

FPGA to the Cloud

By

Moundji Kazi-Tani, P.Eng.

B. Eng. (Electrical), Concordia University, 2006

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering

In the
School of Engineering Science
Faculty of Applied Science

© Moundji Kazi-Tani
SIMON FRASER UNIVERSITY
Spring 2017

Copyright in this work rests with the author. Please ensure that any reproduction or re-use is done in accordance with the relevant national copyright legislation.

Approval

Name: Moundji Kazi-Tani, P.Eng.

Degree: Master of Engineering

Title: FPGA to the Cloud

Examining Committee: **Chair: Dr. Ash M. Parameswaran, P.Eng.**
Professor

Dr. Craig Scratchley, P.Eng.
Senior Supervisor
Senior Lecturer

Victor Gusev Lesau
Industry Supervisor
Electrical Engineer
CEO, CuePath Innovation

Date Presented/Approved: March 29, 2017

Abstract

FPGAs are enabling more applications to be put to the market at a fraction of the cost of ASICs and with a much faster deployment rate. However, the wide range of FPGA brands and types currently available on the market; could overwhelm first time users when choosing a suitable FPGA for a given application. Furthermore, intermediate-to-advanced FPGA users may desire to evaluate some new FPGAs before committing to a purchase. FPGA to the Cloud is a web application that allows users to interact with FPGA evaluation kits remotely on a try-before-you-buy or pay-per-use model. The end user would access a web site where the web application is hosted. The end user would select an FPGA evaluation board from a list, and would be given direct remote access to said FPGA board; with programming tools. The user could use available sample FPGA design files, or upload user-created FPGA design files; for testing and evaluation. The project-prototype is based on the ZedBoard which uses Xilinx's Zynq-7000 FPGA. The web application was developed using Laravel's PHP framework.

Keywords: Web Application; Web Design; FPGA Application; PHP; Frameworks;
Web Hosting

To Bill

Acknowledgements

This project would not have been possible without the support of many people. I would like to thank Victor Gusev Lesau without whom the project would not have started nor finished!

I would like to thank the examining committee members for dedicating their time and energy in reviewing the project.

As well, I wish to express my gratitude to the late Dr. William A. Gruver for his thoughtful guidance. His wisdom and inspiration made a significant impact on the people around him.

Furthermore, I would like to thank my family for their never-ending support and patience.

Table of Contents

Approval.....	ii
Abstract.....	iii
Dedication.....	iv
Acknowledgements.....	v
Table of Contents.....	vi
List of Tables.....	ix
List of Figures.....	x
List of Acronyms.....	xi
Chapter 1. Introduction.....	1
1.1. Motivation.....	1
1.2. Application Summary.....	2
1.3. Academic Objectives.....	3
Chapter 2. Project Overview.....	4
2.1. Organization.....	4
2.2. FPGA Development.....	4
2.3. Web Application.....	5
Chapter 3. System Architecture.....	7
3.1. The Hardware Server.....	8
3.2. The Web Application.....	8
3.3. The End User.....	9
Chapter 4. System Design Choices.....	10
4.1. FPGA Evaluation Board Selection - ZedBoard.....	10
4.2. Hardware Server OS - Ubuntu.....	11
4.2.1. Virtual Machine Software – VMware.....	12
4.3. Web Cam.....	12
4.4. Web Scripting Language Selection - PHP.....	13
4.4.1. Frameworks - Laravel.....	13
4.4.2. IDE Tools - Homestead.....	14
4.5. Web Hosting Services - AWS.....	14
4.6. Software Version Control System and Online Repository – BitBucket.....	15
4.7. Web Based Terminal Emulator – Shell in a Box.....	15
Chapter 5. System Implementation: FPGA Board Component.....	17
5.1. ZedBoard Setup and Configuration.....	17
5.2. LED Binary Counter on the ZedBoard Using the Zynq-7000 FPGA.....	18
5.2.1. Step 1: Creating a New Project in Vivado.....	18
5.2.2. Step 2: Creating a Block Design.....	20
5.2.3. Step 3: Writing the Software Application.....	28
5.2.4. Step 4: Programming the ZedBoard and Running the C Application.....	32

5.3. Other FPGA Designs.....	33
Chapter 6. System Implementation: Web Application	34
6.1. Tree View of the Web Application.....	34
6.2. Model-View-Controller Code Structure	36
6.2.1. The Models.....	36
6.2.2. The Controllers.....	37
6.2.3. The View	38
6.3. Services	40
Chapter 7. Website Layout	42
7.1. Landing Page	42
7.1.1. Top Menu and Image Slider.....	42
7.1.2. Server and FPGA Boards Selection Menu.....	43
7.1.3. Server and FPGA Scheduling Tool.....	44
7.2. Remote Server Page.....	45
7.2.1. File Manager and Program Tool	46
Chapter 8. Conclusion	49
8.1. FPGA Applications Development: What Was Learned	49
8.2. Web Application: What Was Learned	50
8.3. Known Bugs	50
8.4. Future Work.....	51
References.....	55
Appendix A. VMware Workstation Player and Ubuntu 14.04.2 Virtual Machine Installation Guide	57
Appendix B. Hardware Server Configuration Guide.....	68
Appendix C. Open SSH Installation on the Hardware Server.....	71
Appendix D. Motion (Webcam Stream) Installation on the Hardware Server	72
Appendix E. C-Kermit (Serial Terminal) Installation on the Hardware Server	73
Appendix F. Git Installation on the Hardware Server	74
Appendix G. Shell in a Box Installation on the Hardware Server.....	75
Appendix H. Network Ports Configuration.....	76
Appendix I. Xilinx Vivado Design Suite Installation Guide	77
Appendix J. USB Serial (COM) Port Properties.....	87
Appendix K. Vivado Zynq-7000 FPGA Design Guide A: LED Binary Counter	88

Appendix L.	Vivado Zynq-7000 FPGA Design Guide B: LED Scanner Light	113
Appendix M.	Vivado Zynq-7000 FPGA Design Guide C: LED – UART IO	115
Appendix N.	Vivado Zynq-7000 FPGA Design Guide D: Peripherals Tests	117
Appendix O.	Vivado Zynq-7000 FPGA Design Guide E: Memory Tests	120
Appendix P.	Vivado Zynq-7000 FPGA Design Guide F: Zynq DRAM Tests	123

List of Tables

Table 1 ZedBoard Boot Mode Jumper Settings	17
--	----

List of Figures

Figure 1 Top Level Architecture	7
Figure 2 The ZedBoard Featuring the Xilinx Zynq-7000	11
Figure 3 ZedBoard Boot Mode Jumper Settings - Option 1	18
Figure 4 Project Name and Location in Vivado.....	19
Figure 5 Board Selection in Vivado	20
Figure 6 Create Block Design in Vivado	21
Figure 7 Add IP in Vivado.....	22
Figure 8 Add Zynq7 Processing System IP in Vivado.....	23
Figure 9 Run Block Automation in Vivado	23
Figure 10 Added AXI GPIO IP in Vivado	24
Figure 11 Connect the AXI GPIO IP to the Board LED 8 Bits in Vivado.....	24
Figure 12 Regenerate Layout (I.E. Refresh) in Vivado	25
Figure 13 Create HDL Wrapper in Vivado	26
Figure 14 Generate Bitstream	27
Figure 15 Export Hardware And Bitstream in Vivado.....	28
Figure 16 Create a New Board Support Package in SDK.....	29
Figure 17 Create an Application Project in SDK	30
Figure 18 Select Hello World Template in SDK	30
Figure 19 LED Binary Counter C code in SDK	32
Figure 20 Program FPGA in SDK.....	32
Figure 21 Running the C Application on the Hardware in SDK.....	33
Figure 22 The Web Application's Structure Main Overview	35
Figure 23 Models, Controllers, and Helpers Source Code Tree.....	37
Figure 24 The View Source Code Tree	40
Figure 25 Landing Page with Image Slider and Top Menu	42
Figure 26 Top Menu as Seen by Logged-In Users	43
Figure 27 Server and FPGA Board Selection Menu	43
Figure 28 Server and FPGA Board - Administrative Settings.....	44
Figure 29 Server and FPGA Board Scheduling Tool	45
Figure 30 Remote Server Page: Conneciton to Remote FPGA Hardware Server.....	46
Figure 31 File Manager and Program Tool	47
Figure 32 Program Tool: Programming Progress	48

List of Acronyms

AI	Artificial Intelligence
ASIC	Application Specific Integrated Circuit
AWS	Amazon Web Service
CLI	Command Line Interface
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
DB	Database
EC2	Elastic Cloud Computing
FPGA	Field-Programmable Gate Array
GP	General Purpose
GPIO	General Purpose Input/Output
GPU	Graphics Processing Unit
IC	Integrated Circuit
IDE	Integrated Development Environment
IP	Intellectual Property; Internet Protocol
ISO	International Standard Organization
JS	JavaScript
LED	Light-Emitting Diode
MVC	Model-View-Controller
OS	Operating System
PL	Programmable Logic
PLD	Programmable Logic Device
PS	Processor System
SDK	Software Development Kit
SFU	Simon Fraser University
SSH	Secure Shell
TOS	Terms of Service
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface
URI	Universal Resource Identifier
URL	Universal Resource Locator
URN	Universal Resource Name

VCS
WWW

Version Control System
World Wide Web

Chapter 1. Introduction

1.1. Motivation

FPGA usage has been on the rise over the last ten years. And such increase is projected to continue over the next ten years. The FPGA market size was valued at USD 5.27 billion in 2014, and it is expected to reach USD 14 billion by 2024 [1] - [2]. The telecom sector, which accounts for 33% of the current FPGA market share is seeing continued growth [1]. And as more ICs are making their way into the average vehicle, the automotive industry is another target for FPGA vendors; at 17% of the current FPGA market share [1].

Furthermore, FPGAs are becoming the flavor of choice (over CPUs and GPUs) for future AI services and applications. FPGAs run faster than software, consume less power than CPUs or GPUs, and could be reprogrammed. Intel¹ estimates that FPGAs will run 30% of data center servers by the year 2020 [3]. In neural networks; FPGAs are currently aimed at inferencing (evaluating already trained neural networks) for applications such as image recognition, speech recognition, and language translation [4].

With numerous FPGA manufacturers around, and several types of FPGA evaluation kits available, selecting the right FPGA for a given application could be a daunting task to a student, a starting hobbyist or even a seasoned engineer. Even advanced FPGA users have to rely mostly on the manufacturers' specifications before committing to an FPGA selection.

To help users make the appropriate FPGA selection; there is: FPGA to the Cloud. FPGA to the cloud is a project-prototype that allows users from anywhere on the web, to gain remote-access to several types of FPGA development kits on a try-before-you-buy model, pay-per-use, subscription or other models.

¹ Intel acquired FPGA maker Altera in 2015 [9]

1.2. Application Summary

The purpose of this project-prototype is to provide would-be users with an online web application to access FPGA-based evaluation boards remotely:

- To learn basic fundamentals about FPGA design and programming
- To test their own FPGA designs
- To try different types or brands of FPGA development kits

Users could explore various designs, types, or brands of FPGAs without the need of purchasing any particular FPGA evaluation board, nor worrying about the upfront relative high cost of such a purchase.

Example of applications:

- Students wanting to learn/apply FPGA-design basics, but are on a limited budget and can't afford the relative high cost of FPGA evaluation board kits
- Users may want to test or try different FPGA evaluation kits before deciding on a specific brand or model to commit to
- Academic institutions: no need for large lab space. Just a small room with servers and a selection of FPGA boards. Students can access these boards remotely through a web application from the classroom or anywhere else
- Third party vendors could offer such a platform on a try-before-you-buy model
- Application engineers or technical sales engineers could demo the system's functionality to a customer without having to carry around the complete system as well as heavy and bulky test equipment
- Open source users or independent developers who are not willing to purchase a specific FPGA evaluation kit, but rather rent-per-use
- Developers who need to test their code on multiple embedded systems; after each software build, would not need to have such systems setup locally

1.3. Academic Objectives

Through this project the student (under the supervision of the industry supervisor) was required to learn, design, develop, test and debug all aspects of the project, which covered the following elements:

- Learn the fundamentals of FPGA design
- Create several basic FPGA designs
- Learn current web development and design methodologies, languages, tools, and the use of online hosting services
- Learn and use version control software tools

Chapter 2. Project Overview

2.1. Organization

The overall project tasks were divided into two major components:

- FPGA development: All aspects related to the FPGA hardware and software; including FPGA design tools and programming bitstream files, as well as Server and OS configurations
- Web application: All aspects related to the web development, design and interface; both front end and back end

2.2. FPGA Development

The first step of the project was to pick an FPGA evaluation platform; for the purpose of building the proof of concept around it. The ZedBoard Development Board (by Digilent) using the Zynq-7000 ARM/FPGA SoC (by Xilinx) was chosen (More details in section 4.1).

The ZedBoard along with the Xilinx Vivado software suite were used to develop from scratch several basic FPGA-based designs to allow the integration of the hardware component (FPGA setup) with the software component of the project (web application). Several basic FPGA designs were created such as:

- Blinking LEDs
- UART user input mapped to LEDs
- Peripherals Tests
- Memory Tests
- DRAM Tests

These demo-designs help in three ways:

- Provide the web developer with some simple FPGA designs to develop and test the web application component of the project
- Provide the end user whom would be accessing a given FPGA setup online remotely; some sample FPGA files to get familiar with how the web application works
- Use of such sample FPGA design file during the demo of this project

2.3. Web Application

The second main component of the project is the web application. This is the larger portion of the project as it sets up the required platform for providing web access to a given FPGA evaluation board kit.

The web application component covers all that is web related, from web design to web development, to web hosting, etc....

The website (hosting and providing this service) offers the following functionalities:

- User account creation and login
- User and administrative level access to user and website settings
- Menu selection: choosing the type of the desired FPGA board (to connect to)
- Booking/calendar application to reserve time slots for a given FPGA board
- Live web cam stream showing the FPGA board the user is connected to (to monitor any visual outputs, if applicable)
- File manager: to upload and program the FPGA; with user-created files (or to load sample demo files available on the server)
- Power cycle access to reboot the FPGA board remotely
- Terminal access to the server, for a more advanced level of control

- UART console to provide input/output access to the FPGA (if applicable)

Chapter 3. System Architecture

The architecture of this project-prototype could be split into three major components:

1. The hardware server (I.E. the FPGA evaluation board server). This includes the FPGA evaluation board in question
2. The Cloud, which hosts the web application and manages all access and interaction between the hardware server and the end user. As well as user-access management
3. The end user, whom uses the web application through the Cloud to access an FPGA evaluation board remotely

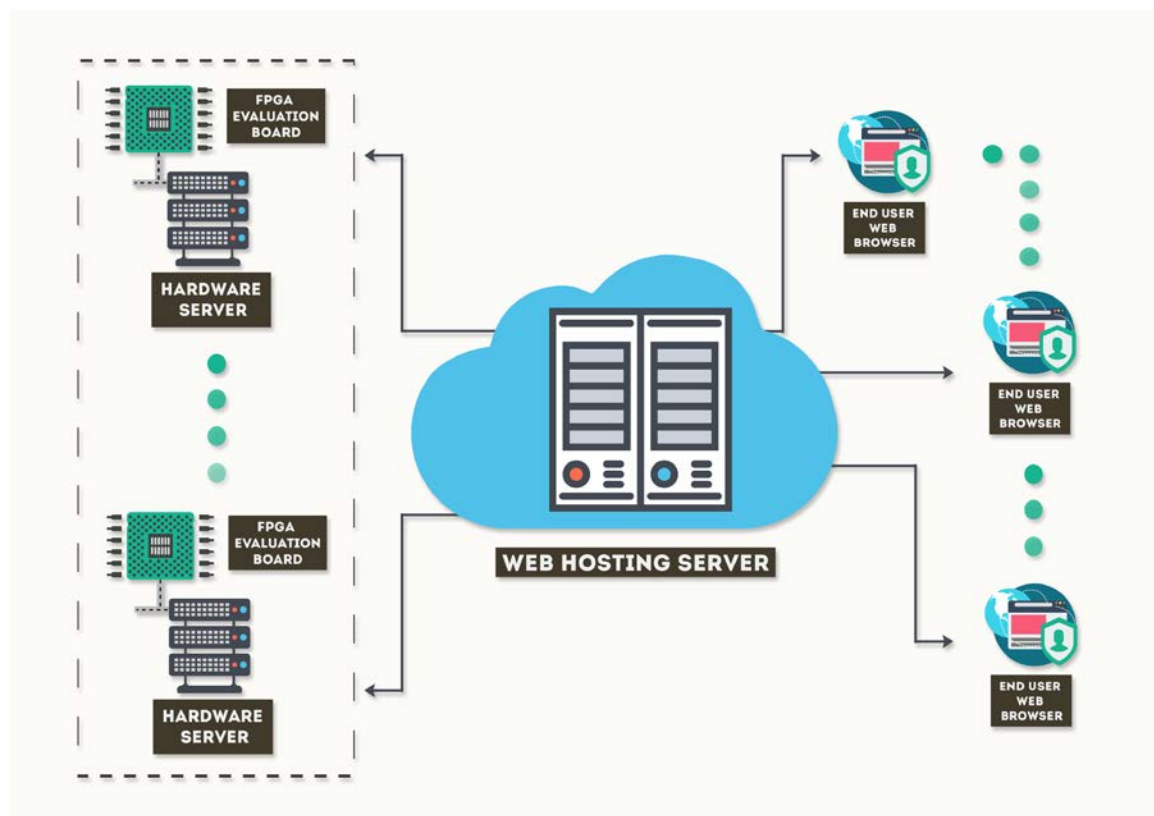


Figure 1 Top Level Architecture

3.1. The Hardware Server

The hardware server consists of two main sub-components:

1. A server. Or any PC-based machine
2. An FPGA evaluation board

Other dependent components:

- All required OS, software and configurations to ensure the server's network accessibility and functionality
- All required software tools, drivers and cables to connect the FPGA evaluation board to the server

3.2. The Web Application

This is the main component of the project. The web application is the center point where it would allow online users to interact with various FPGA evaluation boards remotely, without the need for a direct physical access to said FPGA boards. The web application accomplishes the following:

- Front end access point to the FPGA evaluation boards:
 - Access management to sample demo FPGA design files
 - Demo files synchronization with soft updates (users have the option to update or not)
 - Users' ability to upload their own FPGA bitstream files
 - Remote FPGA programming utility
 - Remote power-cycle of the FPGA board
- FPGA boards schedule, availability and booking management
- User account profile creation as well as email and password management

- Webcam access to remote FPGA evaluation boards
- File management system for users' online directories and hosting space directories
- CLI access to remote servers; for advanced users
- Servers and FPGA boards inventory, usage availability and online/offline status management database
- Hosting and website administrative management

3.3. The End User

The simplicity of the web application is that all that the end user needs is a web browser with web access. The web application is available online through a website. The end users do not need to install any software tools, or plugins to use the web application. It's as simple as accessing a web page, creating an account (for first time users), selecting an FPGA board type, and the end user can start programming their bitstream files immediately into a remote FPGA board. All this, while visually seeing the FPGA board they are working with through a live webcam stream.

The end user is presented with a web page that follows the accustomed visual format and functionality of common websites. The web page includes a registration and a login function, an FPGA board type selection, as well as a file manager that handles selecting sample FPGA bitstream files, or user-defined FPGA design files that could be uploaded.

Chapter 4. System Design Choices

4.1. FPGA Evaluation Board Selection - ZedBoard

The objective of the project is to allow remote access through the cloud to a variety of FPGA development kits or evaluation boards. However, to provide a proof of concept of the project-prototype only one or two FPGA boards would be required.

After examining several FPGA boards and having budgetary constraints, we have selected the ZedBoard Development Board (by Digilent) using the Zynq-7000 ARM/FPGA SoC (by Xilinx). The ZedBoard is able to accommodate a wide range of applications, with features such as:

- Xilinx Zynq-7000 AP SoC XC7Z020-CLG484
- Dual-core ARM Cortex™-A9
- 512 MB DDR3
- 256 MB Quad-SPI Flash
- 4 GB SD card
- Onboard USB-JTAG Programming
- 10/100/1000 Ethernet
- USB OTG 2.0 and USB-UART
- Analog Devices ADAU1761 SigmaDSP® Stereo, Low Power, 96 kHz, 24-Bit Audio Codec
- Analog Devices ADV7511 High Performance 225 MHz HDMI Transmitter (1080p HDMI, 8-bit VGA, 128x32 OLED)
- PS & PL I/O expansion (FMC, Pmod, XADC)

Furthermore, the ZedBoard online community is a very large and active one. This was a crucial resource that was freely available to the project developer; to learn basic FPGA design fundamentals from scratch, as well as to learn how to use the Xilinx design suite tools *Vivado* and *SDK*.

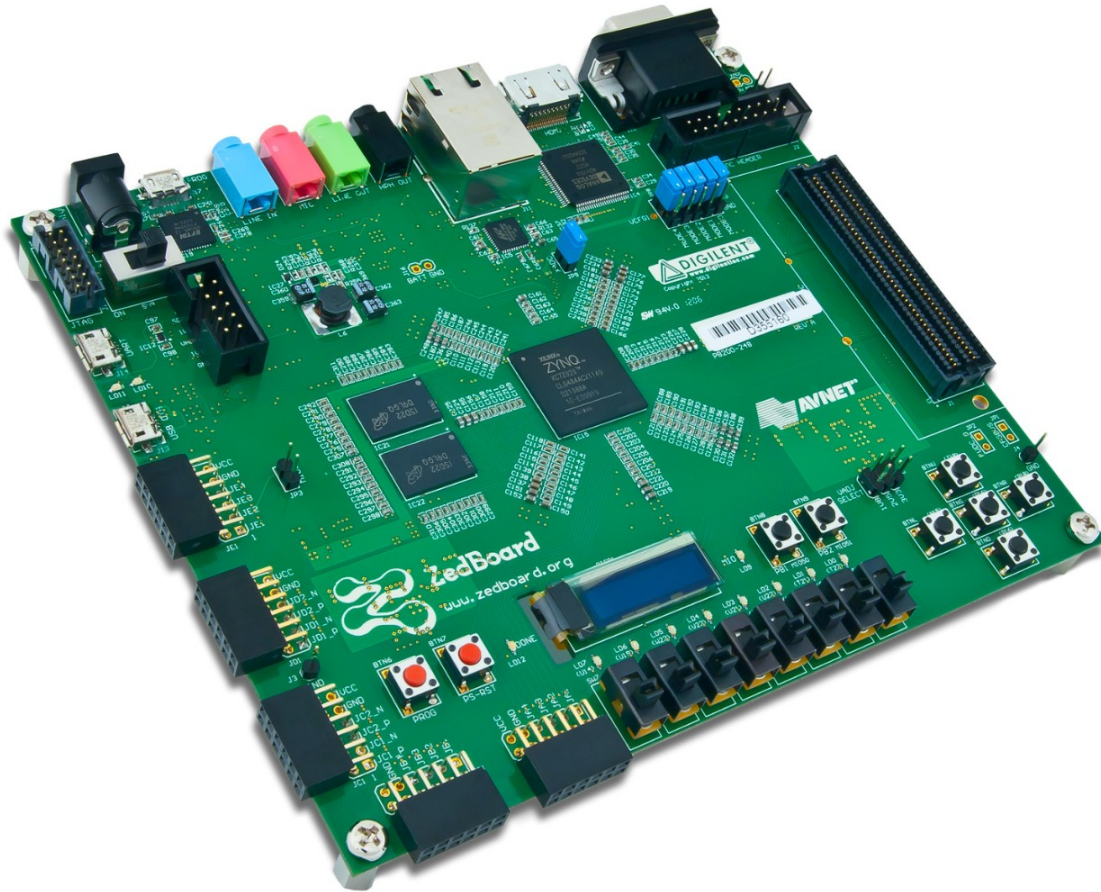


Figure 2 The ZedBoard Featuring the Xilinx Zynq-7000

4.2. Hardware Server OS - Ubuntu

The hardware server OS; which the FPGA boards would be connected to; needed to satisfy a few conditions:

- Be supported by FPGA board vendors. That is the OS is supported by the IDEs, design tools, drivers, etc...
- Be compatible with online hosting services

- Be supported by virtual machine software tools (See section 4.2.1)
- Be easily available and widely supported

The final choice was made to use Ubuntu 14.04.2 LTS as the hardware servers' OS.

4.2.1. Virtual Machine Software – VMware

For development purposes and to keep costs down, a decision was made to not purchase a server, or a PC-based machine, but rather to use a virtual machine setup that would run Ubuntu 14.04.2 LTS. A virtual machine setup would provide the benefit of portability and a very flexible bring up. The choice was made to use VMware Workstation Player² virtual machine solutions. Alternatively, one could also use Oracle's VM VirtualBox.

4.3. Web Cam

When a user has established a connection with a remote server; to interact with an FPGA evaluation board, the web application would launch a live streaming feed that would show the user the FPGA board they are interacting with. For example, if the user uploads and programs the FPGA with a bitstream file that say makes the LEDs on the FPGA board blink in a particular pattern, the user would then be able to see the LEDs blink live on the web cam live feed.

To serve this purpose, simple off the shelf web cams are used and connected to each hardware server with an FPGA board. The web cam software would be running on the remote hardware server (locally in the hardware server). Linux's Motion is an open source software that would be used to manage the web cam feed.

² VMware Workstation Player is available for free only for non-commercial, non-production environments.

4.4. Web Scripting Language Selection - PHP

To manage remote access to an FPGA evaluation board, a web application needed to be developed from the ground up to provide such a service. Having very minimal knowledge in current web development design tools and scripting languages; picking a scripting language for the development of the web application was no easy task. Current popular web scripting languages and programming languages were looked at, such as: PHP, Ruby, JavaScript and Python. Online resources and having active online communities would be an important element for self-learning a new scripting language.

After looking at the most fundamental back-end functionalities of our web application; PHP seemed to offer a more suitable range of libraries that would make our back-end development smoother. And while it wasn't obvious at the time, PHP would ultimately be a good choice for the front-end as well.

Initially, the overall scope of the web application was heavily underestimated! With no understanding of current web applications' complexities, the path taken into developing the web application for this project would have been a more suitable approach during the late 90s, with plain and static websites! Initial scripting was done in PHP, using a direct approach. That is; if a given functionality was needed, the coding was done from scratch to achieve that specific purpose. However, as the web application grew, and the development started to shift to the front-end, it became clear that such an outdated methodology would not be enough. The complexity rose, but the web application development method could not keep up! Scalability and expandability needed to be taken into consideration.

4.4.1. Frameworks - Laravel

Frameworks vary based on what one is trying to achieve. Having already invested time in PHP, naturally, PHP-based frameworks were looked at first. There are PHP frameworks which can handle things like database abstraction, passwords' authentication, sending email, and interacting with the web server. Such frameworks would be very suitable to our project-prototype. The selection was made to use Laravel; a PHP-based framework.

Frameworks are like libraries that try to add another layer that sits on top of a given programming language to provide efficiency, reusability and most importantly a level of standardisation to the code. At first, frameworks may seem pointless as it is much easier to just build something from scratch using the base language. But it would soon become apparent in a large application or multi-team environments that it is very important that some level of code standardisation is adhered to. Without a framework, larger projects will often bloat and head down paths of no return where scalability and flexibility can no longer be achieved. The code would become difficult to comprehend and would have numerous dependencies where modifying something could do who knows what! Frameworks try to abstract a lot of low level logic and methodologies, and provide a modular way of programming, keeping things nice and separated, allowing the developer to focus on the intended objective of the application rather than the low level technical details.

At first, the concept was difficult to grasp. Using Laravel seemed to be an overkill with our early back-end web application functionalities. However, as the project progressed towards the front-end aspect of the web application; such as: Menus, UI, user profile creation, password encryption, storage and authentication, DB management, the overhead that comes with the framework was put to good use.

4.4.2. IDE Tools - Homestead

One great advantage of using Laravel is that it comes with its own IDE. Homestead is the official, pre-packaged Vagrant "box" that provides the developers with a development environment without requiring one to install PHP, a virtual machine, a web server, or any other server software on one's local machine. Homestead is an all-in-one Laravel PHP IDE. This allows for development and testing of the web application on a local machine then do the testing in a local virtual machine setup, before pushing the code changes to the hosting website.

4.5. Web Hosting Services - AWS

While the web application is developed and tested in a local virtual machine setup, the application must be mirrored to an online hosting service to be accessible from the World Wide Web. Several online hosting services were looked at with the

following two elements in mind: a cost-effective solution and a hosting server that was Linux-based to be able to run our web application. Amazon Web Services were selected with their EC2 solution. It offered a 12 month free tier solution and a pay-as-you-go afterwards.

AWS' solution offers 750 hours of Amazon EC2 Linux or RHEL or SLES t2.micro instance usage (1 GiB of memory and 32-bit and 64-bit platform support) [5].

Another web hosting application which was considered is Heroku. However, after initial research, it was discovered that Heroku didn't support some libraries required for our web application. Also, with Heroku it would not possible to SSH into the hosting server to debug issues or do custom installations and configurations.

4.6. Software Version Control System and Online Repository – BitBucket

As with any software based project, there would be a need to keep track of code changes, to have the ability to roll back, and to easily deploy the source code amongst multiple users and/or platforms. Again, keeping costs down, several free version software tracking tools were looked at. Privacy was another element. The project source code was to not be made publicly available during the development stage.

The choice was made to use BitBucket by Atlassian, which uses GIT as the version control system.

4.7. Web Based Terminal Emulator – Shell in a Box

One part of the web application is to provide a CLI utility which would provide direct yet limited access to the remote server of which the user is connected to; for their FPGA board interaction. While the web application does provide the UI required to program the FPGA with a set of bitstream files, terminal access to the hardware server in question could be of great benefit to advanced users, who have the know-how to make configuration changes on the fly while they are testing their own FPGA design files.

To allow such terminal access through the web, several web based terminal emulators were looked at. The selection came down to two options: Shell in a Box and AjaxTerm. The most sought out feature was that the terminal emulator would not require the end user to install any plugins to be used.

Both AjaxTerm and Shell in a Box allow access to a remote server using an emulated terminal in a JavaScript-enabled web browser. Shell in a Box is written in C; AjaxTerm is written in Python. AjaxTerm works slightly differently from Shell in a Box. While Shell in a Box runs a full terminal emulator written in JavaScript on the browser, AjaxTerm does terminal emulation on the server side and send lower-level screen updates to the browser.

Both tools were tested. Ultimately, Shell in a Box was selected. During testing, Shell in a Box provided a much smoother terminal interaction with minimal lag. Another advantage to having chosen Shell in a Box is that it has some unofficial forks over at GitHub which have come handy to fix some bugs specific to the web application.

Chapter 5. System Implementation: FPGA Board Component

5.1. ZedBoard Setup and Configuration

The ZedBoard purchased came with all the necessary software tools and licenses needed. Installation of such tools is simple and straightforward. Consult Appendix I; for more details about Xilinx’s Vivado Design Suite software installation.

The first step into setting up the ZedBoard was to follow the ZedBoard *Getting Started Guide* [6]. This would ensure that a given PC or server meets the minimum requirements to communicate with a ZedBoard, and that all drivers are functional at a minimal level.

The next step was to go through the ZedBoard *Hardware User’s Guide* and the *Configuration and Booting Guide* [7] - [8]. These two guides help the user get familiar with the various components and peripherals available, along with various booting and configuration options on the ZedBoard, which would depend on the application at hand. For our application use case; either option shown in Table 1 would work.

Table 1 ZedBoard Boot Mode Jumper Settings

Jumper	Option 1	Option 2
JP7	GND	GND
JP8	GND	GND
JP9	GND	3V3
JP10	GND	3V3
JP11	GND	GND

The three guides mentioned above are the basic required reading before venturing into the first FPGA design: LED Binary Counter.

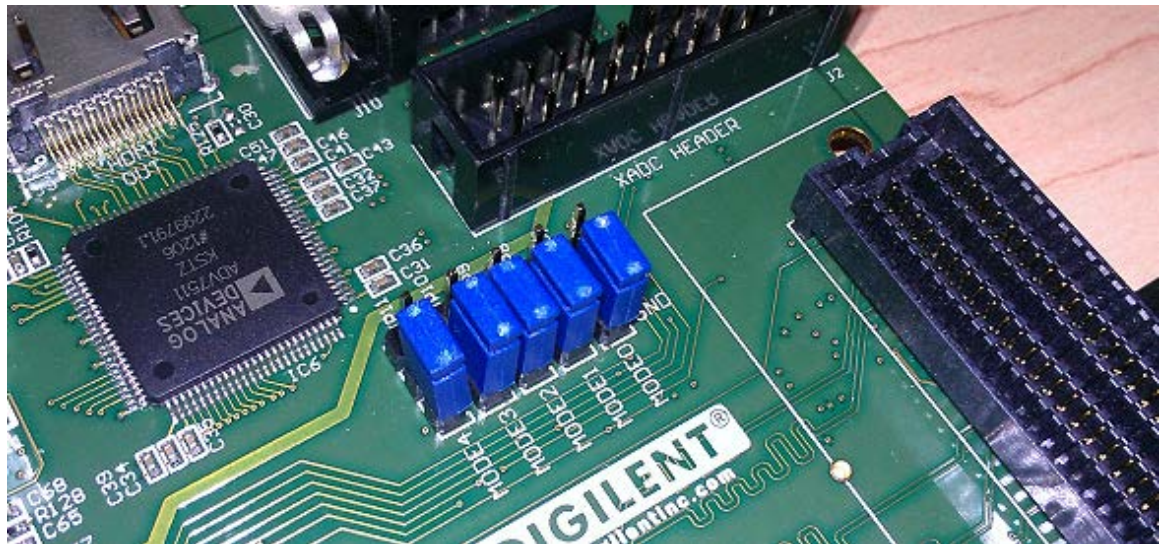


Figure 3 ZedBoard Boot Mode Jumper Settings - Option 1

5.2. LED Binary Counter on the ZedBoard Using the Zynq-7000 FPGA

A few basic Zynq-based FPGA designs were created to provide some sample designs for development, as well as for demonstration purposes. That is, while developing the web application to remotely access the FPGA board in question, there was a need to have at least two FPGA sample designs available; to verify that commands such as programming the bitstream files into the FPGA is indeed working remotely through the web application. The first such of sample FPGA design files is a simple LED binary counter.

Sections 5.2.1 through 5.2.4 describe the process of creating the hardware and the software application to run a simple LED binary counter application on the FPGA board.

5.2.1. Step 1: Creating a New Project in Vivado

1. Launch the Vivado IDE and select Create New Project
2. Enter a project name of your choice in the Project Name field and choose a Project Location, and click Next

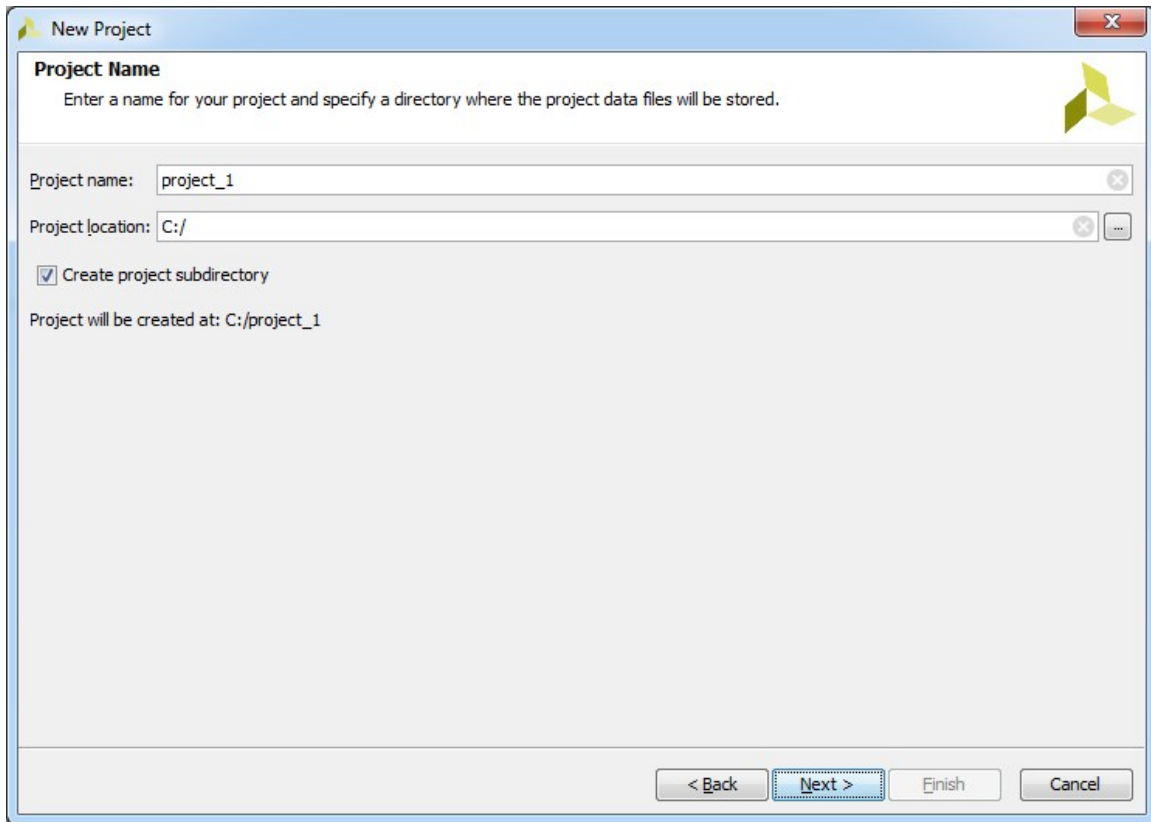


Figure 4 Project Name and Location in Vivado

3. Select RTL Project option in the Project Type form, and click Next
4. In the Add Sources window, select VHDL as the Target Language and Simulator Language. Click Next
5. There won't be a need to add files, IPs or constraints so click Next
6. In the Default Part windows, click the Boards icon and choose ZedBoard Zynq Evaluation and Development Kit, and click Next then Finish

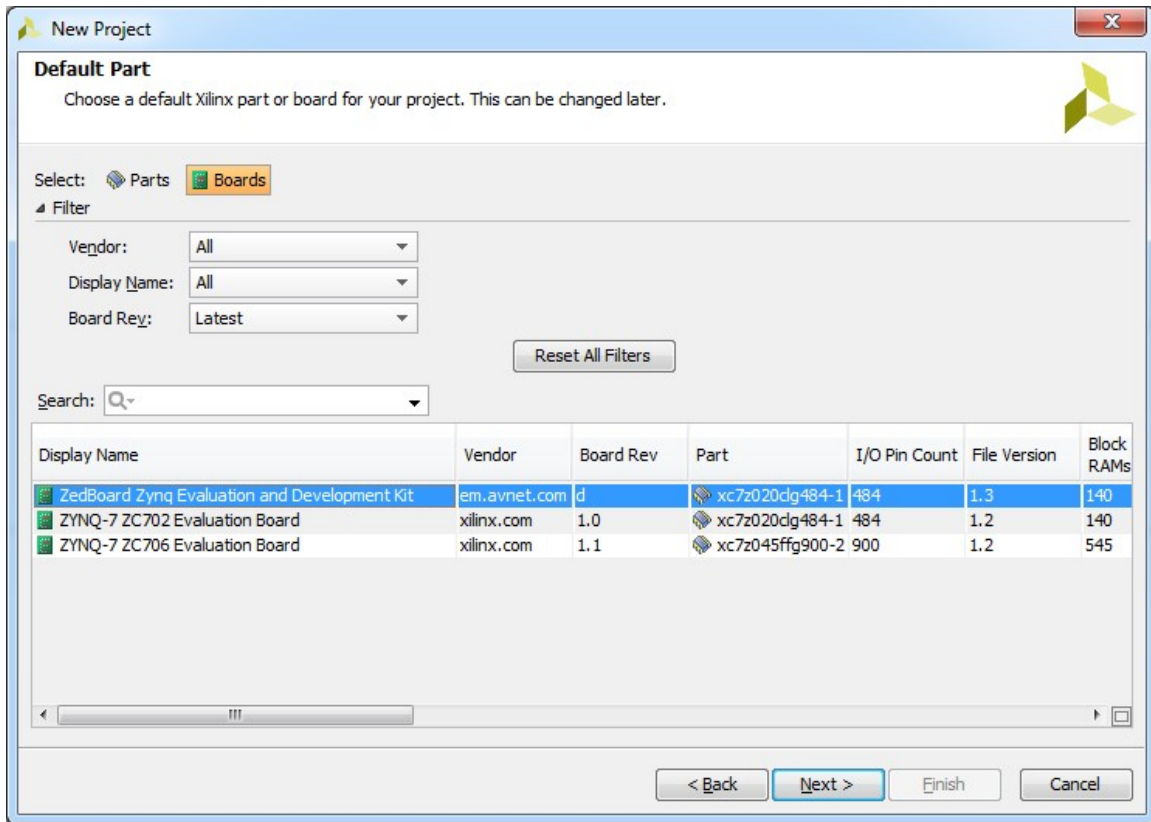


Figure 5 Board Selection in Vivado

5.2.2. Step 2: Creating a Block Design

To control the LEDs from the PS, one needs to create a block design and add a Zynq SOC block to it, then add an AXI GPIO IP core which would be used by the Zynq SOC to control the LEDs.

1. In the Flow Navigator expand the IP Integrator and click Create Block Design. A Create Design window will pop up, keep the same name design_1 and click OK

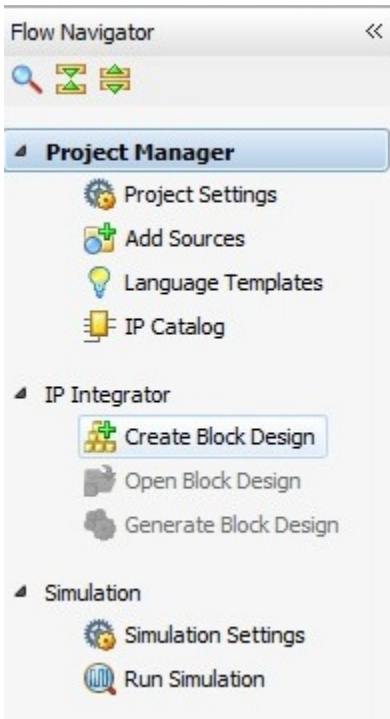


Figure 6 Create Block Design in Vivado

2. After the Diagram window opens, click on the Add IP icon from the left panel of the Diagram page to add IP blocks. Alternatively, one could right-click on any empty area in the Diagram section and choose Add IP

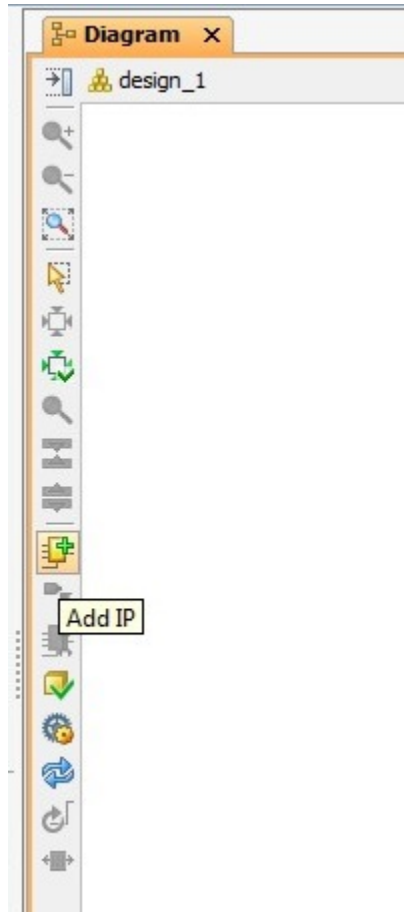


Figure 7 Add IP in Vivado

3. In the search bar type Zynq, and choose (double-click) ZYNQ7 Processor System, this will add a Zynq Processing System IP(the ARM Cortex A9 processor)

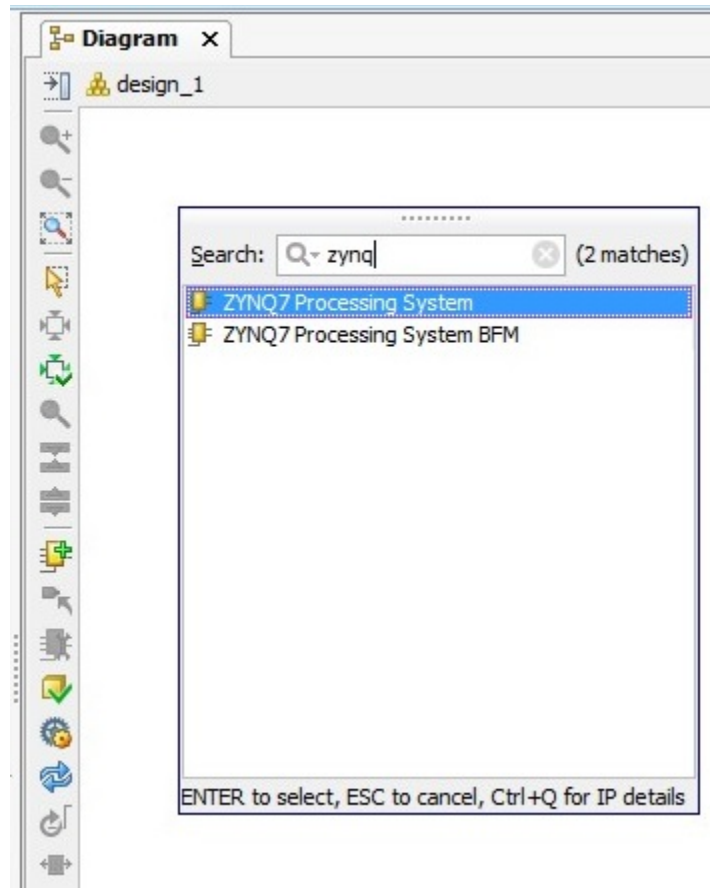


Figure 8 Add Zynq7 Processing System IP in Vivado

4. Click on Run Block Automation. This would configure the Zynq IP according to the default settings per the selected board (The ZedBoard). This is called Design Assistance, and it appears whenever there is a possibility that Vivado could automate some design steps. Keep the default settings and click OK

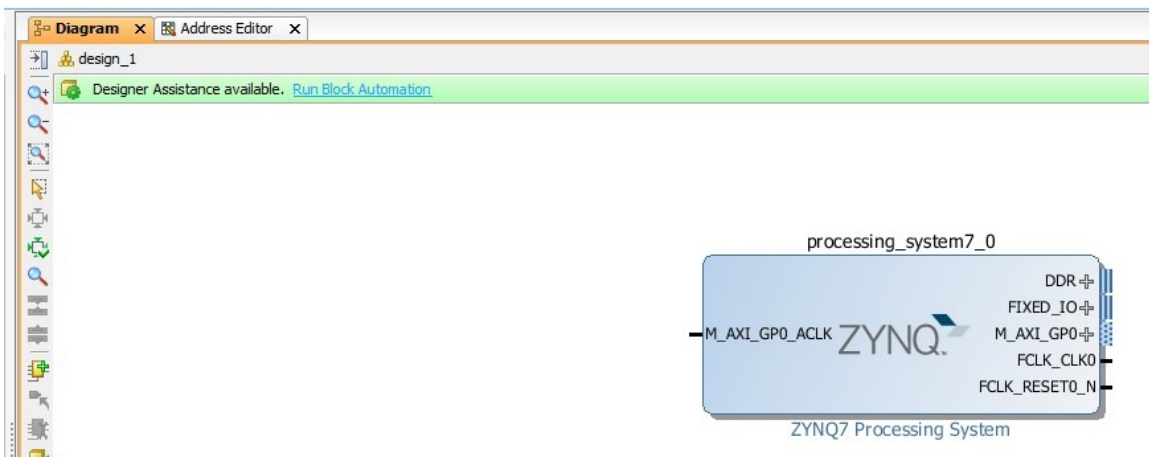


Figure 9 Run Block Automation in Vivado

- Again click on the Add IP icon and this time; in the search bar type GPIO, and choose the AXI GPIO. The AXI GPIO IP should appear in the Diagram

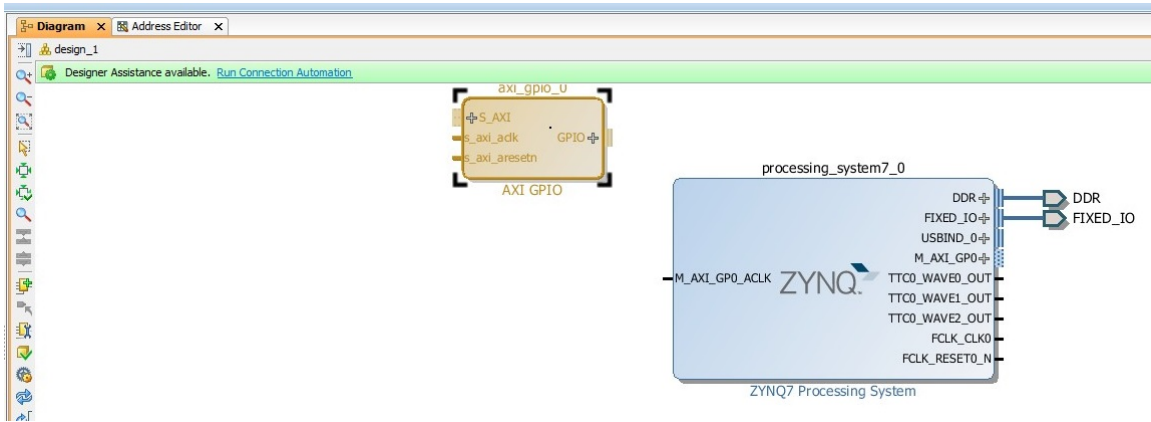


Figure 10 Added AXI GPIO IP in Vivado

- Double click on the AXI GPIO IP, under Board Interface choose leds 8bits and click OK

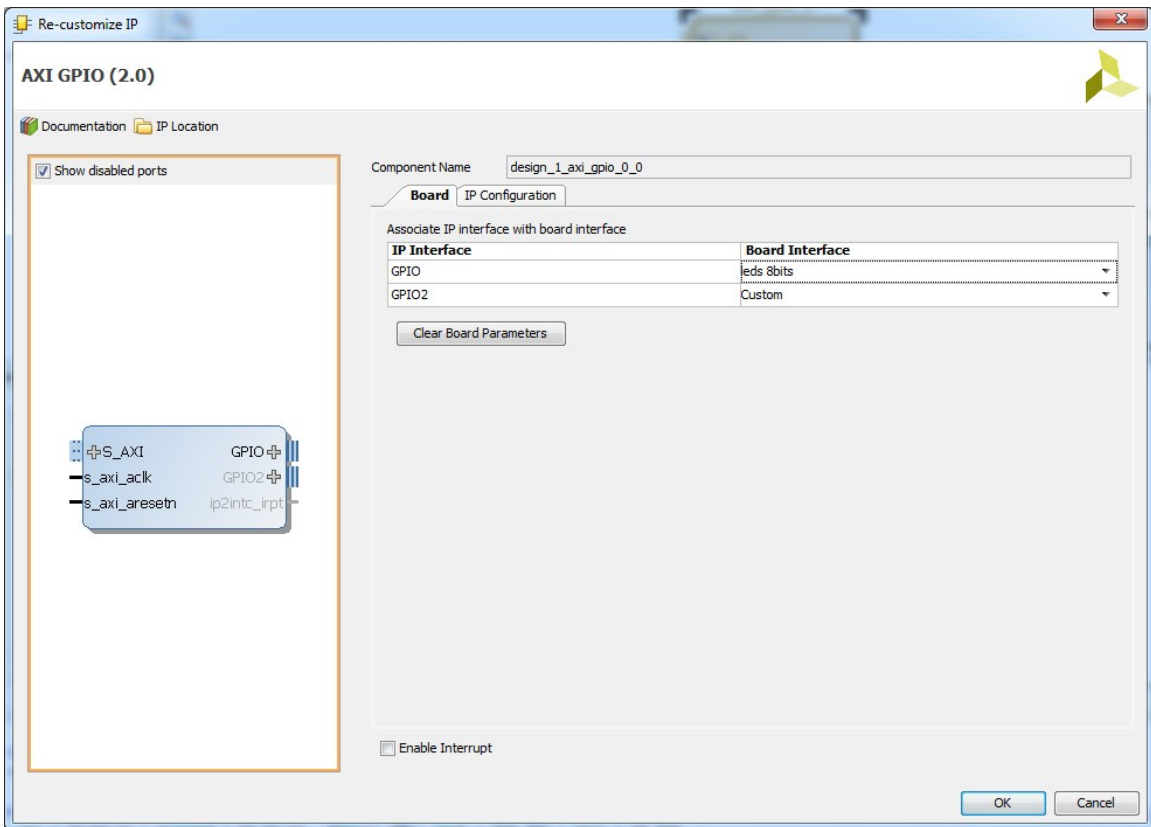


Figure 11 Connect the AXI GPIO IP to the Board LED 8 Bits in Vivado

7. Now the design assistance will again show the option to run Connection Automation, Click on it and choose All Automation then click ok. Here, one is accepting the suggestion of connecting the AXI GPIO IP to the Zynq. Vivado will add 2 IPs, these IPs are the processor system reset and AXI interconnect
 - a. The processor system reset provides the reset signals to all peripherals and interconnects in the PL side of the Zynq according to reset signal given from the Zynq PS
 - b. The AXI interconnect is responsible for creating an interface between the Zynq PS master interface GP port and the AXI GPIO IP
8. Click on the Regenerate Layout icon to rearrange the blocks. This is purely a visual rearrangement to fit the window/screen

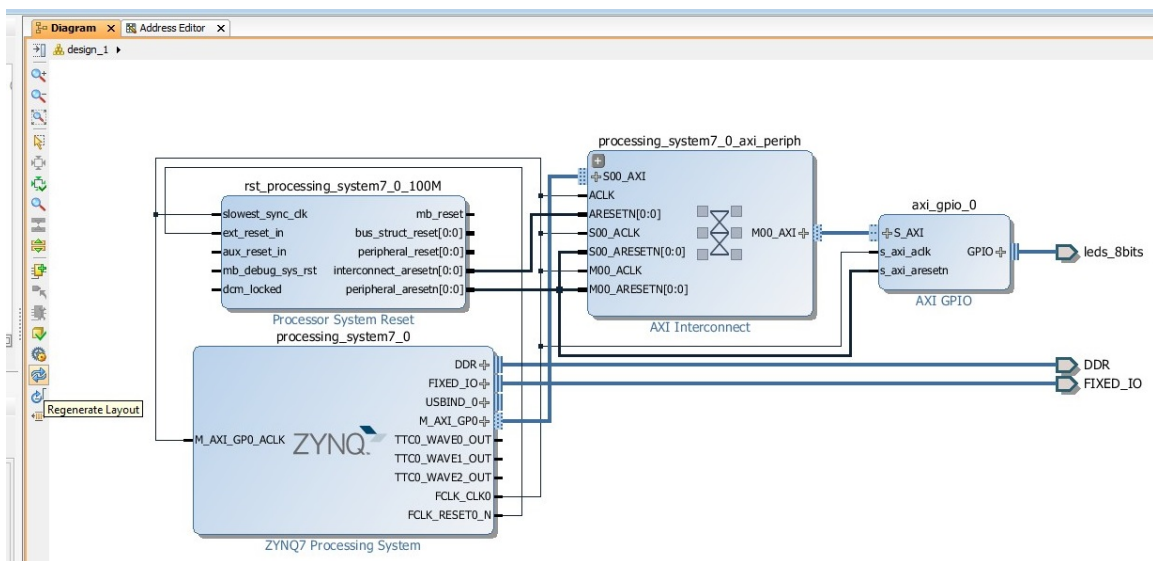


Figure 12 Regenerate Layout (I.E. Refresh) in Vivado

9. In the Design window, click on the Sources tab, right-click on the design_1 block design under Design Sources folder and choose create HDL wrapper, this options creates a top level VHDL file for the block design which can be used to generate the bitstream file

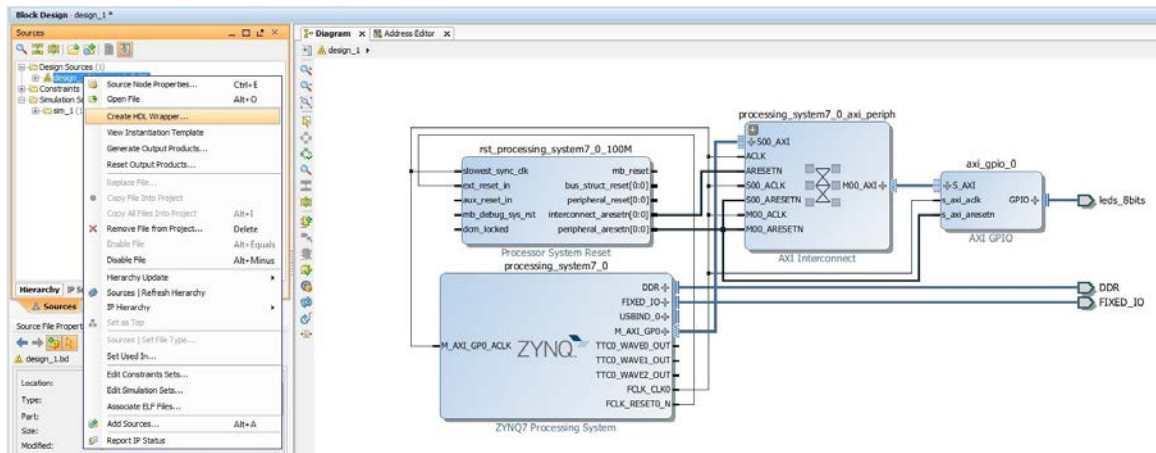


Figure 13 Create HDL Wrapper in Vivado

10. Keep the default setting Let Vivado manage wrapper and auto-update and click OK

- c. Note: Click on the Address Editor, one shall find the processing system connected to the AXI GPIO IP, with an automatically given offset address. This address will be used by the Zynq processor to communicate with the AXI GPIO IP which is connected to the LEDs. In our example the offset address is 0x41200000

11. In the Flow Navigator, under Program and Debug click on Generate Bitstream. Alternatively, from the top menu Flow → Generate Bitstream. During this step the tool creates a bit file for programming the PL of the Zynq. Accept to save the project

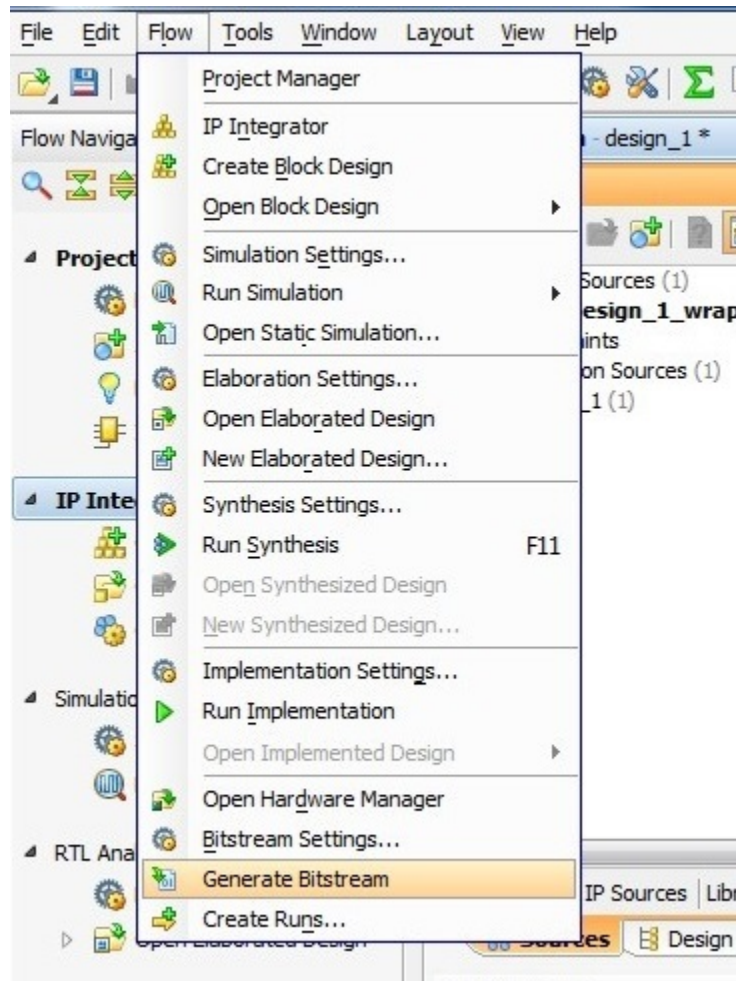


Figure 14 Generate Bitstream

12. Click Yes to No Implementation Results Available. This would synthesize the design. Depending on the PC's available resources and specs, the process should take a few minutes, potentially longer on slower PCs
13. Bit Generation Complete message should pop up, close/cancel it then go to File → Export → Export Hardware. Check the Include bitstream and click ok. This exports the hardware files generated to the SDK

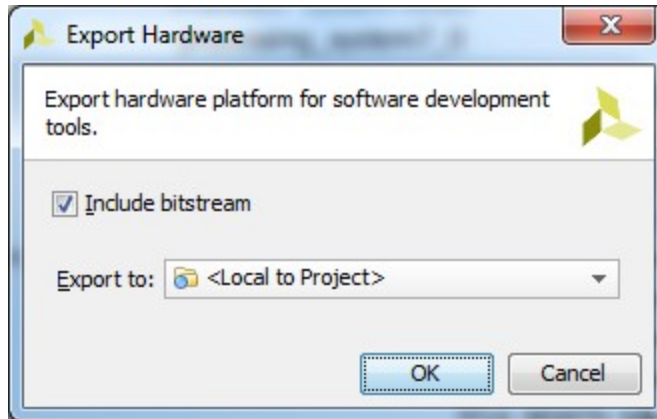


Figure 15 Export Hardware And Bitstream in Vivado

5.2.3. Step 3: Writing the Software Application

During this step one would export the hardware files to the SDK to create a C project for controlling the LEDs from the Zynq PS.

1. In Vivado go to File → Launch SDK, then click OK. The SDK will open and in the Project Explorer window one could see the exported hardware files. In the SDK go to Files → New → board support package then click finish. This creates a board support package with all the necessary functions for driving the hardware that was designed in Vivado; in the previous steps

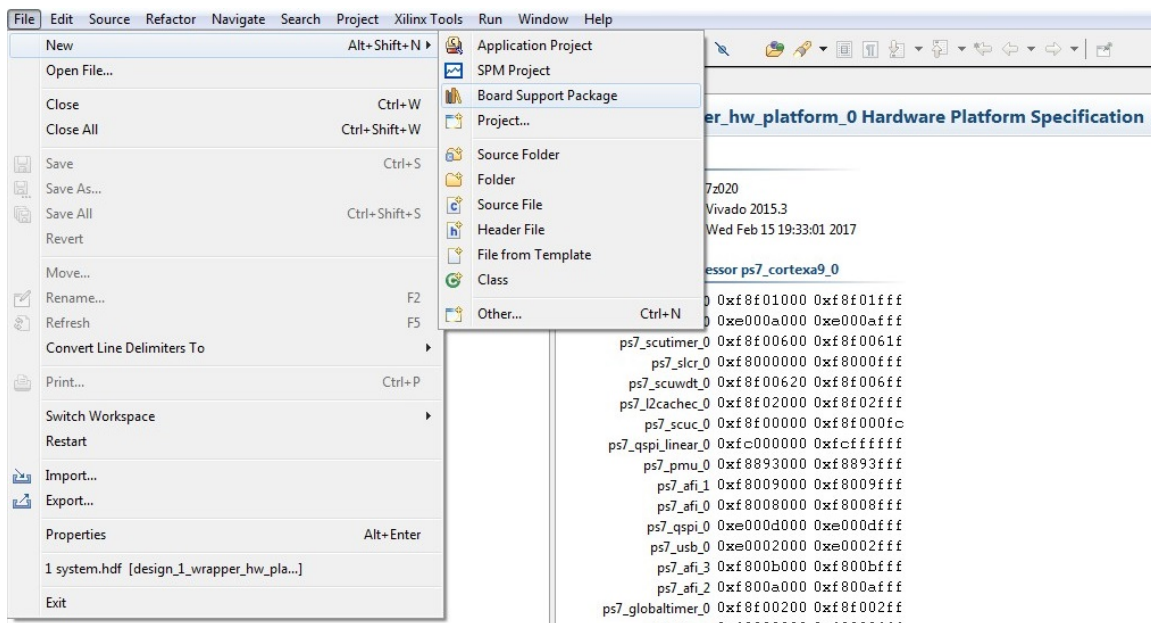


Figure 16 Create a New Board Support Package in SDK

2. The Board Support Package Settings window would pop up, keep all default settings and click Finish then OK
3. Create an application project by going to File → New → Application Project. Enter a name in the project name field and under the Board Support Package choose Use existing then click Next

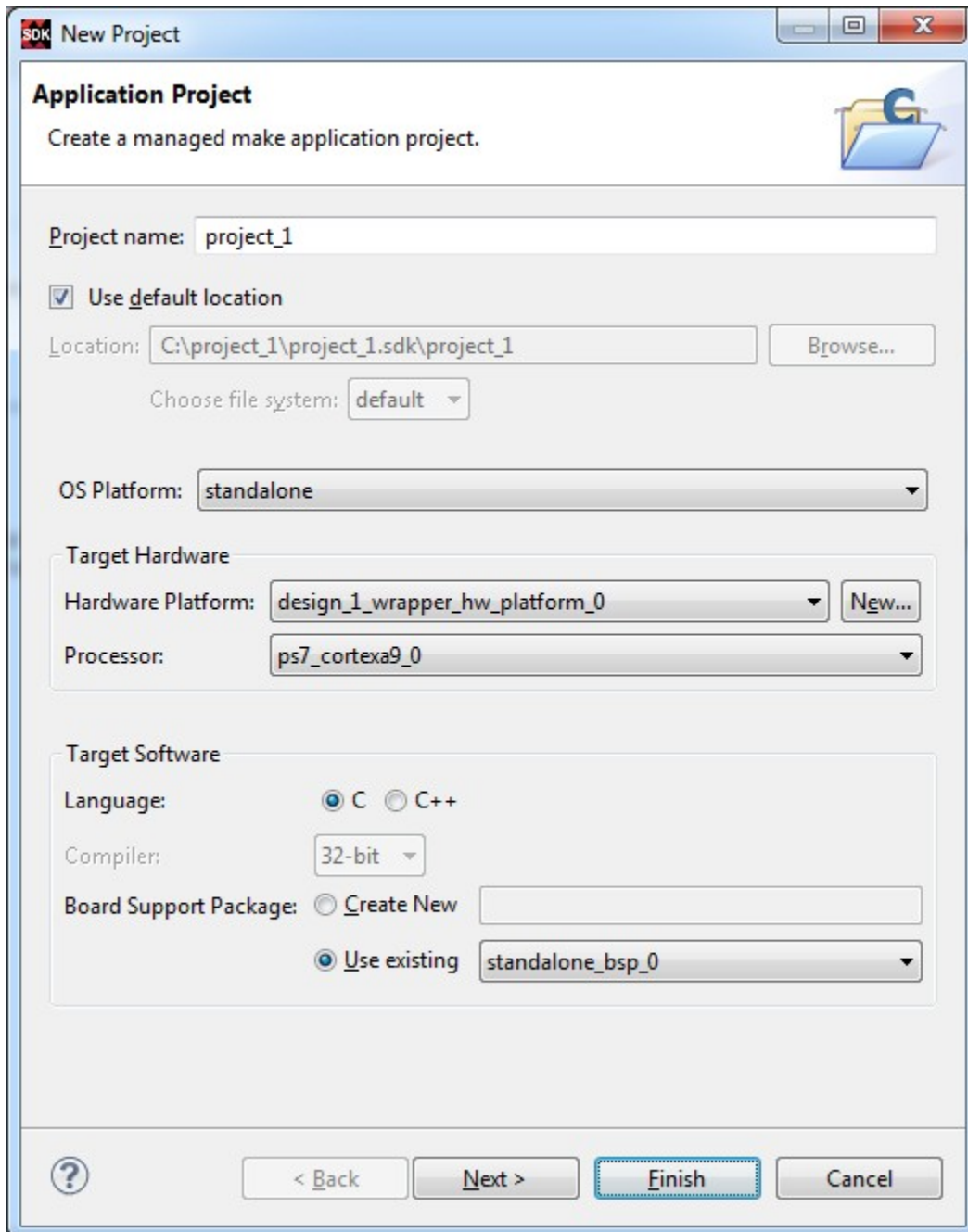


Figure 17 Create an Application Project in SDK

4. Choose Hello World from the Available Templates then click Finish

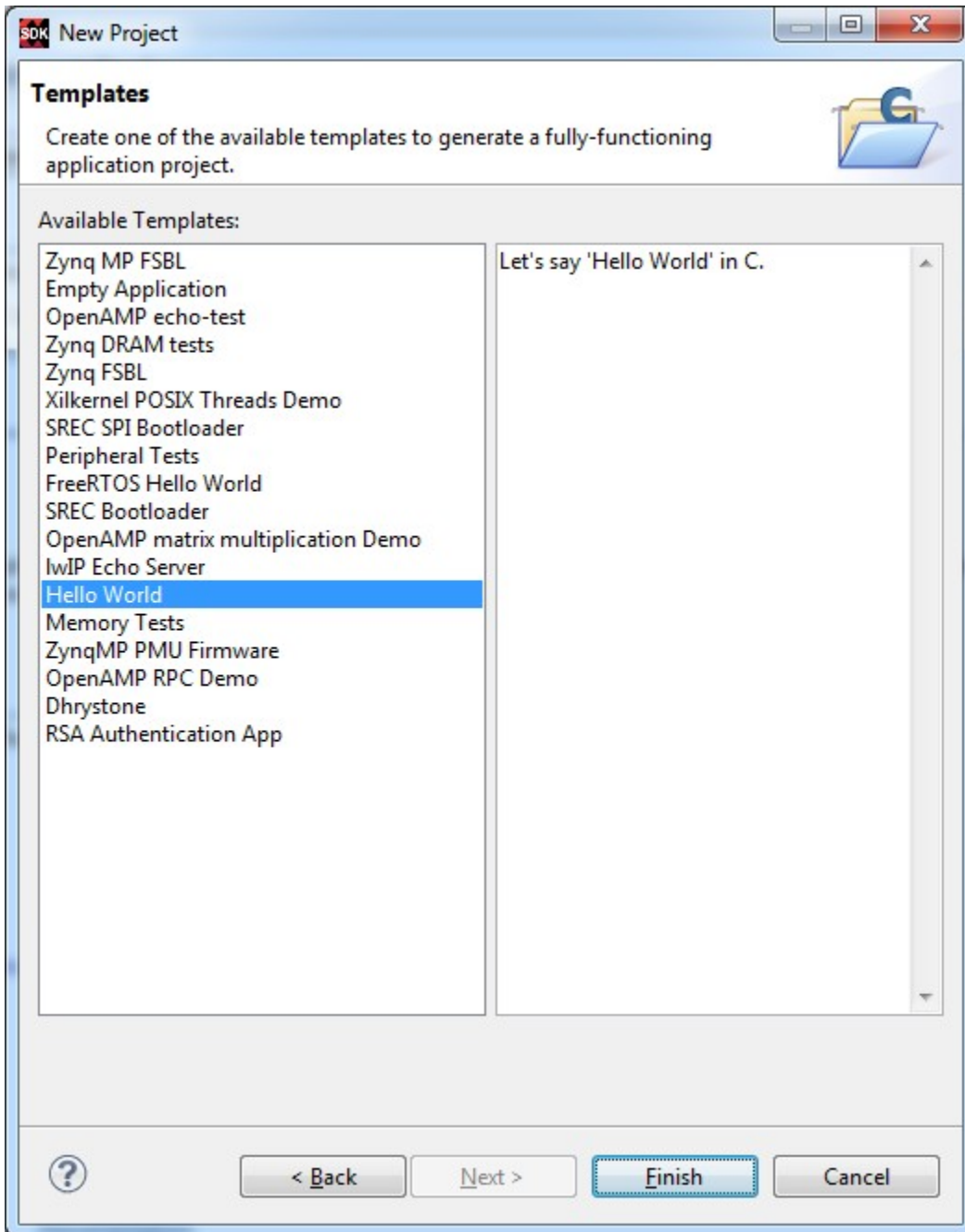
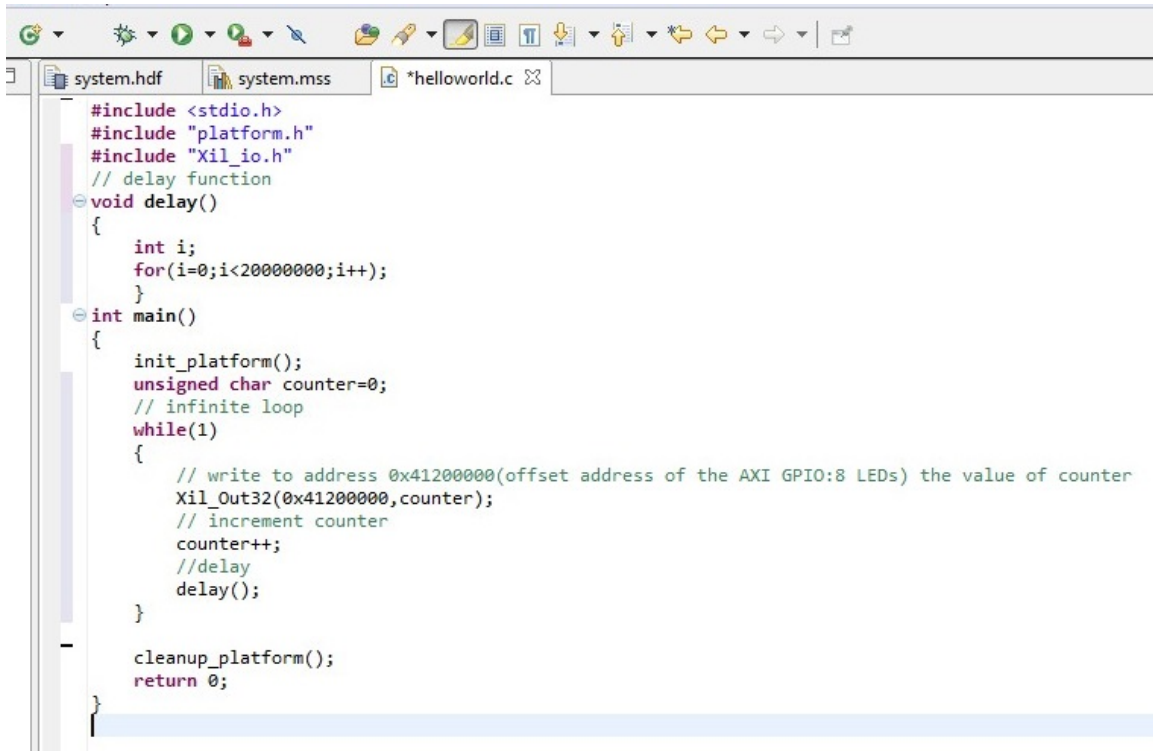


Figure 18 Select Hello World Template in SDK

5. In the Project Explorer left panel, expand the src folder under the project_1 folder and double click on Helloworld.c. The source code will be visible in the main window. Replace the code within it with the following code:

```
1. #include <stdio.h>
2. #include "platform.h"
3. #include "Xil_io.h"
4. // delay function
5. void delay()
6. {
7.     int i;
8.     for(i=0;i<20000000;i++);
9. }
10. int main()
11. {
12.     init_platform();
13.     unsigned char counter=0;
14.     // infinite loop
15.     while(1)
16.     {
17.         // write to address 0x41200000(offset address
of the AXI GPIO:8 LEDs) the value of counter
18.         Xil_Out32(0x41200000,counter);
19.         // increment counter
20.         counter++;
21.         //delay
22.         delay();
23.     }
24.
25.     cleanup_platform();
26.     return 0;
27. }
```



```
#include <stdio.h>
#include "platform.h"
#include "Xil_io.h"
// delay function
void delay()
{
    int i;
    for(i=0;i<20000000;i++);
}
int main()
{
    init_platform();
    unsigned char counter=0;
    // infinite loop
    while(1)
    {
        // write to address 0x41200000(offset address of the AXI GPIO:8 LEDs) the value of counter
        Xil_Out32(0x41200000,counter);
        // increment counter
        counter++;
        //delay
        delay();
    }

    cleanup_platform();
    return 0;
}
```

Figure 19 LED Binary Counter C code in SDK

5.2.4. Step 4: Programming the ZedBoard and Running the C Application

1. In the SDK click on the program FPGA icon then click program; to program the Zynq PL with the bitstream

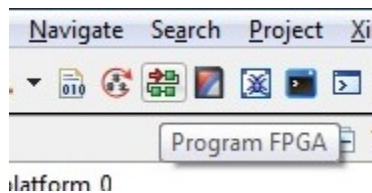


Figure 20 Program FPGA in SDK

2. Right-click on the project_1 folder in the Project Explorer panel then go to Run As → 4 Launch on Hardware (GDB). The application should run on the board and the LEDs should show the binary values of the counter

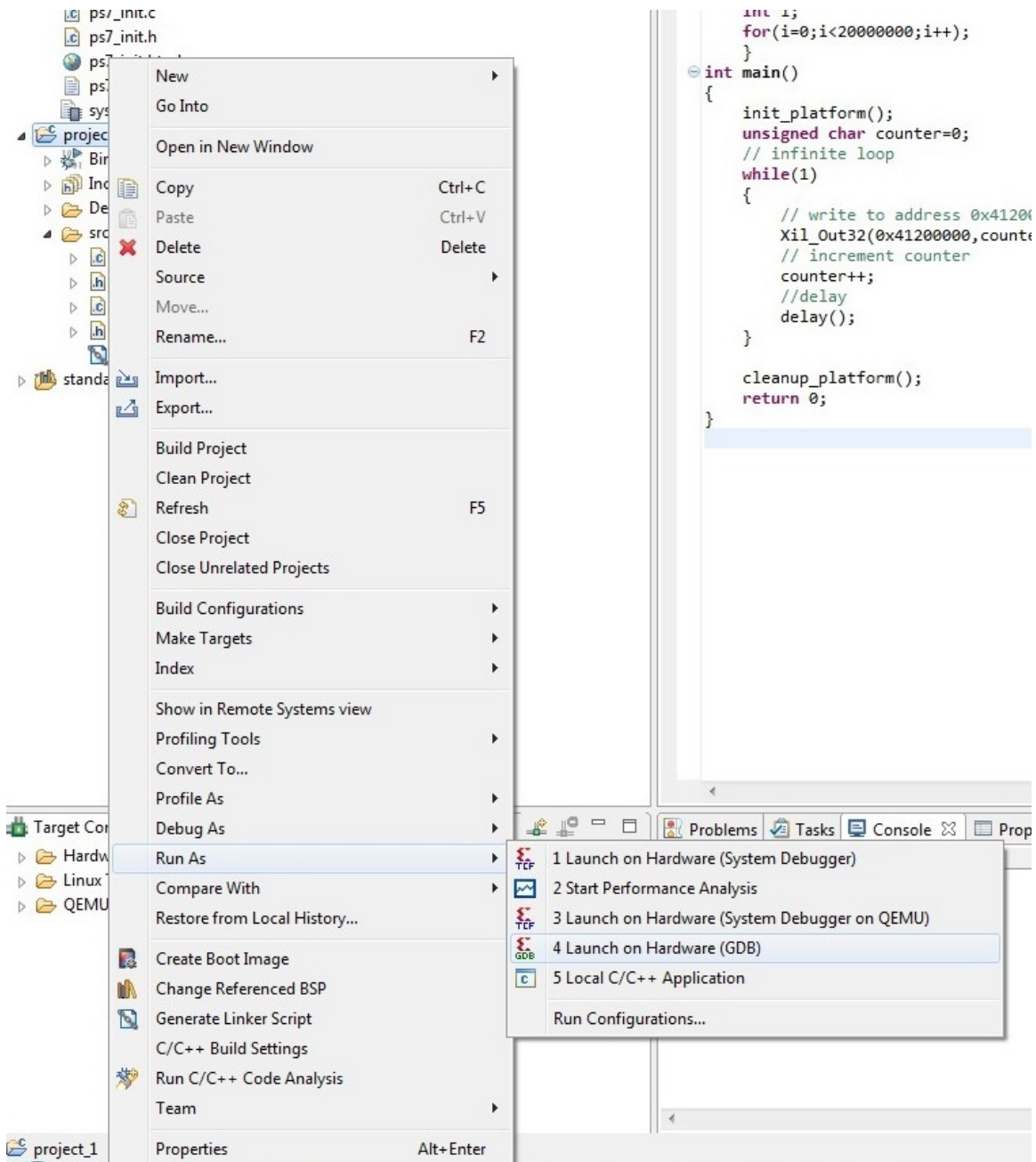


Figure 21 Running the C Application on the Hardware in SDK

5.3. Other FPGA Designs

Full instructions with complete diagrams for the LED Binary Counter as well as the remaining FPGA designs that were developed and tested to be used as a demo for the project-prototype could be found in Appendices D through H.

Chapter 6. System Implementation: Web Application

6.1. Tree View of the Web Application

As mentioned previously, the web application manages the front end (end user interface and interaction) as well as the back end (Interaction with the remote hardware server and the FPGA board in question).

The folder structure view shown in Figure 22 provides the main overview of the web application source code structure:

- `../app`: The web application top level core code definition. This is similar to class definitions
- `../config`: All the web application's configuration settings are stored in this folder. Example: remote connection database settings
- `../public`: Contains the `index.php` (I.E. the homepage, or pointer to the homepage) which is the entry point for all requests entering the web application
- `../public/assets`: JavaScript, CSS and images used throughout the web application
- `../public/filemanager`: Source code for the File Manager plugin as well as storage location for user uploaded content
- `../serversD`: Contains daemon source code for the hardware servers' online availability status check utility
- `../storage`: Contains event logs, cache, sessions, etc...
- `../composer.json`: Dependency file manager. Composer will manage the dependencies required on a project by project basis. This means that Composer will pull in all the required libraries, dependencies and manage them all in one place

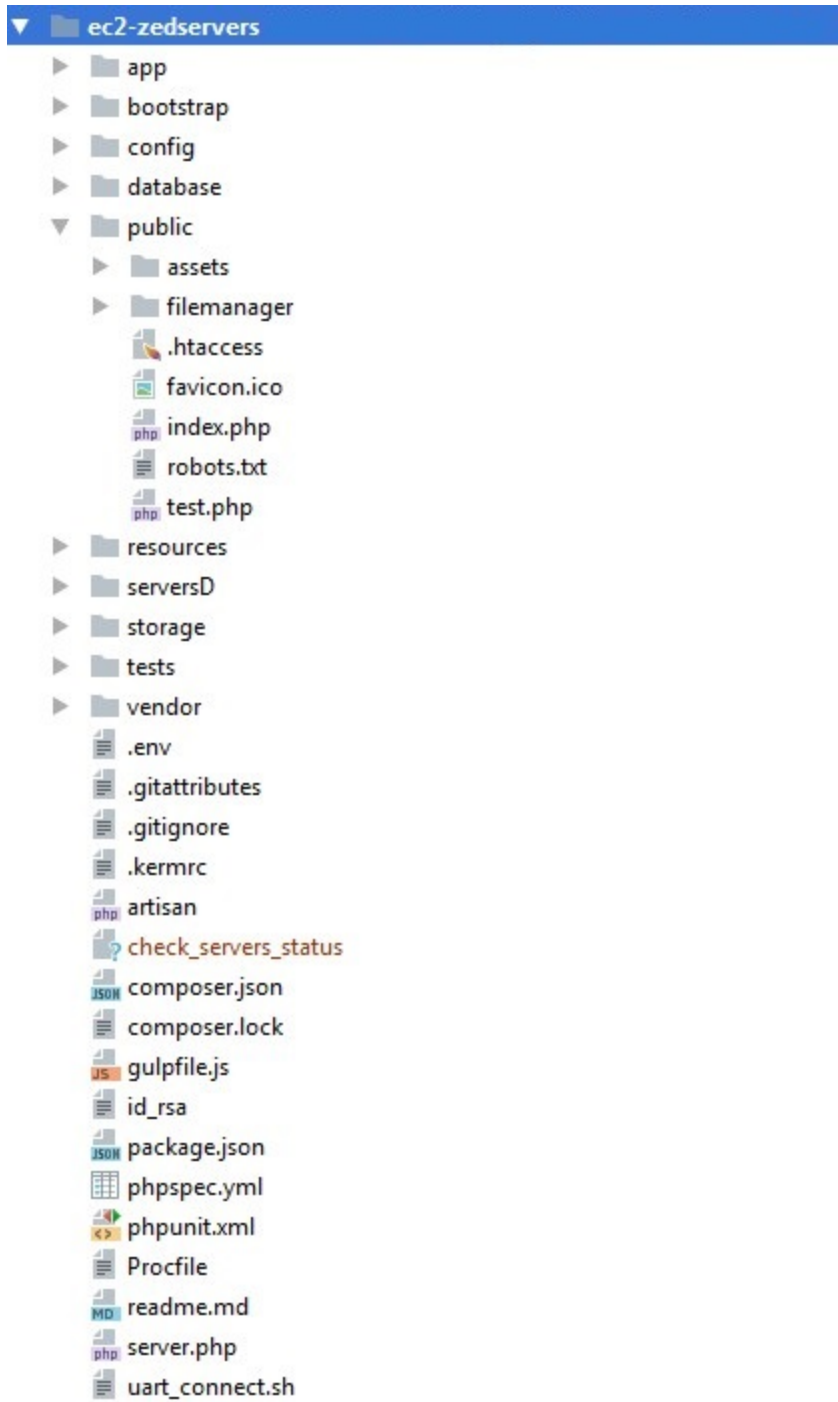


Figure 22 The Web Application's Structure Main Overview

6.2. Model-View-Controller Code Structure

Model View Controller is a software architecture that separates the application logic from the rest of the user interface. This is achieved by separating the application into three parts: the model, the view, and the controller.

The model manages essential behaviors and data of the application. It can respond to requests for information, respond to instructions to change the state of its information, or notify observers in event-driven systems when information changes. This could be a database. It is the data and data-management aspect of the application.

The controller receives user input and makes calls to model objects and the view to perform appropriate actions.

The view is the user interface element of the application. It'll render data from the model into a form that is suitable for the user interface.

All in all, these three components work together to create the three basic components of MVC.

6.2.1. The Models

Figure 23 shows the models used in our web application:

- `../app/Http/Models/Auth_model.php`: DB model for user authentication
- `../app/Http/Models/Homes.php`: DB model for server and web application settings
- `../app/Http/Models/Users.php`: DB model for user settings
- `../app/Http/Reservation.php`: DB model for FPGA board online booking
- `../app/Http/Server.php`: DB model for amount of servers available and their online/offline status. Our project-prototype has only 1-2 virtual servers. However, the web application supports as many servers as could be needed

- ../app/Http/Tab.php: DB model for tabs open in the web based emulated terminal (CLI access to the remote server through Shell in a Box)
- ../app/Http/User.php: Additional DB model for user passwords

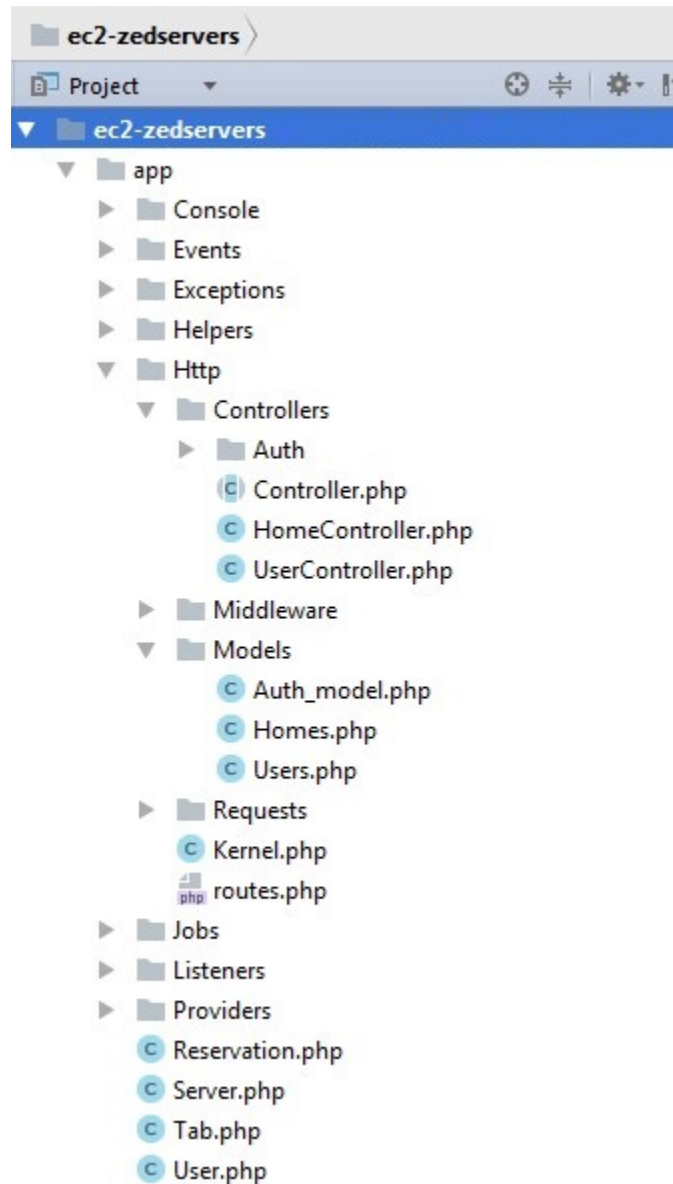


Figure 23 Models, Controllers, and Helpers Source Code Tree

6.2.2. The Controllers

In our web application Laravel's controllers are used to perform all page-based actions or user-based actions. Page-based actions are actions such as loading a frame

or refreshing text displayed on the web page. User-based actions are actions such user-login, user-registration, reset password request, etc....

The folder structure in Figure 23 shows the two main controllers used in the web application:

- `../app/Http/Controllers/HomeController.php`: This is the controller for all the web application's actions. It contains all the code specific to the web application's actions
- `../app/Http/Controllers/UserController.php`: This is the controller for all the user's actions. It contains all the code related to the user's actions

6.2.3. The View

The third component of the MVC structure is the view. In essence, this is what the user sees on any given website. It's the user interface; the web page.

Figure 24 shows the various UI (I.E. web pages) that the end user could interact with on the web application.

- `../app/resources/views/admin/admin_blade.php`: Administrative settings page. Accessible only by users given administrative privileges
- `../app/resources/views/admin/rserver.blade.php`: The remote hardware server page. This is the workshop web page! That is, the interactive web page that allows the connected user to interact with a remote FPGA evaluation board
- `../app/resources/views/admin/admin_menu.blade.php`: Web site navigation menu for administrative users
- `../app/resources/views/admin/guest_menu.blade.php`: Web site navigation menu for standard users
- `../app/resources/views/admin/footer.blade.php`: Footer content of any given web page

- `../app/resources/views/admin/header.blade.php`: Header content of any given web page
- `../app/resources/views/admin/changePassword.blade.php`: Change user's password form web page
- `../app/resources/views/admin/login.blade.php`: User login web page
- `../app/resources/views/admin/profile.blade.php`: User profile (I.E. Dashboard) web page
- `../app/resources/views/admin/register.blade.php`: User registration web page
- `../app/resources/views/admin/home.blade.php`: The main landing web page (I.E. the homepage of the web site)

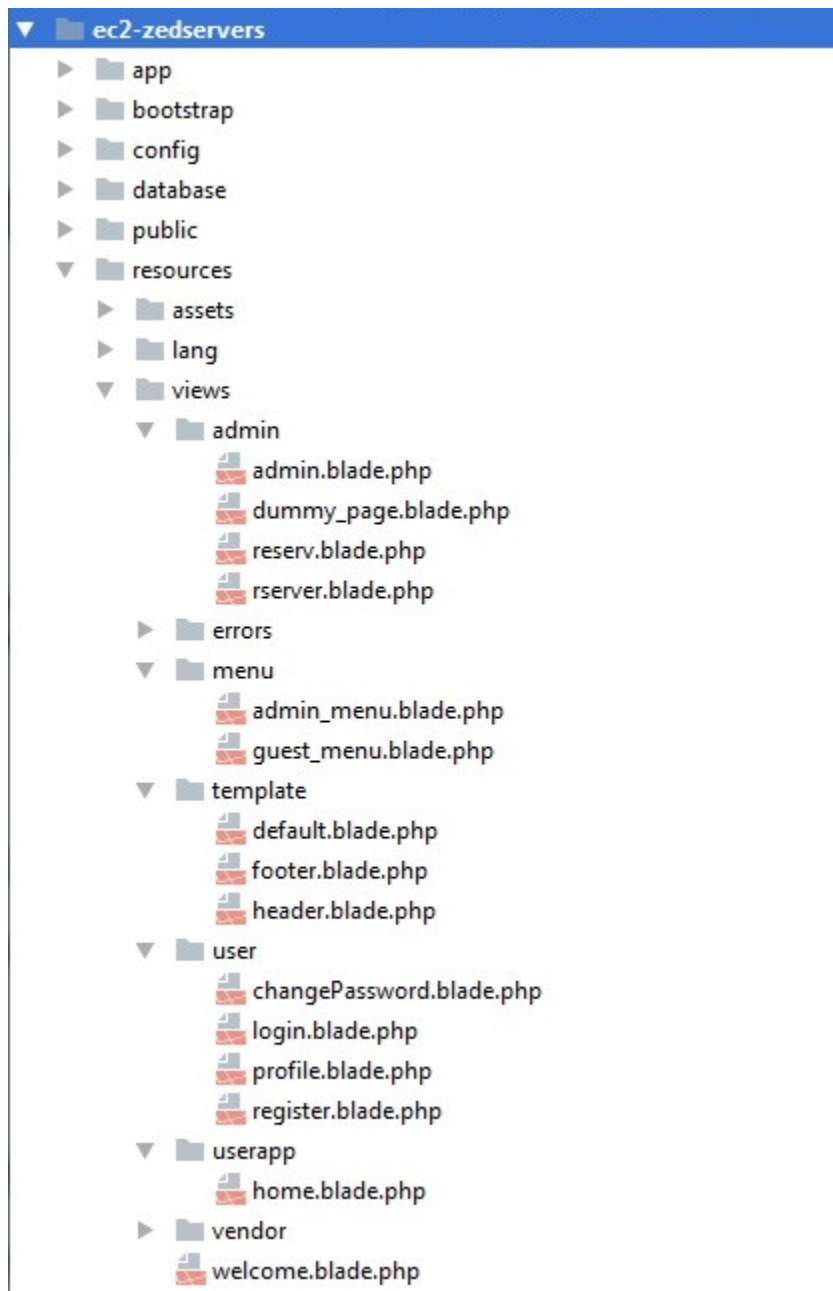


Figure 24 The View Source Code Tree

6.3. Services

The advantages of using a framework such as Laravel is that it come pre-packaged with commonly used services such as authentication, file management, mail management, etc... One such type of services is Helpers.

Helpers are commonly used “tools” in programming such as path functions (set path, find path, etc...) or string manipulation (change string to lower case, upper case, camel case, etc...) or array functions (array set, sort, delete, etc...).

Laravel includes a variety of "helper" PHP functions. Many of these functions are used by the framework itself; however, the developer is free to use them, or create custom helper functions.

Figure 23 shows the helpers folder within the source code tree.

Chapter 7. Website Layout

For the purpose of this project-prototype; the web application is hosted on Amazon Web Services. The following section showcases various screenshots of the web application website as seen by the end user. As well as highlighting the various features of the web application

7.1. Landing Page

The landing page is the main page of the web application. It contains an side-scrolling image slider, a top menu to navigate to various sections of the website, the server selection menu, the scheduling tool as well as the typical, contact us, and about us sections.

7.1.1. Top Menu and Image Slider

Figure 25 shows the top section of the landing view, which includes the image slider and the top menu; as seen by non-logged-in users.

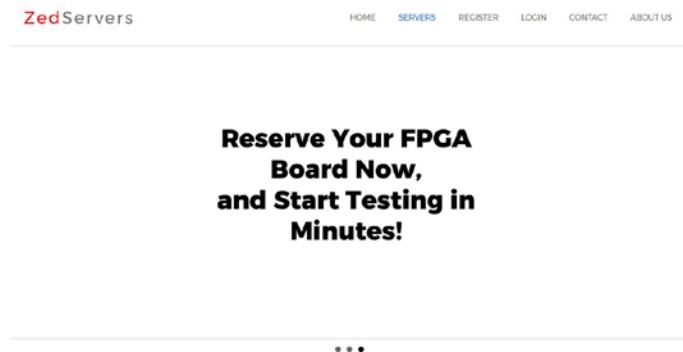


Figure 25 Landing Page with Image Slider and Top Menu

Figure 26 shows the top menu as seen by logged users. Note that the *ADMIN* option is only visible to users with administrative privileges.



Figure 26 Top Menu as Seen by Logged-In Users

7.1.2. Server and FPGA Boards Selection Menu

Figure 27 shows the server and FPGA board type selection menu. The user can navigate through the board tabs by clicking the desired board. Each board type (selected tab) would show a list of servers (buttons). A red button means that the server in question is offline. A yellow button means that the server is online, but is currently being used by any another user. A green button means that the server is online and is available for use.

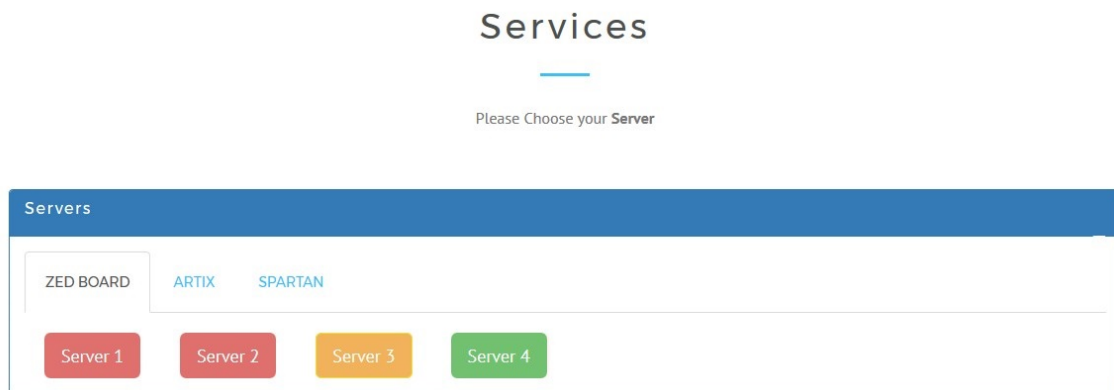


Figure 27 Server and FPGA Board Selection Menu

The webmaster (a user with administrative privileges) could add, remove, or make a given server accessible in the settings page shown in Figure 28.

Enter IP Address	Enable
0.0.0.0	<input checked="" type="checkbox"/>
1.1.1.1	<input checked="" type="checkbox"/>
2.2.2.2	<input checked="" type="checkbox"/>
3.3.3.3	<input checked="" type="checkbox"/>
*IP Address	<input type="checkbox"/>

Save Rename Delete

Figure 28 Server and FPGA Board - Administrative Settings

7.1.3. Server and FPGA Scheduling Tool

Users could make their selection of an FPGA board type and a server. The scheduling tool would then display a two-week window of 24 hour time slots. The user then could see if a given board or server is free or already reserved by another user. The user could make their selection to reserve some hours, and click on a reserve button to finalize the reservation. The user could also see their own time slots that were booked in a prior session. The scheduling tool contains a rolling 14-day window, with 24 hour slots per day.

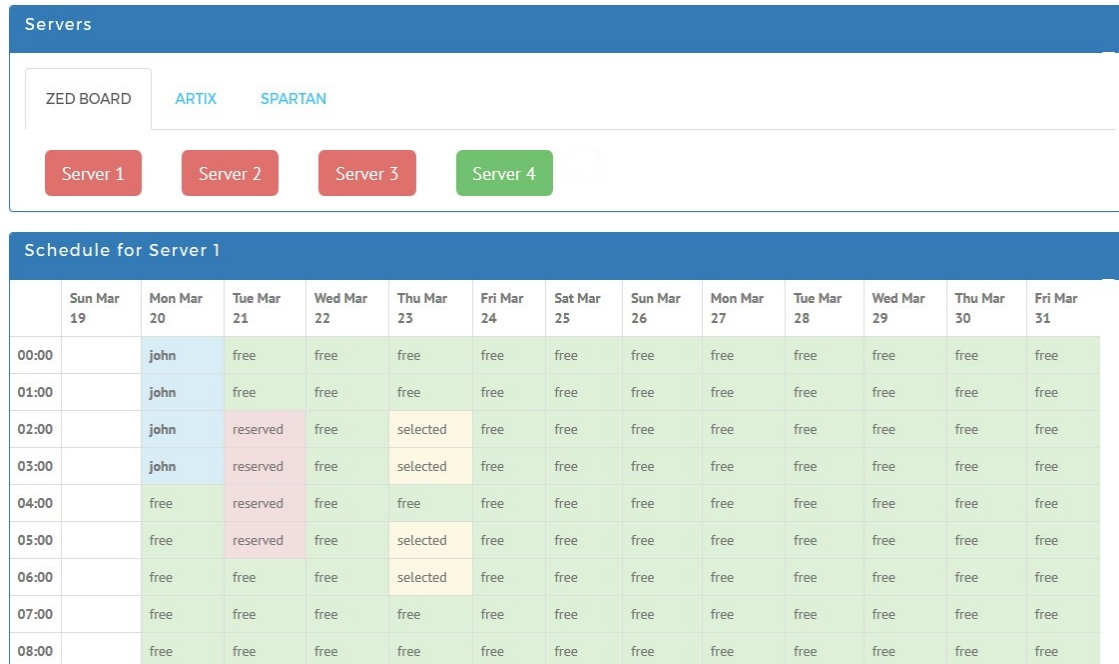
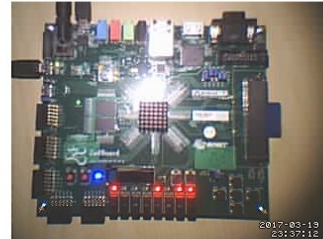


Figure 29 Server and FPGA Board Scheduling Tool

7.2. Remote Server Page

Once the user is granted access to an online and available remote server, the user is redirected to the remote server page. In this page, users are remotely connected to a hardware server, where the FPGA board is connected. Through the server page, users interact with the FPGA platform: Load FPGA demo files, upload user-created FPGA design files, program FPGA design files into the FPGA in question, enable/disable the webcam stream, and the ability to power-cycle (hard reset) the FPGA board. Users also have access to a web-based terminal emulator. This gives them full access to their directory space within the remote hardware server.



```
TERMINAL  UART
TAB 1  +
* Documentation: https://help.ubuntu.com/
27 packages can be updated.
26 updates are security updates.
New release '16.04.2 LTS' available.
Run 'do-release-upgrade' to upgrade to it.
WARNING: Security updates for your current Hardware Enablement Stack
ended on 2016-08-04:
* http://wiki.ubuntu.com/1404_HWE_EOL
There is a graphics stack installed on this system. An upgrade to a
configuration supported for the full lifetime of the LTS will become
available on 2016-07-21 and can be installed by running 'update-manager'
in the Dash.
john@ubuntu:~$
```

Figure 30 Remote Server Page: Connection to Remote FPGA Hardware Server

7.2.1. File Manager and Program Tool

In the remote server page, users have access to the File Manager tool (shown in Figure 31). The tool allows the users to:

- Browse existing demo FPGA files (bitstream and elf files)
- Upload user-created FPGA files (bitstream and elf files)
- Create, remove folders and files
- Program the FPGA with selected FPGA files (bitstream and elf files)

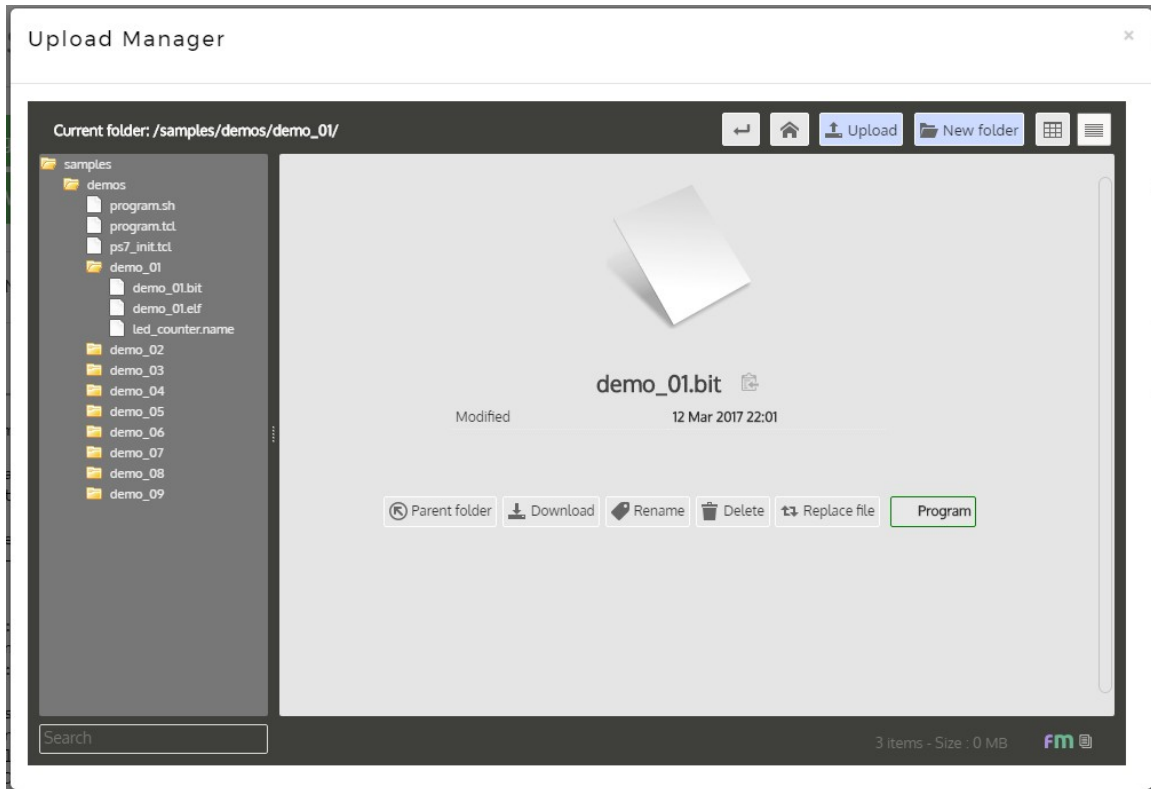


Figure 31 File Manager and Program Tool

Once the user selects the desired FPGA files to be programmed (Figure 31), the Program tool would run a shell script on the remote hardware server; of which the FPGA board is connected to. The shell executes the programming on the remote server (using pre-install FPGA programming tools). The messages displayed during the programming process are streamed back on the web application; to allow users to follow the programming progress (Figure 32).

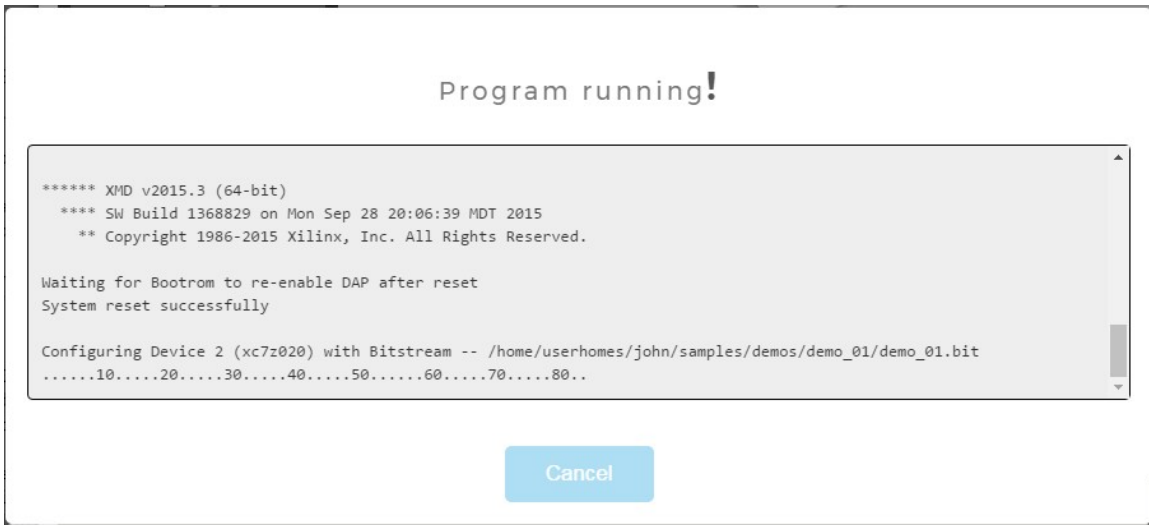


Figure 32 Program Tool: Programming Progress

Chapter 8. Conclusion

Despite the rise of FPGA applications in recent years, access to FPGA evaluation boards is mainly still limited to professional environments and engineering classes within academic institutions. Such limited access is attributed mostly to cost and to the large variety of FPGA selection that could overwhelm would be users. The project's purpose was to develop a working proof of concept; of an engineering web application that would put more FPGA evaluation boards in the virtual hands of users.

The idea in principle is to allow users, through a web application, to connect remotely to FPGA evaluation boards, where they could upload their own bitstream files to test their own FPGA design files; on a user-selected matching FPGA evaluation board. The web application would offer a live web cam stream showing the FPGA board that the user is interacting with remotely, a file manager utility that allows the user to upload and store their own FPGA design files, a program-the-FPGA web tool, a web-based terminal for advanced users, as well as a power-cycle tool to perform hard resets on the FPGA.

The project's two core objectives were:

- Learn some basic and practical FPGA design methodologies to create basic FPGA applications to use as samples to demonstrate the project-prototype in a real world scenario
- Develop a web application using current web development and design methodologies

8.1. FPGA Applications Development: What Was Learned

To create some basic demo sample files, basic FPGA design methodologies had to be learned first. This was achieved by following several online tutorials and guides provided by Xilinx:

- Usage and setup of the hardware FPGA evaluation board kit (ZedBoard with Zynq-7000)

- Usage of Xilinx FPGA software design suite: Vivado. From basic FPGA designs (using PL and PS) to synthesis, and generating bitstreams
- The use of Xilinx's SDK to create application software that would run on a given FPGA hardware design (bitstreams)
- Interfaced various input, output and other peripherals with the FPGA (OLED, LED, UART, RAM, etc.)

8.2. Web Application: What Was Learned

In order to complete the web application a significant effort was made to learn a wide range of web development tools, as this is the largest component of the project:

- Web development with PHP
- Used frameworks such as Laravel; for web development
- Used Amazon Web Services (AWS) and used one of their products (EC2) to host the web application
- Used online repositories (BitBucket) and version software tracking tools (GIT)
- Created various web applications that serve several key functions such as: FPGA board time reservation, user account creation, email confirmation, user access management, remote server access and management, and servers' online/offline status checks
- Used MySQL to manages databases that store user and server info

8.3. Known Bugs

Despite extensive testing and validation of the web application, no software is ever 100% bug-free! There is one known bug within the web application. The bug is related to the web application's reservation/scheduling tool.

The scheduling tool allows the user to reserve some time slots for future use. For example, the user may reserve FPGA board A on server B from 11 AM to 3PM on a specified date within a two-week window.

The bug is such that if the user attempts to make a reservation that would take place between 4PM and 12AM on the same day, while the local time of the user is making this reservation is between 4PM and 12AM; the scheduling app will mark the reservation date as day +1. This is assumed to be a case of Pacific Standard Time being 8 hours behind GMT. However, all web hosting servers, software and variables used have been verified to be all in sync. After several unsuccessful attempts to resolve this bug, it was decided to suspend all subsequent efforts to find a resolution as future work.

8.4. Future Work

While the completed project-prototype did serve its purpose to showcase the web application in a real world usage scenario, some work could be done to enhance the web application.

- In the current website layout of the web application, when the user launches the file manager tool (to program the FPGA or to upload user-created content), the file manager pop-up would obstruct the view of the webcam stream. Both the webcam view frame and the file manager pop-up are static and cannot be moved or resized by the end user. The web application layout should be updated to re-arrange the layout to not have the file manager obstruct the webcam stream view. Effort estimate to add such a feature is small
- When a user uses the reserve/scheduling tool within the web application to book an FPGA board for future use; the user would get a confirmation email of all future bookings made by the user. The reserved time slots are also visible in the user's online dashboard; within the web application. An enhancement would be to add a calendar event, of reserved time slots emailed to the user. Thus the user would have a calendar event reminder on their PC, laptop, tablet, or smartphone. Effort estimate to add such a feature is small

- Address the time zone scheduling bug highlighted in section 7.3
- Currently the UART output (if applicable) from the FPGA becomes visible only when the end user activates the UART tab within the web application. Thus, the user would not be able to view past UART events, like boot-up details (if applicable) and such. An enhancement to the web application could be inspired by Apple's MacOS X Console tool. That is the UART output of the FPGA could be buffered into a file, at all times. Thus when the user activates the UART tab within the web application, the user would see the current UART output, and would have the ability to scroll up within the window to view past events, piped from the UART buffered file. Effort estimate to add such a feature is small to medium
- To develop and test this project-prototype, the choice was made to use the ZedBoard with Xilinx's Zynq-7000 FPGA. While the web application is FPGA and board independent, the web application's program-the-FPGA function uses a shell script on the remote hardware server that is currently specific only to Xilinx's Vivado suite. That is, any other Xilinx FPGA type supported by Vivado would work in this web application without the need for any modification to the hardware server nor to the web application. However, if the user chooses to use an FPGA from another vendor, then the appropriate software and license must be installed on the hardware server, as well as creating a specific shell script to program the FPGA. In such a case, the web application would remotely execute the appropriate program-the-FPGA shell script; matching the FPGA type. Effort estimate to add such a feature is small to medium
- Currently, the remote servers are setup to support only one FPGA board connection at any given time. This is inefficient. A single server, in theory could have multiple FPGA boards connected to it. However, some software would need to be developed to allow the web application to distinguish and route the correct web traffic to the intended FPGA board target, within the same hardware server. Alternatively, one could run several virtual machines per server, where each virtual machine would be linked to one FPGA board

only. Thus, a single physical server would appear as several logical units. Effort estimate to add such a feature is medium

- The current web application was mainly developed and tested using popular web browsers Firefox and Chrome. Some additional work would be needed to support Microsoft's Internet Explorer. Effort estimate to add such a feature is medium to large
- In the appendices, a series of guides are provided showing how to configure the FPGA hardware servers, along with the basic required software installations. Such server configuration is necessary to allow for a successful interaction between the hardware server and the web application. Ideally, this process should be automated by the creation of an install package. Effort estimate to add such a feature is small to medium
- The current project-prototype uses Linux as the OS for the hardware server (server connected to the FPGA board). If the user chooses to use a Windows-based server, some elements of the web application as well as the hardware server implementation would need to be revisited. Effort estimate to add such a feature is large
- Some FPGA evaluation boards have an on-board audio line out, as well as a video output; for advanced FPGA applications. It would be greatly beneficial to have such AV outputs connected to the hardware server, and streamed onto the web application; in a manner similar to the current webcam stream view of the FPGA board. Effort estimate to stream the audio line out is medium. However, the effort to stream the video out is large. And, it would require the addition of a video capture card to the hardware server
- Currently the web application has two levels of user privileges: Administrative users (example: the webmaster or IT) and regular users (the targeted end user on the web). Users with administrative privileges would be able to add, remove, and update the number of boards or servers that could be made accessible to the web application. They could also manage the scheduling tool such as limiting the number of hours a given user can reserve a given FPGA board, per day, or week. As well as online/offline server status checks

timeout limits, etc.... A third user privilege category being proposed is on a group level. That is, if the application is being used by company A, then they would have their own internal administrative users that would have a group-level privilege. This group level would sit a step below admin-level users. Effort estimate to add such a feature is small

- The web application has several tools to allow the user to interact and program an FPGA board remotely. While all the tools are accessible to all users, some tools are more tailored for extended debugging and testing, such as the web terminal emulator that offers CLI access to the remote hardware server. Another testing tool that could be added is to allow the user to interact with their software application that would be running on the FPGA, from the terminal. That is in addition of the current demo files that are included in the web application, another demo that could be added to allow the user to enter a command from a CLI (similar to the application shown in Appendix M) to interact with the IOs on the FPGA board such as the LEDs. Or even run a bunch of wrapper scripts that would display messages on the OLED. Effort estimate to add such a feature is small to medium

References

- [1] Grand View Research, "Global FPGA Market Size To Reach USD 14.2 Billion By 2024," Grand View Research, December 2016. [Online]. Available: <http://www.grandviewresearch.com/press-release/global-fpga-market>. [Accessed 13 February 2017].
- [2] Global Market Insights, "FPGA Market Size By Application," Global Market Insights, February 2016. [Online]. Available: <https://www.gminsights.com/industry-analysis/field-programmable-gate-array-fpga-market-size>. [Accessed 13 February 2017].
- [3] B. Darrow, "The First Chip From Intel's Altera Buy Will Be out in 2016," 18 November 2015. [Online]. Available: <http://fortune.com/2015/11/18/intel-xeon-fpga-chips/>. [Accessed 11 March 2017].
- [4] R. Miller, "Intel Unveils FPGA to Accelerate Neural Networks," 15 November 2016. [Online]. Available: <http://datacenterfrontier.com/intel-unveils-fpga-to-accelerate-ai-neural-networks/>. [Accessed 11 March 2017].
- [5] Amazon Web Services, "Amazon Elastic Compute Cloud," Amazon Web Services, [Online]. Available: <https://aws.amazon.com/ec2/>. [Accessed 13 February 2017].
- [6] Avnet Electronics Marketing, "ZedBoard Getting Started Guide," 30 January 2014. [Online]. Available: <http://zedboard.org/sites/default/files/documentations/GS-AES-Z7EV-7Z020-G-V7.pdf>. [Accessed 28 September 2014].
- [7] Avnet Electronics Marketing, "ZedBoard Hardware User's Guide," 27 January 2014. [Online]. Available: http://zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf. [Accessed 16 February 2017].
- [8] Avnet Electronics Marketing, "Configuration and Booting Guide," 17 August 2012. [Online]. Available:

https://forums.xilinx.com/xlnx/attachments/xlnx/ELINUX/8461/1/ZedBoard_boot_guide_IDS14_1_v1_1.pdf. [Accessed 9 November 2014].

- [9] B. Darrow, "Official At Last: Intel Completes \$16.7 Billion Buy of Altera," 28 December 2015. [Online]. Available: <http://fortune.com/2015/12/28/intel-completes-altera-acquisition/>. [Accessed 11 March 2017].

Appendix A. VMware Workstation Player and Ubuntu 14.04.2 Virtual Machine Installation Guide

To set up an Ubuntu (Linux) virtual machine in a Windows OS, first, one needs to download the Ubuntu ISO image and the VMware Workstation Player. The following guide uses:

- Ubuntu 14.04.2 LTS
- VMware Workstation Player 12.1.1

Second, one would start by installing the VMware Player. Then, one would create a new virtual machine using the VMware Player software. Installing VMware Workstation Player is simple and straightforward. Run the installation executable and follow the install prompts; typical of any Windows-based software install. Once complete, and if prompted, reboot the PC.

To create an Ubuntu virtual machine with VMware Player in a Windows 7 environment, perform the following steps:

1. Launch VMware Workstation 12 Player
2. On the right side, click Create a New Virtual Machine

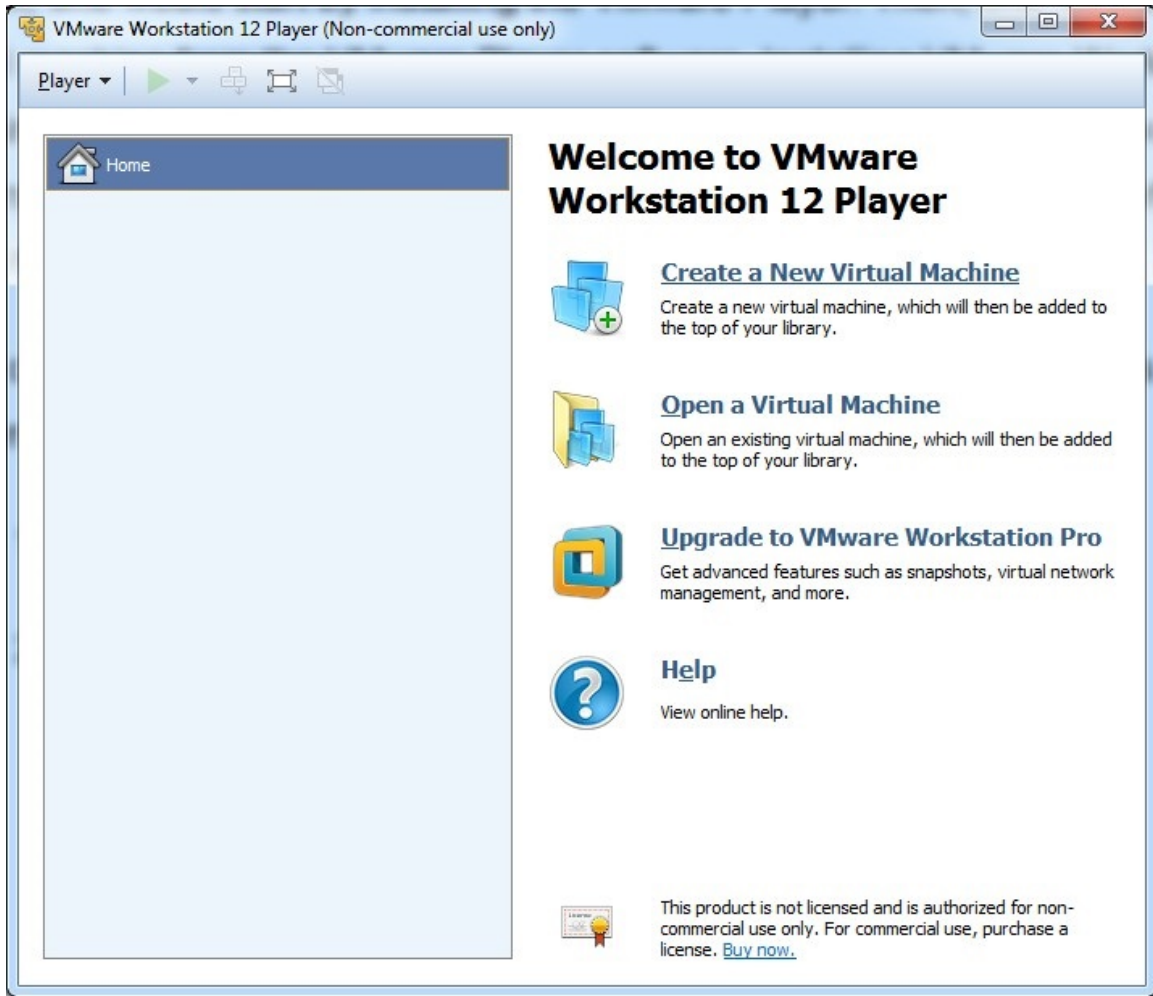


Figure A1 Create a New Virtual Machine

3. Select Installer disc image file (ISO) and Browse to the location of the ubuntu-14.04.2-desktop-amd64.iso, then click Next

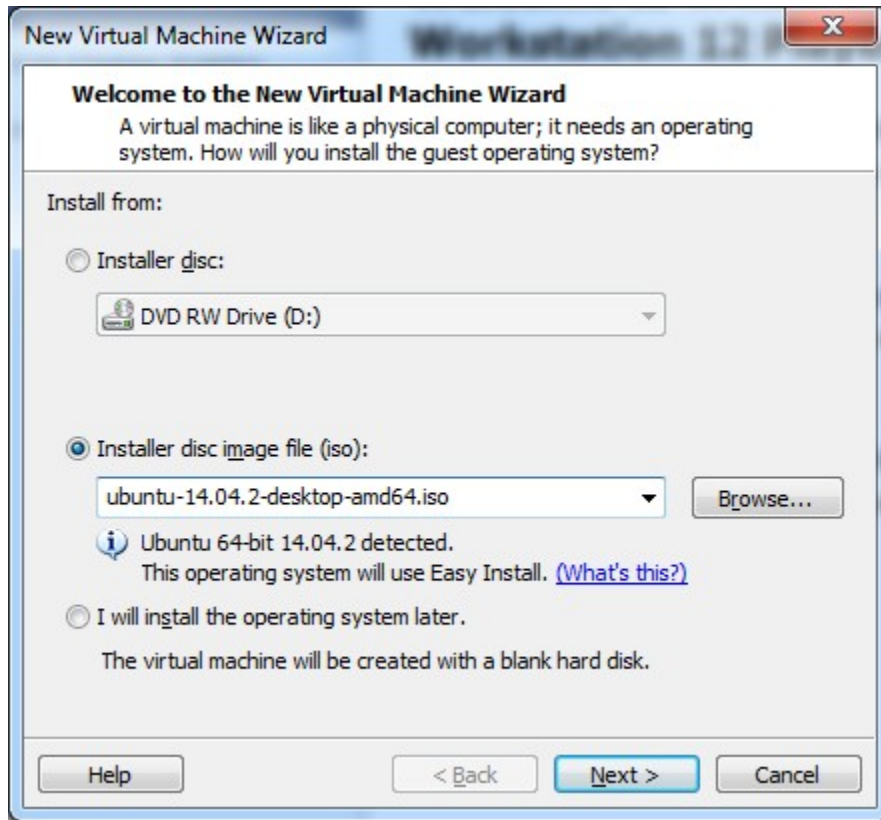


Figure A2 Select the Ubuntu 14.04.2 LTS ISO Image

4. Create the user profile

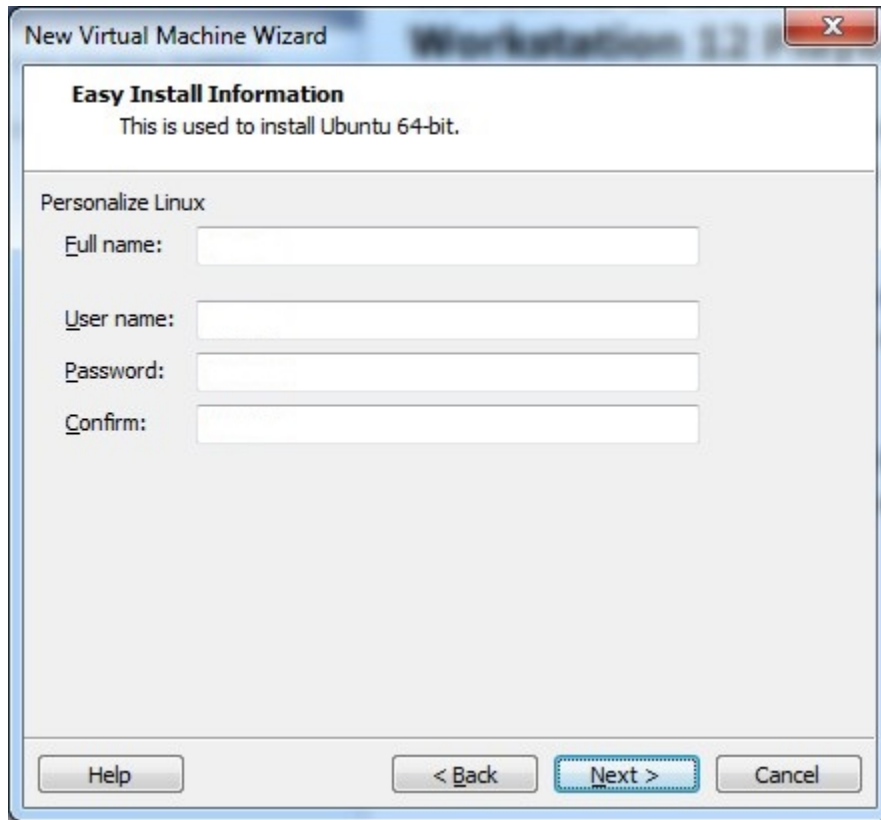


Figure A3 Create a User Profile

5. Provide a name to the virtual machine, or accept the default. It's recommended that you give more details in the name rather than just a generic name, create a name that would specify Ubuntu's version, 32 or 64 bits, the date installed, etc... As one could create several virtual machines, detailed names would allow for easier identification. Choose a different location to store the virtual machine disk file(s), or accept the default location

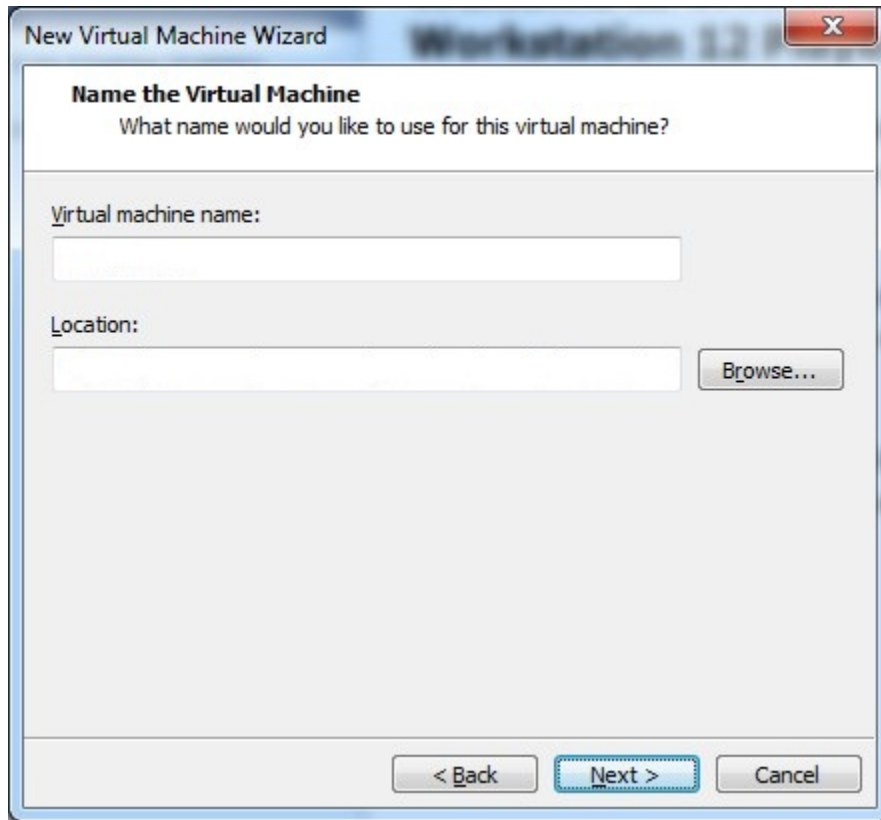


Figure A4 Provide a Name and Location to the Virtual Machine

6. Specify the disk capacity to 40G, or more. Note that it's not straightforward to change the disk size of the virtual image at a later time. Furthermore, if one sets up the disk image to be too large that goes unused, then again it is wasted space and not straightforward to reduce it
7. Depending on the host OS, and whether the virtual machine image would be copied to other locations, storing the virtual disk as a single file might be a better or worse option than splitting the virtual disk into multiple files. If one doesn't know what to do, use the default setting

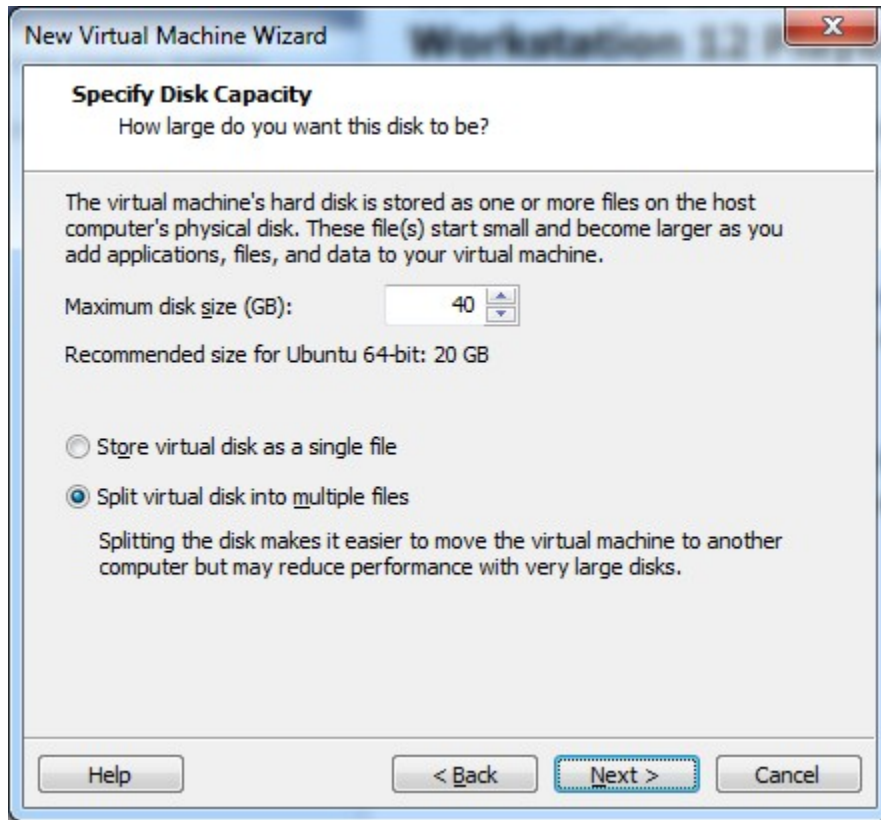


Figure A5 Specify Disk Capacity

8. One could customize the Hardware now or later. Hardware customizations allows the user to allocate more or less resources to the virtual machine. One could start with the default settings, and adjust as needed at a later time.
Click Finish
 - a. Recommendation for hardware customization to improve the performance of the virtual machine if needed: Increase the memory allocated, increase the number of processors allocated, and set the network adapter to Bridged and check Replicate physical network connection state

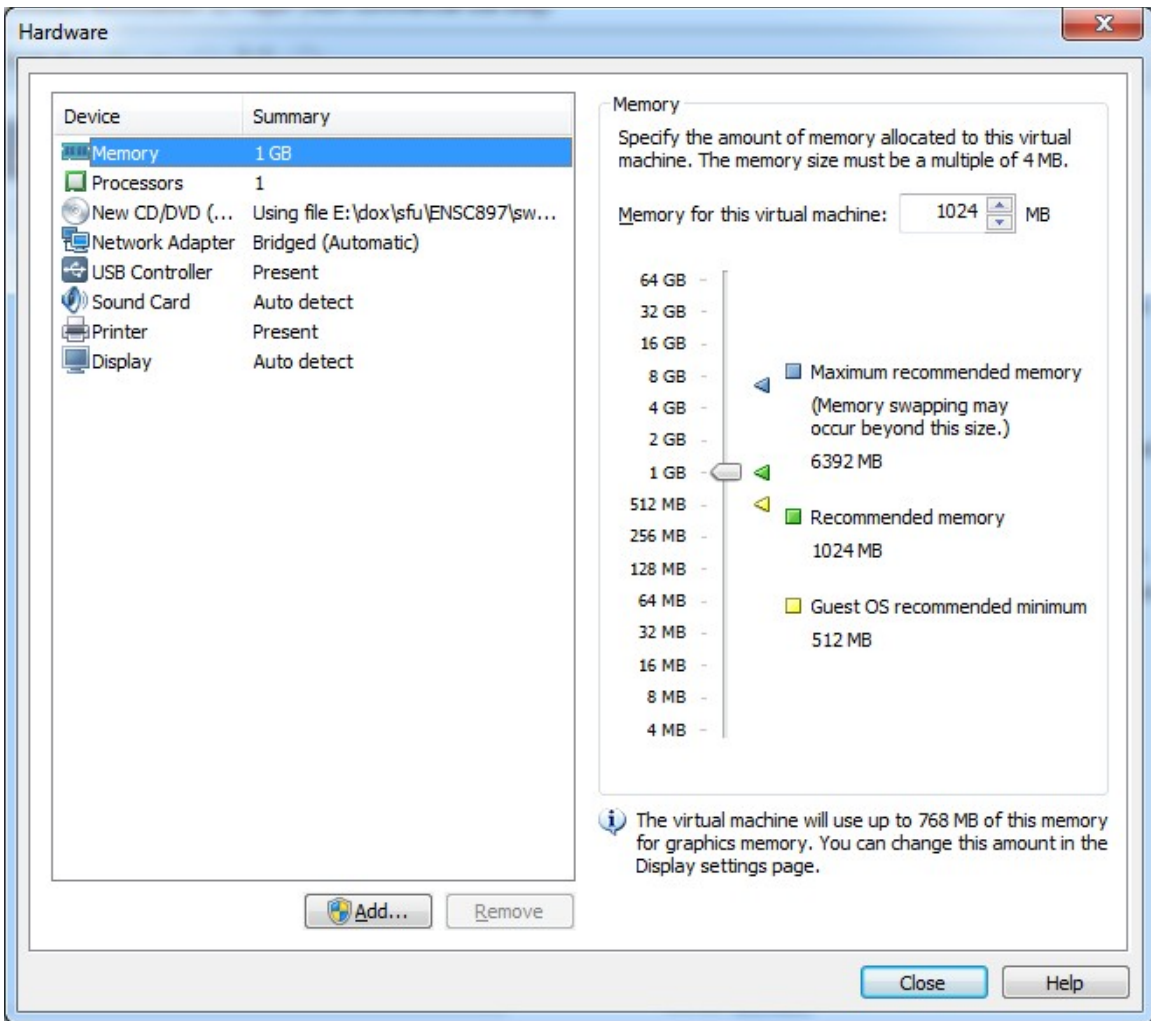


Figure A6 Memory Allocation

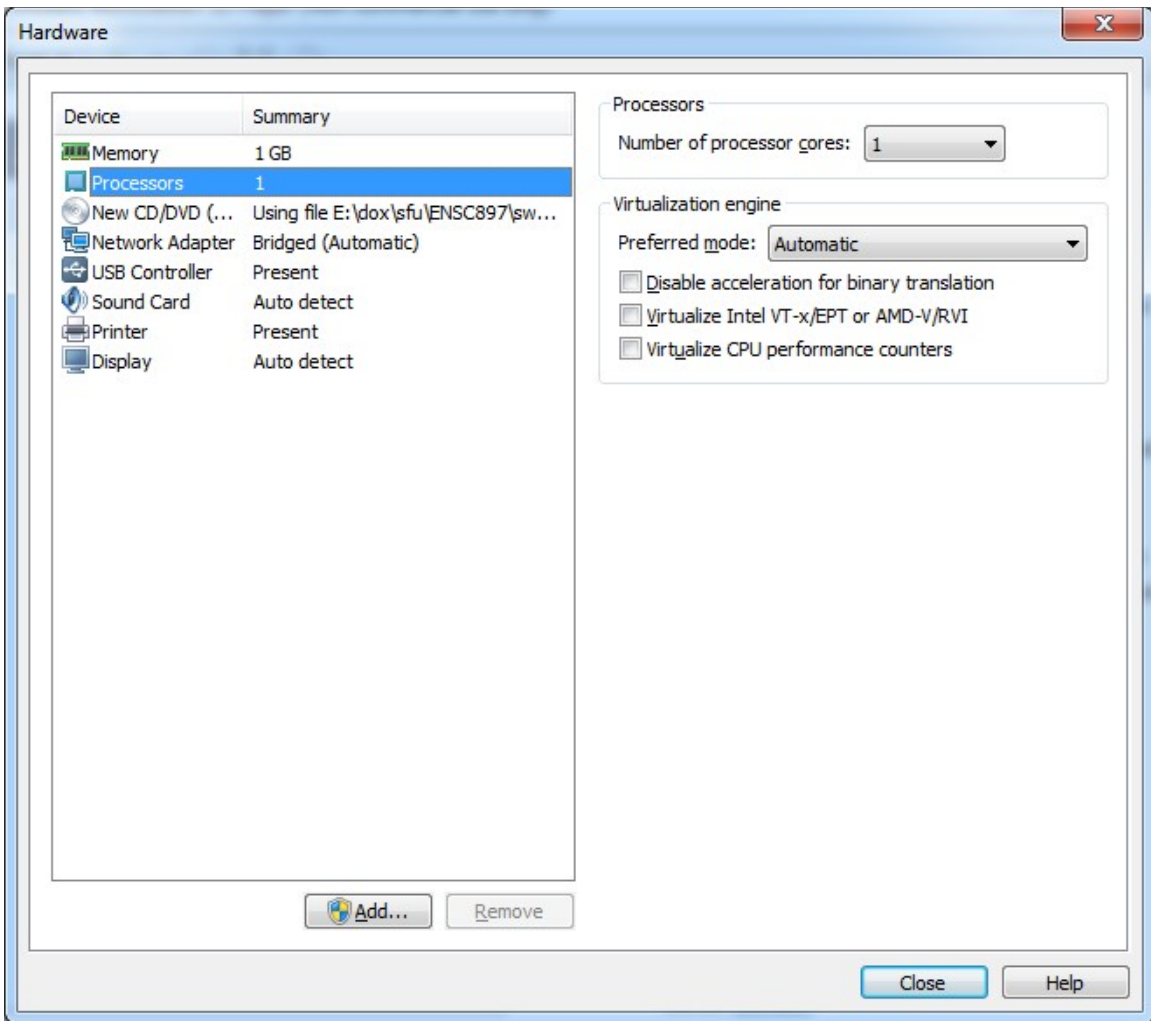


Figure A7 Processors Allocation

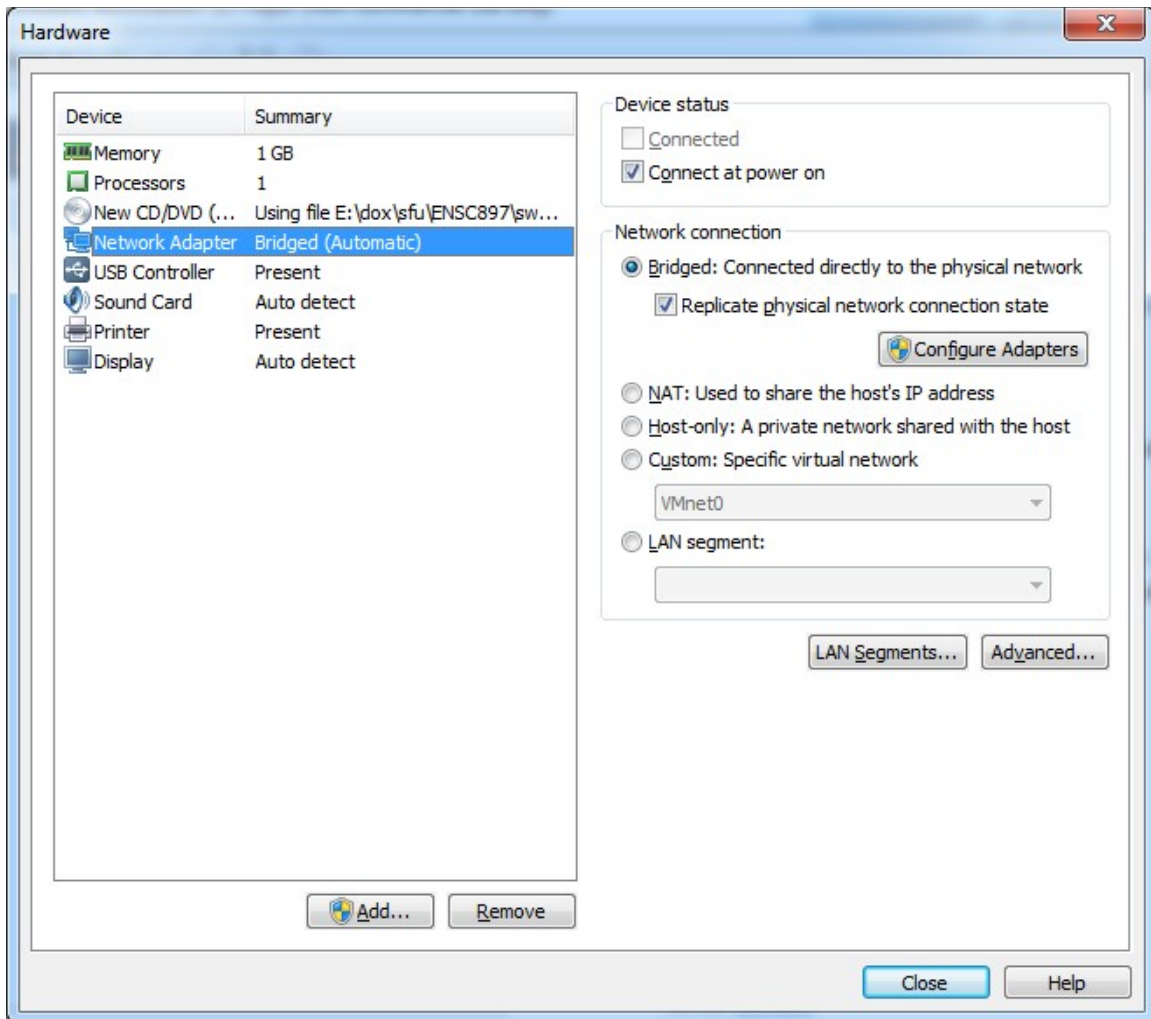


Figure A8 Network Adapter Configuration

9. The install process of Ubuntu should start at this point and would take around 10-15 minutes, depending on the PC resources available. Once the installation is complete, the user could login immediately to the Ubuntu virtual machine

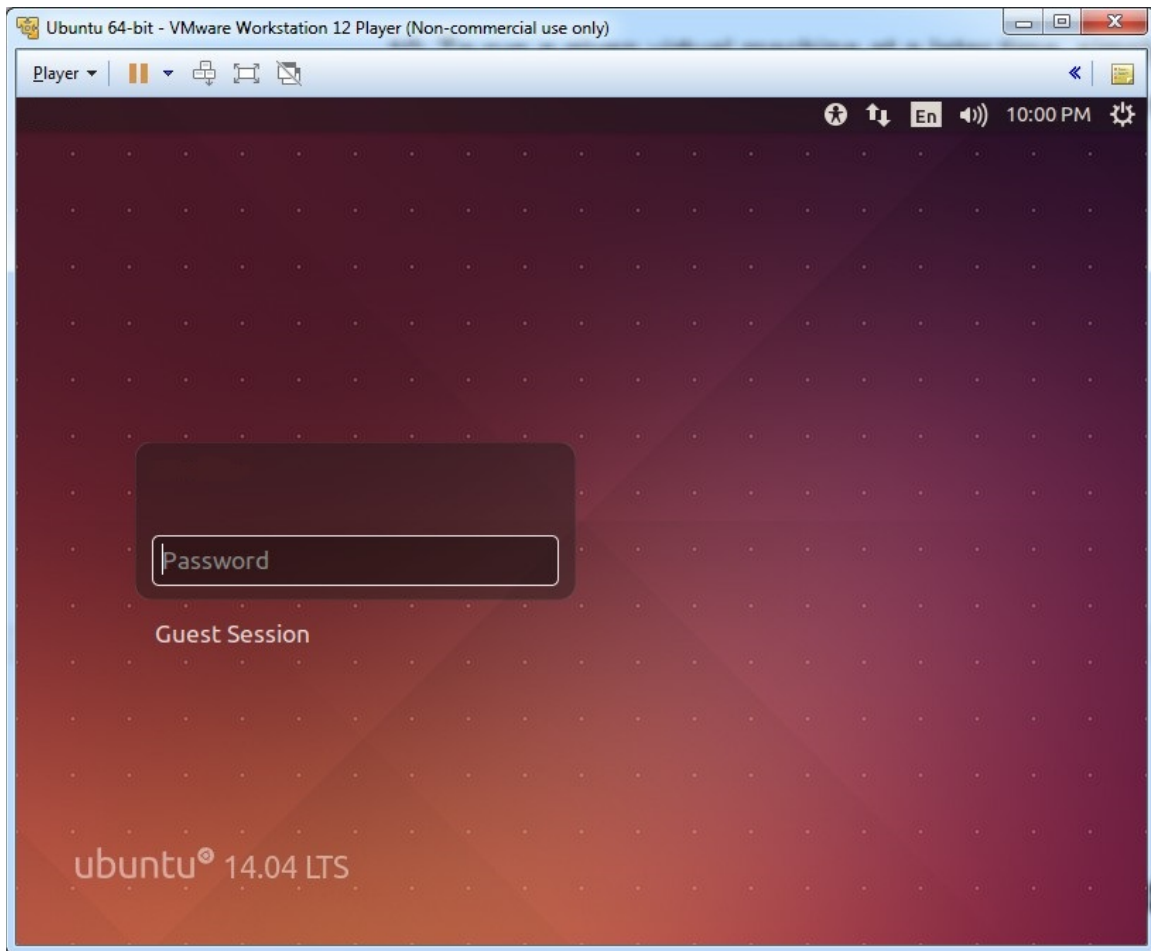


Figure A9 Login Page of the Ubuntu 14.04.2 OS as a Virtual Machine

10. To run a given virtual machine at a later time, simply launch VMware Workstation 12 Player, and select the virtual machine to be powered-on from the left side, then click Play virtual machine

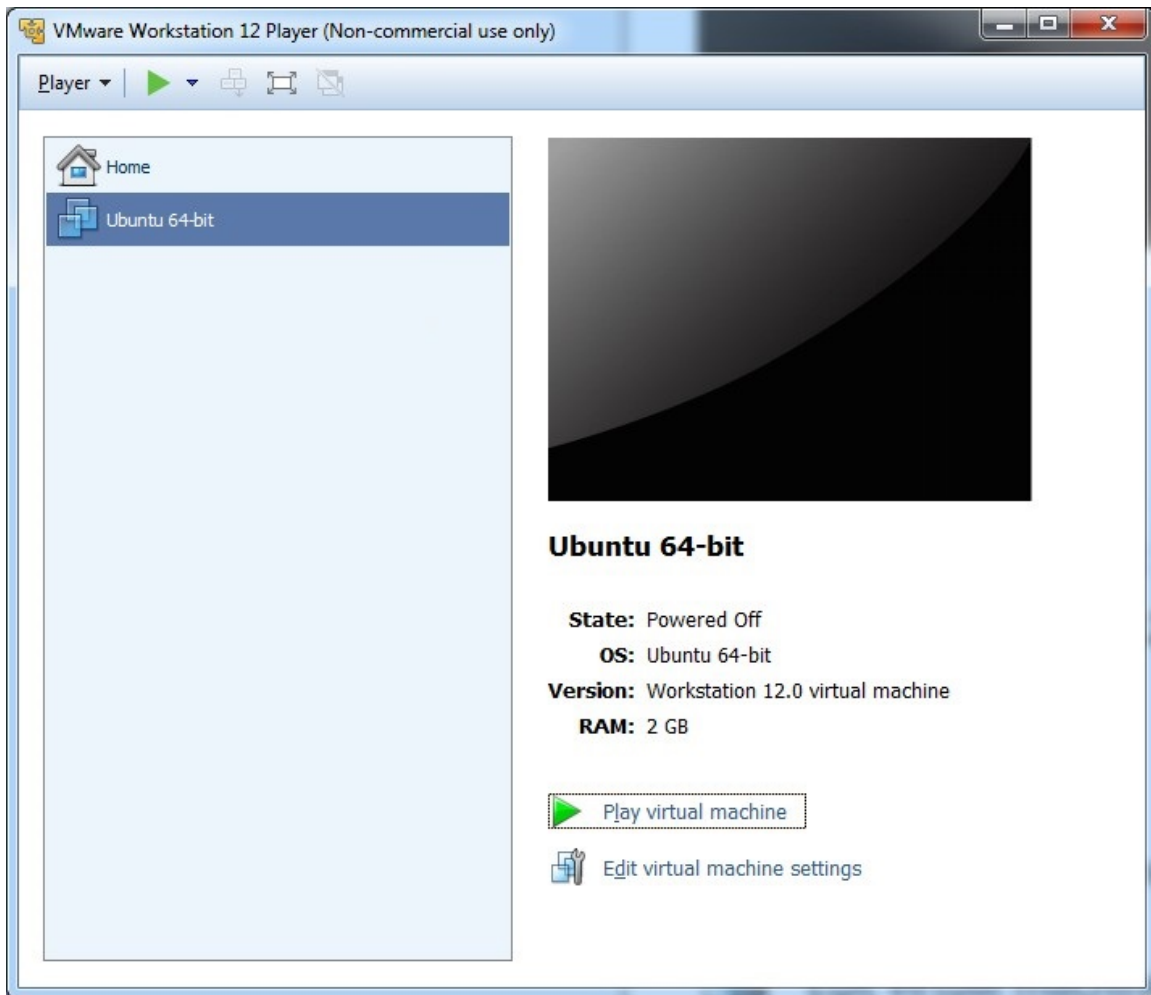


Figure A10 Play Virtual Machine from VMware Workstation 12 Player

This completes the VMware Workstation Player and Ubuntu Virtual Machine installation.

Appendix B. Hardware Server Configuration Guide

The hardware server's OS where the FPGA evaluation board is connected to, (in this case the virtual machine running Ubuntu 14.04.2 LTS) needs to be configured to allow for remote network access (by opening required ports for SSH, Shell in a Box and the webcam stream), and to allow for full interaction with the web application such as creating user-defined directories, creating user-defined files, and to program the FPGA with bitstream files. Also, in order for the web application to communicate securely over SSH with AWS's hosting server, a copy of the public key is created on the hardware server.

The configuration is done by running a shell script. This shell script must be run on every newly configured server (Running Linux). Copy and paste the following shell script into the server to be run on:

```
#!/bin/bash
exec 2>/var/log/init_errors.log || { echo 'LOG PERMISSIONS FAIL'
; exit 1; }

#create user
if ! id -u usercreator >/dev/null 2>&1; then
    useradd -d /home/usercreator -s /bin/bash -p $(echo
e485a60ce2a0457653daa7c84265c688 | openssl passwd -1 -stdin)
usercreator
fi

#create user home directory
if [ ! -d "/home/usercreator" ]; then
    mkdir /home/usercreator
    chown usercreator:usercreator /home/usercreator
    chmod 755 /home/usercreator
fi

#set sudo permissions for exec useradd without password
echo 'usercreator    ALL=NOPASSWD: ALL' >> /etc/sudoers

#create home directories for users what will be created
if [ ! -d "/home/userhomes" ]; then
    mkdir /home/userhomes/
    chown usercreator:usercreator /home/userhomes
    chmod 755 /home/userhomes
fi

#create directory to store SSH keys
if [ ! -d "/home/usercreator/.ssh" ]; then
    mkdir /home/usercreator/.ssh
    chown usercreator:usercreator /home/usercreator/.ssh
```



```

    chmod 700 /home/usercreator/.ssh
fi

#create file to store SSH public keys
touch /home/usercreator/.ssh/authorized_keys
chown usercreator:usercreator
/home/usercreator/.ssh/authorized_keys
chmod 700 /home/usercreator/.ssh/authorized_keys

#create config file for motion web cam software
if [ ! -d "/home/usercreator/.motion" ]; then
    mkdir /home/usercreator/.motion
fi

> /home/usercreator/.motion/motion.conf
echo 'webcam_port 80' >> /home/usercreator/.motion/motion.conf
echo 'webcam_localhost off' >>
/home/usercreator/.motion/motion.conf
echo 'webcam_maxrate 100' >>
/home/usercreator/.motion/motion.conf
echo 'output_normal off' >> /home/usercreator/.motion/motion.conf
echo 'auto_brightness on' >>
/home/usercreator/.motion/motion.conf
echo 'height 480' >> /home/usercreator/.motion/motion.conf
echo 'width 640' >> /home/usercreator/.motion/motion.conf
echo 'webcam_quality 100' >>
/home/usercreator/.motion/motion.conf
echo 'switchfilter on' >> /home/usercreator/.motion/motion.conf

chown -R usercreator:usercreator /home/usercreator/.motion
chmod 755 -R /home/usercreator/.motion

#create script to start motion
> /home/usercreator/start_mot.sh
echo '#! /bin/bash' >> /home/usercreator/start_mot.sh
echo 'if ! id -u "$1" >/dev/null 2>&1; then' >>
/home/usercreator/start_mot.sh
echo 'sudo useradd -d /home/userhomes/$1 -s /bin/bash -p $(echo
$2 | openssl passwd -1 -stdin) $1' >>
/home/usercreator/start_mot.sh
echo 'fi' >> /home/usercreator/start_mot.sh
echo 'sudo usermod -a -G dialout $1' >>
/home/usercreator/start_mot.sh
echo 'if [ ! -d "/home/userhomes/$1" ]; then' >>
/home/usercreator/start_mot.sh
echo 'sudo mkdir /home/userhomes/$1' >>
/home/usercreator/start_mot.sh
echo 'sudo cp /root/.bashrc /home/userhomes/$1' >>
/home/usercreator/start_mot.sh
echo 'sudo chown -R $1:$1 /home/userhomes/$1' >>
/home/usercreator/start_mot.sh
echo 'sudo chmod -R 755 /home/userhomes/$1' >>
/home/usercreator/start_mot.sh
echo 'fi' >> /home/usercreator/start_mot.sh
echo 'if ! pgrep "motion" > /dev/null; then' >>
/home/usercreator/start_mot.sh

```

```

echo 'sudo nohup motion > /dev/null 2> /dev/null < /dev/null &'
>> /home/usercreator/start_mot.sh
echo 'fi' >> /home/usercreator/start_mot.sh
echo 'exit' >> /home/usercreator/start_mot.sh

#create script to stop motion
> /home/usercreator/stop_mot.sh
echo '#! /bin/bash' >> /home/usercreator/stop_mot.sh
echo 'sudo pkill motion' >> /home/usercreator/stop_mot.sh

chmod +x /home/usercreator/start_mot.sh
chmod +x /home/usercreator/stop_mot.sh

#put public RSA key to file
echo 'ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQDAQDXWjhpwdywnHiEyhmEfg+cE1cf5abfN8QX2
ghv5rSDtnoVFF2HAqhgl4WuleeOdGSu4bx9iFWgtQFEVz0bP41aWdi4rdEPPYfH8l
yuUe9f69pfMK9MtOf3FzYQy+icPVRnaNYelqLtt2OK9obI+XCdQlJGdRopVhs6pYg
JjCjwd24WIHdPT9/h0ec4eDvRqlgk4bZBYOdVCKCul6d+dl73vySjw6z59xONWI6m
fA7kZMfAO1Oz8drLP6p3R6LMQjmTM2Aq13Jbp6FhdH59lecYC20iIPziDgLJi0zqz
YIKcwtcDfYiIuXdKHp5ax/t0Ecd8nz2n/XmPnvaR7HyjSdr webdev@ip-172-31-
45-129' >> /home/usercreator/.ssh/authorized_keys

#check log
if [ -s /var/log/init_errors.log ]; then
echo 'Error. see log at /var/log/init_errors.log'
fi

```

Save the file, and execute the shell script.

This completes the configuration required for the hardware server.

Appendix C. Open SSH Installation on the Hardware Server

The web application needs remote access to the hardware servers to interact with the FPGA evaluation boards remotely. SSH is used within the web application to establish a secure remote connection. To install Open SSH on the hardware server do the following:

1. Open a terminal
2. Run: `apt-get install openssh-server`
3. Open a text editor. Example: `gedit /etc/ssh/sshd_config`
4. Update the following variable
 - a. Port 50000
5. Run: `service ssh restart`

Note that sudo privileges may be needed to perform such an install.

Appendix D. Motion (Webcam Stream) Installation on the Hardware Server

The web application makes use of a webcam connected to the hardware server, which streams a top view of the FPGA evaluation board. This allows the end user to see any visual changes; specific their FPGA application such as: blinking LEDs, or messages shown on the OLED. The choice was made to use Motion, which is a Linux-based webcam stream software. To install Motion on the hardware server do the following:

1. Open a terminal
2. Run: `apt-get install motion`

Note that sudo privileges may be needed to perform such an install.

Appendix E. C-Kermit (Serial Terminal) Installation on the Hardware Server

The web application allows the end user to access the UART port of the FPGA through the serial ports of the hardware server. To allow for such an interaction on the serial communication channel between the FPGA's UART port and the hardware server's serial port, a software acting as a terminal point should be installed. The choice was made to use C-Kermit; due to its advanced features. To install C-Kermit do the following:

1. Open a terminal
2. `sudo apt-get install ckermit`
3. `sudo apt-get remove modemmanager`
4. `sudo usermod -a -G dialout $USER`

Note that sudo privileges may be needed to perform such an install. Please see Appendix J for COM port settings.

Appendix F. Git Installation on the Hardware Server

Having the source code developed for this project available on an online repository; it is necessary to install a version control software to be able to clone such source code into the Hardware Server (running Ubuntu OS), or any other Linux-based machine. The choice was made to use Git. To install Git do the following:

1. Open a terminal
2. Run: `apt-get install git`

Note that sudo privileges may be needed to perform such an install.

Appendix G. Shell in a Box Installation on the Hardware Server

The web application uses Shell in a Box as the web-based terminal emulator which provides the look and feel of a native shell. The terminal window is presented as a frame within the web application's web page. The terminal is available for advanced user who want to do on the fly edits, or want to run additional FPGA programming commands.

To install Shell in a Box on an Ubuntu machine, first the user needs to install the dependencies, clone the source files from the git repository, install the package, and finally edit a configuration file. Open a terminal in Ubuntu then run the following commands:

1. `apt-get install git libssl-dev libpam0g-dev zlib1g-dev dh-autoreconf`
2. `git clone https://github.com/shellinabox/shellinabox.git && cd shellinabox`
3. `dpkg-buildpackage -b`
4. `cd ../`
5. `dpkg -i shellinabox_{ver}_{arch}.deb`
6. Open a text editor. Example: `gedit /etc/default/shellinabox`
7. Update the following variables:
 - a. `SHELLINABOX_PORT=88`
 - b. `SHELLINABOX_ARGS="--no-beep --disable-ssl --messages-origin '*'"`
8. `service shellinabox restart`

Note that sudo privileges may be needed to perform such an install.

Appendix H. Network Ports Configuration

In order for the web application to be able to interact with the hardware servers remotely through the web; some network ports on the local network relative to the hardware server must be enabled and forwarded properly. That is in a given LAN, where hardware server A is connected to; the router/switch that is managing the LAN in question must have port forwarding enabled and have the appropriate specific traffic forwarded to hardware server A. The web application uses four different ports. The following ports must be enabled and forwarded to the local IP number matching the hardware server in question:

1. SSH (secure remote shell connection): port 50000
2. Shell in a Box (web terminal emulator): port 88
3. Motion (webcam): port 80
4. Remote power cycle (Hard FPGA board reset): port 8080

Appendix I. Xilinx Vivado Design Suite Installation Guide

This guide demonstrates the steps required to install the Xilinx Vivado Design Suite on Windows 7 and Ubuntu 14.04.2. The install procedure is almost the same for both Windows and Ubuntu operating systems. For Windows, the user needs only to run the installer, and follow the prompts. However, for Ubuntu, there are a few extra steps required. For this reason, the guide would describe the installation process for Vivado under the Ubuntu OS.

1. In Ubuntu, open a terminal and find the directory that contains the uncompressed Vivado installation package.
 - a. Example:
`./home/username/Downloads/Xilinx_Vivado_SDK_2015.3_0929_1/`
2. Change the installation package directory and its subdirectories' permissions to full allow full read/write access
 - a. Example (sudo may be required): `chmod -R 777`
`./home/username/Downlaods/Xilinx_Vivado_SDK_2015.3_0929_1/`
 - b. Verify that all files within the target directories have had their permissions changed as intended
3. Navigate into the install package directory and run the installer using sudo. This would run the installer and the GUI installation process would start
 - a. Example: `sudo ./xsetup`

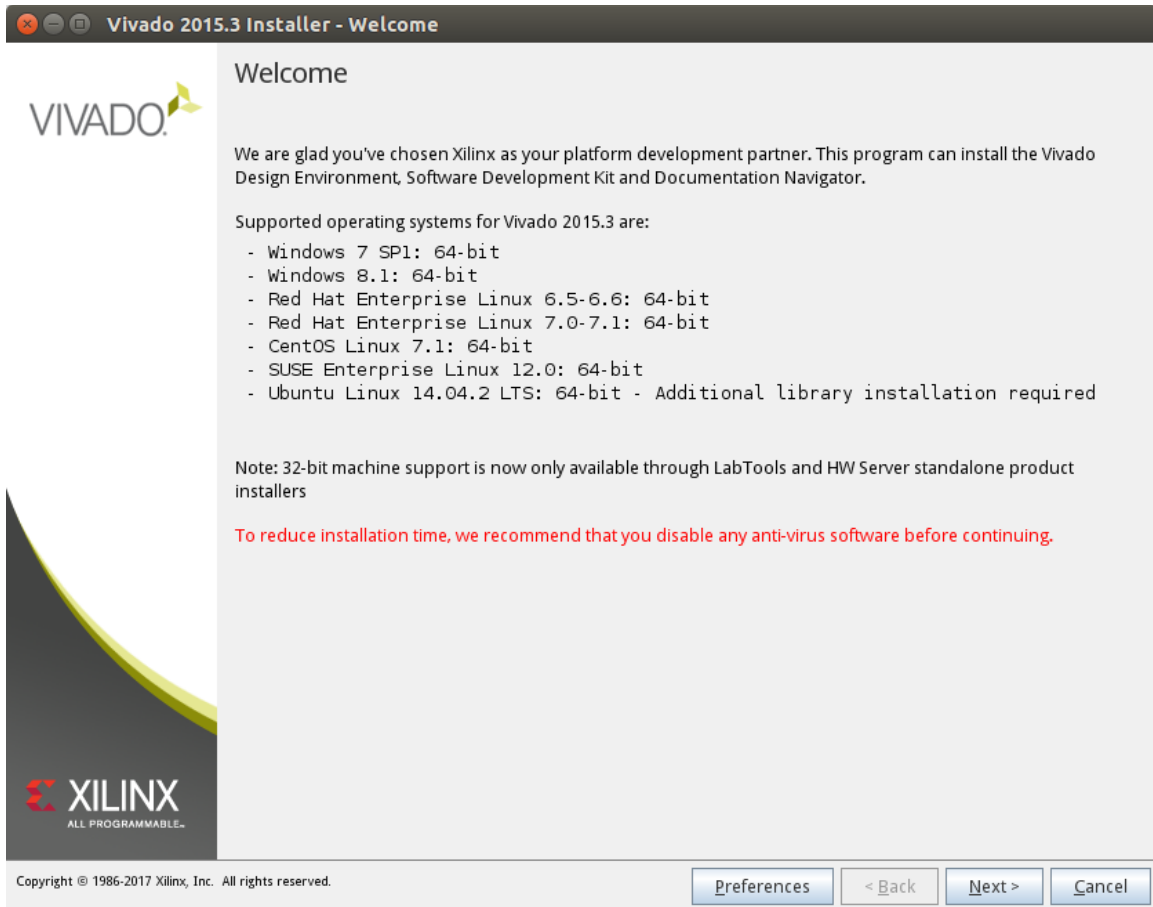


Figure I1 Xilinx's Vivado Installer

4. Click Next. Select I Agree to all three TOS and click Next

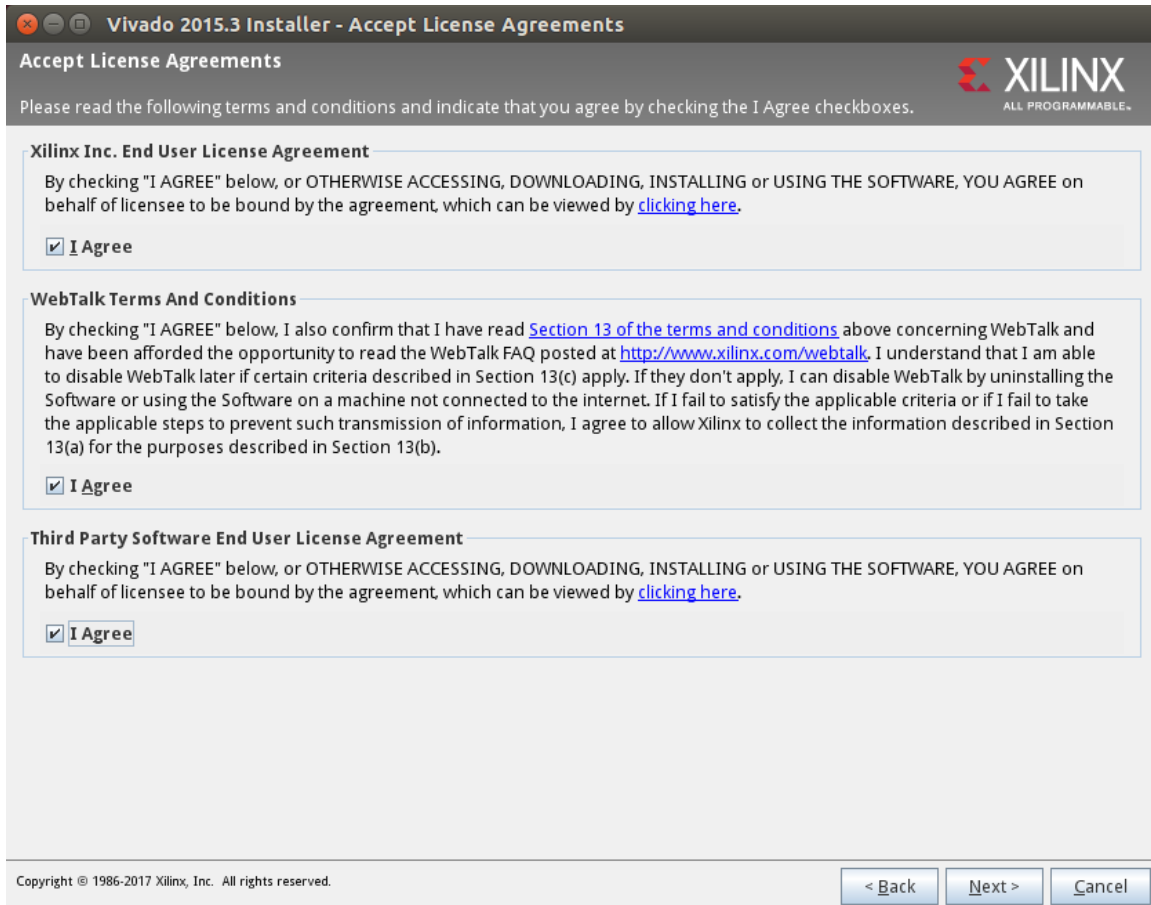


Figure I2 Accept License Agreements in Vivado Installer

5. Select the edition to be installed. This would be dependent on the acquired license

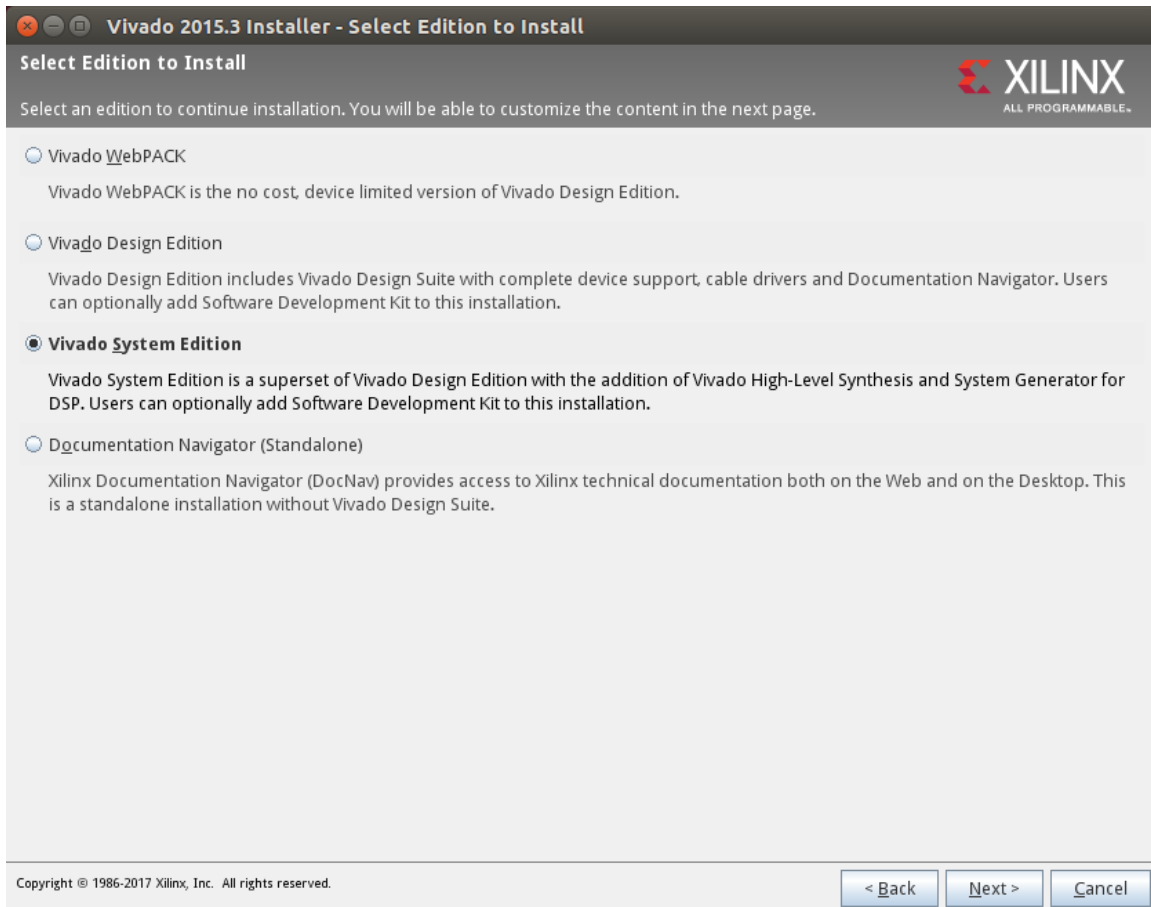


Figure I3 Vivado Editon Selection

6. Ensure that under Design Tools: Vivado Design Suite and Software Development Kit are selected. This is a must! DocNav is basically the software documentation/help, and it should be selected as well. Under Devices, the user should make the selection dependent on the type of FPGA they would be using. For this project, a Zynq-7000 FPGA was used. It doesn't hurt to select all. However, that would also depend on the acquired license and on space available
 - a. Note that while installing Vivado under Linux, the cable drivers would not be installed automatically as is the case with the Windows installer. Xilinx's user guide UG973 (found online) provides the steps required, which are shown towards the end of this section

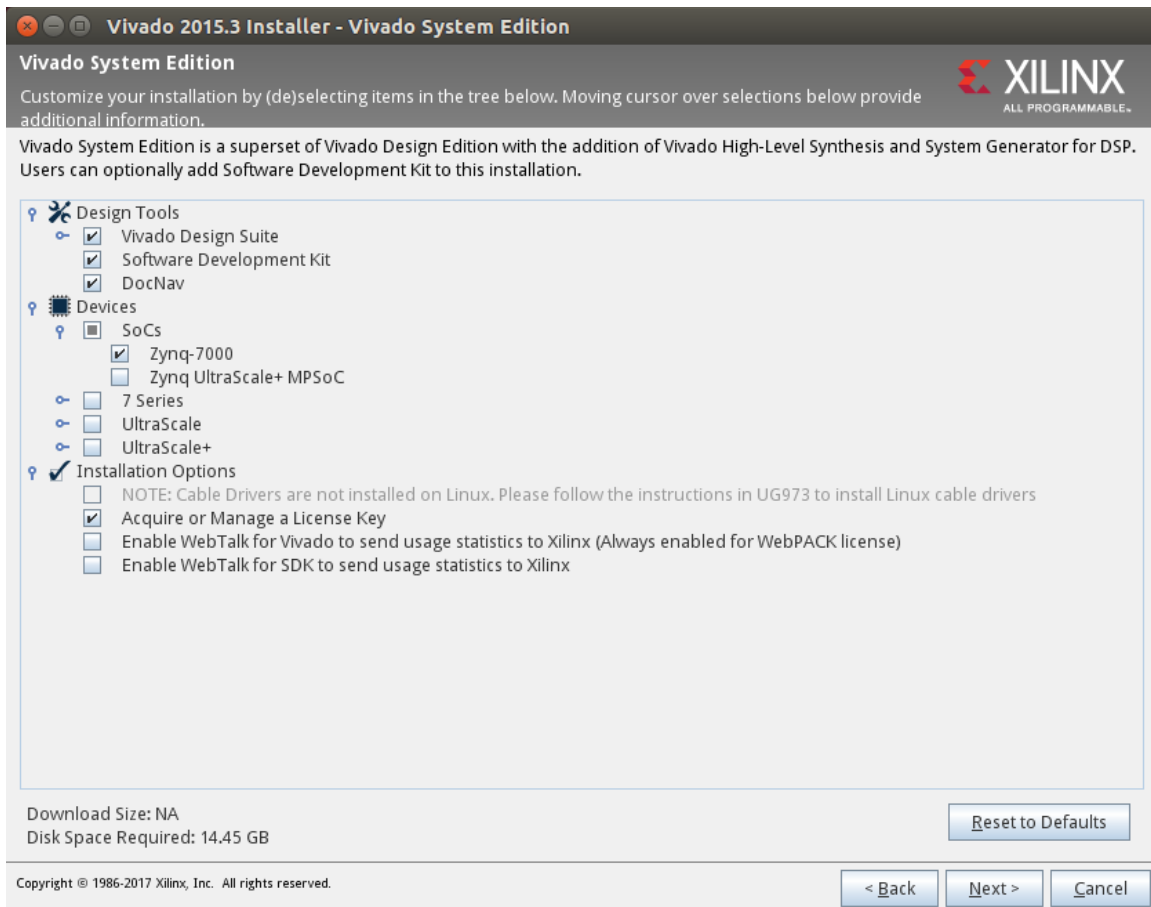


Figure I4 Design Tools and Device Support Selection in Vivado Installer

7. Select Acquire or Manage a License Key to be guided through the license management steps; after the installation is complete
8. Keep the default installation directory and click Next

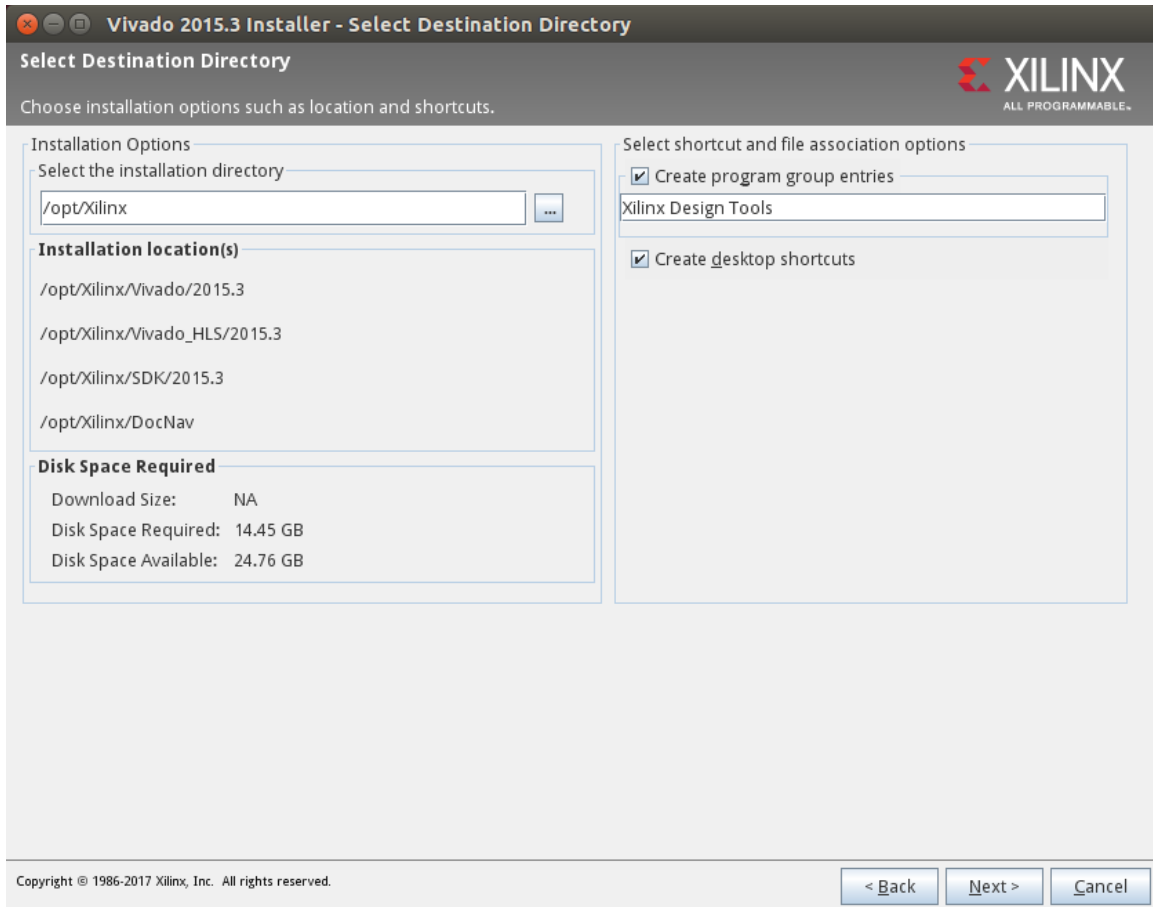


Figure I5 Installation Directory in Vivado Installer

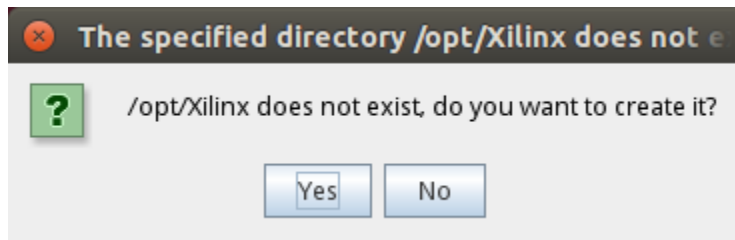


Figure I6 Create Installation Directory Prompt in Vivado Installer

9. Review the Installation Summary, then click Install

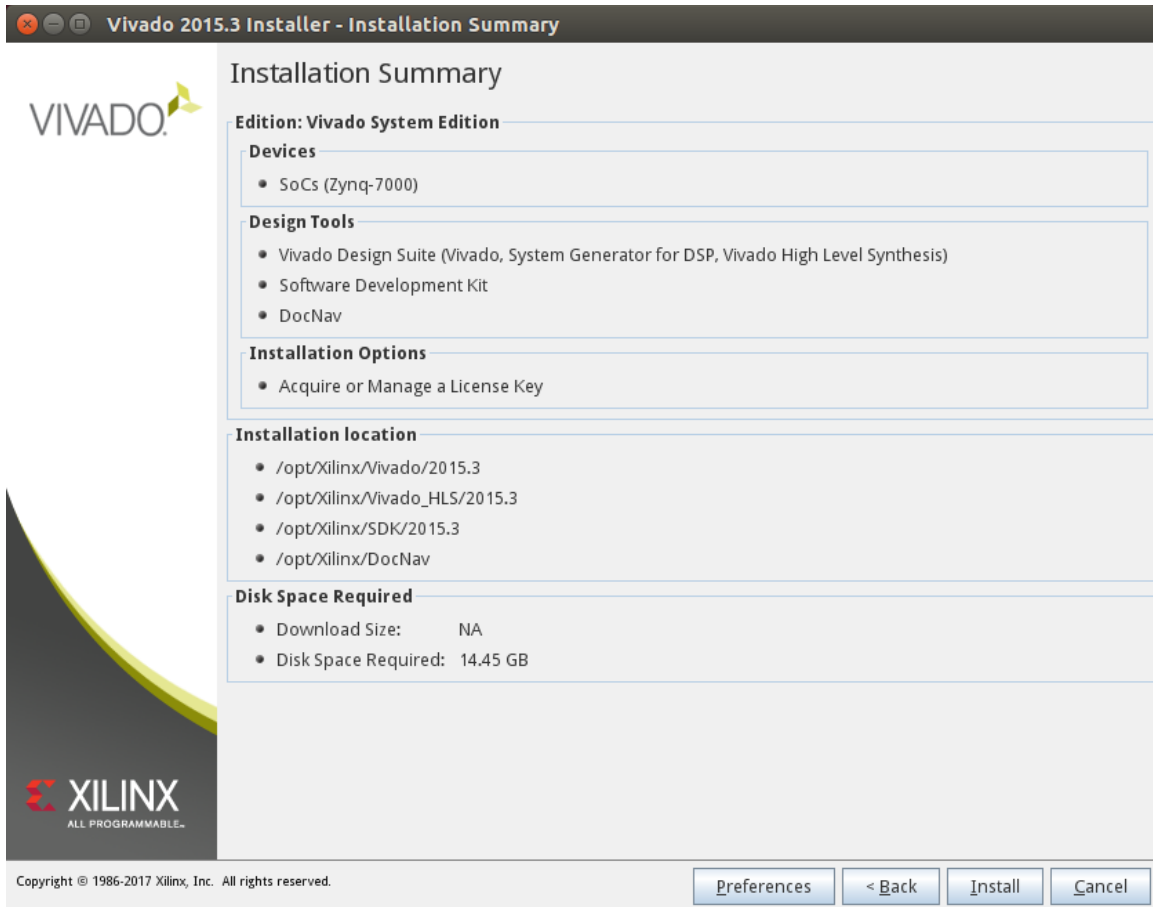


Figure I7 Installation Summary Review in Vivado Installer

10. The installation should take anywhere between 15-20 minutes



Figure I8 Vivado Installation Progress

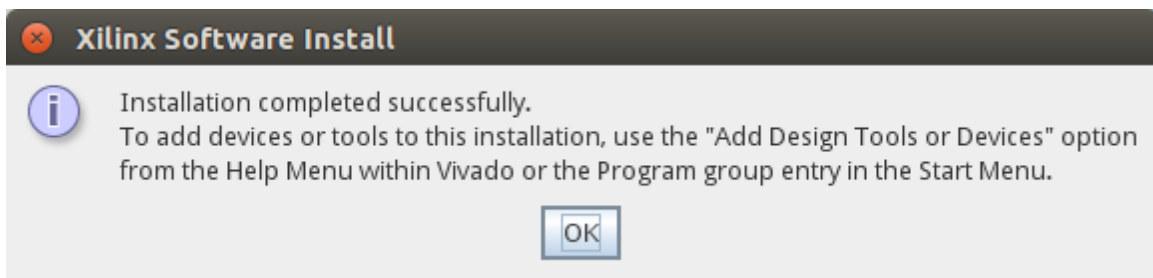


Figure I9 Vivado Installation Complete

11. Once the installation is complete, the UI would prompt the user for the licence management. If the license is yet to be obtained, select Obtain License and follow the GUI as well as the instructions that came with the license voucher. If the license was already obtained from other sources, or the user is performing a re-install then select Load License and browse to the license file

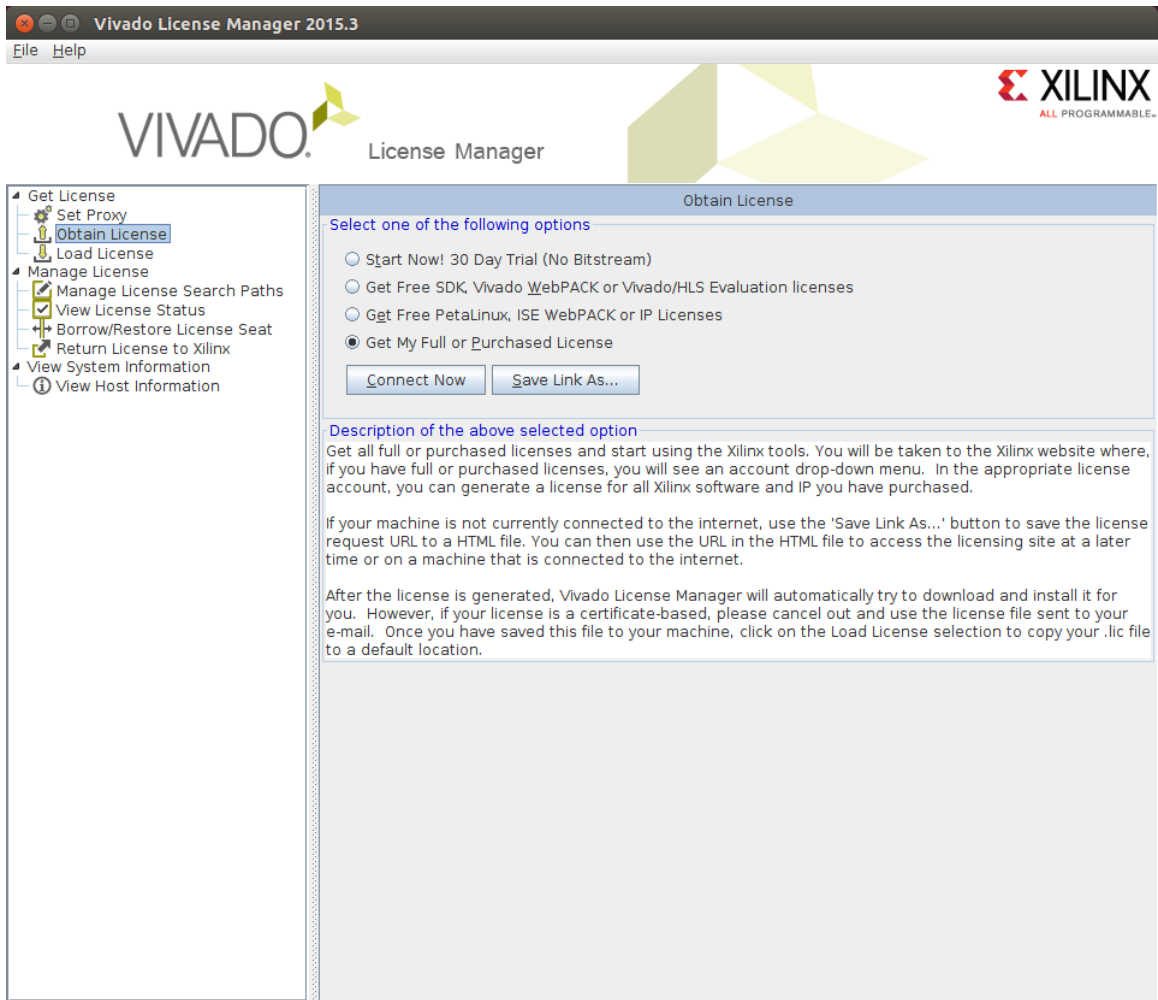


Figure I10 Vivado License Manager – Obtain License

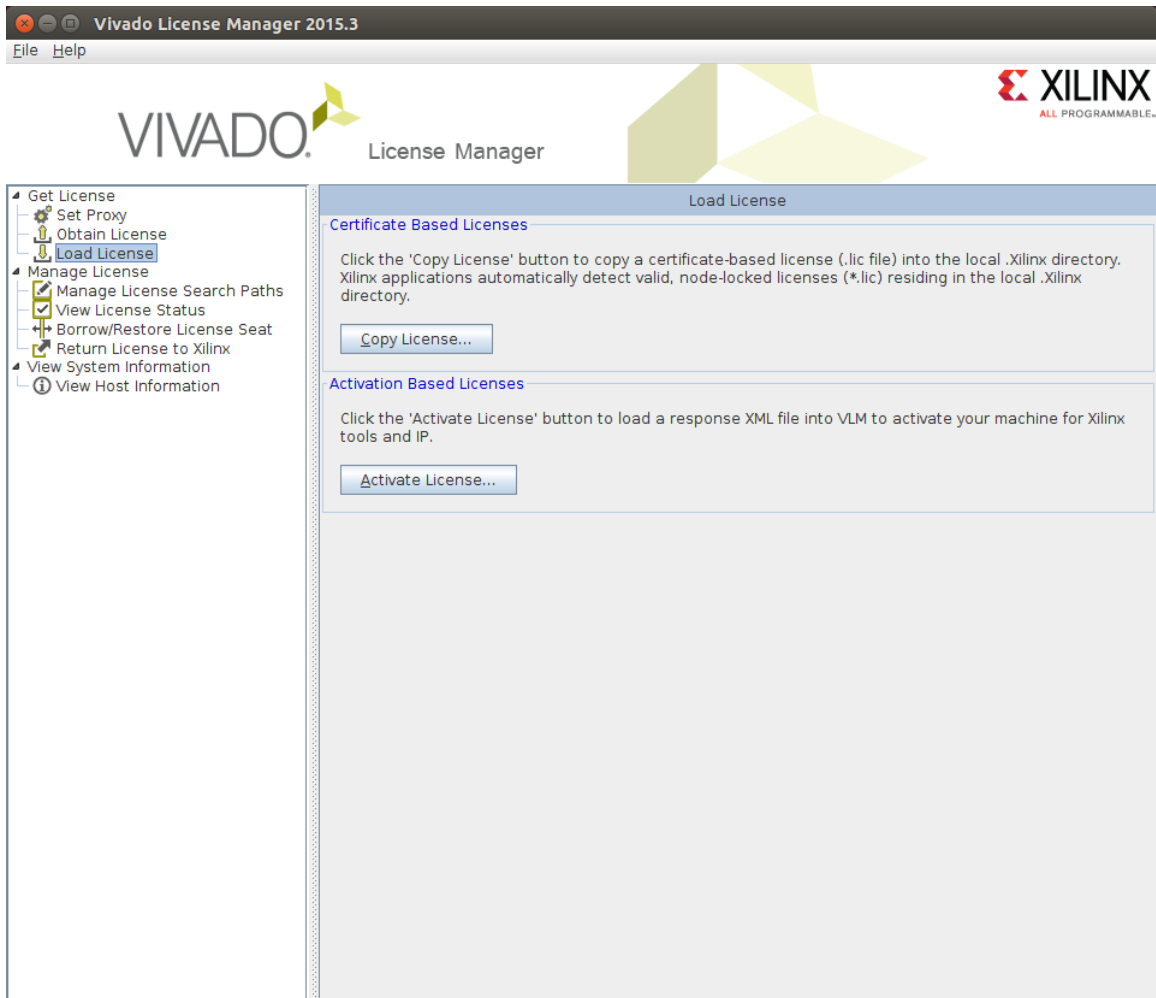


Figure I11 Vivado License Manager – Load License

12. To install the Xilinx cable drivers on Linux (cable drivers are automatically installed during the Vivado suite installation process under Windows), assuming the user has installed Vivado version 2015.3 at the default directory, open a terminal and execute the following commands:

- a. `cd`
`/opt/Xilinx/Vivado/2015.3/data/xicom/cable_drivers/lin64/install_script/install_drivers`
- b. `sudo ./install_drivers`
- c. `reboot`

This completes the Vivado suite installation.

Appendix J. USB Serial (COM) Port Properties

To access the UART of a given FPGA evaluation board, first connect the USB cable to the UART port on the FPGA board and the other end to a USB port on the hardware server. Open a UART terminal window (such as Tera Term), and create a serial port connection based on the port number attributed to the ZedBoard, with the following settings: 115200, 8, none, 1, none.

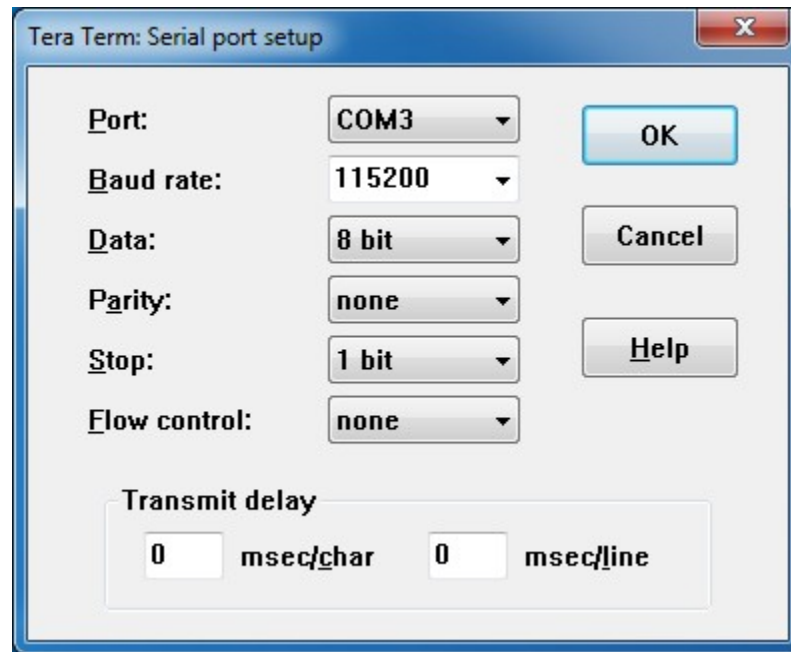


Figure J1 Zynq-7000 FPGA UART Serial Port Setup

The settings shown above are also applicable for use with C-Kermit; (installation details show in Appendix E)

Appendix K. Vivado Zynq-7000 FPGA Design Guide A: LED Binary Counter

This is a full step-by-step guide that shows how to create an FPGA design of a simple LED binary counter application using the ZedBoard which uses Xilinx's Zynq-7000 series FPGA. All the Vivado design guides shown in the appendices assume that the user has gone through the basic guides for the ZedBoard mentioned in section 5.1.

1. Launch the Vivado IDE and select Create New Project



Figure K1 Create a New Project in Vivado

2. Enter a project name of your choice in the Project Name field and choose a Project Location, and click Next

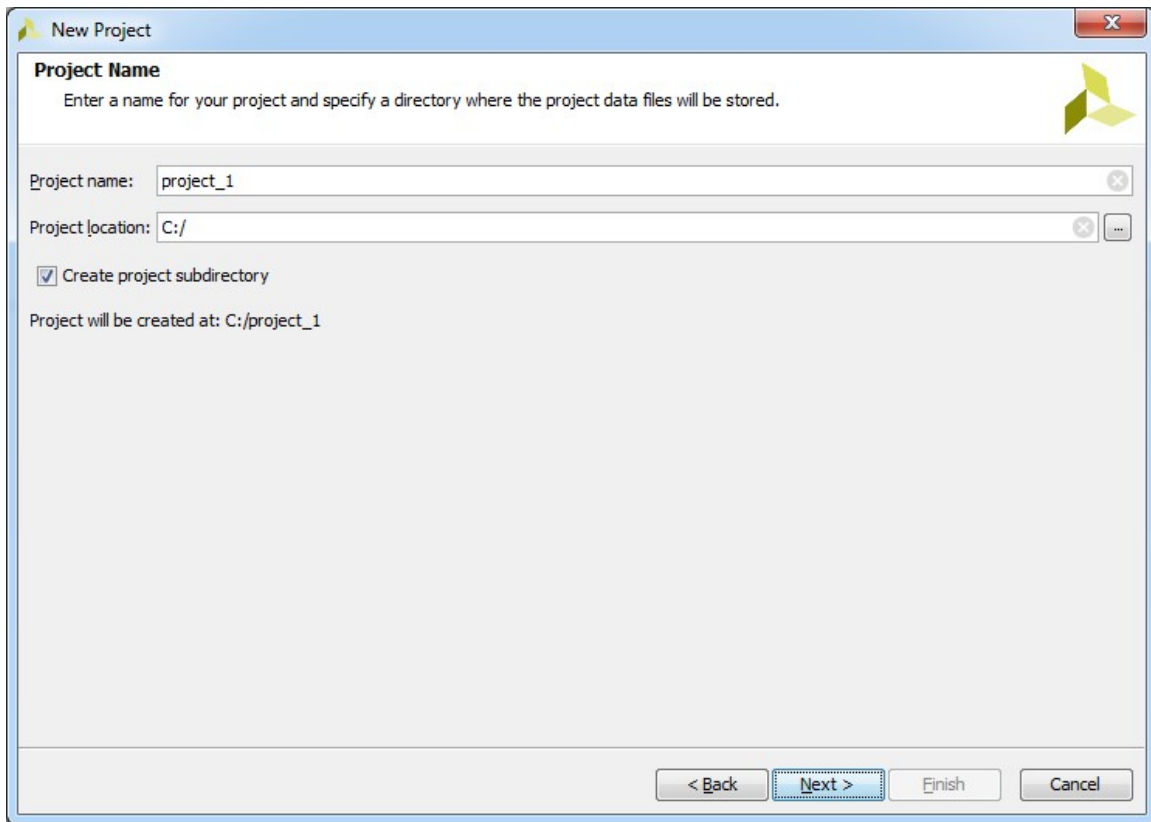


Figure K2 Specify the Project Name and Location in Vivado

3. Select RTL Project option in the Project Type form, and click Next

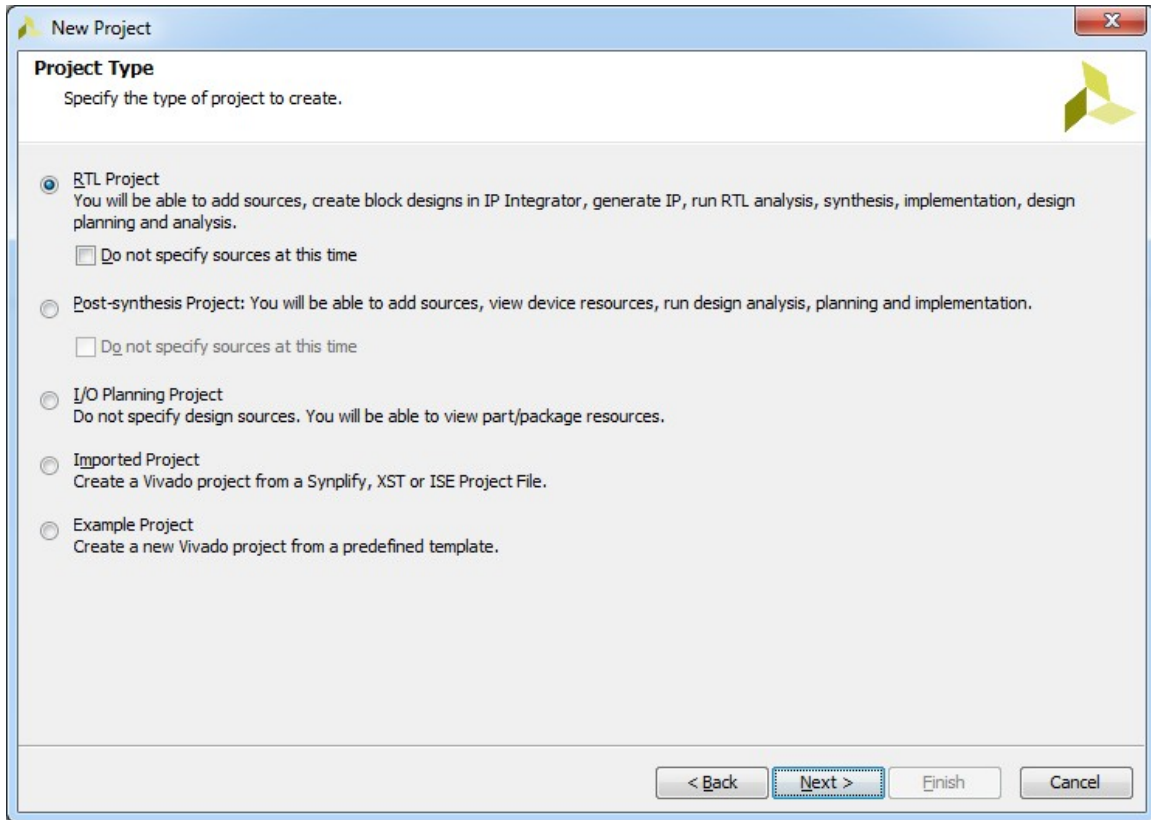


Figure K3 Set RTL as the Project Type in Vivado

4. In the Add Sources window, select VHDL as the Target Language and Simulator Language. Click Next

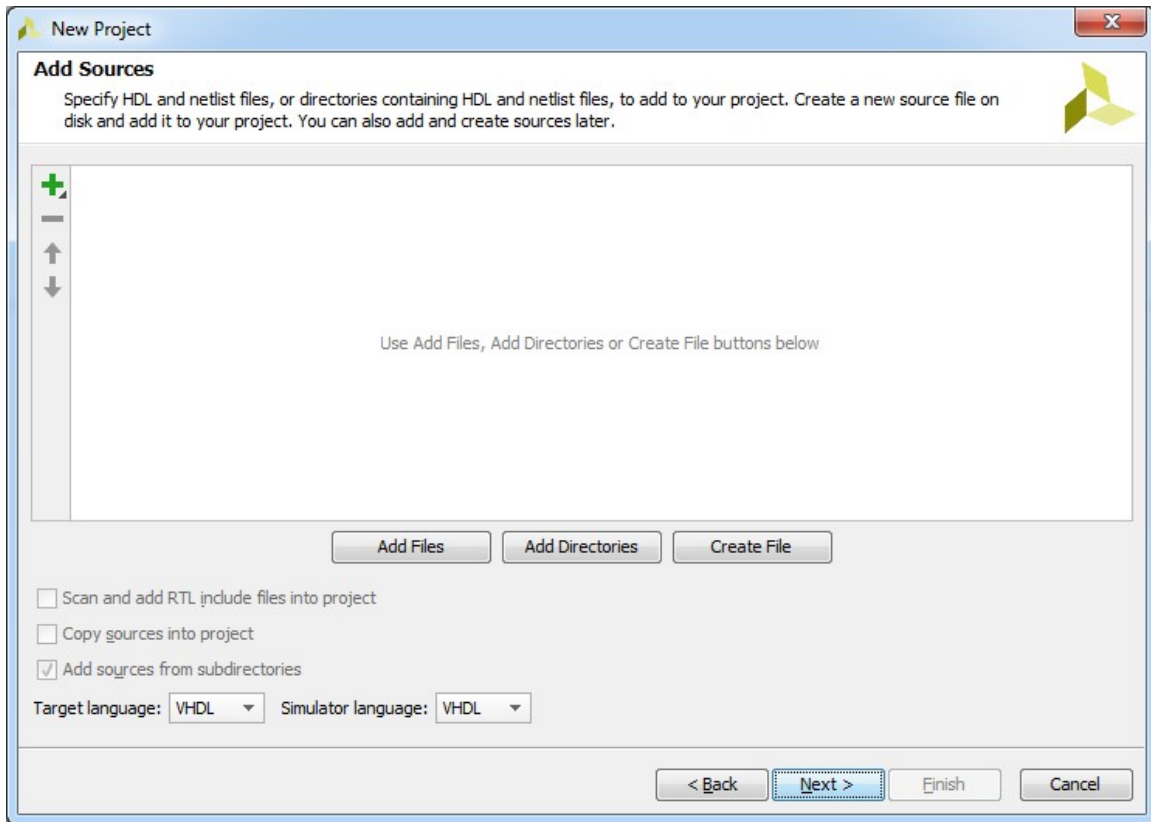


Figure K4 Set the Target and Simulator Language to VHDL in Vivado

5. There won't be a need to add files, IPs or constraints so click Next three times
6. In the Default Part windows, click the Boards icon and choose ZedBoard Zynq Evaluation and Development Kit, and click Next then Finish

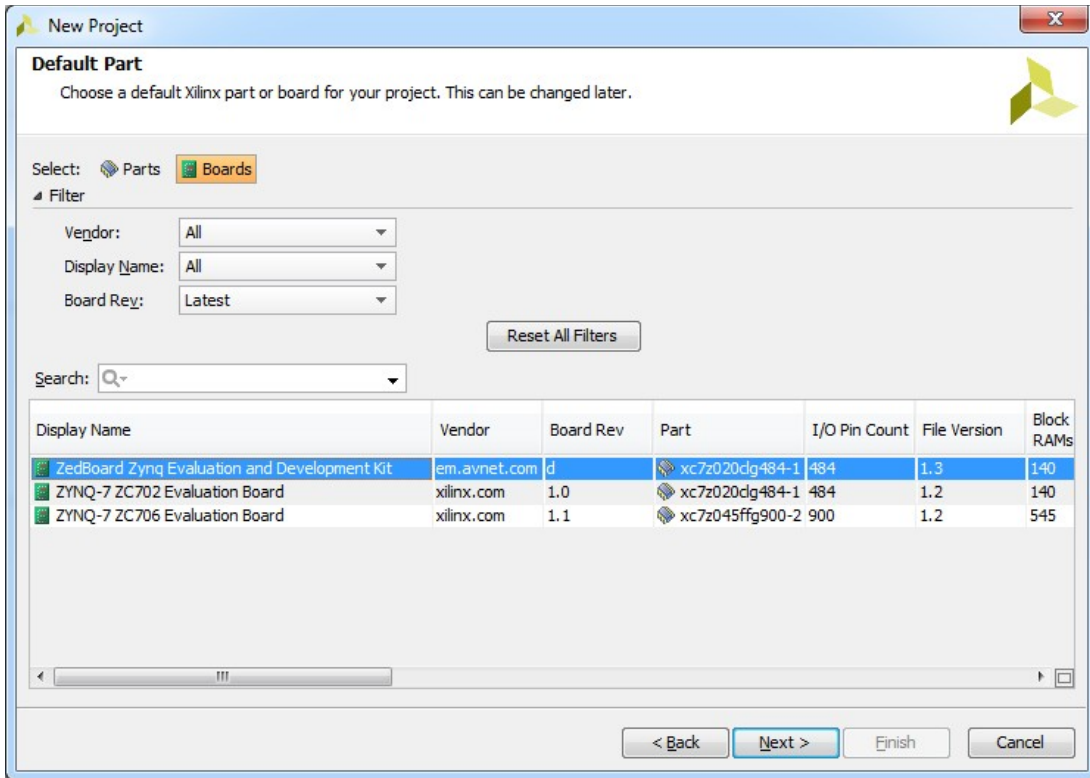


Figure K5 Select Boards: ZedBoard Zynq in Vivado

7. In the Flow Navigator expand the IP Integrator and click Create Block Design.
A Create Design window will pop up, keep the same name

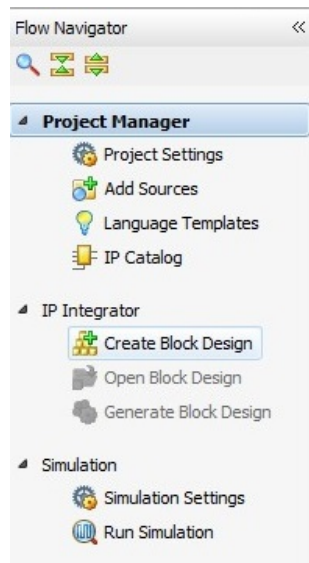


Figure K6 In the Flow Navigator: Create Block Design in Vivado

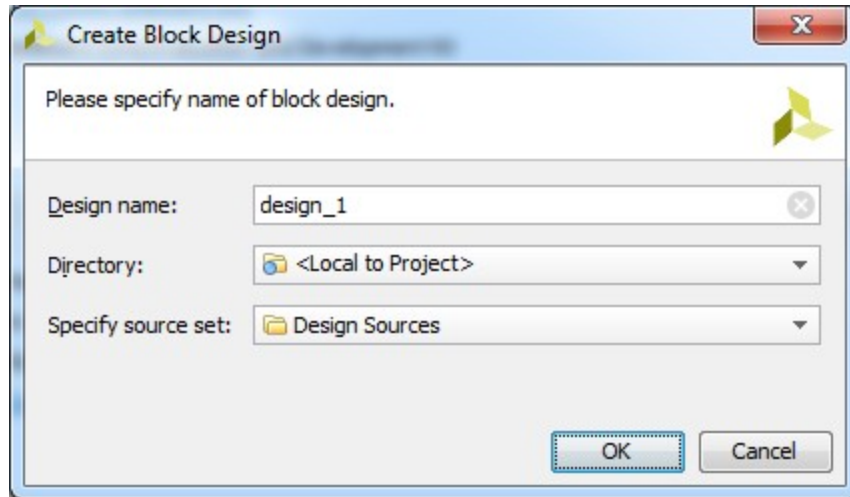


Figure K7 Keep the Default Design Name in Vivado

8. After the Diagram window opens, click on the Add IP icon from the left panel of the Diagram page to add IP blocks. Alternatively, one could right-click on any empty area in the Diagram section and choose Add IP

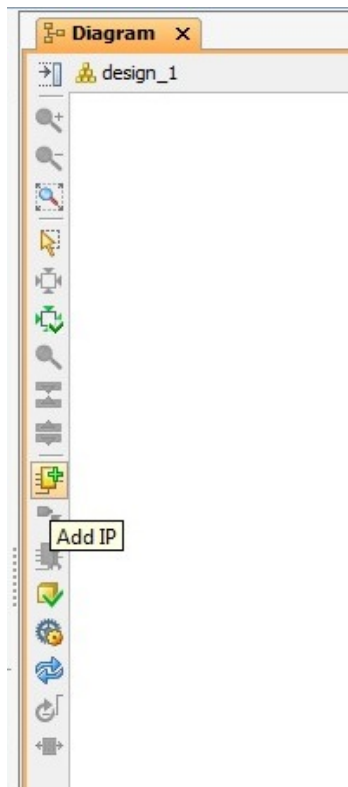


Figure K8 Add IP in Vivado

9. In the search bar type Zynq, and choose (double-click) ZYNQ7 Processor System, this will add a Zynq Processing System IP(the ARM Cortex A9 processor)

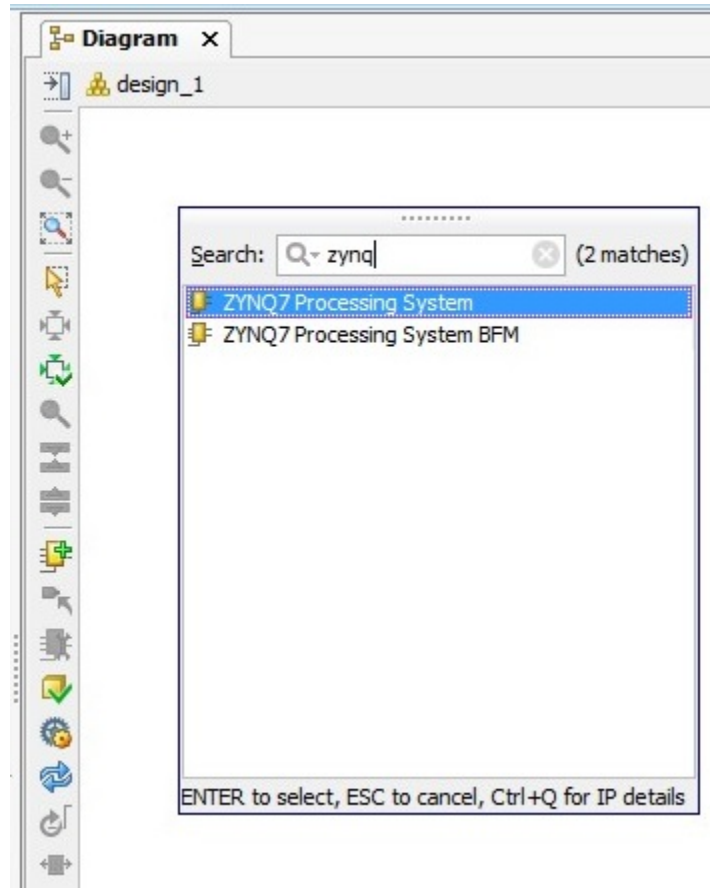


Figure K9 Select Zynq7 Processing System IP in Vivado

10. Click on Run Block Automation. This would configure the Zynq IP according to the default settings per the selected board (The ZedBoard). This is called Design Assistance, and it appears whenever there is a possibility that Vivado could automate some design steps. Keep the default settings and click OK

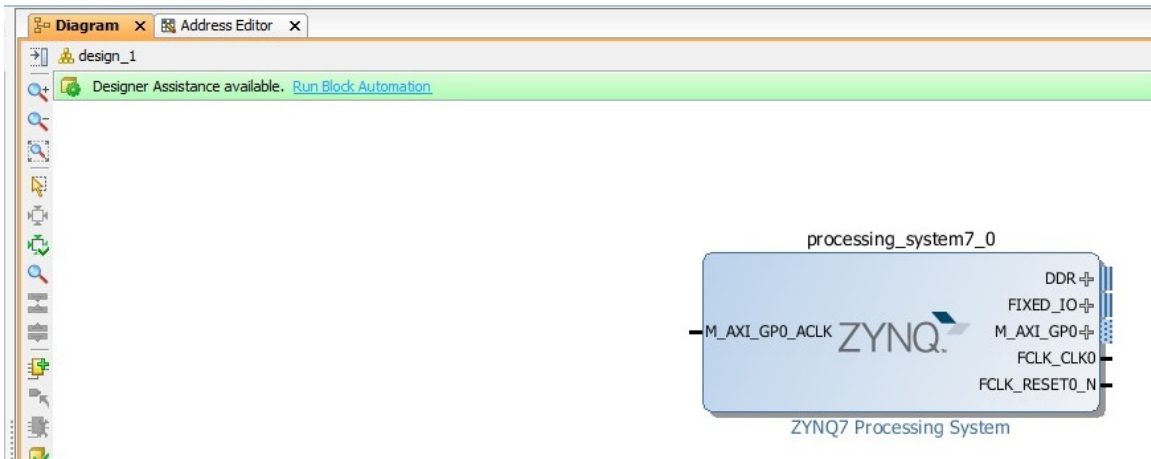


Figure K10 Run Block Automation in Vivado

11. Again click on the Add IP icon and this time; in the search bar type GPIO, and choose the AXI GPIO. The AXI GPIO IP should appear in the Diagram

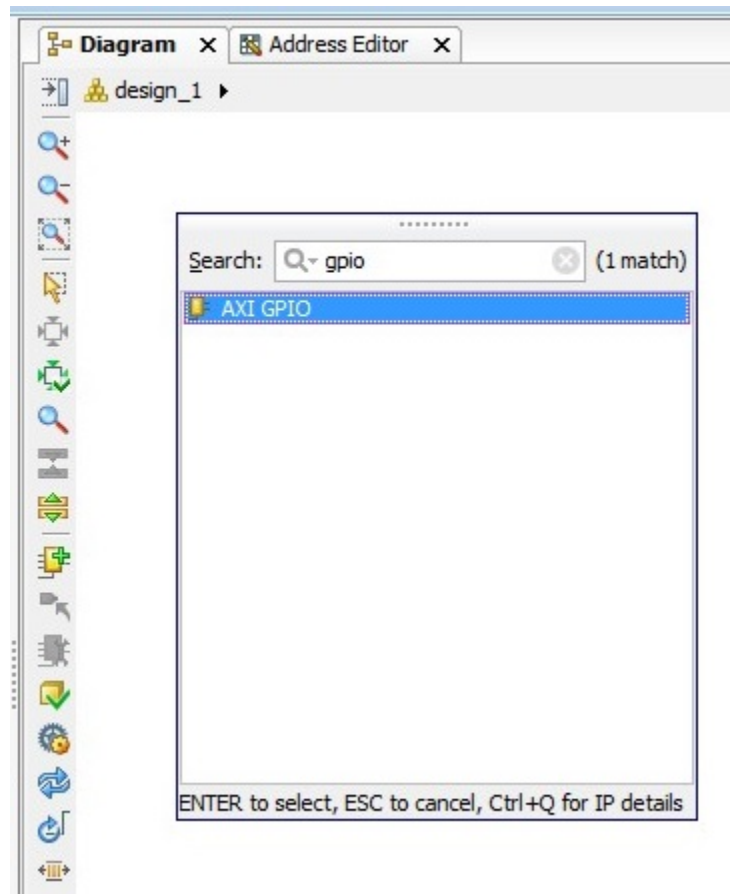


Figure K11 Add AXI GPIO IP in Vivado

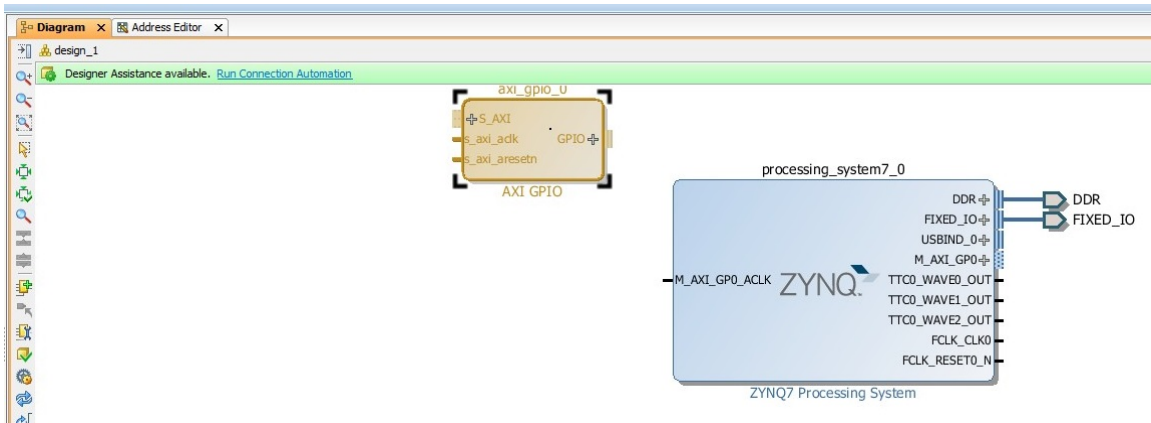


Figure K12 Double-Click the AXI GPIO for Further Configuration in Vivado

12. Double click on the AXI GPIO IP, under Board Interface choose leds 8bits and click OK

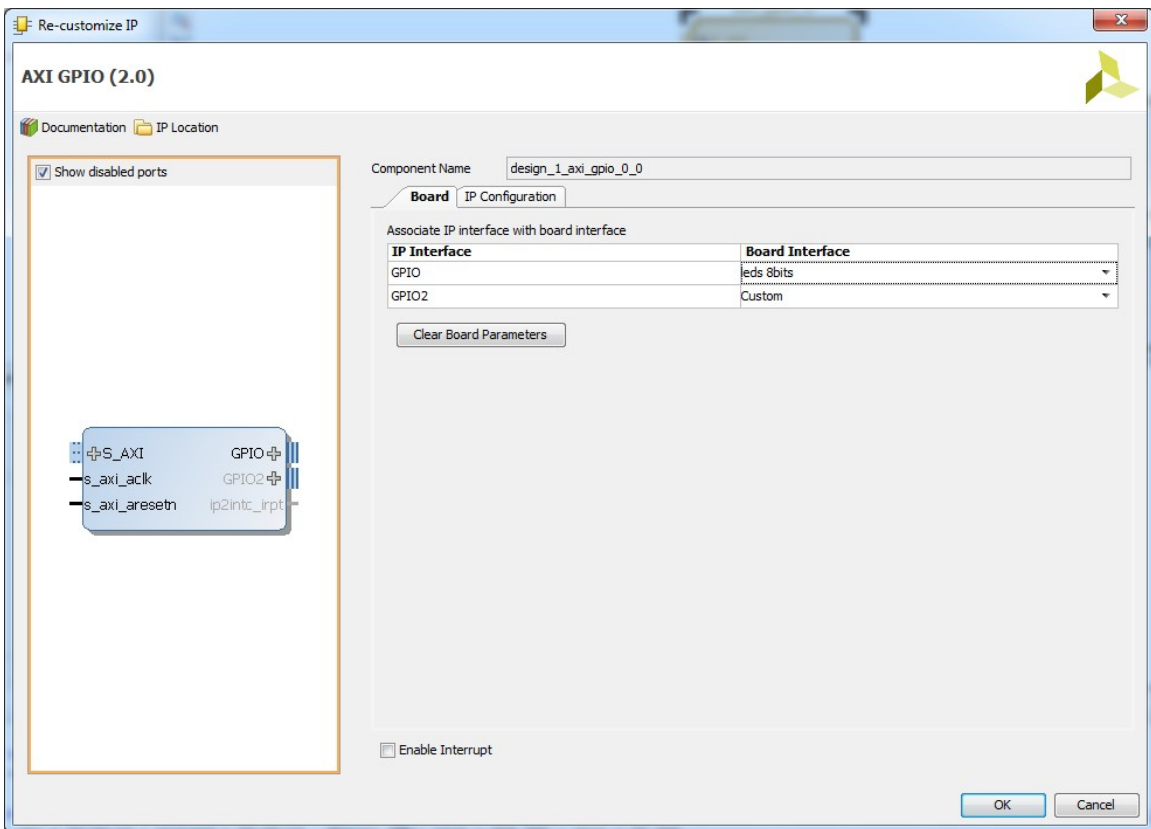


Figure K13 Set the Board Interface for GPIO to LEDs 8Bits in Vivado

13. Now the design assistance will again show the option to run Connection Automation, Click on it and choose All Automation then click ok. Here, one is

accepting the suggestion of connecting the AXI GPIO IP to the Zynq. Vivado will add 2 IPs, these IPs are the processor system reset and AXI interconnect

- a. The processor system reset provides the reset signals to all peripherals and interconnects in the PL side of the Zynq according to reset signal given from the Zynq PS
- b. The AXI interconnect is responsible for creating an interface between the Zynq PS master interface GP port and the AXI GPIO IP

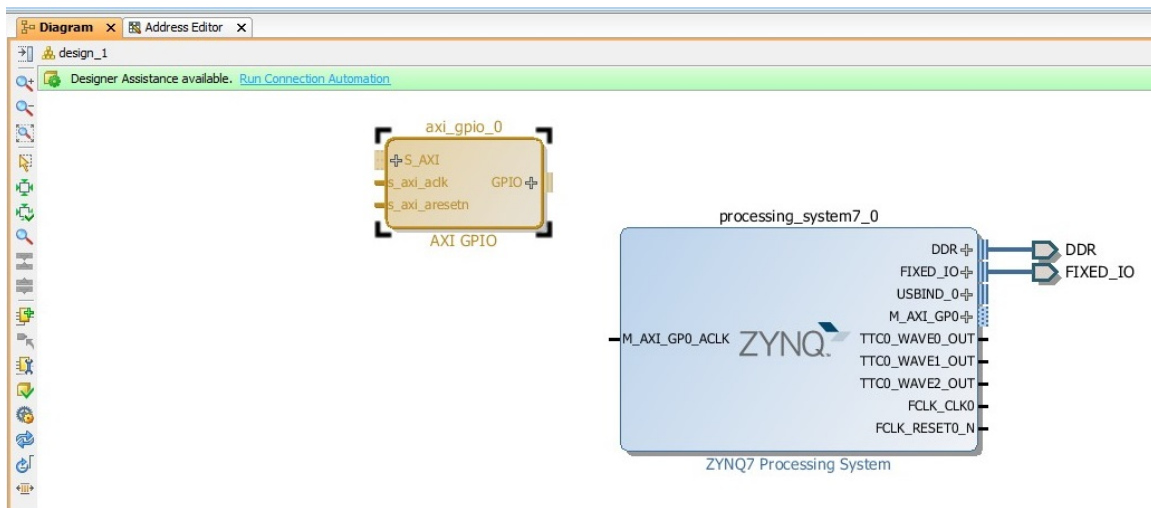


Figure K14 Run Connection Automation in Vivado

14. Click on the Regenerate Layout icon to rearrange the blocks. This is purely a visual rearrangement to fit the window/screen

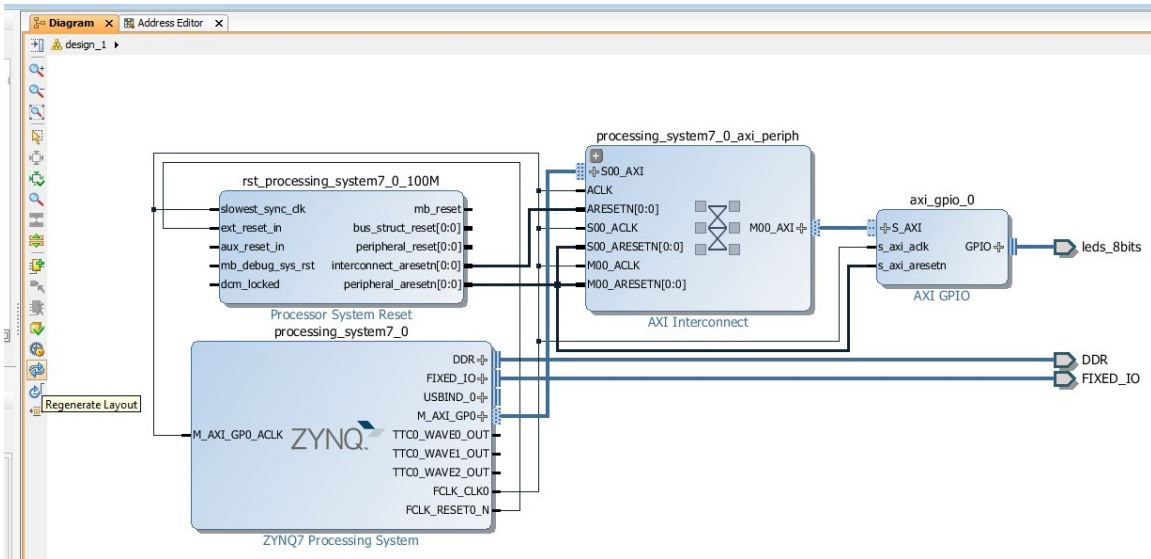


Figure K15 Regenerate Layout in Vivado

15. In the Design window, click on the Sources tab, right-click on the design_1 block design under Design Sources folder and choose create HDL wrapper, this options creates a top level VHDL file for the block design which can be used to generate the bitstream file

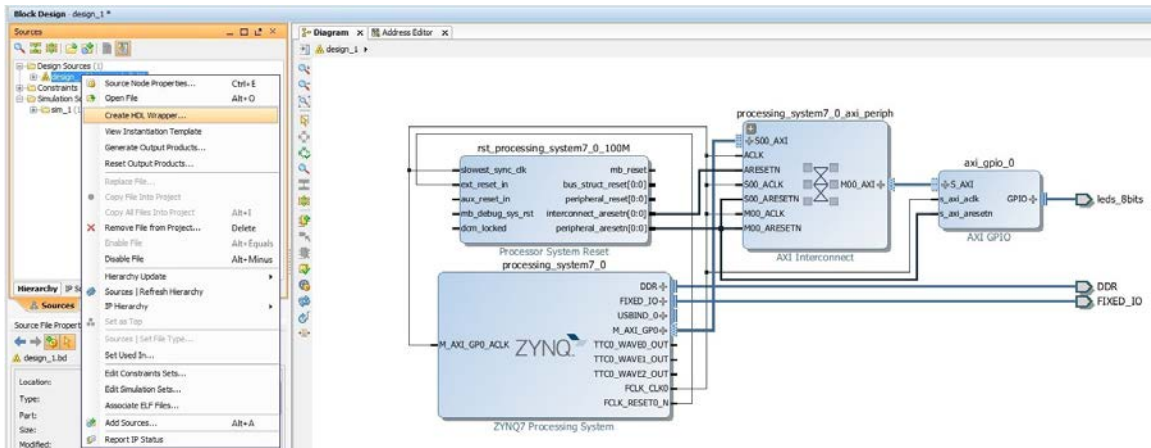


Figure K16 Create HDL Wrapper in Vivado

16. Keep the default setting Let Vivado manage wrapper and auto-update and click OK

- c. Note: Click on the Address Editor, one shall find the processing system connected to the AXI GPIO IP, with an automatically given

offset address. This address will be used by the Zynq processor to communicate with the AXI GPIO IP which is connected to the LEDs. In our example the offset address is 0x41200000

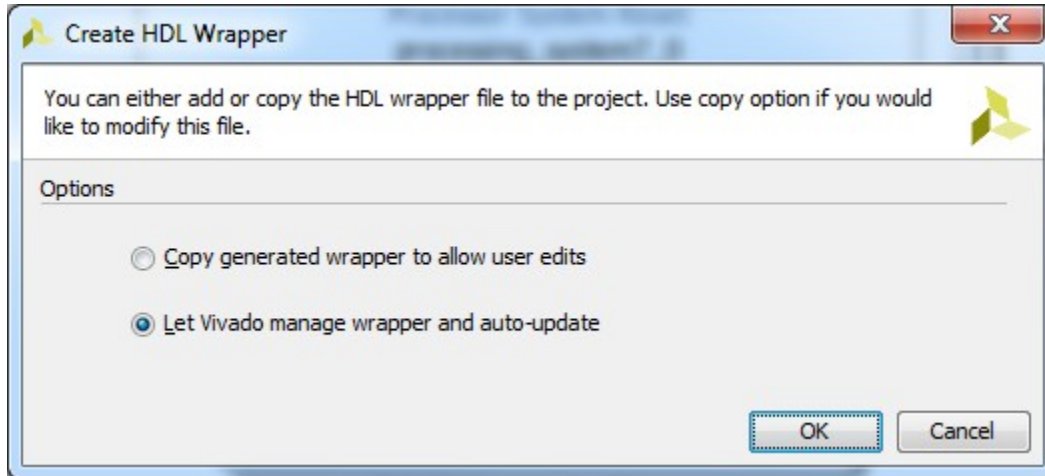


Figure K17 Allow Vivado to Manage the Wrapper and Auto-Update

17. In the Flow Navigator, under Program and Debug click on Generate Bitstream. Alternatively, from the top menu Flow → Generate Bitstream. During this step the tool creates a bit file for programming the PL of the Zynq. Accept to save the project

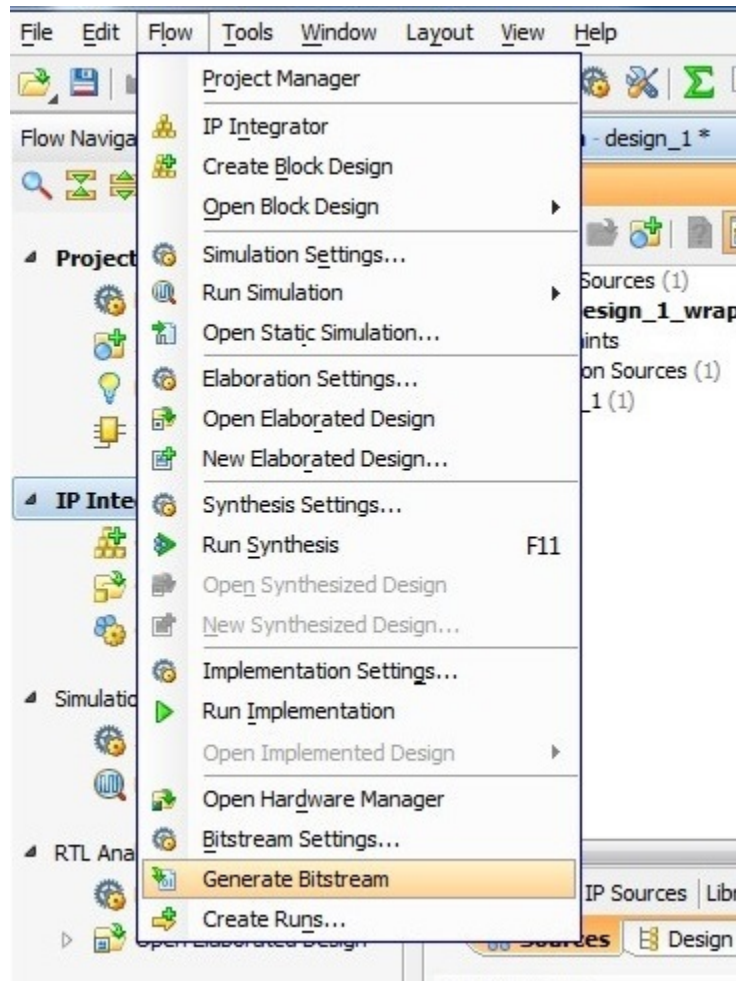


Figure K18 Generate the Bitstream in Vivado

18. Click Yes to No Implementation Results Available. This would synthesize the design. Depending on the PC's available resources and specs, the process should take a few minutes, potentially longer on slower PCs

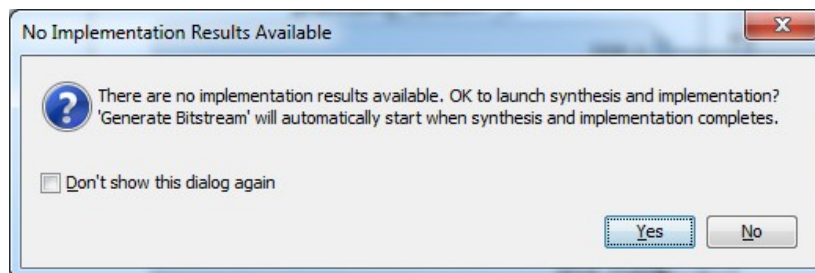


Figure K19 Allow Vivado to Launch Synthesis and Implementation

19. Bit Generation Complete message should pop up, close/cancel it then go to File → Export → Export Hardware. Check the Include bitstream and click ok. This exports the hardware files generated to the SDK

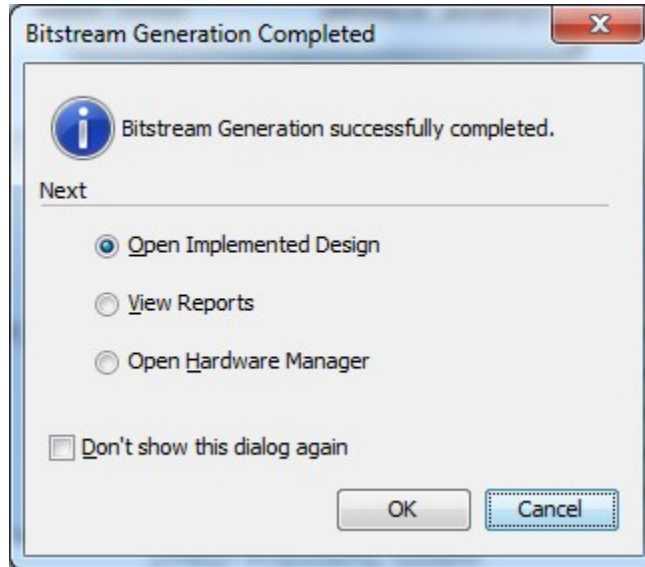


Figure K20 Bitstream Generation Successful in Vivado

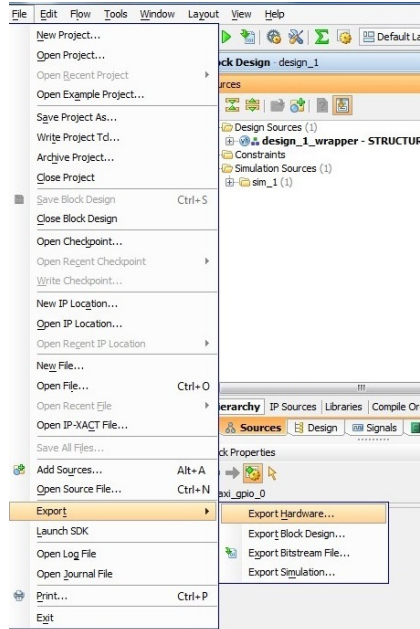


Figure K21 Export the Hardware Design to the SDK from Vivado

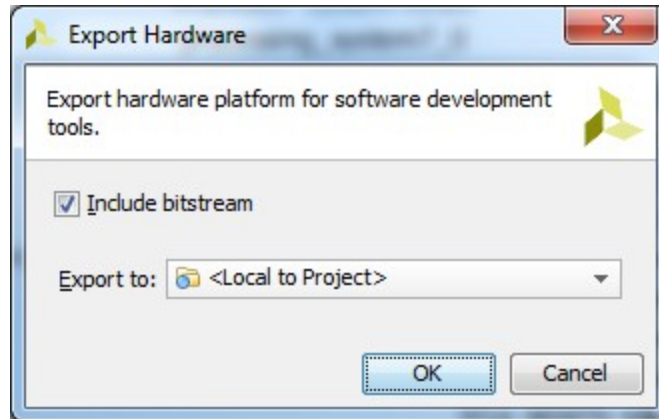


Figure K22 Include the Bitstream in the Export from Vivado

20. In Vivado go to File → Launch SDK, then click OK. The SDK will open and in the Project Explorer window one could see the exported hardware files. In the SDK go to Files → New → board support package then click finish. This creates a board support package with all the necessary functions for driving the hardware that was designed in Vivado; in the previous steps

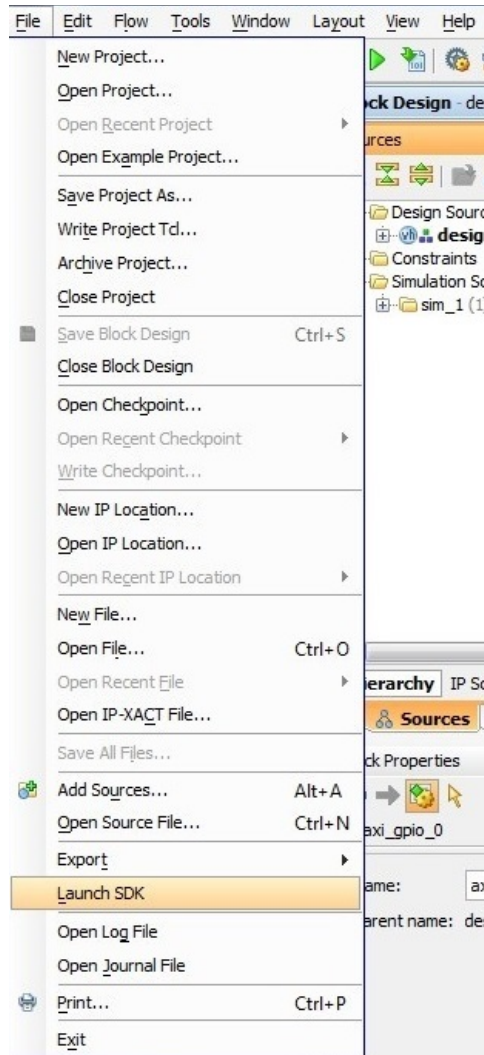


Figure K23 Launch SDK from Vivado

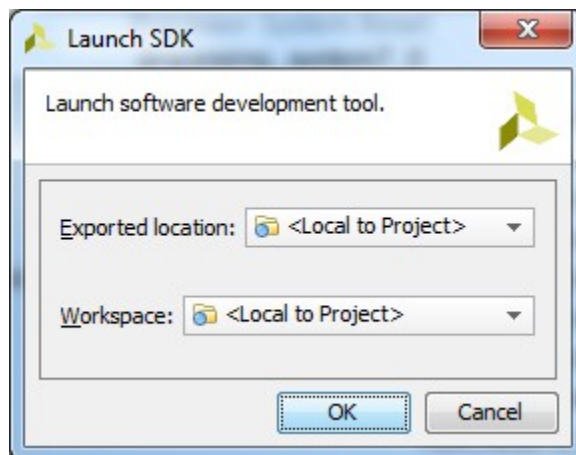


Figure K24 Use Default Location When Launching SDK from Vivado

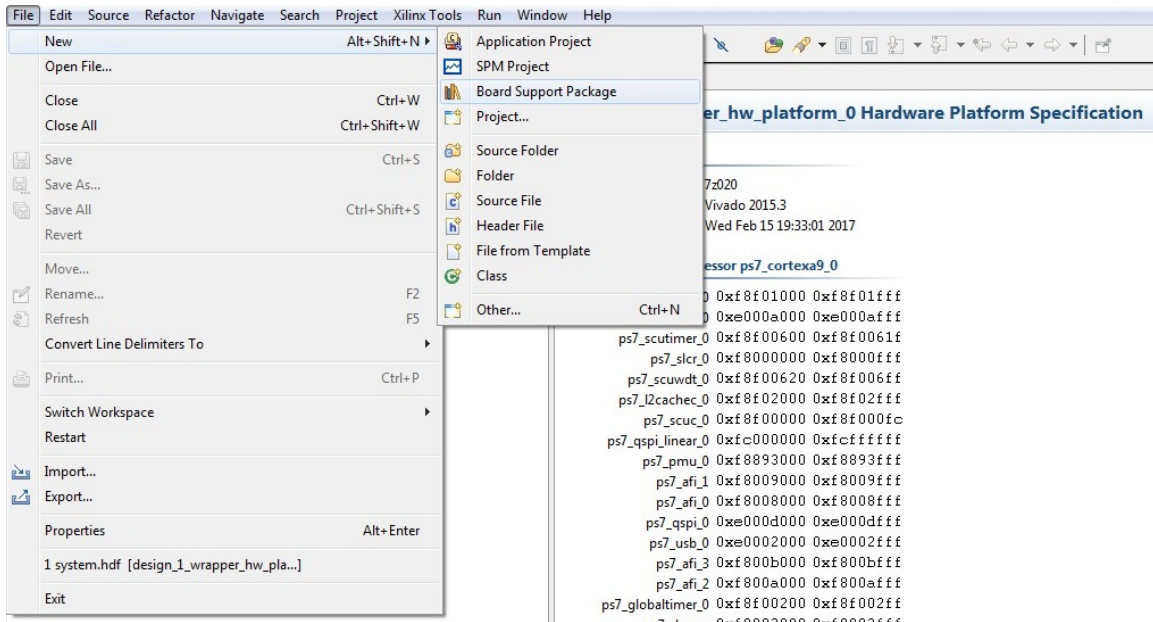


Figure K25 Create a New Board Support Package in SDK

21. The Board Support Package Settings window would pop up, keep all default settings and click Finish then OK

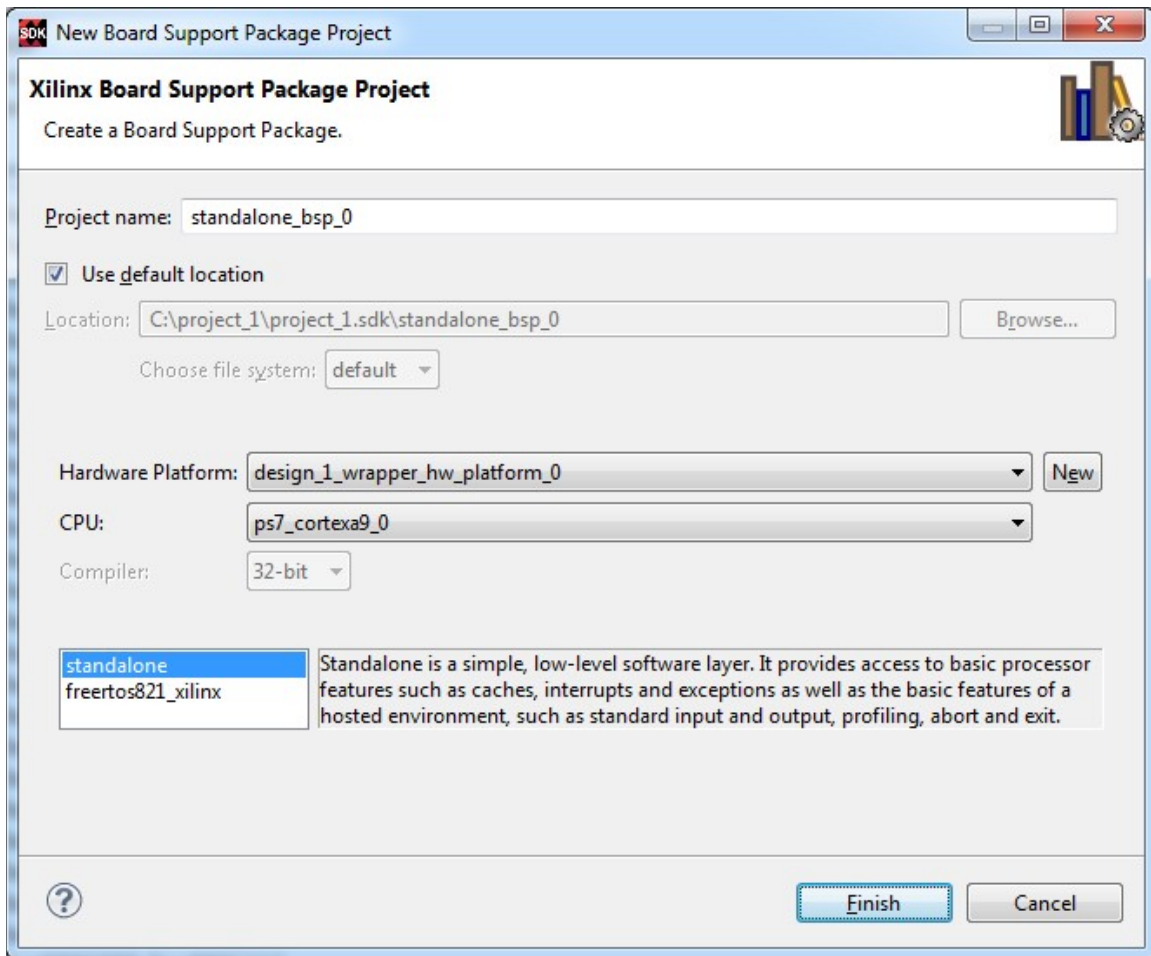


Figure K26 Use Default Board Support Package Settings in SDK

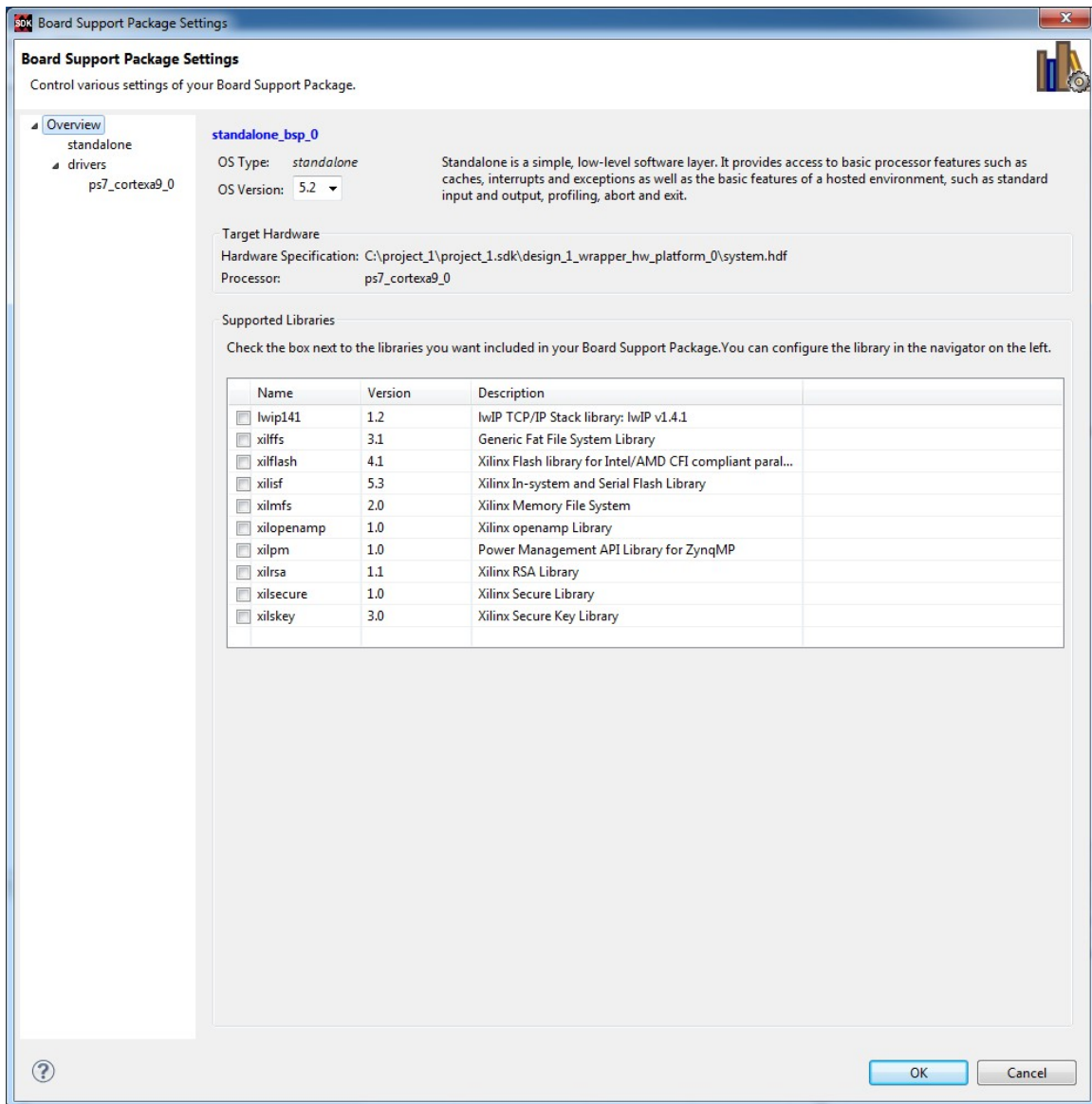


Figure K27 Use Default Board Support Package Standalone Settings in SDK

22. Create an application project by going to File → New → Application Project.
Enter a name in the project name field and under the Board Support Package choose Use existing then click Next

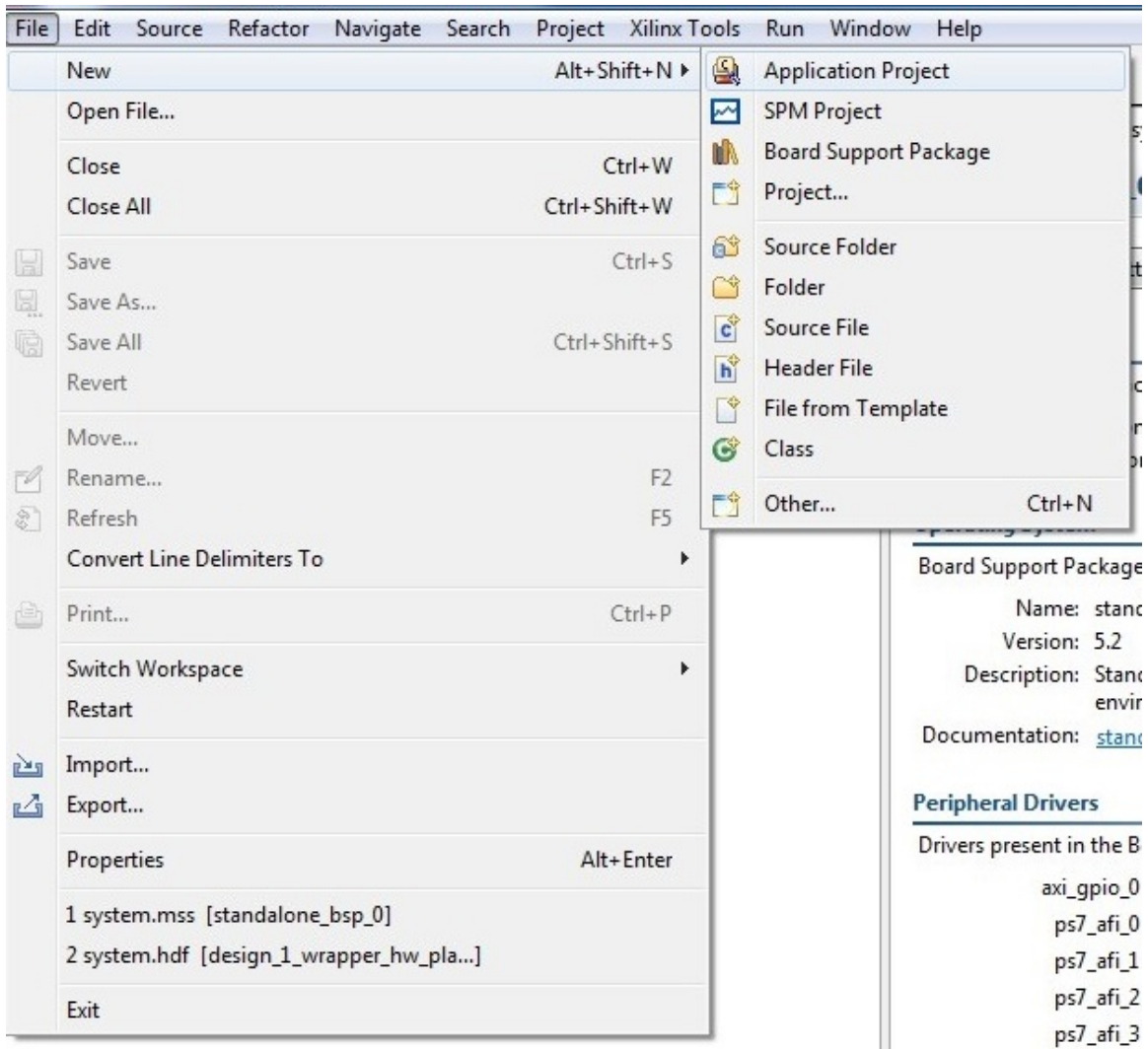


Figure K28 Create a New Application Project in SDK

23. Choose Hello World from the Available Templates then click Finish

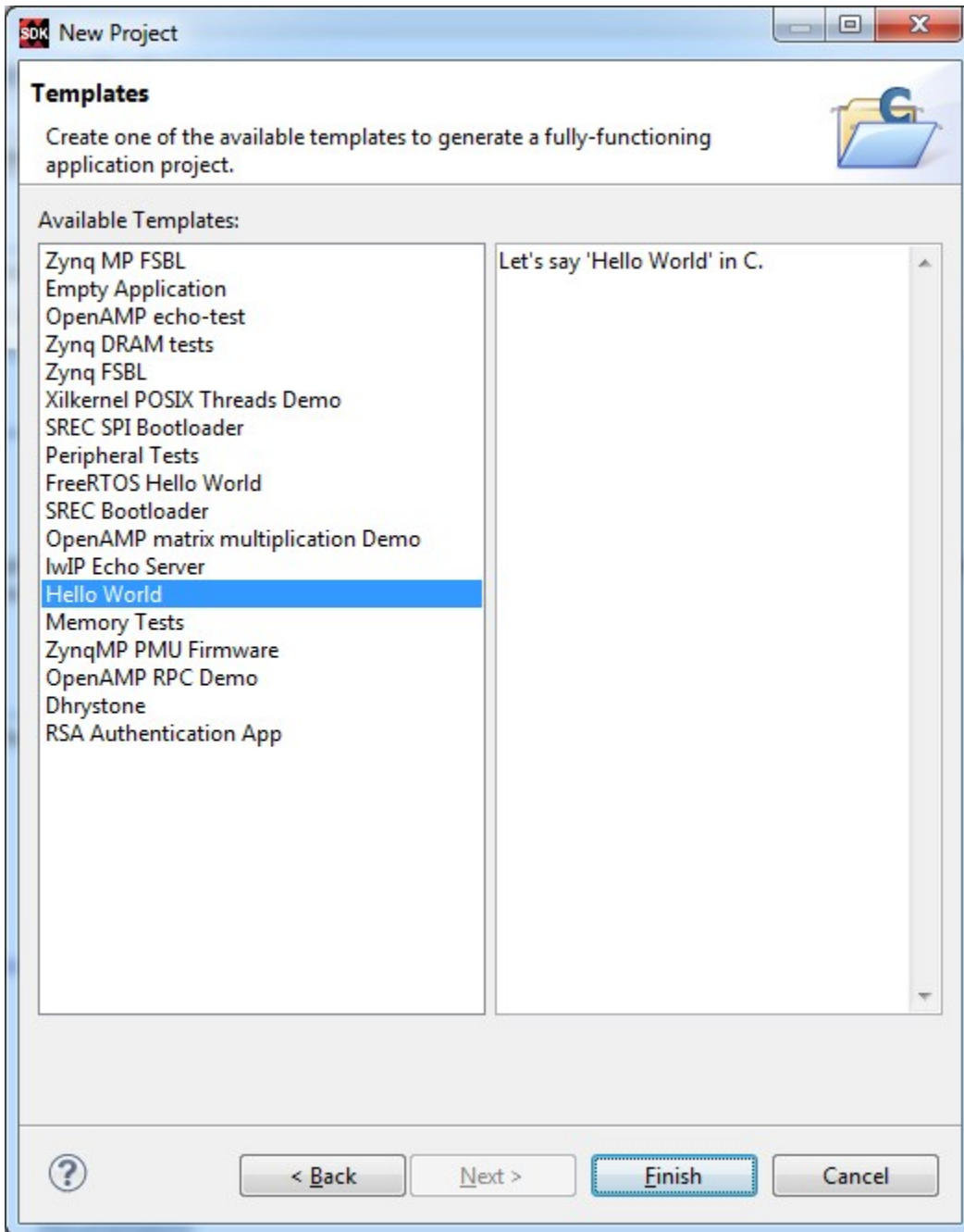


Figure K29 Select the Hello World Template in SDK

24. In the Project Explorer left panel, expand the src folder under the project_1 folder and double click on Helloworld.c. The source code will be visible in the main window. Replace the code within it with the following code:

```
1. #include <stdio.h>
2. #include "platform.h"
3. #include "Xil_io.h"
```



```

4. // delay function
5. void delay()
6. {
7.     int i;
8.     for(i=0;i<20000000;i++);
9. }
10. int main()
11. {
12.     init_platform();
13.     unsigned char counter=0;
14.     // infinite loop
15.     while(1)
16.     {
17.         // write to address 0x41200000(offset address
of the AXI GPIO:8 LEDs) the value of counter
18.         Xil_Out32(0x41200000,counter);
19.         // increment counter
20.         counter++;
21.         //delay
22.         delay();
23.     }
24.
25.     cleanup_platform();
26.     return 0;
27. }

```

The screenshot shows an IDE window with the following code:

```

#include <stdio.h>
#include "platform.h"
#include "Xil_io.h"
// delay function
void delay()
{
    int i;
    for(i=0;i<20000000;i++);
}
int main()
{
    init_platform();
    unsigned char counter=0;
    // infinite loop
    while(1)
    {
        // write to address 0x41200000(offset address of the AXI GPIO:8 LEDs) the value of counter
        Xil_Out32(0x41200000,counter);
        // increment counter
        counter++;
        //delay
        delay();
    }

    cleanup_platform();
    return 0;
}

```

Figure K30 Edit the Hello World Source Code

25. In the SDK click on the program FPGA icon then click program; to program the Zynq PL with the bitstream

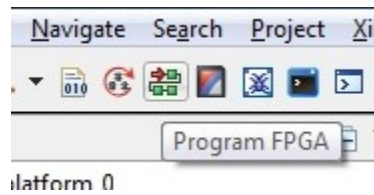


Figure K31 Program the FPGA in SDK

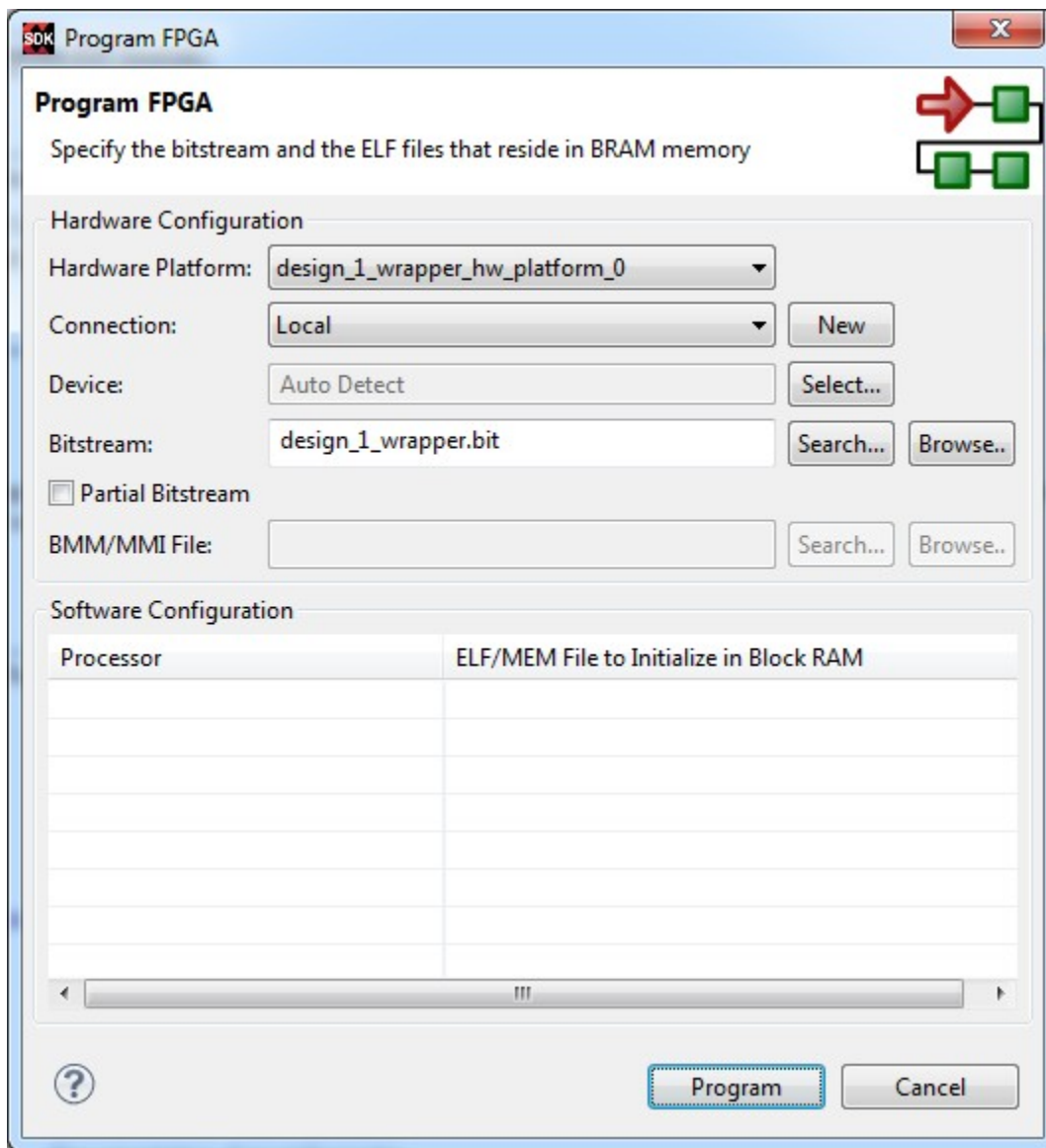


Figure K32 Keep Default Program Settings in SDK

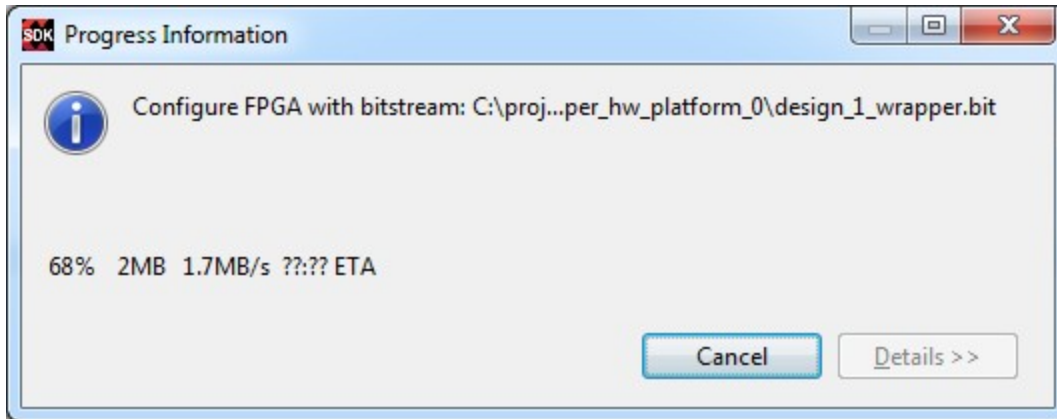


Figure K33 Programming Progress Bar in SDK

26. Right-click on the project_1 folder in the Project Explorer panel then go to Run As → 4 Launch on Hardware (GDB). The application should run on the board and the LEDs should show the binary values of the counter

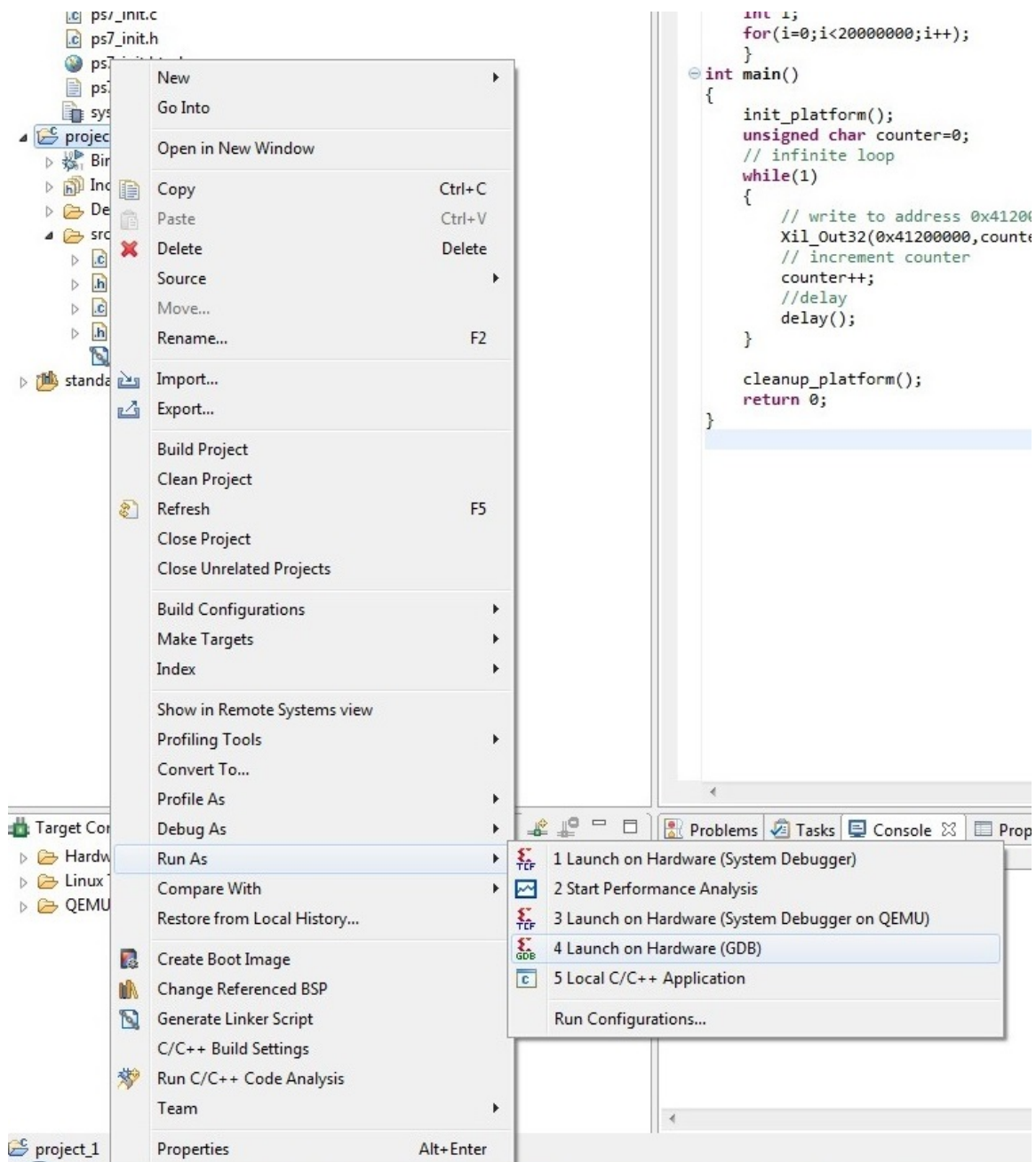


Figure K34 Launch the Application SW on the Hardware in SDK

This completes the design and software application of the LED binary counter.

Appendix L. Vivado Zynq-7000 FPGA Design Guide B: LED Scanner Light

This guide covers a variant of an LED software application that is used on the Zynq-7000 FPGA. While testing the web application, it was important to have at least two FPGA applications that are visually distinct, in order for the project developer to see that indeed the FPGA programming was successful, beyond the success messages on the terminal. For this purpose, the application developed in Appendix K is slightly modified to obtain a different visual output from the LEDs; on the FPGA evaluation boards.

1. Follow the guide in Appendix K all the way to and including step 24. However, in this case use the following source code:

```
1. #include <stdio.h>
2. #include "platform.h"
3. #include "Xil_io.h"
4.
5. // delay function
6. void delay()
7. {
8.     int i;
9.     for(i=0;i<8000000;i++);
10. }
11.
12. int main()
13. {
14.     init_platform();
15.     int bit=1;
16.     int i;
17.
18.     // clear screen and display demo name on uart
19.     printf("%c[2J",27);
20.     printf("\n\r\n\rLED    KNIGHT    RIDER    EFFECT:    8
    LEDs\n\r");
21.
22.     // infinite loop
23.     while(1)
24.     {
25.         // write value to LEDs, then shift bit to the
left, then Loop 8 times
26.         for (i=0; i<7; i++) {
27.             Xil_Out32(0x41200000,bit);
28.             printf(" ");
29.             fflush(stdout);
30.             bit<<=1;
31.             delay();
32.         }
```

```

33.
34.         // write value to LEDs, then shift bit to the
           right, then Loop 8 times
35.         for (i=0; i<7; i++) {
36.             Xil_Out32(0x41200000,bit);
37.             printf("\b");
38.             fflush(stdout);
39.             bit>>=1;
40.             delay();
41.         }
42.     }
43.
44.     cleanup_platform();
45.     return 0;
46. }

```

2. Then follow the guide in Appendix K from step 25 onwards

This completes the design and software application of the LED scanner light.

Appendix M. Vivado Zynq-7000 FPGA Design Guide

C: LED – UART IO

This guide covers another variant of the LED software application that is used on the Zynq-7000 FPGA. However, in this software application, the UART was added to the application to allow for an IO user interaction. Once the FPGA evaluation board's UART is accessible through the serial port terminal, the user could press any key on the keyboard/terminal, and the ASCII value of the key stroke would be shown as binary on the FPGA boards' LEDs, as well as on the UART terminal. For this purpose, the application developed in Appendix K is slightly modified to obtain a different visual output from the LEDs; on the FPGA evaluation boards.

1. Follow the guide in Appendix K all the way to and including step 24. However, in this case use the following source code:

```
1. #include <stdio.h>
2. #include "platform.h"
3. #include "xil_io.h"
4. #include "xparameters.h"
5. #include "xuartps_hw.h"
6.
7. int main()
8. {
9.     init_platform();
10.
11.     int keyPress;
12.
13.     // clear screen and display demo name on uart
14.     xil_printf("%c[2J",27);
15.     xil_printf("\n\rUART INPUT ASCII ON LEDS");
16.     xil_printf("\n\rPress any key on the keyboard to
    see its ASCII binary value on the LEDs\n\r");
17.
18.     while(1)
19.     {
20.         // read the byte from UART
21.         keyPress = XUartPs_RecvByte(0xE0001000);
22.
23.         // display keyboard key value in hex/dec on uart
24.         xil_printf("ASCII value --> Dec: %03d\t\tHex:
    0x%02x\n\r",keyPress,keyPress);
25.
26.         // send ASCII value to LEDs
27.         Xil_Out32(0x41200000 ,keyPress);
28.     }
29.
30.     cleanup_platform();
```

```
31.  
32.     return 0;  
33. }
```

2. Then follow the guide in Appendix K from step 25 onwards
3. Open a serial terminal as shown in Appendix J

This completes the design and software application of the LED – UART IO.

Appendix N. Vivado Zynq-7000 FPGA Design Guide D: Peripherals Tests

While three FPGA designs and applications were sufficient for the development, testing and potential demonstration of the project-prototype, a few more designs were added to learn a bit more about basic FPGA applications. In this section, the main FPGA hardware design described in Appendix K would be reused, but with a new software application: peripherals testing. That is, the FPGA would run a self-test to check if all connected peripherals are working. Such as the LEDs, the GPIOs, and interrupts. The results of the test would be displayed on the UART.

1. Follow the guide in Appendix K all the way to and including step 22
2. Choose Peripherals Tests from the Available Templates then click Finish

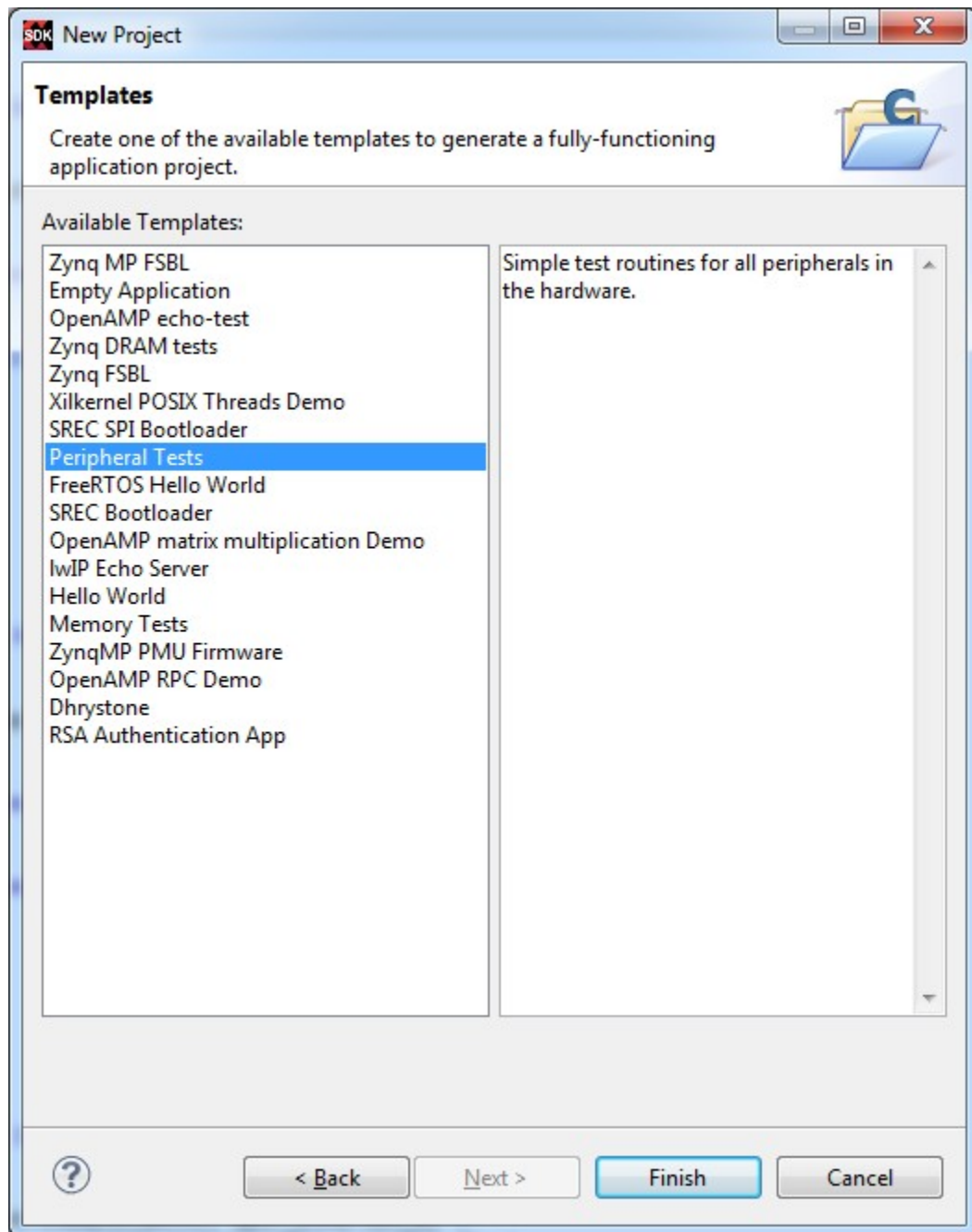
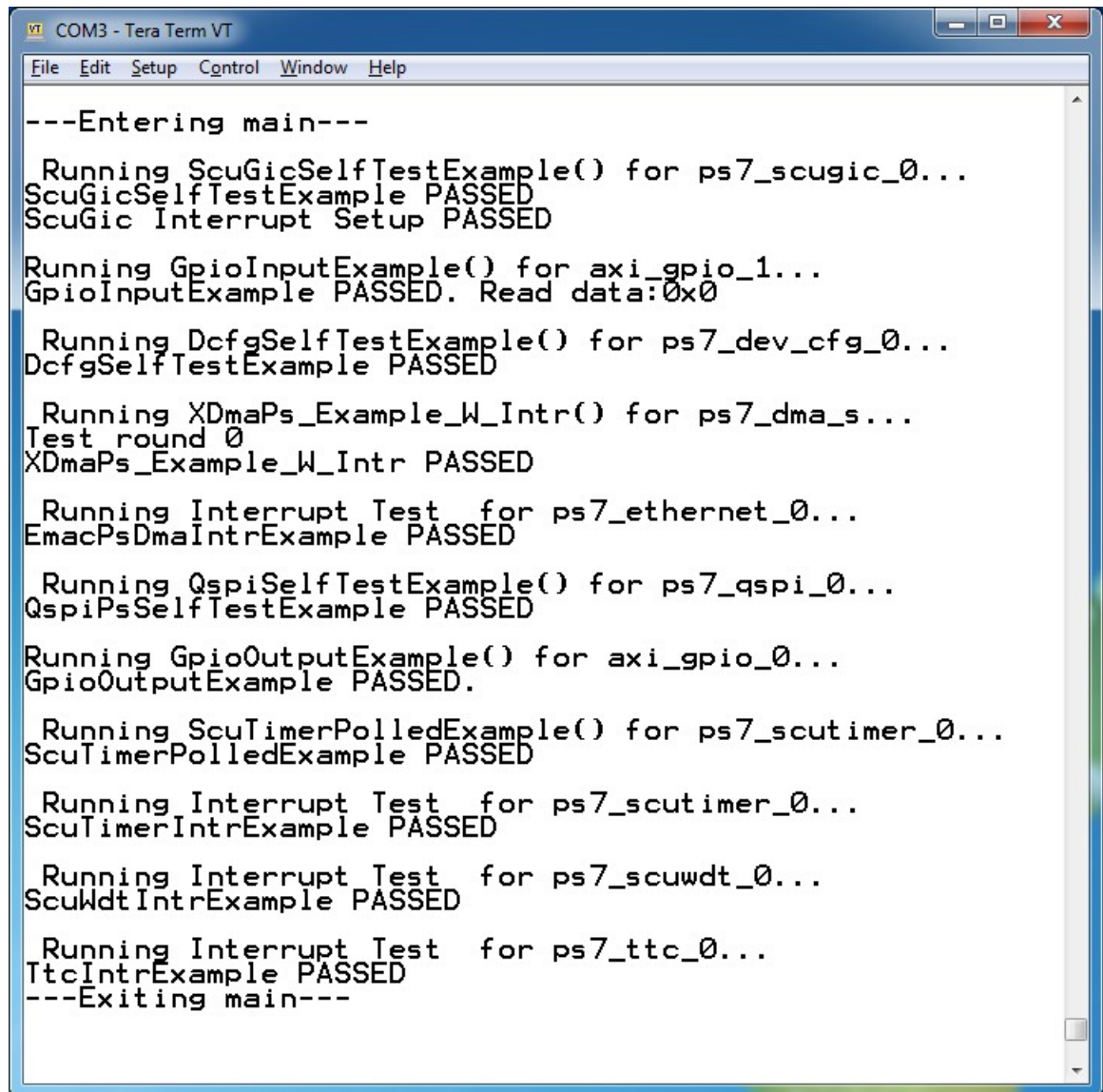


Figure N1 Select the Peripherals Tests Template in SDK

3. In the SDK click on the program FPGA icon then click program; to program the Zynq PL with the bitstream. Note that if the user has not power-cycled or reset the FPGA, there is no need to reprogram the FPGA as we would be using the same hardware. Though, it doesn't hurt to reprogram
4. Open a serial terminal as shown in Appendix J

5. In the SDK, on the left side panel, right-click on the project folder Periph_Test in the Project Explorer panel then go to Run As → 4 Launch on Hardware (GDB). The application should run, and the peripheral tests and results should be displayed on the UART terminal



```
COM3 - Tera Term VT
File Edit Setup Control Window Help

---Entering main---

Running ScuGicSelfTestExample() for ps7_scugic_0...
ScuGicSelfTestExample PASSED
ScuGic Interrupt Setup PASSED

Running GpioInputExample() for axi_gpio_1...
GpioInputExample PASSED. Read data:0x0

Running DcfgSelfTestExample() for ps7_dev_cfg_0...
DcfgSelfTestExample PASSED

Running XDmaPs_Example_W_Intr() for ps7_dma_s...
Test round 0
XDmaPs_Example_W_Intr PASSED

Running Interrupt Test for ps7_ethernet_0...
EmacPsDmaIntrExample PASSED

Running QspiSelfTestExample() for ps7_qspi_0...
QspiPsSelfTestExample PASSED

Running GpioOutputExample() for axi_gpio_0...
GpioOutputExample PASSED.

Running ScuTimerPolledExample() for ps7_scutimer_0...
ScuTimerPolledExample PASSED

Running Interrupt Test for ps7_scutimer_0...
ScuTimerIntrExample PASSED

Running Interrupt Test for ps7_scuwdt_0...
ScuWdtIntrExample PASSED

Running Interrupt Test for ps7_ttc_0...
TtcIntrExample PASSED
---Exiting main---
```

Figure N3 UART Output of the Zynq-7000 FPGA Peripherals Tests

This completes the design and software application of the peripherals tests.

Appendix O. Vivado Zynq-7000 FPGA Design Guide E: Memory Tests

Continuing in building basic FPGA applications to be run on the ZedBoard, the next application is to run memory tests which tests Memory Regions present in the hardware.

1. Follow the guide in Appendix K all the way to and including step 22
2. Choose Memory Tests from the Available Templates then click Finish

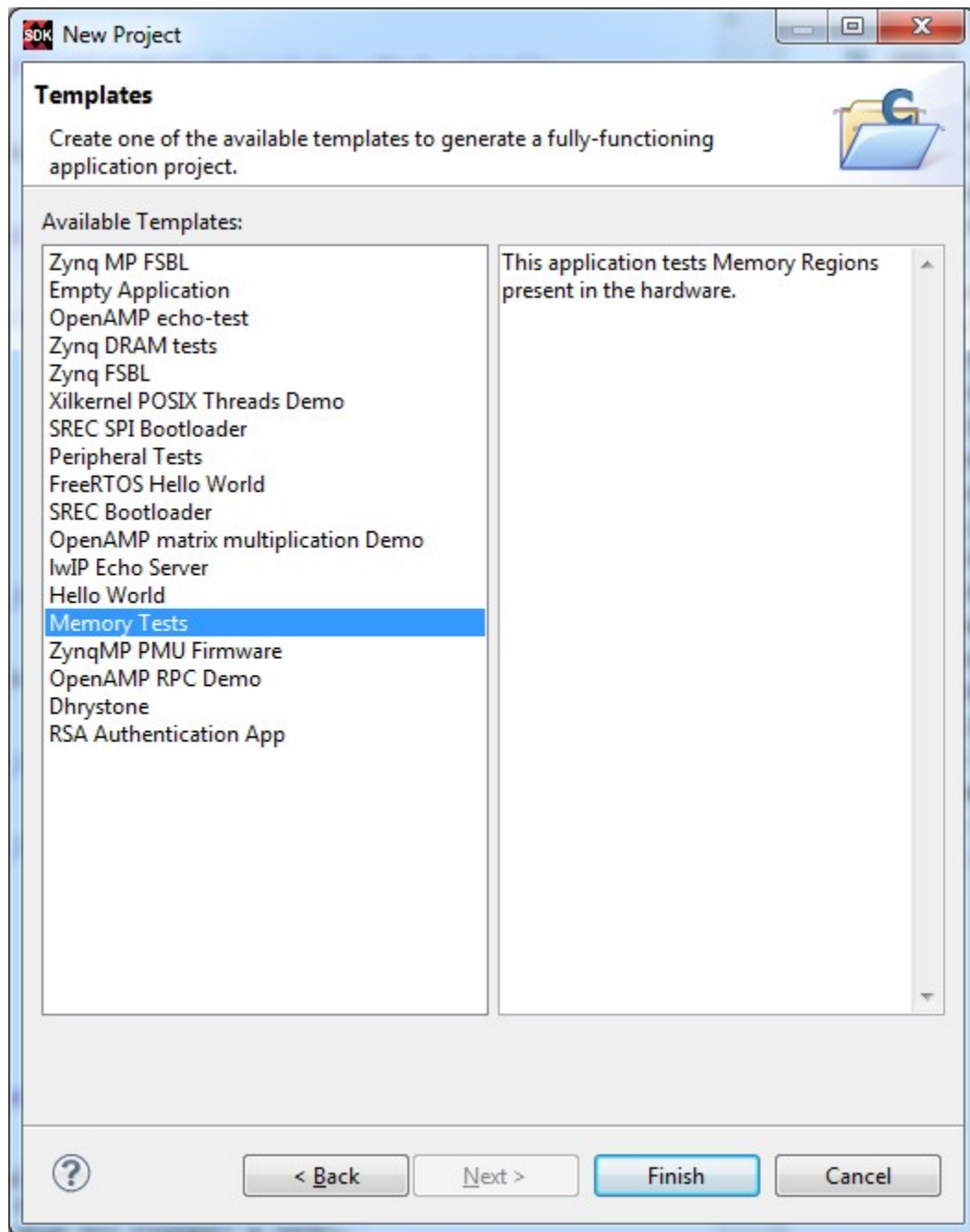
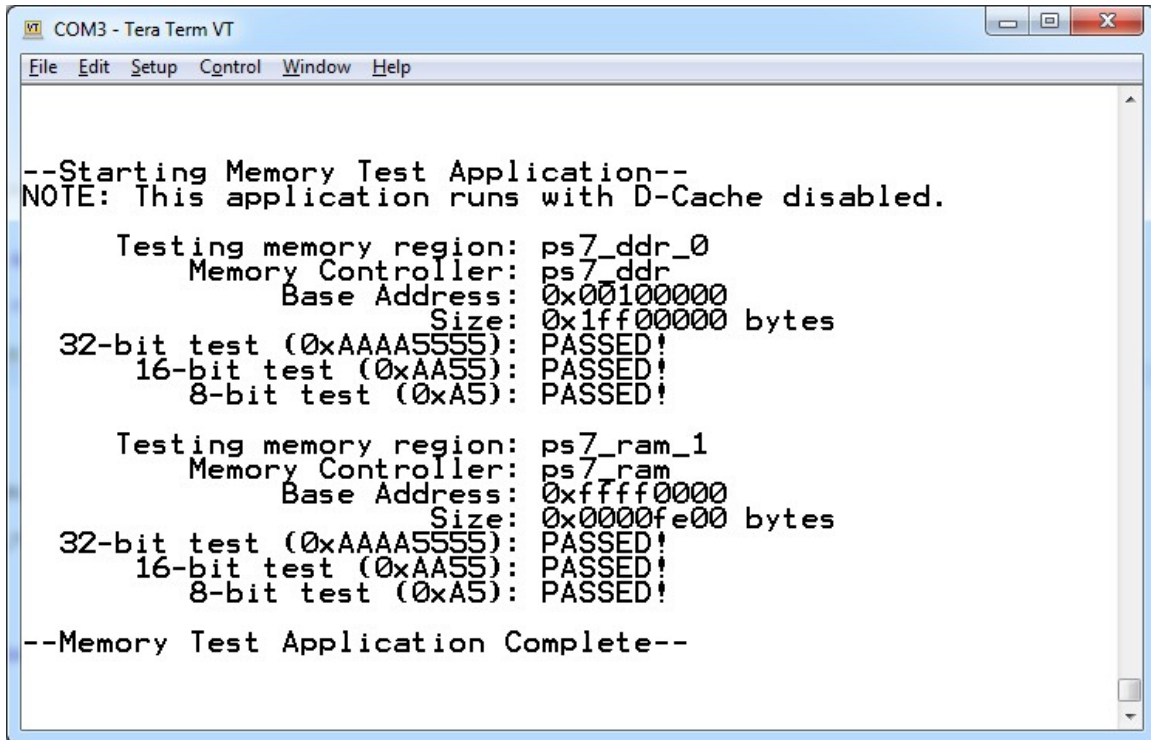


Figure O1 Select the Memory Tests Template in SDK

3. Open a UART terminal as shown in Appendix J
4. In the SDK, on the left side panel, right-click on the project folder Mem_Test in the Project Explorer panel then go to Run As → 4 Launch on Hardware (GBD). The application should run, and the peripheral tests and results should be displayed on the UART terminal



```
COM3 - Tera Term VT
File Edit Setup Control Window Help

--Starting Memory Test Application--
NOTE: This application runs with D-Cache disabled.

  Testing memory region: ps7_dds_0
    Memory Controller: ps7_dds
      Base Address: 0x00100000
      Size: 0x1ff00000 bytes
    32-bit test (0xAAAA5555): PASSED!
    16-bit test (0xAA55): PASSED!
    8-bit test (0xA5): PASSED!

  Testing memory region: ps7_ram_1
    Memory Controller: ps7_ram
      Base Address: 0xffff0000
      Size: 0x0000fe00 bytes
    32-bit test (0xAAAA5555): PASSED!
    16-bit test (0xAA55): PASSED!
    8-bit test (0xA5): PASSED!

--Memory Test Application Complete--
```

Figure O2 UART Output of the Zynq-7000 FPGA Memory Tests

This completes the design and software application of the memory test.

Appendix P. Vivado Zynq-7000 FPGA Design Guide F: Zynq DRAM Tests

The fifth FPGA sample application runs out of OCM and performs memory tests and read/write eye measurements on Zynq DRAM. The test is interactive and would perform:

- Memory test
 - Read eye measurement
 - Write eye measurement
1. Follow the guide in Appendix K all the way to and including step 22
 2. Choose Zynq DRAM Tests from the Available Templates then click Finish

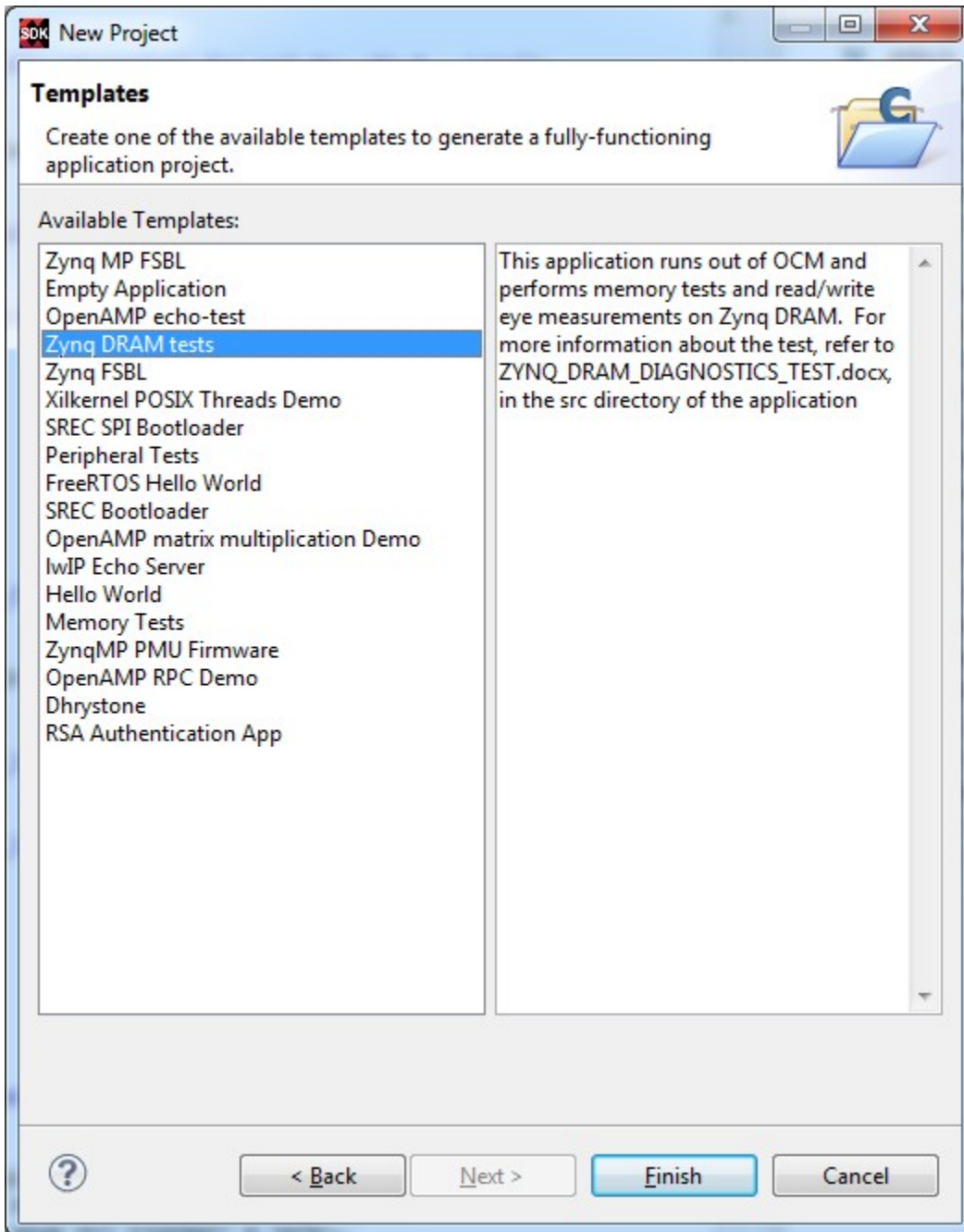


Figure P1 Select the Zynq DRAM Tests Template in SDK

3. Open a UART terminal as shown in Appendix J
4. In the SDK, on the left side panel, right-click on the project folder ZynqDRAM_Test in the Project Explorer panel then go to Run As → 4 Launch on Hardware (GDB). The application should run, and the peripheral tests and results should be displayed on the UART terminal


```

-----
----- ZYNQ DRAM DIAGNOSTICS TEST -----
-----
Select one of the options below:
## Memory Test ##
Bus Width = 32, XADC Temperature = 52.6004
's' - Test 1MB length from address 0x100000
'1' - Test 32MB length from s 0x100000
'2' - Test 64MB length from address 0x100000
'3' - Test 128MB length from address 0x100000
'4' - Test 255MB length from address 0x100000
'5' - Test 511MB length from address 0x100000
'6' - Test 1023MB length from address 0x100000
## Read Data Eye Measurement Test
'r' - Measure Read Data Eye
## Write Data Eye Measurement Test
'i' - Measure Write Data Eye
Other options for Write Eye Data Test:
'f' - Fast Mode: Toggles Fast mode - ON/OFF
'c' - Centre Mode: Toggles Centre mode - ON/OFF
'e' - Vary the size of memory test for Read/Write Eye Measurement tests
## Data Cache Enable / Disable Option:
'z' - D-Cache Enable / Disable
## Other options
'v' - Verbose Mode ON/OFF

Option Selected :

```

Figure P2 Interactive UART Menu of the Zynq DRAM Tests

The memory test has the following options, which could be used by using the keyboard as input.

Table P1 Zynq DRAM Memory Test Options

Option	Test Start Address	Test Length
's' ("short")	0x100000	1MB
'1'	0x100000	32MB
'2'	0x100000	64MB
'3'	0x100000	128MB
'4'	0x100000	255MB
'5'	0x100000	511MB
'6'	0x100000	1023MB

Each memory test consists of 15 sub-tests using different data patterns. In each sub-test, the entire range is first written sequentially, and then read and compared against the expected value. The 15 patterns are:

Table P2 Zynq DRAM Sub Memory Tests

Sub-test	Description
0	Incrementing pattern, unique value per memory location (data = address)
1	All 0
2	All 0xffffffff
3	All 0xAAAAAAAA
4	All 0x55555555
5	Alternating 0x00000000 and 0xFFFFFFFF
6	Alternating 0xFFFFFFFF and 0x00000000
7	Alternating 0x55555555 and 0xAAAAAAAA
8	Alternating 0xAAAAAAAA and 0x55555555
9	Aggressor pattern identical on all 8 bits
10	Aggressor pattern with one bit inverted, x8 times (1 per bit)
11-14	Pseudo random patterns with different seeds

Additional tests available are the read data eye test and the write data eye test. It is left to the reader as an exercise to experiment with them. Other menu options are described in Table H2.

Table P3 Zynq DRAM Memory Test Options

Options	Name	Description
'f'	Fast	Toggle 'fast' mode on/off. In fast mode, the memory test used during eye measurements uses less sub-tests and therefore runs about twice as fast, at the cost of being slightly more optimistic. By default 'fast' is on.
'c'	Center	Toggle 'center' mode on/off. When enabled, the write eye measurement result is immediately programmed into the DDR controller.
'e'	Eye test size	Vary the size of the memory test used at each step of a read/write eye measurement functions. The default value is 1MB, resulting in fastest speed at the cost of producing slightly optimistic results. Hitting this key repeatedly varies the test size circularly between the values 1, 2, 4, 8, 16, 32MB. Note that using the value 4 will quadruple the test run time.
'v'	Verbose	Toggle verbose mode on/off. If on and errors occur during a memory test, the first 10 errors in each sub-test are printed.
'z'	D-Cache Enable/Disable	Toggle D-cache – enable/disable.

This completes the design and software application of Zynq DRAM tests.