數位系統導論實驗

Lab 11

Single-MAC FIR filter

助教：蔣宗廷

Outline

- Goal
- Review of FIR filters
- Single-MAC (multiply accumulate)-based implementation
- Top-down design flow & simulation
- Homework

Goal

本次課堂將介紹使用FIR Filter (有限脈衝響應濾波器)的循序電路設計及狀態機的行為，並為下一堂Lab的DNN加速器的乘積累加運算做準備。

Finite Impulse Response(FIR) Filters

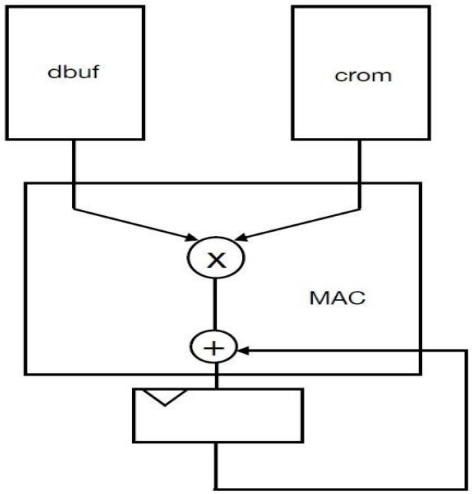
$$y[n] = \sum_{i=0}^{k-1} x[n-i]h[i]$$

x[n]與y[n]分別為input與output第n個sample，其中y[n]為包含x[n]與前k-1個input sample之加權平均，其權重為h[i]。

FIR Implementation

- 每個output sample y[n]需要k個乘法 (x[n-i]＊h[i])和k-1個加法。
- 利用一個乘法器與一個累加器，可以用k個cycle完成1個output計算。

Single-MAC FIR



crom：濾波係數
dbuf：input data
mpy：乘法器（濾波係數×inputdata）
add：加法器（mpy結果累加）

MAC兩個輸入端分別為data及濾波係數，每個cycle只能做一次完整乘加運算，也就是將一筆data乘上一個coefficient，接著再進入累加器，與先前累加值相加後輸出。

Input address	4	3	2	1	0
Input data	5	6	7	8	9
coefficient					x 2
coefficient address					0

Input address	4	3	2	1	0
Input data	5	6	7	8	9
coefficient					x 2
coefficient address					0

Input address	5	4	3	2	1	0
Input data	4	5	6	7	8	9
coefficient					x 2	x 3
coefficient address					0	1

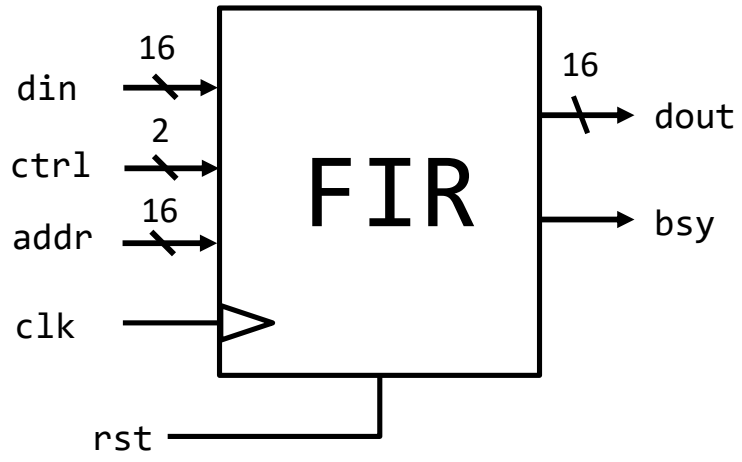
Input address	5	4	3	2	1	0
Input data	4	5	6	7	8	9
coefficient					x 2	x 3
coefficient address					0	1

Data 1
cycle 1 第一筆data送入，並且運算9x2
cycle 2 運算0x3 + 第1個cycle的值
cycle 3 運算0x4 + 第2個cycle的值
cycle 4 運算0x3 + 第3個cycle的值
cycle 5 運算0x2 + 第4個cycle的值

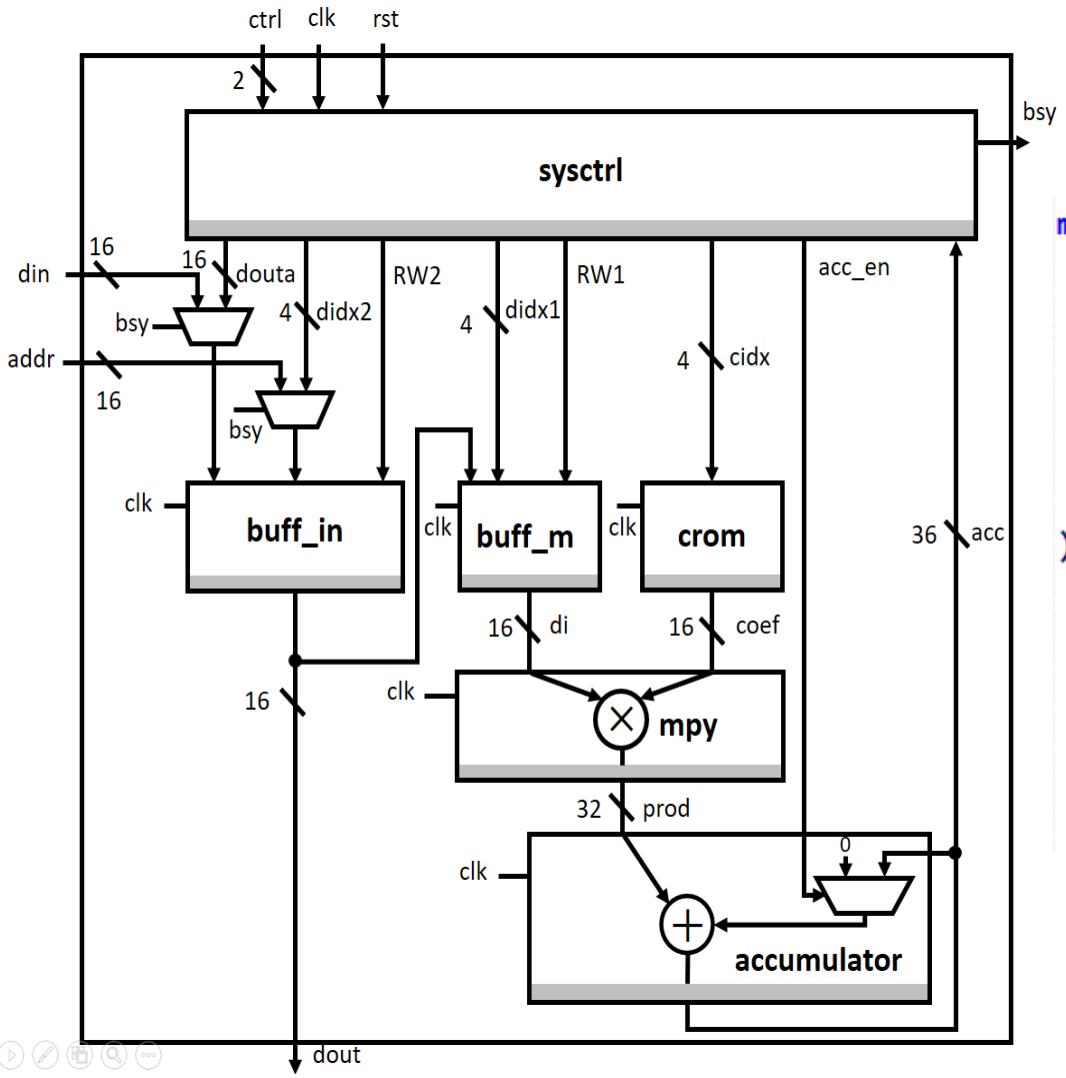
Data 2
cycle 6 輸出第一筆運算結果，data向前推進，並且運算8x2
cycle 7 運算9x3 + 第6個cycle的值
cycle 8 運算0x4 + 第7個cycle的值
cycle 9 運算0x3 + 第8個cycle的值
cycle 10 運算0x2 + 第9個cycle的值

Top-level design

- 硬體架構 - FIR Implementation
  - 首先展示我們會用到的硬體元件以及它們所需要的規格，全部結合在一起後以得到 Implementation FIR 的硬體架構。



Structural design




```
module fir(
    input wire clk,
    input wire rst,
    input wire [1:0] ctrl,
    input wire [15:0] addr,
    input wire [15:0] din,
    output wire [15:0] dout,
    output wire bsy
);
    wire RW1,RW2;
    wire [3:0] cidx;
    wire [5:0] didx1, didx2;
    wire [15:0] coef, di, douta;
    wire [31:0] prod;
    wire [35:0] acc;
    wire acc_en;
```

```
sysctrl ctrl_m(
    .clk(clk),
    .rst(rst),
    .ctrl(ctrl),
    .acc(acc),
    .RW1(RW1),
    .RW2(RW2),
    .didx1(didx1),
    .didx2(didx2),
    .cidx(cidx),
    .douta(douta),
    .acc_en(acc_en),
    .bsy(bsy)
);
```

```
dbuf buff_in(
    .clk(clk),
    .din((bsy)?douta:din),
    .didx((bsy)?didx2:addr[7:2]),
    .RW(RW2),
    .di(dout)
);
```

```
dbuf buff_m(
    .clk(clk),
    .din(dout),
    .didx(didx1),
    .RW(RW1),
    .di(di)
);
```

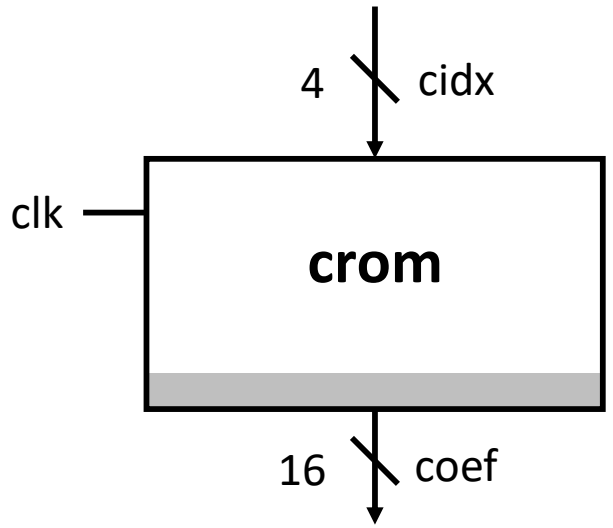
```
crom coef_m(
    .clk(clk),
    .cidx(cidx),
    .coef(coef)
);
```

```
multi mult_m(
    .clk(clk),
    .prod(prod),
    .coef(coef),
    .di(di)
);
```

```
accumulator acc_m(
    .clk(clk),
    .prod(prod),
    .acc(acc),
    .acc_en(acc_en)
);
```

Submodule : crom

- 將會用來運算的係數資料固定在記憶體裡 (ROM)

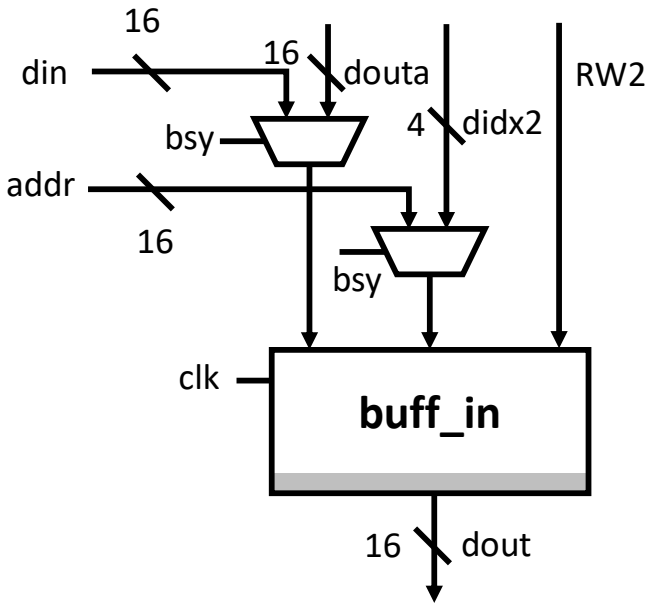



```
crom coef_m(
    .clk(clk),
    .cidx(cidx),
    .coef(coef)
);
```

```
module crom(
    input wire clk,
    input wire [3:0] cidx,
    output reg [15:0] coef
);
    parameter coefficent00 = 16'b0000_0000_0000_0001; //1
    parameter coefficent01 = 16'b0000_0000_0000_0010;
    parameter coefficent02 = 16'b0000_0000_0000_0011;
    parameter coefficent03 = 16'b0000_0000_0000_0100;
    parameter coefficent04 = 16'b0000_0000_0000_0101; //5
    parameter coefficent05 = 16'b0000_0000_0000_0100;
    parameter coefficent06 = 16'b0000_0000_0000_0011;
    parameter coefficent07 = 16'b0000_0000_0000_0010;
    parameter coefficent08 = 16'b0000_0000_0000_0001; //1
    parameter coefficent09 = 16'b0000_0000_0000_0000;
    parameter coefficent10 = 16'b0000_0000_0000_0000;
    parameter coefficent11 = 16'b0000_0000_0000_0000;
    parameter coefficent12 = 16'b0000_0000_0000_0000;
    parameter coefficent13 = 16'b0000_0000_0000_0000;
    parameter coefficent14 = 16'b0000_0000_0000_0000;
    parameter coefficent15 = 16'b0000_0000_0000_0000;

    always @(posedge clk)begin
        case (cidx)
            4'b0000: coef <= coefficent00;
            4'b0001: coef <= coefficent01;
            4'b0010: coef <= coefficent02;
            4'b0011: coef <= coefficent03;
            4'b0100: coef <= coefficent04;
            4'b0101: coef <= coefficent05;
            4'b0110: coef <= coefficent06;
            4'b0111: coef <= coefficent07;
            4'b1000: coef <= coefficent08;
            4'b1001: coef <= coefficent09;
            4'b1010: coef <= coefficent10;
            4'b1011: coef <= coefficent11;
            4'b1100: coef <= coefficent12;
            4'b1101: coef <= coefficent13;
            4'b1110: coef <= coefficent14;
            4'b1111: coef <= coefficent15;
        endcase
    end

endmodule
```

Submodule : dbuf buff_in

- 將會用來運算的input資料全部寫進buff_in裡面暫存

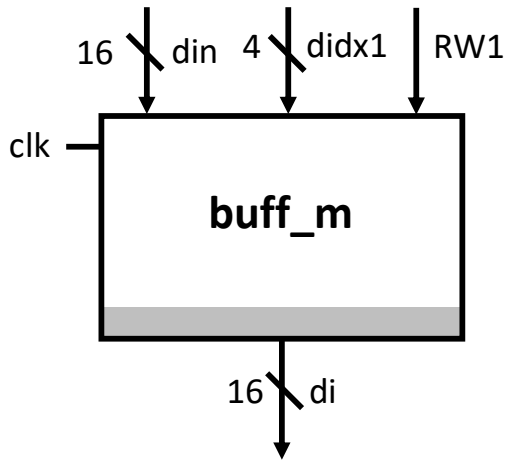



```
dbuf buff_in(
    .clk(clk),
    .din((bsy)?douta:din),
    .didx((bsy)?didx2:addr[7:2]),
    .RW(RW2),
    .di(dout)
);
```

```
module dbuf(
    input wire clk,
    input wire [15:0] din, //input
    input wire [5:0] didx, //address
    input wire RW,
    output reg [15:0] di
);
    reg [15:0] mem [0:63];

    // RW = 1 (write mode)
    // RW = 0 (read mode)
    always @(posedge clk) begin
        if(RW)
            mem[didx] <= din;

        di <= RW ? din : mem[didx];
    end

endmodule
```

Submodule : dbuf buff_m

- 將buff_in的資料搬到buff_m裡準備進行乘積累加運算

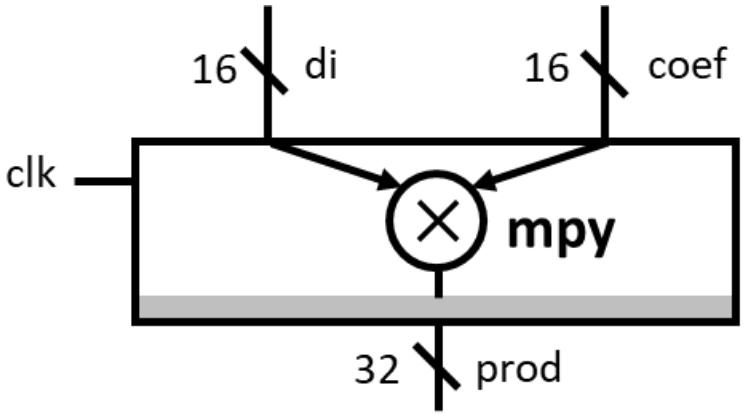



```
dbuf buff_m(
    .clk(clk),
    .din(dout),
    .didx(didx1),
    .RW(RW1),
    .di(di)
);
```

```
module dbuf(
    input wire clk,
    input wire [15:0] din, //input
    input wire [5:0] didx, //address
    input wire RW,
    output reg [15:0] di
);
    reg [15:0] mem [0:63];

    // RW = 1 (write mode)
    // RW = 0 (read mode)
    always @(posedge clk) begin
        if(RW)
            mem[didx] <= din;

        di <= RW ? din : mem[didx];
    end

endmodule
```

Submodule : mpy

- 把暫存在buff_m跟crom的資料相乘

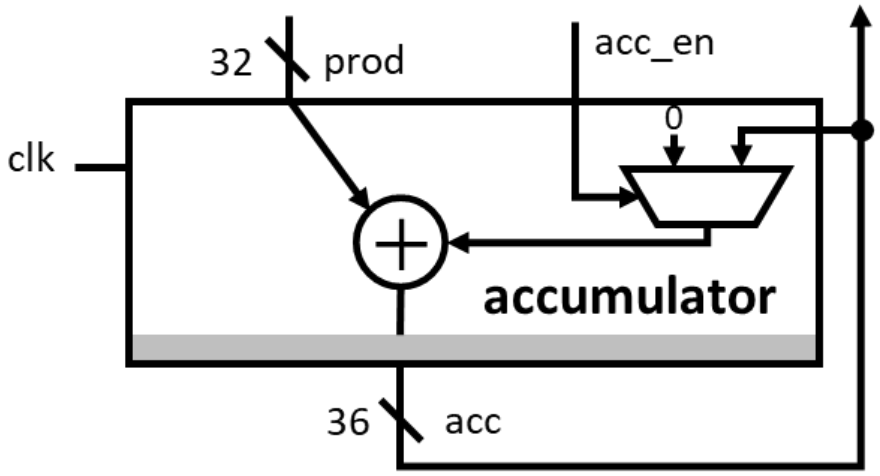



```
multi mult_m(
    .clk(clk),
    .prod(prod),
    .coef(coef),
    .di(di)
);
```

```
module multi(
    input wire clk,
    input wire [15:0] coef,
    input wire [15:0] di,
    output reg [31:0] prod
);
    always @(posedge clk) begin
        prod <= coef * di;
    end

endmodule
```

Submodule : accumulator

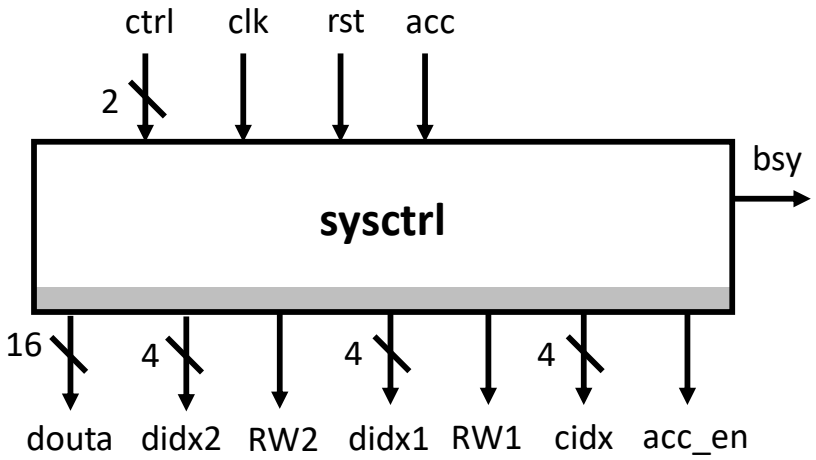
- 將mpy做完的乘法運算進行累加




```
accumulator acc_m(
    .clk(clk),
    .prod(prod),
    .acc(acc),
    .acc_en(acc_en)
);
```

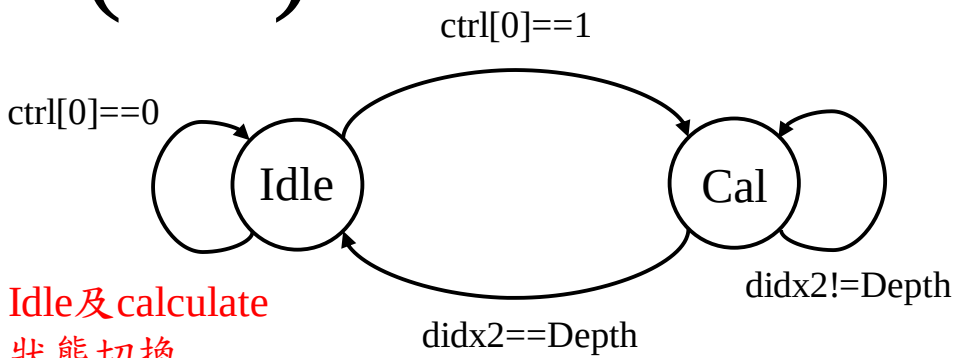
```
module accumulator(
    input wire clk,
    input wire [31:0] prod,
    input wire acc_en,
    output reg [35:0] acc
);

    always @(posedge clk) begin
        acc <= {{4{prod[31]}},prod[31:0]} + ((acc_en)?acc:0);
    end


endmodule
```

Submodule : sysctrl (1/2)




```
module sysctrl(
    input wire clk,
    input wire rst,
    input wire [1:0]ctrl,
    input wire [35:0] acc,
    output reg RW1,
    output reg RW2,
    output reg [5:0] didx1,
    output reg [5:0] didx2,
    output reg [3:0] cidx,
    output reg [15:0] douta,
    output reg acc_en,
    output reg bsy
);
    reg CS, NS;
    reg [3:0]counter;
    reg cnt_en;
```

```
//current state register
always@(posedge clk or negedge rst) begin
    if(~rst)
        CS<=Idle;
    else
        CS<=NS;
end

//next state logic
always@(*) begin
    case(CS)
        Idle:
            begin
                if(ctrl[0])
                    NS=Calculate;
                else
                    NS=Idle;
            end
        Calculate:
            begin
                if(didx2==Depth)
                    NS=Idle;
                else
                    NS=Calculate;
            end
        default:
            begin
                NS=Idle;
            end
    endcase
end
```

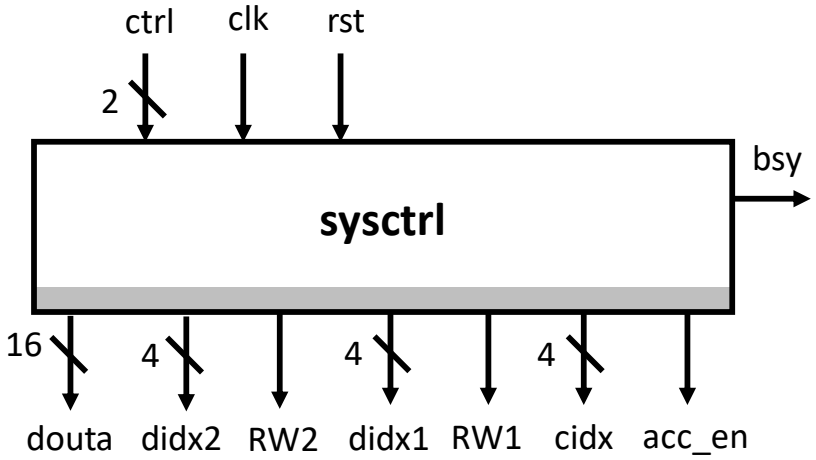


Idle及calculate
狀態切換

```
//output logic
always@(*) begin
    case(CS)
        Idle:
            begin
                RW2=&(~didx2);
                RW1=1'b0;
                acc_en=1'b0;
                cnt_en=1'b0;
                bsy=1'b0;
            end
        Calculate:
            begin
                RW2=(~counter[2])&(counter[1])&(counter[0]);
                RW1=&(~counter);
                acc_en=(counter[2])|(~counter[1])|(counter[0]);
                cnt_en=1'b1;
                bsy=1'b1;
            end
        default:
            begin
                RW1=1'b0;
                RW2=1'b0;
                acc_en=1'b0;
                cnt_en=1'b0;
                bsy=1'b0;
            end
    endcase
end
```

Idle及calculate
狀態行為

Submodule : sysctrl (2/2)




```
//didx2
always @(posedge clk or negedge rst)begin
    if(~rst) begin
        didx2<= 6'd0;
    end
    else if(ctrl[1]) begin
        didx2<= 6'd0;
    end
    else if(counter[2]&counter[1]&counter[0]) begin
        didx2<= didx2+1'b1;
    end
end
```

```
//didx1
always @(posedge clk or negedge rst)begin
    if(~rst) begin
        didx1<= 6'd8;
    end
    else if(counter[3]) begin
        didx1<= didx1;
    end
    else if((~|didx1) | ctrl[1])begin
        didx1<= 6'd8;
    end
    else begin
        didx1<= didx1-1'b1;
    end
end
```

控制
dbuf data address

```
//counter
always @(posedge clk or negedge rst)begin
    if(~rst) begin
        counter<=4'd0;
    end
    else if(counter[3] | ctrl[1])begin
        counter<=4'd0;
    end
    else if(cnt_en)begin
        counter<=counter+1'b1;
    end
end
```

計數器

```
//dout
always@(posedge clk)begin
    douta <= acc[15:0];
end
```

輸出結果

```
//cidx
always @(posedge clk or negedge rst)begin
    if(~rst) begin
        cidx<= 4'd0;
    end
    else if(counter[3] | ctrl[1]) begin
        cidx<= 4'd0;
    end
    else begin
        cidx<= cidx+1'b1;
    end
end
```

控制
crom data address

Testbench

```
// define a ram store input signal
reg [15:0] mem1[0:35];
reg [15:0] mem2[0:35];

// read data from disk
initial begin
    $readmemb("./databin3.mem" , mem1);
    $readmemb("./ans.mem" , mem2);
end
```

讀取input資料及ans資料

```
for (i=1;i<37;i=i+1) begin
    @(posedge clk);
    din <= mem1[i-1];
    addr <= (i*4);
end

@(posedge clk);
din <= mem1[0];
addr <= 0;
```

放入input資料

```
$display("start run");

@(posedge clk);
ctrl <= 2'b01;
@(posedge clk);
ctrl <= 2'b00;

$display("polling busy");
@(posedge clk);

while(bsy!=1'b0) begin
    @(posedge clk);
end

$display("operation finish");
@(posedge clk);
for (i=1;i<37;i=i+1) begin
    addr <= (i*4);
    @(posedge clk);
    @(posedge clk);
    ans[i-1]<=dout;
end
```

進行運算

```
for(i=0;i<36;i=i+1) begin
    if(mem2[i]===16'dx && ans[i]===16'dx) begin
        $display("======================");
        $display("=      Correct[%02d]   =",i);
        $display("======================");
        $display("your answer:%d\nreal answer:%d",ans[i],mem2[i]);
    end
    else begin
        casez({|(mem2[i]^ans[i])})
            1'b1: begin
                $display("======================");
                $display("=        Wrong[%02d]   =",i);
                $display("======================");
                $display("your answer:%d\nreal answer:%d",ans[i],mem2[i]);
            end
            1'b0: begin
                $display("======================");
                $display("=      Correct[%02d]   =",i);
                $display("======================");
                $display("your answer:%d\nreal answer:%d",ans[i],mem2[i]);
            end
            1'bx: begin
                $display("======================");
                $display("=        Wrong[%02d]   =",i);
                $display("======================");
                $display("your answer:%d\nreal answer:%d",ans[i],mem2[i]);
            end
        endcase
    end
end
```

輸出結果

Result

Testbench模擬結果

```
=      Correct[00]   =
your answer:     x
real answer:     x
=      Correct[01]   =
your answer:     x
real answer:     x
=      Correct[02]   =
your answer:     x
real answer:     x
=      Correct[03]   =
your answer:     x
real answer:     x
=      Correct[04]   =
your answer:     x
real answer:     x
=      Correct[05]   =
your answer:     x
real answer:     x
=      Correct[06]   =
your answer:     x
real answer:     x
=      Correct[07]   =
your answer:     x
real answer:     x
=      Correct[08]   =
your answer:   101
real answer:   101
=      Correct[09]   =
your answer:   125
real answer:   125
=      Correct[10]   =
your answer:   150
real answer:   150
=      Correct[11]   =
your answer:   175
real answer:   175
=      Correct[12]   =
your answer:   200
real answer:   200
=      Correct[13]   =
your answer:   225
real answer:   225
=      Correct[14]   =
your answer:   250
real answer:   250
=      Correct[15]   =
your answer:   275
real answer:   275
=      Correct[16]   =
your answer:   284
real answer:   284
=      Correct[17]   =
your answer:   276
real answer:   276
=      Correct[18]   =
your answer:   250
real answer:   250
=      Correct[19]   =
your answer:   205
real answer:   205
=      Correct[20]   =
your answer:   140
real answer:   140
=      Correct[21]   =
your answer:    86
real answer:    86
=      Correct[22]   =
your answer:    44
real answer:    44
=      Correct[23]   =
your answer:    15
real answer:    15
=      Correct[24]   =
your answer:     0
real answer:     0
=      Correct[25]   =
your answer:     0
real answer:     0
```

Zedboard模擬結果

```
Zynq_BASE mapping successful :
0x80000000 to 0xb6efc000, size = 1024
start run
polling busy
operation finish
accelerator result[00] = 5
accelerator result[01] = 3
accelerator result[02] = 7
accelerator result[03] = 14
accelerator result[04] = 25
accelerator result[05] = 39
accelerator result[06] = 57
accelerator result[07] = 78
accelerator result[08] = 101
accelerator result[09] = 125
accelerator result[10] = 150
accelerator result[11] = 175
accelerator result[12] = 200
accelerator result[13] = 225
accelerator result[14] = 250
accelerator result[15] = 275
accelerator result[16] = 284
accelerator result[17] = 276
accelerator result[18] = 250
accelerator result[19] = 205
accelerator result[20] = 140
accelerator result[21] = 86
accelerator result[22] = 44
accelerator result[23] = 15
accelerator result[24] = 0
accelerator result[25] = 0
```

HW

- 作業內容：因為FIR Filter係數對稱的關係試著縮減重複的係數達到減少input data與coefficient之間的運算
- 評分方式：課程範例 50% (Testbench測試成功25%，Zedboard測試成功25%)
  作業練習 30% (Testbench測試成功15%，Zedboard測試成功15%)
  隨堂練習 20%
- Demo地點：工程一館206

Appendix

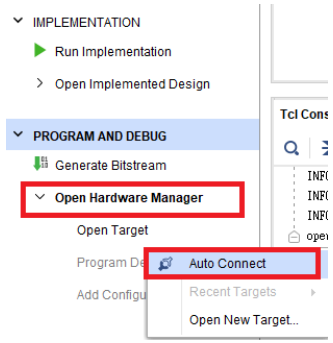
- 編譯 C for ARM
- 燒錄至FPGA
- 於Terminal輸出結果

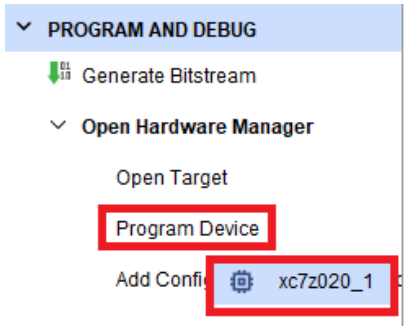
編譯 C for ARM

- 打開cmd 輸入指令: arm-none-linux-gnueabi-gcc FIR.c -o FIR
- 將FIR執行檔放入SD卡

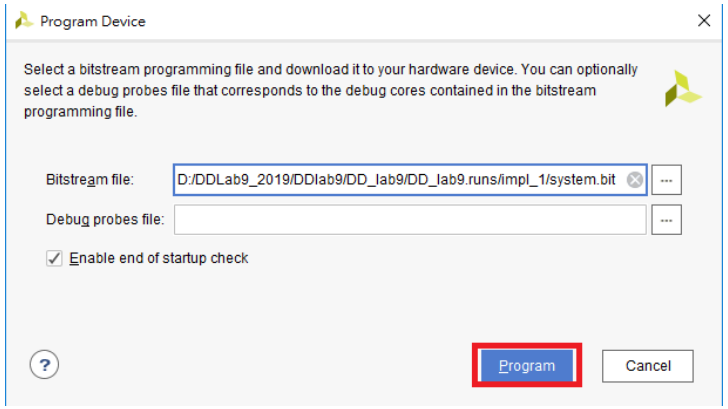
燒錄至FPGA

- 將USB線連接Zedboard PROG port與電腦
- 開啟DD_FIR.xpr
- 點選左下角Open Hardware Manager再點擊Open target並選擇Auto Connect





- 點擊Program Device，並選xc7z020_1
- 燒錄程式到FPGA上



於Terminal輸出結果

- 開啟MobaXTerm，並掛載file system
  - mount /dev/mmcblk0p1 /mnt
- 將目錄切換到SD卡
  - cd /mnt
- 設定FIR的權限為可執行
  - chmod +x FIR
- 執行FIR
  - ./FIR