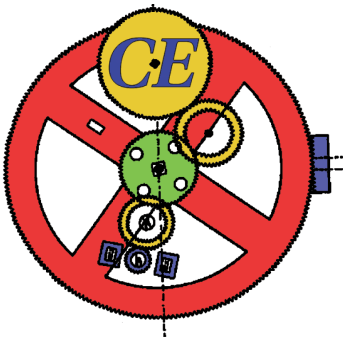


Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2015

MSc THESIS

An Incremental VON-Based Debug System for Commercial FPGA Architecture

Roshan Kumar Gupta

Abstract

Electronic companies are increasingly using field-programmable gate arrays in various domains such as application acceleration, complex digital designs or ASIC prototyping. The Verification phase holds a significant place in the FPGA design development process. A key challenge during verification is observability. This is defined as the ability to view all internal states of a circuit. Due to poor observability, a significant portion of designer's effort is spent in this phase, specifically performing the debugging task. A common solution to improve observability is using embedded logic analyzers (ELA) that inserts trace-buffers into the design to record on-chip signal values. When on-chip memory is used for observation it is termed as trace-buffers. This approach has limitations such as slow debug cycles, pre-determining the signals to be traced or using logic resources on FPGA.
This work proposes a new debug system for improving the observability while overcoming the limitations of ELAs. The proposed debug system extends a recent technique referred as virtual overlay network (VON) for commercial FPGA device. This network can be perceived as built on top of initial circuit mapping and multiplexes all circuit signals to the on-chip memory for observation. It overcomes the limitation of commercial debug tools based on ELAs. We investigate the factors that influence the performance of VON for Xilinx Virtex, as it constitutes the core of debug system. We demonstrate that a new bit-stream to program the FPGA connecting hundreds of signals to the on-chip memory can be generated in less than 630 seconds, during debug cycle, for a fairly large circuit having normal re-compilation time of more than 5 hours. The proposed system proves to be a promising way of improving observability and potentially reducing the debug turn time with zero area overhead. Currently, the system is limited to work with Xilinx Virtex family of devices.

CE-MS-2015-09

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

An Incremental VON-Based Debug System for Commercial FPGA Architecture

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Roshan Kumar Gupta
born in Patna, India

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

An Incremental VON-Based Debug System for Commercial FPGA Architecture

by Roshan Kumar Gupta

Abstract

Electronic companies are increasingly using field-programmable gate arrays in various domains such as application acceleration, complex digital designs or ASIC prototyping. The Verification phase holds a significant place in the FPGA design development process. A key challenge during verification is observability. This is defined as the ability to view all internal states of a circuit. Due to poor observability, a significant portion of designer's effort is spent in this phase, specifically performing the debugging task. A common solution to improve observability is using embedded logic analyzers (ELA) that inserts trace-buffers into the design to record on-chip signal values. When on-chip memory is used for observation it is termed as trace-buffers. This approach has limitations such as slow debug cycles, pre-determining the signals to be traced or using logic resources on FPGA.

This work proposes a new debug system for improving the observability while overcoming the limitations of ELAs. The proposed debug system extends a recent technique referred as virtual overlay network (VON) for commercial FPGA device. This network can be perceived as built on top of initial circuit mapping and multiplexes all circuit signals to the on-chip memory for observation. It overcomes the limitation of commercial debug tools based on ELAs. We investigate the factors that influence the performance of VON for Xilinx Virtex, as it constitutes the core of debug system. We demonstrate that a new bit-stream to program the FPGA connecting hundreds of signals to the on-chip memory can be generated in less than 630 seconds, during debug cycle, for a fairly large circuit having normal re-compilation time of more than 5 hours. The proposed system proves to be a promising way of improving observability and potentially reducing the debug turn time with zero area overhead. Currently, the system is limited to work with Xilinx Virtex family of devices.

Laboratory : Computer Engineering
Codenumber : CE-MS-2015-09

Committee Members :

Advisor:	Stephan Wong, CE, TU Delft
Chairperson:	Stephan Wong, CE, TU Delft
Member:	Arjan van Genderen, CE, TU Delft
Member:	Arjan Palm, EGD, ASML Netherlands B.V.
Member:	Rene van Leuken, CAS, TU Delft

Dedicated to my family and friends

Contents

List of Figures

List of Tables

List of Acronyms

ASIC Application Specific Integrated Circuit

ASML Advanced Semiconductor Materials Lithography

BLE Basic logic element

CAD Computer aided design

VON Virtual Overlay Network

FPGA Field Programmable Gate Array

IC Integrated Chips

ELA Embedded Logic Analyzers

VTR Verilog-to-routing

I/O Input/Output

RAM Random access memory

DSP Digital signal processing

VTB VTR-to-bitstream

MWBM Maximum Weighted Bipartite Matching

VPR Versatile Pack and Route

ECO Engineering Change Order

LUT Look Up Table

PIP Programmable Interconnect Points

TB Trace Buffer

ADL Architecture Description Language

Acknowledgement

This Master Thesis constitutes the final part of my study program required to obtain a degree of Master of Science in Embedded Systems at Delft University of Technology, The Netherlands. During the first year of my master's program I learned that it is very important to have industrial experience along-with theoretical expertise and hence, I decided to pursue my master thesis at ASML while having a strong focus on academic research.

In the beginning, I was presented with a raw idea and was asked to develop it. I took the complete ownership of the initial idea, developed it further and worked towards finding a solution for it. The topic involved working on the lower abstract layers of the FPGA design aids & architecture technologies with complete focus on every related aspect of it. I felt challenged and motivated by the amount of work, detailed knowledge and complexity involved, and the fact that it have the potential to directly impact the FPGA development process. During the entire duration of my master thesis I was exposed to scientific fields that I had no previous experience, had the opportunity to remarkably improve my scientific skills and gained the ability to independently pursue a research work.

I would like to express deepest gratitude to my supervisor at university, Stephan Wong, and at company, Arjan Palm for guiding me throughout the thesis project with their expertise & experience. During my meetings with both of them, I was shown the right path to do research, apply critical thinking and how to manage the complete work. Next, I would like to thank Mr. Paul van der Heijde for believing in me and offering the graduation project. I would also thank, Arjan van Genderen, for regularly discussing the topic in detail and provide feedback throughout the project. I would also like to extend my thanks to Rene van Leuken for being the external member in my thesis committee.

Many thanks goes to all my fellow colleagues Arnica Ajay Agarwal, Razvan Nane, Anthony Brandon, Dan, Mike, Anupama, Komal and rest of the people from EGD group at ASML and CE Dept. at TU Delft for their ideas, reviews & feedbacks.

Last but not least, I would like to thank my family and especially my brother who provided the financial support for my master studies and their mental support throughout my entire study period.

Roshan Kumar Gupta
Delft, The Netherlands
December 9, 2015

1 Introduction

This thesis describes the design and implementation of a debug system for FPGA verification that reduces the debugging time and improves observability. This chapter provides an introduction to the problem. It begins with the context in which the work originated and afterwards, we discuss the motivation for an incremental trace-based debug system. We refer to incremental compilation as the possibility to independently run individual stages of a tool that compile circuit designs for FPGAs. Subsequently, we discuss the problem statement that is researched in this thesis and the contributions made by this work. The approach section discusses the methodology adopted to analyze the problem statement. Finally, the overview of the rest of the work is given.

1.1 Context

Moore's law is one of the important laws of digital electronics that have contributed towards world economic growth during previous decades. IBM has recently shown a 7nm prototype chips containing approximately 20 billion transistors with four times more performance. With such an advancement in technology, today's Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Arrays (FPGA) are bound to become more complex. Recently, electronic companies have started to increasingly use FPGAs in various domains such as application accelerators, implementing complex digital designs or for FPGA prototyping of ASIC designs. Since FPGAs are used extensively, it is critical for the digital designs implemented on these devices to function properly otherwise the consequences could be costly or even life-threatening. The previous statement and the growing complexity of digital designs exemplifies the need for design verification. Incidents like the Intel FDIV bug are evidence to the same.

Recently, FPGA prototyping is increasingly used for ASIC verification, that is, the method to prototype the ASIC design on FPGA for hardware verification and early software development. Verification is an important phase of the design development since it drives the time, cost and quality of the complete FPGA product development. It is equally important and follows the same procedure whether it is performed for specific FPGA design methodology or during the use of FPGA as prototyping platform.

A worldwide study conducted by Wilson research group, commissioned by Mentor Graphics in 2014 shows that 45% [1] of the industry used FPGA prototyping for IC verification and this is expected to sharply increase with the newer FPGAs having larger gate counts. FPGA platforms are a considerate choice for verification as they are faster than logic simulation in terms of maximum design frequency with higher design coverage. While still costing much less than an ASIC prototype fabrication with almost no lead time compared to the waiting time for the tapeout. [2] shows that IBM engineers reported that full chip-level testing using a multi-FPGA prototype is 100,000 times faster than software

simulation which emphasize the use of FPGAs for the mentioned purpose. Moreover, todays FPGA designs have grown in complexity with 56% [1] of FPGAs containing one or more embedded processors along with complex network-on-a-chip interconnect. Out of the total FPGA project time, 46% [1] is spent on verification as of 2014. The average mean time spent by an FPGA design engineer (in an FPGA project team composed of development and verification engineers) doing just verification is 51% [1] and verification engineers are spending 43% [1] of their time specifically on debugging tasks.

A key requirement during any verification procedure concerning FPGAs is observability, that is, the ability to view all internal states of the circuit, analogous to the ability of software debugger to view the values of variables. Observability is key to verifying behavior and track down bugs in circuit design. However, limited or no observability of internal signals is an inherent drawback of FPGAs. It causes the internal signals to be brought out through the I/O pins of FPGA for observation. These I/O resources are limited and depends on the physical size of the device. Unlike simulators that can provide full observability into all the circuit signals, FPGAs can provide only a small subset to be observed through the I/O pins. During debugging it is imperative to observe any signal instead of just observing handful of signals through the I/O pins.

A common solution for increasing observability is to use embedded logic analyzer (ELA) tools based on the trace-based approach of inserting trace-buffers into the circuit design and connecting signals to them for observation. Trace buffers are formed of on-chip memory elements that record the history of the subset of internal signals connected to them during circuit operation. This history is then used to analyze the state of the circuit signals to perform debugging tasks and can further be extracted for offline analysis. Although, ELAs provide more observability than a standard logic analyzer that can only observe the I/O pins, they have several limitations. Providing new and efficient ways of increasing the observability can be perceived as one of the key technologies required. In the context of overcoming the limitations of ELAs thereby improving observability within FPGAs, this work has been carried out.

1.2 Motivation

A number of commercially available standard debugging tools are offered by FPGA vendors based on the concept of ELAs. These tools use the on-chip memory as trace-buffers and add probes directly to the RTL design to make specific signal available for observation via trace-buffers but suffers from following limitations.

1. ELAs require extensive LUT and memory resources on the FPGA to implement debug instrumentation, this limits their performance in case enough FPGA resources are not present. It can be termed as area overhead of using ELAs.

2. The signals that a designer wishes to observe must be pre-determined during the insertion of debug instrumentation, that is, before the circuit is operational and the nature of the bug known. This means that many debug cycles may be required to observe different subsets of signals to track down the bug. A debug cycle is the complete process of selecting a different combination of circuit signal, long place-and-route times of changing probes to observe them and re-synthesize the design

to program the FPGA again. The time taken to perform one such cycle is termed as debug turn time. With the increasing complexity of the designs, debug turn times tends to be in hours that hugely affects the debug productivity. Incremental routing can be used to connect these ELAs without a complete recompile but are still slow and requires the entire design to be loaded into the memory [3] [4] [5].

3. The ELAs may influence the initial mapping and timing characteristics of the circuit being instrumented thereby hiding potential bugs.

These limitations are also recognized by the FPGA designers at Advanced Semiconductor Materials Lithography (ASML) where this work has been carried out. ASML is a high-tech company that builds lithography machines for manufacturing chips being a world leader in its domain. FPGAs are solely used in their machines to implement various functionality. Since designers at ASML use standard debugging tools, they also felt limited by their drawbacks during FPGA design development. Hence, a debug system is wished for that can reduce the debug turn time, have no area overhead and improves observability of internal signals by connecting hundreds of them to debug instrumentation with-in FPGA. Such a system will be able to address the more general problem of overcoming the limitations of ELAs thereby improving the observability as well as serving the need of such a system specific to designer's at ASML.

The purpose of this work is to propose and demonstrate a debug system that overcomes the existing limitations of ELAs by using a technique called virtual overlay network (VON). This network is incrementally built on top of existing physical mapping of a placed-and-routed FPGA design, multiplexing all the circuit signals to trace-buffers and can be merged together with the physical mapping of circuit during debugging. It can be perceived as a virtual mesh layer on top of existing design and hence is referred as virtual overlay network and is further explained in Section 2.5 of next chapter. The next section describes the problem statement that will be researched in this work and highlights the contributions.

1.3 Problem statement and contribution

The virtual overlay network (VON) technique is only known to be demonstrated on a hypothetical FPGA architecture with simplifying assumptions. Therefore, it is imperative to implement it for a realistic architecture, evaluate its performance and to know its feasibility towards using it as a core technique in an debug system. A realistic FPGA architecture can be best represented by a commercially available device. Xilinx Virtex family of FPGA devices represent a comprehensive generation of FPGA architecture and is used for the purpose of this work. The insights gathered from this work then can serve as a prototype to design and build a complete system that overcomes the limitation of embedded logic analyzer thereby improving observability. The problem statement of this thesis is:

For a commercial FPGA architecture (like Xilinx Virtex family), what will be the performance of a virtual overlay network based debug system?

To answer this question, we first need to extend this technique for such an FPGA architecture and build a debug system using a CAD flow that allows open access to its different stages such as packing, placing or routing. Afterward, we need to identify and measure the factors that influence the performance of debug system. Such a debug system will consists of various stages. It will take a circuit's HDL description and will generate bit-stream to program the device. It will be realized using a open-source CAD flow known as VTR [6]. Performance of this tool for a commercial architecture and generation of a valid bit-stream after the insertion of this instrumentation that can be used to program the device is demonstrated.

The work described in this thesis has the following unique characteristics:

1. It extends the recently proposed virtual overlay network for a commercial architecture.

2. A new debug flow that can generate bit-stream for a commercial architecture alongwith debug instrumentation.

Consequently, the contribution of this work is to extend and demonstrate the feasibility of a recently proposed (Overlay network) trace-based approach on a commercial architecture and implement a debug system based on it. These unique characteristics are not found in any other approach. It also overcomes the limitations of ELAs mentioned in previous section that are based on trace-based approaches by using incremental compilation method (described in Section 2.2). Implementation details and associated challenges of realizing this work for Xilinx Virtex-6 FPGA are described in the implementation section. The new debug flow developed to do this work are the first to demonstrate the capability on a realistic device and suggest ways to improve on the limitations of ELAs. Along with the unique characteristics, it contributes towards answering the following questions:

1. What is the impact of an overlay network on the critical path delay?

2. Is the overlay network approach suitable enough for a commercial architecture?

3. How much runtime saving can be made during debug cycle?

The methodology adopted to achieve the contributions and to carry out the research on problem statement, mentioned above is discussed in the next section.

1.4 Approach

In order to realize such a debug system, we reviewed existing and recently proposed techniques to ascertain the possibility of coming up with a solution that is feasible and meets the requirements. The virtual overlay network [7] is identified as having the potential to address above mentioned problems. Similar to commercial trace-based ELAs, this technique utilizes trace-based approach to incrementally embed a network on top of existing circuit mapping while reclaiming on-chip memory as trace-buffers. It is demonstrated as a tool known as QuickTrace [7].

The proposed debug system extends this tool for a commercial architecture. It enable the designers to run the design live on FPGA, record the history and extract the information from the trace buffers with techniques like device read-back [8] for offline analysis. Logic on the chip can be used as a trigger to control the trace buffers. This system only makes use of the resources that are left after the initial placement and routing of the user circuit, virtually having zero area overhead. The selection of signals can be deferred till the actual time of debugging and does not require the designer to make a pre-selection during the initial circuit compilation. It significantly reduces the debug turn time required to change the set of signals that are connected to the trigger and trace unit. This is accomplished by embedding the overlay network over the initially placed-and-routed design. Whenever a new set of signals need to be observed, only the embedded network needs to re-configured.

The Quicktrace is promising enough to be one of the key technologies in its domain but is only demonstrated on a theoretical architecture with simplifying assumptions. That makes it important to demonstrate its working on a commercial architecture. This work evaluates the performance of this approach on a Xilinx Virtex-6 commercial FPGA [9] found on ML605 evaluation kit. The Virtex 6 FPGA is a realistic, commercial and complex FPGA architecture as compared with the hypothetical architecture.

The VTR CAD flow [6] is used for the realization of the proposed debug system. This CAD flow is distributed over multiple intermediate stage to synthesize, pack, place and route the verilog user circuit with respect to the FPGA architecture and is shown in Figure 1.1. It is an open source CAD flow that provides independent access to its individual stages for easy modification of its core algorithms to facilitate CAD and FPGA research whilst it does not have the capability to generate a bit-stream for actual FPGA devices. This CAD flow is modified to implement the debug system as shown in Figure 1.2 by adding two more intermediate stages making it possible to generate bit-stream with debugging instrumentation for commercial architecture.

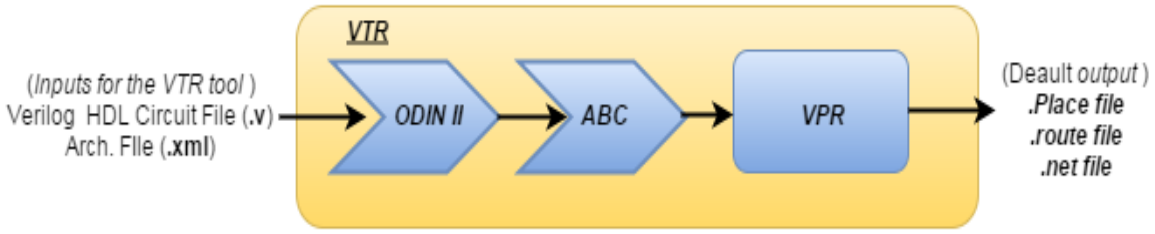


Figure 1.1: Normal VTR Flow

The output from the VPR stage is used to incrementally insert the debug instrumentation, that is, embedding the overlay network that adds new routing to the initial route solution connecting circuit signals with trace buffers while preserving initial un-instrumented routing during the QuickTrace stage. Then this debug instrumented routing is subjected to physical design rules via Xilinx DRC, and upon successful pass, bit-stream is created that can be programmed on a physical device (in this case Virtex-6 FPGA) during the bit-generation stage. The new tool flow, proposed in Figure 1.2 lever-

ages the functionality of different tools used to realize it. We further explain these tools in the background section.

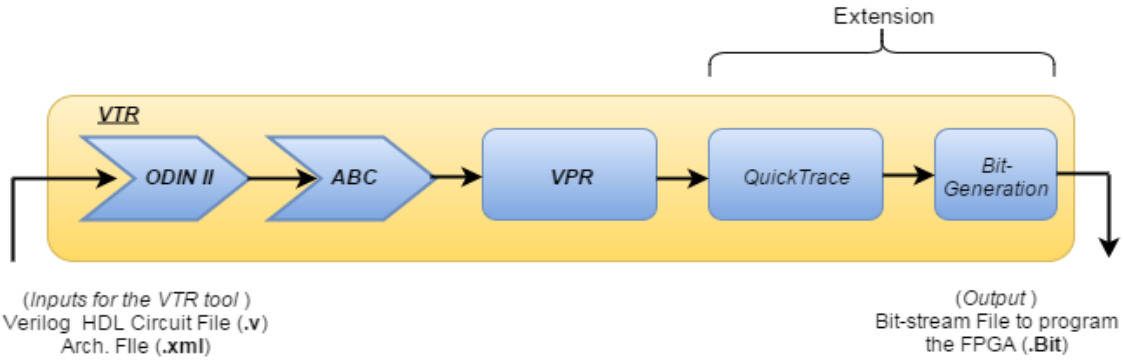


Figure 1.2: Proposed VTR Debug flow

The results, challenges and involved work of using this approach and proposed flow may differ between FPGA architectures, especially in contrast with theoretical architectures. Currently, the debug system only supports the Xilinx Virtex-6 family of devices, but it is possible to extend it for other family/generation of devices with minimal effort. Nevertheless, this could be used on any FPGA where it is possible to augment the debug instrumentation along with existing circuitry.

1.5 Thesis Overview

The remainder of the thesis is structured as follows: Chapter 2 describes the background and related work, explaining the different concepts required to understand this work such as VTR or trace-based approach, as well as related work that highlights the difference between this work and similar work. Chapter 3 explain the modeling of Virtex-6 via Architecture description language. Chapter 4 describes the implementation, explaining the complexities involved while using a commercial architecture in conjunction with VTR that limits exploration of different features present on them, an extension of VON and the proposed debug system. Further, Chapter 5 elaborates about the methodology and metrics used to perform the experiments and reports the obtained results. Chapter 6 contains the conclusion of thesis and also suggests future work with which this work can be extended and/or refined.

2 Background and Related Work

The sections below state the topics in detail that will help the reader to understand better, the work presented in this thesis. Section 2.1 describes the scan and trace-based approach of FPGA debug and advantages of the latter over former. Section 2.2 explains about the incremental compilation for FPGA CAD tools. Sections 2.3, 2.4, and 2.5 explains about the VTR and VTB tools respectively. Section 2.3 also describes the architecture description language used within the VTR CAD flow in order to model the Xilinx Virtex-6 FPGA. Rest of the sections describes about virtual overlay network and commercially available debugging tools. Section 2.7 highlights the differences between this work and other similar works. Finally, Section 2.8 concludes the chapter.

2.1 Scan and Trace Based Technique

Observability of the internal signals in a FPGA can be improved by using scan or trace-based techniques. Scan-based approach captures the state of memory elements of FPGA for observation by serially shifting it out via scan chains over an I/O pin or an interface such as JTAG. All the memory elements on the chip, that is, flip-flops and embedded memory blocks contains the values which represent the state of the FPGA. This technique relies on architectural features like device read back in FPGAs or by connecting internal flip-flops sequentially so that their data can be serially shifted out when triggered by a control signal. This can be implemented using general purpose soft logic [10], although it can provide complete visibility into the state of flip-flops in the design, it requires the circuit to be halted before scan-out with prohibitive area and delay costs. Reference [10] showed that average overhead for full scan is 84% additional area and Reference [11] reported that viewing each flip-flop using device read back can take 2 to 8 seconds. For using this technique, the circuit needs to halt for every clock cycle if we want to observe values of memory elements for every clock cycle, which immensely increases the debug time.

The trace-based technique is another method that functions by leveraging the presence of embedded memory blocks on an FPGA. It utilizes a portion of the embedded memory resources, using it as trace buffers to record a small subset of the internal circuit values during normal device operation. These trace buffers controlled by a trigger, connected to the signals to be traced are pre-inserted into the circuit before compilation. They record a window of the history as circuit operates in real-time. In FPGAs with readback capability, the data in trace-buffers can be extracted for offline analysis. This approach does not require the circuit to halt, has very less area and delay cost, provides capability to test the system with real-time stimulus over the former approach. Xilinx Chipscope Pro, Altera SignalTap II and Synopsys identify [5] [12] [13] as examples of such trace based debugging tools, also referred as trace-based IPs. These tools also have

disadvantages such as time-consuming re-compilation if the designer wishes to change the set of signals being observed, limited by the on-chip memory resources, may influence the initial placement and routing of the circuit.

This work focuses on the trace-based approach and tries to improve on the drawbacks of commercial trace IP offerings by extending the virtual overlay network for a commercial architecture.

2.2 Incremental CAD

To program a circuit onto a FPGA, different types of CAD tools are used depending on the designer's preference. Examples of such CAD tools are Xilinx ISE tool flow, Altera Quartus tool which are commercial offerings or VTR project being an open source project. The entire CAD process to implement a circuit onto a FPGA consists of the following steps [**?**]:

1. Synthesis/Logic Optimization - Performs multilevel minimization of the boolean equations to optimize area, delay or both.

2. Technology mapping - Transforms the boolean equations into a circuit of FPGA logic blocks as well as performs area or delay optimizations.

3. Packing and Placement - Packs the related logic blocks together and selecting the specific location for each logic block onto the FPGA while optimizing for the wire length.

4. Routing - Physically connecting the placed logic blocks with the routing resources such as wires, connection or switch box etc., distributed inside an FPGA to form the interconnect fabric.

On an abstract-level all these tools have synthesis, technology mapping, pack and place and routing as intermediate stages within the tool flow and are implemented by stage-specific tool-sets. Each tool is unaware of the complexity or implementation of the other and the mechanism for linking them together, but they have a cyclic dependency and there is a strong interaction between them. For the sake of simplicity, only the idea of incremental CAD flow is mentioned and not explained since it is not the focus of this work. Often a design under goes many iterations even after being implemented on the FPGA. To eliminate the need for full design recompilation, that is, end-to-end from the synthesis stage to the routing stage in case of any change, the tool needs to perform incremental design modifications instead of starting from scratch. Local changes to the design can be encapsulated as changes for a specific stage and a re-run of only that stage is needed to implement the change. This independent re-running of only specific stages is known as incremental flow, although they still need to be feed with same input file as used in default case. For example, if only the synthesis stage is required to run for modification then it will be called incremental synthesis, likewise incremental routing and/or placement. The benefit of this idea is that it lets the designer save significant amount of development time. Reference [15] explains about the challenge, benefits and

ways of achieving this idea. For the present day, tools mentioned above do implement incremental compilation at their own level of abstractions, varying from open source to vendor-specific. Figure 2.1 shows the Xilinx ISE tool and outline the different parts of the tool that can be independently used.

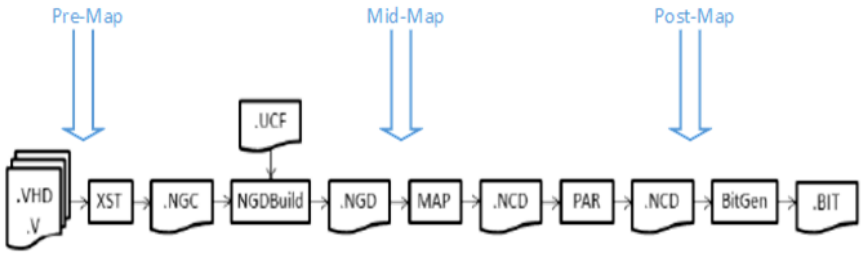


Figure 2.1: Typical Xilinx ISE flow that can be used incrementally

2.3 VTR CAD Flow

To understand the CAD flow, an initial introduction of the architecture description language and BLIF netlist is presented in the next sections and then the CAD flow is explained in the subsequent section.

2.3.1 Architecture Description Language

The FPGAs physically contain several resources on them such as I/O blocks, RAMs, complex logical blocks consisting of LUTs or flip-flops and routing infrastructure. With the increasing complexity, density and efficiency of the FPGAs, these blocks undergo change as well with every new generation of devices. For example, complex logic blocks may consist of more number of smaller components called primitives or they may have different Look-up table (LUT) configurations. The Open source CAD tools lack the ability to target complex commercial architectures. This limits the researchers, who do not have the capability to explore new avenues relative to these new architectures because of the proprietary nature of commercial CAD tools and FPGA architecture. Moreover it is not always possible to physically realize a new FPGA architecture just for experimental purposes. Hence, a language is developed that can model any type of hypothetical or real FPGA architecture, purely for research purposes.

An Architecture Description language (ADL) [16] can precisely express different types of blocks and interconnects present on a FPGA at different complexity/hierarchy level, described in extensible markup language (XML) file format. The designer can describe complex logic blocks with arbitrary internal routing structure using this language, it permits arbitrary levels of hierarchy within the logic block, that is, a logical block can be defined as a block within top level complex block and so on. It can define different configurations of a block within the architecture which can represent different functionality and relevant portion of interconnects. It also models the timing specifications of

the primitives and interconnects. This language provide the means to forward the FPGA research by letting the designer describe hypothetical architectures and experiment with new ideas. The architecture file used as input to VTR along with Verilog circuit is written using this language. More information about this language with examples can be found in reference [17].

2.3.2 Verilog-To-Routing (VTR)

The VTR tool is an open source CAD flow for synthesizing verilog circuits onto hypothetical FPGA architectures. It is a world-wide collaborative project involving multiple research groups to provide a complete, flexible, robust and open-source framework for conducting FPGA architecture & CAD research and development. The CAD flow takes a verilog hardware description of digital circuits, and a xml file describing the target architecture as it's input in the beginning and then elaborates, synthesizes, packs, places and routes the circuit as per the input architecture and performs timing & power analysis [18] . The VTR v7.0 CAD flow is the base tool upon which this work is built upon and hence the description helps the reader better understand the context.

It comprises of three core tools as shown in Figure 1.1:

- ODIN II [19] responsible for Verilog elaboration and front-end hard-block synthesis, it takes in the Verilog circuit, interprets and converts the Verilog syntax into logical netlist targeting the soft-logic on the FPGA while other constructs into 'hard logic' blocks on the FPGA and outputs a BLIF [20] netlist.

- ABC [21] is used for technology independent logic synthesis and technology mapping of logic onto LUTs and flip-flops. It takes the BLIF netlist from ODIN II as input and performs on the soft-logic part of the BLIF.

- Third and last tool in the flow is VPR [22] that performs physical synthesis and timing analysis. It takes in the BLIF netlist from the previous stage and the architecture file (.xml) as input. Afterwards, packs, places and routes the circuit. It generates three output file namely .net, .place and .route files. It performs all the physical optimization of the logic primitives into the complex logic and other blocks mentioned in the architecture file. Subsequently, it also performs the timing analysis, area and power estimation [6]. The VPR implements a timing-driven packing and routing algorithm with lot of enhancements as compared to its previous releases such as support for carry chains.

While using the VTR flow, it is possible to run individual stages and analyze the output. For example, to only use the VPR tool, the user has to create the BLIF netlist and describe logical block in the FPGA architecture description file. The user can even run the individual stages within VPR i.e. only packing, placement or routing, but needs to provide the appropriate input for that stage. The complete VTR CAD flow can be treated as an important contribution towards advancing the FPGA architecture and related CAD flow research and development.

2.4 VTB

VTR-to-Bitstream (VTB) [23] is an open source extension of the VTR CAD flow that takes a Verilog input and architecture description file of a commercial FPGA and produces a routed netlist which is converted into a valid bit-stream after running Xilinx Design rule check that can be programmed onto a Xilinx Device. The author's intention behind developing this tool was to present a unique comparison between the quality of results generated by the academic and commercial CAD flows [23], but for the purpose of this work, we will use it to generate the bit-stream for a physical Xilinx device, and also to investigate that whether the routing of debug instrumented circuit confirms with the physical design rules of Xilinx or not.

The VTR CAD flow is developed to target hypothetical FPGA architectures and hence, the VPR is capable of only producing simple routing networks relative to commercial CAD flows supporting only horizontal and vertical routing tracks symmetric across both channels. For a commercial architecture this is different which also supports diagonal and L-shaped wires. VTB bridges this gap by extracting the routing graph from Xilinx device database and stitching it with the graph produced by VPR and it is possible to do so because the VPR's router can work with routing graphs generated externally till it satisfies the graph requirements; xdlrc2vpr tool within VTB performs the previous step. VPR then normally produces .net, .place and .route files using this routing graph which is then processed by RapidSmith [24] to generate a single text-based human-readable Xilinx Design File (XDL) which captures all LUT masks taken from BLIF netlist, packing and all net connections from .net file and placement sites on FPGA from .place file. Another internal tool called route2xdl is used to add routing switches (PIPs) to the previously generated XDL. As a last step, the XDL is used to generate the NCD file via the xdl2ncd tool (part of ISE tool flow [25] which is used as input for generating the bit-stream using Bitgen tool (Again, part of ISE tool flow). The complete flow is known as VTR-to-Bit-stream [23] as it provides a path from using the packed, placed and routed circuit to generate valid bit-stream.

2.5 Virtual Overlay Network

There are several concepts which have been proposed to improve the observability within a FPGA and are mentioned in the related work. One such concept is virtual overlay network proposed by Hung and Wilton [26] that holds the potential of providing software simulator like observability for FPGA as hypothesized by the authors. The details of this virtual overlay network is explained below as it forms the base of this work. As compared with the incremental debug flow Figure 2.2, the authors have proposed a debug flow which comprises of two stages as in Figure 2.3:

- Compile time - After the normal circuit compilation, i.e., packing, placement and routing, the virtual overlay network is embedded into the un-instrumented circuit that incurs a one time overhead of embedding the network. This runtime overhead varies depending on the size of circuit.

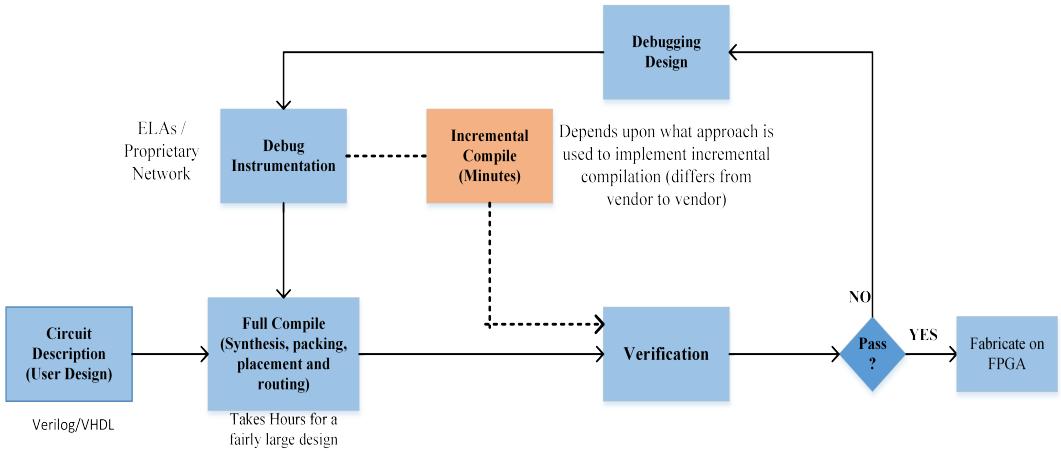


Figure 2.2: Existing incremental flow

- Debug time - For the purpose of debugging, the embedded network can be re-configured to connect the designer's choice of signals to the trace buffers for observation. The network can be re-configured several times and within minutes. The concept behind this network is explained in the rest of the section.

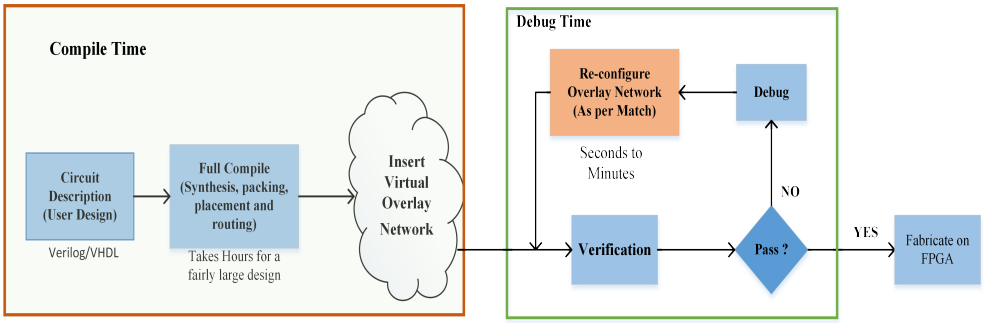


Figure 2.3: Incremental Debug flow using Virtual overlay network

The virtual overlay network in Figure 2.4 depicts multiplexs all on-chip signals to all trace-buffer pins via routing multiplexers, in contrast to a point-to-point network in between a on-chip signal source and a specific trace-buffer pin using dedicated routing multiplexers as shown in Figure 2.5. FPGAs are highly re-configurable given the presence of abundant routing multiplexers; this network is built using these multiplexers left after the initial circuit routing, at the same time reducing the area overhead of debug instrumentation. Building this overlay network is treated as a routing problem which can be represented as a routing resource graph G(V,E), where V = $V_{signals} \cup V_{routing} \cup V_{trace}$ and E = set of unused routing tracks between these vertices; $V_{signals}$ = set of all traceable

circuit signals, $V_{routing}$ = set of unused routing multiplexer and V_{trace} = set of trace-buffer inputs. For better understanding, an example is depicted in Figure 2.6. Here all the signals can be routed to either of two trace-buffer inputs for observation. Each routing multiplexer can have a fan-in of more than one. This allow the multiplexing of signals, where a designer can observe any single signal in circuit, and a limited selection of any two signal, as defined by the Cartesian product of two signals sets. For example, {A,B} x {C,D,E} = {AC, AD, AE, BC, BD, BE}. A routing solution like this can be treated as a disjoint union of trees, where each tree is rooted at a trace-buffer input and leaves being the circuit signals that it connects. Such an arrangement allow signal selections to be made for each trace-buffer input irrespective of other trace-buffer input, differentiating it from general routing resource graph and allowing more accessibility to circuit signals.

The network can be further modified as graph $G^{'}(V^{'}, E^{'})$, where $V^{'} = V_{signals} \cup V_{trace}$, and $E^{'}$ = set of edges showing the connectivity between a signal and trace pin. It is also possible for a signal to be a leaf of multiple trees while occupying few more routing multiplexers and increasing the flexibility of observing any combination of two signals, as depicted in Figure 2.7. So, to summarize the virtual overlay network is built on top of existing circuit connections, using routing multiplexers to create new connection in order to forward the signals from their source to multiple trace-buffer input pins for observation and can be configured during debug time merge with the existing routing.

2.5.1 Network Matching

During debug-time the designer's choice of signal is routed to a trace-buffer input pin. But it introduces an assignment problem, that is, which signal to connect to which trace-buffer pin, to maximize the chances of observing different combinations of signals. This assignment problem can be solved by further treating the virtual overlay network as a bipartite graph $G^{'}(U_b, V_b, E_b)$. It is a graph whose vertices can be divided into two disjoint sets independent of each other, every edge connects a vertex in one set with a vertex in

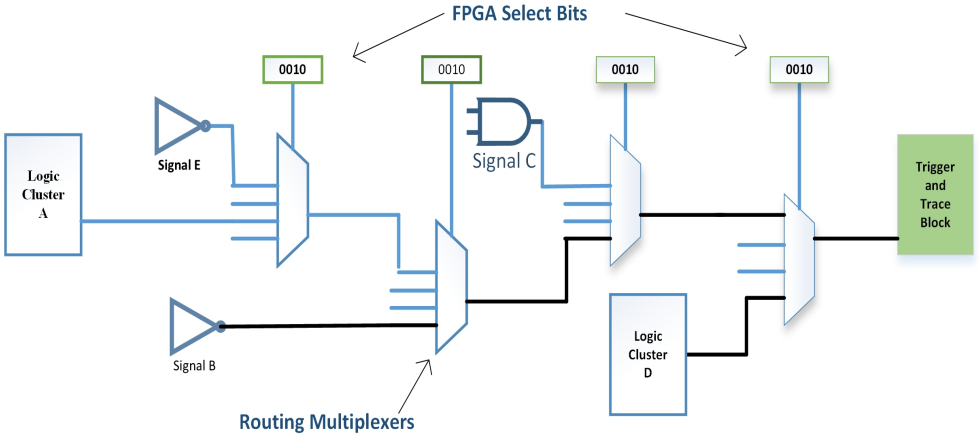


Figure 2.4: Virtual overlay network multiplexing circuit signals to all the available trace-buffers

another set and edges must not exist between the vertexes of same set. Now, substituting $U_b = V_{Signals}$ - set of all circuit signals, $V_b = V_{trace}$ - set of all trace buffer inputs and $E_b = E^{'}$ - set of edges which represent the network connectivity between two vertices. A matching of graph G_b represents a subgraph that has none of its edges sharing a common vertex. A maximum matching is the largest such subgraph that can be formed. With this property it is very convenient to decide which signal to forward to each trace pin: given that each trace pin can only support one connection, hence each node in V_{trace} must have at most one edge. The maximum number of edges that can exist is the minimum number of the cardinality of either vertex sets. Such maximum matching algorithm returns a best effort partial assignment, but optimal where the maximum number of signals possible are forwarded over the network for observation. In some scenarios, designer may wish to specifically observe some signals or trigger signals, then these preferences can be encoded by adding weights W to each edge of the bipartite graph and using a weighted version of the maximum matching algorithm, which returns largest matched subgraph possible having maximum sum of all weights on matched edges; This variant is termed as maximum weighted match algorithm. By using the above explained algorithm, select bits for each of routing multiplexers can be computed and programmed on the device using either static or dynamic-partial reconfiguration. For further details, please refer to [7].

2.6 Commercial Trace IPs

As mentioned earlier, trace based techniques utilize embedded memory on the FPGA as trace buffers to record a small subset of internal signal state during normal device operation. Almost all the trace IPs in the commercial market uses this technique, examples of them are Xilinx Chipscope Pro [12], Altera SignalTap II [5] and Synopsys Identify [13]. This section briefly explains these tools. Trace IP from Xilinx, Chipscope Pro requires the designer to choose the signals as well as the debug core to be inserted before implementation. Trigger & trace signals can be changed incrementally using FPGA editor. Although Xilinx support partial re-configuration that allows to skip a full re-compile. This feature has limitations while using the Chipscope Pro and may also require extra

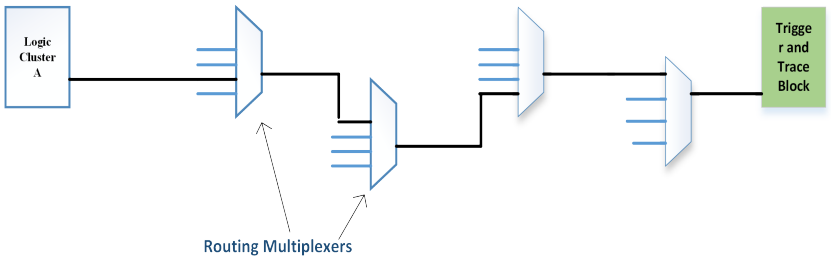


Figure 2.5: Point-to-point network using dedicated multiplexers

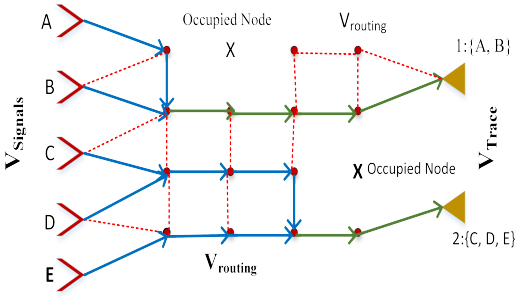


Figure 2.6: Union of signal trees: each having a trace pin as root

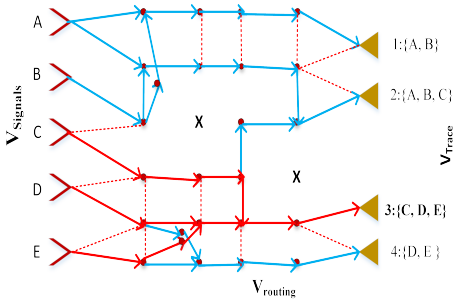


Figure 2.7: Overlay network: signal tree for trace-input 3 is highlighted in red

design effort of running a separate program.

SignalTap II from Altera tries to exploit the incremental compilation by partitioning the design and in case tries to re-compile only that partition instead of complete design. It also requires the designer to make the signal selection and core insertion before implementation but is more integrated than ChipScope. Both the tool use partitioning, core insertion and signal selection before compilation.

Synopsys Identify provides the observability in FPGA at RTL level and adds the debug instrumentation into the circuit before compilation. It gives the flexibility of changing the signal without a full re-compile. Since, the instrumentation is added before compilation, it have certain area overhead and changes the initial pattern of circuit. This work tries to demonstrate that how these limitations can be re-solved and is further explained in related work section.

2.7 Related Work

As mentioned above, the challenge of improving observability while debugging in FPGA can be overcome by using either scan or trace-based techniques. The trace-based approach has an advantage over scan-based while incremental CAD should be used that helps in avoiding time critical compilation stages. This work proposes a new incremental debug flow based on virtual overlay network that enhances the observability within FPGA. This debug flow is based on VTR CAD flow and has the ability of generating valid bit-stream for Virtex-6 devices. Incremental synthesis/routing techniques allow faster re-compilation and preserve the placement and route of the design under test. Adding debug instrumentation after the original circuit compilation avoids influence on initial placement and routing thereby preserving as much of the original solution as possible while adding debugging functionality. There is often unused logic and routing resources like embedded memory or routing multiplexer leftover after initial circuit placement on a FPGA, which can be reclaimed incrementally and used for implementing debug instrumentation. With incremental approach, full re-compilation can be avoided, allowing faster debug cycles and changes in the debug instrumentation as depicted in Figure 2.2.

Reference [4] proposes instrumenting FPGA bit-stream, with debugging hardware

to improve debugging productivity. They inserted unconnected embedded logic analyzers prior to placing and routing, and afterwards modified the bit-stream at low level to connect the ELA with trace signals. They also described that how this process can be automated using JHDL, JBits or JRoute, however it may not scale up for observing thousands of signals, requires design effort of changing bit-stream for every ELA and significant area overhead due to pre-inserting trace-buffers. Poulos, et al. [27] also proposed bit-stream modification to improve debug productivity. They connected signals of interest to the routing muxes, modifying the design prior to synthesis, muxes forwarded the signals to the FPGA I/O pins where it can be observed via external logic analyzer, and if it is wished to observe different signal. Then bit-stream modification can change what is forwarded through routing muxes. This approach does not use any trace buffer but do influence the placement and routing of initial circuit, introduces area overhead, and exposes the designer to error prone process of changing individual bits in bit-stream.

Incremental trace buffer insertion and its limitations by Hung and Wilton [3] [28] proposed how trace-buffers can be inserted into design. The trace-buffers only observe without modifying the functionality of the design under test and trace signals can be routed to any trace buffer input for observation. Reference [29] described a method to reclaim spare FPGA resources for debug instrumentation and used automated signal selection techniques [30] [31] to connect influential signals with trace buffers. Although they all differ from this work that no such concept of embedding a network was explored here but it did identify these uniques features of trace-buffers and reclaiming left-over resources. Most similar to this work is commercial tool Tektronix Certus [32], which implements a non-blocking proprietary observation network built out of general purpose logic to observe a large subset of signals, during initial circuit compilation. This network can be collapsed to observe a small subset of signal during runtime. Another product namely, Altera SignalProbe [33] uses ECO techniques to multiplex upto 256 signals to each reserved I/O pin for external analysis. This work differs in that, it does not use general purpose logic for inserting debug instrumentation and does not require pre-selection of signal.

This work expands on and demonstrates an application of the work done by Hung, et al. [7] that holds the potential of being a efficient trace IP. They implemented the concept of virtual overlay network for a hypothetical architecture and encapsulated it as a tool named Quicktrace using VTR CAD flow. It was a extension of their own work [26]. It used a hypothetical architecture with simplified assumptions. They found that it is possible to trace 80-90% signals while reclaiming left-over resources with debug turn time within minutes. They hypothesized that there was no reason why their techniques would not work on commercial FPGA devices but did not verify the same. Moreover, the results presented in the paper were not obtained with practical cases. In this work, the feasibility and performance of QuickTrace is demonstrated on a commercial Xilinx Virtex-6 FPGA and expanded to generate a valid bit-stream, so that a complete debug path using VTR CAD flow can be realized. The VTR CAD flow in conjunction with VTB tool is used to map the circuit onto an FPGA, incrementally insert the debug instrument and generate the bit-stream. With this work, it is being shown how the limitations of ELAs and observability of the FPGAs can be improved and implemented for other physical FPGAs. All the related projects were published after 2013, which

highlights that the present thesis deals with currently open research problems and recent engineering work.

2.8 Conclusions

This chapter presents the different concepts and technologies necessary to understand the rest of the thesis. It begins with a brief explanation of the scan and trace-based approach. It highlights the fact that trace-based approaches are better than scan-based approaches in terms of faster debugging time with less area overhead. Afterward, it details about the incremental CAD flow that makes it possible to independently run individual stages of the tool thereby saving compilation time. Subsequently, the VTR CAD flow and architecture description language is explained that forms the base of the debug system proposed in this thesis. VTR only understands the architecture description written in XML format of an FPGA in order to know the properties of FPGA device being and then perform tool execution. Subsequently, the concept of virtual overlay network explains that how we can embed a flexible network on top of existing place-and-routed circuit design, multiplexing circuit signals to trace buffers that can be merged together onto the existing mapping when debugging is required. The network connectivity describes about the flexibility of such a overlay network. We have used this particular concept to build the debug system.

Finally, this chapter also outlines the differences between this thesis and related/similar works. It briefly describes the commercial trace IPs as well. The commercial trace IPs requires the designer's to pre-select the trace signals, extensively uses FPGA resources and have long debug cycles. We can follow from the earlier discussion on related work that the use of overlay network for commercial architecture is unique and the debug system based on it overcomes the limitations of commercial trace IPs like Chipscope or SignalTap.

Architecture description of Virtex-6

3

This chapter describes the architectural parameters of Xilinx Virtex-6 family of devices that are translated into a XML format using architecture description file. The VTR tool is not capable of generating a bit-stream for any FPGA architecture because it does not have a device database of supported FPGA architecture (unlike other proprietary FPGA CAD tools like Quartus or Vivado) that contains detailed information about different properties and associated parameters of devices used to implement the circuit. The reason for this lack in functionality of VTR is that, its a open source tool that is mainly used for FPGA architecture and CAD research, and there is no such device database for hypothetical architectures that are used for experiments. Hence, a architecture description language is used instead to model and describe the FPGA architecture being used. The VTR takes in this description as an input, extract information about the FPGA device and perform all the operations to compile the circuit. Section 3.1 presents an overview of the Virtex-6 FPGA and the reasons for using this particular device. Sections 3.2, 3.3, and 3.4 explains the modeling of configurable logic blocks, interconnect resources and Block RAMs using this language respectively. Subsequently, Section 3.5 concludes the chapter.

3.1 Virtex-6 FPGA Overview

The Virtex-6 architecture is the preferred choice for this work because it is the only architecture supported by the VTB tool as well as the newest architecture fully supported by the RapidSmith [24] CAD tools used within VTB tool. No other tool exists that can be used as an alternative to these tools for Xilinx devices. Closed proprietary device databases and unsupported interfaces are responsible for constraining the open source tools to use these architectures. Moreover, in recent times Xilinx has stopped the support for Xilinx design language (XDL) and Native circuit description (NCD proprietary netlist format while migrating to their new tool flow called Vivado, blocking the use of their new FPGA devices for architecture or CAD research. The XDL and NCD support related to the Xilinx FPGA devices are vital if we wish to use them in academic research. We can intuitively assume that the results will not change significantly within the Virtex family devices given the architecture remain the same concerning LUTs, Block RAMs or routing infrastructure. It is worth noting that, replicating the work for latest architectures using newer tools would involve exponential amount of work given closed-source nature of involved tools, architecture databases and no support related to them by their respective vendors.

The key enabling component is a detailed architecture description file that models as many features present on the physical device as possible, the VTR takes in this file and then performs synthesis, technology mapping, packing, placement and routing in

correlation to it. It is important to understand the Virtex-6 architecture to describe it in (.xml) format using the architecture description language mentioned in section 2.3.1. VTR does not support all the features available on the physical device and hence constrains the functionality that can be derived from them. The sections below explain the design choices made for structure of different Virtex-6 tiles in conjunction with VTR constrains. It also highlights how various architectural parameters were determined.

Virtex-6 is an array or island style FPGA, consisting of an array of logic blocks and routing channels arranged in two dimensional grid of tiles. It is termed so because configurable logic blocks look like islands in a sea of routing interconnects. There are different types of tiles present on-chip such as CLB, BRAM or interconnect and tiles of the same type are typically identical. The tiles relevant to this work are presented below. Like other FPGAs, Virtex-6 devices are also configured by loading application-specific configuration data known as Bit-stream into internal memory and is generated by the BitGen [25] program, part of Xilinx ISE tool flow. Some important features supported by Virtex-6 that highlights the adaptability of this work are dynamic and partial reconfiguration and device read-back support. The reconfiguration features allows the designer to access and modify block specific configuration bits, status and control registers, even providing the flexibility to reconfigure a portion of FPGA while the rest of the logic is active, that is, live reconfiguration. Device read-back [34] enables the designer to extract and dump the configuration memory via SelectMAP, ICAP or JTAG interfaces of memory elements (LUTs, BRAMs, SRL16), internal CLB and IOB registers. There are two flavors of this feature: Readback Verify which only reads the memory elements and Readback Capture which behaves as superset of readback verify and captures internal CLB and IOB registers alongwith configuration memories. Both of the features can be used to adapt this work for live debugging.

3.2 CLB Tiles

Configurable logic blocks is the fundamental building block of a Xilinx FPGA, being the main logic resources for implementing combinational and sequential logic and consists of LUTs, flip-flops and multiplexers. Their internal structure varies with different vendors or product families. Virtex-6 CLB [35] spans over one tile and is divided into two entities called SLICEL and SLICEM (commonly referred as Slices). Each slice contains 4 LUTs, 8 flip-flops, 1 arithmetic and cary chain for SLICEL. For SLICEM - 256-bit DRAM and 128bit shift register in addition to SLICEL. Each slice implements a 6-input LUT with 1-output, fracturable into two 5-input LUT with separate output but with common inputs. The storage elements can be used as D flip-flops or latches. The output of each flip-flop (4 in case of 6-input LUT or 8 in case of 5-input LUT when the first 4 storage elements are not used as latches) is directly connected to an output pin of the slice. Each CLB is connected to a switch matrix for routing to other FPGA resources and carry chains connects vertically in a column from one slice to the one above. The output and input pin of the slices connect to the adjacent interconnect tiles. For this work, only SLICEL is modeled, as distributed RAM option implemented by SLICEM is not supported by VTR.

The CLB model in this work is made up of two logic slices, each logic slice contain

four basic logic elements where each BLE contains a 6-input LUT (fracturable into 5-input LUT), followed by two flip-flops. A bypass input (AX) used to reach either flip-flop, or to feed XADDER carry-in directly byspassing the LUT. A combinational output (A) as output O6, a sequential output (AQ) and output (AMUX) shared between secondary flip-flop, LUT outputs O6 and O5 and the COUT or sum from adder. Each slice contains one CIN and COUT connected as chain through all four BLEs. Logic slice clock enable, set/reset and wide multiplexers representing MUXF7/F8 hardened resources are not modeled because of being constrained by VTR which does not support them although architecture description language do support complex hierarchy and it is approximately possible to model the complete Virtex-6.

3.3 Interconnect Resources

The interconnect tiles such as switch or connection boxes are the primary location of the programmable routing resources that are spread throughout the FPGA. Each interconnect tile is paired with a non-interconnect tile such as CLB or IO blocks. The routing network consists of pre-fabricated wiring segments and programmable switches that are organized in horizontal and vertical routing channels and generally is a proprietary information from the FPGA vendors including other information like no. of physical tracks in routing channels, directionality of wires, type of switch and connection block etc. In Xilinx FPGAs, wire segments can be connected with each other in either channels via programmable interconnect points(PIPs). The interconnect wire segments span fixed distances known as Manhattan distances (L) ranging from L = 1,2,4 and 16(bi-dir) connecting logic blocks. The non-interconnect tiles uses the adjacent interconnect tiles to connect with resources of other tiles. For example, the output of a flip-flop can be routed to a CLB tile four blocks away using wire length of L= 2 or 4 depending upon the requirement and congestion constraints. Since, VPR is not capable of modeling L-shaped or diagonal wires used by XilinX FPGAs, a pre-processed routing graph with this feature is directly imported into VPR stage.

Architecture Parameters		**Value**
Logic Cluster Size	N	8
Lookup Table Size (Fracturable)	K	6
Inputs Per Cluster	I	56
Channel segment Length	L	1,2,4 & 16
Cluster Input flexibility	$F_{c_{in}}$	1
Cluster Output flexibility	$F_{c_{out}}$	1
BRAM data width (used)		32 bits
BRAM address width (used)		11 bits

Table 3.1: Architecture parameters used as per Xilinx Virtex-6 FPGA

The architecture parameters based on the target device are presented in Table 3.1 , where few parameters are experimentally determined. The internal structure of an

island style FPGA containing the configurable logic blocks, connection box, switch box and other routing resources is depicted in Figure 3.1. Figure 3.2 shows the internal of a connection box highlighting the cluster input/output flexibility.

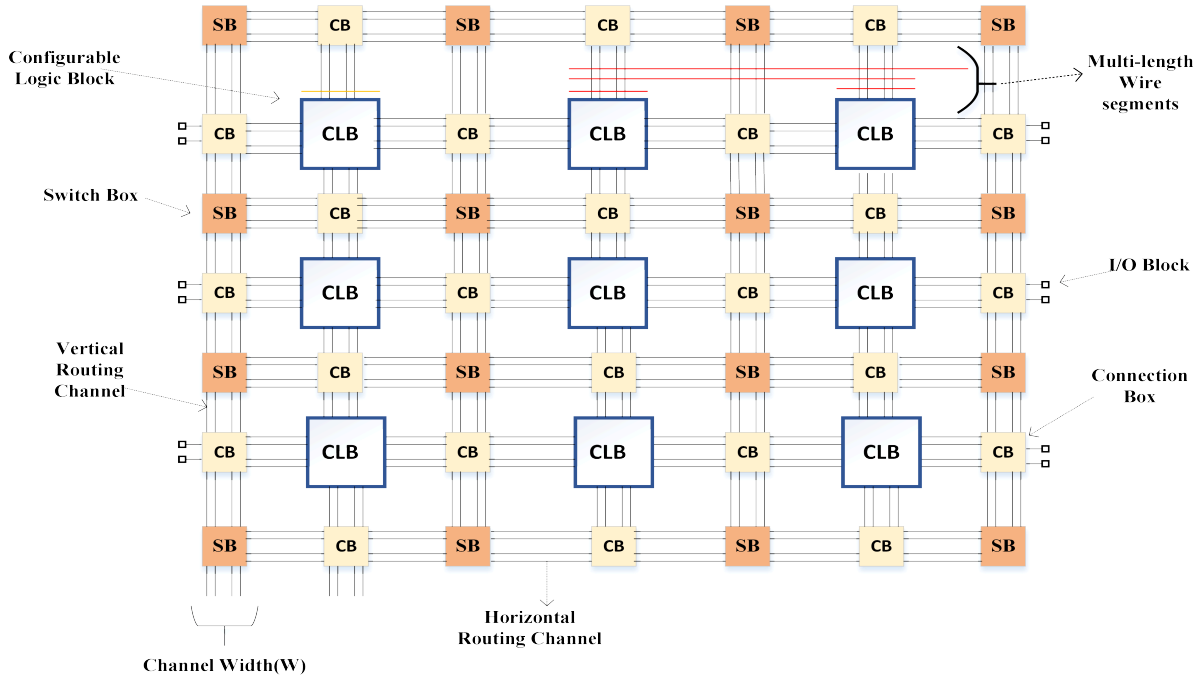
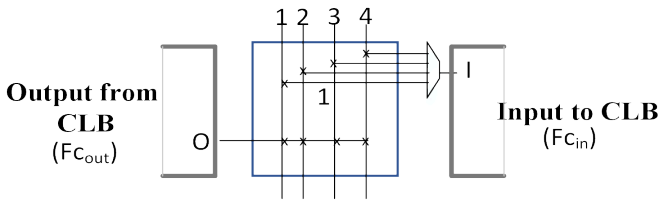


Figure 3.1: Internal structure of FPGA showing CLBs, CB, SB and routing resources



Internal structure of a connection box

Figure 3.2: Internal structure of a connection box

3.4 Block RAM

The Virtex-6 FPGA contains Block RAM tiles that can store upto 36K bits of data and can be configured as either two independently controlled 18Kb RAMs, or one 36Kb RAM [36]. The BRAMs support different aspect ratios ranging from 1bit X 32K - to - 72bit X 512, offering different combinations of data width and depth and can be cascaded to enable a deeper and wider implementation. A wider configuration (for example: 72bits X 512) will store just 512 entries while comparatively narrow configuration will have more entries. The BRAMs can operate with simple or true dual port requiring just one clock edge for read and write operations. Simple dual port means, one port is used for read-only operation and another for write-only while both port can be used simultaneously while true dual port means using both the ports for read and write operation. The BRAMs have built-in FIFO support, this dedicated logic can be used to track read and write address and status of FIFO. The FIFO has full, almost full, empty and almost empty status signals and can be configured for different widths.

Xilinx Virtex-6 Resources		**Value**
Input/Output Blocks	I/O	600
Configurable Logic Blocks (Slices)	SLICEL	37680
Digital Signal Processing Slices	DSP48E1	768
Block RAMs (36 Kb)	BRAM	416
Minimum Route Channel Width	W_{min}	18
FPGA Array Size		102 x 240

Table 3.2: Based on Xilinx Virtex-6 LXT-FF1156 Device-Package Combination

Again, the BRAM modeled for this work via architecture description language contains both 36K or 18K configuration, simple and true dual port memory and complete aspect ratio. BRAMs as hardened FIFO is not modeled for this work, as VTR do not support this configuration. The logical and routing resources present on the Virtex-6 FPGA are listed in Table 3.2.

3.5 Conclusions

This chapter describes the modeling of Virtex-6 FPGA using architecture description language for use with the VTR CAD tool. Since, VTR itself is not capable of generating bit-stream for realistic architectures, it is extended with the VTB tool to generate bit-stream for Xilinx Virtex devices. The VTB tool leverages the Rapidsmith tool that is based on XDL and provides a framework to use modern Xilinx Devices in academic research related to low-level FPGA CAD tools like the debug system presented in this thesis. The architecture description language is capable of describing any hypothetical or realistic architecture in XML format. Although, the architecture description language can describe most of the complex resources present on the Virtex-6 device, the VTR is not capable of using them. Hence, the debug system is constrained to use only those blocks on the FPGA that VTR can operate with. We experimentally determine the

architectural parameters of Virtex-6 as this information is proprietary while modeling the Virtex-6 for this work. The next chapter describes the extension of overlay network for this architecture and implementation of the proposed debug system.

4 Implementation

This chapter describes the various intricacies of the implementation and design choices that lead to the realization of this work. It starts with Section 4.1 that describes how a BRAM can be used to implement both trigger and trace functionality. Afterward, Section 4.2 explains about the extension of the QuickTrace tool for the Virtex-6 architecture and VTR 7. Subsequently, in Section 4.3 we explain the proposed debug system that will facilitate the generation of bit-stream for Virtex-6 with debugging functionality. Finally, Section 4.4 concludes the chapter. In next paragraph, we will discuss the considerations and design choices made to proceed with the implementation.

As initial step towards realizing this work, we proposed few hardware designs that tries to address the topic of improving observability while overcoming the limitations of ELAs. These hardware designs had limitations such as additional cost of hardware components, area overhead of placing them over printed circuit board while also occupying the I/O resources of the FPGA that can be used for other important purposes. So, instead of hardware alternatives, we considered an approach that do not have these limitations. The requirements for such an approach are: No hardware components to eliminate area overhead and component cost, incrementally inserting the debug instrumentation for faster-debugging cycles, possibility to extract the content of trace buffers to off-chip memory for later analysis and to remove the instrumentation when not needed.

The proposed debug system in Section 4.3 do meets these requirements and serves as a prototype that demonstrate the feasibility of such an approach. Given the closed source nature of commercial trace IPs, it was not possible to use them and hence open source VTR tool is used to realize the debug system. We chose overlay network as the core debug instrument in the proposed system because of its features like use of only spare FPGA resources left after initial circuit mapping, the ability to reconfigure the network during debug and to connect trace signals to trace-buffers with varying level of flexibility. The debug system uses the overlay network as its core debugging technique by incrementally inserting it into the design under consideration and is based on the VTR CAD tool. Since, the overlay network was only demonstrated for a hypothetical architecture, it is important to know its performance for commercial architecture. We now explain the different aspects of implementation, that is, how a block RAM can be used for both tracing the signals and triggering on the circuit logic to control the trace-buffers, extension of QuickTrace to implement overlay network for Virtex-6 and an end-to-end description of proposed debug system that encapsulates the Quicktrace to provide debugging functionality. The debug system exhibits the approach mentioned in Section 1.4 of this thesis.

4.1 Using RAM for trigger and trace

For the purpose of this work, the circuit signals routed to trace buffers are termed as trace signals and circuit signals used for trigger purposes are referred as trigger signals. As Virtex-6 FPGA contain dual-ported RAM, which allow independent read or write operation to be performed on each memory location, using separate address and data lines. A unused BRAM can be reclaimed as trace buffer and then circuit signals can be connected to its data-input lines, to store history of the signal values. In-order to have the trace buffers continuously sample the state of its input signals at every clock edge, the write-enable input of the BRAM is fixed to high. Now, the tracing can be stopped by gating the global signal for the entire design or by gating just the debug instrumentation specific clock. The latter option was not explored because the VTR does not support multi-clock designs as of know, but promises to do so in future. In this way, the trace buffers contain the history till the last clock cycle before the trigger and upto available RAM depth. For the trigger functionality, a restricted model of trigger unit that can only implement a 2 input function and can be used to check a user-defined set of signals for a specific pattern is used. The advantage offered by this trigger unit is the flexibility with which it can be implemented. Using simple dual port mode of the BRAM, the read-only port can implement trigger operations while the write-only port implements trace operations. Since, both ports share the same memory, there is a risk that trace data can overwrite read-only the trigger lookup mask and can be avoided using write-masking. Although, write masking in commercial FPGAs is only available at byte-level granularity instead of proposed bit-level masking.

The trace unit can be more efficiently implemented by using the dedicated FIFO functionality present in Virtex-6 devices, but as mentioned previously VTR does not yet support BRAM as hardened FIFO. It will be straight-forward to use it, once VTR adds support for the it. There exist different ways to implement the trigger unit as well, such as using a centrally located BRAM as a multi-output lookup table where different combinations of user signals at the address lines will cause a unique memory cell to be read and the value stored at this location can be used for starting or stopping the tracing of signals. Or a centralized trigger unit made up logic slices that implement a control signal connected to trace buffers [37].

For this work, we simulate the impact of a trigger unit and signal by reserving 32 bits in every trace-buffer during debug time. These memory locations will not be available for the tracing while the trigger signals will be assigned highest priority. This will constrain the embedded network to account for trigger functionality. Realizing a trigger unit that would fit all situations and designs is out of scope for this work and it also holds no significance for the unit to be functional or not. The essential point is that the effect of trigger signals can be accounted for overlay network in terms of its flexibility and debug-time matching of trace and trigger signals, this is illustrated more in the results chapter.

4.2 QuickTrace

The virtual overlay network concept was implemented for a hypothetical architecture and was made available as a tool called Quicktrace [7] for an initial version of VTR (VTR 1.0). This implementation used an architecture which is simplistic in nature in-terms of logic cluster, memory and interconnect resources, together representing a FPGA fabric but does not exist as a physical device and hence referred as hypothetical architecture. This architecture has a connection box flexibility of $Fc_{in} = 0.15$ and $Fc_{out} = 0.10$, where Fc means the number of wires in the routing channel that each logic pin connect to, termed as cluster flexibility. Fc_{in} and Fc_{out} stands for cluster input and output flexibility of the logic cluster's input and output pins respectively. Since it is not a physical FPGA, the VPR router determines the minimum FPGA array size and routing channel width to successfully route a circuit on this hypothetical architecture. These parameters are critical for a successful routing of the circuit as they represent the flexibility of interconnect resources. The number of physical wires/tracks in a routing channel is termed as the routing channel width of the architecture. Equal value of channel width is used for both horizontal and vertical channels. Generally, the interconnect delays are greater than the logic delays of the designed circuit. An efficient routing algorithm tries to reduce the total wiring area and critical-path timing to improve the performance of the circuit. To achieve this, the router needs interconnect information, this makes problem of routing dependent on target architecture, which varies from vendor to vendor. This kind of simplified architecture with flexible interconnect resource does not represent a realistic commercial architecture, where these flexibilities are restricted.

For this work, the Xilinx Virtex-6 xc6vlx240t-ffg1156 FPGA device, which is present on the ML605 evaluation kit is used. While targeting an existing device there is no flexibility to change the concerned parameters as it have prefabricated resources on it. Since, its a physical/real device, it is not possible to change its FPGA array size and routing channel width. The FPGA array size for this device is 102 X 240, the minimum routing channel width (experimentally determined) to successfully route a circuit is 18, Cluster Input flexibility $Fc_{in} = 1$ (i.e connects to just one wire on the routing channel) and Cluster output flexibility $Fc_{out} = 1$. The interconnect information such as: routing channel width, cluster input/output flexibility, or switch box flexibility (F_s) are proprietary information that is not available for public and is treated as a secret recipe for commercial FPGAs by their vendors. So, it is only possible to determine it experimentally or via educated guess. For channel width determination, VPR was made to route a circuit with a manually set channel width on the target architecture, the width was reduced untill the VPR failed to route the circuit. For a channel width of 18 VPR was able to route the circuit but failed when we further reduced the number and therefore we identified 18 as the minimum channel width. To save silicon area, commercial FPGAs implement connection box via multiplexers, therefore only one track can be connected to the input pin, hence $Fc_{in} = 1$ and $Fc_{out} = 1$. In Xilinx Virtex-6 architecture, wire segments in the routing fabric spans multiple length [38] [39] to provide inter-cluster connections, channel segment length (L) = 1, 2, 4, 16 is used in the Xilinx routing fabric, where wire segment having L = 1 spans through one CLB only, L= 2 spans two CLBs and so on. The tool is extended for this physical device and made to work within

the ambit of architecture's resource to determine the performance of such a network for a practical case.

A new debug system based on VTR is realized to implement the tool for Virtex-6 and generate a valid bit-stream that can be programmed on the device, this new flow is explained in Section 4.3. Quicktrace was implemented using the FPGA CAD tool VPR 6.0, that forms part of VTR 1.0. The latest version of this tool is VTR 7.0, that offers improved capabilities and support for new logic blocks but implements a new internal code structure. So, it made sense to make the tool compatible with the new version and leverage on these improvements. Therefore, we re-factored the tool and re-wrote few modules as per the new structure to first make it compatible with the latest VTR 7.0, and then use it onwards. The tool is divided into three intermediate stages: ***overlay, match and collapse***, together they form the debug instrumentation and integrated into the proposed debug system. The highlighted green part in Figure 4.1 shows the extended Quicktrace. This new These stages are briefly explained in the sections below.

4.2.1 Overlay

In this stage, the overlay network is incrementally inserted over the already packed, placed and routed circuit using the reclaimed resources not used in the initial mapping. To achieve this, the VPR [40] (part of VTR CAD flow) needs to be modified, specifically for the routing stage. Routing of a circuit on a given architecture is an NP Complete problem, which can be separated in two phases based on divide and Conquer paradigm: a global routing which divides the routing fabric into smaller regions and decides region-to-region paths for all nets while optimizing some given function and then: a detailed routing builds the actual connections using specific wiring segments for each net by searching within the fragmented regions. Detailed routing algorithms construct a directed graph representing the physical connections from the available routing resources, several algorithms for this exists such as maze router, A* Search routing or pathfinder.

VPR's router is based on a modified version of pathfinder algorithm [41], which tries to find the shortest path while allowing overuse of routing resource based on a cost function applied to all the nodes. In first iteration, all nets are routed once and costs are calculated for all nodes in the graph, in subsequent iterations nets are re-routed or ripped up until no overused resources exist. By default, VPR uses this algorithm in two different flavors to route the circuits: rout-ability driven router which optimizes the routing tracks, and Timing-driven router that optimizes the circuit speed. For this work, the latter version of router is used as the routing channel width of the architecture is fixed. To insert the overlay network, the VPR router was modified [7] to allow over-use of routing resources connecting a single trace signal to multiple RAM pins and to maintain the assumptions - that no existing circuit blocks or routes will be moved or ripped from the initial mapping. This custom version of VPR [7] is used here as well, to better preserve the original behavior of network but with modifications as per target architecture.

A normal CAD tool builds a circuit mapping where each routing resources can be used at most once, to connect one net source to one (or more) net sinks. While this network requires multiple net sources to feed a single RAM sink in multiplexed manner, so that all

the circuit signals can be multiplexed to a requested number of RAM input pins creating a flexible network. Therefore, we can ascertain the select bits of the routing multiplexers during debug and observe a particular signal. This is made possible, by modifying the pathfinder algorithm to iteratively resolve routing resources that become overused, by slowly increasing their costs so that only the most critical nets can afford them through a technique termed as negotiated congestion [41] and modifying the routing cost function that allows overuse of resources. This algorithm attempts to connect all circuit signals to the reclaimed RAM inputs, allowing routing multiplexers to be overused and terminates whenever a RAM input pin is found via a timing driven directed search strategy. During this network creation, the circuit signals have two options: either they can establish a new connection to RAM input or use an existing path. The order in which the nets are routed affects the network topology: those processed first will consume the resources that suited them most, causing other nets to work around those connections. This net ordering effect is neutralized by the modified routing cost function, allowing existing connections (of overlay network) to be ripped up or re-routed for a globally better solution. The default timing driven routing cost function used by VPR for all nodes is,

$$node\_cost = back\_cost + (1.0 \text{ x } Criticality) \text{ x } this\_node\_cost + a\_fac \text{ x } expected\_cost,$$

where back_cost is the congestion cost upto the current node, this_node_cost - cost of this node under consideration weighted by net's criticality and the expected_cost to the target scaled by an aggressiveness factor - this represents that how aggressive directed search used by the timing-driven router is [40]. To make the nodes representing routing multiplexers overused in the routing resources graph that are already being used by other connections, expected_cost was discarded and this_node_cost was discounted by node_occupancy that indicates how many overlay nets it already belongs to,

$$node\_cost = back\_cost + (1.0 \times Criticality) \times this\_node\_cost \div node\_occupancy.$$

The more nets that already passes through such a node, more the chances of it not being moved in subsequent routing iterations while a lower cost will cause the node to be removed from the heap sooner than it would have been otherwise, making the routing algorithm to follow an established connection to the RAM input. Using this modified router, first all the spare resources are determined and then a virtual overlay network on top of existing mapping is incrementally constructed once per circuit with a parameter called network connectivity(C). The term 'C' represents the target number of independent RAM inputs for each signal. Towards the end of routing phase, those connections that get congested and hence un-routable due to this parameter are discarded from the network. While in the original tool, the value of C is adaptively reduced for congested connections and if the congestion is unresolved then it is discarded from network, this was not explored here because of the complexity involved. This does not affect the result because network still have appreciably high number of connection. Once a feasible overlay network is constructed, the signals connected to each trace-pin is extracted on a text file (overlay.route) which is used in the next stage to build bipartite graph. The routes added to the circuit via this network may become the new critical path. Effect of overlay network on the critical path as well as the runtime overhead of incrementally inserting this network are discussed in the results chapter.

4.2.2 Match

Given the overlay network, that connects circuit signals to RAM pins, henceforth referred as trace pins and circuit signals as trace signals, there exist a decision making problem that which trace signal to forward to a trace pin (mentioned in section network matching). As input, this stage takes the overlay.route file which contains the information of the overlay network and a set of signals S_{trig} and S_{trace} : subset of $V_{signals}$ that represents designer's choice for trigger and trace signals. With these information, a bipartite subgraph with non-zero weight on the edges applied to only those signals that have been selected is created. A maximum weighted bipartite matching algorithm from the LEMON C++ library [42] is used to find an optimal network configuration as per the custom bipartite subgraph. The library provides efficient implementation of this algorithm, in terms of the runtime and complexity, and requires a bipartite graph with a weighted edge map as input. Once the matching is done, it is written into a text file (overlay.match) that will be used in the next stage.

4.2.3 Collapse

As a final stage, it reads the matchings from previous stage and connects the specified trace signals with the trace pins. Match file contains the information about which trace signal connects to which trace pin on a particular trace-buffer. With this information, the routing resources graph is parsed, starting at leaf node of the desired V_{signal}, moving through all $V_{routing}$ multiplexers belonging to this signal tree towards its root, V_{trace}, simultaneously setting each $V_{routing}$ multiplexer to forward the output from previous node, thus determining the select bits for routing multiplexers. The overlay network is merged onto the existing placed-and-routed mapping of the circuit with these connections and a valid VPR routing is created. The select bits of the multiplexers can be used to program the FPGA using static or dynamic configuration. We use the VPR routing to generate bit-stream and statically configure the FPGA. The next section briefly explains about the new proposed flow and how it can be utilized for faster debugging.

4.3 Proposed debug system

The VTR CAD flow is a widely used open source academic CAD flow that is used to conduct FPGA CAD and architecture research. Although this CAD flow does not route any real FPGAs, if core algorithms are modified, it can be effectively used to route real/commercial FPGA. The proposed debug system (see Figure 4.1) takes a circuit file (.v) and an architecture description file (.xml) as input, inserts debug instrumentation to observe internal FPGA signals and generates a valid bit-stream. The extended VTR flow is realized for the first time and it is possible to independently use it for any verilog design although only for Virtex devices. The system is divided into five intermediate stages, first and second stage are responsible for the synthesis and technology mapping, third performs the packing, placement and initial routing of the circuit. In the fourth stage, debug instrumentation is added, that is, embedding a virtual overlay network on top of existing placed-and-routed design that can be used for connecting the set of

signals to trace-buffers that a designer wishes to observe. In the last stage, VTB tool is used to convert the VPR routing into NCD format, typical to Xilinx, which is used as input to invoke BitGen tool from Xilinx. Output of the last stage is the bit-stream that configure/re-configures the FPGA.

As we depict in Figure 4.1, a Verilog HDL (.v) circuit and a architecture description (.xml) of Virtex-6 is the initial input to the flow. The ODIN II synthesizes the circuit and generates a .blif file (Berkeley Logic Interchange Format, an academic format for electronic netlist [20]) and ABC takes this .blif file and performs technology mapping i.e. transforming the boolean equations into a netlist of logic blocks present on the FPGA. These steps perform area and delay optimization of the circuit on FPGA and, therefore, do affect the quality of result for onward stages.

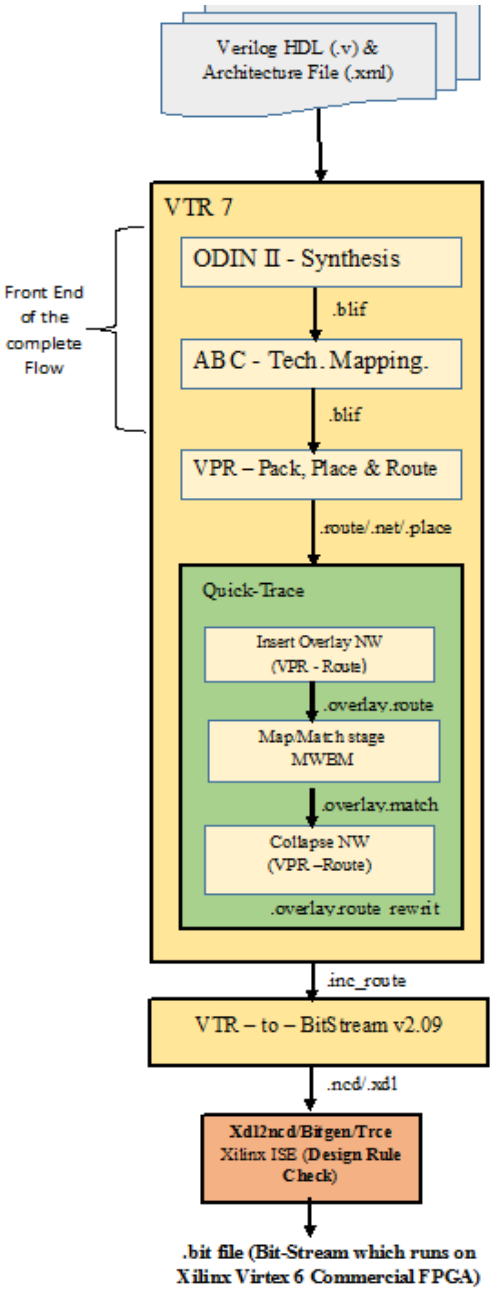


Figure 4.1: Proposed debug system

Commercial tools employ a high level of optimization at these initial stages to reduce area and delay requirements of the circuit [23] [43]. The VPR stage takes in the processed netlist from previous stage and packs, place and routes the circuit as per the Xilinx Virtex-6 architecture. The VPR generates a .route file (a text based file describing the interconnection between different resources on the FPGA). The flexibility of the VTR lies in the fact that the intermediate stages are not tied to work only with the files generated internally; hence the inputs to intermediate stages can be imported from other tools given they meet the requirements needed for that stage. Afterward, the quicktrace stage based on VPR, have three intermediate stages and inserts the debug instrumentation when required. The latter two stages of the quicktrace are used to change rapidly the set of signals to be observed and re-configure the overlay network. Collapse stage generates a ".inc_route" file that contains the new routing connections capturing the signals of interest. Since, it is possible to run in-

dividual stages of the VPR, only the route stage is invoked after choosing the new set of signals making it possible to have faster debug cycles. The incremental routing information (.inc_route file) is then passed on to the VTB tool that generates the proprietary NCD file for Xilinx Devices. As the last step, the TRCE tool from Xilinx ISE is invoked to perform Design rule checks on the VPR routing to comply with design integrity and to perform static timing analysis of the design under test. If it passes the check, then bit-stream is generated via BitGen tool to program the FPGA. It is possible to run the last two stage of the debug system as many times as required to perform the debugging task.

The uniqueness of this system is that, it have the capability to generate bit-stream with debug instrumentation that can be used to debug user designs by connecting hundred of signals to the trace-buffers while having faster debug-turn time for a commercial FPGA for the first time. Insertion of overlay network adds new routes in the circuit (that is path from a signal source to a trace buffer or trigger signals) that may increase the critical path of the circuit. The system also has a runtime overhead of inserting such a network although it is a one-time overhead. Once this network is embedded it can be re-configured as many times as necessary with out going through costly re-compilation time and hence, significantly improving the debug turn time. The performance of overlay network outline the overall efficiency of the debug system and is explained in detail in the next chapter.

4.4 Conclusions

This chapter presents the implementation of an incremental VON based debug system that tries to overcome the limitations of commercially available debug tools and improves the observability. It begins with the brief explanation about the requirements concerning the debug system, and the design choices made to ascertain its structure. The overlay network and open source VTR tool forms the core technique for the debug instrumentation and base incremental CAD flow respectively, to realize the debug flow. Afterward, we extend QuickTrace tool for Virtex-6 architecture. It is re-factored first to be compatible with the latest version of VTR and then extended for the current scenario. We also modified the core routing algorithm of the VPR tool within the VTR CAD tool to incrementally insert such a network into the design. We directly import the routing resources graph of Virtex-6 into VPR and stitch it with the internally generated routing graph, to make the VPR capable of generating valid routing for Xilinx devices with the help of VTB tool. Subsequently, we briefly explain the different stage of QuickTrace tool. Finally, we realize the proposed debug system by synchronizing the different components together to exhibit the performnce of overlay network and demonstrate the feasibility of the approach. This debug system has the potential of overcoming the limitations of ELAs while improving the observability in FPGAs.

In the next chapter we describe the methodology, test platform and metrics that will be used to measure the performance of overlay network while sketching the efficiency of debug system with detailed analysis of the associated results.

5 Results

The chapter present the findings of this work. The methodology and metrics for measuring the performance of a virtual overlay network on a real commercial architecture are described in Section 5.1. It also explains the test platform and benchmark circuits. Flexibility of the network for observing the trace signals and time taken to perform one debug cycle is the primary concern, as it is the focal point of this work. The results of the experiments are presented in Section 5.2 onwards and demonstrate the feasibility of such an instrumentation as well as its potential of being an independent trace IP. Subsequently, Section 5.8 concludes the chapter. The debug system does overcome some of the limitations of commercial tools . The runtime overhead of inserting the network, critical path, fraction of user signals that can be traced, match-stage runtime, et cetera related to incremental insertion of network and generating bit-stream are reported.

5.1 Methodology

To evaluate the performance and feasibility of the proposed debug system, six benchmark circuits are used for the experiments. The experiments investigate the different parameters related to embedding the overlay network and configuring it to trace upto all circuit signals (when there is enough trace buffer capacity) for the benchmarks. Using VTR 7.0, these six benchmarks are packed, placed and routed un-instrumented as normal onto the Virtex-6 architecture (As described in Table 3.2 and Table 3.1) to generate the baseline statistics. Table 5.1 shows the number of user signals (sequential and combinational signals) in each of the circuit available for trigger and trace input, resource usage on the target architecture and total RAM blocks that can be reclaimed as trace buffers. It also shows the number of spare RAM pin = number of trace-buffer blocks x data-width of a trace-buffer block.

Benchmark Circuit	I/Os	SLICEL	DSP48E1	**BRAM**	User Signals	Spare RAM Pins
MkPktMerge	467	58	0	14	1,002	12,864
Stereovision0	354	2,940	0	0	18,492	13,312
Stereovision1	331	7,783	564	0	46,913	13,312
BGM	289	11,459	22	0	42,485	13,312
LU32PEEng	216	24,661	64	152	115,959	8,448
Mcml	69	25,175	177	159	136,431	8,224

Table 5.1: Benchmark resource usage

There are different factors related to setting up of the experiment that are considered

since they might lead to deviation in measured parameters and are briefly discussed here. In the flow, packing algorithm is not constrained to group un-related logic elements together into the same logic cluster to minimize area. The reason being a fixed FPGA size in contrast to a minimum-sized FPGA, where densely packing is not important unless the final implementation does not fit on the device. Moreover, [44] observed that fully packing may leave the circuit un-routable because of higher peak routing demand in congested regions, given fixed channel width mitigating the same is not possible.

Placement of the circuit is performed onto the fixed array size with the objective of optimizing wire-length and timing of the circuit. During the routing of the circuit, timing driven router algorithm is used to route the circuit that focuses on optimizing the timing characteristic. Uni-directional routing tracks and Wilton switch box are employed for the routing interconnects. [**?**] shows that improvement in area and delay can be achieved if such a configuration is utilized during routing. The normally compiled un-instrumented circuits are then used to embed the overlay network using modified VPR. Once it is inserted in the circuit, match and collapse stages can be used to quickly re-configure the network with new signal selections. Same assumptions as in [7] are made: any free BRAM block can be reclaimed as trace buffers, only spare resources being used to insert the network and no existing circuit mapping will be re-routed or moved.

Since we insert the debug instrumentation incrementally after the initial circuit compilation, only gate-level signals are observable. These signals may not have a direct resemblance to the RTL or HDL level signals with which designers are mostly familiar. Using attributes related to synthesis/technology mapping tools that preserve the names of the combinational signals, it is possible to circumvent this problem (For example: A designer can specify '-Keep' for ABC tool or 's' in Xilinx ISE to preserve names of the signals). If register re-timing of the design is avoided, then both commercial and academic tools preserve the name of sequential signals. Another way of preserving signal name is using less aggressive optimization of the circuit. With sufficient information about the sequential signals, related combinational signals can be computed. Table 5.1 shows the user signals (both sequential and combinational signals) that can be used for trigger and trace inputs, resource usage on the target architecture and BRAM that can be reclaimed as trace-buffers.

5.1.1 Metrics

Parameters that will be used to report the results are Runtime overhead, critical path delay, Fraction of trace signals that can be observed, match runtime, network connectivity and debug turn overhead. VPR runtime is the time it takes to pack, place and route the circuit on a given architecture with no-instrumentation. Since, the overlay network is inserted into the circuit after initial compilation, it incurs a one-time overhead of re-routing the circuit with the added connections of the network. So, **Runtime overhead** is equal to the sum of initial VPR runtime and the time taken to just run the overlay stage.

Critical path delay represents the longest delay that a circuit path have thereby limiting the circuit's operational speed (Maximum Frequency). It is a measure of the timing characteristic of the circuit and is important here because the network insertion

potentially affects this parameter.

An interesting feature of the overlay network is that it can be realized by connecting each signal to more than one trace pin. This feature determines the flexibility of inserted debug instrumentation and is denoted by **Network Connectivity (C)** parameter, that represents the target number of unique trace pins for each user signals. The connection between each signal with different trace pins in this way can be perceived as one-to-many relationship. This feature increases the probability of observing the signal at a trace pin, even when the most preferred pins are taken by other signals due the random net ordering.

Number of signals that can be forwarded simultaneously by the overlay network given a set of signal selections to a trace buffer, that is, BRAM either for triggering or tracing is represented by the fraction of trigger and trace signals reachable out of total signals. Every time a new signal selection is made, it is used to re-configure the overlay network and forward the selections to a trace-buffer pin. The time taken to match these onto the network is referred as **match runtime**.

Once the network is re-configured, it is subjected to Design Rule Checks that make sure no physical design rule violation occurs, this is essential for bit generation, afterward it is used to generate a valid bit-stream in order to statically reconfigure the FPGA device to observe the new signals. The time taken to generate a new bit-stream is referred as **debug turn overhead**. It includes the time taken to select a new set of trigger and trace signal, re-configuring the overlay network as per this selection and entering the ISE tool flow in order to invoke and run the TRCE and BitGen tool.

These parameters and their effects on the debug system are further explained in the next sections.

5.1.2 Test Platform and Benchmark circuits

For implementing the approach and to run the experiments, a platform running on Intel(R) Core(TM) i5-4690 @ 3.5GHz with 16Gb of memory and Ubuntu operating system was used. For the results in this chapter, the data and address width of the trace buffers are kept constant at 32-bits and 11-bits respectively. Data-width of 32 signifies that 32 trace signals can be connected to each trace buffer for observation, each trace buffer having a depth of 1152. The trigger signals have higher priority than trace signals, so that the former always get routed first claiming potential pins on trace buffers for triggering. This is done in this way, because we assume that a useful trace is one, that is, captured after a triggering event. The trace signals are routed in random order. A single overlay network delivers the signals for both triggering and tracing.

The benchmarks circuits used to generate the baseline data are available as part of the VTR project [6], and represents realistic, heterogeneous and varying size circuits written in verilog HDL. It includes applications like: a Monte Carlo simulation of photons (MCML), Linear system solver using LU Decomposition method (LU32PEeng), another application using Monte Carlo simulation for financial purposes (BGM) and research projects like (stereovisionX and mkPktMerge).

As shown in Table 5.1, mkPktMerge contains the minimum number of user signals, while the MCML contains maximum number of user signals that are available for trac-

ing. Spare RAM pins represents the maximum number of pins that can be used to trigger/trace given the number of trace-buffers. In this work trace buffer width of 32 is being used, hence spare RAM pins is calculated by multiplying the number of unused BRAMs by 32 (Data-width) . StereovisionX and BGM does not use any BRAM on the FPGA unlike others and thus 100% of the available BRAMs can be reclaimed as trace-buffers. Except one, in all the benchmarks trace capacity is not enough to trace all the signals. For cases when there is not enough trace capacity, and when it is not intended to trace 100% of the signals, a random subset is chosen. Automated signal selection techniques as in [31] [29] [46] can be used to make signal selection, but taking multiple random samples as suggested in [7] will give a better understanding of the debugging technique while capturing a designer's intent to choose any desired signal. For a total of 600 data-points, i.e. using 10 random signal sets for each trace-fraction in 0.1 increments, at C = 3 and for LU32PEeng at C = 5, for a total of six benchmarks is used to report the results related to trace signals.

In most cases, average of multiple runs are presented as result. There is variation between runs due to several factors like placement of circuit blocks, selection of trigger and trace inputs, order of nets in which they are routed, platforms memory usage, number of routing iterations to resolve congestions and behavior of routing legalization heuristics involved. Iteration of router while inserting the overlay network to resolve the congestion also had effects on the critical path and number of observable user signals. To achieve consistency in results, number of routing iteration is kept constant through out all the experiments. It can be said with confidence that result presented here can be reproduced if necessary while using the same random seeds that control the randomization. Results pertaining to each parameter and their related factors are described in the next sections.

5.2 Critical-Path Delay (CPD)

The longest logic path in the circuit is considered to be the critical path as it dictates the operational frequency of the circuit, delay on this path is known as critical path delay. As stated earlier, the benchmark circuit were compiled initially with no debug instrumentation, and then the virtual overlay network is built over it, to multiplex circuit signals to the trace buffer inputs while reclaiming the spare RAM blocks. With higher values of 'C', the routing congestion increases because the network tries to connect each signal with target no. of trace pins and may not be possible to find a valid routing solution. In such scenarios, the tool employs heuristic methods to iteratively discards invalid connections for which congestion is not resolved. For the results, CPD is recorded after ten such overlay iterations to resolve the congestion for target values of C.

Table 5.2 shows the critical-path delay (in nano-seconds) before and after the insertion of overlay network for each benchmark circuits. The column titled ***Base*** refers to CPD of un-instrumented circuit, i.e. at C = 0, while rest of the columns shows CPD at other values of network connectivity (C). Since, we add the overlay network incrementally, the critical path is entirely due to the new routes added by this network.

As compared to the CPD reported in [7] that shows a maximum increase of 72% for a benchmark at C= 50 and noticeably just a 2% increase for largest benchmark, here results show a different trend. The CPD slightly increase for all benchmarks except two,

for (C <5) while sharply increasing for higher values of C, worse at C = 50, rising upto 70.94 ns from 5.28 ns for Stereovision0 benchmark. The reason for such a trend is the fact that real FPGAs contain only a limited number of routing tracks (Refer to Table 3.2) in contrast to the hypothetical architecture where the number of routing tracks can be increased arbitrarily, as used in the referenced paper. Hence, these results present the parameter in realistic scenario and shows the actual affects on the same.

Circuit	Network Connectivity (C)								
	Base	1	2	3	5	10	20	30	50
MkPktMerge	6.99	8.63	12.10	6.99	18.84	21.03	43.10	48.41	73.70
Stereovision0	5.28	5.28	8.38	12.76	21.10	26.14	37.65	59.05	70.94
Stereovision2	20.49	20.49	20.49	20.49	26.85	47.36	59.84	91.12	93.82
BGM	29.27	29.27	31.23	29.27	38.12	37.73	73.76	83.45	100.93
LU32PEEng	91.68	91.68	99.15	98.59	91.68	110.89	134.89	152.58	152.64
Mcml	95.40	95.40	95.40	95.40	96.17	135.44	149.31	165.02	NA

Table 5.2: Effect of overlay network on critical-path delay(ns) at different network connectivity (C) as compared with base delay

Interestingly, this delay remains constant at lower values of C suggesting that commercial architectures are more suited for less flexible overlay network. For all circuits, except MkPktMerge (smallest benchmark), at C = 1 there is no change in critical path delay. The MkPktMerge and Stereovision have the lowest base critical path, that remains constant at network connectivity C = 3 and 1 respectively, but increases at other values of C. For MCML (largest benchmark), this delay almost remains constant till C = 5 relative to the base value. In case of other circuits, the delay remains constant for C = 3, except LU32PEEng, for which C = 5. In rest of the cases, benchmarks suffered with a significant increase in there critical-path delay. For smallest benchmark i.e. mkPktMerge, at C = 50, an increase over 1200% while, for the biggest benchmarks (LU32PEEnf and MCML) an increase of approx 60% at C = 50. One valid reason to explain this significant increase is the fact that all the benchmarks were constrained to a fixed minimum channel width as opposed to [7] which used arbitrary routing channel width.

In scenarios where no change in the delay is observed shows that it was possible to add the overlay network without affecting the base critical path but with reduced connectivity, which is realistic given the architecture have abundant routing resources and less/no congestion in the regions that have BRAM tiles. Since, some of the benchmarks did not use any BRAMs, there was no congestion in the nearby routing regions. Benchmarks with higher base CPD are less likely to experience any increase because the new paths may afford to be longer without becoming a critical path. Hence, significantly high change is observed in circuits having low critical path and vice-versa.

A **trade-off** exist between the critical-path and flexibility of the overlay network, that should be considered. Basically, lower connectivity improves the circuit speed but that means less flexible network and vice-versa. One point worth consideration is that, the nature of benchmarks and CAD tools involved also affects this factor, because critical-path delay comprises of both logic and routing delay. These CAD tools are known to

produce slower circuits as compared to their commercial counterparts due to poor design synthesis and technology mapping [23] [43].

The debug instrumentation is added to improve the visibility within the FPGA to track and find design bugs. It is unlikely that the circuit will be run at its maximum operating frequency during debugging, maybe already limited by off-chip communication and hence, increase in delay may not be a critical issue at all. To improve this further, pipelining techniques can be used to reduce the effect of routing delay, as signal latency does not affect its observability. Utilizing this technique will cost very less or no silicon area on the commercial FPGAs given their high gate counts or they can be implemented with reclaimed resources as well.

5.3 Runtime Overhead

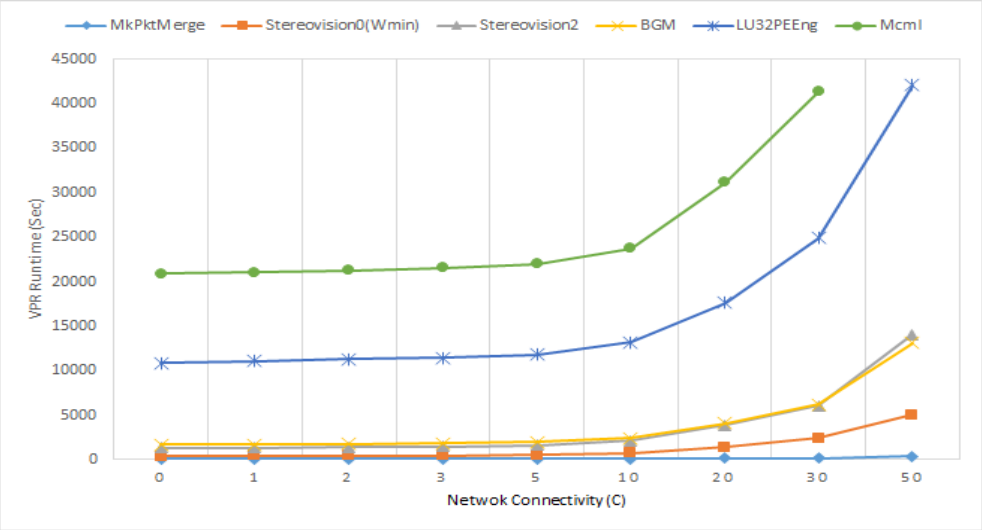


Figure 5.1: Runtime Overhead: total VPR Runtime for embedding the overlay Network; C = 0 indicates baseline runtime

The overlay network is embedded once per circuit using the spare resources during the overlay stage after the initial compilation of the circuit. This incurs an one time overhead of embedding the network in the circuit. Values greater than zero for Network connectivity C represent the total VPR runtime for default VPR stages used for circuit compilation (packing, placement and routing) plus an additional runtime of overlay stage for embedding the network while C = 0 signifies the base VPR runtime for compiling the benchmark circuit without debug system. Different values of C represents the target flexibility used to construct the network. The difference in runtime between C = 0 (base) and other values of C shows the additional overhead. It is considered as a one-time overhead because after this, the network will only be re-configured if a different set of signal needs to be traced.

Figure 5.1 shows the total VPR runtime for compiling each benchmark circuit and constructing overlay network at each C with fixed channel width, averaged over five runs.

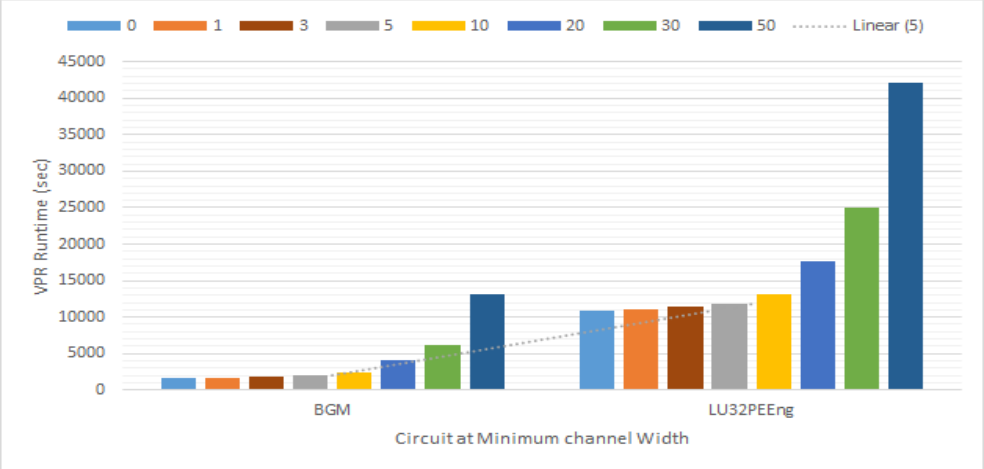


Figure 5.2: Runtime Overhead: For Bgm and LU32PEeng benchmark; C = 0 indicates baseline runtime

This factor can not be co-related to the runtime overhead reported in [7] because of the difference in FPGA architecture used. The same tool takes 5.81 and 3.02 hours (base) in this case as compared to 11.38 and 9.44 hours in case of hypothetical architecture, for Mcml and LU32PEeng respectively.

The runtime increases almost linearly with increase in the network connectivity value, interestingly the gradient rising more sharply when approaching higher values of C. The reason being increased routing search space that tool have to process in order to find the target number of connections to a trace pin for a circuit signal. It reflects the breaking point where it is no longer beneficial to stress the tool to achieve a higher network connectivity for observing the signals, as the cost of constructing such a network increases prohibitively.

On an average for C = 3, there is 17% increase in the runtime compared to baseline with no/or little change in critical path delay while rising to 54% at C=10. For the smallest benchmark (mkPktMerge), it was worse at C = 50, when it took 312.16 seconds to construct the network as compared to just 59.17 for base runtime. In case of the two largest benchmarks (LU32PEEng and Mcml), there is a moderate overhead of just 1% - 21% for C = 1 to 10 compared to base but increases sharply afterward. Figure 5.2 depicts the runtime for a moderate size and a largest size benchmark (BGM and LU32PEEeng) at different values of C. As expected, runtime increases with network connectivity due to increased stress on the tool, but both the benchmark follows the same trend, that is, increasing moderately with 13% rise at C = 5, while sharply rising towards the end.

One intuition worth consideration is that the benchmarks are compiled with fixed routing channel width given the nature of commercial architecture, if it would be possible to increase this channel width to provide positive routing slack then it might be possible to reduce the runtime overhead. Figure 5.3 shows the runtime for StereovisionO benchmark at channel width 18 and 26 (44% extra tracks) respectively. As expected,

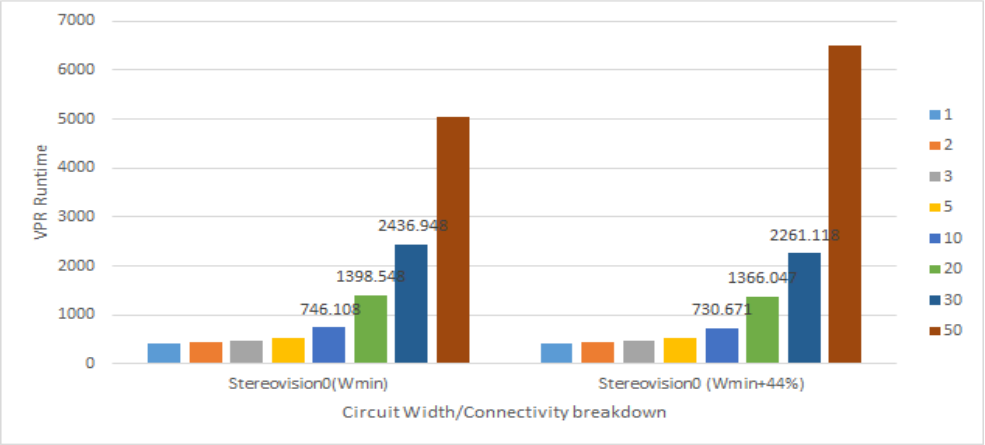


Figure 5.3: Runtime when extra routing slack is introduced: Stereovision0

there is a slight improvement in the runtime at C = 10, 20, and 30 when positive routing slack is introduced, but interestingly gets worse at C = 50 , due to extra stress on the router to find optimal routes. It is not possible to get more insight in this intuition given the closed nature of commercial tools. It also contradicts with the related work [7] that have reported about improvement in this parameter with the same intuition.

5.4 Network Connectivity

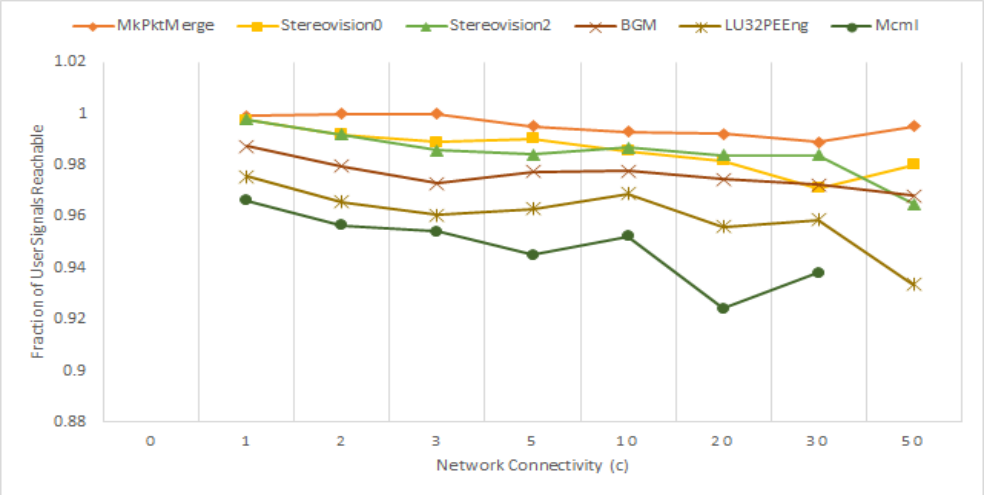


Figure 5.4: Fraction of user signals reachable at varying C ; Circuit signals normalized to their absoulute number for each benchmark

The overlay network multiplexes the circuit signals to the trace-buffer inputs, instead of creating point-to-point connection in between a signal and trace input. Network connectivity specifically means the number of independent trace-buffer inputs that each signal belongs to, given the number of trace buffer inputs and circuit signal that exist in a circuit (for example, C = 5 represents that each incrementally connected circuit signal can be routed to 5 different trace inputs thereby increasing the chance of observing it even when another signal captures the preferential trace pin). Figure 5.4 illustrates the fraction of signals reachable through the overlay network at varying C values for different benchmarks, when averaged over five runs. C = 0 indicates no instrumentation.

For the smallest benchmark it was possible to connect 100% of circuit signals at lower values of C and remained negligibly changed for other, because of abundant spare resources left after initial circuit compilation. In all other cases, increase in the number of connectivity C have little effect on the signal reachability whilst degrading. With increase in C value, the signal density at individual trace-buffer pins increases which causes routing congestion and hence the drop in signal reachability. Mcml being the largest benchmark that uses 159/416 BRAM, it was possible to reach approximately 95% of the signals, although the decline with C remains.

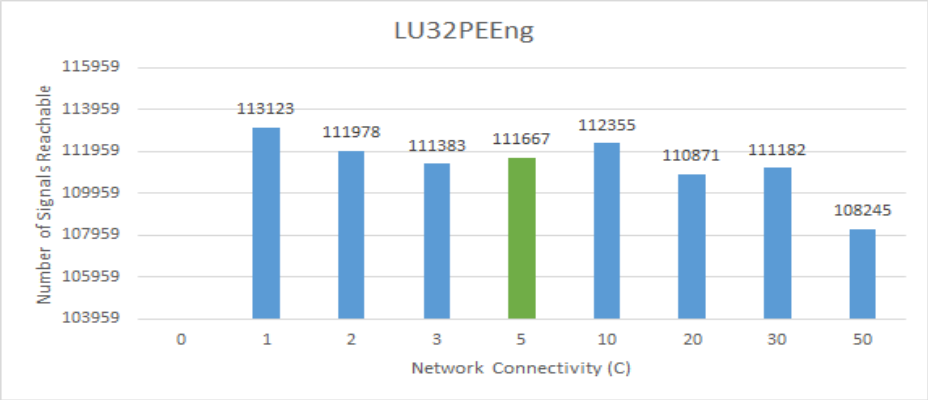


Figure 5.5: Fraction of user signals reachable for LU32PEEng at C; Green bar indicates signal reachability with no change in critical-path delay

Figure 5.5 exhibits the data for LU32PEEng benchmark, that specifically shows signal reachability at Connectivity C = 5 (green bar) where it was possible to construct an overlay network without any penalty on critical-path delay and with only 8% runtime overhead. This demonstrates the feasibility of applying this technique on commercial architectures, while compromising on the flexibility of overlay network (that is, restricting C at no more than 10) gives a solution that have fair increase in the runtime with no/little change in critical-path delay for a decent size circuit. Although, other factors also affect these results, it will be safe to assume that, implementing this technique in commercial CAD tools with full integration will be more efficient given the highly optimized nature

of the tool for there associated architectures and complete knowledge of the involved parameters.

5.5 Trace and trigger Match

Trace and trigger match represents the number of signals that can be traced or used for triggering given the trace capacity and forwarded to the trace buffers for observation via overlay network. The maximum number of signals that can be traced is directly proportional to the amount of available trace capacity. The match stage runs the maximum matching algorithm to find a primal solution on the associated weighted bipartite graph to return the number of signals that can be traced and used for triggering.

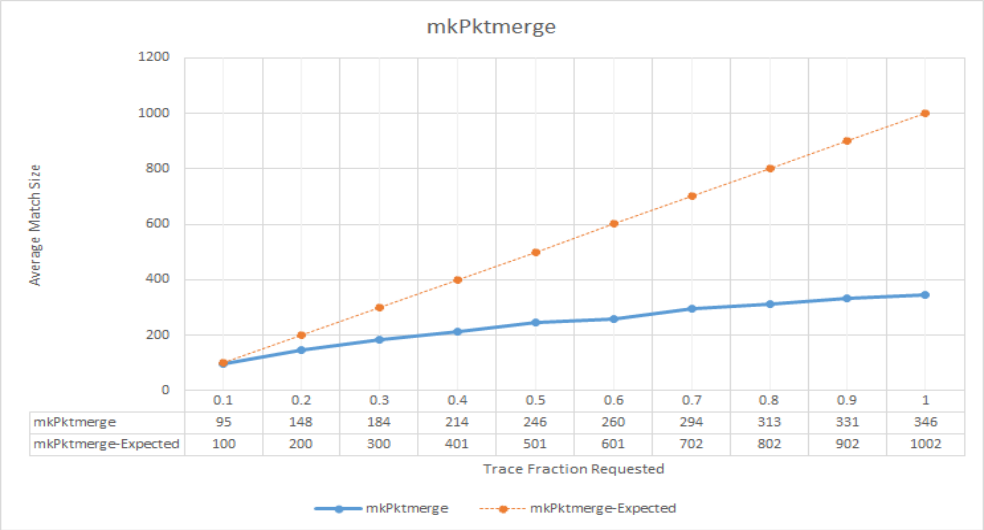


Figure 5.6: Trace match size for mkPktmerge

In all the cases, it was possible to obtain a full trigger match, that is, all the 32 signals are connected to the RAM pins. This simulates the effect of having a trigger unit as these pins are no longer available for tracing. For the trace signal, only partial match was possible. Figure 5.6 to Figure 5.11 illustrates the maximum size of trace signals that was matched compared to expected size of trace signals for a requested trace fraction. A trace fraction of 0.2 represents 20% of the available trace pins after discounting the number of pins blocked by trigger signals, that is, for example 2579 pins out of 12896 trace pins for stereovision0 benchmark. Maximum trace capacity denotes the total available spare RAM pins in a circuit. A match at different trace fraction (normalized to the available trace capacity) for every benchmark is requested and the obtained match are plotted in the referenced figures. The dotted line represent the number of signals that a designer expects at a given trace-fraction for a complete trace match while the solid lines shows number of trace signals returned by the matching algorithm that will be forwarded over the overlay network for observation. It can be observed in the figures that for all the cases, both the lines do not overlap meaning that a full match was not possible and only a partial match was returned. In Figure 5.6 (For the benchmark mkPktmerge), more than 90%, 72% and 61% of the requested trace signal can be matched at trace fraction 0.1, 0.2 and 0.3 respectively, declining to just 34% at trace fraction of 1, while for other benchmarks these number are more worse, significantly affecting the efficiency

of involved debugging technique. The reason for this behavior is explained in the next paragraph. The overlay network is re-configured with this partial match to observe the signals.

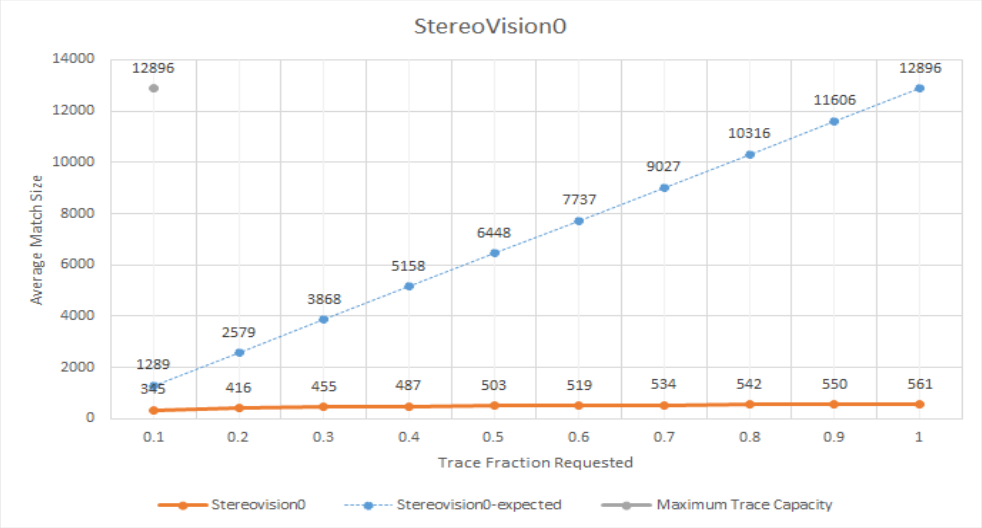


Figure 5.7: Trace match size for Stereovision0

There is a significant gap between the expected and obtained number of trace signals that can be observed for all the benchmarks in Figure 5.6 to Figure 5.11. This gap shows the limitation of the proposed debug flow. Lower the gap between expected and obtained number of trace signal matches, the more efficient proposed debug flow will be as it will be possible to observe more trace signals. Following are the reasons that tries to explain this gap:

- First, in case of all benchmark circuits, it was not possible to incrementally connect all the internal signals to the overlay network due to the routing congestion that may leave the circuit un-routable on the device and hence, these signals were discarded from the overlay network configuration.
- Second, during the packing stage local nets are absorbed within the logic cluster and these nets are unable to exit the cluster due to lack of free resources in there vicinity unlike global nets. It was also not possible to make global routes for these nets, so that they can exit the cluster via these routes towards the trace-buffers because of constraints imposed by VPR while using Virtex-6 architecture. This particular reason significantly limits the number of signals being traced. More information about the routing infrastructure in Virtex-6 may help improve it.
- Third reason being the signal density at each trace-buffer pin. It can be assumed that each pin on an average will be connected to the number of signals w.r.t network connectivity C. For example, in LU32PEeng circuit with an overlay network of C = 5, means that a trace pin will be connected to (111667 * 5) / 8184 = 68 (approx.) different signals. So, if 1 signal is forwarded to a trace-buffer input, the

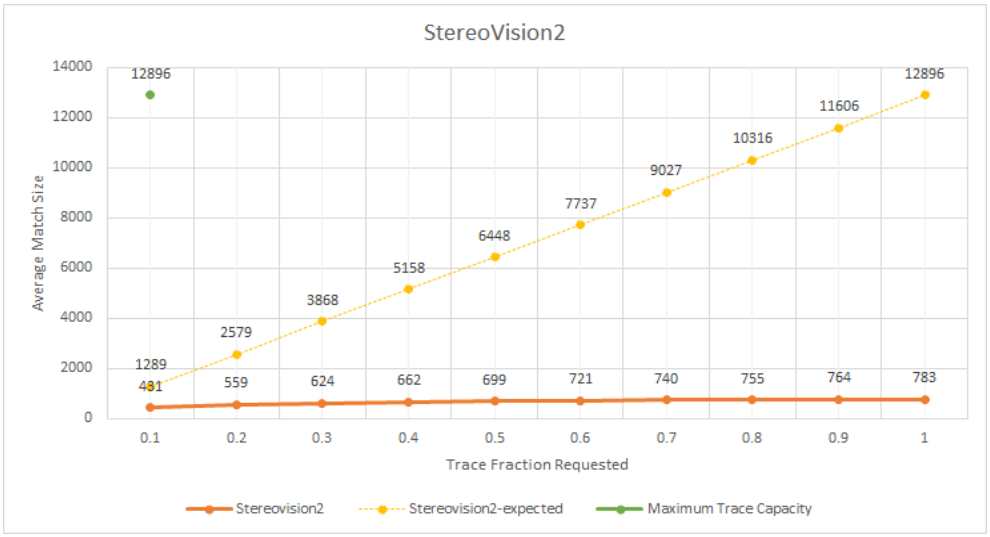


Figure 5.8: Trace match size for Stereovision2

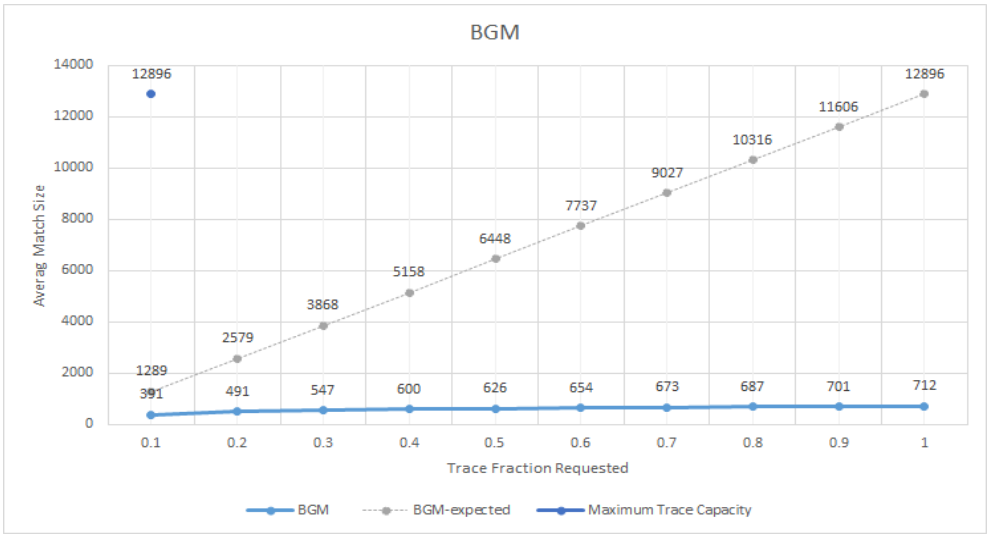


Figure 5.9: Trace match size for Bgm

rest 61 signals are blocked from reaching this preferred pin and hence either barred from being observed or routed to a comparatively less preferred trace pin. The fixed channel width and low cluster output flexibility (Fc_out) of the Virtex-6 architecture further intensifies the effect of this behavior.

Figures 5.6 to 5.11 depicts the maximum number of trace matches obtained when requested at different trace fractions for all the six benchmarks.

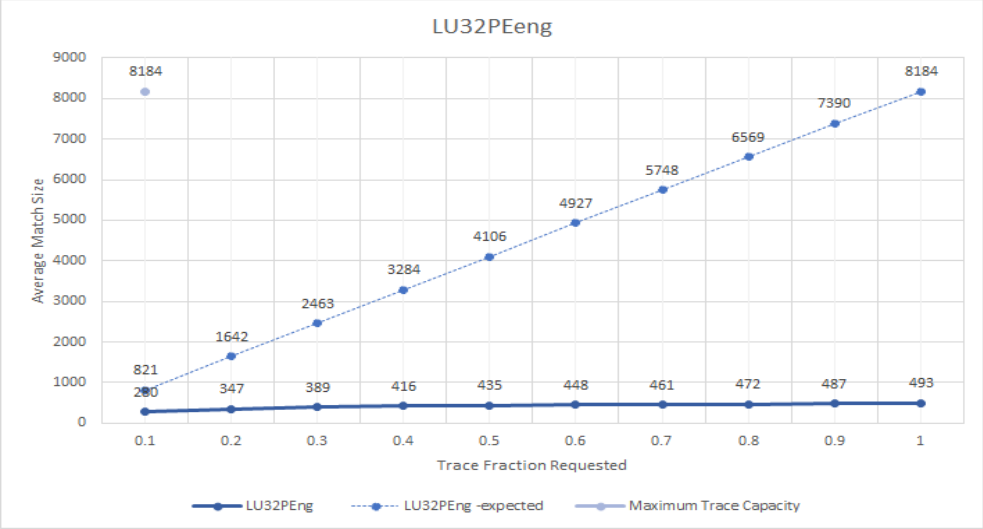


Figure 5.10: Trace match size for LU32PEeng

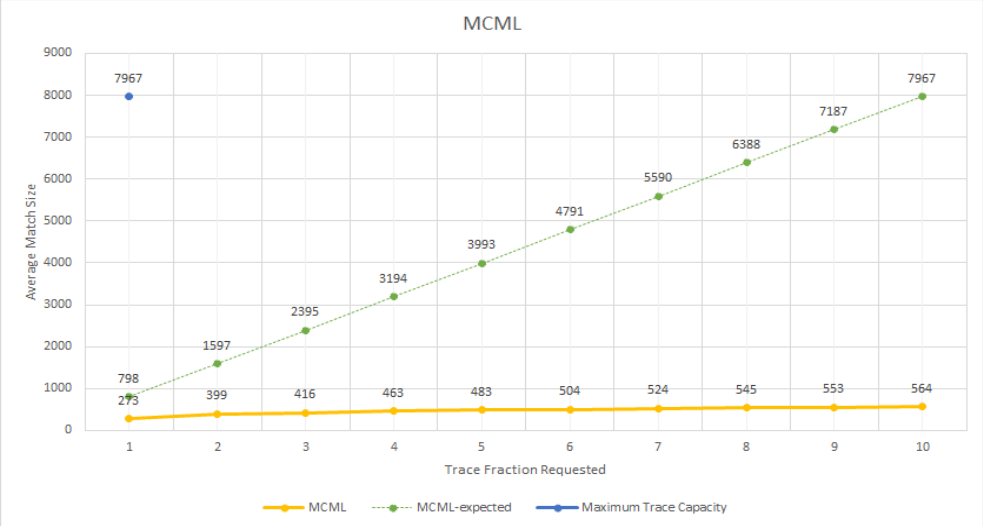


Figure 5.11: Trace match size for Mcml

5.6 Match Runtime

Figure 5.12 represents the average match runtime to select a new set of signals to be traced and to find a network configuration for the pre-embedded overlay network as per this selection for all the six benchmarks. During the match stage a blocking bipartite graph is built having trace signals and pins as vertices and connections between them as edges in order to apply network flow technique to find the new network configuration, that is, which signal will be routed to a particular trace pin. The runtime includes building such a graph and finding a maximum weighted bipartite sub-graph from this graph using Lemon C++ framework library [47] that defines the new network configuration. For all the benchmark circuits except LU32PEeng, it was possible to return a match solution in less than a second, while for LU32PEeng it was 1.13 seconds. Since, the number of trace signal slightly varies across various data-points, it can be observed in the Figure 5.12 that the runtime is constant within a range of trace fraction for different benchmarks and then shows a slight increase. The runtime and number of signals being traced seems to have a linear relation and is expected to increase with availability of more signals that can be traced.

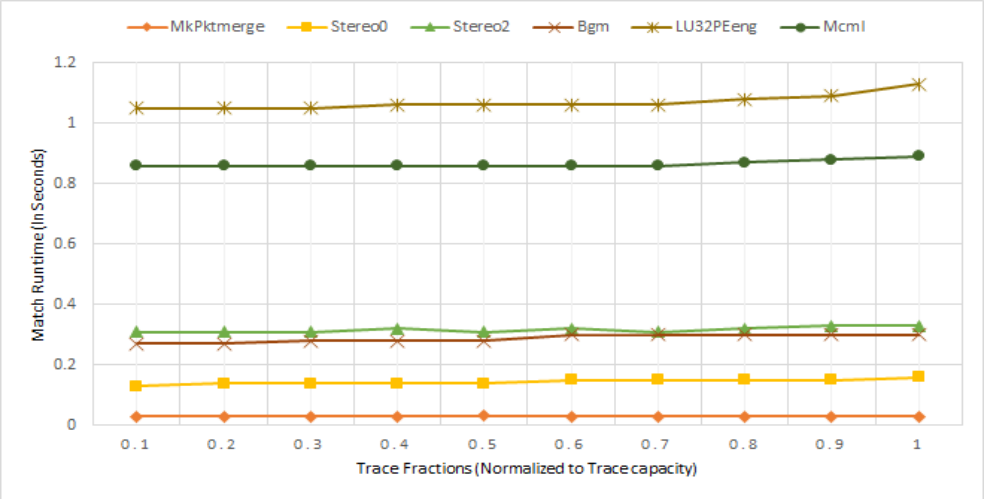


Figure 5.12: Runtime to select a different set of signal (match) and re-configuring the overlay network

5.7 Debug Turn Overhead

In the proposed debug flow, first the circuit is normally compiled, afterward overlay network is incrementally inserted onto the original mapping to trace the signals and then this network is re-configured as per the designers intent to observe an arbitrary set of circuit signal. Once a new network configuration is obtained, the circuit is incrementally routed to generate a new bit-stream that can be used to program the device and is referred as a debug turn . The debug turn overhead represents the time consumed during one such debug turn. It includes the time taken by the match and Collapse stage, conversion of VPR routing to Xilinx proprietary XDL netlist and the time taken by the TRCE and BitGen tool: to run design rule checks and perform timing analysis and generate the bit-stream respectively. Figure 5.13 depicts the flow of debug cycle.

Benchmark	Match	Collapse	TRCE	BitGen	Total
mkPktMerge	0.03	17.06	11	38.15	66.24
Stereovision0	0.16	19.38	17	48.85	85.39
Stereovision2	0.33	28.88	42	75	146.21
Bgm	0.3	33.59	36	92	161.89
LU32PEeng	1.13	282.47	88	232	603.6
Mcml	0.89	126.4	289	212	628.29

Table 5.3: Debug time overhead for one debug turn (In Seconds)

Table 5.3 shows average debug turn overhead (in seconds) for every benchmark circuit using the proposed debug flow. The column titled 'Total' represents time taken across all the stages together and rest of the columns correspond to individual timings of different stages. In all the cases, it is possible to generate a valid bit-stream along-with debug instrumentation in less than 630 seconds for LU32PEeng and Mcml benchmark, and for all circuits, in less than 162 seconds. For the smallest benchmark, mkPktMerge, this shows a saving of 39% in runtime over a full recompilation and 97% in case of largest benchmark, Mcml, thereby significantly improving a designer's productivity while debugging.

5.8 Conclusions

This chapter describes measurements done to determine the performance of overlay network and the implemented debug system. It highlights the shift in performance when a commercial architecture is used and the contribution of different factors to this performance when we use such an architecture. First, the methodology is explained. Afterwards, different performance metrics and test platform are identified. A set of six benchmark circuit that covers a wide range of FPGA designs are used to run the experiments. Subsequently, the metrics related to the performance of overlay network and the debug system are measured independently. Using the measurements, debug turn overhead parameter of the debug system is formulated and measured. Finally, we try to compare the per-

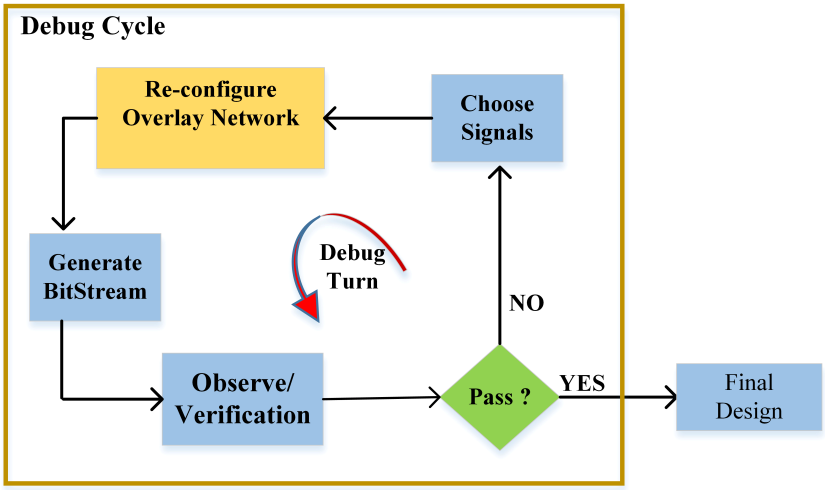


Figure 5.13: Debug-cycle: cycle may be repeated multiple times

formance of overlay network in case of both commercial and hypothetical architecture, wherever possible.

The following metrics are used to gather the results and demonstrates the performance of VON for commercial architecture: Critical path delay (CPD) of the circuit after inserting the debug instrumentation. There is no or very less change in the CPD at $C \leq 5$ for larger benchmarks but significantly increases for $C > 5$.

The runtime overhead also follows the same trend, that is, on an average 54% increase in base runtime for $C \leq 10$, but prohibitively increasing afterward.

For network connectivity, we found that it is possible to reach more that 95% of circuit signals for tracing at all network connectivity levels via overlay network. Although, it was only possible to observe few hundreds of circuit signals via trace-buffers because of constraints imposed by commercial architecture. Whilst the selection of a different set of signal and mapping them to trace-buffers is found to take less than 2 seconds as depicted by match runtime.

The debug turn overhead of the proposed system for the largest benchmark evaluated was found to be 630 seconds with major portion of this time being consumed by the ISE tool flow.

To summarize the performance of VON, it is efficient to implement the VON at lower network connectivity levels of upto $C \leq 5$, afterward the cost of using such a debug instrumentation rises prohibitively in terms of runtime overhead and CPD. We emphasize the fact that it will be beneficial, if we implement this technique with less flexibility (that is, at $C \leq 10$). The reason being, at higher connectivity levels, there is slight difference in signal reachability and fraction of signals that can be traced and we can safely assume that it will not affect the achieved observability.

These results show the capability of proposed system to generate the bit-stream for a commercial device along-with debug instrumentation, that is, a pre-embedded overlay network that can be reconfigured as many times as required, in less than 630 seconds

during each debug turn, instead of requiring a complete recompilation of the circuit during debug cycle that would normally be in hours (For example, the Mcml benchmark would take approximately 5.5 hours).

6 Conclusion

6.1 Summary

Chapter 2 presents the different concepts and technologies necessary to understand the rest of the thesis. It begins with a brief explanation of scan and trace-based approach. It highlights the fact that trace-based approaches are better than scan-based approach in terms of faster debugging with less area overhead. Afterwards, it details about the incremental CAD flow that makes it possible to independently run individual stages of the tool thereby saving compilation time. Next, the VTR CAD flow and architecture description language is explained that forms the base of the debug system proposed in this thesis. VTR only understands the architecture description written in XML format of an FPGA in order to know the properties of FPGA device being and then perform tool execution. Subsequently, the concept of virtual overlay network explains that how we can embed a flexible network on top of existing place-and-routed circuit design, multiplexing circuit signals to trace buffers that can be merged together onto the existing mapping when debugging is required. The network connectivity describes about the flexibility of such a overlay network. We have used this particular concept to build the debug system.

Finally, chapter 2 also outlines the differences between this thesis and related/similar works. It briefly describes the commercial trace IPs as well. The commercial trace IPs requires the designer's to pre-select the trace signals, extensively uses FPGA resources and have long debug cycles. We can follow from the earlier discussion on related work that the use of overlay network for commercial architecture is unique and the debug system based on it overcomes the limitations of commercial trace IPs like Chipscope or SignalTap.

Chapter 3 describes the modeling of Virtex-6 FPGA using architecture description language for use with the VTR CAD tool. Since, VTR itself is not capable of generating bit-stream for realistic architectures, it is extended with the VTB tool to generate bit-stream for Xilinx Virtex devices. The VTB tool leverages the Rapidsmith tool that is based on XDL and provides a framework to use modern Xilinx Devices in academic research related to low-level FPGA CAD tools like the debug system presented in this thesis. The architecture description language is capable of describing any hypothetical or realistic architecture in XML format. Although, the architecture description language can describe most of the complex resources present on the Virtex-6 device, the VTR is not capable of using them. Hence, the debug system is constrained to use only those blocks on the FPGA that VTR can operate with. We experimentally determine the architectural parameters of Virtex-6 as this information is proprietary while modeling the Virtex-6 for this work.

Chapter 4 presents the implementation of an incremental VON based debug system that tries to overcome the limitations of commercially available debug tools and

improves the observability. It begins with the brief explanation about the requirements concerning the debug system, and the design choices made to ascertain its structure. The overlay network and open source VTR tool forms the core technique for the debug instrumentation and base incremental CAD flow respectively, to realize the debug flow. Afterward, we extend QuickTrace tool for Virtex-6 architecture. It is re-factored first to be compatible with the latest version of VTR and then extended for the current scenario. We also modified the core routing algorithm of the VPR tool within the VTR CAD tool to incrementally insert such a network into the design. We directly import the routing resources graph of Virtex-6 into VPR and stitch it with the internally generated routing graph, to make the VPR capable of generating valid routing for Xilinx devices with the help of VTB tool. Subsequently, we briefly explain the different stage of QuickTrace tool. Finally, we realize the proposed debug system by synchronizing the different components together to exhibit the performnce of overlay network and demonstrate the feasibility of the approach. This debug system has the potential of overcoming the limitations of ELAs while improving the observability in FPGAs.

Chapter 5 describes measurements done to determine the performance of overlay network and the implemented debug system. It highlights the shift in performance when a commercial architecture is used and the contribution of different factors to this performance when we use such an architecture. First, the methodology is explained. Afterwards, different performance metrics and test platform are identified. A set of six benchmark circuit that covers a wide range of FPGA designs are used to run the experiments. Subsequently, the metrics related to the performance of overlay network and the debug system are measured independently. Using the measurements, debug turn overhead parameter of the debug system is formulated and measured. Finally, we try to compare the performance of overlay network in case of both commercial and hypothetical architecture, wherever possible.

The following metrics are used to gather the results and demonstrates the performance of VON for commercial architecture: Critical path delay (CPD) of the circuit after inserting the debug instrumentation. There is no or very less change in the CPD at $C \leq 5$ for larger benchmarks but significantly increases for $C > 5$.

The runtime overhead also follows the same trend, that is, on an average 54% increase in base runtime for $C \leq 10$, but prohibitively increasing afterwards.

For network connectivity, we found that it is possible to reach more that 95% of circuit signals for tracing at all network connectivity levels via overlay network. Although, it was only possible to observe few hundreds of circuit signals via trace-buffers because of constraints imposed by commercial architecture. Whilst the selection of a different set of signal and mapping them to trace-buffers is found to take less than 2 seconds as depicted by match runtime.

The debug turn overhead of the proposed system for the largest benchmark evaluated was found to be 630 seconds with major portion of this time being consumed by the ISE tool flow.

To summarize the performance of VON, it is efficient to implement the VON at lower network connectivity levels of upto $C \leq 5$, afterward the cost of using such a debug instrumentation rises prohibitively in terms of runtime overhead and CPD. We emphasize the fact that it will be beneficial, if we implement this technique with less

flexibility (that is, at C $\leq$ 10). The reason being, at higher connectivity levels, there is slight difference in signal reachability and fraction of signals that can be traced and we can safely assume that it will not affect the achieved observability.

These results show the capability of proposed system to generate the bit-stream for a commercial device along-with debug instrumentation, that is, a pre-embedded overlay network that can be reconfigured as many times as required, in less than 630 seconds during each debug turn, instead of requiring a complete recompilation of the circuit during debug cycle that would normally be in hours (For example, the Mcml benchmark would take approximately 5.5 hours).

6.2 Conclusion

During design verification phase, designers begin a debugging procedure to ascertain the root cause of erroneous and unexpected circuit behavior. The embedded logic analyzer utilizing trace-based approach is a common technique to implement this debug procedure where trace-buffers are inserted into the circuit and a small pre-determined subset of signals is recorded in these buffers at the event of interest. Existing academic work and commercial trace IPs that follows the concept of ELAs, requires a designer to pre-select the signals that they wish to observe in the beginning of circuit compilation, after that point-to-point connections are made in between the trace buffer and chosen circuit signal. If the designer wishes to observe another subset of signals, he/she again needs to go through lengthy compilation process prohibiting faster debug cycles. Further, these tools also uses the on-chip resources to implement debug instrumentation. Hence, we proposed a incremental VON based debug system that overcomes the limitations of ELAs and improves the observability within FPGAs.

The problem statement of this thesis was:

For a commercial FPGA architecture (like Xilinx Virtex devices), what will be the performance of a virtual overlay network (VON) based debug system?

To answer this question, the thesis had the following main goals:

- Extending and implementing the virtual overlay network concept for a commercial architecture.
- Realizing a debug system that uses the extended VON as its debugging instrument and is based on a incremental CAD tool.
- Determining the performance of the different factors related to virtual overlay network in order to demonstrate the feasibility of proposed debug system.

To achieve the goals mentioned above, a debug system for a commercial architecture is realized with the objective of improving on the limitations of commercial trace IPs by extending a promising trace-based technique [7] for a commercial architecture. It intends to incrementally insert a overlay network into the user circuit after initial placement and routing, that can be reconfigured during debug cycle as many times as the designer

needs to determine the root cause of the bug. This system employs overlay network that multiplexes all circuit signals to the trace buffers instead of point-to-point connections, uses left-over routing multiplexers rather than soft logic on FPGA unlike [32] and reclaims the remnant on-chip memory as trace-buffers. It is based on the open source VTR CAD flow and leverages the functionality of VTB tool: to take verilog HDL circuit and a architecture description file as input and generate bit-stream for the Virtex-6 device along-with debug instrumentation.

To answer the research question, we perform the experiments with a set of six benchmark circuits that evaluate the different factors related to overlay network and to determine the debug turn time of the proposed system. Since, the VON technique is used as the core of the debug system, its efficacy lies in the performance of overlay network.

The experiments have shown that for all the benchmark circuits that were investigated, it was possible to reclaim 100% of the left-over memories as trace buffers, incrementally insert the overlay network and pass the Xilinx DRC check to generate the bit-stream except one (DRC reports two design warnings for Mcml benchmark). For the overlay network, it was possible to connect 95% - 100% (for largest and smallest benchmark respectively) circuit signals to the embedded network while increasing the CAD runtime by 17% for optimal cases.

The increase in cirtical path delay is entirely due to the connections added by the embedded network. For all the benchmarks, the CPD on an average increased by 25% up till network connectivity (C) = 3 while rising on an average by 83% at C = 5. It only gets worse at increasing values of C. The ratio of increase over the base CPD is high for benchmarks that have short un-instrumented critical path delay comparative to the ones that have long delay, and hence significantly differs for every benchmark. It was also possible to embed the network without affecting the CPD at specific values of C. It also answers Question-1 presented in Section 1.3 (see Table 5.2).

In contrast to the result presented in [7], the insertion of an overlay network at higher values (>5) of network connectivity C, to build a more flexible network, significantly increases the critical path delay of instrumented circuit and runtime overhead of the system. This indicates that a trade-off exist between the flexibility of network and other equally important metrics like CPD or runtime, in case of commercial architecture.

Interestingly at lower values of C $\leq$5, for all except one benchmark, there is a slight increase of 0 - 17% in critical path delay and a maximum of 36% increase in runtime, while having a signal reachability of approximately 97.7%. This signifies the suitability of network as debug instrumentation at lower connectivity levels, that is, a less flexible network. It is worthwhile to note that using a less flexible network does not affect the debugging capability much in terms of signal reachability. There is a slight difference of 0.1% in signal reachability at different values of C. This outlines the answer for Question-2. It also shows that how commercial architecture constrains the network with fixed channel width and cluster input/output flexibility. Due to the reasons described in Section 4.5, the proposed debug system is limited by the number of signals that can be traced simultaneously, however it is possible to improve this parameter if more information about the architecture involved is made available and/or VTR tool is extended to support the complex routing resources of commercial devices.

While there is substantial compile time overhead of inserting the overlay network

once for every circuit. The embedded network can be re-configured to connect designer's choice of trace signals and generate new bit-stream to re-program the device in less than 630 seconds, as investigated for largest benchmark (Mcml), throughly improving the debug time overhead. The proposed system improves the debug time overhead on an average by 90% for all benchmarks except mkPktmerge that shows an improvement of 37%, required to perform a debug cycle as compared with full re-compilation to observe a new set of signal. This runtime saving during the debug cycle addresses Question-3.

The proposed debug system overcomes the limitations of commercial trace IPs in following ways:

- Deferring the signal selection process to debug time.
- Only utilizing routing multiplexers left after initial place-and-routed design, that is, virtually no area overhead.
- Faster debug cycle time thereby improving designer's productivity and providing more time to extensively test the design.

Like other commercial tarce IPs, this system is also limited by the on-chip memory and basically depends on the actual memory resources of FPGA device used. If there is no free memory block then it will not be possible to use them as trace-buffers. So, it can be concluded that the on-chip memory is not a limitation to ELAs, and should be interpreted as normal resources on the FPGA having no influence on the design of debugging system.

As compared with existing approaches: The approach presented in this work tries to provide a concise end-to-end solution for enhancing observability in FPGAs. It reduces the debug turn time to 630 seconds compared to a full re-compile time of 5.5 hours for the largest benchmark and 66 seconds from 1.8 minutes for the smallest benchmark, without affecting maximum operating frequency (in optimal cases). This work can be encapsulated as a tool and made available for direct use. While other academic work required the user to interact with underlying complexity making it difficult to follow their technique and only demonstrated part of the process.

The *contribution of this work* is that, it demonstrates the performance of overlay network for a realistic architecture and proposes a debug system based on VTR CAD tool that overcomes the drawbacks of ELAs providing faster debug cycles. This approach serves as a prototype to exhibit the potential of such a concept that can be further developed into a full-suite debugging tool.

There are several ways in which the proposed debug system and overlay network itself can be used or extended to further improve the observability inside FPGA's and are discussed in the next section.

6.3 Future Work

The following suggestions for the future work are categorized in implementation enhancements and research opportunities.

6.3.1 Implementation enhancements

- The proposed debug system and overlay network should be investigated from the perspective of making them more adaptable to the commercial devices, so that they can be used generically with least effort and works with different architectures from several vendors.

- The system is restricted by the number of signals that can be traced simultaneously, it will be interesting to further explore that how it can be improved while having the same constraints. In future, if more insights are available into the routing infrastructure of physical devices and support for them in VTR CAD flow, then it will be straightforward to improve this functionality.

6.3.2 Research opportunities

- An interesting research avenue is the application of this approach onto ZUMA architecture [48] [49] in conjunction with the intricacies of virtual overlay network. These architectures implement Virtual FPGAs, referred as FPGA overlays, that is, an open source, cross-compatible embedded FPGA architecture which is superimposed on top of a physical FPGA, and termed as FPGA-on-an-FPGA. It's a trending technology and is gaining traction in the FPGA research community. It will be fair to assume that application of proposed debug technique onto Virtual FPGA's will not be limited by the constraints imposed by the physical devices. Hence it may prove to be cross-compatible across different architectures while still preserving its very own performance. Upon further integration even new design CAD flows can be developed.

- The application of this approach in combination to a data capture module may also prove beneficial in scenarios where the designer's intent is to extract the FPGA's internal state at different reference points in time, to have data dumps for further analysis.

To conclude, it is possible to improve on the limitations of commercial trace IPs, at the same time increasing the observability within FPGAs and it is shown that how debug productivity can be improved during FPGA design and prototyping. Moreover, these techniques can be extended for other applications to advance research in related domains.

Bibliography

[1] H. Foster, "The 2014 Wilson Research Group Functional Verification Study." [Online]. Available: https://blogs.mentor.com/verificationhorizons/blog/2015/01/21/prologue-the-2014-wilson-research-group-functional-verification-study/

[2] S. Asaad, R. Bellofatto, B. Brezzo, C. Haymes, M. Kapur, B. Parker, T. Roewer, P. Saha, T. Takken, and J. Tierno, "A Cycle-accurate, Cycle-reproducible Multi-FPGA System for Accelerating Multi-core Processor Simulation," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 153–162. [Online]. Available: http://doi.acm.org/10.1145/2145694.2145720

[3] E. Hung and S. Wilton, "Incremental Trace-buffer Insertion for FPGA Debug," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 4, pp. 850–863, April 2014.

[4] P. Graham, B. Nelson, and B. Hutchings, "Instrumenting Bitstreams for Debugging FPGA Circuits," in *Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on*, March 2001, pp. 41–50.

[5] A. Corporation, "Quartus II Handbook Version 12.1 Volume 3." [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/qts/archives/quartusii_handbook_121.pdf

[6] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 2, pp. 6:1–6:30, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2617593

[7] E. Hung and S. J. E. Wilton, "Accelerating FPGA Debug: Increasing Visibility Using A Runtime Reconfigurable Observation And Triggering Network," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 19, no. 2, pp. 14:1–14:23, Mar. 2014. [Online]. Available: http://doi.acm.org/10.1145/2566668

[8] Xilinx, "Configuration And Readback of Virtex FPGAs Using JTAG boundary-scan." [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp139.pdf

[9] "Virtex 6 FPGA Datasheet." [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds152.pdf

[10] T. Wheeler, P. Graham, B. Nelson, and B. Hutchings, "Using Design-level Scan To Improve FPGA Design Observability And Controllability For Functional Verification," in *Field-Programmable Logic and Applications*, ser. Lecture Notes in Computer Science, G. Brebner and R. Woods, Eds.

Springer Berlin Heidelberg, 2001, vol. 2147, pp. 483–492. [Online]. Available: http://dx.doi.org/10.1007/3-540-44687-7_50

[11] Y. Iskander, C. Patterson, and S. Craven, "Improved Abstractions And Turnaround Time For FPGA Design Validation And Debug," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, Sept 2011, pp. 518–523.

[12] Xilinx, "Chipscope Pro Software And Cores, User Guide." [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_4/chipscope_pro_sw_cores_ug029.pdf

[13] Synopsys, "Identify, Simulator Like Visibility Into Hardware Debug." [Online]. Available: http://www.synopsys.com/tools/implementation/FPGAimplementation/capsulemodule/identify_ds.pdf

[14] D. F. G. Prado, "Tutorial on FPGA Routing," 2006. [Online]. Available: http://sisbib.unmsm.edu.pe/BibVirtualdata/publicaciones/electronica/n17_2006/a04.pdf

[15] O. Coudert, J. Cong, S. Malik, and M. Sarrafzadeh, "Incremental CAD," in *Computer Aided Design, 2000. ICCAD-2000. IEEE/ACM International Conference on*, Nov 2000, pp. 236–243.

[16] J. Luu, J. H. Anderson, and J. S. Rose, "Architecture Description And Packing For Logic Blocks With Hierarchy, Modes And Complex Interconnect," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '11. New York, NY, USA: ACM, 2011, pp. 227–236. [Online]. Available: http://doi.acm.org/10.1145/1950413.1950457

[17] "VPR 6.0 Beta Architecture Description Language." [Online]. Available: http://www.eecg.utoronto.ca/vpr/arch_language.html

[18] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, "The VTR Project: Architecture And CAD For FPGAs From Verilog To Routing," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 77–86. [Online]. Available: http://doi.acm.org/10.1145/2145694.2145708

[19] P. Jamieson, K. Kent, F. Gharibian, and L. Shannon, "Odin II - An Open-source Verilog HDL Synthesis Tool For CAD Research," in *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*, May 2010, pp. 149–156.

[20] "Berkeley Logic Interchange Format (BLIF)." [Online]. Available: https://www.ece.cmu.edu/~ee760/760docs/blif.pdf

[21] A. M. et al, "ABC: A System For Sequential Synthesis And Verification." [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/

[22] J. Luu, J. H. Anderson, and J. S. Rose, "Architecture Description And Packing For Logic Blocks With Hierarchy, Modes And Complex Interconnect," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '11. New York, NY, USA: ACM, 2011, pp. 227–236. [Online]. Available: http://doi.acm.org/10.1145/1950413.1950457

[23] E. Hung, "Mind The (Synthesis) Gap: Examining Where Academic FPGA Tools Lag Behind Industry," in *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*, Sept 2015, pp. 1–4.

[24] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "Rapidsmith: Do-It-Yourself CAD Tools For Xilinx FPGAs," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, Sept 2011, pp. 349–355.

[25] "Xilinx Command Line Tools User Guide." [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/devref.pdf

[26] E. Hung and S. J. Wilton, "Towards Simulator-Like Observability For FPGAs: A Virtual Overlay Network For Trace-Buffers," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '13. New York, NY, USA: ACM, 2013, pp. 19–28. [Online]. Available: http://doi.acm.org/10.1145/2435264.2435272

[27] Z. Poulos, Y.-S. Yang, J. Anderson, A. Veneris, and B. Le, "Leveraging Reconfigurability To Raise Productivity In FPGA Functional Debug," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, March 2012, pp. 292–295.

[28] E. Hung and S. Wilton, "Limitations Of Incremental Signal-Tracing For FPGA Debug," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, Aug 2012, pp. 49–56.

[29] ——, "Speculative Debug Insertion For FPGAs," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, Sept 2011, pp. 524–531.

[30] H. Ko and N. Nicolici, "Algorithms For State Restoration And Trace-Signal Selection For Data Acquisition In Silicon Debug," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 2, pp. 285–297, Feb 2009.

[31] X. Liu and Q. Xu, "On Signal Selection For Visibility Enhancement In Trace-Based Post-Silicon Validation," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 31, no. 8, pp. 1263–1274, Aug 2012.

[32] Tektronix, "Certus Debug Suite." [Online]. Available: http://www.tek.com/sites/tek.com/files/media/media/resources/Certus_Debug_Suite_Datasheet_54W-28030-1_4.pdf

[33] Altera, "About Signalprobe." [Online]. Available: http://quartushelp.altera.com/14.0/mergedProjects/program/sipro/comp_intro_signalprobe.htm

[34] "Virtex-6 FPGA Configuration." [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug360.pdf

[35] "Virtex 6 FPGA CLB User Guide." [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug364.pdf

[36] "Virtex 6 FPGA Memory Resources User Guide." [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug363.pdf

[37] B. Hutchings and J. Keeley, "Rapid Post-Map Insertion Of Embedded Logic Analyzers For Xilinx FPGAs," in *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, May 2014, pp. 72–79.

[38] D. Gomez and M. Ciesielski, "A Tutorial On FPGA Routing." Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, USA, 2008. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.9313&rep=rep1&type=pdf

[39] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, and M. French, "Torc: Towards An Open-Source Tool Flow," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '11. New York, NY, USA: ACM, 2011, pp. 41–44. [Online]. Available: http://doi.acm.org/10.1145/1950413.1950425

[40] "VPR User Manual 7.0." [Online]. Available: https://github.com/verilog-to-routing/vtr-verilog-to-routing/blob/master/vpr/VPR_User_Manual_7.0.pdf

[41] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," in *Proceedings of the 1995 ACM Third International Symposium on Field-programmable Gate Arrays*, ser. FPGA '95. New York, NY, USA: ACM, 1995, pp. 111–117. [Online]. Available: http://doi.acm.org/10.1145/201310.201328

[42] "Lemon C++ Graph Library." [Online]. Available: https://lemon.cs.elte.hu/trac/lemon

[43] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Timing-driven titan: Enabling large benchmarks and exploring the gap between academic and commercial cad," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 2, pp. 10:1–10:18, Mar. 2015. [Online]. Available: http://doi.acm.org/10.1145/2629579

[44] E. Hung, F. Eslami, and S. Wilton, "Escaping The Academic Sandbox: Realizing VPR circuits On Xilinx Devices," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, April 2013, pp. 45–52.

[45] U. Farooq, Z. Marrakchi, and H. Mehrez, "FPGA Architectures: An Overview," in *Tree-based Heterogeneous FPGA Architectures.* Springer New York, 2012, pp. 7–48. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-3594-5_2

[46] H. Ko and N. Nicolici, "Algorithms For State Restoration And Trace-Signal Selection For Data Acquisition In Silicon Debug," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 2, pp. 285–297, Feb 2009.

[47] B. Dezs, A. Jüttner, and P. Kovács, "Lemon - An Open Source C++ Graph Template Library," *Electron. Notes Theor. Comput. Sci.*, vol. 264, no. 5, pp. 23–45, Jul. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.entcs.2011.06.003

[48] A. Brant and G. Lemieux, "Zuma: An Open FPGA Overlay Architecture," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, April 2012, pp. 93–96.

[49] T. Wiersema, A. Bockhorn, and M. Platzner, "Embedding FPGA Overlays Into Configurable Systems-On-Chip: Reconos meets zuma," in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, Dec 2014, pp. 1–6.

Biography

Roshan Kumar Gupta is a computer scientist with strong interest in embedded software, computer architecture, hardware design, and technology management. His hobbies are reading about world economics and biking. He completed his Bachelor's degree in Electronics and Communication track in the year 2011, from a reputed institution in India. From 2010 to 2011, Roshan was the president of IEEE student chapter at SIT, India. He also worked for an year in the IT industry as a software engineer. Afterward, from 2013 he continued his studies at TU Delft, working towards a Master's degree in Embedded Systems. In 2014, Roshan performed an internship at ASML in Veldhoven, working on diagnostics tool that aims to reduce the downtime of their lithography machine. He will return to ASML in full-time position in 2016.