Getting Started in ECE 544 Building your first Embedded System

Vivado/Nexys 4 DDR - Revision 3.0 (Last updated: 19-Dec-2016 for Vivado 2016.2)

By
Chetan Bornarkar
bchetan@pdx.edu

Disclaimer:

I have tried to be as accurate as possible and have taken into account the mistakes I did when I was building my first embedded system in my course work. I have always felt that learning by mistakes is a good way to learn. So don't worry if you can't get it to work in the first try, make mistakes and learn. If you are not able to get it to work by yourself, always go to the d2l discussion page or the Xilinx discussion forums. If you find any problems please communicate within yourself to solve it as much as possible. This document is mainly based on the previous versions provided by Prof. Roy Kravitz.

Table of Contents

Revision History

Revision	By	Date	Description
1.0	RK	31-Dec-14	First Release. Created from the ISE/Nexys4 Getting Started Guide
2.0	RK	30-Dec-15	Modified for Vivado 2015.4 and the Nexys4 DDR board
3.0	SY	24-SEP-16	Modified for the Colorwheel implementation project for ECE 544, PSU

A brief word from the author

Getting the initial start by building your first embedded system is a hard work in itself. You will be introduced to a lot of tools which you might have never used or never even heard off. But do not worry. It is not an impossible task to complete. It is a very easy and simple work, once you know what and how to do it. So start off with a clear, peaceful mind and go forth. There is a lot of documentation available for the usage of the tools and the common errors you might face. I hope this document will be a good start off for your course, as you will be building a lot of embedded systems, debugging them and making them work by the end of this course. Fear not, go ahead and "May the force be with you".

Using this guide

The purpose of this guide/tutorial is to help you create your first Microblaze-based embedded project – a system and application that performs pulse-width modulation, detection and implements a color wheel. This document does not replace the Xilinx documentation, nor is it intended to be an exhaustive How-To, but my hope is that this document will make your first excursion into working with embedded systems in the Xilinx context a bit less mysterious and a whole lot less overwhelming.

I have tried to provide screen-by-screen and step-by-step instructions to guide you through the process of creating a target hardware platform and a software application. While you may be tempted to blindly follow the steps, I encourage and implore you **not to**. You will be using SDK and Vivado throughout the entire term and you will be creating your own custom peripherals, target hardware and software applications – you need to understand how to make these complex and sometimes quirky tools do your bidding. Blindly following the steps in this guide without attempting to understand the process and what the tools are doing for you is a recipe for problems later in the term. You don't want to be learning how to use the tools two nights before a project is due. Take the time to explore the options available to you as you work through this guide. Take the time to read the documentation (and you will be reading lots of it) and watch the quick-take videos. Take the initiative to work through the process on a system of your own creation. Using this guide will get you going but it will not really help you climb the learning curve towards mastering the tools. That's up to you.

A FINAL NOTE. When you get stuck (and I can almost guarantee that you will at some point during the term) we will try to help, but we are not omniscient. We have to do the same things you should to be doing – examine the log files, try to make sense out of obscure error messages, search Google, search the Xilinx knowledge base and search the Xilinx User forums. We simply cannot do that for a class full of students so try hard to figure out the problem by yourself before giving up. If you do give up, post your problem in the discussion forums on D2L – it's possible somebody else encountered and overcame a similar problem. These tools work pretty well most of the time so as much as you'd like to blame the tools or cast dispersion on Xilinx or blame the instructor or the documentation – look closely at your work…like it or not, the problem is probably there. 'Nuf said.

The Xilinx embedded tool chain

The Xilinx embedded tool chain consists of two major GUI-oriented applications. Xilinx provides additional components that are not part of the process for creating Xilinx FPGA-based embedded system but are integrated into Vivado. These ancillary components include the integrated logic simulator, the integrated logic analyzer and the IP Create and Package wizard. Xilinx also provides High-Level synthesis (C code -> HDL) and DSP generation tools with the Vivado System Edition. We will not use either HLS or DSP for this course but you are welcome to incorporate them into your final project if you desire.

The major components in the Xilinx embedded tool chain are:

- **Vivado** – This is the project manager and synthesis and place and route tool for Series-7 and Ultrascale Xilinx-based designs. It is the core of Xilinx's tool technology. The Vivado Project Manager is used to add the modules in the design to a project and run the synthesis, place and route, timing analysis and configuration file generation and download processes.

- **IP Integrator (IPI)** – IP Integrator is used to generate embedded system hardware. Its GUI provides a block diagram-oriented method to create Microblaze-based (used in this course) or Zynq (Dual ARM core)-based systems customized for your application. The IP Integrator provides a rich set of peripheral modules (called Xilinx IP) that can be added to your system and interconnected through a connection wizard. You can also include third-party IP and your own custom peripherals. In fact, I designed, implemented and packaged the two pieces of IP that you will use this term; Nexys4IO provides an interface the LEDs, switches, pushbuttons and Seven Segment displays on the Nexys4. PmodENC provides an interface to the Pmod rotary encoder and PmodOledRGB provides an interface to the PmodOled RGB display that you will use in the course. As a SoC system designer, you want to treat the Xilinx IP and third-party IP as black-boxes with well-defined interfaces which your application accesses through vendor-provided drivers. The good thing about IP Integrator is that it greatly simplifies embedded system hardware design allowing you to focus on your application-specific hardware and software. The bad news is there is a lot of behind-the-scenes processing going on – when everything works well it feels like "magic" and is way-cool, but when it doesn't…well, be prepared to spend time on Google, in the Xilinx knowledge base and user forums, poring through pages of log files and reading all sorts of documentation.

- **SDK –** This is the Xilinx Software Development Kit. Based on the Eclipse platform (an open-source industry standard used heavily by Java developers) and the GNU tool chain. The Eclipse GUI for the SDK provides the mechanisms to create and debug C and C++ application projects and code. To use the SDK you first import the hardware platform from Vivado and then build one or more board support platforms for your application. A board support platform includes the OS and drivers that will be used in your application. Project 1 and Project 2 use the "standalone" environment and whatever drivers are needed for the peripherals (custom and Xilinx IP) used by your application. The "standalone" environment provides the framework for the Xilinx drivers, most notably support for interrupt handling. Once you have imported the hardware platform and built a software environment you can then import/create and debug one or more application projects. Building a board support platform is mostly automated.

Building your first Microblaze-based embedded system

Vivado, the IP Integrator and SDK work together to provide an integrated environment for building SoC hardware and software for Xilinx FPGAs. You (the system creator), make use of these tools to specify the hardware to be used by your software application. You then create the software drivers and application program needed to implement your target application. In the remainder of this guide I will attempt to lead you through the tasks needed to build an HDL implementation of a full-blown embedded system platform consisting of a 32-bit CPU and associated peripherals. We will test the system by compiling, linking and loading an application which makes use of the hardware and the associated drivers.

Our target application is a program that exercises the PWM (pulse width modulation) operation. The PWM wave is internally generated by the signals for the onboard RGB leds. You just have to extract the signal and pass it to the detection module. The Oled RGB display driver has the required library functions to implement many of the functionalities of the project. A square or a rectangle is to be displayed which is filled with the color and the color is to be reproduced on the RGB leds onboard.

NOTE: WHILE THE NEXYS4 DDR CONTAINS A VARIETY OF HUMAN INTERFACE FUNCTION (BUTTONS, SWITCHES,LEDS, SEVEN SEGMENT DISPLAY) IT DOES NOT INCLUDE THE TWO EXTERNAL COMPONENTS THE DIGILENT PMODENC™ PROVIDES A ROTARY ENCODER WITH PUSHBUTTON AND A SLIDESWITCH AND THE PMODOLEDrgb DISPLAY WHICH PROVIDES 96 x 64 OLED RGB DISPLAY WITH A 16 BIT COLOR RESOLUTION . THE DIGILENT PMODENC™ SHOULD BE CONNECTED TO THE BOTTOM ROW OF CONNECTOR JD ON THE NEXYS4 BOARD. THE DIGILENT PMODOLEDrgb SHOULD BE CONNECTED TO THE CONNECTOR JA.

Building the hardware and executing the test program is a multiple step process. The major tasks are:

1. Create a Vivado RTL project and add the ECE 544 IP to the IP Catalog
2. Create the embedded system using the Create Block Design wizard (IP Integrator)
3. Configure the individual IP blocks and create the external ports for the embedded system (IP Integrator)
4. Synthesize, place and route and generate configuration files (.bit, .bmm) for the FPGA (Vivado)
5. Export the target hardware platform to SDK and create the target software environment (SDK)
6. Import and execute the test application (SDK)

These tasks are described in more detail in the sections that follow.

Task 0 – Get familiar with Vivado and IP Integrator

Wait…who said anything about a Task 0…the list in the previous section started with Task1?? Simply put, the rest of this document will make the most sense only if you have done some prep work. Those of you who have used the Xilinx tools in the past are most likely familiar with Vivado. You know much of what you need to know, except perhaps how to create the target embedded system. Fortunately doing that is a few mouse clicks away.

For those readers who are new to the Xilinx tool chain and Vivado, your best bet is to visit www.xilinx.com . There are a number of documents and videos to help you get started. Here are a few good jumping off places:

http://www.xilinx.com/training/index.htm

For those readers who are familiar with the Xilinx tool chain or ASIC and FGPA tool chains from other vendors, DocNav (the Vivado document catalog) provides a good jumping off point. DocNav can be started from the Vivado opening screen or from the Start Menu and provides access to tutorials, videos, reference manuals, user guides and the like. I prefer to start in the *Design Hub* view but the *Catalog* view provides a search function with filtering to narrow down the list. Consider watching a number of the Quick-take videos (also available in the Xilinx channel on YouTube); they provide a good introduction. Work though one or more of the tutorials before diving into the user manuals and reference guides. There are literally thousands of pages of documentation – it is overwhelming but, please, don't skip this type of preparation and simply dive into this Getting Started guide. If you do that you will quickly get lost in the details. I easily spent 5 or 6 hours looking at videos and reading before trying to create my first design…it was worth the effort.

Task 1 – Create a Vivado Project and add the ECE544 IP to the IP Catalog

The Vivado IP catalog provides point-and-click (and more conveniently, drag-and-drop) access to Xilinx IP, third-party and custom IP. Custom and third-party can be brought into the IP catalog by adding one or more repositories to the catalog. A repository is a directory that contains the Verilog and/or VHDL source code, documentation and the configuration and data files necessary to integrate a custom peripheral into a Microblaze or Zynq-based embedded system. Our first task is to create a new Vivado project and add the ECE 544 IP repository to the IP catalog. There is a single repository containing the ECE 544 custom peripherals (both the hardware and the drivers).

NOTE: IF YOU ARE PLANNING TO MAKE USE OF THE WCC LAB PC'S AND NEXYS4 BOARDS STORE YOUR REPOSITORY AND PROJECTS ON A FLASH DRIVE.

Step	Screen	Action	Explanation/Comments
1	*Xilinx Design Tools/Vivado 2015.4/Vivado 2015.4* in your *Start* menu NOTE: THIS IS THE PATH ON MY PC, YOURS MAY BE DIFFERENT	Starts Vivado and brings up the start screen	The Getting Started system has a top level module that instantiates the embedded system you will create with IP Integrator.
2	Create New Project	Click on *Create New Project* and then click on *Next>*. Browse to the directory you want to create the project in and name the project. Check the *Create project subdirectory box*. Click on *Next >*	Note: Name of directories and sub-folders should not have whitespaces.
3	Project Type	Select *RTL Project*. Do not specify the sources, we will add them later. Click on *Next >*	
4	Default Part	Category is General Purpose; Family and Sub-Family is Artix 7; Device is XC7A100T; Package is csg324; Speed is -1, Temp Grade is C (part is xc7a100tcsg324-1). Click on *Next >*	
5	New Project Summary	Click on *Finish >*	
6		Download the project 1 release and unzip *ip_repo_544_p1.zip* to your working directory for ECE 544 projects.	This should create the folder *ip_repo_544_p1*
7	Vivado main screen	Select Flow Navigator/Project Manager/IP Catalog. Right-click in the IP Catalog pane and select *Add Repository*. Highlight the path to *ip_repo_544_p1* and click on *Select*. When the IP Catalog is refreshed you should see a new folder called User Repository. Open the AXI Peripheral and the ECE544 IP folder and you should see new peripherals *Nexys4io_v2.0*, *PmodENC_v1.0* and *PmodOLEDrgb_v1.0*. If you don't see them try searching for them using the Search Bar	You can also right click in the IP Catalog pane and select *IP Settings…* to add the repository.
8	Project Manager/Add Sources	Add n4fpga.v (design source). Select n4fpga.xdc or n4ddrfpga.xdc (constraints file) depending on your fpga board from your project 1/getting started release directory. You will have to do this in two steps – one for the design file and the other for the constraints file	You may choose (or not) to copy the files to the project –it's up to you…but I always copy the files so I know where

			they are.

Task 2 – Create the embedded system (IP Integrator/Block Design Generator)

We have our top level module and have applied constraints (pin constraints, not timing constraints), but note that there is a **?** next to EMBSYS in the Sources/Hierarchy tab (you may have to expand n4fpga by clicking on the + box next to the file name). This is because you haven't created/configured the embedded system *(embsys)* yet. We will do that now.

Step	Screen	Action	Explanation/Comments
1	Vivado main screen	Select *Flow Navigator/IP Integrator/Create Block Design*. This will bring up the Create Block Design dialog. Name the design *embsys* and click *OK*. This will open the Block design screen (which will be empty). Click on *Add IP*. You may want to maximize the diagram.	The name of the block diagram must match the name in the n4fpga.v top level module
2	Block Design Diagram	Search for "microblaze" in the *Add IP* dialog and drag or double-click it into your diagram.	
3	Block Design Diagram	Search for *nexys4IO_0, PmodENC, PmodOledrgb, AXI GPIO, AXI-timer*, Fixed Interval Timer and *AXI Uartlite;* add an instance of each to the design.	
4	Block Design Diagram	Click on *Run Block Automation*. This step takes much (but not all) of the guesswork out of constructing a Microblaze-based system.	
5	Run Block Automation	In the Run Block Automation dialog set the: • Local Memory to *128K* • Local Memory ECC to *None* • Cache Configuration to *None* • Debug Module to *Debug Only* • Peripheral AXI port to *Enabled* • Interrupt controller to "*checked*" (we want to include it) • Clock Connection to *New Clocking Wizard (100MHz)* Click on *OK*	Most of these are the defaults but I thought I'd list them to keep you from getting nervous. Block Automation can run for several minutes so relax…you should end up with a block diagram that includes 10 blocks.
6	Block Design Diagram	Click on *Run Connection Automation*. Another seemingly magical step that wires most of your system together.	
7	Run Connection Automation	Select *All Automation*. Since we are using *btnCpuReset* as our reset signal we should set the reset polarity for the clk wizard and the reset clk wizard to polarity low. In the left pane click on *clk_wiz_1/reset* and set the reset polarity to *ACTIVE LOW*. Do the same for *reset_clk_wiz_1_100M/ext_reset_in*. The defaults for the other connections are fine. Click on *OK.*	Connection Automation can also run for several minutes…you should end up with all but your external ports connected.
8	Block Design Diagram	When Connection Automation has finished, right-click in the Diagram pane and select *Regenerate Layout*. This should tidy up what has become a bit of a wiring mess.	

Task 3 - Configure the IP blocks and create the external ports (Block Design Generator)

The Block Design Generator has brought us more than 90% of the way towards building our embedded system, but there is still work to be done. We have to customize several of the IP blocks and we have to create the external ports for our embedded system before we are finished. This is also done in the Design pane of the Block Design Generator.

Step	Screen	Action	Explanation/Comments
1	Block Design Diagram	Configure the clock generator. Right-click on the *clk_wiz_1* block and select *Customize Block…* Configuration: • Clocking Options tab/ Input Clock/Primary/Source is *single-ended clock capable* pin. Remaining defaults are OK • Output Clocks tab/Enable Optional Inputs/Outputs – deselect *reset* –it is not used • Output Clocks tab – check *clk_out2*, output freq 50.00 in the *requested* column. Click on *OK*	You may have to scroll down to see some of the options…most notably on the Output Clocks tab Checking the clk_out2 enables extra clock signals from the clocking wizard. The value set their frequencies in MHz.
2	Block Design Diagram	Configure the Fixed Interval Timer. Right-click on the *fit_timer_0* block and select *Customize Block…* Configuration: • Number of Clocks is *2500* • Allowed Inaccuracy is *0* Click on *OK*	The system clock is running at 100MHz. We want a timer “tick” of 40KHz. 100MHz/40KHz = 2500.
3	Block Design Diagram	Configure the AXI Timer. Right-click on the *axi_timer_0* block and select *Customize Block…* Configuration: • Width of the counter is *32-bits* • Timer 2 is *disabled* Click on *OK*	
4	Block Design Diagram	Configure the AXI Uartlite. Right-click on the *axi_uartlite_0* block and select *Customize Block…* Configuration: • Baud Rate is *19200* • Data Bits is *8* • Parity is No Parity Click on *OK*	You can select a different Baud rate but you must be consistent in whatever you connect the serial port to.
5	Block Design Diagram	Configure the AXI GPIO. Right-click on the *axi_gpio_0* block and select *Customize Block…* Configuration: • GPIO width is *8-bits*, *All Inputs* is *“checked”* • Enable Dual Channel is *“checked”* • GPIO2 width is *8-bits*, *All Outputs* is *“checked”* Click on *OK*	

6	Block Design Diagram	Make the external connections to *clk_wiz_1* and *rst_clk_wiz_1_100M* • Delete the now orphaned *diff_clock_0* and *reset_rtl_0* if they are in the diagram • Right-click on the *clk_in1* pin on *clk_wiz_1* and select *Make External*. This will create an input port named *clk_in1* (no surprise there.) Right-click on the *clk_in1* port and select *External Port Properties*. Change the name of the port to *sysclk* • Right-click on the *ext_reset_in* pin on *rst_clk_wiz_1_100M*. Make the port external and change its name to *sysreset_n* • Draw a line from *clk_wiz_1/locked* to the *rst_clk_wiz_1_100M/dcm_locked* input to make a connection if it doesn't exist. • Draw a line from *clk_out2* to the *ext_spi_clk* on *PmodOLEDrgb_0*	Hovering your mouse cursor over the pins will cause a pencil icon to appear when the component is selected. Press the DEL key on your keyboard or right-click and select Delete
7	Block Design Diagram	Make the connections to *fit_timer_0* • Draw a line from *rst_clk_wiz_1_100M/peripheral reset* to the *Rst* input on *fit_timer_0* to make a connection • Draw a line from *clk_wiz1/clk_out_1* to the *Clk* input on *fit_timer_0* to make a connection • Connect the *Interrupt* output on *fit_timer_0* to the *In0* pin on *microblaze_0_xlconcat* • Connect the *AXI Timer interrupt pin* to the *In1* pin on *microblaze_0_xlconcat.*	*microblaze_0_xlconcat* block does as the name suggests. It concatenates its input to single vectored output dout[]. The Microblaze interrupt controller can handle multiple interrupt sources with the lowest input bit (In0 in this case) being the highest priority interrupt.
8	Block Design Diagram	Make the connections to *nexys4io_0* • Select the btnU, btnD, btnC, btnL, btnR, sw[15:0] and make them external • Select the led[15:0] outputs and make them external • Select all of the RGB1 and RGB2 outputs and make them external • Select seg[6:0], dp and an[7:0] and make them external.	
9	Block Design Diagram	Make the connections to *PmodENC_0* and *PmodOLEDrgb_0* • Select the PmodENC pins and make them external • Select the *PmodOLEDrgb_out* pin and make it external	
10	Block Design Diagram	Make the connections to *axi_gpio_0* and *axi_timer_0* • Select *axi_timer_0/generateout0* and connect it to *nexys4IO/ clock* • Select *the axi_gpio_0/GPIO* and *axi_gpio_0/GPIO2* interface and rename the signals to *gpio_0_GPIO* and *gpio_0_GPIO2,* respectively • Make the ports external and rename, if no external ports are available.	We won't use the *axi_timer_0* interrupt in this application, but there may be applications where we could.
11	Block Design Diagram	Double check the wiring to make sure all of the signals are connected properly. Regenerate the layout to neaten it up and save the Block Design (*File/Save Block Design*). ***The project release contains a file docs/Getting Started/gs_embsys_schematic.pdf that is a schematic of the completed embedded system. CHECK YOUR DESIGN AGAINST IT.*** Check that all of the AXI devices have been assigned an address range by looking at the *Address Editor* tab.	The layout orientation of the design may differ from the one generated on my device The comparison is to be made between the connections and check if all blocks are present.

12	Vivado main screen	Select *Flow Navigator/IP Integrator/Generate Block Design*. The IP Integrator will then assemble all of the HDL from all of your peripherals and the Microblaze, create tcl scripts and makefiles, and, in general, build an HDL version of your embedded system and its peripherals. If you've configured and connected everything properly the block design will be generated successfully. If generation fails check all of your connections, make necessary changes, and generate the outputs again.	Generating your block design could take several minutes.

Task 4 - Synthesize, Implement and Generate bitstream for the FPGA (Vivado)

The file "embsys.v" is generated from the block diagram and it contains a module "embsys" which has all the pins/ports name used by the design. These are the pins which we use to connect the external ports of the Nexys board by instantiating the module of embsys in the top level module of n4fpga.v file. Having succeeded at Tasks 2 and 3 your embedded system should have been added to your project. If there is still a **?** next to the EMBSYS icon in the Hierarchy window you may have not named your block design "embsys". Edit the top level n4fpga.v file to match the name of your block design. That should remove the **?** but the design may not synthesize correctly because there is a mismatch between the port properties and names of your block design and the wires defined in n4fpga.v. You must correct any mismatches before attempting to synthesize the design. Synthesis could take upwards of 20 minutes per run so you want to do your best to make sure the port widths, direction, etc. are correct.

The last step before attempting to synthesize, route and generate a bitstream is to make sure the ports in the embedded system are mapped correctly to the ports in the EMBSYS instantiation in n4fpga.v. The comparison is done manually (visually) by looking at the instantiation in n4fpga.v and comparing it to either a template that can be provided by the IP Integrator or the "embsys.v" file which contains all the pins.

To view the template of IP Integrator go to *Project Manager/Sources pane*, right-click on EMBSYS and select *View Instantiation Template*. You can right-click on that tab to float the window and then double click and right click on n4fpga.v in the *Hierarchy* screen to display its contents in a floating window. Do a port by port comparison between the instantiation template and the EMBSYS instantiation in n4fpga.v. You want the port names to match and be connected to the appropriate top level ports.

Once you have reconciled the port names save n4fpga.v and execute *Synthesis*. Check all warnings carefully, and in particular, be on the lookout for unconnected signals, port width mismatches, signals driven to 0 and/or optimized out, etc. If synthesis fails, check the port names and widths again, open the Block design and look for missing inputs or outputs, and, in general, trust but verify. For example, even though I double checked when I first built my Getting Started system synthesis failed. It turns out that I had forgotten to make led external in my embedded system and misspelled sysreset_n (syreset_n).

NOTE: THERE WILL BE UNCONNECTED SIGNALS (LIKE THE SOME OF THE JB AND JC PORTS) THAT WILL GENERATE WARNINGS BUT ARE OK. THAT'S WHY IT'S IMPORTANT TO UNDERSTAND THE DESIGN AND CHECK YOUR WORK CAREFULLY.

After you have successfully synthesized the design and resolved the warnings (my design shows 195 warnings after synthesis), *Implement* and *Generate Bitstream* like you did in ECE 540. This is a big design that will take many minutes to place and route. Be patient, and once again, check the logs carefully when a process is done (my design shows 233 warnings in total after generating the bitstream).

NOTE: IF YOU GET AN ERROR "[Drc 23-20] RULE VIOLATION (BIVC-1) BANK IO STANDARD VCC - CONFLICTING VCC VOLTAGES" WHILE RUNNING IMPLEMENTATION THEN GO TO FLOWNAVIGATOR/SYNTHESIS/OPEN SYNTHESIS DESIGN/SCHEMATIC. IN THE SCHEMATIC YOU SHOULD SEE THREE OPTIONS OF CELLS, I/O PORTS, NETS. SELECT I/O PORTS AND SCROLL DOWN TO THE PIN WHICH HAS VOLTAGE OTHER THAN 3.3V. CHANGE THE SELECTION TO LVCMOS33. THIS ERROR IS IN THE .xdc FILE. AFTER YOU MAKE THE CHANGES SAVE THE CHANGES AND OVERWRITE THE .xdc FILE. RE-RUN THE IMPLEMENTATION AND GENERATE BIT-STREAM.

The last step in the hardware creation process is to export the design (to SDK). Exporting the design packages the bitstream, drivers, etc. for handoff to SDK. To export a design select File/Export/Export Hardware from the Vivado main screen and include the bitstream.

Pat yourself on the back - you have created your first target hardware platform with the Xilinx embedded tool chain. With the hardware platform complete it's time to move on to the application software.

Task 5 - Import the target hardware platform to SDK and create the target software environment (SDK)

Software development/debug in the Xilinx embedded system tool chain is done using the SDK.

The Xilinx SDK used for this course is based on a Microblaze port of the GNU tool chain. Since the GNU tools are command line based, Xilinx has wrapped them in an Eclipse-based GUI. Eclipse is an open source IDE (Integrated Development Environment) that is used extensively in the Java world. Eclipse has been adapted to many CPU architectures, many tool chains, and many vendor product offerings.

One Eclipse concept worth mentioning is "Perspectives." The GUI changes depending on what the user is doing. For example, there is one "Perspective" for code development and another for program debug. It's not the intent of this guide to provide a deep look at the SDK. There are a number of tutorials (called cheat sheets in the Eclipse vernacular) that do a far better job than I could. For this example our task is to:

- configure the SDK for our target hardware
- create an appropriate board support package
- import/execute a working application

In this section we will take care of the first two items.

Step	Screen	Action	Explanation/Comments
1	Vivado main screen	Open the SDK from Vivado by selecting *File/Launch SDK* from the Vivado main screen. Doing this should cause SDK to import the project and bitstream into the SDK.	You can open SDK from the *Start* menu, as well, but for the first time after you export the hardware it seems better to open SDK from Vivado.
2	Select a workspace	SDK may suggest a path that is "Local to Project" if you open it from Vivado. You may also specify your own workspace directory. NOTE: The SDK may display a welcome screen or pane. If it does, take a few minutes to review introductory videos and tutorials. Closing the Welcome tab will bring up (or expand) the SDK main screen.	The Xilinx SDK does not readily lend itself to keeping all of your projects in a single workspace. In other words, keeping your SDK workspace local to your project may be a wise choice even though it makes it inconvenient to share software between different hardware systems.
3	Project Explorer	If you successfully exported your design from the previous step the Xilinx SDK should indicating that it is importing a hardware project. Once the import is complete the SDK Project Explorer should show a hardware system called *n4fpga_hw_platform_0*. The SDK allows	

		you to create your own hardware systems to try different configurations. You should also see a file called *system.hdf* open in the editing pane.	
4	Project Explorer	Select the *File/New/ Board Support Package* menu item. NOTE: The SDK gives several Operating System choices. These include the Xilinx standalone OS and xilkernel (a small Real-Time kernel) and freertos which is an open source Real-Time Operating System. We will use the standalone OS for this project. The defaults should be OK to create a standalone board support package that can be used for the project. Check that the Board Support package will be built for the correct hardware platform (in this case *n4fpga_hw_platform_0*). Click on *Finish*. This should bring up the *Board Support Package Settings* dialog box Click through the BSP settings. You should see *stdin* and *stdout* assigned to *axi_uartlite_0*. In the drivers dialog you should see that all of the drivers have been successfully pulled from the IP repository and brought into your design. More to the point, make sure that the *PmodENC*, *PmodOLEDrgb* and *Nexys4IO* drivers are selected. If they are not (you see *none* or *generic*) you will need to make SDK aware of the location of your repository (see step 5) Click on *OK*	The default setting for SDK is to automatically do a system or program build when a change to any of the files has been made and save. You can disable this by deselecting Project/Build Automatically. If automatic builds are enabled SDK will try to build the drivers and board support package. Let it do so; it's quick and doesn't hurt anything… and it's better to find out sooner, rather than later, if you are going to run into problems building your software.
5	Project Explorer	NOTE: I HAVE NOT HAD TO DO THIS SINCE I SWITCHED TO VIVADO 2014.4. I'M HOPING YOU'RE JUST AS FORTUNATE. Select the *Xilinx Tools/Repositories* menu item. Add a new Global Repository. Browse to the directory that includes your *IP repository (ip_repo)*. Rescan the repositories and return to *system.mss* in the edit pane. This time the correct drivers should be loaded, or if not, there should be a dropdown that lets you select them. Click on *OK*	SDK needs to know where the drivers for the custom peripherals are located.

6	Project Explorer	Check that the hardware system and board support package look correct.	
7	Project Explorer	Double-click on *system.hdf* in the Project Explorer navigation tree if it is not visible in the center pane of the Project Explorer. Check (once again) that all of your devices have been included and all have valid address ranges. The address ranges are where the registers and memory controlling the peripheral are based. If it doesn't look like the hardware system was imported correctly, go back to Vivado, check that the system is complete and correct, and then export the project again. Your hardware should have a valid address map (all peripherals assigned). The target device should be an *7a100*t. The IP blocks present in the design should include the Microblaze and its modules, *axi_uartlite_0* (the serial port), *axi_timer_0* (timer), *microblaze_0_axi_intc* (the interrupt controller), *nexys4IO_0* (the Nexys4 interface), *PmodOLEDrgb_0* (the OLED display),*PmodENC_0* (rotary encoder) *fit_timer_0* (the fixed interval timer) and *axi_gpio_0* (the general purpose I/O port). .	
8	Project Explorer	Double-click on *system.mss* in the Project Explorer navigation tree. The file should be in *standalone_bsp_0*. Your board support package should be targeted to your hardware platform with its target processor *microblaze_0*. The Operating System should be the standalone OS and there should be peripheral drivers for all of the devices except *fit_timer_0* which is a standalone (e.g. no driver or AXI bus connection) peripheral. Confirm that the peripheral drivers for nexys4io, pmodOLEDrgb and pmodENC are the correct drivers and are not generic or none	
9	Project Explorer	Expand the *standalone_bsp_0/ microblaze_0/include*, scroll down and find the *xparameters.h file.* Scroll and find the *Definitions* for the *Nexys4IO* and replace the "*base address and high address*" to the values given in the file *system.hdf.*	The first column gives the base address and the next column gives the high address

Task 6 – Import and execute the test application (SDK)

We're almost there. So far we have synthesized and implemented our target hardware system (including the embedded computer system) and prepared the SDK for application development and debug by importing the hardware description and building a software platform for the application to run on. All that's left is to create the target application, download the target hardware system to the FPGA, download the application to the CPU memory (128KB of Block RAM in the target hardware system) and then run and/or debug the program. Our target application for this guide is the system test application called *ece544ip_test.c*. The other library files are *platform.c, platform.h and platform_config.h.*

Step	Screen	Action	Explanation/Comments
1	*Xilinx Design Tools/SDK 2015.4/Xilinx SDK 2015.4* in your *Start* menu NOTE: THIS IS THE PATH ON MY PC, YOURS MAY BE DIFFERENT	Reopen the SDK if you have exited it. SDK should remember your last project (created in Task 5)	
2	Project Explorer	Select the *File/New/Application Project* menu item.	
3	New Project/ Application Project	Specify a name for the project; The Target Hardware should be *n4fpga_hw_platform_0* and the target processor should be *microblaze_0*. The Language should be C and the Board Support Package should be *Use existing* and your board support package (*standalone_bsp_0*). The default location is OK. Click on *Next*	This step creates a (mostly empty) framework for a C application program. You can create and work with more than one C application in a software platform and more than one software platform for a hardware platform
4	New Project/Templates	Select *Empty Application* and Click on *Finish*	
5	Project Explorer	Generate a linker script by right-clicking on your software project and selecting *Generate Linker Script.* This will open a linker script dialog. Accept the defaults by clicking on *Generate*.	NOTE: You don't really need to do this for this project because a default link script will be created for you. However, you may implement systems with more complicated memory maps so it's good to know you can do this.
6	Project Explorer	Import the *(ece544_IP_Test)* application by right-clicking on the */src* folder in the project and selecting *Import*.	

7	Select	Select *General*/*File System* and click on *Next >*	SDK can import projects from a number of sources (an existing SDK project, for example). For this example we are just going to import the C source code files that make up the application
8	File System	Browse to the *project1 release\software* directory. Select all of the source (.h and .c) files. Click on *Finish* Unless you have disabled the *Build Automatically* option in the Project menu the SDK will attempt to compile, link and generate an executable image (*(proj_name).elf*). If *Build Automatically* is disabled you can force a build by selecting the *Project/Build All* menu item. The build should succeed and the SDK console should show that *(proj_name).elf* passed the elf check. If this is not the case fix the problem(s) (compile errors, for example), save the file and build again.	Once you have successfully built the target application you are ready to download the hardware configuration to the FPGA and the application image to the Microblaze memory.
9	Project Explorer	You are now ready to configure the FPGA with the target hardware system. Connect the PMODs to your board (the PmodENC into the JD header and the PmodOLEDrgb into the JA header). Connect the board to your PC with the USB cable and power up the board. Your PC should find the Xilinx cable drivers and open a hardware session with the Nexys4 DDR. Select the *Xilinx Tools/Program FPGA…* menu item.	
10	Program FPGA	Selecting Xilinx Tools/Program FPGA should bring up a dialog box pointing to *n4fpga.bit* and *n4fpga.mmi* files you exported. If not, browse to the files. *n4gpga_hw_platform_0* should be selected as the hardware platform. We will initialize the bitstream with the bootloop (the default) since that allows for download and debugging from Eclipse. Click on *Program* If all goes well you should get a dialog box saying the FPGA configuration is complete. Initializing the bitstream and downloading it to the FPGA could take a couple of minutes.	You can also initialize the bitstream to include your executable, meaning the program will start up automatically once the FPGA is configured.
11	Project Explorer	The final step before running the program is to download the application image (*(proj_name).elf* in this case) to the CPU memory. There are two run environments – *Run as…* and *Debug as… Debug as…* executes the program under the GNU debugger. The GNU debugger allows you to set breakpoints, look at variables, single step, etc. *Run as…* simply loads and runs the program – no debugger. Since you will most likely be debugging a program we will run the program under the debugger. Right Click on the application project and then select the *Run/Debug As/Launch on Hardware* menu item. SDK will rebuild the software environment and application (if	We added an *axi_ uartlite* to serve as a console for the app. You can connect to a terminal emulator on your PC (we will do this in project 2) but you can also assign *stdin* and *stdout* to *axi_uartlite_0*. This is done by connecting STDIO to the console

		necessary), initialize the debugger and then download the executable file. If SDK is successful it will open the Debug Perspective and wait for you to execute debug commands. There are icons to single step into functions, single step around functions, stop, start, set breakpoints, etc. Refer to the SDK tutorials	in either a Run or Debug configuration. Don't forget to set the baud rate to match what you configured in *axi_uartlite_0* (19200 if you followed the instructions.)
11	Xilinx Software Development Kit Debug Perspective	Test the operation of the program. Experiment with the debugger commands and when you are confident that program seems to be working click on the green arrow. This will cause the program to run until it hits a breakpoint or exit. You can stop the program by clicking on the red box (stop). You can reset the system by pressing BtnCpuReset but you may have to download the FPGA contents and the program again to restart it. I didn't put any effort into ensuring that the *ece544_IP_Test* program gracefully exits and restarts.	NOTE: Most embedded system programs do not have a mechanism to exit. The programs run in an infinite loop that many of us call "the main loop."