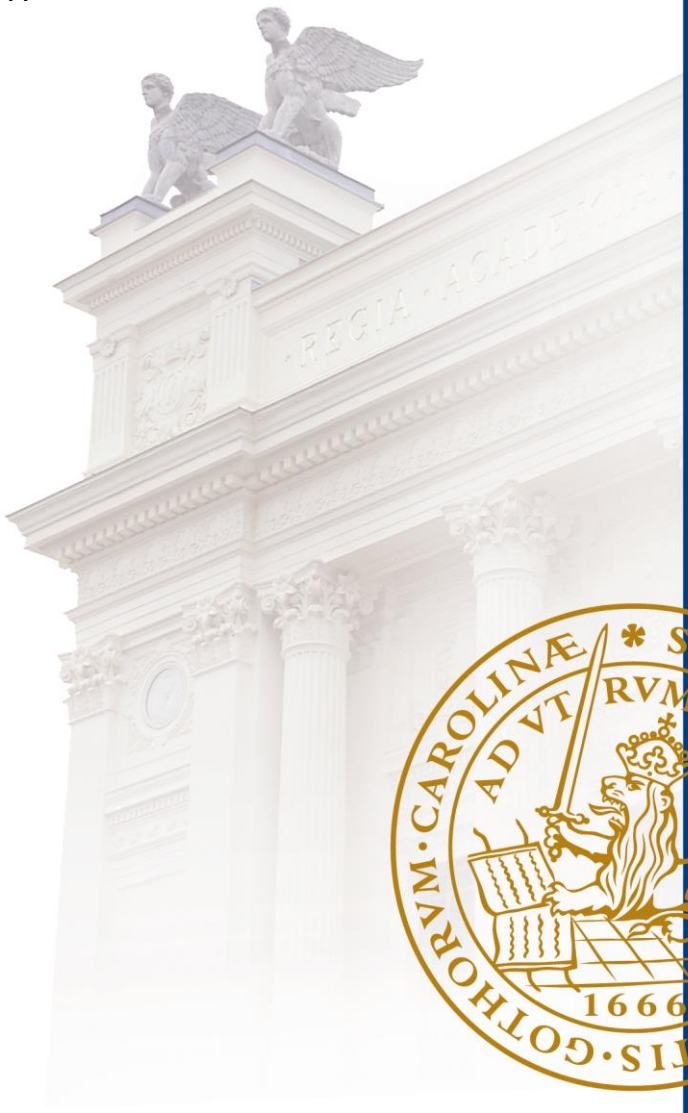
Master's Thesis

Design and Implementation of an Embedded Vision System for Industrial Robots

Kazi Asifuzzaman

Department of Computer Science
Faculty of Engineering LTH
Lund University, 2013



ISSN 1650-2884
LU-CS-EX: 2013-27

LUND
UNIVERSITY

Master Thesis

Design and Implementation of an Embedded Vision System for Industrial Robots

Author:
Kazi Asifuzzaman

Supervisor:
Flavius Gruian

The Thesis Work Has Been Conducted at
The Department of Computer Science of
Lund University, Sweden

June 2013

Abstract

Enhanced vision system attached to any industrial robot undoubtedly increases its accuracy and precision. Today, vision in industrial robots is facilitated almost exclusively via external or fixed cameras which may cause rather inconvenience due to obstacles in the line of sight. In this master thesis project, an Embedded Vision System is designed and developed to stream live video of a robot‘s focus point while being attached to its arm/actuator. Within the scope of this work, a working prototype has been achieved which is capable of producing live video stream on 640 X 480 resolution with a frame rate slightly above 9 FPS, having 256 colors for each pixel, displayable on a regular LCD display monitor. The system has been realized in a Spartan 6 platform and an Aptina image sensor has been used to acquire pixel information. I2C interfacing has been used to program the image sensor, data transfers have been facilitated by DMA cores, an off-chip DDR2 memory has been used for frame buffer and HDMI has been used as video out. Feasibility of adding Ethernet transmission capability has also been investigated.

Keywords: Embedded System, Digital ASIC, Video Streaming, Robot Vision

Acknowledgements

I would like to take the time to express my gratitude towards my supervisor Flavius Gruian for his valuable support and direction all along the project. Without his sincere help and patience, this project might not have been completed.

I would like to further thank Klas Nilsson for providing me this opportunity to work on this project, Sven Gestegard Robertz for sharing comprehensive knowledge on Computer Networks, Lars Nilsson and Anders Bruce for their super-efficient support on system logistics.

Finally, I thank my parents for always keeping faith in me, my family members and friends for happily taking endless pain for me, and my wife for her extensive support and understanding during this important period of my life.

Abbreviations

FPGA = Field Programmable Gate Array

ASIC = Application Specific Integrated Circuit

HDMI = High-Definition Media Interface

VGA = Video Graphics Array

FPS = Frame per Second

DMA = Direct Memory Access

EDK = Embedded Design Kit

ISE = Integrate Software Environment

XST = Xilinx Synthesis Technology

XPS = Xilinx Platform Studio

OS = Operating System

LED = Light Emitting Diode

PLL = Phase-Locked Loop

RGB = Color scheme combining Red, Green and Blue

YCrCb = Color scheme with Intensity, Red and Blue

LCD = Liquid Crystal Display

FIFO = First In First Out

CMOS = Complementary Metal Oxide Semiconductor

TFT = Thin Film Transistor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis report presents the development and implementation of an embedded vision system targeted for industrial robots. We have developed a prototype as an embedded camera design on FPGA platform. The completed design is supposed to fit on a robots actuator/arm to stream close up video of its activity area in order to increase the precision and accuracy. Such devices might come up as an alternative solution of using multiple fixed cameras to assist industrial robots.

1.1 Embedded vision systems in FPGAs

There has been a tremendous development in embedded vision systems in the last decade in response to the huge consumer desire for mobile phone cameras. This has triggered electronic giants to invest more resources for developing miniature cameras attached with the mobile phones. In fact, for few years, quality and capability of mobile cameras remained as one of the most important factors for choosing over mobile phones. Being in the race, mobile phone cameras, nowadays, have very advanced features like night vision, face detection and many more. FPGAs are reliable and most convenient platform to develop and test phased development of such technologies. FPGAs provide a flexible and easy design platform for developing embedded vision system where desired requirements and functions can be analyzed with several components and architectures until it fits the purpose.

1.2 Vision in industrial robots

Today, vision for industrial robots is almost exclusively served by external or fixed cameras due to their size and processing requirements [1][2][3]. Thus, the precision and accuracy assistance for the robot from these cameras cannot go beyond a limit for reasons corresponding to obstacles on the line of sight, distance of the activity area from the camera(s) etc. An alternative approach would be to use multiple fixed cameras but which would subsequently increase cost and complexity. Therefore, it is worth investigating the possibility and feasibility of developing an embedded vision which can reside on the tip of the robotic actuator/arm as well as capable enough to serve the purpose.

Key issues regarding a robot vision usually includes its physical size, physical position, processing capabilities, communication options, feasibility of performing real time operations and flexibility of phased development. The physical dimension of the design is very important as it primarily decides where the camera can be conveniently mounted. If the system is too big to fit on robots arm or actuator then it needs to be placed in a static position. Different forms of physical positions of a robot vision effectively influences the internal functions and behaviors. For example, if a robot vision is mounted in a stationary platform, it may not have to calculate the relative distance between the actuator and the object. On the other hand, a vision system mounted on a moving robot actuator has to process the relative distance calculation between the actuator and the object dynamically for every move it makes. Communication capability is a vital factor as most of the robot vision systems are supposed to be involved in decision making process using the acquired vision. To do that, in most of the cases, the vision data is transmitted to a powerful processing unit usually a PC to process the consequent decision. This transmission can take place in different forms considering the nature of the operation and placement of the camera. Ethernet transmission system is often the obvious choice while wireless transmission systems can also be given a thought where connecting a wire to the tip of the robots actuator is inconvenient for particular reasons. Real time operations requires a faster execution of image processing algorithms which is, in many cases are not achievable through software execution on general purpose microprocessors. Instead, executing time consuming algorithms in hardware is a clever way to deal with the problem. Certain flexibility in design process is always a great advantage to have in order to implement new requirements on an existing design without spending too much time and money on redesigning.

1.3 Motivation and scope of the thesis

In our project, implementation of the embedded hardware is been targeted to improve the vision capabilities in industrial robots which in many cases is a vital area of importance. With the increased visibility and closeness of the proposed system, we can introduce improved precision and refinement to robots performance. To pursue this goal, robust vision system needs to be developed, which would be able to participate in robots decision making process through implementing advanced image processing algorithms. So, we needed to achieve a working embedded vision system which can be conveniently placed on robots manipulator, flexible for reconfiguration and capable of transmitting digital image information to a processing unit. Within the scope of this master thesis project, we have developed a complete embedded vision system which complies with such requirements and explored the feasibilities of adding an Ethernet transmission system.

1.4 Contributions

The author was the only student to carry out the project with close supervision and guidance from the supervisor. The author started the project from scratch and contributed till achieving a working design. In the project, several off-the-shelf Xilinx IPs were used as well the component to generate VGA signals from pixel data information and the component to generate TMDS signals were modified from a previous project conducted at the department and a reference design, respectively. The main challenge was to organize and use these components in the right way to achieve a properly functioning design. Along with integrating the existing components, the author developed the pixel data processing unit, interfaced the image sensor, analyzed and determined the correct configurations for off-the-shelf IPs, investigated and reported the bottleneck of the design.

1.5 Organization and outline of this report

This report has been organized in nine separate chapters which include information on development of the prototype, work flow, design environments, components that have been used, different approaches that have been considered, knowledge and experience gained from this project. Chapter 1 corresponds to a brief introduction to the current situation of the problem area, recent developments and scope of the project. Chapter 2 illustrates overview of the final design along with features that have been achieved. Chapter 3 consists of a brief description of design tools that have been used in the design

process. Chapter 4 focuses on the complete work flow of a single project starting from preparation, key design phases and debugging techniques. Chapter 5, 6 and 7 consist details on development and installment of different components used in the project. Chapter 8 briefly presents different project approaches which have been tried on the way to the final design and specific obstacles encountered. The report ends with Chapter 9 discussing concluding statement and notes on future development.

Chapter 2

Overview of the Prototype

The prototype of this design has been implemented using an Aptina MT9D112 image sensor, and a Spartan 6 XC6SLX45 FPGA. HDMI video output has been implemented instead of the traditional VGA output for video streaming.

FIGURE 2.1: FPGA, Image sensor and display unit of the prototype

2.1 Key features of the design

Any embedded system has a particular set of requirements to meet. Engineers tend to develop embedded systems which have a particular purpose that may not be served with general purpose / existing designs.

In this system, an Aptina MT9D112 image sensor has been used for generating pixel data. This sensors core pixel array is dimensioned by 1722 X 1262 pixels which are capable to produce images of 1600 X 1200 pixels in full resolution [7]. In response to the particular requirement of getting close-up views of the activity area, a pixel array of 640 X 480 dimensions has been used roughly from the center of the main pixel array of the sensor.

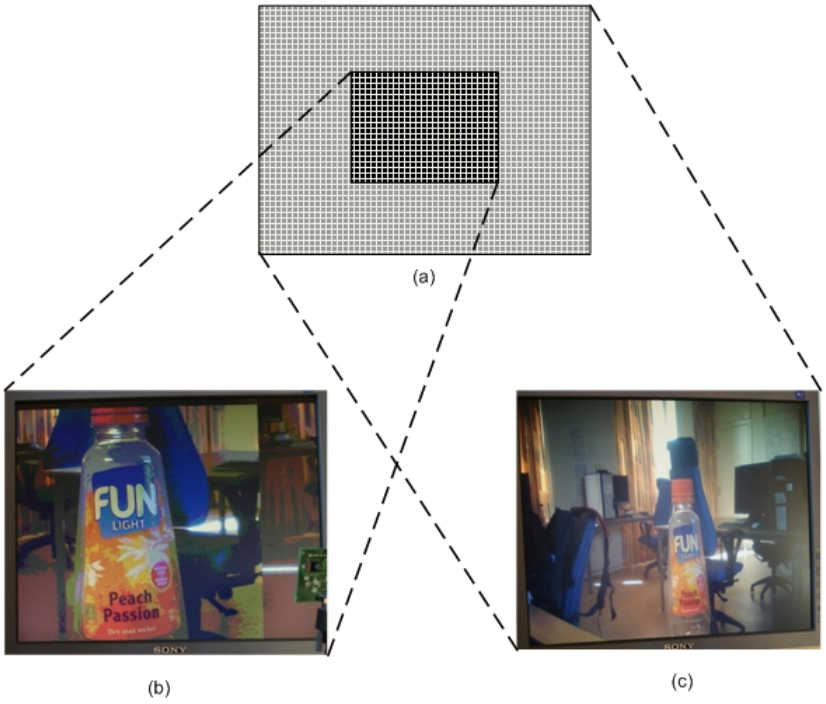


FIGURE 2.2: (a) Sensors pixel array used in full resolution - 1600X1200 (grey) and sensors pixel array used in the design 640X480 (black). (b) Output of the design focused and up-close to the object. (c) Output of the reference design object not isolated; processing unnecessary details of the surrounding.

The color of a pixel is defined in RGB 3:3:2 format, capable of displaying 256 colors of each pixel. Several approaches have been tested for implementing RGB 5:6:5 format, targeting 65K colors for each pixel. But that subsequently increased system overhead as well as the system clock speed needed to be doubled to handle the increased pixel data processing. So, RGB 3:3:2 color format deemed to be the most suitable to serve the purpose of the system which also remained consistent with the requirements.

Frame rate of the system has been measured slightly above 9 FPS while acquiring the picture from the sensor. The frame rate limitation is due the factor that, the soft core DMA units that have been used in the design are not apparently specialized for handling high data throughput of video applications. Whenever we have set the design to deal with a higher data throughput by increasing frame rate or implementing a better color scheme, the functions handling DMA units exited indeterminately. We have found this

only after completing the design and therefore reverting to other alternatives was not feasible.

To meet the communication need of the vision system, we planned to implement the Ethernet transmission capability on the vision system. Despite of our extensive efforts to achieve a working Ethernet transmission system, we could not acquire a functioning design within the time period of the project. However, procedure and techniques tested for implementing the Ethernet transmission system has helped to achieve a solid knowledge base on the matter which would greatly help to proceed into future developments.

We have developed this vision system on a FPGA platform. Using FPGAs for such systems gives certain advantages over the ASIC design approach and general purpose processor based designs. Considering future enhancements of this design, its flexibility and adaptability to design changes are very high as redesigning logic circuits on an FPGA is considerably easy and convenient. In the advanced implementation phase, if the system is required to perform real time operations, some image processing algorithms may not be executed in software running on general purpose processor systems as the processing is excessively time consuming. FPGAs give the flexibility to execute such image processing algorithms in hardware which is way faster than software execution and therefore capable of complying with real time operations.

2.2 Related works

Abdul Waheed Malik et al presented a realtime machine vision system using FPGA and Soft-core processor. They concluded that, using soft core processors improved flexibility and achieved rapid development time. Components realized in hardware is capable to accelerate performance to handle critical applications. They could also achieve higher frame rate, low latency and lower power consumption [4].

Henry Andrian et al presented an Embedded CMOS imaging system for Real-Time Robotic Vision which provided information for mobile manipulation control. Their design was implemented using a CMOS image board and a TI DSP board and could achieve a 30 FPS frame rate in 640 by 480 pixels resolution [5].

Shinichi Hirai et al presented a realtime FPGA based vision system which implements vision algorithms using C/C++ based logic circuit design. They catagorized vision system approaches into three main areas namely, general purpose MPU in software based approach, ASIC based approach and FPGA based vision system. They concluded that FPGA based vision systems are the optimum choice as they are faster than MPU

in softwrae based approach as well as offers more flexbility and cost efficiency than ASIC based approach [6].

2.3 Device utilization and power analysis

Our final design uses 17% of the available slice registers and 38% of the available slice LUTs of the Spartan 6 LX45 FPGA.

Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	9,552	54,576	17%
Number of Slice LUTs	10,387	27,288	38%
Number of RAMB16BWERs	92	116	79%

TABLE 2.1: Device utilization

The design has a system clock of 25 MHz and the integrated PLL on the sensor produces a pixel clock of 50 MHz. In this configuration, power consumtion of the system are the following:

Dynamic	**Quiescent**	**Total**
0.0457	1.006	1.464

TABLE 2.2: Supply power (in watt)

2.4 Main components

An embedded system is constituted of many components, usually consisting a microprocessor (Microblaze, PowerPC), ready to use IPs (PLB Bus, IIC Interface, GPIO) and some custom logic cores.

•**Microblaze** is a soft core processor from Xilinx which is widely used in designs developed in Xilinx platforms.

•**Processor Local Bus (PLB) 4.6** is the standard bus system from Xilinx which facilitates the connection to several components on a design and their communication and interaction.

• **Local Memory Bus (LMB)** provides a fast connection between the processor and other peripherals, specially the BRAM.

•**Block RAM (BRAM)** is an on chip memory block typically smaller in size in comparison to external memories.

Name of Components	Number of Instances
Microblaze	1
Processor Local Bus(PLB) 4.6	3
Local Memory Bus (LMB) 1.0	2
PLBV46 to PLBV46 Bridge	2
Block RAM (BRAM) Block	2
LMB BRAM Controller	2
Multiport Memory Controller (DDR/DDR2/SDRAM)	1
XPS BRAM COntroller	1
XPS UART (Lite)	1
XPS 10/100 Ethernet MAC Lite	1
XPS General Purpose IO	2
Clock Generator	1
Microblaze Debug Module	1
Processor System Reset Module	1
XPS Interrupt Controller	1
XPS Central DMA Controller	2
XPS Timer/Counter	1
Phase Locked Loop	1
XPS IIC Interface	1
Chipscope Integrated Controller	1
Chipscope PLBv46 Integrated Bus Anlyzer (IBA)	1
Chipscope Integrated Logic Anlyzer (ILA)	1
DATABIND_PLB	1
PLB_VGA	1

TABLE 2.3: Name and number of instances of components in the design

- **Multiport Memory Controller (MPMC)** provides the interfacing for several types of memories with multiport operation support.
- **XPS UART** is XPS Universal Asynchronous Receiver Transmitter used for serial data transfer.
- **XPS Ethernet MAC Lite** provides the Ethernet interface for embedded designs.
- **XPS GPIO** core can be used to check out data at a particular point of the design.
- **Central DMA Controller** assists high speed data transfer without processor interruption.
- **Clock Generator and Phase Locked Loop** are used to generate different clocks with desired frequencies and phases.
- **Chipscope** modules are used to debug embedded hardware.
- **XPS IIC** provides two wire interface to IIC supported modules.

Among the two custom IP cores of the design,

- **Pixel Data Processing Unit (DATABIND PLB)** is responsible for sampling RAW pixel data from the sensor, prepare a word consisting 4 pixel data and push it to READ FIFO attached to it, for transferring them to the DDR2 memory through DMA transfers.

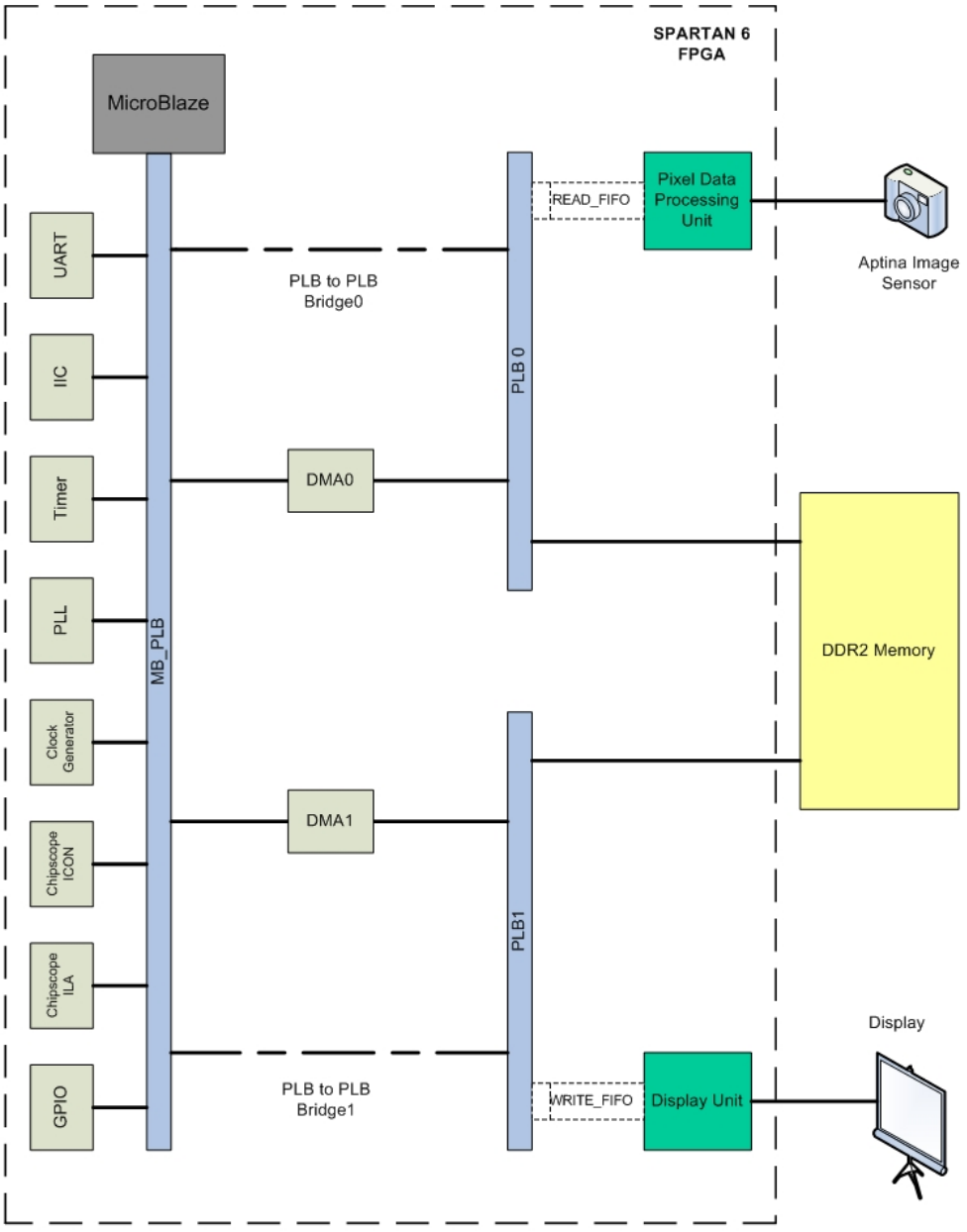


FIGURE 2.3: The system diagram

- **Display Unit (PLB VGA)** receives pixel data from the DDR2 memory through

DMA transfers to its WRITE FIFO and generates video signals to display corresponding frames on the monitor.

Chapter 3

Design Environment

We have used Xilinx EDK and ISE tools to implement the prototype of the design. Digilent adept has been used to program the board. The FPGA platform was an ATLYS development board from Digilent, equipped with a Spartan 6 XC6SLX45 FPGA and Xilkernel was used as the embedded OS.

3.1 Xilinx Platform Studio (XPS)

Xilinx ISE Design Suite 12.3 has been used to complete the design. This tool suite is provided by Xilinx to be used for development purposes with Xilinx FPGAs and platforms. Xilinx platform studio (XPS) is used to design the hardware platform and develop the software applications. The tool offers a wizard named Base System Builder (BSB) to build a basic embedded system with desired components selecting the processor type, system frequency, bus type, IO modules and memory options. This basic design can then be modified by adding other required peripherals, custom IPs and software applications. The design can be compiled and synthesized within the same tool and its functionalities can be tested on FPGAs.

In the Project tab of XPS (Figure 3.1), Microprocessor Hardware Specification (MHS) and Microprocessor Software Specification (MSS) file contains hardware and software system details which are automatically generated while generating a new design with BSB wizard as well as duly updated when any hardware changes are made to the design. User Constraint File (UCF) contains physical pin layout information associating system signals to designated physical pins on the board. In the Application tab, available software application projects are listed and ready to use IPs are listed in the IP Catalog tab. In the System Assembly View, Bus Interfaces tab provides an easy to understand

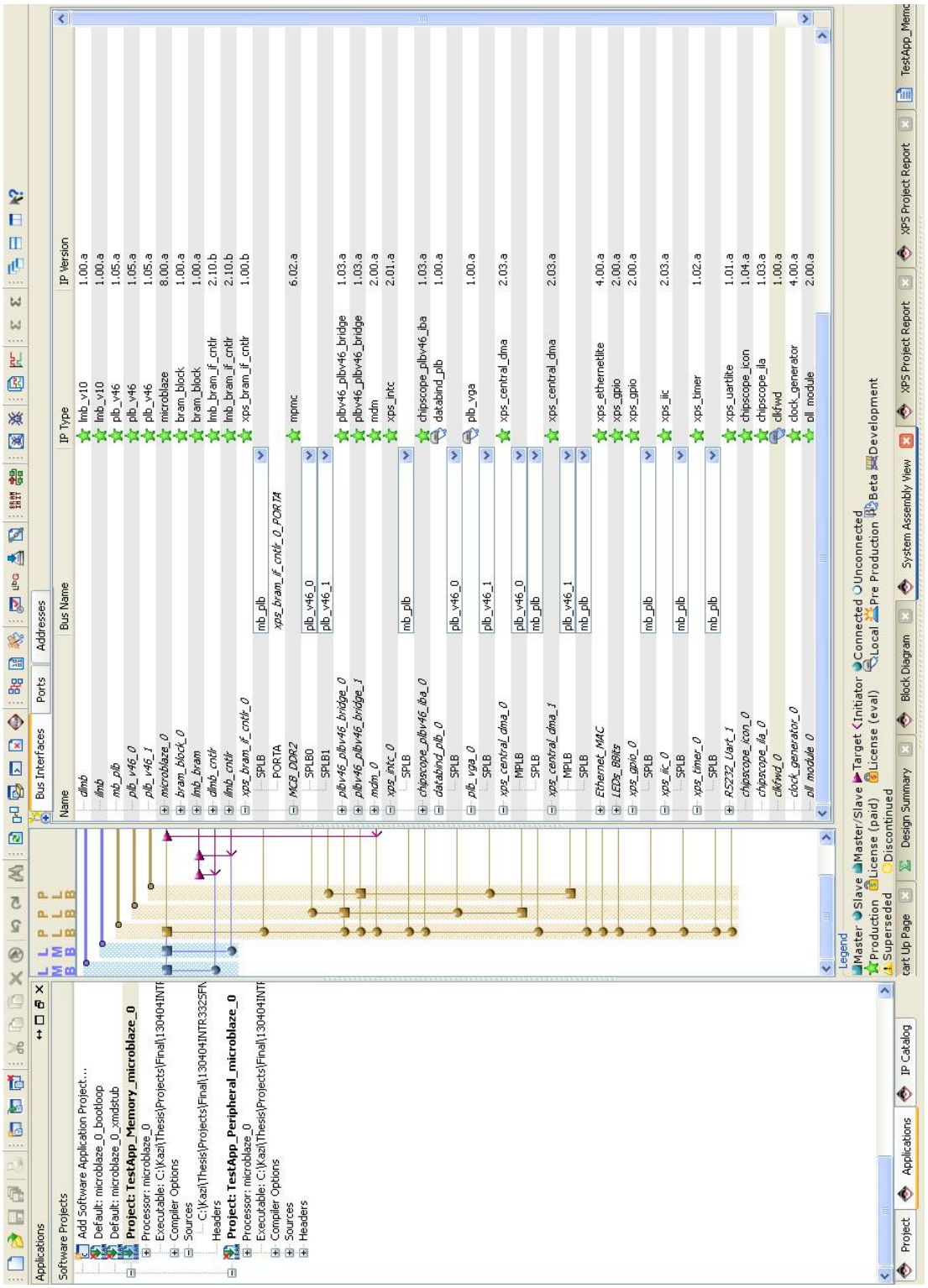


FIGURE 3.1: Xilinx Platform Studio (XPS) IDE

interface of the bus system where components can be connected and disconnected to particular buses conveniently using the GUI. The Ports tab displays interconnection of signals within the design and the Addresses tab contains the memory mapping information of the design.

3.2 Xilinx Project Navigator

In an embedded system design project, if there is a need to develop a custom hardware IP, the best way to proceed is to develop the hardware in a standalone project in ISE and test its functionalities with simulations before adding it to the EDK project.

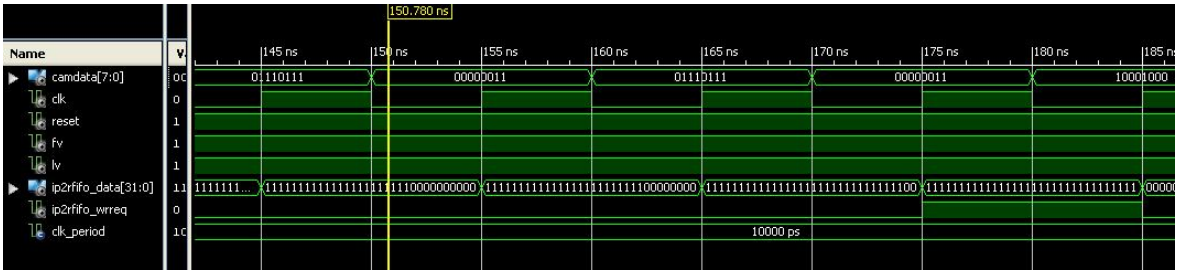


FIGURE 3.2: ISim Simulation of DATABIND custom IP Core used in the project

Because, hardware synthesis in EDK tools are considerably time consuming as well as minor debugging can be severely complicated. Custom IP cores in our design have been developed in ISE Project Navigator and have been vigorously tested by test bench simulations with ISim before adding to the main project. Figure 3.2 shows a snapshot of a hardware simulation of one of the custom IPs of the project.

3.3 Digilent Adept

Digilent Adept is an interfacing application from Digilent which is needed to program the FPGA board with bit files generated from the design project. After installing, Digilent plug-in files needed to be placed in ISE Design Suit Installation folder for proper functioning. This plug-in contains two files of .dll and .xml format.

3.4 FPGA Platform

The ATLYS development board has a Spartan 6 XC6SLX45 FPGA, a mid-level FPGA from Xilinx. This particular FPGA has 6822 slices, 2.1 Mbits of fast block RAM, 4

clock generators and 6 PLLs, realized in a 45 nm process. Furthermore, the ATLYS development board offers 128 MByte of external DDR2 memory, two HDMI video in, two HDMI video out, Trimode (10/100/1000) Ethernet interface, USB UART, AC-97 Audio Codec, USB HID Host and a high speed VHDC connector [9]

3.5 Embedded OS : Xilkernel

In our project, Xilkernel v5.00.a has been used as the embedded OS. Xilkernel provides higher flexibility in threading and scheduling for complex embedded systems. For our project, we had to go through and study basic principles of Xilkernel, the way it works and how it is used in an embedded system design. To use Xilkernel OS in a XPS design, Xilkernel should be selected in OS option in Software Platform Settings tab. In OS and Lib Configuration tab, name and priority of pthreads are also needed to be defined. Furthermore, in Compiler Option wizard, xilkernel should be stated in the Libraries to link against section of paths and options tab. Even though we have used only one thread in our application, we configured the support for multiple threads for future flexibility of adding other threads so that different scheduling options can be conveniently applied. UART has been configured as standard input and output.

Chapter 4

Work Flow

In this chapter we would discuss the particular work flow that we have used for implementing the prototype starting from preparation, major steps in the development phases and debugging.

4.1 Knowledge base and preparation

Embedded system design is a specialized area in Electronic Design which requires an engineer to have knowledge and proficiency both in hardware design and software development tools and techniques as well as hands on knowledge on using embedded system development platforms. It is often not possible to learn the techniques and issues embedded system design by only going through the theoretical materials because of the practical nature of the design procedure. From a design specification it may seem very simple to develop an Embedded System fulfilling the requirements, but the real hurdle is faced during the actual implementation. In addition, due to rapidly developing technologies, platforms, tools and techniques are being updated frequently. So, it is very important to review the current release of the tools, understand the design flow for the intended EDK version, targeted FPGA platform and compatibility issues (if any).

We started the preparation by looking into EDK Concepts, Tools and Techniques, a detailed guide from Xilinx on Embedded System design. We followed the step by step exercises listed in the guide to achieve an overall idea of design process of embedded systems starting with creating a new project with base system builder, understanding the project files, writing and testing software applications, developing and attaching a custom IP to the design. This guide turned out to be a very useful material to follow for achieving hands on experience on introductory embedded system design.

To get familiar with the particular board (ATLYS), we have also completed the six practice labs provided by Xilinx for the board which included basic hardware design, adding an IP from catalog, attaching a custom IP, writing software applications and debugging of embedded systems.

4.2 Development of custom hardware IPs

Creating custom IP and attaching it to an embedded project is often regarded as one of the least understood processes in XPS [19]. After generating a basic project with base system builder, we focused on developing custom IPs to serve specific requirements of the project which could not be met with existing, ready to use IPs. There are several aspects of creating a custom IP which could be very confusing for a new designer to deal with.

A custom IP will need the IPIF interface (being a master or slave) to be added along with the user logic to be able to interact with system buses (PLB, AXI). The designer also needs to determine if the intended custom IP needs services like Software reset, Read/Write FIFO, Interrupt control, User logic software register, User logic memory space and Data phase timer. Understanding the functionality and necessity of these services are not straight forward and often requires iterative execution of these steps with varying configurations to figure out the best settings for the IP. It is convenient to check the option of Generating ISE and XST project files to help implementing the peripheral using XST flow at the last step of Create Peripheral wizard. Development of custom peripheral in XST flow saves hours of work as the synthesis process is way faster than synthesizing hardware in XPS. And it also provides an easier platform to debug the Custom IP separately

In our project, we needed to use Read/Write FIFO for both of the custom IP cores because in this video streaming system, there would be data transfers between buses and components which might be running on different clocks as well as could be using different clock sources. In such systems, FIFOs are needed to buffer sequential data that are going to be transferred over a system running on multiple clock frequencies generated from different sources [20].

In the FIFO settings wizard, we had set values for FIFO size, dimension and calculation method reflecting the assessment of the data flow through these FIFOs, so that they never get full or empty. We could observe this using Chipscope Pro as well as using LEDs on the FPGA board. Several instances of the project were developed with FIFOs of various sizes and dimensions to determine the optimum setting that is consistent with

the system data generation, transfer and consumption rate. The instruction and data cache was not enabled for the Microblaze soft core processor. Previous steps create a XST project with the name of the custom IP at the XPS project directory. Then we ported the custom logic into the XST project without manipulating the IPIF signals. Once the porting is done, the design is then tested with hardware simulations to check its functionality. When it seems to work as expected then it is added to XPS project prompting the Create and Import peripheral wizard once again.

After importing the custom IP in the XPS design it will be available under the USER category in the IP catalog tab. After adding the custom IP to the project from the IP catalog, it needs to be connected to the bus, memory addresses need to be generated, input output signals need to be connected with desired signals of the system and externals signals (if any) need to be tied with physical pins using User Constraint File (.ucf).

This whole process is repeated, every time we add a custom IP with different settings for testing purposes. In such a way, after numerous attempts, we could finally decide the best settings and configuration of a custom IP which is finally added to the design to serve the purpose.

4.3 Xilinx IPs

Fortunately, we did not have to develop all peripherals of the design in the process of developing custom IP. There are ready to use IPs from Xilinx in IP catalog tab in XPS. However, it is sometimes tricky to understand their particular functionality and implementation techniques. Generally, the PDF datasheet provided with these IPs includes information on their features, functional descriptions, signal specifications and how they work. But it does not normally state how the core needs to be used in an embedded system design.

There are number of API drivers for each Xilinx IPs through which they can be operated and used within an embedded system design. These APIs provide an easy interface for the designer to use the complex IPs in a design. The APIs are responsible to handle to hardware level complexity.

If there are multiple instances of an IP in the same design then it is very important to instantiate right instance in software application with correct device id to ensure correct use of the IP. Details of all APIs for Xilinx IPs are listed in API documentation. Driver sources usually consist of examples and explanations of how to integrate and use a Xilinx IP in an embedded system design.

4.4 Testing and debugging

Testing and debugging are continuous processes in hardware development. Testing helps to ensure if the certain hardware units are working as expected. In embedded system design there are several ways to test and debug. Depending on the nature of the investigation, certain tools/techniques are used.

4.4.1 Chipscope Pro

Chipscope Pro is the debugging tool provided with Xilinx ISE design suite. It facilitates to capture bus transactions and individual design signals in real-time. In our project this tool has been used extensively for debugging and investigation purposes.

To add Chipscope peripherals in a design, a Chipscope integrated controller (ICON) needs to be added in the design from IP catalog. To investigate bus signals, Integrated Bus Analyzer (IBA) needs to be added via debug configuration wizard. Integrated Logic Analyzer (ILA) is needed to probe individual signals (i.e. signals is custom peripherals).

Even though custom peripherals are added into design after being verified with extensive simulations, they may not behave as expected in actual hardware. So, quite often it happened that a custom IP is set and added in a design and yet we could not get the desired results. Then, we needed to look into the signals in actual hardware run to figure out which part of the design/module has been compromised.

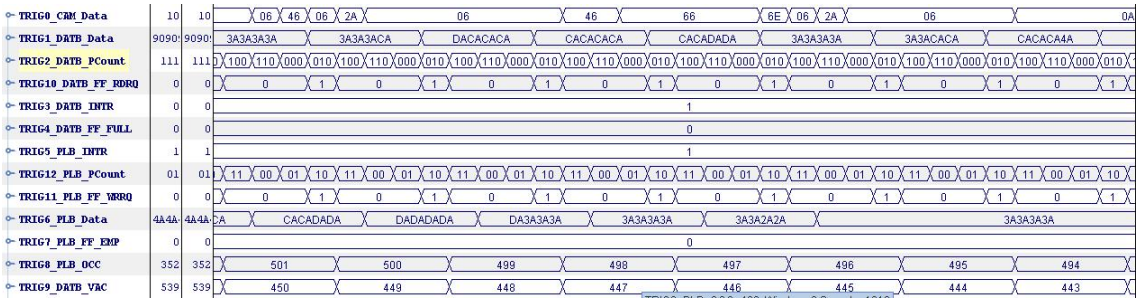


FIGURE 4.1: Investigating signals with Chipscope

In the configuration wizard of the ILA peripheral, signals of interest are associated with particular triggers. After synthesizing the design a Core Inserter (.cdc) file is generated in the project directory. Opening the Chipscope Pro Analyzer, we need to import .cdc file to list all signals of interest in the Chipscope waveform. Then, upon programming the board with .bit file, we select the trigger condition (certain signal containing certain value) to observe the signal activity in the triggered period. Triggers can also be combined with AND/OR operators to narrow down scenarios to observe.

Integrated Bus Analyzer (IBA) is also used in a similar procedure. Memory addresses of components are used to locate particular bus transactions.

4.4.2 Software printouts

Universal Asynchronous Receiver Transmitter (UART) can be used as an effective debugging tool by printing software printouts on HyperTerminal. Print functions in several points of the software application helps to determine if the program in being stuck somewhere. Also, if there is need to investigate a value calculated in the software it can also be printed out via HyperTerminal. In several phases in our project, we have used this technique to determine if the low level device drivers and APIs are configured, initialized and working correctly.

In our project, software printouts were used to read return values from the sensor read-/write operations via I2C interface. It was an important operation because to determine if we actually could communicate with the sensor, we needed to read certain register on the sensor chip and match the value with the reference value. Software printouts were also used to calculate and print frame counts and memory contents of the frame buffer.

4.4.3 GPIO

XPS General Purpose Input Output (GPIO) has been used in the project to stream data transactions from particular signals on the design to check if they are holding the correct value. Data stream debugging through GPIO provided a live streaming of the design and we could identify potential mismatches in data transmission which was fixed later on.

Although being a powerful debugging tool, Chipscope can only capture a certain number of samples at a time. That is, it cannot be investigated how frequently a signal of interest is changing its value in real time. For example, if it is needed to see when the READ FIFO if getting full, Chipscope can only provide one or few scenarios within the limited samples captured, at a time. So, to observe such signals which we suspected for being continuously high or low during the whole run time, we tied them to the LEDs on the FPGA board. It gave us a clear run-time impression of the signals.

Chapter 5

Sensor Control and Pixel Data Processing Unit

This chapter contains two main parts: Image sensor and Pixel data processing. In the image sensor part we talk about technical features, functional descriptions, control interface and ways of configuring the image parameters. In the second part we discuss how that output data stream from the sensor was processed and used in the system.

5.1 Image sensor

In our project we have used the VmodCAM module from Digilent which consists of two Aptina MT9D112 image sensors. We planned to develop a multi camera vision system as the final design and primarily we started the implementation using one sensor. Unfortunately, we were unable to start the integration process for multi camera vision system due to time constraints. The image sensor board is connected to the FPGA board via a VHDC connector and is programmed through an I2C interface.

5.1.1 Technical features and functional descriptions

Aptina MT9D112 image sensor consist a 2 megapixel color sensor array capable of generating a maximum resolution of 1600 X 1200 pixels. Several output formats can be selected as well as scaling, different special effects, noise reduction and image correction algorithms can be imposed on video output. An integrated PLL (Phase Locked Loop) is present to generate internal clocks for several resolutions and frame rates.

5.1.2 Programming the sensor

Programming the sensor was a tricky part of the project and has consumed considerable amount of time as the process included understanding of the protocol which needs to be followed to program the sensor as well as speculating the I2C interface. The sensor requires to be programed via a two wire serial interface. First, a functioning I2C interface needed to be implemented to communicate with the sensor and then a series of read and write register operations needed to configure the sensor with desired settings.

5.1.2.1 I2C interface

I2C is one of the standard interfacing techniques for connecting external peripherals to a system. It provides a two wire interface to the target module naming SDA (Serial Data) and SCL (Serial Clock). Data transmission on the I2C bus is initiated with a START condition and ends with a STOP condition. The sensor acts as a slave device on the bus.

To implement the I2C interfacing, protocols and hardware mechanisms of XPS IIC core was thoroughly studied as well as implementation mechanisms were primarily tested on a Temperature Sensor on a separate project.

5.1.2.2 Control interface

The control interface of the sensor is operated via reading and writing hardware registers of the sensor core. Each register write includes sending of a start condition, 8 bit device address, upper byte of 16 bit register address, lower byte of register address, upper byte of the 16 bit Data, lower byte of data and stop condition. Figure 5.1 represents sequential steps for performing one write operation via the I2C interface.

We have used XPS IIC IP to generate I2C signals. Xilinx APIs were used to send and receive data to the bus for writing and reading, respectively. There were two types of writing and reading of sensor chip control registers: hardware registers and driver variables. Hardware registers could be written and read directly using their address. On the other hand, content of driver variables needed to be written or read via specific hardware registers. To write a driver variable, its address needs to be written in one specific hardware register and then, data needs to be written in another specific hardware register.

FIGURE 5.1: Performing a Register Write Operation to the Sensor

5.1.2.3 Configuring image parameters

Configuring the sensor includes precise steps to be carried out via the I2C interface. Each configuration setting require one or more register read/write(s) via the I2C bus. Figure 5.2 shows the steps to configure an image sensor. Each of these steps are performed using one or more write operations which further executes the steps described in Figure 5.1 for each operation. The first step is to read a specific register to get the device ID of the sensor to be sure that the sensor and I2C interface is functional. Consecutive steps include resetting the MCU, releasing from reset and setting slew rate.

We have used the internal PLL to produce a separate pixel clock from the master clock. The relation of generating a pixel clock from master clock is as follows,

$$pixelclock = \frac{masterclock \times M}{(N+1) \times 8}$$

Where, the values of M and N can be set by writing their values in specific registers. In our project, pixel clock of 50 MHz has been generated with a master clock of 25 MHz having M and N set to 16 and 0, respectively.

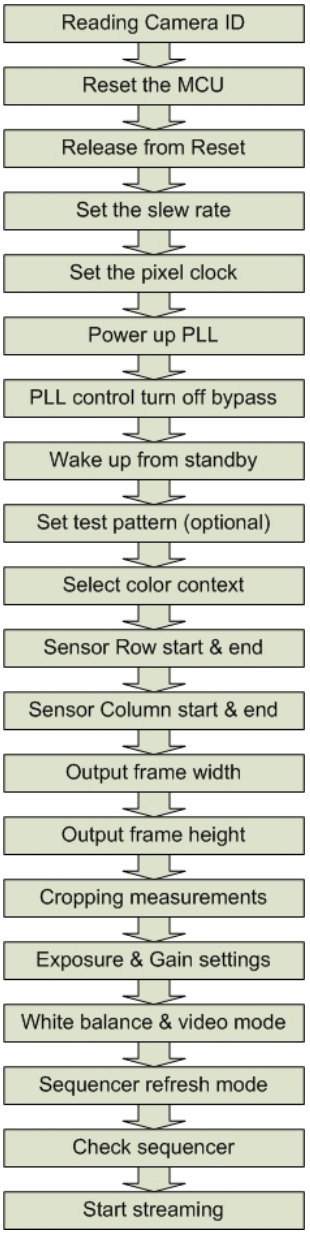


FIGURE 5.2: Programming the Image Sensor

The next steps on the configuration process powers up PLL, turns off bypass and wake up from standby. Remaining configurations defines color format, start and end of sensor row and column, output height and width, cropping measurements, exposure and gain settings, white balance and video capture mode. The last configuration setting starts the video streaming of specified settings.

Additional configuration options include defect correction, noise reduction, aperture correction, gain and threshold for 1D and 2D correction and generating test patterns.

5.2 Pixel data processing

The Aptina MT9D112 image sensor is capable of delivering image outputs in several formats such as RGB, Bayer and YCrCb. We have used the RGB 4:4:4 output format of the camera and then trimmed it into RGB 3:3:2 format due to the data throughput bottleneck caused by the DMA units. Pixels of RGB 3:3:2 format are capable of displaying 256 effective colors for each pixel.

In our design, each frame consists of 640 X 480 = 307200 effective pixels surrounded by blank pixels of horizontal and vertical blanking.

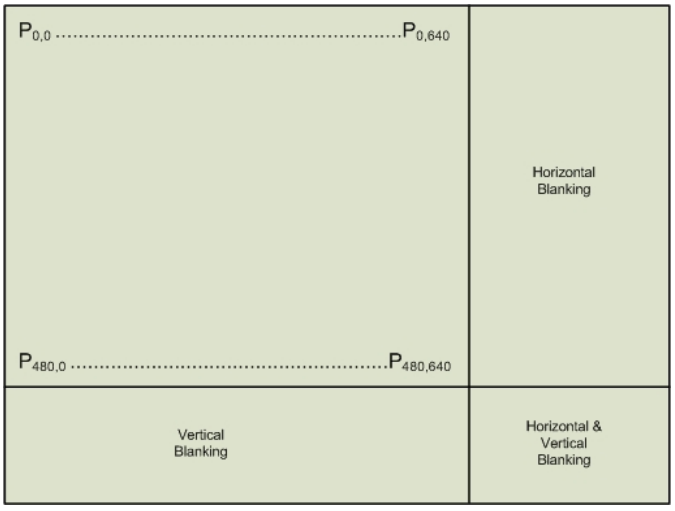


FIGURE 5.3: Pixel placement on the frame (640 X 480)

The sensor produces data on pixel clock - an output clock from the sensor. The DATABIND core sample data on that pixel clock when the Frame valid and Line valid signals from the sensor are high. For each pixel, on the odd byte, 4 bits of R values and 4 bits of G values are received and on the even byte, 4 bits of B values are received. In this way, on every 8th clock, 4 pixel data is produced with RGB 3:3:2 format and pushed together in the READ FIFO as one word (32 bits) to be transferred to DDR2 memory via DMA.

The DATABIND core is configured to generate interrupt requesting DMA transfer when the READ FIFO vacancy count goes below 512 (out of 2048). When a READ FIFO gets full then it cannot further accept incoming data and therefore the data would be

discarded making the image having missing pixels. To prevent this, the READ FIFO is configured to have a DMA transfer whenever it is 75% full.

Chapter 6

Display Unit

The display unit re-translates the pixel information into images and facilitates a visual representation on the standard LCD screen. This includes use of a second pixel clock for displaying pixels, translating the color codes and refreshing the screen.

6.1 Display interface

Standard displays of today (i.e LCD monitors, TVs) support multiple forms of input signals such as VGA, HDMI etc. RGB values could be directly used to produce VGA outputs. But in our project we have used HDMI (High Definition Multimedia Interface) output as HDMI has certain advantages and it allowed us to have some options open for future developments. HDMI uses digital interface and therefore digital images can directly be transmitted instead of converting it to analog format. Standard HDMI cable supports high data bandwidth for high resolution videos. It is capable of loss-less transmission ensuring a high video quality. It also has audio support (optional) within the single cable. HDMI uses TMDS (Transition-Minimized Differential Signaling) which have been produced with RGB values received as pixel data from the image sensor.

6.2 Display core

Several approaches had been made to implement the display core. Initially, an off-the-shelf IP, XPS TFT was first tried to be used as the display unit. After discovering certain incompatibility issues with that core a custom core was developed which received pixel data in RGB format and produced TMDS signals for HDMI output.

Primarily, XPS TFT core were extensively studied to be used in our project. However, after ample investigation, it has been found that, XPS TFT uses a Chrontel CH7301C DVI/HDMI encoder which was not present in our particular board therefore usage of that IP core was discontinued for the project.

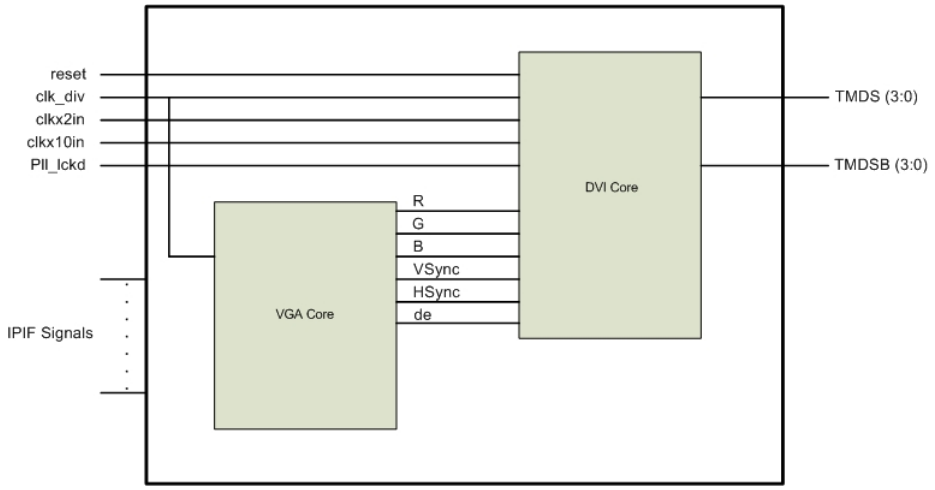


FIGURE 6.1: Organization of the display unit (Simplified)

Later on, a DVI core from a reference design provided from Digilent for the ATLYS board was used to generate HDMI compliant TMDS signals from VGA output signals. The clock signals to the DVI core were needed to be generated by a Phase Locked Loop (PLL) unit and were configured following this relation:

$$Clkoutn = \frac{Clkin1 \frac{C\_CLKFBOUT\_MULT}{C\_DIVCLK\_DIVIDE}}{C\_CLKOUTn\_DIVIDE}$$

The display unit is configured to generate interrupt requesting DMA transfer when the WRITE FIFO occupancy count goes below 512 (out of 2048). Occupancy count describes how occupied the FIFO is. For example, if the occupancy count is zero that means the FIFO is empty. If the FIFO gets empty, the display unit will not receive any pixel data and that would compromise the video image. Therefore to avoid such undesired scenarios, WRITE FIFO generates interrupt requesting DMA transfer whenever the FIFO becomes 75% empty.

Chapter 7

Ethernet Transmission

The key reason of using a robotic vision system is to involve it with the robot control. The idea is, image acquired from the robot vision should contribute in robots decision making. Robots without vision capability generally work with static or predefined measurements. Vision systems allow the robots to act in a more human-like manner. In order to do that, the image acquired by the vision system needs to be transmitted to a processing unit which is responsible for robot control.

To add a transmission capability to our design, we have investigated the feasibility of having an Ethernet transmission system within the scope of this project. Standalone project was initiated to setup and verify Ethernet transmission capabilities on the ATLYS FPGA board. However, due to time shortage, a working design could not be achieved. This chapter will discuss the steps taken to initiate Ethernet transmission.

We started with XPS Ethernet Lite core which could communicate with the on board Ethernet interface. We created a separate XPS project with Ethernet Lite, Multi-port Memory Controller and UART support creating similar environments as the main project so that when it works it could be directly added to the main project.

First, basic features and hardware organization of the Ethernet Lite core were studied including, PLB module interface, Ethernet protocol and transmit interface. For any data to be sent over the Ethernet needs to be in a correctly formed Ethernet Data Frame. The Ethernet data frame should have the following format:

Preamble	Start of Frame Delimiter	Destination Address	Source Address	Type/ Length	Data	Pad	Frame Check Sequence
7 Bytes	1 Byte	6 Bytes	6 Bytes	2 Bytes	0-1500 Bytes	0-46 Bytes	4 Bytes

FIGURE 7.1: Ethernet Data Frame

Following the standard procedure, a program was written using the device drivers for the XPS Ethernet Lite core. Wireshark, a network packet analyzer was installed in another windows machine to observe packets transmitted over the Ethernet. Even though all the drivers returned success, no packets were captured which was transmitted from the FPGA board.

ATLYS base system builder automatically creates reference applications for the Ethernet MAC, which were compiled and tested as well. From the suspicion that the Wireshark installed on the windows machine were not able to properly capture packets, a Linux machine were used further to use Wireshark for capturing packets. In either of the cases, no packets were found to be transferred from the FPGA board. The reference applications were tested in several configurations having the MDIO module as well as internal loopback enabled and disabled.

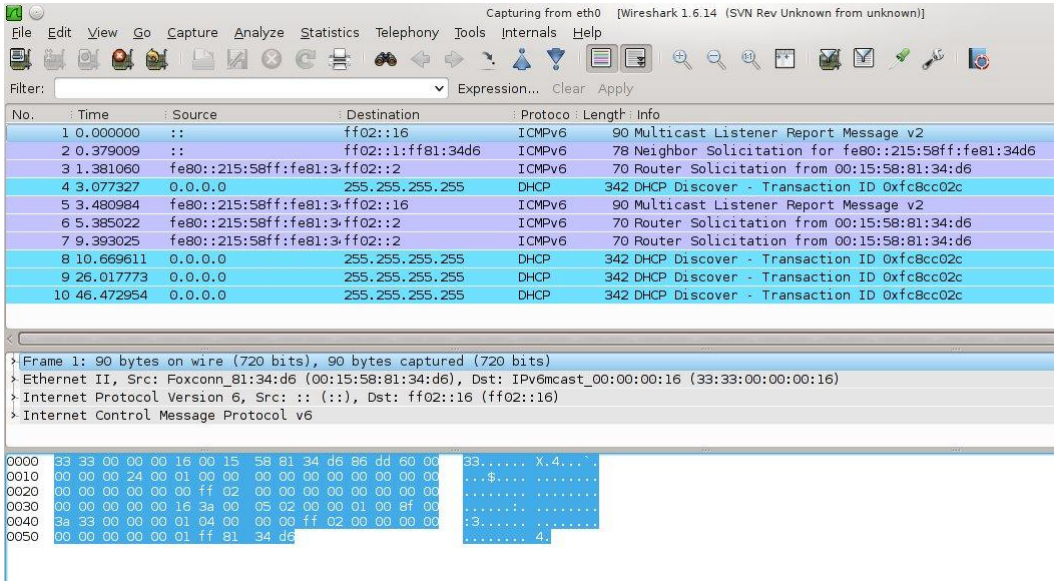


FIGURE 7.2: Capturing Ethernet packets with Wireshark

In the next step of the debugging process, Chipscope cores were inserted into the project and output signals of the Ethernet core were examined. On the Chipscope analyzer, we could actually see the Ethernet packets being in the correct form which were sent. But, still we could not locate any packet that was sent from the board. Assuming problems with cables, test packets were sent from another machine which were successfully received and captured by the Wireshark in the Linux machine.

Doubting if the Ethernet interface on the FPGA board was functioning correctly, all these steps were repeated on another FPGA board (NEXYS3) having similar Ethernet interface. At this stage, it was found that there were floating pins in the design which

needed to be tied to ground and at the same time details on the constraint file were also investigated. However, the problem still remained.

Presuming a possible license requirement as well as version incompatibility, all of these steps were further executed for both of the boards in another workstation having a completely separate installation (upgraded) and licensing. Unfortunately, this also did not return any success.

However, we did test a reference design implemented in XST flow which could successfully send packets from the board over Ethernet. But we found the project just days before our project completion. Porting the project into our main project deemed to be extensively prolonging and therefore discontinued due to specific time constraints of the project.

Chapter 8

Road to the Final Design

As one might assume that the final architecture of this design was pre-determined and was implemented right away. Truth to be told, the final design is a result of continuous evolvement of projects. In this chapter, we will discuss, from where we started, how the design has been transformed to its final stage, alternative implementation techniques that have been tested and difficulties that we have faced during implementation of this project.

8.1 NEXYS 2 Board (Spartan 3) and Pixel Plus Sensor (Discontinued)

We started working on this project initially with a different FPGA platform and image sensor. The FPGA board was a NEXYS 2 board from Digilent equipped with a Spartan 3 FPGA and standard VGA output. The image sensor was PixelPlus PO2030N with a pixel array of 640 X 480. This setup was discontinued after we decided to implement a multi camera system and we continued with the new set of FPGA board and image sensor to develop a new streaming system from the scratch.

8.2 Evolution of designs

This section would briefly explain the approaches that have been taken in different phases of the project and which eventually led us to the final design.

8.2.1 Attempt P1, On chip memory based VGA display

At the very first stage, the challenge was to display anything on the screen via the HDMI port to ensure that the TMDS signaling is working and being recognized by the LCD monitor. In this phase, the system could read color codes written in BRAM and could display them on the screen.

8.2.2 Attempt P2, External memory based VGA display using PLB bus

This system was modified from a project provided by the supervisor which was conducted at the department at an earlier stage. At this stage, the system was able to read external memory locations for pixel data and could display them on the screen. The display unit was connected to the PLB bus and had a WRITE FIFO attached to it for reading pixel data from the memory. At this point, stable horizontal lines could be drawn on the screen.

8.2.3 Attempt P3, Pixel data processing unit as a master on the PLB

The development of the project was stuck for quite a while at this stage. The pixel data processing unit, which received pixel data from the sensor, was unable to write anything on the PLB bus. One of the several approaches that had been tried to resolve this was attaching the custom IP as a master instead of a slave on the PLB bus assuming that it was not being able to write anything on the bus being a slave. But, even after adding the IP as a master on the bus, we were still unable to receive any data from the core. It was then realized that, the pixel data processing unit being a master or slave on the bus was not the primary reason of the problem.

8.2.4 Attempt P4, Direct DMA transfer from ReadFIFO to WriteFIFO

In continuation to the efforts of receiving data from the sensor, a direct DMA transfer between READ FIFO and WRITE FIFO bypassing the external memory were implemented. This configuration resulted with a shaky and blinking screen apparently displaying partials of the test pattern image (color bars). Color bar test pattern image were used to easily detect if we were receiving any partials of the image. If we were not using any test pattern image, it would have been hard to detect partials of a real image as there would have not been any patterns to be recognized in a scrambled image.

8.2.5 Attempt P5, Data transfer using single custom Read/Write FIFO

Instead of using one READ FIFO and one WRITE FIFO on both ends, a custom READ/WRITE FIFO was developed in XST flow in order to streamline the synchronous data transfer. But it lacked signals and mechanisms to be connected with the PLB bus. Implementing PLB protocols manually in the IP was not a convenient process either.

8.2.6 Attempt P6, Using VDMA with PLB bus system

In the process of looking for other alternatives, VDMA (Video DMA) came up seemingly as a promising solution. However, VDMA core was not available right away in the IP catalog. For using in a PLB based system, it needed to be generated with Logicore and added to the XPS project. VDMA was generated and added to the project using the due procedure. To port this IP in the PLB based system we faced similar inconveniences as the previous attempt and unfortunately, we could not achieve a working design using VDMA. Another alternative was to migrate the whole system into AXI based system where a VDMA IP was available right away but that did not seemed to be reasonable choice as we literally had to redo everything from the scratch.

8.2.7 Attempt P7, Separate DMA units for ReadFIFO and WriteFIFO

After putting unsuccessful efforts of using Custom READ/WRITE FIFO and VDMA, we returned back to the closest working version of the project with minor modification of adding another DMA unit to facilitate completely separate DMA transfers on both sides.

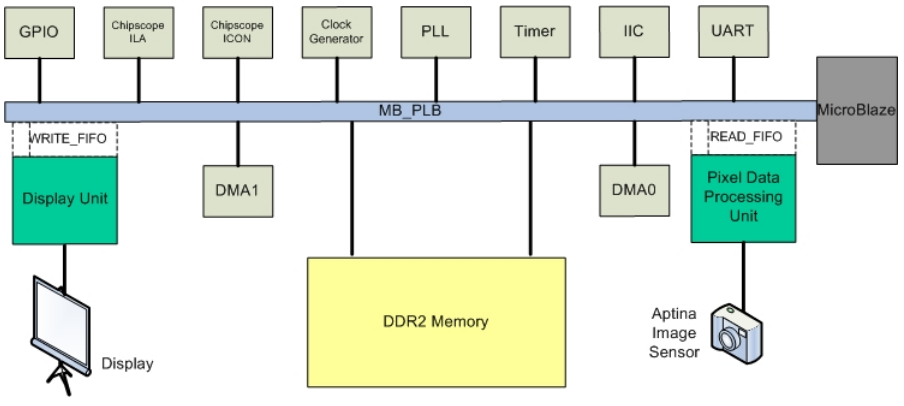


FIGURE 8.1: Block diagram of one of the intermediate design phases (attempt P7)

8.2.8 Attempt P8, Attempt P7 having three PLB busses and two bridges

The only PLB bus of the system had apparently been overloaded with data transactions of the video application. In order to reduce the data traffic on the bus we introduced separate buses for both ends to communicate with the external memory. Two bridges connected the additional buses with the main bus which is connected with Microblaze soft core processor. Each additional bus had a DMA unit connected to them as a master. Both of the DMA units were connected to the main bus as slaves. This configuration resulted a stable frame on the display with noises. At this point, only the display unit (PLB_VGA) generated interrupts to initiate DMA transfer.

8.2.9 Attempt P9, Attempt P8 implemented with both interrupts

During the run, once in a while the stable frame of the attempt P8 project was getting scrambled and READ FIFO was getting full. To deal with this problem, we redesigned the project to have both modules generating separate interrupts for initiating DMA transfers. To implement a 65K color scheme, RGB 5:6:5 color format had been implemented with a System Clock and Pixel Clock of 50 MHz. At this phase we could solve the problem of random scrambling of frames but we still had noises in the image.

8.2.10 Attempt P10, Attempt P9 with 25 MHz system Clock and 256 color

After closely investigating the run time signals of the system with Chipscope, we found that 50 MHz of pixel clock from the image sensor actually corresponds in half pace as it takes two clock cycles to generate information for one pixel. Having the system clock at 50 MHz is actually twice as faster than the image sensor is actually producing the pixels. As a result, the system was being overwritten by the old values of the pixels. So, we reduced the system clock to 25 MHz while keeping the pixel clock at 50 MHz. It reduced the noises but did not completely eliminate them. A further revision in the user constraint file (.ucf) reversing the bit direction of the input signal from the sensor resulted the display with correct color.

8.3 Other alternatives investigated

In addition to steps which have been implemented and tested in the design, there were many other approaches which were investigated and explored but were not implemented in the design.

During the time when, pixel data processing unit was not being able to write anything on the external DDR2 memory, it was suspected that there might be something wrong with the memory interface. So, alternative memory interfaces were investigated such as NPI (Native Port Interface), VFBC (Video Frame Buffer Controller) and MIG (Memory Interface Generator). While having trouble making DMAs working as fast as needed, AXI VDMA seemed as a better alternative. The idea was to use the AXI VDMA using PLB to AXI Bridge in the PLB based system as PLB VDMA was not available right away.

Even though we had thoroughly studied about these alternative approaches but none of them were actually tried in practical design as we had decided to proceed with other approaches described in the previous section.

8.4 Difficulties encountered

This section discusses about some of the difficulties that we had encountered during the implementation of the project.

8.4.1 ODDR2 component

In our project, we needed to forward the system clock to one of the external pins to provide master clock for the image sensor chip. But, specifically for Spartan 6 architectures, a global net cannot be forwarded directly to any of the external pins as it may cause excessive delay. A clock forwarding technique needed to be used to solve the problem. An ODDR2 (Dual Data Rate Output) component was initiated to produce double data rate signals exiting the FPGA.

8.4.2 I/O Standards of UCF constraints

Determining the appropriate I/O standards for user constraints was not quite straight forward as well. In our project we have used the following I/O standards:

Standard	**Description**	**Use**
LVCMOS33	Low Voltage CMOS	General Purpose
TMDS_33	Transition Minimized Differential Signaling	HDMI
I2C	Inter Integrated Circuit	I2C Signals
SSTL18_II	Stub Series Terminated Logic	DDR2 SDRAM
DIFF_SSTL18_II	Pseudo Differential Stub Series Terminated Logic	DDR2 SDRAM

TABLE 8.1: I/O standards used in the design

8.4.3 UART driver

To use the USB-UART as a COM port, a driver from EXAR needed to be installed. The driver installation package included files of .inf, .sys and .cat format. The process of generating batch (.bat) file from these files was also examined.

Chapter 9

Conclusion

In this master thesis project, design and implementation of an embedded vision system for industrial robots has been presented. The prototype implemented on a Spartan 6 platform using an Aptina image sensor is capable of video streaming on 640 X 480 resolution with a frame rate slightly over 9 FPS. Considerable measures were taken to implement Ethernet transmission capability but unfortunately, that had to be left unfinished due to time constraints. This design would serve as a convenient primary base for implementing advanced image processing algorithms for robot vision. The FPGA platform gives an enhanced flexibility for redesigning and reconfiguring the system with updated configurations.

One of the earliest future developments of the design should be acquiring a working data transmission system. It can be started with the standalone project generated to implement Ethernet transmission or it can go a level up by implementing wireless transmission system such as WIFI or Bluetooth. Fortunately, such devices which are ready to be used with our prototype are already available on the market. Further improvements can be introduced by implementing object detection and tracking.

Today, numerous functions can be desired in a robot vision. Research and development in such areas are a continuous process which could take years or decades to develop and test new features. Our prototype would provide a convenient primary platform for such functionalities to be tested and implemented. This platform would allow a kick-start for any present or future enhancements of robot vision to be made.

Developing this vision system has greatly contributed in our knowledge and skills development in various areas of todays computing world. We could acquire in-depth knowledge on digital vision systems, pixel definition in RGB format, generation of digital image from images sensor, and displaying techniques. Programming the image sensor with I2C

interface was particularly a very important expertise to achieve as I2C interfacing is widely used to interface many other modules in different platforms.

This project also allowed us to have practical knowledge of developing a complete embedded system. During the implementation we have came across many important design issues in embedded system including using a soft core processor, initiating embedded operating system, creating application threads, developing custom hardware IPs and their attachment with the main system, interrupt mechanisms, testing and debugging of embedded hardware, using low level device drivers and understanding the usage of FIFO, DMA and memory controller within an embedded system design. These experiences would certainly provide a strong platform for us to work further within the area of embedded system design.

To conclude, designing the embedded vision system has been a great challenge which we could successfully complete. This has greatly inspired us to be engaged in further projects with solid confidence.

Bibliography

[1] National Instruments. " Vision Guided Robotics," 2013, http://www.ni.com/white-paper/10607/en

[2]Ahmed Sh. Khusheef, Ganesh Kothapalli and Majid Tolouei-Rad," An Approach for Integration of Industrial Robotwith Vision System and Simulation Software,"World Academy of Science, Engineering and Technology 58, 2011.

[3] E.C. Dean-Leon, V. Parra-Vega, A. Espinosa-Romero and J. Fierro, " Dynamical Image-based PID Uncalibrated Visual Servoing with Fixed Camera for Tracking of Planar Robots with a Heuristical Predictor," *2nd IEEE International Conference on Industrial Informatics,* Berlin, Germany, 26-26 June 2004, pp. 339 - 345.

[4] Abdul Waheed Malik, Benny Thornberg, Xiaozhou Mengand Muhammad Imran, "Real-time machine vision system using FPGA and soft-core processor",Proc. SPIE8437, Real-Time Image and Video Processing 2012, 84370Z (June 1, 2012);

[5] Henrry Andrian and Kai-Tai Song, " Embedded CMOS imaging system for real-time robotic vision," 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005. (IROS 2005).

[6] Shinchi Hirai, Masakazu Zakoji, Akihiro Masubuchi, Tatsuhiko Tsuboi, " Realtime FPGA-Based Vision System," Journal of Robotics and Mechatronics vol. 17 No.4, 2005.

[7]Aptina Imaging."Aptina MT9D112 Data Sheet," 2013, http://www.aptina.com/ products/soc/mt9d112d00stc/

[8] Xilinx Inc."Spartan-6 Family Overview," 2013, http://www.xilinx.com/ support/-documentation/data_sheets/ds160.pdf

[9]Digilent Inc."Atlys Board Reference Manual,"2013, http://www.digilentinc.com/Data /Products/ATLYS/Atlys_rm.pdf

[10] Digilent Inc."VmodCAM Reference Manual," 2013, http://www.digilentinc.com/ Data/Products/VMOD-CAM/VmodCAM_rm.pdf

[11]Using the I2C Bus, 2013, http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial-.html

[12]Xilinx Homepage, 2013, http://www.xilinx.com/

[13]Digilent Homepage, 2013, http://digilentinc.com/

[14] Xilinx Inc."Atlys Spartan 6 - HDMI output query," 2013, http://forums.xilinx.com/t5/Spartan-Family-FPGAs/Atlys-Spartan-6-HDMI-output-query/td-p/128370

[15] Xilinx Inc."Clock issue about spartan 6," 2013,http://forums.xilinx.com/t5/Spartan-Family-FPGAs/Clock-issue-about-spartan-6/td-p/67284

[16] Xilinx Inc."Spartan 6 Libraries Guide for HDL Designs," 2013, http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_4/spartan6_hdl.pdf

[17] Xilinx Inc. " Spartan 6 FPGA SelectIO Resources," 2013, http://www.xilinx.com/support/documentation/user_guides/ug381.pdf

[18]Wikipedia Homepage, 2013, http://www.wikipedia.org/

[19] Xilinx Inc. " EDK Concepts, Tools, and Techniques," Page 53, http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/edk_ctt.pdf

[20] Cypress Semiconductor, " Understanding Synchronous FIFOs," http://www.cypress.com/?docID=34992

Appendix A

Microprocessor Hardware Specification (MHS) File

PARAMETER VERSION = 2.1.0
PORT fpga_0_RS232_Uart_1_RX_pin = fpga_0_RS232_Uart_1_RX_pin, DIR = I
PORT fpga_0_RS232_Uart_1_TX_pin = fpga_0_RS232_Uart_1_TX_pin, DIR = O
PORT fpga_0_MCB_DDR2_mcbx_dram_addr_pin = fpga_0_MCB_DDR2_mcbx_dram_addr_pin,
DIR = O, VEC = [12:0]
PORT fpga_0_MCB_DDR2_mcbx_dram_ba_pin = fpga_0_MCB_DDR2_mcbx_dram_ba_pin,
DIR = O, VEC = [2:0]
PORT fpga_0_MCB_DDR2_mcbx_dram_ras_n_pin = fpga_0_MCB_DDR2_mcbx_dram_ras_n_pin,
DIR = O
PORT fpga_0_MCB_DDR2_mcbx_dram_cas_n_pin = fpga_0_MCB_DDR2_mcbx_dram_cas_n_pin,
DIR = O
PORT fpga_0_MCB_DDR2_mcbx_dram_we_n_pin = fpga_0_MCB_DDR2_mcbx_dram_we_n_pin,
DIR = O
PORT fpga_0_MCB_DDR2_mcbx_dram_cke_pin = fpga_0_MCB_DDR2_mcbx_dram_cke_pin,
DIR = O
PORT fpga_0_MCB_DDR2_mcbx_dram_clk_pin = fpga_0_MCB_DDR2_mcbx_dram_clk_pin,
DIR = O
PORT fpga_0_MCB_DDR2_mcbx_dram_clk_n_pin = fpga_0_MCB_DDR2_mcbx_dram_clk_n_pin,
DIR = O
PORT fpga_0_MCB_DDR2_mcbx_dram_dq_pin = fpga_0_MCB_DDR2_mcbx_dram_dq_pin,
DIR = IO, VEC = [15:0]
PORT fpga_0_MCB_DDR2_mcbx_dram_dqs_pin = fpga_0_MCB_DDR2_mcbx_dram_dqs_pin,
DIR = IO
PORT fpga_0_MCB_DDR2_mcbx_dram_dqs_n_pin = fpga_0_MCB_DDR2_mcbx_dram_dqs_n_pin,
DIR = IO
PORT fpga_0_MCB_DDR2_mcbx_dram_udqs_pin = fpga_0_MCB_DDR2_mcbx_dram_udqs_pin,
DIR = IO
PORT fpga_0_MCB_DDR2_mcbx_dram_udqs_n_pin = fpga_0_MCB_DDR2_mcbx_dram_udqs_n_pin,
DIR = IO
PORT fpga_0_MCB_DDR2_mcbx_dram_udm_pin = fpga_0_MCB_DDR2_mcbx_dram_udm_pin,
DIR = O
PORT fpga_0_MCB_DDR2_mcbx_dram_ldm_pin = fpga_0_MCB_DDR2_mcbx_dram_ldm_pin,
DIR = O
PORT fpga_0_MCB_DDR2_mcbx_dram_odt_pin = fpga_0_MCB_DDR2_mcbx_dram_odt_pin,
DIR = O

PORT fpga_0_MCB_DDR2_rzq_pin = fpga_0_MCB_DDR2_rzq_pin, DIR = IO
PORT fpga_0_MCB_DDR2_zio_pin = fpga_0_MCB_DDR2_zio_pin, DIR = IO
PORT fpga_0_Ethernet_MAC_PHY_tx_clk_pin = fpga_0_Ethernet_MAC_PHY_tx_clk_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_rx_clk_pin = fpga_0_Ethernet_MAC_PHY_rx_clk_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_crs_pin = fpga_0_Ethernet_MAC_PHY_crs_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_dv_pin = fpga_0_Ethernet_MAC_PHY_dv_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_rx_data_pin = fpga_0_Ethernet_MAC_PHY_rx_data_pin, DIR = I, VEC = [3:0]
PORT fpga_0_Ethernet_MAC_PHY_col_pin = fpga_0_Ethernet_MAC_PHY_col_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_rx_er_pin = fpga_0_Ethernet_MAC_PHY_rx_er_pin, DIR = I
PORT fpga_0_Ethernet_MAC_PHY_rst_n_pin = fpga_0_Ethernet_MAC_PHY_rst_n_pin, DIR = O
PORT fpga_0_Ethernet_MAC_PHY_tx_en_pin = fpga_0_Ethernet_MAC_PHY_tx_en_pin, DIR = O
PORT fpga_0_Ethernet_MAC_PHY_tx_data_pin = fpga_0_Ethernet_MAC_PHY_tx_data_pin, DIR = O, VEC = [3:0]
PORT fpga_0_Ethernet_MAC_PHY_MDC_pin = fpga_0_Ethernet_MAC_PHY_MDC_pin, DIR = O
PORT fpga_0_Ethernet_MAC_PHY_MDIO_pin = fpga_0_Ethernet_MAC_PHY_MDIO_pin, DIR = IO
PORT fpga_0_Ethernet_MAC_MDINT_pin = fpga_0_Ethernet_MAC_MDINT_pin, DIR = I, SIGIS = INTERRUPT
PORT fpga_0_clk_1_sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
PORT fpga_0_rst_1_sys_rst_pin = sys_rst_s, DIR = I, SIGIS = RST, RST_POLARITY = 0
PORT plb_vga_0_TMDS_pin = plb_vga_0_TMDS, DIR = O, VEC = [3:0]
PORT plb_vga_0_TMDSB_pin = plb_vga_0_TMDSB, DIR = O, VEC = [3:0]
PORT xps_iic_0_Sda = xps_iic_0_Sda, DIR = IO
PORT xps_iic_0_Scl = xps_iic_0_Scl, DIR = IO
PORT clkfwd_0_pwdn_pin = clkfwd_0_pwdn, DIR = O
PORT clkfwd_0_mclkex_pin = clkfwd_0_mclkex, DIR = O
PORT clkfwd_0_vdd_pin = clkfwd_0_vdd, DIR = O

PORT plb_vga_0_WFifoemptyLed_pin = plb_vga_0_WFifoemptyLed, DIR = O
PORT clkfwd_0_resetfwd_pin = clkfwd_0_resetfwd, DIR = O
PORT databind_plb_0_CamData_pin = databind_plb_0_CamData, DIR = I, VEC = [0:7]
PORT databind_plb_0_FV_pin = databind_plb_0_FV, DIR = I
PORT databind_plb_0_LV_pin = databind_plb_0_LV, DIR = I
PORT databind_plb_0_fifofull_pin = databind_plb_0_fifofull, DIR = O
PORT plb_vga_0_PLB_INTR_pin = plb_vga_0_PLB_INTR, DIR = O, VEC = [0:0]
PORT databind_plb_0_DATBIntr_pin = databind_plb_0_DATBIntr, DIR = O, VEC = [0:0]
PORT plb_vga_0_IP2INTC_Irpt_pin = plb_vga_0_IP2INTC_Irpt, DIR = O, SIGIS = INTERRUPT, SENSITIVITY = LEVEL_HIGH
PORT databind_plb_0_IP2INTC_Irpt_pin = databind_plb_0_IP2INTC_Irpt, DIR = O, SIGIS = INTERRUPT, SENSITIVITY = LEVEL_HIGH
PORT clkfwd_0_pclkin_pin = clkfwd_0_pclkin, DIR = I

BEGIN microblaze
PARAMETER INSTANCE = microblaze_0
PARAMETER C_USE_BARREL = 1
PARAMETER C_DEBUG_ENABLED = 1
PARAMETER HW_VER = 8.00.a
BUS_INTERFACE DLMB = dlmb
BUS_INTERFACE ILMB = ilmb
BUS_INTERFACE DPLB = mb_plb
BUS_INTERFACE IPLB = mb_plb
BUS_INTERFACE DEBUG = microblaze_0_mdm_bus
PORT MB_RESET = mb_reset
PORT INTERRUPT = microblaze_0_Interrupt
END

BEGIN plb_v46
PARAMETER INSTANCE = mb_plb
PARAMETER HW_VER = 1.05.a
PORT PLB_Clk = clk_50_0000MHzPLL0
PORT SYS_Rst = sys_bus_reset
END

BEGIN lmb_v10
PARAMETER INSTANCE = ilmb
PARAMETER HW_VER = 1.00.a
PORT LMB_Clk = clk_50_0000MHzPLL0
PORT SYS_Rst = sys_bus_reset
END

BEGIN lmb_v10
PARAMETER INSTANCE = dlmb
PARAMETER HW_VER = 1.00.a
PORT LMB_Clk = clk_50_0000MHzPLL0
PORT SYS_Rst = sys_bus_reset
END

BEGIN lmb_bram_if_cntlr
PARAMETER INSTANCE = dlmb_cntlr
PARAMETER HW_VER = 2.10.b
PARAMETER C_BASEADDR = 0x00000000
PARAMETER C_HIGHADDR = 0x0000ffff
BUS_INTERFACE SLMB = dlmb
BUS_INTERFACE BRAM_PORT = dlmb_port
END

BEGIN lmb_bram_if_cntlr
PARAMETER INSTANCE = ilmb_cntlr
PARAMETER HW_VER = 2.10.b
PARAMETER C_BASEADDR = 0x00000000
PARAMETER C_HIGHADDR = 0x0000ffff
BUS_INTERFACE SLMB = ilmb
BUS_INTERFACE BRAM_PORT = ilmb_port
END

BEGIN bram_block
PARAMETER INSTANCE = lmb_bram
PARAMETER HW_VER = 1.00.a
BUS_INTERFACE PORTA = ilmb_port
BUS_INTERFACE PORTB = dlmb_port

END

BEGIN xps_uartlite
PARAMETER INSTANCE = RS232_Uart_1
PARAMETER C_BAUDRATE = 9600
PARAMETER C_DATA_BITS = 8
PARAMETER C_USE_PARITY = 0
PARAMETER C_ODD_PARITY = 0
PARAMETER HW_VER = 1.01.a
PARAMETER C_BASEADDR = 0x84000000
PARAMETER C_HIGHADDR = 0x8400ffff
BUS_INTERFACE SPLB = mb_plb
PORT RX = fpga_0_RS232_Uart_1_RX_pin
PORT TX = fpga_0_RS232_Uart_1_TX_pin
END

BEGIN mpmc
PARAMETER INSTANCE = MCB_DDR2
PARAMETER C_NUM_PORTS = 2
PARAMETER C_PORT_CONFIG = 1
PARAMETER C_MCB_LOC = MEMC3
PARAMETER C_MEM_CALIBRATION_SOFT_IP = TRUE
PARAMETER C_MEM_SKIP_IN_TERM_CAL = 0
PARAMETER C_MEM_SKIP_DYNAMIC_CAL = 0
PARAMETER C_MCB_RZQ_LOC = L6
PARAMETER C_MCB_ZIO_LOC = C2
PARAMETER C_MEM_PARTNO = MT47H64M16XX-25E
PARAMETER C_MEM_ODT_TYPE = 3
PARAMETER C_MEM_DATA_WIDTH = 16
PARAMETER C_DDR2_DQSN_ENABLE = 1
PARAMETER C_PIM0_BASETYPE = 2
PARAMETER HW_VER = 6.02.a
PARAMETER C_PIM1_BASETYPE = 2
PARAMETER C_PIM2_BASETYPE = 0
PARAMETER C_MPMC_BASEADDR = 0x98000000
PARAMETER C_MPMC_HIGHADDR = 0x9FFFFFFF
BUS_INTERFACE SPLB0 = plb_v46_0
BUS_INTERFACE SPLB1 = plb_v46_1

PORT MPMC_Clk0 = clk_50_0000MHzPLL0
PORT MPMC_Rst = sys_periph_reset
PORT mcbx_dram_addr = fpga_0_MCB_DDR2_mcbx_dram_addr_pin
PORT mcbx_dram_ba = fpga_0_MCB_DDR2_mcbx_dram_ba_pin
PORT mcbx_dram_ras_n = fpga_0_MCB_DDR2_mcbx_dram_ras_n_pin
PORT mcbx_dram_cas_n = fpga_0_MCB_DDR2_mcbx_dram_cas_n_pin
PORT mcbx_dram_we_n = fpga_0_MCB_DDR2_mcbx_dram_we_n_pin
PORT mcbx_dram_cke = fpga_0_MCB_DDR2_mcbx_dram_cke_pin
PORT mcbx_dram_clk = fpga_0_MCB_DDR2_mcbx_dram_clk_pin
PORT mcbx_dram_clk_n = fpga_0_MCB_DDR2_mcbx_dram_clk_n_pin
PORT mcbx_dram_dq = fpga_0_MCB_DDR2_mcbx_dram_dq_pin
PORT mcbx_dram_dqs = fpga_0_MCB_DDR2_mcbx_dram_dqs_pin
PORT mcbx_dram_dqs_n = fpga_0_MCB_DDR2_mcbx_dram_dqs_n_pin
PORT mcbx_dram_udqs = fpga_0_MCB_DDR2_mcbx_dram_udqs_pin
PORT mcbx_dram_udqs_n = fpga_0_MCB_DDR2_mcbx_dram_udqs_n_pin
PORT mcbx_dram_udm = fpga_0_MCB_DDR2_mcbx_dram_udm_pin
PORT mcbx_dram_ldm = fpga_0_MCB_DDR2_mcbx_dram_ldm_pin
PORT mcbx_dram_odt = fpga_0_MCB_DDR2_mcbx_dram_odt_pin
PORT rzq = fpga_0_MCB_DDR2_rzq_pin
PORT zio = fpga_0_MCB_DDR2_zio_pin
PORT MPMC_PLL_Lock = Dcm_all_locked
PORT MPMC_Clk_Mem_2x = clk_600_0000MHzPLL0_nobuf
PORT MPMC_Clk_Mem_2x_180 = clk_600_0000MHz180PLL0_nobuf
END

BEGIN xps_ethernetlite
PARAMETER INSTANCE = Ethernet_MAC
PARAMETER HW_VER = 4.00.a
PARAMETER C_BASEADDR = 0x81000000
PARAMETER C_HIGHADDR = 0x8100ffff
BUS_INTERFACE SPLB = mb_plb
PORT PHY_tx_clk = fpga_0_Ethernet_MAC_PHY_tx_clk_pin
PORT PHY_rx_clk = fpga_0_Ethernet_MAC_PHY_rx_clk_pin
PORT PHY_crs = fpga_0_Ethernet_MAC_PHY_crs_pin
PORT PHY_dv = fpga_0_Ethernet_MAC_PHY_dv_pin
PORT PHY_rx_data = fpga_0_Ethernet_MAC_PHY_rx_data_pin
PORT PHY_col = fpga_0_Ethernet_MAC_PHY_col_pin
PORT PHY_rx_er = fpga_0_Ethernet_MAC_PHY_rx_er_pin

PORT PHY_rst_n = fpga_0_Ethernet_MAC_PHY_rst_n_pin
PORT PHY_tx_en = fpga_0_Ethernet_MAC_PHY_tx_en_pin
PORT PHY_tx_data = fpga_0_Ethernet_MAC_PHY_tx_data_pin
PORT PHY_MDC = fpga_0_Ethernet_MAC_PHY_MDC_pin
PORT PHY_MDIO = fpga_0_Ethernet_MAC_PHY_MDIO_pin
END

BEGIN xps_gpio
PARAMETER INSTANCE = LEDs_8Bits
PARAMETER C_ALL_INPUTS = 0
PARAMETER C_GPIO_WIDTH = 8
PARAMETER C_INTERRUPT_PRESENT = 0
PARAMETER C_IS_DUAL = 0
PARAMETER HW_VER = 2.00.a
PARAMETER C_BASEADDR = 0x81420000
PARAMETER C_HIGHADDR = 0x8142ffff
BUS_INTERFACE SPLB = mb_plb
END

BEGIN clock_generator
PARAMETER INSTANCE = clock_generator_0
PARAMETER C_CLKIN_FREQ = 100000000
PARAMETER C_CLKOUT0_FREQ = 600000000
PARAMETER C_CLKOUT0_PHASE = 0
PARAMETER C_CLKOUT0_GROUP = PLL0
PARAMETER C_CLKOUT0_BUF = FALSE
PARAMETER C_CLKOUT1_FREQ = 600000000
PARAMETER C_CLKOUT1_PHASE = 180
PARAMETER C_CLKOUT1_GROUP = PLL0
PARAMETER C_CLKOUT1_BUF = FALSE
PARAMETER C_CLKOUT2_FREQ = 25000000
PARAMETER C_CLKOUT2_PHASE = 0
PARAMETER C_CLKOUT2_GROUP = NONE
PARAMETER C_CLKOUT2_BUF = TRUE
PARAMETER C_EXT_RESET_HIGH = 0
PARAMETER HW_VER = 4.00.a
PARAMETER C_CLKOUT3_FREQ = 100000000
PARAMETER C_CLKOUT3_GROUP = PLL0

PARAMETER C_CLKOUT3_BUF = TRUE
PARAMETER C_CLKOUT3_PHASE = 0
PARAMETER C_CLKOUT4_FREQ = 75000000
PARAMETER C_CLKOUT4_GROUP = PLL0
PARAMETER C_CLKOUT5_FREQ = 50000000
PARAMETER C_CLKOUT5_GROUP = PLL0
PARAMETER C_CLKOUT5_BUF = FALSE
PARAMETER C_CLKOUT6_FREQ = 50000000
PARAMETER C_CLKOUT6_PHASE = 180
PARAMETER C_CLKOUT6_GROUP = PLL0
PARAMETER C_CLKOUT6_BUF = FALSE
PORT CLKIN = dcm_clk_s
PORT CLKOUT0 = clk_600_0000MHzPLL0_nobuf
PORT CLKOUT1 = clk_600_0000MHz180PLL0_nobuf
PORT CLKOUT2 = clk_50_0000MHzPLL0
PORT RST = sys_rst_s
PORT LOCKED = Dcm_all_locked
PORT CLKOUT3 = clk_100_MHzPLL0
PORT CLKOUT4 = clk_75_MHzPLL0
PORT CLKOUT5 = clk_2x_nobuf
PORT CLKOUT6 = clk_2x_nobuf_180
END

BEGIN mdm
PARAMETER INSTANCE = mdm_0
PARAMETER C_MB_DBG_PORTS = 1
PARAMETER C_USE_UART = 1
PARAMETER HW_VER = 2.00.a
PARAMETER C_BASEADDR = 0x84400000
PARAMETER C_HIGHADDR = 0x8440ffff
BUS_INTERFACE SPLB = mb_plb
BUS_INTERFACE MBDEBUG_0 = microblaze_0_mdm_bus
PORT Debug_SYS_Rst = Debug_SYS_Rst
END

BEGIN proc_sys_reset
PARAMETER INSTANCE = proc_sys_reset_0
PARAMETER C_EXT_RESET_HIGH = 0

PARAMETER HW_VER = 3.00.a
PORT Slowest_sync_clk = clk_50_0000MHzPLL0
PORT Ext_Reset_In = sys_rst_s
PORT MB_Debug_Sys_Rst = Debug_SYS_Rst
PORT Dcm_locked = Dcm_all_locked
PORT MB_Reset = mb_reset
PORT Bus_Struct_Reset = sys_bus_reset
PORT Peripheral_Reset = sys_periph_reset
END

BEGIN xps_intc
PARAMETER INSTANCE = xps_intc_0
PARAMETER HW_VER = 2.01.a
PARAMETER C_BASEADDR = 0x81800000
PARAMETER C_HIGHADDR = 0x8180ffff
BUS_INTERFACE SPLB = mb_plb
PORT Intr = plb_vga_0_IP2INTC_Irpt & databind_plb_0_IP2INTC_Irpt & xps_timer_0_Interrupt
PORT Irq = microblaze_0_Interrupt
END

BEGIN bram_block
PARAMETER INSTANCE = bram_block_0
PARAMETER HW_VER = 1.00.a
BUS_INTERFACE PORTA = xps_bram_if_cntlr_0_PORTA
PORT BRAM_Clk_B = clk_50_0000MHzPLL0
END

BEGIN xps_bram_if_cntlr
PARAMETER INSTANCE = xps_bram_if_cntlr_0
PARAMETER HW_VER = 1.00.b
PARAMETER C_SPLB_NATIVE_DWIDTH = 32
PARAMETER C_BASEADDR = 0x83c10000
PARAMETER C_HIGHADDR = 0x83c1ffff
BUS_INTERFACE SPLB = mb_plb
BUS_INTERFACE PORTA = xps_bram_if_cntlr_0_PORTA
END

BEGIN xps_central_dma
PARAMETER INSTANCE = xps_central_dma_0
PARAMETER HW_VER = 2.03.a
PARAMETER C_BASEADDR = 0x80220000
PARAMETER C_HIGHADDR = 0x8022ffff
PARAMETER C_FIFO_DEPTH = 32
PARAMETER C_RD_BURST_SIZE = 16
PARAMETER C_WR_BURST_SIZE = 16
BUS_INTERFACE MPLB = plb_v46_0
BUS_INTERFACE SPLB = mb_plb
PORT IP2INTC_Irpt = xps_central_dma_0_IP2INTC_Irpt
END

BEGIN xps_timer
PARAMETER INSTANCE = xps_timer_0
PARAMETER HW_VER = 1.02.a
PARAMETER C_BASEADDR = 0x83c00000
PARAMETER C_HIGHADDR = 0x83c0ffff
BUS_INTERFACE SPLB = mb_plb
PORT Interrupt = xps_timer_0_Interrupt
END

BEGIN pll_module
PARAMETER INSTANCE = pll_module_0
PARAMETER HW_VER = 2.00.a
PARAMETER C_CLKFBOUT_MULT = 10
PARAMETER C_CLKIN1_PERIOD = 10.0
PARAMETER C_CLKOUT1_DIVIDE = 10
PARAMETER C_CLKOUT2_DIVIDE = 5
PARAMETER C_DIVCLK_DIVIDE = 4
PARAMETER C_CLKIN1_BUF = true
PARAMETER C_CLKOUT1_BUF = true
PARAMETER C_CLKOUT2_BUF = true
PARAMETER C_EXT_RESET_HIGH = 0
PORT CLKFBOUT = pll_module_0_CLKFBOUT
PORT CLKFBIN = pll_module_0_CLKFBOUT
PORT CLKIN1 = dcm_clk_s
PORT RST = sys_rst_s

PORT CLKOUT0 = dvi_clk_10x
PORT CLKOUT1 = dvi_clk
PORT CLKOUT2 = dvi_clk_2x
PORT LOCKED = pll_module_0_LOCKED
END

BEGIN plb_vga
PARAMETER INSTANCE = plb_vga_0
PARAMETER HW_VER = 1.00.a
PARAMETER C_BASEADDR = 0xc9c00000
PARAMETER C_HIGHADDR = 0xc9c0ffff
BUS_INTERFACE SPLB = plb_v46_1
PORT clk_div = dvi_clk
PORT pll_lckd = pll_module_0_LOCKED
PORT clkx2in = dvi_clk_2x
PORT clkx10in = dvi_clk_10x
PORT rst = clkfwd_0_sys_rst_inv
PORT TMDS = plb_vga_0_TMDS
PORT TMDSB = plb_vga_0_TMDSB
PORT IP2INTC_Irpt = plb_vga_0_IP2INTC_Irpt
PORT WFifoemptyLed = plb_vga_0_WFifoemptyLed
PORT CamDataTestOut = plb_vga_0_CamDataTestOut_to_chipscope_ila_0
PORT DBG_FIFO_Empty = plb_vga_0_DBG_FIFO_Empty_to_chipscope_ila_0
PORT DBG_FIFO_Occupancy = plb_vga_0_DBG_FIFO_Occupancy_to_chipscope_ila_0
PORT PLB_INTR = plb_vga_0_PLB_INTR
PORT WF_RdRq = plb_vga_0_WF_RdRq
PORT pcnts = plb_vga_0_pcnts
END

BEGIN xps_iic
PARAMETER INSTANCE = xps_iic_0
PARAMETER HW_VER = 2.03.a
PARAMETER C_SCL_INERTIAL_DELAY = 10
PARAMETER C_SDA_INERTIAL_DELAY = 10
PARAMETER C_BASEADDR = 0x81600000
PARAMETER C_HIGHADDR = 0x8160ffff
BUS_INTERFACE SPLB = mb_plb
PORT Sda = xps_iic_0_Sda

PORT Scl = xps_iic_0_Scl
END

BEGIN clkfwd
PARAMETER INSTANCE = clkfwd_0
PARAMETER HW_VER = 1.00.a
PORT clkin = clk_50_0000MHzPLL0
PORT pwdn = clkfwd_0_pwdn
PORT mclkex = clkfwd_0_mclkex
PORT vdd = clkfwd_0_vdd
PORT resetin = sys_rst_s
PORT resetfwd = clkfwd_0_resetfwd
PORT sys_rst_inv = clkfwd_0_sys_rst_inv
PORT pclkin = clkfwd_0_pclkin
PORT pclkoutbuf = clkfwd_0_pclkoutbuf
END

BEGIN chipscope_icon
PARAMETER INSTANCE = chipscope_icon_0
PARAMETER HW_VER = 1.04.a
PARAMETER C_NUM_CONTROL_PORTS = 2
PORT control1 = chipscope_plbv46_iba_0_icon_ctrl
PORT control0 = chipscope_ila_0_icon_control
END

BEGIN chipscope_plbv46_iba
PARAMETER INSTANCE = chipscope_plbv46_iba_0
PARAMETER HW_VER = 1.03.a
PARAMETER C_NUM_DATA_SAMPLES = 1024
PARAMETER C_USE_MU_5_RD_DBUS = 1
PARAMETER C_USE_MU_4_WR_DBUS = 1
BUS_INTERFACE MON_PLB = mb_plb
PORT chipscope_icon_control = chipscope_plbv46_iba_0_icon_ctrl
PORT PLB_Clk = clk_50_0000MHzPLL0
END

```
BEGIN xps_gpio
PARAMETER INSTANCE = xps_gpio_0
PARAMETER HW_VER = 2.00.a
PARAMETER C_ALL_INPUTS = 1
PARAMETER C_BASEADDR = 0x81400000
PARAMETER C_HIGHADDR = 0x8140ffff
BUS_INTERFACE SPLB = mb_plb
PORT GPIO_IO_I = databind_plb_0_CamDatCHK
END


BEGIN plb_v46
PARAMETER INSTANCE = plb_v46_0
PARAMETER HW_VER = 1.05.a
PORT PLB_Clk = clk_50_0000MHzPLL0
PORT SYS_Rst = sys_bus_reset
END


BEGIN plb_v46
PARAMETER INSTANCE = plb_v46_1
PARAMETER HW_VER = 1.05.a
PORT PLB_Clk = clk_50_0000MHzPLL0
PORT SYS_Rst = sys_bus_reset
END


BEGIN xps_central_dma
PARAMETER INSTANCE = xps_central_dma_1
PARAMETER HW_VER = 2.03.a
PARAMETER C_BASEADDR = 0x80200000
PARAMETER C_HIGHADDR = 0x8020ffff
PARAMETER C_FIFO_DEPTH = 32
PARAMETER C_RD_BURST_SIZE = 16
PARAMETER C_WR_BURST_SIZE = 16
BUS_INTERFACE MPLB = plb_v46_1
BUS_INTERFACE SPLB = mb_plb
PORT IP2INTC_Irpt = xps_central_dma_1_IP2INTC_Irpt
END
```

```
BEGIN plbv46_plbv46_bridge
PARAMETER INSTANCE = plbv46_plbv46_bridge_0
PARAMETER HW_VER = 1.03.a
PARAMETER C_NUM_ADDR_RNG = 2
PARAMETER C_BRIDGE_BASEADDR = 0x86220000
PARAMETER C_BRIDGE_HIGHADDR = 0x8622ffff
PARAMETER C_RNG0_BASEADDR = 0x48000000
PARAMETER C_RNG0_HIGHADDR = 0x4fffffff
PARAMETER C_RNG1_BASEADDR = 0xce200000
PARAMETER C_RNG1_HIGHADDR = 0xce20ffff
BUS_INTERFACE MPLB = plb_v46_0
BUS_INTERFACE SPLB = mb_plb
PORT IP2INTC_Irpt = plbv46_plbv46_bridge_0_IP2INTC_Irpt
END

BEGIN plbv46_plbv46_bridge
PARAMETER INSTANCE = plbv46_plbv46_bridge_1
PARAMETER HW_VER = 1.03.a
PARAMETER C_NUM_ADDR_RNG = 2
PARAMETER C_BRIDGE_BASEADDR = 0x86200000
PARAMETER C_BRIDGE_HIGHADDR = 0x8620ffff
PARAMETER C_RNG0_BASEADDR = 0x98000000
PARAMETER C_RNG0_HIGHADDR = 0x9FFFFFFF
PARAMETER C_RNG1_BASEADDR = 0xc9c00000
PARAMETER C_RNG1_HIGHADDR = 0xc9c0ffff
BUS_INTERFACE MPLB = plb_v46_1
BUS_INTERFACE SPLB = mb_plb
PORT IP2INTC_Irpt = plbv46_plbv46_bridge_1_IP2INTC_Irpt
END

BEGIN databind_plb
PARAMETER INSTANCE = databind_plb_0
PARAMETER HW_VER = 1.00.a
PARAMETER C_BASEADDR = 0xce200000
PARAMETER C_HIGHADDR = 0xce20ffff
BUS_INTERFACE SPLB = plb_v46_0
PORT CamData = databind_plb_0_CamData
PORT FV = databind_plb_0_FV
```

PORT LV = databind_plb_0_LV
PORT fifofull = databind_plb_0_fifofull
PORT CamDatCHK = databind_plb_0_CamDatCHK
PORT reset = sys_rst_s
PORT IP2INTC_Irpt = databind_plb_0_IP2INTC_Irpt
PORT CamDatCHKin = databind_plb_0_CamDatCHKin_to_chipscope_ila_0
PORT PCount = databind_plb_0_PCount_to_chipscope_ila_0
PORT DATBVAC = databind_plb_0_DATBVAC
PORT DATBIntr = databind_plb_0_DATBIntr
PORT FIFORdRq = databind_plb_0_FIFORdRq
PORT clk = clkfwd_0_pclkoutbuf
END

BEGIN chipscope_ila
PARAMETER INSTANCE = chipscope_ila_0
PARAMETER HW_VER = 1.03.a
PARAMETER C_TRIG0_UNITS = 1
PARAMETER C_NUM_DATA_SAMPLES = 2048
PARAMETER C_TRIG0_TRIGGER_IN_WIDTH = 8
PARAMETER C_TRIG1_UNITS = 1
PARAMETER C_TRIG1_TRIGGER_IN_WIDTH = 32
PARAMETER C_TRIG2_UNITS = 1
PARAMETER C_TRIG2_TRIGGER_IN_WIDTH = 3
PARAMETER C_TRIG3_UNITS = 1
PARAMETER C_TRIG3_TRIGGER_IN_WIDTH = 1
PARAMETER C_TRIG4_UNITS = 1
PARAMETER C_TRIG4_TRIGGER_IN_WIDTH = 1
PARAMETER C_TRIG5_UNITS = 1
PARAMETER C_TRIG5_TRIGGER_IN_WIDTH = 1
PARAMETER C_TRIG6_UNITS = 1
PARAMETER C_TRIG6_TRIGGER_IN_WIDTH = 32
PARAMETER C_TRIG7_UNITS = 1
PARAMETER C_TRIG7_TRIGGER_IN_WIDTH = 1
PARAMETER C_TRIG8_UNITS = 1
PARAMETER C_TRIG8_TRIGGER_IN_WIDTH = 12
PARAMETER C_TRIG9_UNITS = 1
PARAMETER C_TRIG9_TRIGGER_IN_WIDTH = 12
PARAMETER C_TRIG10_UNITS = 1

PARAMETER C_TRIG10_TRIGGER_IN_WIDTH = 1
PARAMETER C_TRIG11_UNITS = 1
PARAMETER C_TRIG11_TRIGGER_IN_WIDTH = 1
PARAMETER C_TRIG12_UNITS = 1
PARAMETER C_TRIG12_TRIGGER_IN_WIDTH = 2
PORT chipscope_ila_control = chipscope_ila_0_icon_control
PORT CLK = clk_50_0000MHzPLL0
PORT TRIG0 = databind_plb_0_CamDatCHKin_to_chipscope_ila_0
PORT TRIG1 = databind_plb_0_CamDatCHK
PORT TRIG2 = databind_plb_0_PCount_to_chipscope_ila_0
PORT TRIG3 = databind_plb_0_IP2INTC_Irpt
PORT TRIG4 = databind_plb_0_fifofull
PORT TRIG5 = plb_vga_0_IP2INTC_Irpt
PORT TRIG6 = plb_vga_0_CamDataTestOut_to_chipscope_ila_0
PORT TRIG7 = plb_vga_0_DBG_FIFO_Empty_to_chipscope_ila_0
PORT TRIG8 = plb_vga_0_DBG_FIFO_Occupancy_to_chipscope_ila_0
PORT TRIG9 = databind_plb_0_DATBVAC
PORT TRIG10 = databind_plb_0_FIFORdRq
PORT TRIG11 = plb_vga_0_WF_RdRq
PORT TRIG12 = plb_vga_0_pcnts
END

Appendix B

Microprocessor Software Specification (MSS) File

```
PARAMETER VERSION = 2.2.0
BEGIN OS
PARAMETER OS_NAME = xilkernel
PARAMETER OS_VER = 5.00.a
PARAMETER PROC_INSTANCE = microblaze_0
PARAMETER systmr_dev = xps_timer_0
PARAMETER stdin = RS232_Uart_1
PARAMETER stdout = RS232_Uart_1
PARAMETER sysintc_spec = xps_intc_0
PARAMETER static_pthread_table = ((my_main,1))
END

BEGIN PROCESSOR
PARAMETER DRIVER_NAME = cpu
PARAMETER DRIVER_VER = 1.13.a
PARAMETER HW_INSTANCE = microblaze_0
PARAMETER COMPILER = mb-gcc
PARAMETER ARCHIVER = mb-ar
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = bram
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = dlmb_cntlr
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = bram
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = ilmb_cntlr
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = generic
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = lmb_bram
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = uartlite
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = RS232_Uart_1
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = mpmc
PARAMETER DRIVER_VER = 4.01.a
PARAMETER HW_INSTANCE = MCB_DDR2
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = emaclite
PARAMETER DRIVER_VER = 3.01.a
PARAMETER HW_INSTANCE = Ethernet_MAC
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = gpio
PARAMETER DRIVER_VER = 3.00.a
PARAMETER HW_INSTANCE = LEDs_8Bits
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = generic
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = clock_generator_0
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = uartlite
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = mdm_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = generic
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = proc_sys_reset_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = intc
PARAMETER DRIVER_VER = 2.01.a
PARAMETER HW_INSTANCE = xps_intc_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = bram
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = xps_bram_if_cntlr_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = dmacentral
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = xps_central_dma_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = tmrctr
PARAMETER DRIVER_VER = 2.01.a
PARAMETER HW_INSTANCE = xps_timer_0
END
```

```
BEGIN DRIVER
PARAMETER DRIVER_NAME = plb_vga
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = plb_vga_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = iic
PARAMETER DRIVER_VER = 2.01.a
PARAMETER HW_INSTANCE = xps_iic_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = gpio
PARAMETER DRIVER_VER = 3.00.a
PARAMETER HW_INSTANCE = xps_gpio_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = generic
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = plb_v46_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = generic
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = plb_v46_1
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = dmacentral
PARAMETER DRIVER_VER = 2.00.a
PARAMETER HW_INSTANCE = xps_central_dma_1
END
```

BEGIN DRIVER
PARAMETER DRIVER_NAME = generic
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = plbv46_plbv46_bridge_0
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = generic
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = plbv46_plbv46_bridge_1
END

BEGIN DRIVER
PARAMETER DRIVER_NAME = databind_plb
PARAMETER DRIVER_VER = 1.00.a
PARAMETER HW_INSTANCE = databind_plb_0
END

Appendix C

User Constraint File (UCF)

Net fpga_0_RS232_Uart_1_RX_pin LOC=A16 — IOSTANDARD=LVCMOS33;
Net fpga_0_RS232_Uart_1_TX_pin LOC=B16 — IOSTANDARD=LVCMOS33;
Net fpga_0_MCB_DDR2_rzq_pin IOSTANDARD = SSTL18_II;
Net fpga_0_MCB_DDR2_zio_pin IOSTANDARD = SSTL18_II;
Net fpga_0_Ethernet_MAC_PHY_tx_clk_pin LOC=K16 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_clk_pin LOC=K15 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_crs_pin LOC=C18 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_dv_pin LOC=F17 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin¡3¿ LOC=F15 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin¡2¿ LOC=E16 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin¡1¿ LOC=H14 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_data_pin¡0¿ LOC=G16 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_col_pin LOC=C17 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rx_er_pin LOC=F18 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_rst_n_pin LOC=G13 — TIG — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_tx_en_pin LOC=H15 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin¡3¿ LOC=K13 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin¡2¿ LOC=K14 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin¡1¿ LOC=H13 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_tx_data_pin¡0¿ LOC=H16 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_MDC_pin LOC=F16 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_PHY_MDIO_pin LOC=N17 — IOSTANDARD=LVCMOS33;
Net fpga_0_Ethernet_MAC_MDINT_pin LOC=L16 — IOSTANDARD=LVCMOS33;

Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100000 kHz HIGH 50 %;
Net fpga_0_clk_1_sys_clk_pin LOC=L15 — IOSTANDARD=LVCMOS33;

PIN "clock_generator_0/clock_generator_0/PLL0_CLKOUT2_BUFG_INST.O"
CLOCK_DEDICATED_ROUTE = FALSE;

Net fpga_0_rst_1_sys_rst_pin TIG;
Net fpga_0_rst_1_sys_rst_pin LOC=T15 — IOSTANDARD=LVCMOS33;

NET databind_plb_0_fifofull_pin LOC = M14 — IOSTANDARD = LVCMOS33 ;
NET plb_vga_0_WFifoemptyLed_pin LOC = U18 — IOSTANDARD = LVCMOS33 ;

NET databind_plb_0_DATBIntr_pin¡0¿ LOC = M13 — IOSTANDARD = LVCMOS33 ;
NET plb_vga_0_PLB_INTR_pin¡0¿ LOC = N14 — IOSTANDARD = LVCMOS33 ;

NET databind_plb_0_IP2INTC_Irpt_pin LOC = P16 — IOSTANDARD = LVCMOS33
;
NET plb_vga_0_IP2INTC_Irpt_pin LOC = L14 — IOSTANDARD = LVCMOS33 ;

NET plb_vga_0_TMDS_pin¡0¿ LOC = D8 — IOSTANDARD = TMDS_33 ;
NET plb_vga_0_TMDSB_pin¡0¿ LOC = C8 — IOSTANDARD = TMDS_33 ;
NET plb_vga_0_TMDS_pin¡1¿ LOC = C7 — IOSTANDARD = TMDS_33 ;
NET plb_vga_0_TMDSB_pin¡1¿ LOC = A7 — IOSTANDARD = TMDS_33 ;
NET plb_vga_0_TMDS_pin¡2¿ LOC = B8 — IOSTANDARD = TMDS_33 ;
NET plb_vga_0_TMDSB_pin¡2¿ LOC = A8 — IOSTANDARD = TMDS_33 ;
NET plb_vga_0_TMDS_pin¡3¿ LOC = B6 — IOSTANDARD = TMDS_33 ;
NET plb_vga_0_TMDSB_pin¡3¿ LOC = A6 — IOSTANDARD = TMDS_33 ;

NET xps_iic_0_Scl LOC = V12 — IOSTANDARD = I2C; # Bank = 2, Pin name =
*IO_L19N, Sch name = EXP-IO6_N
NET xps_iic_0_Sda LOC = P11 — IOSTANDARD = I2C; # Bank = 2, Pin name =
*IO_L20N, Sch name = EXP-IO7_N

NET databind_plb_0_CamData_pin¡7¿ LOC = M10 — IOSTANDARD = LVCMOS33
— IN_TERM = UNTUNED_SPLIT_50;
NET databind_plb_0_CamData_pin¡6¿ LOC = U11 — IOSTANDARD = LVCMOS33 —
IN_TERM = UNTUNED_SPLIT_50;
NET databind_plb_0_CamData_pin¡5¿ LOC = V15 — IOSTANDARD = LVCMOS33 —
IN_TERM = UNTUNED_SPLIT_50;
NET databind_plb_0_CamData_pin¡4¿ LOC = U16 — IOSTANDARD = LVCMOS33 —
IN_TERM = UNTUNED_SPLIT_50;
NET databind_plb_0_CamData_pin¡3¿ LOC = U15 — IOSTANDARD = LVCMOS33 —

IN_TERM = UNTUNED_SPLIT_50;
NET databind_plb_0_CamData_pin¡2¿ LOC = U13 — IOSTANDARD = LVCMOS33 — IN_TERM = UNTUNED_SPLIT_50;
NET databind_plb_0_CamData_pin¡1¿ LOC = M11 — IOSTANDARD = LVCMOS33 — IN_TERM = UNTUNED_SPLIT_50;
NET databind_plb_0_CamData_pin¡0¿ LOC = R11 — IOSTANDARD = LVCMOS33 — IN_TERM = UNTUNED_SPLIT_50;

NET databind_plb_0_FV_pin LOC = T12 — IOSTANDARD = LVCMOS33 — IN_TERM = UNTUNED_SPLIT_50;
NET databind_plb_0_LV_pin LOC = N10 — IOSTANDARD = LVCMOS33 — IN_TERM = UNTUNED_SPLIT_50;

NET clkfwd_0_pclkin_pin LOC = R10 — IOSTANDARD = LVCMOS33 — IN_TERM = UNTUNED_SPLIT_50;

NET clkfwd_0_pwdn_pin LOC = T11 — IOSTANDARD = LVCMOS33; # PWDN 0
NET clkfwd_0_mclkex_pin LOC = N11 — IOSTANDARD = LVCMOS33; # MCLK 100 MHz
NET clkfwd_0_vdd_pin LOC = V16 — IOSTANDARD = LVCMOS33; # VDD = 1
NET clkfwd_0_resetfwd_pin LOC = V13 — IOSTANDARD = LVCMOS33;

CONFIG VCCAUX=3.3; # Valid values are 2.5 and 3.3

CONFIG MCB_PERFORMANCE= EXTENDED;

NET fpga_0_MCB_DDR2_mcbx_dram_clk_pin LOC = G3 — IOSTANDARD = DIFF_SSTL18_II; # Bank = 3, Pin name = IO_L46P_M3CLK, Sch name = DDR-CK_P

NET fpga_0_MCB_DDR2_mcbx_dram_clk_n_pin LOC = G1 — IOSTANDARD = DIFF_SSTL18_II; # Bank = 3, Pin name = IO_L46N_M3CLKN, Sch name = DDR-CK_N

NET fpga_0_MCB_DDR2_mcbx_dram_cke_pin LOC = H7 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L53P_M3CKE, Sch name = DDR-CKE

NET fpga_0_MCB_DDR2_mcbx_dram_ras_n_pin LOC = L5 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L43P_GCLK23_M3RASN, Sch name = DDR-RAS

NET fpga_0_MCB_DDR2_mcbx_dram_cas_n_pin LOC = K5 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L43N_GCLK22_IRDY2_M3CASN, Sch name = DDR-CAS

NET fpga_0_MCB_DDR2_mcbx_dram_we_n_pin LOC = E3 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L50P_M3WE, Sch name = DDR-WE

NET fpga_0_MCB_DDR2_rzq_pin LOC = L6; # Bank = 3, Pin name = IO_L31P, Sch name = RZQ

NET fpga_0_MCB_DDR2_zio_pin LOC = C2; # Bank = 3, Pin name = IO_L83P, Sch name = ZIO

NET fpga_0_MCB_DDR2_mcbx_dram_ba_pin¡0¿ LOC = F2 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L48P_M3BA0, Sch name = DDR-BA0

NET fpga_0_MCB_DDR2_mcbx_dram_ba_pin¡1¿ LOC = F1 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L48N_M3BA1, Sch name = DDR-BA1

NET fpga_0_MCB_DDR2_mcbx_dram_ba_pin¡2¿ LOC = E1 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L50N_M3BA2, Sch name = DDR-BA2

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡0¿ LOC = J7 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L47P_M3A0, Sch name = DDR-A0

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡1¿ LOC = J6 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L47N_M3A1, Sch name = DDR-A1

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡2¿ LOC = H5 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L49N_M3A2, Sch name = DDR-A2

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡3¿ LOC = L7 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L45P_M3A3, Sch name = DDR-A3

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡4¿ LOC = F3 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L51N_M3A4, Sch name = DDR-A4

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡5¿ LOC = H4 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L44P_GCLK21_M3A5, Sch name = DDR-A5

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡6¿ LOC = H3 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L44N_GCLK20_M3A6, Sch name = DDR-A6

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡7¿ LOC = H6 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L49P_M3A7, Sch name = DDR-A7

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡8¿ LOC = D2 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L52P_M3A8, Sch name = DDR-A8

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡9¿ LOC = D1 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L52N_M3A9, Sch name = DDR-A9

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡10¿ LOC = F4 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L51P_M3A10, Sch name = DDR-A10

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡11¿ LOC = D3 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L54N_M3A11, Sch name = DDR-A11

NET fpga_0_MCB_DDR2_mcbx_dram_addr_pin¡12¿ LOC = G6 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L53N_M3A12, Sch name = DDR-A12

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡0¿ LOC = L2 — IOSTANDARD = SSTL18_II; # Bank = 3, Pin name = IO_L37P_M3DQ0, Sch name = DDR-DQ0

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡1¿ LOC = L1 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L37N_M3DQ1, Sch name = DDR-DQ1

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡2¿ LOC = K2 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L38P_M3DQ2, Sch name = DDR-DQ2

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡3¿ LOC = K1 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L38N_M3DQ3, Sch name = DDR-DQ3

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡4¿ LOC = H2 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L41P_GCLK27_M3DQ4, Sch name = DDR-DQ4

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡5¿ LOC = H1 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L41N_GCLK26_M3DQ5, Sch name = DDR-DQ5

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡6¿ LOC = J3 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L40P_M3DQ6, Sch name = DDR-DQ6

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡7¿ LOC = J1 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L40N_M3DQ7, Sch name = DDR-DQ7

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡8¿ LOC = M3 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L36P_M3DQ8, Sch name = DDR-DQ8

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡9¿ LOC = M1 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L36N_M3DQ9, Sch name = DDR-DQ9

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡10¿ LOC = N2 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L35P_M3DQ10, Sch name = DDR-DQ10

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡11¿ LOC = N1 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L35N_M3DQ11, Sch name = DDR-DQ11

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡12¿ LOC = T2 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L33P_M3DQ12, Sch name = DDR-DQ12

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡13¿ LOC = T1 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L33N_M3DQ13, Sch name = DDR-DQ13

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡14¿ LOC = U2 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L32P_M3DQ14, Sch name = DDR-DQ14

NET fpga_0_MCB_DDR2_mcbx_dram_dq_pin¡15¿ LOC = U1 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L32N_M3DQ15, Sch name = DDR-DQ15

NET fpga_0_MCB_DDR2_mcbx_dram_udqs_pin LOC = P2 — IOSTANDARD = DIFF_SSTL18_II;
Bank = 3, Pin name = IO_L34P_M3UDQS, Sch name = DDR-UDQS_P

NET fpga_0_MCB_DDR2_mcbx_dram_udqs_n_pin LOC = P1 — IOSTANDARD = DIFF_SSTL18_II;
Bank = 3, Pin name = IO_L34N_M3UDQSN, Sch name = DDR-UDQS_N

NET fpga_0_MCB_DDR2_mcbx_dram_dqs_pin LOC = L4 — IOSTANDARD = DIFF_SSTL18_II;
Bank = 3, Pin name = IO_L39P_M3LDQS, Sch name = DDR-LDQS_P

NET fpga_0_MCB_DDR2_mcbx_dram_dqs_n_pin LOC = L3 — IOSTANDARD = DIFF_SSTL18_II;
Bank = 3, Pin name = IO_L39N_M3LDQSN, Sch name = DDR-LDQS_N

NET fpga_0_MCB_DDR2_mcbx_dram_ldm_pin LOC = K3 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L42N_GCLK24_M3LDM, Sch name = DDR-LDM

NET fpga_0_MCB_DDR2_mcbx_dram_udm_pin LOC = K4 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L42P_GCLK25_TRDY2_M3UDM, Sch name = DDR-UDM

NET fpga_0_MCB_DDR2_mcbx_dram_odt_pin LOC = K6 — IOSTANDARD = SSTL18_II;
Bank = 3, Pin name = IO_L45N_M3ODT, Sch name = DDR-ODT