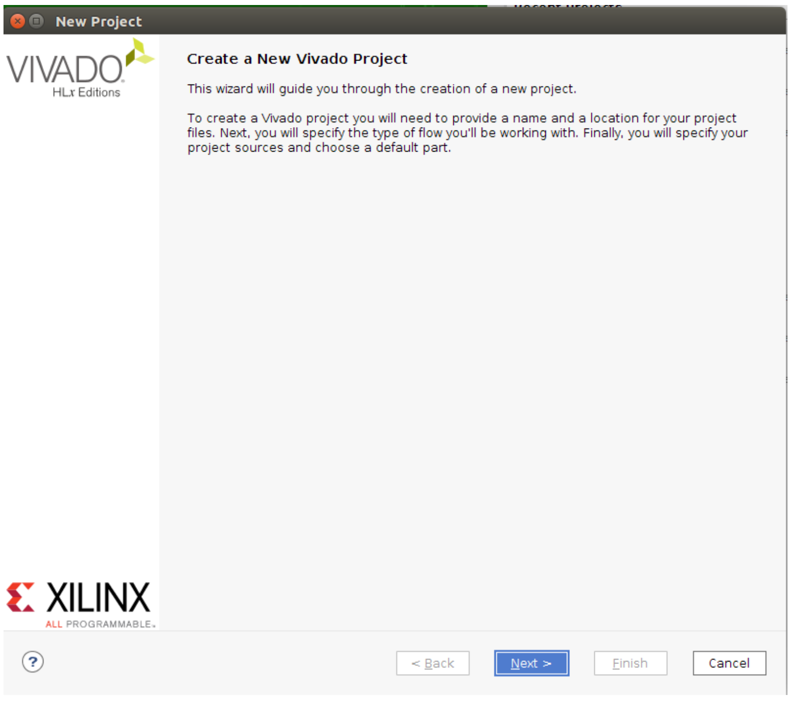
Xilinx Zynq FPGA, TI DSP, MCU 기반의
프로그래밍 및 회로 설계 전문가 과정

FPGA STEP MOTOR CONTROL

1. Vivado New Project

Next



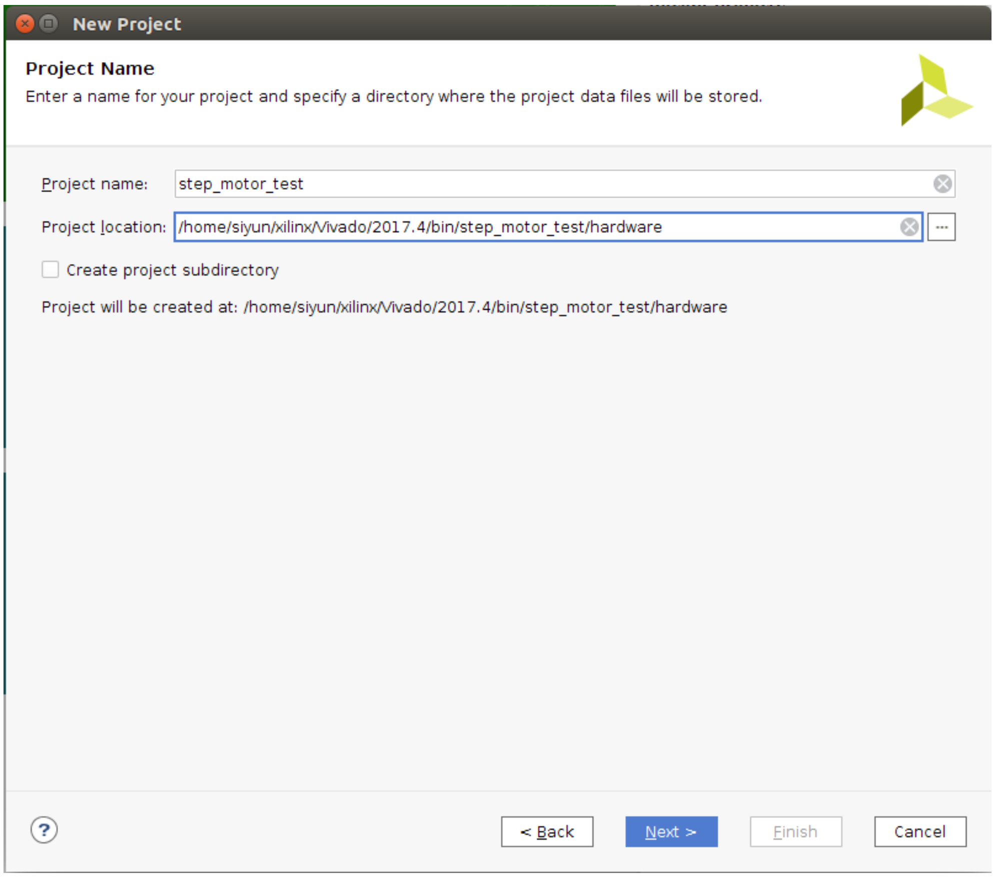
New Project

Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

Project Name : step_motor_test
Project Location : /home/siyun/vivado_workspace/step_motor_test/hardware/



New Project

Project Name

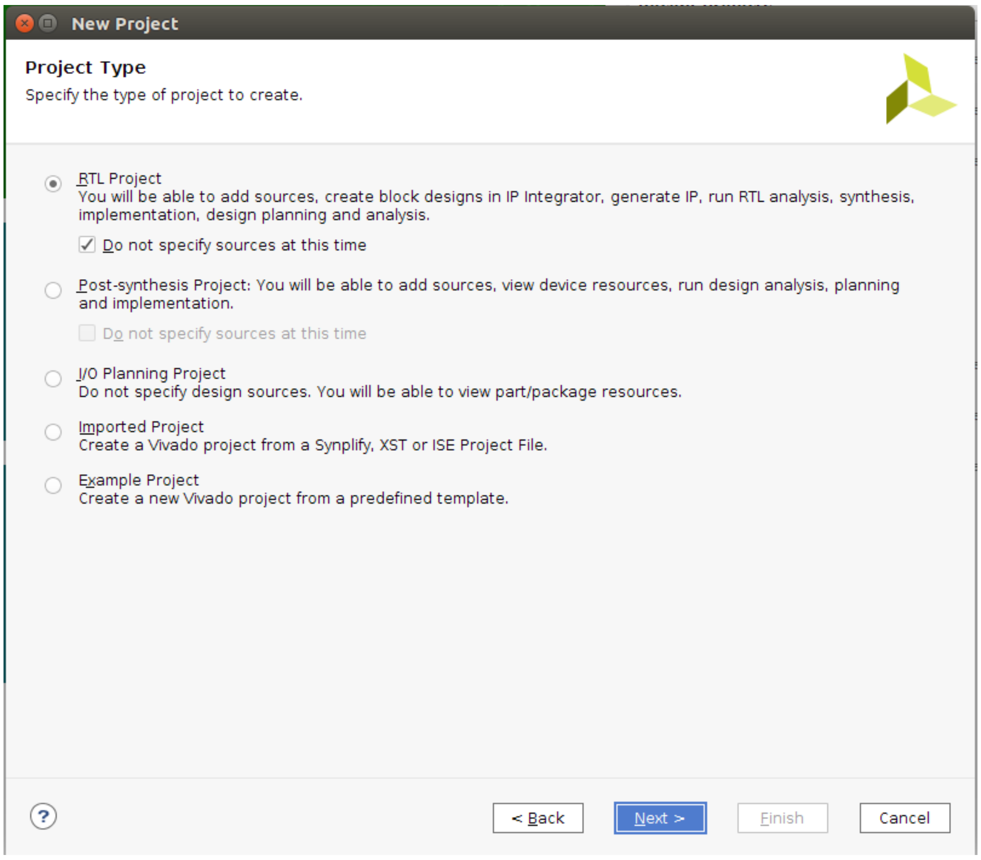
Enter a name for your project and specify a directory where the project data files will be stored.

Project name: step_motor_test

Project location: /home/siyun/xilinx/Vivado/2017.4/bin/step_motor_test/hardware

☐ Create project subdirectory

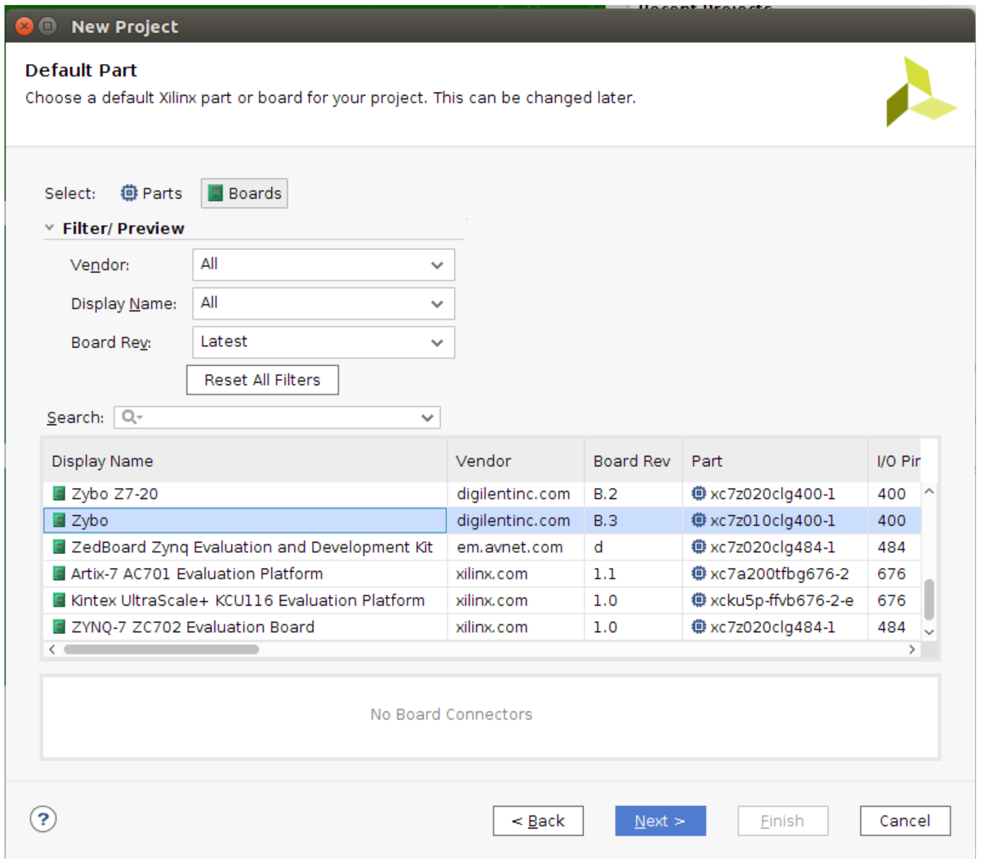
Project will be created at: /home/siyun/xilinx/Vivado/2017.4/bin/step_motor_test/hardware


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.

Select: Parts [✓ Boards]

Filter/ Preview

Vendor: All

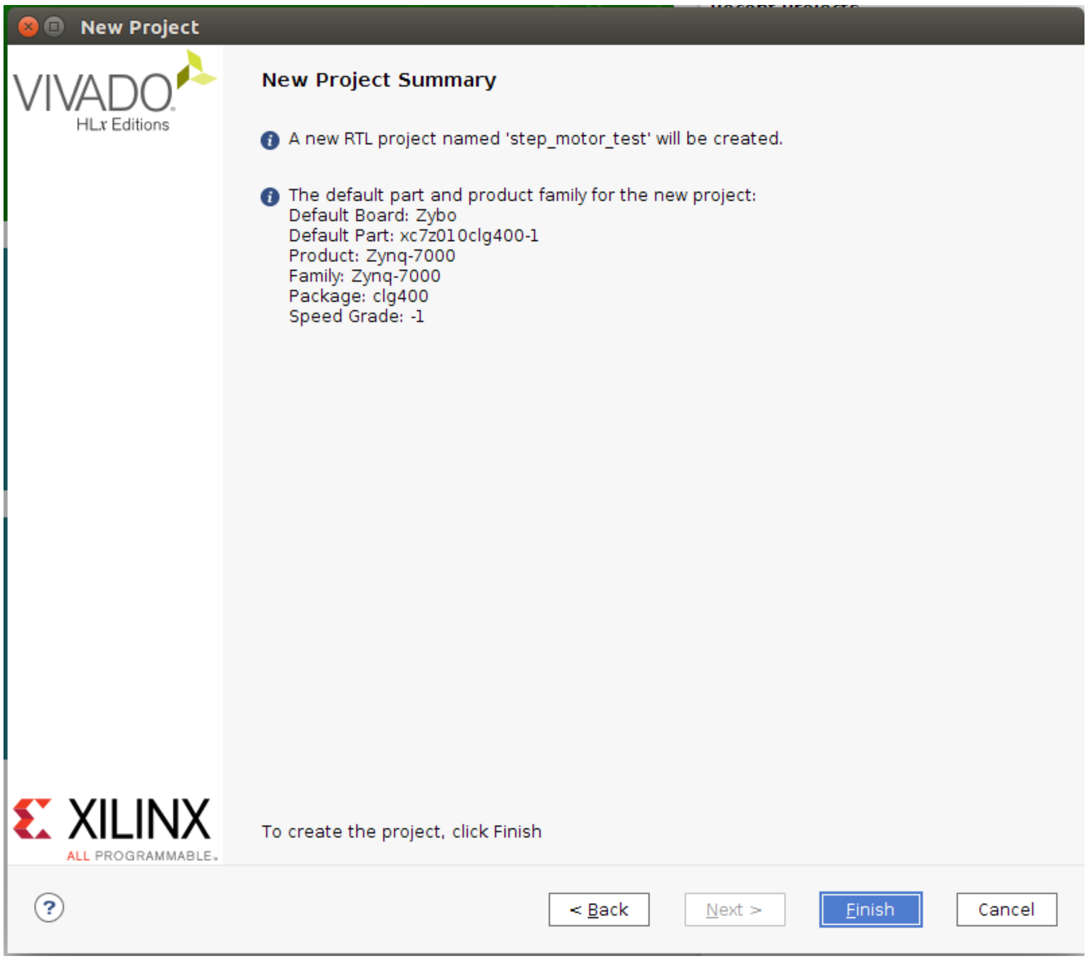
Display Name: All

Board Rev: Latest

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	xc7a200tfbg676-2	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

finish



New Project

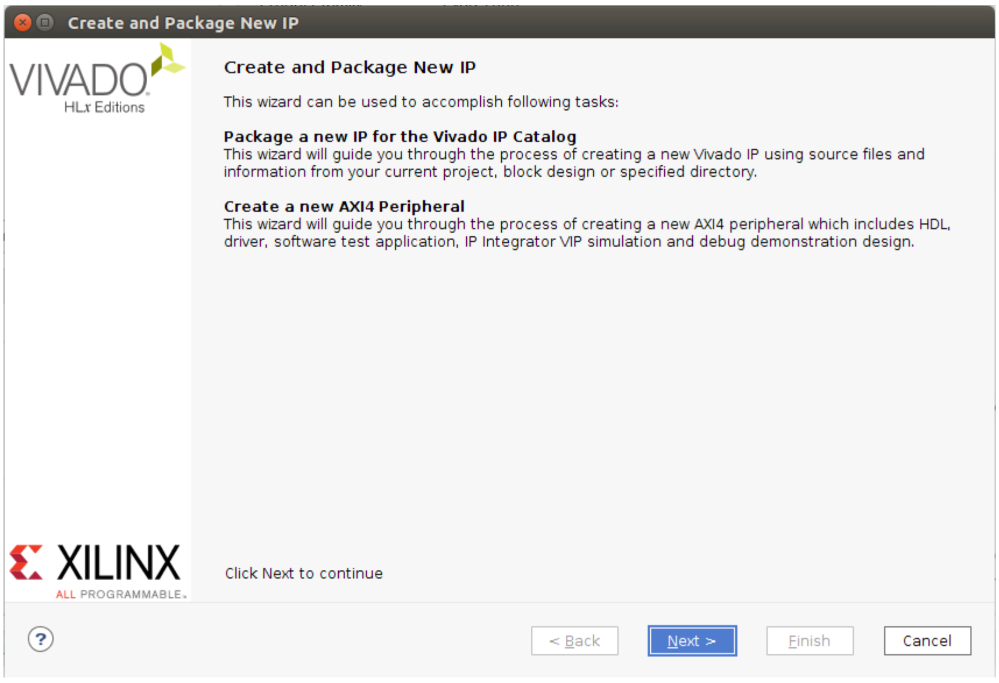
New Project Summary

A new RTL project named 'step_motor_test' 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

To create the project, click Finish

Tools → create and package New IP



Create and Package New IP

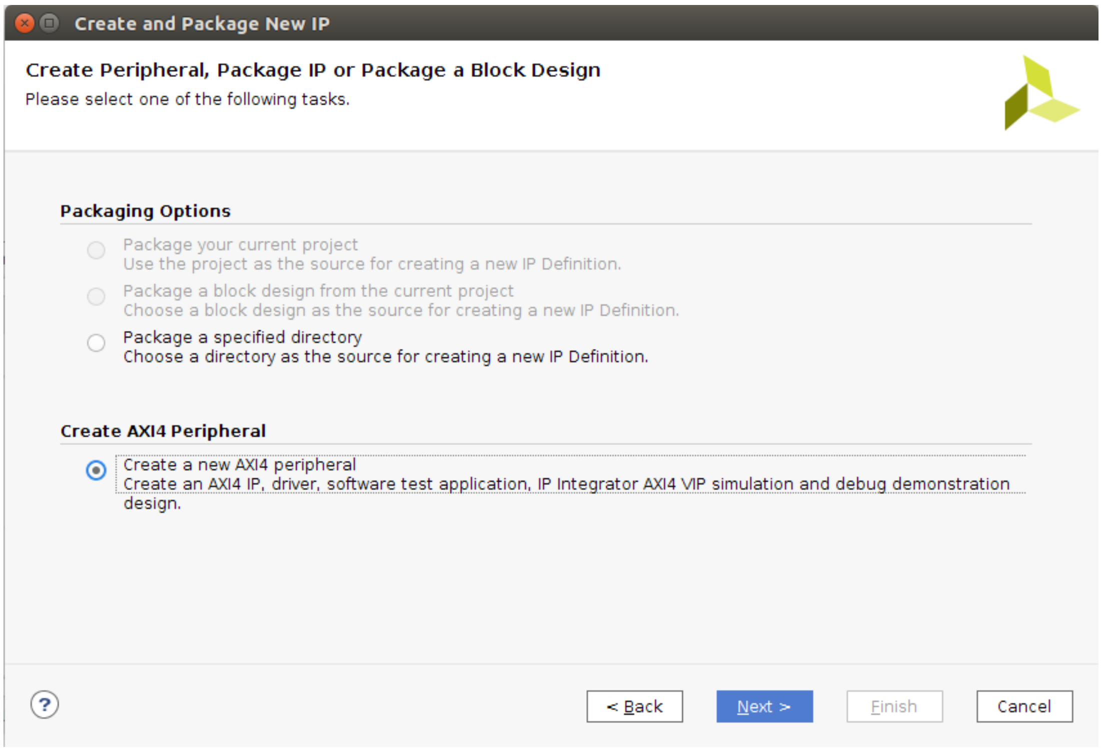
Create and Package New IP

This wizard can be used to accomplish following tasks:

Package a new IP for the Vivado IP Catalog
This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project, block design or specified directory.

Create a new AXI4 Peripheral
This wizard will guide you through the process of creating a new AXI4 peripheral which includes HDL, driver, software test application, IP Integrator VIP simulation and debug demonstration design.

Click Next to continue


Create and Package New IP

Create Peripheral, Package IP or Package a Block Design

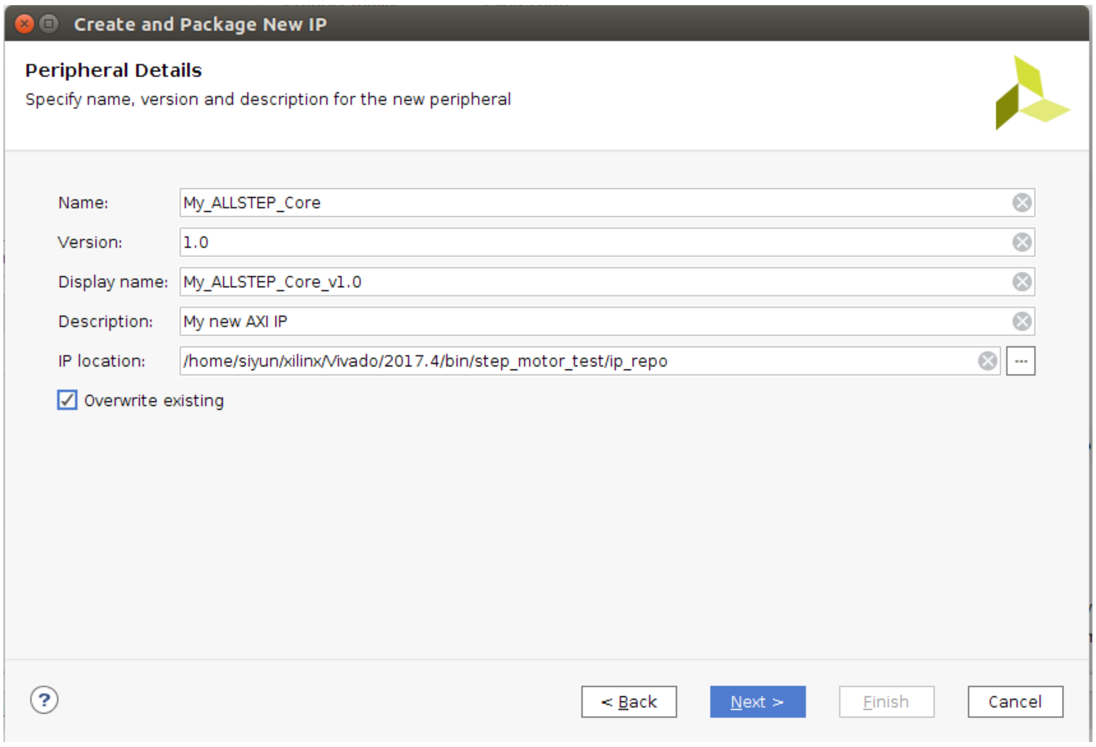
Please select one of the following tasks.

Packaging Options

- ○ Package your current project
  Use the project as the source for creating a new IP Definition.
- ○ Package a block design from the current project
  Choose a block design as the source for creating a new IP Definition.
- ○ Package a specified directory
  Choose a directory as the source for creating a new IP Definition.

Create AXI4 Peripheral

- [✓ ●] Create a new AXI4 peripheral
  Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.

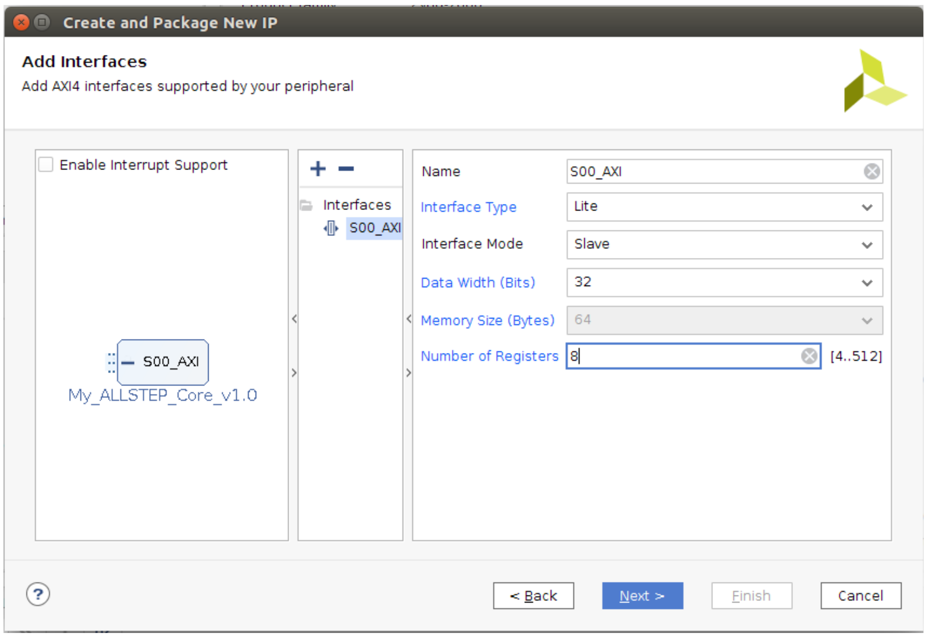

Create and Package New IP

Peripheral Details

Specify name, version and description for the new peripheral

Name:	My_ALLSTEP_Core
Version:	1.0
Display name:	My_ALLSTEP_Core_v1.0
Description:	My new AXI IP
IP location:	/home/siyun/xilinx/Vivado/2017.4/bin/step_motor_test/ip_repo

[✓] Overwrite existing


Create and Package New IP

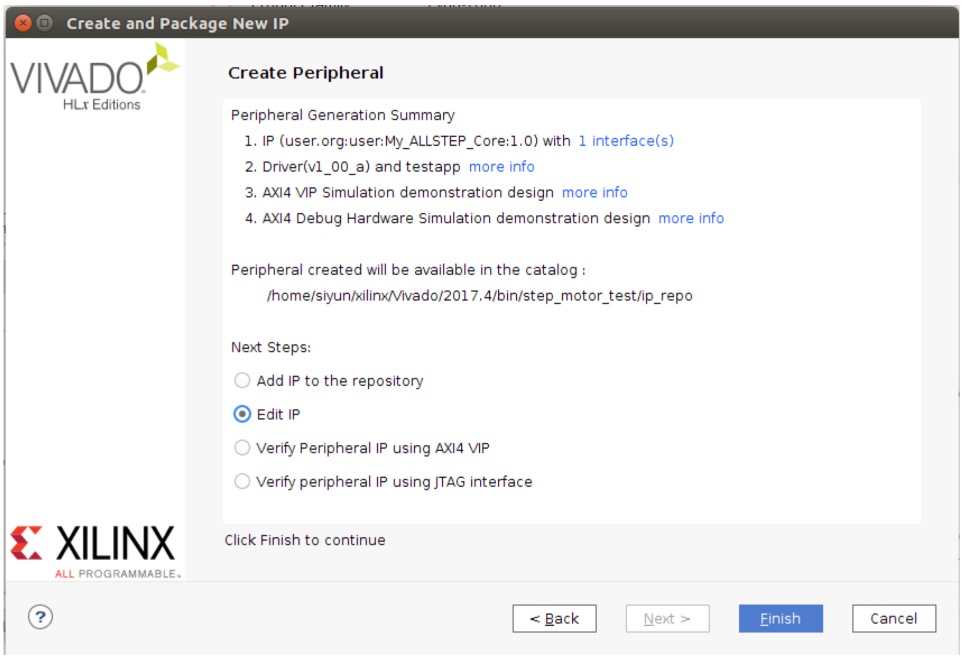
Add Interfaces

Add AXI4 interfaces supported by your peripheral

Enable Interrupt Support

Interfaces
S00_AXI

Name: S00_AXI
Interface Type: Lite
Interface Mode: Slave
Data Width (Bits): 32
Memory Size (Bytes): 64
Number of Registers: 8 [4..512]


Create and Package New IP

Create Peripheral

Peripheral Generation Summary
1. IP (user.org:user:My_ALLSTEP_Core:1.0) with 1 interface(s)
2. Driver(v1_00_a) and testapp more info
3. AXI4 VIP Simulation demonstration design more info
4. AXI4 Debug Hardware Simulation demonstration design more info

Peripheral created will be available in the catalog :
/home/siyun/xilinx/Vivado/2017.4/bin/step_motor_test/ip_repo

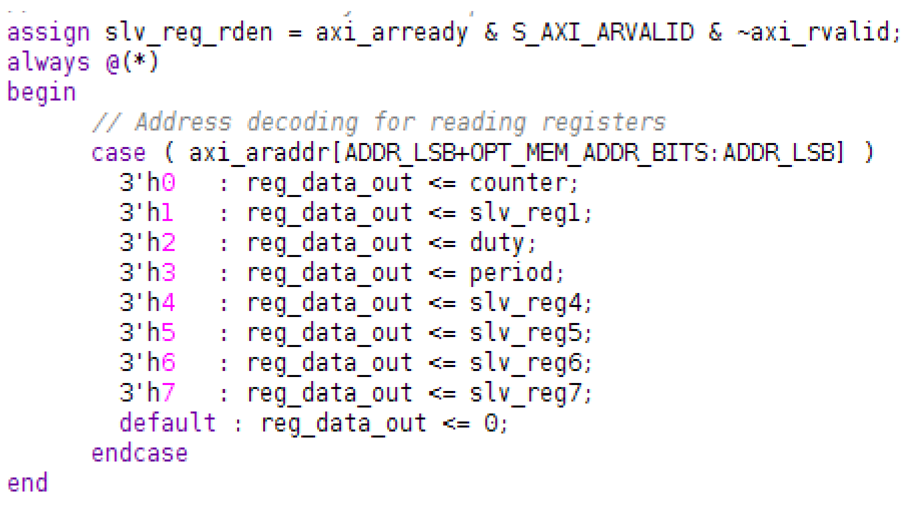
Next Steps:
- Add IP to the repository
- [✓ Edit IP]
- Verify Peripheral IP using AXI4 VIP
- Verify peripheral IP using JTAG interface

Click Finish to continue

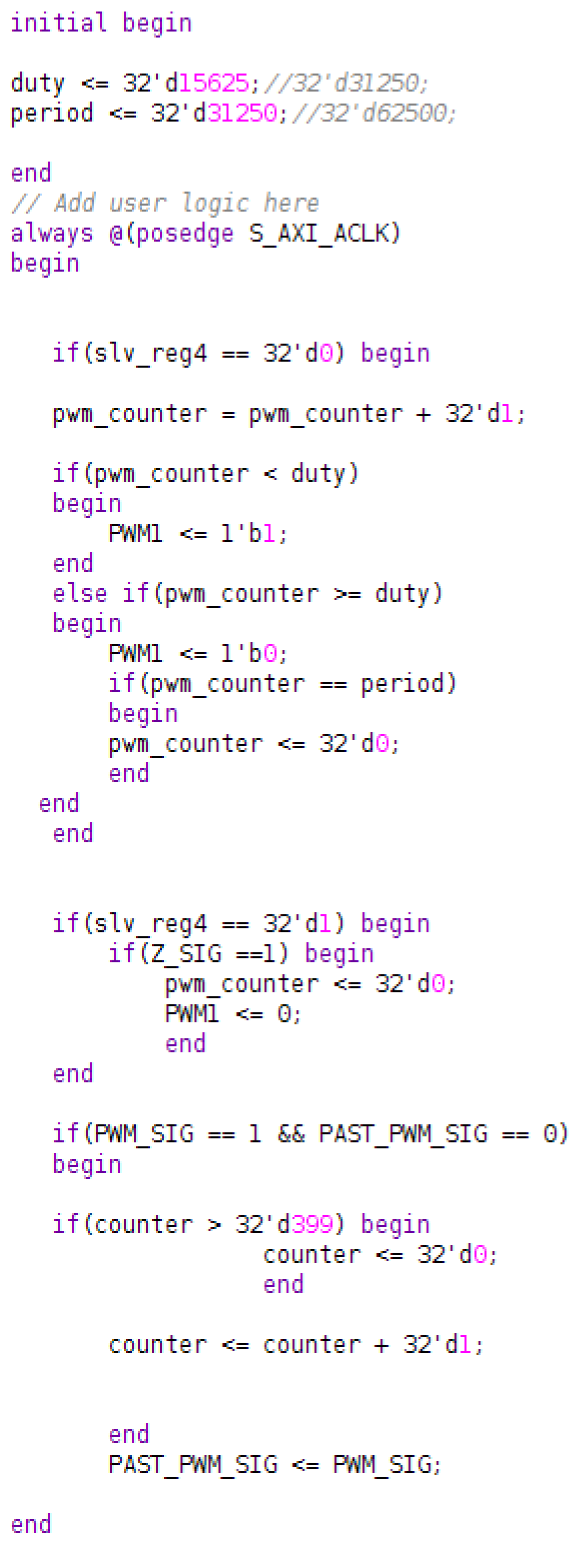
finish

먼저 sub module 부터 수정

```
        // Users to add ports here
        input wire PWM_SIG,
        input wire Z_SIG,
        output reg PWM1,
        // User ports ends
    // user reg
      reg PAST_Z_SIG;
       reg PAST_PWM_SIG;
       reg [31:0] counter;
       reg [31:0] O_counter;
       reg [31:0] PAST_PWM_DUTY;
       reg [31:0] PAST_PWM_PERIOD;
       reg [31:0] pwm_counter;
       reg [31:0] duty;
       reg [31:0] period;
```

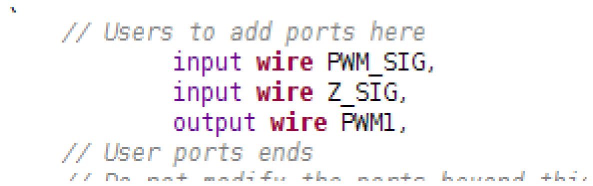


```
assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
always @(*)
begin
      // Address decoding for reading registers
      case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
        3'h0   : reg_data_out <= counter;
        3'h1   : reg_data_out <= slv_reg1;
        3'h2   : reg_data_out <= duty;
        3'h3   : reg_data_out <= period;
        3'h4   : reg_data_out <= slv_reg4;
        3'h5   : reg_data_out <= slv_reg5;
        3'h6   : reg_data_out <= slv_reg6;
        3'h7   : reg_data_out <= slv_reg7;
        default : reg_data_out <= 0;
      endcase
end
```

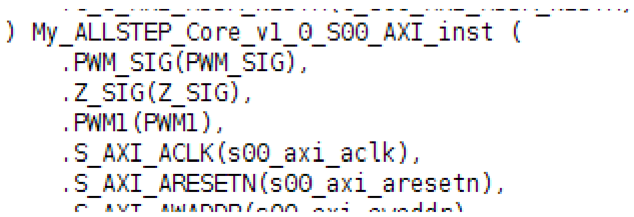


```
initial begin

duty <= 32'd15625;//32'd31250;
period <= 32'd31250;//32'd62500;

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


   if(slv_reg4 == 32'd0) begin

   pwm_counter = pwm_counter + 32'd1;

   if(pwm_counter < duty)
   begin
       PWM1 <= 1'b1;
   end
   else if(pwm_counter >= duty)
   begin
       PWM1 <= 1'b0;
       if(pwm_counter == period)
       begin
       pwm_counter <= 32'd0;
       end
  end
   end


   if(slv_reg4 == 32'd1) begin
       if(Z_SIG ==1) begin
           pwm_counter <= 32'd0;
           PWM1 <= 0;
           end
   end

   if(PWM_SIG == 1 && PAST_PWM_SIG == 0)
   begin

   if(counter > 32'd399) begin
                   counter <= 32'd0;
                   end

       counter <= counter + 32'd1;


       end
       PAST_PWM_SIG <= PWM_SIG;

end
```

top module

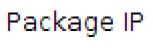


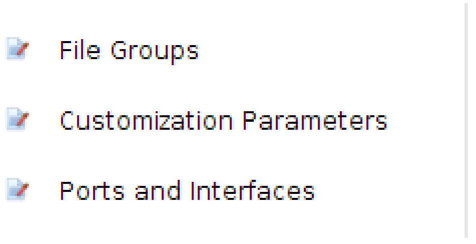
```
    // Users to add ports here
            input wire PWM_SIG,
            input wire Z_SIG,
            output wire PWM1,
    // User ports ends
    // Do not modify the ports beyond thi
```

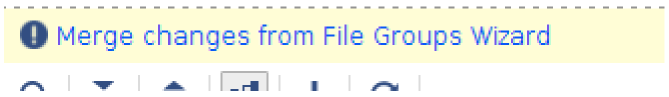


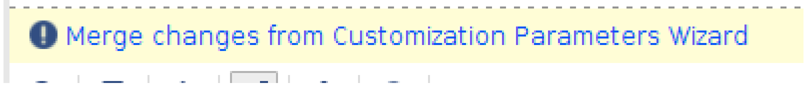
```
) My_ALLSTEP_Core_v1_0_S00_AXI_inst (
    .PWM_SIG(PWM_SIG),
    .Z_SIG(Z_SIG),
    .PWM1(PWM1),
    .S_AXI_ACLK(s00_axi_aclk),
    .S_AXI_ARESETN(s00_axi_aresetn),
```

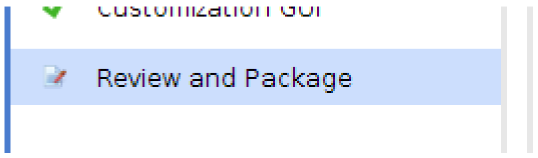
이후 package ip

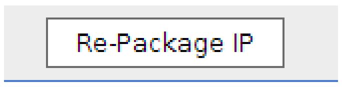




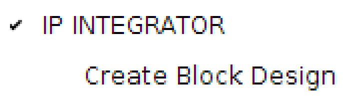


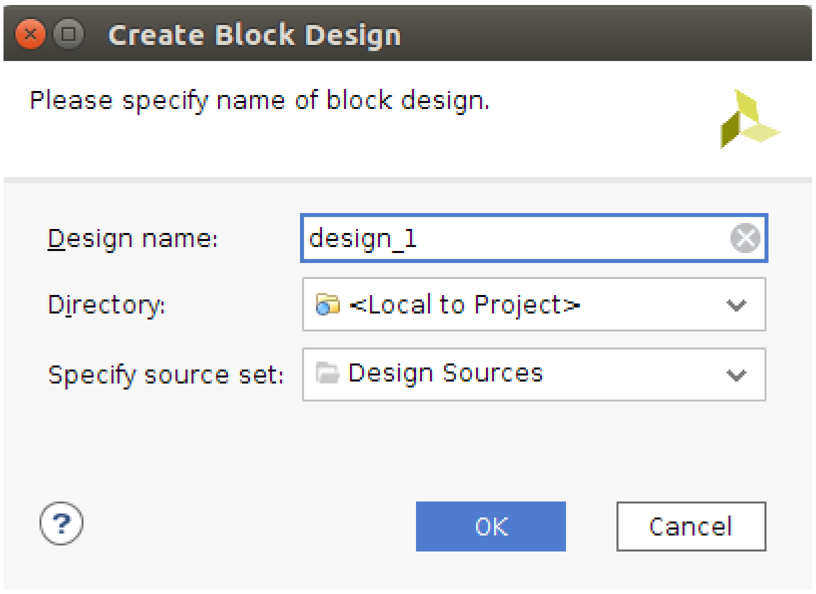






그 후 원래 프로젝트로 돌아와서
create block design



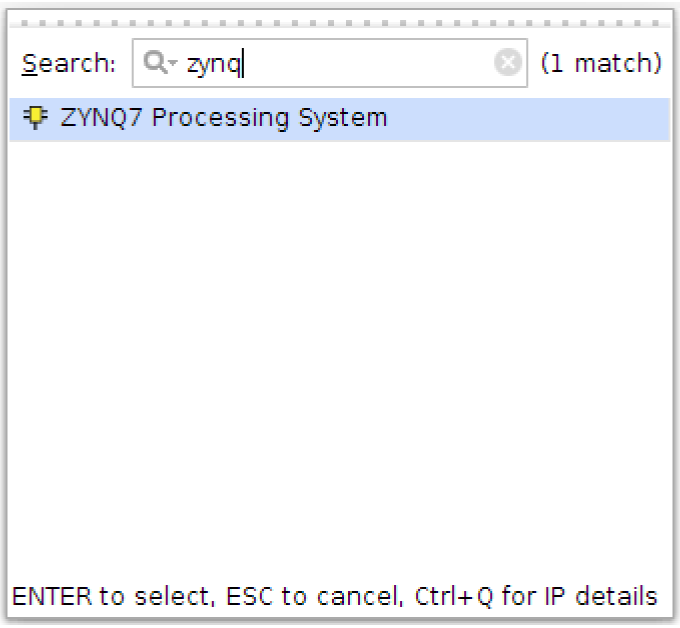

Create Block Design

Please specify name of block design.

Design name: design_1

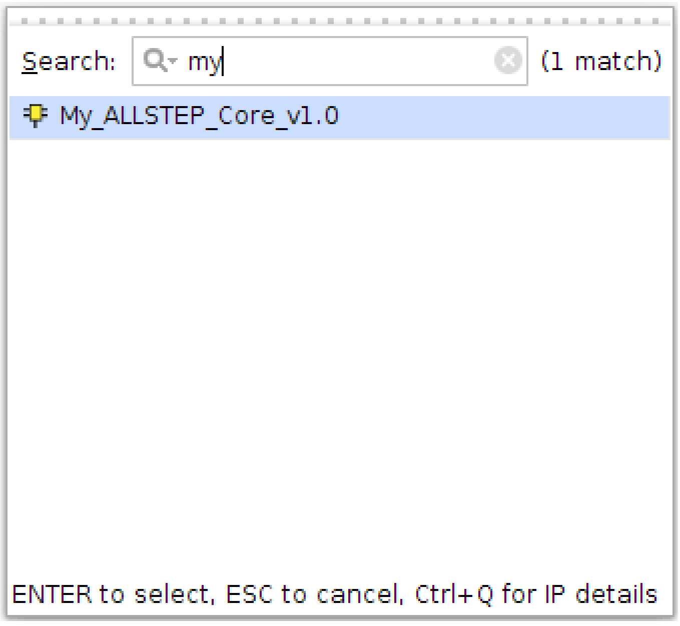
Directory: <Local to Project>

Specify source set: Design Sources


Search: zynq (1 match)

ZYNQ7 Processing System

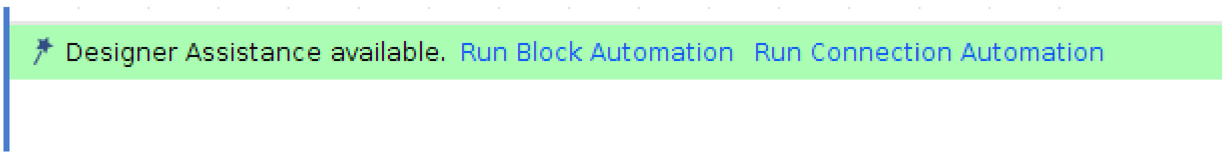
ENTER to select, ESC to cancel, Ctrl+Q for IP details


Search: my (1 match)

My_ALLSTEP_Core_v1.0

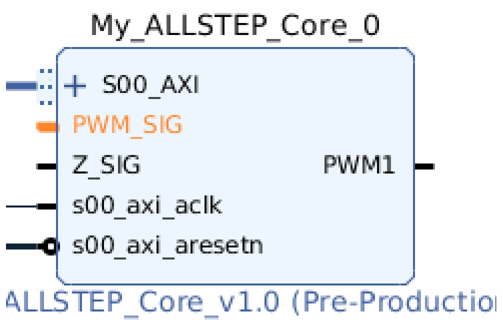
ENTER to select, ESC to cancel, Ctrl+Q for IP details

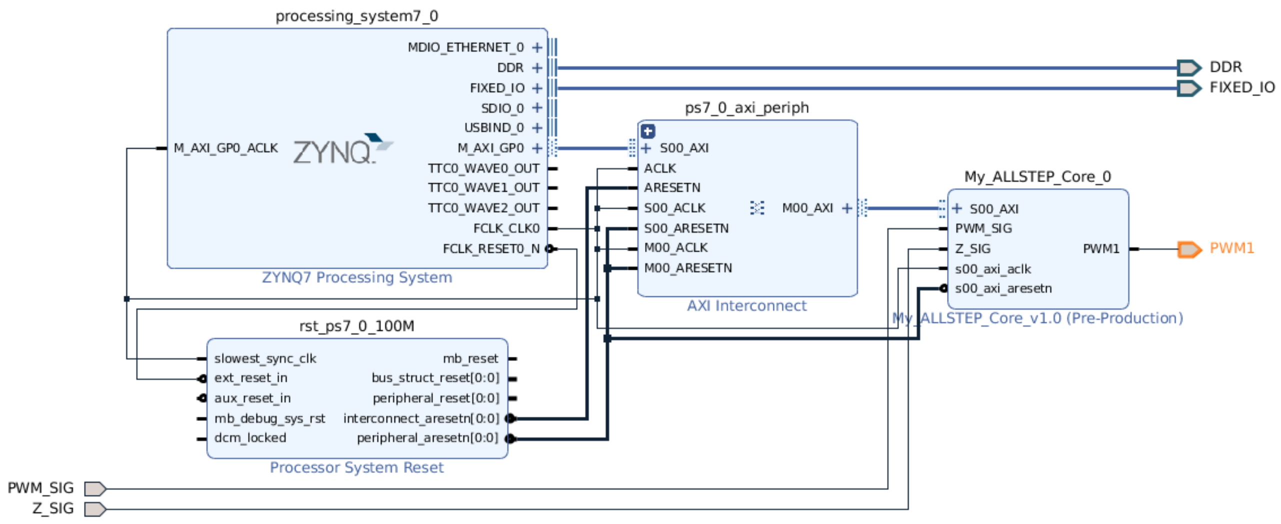
add ip


Designer Assistance available. Run Block Automation Run Connection Automation

run block automation 후
run connection automation

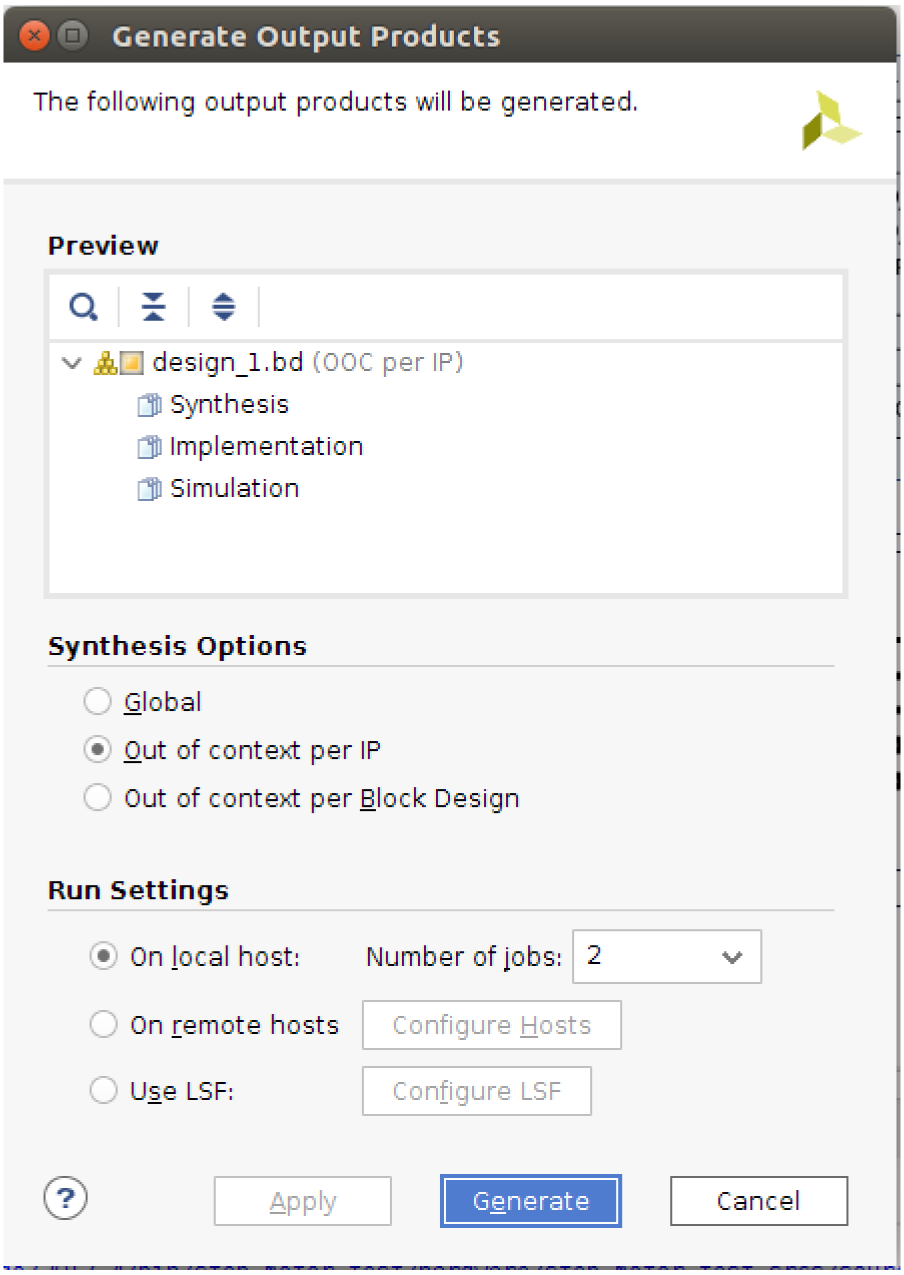
pin 하나 하나 선택하여 오른쪽 클릭 후 create port





회로 오류 검사

generate output product



Generate Output Products

The following output products will be generated.

Preview

design_1.bd (OOC per IP)
- Synthesis
- Implementation
- Simulation

Synthesis Options

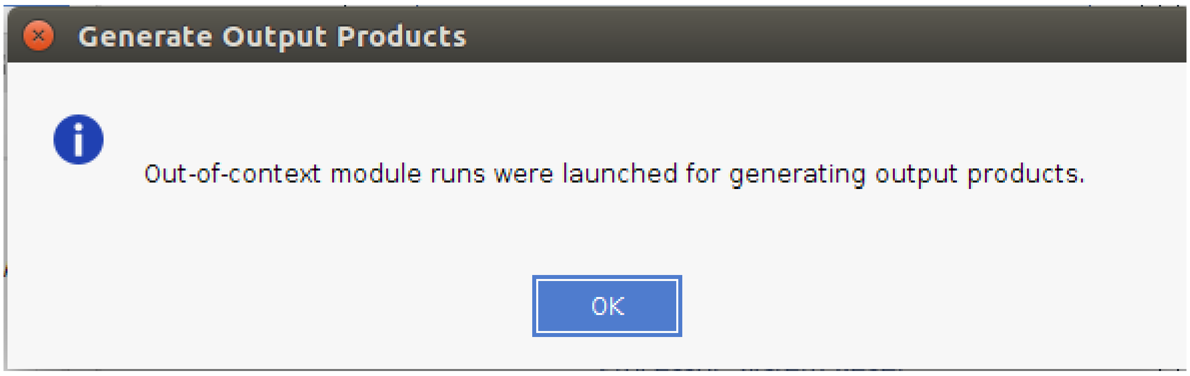
Global [✓ Out of context per IP] Out of context per Block Design

Run Settings

[✓ On local host:] Number of jobs: 2

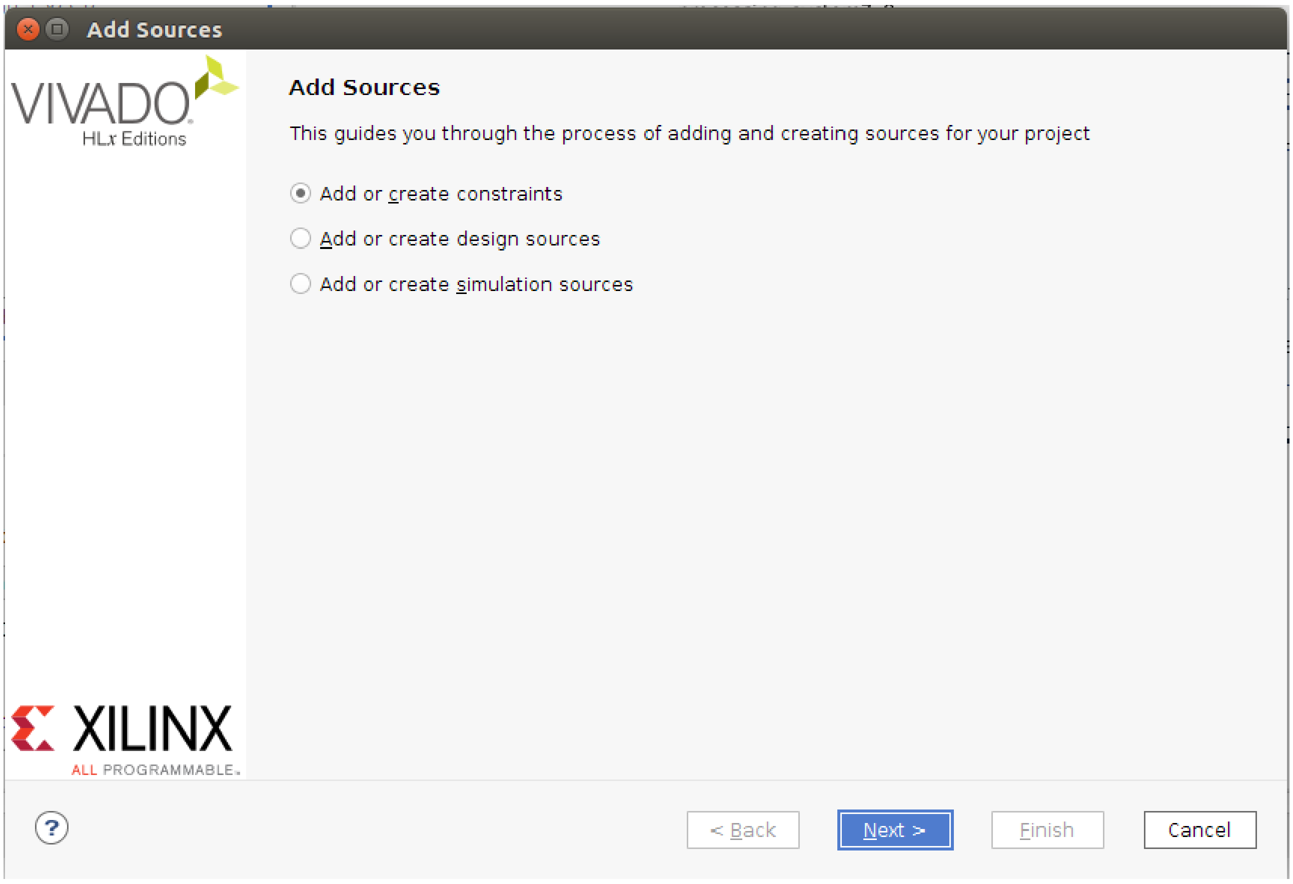
On remote hosts Configure Hosts

Use LSF: Configure LSF



Generate Output Products

Out-of-context module runs were launched for generating output products.

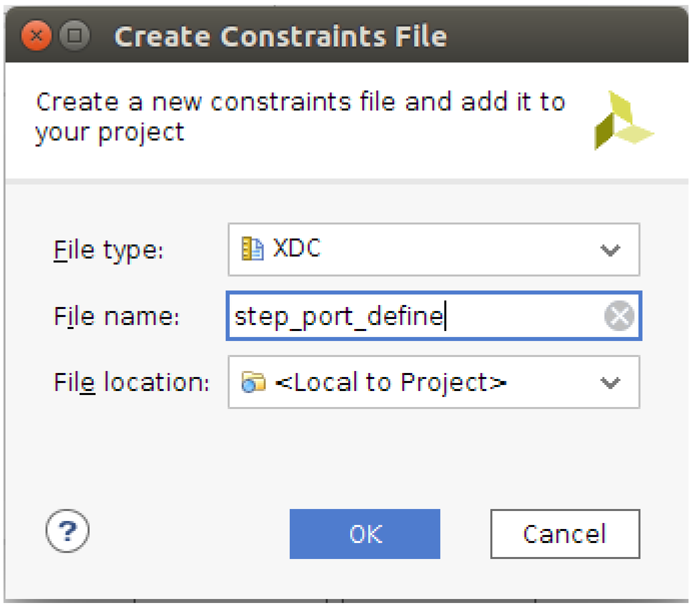

Add Sources

Add Sources

This guides you through the process of adding and creating sources for your project

● Add or create constraints
○ Add or create design sources
○ Add or create simulation sources

consraint add source

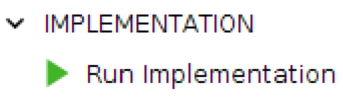

Create Constraints File

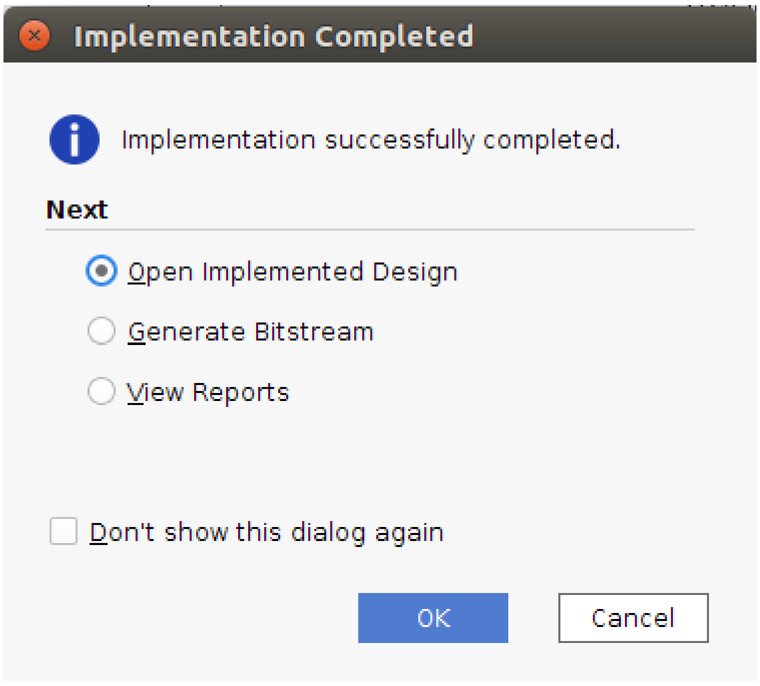
Create a new constraints file and add it to your project

File type: XDC

File name: step_port_define

File location: <Local to Project>



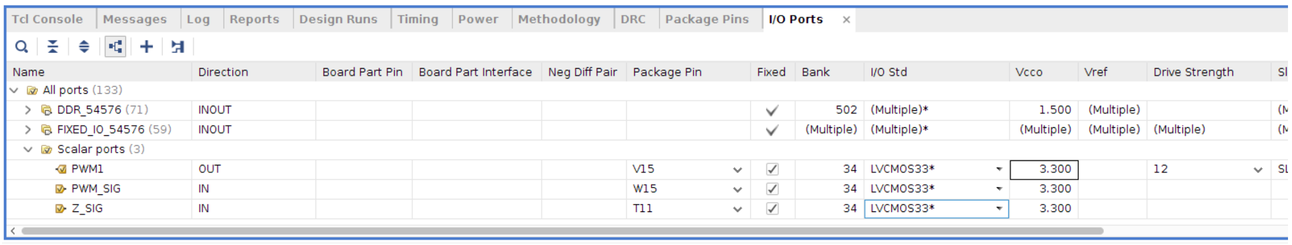

Implementation Completed

Implementation successfully completed.

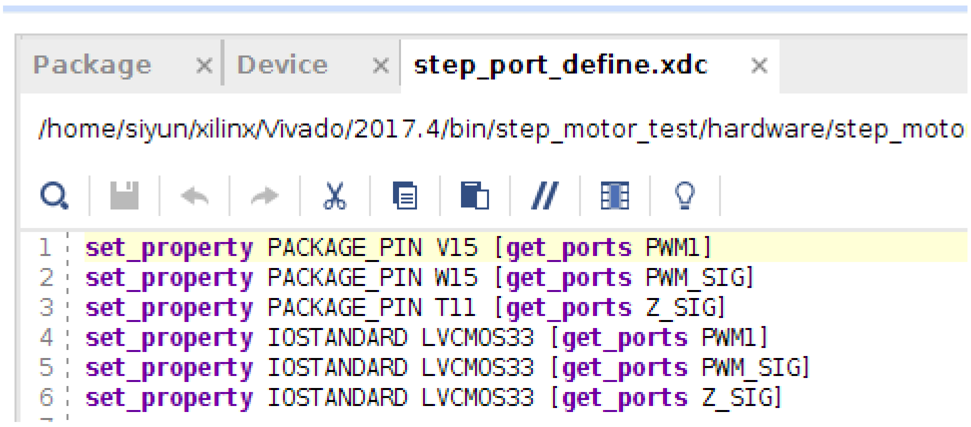
Next

[✓ Open Implemented Design]
Generate Bitstream
View Reports

☐ Don't show this dialog again

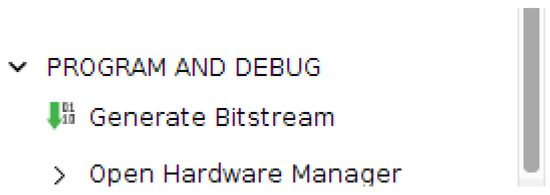


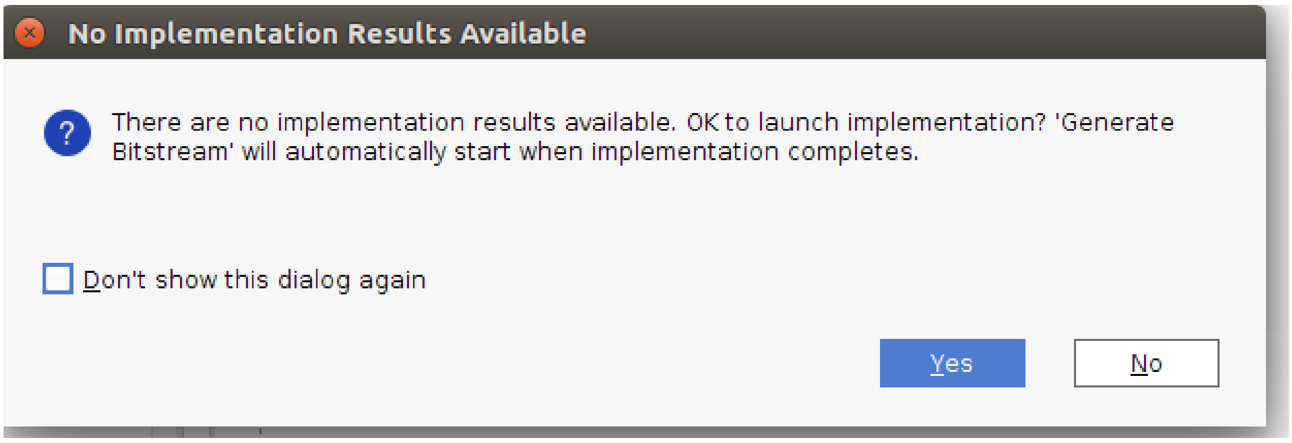
Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Sl
All ports (133)												
DDR_54576 (71)	INOUT					✓	502	(Multiple)*	1.500	(Multiple)		(M
FIXED_IO_54576 (59)	INOUT					✓	(Multiple)	(Multiple)*	(Multiple)	(Multiple)	(Multiple)	(M
Scalar ports (3)												
PWM1	OUT				V15	✓	34	LVCMOS33*	3.300		12	Sl
PWM_SIG	IN				W15	✓	34	LVCMOS33*	3.300			
Z_SIG	IN				T11	✓	34	LVCMOS33*	3.300			



/home/siyun/xilinx/Vivado/2017.4/bin/step_motor_test/hardware/step_moto

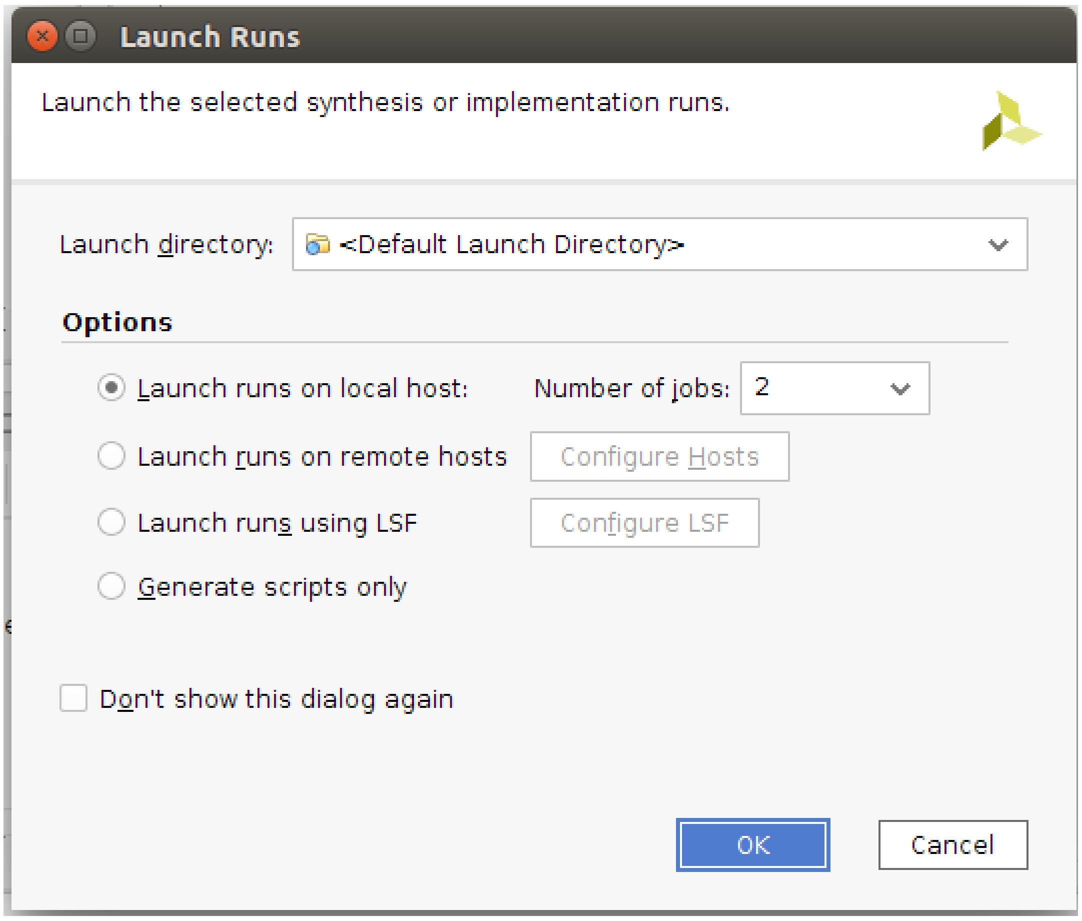
```
set_property PACKAGE_PIN V15 [get_ports PWM1]
set_property PACKAGE_PIN W15 [get_ports PWM_SIG]
set_property PACKAGE_PIN T11 [get_ports Z_SIG]
set_property IOSTANDARD LVCMOS33 [get_ports PWM1]
set_property IOSTANDARD LVCMOS33 [get_ports PWM_SIG]
set_property IOSTANDARD LVCMOS33 [get_ports Z_SIG]
```




No Implementation Results Available

There are no implementation results available. OK to launch implementation? 'Generate Bitstream' will automatically start when implementation completes.

☐ Don't show this dialog again


Launch Runs

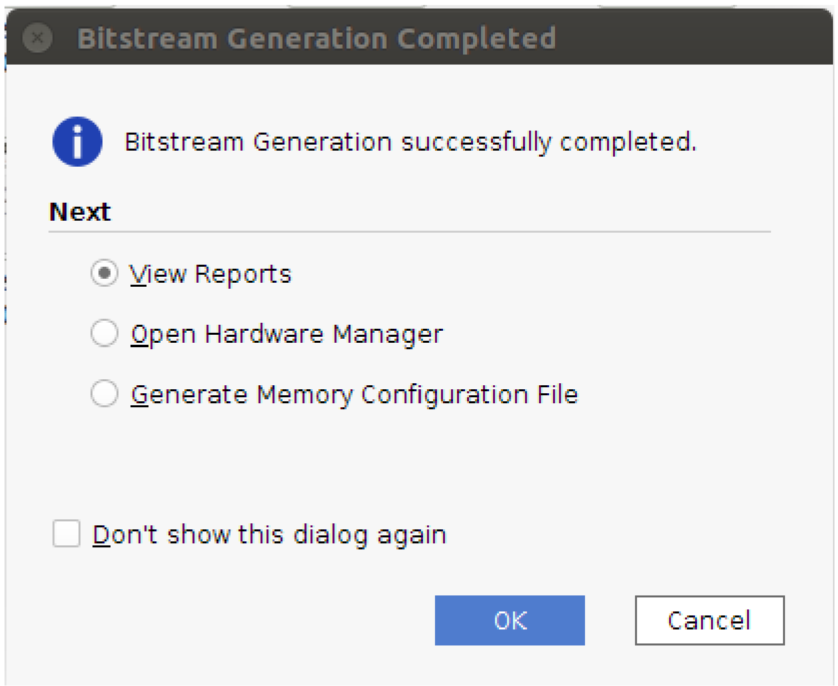
Launch the selected synthesis or implementation runs.

Launch directory: <Default Launch Directory>

Options

- [✓ Launch runs on local host:] Number of jobs: 2
- Launch runs on remote hosts
- Launch runs using LSF
- Generate scripts only

☐ Don't show this dialog again

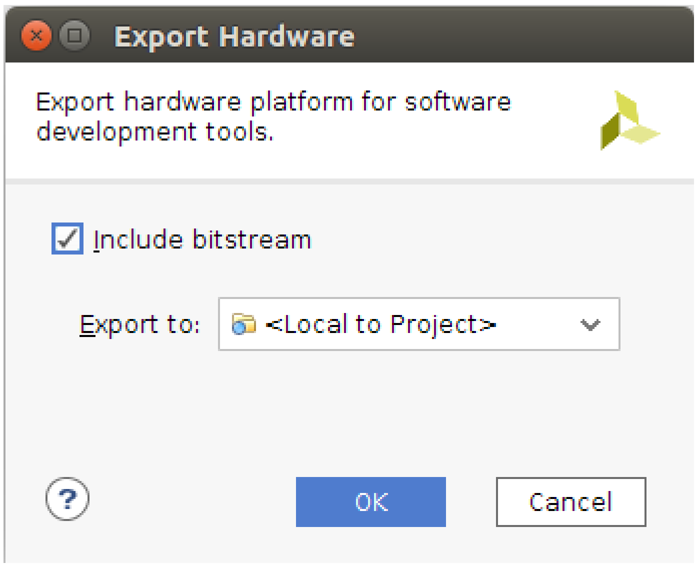

Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

- [✓ View Reports]
- Open Hardware Manager
- Generate Memory Configuration File

☐ Don't show this dialog again

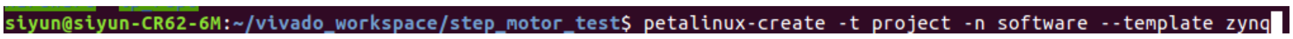


Export Hardware

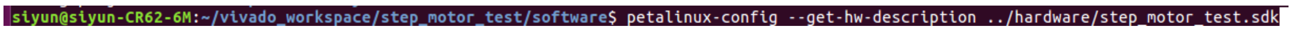
Export hardware platform for software development tools.

[✓] Include bitstream

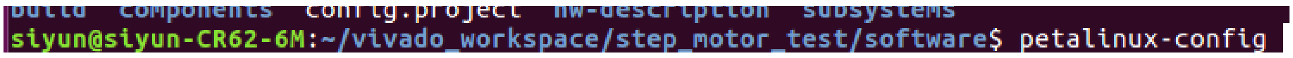
Export to: <Local to Project>



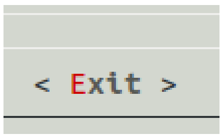
```
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test$ petalinux-create -t project -n software --template zynq
```

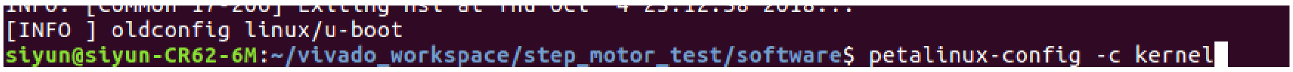


```
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test/software$ petalinux-config --get-hw-description ../hardware/step_motor_test.sdk
```

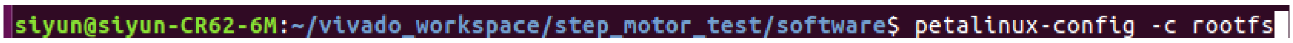


```
build   components   config.project   hw-description   subsystems
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test/software$ petalinux-config
```

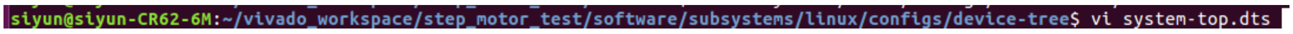




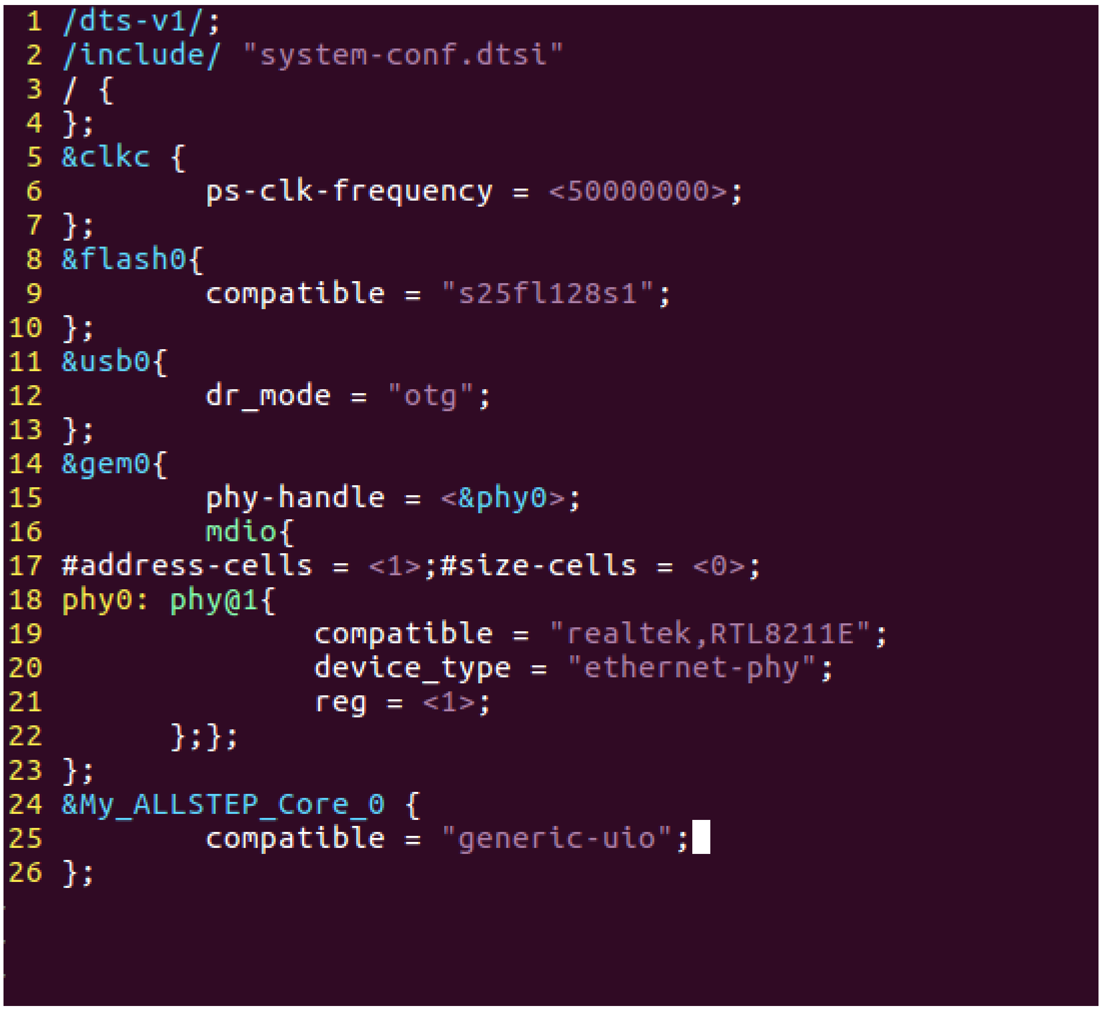
```
INFO: [Common 17-206] Exiting hsi at Thu Oct  4 23:12:38 2018...
[INFO ] oldconfig linux/u-boot
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test/software$ petalinux-config -c kernel
```



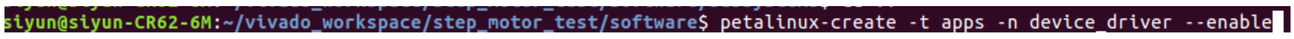
```
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test/software$ petalinux-config -c rootfs
```



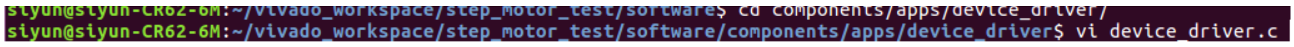
```
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test/software/subsystems/linux/configs/device-tree$ vi system-top.dts
```



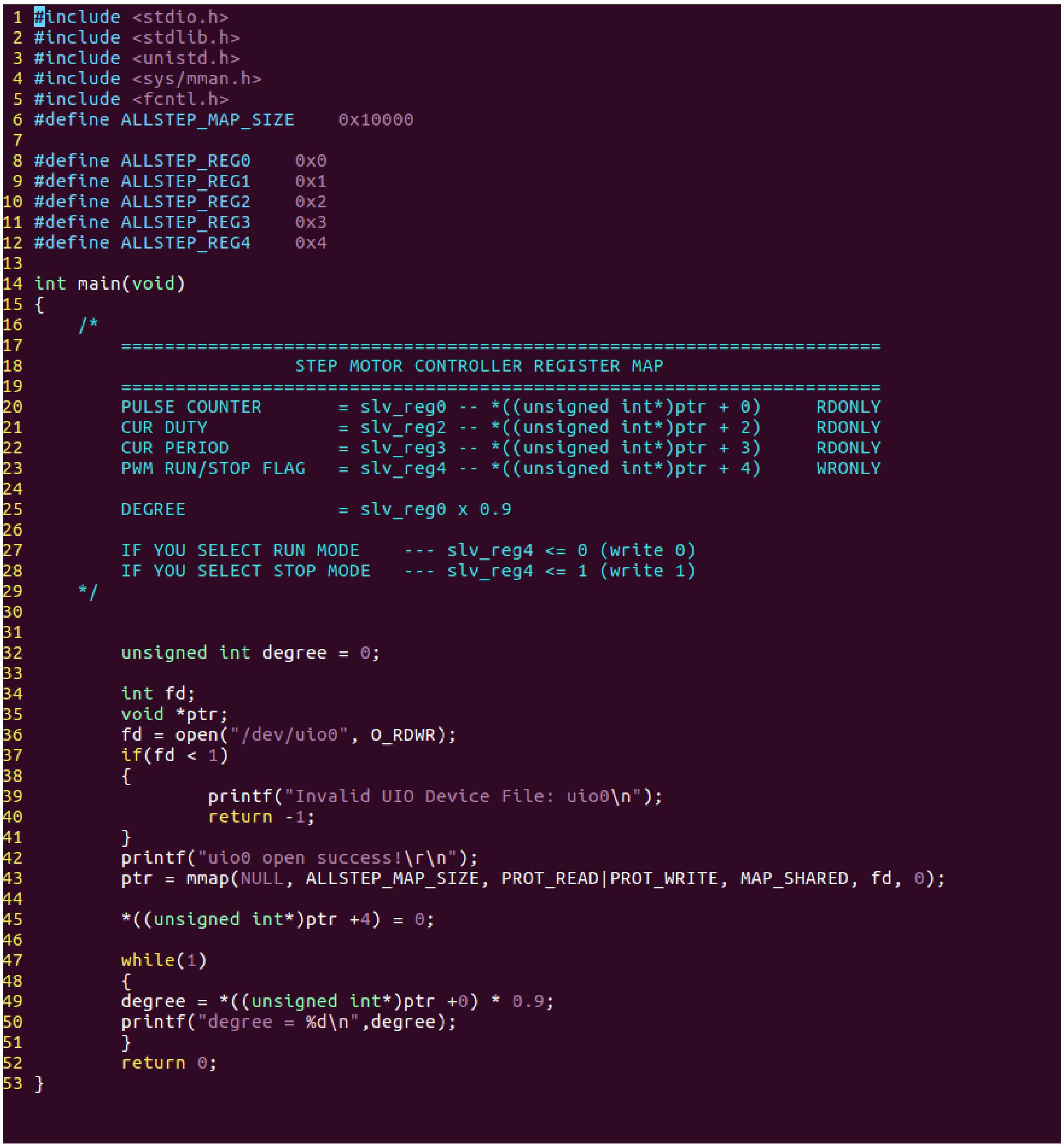
```
/dts-v1/;
/include/ "system-conf.dtsi"
/ {
};
&clkc {
        ps-clk-frequency = <50000000>;
};
&flash0{
        compatible = "s25fl128s1";
};
&usb0{
        dr_mode = "otg";
};
&gem0{
        phy-handle = <&phy0>;
        mdio{
#address-cells = <1>;#size-cells = <0>;
phy0: phy@1{
                compatible = "realtek,RTL8211E";
                device_type = "ethernet-phy";
                reg = <1>;
        };};
};
&My_ALLSTEP_Core_0 {
        compatible = "generic-uio";
};
```



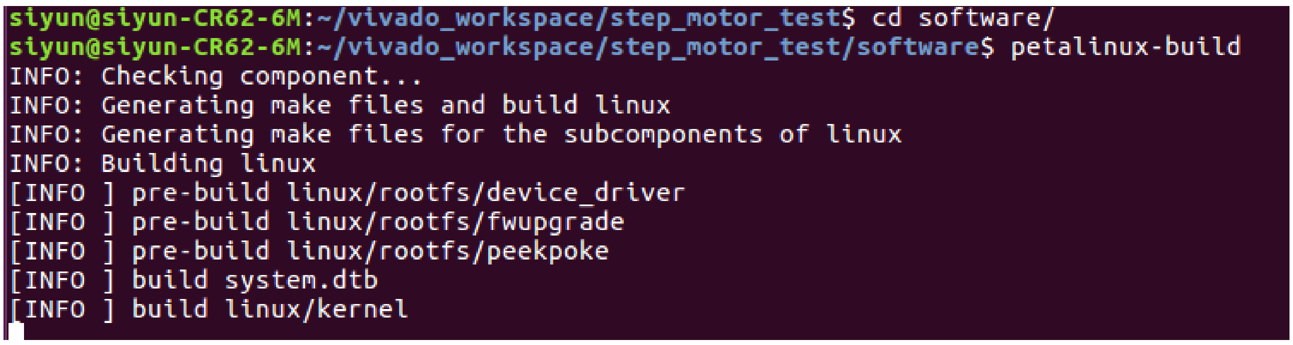
```
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test/software$ petalinux-create -t apps -n device_driver --enable
```



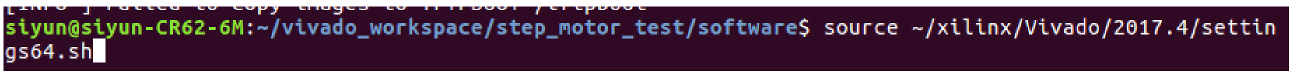
```
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test/software$ cd components/apps/device_driver/
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test/software/components/apps/device_driver$ vi device_driver.c
```



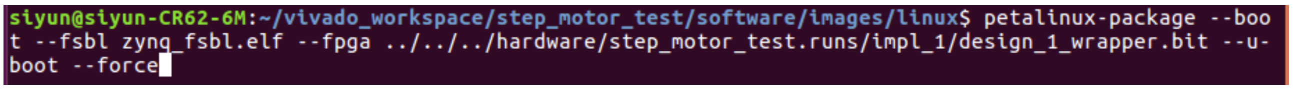
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#define ALLSTEP_MAP_SIZE    0x10000

#define ALLSTEP_REG0    0x0
#define ALLSTEP_REG1    0x1
#define ALLSTEP_REG2    0x2
#define ALLSTEP_REG3    0x3
#define ALLSTEP_REG4    0x4

int main(void)
{
    /*
        ======================================================================
                        STEP MOTOR CONTROLLER REGISTER MAP
        ======================================================================
        PULSE COUNTER       = slv_reg0 -- *((unsigned int*)ptr + 0)     RDONLY
        CUR DUTY            = slv_reg2 -- *((unsigned int*)ptr + 2)     RDONLY
        CUR PERIOD          = slv_reg3 -- *((unsigned int*)ptr + 3)     RDONLY
        PWM RUN/STOP FLAG   = slv_reg4 -- *((unsigned int*)ptr + 4)     WRONLY

        DEGREE              = slv_reg0 x 0.9

        IF YOU SELECT RUN MODE    --- slv_reg4 <= 0 (write 0)
        IF YOU SELECT STOP MODE   --- slv_reg4 <= 1 (write 1)
    */


        unsigned int degree = 0;

        int fd;
        void *ptr;
        fd = open("/dev/uio0", O_RDWR);
        if(fd < 1)
        {
                printf("Invalid UIO Device File: uio0\n");
                return -1;
        }
        printf("uio0 open success!\r\n");
        ptr = mmap(NULL, ALLSTEP_MAP_SIZE, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);

        *((unsigned int*)ptr +4) = 0;

        while(1)
        {
        degree = *((unsigned int*)ptr +0) * 0.9;
        printf("degree = %d\n",degree);
        }
        return 0;
}
```



```
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test$ cd software/
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test/software$ petalinux-build
INFO: Checking component...
INFO: Generating make files and build linux
INFO: Generating make files for the subcomponents of linux
INFO: Building linux
[INFO ] pre-build linux/rootfs/device_driver
[INFO ] pre-build linux/rootfs/fwupgrade
[INFO ] pre-build linux/rootfs/peekpoke
[INFO ] build system.dtb
[INFO ] build linux/kernel
```



```
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test/software$ source ~/xilinx/Vivado/2017.4/settin
gs64.sh
```



```
siyun@siyun-CR62-6M:~/vivado_workspace/step_motor_test/software/images/linux$ petalinux-package --boo
t --fsbl zynq_fsbl.elf --fpga ../../../hardware/step_motor_test.runs/impl_1/design_1_wrapper.bit --u-
boot --force
```

verilog all code sub module

```
`timescale 1 ns / 1 ps

        module My_ALLSTEP_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        = 5
        )
        (
                // Users to add ports here
     input wire PWM_SIG,
     input wire Z_SIG,
     output reg PWM1,
                // 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, acceped 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, acceped 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, acceped 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
	);

	// user reg
	  reg PAST_Z_SIG;
reg PAST_PWM_SIG;
reg [31:0] counter;
reg [31:0] O_counter;
reg [31:0] PAST_PWM_DUTY;
reg [31:0] PAST_PWM_PERIOD;
reg [31:0] pwm_counter;
reg [31:0] duty;
reg [31:0] period;
	// 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 = 2;
//----------------------------------------------
//-- Signals for user logic register space example
//------------------------------------------------
//-- Number of Slave Registers 8
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;
reg [C_S_AXI_DATA_WIDTH-1:0]        slv_reg4;
reg [C_S_AXI_DATA_WIDTH-1:0]        slv_reg5;
reg [C_S_AXI_DATA_WIDTH-1:0]        slv_reg6;
reg [C_S_AXI_DATA_WIDTH-1:0]        slv_reg7;
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

    // 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;
              slv_reg4 <= 0;
              slv_reg5 <= 0;
              slv_reg6 <= 0;
              slv_reg7 <= 0;
            end
          else begin
            if (slv_reg_wren)
              begin
                case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
                  3'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
                  3'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
                  3'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
                  3'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
                  3'h4:
                    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 4
                        slv_reg4[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                      end
                  3'h5:
                    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 5
                        slv_reg5[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                      end
                  3'h6:
                    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 6
                        slv_reg6[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
                      end
                  3'h7:
                    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 7
                        slv_reg7[(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;
                              slv_reg4 <= slv_reg4;
                              slv_reg5 <= slv_reg5;
                              slv_reg6 <= slv_reg6;
                              slv_reg7 <= slv_reg7;
                            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
              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_awready 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 acceped 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_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] )
                  3'h0   : reg_data_out <= counter;
                  3'h1   : reg_data_out <= slv_reg1;
                  3'h2   : reg_data_out <= duty;
                  3'h3   : reg_data_out <= period;
                  3'h4   : reg_data_out <= slv_reg4;
                  3'h5   : reg_data_out <= slv_reg5;
                  3'h6   : reg_data_out <= slv_reg6;
                  3'h7   : reg_data_out <= slv_reg7;
                  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 dada
                if (slv_reg_rden)
                  begin
                    axi_rdata <= reg_data_out;     // register read data
                  end
              end
          end

initial begin

duty <= 32'd15625;//32'd31250;
period <= 32'd31250;//32'd62500;

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


             if(slv_reg4 == 32'd0) begin

             pwm_counter = pwm_counter + 32'd1;

             if(pwm_counter < duty)
             begin
                 PWM1 <= 1'b1;
             end
             else if(pwm_counter >= duty)
             begin
                 PWM1 <= 1'b0;
                 if(pwm_counter == period)
                 begin
                 pwm_counter <= 32'd0;
                 end
            end
             end
            if(slv_reg4 == 32'd1) begin
               if(Z_SIG ==1) begin
                  pwm_counter <= 32'd0;
                  PWM1 <= 0;
                  end
            end

            if(PWM_SIG == 1 && PAST_PWM_SIG == 0)
            begin

            if(counter > 32'd399) begin
                counter <= 32'd0;
                end

               counter <= counter + 32'd1;


               end
               PAST_PWM_SIG <= PWM_SIG;
               /*
               if(Z_SIG == 1'b1)
               begin
                  O_counter <= 32'd1;
                  end
               if(Z_SIG == 1'b0) begin
                  O_counter <= 32'd0;
                  end
                 */
          end
   //
   //slv_reg0 = pulse counter
   // slv_reg1 = Z counter
   // slv_reg2 = step motor duty
   // slv_reg3 = step motor period
   //slv_reg4 = PWM_RUN/STOP_FLAG
   // User logic ends

          endmodule
```

Verilog code top module

```
`timescale 1 ns / 1 ps

          module My_ALLSTEP_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        = 5
          )
          (
                    // Users to add ports here
           input wire PWM_SIG,
		input wire Z_SIG,
		output wire PWM1,
		// 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_ALLSTEP_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_ALLSTEP_Core_v1_0_S00_AXI_inst (
	    .PWM_SIG(PWM_SIG),
	    .Z_SIG(Z_SIG),
	    .PWM1(PWM1),
		.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
```