TRC1000 RecoNode v1.1

Xilinx ISE 14.7 Tutorial

Sangjun Eom

Date: 7.10.2018

PURDUE
POLYTECHNIC

Table of Contents

1. **Hardware Setup**
2. **Copy an Existing Project**
    - **A. Test the Project**
        - a. Open the Project on Xilinx Platform Studio
        - b. Import the Project to Xilinx SDK
        - c. Program the Hardware
    - **B. Modify the Project**
        - a. Add a New Software Module
        - b. Add a New IP Core (Add / Change Wedges)
        - c. Create a New IP Core
3. **Create a New Project**
4. **Troubleshooting**

HARDWARE SETUP

1. CPU and I/O

The RecoNode is a reconfigurable computational node for creating heterogeneous wireless control networks.

Each node includes a hard-core PowerPC CPU (reconfigurable software), an FPGA (reconfigurable computational hardware), and two MorphingBus peripheral I/O buses (reconfigurable I/O hardware).

From 1 to n nodes can be configured to create an integrated control network using PBO/RT software.

1. Hardware Setup

You can either use…

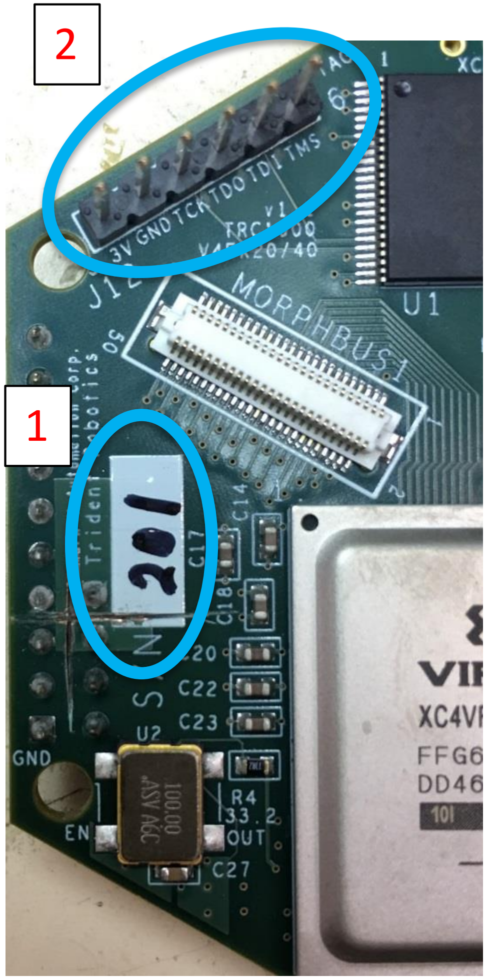
- New RecoNode TRC1000 v1.1 (S/N 200 – 215)
- Old RecoNode TRC1000 v1.0 (S/N 000 – 199)

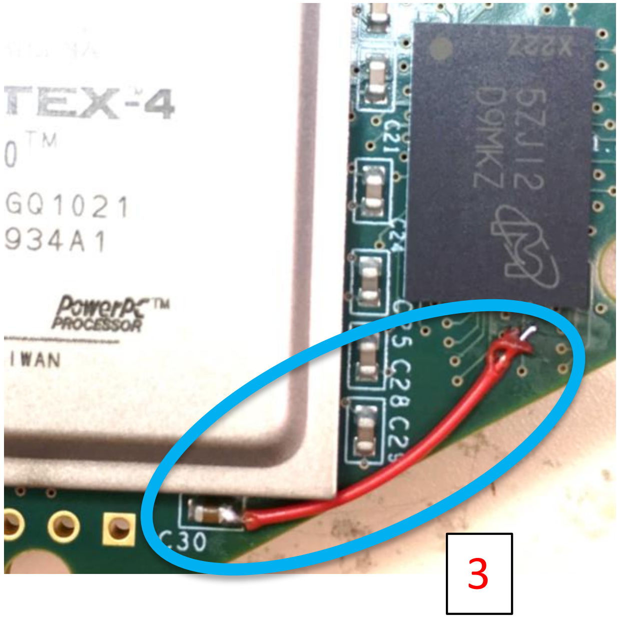
Both RecoNode versions 1.1 and 1.0 are based on the Xilinx Virtex4 FPGA with onboard PROM and DRAM.

The MorphingBus depends on the configuration of the FPGA for proper operation, so we have a few “standard I/O stacks” to make start-up easy.

See the RecoNode Morphing Bus manual for I/O options.

1. Identifying a Good RecoNode v1.1





TRC1000 v1.1

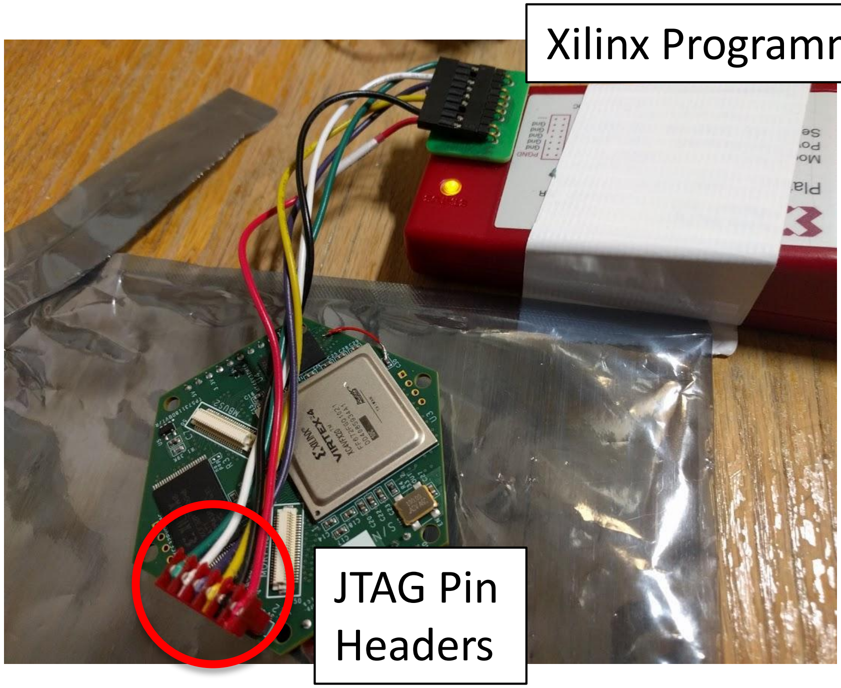
SN # 200 - 215

The RecoNode PCB includes some routing errors that must be fixed with “blue wires” for proper operation for either v1.0 or v1.1. If your RecoNode has a serial number (Fig. 1), it should be ready to go.

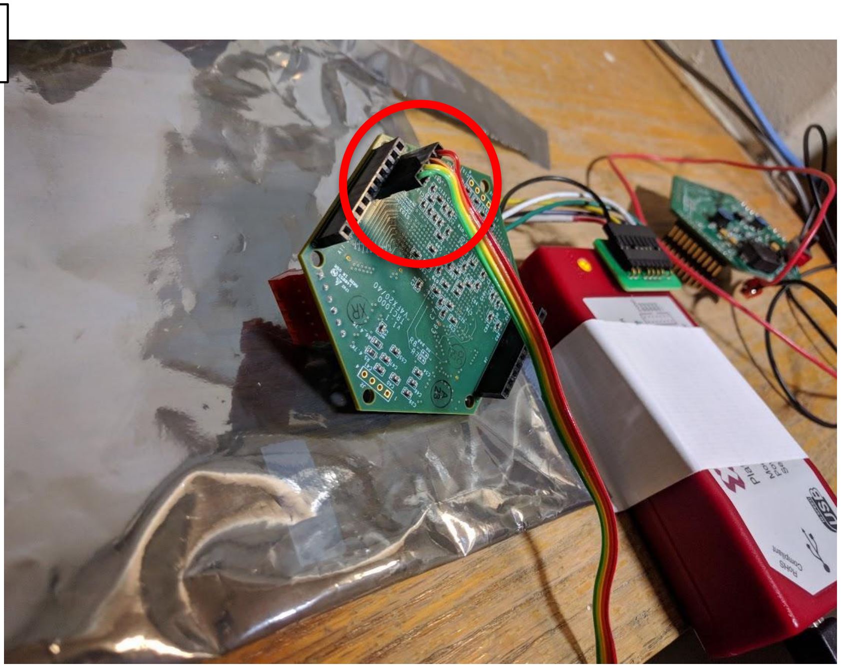
Some indicators of fixes include:

- JTAG plug has been relocated to bottom side (Fig. 2)
- Power wire added to DRAM (Fig. 3)

1. Hardware Setup


Xilinx Programmer

JTAG Pin Headers



Connect Xilinx Programmer to JTAG Pin Headers on RecoNode

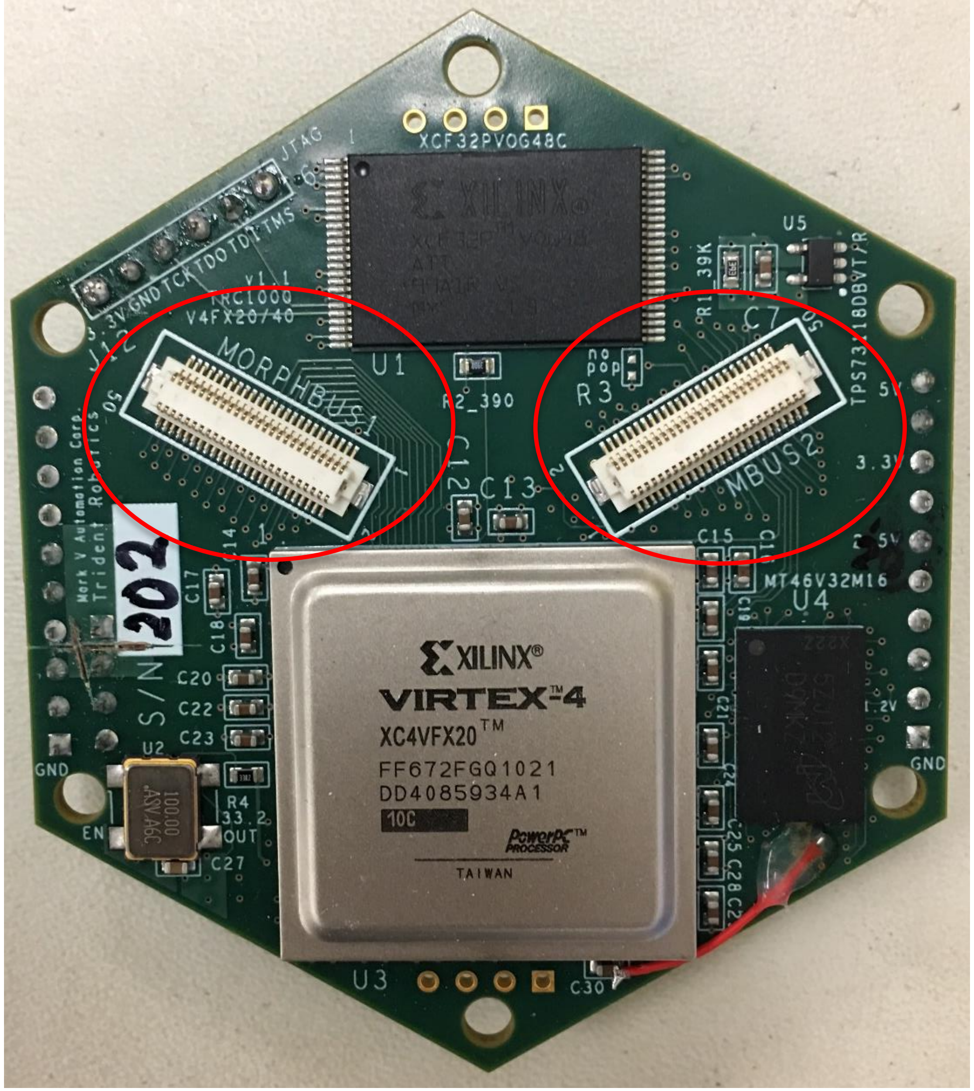
Connect UART to UART Pin Headers (J5) located on bottom side of RecoNode

1. Hardware Setup

Supply **3.7** Volts to the power board. TRC 1000 should be connected to JTAG, serial communication cable, and power board

```
If (The board is not programmed) then
    If the current is around 0.2 – 0.3 A then
        Pass, it is good
    If the current is > 0.3 A, then
        Turn off the power supply. Something might be shorted.

If (The chip is programmed) then
    If The current rises from 0.2 – 0.3 A to 0.5 – 0.6 A. then
        Pass, it is good
    If the current is > ~0.65 A - after programmed then
        Turn off the power supply. Something might be shorted.
```

1. Hardware Setup – MorphingBus



Standard I/O Stack

RecoNode has two MorphingBus connectors where you can stack I/O wedges in a double-helix.

The *CRL Standard Stack* is composed of

Morphing Bus 1 (left in Fig)

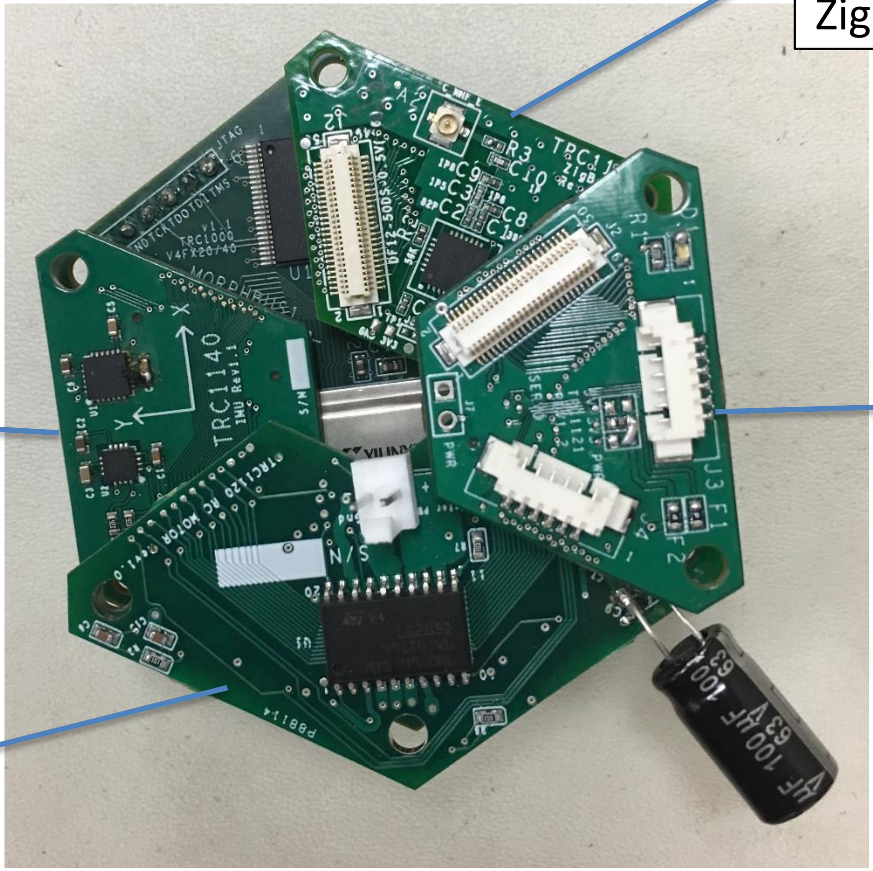
- TRC1140 – IMU Wedge (1st level)
- TRC1120 – Motor Wedge (2nd level)
- TRC1121 – Servo Wedge (3rd level)

Morphing Bus 2 (right in Fig)

- TRC1150 – Zigbee Wedge (1st level)

However, you can customize your stack and change the number & order of the I/O wedges. Different projects use different I/O configurations and the FPGA configuration in the XPS must reflect the physical stack.

1. Hardware Setup – Standard I/O Stack



TRC1150
Zigbee Wedge

1st Level

1st Level

TRC1140
IMU Wedge

TRC1121
Servo Wedge

3rd Level

2nd Level

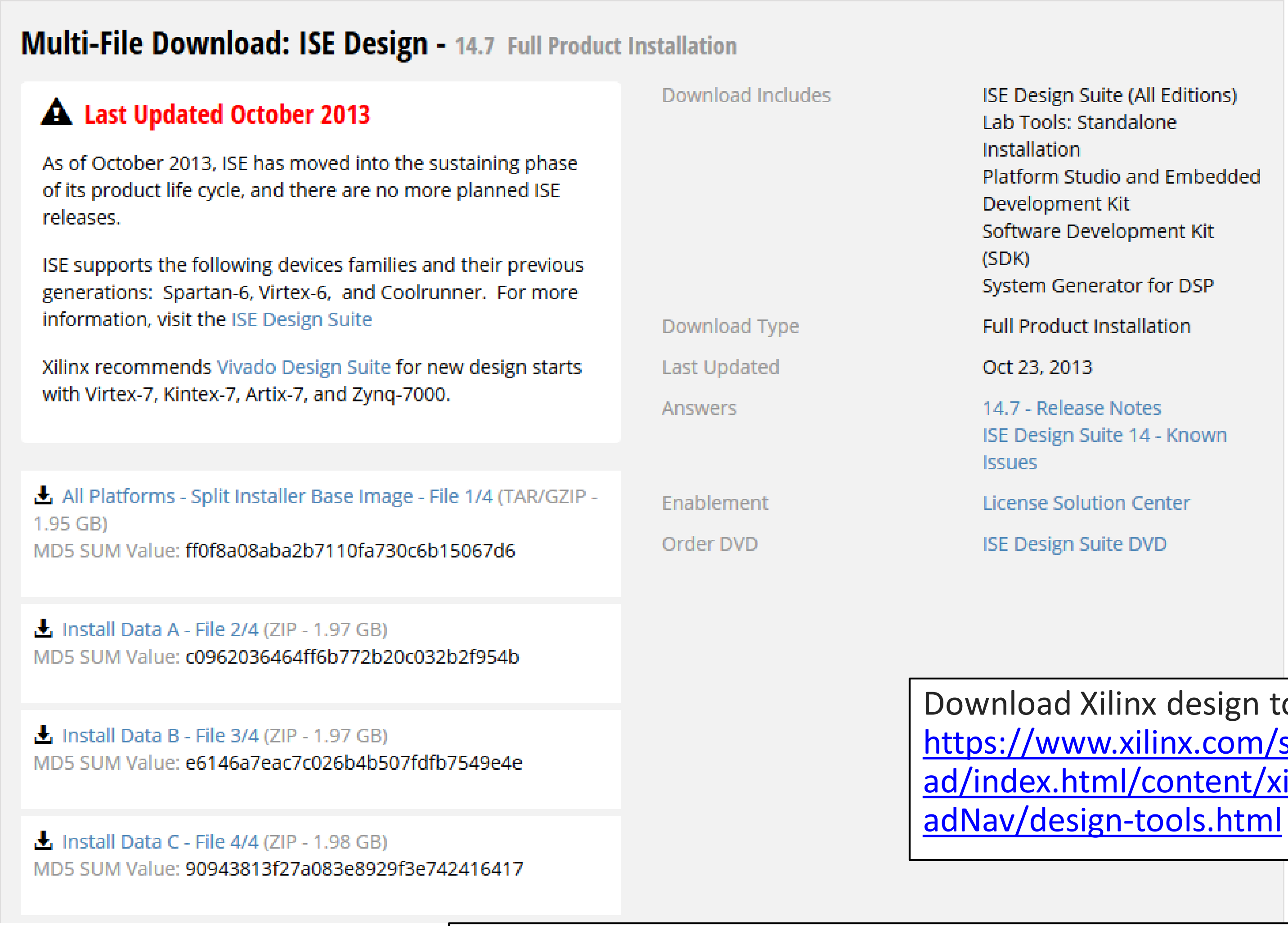
TRC1120
Motor Wedge
(Double)

Standard Stack

2. Copy an Existing Project

First, let’s install ISE 14.7 software from Xilinx.

2. Installation of Software



Multi-File Download: ISE Design - 14.7 Full Product Installation

Last Updated October 2013

As of October 2013, ISE has moved into the sustaining phase of its product life cycle, and there are no more planned ISE releases.

ISE supports the following devices families and their previous generations: Spartan-6, Virtex-6, and Coolrunner. For more information, visit the ISE Design Suite

Xilinx recommends Vivado Design Suite for new design starts with Virtex-7, Kintex-7, Artix-7, and Zynq-7000.

Download Includes	ISE Design Suite (All Editions) Lab Tools: Standalone Installation Platform Studio and Embedded Development Kit Software Development Kit (SDK) System Generator for DSP
Download Type	Full Product Installation
Last Updated	Oct 23, 2013
Answers	14.7 - Release Notes ISE Design Suite 14 - Known Issues
Enablement	License Solution Center
Order DVD	ISE Design Suite DVD

All Platforms - Split Installer Base Image - File 1/4 (TAR/GZIP - 1.95 GB)
MD5 SUM Value: ff0f8a08aba2b7110fa730c6b15067d6

Install Data A - File 2/4 (ZIP - 1.97 GB)
MD5 SUM Value: c0962036464ff6b772b20c032b2f954b

Install Data B - File 3/4 (ZIP - 1.97 GB)
MD5 SUM Value: e6146a7eac7c026b4b507fdfb7549e4e

Install Data C - File 4/4 (ZIP - 1.98 GB)
MD5 SUM Value: 90943813f27a083e8929f3e742416417

Download Xilinx design tools from here: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html

If they ask for licenses, input this to ”path to license”: 2100@marina.ecn.purdue.edu (purchased by Dr. Richard M. Voyles for CRL), (**Last update 12/12/2016**)

2. A. Test the Project

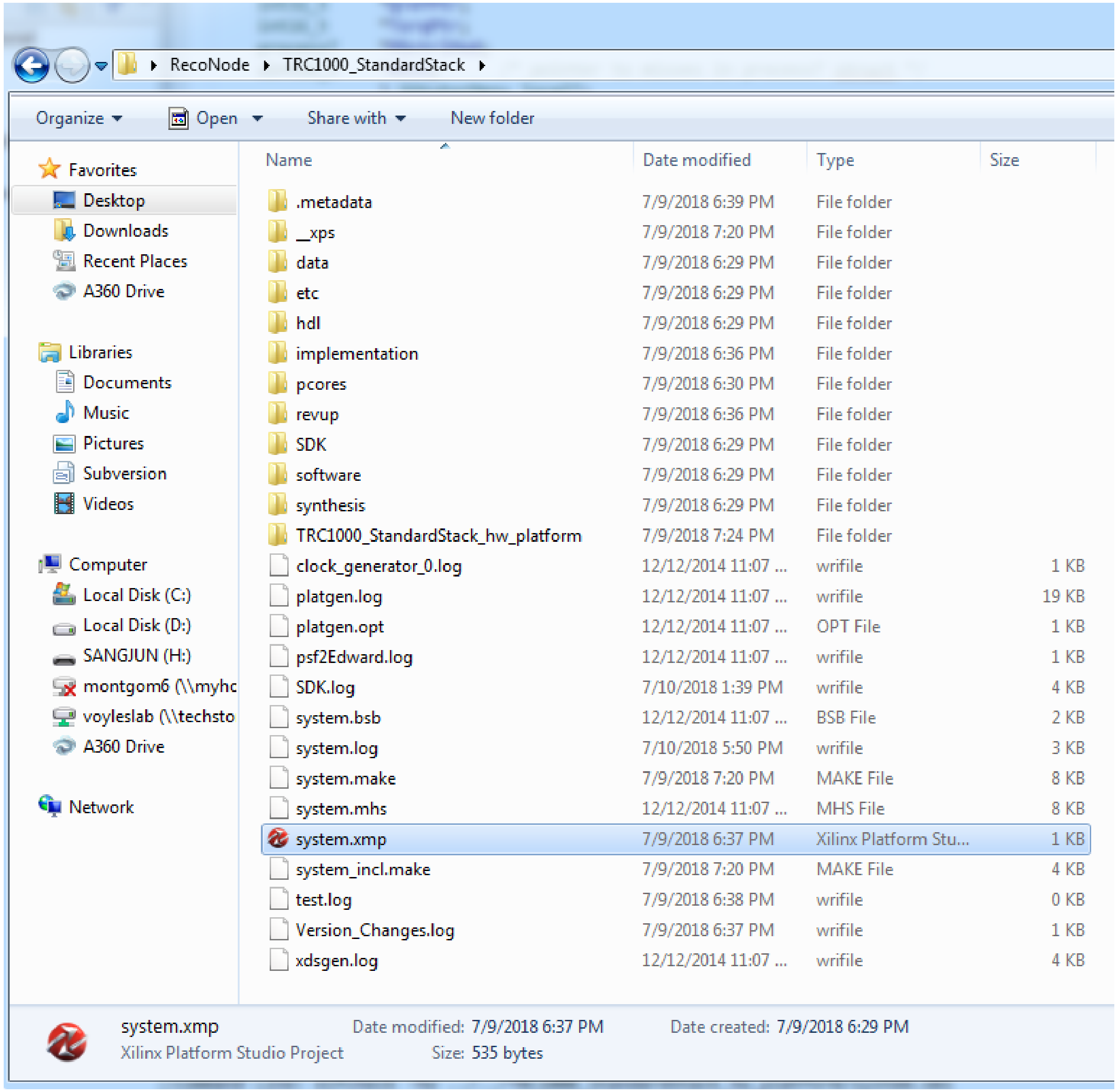
a. Open on Xilinx Platform Studio
b. Import the Project to Xilinx SDK
c. Program the Hardware

We will program the hardware with an existing project and test it.

OPEN EXISTING PROJECT WITH XPS

a. Xilinx Platform Studio (XPS) Provides Hardware Configuration Tools
   - The XPS allows the configuration of the FPGA hardware
   - VHDL and Verilog code can be written for custom logic
   - IP cores can be embedded from the Xilinx library
   - Signals can be routed to different pins

2. A. a. Xilinx Platform Studio



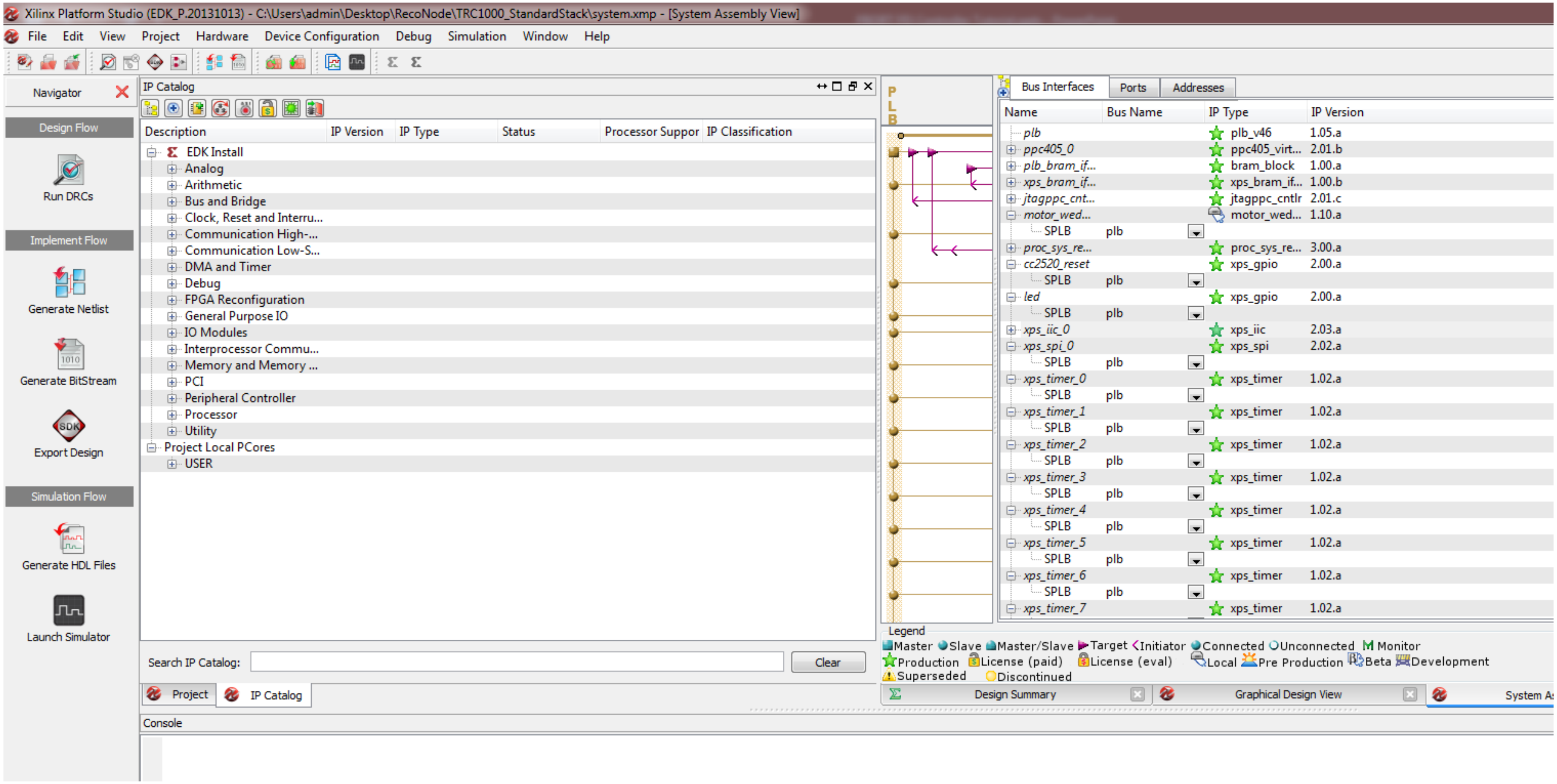
Name	Date modified	Type	Size
.metadata	7/9/2018 6:39 PM	File folder	
_xps	7/9/2018 7:20 PM	File folder	
data	7/9/2018 6:29 PM	File folder	
etc	7/9/2018 6:29 PM	File folder	
hdl	7/9/2018 6:29 PM	File folder	
implementation	7/9/2018 6:36 PM	File folder	
pcores	7/9/2018 6:30 PM	File folder	
revup	7/9/2018 6:36 PM	File folder	
SDK	7/9/2018 6:29 PM	File folder	
software	7/9/2018 6:29 PM	File folder	
synthesis	7/9/2018 6:29 PM	File folder	
TRC1000_StandardStack_hw_platform	7/9/2018 7:24 PM	File folder	
clock_generator_0.log	12/12/2014 11:07 ...	wrifile	1 KB
platgen.log	12/12/2014 11:07 ...	wrifile	19 KB
platgen.opt	12/12/2014 11:07 ...	OPT File	1 KB
psf2Edward.log	12/12/2014 11:07 ...	wrifile	1 KB
SDK.log	7/10/2018 1:39 PM	wrifile	4 KB
system.bsb	12/12/2014 11:07 ...	BSB File	2 KB
system.log	7/10/2018 5:50 PM	wrifile	3 KB
system.make	7/9/2018 7:20 PM	MAKE File	8 KB
system.mhs	12/12/2014 11:07 ...	MHS File	8 KB
system.xmp	7/9/2018 6:37 PM	Xilinx Platform Stu...	1 KB
system_incl.make	7/9/2018 7:20 PM	MAKE File	4 KB
test.log	7/9/2018 6:38 PM	wrifile	0 KB
Version_Changes.log	7/9/2018 6:37 PM	wrifile	1 KB
xdsgen.log	12/12/2014 11:07 ...	wrifile	4 KB

system.xmp Date modified: 7/9/2018 6:37 PM Date created: 7/9/2018 6:29 PM
Xilinx Platform Studio Project Size: 535 bytes

We are starting with an existing project, so all logic has been already defined.

Open the existing *.xmp file so we can export the hardware definition to the software environment.

2. A. a. Xilinx Platform Studio



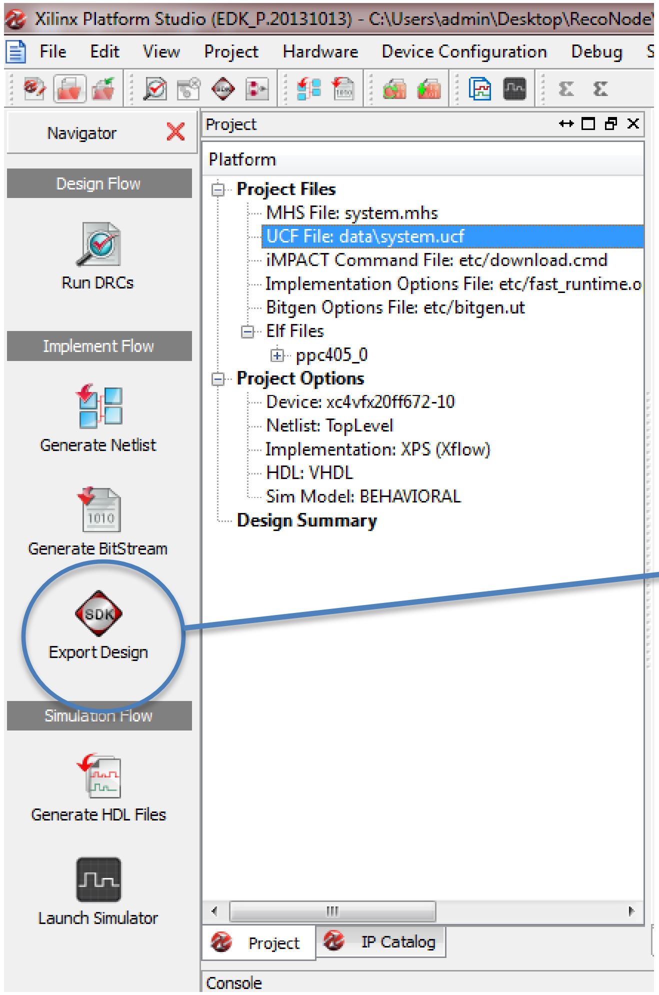
Description	IP Version	IP Type	Status	Processor Suppor	IP Classification
⊟ EDK Install					
⊞ Analog					
⊞ Arithmetic					
⊞ Bus and Bridge					
⊞ Clock, Reset and Interru...					
⊞ Communication High-...					
⊞ Communication Low-S...					
⊞ DMA and Timer					
⊞ Debug					
⊞ FPGA Reconfiguration					
⊞ General Purpose IO					
⊞ IO Modules					
⊞ Interprocessor Commu...					
⊞ Memory and Memory ...					
⊞ PCI					
⊞ Peripheral Controller					
⊞ Processor					
⊞ Utility					
⊟ Project Local PCores					
⊞ USER					

Bus Interfaces | Ports | Addresses

Name	Bus Name	IP Type	IP Version
plb		plb_v46	1.05.a
⊞ ppc405_0		ppc405_virt...	2.01.b
⊞ plb_bram_if...		bram_block	1.00.a
⊞ xps_bram_if...		xps_bram_if...	1.00.b
⊞ jtagppc_cnt...		jtagppc_cntlr	2.01.c
⊟ motor_wed...		motor_wed...	1.10.a
SPLB	plb		
⊞ proc_sys_re...		proc_sys_re...	3.00.a
⊟ cc2520_reset		xps_gpio	2.00.a
SPLB	plb		
⊟ led		xps_gpio	2.00.a
SPLB	plb		
⊞ xps_iic_0		xps_iic	2.03.a
⊟ xps_spi_0		xps_spi	2.02.a
SPLB	plb		
⊟ xps_timer_0		xps_timer	1.02.a
SPLB	plb		
⊟ xps_timer_1		xps_timer	1.02.a
SPLB	plb		
⊟ xps_timer_2		xps_timer	1.02.a
SPLB	plb		
⊟ xps_timer_3		xps_timer	1.02.a
SPLB	plb		
⊟ xps_timer_4		xps_timer	1.02.a
SPLB	plb		
⊟ xps_timer_5		xps_timer	1.02.a
SPLB	plb		
⊟ xps_timer_6		xps_timer	1.02.a
SPLB	plb		
⊟ xps_timer_7		xps_timer	1.02.a

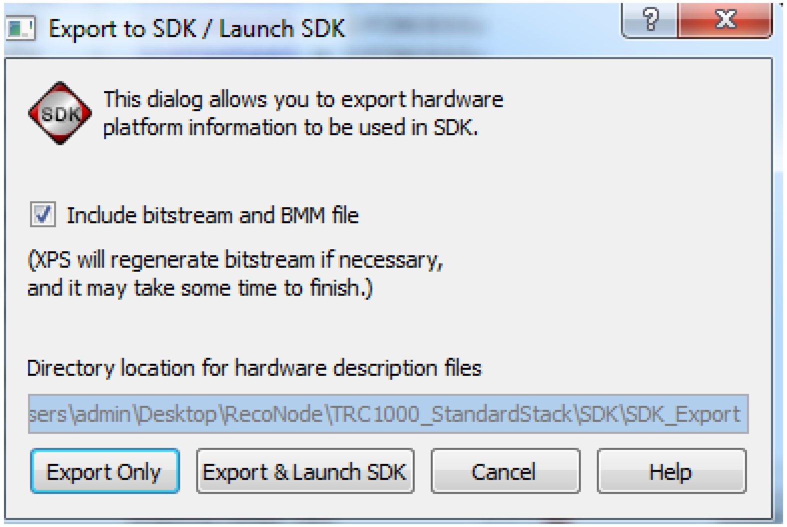
2. A. a. Xilinx Platform Studio

We want to export the existing design and launch the SDK.



Xilinx Platform Studio (EDK_P.20131013) - C:\Users\admin\Desktop\RecoNode\

- Project Files
  - MHS File: system.mhs
  - UCF File: data\system.ucf
  - iMPACT Command File: etc/download.cmd
  - Implementation Options File: etc/fast_runtime.o
  - Bitgen Options File: etc/bitgen.ut
  - Elf Files
    - ppc405_0
- Project Options
  - Device: xc4vfx20ff672-10
  - Netlist: TopLevel
  - Implementation: XPS (Xflow)
  - HDL: VHDL
  - Sim Model: BEHAVIORAL
- Design Summary



Export to SDK / Launch SDK

This dialog allows you to export hardware platform information to be used in SDK.

☑ Include bitstream and BMM file

(XPS will regenerate bitstream if necessary, and it may take some time to finish.)

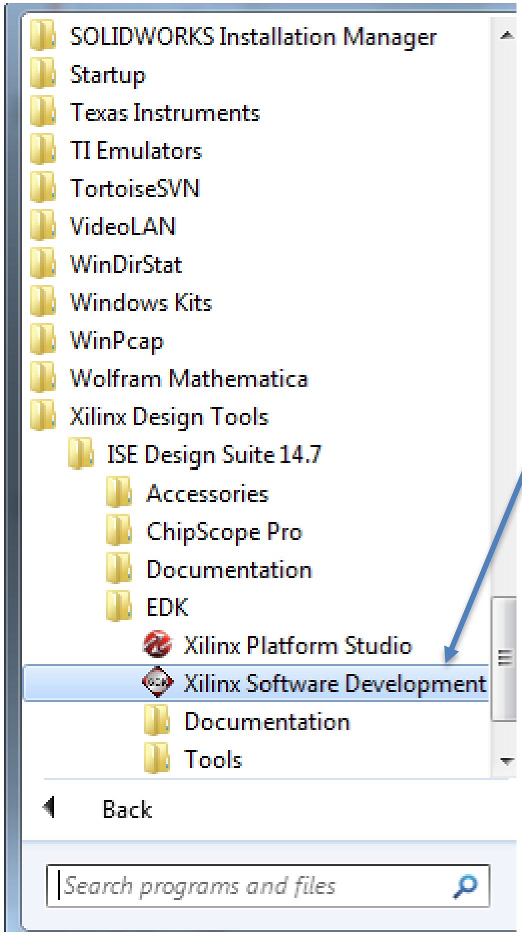
Directory location for hardware description files

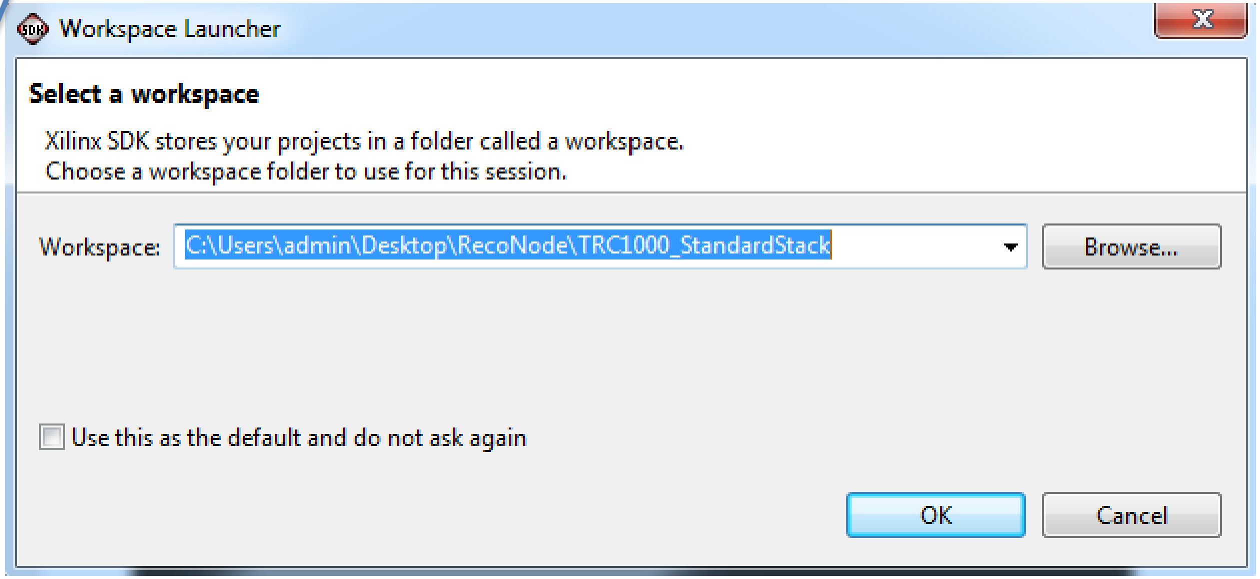
sers\admin\Desktop\RecoNode\TRC1000_StandardStack\SDK\SDK_Export

b. Import the Project to Xilinx SDK

2. A. b. Xilinx SDK

Open Xilinx Software Development Kit (SDK) and select the same workspace that contains your ***.xmp*** file





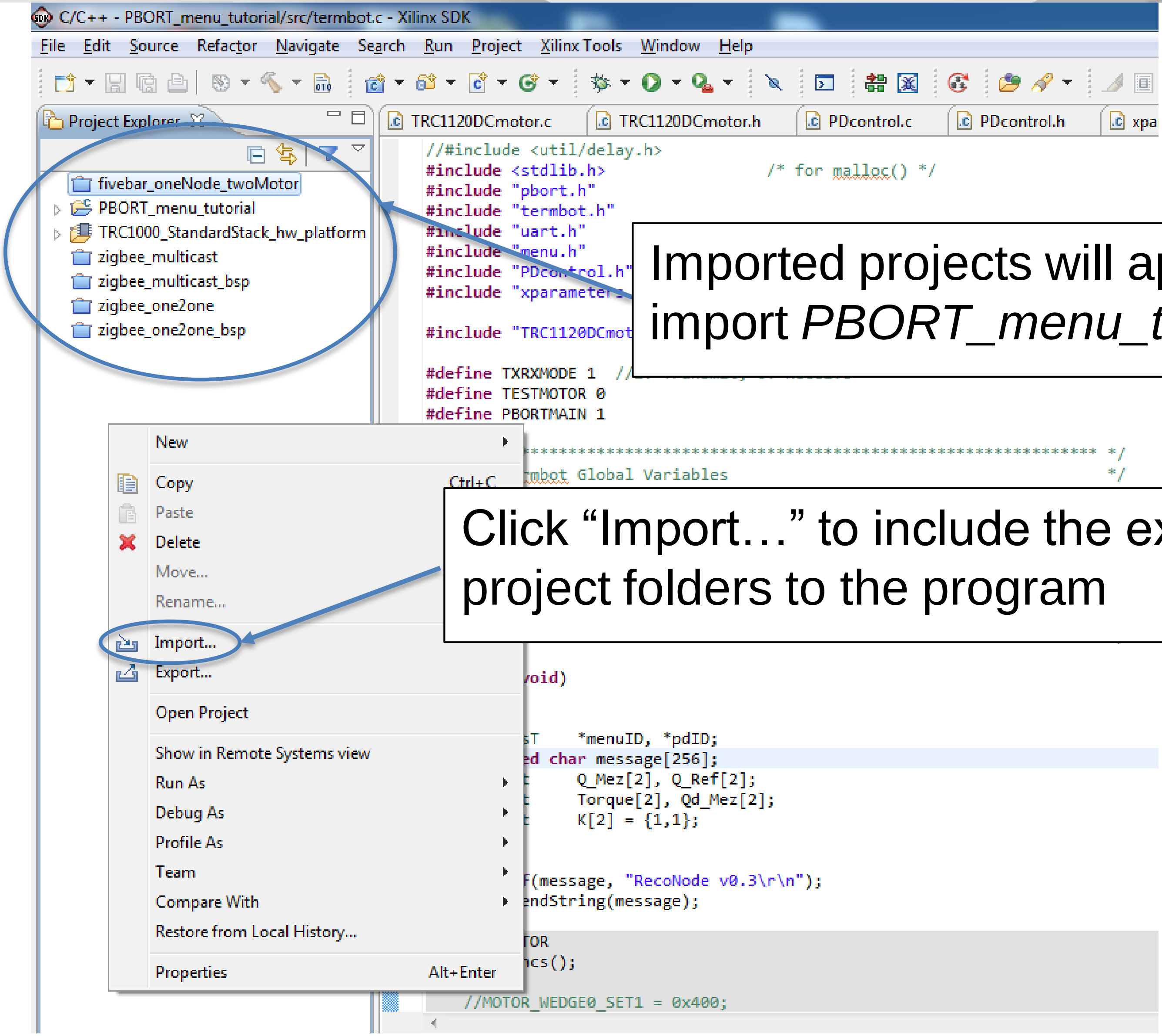
Select a workspace

Xilinx SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\admin\Desktop\RecoNode\TRC1000_StandardStack

☐ Use this as the default and do not ask again

2. A. b. Xilinx SDK

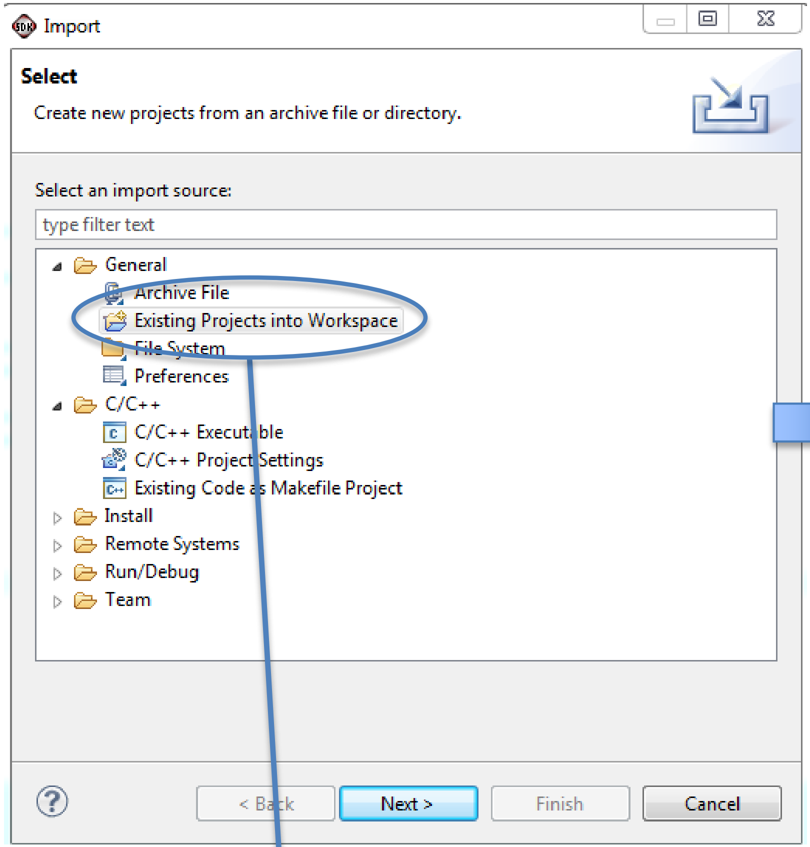


```
//#include <util/delay.h>
#include <stdlib.h>                /* for malloc() */
#include "pbort.h"
#include "termbot.h"
#include "uart.h"
#include "menu.h"
#include "PDcontrol.h"
#include "xparameter

#include "TRC1120DCmot

#define TXRXMODE 1  //
#define TESTMOTOR 0
#define PBORTMAIN 1

******************************************************* */
rmbot Global Variables                                   */

void)

sT     *menuID, *pdID;
ed char message[256];
t      Q_Mez[2], Q_Ref[2];
t      Torque[2], Qd_Mez[2];
t      K[2] = {1,1};

f(message, "RecoNode v0.3\r\n");
endString(message);

TOR
ics();
  //MOTOR_WEDGE0_SET1 = 0x400;
```

Imported projects will appear here. Please import *PBORT_menu_tutorial* folder

Click “Import…” to include the existing project folders to the program

2. A. b. Xilinx SDK



Import

Select

Create new projects from an archive file or directory.

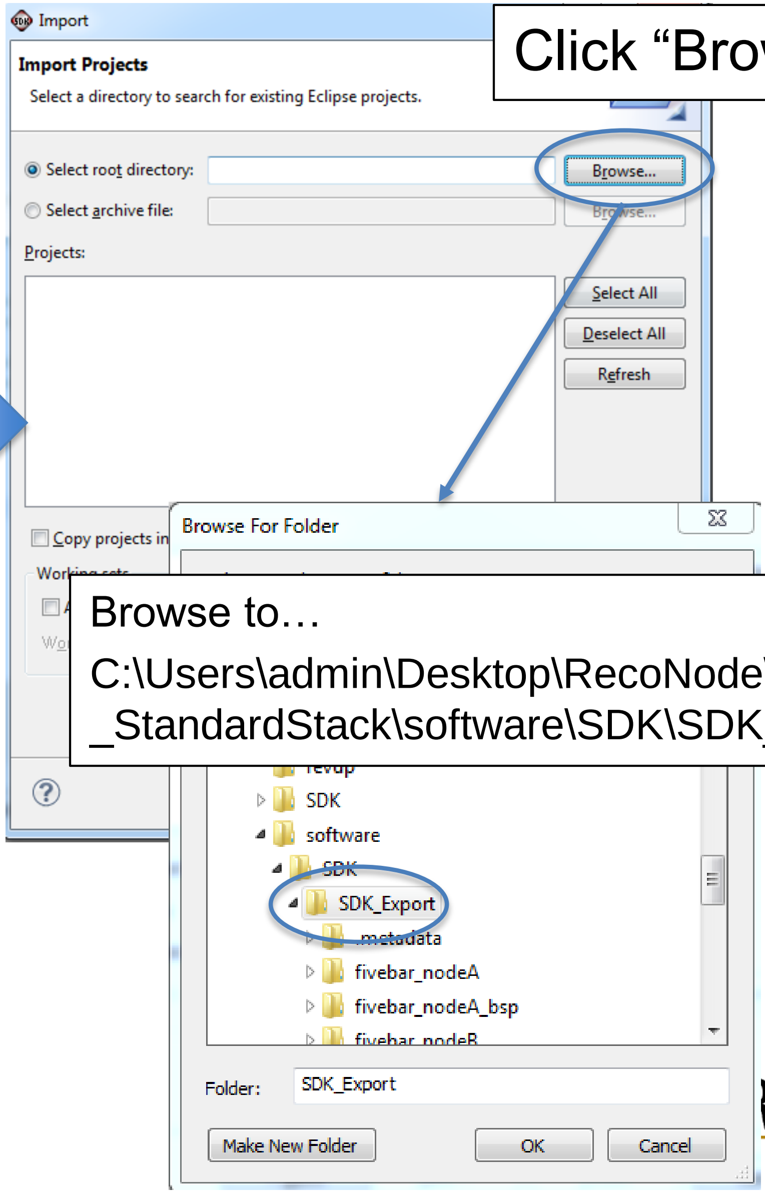
Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- C/C++
  - C/C++ Executable
  - C/C++ Project Settings
  - Existing Code as Makefile Project
- Install
- Remote Systems
- Run/Debug
- Team

< Back | Next > | Finish | Cancel

Continue with
“Existing Projects into Workspace”



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: [Browse...]

○ Select archive file: [Browse...]

Projects:

Select All | Deselect All | Refresh

□ Copy projects in

Working sets

Browse For Folder

- revup
- SDK
- software
  - SDK
    - SDK_Export
      - .metadata
      - fivebar_nodeA
      - fivebar_nodeA_bsp
      - fivebar_nodeB

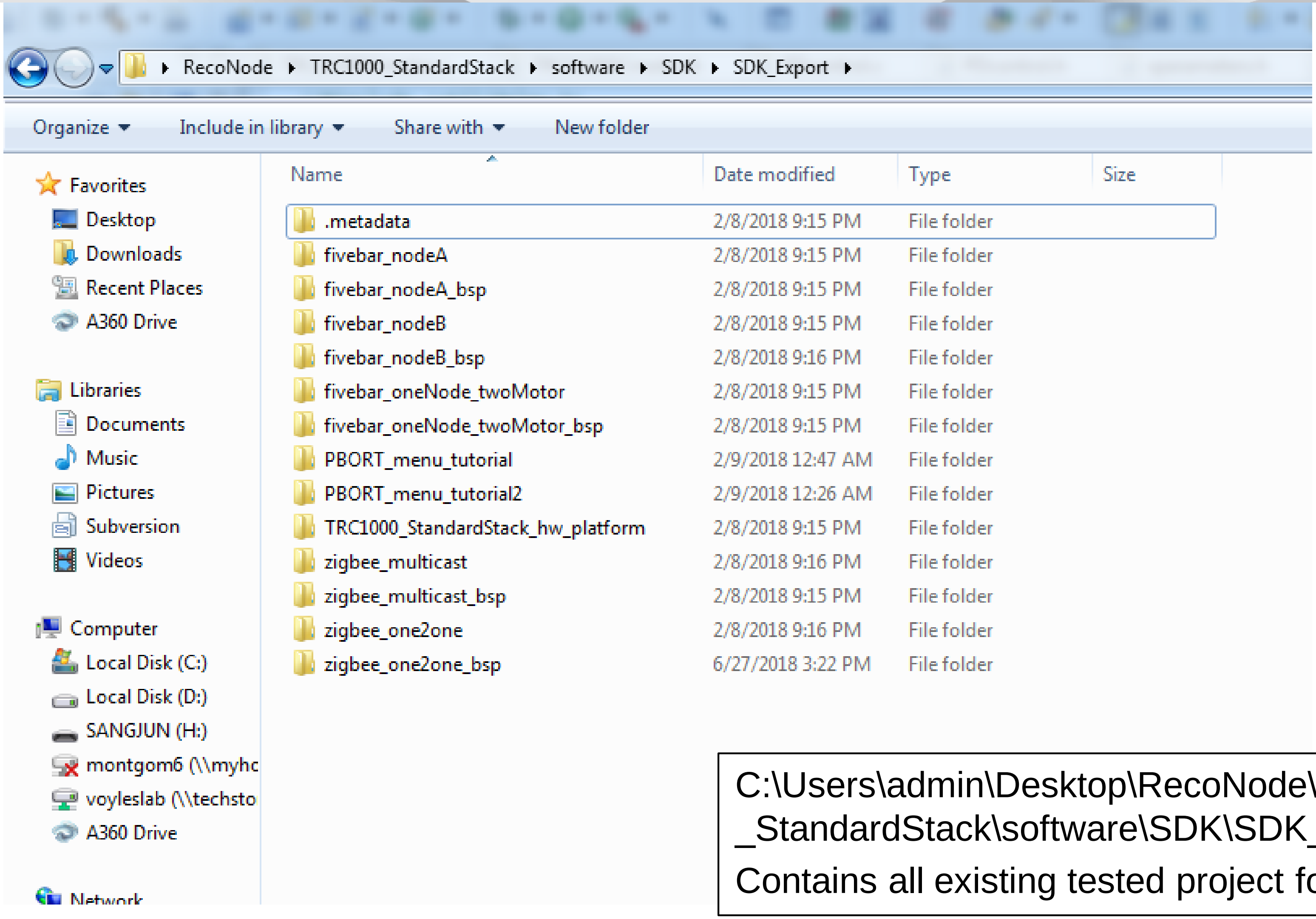
Folder: SDK_Export

Make New Folder | OK | Cancel

Click “Browse…”

Browse to…
C:\Users\admin\Desktop\RecoNode\TRC1000_StandardStack\software\SDK\SDK_Export

2. A. b. Xilinx SDK

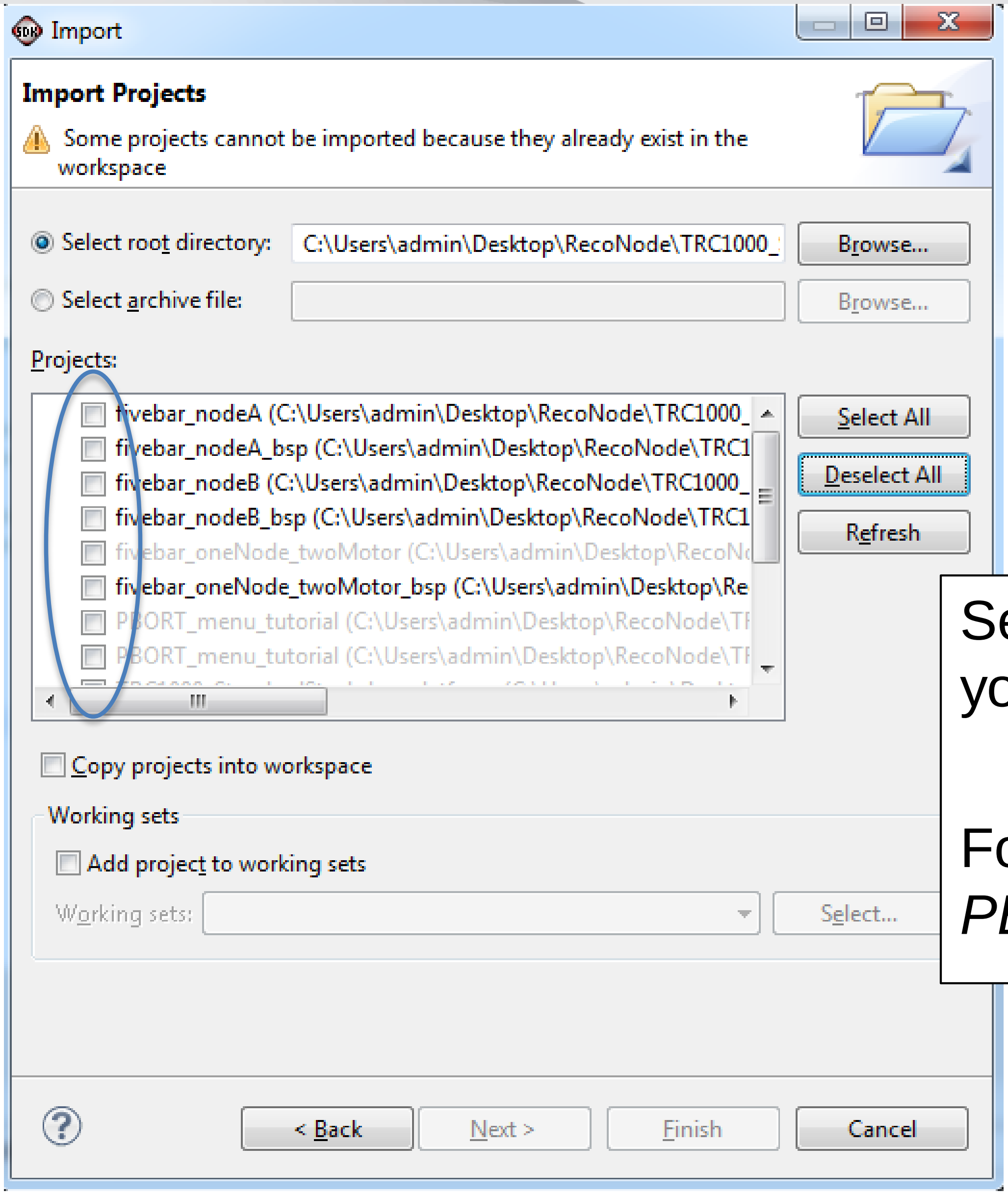


Name	Date modified	Type	Size
.metadata	2/8/2018 9:15 PM	File folder	
fivebar_nodeA	2/8/2018 9:15 PM	File folder	
fivebar_nodeA_bsp	2/8/2018 9:15 PM	File folder	
fivebar_nodeB	2/8/2018 9:15 PM	File folder	
fivebar_nodeB_bsp	2/8/2018 9:16 PM	File folder	
fivebar_oneNode_twoMotor	2/8/2018 9:15 PM	File folder	
fivebar_oneNode_twoMotor_bsp	2/8/2018 9:15 PM	File folder	
PBORT_menu_tutorial	2/9/2018 12:47 AM	File folder	
PBORT_menu_tutorial2	2/9/2018 12:26 AM	File folder	
TRC1000_StandardStack_hw_platform	2/8/2018 9:15 PM	File folder	
zigbee_multicast	2/8/2018 9:16 PM	File folder	
zigbee_multicast_bsp	2/8/2018 9:15 PM	File folder	
zigbee_one2one	2/8/2018 9:16 PM	File folder	
zigbee_one2one_bsp	6/27/2018 3:22 PM	File folder	

C:\Users\admin\Desktop\RecoNode\TRC1000_StandardStack\software\SDK\SDK_Export

Contains all existing tested project folders.

2. A. b. Xilinx SDK



Import

Import Projects

⚠ Some projects cannot be imported because they already exist in the workspace

● Select root directory: C:\Users\admin\Desktop\RecoNode\TRC1000_

○ Select archive file:

Projects:

□ fivebar_nodeA (C:\Users\admin\Desktop\RecoNode\TRC1000_
□ fivebar_nodeA_bsp (C:\Users\admin\Desktop\RecoNode\TRC1
□ fivebar_nodeB (C:\Users\admin\Desktop\RecoNode\TRC1000_
□ fivebar_nodeB_bsp (C:\Users\admin\Desktop\RecoNode\TRC1
□ fivebar_oneNode_twoMotor (C:\Users\admin\Desktop\RecoNo
□ fivebar_oneNode_twoMotor_bsp (C:\Users\admin\Desktop\Re
□ PBORT_menu_tutorial (C:\Users\admin\Desktop\RecoNode\TF
□ PBORT_menu_tutorial (C:\Users\admin\Desktop\RecoNode\TF

□ Copy projects into workspace

Working sets

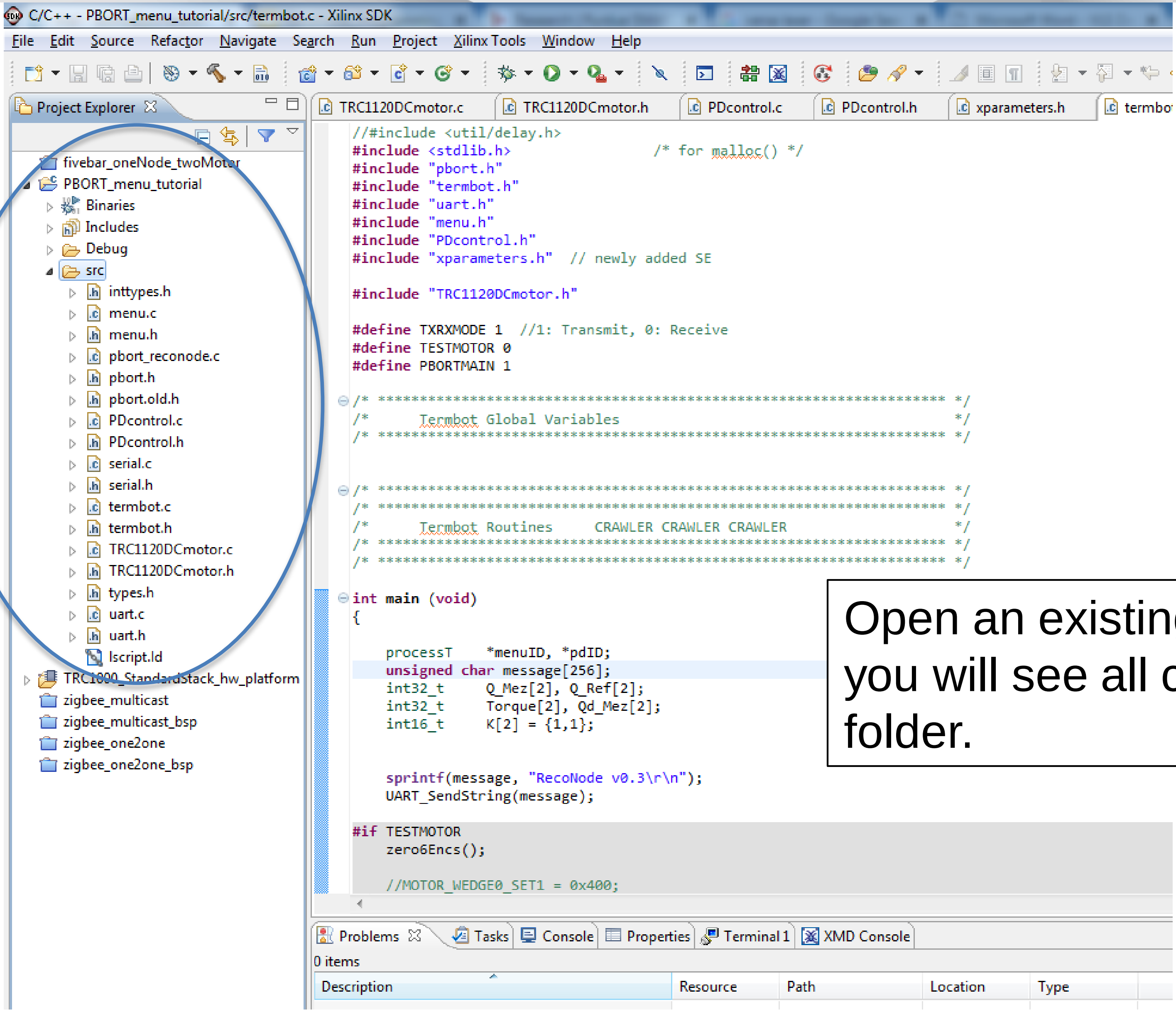
□ Add project to working sets

Working sets:

Select the project folders that you want to import.

For this tutorial, we want *PBORT_menu_tutorial* folder.

2. A. b. Xilinx SDK



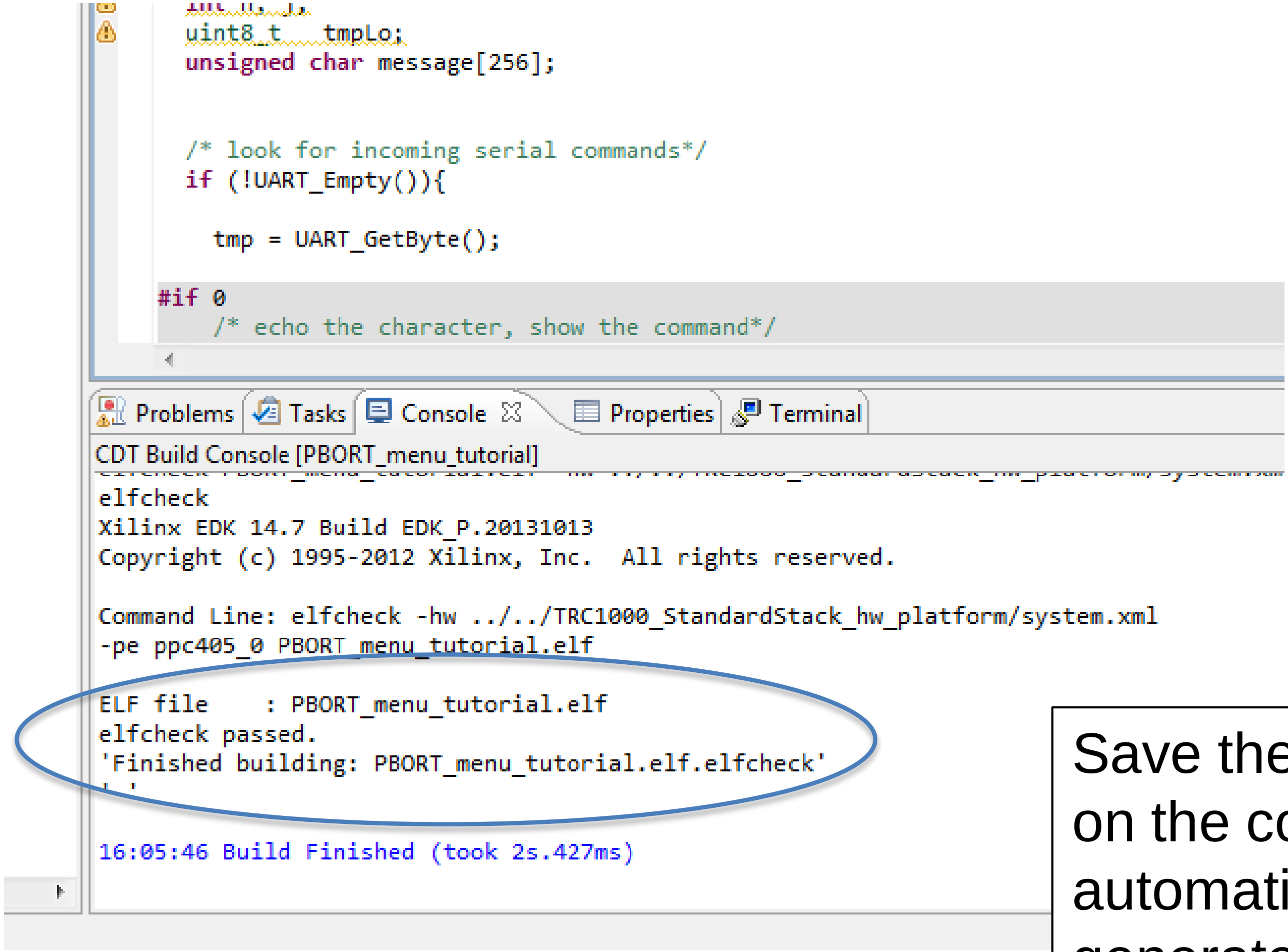
- fivebar_oneNode_twoMotor
- PBORT_menu_tutorial
  - Binaries
  - Includes
  - Debug
  - src
    - inttypes.h
    - menu.c
    - menu.h
    - pbort_reconode.c
    - pbort.h
    - pbort.old.h
    - PDcontrol.c
    - PDcontrol.h
    - serial.c
    - serial.h
    - termbot.c
    - termbot.h
    - TRC1120DCmotor.c
    - TRC1120DCmotor.h
    - types.h
    - uart.c
    - uart.h
    - lscript.ld
- TRC1000_StandardStack_hw_platform
- zigbee_multicast
- zigbee_multicast_bsp
- zigbee_one2one
- zigbee_one2one_bsp

```
//#include <util/delay.h>
#include <stdlib.h>              /* for malloc() */
#include "pbort.h"
#include "termbot.h"
#include "uart.h"
#include "menu.h"
#include "PDcontrol.h"
#include "xparameters.h"  // newly added SE

#include "TRC1120DCmotor.h"

#define TXRXMODE 1  //1: Transmit, 0: Receive
#define TESTMOTOR 0
#define PBORTMAIN 1

/* ******************************************************************** */
/*      Termbot Global Variables                                        */
/* ******************************************************************** */


/* ******************************************************************** */
/* ******************************************************************** */
/*      Termbot Routines     CRAWLER CRAWLER CRAWLER                    */
/* ******************************************************************** */
/* ******************************************************************** */

int main (void)
{

    processT     *menuID, *pdID;
    unsigned char message[256];
    int32_t      Q_Mez[2], Q_Ref[2];
    int32_t      Torque[2], Qd_Mez[2];
    int16_t      K[2] = {1,1};


    sprintf(message, "RecoNode v0.3\r\n");
    UART_SendString(message);

#if TESTMOTOR
    zero6Encs();

    //MOTOR_WEDGE0_SET1 = 0x400;
```

Open an existing project and you will see all c files in *src* folder.

c. Program the Hardware

2. A. c. Program the Hardware



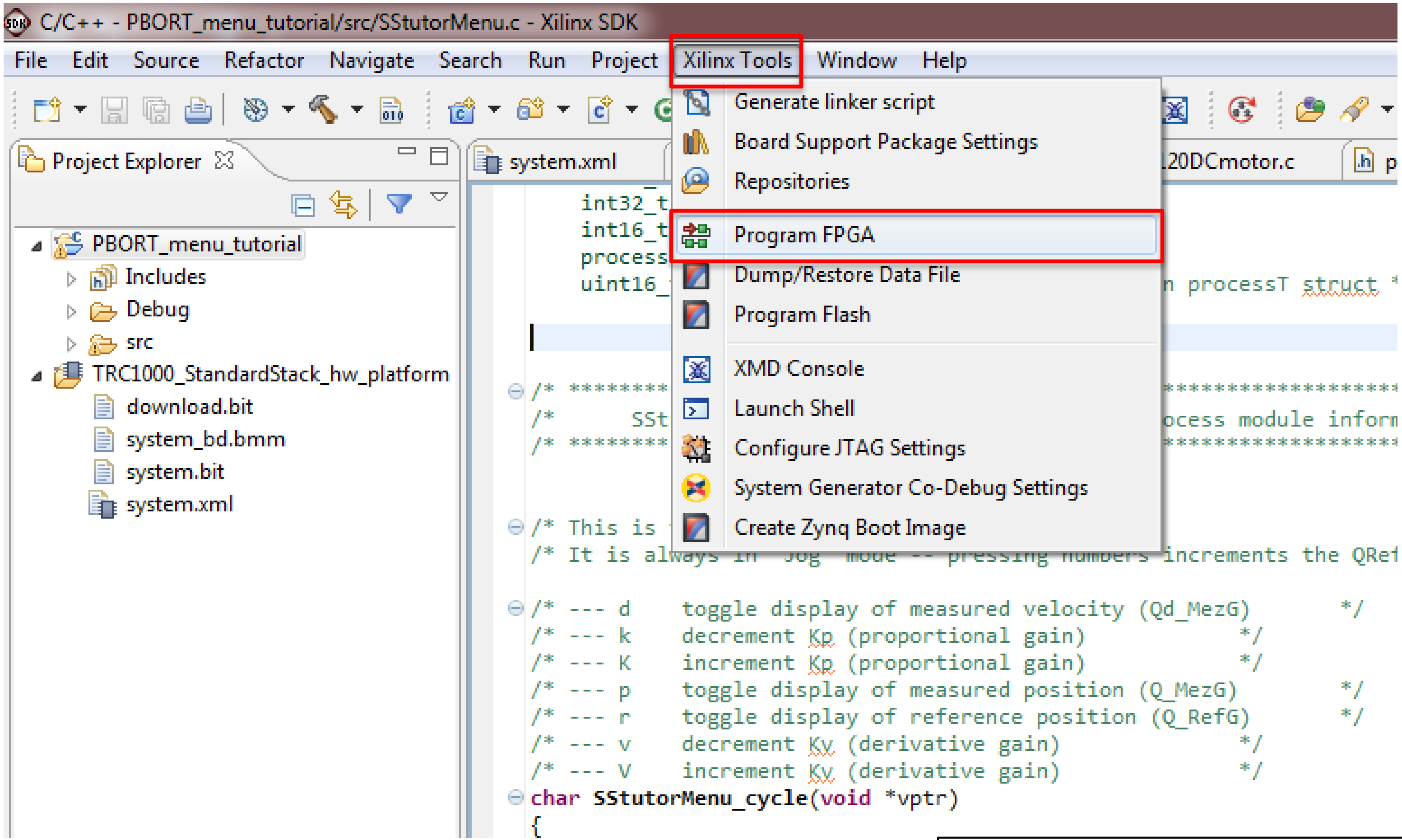
```
  uint8_t   tmpLo;
  unsigned char message[256];


  /* look for incoming serial commands*/
  if (!UART_Empty()){

    tmp = UART_GetByte();

#if 0
    /* echo the character, show the command*/
```

CDT Build Console [PBORT_menu_tutorial]

```
elfcheck
Xilinx EDK 14.7 Build EDK_P.20131013
Copyright (c) 1995-2012 Xilinx, Inc.  All rights reserved.

Command Line: elfcheck -hw ../../TRC1000_StandardStack_hw_platform/system.xml
-pe ppc405_0 PBORT_menu_tutorial.elf

ELF file     : PBORT_menu_tutorial.elf
elfcheck passed.
'Finished building: PBORT_menu_tutorial.elf.elfcheck'
' '

16:05:46 Build Finished (took 2s.427ms)
```

Save the changes you made on the code, xilinx will automatically compile and generate elf file.

2. A. c. Program the Hardware

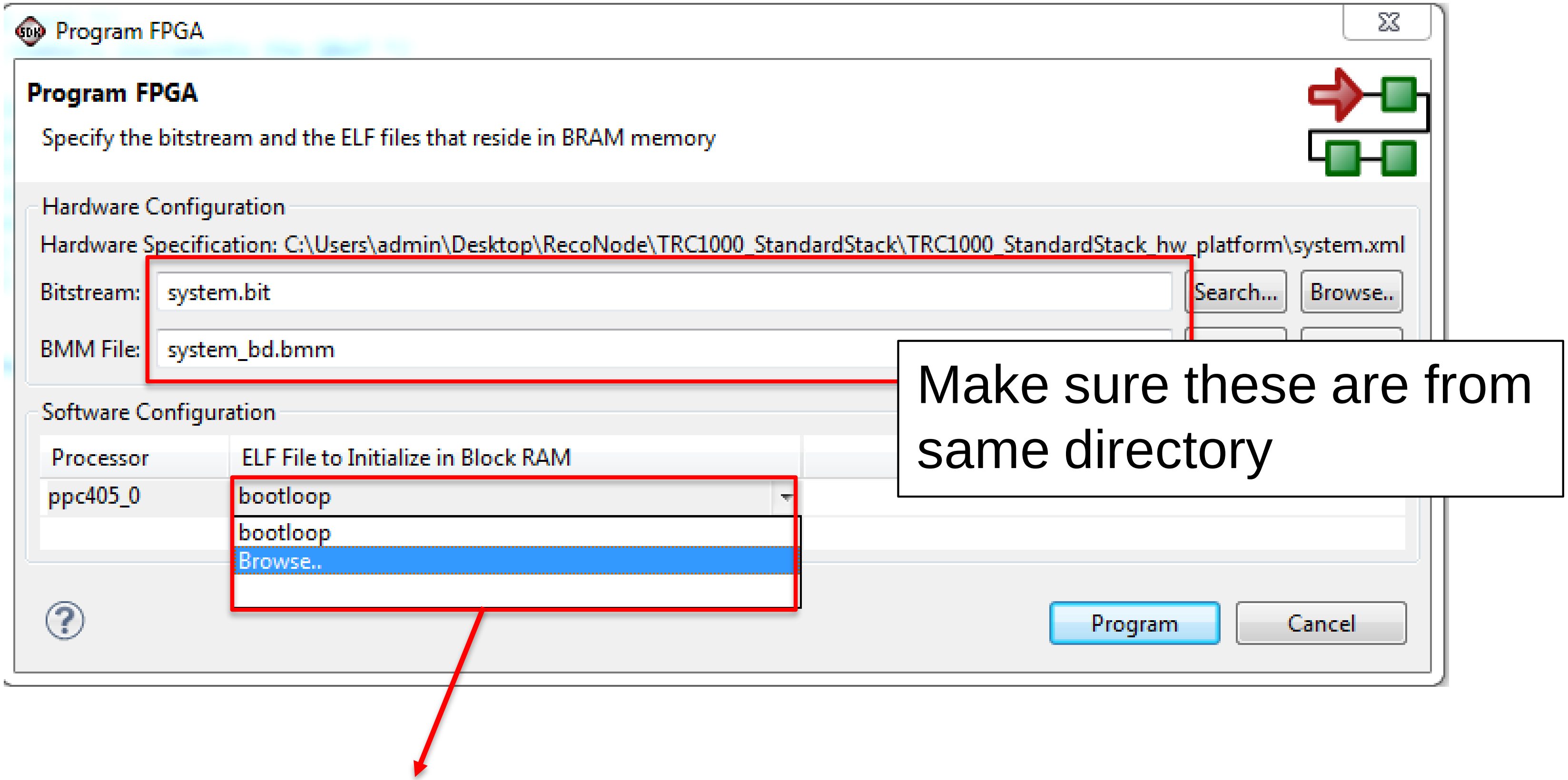


```
    int32_t
    int16_t
    process
    uint16_                                          n processT struct *

/* ********                                          *******************
/*       SSt                                         ocess module inform
/* ********                                          *******************

/* This is
/* It is always in  Jog  mode -- pressing numbers increments the QRef

/* --- d    toggle display of measured velocity (Qd_MezG)       */
/* --- k    decrement Kp (proportional gain)            */
/* --- K    increment Kp (proportional gain)            */
/* --- p    toggle display of measured position (Q_MezG)        */
/* --- r    toggle display of reference position (Q_RefG)       */
/* --- v    decrement Kv (derivative gain)              */
/* --- V    increment Kv (derivative gain)              */
char SStutorMenu_cycle(void *vptr)
{
```

Go to

Xilinx Tools / Program FPGA

2. A. c. Program the Hardware



Program FPGA

Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

Hardware Specification: C:\Users\admin\Desktop\RecoNode\TRC1000_StandardStack\TRC1000_StandardStack_hw_platform\system.xml

Bitstream: system.bit

BMM File: system_bd.bmm

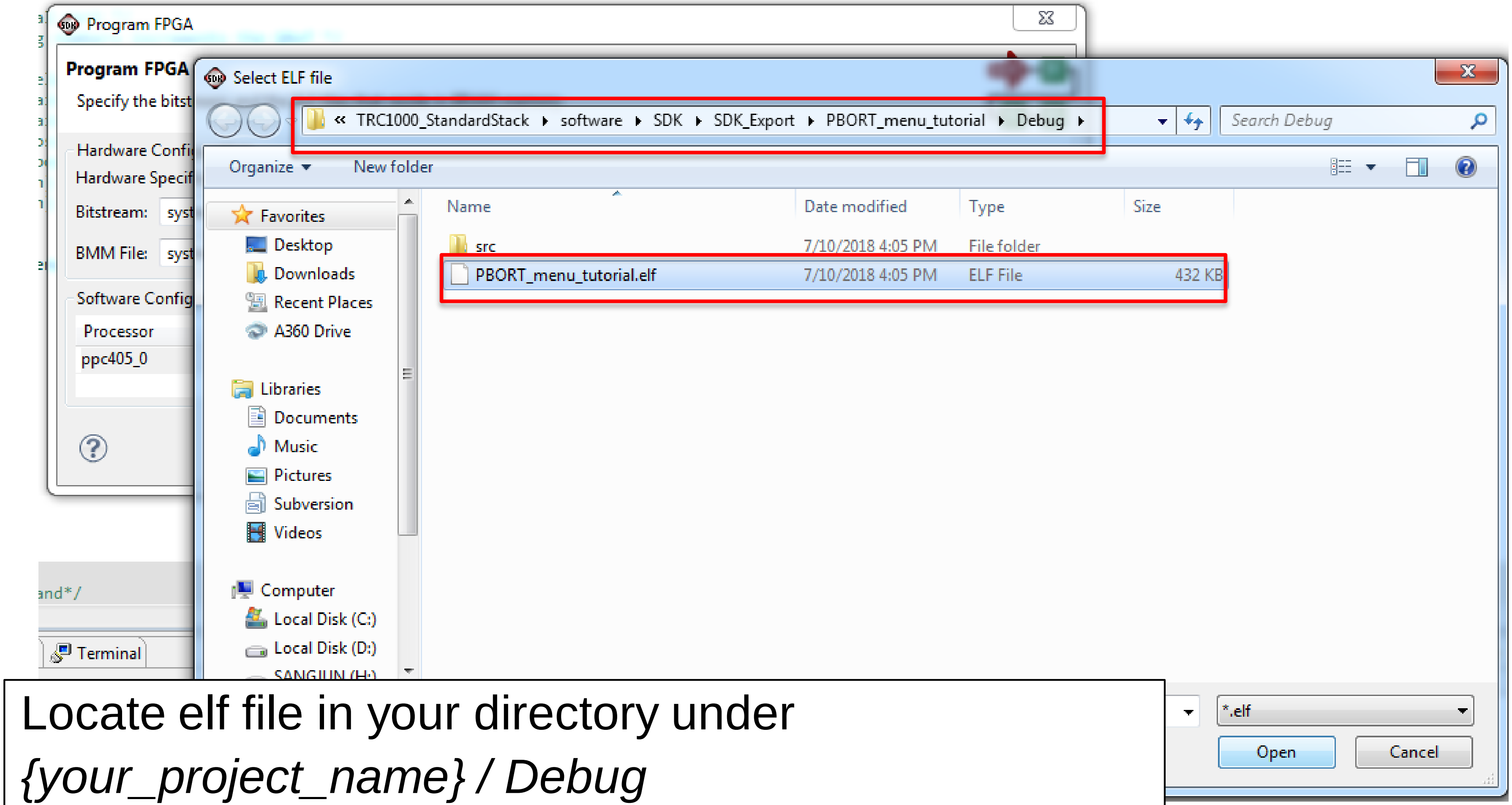
Software Configuration

Processor	ELF File to Initialize in Block RAM
ppc405_0	bootloop

Make sure these are from same directory

Browse for elf file created for your project.

2. A. c. Program the Hardware



Name	Date modified	Type	Size
src	7/10/2018 4:05 PM	File folder	
PBORT_menu_tutorial.elf	7/10/2018 4:05 PM	ELF File	432 KB

Locate elf file in your directory under *{your_project_name} / Debug*

Open elf file for programming your hardware.

2. B. Modify the Project

a. Add a New Software Module

b. Add a New IP Core (Add / Change Wedges)

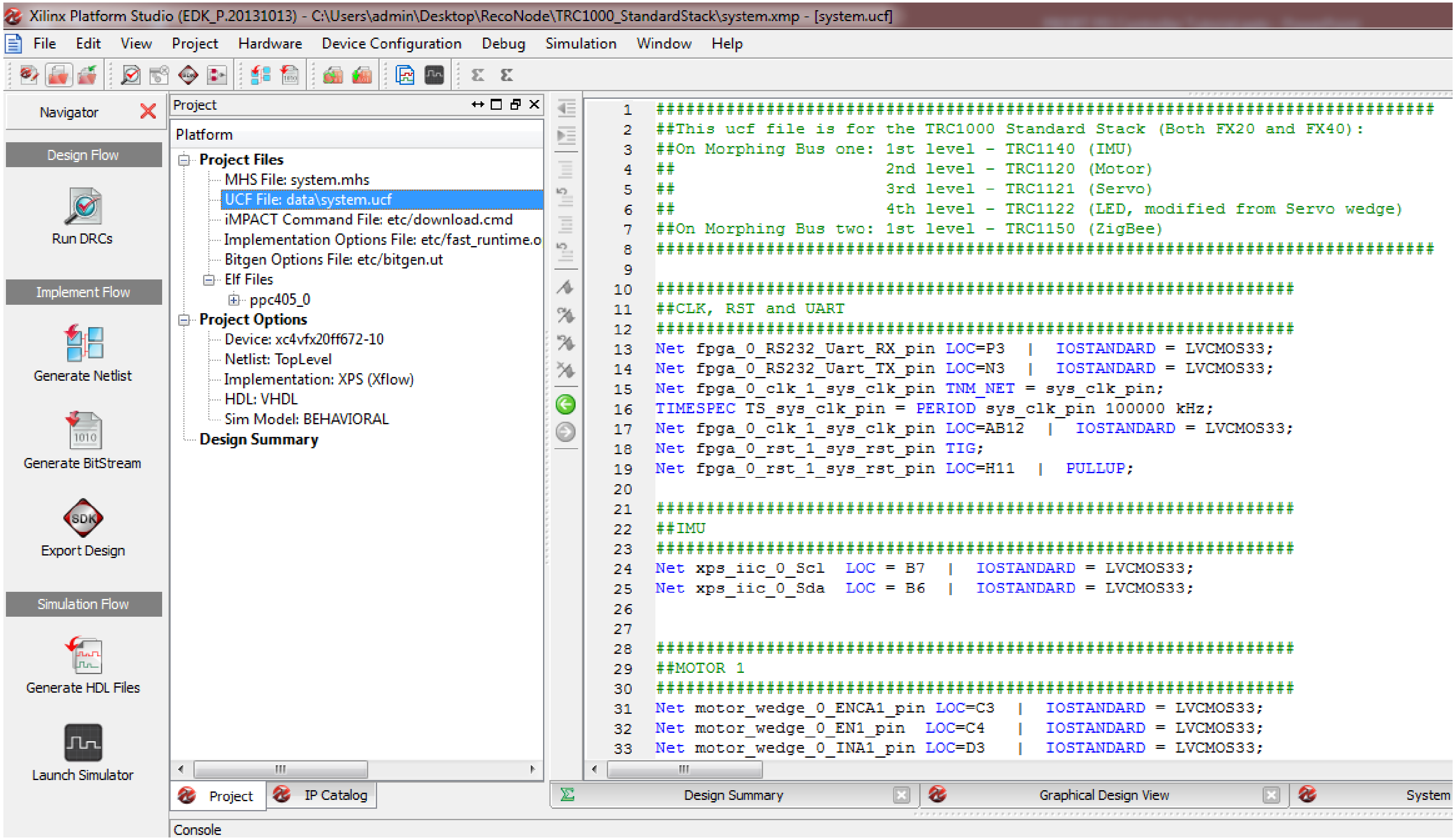
c. Create a New IP Core

a. Add a New Software Module

b. Add a New IP Core (Add / Change Wedges)

By adding IP core, you can add more wedges and customize your stack.

2. B. b. Add a New IP Core - Update this

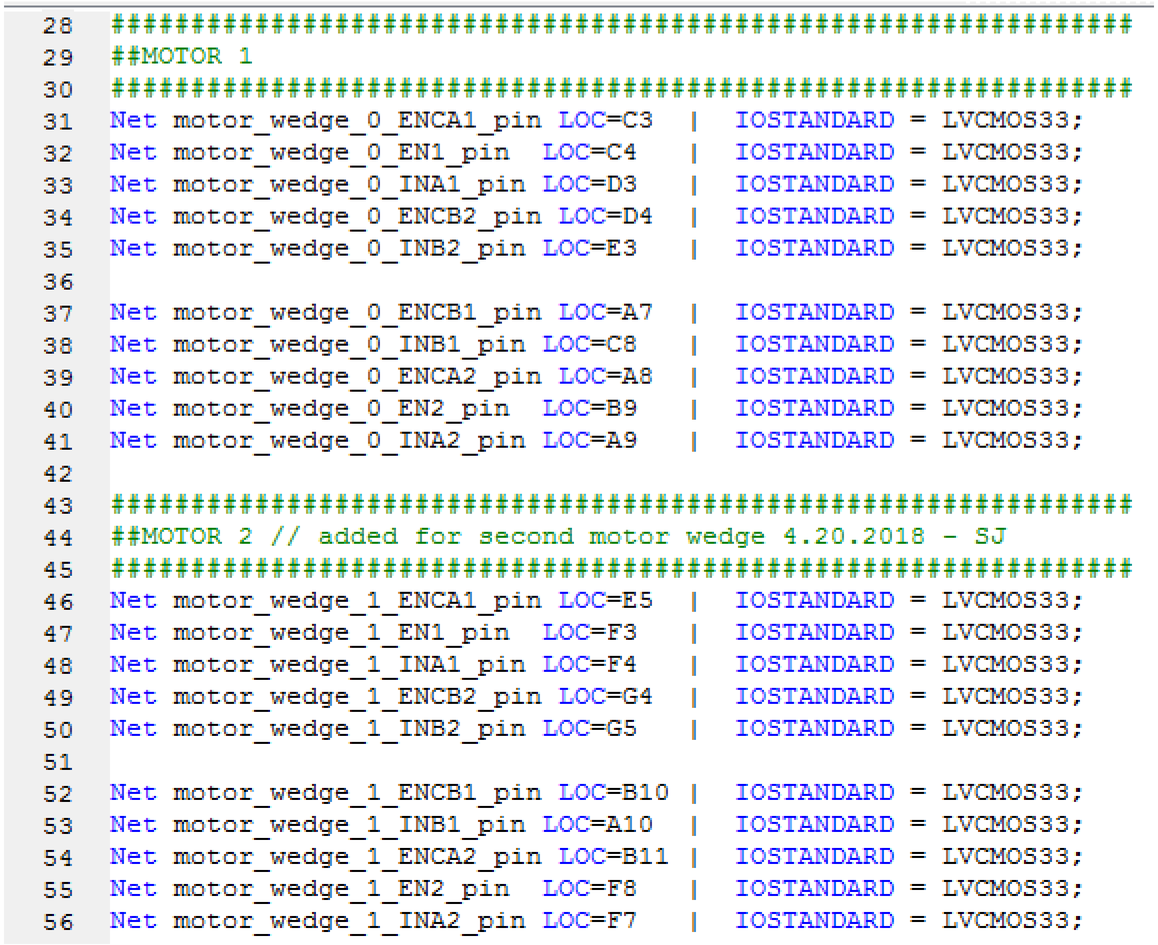


- Project Files
  - MHS File: system.mhs
  - UCF File: data\system.ucf
  - iMPACT Command File: etc/download.cmd
  - Implementation Options File: etc/fast_runtime.o
  - Bitgen Options File: etc/bitgen.ut
  - Elf Files
    - ppc405_0
- Project Options
  - Device: xc4vfx20ff672-10
  - Netlist: TopLevel
  - Implementation: XPS (Xflow)
  - HDL: VHDL
  - Sim Model: BEHAVIORAL
- Design Summary

```
##############################################################################
##This ucf file is for the TRC1000 Standard Stack (Both FX20 and FX40):
##On Morphing Bus one: 1st level - TRC1140 (IMU)
##                     2nd level - TRC1120 (Motor)
##                     3rd level - TRC1121 (Servo)
##                     4th level - TRC1122 (LED, modified from Servo wedge)
##On Morphing Bus two: 1st level - TRC1150 (ZigBee)
##############################################################################

################################################################
##CLK, RST and UART
################################################################
Net fpga_0_RS232_Uart_RX_pin LOC=P3  |  IOSTANDARD = LVCMOS33;
Net fpga_0_RS232_Uart_TX_pin LOC=N3  |  IOSTANDARD = LVCMOS33;
Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100000 kHz;
Net fpga_0_clk_1_sys_clk_pin LOC=AB12  |  IOSTANDARD = LVCMOS33;
Net fpga_0_rst_1_sys_rst_pin TIG;
Net fpga_0_rst_1_sys_rst_pin LOC=H11  |  PULLUP;

################################################################
##IMU
################################################################
Net xps_iic_0_Scl  LOC = B7  |  IOSTANDARD = LVCMOS33;
Net xps_iic_0_Sda  LOC = B6  |  IOSTANDARD = LVCMOS33;


################################################################
##MOTOR 1
################################################################
Net motor_wedge_0_ENCA1_pin LOC=C3   |  IOSTANDARD = LVCMOS33;
Net motor_wedge_0_EN1_pin  LOC=C4    |  IOSTANDARD = LVCMOS33;
Net motor_wedge_0_INA1_pin LOC=D3    |  IOSTANDARD = LVCMOS33;
```

PURDUE
POLYTECHNIC

2. B. b. Add a New IP Core - Update this

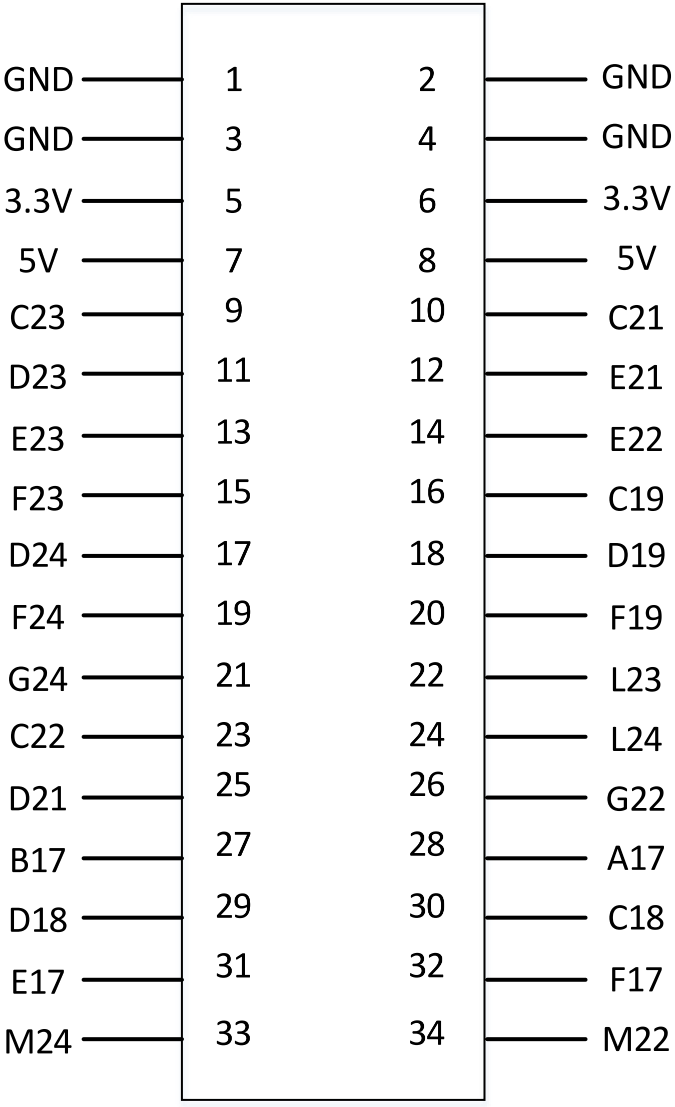


```
################################################################
##MOTOR 1
################################################################
Net motor_wedge_0_ENCA1_pin LOC=C3  |  IOSTANDARD = LVCMOS33;
Net motor_wedge_0_EN1_pin  LOC=C4   |  IOSTANDARD = LVCMOS33;
Net motor_wedge_0_INA1_pin LOC=D3   |  IOSTANDARD = LVCMOS33;
Net motor_wedge_0_ENCB2_pin LOC=D4  |  IOSTANDARD = LVCMOS33;
Net motor_wedge_0_INB2_pin LOC=E3   |  IOSTANDARD = LVCMOS33;

Net motor_wedge_0_ENCB1_pin LOC=A7  |  IOSTANDARD = LVCMOS33;
Net motor_wedge_0_INB1_pin LOC=C8   |  IOSTANDARD = LVCMOS33;
Net motor_wedge_0_ENCA2_pin LOC=A8  |  IOSTANDARD = LVCMOS33;
Net motor_wedge_0_EN2_pin  LOC=B9   |  IOSTANDARD = LVCMOS33;
Net motor_wedge_0_INA2_pin LOC=A9   |  IOSTANDARD = LVCMOS33;

################################################################
##MOTOR 2 // added for second motor wedge 4.20.2018 - SJ
################################################################
Net motor_wedge_1_ENCA1_pin LOC=E5  |  IOSTANDARD = LVCMOS33;
Net motor_wedge_1_EN1_pin  LOC=F3   |  IOSTANDARD = LVCMOS33;
Net motor_wedge_1_INA1_pin LOC=F4   |  IOSTANDARD = LVCMOS33;
Net motor_wedge_1_ENCB2_pin LOC=G4  |  IOSTANDARD = LVCMOS33;
Net motor_wedge_1_INB2_pin LOC=G5   |  IOSTANDARD = LVCMOS33;

Net motor_wedge_1_ENCB1_pin LOC=B10 |  IOSTANDARD = LVCMOS33;
Net motor_wedge_1_INB1_pin LOC=A10  |  IOSTANDARD = LVCMOS33;
Net motor_wedge_1_ENCA2_pin LOC=B11 |  IOSTANDARD = LVCMOS33;
Net motor_wedge_1_EN2_pin  LOC=F8   |  IOSTANDARD = LVCMOS33;
Net motor_wedge_1_INA2_pin LOC=F7   |  IOSTANDARD = LVCMOS33;
```

Morphing Bus 1 on TRC1000

GND	1	2	GND
GND	3	4	GND
3.3V	5	6	3.3V
5V	7	8	5V
B7	9	10	B6
A7	11	12	C3
C8	13	14	C4
A8	15	16	D3
B9	17	18	D4
A9	19	20	E3
B10	21	22	E5
A10	23	24	F3
B11	25	26	F4
F8	27	28	G4
F7	29	30	G5
G9	31	32	H3
G7	33	34	H4
H7	35	36	J3
H6	37	38	J4
J9	39	40	K3
J5	41	42	K6
K8	43	44	L5
K7	45	46	L7
L10	47	48	M5
L9	49	50	M6

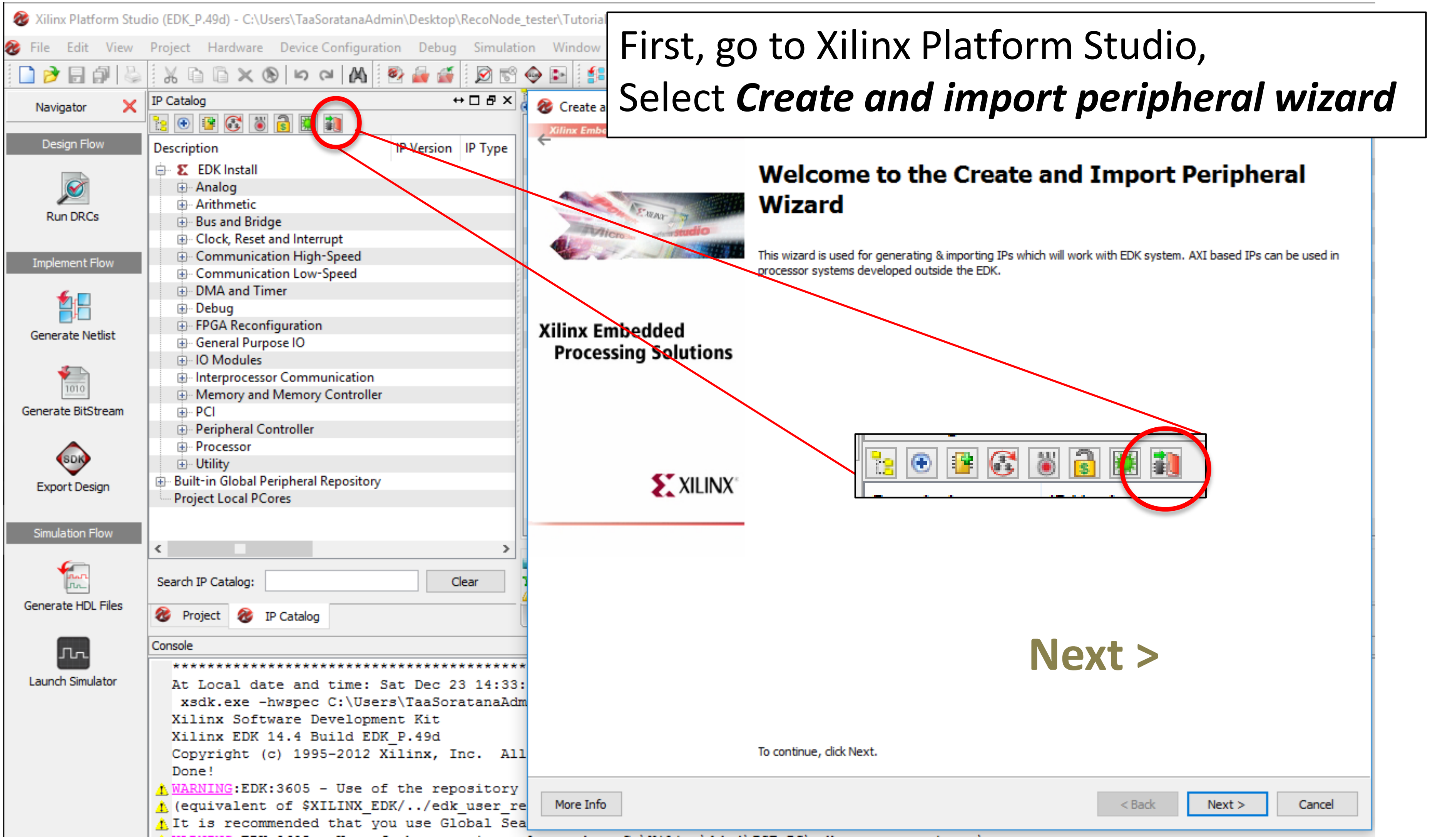
Morphing Bus 2 on TRC1000



GND	1	2	GND
GND	3	4	GND
3.3V	5	6	3.3V
5V	7	8	5V
C23	9	10	C21
D23	11	12	E21
E23	13	14	E22
F23	15	16	C19
D24	17	18	D19
F24	19	20	F19
G24	21	22	L23
C22	23	24	L24
D21	25	26	G22
B17	27	28	A17
D18	29	30	C18
E17	31	32	F17
M24	33	34	M22

c. Create a New IP Core

2. B. c. Create a New IP Core



First, go to Xilinx Platform Studio,
Select ***Create and import peripheral wizard***

Xilinx Platform Studio (EDK_P.49d) - C:\Users\TaaSoratanaAdmin\Desktop\RecoNode_tester\Tutorial

```
*****************************************
At Local date and time: Sat Dec 23 14:33:
 xsdk.exe -hwspec C:\Users\TaaSoratanaAdm
Xilinx Software Development Kit
Xilinx EDK 14.4 Build EDK_P.49d
Copyright (c) 1995-2012 Xilinx, Inc.  All
Done!
WARNING:EDK:3605 - Use of the repository
(equivalent of $XILINX_EDK/../edk_user_re
It is recommended that you use Global Sea
```

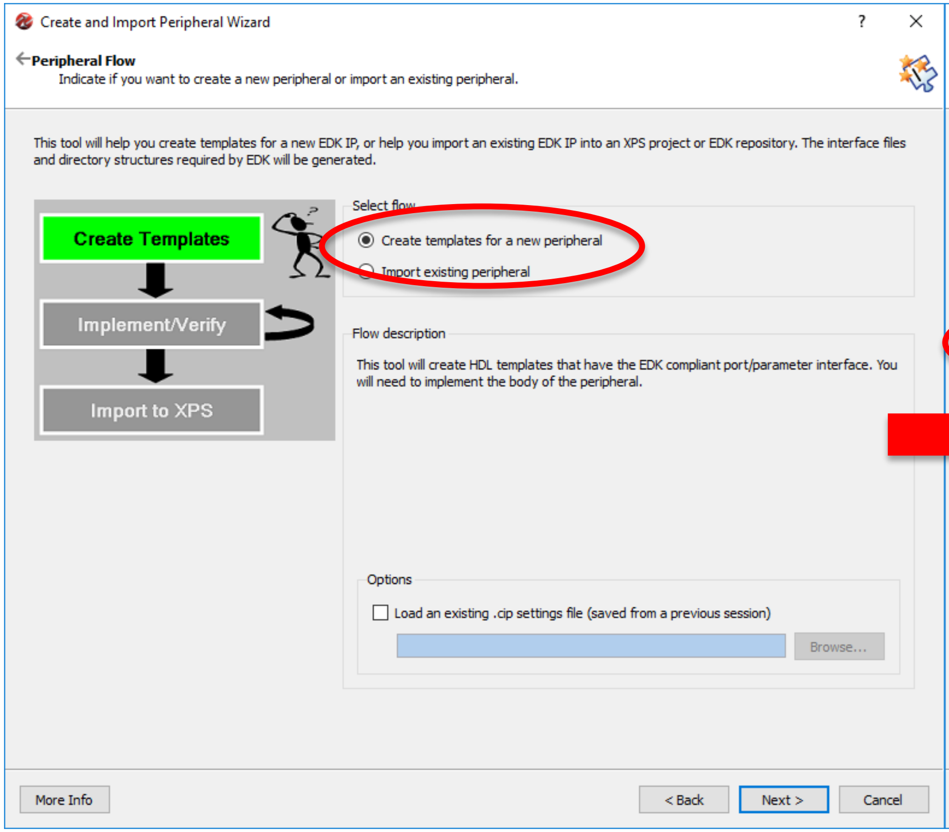
Welcome to the Create and Import Peripheral Wizard

This wizard is used for generating & importing IPs which will work with EDK system. AXI based IPs can be used in processor systems developed outside the EDK.

To continue, click Next.

Next >

2. B. c. Create a New IP Core



Create and Import Peripheral Wizard

Peripheral Flow

Indicate if you want to create a new peripheral or import an existing peripheral.

This tool will help you create templates for a new EDK IP, or help you import an existing EDK IP into an XPS project or EDK repository. The interface files and directory structures required by EDK will be generated.

Select flow

● Create templates for a new peripheral

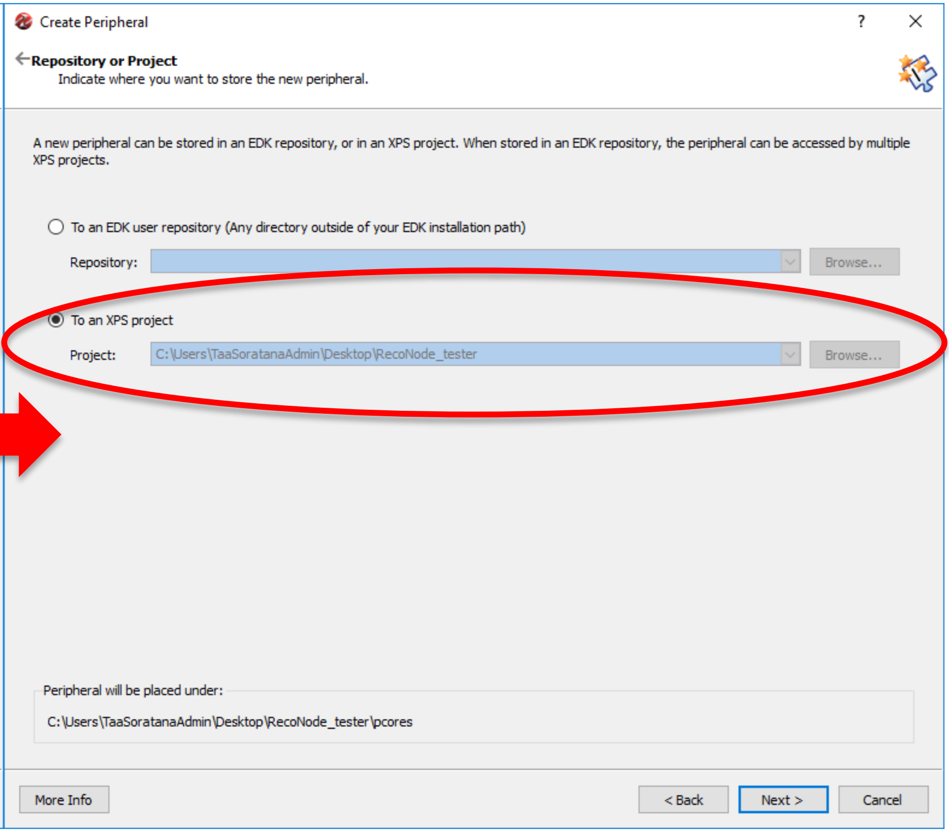
○ Import existing peripheral

Flow description

This tool will create HDL templates that have the EDK compliant port/parameter interface. You will need to implement the body of the peripheral.

Options

□ Load an existing .cip settings file (saved from a previous session)



Create Peripheral

Repository or Project

Indicate where you want to store the new peripheral.

A new peripheral can be stored in an EDK repository, or in an XPS project. When stored in an EDK repository, the peripheral can be accessed by multiple XPS projects.

○ To an EDK user repository (Any directory outside of your EDK installation path)

Repository:

● To an XPS project

Project: C:\Users\TaaSoratanaAdmin\Desktop\RecoNode_tester

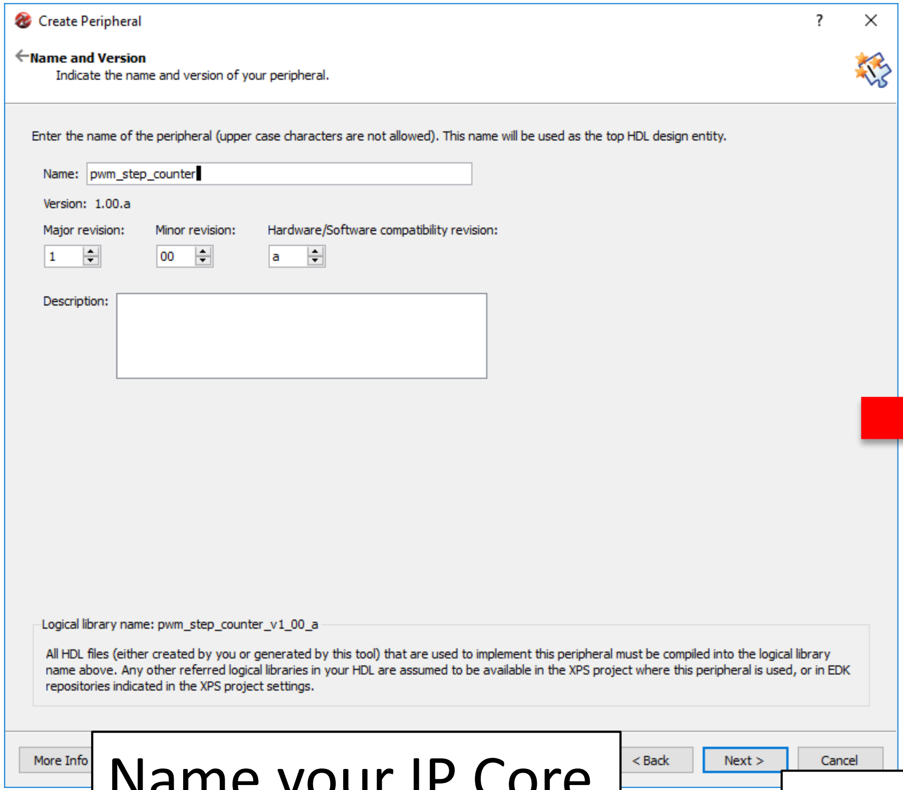
Peripheral will be placed under:

C:\Users\TaaSoratanaAdmin\Desktop\RecoNode_tester\pcores

Select ***Create template for a new peripheral***

Select ***Export to XPS Project***

2. B. c. Create a New IP Core



Create Peripheral

Name and Version

Indicate the name and version of your peripheral.

Enter the name of the peripheral (upper case characters are not allowed). This name will be used as the top HDL design entity.

Name: pwm_step_counter

Version: 1.00.a

Major revision: 1

Minor revision: 00

Hardware/Software compatibility revision: a

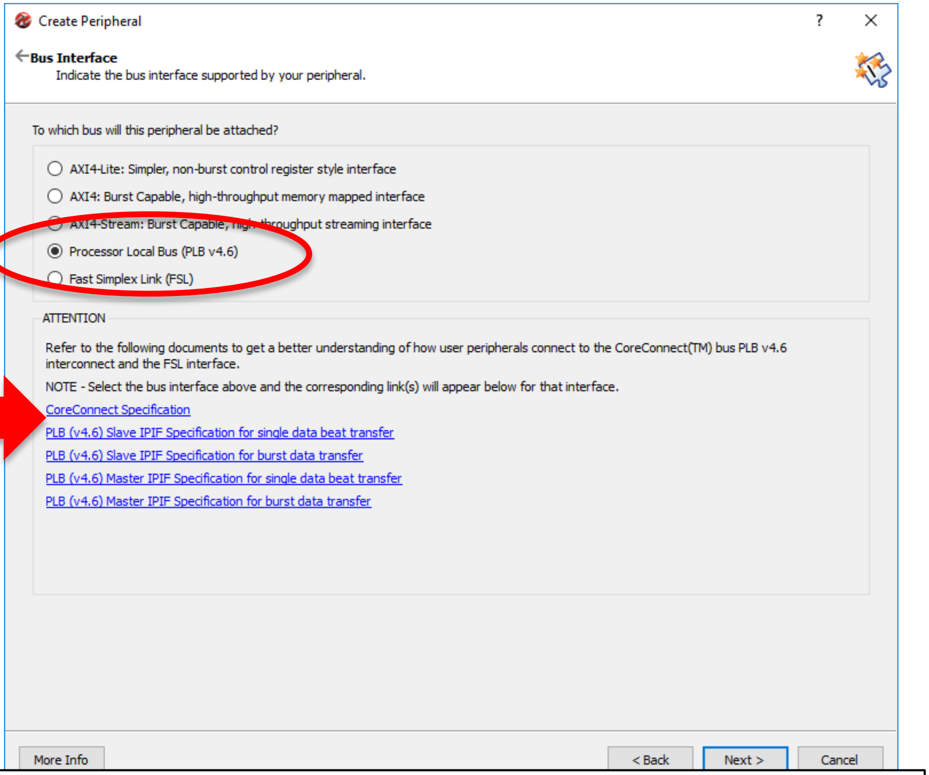
Description:

Logical library name: pwm_step_counter_v1_00_a

All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compiled into the logical library name above. Any other referred logical libraries in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.

Name your IP Core

In this example, we are creating a PWM step counter as a new IP core.



Create Peripheral

Bus Interface

Indicate the bus interface supported by your peripheral.

To which bus will this peripheral be attached?

○ AXI4-Lite: Simpler, non-burst control register style interface
○ AXI4: Burst Capable, high-throughput memory mapped interface
○ AXI4-Stream: Burst Capable, high-throughput streaming interface
● Processor Local Bus (PLB v4.6)
○ Fast Simplex Link (FSL)

ATTENTION

Refer to the following documents to get a better understanding of how user peripherals connect to the CoreConnect(TM) bus PLB v4.6 interconnect and the FSL interface.

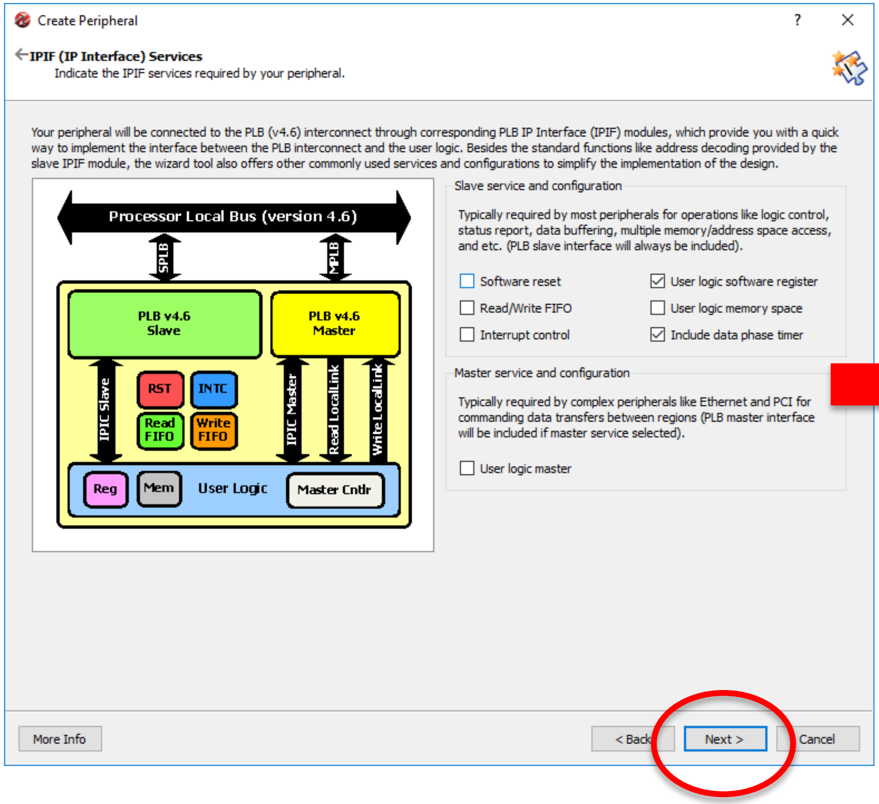
NOTE - Select the bus interface above and the corresponding link(s) will appear below for that interface.

CoreConnect Specification
PLB (v4.6) Slave IPIF Specification for single data beat transfer
PLB (v4.6) Slave IPIF Specification for burst data transfer
PLB (v4.6) Master IPIF Specification for single data beat transfer
PLB (v4.6) Master IPIF Specification for burst data transfer

Select Processor Local Bus

* Unless you use AXI in your XPS Project

2. B. c. Create a New IP Core



Create Peripheral

IPIF (IP Interface) Services

Indicate the IPIF services required by your peripheral.

Your peripheral will be connected to the PLB (v4.6) interconnect through corresponding PLB IP Interface (IPIF) modules, which provide you with a quick way to implement the interface between the PLB interconnect and the user logic. Besides the standard functions like address decoding provided by the slave IPIF module, the wizard tool also offers other commonly used services and configurations to simplify the implementation of the design.

Slave service and configuration

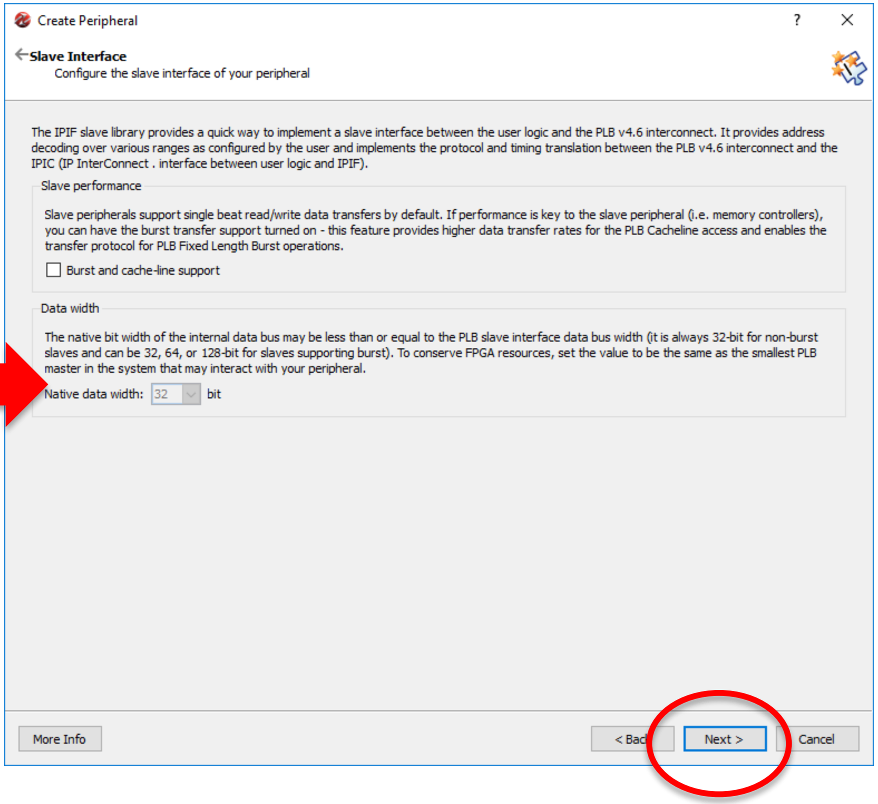
Typically required by most peripherals for operations like logic control, status report, data buffering, multiple memory/address space access, and etc. (PLB slave interface will always be included).

- □ Software reset
- □ Read/Write FIFO
- □ Interrupt control
- ☑ User logic software register
- □ User logic memory space
- ☑ Include data phase timer

Master service and configuration

Typically required by complex peripherals like Ethernet and PCI for commanding data transfers between regions (PLB master interface will be included if master service selected).

- □ User logic master



Create Peripheral

Slave Interface

Configure the slave interface of your peripheral

The IPIF slave library provides a quick way to implement a slave interface between the user logic and the PLB v4.6 interconnect. It provides address decoding over various ranges as configured by the user and implements the protocol and timing translation between the PLB v4.6 interconnect and the IPIC (IP InterConnect . interface between user logic and IPIF).

Slave performance

Slave peripherals support single beat read/write data transfers by default. If performance is key to the slave peripheral (i.e. memory controllers), you can have the burst transfer support turned on - this feature provides higher data transfer rates for the PLB Cacheline access and enables the transfer protocol for PLB Fixed Length Burst operations.

- □ Burst and cache-line support

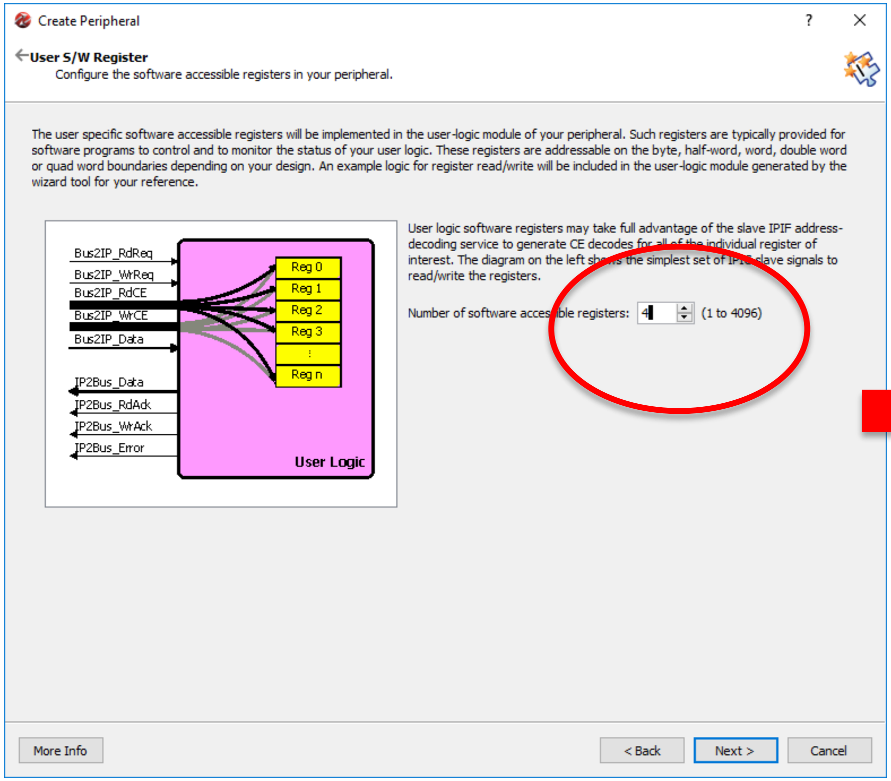
Data width

The native bit width of the internal data bus may be less than or equal to the PLB slave interface data bus width (it is always 32-bit for non-burst slaves and can be 32, 64, or 128-bit for slaves supporting burst). To conserve FPGA resources, set the value to be the same as the smallest PLB master in the system that may interact with your peripheral.

Native data width: 32 bit

Proceed to next step

2. B. c. Create a New IP Core



Create Peripheral

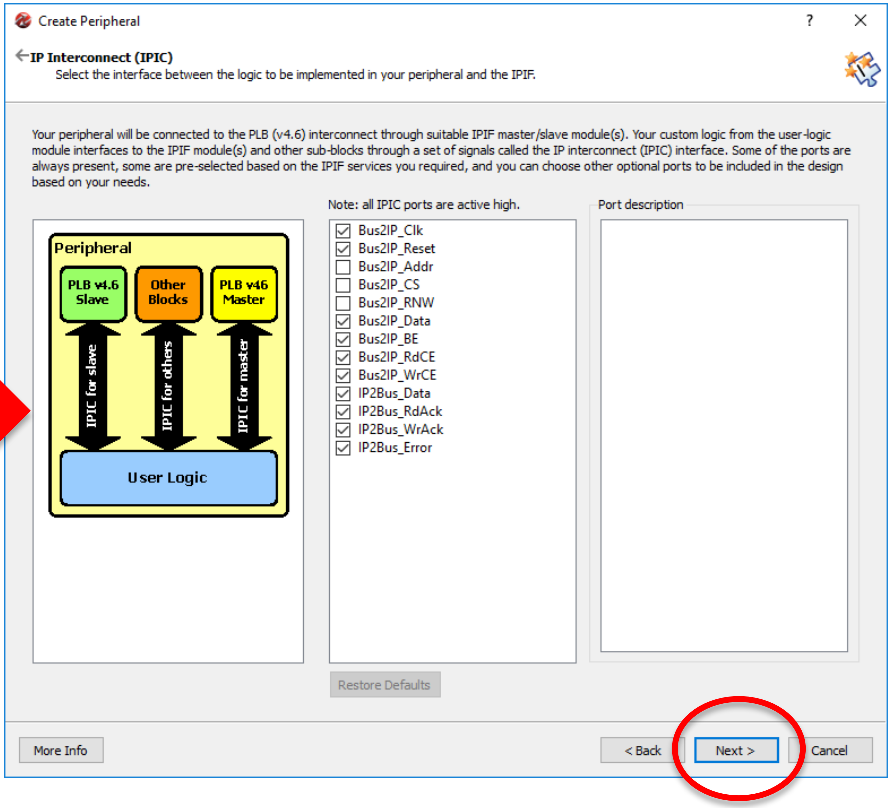
User S/W Register

Configure the software accessible registers in your peripheral.

The user specific software accessible registers will be implemented in the user-logic module of your peripheral. Such registers are typically provided for software programs to control and to monitor the status of your user logic. These registers are addressable on the byte, half-word, word, double word or quad word boundaries depending on your design. An example logic for register read/write will be included in the user-logic module generated by the wizard tool for your reference.

User logic software registers may take full advantage of the slave IPIF address-decoding service to generate CE decodes for all of the individual register of interest. The diagram on the left shows the simplest set of IPIF slave signals to read/write the registers.

Number of software accessible registers: 4 (1 to 4096)



Create Peripheral

IP Interconnect (IPIC)

Select the interface between the logic to be implemented in your peripheral and the IPIF.

Your peripheral will be connected to the PLB (v4.6) interconnect through suitable IPIF master/slave module(s). Your custom logic from the user-logic module interfaces to the IPIF module(s) and other sub-blocks through a set of signals called the IP interconnect (IPIC) interface. Some of the ports are always present, some are pre-selected based on the IPIF services you required, and you can choose other optional ports to be included in the design based on your needs.

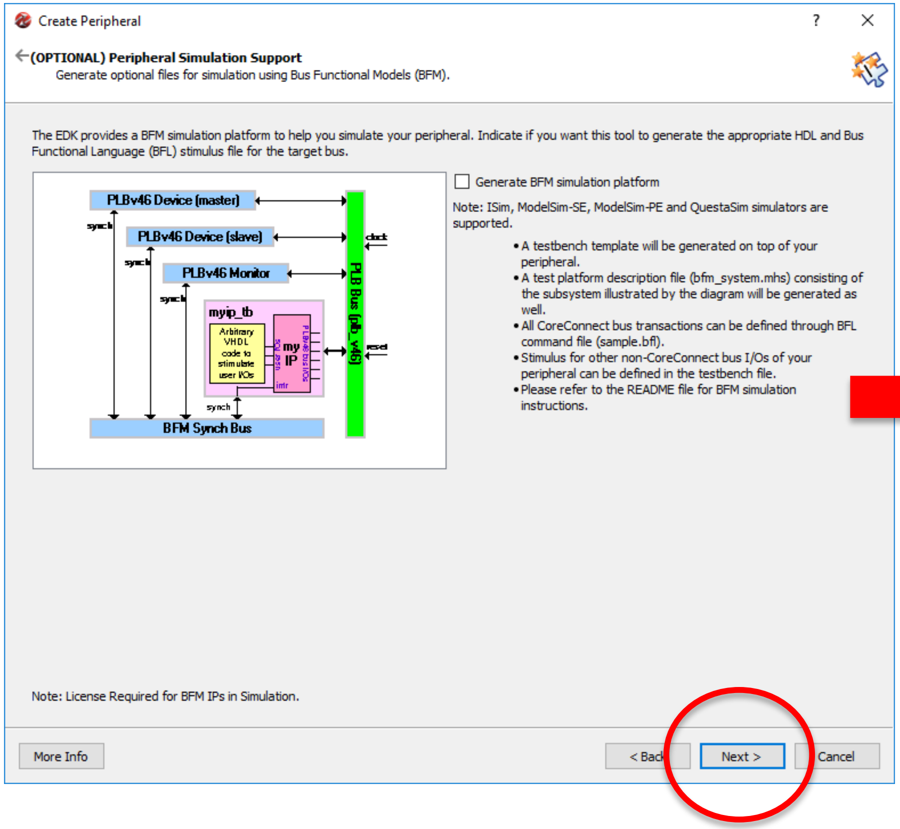
Note: all IPIC ports are active high.

- ☑ Bus2IP_Clk
- ☑ Bus2IP_Reset
- □ Bus2IP_Addr
- □ Bus2IP_CS
- □ Bus2IP_RNW
- ☑ Bus2IP_Data
- ☑ Bus2IP_BE
- ☑ Bus2IP_RdCE
- ☑ Bus2IP_WrCE
- ☑ IP2Bus_Data
- ☑ IP2Bus_RdAck
- ☑ IP2Bus_WrAck
- ☑ IP2Bus_Error

Port description

Select **4 registers**

2. B. c. Create a New IP Core



Create Peripheral

(OPTIONAL) Peripheral Simulation Support

Generate optional files for simulation using Bus Functional Models (BFM).

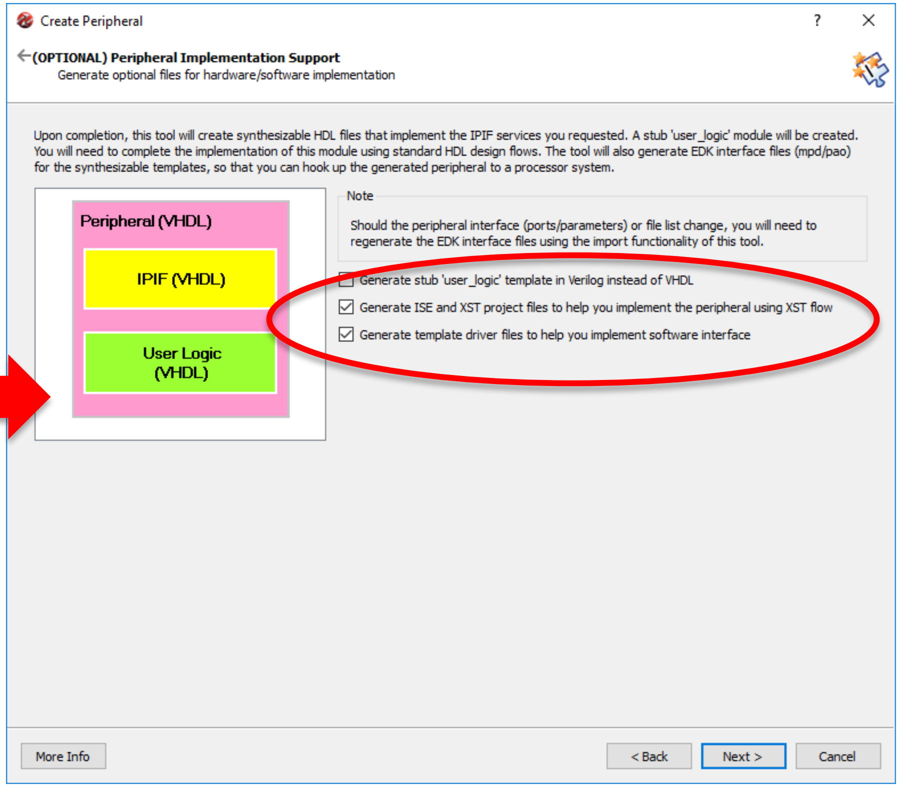
The EDK provides a BFM simulation platform to help you simulate your peripheral. Indicate if you want this tool to generate the appropriate HDL and Bus Functional Language (BFL) stimulus file for the target bus.

☐ Generate BFM simulation platform

Note: ISim, ModelSim-SE, ModelSim-PE and QuestaSim simulators are supported.

- A testbench template will be generated on top of your peripheral.
- A test platform description file (bfm_system.mhs) consisting of the subsystem illustrated by the diagram will be generated as well.
- All CoreConnect bus transactions can be defined through BFL command file (sample.bfl).
- Stimulus for other non-CoreConnect bus I/Os of your peripheral can be defined in the testbench file.
- Please refer to the README file for BFM simulation instructions.

Note: License Required for BFM IPs in Simulation.



Create Peripheral

(OPTIONAL) Peripheral Implementation Support

Generate optional files for hardware/software implementation

Upon completion, this tool will create synthesizable HDL files that implement the IPIF services you requested. A stub 'user_logic' module will be created. You will need to complete the implementation of this module using standard HDL design flows. The tool will also generate EDK interface files (mpd/pao) for the synthesizable templates, so that you can hook up the generated peripheral to a processor system.

Note

Should the peripheral interface (ports/parameters) or file list change, you will need to regenerate the EDK interface files using the import functionality of this tool.

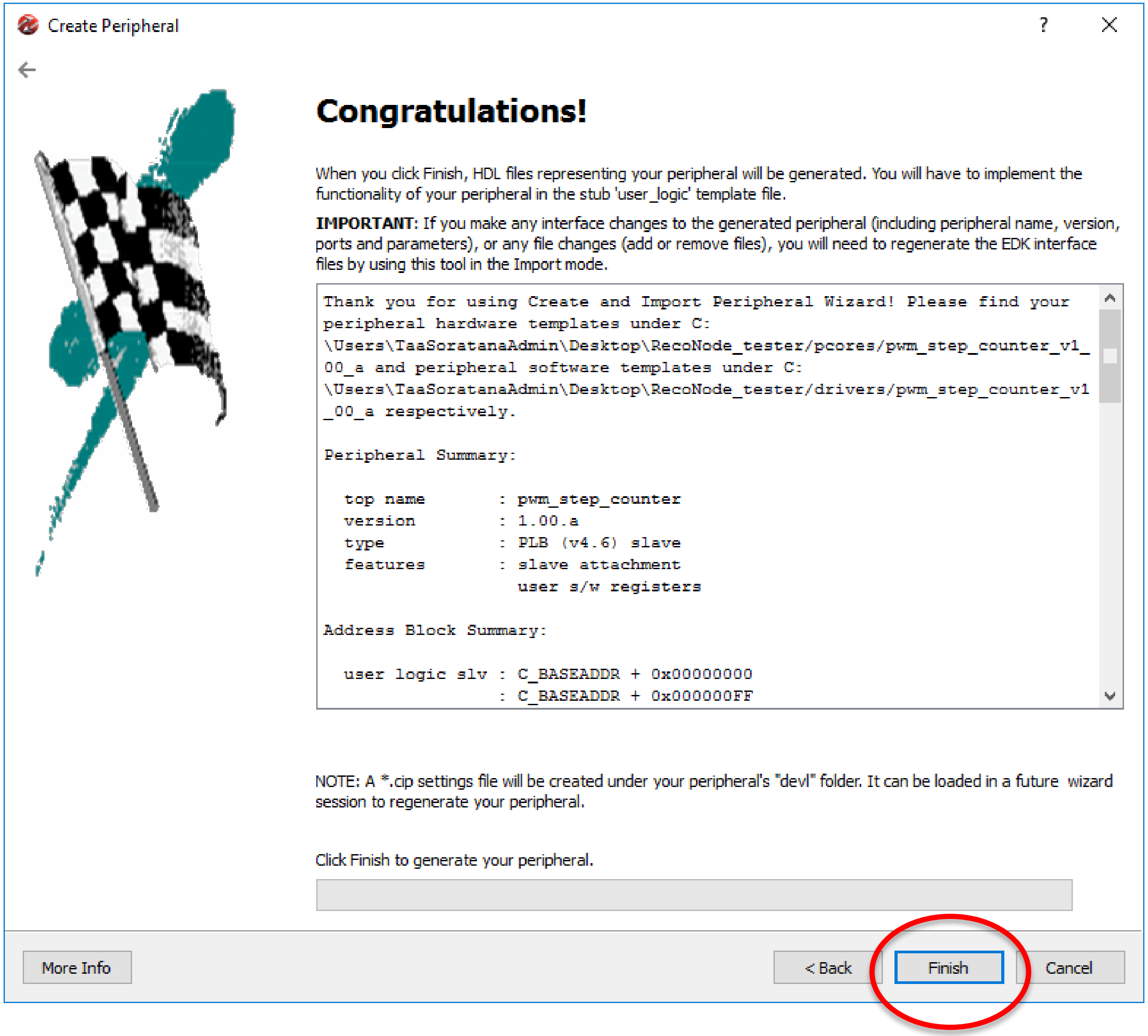
☐ Generate stub 'user_logic' template in Verilog instead of VHDL

☑ Generate ISE and XST project files to help you implement the peripheral using XST flow

☑ Generate template driver files to help you implement software interface

Check *“**Generate ISE and XST project files**”*
And *“**Generate template driver files**”*

2. B. c. Create a New IP Core



Create Peripheral

Congratulations!

When you click Finish, HDL files representing your peripheral will be generated. You will have to implement the functionality of your peripheral in the stub 'user_logic' template file.

IMPORTANT: If you make any interface changes to the generated peripheral (including peripheral name, version, ports and parameters), or any file changes (add or remove files), you will need to regenerate the EDK interface files by using this tool in the Import mode.

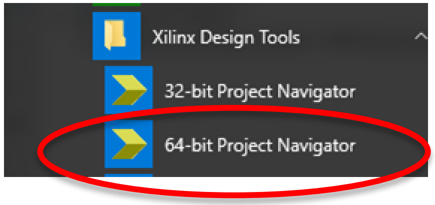
```
Thank you for using Create and Import Peripheral Wizard! Please find your
peripheral hardware templates under C:
\Users\TaaSoratanaAdmin\Desktop\RecoNode_tester/pcores/pwm_step_counter_v1_
00_a and peripheral software templates under C:
\Users\TaaSoratanaAdmin\Desktop\RecoNode_tester/drivers/pwm_step_counter_v1
_00_a respectively.

Peripheral Summary:

  top name       : pwm_step_counter
  version        : 1.00.a
  type           : PLB (v4.6) slave
  features       : slave attachment
                   user s/w registers

Address Block Summary:

  user logic slv : C_BASEADDR + 0x00000000
                 : C_BASEADDR + 0x000000FF
```

NOTE: A *.cip settings file will be created under your peripheral's "devl" folder. It can be loaded in a future wizard session to regenerate your peripheral.

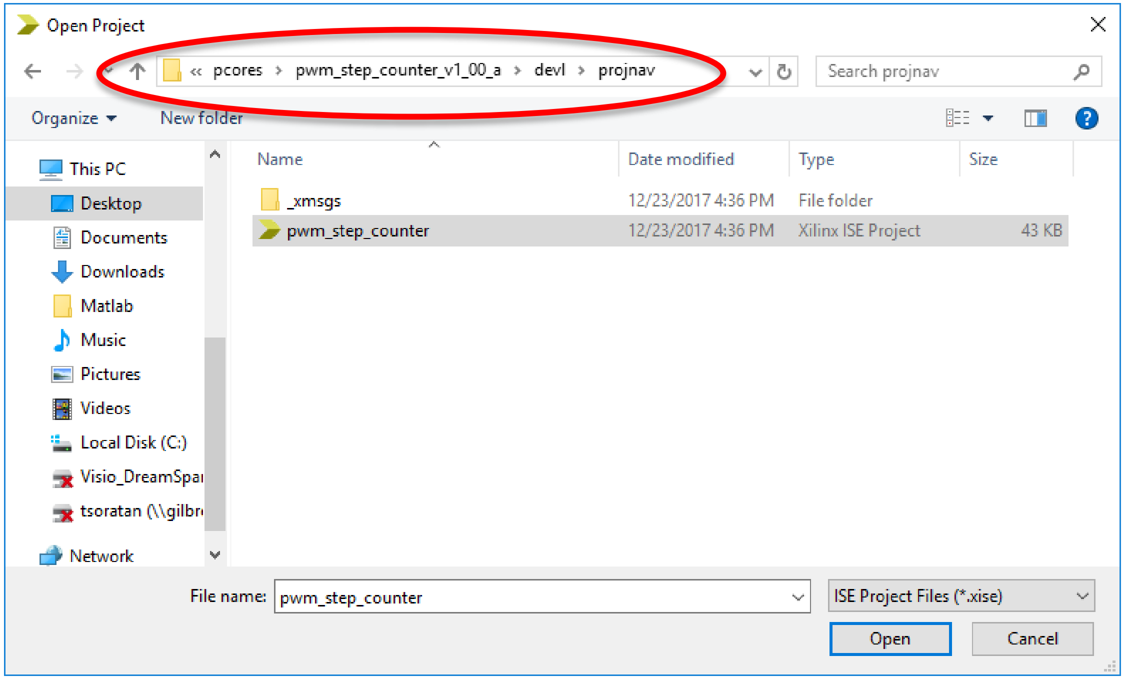
Click Finish to generate your peripheral.

2. B. c. Create a New IP Core

Go to ***Files / Open Project***



Open ***Project Navigator***



« pcores › pwm_step_counter_v1_00_a › devl › projnav

Name	Date modified	Type	Size
_xmsgs	12/23/2017 4:36 PM	File folder	
pwm_step_counter	12/23/2017 4:36 PM	Xilinx ISE Project	43 KB

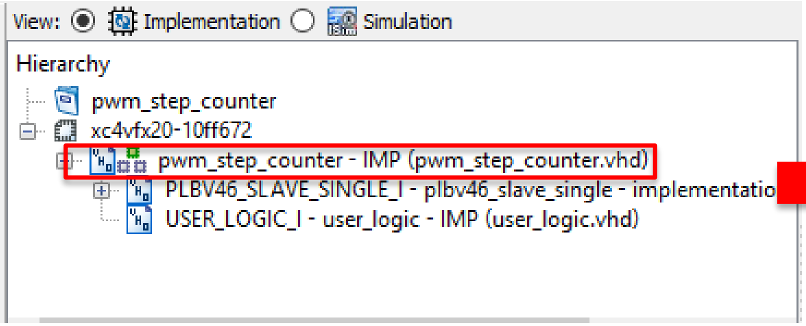
File name: pwm_step_counter

ISE Project Files (*.xise)

Go to your XPS project folder,
Then ***pcore/your_IP_core_name/devl/projnav***
Select the ISE project file located inside the directory

2. B. c. Create a New IP Core

Open ***Project Navigator***



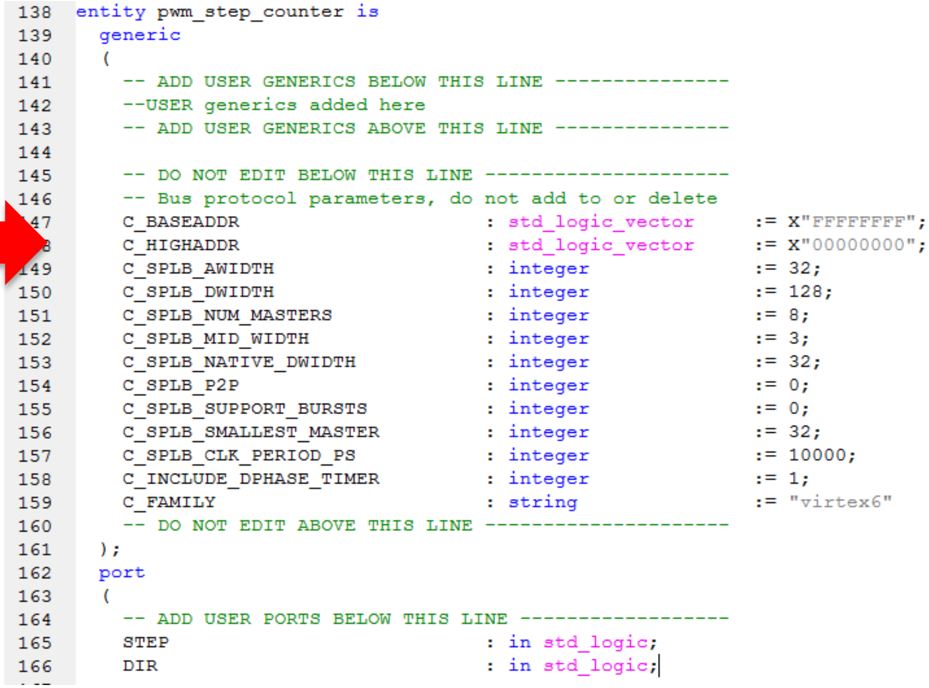
View: ● Implementation ○ Simulation

Hierarchy

- pwm_step_counter
- xc4vfx20-10ff672
  - pwm_step_counter - IMP (pwm_step_counter.vhd)
    - PLBV46_SLAVE_SINGLE_I - plbv46_slave_single - implementatio
    - USER_LOGIC_I - user_logic - IMP (user_logic.vhd)

Double click on pwm_step_counter.vhd

Write a VHDL code for your new IP Core

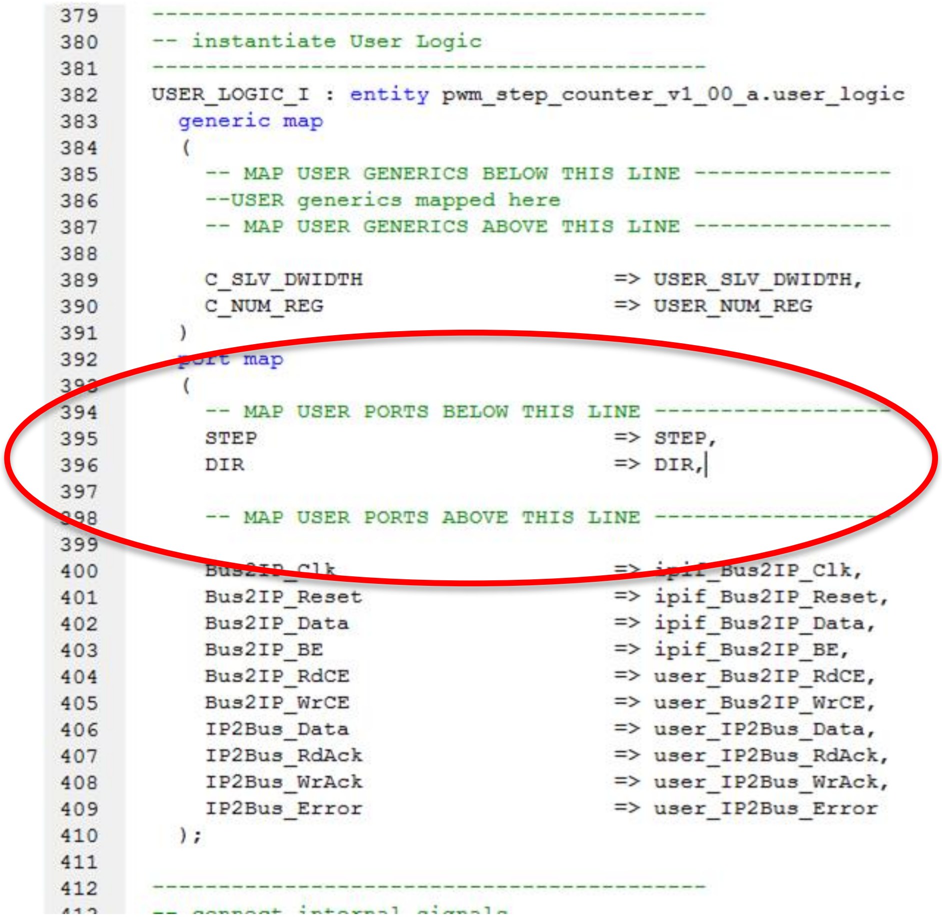


```
entity pwm_step_counter is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_BASEADDR                     : std_logic_vector     := X"FFFFFFFF";
    C_HIGHADDR                     : std_logic_vector     := X"00000000";
    C_SPLB_AWIDTH                  : integer              := 32;
    C_SPLB_DWIDTH                  : integer              := 128;
    C_SPLB_NUM_MASTERS             : integer              := 8;
    C_SPLB_MID_WIDTH               : integer              := 3;
    C_SPLB_NATIVE_DWIDTH           : integer              := 32;
    C_SPLB_P2P                     : integer              := 0;
    C_SPLB_SUPPORT_BURSTS          : integer              := 0;
    C_SPLB_SMALLEST_MASTER         : integer              := 32;
    C_SPLB_CLK_PERIOD_PS           : integer              := 10000;
    C_INCLUDE_DPHASE_TIMER         : integer              := 1;
    C_FAMILY                       : string               := "virtex6"
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    STEP                           : in std_logic;
    DIR                            : in std_logic;
```

For this example...
Add these line to the user defined port in entity block in pwm_step_counter.vhd

```
STEP                : in std_logic;
DIR                 : in std_logic;
```

2. B. c. Create a New IP Core

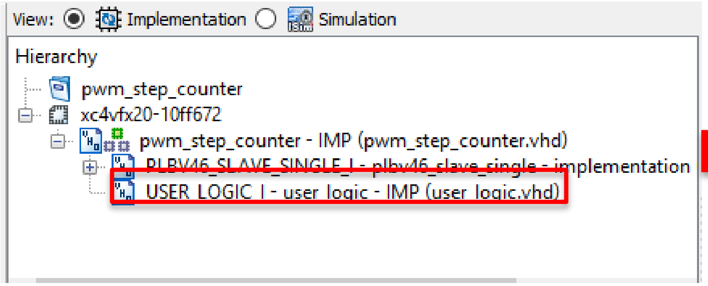


```
------------------------------------------
-- instantiate User Logic
------------------------------------------
USER_LOGIC_I : entity pwm_step_counter_v1_00_a.user_logic
  generic map
  (
    -- MAP USER GENERICS BELOW THIS LINE ---------------
    --USER generics mapped here
    -- MAP USER GENERICS ABOVE THIS LINE ---------------

    C_SLV_DWIDTH                    => USER_SLV_DWIDTH,
    C_NUM_REG                       => USER_NUM_REG
  )
  port map
  (
    -- MAP USER PORTS BELOW THIS LINE ------------------
    STEP                            => STEP,
    DIR                             => DIR,

    -- MAP USER PORTS ABOVE THIS LINE ------------------

    Bus2IP_Clk                      => ipif_Bus2IP_Clk,
    Bus2IP_Reset                    => ipif_Bus2IP_Reset,
    Bus2IP_Data                     => ipif_Bus2IP_Data,
    Bus2IP_BE                       => ipif_Bus2IP_BE,
    Bus2IP_RdCE                     => user_Bus2IP_RdCE,
    Bus2IP_WrCE                     => user_Bus2IP_WrCE,
    IP2Bus_Data                     => user_IP2Bus_Data,
    IP2Bus_RdAck                    => user_IP2Bus_RdAck,
    IP2Bus_WrAck                    => user_IP2Bus_WrAck,
    IP2Bus_Error                    => user_IP2Bus_Error
  );

------------------------------------------
-- connect internal signals
```

Add these line to the port map block, inside USER_LOGIC_I block

```
STEP                => STEP,
DIR                 => DIR,
```

2. B. c. Create a New IP Core

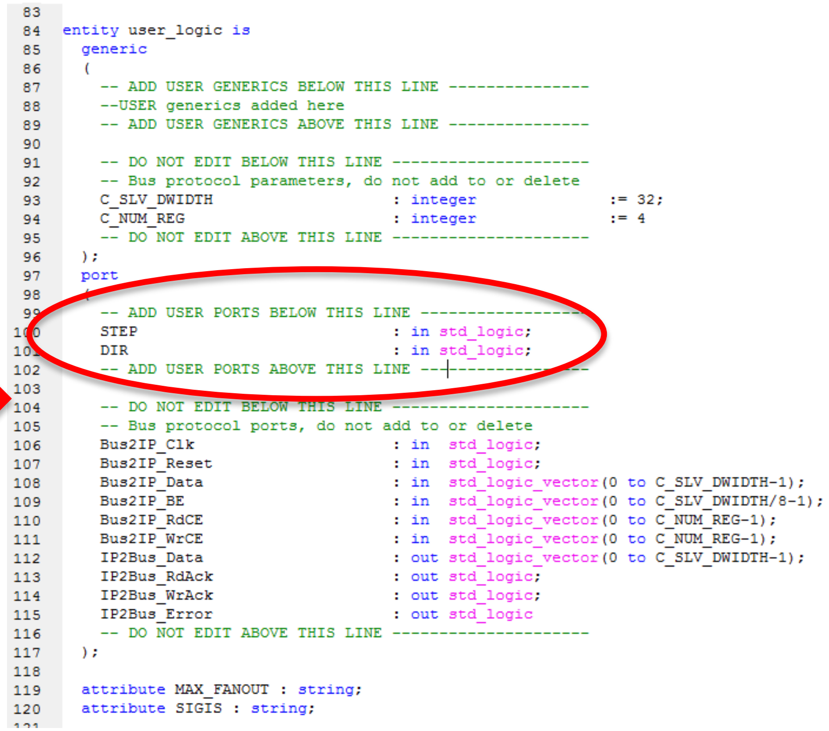


View: ● Implementation ○ Simulation

Hierarchy

- pwm_step_counter
- xc4vfx20-10ff672
  - pwm_step_counter - IMP (pwm_step_counter.vhd)
    - PLBV46_SLAVE_SINGLE_I - plbv46_slave_single - implementation
    - USER_LOGIC_I - user_logic - IMP (user_logic.vhd)

Double click user_logic.vhd

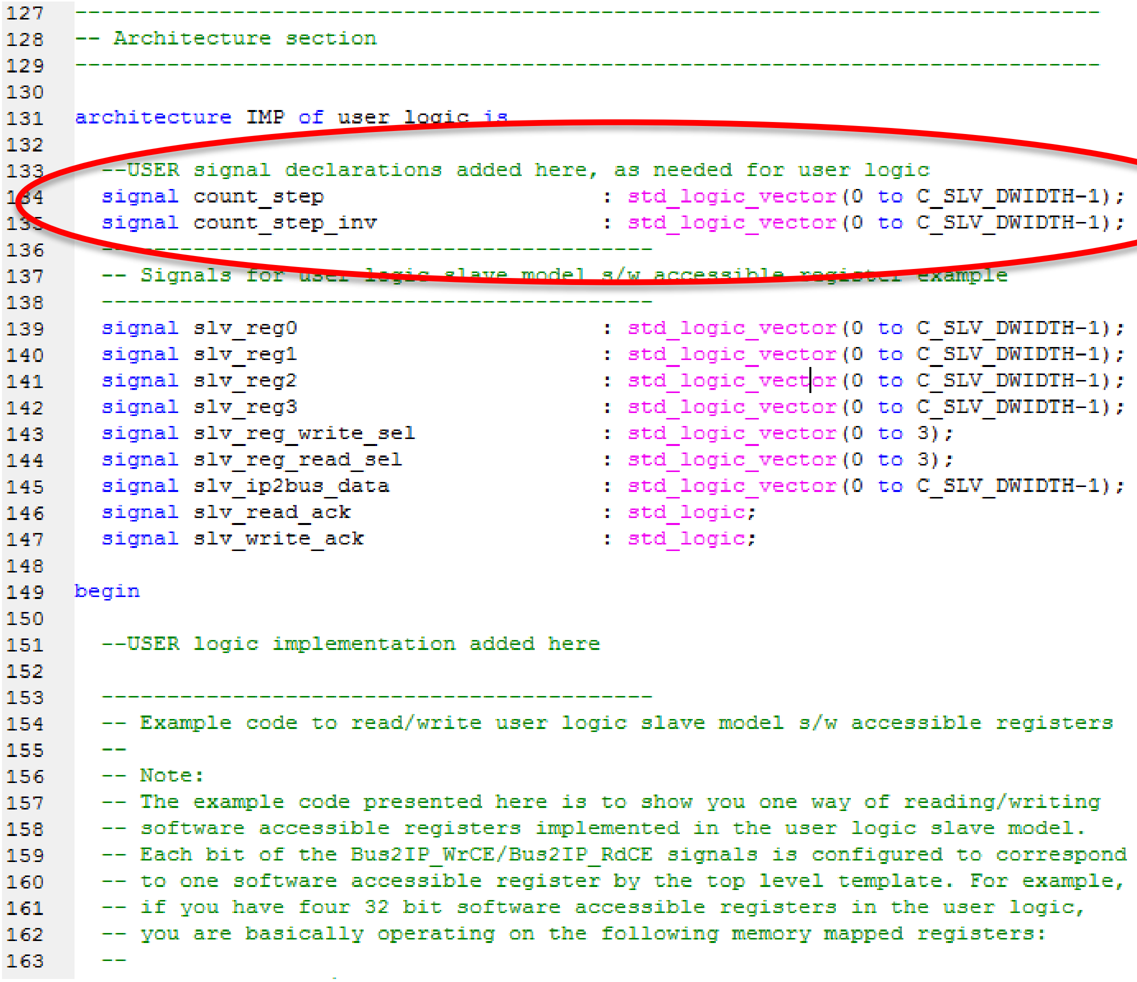


```
entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_DWIDTH                   : integer              := 32;
    C_NUM_REG                      : integer              := 4
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    STEP                           : in std_logic;
    DIR                            : in std_logic;
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                     : in  std_logic;
    Bus2IP_Reset                   : in  std_logic;
    Bus2IP_Data                    : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                      : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                    : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                   : out std_logic;
    IP2Bus_WrAck                   : out std_logic;
    IP2Bus_Error                   : out std_logic
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );

  attribute MAX_FANOUT : string;
  attribute SIGIS : string;
```

Add user defined port in entity block in user_logic.vhd

```
STEP                : in std_logic;
DIR                 : in std_logic;
```

2. B. c. Create a New IP Core



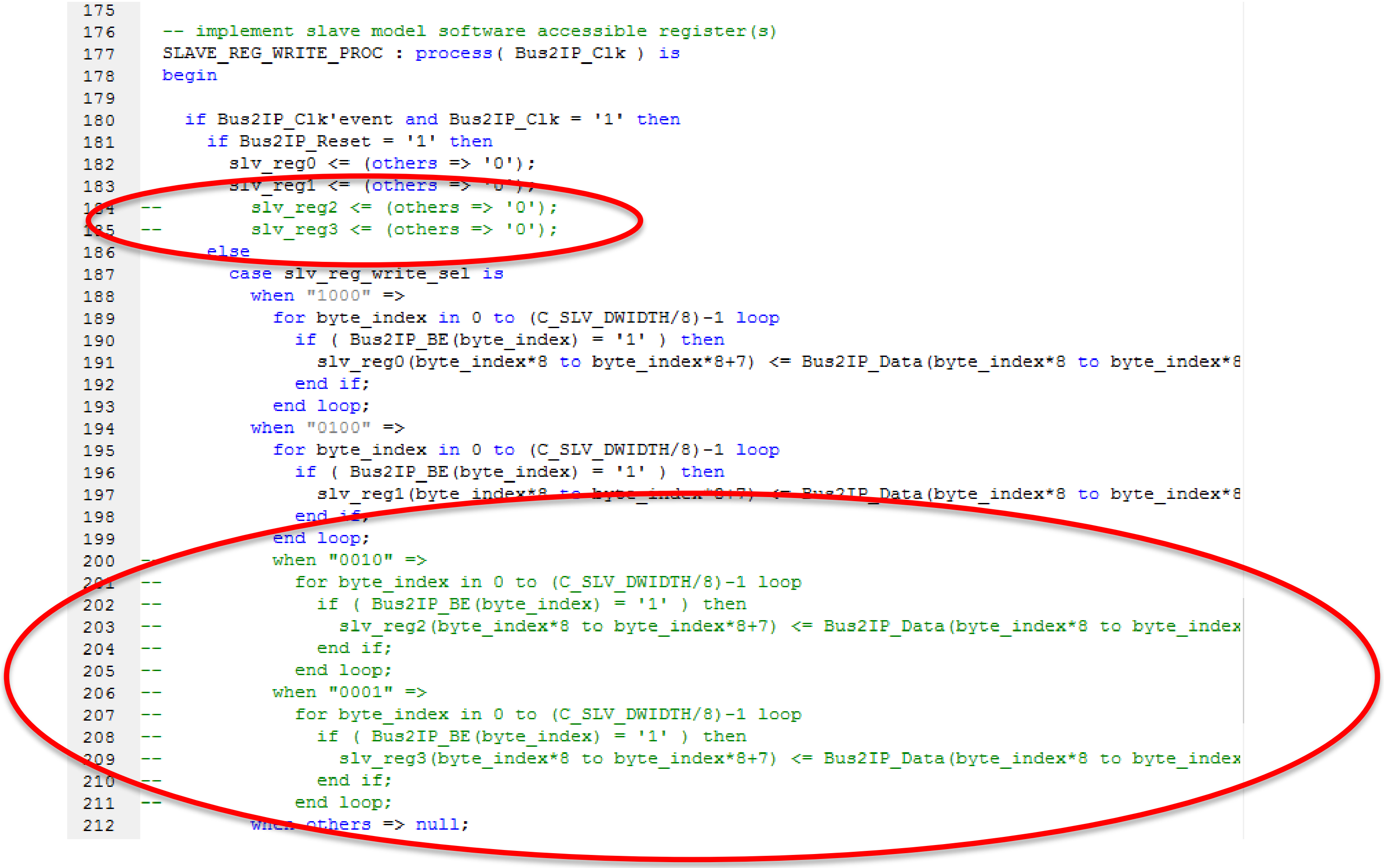
```vhdl
------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic
  signal count_step                     : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal count_step_inv                 : std_logic_vector(0 to C_SLV_DWIDTH-1);
  ------------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ------------------------------------------
  signal slv_reg0                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg1                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg2                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg3                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg_write_sel              : std_logic_vector(0 to 3);
  signal slv_reg_read_sel               : std_logic_vector(0 to 3);
  signal slv_ip2bus_data                : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_read_ack                   : std_logic;
  signal slv_write_ack                  : std_logic;

begin

  --USER logic implementation added here

  ------------------------------------------
  -- Example code to read/write user logic slave model s/w accessible registers
  --
  -- Note:
  -- The example code presented here is to show you one way of reading/writing
  -- software accessible registers implemented in the user logic slave model.
  -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
  -- to one software accessible register by the top level template. For example,
  -- if you have four 32 bit software accessible registers in the user logic,
  -- you are basically operating on the following memory mapped registers:
  --
```

Go to architecture section, add

```
signal count_step           : std_logic_vector(0 to C_SLV_DWIDTH-1);
signal count_step_inv     : std_logic_vector(0 to C_SLV_DWIDTH-1);
```

These are the “signals,” or variables, in our coding logic

2. B. c. Create a New IP Core



```
 -- implement slave model software accessible register(s)
 SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
 begin

  if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
   if Bus2IP_Reset = '1' then
    slv_reg0 <= (others => '0');
    slv_reg1 <= (others => '0');
--     slv_reg2 <= (others => '0');
--     slv_reg3 <= (others => '0');
   else
    case slv_reg_write_sel is
     when "1000" =>
      for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
       if ( Bus2IP_BE(byte_index) = '1' ) then
        slv_reg0(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8
       end if;
      end loop;
     when "0100" =>
      for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
       if ( Bus2IP_BE(byte_index) = '1' ) then
        slv_reg1(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index*8
       end if;
      end loop;
--      when "0010" =>
--       for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
--        if ( Bus2IP_BE(byte_index) = '1' ) then
--         slv_reg2(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index
--        end if;
--       end loop;
--      when "0001" =>
--       for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
--        if ( Bus2IP_BE(byte_index) = '1' ) then
--         slv_reg3(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to byte_index
--        end if;
--       end loop;
     when others => null;
```

Go to implement "**slave model software accessible register(s)**" subsection, comment out the line shown above

This makes slave register 2 and 3 (slv_reg2, slv_reg3) **read only**.

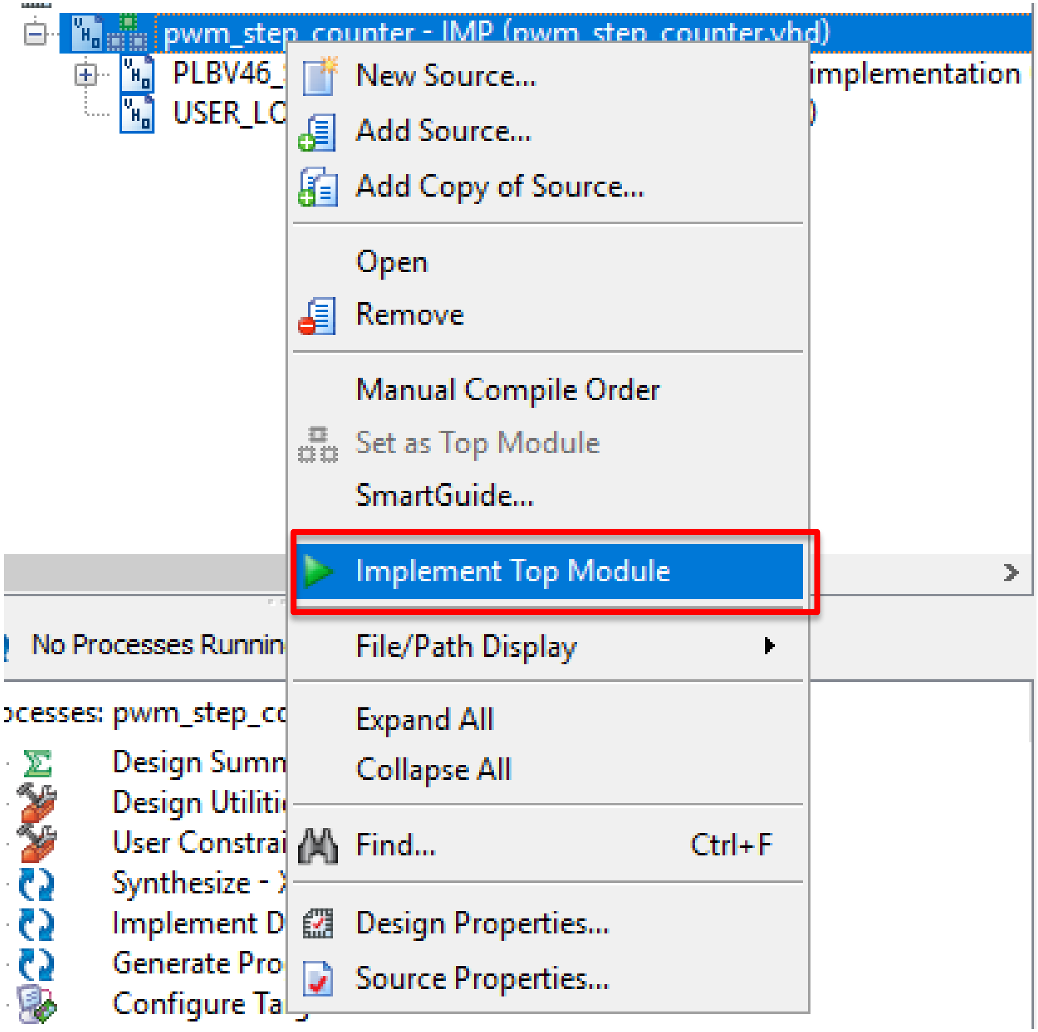
2. B. c. Create a New IP Core

Now we are creating a logic, which can count the step pulses (from STEP), with regard to the current state of direction pin (defined as DIR).

Add these line right before `end IMP;`

```
-- process (@var) == if the system detect the change in @var, this section of code will activate
process (STEP)  begin
    -- if the change is rising edge (low reading to high reading) and the reader (slv_reg0) is
    -- x"00000001"
    if rising_edge(STEP) and slv_reg0 = x"00000001" then
        -- if the direction is 1, we count up, else we count down
           if DIR = '1' then
                      count_step <= count_step + 1;
                      count_step_inv <= count_step_inv - 1;
           else
                      count_step <= count_step - 1;
                      count_step_inv <= count_step_inv + 1;
           end if;
    end if;
    -- if we receive reset signal (slv_reg1 = x"00000000"), then count_step and count_step_inv
    -- will be set to 0
    if slv_reg1 = x"00000000" then
           count_step <= x"00000000";
           count_step_inv <= x"00000000";
    end if;
    -- send the counter value to the output registers (slv_reg2 and slv_reg3)
    slv_reg2 <= count_step;
    slv_reg3 <= count_step_inv;
end process;
```

2. B. c. Create a New IP Core

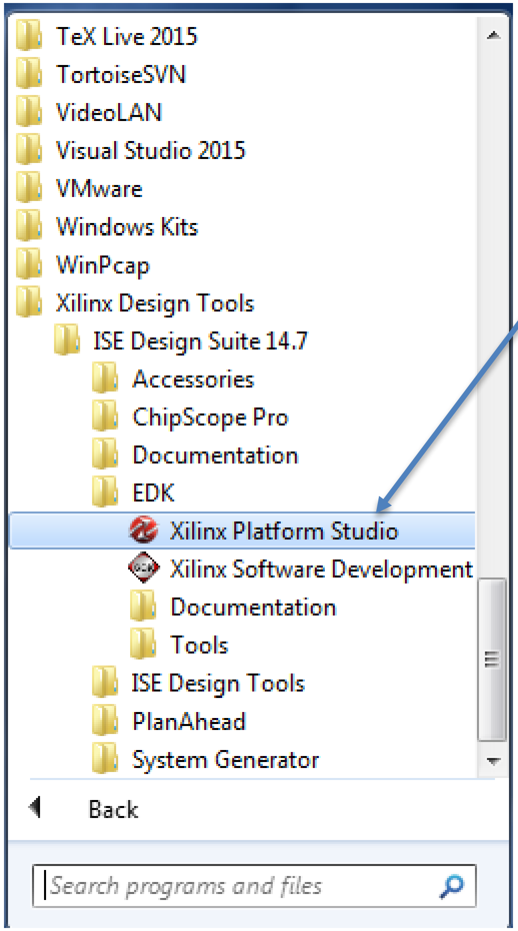


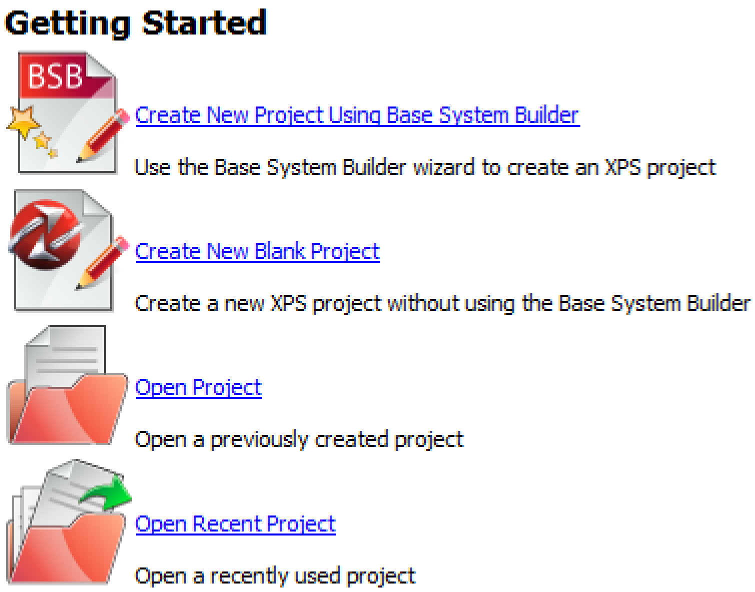
Now, compile the vhd by right click the pwm_step_counter.vhd and select ***"Implement Top Module"***

3. Creating a New Project

3. Xilinx Platform Studio

Open Xilinx Platform Studio, and create a new project (go to **File** > **New BSB Project**)





Getting Started

Create New Project Using Base System Builder

Use the Base System Builder wizard to create an XPS project

Create New Blank Project

Create a new XPS project without using the Base System Builder

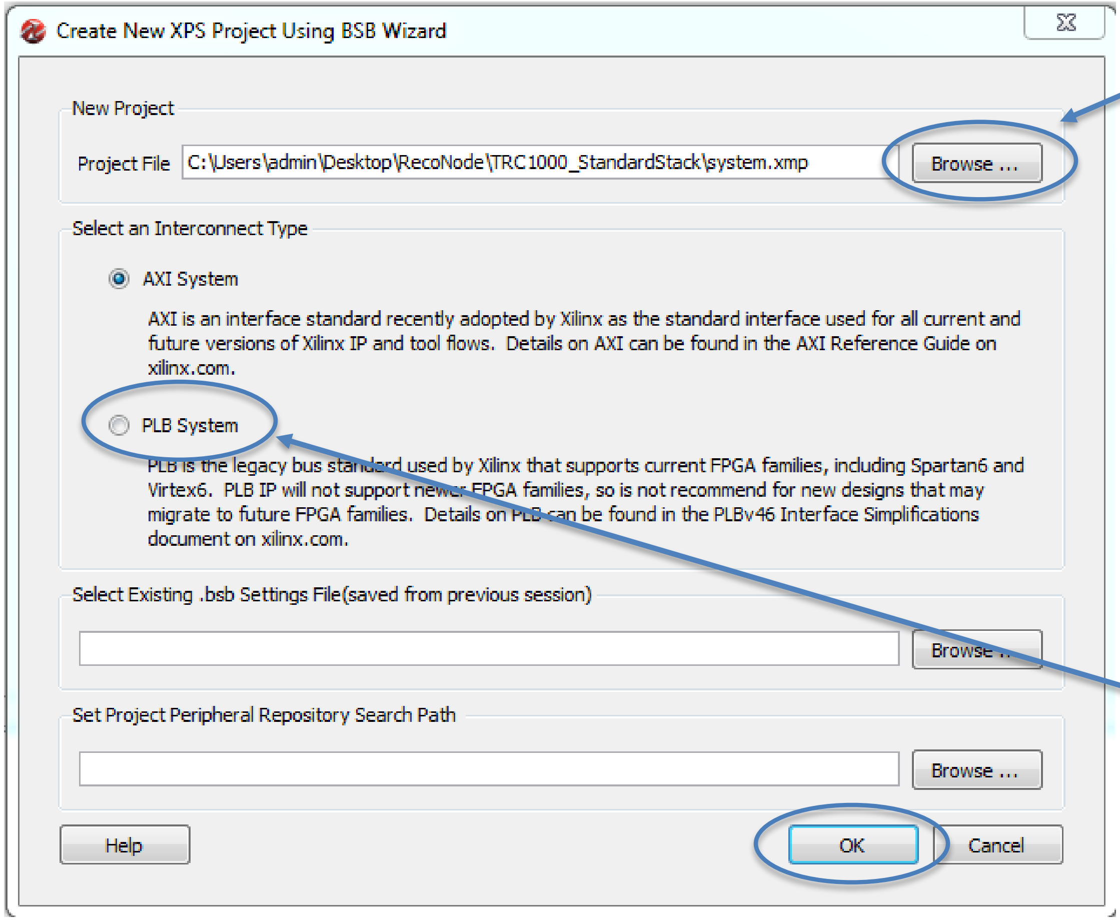
Open Project

Open a previously created project

Open Recent Project

Open a recently used project

3. Xilinx Platform Studio



Create New XPS Project Using BSB Wizard

New Project

Project File: C:\Users\admin\Desktop\RecoNode\TRC1000_StandardStack\system.xmp

Select an Interconnect Type

● AXI System

AXI is an interface standard recently adopted by Xilinx as the standard interface used for all current and future versions of Xilinx IP and tool flows. Details on AXI can be found in the AXI Reference Guide on xilinx.com.

○ PLB System

PLB is the legacy bus standard used by Xilinx that supports current FPGA families, including Spartan6 and Virtex6. PLB IP will not support newer FPGA families, so is not recommend for new designs that may migrate to future FPGA families. Details on PLB can be found in the PLBv46 Interface Simplifications document on xilinx.com.

Select Existing .bsb Settings File(saved from previous session)

Set Project Peripheral Repository Search Path

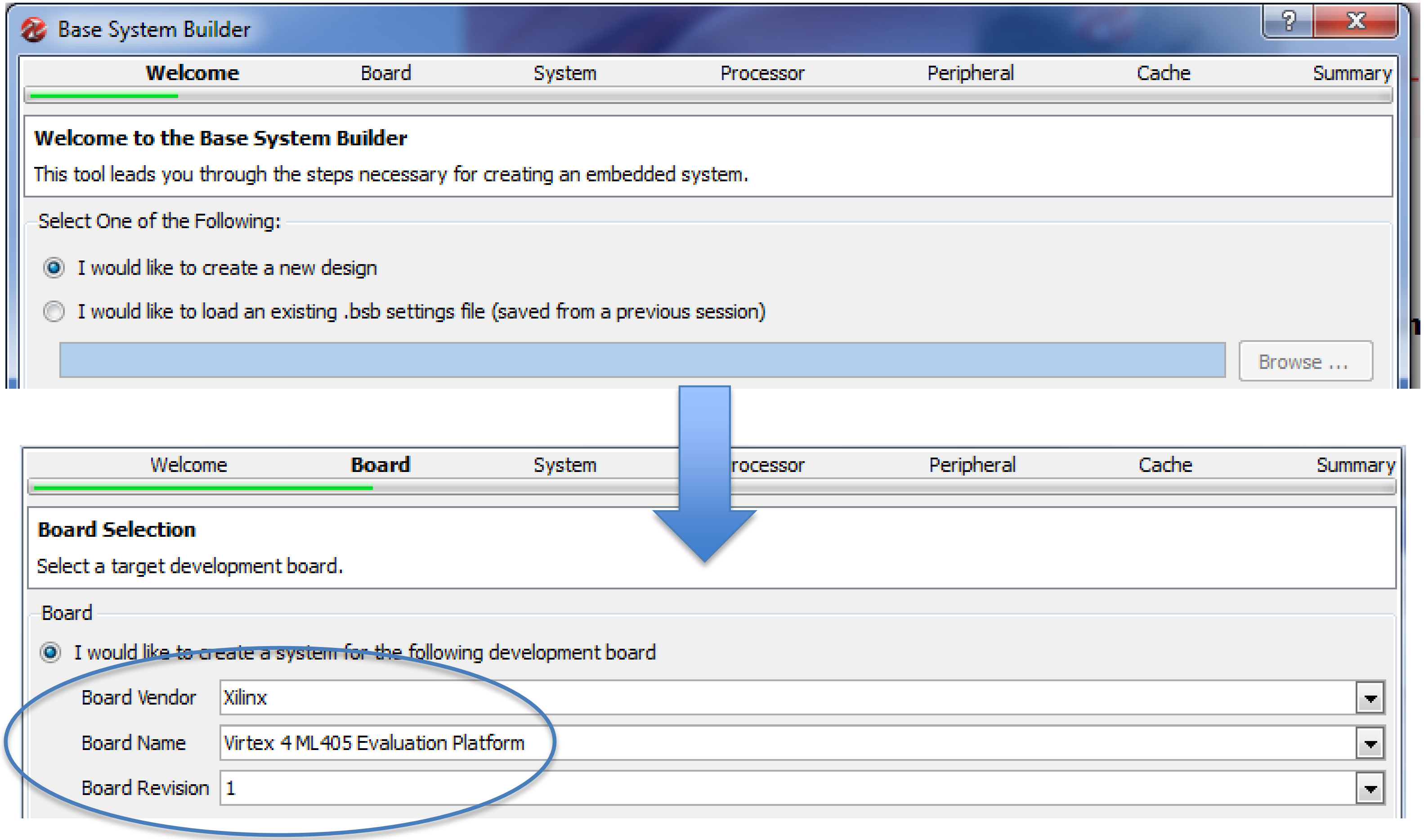
- Browse to the location for your project
- Enter the directory name
- Save the *.xmp* file there

* Your directory name can't be too long or contains special characters. Make it simple (ex. locate in desktop)

- Choose PLB

*Our RecoNode uses Xilinx Virtex-4, which is supported by PLB system.

3. Xilinx Platform Studio



Welcome to the Base System Builder

This tool leads you through the steps necessary for creating an embedded system.

Select One of the Following:

● I would like to create a new design

○ I would like to load an existing .bsb settings file (saved from a previous session)

Board Selection

Select a target development board.

Board

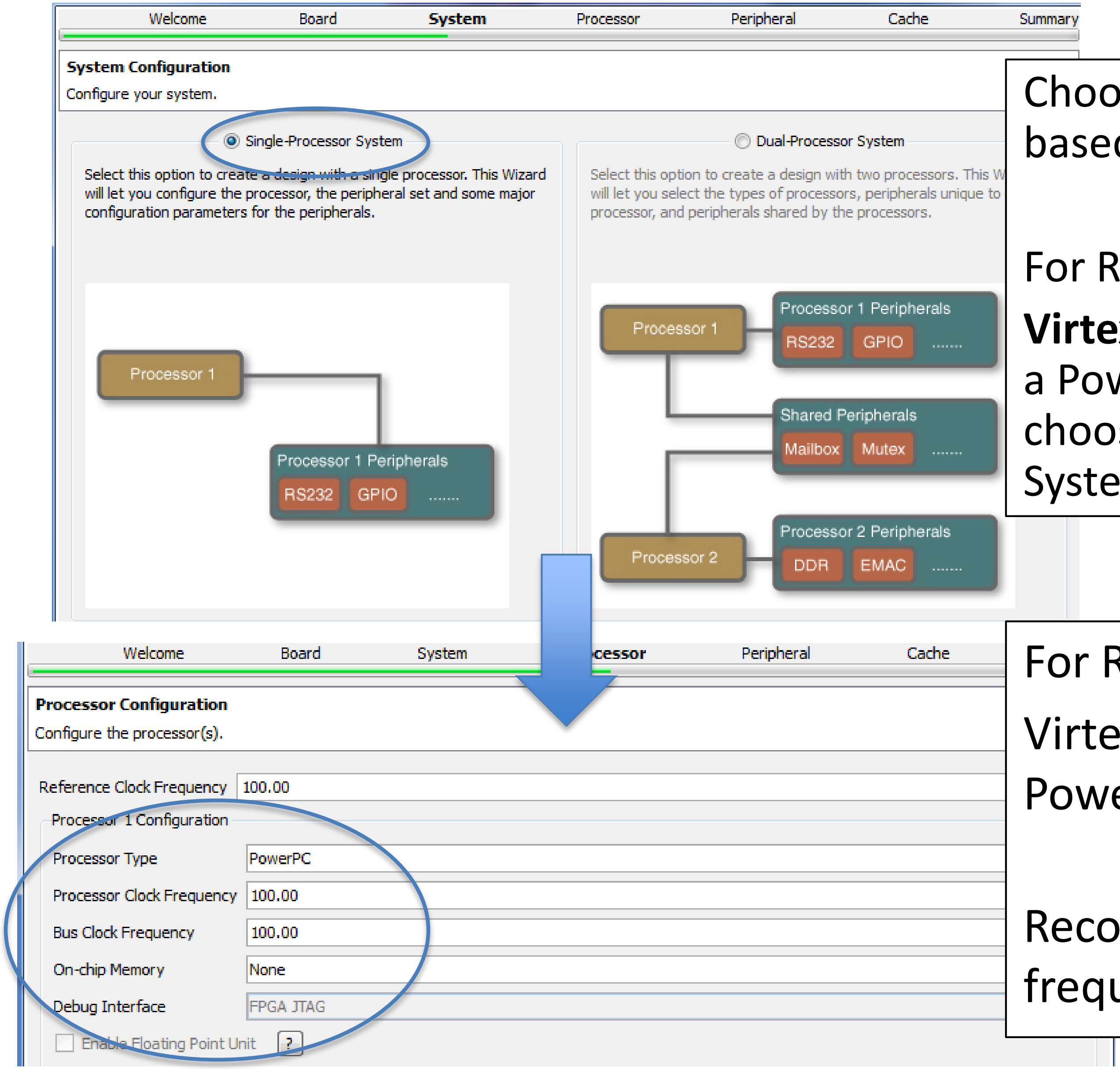
● I would like to create a system for the following development board

Board Vendor	Xilinx
Board Name	Virtex 4 ML405 Evaluation Platform
Board Revision	1

Choose Xilinx for Vendor, and Virtex 4 ML 405 for Board Name

* The **Virtex-4 FPGA XC4VFX20-FF672-10** is on the RecoNode.

3. Xilinx Platform Studio



System Configuration

Configure your system.

● Single-Processor System

Select this option to create a design with a single processor. This Wizard will let you configure the processor, the peripheral set and some major configuration parameters for the peripherals.

○ Dual-Processor System

Select this option to create a design with two processors. This W will let you select the types of processors, peripherals unique to processor, and peripherals shared by the processors.

Processor Configuration

Configure the processor(s).

Reference Clock Frequency 100.00

Processor 1 Configuration

Processor Type	PowerPC
Processor Clock Frequency	100.00
Bus Clock Frequency	100.00
On-chip Memory	None
Debug Interface	FPGA JTAG

□ Enable Floating Point Unit ?

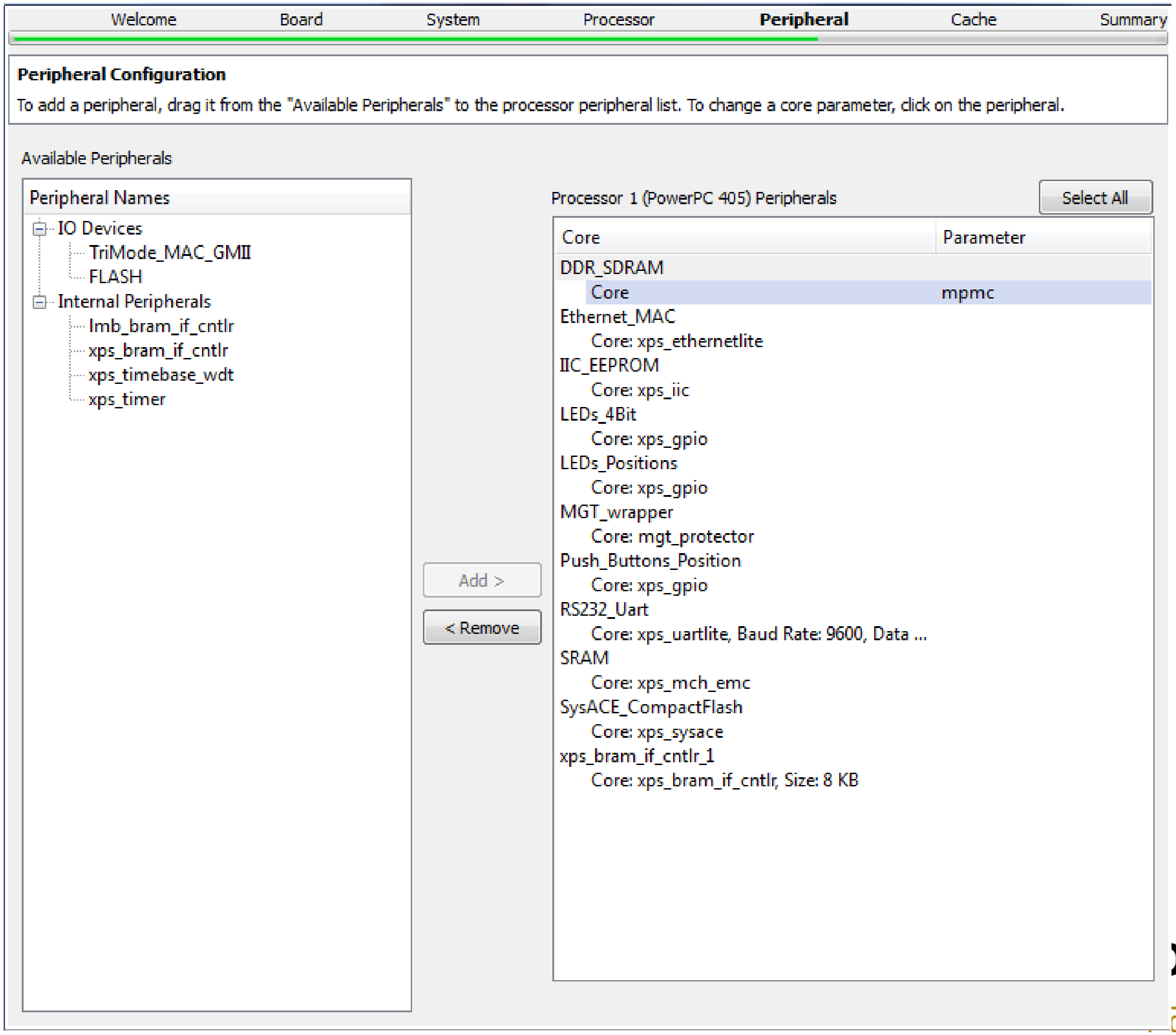
Choose Processor system based on your Chip.

For RecoNode V1.1, we uses **Virtex-4: XC4VFX20** – This has a PowerPC processor, thus choose Single-Processor System.

For RecoNode V1.1, we uses Virtex-4: XC4VFX20 – choose PowerPC processor.

RecoNode has **100MHz** clock frequency.

3. Xilinx Platform Studio



Peripheral Configuration

To add a peripheral, drag it from the "Available Peripherals" to the processor peripheral list. To change a core parameter, click on the peripheral.

Available Peripherals

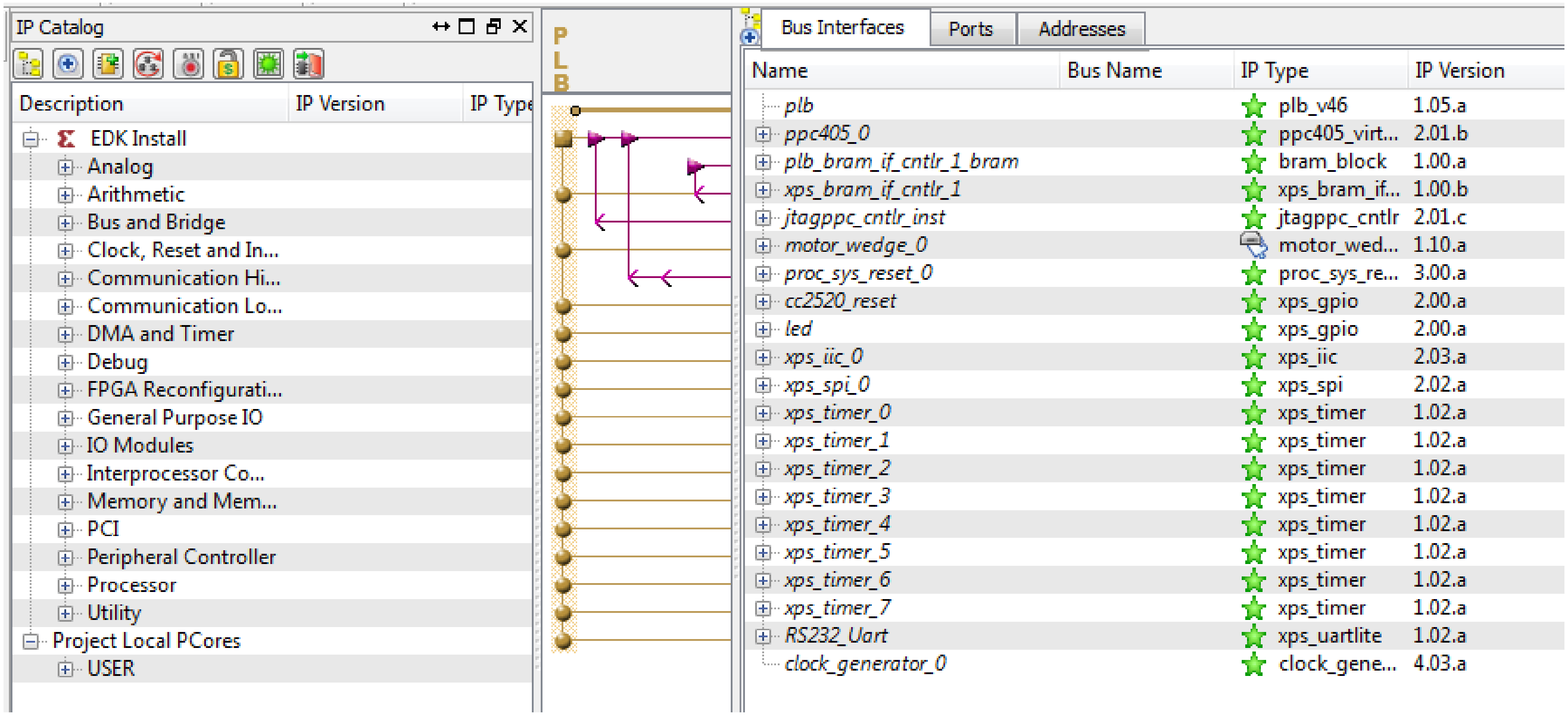
Peripheral Names

- IO Devices
  - TriMode_MAC_GMII
  - FLASH
- Internal Peripherals
  - lmb_bram_if_cntlr
  - xps_bram_if_cntlr
  - xps_timebase_wdt
  - xps_timer

Processor 1 (PowerPC 405) Peripherals

Core	Parameter
DDR_SDRAM	
Core	mpmc
Ethernet_MAC	
Core: xps_ethernetlite	
IIC_EEPROM	
Core: xps_iic	
LEDs_4Bit	
Core: xps_gpio	
LEDs_Positions	
Core: xps_gpio	
MGT_wrapper	
Core: mgt_protector	
Push_Buttons_Position	
Core: xps_gpio	
RS232_Uart	
Core: xps_uartlite, Baud Rate: 9600, Data ...	
SRAM	
Core: xps_mch_emc	
SysACE_CompactFlash	
Core: xps_sysace	
xps_bram_if_cntlr_1	
Core: xps_bram_if_cntlr, Size: 8 KB	

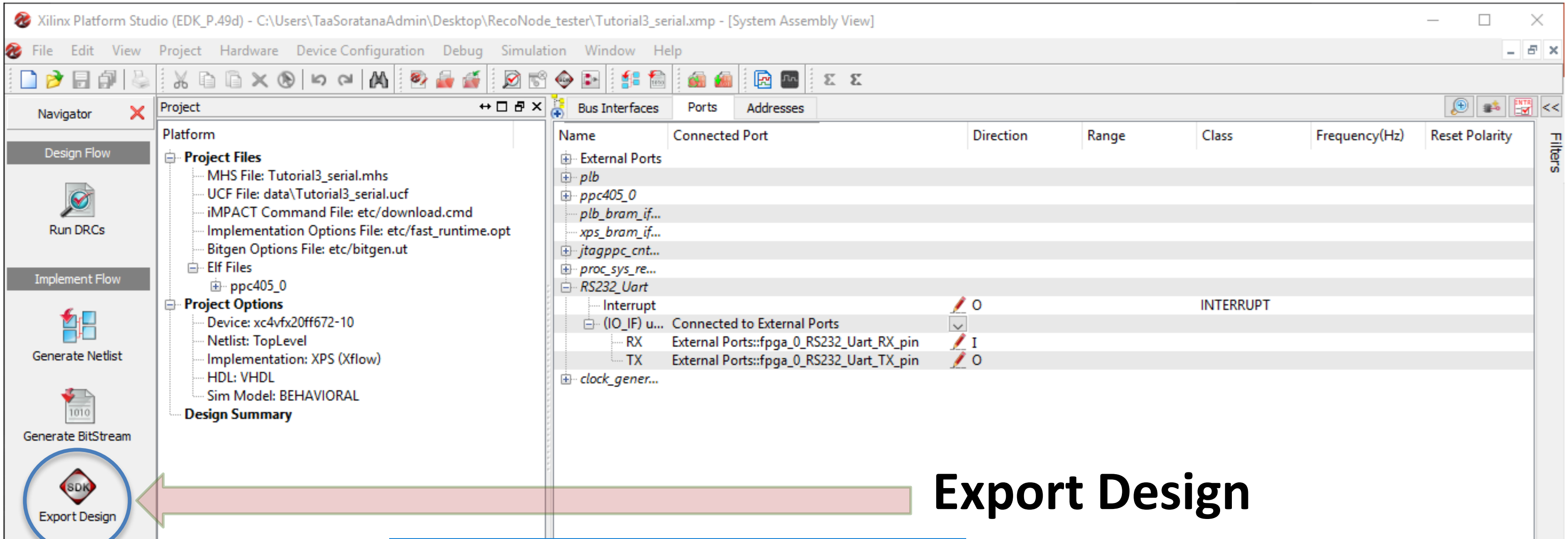
3. Xilinx Platform Studio



Description	IP Version	IP Type
EDK Install		
Analog		
Arithmetic		
Bus and Bridge		
Clock, Reset and In...		
Communication Hi...		
Communication Lo...		
DMA and Timer		
Debug		
FPGA Reconfigurati...		
General Purpose IO		
IO Modules		
Interprocessor Co...		
Memory and Mem...		
PCI		
Peripheral Controller		
Processor		
Utility		
Project Local PCores		
USER		

Name	Bus Name	IP Type	IP Version
plb		plb_v46	1.05.a
ppc405_0		ppc405_virt...	2.01.b
plb_bram_if_cntlr_1_bram		bram_block	1.00.a
xps_bram_if_cntlr_1		xps_bram_if...	1.00.b
jtagppc_cntlr_inst		jtagppc_cntlr	2.01.c
motor_wedge_0		motor_wed...	1.10.a
proc_sys_reset_0		proc_sys_re...	3.00.a
cc2520_reset		xps_gpio	2.00.a
led		xps_gpio	2.00.a
xps_iic_0		xps_iic	2.03.a
xps_spi_0		xps_spi	2.02.a
xps_timer_0		xps_timer	1.02.a
xps_timer_1		xps_timer	1.02.a
xps_timer_2		xps_timer	1.02.a
xps_timer_3		xps_timer	1.02.a
xps_timer_4		xps_timer	1.02.a
xps_timer_5		xps_timer	1.02.a
xps_timer_6		xps_timer	1.02.a
xps_timer_7		xps_timer	1.02.a
RS232_Uart		xps_uartlite	1.02.a
clock_generator_0		clock_gene...	4.03.a

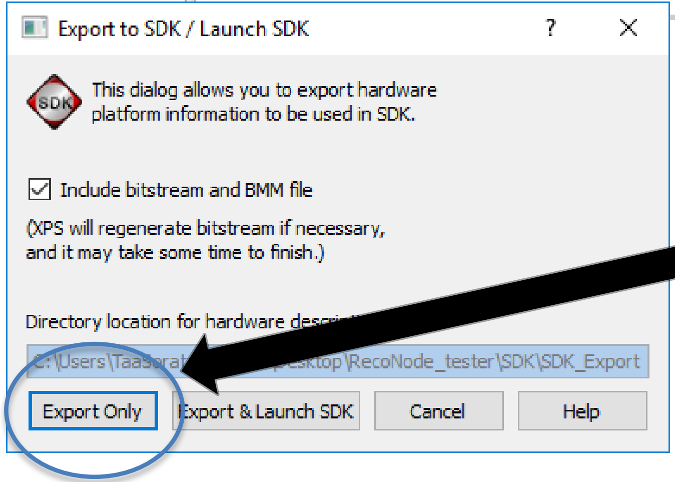
3. Xilinx Platform Studio



Platform
- Project Files
  - MHS File: Tutorial3_serial.mhs
  - UCF File: data\Tutorial3_serial.ucf
  - iMPACT Command File: etc/download.cmd
  - Implementation Options File: etc/fast_runtime.opt
  - Bitgen Options File: etc/bitgen.ut
  - Elf Files
    - ppc405_0
- Project Options
  - Device: xc4vfx20ff672-10
  - Netlist: TopLevel
  - Implementation: XPS (Xflow)
  - HDL: VHDL
  - Sim Model: BEHAVIORAL
- Design Summary

Name	Connected Port	Direction	Range	Class	Frequency(Hz)	Reset Polarity
External Ports						
plb						
ppc405_0						
plb_bram_if...						
xps_bram_if...						
jtagppc_cnt...						
proc_sys_re...						
RS232_Uart						
Interrupt		O		INTERRUPT		
(IO_IF) u...	Connected to External Ports					
RX	External Ports::fpga_0_RS232_Uart_RX_pin	I				
TX	External Ports::fpga_0_RS232_Uart_TX_pin	O				
clock_gener...						

Export Design



Export to SDK / Launch SDK

This dialog allows you to export hardware platform information to be used in SDK.

☑ Include bitstream and BMM file

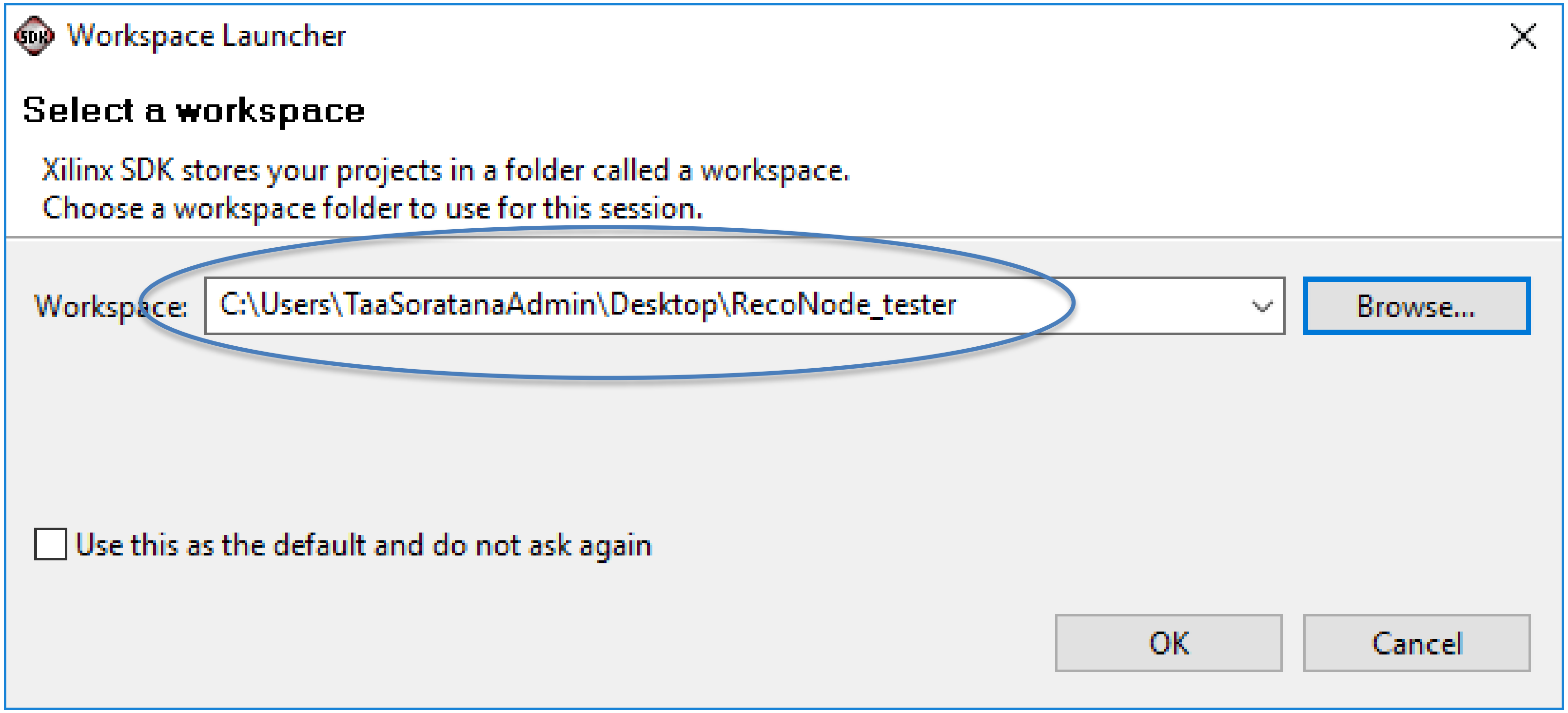
(XPS will regenerate bitstream if necessary, and it may take some time to finish.)

Directory location for hardware descri

C:\Users\Taasorat esktop\RecoNode_tester\SDK\SDK_Export

To SDK!

3. Xilinx SDK



Workspace Launcher

Select a workspace

Xilinx SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

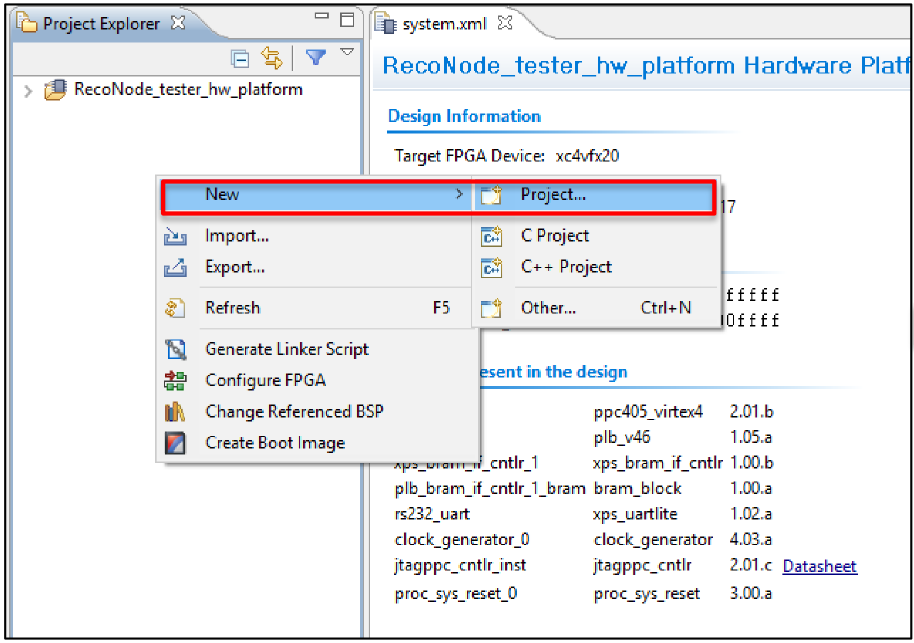
Workspace: C:\Users\TaaSoratanaAdmin\Desktop\RecoNode_tester

□ Use this as the default and do not ask again

In Xilinx SDK, Select the workspace that contain your ***.xmp*** file

3. Xilinx SDK

Create a code project in Xilinx SDK



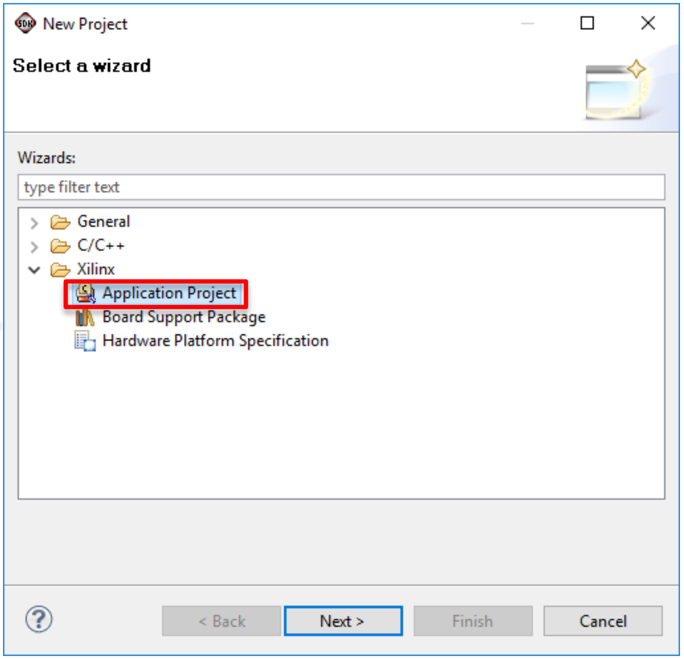
RecoNode_tester_hw_platform Hardware Platf

Design Information

Target FPGA Device: xc4vfx20

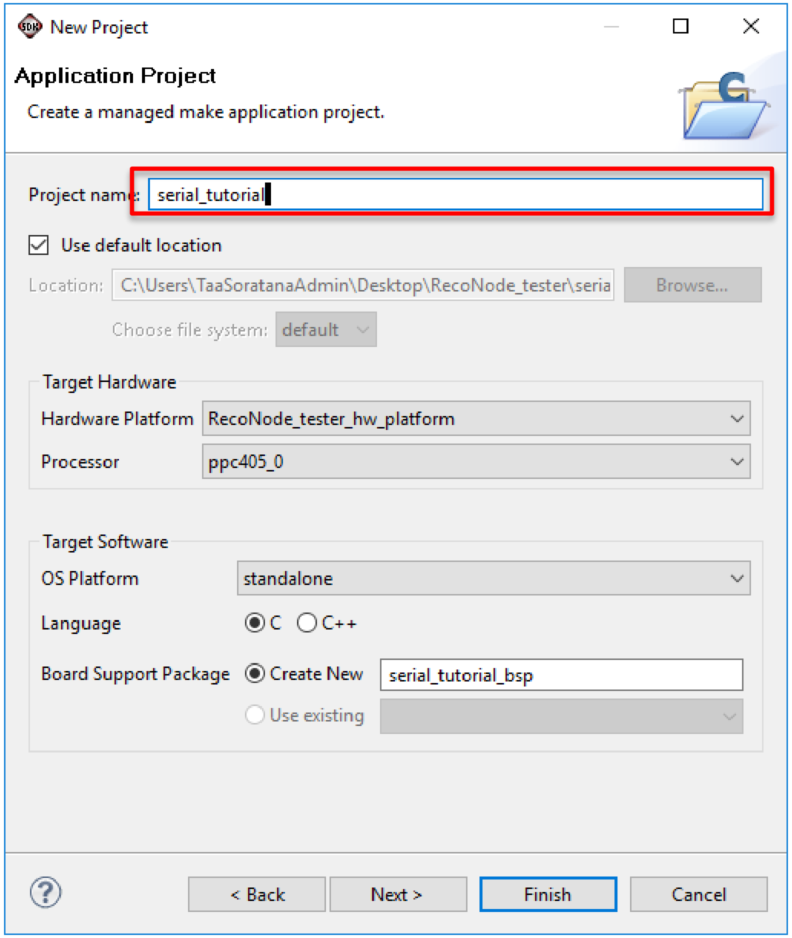
esent in the design

	ppc405_virtex4	2.01.b	
	plb_v46	1.05.a	
xps_bram_if_cntlr_1	xps_bram_if_cntlr	1.00.b	
plb_bram_if_cntlr_1_bram	bram_block	1.00.a	
rs232_uart	xps_uartlite	1.02.a	
clock_generator_0	clock_generator	4.03.a	
jtagppc_cntlr_inst	jtagppc_cntlr	2.01.c	Datasheet
proc_sys_reset_0	proc_sys_reset	3.00.a	



Right click in **Project Explorer**, Select **New** > **Project .**
Then **Xilinx > Application Project**

3. Xilinx SDK



New Project

Application Project

Create a managed make application project.

Project name: serial_tutorial

☑ Use default location

Location: C:\Users\TaaSoratanaAdmin\Desktop\RecoNode_tester\seria

Choose file system: default

Target Hardware

Hardware Platform: RecoNode_tester_hw_platform

Processor: ppc405_0

Target Software

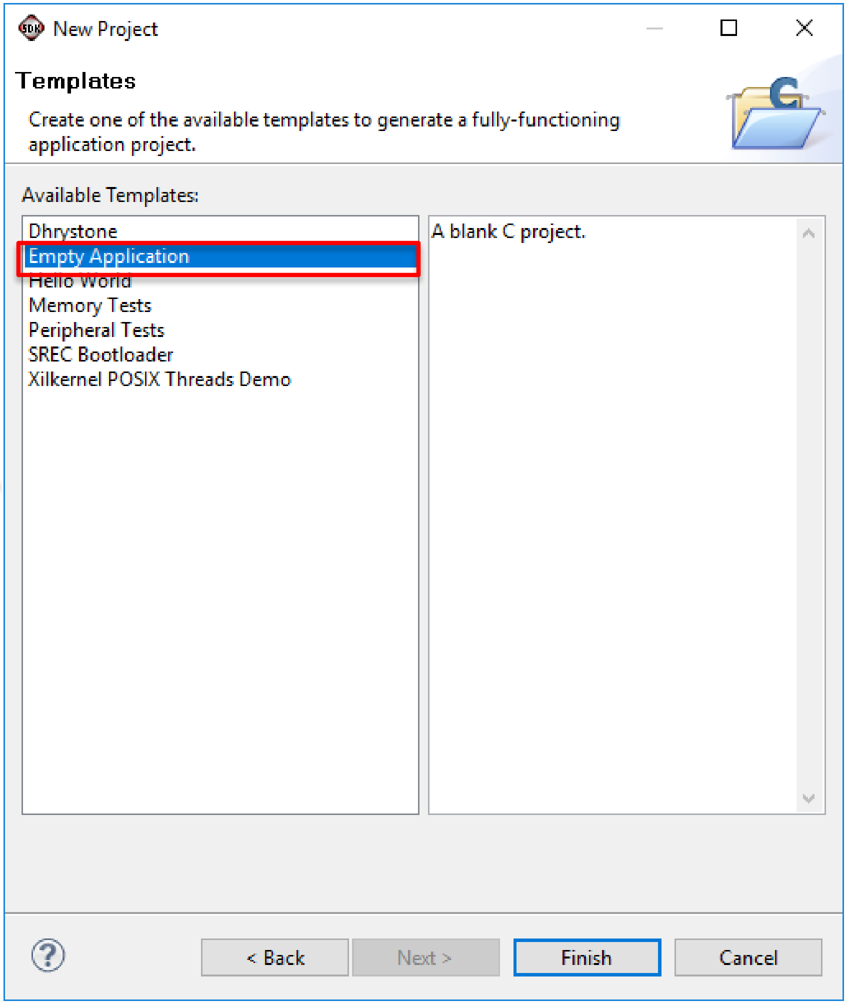
OS Platform: standalone

Language: ● C ○ C++

Board Support Package: ● Create New serial_tutorial_bsp

○ Use existing

Put your project name



New Project

Templates

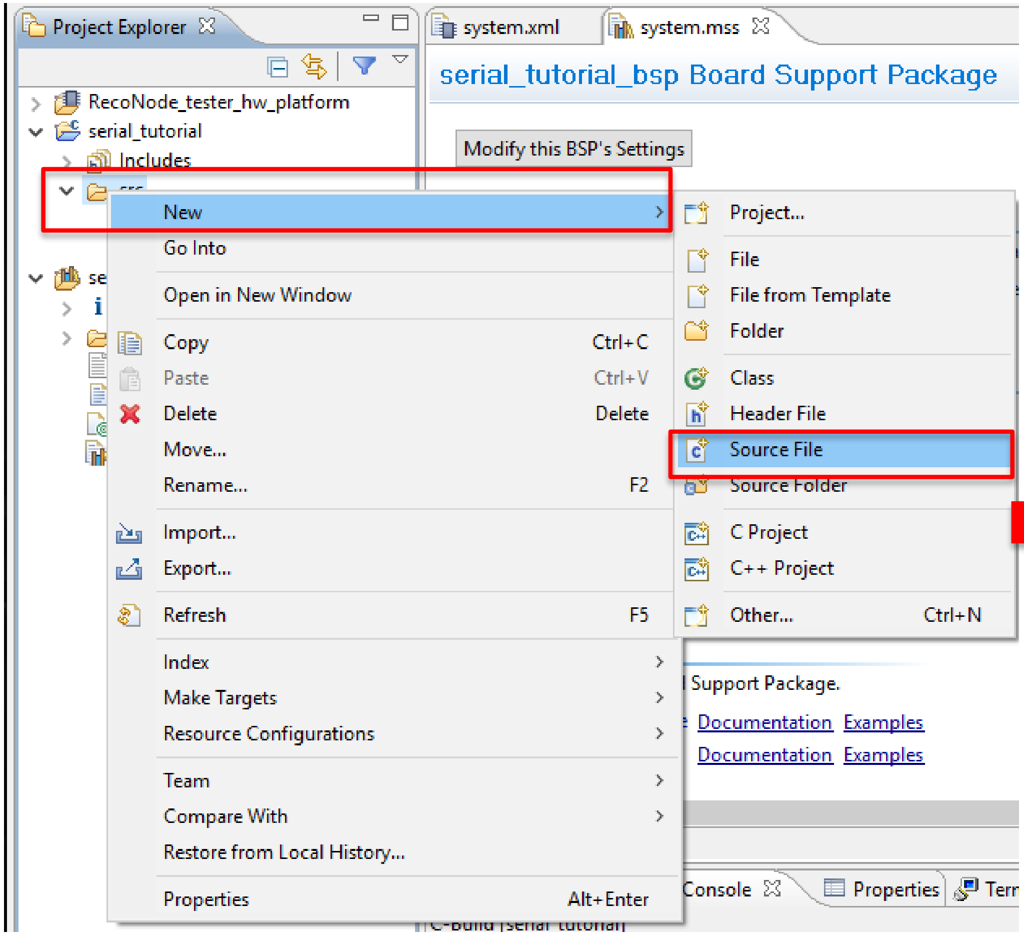
Create one of the available templates to generate a fully-functioning application project.

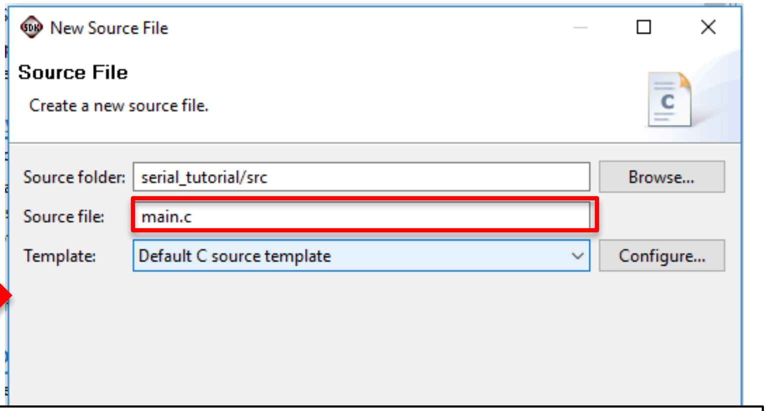
Available Templates:

- Dhrystone
- Empty Application
- Hello World
- Memory Tests
- Peripheral Tests
- SREC Bootloader
- Xilkernel POSIX Threads Demo

A blank C project.

3. Xilinx SDK





New Source File

Source File

Create a new source file.

Source folder: serial_tutorial/src

Source file: main.c

Template: Default C source template

Right click from ***Project Explorer***, go to ***New / Source File***

Put your main source file name. By default, it is **main.c**, but you should put a unique name that you would know that it is a main file.

3. Xilinx SDK

```c
#include <stdio.h>

int main()
{
    xil_printf("Hello From the Other Side! \n\r");
    return 0;
}
```

Sample code of printing on serial communication (UART).

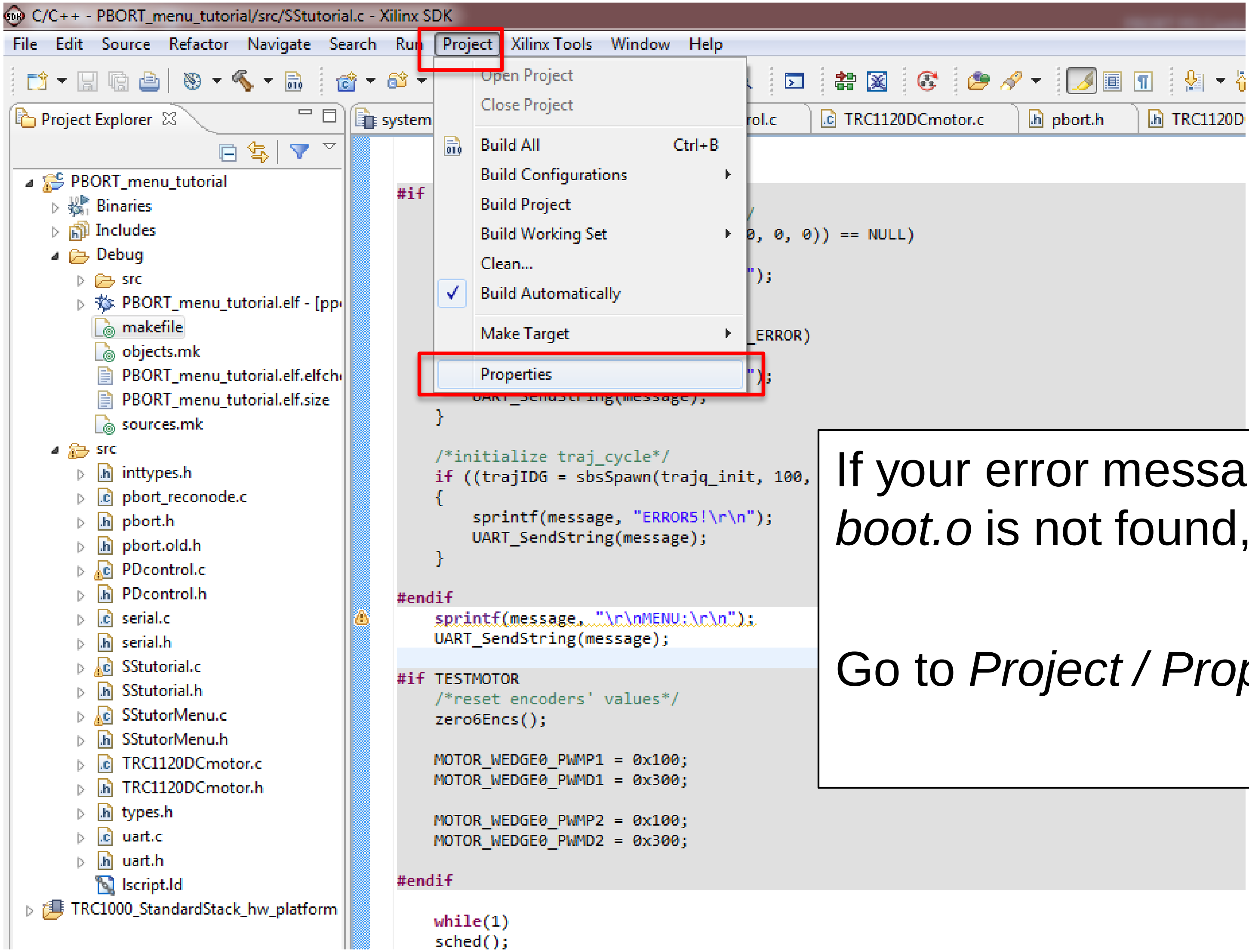
Add your code in your main source file.
This differs to what you want to achieve from your SDK.

Troubleshooting

4. Troubleshooting

Troubleshooting

Error: Cannot find *boot.o*



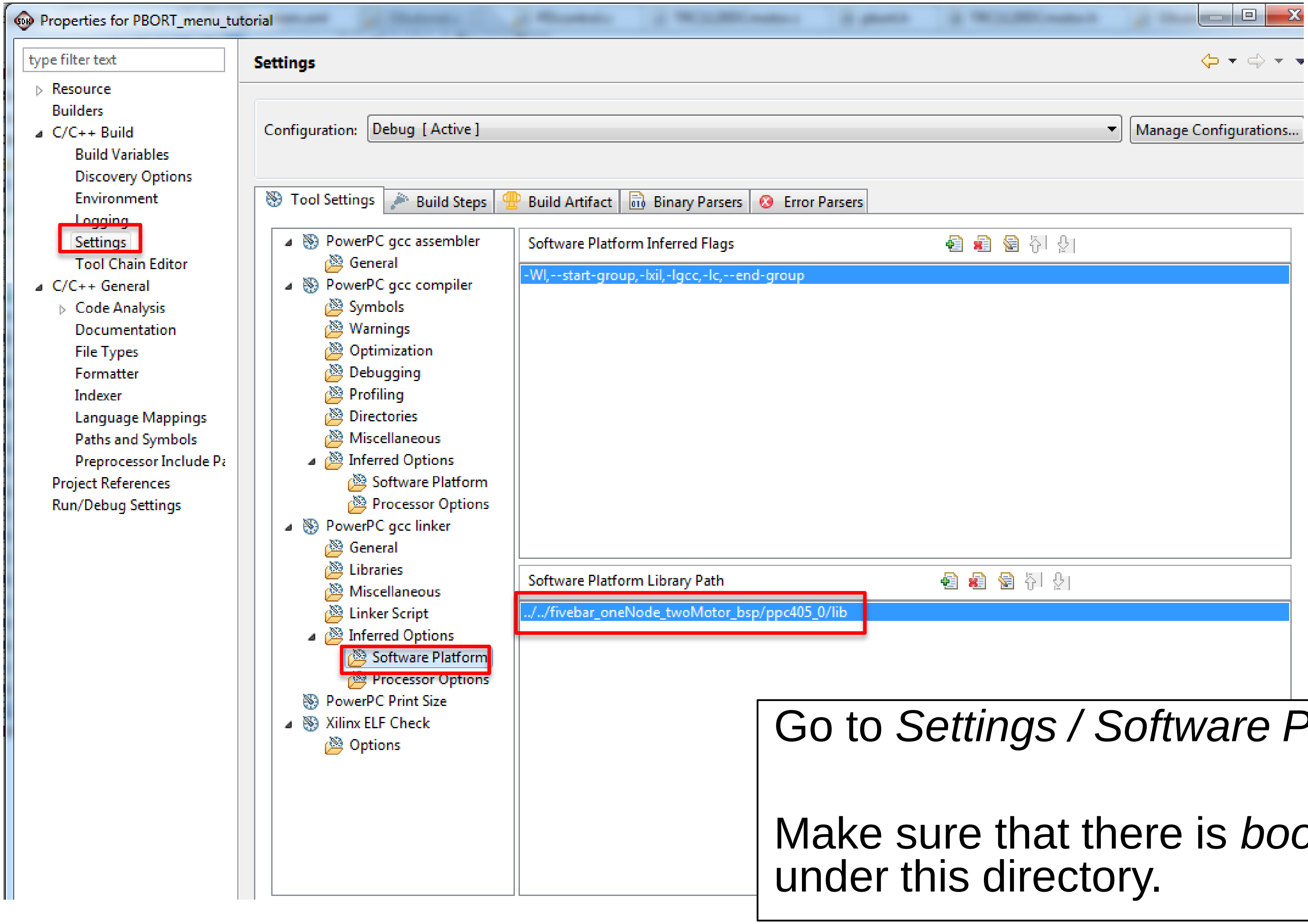
```
#if                                  /
                                     0, 0, 0)) == NULL)

                                     ");

                                     _ERROR)

                                     ");
        UART_SendString(message);
    }

    /*initialize traj_cycle*/
    if ((trajIDG = sbsSpawn(trajq_init, 100,
    {
        sprintf(message, "ERROR5!\r\n");
        UART_SendString(message);
    }

#endif
    sprintf(message, "\r\nMENU:\r\n");
    UART_SendString(message);

#if TESTMOTOR
    /*reset encoders' values*/
    zero6Encs();

    MOTOR_WEDGE0_PWMP1 = 0x100;
    MOTOR_WEDGE0_PWMD1 = 0x300;

    MOTOR_WEDGE0_PWMP2 = 0x100;
    MOTOR_WEDGE0_PWMD2 = 0x300;

#endif

    while(1)
    sched();
```

If your error message says that *boot.o* is not found,

Go to *Project / Properties*

Troubleshooting

Error: Cannot find *boot.o*



Configuration: Debug [Active]

Software Platform Inferred Flags

-Wl,--start-group,-lxil,-lgcc,-lc,--end-group

Software Platform Library Path

../../fivebar_oneNode_twoMotor_bsp/ppc405_0/lib

Go to *Settings / Software Platform*

Make sure that there is *boot.o* file under this directory.

Troubleshooting

If you need information on PBO/RT or looking for module library for RecoNode, they are posted on Dr. Voyles' website. The links are below.

Port-Based Objects / Real-Time (PBO/RT)

http://web.ics.purdue.edu/~rvoyles/Help/PBORT/pbort.help.html

PD Controller (Refer to RecoNode/TRC1120)

http://web.ics.purdue.edu/~rvoyles/Help/PBORT/PDcontrol.module.html