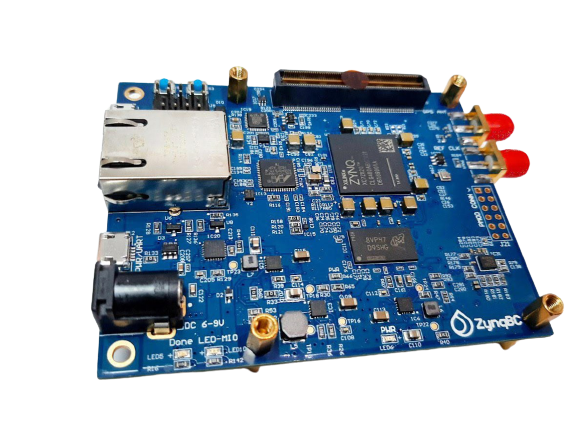
**Using the DAC on ConnectBOB mezzanine board of ZynqBC**



# Overview

This guide will provide a step-by-step walk-through of creating a hardware design using the Vivado IP Integrator for the zynqBC and running a BME280 sensor.

At the end of this tutorial, you will have:

* Created a hardware design for BME sensor.
* Created a .C project in XIlinx Vivado SDK and running the BME functionality using the given functions in C

## **Prerequisites**

#### **Hardware**

* **zynqBC Development Board and JTAG programmer for programming ZYNQ board**

#### **Software**

* **Xilinx Vivado 2020.2 with the vitis.**

#### **Board Support Files**

* **zynqBC Support Files**
  + Follow this link to download the board files from the internet and copy them in the board files folder i.e.
    - *C:\Xilinx\Vivado\2020.2\data\boards\board\_files*

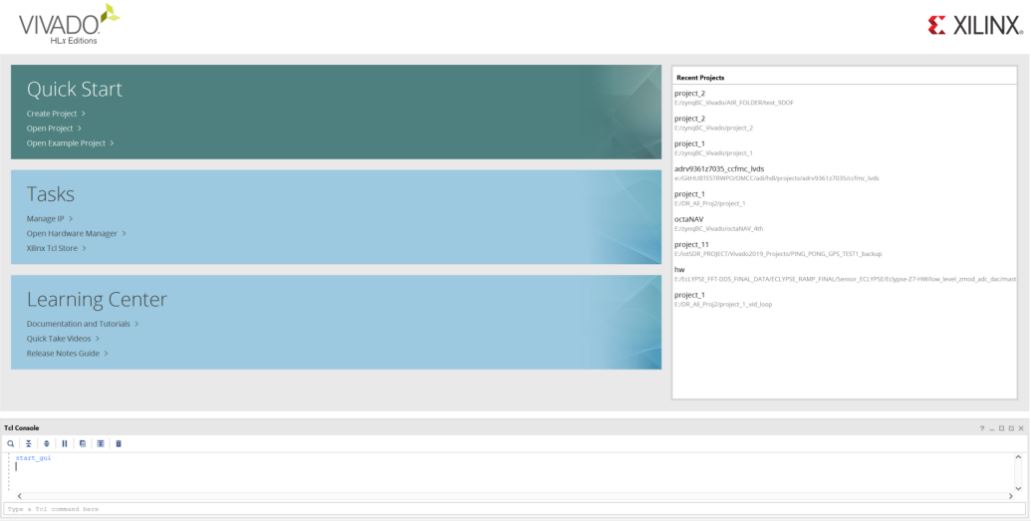
*or*

* + - *E:\Xilinx\Vivado\2020.2\data\boards\board\_files*

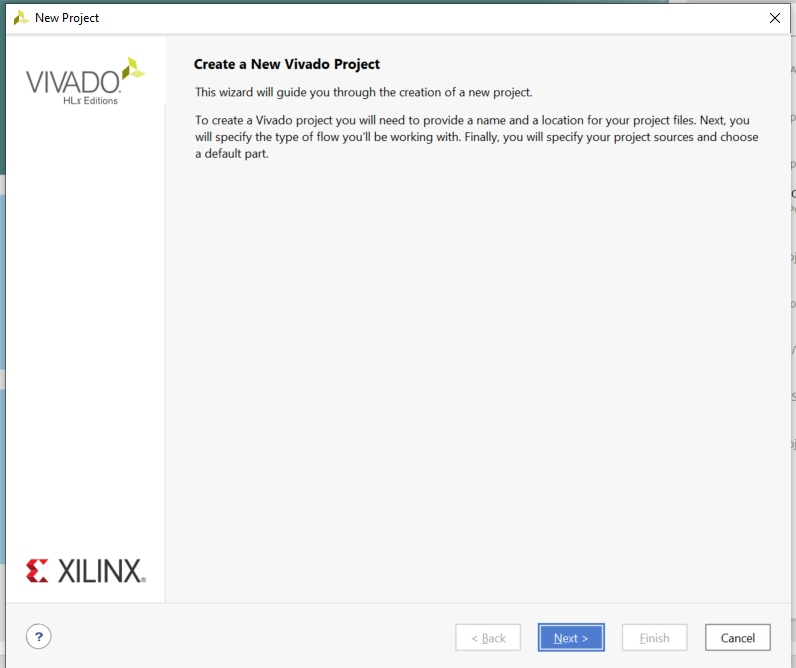
### **1. Creating a New Project**

When you first run Vivado this will be the main start window where you can create a new project or open a recent one.

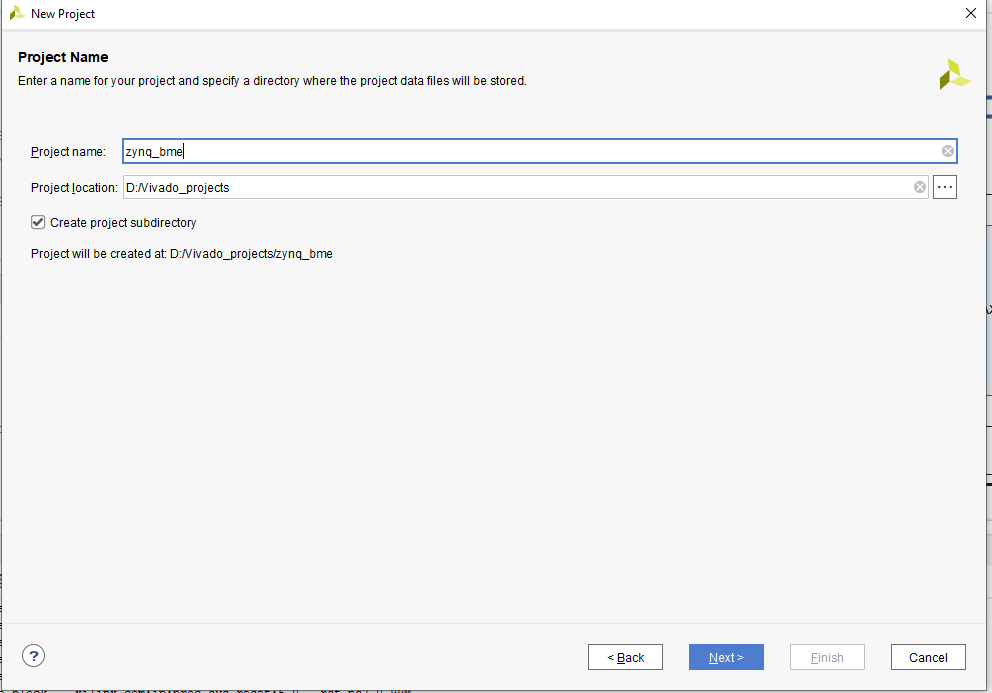
* 1. Click on Create New Project.



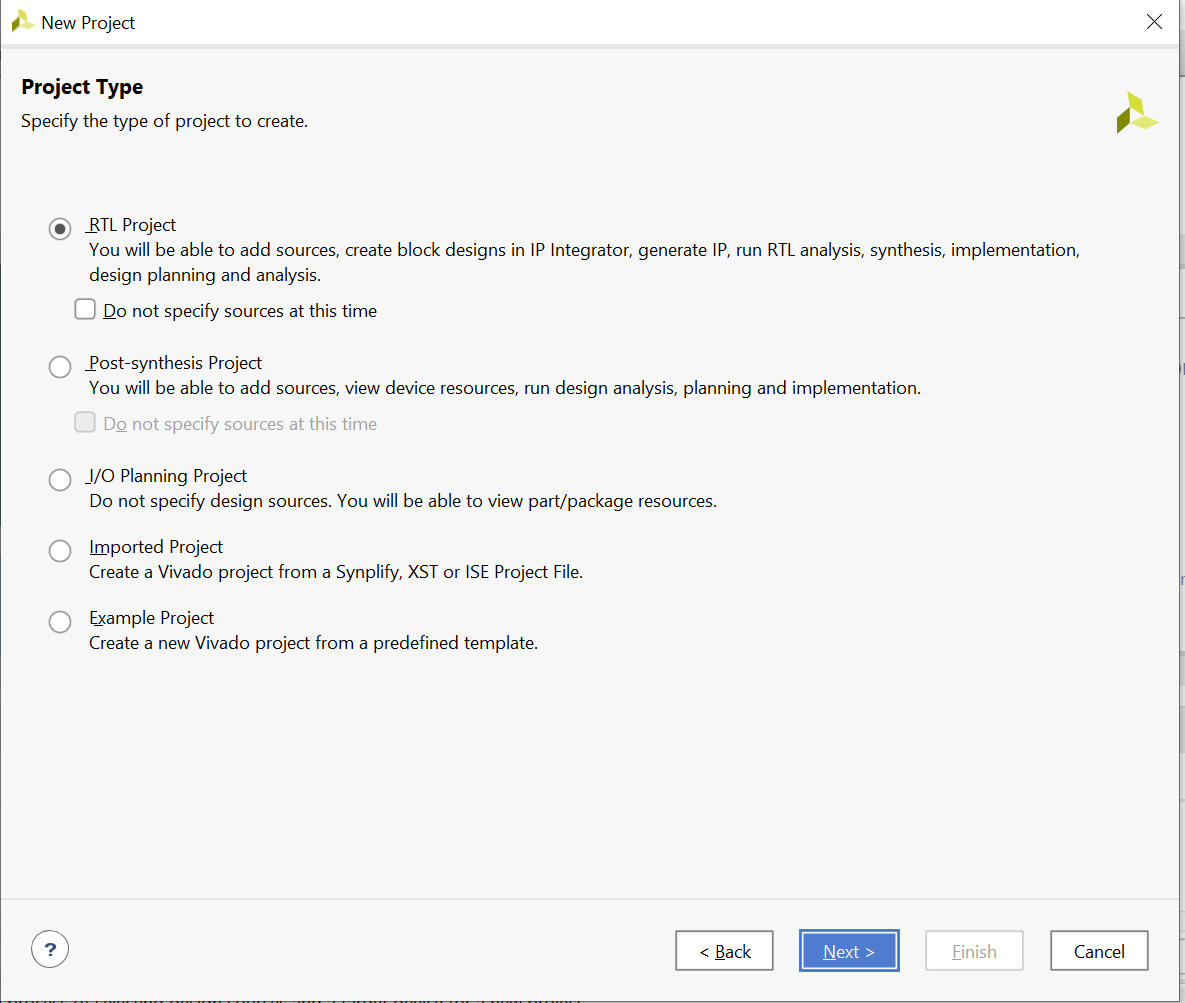
* 1. You will be presented with the project creation wizard. Click Next.



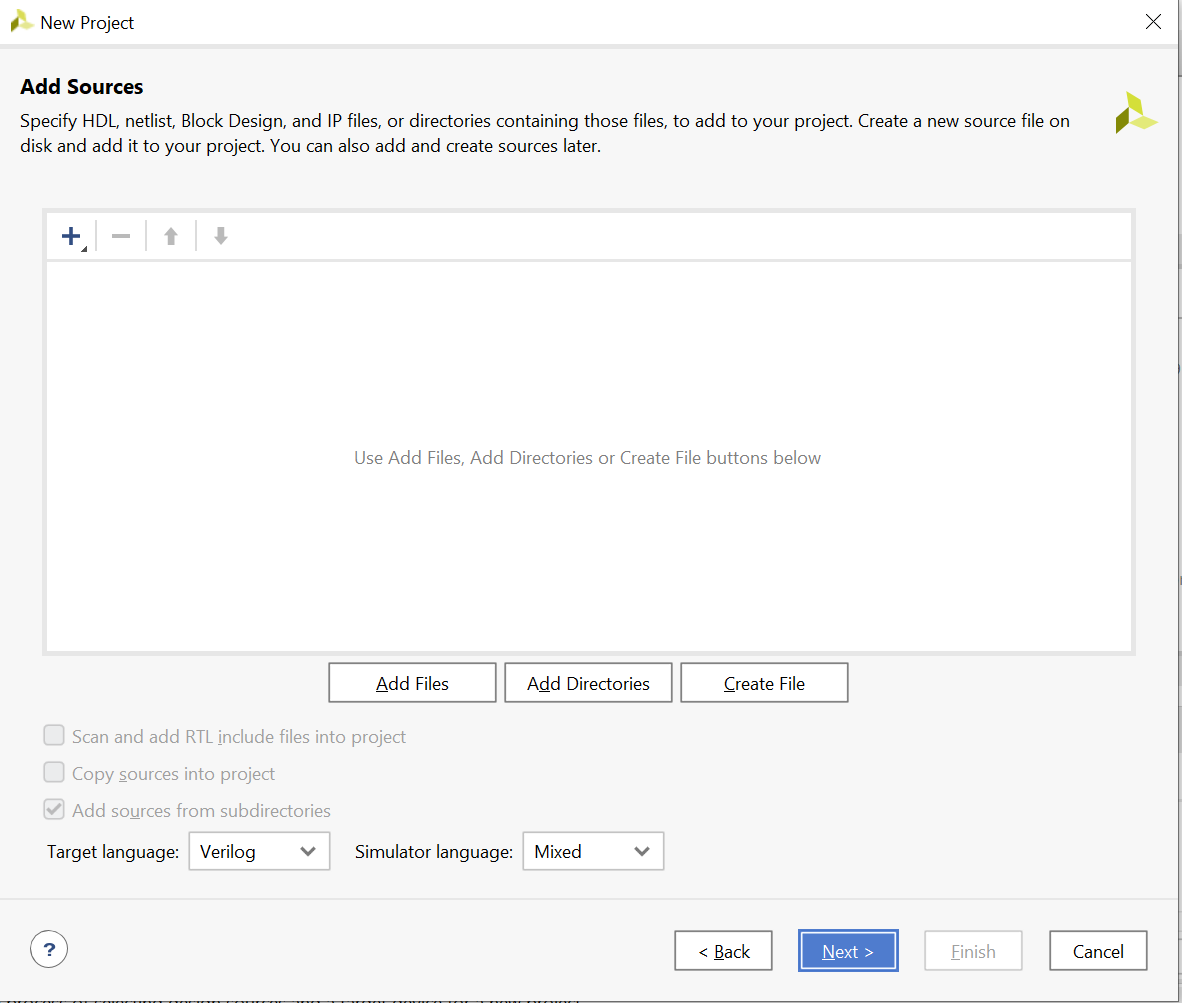
* 1. Enter a project name and location the click **Next**.



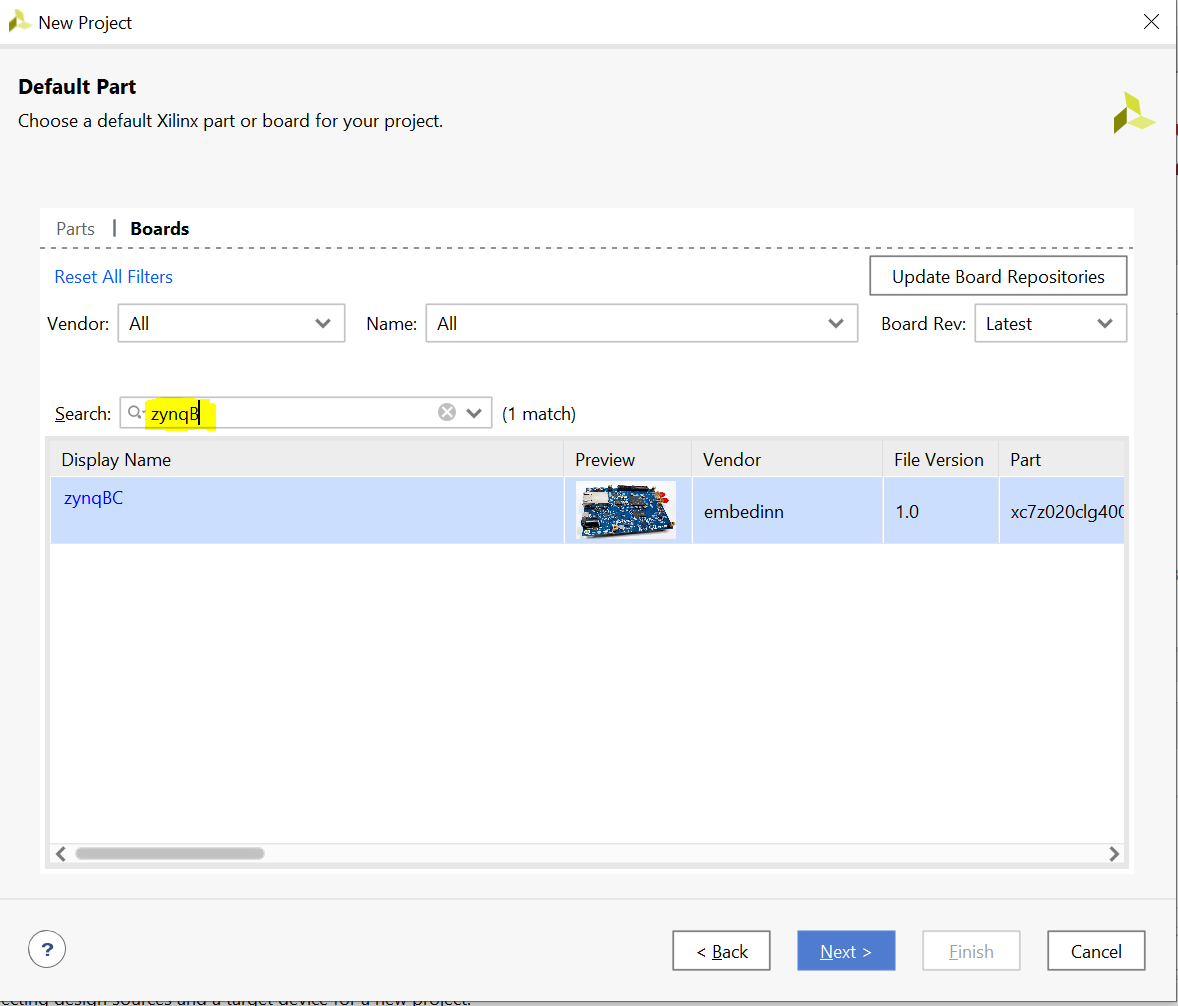
* 1. Select **RTL Project** and click **Next**.



* 1. This demo does not use any existing sources, existing IP or constraints. Click through the next three screens.

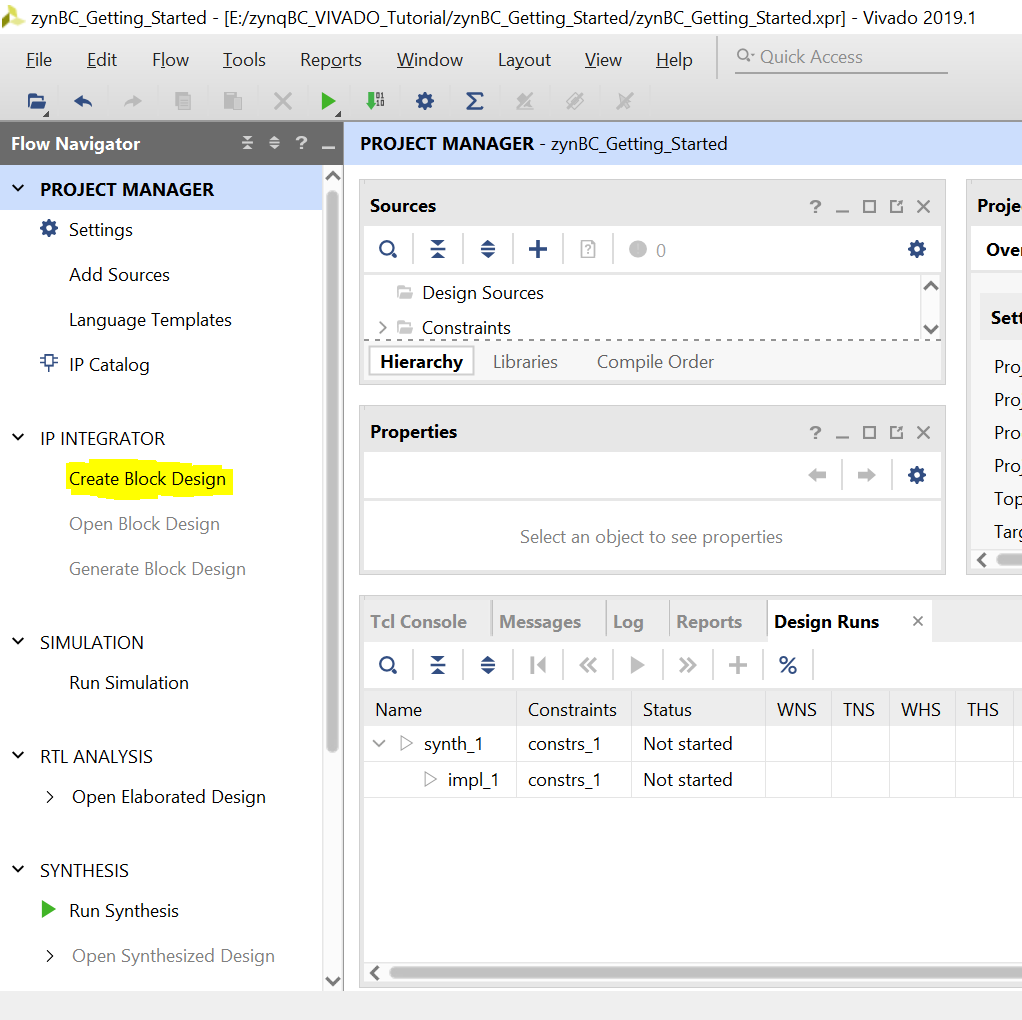


* 1. Select Boards and select the zynqBC board file. Click Next and then Finish. Just make sure before this step that board files are placed already.

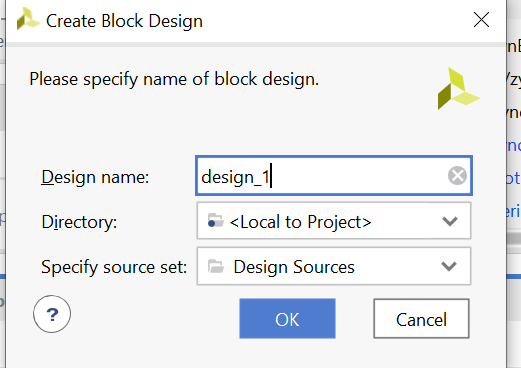


### **Creating a New Block Design**

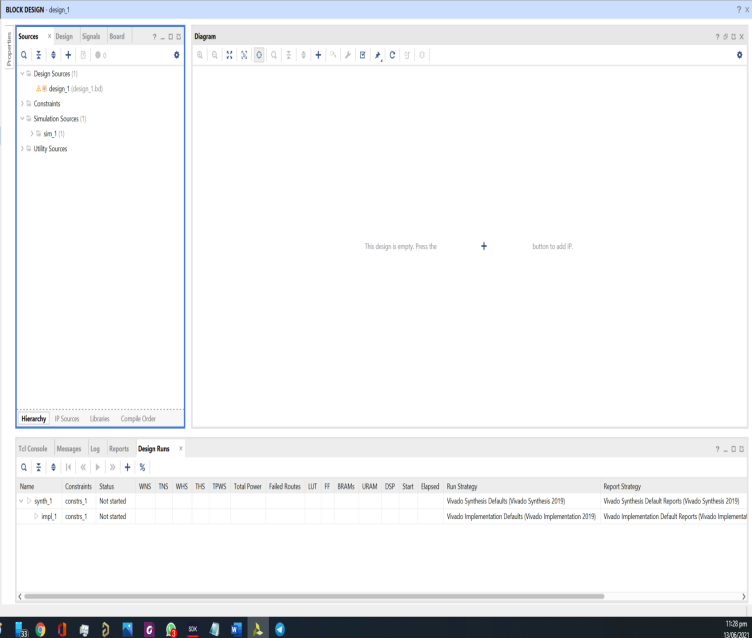
* 1. Once the process has completed, click **Create Block Design** in the flow navigator.



* 1. Click Ok

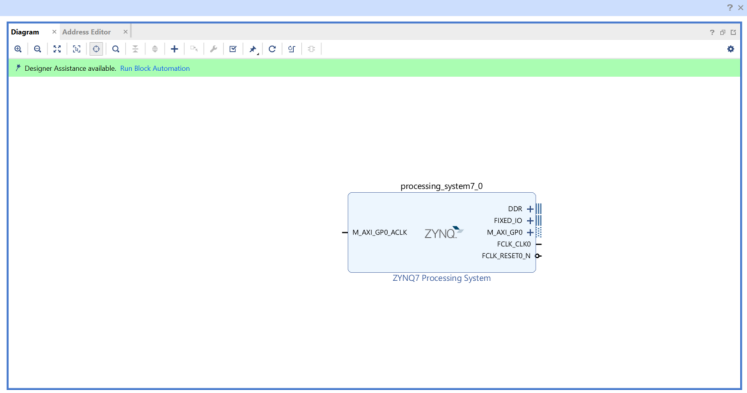


* 1. A blank Block Design will open up.

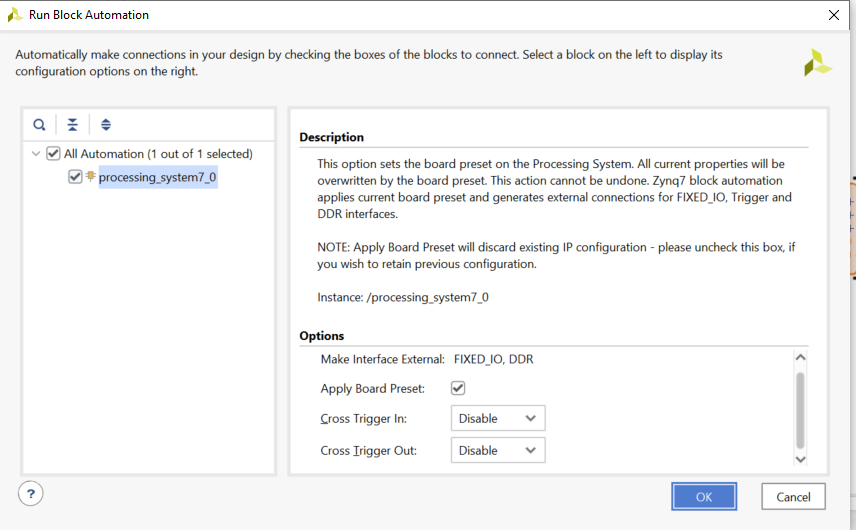


### **Add the Zynq IP & SPI Blocks**

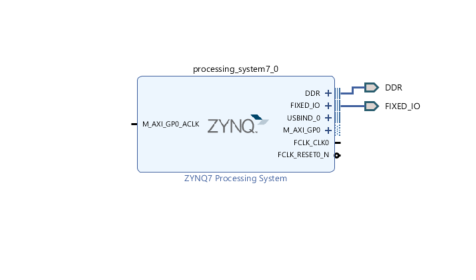
* 1. Click the  **Add IP** button and search for ZYNQ. Double click on **ZYNQ 7 Processing System** to place the bare Zynq block.



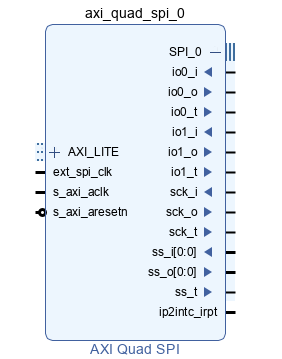
* 1. Click the **Run Block Automation** link



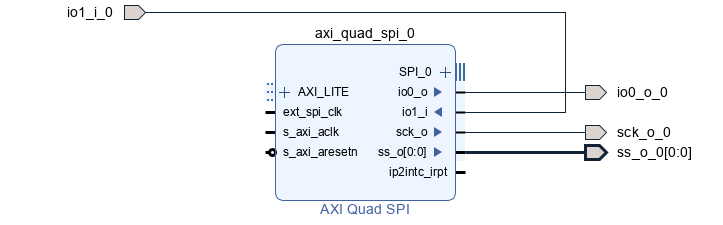
Your Zynq block should now look like the picture below.



* 1. Click on **ADD IP** button and search for axi\_quad\_spi to place it in design. In SPI block expand SPI\_0. The block diagram look like this.

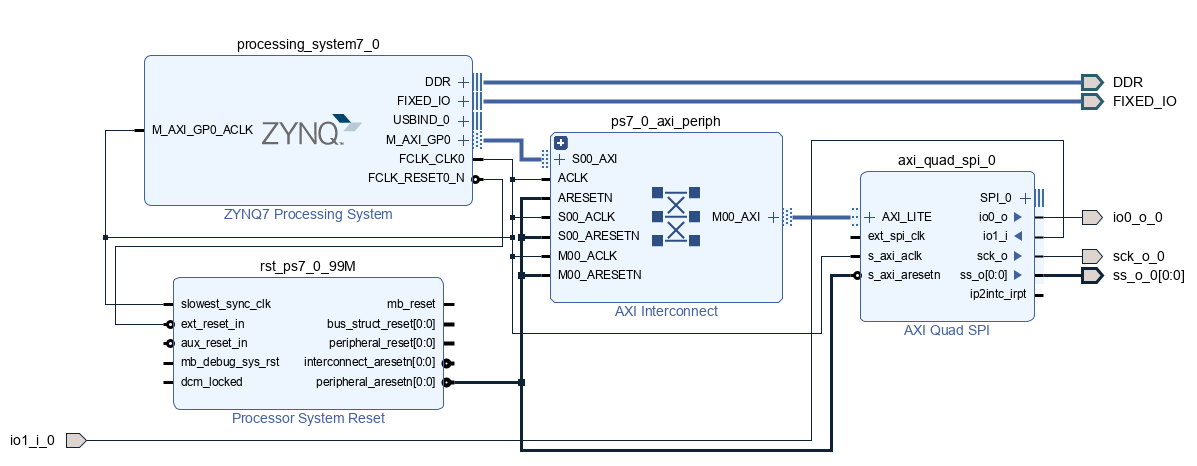


* 1. In this SPI block right click on io0\_o and click on make it external. Similarly do this for io1\_i, sck\_o and ss\_o[0:0]. Unexpand SPI\_0, block diagram look like this.



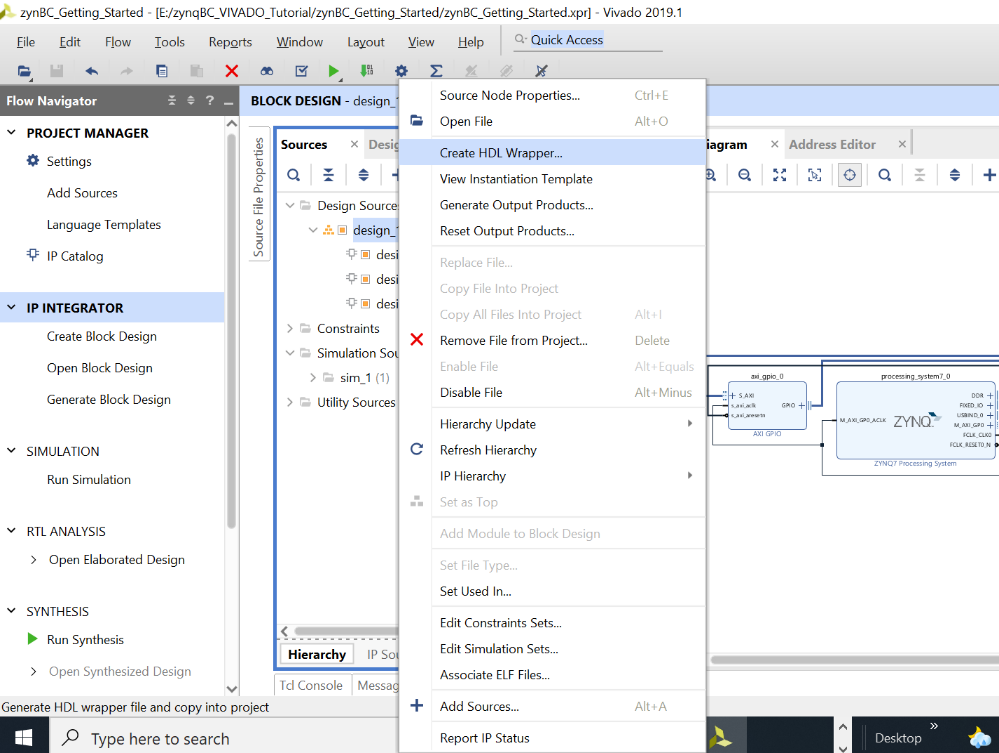
Now click on **Run Connection Automation** again and make sure to check All automation box. **Press OK**

* 1. Your new block design must look like this



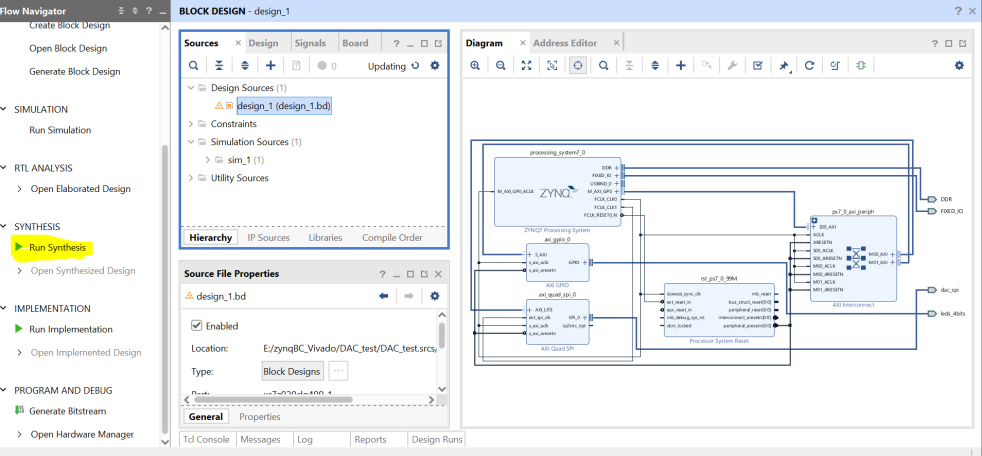
### **Generate HDL Wrapper and Validate Design**

* 1. Select  Validate Design. This will check for design and connection errors.
  2. After the design validation step, we will proceed with creating a HDL System Wrapper. In the block design window, under the **Design Sources** tab, right-click on the block diagram file. We labelled it “design\_1.bd” and select **Create HDL Wrapper**.

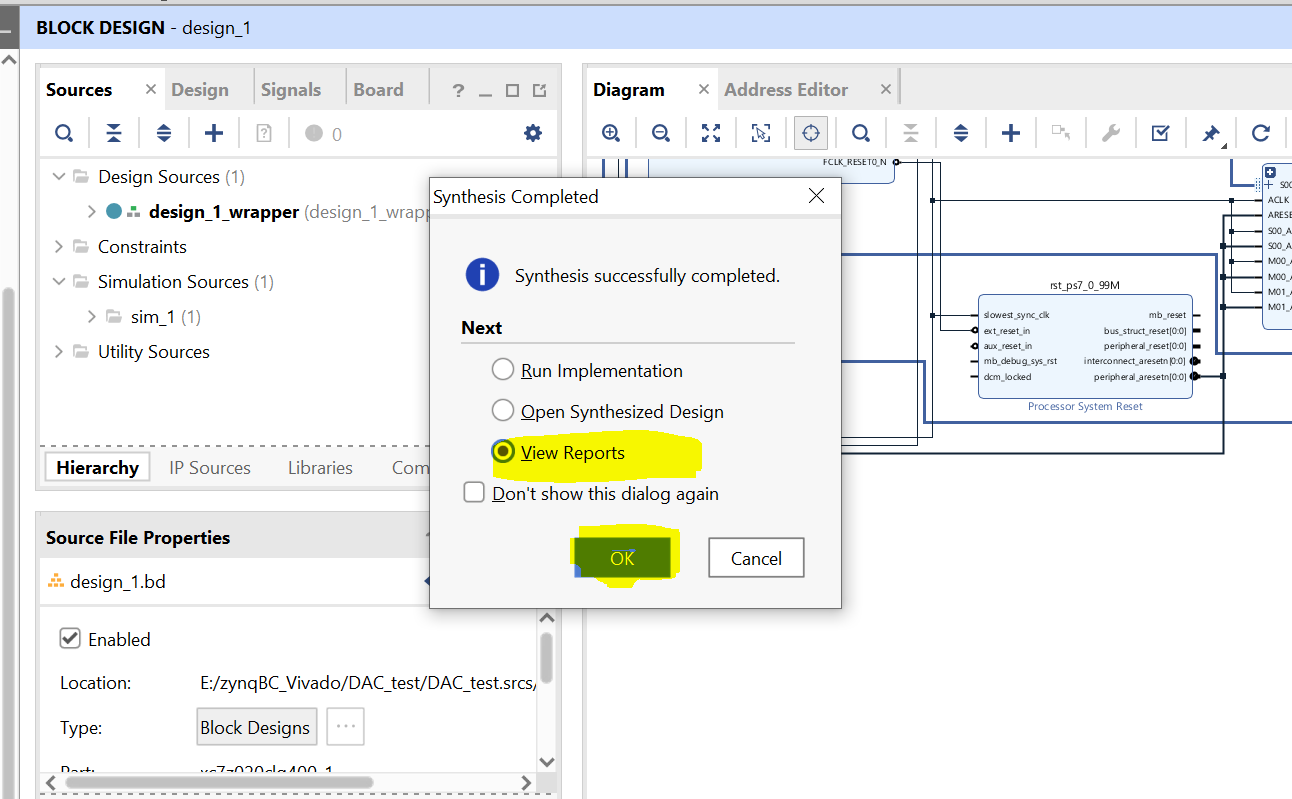


This will create a top module in Verilog and will allow you to generate a bitstream. But before that we need to set the XDC file for the GPIO which is added for LDAC

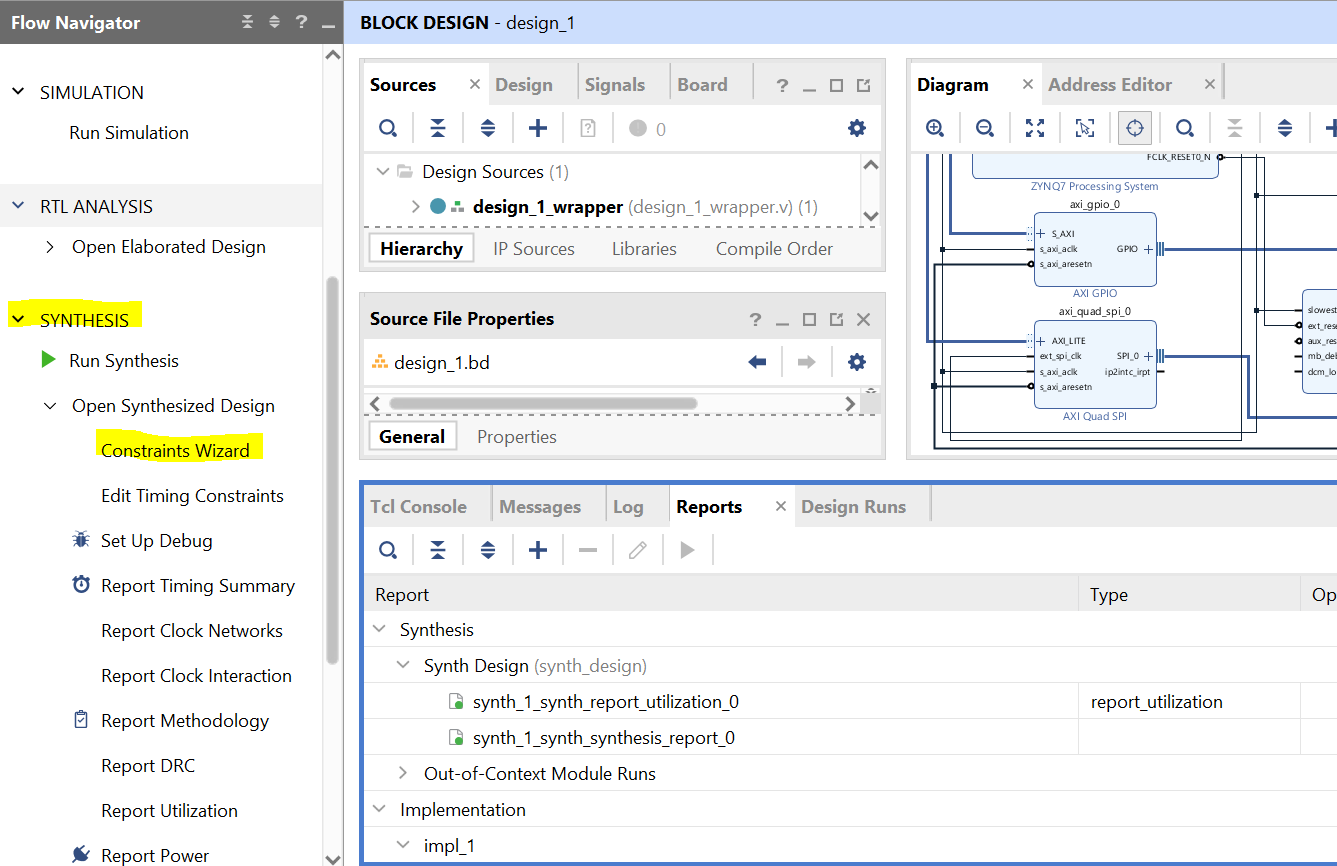
1. **Generate Synthesis**
   1. Click on Run Synthesis



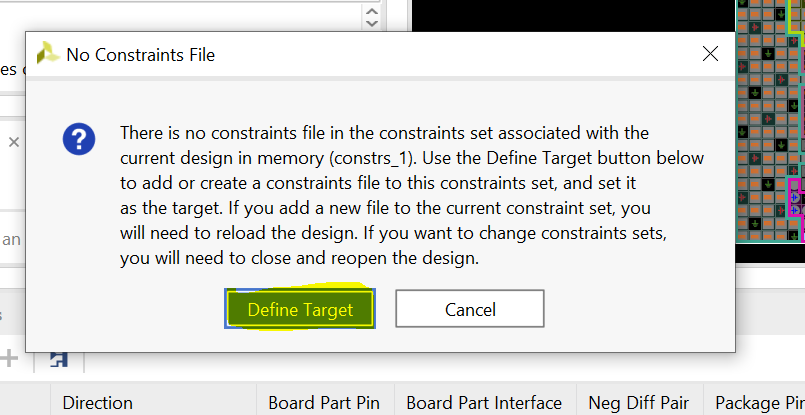
* 1. Once Completed, a message box will appear. Check **View Report** and Press OK

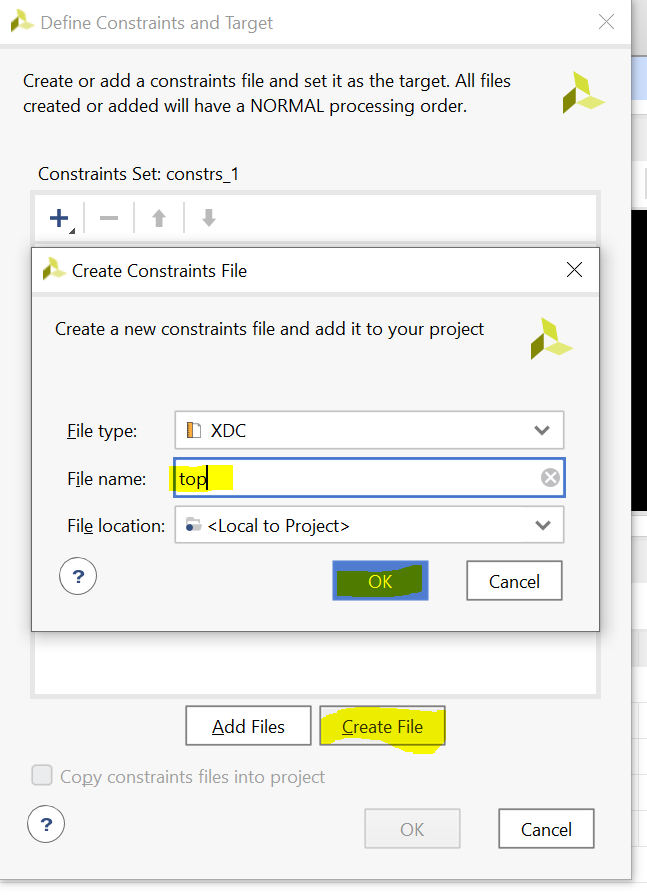


* 1. Click on **Constraints Wizards** under Synthesis

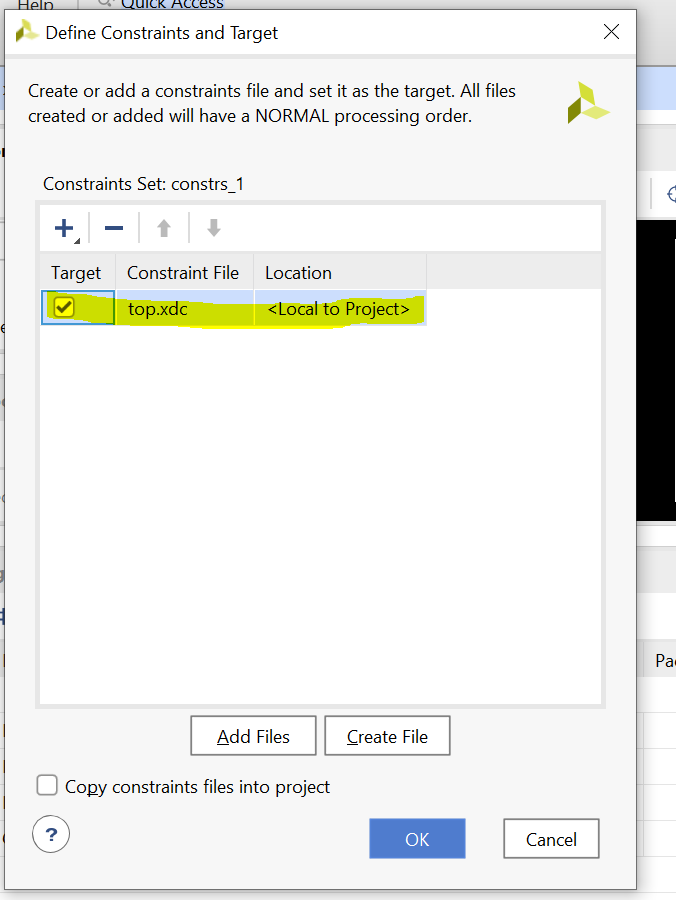


* 1. Pop up will appear with No Constraints file. Click Define Target 🡪 Create File and give top as file name

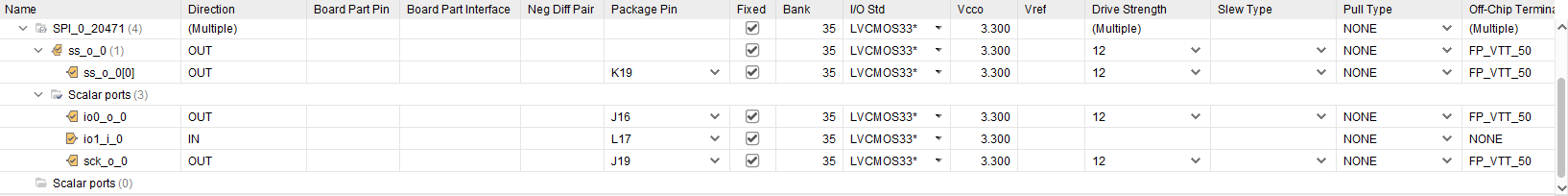




* 1. Check the top.xdc block and Press **OK**

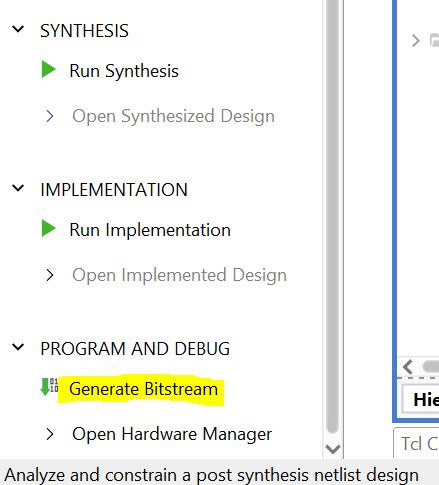


* 1. Under IO Ports, select the pin for SPI. The voltage standard will be LVCMOS33 and pins configuration is shown in figure below.

****

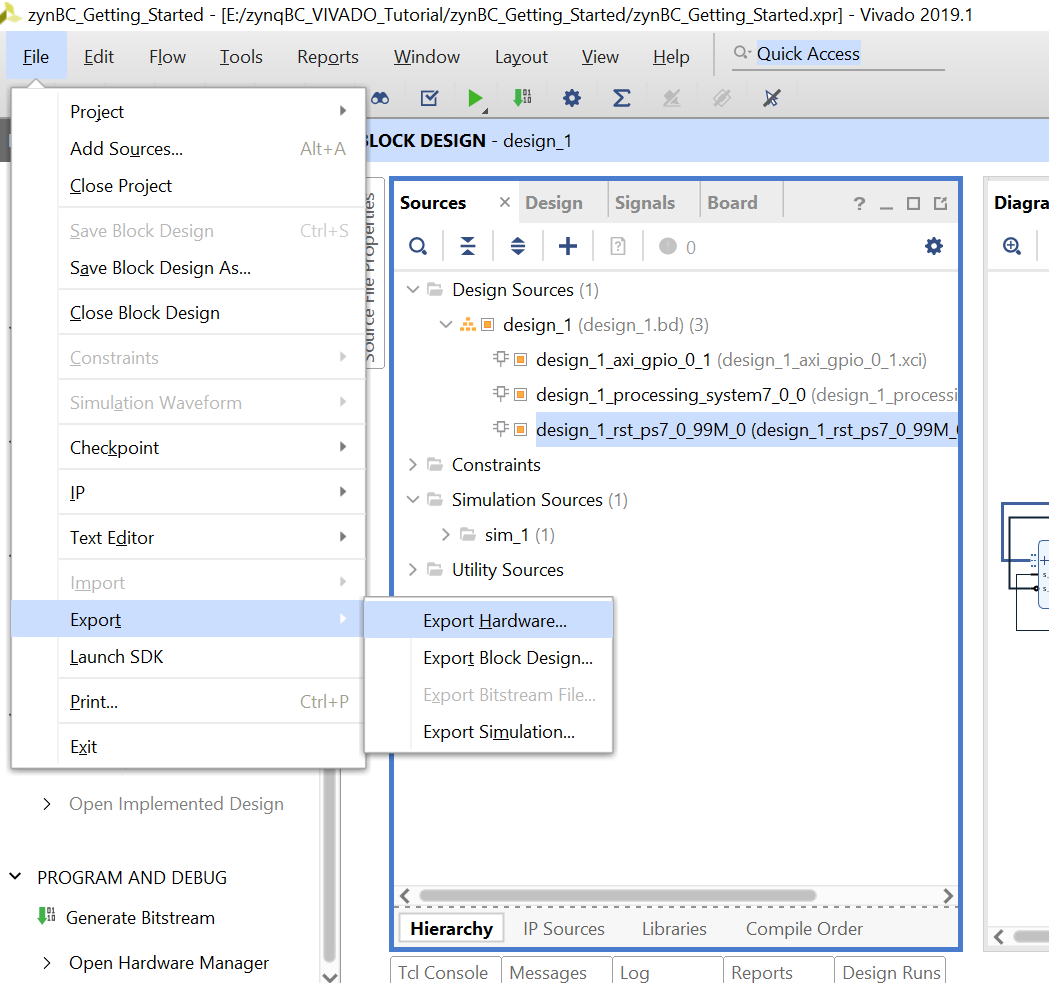
* 1. Press CTRL+S to save the xdc file settings and press **Ok**

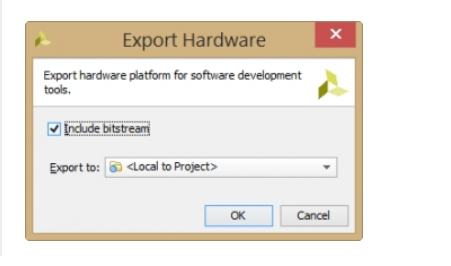
1. **Generate the Bitstream**
   1. Click on **Generate Bitstream** at the bottom of the Flow Navigator. Wait for the process to complete and click OK.



### **Export hardware files for SDK**

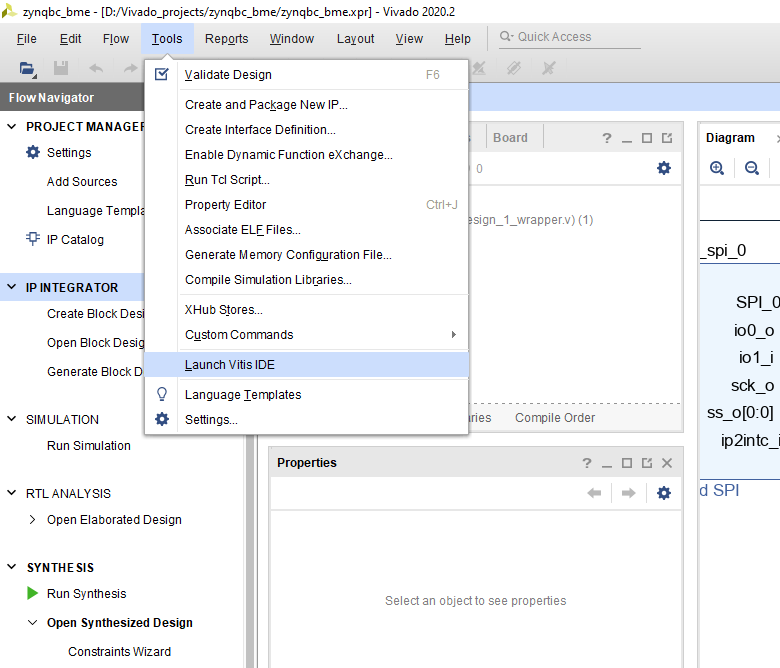
* 1. Go to file→Export→Export Hardware… Make sure to check the box for **Include bitstream** then click **OK**.



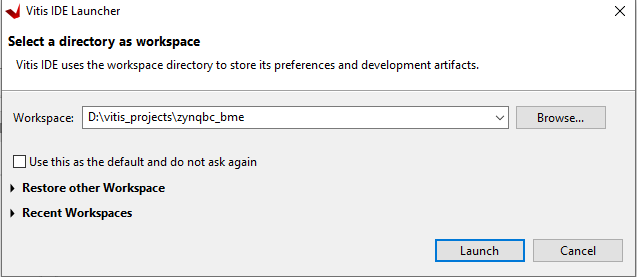


### **Launch Vitis Ide**

* 1. Once bitstream is generated, Go to **Tools→Launch Vitis** IDE and click **OK**.

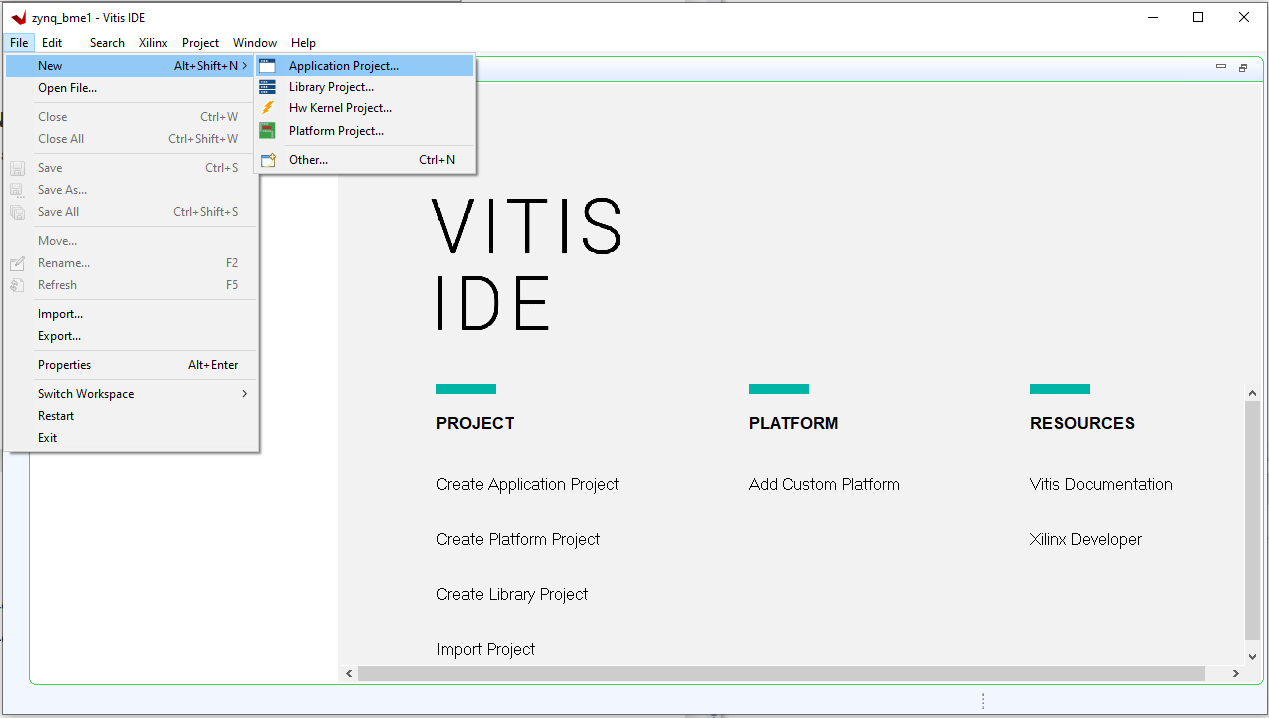


* 1. Set your worksapce. And click on Launch.

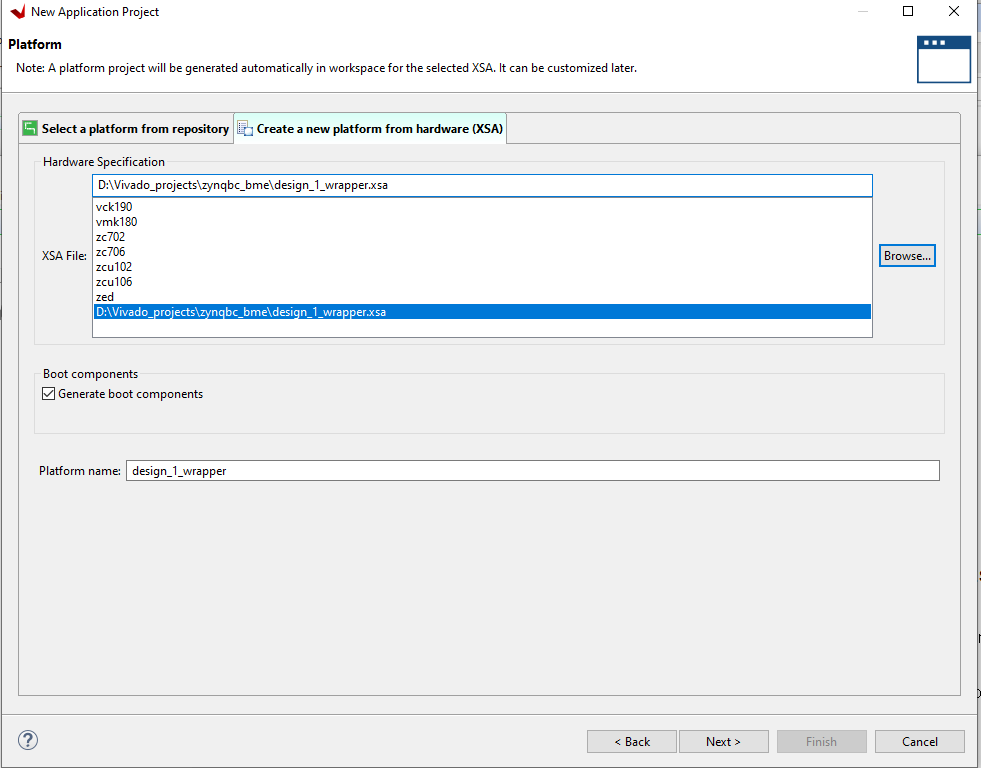


### **Create an Empty Hello World Application on Vitis Ide**

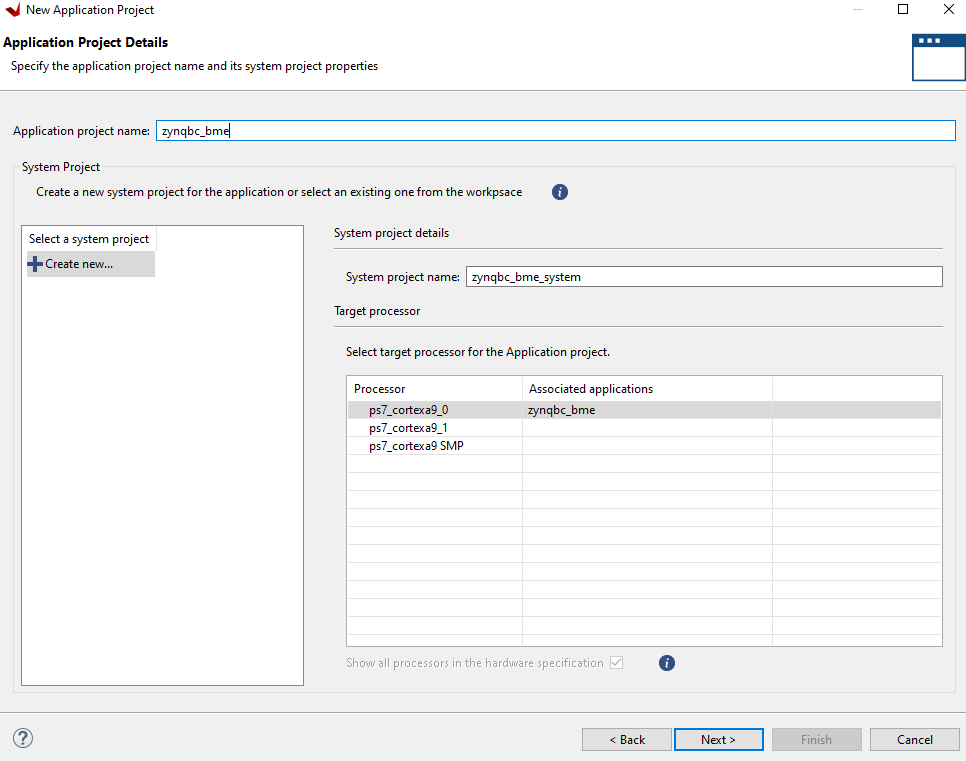
* 1. Click on File and Create an Application Project. Click on next.



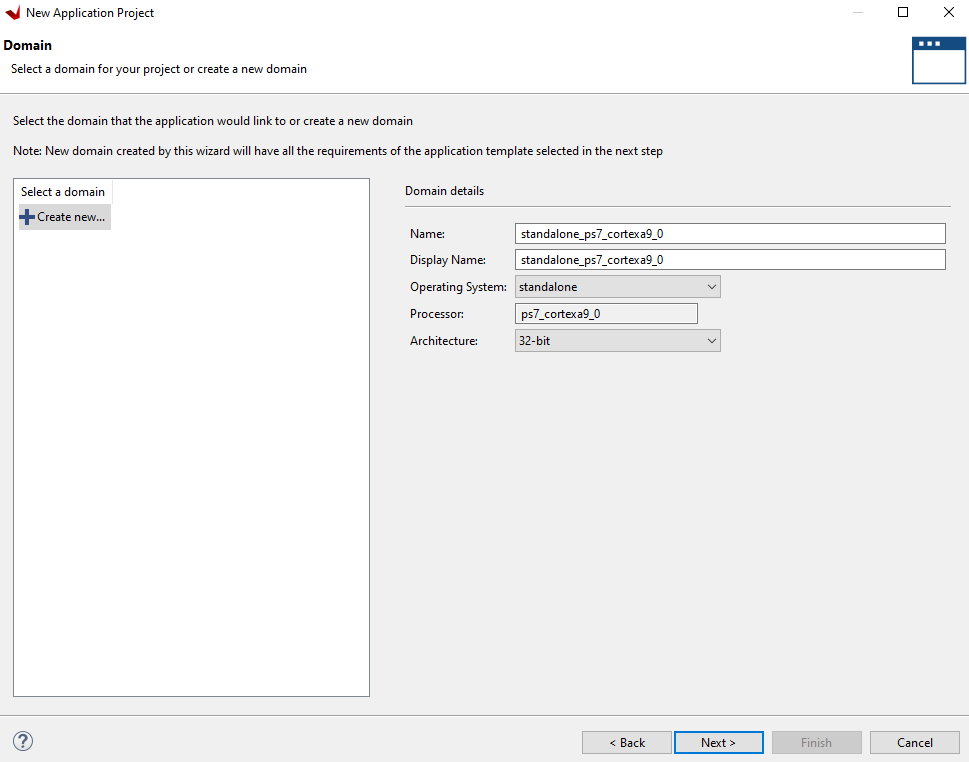
* 1. Go to **create a new platform hardware(XSA)** tab and browse for your design file. Then click Next.



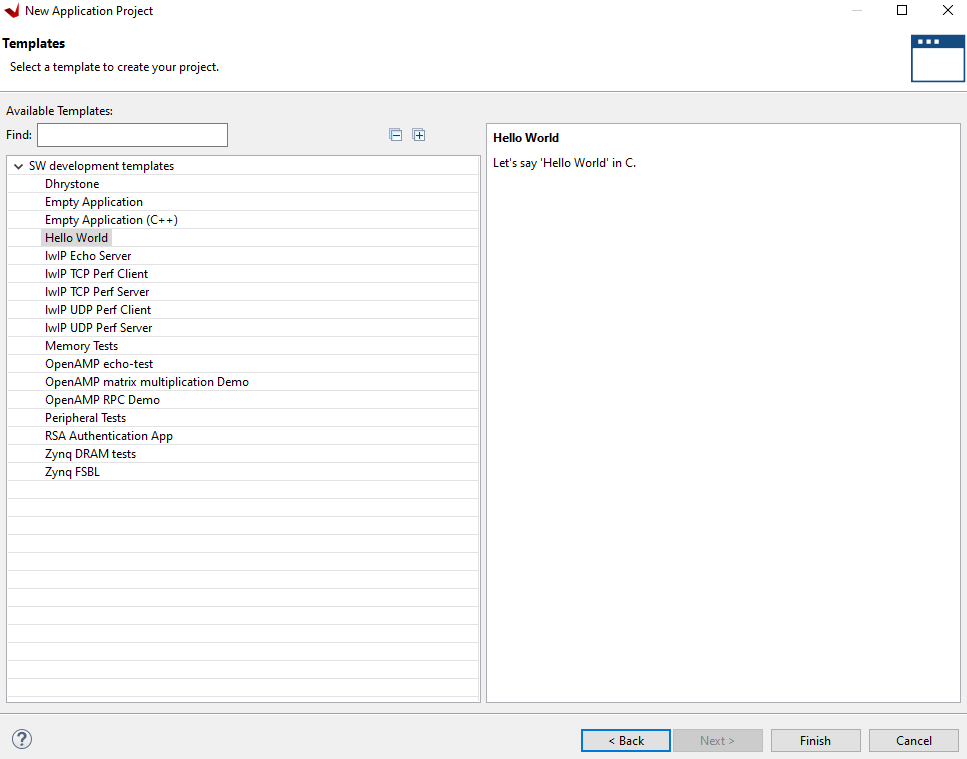
* 1. Give the name of your project and click Next.



* 1. Again click Next.



* 1. Select Hello World and click Finish.



* 1. Copy the following code in helloworld.c

**#include** <stdio.h>

**#include** "platform.h"

**#include** "xil\_printf.h"

**#include** "xstatus.h"

**#include** "xspi.h"

**#include** "Xil\_io.h"

**#include** <math.h>

XSpi\_Config \*ConfigPtr; /\* Pointer to Configuration data \*/

XSpi Spi;

**float** **readbmetemp**();

**float** **readbmepress**();

**float** **readbmehumid**();

**void** **BMEconfig**();

**void** **readBuff**();

**int** **spiInit**(**void**);

**#define** BME280\_REGISTER\_DIG\_TEMP\_START 0x88

**#define** BME280\_REGISTER\_DIG\_PRES\_START 0x8E

**#define** BME280\_REGISTER\_DIG\_HUM\_START 0xE1

//function to read from bme registers using spi

**void** **BME\_Read**(u8 Add, u8 \* buffer, u8 N){

u8 Write[N+1];

Write[0] = Add | 0x80; //R/W = 1

XSpi\_Transfer(&Spi, Write, buffer, N+1);

}

//function to write in bme registers using spi

**void** **BME\_Write**(u8 Add, u8 val){

u8 Write[2], Read[2];

Write[0] = Add & 0x7F; //R/W = 0

Write[1] = val;

XSpi\_Transfer(&Spi, Write, Read, 2);

}

**void** **BME\_Wait**(**void**) {

u8 Read[2];

**do**{

BME\_Read(0xF3, Read, 1 );

} **while** (Read[1] != 0);

}

//declaring global variables

**struct** GLOBALS {

int32\_t t\_fine;

int32\_t read\_dig\_t1, read\_dig\_t2, read\_dig\_t3;

int64\_t read\_dig\_p1, read\_dig\_p2, read\_dig\_p3, read\_dig\_p5, read\_dig\_p4, read\_dig\_p6, read\_dig\_p7, read\_dig\_p8, read\_dig\_p9;

int32\_t read\_dig\_h1, read\_dig\_h2, read\_dig\_h3, read\_dig\_h5, read\_dig\_h4, read\_dig\_h6;

u8 readregisters[10];

} g;

//function to read trimming parameters

**void** **trimming\_readout**(){

u8 readtemp[10];

u8 readpress[20];

u8 readhumid[10];

u8 readhumidA1[2];

BME\_Read(BME280\_REGISTER\_DIG\_TEMP\_START,readtemp,6);

g.read\_dig\_t1 = (**unsigned** **short**)(readtemp[1] + (u16)readtemp[2]\*256);

g.read\_dig\_t2 = (**signed** **short**)(readtemp[3] + (u16)readtemp[4]\*256);

g.read\_dig\_t3 = (**signed** **short**)(readtemp[5] + (u16)readtemp[6]\*256);

BME\_Read(BME280\_REGISTER\_DIG\_PRES\_START,readpress,18);

g.read\_dig\_p1 = (**unsigned** **short**)(readpress[1] + (u16)readpress[2]\*256);

g.read\_dig\_p2 = (**signed** **short**)(readpress[3] + (u16)readpress[4]\*256);

g.read\_dig\_p3 = (**signed** **short**)(readpress[5] + (u16)readpress[6]\*256);

g.read\_dig\_p4 = (**signed** **short**)(readpress[7] + (u16)readpress[8]\*256);

g.read\_dig\_p5 = (**signed** **short**)(readpress[9] + (u16)readpress[10]\*256);

g.read\_dig\_p6 = (**signed** **short**)(readpress[11] + (u16)readpress[12]\*256);

g.read\_dig\_p7 = (**signed** **short**)(readpress[13] + (u16)readpress[14]\*256);

g.read\_dig\_p8 = (**signed** **short**)(readpress[15] + (u16)readpress[16]\*256);

g.read\_dig\_p9 = (**signed** **short**)(readpress[17] + (u16)readpress[18]\*256);

BME\_Read(0xA1,readhumidA1,1);

BME\_Read(BME280\_REGISTER\_DIG\_HUM\_START,readhumid,7);

g.read\_dig\_h1 = (**unsigned** **char**)readhumidA1[1];

g.read\_dig\_h2 = (**signed** **short**)(readhumid[1] + (u16)readhumid[2]\*256);

g.read\_dig\_h3 = (**unsigned** **char**)readhumid[3];

g.read\_dig\_h4 = (**signed** **short**)((readhumid[5] & 0x0F) + (u16)readhumid[4]\*16);

g.read\_dig\_h5 = (**signed** **short**)(((readhumid[5] & 0xF0)>>4) + ((u16)readhumid[6]\*16));

g.read\_dig\_h6 = (**signed** **char**)readhumid[7];

}

**int** **main**()

{

init\_platform();

spiInit();

BME\_Write(0xE0, 0xB6); //Reset the Device

BME\_Wait();

trimming\_readout();

**while**(1){

readBuff();

**float** temp1=readbmetemp(); //Reading temperature

**for**(**int** i=0; i<10;i++){};

**float** press=readbmepress(); //Reading pressure

**for**(**int** i=0; i<10;i++){};

**float** humid = readbmehumid(); //Reading humidity

**for**(**int** i=0; i<10;i++){};

}

cleanup\_platform();

**return** 0;

}

//function to initialise spi

**int** **spiInit**(**void**){

**int** Status;

ConfigPtr = XSpi\_LookupConfig(0U);

**if** (ConfigPtr == NULL) {

**return** XST\_DEVICE\_NOT\_FOUND;

}

Status = XSpi\_CfgInitialize(&Spi, ConfigPtr,ConfigPtr->BaseAddress);

**if** (Status != XST\_SUCCESS) {

**return** XST\_FAILURE;

}

Status = XSpi\_SetOptions(&Spi, XSP\_MASTER\_OPTION );

**if** (Status != XST\_SUCCESS) {

**return** XST\_FAILURE;

}

XSpi\_Start(&Spi);

XSpi\_IntrGlobalDisable(&Spi);

XSpi\_SetSlaveSelect(&Spi, 0x01);

}

**void** **readBuff**(){

BME\_Write(0xF2, 0x02); //Calculate only humidity

BME\_Write(0xF4, 0x45); //Calculate only temperature in Forced Mode

BME\_Read(0xF7,g.readregisters,8);

}

//function to calculate temperature

**float** **readbmetemp**(){

int32\_t tvar0, tvar1=0;

int32\_t tvar2=0;

int32\_t read\_adc = {(g.readregisters[6]>>4) + (int32\_t)g.readregisters[5] \* 16 + (int32\_t)g.readregisters[4] \* 256 \* 16};

tvar0 = (read\_adc>>3)-(g.read\_dig\_t1<<1);

tvar1 = (tvar0\*g.read\_dig\_t2)>>11;

tvar2 = ((((tvar0 >> 1)\*(tvar0 >> 1))>>12)\*(g.read\_dig\_t3))>>14;

g.t\_fine = tvar1 + tvar2 ;

int32\_t T = (g.t\_fine \* 5 + 128) / 256;

**return** (**float**) T/100.0;

}

//function to calculate pressure

**float** **readbmepress**(){

int64\_t pvar1=0;

int64\_t pvar2=0;

int64\_t pvar3=0;

int64\_t pvar4=0;

int32\_t adc\_p = {(g.readregisters[3]>>4) + (int32\_t)g.readregisters[2] \* 16 + (int32\_t)g.readregisters[1] \* 256 \* 16};

pvar1 = ((int64\_t)g.t\_fine) - 128000;

pvar2 = pvar1 \* pvar1 \* (int64\_t)g.read\_dig\_p6;

pvar2 = pvar2 + ((pvar1 \* (int64\_t)g.read\_dig\_p5) \* 131072);

pvar2 = pvar2 + (((int64\_t)g.read\_dig\_p4) \* 34359738368);

pvar1 = ((pvar1 \* pvar1 \* (int64\_t)g.read\_dig\_p3) / 256) + ((pvar1 \* ((int64\_t)g.read\_dig\_p2) \* 4096));

pvar3 = ((int64\_t)1) \* 140737488355328;

pvar1 = (pvar3 + pvar1) \* ((int64\_t)g.read\_dig\_p1) / 8589934592;

**if** (pvar1 == 0) {

**return** 0; // avoid exception caused by division by zero

}

adc\_p <<= 4;

pvar4 = 1048576 - adc\_p;

pvar4 = (((pvar4 \* 2147483648) - pvar2) \* 3125) / pvar1;

pvar1 = (((int64\_t)g.read\_dig\_p9) \* (pvar4 / 8192) \* (pvar4 / 8192)) / 33554432;

pvar2 = (((int64\_t)g.read\_dig\_p8) \* pvar4) / 524288;

pvar4 = ((pvar4 + pvar1 + pvar2) / 256) + (((int64\_t)g.read\_dig\_p7) \* 16);

**float** P = pvar4 / 256.0;

//float P = (uint32\_t)(((pvar4 / 2) \* 100) / 128);

**return** P;

}

//function to calculate humidity

**float** **readbmehumid**(){

int32\_t hvar1, hvar2, hvar3, hvar4, hvar5;

int32\_t adc\_h = {(g.readregisters[8]) + (int32\_t)g.readregisters[7] \* 16 \*16};

hvar1 = g.t\_fine - ((int32\_t)76800);

hvar2 = (int32\_t)(adc\_h \* 16384);

hvar3 = (int32\_t)(((int32\_t)g.read\_dig\_h4) \* 1048576);

hvar4 = ((int32\_t)g.read\_dig\_h5) \* hvar1;

hvar5 = (((hvar2 - hvar3) - hvar4) + (int32\_t)16384) / 32768;

hvar2 = (hvar1 \* ((int32\_t)g.read\_dig\_h6)) / 1024;

hvar3 = (hvar1 \* ((int32\_t)g.read\_dig\_h3)) / 2048;

hvar4 = ((hvar2 \* (hvar3 + (int32\_t)32768)) / 1024) + (int32\_t)2097152;

hvar2 = ((hvar4 \* ((int32\_t)g.read\_dig\_h2)) + 8192) / 16384;

hvar3 = hvar5 \* hvar2;

hvar4 = ((hvar3 / 32768) \* (hvar3 / 32768)) / 128;

hvar5 = hvar3 - ((hvar4 \* ((int32\_t)g.read\_dig\_h1)) / 16);

hvar5 = (hvar5 < 0 ? 0 : hvar5);

hvar5 = (hvar5 > 419430400 ? 419430400 : hvar5);

uint32\_t H = (uint32\_t)(hvar5 / 4096);

**return** (**float**)H / 1024.0;

}