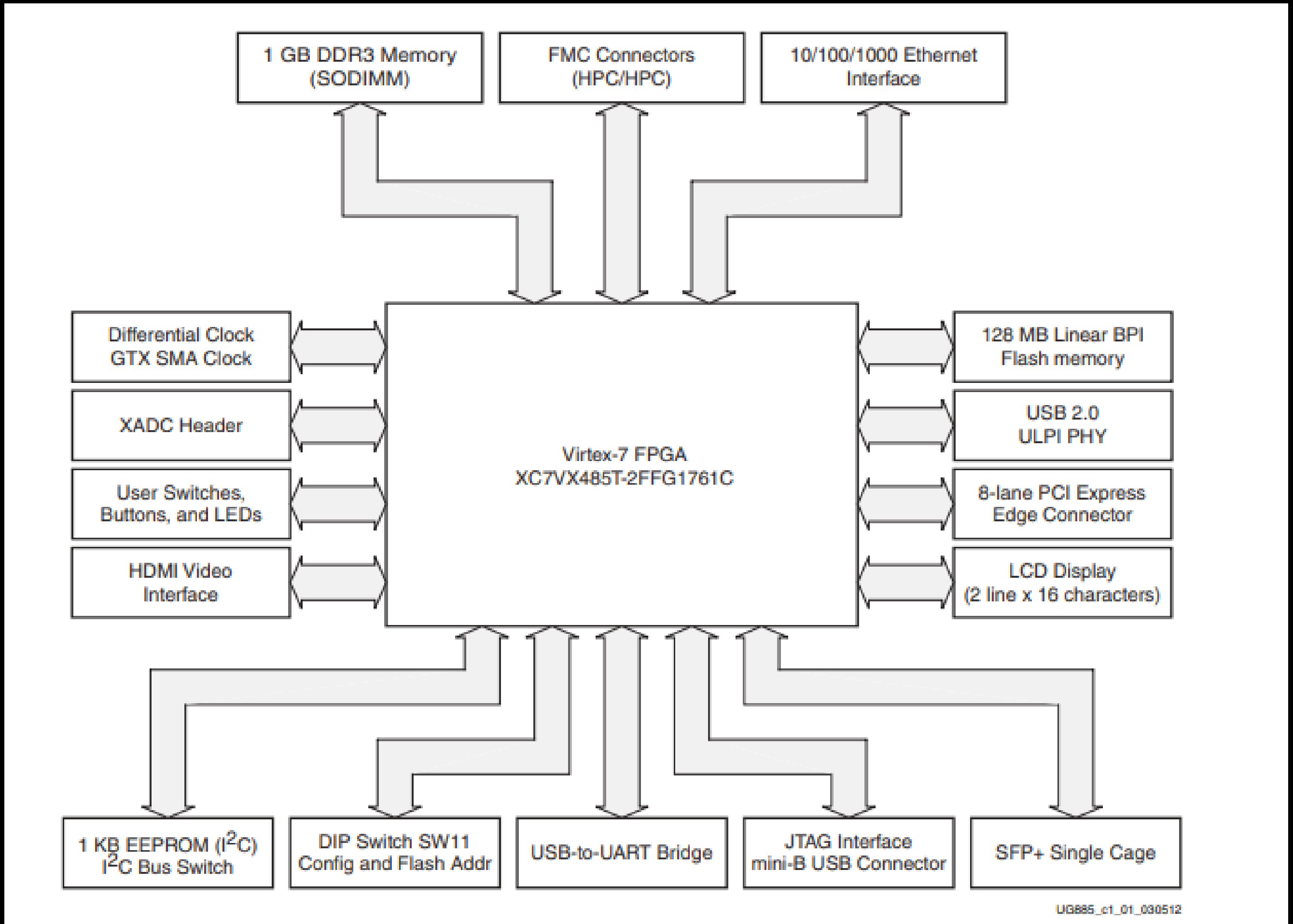
FPGA Lab
Vivado Design Suit

Evaluation Board Overview (VC707)

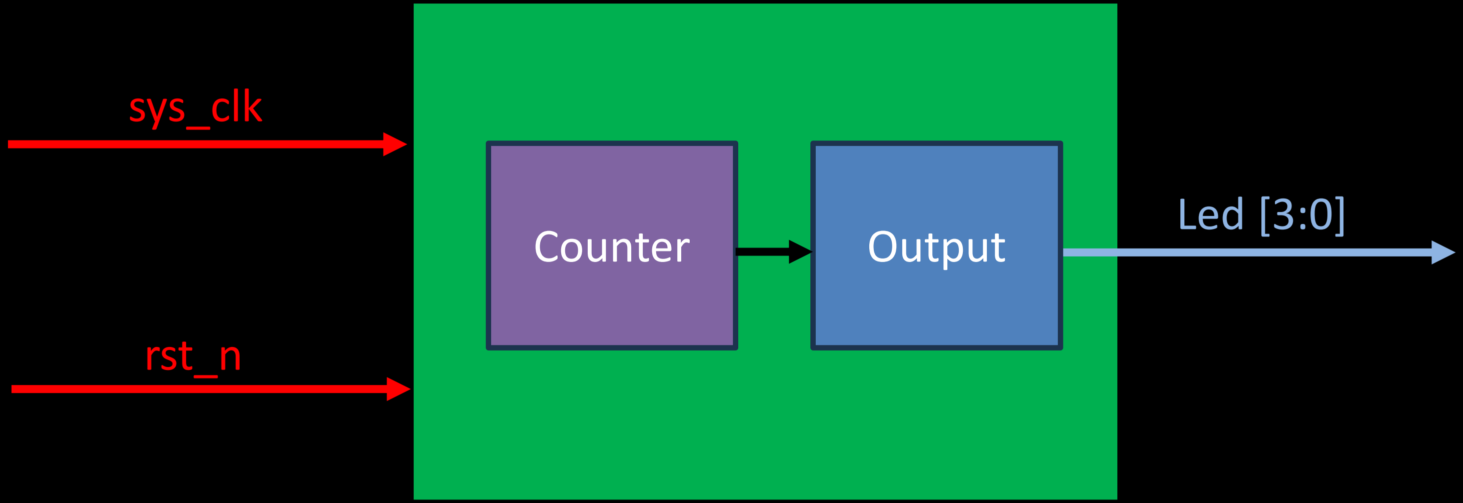


1 GB DDR3 Memory (SODIMM)
FMC Connectors (HPC/HPC)
10/100/1000 Ethernet Interface
Differential Clock GTX SMA Clock
XADC Header
User Switches, Buttons, and LEDs
HDMI Video Interface
Virtex-7 FPGA XC7VX485T-2FFG1761C
128 MB Linear BPI Flash memory
USB 2.0 ULPI PHY
8-lane PCI Express Edge Connector
LCD Display (2 line x 16 characters)
1 KB EEPROM (I2C) I2C Bus Switch
DIP Switch SW11 Config and Flash Addr
USB-to-UART Bridge
JTAG Interface mini-B USB Connector
SFP+ Single Cage
UG885_c1_01_030512

Figure 1-1: **VC707 Board Block Diagram**

Lab1 - Shift_Led

設計一個電路，每經過10毫秒就切換Led燈號



Lab1 - Shift_Led

Port define

```
`timescale 1ns / 1ps
module shift_led(
    input   wire            sys_clk ,
    input   wire            rst_n   ,
    output  reg     [3:0]   led
    );

    reg [27:0]  cnt      ;
    wire        add_cnt  ; // 啟用計數器
    wire        end_cnt  ; // 記數完成
```

Lab1 - Shift_Led

Counter

```
    always@(posedge sys_clk or negedge rst_n)begin
        if(~rst_n)
            cnt <= 'd0;
        else if(add_cnt)begin
            if(end_cnt)
                cnt <= 'd0;
            else
                cnt <= cnt + 1'b1;
        end
        else
            cnt <= 'd0;
    end

    assign add_cnt = 1;
    // sys_clk 200MHz
    assign end_cnt = add_cnt && cnt == 200000000 - 1;
```

Lab1 - Shift_Led

Output LED

```
    always@(posedge sys_clk or negedge rst_n)begin
        if(~rst_n)
            led <= 4'b1110;
        else if(end_cnt)
            led <= {led[2:0],led[3]};
        else
            led <= ~led;
    end
endmodule
```

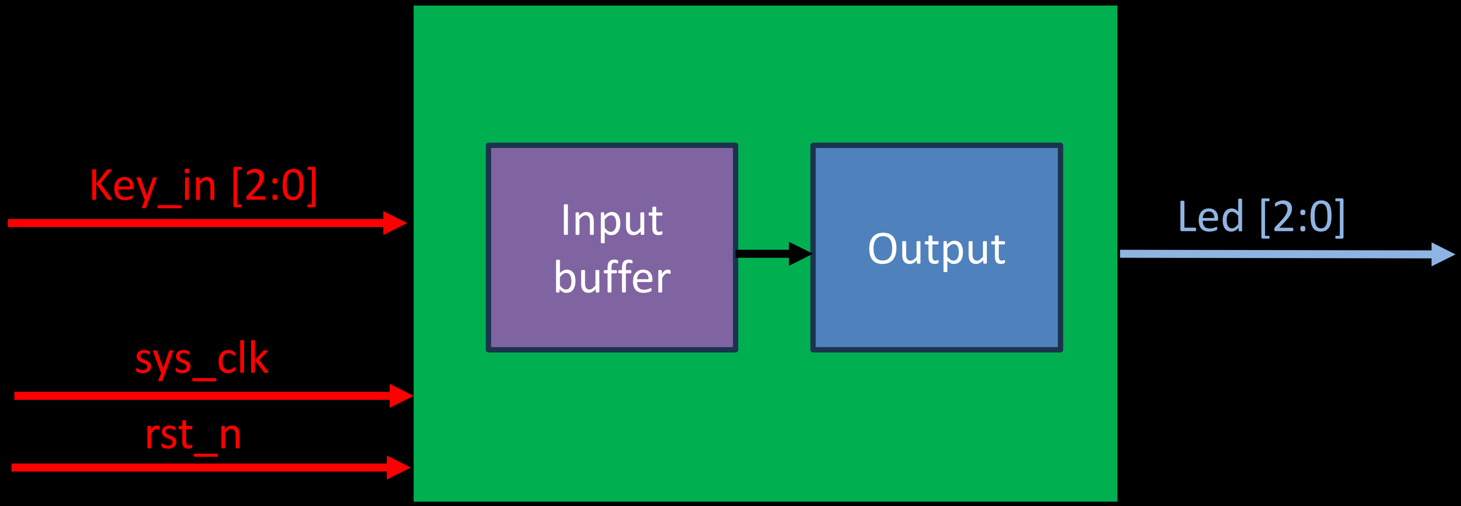
Lab1 - Shift_Led

Constrints

```
set_property PACKAGE_PIN AM39 [get_ports {led[0]}]
set_property PACKAGE_PIN AN39 [get_ports {led[1]}]
set_property PACKAGE_PIN AR37 [get_ports {led[2]}]
set_property PACKAGE_PIN AT37 [get_ports {led[3]}]
set_property PACKAGE_PIN AR40 [get_ports rst_n]
set_property PACKAGE_PIN E19 [get_ports sys_clk]
set_property IOSTANDARD LVCMOS18 [get_ports {led[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports rst_n]
set_property IOSTANDARD LVCMOS18 [get_ports sys_clk]
```

Lab2 - Key_LED

設計一個電路，燈號會依照按鍵輸入變化



Lab2 - FPGA_LED

Port define

```
`timescale 1ns / 1ps
module key_led(
    input   wire        sys_clk ,
    input   wire        rst_n   ,
    input   wire  [2:0] key_in  ,
    output  reg   [2:0] led
    );

reg     key_in_d0   ;
reg     key_in_d1   ;
reg     key_in_d2   ;
reg     key_in_dd0  ;
reg     key_in_dd1  ;
reg     key_in_dd2  ;
```

Lab2 - FPGA_LED

Input buffer

```
always@(posedge sys_clk or negedge rst_n)begin
    if(~rst_n)begin
        key_in_d0 <= 1'b0;
        key_in_d1 <= 1'b0;
        key_in_d2 <= 1'b0;
    end
    else begin
        key_in_d0 <= key_in[0];
        key_in_d1 <= key_in[1];
        key_in_d2 <= key_in[2];
    end
end
```

Lab2 - FPGA_LED

Input buffer

```
always@(posedge sys_clk or negedge rst_n)begin
    if(~rst_n)begin
        key_in_dd0 <= 1'b0;
        key_in_dd1 <= 1'b0;
        key_in_dd2 <= 1'b0;
    end
    else begin
        key_in_dd0 <= key_in_d0;
        key_in_dd1 <= key_in_d1;
        key_in_dd2 <= key_in_d2;
    end
end
```

Lab2 - FPGA_LED

Output Led

```
always@(posedge sys_clk or negedge rst_n)begin
    if(~rst_n)
        led[0] <= 1'b1;
    else if(~key_in_dd0)
        led[0] <= 1'b0;
    else
        led[0] <= 1'b1;
end
always@(posedge sys_clk or negedge rst_n)begin
    if(~rst_n)
        led[1] <= 1'b1;
    else if(~key_in_dd1)
        led[1] <= 1'b0;
    else
        led[1] <= 1'b1;
end
```

Lab2 - FPGA_LED

Output Led

```
always@(posedge sys_clk or negedge rst_n)begin
    if(~rst_n)
        led[2] <= 1'b1;
    else if(~key_in_dd2)
        led[2] <= 1'b0;
    else
        led[2] <= 1'b1;
end
```

Lab2 - Shift_Led

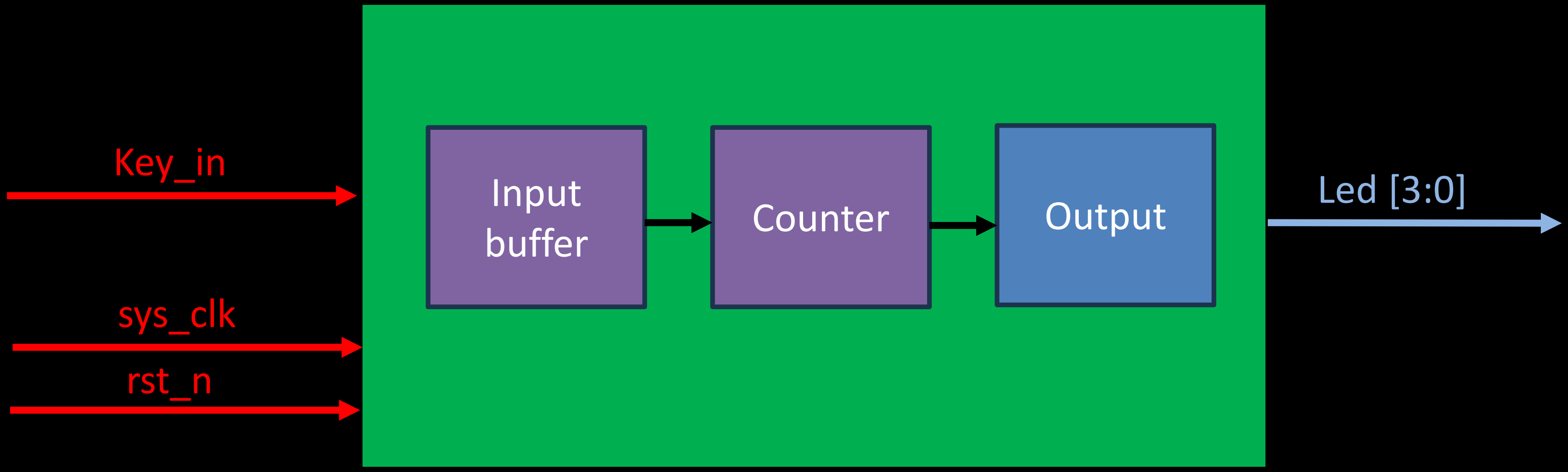
Constrints

```
set_property PACKAGE_PIN AU39 [get_ports rst_n]
set_property PACKAGE_PIN AM39 [get_ports {led[0]}]
set_property PACKAGE_PIN AN39 [get_ports {led[1]}]
set_property PACKAGE_PIN AR37 [get_ports {led[2]}]
set_property PACKAGE_PIN AU38 [get_ports {key_in[2]}]
set_property PACKAGE_PIN AV39 [get_ports {key_in[1]}]
set_property PACKAGE_PIN AW40 [get_ports {key_in[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {key_in[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {key_in[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {key_in[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports rst_n]
set_property IOSTANDARD LVCMOS18 [get_ports sys_clk]

set_property PACKAGE_PIN E19 [get_ports sys_clk]
```

Lab3 - Key_Debounce

設計一個電路，燈號會依照按鍵輸入變化，且按鍵按下有防抖動的功能



Lab3 - Key_Debounce

Port define

```
`timescale 1ns / 1ps
module key_debounce(
    input   wire            clk     ,
    input   wire            rst_n   ,
    input   wire            key_in  ,
    output  wire     [3:0]   led
    );

parameter   CNT_MAX = 1000000 - 1;

reg [1:0]   key_dd          ;
reg [21:0]  cnt_time        ;
wire        add_cnt_time    ;
wire        end_cnt_time    ;

reg         flag            ;
reg         key_flag        ;
reg [3:0]   led_r           ;
```

Lab3 - Key_Debounce

Input buffer

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        key_dd <= 'd0;
    else
        key_dd <= {key_dd[0],key_in};
end
```

Lab3 - Key_Debounce

Counter

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        cnt_time <= 'd0;
    else if(add_cnt_time)begin
        if(end_cnt_time)
            cnt_time <= cnt_time;
        else
            cnt_time <= cnt_time + 1'b1;
    end
    else
        cnt_time <= 'd0;
end

assign add_cnt_time = key_dd[0] == 1'b0;
assign end_cnt_time = add_cnt_time && cnt_time == CNT_MAX;
```

Lab3 - Key_Debounce

Flag

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        flag <= 'd0;
    else if(cnt_time == CNT_MAX)
        flag <= 1'b1;
    else if(key_dd[1])
        flag <= 1'b0;
end

always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        key_flag <= 'd0;
    else if(cnt_time == CNT_MAX && flag == 1'b0)
        key_flag <= 1'b1;
    else if(key_dd[1])
        key_flag <= 1'b0;
end
```

Lab3 - Key_Debounce

Output Led

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        led_r <= 4'b0001;
    else if(key_flag)
        led_r <= {led_r[2:0] , led_r[3]};
    else if(key_dd[1] == 1'b1)
        led_r <= led_r;
end

assign led = led_r;
endmodule
```

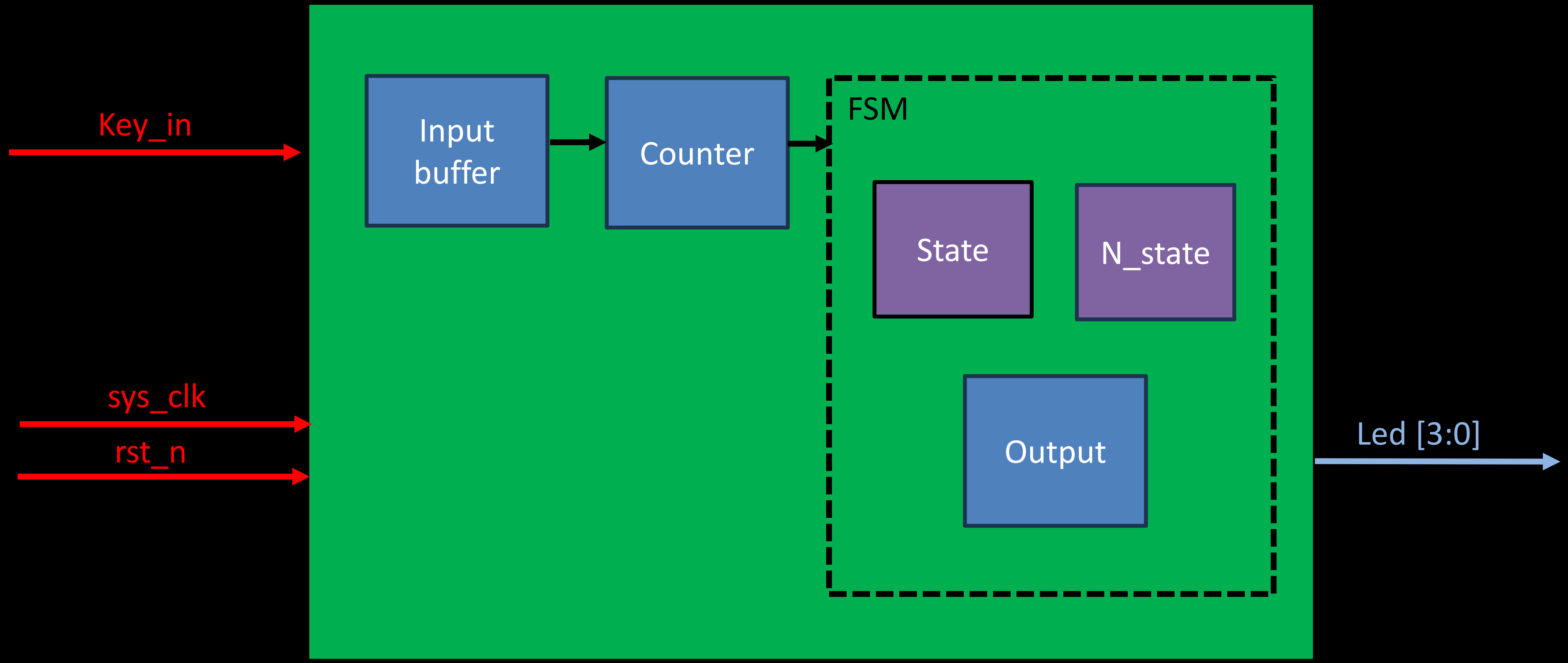
Lab3 - Key_Debounce

Constrints

```
set_property PACKAGE_PIN AM39 [get_ports {led[0]}]
set_property PACKAGE_PIN AN39 [get_ports {led[1]}]
set_property PACKAGE_PIN AR37 [get_ports {led[2]}]
set_property PACKAGE_PIN AT37 [get_ports {led[3]}]
set_property PACKAGE_PIN AR40 [get_ports rst_n]
set_property PACKAGE_PIN E19 [get_ports clk]
set_property PACKAGE_PIN AW40 [get_ports key_in]
set_property IOSTANDARD LVCMOS18 [get_ports {led[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports key_in]
set_property IOSTANDARD LVCMOS18 [get_ports rst_n]
```

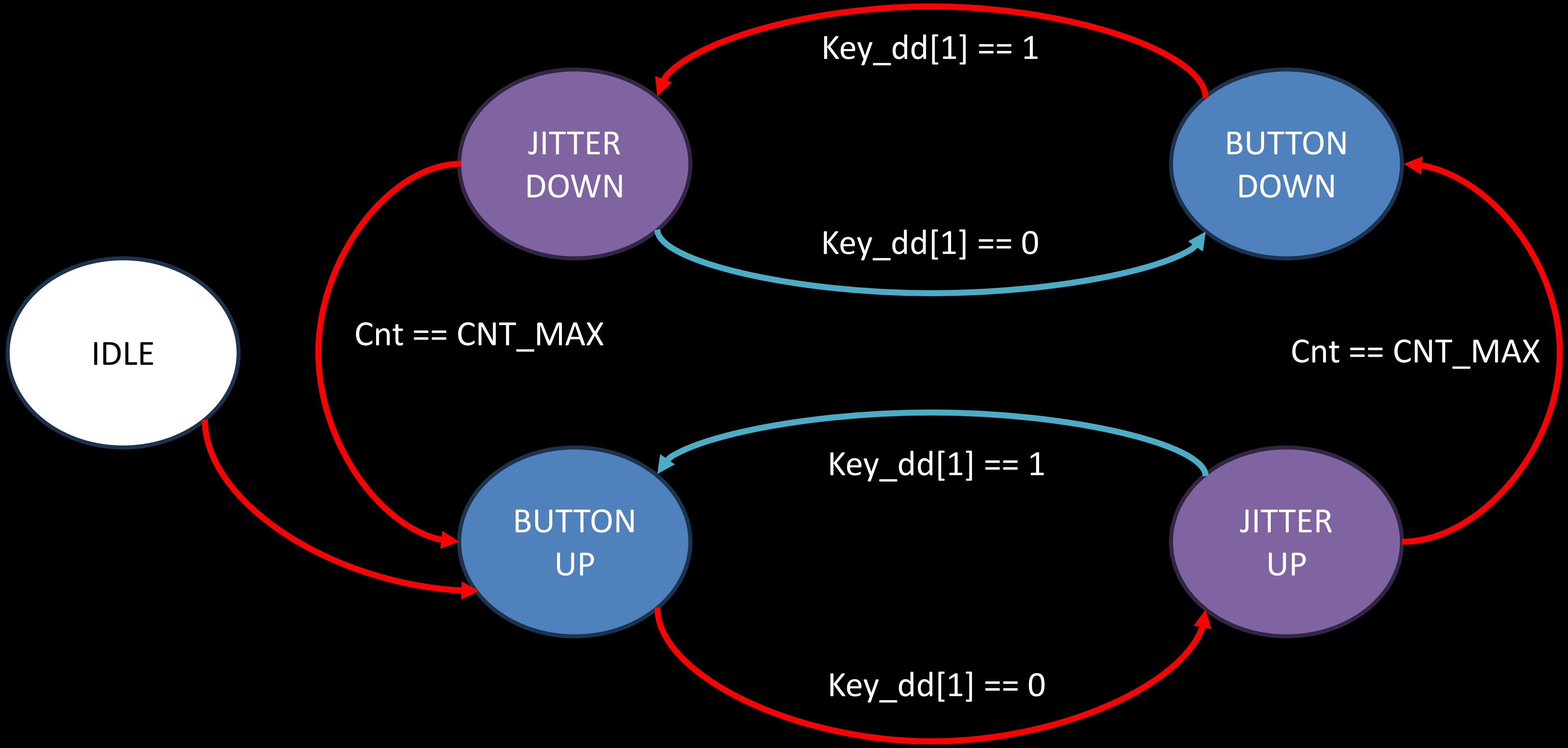
Lab4 - Key_FSM

設計一個狀態機電路，燈號會依照按鍵輸入變化，且按鍵按下有防抖動的功能



Lab4 - Key_FSM

設計一個電路，燈號會依照按鍵數入變化，且按鍵按下有防抖動的功能



Lab4 - Key_FSM

Port define

```
`timescale 1ns / 1ps
module key_debounce(
    input   wire            clk     ,
    input   wire            rst_n   ,
    input   wire            key_in  ,
    output  wire      [3:0] led
    );
```

Lab4 - Key_FSM

State define

```
parameter    CNT_MAX     = 40000000 - 1;
parameter    IDLE        = 3'b000;
parameter    BUTTON_UP   = 3'b001;
parameter    JITTER_UP   = 3'b010;
parameter    BUTTON_DOWN = 3'b011;
parameter    JITTER_DOWN = 3'b100;

reg [4:0]    state          ;
reg [1:0]    key_dd         ;
reg [31:0]   cnt_jitter1    ;
reg [31:0]   cnt_jitter2    ;

reg          key_flag       ;
reg [3:0]    led_r          ;
```

Lab4 - Key_FSM

N_state

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        state <= IDLE;
    case(state)
        IDLE        : state = BUTTON_UP;
        BUTTON_UP   : begin
            if(key_dd[1] == 1'b0)
                state <= JITTER_UP;
            else
                state <= BUTTON_UP;
        end
        JITTER_UP   : begin
            if(key_dd[1] == 1'b1)
                state <= BUTTON_UP;
            else if(cnt_jitter1 == CNT_MAX)
                state <= BUTTON_DOWN;
            else
                state <= JITTER_UP;
        end
```

Lab4 - Key_FSM

N_state

```
            BUTTON_DOWN    : begin
                if(key_dd[1] == 1'b1)
                    state <= JITTER_DOWN;
                else
                    state <= BUTTON_DOWN;
            end
            JITTER_DOWN    : begin
                if(cnt_jitter2 == CNT_MAX)
                    state <= BUTTON_UP;
                else if(key_dd[1] == 1'b0)
                    state <= BUTTON_DOWN;
                else
                    state <= JITTER_DOWN;
            end
            default : state = IDLE;
        endcase
end
```

Lab4 - Key_FSM

Input buffer

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        key_dd <= 'd0;
    else
        key_dd <= {key_dd[0],key_in};
end
```

Lab4 - Key_FSM

Counter UP

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        cnt_jitter1 <= 'd0;
    else if(state == JITTER_UP && key_dd[1] == 1'b0)begin
        if(cnt_jitter1 == CNT_MAX)
            cnt_jitter1 <= 'd0;
        else
            cnt_jitter1 <= cnt_jitter1 + 1'b1;
    end
    else
        cnt_jitter1 <= 'd0;
end
```

Lab4 - Key_FSM

Counter DOWN

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        cnt_jitter2 <= 'd0;
    else if(state == JITTER_DOWN && key_dd[1] == 1'b1)begin
        if(cnt_jitter2 == CNT_MAX)
            cnt_jitter2 <= 'd0;
        else
            cnt_jitter2 <= cnt_jitter2 + 1'b1;
    end
    else
        cnt_jitter2 <= 'd0;
end
```

Lab4 - Key_FSM

Flag

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        key_flag <= 'd0;
    else if(cnt_jitter1 == CNT_MAX && state == JITTER_UP)
        key_flag <= 1'b1;
    else if(key_dd[1])
        key_flag <= 1'b0;
end
```

Lab4 - Key_FSM

Output Led

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        led_r <= 4'b0001;
    else if(key_flag)
        led_r <= {led_r[2:0] , led_r[3]};
    else if(key_dd[1] == 1'b1)
        led_r <= led_r;
end

assign led = led_r;

endmodule
```

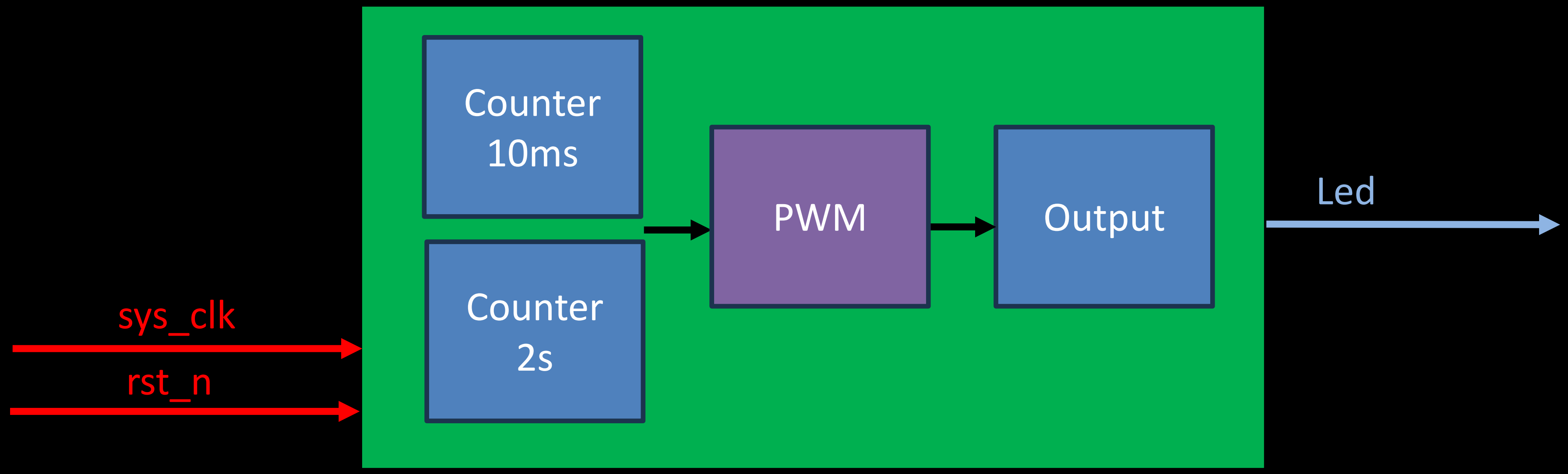
Lab3 - Key_Debounce

Constrints

```
set_property PACKAGE_PIN AM39 [get_ports {led[0]}]
set_property PACKAGE_PIN AN39 [get_ports {led[1]}]
set_property PACKAGE_PIN AR37 [get_ports {led[2]}]
set_property PACKAGE_PIN AT37 [get_ports {led[3]}]
set_property PACKAGE_PIN AR40 [get_ports rst_n]
set_property PACKAGE_PIN E19 [get_ports clk]
set_property PACKAGE_PIN AW40 [get_ports key_in]
set_property IOSTANDARD LVCMOS18 [get_ports {led[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports key_in]
set_property IOSTANDARD LVCMOS18 [get_ports rst_n]
```

Lab5 - PWM

設計一個隨時間變化亮度的呼吸燈



Lab5 - PWM

Port define

```
`timescale 1ns / 1ps
module breath_led(
    input   wire    clk     ,
    input   wire    rst_n   ,
    output  wire    led
    );

    parameter   CNT_10MS        = 20000000 - 1;
    parameter   CNT_2S          = 200 - 1;
    parameter   CHANGE_TIME    = 100 - 1;
    parameter   PWM_OFFSET      = 5000;

    reg     [19:0]  cnt_10ms;
    reg     [7:0]   cnt_2s;
    reg             pwm;
    reg     [19:0]  duty_cycle;
    reg             work_flag;
```

Lab5 - PWM

Counter

```
    always@(posedge clk or negedge rst_n)begin
        if(~rst_n)
            cnt_10ms <= 'd0;
        else if(cnt_10ms == CNT_10MS)
            cnt_10ms <= 'd0;
        else
            cnt_10ms <= cnt_10ms + 1'b1;
    end

    always@(posedge clk or negedge rst_n)begin
        if(~rst_n)
            cnt_2s <= 'd0;
        else if(cnt_10ms == CNT_2S)
            cnt_2s <= 'd0;
        else
            cnt_2s <= cnt_2s + 1'b1;
    end
```

Lab5 - PWM

pwm

```
    always@(posedge clk or negedge rst_n)begin
        if(~rst_n)
            work_flag <= 1'b0;
        else if(cnt_2s == CHANGE_TIME && cnt_10ms == CNT_10MS)
            work_flag <= 1'b1;
        else if(cnt_2s == CNT_2S && cnt_10ms == CNT_10MS)
            work_flag <= 1'b0;
    end

    always@(posedge clk or negedge rst_n)begin
        if(~rst_n)
            duty_cycle <= 'd0;
        else if(work_flag == 1'b0)
            duty_cycle <= (cnt_10ms == CNT_10MS) ? duty_cycle + PWM_OFFSET : duty_cycle;
        else if(work_flag == 1'b1)
            duty_cycle <= (cnt_2s == CNT_2S) ? duty_cycle - PWM_OFFSET : duty_cycle;
    end
```

Lab5 - PWM

Output led

```
    always@(posedge clk or negedge rst_n)begin
        if(~rst_n)
            pwm <= 1'b0;
        else if(cnt_10ms < duty_cycle)
            pwm <= 1'b1;
        else
            pwm <= 1'b0;
    end

   assign  led = ~pwm;

endmodule
```

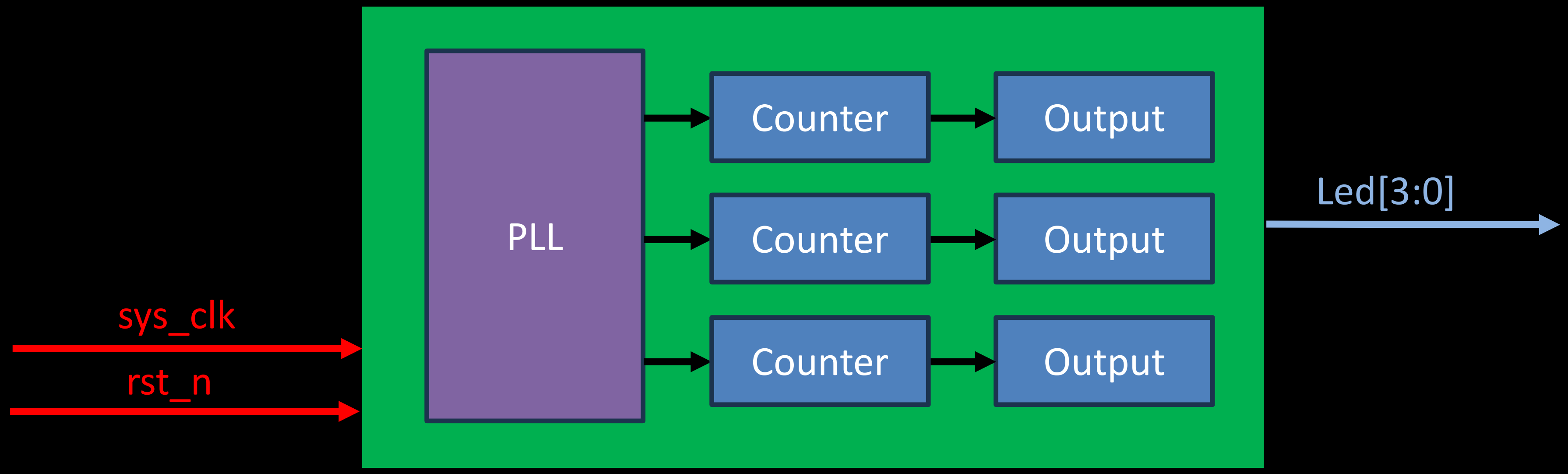
Lab5 - PWM

Constrints

```
set_property PACKAGE_PIN AM39 [get_ports {led[0]}]
set_property PACKAGE_PIN AR40 [get_ports rst_n]
set_property PACKAGE_PIN E19 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports rst_n]
```

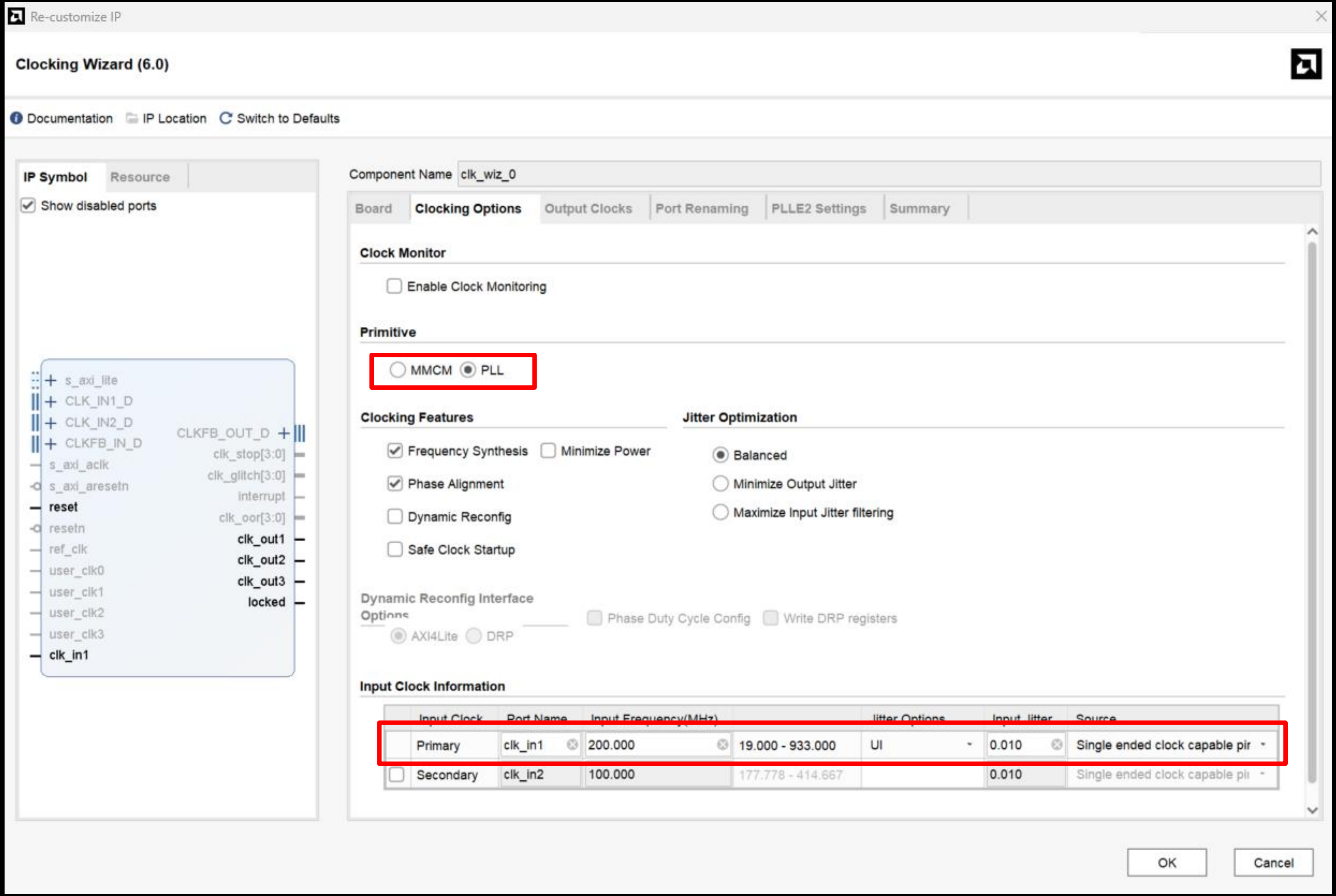
Lab6 - PLL

使用PLL IP使三個LED燈以不同頻率亮滅



Lab6 – PLL IP Setting

PLL



[✓] Show disabled ports

Component Name: clk_wiz_0

Clock Monitor

[ ] Enable Clock Monitoring

Primitive

MMCM [✓ PLL]

Clocking Features

[✓] Frequency Synthesis [ ] Minimize Power

[✓] Phase Alignment

[ ] Dynamic Reconfig

[ ] Safe Clock Startup

Jitter Optimization

[✓ Balanced]

Minimize Output Jitter

Maximize Input Jitter filtering

Dynamic Reconfig Interface Options

AXI4Lite DRP

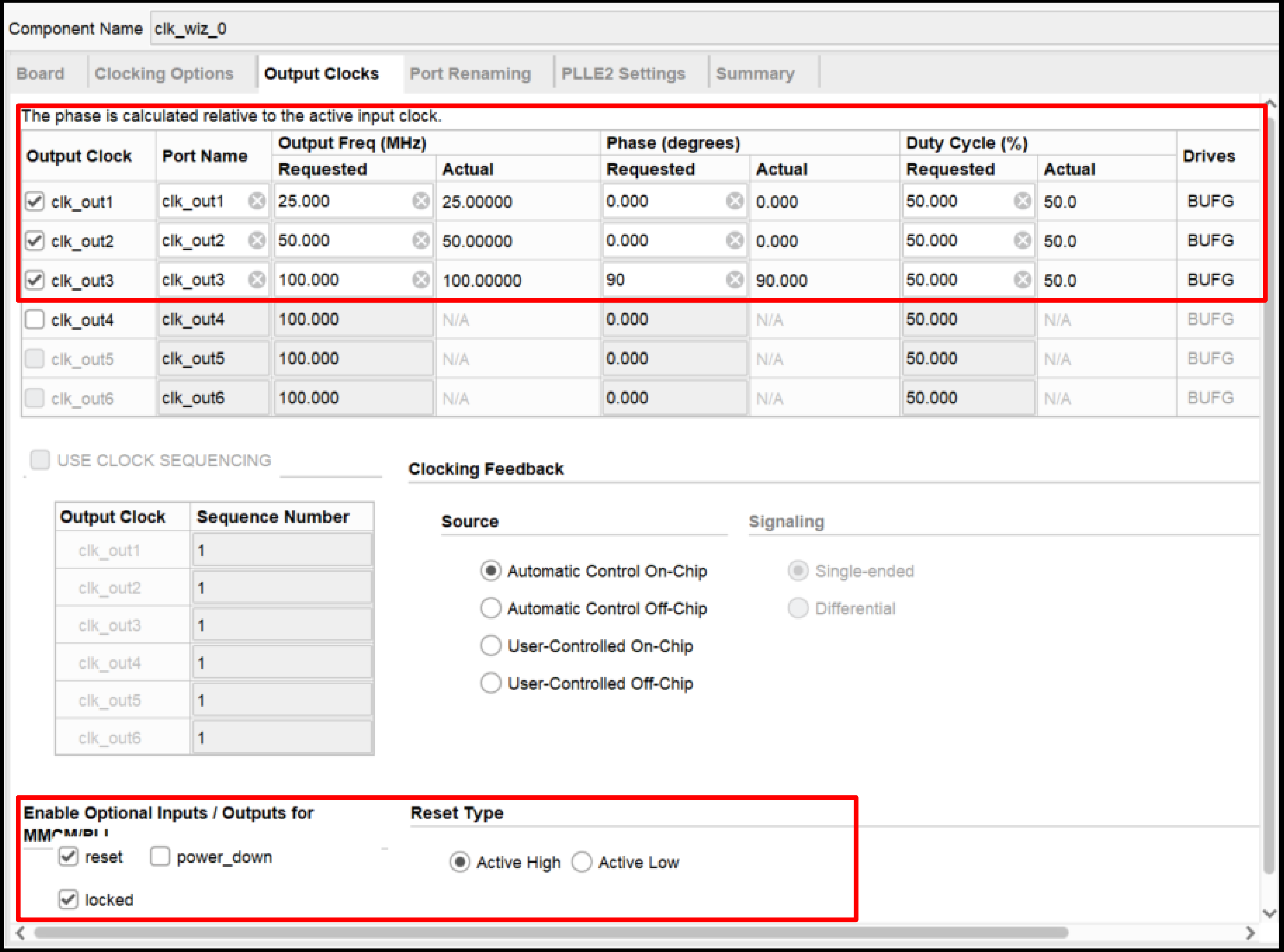
[ ] Phase Duty Cycle Config [ ] Write DRP registers

Input Clock Information

	Input Clock	Port Name	Input Frequency(MHz)		Jitter Options	Input Jitter	Source
	Primary	clk_in1	200.000	19.000 - 933.000	UI	0.010	Single ended clock capable pir
[ ]	Secondary	clk_in2	100.000	177.778 - 414.667		0.010	Single ended clock capable pi

Lab6 – PLL IP Setting

PLL



Component Name clk_wiz_0

Board | Clocking Options | **Output Clocks** | Port Renaming | PLLE2 Settings | Summary

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)		Drives
		Requested	Actual	Requested	Actual	Requested	Actual	
✓ clk_out1	clk_out1	25.000	25.00000	0.000	0.000	50.000	50.0	BUFG
✓ clk_out2	clk_out2	50.000	50.00000	0.000	0.000	50.000	50.0	BUFG
✓ clk_out3	clk_out3	100.000	100.00000	90	90.000	50.000	50.0	BUFG
☐ clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
☐ clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
☐ clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG

☐ USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1

Clocking Feedback

Source

● Automatic Control On-Chip
○ Automatic Control Off-Chip
○ User-Controlled On-Chip
○ User-Controlled Off-Chip

Signaling

● Single-ended
○ Differential

Enable Optional Inputs / Outputs for MMCM/PLL

✓ reset ☐ power_down

✓ locked

Reset Type

[✓ Active High] Active Low

Lab6 - PLL

Port define

```
`timescale 1ns / 1ps
module pll_test(
    input   wire        clk_in1_p     ,
    input   wire        clk_in1_n     ,
    input   wire        rst_n    ,
    output  reg [2:0]  led
    );

    parameter   CNT_MAX = 200_000_000 - 1;

    reg [27:0]  timer1      ;
    reg [27:0]  timer2      ;
    reg [27:0]  timer3      ;
    wire        clk_out1    ;
    wire        clk_out2    ;
    wire        clk_out3    ;
    wire        locked      ;
```

Lab6 - PLL

Counter

```
always@(posedge clk_out1 or negedge rst_n)begin
        if(~rst_n)
            timer1 <= 'd0;
        else if(locked == 1'b1)
            timer1 <= (timer1 == CNT_MAX) ? 'd0 : timer1 + 1'b1;
        else
            timer1 <= 'd0;
    end

    always@(posedge clk_out2 or negedge rst_n)begin
        if(~rst_n)
            timer2 <= 'd0;
        else if(locked == 1'b1)
            timer2 <= (timer2 == CNT_MAX) ? 'd0 : timer2 + 1'b1;
        else
            timer2 <= 'd0;
    end
```

Lab6 - PLL

Counter

```
    always@(posedge clk_out3 or negedge rst_n)begin
        if(~rst_n)
            timer3 <= 'd0;
        else if(locked == 1'b1)
            timer3 <= (timer3 == CNT_MAX) ? 'd0 : timer3 + 1'b1;
        else
            timer3 <= 'd0;
    end
```

Lab6 - PLL

Output Led

```
always@(posedge clk_out1 or negedge rst_n)begin
        if(~rst_n)
            led[0] <= 1'b1;
        else if(timer1 == CNT_MAX)
            led[0] <= ~led[0];
        else
            led[0] <= led[0];
    end

    always@(posedge clk_out2 or negedge rst_n)begin
        if(~rst_n)
            led[1] <= 1'b1;
        else if(timer2 == CNT_MAX)
            led[1] <= ~led[1];
        else
            led[1] <= led[1];
    end
```

Lab6 - PLL

Output Led

```
    always@(posedge clk_out3 or negedge rst_n)begin
        if(~rst_n)
            led[2] <= 1'b1;
        else if(timer3 == CNT_MAX)
            led[2] <= ~led[2];
        else
            led[2] <= led[2];
    end
```

Lab6 - PLL

PLL IP Inst

```
clk_wiz_0 inst_clk(
        // Clock out ports
        .clk_out1(clk_out1),     // output clk_out1
        .clk_out2(clk_out2),     // output clk_out2
        .clk_out3(clk_out3),     // output clk_out3
        // Status and control signals
        .reset(reset), // input reset
        .locked(locked),       // output locked
        // Clock in ports
        .clk_in1_p(clk_in1_p),    // input clk_in1_p
        .clk_in1_n(clk_in1_n)    // input clk_in1_n
);

endmodule
```

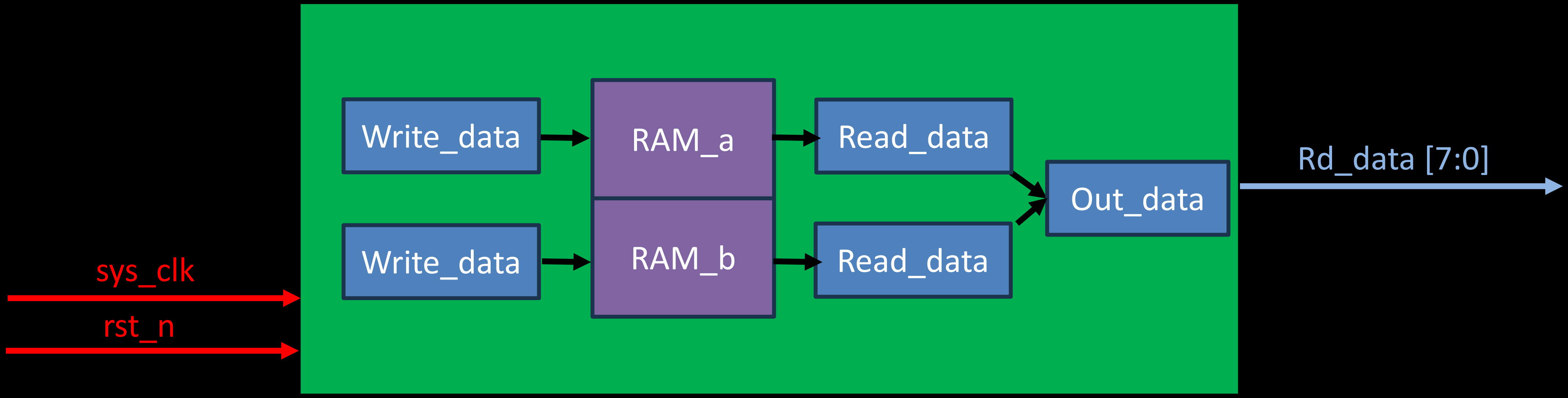
Lab6 - PLL

Constrints

```
set_property PACKAGE_PIN AM39 [get_ports {led[0]}]
set_property PACKAGE_PIN AN39 [get_ports {led[1]}]
set_property PACKAGE_PIN AR37 [get_ports {led[2]}]
set_property PACKAGE_PIN AT37 [get_ports {led[3]}]
set_property PACKAGE_PIN AR40 [get_ports rst_n]
set_property IOSTANDARD LVCMOS18 [get_ports {led[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports rst_n]
set_property BOARD_PART_PIN {clk_p} [get_ports clk_in1_p]
set_property BOARD_PART_PIN {clk_n} [get_ports clk_in1_n]
```

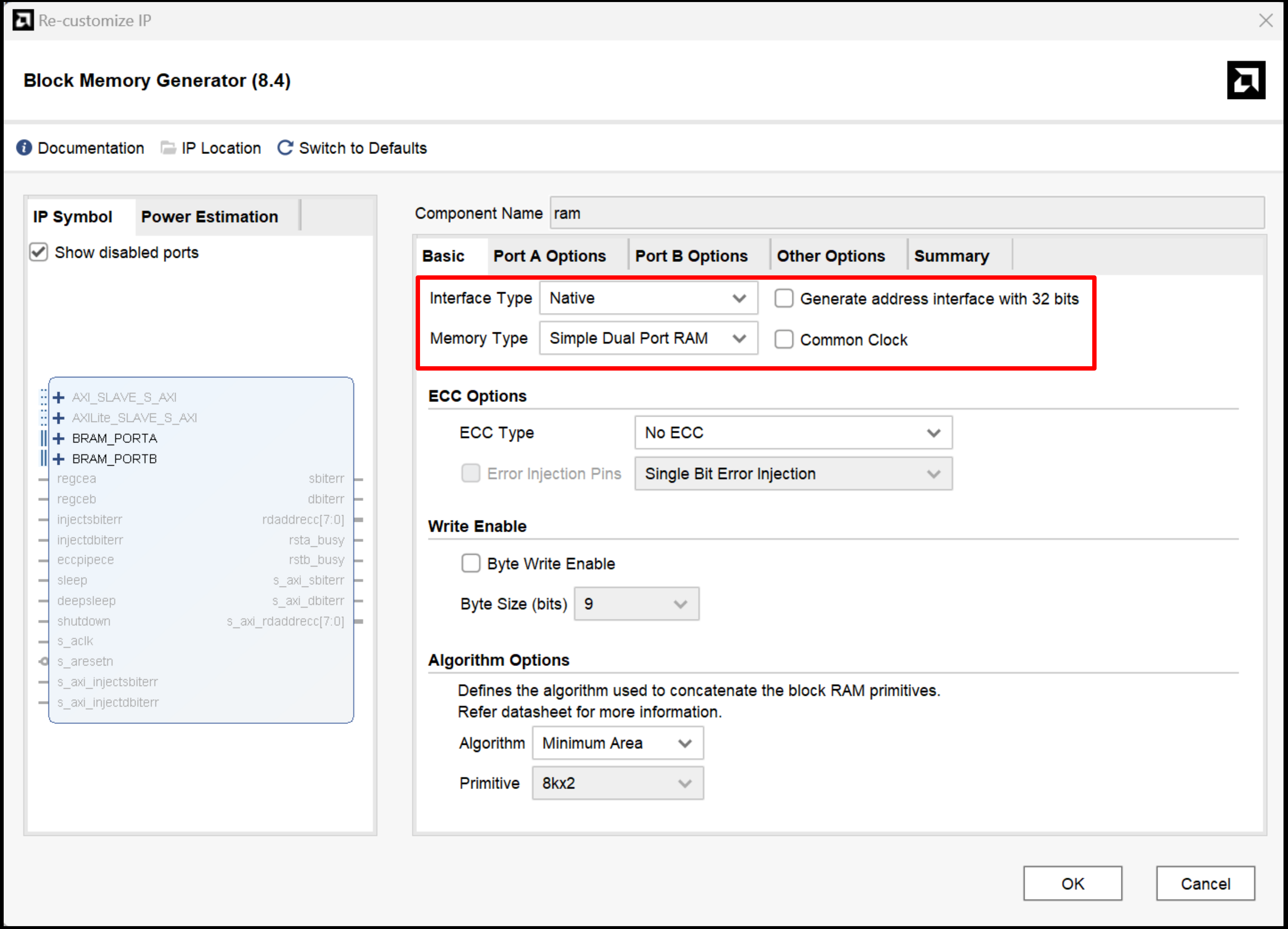
Lab7 - RAM

對RAM IP做資料寫入並從RAM讀取資料



Lab7 – RAM IP Setting

RAM



Re-customize IP

Block Memory Generator (8.4)

Documentation | IP Location | Switch to Defaults

☑ Show disabled ports

Component Name: ram

Basic | Port A Options | Port B Options | Other Options | Summary

Interface Type: Native
☐ Generate address interface with 32 bits
Memory Type: Simple Dual Port RAM
☐ Common Clock

ECC Options

ECC Type: No ECC
☐ Error Injection Pins: Single Bit Error Injection

Write Enable

☐ Byte Write Enable
Byte Size (bits): 9

Algorithm Options

Defines the algorithm used to concatenate the block RAM primitives.
Refer datasheet for more information.

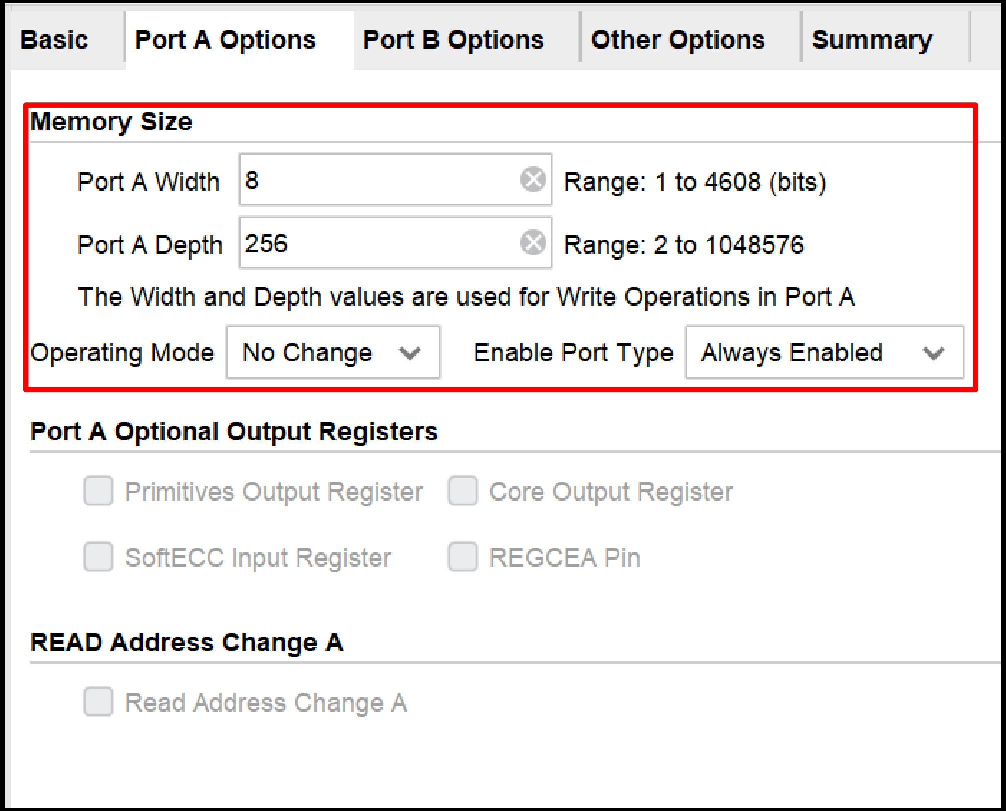
Algorithm: Minimum Area
Primitive: 8kx2

OK | Cancel

AMD

Lab7 – RAM IP Setting

RAM



Basic | Port A Options | Port B Options | Other Options | Summary

Memory Size

Port A Width: 8 — Range: 1 to 4608 (bits)

Port A Depth: 256 — Range: 2 to 1048576

The Width and Depth values are used for Write Operations in Port A

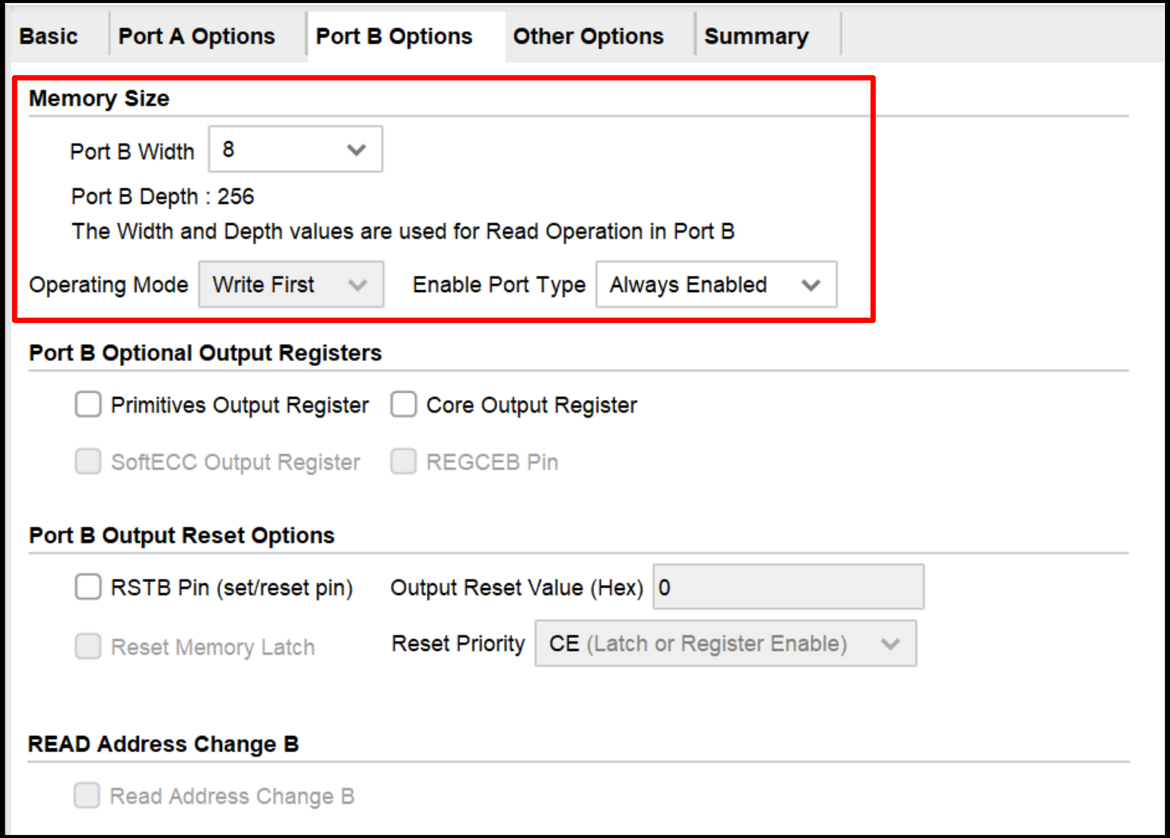
Operating Mode: No Change — Enable Port Type: Always Enabled

Port A Optional Output Registers

- [ ] Primitives Output Register
- [ ] Core Output Register
- [ ] SoftECC Input Register
- [ ] REGCEA Pin

READ Address Change A

- [ ] Read Address Change A



Basic | Port A Options | Port B Options | Other Options | Summary

Memory Size

Port B Width: 8

Port B Depth : 256

The Width and Depth values are used for Read Operation in Port B

Operating Mode: Write First — Enable Port Type: Always Enabled

Port B Optional Output Registers

- [ ] Primitives Output Register
- [ ] Core Output Register
- [ ] SoftECC Output Register
- [ ] REGCEB Pin

Port B Output Reset Options

- [ ] RSTB Pin (set/reset pin) — Output Reset Value (Hex): 0
- [ ] Reset Memory Latch — Reset Priority: CE (Latch or Register Enable)

READ Address Change B

- [ ] Read Address Change B

Lab7 – RAM IP Setting

RAM

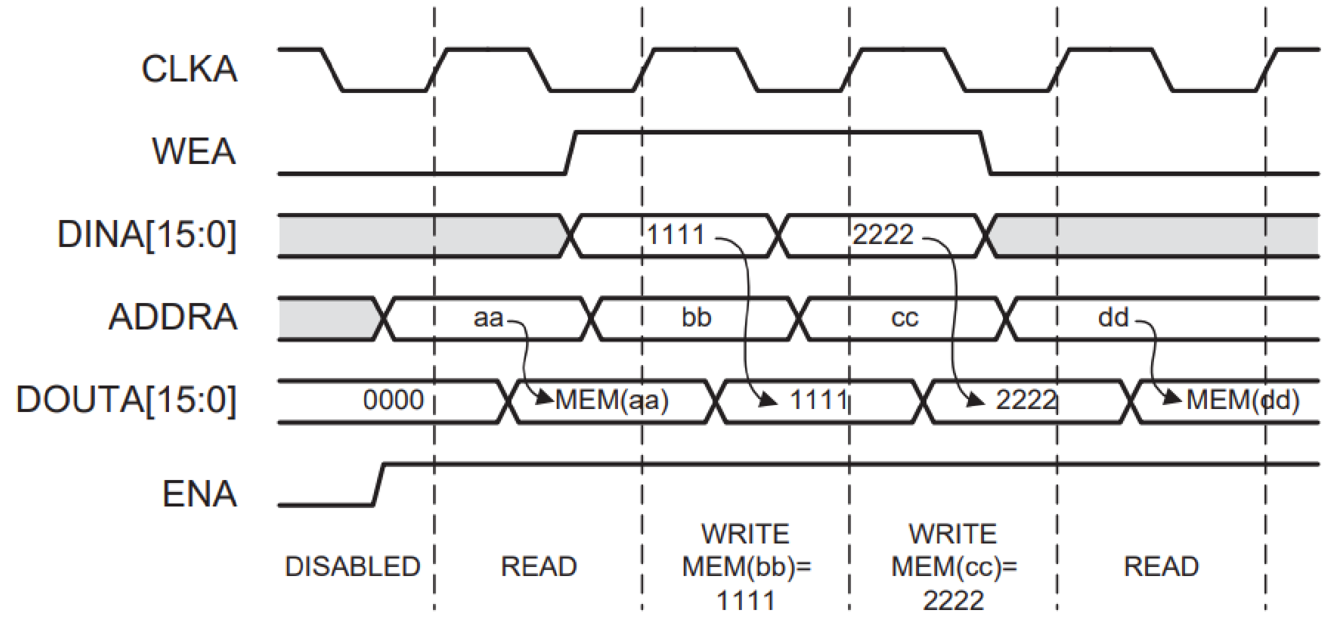


Figure 3-9: **Write First Mode Example**

Lab7 - RAM

Port define

```
`timescale 1ns / 1ps
module ram_pp(
    input   wire        clk     ,
    input   wire        rst_n   ,
    output  reg   [7:0] rd_data
    );

parameter   MAX = 256 - 1;
reg                 wr_rama_en      ;
reg                 wr_ramb_en      ;
reg       [7:0]     wr_addr_a       ;
reg       [7:0]     rd_addr_a       ;
wire      [7:0]     din_a           ;
wire      [7:0]     dout_a          ;
reg       [7:0]     wr_addr_b       ;
reg       [7:0]     rd_addr_b       ;
wire      [7:0]     din_b           ;
wire      [7:0]     dout_b          ;
reg                 wr_rama_dd      ;
```

Lab7 - RAM

Ram Enable

```
always@(posedge clk or negedge rst_n)begin
    if(rst_n == 1'b0)
        wr_rama_en <= 1'b0;
    else if((wr_addr_b == MAX) && (wr_rama_en == 1'b0))
        wr_rama_en <= 1'b1;
    else if((wr_addr_a == MAX) && (wr_rama_en == 1'b1))
        wr_rama_en <= 1'b0;
    else
        wr_rama_en <= wr_rama_en;
end

always@(*)begin
    wr_ramb_en <= ~wr_rama_en;
end
```

Lab7 - RAM

Ram_a Read & Write

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        wr_addr_a <= 'd0;
    else if(wr_rama_en)
        wr_addr_a <= (wr_addr_a == MAX) ? 'd0 : wr_addr_a + 1'b1;
    else
        wr_addr_a <= 'd0;
end

assign din_a = wr_addr_a;

always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        rd_addr_a <= 'd0;
    else if(wr_rama_en)
        rd_addr_a <= (rd_addr_a == MAX) ? 'd0 : rd_addr_a + 1'b1;
    else
        rd_addr_a <= 'd0;
end
```

Lab7 - RAM

Ram_b Read & Write

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        wr_addr_b <= 'd0;
    else if(wr_ramb_en)
        wr_addr_b <= (wr_addr_b == MAX) ? 'd0 : wr_addr_b + 1'b1;
    else
        wr_addr_b <= 'd0;
end

assign din_b = wr_addr_b;

always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        rd_addr_b <= 'd0;
    else if(wr_ramb_en)
        rd_addr_b <= (rd_addr_b == MAX) ? 'd0 : rd_addr_b + 1'b1;
    else
        rd_addr_b <= 'd0;
end
```

Lab7 - RAM

Ram Inst

```
ram ram_a (
  .clka(clk),    // input wire clka
  .wea(wr_rama_en),       // input wire [0 : 0] wea
  .addra(wr_addr_a),  // input wire [7 : 0] addra
  .dina(din_a),    // input wire [7 : 0] dina
  .clkb(clk),    // input wire clkb
  .addrb(rd_addr_a),  // input wire [7 : 0] addrb
  .doutb(dout_a)  // output wire [7 : 0] doutb
);

ram ram_b (
  .clka(clk),    // input wire clka
  .wea(wr_ramb_en),       // input wire [0 : 0] wea
  .addra(wr_addr_b),  // input wire [7 : 0] addra
  .dina(din_b),    // input wire [7 : 0] dina
  .clkb(clk),    // input wire clkb
  .addrb(rd_addr_b),  // input wire [7 : 0] addrb
  .doutb(dout_b)  // output wire [7 : 0] doutb
);
```

Lab7 - RAM

Output

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        wr_rama_dd <= 1'b0;
    else
        wr_rama_dd <= wr_rama_en;
end

always@(*)begin
    if(wr_rama_dd)
        rd_data = dout_b;
    else
        rd_data = dout_a;
end
endmodule
```

Lab7 - RAM

tb port define & inst

```
`timescale 1ns / 1ps
module tb_ram_pp();
parameter   MAX = 256 - 1;

reg         clk;
reg         rst_n;
wire [7:0]  rd_data;

ram_pp #(
    .MAX(MAX))
inst_ram_pp(
    .clk(clk),
    .rst_n(rst_n),
    .rd_data(rd_data));
```

Lab7 - RAM

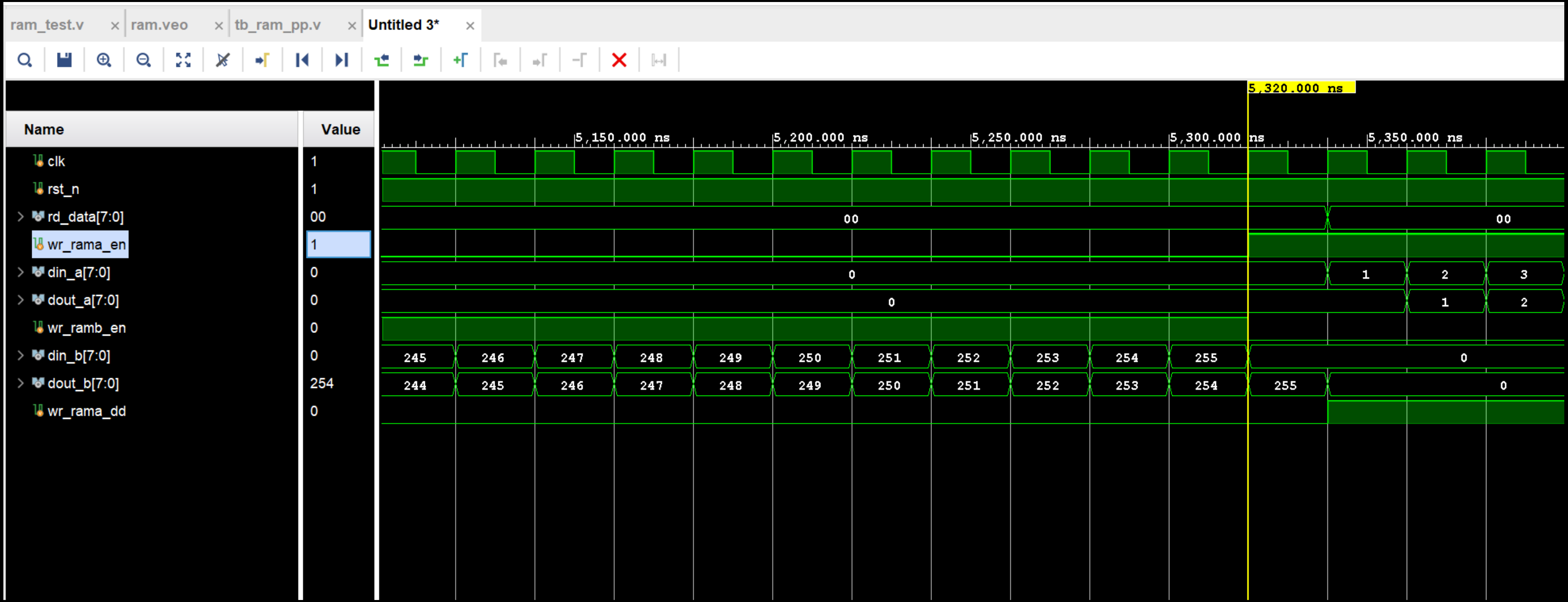
tb initial block

```
initial begin
    clk = 1;
    forever#(10) clk = ~clk ;
end

initial begin
    rst_n <= 0;
    #200
    rst_n <= 1;
end
endmodule
```

Lab7 - RAM

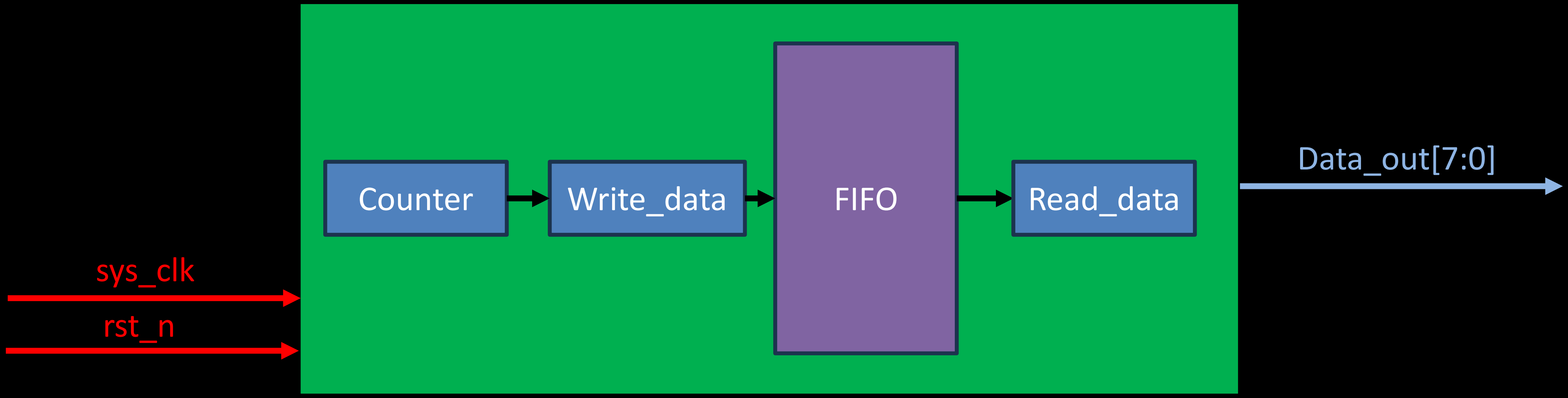
Simulation



Name	Value
clk	1
rst_n	1
rd_data[7:0]	00
wr_rama_en	1
din_a[7:0]	0
dout_a[7:0]	0
wr_ramb_en	0
din_b[7:0]	0
dout_b[7:0]	254
wr_rama_dd	0

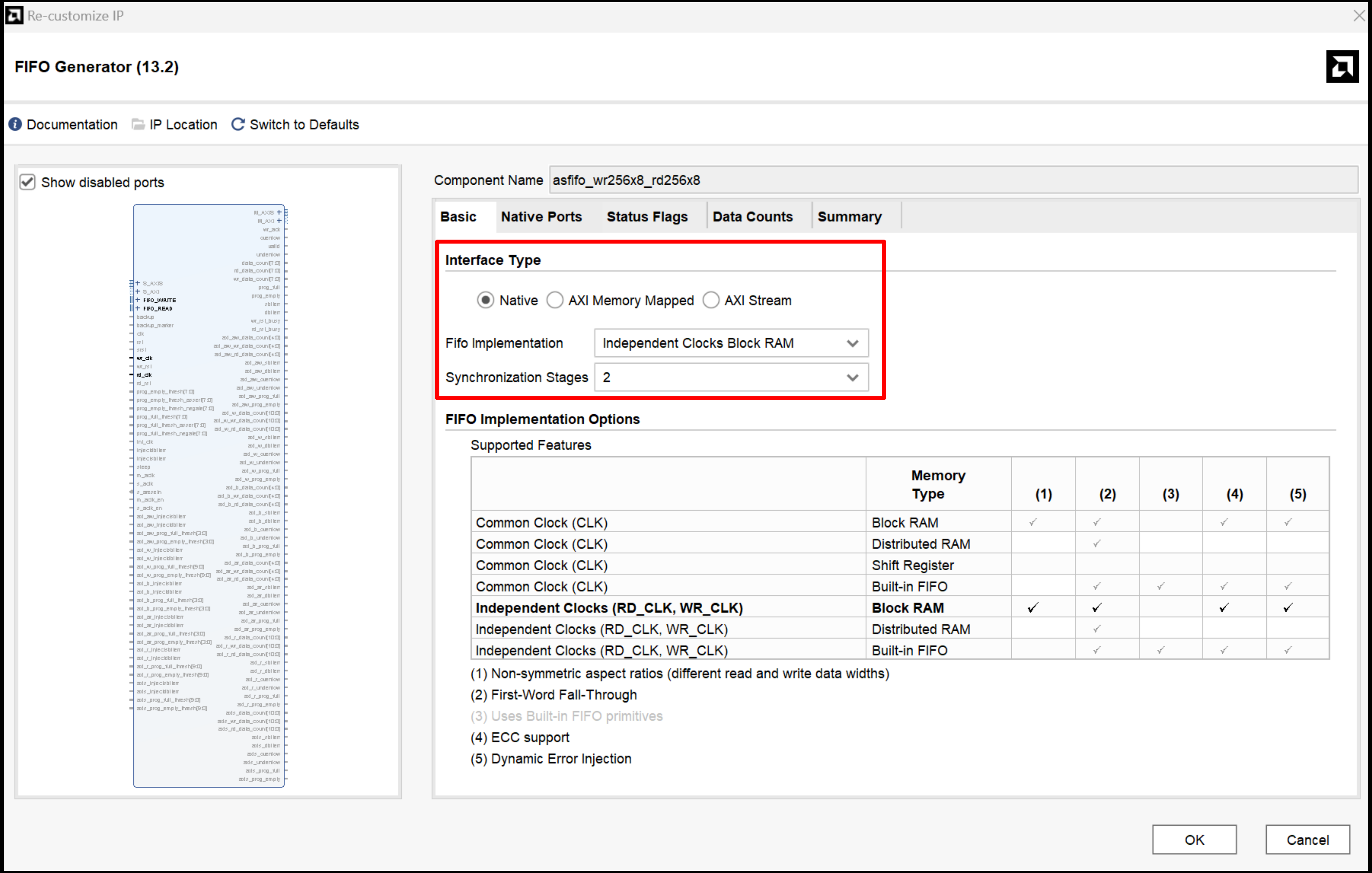
Lab8 - FIFO

對FIFO IP做資料寫入並從FIFO讀取資料



Lab8 – FIFO IP Setting

FIFO



Re-customize IP

FIFO Generator (13.2)

Documentation | IP Location | Switch to Defaults

☑ Show disabled ports

Component Name: asfifo_wr256x8_rd256x8

Basic | Native Ports | Status Flags | Data Counts | Summary

Interface Type

[✓ Native] AXI Memory Mapped AXI Stream

Fifo Implementation: Independent Clocks Block RAM

Synchronization Stages: 2

FIFO Implementation Options

Supported Features

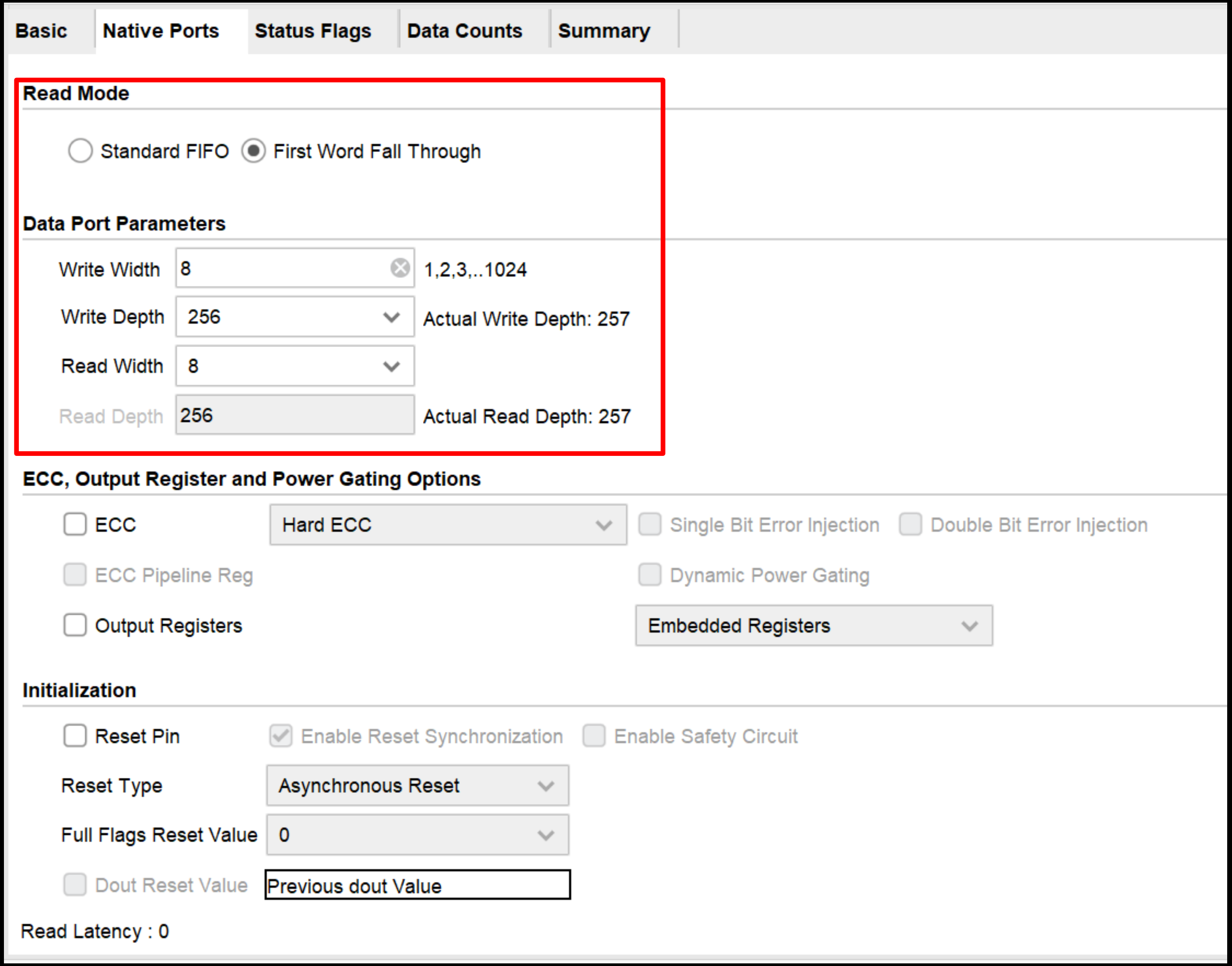
	Memory Type	(1)	(2)	(3)	(4)	(5)
Common Clock (CLK)	Block RAM	✓	✓		✓	✓
Common Clock (CLK)	Distributed RAM		✓			
Common Clock (CLK)	Shift Register					
Common Clock (CLK)	Built-in FIFO		✓	✓	✓	✓
Independent Clocks (RD_CLK, WR_CLK)	**Block RAM**	✓	✓		✓	✓
Independent Clocks (RD_CLK, WR_CLK)	Distributed RAM		✓			
Independent Clocks (RD_CLK, WR_CLK)	Built-in FIFO		✓	✓	✓	✓

(1) Non-symmetric aspect ratios (different read and write data widths)
(2) First-Word Fall-Through
(3) Uses Built-in FIFO primitives
(4) ECC support
(5) Dynamic Error Injection

OK | Cancel

Lab8 – FIFO IP Setting

FIFO



Basic | Native Ports | Status Flags | Data Counts | Summary

Read Mode

Standard FIFO [✓ First Word Fall Through]

Data Port Parameters

Write Width	8	1,2,3,..1024
Write Depth	256	Actual Write Depth: 257
Read Width	8	
Read Depth	256	Actual Read Depth: 257

ECC, Output Register and Power Gating Options

- ☐ ECC — Hard ECC — ☐ Single Bit Error Injection — ☐ Double Bit Error Injection
- ☐ ECC Pipeline Reg — ☐ Dynamic Power Gating
- ☐ Output Registers — Embedded Registers

Initialization

- ☐ Reset Pin — ✓ Enable Reset Synchronization — ☐ Enable Safety Circuit
- Reset Type: Asynchronous Reset
- Full Flags Reset Value: 0
- ☐ Dout Reset Value: Previous dout Value

Read Latency : 0

Lab8 – FIFO IP Setting

FIFO

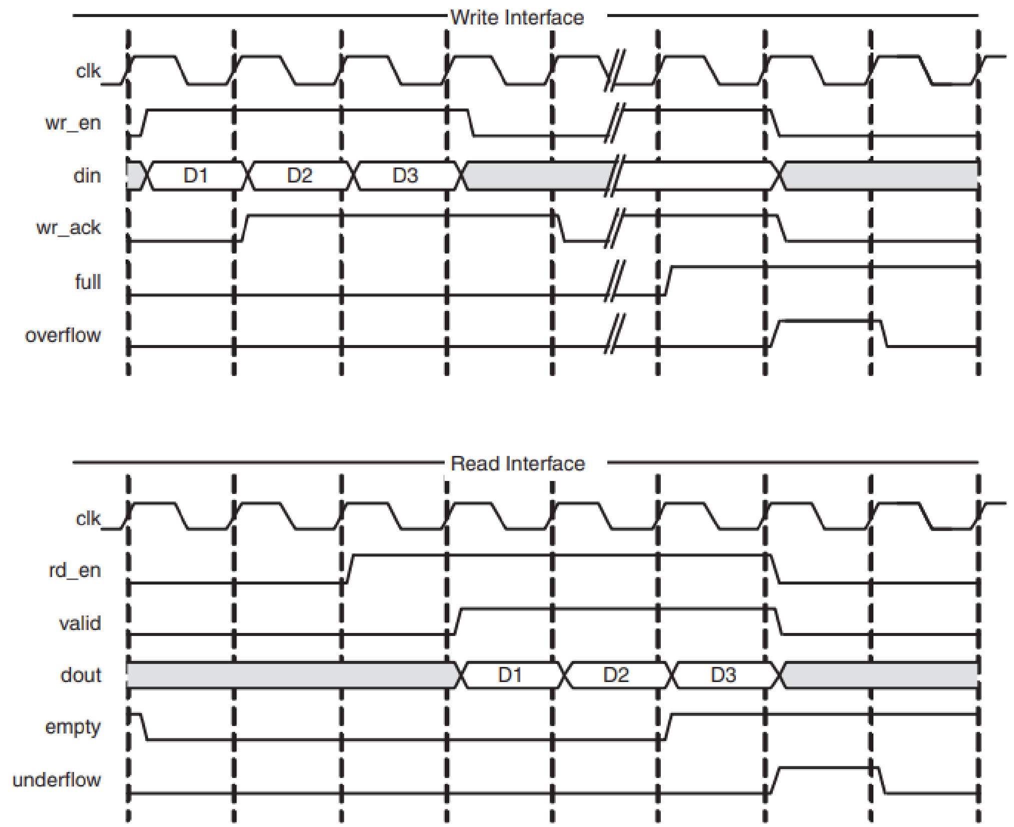


Figure 3-7: **Handshaking Signals for a FIFO with Common Clocks**

Lab8 - FIFO

Port define

```
`timescale 1ns / 1ps
module fifo_test(
    input   wire        clk      ,
    input   wire        rst_n    ,
    output  wire [7:0]  data_out
    );

parameter   MAX         =   256-1;
parameter   RD_START    =   128-1;
reg              wr_en         ;
reg              wr_flag       ;
reg    [8:0]     wr_cnt;       ;
reg    [7:0]     wr_data       ;
wire             full,empty    ;
reg              rd_en         ;
reg              rd_start      ;
wire   [7:0]     rd_data       ;
```

Lab8 - FIFO

Counter

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        wr_cnt <= 1'b0;
    else if(wr_flag)
        wr_cnt <= (wr_cnt == MAX) ? 'd0 : wr_cnt + 1'b1;
    else
        wr_cnt <= 'd0;
end
```

Lab8 - FIFO

Flag

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        wr_flag <= 1'b1;
    else if(wr_cnt == MAX && wr_flag == 1'b1)
        wr_flag <= 1'b0;
    else if(empty == 1'b1)
        wr_flag <= 1'b1;
    else
        wr_flag <= wr_flag;
end

always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        wr_en <= 1'b0;
    else
        wr_en <= wr_flag;
end
```

Lab8 - FIFO

Write_Data

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        wr_data <= 'd0;
    else
        wr_data <= wr_cnt;
end
```

Lab8 - FIFO

Read_Data

```
always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        rd_start <= 1'b0;
    else if(wr_cnt == RD_START)
        rd_start <= 1'b1;
    else
        rd_start <= 1'b0;
end

always@(posedge clk or negedge rst_n)begin
    if(~rst_n)
        rd_en <= 1'b0;
    else if(rd_start)
        rd_en <= 1'b1;
    else if(empty)
        rd_en <= 1'b0;
end
```

Lab8 - FIFO

FIFO Inst

```
asfifo_wr256x8_rd256x8 fifo_inst (
  .wr_clk(clk),  // input wire wr_clk
  .rd_clk(clk),  // input wire rd_clk
  .din(wr_data),        // input wire [7 : 0] din
  .wr_en(wr_en),    // input wire wr_en
  .rd_en(rd_en),    // input wire rd_en
  .dout(rd_data),       // output wire [7 : 0] dout
  .full(full),      // output wire full
  .empty(empty)    // output wire empty
);
```

Lab8 - FIFO

tb port define & inst

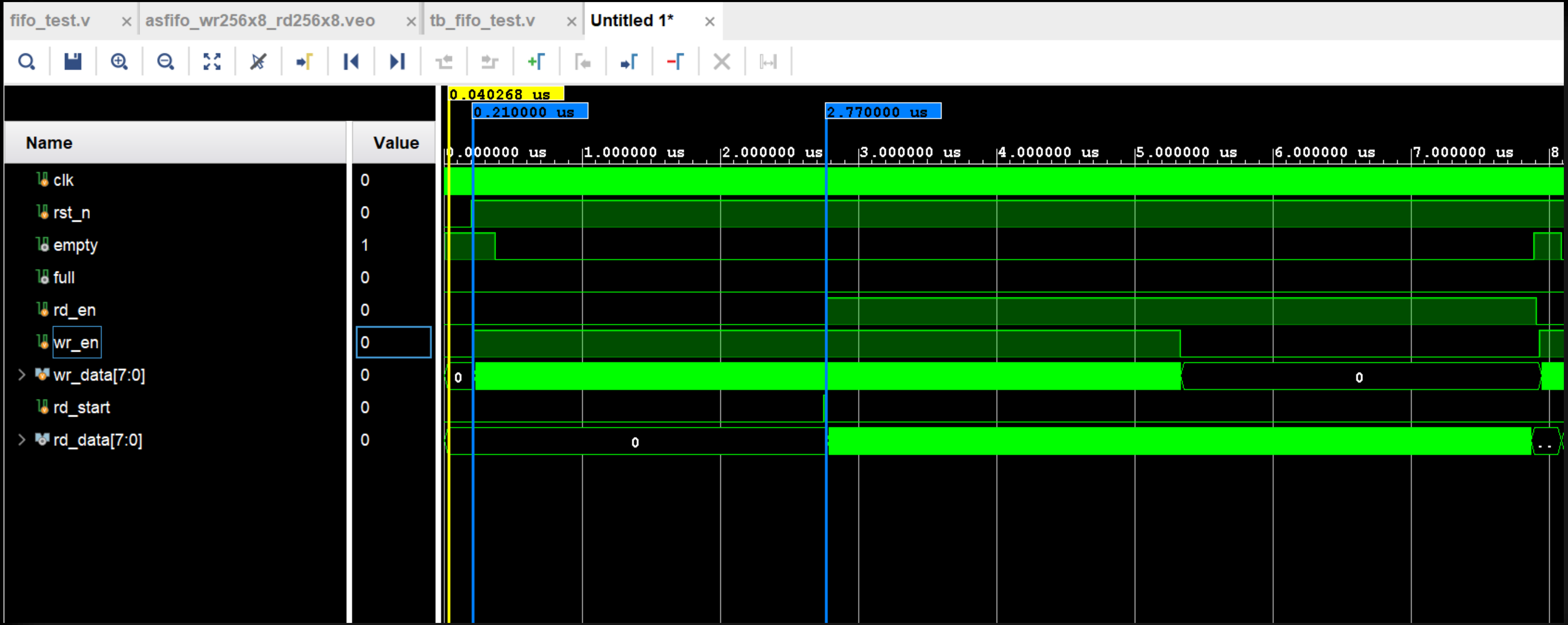
```
`timescale 1ns / 1ps
module tb_fifo_test();

    parameter   MAX          =   256-1;
    parameter   RD_START     =   128-1;
    reg         clk     ;
    reg         rst_n   ;
    wire [7:0]  data_out ;

    fifo_test#(
        .MAX(MAX),
        .RD_START(RD_START))
    inst_fifo_test(
        .clk(clk),
        .rst_n(rst_n),
        .data_out(data_out));
```

Lab8 - FIFO

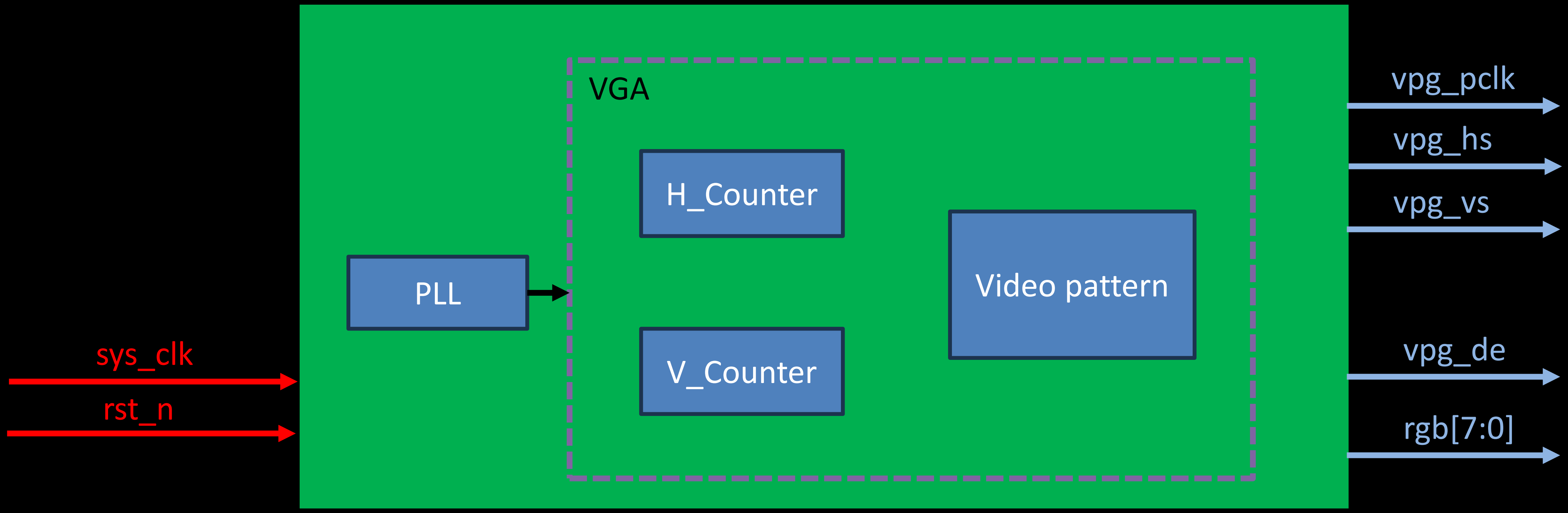
Simulation



Name	Value
clk	0
rst_n	0
empty	1
full	0
rd_en	0
wr_en	0
wr_data[7:0]	0
rd_start	0
rd_data[7:0]	0

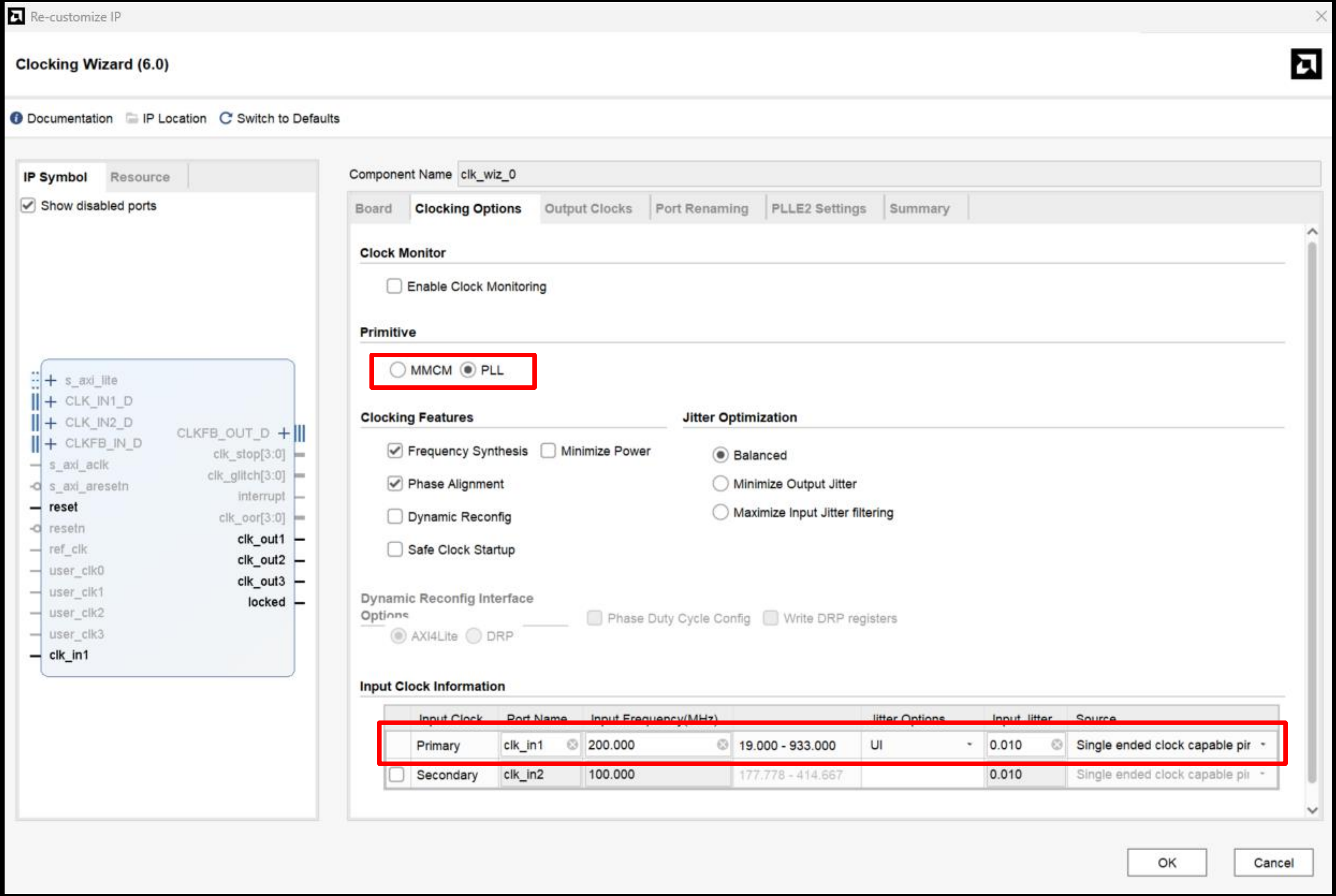
Lab9 - VGA

輸出固定的VGA影像



Lab9 – VGA

PLL



[✓] Show disabled ports

Component Name: clk_wiz_0

Clock Monitor

[ ] Enable Clock Monitoring

Primitive

MMCM [✓ PLL]

Clocking Features

[✓] Frequency Synthesis [ ] Minimize Power

[✓] Phase Alignment

[ ] Dynamic Reconfig

[ ] Safe Clock Startup

Jitter Optimization

[✓ Balanced]

Minimize Output Jitter

Maximize Input Jitter filtering

Dynamic Reconfig Interface Options

[✓ AXI4Lite] DRP

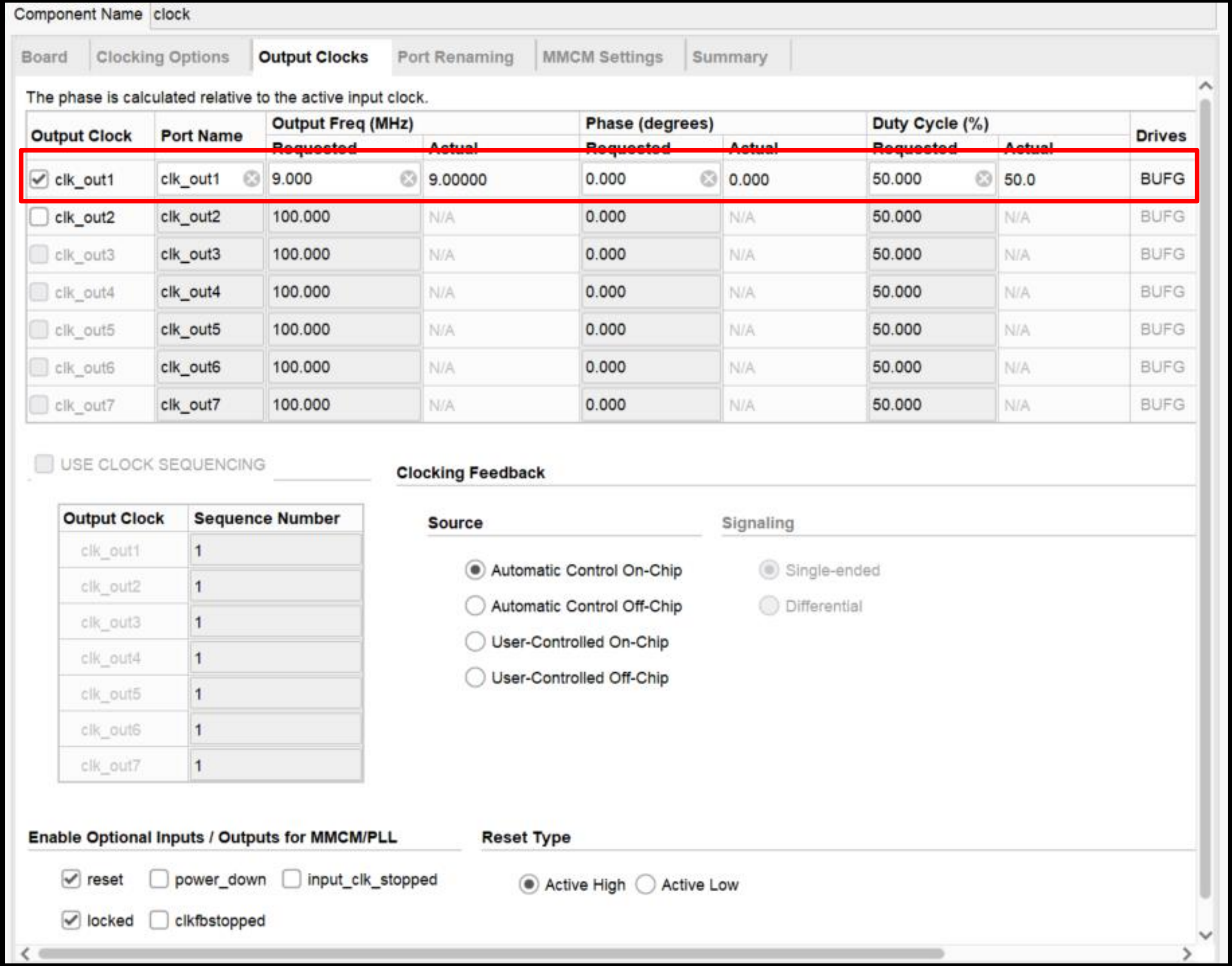
[ ] Phase Duty Cycle Config [ ] Write DRP registers

Input Clock Information

	Input Clock	Port Name	Input Frequency(MHz)		Jitter Options	Input Jitter	Source
	Primary	clk_in1	200.000	19.000 - 933.000	UI	0.010	Single ended clock capable pir
[ ]	Secondary	clk_in2	100.000	177.778 - 414.667		0.010	Single ended clock capable pi

Lab9 – VGA

PLL



Component Name clock

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)		Drives
		Requested	Actual	Requested	Actual	Requested	Actual	
✓ clk_out1	clk_out1	9.000	9.00000	0.000	0.000	50.000	50.0	BUFG
☐ clk_out2	clk_out2	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
☐ clk_out3	clk_out3	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
☐ clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
☐ clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
☐ clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
☐ clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	BUFG

☐ USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

Source

- [✓ Automatic Control On-Chip]
- Automatic Control Off-Chip
- User-Controlled On-Chip
- User-Controlled Off-Chip

Signaling

- [✓ Single-ended]
- Differential

Enable Optional Inputs / Outputs for MMCM/PLL

✓ reset ☐ power_down ☐ input_clk_stopped

✓ locked ☐ clkfbstopped

Reset Type

[✓ Active High] Active Low

Lab9 - VGA

Port define

```
`timescale 1ns / 1ps
module vga_shift(
input  wire          clk       ,
input  wire          rst_n     ,
output wire          vpg_pclk  ,
output reg           vpg_de    ,
output wire          vpg_disp  ,
output reg           vpg_hs    ,
output reg           vpg_vs    ,
output reg [23:0]    rgb
    );
wire         rst     ;
reg   [12:0] cnt_h   ;
reg   [12:0] cnt_v   ;
reg   [11:0] x       ;
reg          flag_x  ;
reg   [11:0] y       ;
reg          flag_y  ;
wire         locked1 ;
```

Lab9 - VGA

Parameter

```
parameter    H_TOTAL = 525 - 1    ;
parameter    H_SYNC  = 41 - 1     ;
parameter    H_START = 43 - 1     ;
parameter    H_END = 523 - 1      ;
parameter    V_TOTAL = 286 - 1    ;
parameter    V_SYNC = 10 - 1      ;
parameter    V_START = 12 - 1     ;
parameter    V_END = 284 - 1      ;
parameter    SQUARE_X = 150       ;
parameter    SQUARE_Y = 150       ;
parameter    SCREEN_X = 480       ;
parameter    SCREEN_Y = 272       ;
```

Lab9 - VGA

CLK Inst

```
assign vpg_disp = 1'b1;
assign rst = ~rst_n;

clock instance_name
   (
    // Clock out ports
    .clk_out1(vpg_pclk),     // output clk_out1
    // Status and control signals
    .reset(rst), // input reset
    .locked(locked),       // output locked
   // Clock in ports
    .clk_in1(clk)      // input clk_in1
);
```

Lab9 - VGA

Counter

```
always@(posedge vpg_pclk)begin
    if(rst)
        cnt_h <= 'd0;
    else if(cnt_h == H_TOTAL)
        cnt_h <= 'd0;
    else
        cnt_h <= cnt_h + 1'b1;
end

always@(posedge vpg_pclk)begin
    if(rst)
        cnt_v <= 'd0;
    else if(cnt_v == V_TOTAL && cnt_h == H_TOTAL)
        cnt_v <= 'd0;
    else if(cnt_h == H_TOTAL)
        cnt_v <= cnt_v + 1'b1;
end
```

Lab9 - VGA

H & V Sync Flag

```
always@(posedge vpg_pclk)begin
    if(rst)
        vpg_hs <= 1'b1;
    else if(cnt_h == H_TOTAL)
        vpg_hs <= 1'b1;
    else if(cnt_h == H_TOTAL)
        vpg_hs <= 1'b0;
end

always@(posedge vpg_pclk)begin
    if(rst)
        vpg_vs <= 1'b1;
    else if(cnt_v == V_TOTAL && cnt_h == H_TOTAL)
        vpg_vs <= 1'b1;
    else if(cnt_v == V_SYNC && cnt_h == H_TOTAL)
        vpg_vs <= 1'b0;
end
```

Lab9 - VGA

VGA Enable Signal

```
always@(posedge vpg_pclk)begin
    if(rst)
        vpg_de <= 1'b1;
    else if((cnt_h >= H_START) && (cnt_h < H_END) && (cnt_v
>= V_START) && (cnt_v < V_END))
        vpg_de <= 1'b1;
    else
        vpg_de <= 1'b0;
end
```

Lab9 - VGA

Output Video X-axis

```
always@(posedge vpg_pclk)begin
    if(rst)
        x <= 'd0;
    else if(flag_x == 1'b0 && cnt_v == V_TOTAL && cnt_h == H_TOTAL)
        x <= x + 1'b1;
    else if(flag_x == 1'b1 && cnt_v == V_TOTAL && cnt_h == H_TOTAL)
        x <= x - 1'b1;
end

always@(posedge vpg_pclk)begin
    if(rst)
        flag_x <= 1'b0;
    else if(flag_x == 1'b0 && cnt_v == V_TOTAL && cnt_h == H_TOTAL && x ==(H_END - H_START - SQUARE_X - 1'b1))
        flag_x <= 1'b1;
    else if(flag_x == 1'b1 && cnt_v == V_TOTAL && cnt_h == H_TOTAL)
        flag_x <= 1'b0;
end
```

Lab9 - VGA

Output Video Y-axis

```
always@(posedge vpg_pclk)begin
    if(rst)
        y <= 'd0;
    else if(flag_y == 1'b0 && cnt_v == V_TOTAL && cnt_h == H_TOTAL)
        y <= y + 1'b1;
    else if(flag_y == 1'b1 && cnt_v == V_TOTAL && cnt_h == H_TOTAL)
        y <= y - 1'b1;
end

always@(posedge vpg_pclk)begin
    if(rst)
        flag_y <= 1'b0;
    else if(flag_y == 1'b0 && cnt_v == V_TOTAL && cnt_h == H_TOTAL && y ==(V_END - V_START - SQUARE_Y - 1'b1))
        flag_y <= 1'b1;
    else if(flag_y == 1'b1 && cnt_v == V_TOTAL && cnt_h == H_TOTAL && y =='d1)
        flag_y <= 1'b0;
end
```

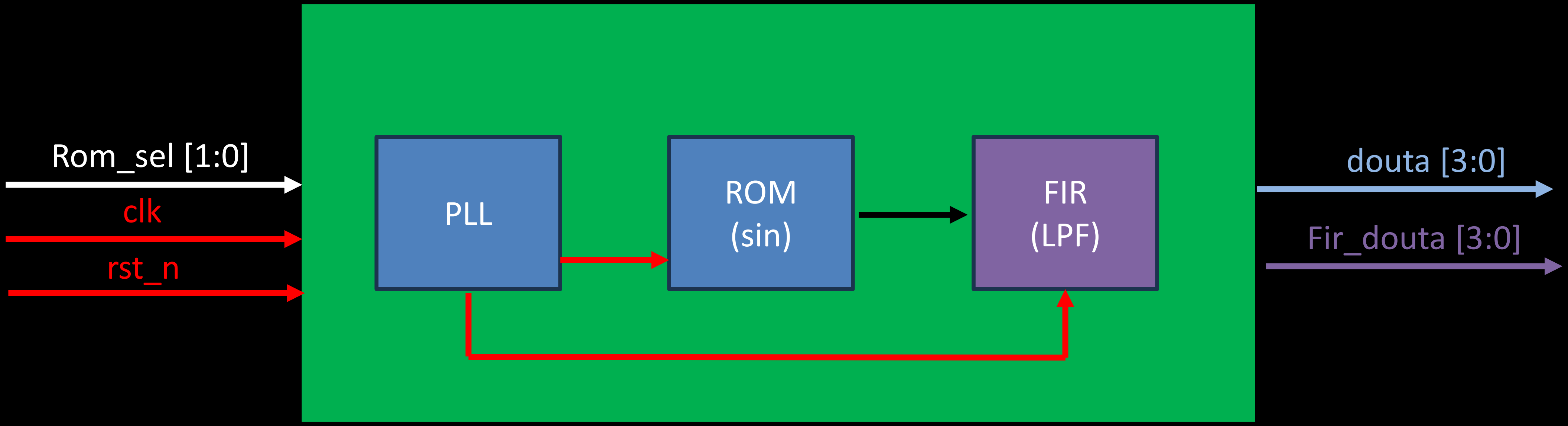
Lab9 - VGA

Output RGB Block

```
always@(posedge vpg_pclk)begin
    if(rst)
        rgb <= 'd0;
    else if(cnt_h >= H_START+x && cnt_h <= H_START+SQUARE_X+x && cnt_v >= V_START+y && cnt_v <= V_START+SQUARE_Y+y)
        rgb <= 24'hFFB6C1;
    else if(cnt_h >= H_START && cnt_h < H_END && cnt_v >= V_START && cnt_v <= V_END && cnt_h[4:0] >= 'd20)
        rgb <= 24'h00FF00;
    else if(cnt_h>=H_START && cnt_h<H_END && cnt_v>=V_START && cnt_v<=V_END && (cnt_h[4:0]>='d10 && cnt_h[2:0]<'d20))
        rgb <= 24'h0000FF;
    else if(cnt_h >= H_START && cnt_h < H_END && cnt_v >= V_START && cnt_v <= V_END && cnt_h[4:0] < 'd10)
        rgb <= 24'hFF0000;
    else
        rgb <= 'd0;
end
endmodule
```

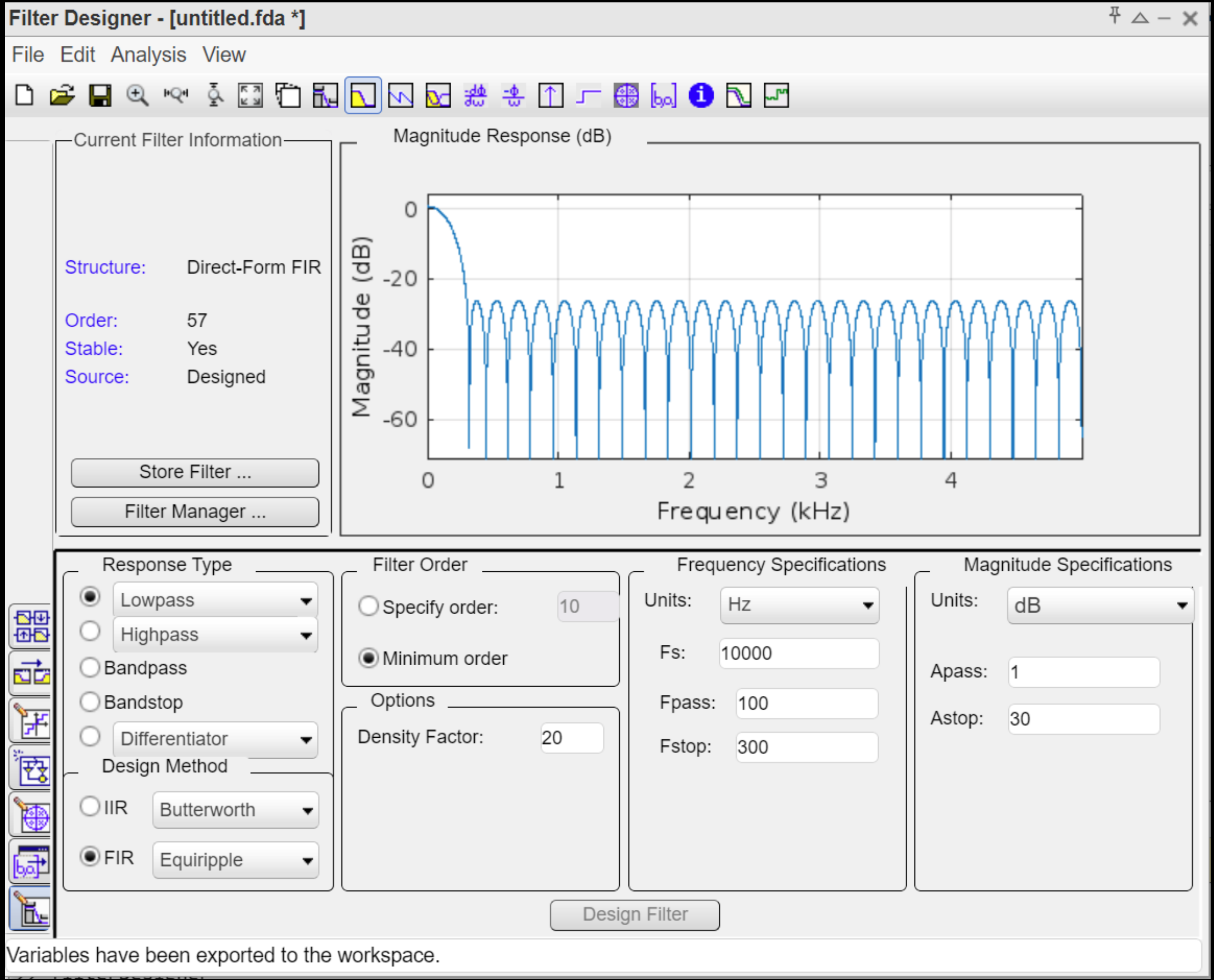
Lab10 – FIR Low-pass filter

設計一個低通濾波器電路



Lab10 - FIR Low-pass filter

用Matlab製作FIR LPF (生成fir.coe)



Current Filter Information

Structure: Direct-Form FIR

Order: 57

Stable: Yes

Source: Designed

Response Type
[✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method
IIR Butterworth
[✓ FIR] Equiripple

Filter Order
Specify order: 10
[✓ Minimum order]

Options
Density Factor: 20

Frequency Specifications
Units: Hz
Fs: 10000
Fpass: 100
Fstop: 300

Magnitude Specifications
Units: dB
Apass: 1
Astop: 30

Variables have been exported to the workspace.

AMD

Lab10 - FIR Low-pass filter

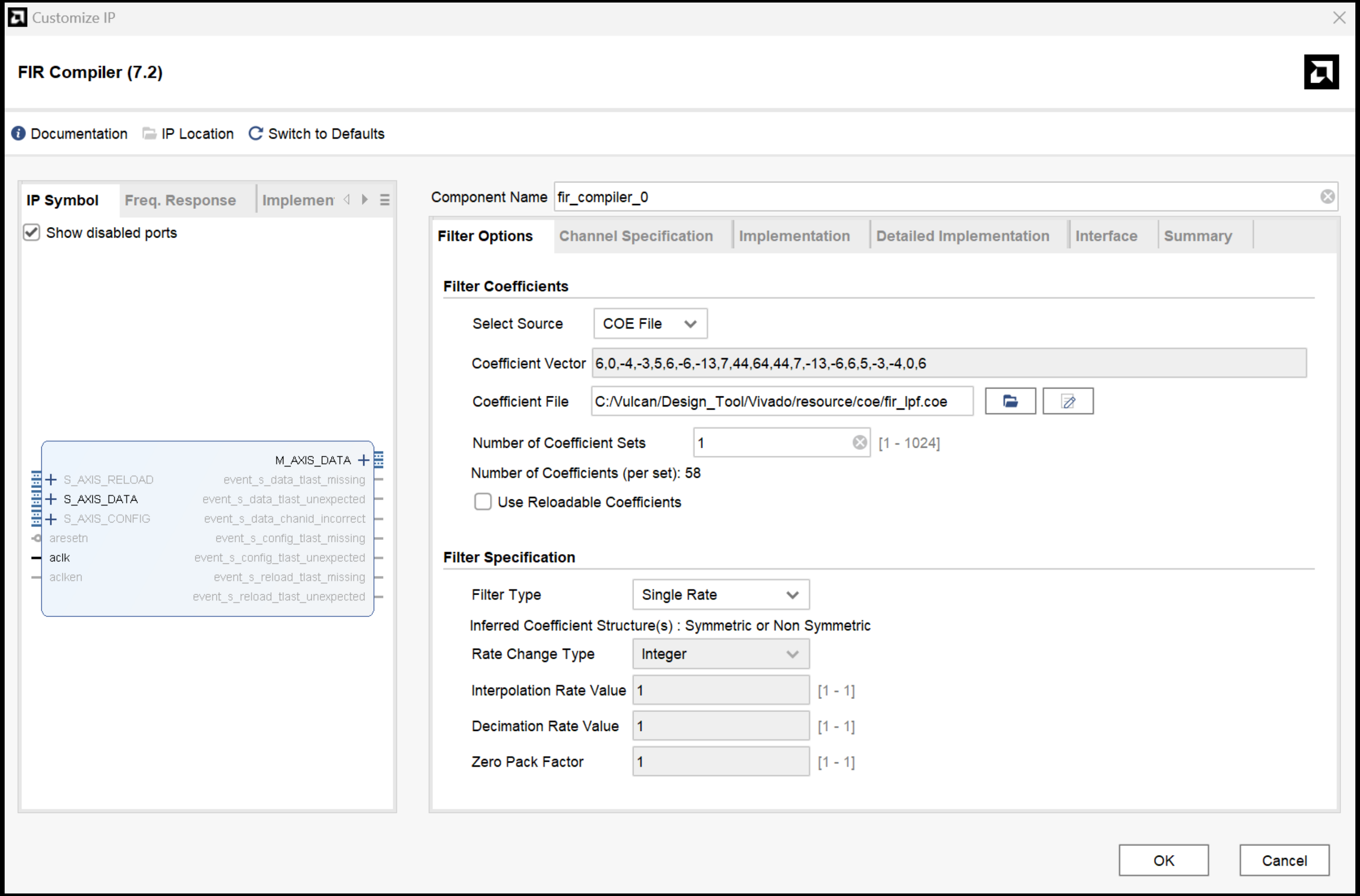
用Matlab製作FIR LPF (生成sin.coe)

```
width = 8;                                               %rom位寬
depth = 1024;                                            %rom深度
x = linspace(0,2*pi,depth);                              %一周期1024採樣點
y_sin = sin(x);                                          %生成餘弦數據
y_sin = round(y_sin*(2^(width-1)-1))+2^(width-1)-1;      %將餘弦數據化整

fid=fopen('~sin.coe','w');                               %創建.coe文件
fprintf(fid,'%d,\n',y_sin);                              %寫入文件
fclose(fid);                                             %關閉文件
```

Lab10 - FIR Low-pass filter

FIR IP Setting



Customize IP

FIR Compiler (7.2)

Documentation IP Location Switch to Defaults

IP Symbol
Freq. Response
Implemen
Show disabled ports
M_AXIS_DATA
S_AXIS_RELOAD
S_AXIS_DATA
S_AXIS_CONFIG
aresetn
aclk
aclken
event_s_data_tlast_missing
event_s_data_tlast_unexpected
event_s_data_chanid_incorrect
event_s_config_tlast_missing
event_s_config_tlast_unexpected
event_s_reload_tlast_missing
event_s_reload_tlast_unexpected

Component Name fir_compiler_0

Filter Options | Channel Specification | Implementation | Detailed Implementation | Interface | Summary

Filter Coefficients

Select Source: COE File

Coefficient Vector: 6,0,-4,-3,5,6,-6,-13,7,44,64,44,7,-13,-6,6,5,-3,-4,0,6

Coefficient File: C:/Vulcan/Design_Tool/Vivado/resource/coe/fir_lpf.coe

Number of Coefficient Sets: 1 [1 - 1024]

Number of Coefficients (per set): 58

☐ Use Reloadable Coefficients

Filter Specification

Filter Type: Single Rate

Inferred Coefficient Structure(s) : Symmetric or Non Symmetric

Rate Change Type: Integer

Interpolation Rate Value: 1 [1 - 1]

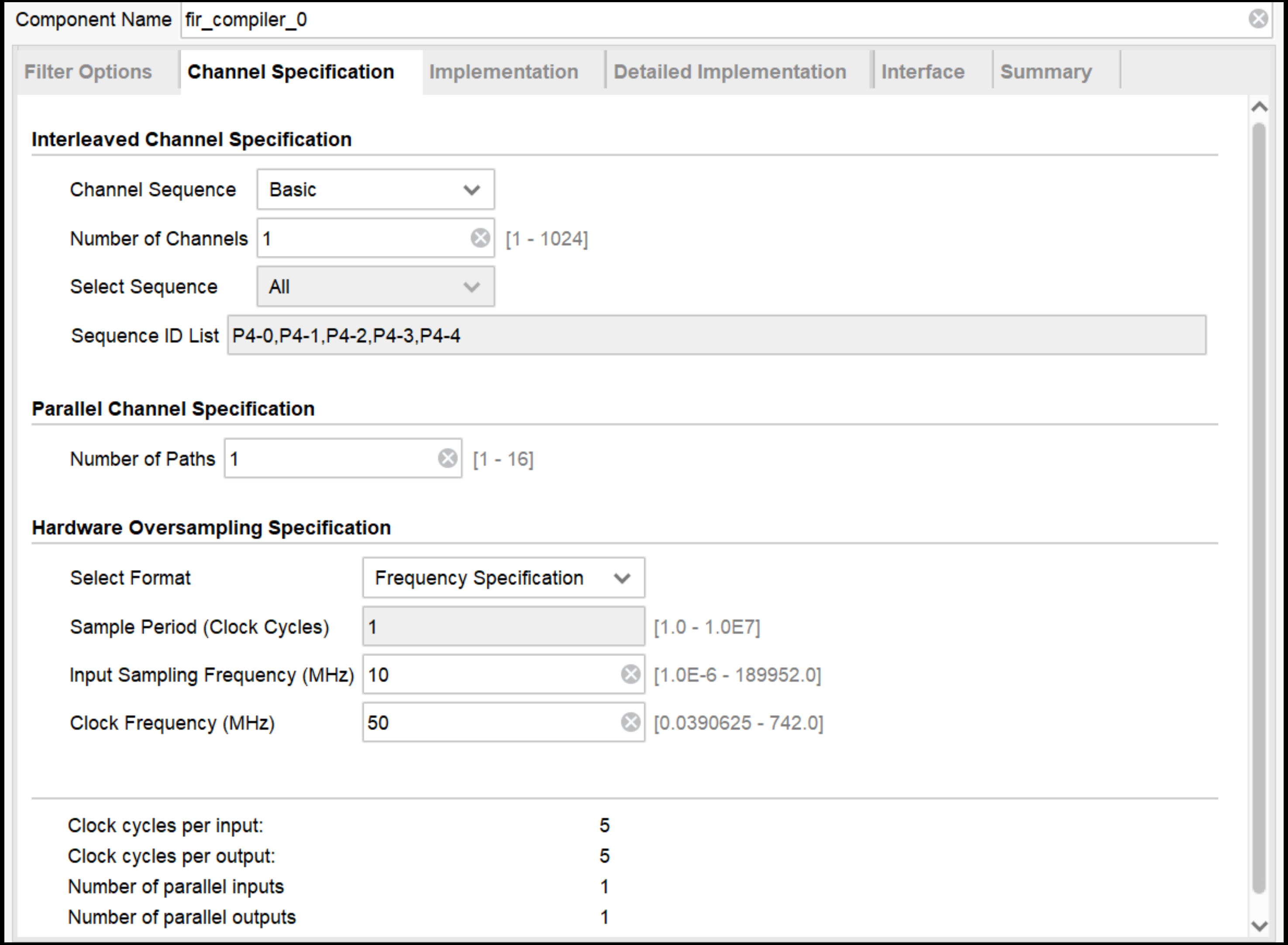
Decimation Rate Value: 1 [1 - 1]

Zero Pack Factor: 1 [1 - 1]

OK Cancel

Lab10 - FIR Low-pass filter

FIR IP Setting



Component Name: fir_compiler_0

Interleaved Channel Specification

Channel Sequence	Basic	
Number of Channels	1	[1 - 1024]
Select Sequence	All	
Sequence ID List	P4-0,P4-1,P4-2,P4-3,P4-4	

Parallel Channel Specification

Number of Paths	1	[1 - 16]

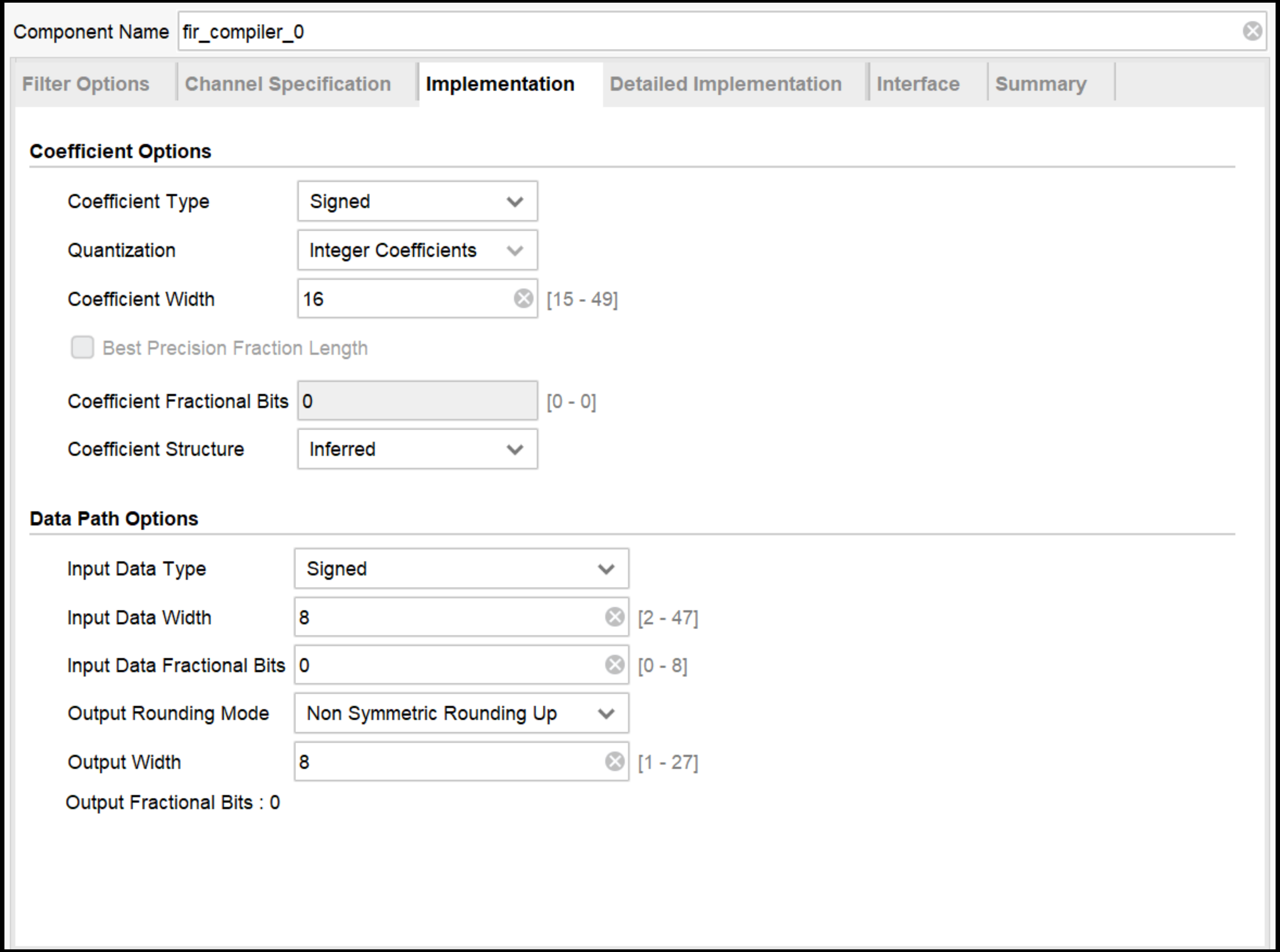
Hardware Oversampling Specification

Select Format	Frequency Specification	
Sample Period (Clock Cycles)	1	[1.0 - 1.0E7]
Input Sampling Frequency (MHz)	10	[1.0E-6 - 189952.0]
Clock Frequency (MHz)	50	[0.0390625 - 742.0]

Clock cycles per input:	5
Clock cycles per output:	5
Number of parallel inputs	1
Number of parallel outputs	1

Lab10 - FIR Low-pass filter

FIR IP Setting



Component Name: fir_compiler_0

Filter Options | Channel Specification | Implementation | Detailed Implementation | Interface | Summary

Coefficient Options

Coefficient Type	Signed	
Quantization	Integer Coefficients	
Coefficient Width	16	[15 - 49]
☐ Best Precision Fraction Length		
Coefficient Fractional Bits	0	[0 - 0]
Coefficient Structure	Inferred	

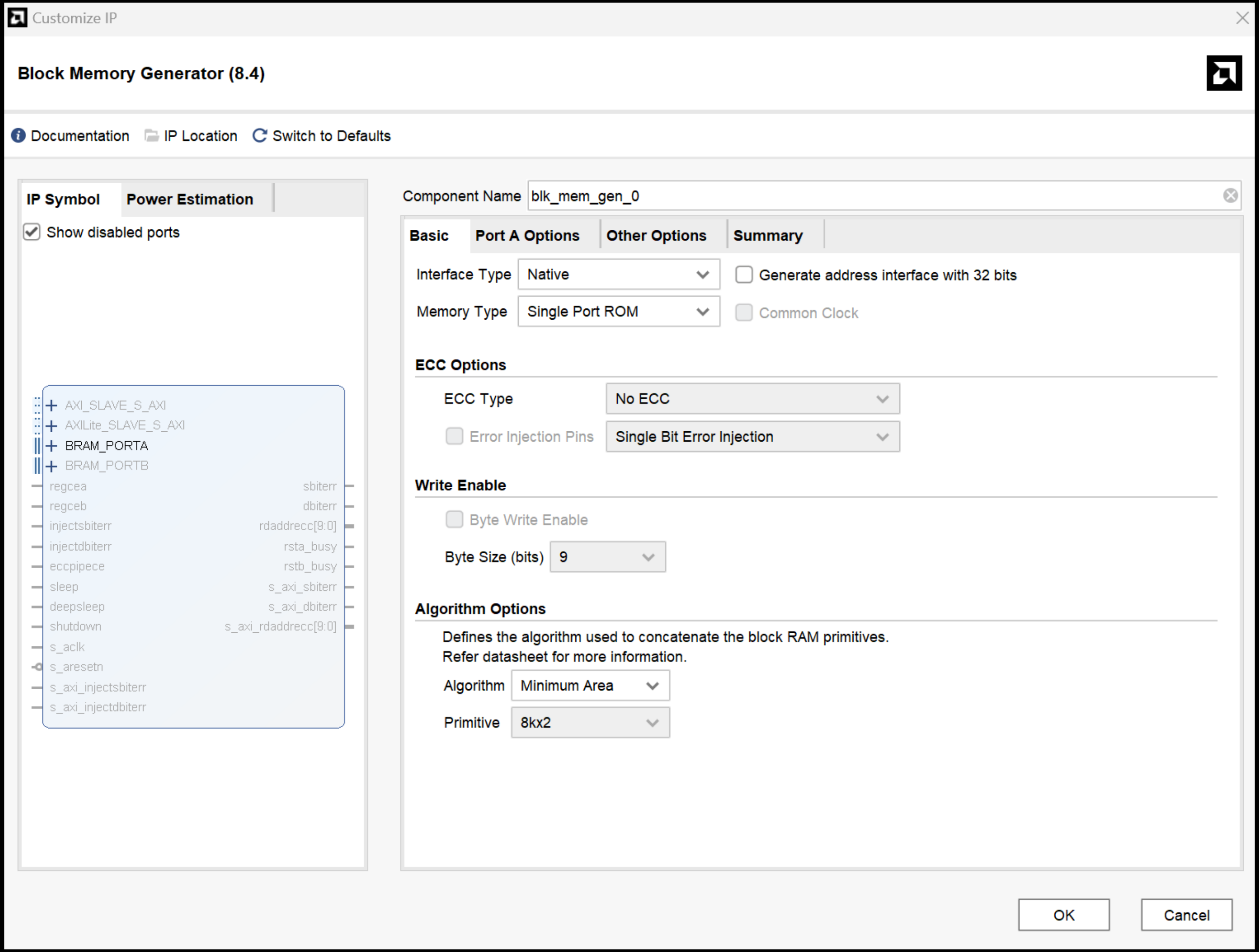
Data Path Options

Input Data Type	Signed	
Input Data Width	8	[2 - 47]
Input Data Fractional Bits	0	[0 - 8]
Output Rounding Mode	Non Symmetric Rounding Up	
Output Width	8	[1 - 27]

Output Fractional Bits : 0

Lab10 - FIR Low-pass filter

ROM IP Setting



Customize IP

Block Memory Generator (8.4)

Documentation IP Location Switch to Defaults

IP Symbol | Power Estimation

Show disabled ports [✓]

Component Name: blk_mem_gen_0

Basic | Port A Options | Other Options | Summary

Interface Type: Native
Generate address interface with 32 bits [ ]
Memory Type: Single Port ROM
Common Clock [ ]

ECC Options

ECC Type: No ECC
Error Injection Pins [ ]: Single Bit Error Injection

Write Enable

Byte Write Enable [ ]
Byte Size (bits): 9

Algorithm Options

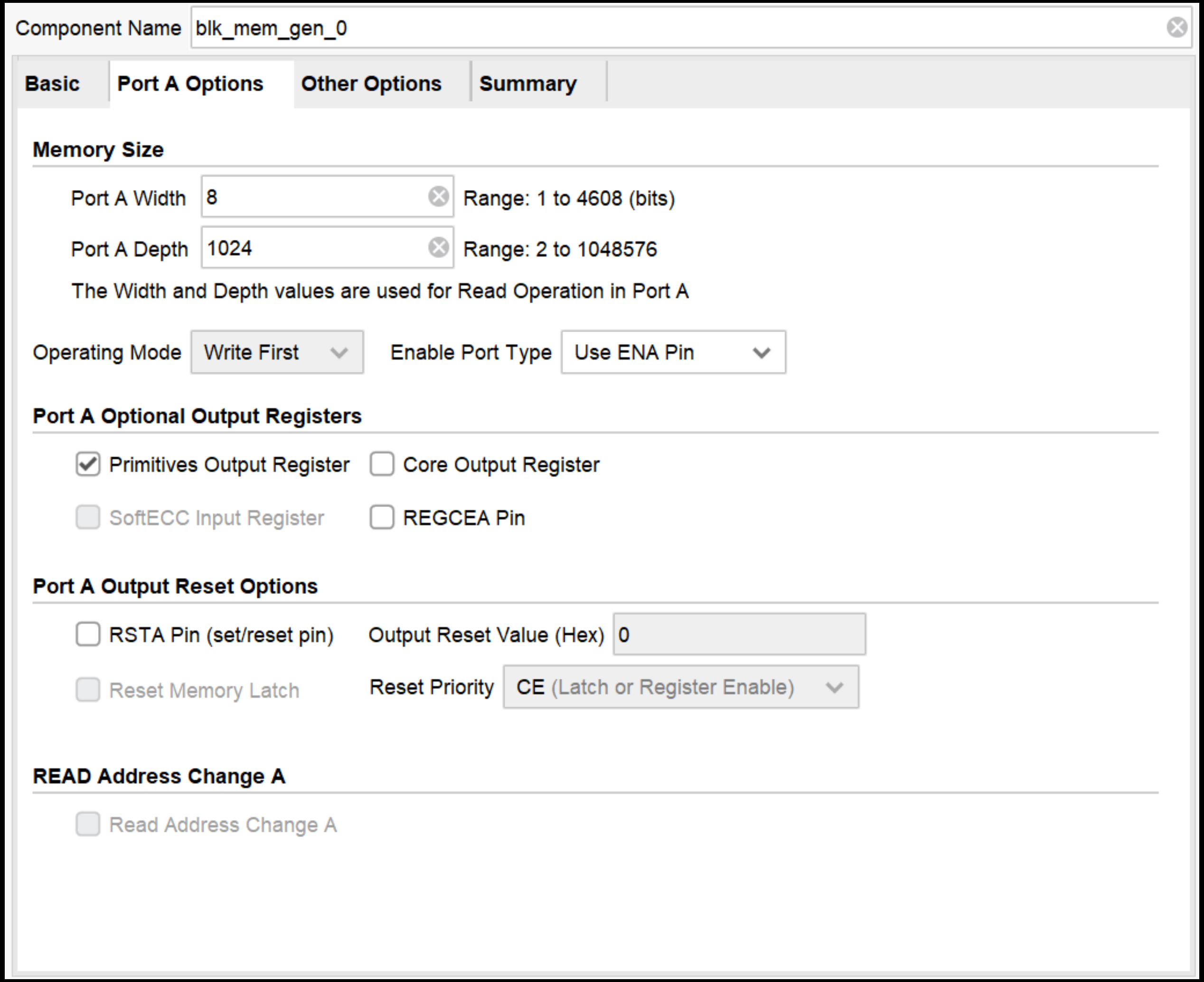
Defines the algorithm used to concatenate the block RAM primitives.
Refer datasheet for more information.

Algorithm: Minimum Area
Primitive: 8kx2

OK Cancel

Lab10 - FIR Low-pass filter

ROM IP Setting



Component Name: blk_mem_gen_0

Basic | Port A Options | Other Options | Summary

Memory Size

Port A Width: 8 — Range: 1 to 4608 (bits)

Port A Depth: 1024 — Range: 2 to 1048576

The Width and Depth values are used for Read Operation in Port A

Operating Mode: Write First — Enable Port Type: Use ENA Pin

Port A Optional Output Registers

- [✓] Primitives Output Register
- [ ] Core Output Register
- [ ] SoftECC Input Register
- [ ] REGCEA Pin

Port A Output Reset Options

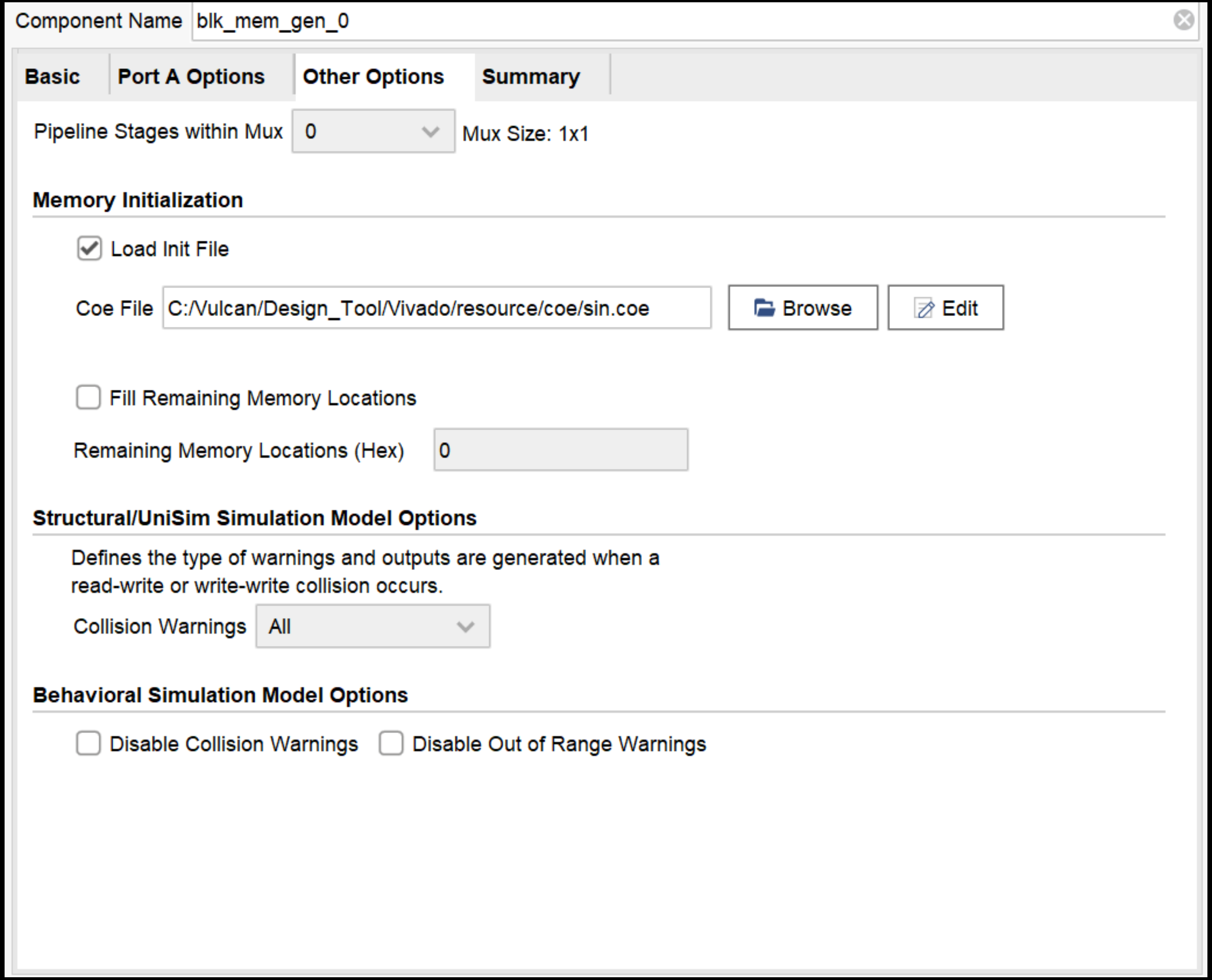
- [ ] RSTA Pin (set/reset pin) — Output Reset Value (Hex): 0
- [ ] Reset Memory Latch — Reset Priority: CE (Latch or Register Enable)

READ Address Change A

- [ ] Read Address Change A

Lab10 - FIR Low-pass filter

ROM IP Setting



Component Name: blk_mem_gen_0

Basic | Port A Options | Other Options | Summary

Pipeline Stages within Mux: 0 — Mux Size: 1x1

Memory Initialization

[✓] Load Init File

Coe File: C:/Vulcan/Design_Tool/Vivado/resource/coe/sin.coe (Browse) (Edit)

[ ] Fill Remaining Memory Locations

Remaining Memory Locations (Hex): 0

Structural/UniSim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

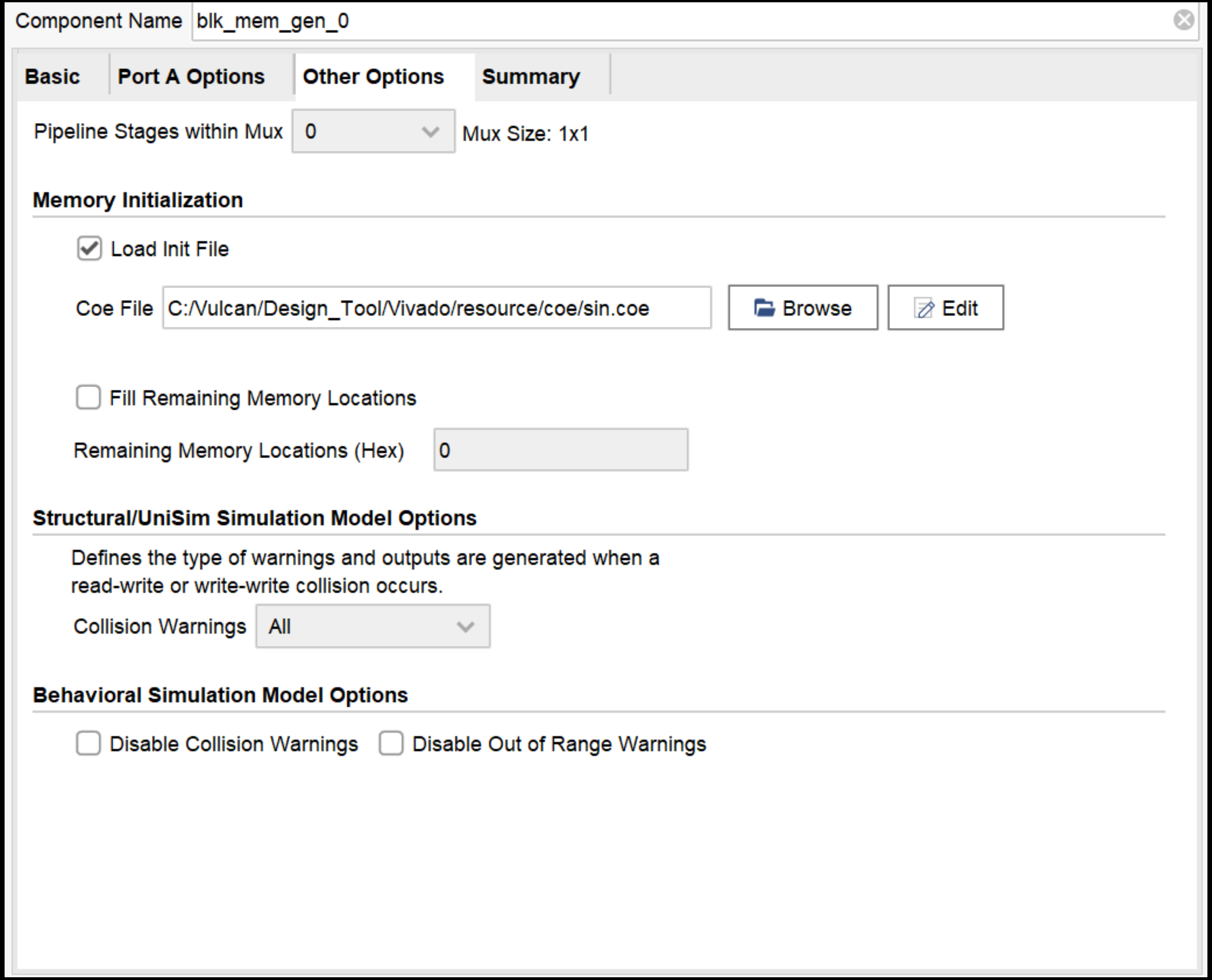
Collision Warnings: All

Behavioral Simulation Model Options

[ ] Disable Collision Warnings [ ] Disable Out of Range Warnings

Lab10 - FIR Low-pass filter

ROM IP Setting



Component Name blk_mem_gen_0

Pipeline Stages within Mux: 0 — Mux Size: 1x1

Memory Initialization

[✓] Load Init File

Coe File: C:/Vulcan/Design_Tool/Vivado/resource/coe/sin.coe

[ ] Fill Remaining Memory Locations

Remaining Memory Locations (Hex): 0

Structural/UniSim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

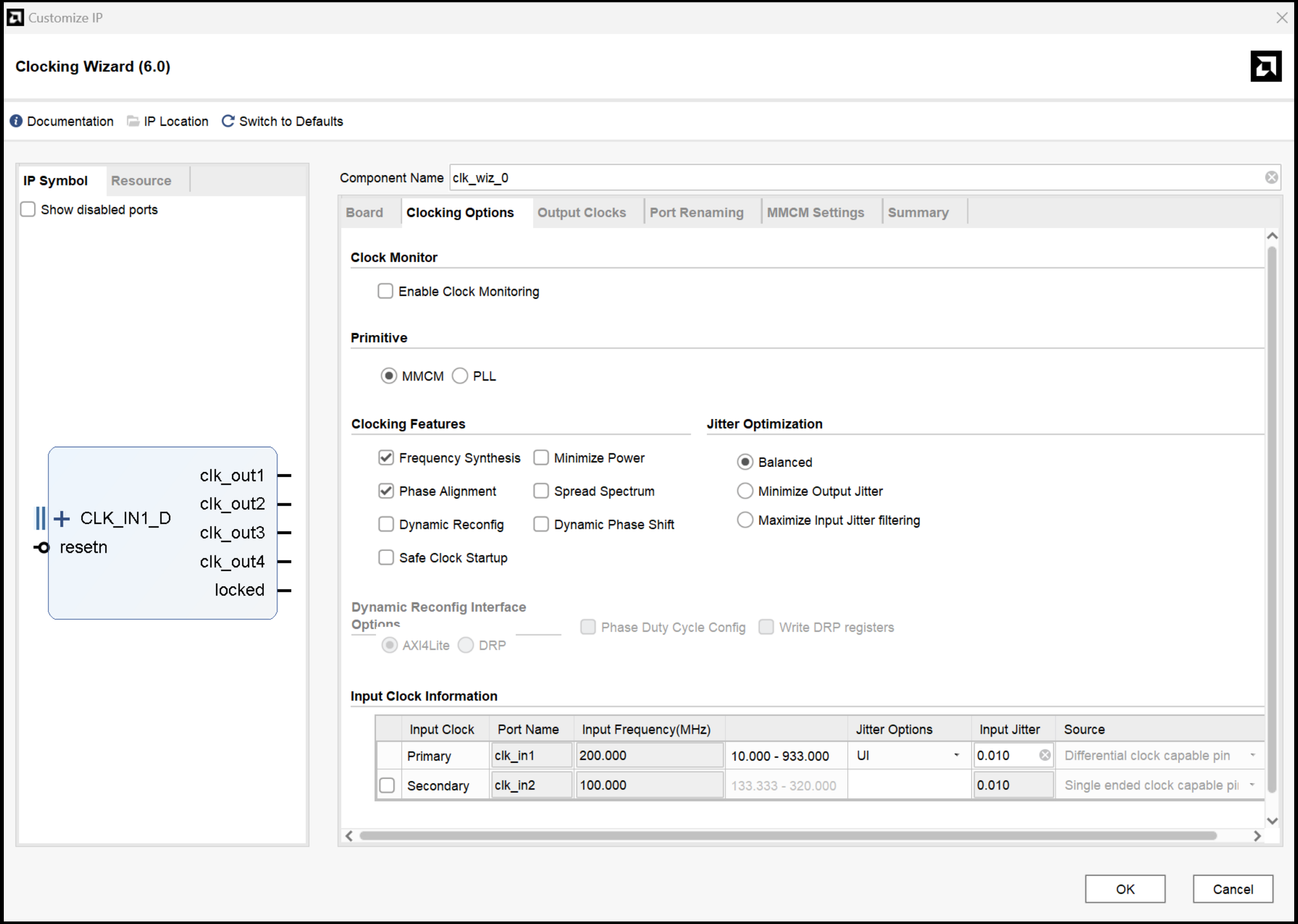
Collision Warnings: All

Behavioral Simulation Model Options

[ ] Disable Collision Warnings [ ] Disable Out of Range Warnings

Lab10 - FIR Low-pass filter

Clock IP Setting



Customize IP

Clocking Wizard (6.0)

Documentation | IP Location | Switch to Defaults

IP Symbol | Resource

☐ Show disabled ports

Component Name: clk_wiz_0

Board | **Clocking Options** | Output Clocks | Port Renaming | MMCM Settings | Summary

Clock Monitor

☐ Enable Clock Monitoring

Primitive

● MMCM ○ PLL

Clocking Features

☑ Frequency Synthesis ☐ Minimize Power
☑ Phase Alignment ☐ Spread Spectrum
☐ Dynamic Reconfig ☐ Dynamic Phase Shift
☐ Safe Clock Startup

Jitter Optimization

● Balanced
○ Minimize Output Jitter
○ Maximize Input Jitter filtering

Dynamic Reconfig Interface Options

● AXI4Lite ○ DRP ☐ Phase Duty Cycle Config ☐ Write DRP registers

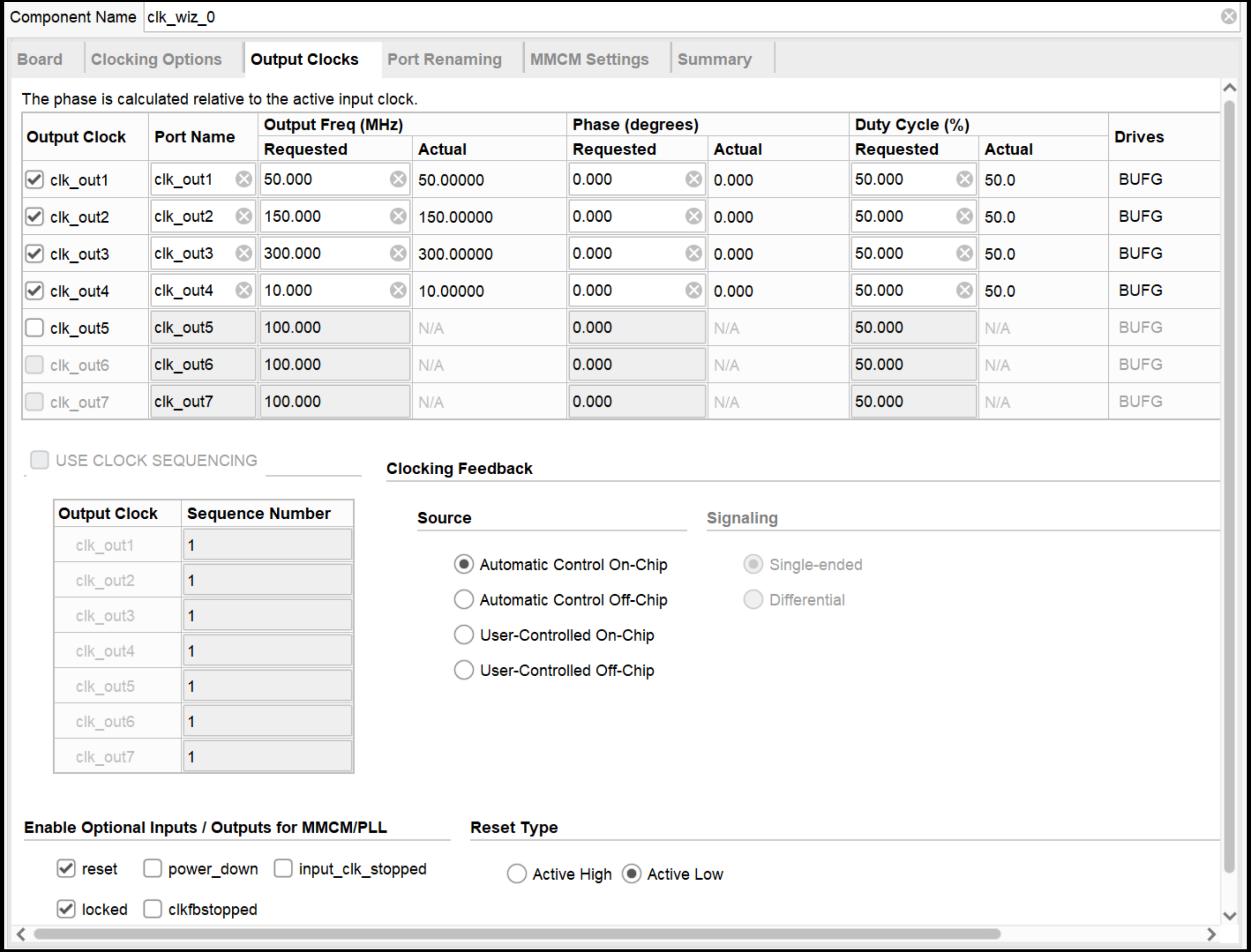
Input Clock Information

	Input Clock	Port Name	Input Frequency(MHz)		Jitter Options	Input Jitter	Source
	Primary	clk_in1	200.000	10.000 - 933.000	UI	0.010	Differential clock capable pin
☐	Secondary	clk_in2	100.000	133.333 - 320.000		0.010	Single ended clock capable pi

OK | Cancel

Lab10 - FIR Low-pass filter

Clock IP Setting



Component Name: clk_wiz_0

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)		Drives
		Requested	Actual	Requested	Actual	Requested	Actual	
✓ clk_out1	clk_out1	50.000	50.00000	0.000	0.000	50.000	50.0	BUFG
✓ clk_out2	clk_out2	150.000	150.00000	0.000	0.000	50.000	50.0	BUFG
✓ clk_out3	clk_out3	300.000	300.00000	0.000	0.000	50.000	50.0	BUFG
✓ clk_out4	clk_out4	10.000	10.00000	0.000	0.000	50.000	50.0	BUFG
☐ clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
☐ clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG
☐ clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	BUFG

☐ USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

Source: [✓ Automatic Control On-Chip] Automatic Control Off-Chip / User-Controlled On-Chip / User-Controlled Off-Chip

Signaling: [✓ Single-ended] Differential

Enable Optional Inputs / Outputs for MMCM/PLL

✓ reset ☐ power_down ☐ input_clk_stopped

✓ locked ☐ clkfbstopped

Reset Type

Active High [✓ Active Low]

Lab10 - FIR Low-pass filter

Port define

```
`timescale 1ns / 1ps
module fir_top(
    input clk,
    input rst_n,
    input [1:0] rom_sel,
    output [7:0] douta,
    output [7:0] fir_out_data
    );

    wire        clk_1;
    wire        clk_2;
    wire        clk_3;
    wire        clk_10m;
    wire        clk_rom;

    reg  [9:0]  addra;
    wire [7:0]  fir_in_data;
```

Lab10 - FIR Low-pass filter

clock

```verilog
assign clk_rom = (rom_sel == 0)?clk_1:((rom_sel == 1)?clk_2 : clk_3);

clk_wiz_0 clk_inst
   (
    // Clock out ports
    .clk_out1(clk_1),     // output clk_out1
    .clk_out2(clk_2),     // output clk_out2
    .clk_out3(clk_3),     // output clk_out3
    .clk_out4(clk_10m),     // output clk_out4
    // Status and control signals
    .resetn(rst_n), // input resetn
    .locked(),       // output locked
   // Clock in ports
    .clk_in1(clk)
);
```

Lab10 - FIR Low-pass filter

Rom

```
always@(posedge clk_rom or negedge rst_n)begin
    if(~rst_n)
        addra <= 'd0;
    else
        addra <= addra + 1'b1;
end

blk_mem_gen_0 rom_inst (
  .clka(clk_rom),    // input wire clka
  .ena(1'b1),        // input wire ena
  .addra(addra),   // input wire [9 : 0] addra
  .douta(douta)   // output wire [7 : 0] douta
);
```

Lab10 - FIR Low-pass filter

FIR_Dout

```
assign fir_in_data = douta - 1'd128;

fir_compiler_0 fir_inst (
  .aclk(clk),                              // input wire aclk
  .s_axis_data_tvalid(clk_10m),  // input wire s_axis_data_tvalid
  .s_axis_data_tready(),  // output wire s_axis_data_tready
  .s_axis_data_tdata(fir_in_data),    // input wire [7 : 0] s_axis_data_tdata
  .m_axis_data_tvalid(),  // output wire m_axis_data_tvalid
  .m_axis_data_tdata(fir_out_data)    // output wire [7 : 0] m_axis_data_tdata
);
```

Lab10 - FIR Low-pass filter

Tb_fir

```
`timescale 1ns / 1ps
module fit_top_tb();

reg             clk;
reg             rst_n;
reg   [1:0]     rom_sel;
wire  [7:0]     douta;
wire  [7:0]     fir_out_data;

initial clk = 0;
always#10 clk = ~clk;

fir_top fir_top_inst(
    .clk(clk),
    .rst_n(rst_n),
    .rom_sel(rom_sel),
    .douta(douta),
    .fir_out_data(fir_out_data)
);
```

Lab10 - FIR Low-pass filter

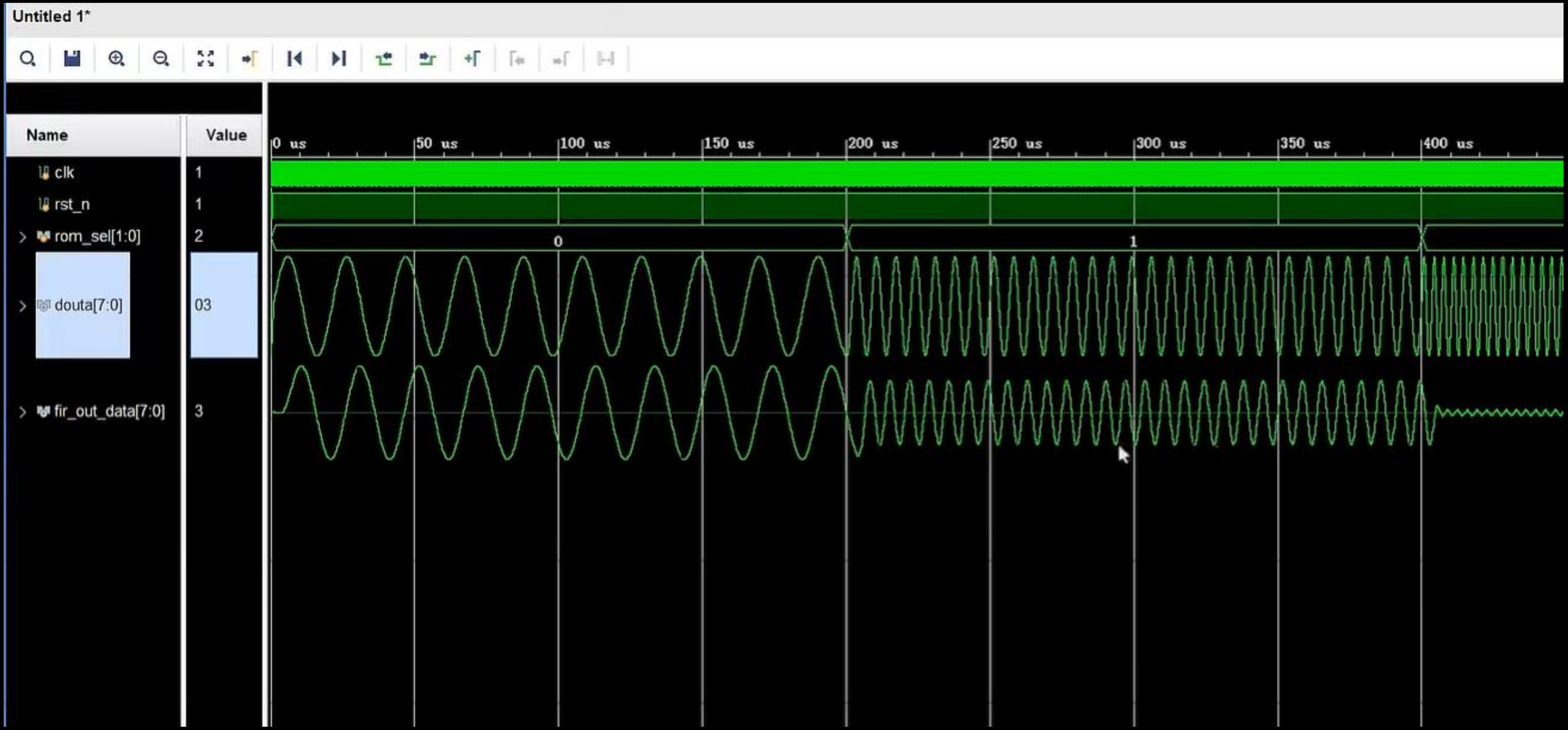
Tb_fir

```
initial begin
    rst_n = 0;
    rom_sel = 0;
    #200;
    rst_n = 1'b1;
    #200000;
    rom_sel = 1;
    #200000;
    rom_sel = 2;
    #200000;
    $stop;
end

endmodule
```

Lab10 - FIR Low-pass filter

Simulation


Untitled 1*
Name
Value
0 us
50 us
100 us
150 us
200 us
250 us
300 us
350 us
400 us
clk
1
rst_n
1
rom_sel[1:0]
2
0
1
douta[7:0]
03
fir_out_data[7:0]
3