(1) Your monitors and how sound you think they can guard the design

#Monitor(flag)

```
// Property Logic
/***** whether the change is right *****/
//assign p = initialized && (serviceTypeOut == `SERVICE_OFF) && (itemTypeOut == `ITEM_NONE) && (outExchange != inputValue); // catch the bug
assign p = initialized && (serviceTypeOut == `SERVICE_OFF) &&
         (inputValue != (outExchange + ((itemTypeOut==`ITEM_A)*`COST_A+(itemTypeOut==`ITEM_B)*`COST_B+(itemTypeOut==`ITEM_C)*`COST_C)));
```

如同上圖所示，我把 p 修改成更 aggressive 的版本. 每次機器輸出結果的時候，都確認投入的金額是否與輸出的商品+金額相同. 不像先前，只看購買失敗的結果. 我認為這應該能把所有錯誤的情況都抓出來!

(2) Test patterns and their verification results

#生成測資一: notenough.pattern (沒有足夠的投入零錢)

```
22 11
XXXXXXXXXX0
01110000001
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
```

```
00000000000000000
01000000000000000
10010000000000000
10000000000000000
10000000000000000
10000000000000000
10000000000000000
10000010000000000
10000100000000000
10000110000000000
00000110000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
```

#生成測資二: nocoin.pattern(找零時，零錢不夠，取消交易)

```
33 11
XXXXXXXXXX0
01001000001
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
01001000001
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
00XXXXXXXX1
```

```
00000000000000000
01000000000000000
10010000000000000
10010000000000000
10010000000000000
10010000000000000
10010000000000000
10010010000000000
10010100000000000
00010100000000000
01000000000000000
01000000000000000
01000000000000000
10010000000000000
10010000000000000
10010000000000000
10010000000000000
10010000000000000
10000000000000000
10000000000000000
10000000000010000
10000000000010000
10000000000010000
00000000000010000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
01000000000000000
```

#生成測資三—success.pattern(交易成功，三種品項各一次)

```
34 11
XXXXXXXXX0
01000001001
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
10000000011
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
11101101001
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
00XXXXXXX1
```

```
00000000000000000
01000000000000000
10010000000000000
10010000000000000
10010000000000000
10010000000000000
10010000000000000
10010010000000000
10010100000000000
00010100000000000
01000000000000000
01000000000000000
01000000000000000
10100000000000000
10100000000000000
10100000000000000
10100000000010000
10100000000100000
10100000000110000
10100000000110000
10100000010110000
10100000010110000
00100000010110000
01000000000000000
10110000000000000
10110000000000000
10110000000000000
10110000000000000
10110000010000000
10110000010000000
00110000010000000
01000000000000000
01000000000000000
01000000000000000
```

#總結

在相對 aggressive 的 monitor，以及一一討論各種情況下，都沒有發現 monitor 有再被觸發的時機，因此有充足的信心認為這個修改後的版本是正確的.

另一方面，我也有使用 v3 的驗證功能，像是

- verify sim p1
- verify umc p1
- verify umc –noprove p1

但都沒有偵測出 counter-example.

值得一提的是, verify pdf p1 有在 level 2 有檢測出一個 counter-example，然而在輸出成 pattern 檔以後，查無異狀；模擬後也沒能讓 monitor 的 p=1 成立. 因此懷疑是操作不當所導致!