AGH University of Science and Technology
Cracow
Department of Electronics

Custom processor design in FPGA laboratory

Tutorial 3

Implementation of the system with a sequential accelerator

Author: Paweł Russek

ver. 2018.03.20

1. Introduction

1.1. Objectives

This tutorial presents the step by step procedure of creating hardware accelerated design for Xilinx's Zynq All-programmable System-on-chip (PsoC). The **programmable logic** of Zynq (i.e. FPGA) will be used to implement the custom cordic coprocessor. The coprocessor will be used to offload the Zynq's **processor subsystem (**i.e. ARM**)** to off-load sin/cos operations.

We will use the custom cordic processor that was prepared in the Tutorial 1 of the laboratory. The **Zynq project** will by deployed in Vivado Design Suit tool. The project consist of the **hardware** and **software** parts The result of the hardware development is the programming bitstream for the ARM's programmable logic, and the effect of the software project is a binary executable for the ARM subsystem.

The tutorial shows both the **hardware** and **software** implementation steps for the Zynq chip in Vivado. First, the hardware will be created in Vivado, and its configuration data exported for the software development kit (SDK). Then, the software is developed, debugged, and run on Zedboard using the Vivado SDK tool.

1.2. Prerequisites

Before starting this laboratory student should complete Tutorial 1 of the laboratory. Understanding and basic knowledge of Verilog and C programming language will be necessary to complete the tutorial. Prior experience with the Xilinx's Vivado tool will be an advantage.

1.3. Software and hardware

- Vivado v2016.2 (64-bit).
- ZedBoard Development Platform.
- The t3_cordic_src.zip file that is an addition to this document.

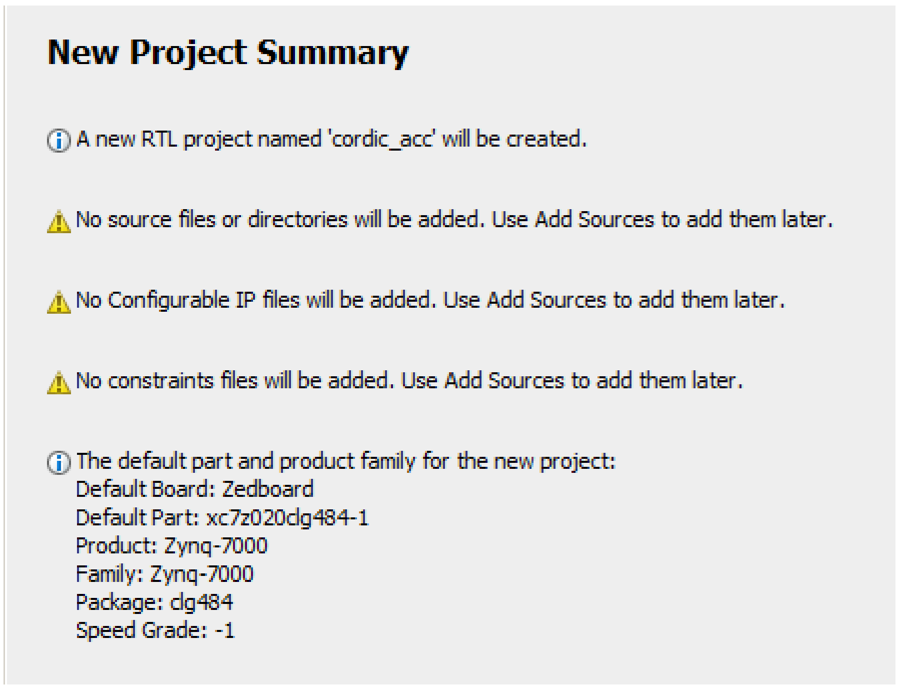
2. Creating Cordic IP

Before the RTL code of the cordic processor can be used in our Vivado project, it has to be wrapped as the Vivado library element that conforms the Vivado IP format. We will run Vivado and create the cordic accelerator as the peripheral IP for AXI4 bus.

AXI4 is a SoC bus standard that is used by ARM to communicate with its peripherals. We will use the template code for the AXI4 IP that will be customized by the cordic RTL code to implement desired functionality.

a) **Launch Vivado on your computer.**

b) **Create new project "cordic_acc" for Zedboard.**
Check your project settings in the figure below.



New Project Summary

A new RTL project named 'cordic_acc' will be created.

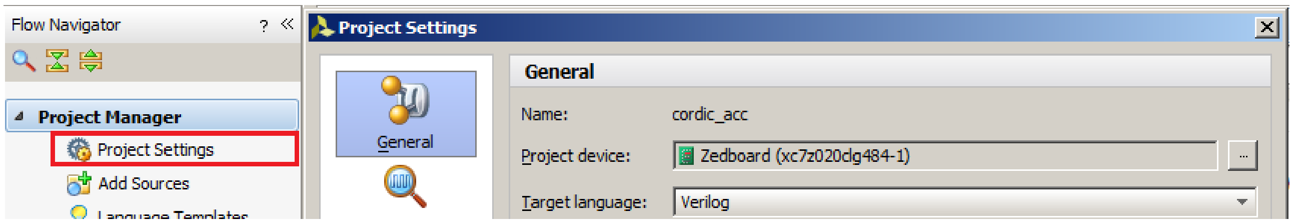
No source files or directories will be added. Use Add Sources to add them later.

No Configurable IP files will be added. Use Add Sources to add them later.

No constraints files will be added. Use Add Sources to add them later.

The default part and product family for the new project:
Default Board: Zedboard
Default Part: xc7z020clg484-1
Product: Zynq-7000
Family: Zynq-7000
Package: clg484
Speed Grade: -1

c) **Select "Project settings" in the "Flow navigator" window under "Project manager". Change "Target language" to Verilog.**



General

Name: cordic_acc

Project device: Zedboard (xc7z020clg484-1)

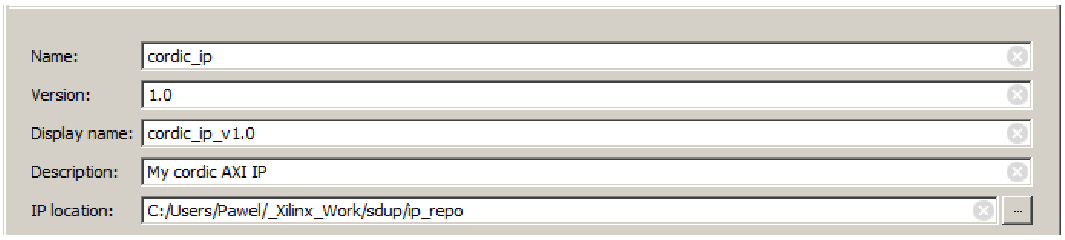
Target language: Verilog

d) **Launch "Tools → Create and Package IP ..." form the main window menu.**
Click Next button

e) **Select "Create a new AXI4 peripheral" in the *"Create Peripheral, Package IP or Package a Block Diagram"* window.**
Click Next button

f) In *“Peripheral Details”* window name the IP as “cordic_ip” and note the IP directory location.

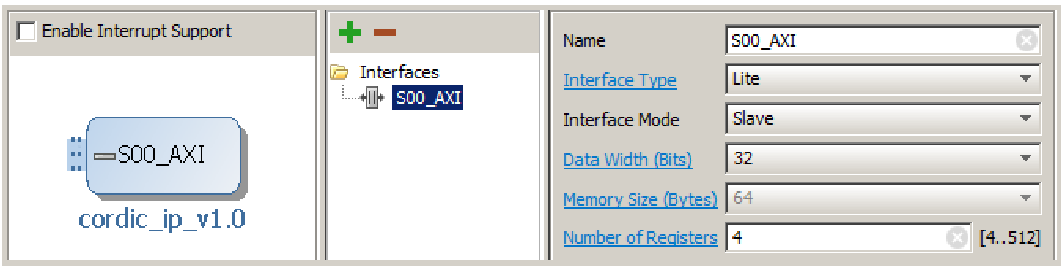
You can select the target IP directory according your preferences but we advise to leave the default location: “~/iprepo”



Name:	cordic_ip
Version:	1.0
Display name:	cordic_ip_v1.0
Description:	My cordic AXI IP
IP location:	C:/Users/Pawel/_Xilinx_Work/sdup/ip_repo

Click Next button

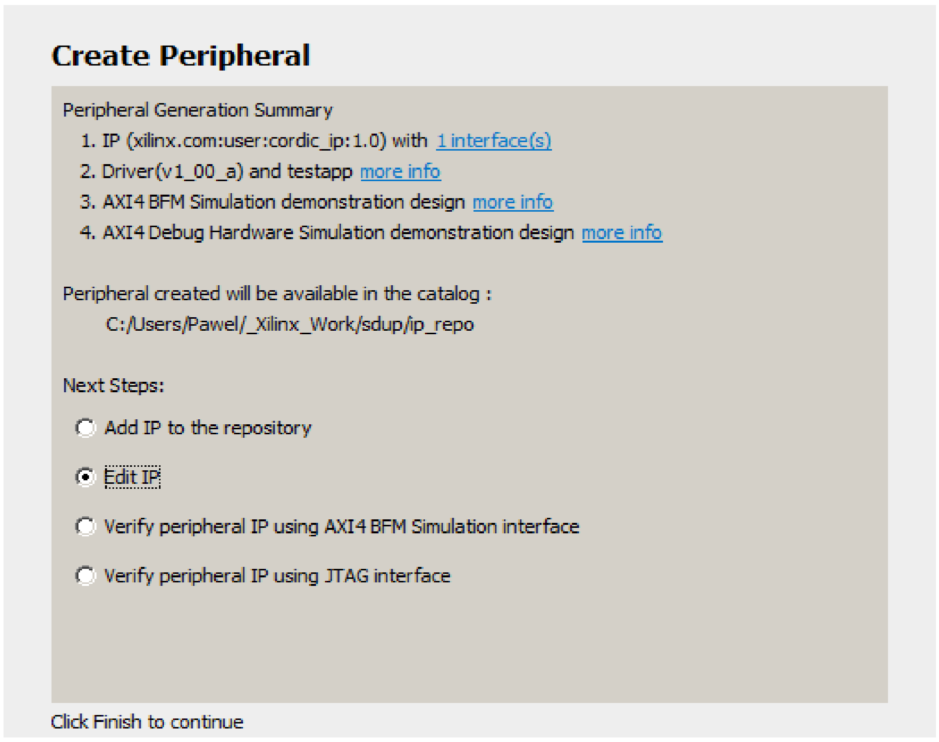
g) Leave, as proposed, one Slave AXI Lite interface for your IP.



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

Click Next button

h) Verify your settings in the figure below


Create Peripheral

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

Peripheral created will be available in the catalog :
C:/Users/Pawel/_Xilinx_Work/sdup/ip_repo

Next Steps:
- ○ Add IP to the repository
- ● Edit IP
- ○ Verify peripheral IP using AXI4 BFM Simulation interface
- ○ Verify peripheral IP using JTAG interface

Click Finish to continue

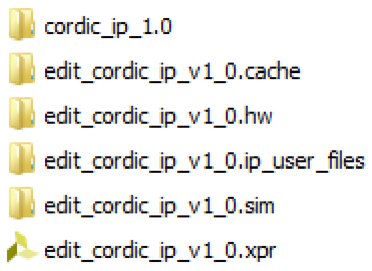
Read information in boxes that appears on click of "more info" links.

Select "Edit IP".

Click Finish button

i) The Vivado project "edit_cordic_ip" should open automatically in the separate Window.

Also, you should be able to find the "cordic_ip" design files in the selected "ip_repo" directory location.


cordic_ip_1.0
edit_cordic_ip_v1_0.cache
edit_cordic_ip_v1_0.hw
edit_cordic_ip_v1_0.ip_user_files
edit_cordic_ip_v1_0.sim
edit_cordic_ip_v1_0.xpr

3. Customizing Cordic IP

You have generated IP module for the AXI-Light peripheral module in Section 2. AXI Light is a subclass if the AXI4 bus standard. The AXI Light peripheral can be connected to Zynq's ARM subsystem, and we will do that later.

Created Verilog template code contains only logic for the AXI-Light interface that allow CPU to read/write data to the four memory mapped registers slv_reg(0-3) only . The Vivado's user work is to customize IP and add the peripheral specific functionality that is also called **user algorithm**. Figure 1 present the proposed architecture of the Vivado's peripheral accelerator IP.

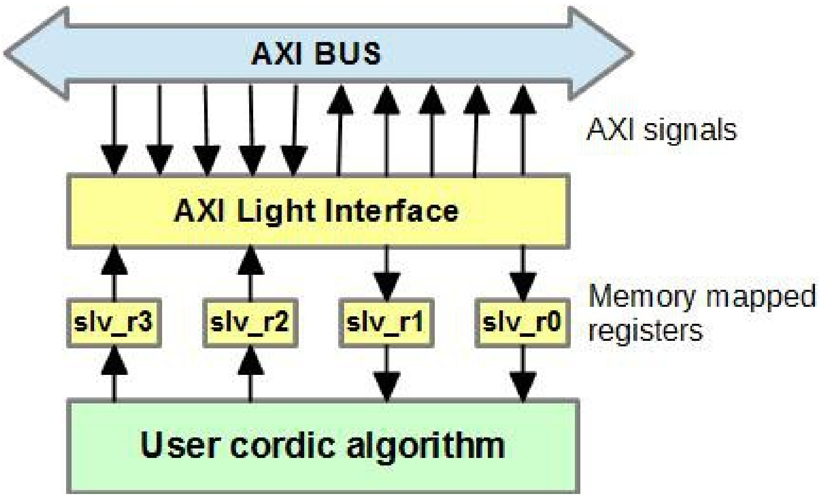


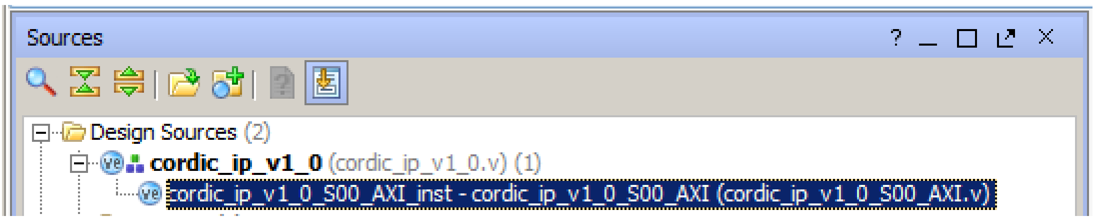
Figure 1

Now, we will include the the cordic_rtl module into the generated template Verilog code.

For the Verilog code of the cordic_rtl module, please, find the cordic_rtl.v file, included in t3_cordic_src.zip file.

3.1. Creating cordic_ip Verilog code

a) **Open cordic_ip_v1_0_S00_AXI module definition in cordic_ip_v1_0_S00_AXI.v file that is located in project sources.**


Sources
Design Sources (2)
cordic_ip_v1_0 (cordic_ip_v1_0.v) (1)
cordic_ip_v1_0_S00_AXI_inst - cordic_ip_v1_0_S00_AXI (cordic_ip_v1_0_S00_AXI.v)

b) **Add the cordic_rtl module definition to the cordic_ip_v1_0_S00_AXI.v file**

You can simply copy the Verilog code of cordic_rtl form the coridic_rtl.v and past it at the end of cordic_ip_v1_0_S00_AXI.v file

```
// Add user logic here

// User logic ends

endmodule

///////////////////////////////////////////////////////////////////////////////////

module cordic_rtl( clock, reset, start, angle_in, ready_out, sin_out, cos_out);
parameter integer W = 12; //Fixed-point representation precision fixpoint(2:10)
parameter FXP_MUL = 1024;
```

c) **Instantiate cordic_ip in the cordic_ip_v1_0_S00_AXI module.**

Copy and paste the code below into the user code section of cordic_ip_v1_0_S00_AXI.v

```
// Add user logic here

  //We create acive-high reset signal for cordic_ip module
  wire ARESET;
  assign ARESET = ~S_AXI_ARESETN;

  //Code to transfer output data from cordic processor to output registers
  wire [C_S_AXI_DATA_WIDTH-1:0]     slv_wire2;
  wire [C_S_AXI_DATA_WIDTH-1:0]     slv_wire3;
  always @( posedge S_AXI_ACLK )
  begin
     slv_reg2 <= slv_wire2;
     slv_reg3 <= slv_wire3;
  end

  //Assign zeros to unused bits
  assign slv_wire2[31:1] = 31'b0;
  assign slv_wire3[15:12] = 4'b0;
  assign slv_wire3[31:28] = 4'b0;

  cordic_rtl cordic_rtl_inst( S_AXI_ACLK,     //clock,
                              ARESET,          //reset,
                              slv_reg0[0],     //start
                              slv_reg1[11:0], //angle_in
                              slv_wire2[0],    //ready_out,
                              slv_wire3[11:0],//sin_out,
                              slv_wire3[27:16]//cos_out
                              );

  // User logic ends
```

Note that we have mapped cordic_ip ports to the four slv_reg(0-1) register bits:
start → slv_reg0[0]
angle_in → slv_reg1[11:0]
ready_out → slv_reg2[0],
sin_out → slv_reg3[11:0]
cos_out → slv_reg3[27:16]

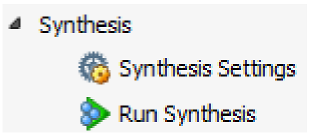
d) Remove all write operations to the slv_reg2 and slv_reg3 registers in the cordic_ip_v1_0_S00_AXI module.

The generated template code contains write operations to the slv_reg2 and slv_reg3 registers. We want our cordic_rtl to write data to those registers only.
In the template code, locate each "slv_reg2" and "slv_reg3" write code occurrence and comment the corresponding lines. List of changes is given below:

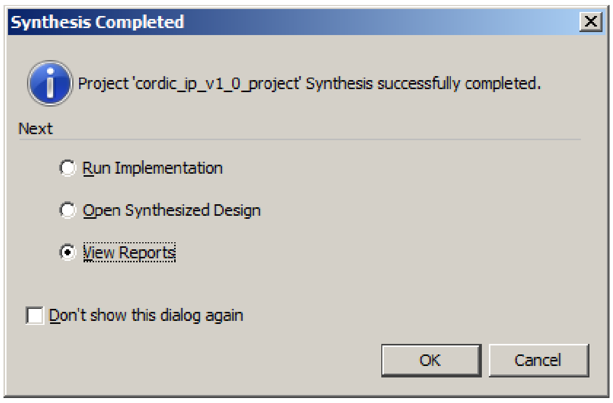
1. //slv_reg2 <= 0;
   //slv_reg3 <= 0;
2. //slv_reg2[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
   //slv_reg3[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
3. //slv_reg2 <= slv_reg2;
   //slv_reg3 <= slv_reg3;

Note, to leave "slv_reg2 <= slv_wire2;" and "slv_reg3 <= slv_wire3;" statements that have just been added in the previous tutorial step.

e) Perform trial synthesis to check code correctness.
Select "Run synthesis" from "Flow navigator"



Select "View report" when synthesis has finished


Synthesis Completed

Project 'cordic_ip_v1_0_project' Synthesis successfully completed.

Next

( ) Run Implementation

( ) Open Synthesized Design

(●) View Reports

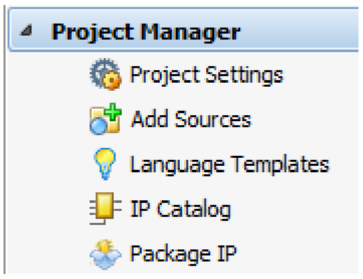
[ ] Don't show this dialog again

OK Cancel

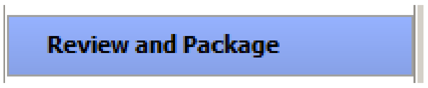
Check Tcl console for errors. If there are no errors go to next step.

f) Package and close IP project

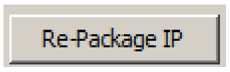
Select “Package IP” form “Flow navigator”



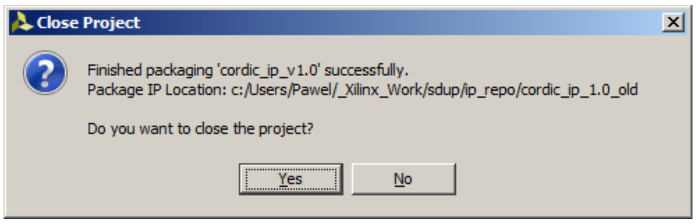
Select “Review and package” in “Package IP” window



Press “Re-package IP” button



In the dialog window select “Yes” to close “edit_cordic_ip_v1_0_project”



Finished packaging 'cordic_ip_v1.0' successfully.
Package IP Location: c:/Users/Pawel/_Xilinx_Work/sdup/ip_repo/cordic_ip_1.0_old

Do you want to close the project?

g) Go to the “cordic_acc” project window.

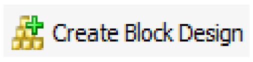
Note that we have created the Vivado “cordic_acc” project at the very start of this tutorial. It should be already opened on your computer, but if you have closed it, you can reopen it form the respective directory location.

4. Creating accelerated ARM-based design in Vivado

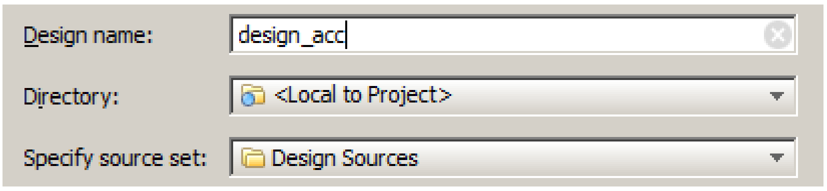
In this section, we will use Vivado's block diagram to integrate Zynq's ARM subsystem with FPGA-located cordic accelerator. We will use AXI bus as a system interconnect.

a) Create a new block diagram named “design_acc”

Click “Create block diagram” in “Flow navigator” window



Put the name “design acc” in “Create block design” dialog.

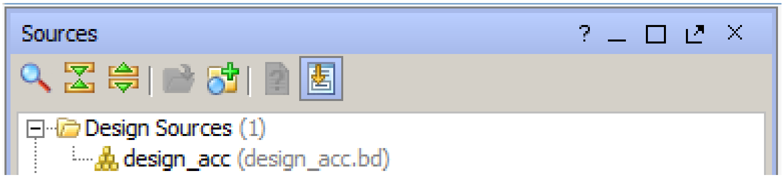

Design name: design_acc

Directory: <Local to Project>

Specify source set: Design Sources

and click “Ok”

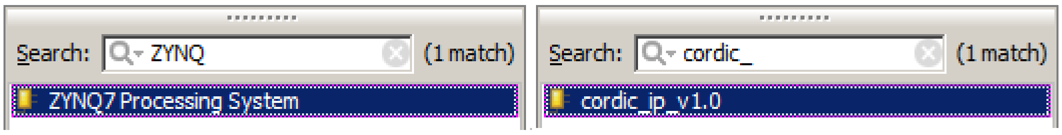
The new element “design_acc.bd” will appear in “Sources” window.



b) Add “ZYNQ processing system” and “cordic_ip_v1.0” IP cores to the design block

Click “Add IP” icon in the block diagram side menu

and select “ZYNQ processing system” element in the window menu. Repeat for “cordic_ip_v1.0”.


Search: ZYNQ (1 match)

ZYNQ7 Processing System

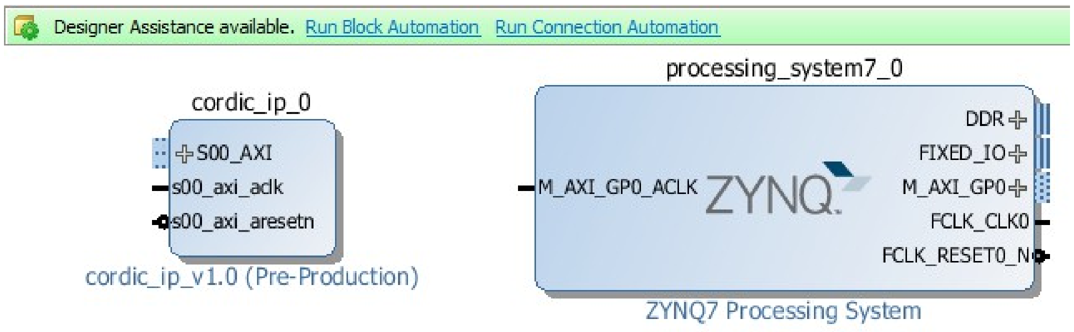
Search: cordic_ (1 match)

cordic_ip_v1.0

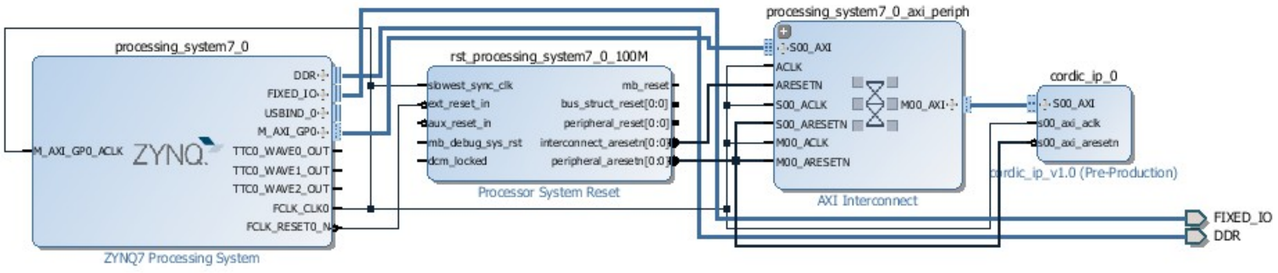
Now, the selected elements should appear in the block diagram window.

c) Use design connection and block automation to create design

Click “Run Block Automation” and “Run Connection Automation” in designer assistance panel

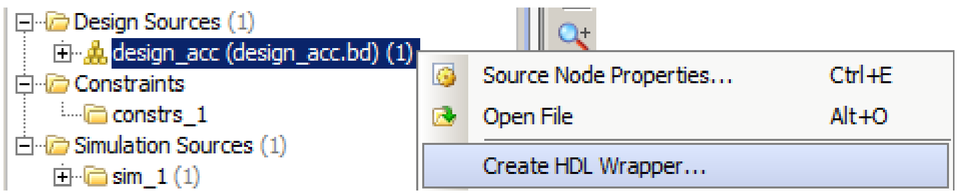


Our design is now ready for bitstream generation and export to the Vivado SDK tool.

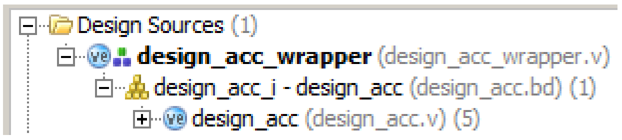


d) Create HDL wrapper for the block diagram.

Call context menu for the block diagram in “Source” window and select “Create HDL wrapper”

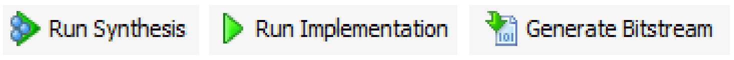


Vivado will generate the Verilog file “design_acc_wrapper.v” to instantiate the block design component

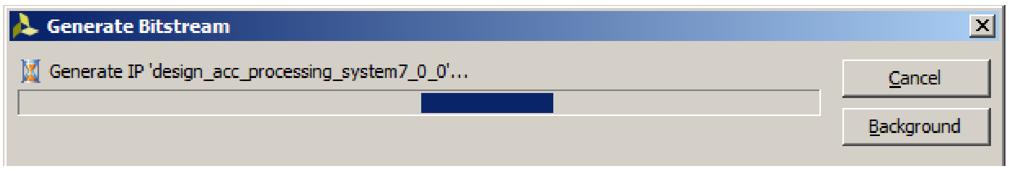

Design Sources (1)
design_acc_wrapper (design_acc_wrapper.v)
design_acc_i - design_acc (design_acc.bd) (1)
design_acc (design_acc.v) (5)

e) Generate bitstream.

From the Flow navigator panel select one after another: “Run synthesis”, “Run implementation” ,and “Generate bitstream”

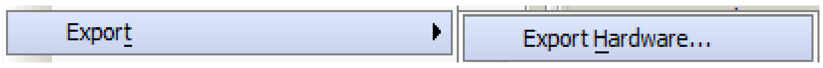


Wait until processes complete. It may take a moment ....

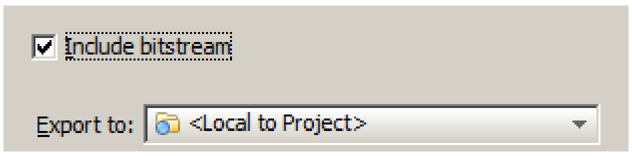


f) Export design and launch Vivado SDK tool.

From the main menu select File → Export → Export Hardware



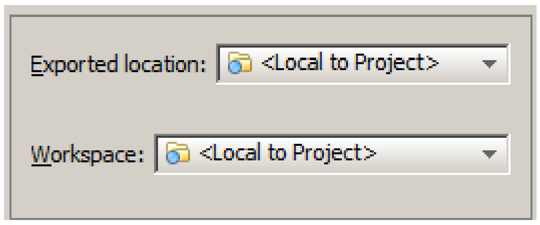
Select "Include bitstream" and local files location


☑ Include bitstream

Export to: <Local to Project>

From the main menu select File →Launch SDK

Choose export and the workspace location in the local directory

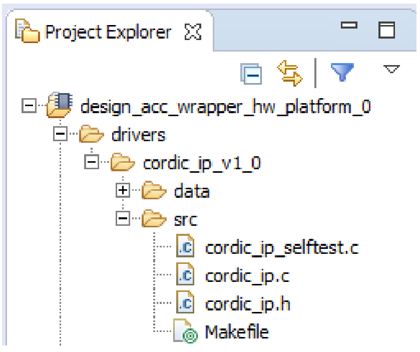
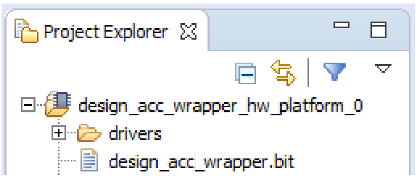
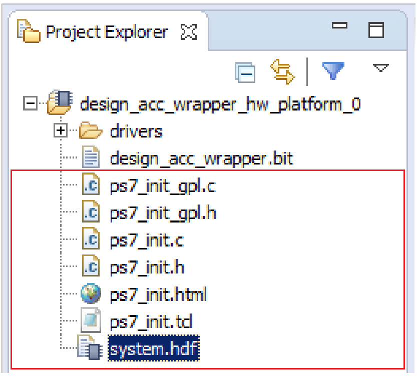

Exported location: <Local to Project>

Workspace: <Local to Project>

The Vivado SDK project should appear in the SDK application window.

5. Creating application in Vivado SDK

The SDK GUI is now opened by the Vivado tool. The project that is automatically created contains hardware platform files for our accelerated design. In the "Project explorer" window you can find: a) a driver source files for the cordic module, b) bitstream configuration file for FPGA, and c) other files that are the processing system specific.

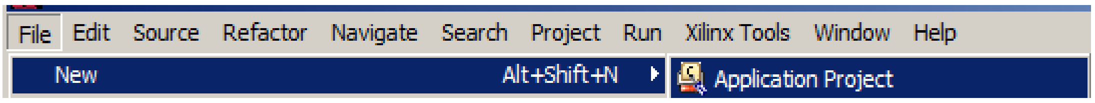
a)	b)	c)
		
Project Explorer design_acc_wrapper_hw_platform_0 drivers cordic_ip_v1_0 data src cordic_ip_selftest.c cordic_ip.c cordic_ip.h Makefile	Project Explorer design_acc_wrapper_hw_platform_0 drivers design_acc_wrapper.bit	Project Explorer design_acc_wrapper_hw_platform_0 drivers design_acc_wrapper.bit ps7_init_gpl.c ps7_init_gpl.h ps7_init.c ps7_init.h ps7_init.html ps7_init.tcl system.hdf

Explore the files in Project Explorer before you proceed.

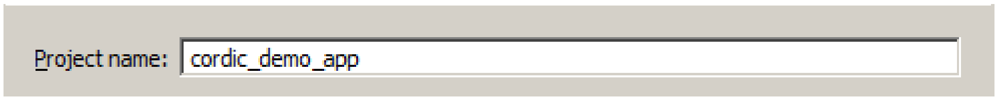
Now, we will create a demo application for our hardware design. The application will accept an angle value from the user, run accelerator to calculate sine and cosine values, and return results to the user. The build-in UART interface of the ARM subsystem will be used as an user input/output interface. We will develop an application in the C language.

a) Add an application project and board support package (BSP) files

Select File → New → Application Project form the main menu.

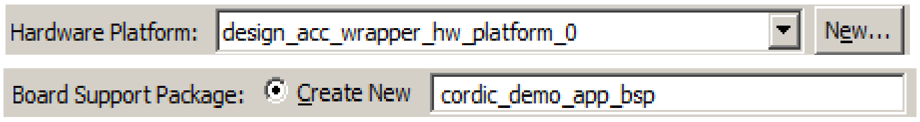


In the “New Project” dialog, enter “cordic_demo_app” as the project name.



Project name: cordic_demo_app

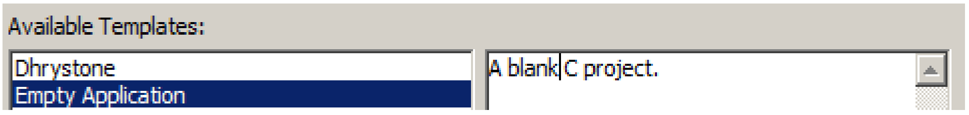
Leave the names of the hardware platform and board support package as suggested.



Hardware Platform: design_acc_wrapper_hw_platform_0

Board Support Package: Create New cordic_demo_app_bsp

Click the “Next” button.
Select “Empty application” from the available templates.



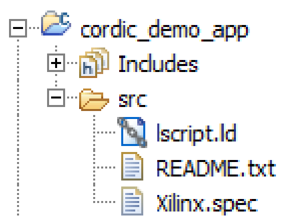
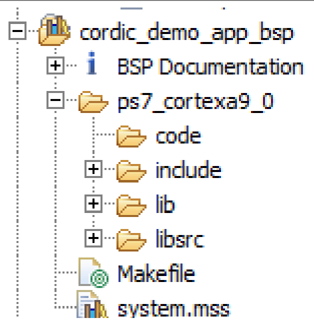
Available Templates:

Dhrystone
Empty Application

A blank C project.

Click the “Finish” button.

You have created two new elements in Project Explorer a) “cordic_demo_app” application project and b) cordic_demo_app_bsp

a)	b)
	

“Cordic_demo_app” will contain your application source files. “Cordic_demo_app_bsp” contains files that abstract the hardware platform. Thanks to that abstraction layer your application will be portable and it can run on any processor platform which provides the compatible functionality (like UART and cordic accelerator for example).

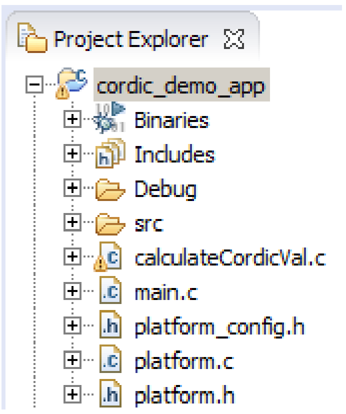
b) Add source files to the application project.

Locate the source files in the provided tutorial sources:
main.c", "calculateCordicVal.c", "plafrorm.c", "platform.h", "platform_config.h"

Copy all provided source files to the "cordic_demo-app" folder of your SDK application using Windows Explorer.
Note. Typically, the application folder is a subfolder in your SDK project location directory: "<*Project directory location*>\cordic_acc\cordic_acc.sdk\cordic_demo_app"

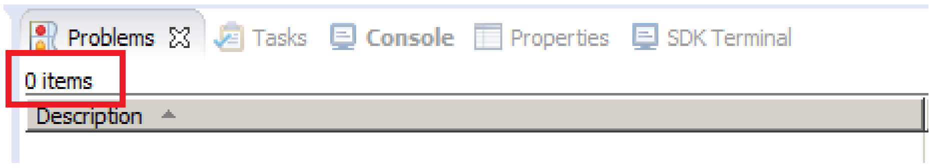
Refresh the "Project Explorer " view by the "File → Refresh" command in the main window menu.

The added files should appear in the "cordic_demo_app" project.



c) Build the application

Select "Project → Build all" and check "Problems" panel to verify the application was built without problems.



Eliminate problems if any.

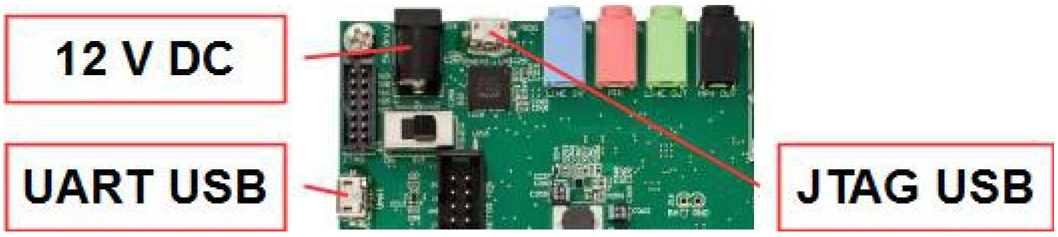
You have binary software file for your ZYNQ platform ready. Now you can run the software on the development board.

6. Running application in Vivado SDK

In this section, we will connect the Zedboard development kit to the PC, and we will download FPGA bitstream and software binaries to run the application. For Zedboard, we use the JTAG interface for FPGA configuration, software download and debugging (if necessary).

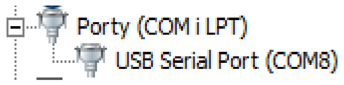
a) Connect Zedboard to the PC and power supply

Connect 12 V adapter to Zedboard power socket, and two USB cables to JTAG and UART ports.



b) Find your USB serial port number in Windows Device Manager

Select Device Manager in Windows Control Panel and expand ports branch.



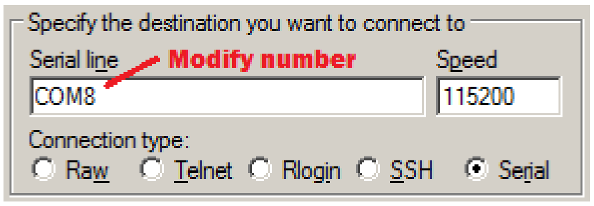
Along with the picture, we will use COM8 in this tutorial.

c) Open PuTTY application for serial communication with Zedboard

You can also use another serial communication software like RealTerm for example.

d) Configure PuTTY connection and open terminal window

Select "Serial" interface, COM name, and 115200 baud rate.



Specify the destination you want to connect to

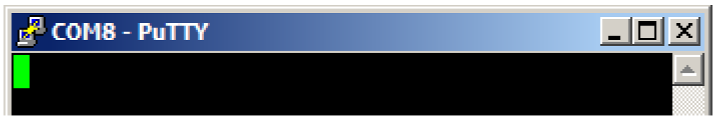
Serial line **Modify number** Speed

COM8 115200

Connection type:

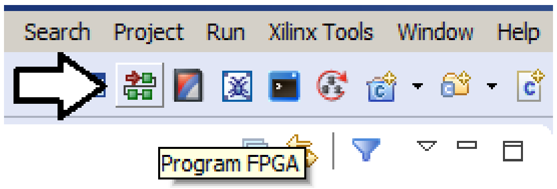
Raw Telnet Rlogin SSH [✓ Serial]

Click OK
Your terminal window is now opened

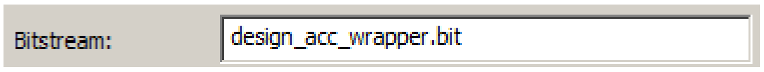


e) Configure FPGA with the bitstream file

Go to Vivado SDK and select the “Program FPGA” button in main menu.

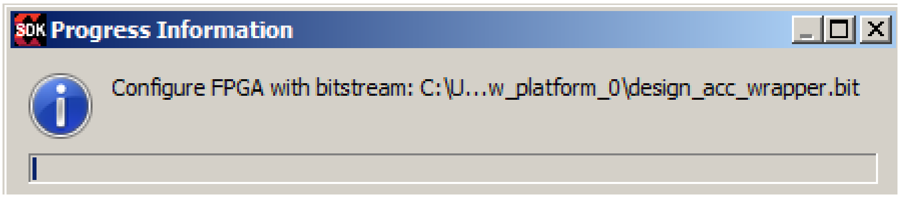


Check if the bitstream file is “design_acc_wrapper.bit” and click OK.



Bitstream: design_acc_wrapper.bit

Wait until configuration completes.



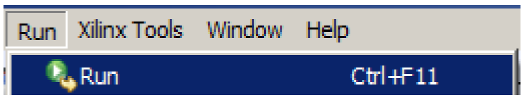
Progress Information

Configure FPGA with bitstream: C:\U...w_platform_0\design_acc_wrapper.bit

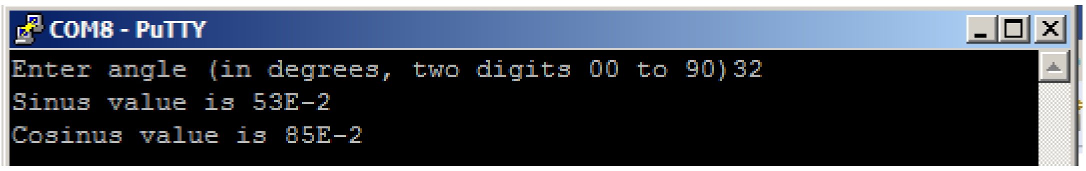
Proceed to the next step if no errors.

f) Run application

Select Run → Run from the main menu



Check the terminal window for the application output string. Enter your sngle value and check results.



COM8 - PuTTY

```
Enter angle (in degrees, two digits 00 to 90)32
Sinus value is 53E-2
Cosinus value is 85E-2
```

7. Exercises

1. Modify the software application to allow the user enter any angle value in range from 0 to 360 degree.