

Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

PWM + eCAP module
In FPGA zybo base
device_driver

강사 : Innova Lee(이 상훈)

학생 : 김 시윤

1. make custom ip My_PWM_Core & My_ECAP_Core

1. Vivado 에서 New Project 를 만든다.

New Project

Project Name
Enter a name for your project and specify a directory where the project data files will be stored.

Project name:

Project location:

☐ Create project subdirectory

Project will be created at: /home/siyun/pwm_ecap_present/hardware

New Project

Project Type
Specify the type of project to create.

☒ **RTL Project**
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
☒ Do not specify sources at this time

☐ **Post-synthesis Project:** You will be able to add sources, view device resources, run design analysis, planning and implementation.
☐ Do not specify sources at this time

☐ **I/O Planning Project**
Do not specify design sources. You will be able to view part/package resources.

☐ **Imported Project**
Create a Vivado project from a Synplify, XST or ISE Project File.

☐ **Example Project**
Create a new Vivado project from a predefined template.

New Project

Default Part

Choose a default Xilinx part or board for your project. This can be changed later.

Parts

Boards

Filter/ Preview

Vendor:

All

Display Name:

All

Board Rev:

Latest

Reset All Filters

Search:

Q

| Display Name | Vendor | Board Rev | Part | I/O Pir |
|---|-----------------|-----------|--------------------|---------|
| Zybo Z7-20 | digilentinc.com | B.2 | xc7z020clg400-1 | 400 |
| Zybo | digilentinc.com | B.3 | xc7z010clg400-1 | 400 |
| ZedBoard Zynq Evaluation and Development Kit | em.avnet.com | d | xc7z020clg484-1 | 484 |
| Artix-7 AC701 Evaluation Platform | xilinx.com | 1.1 | xc7a200tfg676-2 | 676 |
| Kintex UltraScale+ KCU116 Evaluation Platform | xilinx.com | 1.0 | xcku5p-ffvb676-2-e | 676 |
| ZYNQ-7 ZC702 Evaluation Board | xilinx.com | 1.0 | xc7z020clg484-1 | 484 |

No Board Connectors

?

< Back

Next >

Finish

Cancel

New Project

VIVADO
HLx Editions

New Project Summary

- A new RTL project named 'project_1' will be created.
- The default part and product family for the new project:
 - Default Board: Zybo
 - Default Part: xc7z010clg400-1
 - Product: Zynq-7000
 - Family: Zynq-7000
 - Package: clg400
 - Speed Grade: -1

XILINX
ALL PROGRAMMABLE

To create the project, click Finish

?

< Back

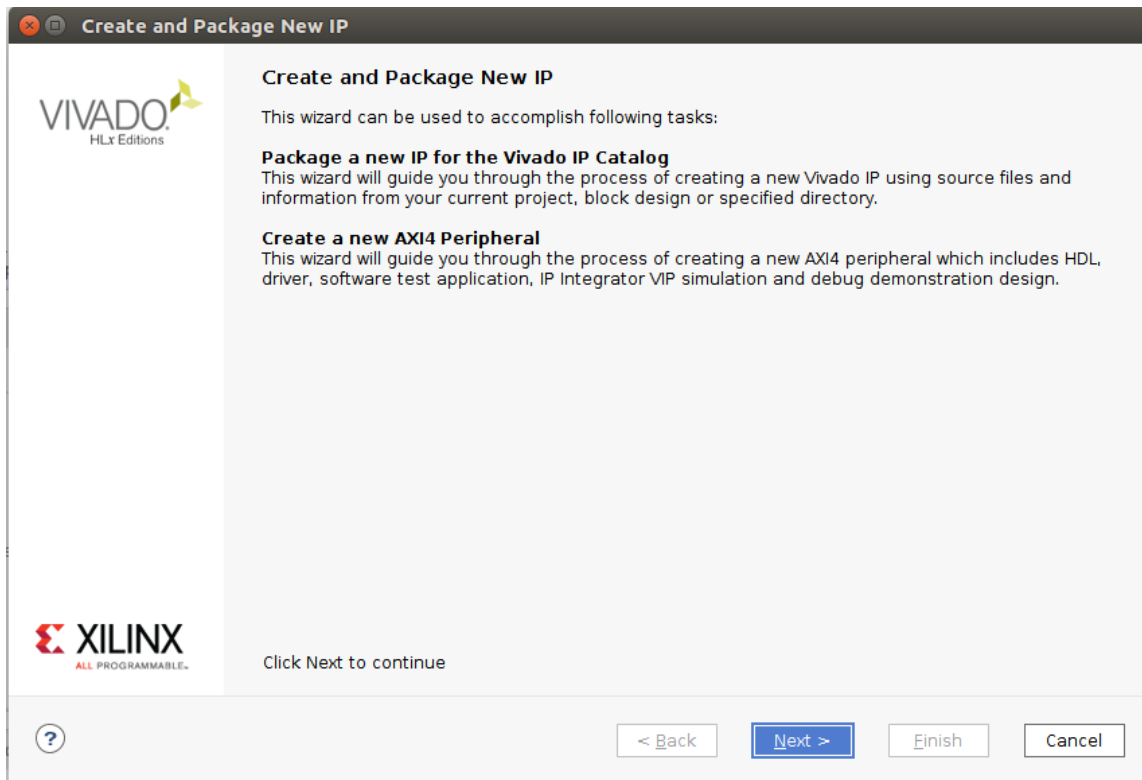
Next >

Finish

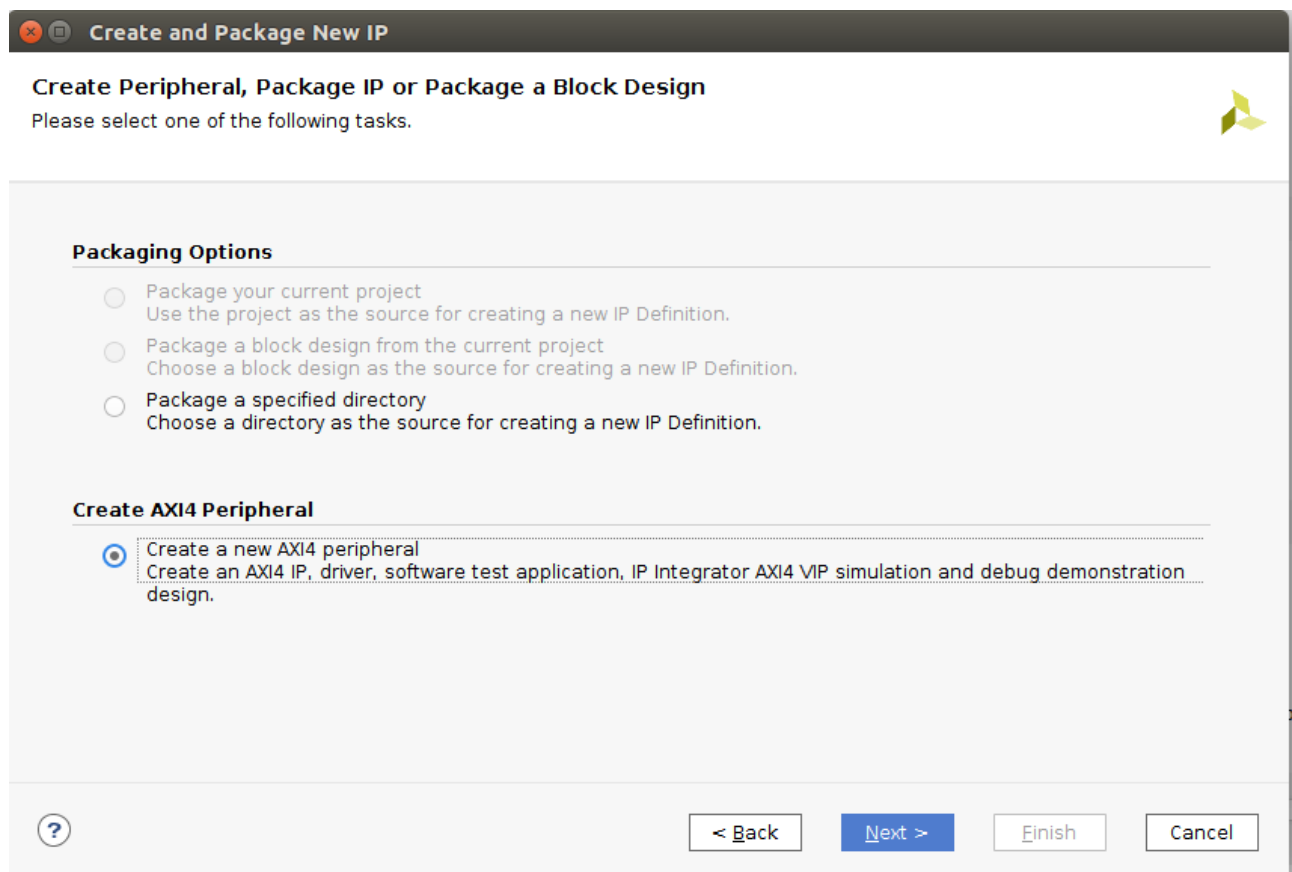
Cancel

2. Vivado 상단에 Tools → Create and Package New IP 를 클릭

Next



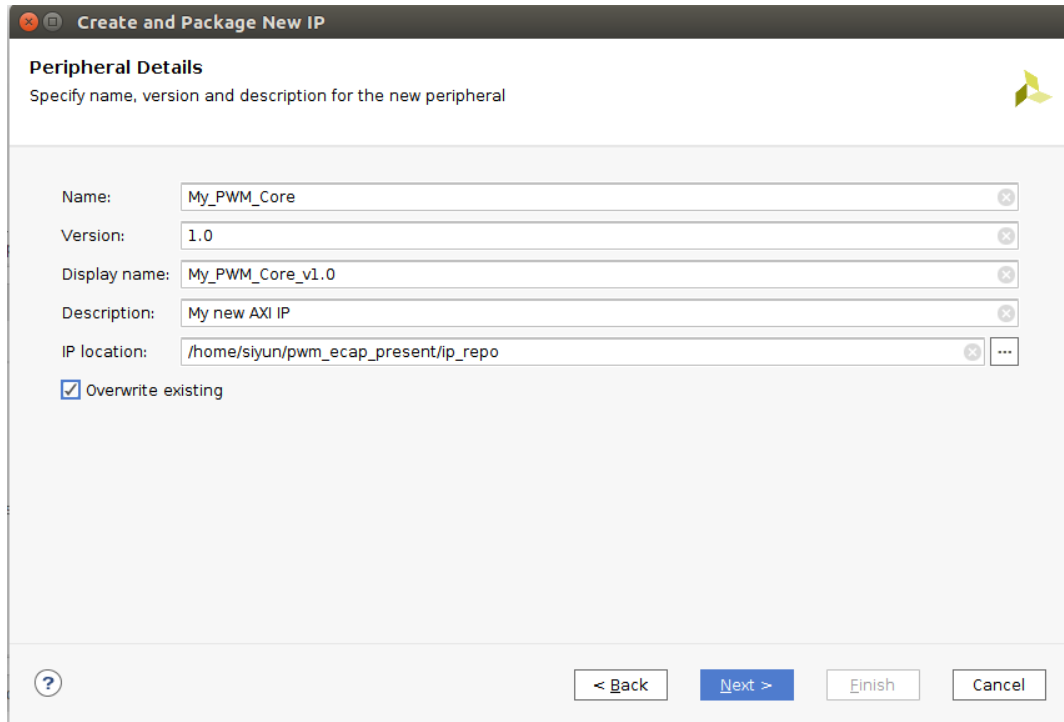
Create a new AXI4 peripheral 선택 후 Next



Custom IP 이름 설정 베릴로그 작성 시 문제가 될 수 있으므로 되도록이면 이름은 동일하게 설정하도록 한다.

상단에 Name 을 My_PWM_Core 로 하면 밑에는 자동으로 설정된다.

이름입력 후 Next



Create and Package New IP

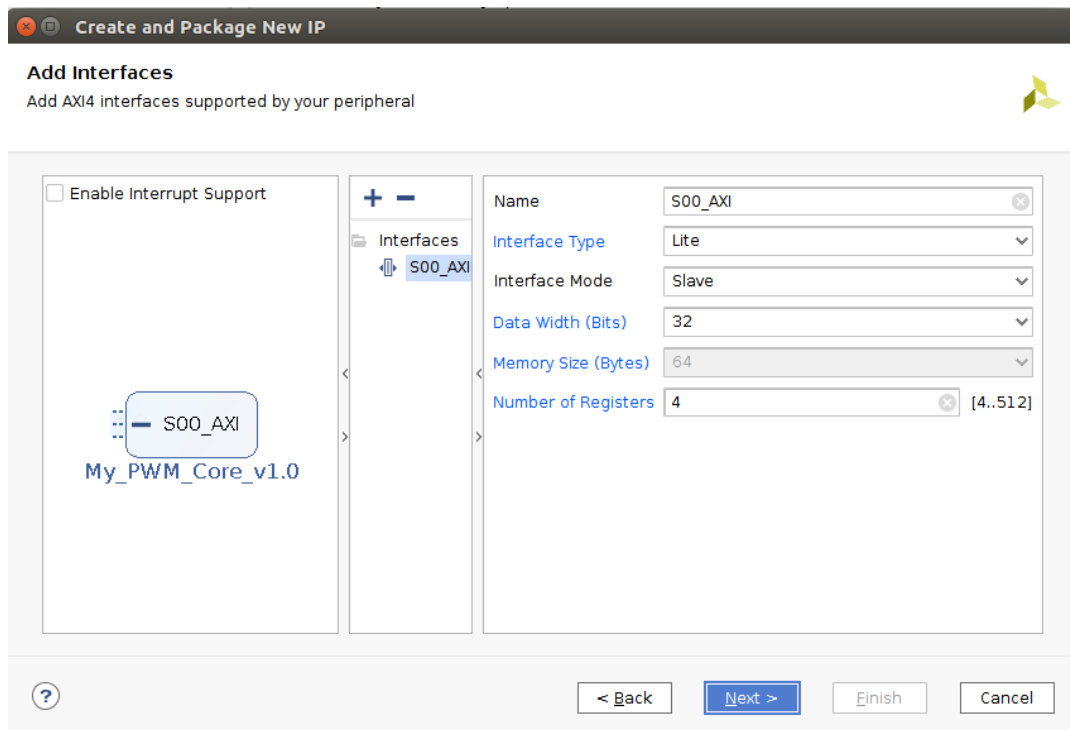
Peripheral Details
Specify name, version and description for the new peripheral

Name: My_PWM_Core
Version: 1.0
Display name: My_PWM_Core_v1.0
Description: My new AXI IP
IP location: /home/siyun/pwm_ecap_present/ip_repo

☒ Overwrite existing

< Back Next > Finish Cancel

아무설정도 해주지 않고 그냥 Next



Create and Package New IP

Add Interfaces
Add AXI4 interfaces supported by your peripheral

☐ Enable Interrupt Support

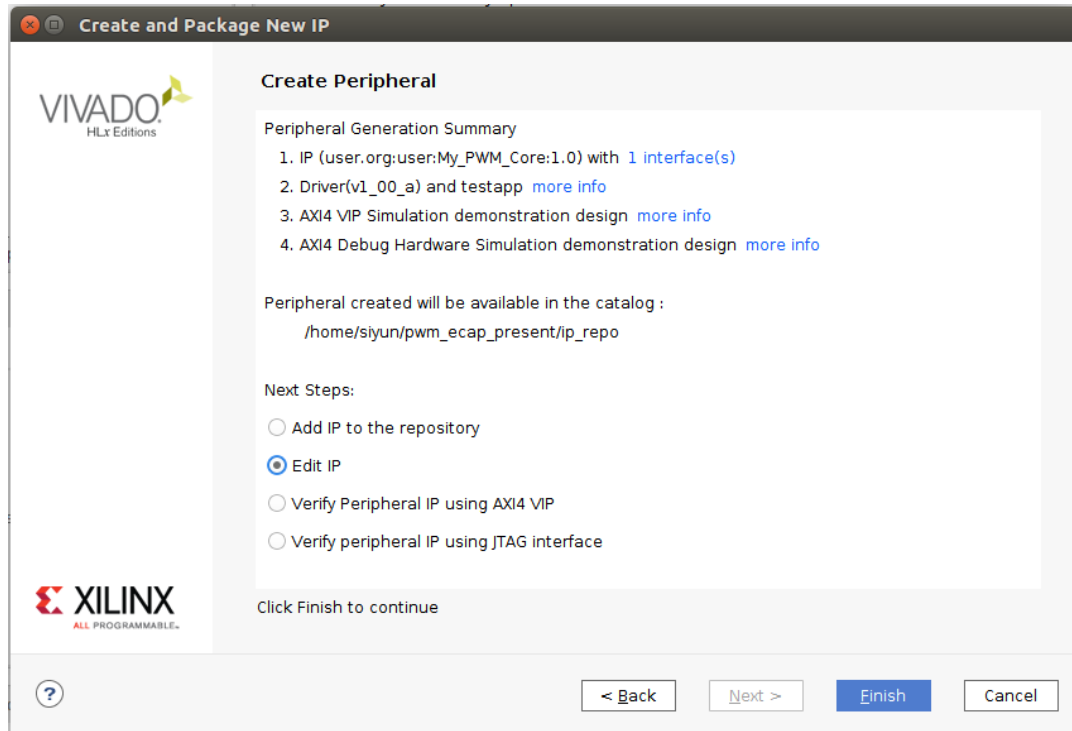
Interfaces
+ S00_AXI

My_PWM_Core_v1.0

Name: S00_AXI
Interface Type: Lite
Interface Mode: Slave
Data Width (Bits): 32
Memory Size (Bytes): 64
Number of Registers: 4 [4..512]

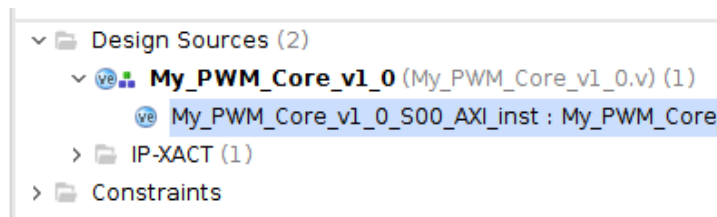
< Back Next > Finish Cancel

Edit IP 선택 후 Finish 클릭



위 작업을 완료 했으면 왼쪽 Source 칸에 이와 같은 Verilog 파일들이 생성된다.

My_PWM_Core_v1_0 은 탑모듈로 하위모듈과 연결을 담당하고 , My_PWM_Core_v1_0_S00_AXI_inst 는 하위모듈로 로직이 들어있다.



우선 하위모듈인 My_PWM_Core_v1_0_S00_AXI_inst 를 먼저 수정하도록 한다.

My_PWM_Core_v1_0_S00_AXI_inst

```
`timescale 1 ns / 1 ps

module My_PWM_Core_v1_0_S00_AXI #
(
    // Users to add parameters here
    parameter integer PWM_COUNTER_MAX = 2000000,
    // User parameters ends
    // Do not modify the parameters beyond this line

    // Width of S_AXI data bus
    parameter integer C_S_AXI_DATA_WIDTH    = 32,
    // Width of S_AXI address bus
    parameter integer C_S_AXI_ADDR_WIDTH    = 4
)
(
    // Users to add ports here
    output wire PWM0,
    // User ports ends
    // Do not modify the ports beyond this line
```

```

// Global Clock Signal
input wire S_AXI_ACLK,
// Global Reset Signal. This Signal is Active LOW
input wire S_AXI_ARESETN,
// Write address (issued by master, accepted by Slave)
input wire [C_S_AXI_ADDR_WIDTH-1 : 0] S_AXI_AWADDR,
// Write channel Protection type. This signal indicates the
// privilege and security level of the transaction, and whether
// the transaction is a data access or an instruction access.
input wire [2 : 0] S_AXI_AWPROT,
// Write address valid. This signal indicates that the master signaling
// valid write address and control information.
input wire S_AXI_AWVALID,
// Write address ready. This signal indicates that the slave is ready
// to accept an address and associated control signals.
output wire S_AXI_AWREADY,
// Write data (issued by master, accepted by Slave)
input wire [C_S_AXI_DATA_WIDTH-1 : 0] S_AXI_WDATA,
// Write strobes. This signal indicates which byte lanes hold
// valid data. There is one write strobe bit for each eight
// bits of the write data bus.
input wire [(C_S_AXI_DATA_WIDTH/8)-1 : 0] S_AXI_WSTRB,
// Write valid. This signal indicates that valid write
// data and strobes are available.
input wire S_AXI_WVALID,
// Write ready. This signal indicates that the slave
// can accept the write data.
output wire S_AXI_WREADY,
// Write response. This signal indicates the status
// of the write transaction.
output wire [1 : 0] S_AXI_BRESP,
// Write response valid. This signal indicates that the channel
// is signaling a valid write response.
output wire S_AXI_BVALID,
// Response ready. This signal indicates that the master
// can accept a write response.
input wire S_AXI_BREADY,
// Read address (issued by master, accepted by Slave)
input wire [C_S_AXI_ADDR_WIDTH-1 : 0] S_AXI_ARADDR,
// Protection type. This signal indicates the privilege
// and security level of the transaction, and whether the
// transaction is a data access or an instruction access.
input wire [2 : 0] S_AXI_ARPROT,
// Read address valid. This signal indicates that the channel
// is signaling valid read address and control information.
input wire S_AXI_ARVALID,
// Read address ready. This signal indicates that the slave is
// ready to accept an address and associated control signals.
output wire S_AXI_ARREADY,
// Read data (issued by slave)
output wire [C_S_AXI_DATA_WIDTH-1 : 0] S_AXI_RDATA,
// Read response. This signal indicates the status of the
// read transfer.
output wire [1 : 0] S_AXI_RRESP,
// Read valid. This signal indicates that the channel is
// signaling the required read data.
output wire S_AXI_RVALID,
// Read ready. This signal indicates that the master can
// accept the read data and response information.
input wire S_AXI_RREADY

```

```
);
```

```

// AXI4LITE signals
reg [C_S_AXI_ADDR_WIDTH-1 : 0]    axi_awaddr;
reg    axi_awready;
reg    axi_wready;
reg [1 : 0]    axi_bresp;
reg    axi_bvalid;
reg [C_S_AXI_ADDR_WIDTH-1 : 0]    axi_araddr;
reg    axi_arready;
reg [C_S_AXI_DATA_WIDTH-1 : 0]    axi_rdata;
reg [1 : 0]    axi_rresp;
reg    axi_rvalid;

// Example-specific design signals
// local parameter for addressing 32 bit / 64 bit C_S_AXI_DATA_WIDTH
// ADDR_LSB is used for addressing 32/64 bit registers/memories
// ADDR_LSB = 2 for 32 bits (n downto 2)
// ADDR_LSB = 3 for 64 bits (n downto 3)
localparam integer ADDR_LSB = (C_S_AXI_DATA_WIDTH/32) + 1;
localparam integer OPT_MEM_ADDR_BITS = 1;
//-----
//-- Signals for user logic register space example
//-----
//-- Number of Slave Registers 4
reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg0;
reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg1;
reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg2;
reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg3;
wire    slv_reg_rden;
wire    slv_reg_wren;
reg [C_S_AXI_DATA_WIDTH-1:0]    reg_data_out;
integer    byte_index;
reg    aw_en;

// I/O Connections assignments

assign S_AXI_AWREADY    = axi_awready;
assign S_AXI_WREADY    = axi_wready;
assign S_AXI_BRESP    = axi_bresp;
assign S_AXI_BVALID    = axi_bvalid;
assign S_AXI_ARREADY    = axi_arready;
assign S_AXI_RDATA    = axi_rdata;
assign S_AXI_RRESP    = axi_rresp;
assign S_AXI_RVALID    = axi_rvalid;
// Implement axi_awready generation
// axi_awready is asserted for one S_AXI_ACLK clock cycle when both
// S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_awready is
// de-asserted when reset is low.

always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        axi_awready <= 1'b0;
        aw_en <= 1'b1;
    end
    else
    begin
        if (~axi_awready && S_AXI_AWVALID && S_AXI_WVALID && aw_en)
        begin
            // slave is ready to accept write address when
            // there is a valid write address and write data
            // on the write address and data bus. This design

```



```

        // expects no outstanding transactions.
        axi_awready <= 1'b1;
        aw_en <= 1'b0;
    end
    else if (S_AXI_BREADY && axi_bvalid)
        begin
            aw_en <= 1'b1;
            axi_awready <= 1'b0;
        end
    else
        begin
            axi_awready <= 1'b0;
        end
    end
end

// Implement axi_awaddr latching
// This process is used to latch the address when both
// S_AXI_AWVALID and S_AXI_WVALID are valid.

always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
        begin
            axi_awaddr <= 0;
        end
    else
        begin
            if (~axi_awready && S_AXI_AWVALID && S_AXI_WVALID && aw_en)
                begin
                    // Write Address latching
                    axi_awaddr <= S_AXI_AWADDR;
                end
            end
        end
    end
end

// Implement axi_wready generation
// axi_wready is asserted for one S_AXI_ACLK clock cycle when both
// S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_wready is
// de-asserted when reset is low.

always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
        begin
            axi_wready <= 1'b0;
        end
    else
        begin
            if (~axi_wready && S_AXI_WVALID && S_AXI_AWVALID && aw_en )
                begin
                    // slave is ready to accept write data when
                    // there is a valid write address and write data
                    // on the write address and data bus. This design
                    // expects no outstanding transactions.
                    axi_wready <= 1'b1;
                end
            end
        end
    end
end

```

```

end

// Implement memory mapped register select and write logic generation
// The write data is accepted and written to memory mapped registers when
// axi_awready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted. Write strobes are used to
// select byte enables of slave registers while writing.
// These registers are cleared when reset (active low) is applied.
// Slave register write enable is asserted when valid address and data are available
// and the slave is ready to accept the write address and write data.
assign slv_reg_wren = axi_wready && S_AXI_WVALID && axi_awready && S_AXI_AWVALID;

always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        slv_reg0 <= 0;
        slv_reg1 <= 0;
        slv_reg2 <= 0;
        slv_reg3 <= 0;
    end
    else begin
        if (slv_reg_wren)
        begin
            case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
                2'h0:
                    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
                        if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                            // Respective byte enables are asserted as per write strobes
                            // Slave register 0
                            slv_reg0[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                        end
                2'h1:
                    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
                        if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                            // Respective byte enables are asserted as per write strobes
                            // Slave register 1
                            slv_reg1[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                        end
                2'h2:
                    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
                        if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                            // Respective byte enables are asserted as per write strobes
                            // Slave register 2
                            slv_reg2[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                        end
                2'h3:
                    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
                        if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                            // Respective byte enables are asserted as per write strobes
                            // Slave register 3
                            slv_reg3[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                        end
                default : begin
                    slv_reg0 <= slv_reg0;
                    slv_reg1 <= slv_reg1;
                    slv_reg2 <= slv_reg2;
                    slv_reg3 <= slv_reg3;
                end
            endcase
        end
    end
end
end

```

```

// Implement write response logic generation
// The write response and response valid signals are asserted by the slave
// when axi_wready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted.
// This marks the acceptance of address and indicates the status of
// write transaction.

always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        axi_bvalid <= 0;
        axi_bresp <= 2'b0;
    end
    else
    begin
        if (axi_awready && S_AXI_AWVALID && ~axi_bvalid && axi_wready && S_AXI_WVALID)
        begin
            // indicates a valid write response is available
            axi_bvalid <= 1'b1;
            axi_bresp <= 2'b0; // 'OKAY' response
        end
        // work error responses in future
    end
    else
    begin
        if (S_AXI_BREADY && axi_bvalid)
        //check if bready is asserted while bvalid is high)
        //(there is a possibility that bready is always asserted high)
        begin
            axi_bvalid <= 1'b0;
        end
    end
    end
end

// Implement axi_arready generation
// axi_arready is asserted for one S_AXI_ACLK clock cycle when
// S_AXI_ARVALID is asserted. axi_arready is
// de-asserted when reset (active low) is asserted.
// The read address is also latched when S_AXI_ARVALID is
// asserted. axi_araddr is reset to zero on reset assertion.

always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        axi_arready <= 1'b0;
        axi_araddr <= 32'b0;
    end
    else
    begin
        if (~axi_arready && S_AXI_ARVALID)
        begin
            // indicates that the slave has accepted the valid read address
            axi_arready <= 1'b1;
            // Read address latching
            axi_araddr <= S_AXI_ARADDR;
        end
    end
    else
    begin
        axi_arready <= 1'b0;
    end
end
end

```

```

end

// Implement axi_arvalid generation
// axi_rvalid is asserted for one S_AXI_ACLK clock cycle when both
// S_AXI_ARVALID and axi_arready are asserted. The slave registers
// data are available on the axi_rdata bus at this instance. The
// assertion of axi_rvalid marks the validity of read data on the
// bus and axi_rresp indicates the status of read transaction. axi_rvalid
// is deasserted on reset (active low). axi_rresp and axi_rdata are
// cleared to zero on reset (active low).
always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        axi_rvalid <= 0;
        axi_rresp <= 0;
    end
    else
    begin
        if (axi_arready && S_AXI_ARVALID && ~axi_rvalid)
        begin
            // Valid read data is available at the read data bus
            axi_rvalid <= 1'b1;
            axi_rresp <= 2'b0; // 'OKAY' response
        end
        else if (axi_rvalid && S_AXI_RREADY)
        begin
            // Read data is accepted by the master
            axi_rvalid <= 1'b0;
        end
    end
end

// Implement memory mapped register select and read logic generation
// Slave register read enable is asserted when valid address is available
// and the slave is ready to accept the read address.
assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
always @(*)
begin
    // Address decoding for reading registers
    case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
        2'h0 : reg_data_out <= slv_reg0;
        2'h1 : reg_data_out <= slv_reg1;
        2'h2 : reg_data_out <= slv_reg2;
        2'h3 : reg_data_out <= slv_reg3;
        default : reg_data_out <= 0;
    endcase
end

// Output register or memory read data
always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        axi_rdata <= 0;
    end
    else
    begin
        // When there is a valid read address (S_AXI_ARVALID) with
        // acceptance of read address by the slave (axi_arready),
        // output the read data
        if (slv_reg_rden)

```

```

        begin
            axi_rdata <= reg_data_out;    // register read data
        end
    end
end

// Add user logic here
reg [31:0] counter = 0;

always @(posedge S_AXI_ACLK) begin
    if(counter < PWM_COUNTER_MAX -1)
        counter <= counter +1;
    else
        counter <= 0;
    end

    assign PWM0 = slv_reg0 < counter ? 1'b0 : 1'b1;
    // User logic ends

endmodule

```

이와같이 수정이 완료 되었으면
 탑모듈도 수정을 해주도록 한다.

My_PWM_Core_v1_0

```

`timescale 1 ns / 1 ps

module My_PWM_Core_v1_0 #
(
    // Users to add parameters here
    parameter integer PWM_COUNTER_MAX = 128,
    // User parameters ends
    // Do not modify the parameters beyond this line

    // Parameters of Axi Slave Bus Interface S00_AXI
    parameter integer C_S00_AXI_DATA_WIDTH    = 32,
    parameter integer C_S00_AXI_ADDR_WIDTH    = 4
)
(
    // Users to add ports here
    output wire PWM0,
    // User ports ends
    // Do not modify the ports beyond this line

    // Ports of Axi Slave Bus Interface S00_AXI
    input wire  s00_axi_aclk,
    input wire  s00_axi_aresetn,
    input wire [C_S00_AXI_ADDR_WIDTH-1 : 0] s00_axi_awaddr,
    input wire [2 : 0] s00_axi_awprot,
    input wire  s00_axi_awvalid,
    output wire s00_axi_awready,
    input wire [C_S00_AXI_DATA_WIDTH-1 : 0] s00_axi_wdata,
    input wire [(C_S00_AXI_DATA_WIDTH/8)-1 : 0] s00_axi_wstrb,
    input wire  s00_axi_wvalid,
    output wire s00_axi_wready,
    output wire [1 : 0] s00_axi_bresp,
    output wire s00_axi_bvalid,

```

```

        input wire s00_axi_bready,
        input wire [C_S00_AXI_ADDR_WIDTH-1 : 0] s00_axi_araddr,
        input wire [2 : 0] s00_axi_arprot,
        input wire s00_axi_arvalid,
        output wire s00_axi_arready,
        output wire [C_S00_AXI_DATA_WIDTH-1 : 0] s00_axi_rdata,
        output wire [1 : 0] s00_axi_rresp,
        output wire s00_axi_rvalid,
        input wire s00_axi_rready
    );
// Instantiation of Axi Bus Interface S00_AXI
    My_PWM_Core_v1_0_S00_AXI # (
        .C_S_AXI_DATA_WIDTH(C_S00_AXI_DATA_WIDTH),
        .C_S_AXI_ADDR_WIDTH(C_S00_AXI_ADDR_WIDTH),
        .PWM_COUNTER_MAX(PWM_COUNTER_MAX)
    ) My_PWM_Core_v1_0_S00_AXI_inst (
        .PWM0(PWM0),
        .S_AXI_ACLK(s00_axi_aclk),
        .S_AXI_ARESETN(s00_axi_aresetn),
        .S_AXI_AWADDR(s00_axi_awaddr),
        .S_AXI_AWPROT(s00_axi_awprot),
        .S_AXI_AWVALID(s00_axi_awvalid),
        .S_AXI_AWREADY(s00_axi_awready),
        .S_AXI_WDATA(s00_axi_wdata),
        .S_AXI_WSTRB(s00_axi_wstrb),
        .S_AXI_WVALID(s00_axi_wvalid),
        .S_AXI_WREADY(s00_axi_wready),
        .S_AXI_BRESP(s00_axi_bresp),
        .S_AXI_BVALID(s00_axi_bvalid),
        .S_AXI_BREADY(s00_axi_bready),
        .S_AXI_ARADDR(s00_axi_araddr),
        .S_AXI_ARPROT(s00_axi_arprot),
        .S_AXI_ARVALID(s00_axi_arvalid),
        .S_AXI_ARREADY(s00_axi_arready),
        .S_AXI_RDATA(s00_axi_rdata),
        .S_AXI_RRESP(s00_axi_rresp),
        .S_AXI_RVALID(s00_axi_rvalid),
        .S_AXI_RREADY(s00_axi_rready)
    );


    // Add user logic here

    // User logic ends

endmodule

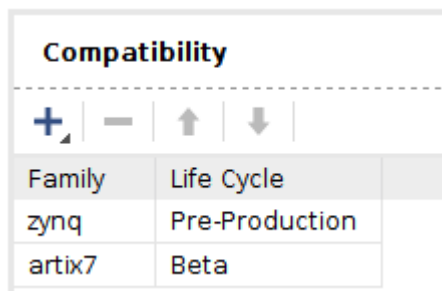
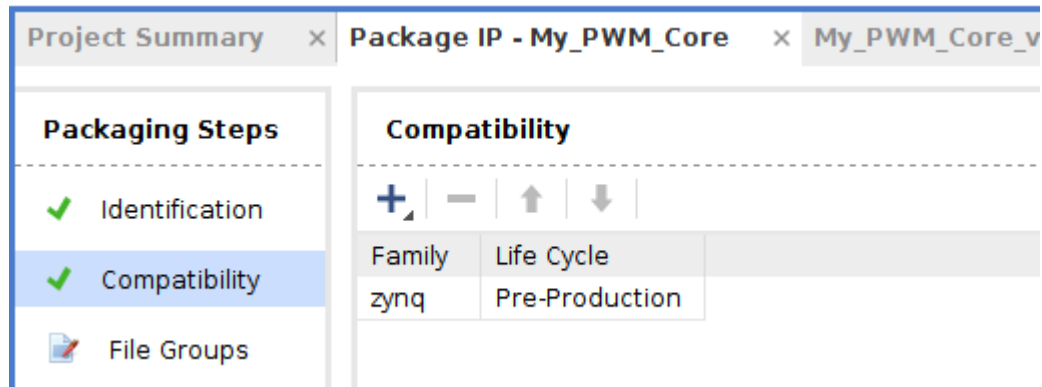
```

수정이 완료되었으면 왼쪽 바에서 Package IP 를 클릭해 주도록 한다.

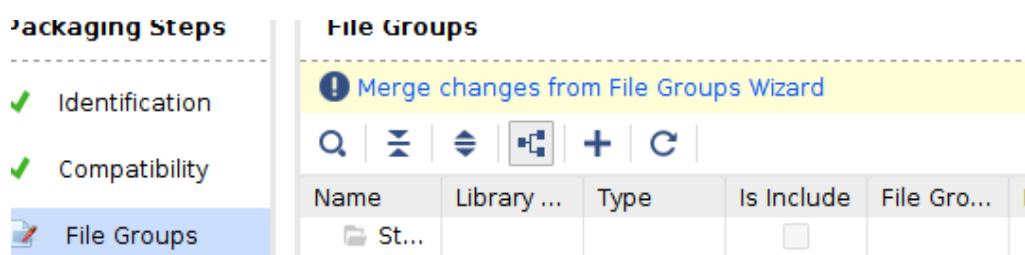
 IP Catalog

Package IP

Compatibility 에서 + 버튼 클릭후 artix7 을 추가하여 두번째 사진과 같이 만들어준다

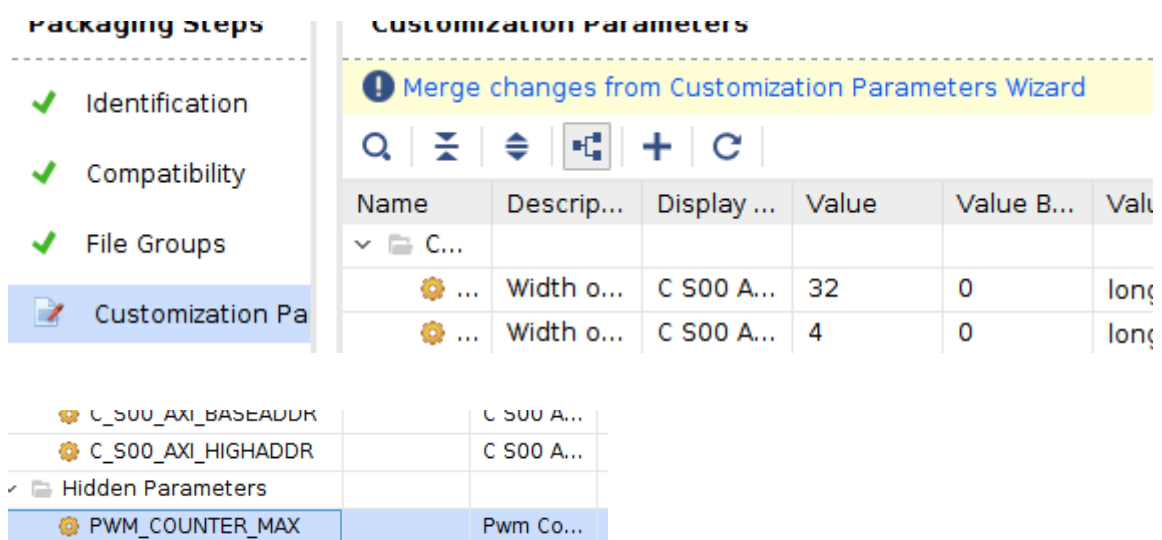


File Groups 를 클릭후 Merge Chages from File Groups Wizard 를 클릭한다.



Customaization parameters 를 클릭후 Merge 를 하면 Hidden 폴더 아래 내가 만든 파라미터(PWM_COUNTER_MAX)가 나타날 것이다.

PWM_COUNTER_MAX 를 더블 클릭후 열어준다.



더블 클릭하면 이와같은 창이 뜰것이다. 파라미터를 아래와 동일하게 설정해 준다.

✕

⌵

Edit IP Parameter

Use the options below to customize how the parameter will appear in the Customization GUI for users of the IP.

Name:

PWM_COUNTER_MAX

☒ Visible in Customization GUI

☒ Show Name

Display Name:

Pwm Counter Max

Tooltip:

Format:

long

Editable:

Yes

Dependency:

No

☒ Specify Range

Type:

Range of integers

Minimum:

0

Maximum:

3000000

☒ Show Range

Show As:

Not Applicable

Layout:

Not Applicable

Default Value:

128

OK

Cancel

설정 완료 후 Customization GUI 를 클릭하면 Pwm Counter Max 가 그룹에 속해있지 않을 것이다.
Pwm Counter Max 를 드래그하여 Page 0 폴더 안에 넣어준다.

✓ Compatibility

✓ File Groups

✓ Customization Par

Ports and Interfac

Addressing and M

Customization GUI

Review and Packa

Window

Component Name

Page 0

C S00 AXI DATA WIDTH

C S00 AXI ADDR WIDTH

C S00 AXI BASEADDR

C S00 AXI HIGHADDR

Pwm Counter Max

Hidden Parameters

→

Window

Component Name

Page 0

Pwm Counter Max

C S00 AXI DATA WIDTH

C S00 AXI ADDR WIDTH

C S00 AXI BASEADDR

C S00 AXI HIGHADDR

Hidden Parameters

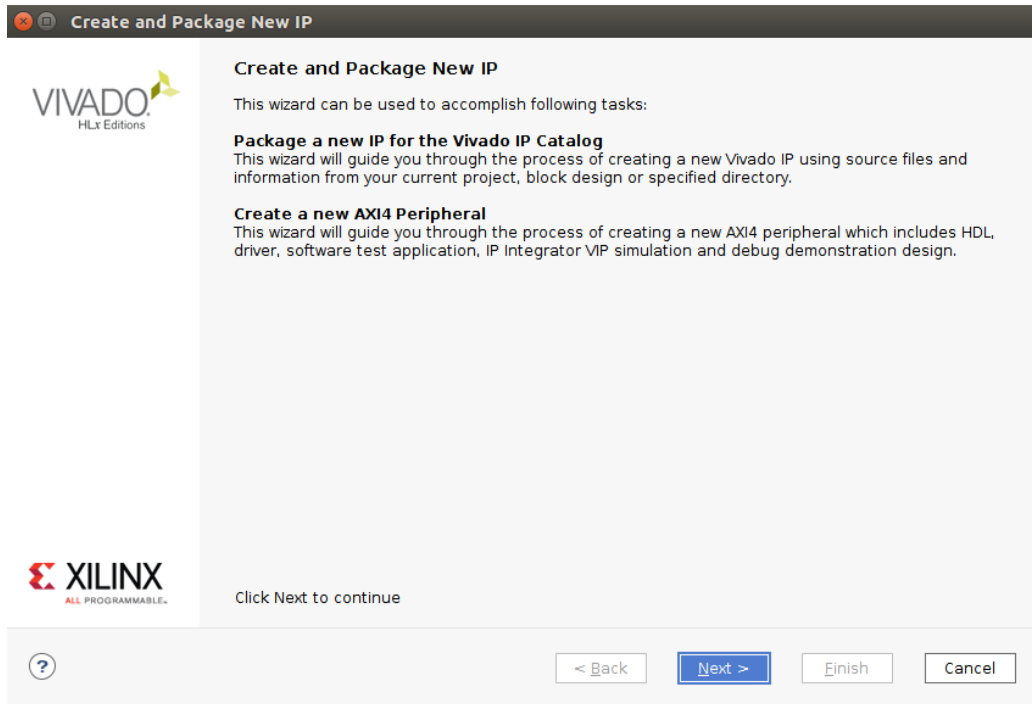
그후 Review and Package 에 가서 IP 를 생성해준다.

이제 PWM Custom IP 는 완성되었다. ECAP 도 이와같은 방식으로 추가 시켜주도록 한다.

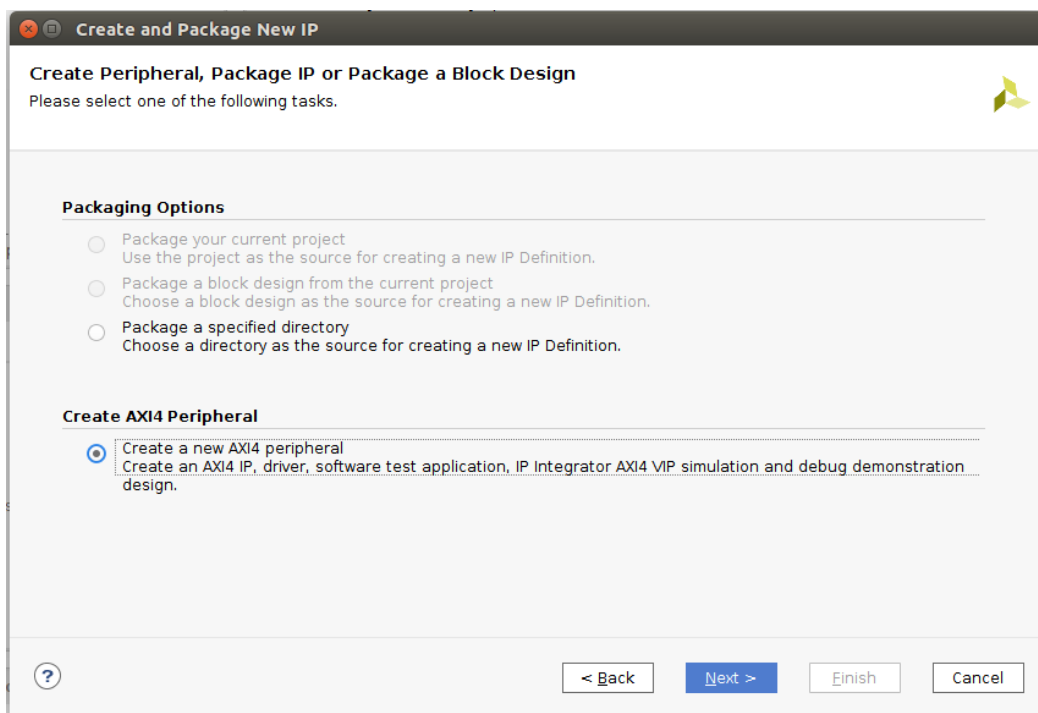
Ecap custom IP create

아까와 같은 방식으로 Vivado 에서 Tools → Create and Package New IP 를 선택한다.

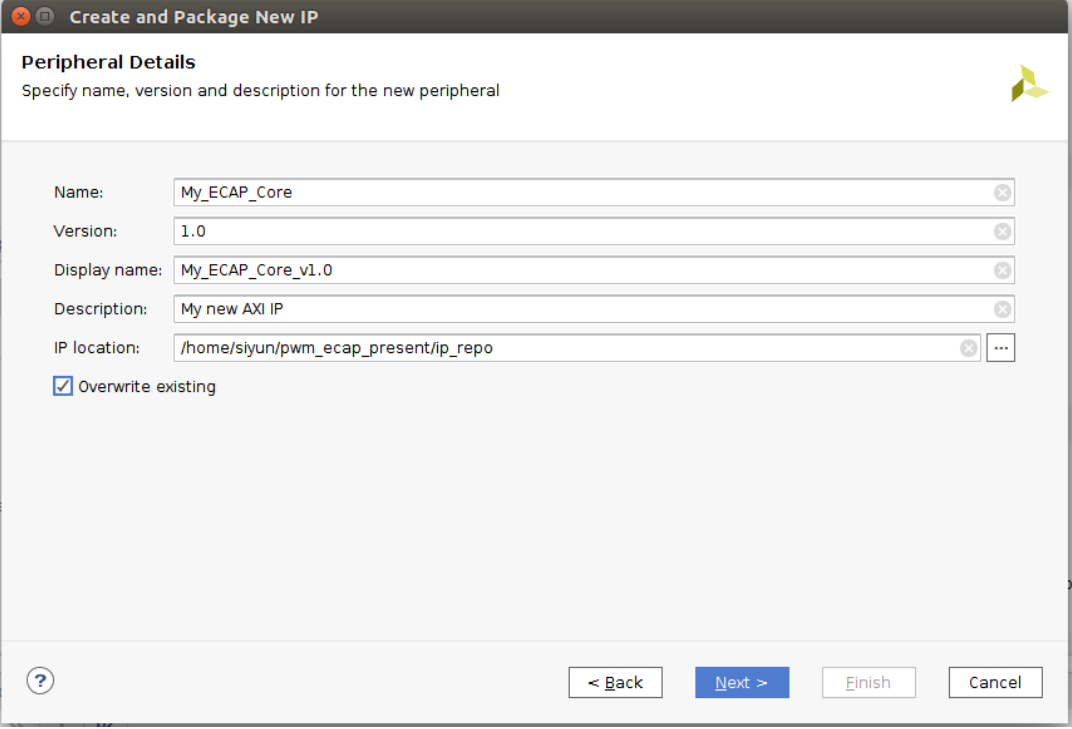
Next



Create a new AXI4 peripheral 선택 후 Next



이름 설정 Name 에 My_ECAP_Core 입력 후 Next



Create and Package New IP

Peripheral Details
Specify name, version and description for the new peripheral

Name: My_ECAP_Core

Version: 1.0

Display name: My_ECAP_Core_v1.0

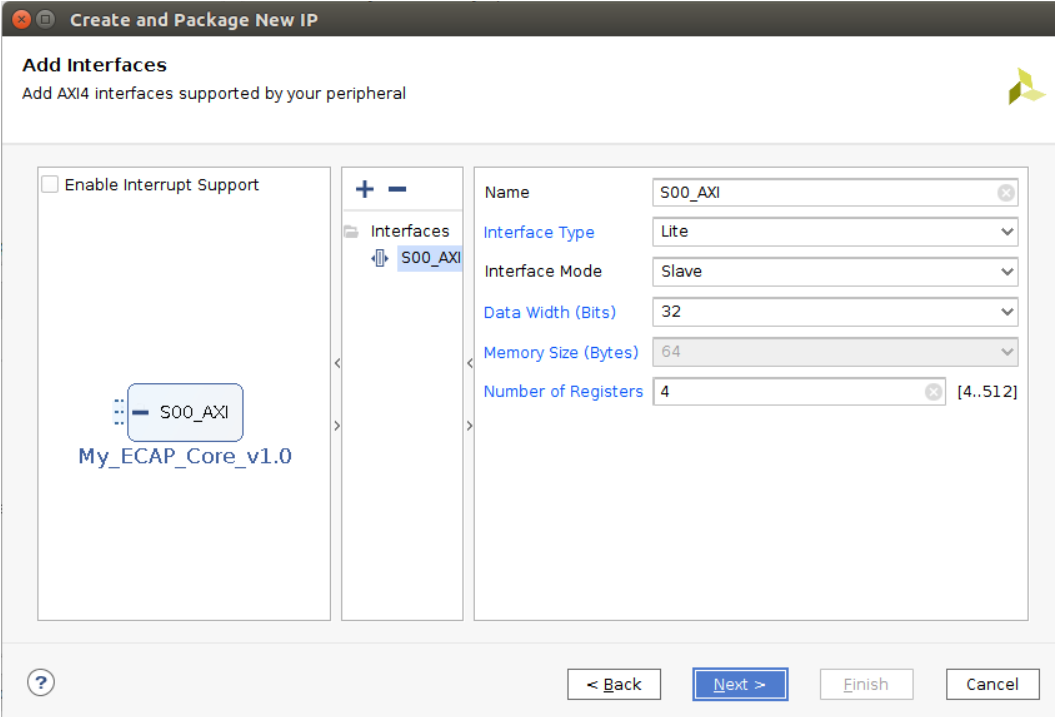
Description: My new AXI IP

IP location: /home/siyun/pwm_ecap_present/ip_repo

☒ Overwrite existing

< Back Next > Finish Cancel

아무 설정도 해주지않고 Next



Create and Package New IP

Add Interfaces
Add AXI4 interfaces supported by your peripheral

☐ Enable Interrupt Support

Interfaces

- S00_AXI

S00_AXI

My_ECAP_Core_v1.0

Name: S00_AXI

Interface Type: Lite

Interface Mode: Slave

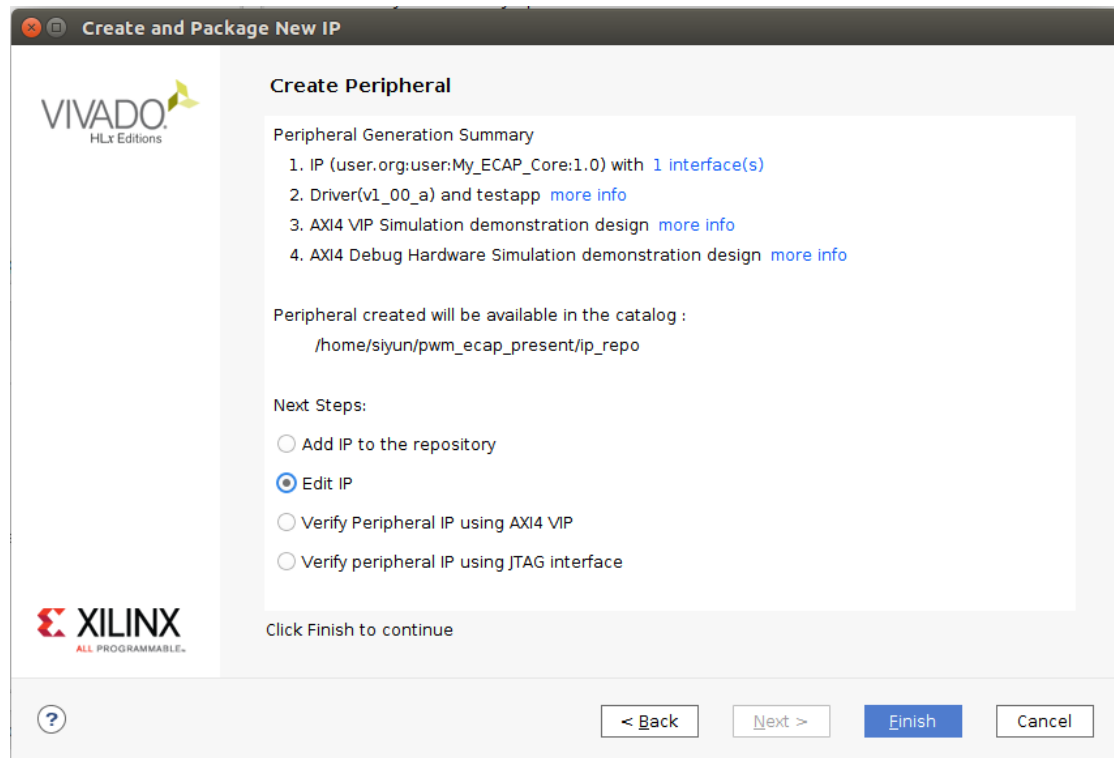
Data Width (Bits): 32

Memory Size (Bytes): 64

Number of Registers: 4 [4..512]

< Back Next > Finish Cancel

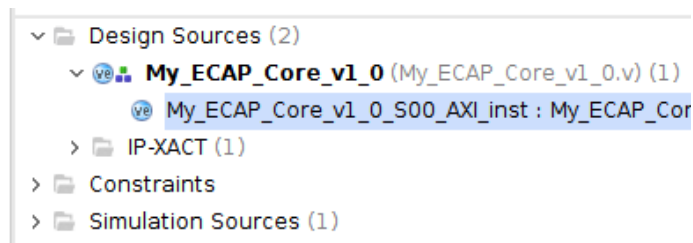
Edit IP 선택 후 Finish



아까와 마찬가지로 좌측 Source 에 베릴로그 코드가 두개 생길것이다.

위에있는게 탑모듈이고 아래있는게 하위 모듈이다.

아까와 마찬가지로 하위모듈을 먼저 수정하도록 한다.



My_ECAP_Core_v1_0_S00_AXI_inst

```
`timescale 1 ns / 1 ps

module My_ECAP_Core_v1_0_S00_AXI #
(
    // Users to add parameters here

    // User parameters ends
    // Do not modify the parameters beyond this line

    // Width of S_AXI data bus
    parameter integer C_S_AXI_DATA_WIDTH      = 32,
    // Width of S_AXI address bus
    parameter integer C_S_AXI_ADDR_WIDTH      = 4
)
(
    // Users to add ports here
    input wire pwm_sig,
```

```

// User ports ends
// Do not modify the ports beyond this line

// Global Clock Signal
input wire S_AXI_ACLK,
// Global Reset Signal. This Signal is Active LOW
input wire S_AXI_ARESETN,
// Write address (issued by master, accepted by Slave)
input wire [C_S_AXI_ADDR_WIDTH-1 : 0] S_AXI_AWADDR,
// Write channel Protection type. This signal indicates the
// privilege and security level of the transaction, and whether
// the transaction is a data access or an instruction access.
input wire [2 : 0] S_AXI_AWPROT,
// Write address valid. This signal indicates that the master signaling
// valid write address and control information.
input wire S_AXI_AWVALID,
// Write address ready. This signal indicates that the slave is ready
// to accept an address and associated control signals.
output wire S_AXI_AWREADY,
// Write data (issued by master, accepted by Slave)
input wire [C_S_AXI_DATA_WIDTH-1 : 0] S_AXI_WDATA,
// Write strobes. This signal indicates which byte lanes hold
// valid data. There is one write strobe bit for each eight
// bits of the write data bus.
input wire [(C_S_AXI_DATA_WIDTH/8)-1 : 0] S_AXI_WSTRB,
// Write valid. This signal indicates that valid write
// data and strobes are available.
input wire S_AXI_WVALID,
// Write ready. This signal indicates that the slave
// can accept the write data.
output wire S_AXI_WREADY,
// Write response. This signal indicates the status
// of the write transaction.
output wire [1 : 0] S_AXI_BRESP,
// Write response valid. This signal indicates that the channel
// is signaling a valid write response.
output wire S_AXI_BVALID,
// Response ready. This signal indicates that the master
// can accept a write response.
input wire S_AXI_BREADY,
// Read address (issued by master, accepted by Slave)
input wire [C_S_AXI_ADDR_WIDTH-1 : 0] S_AXI_ARADDR,
// Protection type. This signal indicates the privilege
// and security level of the transaction, and whether the
// transaction is a data access or an instruction access.
input wire [2 : 0] S_AXI_ARPROT,
// Read address valid. This signal indicates that the channel
// is signaling valid read address and control information.
input wire S_AXI_ARVALID,
// Read address ready. This signal indicates that the slave is
// ready to accept an address and associated control signals.
output wire S_AXI_ARREADY,
// Read data (issued by slave)
output wire [C_S_AXI_DATA_WIDTH-1 : 0] S_AXI_RDATA,
// Read response. This signal indicates the status of the
// read transfer.
output wire [1 : 0] S_AXI_RRESP,
// Read valid. This signal indicates that the channel is
// signaling the required read data.
output wire S_AXI_RVALID,
// Read ready. This signal indicates that the master can
// accept the read data and response information.

```

```

        input wire S_AXI_RREADY

    );

    // AXI4LITE signals
    reg [C_S_AXI_ADDR_WIDTH-1 : 0]    axi_awaddr;
    reg    axi_awready;
    reg    axi_wready;
    reg [1 : 0]    axi_bresp;
    reg    axi_bvalid;
    reg [C_S_AXI_ADDR_WIDTH-1 : 0]    axi_araddr;
    reg    axi_arready;
    reg [C_S_AXI_DATA_WIDTH-1 : 0]    axi_rdata;
    reg [1 : 0]    axi_rresp;
    reg    axi_rvalid;

    // Example-specific design signals
    // local parameter for addressing 32 bit / 64 bit C_S_AXI_DATA_WIDTH
    // ADDR_LSB is used for addressing 32/64 bit registers/memories
    // ADDR_LSB = 2 for 32 bits (n downto 2)
    // ADDR_LSB = 3 for 64 bits (n downto 3)
    localparam integer ADDR_LSB = (C_S_AXI_DATA_WIDTH/32) + 1;
    localparam integer OPT_MEM_ADDR_BITS = 1;
    //-----
    //-- Signals for user logic register space example
    //-----
    //-- Number of Slave Registers 4
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg0;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg1;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg2;
    reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg3;
    wire    slv_reg_rden;
    wire    slv_reg_wren;
    reg [C_S_AXI_DATA_WIDTH-1:0]    reg_data_out;
    integer    byte_index;
    reg    aw_en;

    // I/O Connections assignments

    assign S_AXI_AWREADY    = axi_awready;
    assign S_AXI_WREADY    = axi_wready;
    assign S_AXI_BRESP    = axi_bresp;
    assign S_AXI_BVALID    = axi_bvalid;
    assign S_AXI_ARREADY    = axi_arready;
    assign S_AXI_RDATA    = axi_rdata;
    assign S_AXI_RRESP    = axi_rresp;
    assign S_AXI_RVALID    = axi_rvalid;
    // Implement axi_awready generation
    reg [31:0] duty;
    reg [31:0] period;
    reg [31:0] duty_e;
    reg [31:0] period_e;
    reg [31:0] cap1;
    reg [31:0] cap2;
    reg [31:0] cap3;
    reg past_pwm_sig;
    reg [3:0] rising_edge_flag;
    reg [31:0] counter;
    // axi_awready is asserted for one S_AXI_ACLK clock cycle when both
    // S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_awready is
    // de-asserted when reset is low.

    always @( posedge S_AXI_ACLK )

```

```

begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_awready <= 1'b0;
      aw_en <= 1'b1;
    end
  else
    begin
      if (~axi_awready && S_AXI_AWVALID && S_AXI_WVALID && aw_en)
        begin
          // slave is ready to accept write address when
          // there is a valid write address and write data
          // on the write address and data bus. This design
          // expects no outstanding transactions.
          axi_awready <= 1'b1;
          aw_en <= 1'b0;
        end
      else if (S_AXI_BREADY && axi_bvalid)
        begin
          aw_en <= 1'b1;
          axi_awready <= 1'b0;
        end
      else
        begin
          axi_awready <= 1'b0;
        end
      end
    end
  end

  // Implement axi_awaddr latching
  // This process is used to latch the address when both
  // S_AXI_AWVALID and S_AXI_WVALID are valid.

  always @( posedge S_AXI_ACLK )
  begin
    if ( S_AXI_ARESETN == 1'b0 )
      begin
        axi_awaddr <= 0;
      end
    else
      begin
        if (~axi_awready && S_AXI_AWVALID && S_AXI_WVALID && aw_en)
          begin
            // Write Address latching
            axi_awaddr <= S_AXI_AWADDR;
          end
        end
      end
    end
  end

  // Implement axi_wready generation
  // axi_wready is asserted for one S_AXI_ACLK clock cycle when both
  // S_AXI_AWVALID and S_AXI_WVALID are asserted. axi_wready is
  // de-asserted when reset is low.

  always @( posedge S_AXI_ACLK )
  begin
    if ( S_AXI_ARESETN == 1'b0 )
      begin
        axi_wready <= 1'b0;
      end
    else
      begin

```

```

if (~axi_wready && S_AXI_WVALID && S_AXI_AWVALID && aw_en )
begin
    // slave is ready to accept write data when
    // there is a valid write address and write data
    // on the write address and data bus. This design
    // expects no outstanding transactions.
    axi_wready <= 1'b1;
end
else
begin
    axi_wready <= 1'b0;
end
end
end

// Implement memory mapped register select and write logic generation
// The write data is accepted and written to memory mapped registers when
// axi_awready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted. Write strobes are used to
// select byte enables of slave registers while writing.
// These registers are cleared when reset (active low) is applied.
// Slave register write enable is asserted when valid address and data are available
// and the slave is ready to accept the write address and write data.
assign slv_reg_wren = axi_wready && S_AXI_WVALID && axi_awready && S_AXI_AWVALID;

always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        slv_reg0 <= 0;
        slv_reg1 <= 0;
        slv_reg2 <= 0;
        slv_reg3 <= 0;
    end
    else begin
        if (slv_reg_wren)
        begin
            case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
                2'h0:
                    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
                    if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                        // Respective byte enables are asserted as per write strobes
                        // Slave register 0
                        slv_reg0[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                    end
                2'h1:
                    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
                    if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                        // Respective byte enables are asserted as per write strobes
                        // Slave register 1
                        slv_reg1[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                    end
                2'h2:
                    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
                    if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                        // Respective byte enables are asserted as per write strobes
                        // Slave register 2
                        slv_reg2[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                    end
                2'h3:
                    for ( byte_index = 0; byte_index <= (C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
                    if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                        // Respective byte enables are asserted as per write strobes

```

```

        // Slave register 3
        slv_reg3[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
    end
    default : begin
        slv_reg0 <= slv_reg0;
        slv_reg1 <= slv_reg1;
        slv_reg2 <= slv_reg2;
        slv_reg3 <= slv_reg3;
    end
endcase
end
end
end

// Implement write response logic generation
// The write response and response valid signals are asserted by the slave
// when axi_wready, S_AXI_WVALID, axi_wready and S_AXI_WVALID are asserted.
// This marks the acceptance of address and indicates the status of
// write transaction.

always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        axi_bvalid <= 0;
        axi_bresp <= 2'b0;
    end
    else
    begin
        if (axi_awready && S_AXI_AWVALID && ~axi_bvalid && axi_wready && S_AXI_WVALID)
        begin
            // indicates a valid write response is available
            axi_bvalid <= 1'b1;
            axi_bresp <= 2'b0; // 'OKAY' response
        end
        // work error responses in future
    end
    else
    begin
        if (S_AXI_BREADY && axi_bvalid)
        //check if bready is asserted while bvalid is high)
        //(there is a possibility that bready is always asserted high)
        begin
            axi_bvalid <= 1'b0;
        end
    end
end
end

// Implement axi_arready generation
// axi_arready is asserted for one S_AXI_ACLK clock cycle when
// S_AXI_ARVALID is asserted. axi_arready is
// de-asserted when reset (active low) is asserted.
// The read address is also latched when S_AXI_ARVALID is
// asserted. axi_araddr is reset to zero on reset assertion.

always @( posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        axi_arready <= 1'b0;
        axi_araddr <= 32'b0;
    end
    else

```



```

begin
  if (~axi_arready && S_AXI_ARVALID)
    begin
      // indicates that the slave has accepted the valid read address
      axi_arready <= 1'b1;
      // Read address latching
      axi_araddr <= S_AXI_ARADDR;
    end
  else
    begin
      axi_arready <= 1'b0;
    end
  end
end

// Implement axi_rvalid generation
// axi_rvalid is asserted for one S_AXI_ACLK clock cycle when both
// S_AXI_ARVALID and axi_arready are asserted. The slave registers
// data are available on the axi_rdata bus at this instance. The
// assertion of axi_rvalid marks the validity of read data on the
// bus and axi_rresp indicates the status of read transaction. axi_rvalid
// is deasserted on reset (active low). axi_rresp and axi_rdata are
// cleared to zero on reset (active low).
always @(posedge S_AXI_ACLK)
begin
  if ( S_AXI_ARESETN == 1'b0 )
    begin
      axi_rvalid <= 0;
      axi_rresp <= 0;
    end
  else
    begin
      if (axi_arready && S_AXI_ARVALID && ~axi_rvalid)
        begin
          // Valid read data is available at the read data bus
          axi_rvalid <= 1'b1;
          axi_rresp <= 2'b0; // 'OKAY' response
        end
      else if (axi_rvalid && S_AXI_RREADY)
        begin
          // Read data is accepted by the master
          axi_rvalid <= 1'b0;
        end
    end
  end
end

// Implement memory mapped register select and read logic generation
// Slave register read enable is asserted when valid address is available
// and the slave is ready to accept the read address.
assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
always @(*)
begin
  // Address decoding for reading registers
  case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
    2'h0 : reg_data_out <= period;
    2'h1 : reg_data_out <= duty;
    2'h2 : reg_data_out <= slv_reg2;
    2'h3 : reg_data_out <= slv_reg3;
    default : reg_data_out <= 0;
  endcase
end

```

```

// Output register or memory read data
always @(posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        axi_rdata <= 0;
    end
    else
    begin
        // When there is a valid read address (S_AXI_ARVALID) with
        // acceptance of read address by the slave (axi_arready),
        // output the read data
        if (slv_reg_rden)
        begin
            axi_rdata <= reg_data_out; // register read data
        end
    end
end

// Add user logic here
always @(posedge S_AXI_ACLK )
begin
    if ( S_AXI_ARESETN == 1'b0 )
    begin
        axi_rdata <= 0;
    end
    else
    begin
        // When there is a valid read address (S_AXI_ARVALID) with
        // acceptance of read address by the slave (axi_arready),
        // output the read data
        if (slv_reg_rden)
        begin
            axi_rdata <= reg_data_out; // register read data
        end
    end
end

// Add user logic here
always @(posedge S_AXI_ACLK) begin

if(pwm_sig == 1 && past_pwm_sig ==0) begin
    rising_edge_flag = rising_edge_flag + 4'd1;

    if(rising_edge_flag == 4'd1) begin
        cap1<=counter;
    end

    else if(rising_edge_flag == 4'd2) begin
        cap3<=counter;

        duty_e <= cap2 - cap1;
        duty <= duty_e;

        if(cap3 > cap1) begin

            period_e <= cap3 - cap1;
            end
            else if(cap1 > cap3) begin

                period_e <= cap1 - cap3;
            end
        end
    end

```

```

        end

        period <= period_e;

        counter <= 32'd0;
        rising_edge_flag <= 4'd0;
    end

end

else if(pwm_sig == 0 && past_pwm_sig ==1) begin
    cap2<=counter;
end
past_pwm_sig <= pwm_sig;
counter <= counter + 32'd1;

end
// User logic ends

endmodule

```

My_ECAP_Core_v1_0

```

`timescale 1 ns / 1 ps

module My_ECAP_Core_v1_0 #
(
    // Users to add parameters here

    // User parameters ends
    // Do not modify the parameters beyond this line

    // Parameters of Axi Slave Bus Interface S00_AXI
    parameter integer C_S00_AXI_DATA_WIDTH    = 32,
    parameter integer C_S00_AXI_ADDR_WIDTH    = 4
)
(
    // Users to add ports here
    input wire pwm_sig,
    // User ports ends
    // Do not modify the ports beyond this line

    // Ports of Axi Slave Bus Interface S00_AXI
    input wire s00_axi_aclk,
    input wire s00_axi_aresetn,
    input wire [C_S00_AXI_ADDR_WIDTH-1 : 0] s00_axi_awaddr,
    input wire [2 : 0] s00_axi_awprot,
    input wire s00_axi_awvalid,
    output wire s00_axi_awready,
    input wire [C_S00_AXI_DATA_WIDTH-1 : 0] s00_axi_wdata,
    input wire [(C_S00_AXI_DATA_WIDTH/8)-1 : 0] s00_axi_wstrb,
    input wire s00_axi_wvalid,
    output wire s00_axi_wready,
    output wire [1 : 0] s00_axi_bresp,
    output wire s00_axi_bvalid,
    input wire s00_axi_bready,
    input wire [C_S00_AXI_ADDR_WIDTH-1 : 0] s00_axi_araddr,
    input wire [2 : 0] s00_axi_arprot,

```

```

        input wire s00_axi_arvalid,
        output wire s00_axi_arready,
        output wire [C_S00_AXI_DATA_WIDTH-1 : 0] s00_axi_rdata,
        output wire [1 : 0] s00_axi_rresp,
        output wire s00_axi_rvalid,
        input wire s00_axi_rready
    );
// Instantiation of Axi Bus Interface S00_AXI
    My_ECAP_Core_v1_0_S00_AXI # (
        .C_S_AXI_DATA_WIDTH(C_S00_AXI_DATA_WIDTH),
        .C_S_AXI_ADDR_WIDTH(C_S00_AXI_ADDR_WIDTH)
    ) My_ECAP_Core_v1_0_S00_AXI_inst (
        .pwm_sig(pwm_sig),
        .S_AXI_ACLK(s00_axi_aclk),
        .S_AXI_ARESETN(s00_axi_aresetn),
        .S_AXI_AWADDR(s00_axi_awaddr),
        .S_AXI_AWPROT(s00_axi_awprot),
        .S_AXI_AWVALID(s00_axi_awvalid),
        .S_AXI_AWREADY(s00_axi_awready),
        .S_AXI_WDATA(s00_axi_wdata),
        .S_AXI_WSTRB(s00_axi_wstrb),
        .S_AXI_WVALID(s00_axi_wvalid),
        .S_AXI_WREADY(s00_axi_wready),
        .S_AXI_BRESP(s00_axi_bresp),
        .S_AXI_BVALID(s00_axi_bvalid),
        .S_AXI_BREADY(s00_axi_bready),
        .S_AXI_ARADDR(s00_axi_araddr),
        .S_AXI_ARPROT(s00_axi_arprot),
        .S_AXI_ARVALID(s00_axi_arvalid),
        .S_AXI_ARREADY(s00_axi_arready),
        .S_AXI_RDATA(s00_axi_rdata),
        .S_AXI_RRESP(s00_axi_rresp),
        .S_AXI_RVALID(s00_axi_rvalid),
        .S_AXI_RREADY(s00_axi_rready)
    );


    // Add user logic here

    // User logic ends

endmodule

```

작성이 완료되었으면 아까와 마찬가지로 Package IP 를 클릭한다.

 IP Catalog


Package IP

Compatiibility 에 artix7 을 추가한다.

Packaging Steps

✓ Identification

✓ Compatibility

 File Groups

Compatibility

+

−

↑

↓

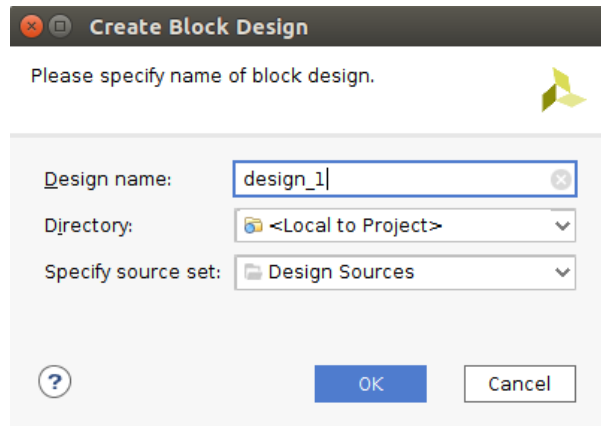
| Family | Life Cycle | |
|--------|----------------|--|
| zynq | Pre-Production | |
| artix7 | Beta | |

File Groups 와 Customization Parameters 두개 다 Merge changes from Customization 을 클릭해준다.
우리가 만든 커스텀 파라미터가 없어서 아까와 같이 파라미터 설정은 없다.

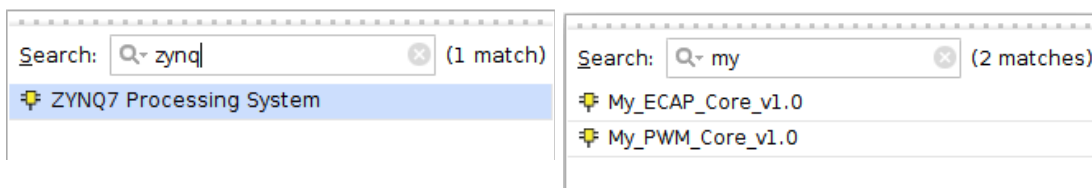


다 설정한후 Review 에서 IP 를 생성한다.

생성후 다시 비바도로 돌아오면 Create Block Design 을 해준다.

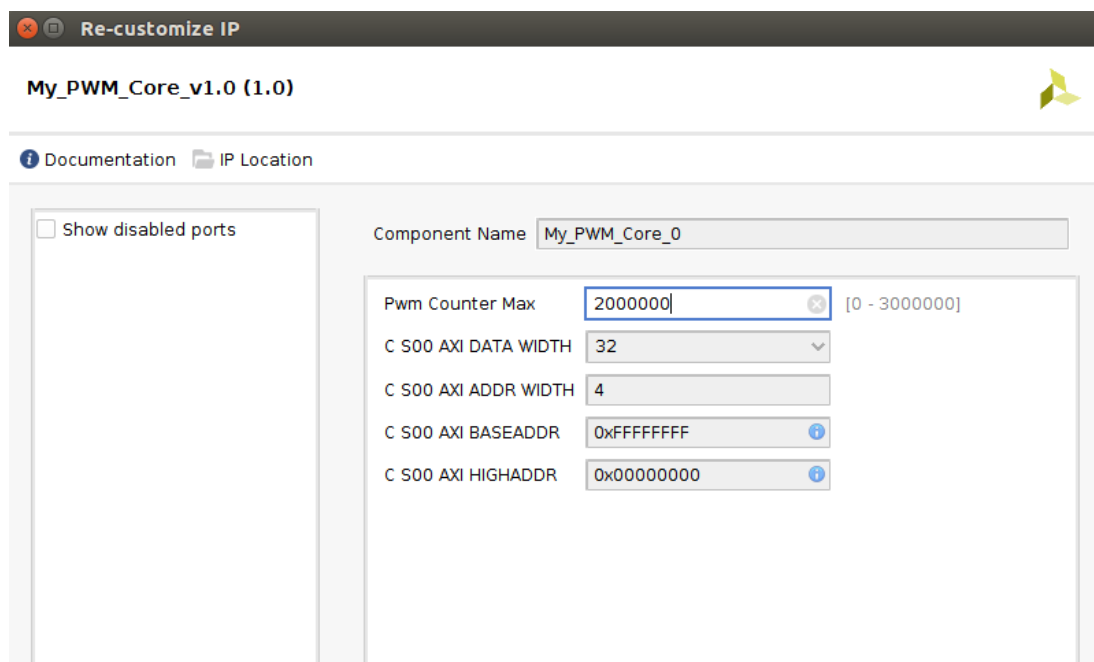


IP 추가 -----



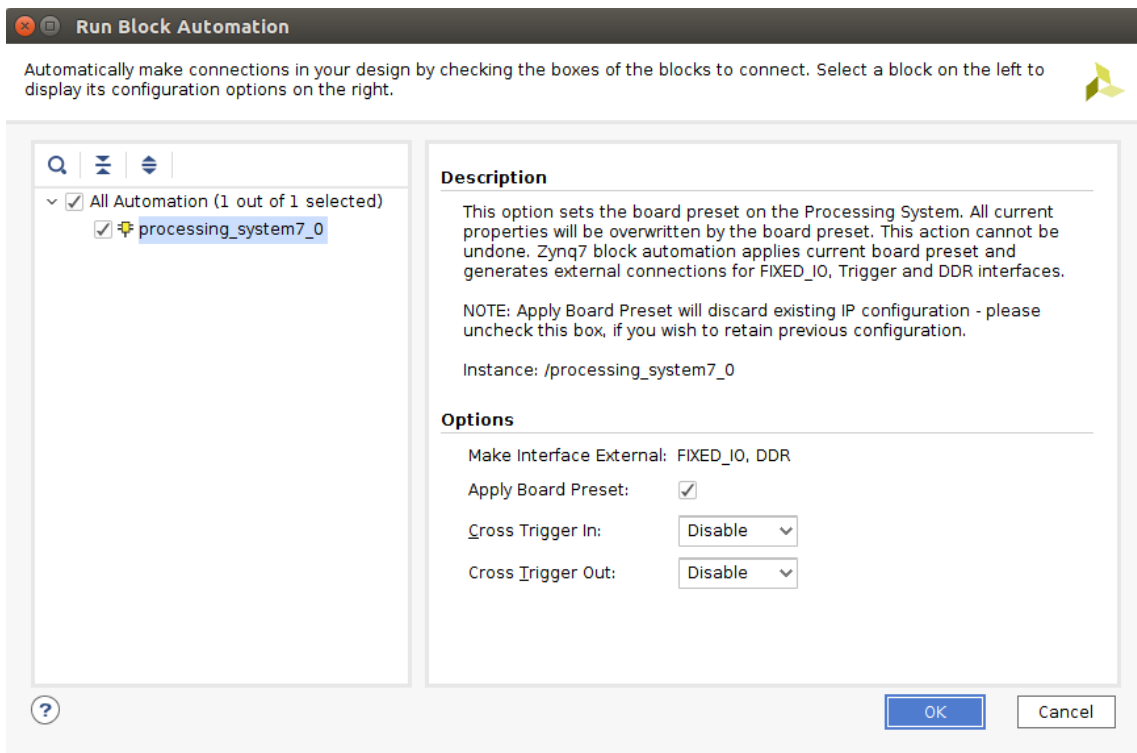
추가할때 먼저 Zynq 프로세서 → My_PWM_Core → My_ECAP_Core 순으로 추가하도록 한다!

추가가 완료되었으면 My_PWM_Core 블럭을 더블클릭하여 아래와 같이 설정한 후 확인 해준다.

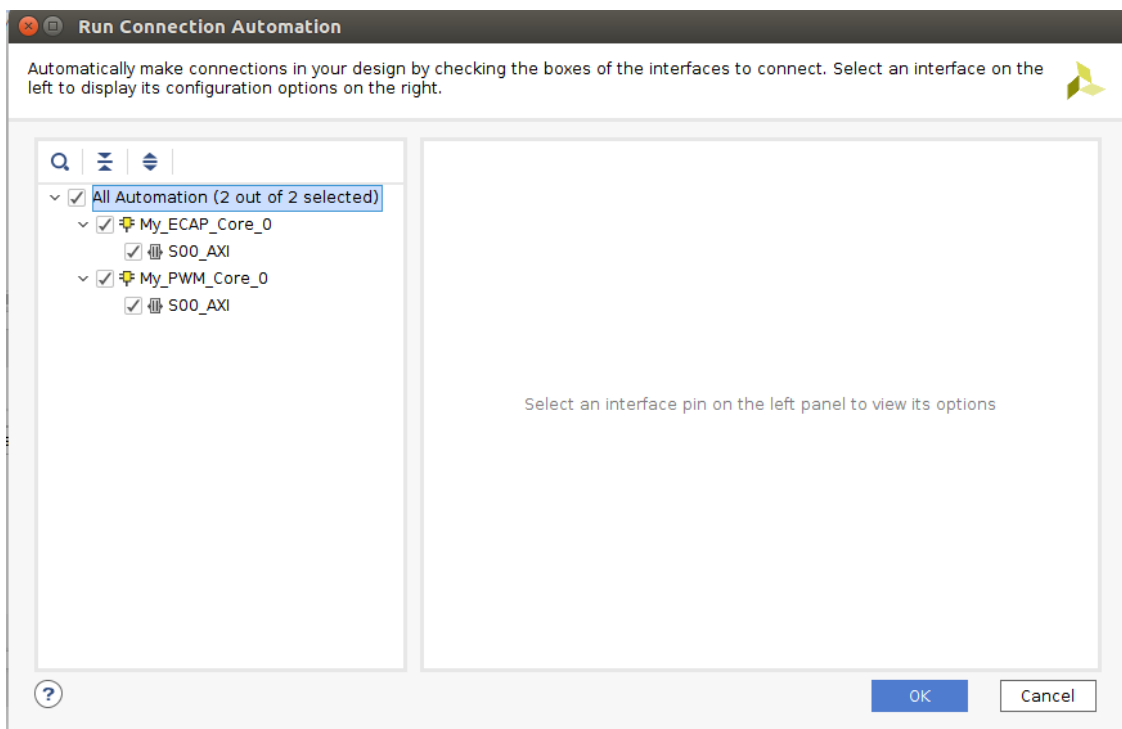


★ Designer Assistance available. [Run Block Automation](#) [Run Connection Automation](#)

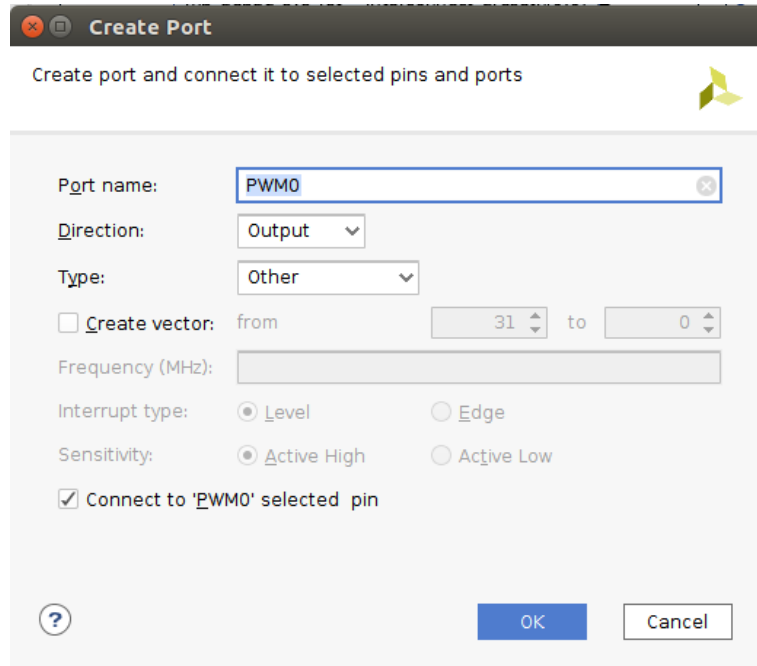
그후 Run Block Automation 을 아래와 같이 해준다.



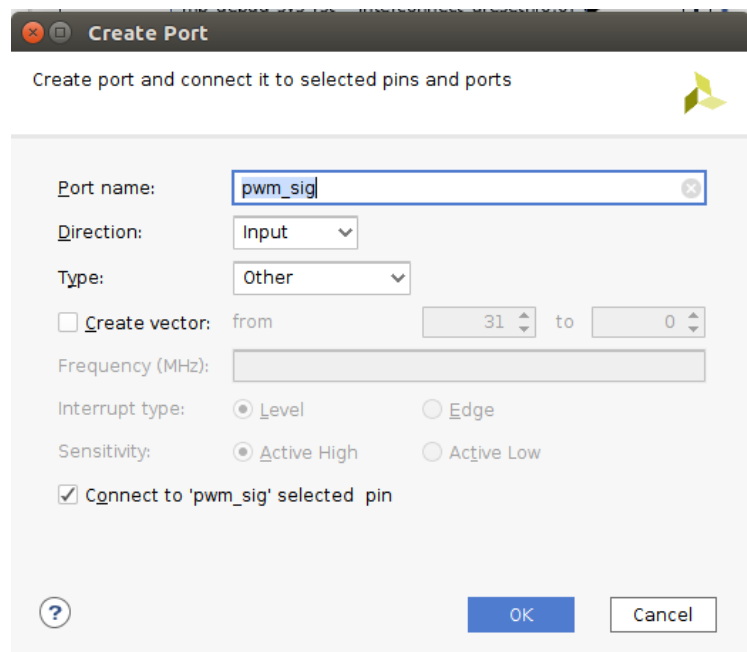
그후 Run Connection Automation 을 아래와 같이 해준다.



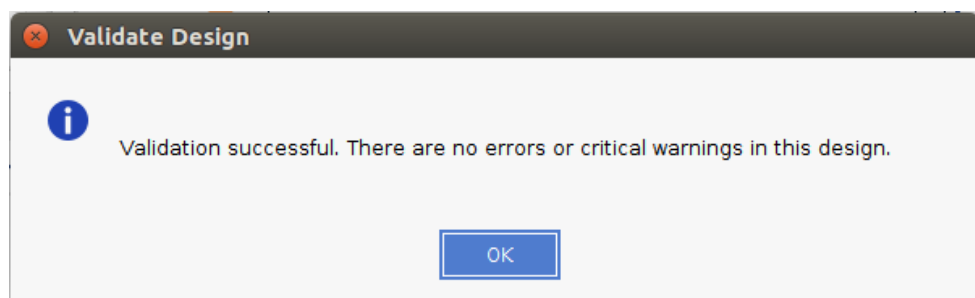
그후 My_PWM_Core Block 에서 PWM0 핀을 선택 후 오른쪽 클릭하여 Create Port 를 클릭 후 OK 를 클릭하여 Port 를 생성해준다.



My_ECAP_Core 블록에 pwm_sig 핀을 클릭하여 오른쪽 선택 후 Create Port 를 클릭한 후 Ok 를 하여 port 를 생성해 준다.

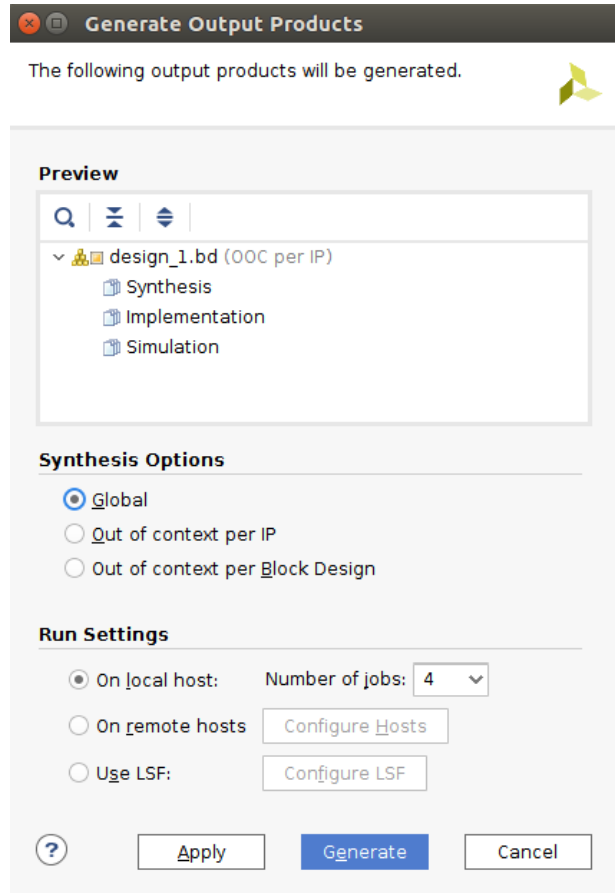


그 후 회로에 오류가 있는지 확인하기 위하여 Validate 버튼을 ☒ 클릭 해 준다.

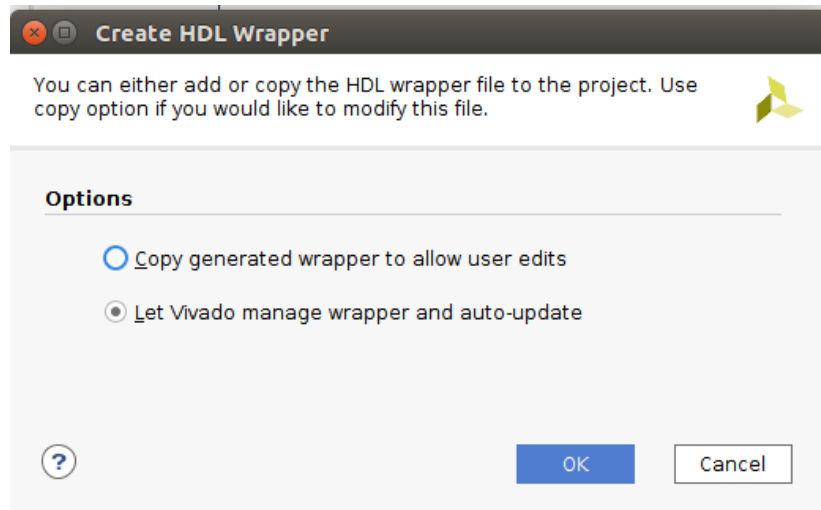


오류가 없다면 이와같은 창이 나올것이다. 그럼 일단 성공이다.

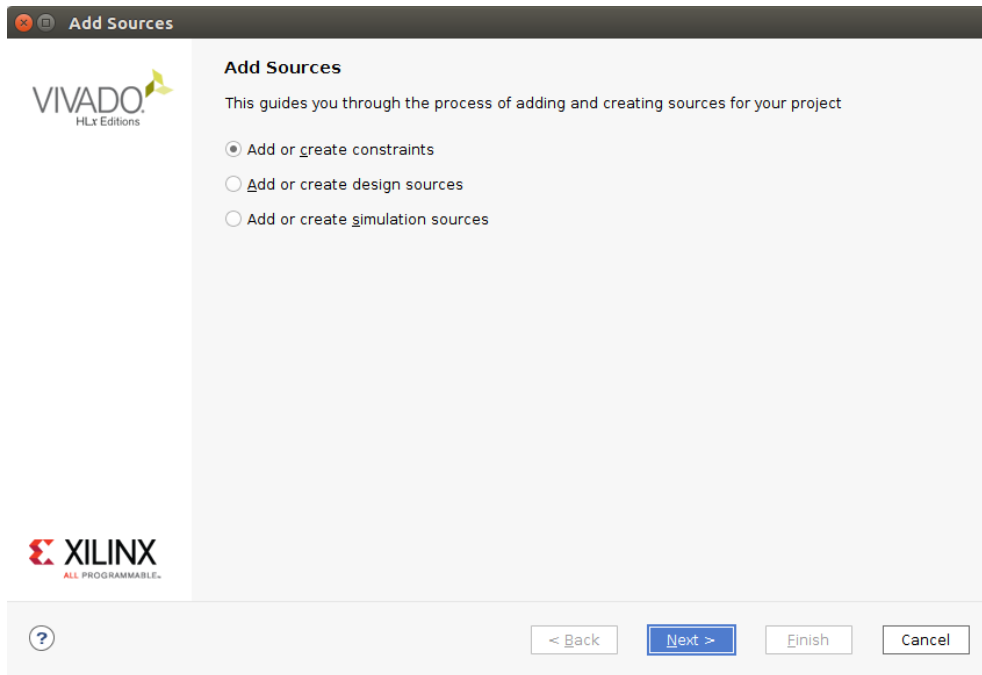
Source 창에 가서 design_1.bd 파일을 오른쪽 클릭 후 Generate Output Products 를 아래와 같이 해준후 Generate 를 클릭 해 준다.



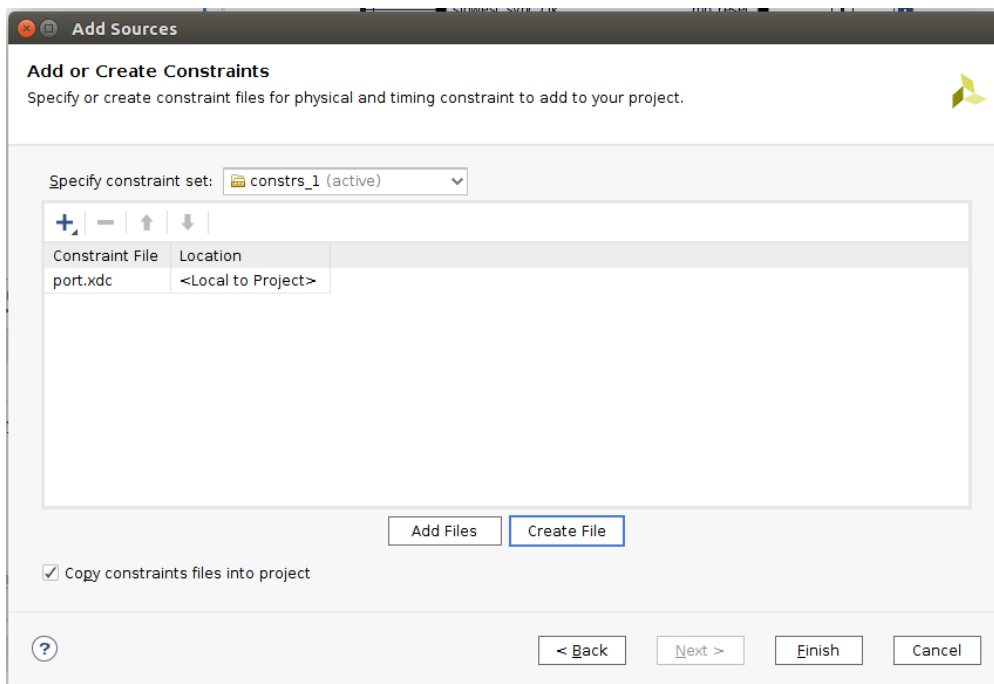
Generate 가 완료 되었다면, 다시 오른쪽 클릭후 Create HDL Wrapper 를 해준다.



이제 Implement 후 포트정보를 기록하기 위해 constraints 소스를 추가 하도록 한다.



나는 이름을 port 로 하여 추가하였다.



추가를 완료하였다면 Run Implementation 을 진행한다.

▼ IMPLEMENTATION

▶ Run Implementation

> Open Implemented Design

Implementaion 이 완료되면 I/O port 로 이동하여 port 설정을 아래와 같이 해주도록 한다.

| Name | Direction | Board Part Pin | Board Part Interface | Neg Diff Pair | Package Pin | Fixed | Bank | I/O Std | Vcco | Vref |
|-----------------------|-----------|----------------|----------------------|---------------|-------------|-------|------------|-------------|------------|----------|
| All ports (132) | | | | | | | | | | |
| > DDR_16577 (71) | INOUT | | | | | ✓ | 502 | (Multiple)* | 1.500 | (Multipl |
| > FIXED_IO_16577 (59) | INOUT | | | | | ✓ | (Multiple) | (Multiple)* | (Multiple) | (Multipl |
| Scalar ports (2) | | | | | | | | | | |
| PWM0 | OUT | | | | V15 | ✓ | 34 | LVC MOS33* | 3.300 | |
| pwm_sig | IN | | | | T11 | ✓ | 34 | LVC MOS33* | 3.300 | |

Scalar ports 에 PWM0 은 PWM 신호가 나오는 OUTPUT 이며
pwm_sig 는 ecap 모듈에 들어가는 PWM_SIGNAL 을 의미한다.

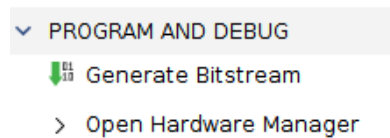
위와 같이 설정해 준후 저장버튼 (ctrl + s) 를 눌러주고 확인을 누르면 xdc 파일에 저절로 포트 정보가 업로드 될 것이다.
업로드 되지 않았다면 아래와 같이 작성하도록 한다.

```

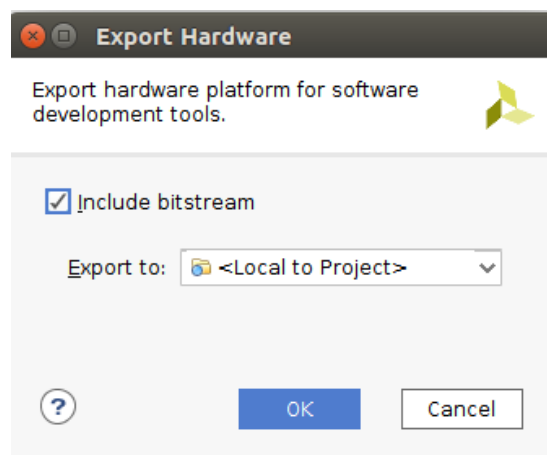
Package x Device x port.xdc x
1 set_property PACKAGE_PIN V15 [get_ports PWM0]
2 set_property PACKAGE_PIN T11 [get_ports pwm_sig]
3 set_property IOSTANDARD LVC MOS33 [get_ports PWM0]
4 set_property IOSTANDARD LVC MOS33 [get_ports pwm_sig]
5

```

여기까지 완료되었으면 이제 하드웨어 설계는 완료되었다. 하드웨어 정보를 밖으로 빼기위해 Generate Bitstream 을 해주도록 한다.



비트스트림이 완료되면 file → Export → Export Hardware 를 클릭하여 아래와같이 Include 를 체크 후 export 해준다



이제 Vivado 에서 하드웨어를 설계하여 밖으로 빼주었다. Software 를 설계하여 리눅스 포팅만 해주면 완료다.

이제 리눅스 커널을 튜닝한다.

리눅스 커널을 틀어 아래와 같이 프로젝트 폴더로 이동한다.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present$
```

1.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present$ petalinux-create -t project -n software --template zynq
```

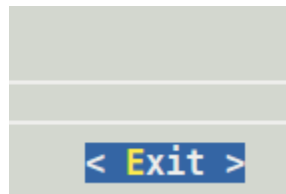
2.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present$ cd hardware/project_1.sdk/
```

3.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/hardware/project_1.sdk$ petalinux-config --get-hw-description -p ../../software/
```

4.



5.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/hardware/project_1.sdk$ cd ../../software/
```

6.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software$ petalinux-config -c u-boot
```

7.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software$ petalinux-build
```

8.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software$ petalinux-create -t apps -n device_driver --enable
```

9.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software$ cd components/apps/device_driver/
```

10.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software/components/apps/device_driver$ vi device_driver.c
```

디바이스 드라이버는 아래와 같이 작성한다. 여기서 duty 는 pwm 모듈의 출력듀티를 설정한다. 그러므로 하고싶은 듀티를 마음대로 바꿔도 무관하다.

device_driver.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>

#define PWM_MAP_SIZE      0x10000
#define ECAP_MAP_SIZE    0x10000

#define PWM_DATA_OFFSET   0x0
#define PWM_TRI_OFFSET    0x4

#define ECAP_PERIOD_OFFSET 0x0
#define ECAP_DUTY_OFFSET  0x1

int main(int argc, char *argv[])
{
    int fd1, fd2, i, duty = 100000;
    char *uiod1;
    char *uiod2;
    void *ptr1;
    void *ptr2;

    printf("eCAP UIO Test\r\n");
    printf("Loopback Test!\r\n");

    uiod1 = argv[1];
    uiod2 = argv[2];

    fd1 = open(uiod1, O_RDWR);
    if(fd1 < 1)
    {
        perror(argv[0]);
        printf("Invalid UIO Device File: %s\n", uiod1);
        return -1;
    }
    printf("%s open success!\r\n", uiod1);

    fd2 = open(uiod2, O_RDWR);
    if(fd2 < 1)
    {
        perror(argv[0]);
        printf("Invalid UIO Device File: %s\n", uiod2);
        return -1;
    }
    printf("%s open success!\r\n", uiod2);

    ptr1 = mmap(NULL, PWM_MAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd1, 0);
    ptr2 = mmap(NULL, ECAP_MAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd2, 0);

    printf("ptr1 ptr2 memory allocation success!\r\n");
    printf("ptr1 : 0x%x\r\n", ptr1);
    printf("ptr2 : 0x%x\r\n", ptr2);

    /*Infinite Loop*/
    while(1)
    {
```

```

*((unsigned *)(ptr1 + PWM_TRI_OFFSET)) = 0;
*((unsigned *)(ptr1 + PWM_DATA_OFFSET)) = duty;

printf("pwm : %d, period = %d, duty = %d\r\n",
      duty,
      *((unsigned int*)ptr2 + ECAP_PERIOD_OFFSET),
      *((unsigned int*)ptr2 + ECAP_DUTY_OFFSET)
      );

duty += 100;
if(duty > 200000)
    duty = 100000;
for(i=0;i<300000;i++)
    ;
}
return 0;
}

```

11.

```

siyun@siyun-CR62-6M:~/pwm_ecap_present/software/components/apps/device_driver$ cd ../../../../

```

12.

```

siyun@siyun-CR62-6M:~/pwm_ecap_present/software$ petalinux-config -c rootfs

```

13.

< Exit >

14.

```

siyun@siyun-CR62-6M:~/pwm_ecap_present/software$ petalinux-config -c kernel

```

15.

< Exit >

16.

```

siyun@siyun-CR62-6M:~/pwm_ecap_present/software$ cd ../hardware/project_1.sdk/

```

17.

```

siyun@siyun-CR62-6M:~/pwm_ecap_present/hardware/project_1.sdk$ petalinux-config --get-hw-description -p ../../software/

```

18.

```

siyun@siyun-CR62-6M:~/pwm_ecap_present/hardware/project_1.sdk$ cd ../../software/

```

19.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software$ petalinux-config
INFO: Checking component...
INFO: Config linux
```

20.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software$ petalinux-config -c kernel
```

21.

< Exit >

22.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software$ petalinux-config -c rootfs
```

23.

< Exit >

24.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software$ cd subsystems/linux/configs/device-tree/
```

25.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software/subsystems/linux/configs/device-tree$ vi system-top.dts
```

26.

system-top.dts 는 아래와 같이 작성하도록 한다.

```
/dts-v1/;
/include/ "system-conf.dtsi"
/ {
};
&clk {
    ps-clk-frequency = <50000000>;
};
&flash0 {
    compatible = "s25fl128s1";
};
&usb0 {
    dr_mode = "otg";
};
&gem0 {
    phy-handle = <&phy0>;
    mdio {
        #address-cells = <1>; #size-cells = <0>;
        phy0: phy@1 {
            compatible = "realtek,RTL8211E";
            device_type = "ethernet-phy";
            reg = <1>;
        };
    };
};
```

```
};
};
&My_PWM_Core_0 {
    compatible = "generic-uiso";
};
&My_ECAP_Core_0 {
    compatible = "generic-uiso";
};
```

27.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software/subsystems/linux/configs/device-tree$ cd ../../../../images/linux/
```

28.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software/images/linux$ petalinux-build
```

29.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software/images/linux$ source ~/xilinx/Vivado/2017.1/settings64.sh
siyun@siyun-CR62-6M:~/pwm_ecap_present/software/images/linux$ source ~/xilinx/SDK/2017.1/settings64.sh
siyun@siyun-CR62-6M:~/pwm_ecap_present/software/images/linux$ petalinux-package --boot --fsbl zynq_fsbl.elf --fpga ../../hardware/project_1.runs/impl_1/design_1_wrapper.bit --u-boot
--force
```

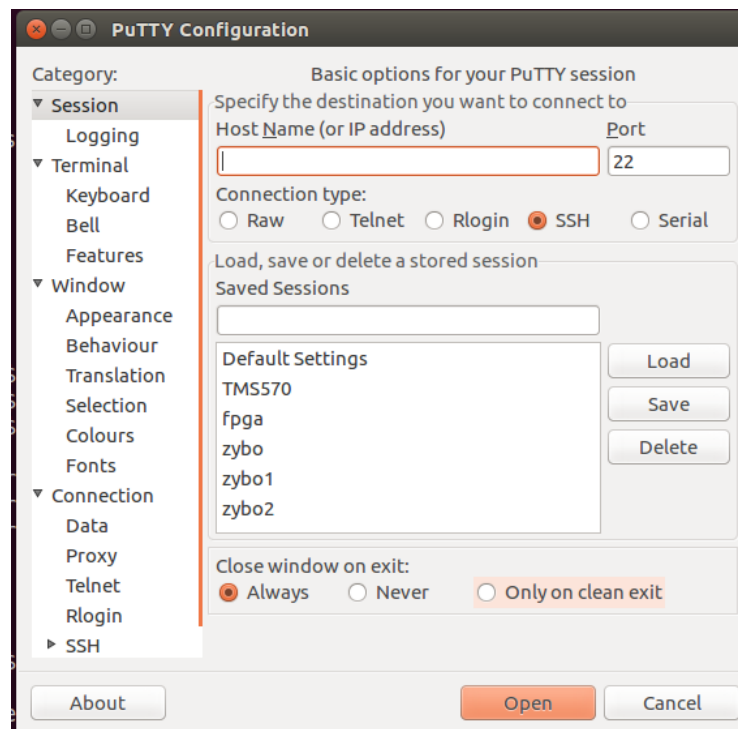
바이너리 이즈 레디가 뜨면 성공이다.

부트 이미지를 SD 카드 BOOT 에 넣어주도록 한다.

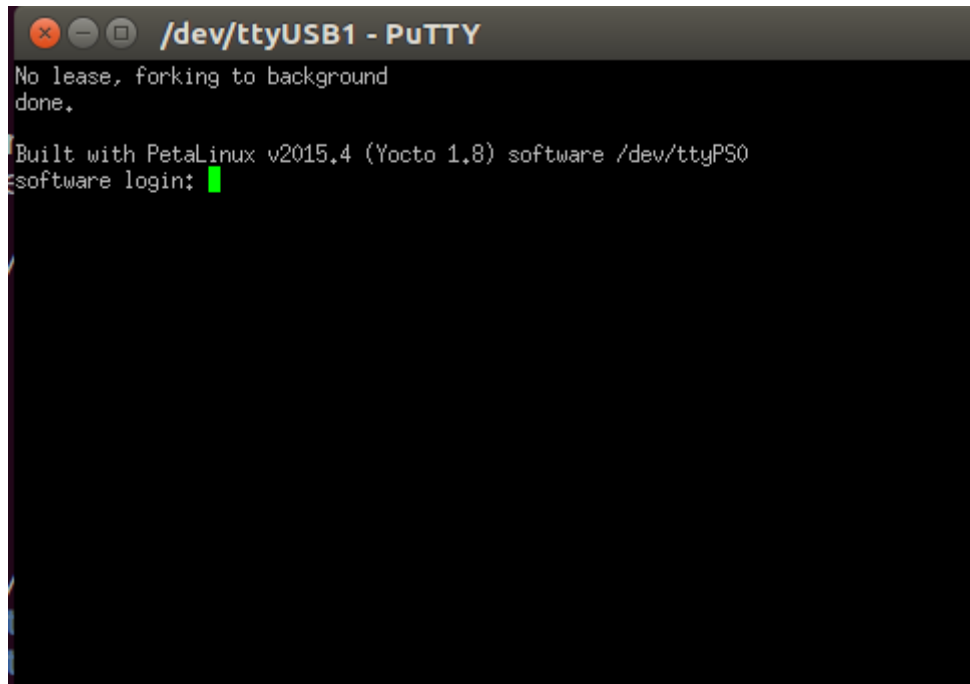
30.

```
siyun@siyun-CR62-6M:~/pwm_ecap_present/software/images/linux$ sudo chmod 666 /dev/ttyUSB1
```

31. puty 를 실행하여 Serial 115200 /dev/ttyUSB1 입력후 오픈해준다



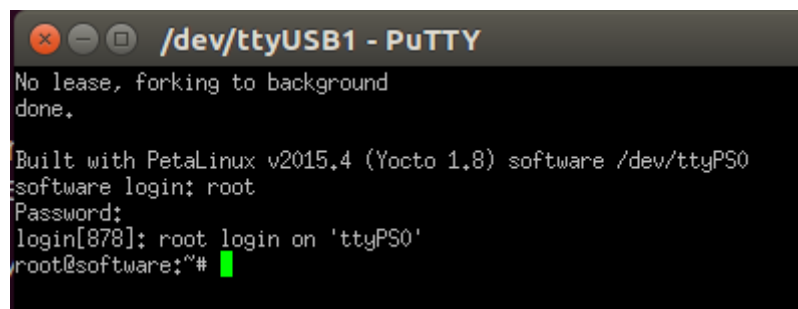
32.



```
/dev/ttyUSB1 - PuTTY
No lease, forking to background
done.

Built with PetaLinux v2015.4 (Yocto 1.8) software /dev/ttyPS0
software login: █
```

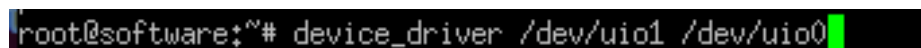
id = root
pw = root



```
/dev/ttyUSB1 - PuTTY
No lease, forking to background
done.

Built with PetaLinux v2015.4 (Yocto 1.8) software /dev/ttyPS0
software login: root
Password:
login[878]: root login on 'ttyPS0'
root@software:~# █
```

로그인을 완료하면 device_driver /dev/uio1 /dev/uio0 을 입력하여 실행시킨다.
uio1 은 pwm 모듈이고 uio0 은 ecap 모듈이다.



```
root@software:~# device_driver /dev/uio1 /dev/uio0 █
```

실행시키면 아래와 같이 동작한다.

클럭 주파수가 100 MHz 이고 PWM 출력은 50Hz 이므로
주기는 2000000 이 나와야 하고 듀티는 변해야한다..

[illegible]

핀 설정은

JC1 과 JC3 을 연결시켜주면 된다.

JC1 = V15 로 PWM 신호가 나오고

JC3 = T11 로 PWM 신호를 캡쳐한다.

