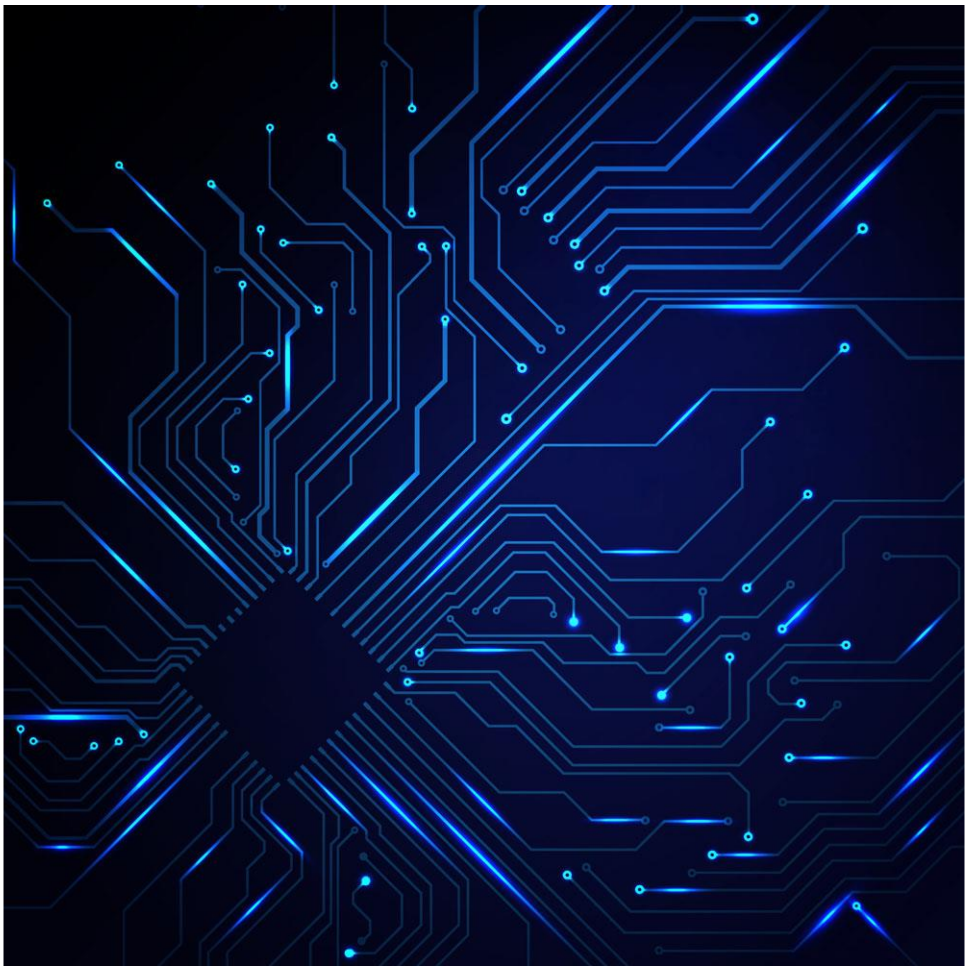


2023/03/29

實驗五

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

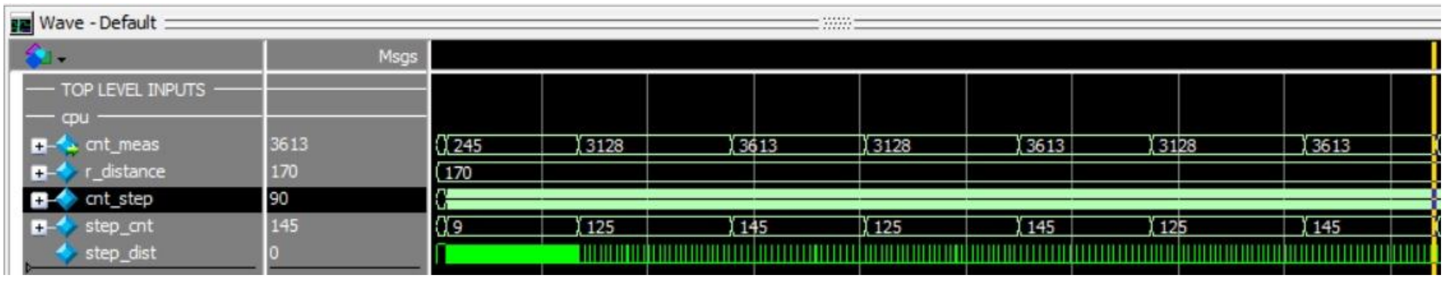
一、 Encoder_2

■ 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，在 600dpi 下量測 1.7μm。
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、step_cnt、step_dist。

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

DE0_CV.sv

```
`timescale 1ns/100ps

//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module DE0_CV(
    //////////// CLOCK //////////
    input                           CLOCK_50,
    input                           CLOCK2_50,
    input                           CLOCK3_50,
    inout                           CLOCK4_50,

    //////////// SDRAM //////////
    // output           [12:0]      DRAM_ADDR,
    // output            [1:0]      DRAM_BA,
    // output                       DRAM_CAS_N,
    // output                       DRAM_CKE,
    // output                       DRAM_CLK,
    // output                       DRAM_CS_N,
    // inout            [15:0]      DRAM_DQ,
    // output                       DRAM_LDQM,
    // output                       DRAM_RAS_N,
    // output                       DRAM_UDQM,
    // output                       DRAM_WE_N,
    //////////// SEG7 //////////
    output          [6:0]      HEX0,
    output          [6:0]      HEX1,
    output          [6:0]      HEX2,
    output          [6:0]      HEX3,
    output          [6:0]      HEX4,
    output          [6:0]      HEX5,

    //////////// KEY //////////
    input           [3:0]      KEY,
    input                      RESET_N,

    //////////// LED //////////
    output          [9:0]      LEDR,

    //////////// PS2 //////////
    // inout                       PS2_CLK,
    // inout                       PS2_CLK2,
    // inout                       PS2_DAT,
    // inout                       PS2_DAT2,

    // //////////// microSD Card //////////
    // output                      SD_CLK,
    // inout                       SD_CMD,
    // inout        input            [9:0]      SW
    //////////// SW //////////
    input           [9:0]      SW

    //////////// VGA //////////
    // output           [3:0]      VGA_B,
    // output           [3:0]      VGA_G,
    // output                      VGA_HS,
    // output           [3:0]      VGA_R,
    // output                      VGA_VS,

    //////////// GPIO_0, GPIO_0 connect to GPIO Default //////////
    // inout           [35:0]      GPIO_0,

    //////////// GPIO_1, GPIO_1 connect to GPIO Default //////////
    // inout           [35:0]      GPIO_1
);



//=======================================================
//  REG/WIRE declarations
//=======================================================
//=========================================================
//  Structural coding
//=========================================================

logic [13:0] r_LPF_threshold;
logic [13:0] r_distance;
logic reset_n;

logic rx;
logic tx_data;
logic tx_end_flag;

logic step_col; // output

assign clk = CLOCK_50;
assign reset_n = RESET_N;
assign enc = SW[0]; // Actually 從 GPIO 進來
assign r_LPF_threshold = 14'd20;
assign r_distance = 14'd170; //1.70um (待測的距離)
// assign buad_setting = 2'b00;

Encoder ens_sys(
    .clk(clk),
    .rst(reset_n),
    .r_LPF_threshold(r_LPF_threshold), // 濾波門檻值
    .r_distance(r_distance),
    .enc (enc),
    .step_col(step_col) // 每經過 r_distance 就通知印表機印一條線
);




endmodule
```

呼叫主程式 Encoder 模組，並套入對應變數。

Encoder.sv

```
`timescale 1ns/100ps

module Encoder(
    input logic clk,
    input logic rst,
    input logic [13:0] r_LPF_threshold,
    input logic [13:0] r_distance,
    input logic enc, // signal (有雜訊的)

    output logic step_col
    // 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(r_LPF_threshold),  //  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
    logic rst_cnt;
    always_ff @( posedge clk )
    begin
        if (~rst | rst_cnt) 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


    logic cnt_meas_rdy;

    // 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) begin
            ps <= T0;
            cnt_meas_rdy <= 0;
          end
        else begin
            ps <= ns;
            cnt_meas_rdy <= load;
        end
    end

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

            T1: begin
                if (enc_pos) ns = T2; // T2: load
                else enable = 1;
            end

            T2: begin
               load = 1;
               rst_cnt = 1;
               ns = T1;
            end
        endcase
    end

    gen_step_col gen_step_col_1(
        .clk (clk),
        .rst (rst),
        .cnt_meas_rdy (cnt_meas_rdy), // cnt_meas_rdy 通知說 cnt_meas 是可以用的值((因為初始化時
cnt_meas 是亂的
        .cnt_meas (cnt_meas), //真正的 cnt 值放這裡
        .r_distance (r_distance),
        .step_col(step_col)
    );

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(cnt_meas)，再當計數到 r_distance 的距離後，利用 gen_step_col 來產生 step_col 信號。

Low_Pass_Filter_4ENC.sv

```
`timescale 1ns/100ps

// 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); // 大於
r_LPF_threshold_enc 才認為穩定
```

```
                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

```
`timescale 1ns/100ps

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
```

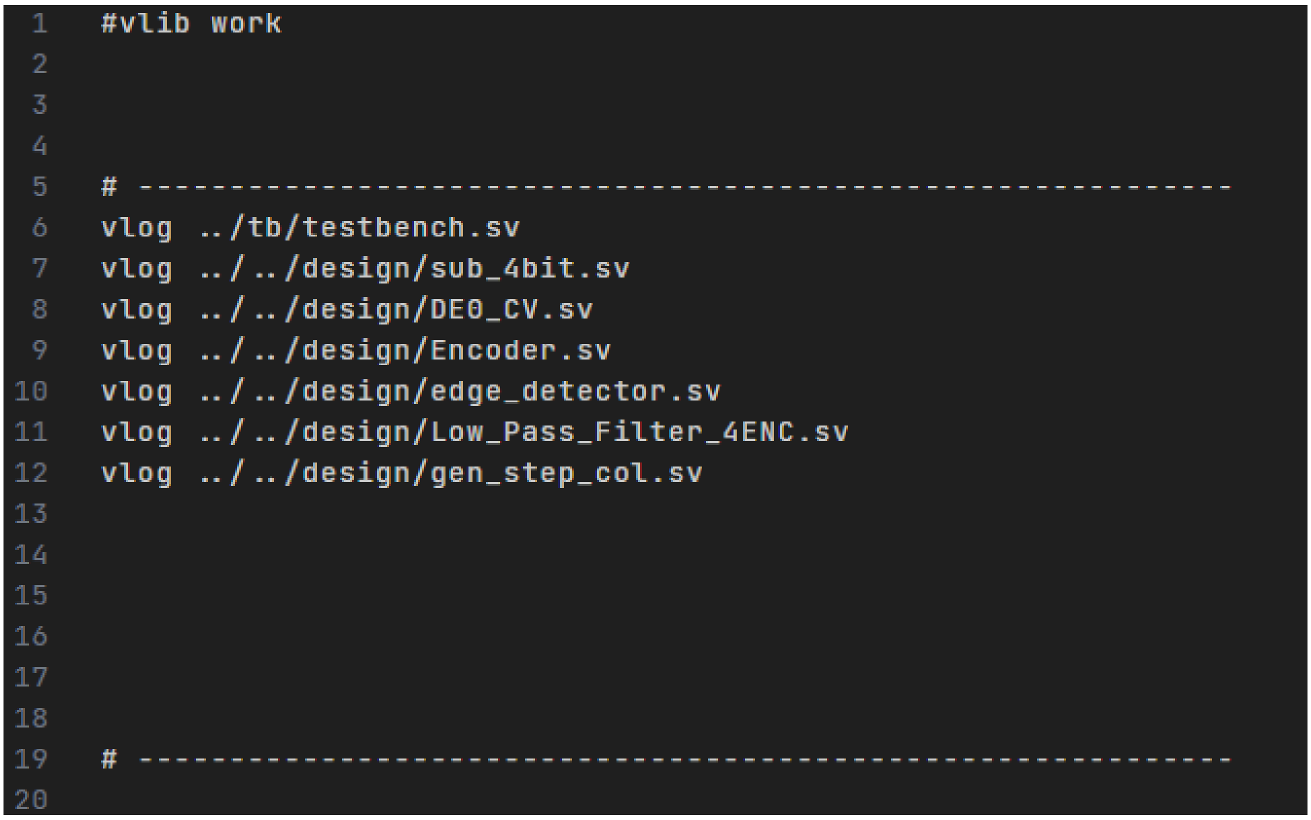
edge_detector 也是經由取樣來看，要不要輸出一個 pulse。

gen_step_col.sv

```
`timescale 1ns/100ps

module gen_step_col(
    input logic clk,
    input logic rst,
    input logic cnt_meas_rdy, // cnt_meas_rdy 通知說 cnt_meas 是可以用的值((因為初始化時 cnt_meas
是亂的
    input logic [15:0] cnt_meas, //真正的 cnt 值放這裡
    input logic [13:0] r_distance,
    output logic step_col
);
    logic [31:0] mul_result;
    logic [31:0] cnt;
    logic cp1;
    logic load_cnt_1;

    always_ff @(posedge clk) begin
        if (!rst)
            mul_result <= 'b0;
        else
            mul_result <= cnt_meas * r_distance / 4233;
    end

    // cnt1
    always_ff @(posedge clk) begin
        if (load_cnt_1)
            cnt <= mul_result;
        else if (cnt)
            cnt <= cnt - 1'b1;
    end

    typedef enum { START, CHK_MEAS_RDY_1, CHK_MEAS_RDY_2, LOAD_CNT, GEN_STEP_C }
fsm_state_enum;
    fsm_state_enum ps, ns;

    // FSM1
    always_ff @(posedge clk) begin
        if (!rst)
            ps <= START;
        else
            ps <= ns;
    end

    always_comb begin
        cp1 = 0;
        load_cnt_1 = 0;
        step_col = 0;
        ns = ps;
        unique case(ps)
            START: begin
                load_cnt_1 = 1;
                ns = CHK_MEAS_RDY_1;
            end

            CHK_MEAS_RDY_1: begin
                if (cnt_meas_rdy)
                    ns = CHK_MEAS_RDY_2;
            end

            CHK_MEAS_RDY_2: begin
                if (cnt_meas_rdy)
                    ns = LOAD_CNT;
            end

            LOAD_CNT: begin
                load_cnt_1 = 1;
                ns = GEN_STEP_C;
            end

            GEN_STEP_C: begin
                if (~|cnt) begin
                    load_cnt_1  = 1;
                    step_col = 1;
                end
                ns = GEN_STEP_C;
            end
        endcase
    end
endmodule
```

先計算出 mul_result 的值(為一個 r_distance 的距離所需要的 cnt 數)，當 cnt_meas_rdy 時將 mul_result 存入 cnt 中計數，當計數到 0 時即為 r_distance 的距離。

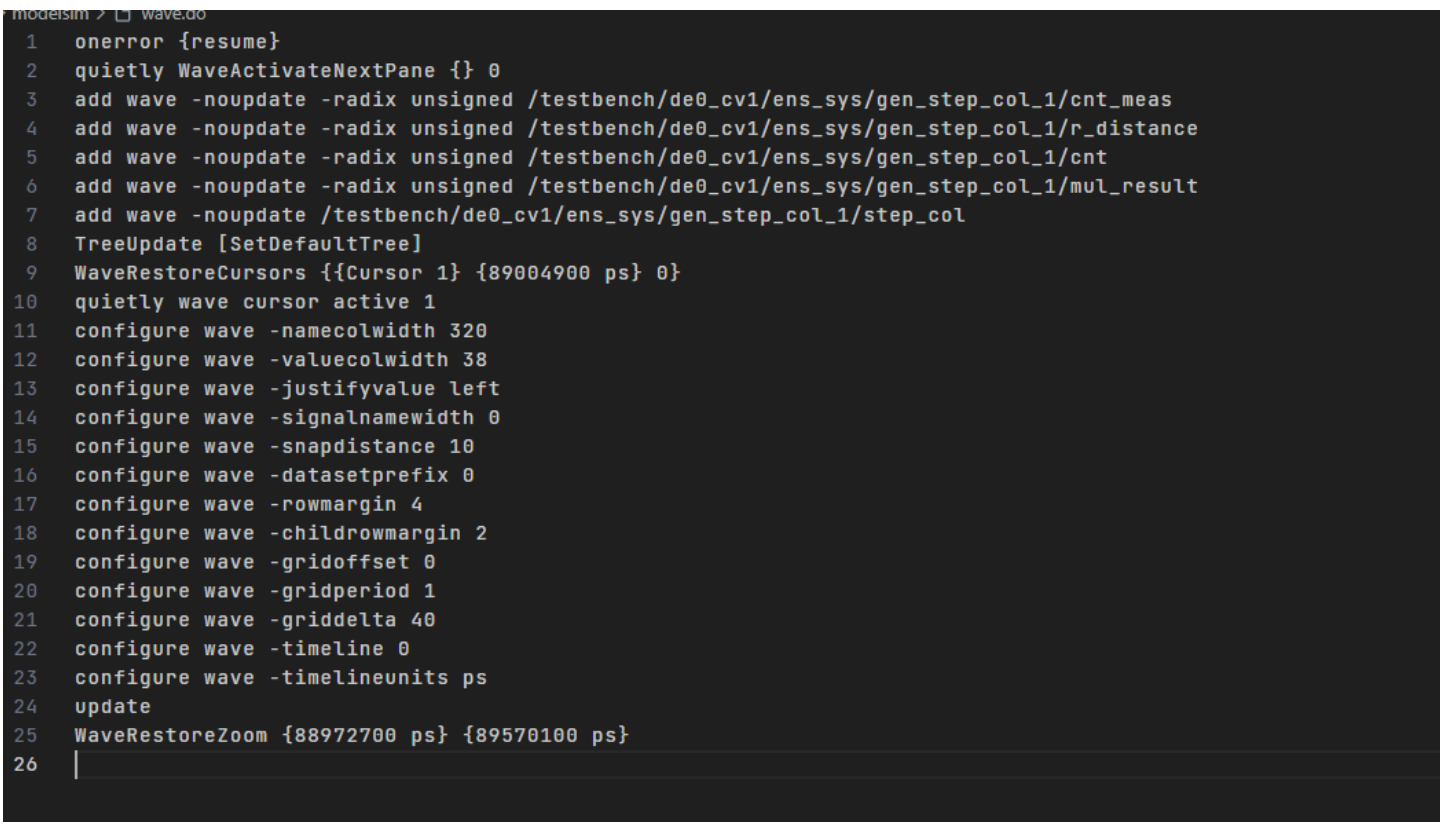
compile.do



```
#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
vlog ../../design/gen_step_col.sv






# ------------------------------------------------------------

```

Compile 所需要使用的檔案。

Wave.do



```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -radix unsigned /testbench/de0_cv1/ens_sys/gen_step_col_1/cnt_meas
add wave -noupdate -radix unsigned /testbench/de0_cv1/ens_sys/gen_step_col_1/r_distance
add wave -noupdate -radix unsigned /testbench/de0_cv1/ens_sys/gen_step_col_1/cnt
add wave -noupdate -radix unsigned /testbench/de0_cv1/ens_sys/gen_step_col_1/mul_result
add wave -noupdate /testbench/de0_cv1/ens_sys/gen_step_col_1/step_col
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {89004900 ps} 0}
quietly wave cursor active 1
configure wave -namecolwidth 320
configure wave -valuecolwidth 38
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 ps
update
WaveRestoreZoom {88972700 ps} {89570100 ps}

```

輸出所需要的信號線，如：r_distane、cnt、mul_result、step_col，個信號的意義在” 模擬結果與結果說明”中詳述對應的參數名與意義。

Testbench.sv

```
`timescale 1ns/100ps
module testbench;
    logic [15:0]    a;
    logic [6:0]     HEX0;
    logic [6:0]     HEX1;
    logic [6:0]     HEX2;
    logic [6:0]     HEX3;
    logic [6:0]     HEX4;
    logic [6:0]     HEX5;
    logic [3:0]     KEY;
    logic [9:0]     SW;
    logic [9:0]     LEDR;

    logic CLOCK_50;
    logic rst_n;

DE0_CV de0_cv1(
    // .a(a),
    //////////// CLOCK //////////
    .CLOCK_50   (CLOCK_50),
    .CLOCK2_50  (),
    .CLOCK3_50  (),
    .CLOCK4_50  (),

    //////////// SEG7 //////////
    .HEX0(),
    .HEX1(),
    .HEX2(),
    .HEX3(),
    .HEX4(),
    .HEX5(),

    //////////// KEY //////////
    .KEY        (KEY),
    .RESET_N    (rst_n),

    //////////// LED //////////
    .LEDR       (LEDR),



    //////////// SW //////////
    .SW         (SW)
);

    //assign KEY[0] = rst_n;
    logic enc;

    logic [15:0]r_distance;
```

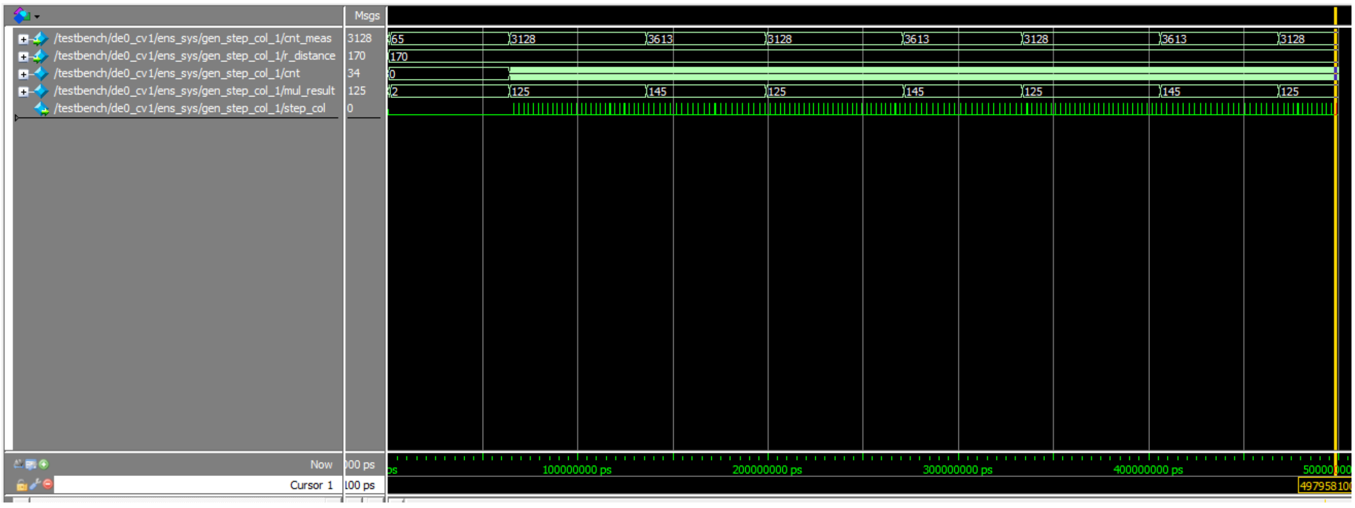
```
always #10 CLOCK_50 = ~CLOCK_50;

always begin
    //3128//
    #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 = 0;     // signal

    #100    enc = 0;     //noise
    #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;

    #20000  enc = 0;     // signal

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

    #20000  enc = 1;
    //3128//

    //3613//
    #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 = 0;    // signal

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

        #9000   enc = 1;    // signal

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

        #10100  enc = 0;    // signal

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

        #20000  enc = 0;    // signal

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

        #20000  enc = 1;
        //3613//

    end

    assign SW[0] = enc;

    assign a = r_distance;

    initial begin
```

```
        CLOCK_50 = 0;
        rst_n = 0;
        enc = 0;
        r_distance = 170;
        #20 rst_n = 1;;
        #500000 $stop;
    end


endmodule
```

此為老師提供的 testbench.sv，運用此 testbench.sv 模擬有雜訊的 encoder 信號。

■ 模擬結果與結果說明：



```
                                                   Msgs
/testbench/de0_cv1/ens_sys/gen_step_col_1/cnt_meas    3128
/testbench/de0_cv1/ens_sys/gen_step_col_1/r_distance  170
/testbench/de0_cv1/ens_sys/gen_step_col_1/cnt         34
/testbench/de0_cv1/ens_sys/gen_step_col_1/mul_result  125
/testbench/de0_cv1/ens_sys/gen_step_col_1/step_col    0
```

模擬結果如上圖，且對照老師的變數命名如下：

cnt_means	->cnt_meas: 代表 encoder 產生一個 pulse 的時間內 cnt 數了 cnt_meas 次
r_distane	->r_distance: 代表需要量測的距離
cnt	->cnt_step: 計數(從 step_cnt 數到 0)
mul_result	->step_cnt: cnt 應數到多少才是 r_distance 的距離
step_col	->step_dist: 當 cnt 數到 r_distance 的距離時輸出 1

■ 結論與心得：

本次的 LAB 延續了上次的專案內容，透過上次專案最終輸出的 cnt_meas，計算出應該有多少的 cnt 數才能表示 r_distance 的距離。當 r_distance 的距離達到時，就會輸出信號。

在課堂上，我們學到了許多編寫硬體程式的技巧，例如每經過一個模組就存入 D-FF，以確保系統的穩定性等等。本次專案利用硬體的方式計算距離，我覺得非常有趣且新穎，期待下一個專案也能帶給我們同樣實用且有趣的體驗！