CANape (Measurement and Calibration Tool) Setup Manual – Version 1

Author: Ximu Zhang

Created Date: 02/02/2021

Objective

- CANape is a software tool used to measure and calibrate variables in the applications running in any microprocessors. It supports various protocols for data transfer (CAN, Ethernet, etc). At the protocol layer, it requires the XCP protocol for communication. This manual shows how to establish communication between CANape and ARM Cortex-A9 on the Xilinx ZYNQ-7000 Zedboard through XCP on Ethernet. Communication between FPGA and ARM will also be covered in this manual.

Software

- Vector CANape 17.0 (Demo Version)
- Xilinx Vivado Design Suite - HLx Editions 2018.2

Hardware Platform

- Xilinx ZYNQ-7000 Zedboard, containing
    - FPGA
    - ARM Cortex-A9 (dual cores)

Prerequisite

- (optional) A FPGA model built using Xilinx System Generator in MATLAB Simulink.

Example

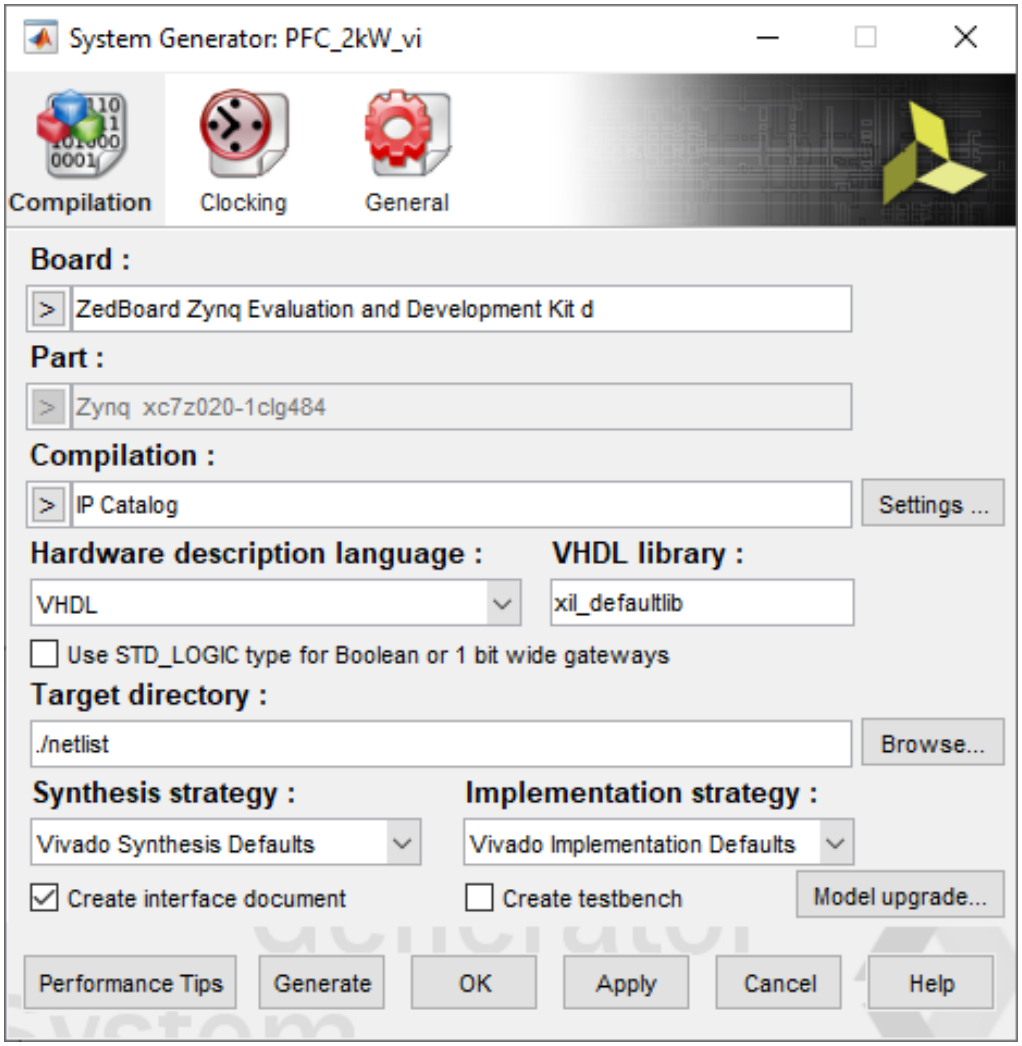
- PFC_2kW_vi

References

- https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/oslib_rm.pdf
- https://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf
- https://support.vector.com/kb?id=kb_article_view&sysparm_article=KB0011316&sys_kb_id=b9e40ea41b2614148e9a535c2e4bcb69&spa=1
- https://assets.vector.com/cms/content/application-areas/ecu-calibration/xcp/XCP_ReferenceBook_V3.0_EN.pdf
- https://assets.vector.com/cms/content/know-how/_application-notes/AN-IMC-1-024_How_do_I_create_an_A2L_file_from_CANape.pdf
- http://read.pudn.com/downloads200/ebook/942888/ASAP2.pdf
- https://cdn.intrepidcs.net/support/ASAP2Editor/icsASAP2Editor.pdf

Vivado Part

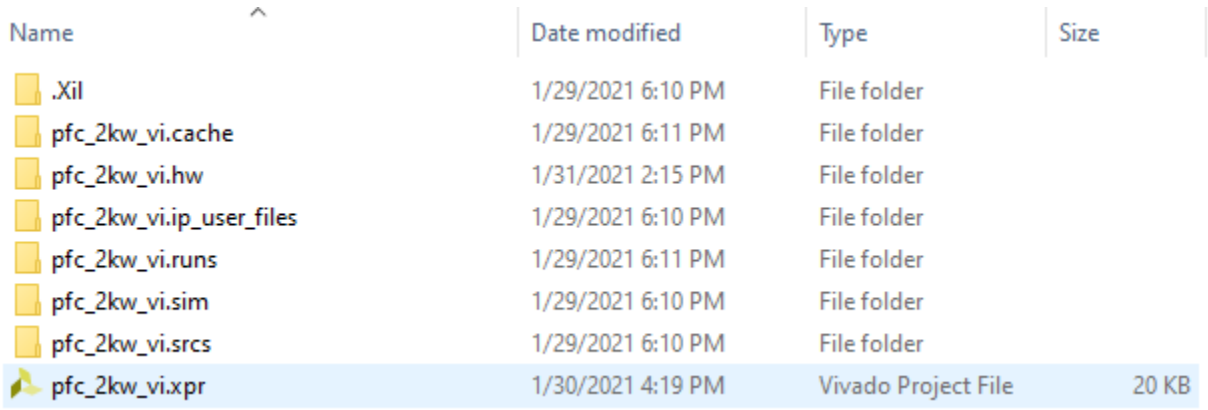
Step 1: generate Vivado project in MATLAB Simulink (If FPGA Is Used)



Step 2: Open Vivado Project

Path: ".\PFC_2kW_vi\PFC_2kW_vi\netlist\ip_catalog".

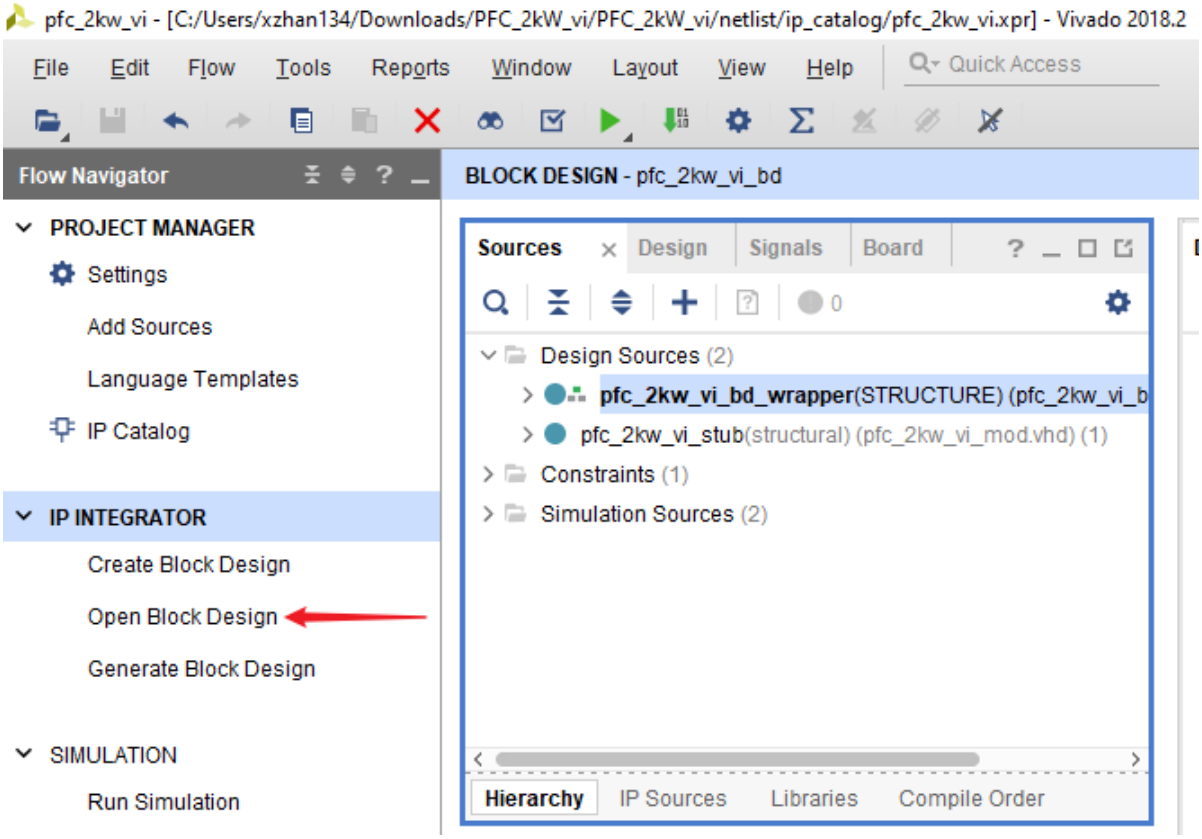
If you do not use FPGA, then you can open Vivado and create a project.



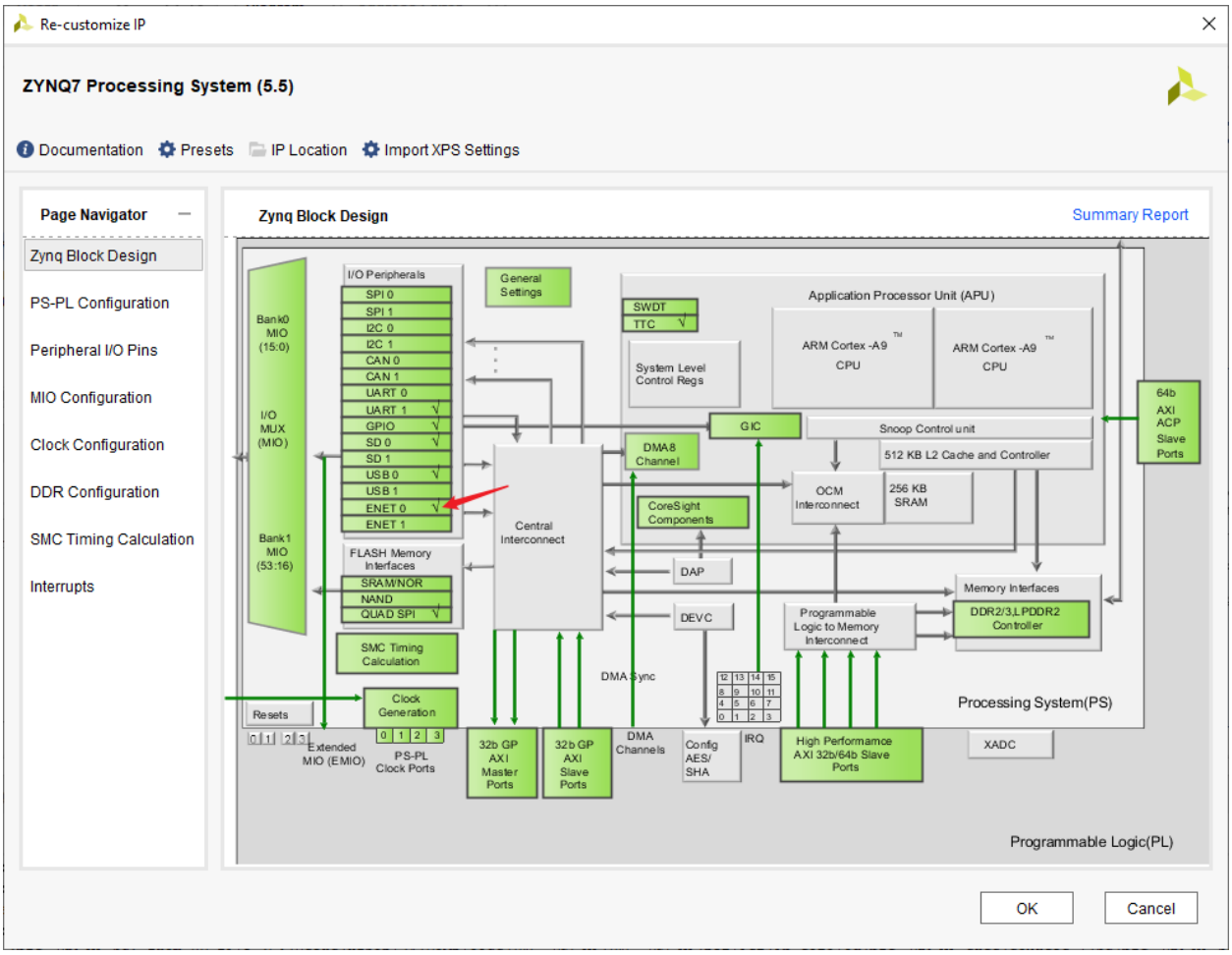
Name	Date modified	Type	Size
.Xil	1/29/2021 6:10 PM	File folder	
pfc_2kw_vi.cache	1/29/2021 6:11 PM	File folder	
pfc_2kw_vi.hw	1/31/2021 2:15 PM	File folder	
pfc_2kw_vi.ip_user_files	1/29/2021 6:10 PM	File folder	
pfc_2kw_vi.runs	1/29/2021 6:11 PM	File folder	
pfc_2kw_vi.sim	1/29/2021 6:10 PM	File folder	
pfc_2kw_vi.srcs	1/29/2021 6:10 PM	File folder	
pfc_2kw_vi.xpr	1/30/2021 4:19 PM	Vivado Project File	20 KB

Step 3: Configure Zedboard

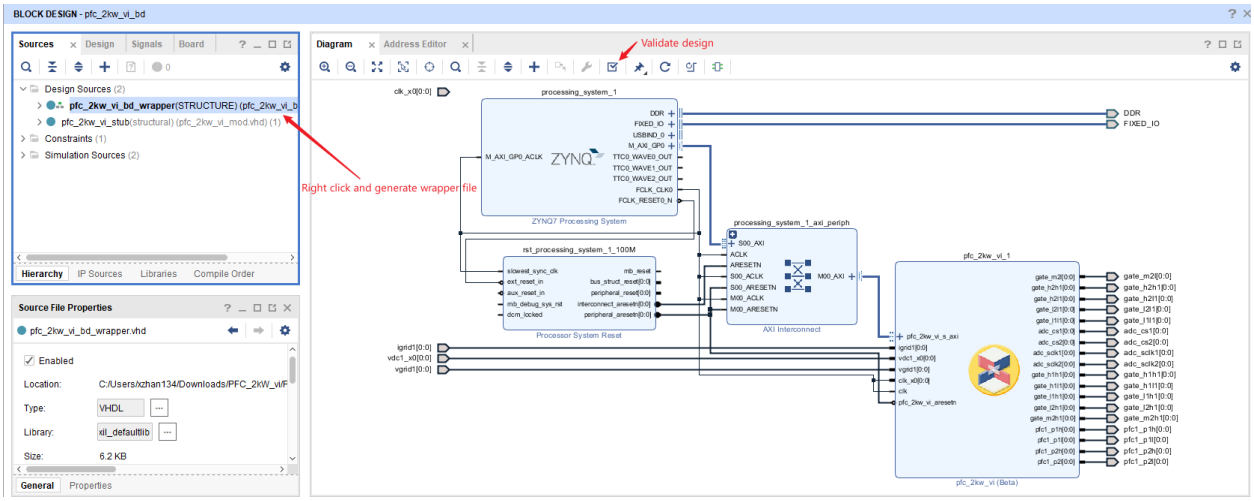
Open block design.



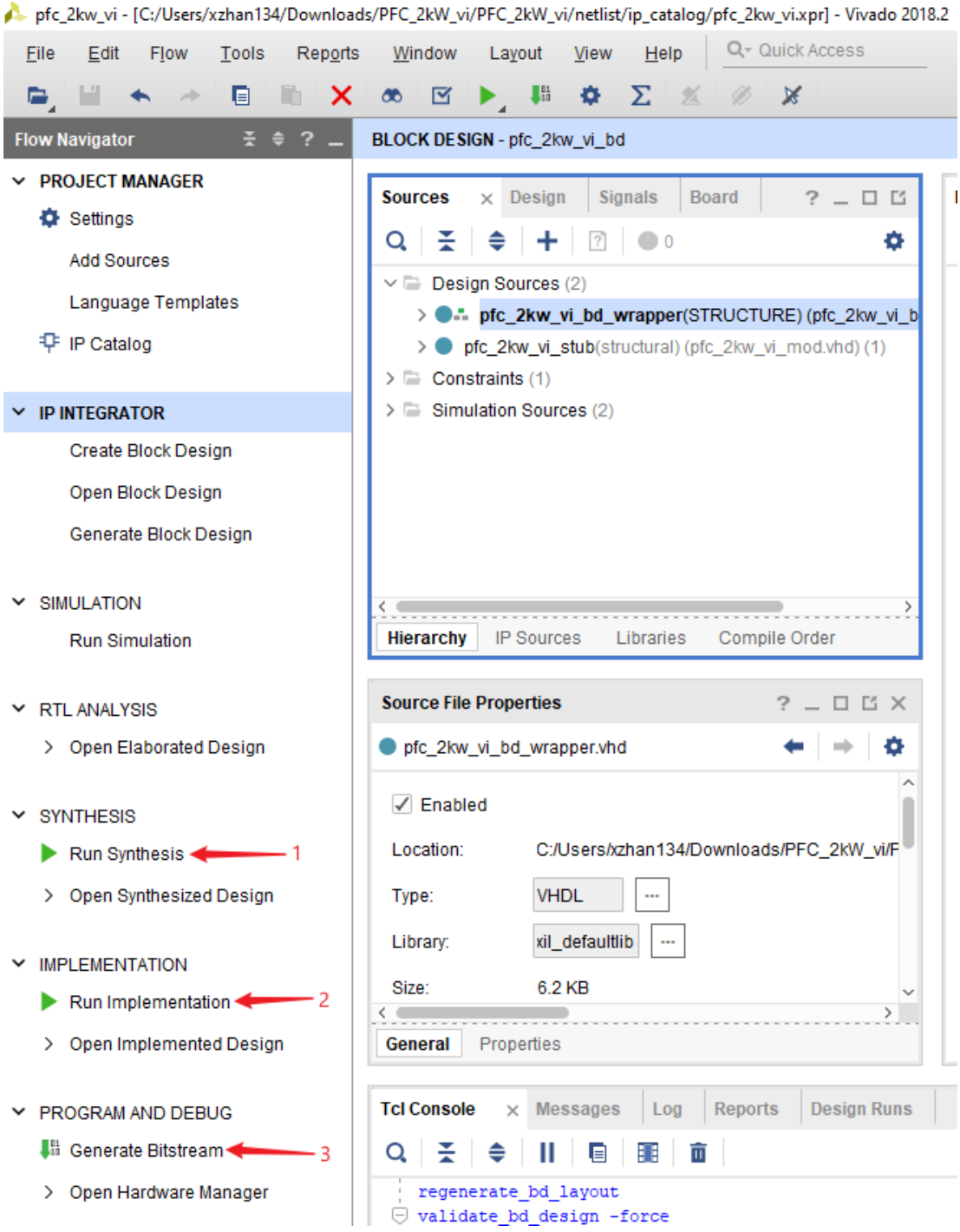
Using the default board setting is OK and you can also customize your board. The Ethernet port is necessary and should be checked in the system configuration.



Next, validate design and generate the wrapper file.

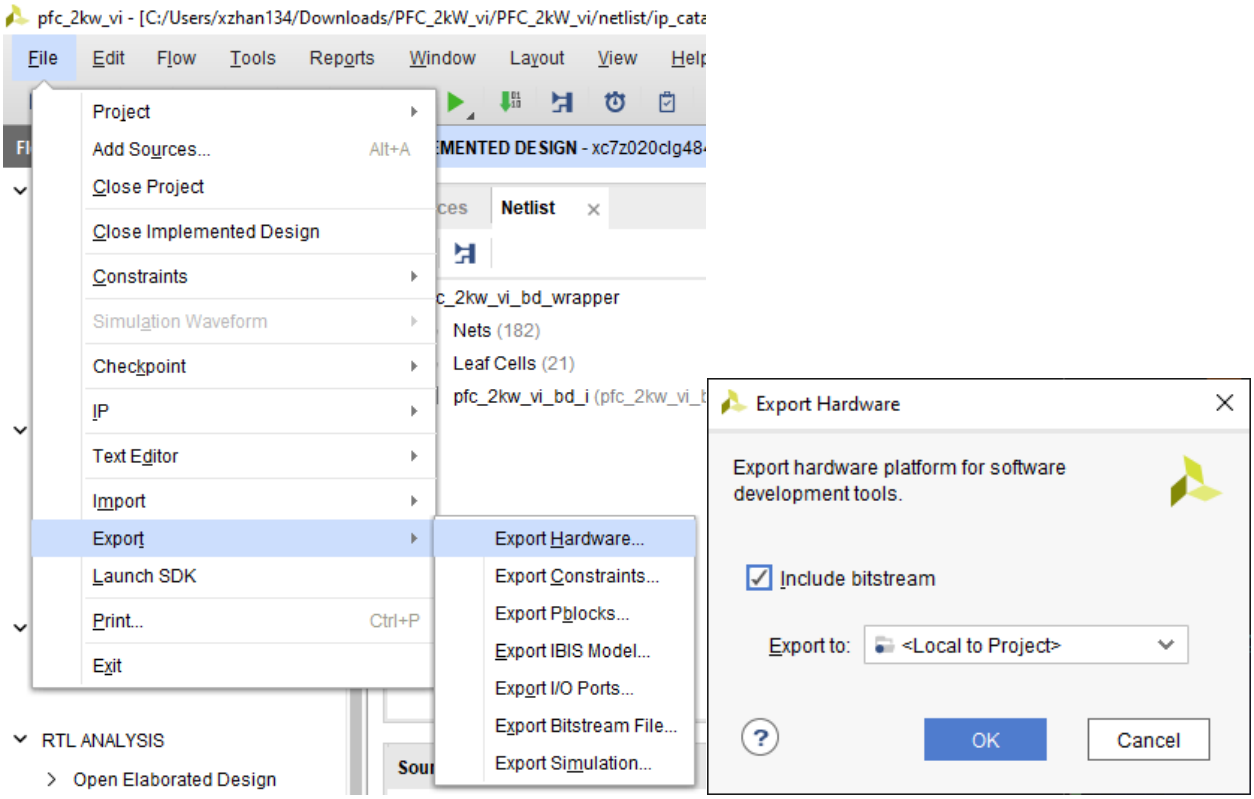


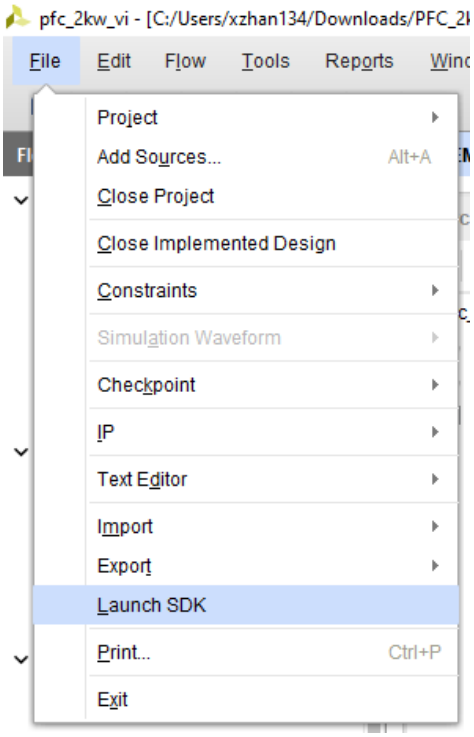
Step 4: Run Synthesis, Run Implementation and Generate Bitstream



Step 5: Export Hardware and Launch SDK

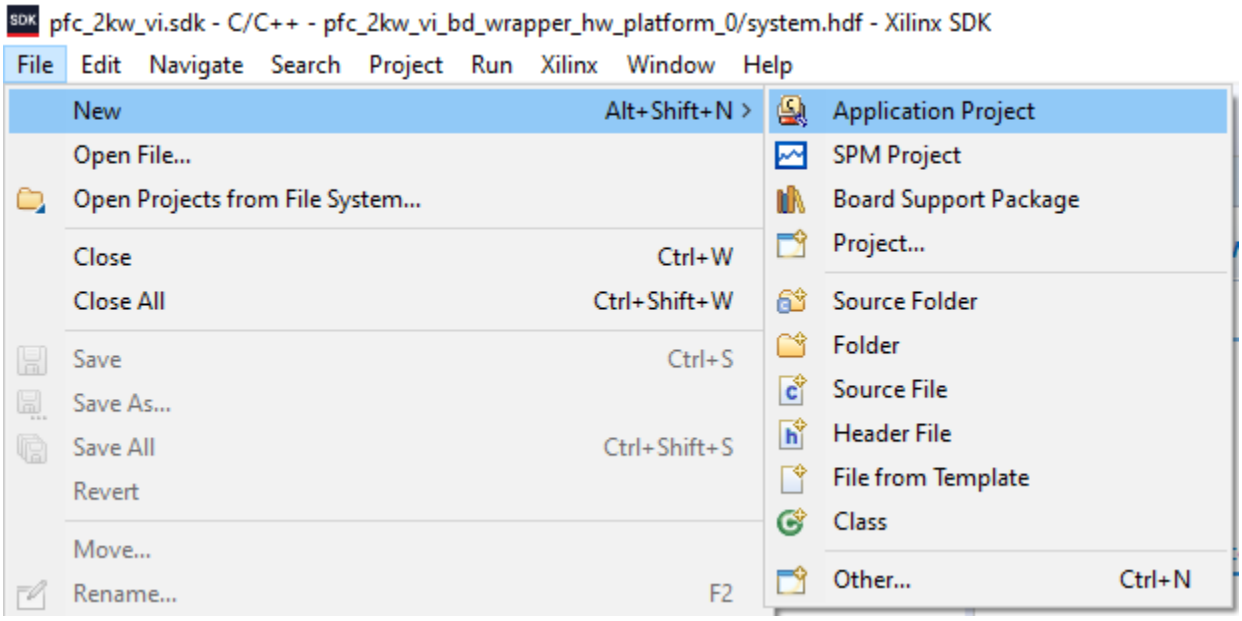
Export hardware and remember to check “Include bitstream”. Next, launch SDK.

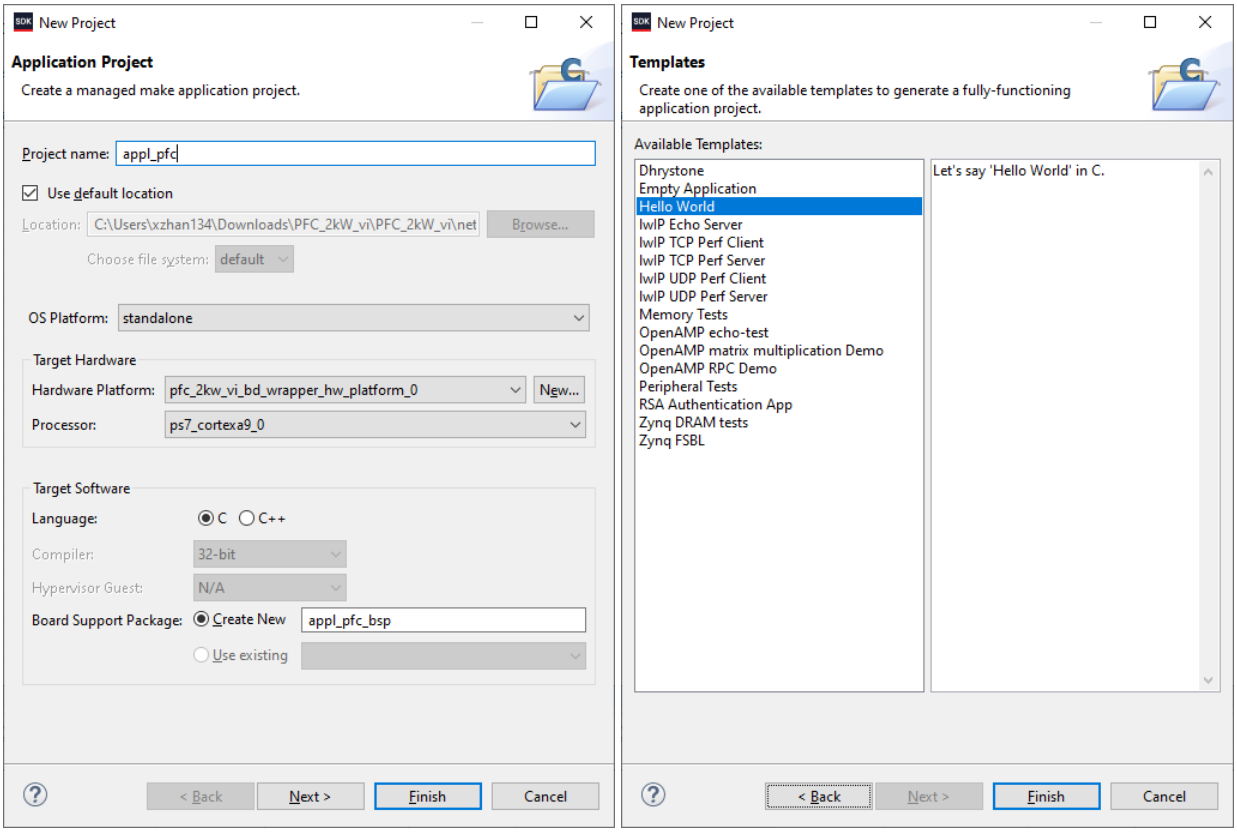




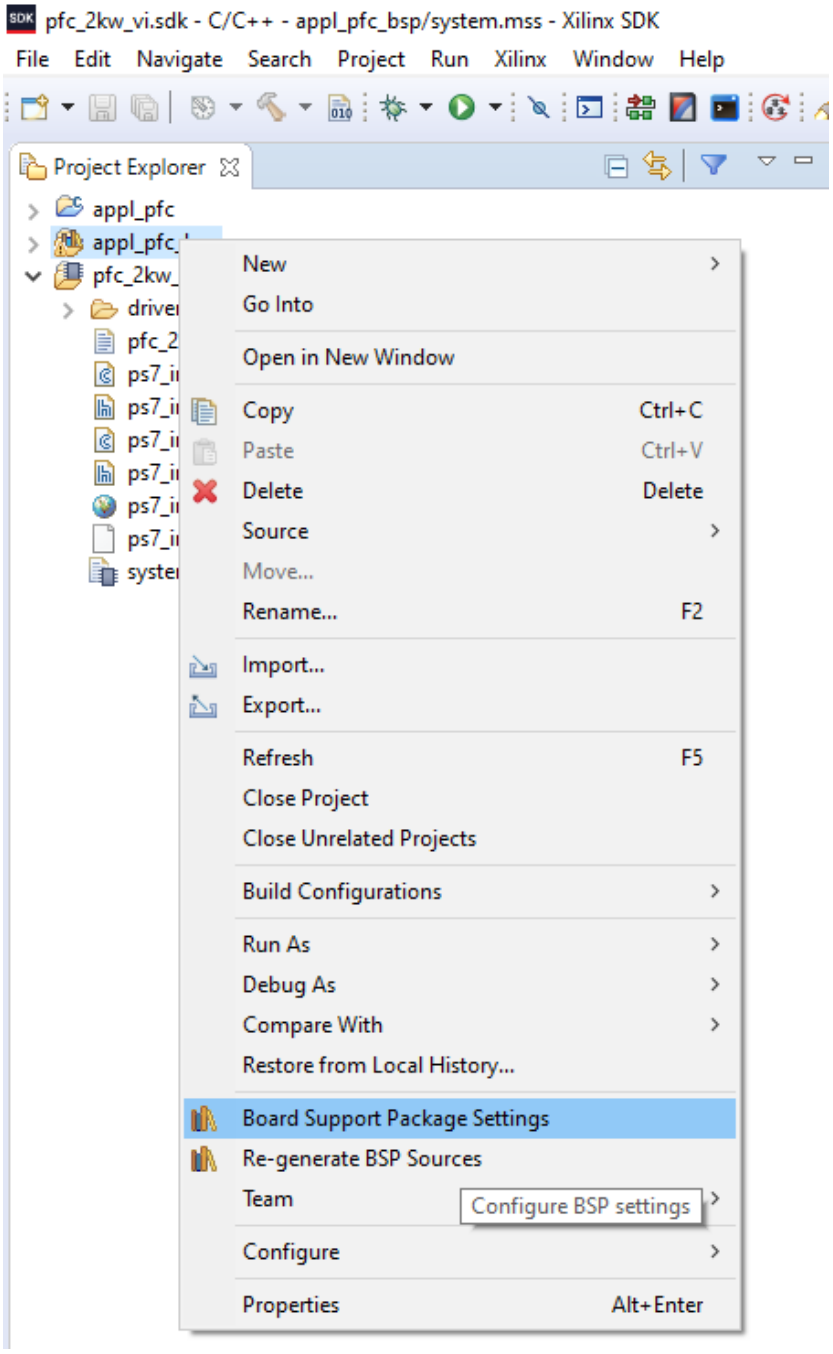
Step 6: Create Application Project and Board Support Package (BSP)

Create an application project with a board support package.

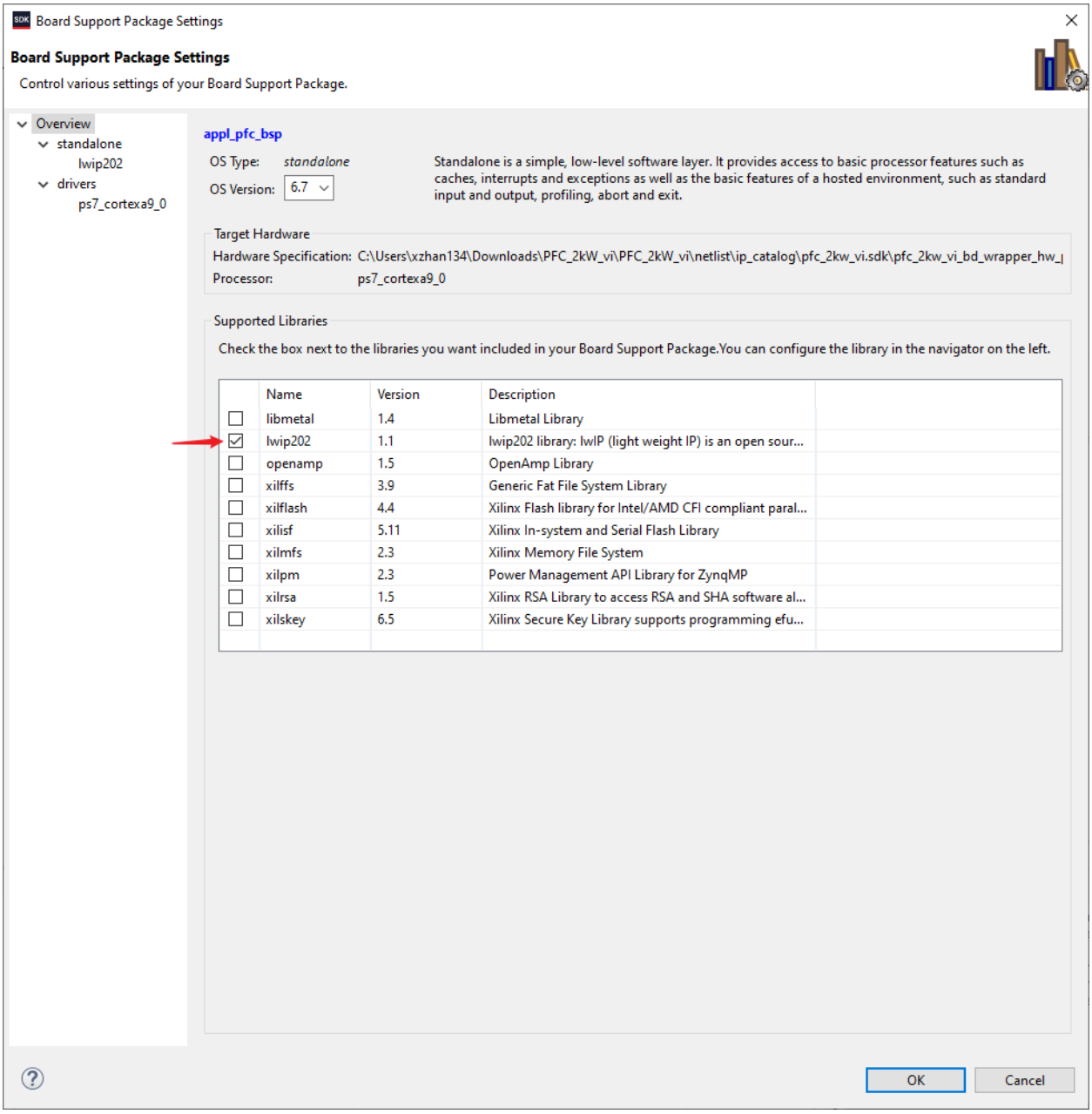




Right click BSP and open “Board Support Package Settings”.

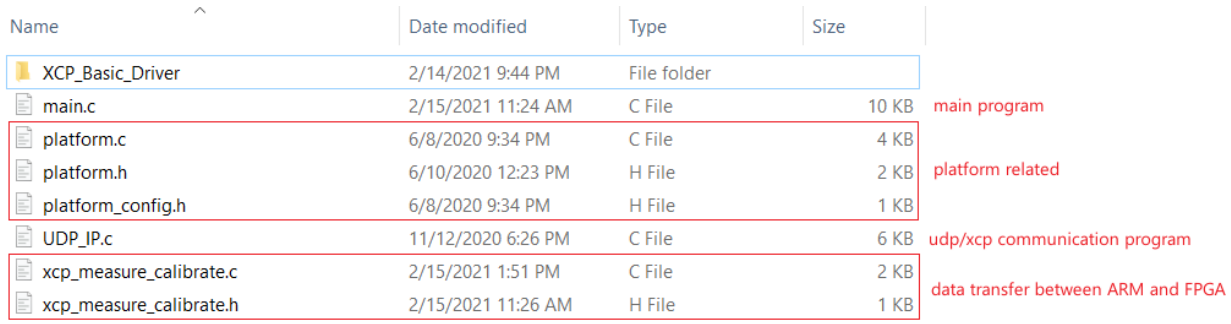


Check **lwip202** library and click “OK”. The BSP will be re-generated.


Board Support Package Settings
Board Support Package Settings
Control various settings of your Board Support Package.
Overview
standalone
lwip202
drivers
ps7_cortexa9_0
appl_pfc_bsp
OS Type: standalone
OS Version: 6.7
Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts and exceptions as well as the basic features of a hosted environment, such as standard input and output, profiling, abort and exit.
Target Hardware
Hardware Specification: C:\Users\xzhan134\Downloads\PFC_2kW_vi\PFC_2kW_vi\netlist\ip_catalog\pfc_2kw_vi.sdk\pfc_2kw_vi_bd_wrapper_hw_
Processor: ps7_cortexa9_0
Supported Libraries
Check the box next to the libraries you want included in your Board Support Package.You can configure the library in the navigator on the left.
Name
Version
Description
libmetal 1.4 Libmetal Library
lwip202 1.1 lwip202 library: lwIP (light weight IP) is an open sour...
openamp 1.5 OpenAmp Library
xilffs 3.9 Generic Fat File System Library
xilflash 4.4 Xilinx Flash library for Intel/AMD CFI compliant paral...
xilisf 5.11 Xilinx In-system and Serial Flash Library
xilmfs 2.3 Xilinx Memory File System
xilpm 2.3 Power Management API Library for ZynqMP
xilrsa 1.5 Xilinx RSA Library to access RSA and SHA software al...
xilskey 6.5 Xilinx Secure Key Library supports programming efu...
OK
Cancel

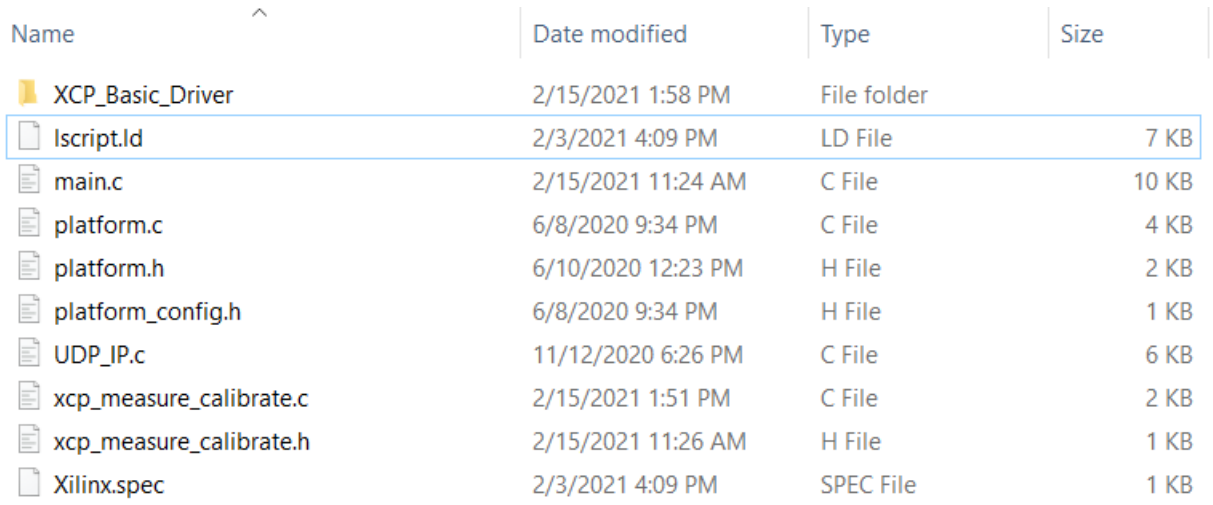
Step 7: Copy the Sample Program to the Application Folder, Add Directory and Configure the Compiler

Copy all the files in the “Sample Program”.



Paste in the SDK project under the following directory:

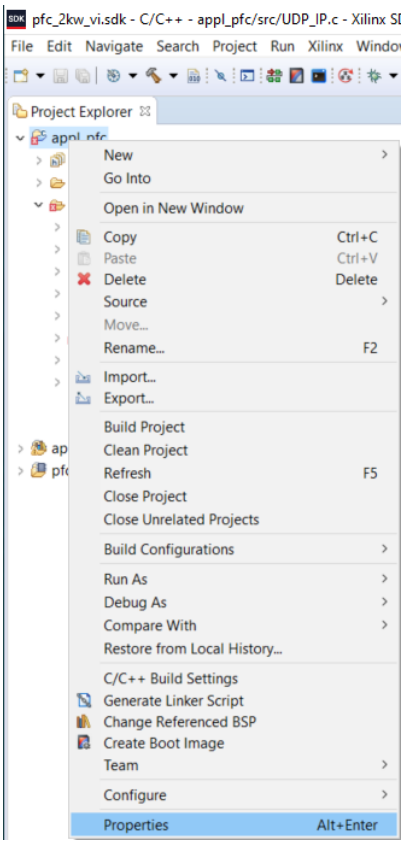
.\PFC_2kW_vi\PFC_2kW_vi\netlist\ip_catalog\pfc_2kw_vi.sdk\appl_pfc\src

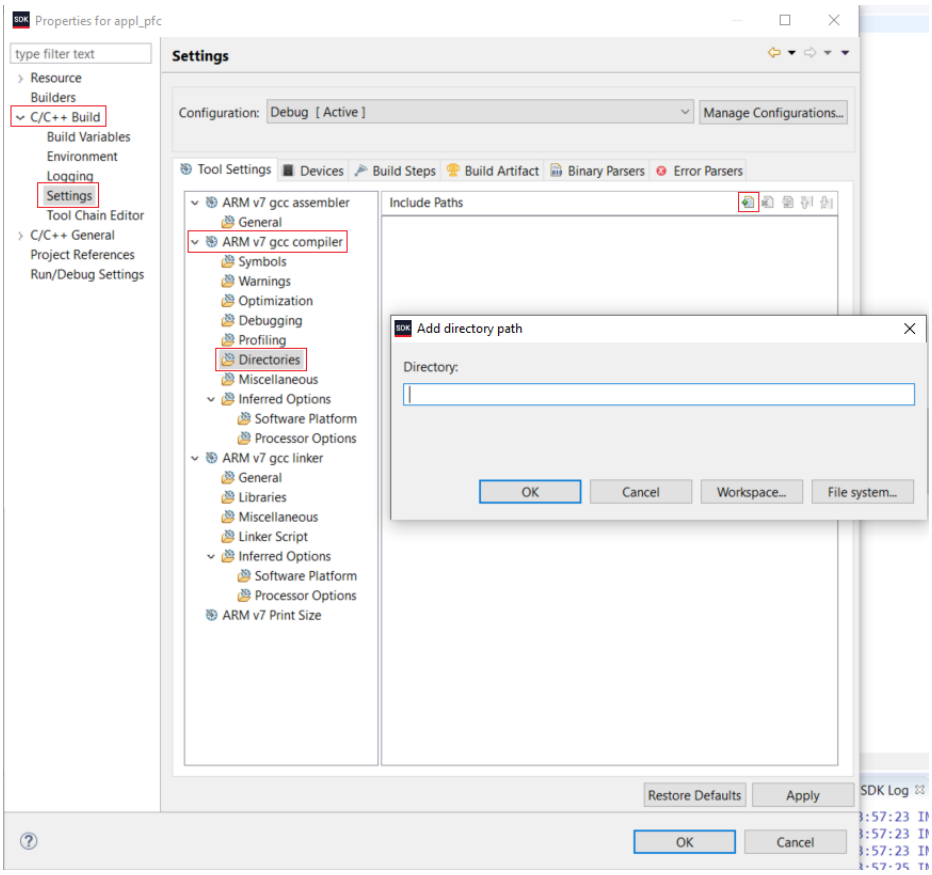


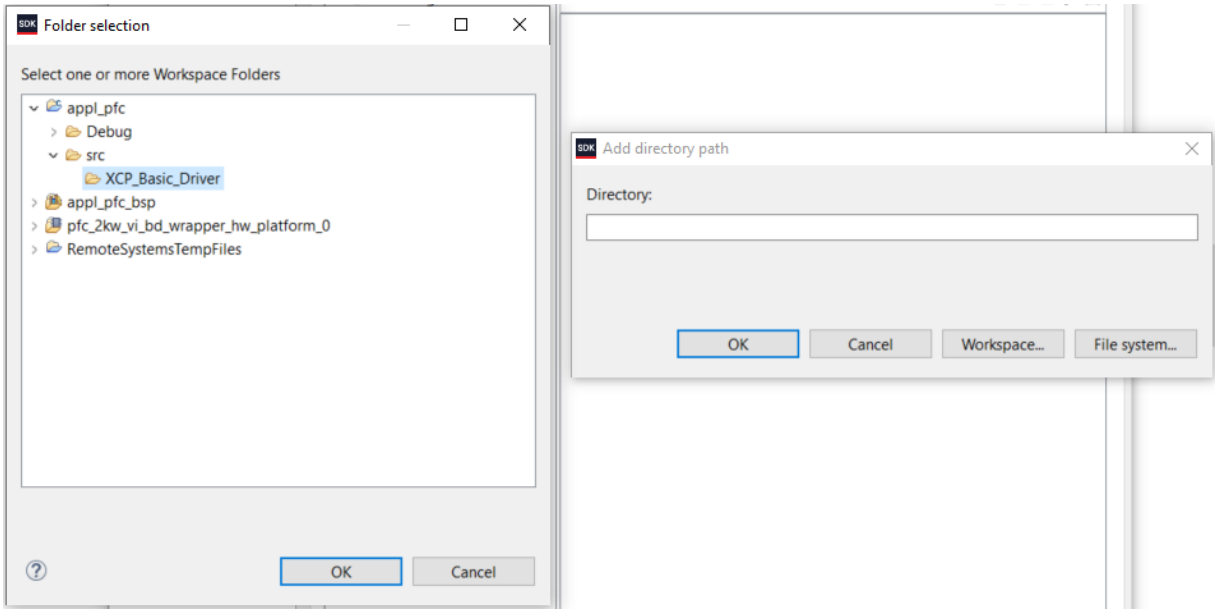
When pasting the files, you will see a message showing whether to replace the existing files. Choose replacing all the files.

The next step is to add directory for the compiler so that the gcc compiler knows the location of header files in the application folder. Right-click the application folder and click “**Properties**”. Then in the following window, click “**C/C++ Build**”, “**Settings**”, “**ARM v7 gcc compiler**”, “**Directories**” and add path of the header files.

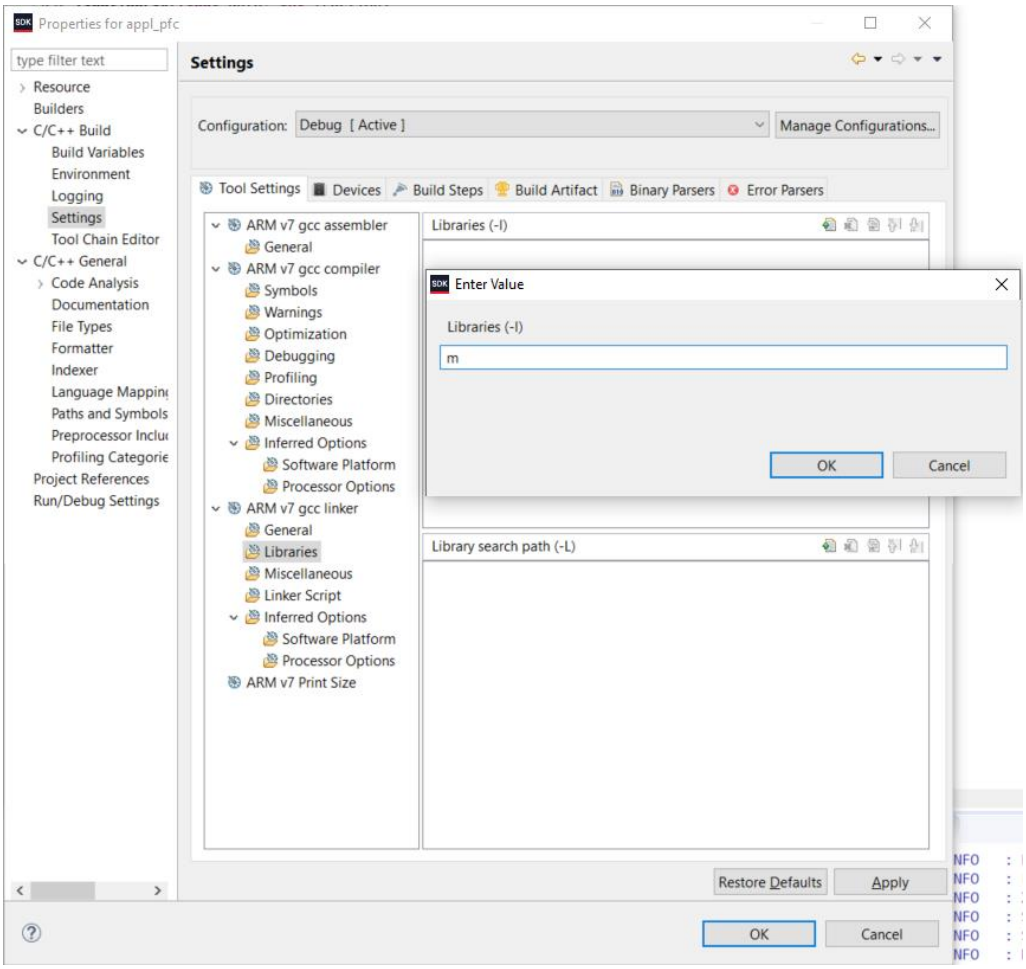
In the following window, click “**Workspace…**” and choose “**XCP_Basic_Driver**”, then click “**OK**” to add the include path. Do the same operation again but at this time choose “**src**”. Once you add the include path, click “**Apply**” or “**OK**”, the program will recompile.


pfc_2kw_vi.sdk - C/C++ - appl_pfc/src/UDP_IP.c - Xilinx SI
File Edit Navigate Search Project Run Xilinx Windo
Project Explorer
appl_pfc
New
Go Into
Open in New Window
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Source
Move...
Rename... F2
Import...
Export...
Build Project
Clean Project
Refresh F5
Close Project
Close Unrelated Projects
Build Configurations
Run As
Debug As
Compare With
Restore from Local History...
C/C++ Build Settings
Generate Linker Script
Change Referenced BSP
Create Boot Image
Team
Configure
Properties Alt+Enter


Properties for appl_pfc
type filter text
Resource
Builders
C/C++ Build
Build Variables
Environment
Logging
Settings
Tool Chain Editor
C/C++ General
Project References
Run/Debug Settings
Settings
Configuration: Debug [Active]
Manage Configurations...
Tool Settings
Devices
Build Steps
Build Artifact
Binary Parsers
Error Parsers
ARM v7 gcc assembler
General
ARM v7 gcc compiler
Symbols
Warnings
Optimization
Debugging
Profiling
Directories
Miscellaneous
Inferred Options
Software Platform
Processor Options
ARM v7 gcc linker
General
Libraries
Miscellaneous
Linker Script
Inferred Options
Software Platform
Processor Options
ARM v7 Print Size
Include Paths
Add directory path
Directory:
OK
Cancel
Workspace...
File system...
Restore Defaults
Apply
OK
Cancel
SDK Log



Another step that needs to do is to configure the compiler. The gcc compiler doesn't link "**math.h**" library by default. To configure this, add "m" under "**ARM v7 gcc linker\Libraries**" as shown in the following figure.



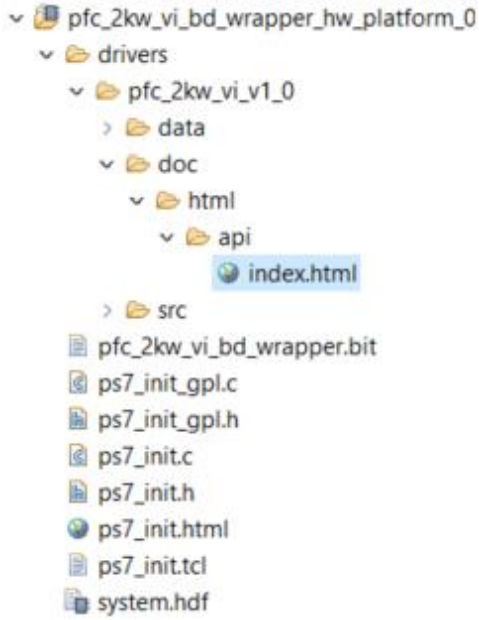
Note: there is a bug in the Xilinx SDK 2018.2 version related to LWIP library. Sometimes the compiler cannot find the path of this library although it does exist in the BSP.

For such case, try to regenerate the application and BSP again, and this issue may be solved. Or you can use a newer version of Xilinx SDK.

Step 8: FPGA Configuration and Setup

If you don't use FPGA part, then you can jump to the **step 10**.

If you use FPGA, after the synthesis and launching the SDK, you will find the hardware platform in the Project Explorer. Under the directory "**<name of the hardware platform>\drivers\<FPGA model name>\doc\api\index.html**", you can find the documentation of the FPGA APIs.



The following figure shows a glimpse of the documentation. By using this APIs, you can access the input and output gateways you defined in the FPGA model.

For output gateways, they are only-readable so you can read values from those gateways but you cannot write to them (read API available). For input gateways, you will find two APIs for one gateway: one is used to write values to that gateway, another is used to read value from that gateway.

Each API name is composed of FPGA model name and gateway name in the model plus read/write. Input datatype and output datatype are also specified in the documentation.

int pfc_2kw_vi_Initialize(pfc_2kw_vi* InstancePtr, u16 DeviceId)

Initialize pfc_2kw_vi pointer with a specifc instance of the device based device id.
@param InstancePtr to be initialized
@param DeviceId device id assigned to the device in xparameters.h

@return int XST_SUCCESS if initialization is successful

void pfc_2kw_vi_vdc_ref_write(pfc_2kw_vi *InstancePtr, u32 Data);

Write to vdc_ref gateway of pfc_2kw_vi. Assignments are LSB-justified.

@param InstancePtr is the vdc_ref instance to operate on.
@param Data is value to be written to gateway vdc_ref.

@return None.

@note .

u32 pfc_2kw_vi_vdc_ref_read(pfc_2kw_vi *InstancePtr);

Read from vdc_ref gateway of pfc_2kw_vi. Assignments are LSB-justified.

@param InstancePtr is the vdc_ref instance to operate on.

@return u32
@note .

void pfc_2kw_vi_vdc1_offset_write(pfc_2kw_vi *InstancePtr, u32 Data);

Write to vdc1_offset gateway of pfc_2kw_vi. Assignments are LSB-justified.

@param InstancePtr is the vdc1_offset instance to operate on.
@param Data is value to be written to gateway vdc1_offset.

@return None.

@note .

To execute the FPGA model along with your C code, you need to initialize the model in your program.

To begin with, the header files of FPGA needs to be included. Go through the BSP, you can find two header files containing the name of your FPGA model. In the program, I also highlight the location where you need to include the header file. We only need **pfc_2kw_vi.h** in the application. Add the FPGA header file in "**main.c**" and "**xcp_measure_calibration.h**" files.

The next is to declare the FPGA object in "**main.c**". You may have found that in the API documentation, in every function, there is a pointer named "***InstancePtr**". This pointer actually points to the FPGA model itself.

```
/* declare FPGA variable here... */
/* example
###########################################
cm cm_instance;
###########################################
*/
pfc_2kw_vi pfc_instance;
```

The initialization will be done in the main function in the "**main.c**" file.
"**XPAR_PFC_2KW_VI_0_DEVICE_ID**" is the device ID of the FPGA model. You can find it in "**xparameters.h**".

```
/* init FPGA here... */
/* example
########################################
status = cm_Initialize(&cm_instance, XPAR_CM_1_DEVICE_ID);
########################################
*/
status = pfc_2kw_vi_Initialize(&pfc_instance, XPAR_PFC_2KW_VI_0_DEVICE_ID);
if (status != XST_SUCCESS)
{
    xil_printf("FPGA initialization failed!\r\n");
    return XST_FAILURE;
}
else
    xil_printf("FPGA initialization succeed!\r");
```

Now, the FPGA should work!

Step 9: Measurement/Calibration in C Code

Note that the FPGA model can work now, but it doesn't mean everything at the software side is finished. CANape communicates with the Xilinx Zedboard through an Ethernet port but this happens at the ARM (microprocessor) side.

For measurement and calibration of FPGA input and output gateways, communication between ARM and FPGA is required. This communication can be simply done by using autogenerated APIs. Communication codes are written in the "xcp_measure_calibrate.c" and "xcp_measure_calibrate.h".

Depending on the actual FPGA model, this part of code needs to be modified by users. To keep consistent with the types of variables in CANape, let's classify the variables into measurement and parameter/characteristic. Measurement represents the variables that are only readable (outputs). Parameter/characteristic represents the variables that are not only readable but also writeable (inputs).

Users need to define the measurement and parameter/characteristic in the program as shown in the following figure. These variables are global and the data types should fit the real value data types. To use FPGA APIs, the FPGA object is required. The object is already declared in "**main.c**" so here we use key word "**extern**" to specify this object variable is declared somewhere else.

```
// measurement variables declare
/* add measurement variables here...
############################
static float xcp_ia, xcp_ib, xcp_ic;
############################
*/
static int kp_read;
static int ki_read;

// characteristics variables declare
/* add characteristics variables here...
############################
static u8 xcp_pi_rst = 1;
############################d
*/
static int kp;
static int ki;

// FPGA
/* add extern FPGA variables here...
############################
extern cm cm_instance;
############################d
*/
extern pfc_2kw_vi pfc_instance;
```

In "**xcp_measure_calibrate.c**", there are three functions "**calibarteVarsInit()**", "**xcpVarsRead100us()**" and "**xcpVarsUpdate()**".

"**calibarteVarsInit()**" is used to initialize inputs in the FPGA model. Users should manually use APIs and add code in this function.

"**xcpVarsRead100us()**" is used for measurement purpose. A timer with 100us period will call this function to update all the measurement defined in the C code with the outputs from FPGA model (the timer is setup and configured in the sample program). Users need to manually add code in this function to use APIs to do the update.

"**xcpVarsUpdate()**" is used for calibration purpose. In the sample program, this function is called for every 2ms. Users need to add code in this function to specify which variables will be updated from ARM to FPGA.

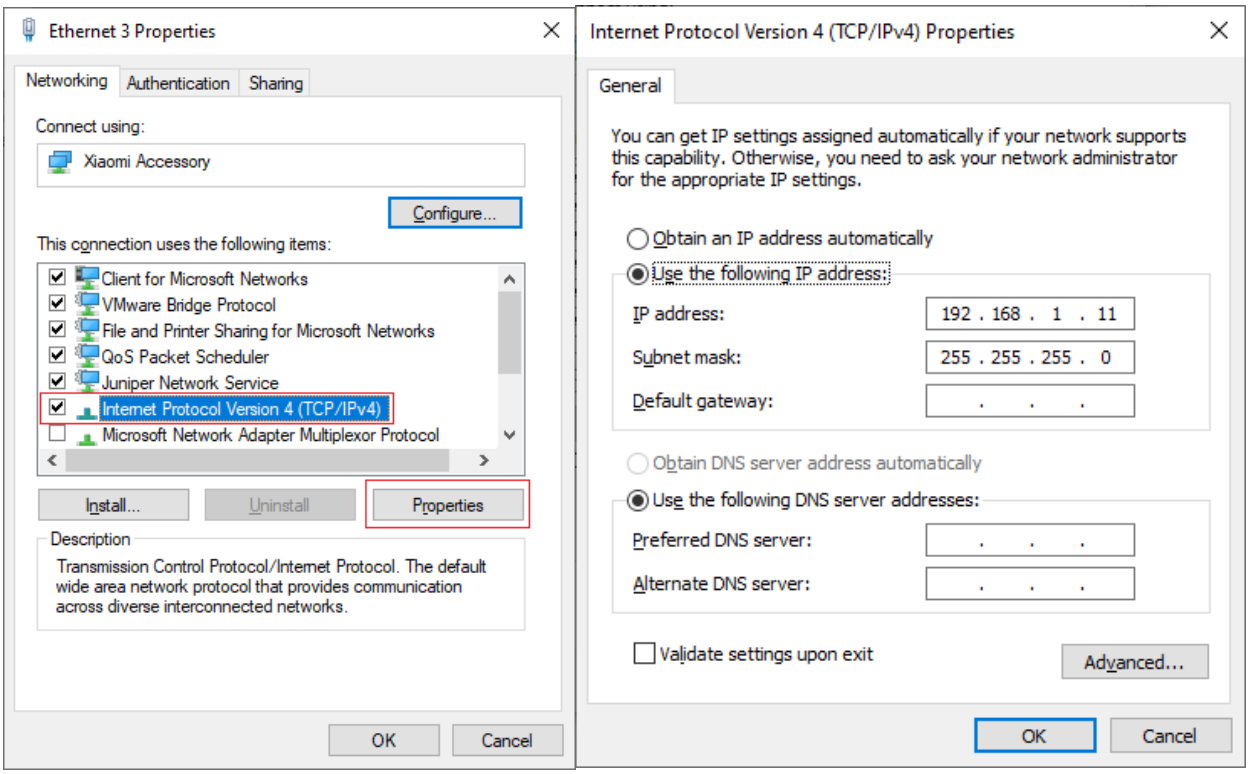
The following figure shows an example. "**calibarteVarsInit()**" assigns the values in kp and ki to the input gateway pi_v_kp and pi_v_ki in the FPGA model. "**xcpVarsRead100us()**" assigns the values at pi_v_kp and pi_v_ki gateways to the variables kp_read and ki_read in the program for every 100us. "**xcpVarsUpdate()**" is similar to "**calibarteVarsInit()**". It updates pi_v_kp and pi_v_ki in the FPGA model by using the values in variables kp and ki for every 2ms.

```
int calibarteVarsInit()
{
    /* Init input values in FPGA model here... */
    pfc_2kw_vi_pi_v_kp_write(&pfc_instance, kp);
    pfc_2kw_vi_pi_v_ki_write(&pfc_instance, ki);
    return 0;
}

int xcpVarsRead100us()
{
    /* read output data from FPGA */
    kp_read = pfc_2kw_vi_pi_v_kp_read(&pfc_instance);
    ki_read = pfc_2kw_vi_pi_v_ki_read(&pfc_instance);
    return 0;
}

int xcpVarsUpdate()
{
    /* update */
    pfc_2kw_vi_pi_v_kp_write(&pfc_instance, kp);
    pfc_2kw_vi_pi_v_ki_write(&pfc_instance, ki);
    return 0;
}
```

Again, if the FPGA part is not used, the communication between FPGA and microprocessor (ARM) is not necessary and this step can be skipped.

Step 10: Ethernet Port Configuration

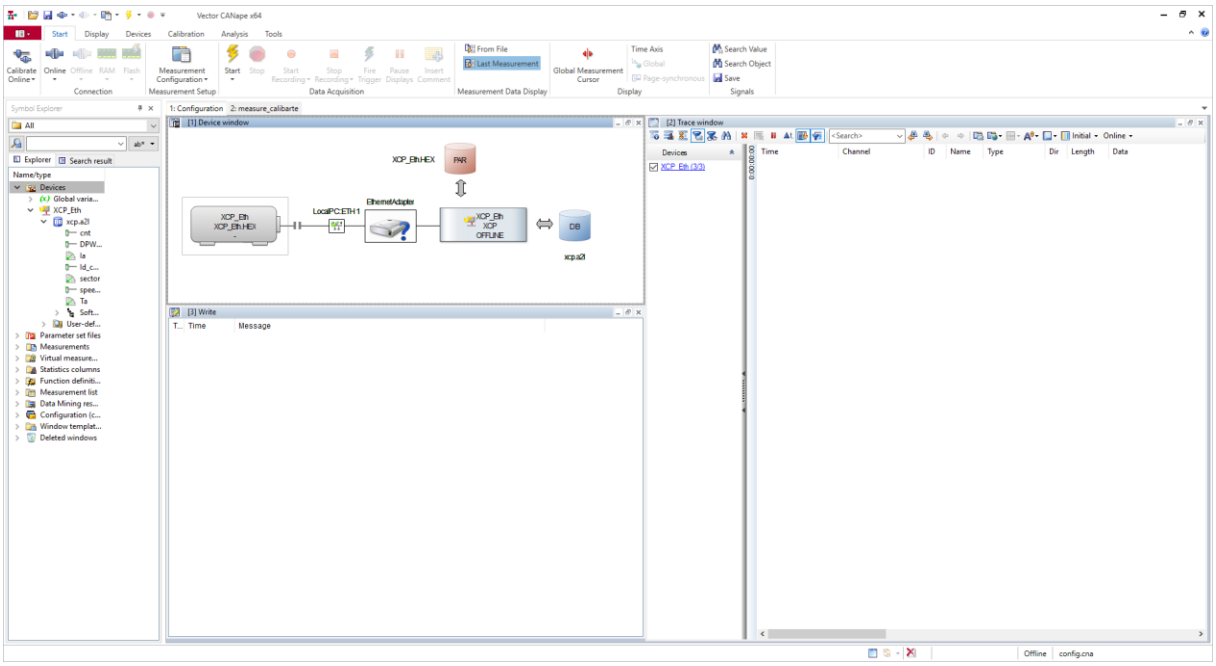
To successfully establish the communication between ARM and CANape through the ethernet. The port at PC side also needs to be configured. In Network Connections (Windows), right-click the ethernet port that the board connects to and open the properties. Then choose **Internet Protocol Version 4 (TCP/IPv4)** and open the properties. Change to use static IP address and manually input the address shown in the following figure.



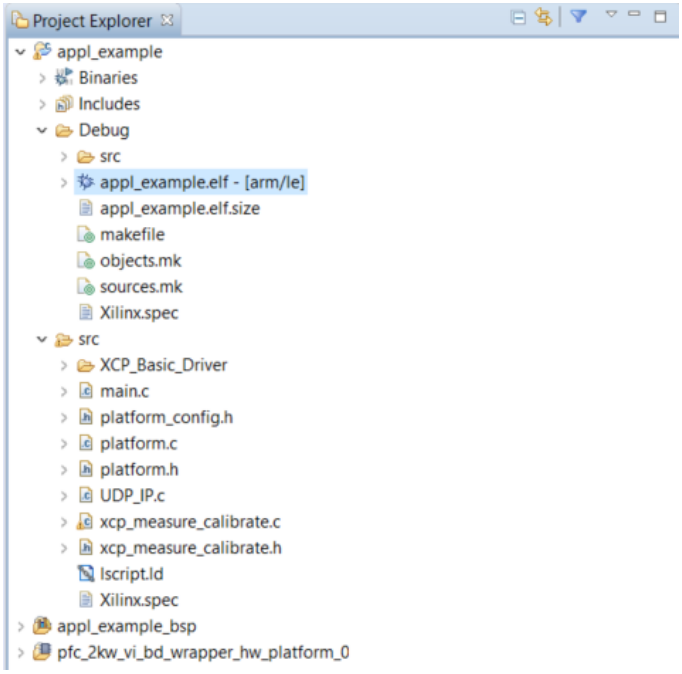
CANape Part

Step 11: CANape Introduction

Finally, we can move to CANape part. The CANape project is already created and it can be found under CANape folder. The following figure shows the CANape interface.



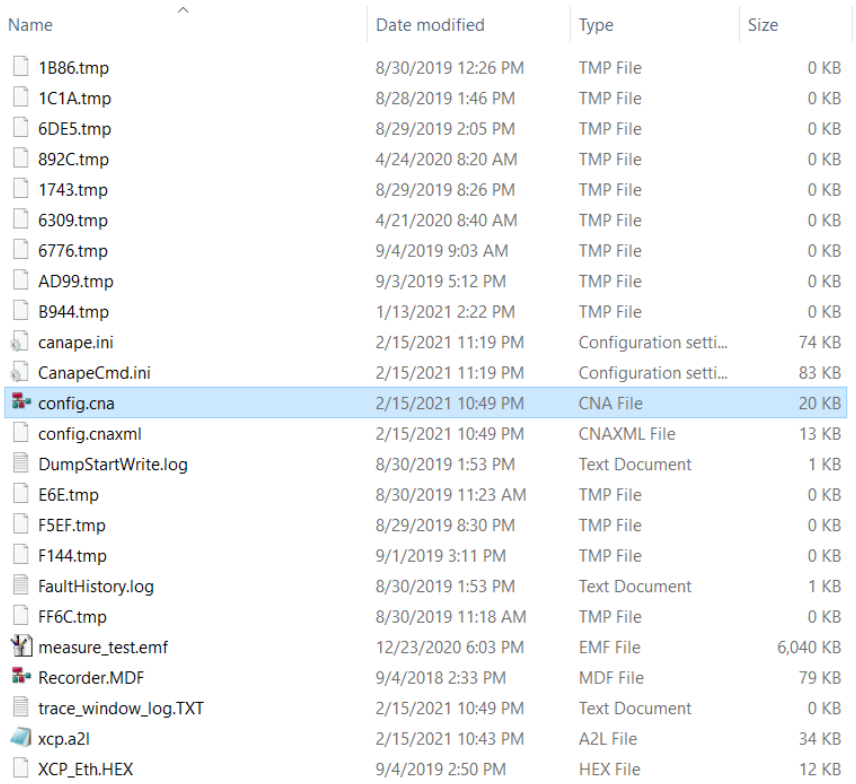
The software requires a database to work called A2L file. To help build the A2L file, an executable file is necessary. An executable file will be automatically generated after the application is successfully compiled in the SDK. The file is named **<application project name>.elf** and it can be found under the path "**<application project name>\Debug\<application project name>.elf**".



Step 12: More about CANape

A CANape project is provided and most configurations have been done already. To use this CANape project for your projects, only a small modification is required which is related A2L file. So, the rest of the manual will mainly focus on the A2L modification and communication between CANape and our Zedboard.

If you can afford a CANape license, then it is much more convenient to modify the A2L file and you don't need to waste time to write the database (A2L).



Name	Date modified	Type	Size
1B86.tmp	8/30/2019 12:26 PM	TMP File	0 KB
1C1A.tmp	8/28/2019 1:46 PM	TMP File	0 KB
6DE5.tmp	8/29/2019 2:05 PM	TMP File	0 KB
892C.tmp	4/24/2020 8:20 AM	TMP File	0 KB
1743.tmp	8/29/2019 8:26 PM	TMP File	0 KB
6309.tmp	4/21/2020 8:40 AM	TMP File	0 KB
6776.tmp	9/4/2019 9:03 AM	TMP File	0 KB
AD99.tmp	9/3/2019 5:12 PM	TMP File	0 KB
B944.tmp	1/13/2021 2:22 PM	TMP File	0 KB
canape.ini	2/15/2021 11:19 PM	Configuration setti...	74 KB
CanapeCmd.ini	2/15/2021 11:19 PM	Configuration setti...	83 KB
config.cna	2/15/2021 10:49 PM	CNA File	20 KB
config.cnaxml	2/15/2021 10:49 PM	CNAXML File	13 KB
DumpStartWrite.log	8/30/2019 1:53 PM	Text Document	1 KB
E6E.tmp	8/30/2019 11:23 AM	TMP File	0 KB
F5EF.tmp	8/29/2019 8:30 PM	TMP File	0 KB
F144.tmp	9/1/2019 3:11 PM	TMP File	0 KB
FaultHistory.log	8/30/2019 1:53 PM	Text Document	1 KB
FF6C.tmp	8/30/2019 11:18 AM	TMP File	0 KB
measure_test.emf	12/23/2020 6:03 PM	EMF File	6,040 KB
Recorder.MDF	9/4/2018 2:33 PM	MDF File	79 KB
trace_window_log.TXT	2/15/2021 10:49 PM	Text Document	0 KB
xcp.a2l	2/15/2021 10:43 PM	A2L File	34 KB
XCP_Eth.HEX	9/4/2019 2:50 PM	HEX File	12 KB

"config.cna" is the CANape project file. Double click it to open the CANape project.

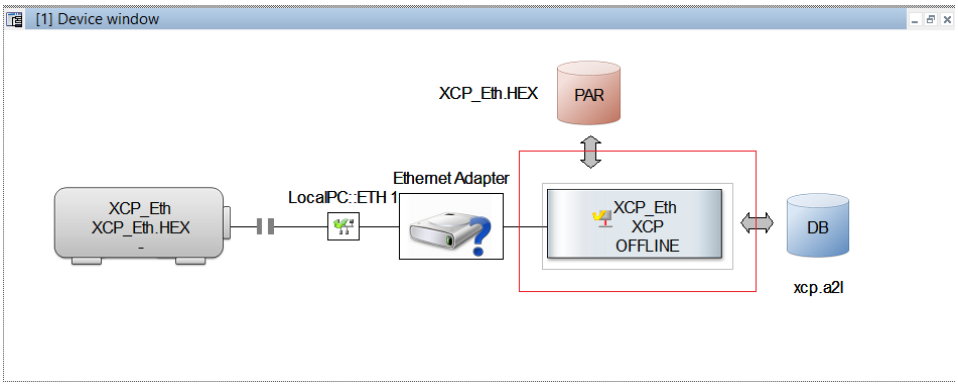
A2L file is the most important component in the CANape project. To communicate with our board (ECU), An A2L file is necessary. This file stores lots of information including the protocol layer information, variable definitions for measurement or calibration, etc. The measurement and calibration variables in the CANape are independent with those in the program. But we can connect the variables together so that variables at two sides are able to synchronize. This connection is done by using the "LINK_MAP" parameter in the **MEASUREMENT** and **CHARACTERISTIC** definitions in the A2L file.

Step 13: Basic Setting for Your Own Project

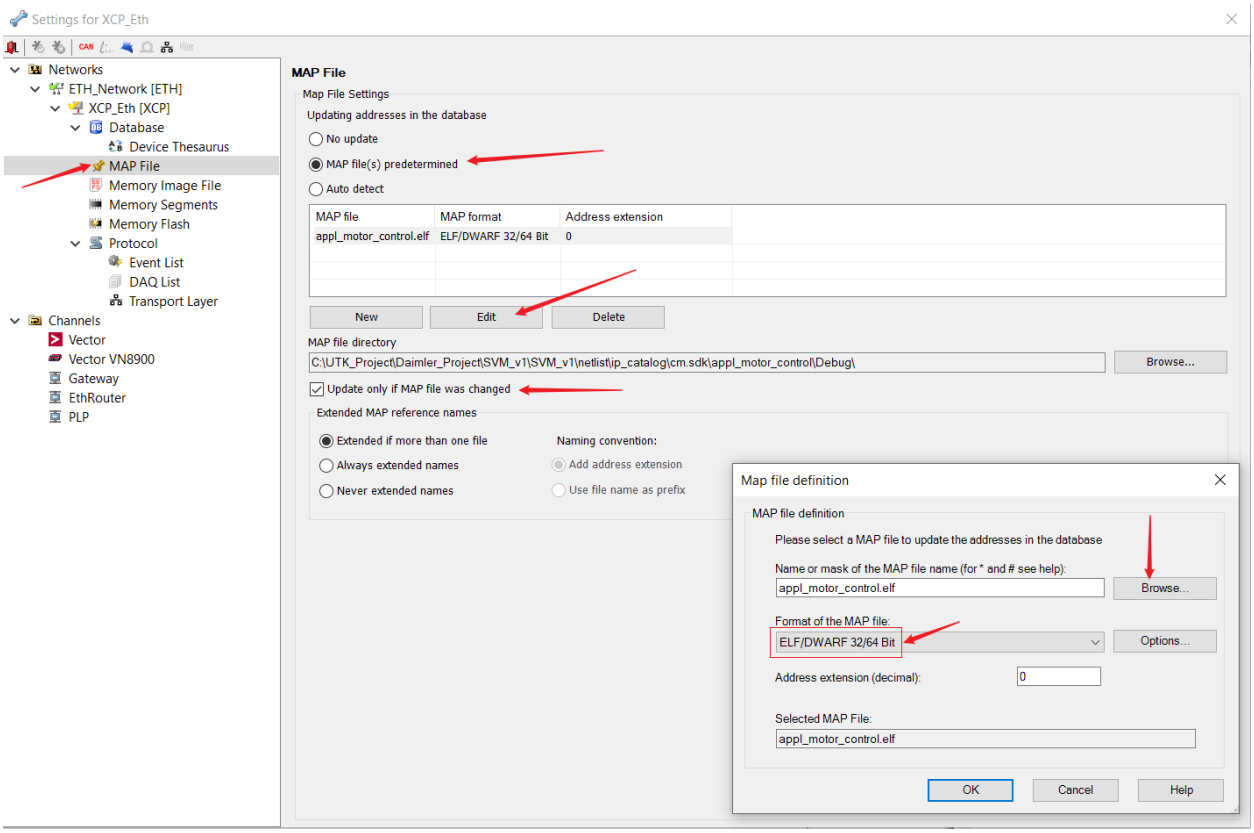
After obtaining the sample CANape project, several steps are required to be done in order to use it in your own project.

For the first time to open this CANape project, the path of the .elf file should be specified in the project settings.

1. Double-click the highlighted component in the following figure.

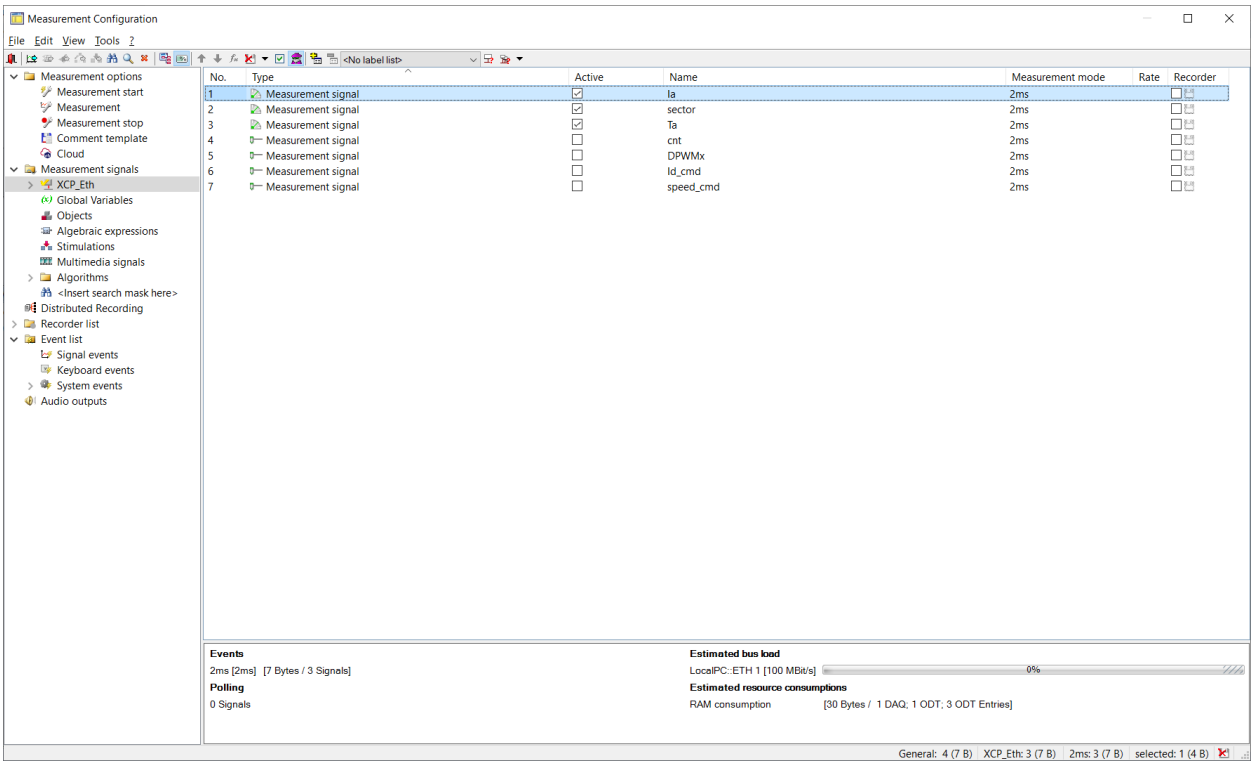


2. In MAP File, edit the predetermined MAP file by giving the directory of your own one. Make sure choose the correct format of the MAP file (**ELF/DWARF 32/64 Bit**)! Check the box to update only if MAP file was changed.

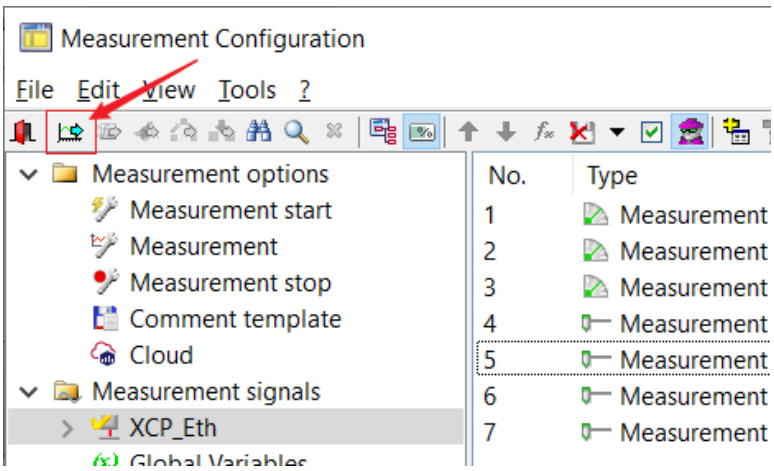


Step 14: Measurement Configuration

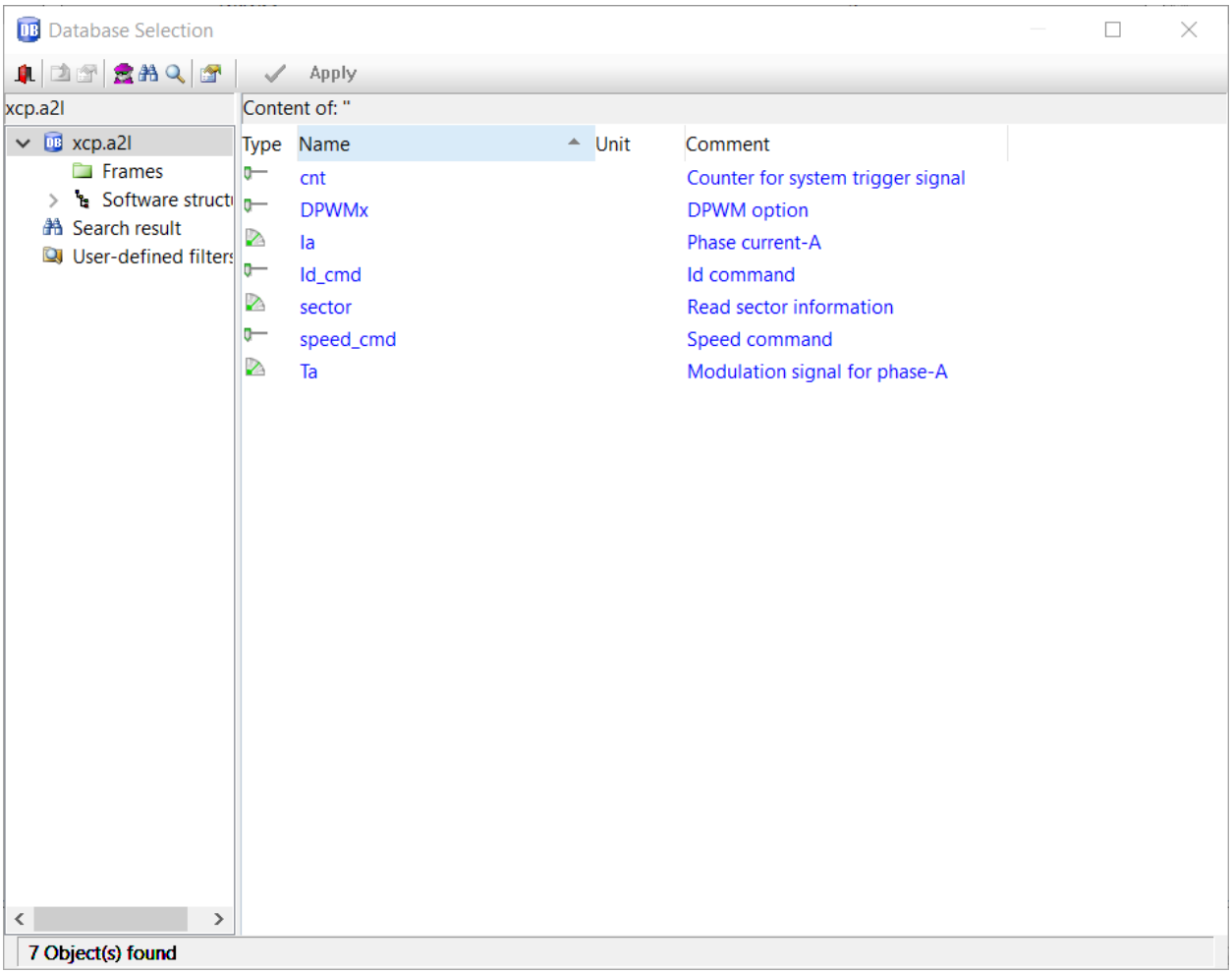
Click “Measurement Configuration”, choose “Signals” to open the measurement configuration window. This window organizes the parameters and measurements added to the CANape project. Measurement signal (Measurement signal) can only be added in the measurement windows while Measurement signal (Parameter) can be added into either measurement windows or calibration windows. Check active box of a signal will enable its display in the measurement windows. Measurement mode specifies the time period of measurement for that signal.



After modifying the A2L file and updating the project, the signals won’t automatically update in this window. That is, it requires manual updates. Right-click the signals in the window and choose delete to remove this signal. To add a signal, click the second icon to insert a new signal into the measurement.



The added signals are highlighted in blue while the non-added signals are black. Choose the signal that you want to add and then click **Apply** or you can also double-click to add that signal.



For current version of software, it supports measurement for 100us (DAQ), 2ms (DAQ) and polling mode. To support different DAQ measurement, the source code needs a small modification and DAQ section in the A2L file also needs modification.

Step 15: Measurement/Characteristic Definition

In A2L file, the measurement is defined in the format shown in the following figure.

```
/begin MEASUREMENT Ia "Phase current-A"
  FLOAT32_IEEE NO_FORMULA 0 0 -50 50
  READ_WRITE
  BYTE_ORDER MSB_LAST
  ECU_ADDRESS 0x20222C
  ECU_ADDRESS_EXTENSION 0x0
  FORMAT "%.3"
  /begin IF_DATA CANAPE_EXT
    100
    LINK_MAP "xcp_ia" 0x20222C 0x0 0 0x0 1 0x1 0x0
    DISPLAY 0 -50 50
  /end IF_DATA
/end MEASUREMENT
```

The definition of the characteristic is shown in the following figure.

```
/begin CHARACTERISTIC Id_cmd "Id command"
  VALUE 0x20225C __FLOAT32_IEEE_S 100 NO_FORMULA 0 2
  ECU_ADDRESS_EXTENSION 0x0
  EXTENDED_LIMITS -300 300
  BYTE_ORDER MSB_LAST
  FORMAT "%.3"
  /begin IF_DATA CANAPE_EXT
    100
    LINK_MAP "xcp_id_cmd" 0x20225C 0x0 0 0x0 1 0x1 0x0
    DISPLAY 0 0 2
  /end IF_DATA
/end CHARACTERISTIC
```

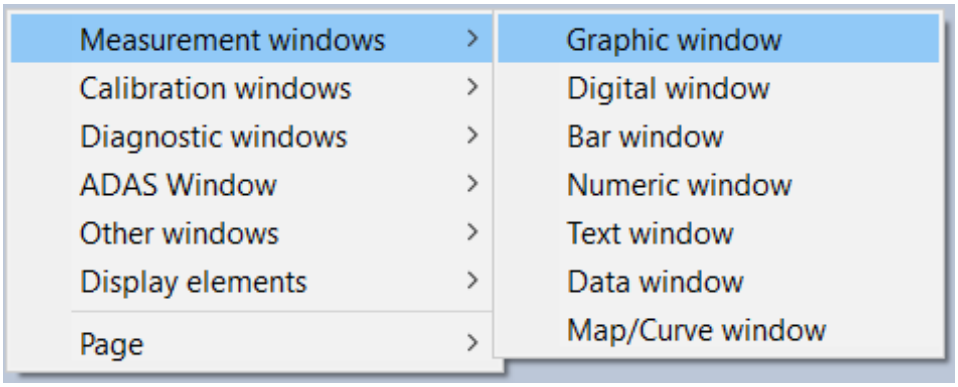
To understand the structure of the variable definitions above and parameters in the definitions, it is helpful to read these two references.

- http://read.pudn.com/downloads200/ebook/942888/ASAP2.pdf
- https://cdn.intrepidcs.net/support/ASAP2Editor/icsASAP2Editor.pdf

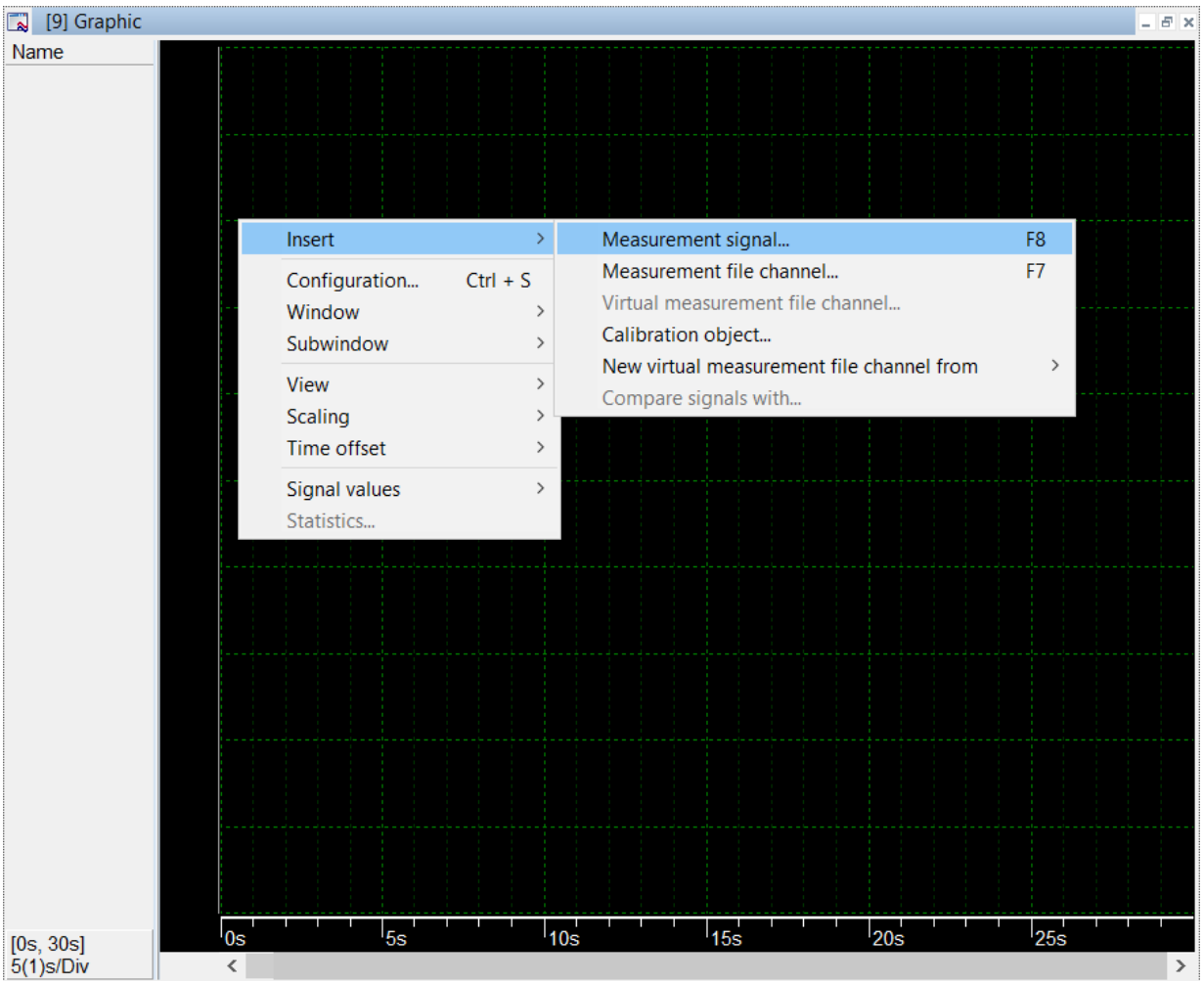
If you have a valid CANape license, you can use its A2L editor – ASAP2 Studio to modify. In ASAP2 Studio, the MAP file is loaded with all the variables available and those variables can be directly added to the A2L file as a measurement or a parameter.

Step 16: Build CANape Interface

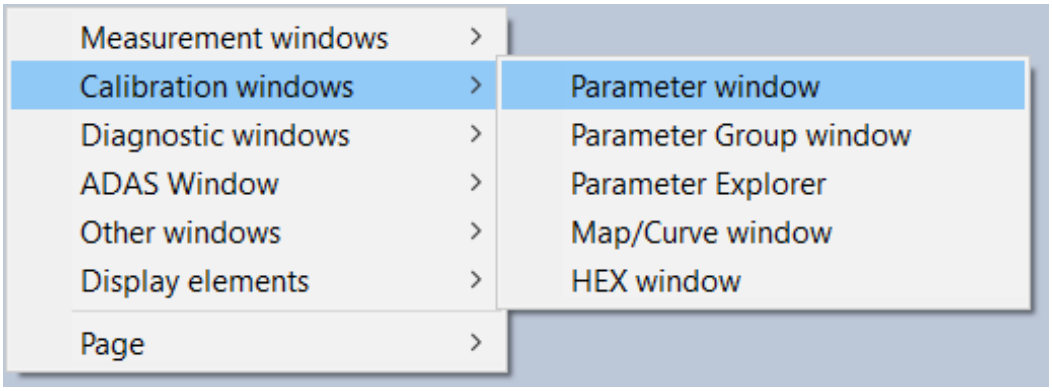
Right click the blank space, you can add measurement windows.



Right click the measurement window, you can choose to add measurement signals.

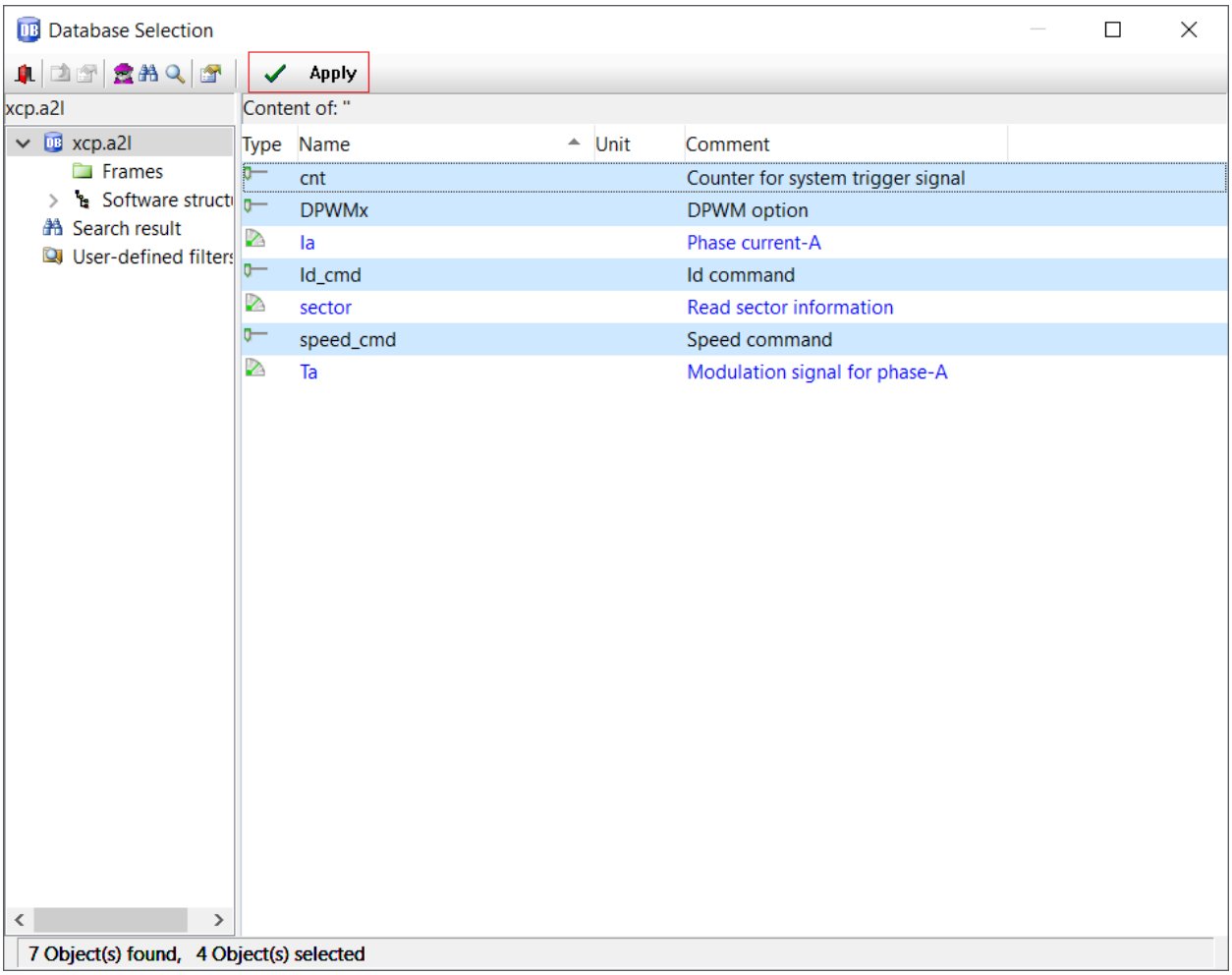


Right click the blank space, you can add calibration windows.

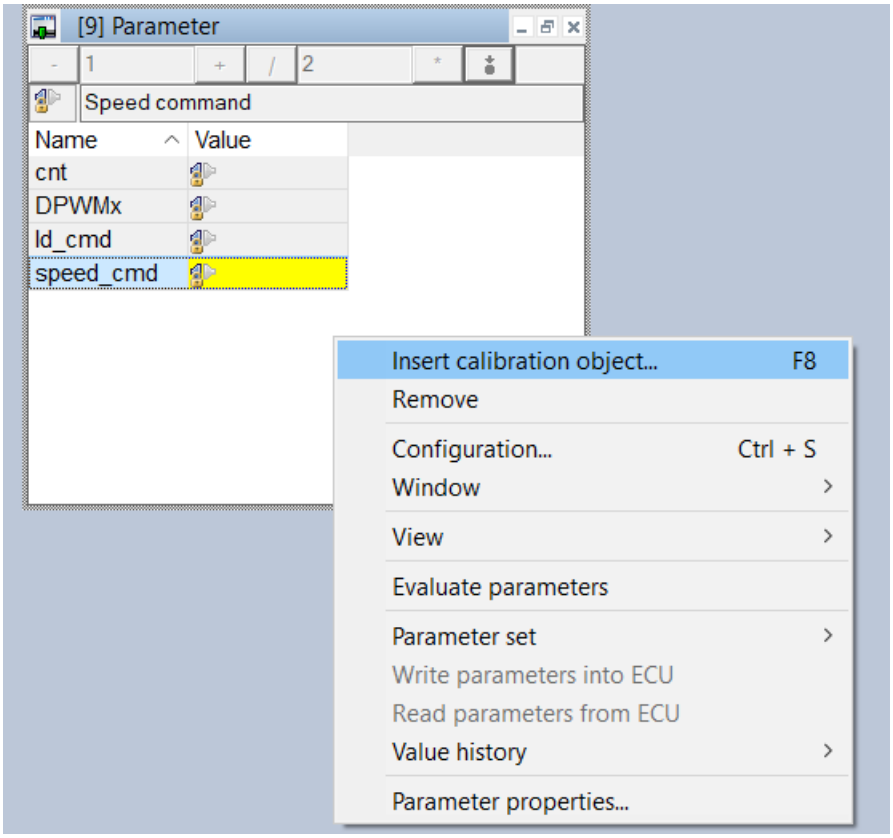


Next, in "**Database Selection**" window you will choose the parameters that you want to add into the calibration window and click "**Apply**". And then, the calibration window will appear.

Here I choose all the available parameters.



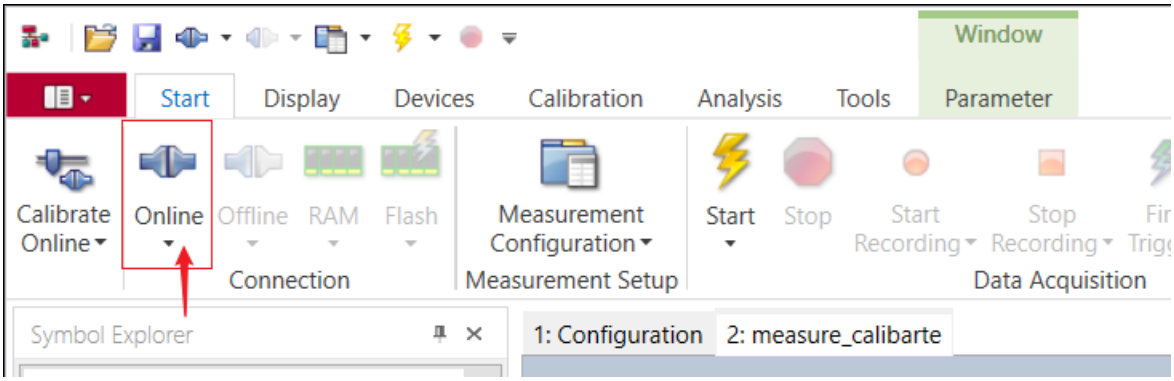
You can still adjust parameters in the calibration window by right clicking.


[9] Parameter
1
2
Speed command
Name
Value
cnt
DPWMx
ld_cmd
speed_cmd
Insert calibration object... F8
Remove
Configuration... Ctrl + S
Window
View
Evaluate parameters
Parameter set
Write parameters into ECU
Read parameters from ECU
Value history
Parameter properties...

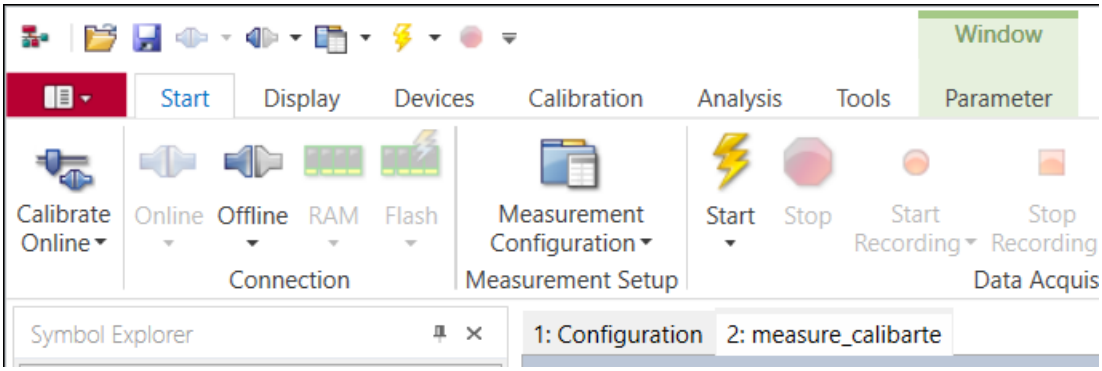
Step 17: Use CANape to Measure and Calibrate

Now, all the necessary steps have been taken. And the communication at both PC side and embedded system side are ready to go.

To establish the connection, the microprocessor should run the program at first. And then, in CANape, click "**Online**" button to connect to the embedded system.

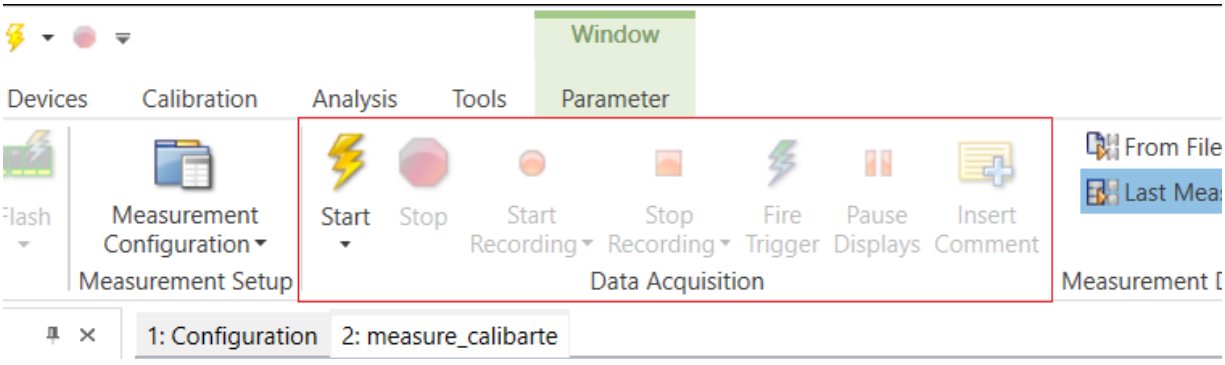


If the connection is successfully established, the "**Online**" button will go grey and the "**Offline**" button, which is used to disconnect, is on.



Now, you can use the calibration window to online modify variables' values.

To obtain the measurement results, you will use buttons in "**Data Acquisition**" area. Click "**Start**" button to start to show the measurement results in the measurement windows. "**Stop**" button is used to stop measurement.



End

This manual describes some necessary steps to establish a connection between CANape software and ZYNQ-7000 Zedboard (embedded system, microprocessor) for measurement and calibration purpose.

This manual doesn’t cover the details in communication coding, and only mentions some operations in CANape. Users can have their explorations on coding part and other CANape operations.