SoC Verification HW 1.4 Arvind Singh Rathore R10943157

In Hw 1.3, I found and fixed the bug which forced the machine to eat money.

However, I also observe that if we don't initialize the machine, it starts behaving weirdly and gives unexpected output. So, it becomes important to initialize the machine first. I have written a monitor to ensure that the initialize condition is always monitored.

```
119  assign checkforinitialized = !initialized && ((serviceTypeOut != `SERVICE_OFF) || (outExchange != 0) || (itemTypeOut != `ITEM_NONE));
```

Here I input a pattern to the machine in which the initialized signal never became 0, i.e not initialized. I observe that the machine gives some unexplainable output as indicated by the monitor written by me above (last bit checkforinitialized is 1). Ideally, we want the output to be 0 if we do not initialize.

Not initialized

```
 1 38 11
 2 XXXXXXXXXX1
 3 11000110001
 4 00XXXXXXXX1
 5 00XXXXXXXX1
 6 00XXXXXXXX1
 7 00XXXXXXXX1
 8 00XXXXXXXX1
 9 00XXXXXXXX1
10 00XXXXXXXX1
```

Unexplainable random output

```
 1 0000000000000000
 2 0100000000000001
 3 1011000000000001
 4 1011000000000001
 5 1011000000000001
 6 1011000000000001
 7 1011000000000001
 8 1000000000000001
 9 1000000000000001
10 1000000000100001
```

So, in order to fix this weird situation, I modify the code as

```
if(initialized)
begin
  case (serviceTypeOut)
    `SERVICE_ON   : begin
```

And correspondingly I got the correct expected output on simulating again.

```
 1 0000000000000000
 2 0000000000000000
 3 0000000000000000
 4 0000000000000000
 5 0000000000000000
 6 0000000000000000
 7 0000000000000000
 8 0000000000000000
 9 0000000000000000
10 0000000000000000
```

Second monitor I wrote is to check if correct change is being tendered back to the customer

```
assign checkitemvalue = itemTypeOut == `ITEM_A ? `COST_A :
                        itemTypeOut == `ITEM_B ? `COST_B :
                        itemTypeOut == `ITEM_C ? `COST_C : 8'd0;

assign checkcorrectchange = initialized && (serviceTypeOut == `SERVICE_OFF) && ( outExchange != inputValue - checkitemvalue);
```

So, I think with these additional 2 monitors who check for the initialize condition as well as the most important part-the tendering of the correct change, they can guard the design well.

Moving on to the test pattens and verification results:
I have written a random pattern generator code rand.cpp
I use this function to generate test cases with different sets of inputs and to ensure proper cycle to complete the exchange, I have given 20 waiting cycles.

I created 3 test pattern files with different headers (Total number of input patterns including waiting cycle as described in V3 tutorial).

Input Pattern File name	Output File name	Total headers
Test1.pattern	Test1.output	10000
Test2.pattern	Test2.output	100000
Test3.pattern	Test3.output	1000000

```
mikasa@cthulhu:~/socv/Homework1/vending-simple$ .././V3/v3
v3> read rtl vending.v

v3> sim ntk -input Test1.pattern -output Test1.output
10000 Patterns are Simulated from Test1.pattern  (time = 1.2105  sec)

v3> sim ntk -input Test2.pattern -output Test2.output
100000 Patterns are Simulated from Test2.pattern  (time = 10.459  sec)

v3> sim ntk -input Test3.pattern -output Test3.output
1000000 Patterns are Simulated from Test3.pattern  (time = 100.69  sec)
```

Of course, I can't manually open each simulated output file and check the output for any discrepancies, I wrote a simple python code to check the last few monitor bits of the output.

```
mikasa@cthulhu:~/socv/Homework1/vending-simple$ python3 arvindbugmonitor.py Test1.output
Total 0 bug(s) based on checkforinitialized, p and checkcorrectchange monitors!
mikasa@cthulhu:~/socv/Homework1/vending-simple$ python3 arvindbugmonitor.py Test2.output
Total 0 bug(s) based on checkforinitialized, p and checkcorrectchange monitors!
mikasa@cthulhu:~/socv/Homework1/vending-simple$ python3 arvindbugmonitor.py Test3.output
Total 0 bug(s) based on checkforinitialized, p and checkcorrectchange monitors!
mikasa@cthulhu:~/socv/Homework1/vending-simple$
```

Since, I do not get any bugs after simulation of so many cases, I think the code is very well guarded by my monitors. Also since the main case of the bug could be majorly the case of not tendering the correct change as fixed in hw 1.3, and I since I have written a monitor to check the output change, I think I'm satisfied.