FPGA Platform for Debug

Master Thesis in System-on-Chip Design

By
Dandan Yan

Supervisors:
Industry: Andreas Magnusson
KTH: Prof. Ahmed Hemani

Examiner:
Prof. Ahmed Hemani

ERICSSON AB

DEPARTMENT OF ELECTRONIC SYSTEMS
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
KUNGLIGA TEKNISKA HÖGSKOLAN

Stockholm, November 2011
“For the things we have to learn before we can do them, we learn by doing them.”

Aristotle, 384BC (Stageira) – 322BC (Chalcis)
This page has been intentionally left blank.
Abstract

FPGA-based prototyping is increasingly used in ASIC developments today to allow hardware/software co-design well ahead of chip fabrication and acceleration of functional verification of ASIC designs and the surrounding system. This thesis addresses an FPGA platform that can be applied in the development of next generation target debugger system. This FPGA platform can also be used for prototyping and verification acceleration of Ericsson ASIC IP:s.

Micro Ericsson ASIC (μ-EA) system is implemented in Xilinx Virtex-6 FPGA board with the SERDES port that can be tested at 1.25/2.5Gbps. This design costs 62% of FPGA device resources at a frequency of 2.5MHz. The access of Trace Buffer, MEMORY and DSP inside μ-EA proves achievable as well. Moreover, DSP core inside μ-EA is capable of running software and sending out trace messages to debug block, using DebugTool through a Nexus trace probe.

This FPGA platform combined with probe and debug tools developed by Ericsson Digital ASIC Unit can be used to verify and validate the next generation debugger systems before ASIC arrives.
Acknowledgements

I am grateful towards Ericsson AB for offering the opportunity of this master thesis at Digital ASIC department in Kista. I would like to express my thanks to the section manager, Hans-Olov Eriksson for making this thesis possible and professor from KTH, Ahmed Hemani for kindness to be my examiner as well as supervisor.

I offer my sincerest gratitude to my supervisor, Andreas Magnusson, who has supported me throughout my thesis with his patience and knowledge. Without his encouragement, guidance and support from the initial to the final level, this thesis will not be smoothly accomplished.

I have been aided a lot regarding usage of DebugTool by Mats Fredriksson, a fine specialist who contributed a great deal of time and effort to setup DebugTool environment for this thesis.

Lars Johan Fritz has offered much knowledge with respect to Nexus probe and given useful guidance in smooth setup of Nexus probe along with DebugTool.

Michael Carlsson C has provided a comprehensive knowledge of MEMORY and offered a great help to resolve the MEMORY access problem.

Anders Nordström, a FPGA designer and Stefan Rydback, a technical support specialist from Xilinx, have offered much advice and help regarding FPGA design tools.

In my daily work, a number of colleagues have graciously provided their help on the issues covered in this thesis. In this regard, I am grateful to Jakob Brundin, Orri Tomasson, Ulf Sällström XB and Ioannis Savvidis.
Table of Contents

CHAPTER 1 ......................................................................................................................................... 1

INTRODUCTION ................................................................................................................................... 1

1.1 BACKGROUND ............................................................................................................................ 1

1.1.1 A Brief Overview of EA System ...................................................................................... 1

1.1.2 An Overview of Debugger System ................................................................................. 1

1.1.3 FPGA-based Prototyping .............................................................................................. 2

1.2 THESIS GOALS .................................................................................................................... 3

1.3 THESIS STRUCTURE ........................................................................................................... 4

CHAPTER 2 ....................................................................................................................................... 5

AN OVERVIEW OF MICRO SYSTEM ............................................................................................ 5

2.1 ARCHITECTURAL OVERVIEW ............................................................................................ 5

2.1.1 Debug .................................................................................................................................. 6

2.1.2 MEMORY Controller ........................................................................................................ 6

2.1.3 DSP ..................................................................................................................................... 7

2.1.4 BUS .................................................................................................................................... 7

2.1.5 MISC .................................................................................................................................. 7

2.1.6 Clock Generation ............................................................................................................. 8

2.2 WORKING ENVIRONMENT OVERVIEW ............................................................................ 8

2.2.1 Operating principle ........................................................................................................... 9

2.2.2 Hardware .......................................................................................................................... 10

2.2.3 Environemnt and design tools ........................................................................................ 10

2.2.4 Language ........................................................................................................................ 12

CHAPTER 3 ....................................................................................................................................... 13

IMPLEMENTATION OF μ-EA ON FPGA ..................................................................................... 13

3.1 TARGETING DEBUG ON FPGA ........................................................................................... 13

3.1.1 Modifications on Debug Core ....................................................................................... 14

3.1.2 Adding GTX in Debug Top Level .................................................................................... 14

3.1.3 Setting up Debugging Environment .............................................................................. 19

3.1.4 Accessing Trace Buffer .................................................................................................... 20

3.2 ADDING MEMORY .................................................................................................................. 22

3.2.1 Configuration of RAM ................................................................................................. 23

3.2.2 Accessing MEMORY ........................................................................................................ 24

3.3 ADDING DSP .......................................................................................................................... 28

3.4 ADDING MISC BLOCK ........................................................................................................... 29

3.5 CLOCK CONFIGURATION ....................................................................................................... 30

3.6 SYNTHESIS AND IMPLEMENTATION RESULTS ............................................................. 31

3.6.1 FPGA Design Flow ........................................................................................................ 31

3.6.2 Synthesis Result ............................................................................................................. 33

3.6.3 Place & Route Result ...................................................................................................... 35

3.6.4 Post PAR Static Timing Analysis Result ....................................................................... 36

3.7 SUMMARY .............................................................................................................................. 37

CHAPTER 4 ..................................................................................................................................... 38

SERDES TEST ................................................................................................................................. 38

4.1 DESIGNING AURORA RECEIVER ....................................................................................... 38

4.1.1 Architecture Overview ..................................................................................................... 38

4.1.2 Implementation Flow ....................................................................................................... 39

4.2 VERIFYING DEBUG GTX TX .............................................................................................. 42
4.2.1 Architecture Overview ................................................................. 42
4.2.2 Implementation Flow ................................................................. 43
4.2.3 Problems Encountered ............................................................... 45
4.2.4 Results .......................................................................................... 46
4.3 SUMMARY .......................................................................................... 50

CHAPTER 5 ................................................................................................. 51

SW EXECUTION AND TRACING ............................................................. 51

5.1 OVERVIEW OF SW EXECUTION ....................................................... 51
5.2 IMPLEMENTATION .............................................................................. 53
  5.2.1 Function of basic program example .............................................. 53
  5.2.2 Problems encountered ............................................................... 55
5.3 RESULTS OF SW EXECUTION ............................................................ 59
5.4 SUMMARY .......................................................................................... 61

CHAPTER 6 ................................................................................................. 63

CONCLUSION ............................................................................................ 63

  6.1 ACHIEVEMENTS ............................................................................... 63
  6.2 FUTURE WORK .................................................................................. 64

APPENDICES ............................................................................................. 65

  APPENDIX A: APPLICATION OF TAP REGISTERS ................................. 65
  APPENDIX B: FMC CONNECTION PINOUT ............................................ 68

REFERENCES ............................................................................................ 71
List of Figures

Figure 1-1: Debugger system overview.................................................................2

Figure 2-1: Architectural Overview of µ-EA .........................................................6
Figure 2-2: Overview of working environment.....................................................9

Figure 3-1: TXOUTCLK Drives TXUSRCLK2 (2-Byte Mode) [4].........................17
Figure 3-2: MEMORY interface [10]....................................................................22
Figure 3-3: Single-port RAM [5]..........................................................................23
Figure 3-4: RAM IP Symbol..................................................................................24
Figure 3-5: Address Definition ...........................................................................26
Figure 3-6: MISC Block Interface .......................................................................30
Figure 3-7: Block Diagram of Clock Generation ..................................................31
Figure 3-8: FPGA Design Flow .........................................................................32

Figure 4-1: Aurora 8B/10B Simplex Channel Overview [8].................................39
Figure 4-2: Channel Setup Procedure .................................................................41
Figure 4-3: GTX TX and Aurora RX Channel Overview .....................................42
Figure 4-4: Transition from Align to Verify (oscilloscope) ..................................46
Figure 4-5: Transition from Align to Verify (simulation) ....................................46
Figure 4-6: Init sequence received on RX ...........................................................48
Figure 4-7: Bond sequence received on RX .........................................................48
Figure 4-8: Verify sequence received on RX .......................................................49
Figure 4-9: Data phase .........................................................................................49

Figure 5-1: JTAG and Trace Interface between Debug and DSP .......................52
Figure 5-2: BUS Interface ..................................................................................58
Figure 5-3: Trsetup Register Configuration ........................................................59
Figure 5-4: Trdecode Result ..............................................................................60
List of Tables

Table 3-1: Configuration of GTX TX.................................................................16
Table 3-2: PLL Divider Attribute and Common Values .................................18
Table 3-3: PLL Divider Settings for Aurora Protocol ...................................19
Table 3-4: TX PLL Output Divider Setting ....................................................19
Table 3-5: Device utilization summary of synthesis ....................................34
Table 3-6: Device utilization summary of synthesis ....................................36
Table 3-7: Derived Constraints for TS_SYSCLK_P ......................................36

Table 4-1: Line rate settings .........................................................................43
Table 4-2: Aurora 8B/10B Ordered Sets .......................................................47
Table 4-3: Valid Data Characters and Valid K (control) Characters .............47

Table A-1: TAP data registers to configure GTX...........................................67

Table B-1: JTAG 1 Connector Pinout .............................................................68
Table B-2: JTAG 2 Connector Pinout .............................................................68
Table B-3: Samtec Connector ERx8 High Speed Serial for Transmitter 1 (High Speed Simplex) .................................................................69
Table B-4: Samtec Connector ERx8 High Speed Serial Transmitter 2 (High Speed Simplex) .................................................................69
Table B-5: Samtec Connector ERx8 High Speed Serial for Receiver 1 (High Speed Simplex) .................................................................70
Table B-6: Samtec Connector ERx8 High Speed Serial for Receiver 2 (High Speed Simplex) .................................................................70
List of Equations

Equation 3-1........................................................................................................... 18
Equation 3-2........................................................................................................... 18
Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>DBG</td>
<td>DeBuG</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital Signal Processor</td>
</tr>
<tr>
<td>ECC</td>
<td>Error Correcting Code</td>
</tr>
<tr>
<td>FMC</td>
<td>FPGA Mezzanine Connector</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field-Programmable Gate Array</td>
</tr>
<tr>
<td>GTX</td>
<td>Gigabit Transceiver</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HPC</td>
<td>High Pin Count</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>IWD</td>
<td>InterWork Description</td>
</tr>
<tr>
<td>LDM</td>
<td>Local Data Memory</td>
</tr>
<tr>
<td>LPM</td>
<td>Local Program Memory</td>
</tr>
<tr>
<td>MGT</td>
<td>Multi-Gigabit Transceiver</td>
</tr>
<tr>
<td>NGC</td>
<td>Native Generic Circuit</td>
</tr>
<tr>
<td>NGD</td>
<td>Native Generic Database</td>
</tr>
<tr>
<td>PAR</td>
<td>Place and Route</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>PLL</td>
<td>Phase-Locked Loop</td>
</tr>
<tr>
<td>RTT</td>
<td>Real Time Trace</td>
</tr>
<tr>
<td>RX</td>
<td>Receiver</td>
</tr>
<tr>
<td>SGMII</td>
<td>Serial Gigabit Media Independent Interface</td>
</tr>
<tr>
<td>SMA</td>
<td>SubMiniature version A</td>
</tr>
<tr>
<td>SW</td>
<td>SoftWare</td>
</tr>
<tr>
<td>TAP</td>
<td>Test Access Port</td>
</tr>
<tr>
<td>TB</td>
<td>Trace Buffer</td>
</tr>
<tr>
<td>TX</td>
<td>Transmitter</td>
</tr>
<tr>
<td>XST</td>
<td>Xilinx Synthesis Technology</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Chapter one provides an introduction to this thesis work. The background knowledge along with the motivation behind conducting this work is presented. Furthermore, an overview of the structure of the remaining portions of the report is described.

1.1 Background

1.1.1 A Brief Overview of EA System

Ericsson ASIC (EA) system is a highly integrated system with extreme complexity in the latest technologies. The generic EA architecture consists of a debug block with SERDES interface, a MEMORY, a few Accelerators and a number of DSP:s.

1.1.2 An Overview of Debugger System

The debugger system consists of the following units:

---

1 Developed for specific applications.
2 Here MEMORY stands for a proper noun specified by Ericsson. It should be distinguished from a common noun memory in this article.
CHAPTER 1. INTRODUCTION

- Hardware: EA system and Nexus trace probe.
- Software: DebugTool\(^3\).

The Nexus trace probe configures the TAP\(^4\) registers in debug block via JTAG interface and collects trace data from debug at 3.125Gbps through a high speed serial channel in four lanes. The probe communicates with DebugTool through Ethernet.

DebugTool is used for the development of application software running on DSPs. It contains all tools needed for building an application for an Ericsson ASIC.

Figure 1-2 demonstrates an overview of debugger system.

![Debugger System Overview](image)

**Figure 1-1: Debugger system overview**

### 1.1.3 FPGA-based Prototyping

FPGA-based prototypes are increasingly used in ASIC developments today to allow rapid design iteration and to enable early development of the surrounding system \([1]\). Compared to the expensive and long-term development of ASICs, FPGAs significantly improve demand of time-to-market and reduce the effort, cost and risk of hardware implementation. More importantly, FPGA-based

\(^3\) Debugging tools developed by Ericsson.

\(^4\) Test Access Port.
prototyping allows hardware/software co-design well ahead of chip fabrication and dramatically accelerates functional verification of ASIC and SoC designs. Additionally, high-performance FPGA-based prototyping system allows developers to perform real-world testing with real-world high-speed interfaces.

Considering all these benefits brought by FPGAs, a scheme that implements a micro-architecture of Ericsson ASIC (EA) on FPGA is proposed. This FPGA platform combined with probe and debug tools developed by Ericsson Digital ASIC Unit can be used to verify and validate the next generation of debugger system before ASIC arrives.

### 1.2 Thesis Goals

Primarily, this thesis aims at developing and evaluating a FPGA platform that can be applied in the development of next generation of the inhouse target debugger system. This FPGA platform can also be used for prototyping and verification acceleration of Ericsson ASIC IP:s.

In more specific terms, the following objectives are scheduled to gain in this thesis work:

1) DEBUG prototyping and verification acceleration.
   - Millions of trace messages can be sent to the probe in a few seconds.
   - Hands on verification from a user perspective directly via Debug-tools.
   - Platform to concept prove new ideas with tools and real data.

2) Verification of serial interfaces.
   - Serial interface between probe and debug can be tested at 3.125Gbps, which is not possible in palladium today.

3) Probe development platform.
   - The probe will be ready and bug free when ASIC arrives.
4) Debug tool development platform.
   • Debug tools can get feedback faster when a complete system with SERDES support is available before the ASIC arrives.

5) DSP verification acceleration.
   • Special instructions that take days with directed tests in SIM can be verified in seconds [3].

1.3 Thesis Structure

The rest of the thesis report is organized as follows. Chapter two gives an overview of μ-EA system and working environment including operating principles, hardware and design tools. Following that, chapter three presents the implementation procedure of targeting μ-EA system on FPGA from adding each design block in sequence to synthesis and implementation processes with the help of Xilinx ISE software. Chapter four presents SERDES test procedure along with the experiment results from oscilloscope and Chipscope. Chapter five presents running software execution on DSP with trace messages successfully sent out by DebugTool. The report concludes with a discussion in chapter six about the achievements of this work and the future work.
Chapter 2

An Overview of Micro System

This chapter provides the basics of µ-EA system and the working environment for this thesis. It aims at offering the reader a general picture of µ-EA system and the initial working environment.

2.1 Architectural Overview

In this thesis, a µ-EA design is implemented in Xilinx ML605 FPGA board. µ-EA is a minor part of EA. It mainly consists of a debug block, a minor MEMORY and one DSP. The main feature which differs it from EA is that µ-EA contains only one DSP and no accelerators. In addition, the system frequency decreases to 2.5MHz. This new mini EA system is named as Athena.

The generic architecture of the µ-EA system is illustrated in Figure 2-1. The initial design consists of the following blocks:

- 1 debug block.
- MEMORY with a few ports.
- 1 or 2 DSP:s.
- BUS.
• Parts of MISC-block.

2.1.1 Debug

The debug block provides a mean to enable both traditional static target debugging and non intrusive real time target debugging while the system is up and running [2]. It can be controlled through JTAG or BUS-interface. The trace information can be routed to MEMORY or sent out on a 3.125Gbps serial interface [2].

2.1.2 MEMORY Controller

The MEMORY Controller is a block that enables sharing of memory resources between several units (accelerators, DSPs and interface blocks) in types of read and write. It always has a MEMORY and it might also have a separate semaphore memory. The MEMORY Controller can also be designed to support access to an external memory. There is an interface used for DSP and other units that are connected to the MEMORY to transfer data to or from the MEMORY.
2.1.3 DSP

DSP is applied in Ericsson ASIC architecture based platforms, setting a new standard in performance with a slew of micro-architectural enhancements. A dynamic real time trace debugging feature has been implemented in the DSP core, improving the debugging capability of the Ericsson ASIC software. The trace messages generated by DSP will be collected by Nexus compliant debug tools [19].

The DSP core mainly has the following units:

- 1 Program control unit (PCU) with local program memory (LPM)
- 1 Data address arithmetic unit (DAAU) with three AGU:s and local data memory (LDM)
- 1 MEMORY interface. It also handles program/data transfers to/from MEMORY.

The local program memory (LPM) and local data memory (LDM) can be accessed through MEMORY interface to transfer program or data to/from MEMORY.

2.1.4 BUS

BUS mails are used for communication between DSP:s, the CPU, accelerators and interfaces. Messages on the BUS are called Mail, including 24 bit header and 104 bit payload. The messages are passed in a ring that includes all DSP:s and other units in the system.

2.1.5 MISC

MISC block makes debug block possible to control Reset, Boot, Stop and CoreDump of DSPs.
2.1.6 Clock Generation

The ML605 has three FPGA fabric clock sources available on board\(^5\). There is one 2.5V LVDS differential 200 MHz oscillator (U11) soldered onto the board and wired to an FPGA global clock input. The 200 MHz signal names are SYSCLK\_N and SYSCLK\_P, which are used to provide the global clock source.

There is a clock unit generated by Clocking Wizard to provide a 5MHz clock by dividing the input 200 MHz clock. It was meant to take this 5MHz clock as the system clock for all units in \(\mu\)-EA system at first. However, to ease the heavy logic burden and get Place & Route process completed, the system frequency has to switch to 2.5MHz ultimately. This is done by adding a clock divider with input of 5MHz in the design, since 5MHz is the lowest frequency that can be generated by Clocking Wizard.

2.2 Working Environment Overview

The debugger system includes \(\mu\)-EA, trace probe and DebugTool, as shown in Figure 2-2.

\(^5\) Refer to Table 1-7 in *ML605 Hardware User Guide* [9].
2.2.1 Operating principle

In this thesis, DebugTool is mainly used for verification of debug features in DSP. Actually, DebugTools define a complete tool set for the development of application software running on Ericsson ASIC DSPs.

The trace probe supplies an interface for DebugTool to probe the debug features of µ-EA. It communicates with DebugTool through Ethernet. There exists a built-in JTAG controller in the trace probe for target debug and trace setup. The trace probe also collects trace data from DSP at 3.125Gbps through a high speed serial channel in four lanes.

DSP is able to run some software execution via DebugTool, handling trace messages transferred to MEMORY or probe.

The user can use DebugTool through JTAG-Probe to do the following operations on the µ-EA:

- Reset, Boot, Stop, and CoreDump of DSP:s.
CHAPTER 2. AN OVERVIEW OF MICRO SYSTEM

- Access DSP internal registers and memories.
- Set up break-points, etc.
- Access the MEMORY.
- Access the BUS.
- Set up RTT and the serial GTX port.
- Collect trace messages.

2.2.2 Hardware

Xilinx Virtex-6 FPGA ML605 board is selected as base platform to develop the µ-EA system because of its high performance, serial connectivity, and large logic size. It is based on the XC6VLX240T-1FFG1156 Virtex-6 FPGA. This FPGA contains 241,152 logic cells, a rating that reflects the increased logic capacity offered by the 6-input LUT architecture [18].

The trace probe supplies a JTAG interface and Nexus serial interface for DebugTool to enable the debug features of µ-EA. Before Nexus probe 2.0 arrived, we were working on the old probe, which was developed on Xilinx Virtex-5 FPGA ML507 board. The Nexus probe 2.0 is developed based on Xilinx Virtex-5 FPGA ML507 board by Ericsson.

SamTec adapter board and cable are customized to connect ML605 to Nexus probe 2.0 for serial interface testing.

2.2.3 Environment and design tools

The working environment is based on UNIX and Windows Vista.

First, we start from working on GUI (Graphical User Interface). The starting design tools are Xilinx ISE Design Suite 13.1, including ISE Project Navigator, CORE Generator, PlanAhead, iMPACT, ChipScope Pro and so on.

After the design is successfully implemented on FPGA, DebugTool is fully utilized to control and debug the µ-EA system.
In the end, the synthesis and place&route procedures should be transferred from GUI to scripts (shell, TCL, etc.), to make it easily understandable and executable for ASIC or FPGA designers to utilize.

Some experiment equipments like oscilloscope are also used to assist the work.

**Overview of ISE software**

Xilinx ISE is a software tool produced by Xilinx for synthesis and analysis of HDL designs, which enables the developer to synthesize their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimulus, and configure the target device with the programmer [17].

The ISE software controls all aspects of the design flow. The Project Navigator is an interface for developers to access all of the design entry and design implementation tools, and also access the files and documents associated with the project [12].

For information on how to use ISE software, refer to *ISE In-Depth Tutorial* [12].

**Overview of DebugTool**

Ericsson DebugTools package provides a complete tool set for the development of application software running on Ericsson ASIC DSPs. This includes the entire design flow, code generation, software verification, system level test and finally test and trouble shooting using the actual hardware [15]. The mainly used DebugTools in this thesis work are listed as below:

- **Flasm**: Ericsson ASIC Assembler. It translates assembly source code into binary machine code and outputs an object file.

- **Flink**: DebugTools linker. It links one or more object files into executables and shared libraries.

- **JDB**: Low level IR/DR debugger. It reaches each register through straight forward *ir* and *dr* commands.
• Flunk: Graphical debugger.

For information on how to setup and use DebugTool, refer to the FPGA Debug User Guide [20].

**Overview of Scripts**

The scripts are text files containing the desired commands and options that can be run by EDA tools. Scripting doesn’t require as much tool knowledge and can make compiling design easier and faster. Advanced scripting makes designers able to:

• Run a compile multiple times using different options.
• Write to different directories.
• Run other command line tools [14].

The synthesis and PAR processes are controlled by the scripts with the tool Xilinx ISE software. The tools are called in the following sequence: synthesis tool (XST), NGDBuild, MAP, Place-and-Route (PAR), TRACE (optional) and BitGen. The scripts contain all the commands and options for synthesis and PAR and are customized to the corresponding design.

For information on how to create and run scripts, refer to FPGA Debug User Guide [20].

**2.2.4 Language**

μ-EA system is developed using VHDL. Additionally, a small program written in assembly language is used for software execution.
Chapter 3

Implementation of μ-EA on FPGA

This chapter discusses the procedure to target μ-EA system on FPGA board, using Xilinx ISE tools. It begins by targeting debug block on FPGA and continues by adding MEMORY, DSP and a minor MISC block to get a complete μ-EA system. Furthermore, parts of synthesis and PAR results as well as static timing analysis are also presented. In the end, a brief summary is addressed.

3.1 Targeting Debug on FPGA

The original debug block\(^6\) is trimmed to support two DSPs and has no SERDES block, providing more convenience to start with this block. This section first presents the modifications made to debug core, then focuses on implementation of SERDES module inside debug, utilizing GTX transceiver of Virtex-6 FPGA, and ends up on discussing setting up debugging environment and testing access of trace buffer.

---

\(^6\) There are different types of EA system and this debug block is one of them.
3.1.1 Modifications on Debug Core

To make debug block fit FPGA design, a few modifications have been applied, including recreating some of the memories and flipflops.

3.1.1.1 Regenerating Memory

Some memories in this original ASIC design cannot fit in FPGA. Thus they are recreated by the Xilinx Block Memory Generator using embedded block RAM resources in Xilinx FPGAs.

3.1.1.2 Recreating flipflops

The meta-flipflop which is aimed at reducing meta-stability is replaced with RTL design implemented by double flipflops. The reset pipe stage for the purpose of delaying reset signal in one clock cycle is also recreated, realized as one flipflop.

3.1.2 Adding GTX in Debug Top Level

The Virtex-6 FPGA GTX transceiver is selected to provide a high-speed serial link between debug and probe. In order to effectively customize GTX for this specific design, an effort has been made to dig into details of GTX functional blocks and working principles. A brief introduction to GTX Transceiver is addressed at the beginning, followed with a detailed discussion regarding settings of GTX.

3.1.2.1 GTX Transceiver and Tool Overview

Overview

The Virtex-6 FPGA GTX transceiver is a power-efficient transceiver. It is highly configurable and tightly integrated with the programmable logic resources of the FPGA [4].
Virtex-6 FPGA GTX Transceiver Wizard

The Virtex-6 FPGA GTX Transceiver Wizard is a kind of CORE Generator tool. It is preferred to perform custom configuration of GTX transceiver and generate a wrapper to instantiate a GTX transceiver primitive called GTXE1.

Transmitter

Each GTX transceiver contains an independent transmitter and receiver. The transmitter transfers the parallel data to the high-speed serial data, while the receiver does the opposite operation. Applications transmit data through the GTX transceiver by writing data to the TXDATA port on the positive edge of the parallel clock TXUSRCLK2, which is determined by the TX line rate, the width of the TXDATA port, and whether or not 8B/10B encoding is enabled. The width of the TXDATA port can be configured to be one, two, or four bytes wide. In this design, only TX is utilized to transmit data from debug to probe through a high speed serial link.

Shared Transceiver Features

The dynamic reconfiguration port (DRP) allows the dynamic change of parameters of the GTXE1 primitive, which is an alternative to configure 3.125 Gbps line rate. An effort has been made to study on DRP in order to find the possibility of reaching 3.125 Gbps line rate. The DRP interface possesses an address bus (DADDR) and separated data buses for reading (DRPDO) and writing (DI) configuration data to the GTXE1 primitive. There are also some control signals that implement read and write operations, indicate operation completion, or indicate the availability of data, including an enable signal (DEN), a read/write signal (DWE), and a ready/valid signal (DRDY). Refer to the Virtex-6 FPGA Configuration User Guide [11] for detailed descriptions and timing diagrams of the DRP operations. Refer to [16] and DRP Address Map of the GTX Transceiver in [4] for a DRP map of the GTX transceiver attributes.
CHAPTER 3. IMPLEMENTATION OF μ-EA ON FPGA

3.1.2.2 GTX Configurations

This section shows how to configure and use the functional blocks inside the GTX transmitter. Table 3-1 describes the settings for GTX TX. For detailed definitions of GTX TX elements, refer to [4]

<table>
<thead>
<tr>
<th>Options</th>
<th>Settings</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol Template</td>
<td>Aurora 2 byte</td>
<td>Aurora 2 byte Multilane protocol is preferred for GTX. There are 4 high speed</td>
</tr>
<tr>
<td></td>
<td>Multilane</td>
<td>serial lanes to transmit data.</td>
</tr>
<tr>
<td>Initial Line Rate</td>
<td>2.5Gbps</td>
<td>The default line rate is set to 2.5Gbps. Line rate 1.25 Gbps can be configured</td>
</tr>
<tr>
<td></td>
<td></td>
<td>by setting the TAP register TXRATE. And Dynamic Reconfigurable Ports is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>preferred to make line rate of 3.25Gbps possible. The details of line rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>configuration are demonstrated in Section 1.2.3, Line rate configuration.</td>
</tr>
<tr>
<td>Reference CLK</td>
<td>125MHz</td>
<td>A high-quality, low-jitter, 125-MHz LVDS clock is generated from an internal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>crystal oscillator of FPGA. This differential clock is used as the reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk of GTX. It should be assigned to the pin locations H5(refclkn) and H6(refclkp)</td>
</tr>
<tr>
<td>8B/10B encoder</td>
<td>Bypass</td>
<td>8B/10B encoder is already included in Aurora core inside the debug block, so</td>
</tr>
<tr>
<td></td>
<td></td>
<td>that data stream is passed to transmission with no conversion.</td>
</tr>
<tr>
<td>TXDATA (Data Path Width)</td>
<td>20 bits</td>
<td>The width of the port is configured to be two bytes wide. Since the 8B/10B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>encoder is bypassed, the FPGA interface width is configured to 20 bits.</td>
</tr>
<tr>
<td>Use Dynamic Reconfiguration Port</td>
<td>On</td>
<td>Select this option to have the dynamic reconfiguration port signals available</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to the application. This feature is reserved for further configuration of 3.125Gbps line rate.</td>
</tr>
</tbody>
</table>

Table 3-1: Configuration of GTX TX
The FPGA clocking resources are used to drive the parallel clocks for GTX TX interface, which is demonstrated in Figure 3-1. TXOUTCLK\(^7\) is used to drive multiple GTX user clocks for 2-byte mode\(^8\).

Figure 3-1: TXOUTCLK Drives TXUSRCLK2 (2-Byte Mode) [4]

### 3.1.2.3 Line rate configuration

This section provides basic knowledge needed to understand on line rate configuration and presents how to get different line rates by setting PLL divider attributes and values.

---

\(^7\) TXOUTCLK is derived from MGTREFCLK0 [P/N] or MGTREFCLK1 [P/N].

\(^8\) TX_DATA_WIDTH = 16 or 20 depends on whether 8B/10B encoder is enabled.
CHAPTER 3. IMPLEMENTATION OF μ-EA ON FPGA

The line rate of 1.25Gbps and 2.5Gbps can be achieved by setting TXRATE of TAP register\(^9\) via DebugTool-probe. Refer to Appendix A: Application of Tap Registers for the definitions of GTX TAP registers. However, we need to find an alternative method in order to reach 3.125 Gbps line rate, which is unachievable through TXRATE register. Thus the dynamic reconfiguration port is considered to reach this target, which will be carried out as a trial in the future.

Each GTX transceiver contains one TX PLL and one RX PLL. The PLL output has a divider that can divide the output frequency by one, two, or four. The PLL output frequency is determined by a factor of \(M\) and the feedback dividers, \(N_1\) and \(N_2\), as shown in Equation 3-1.

\[
 f_{PLLCLKout} = f_{PLLCLKin} \times \frac{N_1 \times N_2}{M} \quad \text{Equation 3-1}
\]

Equation 3-2 shows how to determine the line rate (Gb/s). \(D\) is the PLL output divider that resides in the clock divider block [4].

\[
 f_{LineRate} = \frac{f_{PLLCLKin} \times 2}{D} \quad \text{Equation 3-2}
\]

Table 3-2 lists the actual attribute and commonly used divider values [4].

<table>
<thead>
<tr>
<th>Factor</th>
<th>Attribute Name</th>
<th>Valid Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>(M)</td>
<td>TXPLL_DIVSEL_REF</td>
<td>1, 2</td>
</tr>
<tr>
<td>N1</td>
<td>TXPLL_DIVSEL45_FB</td>
<td>4, 5</td>
</tr>
<tr>
<td>N2</td>
<td>TXPLL_DIVSEL_FB</td>
<td>2, 4, 5</td>
</tr>
<tr>
<td>D</td>
<td>TXPLL_DIVSEL_OUT</td>
<td>1, 2, 4</td>
</tr>
</tbody>
</table>

Table 3-2: PLL Divider Attribute and Common Values

\(^9\) TAP data registers can be used to configure GTX, debug block, etc.
The initial line rate is set to 2.5Gbps. The D divider value can be changed dynamically for multiple line rates through TXRATE port. Table 3-3 shows PLL divider settings for different line rates.

<table>
<thead>
<tr>
<th>Standard</th>
<th>Line Rate (Gb/s)</th>
<th>Internal Data Width</th>
<th>PLL Frequency (GHz)</th>
<th>REFCLK (MHz)</th>
<th>N1</th>
<th>N2</th>
<th>D</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aurora</td>
<td>2.5</td>
<td>20b</td>
<td>1.25</td>
<td>125</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.25</td>
<td>20b</td>
<td>1.25</td>
<td>125</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3-3: PLL Divider Settings for Aurora Protocol

The control for the serial divider is described in Table 3-4. For details about the usage of TXRATE port, refer to Appendix A: Application of TAP registers.

<table>
<thead>
<tr>
<th>Line Rate Range (Gb/s)</th>
<th>D Divider Value</th>
<th>Static Setting via Attribute</th>
<th>Dynamic Control via Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.40 to 6.60</td>
<td>1</td>
<td>TXPLL_DIVSEL_OUT = 1</td>
<td>TXRATE = 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TXRATE = 00</td>
<td></td>
</tr>
<tr>
<td>1.20 to 3.3</td>
<td>2</td>
<td>TXPLL_DIVSEL_OUT = 2</td>
<td>TXRATE = 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TXRATE = 00</td>
<td></td>
</tr>
<tr>
<td>0.60 to 1.65</td>
<td>4</td>
<td>TXPLL_DIVSEL_OUT = 4</td>
<td>TXRATE = 01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TXRATE = 00</td>
<td></td>
</tr>
</tbody>
</table>

Table 3-4: TX PLL Output Divider Setting

3.1.3 Setting up Debugging Environment

This part of work is based on the old probe, which implements only one lane for serial trace data transmission.

Connection with old probe

The ML605 board possesses a J41 2x7 header below LCD, which is used as JTAG connector. A self-made cable is used to connect JTAG ports of ML605 board to the old probe.
The serial channel is connected via an optical cable.

**Connection with latest probe**

There is an onboard high-speed VITA-57 FMC\textsuperscript{10} HPC\textsuperscript{11} expansion connector on Xilinx ML605 FPGA board, which is used to provide Nexus data transmission port and JTAG port through a Samtec adapter. The Samtec connector is used to connect the FMC HPC connector on ML605 board to the probe that implements the Nexus serial interface. It contains 2 Nexus data receiving ports, 2 Nexus data transmitting ports and 2 separate JTAG ports. Refer to Appendix B: FMC Connection Pinout.

At beginning, we scheduled to verify the serial link connected between ML605 board and latest probe via Samtec adapter and cable as well. However, the Samtec adapter has not arrived yet when finalizing the thesis, so this part of work is left to further work in the future.

**Setting up DebugTool**

When you get back correct ID code from debug block in JDB or Flunk terminal, it indicates that JTAG connection and the DebugTool are successfully setup. For instructions on how to setup DebugTool, refer to *FPGA Debug User Guide* [20].

### 3.1.4 Accessing Trace Buffer

Accessing trace buffer can be accomplished either via JDB or Flunk. The procedure to transfer data to or from TB using JDB is presented as below:

- Load \textit{tbdata} register, which stores data transferred to or from TB memory and clear register value to all zeros.

- Load \textit{tbaddr} register, which contains TB address control information: read or write, increase address or not, address. Set it to the write mode and define the

\textsuperscript{10} FPGA Mezzanine Connector.

\textsuperscript{11} High Pin Count.
beginning address as well as whether to increase address after every write or read operation.

- Load *tbdata* register again and assign some 128 bits data value.
- Then load *tbaddr* register and set as read mode.
- Load *tbdata* register to view the written data and check if they are valid.

See [16] for more details about above TAP registers.

The procedure to access TB using Flunk is much easier compared to JDB. For details regarding Flunk operation, refer to *FPGA Debug User Guide* [20].

Figure 3-2 shows the results of accessing TB. We write 8 words into trace buffer from address 0 and then read out data from address 0 to 0x0f.

```
athanal1/dsp0> radix 16
athanal1/dsp0> tb[0]=1 2 3 4 5 6 7 8
athanal1/dsp0> dump tb 0 0x0f
0x000e: 0x0001 0x0002 0x0003 0x0004 0x0005 0x0006 0x0007 0x0008
0x0008: 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
athanal1/dsp0> |
```

Figure 3-2: Results of accessing TB

In brief, it is proved that accessing TB can be successfully achieved.
3.2 Adding MEMORY

Figure 3-3 illustrates the MEMORY interface with DSP/Units.

There are 4 RAM banks residing in the MEMORY. The depth of each RAM is reduced to 4096 bits to make it not cost beyond the RAM resources in Xilinx FPGA. They are also recreated by Xilinx Block Memory Generator. The configuration of the RAM is the most critical factor that will affect the further read and write operations of MEMORY. Thus a proper setting of RAM block plays an extremely important role in the whole system.
3.2.1 Configuration of RAM

Memory Types

The Block Memory Generator core uses embedded block RAM to generate five types of memories. Single-port RAM is selected for RAM in MEMORY, which allows Read and Write access to the memory through a single port, as shown in Figure 3-4.

![Figure 3-4: Single-port RAM [5]](image)

Memory Algorithm

Proper selection of memory algorithm will affect the write and read performance of MEMORY seriously. For example, in this design, the Minimum Area Algorithm will cause RAM output data located not only in specific written address but also in other addresses. As a result, the Fixed Primitive Algorithm with size of 1kx18 is chosen.

The fixed primitive algorithm allows the user to select a single block RAM primitive type. The core will build the memory by concatenating this single primitive type in width and depth [5].

Operating Mode

Write first mode is selected. In write first mode, the input data is simultaneously written into memory and driven on the data output.

---

12 Refer to [5] for descriptions of five types of memories.
CHAPTER 3. IMPLEMENTATION OF µ-EA ON FPGA

Output Registers

There are two optional output register stages at two places: at the output of the block RAM primitives and at the output of the core. In this design, adding output registers will cause output data delay one clock cycle.

Byte-Write Enable

8-bit byte write enable port is selected.

Enable Pin

Enable pin is chosen to control the operation of the memory. When it is deasserted, no Read, Write, or reset operations are performed on the respective port.

Figure 3-5 presents the IP Symbol of the generated RAM.

![Figure 3-5: RAM IP Symbol](image)

3.2.2 Accessing MEMORY

Accessing MEMORY is one of the most difficulty parts and consumes a great deal of time and efforts. In this section, instead of guiding users on how to operate, we focus on the rough process about how to get it work.
Access of MEMORY is done via DebugTool-probe. Refer to *FPGA Debug User Guide* [20] for more information on operation.

At the beginning, there appear some strange behaviors when reading data from MEMORY. They are as bellow:

- Output data delays one clock cycle. The output data will not show up in the written address until the second read operation.
- Output data appears in unwritten addresses. The output data is not only placed in the exact written address, but also jumps to other address lines.

Since it is a quite large and complex design, including many levels, it is suggested to start observation from the design module at the bottom. To figure out the issue, we take advantage of Chipscope to view internal signal transitions on MEMORY interface and ports of RAM wrapper, which is in the deepest block hierarchy.

As triggering conditional ports and analyzing signal behaviors in waveform result, the following two abnormal behaviors are observed:

- *addra* port of RAM wrapper did not show correct address value.
- By watching on memory contents during write and read operations, it is possible to figure out in which part is the problem existed. When doing write operation, it is observed that the data is written into correct address and no other data appears in other memory locations. However, there comes out some data in every other address row after reading procedure. It is obvious that some uncertainties cause the output data errors.

To solve the issue about *addra*, it is of essence to provide some basic information about MEMORY address specification first. For more details about MEMORY interface protocol, refer to [10].
As can be seen from Figure 3-2 MEMORY Interface, when there is a request for a new access to MEMORY from other units, a command word is sent on the \textit{wr_data} bus concurrently. When accessing the MEMORY, the command word bit[31:0] specifies address in MEMORY address where data is transferred to or from. It is illustrated in Figure 3-6, Address Definition.

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{Figure3-6.png}
\caption{Address Definition}
\end{figure}

Bit[2:0] – Decide in which slot the data is located among 8 slots in the specific bank and row.

Bit[4:3] – Select in which bank data is placed. The width is flexible, which is determined by number of banks.

Bit[17:5] – Define the row where data is transferred. The width is flexible, which is determined by the memory depth.


The \textit{addra} port of RAM wrapper composes Bit[17:5] of command word. According to observation on Chipscope waveform result, this port did not function correctly. The reason lies in different bank selection. This type of MEMORY is designed for 4 banks and needs 2 bits for bank selection. If the bank number is reduced to 2, there will be only 1 bit used for bank selection, thus \textit{addra} is assigned the wrong value. For MEMORY of \(\mu\)-EA, there are only two banks at the beginning for the purpose of trimming design logics and saving FPGA device resource. That is where the problem arises. As a result, the solution is to increase bank number to 4 corresponding to specific requirements for this kind of MEMORY.
The second issue is related to read process of RAM wrapper. By digging into the RAM instantiations, we noticed that it is the following two settings that affect the memory’s functionality:

- **Selective Memory Algorithm.** The Block Memory Generator core arranges block RAM primitives according to one of three algorithms: the minimum area algorithm, the low power algorithm and the fixed primitive algorithm [5]. At first, ‘Minimum area algorithm’ was selected. It is this algorithm that makes the output data appear in different memory locations. To work around the problem, change the algorithm to 'fixed primitive' with size of 1kx18.

- **Optional Output Registers.** The user may choose to include register stages at two places: at the output of the block RAM primitives and at the output of the core [5]. It is those output registers that cause the output data delay. When we first read from MEMORY, it outputs the previous data stored in registers. Thus we need to do another read operation to get the current written data. The solution is to remove those output registers.

In summary, we ultimately manage to access MEMORY via DebugTool and probe after many tough trials assisted with the tool Chipscope. The experience obtained in this part of work can be concluded as follows:

- See through the appearance to the essence and think from the most basic parts to resolve the problem when there is an immense design with quite complex structure. Otherwise it is easy to get lost.

- Take proper tools as an assistant to detect the problem, which will provide you a quite effective way to find out the issue.
3.3 Adding DSP

The DSP core is taken from Holly\textsuperscript{13}, and is renamed Atolly corresponding to Athena core. It has the internal memories trimmed and some units removed for the purpose of lightening overwhelming design logics which cost a great deal of FPGA device resource.

Initially, it was planned to implement design with two DSPs on FPGA. But unexpectedly that the design with two DSPs utilizes 139\% of FPGA resources after synthesis, making followed Place & Route incomplete. A great many means were tried to prune down the size of the design in order to fit it in FPGA, as shown in the followings:

- Slow down the system frequency to 5MHz to ease the pressure of synthesis problem.
- Reduce MEMORY size. The depth of each RAM bank is reduced to 4096 bits.
- Apply clock constraints during synthesis process.
- Set optimal synthesis strategy. Change the design goals and design strategy to Area reduction and Maximum LUT reduction respectively. For details about settings of process properties for synthesis and implementation, refer to the shell script including synthesis.sh, top.tcl and PAR.sh, which are introduced in \textit{FPGA Debug User Guide} \textsuperscript{20}.
- Remove two computational units and two timers of DSP to keep the design more compact, which will not affect the performance of DSP.

However, after all the efforts, the device resource consumption only decreases slightly, from 139\% to 135\%. DSP consumes most of the device resources. As a result, it was finally decided to pick an alternative solution that is removing one DSP from the original design. This new design with only one DSP occupies 70\% FPGA resources after synthesis.

\textsuperscript{13} Holly is a type of DSP core.
A successful access of DSP involves the following efforts:

- Have DebugTool setup and ready for Athena core.

- Regenerate and prune memories in LDM and LPM to get those memories accessible and fit FPGA. The memories in LDM and LPM are recreated by Xilinx CORE Generator since the original memories are too big. The size of memory in LDM is reduced to 8x16, and in LPM is reduced to 512x64.

See Chapter 5.2.2, which presents a more detailed discussion on getting DSP work in FPGA.

In conclusion, the design including a debug, a minor MEMORY and one DSP is synthesized on FPGA with both DSP memories and registers accessible. The question of seeking proper solution to fix synthesis difficulty is at the center of this part of work. And an alternative solution using only one DSP in µ-EA system is taken ultimately, which reduces the usage of FPGA resources to a greatest degree. Another issue is concerning failed access of DSP memory at first, which forced us to make a decision to regenerate RAM in LDM and LPM. This measure caused Place & Route phase incomplete and we have to take steps to trim the memory size to ease the heavy logics.

### 3.4 Adding MISC block

In this thesis work, a minor part of misc block is implemented, which makes debug block possible to control Reset, Boot, Stop and CoreDump of DSPs. The simplified misc block interface is shown in Figure 3-7.
CHAPTER 3. IMPLEMENTATION OF μ-EA ON FPGA

The input data port determines which DSP to be enabled. It sends out a pulse to one of the four outputs: RESET, DIRQ, SIRQ, and BIRQ, which are selected by sel port, to enable one of those functions. Bit 4 of sel which stands for ACCELERATOR RESET is unused.

The following TAP registers are used to control input of misc block and operation of DSP:

- resetIrqR: Control data[1:0].
- resetIrqSelR: Control sel[4:0].

For definition and usage of above registers, refer to [16].

One thing needs to mention is that because of this trimmed misc block that does not have override function, debug tool Flunk can’t boot DSP at first. Some efforts are made to get Flunk work for Athena core as well with help from the DebugTool designer.

3.5 Clock Configuration

The system clock 2.5MHz supplied for all the units is derived from differential 200MHz oscillator through a clock unit generated by Clocking Wizard and a clock divider. The 200 MHz signals are SYSCLK_N and SYSCLK_P, which are
assigned to the pin H9 and J9 in user constraint file. The block diagram of clock generation is illustrated in Figure 3-8.

![Block Diagram of Clock Generation](image)

Figure 3-8: Block Diagram of Clock Generation

### 3.6 Synthesis and Implementation Results

Making μ-EA system fit in FPGA is what we have fought toward throughout the whole thesis work. It is extremely meaningful to view the final synthesis and PAR results especially on device utilization, which prompts us to seek various means to release. Before digging into more details on that, it is helpful to briefly review FPGA design flow.

The following synthesis and PAR results and post static timing analysis presented are achieved by performing Xilinx ISE tools. The results obtained from running in batch mode by Synplify_Pro are slightly different, for different FPGA vendors provide a slight margin for their synthesis and timing results.

#### 3.6.1 FPGA Design Flow

In general, FPGA design flow contains the following five phases:
CHAPTER 3. IMPLEMENTATION OF µ-EA ON FPGA

Figure 3-9: FPGA Design Flow

**Design Entry**

There are different techniques for design entry: schematic based, Hardware Description Language and combination of both. Hardware Description Language is used in this thesis work.

**Synthesis**

This process translates VHDL or Verilog code into a device netlist format, which is saved to an NGC\textsuperscript{14} file.

**Implementation**

This process consists of a sequence of three steps:

- **Translate** process combines all the input netlists and constraints to a logic design file, which is saved as an NGD (Native Generic Database) file. This can be done using NGD Build program. A file named UCF (User Constraints File) defines constraints, assigning the ports in the design to the physical elements of the targeted device and specifying time requirements of the design.

\textsuperscript{14} Native Generic Circuit file. It is produced by Xilinx® Synthesis Technology (XST).
• **Map** process divides the whole circuit with logical elements into sub blocks, making them possible to fit the FPGA logic blocks.

• **Place and Route** process places the sub blocks from the map process into logic blocks according to the constraints and connects the logic blocks, using PAR program. It outputs NCD file which consists the routing information.

**Device Programming**

Load design on FPGA. The design is converted to a bit stream (a .bit file) which can be used to configure the target FPGA device by BITGEN program.

**Design Verification**

Verification can be done at different stages of the process steps:

• Behavioral Simulation (RTL Simulation).

• Functional simulation (Post Translate Simulation).

• Static Timing Analysis.

Static Timing Analysis is performed in this thesis work, which can be done after MAP or PAR processes. Post MAP timing report lists signal path delays of the design derived from the design logic. Post Place and Route timing report incorporates timing delay information to provide a comprehensive timing summary of the design.

### 3.6.2 Synthesis Result

Part of the synthesis report is invoked here to provide a general picture of device utilization of the design after synthesis process. Device utilization summary from synthesis process is illustrated in Table 3-5 as below.
<table>
<thead>
<tr>
<th>Device Utilization Summary</th>
<th>Number</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Slice Logic Utilization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slice Registers</td>
<td>24065 out of 301440</td>
<td>7%</td>
</tr>
<tr>
<td>Slice LUTs</td>
<td>105863 out of 150720</td>
<td>70%</td>
</tr>
<tr>
<td>used as Logic</td>
<td>105701 out of 150720</td>
<td>70%</td>
</tr>
<tr>
<td>used as Memory</td>
<td>162 out of 58400</td>
<td>0.2%</td>
</tr>
<tr>
<td>used as RAM</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td><strong>Slice Logic Distribution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LUT Flip Flop pairs</td>
<td>111011</td>
<td></td>
</tr>
<tr>
<td>with an unused Flip Flop</td>
<td>86946 out of 111011</td>
<td>78%</td>
</tr>
<tr>
<td>with an unused LUT</td>
<td>5148 out of 111011</td>
<td>4%</td>
</tr>
<tr>
<td>fully used LUT-FF pairs</td>
<td>18917 out of 111011</td>
<td>17%</td>
</tr>
<tr>
<td>unique control sets</td>
<td>777</td>
<td></td>
</tr>
<tr>
<td><strong>IO Utilization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOs</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>bonded IOBs</td>
<td>23 out of 600</td>
<td>3%</td>
</tr>
<tr>
<td><strong>Specific Feature Utilization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block RAM/FIFO</td>
<td>120 out of 416</td>
<td>28%</td>
</tr>
<tr>
<td>using Block RAM only</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>BUFG/BUFGCTRLs</td>
<td>9 out of 32</td>
<td>28%</td>
</tr>
<tr>
<td>DSP48E1s</td>
<td>214 out of 768</td>
<td>27%</td>
</tr>
</tbody>
</table>

Table 3-5: Device utilization summary of synthesis

A LUT Flip Flop pair for this architecture represents one LUT paired with one Flip Flop within a slice. A control set is a unique combination of clock, reset, set, and enable signals for a registered element.
3.6.3 Place & Route Result

Part of the PAR report is invoked here to provide a general picture of device utilization of the design after PAR process, which owns a slight difference from that of synthesis report. Device utilization summary from PAR process is illustrated in Table 3-6 as below.

<table>
<thead>
<tr>
<th>Device Utilization Summary</th>
<th>Number</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Slice Logic Utilization</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slice Registers</td>
<td>24,392 out of 301,440</td>
<td>8%</td>
</tr>
<tr>
<td>used as Flip Flops</td>
<td>24,068</td>
<td></td>
</tr>
<tr>
<td>used as Latches</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>used as Latch-thrus</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>used as AND/OR logics</td>
<td>323</td>
<td></td>
</tr>
<tr>
<td><strong>Slice LUTs</strong></td>
<td>94,330 out of 150,720</td>
<td>62%</td>
</tr>
<tr>
<td>used as logic</td>
<td>94,061 out of 150,720</td>
<td>62%</td>
</tr>
<tr>
<td>used as Memory</td>
<td>118 out of 58,400</td>
<td>1%</td>
</tr>
<tr>
<td>used exclusively as route-thrus</td>
<td>151</td>
<td></td>
</tr>
<tr>
<td><strong>Slice Logic Distribution</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>occupied Slices</td>
<td>26,250 out of 37,680</td>
<td>69%</td>
</tr>
<tr>
<td>LUT Flip Flop pairs used</td>
<td>96,678</td>
<td></td>
</tr>
<tr>
<td>with an unused Flip Flop</td>
<td>73,454 out of 96,678</td>
<td>75%</td>
</tr>
</tbody>
</table>
with an unused LUT | 2,348 out of 96,678 | 2%
| fully used LUT-FF pairs | 20,876 out of 96,678 | 21%

Table 3-6: Device utilization summary of synthesis

### 3.6.4 Post PAR Static Timing Analysis Result

It is meaningful to execute timing analysis, during which timing violations can be observed and resolved. Setup and hold violations usually occur due to unconstrained or improperly constrained data and clock paths. Global constraints are not always adequate to ensure that all paths are constrained, for data and clock paths can cross domain boundaries. To eliminate setup or hold violations, the first step is to analyze the failed path using Timing Analyzer, then use Constrains Editor to create timing groups and timing constraints for the design in the UCF. It should be ensured that no more failing timing paths are addressed by the Timing Analyzer. Part of timing analysis report is cited below, indicating that all timing constraints have been met, all setup or hold violations have been eliminated, and the maximum frequency that can be achieved is 24.976MHz.

Derived Constraint Report is shown in Table 3-7. All constraints were met.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Period Requirement</th>
<th>Actual Period</th>
<th>Timing Errors</th>
<th>Paths Analyzed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Direct</td>
<td>Derivative</td>
<td>Direct</td>
</tr>
<tr>
<td>TS_SYSCLK_P</td>
<td>5.000ns</td>
<td>2.800ns</td>
<td>0.036ns</td>
<td>0</td>
</tr>
<tr>
<td>TS_clk_unit_i_clkout0</td>
<td>200.000ns</td>
<td>1.429ns</td>
<td>N/A</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3-7: Derived Constraints for TS_SYSCLK_P
3.7 Summary

Implementation of µ-EA on FPGA, which serves as a foundation, has a profound effect on the subsequent work. The highlights of this work are GTX configuration, access of MEMORY and synthesis with DSP. The greatest difficulties encountered were accessing MEMORY and completing synthesis on FPGA.

The process to get through synthesis and make design fit FPGA was filled with great difficulty, because the design is quite dense and congested, and contains too much logic, which leads to excessive consumption of FPGA device resources. A great effort has been made to cut the FPGA device resource cost. Savings are gained in optimizing synthesis strategy, slowing down clock frequency, trimming memories as well as removing some units. But those savings are modest, totaling around just 4 percent of device utilization. An alternative solution has been proposed, that is to remove one DSP from the design, considering it occupies majority of the resource. Using only one DSP in µ-EA system saves 65% device resource consumption, enough to make this design routable.

Finally, the whole µ-EA system with debug block, a minor MEMORY, one DSP and a minor misc block is successfully targeted on FPGA, while access of TB, MEMORY and DSP is accomplished. It costs 62% of FPGA resources at a low system frequency of 2.5MHz after Place & Route process.

In brief, the main experience obtained in this chapter is that the trade off between performance and feasibility of the µ-EA system is of essence to be taken into account by designers.
Chapter 4

SerDes Test

The serial link is used to send out trace messages to the Nexus probe. It needs to be set up correctly through a procedure before one can use it. The following sections describe the procedure to successfully set up a high speed serial link. This part of work first starts with designing an aurora receiver, then proceeds to use it to verify the functionality of GTX transceiver inside debug block, and validate the high speed serial link.

4.1 Designing Aurora receiver

4.1.1 Architecture Overview

Before starting to set up a high speed serial link of debug GTX, we need to develop a simplex aurora receiver in another ML605 FPGA board (board 1) first. It is because that the Nexus probe 2.0 with SamTec adapter board\textsuperscript{15} is not ready yet. The simplex aurora receiver should be verified before connecting to GTX TX.

\footnote{\textsuperscript{15}Used to connect debug platform with Nexus probe 2.0 through JTAG and serial interface.}
To verify the aurora receiver, a simplex aurora transmitter is also implemented in ML605 FPGA board (board 2). The TX outputs are connected with RX inputs through an optical cable from SMA connectors on FPGA board. Then probe on sent TXDATA (20 bits) and received RXDATA (20 bits) by Chipscope to check if they are the same. Figure 4-1 shows the channel connection between Aurora Transmitter and Aurora Receiver.

Figure 4-1: Aurora 8B/10B Simplex Channel Overview [8]

### 4.1.2 Implementation Flow

Both aurora simplex RX and TX cores are generated by Xilinx Core Generator software, using one lane at 2.5Gbps line rate. The GTXs are located to the GTXE_X0Y18 site on each board, for this Multi-Gigabit Transceiver (MGT) location and pins are connected to the MGT SMAs on the ML605. The reference clock for Aurora core comes from a physical device on the board, which is a 125 MHz differential clock, derived from Ethernet SGMII Clock, connected to the H6/H5 pins in UCF file. The TX and RX data width is set to 20bits.

---

16 SMA (SubMiniature version A) connectors are coaxial RF connectors.
A back channel with sideband initialization signals is used for TX and RX cores to communicate their initialization state. Messages will be delivered from the RX sideband initialization ports to the TX side to indicate which of the sideband initialization signals are asserted. The initialization ports are called ALIGNED, BONDED, VERIFY, and RESET; one set for the TX side with a TX_ prefix, and one set for the RX side with an RX_ prefix [7]. The bonded port is only used for multi-lane cores. It is important that the sideband ports of TX and RX should be connected to each other.

Aurora core uses active low reset signal. But ML605 board has active high reset button\(^\text{17}\). This logic is presented in `<component_name>_reset_logic.v[hd]` under example_design directory.

On the other hand, Xilinx ML605 provides 200 MHz differential clock which can be used for INIT_CLK of Aurora design.

After programming Aurora TX and RX to each FPGA board, connecting serial TX output pins and RX input pins together, and also sideband ports of TX and RX, it’s time to validate the serial link.

### 1. Channel Initialization

Initialization of an Aurora 8B/10B channel consists of three stages:

- **Lane Initialization**: This procedure is performed individually on each lane to activate the link and align the received data to the proper boundaries.

- **Channel Bonding**: This procedure bonds the individual lanes into a single data channel. Channel bonding is not required and will be bypassed if there is only a single link.

- **Channel Verification**: This procedure performs any alignment needed to map received data to the user interface and verifies the ability of the channel to transfer valid data [8].

---

\(^\text{17}\) This needs to pay attention during design.
When channel verification is completed, the Aurora 8B/10B interface can immediately start to transmit data.

The channel setup procedure is described in Figure 4-2.

![Figure 4-2: Channel Setup Procedure](image)

The result shows that RX_ALIGNED, RX_BONDED and RXVERIFY are asserted immediately after power on. It means that RX module has completed the above three stages: lane initialization, channel bonding and verification.

Then, the lane_up and channel_up signals of RX are asserted, indicating that the channel between TX and RX is successfully established and is ready to send and receive data.

Finally, an Aurora 8B/10B channel is ready for data transmission and reception.

2. **Data transmission and reception**

After channel initialization completes, TX will immediately start sending data to RX. The data received on RX side is probed by Chipscope. The result shows that the data received exactly matches what transmitter has sent.

In conclusion, the Aurora simplex receiver is working as expected.
4.2 Verifying debug GTX TX

This section describes the procedures and the efforts made to get 1-lane serial link work between GTX and Aurora simplex RX, which provides preparation for 4-lane high speed serial link with the coming probe.

4.2.1 Architecture Overview

Debug GTX TX outputs are connected with serial data inputs of Aurora simplex receiver, in order to test the serial interface of debug GTX. Because of the limitation of serial interface on ML605 board, one lane is implemented first, with 2.5Gbps serial speed, as groundwork to test trace functions of DSP. Figure 4-3 illustrates the channel connection between GTX TX and Aurora receiver.

![Diagram](image.png)

Figure 4-3: GTX TX and Aurora RX Channel Overview

For SerDes and Aurora configuration, the following TAP registers are involved:

- `gtxctrlR`
- `gtxdivselR`
- `auroraConfigR`
- `tracesetupR`
The configuration of above TAP registers is realized by DebugTool function JDB.

For details about \texttt{gtxctrlR} and \texttt{gtxdivselR}, see Appendix A: Application of TAP registers, Table A-1: TAP data registers to configure GTX. As for \texttt{auroraConfigR} and \texttt{tracesetupR}, refer to \textit{SW IWD for Trace and Debug in Hermes} [16].

4.2.2 Implementation Flow

1. Channel Initialization

1) Setup GTX first

GTX in debug block should be set up correctly before proceeding to the channel setup phase. GTX reset is done by setting \texttt{gtxctrlR} register through JTAG-probe. Remember to pull down reset signal after reset is done. When GTX reset is done, resetdone and PLL locked signals will go high, which can be checked by reading the value of gtxstatus.

Line rate is set by writing value into \texttt{gtxdivselR}. The relation between line rate and TXRATE is shown in Table 4-1.

<table>
<thead>
<tr>
<th>Line Rate (Gbps)</th>
<th>TXRATE</th>
<th>D divider</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>1.25</td>
<td>10</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4-1: Line rate settings

2) Channel setup

The Aurora core in debug block has no sideband ports, so that the channel initialization is controlled by JTAG manually. The user needs to carry out a handshaking procedure with the receiver.

There is a tap register called \texttt{auroraConfigR}, which is used to deal with the init, bond, verify and data transmission procedures.
First, we should enable Aurora interface by setting bit 4 in \textit{auroraConfigR}. The number of lanes is also configured through \textit{auroraConfigR} bit 2.

After that, we are ready to go through the channel initialization procedures. Set init state in \textit{auroraConfigR}, then an init training sequence will be sent to RX from TX through serial link. When RX receives the init sequence and recognizes it, it will respond with align signal asserted. Then the user can step to the next phase: write bond state in \textit{auroraConfigR}, if there are multiple lanes. Otherwise, the bond phase is bypassed and the user can proceed to verify phase.

After init, bond and verify phases are successfully completed, and align, bond and verify signals of RX are asserted, the user can set data mode in \textit{auroraConfigR}, so that the channel is ready to transmit data. Initially, Aurora will send idle sequence on the channel.

2. **Data transmission and reception**

During data operation, the following three types of data are transmitted over an Aurora 8B/10B channel:

- **Idle sequence**: It is transmitted whenever there is no data to send.

- **User protocol data units (user PDUs)**: The trace data can be sent out by configuring \textit{trSetupR} register. We enable Time Reference Messages for testing purpose.

- **Channel PDU encapsulated sequence**: It demarcates the beginning and end of channel PDUs within the serial data stream, using /SCP/ (/K28.2/K27.7/) ordered set to mark the start of channel PDUs, and /ECP/ (/K29.7/K30.7/) ordered set to mark the end of channel PDUs.

The receiver will de-serialize and decode the received data. By comparing the received data with sent data through Chipscope, we can verify the Aurora channel.
4.2.3 Problems Encountered

To verify if the channel was setup correctly, first check if aligned, bonded, verified signals of RX are asserted, and then probe on rx_data to see if they match the sent sequence. The issue is that aligned, bonded, verified signals of RX did not turn high at first. Since the receiver has already been validated, the problem should be in TX of debug block.

The most direct way is to probe on pins txn and txp and make sure what TX sends out with the assistance of oscilloscope. By comparing the sending out waveform on oscilloscope with simulation result, we found that the actual waveform is in reverse order. Furthermore, the line rate is 1.25Gbps, half of the expected speed. A detailed discussion on reasons and solutions is carried out in the following part.

**Problem 1:** GTX TX sends out data in reverse order.

**Reason:** The Aurora core in debug provides big-endian data. The most significant byte is stored in the lowest address. But the GTX supports little-endian data. It serializes the rightmost parallel bit (LSB) first and the first byte is received on RXDATA (7 downto 0). Thus the sent out data is in reverse order, making receiver can not recognize the correct training sequence.

**Solution:** Reverse order of TXDATA[19:0], which goes to SERDES, so that GTX sends LSB first.

**Problem 2:** The line rate, which is supposed to be set as 2.5Gbps, is 1.25 Gbps.

**Reason:** The key factor is the setting of D divider value. Since we are not applying exactly the same protocol as described in *Virtex-6 FPGA GTX Transceivers User Guide* [4], there appears a mismatch divider setting.

**Solution:** Reconfigure D value according to Equation 3-2.
CHAPTER 4. SERDES TEST

After above improvements, the serial channel is finally established. Figure 4-4 displays the waveform captured on oscilloscope, showing the transition from Align phase to Verify phase, which coincides with the simulation result as described in Figure 4-5. And the default line rate is configured as 2.5Gbps, which can be changed by gtxDivSelR[2:1] - TXRATE.

Figure 4-4: Transition from Align to Verify (oscilloscope)

Figure 4-5: Transition from Align to Verify (simulation)\(^{18}\)

### 4.2.4 Results

The training sequence is sent from TX to RX under the control of auroraConfigR register by DebugTool. We use Chipscope to capture sequence and data received on RX end for verification purpose.

Table 4-2 and 4-3 provide reference information about training sequence. Code groups are 8B/10B encoded byte pairs. All data in Aurora 8B/10B is sent as code groups. Table 4-2 defines the ordered sets of special characters used by Aurora 8B/10B channel partners.

<table>
<thead>
<tr>
<th>Ordered Set</th>
<th>Designator</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>/I/</td>
<td>/K/, /R/, /A/ sequence</td>
</tr>
<tr>
<td>Verification</td>
<td>/V/</td>
<td>/K28.5/D8.7/D8.7/D8.7/</td>
</tr>
</tbody>
</table>

\(^{18}\) This result is supported by Andreas Magnusson, an ASIC designer from Ericsson AB.
Table 4-2: Aurora 8B/10B Ordered Sets

The Aurora 8B/10B protocol uses 8B/10B line coding containing Data characters and K characters, as shown in Table 4-3.

<table>
<thead>
<tr>
<th>Data Byte Name</th>
<th>Bytes</th>
<th>Special Code Name</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>D24.0</td>
<td>0x18</td>
<td>K27.7</td>
<td>0xFB</td>
</tr>
<tr>
<td>D10.2</td>
<td>0x4A</td>
<td>K28.0</td>
<td>0x1C</td>
</tr>
<tr>
<td>D8.7</td>
<td>0xE8</td>
<td>K28.2</td>
<td>0x5C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K28.3</td>
<td>0x7C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K28.5</td>
<td>0xBC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K29.7</td>
<td>0xFD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K30.7</td>
<td>0xFE</td>
</tr>
</tbody>
</table>

Table 4-3: Valid Data Characters and Valid K (control) Characters

1. Init phase

During initialization phase, TX will send out sequence /SP/, which is 0XBC4A4A4A according to Table 4-2 and 4-3. The sequence received is shown in Figure 4-6, which is consistent with theoretical value. Notice that align and lane_up signals are asserted, indicating channel align phase is completed.
2. Bond phase

The /I/ sequence is sent out during bonding phase, which is the combination of /K/, /R/ and /A/ sequence. The corresponding coding characters are 0xBC, 0x1C, 0x7C. See Table 4-2 for the definition of /I/, /K/, /R/ and /A/. Since there is only one lane implemented, bonding phase is not necessary. But RX can still receive the bonding sequence, as shown in the following figure.

3. Verify phase

The verification sequence consists of the following pattern: four-symbol /V/ ordered set which is 0xBCE8E8E8 in detail. See Table 4-2 for the definition of /V/. Figure 4-8 displays the received verify sequence with verify and rx_channel_up asserted, indicating initialization is completed and channel is setup.
Figure 4-8: Verify sequence received on RX

4. Data phase

After channel is ready, TX will send out idle sequence at the beginning of operation, indicating that there is no data. The idle ordered set consists of three code groups: /A/, /K/, and /R/. The 2-byte SCP code group is added to the beginning of the frame data to indicate the start of frame, and a 2-byte ECP set is sent after the frame ends to indicate the end of frame.

One Time Reference Message is sent every 256 clock cycles, which is set by trSetupR[7:6]. See [16] for definition of Trace Messages.

Figure 4-9 shows the sequence and data received on RX in the following order: idle sequence, SCP, TRM, ECP and idle sequence.
4.3 Summary

The serial interface of debug is verified at line rate of 2.5Gbps or 1.25Gbps, which is configured by TAP register $gtxDivSelR$. It is also capable of sending out trace messages under control of $trSetupR$ register via DebugTool-probe. This chapter provides foundation for further testing serial interface between debug and the latest probe, while the implementation of line rate of 3.125Gbps is still on the way, which remains as future work to improve.
Chapter 5

SW Execution and Tracing

This chapter discusses running software execution on DSP and sending out trace messages to debug block, using DebugTool. We mainly focus on implementation of the program and the problems encountered, and then demonstrate the result. Furthermore, a summary is delivered in the end.

5.1 Overview of SW Execution

After design is implemented in FPGA, it’s time to do SW execution to verify DSP and test debugger. Figure 5-1 shows the JTAG connection and trace data interface between debug and DSP. Through the JTAG interface, debug can access DSP memories and registers, which enables users to follow and control values changed in memories and registers. Thus the contents in DSP are visible and transparent.

DSP can send out millions of real time trace messages to TB, MEMORY or SERDES under control of DebugTool. This behavior enables users to know what DSP is doing and handle different situations manually, making DSP more observable and traceable.
SW execution procedure contains the following steps:

- Load program to MEMORY. Dump MEMORY to check if the code is successfully loaded.

- When DSP boots, it reads assembly program from MEMORY and stores it in LPM.

- DSP starts execution from addr0 in LPM.

It is done through DebugTool and probe.

DebugTool controls all aspects of SW execution, from loading assembly program to booting and stepping DSP, and also accessing the registers and memories. For information on how to do SW execution, refer to *FPGA Debug User Guide* [20].
5.2 Implementation

5.2.1 Function of basic program example

A little program called basic_test_debug_fpga.asm is produced to run on DSP, which is able to send out a few BUS-mails, ownership trace messages and then idles.

First, the program is loaded to MEMORY through the following tools and commands: flas, flink, flunk and loadobj. For details of execution, refer to *FPGA Debug User Guide* [20].

When DSP boots, it will read program from MEMORY and store it in LPM. Then DSP starts executing the program from addr0 in LPM. The code will send 3 BUS-mails to debug to set up a trace buffer in MEMORY. Then it will send out 3 ownership trace messages to debug. Finally, the trace messages are written into MEMORY through the trace buffer in debug. After that, the program will enter an infinity-loop.

The following section gives a detailed description of program flow.

Initializing Mailbox

The DSP core can communicate with other DSP:s (and units) in the system by sending and receiving messages. The messages are transported on a BUS connecting all DSP:s and other units. Incoming messages are placed directly in a configurable buffer in LDM, which is called *mailbox*. The first part of the code is to set up a mailbox in LDM, including the following information: Mailbox start pointer, length, read & write pointer, configuration details, mailbox enabled.

Sending BUS-enable-mail to DEBUG

When debug receives the BUS-enable-mail, it will prepare itself ready for BUS communication.
Setting up MEMORY Buffer

The trace buffer set up in MEMORY is used to store trace data. MEMORY address range is from 23’d0 to 23’h 5FFFFFF and the data width is aligned in 128 bits. It is configured in a JTAG register called MEMORYbSetupR with 46 bits. The low 23 bits indicates the MEMORY buffer start address and the high 23 bits determines the MEMORY buffer stop address. A MEMORY buffer is created from address 0x4000 to 0x4800. It is done through the following assembly code:

```
mv 0x9300,a0h | mv 0x001D,a0l
mv 0xC000,a1h | mv 0x0000,a1l
mv 0x0000,a2h | mv 0x0024,a2l
mv 0x0000,a3h | mv 0x4000,a3l
```

$a0$, $a1$, $a2$ and $a3$ are DSP accumulator registers with 32 bits width. They are shown in low 16 bits and high 16 bits separately.

0x9300 is BUS identifier for debug block. $a0l$ register stores the instruction code for MEMORYbSetupR register. For detailed definition of MEMORYbSetupR register, refer to SW IWD for Trace and Debug in Hermes [16].

The start address is set in $a3$ [22:0] and the stop address is stored in $a2$ [13:0] and $a3$ [31:23].

Setting up Trace Function

The trace functionality is configured through the JTAG trace setup register called trsetup with 31 bits width. By setting the value of trsetup register, the following functions are enabled:

- **BUS_TRACE_EN (trSetupR[3])**, enables writing BUS trace messages into the BUS trace fifo.

- **NXS_TRACE_EN (trSetupR[2])**, enables sending DSP trace to trace buffer.

- **TB_MEMORY_EN (trSetupR[1:0])**, enables emptying the trace buffer into the MEMORY.
It is customized by the following code:

\[
\begin{align*}
\text{mv 0x0000,a2h} & \quad | \quad \text{mv 0x0000,a2l} \\
\text{mv 0x0000,a3h} & \quad | \quad \text{mv 0x000e,a3l}
\end{align*}
\]

The detailed definition of \textit{trsetup} register can be found in \textit{SW IWD for Trace and Debug in Hermes} [16].

\textbf{Setting up Real Time Trace Register}

\textit{rttconf} is a DSP real time trace configuration register. The ownership trace messages are sent out by enabling OT in \textit{rttconf} register. The message will also be sent out after a write to the PSID register when OT is enabled. It is realized in the code below:

\[
\begin{align*}
\text{mv 0x3,a5l} \\
\text{mv a5l,rttconf} \\
\text{mv a5l,psidl} \\
\text{mv a5l,psidl}
\end{align*}
\]

We write into PSID register twice after enabling OT in \textit{rttconf}, so that DSP will send out three trace messages to debug.

\textbf{Entering an Infinity Loop}

In the end, the program will enter an infinity loop.

\textbf{5.2.2 Problems encountered}

This part discusses the problems encountered during execution of software. The most difficult problem was to seek proper approaches to complete the implementation process and make design fit on FPGA.

\textbf{Setting up Athena environment}

The first thing is to set up working environment for Athena by creating a new view and build a new set of tools. Then add module support for Athena to get Fladb/Flunk to start on Athena and thus be able to test booting DSPs.
To add the new module support, a new DebugTool setup script named ‘athena_dev’ is created with the environment variables and tools for Athena setup. The command is as follows:

```
flexsetup athena_dev
```

The chip information for Athena is defined in newly created athena.cluc. The board and JTAG information is configured in athena_tb.board and athena_tb.jtag as well.

**Accessing DSP Memory**

After booting DSP, we can start to test accessing memories and registers of DSP in Flunk. Refer to the *FPGA Debug User Guide* [20] for instructions on how to access DSP.

The registers of DSP can be successfully written and read. However, accessing LDM and LPM of DSP failed at the beginning. The problem should lie in the RAM wrapper of LDM and LPM. After carefully analyzing the behavioral model of RAM, we carried out the following steps to solve the problem.

First, there appears some initial memory VHDL code in the start of each RAM, which will assign data in memory to all zeroes. Meanwhile, the `wait on signal’transaction` does not apply in synthesis process. However, moving those signals to the sensitive list will make the memories always have contents of zeroes. As a result, it is decided to remove the initial memory code and re-synthesize.

The unexpected thing is that XST (Xilinx Synthesis Technology) gives a warning saying Area constraint could not be met after this kind of modification. We have to look for another way to make it work. The most convenient way is to replace the original RAM with a new RAM created by CORE Generator. However, this change makes the Place&Route process failed, left thousands of nets unrouted.
In general, the cause of this behavior is either overly difficult constraints, putting too much logic into this device, or an issue with the implementation. To achieve a fully routed design, try the following methods: Ease the constraints (if any); Remove some logic from the design; Or change the placement before running router again.

In our case, it is because the design is very dense and congested, a little change will make the router unable to finish the design and meet the requirements. Making such an extremely congested design fit FPGA is quite tricky and requires overall considerations.

To get the design completely routed, the following two methods are applied:

1. Choose the minimum RAM. There are four different types of RAM available for LDM and LPM separately. The smallest size is decided for LDM and LPM. This is done by setting the attributes in cfgPack.vhd, a package file for configuring memory sizes and version of core.

2. Reduce the size of RAM. To lower the oversize logic, the 8 bits ECC (Error Correcting Code) of LPM is removed on the premise of not affecting the functionality of design. Another big reduction is trimming the address width of RAM in LDM to only 3 bits, which is only 8 slots, thus the RAM size of LDM decreases to 8x16.

Finally, with plenty of efforts, the design is successfully routed and programming file is generated. With the newly generated RAM, memories of DSP can be managed to access in the end, offering a chance for the next execution.
Getting BUS work

To get SW execution work properly, one important thing is that \textit{trSetupR} register should be configured correctly. It is set by BUS mails. There is an input signal called \texttt{msg\_start\_in} of DSP BUS interface which should toggle for every rising edge of system clock. Then \texttt{msg\_start\_out} signal of DSP is sent to \texttt{msg\_start\_in} of debug. The \texttt{msg\_start\_out} signal of debug is left open. The BUS interface delays by two clock cycles. The BUS data is communicated between DSP and debug in a circular way. The BUS communication structure is shown in Figure 5-2.

![Figure 5-2: BUS Interface](image)

However, although generating clock toggle as BUS input message of DSP only consumes very little logic, it still burdens the router to complete implementation process. The solution is to compromise by reducing the system clock again. Since the smallest clk frequency that can be generated from CORE Generator is limited to 5MHz, to achieve a smaller clk, a clock divide-by-2 circuit is introduced and thus 2.5MHz frequency is generated. Finally, the Place&Route process is completed and BUS mails can be enabled and successfully sent out to debug.
5.3 Results of SW Execution

Configuration of trSetupR

When DSP0 boots, it starts executing the program and configuring trSetupR register. The settings of trSetupR register are as follows:

```
<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(CT)</td>
<td>0</td>
</tr>
<tr>
<td>(CTD)</td>
<td>0</td>
</tr>
<tr>
<td>(EVTI)</td>
<td>0</td>
</tr>
<tr>
<td>(EVT0)</td>
<td>0</td>
</tr>
<tr>
<td>(MB)</td>
<td>0x0000</td>
</tr>
<tr>
<td>(CFM)</td>
<td>0</td>
</tr>
<tr>
<td>(CM)</td>
<td>0</td>
</tr>
<tr>
<td>(CVM)</td>
<td>0</td>
</tr>
<tr>
<td>(CVMR)</td>
<td>0x0115</td>
</tr>
<tr>
<td>(CPS)</td>
<td>0</td>
</tr>
<tr>
<td>(TS)</td>
<td>0</td>
</tr>
<tr>
<td>(TP)</td>
<td>0</td>
</tr>
<tr>
<td>(PM)</td>
<td>0</td>
</tr>
<tr>
<td>(CMR)</td>
<td>0x0000</td>
</tr>
<tr>
<td>(CM)</td>
<td>0</td>
</tr>
<tr>
<td>(CMV)</td>
<td>0</td>
</tr>
<tr>
<td>(CMVR)</td>
<td>0</td>
</tr>
<tr>
<td>(CMR)</td>
<td>0x0000</td>
</tr>
<tr>
<td>(CM)</td>
<td>0</td>
</tr>
<tr>
<td>(CMV)</td>
<td>0</td>
</tr>
<tr>
<td>(CMVR)</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5-3: Trsetup Register Configuration

The trace data can be collected from DSP or BUS or both of them and Time Reference Messages can also be enabled and sent out. In this case, we use circular MEMORY trace buffer to collect trace data from both and do not issue any Time Reference Messages. The start and stop address of MEMORY trace buffer is set from 0x4000 to 0x4800. TB read pointer has moved to same place with TB write pointer, and MEMORY write pointer points to 0x4020, indicating that trace messages have been sent to MEMORY.

Trace Messages in MEMORY

The program defines some specific values of BUS mails and ownership trace messages for the purpose of a more accurate verification. They are done by the following code:
//Send a BUS mail directly to TB.

mv 0xFC00,a0h  | mv 0x0000,a0l
mv 0x1234,ah   | mv 0x5678,al
mv 0xabcd,a2h  | mv 0xdead,a2l
mv 0x9876,a3h  | mv 0xbeef,a3l

mv 0, piwa     | mv a0, pid
mv 1, piwa     | mv a1, pid
mv 2, piwa     | mv a2, pid
mv 3, piwa     | mv a3, pid

//Send OTM to MEMORY

mv 0x3,a5l
mv a5l,rttconf
mv 0xdead, a5l
mv a5l,psidl
mv 0xbeef, a5l
mv a5l,psidl

Decode the trace messages in MEMORY buffer and verify them. The result is shown in Figure 5-4.

```
athenas/dsp08 $trdecode -d 0x4000 0x40ef
Trace messages found in cn[0x4000.0x40ef]:
0x00000000 1: RTM  TCODE = 58  HEADER = 0xfc00  PAYLD1 = 0x58761234  TSTAMP = 0x351
0x00040100 2: RTM2  TCODE = 60  PAYLD2 = 0xdeadabac PAYLD3 = 0xbeef9876  TSTAMP = 0x352
0x00040100 3: OTM  TCODE = 2  SRC = dsp0  PID = 0x0  TSTAMP = 0x0db
0x00040100 4: OTM  TCODE = 2  SRC = dsp0  PID = 0xdead  TSTAMP = 0x0dd
0x00040100 5: OTM  TCODE = 2  SRC = dsp0  PID = 0xbeef  TSTAMP = 0x0df

OTM: Total: 5  Seg: 0  Skips: 2
RMB: Total: 2 (1 PART1, 1 PART2)

MSS statistics: Total: 5
LAT statistics: Total: 0
```

Figure 5-4: Trdecode Result

By checking the specification of BUS trace and OTM, and comparing received messages with those sent out from program, it is obvious that the trace messages written into MEMORY buffer are exactly the same with those specified in program. TSTAMP for the second half of BUS trace message increases by one after the first half, as defined in [16]. And TSTAMP increases every two clk cycles for each received OTM. As a result, it is confirmed that all the trace data has been successfully sent out to MEMORY. Execution of the assembly program
is finally completed and all the expected functions are realized. For definitions of OTM and BUS trace, refer to [16].

5.4 Summary

By doing software execution, it is possible to verify the dynamic real time trace debugging feature in Phoenix III core, and what’s more, enhance the debugging capability of the Ericsson ASIC software.

In this chapter, it has been proved that the trace messages sent out from DSP0 can be successfully observed and verified correctly during ongoing execution of software. This behavior delivers the following advantages: observation of internal information, reconstruction of the complete program flow and setup of watch point conditions.

The main difficulties reside in the following two parts:

1. Setting up Athena environment. At first, the debugging tool Flunk doesn’t support Athena core and it did take some time to discover the hidden unmatched problems between the design and the tools. In the end, the problem was solved and Flunk was improved to be compatible with Athena core.

2. Synthesis & implementation. This procedure is one of the most difficult parts throughout the whole thesis work and also extremely time-consuming, for the design logic is heavily overwhelming for ML605 FPGA board, which is already the latest and biggest one we can select at that time. To work around the tricky situation, the primary method is to lower the heavy logics.

In this chapter, the following two methods are applied:

- Trimming memories. Reducing the memory size is only one aspect. Another modification is to re-create the memory by Xilinx CORE
Generator since applying the RAM in behavioral model failed implementation process. The reason is that the generated RAM wrapper utilize the resource of memory blocks in FPGA, thus they save the usage of LUTs.

- Lowering clock frequency. Slowing down the system clock frequency will ease the heavy burden on FPGA resource requirements. It is the last choice if no other parts of design can be improved.
Chapter 6

Conclusion

6.1 Achievements

In brief, the set goals for the thesis have been largely accomplished. A \(\mu\)-EA system constituting a debug block, a minor MEMORY and one DSP has been prototyped in Xilinx ML605 FPGA board. The main achievements are listed as below:

- This \(\mu\)-EA system has been successfully synthesized on Virtex-6 FPGA board, with resource consumption of 62% after Place & Route at a frequency of 2.5MHz. TB, MEMORY and DSP were accessible as well.

- The high speed serial interface between debug and probe can be tested at 2.5Gbps and 1.25Gbps.

- DSP is capable of executing software to send out trace messages and verify special instructions, proving DSP verification acceleration possible.

The challenges encountered during this thesis work mainly reside in the following two parts:

- Get access of MEMORY, which consumes a great deal of time and effort.
CHAPTER 6. CONCLUSION

- Synthesis and PAR processes.

It is essential to highlight that synthesis and PAR are full of troubles and trials throughout the whole thesis work and quite time-consuming, because the design is quite dense and congested, and contains overwhelming logic, which leads to excessive consumption of FPGA resources.

The main experience obtained in this thesis is summarized as follows:

- It is of essence to take the trade off between performance and feasibility of the µ-EA system into account by designers.

- When we encounter problems on a quite complex and large design, it’s good to think from the most basic parts of a design to resolve the problem. Otherwise it is easy to get lost among a great deal of information.

- Take proper tools as an assistant to detect the problem, which will provide you an effective way to find out the issue.

- It is of great importance to effectively cooperate with other designers.

6.2 Future Work

The current work proves that SERDES port of debug is capable to send out trace messages, indicating the SERDES functionality validated. It is scheduled at the beginning to verify serial interface between debug and the latest probe with the Samtec adapter, which is used to provide connection between debug and the latest probe for serial interface testing. Due to the delay deliver of Samtec adapter, this part of work is yet to be done.

Another piece of work suggested is to test serial interface at 3.125Gbps on basis of experiments using Dynamic Reconfigurable Ports of GTX, when there is sufficient time, while the speed of 2.5Gbps and 1.25Gbps has been realized and configurable.
Appendices

Appendix A: Application of TAP registers

TAP data registers are controlled via JTAG to setup GTX, including reset, PLL locked done, and selection of the line rate. The line rate is determined by changing the D divider value through TXRATE port. The following TAP data registers are illustrated in Table A-1 to configure GTX.

<table>
<thead>
<tr>
<th>Name</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtxDivSelR (out)</td>
<td>11</td>
<td>Active-High PLL signals provide power down (Not included in GTX. If needed, the pins could be pull out.)</td>
</tr>
<tr>
<td></td>
<td>[10]</td>
<td>[9] Enable PLL lock detector (Not included in GTX.)</td>
</tr>
<tr>
<td></td>
<td>[8:7]</td>
<td>TXRATE, lane 4, controls the setting for the TX serial clock divider for low line rate support</td>
</tr>
<tr>
<td></td>
<td>[6:5]</td>
<td>TXRATE, lane 3, controls the setting for the TX serial clock divider for low line rate support</td>
</tr>
<tr>
<td></td>
<td>[4:3]</td>
<td>TXRATE, lane 2, controls the setting for the TX serial clock divider for low line rate support</td>
</tr>
<tr>
<td></td>
<td>[2:1]</td>
<td>TXRATE, lane 1, controls the setting for the TX serial clock divider for low line rate support</td>
</tr>
<tr>
<td></td>
<td>[0]</td>
<td>PLL reset (active-High)</td>
</tr>
<tr>
<td>gtxStatus (in)</td>
<td>12</td>
<td><strong>Lane 4 configurations:</strong> TXRESETDONE, GTX is ready for use</td>
</tr>
<tr>
<td></td>
<td>[11]</td>
<td>TXRATEDONE, TX rate change completion</td>
</tr>
<tr>
<td></td>
<td>[10]</td>
<td>Lane 4, PLL locked</td>
</tr>
<tr>
<td></td>
<td>[8]</td>
<td>TXRESETDONE, GTX is ready for use</td>
</tr>
<tr>
<td></td>
<td>[7]</td>
<td>TXRATEDONE, TX rate change completion</td>
</tr>
<tr>
<td></td>
<td>[6]</td>
<td>Lane 3, PLL locked</td>
</tr>
<tr>
<td></td>
<td>[5]</td>
<td>TXRESETDONE, GTX is ready for use</td>
</tr>
<tr>
<td>Lane 1 configurations:</td>
<td>Lane 4 configurations:</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>[1] TXRATEDONE, TX rate change completion</td>
<td>[85] TX reset, PCS TX system reset</td>
<td></td>
</tr>
<tr>
<td>[0] Lane 1, PLL locked</td>
<td>[84:81] TXDIFFCTRL, Specifies the differential swing level for the transmitter main driver</td>
<td></td>
</tr>
</tbody>
</table>

Lane 4 configurations:

- [87:86] TX power down
- [85] TX reset, PCS TX system reset
- [84:81] TXDIFFCTRL, Specifies the differential swing level for the transmitter main driver
- [80:76] TXPOSTEMPHASIS
- [75:72] TXPREEMPHASIS
- [71] TXDLYALIGNMONENB, reserved, tied High.
- [70] GTXTXRESET, start the full TX GTX reset sequence
- [69:67] TXENPBUSSTST, Enables the PBUSS Transmission control port
- [66] TXPBUSSFORCERERR, Enables the PBUSS force error control port

Lane 3 configurations:

- [65:64] TX power down
- [63] TX reset, PCS TX system reset
- [62:59] TXDIFFCTRL, Specifies the differential swing level for the transmitter main driver
- [58:54] TXPOSTEMPHASIS
- [53:50] TXPREEMPHASIS
- [49] TXDLYALIGNMONENB, reserved, tied High.
- [48] GTXTXRESET, start the full TX GTX reset sequence
- [47:45] TXENPBUSSTST, Enables the PBUSS Transmission control port
- [44] TXPBUSSFORCERERR, Enables the PBUSS force error control port

Lane 2 configurations:

- [43:42] TX power down
APPENDIX A: APPLICATION OF TAP REGISTERS

Table A-1: TAP data registers to configure GTX

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>TX reset, PCS TX system reset</td>
</tr>
<tr>
<td>40:37</td>
<td>TXDIFFCTRL, Specifies the differential swing level for the transmitter main driver</td>
</tr>
<tr>
<td>36:32</td>
<td>TXPOSTEMPHASIS</td>
</tr>
<tr>
<td>31:28</td>
<td>TXPREEMPHASIS</td>
</tr>
<tr>
<td>27</td>
<td>TxDLYALIGNMONENB, reserved, tied High.</td>
</tr>
<tr>
<td>26</td>
<td>GTXTXRESET, start the full TX GTX reset sequence</td>
</tr>
<tr>
<td>25:23</td>
<td>TXENPBUSTSTST, Enables the PBUSS Transmission control port</td>
</tr>
<tr>
<td>22</td>
<td>TXPBUSSFORCEERR, Enables the PBUSS force error control port</td>
</tr>
<tr>
<td>21:20</td>
<td>TX power down</td>
</tr>
<tr>
<td>00</td>
<td>P0 (normal operation)</td>
</tr>
<tr>
<td>01</td>
<td>P0s (low recovery time power down)</td>
</tr>
<tr>
<td>10</td>
<td>P1 (longer recovery time/Receiver detection still on)</td>
</tr>
<tr>
<td>11</td>
<td>P2 (lowest power state)</td>
</tr>
<tr>
<td>19</td>
<td>TX reset, PCS TX system reset (active high)</td>
</tr>
<tr>
<td>18:15</td>
<td>TXDIFFCTRL, Specifies the differential swing level for the transmitter main driver</td>
</tr>
<tr>
<td>14:10</td>
<td>TXPOSTEMPHASIS</td>
</tr>
<tr>
<td>9:6</td>
<td>TXPREEMPHASIS</td>
</tr>
<tr>
<td>5</td>
<td>TxDLYALIGNMONENB, reserved, tied High.</td>
</tr>
<tr>
<td>4</td>
<td>GTXTXRESET, start the full TX GTX reset Sequence(active high)</td>
</tr>
<tr>
<td>3:1</td>
<td>TXENPBUSTSTST, Enables the PBUSS Transmission control port</td>
</tr>
<tr>
<td>0</td>
<td>TXPBUSSFORCEERR, Enables the PBUSS force error control port</td>
</tr>
</tbody>
</table>

Lane 1 configurations:

- TX power down
  - 00: P0 (normal operation)
  - 01: P0s (low recovery time power down)
  - 10: P1 (longer recovery time/Receiver detection still on)
  - 11: P2 (lowest power state)
Appendix B: FMC Connection Pinout

A Samtec adapter is connected to FMC HPC connector on ML605 board to provide the following connectors:

- 4 Samtec connectors: 2 for serial nexus receiver and 2 for serial nexus transmitter.
- 2 separate JTAG ports, as two 2x8 pinheader connectors (2.54 mm spacing).

It’s one table per connector on the board, as listed below.

<table>
<thead>
<tr>
<th>J64 FMC HPC Pin</th>
<th>JTAG Pin</th>
<th>I/O</th>
<th>Pin Number</th>
<th>Pin Number</th>
<th>I/O</th>
<th>JTAG Pin</th>
<th>J64 FMC HPC Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>K7</td>
<td>TRST_N</td>
<td>IN</td>
<td>1</td>
<td>2</td>
<td>GND</td>
<td>K6</td>
<td></td>
</tr>
<tr>
<td>K8</td>
<td>TDI</td>
<td>IN</td>
<td>3</td>
<td>4</td>
<td>GND</td>
<td>K6</td>
<td></td>
</tr>
<tr>
<td>K10</td>
<td>TDO</td>
<td>OUT</td>
<td>5</td>
<td>6</td>
<td>GND</td>
<td>K6</td>
<td></td>
</tr>
<tr>
<td>K11</td>
<td>TMS</td>
<td>IN</td>
<td>7</td>
<td>8</td>
<td>GND</td>
<td>K6</td>
<td></td>
</tr>
<tr>
<td>K13</td>
<td>TCK</td>
<td>IN</td>
<td>9</td>
<td>10</td>
<td>GND</td>
<td>K6</td>
<td></td>
</tr>
<tr>
<td>J6</td>
<td>NC</td>
<td>/</td>
<td>11</td>
<td>12</td>
<td>GND</td>
<td>K6</td>
<td></td>
</tr>
<tr>
<td>J7</td>
<td>NC</td>
<td>/</td>
<td>13</td>
<td>14</td>
<td>GND</td>
<td>K6</td>
<td></td>
</tr>
<tr>
<td>K14</td>
<td>BUSY</td>
<td>OUT</td>
<td>15</td>
<td>16</td>
<td>GND</td>
<td>K6</td>
<td></td>
</tr>
</tbody>
</table>

Table B-1: JTAG 1 Connector Pinout

<table>
<thead>
<tr>
<th>J64 FMC HPC Pin</th>
<th>JTAG Pin</th>
<th>I/O</th>
<th>Pin Number</th>
<th>Pin Number</th>
<th>I/O</th>
<th>JTAG Pin</th>
<th>J64 FMC HPC Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>K22</td>
<td>TRST_N</td>
<td>IN</td>
<td>1</td>
<td>2</td>
<td>GND</td>
<td>K36</td>
<td></td>
</tr>
<tr>
<td>K23</td>
<td>TDI</td>
<td>IN</td>
<td>3</td>
<td>4</td>
<td>GND</td>
<td>K36</td>
<td></td>
</tr>
<tr>
<td>K31</td>
<td>TDO</td>
<td>OUT</td>
<td>5</td>
<td>6</td>
<td>GND</td>
<td>K36</td>
<td></td>
</tr>
<tr>
<td>K32</td>
<td>TMS</td>
<td>IN</td>
<td>7</td>
<td>8</td>
<td>GND</td>
<td>K36</td>
<td></td>
</tr>
<tr>
<td>K34</td>
<td>TCK</td>
<td>IN</td>
<td>9</td>
<td>10</td>
<td>GND</td>
<td>K36</td>
<td></td>
</tr>
<tr>
<td>J36</td>
<td>NC</td>
<td>/</td>
<td>11</td>
<td>12</td>
<td>GND</td>
<td>K36</td>
<td></td>
</tr>
<tr>
<td>J37</td>
<td>NC</td>
<td>/</td>
<td>13</td>
<td>14</td>
<td>GND</td>
<td>K36</td>
<td></td>
</tr>
<tr>
<td>K35</td>
<td>BUSY</td>
<td>OUT</td>
<td>15</td>
<td>16</td>
<td>GND</td>
<td>K36</td>
<td></td>
</tr>
</tbody>
</table>

Table B-2: JTAG 2 Connector Pinout
# APPENDIX B: FMC CONNECTION PINOUT

## Table B-3: Samtec Connector ERx8 High Speed Serial for Transmitter 1 (High Speed Simplex)

<table>
<thead>
<tr>
<th>J64 FMC HPC Pin</th>
<th>FMC Schematic Net Name</th>
<th>Functions</th>
<th>I/O</th>
<th>Pin Number</th>
<th>Pin Number</th>
<th>Functions</th>
<th>FMC Schematic Net Name</th>
<th>J64 FMC HPC Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>FMC_HPC_DP0 C2M_P</td>
<td>MGTTXP3 113</td>
<td>OUT</td>
<td>1</td>
<td>2</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>C3</td>
<td>FMC_HPC_DP0 C2M_N</td>
<td>MGTTXN3 113</td>
<td>OUT</td>
<td>3</td>
<td>4</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A21</td>
<td>GND</td>
<td>GND</td>
<td></td>
<td>5</td>
<td>6</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A22</td>
<td>FMC_HPC_DP1 C2M_P</td>
<td>MGTTXP2 113</td>
<td>OUT</td>
<td>7</td>
<td>8</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A23</td>
<td>FMC_HPC_DP1 C2M_N</td>
<td>MGTTXN2 113</td>
<td>OUT</td>
<td>9</td>
<td>10</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A24</td>
<td>GND</td>
<td>GND</td>
<td></td>
<td>11</td>
<td>12</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A26</td>
<td>FMC_HPC_DP2 C2M_P</td>
<td>MGTTXP1 113</td>
<td>OUT</td>
<td>13</td>
<td>14</td>
<td>GEN IO0</td>
<td>FMC_HPC_LA27_P</td>
<td>C26</td>
</tr>
<tr>
<td>A27</td>
<td>FMC_HPC_DP2 C2M_N</td>
<td>MGTTXN1 113</td>
<td>OUT</td>
<td>15</td>
<td>16</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A28</td>
<td>GND</td>
<td>GND</td>
<td></td>
<td>17</td>
<td>18</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A30</td>
<td>FMC_HPC_DP3 C2M_P</td>
<td>MGTTXP0 113</td>
<td>OUT</td>
<td>19</td>
<td>20</td>
<td>GEN IO3</td>
<td>FMC_HPC_LA27_N</td>
<td>C27</td>
</tr>
<tr>
<td>A31</td>
<td>FMC_HPC_DP3 C2M_N</td>
<td>MGTTXN0 113</td>
<td>OUT</td>
<td>21</td>
<td>22</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>

## Table B-4: Samtec Connector ERx8 High Speed Serial Transmitter 2 (High Speed Simplex)

<table>
<thead>
<tr>
<th>J64 FMC HPC Pin</th>
<th>FMC Schematic Net Name</th>
<th>Functions</th>
<th>I/O</th>
<th>Pin Number</th>
<th>Pin Number</th>
<th>Functions</th>
<th>FMC Schematic Net Name</th>
<th>J64 FMC HPC Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>B32</td>
<td>FMC_HPC_DP7 C2M_P</td>
<td>MGTTXP0 112</td>
<td>OUT</td>
<td>1</td>
<td>2</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>B33</td>
<td>FMC_HPC_DP7 C2M_N</td>
<td>MGTTXN0 112</td>
<td>OUT</td>
<td>3</td>
<td>4</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A33</td>
<td>GND</td>
<td>GND</td>
<td></td>
<td>5</td>
<td>6</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A34</td>
<td>FMC_HPC_DP4 C2M_P</td>
<td>MGTTXP3 112</td>
<td>OUT</td>
<td>7</td>
<td>8</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A35</td>
<td>FMC_HPC_DP4 C2M_N</td>
<td>MGTTXN3 112</td>
<td>OUT</td>
<td>9</td>
<td>10</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A36</td>
<td>GND</td>
<td>GND</td>
<td></td>
<td>11</td>
<td>12</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>B36</td>
<td>FMC_HPC_DP6 C2M_P</td>
<td>MGTTXP1 112</td>
<td>OUT</td>
<td>13</td>
<td>14</td>
<td>GEN IO0</td>
<td>FMC_HPC_HB21_P</td>
<td>E36</td>
</tr>
<tr>
<td>B37</td>
<td>FMC_HPC_DP6 C2M_N</td>
<td>MGTTXN1 112</td>
<td>OUT</td>
<td>15</td>
<td>16</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A37</td>
<td>GND</td>
<td>GND</td>
<td></td>
<td>17</td>
<td>18</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A38</td>
<td>FMC_HPC_DP5 C2M_P</td>
<td>MGTTXP2 112</td>
<td>OUT</td>
<td>19</td>
<td>20</td>
<td>GEN IO3</td>
<td>FMC_HPC_HB21_N</td>
<td>E37</td>
</tr>
<tr>
<td>A39</td>
<td>FMC_HPC_DP5 C2M_N</td>
<td>MGTTXN2 112</td>
<td>OUT</td>
<td>21</td>
<td>22</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>
## APPENDIX B: FMC CONNECTION PINOUT

### Table B-5: Samtec Connector ERx8 High Speed Serial for Receiver 1 (High Speed Simplex)

<table>
<thead>
<tr>
<th>J64 FMC HPC Pin</th>
<th>FMC Schematic Net Name</th>
<th>Functions</th>
<th>I/O</th>
<th>Pin Number</th>
<th>Pin Number</th>
<th>I/O</th>
<th>Functions</th>
<th>FMC Schematic Net Name</th>
<th>J64 FMC HPC Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>FMC_HPC_DP1_M2C_P</td>
<td>MGTRXP2_113</td>
<td>IN</td>
<td>1</td>
<td>2</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A3</td>
<td>FMC_HPC_DP1_M2C_N</td>
<td>MGTRXN2_113</td>
<td>IN</td>
<td>3</td>
<td>4</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A4</td>
<td>GND</td>
<td>GND</td>
<td>/</td>
<td>5</td>
<td>6</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>C6</td>
<td>FMC_HPC_DP0_M2C_P</td>
<td>MGTRXP3_113</td>
<td>IN</td>
<td>7</td>
<td>8</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>C7</td>
<td>FMC_HPC_DP0_M2C_N</td>
<td>MGTRXN3_113</td>
<td>IN</td>
<td>9</td>
<td>10</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A5</td>
<td>GND</td>
<td>GND</td>
<td>/</td>
<td>11</td>
<td>12</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A6</td>
<td>FMC_HPC_DP2_M2C_P</td>
<td>MGTRXP1_113</td>
<td>IN</td>
<td>13</td>
<td>14</td>
<td>I/O</td>
<td>GEN_IO0</td>
<td>FMC_HPC_LA06_P</td>
<td>C10</td>
</tr>
<tr>
<td>A7</td>
<td>FMC_HPC_DP2_M2C_N</td>
<td>MGTRXN1_113</td>
<td>IN</td>
<td>15</td>
<td>16</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A8</td>
<td>GND</td>
<td>GND</td>
<td>/</td>
<td>17</td>
<td>18</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A10</td>
<td>FMC_HPC_DP3_M2C_P</td>
<td>MGTRXP0_113</td>
<td>IN</td>
<td>19</td>
<td>20</td>
<td>I/O</td>
<td>GEN_IO3</td>
<td>FMC_HPC_LA06_N</td>
<td>C11</td>
</tr>
<tr>
<td>A11</td>
<td>FMC_HPC_DP3_M2C_N</td>
<td>MGTRXN0_113</td>
<td>IN</td>
<td>21</td>
<td>22</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>

### Table B-6: Samtec Connector ERx8 High Speed Serial for Receiver 2 (High Speed Simplex)

<table>
<thead>
<tr>
<th>J64 FMC HPC Pin</th>
<th>FMC Schematic Net Name</th>
<th>Functions</th>
<th>I/O</th>
<th>Pin Number</th>
<th>Pin Number</th>
<th>I/O</th>
<th>Functions</th>
<th>FMC Schematic Net Name</th>
<th>J64 FMC HPC Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>B12</td>
<td>FMC_HPC_DP7_M2C_P</td>
<td>MGTRXP0_112</td>
<td>IN</td>
<td>1</td>
<td>2</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>B13</td>
<td>FMC_HPC_DP7_M2C_N</td>
<td>MGTRXN0_112</td>
<td>IN</td>
<td>3</td>
<td>4</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A13</td>
<td>GND</td>
<td>GND</td>
<td>/</td>
<td>5</td>
<td>6</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A14</td>
<td>FMC_HPC_DP4_M2C_P</td>
<td>MGTRXP3_112</td>
<td>IN</td>
<td>7</td>
<td>8</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A15</td>
<td>FMC_HPC_DP4_M2C_N</td>
<td>MGTRXN3_112</td>
<td>IN</td>
<td>9</td>
<td>10</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A16</td>
<td>GND</td>
<td>GND</td>
<td>/</td>
<td>11</td>
<td>12</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>B16</td>
<td>FMC_HPC_DP6_M2C_P</td>
<td>MGTRXP1_112</td>
<td>IN</td>
<td>13</td>
<td>14</td>
<td>I/O</td>
<td>GEN_IO0</td>
<td>FMC_HPC_LA14_P</td>
<td>C18</td>
</tr>
<tr>
<td>B17</td>
<td>FMC_HPC_DP6_M2C_N</td>
<td>MGTRXN1_112</td>
<td>IN</td>
<td>15</td>
<td>16</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A17</td>
<td>GND</td>
<td>GND</td>
<td>/</td>
<td>17</td>
<td>18</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>A18</td>
<td>FMC_HPC_DP5_M2C_P</td>
<td>MGTRXP2_112</td>
<td>IN</td>
<td>19</td>
<td>20</td>
<td>I/O</td>
<td>GEN_IO3</td>
<td>FMC_HPC_LA14_N</td>
<td>C19</td>
</tr>
<tr>
<td>A19</td>
<td>FMC_HPC_DP5_M2C_N</td>
<td>MGTRXN2_112</td>
<td>IN</td>
<td>21</td>
<td>22</td>
<td>I/O</td>
<td>RESERVED</td>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>
References


[8] Aurora 8B/10B Protocol Specification, SP002 (v2.2), Xilinx, April 19, 2010


[10] MEMORYI HW IWD, General, Michael Carlsson C, Ericsson internal, March 10, 2010


[12] ISE In-Depth Tutorial, UG695 (v13.1), Xilinx, March 1, 2011


[16] SW IWD for Trace and Debug in Hermes, Andreas Magnusson, January 10, 2011

