AGH University of Science and Technology
Cracow
Department of Electronics

Custom system design in FPGA laboratory

Tutorial 2

Simulation of the AXI-based accelerated system

Author: Paweł Russek

ver. 2018.01.09

1. Introduction

1.1. *Glossary*

AXI-Light – is a sub-class of AXI bus standard that is a system-on-chip bus standard used *e.g.* by ARM processors for peripheral interfacing. AXI is a part of Advanced Microcontroller Bus Architecture (Advanced Microcontroller Bus Architecture) that is introduced by ARM as an open-standard for on-chip interconnect.

Microblaze – a processor design for Xilinx's FPGAs. It is called a soft-processor as it does not exist as a hard-wired element. It is essentially an IP-Core for FPGA-based SoCs and it comes into existence via FPGA configuration. Microblaze uses AXI bus for pheripheral interconnect.

1.2. *Objectives*

The goal of this tutorial is to present the step-by-step procedure of creating the accelerator module for sin/cos calculations for the AXI-base system-on-chips.

We will create custom peripheral module for the cordic algorithm. In this process the custom cordic processor that was created in the Tutorial 1 of this laboratory classes will be used. We will add AXI-Light interface to the cordic processor to enable its use in the SoC designs with ARM processors.

For the verification of the new cordic IP-Core we will use Microblaze processor system. The simulation is an essential part of the custom peripheral design. The reason to create a Microblaze-based SoC is the simplicity of this soft-processor that enables system simulation.

For the creation of the cordic IP-Core and SoC design, we will use Xilinx's Vivado development software.

1.3. *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.

1.4. *Software and hardware*

- Vivado v2016.2 (64-bit)
- The cordic_rtl.v file that is a result of Tutorial

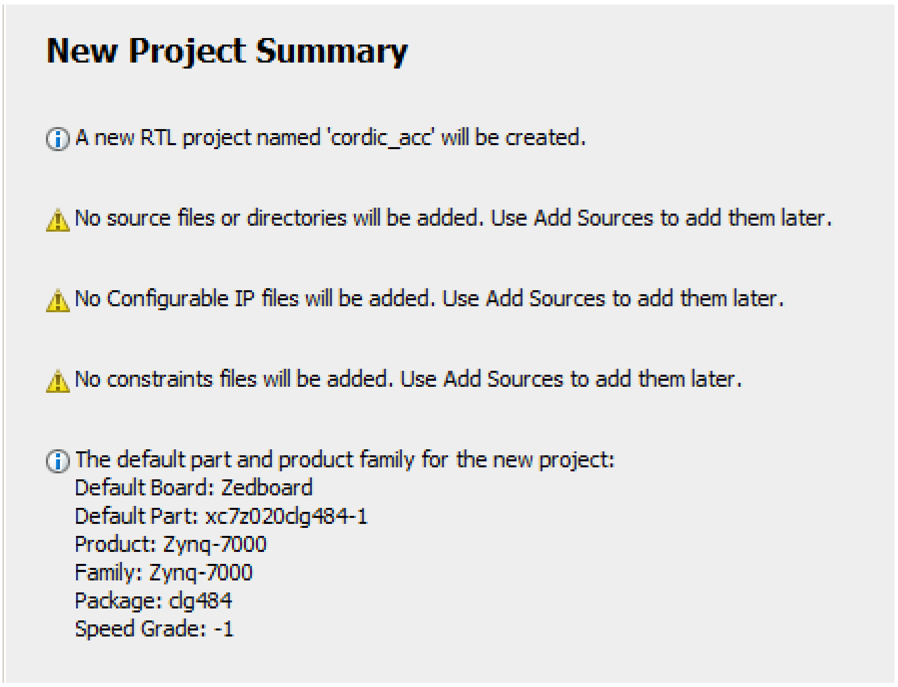
2. Creating Cordic IP

Before our cordic processor code can be incorporated into the Microblaze system, it has to be converted into the IP block that conforms the AXI-Light standard.

We will use Vivado wizard tool to create the template HDL code for AXI-Light peripheral. Next, to implement the desired peripheral functionality, we will add to the project our cordic processor code.

2.1 Launch Vivado on your computer.

2.2 Create new project "cordic_test" for Zedboard.
Check your project settings in the figure below.



New Project Summary

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

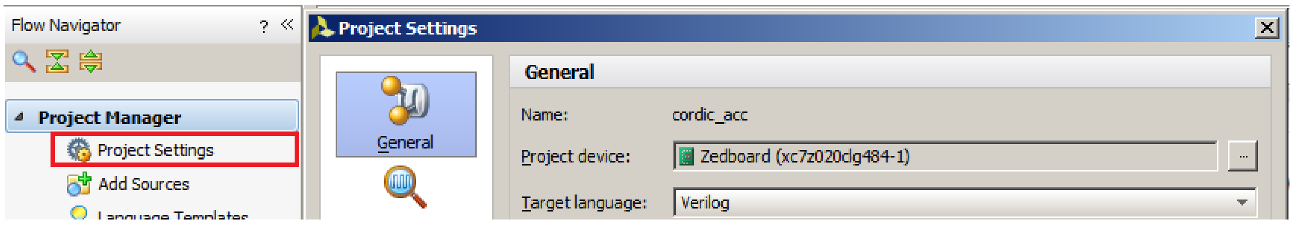
No source files or directories will be added. Use Add Sources to add them later.

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

2.3 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)

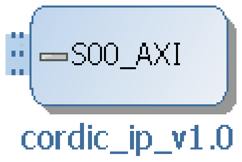
Target language: Verilog

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

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

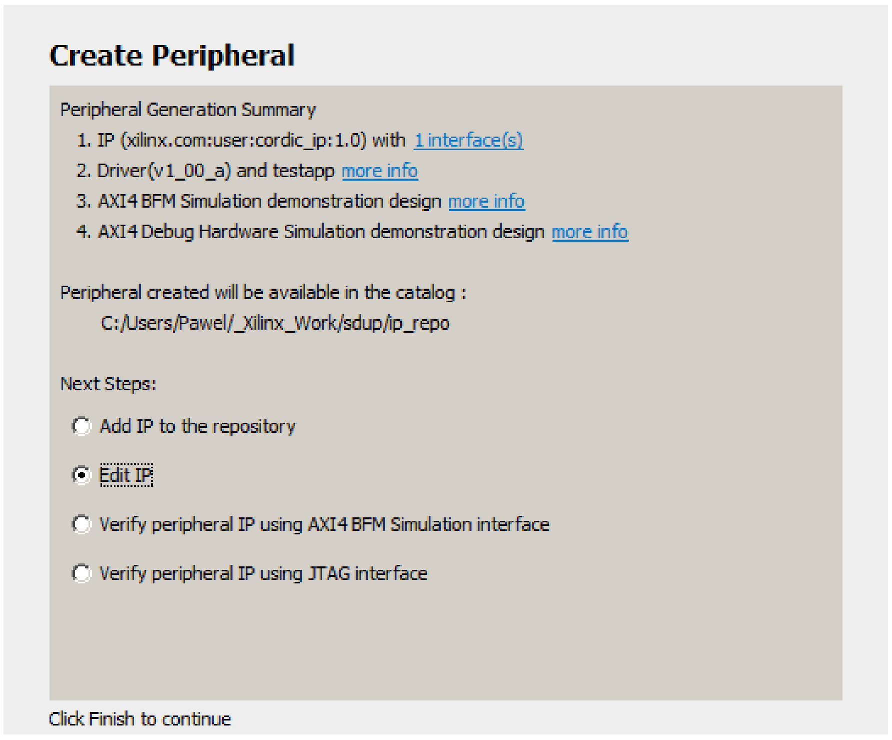
2.6 In *"Peripheral Details" window"* name the IP as "cordic_ip" and note the IP directory location.
You can select the directory according your preferences or leave the default location.
Click Next button

2.7 Leave one Slave AXI Lite interface for your IP.



Click Next button

2.8 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

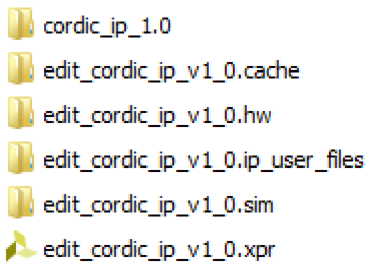
Next Steps:
- ○ Add IP to the repository
- [✓ Edit IP]
- ○ Verify peripheral IP using AXI4 BFM Simulation interface
- ○ Verify peripheral IP using JTAG interface

Click Finish to continue

Read information in boxes that appear on click of "more info" links.
Select "Edit IP".
Click Finish button

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

You should be able to find the "edit_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

The new IP will be added to "cordic_acc" project later. You can edit IP functionality any time opening "edit_cordic_ip" project in Vivado.

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, and the peripheral can be connected both to Zynq's ARM subsystem and Microblaze system.

However, created Verilog code for the IP contains only logic for the AXI-Light interface and four memory mapped registers r0-r3. 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 architecture of the Vivado's peripheral IP.

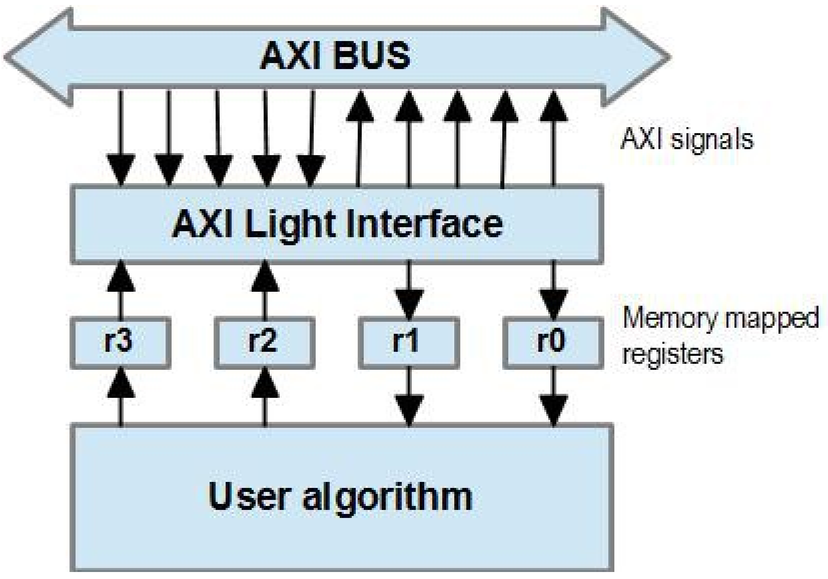


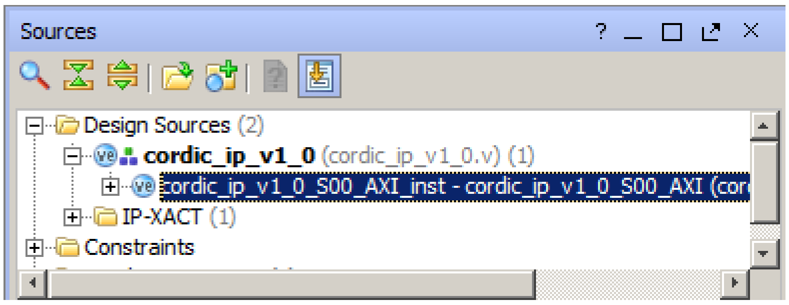
Figure 1

3.1 Open cordic_ip in Vivado

Go to "./ip_repo" subdirectory of your main project directory and open "edit_cordic_ip.xpr" if it is not already open on your computer (see step 2.9).

3.2 Locate and open the "cordic_ip_v1_0_S00_AXI.v" file in the Vivado editor

Go to "Design sources" and click "cordic_ip_v1_0_S00_AXI".


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 (cor
IP-XACT (1)
Constraints

The "cordic_ip_v1_0_S00_AXI.v" file wiil appaer in editor window.

3.2 Add "cordic_rtl" module definition at the end of the "cordic_ip_v1_0_S00_AXI.v" file.

The "cordic_rtl" module is a module created in Tutorial 1 of this laboratory. Copy and paste the code for cordic_rtl.v to "cordic_ip_v1_0_S00_AXI.v"

The cordic_rtl.v file can be downloaded here.

3.3 Locate the "Add user logic" section in "cordic_ip_v1_0_S00_AXI.v" and add "cordic_rtl" module instantiation.

The corresponding instantiation code is given below:

```
// Add user logic here


  //Reset signal for cordic processor
  wire ARESET;
  assign ARESET = ~S_AXI_ARESETN;

  //Transfer output 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,       //angle_in
                            slv_wire2[0],   //ready_out,
                            slv_wire3[11:0],//sin_out,
                            slv_wire3[27:16]//cos_out
                            );

// User logic end
```

Note, that the four peripheral registers r0-r3 are named *slv_reg0 – slv_reg3* in the Verilog code. We have connected “cordic_rtl” ports to the bits of the AXI peripheral registers. The connection mapping is summarised in Table 1.

cordic_rtl port	**AXI mapped register**	**Direction**	**Port width**
start	slv_reg0(0)	input	1
angle_in	slv_reg1(11:0)	input	12
ready_out	slv_reg2(0)	output	1
sin_out	slv_reg3(11:0)	output	12
cos_out	slv_reg3(27:16)	output	12

3.4 Find write assignments to slv_reg2 and slv_reg3 in the peripheral code. We have “cordic_rtl” to be the only signal source for slv_reg2 and slv_reg3 so the rest of the assignments has to be commented.

1:
```
if ( S_AXI_ARESETN == 1'b0 )
begin
       slv_reg0 <= 0;
       slv_reg1 <= 0;
       //slv_reg2 <= 0;
       //slv_reg3 <= 0;
end
```

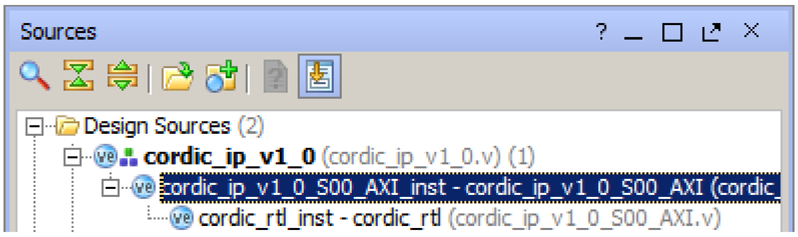
2:
```
//slv_reg2[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
```

3:
```
//slv_reg3[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];
```

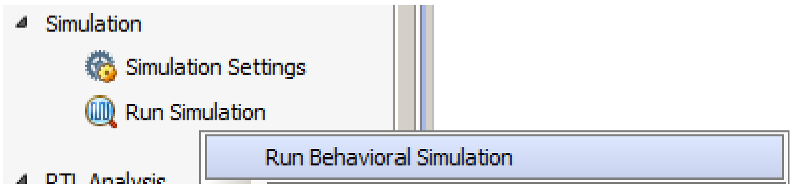
Alternatively to the steps 3.2, 3.3, and 3.5, you can take the modified cordic_ip_v1_0_S00_AXI.v form the repository here.

3.5 Check the correctness of your project.

The file structure should look like (no error icons)


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
cordic_rtl_inst - cordic_rtl (cordic_ip_v1_0_S00_AXI.v)

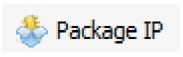
Also, for ultimate code check, run the trial project simulation.



Check for possible errors and problems in "**Tcl Console**". If there is no errors, proceed with the tutorial. Correct problems, otherwise.

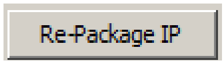
3.6 Update IP library and close IP project

Select "**Package IP" form "Project manager"**



Select "**Review and Package**"

Select "Re-Package IP" and select to close the "edit_cordic_ip" project.



The last action updated data in the IP library and closed the "edit_cordic_ip" project.

4. Creating Microblaze accelerated project

In this part of the tutorial we will create the Microblaze system with the cordic accelerator connected to the processor AXI bus. The block diagram of the final system is presented in Figure 2.
The Microblaze processor features the **harvard architecture**, as it has separate data and instruction interfaces. Further, Microblaze can be connected to the Local Memory Bus (LMB) or AXI bus. Consequently, Microblaze has four bus interfaces: I_LMB, D_LMB (Instruction and Data LMBs), I_AXI, and D_AXI (Instruction and Data AXI). The LMB bus is a proprietary Xilinx solutions that is dedicated for Block RAMs interfacing. Block RAMs are internal FPGA memory blocks that have two access ports (dual-port memories). Thus, BRAMs allows the system for the two memory operations at the same clock cycle. Consequently, I_LMB and D_LMB can be connected to the same memory block (see Figure 2).

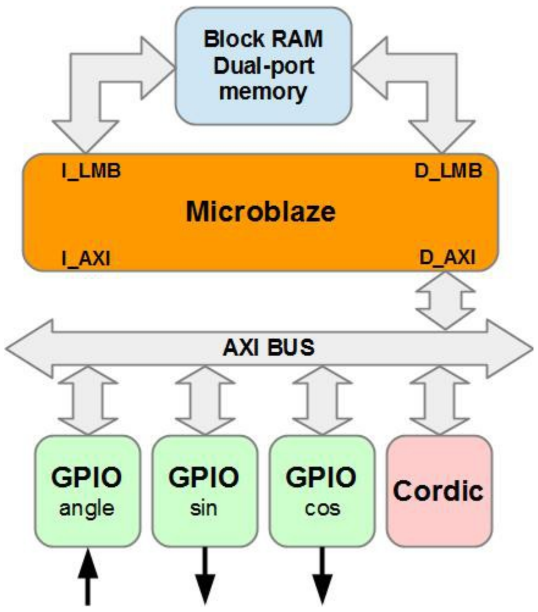
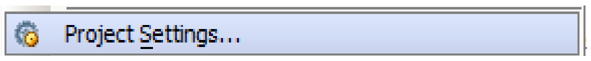


Figure 2

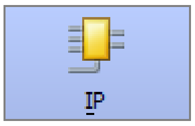
In our test system, we will use LMB bus to connect program and data memory as dual port BRAM, and AXI to connect peripherals. We will use GPIO to system input/output interface. The In our scheme, we want Microblaze to read the binary angle value form GPIO_angle, run calculations in Cordic accelerator , and send results to GPIO_sin and GPIO_cos.

4.1. Adding "cordic_ip" library to the "cordic_acc" project

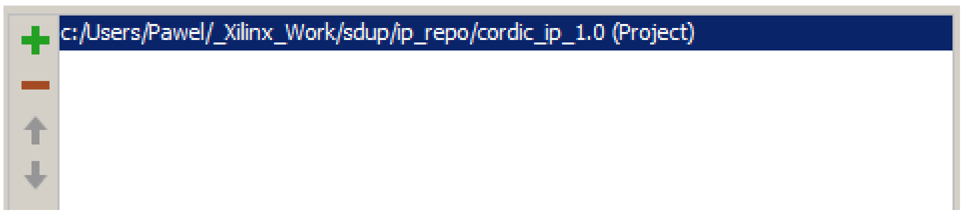
4.1.1 Select "Tools"->"Project settings" for the menu



4.1.2 Select "IP" in the side menu



4.1.3 Add "./ip_repo/cordic_ip_1.0" catalog to the repositories list

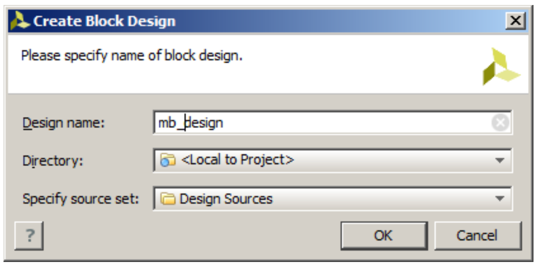

c:/Users/Pawel/_Xilinx_Work/sdup/ip_repo/cordic_ip_1.0 (Project)

4.1.4 Click "OK" button in the "Project settings" window to complete

4.2. Creating the new block diagram

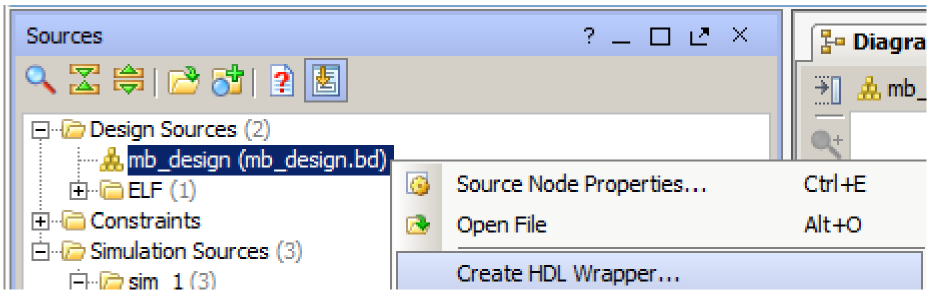
4.2.1 Select "Create block design"

4.2.2 Create "mb_design"


Create Block Design
Please specify name of block design.
Design name: mb_design
Directory: <Local to Project>
Specify source set: Design Sources
?
OK
Cancel

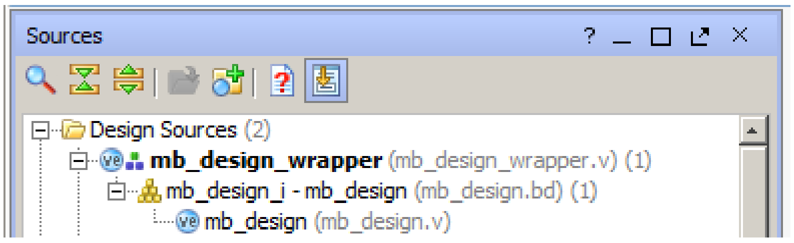
4.2.3 Create HDL wrapper for the block design

Right-click "**mb_design**" entry in the "**Sources**" window, and select "Create HLD wrapper"



4.2.4 Select "Allow Vivado to manage wrapper and autoupdate"

4.2.5 Verify your "Sources" window with the fugure below

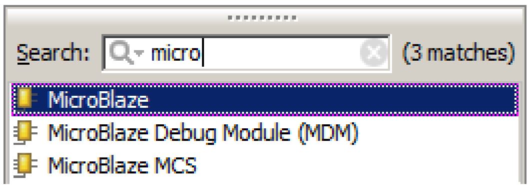

Sources
Design Sources (2)
mb_design_wrapper (mb_design_wrapper.v) (1)
mb_design_i - mb_design (mb_design.bd) (1)
mb_design (mb_design.v)

4.3. Creating Microblaze system block diagram

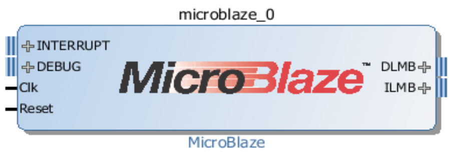
4.3.1 Add microblaze to the design

Select add IP button from the block diagram side menu

Select microblaze IP form the list


Search: micro (3 matches)
MicroBlaze
MicroBlaze Debug Module (MDM)
MicroBlaze MCS

The Microblaze IP appears in the block diagram



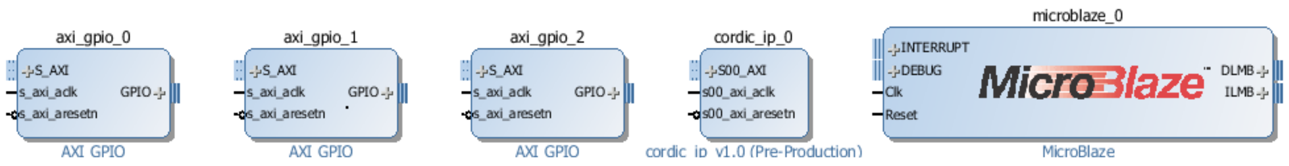
4.3.2 Add three GPIO IPs

Repeat the 4.3.2 procedure for GPIO three times

4.3.3 Add cordic_ip

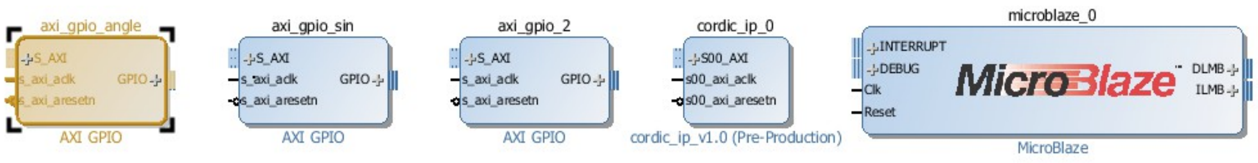
Repeat the 4.3.2 procedure for cordic_ip_1.0

4.3.4 Verify your results in figure below



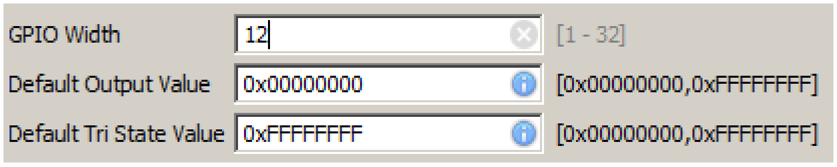
4.3.4 Rename GPIOs

Rename "axi_gpio_0" to axi_gpio_angle
Rename "axi_gpio_0" to axi_gpio_sin
Rename "axi_gpio_0" to axi_gpio_cos



4.3.5 Set “axi_gpio_angle” bit-width to 12 bits

Double-click “axi_gpio_angle” symbol and change “GPIO Width”



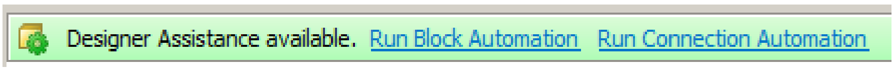
GPIO Width	12	[1 - 32]
Default Output Value	0x00000000	[0x00000000,0xFFFFFFFF]
Default Tri State Value	0xFFFFFFFF	[0x00000000,0xFFFFFFFF]

4.3.6 Set “axi_gpio_sin” and “axi_gpio_cos” bit-width to 12 bits

Repeat 4.3.5 for “axi_gpio_sin” and “axi_gpio_cos”

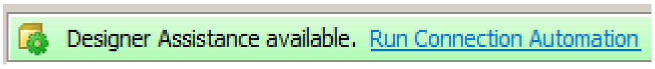
4.3.7 Run “Block automation”

Click “Block automation” in the “Designer assistance” menu

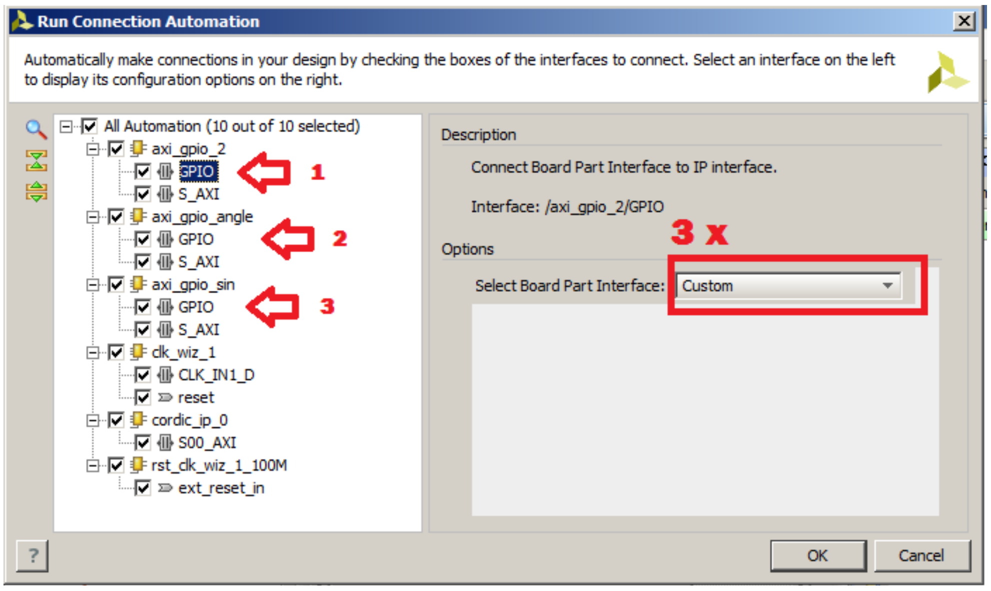

Designer Assistance available. Run Block Automation Run Connection Automation

4.3.8 Run “Connection automation”

Click “Connection automation” in the “Designer assistance” menu



Select “Custom” for “Board Part Interface” for all three GPIOs !!!



Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

Description

Connect Board Part Interface to IP interface.

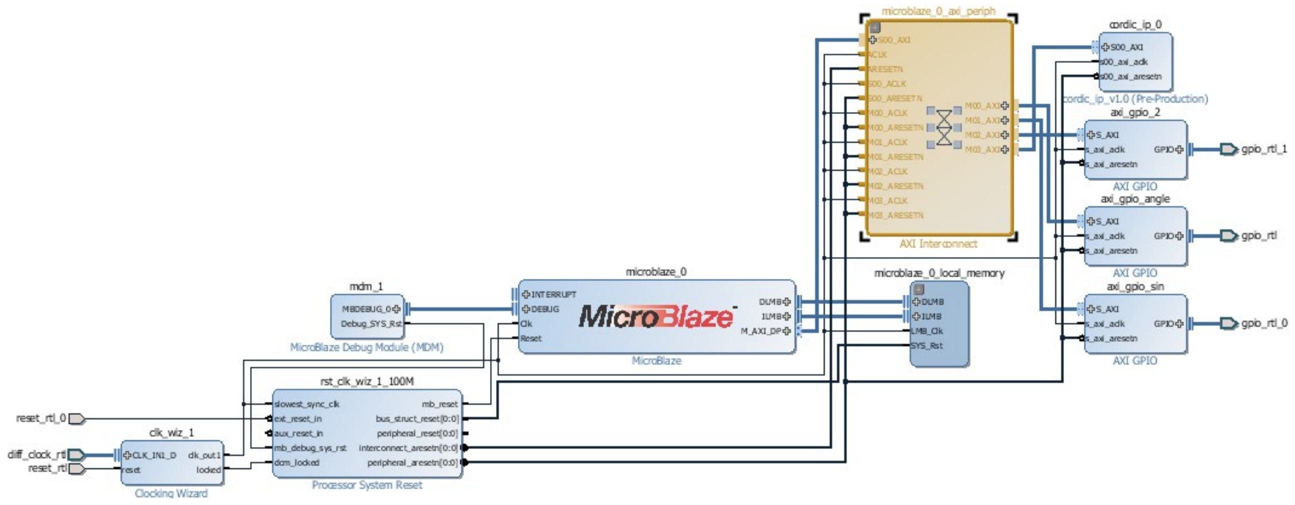
Interface: /axi_gpio_2/GPIO

Options

Select Board Part Interface: Custom

Click OK

The block diagram will appear



4.3.9 Change the port names

Change “gpio_rtl_0” to “gpio_angle”
Change “gpio_rtl_1” to “gpio_sin”
Change “gpio_rtl_2” to “gpio_cos”

Change “reset_rtl” to “reset”
Change “reset_rtl_0” to “reset_n”
Change “diff_clock_rtl” to “clock”

4.3.10 Verify your “mb_design_wrapper.v”

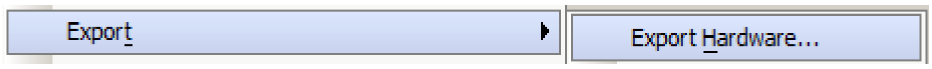
Check if the wrapper corresponds to the block diagram. Generate the wrapper again if necessary (see section 4.2)

5. Export hardware platform to SDK

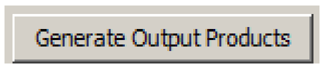
In this section, we will create the hardware platform specification for the Vivado Software Development Kit (SDK). The specification will be necessary to build Board Support Platform (hardware abstraction layer) that allow programmer to write portable programs for the Microblaze system.

5.1 Export hardware information

Select “File”->”Export”->”Export hardware” form the main Vivado menu.

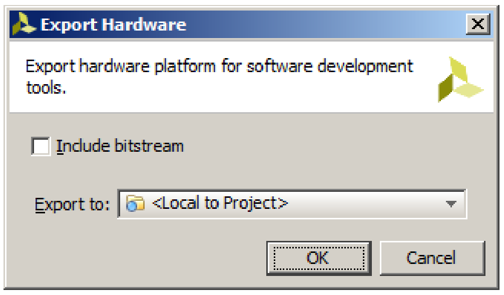


Select “Generate Output Products” in the “Generate Output Products” window



Click OK when generation is completed

Export to the local project folder: <Local to Project>



Export hardware platform for software development tools.

☐ Include bitstream

Export to: <Local to Project>

Click OK

6. Build software program for Microblaze

We will build the code that reads binary angle value from GPIO_angle, sends it to Cordic accelerator , read the results form the accelerator, and sends sinus (cosinus) value to GPIO_sin (GPIO_cos).

First the GPIOs are initialised (GPIO_angle as input, GPIO_sin and GPIO cos as outputs) the data is read from GPIO_angle.

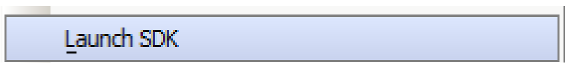
Then data is written into *slv_reg1(11:0)* register and the conversion is started with the *start* bit set in *slv_reg0(0).*

Later, the Micoblaze waits for the end of conversion what is signalled by *ready_out* bit in *slv_reg2(0).*

At the end, results are read from *slv_reg3* register and send to GPIO_sin and GPIO_cos.

6.1 Open Vivado SDK

Select “File”->”Launch SDK” in Vivado

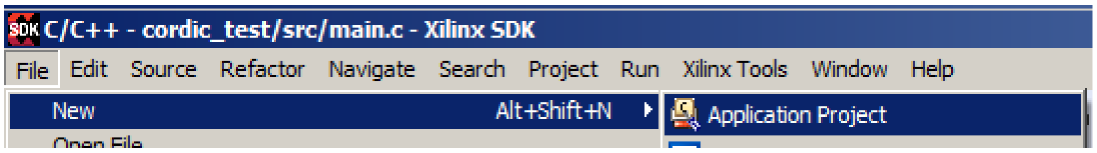


Click OK

The Microblaze software project opens in Vivado SDK window

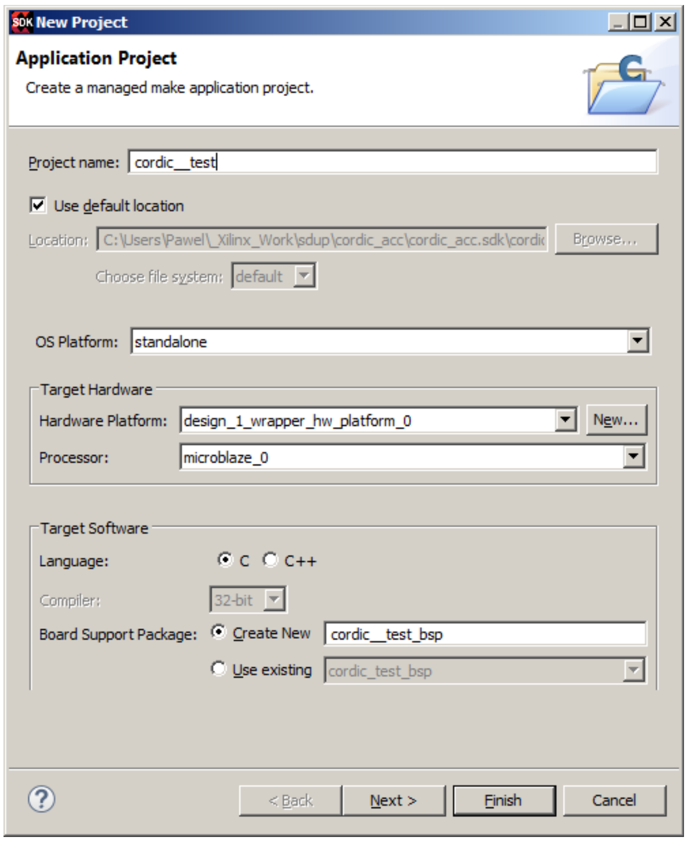
6.2 Create the new application project “cordic_test” in Vivado SDK

Select “File”->”New”->”Application Project”


C/C++ - cordic_test/src/main.c - Xilinx SDK
File Edit Source Refactor Navigate Search Project Run Xilinx Tools Window Help
New Alt+Shift+N
Open File
Application Project

Edit project name in the “New project” window.

Enter “cordic_test”



Project name: cordic__test

[✓] Use default location

Location: C:\Users\Pawel_Xilinx_Work\sdup\cordic_acc\cordic_acc.sdk\cordic

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: design_1_wrapper_hw_platform_0

Processor: microblaze_0

Target Software

Language: [✓ C] C++

Compiler: 32-bit

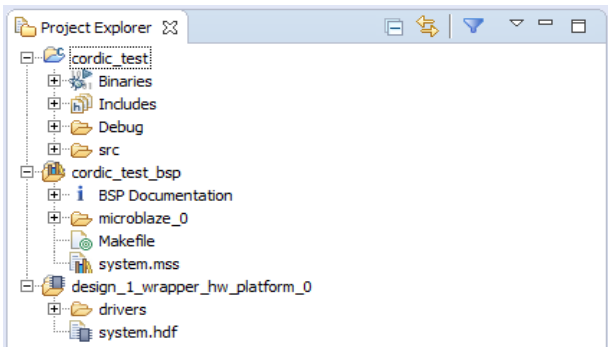
Board Support Package: [✓ Create New] cordic__test_bsp

Use existing cordic_test_bsp

Click Finish

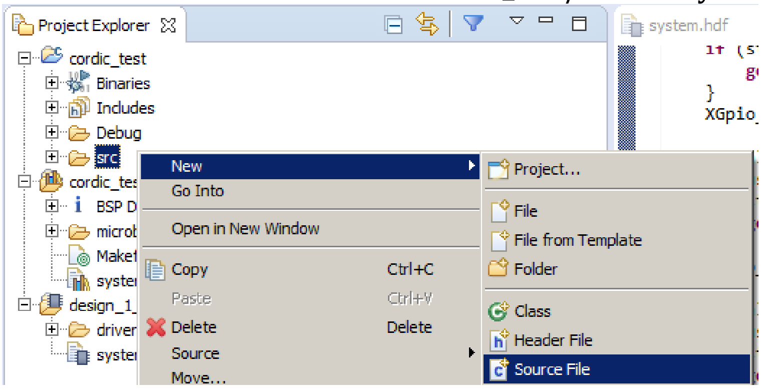
6.3 Verify your project hierarchy

The project contains:
cordic_test – microblaze application sources
cordic_test_bsp – board support platform sources
design_1_wrapper_hw_platform_0 – hardware platform specification and peripheral drivers.

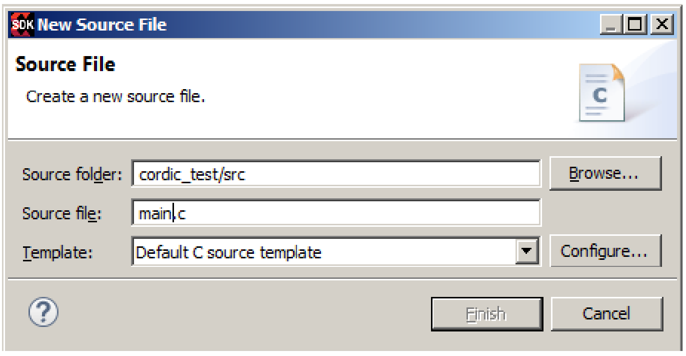


6.4 Add the "main.c" source file to "cordic_test/src"

Select "New"->"Source File" from "cordic_test/src" entry context menu



Enter "main.c" name in the "New Source File" window



Source File

Create a new source file.

Source folder: cordic_test/src

Source file: main.c

Template: Default C source template

Click Finish

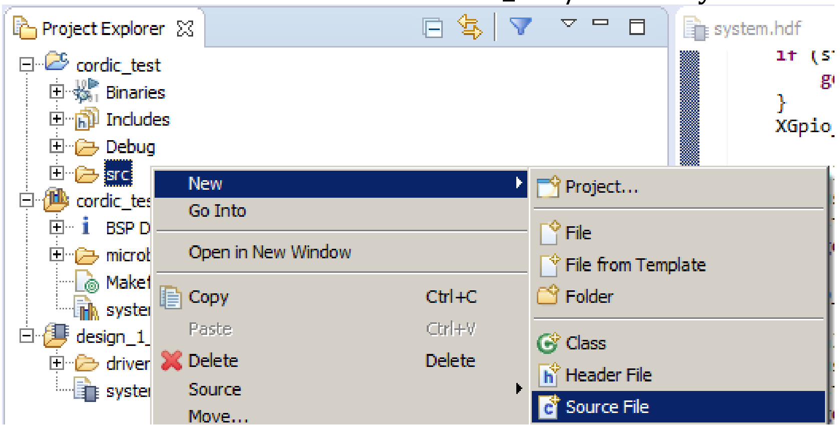
6.5 Insert the microblaze source code into “main.c”

Copy the code below into “main.c”

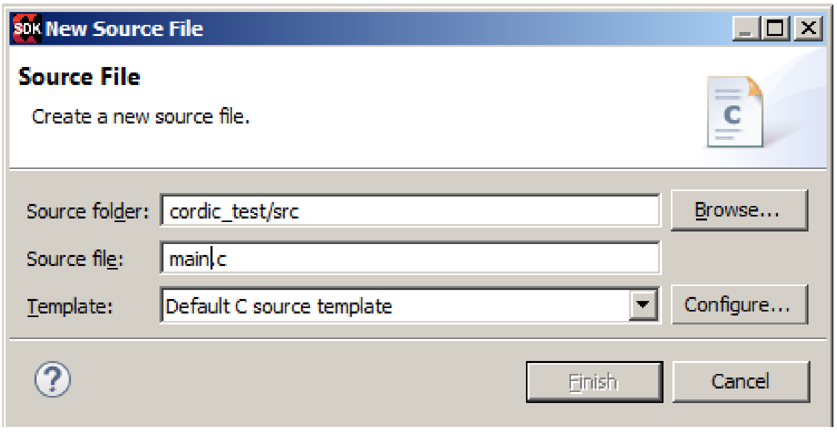
```c
#include "xparameters.h"
#include "xgpio.h"
#include "cordic_ip.h"

/***************************** user definitions *********************************/
#define CHANNEL 1

//Cordic processor base addres redefinition
#define CORDIC_BASE_ADDR      XPAR_CORDIC_IP_0_S00_AXI_BASEADDR

//Cordic processor registers' offset redefinition
#define CONTROL_REG_OFFSET    CORDIC_IP_S00_AXI_SLV_REG0_OFFSET
#define ANGLE_REG_OFFSET      CORDIC_IP_S00_AXI_SLV_REG1_OFFSET
#define STATUS_REG_OFFSET     CORDIC_IP_S00_AXI_SLV_REG2_OFFSET
#define RESULT_REG_OFFSET    CORDIC_IP_S00_AXI_SLV_REG3_OFFSET
#define RESULT_REG_SIN(param) ((u32)param & (u32)(0x00000FFF))
#define RESULT_REG_COS(param) (((u32)param  & (u32)(0x0FFF0000)) >> 16 )

/****************************** Main function *********************************/

int main(){
int status;
XGpio angleGpio,  sinGpio, cosGpio;
u32 data;
u32 result, sin, cos;

/* Initialize driver for the input angle GPIOe */
      status = XGpio_Initialize(&angleGpio, XPAR_AXI_GPIO_ANGLE_DEVICE_ID);
      if (status != XST_SUCCESS) {
            goto FAILURE;
      }
      XGpio_SetDataDirection(&angleGpio, CHANNEL, 0xFFF);

/* Initialize driver for the output sin GPIO  */
      status = XGpio_Initialize(&sinGpio, XPAR_AXI_GPIO_SIN_DEVICE_ID);
      if (status != XST_SUCCESS) {
            goto FAILURE;
      }
      XGpio_SetDataDirection(&sinGpio, CHANNEL, 0x000);

/* Initialize driver for the output sin GPIO  */
      status = XGpio_Initialize(&cosGpio, XPAR_AXI_GPIO_COS_DEVICE_ID);
      if (status != XST_SUCCESS) {
            goto FAILURE;
      }
      XGpio_SetDataDirection(&cosGpio, CHANNEL, 0x000);

//Read angle binary data from angle GPIO. fxp(12:10) format
      data = XGpio_DiscreteRead(&angleGpio, CHANNEL);


//Send data to data register of cordic processor
      CORDIC_IP_mWriteReg(CORDIC_BASE_ADDR, ??????????????, data);

//Start cordic processor - pulse start bit in control register
      CORDIC_IP_mWriteReg(CORDIC_BASE_ADDR, ??????????????, 1);
      CORDIC_IP_mWriteReg(CORDIC_BASE_ADDR, ??????????????, 0);
//Wait for ready bit in status register
      while( (CORDIC_IP_mReadReg(CORDIC_BASE_ADDR, ??????????????) & 0x01) == 0);

//Get results
      result = CORDIC_IP_mReadReg(CORDIC_BASE_ADDR, ??????????????);

//Extract sin and cos from 32-bit register data
      sin = RESULT_REG_SIN( result );
      cos = RESULT_REG_COS( result );

//Send to GPIO
      XGpio_DiscreteWrite(&sinGpio, CHANNEL, sin);
      XGpio_DiscreteWrite(&cosGpio, CHANNEL, cos);

/* Failure or end trap */
FAILURE:
      while(1);

}
```

6.4 Add the “main.c” source file to “cordic_test/src”

Select “New”->”Source File” from “cordic_test/src” entry context menu



Enter “main.c” name in the “New Source File” window


New Source File

Source File

Create a new source file.

Source folder: cordic_test/src

Source file: main.c

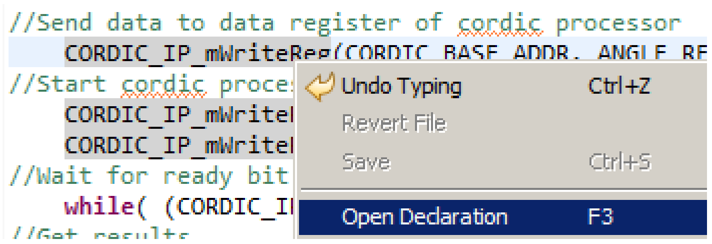
Template: Default C source template

Click Finish

6.5 Replace question mark strings with correct symbols

All hardware platform definitions are included in “xparameters.h” header file. The file contains ids, base addresses, and offsets of peripheral devices,

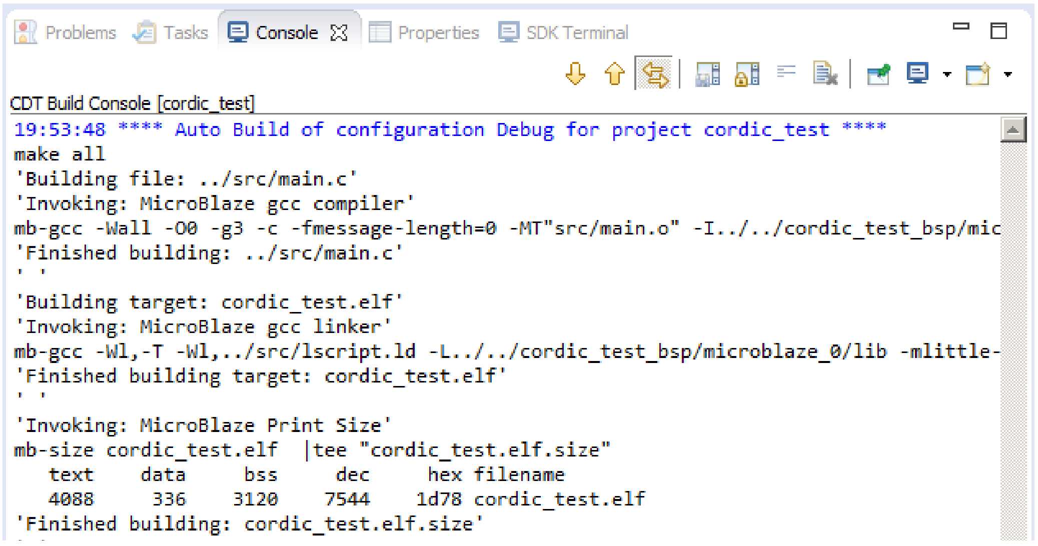
You can easily find the definitions and descriptions of the used function by selecting “Open declaration F3” in the context menu (right click the selected symbol, variable or function).



```
//Send data to data register of cordic processor
    CORDIC_IP_mWriteReg(CORDIC_BASE_ADDR, ANGLE_RE
//Start cordic proce
    CORDIC_IP_mWrite
    CORDIC_IP_mWrite
//Wait for ready bit
    while( (CORDIC_I
//Get results
```

6.6 Build the application

Select “Project” → “Build all” from the SDK window menu.



```
19:53:48 **** Auto Build of configuration Debug for project cordic_test ****
make all
'Building file: ../src/main.c'
'Invoking: MicroBlaze gcc compiler'
mb-gcc -Wall -O0 -g3 -c -fmessage-length=0 -MT"src/main.o" -I../../cordic_test_bsp/mic
'Finished building: ../src/main.c'
' '
'Building target: cordic_test.elf'
'Invoking: MicroBlaze gcc linker'
mb-gcc -Wl,-T -Wl,../src/lscript.ld -L../../cordic_test_bsp/microblaze_0/lib -mlittle-
'Finished building target: cordic_test.elf'
' '
'Invoking: MicroBlaze Print Size'
mb-size cordic_test.elf  |tee "cordic_test.elf.size"
   text    data     bss     dec     hex filename
   4088     336    3120    7544    1d78 cordic_test.elf
'Finished building: cordic_test.elf.size'
```

Ensure that the console is free of errors.

Now, you have **cordic_test.elf** file that contains the application image data. We will use that file to initialize Microblaze program memory before simulation is started.

7. Simulation of accelerated Microblaze system

7.1 Go to Vivado “cordic_test” project window

7.2 Create “mb_design_tb” testbench simulation file

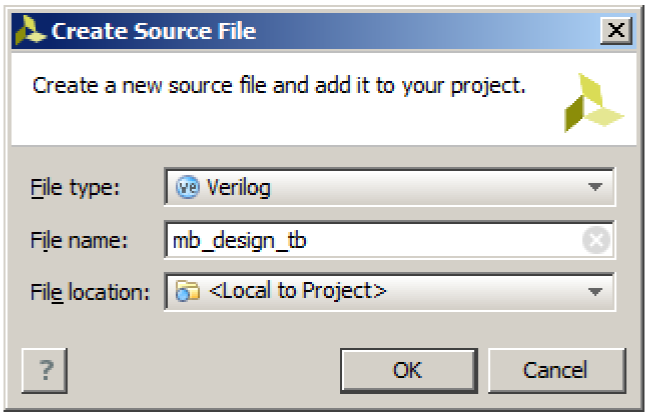
Click “Add sources” in the “Sources” window menu

And follow dialog windows sequence that will appear.

Select “Add or create simulation sources”

Click “Create File” button

Enter “mb_design_tb” file name



Create Source File

Create a new source file and add it to your project.

File type: Verilog

File name: mb_design_tb

File location: <Local to Project>

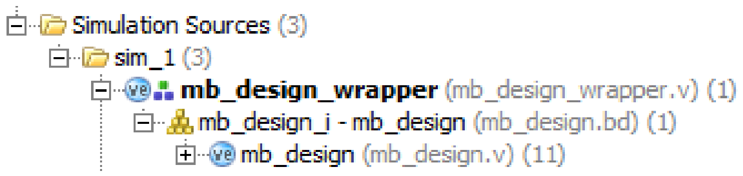
Click OK

7.3 Enter the simulation testbench code into “mb_design_tb.v”

Copy and paste the Verilog code given below into “mb_design_tb.v”

```
module mb_design_tb();

reg clk_n, clk_p;
wire [11:0] angle;
wire [11:0] sin;
wire [11:0] cos;
reg reset, reset_n;

real r_angle = 1024*3.14*0.2;
real r_sin, r_cos;
// Dip switches stimulus
assign angle = r_angle;
// Reset stimulus
initial
begin
   reset = 1'b1;
   reset_n = 1'b0;
#10 reset = 1'b0;
   reset_n = 1'b1;
end
// Clocks stimulus
initial
begin
   clk_n = 1'b0; //set clk to 0
   clk_p = 1'b1;
end
always
begin
#5 clk_n = ~clk_n; //toggle clk every 5 time units
   clk_p = ~clk_p; //toggle clk every 5 time units
end
//Put sin and cos as real values
always @*
begin
   r_sin = sin;
   r_cos = cos;
   r_sin = r_sin / 1024;
   r_cos = r_cos / 1024;
end
//Instantiate tested module
mb_design_wrapper mb_design_inst ( clk_n, clk_p, angle, cos, sin, reset, reset_n);

endmodule
```

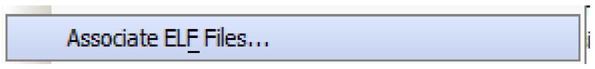
7.3 Verify your design hierarchy in Simulation Sources folder



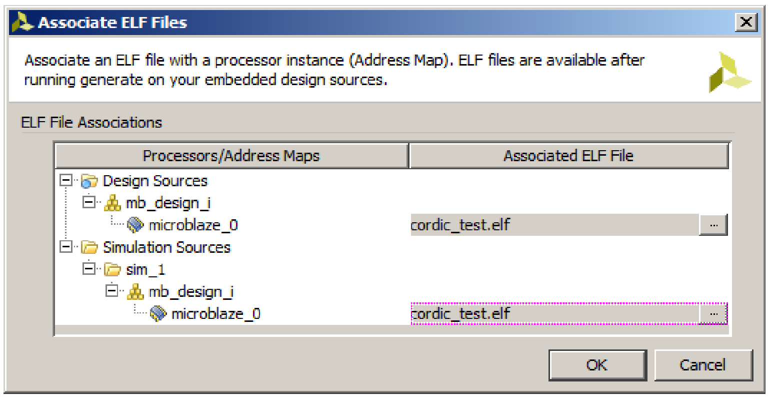
Simulation Sources (3)
sim_1 (3)
mb_design_wrapper (mb_design_wrapper.v) (1)
mb_design_i - mb_design (mb_design.bd) (1)
mb_design (mb_design.v) (11)

7.5 Associate “cordic_test.elf” file with the Microblaze program memory

Select “Tools”->”Associate ELF files” from the program menu



Select “cordic_test.elf” file from the SDK project menu:
“.\cordic_test\cordic_test.sdk\cordic_test\Debug\cordic_test.elf”


Associate ELF Files

Associate an ELF file with a processor instance (Address Map). ELF files are available after running generate on your embedded design sources.

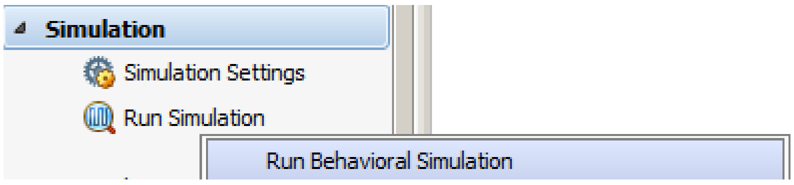
ELF File Associations

Processors/Address Maps	Associated ELF File
Design Sources	
mb_design_i	
microblaze_0	cordic_test.elf
Simulation Sources	
sim_1	
mb_design_i	
microblaze_0	cordic_test.elf

Click OK

7.6 Run simulation

Click “Run simulation”->”Run Behavioural Simulation” in the flow navigator



7.7 Verify results

Check if the GPIO binary data corresponds each other. Is the *sin* and *cos* value of the given *angle* correct?