Lab5 IIR Filter

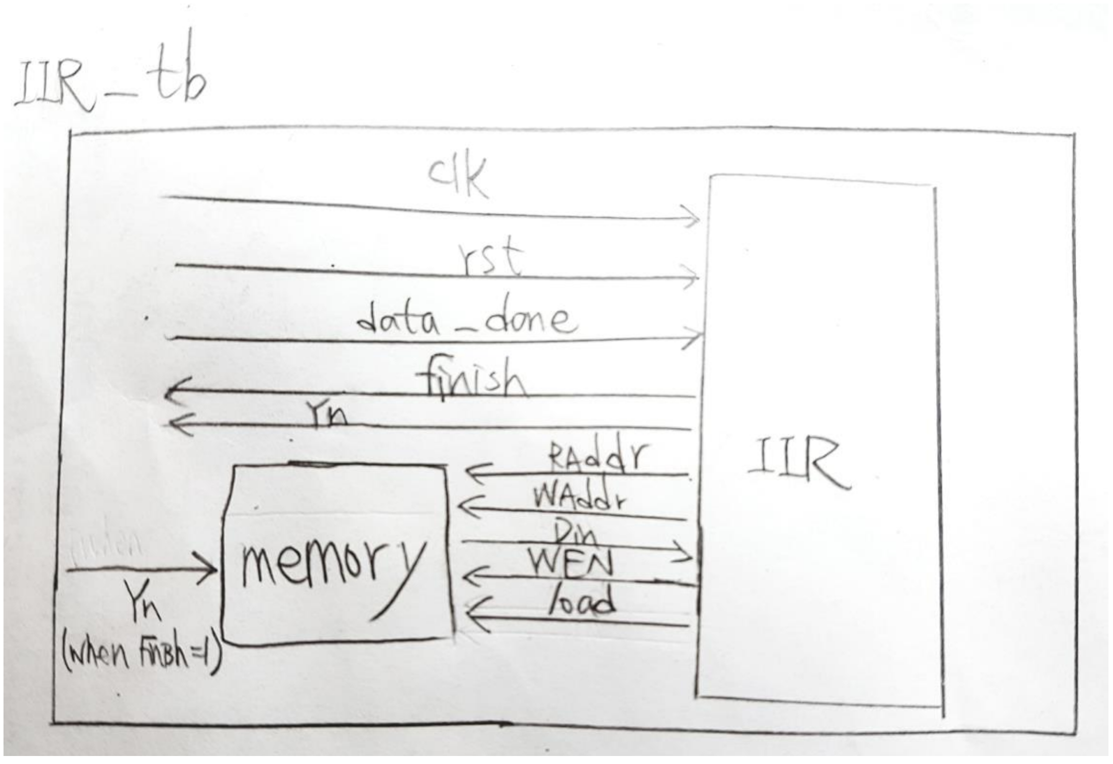
105031212 吳紹齊

一、 目標

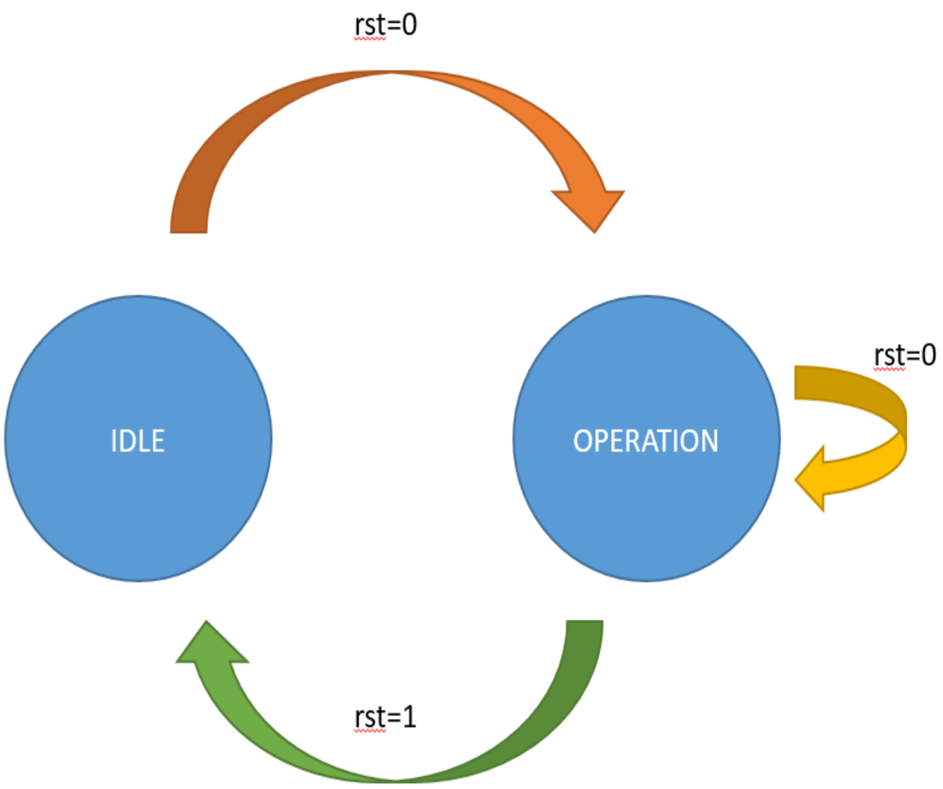
用 IIR 數位濾波器的原理，將造成音波不平穩的地方，利用取權重的方式，將音頻變得更平滑，進而消除音頻中的噪音。

二、 設計

IIR Filter 電路 (IIR.v) 、Test bench (IIR_tb.v)、Memory 的關係如下圖所示：

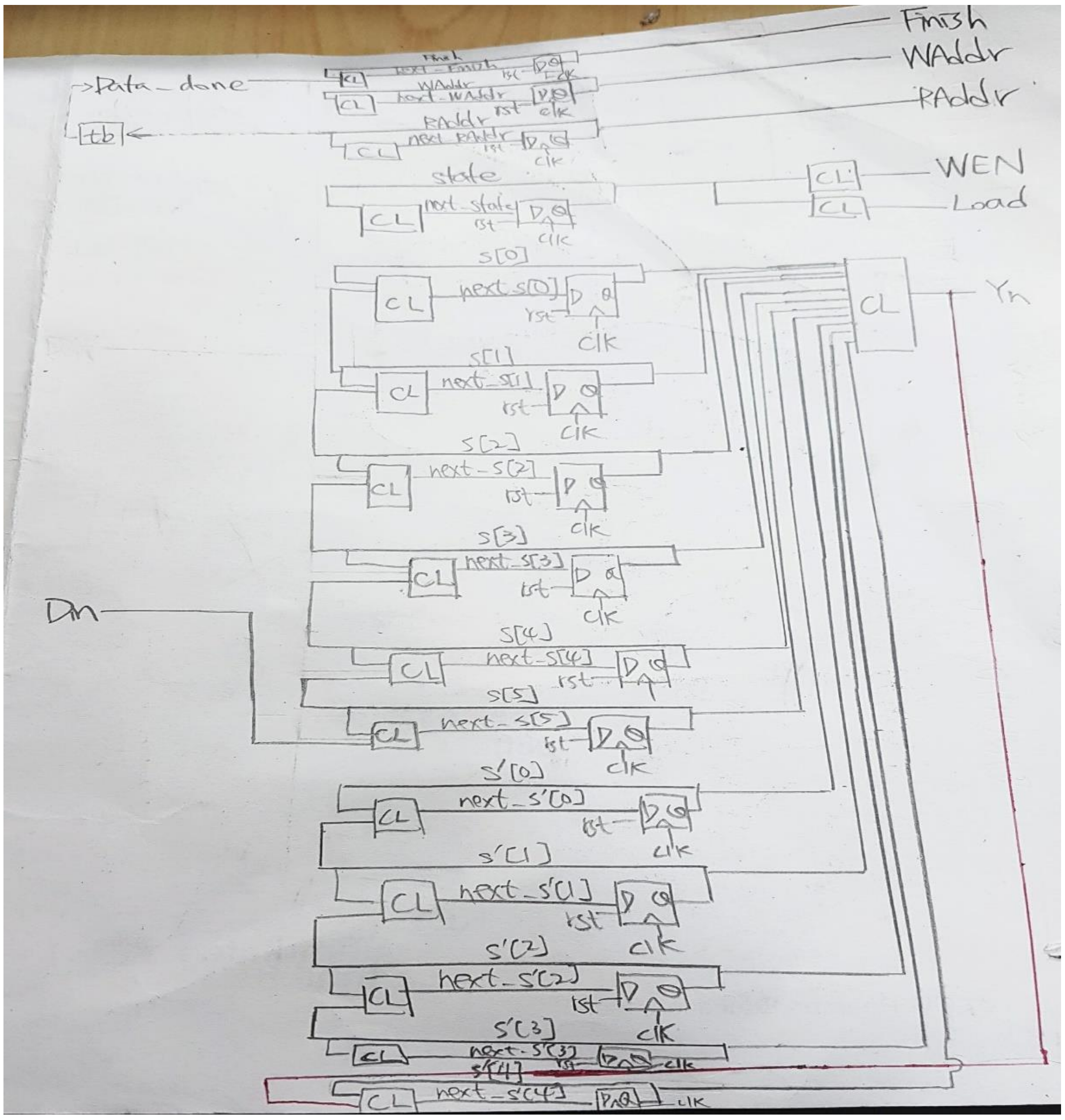


我設計了有兩個 State 的 FSM 來達成題目要求，以下是 IIR 的 State Transition Graph：

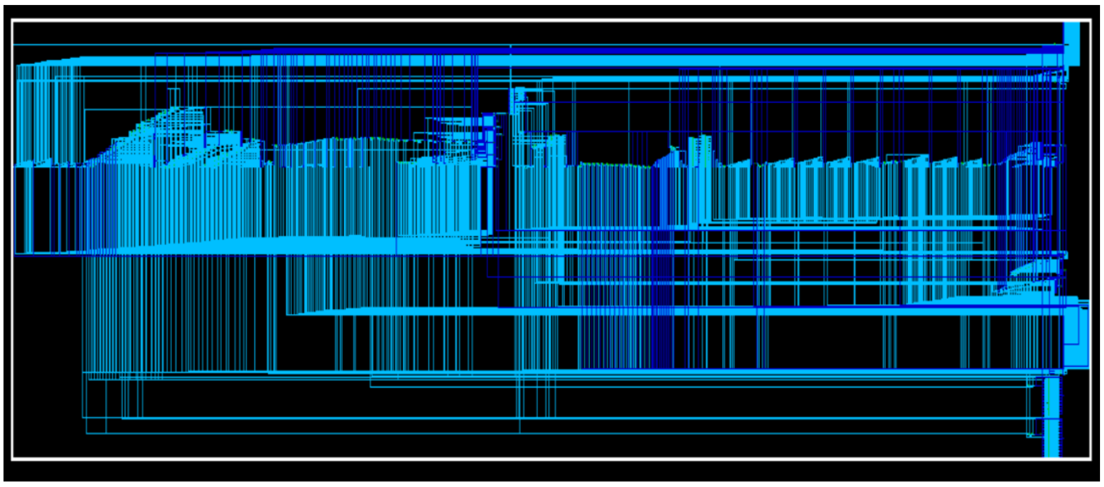


1. 若 rst = 1，回到 IDLE，全部 DFF 歸零。
2. 進入 OPERATION 後，WEN、load 拉起來，RAddr、WAddr 每個 cycle 都往上加一。
3. 直到 RAddr 加到最後一個，data_done 會舉起來，IIR 接著把 Finish 拉起來，最後 tb 會丟答案給 dat。
4. 每個音頻訊號運算完後 IIR 都會輸出答案 Yn。

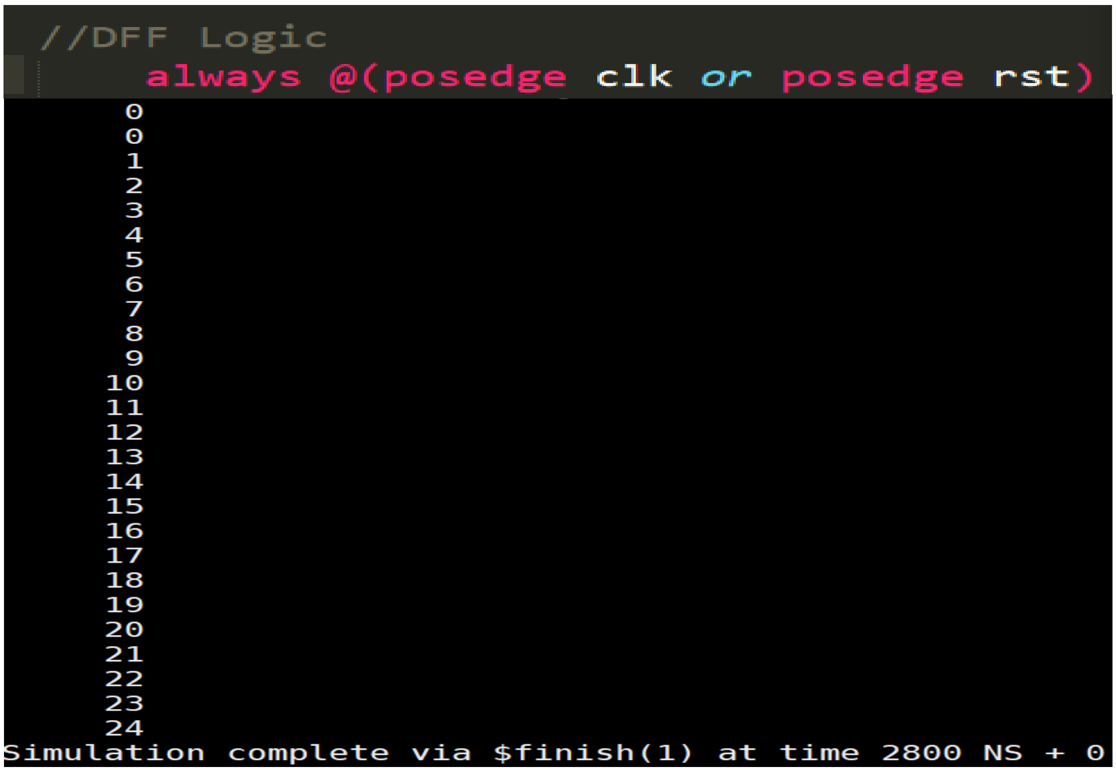
Block Diagram：



Dv 模擬電路：



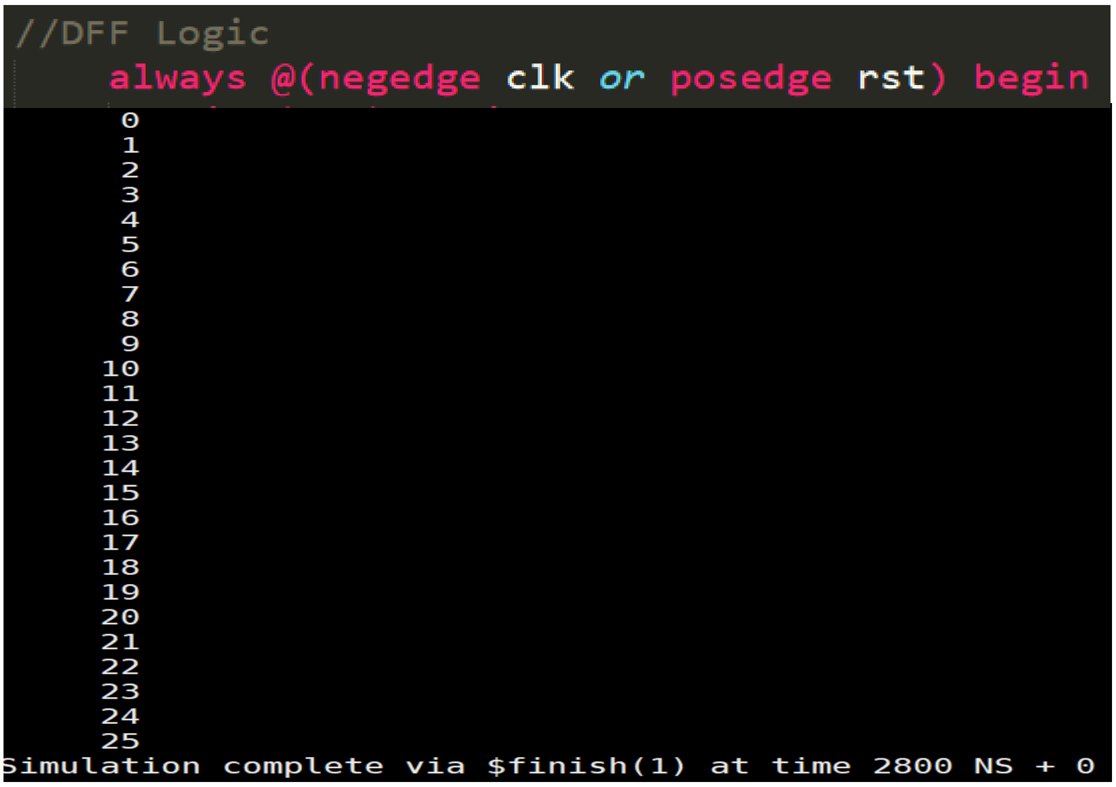
Trace Address：(以 test 為例)



```
//DFF Logic
    always @(posedge clk or posedge rst)
```

```
          0
          0
          1
          2
          3
          4
          5
          6
          7
          8
          9
         10
         11
         12
         13
         14
         15
         16
         17
         18
         19
         20
         21
         22
         23
         24
Simulation complete via $finish(1) at time 2800 NS + 0
```

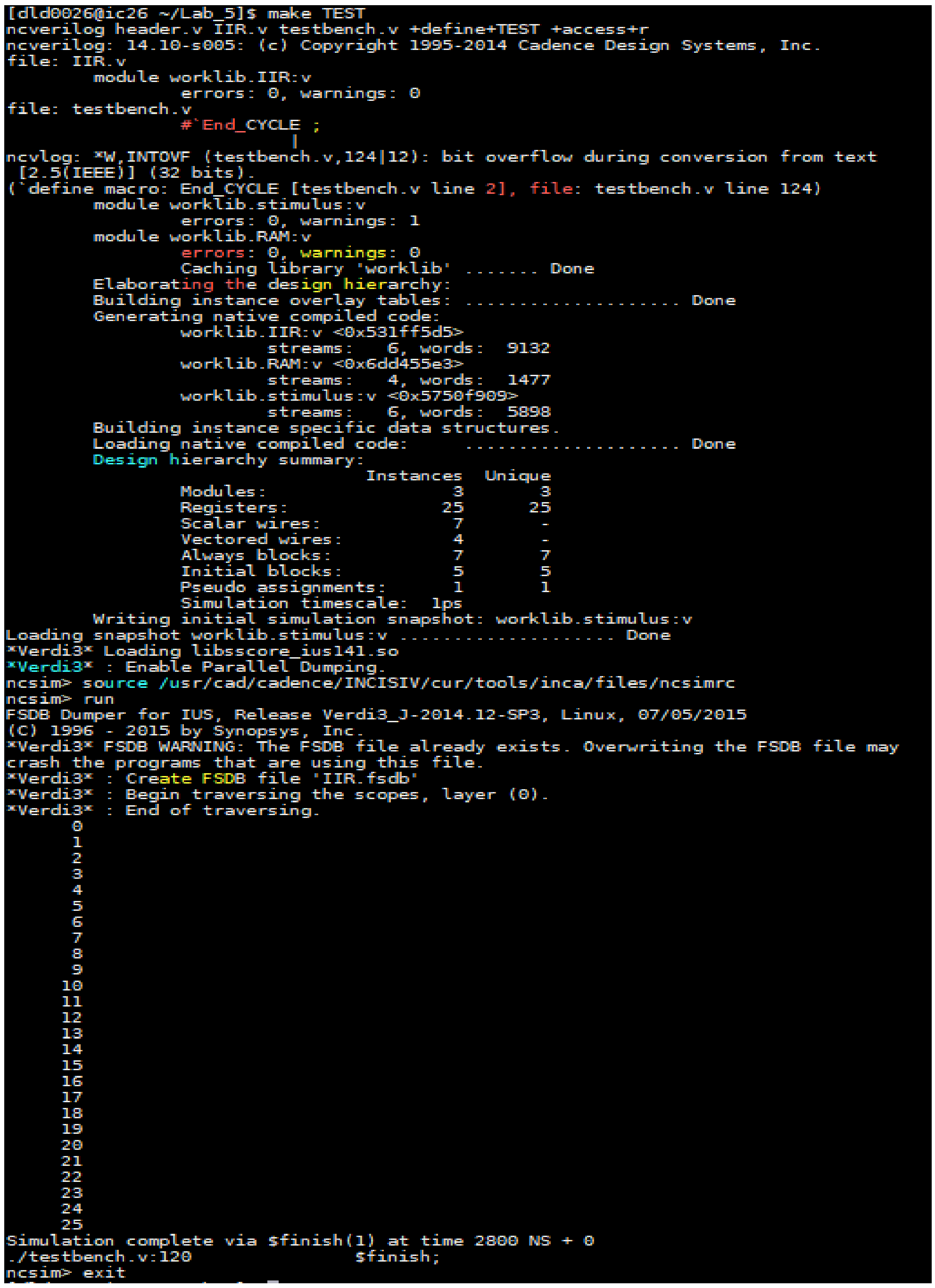
上面是錯誤的，這樣的錯誤會導致 Addr[25] 來不及，經過 nWave 的觀察以及更多的嘗試，發現最簡單的處理方法是換成 negedge clk。



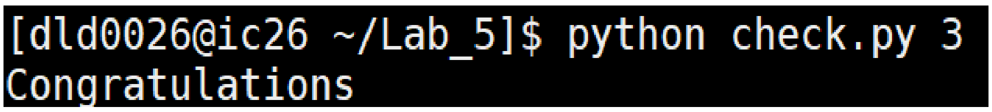
```
//DFF Logic
    always @(negedge clk or posedge rst) begin
```

```
          0
          1
          2
          3
          4
          5
          6
          7
          8
          9
         10
         11
         12
         13
         14
         15
         16
         17
         18
         19
         20
         21
         22
         23
         24
         25
Simulation complete via $finish(1) at time 2800 NS + 0
```

Make Sim：



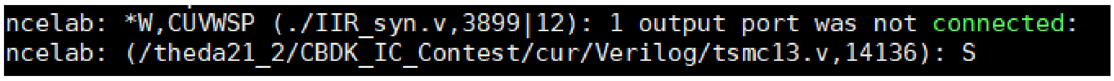
```
[dld0026@ic26 ~/Lab_5]$ make TEST
ncverilog header.v IIR.v testbench.v +define+TEST +access+r
ncverilog: 14.10-s005: (c) Copyright 1995-2014 Cadence Design Systems, Inc.
file: IIR.v
        module worklib.IIR:v
                errors: 0, warnings: 0
file: testbench.v
                #`End_CYCLE ;
                          |
ncvlog: *W,INTOVF (testbench.v,124|12): bit overflow during conversion from text
 [2.5(IEEE)] (32 bits).
(`define macro: End_CYCLE [testbench.v line 2], file: testbench.v line 124)
        module worklib.stimulus:v
                errors: 0, warnings: 1
        module worklib.RAM:v
                errors: 0, warnings: 0
                Caching library 'worklib' ....... Done
        Elaborating the design hierarchy:
        Building instance overlay tables: .................... Done
        Generating native compiled code:
                worklib.IIR:v <0x531ff5d5>
                        streams:   6, words:  9132
                worklib.RAM:v <0x6dd455e3>
                        streams:   4, words:  1477
                worklib.stimulus:v <0x5750f909>
                        streams:   6, words:  5898
        Building instance specific data structures.
        Loading native compiled code:     .................... Done
        Design hierarchy summary:
                         Instances  Unique
                Modules:         3       3
                Registers:      25      25
                Scalar wires:    7       -
                Vectored wires:  4       -
                Always blocks:   7       7
                Initial blocks:  5       5
                Pseudo assignments:      1       1
                Simulation timescale:  1ps
        Writing initial simulation snapshot: worklib.stimulus:v
Loading snapshot worklib.stimulus:v .................... Done
*Verdi3* Loading libsscore_ius141.so
*Verdi3* : Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
FSDB Dumper for IUS, Release Verdi3_J-2014.12-SP3, Linux, 07/05/2015
(C) 1996 - 2015 by Synopsys, Inc.
*Verdi3* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file may
crash the programs that are using this file.
*Verdi3* : Create FSDB file 'IIR.fsdb'
*Verdi3* : Begin traversing the scopes, layer (0).
*Verdi3* : End of traversing.
          0
          1
          2
          3
          4
          5
          6
          7
          8
          9
         10
         11
         12
         13
         14
         15
         16
         17
         18
         19
         20
         21
         22
         23
         24
         25
Simulation complete via $finish(1) at time 2800 NS + 0
./testbench.v:120             $finish;
ncsim> exit
```



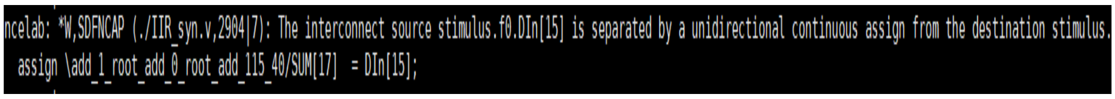
```
[dld0026@ic26 ~/Lab_5]$ python check.py 3
Congratulations
```

Make Syn：(以 test 為例，太長了，擷取片段)

常見 Warning：



```
ncelab: *W,CUVWSP (./IIR_syn.v,3899|12): 1 output port was not connected:
ncelab: (/theda21_2/CBDK_IC_Contest/cur/Verilog/tsmc13.v,14136): S
```



```
ncelab: *W,SDFNCAP (./IIR_syn.v,2904|7): The interconnect source stimulus.f0.DIn[15] is separated by a unidirectional continuous assign from the destination stimulus.
  assign \add_1_root_add_0_root_add_115_40/SUM[17]  = DIn[15];
```

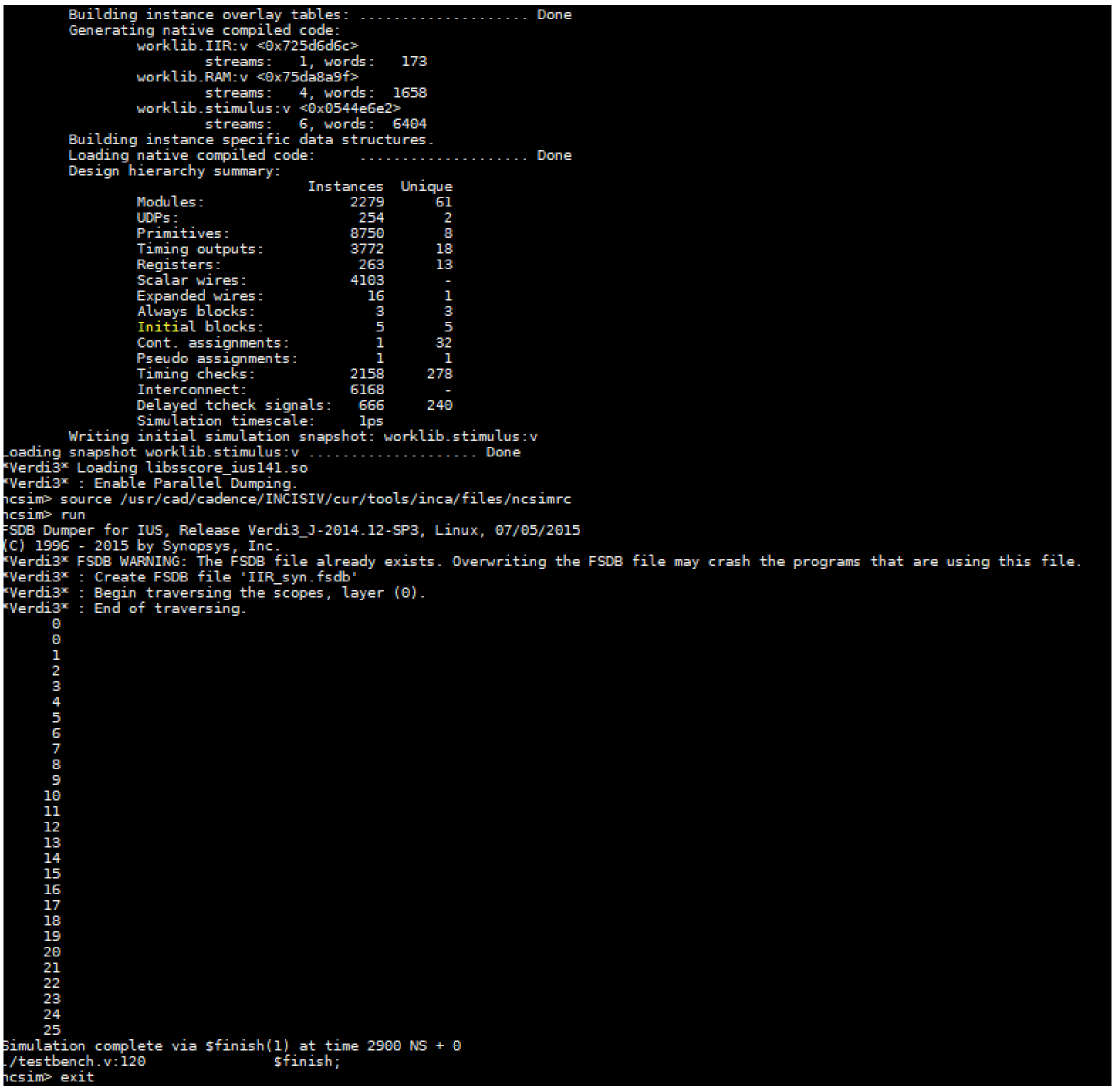
以上是出現的兩個 WARNING，我猜測主要的來源是因為

s[5] = {{2{DIn[15]}}, DIn, {7{1'b0}}} 還有 Yn = {rlt[24], rlt[21:7]}

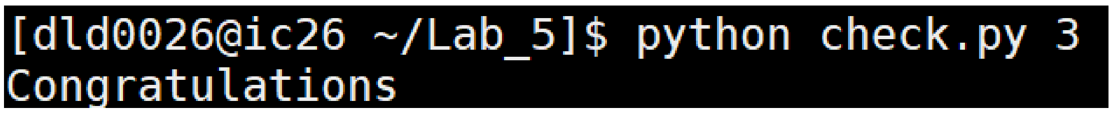
這兩者都算是用手動的方式，把完整的 data (Din 或是 rlt) 拆開，

組成我們要的 data，造成有些電路在這裡斷掉 (沒繼續接下去)。

結果：



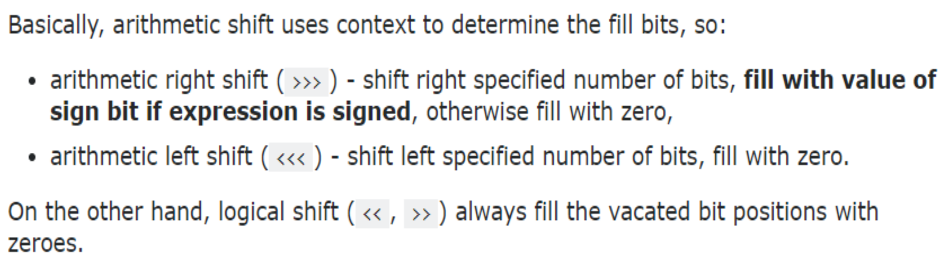
```
        Building instance overlay tables: .................... Done
        Generating native compiled code:
                worklib.IIR:v <0x725d6d6c>
                        streams:   1, words:   173
                worklib.RAM:v <0x75da8a9f>
                        streams:   4, words:  1658
                worklib.stimulus:v <0x0544e6e2>
                        streams:   6, words:  6404
        Building instance specific data structures.
        Loading native compiled code:     .................... Done
        Design hierarchy summary:
                             Instances  Unique
                Modules:          2279      61
                UDPs:              254       2
                Primitives:       8750       8
                Timing outputs:   3772      18
                Registers:         263      13
                Scalar wires:     4103       -
                Expanded wires:     16       1
                Always blocks:       3       3
                Initial blocks:      5       5
                Cont. assignments:   1      32
                Pseudo assignments:  1       1
                Timing checks:    2158     278
                Interconnect:     6168       -
                Delayed tcheck signals:  666     240
                Simulation timescale:     1ps
        Writing initial simulation snapshot: worklib.stimulus:v
Loading snapshot worklib.stimulus:v .................... Done
*Verdi3* Loading libsscore_ius141.so
*Verdi3* : Enable Parallel Dumping.
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
FSDB Dumper for IUS, Release Verdi3_J-2014.12-SP3, Linux, 07/05/2015
(C) 1996 - 2015 by Synopsys, Inc.
*Verdi3* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file may crash the programs that are using this file.
*Verdi3* : Create FSDB file 'IIR_syn.fsdb'
*Verdi3* : Begin traversing the scopes, layer (0).
*Verdi3* : End of traversing.
           0
           0
           1
           2
           3
           4
           5
           6
           7
           8
           9
          10
          11
          12
          13
          14
          15
          16
          17
          18
          19
          20
          21
          22
          23
          24
          25
Simulation complete via $finish(1) at time 2900 NS + 0
./testbench.v:120               $finish;
ncsim> exit
```



```
[dld0026@ic26 ~/Lab_5]$ python check.py 3
Congratulations
```

三、 心得與討論

1. 因為對於 syn 的了解還不夠充分，所以 sim 沒錯可是 syn 有錯的時候，我會從調整 clk 開始試試看，之後再試試把 reg 都接上 DFF，過程再看看 nWave，如果真的再不行就會稍微改變一下設計，最近寫 Lab 大概都是這樣，暑假應該會把 Lab 都拿出來重新打一次，想辦法一開始就寫出能順利合成電路的 code，或是有效率的找到 sim 跟 syn 的盲點。
2. 嘗試用更多 always block，分為 DFF Logic (訂 next_x 的值) OP Logic (主要運算)、Output Logic (輸出要求的值)、Next_State Logic(管理 state)，效果很好，debug 的時候非常方便。
3. 這次卡關最久是因為 state 的 register 的 bit 給錯(太小)，導致 state 完全跑不動，這種問題真的要碰到才會知道。
4. 因為 Lab2 在寫算術位移的時候是自己寫 extend bit，”>>>” 這個符號的用法是助教講解時才知道，這是第一次用，所以還是研究了一下，後來發現超級好用!!!!!!!!



Basically, arithmetic shift uses context to determine the fill bits, so:

- arithmetic right shift (`>>>`) - shift right specified number of bits, **fill with value of sign bit if expression is signed**, otherwise fill with zero,
- arithmetic left shift (`<<<`) - shift left specified number of bits, fill with zero.

On the other hand, logical shift (`<<` , `>>`) always fill the vacated bit positions with zeroes.