-	01

현재 상태 (c_change_state)	c_state	temp_c_state	출력 (change_out)
	S24)		
not_take	if(c_state ≤ S20)	-	00
take	-	if(temp_c_state = S0)	00
take	-	if(S1 ≤ temp_c_state ≤ S4)	01
take	-	if(S5 ≤ temp_c_state ≤ S25)	10

3. c_state, n_state 상태표

- **잔돈 반환에 따른 상태표**

S0

현재 상태(c_state)	잔돈 반환 (change_take)	c_change_state	다음 상태(n_state)
S0	1	X	S0
S0	X	1 (take)	S0

S1 ~ S4 : S(n)는 Sn을 의미, 즉 S(1)은 S1

현재 상태(c_state)	잔돈 반환 (change_take)	c_change_state	다음 상태(n_state)
S(n)	1	X	S(n-1)
S(n)	X	1 (take)	S(n-1)

S5 ~ S20 : S(n)는 Sn을 의미, 즉 S(1)은 S1

현재 상태(c_state)	잔돈 반환 (change_take)	c_change_state	다음 상태(n_state)
S(n)	1	X	S(n-5)
S(n)	X	1 (take)	S(n-5)

S21 ~ S25 : S(n)는 Sn을 의미, 즉 S(1)은 S1

현재 상태(c_state)	잔돈 반환 (change_take)	c_change_state	다음 상태(n_state)
S(n)	1	X	S15
S(n)	X	1 (take)	S15

- **음료 호출에 따른 상태표**

S0 ~ S9 : S(n)는 Sn을 의미, 즉 S(1)은 S1

현재 상태(c_state)	음료 호출(beverage_take)	다음 상태(n_state)
S(n)	0	S(n)
S(n)	1	S(n)

S10 ~ S20 : S(n)는 Sn을 의미, 즉 S(1)은 S1

현재 상태(c_state)	음료 호출(beverage_take)	다음 상태(n_state)
S(n)	0	S(n)
S(n)	1	S(n - 10)

S21 ~ S25 : S(n)는 Sn을 의미, 즉 S(1)은 S1

현재 상태(c_state)	음료 호출(beverage_take)	다음 상태(n_state)
S(n)	0	S20
S(n)	1	S10

- **동전 입력에 따른 상태표**

S0 ~ S20 : S(n)는 Sn을 의미, 즉 S(1)은 S1

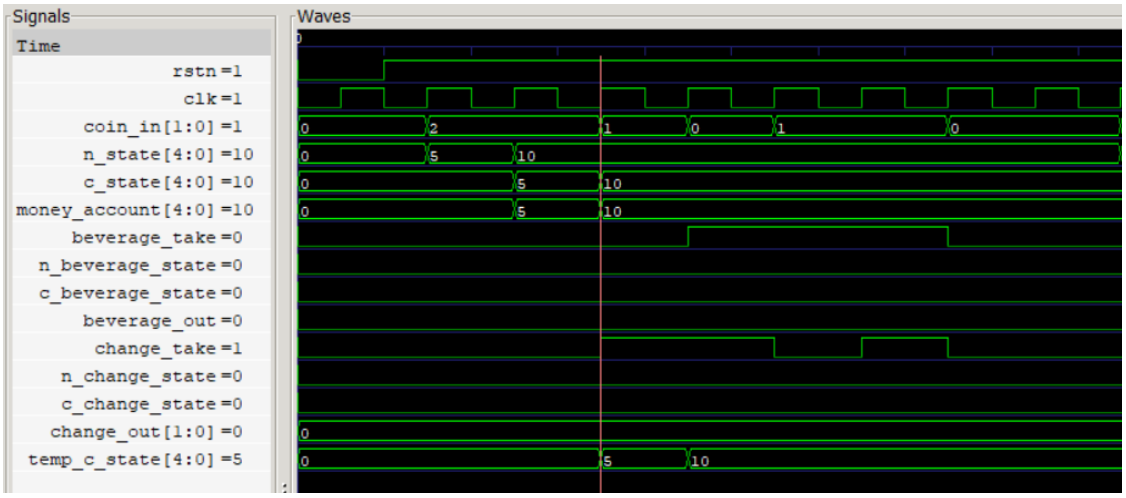
현재 상태(c_state)	동전 입력(coin_in)	다음 상태(n_state)
S(n)	1	S(n + 1)
S(n)	2	S(n + 5)

S21 ~ S25 : S(n)는 Sn을 의미, 즉 S(1)은 S1

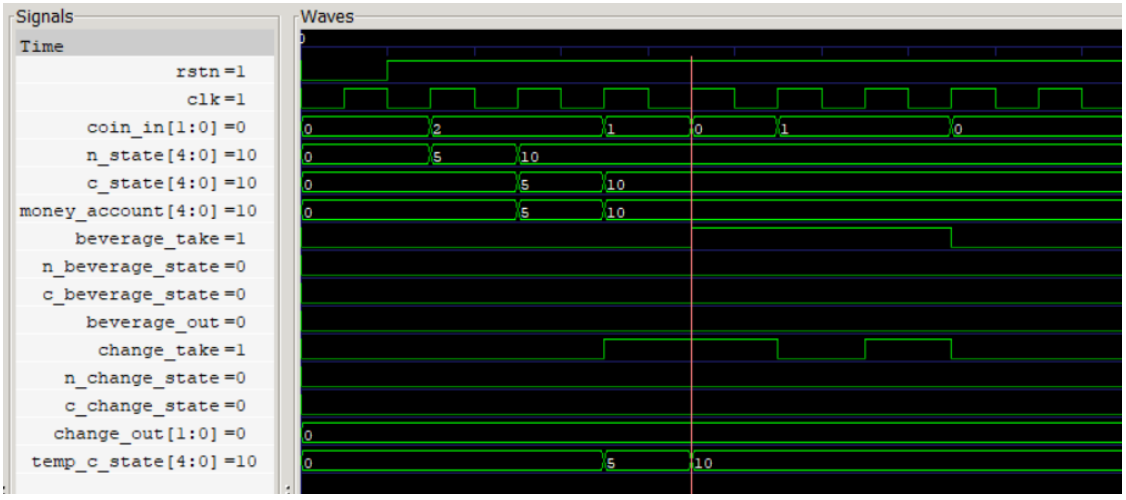
현재 상태(c_state)	동전 입력(coin_in)	다음 상태(n_state)
S(n)	1	S21
S(n)	2	S25

3) 결과 분석

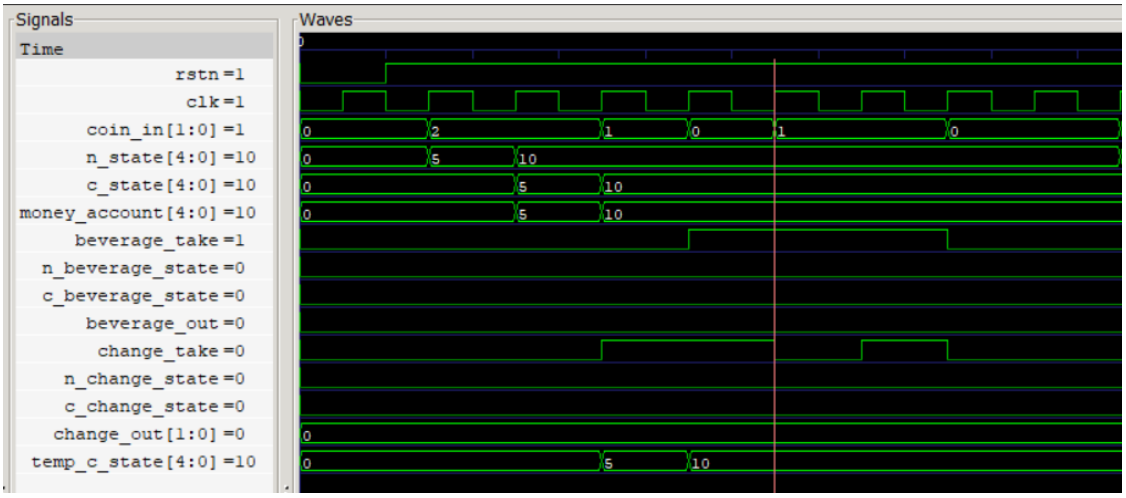
1. 동시 입력의 경우



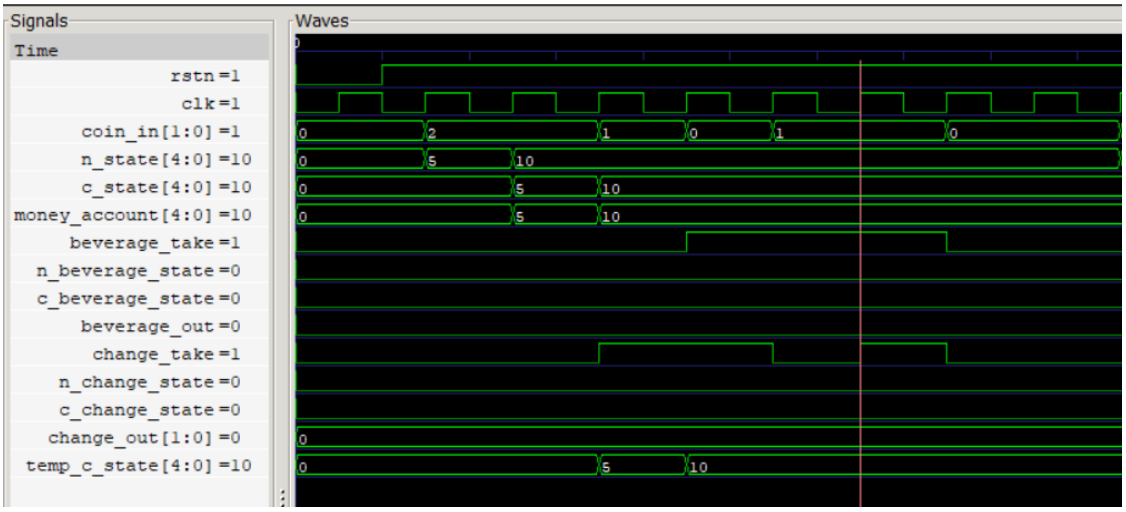
잔돈 반환과 동전 입력이 동시에 들어오는 경우 입력을 무시한다.



음료 호출과 잔돈 반환이 동시에 들어오는 경우 입력을 무시한다.

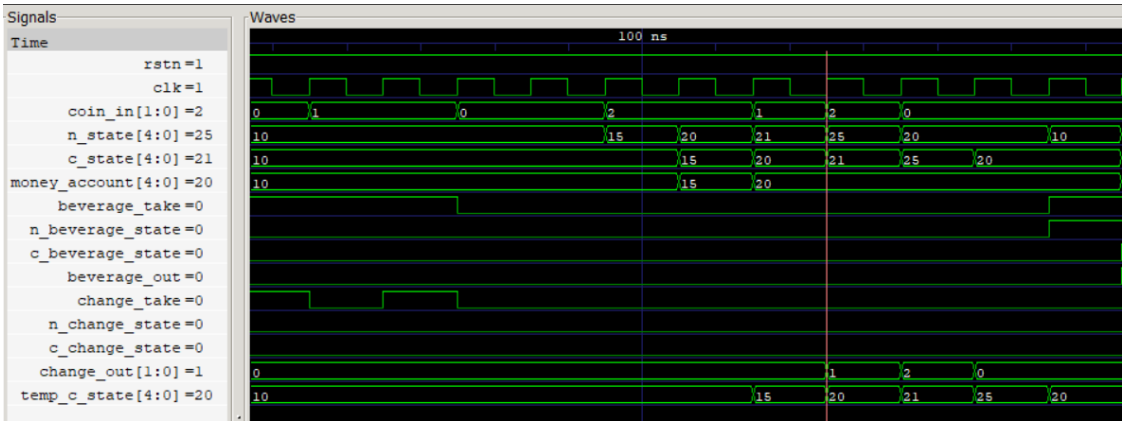


음료 호출과 동전 입력이 동시에 들어오는 경우 입력을 무시한다.

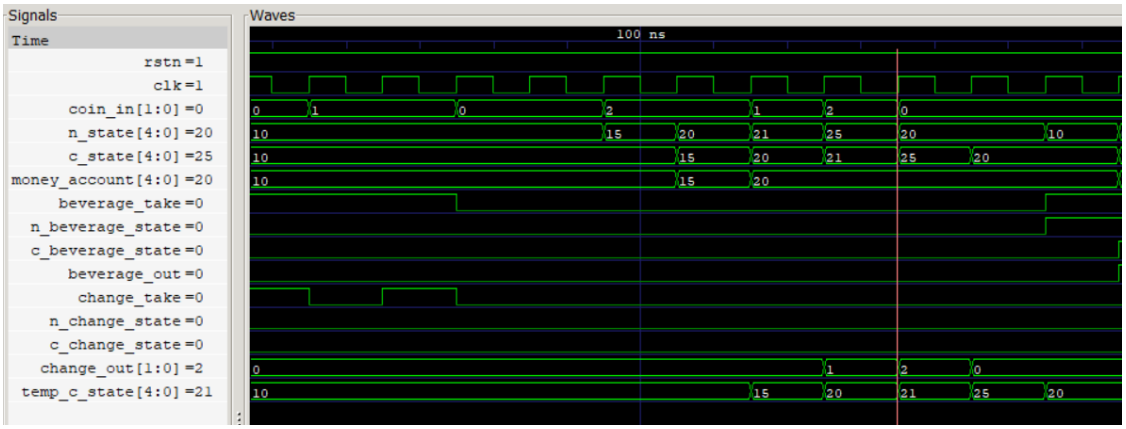


음료 호출과 잔돈 반환과 동전 입력이 모두 동시에 들어오는 경우 입력을 무시한다.

2. 초과금액이 발생한 경우

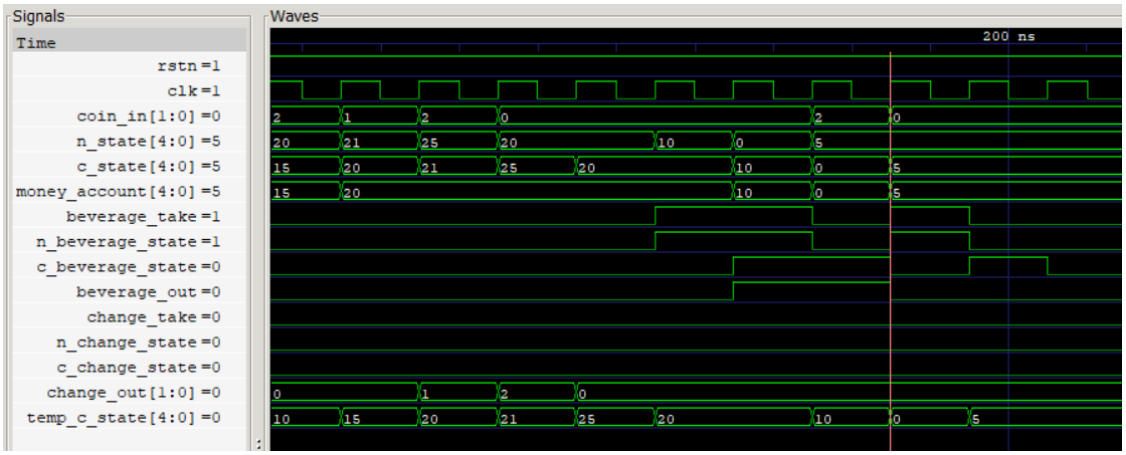


자판기 안에 20,000원이 있는 상황에서 1,000원을 동전 입력하면 자판기의 안에 돈이 21,000원이 되어 자동으로 1,000원이 반환된다.



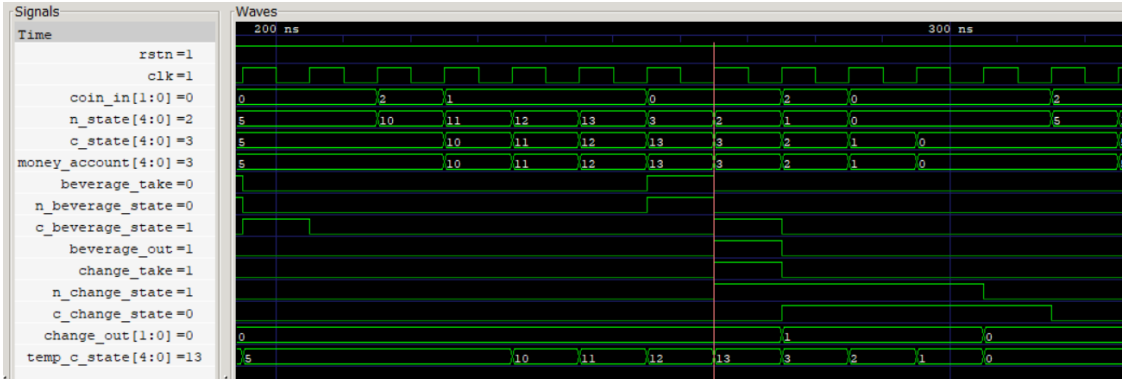
자판기 안에 20,000원이 있는 상황에서 5,000원을 동전 입력하면 자판기의 안에 돈이 25,000원이 되어 자동으로 5,000원이 반환된다.

3. 자판기의 돈이 음료의 가격(10,000원)보다 적을 경우

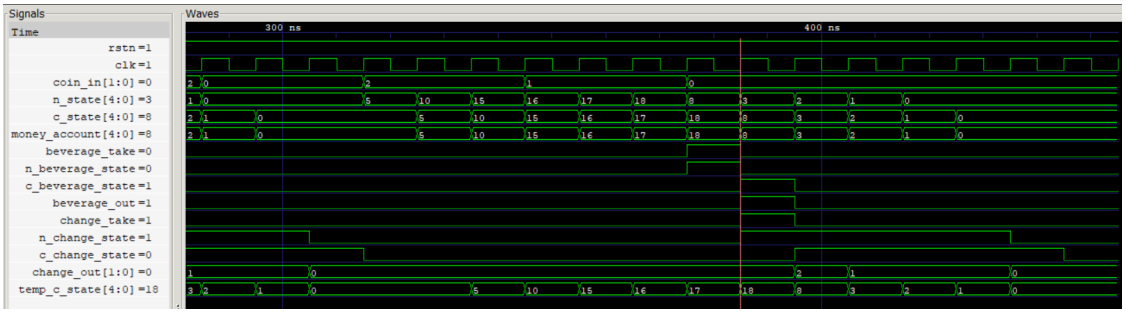


자판기 안에 돈이 5,000원이 있는 상황에서 음료 호출 하면 다음 cycle에서 beverage_out이 0이 되어 음료가 호출되지 않은 것을 볼 수 있다.

4. 잔돈 반환의 경우



자판기 안에 돈이 3,000원이 있는 상황에서 잔돈 반환을 하면 다음 3번의 cycle에서 1,000원이 3번 반환되는 것을 볼 수 있다.



자판기 안에 돈이 8,000원이 있는 상황에서 잔돈 반환을 하면 다음 cycle에서 5,000원이 반환되고 그 다음 3번의 cycle에서 1,000원이 반환되는 것을 볼 수 있다.