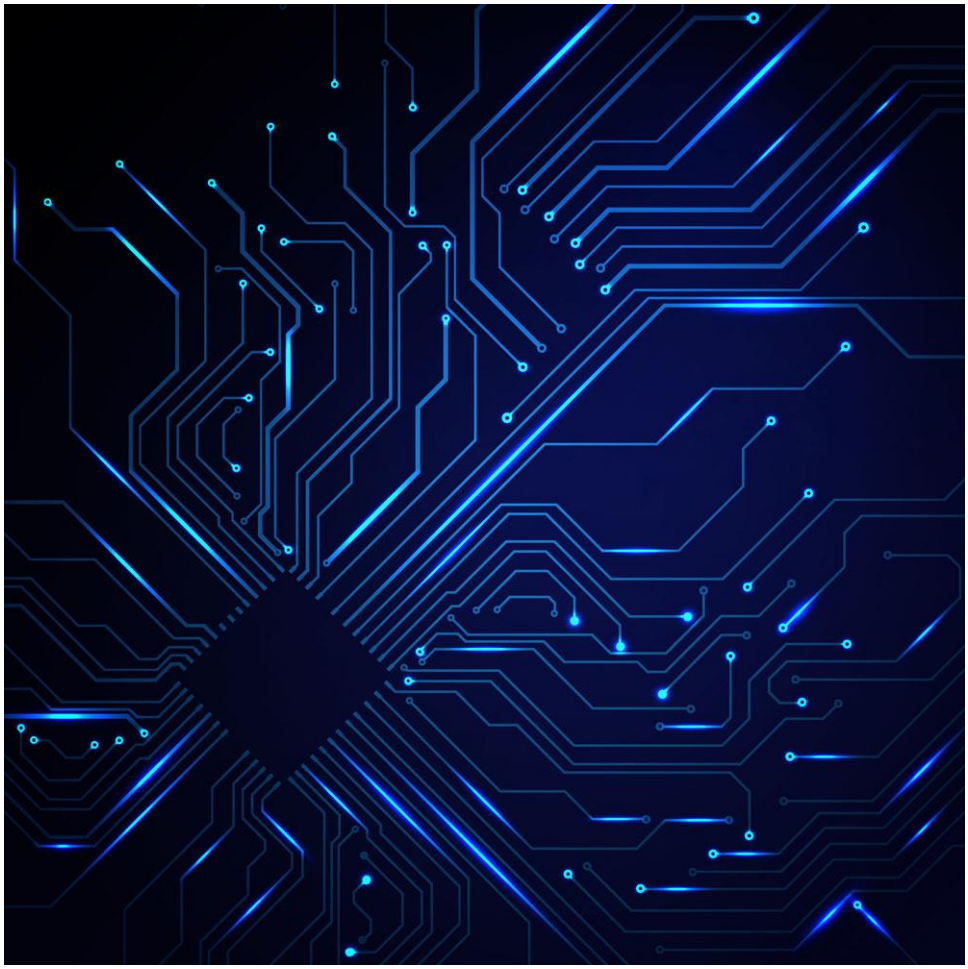


2023/03/22

實驗四

姓名：吳秉宸　　學號：00957202

班級：資工 4A

E-mail：evan20010126@gmail.com

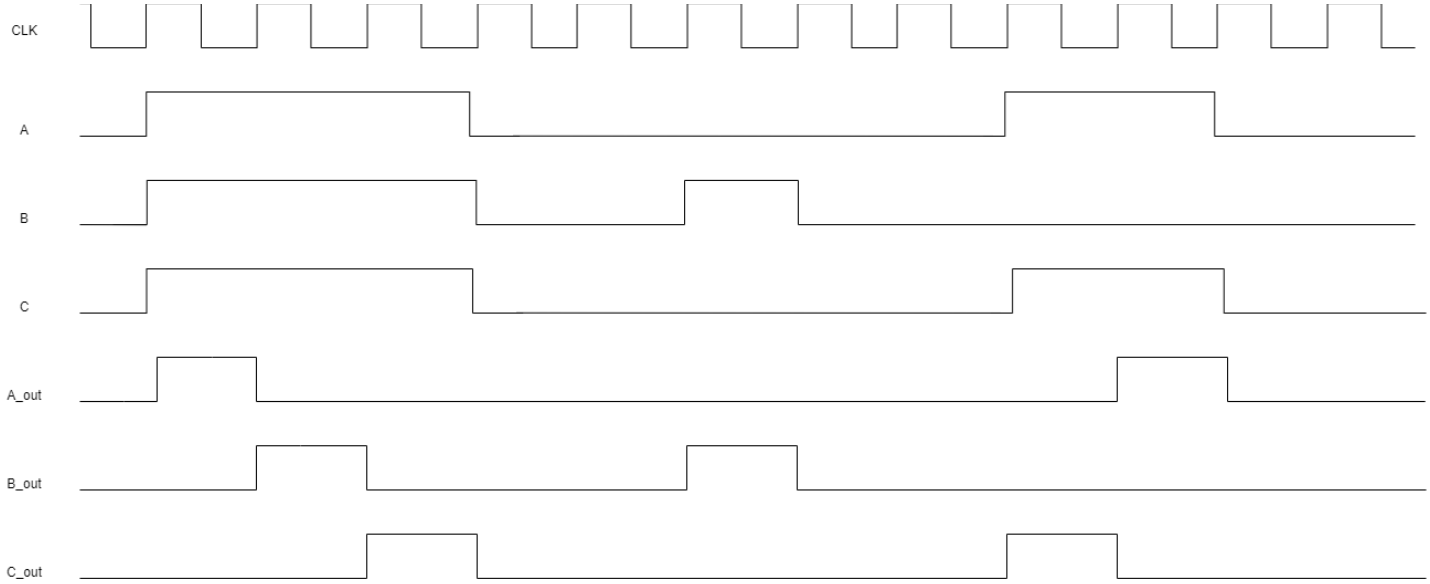
注意

1. 繳交時一律轉 PDF 檔
2. 繳交期限為
   隔週三上午九點
3. 一人繳交一份
4. 檔名：學號_HW?.pdf
   檔名請按照作業檔名格式進行填寫
   未依照格式不予批改

一、 仲裁

■ 實驗說明：

1. 除了 CLK 外有 3 個 1bit 輸入：A、B、C 和 3 個 1bit 輸出：A_out、B_out、C_out。
2. 當 A=1 時 A_out 輸出會 1 個 CLK 的 1（B、C 同理)，但同時只會有一個輸出為 1，所以必須考慮優先順序。
3. 預設的優先順序預設為 A => B => C。
   當 A 輸出完之後優先順序變為 B => C => A;
   當 B 輸出完之後優先順序變為 C => A => B;
   當 C 輸出完之後優先順序變為 A => B => C



注意 B_out 第二次輸出 1 的地方，因這時的狀態已經判斷到 B，所以下一個輸出是 C_out 而不是 A_out。

```
always #10 clk = ~clk;

initial begin
clk = 0;        rst = 0;
        #10 rst = 1;
        #20 a = 1;      b = 1;  c = 1;
        #60 a = 0;      b = 0;  c = 0;
        #60 a = 0;      b = 1;  c = 0;
        #20 a = 0;      b = 0;  c = 0;
        #60 a = 1;      b = 0;  c = 1;
        #40 a = 0;      b = 0;  c = 0;
        #1000 $stop;
end
```

■ 系統架構程式碼、測試資料程式碼與程式碼說明

截圖請善用 win+shift+S

Lab4_1.sv

```
module lab4_1 (
    input clk,
    input rst,
    input A,
    input B,
    input C,
    output logic A_out,
    output logic B_out,
    output logic C_out
);

typedef enum { T0, T1, T2 } fsm1_state_enum;

fsm1_state_enum fsm1_ns, fsm1_ps;

always_ff @(posedge clk) begin
    if (rst)
        fsm1_ps <= T0;
    else
        fsm1_ps <= fsm1_ns;
end

always_comb begin
    fsm1_ns = fsm1_ps;
    A_out = 0;
    B_out = 0;
    C_out = 0;
    case (fsm1_ps)
        /*
        T0 ~ T2 are different priorities.
        T0: A > B > C;
        T1: B > C > A;
        T2: C > A > B;
        */
        T0: begin
            if (A) begin
                A_out = 1;
                fsm1_ns = T1;
            end else if (B) begin
                B_out = 1;
                fsm1_ns = T2;
            end else if (C) begin
                C_out = 1;
                fsm1_ns = T0;
```

```
            end
        end

        T1: begin
            if (B) begin
                B_out = 1;
                fsm1_ns = T2;
            end else if (C) begin
                C_out = 1;
                fsm1_ns = T0;
            end else if (A) begin
                A_out = 1;
                fsm1_ns = T1;
            end
        end
        T2: begin
            if (C) begin
                C_out = 1;
                fsm1_ns = T0;
            end else if (A) begin
                A_out = 1;
                fsm1_ns = T1;
            end else if (B) begin
                B_out = 1;
                fsm1_ns = T2;
            end
        end

    endcase
end

endmodule
```

依照題目要求宣告 input、output 接角，之後定義三個 state，T0、T1、T2，為不同優先序的狀態，T0 為 A->B->C；T1 為 B->C->A；T2 為 C->A->B，並在 fsm 中以 if elseif else 優先序的概念實作。且宣告 flip flop 以 clk 控制 state。

compile.do

```
#vlib work



#
----------------------------------------
--------------------
vlog ../tb/testbench.sv
vlog ../../design/lab4_1.sv
vlog ../../design/sub_4bit.sv
vlog ../../design/DE0_CV.sv

```

編譯 lab3_3.sv 與 testbench.sv 檔案。

Wave.do

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -divider {TOP LEVEL INPUTS}
add wave -noupdate -divider cpu
add wave -noupdate /testbench/clk
add wave -noupdate /testbench/rst
add wave -noupdate /testbench/a
add wave -noupdate /testbench/b
add wave -noupdate /testbench/c
add wave -noupdate /testbench/A_out
add wave -noupdate /testbench/B_out
add wave -noupdate /testbench/C_out
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {42 ps} 0}
quietly wave cursor active 1
configure wave -namecolwidth 150
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ns
update
WaveRestoreZoom {0 ps} {626 ps}

```

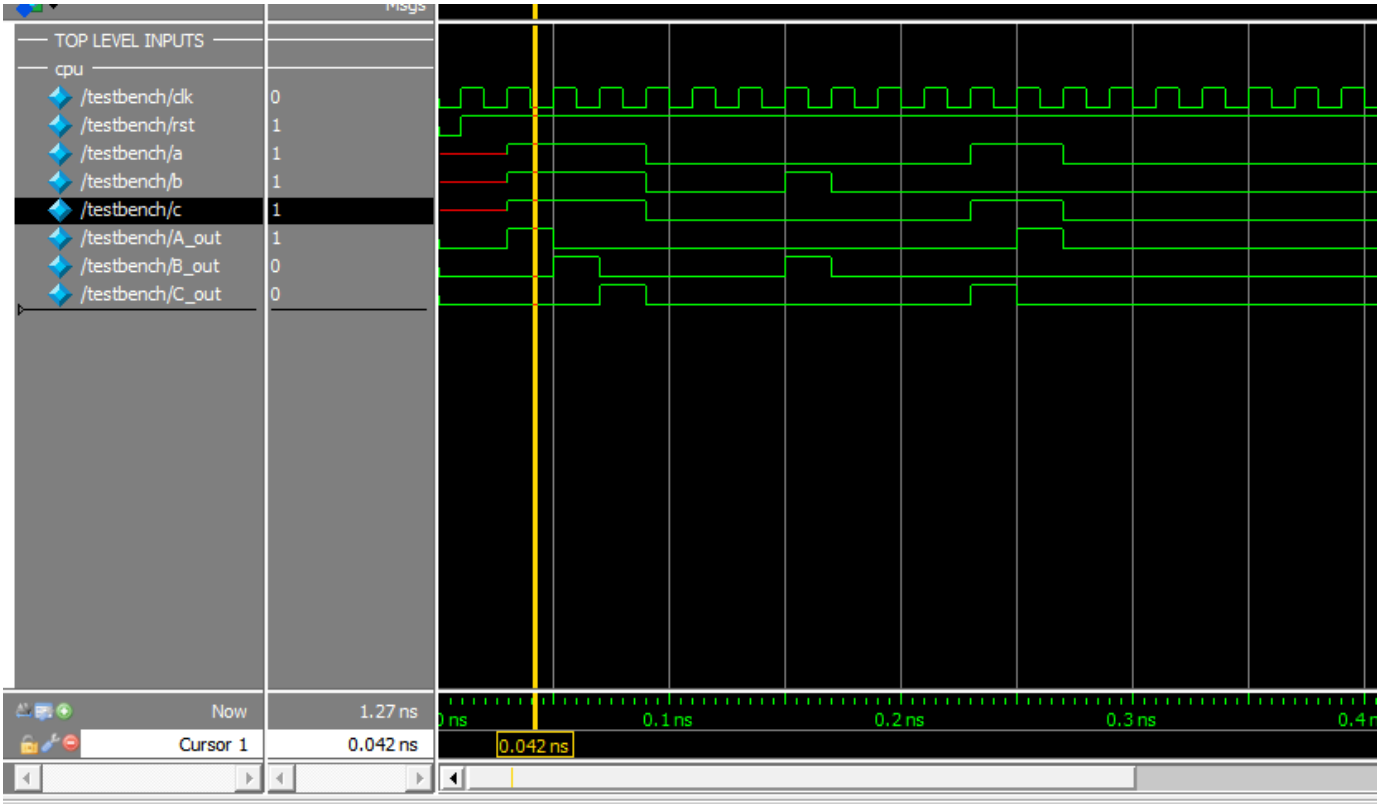
將 clk、rst、A、B、C、A_out、B_out、C_out 的接角輸出。

Testbench.sv

```
module testbench;

    logic clk;
    logic rst;
    logic a;
    logic b;
    logic c;
    logic A_out;
    logic B_out;
    logic C_out;

    lab4_1 u1(
        .clk(clk),
        .rst(~rst),
        .A(a),
        .B(b),
        .C(c),
        .A_out(A_out),
        .B_out(B_out),
        .C_out(C_out)
    );

    always #10 clk = ~clk;

    initial begin
        rst = 0; clk = 0;
        #10 rst = 1;
        #20 a = 1; b = 1; c = 1;
        #60 a = 0; b = 0; c = 0;
        #60 a = 0; b = 1; c = 0;
        #20 a = 0; b = 0; c = 0;
        #60 a = 1; b = 0; c = 1;
        #40 a = 0; b = 0; c = 0;
        #1000 $stop;
    end
endmodule
```

每 20 單位時間為 clk 的一個週期，依照題目將 A、B、C 賦值。

■ 模擬結果與結果說明：



由上圖可看出輸出波形與 LAB 中給的波型相同。

二、 Encoder

- **實驗說明:**
- **系統架構程式碼與程式碼說明**

  1. 以 50MHz 當作系統時脈
  2. 輸入信號：synr_enc，為輸入待測信號（encoder）
  3. 輸出數值：Encoder 週期的計數（用 50MHz 計數）

  PS. 運用 LPF,Edge detector, counter, FSM 電路設計上述電路

- **系統架構程式碼與程式碼說明**

  Low_Pass_Filter_4ENC.sv

```
// Designer Mao-Hsu Yen
// 0.
// +FHDR----------------------------------------------------------------------
// Copyright (c) 2021 XXXXX, Inc. All rights reserved
// ANSER Confidential Proprietary
// ---------------------------------------------------------------------------
// FILE NAME : Low_Pass_Filter.v
// DEPARTMENT : CSE NTOU
// AUTHOR : Mao-Hsu Yen
// AUTHOR'S EMAIL : ymh@mail.ntou.edu.tw
// ---------------------------------------------------------------------------
// RELEASE HISTORY
// VERSION DATE AUTHOR DESCRIPTION
// 1.0 2021-06-06 initial version
// 2.0
// ---------------------------------------------------------------------------
// KEYWORDS :
// ---------------------------------------------------------------------------
// PURPOSE :
//
// ---------------------------------------------------------------------------
// PARAMETERS
//
//  Low_Pass_Filter_4ENC
//  r_LPF_threshold_enc[9:0]    :
//      1. If (r_LPF_threshold_enc[9:0] > 0)  LPF_threshold = r_LPF_threshold_enc X 0.0124 ms
//      2. If (r_LPF_threshold_enc[9:0] = 0x000, it should be in Bypass.
//
//
// ---------------------------------------------------------------------------
// REUSE ISSUES
// Reset Strategy : Asynchronous, active high system level reset
```

```
// Clock Domains : clk_sys(or clk)
// Critical Timing : N/A
// Test Features : N/A
// Asynchronous I/F :
// Scan Methodology : N/A
// Instantiations :
// Synthesizable : Y
// Other :
// -FHDR----------------------------------------------------------------------


//// for encoder
module Low_Pass_Filter_4ENC
(
    output logic sig_filter,
    input signal,
    input [13:0] r_LPF_threshold_enc,  //   Unit : 0.08us  /// 2^3 =
8,  r_LPF_threshold_enc=0 => By Pass
    input clk,
    input reset
);
//// ---------------- --------------
//  counter[12:0]
//  clk =100MHZ 時   cycle time = 1/100M = 0.01 us
//  LPF_threshold:
//              1. 我們使用 counter[12:0] 的 3 bits counter[2:0] 當成 單位 LPF_threshold 為
0.01us x 8 = 0.08 us
//              2. 也就是說低於  LPF_threshold = r_LPF_threshold_enc X 0.08 us  的訊號變化都會
被濾掉
//              3. 注意：是信號 high or low 維持時間需 > LPF_threshold 才不會被濾掉。
//
//  r_LPF_threshold_enc[9:0]:
//      1. If (r_LPF_threshold_enc[9:0] > 0)  LPF_threshold = r_LPF_threshold_enc X 0.01024
ms
//      2. If (r_LPF_threshold_enc[9:0] = 0x000, it should be in Bypass.
//
//
//
//// ---------------- internal constants --------------
    parameter N = 13 ;

    logic [N-1 : 0] counter;                            // timing regs
    logic reset_counter;
    logic LPF_threshold;
    //assign counter
    logic [4:0] q;
    always @(posedge clk)
        begin
            if(~reset)
                begin
```

```
                    q               <= 2'b0;
                    sig_filter      <= signal;
                    //sig_filter        <= 0;
                    reset_counter   <= 0;
                    LPF_threshold   <= 0;
                end
            else
                begin
                    q[4:0]              <= {q[3:0], signal};
                    if (LPF_threshold)  sig_filter <= q[4];
                    reset_counter       <= q[1]^q[0];
                    //LPF_threshold         <= (counter[N-1: 4] >= r_LPF_threshold_enc);
                    LPF_threshold       <= (counter[N-1: 0] >= r_LPF_threshold_enc);
                end
        end

    always @(posedge clk)
        begin
            if(~reset | reset_counter)
                counter <= 0;
            else
                if (~counter[N-1]) counter <= counter + 1;
        end


endmodule
```

使用老師提供的範例程式碼來完成 Low_Pass_filter，主要是利用 counter 計數來取樣，最終輸出沒有雜訊的訊號。

Edge_detector.sv

```
module edge_detector(
    input logic rst,
    input logic clk,
    input logic enc_filter,
    output logic enc_pos
);
    logic s_signal, d_signal;
    // posedge of Encoder
    always_ff @(posedge clk)
    begin
        if (~rst) begin
            s_signal <= 1'b1;
            d_signal <= 1'b1;
            enc_pos <= 1'b0;
        end else begin
            {d_signal, s_signal} <= {s_signal, enc_filter}; // enc_filter 是 low pass 過的信號
(i.e 最新的信號)
            enc_pos <= s_signal & ~d_signal;
        end
    end
endmodule
```

edge_detector 也是經由取樣來看，要不要輸出一個 pulse。

Encoder.sv

```
module Encoder(
    input logic rst,
    input logic clk,
    input logic enc // signal
    // output logic [3:0]s
    );

    logic enc_filter;
    logic enc_pos;
    logic [15:0] cnt;
    logic [15:0] cnt_meas;
    logic enable;
    logic load;

    Low_Pass_Filter_4ENC u1
    (
        .sig_filter(enc_filter),
        .signal(enc),
        .r_LPF_threshold_enc(14'd200),  //  Unit : 0.08us  /// 2^3 =
8,  r_LPF_threshold_enc=0 => By Pass
        .clk(clk),
        .reset(rst)
```

```
);

edge_detector u2(
    .rst(rst),
    .clk(clk),
    .enc_filter(enc_filter),
    .enc_pos(enc_pos) // 產生一個 pulse
);

// counter
always_comb
begin
    if (~rst) cnt <= 0;
    else if (enable) cnt <= cnt + 1;
end

// DFF
always_ff @( posedge clk )
begin
    if (~rst) cnt_meas <= 0;
    else if (load) cnt_meas <= cnt;
end



// FSM
// typedef enum { T0, T1, T2, T3, T4, T5, T6 } fsm_state_enum;
// fsm_state_enum ns, ps;

// always_ff @(posedge clk) begin
//     if (~rst)
//         ps <= T0;
//     else
//         ps <= ns;
// end

always_comb
// begin
//     enable = 0;
//     load = 0;
//     case (ps)
//         T0: begin
//             ns = T1;
//         end

//         T1: begin
if (enc_pos == 1) begin
    enable = 1;
end else begin
    load = 1;
```

```
    end
    // ns = T1;
    //     end
    // endcase
    // end

endmodule
```

encoder 中，先是利用 LPF 將雜訊去除，再利用 edge_detector 將輸入信號為 1 轉成一個 pulse，再控制 enable、load 信號，來控制 counter 的計數與 load 進 D-Flip-Flop。

Testbench.sv

```
module testbench;

    logic clk;
    logic rst;
    logic enc;


    Encoder test(
        .rst(rst),
        .clk(clk),
        .enc(enc)
    );


    always #10 clk = ~clk;

    initial begin
        rst = 0; clk = 0;
        #15 rst = 1;

        #100 enc = 0; // noise
        #100 enc = 1;
        #100 enc = 0;
        #100 enc = 1;
        #100 enc = 0;
        #100 enc = 1;
        #100 enc = 0;
        #10000 enc = 0; //signal

        #100 enc = 0; // noise
        #100 enc = 1;
        #100 enc = 0;
        #100 enc = 1;
        #100 enc = 0;
```

```
        #100 enc = 1;
        #100 enc = 0;
        #100 enc = 1;

        #10000 enc = 1; //signal

        #100 enc = 0; // noise
        #100 enc = 1;
        #100 enc = 0;
        #100 enc = 1;
        #100 enc = 0;
        #100 enc = 1;
        #100 enc = 0;

        #10000 enc = 0; //signal

        #100 enc = 0; // noise
        #100 enc = 1;
        #100 enc = 0;
        #100 enc = 1;
        #100 enc = 0;
        #100 enc = 1;
        #100 enc = 0;
        #100 enc = 1;

        #10000 enc = 1; //signal



        #1000 $stop;
    end
endmodule
```

在 testbench 中有製造一些雜訊與一些真正的 signal，重複幾次，模擬真實情況。

Compile.do

```
#vlib work



# ----------------------------------------------------------------
vlog ../tb/testbench.sv
vlog ../../design/sub_4bit.sv
vlog ../../design/DE0_CV.sv
vlog ../../design/Encoder.sv
vlog ../../design/edge_detector.sv
vlog ../../design/Low_Pass_Filter_4ENC.sv


```

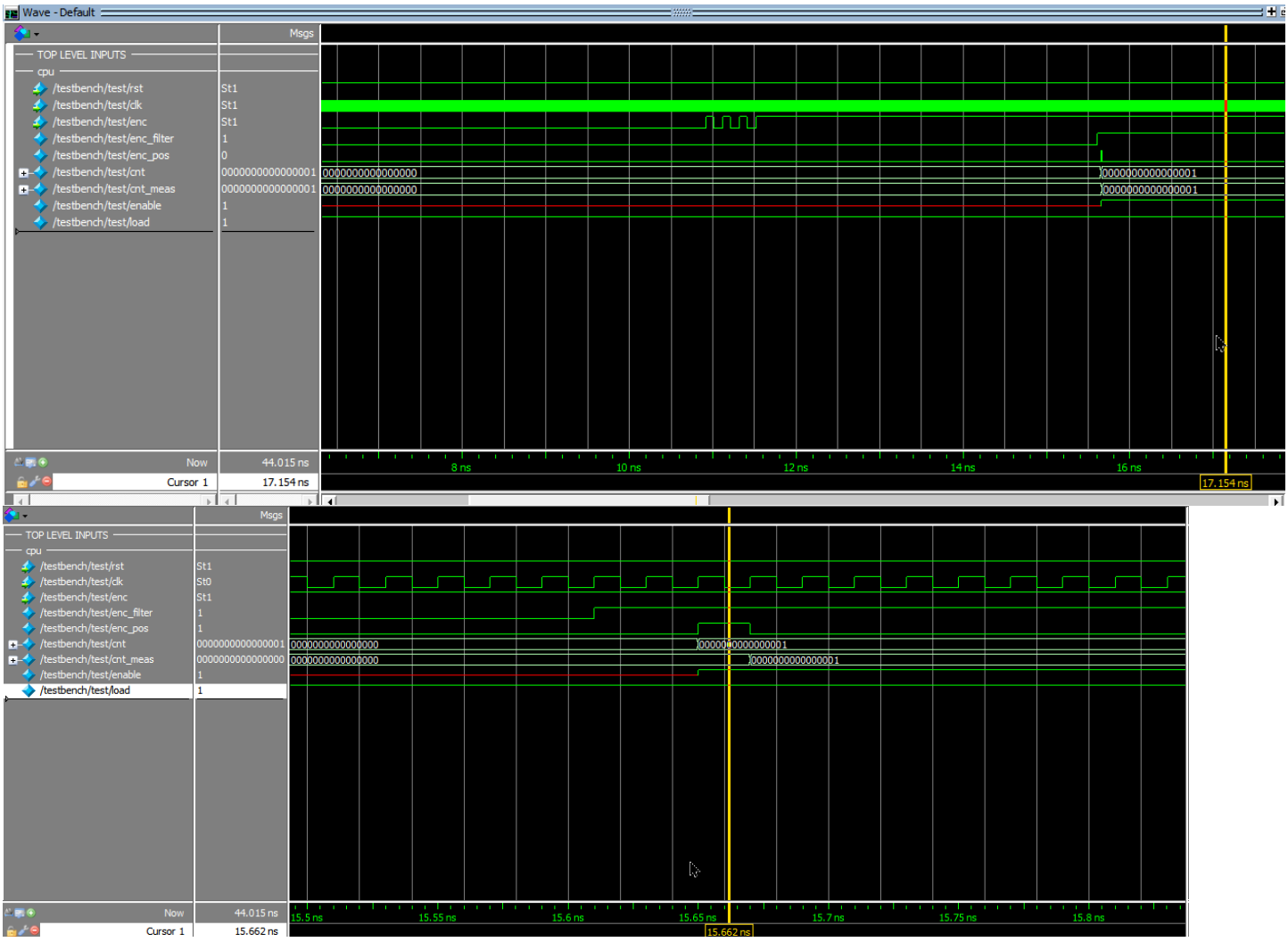
Compile 所需要使用的檔案。

Wave.do

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -divider {TOP LEVEL INPUTS}
add wave -noupdate -divider cpu
add wave -noupdate /testbench/test/rst
add wave -noupdate /testbench/test/clk
add wave -noupdate /testbench/test/enc
add wave -noupdate /testbench/test/enc_filter
add wave -noupdate /testbench/test/enc_pos
add wave -noupdate /testbench/test/cnt
add wave -noupdate /testbench/test/cnt_meas
add wave -noupdate /testbench/test/enable
add wave -noupdate /testbench/test/load
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {8403 ps} 0}
quietly wave cursor active 1
configure wave -namecolwidth 207
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ns
update
WaveRestoreZoom {6307 ps} {17861 ps}

```

輸出所需要的信號線，如：rst, clk, enc(輸入信號), enc_filter, enc_pos, , cnt, cnt_meas。

■ 模擬結果與結果說明：



由第一張波型圖中可以發現，原本 enc 的雜訊被 Low pass filter 消除。

由第二張波型圖中可以發現，cnt 在 en_pos 是 1 時開始計數，0 時則 load 進 cnt_meas。

■ 結論與心得：

這次的實驗第一個作業老師有講解過三種不同的 arbiter，這是其中的輪詢，因為上課有講解過，很快的就完成第一題的作業；在第二個作業設計控制 enable 與 load 信號的 FSM 有點卡卡的，不太知道怎麼下手，希望下次能快速的完成 LAB。