**Lab 4 Finite State Machine**

**實驗報告**

組長:劉奇泓 109033135

組員:洪聖祥109062315

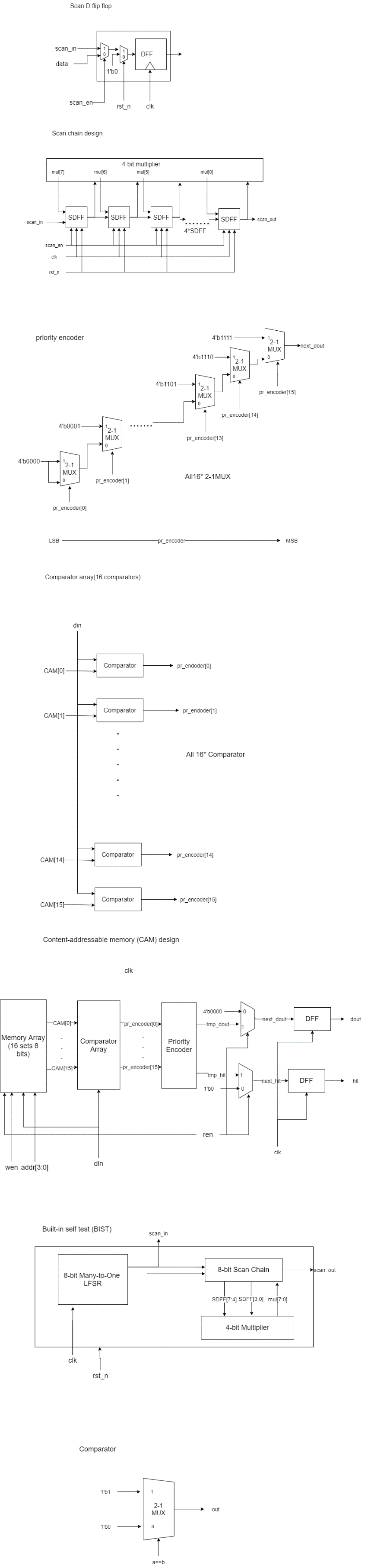
1. **Content-addressable memory (CAM)**

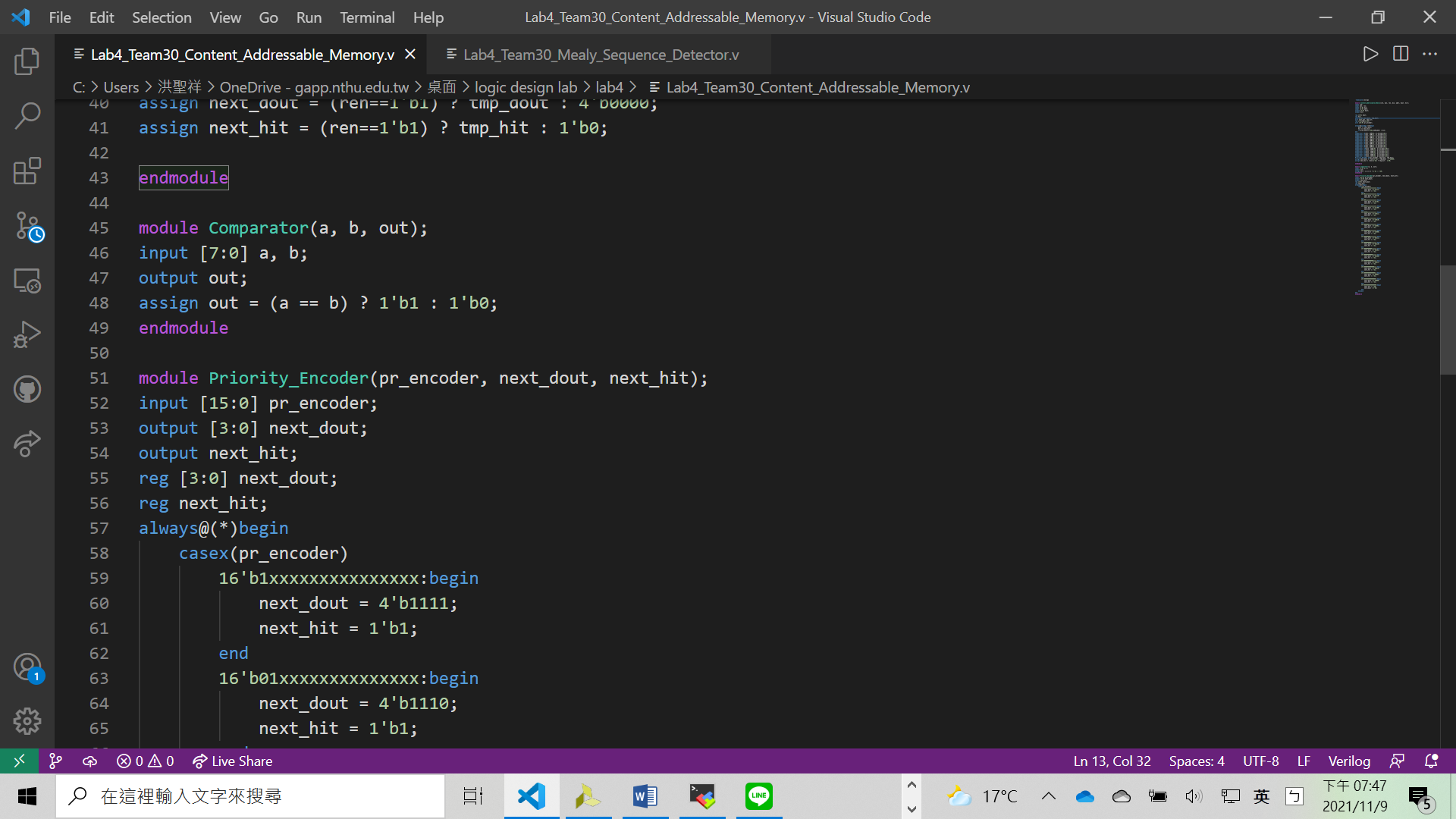
CAM 有5個input 2個output，它的功能是當ren = 1’b1時會讀取memory array的值並跟din比較是否相同。如果有多個address符合條件則輸出dout = 最大的address hit = 1’b1。當wen = 1’b1, ren = 1’b0則把din寫進memory array 中的addr位址。

CAM可以分成2個sequential circuit和3個combinational circuit。2個sequential circuit分別儲存並在positive edge更新兩個output dout和hit的值。3個combinational circuit分別為1個8-bit memory array、1個comparator array和1個priority encoder。

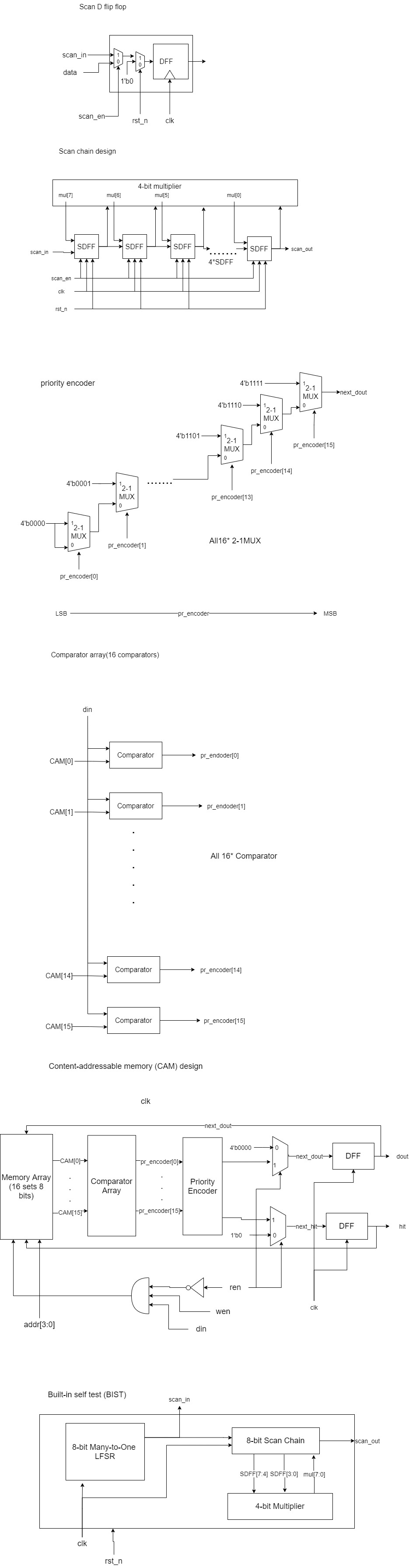
Comparator array

Block diagram:



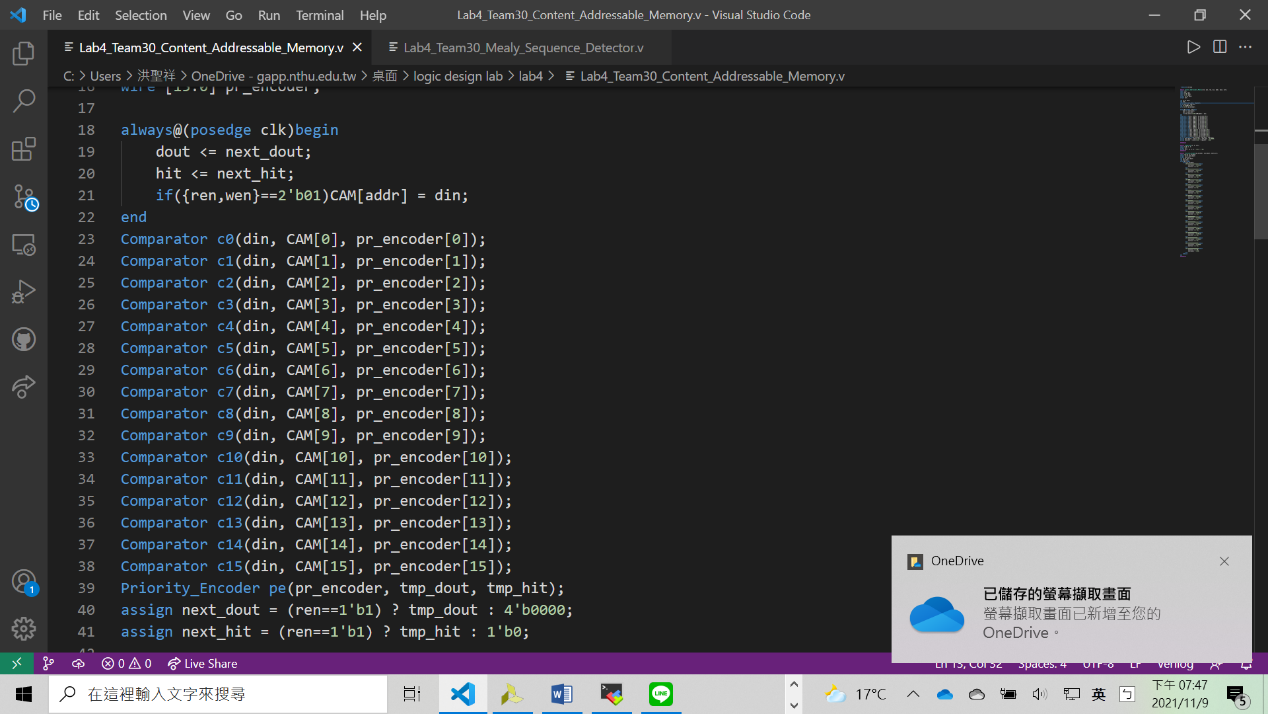
 Code

Comparator 用簡單的條件a == b來決定out 是1’b1還是1’b0。



Comparator array 總共有16個comparators。第一個comparator CAM[0]跟din比較。如果兩個相等，則輸出1’b1否則輸出0 並把輸出值指定給pr\_encoder[0]。第二個comparator則是CAM[1]跟din比較並把輸出值指定給pr\_encoder[1]。第i個comparator CAM[i-1]跟din比較並把輸出值指定給pr\_encoder[i-1]。

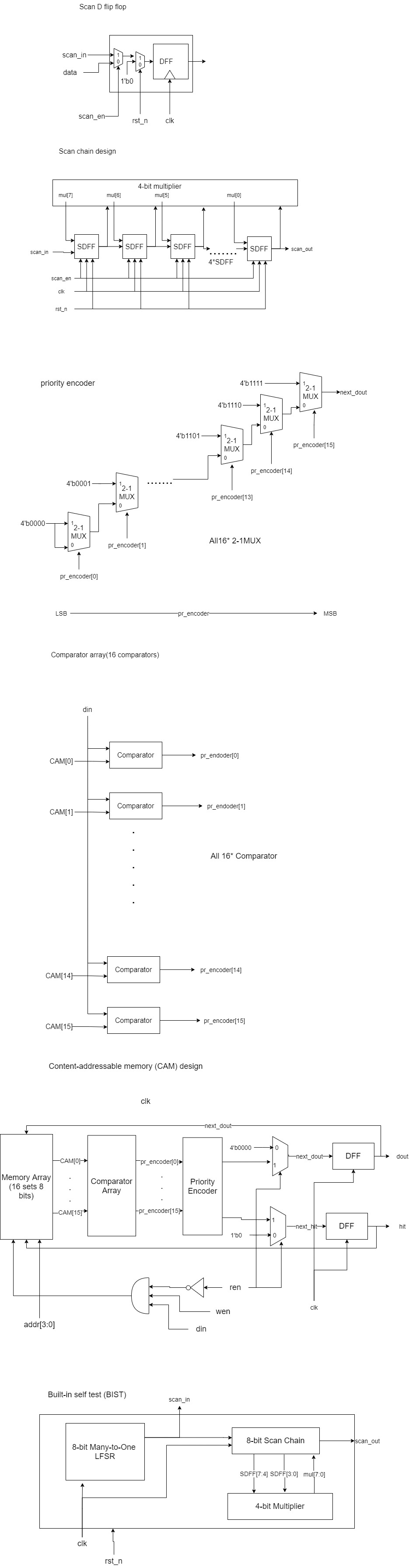
Code



instance 16個comparator以din CAM為input pr\_encoder為output 來實作Comparator Array

Priority encoder

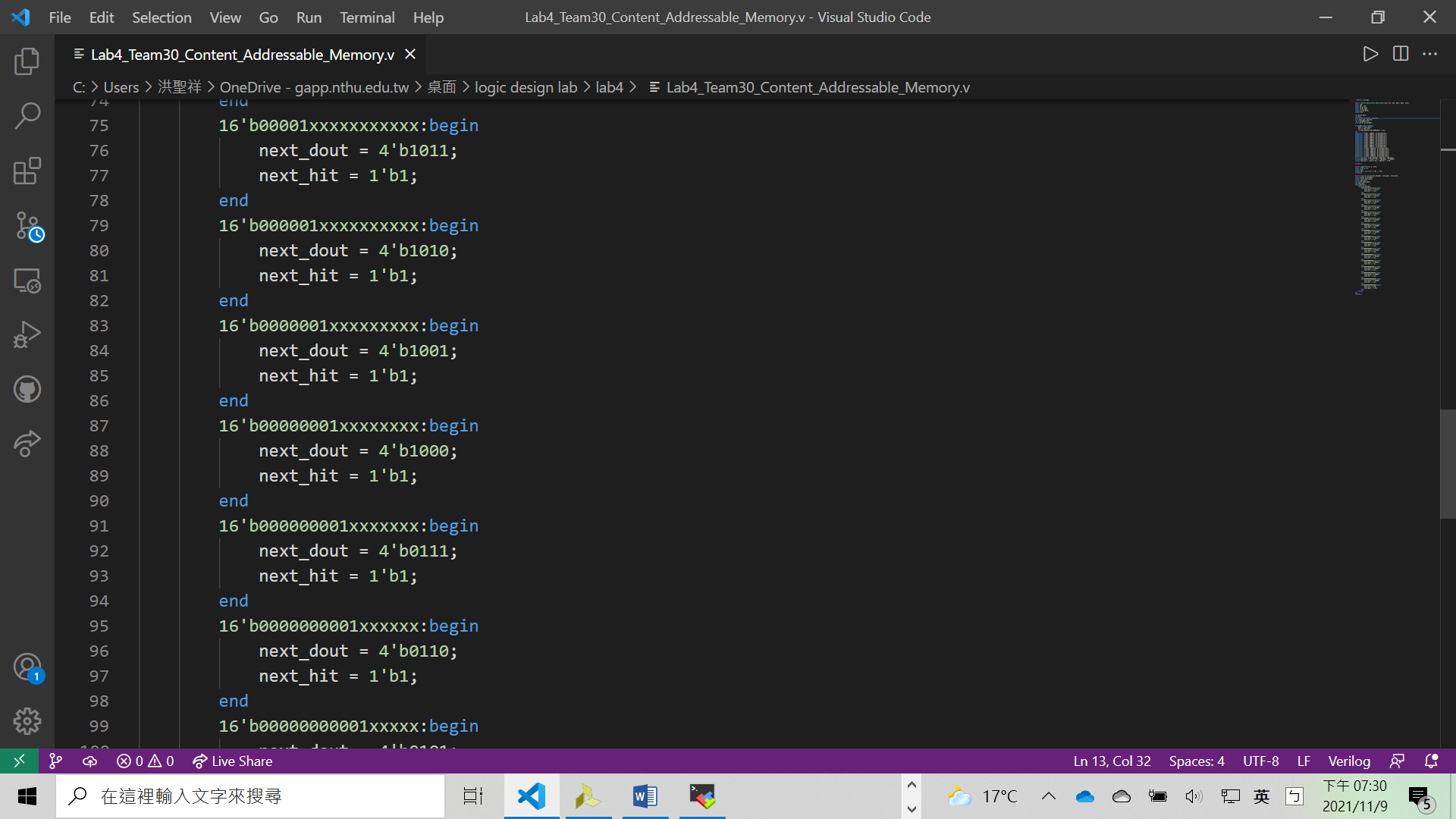
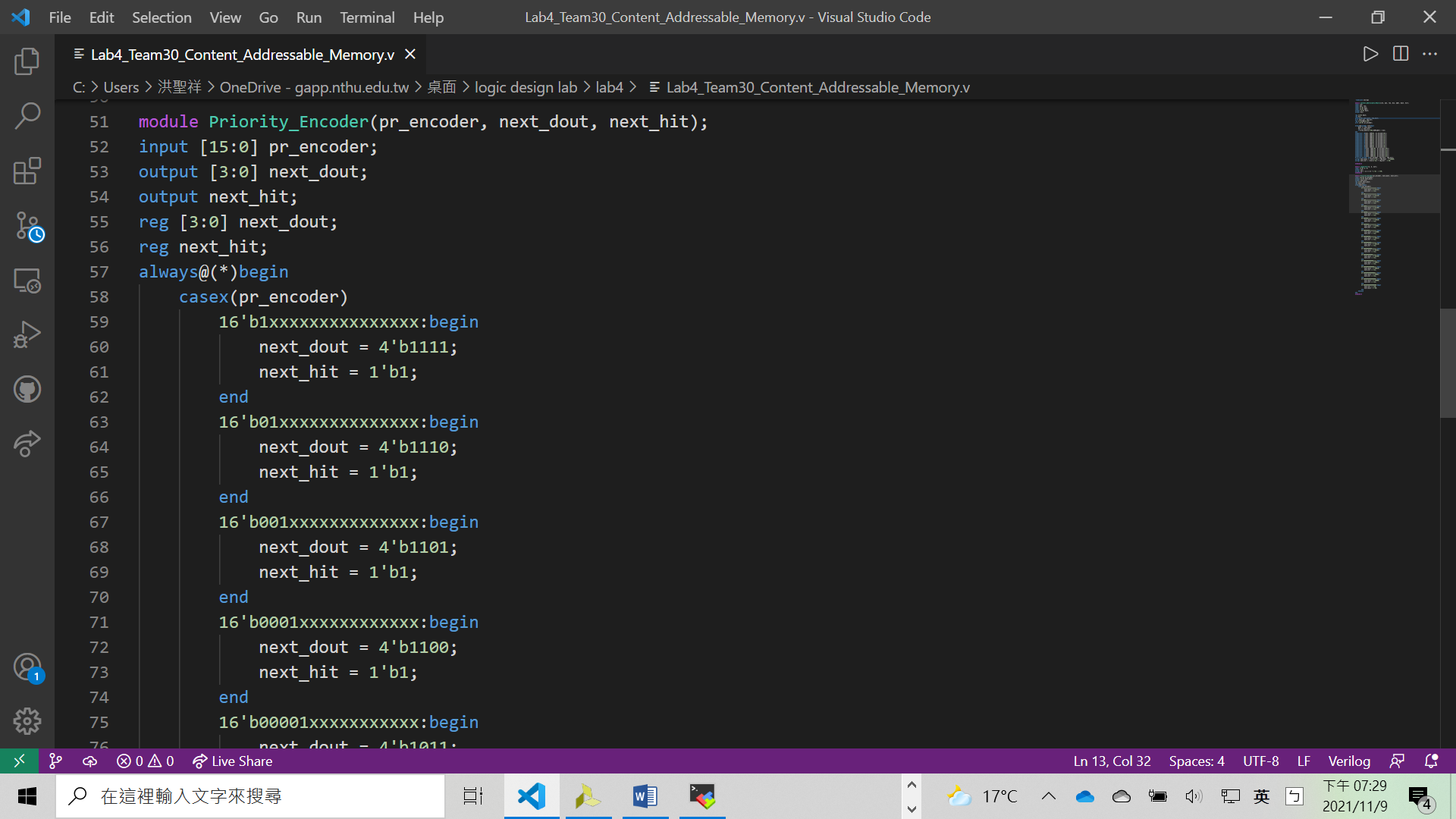
Block diagram:



2^n to n Priority Encoder的功能是當2^n bit input值的每一個bit中有2個以上的bit = 1’b1，可以決定一個優先順序來決定output。這時用普通的encoder會出現問題，因為普通的encoder限制只能有一個bit = 1’b0其他的bit則是1’b1。我16 to 4 Priority encoder的實作是用16個2-1 MUX串起來。

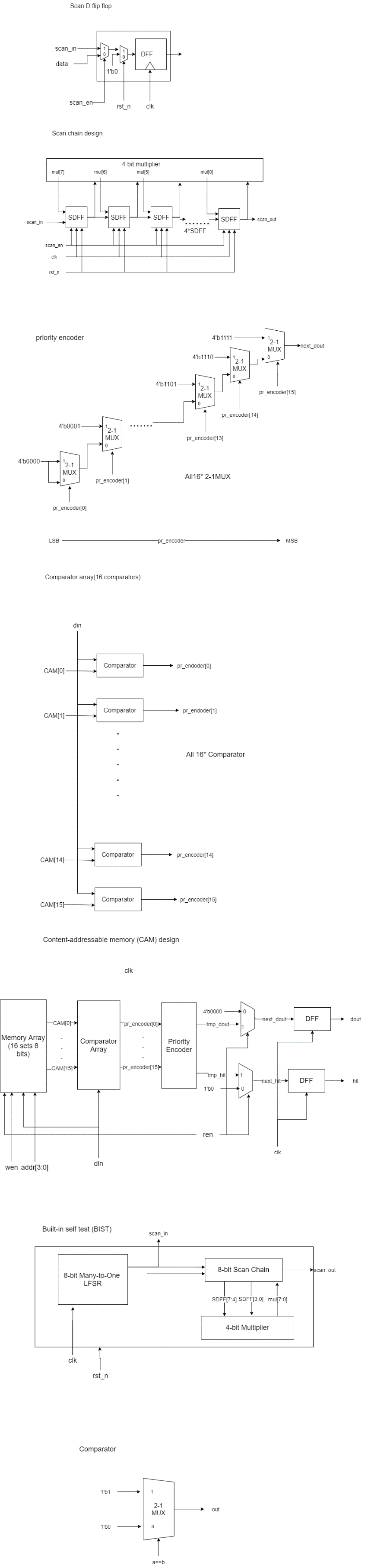
當16 bit input中有不只一個bit是1’b1，我們需要優先順序(這裡是significant bit)最高的輸入將會被優先輸出。

Code 片段:



用casex舉出16種可能，就可以知道next\_dout是什麼(ex:pr\_encoder = 16’b0100101010101011，第15個bit的優先度最高，因此next\_dout = 4’b1110) next\_hit代表pr\_encoder是否為16’d0，若是則為1’b0反之為1’b1。

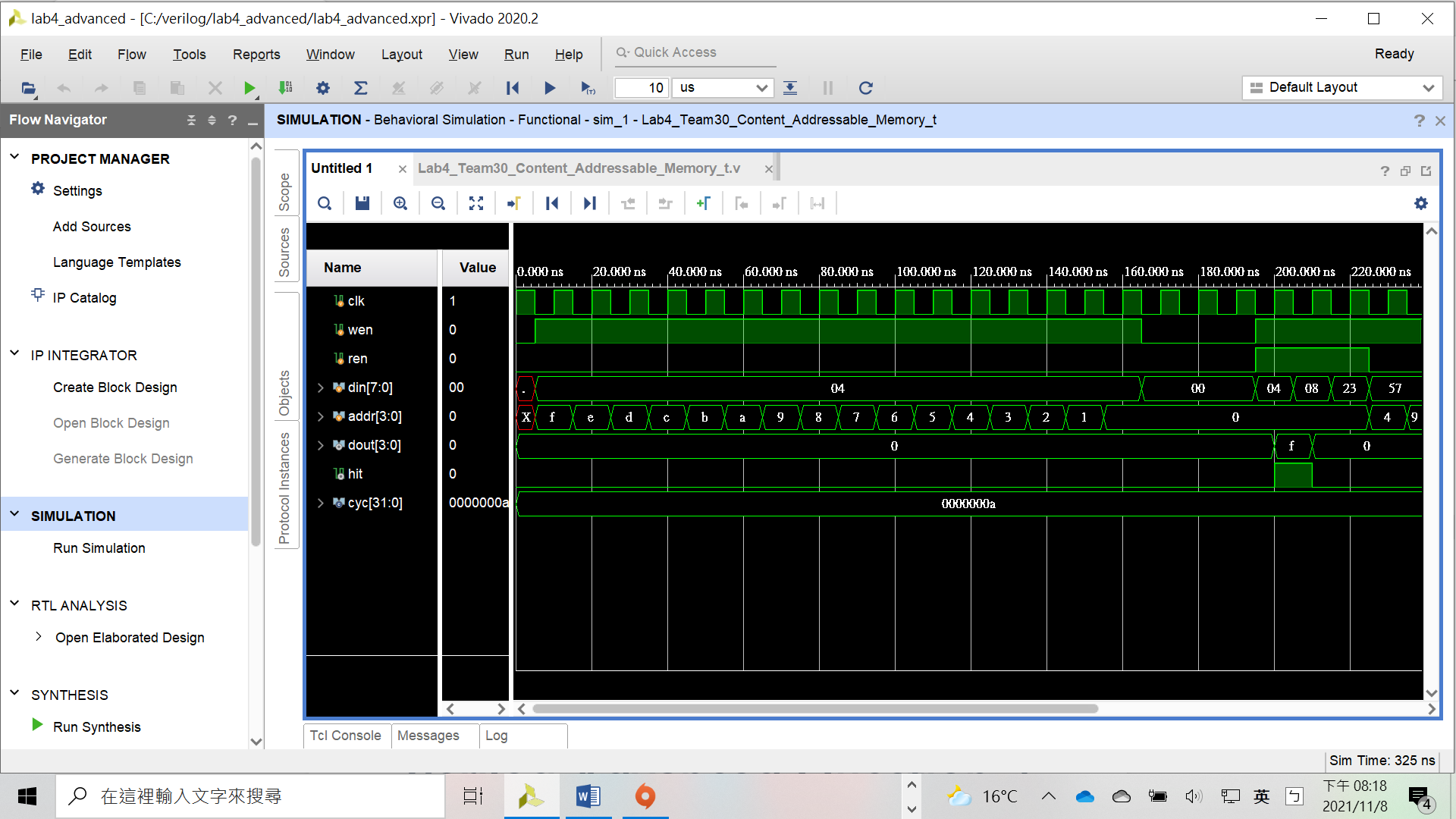
CAM:

Block Diagram:

CAM中的memory array 用reg [7:0]CAM[15:0]來模擬。當ren == 1’b1時，next\_dout = tmp\_dout，next\_hit = tmp\_hit。tmp\_dout、tmp\_hit則是從memory array取值再經過comparator array 和priority encoder。{ren,wen} = 2’b01時，把din存進CAM[addr]中。每一個posedge output hit、dout都會更新他的值dout <= next\_dout hit <= next\_hit。

波形圖:

下圖為確認當memory中的每一個address都是同一個值，當din是address中的值是否會輸出最大address而且hit = 1’b1，並且din是memory中部存在的值是否輸出1’b0而且hit = 1’b0。順帶檢查ren = 1’b1 wen = 1’b1時，會執行read的動作。

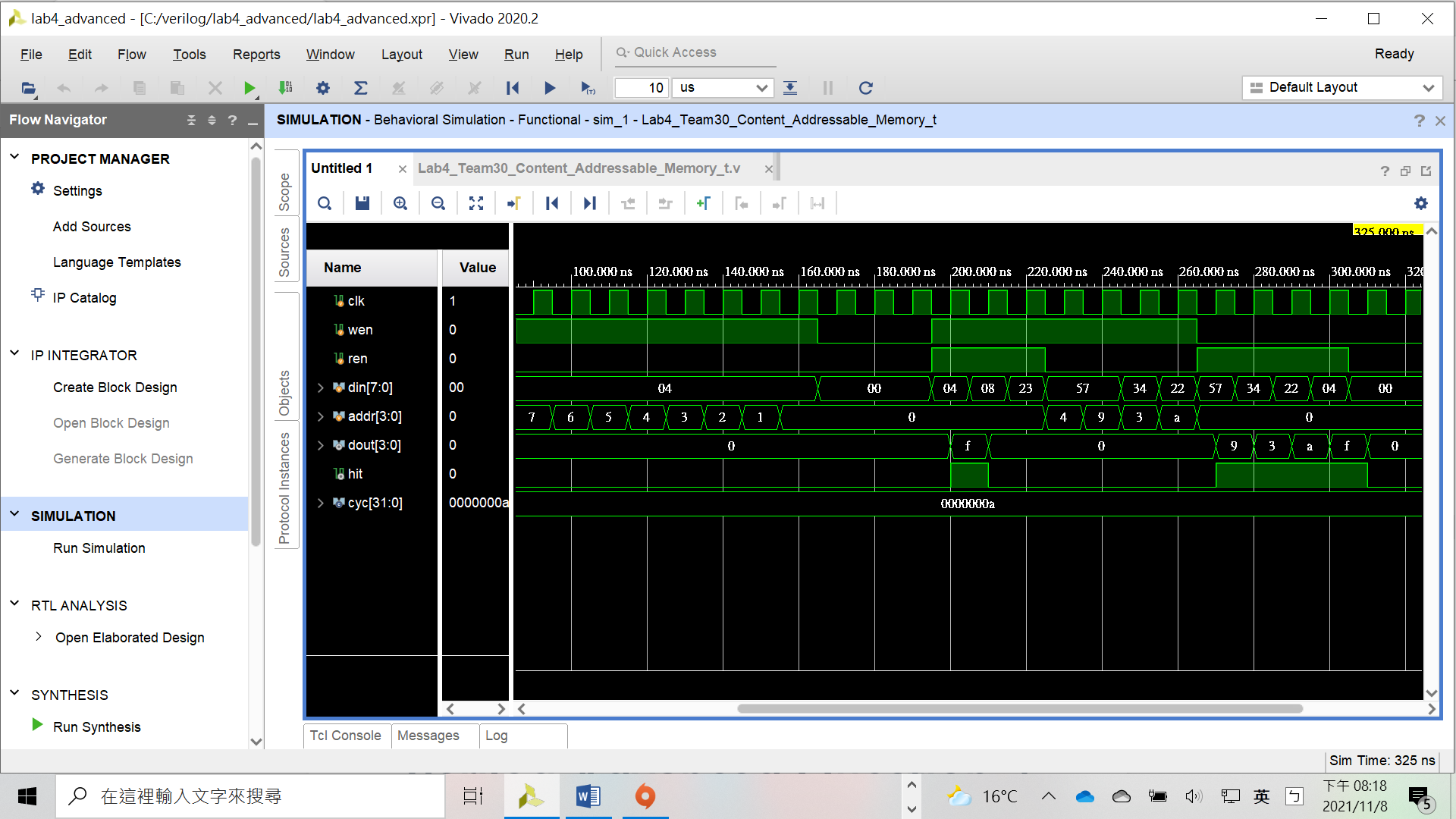


確認當ren,wen = 1’b1，din在memory中是那些address並輸出最大的address

確認當ren,wen = 1’b1，din在memory中是那些address如果沒有符合的address，dout 輸出4’b0000，hit = 1’b0

Memory中所有位址都是4’d4

下圖為多試同一個din不同的addr，在ren = 1’b1是否會輸出最大的address



Ren = 1’b1 wen = 1’b0，din = 4’d57在memory中有2個位址(4、9)，確認輸出最大的位址9。確認其他din在memory中的位址確實輸出到dout

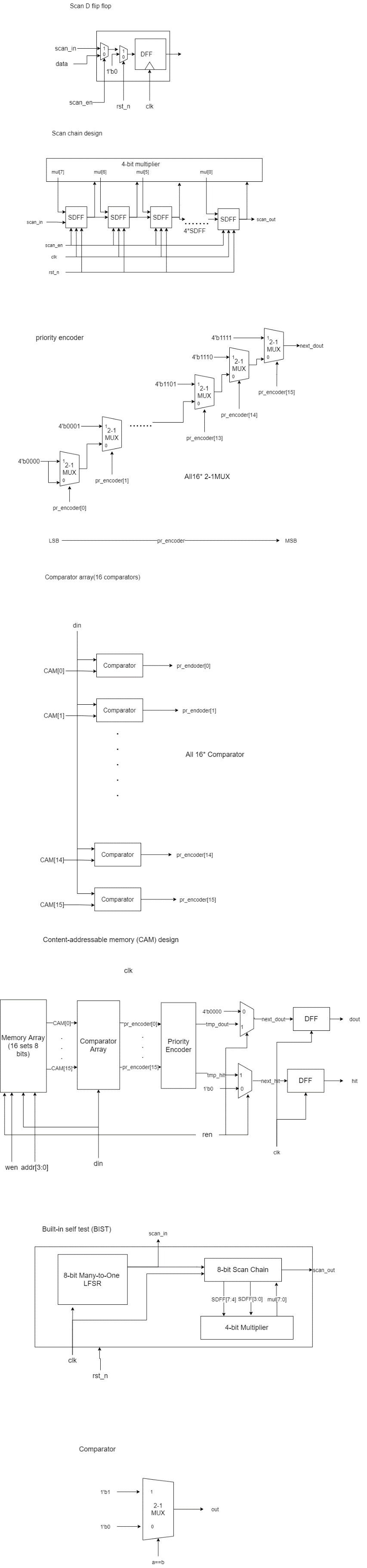
wen = 1’b1 ren = 1’b0，將din = 4’d57存進memory中的4和9的位址

**2.Scan Chain Design**

Scan Chain Design 由8個Scan D-Flip Flop和一個4bit multiplier 所構成。

Scan D-Flip Flop

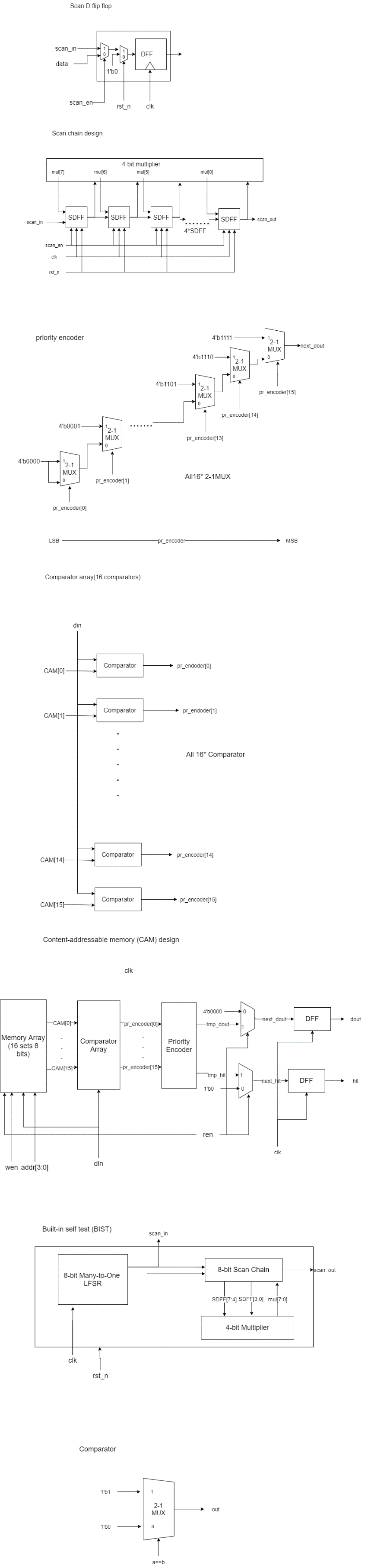
Block diagram:



Scan DFF由兩個input signal rst\_n和scan\_en來決定甚麼值為DFF的input。rst\_n = 1’b0，DFF的input為1’b0，rst\_n = 1’b1，scan\_en = 1’b1，DFF的input為scan\_in，scan\_en = 1’b0，DFF的input 為data。

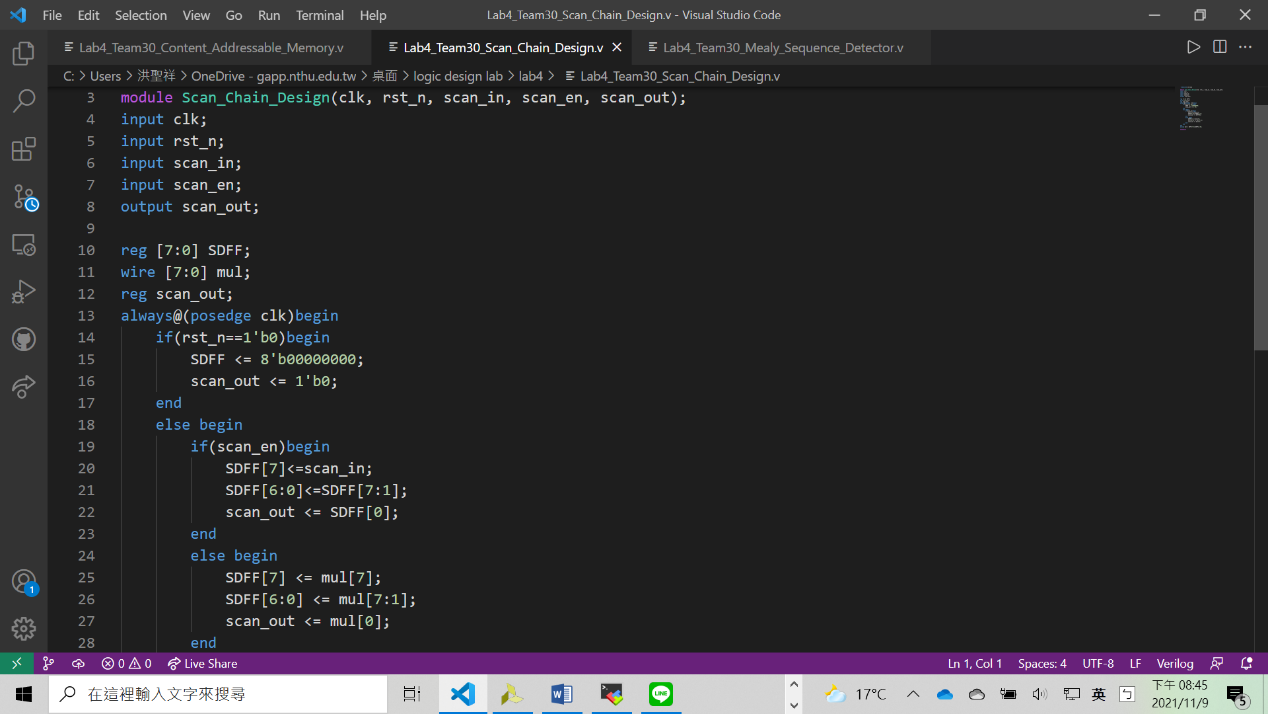
Scan Chain Design

Block diagram



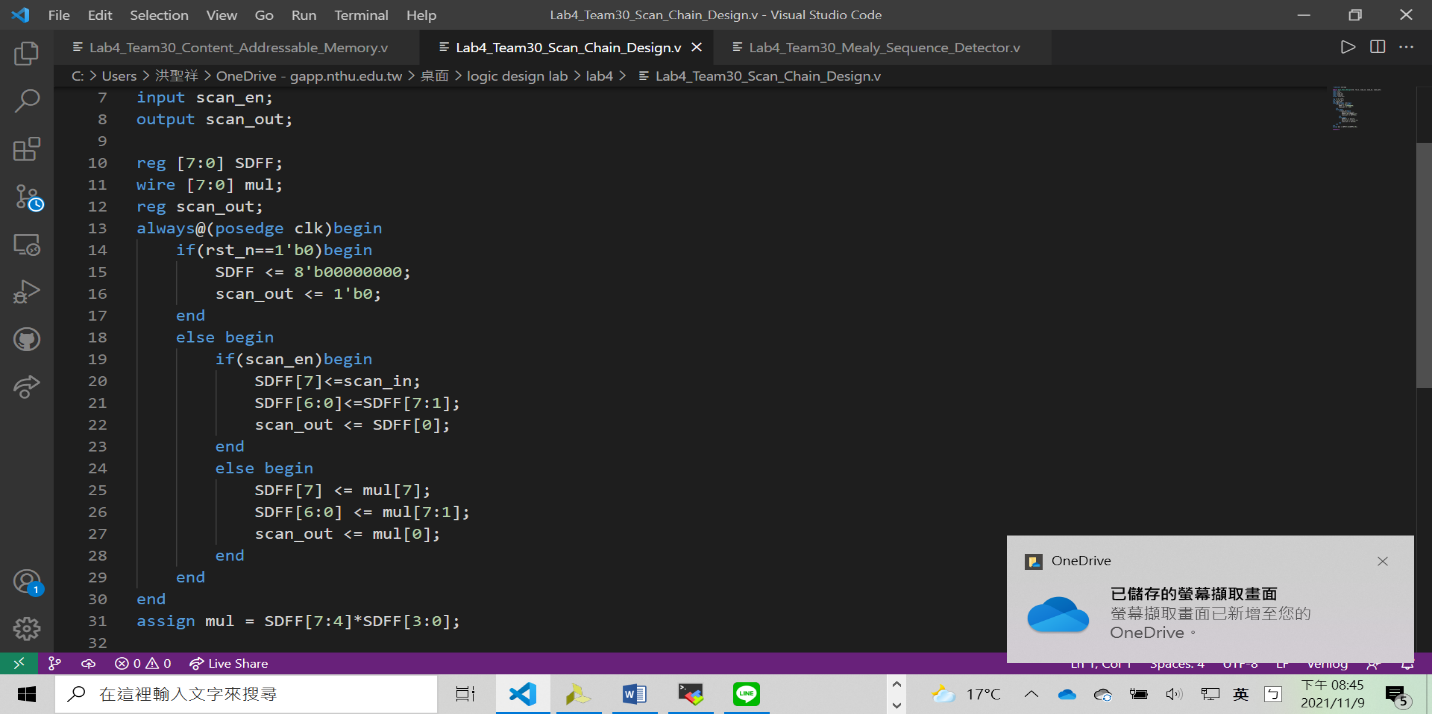
Scan Chain Design 是當rst\_n = 1’b1 positive edge、scan\_en = 1’b1時，scan\_in會1個bit 1個bit從左傳到右最後到output scan\_out。scan\_en = 1’b0時，4bit multiplier的每一個bit會傳入SDFF並1個bit 1個bit從左傳到右最後到output scan\_out。

Code:



scan\_in 1個bit 1個bit從左傳到右

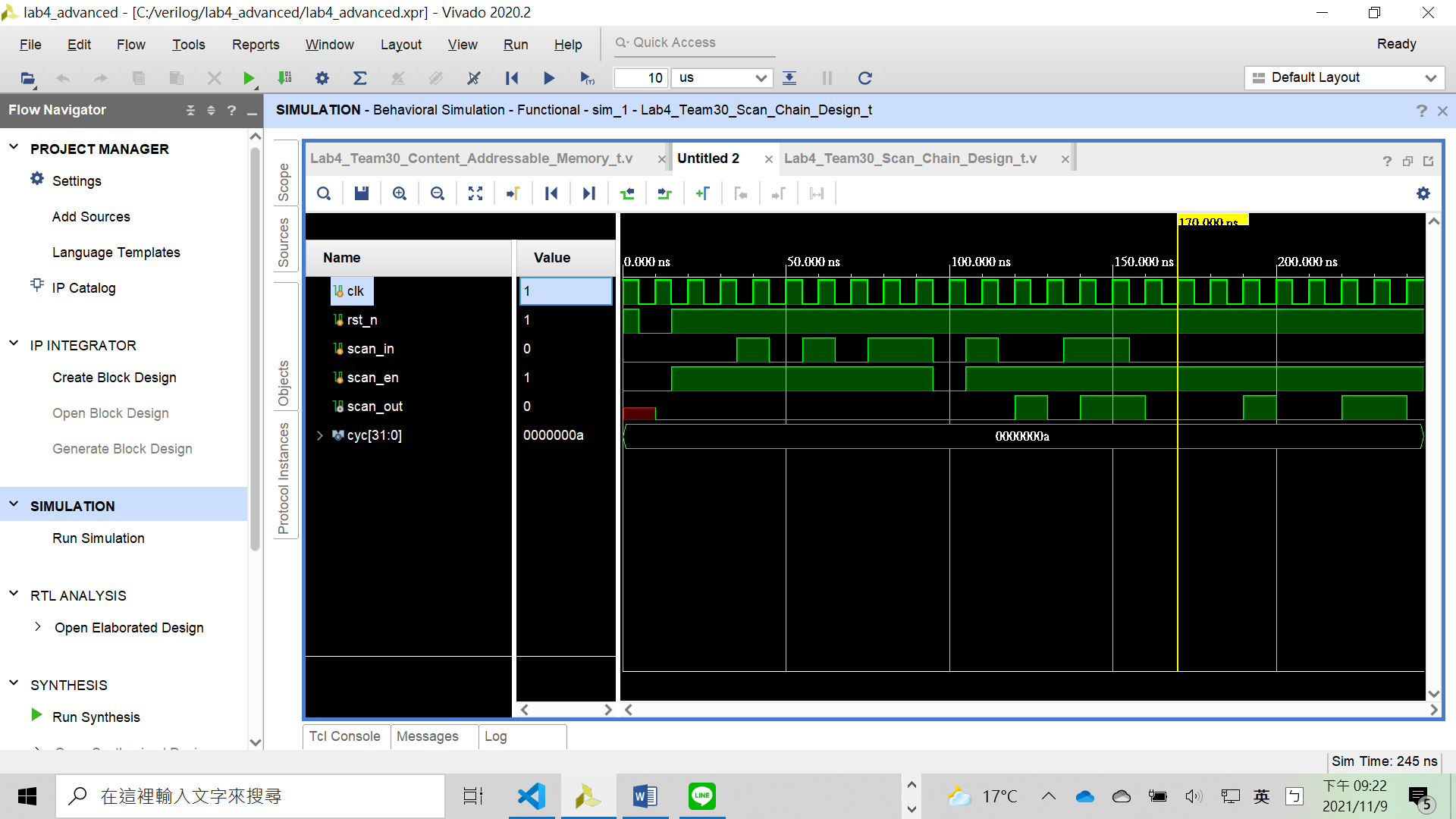
4 bit multiplier的output接到對應的SDFF



4 bit multiplier

波形圖:

下圖測試在rst\_n = 1’b0後rst\_n = 1’b1、scan\_en = 1’b1經過8個positive edge 後scan\_en = 1’b0再經過7個positive edge，scan\_out的值是否正確，並確認是否能overlap scan\_in



確認兩波形圖一樣(scan\_in scan\_out overlap)

4 bit multiplier的2個input為1101以及0100，因此output為00110100。因為least significant bit輸入最右邊的SDFF所以最早輸出的output scan\_out為least significant

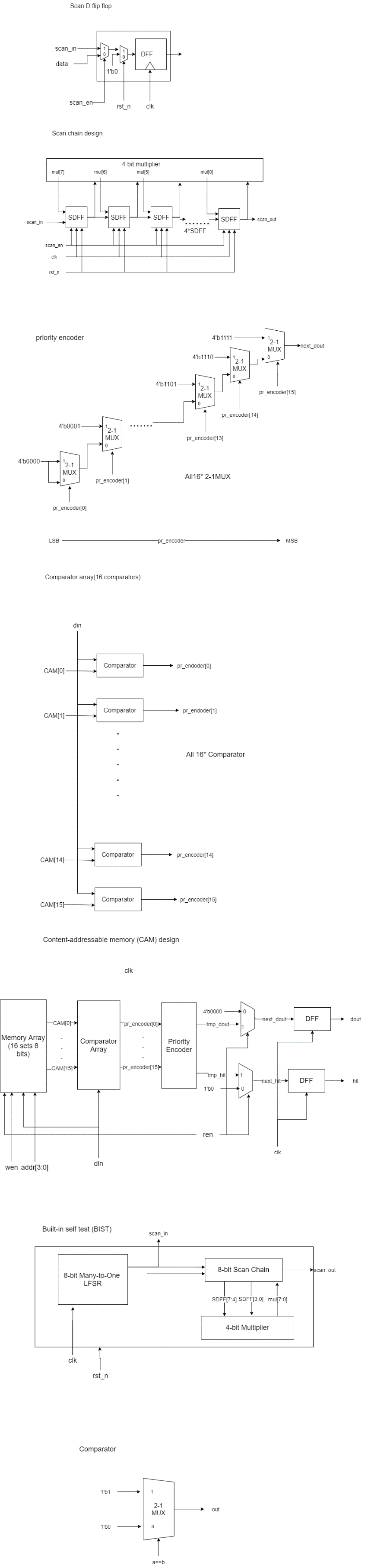
Bit，因此scan\_out依序為multiplier每個bit反過來00101100。

scan\_in依序為0,0,1,0,1,0,1,1

因為最早輸入的scan\_in最早傳到最右邊的SDFF，因此我們的2個4bit multiplier的input 為上述的值反過來，也就是11010100，因此2個input為1101以及0100。

1. **Built-in self test (BIST)**

Block diagram:

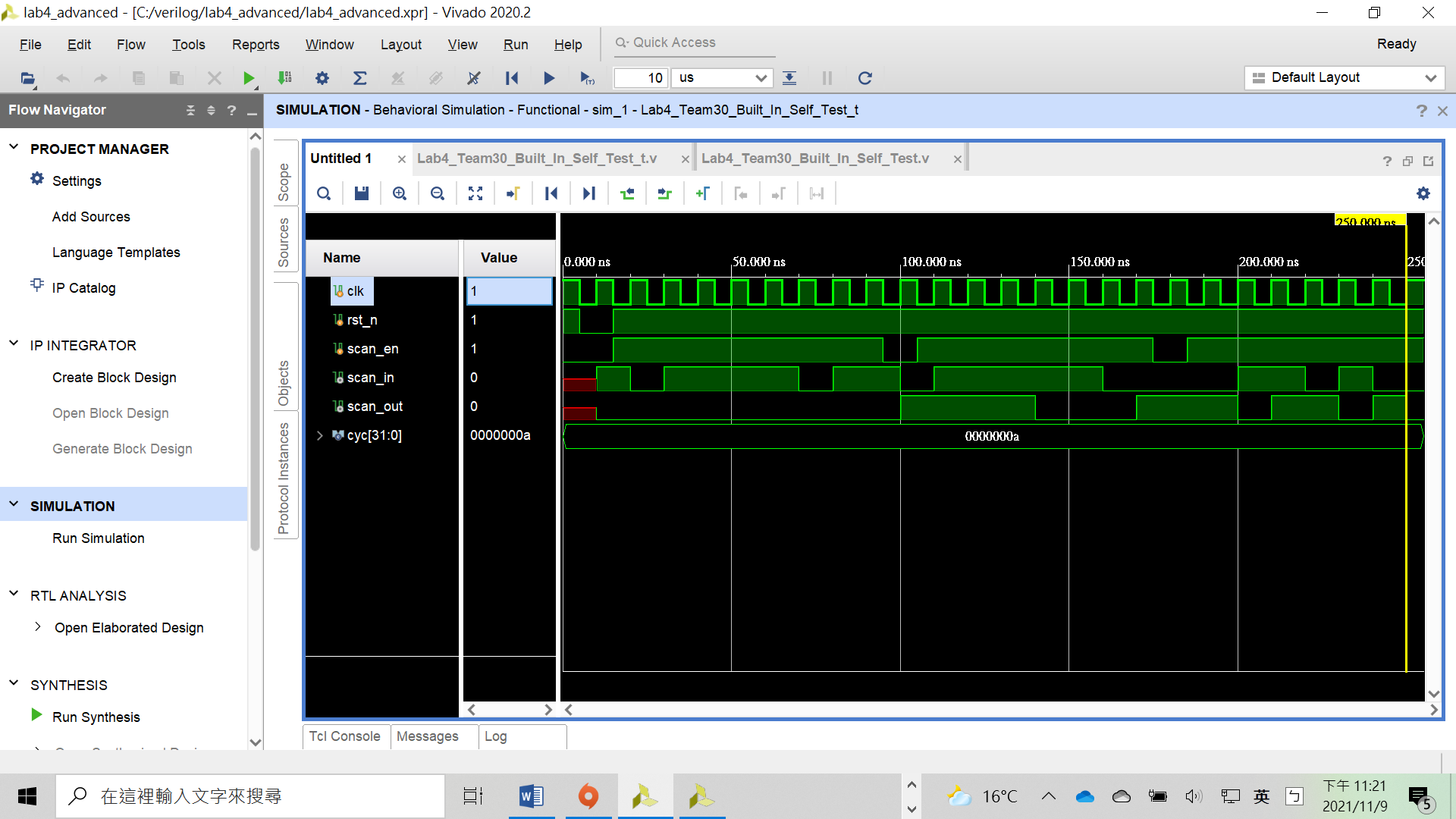


BIST由Many-to-One LSFR 、8 bit Scan Chain和4 bit multiplier所組成。只是將advanced question2 的scan chain design以Many-to-One LSFR output的most significant bit為scan chain的input。

波形圖:

下圖為確認Many-to-One LSFR是否正確，以及scan\_en = 1’b0後scan\_en 再次 = 1’b1後scan\_out的值是否為4 bit multiplier

輸出的值。也確認scan\_in scan\_out可以overlap。



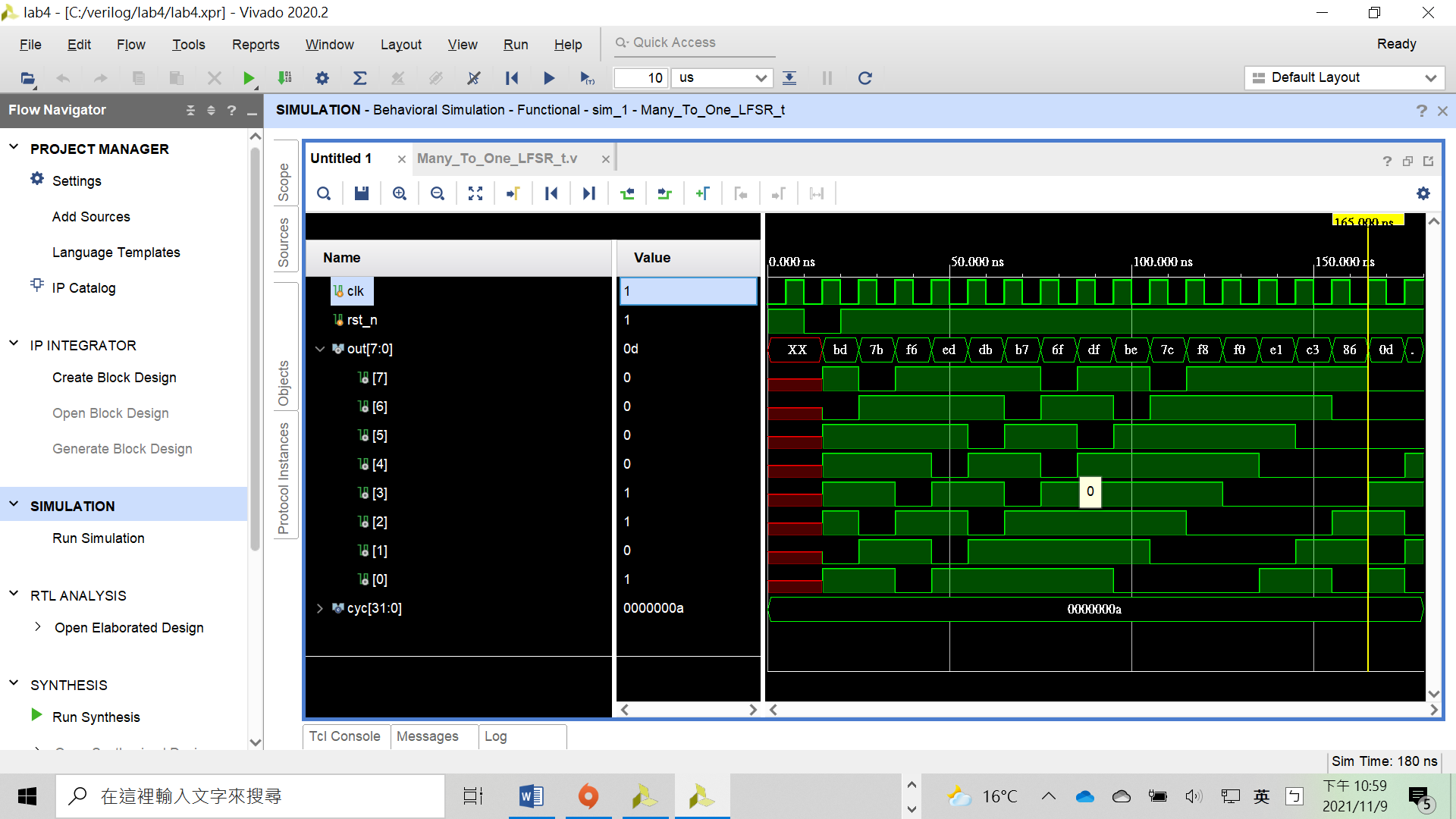
11\*13 = 143 =

8’b10001111

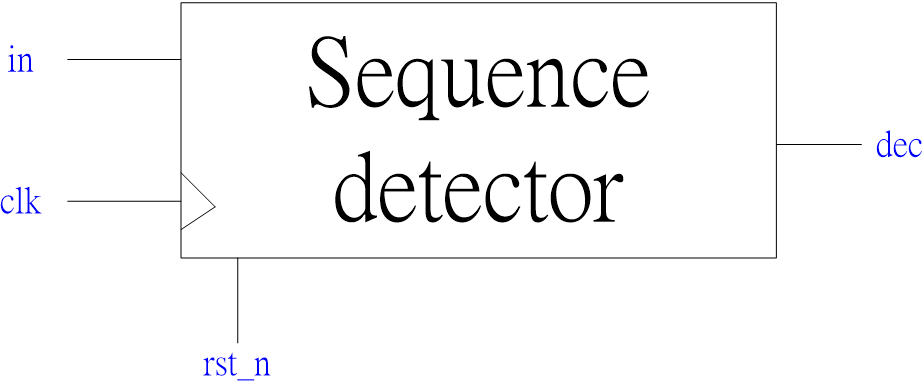
從lsb輸出，scan\_out依序為11110001

波形圖與M\_T\_O\_LFSR output的most significant 相符

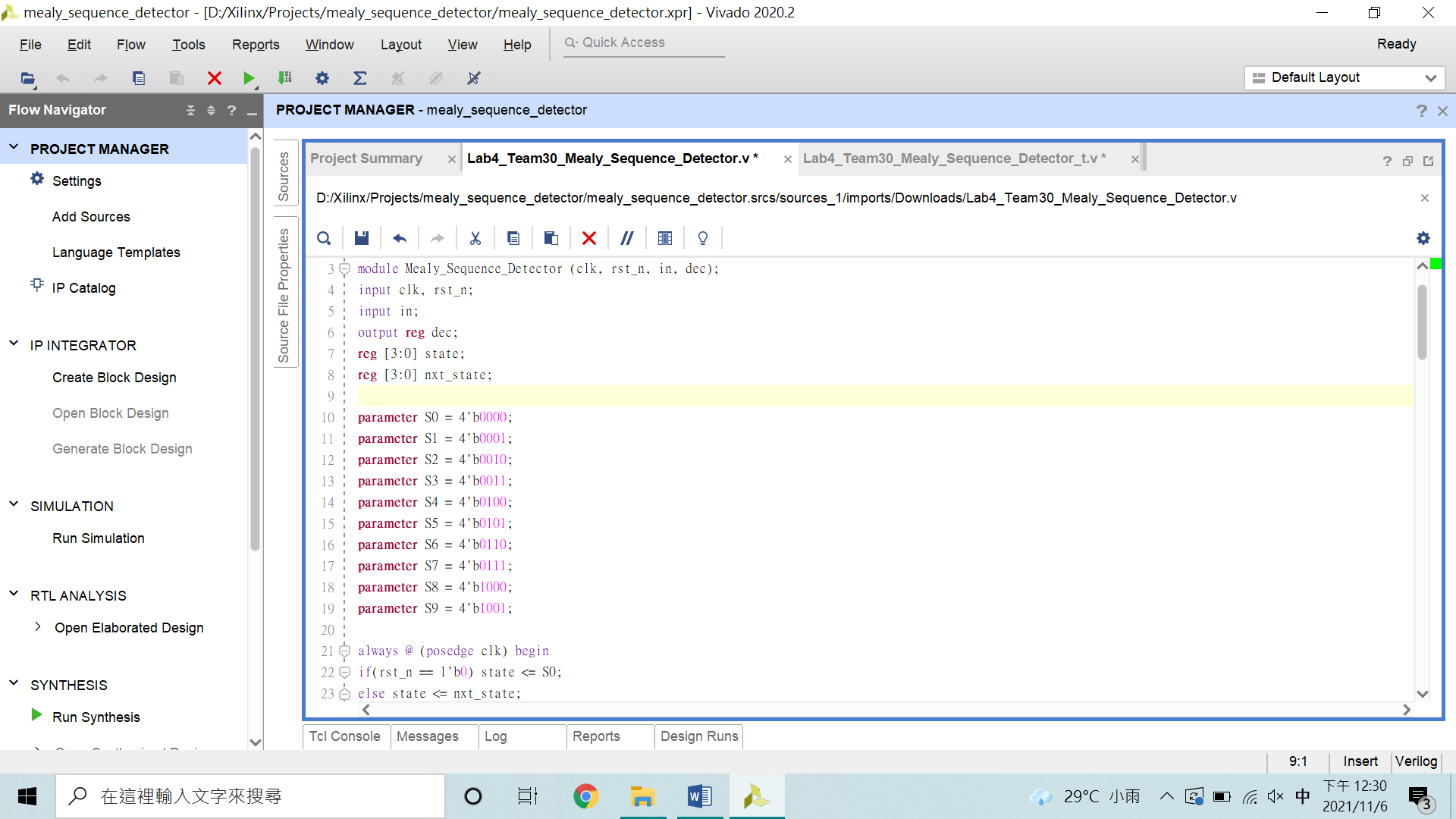
前8個cycle Multiplier的input 為1011 和1101

Many\_To\_One\_LFSR波形圖

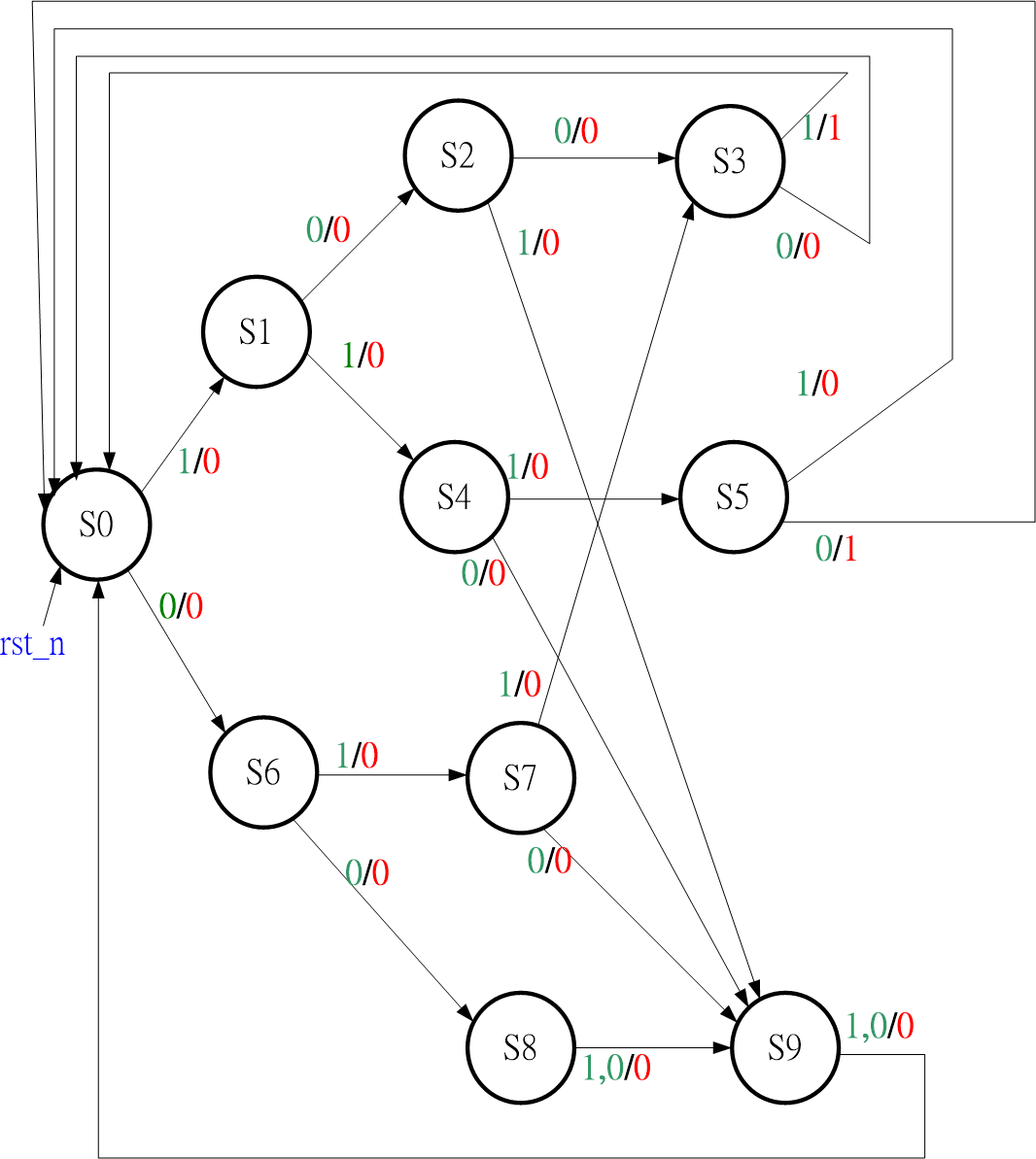
1. **Mealy machine sequence detector**

****

這題的mealy machine sequence detector要偵測0111、1001、1110三個sequence，並在輸入四個input後就返回到initial state再次進行偵測，input有clk、rst\_n、in，output是dec，module內部有state、nxt\_state來運作mealy machine，另外還有9個parameter(S0~S9)來定義各個state。

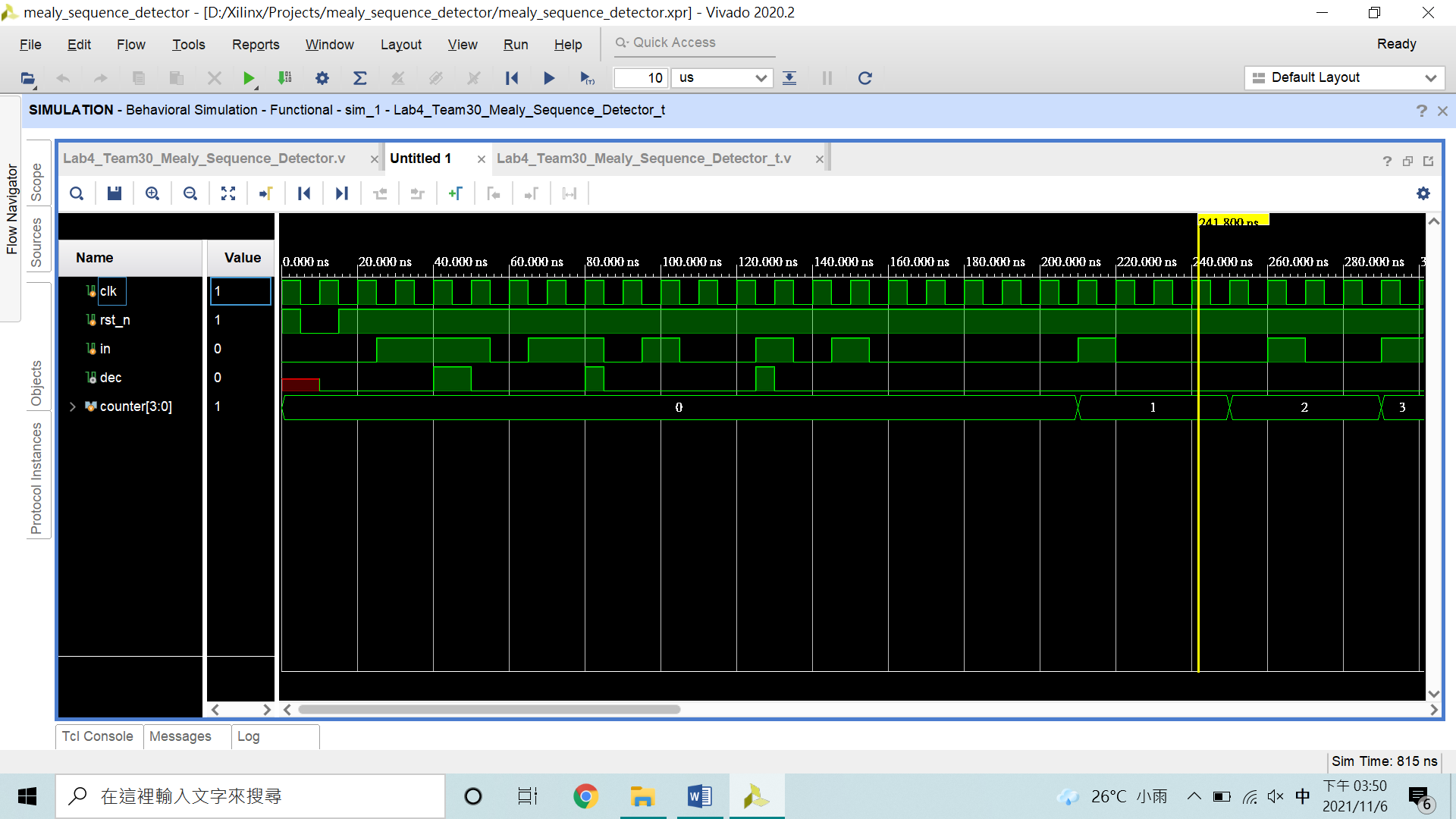


State diagram:

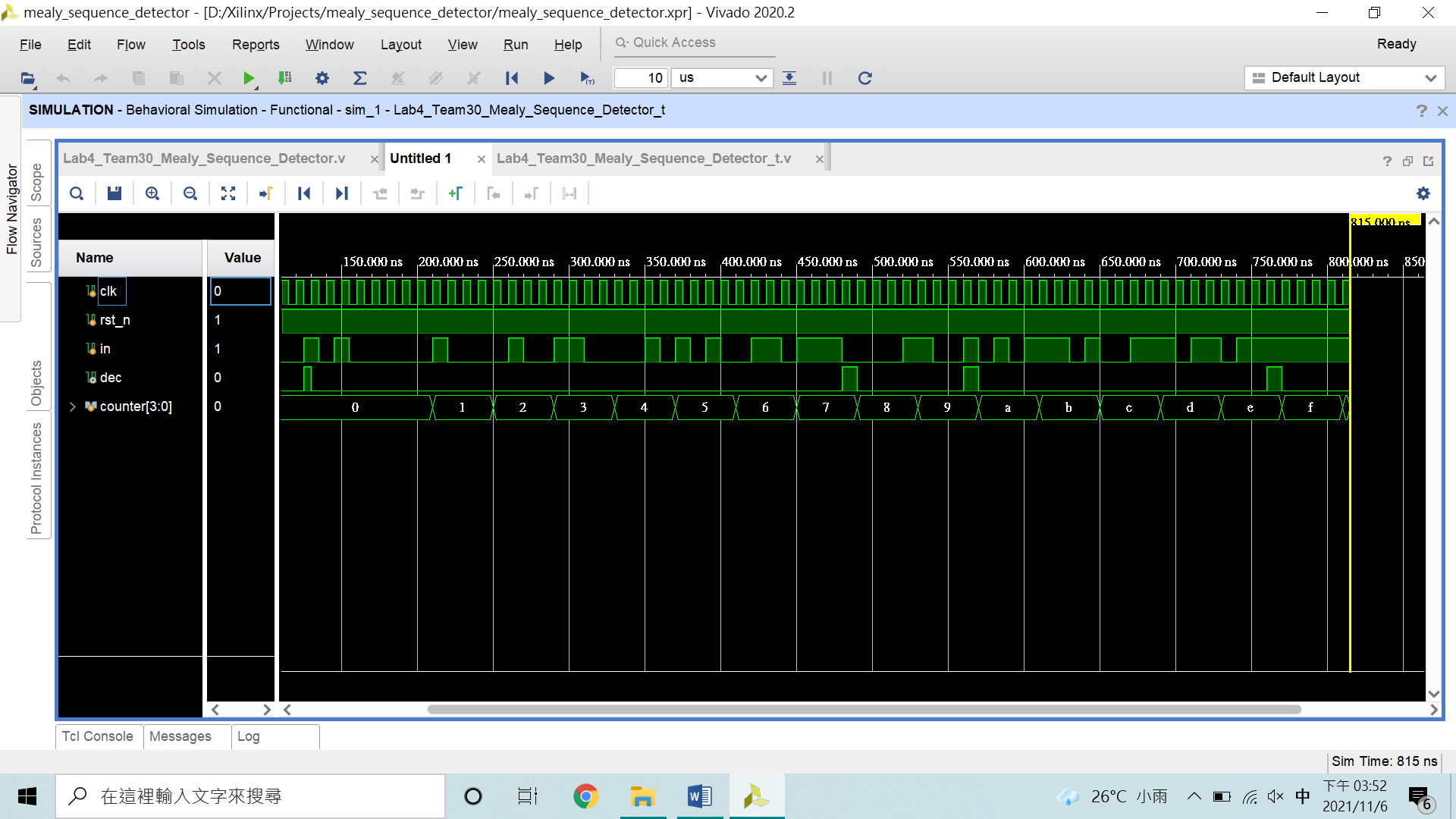


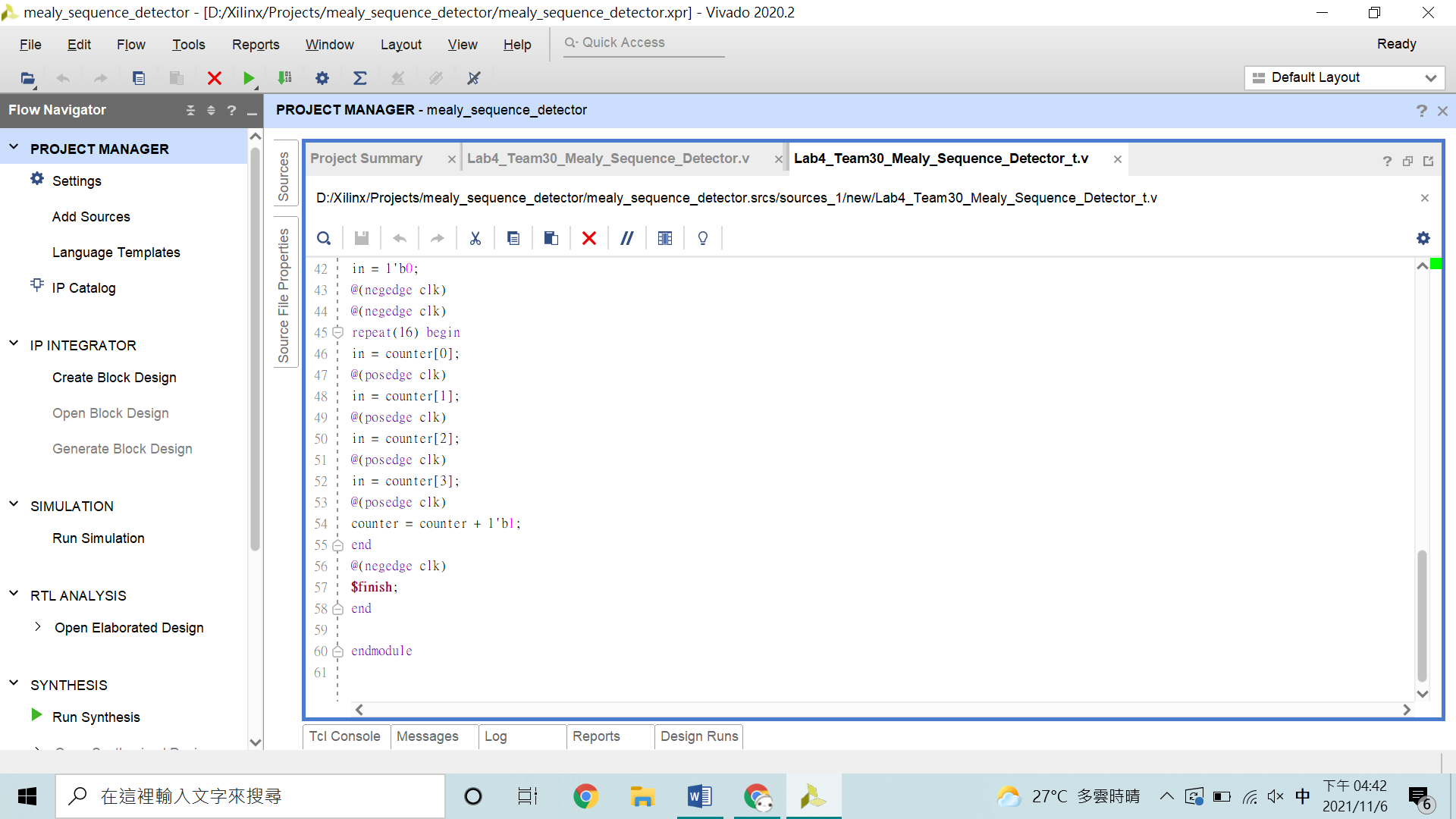
rst\_n == 0時state會到S0，然後開始接收input sequence並隨著clk變換state，如果連續在四個state接收到0111、1110或1001的話會output 1，不是以上的sequence則output 0，而因為是mealy machine，output會由state和input決定，所以輸出是asynchronous的。所有的路徑都會在4個cycle之後再次回到S0，重新偵測新的4-bit sequence。

**波形圖:**



先按照講解ppt上的波形範例測試，確認波形無誤



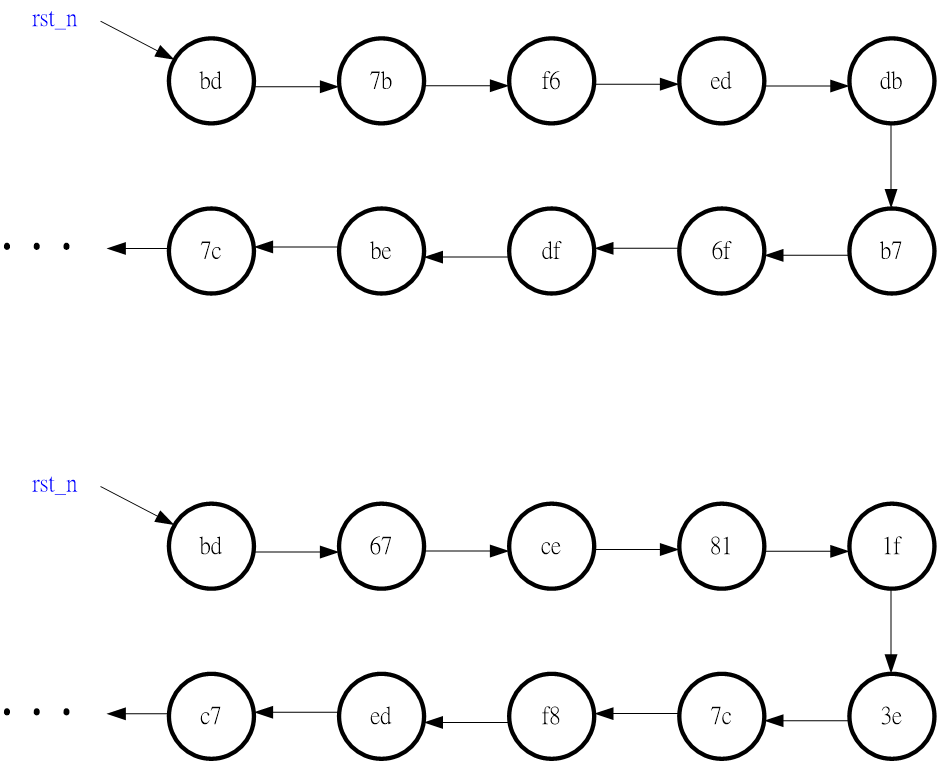


再來利用一個4-bit counter來輸入input sequence，以四個cycle為單位，依序輸入counter[0] ~ counter[3]，然後再將counter + 1，如此一來便能測試所有的4-bit input sequence組合，可以看到只有在sequence為0111、1001、1110時dec才會output 1。

1. **Basic question 3、4**

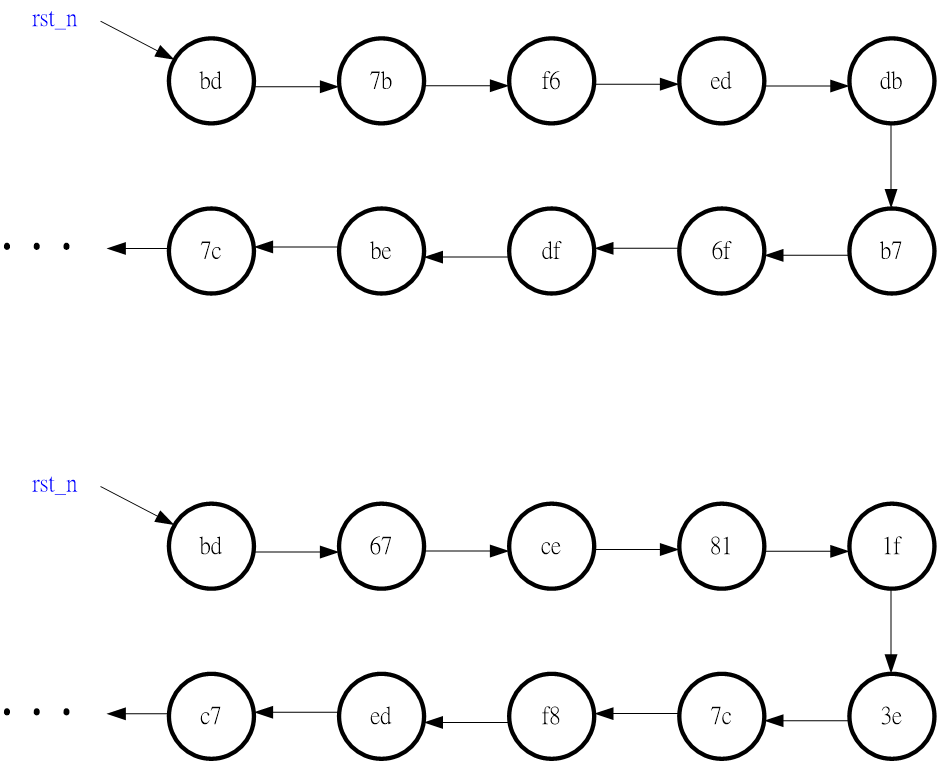
* **Many-to-one LFSR**

State diagram:



* **One-to-many LFSR**

State diagram:



這兩種LFSR如果把DFFs reset成8’d0，會使DFF xor之後的結果永遠都是0，造成每個state的output都是0，就失去本來要產生亂數的目的了。

1. **心得**

洪聖祥: 這次lab終於比較簡單，spec上block diagram也直接畫出來，code的部分就直接照著圖上打就好了。 這次比較花時間的地方是寫report。雖然說spec已經把block diagram的大概方向畫出來了，但要詳細解釋設計細節就比較困難了。從這次lab我發現如果有block diagram再寫code是一件很簡單的事情，因為只要描述這個block的行為然後把block跟block連起來就行了。

劉奇泓:這次的lab第一次開始寫FSM，我熟悉了mealy machine和moore machine的差別，也了解把每個state定義清楚，並畫好完整的state diagram是很重要的!尤其是在第四題mealy sequence detector把state diagram畫好後照著打出來，就能很快地做好了，除此之外這次的lab也學到了LFSR的運作，並在fpga題運用，我學會了如何用硬體的角度來製作亂數產生器。希望這些經驗都能累積起來，在下次的lab提升效率，使我有能力處理更多難關。

1. **分工**

洪聖祥:advanced question 1、2、3，report 1、2、3

劉奇泓: basic question 3、4 state diagram ，advanced question 4、fpga題，report 4、fpga題。