AGH University of Science
and Technology
Cracow
Department of Electronics

Custom system design in FPGA laboratory

Tutorial 5

Implementation of the system with pipelined accelerator

Author: Paweł Russek

ver. 2018.03.15

1. Introduction

1.1. Objectives

In this tutorial, the design process of the custom architecture that incorporates the pipelined cordic accelerator will be presented. The custom processing system will be implemented in Zynq Programmable System-on-Chip. We will use Zynq's ARM subsystem to run an a software application that will off-load sinus and cosinus calculations to the cordic accelerator. The cordic pipelined accelerator will be implemented in Zynq's FPGA subsystem. For a pipeline accelerator to make sense, it is necessary that the application takes advantage of the data set processing were many arguments simulanously are input for the accelerated function.

We will show the simple solution that allow us to connect a pipelined accelerator to the CPU using the FIFO buffer and AXI -Stream interface. The FIFO is a convenient component that enable the smooth data processing in the pipelined accelerators. While, AXI-Stream is a steaming bus standard within ARM's AMBA specification that intended for serial communication peripherals (e.g. Ethernet), but it is also useful for accelerated computing. The AXI-Stream bus standard will be explained in this tutorial.

We will use Xilinx's Vivado Design Suite to create hardware and software part of the design. The project will be run on Zedboard that is popular Zynq-based development platform.

1.2. Prerequisites

As the understanding of the architecture of the pipelined cordic accelerator is essential to continue this content. It is necessary for the student to finish “The pipelined cordic accelerator” tutorial of this course first.

The prior experience in designing with Vivado Desig Suite will be advantage while completing this exercise.

The basic understanding of Verilog HDL and C programming languages will be useful to complete this tutorial.

To complete this instruction, the student will also need the accompanying archive file that contains the library files of the cordic pipelined accelerator and the software application sources.

2. AXI-Stream bus

AXI is part of ARM Advanced Microcontroller Bus Architecture (AMBA). AMBA 4.0, released in 2010, includes the second version of AXI. There are three types of AXI4 interfaces: AXI, AXI-Light, and AXI-Stream. AXI-Stream is intended for high speed streaming data. The AXI4-Stream bus removes the requirement for an address.

For the discussion in this tutorial, it is sufficient to consider AXI-Stream as a five-signal interface that consist of: *clk*, *tdata*[*w-1:0*], *tvalid*, *tready*, and *tlast* signals. The *tdata* is a *w*-line bus signal, and the rest are single line signals. The direction of the signals depends on the type of communication peer. We have master and slave devices. The figure and table below summarizes the AXI-Stream signals based on the device type.

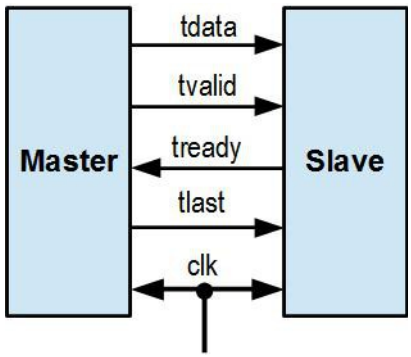


Figure 1

Master		Slave	
Signal	**Direction**	**Signal**	**Direction**
clk	in	clk	in
tdata[w-1:0]	out	tdata[w-1:0]	in
tvalid	out	tvalid	in
tready	in	tready	Out
tlast	Out	Tlast	in

Function of the signals is as follows:

- **clk** – external synchronisation signal. AXI-Stream transactions are done on rising clk edge.
- **tdata**[w-1:0] – binary data transmitted from the master to slave device. Width of the data can be 32, 64, 128, 256, or 512 bits.
- **tvalid** – indicates valid data on tdata. Active high.
- **tready** – indicates that slave is ready to receive data. Active high.

It is worth to note that tdata is transmitted from master to slave when both tvalid and tready are active. After data transmission, master should withdraw tvalid or issue a new data word.

- **tlast** – indicates that end of the data bulk. The signal not obligatory, but it is useful in some hardware configurations to mark the end of data packet/frame. The Ethernet peripheral can know to start issuing the network packet for example.

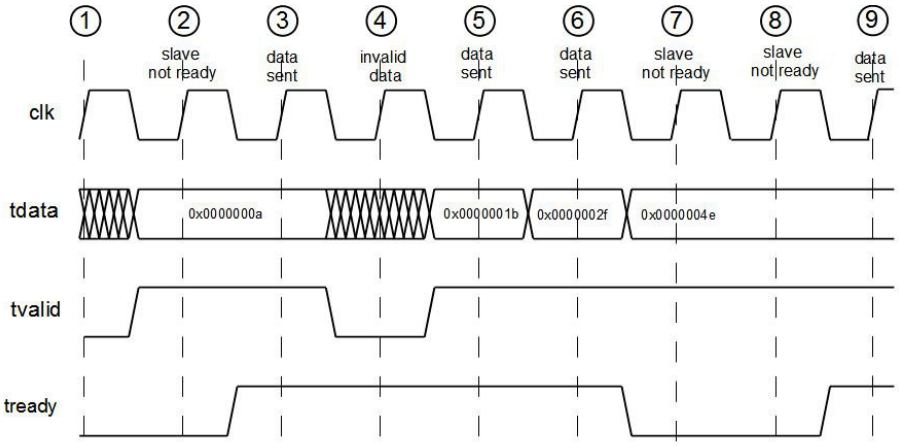


Figure 2

Figure 2 presents an example timing diagram for the AXI-Stream transmission. Master sends 32-bit data to the slave. Words 0x0000000a (3), 0x0000001b (5), 0x0000002f (6), and 0x0000004e are sent (9). Words 0x0000000a and 0x0000004e wait for the active tready (2, 7, 8) . Words 0x0000001b and 0x0000002f are transmitted in one clock cycle (5, 6), although word 0x0000001b is not available immediately (4).

As AXI-Stream does not have an address signal it cannot be memory-mapped. Consequently, AXI-Stream peripherals requires a bridge element to be connected to ARM. Usually, one uses AXI-Stream to AXI or AXI-Light bridges.

Maximum throughput of AXI-Stream i.e. one data transmission per clock cycle can only be achieved if no data waiting states exist. Transmission gaps can exist if CPU ir DMA controller cannot keep pace with AXI-Stream transmission clock. On the other hand, memory data movement, like memory to peripheral coping, is more efficient if performed in data series. Thus, the FIFO (First-In First-Out) buffer is usually use to smooth AXI read/write transaction for seamless AXI-Stream operation. The scenario is depicted in Figure 3.

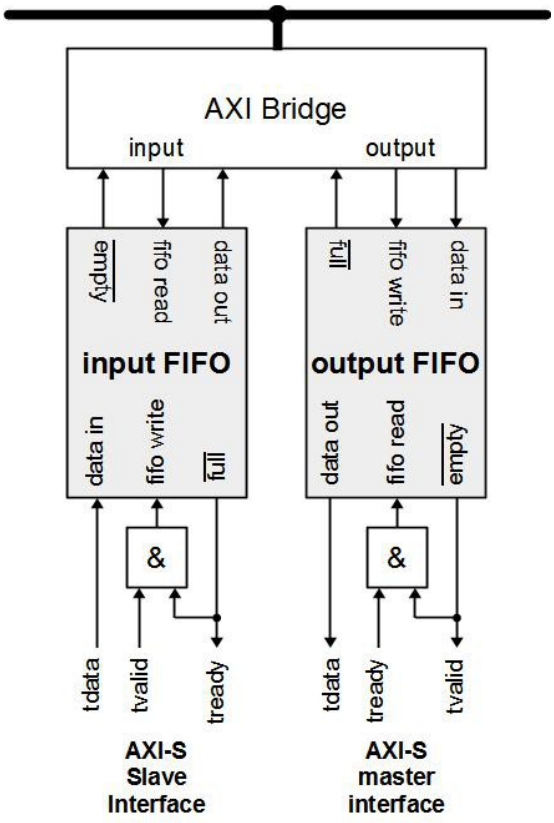


Figure 3

Our FIFOs has two ports: one for data input and one for data output. The FIFO's input port is used to place data in the buffer, and the FIFO output port is use to take data out of buffer. The data elements are organised in the FIFO in such an order that data element that was first put into the buffer is also first to be get. The number of data elements that can be stored in the FIFO is called the FIFO length. The FIFO input port has three signals: *data_in*, *fifo write*, and *full* signal. The FIFO output port has three signals: *data_out*, *fifo read*, and *empty* signal. If there is no data in the FIFO, the *empty* signal is activated. Similarly, If no free space exist in the

FIFO, the *full* signal is activated. The *fifo read* and *fifo write* signals are data read/write strobes respectively.

As it can be seen in Figure 3, the FIFO interface can be easily transformed to AXI-Stream interface. We have two FIFOs in the figure: one for input data stream and one for output data stream. We created the AXI-Stream Master interface from the output FIFO and the AXI-Stream Slave out of the input FIFO. The AND gates provide correct strobes for the *fifo read* and *fifo write* signals.

Mind, that clock signal that is essential for the system to work is not presented in Figure 3.

3. AXI-Stream loopback with accelerator

AXI-Stream allows designer for simple connection of the pipelined accelerator to the microprocessor system. In the scenario, the accelerator reads the stream of input data from AXI-Stream Master *tdata* and sends results to the AXI-Stream Slave *tdata*. The connection of the accelerator to AXI-Stream channel is mostly simplified if the iteration interval (II) of the accelerator is one clock cycle (*i.e.* it accepts the input data element every clock cycle). The solution is presented in Figure 4.

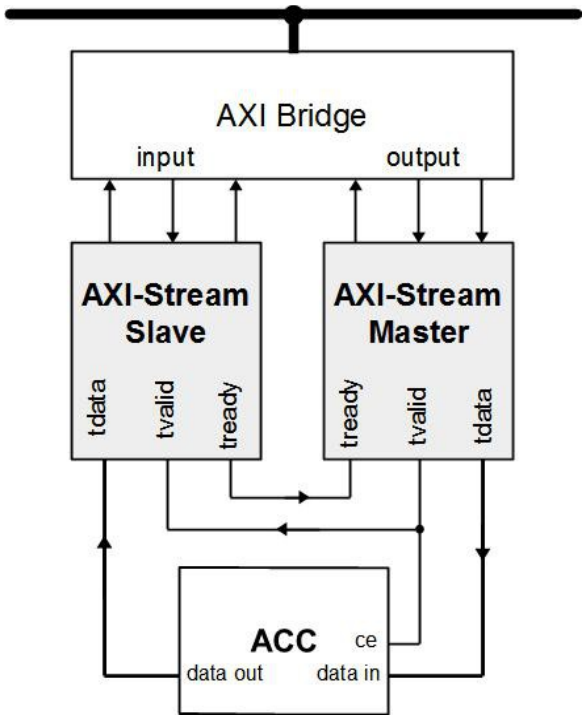


Figure 4

The handshake signals (t*data*, *tvalid*) of AXI-Stream Master can be directly connected to respective signals of AXI-Stream Slave. The pipelined accelerator has only *data in,data out* and *ce* signals. The *ce* marks valid *data in,* so it is connected to Master's *tvalid.* As II =1, the accelerator produces results at each clock cycle, if only *ce* is active. Such Master's *tvalid* can also be directly connected to slave's *tvalid.*

Processing has to be stalled if slave's FIFO is full, so slave's *tready* is connected to master's *tready.*

We have to remember that a pipelined accelerator always introduces some latency (pipeline delay) to the data loop. Consequently, data exists in the accelerator registers when processing stops. Therefore, some dummy data has to follow the valid data set to push the significant results out of the accelerator to the AXI-Stream read channel. The number of the dummy elements has to equal to the latency value.

4. Hardware design

In this section we, will create the hardware for the processor system with the pipelined cordic accelerator.

We will create system that is presented in Figure 5.

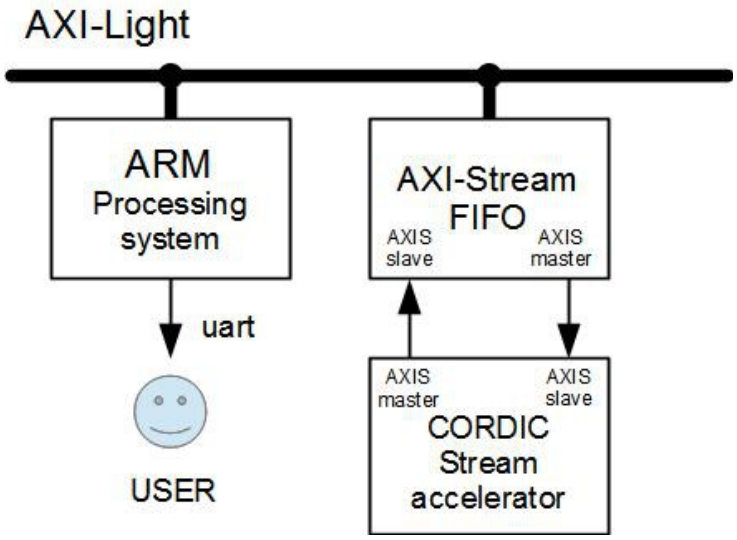


Figure 5

We will use the AXI-Stream FIFO component to connect the cordic accelerator to the ARM processor sub-system. AXI-Stream works as the AXI-Light to AXI-Stream bridge with the two FIFOs, one FIFO for data sending and one for receiving. The ARM runs an application that interacts with the user using an UART text terminal.

4.1. Preparing IP repository

We will use already created IP of the cordic accelerator that is delivered in a zip file that is a part of the laboratory materials. We must prepare the IP files on the local disk drive.

1. Locate and uncompress *tut5_src.zip* file.

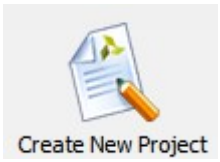
   Find the *ip_repo* directory and *cordic_stream_acc_1.0* subdirectory. You can copy *cordic_stream_acc_1.0* to your own IP repository, if you already have one.
   You should have *ip_repo* directory as a product of the completion of the previous tutorials. However, you can also keep *cordic_stream_acc_1.0* where it has been uncompressed but note its location.

4.2. Creating the new Vivado project

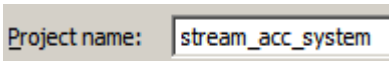
We will design our accelerated processor system as a new Vivado project.

1. Run Vivado hardware design environment

2. Create the new project “stream_acc_system.xpr.” in the Vivado welcome window.

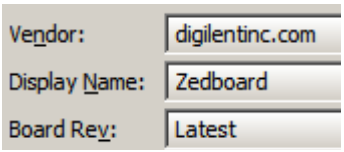


Select “Create New Project” from the “Quick Start” menu.



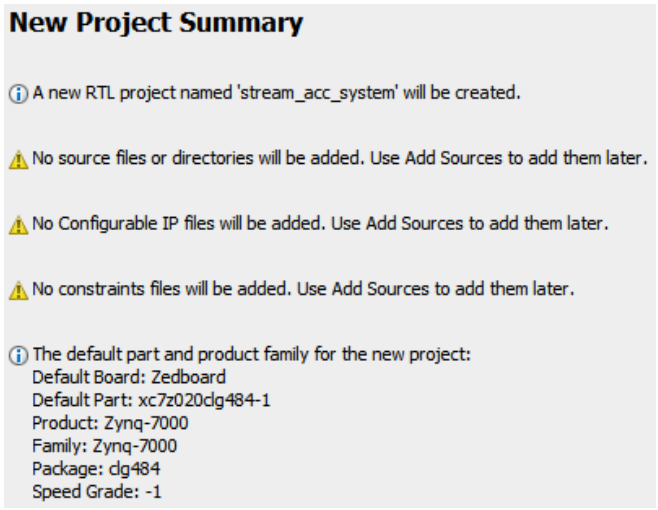
Enter the “stream_acc_system.xpr.” project name in the “New Project” dialog.

3. Click the “Next” button till the“Default Part” selection appears, and select “Zedboard” from the boards list.



Click “Next” button.

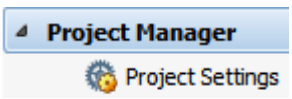
4. Verify your project settings.



Click “Finish” button if it is correct.

4.3. Adding the cordic accelerator to the project's IP repository

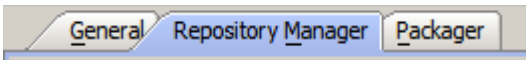
1. Open “Project Settings” in “Project Manager”.



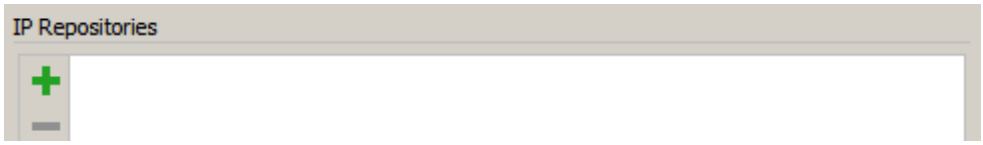
2. Select “Verilog” HDL as default target language

3. Click “IP” icon and switch to the “IP” window.

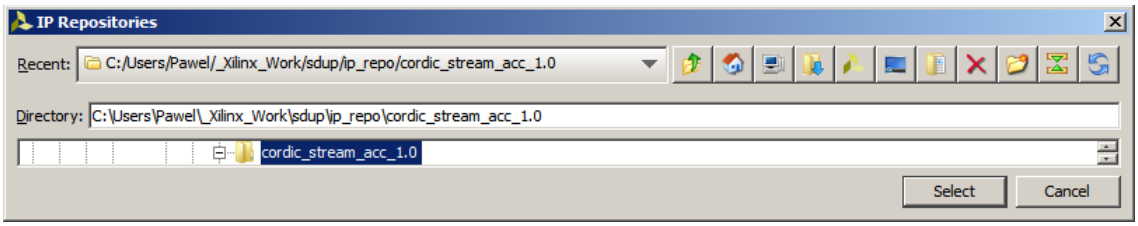
4. Click and select the “Repository Manager” tab.



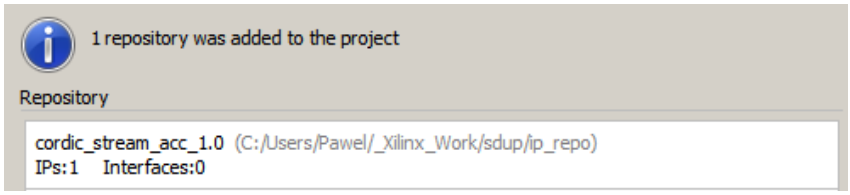
5.Press the green “+” button to start adding a new IP to the repository.



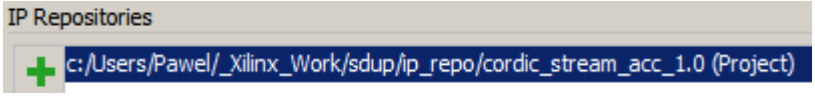
6. Select the location of “*cordic_stream_acc_1.0”* directory in the “IP Repositories” dialog.



7. Click “OK” button in the “IP Repository” dialog to confirm the new repository added to the project.

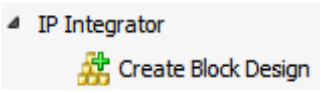


8. Click “OK” button to close the “Project Settings” dialog.

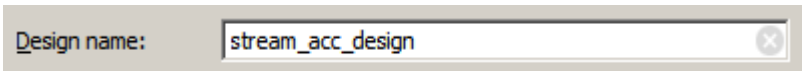


4.4. Create a block diagram of the system

1. Click the“Create Block Design” icon in the “Flow navigator” window.



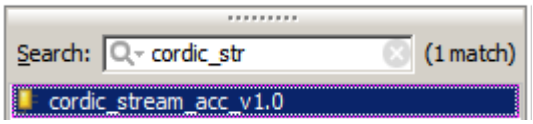
2. Write the “*stream_acc_design*” name in the “Create Block Design” dialog.



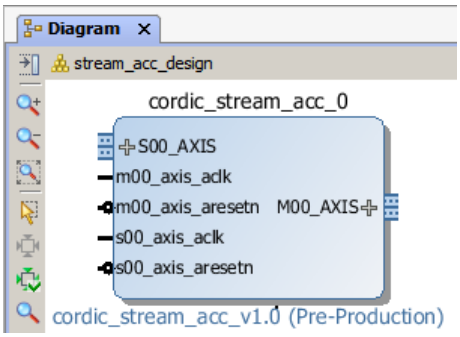
3. Insert “cordic_stream_acc” element to the block design.

Select “Add IP” in the “Diagram” menu

Select “cordic_stream_acc” in the dialog window



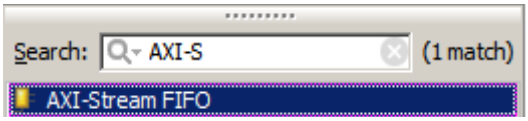
The “cordic_stream_acc” should appear in the diagram.



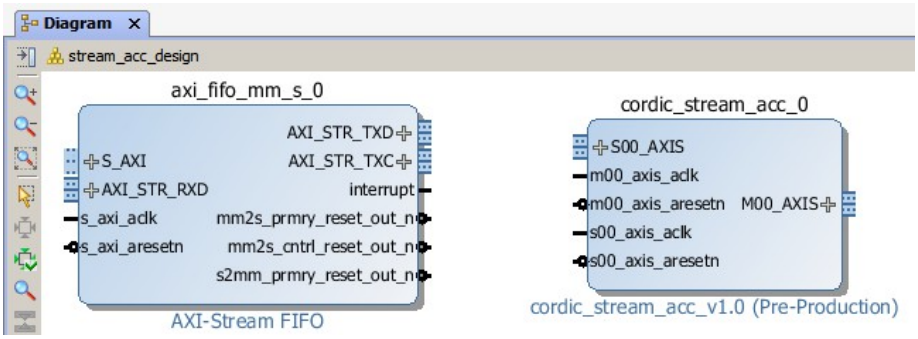
4. Insert “AXI-Stream FIFO” element to the design.

Select “Add IP” in the “Diagram” menu

Select “AXI-Stream FIFO” in the dialog window



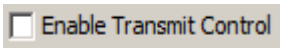
The “AXI-Stream FIFO ” should appear in the diagram.



5. Configure “AXI-Stream FIFO” IP

Double click the “AXI-Stream FIFO” element. The configuration window should appear.

Uncheck “Enable Transmit Control”.



Select “4096” in “Transmit Fifo depth”.

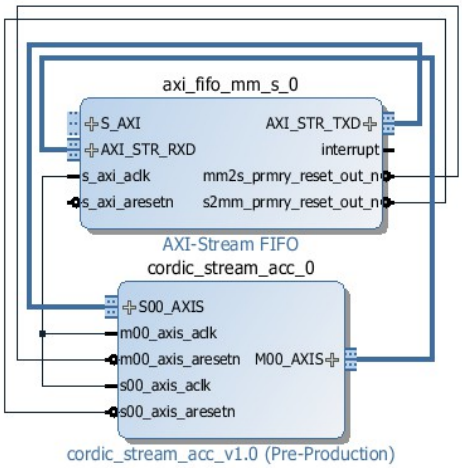
Select “4096” in “Received Fifo depth”.

Check TSTRB to add the tstrb port. It is not used in this project but we will avoid warnings if TSTRB is enabled

Click “OK”button.

6. Connect “AXI-Stream FIFO” to “cordic_stream_acc” according to the table and figure below.

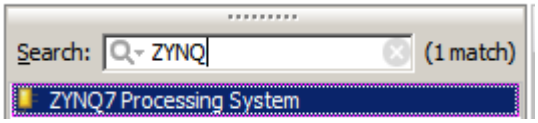
FROM: **AXI-Stream FIFO**		TO: **cordic_stream_acc**	
Port name	**Description**	**Port name**	**Description**
AXI_STR_RXD	AXI-Stream Receive AXI-Steam slave	M00_AXIS	AXI-Steam master
AXI_STR_TXD	AXI-Stream Transmit AXI-Steam master	S00_AXIS	AXI-Steam slave
s_axi_aclk	Clock signal	m00_axis_aclk	master clock
		s00_axis_aclk	slave clock
m2mm_prmry_reset_out_n	To transmit AXI-S reset out	m00_axis_areset	master AXI-S reset in
s2mm_prmry_reset_out_n	To receive AXI-S reset out	s00_axis_areset	slave AXI-S reset in



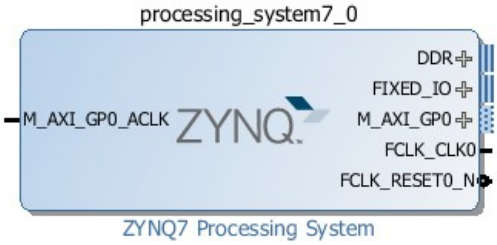
7. Insert “ZYNQ Processing System” element to the design.

Select “Add IP” in the “Diagram” menu

Select “AXI-Stream FIFO” in the dialog window

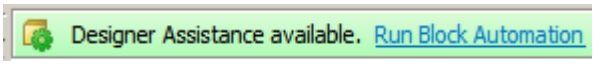


The “ ZYNQ Processing System ” should appear in the diagram.

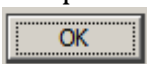


8. Use the “Designer assistance” menu to complete the design

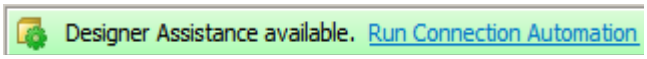
Click “Run Block Automation”



Accept default settings Click OK in the dialog

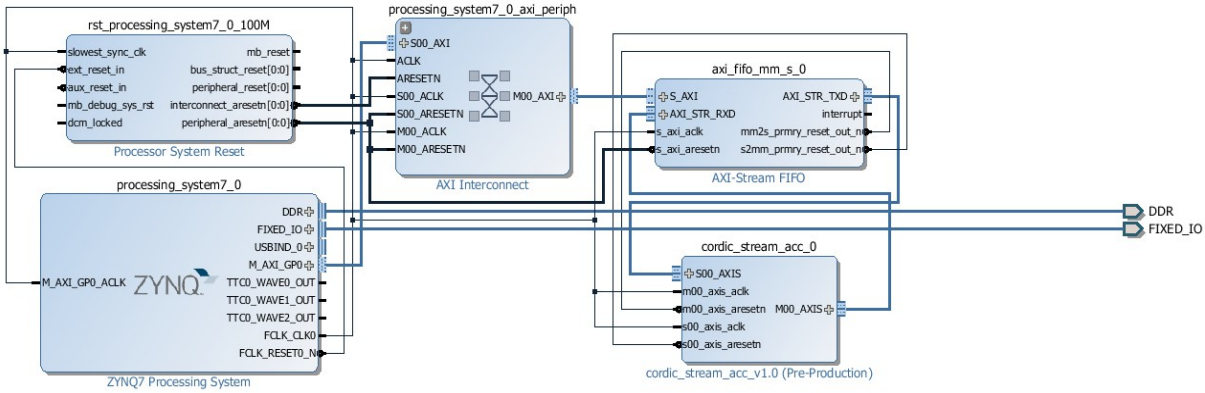


Click “Run Connection Automation”



Accept default settings Click OK in the dialog

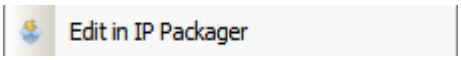
Block diagram is ready



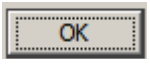
4.5. Editing cordic stream accelerator source code

1. Open the “cordic_stream_acc_0” source code in the separate Vivado window

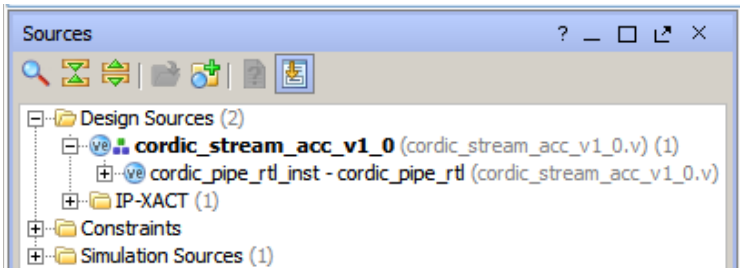
Right-click the “cordic_stream_acc_0” element and select “Edit in IP packager” in the context menu



Click the “OK” button



A separate Vivado project that contains the cordic pipe accelerator and AXI-Stream wrapper appears



The “cordic_pipe_rtl” component is a Verilog module You know from the “Pipelined cordic processor” tutorial. The “cordic_stream_acc_v1” module is a wrapper that provides AXI-Stream signals for master and slave interfaces:

```
module cordic_stream_acc_v1_0
(
		// Ports of Axi Slave Bus Interface S00_AXIS
		input wire  s00_axis_aclk,
		input wire  s00_axis_aresetn,
		output wire  s00_axis_tready,
		input wire [C_S00_AXIS_TDATA_WIDTH-1 : 0] s00_axis_tdata,
		input wire [(C_S00_AXIS_TDATA_WIDTH/8)-1 : 0] s00_axis_tstrb,
		input wire  s00_axis_tlast,
		input wire  s00_axis_tvalid,

		// Ports of Axi Master Bus Interface M00_AXIS
		input wire  m00_axis_aclk,
		input wire  m00_axis_aresetn,
		output wire  m00_axis_tvalid,
		output wire [C_M00_AXIS_TDATA_WIDTH-1 : 0] m00_axis_tdata,
		output wire [(C_M00_AXIS_TDATA_WIDTH/8)-1 : 0] m00_axis_tstrb,
		output wire  m00_axis_tlast,
		input wire  m00_axis_tready
);
```

2. Find instantiation of the “cordic_pipe_rtl” module in the “cordic_stream_acc_v1.v” code.

```
//Instantiate cordic pipelined module here
    cordic_pipe_rtl cordic_pipe_rtl_inst(    s00_axis_aclk,          //clock
                                             rst,                    //reset
                                             s00_axis_tvalid,        //ce
                                             <TBD1>,                  //angle_in
                                             <TBD2>,                  //sin_out,
                                             <TBD3>,                  //cos_out,
                                             m00_axis_tvalid         //valid_out
                                         );
```

3. Complete the port names where the <TBD> note exist in the code

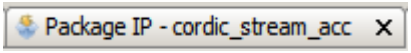
Find a signal name of the *tdata* in the slave AXIS port.
<TBD1>: Connect slave's *tdata* to the *angle_in* port of the “cordic_pipe_rtl”

Find a signal name of the *tdata* in the master AXIS port.
<TBD2>: Connect bits [15:0] of master's *tdata* to the *sin_out* port of the “cordic_pipe_rtl”
<TBD3>: Connect bits [31:16] of master's *tdata* to the *cos_out* port of the “cordic_pipe_rtl”

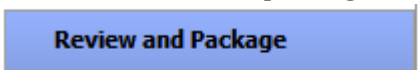
4. Save the the changes and close the project

Select “Save File” in the “File” menu

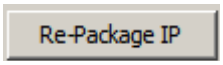
Skip to the “Package IP” tab



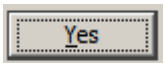
Select “Review and package” in the window menu



Click the “Re-Package IP” button



Click “Yes” to close the project

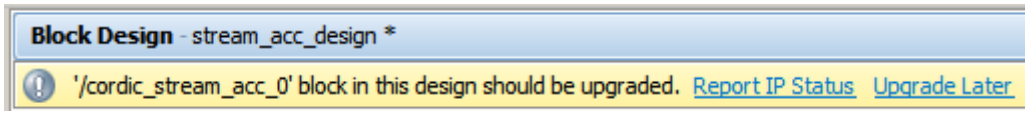


4.6. Updating IP library in the Vivado “stream_acc_system” project

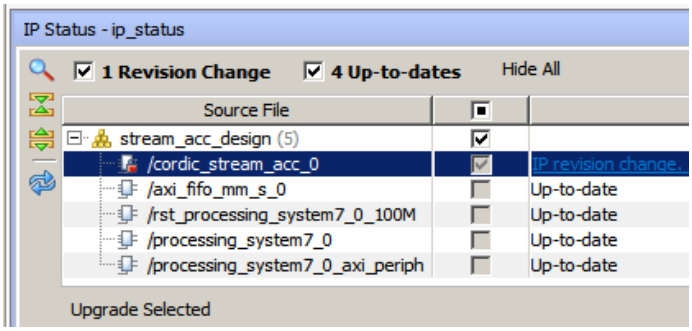
The “cordic_stream_acc” element has been modified in previous step, so we have to update the repository in the “stream_acc_system” project. Vivado itself notifies that the repository has changes and asks for its update.

1. Refresh the “cordic_stream_acc” library element in the project.

   Click “report IP status” in the warning that appears in the “Block Design” window.

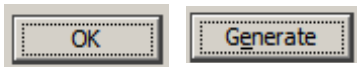


   Now, You can see in the “IP Status” window at the bottom of the screen that the “cordic_stream_acc” has changed.

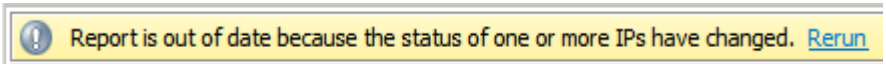


   Click the “Upgrade Selected” button.

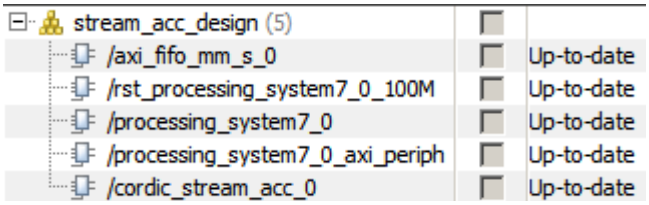
   Click “OK” in the “Upgrade IP” dialog and “Generate” in the “Generate Output products” dialog for the default settings.



2. Click “Rerun” to refresh the IP report and check the “cordic_stream_acc” status again.



   All IPs should be up-to-date



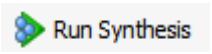
4.7. Generate the bitstream file, export hardware and launch Vivado SDK

1. Create HDL wrapper for the “stream_acc_system” block diagram.

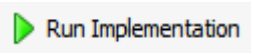
   Right-click the block diagram name and select “Create HDL wrapper” from the context menu.

2. Run hardware synthesis, implementation, and bitstream generation.

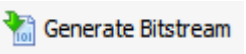
   Click “Run synthesis” in “Flow navigator”



   Click “Run Implementation” in “Flow navigator”

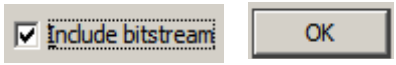


   Click “Generate bitstream” in “Flow navigator”



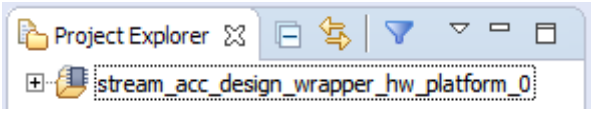
   Select “File->Export->Export Hardware” form the Vivado menu

   Check “Include Bitstream” in the “Export hardware” dialog and click “OK”



   Select “File->Launch SDK” form the Vivado menu

   Vivado SDK has been launched and its window has appeared.
   The hardware platform support files should be visible in “Project Explorer”



4.8. Preparing the “find_max” application in Vivado SDK

In this section we will use our accelerated system to run application that finds a maximum of the function. We will use an exhaustive search for the function minimum. I requires to calculate the function value for a given argument range. As the same sinus and cosinus calculations have to be performed for a many element data set, the pipelined cordic accelerator comes to be handy.

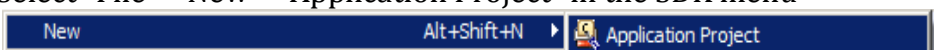
The function

$$f(x) = sin(x) + cos(x)$$

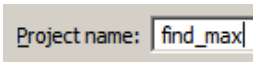
will be examined.

1. Create a new application project in Vivado SDK

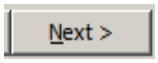
   Select “File → New → Application Project” in the SDK menu

   Enter “find_max” in the “Project name” text field

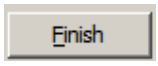
   Click “Next” to skip to next the “Templates” tab

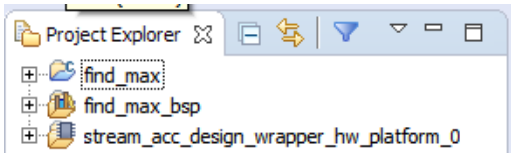
   Select “Empty Application”

   Click “Finish”

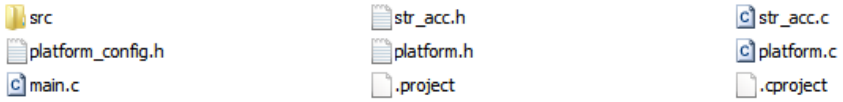
   The “find_max” application and BSP appear in “Project Explorer”

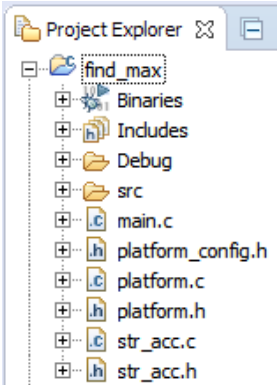
2. Copy the application source files included in “tut5_src.zip” the to the find_max application directory.

   Locate “src” folder in the uncompressed “tut5_src.zip” directory

   Copy files from the “src” folder to the “find_max” SDK application folder:
   “*~\stream_acc_system\stream_acc_system.sdk\find_max*”

3. Select “File → Refresh” to update the application view in “Project Explorer”

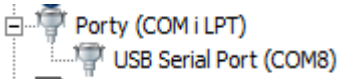


4.9. Connect Zedboard to the PC and power supply

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

2. Find your USB serial port number in Windows Device Manager

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



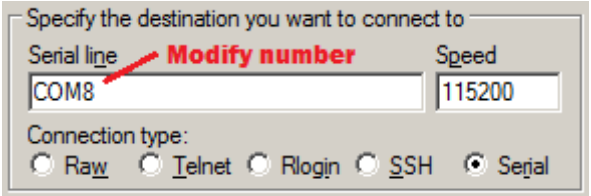
   Along with the picture, we will use COM8 in this tutorial.

3. Open PuTTY application for serial communication with Zedboard

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

4. Configure PuTTY connection and open terminal window

   Select “Serial” interface, COM name, and 115200 baud rate.

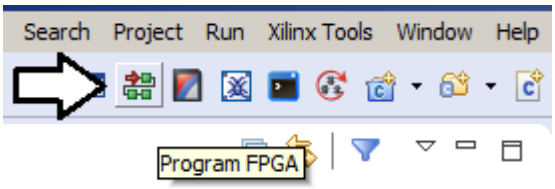


Click OK
Your terminal window is now opened

4.10. Compile and run application

1. Download bitstream to the FPGA

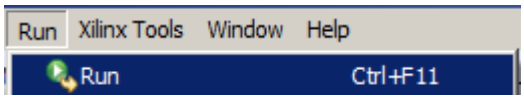
Go to Vivado SDK and select the "Program FPGA" button in main menu.



Wait for configuration process to complete

2. Run application

Select "Run → Run" from the main menu



3. Check the terminal for the output.

5. Exercise

Modify the find_max code and create application that find both the minimum and the maximum values in range [– π/2, – π/2].

Hint

The debugger tool that is embedded in the Vivado SDK tool will help to deploy your design.