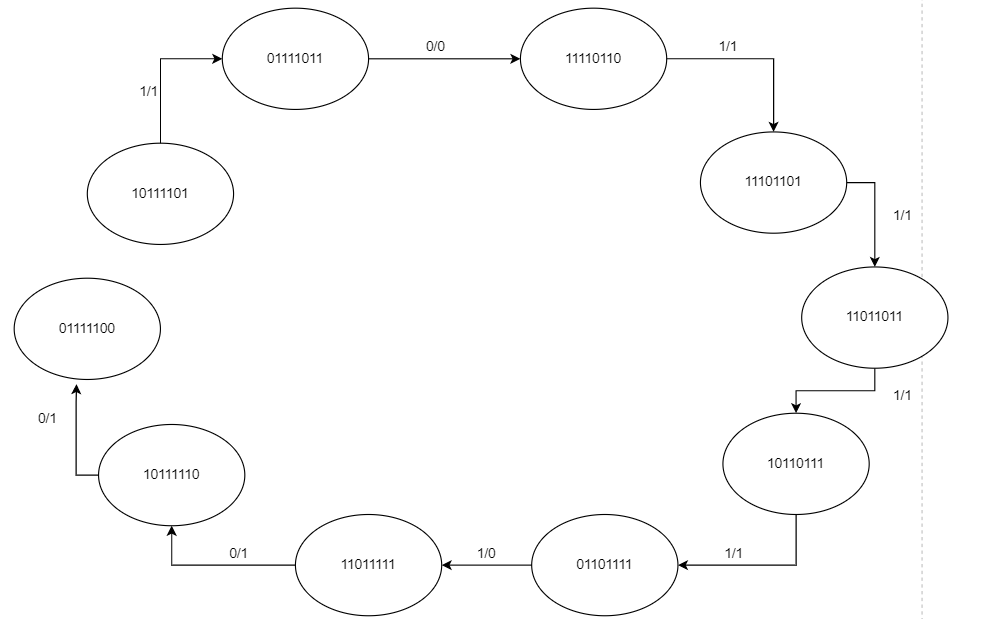
Lab 4 Team 20 report

109062222 徐嘉徽

109062119 李佳栩

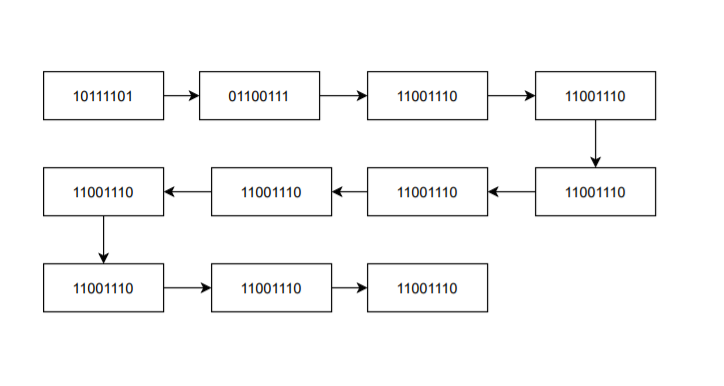
Basic 3:

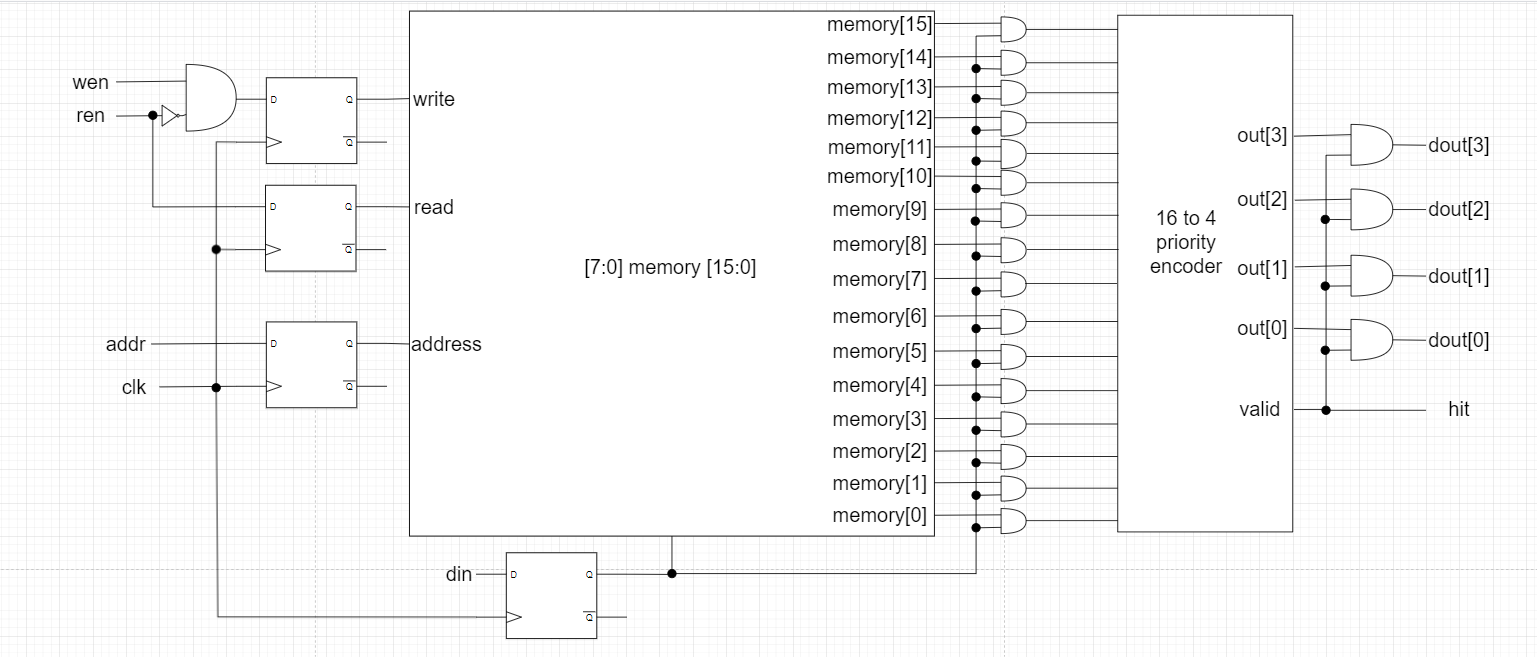


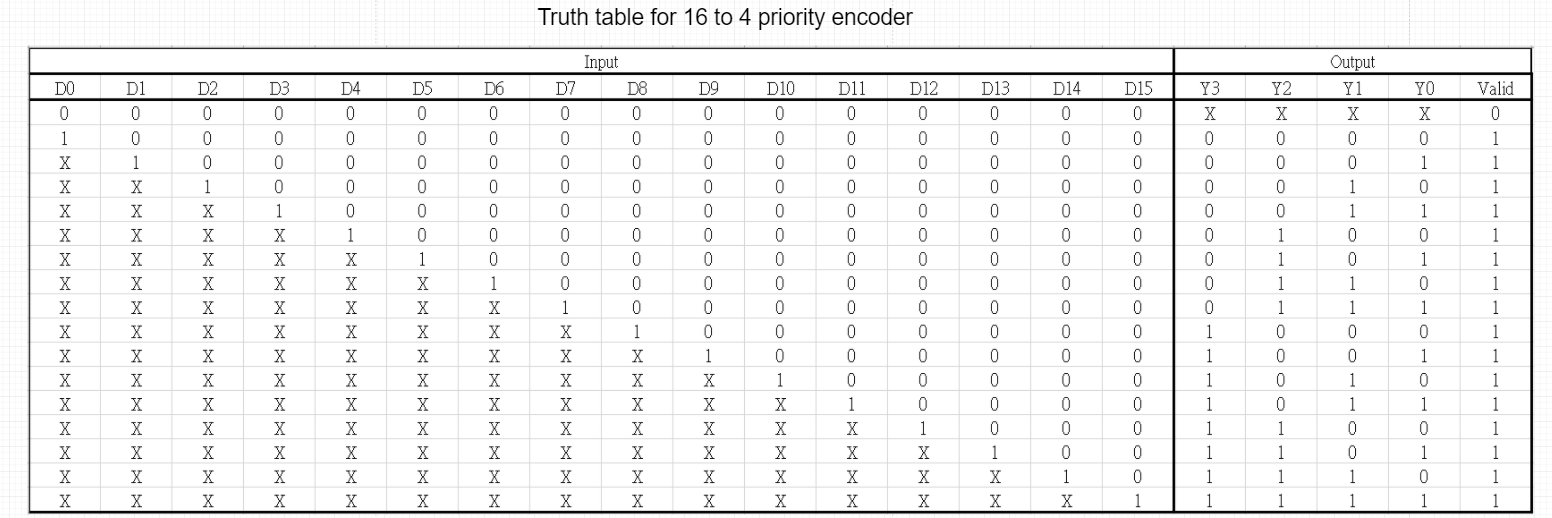
如果將DFF reset到8’b0，會導致之後的每個DFF[7:0]值都是8’b0，out也永遠是1’b0。

Basic\_4:

如果將DFF reset到8’b0，會導致之後的每個DFF[7:0]值都是8’b0，out也永遠是1’b0。



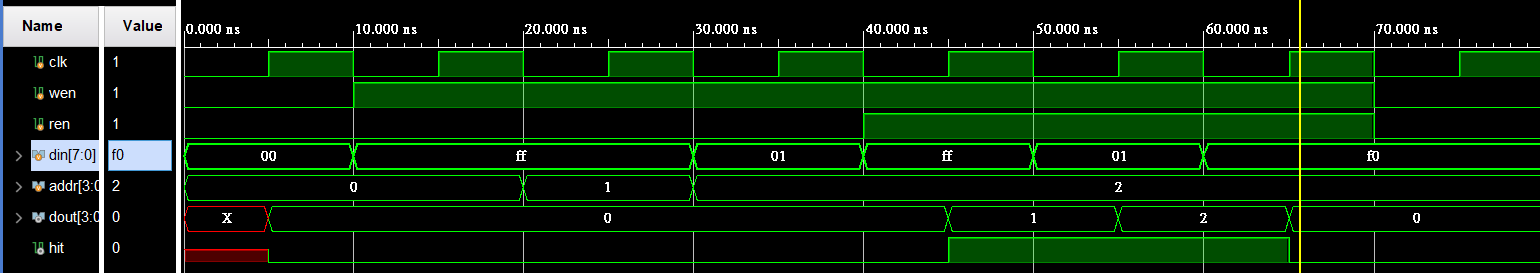
Advanced 1:



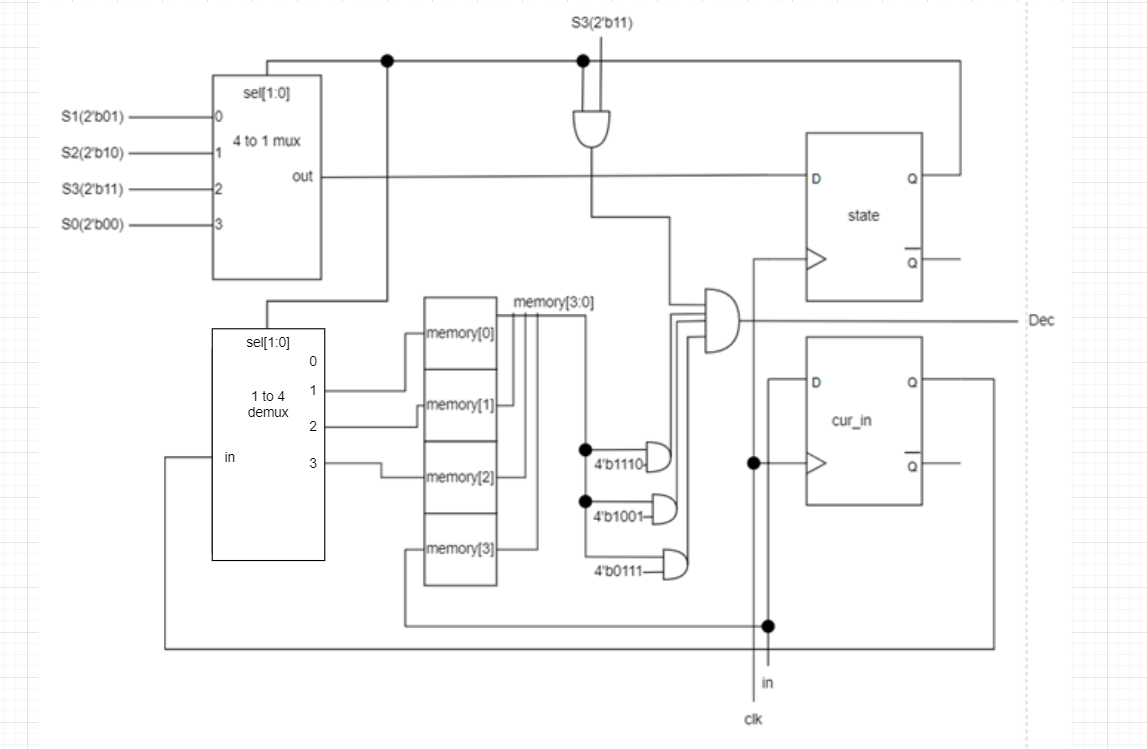
首先，我們利用lab 3 basic question 2中的Memory Array作一點改變，每個word設定為8個bit( [7:0] )，且總共有16個word( [15:0] )。在這次lab中，我們對Memory Array作了以下改變 : 當ren==1時，memory會同時讀出16個位置中的每個word，而當ren==0 && wen==1時，會將din的值存入到addr在memory的相應位置且不會有任何word被memory輸出。所以當ren==0 && wen==1時，output跟hit不重要，而當ren==1時，我們再將memory所輸出的16個word[7:0]分別與din[7:0] and起來，也就是比較每個word是否與din的值相同。比較完後會有三種情況 : (1)所有word都跟din不同，hit=0 (2)有一個word與din相同，輸出位置且hit=1 (3)有超過一個word與din相同，只輸出memory位置最高的，hit=1。為了完成(1)(2)(3)，我們利用16 to 4 priority encoder，這樣便能只輸出最高的memory位置，而valid的值表示有沒有word跟din一樣，也就是hit代表的意思。但16 to 4 priority encoder較難利用truth table實做出來，附上他的truth table顯示它的功用。

Testbench的部分，我們將10ns設為一個clk cycle，並且測試上述的(1)(2)(3)種情況。首先讓ren=0,wen=1,din= 8’b11111111 (ff) 且addr=4’b0000，並過一個cycle後讓addr=4’b0001，這樣便讓8’b11111111存到memory[0]和memory[1]中。再一個cycle後讓din=8’b00000001 (01)且addr=4’b0010，讓8’b00000001存入memory[2]。

下一個cycle將ren=1,wen=1，並每隔一個cycle讓din=8’b11111111(ff)、8’b00000001(01)、8’b11110000(f0)，以分別測試第(3)，第(2)，第(1)種情況，跑出來的waveform如下 :

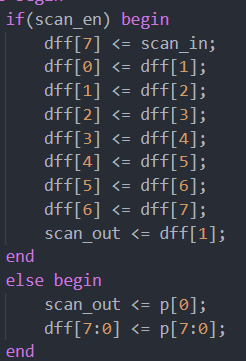


每種情況所對應的dout跟hit皆符合，代表此設計符合CAM的功能。

 Advanced 4:

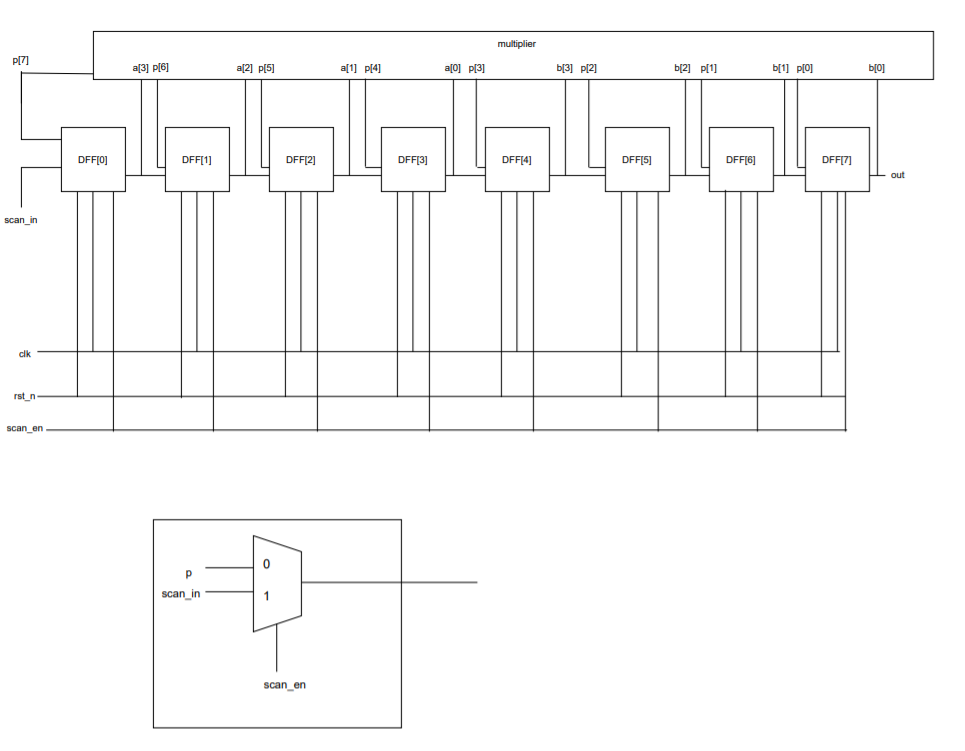
Advance\_2:

這題是要我們做出一個scan-chain，總共有三個階段，分別是scan\_in, capture, scan\_out，一開始我寫的時候有分成三個if-else，後來有同學在討論區上面問說in跟out重疊的情況，後來仔細想想才發現是可以重疊的，所以寫成，當scan\_en等於0的時候，我的dff再去抓p的值，也就是a\*b。



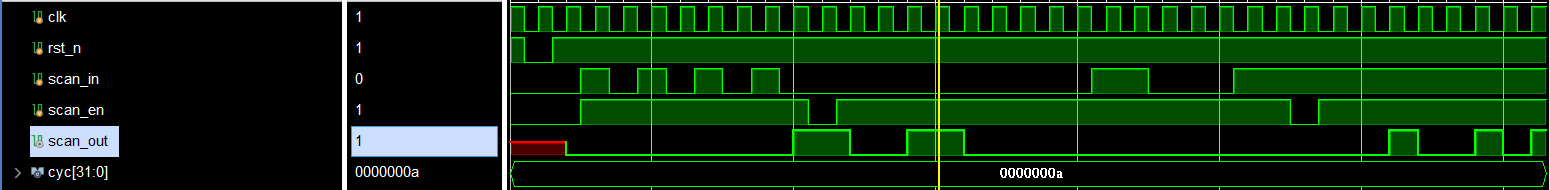
另外我的combinational circuits則是負責寫乘法器的部分，我原本寫在sequential circuits裡面，但沒想到的是sequential circuits裡面是用舊的值，導致我的答案會錯，後來拉出來寫就可以準確抓到a跟b的值。

下圖是整個scan\_chain的示意圖。



測試方法：

餵給scan\_chain不同的值觀察output，不過因為每次都是檢查一個bit太慢了，而且scan\_in的順序是反過來的，在檢查的時候也很麻煩，所以我在testbench有把a,b,p給設成output，以方便觀察。我用的數字是5\*5和6\*12，結果都有滿足我的答案。

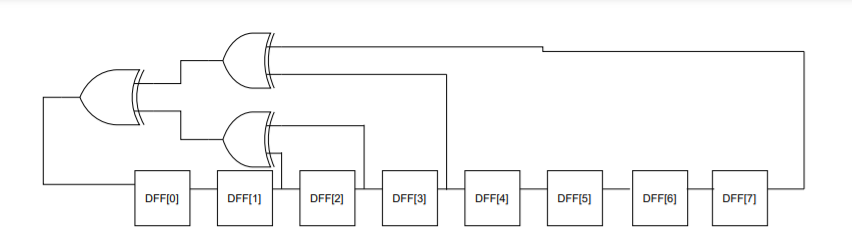


Advance\_3:

這題是利用basic和advance的題目做成的題目，認識了BIST，主要作法是把LFSR產生的值當成scan\_in傳給Scan\_Chain，唯一要做的就是把LFSR改成每次只傳MSB就好了。

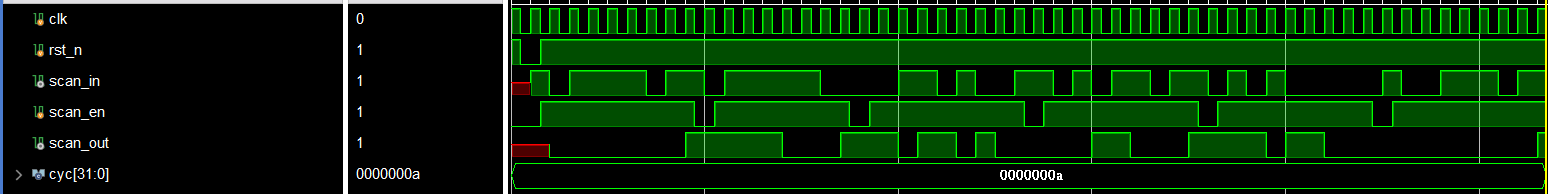
下圖是LFSR的示意圖。

DFF[7]是我的output。

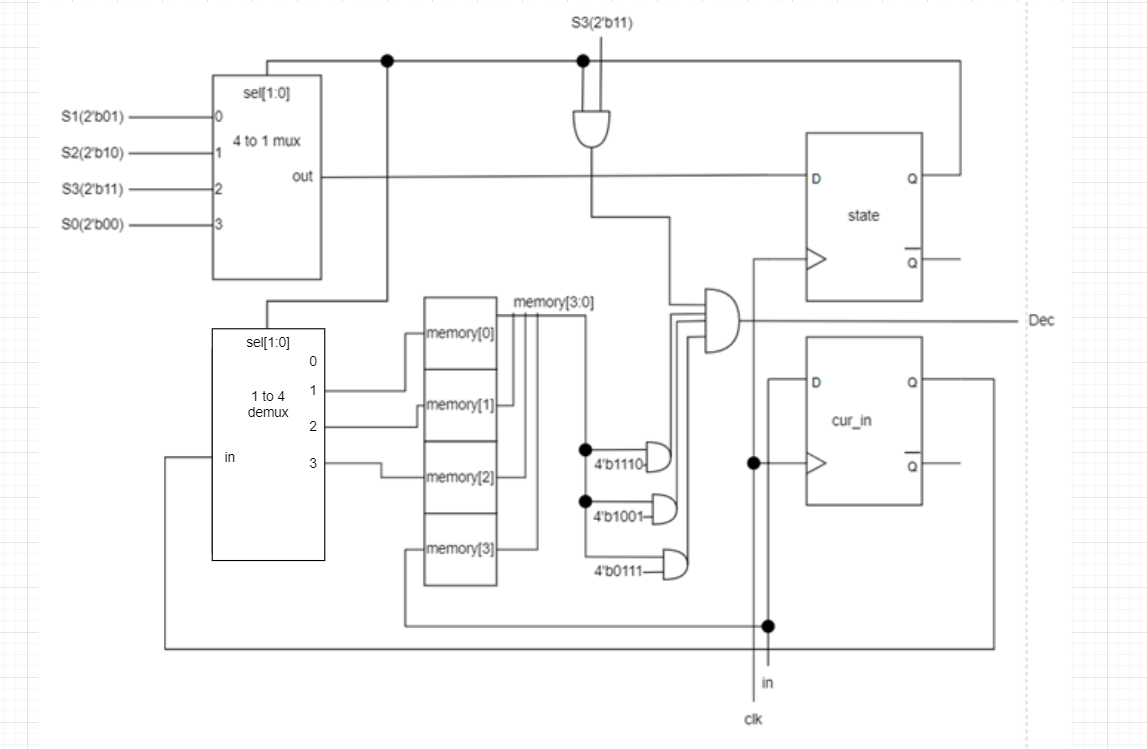


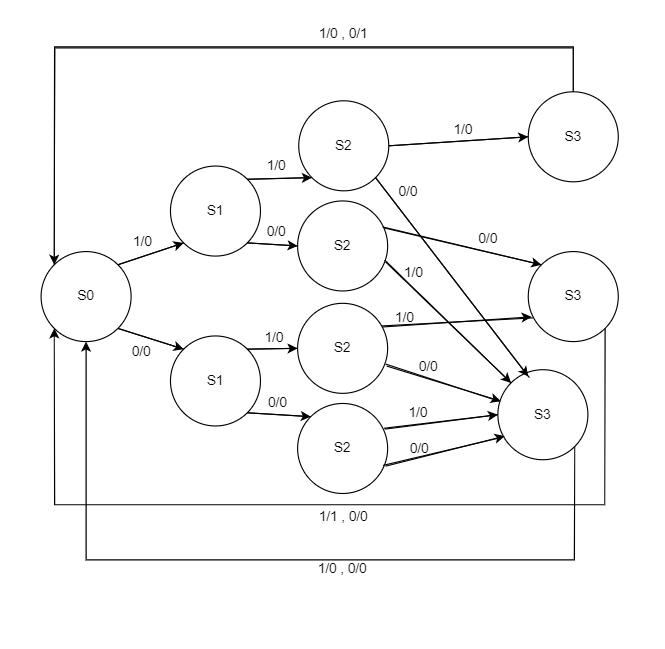
測試方法：

控制Scan\_in, capture, Scan\_out的階段，觀察output有沒有等於a\*b。一開始先reset，然後開始丟數字進去，那我是每scan到8個數字就scan\_out，這樣也可以順便測試overlap的狀況。



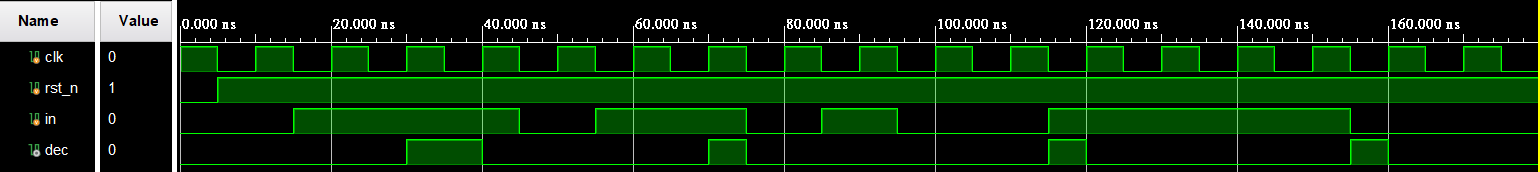
Advance\_4:





這題利用四種state : S0、S1、S2、S3的mealy machine來完成sequence detector。首先利用兩個DFF代表state與cur\_in。在state的input(也就是next\_state)利用4 to 1 mux來完成state轉換，並在sel接上state的output，當sel==S0，next\_state=S1、sel==S1，next\_state=S2、sel==S2，next\_state=S3、sel==S3，next\_state=S0。在每一個cycle，都會將in記錄成cur\_in，並利用一個memory[3:0]來記錄sequence的每一個bit，我們利用1 to 4 demux來完成這件事，選擇方法如圖，memory[3]則是直接將接上in，不受到clk的影響。由於我們只有在S3時會記錄完4個bit並決定sequence是否符合4’b1110、4’b1001、4’b0111，因此我們將memory[3:0]分別跟4’b1110、4’b1001、4’b0111 判斷是否一樣，並再將結果or起來，最後再判斷state是否為S3，便能決定Dec。

Testbench的部分我們，我們將10ns設為一個clk cycle。並測試過每個會讓dec=1的sequence，也就是4’b1110、4’b1001、4’b0111，同時也測試了幾個隨機的input sequence，以觀察dec是否只會在S3時變成1。測試結果如下:



分工: 徐嘉徽寫1、4，李佳栩寫2、3 ，report的分工也一樣

心得:

徐嘉徽：這次要利用state transition來設計出題目要求的功能及電路，雖然code變的簡短許多，但要思考的事情也變得更多，尤其是要注意每個clk cycle時的state是否正確，以及next\_state是否符合要求。最重要的是這次的advanced有使用CAD來測試，上次沒有使用CAD測試，導致最後改出來的結果跟預期相差很大，我們之後也會多加利用CAD測試，才不會再出現意料之外的錯誤。

李佳栩：這次我負責的是2.3題，雖然不是FSM的題目，不過也讓我在寫code的時候發現自己容易粗心的毛病，這次的內容也比較簡單，根據圖片接起來就可以了，不過也是有些小問題要注意，像是我一直都很不熟悉sequential circuits 跟combinaitonal circuits分開的寫法。還有經過前一次lab的教訓，這次都有使用CAD測試，希望這次的作業能夠達到我們給自己的目標。