

# Vivado + Vivado HLS Tutorial

ECEN 625 – Winter 2019

## 1 Setup

You can install Vivado on your local machine (<https://www.xilinx.com/support/download.html>). If you do this, I suggest you install Vivado 2018.2 Design or System Edition on an Ubuntu 16.04 machine. If you prefer, you can install Vivado on a Windows machine. I haven't tested this.

Vivado HLS requires a license. You can access the department Xilinx license server by setting the following environment variable. This means you must either be on the university network, or connected to the CAEDM VPN.

```
export LM_LICENSE_FILE=2100@ece-xilinx.byu.edu
```

### 1.1 Installing Boards

If you are using a Digilent board, such as the Zedboard, you need to setup the board files in Vivado. See <https://github.com/Digilent/vivado-boards/>.

### 1.2 Running Vivado

Before you can run the Vivado tools you should first run the configuration script:

```
source /opt/Xilinx/Vivado/2018.2/settings64.sh
```

This will add the tools to your PATH. If you don't want to setup your own machine, contact me and I can give you access to a server with Vivado 2018.2 installed.

To run Vivado, simply type

```
vivado
```

## 2 Hello World (Hardware)

### 2.1 Creating the Project

After launching Vivado, follow these steps to create a hardware project:

1. Create new project..., and choose a project name and location. Next. Choose an RTL project. Next.
2. On the next screen, click *Boards* at the top, and choose your board (ie. Zedboard). Click Finish.

### 2.2 Creating a Base Design

In these steps we will create a basic system, containing only the Zynq processing system (PS).

1. Click *Create Block Design*, and click *OK* on the popup.
2. Add the *ZYNQ7 Processing System* IP to the design (right-click, Add IP).
3. A green banner should appear with a link to *Run Block Automation*. Run this. This will configure the ZYNQ for your board.
4. The *FCLK\_CLK0* output of the PS will serve as your system clock. It is set to 100MHz by default. Connect it to the *M\_AXI\_GP0\_ACLK* input.
5. Generate a top-level module: In the *Sources* window, right-click on your block design (*design\_1.bd*) and select *Create HDL Wrapper*. Use the option to *Let Vivado manager wrapper and auto-update*.

### 2.3 Synthesizing the hardware

1. Run *Generate Bitstream*.

2. Once the bitstream generation is complete, export the hardware. File→Export→Export Hardware. Check *Include Bitstream*, and choose a location to store the Hardware Description File (.hdf).



### 3 Hello World (Software)

Run the Xilinx SDK (`xsdk`), and choose a workspace location.

#### 3.1 Create SDK Projects


1. Create the hardware project. New Project→Xilinx→Hardware Platform Specification. Chose a project name, and indicate the location of your exported .hdf file.
2. Create the board support package project. This will includes all of the system support software (drivers, etc.). New Project→Xilinx→Board Support Package. Indicate the hardware platform project you created in the last step, and choose the *standalone* OS.
3. In the *Board Support Package Settings* popup, go to the *standalone* menu, and change *stdout* to use *coresight\_comp\_0*. This ensures that when your application prints to stdout, it will be sent over the virtual console over the JTAG, and not on the physical UART pins present on the board.
4. Create your application project. New Project→Xilinx→Application Project. Choose an application name (ie. HelloWorld), and configure the project to use the Board Support Package project you created in the last step.
5. Open and inspect the code found in `src/helloworld.c`.

#### 3.2 Run the Application on the Board

1. Configure the FPGA with the  menu button.
2. Right-click on your application project, choose Run As→Launch on Hardware (System Debugger).
3. To view the program output, you will need to use the console selector button  in the *Console* window to select the *TCF Debug Virtual Terminal - ARM Cortex-A9 MPCore #0* console.
4. You should see the message *Hello World*.

### 4 Exporting your HLS as IP

This section discusses how you can export your IP from Vivado HLS to be used in a Vivado project. Since our goal is to communicate with the HLS IP from software, we will add a Slave AXI connection to our HLS IP core so that it can be connected to the ARM AXI bus.

1. Run Vivado HLS and open your project from the last assignment.
2. In Vivado HLS, add a directive to your top-level hardware function. Choose the *INTERFACE* directive type, and change the mode to an AXI4-Lite Slave (*s\_axilite*).
3. Run C Synthesis.
4. Click the Export RTL button , and make sure the Format Selection is set to *IP Catalog*.
5. Close Vivado HLS.

### 5 Adding your IP to Vivado

1. Launch Vivado, open your existing project, and open the block design.
2. If you do not already have a *Processor System Reset* IP, add one to your design. This will use the reset signal output by the processing system to reset IP in the FPGA fabric.
  - (a) Connect the system clock (*FCLK\_CLK0* from PS) to the *slowest\_sync\_clk* input.
  - (b) Connect the processor reset output (*FCLK\_RESET0\_N*) to the *ext\_reset\_in* input.
3. If you do not already have a *AXI Interconnect* IP, add one to your design. This is the bus that will allow the ARM CPU to communicate with the IP implemented in the FPGA fabric.

- (a) Configure the bus to have 1 Slave Interface and 1 Master Interface.
  - (b) Connect the PS bus master (*M\_AXI\_GP0* from PS) to the *S00\_AXI* slave port.
  - (c) Connect your clock (*FCLK\_CLK0* from PS) to all the clock inputs (*\*ACLK*)
  - (d) Connect your interconnect reset (*interconnect\_aresetn* from *Processor System Reset*) to the *ARESETN* input.
  - (e) Connect your peripheral reset (*peripheral\_aresetn* from *Processor System Reset*) to the other reset inputs (*\*ARESETN*)
4. Add your HLS IP:
  - (a) Open the IP catalog
  - (b) Right-click, *Add Repository*
  - (c) Navigate to your HLS IP found in *your\_hls\_project/your\_solution/impl/ip*.
  - (d) Go back to your block design and add the HLS IP to your design.
5. Connect up your HLS IP:
  - (a) Connect the clock (*FCLK\_CLK0* from PS) to the clock input (*ap\_clk*)
  - (b) Connect the reset (*peripheral\_aresetn* from *Processor System Reset*) to the reset input (*ap\_rst\_n*)
  - (c) Connect the bus (*M00\_AXI* from the *AXI Interconnect*) to the bus slave port (*s\_axi\_AXILiteS*)
6. Assign an address to your HLS IP. Open the *Address Editor*, find your IP, right-click *Assign Addresses*.
7. Run *Generate Bitstream*.
8. Export the hardware with bitstream, and be sure to choose the same location as last time, overwriting the existing hardware description file.
9. Close Vivado

## 6 Communicating with your HLS IP from Software

1. Launch Xilinx SDK and reopen your existing workspace.
2. You should see a prompt indicating that the hardware has changed. Click *Yes* to update your software libraries to support the new hardware system.
3. Right-click on your BSP project and click *Re-generate BSP sources*.
4. Right-click on your BSP project, click *Board Support Package Settings*, go to drivers and make sure you can find your IP core listed. For the in-class demo, my IP core was called *sum\_array*, so I can find the IP named *sum\_array\_0* in the list and verify that the driver called *sum\_array* is selected. Click OK to close the settings window.
5. In your BSP project, you can look in *ps7\_cortexa9-0/include* to find the header files for your driver. Mine are called *xsum\_array.h* and *xsum\_array\_hw.h*.
6. Include the necessary header file in your application code and write software to test that you can start your IP, wait for it to complete, and retrieve the return value. As with most Xilinx drivers, you will need to call the initialization function first (*XSum\_array\_Initialize*). The first argument is a struct variable that you should define and pass in by pointer, the second is the device ID (*XPAR\_XSUM\_ARRAY\_0\_DEVICE\_ID*), that can be used by including *xparameters.h*.