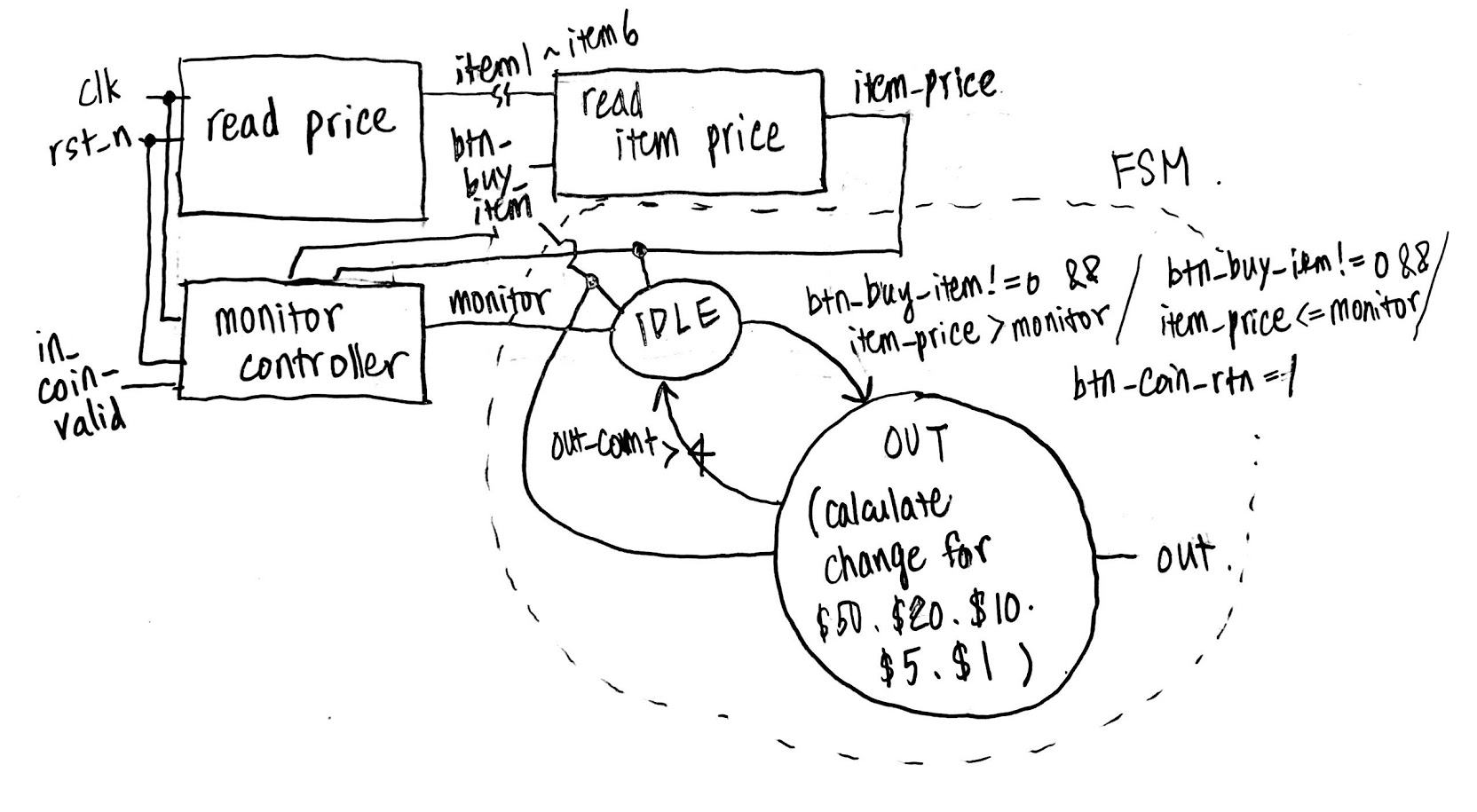
Block Diagram (with FSM implementation)

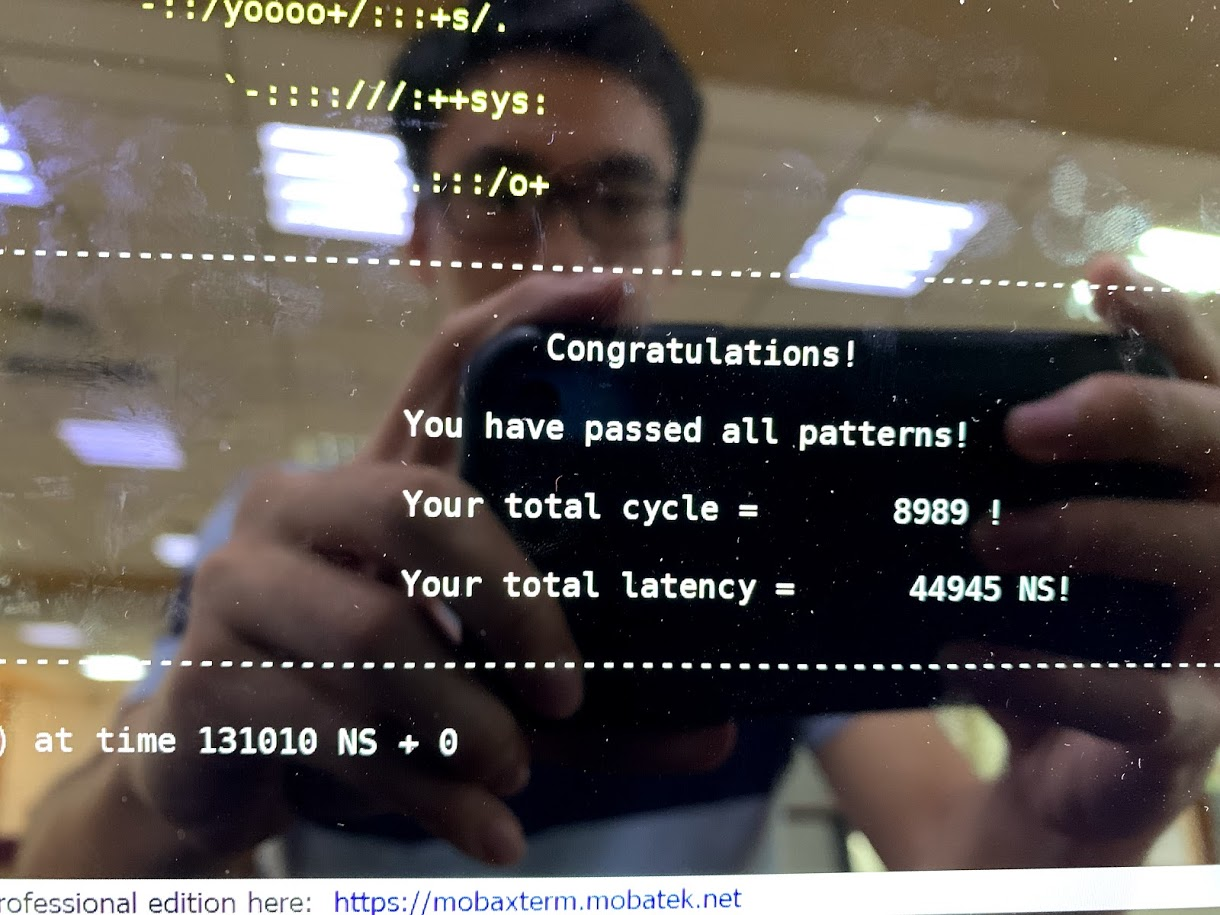


**設計優化過程**

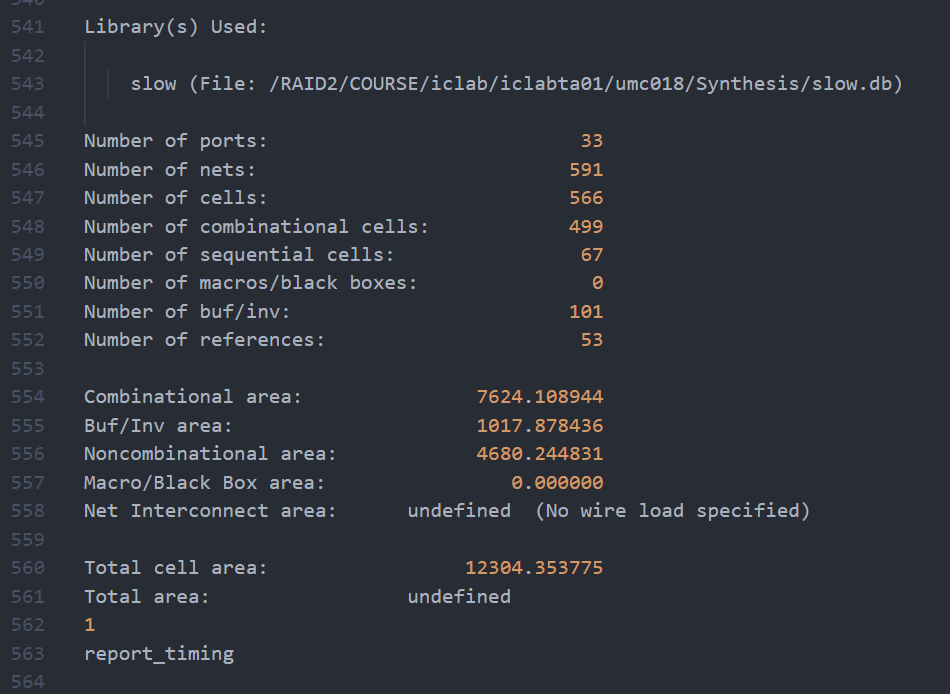
版本1:

最初的版本在計算找零錢方面幾乎全引用以前lab05的code，只有另外外加讀價、判斷購買商品價格、顯示器控制等三個功能。但助教本來就有在lab的時候提到，那個找零錢state的設計是最基礎好懂的，但就會花很長的時間完成計算，而且，也十分耗面積。

時間方面看下圖就知道latency長的驚人，因為單單找一種零錢就要跳好多次state，所以從btn\_buy\_item到out\_valid就會有好幾個cycle的delay，這個問題在下面的版本2有進行解決。



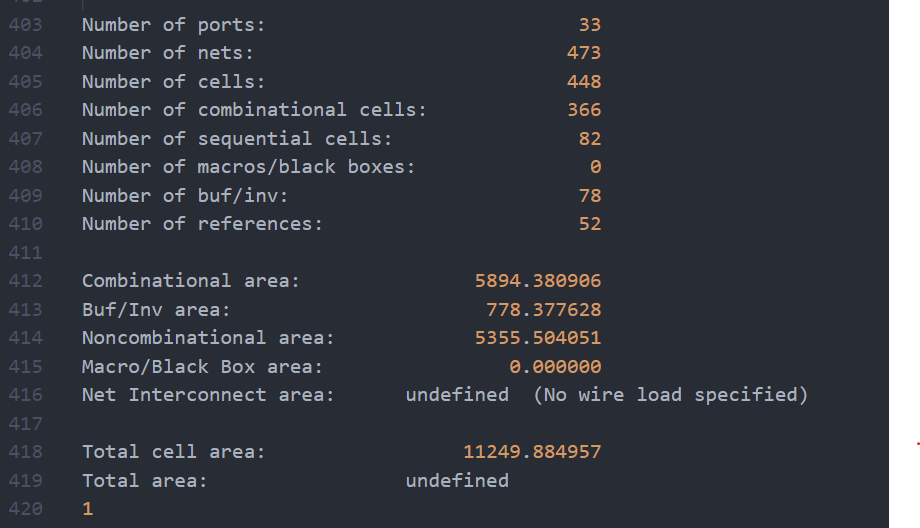
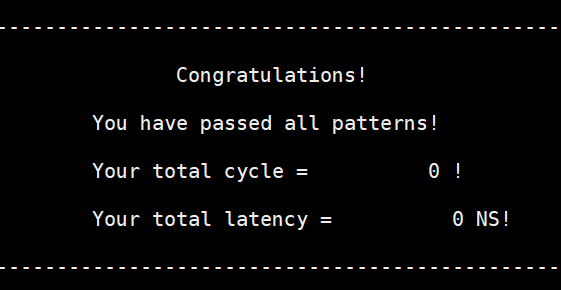
把每種零錢都存入一個register也花費了很多空間，$50, $20, $10, $5, $1都要有一個register去存，十分地耗面積，這個問題我也有在版本2改良。這個設計版本的總面積如下：



版本2:

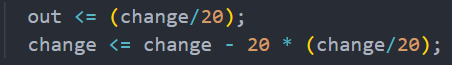
這次我把運算過程用到的好幾個state都省略掉，只留下IDLE和OUT，而計算就都在OUT裡進行，所以latency就能降為0，因為馬上算，馬上輸出，就不會有那麼多的delay了!

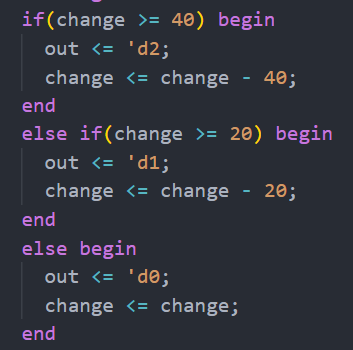
再者，這樣即算即輸出的設計也省了很多面積，因為就不用另外的register去存每一個零錢找多少 (雖然這樣的設計會用到除法器，而除法器相較占空間，這個問題將再版本3再進行解決)，但整體而言面積有被縮小。

版本3:

在此版本，latency一樣為0，主要是針對面積的部分進行改良，而改良就是拿掉幾個先前提到的除法器。如下圖，就舉找$20為例，版本2的零錢數目是用除法器去算要找多少個$20，但前面已經找完了$50，也就是說剩下最多$49，能給的$20最多兩個，最少零個，共三總可能，在版本3用if-else去實現就不用用到除法器了，省下了許多面積!

 版本2

版本3

