這裡整理了一下這幾週寄信來問和TA hour來問的問題。

cpu.ys —> 拿來產生cpu_syn.v
我們拿 cpu_syn.v 喂test pattern進去，也就是我們會拿cpu_syn.v, instruction_memory.v, data_memory.v, testbench.v 去做模擬，也就是 "iverilog -D T1-T10 -f cpu_modified.f" 做的事情。

如果都對了，我們會再去做 "yosys -l cpu.yslog -q cpu_modified.ys" ，去看出來的log裡面的timing和area，來決定後面27%的分數。

所以可能跑出來的cpu_syn.v，cpu_modified.f 沒辦法過，所以才會要求大家交上cpu.f 和用到的module，做基本的模擬看正確性 (前20%)。

下面列出幾個來問的問題，下面是用簡單alu來表示：

(1) asynchronous active low reset

請同學要如下圖那樣，depend on我們給的 i_rst_n 去做reset，且要確定是negedge i_rst_n，因為我們reset時間點是在negedge前1/4個cycle，不然可能出現作業三模擬可以過，但是作業四出來的cpu_syn.v不會過。(右邊是我用的ys檔)

```
module test( // alu
    input i_clk,
    input i_rst_n,
    input [2:0] op,
    input [7:0] a,
    input [7:0] b,
    output [7:0] c
);
    always @(posedge i_clk or negedge i_rst_n) begin
        if (~i_rst_n) begin

        end else begin

        end
    end
endmodule
```

```
### Read verilog files
read_verilog test.v

### Constraints
write_file cpu.constr <<EOT
set_driving_cell BUF_X2
set_load 0.01
EOT

### Map to gate level
synth -top test; flatten;
write_verilog -noattr test_syn.v

### Map to tech library
dfflibmap -liberty stdcells.lib
abc -constr cpu.constr -D 1000 -liberty stdcells.lib
```

(2) DLATCH

combinational block (always @(*)) 裡面 if下有出現的reg，else也要有；case/default一樣。不然會合出DLATCH (else behavior也要定義，如果沒寫，default就會是上一個cycle的值，為了維持住上一個cycle的值，tool會合成出一個DLATCH來做這件事)，如下圖。遇到時就把 *_syn.v 檔打開來去看DLATCH出現在哪邊 (下圖是c_w)，去把else, default補滿。

```
reg [7:0] c_r, c_w;

assign c = c_r;

always @(*) begin
    case (op)
        3'b000: begin
            c_w = a+b;
        end
        3'b001: begin
            c_w = a-b;
        end
    endcase
end

always @(posedge i_clk or negedge i_rst_n) begin
    if (~i_rst_n) begin
        c_r <= 0;
    end else begin
        c_r <= c_w;
    end
end
```

```
always @(posedge i_clk or negedge i_rst_n)
  if (!i_rst_n)
    c_r[6] <= 0;
  else
    c_r[6] <= c_w[6];
always @(posedge i_clk or negedge i_rst_n)
  if (!i_rst_n)
    c_r[7] <= 0;
  else
    c_r[7] <= c_w[7];
\$_DLATCH_P_   _157_ (
  .D(_000_[0]),
  .E(_001_),
  .Q(c_w[0])
);
\$_DLATCH_P_   _158_ (
  .D(_000_[1]),
  .E(_001_),
  .Q(c_w[1])
);
\$_DLATCH_P_   _159_ (
  .D(_000_[2]),
  .E(_001_),
  .Q(c_w[2])
);
```

修完之後，就不會有DLATCH產生。那同學下圖可以看到tool自動幫我們把 c_w 變成wire，這是之前一直強調雖然是宣告reg，但其實功用是wire(為了寫if, else，要用always block寫，用always block寫就要用reg，但如果只是單純combinational的部分，就要用always @(*))，而c_r放在sequential block裡面tool就確實是合出reg。

```
reg [7:0] c_r, c_w;

assign c = c_r;

always @(*) begin
    case (op)
        3'b000: begin
            c_w = a+b;
        end
        3'b001: begin
            c_w = a-b;
        end
        default: begin
            c_w = 0;
        end
    endcase
end

always @(posedge i_clk or negedge i_rst_n) begin
    if (~i_rst_n) begin
        c_r <= 0;
    end else begin
        c_r <= c_w;
    end
end
```

```
input [7:0] a;
input [7:0] b;
output [7:0] c;
reg [7:0] c_r;
wire [7:0] c_w;
input i_clk;
input i_rst_n;
input [2:0] op;
```

(3) reg不能同時在sequential block和combinational block

這樣tool會視為是wire，而形成combinational loop，所以產生出來的 *_syn.v 會爛掉。所以大家要注意一下。那log檔也會告訴你大概是出現在哪邊。

```
always @(*) begin
    case (op)
        3'b000: begin
            c_w = a+b;
        end
        3'b001: begin
            c_w = a-b;
        end
        default: begin
            c_w = 0;
        end
    endcase
end

always @(posedge i_clk or negedge i_rst_n) begin
    if (~i_rst_n) begin
        c_r <= 0;
        c_w <= 0;
    end else begin
        c_r <= c_w;
    end
end
```

```
/* Generated by Yosys 0.9 (git sha1 UNKNOWN, clang

module test(i_clk, i_rst_n, op, a, b, c);
  input [7:0] a;
  input [7:0] b;
  output [7:0] c;
  wire [7:0] c_r;
  wire [7:0] c_w;
  input i_clk;
  input i_rst_n;
  input [2:0] op;
  assign c = 8'h00;
  assign c_r = 8'h00;
  assign c_w = 8'h00;
endmodule
```

```
Warning: Driver-driver conflict for \c_w [7]
stant.
Warning: Driver-driver conflict for \c_w [6]
stant.
Warning: Driver-driver conflict for \c_w [5]
stant.
Warning: Driver-driver conflict for \c_w [4]
stant.
Warning: Driver-driver conflict for \c_w [3]
stant.
Warning: Driver-driver conflict for \c_w [2]
stant.
Warning: Driver-driver conflict for \c_w [1]
stant.
Warning: Driver-driver conflict for \c_w [0]
stant.
```

(4) combinational loop

tool會告訴你有logic loop (如下圖)，所以請同學避免出現logic loop。

```
always @(*) begin
    case (op)
        3'b000: begin
            c_w = c_w+b;
        end
        3'b001: begin
            c_w = a-b;
        end
        default: begin
            c_w = 0;
        end
    endcase
end
```

```
Warning: found logic loop in module test:
    cell $add$test.v:17$2 ($add)
    cell $procmux$7 ($pmux)
    wire $add$test.v:17$2_Y [0]
    wire \c_w [0]
Warning: found logic loop in module test:
    cell $add$test.v:17$2 ($add)
    cell $procmux$7 ($pmux)
    wire $add$test.v:17$2_Y [1]
    wire \c_w [0]
Warning: found logic loop in module test:
    cell $add$test.v:17$2 ($add)
    cell $procmux$7 ($pmux)
    wire $add$test.v:17$2_Y [2]
    wire \c_w [0]
```

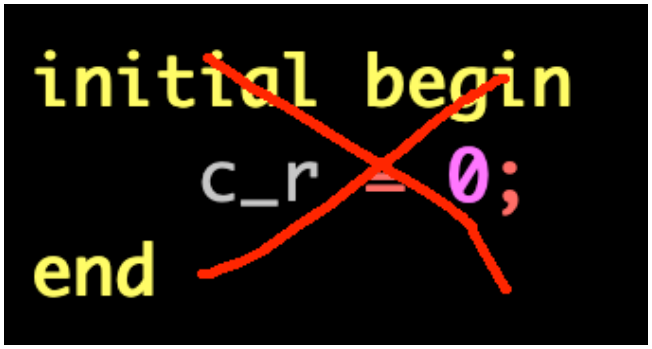
儘量把算出來的東西再用一個變數去接起，往前傳，如下圖。

```
reg [7:0] temp;

always @(*) begin
    case (op)
        3'b000: begin
            temp = a+b;
            c_w = temp;
        end
        3'b001: begin
            c_w = a-b;
        end
        default: begin
            c_w = 0;
        end
    endcase
end
```

(5) 不要有initial block 在 design裡面

initial block通常出現在testbench裡面，是不可合成的東西。所以要initialize一些值，請在active low reset那邊做。



再來是register_file那邊，同學可能不太會處理。
附圖補上大概是怎麼處理。

```
reg [DATA_W-1:0] registers   [31:0];
reg [DATA_W-1:0] registers_w [31:0];

always @(*) begin
    for (idx = 0; idx < 32; idx = idx + 1) begin
        registers_w[idx] = registers[idx];
    end
    if (i_RegWrite) begin
        registers_w[i_write_register] = i_write_data;
    end
end

always @(posedge i_clk or negedge i_rst_n) begin
    if (!i_rst_n) begin
        o_write_data1_r <= 0;
        o_write_data2_r <= 0;
        for (idx = 0; idx < 32; idx = idx + 1) begin
            registers[idx] <= 0;
        end
    end else begin
        o_write_data1_r <= o_write_data1_w;
        o_write_data2_r <= o_write_data2_w;
        for (idx = 0; idx < 32; idx = idx + 1) begin
            registers[idx] <= registers_w[idx];
        end
    end
end
```

可以看到combinational block我有if，但是沒有else。原因是 if 外面 (越外面) 的事情會先做，裡面的事情再去覆蓋掉外面定義的behavior，越裡面就是越後面才做的事。

所以如果有一堆nested if, else，signal又很多，可以在最外面先寫一次 “等於上一個cycle的behavior”，裡面每個if在各自定義需要的邏輯，就可以避免掉每個都要填的難事。(當然每個都填一定對)

這樣合成出來，上圖的register_w會都變成wire，然後64*32 bits的registers大概面積佔8000多

```
Warning: Replacing memory \registers with list of registers. See register_file.v:59
Warning: Replacing memory \registers_w with list of registers. See register_file.v:47
```

跑的時候，如果有出現上面的warning，可以忽略，他只是幫我們展開而已。

combinational block裡面，賦值一定要用"="，不然會合出DLATCH。習慣是sequential用"<="，combinational用"="。

最後是iverilog的地方，"iverilog -D T1-T10 -f cpu_modified.f" 的時候出現下圖那樣可以忽略，那是testbench多傳parameter給他，不會有影響。

```
testbench.v:149: warning: parameter ADDR_W not found in test_cpu.u_cpu.
testbench.v:148: warning: parameter DATA_W not found in test_cpu.u_cpu.
testbench.v:147: warning: parameter INST_W not found in test_cpu.u_cpu.
```

Report部分，大家注意一下。

沒有寫pipeline的，請想想要如何為這個structure設計，畢竟data memory, instruction memory的behavior和課本上的pipeline不一樣，所以hazard也不同。會視大家對hazard的處理，來評分。如果report有再想更多速度上面的優化，會再加進來評分。

問題的部分盡量回答，每個module的latency那題可以不寫，但critical path有做出來的就要寫。

critical path是在*.yslog檔裡面，可以截圖貼上就好。(如下圖)

```
ABC: Start-point = pi2 (\i_write_register [3]).  End-point = po448 (\registers_w[25] [0]).
ABC: + write_blif <abc-temp-dir>/output.blif
```