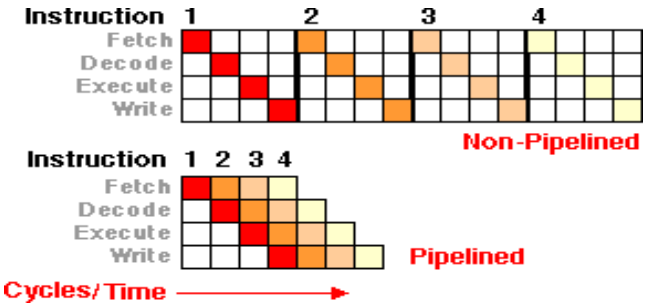
103060004 劉士弘

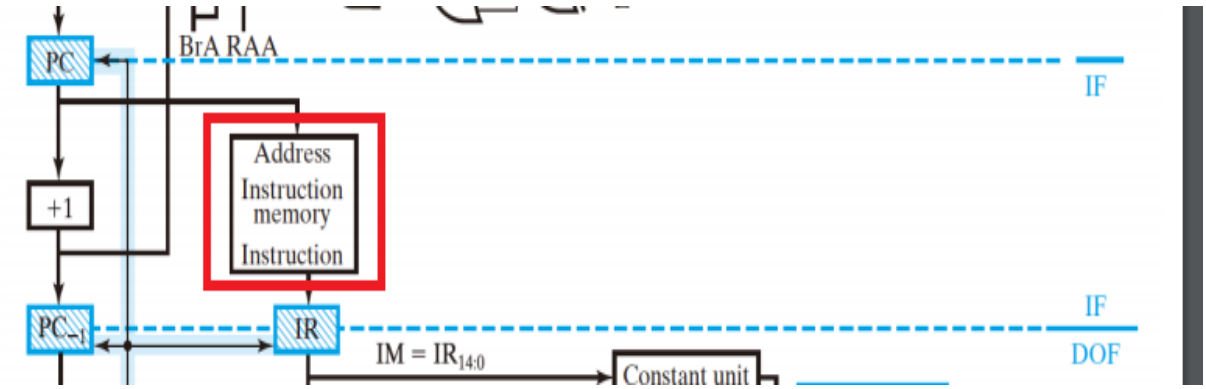
DSD_Final

這次我們主要要實做的事 pipeline，而 pipeline 的好處就在於它可以把過程分成好幾個階段，不會像一般的程式，需要等到一個城市跑完之後才可以去跑下一個，以右圖為例；



可以看見，節省了許多時間，所以我這次設計的概念，就是依照老師講義上的圖，把流程分成了 4 個 state，IF、DOF、EXE、WB。

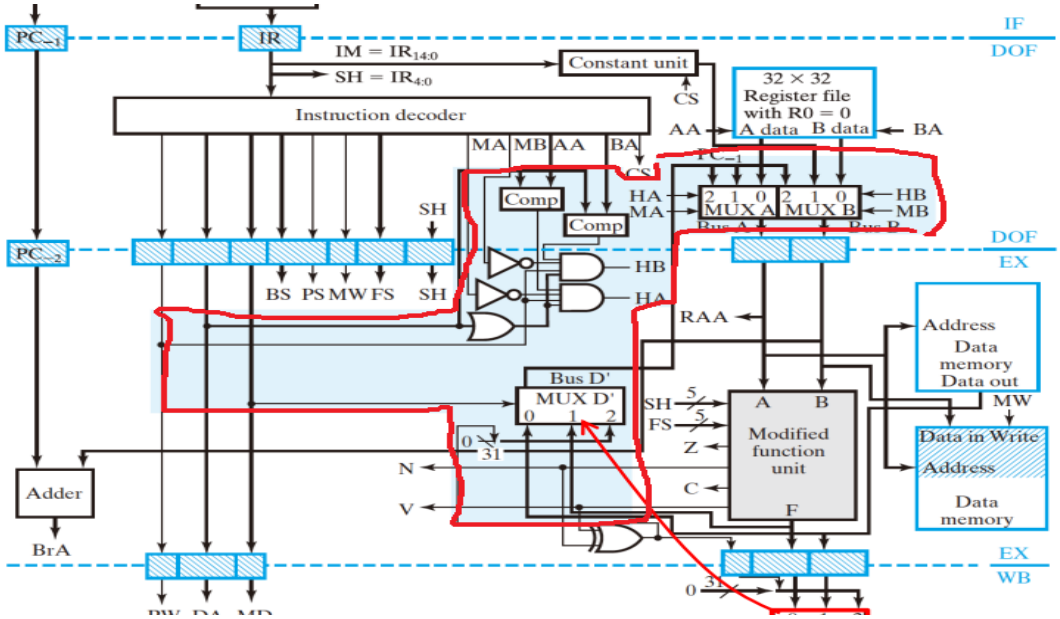
第一階段，IF_state，就是從 I_memory 中讀取這一個 cycle 所需要的值出來，之後則會把這個 instruction 丟到下一個 state，也就是 DOF_state，這個 state，主要是把 instruction decode 出來，然後把下一個 EXE_state 需要用到的東西，傳下去，而 MUXA 和 MUXB 所選出來的東西，則是會看你的 instruction，是做 R_type 或者 I_type 或是 others。



```
    Register_File Register_file(
    .clk(clk),
    .rst_n(rst_n),
    .instruction(instruction),
    .AA(DOF_AA),
    .BA(DOF_BA),
    .BUS_D(BUS_D),
    .RW(WB_RW),
    .DA(WB_DA),
    .A_data(A_data),
    .B_data(B_data)
);
    Decoder  instruction_decode(
      .clk(clk),
      .rst_n(rst_n),
      .instruction(instruction),
      .flush(IC_sel) ,
      .RW(DOF_RW),
      .MD (DOF_MD),
      .BS(DOF_BS) ,
      .PS(DOF_PS) ,
      .MW(DOF_MW) ,
      .FS(DOF_FS) ,
      .MB(MB) ,
      .MA (MA),
      .CS(CS) ,
      .DA(DOF_DA) ,
      .AA(DOF_AA) ,
      .BA(DOF_BA)
  );
```

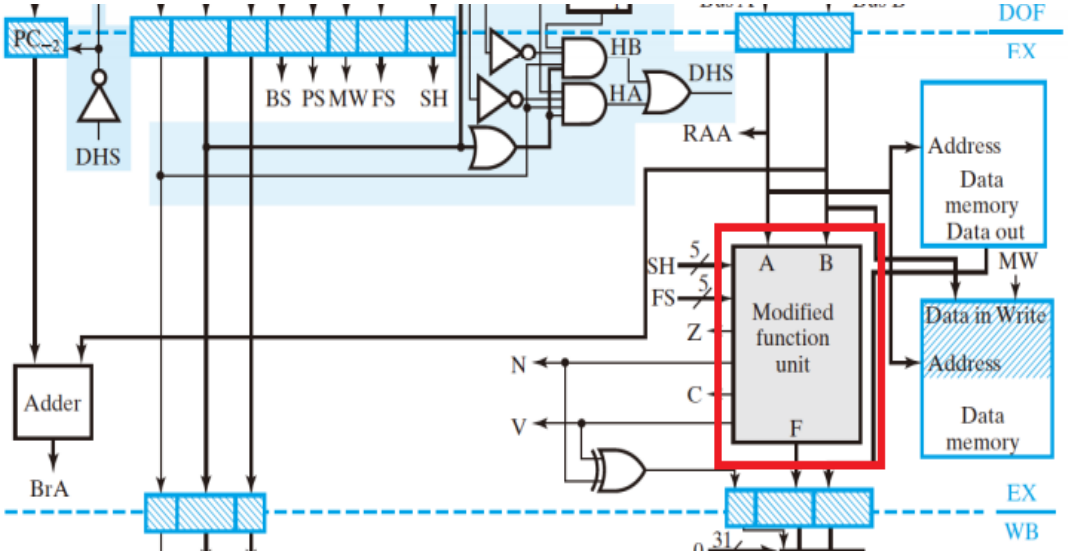
```
MUX_A   Mux_A(
.clk(clk),
.rst_n(rst_n),
.EX_Hazard_A(EX_Hazard_A),
.WB_Hazard_A(WB_Hazard_A),
.Forward(Forward),
.BUS_D(BUS_D),
.pc_1(pc_1),
.A_data(A_data) ,
.MA(MA),
.Bus_A(Bus_A),
.EX_MD(EX_MD)

) ;
MUX_B Mux_B(
.constant(constant),
.B_Data(B_data),
.MB(MB) ,
.Bus_B(Bus_B),
.EX_Hazard_B(EX_Hazard_B),
.WB_Hazard_B(WB_Hazard_B),
.Forward(Forward),
.BUS_D(BUS_D)
) ;
```

在 DOF_state 時，有可能會發生 data_hazard，就是當 DOF 需要的 regsiter 的值和要被 WB 回去的 register 或 EXE 中的 register 的值一樣的時候，因為還沒 write back，所以我解決的方法，就是使用 Forwording，把現在 write_back 的值直接塞到 DOF 裡面，讓 DOF 可以取到正確的資訊，已完成這個 state。



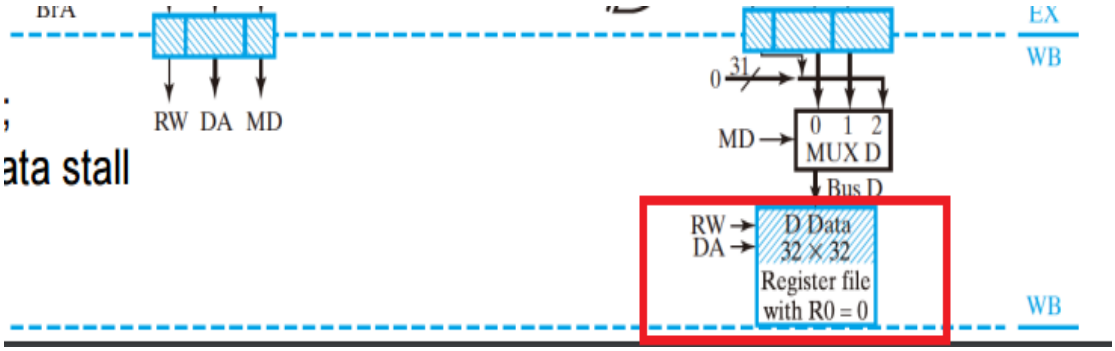

```
always @(*) begin
if((EX_RW != 0) && (EX_DA != 5'b00000) && (MA == 0) && (EX_DA == DOF_AA))
    begin
        EX_Hazard_A = 1'b1;
    end
else
    begin
        EX_Hazard_A = 1'b0;
    end
if((EX_RW != 0) && (EX_DA != 5'b00000) && (MB == 0) && (EX_DA == DOF_BA))
    begin
        EX_Hazard_B = 1'b1;
    end
else
    begin
        EX_Hazard_B = 1'b0;
    end
if((WB_RW != 0) && (WB_DA != 5'b00000) && (MA == 0) && (WB_DA == DOF_AA))
    begin
        WB_Hazard_A = 1'b1;
    end
else
    begin
        WB_Hazard_A = 1'b0;
    end
if((WB_RW != 0) && (WB_DA != 5'b00000) && (MB == 0) && (WB_DA == DOF_BA))
    begin
        WB_Hazard_B = 1'b1;
    end
else
    begin
        WB_Hazard_B = 1'b0;
    end
```

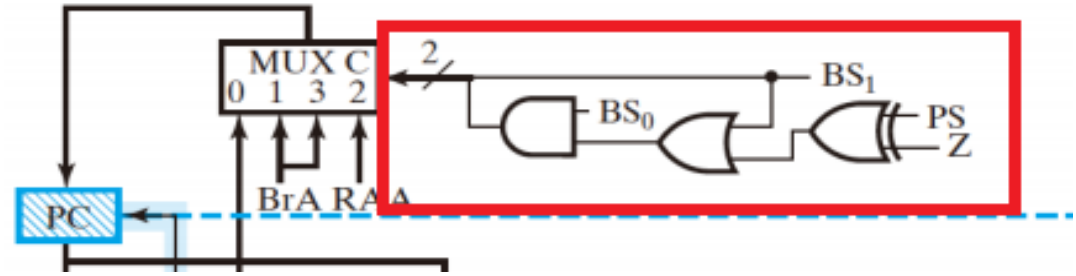
下一個 state 就是 EXE_state，需要做 function_unit 的運算，此部分跟上次作業並無太大差別，最主要改變只有在 bit 數從 16bit 變成 32bit，而出來的值會需要和 D_memory 出來的值以及 N 和 Z 做 exclusive 的值進到 MUX_D，讓 WB_state 去做判斷看要取出哪一個值。




```
Function_Unit Function_unit(
.clk(clk),
.rst_n(rst_n),
.A(EX_BUS_A),
.B(EX_BUS_B),
.FS(EX_FS),
.SH(EX_SH),
.Z(Z),
.N(N),
.C(C),
.V(V),
.F(F)

) ;
```

最後 WB_state，則是把值寫到 register 裡面，用 MUX_D 選出來的值，而此時 PC 會+1，而這時又會出現問題，因為如果出現 branch 的話，PC 值則會跳動，而這時前面讀進來的 instruction 則不能夠用，所以我就把讀進來的 instruction 給 flush 掉，而 branch 後的 PC 值，則是 BRA 和 RAA 這兩個 register 的值決定。



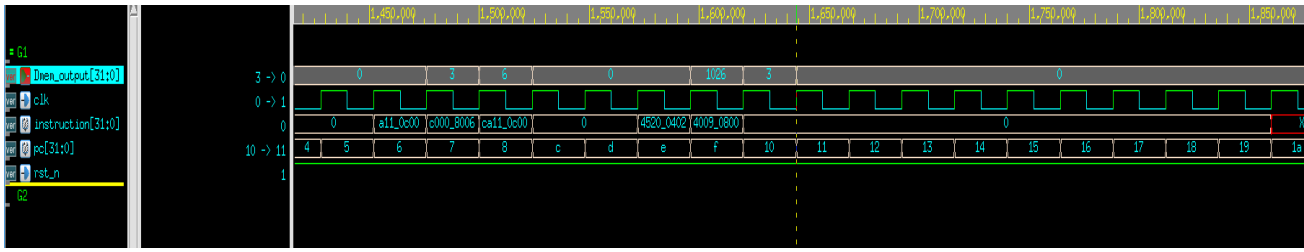



```
    always @(*) begin //判斷branch
        if(!DOF_flush) begin
            C_sel = {EX_BS[1], (((EX_PS ^ Z) | EX_BS[1])&EX_BS[0])};
        end else begin
            C_sel = 0;
        end
    end

* MUXC */
    always @(*) begin
        pc_next = pc + 1;
        case(C_sel)
            2'b01 : pc_next = BrA;
            2'b11 : pc_next = BrA;
            2'b10 : pc_next = RAA;
        endcase

    end
 endmodule
```

下面是我測試的 testcase :GCD 以及 Summation

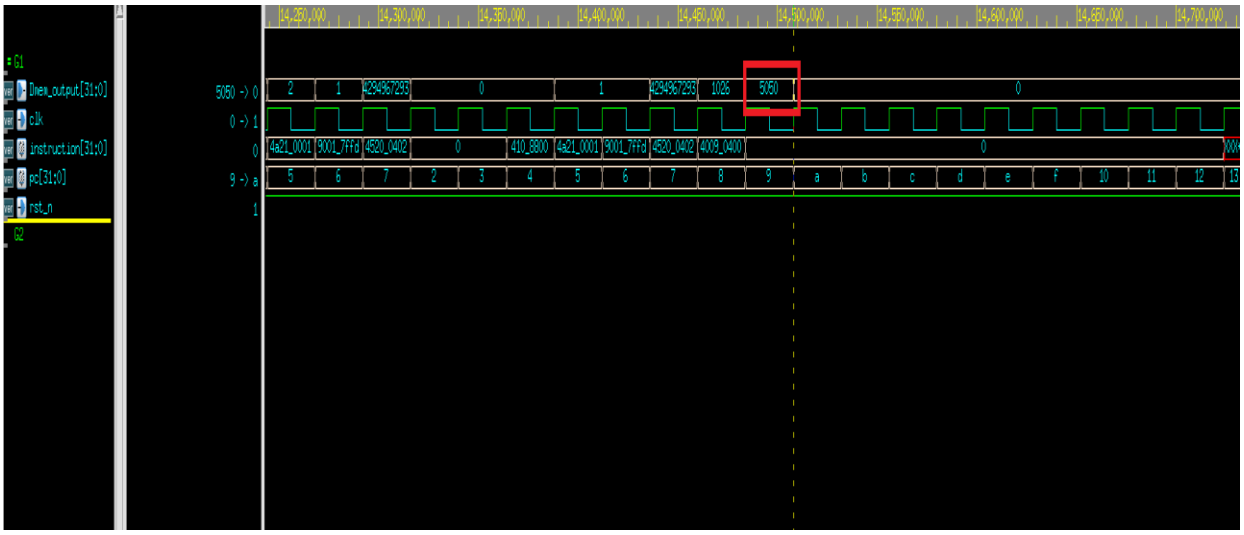
1. GCD(27 和 12 的 GCD)

```
[                 1290] Read Pipeline.I_mem.read_mem[00c]: 45200402
[                 1310] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx
[                 1310] Read Pipeline.I_mem.read_mem[00d]: 40090800
[                 1330] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx
[                 1330] Read Pipeline.I_mem.read_mem[00e]: 00000000
[                 1350] Read Pipeline.d_mem.read_mem[402]: 00000000
[                 1350] Read Pipeline.I_mem.read_mem[00f]: 00000000
 *****Register[18] IS          3 *****
[                 1370] Write to 00000003 Pipeline.d_mem.write_mem[402]
[                 1370] Read Pipeline.I_mem.read_mem[010]: 00000000
[                 1390] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx
[                 1390] Read Pipeline.I_mem.read_mem[011]: 00000000
[                 1410] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx
[                 1410] Read Pipeline.I_mem.read_mem[012]: 00000000
[                 1430] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx
[                 1430] Read Pipeline.I_mem.read_mem[013]: 00000000
[                 1450] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx
[                 1450] Read Pipeline.I_mem.read_mem[014]: 00000000
[                 1470] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx
[                 1470] Read Pipeline.I_mem.read_mem[015]: 00000000
[                 1490] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx
```

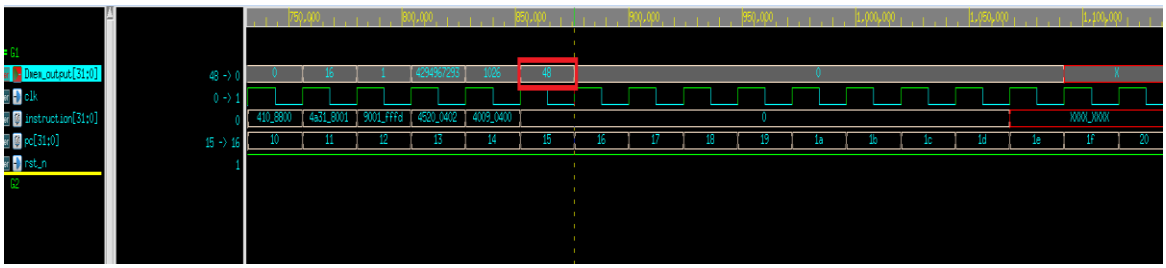


2. Summation(1+~100)

```
[                11970] Read Pipeline.I_mem.read_mem[003]: 4a210001

[                11990] Read Pipeline.d_mem.read_mem[3b9]: xxxxxxxx

[                11990] Read Pipeline.I_mem.read_mem[004]: 90017ffd

[                12010] Read Pipeline.d_mem.read_mem[001]: xxxxxxxx

[                12010] Read Pipeline.I_mem.read_mem[005]: 45200402

 *****Register[ 1] IS       5050 *****
[                12030] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx

[                12030] Read Pipeline.I_mem.read_mem[006]: 40090400

 *****Register[ 2] IS       5050 *****
[                12050] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx

[                12050] Read Pipeline.I_mem.read_mem[007]: 00000000

[                12070] Read Pipeline.d_mem.read_mem[402]: xxxxxxxx

[                12070] Read Pipeline.I_mem.read_mem[008]: 00000000

 *****Register[18] IS       5050 *****
[                12090] Write to 000013ba Pipeline.d_mem.write_mem[402]

[                12090] Read Pipeline.I_mem.read_mem[009]: 00000000

[                12110] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx
```



3. multipilication



```
*****Register[1] IS         48 *****
*****Register[2] IS         16 *****
[                876] Write to 00000030 Pipeline.d_mem.write_mem[402]

[                876] Read Pipeline.I_mem.read_mem[015]: 00000000

[                900] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx

[                900] Read Pipeline.I_mem.read_mem[016]: 00000000
```

4. my_test

```
                 7260] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx

                 7260] Read Pipeline.I_mem.read_mem[007]: 00000000

                 7284] Read Pipeline.d_mem.read_mem[402]: xxxxxxxx

                 7284] Read Pipeline.I_mem.read_mem[008]: 00000000

*****Register[1] IS        1275 *****
*****Register[2] IS           0 *****
                 7308] Write to 000004fb Pipeline.d_mem.write_mem[402]

                 7308] Read Pipeline.I_mem.read_mem[009]: 00000000

                 7332] Read Pipeline.d_mem.read_mem[000]: xxxxxxxx
```

這次 project 遇到的困難其實蠻多的，一開始的讀檔進來就想了很久，最後研究出用 preload 這個參數弄進來，後面又發現，register 值會晚一個 cycle 近來，所以我就讓他早一個 cycle 先 preload 進來做判斷，還有 forwardind，因為一開始沒有使用 forwarding，所以跑出來的結果非常奇怪，最後才發現是因為 data_hazard 的關係。

Systhesis:

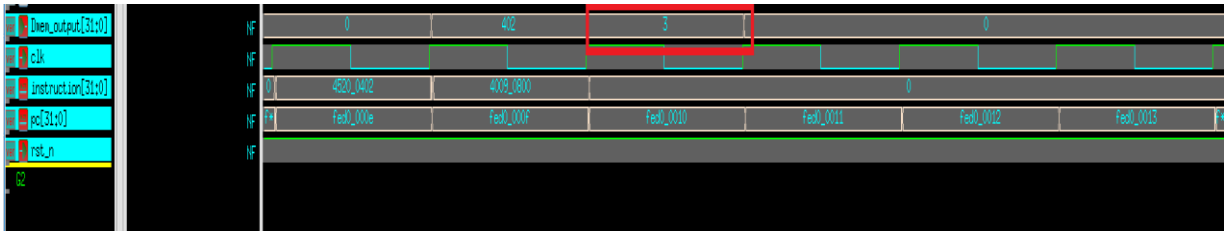
Timing Report:

```
  Dmem_output_reg[21]/CK (EDFFXL)                         0.00      10.00 r
  library setup time                                     -0.33       9.67
  data required time                                                 9.67
  --------------------------------------------------------------------------
  data required time                                                 9.67
  data arrival time                                                 -9.67
  --------------------------------------------------------------------------
  slack (MET)                                                        0.00


  Startpoint: Dmem_input[6]
              (input port)
  Endpoint: Data_addr[6]
            (output port)
  Path Group: default
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  input external delay                     0.00       0.00 f
  Dmem_input[6] (in)                       0.00       0.00 f
  U426/Y (AO22X1)                          0.30       0.30 f
  Mux_A/Forward[6] (MUX_A)                 0.00       0.30 f
  Mux_A/U14/Y (AOI22XL)                    0.14       0.44 r
  Mux_A/U13/Y (NAND2XL)                    0.09       0.53 f
  Mux_A/Bus_A[6] (MUX_A)                   0.00       0.53 f
  U331/Y (OAI2BB2XL)                       0.17       0.70 f
  Data_addr[6] (out)                       0.00       0.70 f
  data arrival time                                   0.70

  max_delay                               10.00      10.00
  output external delay                    0.00      10.00
  data required time                                 10.00
  -----------------------------------------------------------
  data required time                                 10.00
  data arrival time                                  -0.70
  -----------------------------------------------------------
  slack (MET)                                         9.30
```

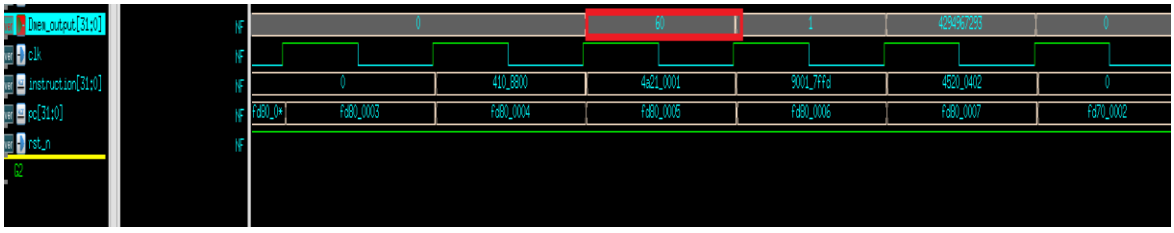
Area report:

```
****************************************
Report : area
Design : pipeline
Version: K-2015.06-SP1
Date   : Wed Jan 18 01:17:40 2017
****************************************

Library(s) Used:

    slow (File: /theda21_2/CBDK_IC_Contest/cur/SynopsysDC/db/slow.db)

Number of ports:                         1403
Number of nets:                          7228
Number of cells:                         5842
Number of combinational cells:           4434
Number of sequential cells:              1397
Number of macros/black boxes:               0
Number of buf/inv:                        865
Number of references:                      52

Combinational area:              46019.907791
Buf/Inv area:                     4812.128930
Noncombinational area:           43894.762604
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                 89914.670395
Total area:                 undefined

***** End Of Report *****
```

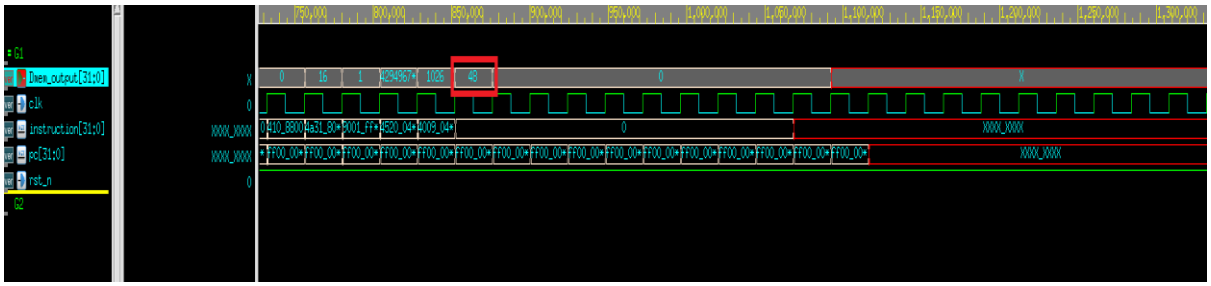
合成後的 gcd:



合成過後的 summation:



合成後的 mult:



合成後的 my_test:

合成問題:

一開始的問題是，當 RAM 還沒有 output 出來時，我的 instruction_next，就會先讀到值，所以會是 unknow，導致後面都讀不到值，而後面則是合成過後的 delay，會導致我讀值全部慢一個 cycle，後來我讓 clk 也 delay，就解決了這個問題。

完成部分;

Completion

1.	Four-stage pipeline	(完成)
2.	Data forwarding for data hazard	(完成)
3.	Branch prediction	(完成)
4.	My testcase design	(完成)
5.	Synthesis	(完成)

心得:

這次的 final project，讓我真正的了解到合成的奧妙所在，也是做得最久的一次，也讓我好好的去研究合成這個東西，及它會造成的效果，而最後看到結果也非常開心。

也謝謝助教和教授幫伍們 debug，最後祝助教和教授春節快樂。