Yosys+Odin-II: The Odin-II Partial Mapper with Yosys Coarse-grained Netlists in VTR

by

Seyed Alireza Damghani

Bachelor with Honours of Computer Science, University of Tehran, 2018

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor:	Kenneth B. Kent, Ph.D., Computer Science
Examining Board:	Gerhard Dueck, Ph.D., Computer Science, Chair
	Saqib Hakak, Ph.D., Computer Science
	Richard Tervo, Ph.D., Electrical and Computer Engineering

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

November, 2021

© Seyed Alireza Damghani, 2021

Abstract

Verilog-to-routing (VTR) provides users with an entire flow from the Verilog circuit description to a final FPGA programming configuration. The VTR front-end interface for Verilog compilation, Odin-II, lacks support for the Verilog-2005 standard and SystemVerilog. However, Odin-II provides complex partial mapping for balancing soft logic and hard blocks such as embedded multipliers, adders, and memories in FPGA architectures. Yosys, an open framework for Register Transfer Level (RTL) synthesis, provides extensive support for HDLs. However, the Yosys flow forces the user to decide the discrete circuit implementation manually. The approach taken by Yosys is to map all discrete components into available hard blocks or to explode them in low-level logic when not available. This thesis proposes improving device utilization and simplifying the flow by automating hard logic decisions with architecture awareness. Yosys, as a front-end HDL elaborator, in combination with the Odin-II partial mapper, is added to the VTR flow.

Acknowledgements

First and foremost, I would like to thank my family, especially my father, who have supported me for all the challenging journeys in my life. Without their support, I wouldn't achieve such accomplishments, but I couldn't be where I am currently. My father was always a great mental supporter for me, who helped me grow an ethical, kind, and proper mindset. I wish I could pay back at least a small portion of what he has done for me during my life.

Second, I would like to thank my supervisor, Dr. Kenneth Kent, for his kind and supportive behaviour during my Master's journey. I can definitely say I have felt he is my friend who backs me up in most academic and real-life problems. Someone, who starts a new life far away from their family, should be lucky to have such a caring friend and a supportive supervisor.

Next, I would like to thank all VTR organization members for providing a warm and friendly environment for all contributors. Mainly, I would like to thank Dr. Vaughn Betz, who shared his extensive FPGA knowledge with all members to improve the quality of work and transfer his valuable knowledge to all students. Moreover, I want to thank Google, especially Tim Ansell, for providing financial support for my Master's study.

Last but not least, I would like to thank Stephen Mackay for sharing his extensive literature knowledge and help to produce professional documents. Absolutely, your guidance helped to enhance the quality of our literature.

Table of Contents

Abstract ii

Acknowledgments iii

Table of Contents iv

List of Tables vii

List of Figures viii

Abbreviations xi

1 Introduction 1

1.1 Verilog-to-Routing 2

1.2 Odin-II Synthesis and Elaboration CAD Tool 2

1.3 Yosys Synthesis Suite 4

1.4 Contribution 5

2 Background 7

2.1 Digital Circuits 7

2.1.1 System Level 7

2.1.2 High Level 8

2.1.3 Behavioural Level 9

2.1.4 Register Transfer Level 9

2.1.5 Logic Gate Level 10

2.1.6 Physical Gate Level . . . 10
2.1.7 Switch Level . . . 11
2.2 Electronic Design Automation . . . 11
2.3 FPGA Architecture . . . 12
2.4 Logic Synthesis and Technology Mapping . . . 13
2.5 Netlist . . . 14
2.5.1 Berkeley Logic Interchange Format . . . 14
2.6 Odin-II Partial Mapping . . . 16

3 Design **18**
3.1 BLIF I/O Interface . . . 19
3.2 BLIF Elaboration . . . 21
3.2.1 Cell Transformation . . . 21
3.2.2 Complex Logic Cells . . . 22
3.2.2.1 Data Flip-flops . . . 22
3.2.2.2 Multipliers . . . 23
3.2.2.3 Division and Modulo . . . 23
3.2.2.4 Exponentiation . . . 27
3.2.3 Memory Blocks . . . 30
3.2.4 Optimization . . . 31
3.3 Partial Mapping . . . 32
3.3.1 MUX_2 . . . 33
3.3.2 Multiport n-bits Multiplexer . . . 33
3.3.3 Constant-Variable Shift . . . 34

4 Yosys+Odin-II **35**
4.1 External Execution of Yosys . . . 35
4.2 Embedded Yosys API . . . 38

5 Results and Analysis **41**

5.1 Yosys+Odin-II QoR in VTR 43

5.2 Yosys QoR in VTR 47

5.3 Yosys+Odin-II QoR in VTR (Adjusted DSP Usage) 50

5.4 Frequency 54

6 Conclusions and Future Directions **56**

6.1 Future Work 58

Bibliography **69**

A Micro Benchmarks **70**

B FIR Benchmarks **75**

C Yosys+VTR Synthesis Script **80**

Vita

List of Tables

5.1 VTR Modern-Style Flagship Architecture Characteristics 42
5.2 Overview of Synthesizers' Performance over the Heavy Netlist Primitive VTR Benchmarks (geomean over 10 seeds, normalised to Odin-II) 42
5.3 VTR Benchmarks using Yosys+Odin-II Front-end in the VTR Flow (Relative to the VTR Baseline). 44
5.4 VTR Benchmarks using Yosys Front-end in the VTR Flow (Relative to the VTR Baseline). 49
5.5 VTR Benchmarks using Yosys+Odin-II Front-end in the VTR Flow with Multplier Ratio of 50% (Relative to Yosys+Odin-II QoR Experiment). 52

A.1 Micro Benchmarks using Yosys+Odin-II Front-end in the VTR Flow (Relative to the VTR Baseline) 71
A.2 Micro Benchmarks using Yosys Front-end in the VTR Flow (Relative to the VTR Baseline). 72

B.1 FIR Benchmarks using Yosys+Odin-II Front-end in the VTR Flow (Relative to the VTR Baseline) 76
B.2 FIR Benchmarks using Yosys Front-end in the VTR Flow (Relative to the VTR Baseline) 77

List of Figures

1.1 VTR Flow. 3

2.1 Synthesis Types along with Abstraction Levels [1]. 8

2.2 Simple Digital Circuit Represented in BLIF. 15

3.1 Yosys+Odin-II Synthesis Flow. 20

3.2 DFF with Enable Signal (DFFE). 24

3.3 DFF with Synchronous Reset (SDFF). 24

3.4 DFF with Enable Signal and Synchronous Reset (SDFFE). 24

3.5 DFF with Enable Signal prior to Synchronous Reset (SDFFCE) (Enable should be active to be able to reset). 24

3.6 DFF with Set (VCC) and Clear (GND) Signals (DFFSR). 24

3.7 DFF with Set (VCC), Clear (GND) and Enable Signals (DFFSRE). 24

3.8 Combinational Division Network [2]. 25

3.9 Combinational Division Network - CR Cell Structure [2]. 26

3.10 Combinational Division Network - CS Cell Structure [2]. 26

3.11 Constant Exponentiation Circuit. 27

3.12 Variable Exponentiation Circuit. 28

3.13 Yosys Internal Memory Block. 30

3.14 Single Port RAM. 31

3.15 Dual Port RAM. 31

3.16 8-bit Right Barrel Shifter using Multiplexers. 34

5.1 Netlist Primitives for VTR Benchmarks with all Front-ends. 45
5.2 CLBs for VTR Benchmarks with all Front-ends. 45
5.3 W_*min* for VTR Benchmarks with all Front-ends. 45
5.4 Routed Wire Lengths for VTR Benchmarks with all Front-ends. . . . 45
5.5 VTR Flow Sectional Runtime for VTR Benchmarks with Yosys+Odin Front-end 46
5.6 DSP Blocks Trade-off Percentage by Yosys+Odin-II Partial Mapper on VTR Benchmarks. Only Benchmarks With DSP Blocks are Included. 53
5.7 The VTR Benchmarks Frequency Comparison. (Normalised to the VTR Baseline) 54

6.1 Odin-II Plugin Integration for Yosys. 58

A.1 The Micro Benchmarks Frequency Comparison. Benchmarks Resulted in Errors or an Empty Netlists are Removed. (Normalised to the VTR Baseline). 73
A.2 Netlist Primitives for Micro Benchmarks with all Front-ends. 74
A.3 CLBs for Micro Benchmarks with all Front-ends. 74
A.4 W_*min* for Micro Benchmarks with all Front-ends. 74
A.5 Routed Wire Lengths for Micro Benchmarks with all Front-ends. . . 74
A.6 VTR Flow Sectional Runtime for Micro Benchmarks with all Front-ends. 74

B.1 The FIR Benchmarks Frequency Comparison. (Normalised to the VTR Baseline). 78
B.2 Netlist Primitives for FIR Benchmarks with all Front-ends. 79
B.3 CLBs for FIR Benchmarks with all Front-ends. 79
B.4 W_*min* for FIR Benchmarks with all Front-ends. 79

B.5 Routed Wire Lengths for FIR Benchmarks with all Front-ends. . . . 79
B.6 VTR Sectional Runtime for FIR Benchmarks with all Front-ends. . . 79

Abbreviations

AIG	And-Inverter-Graph
$ASIC$	Application-Specific Integrated Circuit
AST	Abstract Syntax Tree
BDD	Binary-Decision-Diagram
$BLIF$	Berkeley Logic Interchange Format
$BRAM$	Block Random Access Memory
CB	Connection Block
CAD	Computer-Aided Design
CLB	Configuration Logic Block
CPD	Critical Path Delay
$CSLA$	Carry Select Adder
$DPRAM$	Dual Port RAM
DSP	Digital Signal Processor
EDA	Electronic Design Automation
ESL	Electronic System Level
$FOSS$	Free and Open-source Software
$FPGA$	Field Programmable Gate Array
FSM	Finite State Machine
HDL	Hardware Description Language
IC	Integrated Circuit
LUT	Look-Up Table
QoR	Quality of Results
RCA	Ripple Carry Adder
ROM	Read-Only Memory
$RR-Graph$	Routing Resource Graph
RTL	Register Transfer Level
$SPRAM$	Single Port RAM
SB	Switch Block
$VHDL$	VHSIC Hardware Description Language
$VHSIC$	Very-High-Speed Integrated Circuit
VPR	Versatile Place and Route
VTR	Verilog-to-Routing
$YOSYS$	Yosys Open SYnthesis Suite

Chapter 1

Introduction

Field Programming Gate Arrays (FPGAs) are integrated circuits (IC) that can be programmed after fabrication. Specific architectural details may vary, but fundamentally an FPGA consists of configurable logic blocks (CLBs) that can be connected via routing channels, composed of connection blocks (CBs) and switch blocks (SBs), to I/O blocks, executing the task described by the digital circuit. CLBs are typically composed of look-up tables (LUTs) and registers [3, 4, 5]. A K-LUT can represent any K-input boolean function by storing the truth table of the function. In the FPGA context, placement and routing are referred to as programming and connecting the logic blocks to optimally deploy the digital circuit logic. FPGAs outperform Application-Specific Integrated Circuits (ASICs) in lowering design and testing costs along with providing a low-cost mass production of post-fabrication configurable chips. Furthermore, FPGAs, as a platform for high-performance computations, provide compelling flexibility and highly reconfigurable hardware [3]. This has led to ubiquitous usage of FPGAs in a wide range of application domains and real-world tasks, such as machine learning [6, 7, 8]. FPGAs and their infrastructure technology are subject to rapid development due to the high demands of application domains. As a result, FPGA architectures must evolve and optimize their efficiency for target

technologies in such application domains.

1.1 Verilog-to-Routing

Verilog-to-Routing (VTR) is an open-source Computer-Aided Design (CAD) tool that provides users with the entire flow from Verilog circuit description to FPGA configuration [9]. Not only is VTR considered as a baseline for novel CAD algorithm evaluation, but it generates various FPGA configurations targeting details of FPGA architectures [10]. In fact, VTR allows engineers to experiment with hypothetical FPGA designs while extracting comprehensive design statistics. As shown in Figure 1.1, VTR is a composite of three CAD tools:

- **Odin-II**: Verilog elaboration, synthesis, and partial technology mapping.
- **ABC**: Netlist logic reduction and LUT mapping.
- **VPR**: Packing, placement, routing and timing analysis.

1.2 Odin-II Synthesis and Elaboration CAD Tool

Odin-II is the front-end interface of VTR for Verilog elaboration and logic synthesis. Essentially, Odin-II acts as a Verilog compiler[1] in the VTR flow, while it performs digital circuit optimization as part of generating a Berkley Logic Interchange Format (BLIF) output. Odin-II currently suffers from the lack of full Verilog language coverage support, such as *repeat*, *edge* and *forever* keywords. However, its strength is in providing a complex partial mapping [11].

Moreover, Odin-II's partial mapper is capable of generating the desired device footprint using a genetic algorithm (GA) in the soft logic instantiation. The genetic

[1]This process is recognized as elaboration

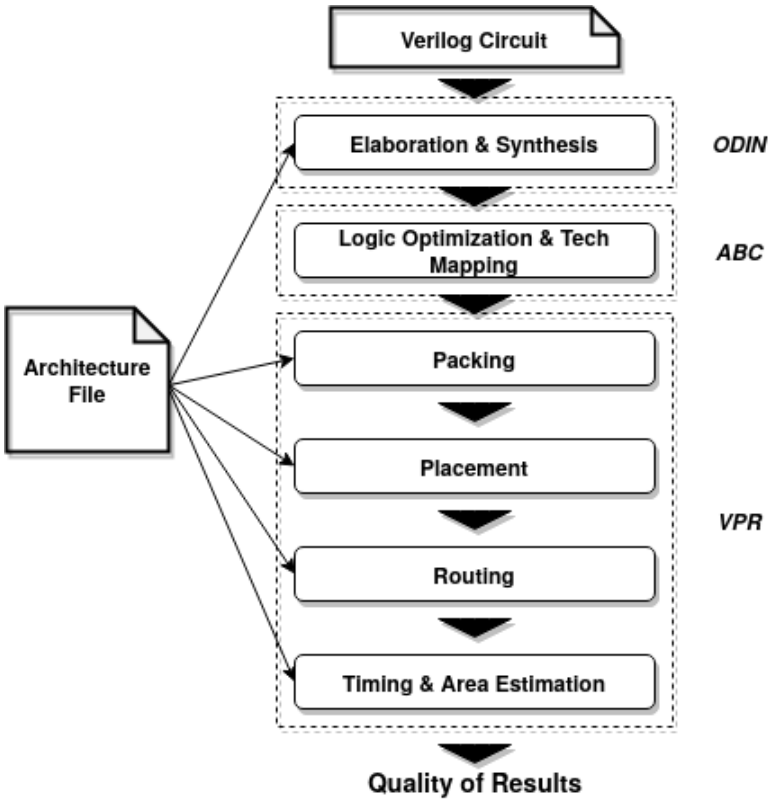


Figure 1.1: VTR Flow.

algorithm allows the synthesis tool to explore the different soft-logic implementations of high-level netlist blocks. The added value of the GA approach is to let the user specify the footprint of the final circuit, while it is driven from the user's goals of area and timing. To date, the Odin-II partial mapper GA has only been deployed for addition operations. Although adders are small circuits, they have an omnipresence in digital circuit design. Ripple Carry Adder (RCA), the smaller and slower adder architecture and Carry Select Adder (CSLA), the larger and faster one, were utilized in the experiments. The Odin-II GA-based partial mapper proved that the circuit changes align with the user's goals with a negligible extension in the Odin II execution time [12]. Further information concerning the GA-based partial mapper is discussed in Section 2.6.

Odin-II can also provide hard/soft logic trade-off decisions, named mixing optimization, that can be initialized before partial mapping [13]. The optimization can be

configured to be performed on either a portion of total hard blocks of an operation or use exactly the number specified by an input parameter. To date, the mixing optimization is fully implemented for multipliers, while the addition operation has been recently developed [14]. The hard block optimization decision is made based on a value calculated from the top to bottom depth-first traversal (DFS) of the netlist, initiating from the potential hard block node. The DFS function increments the maximum of children, while in the case of facing leaf nodes, the value is considered as 1. The approach originated from the assumption that the node count is a correlated metric to the critical path delay design [15].

1.3 Yosys Synthesis Suite

Targeting novel and dynamic FPGA configurations, new and complex digital circuit designs are increasing accordingly [16, 17, 18]. As a result, Hardware Description Languages (HDLs) and logic synthesis tools play a crucial role as the front-end interface of CAD tools. Yosys is an open-source synthesis suite with extensive Verilog-2005 standard support [19]. Whereas most commercial synthesis tools are closed source, Yosys offers a flexible and open-interface synthesis process. For instance, the platform for coarse-grained logic development is provided by this flexibility, while having access to an open synthesis process can be used as the basis for developing synthesis algorithms.

The synthesis defines a complex process used to interpret the HDL into a digital circuit [20]. In other words, the synthesis translates hardware behavioural HDL into soft logic and FPGA architectural hard blocks, such as arithmetic units or memory blocks [21]. Technically, the synthesis complexity originates from the optimization of logic networks or hard/soft logic inference. The consequence of this complexity is a competitive research area, including several contributions of essential techniques

and features to logic synthesis, for instance, And-Inverter Graph (AIG)-based optimization offered by Berkeley ABC [22, 23], and minimizing area to improve runtime and standard cell mapping by Berkeley MVSIS [24, 25].

The Yosys synthesis flow generates logically equivalent netlists of a digital design in an iterative process. During the flow, the synthesis process repeatedly utilizes external tools, such as combinational logic minimization or fine-grain technology mapping, provided by Berkeley ABC, for optimization purposes [1]. While considerable valuable data is lost in these phases, the Yosys flow also loses opportunities for logic inference decisions. In other words, the flow forces the user to decide discrete circuit implementations manually. All discrete components are then mapped into the available hard blocks or entirely exploded into soft logic, ignoring the device capacity.

1.4 Contribution

In this thesis, we specifically aim to extract Odin-II's partial mapping stage. The combination of Yosys and Odin-II synthesizers not only improves the language support of the Odin-II synthesizer, but Yosys could also benefit from the combination of hard and soft logic implementations. Furthermore, to provide the technology mapping flow of the Yosys+Odin-II synthesizer with the extensive support of logical circuits, we add the complete instantiation of the Odin-II partial mapper unsupported logical components, such as modulo and division operations.

In the first phase, Yosys as an HDL elaborator is added to the VTR flow. Using a Tcl script, including Yosys generic pre-synthesis commands, Yosys reads the Verilog files. After resolving the hierarchical relations in the digital designs, it performs high-level elaboration, such as translating process to netlist components, resolving finite state machines (FSM), and detecting memory blocks. Ultimately, Yosys generates a

coarse-grained netlist in BLIF.

Odin-II BLIF reader, with slight changes that have been made in this thesis, reads the coarse-grained netlist components and creates a netlist from the Yosys output BLIF file. Once the entire netlist is created, the newly added stage, called BLIF elaboration, elaborates the netlist components and makes them compatible with the Odin-II partial mapper. Technically, the instantiation of the Odin-II partial mapper unsupported logic is added in this phase to keep the Odin-II synthesis flow untouched. Afterwards, the Odin-II partial mapper performs the partial technology mapping, including logic optimizations and hard/soft logic inference based on the VTR architecture files. It is worth noting that the combination of Odin-II BLIF elaborator and partial mapper provides the new technology mapping stage in the new Yosys+Odin-II synthesizer.

Our contributions include but are not limited to: creating the most compatible Yosys coarse-grained synthesis script for Odin-II partial mapping flow; providing the ability to connect these two synthesizers; making the Yosys generated coarse-grained netlist compatible with the Odin-II partial mapper; adding support for Odin-II partial mapper unsupported logic, such as modulo and division; performing possible optimizations for Yosys elaborated digital circuit; and integrating the Yosys and Yosys+Odin-II synthesizers in the VTR flow.

Chapter 2

Background

This section discusses fundamental aspects of electronic design, FPGA device architecture and technology mapping.

2.1 Digital Circuits

In the digital design process, a digital circuit can specify the design at different levels of abstraction, as shown in Figure 2.1. Usually, digital designs are found in a higher-level abstraction since the design process and effort are more negligible at this level. However, the actual implementation is achieved by exploding the digital design into a lower-level abstraction to determine the functionality [26]. `Synthesis` refers to the automated process of converting a higher-level abstraction into a functionally equivalent design in a lower-level abstraction. `Simulation` is also used to verify the functionality of a digital design. This is the process of verifying output vectors of digital circuits by comparing them with higher-level outputs.

2.1.1 System Level

The system level abstraction comprises the biggest digital building blocks, such as CPUs or computing cores, to represent the digital design. Digital designs at this

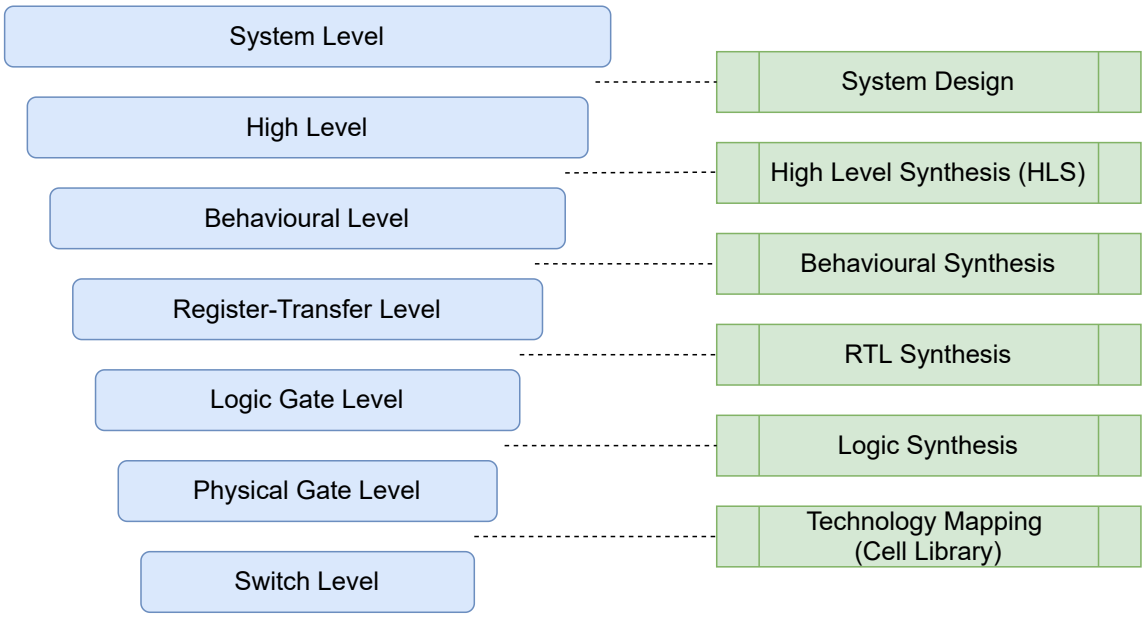


Figure 2.1: Synthesis Types along with Abstraction Levels [1].

level usually are represented using traditional programming languages, like C, C++, and Matlab [27]. Some system level design tools exist to help connecting system level design blocks. However, no common synthesis tool proposed automatic transformation of the system level design into a low-level representation. The IP-XACT file format is proposed in IEEE 1685-2009, representing system level designs and building blocks that can be used in this level of abstraction [28].

2.1.2 High Level

The high-level abstraction is also represented by traditional languages, such as C, C++, and SystemC; however, more limitations are applied compared to the system level. In the high-level abstraction, only applicable and plausible circuits in hardware design are permitted [29]. For instance, usage of pointers in the C language would be limited to only memory inference, and therefore dynamic memory allocation is not possible since no such concept exists in hardware. Most commercial vendors' CAD tools support the transformation of the high-level abstraction to low-level represen-

tation [30, 31, 32, 33], while a few free and open-source software (FOSS) synthesis tools, such as LegUp [34] and Google-XLS [35], propose this feature as well.

2.1.3 Behavioural Level

The behavioural level of abstraction provides advanced data and control flow in digital design, enabling algorithmic descriptions of hardware and synchronization between different blocks and processes. The sequential blocks, i.e., `always` in Verilog and `process` in VHDL [36], provides the imperative programming to be used for data types. Sequential blocks are provided with a sensitivity list, including signals and conditions, that handle the execution and value specification of data types. For instance, once a sensitivity signal changes its value or condition, the sequential block is executed in the simulation process. In the synthesis process, the synthesizer also translates the sequential block into an appropriate datapath using data type components [37].

2.1.4 Register Transfer Level

Data transfer among registers is known as Register Transfer Level (RTL) transfers, while the event-driven (sequential) model is considered the basis for RTL transfers. In this level of abstraction, the digital circuit is represented by combinatorial datapaths, and data type components, such as data flip flops (DFFs) [38]. In Verilog, continuous and registered assignments represent the combinatorial datapaths and sequential components, respectively. Many optimizations and analyses, such as FSM detection and optimization, memory identification and optimization of shareable resources, usually happen at the RTL abstraction. RTL synthesis could easily be inferred as a straightforward transformation of digital components to their gate-level representation. However, it is usually accompanied by numerous modifications and optimizations that make it a highly sophisticated process [37, 39].

2.1.5 Logic Gate Level

In the logic gate-level of abstraction, a digital circuit is described using a netlist, discussed in Section 2.5. A logic gate-level comprises small single-bit basic building blocks, such as AND, OR, NOT and data types (registers). This level of abstraction can be displayed using multiple formats, such as Electronic Design Interchange Format (EDIF), BLIF, and HDL languages such as Verilog and VHDL. The logic gate-level of abstraction usually comes with two primary challenges: finding optimization opportunities and optimal netlist mapping to the available physical gates. Multiple approaches are taken for logic synthesis, in which the simplest one is two-level synthesis, achieved by converting the logic function into a sum-of-products using the Karnaugh map. However, the most straightforward approach would often result in the worst efficiency. Therefore, complicated multi-level approaches [40] are used, which usually turn logic functions into Binary-Decision-Diagrams (BDD) or AIG [41, 22, 23]. The BDD method has the advantage of being in normalized form, while the AIG approach is known for having much better worst-case performance [39].

2.1.6 Physical Gate Level

In the physical gate-level of abstraction, a digital circuit is represented by gates only available in the target fabric [42]. In FPGAs, LUTs are considered as the basic building blocks. However, in other platforms, the basic building blocks would be considered as either NAND, NOR, or NOT gates in addition to data type registers or even larger cells such as half adders and digital signal processors (DSPs). The physical gate-level is usually the lowest level of abstraction for synthesis tools.

2.1.7 Switch Level

The switch level is the lowest level of abstraction so that single transistors operate as the biggest building block of the digital design representation. Technically, the switch level of abstraction specifies the interconnection of transmission gates which are abstractions of an individual metal-oxide semiconductor (MOS) and complementary metal-oxide semiconductor (CMOS). Transistors can be either on or off at the switch level, showing they are conducting or not conducting. Although the switch level could be modelled in HDLs such as Verilog and VHDL, they are not common in modern digital platforms since the physical gate is typically considered the lowest possible abstraction level of digital components.

2.2 Electronic Design Automation

Nowadays, technology has become tightly integrated with human lives. While hardware mainly comprises this integration, generating such technology requires a considerable effort in the design and verification process. Electronic Design Automation (EDA) refers to automating the validation and verification of a given electronic system model at the electronic system level (ESL). EDA tools perform multiple synthesis and verification steps on an ESL design to ensure the correctness of the design against the specification of the system [43].

The system could vary from a simple IC to more complicated platforms such as FPGAs. FPGA vendors have their own closed proprietary EDA tools to make a given design, typically written in behavioural hardware description languages, such as Verilog and VHDL, compatible with their FPGA specifications. However, the fully FOSS EDA-like tools, such as Torc [44], RapidSmith [45, 46], Yosys+nextpnr [47], SymbiFlow [48], and VTR [9, 10], have recently become apparent competitors for such vendors.

Basically, EDA tools comprise multiple CAD tools performing required alterations to the electronic system model. Synthesizers are usually used as the first CAD tool to perform the elaboration and synthesis process in the FPGA EDA tool. Optimization and technology mapping processes may be included in synthesizers or provided by other tools to make the design compatible, based on the target specification. Last but not least, in FPGA EDA flow, a final tool is required to perform the physical implementation, which includes packing, placement, and routing.

2.3 FPGA Architecture

FPGA architecture is usually referred to as the physical layout of a fabricated FPGA platform. The architecture determines the available hard logic blocks and how logic blocks and channels are located in an FPGA device. With the emergence of FOSS EDA tools, exploring the various FPGA architectures became an interesting research area.

Betz et al. [49] proposed a high-level FPGA architecture description language to make this exploration possible. The authors proposed an embedded architecture generator in the VPR tool that translates the high-level architecture description into a detailed FPGA architecture. Odin-II and, more broadly, the VTR tool use this architecture file to determine the hard logic blocks availability and target layout specification. Specifically, the Odin-II partial mapper decides the heterogeneous logic block inference based on a given target architecture file, while the VPR CAD tool implements the Routing Resource Graph (RR-Graph) based upon the architecture file [10]. Regarding the hard blocks, architecture files include the definition of block ports, their size and direction, in addition to corresponding area and power.

The most common hard blocks in the VTR architecture files are memory blocks and DSP blocks, such as multipliers. Odin-II specific configuration for memory blocks

comprises two Random Access Memory (RAM) block types. The first is Single Port RAM (`SPRAM`) that has a single clock signal with a shared address port for `read` and `write` accesses. `SPRAM` can only perform a single operation at a time, i.e., reading from memory or writing into memory in each clock cycle. Another type is Dual Port RAM (`DPRAM`), comprising two separate sets of ports to perform dual access to the memory block. Using `DPRAM`, we can perform two memory accesses at the same time through different ports [50]. As an example, a `read` access can be performed through the first set of ports, while a `write` access can be executed through the second set with a different address.

2.4 Logic Synthesis and Technology Mapping

Logic synthesis is the process of transforming the RTL description of a digital circuit into the logic gate abstraction level. Technology mapping is referred to as converting a general boolean logic network into a netlist of specified components in a technology library. For FPGAs, the technology mapping is realized by mapping the design netlist into a functionally equivalent K-LUT network, where the target FPGA architecture specifies the K value. Despite the synthesis process usually integrating with elaboration, optimization, and technology mapping, the ABC developers make the distinction clear. ABC is primarily responsible for technology mapping in the VTR flow. That means Odin-II does not provide complete technology mapping, as it is not needed. However, Odin-II performs the technology-aware partial mapping by specifying how to pack design components into hard logic blocks.

The partial mapping is executed in two steps, inferencing (identification) and packing stages. First, Odin-II extracts all hardenable components that have the potential to be mapped to hard logic. Then, it decides the hard/soft logic inference of hardenable components in accordance with the target device architecture [11]. There is a

correlation between structural attributes like digital circuit congestion and the properties of the designed product [51]. Early estimation of the final physical structure allows for low-cost manipulating of logical gates, in fact it could improve the downflow placement and routing. Awareness of the architecture structural properties is greatly helpful in this way, as seen with the Odin-II's partial technology mapper and ABC's technology mapping.

2.5 Netlist

In electronic design, a netlist is a list form of the structural circuit implementation. A netlist illustrates the fundamental design building blocks, i.e., the types and architectures, and their logical connections [52]. The structure and implementation of the netlist may vary according to the purpose of any individual project, such as `Logical`, `Instance-based`, `Net-based`, `Hierarchical` or `Physical`. However, all of them share the same goal: describing the circuit. In the VTR project, an `Instance-based` netlist is applied, which means the netlist describes the circuit using component instances [52].

2.5.1 Berkeley Logic Interchange Format

Berkeley Logic Interchange Format (BLIF) is an approach to represent a logic-level hierarchical circuit description in a human-readable format [53]. From the high-level aspect, a circuit is defined as a combinational or sequential network of logic functions, which can be represented as a directed graph of combinational logic cells in addition to sequential logic elements. In BLIF, each node is associated with a function and has a single output signal. Correspondingly, each net or signal is driven with only one driver. A simple digital circuit in BLIF is represented in Figure 2.2. The `.model` represents the flattened hierarchical modules in digital circuit designs.

```
1  | .model Registered_OR
2  | .inputs a b clk
3  | .outputs out
4  |
5  | .names GND
6  | .names VCC
7  | 1
8  |
9  | .names a b OR_output
10 | 1- 1
11 |  -1 1
12 |
13 | .latch OR_output out re clk 3
14 |
15 | .end
```

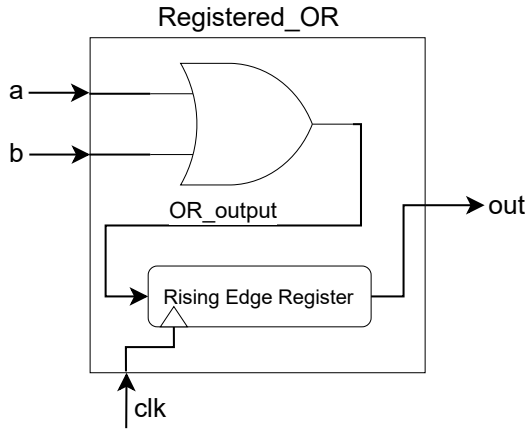


Figure 2.2: Simple Digital Circuit Represented in BLIF.

As an example in Figure 2.2, the module `Registered_OR` comprises three single-bit input signals and a one-bit output pin. The `.names` records are associated with logic gates, whereas they must have only a single output signal. Constant Signals such as the ground (low) or the source (high) can also be defined using the `.names` keyword. For instance, `.names GND` without a number underneath represents the signal named GND connected to the 0, while the `.names VCC` with number 1 shows the signal named VCC that is constantly connected to the voltage common collector. Numbers below the `.names` specify the associated logic gate functionality. The 0 and 1 represent the low and high values, respectively, while the hyphen (-) shows the value of the don't-care. The third `.names` record in Figure 2.2 shows an OR logic gate with `a` and `b` as inputs and `OR_output` as the single output pin. While the input `a` is 1, regardless of the value of the input `b`, the output is 1 and vice versa.

Sequential elements, including latches and flip flops, are represented as `.latch` in BLIF. The `.latch` record should receive five arguments, which the first and second ones are the input and the output signals. The third value represents the type of the latch that could be falling edge (fe), rising edge (re), active high (ah), active low (al), and asynchronous (as). The fourth argument is the latch clocking signal, and the last value specifies the latch's initial value. The latch's initial value could be 0

or 1, representing the high and low value, 2 determining the don't-care state, and 3, the unknown value. Note that all feedback loops in Berkeley Logic Interchange Format must include a generic latch since pure combinational logic is not allowed.

2.6 Odin-II Partial Mapping

As a packing estimator, the Odin-II complex partial mapper determines how to pack digital logic structures into hard logic blocks available on the target FPGA fabric or explode them into soft logic to utilize LUTs. In general, the Odin-II partial mapper performs packing in two stages. The partial mapper extracts hardenable digital logic components in the first phase, called identification or inferencing. Once the design structures are identified, it performs packing based upon available complex logic blocks in the given FPGA architecture file. In the case of soft logic implementation, the base framework of the Odin-II partial mapper provides the instantiation of standard logic cells, such as logical operations, shift, addition, subtraction, multiplication, etc. However, for new logic structures, such as floating point units, the user is responsible for providing a packer for the proposed technology [11].

In addition to the basic technology mapping techniques, Odin-II's partial mapper can generate the desired device footprint using a genetic algorithm in the soft logic instantiation. The GA allows the synthesis tool to explore the different soft-logic implementations of netlist components. The Odin-II partial mapper GA has only been deployed for addition operations with RCA as the smaller and slower architecture and CSLA as the larger and faster design. The GA-based partial mapper affects the components in the netlist towards achieving the desirable circuit footprint. While demonstrating a tangible change in the entire VTR flow, showing that a modified circuit aligns with the user goal is achievable, it requires extra time to find a configuration that best matches the demanded footprint. The extra execution

time depends on various criteria, such as the number of add operations in the benchmark, the available adder types, and the GA parameters. According to experiments performed on Odin-II GA-based partial mapper, the fitness value improvement for a more significant combination of parameters than the enhancement of time is subtle. Furthermore, the improvement of fitness for benchmarks with more operations is marginal. In other words, the experiments prove that the desired configurations could be found earlier in large benchmarks [12].

The Odin-II partial mapping stage can also provide hard/soft logic trade-off decisions, named mixing optimization. Instead of the GA approach that explores different soft logic implementations of an operation, the mixing optimization provides the Odin-II partial mapper with the ability to determine an operation's hard/soft implementation based upon the user request. The optimization can be configured to be performed on either a portion of the total hard blocks of an operation or use exactly the number specified by an input parameter [13]. Previously, the hard/soft logic trade-off has been fully implemented for multipliers, however, the support for the addition operations has been recently developed [14]. Once the FPGA layout is retrieved from the architecture file and the hard blocks' counts are calculated, the optimizer must decide which operation should be implemented on soft and hard logic. This trade-off decision is determined according to an accumulative value calculated from the top to bottom DFS of the netlist, starting from the potential hard block node. Considering the value of leaf (output) nodes as 1, the DFS function increments by one the maximal children's value in each proceeding level to predict the path length using the node count as a correlated metric [15]. In the end, the operation nodes with more significant values will be instantiated with complex logic, while the remainder will be implemented using their soft logic design.

Chapter 3

Design

Describing the details of implementing the Yosys+Odin-II CAD tool, it is worthwhile to first state the higher-level preliminary work that provides a contextual basis for the integration. This section provides a broad overview of the new interlocking phases in the partial mapping flow needed by the Yosys+Odin-II requirements. We then discuss the approaches taken to fully deploy the partially supported or unsupported features in Odin-II logic cell mapping.

Odin-II provides the synthesis and simulation processes for digital circuit designs. The Verilog synthesis includes six phases: parsing, abstract syntax tree (AST) optimization, netlist elaboration, partial technology mapping, netlist optimization, and BLIF generation. The hardware description is elaborated and compiled into an RTL logic netlist in the first three phases. The FPGA technology-dependent process starts to map the elaborated netlist components to soft logic or available complex logic blocks specified in a given architecture from stage four. Once the synthesis flow is completed, Odin-II produces a technology-dependent netlist in a BLIF file. The simulation process initiates after BLIF generation. Odin-II parses the BLIF file and creates an in-memory netlist. Then, the netlist simulation process starts to validate the functional correctness of the generated circuit [54]. Our contribution

is based on the combination of Odin-II's ability to parse a BLIF file and perform the partial mapping on its netlist data structure for a coarse-grained netlist generated by Yosys. Numerous modifications for RTL logic transformation, optimization, and technology mapping are applied to the Odin-II technology mapping to achieve the desired goal. Figure 3.1 depicts the synthesis flow for all synthesizers. The dotted line shows the Odin-II synthesis flow, while the dashed line illustrates the standalone Yosys synthesizer. Yosys+Odin-II is shown using solid lines utilizing the same post-synthesis process as the Odin-II synthesizer. As shown in Figure 3.1, the Yosys+Odin-II technology mapping is performed through three stages: BLIF I/O Interface, BLIF Elaboration, and Partial Mapping.

3.1 BLIF I/O Interface

The Odin-II input interface accepts Verilog and BLIF languages as inputs through command arguments, while it generates the BLIF file as the only output form. In this project, Odin-II's I/O interface was replaced with a generic C++ class. The generic I/O class equips the Odin-II program with reading and writing methods, in which the I/O file format could be determined based on command arguments. A separate class is allocated to each language, including the necessary methods to read and create a netlist or write the netlist correctly. Since our contribution is focused on the Odin-II partial mapping and BLIF parsing, we decided to touch the Odin-II synthesis flow as little as possible. As a result, the Verilog parser functionality is left intact. However, the Odin-II BLIF reader is slightly changed to be able to read Yosys BLIF components. Based on the Yosys generated BLIF file, the Odin-II BLIF reader creates a netlist data structure. Yosys is asked to generate a hierarchical-flattened netlist, which instantiates all modules in the RTL description into the top module. In the end, the Yosys output BLIF file has only one `.model` record. The Odin-II BLIF

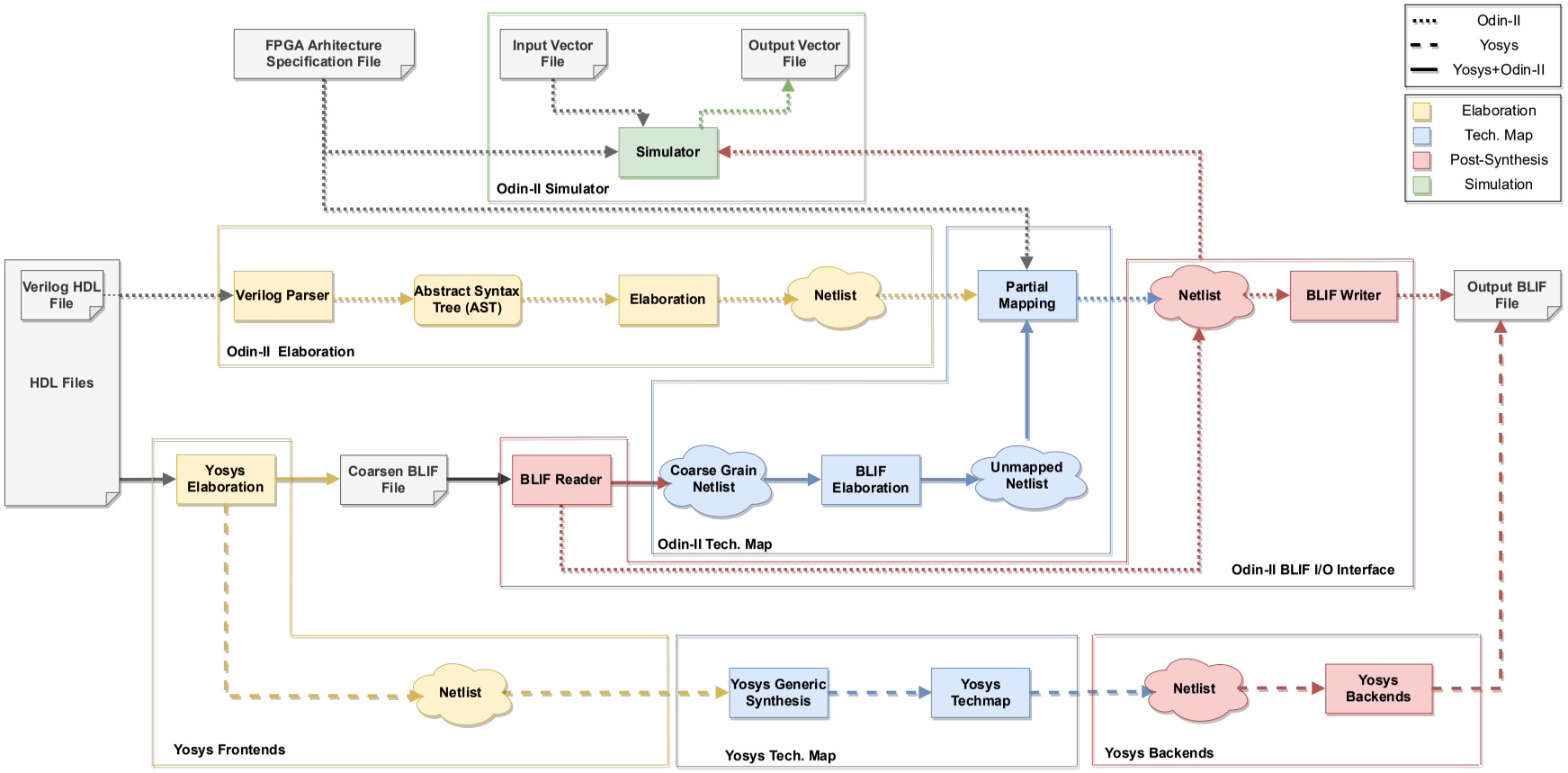


Figure 3.1: Yosys+Odin-II Synthesis Flow.

reader iterates over the input file, and once it faces a cell (represented as `.subckt` in BLIF), it searches through the available sub-circuits. If a corresponding sub-circuit already exists, the reader extracts the information, creates a new object, and adds it to the netlist. Otherwise, it creates a new cell with the current sub-circuit ports information, such as size and name, and stores it for future retrieval.

3.2 BLIF Elaboration

BLIF elaboration is added as a new stage to the Odin-II technology mapping. Technically, the infrastructure of Odin-II and Yosys differ from each other. As a result, a transformation process is required to overcome incompatibility issues. Some logic inferences were not supported in Odin-II previously. We also added support for them in this contribution. As previously described, we tried not to touch the Odin-II synthesis structure. Therefore, resolving newly added components into Odin-II supported components—followed by some optimizations—happens in the BLIF elaboration phase.

3.2.1 Cell Transformation

Cell transformation includes port modification, signal padding, and converting unsupported logic blocks to Odin-II supported ones. The Odin-II logic instantiation process expects logic blocks with I/O ports in the specified format. As a result, ports are modified in terms of order and size in this stage. According to Yosys logic cells, this process is required for bitwise, logical, and binary comparison operations. In the case of available hard logic in the given FPGA architecture, Odin-II requires addition operations with carry-in and carry-out, which are also handled in this phase. In some cases, Odin-II requires logic block inputs with equal sizes. Therefore, the padding process is also performed based on the signedness of ports.

3.2.2 Complex Logic Cells

Some logic, such as division, modulo, and exponentiation, are not fully supported in Odin-II; however, this study adds the low-level instantiation process. For example, Odin-II only supports exponentiation with a constant exponent, and division and modulo are not fully supported. Moreover, Odin-II is built upon the formal data flip-flop cells, while Yosys generates eight different data flip-flop cells with enable, set, clear and asynchronous/synchronous reset. Therefore, not only the unsupported logic instantiations are required to be implemented, but a process is required to transform the remainder of logic cells to the Odin-II supported format.

3.2.2.1 Data Flip-flops

Data Flip-flops (DFFs) are key components in hardware design, especially in FSMs, that can cause sequential loops and form a critical part of control logic [55]. While all DFFs eventually provide the same functionality, Yosys generates eight different DFF types in the RTL level of abstraction with additional abilities. According to the requisite digital design needs, Yosys-generated DFFs vary from asynchronous DFFs to synchronous ones with control signals. One of DFF's applications is to provide control over loops in digital designs. However, asynchronous DFFs are themselves including loops. As discussed earlier in Section 2.5, combinational loops are not permitted in BLIF and also not supported by ABC. Therefore, Yosys is asked to map the DFFs with asynchronous reset to synchronous ones while generating the coarse-grained netlist. The instantiation of the synchronous DFFs is added to Odin-II BLIF elaboration based upon designs depicted in Figures 3.2 to 3.7 [1, 56]. It is worth noting that the checking process of control signals polarity is performed on the synthesizer side. In other words, boxes with an equal sign inside are only in place to show the logic behind the design of each synchronous DFF. However, the polarity equality is checked in the synthesis process, and multiplexer inputs are then

re-ordered based upon the polarity.

3.2.2.2 Multipliers

Multiplication is among the operations that are moderately common in hardware design. The Odin-II partial mapper already provided extensive support for multipliers. However, there are still opportunities to improve the Yosys+Odin-II partial mapper while instantiating the multipliers. More specifically, the constant and variable multiplications have not been classified during the regular Odin-II partial mapping, while in the case of having constant multiplication, the partial mapper could instantiate much more performantly. This thesis separates the constant and variable multiplication operations by checking the multiplicand and multiplier during the synthesis. On the one hand, the constant multiplication operation is reduced to several shift and addition operations based on Equations (3.1) and (3.2), resulting in substantially smaller circuitry and faster propagation rate. On the other hand, the variable multiplication is still using the traditional approach implemented in the Odin-II partial mapper.

$$\begin{aligned} &\forall\ K \in \mathbb{N}\ and\ k \in \{0,1\} \\ &K = k_{i-1}2^{i-1} + k_{i-2}2^{i-2} + ... + k_1 2^1 + k_0 2^0 \end{aligned} \tag{3.1}$$

$$\begin{aligned} M \times K &= k_{i-1}(M \cdot 2^{i-1}) + k_{i-2}(M \cdot 2^{i-2}) + ... + k_1(M \cdot 2^1) + k_0(M \cdot 2^0) \\ &= k_{i-1}(M << i-1) + k_{i-2}(M << i-2) + ... + k_1(M << 1) + k_0(M) \end{aligned} \tag{3.2}$$

3.2.2.3 Division and Modulo

Considering two unsigned binary numbers: D (m-bit) as the dividend and d (n-bit) as the divisor, the division is defined as a process of finding two unsigned binary numbers

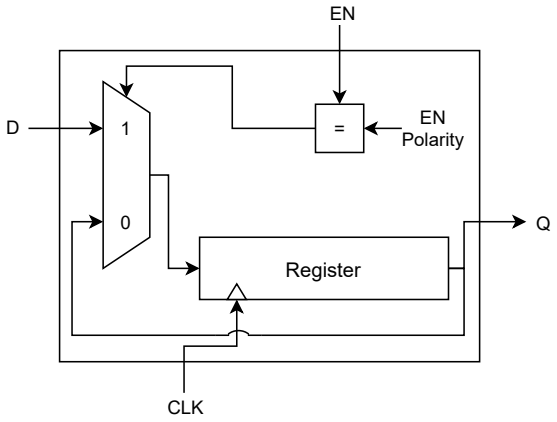


Figure 3.2: DFF with Enable Signal (DFFE).

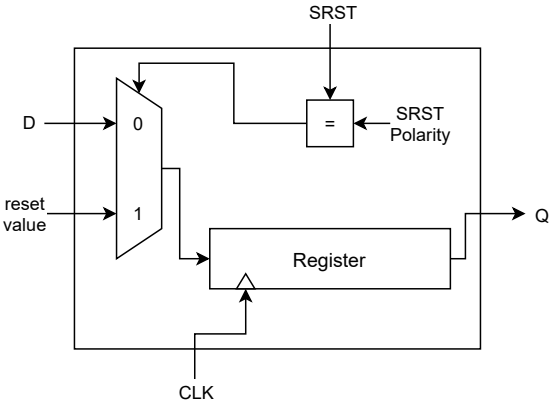


Figure 3.3: DFF with Synchronous Reset (SDFF).

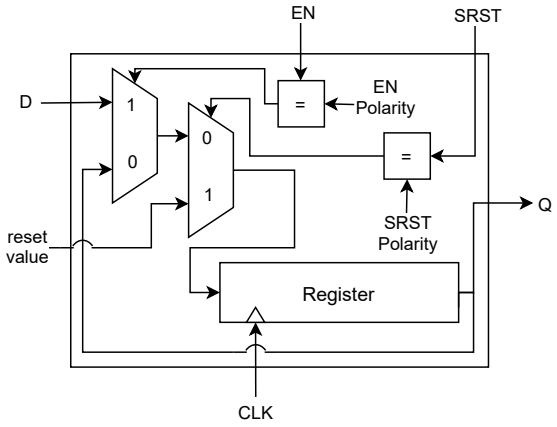


Figure 3.4: DFF with Enable Signal and Synchronous Reset (SDFFE).

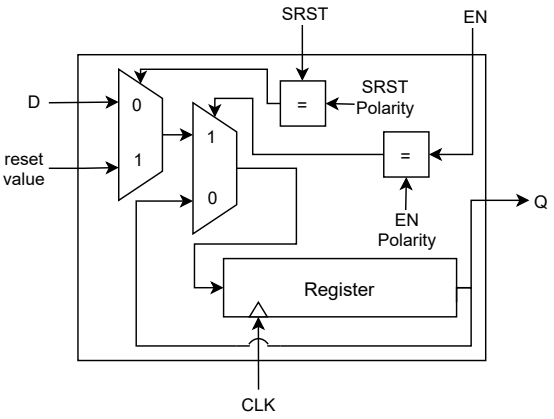


Figure 3.5: DFF with Enable Signal prior to Synchronous Reset (SDFFCE) (Enable should be active to be able to reset).

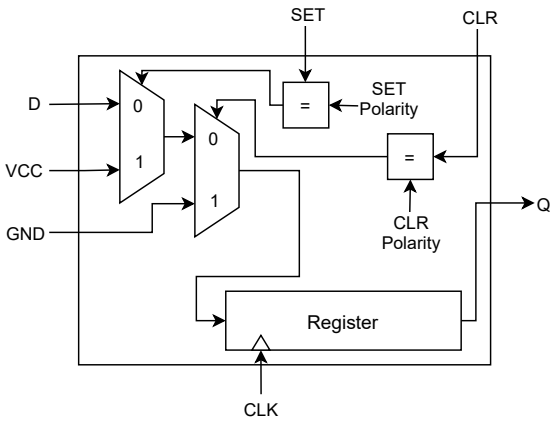


Figure 3.6: DFF with Set (VCC) and Clear (GND) Signals (DFFSR).

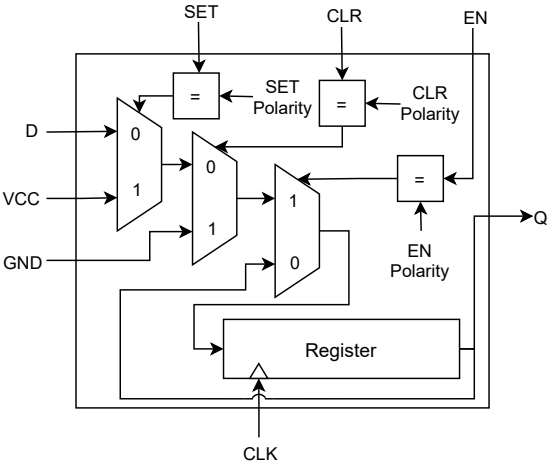


Figure 3.7: DFF with Set (VCC), Clear (GND) and Enable Signals (DFFSRE).

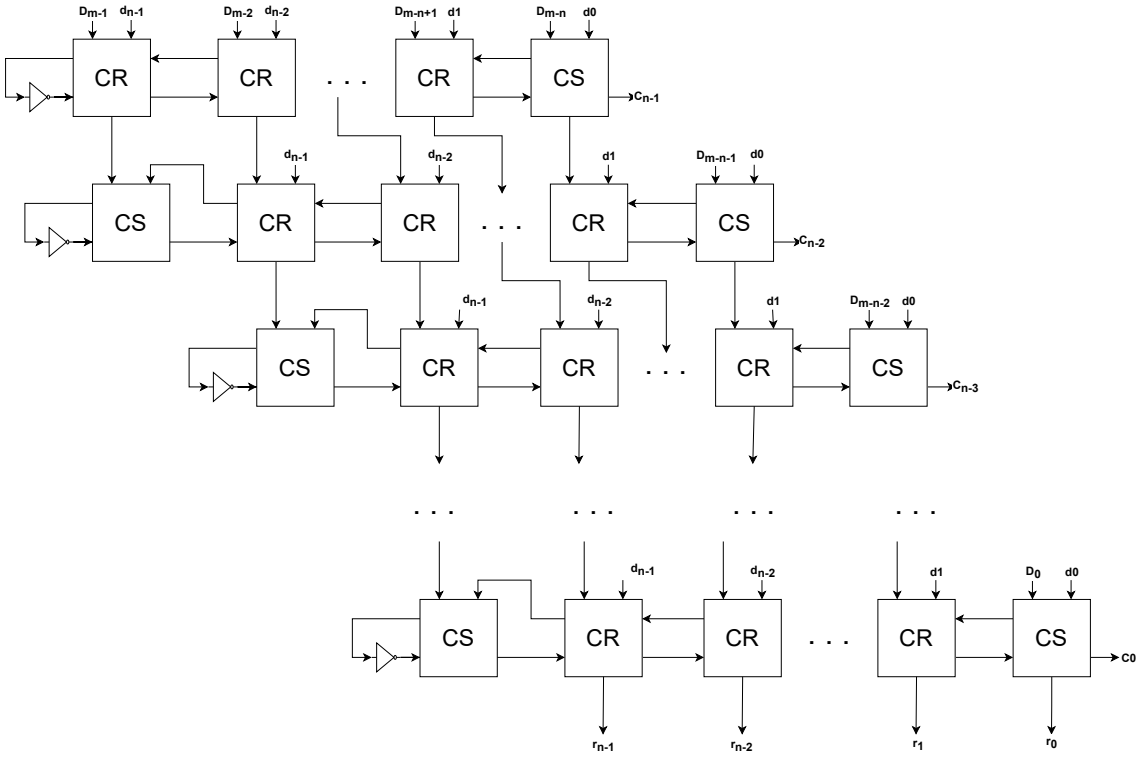


Figure 3.8: Combinational Division Network [2].

C (quotient) and r (remainder) where $d \neq 0$ and $r < d$ and the Equation (3.3) is true:

$$D = C.d + r \tag{3.3}$$

In hardware design, a combinational solution for a division operation is possible that attempts to mimic the process by splitting the division process into a series of shifts and subtractions. Figure 3.8 depicts the schematic representation of combinational division using subtractions and shifts, proposed by Ruiz et al. [2]. In the first step, the divisor is subtracted from the n most significant bits of the dividend. If the subtraction output is positive, the difference is transferred to the next level as the n most significant bits of the altered dividend and the quotient bit is considered 1. However, if the output is negative, the dividend passes to the next level unchanged while the quotient bit is 0. To deploy the combinational division network,

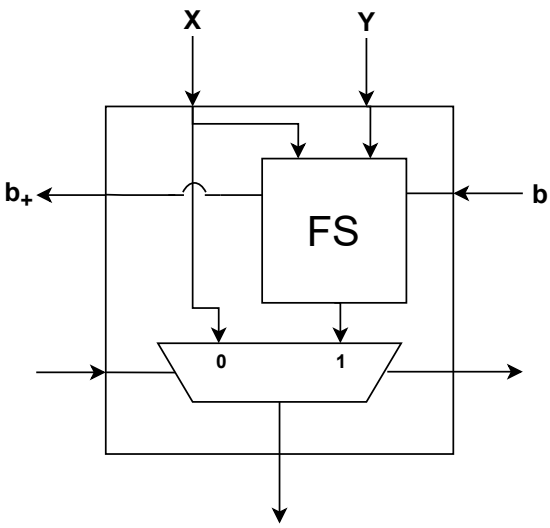


Figure 3.9: Combinational Division Network - CR Cell Structure [2].

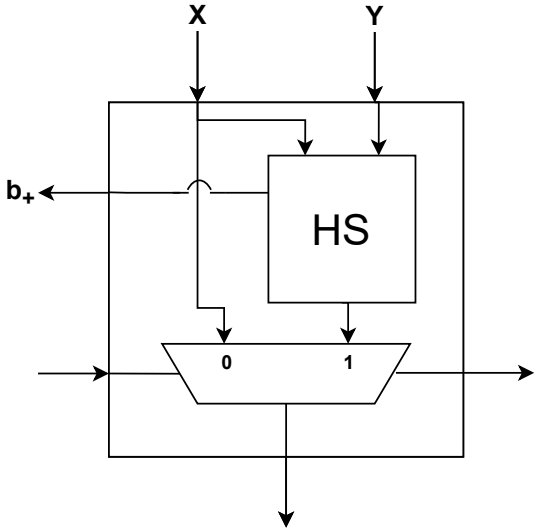


Figure 3.10: Combinational Division Network - CS Cell Structure [2].

full subtractors taking into account the possible borrow of the previous bit with a 2-1 multiplexer are constructed as the cell CR, shown in Figure 3.9. On the other side, a half subtractor with a 2-1 multiplexer is considered for bits without input borrows as depicted in Figure 3.10.

Technically, a few aspects of the division operands have to be considered to utilize the combinational division method. First, it is assumed that the lengths of the dividend and the divisor follow the relationship of $m = 2n - 1$. Secondly, the most significant bit of the divisor always should be 1. Neither of the assumptions implies restrictions since, on the one hand, the size of the operand is adjusted by concatenating with zeros, while on the other hand, shifting the divisor is performed to make the second assumption attainable. Note that we properly transfer the shift to the quotient and remainder. In the Yosys+Odin-II BLIF elaboration, quotient (C) pins are connected to the division cell outputs while the remainder (r) pins are attached to the modulo cell outputs.

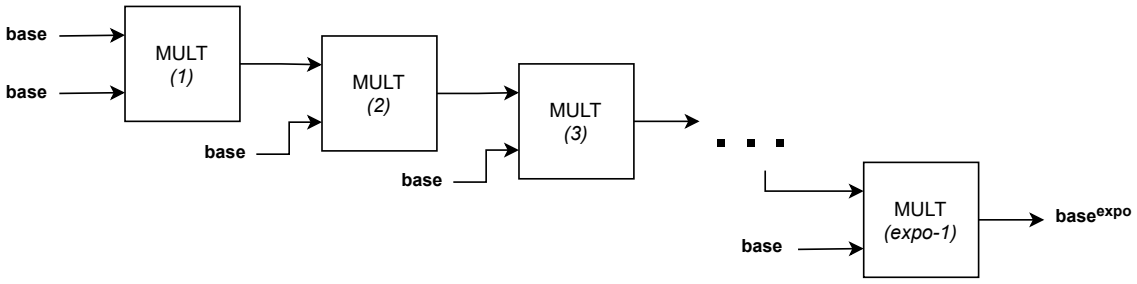


Figure 3.11: Constant Exponentiation Circuit.

3.2.2.4 Exponentiation

Raising an integer number N (base) to the power of P (exponent) consists of multiplying the base by itself P times. As a result, with appropriate multipliers, any integer number N can be raised to any power. Implementation of the exponentiation operation for all possible situations is not standard due to massive resource consumption. However, we provide the rudimentary implementation of the power module to provide users with the ability of synthesizable power operation for possible common situations.

To implement the exponentiation logic, we categorize the power cells into three different processes. First, we check the base and exponent signals to see whether they have a constant value or they are variable signals. Then, the corresponding operation, selected among pure constant exponentiation, constant exponentiation and variable exponentiation, is performed to instantiate the power arithmetic cell.

Pure constant exponentiation comprises a base N to a power of P, where both N and P have a constant value. As a result, this operation is calculated during the synthesis time, and no further such logic instantiation processes are required. Constant exponentiation includes a base N to the power of P, where only the P is constant. Knowing the value of the exponent in this situation provides us with the ability to perform the exponentiation process using at least the $P-1$ number of multiplication cells.

Figure 3.11 depicts the design we utilize in this thesis to instantiate the constant ex-

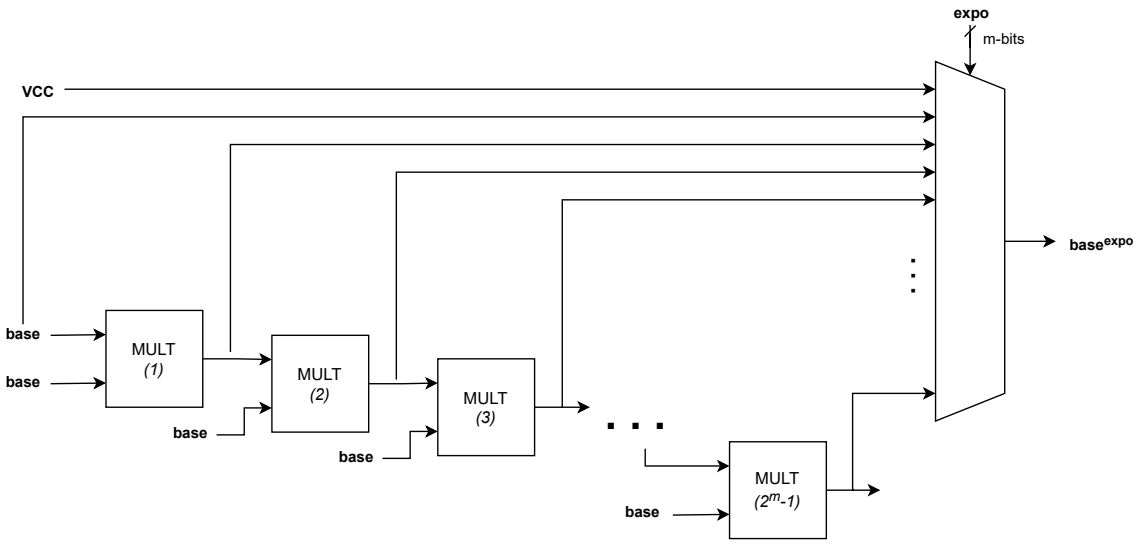


Figure 3.12: Variable Exponentiation Circuit.

ponentiation. Once the exponent value is recognized, a chain of multipliers including the $P-1$ number of multiplication cells is instantiated. The front chain multiplication cell inputs are connected to the base signal, while the previous multiplication cell output feeds the first input of the rest of the multipliers. Consequently, the last multiplier output contains the value of the base N to the power of P, calculated by $P-1$ multiplication cells.

The variable exponentiation is referred to as raising a base N to the power of P, where both N and P are signals that can have unspecified values during synthesis. Figure 3.12 depicts the combinational instantiation process of variable exponentiation. The simplest and most inefficient approach is to infer a chain of multipliers, considering an unlimited number of multiplication cells, that the number of multipliers is equal to the maximum possible value of the exponent. The outcome of the exponentiation operation is achieved using a multiport multiplexer with the exponent signal as the selector and all possible exponentiation values as inputs.

Note that the sequential design with many fewer multiplication cells is possible. A possible sequential power module using a squarer and a multiplier could calculate the P power of the base N in $(P/2)+1$ clock cycle. Using a binary method for

Algorithm 1: Sequential Exponentiation

```
begin
    R ← 1,  i ← m − 1
    while i > 0 do
        1) if p_i = 1 then  R ← R · N
        2) R ← R^2,  i ← i − 1
    end
    if p_0 = 1 then  R ← R · N
end
```

exponentiation, for every P, the Equation (3.4) is true, where p_i could have the value of 0 or 1. Equation (3.5) shows how the P power of the base N can be calculated, whereas Algorithm 1 explains the calculation process in detail.

$$\begin{aligned} P &= p_{m-1}2^{m-1} + p_{m-2}2^{m-2} + ... + p_1 2 + p_0 \\ &= ((..(p_{m-1}2 + p_{m-2}) + ..)2 + p_1)2 + p_0 \end{aligned} \tag{3.4}$$

$$N^P = (..((N^{P_{m-1}})^2 \cdot N^{P_{m-2}})^2 \cdot \ ... \ \cdot N^{p_1})^2 \cdot N^{p_0} \tag{3.5}$$

Although sequential exponentiation seems fascinating, it is not applicable during synthesis since it needs memory access and complicated arithmetic operations. Moreover, users do not always provide a clock signal or expect sequential logic while they design combinatorial circuits. As a result, we decided to utilize the inefficient combinatorial method by letting users benefit from the logic implementation of possible power operations. At the current research point, the decision about exponentiation implementation is not a great concern. This claim is mainly laid on two facts. First, the power operation is not a common operation for the synthesis process. Second, for future developments that are thoroughly discussed in Section 6.1 we plan to conform to the Yosys style instantiating process, in which the approach lets a user provide an RTL design for the technology mapping process of such logic blocks. It is worth considering other commercial tools, such as Quartus [56], only support the exponenti-

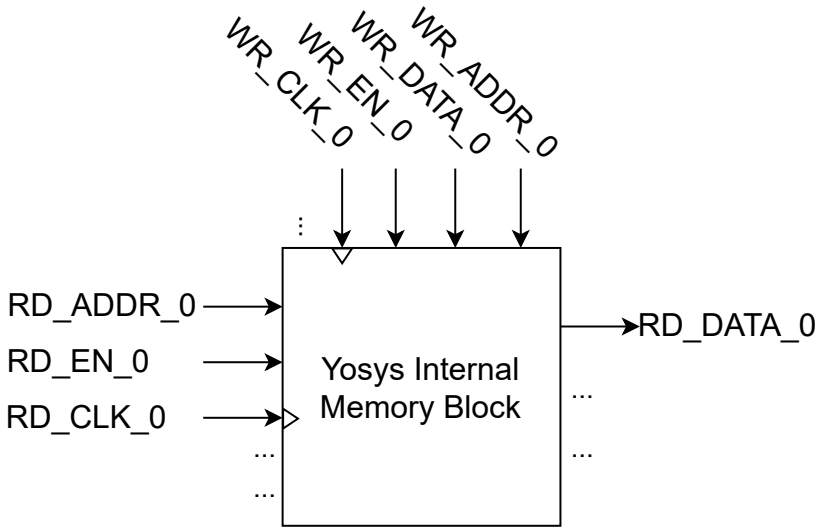


Figure 3.13: Yosys Internal Memory Block.

ation operation where the base is a positive power of 2 with a non-constant exponent, exponentiation operation with constant exponent and pure constant exponentiation.

3.2.3 Memory Blocks

Based on the specified configuration for Yosys, three different memory types could appear in its generated netlist. The VTR primitives Verilog file, including `SPRAM` and `DPRAM` definitions, is always passed to Yosys. Once Yosys encounters VTR primitives in an input Verilog file, it generates the primitive cells intact in the coarse-grained BLIF file. As an example, Yosys does not perform memory transformations to an array of registers on `SPRAM`s and `DPRAM`s, while keeping modular hierarchy, it generates the corresponding records in the BLIF file. Since VTR memory primitives are already supported in Odin-II, the corresponding cells are only created in the BLIF elaboration phase and then passed to the subsequent phases. However, there is no such support for Yosys memory cells resulting from implicit memory inferences. The Yosys internal memory structure appears as a `$mem` sub-circuit in the coarse-grained netlist. Basically, the `$mem` block is a `.subckt` BLIF record, including all memory *read* and *write* signals that have access to the memory instance, as shown

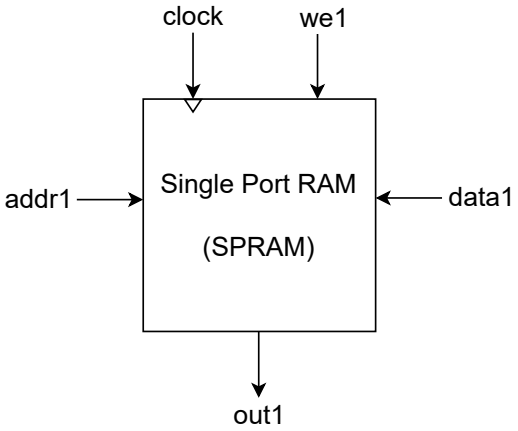


Figure 3.14: Single Port RAM.

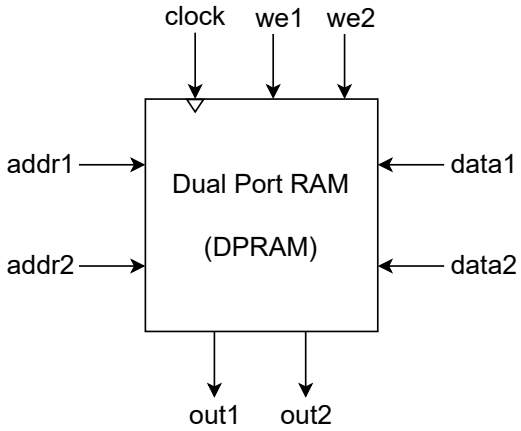


Figure 3.15: Dual Port RAM.

in Figure 3.13. That means memory *read* and *write* addresses are determined as two separate signals. However, as shown in Figures 3.14 and 3.15, the supported memory blocks in Odin-II, i.e., `SPRAM` and `DPRAM`, have a shared address signal for each port. As a result, a fairly complicated process is needed to map the Yosys internal memory blocks into VTR supported memory blocks.

In this thesis, the Yosys memory hard blocks are categorized into *block memories (BRAMs)* and *read-only memories (ROMs)*. `BRAMs` include *read* and *write* accesses to the memory block, while the `ROMs` have only *read* access. To solve the incompatibility issue, we classified the `BRAMs` and `ROMs` based on the number of each access request. Then `BRAMs` and `ROMs` are mapped to `SPRAMs` and `DPRAMs` based on their access types and source of address ports. In the case of having more than two distinct accesses for a single `BRAM` or `ROM` block, we utilize decoded and cascaded multiplexers, using memory enable signals, to map them properly to VTR supported memory blocks.

3.2.4 Optimization

Typically, the optimization process improves the general performance metrics. Synthesizer tools are responsible for performing optimizations such as removing unused

logic and avoiding duplication. ABC handles these cases by performing optimization based on logic expressions. However, ABC cannot perform any optimization on hard blocks since it sees them as black boxes. As a result, Odin-II should perform duplicate hard block removal and sweep away unused logic [57, 58]. In addition to Odin-II optimizations, we discovered that additional optimizations could be employed on Yosys generated hard blocks. Specifically, the implementation of computationally-intensive operations, such as exponentiation and multiplication, could be reduced to achieve more area and speed efficiency. For instance, detecting constant exponentiation operations results in fewer multiplication cells usage, while the constant multiplication could be reduced to shift operations. Furthermore, memory blocks are other potential blocks to be optimized. Yosys generates a block corresponding to an implicit memory request. The memory block includes necessary signals, such as addresses, data and enables. Odin-II calculates the memory dimensions based on the length of address and data ports. While the requested depth of the memory might differ from the inferred depth, Odin-II may infer a large useless memory block based on the connected address width. As an example, when the result of a binary operation is given to memory as an address signal, the final address result will be 32 bits. While the actual depth of memory is not 2 powers of 32, considerable unused memory could be inferred. The extra address bits are pruned to avoid such cases. Odin-II detects this situation utilizing the non-standard parameters generated in the coarse-grained BLIF, showing the declared depth of the memory.

3.3 Partial Mapping

Odin-II's partial mapping starts after the BLIF elaboration phase. Technically, netlist components map to technology-dependent cells in this phase. As mentioned earlier, we aim to not modify the partial mapping phase and prevent any manipu-

lation of Odin-II's synthesis procedure. However, a few essential modifications are applied to the partial mapping, which improves the entire VTR flow performance while Odin-II's synthesis flow functionality remains intact. The changes are outlined as follows:

3.3.1 MUX_2

In the Odin-II infrastructure, the implementation of multiplexers is defined using the one-hot logic representation. The one-hot logic multiplexer is a mux with a one-hot vector as the control signal with a width equal to the number of mux inputs. To instantiate a standard multiplexer with two inputs using one-hot logic, Odin-II has to instantiate an additional `logical_not` gate driven by the selector signal. This could cause area deficiency and a decrease in the speed of the circuits with extra logic. This study adds support for a standard multiplexer instantiation with two single-bit inputs and a single-bit control signal. Although the instantiation of standard and one-hot logic multiplexers is different, we consider both as multiplexer logic cells in the netlist statistic calculation.

3.3.2 Multiport n-bits Multiplexer

Following `MUX_2` instantiation, and due to the substantial need for multiplexers for instantiating other components, the implementation of a multiport n-bit multiplexer `(npbMUX)` is also added to the Odin-II partial mapping. The structure of `npbMUX` is based on the definition of the multiplexer. The instantiation process includes two phases; the first one transforms the `npbMUX` into single bit multiplexers. That means the control signal and the number of input ports remain intact; however, each input port takes one bit after this pass. Afterwards, the single bit `npbMUX`es are decoded into multiplexers with two inputs using `MUX_2` cells with a single bit selector.

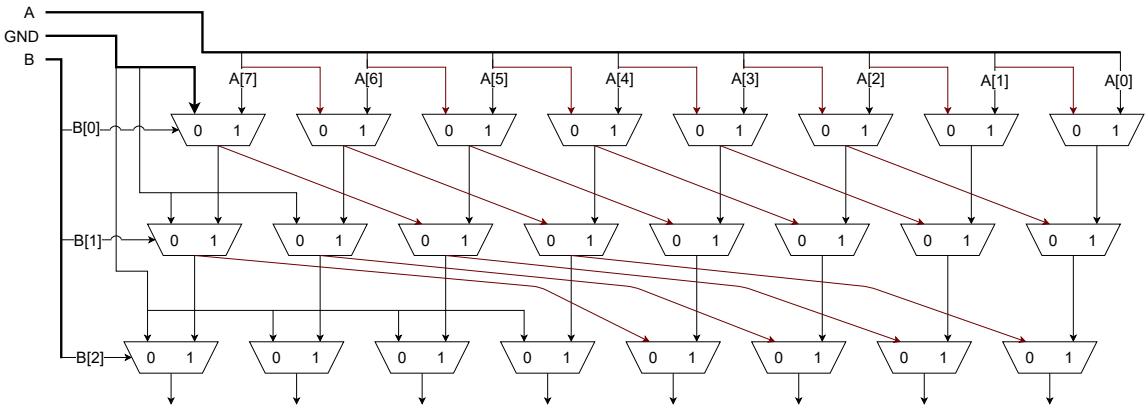


Figure 3.16: 8-bit Right Barrel Shifter using Multiplexers.

3.3.3 Constant-Variable Shift

A constant shift type is referred to as a shift operation where the shift value is constant. In RTL, a constant shift and a variable shift are both shown as shift operations. However, the constant and variable shift difference comes from the drivers of the shift value port. The shift value port is driven by a combination of VCC and GND in a constant shift, while another signal drives it in the variable model. We previously added the inference of constant and variable shift operations by utilizing a multiplexer implementation of a barrel shifter to enhance the performance of the Odin-II synthesis flow in a secondary project. As shown in Figure 3.16, the barrel shifter allows shifting input data, where the amount of the shift is specified via the second input. The constant shift instantiation is embedded in the netlist AST elaboration phase, which is out of the partial mapping reach, and therefore, is not usable for our contribution. To overcome this issue, we moved the detection and implementation of both shift operation types in the partial mapping phase. As a result, the constant shift input manipulation is supported in the Yosys+Odin-II and Odin-II synthesis flows, while the corresponding circuitry similar to Figure 3.16 is created for variable shift operations.

Chapter 4

Yosys+Odin-II

Odin-II parses a BLIF file and loads the netlist into memory with the same data structure that the Verilog parser feeds the synthesis process. This provides us with a potential entry point to the Odin-II flow for extracting the Odin-II partial technology mapping section. To reach the research goal, two approaches are taken to perform the elaboration by Yosys. First, we could execute each tool separately. By doing so, Yosys performs the coarse-grained synthesis on the digital circuit designs using the provided Tcl script in Listing 4.1. Then, the generated BLIF file is passed to Odin-II using command arguments. Second, we could integrate the Yosys elaboration inside Odin-II using the Yosys application programming interface (API).

4.1 External Execution of Yosys

To execute Yosys as an external tool, a list of appropriate commands is required to generate a coarse-grained BLIF file. To this end, a Tcl script is created as shown in Listing 4.1. A few environment variables have been used that should be initialized before executing the script by Yosys. The `PRIMITIVES` is an environment variable showing the path to the VTR primitives Verilog file. The `TCL_CIRCUIT` should be set to the input HDL circuit design file. The `ODIN_TECHLIB` shows the path to the Odin-

II technology library, including the Verilog designs that should be transformed during the coarse-grained BLIF generation. Specifically, DFFs with asynchronous reset using the technology library are transformed to synchronous DFFs. The `adff2dff` Verilog file is adapted from the Yosys common technology library. The `adffe2dff` file is also similar to the previous file with some slight changes that cover the transformation of DFFs with asynchronous reset and data enable signals. To ease the process of setting environment variables and executing Yosys, a bash script called `run_yosys.sh` is created, located in `ODIN-II_ROOT/regression_test/tools`. The `run_yosys.sh` script is inspired by the Odin-II verification script, which is compatible with pre-existing tasks and suites in the Odin-II regression tests.

In Listing 4.1, the `yosys -import` command provides the Yosys commands to the Tcl interpreter. The definitions of `SPRAM` and `DPRAM` are provided to Yosys in the VTR primitives Verilog file. If the VTR primitives file is passed as a library file, Yosys considers memory hard blocks as black boxes. By doing so, the hard block's parameters, such as address and data widths, are no longer adjustable. To overcome this issue, we use the regular Verilog reading routine, but the `keep_hierarchy` at-

```
# Import Yosys commands
yosys -import

# Read VTR baseline library first.
read_verilog -nomem2reg -nolatches $env(PRIMITIVES);
# Read Input HDL Design
read_verilog -nomem2reg -nolatches $env(TCL_CIRCUIT);

#Specify hierarchical information
hierarchy -check -auto-top; autoname;

# Normal synthesis passes
procs; fsm; memory_collect; memory_dff; check;

# Asynchronous dff removal
techmap -map $env(ODIN_TECHLIB)/adff2dff.v;
techmap -map $env(ODIN_TECHLIB)/adffe2dff.v;

# Flattening the design, Removing unsed logic
flatten ; pmuxtree; opt -undriven -full; autoname;

# Output BLIF
write_blif -param -impltf env(TCL_BLIF);
```

Listing 4.1: Yosys+Odin-II Synthesis Tcl Script

tributes for both SPRAM and DPRAM are set in the VTR primitives source file to avoid Yosys resolving them into the top module. The option -nomem2reg prevents Yosys from exploding implicit memories to an array of registers. The option -nolatches is used for both VTR primitives and input circuit design to avoid Yosys generating logic loops. Afterwards, Yosys checks that the design cells match the libraries and detects the top module using hierarchy. The procs command translates the processes to netlist components such as multiplexers, flip-flops, and latches. The fsm performs extraction and optimization of finite state machines. The memory_collect pass collects memories and their ports, then creates a multiport memory cell. The next command, i.e., memory_dff, converts asynchronous memory ports to synchronous ones by merging ports and the related DFFs at their interfaces. The check command eventually examines errors like combinatorial loops, wires with multiple drivers and used wires without any driver.

After performing basic synthesis steps, the techmap command with the input addf2dff transforms DFFs with asynchronous reset to the synchronous form using the design provided by Yosys. The next command follows the same approach but for a modified version of adff2dff with asynchronous reset and data enable signals. The flatten command generates an output netlist with only one module, representing the HDL circuit design's top module. The pmuxtree pass transforms pmux, a subcircuit representing parallel cases, into trees of regular multiplexers. In the autoname passes, Yosys generates an easy-to-read BLIF file by transforming signal names into the short format. Then, the optimization pass is called to make the netlist ready for output. The option undriven ensures that all nets without a driver are removed. The full optimization option is used to remove duplicated inputs in AND, OR and MUX gates. Ultimately, we use the write_blif command to output the coarse-grained BLIF file. The option param prints some additional information about logic cells into the BLIF file, and the impltf option conceals the definition of primary

```
# Coarse-grained BLIF file is already generated
.\odin_II
    -b coarse_grained_netlist.yosys.blif
    -a fpga_architecture.xml
    -o fine_grained_netlist.odin.blif
    --coarsen
    --fflegalize
```

Listing 4.2: Odin-II Sample Command to Perform Partial Mapping for a Coarse-grained BLIF file

netlist ports, i.e., VCC, GND and PAD, in the output.

After interpreting the Tcl script by Yosys, Odin-II starts to perform BLIF elaboration and partial technology mapping using the command shown in Listing 4.2. The `-b` option determines BLIF as the input format. This option is usually used to perform the simulation process for an Odin-II generated BLIF netlist. Although the BLIF reading process for both Yosys and Odin-II outputs are similar, an extra flag is required to notify Odin-II about the coarse-grained style of the input BLIF file, specified by the `--coarsen` keyword. This flag notifies Odin-II that the BLIF elaboration phase is required to be performed. After the BLIF elaboration, the netlist is partially mapped to the target device architecture specified by the `-a` option. Odin-II decides about the hard block logic based on the target FPGA architecture file. If an architecture file is not specified, Odin-II instantiates all netlist components into soft logic, i.e., pure low-level logic gates. The `--fflegalize` is a new option that is added to Odin-II in our contribution. In the version of VTR (commit `1a00ea9`) we use, VPR always assumes DFFs with rising edge sensitivity. As a result, such an option is required for the Odin-II partial mapper to make all DFFs rising edge.

4.2 Embedded Yosys API

Yosys provides its API under its kernel header file. As a result, an alternative way to integrate Yosys with Odin-II is to embed the Yosys execution inside the

```
# Embedded Yosys API
./odin
    -V HDL circuit designs
    -a fpga_architecture.xml
    -o  fine_grain_netlist .odin. blif
    --elaborator yosys
    --fflegalize
```

Listing 4.3: Odin-II Sample Command to Perform Partial Mapping for a Verilog file with Yosys as the elaborator

Odin-II codebase. For this purpose, we add a new elaboration stage to Odin-II, allocated explicitly for Yosys elaboration. A new command argument `--elaborator` is added to Odin-II, specifying the tool that should perform the elaboration. The new option accepts `odin` and `yosys` as input; however, the elaborator is set to Odin-II by default. Listing 4.3 is a sample command to execute the integrated Yosys API inside Odin-II. Algorithm 2 describes the entire flow of Yosys+Odin-II, including the embedded Yosys API to the Odin-II technology mapped output BLIF file. Once Yosys is selected as the elaborator, its object is created in a new parallel thread, and environment variables are specified. Then, the Yosys program is initialized, VTR primitives are passed to Yosys and required attributes are set. As the next step, Yosys performs the elaboration according to the steps described in the Tcl script shown in Listing 4.1. When the elaboration is completed, and the coarse-grained BLIF file is generated, Odin-II parses the BLIF file and loads it into an in-memory netlist. At this step, Odin-II performs BLIF elaboration and available optimizations. Once the netlist is ready for partial mapping, netlist components are mapped to available hard blocks in the target FPGA architecture file or are exploded into soft logic. The netlist is now compatible with the target FPGA; however, duplicated hard blocks and unused logic may exist. As the last step, Odin-II performs unused logic removal and makes the netlist ready for output.

Algorithm 2: Yosys+Odin-II Synthesizer Routine

```
Yosys+Odin-II_Synthesizer (
  Data: HDL circuit design
  Data: Target FPGA architecture file
  Data: Elaborator (odin or yosys)
  Data: Odin-II available options
  Result: Partial mapped fine-grained netlist in BLIF
)
begin
    if elaborator = odin then
        Parse Verilog files
        Perform AST optimization
        Perform netlist elaboration
    else if elaborator = yosys then
        Setup Yosys
        Initialization and reading baseline files                    /* Yosys */
        Perform elaboration
        Generate coarse-grained BLIF file
        Parse coarse-grained BLIF file                  /* back to Odin-II */
        Perform BLIF elaboration
        Perform netlist optimization
    end
    if Memory hard blocks are available then
        Detect memory hard blocks
        Perform memory hard block optimization
        Perform memory hard block mapping
    end
    Perform partial technology mapping
    if GA-based partial mapper then
        Perform device footprint conformity
    else if Arithmetic hard blocks then
        Perform mixing hard/soft logic mapping
    end
    Perform netlist optimization
    Generate fine-grained BLIF file
end
```

Chapter 5

Results and Analysis

This chapter discusses the Quality of results (QoRs) of Yosys+Odin-II and Yosys front-ends in the VTR flow, working along with Odin-II, the VTR flow's conventional synthesizer. The main idea behind this thesis is to develop the Yosys+Odin-II synthesizer utilizing the Odin-II partial mapper, which can customize the logic inference for both hard and soft logic for the Yosys elaborator. However, we also found Yosys interesting option as the VTR flow front-end. Therefore, we execute the VTR flow over the VTR, Micro and finite impulse response (FIR) benchmarks using Odin-II, Yosys+Odin-II, and Yosys front-ends to evaluate their performance. Moreover, we present the VTR benchmarks' QoR with the Yosys+Odin-II synthesizer that has a customized DSP usage to address one of the main incentives behind the thesis goal. Results are collected on a system with 12 Intel(R) Core i7 CPUs with Kaby Lake microarchitecture and 32 GB of RAM. VTR tools are compiled with GCC 9 with full optimization (-O3). QoRs geomean for all benchmarks are calculated across 10 seeds. Unless otherwise mentioned all QoRs are normalised to the VTR baseline, constitued of Odin-II as the synthesizer.

Three sets of experiments are devised in order to evaluate the Yosys+Odin-II and Yosys CAD tools with the VTR baseline. Each set runs over the VTR bench-

Metrics	Value
Process Technology	40nm
LUT Size (K)	6 (fracturable)
BLE per Logic Block (N)	10
Adder Bits per Logic Block	20
Memory Block Size	32Kb
DSP Block Multiplier Size	36×36 (fracturable)
Switchblock Type	Wilton
Wiretype Length	4

Table 5.1: VTR Modern-Style Flagship Architecture Characteristics

Synthesizer	NPs	CLBs	W_{min}	RW	VTR ET
Odin-II	1.00	1.00	1.00	1.00	1.00
Yosys	0.93	0.95	1.08	0.93	0.75
Yosys+Odin-II	0.96	0.95	1.03	0.96	0.84

NP: Netlist Primitive, **RW**: Routed Wirelength, **ET**: Elapsed Time

Table 5.2: Overview of Synthesizers' Performance over the Heavy Netlist Primitive VTR Benchmarks (geomean over 10 seeds, normalised to Odin-II)

marks through the VTR flow using the `k6_frac_N10_frac_chain_mem32K_40nm`, the modern-style flagship architecture. The architecture is designed in 40nm technology with fracturable 6-LUT that can perform on two 5-LUTs with all five inputs shared. This architecture is the most popular one among VTR users, which closely approximates the commercial Stratix IV FPGA architecture. Table 5.1 describes the characteristics of the modern-style flagship architecture. The primary purpose behind using this architecture is to assess the synthesizers' performance over hard logic inference.

In the following sections, we discuss the VTR benchmarks experiments in detail; however, the results of the Micro and FIR benchmarks are provided in Appendix A and Appendix B, respectively. Table 5.2 briefly compares the performance of all three synthesizers, based on the VTR benchmarks, in the VTR flow.

5.1 Yosys+Odin-II QoR in VTR

The first experiment is performed with the same configuration as the VTR regression QoR test to evaluate the performance of Yosys+Odin-II. Each benchmark is run through the new VTR flow, starting from Yosys to perform coarse-grained synthesis. Odin-II performs partial mapping to the target FPGA architecture on the coarse-grained netlist. Then, the netlist is optimized by ABC, followed by performing packing and placement by VPR. A binary search over channel width is conducted to find the minimum routable channel width (W_{min}). Once routing at the relaxed channel width ($1.3 \cdot W_{min}$) is performed, the critical path delay (CPD) and wire length are measured.

Table 5.3 shows the detailed QoR results on VTR benchmarks using Yosys+Odin-II as the synthesizer. In this study, Yosys is added as an external library to the VTR source tree at the fixed version (commit `b96eb888`). The QoR results in Table 5.3 are relative to the VTR flow version (commit `1a00ea9`) that utilizes only Odin-II as the synthesizer. As Murray et al. [10] suggest, we analyze results for VTR benchmarks with greater than 10K netlist primitives to avoid the skewing effect of small benchmarks on the study's outcome. To perform a fair comparison, we evaluate the VTR benchmarks that achieve more than 10K netlist primitives via all synthesizers. Due to the number of I/O differences for the `stereovision0` benchmark compared to the VTR baseline, we ignore it in the geomean calculation. Although we discuss the Yosys front-end in Section 5.2 it is worth mentioning that the Odin-II synthesizer achieves 2% fewer I/Os compared to the other two synthesizers for VTR benchmarks. Additionally, `stereovision1` is also not considered among the heavy netlist primitive VTR benchmark evaluations for the reasons described in Section 5.2.

As shown in Figure 5.1, the Yosys+Odin-II optimizations have reduced the number of netlist primitives by 4% while the logic depth, defnied by ABC as the largest level of non-output node [23], remained intact. Optimization passes include, but

Benchmark	Netlist Primitives		Clocks	Logic Depth		I/Os		CLBs		DSPs		RAMs		W_{min}		Routed Wirelength (1.3 W_{min})		Routed CPD (1.3 W_{min})		Synth Time (Yosys+Odin-II)		ABC Time		Pack Time		Place Time		Route Time (Find W_{min})		VTR Flow Elapsed Time		Peak Memory		
mcml	178,079.00	(1.07×)	1.00	24.00	(0.92×)	36.00	(1.00×)	7,041.00	(1.00×)	27.00	(1.00×)	159.00	(1.00×)	148.00	(1.06×)	1,051,234.00	(1.05×)	42.37	(0.95×)	99.65	(3.91×)	2,504.37	(1.10×)	208.12	(1.09×)	481.04	(1.06×)	1,171.53	(0.91×)	4,697.09	(1.05×)	2.49	(1.03×)	
LU32PEEng	114,964.00	(1.03×)	1.00	102.00	(1.01×)	114.00	(1.00×)	7,347.00	(0.98×)	32.00	(1.00×)	168.00	(1.01×)	114.00	(0.89×)	1,356,784.00	(0.96×)	74.99	(0.97×)	73.79	(6.64×)	657.14	(1.10×)	194.43	(1.02×)	303.14	(0.96×)	3,288.73	(0.45×)	4,766.45	(0.54×)	1.96	(0.98×)	
stereovision2	38,455.00	(0.90×)	1.00	3.00	(1.00×)	149.00	(1.00×)	1,519.00	(0.92×)	302.00	(0.93×)			96.00	(1.20×)	438,120.00	(0.92×)	17.07	(1.07×)	3.62	(2.94×)	3.44	(0.67×)	15.17	(0.69×)	35.15	(0.85×)	422.20	(0.88×)	626.45	(0.89×)	1.44	(0.97×)	
LU8PEEng	34,679.00	(1.02×)	1.00	100.00	(0.99×)	114.00	(1.00×)	2,150.00	(0.98×)	8.00	(1.00×)	45.00	(1.02×)	98.00	(1.02×)	333,054.00	(0.94×)	76.35	(0.99×)	19.03	(7.74×)	65.99	(1.02×)	59.42	(1.09×)	32.81	(0.87×)	218.57	(0.67×)	456.85	(0.83×)	0.58	(0.98×)	
bgm	34,595.00	(1.02×)	1.00	14.00	(1.00×)	257.00	(1.00×)	2,662.00	(0.97×)	11.00	(1.00×)			76.00	(1.03×)	383,081.00	(0.96×)	19.28	(0.98×)	21.62	(7.35×)	138.04	(0.85×)	61.16	(1.24×)	35.88	(0.70×)	141.74	(1.00×)	462.54	(0.96×)	0.66	(0.98×)	
stereovision0	21,554.00	(0.99×)	1.00	5.00	(1.00×)	169.00	(1.08×)	698.00	(0.99×)					50.00	(1.00×)	62,403.00	(1.00×)	3.99	(1.08×)	3.18	(2.92×)	4.51	(0.81×)	6.47	(1.00×)	6.07	(1.08×)	14.49	(1.02×)	47.40	(1.00×)	0.23	(0.98×)	
stereovision1	19,355.00	(0.99×)	1.00	3.00	(1.00×)	115.00	(1.00×)	697.00	(1.03×)	40.00	(1.00×)			84.00	(1.02×)	135,161.00	(1.00×)	5.96	(1.13×)	3.36	(3.43×)	6.83	(0.25×)	6.85	(1.01×)	7.04	(0.96×)	78.33	(0.91×)	121.99	(0.81×)	0.28	(0.97×)	
arm_core	14,087.00	(0.78×)	1.00	20.00	(1.11×)	133.00	(1.00×)	869.00	(0.86×)			25.00	(1.04×)	114.00	(1.02×)	183,533.00	(0.91×)	22.78	(1.24×)	8.77	(7.97×)	31.22	(0.60×)	28.72	(1.42×)	10.79	(0.90×)	120.44	(0.98×)	224.17	(0.89×)	0.24	(0.88×)	
blob_merge	10,163.00	(0.90×)	1.00	5.00	(1.00×)	36.00	(1.00×)	514.00	(0.83×)					62.00	(1.00×)	60,762.00	(0.82×)	14.98	(1.00×)	1.99	(5.85×)	4.19	(0.13×)	10.49	(0.85×)	4.23	(0.84×)	21.66	(0.75×)	51.89	(0.58×)	0.15	(0.89×)	
or1200	4,658.00	(1.03×)	1.00	9.00	(1.13×)	385.00	(1.00×)	252.00	(0.98×)	1.00	(1.00×)	2.00	(1.00×)	88.00	(0.90×)	48,887.00	(1.06×)	8.92	(1.04×)	2.06	(6.44×)	2.91	(0.80×)	5.82	(0.97×)	3.77	(0.95×)	35.30	(1.36×)	56.55	(1.20×)	0.11	(1.00×)	
mkDelayWorker32B	3,892.00	(0.97×)	1.00	5.00	(1.00×)	507.00	(1.00×)	607.00	(1.33×)			43.00	(0.96×)	38.00	(1.00×)	21,435.00	(0.93×)	7.18	(0.95×)	4.31	(6.84×)	5.13	(1.03×)	3.61	(0.96×)	4.50	(1.03×)	19.54	(0.94×)	60.87	(1.02×)	0.32	(1.00×)	
raygentop	2,960.00	(1.03×)	1.00	4.00	(1.33×)	236.00	(1.10×)	117.00	(1.05×)	6.00	(0.75×)	1.00		66.00	(1.14×)	22,170.00	(0.97×)	5.57	(1.17×)	1.46	(6.64×)	0.60	(0.81×)	2.30	(1.21×)	1.53	(1.26×)	7.77	(1.13×)	16.98	(1.17×)	0.07	(1.08×)	
mkSMAdapter4B	2,720.00	(0.98×)	1.00	5.00	(1.25×)	193.00	(1.00×)	172.00	(0.92×)			5.00	(1.00×)	58.00	(1.07×)	18,223.00	(0.81×)	5.67	(1.11×)	1.73	(9.11×)	1.46	(1.00×)	2.51	(1.22×)	1.73	(0.91×)	6.53	(0.75×)	17.18	(0.97×)	0.06	(0.98×)	
sha	2,507.00	(0.91×)	1.00	5.00	(1.67×)	38.00	(1.00×)	130.00	(0.83×)					64.00	(0.97×)	12,267.00	(0.73×)	9.82	(0.98×)	0.61	(0.95×)	1.15	(0.01×)	1.36	(0.83×)	0.83	(0.78×)	4.18	(0.91×)	10.42	(0.06×)	0.06	(0.86×)	
boundtop	1,227.00	(1.08×)	1.00	3.00	(1.00×)	196.00	(1.38×)	38.00	(0.41×)			1.00		42.00	(1.00×)	5,414.00	(1.40×)	2.33	(0.67×)	6.65	(20.78×)	0.47	(1.47×)	0.48	(0.86×)	0.46	(0.87×)	2.00	(1.29×)	11.58	(2.19×)	0.04	(1.02×)	
mkPktMerge	1,158.00	(1.00×)	1.00	2.00	(1.00×)	311.00	(1.00×)	30.00	(1.03×)			15.00	(1.00×)	32.00	(0.89×)	15,343.00	(1.03×)	4.39	(1.04×)	0.43	(5.38×)	0.07	(1.17×)	0.47	(1.07×)	0.78	(1.01×)	9.54	(1.34×)	15.50	(1.20×)	0.07	(1.00×)	
spree	1,123.00	(0.91×)	1.00	17.00	(1.13×)	45.00	(1.00×)	59.00	(0.91×)	1.00	(1.00×)	3.00	(1.00×)	64.00	(0.89×)	12,687.00	(1.02×)	10.78	(0.93×)	1.04	(9.45×)	0.42	(0.74×)	1.47	(0.77×)	0.51	(0.96×)	5.89	(0.87×)	11.16	(0.94×)	0.04	(0.95×)	
diffeq1	888.00	(1.00×)	1.00	6.00	(1.00×)	162.00	(1.00×)	32.00	(1.00×)	5.00	(1.00×)			50.00	(0.81×)	10,996.00	(1.07×)	17.36	(1.01×)	0.09	(4.50×)	0.12	(0.92×)	0.25	(0.89×)	0.48	(0.98×)	4.43	(0.81×)	7.07	(0.86×)	0.04	(0.98×)	
diffeq2	595.00	(0.98×)	1.00	6.00	(1.00×)	66.00	(1.00×)	22.00	(1.10×)	5.00	(0.71×)			66.00	(1.22×)	7,617.00	(0.79×)	12.47	(0.94×)	0.06	(6.00×)	0.06	(0.75×)	0.30	(1.00×)	0.34	(0.85×)	5.77	(0.93×)	8.22	(0.89×)	0.05	(1.03×)	
ch_intrinsics	462.00	(0.94×)	1.00	3.00	(1.00×)	99.00	(1.00×)	70.00	(1.08×)			1.00	(1.00×)	44.00	(0.96×)	1,356.00	(0.96×)	1.90	(0.71×)	0.12	(4.00×)	0.05	(0.28×)	0.08	(0.67×)	0.20	(1.00×)	0.75	(0.94×)	1.98	(0.90×)	0.03	(1.00×)	
stereovision3	334.00	(1.04×)	2.00	5.00	(1.00×)	11.00	(1.00×)	16.00	(1.14×)					30.00	(0.83×)	1,022.00	(1.21×)	2.62	(0.99×)	0.24	(8.00×)	0.11	(1.00×)	0.19	(1.00×)	0.07	(1.17×)	0.36	(1.16×)	1.55	(1.19×)	0.03	(1.01×)	
GEOMEAN	5,428.93	(0.98×)		7.67	(1.06×)	114.75	(1.02×)	277.75	(0.95×)	10.30	(0.94×)	9.56	(1.00×)	64.79	(0.99×)	40,152.52	(0.97×)	10.20	(0.99×)	2.18	(5.58×)	2.75	(0.58×)	3.84	(0.98×)	3.44	(0.94×)	22.09	(0.93×)	51.07	(0.84×)	0.16	(0.98×)	
GEOMEAN (>10K)	48,678.55	(0.96×)		24.29	(1.00×)	115.57	(1.00×)	2,703.93	(0.95×)	29.66	(0.99×)	74.04	(1.02×)	105.45	(1.03×)	494,561.65	(0.96×)	34.94	(1.03×)	21.40	(5.71×)	108.26	(0.86×)	63.26	(1.07×)	63.43	(0.88×)	427.10	(0.78×)	934.12	(0.84×)	0.93	(0.97×)	†
% TOTAL (> 10K)																				2.02%		30.27%		5.05%		8.00%		47.74%		100.0%			

Runtime in Seconds, Memory in GiB, WL in grid tiles, and CPD in ns.

† `stereovision0`, `stereovision1`, and `blob_merge` are excluded

Table 5.3: VTR Benchmarks using Yosys+Odin-II Front-end in the VTR Flow (Relative to the VTR Baseline).

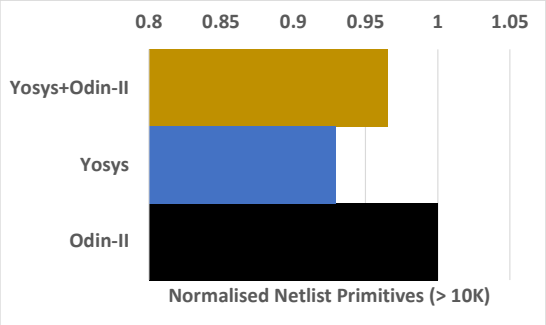


Figure 5.1: Netlist Primitives for VTR Benchmarks with all Front-ends.

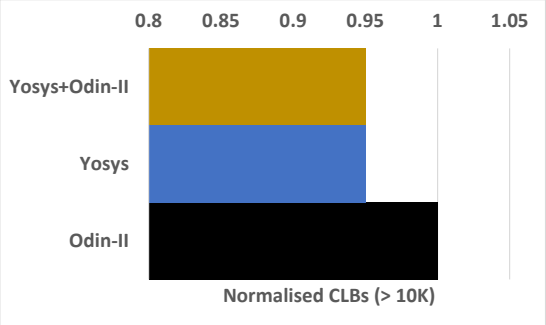


Figure 5.2: CLBs for VTR Benchmarks with all Front-ends.

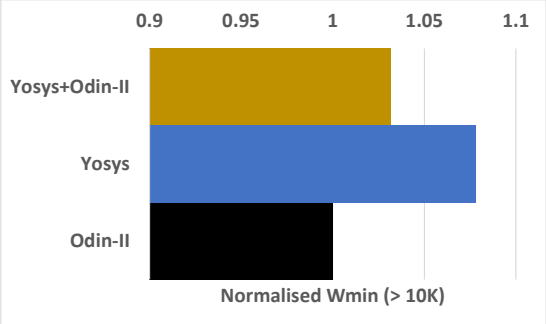


Figure 5.3: W_*min* for VTR Benchmarks with all Front-ends.

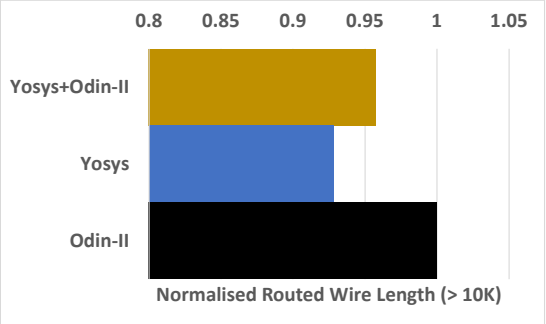


Figure 5.4: Routed Wire Lengths for VTR Benchmarks with all Front-ends.

are not limited to, Yosys `opt_muxtree` to remove dead branches of decision trees, Yosys `opt_expr` and embedded optimizations in the Odin-II BLIF elaboration to perform constant folding, Yosys `opt_clean` and Odin-II unused logic removal to identify and clean up unused wires and cells, and Yosys `opt_merge` that performs resource sharing by merging cells with identical inputs into a single instance. VTR benchmarks with the identical number of I/Os to the VTR baseline are chosen to participate in the QoR comparison. Figure 5.2 illustrates that the number of CLBs has decreased by 5% showing the Yosys+Odin-II synthesizer distinctly improves the device utilization. As depicted in Figures 5.3 and 5.4, with fewer CLBs and 4% less routed wire lengths, while the minimum routable channel width increased by 3%, show that the Yosys+Odin-II synthesized circuits are even more compact.

All synthesis criteria are changed in a reasonable range while the overall VTR flow runtime has reduced considerably. The synthesis time includes the Yosys elaboration,

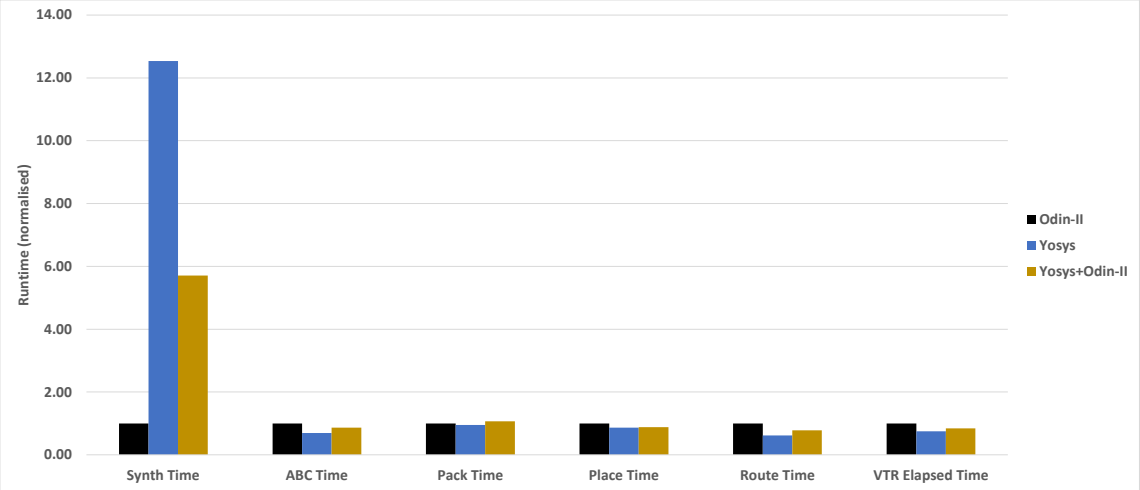


Figure 5.5: VTR Flow Sectional Runtime for VTR Benchmarks with Yosys+Odin Front-end

the Odin-II BLIF reading, and the Odin-II partial mapping processes. Compared to the baseline, the Yosys+Odin-II synthesis time increased by 5.71×. The time augmentation is mainly due to the Yosys elaboration phase. Furthermore, an additional BLIF reading process is added to the synthesis flow, which for large benchmarks could result in a substantial time increase. As shown in Table 5.3, the `stereovision2`, a computer vision application domain benchmark, achieved the minimum synthesis time gain by 2.94×, among benchmarks with more than 10K netlist primitives. On the other hand, the `arm_core` benchmark, a soft processor, achieved the maximum synthesis time 7.97× greater than the baseline. The synthesis time still comprises a small portion of the entire VTR flow runtime, i.e., 2.02%. As a result, the substantial increase in the synthesis time is negligible among the rest of the VTR CAD tools' runtime.

Due to the optimization passes in Yosys, the ABC time is considerably reduced by 14%, mainly accounting for the overall VTR flow runtime reduction. Among the heavy netlist primitive benchmarks, the `arm_core`, achieved the best improvement by 40% ABC time reduction. However, the `mcml`, a medical physics application domain benchmark, and `LU32PEEng`, a mathematical benchmark with extensive RAM usage,

require 10% more than the baseline's ABC optimization time. In terms of placement and routing, the synthesized circuits' quality is enhanced compared to the baseline. As shown in Figure 5.5, the Yosys elaboration improves the placement time by 12%, and decreases the routing time by 22% while the placement time increases by 7%. The overall VTR flow runtime is reduced by 16%, mainly due to the ABC and routing time reductions. The peak memory consumption of the VTR flow is also slightly reduced by 3%.

5.2 Yosys QoR in VTR

The Yosys QoR experiment follows the same approach as the first one but with Yosys as the VTR flow front-end instead of Yosys+Odin-II. Yosys performs the elaboration and synthesis on VTR benchmarks based on a slightly modified script than the one Hung proposed for VTR-to-Bitstream [59, 60]. The ABC commands are removed from the Yosys synthesis script. Instead, the ABC stage of VTR performs the optimization and LUT mapping. The LUT width is considered the Yosys default value. A simple addition operation compatible with Intel architectures is replaced with the `xadder` implementation, specifically proposed for Xilinx architectures by Hung. The other models proposed for Yosys remained intact; however, we added a couple of Verilog modules that perform depth splitting for VTR memory hard blocks. According to these implementations, Yosys recursively splits `SPRAM`s and `DPRAM`s, which have an address port size higher than that specified in the given FPGA layout. The modified version of the Yosys script proposed by Hung is presented in Appendix C. It is worth noting that the same Yosys+Odin-II process to eliminate asynchronous DFFs has been added to the script. The above mentioned changes mostly are included in: `single_port_ram.v` and `dual_port_ram.v` library files to recursively perform depth splitting; `adffe2dff.v` to handle DFFs with asynchronous reset and data enable

signals; `spram_rename.v` and `dpram_rename.v` files for memory hard block renaming process; and the Yosys synthesis script for adding `techmap` and `dffunmap` commands. Table 5.4 elaborates the results of the second experiment, assessing the Yosys synthesis capabilities in the complete VTR flow. We again evaluate benchmarks with a minimum of 10K netlist primitives. However, it is necessary to mention that among the heavy netlist primitive VTR benchmarks the `stereovision1` benchmark shows outlier behaviour using the Yosys synthesizer. This is mainly due to a 3.8× increase in DSP usage, while the minimum routable channel width decreases by more than 50%. The substantial increase in DSP usage results in longer routing paths that cause a more than 27% decrease in the circuit's operating frequency. Consequently, to avoid the skewing effect and to perform a fair comparison among all synthesizers, the `stereovision1` benchmark is neither included in the geomean calculation nor considered among VTR benchmarks with more than 10K netlist primitives in all comparisons discussed in this thesis. As mentioned earlier, to have the comparison of the same set of benchmarks, the `stereovision0` benchmark is also excluded from the heavy netlist primitives VTR benchmarks.

The Yosys synthesizer results have significant QoR changes compared to the VTR baseline. As with Yosys+Odin-II, the number of I/Os across of the heavy netlist primitive VTR benchmarks slightly increased by 2%, compared to the VTR baseline. However, we exclude benchmarks with different numbers of I/Os, as mentioned earlier. The number of pre-packed nets has reduced by 7% for the heavy netlist primitive VTR benchmarks. In spite of the Yosys+Odin-II synthesizer that has not changed the logic depth, Yosys increases it by 7% compared to the VTR baseline. In terms of hard blocks, Yosys utilizes 7% more multiplier DSPs while the hard memory usage decreases by 11%. Since hard memory blocks are explicitly declared in benchmarks, we believe that less RAM hard block usage is caused by how Yosys infers implicit memories based upon the configuration. That means the taken ap-

Benchmark	Netlist Primitives		Clocks	Logic Depth		I/Os		CLBs		DSPs		RAMs		W_{min}		Routed Wirelength ($1.3\,W_{min}$)		Routed CPD ($1.3\,W_{min}$)		Synth Time (Yosys)		ABC Time		Pack Time		Place Time		Route Time (Find W_{min})		VTR Flow Elapsed Time		Peak Memory	
mcml	159,112.00	(0.96×)	1.00	25.00	(0.96×)	36.00	(1.00×)	6,352.00	(0.90×)	26.00	(0.96×)	97.00	(0.61×)	142.00	(1.01×)	954,418.00	(0.96×)	45.28	(1.01×)	211.32	(8.29×)	2,626.23	(1.16×)	125.05	(0.66×)	392.24	(0.86×)	958.87	(0.75×)	4,531.60	(1.01×)	2.28	(0.95×)
LU32PEEng	109,836.00	(0.99×)	1.00	96.00	(0.95×)	114.00	(1.00×)	7,556.00	(1.01×)	32.00	(1.00×)	168.00	(1.01×)	146.00	(1.14×)	1,379,641.00	(0.98×)	74.87	(0.97×)	180.85	(16.26×)	497.59	(0.83×)	195.62	(1.02×)	273.50	(0.87×)	951.01	(0.13×)	2,377.33	(0.27×)	2.34	(1.18×)
stereovision2	39,500.00	(0.93×)	1.00	5.00	(1.67×)	149.00	(1.00×)	1,630.00	(0.99×)	480.00	(1.48×)			98.00	(1.23×)	389,683.00	(0.82×)	14.66	(0.92×)	8.17	(6.64×)	4.87	(0.95×)	15.38	(0.70×)	36.39	(0.88×)	342.05	(0.71×)	628.51	(0.89×)	2.08	(1.40×)
bgm	33,457.00	(0.99×)	1.00	14.00	(1.00×)	257.00	(1.00×)	2,679.00	(0.98×)	11.00	(1.00×)			76.00	(1.03×)	397,723.00	(1.00×)	20.52	(1.04×)	40.46	(13.76×)	47.17	(0.29×)	56.37	(1.14×)	42.85	(0.84×)	157.75	(1.12×)	413.89	(0.86×)	0.65	(0.97×)
LU8PEEng	32,373.00	(0.96×)	1.00	92.00	(0.91×)	114.00	(1.00×)	2,173.00	(0.99×)	8.00	(1.00×)	45.00	(1.02×)	100.00	(1.04×)	332,805.00	(0.94×)	72.28	(0.94×)	51.01	(20.74×)	49.19	(0.76×)	53.91	(0.99×)	34.63	(0.92×)	210.76	(0.64×)	465.02	(0.84×)	0.59	(0.99×)
stereovision0	21,171.00	(0.97×)	1.00	5.00	(1.00×)	169.00	(1.08×)	695.00	(0.99×)					56.00	(1.12×)	62,822.00	(1.00×)	3.80	(1.03×)	6.40	(5.87×)	5.00	(0.89×)	6.85	(1.06×)	5.45	(0.97×)	17.19	(1.21×)	53.95	(1.14×)	0.23	(0.97×)
stereovision1	19,132.00	(0.98×)	1.00	3.00	(1.00×)	115.00	(1.00×)	668.00	(0.99×)	152.00	(3.80×)			40.00	(0.49×)	156,061.00	(1.16×)	7.24	(1.37×)	5.18	(5.29×)	7.59	(0.28×)	6.09	(0.89×)	8.85	(1.20×)	113.44	(1.31×)	200.67	(1.34×)	0.69	(2.38×)
arm_core	14,161.00	(0.78×)	1.00	20.00	(1.11×)	133.00	(1.00×)	857.00	(0.85×)			24.00	(1.00×)	116.00	(1.04×)	178,346.00	(0.89×)	19.58	(1.06×)	16.67	(15.15×)	29.15	(0.56×)	28.92	(1.43×)	9.78	(0.82×)	135.80	(1.10×)	257.73	(1.03×)	0.24	(0.88×)
blob_merge	7,617.00	(0.68×)	1.00	5.00	(1.00×)	36.00	(1.00×)	485.00	(0.79×)					70.00	(1.13×)	71,714.00	(0.96×)	15.35	(1.03×)	9.31	(27.38×)	3.78	(0.12×)	14.16	(1.15×)	3.84	(0.76×)	22.20	(0.76×)	63.13	(0.70×)	0.15	(0.92×)
or1200	4,318.00	(0.95×)	1.00	8.00	(1.00×)	385.00	(1.00×)	241.00	(0.93×)	1.00	(1.00×)	2.00	(1.00×)	98.00	(1.00×)	45,291.00	(0.98×)	9.12	(1.07×)	3.77	(11.78×)	2.83	(0.78×)	5.16	(0.86×)	3.75	(0.95×)	25.32	(0.98×)	47.71	(1.02×)	0.11	(0.98×)
mkDelayWorker32B	3,728.00	(0.93×)	1.00	5.00	(1.00×)	506.00	(1.00×)	459.00	(1.01×)			43.00	(0.96×)	40.00	(1.05×)	23,797.00	(1.03×)	7.16	(0.95×)	10.24	(16.25×)	2.21	(0.45×)	3.70	(0.98×)	4.11	(0.94×)	20.87	(1.01×)	66.48	(1.11×)	0.32	(1.00×)
raygentop	3,036.00	(1.06×)	1.00	3.00	(1.00×)	236.00	(1.10×)	125.00	(1.13×)	18.00	(2.25×)	1.00		52.00	(0.90×)	28,465.00	(1.24×)	6.00	(1.26×)	2.36	(10.73×)	0.56	(0.76×)	1.66	(0.87×)	1.60	(1.32×)	11.43	(1.66×)	22.81	(1.57×)	0.08	(1.24×)
mkSMAdapter4B	2,699.00	(0.97×)	1.00	5.00	(1.25×)	193.00	(1.00×)	176.00	(0.94×)			5.00	(1.00×)	56.00	(1.04×)	19,498.00	(0.86×)	5.47	(1.07×)	3.50	(18.42×)	1.38	(0.95×)	2.17	(1.06×)	1.70	(0.89×)	10.48	(1.20×)	22.74	(1.28×)	0.06	(0.98×)
sha	2,561.00	(0.93×)	1.00	3.00	(1.00×)	38.00	(1.00×)	133.00	(0.85×)					82.00	(1.24×)	12,738.00	(0.76×)	10.37	(1.04×)	1.59	(2.48×)	1.07	(0.01×)	1.41	(0.86×)	0.96	(0.90×)	4.49	(0.97×)	12.07	(0.06×)	0.06	(0.96×)
boundtop	1,499.00	(1.31×)	1.00	3.00	(1.00×)	196.00	(1.38×)	50.00	(0.54×)			1.00		38.00	(0.90×)	5,816.00	(1.51×)	3.58	(1.03×)	3.34	(10.44×)	0.60	(1.88×)	0.63	(1.13×)	0.58	(1.09×)	2.05	(1.32×)	9.07	(1.72×)	0.04	(1.07×)
spree	1,160.00	(0.94×)	1.00	16.00	(1.07×)	45.00	(1.00×)	58.00	(0.89×)	1.00	(1.00×)	3.00	(1.00×)	64.00	(0.89×)	12,188.00	(0.98×)	11.21	(0.96×)	1.48	(13.45×)	0.39	(0.68×)	1.57	(0.83×)	0.49	(0.92×)	4.44	(0.65×)	10.40	(0.88×)	0.04	(0.94×)
mkPktMerge	1,153.00	(0.99×)	1.00	2.00	(1.00×)	311.00	(1.00×)	28.00	(0.97×)			15.00	(1.00×)	36.00	(1.00×)	14,945.00	(1.00×)	4.13	(0.98×)	0.91	(11.38×)	0.13	(2.17×)	0.47	(1.07×)	0.80	(1.04×)	7.56	(1.06×)	14.75	(1.14×)	0.07	(1.00×)
diffeq1	951.00	(1.08×)	1.00	8.00	(1.33×)	162.00	(1.00×)	31.00	(0.97×)	3.00	(0.60×)			62.00	(1.00×)	8,998.00	(0.87×)	15.69	(0.92×)	0.28	(14.00×)	0.14	(1.08×)	0.32	(1.14×)	0.49	(1.00×)	4.25	(0.77×)	7.02	(0.85×)	0.04	(0.94×)
diffeq2	706.00	(1.16×)	1.00	9.00	(1.50×)	66.00	(1.00×)	31.00	(1.55×)	3.00	(0.43×)			66.00	(1.22×)	7,200.00	(0.75×)	12.64	(0.95×)	0.23	(23.00×)	0.17	(2.13×)	0.33	(1.10×)	0.39	(0.98×)	3.92	(0.63×)	6.55	(0.71×)	0.04	(0.95×)
ch_intrinsics	477.00	(0.97×)	1.00	3.00	(1.00×)	99.00	(1.00×)	73.00	(1.12×)			1.00	(1.00×)	40.00	(0.87×)	1,332.00	(0.94×)	1.98	(0.74×)	0.18	(6.00×)	0.04	(0.22×)	0.13	(1.08×)	0.22	(1.10×)	0.89	(1.11×)	2.40	(1.09×)	0.03	(1.01×)
stereovision3	331.00	(1.03×)	2.00	5.00	(1.00×)	11.00	(1.00×)	16.00	(1.14×)					34.00	(0.94×)	682.00	(0.80×)	2.92	(1.11×)	0.34	(11.33×)	0.09	(0.82×)	0.28	(1.47×)	0.06	(1.00×)	0.26	(0.84×)	1.63	(1.25×)	0.03	(1.01×)
GEOMEAN	5,394.03	(0.97×)		7.72	(1.07×)	114.74	(1.02×)	280.72	(0.96×)	12.17	(1.11×)	9.14	(0.95×)	65.26	(1.00×)	39,841.36	(0.96×)	10.46	(1.01×)	4.38	(11.20×)	2.66	(0.56×)	3.95	(1.00×)	3.48	(0.96×)	20.76	(0.87×)	52.12	(0.86×)	0.17	(1.05×)
GEOMEAN (>10K)	46,859.91	(0.93×)		26.00	(1.07×)	115.57	(1.00×)	2,703.34	(0.95×)	32.30	(1.07×)	64.77	(0.89×)	110.20	(1.08×)	479,227.45	(0.93×)	33.61	(0.99×)	46.97	(12.53×)	86.89	(0.69×)	56.65	(0.95×)	61.97	(0.86×)	334.80	(0.61×)	833.74	(0.75×)	1.01	(1.05×) †
% TOTAL (> 10K)																				5.86%		37.52%		5.48%		9.10%		31.78%		100.00%			

Runtime in Seconds, Memory in GiB, WL in grid tiles, and CPD in ns.

† `stereovision0` and `stereovision1` are excluded

Table 5.4: VTR Benchmarks using Yosys Front-end in the VTR Flow (Relative to the VTR Baseline).

proach results in more register usage instead of hard memory blocks. As the circuits' areas are decreased, the CLB usage is reduced by 5%. The minimum routable channel width increases by 8%, while the total routed wire length is decreased by 7%. In general, critical path delays are improved by 1%, whereas the `stereovision2` benchmark achieved the maximum frequency enhancement, by 9%, among all VTR benchmarks.

The VTR flow with Yosys synthesizer performs efficiently concerning the overall VTR flow elapsed time. Comparing to Yosys+Odin that has an extra BLIF reading process, the synthesis time is increased by 219%, roughly 12.53× greater than Odin-II. The ABC runtime significantly improved, executing 31% faster than the VTR baseline and 19% quicker than Yosys+Odin-II. At first glance, we could guess that the ABC runtime reduction primarily results from fine-grained optimization passes performed by Yosys. VPR takes less time to perform packing, placement and relaxed channel width routing compared to the VTR baseline by 0.95×, 0.86×, and 0.61×, respectively. Overall, the ABC optimization time reduction and shorter routing time, along with the absence of an extra BLIF reading process result in about a 10% decrease in the VTR flow runtime using Yosys compared to Yosys+Odin-II. Regarding memory consumption, VTR with Yosys ends up with 5% greater peak memory than the VTR baseline and 8% higher than the VTR flow Yosys+Odin-II synthesizer.

5.3 Yosys+Odin-II QoR in VTR (Adjusted DSP Usage)

The third experiment is devised in order to evaluate the functionality of one of the Odin-II partial mapper abilities to perform hard/soft logic trade-offs. Technically, the third experiment configuration should force Yosys+Odin-II to infer at mini-

mum half of multiplication operations as soft logic even if multiplier hard blocks are available. Table 5.5 elaborates the results of the second experiment, assessing the hard/soft logic trade-off decision capability of Yosys+Odin-II partial mapper. It is worth mentioning that we again evaluate benchmarks with a minimum of 10K netlist primitives. While the number of I/Os and hard memory blocks remain unchanged, the rest of the synthesis criteria increase, apart from DSP usage and relaxed channel width routing time. The results show that the partial mapper softens 57% of multipliers, while the netlist pre-packed blocks and logic depth increase by 17% and 64%, respectively. Technically, by exploding a portion of multiplication operations into soft logic, such an area growth is expected. Regarding the logic depth, `raygentop`, `or1200`, `diffeq1` and `diffeq2` achieve the most logic depth increase. Due to the nature of these benchmarks that have either a chain of multipliers or large multiplication operations, softening multiplier hard blocks increases the synthesized circuits' depth considerably. As the result of netlist primitives increase, VTR uses 23% more CLBs. While the routed wire length increases by 8%, the minimum routable channel width remains unchanged for the heavy netlist primitive VTR benchmarks. Except for the `stereovision2` benchmark, using more soft logic results in the overall critical path delay increase by 9%. Interestingly, the overhead of long routing paths in a pure hard logic implementation overcomes the latency of soft logic performance for the benchmark mentioned above. Therefore, utilizing soft logic implementation of a minimum of 50% of multiplication operations causes shorter critical path delay for the `stereovision2` benchmark.

Figure 5.6 depicts the achieved percentage of DSP block reduction by the Yosys+Odin-II partial mapping optimizer. The `or1200` and `spree` have only one multiplication hard block originally; therefore, they achieve 100% DSP reduction according to the experiment configuration. Among the rest of the benchmarks, `stereovision2`, a computer vision benchmark with extensive fixed-point multiplication, achieves the

Benchmark	Netlist Primitives		Clocks	Logic Depth		I/Os		CLBs		DSPs		RAMs		W_{min}		Routed Wirelength ($1.3\ W_{min}$)		Routed CPD ($1.3\ W_{min}$)		Synth Time (Yosys+Odin-II))		ABC Time		Pack Time		Place Time		Route Time (Find W_{min})		VTR Flow Elapsed Time		Peak Memory	
mcml	209,059.00	(1.17×)	1.00	53.00	(2.21×)	36.00	(1.00×)	9,324.00	(1.32×)	13.00	(0.48×)	159.00	(1.00×)	146.00	(0.99×)	1,283,070.00	(1.22×)	51.17	(1.21×)	112.02	(1.12×)	3,450.75	(1.38×)	224.64	(1.08×)	484.31	(1.01×)	699.87	(0.60×)	5,264.00	(1.12×)	3.18	(1.28×)
LU32PEEng	133,751.00	(1.16×)	1.00	102.00	(1.00×)	114.00	(1.00×)	8,380.00	(1.14×)	16.00	(0.50×)	168.00	(1.00×)	122.00	(1.07×)	1,549,607.00	(1.14×)	75.79	(1.01×)	78.17	(1.06×)	1,168.25	(1.78×)	226.28	(1.16×)	356.84	(1.18×)	1,412.26	(0.43×)	3,549.07	(0.74×)	2.57	(1.31×)
stereovision2	52,901.00	(1.38×)	1.00	12.00	(4.00×)	149.00	(1.00×)	2,455.00	(1.62×)	68.00	(0.23×)	0.00		100.00	(1.04×)	375,423.00	(0.86×)	15.07	(0.88×)	5.73	(1.58×)	21.32	(6.20×)	54.87	(3.62×)	75.54	(2.15×)	180.67	(0.43×)	424.58	(0.68×)	0.80	(0.56×)
bgm	42,378.00	(1.22×)	1.00	37.00	(2.64×)	257.00	(1.00×)	3,190.00	(1.20×)	5.00	(0.45×)	0.00		76.00	(1.00×)	438,027.00	(1.14×)	27.60	(1.43×)	25.47	(1.18×)	236.03	(1.71×)	74.29	(1.21×)	62.47	(1.74×)	92.32	(0.65×)	575.46	(1.24×)	0.79	(1.20×)
LU8PEEng	39,540.00	(1.14×)	1.00	102.00	(1.02×)	114.00	(1.00×)	2,419.00	(1.13×)	4.00	(0.50×)	45.00	(1.00×)	90.00	(0.92×)	377,994.00	(1.13×)	76.40	(1.00×)	20.34	(1.07×)	103.06	(1.56×)	69.74	(1.17×)	62.60	(1.91×)	566.37	(2.59×)	902.76	(1.98×)	0.65	(1.12×)
stereovision1	29,653.00	(1.53×)	1.00	11.00	(3.67×)	115.00	(1.00×)	1,371.00	(1.97×)	20.00	(0.50×)	0.00		86.00	(1.02×)	175,201.00	(1.30×)	8.30	(1.39×)	4.67	(1.39×)	69.86	(10.23×)	19.41	(2.83×)	20.98	(2.98×)	79.77	(1.02×)	227.16	(1.86×)	0.41	(1.46×)
stereovision0	21,554.00	(1.00×)	1.00	5.00	(1.00×)	169.00	(1.00×)	698.00	(1.00×)	0.00		0.00		50.00	(1.00×)	62,403.00	(1.00×)	3.99	(1.00×)	3.99	(1.25×)	7.27	(1.61×)	7.32	(1.13×)	7.52	(1.24×)	14.42	(1.00×)	55.24	(1.17×)	0.23	(0.99×)
arm_core	14,087.00	(1.00×)	1.00	20.00	(1.00×)	133.00	(1.00×)	869.00	(1.00×)	0.00		25.00	(1.00×)	114.00	(1.00×)	183,533.00	(1.00×)	22.78	(1.00×)	10.69	(1.22×)	42.78	(1.37×)	32.72	(1.14×)	13.14	(1.22×)	171.04	(1.42×)	299.18	(1.33×)	0.29	(1.18×)
blob_merge	10,163.00	(1.00×)	1.00	5.00	(1.00×)	36.00	(1.00×)	514.00	(1.00×)	0.00		0.00		62.00	(1.00×)	60,762.00	(1.00×)	14.98	(1.00×)	2.61	(1.31×)	5.25	(1.25×)	11.45	(1.09×)	5.08	(1.20×)	16.97	(0.78×)	52.14	(1.00×)	0.14	(0.99×)
or1200	7,272.00	(1.56×)	1.00	53.00	(5.89×)	385.00	(1.00×)	452.00	(1.79×)	0.00	(0.00×)	2.00	(1.00×)	86.00	(0.98×)	73,205.00	(1.50×)	37.80	(4.24×)	2.47	(1.20×)	6.33	(2.18×)	7.70	(1.32×)	7.85	(2.08×)	19.09	(0.54×)	53.17	(0.94×)	0.13	(1.16×)
raygentop	4,843.00	(1.64×)	1.00	24.00	(6.00×)	236.00	(1.00×)	249.00	(2.13×)	3.00	(0.50×)	1.00	(1.00×)	62.00	(0.94×)	34,043.00	(1.54×)	16.62	(2.99×)	1.79	(1.23×)	2.94	(4.90×)	4.39	(1.91×)	2.92	(1.91×)	6.31	(0.81×)	23.86	(1.41×)	0.09	(1.28×)
mkDelayWorker32B	3,892.00	(1.00×)	1.00	5.00	(1.00×)	507.00	(1.00×)	607.00	(1.00×)	0.00		43.00	(1.00×)	38.00	(1.00×)	21,435.00	(1.00×)	7.18	(1.00×)	5.25	(1.22×)	8.36	(1.63×)	3.93	(1.09×)	5.28	(1.17×)	11.77	(0.60×)	63.54	(1.04×)	0.32	(1.00×)
diffeq2	3,145.00	(5.29×)	1.00	34.00	(5.67×)	66.00	(1.00×)	214.00	(9.73×)	2.00	(0.40×)	0.00		62.00	(0.94×)	28,457.00	(3.74×)	31.86	(2.55×)	0.09	(1.50×)	2.18	(36.33×)	3.01	(10.03×)	1.98	(5.82×)	6.53	(1.13×)	18.18	(2.21×)	0.07	(1.57×)
mkSMAdapter4B	2,720.00	(1.00×)	1.00	5.00	(1.00×)	193.00	(1.00×)	172.00	(1.00×)	0.00		5.00	(1.00×)	58.00	(1.00×)	18,223.00	(1.00×)	5.67	(1.00×)	2.22	(1.28×)	1.61	(1.10×)	2.74	(1.09×)	1.98	(1.14×)	5.38	(0.82×)	17.65	(1.03×)	0.06	(0.99×)
sha	2,507.00	(1.00×)	1.00	5.00	(1.00×)	38.00	(1.00×)	130.00	(1.00×)	0.00		0.00		64.00	(1.00×)	12,267.00	(1.00×)	9.82	(1.00×)	0.73	(1.20×)	1.27	(1.10×)	1.55	(1.14×)	0.99	(1.19×)	2.64	(0.63×)	9.80	(0.94×)	0.05	(1.00×)
diffeq1	2,228.00	(2.51×)	1.00	34.00	(5.67×)	162.00	(1.00×)	131.00	(4.09×)	2.00	(0.40×)	0.00		74.00	(1.48×)	17,639.00	(1.60×)	25.39	(1.46×)	0.13	(1.44×)	1.27	(10.58×)	1.83	(7.32×)	1.49	(3.10×)	7.59	(1.71×)	15.25	(2.16×)	0.06	(1.36×)
spree	1,377.00	(1.23×)	1.00	17.00	(1.00×)	45.00	(1.00×)	78.00	(1.32×)	0.00	(0.00×)	3.00	(1.00×)	64.00	(1.00×)	14,060.00	(1.11×)	12.66	(1.17×)	1.24	(1.19×)	0.71	(1.69×)	1.76	(1.20×)	0.66	(1.29×)	2.57	(0.44×)	9.18	(0.82×)	0.05	(1.03×)
boundtop	1,227.00	(1.00×)	1.00	3.00	(1.00×)	196.00	(1.00×)	38.00	(1.00×)	0.00		1.00	(1.00×)	42.00	(1.00×)	5,414.00	(1.00×)	2.33	(1.00×)	8.95	(1.35×)	0.48	(1.02×)	0.54	(1.13×)	0.53	(1.15×)	2.27	(1.14×)	14.45	(1.25×)	0.04	(1.02×)
mkPktMerge	1,158.00	(1.00×)	1.00	2.00	(1.00×)	311.00	(1.00×)	30.00	(1.00×)	0.00		15.00	(1.00×)	32.00	(1.00×)	15,343.00	(1.00×)	4.39	(1.00×)	0.50	(1.16×)	0.07	(1.00×)	0.51	(1.09×)	0.88	(1.13×)	3.48	(0.36×)	10.37	(0.67×)	0.07	(0.97×)
ch_intrinsics	462.00	(1.00×)	1.00	3.00	(1.00×)	99.00	(1.00×)	70.00	(1.00×)	0.00		1.00	(1.00×)	44.00	(1.00×)	1,356.00	(1.00×)	1.90	(1.00×)	0.14	(1.17×)	0.06	(1.20×)	0.09	(1.13×)	0.23	(1.15×)	0.44	(0.59×)	1.80	(0.91×)	0.03	(1.00×)
stereovision3	334.00	(1.00×)	2.00	5.00	(1.00×)	11.00	(1.00×)	16.00	(1.00×)	0.00		0.00		30.00	(1.00×)	1,022.00	(1.00×)	2.62	(1.00×)	0.24	(1.00×)	0.10	(0.91×)	0.20	(1.05×)	0.07	(1.00×)	0.16	(0.44×)	1.38	(0.89×)	0.03	(0.99×)
GEOMEAN	6,927.21	(1.28×)		13.27	(1.73×)	114.75	(1.00×)	390.80	(1.41×)	7.42	(0.43×)	9.56	(1.00×)	65.65	(1.01×)	47,262.78	(1.18×)	12.81	(1.26×)	2.70	(1.24×)	6.09	(2.21×)	5.99	(1.56×)	5.34	(1.55×)	16.65	(0.75×)	57.97	(1.14×)	0.17	(1.10×)
GEOMEAN (>10K)	39,361.49	(1.17×)		22.27	(1.64×)	105.84	(1.00×)	2,047.60	(1.23×)	13.35	(0.43×)	74.04	(1.00×)	89.55	(1.00×)	291,225.67	(1.08×)	22.36	(1.09×)	13.35	(1.23×)	85.28	(2.19×)	45.08	(1.44×)	43.32	(1.53×)	146.55	(0.84×)	453.21	(1.17×)	0.62	(1.09×)
% TOTAL (> 10K)		(1.00×)																		2.32%		44.98%		6.35%		9.59%		28.49%		100.00%			

Table 5.5: VTR Benchmarks using Yosys+Odin-II Front-end in the VTR Flow with Multplier Ratio of 50% (Relative to Yosys+Odin-II QoR Experiment).

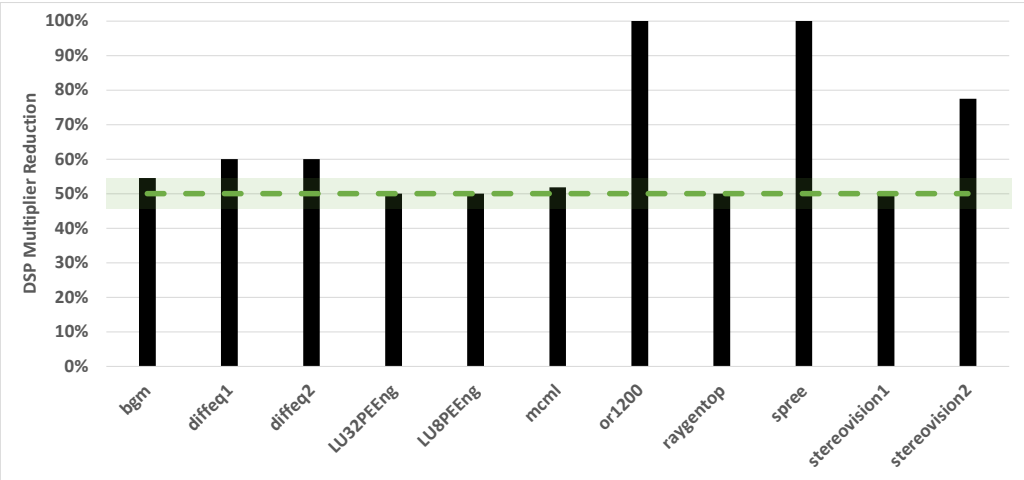


Figure 5.6: DSP Blocks Trade-off Percentage by Yosys+Odin-II Partial Mapper on VTR Benchmarks. Only Benchmarks With DSP Blocks are Included.

highest DSP reduction by 77.48%. The substantial difference between the achieved reduction and the configured ratio is mainly due to the VPR performance. While Yosys+Odin-II generates exactly half the multiplier hard blocks for the `stereovison2` benchmark, the ultimate DSP usage is dependent on how VPR performs packing of small DSP slices to achieve desirable block utilization. As expected, the entire VTR flow runtime increases due to the prolonged synthesis process and logic gate mapping time. Overall, the VTR flow runtime extends by 1.17×, while the Yosys+Odin-II synthesis time and ABC optimization time increases by 1.23× and 2.19×, respectively. VPR takes a longer time to perform packing and placing by 1.44× and 1.53×. Despite the inverse correlation between the routing efficiency and depth metrics [61], the relaxed channel width route time is decreased by 16%. This is a proof of VTR benchmarks RISC-style, showing that they lean on sequential logic that results in longer routing paths even though complex block utilization is fully obtained. In other words, due to the data feedback and overall clock rate bottleneck, some VTR benchmarks show a lower frequency using a pure complex logic implementation. At the same time, they need more time to be fully routed in the FPGA layout.

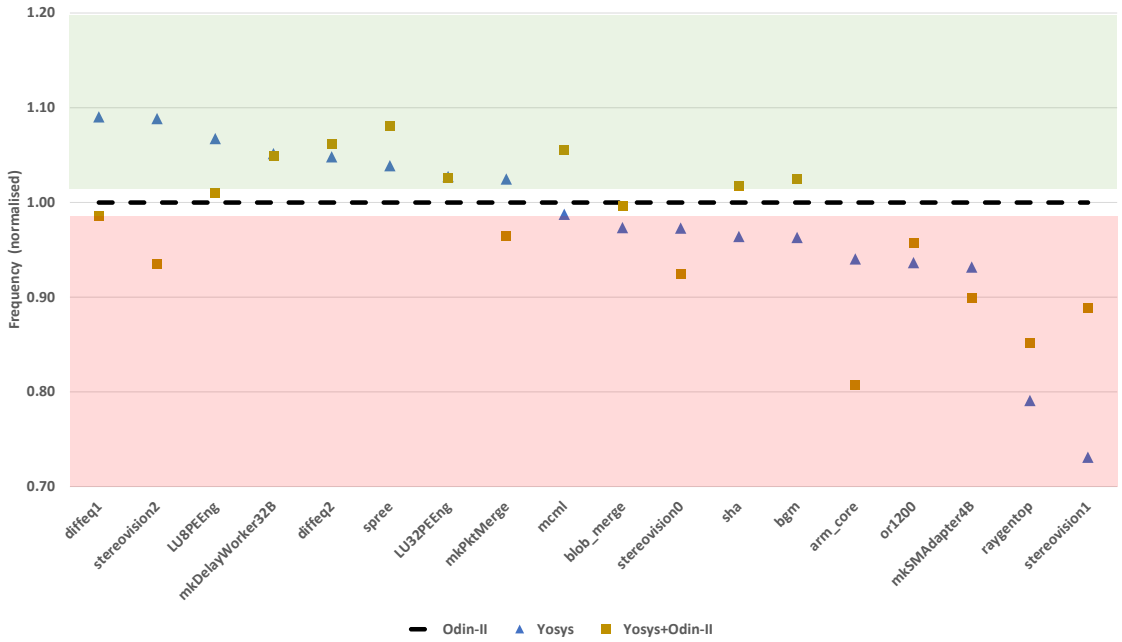


Figure 5.7: The VTR Benchmarks Frequency Comparison. (Normalised to the VTR Baseline)

5.4 Frequency

Figure 5.7 depicts the VTR benchmarks' frequency comparison synthesized by Odin-II, Yosys, and Yosys+Odin-II. The Yosys+Odin-II synthesizer achieves the maximum of 8% frequency improvement for `spree`, a soft microprocessor designed for a rapid exploration environment [62]. However, it is less performant for the `arm_core` soft processor, achieving a 19% weakened frequency. More than 55% of VTR benchmarks achieve a higher frequency or equal to the one obtained by the baseline, while the rest have a decreased frequency. Overall, the frequency of VTR benchmarks with the number of netlist primitives greater than 10K increases by 3.1%. According to the nature of the benchmarks, we can conclude that the Yosys+Odin-II performs better for large benchmarks with more fixed or floating-point multiplication operations, such as `mcml`, `LU32PEEng` and `bgm`.

The `stereovision2` benchmark achieves the highest frequency improvement, a 9% increase, with the Yosys synthesizer compared to the VTR baseline. On the other

hand, `arm_core` obtains the highest frequency degradation among the heavy netlist primitive VTR benchmarks, which is 6% less than the VTR baseline. Figure 5.7 shows the VTR flow with the Yosys front-end is frequency performant for mathematical benchmarks, such as `LU8PEEng` and `diffeq1`. Overall, a 1% critical path delay improvement is achieved by Yosys compared to the baseline.

Chapter 6

Conclusions and Future Directions

We have developed a combination of two leading academic and industrial synthesizers. The Yosys+Odin-II supports the extensive Verilog 2005 standard provided by Yosys as the front-end elaborator tool. Yosys is mainly responsible for elaborating the HDL digital circuits and generating a coarse-grained netlist. As a constrained technology packing estimator, the Odin-II partial mapper performs technology mapping of the coarse-grained netlist to the target device. By adding Yosys+Odin-II as the front-end of the Verilog-to-Routing project, the VTR flow can now synthesize more complex and commercial designs by having better language support alongside the highly configurable partial mapper.

The Yosys+Odin-II implementation utilizes highly tuned CAD algorithms and optimizations. While some unsupported operations have become available in the Odin-II technology mapping flow, Yosys+Odin-II benefits from constant folding and practical logic reduction optimizations. The Odin-II technology mapping flow is equipped with the instantiation of division and modulo operations, which were unsupported previously. The partial support for exponentiation operation is also improved in the Odin-II technology mapping flow. Detecting computationally-intensive operations, such as division and multiplication, with at least one constant input and reducing

them to basic arithmetic operations such as addition or shift is also added. The signal pruning process for logic cells and memory blocks with useless input bits is another feature of the Yosys+Odin-II technology mapper that can avoid the inference of large unused memory blocks.

The control over the process to the user through runtime parameters to the intelligent technology mapping is provided by the Yosys+Odin-II partial mapper. The Odin-II partial mapper provides the optimization for hard/soft logic for a ratio of size. As a result, the circuits synthesized by Yosys+Odin-II can utilize automated hard logic decisions with architecture awareness. The optimizations provided by Yosys+Odin-II resulted in a slight reduction, by 3%, in the device grid tiles, while it showed no tangible impact on the logic depth of digital circuits. Overall, the Yosys+Odin-II synthesizer generates more compact and smaller circuits than Odin-II, whereas the VTR flow with the Yosys+Odin-II front-end outperforms in several aspects, such as final CLB usage and overall routed wire lengths, especially in pure soft logic, such as the FIR benchmarks presented in Appendix B.

The overall VTR flow runtime is decreased by 16%. Although the synthesis time is increased substantially compared to the baseline, it still comprises a negligible portion (2.02%) of the overall VTR flow runtime. The Yosys+Odin-II optimizations reduced the ABC runtime by 14%, mainly accounting for the entire VTR flow runtime reduction. The VTR flow demonstrates more efficient results using Yosys+Odin-II as its front-end regarding placement and routing times. By having more compact generated circuits, the VTR place and routing times are reduced by 12%, and 22%, respectively. However, the VPR pack time increases by 7%.

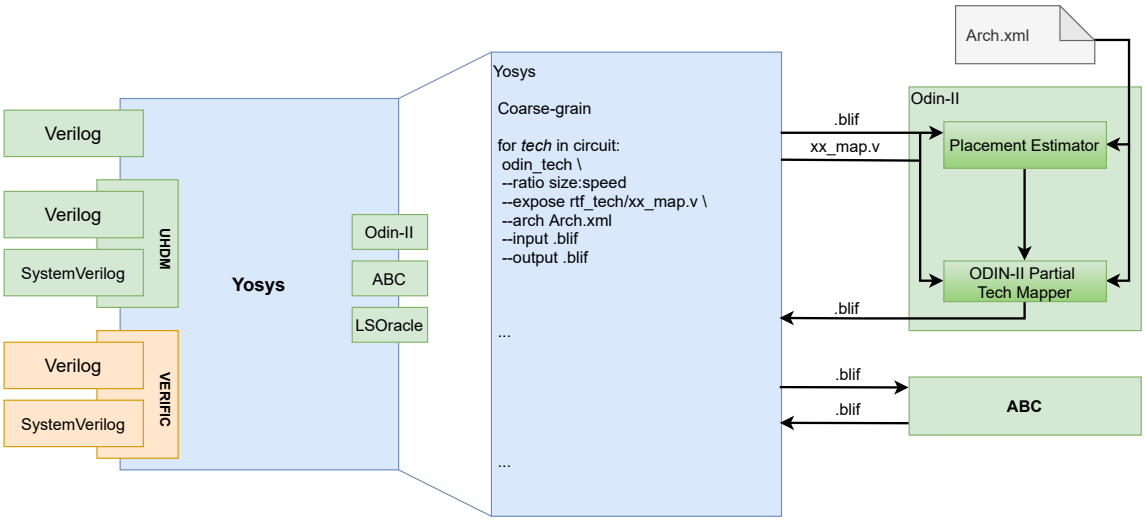


Figure 6.1: Odin-II Plugin Integration for Yosys.

6.1 Future Work

In terms of future work, experiments with more industrial digital circuits such as the Titan benchmarks [63] may yield interesting results. Potential development in the proof of concept integration is to perform experiments with multiple device fabrics. A different layout will affect the number of available hard blocks that could introduce pure hard/soft logic evaluation of the proposed synthesizer. Additionally, different device fabrics would alter the placeability of digital circuits, resulting in distinct Yosys+Odin-II reactions.

The taken approach is the initial implementation of the proposed idea, operating on available discrete components and augmenting the flow by deciding whether or not to explode particular discrete logic into gates. We plan to extract the Odin-II partial mapper and adjoin it as a plugin to the Yosys codebase, as shown in Figure 6.1. The Odin-II plugin will be similar to that of ABC, having Odin-II consume an interchange format, a BLIF in this case. The approach will be to add support for user-created RTL technology mapping files to be used by Odin-II. These files outline different discrete logic constructions based on other RTL technology and/or gate-level logic. Using RTL technology mapping files will allow conforming to the current Yosys

methods and allow both tools to gain this extensibility. The plugin will only operate on a single discrete logic block at a time, iteratively mapping all of them. This, in turn, will allow discrete logic to be described from other discrete logic and propose new approaches to build the digital circuits.

Bibliography

[1] Clifford Wolf. Yosys Open Synthesis Suite, 2016. `http://clifford.at/yosys/`.

[2] Antonio Lloris Ruiz, Encarnación Castillo Morales, Luís Parrilla Roure, Antonio García Ríos, and María José Lloris Meseguer. *Arithmetic and Algebraic Circuits*, volume 201 of *Intelligent Systems Reference Library*. Springer, 2021. `https://doi.org/10.1007/978-3-030-67266-9`.

[3] Justin Richardson, Steven Fingulin, Diwakar Raghunathan, Chris Massie, and Alan George Herman Lam. Comparative analysis of HPC and accelerator devices: Computation, memory, I/O, and power. In *Proceeding of the International Workshop on High-performance Reconfigurable Computer Technology and Applications (HPRCTA'10)*, pages 1–10, 2010.

[4] Steve Kilts. *Advanced FPGA Design: Architecture, Implementation, and Optimization*. John Wiley & Sons, 2007.

[5] Wayne Wolf. *FPGA-based system design*. Pearson Education India, 2004.

[6] Ron Bekkerman, Mikhail Bilenko, and John Langford. *Scaling up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, 2011.

[7] Sunila Gollapudi. *Practical Machine Learning*. Packt Publishing Ltd, 2016.

[8] Eric S. Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian M. Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Maleen Abeydeera, Logan Adams, Hari Angepat, Christian Boehn, Derek Chiou, Oren Firestein, Alessandro Forin, Kang Su Gatlin, Mahdi Ghandi, Stephen Heil, Kyle Holohan, Ahmad El Husseini, Tamás Juhász, Kara Kagi, Ratna Kovvuri, Sitaram Lanka, Friedel van Megen, Dima Mukhortov, Prerak Patel, Brandon Perez, Amanda Rapsang, Steven K. Reinhardt, Bita Rouhani, Adam Sapek, Raja Seera, Sangeetha Shekar, Balaji Sridharan, Gabriel Weisz, Lisa Woods, Phillip Yi Xiao, Dan Zhang, Ritchie Zhao, and Doug Burger. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. *IEEE Micro*, 38(2):8–20, 2018. https://doi.org/10.1109/MM.2018.022071131.

[9] Jason Luu, Jeffrey Goeders, Michael Wainberg, Andrew Somerville, Thien Yu, Konstantin Nasartschuk, Miad Nasr, Sen Wang, Tim Liu, Nooruddin Ahmed, Kenneth B. Kent, Jason Helge Anderson, Jonathan Rose, and Vaughn Betz. VTR 7.0: Next Generation Architecture and CAD System for FPGAs. *ACM Trans. Reconfigurable Technol. Syst.*, 7(2):6:1–6:30, 2014. https://doi.org/10.1145/2617593.

[10] Kevin E. Murray, Oleg Petelin, Sheng Zhong, Jia Min Wang, Mohamed Eldafrawy, Jean-Philippe Legault, Eugene Sha, Aaron Graham, Jean Wu, Matthew J. P. Walker, Hanqing Zeng, Panagiotis Patros, Jason Luu, Kenneth B. Kent, and Vaughn Betz. VTR 8: High-performance CAD and Customizable FPGA Architecture Modelling. *ACM Trans. Reconfigurable Technol. Syst.*, 13(2):9:1–9:55, 2020. https://doi.org/10.1145/3388617.

[11] Peter Jamieson, Kenneth B. Kent, Farnaz Gharibian, and Lesley Shannon. Odin II - An Open-source Verilog HDL Synthesis Tool for CAD Research. In Ron Sass and Russell Tessier, editors, *18th IEEE Annual International Symposium*

on Field-programmable Custom Computing Machines, FCCM 2010, Charlotte, North Carolina, USA, 2-4 May 2010, pages 149–156. IEEE Computer Society, 2010. https://doi.org/10.1109/FCCM.2010.31.

[12] Seyed Alireza Damghani, Jean-Philippe Legault, and Kenneth B. Kent. Desired Footprint by Technology Mapping Modification using a Genetic Algorithm in Odin II. In *International Workshop on Rapid System Prototyping, RSP 2020, Hamburg, Germany, September 24-25, 2020*, pages 1–7. IEEE, 2020. https://doi.org/10.1109/RSP51120.2020.9244862.

[13] Georgiy Krylov, Jean-Philippe Legault, and Kenneth B. Kent. Hard and Soft Logic Trade-offs for Multipliers in VTR. In *23rd Euromicro Conference on Digital System Design, DSD 2020, Kranj, Slovenia, August 26-28, 2020*, pages 40–43. IEEE, 2020. https://doi.org/10.1109/DSD51259.2020.00018.

[14] Harpreet Kaur, Georgiy Krylov, Seyed Alireza Damghani, and Kenneth B. Kent. Heterogeneous Logic Implementation for Adders in VTR. In *International Workshop on Rapid System Prototyping, RSP 2021, Virtual Event, October 10-15, 2021*. IEEE, 2021.

[15] Michael Flawn, Kenneth B. Kent, and Paul Cook. Honours Thesis: Collecting and Analyzing Netlist Statistics in Odin-II. University of New Brunswick, 2019.

[16] Murat Tuna and Can Bülent Fidan. Electronic Circuit Design, Implementation and FPGA-based Realization of a New 3D Chaotic System with Single Equilibrium Point. *Optik*, 127(24):11786–11799, 2016.

[17] Michael Wirthlin. High-reliability FPGA-based Systems: Space, High-energy Physics, and Beyond. *Proceedings of the IEEE*, 103(3):379–389, 2015.

[18] Yaoduo Xu and Xinhua Guo. Application of FPGA and Complex Embedded System in Sports Health Data Monitoring System. *Microprocessors and Microsystems*, page 103445, 2020.

[19] IEEE Standard for Verilog Hardware Description Language. *IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001)*, pages 1–590, 2006.

[20] Gary D. Hachtel and Fabio Somenzi. *Logic Synthesis and Verification Algorithms*. Springer, 2006. https://doi.org/10.1007/0-387-31005-3.

[21] Bo Yan and Kenneth B. Kent. Hard block reduction and synthesis improvements in Odin II. In *2015 International Symposium on Rapid System Prototyping, RSP 2015, Amsterdam, The Netherlands, October 8-9, 2015*, pages 126–132. IEEE, 2015. https://doi.org/10.1109/RSP.2015.7416557.

[22] Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification. 2021. http://www.eecs.berkeley.edu/~alanmi/abc.

[23] Robert K. Brayton and Alan Mishchenko. ABC: an academic industrial-strength verification tool. In Tayssir Touili, Byron Cook, and Paul B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 24–40. Springer, 2010. https://doi.org/10.1007/978-3-642-14295-6_5.

[24] MVSIS Group at Berkeley Studies Logic Synthesis and Verification for VLSI design. MVSIS: Logic Synthesis and Verification. 2021. https://embedded.eecs.berkeley.edu/mvsis/.

[25] Donald Chai, Jie-Hong Jiang, Yunjian Jiang, Yinghua Li, Alan Mishchenko, and Robert Brayton. MVSIS 2.0 User's Manual. *Department of Electrical*

Engineering and Computer Sciences, 2003. https://ptolemy.berkeley.edu/projects/embedded/mvsis/doc/mvsis_20_manual.pdf.

[26] Volnei A Pedroni. *Circuit Design with VHDL*. MIT press, 2020.

[27] Andreas Gerstlauer and Daniel D Gajski. System-level Abstraction Semantics. In *Proceedings of the 15th international symposium on System Synthesis*, pages 231–236, 2002.

[28] IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows. *IEEE Std 1685-2009*, pages 1–374, 2010.

[29] Jiang Long and R. Brayton. A Simple C to Verilog Compilation Procedure for Hardware / Software Verification. 2016.

[30] Intel corporation, 2021. https://www.intel.com/.

[31] Xilinx incorporation, 2021. https://www.xilinx.com.

[32] Synopsys, 2021. https://www.synopsys.com.

[33] Mentor graphics corporation, 2021. https://eda.sw.siemens.com/en-US/.

[34] Andrew Canis, Jongsok Choi, Blair Fort, Bain Syrowik, Ruolong Lian, Yu Ting Chen, Hsuan Hsiao, Jeffrey B. Goeders, Stephen Dean Brown, and Jason Helge Anderson. LegUp High-level Synthesis. In Dirk Koch, Frank Hannig, and Daniel Ziener, editors, *FPGAs for Software Programmers*, pages 175–190. Springer, 2016. https://doi.org/10.1007/978-3-319-26408-0_10.

[35] Accelerated HW Synthesis (XLS). https://github.com/google/xls.

[36] Ieee standard vhdl language reference manual. *IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002)*, pages 1–640, 2009.

[37] Vivek Sagdeo. Abstraction Levels in Verilog: Behavioral, RTL, and Structural. *The Complete Verilog Book*, pages 37–84, 1998.

[38] Eva M Ortigosa, Antonio Cañas, Eduardo Ros, Pilar Martínez Ortigosa, Sonia Mota, and Javier Díaz. Hardware Description of Multi-layer Perceptrons with Different Abstraction Levels. *Microprocessors and Microsystems*, 30(7):435–444, 2006.

[39] Clifford Wolf. Yosys Manual, 2021. `http://www.clifford.at/yosys/files/yosys_manual.pdf`.

[40] R.K. Brayton, G.D. Hachtel, and A.L. Sangiovanni-Vincentelli. Multilevel Logic Synthesis. *Proceedings of the IEEE*, 78(2):264–300, 1990.

[41] Yusuke Matsunaga and Masahiro Fujita. Multi-level Logic Optimization using Binary Decision Diagrams. In *ICCAD*, volume 89, pages 556–559, 1989.

[42] Jamie Bernard. Physical Level Design using Synopsys®. *Scholarly paper, George Mason University*, 2011.

[43] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Tim Cheng. *Electronic Design Automation: Synthesis, Verification, and Test*. Morgan Kaufmann, 2009.

[44] Neil Steiner, Aaron Wood, Hamid Shojaei, Jacob Couch, Peter Athanas, and Matthew French. Torc: Towards an Open-source Tool Flow. In John Wawrzynek and Katherine Compton, editors, *Proceedings of the ACM/SIGDA 19th International Symposium on Field Programmable Gate Arrays, FPGA 2011, Monterey, California, USA, February 27, March 1, 2011*, pages 41–44. ACM, 2011. `https://doi.org/10.1145/1950413.1950425`.

[45] Christopher Lavin, Marc Padilla, Jaren Lamprecht, Philip Lundrigan, Brent E. Nelson, and Brad L. Hutchings. RapidSmith: Do-it-yourself CAD Tools for

Xilinx FPGAs. In *International Conference on Field Programmable Logic and Applications, FPL 2011, September 5-7, Chania, Crete, Greece*, pages 349–355. IEEE Computer Society, 2011. https://doi.org/10.1109/FPL.2011.69.

[46] Travis Haroldsen, Brent E. Nelson, and Brad L. Hutchings. RapidSmith 2: A Framework for BEL-level CAD Exploration on Xilinx FPGAs. In George A. Constantinides and Deming Chen, editors, *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, February 22-24, 2015*, pages 66–69. ACM, 2015. https://doi.org/10.1145/2684746.2689085.

[47] David Shah, Eddie Hung, Clifford Wolf, Serge Bazanski, Dan Gisselquist, and Miodrag Milanovic. Yosys+nextpnr: An Open Source Framework from Verilog to Bitstream for Commercial FPGAs. In *27th IEEE Annual International Symposium on Field-programmable Custom Computing Machines, FCCM 2019, San Diego, CA, USA, April 28 - May 1, 2019*, pages 1–4. IEEE, 2019. https://doi.org/10.1109/FCCM.2019.00010.

[48] Kevin E. Murray, Mohamed A. Elgammal, Vaughn Betz, Tim Ansell, Keith Rothman, and Alessandro Comodi. SymbiFlow and VPR: An Open-source Design Flow for Commercial and Novel FPGAs. *IEEE Micro*, 40(4):49–57, 2020. https://doi.org/10.1109/MM.2020.2998435.

[49] Vaughn Betz and Jonathan Rose. Automatic generation of FPGA routing architectures from high-level descriptions. In Steve Trimberger and Scott Hauck, editors, *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA 2000, Monterey, CA, USA, February 10-11, 2000*, pages 175–184. ACM, 2000. https://doi.org/10.1145/329166.329203.

[50] Andrew Somerville and Kenneth B. Kent. Improving Memory Support in the VTR Flow. In Dirk Koch, Satnam Singh, and Jim Tørresen, editors, *22nd*

International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, August 29-31, 2012, pages 197–202. IEEE, 2012. https://doi.org/10.1109/FPL.2012.6339260.

[51] Victor N. Kravets and Prabhakar Kudva. Understanding Metrics in Logic Synthesis for Routability Enhancement. In Dennis Sylvester, Dirk Stroobandt, Louis Scheffer, and Payman Zarkesh-Ha, editors, *The 5th International Workshop on System-level Interconnect Prediction (SLIP 2003), Monterey, CA, USA, April 5-6, 2003, Proceedings*, pages 3–5. ACM, 2003. https://doi.org/10.1145/639929.639931.

[52] Bruce W Mcgaughy, Peter Frey, and Boris Krichevskiy. System and method for converting a flat netlist into a hierarchical netlist, 2007. US Patent 7,272,805.

[53] Nikos Drakos. Computer Based Learning Unit. *University of Leeds."Network Intrusion Detection: Evasion, Traffic normalization and End-to-end protocol semantics*, 1997.

[54] Maria Patrou, Jean-Philippe Legault, Aaron Graham, and Kenneth B. Kent. Improving Digital Circuit Simulation with Batch-parallel Logic Evaluation. In *22nd Euromicro Conference on Digital System Design, DSD 2019, Kallithea, Greece, August 28-30, 2019*, pages 144–151. IEEE, 2019. https://doi.org/10.1109/DSD.2019.00031.

[55] Uming Ko and Poras T Balsara. High-performance Energy-efficient D-flip-flop Circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(1):94–98, 2000.

[56] Quartus II. Handbook Volume 1: Design and Synthesis. *Altera Corporation*, 2014. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/qts/qts_qii51006.pdf.

[57] Jason Luu, Conor McCullough, Sen Wang, Safeen Huda, Bo Yan, Charles Chiasson, Kenneth B. Kent, Jason Helge Anderson, Jonathan Rose, and Vaughn Betz. On Hard Adders and Carry Chains in FPGAs. In *22nd IEEE Annual International Symposium on Field-programmable Custom Computing Machines, FCCM 2014, Boston, MA, USA, May 11-13, 2014*, pages 52–59. IEEE Computer Society, 2014. https://doi.org/10.1109/FCCM.2014.25.

[58] Kevin E. Murray, Jason Luu, Matthew J. P. Walker, Conor McCullough, Sen Wang, Safeen Huda, Bo Yan, Charles Chiasson, Kenneth B. Kent, Jason Anderson, Jonathan Rose, and Vaughn Betz. Optimizing FPGA Logic Block Architectures for Arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(6):1378–1391, 2020. https://doi.org/10.1109/TVLSI.2020.2965772.

[59] Eddie Hung. Mind the (Synthesis) Gap: Examining Where Academic FPGA Tools Lag Behind Industry. In *25th International Conference on Field Programmable Logic and Applications, FPL 2015, London, United Kingdom, September 2-4, 2015*, pages 1–4. IEEE, 2015. https://doi.org/10.1109/FPL.2015.7294007.

[60] Eddie Hung. VTR-to-Bitstream v2.0 (for VTR7). Access date: August 12th, 2021, http://eddiehung.github.io.

[61] Joey Y Lin, Ashok Jagannathan, and Jason Cong. Placement-driven technology mapping for lut-based fpgas. In *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*, pages 121–126, 2003.

[62] Peter Yiannacouras, Jonathan Rose, and J. Gregory Steffan. SPREE: Microarchitectural Exploration on FPGAs. https://www.eecg.utoronto.ca/~yiannac/docs/warfp05.pdf.

[63] Kevin E. Murray, Scott Whitty, Suya Liu, Jason Luu, and Vaughn Betz. Titan: Enabling large and complex benchmarks in academic CAD. In *23rd International Conference on Field programmable Logic and Applications, FPL 2013, Porto, Portugal, September 2-4, 2013*, pages 1–8. IEEE, 2013. https://doi.org/10.1109/FPL.2013.6645503.

Appendix A

Micro Benchmarks

Benchmark	Netlist Primitives		Clocks	Logic Depth		I/Os		CLBs		DSPs		RAMs	W_{min}	Routed Wirelength ($1.3\ W_{min}$)		Routed CPD ($1.3\ W_{min}$)		Synth Time (Yosys+Odin-II)		ABC Time		Pack Time		Place Time		Route Time (Find W_{min})		VTR Flow Elapsed Time		Peak Memory		
bm_expr_all_mod	1,721.00	(1.00×)	1.00	5.00	(1.00×)	162.00	(1.00×)	167.00	(0.99×)				52.00	(1.04×)	10,924.00	(1.01×)	4.49	(1.07×)	0.16	(8.00×)	0.11	(1.10×)	0.61	(1.00×)	1.33	(1.04×)	7.62	(1.04×)	13.59	(1.07×)	0.06	(1.00×)	
bm_arithmetic_unused_bits	1,175.00	(1.00×)	1.00	1.00	(1.00×)	609.00	(1.00×)	16.00	(1.00×)	4.00	(1.00×)		48.00	(1.00×)	6,938.00	(0.90×)	5.11	(0.95×)	0.08	(4.00×)	0.03	(1.00×)	0.16	(1.00×)	0.81	(1.05×)	6.30	(1.08×)	11.07	(1.07×)	0.06	(1.00×)	
bm_lpm_all	739.00	(1.00×)		5.00	(1.00×)	64.00	(1.00×)	24.00	(1.00×)				46.00	(1.00×)	6,777.00	(1.07×)	4.91	(1.09×)	0.06	(6.00×)	0.26	(1.08×)	0.56	(1.06×)	0.31	(1.11×)	1.68	(0.82×)	3.86	(0.95×)	0.04	(1.00×)	
bm_match4_str_arch	455.00	(0.83×)	1.00	1.00	(1.00×)	52.00	(1.00×)	8.00	(0.08×)	2.00	(1.00×)		58.00	(1.12×)	4,840.00	(1.63×)	7.09	(1.02×)	0.05	(5.00×)	0.01	(1.00×)	0.09	(1.00×)	0.19	(0.59×)	1.52	(0.71×)	2.68	(0.73×)	0.03	(0.92×)	
bm_match5_str_arch	321.00	(0.94×)	1.00	1.00	(1.00×)	91.00	(1.00×)	15.00	(1.00×)	4.00	(1.00×)		44.00	(0.96×)	3,875.00	(0.96×)	10.15	(0.94×)	0.05	(5.00×)	0.01	(0.50×)	0.07	(1.40×)	0.14	(0.93×)	2.61	(0.73×)	4.18	(0.81×)	0.04	(0.99×)	
bm_lpm_concat	312.00	(1.00×)		1.00	(1.00×)	48.00	(1.00×)	36.00	(1.00×)				22.00	(1.00×)	1,365.00	(1.00×)	1.16	(1.00×)	0.03	(3.00×)	0.01	(1.00×)	0.02	(1.00×)	0.16	(1.00×)	1.98	(1.06×)	2.84	(1.06×)	0.03	(1.00×)	
bm_match1_str_arch	308.00	(0.99×)	1.00	1.00	(1.00×)	89.00	(1.00×)	4.00	(0.50×)	3.00	(1.00×)		36.00	(1.00×)	4,108.00	(1.02×)	5.25	(0.97×)	0.04	(4.00×)	0.01		0.03	(1.00×)	0.14	(1.00×)	1.51	(0.98×)	2.73	(1.00×)	0.03	(0.99×)	
bm_match2_str_arch	300.00	(0.76×)	1.00	1.00	(1.00×)	55.00	(1.00×)	15.00	(0.83×)	2.00	(1.00×)		46.00	(0.82×)	3,325.00	(0.70×)	8.23	(0.93×)	0.05	(5.00×)	0.01	(1.00×)	0.09	(1.29×)	0.13	(0.81×)	1.37	(0.67×)	2.34	(0.76×)	0.03	(0.96×)	
bm_functional_test	273.00	(0.99×)	1.00	4.00	(1.00×)	34.00	(1.00×)	26.00	(0.96×)	2.00	(1.00×)		36.00	(1.00×)	2,238.00	(0.96×)	5.93	(0.99×)	0.09	(9.00×)	0.08	(1.14×)	0.19	(0.70×)	0.11	(1.10×)	0.75	(0.93×)	1.86	(0.98×)	0.03	(1.00×)	
bm_match3_str_arch	224.00	(1.00×)	1.00	1.00	(1.00×)	55.00	(1.00×)	13.00	(1.00×)	1.00	(1.00×)		36.00	(1.00×)	1,179.00	(0.88×)	4.84	(1.01×)	0.04	(4.00×)	0.01	(1.00×)	0.03	(1.00×)	0.07	(1.00×)	0.39	(0.91×)	0.98	(1.02×)	0.03	(0.99×)	
bm_base_multiply	217.00	(1.00×)	1.00	1.00	(1.00×)	48.00	(1.00×)	12.00	(1.00×)	3.00	(1.00×)		42.00	(1.17×)	2,145.00	(1.06×)	4.40	(1.00×)	0.06	(6.00×)	0.01	(0.50×)	0.03	(1.50×)	0.08	(1.00×)	1.56	(1.32×)	2.73	(1.22×)	0.03	(1.00×)	
bm_match6_str_arch	206.00	(1.00×)	1.00	1.00	(1.00×)	55.00	(1.00×)	11.00	(1.00×)	2.00	(1.00×)		36.00	(0.86×)	1,544.00	(1.05×)	5.57	(1.00×)	0.04	(4.00×)	0.01	(1.00×)	0.03	(1.00×)	0.08	(1.00×)	0.73	(1.11×)	1.46	(1.07×)	0.03	(1.00×)	
bm_mod	166.00	(1.00×)	1.00	1.00	(1.00×)	67.00	(1.00×)	33.00	(1.00×)				36.00	(0.95×)	692.00	(1.04×)	0.89	(0.96×)	0.03	(3.00×)	0.01		0.02	(1.00×)	0.04	(1.00×)	0.32	(1.03×)	0.87	(1.07×)	0.03	(1.00×)	
bm_and_log	128.00	(1.00×)		1.00	(1.00×)	64.00	(1.00×)	32.00	(1.00×)				34.00	(1.00×)	655.00	(1.00×)	1.47	(1.00×)	0.03	(3.00×)	0.01	(1.00×)	0.02	(1.00×)	0.03	(1.00×)	0.30	(1.03×)	0.82	(1.04×)	0.03	(1.00×)	
generate	85.00	(1.21×)	1.00	1.00	(1.00×)	3.00	(1.00×)	2.00	(1.00×)				30.00	(0.94×)	197.00	(0.92×)	2.11	(0.98×)	0.03	(3.00×)	0.02	(2.00×)	0.02	(1.00×)	0.01	(1.00×)	0.08	(0.89×)	0.40	(1.11×)	0.03	(1.01×)	
bm_dag3_lpm_mod	82.00	(0.89×)	1.00	1.00	(1.00×)	7.00	(1.00×)	10.00	(3.33×)				20.00	(1.25×)	140.00	(1.47×)	3.21	(1.19×)	0.05	(5.00×)	0.00	(0.00×)	0.05	(1.25×)	0.01	(1.00×)	0.06	(2.00×)	0.45	(1.36×)	0.03	(1.02×)	
bm_stmt_all_mod	58.00	(0.98×)	1.00	1.00	(1.00×)	6.00	(1.00×)	14.00	(1.00×)				20.00	(0.67×)	99.00	(0.73×)	0.91	(1.01×)	0.05	(5.00×)	0.02	(1.00×)	0.01	(1.00×)	0.01	(1.00×)	0.10	(0.83×)	0.49	(1.07×)	0.03	(1.00×)	
bm_DL_74381_ALU	52.00	(1.06×)		2.00	(1.00×)	11.00	(1.00×)	2.00	(1.00×)				20.00	(1.00×)	61.00	(0.56×)	3.95	(1.53×)	0.04	(4.00×)	0.03	(1.00×)	0.02	(2.00×)	0.01	(1.00×)	0.02	(0.67×)	0.35	(1.21×)	0.03	(0.98×)	
param_override	47.00	(0.80×)		1.00	(1.00×)	4.00	(1.00×)	8.00	(4.00×)	1.00	(1.00×)		18.00	(0.82×)	192.00	(0.59×)	6.00	(0.97×)	0.05	(5.00×)	0.01	(1.00×)	0.01	(1.00×)	0.01	(1.00×)	0.25	(1.32×)	0.68	(1.17×)	0.03	(1.00×)	
bm_dag3_lpm_log_mod	47.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	2.00	(1.00×)				18.00	(1.00×)	38.00	(0.97×)	2.36	(0.98×)	0.05	(5.00×)			0.06	(6.00×)	0.01	(1.00×)	0.02	(1.00×)	0.36	(1.38×)	0.03	(1.00×)	
parameter	43.00	(0.74×)		1.00	(1.00×)	4.00	(1.00×)	9.00	(4.50×)				16.00	(0.53×)	164.00	(1.27×)	1.84	(1.04×)	0.04	(4.00×)	0.01		0.01	(1.00×)	0.01	(1.00×)	0.10	(2.00×)	0.41	(1.37×)	0.03	(1.01×)	
parameter_2	43.00	(0.74×)		1.00	(1.00×)	4.00	(1.00×)	9.00	(4.50×)				16.00	(0.53×)	164.00	(1.27×)	1.84	(1.04×)	0.04	(4.00×)	0.01	(1.00×)	0.01	(1.00×)	0.01	(1.00×)	0.10	(2.00×)	0.42	(1.40×)	0.03	(1.02×)	
bm_DL_BCD_adder	42.00	(0.95×)		1.00	(1.00×)	9.00	(1.00×)	3.00	(1.50×)				20.00	(0.77×)	142.00	(1.19×)	4.40	(1.12×)	0.03	(3.00×)	0.01	(0.50×)	0.01	(0.50×)	0.01	(1.00×)	0.04	(1.33×)	0.34	(1.17×)	0.03	(1.01×)	
bm_stmt_compare_padding	40.00	(0.98×)	1.00	1.00	(1.00×)	6.00	(1.00×)	9.00	(1.00×)				20.00	(1.00×)	86.00	(1.32×)	0.90	(1.00×)	0.04	(4.00×)	0.01	(1.00×)	0.01	(1.00×)	0.01	(1.00×)	0.05	(1.00×)	0.38	(1.15×)	0.03	(1.00×)	
bm_DL_4_16_encoder	37.00	(1.00×)		1.00	(1.00×)	5.00	(1.00×)	1.00	(1.00×)				56.00	(1.00×)	95.00	(1.00×)	1.02	(1.00×)	0.04	(4.00×)	0.02	(1.00×)	0.01	(1.00×)			0.05	(1.25×)	0.33	(1.18×)	0.03	(1.01×)	
bm_dag1_mod	36.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				20.00	(1.00×)	20.00	(0.69×)	0.68	(0.96×)	0.05	(5.00×)			0.01	(1.00×)			0.03	(1.00×)	0.32	(1.19×)	0.03	(0.99×)	
bm_dag4_mod	36.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				20.00	(1.00×)	20.00	(0.69×)	0.68	(0.96×)	0.05	(5.00×)			0.01	(1.00×)			0.03	(1.00×)	0.32	(1.19×)	0.03	(1.00×)	
bm_dag3_log_mod	36.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				10.00	(1.00×)	36.00	(1.00×)	1.14	(1.00×)	0.05	(5.00×)	0.01	(1.00×)	0.01	(1.00×)			0.03	(1.00×)	0.33	(1.22×)	0.03	(0.99×)	
bm_dag3_mod	35.00	(1.00×)	1.00	1.00	(1.00×)	5.00	(1.00×)	3.00	(1.00×)				20.00	(1.00×)	15.00	(1.00×)	0.76	(1.00×)	0.06	(6.00×)	0.01		0.01	(1.00×)			0.03	(1.00×)	0.34	(1.26×)	0.03	(1.00×)	
if_generate	32.00	(1.00×)						16.00	(1.00×)				2.00	(1.00×)			nan		0.04	(4.00×)	0.01	(1.00×)	0.01	(1.00×)			0.07	(1.00×)	0.39	(1.11×)	0.03	(0.99×)	
case_generate	32.00	(1.00×)						16.00	(1.00×)				2.00	(1.00×)			nan		0.04	(4.00×)			0.01	(1.00×)			0.07	(1.00×)	0.41	(1.17×)	0.03	(1.00×)	
bm_dag3_lpm	31.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	6.00	(3.00×)				20.00	(1.00×)	100.00	(3.57×)	4.96	(1.24×)	0.03	(3.00×)	0.01	(1.00×)	0.02	(2.00×)	0.01		0.03	(1.50×)	0.32	(1.28×)	0.03	(1.02×)	
bm_dag1_log_mod	29.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				10.00	(1.00×)	30.00	(0.83×)	1.11	(0.98×)	0.05	(5.00×)	0.01		0.01	(1.00×)			0.03	(1.00×)	0.32	(1.23×)	0.03	(1.00×)	
bm_dag1_lpm	27.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	3.00	(3.00×)				14.00	(0.54×)	71.00	(2.45×)	3.08	(0.87×)	0.03	(3.00×)			0.01	(1.00×)			0.03	(3.00×)	0.30	(1.36×)	0.03	(1.02×)	
bm_DL_16_1_mux	27.00	(0.96×)		3.00	(1.00×)	20.00	(1.00×)	1.00	(1.00×)				18.00	(0.90×)	29.00	(1.21×)	1.56	(0.79×)	0.04	(4.00×)	0.02	(1.00×)					0.01	(1.00×)	0.27	(1.17×)	0.03	(1.01×)	
bm_if_common	26.00	(0.87×)	1.00	1.00	(1.00×)	7.00	(1.00×)	1.00	(1.00×)				28.00	(1.00×)	62.00	(1.03×)		(1.00×)	0.04	(4.00×)	0.02	(1.00×)					0.01	(1.00×)	0.27	(1.17×)	0.03	(0.99×)	•∗
bm_if_collapse	26.00	(0.87×)	1.00	1.00	(1.00×)	7.00	(1.00×)	1.00	(1.00×)				28.00	(0.74×)	50.00	(1.47×)			0.04	(4.00×)	0.02	(1.00×)	0.00	(0.00×)			0.01	(1.00×)	0.28	(1.17×)	0.03	(1.00×)	∗
bm_dag2_mod	24.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				20.00	(1.00×)	31.00	(1.07×)	0.72	(1.02×)	0.04	(4.00×)	0.01						0.03	(1.00×)	0.30	(1.15×)	0.03	(1.00×)	
bm_dag2_log_mod	24.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				20.00	(1.00×)	31.00	(1.07×)	0.72	(1.02×)	0.04	(4.00×)							0.03	(1.00×)	0.31	(1.19×)	0.03	(0.99×)	
bm_DL_four_bit_adder_continuous_assign	20.00	(1.00×)		2.00	(1.00×)	9.00	(1.00×)	1.00	(1.00×)				32.00	(1.00×)	34.00	(1.00×)	1.82	(1.00×)	0.03	(3.00×)	0.00	(0.00×)					0.02	(2.00×)	0.26	(1.13×)	0.03	(0.99×)	
bm_dag2_lpm	19.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	2.00	(2.00×)				12.00	(0.46×)	42.00	(1.27×)	3.40	(0.90×)	0.03	(3.00×)	0.01		0.01	(1.00×)			0.02	(2.00×)	0.27	(1.29×)	0.03	(1.00×)	
bm_DL_4_bit_shift_register	19.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	1.00	(1.00×)				26.00	(1.00×)	56.00	(1.00×)	0.76	(1.00×)	0.03	(3.00×)	0.01	(0.50×)					0.01	(1.00×)	0.26	(1.13×)	0.03	(1.01×)	
adder_hard_block	18.00	(1.29×)		1.00	(1.00×)	5.00	(1.00×)	2.00	(2.00×)				12.00		50.00		2.76		0.03	(3.00×)	0.01	(1.00×)	0.01				0.02	(1.00×)	0.27	(1.80×)	0.03	(0.92×)	○
bm_DL_BCD_7_segment_without_x	18.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	1.00	(1.00×)				32.00	(1.00×)	76.00	(1.00×)	1.40	(1.00×)	0.04	(4.00×)	0.01	(0.50×)	0.01	(1.00×)			0.02	(1.00×)	0.29	(1.21×)	0.03	(1.01×)	
bm_tester	17.00	(1.00×)	1.00	1.00	(1.00×)	5.00	(1.00×)	4.00	(1.00×)				20.00	(1.00×)	34.00	(0.92×)	0.91	(1.07×)	0.04	(4.00×)	0.02	(2.00×)					0.03	(1.00×)	0.31	(1.15×)	0.03	(0.99×)	
bm_if_reset	17.00	(1.00×)	1.00	1.00	(1.00×)	8.00	(1.00×)	1.00	(1.00×)				28.00	(1.00×)	29.00	(1.00×)		(1.00×)	0.04	(4.00×)	0.02	(1.00×)					0.01	(1.00×)	0.27	(1.23×)	0.03	(1.01×)	•∗
bm_DL_4_bit_comparator	16.00	(1.00×)		2.00	(1.00×)	8.00	(1.00×)	1.00	(1.00×)				26.00	(1.00×)	43.00	(1.00×)	1.61	(1.00×)	0.03	(3.00×)	0.01	(1.00×)					0.01	(1.00×)	0.26	(1.13×)	0.03	(1.00×)	
bm_DL_behavioural_full_adder	13.00	(1.00×)		1.00	(1.00×)	3.00	(1.00×)	2.00	(2.00×)				12.00	(0.46×)	17.00	(0.81×)	2.06	(0.68×)	0.03	(3.00×)			0.01				0.01	(1.00×)	0.26	(1.24×)	0.03	(1.00×)	
bm_add_lpm	11.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	1.00	(1.00×)				18.00	(0.69×)	28.00	(0.97×)	1.58	(0.67×)	0.02	(2.00×)							0.01	(1.00×)	0.23	(1.10×)	0.03	(1.01×)	
bm_DL_2_4_encoder	11.00	(1.00×)		1.00	(1.00×)	3.00	(1.00×)	1.00	(1.00×)				28.00	(1.00×)	32.00	(1.00×)		(1.00×)	0.03	(3.00×)	0.02	(2.00×)					0.01	(1.00×)	0.26	(1.13×)	0.03	(1.01×)	•∗
bm_DL_logic_w_Dff	10.00	(1.00×)	1.00	1.00	(1.00×)	4.00	(1.00×)	1.00	(1.00×)				20.00	(1.00×)	8.00	(1.00×)	1.76	(1.00×)	0.03	(3.00×)	0.01	(1.00×)					0.01	(1.00×)	0.24	(1.14×)	0.03	(1.00×)	
bm_DL_logic_w_Dff2	10.00	(1.00×)	1.00	1.00	(1.00×)	4.00	(1.00×)	1.00	(1.00×)				20.00	(1.00×)	8.00	(1.00×)	1.76	(1.00×)	0.03	(3.00×)	0.00	(0.00×)					0.01	(1.00×)	0.23	(1.10×)	0.03	(1.01×)	
bm_DL_structural_logic2	10.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	1.00	(1.00×)				28.00	(1.00×)	31.00	(1.00×)		(1.00×)	0.03	(3.00×)							0.01	(1.00×)	0.24	(1.14×)	0.03	(1.00×)	•∗
bm_dag1_log	8.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	2.00	(1.00×)				20.00	(1.00×)	25.00	(1.00×)	1.09	(1.00×)	0.03	(3.00×)	0.00	(0.00×)					0.02	(2.00×)	0.26	(1.13×)	0.03	(1.00×)	
bm_dag2_log	8.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	2.00	(1.00×)				20.00	(1.00×)	21.00	(1.00×)	1.06	(1.00×)	0.03	(3.00×)	0.01						0.01	(1.00×)	0.26	(1.18×)	0.03	(1.00×)	
bm_DL_4_1_mux	8.00	(1.00×)		1.00	(1.00×)	6.00	(1.00×)	1.00	(1.00×)				10.00	(1.00×)	14.00	(1.00×)	0.89	(1.00×)	0.02	(2.00×)	0.01	(0.50×)					0.01	(1.00×)	0.25	(1.14×)	0.03	(1.00×)	
bm_DL_Dff_w_synch_reset	6.00	(1.00×)	1.00	1.00	(1.00×)	3.00	(1.00×)	1.00	(1.00×)				26.00	(1.00×)	4.00	(1.00×)	1.68	(1.00×)	0.03	(3.00×)	0.01	(1.00×)					0.01	(1.00×)	0.24	(1.09×)	0.03	(1.00×)	
bm_DL_2_cascaded_flip_flops	6.00	(1.00×)	1.00	1.00	(1.00×)	2.00	(1.00×)	1.00	(1.00×)				26.00	(1.00×)	55.00	(1.00×)	0.55	(1.00×)	0.03	(3.00×)	0.01						0.01	(1.00×)	0.23	(1.10×)	0.03	(1.01×)	
bm_DL_2_1_mux	5.00	(1.00×)		1.00	(1.00×)	3.00	(1.00×)	1.00	(1.00×)				26.00	(1.00×)	6.00	(1.00×)	1.79	(1.00×)	0.02	(2.00×)	0.01	(1.00×)					0.01	(1.00×)	0.25	(1.14×)	0.03	(0.99×)	
bm_DL_structural_logic	5.00	(1.00×)		1.00	(1.00×)	3.00	(1.00×)	1.00	(1.00×)				26.00	(1.00×)	6.00	(1.00×)	1.79	(1.00×)	0.02	(2.00×)	0.00	(0.00×)					0.01	(1.00×)	0.23	(1.10×)	0.03	(0.99×)	
bm_dag3_log	4.00	(1.00×)						2.00	(1.00×)				2.00	(1.00×)			nan		0.03	(3.00×)	0.01						0.01	(1.00×)	0.25	(1.14×)	0.03	(1.00×)	
bm_my_D_latch1	4.00	(1.00×)		1.00	(1.00×)	2.00	(1.00×)	1.00	(1.00×)				20.00	(1.00×)	10.00	(1.00×)	2.17	(1.00×)	0.03	(3.00×)	0.00	(0.00×)					0.01	(1.00×)	0.23	(1.10×)	0.03	(0.99×)	
bm_dag3_lpm_log	4.00	(1.00×)		1.00	(1.00×)			2.00	(1.00×)				2.00	(1.00×)			nan		0.03	(3.00×)	0.00	(0.00×)					0.01	(1.00×)	0.25	(1.14×)	0.03	(0.99×)	
bm_DL_D_flipflop	4.00	(1.00×)	1.00	1.00	(1.00×)	2.00	(1.00×)	1.00	(1.00×)				20.00	(1.00×)	10.00	(1.00×)	0.68	(1.00×)	0.03	(3.00×)							0.01	(1.00×)	0.23	(1.15×)	0.03	(1.01×)	
bm_my_D_latch2	2.00	(1.00×)		1.00	(1.00×)	1.00	(1.00×)						14.00	(1.00×)	9.00	(1.00×)	0.21	(1.00×)	0.02	(2.00×)							0.01	(1.00×)	0.23	(1.15×)	0.03	(1.02×)	
GEOMEAN	33.97	(0.98×)		1.13	(1.00×)	9.01	(1.00×)	3.51	(1.11×)	2.17	(1.00×)		20.27	(0.92×)	93.93	(1.03×)	1.88	(0.99×)	0.04	(3.68×)	0.01	(0.95×)	0.02	(1.10×)	0.04	(0.98×)	0.05	(1.08×)	0.47	(1.14×)	0.03	(1.00×)	
GEOMEAN (>100)	346.90	(0.96×)		1.39	(1.00×)	74.03	(1.00×)	18.97	(0.78×)	2.36	(1.00×)		39.87	(0.99×)	2,618.24	(1.00×)	4.10	(0.99×)	0.05	(4.65×)	0.02	(0.91×)	0.07	(1.05×)	0.14	(0.96×)	1.31	(0.94×)	2.59	(0.98×)	0.03	(0.99×)	
% TOTAL (> 100)																			1.56%		1.12%		3.75%		6.96%		55.07%		100.00%			

Runtime in Seconds, Memory in GiB, WL in grid tiles, and CPD in ns.

○ VPR failed to route in the VTR baseline

• VPR failed to implement an unroutable circuit with a specific channel width factor in the VTR Baseline

∗ VPR failed to implement an unroutable circuit with a specific channel width factor in the VTR flow with Yosys+Odin-II

Table A.1: Micro Benchmarks using Yosys+Odin-II Front-end in the VTR Flow (Relative to the VTR Baseline)

Benchmark	Netlist Primitives		Clocks	Logic Depth		I/Os		CLBs		DSPs		RAMs	W_{min}		Routed Wirelength ($1.3\ W_{min}$)		Routed CPD ($1.3\ W_{min}$)		Synth Time (Yosys+Odin-II)		ABC Time		Pack Time		Place Time		Route Time (Find W_{min})		VTR Flow Elapsed Time		Peak Memory		
bm_expr_all_mod	1,671.00	(0.97×)	1.00	7.00	(1.40×)	162.00	(1.00×)	166.00	(0.98×)				48.00	(0.96×)	11,091.00	(1.03×)	4.21	(1.00×)	0.48	(24.00×)	0.08	(0.80×)	0.53	(0.87×)	1.29	(1.01×)	7.75	(1.06×)	13.70	(1.07×)	0.06	(1.00×)	
bm_match1_str_arch	1,191.00	(3.82×)	1.00	14.00	(14.00×)	89.00	(1.00×)	83.00	(10.38×)	2.00	(0.67×)		52.00	(1.44×)	9,722.00	(2.42×)	9.02	(1.67×)	0.39	(39.00×)	0.69		0.60	(20.00×)	0.56	(4.00×)	2.31	(1.50×)	6.12	(2.25×)	0.04	(1.25×)	
bm_arithmetic_unused_bits	1,171.00	(1.00×)	1.00	1.00	(1.00×)	609.00	(1.00×)	17.00	(1.06×)	4.00	(1.00×)		46.00	(0.96×)	7,337.00	(0.95×)	5.19	(0.97×)	0.16	(8.00×)	0.03	(1.00×)	0.16	(1.00×)	0.77	(1.00×)	6.22	(1.06×)	11.02	(1.07×)	0.06	(1.00×)	
bm_lpm_all	562.00	(0.76×)		4.00	(0.80×)	64.00	(1.00×)	18.00	(0.75×)				48.00	(1.04×)	3,947.00	(0.62×)	3.53	(0.78×)	0.22	(22.00×)	0.13	(0.54×)	0.50	(0.94×)	0.19	(0.68×)	1.42	(0.69×)	3.20	(0.79×)	0.03	(0.91×)	
bm_match4_str_arch	425.00	(0.77×)	1.00	1.00	(1.00×)	52.00	(1.00×)	117.00	(1.14×)	3.00	(1.50×)		40.00	(0.77×)	2,171.00	(0.73×)	7.13	(1.02×)	0.08	(8.00×)	0.02	(2.00×)	0.08	(0.89×)	0.26	(0.81×)	1.45	(0.67×)	3.02	(0.82×)	0.04	(1.02×)	
bm_lpm_concat	312.00	(1.00×)		1.00	(1.00×)	48.00	(1.00×)	36.00	(1.00×)				22.00	(1.00×)	1,365.00	(1.00×)	1.16	(1.00×)	0.03	(3.00×)	0.01	(1.00×)	0.02	(1.00×)	0.16	(1.00×)	1.94	(1.04×)	2.79	(1.04×)	0.03	(0.99×)	
bm_match5_str_arch	306.00	(0.90×)	1.00	1.00	(1.00×)	91.00	(1.00×)	15.00	(1.00×)	6.00	(1.50×)		40.00	(0.87×)	4,072.00	(1.00×)	11.00	(1.02×)	0.09	(9.00×)	0.02	(1.00×)	0.07	(1.40×)	0.21	(1.40×)	2.18	(0.61×)	3.87	(0.75×)	0.04	(1.02×)	
bm_functional_test	287.00	(1.04×)	1.00	4.00	(1.00×)	34.00	(1.00×)	27.00	(1.00×)	1.00	(0.50×)		42.00	(1.17×)	2,130.00	(0.91×)	8.61	(1.44×)	0.19	(19.00×)	0.07	(1.00×)	0.26	(0.96×)	0.12	(1.20×)	0.74	(0.91×)	1.94	(1.03×)	0.03	(0.99×)	
bm_base_multiply	269.00	(1.25×)	1.00	6.00	(6.00×)	32.00	(0.67×)	22.00	(1.83×)	2.00	(0.67×)		36.00	(1.00×)	1,880.00	(0.93×)	5.07	(1.15×)	0.13	(13.00×)	0.07	(3.50×)	0.07	(3.50×)	0.12	(1.50×)	0.69	(0.58×)	1.72	(0.77×)	0.03	(0.91×)	
bm_match2_str_arch	232.00	(0.59×)	1.00	1.00	(1.00×)	55.00	(1.00×)	13.00	(0.72×)	4.00	(2.00×)		46.00	(0.82×)	2,554.00	(0.54×)	8.54	(0.97×)	0.08	(8.00×)	0.02	(2.00×)	0.06	(0.86×)	0.15	(0.94×)	1.84	(0.90×)	3.43	(1.12×)	0.04	(1.15×)	
bm_mod	166.00	(1.00×)	1.00	1.00	(1.00×)	67.00	(1.00×)	33.00	(1.00×)				26.00	(0.68×)	766.00	(1.15×)	0.94	(1.00×)	0.04	(4.00×)	0.01		0.02	(1.00×)	0.04	(1.00×)	0.27	(0.87×)	0.82	(1.01×)	0.03	(0.99×)	
bm_match3_str_arch	157.00	(0.70×)	1.00	1.00	(1.00×)	37.00	(0.67×)	11.00	(0.85×)	1.00	(1.00×)		26.00	(0.72×)	1,004.00	(0.75×)	5.76	(1.20×)	0.07	(7.00×)	0.01	(1.00×)	0.02	(0.67×)	0.06	(0.86×)	0.50	(1.16×)	1.07	(1.11×)	0.03	(0.99×)	
bm_match6_str_arch	139.00	(0.67×)	1.00	1.00	(1.00×)	37.00	(0.67×)	10.00	(0.91×)	1.00	(0.50×)		40.00	(0.95×)	784.00	(0.53×)	5.62	(1.01×)	0.07	(7.00×)	0.01	(1.00×)	0.02	(0.67×)	0.06	(0.75×)	0.44	(0.67×)	1.01	(0.74×)	0.03	(0.96×)	
bm_and_log	128.00	(1.00×)		1.00	(1.00×)	64.00	(1.00×)	32.00	(1.00×)				34.00	(1.00×)	655.00	(1.00×)	1.47	(1.00×)	0.04	(4.00×)	0.01	(1.00×)	0.02	(1.00×)	0.03	(1.00×)	0.30	(1.03×)	0.84	(1.06×)	0.03	(1.00×)	
generate	70.00	(1.00×)	1.00	1.00	(1.00×)	3.00	(1.00×)	2.00	(1.00×)				32.00	(1.00×)	194.00	(0.90×)	2.19	(1.02×)	0.04	(4.00×)	0.02	(2.00×)	0.02	(1.00×)	0.01	(1.00×)	0.08	(0.89×)	0.41	(1.14×)	0.03	(1.01×)	
bm_stmt_all_mod	58.00	(0.98×)	1.00	1.00	(1.00×)	6.00	(1.00×)	14.00	(1.00×)				20.00	(0.67×)	106.00	(0.78×)	0.91	(1.01×)	0.08	(8.00×)	0.02	(1.00×)	0.01	(1.00×)	0.01	(1.00×)	0.10	(0.83×)	0.52	(1.13×)	0.03	(1.00×)	
param_override	45.00	(0.76×)		1.00	(1.00×)	4.00	(1.00×)	7.00	(3.50×)	1.00	(1.00×)		20.00	(0.91×)	247.00	(0.76×)	6.19	(1.00×)	0.07	(7.00×)	0.00	(0.00×)	0.01	(1.00×)	0.02	(2.00×)	0.17	(0.89×)	0.64	(1.10×)	0.03	(0.99×)	
parameter	42.00	(0.72×)		1.00	(1.00×)	4.00	(1.00×)	9.00	(4.50×)				20.00	(0.67×)	139.00	(1.08×)	1.58	(0.89×)	0.06	(6.00×)	0.01		0.01	(1.00×)	0.01	(1.00×)	0.06	(1.20×)	0.40	(1.33×)	0.03	(1.00×)	
parameter_2	42.00	(0.72×)		1.00	(1.00×)	4.00	(1.00×)	9.00	(4.50×)				20.00	(0.67×)	139.00	(1.08×)	1.58	(0.89×)	0.07	(7.00×)	0.00	(0.00×)	0.01	(1.00×)	0.01	(1.00×)	0.06	(1.20×)	0.39	(1.30×)	0.03	(1.01×)	
bm_DL_74381_ALU	41.00	(0.84×)		2.00	(1.00×)	11.00	(1.00×)	2.00	(1.00×)				18.00	(0.90×)	86.00	(0.80×)	2.86	(1.11×)	0.07	(7.00×)	0.02	(0.67×)	0.08	(8.00×)	0.01	(1.00×)	0.03	(1.00×)	0.43	(1.48×)	0.03	(0.99×)	
bm_stmt_compare_padding	40.00	(0.98×)	1.00	1.00	(1.00×)	6.00	(1.00×)	9.00	(1.00×)				20.00	(1.00×)	86.00	(1.32×)	0.90	(1.00×)	0.05	(5.00×)	0.01	(1.00×)	0.01	(1.00×)	0.01	(1.00×)	0.05	(1.00×)	0.39	(1.18×)	0.03	(1.01×)	
bm_DL_4_16_encoder	37.00	(1.00×)		1.00	(1.00×)	5.00	(1.00×)	1.00	(1.00×)				56.00	(1.00×)	95.00	(1.00×)	1.02	(1.00×)	0.04	(4.00×)	0.01	(0.50×)	0.01	(1.00×)			0.04	(1.00×)	0.35	(1.25×)	0.03	(1.01×)	
bm_dag1_mod	36.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				20.00	(1.00×)	28.00	(0.97×)	0.81	(1.14×)	0.07	(7.00×)	0.01		0.01	(1.00×)			0.03	(1.00×)	0.34	(1.26×)	0.03	(1.00×)	
bm_dag4_mod	36.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				20.00	(1.00×)	28.00	(0.97×)	0.81	(1.14×)	0.07	(7.00×)	0.01		0.01	(1.00×)			0.03	(1.00×)	0.35	(1.30×)	0.03	(1.00×)	
bm_dag3_log_mod	36.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				10.00	(1.00×)	19.00	(0.53×)	1.09	(0.96×)	0.08	(8.00×)	0.00	(0.00×)	0.01	(1.00×)			0.03	(1.00×)	0.35	(1.30×)	0.03	(0.99×)	
bm_dag3_mod	35.00	(1.00×)	1.00	1.00	(1.00×)	5.00	(1.00×)	3.00	(1.00×)				20.00	(1.00×)	15.00	(1.00×)	0.76	(1.00×)	0.08	(8.00×)	0.01		0.01	(1.00×)			0.03	(1.00×)	0.37	(1.37×)	0.03	(1.00×)	
bm_DL_BCD_adder	34.00	(0.77×)		1.00	(1.00×)	9.00	(1.00×)	1.00	(0.50×)				28.00	(1.08×)	36.00	(0.30×)	3.87	(0.98×)	0.05	(5.00×)	0.01	(0.50×)	0.01	(0.50×)	0.00	(0.00×)	0.02	(0.67×)	0.31	(1.07×)	0.03	(0.98×)	
bm_dag3_lpm_log_mod	33.00	(0.70×)	1.00	1.00	(1.00×)	5.00	(0.71×)	1.00	(0.50×)				18.00	(1.00×)	21.00	(0.54×)	1.17	(0.49×)	0.07	(7.00×)	0.01		0.01	(1.00×)	0.00	(0.00×)	0.01	(0.50×)	0.31	(1.19×)	0.03	(0.99×)	
if_generate	32.00	(1.00×)						16.00	(1.00×)				2.00	(1.00×)					0.07	(7.00×)	0.00	(0.00×)	0.01	(1.00×)			0.07	(1.00×)	0.43	(1.23×)	0.03	(0.99×)	
case_generate	32.00	(1.00×)						16.00	(1.00×)				2.00	(1.00×)					0.06	(6.00×)			0.01	(1.00×)			0.07	(1.00×)	0.43	(1.23×)	0.03	(1.01×)	
bm_dag1_log_mod	29.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				10.00	(1.00×)	19.00	(0.53×)	1.09	(0.96×)	0.06	(6.00×)			0.01	(1.00×)			0.03	(1.00×)	0.34	(1.31×)	0.03	(1.01×)	
bm_DL_16_1_mux	27.00	(0.96×)		3.00	(1.00×)	20.00	(1.00×)	1.00	(1.00×)				18.00	(0.90×)	29.00	(1.21×)	1.56	(0.79×)	0.05	(5.00×)	0.02	(1.00×)					0.01	(1.00×)	0.28	(1.22×)	0.03	(1.01×)	
bm_if_common	26.00	(0.87×)	1.00	1.00	(1.00×)	7.00	(1.00×)	1.00	(1.00×)				26.00	(0.93×)	39.00	(0.65×)	1.53		0.05	(5.00×)	0.02	(1.00×)					0.01	(1.00×)	0.29	(1.26×)	0.03	(1.00×)	•
bm_if_collapse	26.00	(0.87×)	1.00	1.00	(1.00×)	7.00	(1.00×)	1.00	(1.00×)				30.00	(0.79×)	29.00	(0.85×)	0.83	(0.84×)	0.05	(5.00×)	0.02	(1.00×)	0.00	(0.00×)			0.01	(1.00×)	0.30	(1.25×)	0.03	(1.01×)	
bm_dag2_mod	24.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				20.00	(1.00×)	31.00	(1.07×)	0.72	(1.02×)	0.05	(5.00×)	0.01						0.03	(1.00×)	0.32	(1.23×)	0.03	(1.00×)	
bm_dag2_log_mod	24.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	3.00	(1.00×)				20.00	(1.00×)	31.00	(1.07×)	0.72	(1.02×)	0.05	(5.00×)							0.03	(1.00×)	0.32	(1.23×)	0.03	(1.00×)	
bm_DL_four_bit_adder_continuous_assign	20.00	(1.00×)		2.00	(1.00×)	9.00	(1.00×)	1.00	(1.00×)				32.00	(1.00×)	34.00	(1.00×)	1.82	(1.00×)	0.04	(4.00×)	0.00	(0.00×)					0.02	(2.00×)	0.27	(1.17×)	0.03	(1.01×)	
bm_DL_4_bit_shift_register	19.00	(1.00×)	1.00	1.00	(1.00×)	7.00	(1.00×)	1.00	(1.00×)				26.00	(1.00×)	56.00	(1.00×)	0.76	(1.00×)	0.03	(3.00×)	0.02	(1.00×)					0.01	(1.00×)	0.27	(1.17×)	0.03	(1.01×)	
bm_DL_BCD_7_segment_without_x	18.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	1.00	(1.00×)				32.00	(1.00×)	76.00	(1.00×)	1.40	(1.00×)	0.05	(5.00×)	0.02	(1.00×)	0.01	(1.00×)			0.02	(1.00×)	0.30	(1.25×)	0.03	(1.00×)	
bm_tester	17.00	(1.00×)	1.00	1.00	(1.00×)	5.00	(1.00×)	4.00	(1.00×)				20.00	(1.00×)	34.00	(0.92×)	0.91	(1.07×)	0.06	(6.00×)	0.02	(2.00×)					0.03	(1.00×)	0.34	(1.26×)	0.03	(0.99×)	
bm_if_reset	17.00	(1.00×)	1.00	1.00	(1.00×)	8.00	(1.00×)	1.00	(1.00×)				26.00	(0.93×)	37.00	(1.28×)	0.68		0.06	(6.00×)	0.02	(1.00×)					0.01	(1.00×)	0.29	(1.32×)	0.03	(1.01×)	•
bm_DL_4_bit_comparator	16.00	(1.00×)		2.00	(1.00×)	8.00	(1.00×)	1.00	(1.00×)				18.00	(0.69×)	43.00	(1.00×)	1.46	(0.90×)	0.05	(5.00×)	0.00	(0.00×)					0.01	(1.00×)	0.27	(1.17×)	0.03	(0.99×)	
bm_DL_2_4_encoder	11.00	(1.00×)		1.00	(1.00×)	3.00	(1.00×)	1.00	(1.00×)				28.00	(1.00×)	32.00	(1.00×)		(1.00×)	0.03	(3.00×)	0.02	(2.00×)					0.01	(1.00×)	0.28	(1.22×)	0.03	(1.01×)	•∗
bm_DL_logic_w_Dff	10.00	(1.00×)	1.00	1.00	(1.00×)	4.00	(1.00×)	1.00	(1.00×)				20.00	(1.00×)	8.00	(1.00×)	1.76	(1.00×)	0.03	(3.00×)	0.00	(0.00×)					0.01	(1.00×)	0.25	(1.19×)	0.03	(1.00×)	
bm_DL_logic_w_Dff2	10.00	(1.00×)	1.00	1.00	(1.00×)	4.00	(1.00×)	1.00	(1.00×)				20.00	(1.00×)	8.00	(1.00×)	1.76	(1.00×)	0.03	(3.00×)	0.01	(1.00×)					0.01	(1.00×)	0.25	(1.19×)	0.03	(1.01×)	
bm_DL_structural_logic2	10.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	1.00	(1.00×)				28.00	(1.00×)	31.00	(1.00×)			0.04	(4.00×)	0.01						0.01	(1.00×)	0.25	(1.19×)	0.03	(1.00×)	•∗
bm_dag1_log	8.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	2.00	(1.00×)				20.00	(1.00×)	25.00	(1.00×)	1.09	(1.00×)	0.03	(3.00×)	0.00	(0.00×)					0.02	(2.00×)	0.27	(1.17×)	0.03	(1.01×)	
bm_dag2_log	8.00	(1.00×)		1.00	(1.00×)	4.00	(1.00×)	2.00	(1.00×)				20.00	(1.00×)	21.00	(1.00×)	1.06	(1.00×)	0.04	(4.00×)	0.01						0.01	(1.00×)	0.27	(1.23×)	0.03	(1.01×)	
bm_DL_4_1_mux	8.00	(1.00×)		1.00	(1.00×)	6.00	(1.00×)	1.00	(1.00×)				10.00	(1.00×)	14.00	(1.00×)	0.89	(1.00×)	0.04	(4.00×)	0.01	(0.50×)					0.01	(1.00×)	0.26	(1.18×)	0.03	(1.00×)	
bm_DL_Dff_w_synch_reset	6.00	(1.00×)	1.00	1.00	(1.00×)	3.00	(1.00×)	1.00	(1.00×)				26.00	(1.00×)	4.00	(1.00×)	1.68	(1.00×)	0.03	(3.00×)	0.01	(1.00×)					0.01	(1.00×)	0.25	(1.14×)	0.03	(1.00×)	
bm_DL_2_cascaded_flip_flops	6.00	(1.00×)	1.00	1.00	(1.00×)	2.00	(1.00×)	1.00	(1.00×)				26.00	(1.00×)	55.00	(1.00×)	0.55	(1.00×)	0.03	(3.00×)	0.01						0.01	(1.00×)	0.25	(1.19×)	0.03	(1.00×)	
bm_DL_2_1_mux	5.00	(1.00×)		1.00	(1.00×)	3.00	(1.00×)	1.00	(1.00×)				26.00	(1.00×)	6.00	(1.00×)	1.79	(1.00×)	0.03	(3.00×)	0.01	(1.00×)					0.01	(1.00×)	0.26	(1.18×)	0.03	(1.00×)	
bm_DL_structural_logic	5.00	(1.00×)		1.00	(1.00×)	3.00	(1.00×)	1.00	(1.00×)				26.00	(1.00×)	6.00	(1.00×)	1.79	(1.00×)	0.03	(3.00×)	0.01	(1.00×)					0.01	(1.00×)	0.24	(1.14×)	0.03	(1.00×)	
bm_dag3_log	4.00	(1.00×)						2.00	(1.00×)				2.00	(1.00×)					0.04	(4.00×)							0.01	(1.00×)	0.26	(1.18×)	0.03	(1.00×)	
bm_my_D_latch1	4.00	(1.00×)		1.00	(1.00×)	2.00	(1.00×)	1.00	(1.00×)				20.00	(1.00×)	10.00	(1.00×)	2.17	(1.00×)	0.03	(3.00×)	0.00	(0.00×)					0.01	(1.00×)	0.24	(1.14×)	0.03	(1.00×)	
bm_dag3_lpm_log	4.00	(1.00×)		1.00	(1.00×)			2.00	(1.00×)				2.00	(1.00×)					0.03	(3.00×)	0.01	(1.00×)					0.01	(1.00×)	0.27	(1.23×)	0.03	(0.99×)	
bm_DL_D_flipflop	4.00	(1.00×)	1.00	1.00	(1.00×)	2.00	(1.00×)	1.00	(1.00×)				20.00	(1.00×)	10.00	(1.00×)	0.68	(1.00×)	0.03	(3.00×)	0.01						0.01	(1.00×)	0.24	(1.20×)	0.03	(1.02×)	
bm_my_D_latch2	2.00	(1.00×)		1.00	(1.00×)	1.00	(1.00×)						14.00	(1.00×)	9.00	(1.00×)	0.21	(1.00×)	0.03	(3.00×)							0.01	(1.00×)	0.24	(1.20×)	0.03	(1.01×)	
bm_add_lpm	0.00	(0.00×)		0.00	(0.00×)	0.00	(0.00×)	0.00	(0.00×)										0.04	(4.00×)	0.01								0.30	(1.43×)	0.03	(0.98×)	†
bm_dag1_lpm	0.00	(0.00×)		1.00	(1.00×)	0.00	(0.00×)	0.00	(0.00×)										0.04	(4.00×)	0.01		0.00	(0.00×)					0.30	(1.36×)	0.03	(0.98×)	†
bm_dag2_lpm	0.00	(0.00×)		1.00	(1.00×)	0.00	(0.00×)	0.00	(0.00×)										0.04	(4.00×)	0.01		0.00	(0.00×)					0.30	(1.43×)	0.03	(0.99×)	†
bm_dag3_lpm	0.00	(0.00×)		1.00	(1.00×)	0.00	(0.00×)	0.00	(0.00×)										0.05	(5.00×)	0.01	(1.00×)	0.00	(0.00×)					0.31	(1.24×)	0.03	(0.98×)	†
bm_dag3_lpm_mod	0.00	(0.00×)		1.00	(1.00×)	0.00	(0.00×)	0.00	(0.00×)										0.08	(8.00×)	0.01	(1.00×)	0.00	(0.00×)					0.35	(1.06×)	0.03	(0.95×)	†
bm_DL_behavioural_full_adder	0.00	(0.00×)		0.00	(0.00×)	0.00	(0.00×)	0.00	(0.00×)										0.03	(3.00×)	0.01								0.30	(1.43×)	0.03	(0.98×)	†
adder_hard_block																													0.01	(0.07×)	0.00	(0.00×)	○‡
GEOMEAN	35.07	(0.96×)		1.23	(1.08×)	9.64	(0.97×)	3.85	(1.09×)	2.02	(0.93×)		20.90	(0.95×)	93.57	(0.91×)	1.71	(1.00×)	0.06	(5.40×)	0.02	(1.04×)	0.03	(1.16×)	0.07	(1.10×)	0.05	(0.97×)	0.47	(1.12×)	0.03	(1.00×)	
GEOMEAN (>100)	351.61	(0.97×)		1.92	(1.38×)	67.96	(0.92×)	28.55	(1.17×)	2.19	(0.93×)		37.85	(0.94×)	2,345.15	(0.90×)	4.41	(1.07×)	0.11	(9.58×)	0.03	(1.16×)	0.08	(1.26×)	0.16	(1.09×)	1.22	(0.88×)	2.65	(1.00×)	0.04	(1.01×)	
% TOTAL (> 100)																			3.78%		2.18%		4.49%		7.43%		51.67%		100.00%				

Runtime in Seconds, Memory in GiB, WL in grid tiles, and CPD in ns.

○ VPR failed to route in the VTR baseline

‡ VPR failed to route in the VTR flow with Yosys front-end

† VPR failed to place and route in the VTR flow with Yosys front-end

• VPR failed to implement an unroutable circuit with a specific channel width factor in the VTR Baseline

∗ VPR failed to implement an unroutable circuit with a specific channel width factor in the VTR flow with Yosys front-end

Table A.2: Micro Benchmarks using Yosys Front-end in the VTR Flow (Relative to the VTR Baseline).

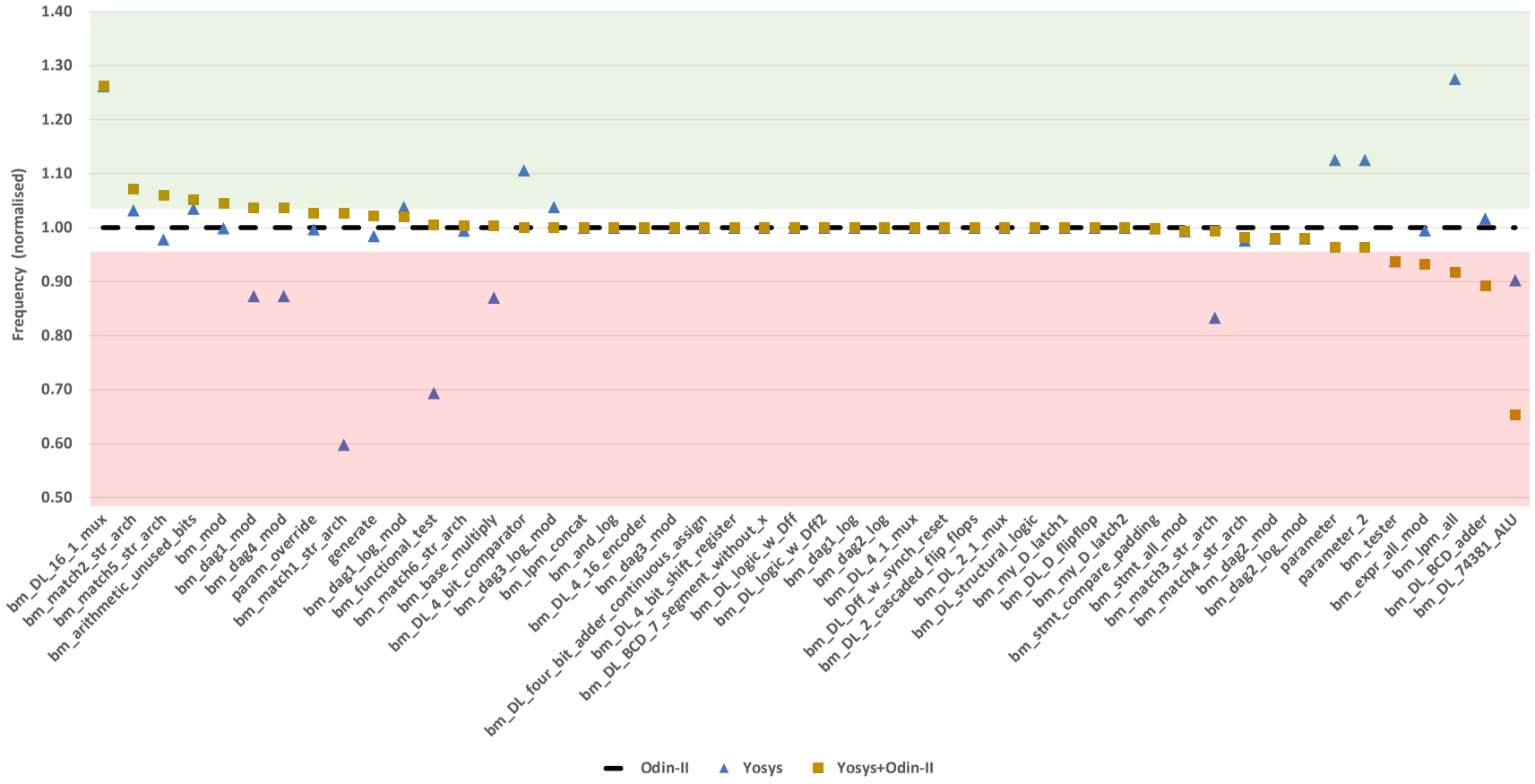


Figure A.1: The Micro Benchmarks Frequency Comparison. Benchmarks Resulted in Errors or an Empty Netlists are Removed. (Normalised to the VTR Baseline).

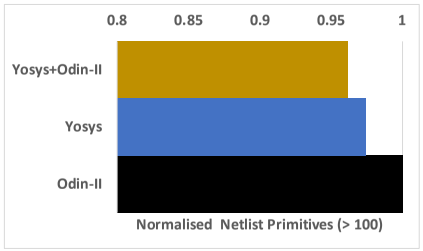


Figure A.2: Netlist Primitives for Micro Benchmarks with all Front-ends.

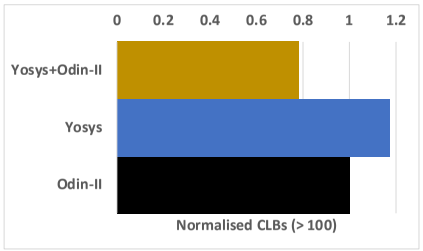


Figure A.3: CLBs for Micro Benchmarks with all Front-ends.

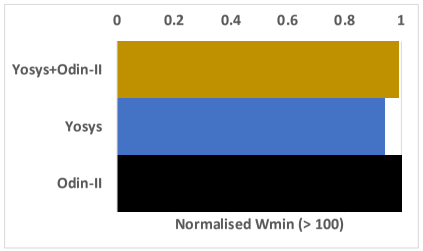


Figure A.4: W_*min* for Micro Benchmarks with all Front-ends.

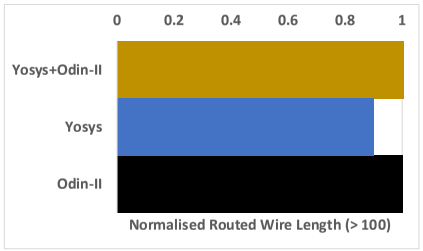


Figure A.5: Routed Wire Lengths for Micro Benchmarks with all Front-ends.

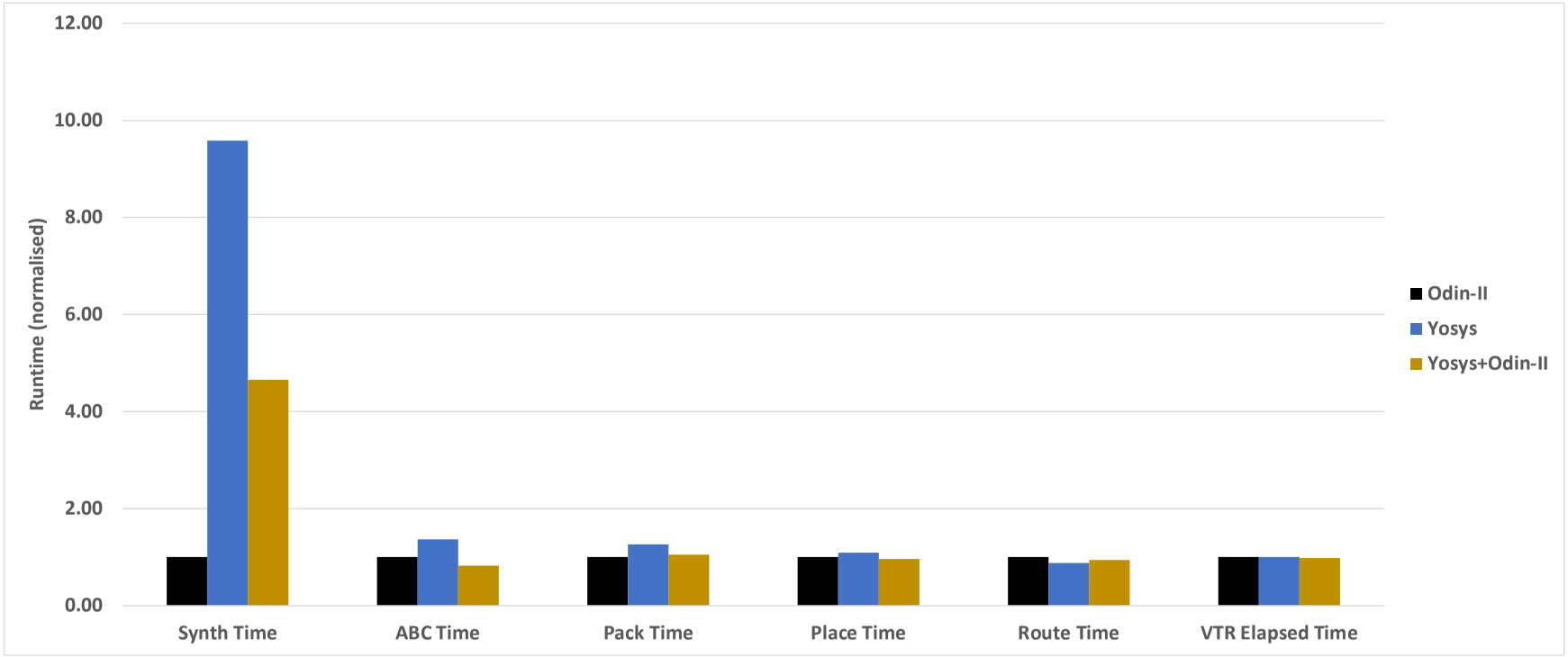


Figure A.6: VTR Flow Sectional Runtime for Micro Benchmarks with all Front-ends.

Appendix B

FIR Benchmarks

Benchmark	Netlist Primitives		Clocks	Logic Depth		I/Os		CLBs		DSPs	RAMs	W_{min}		Routed Wirelength (1.3 W_{min})		Routed CPD (1.3 W_{min})		Synth Time (Yosys+Odin-II))		ABC Time		Pack Time		Place Time		Route Time (Find W_{min})		VTR Flow Elapsed Time		Peak Memory	
ex4LS16_fir	33,208.00	(0.79×)	1.00	1.00	(1.00×)	13.00	(1.00×)	1,242.00	(0.74×)			92.00	(1.02×)	186,894.00	(0.84×)	2.86	(0.86×)	5.54	(3.15×)	1.06	(0.60×)	9.19	(0.79×)	15.57	(0.63×)	62.20	(0.78×)	117.67	(0.73×)	0.39	(0.73×)
ex4PM16_fir_152	22,620.00	(0.82×)	1.00	1.00	(1.00×)	13.00	(1.00×)	824.00	(0.78×)			82.00	(0.95×)	116,347.00	(0.84×)	2.72	(0.85×)	3.26	(3.26×)	0.92	(0.81×)	6.92	(0.83×)	8.53	(0.64×)	29.74	(0.68×)	64.30	(0.73×)	0.27	(0.80×)
ex2PM16_fir_119	17,887.00	(0.82×)	1.00	1.00	(1.00×)	13.00	(1.00×)	663.00	(0.80×)			76.00	(0.97×)	88,117.00	(0.80×)	2.66	(0.94×)	2.29	(3.18×)	0.50	(0.57×)	5.73	(0.81×)	6.49	(0.68×)	25.16	(0.75×)	52.39	(0.77×)	0.22	(0.84×)
ex3PM16_fir_61	9,868.00	(0.84×)	1.00	1.00	(1.00×)	13.00	(1.00×)	367.00	(0.81×)			76.00	(1.09×)	47,236.00	(0.85×)	2.32	(0.80×)	1.14	(3.35×)	0.24	(0.67×)	3.24	(0.81×)	3.17	(0.76×)	12.19	(0.79×)	26.50	(0.81×)	0.13	(0.85×)
ex1LS16_fir_41	6,646.00	(0.85×)	1.00	1.00	(1.00×)	13.00	(1.00×)	237.00	(0.80×)			64.00	(1.00×)	30,145.00	(0.85×)	2.33	(0.76×)	0.80	(3.64×)	0.16	(0.70×)	2.37	(0.83×)	1.78	(0.69×)	5.86	(0.88×)	15.30	(0.85×)	0.10	(0.89×)
ex2BT16_fir_71	6,379.00	(0.72×)	1.00	1.00	(1.00×)	13.00	(1.00×)	252.00	(0.76×)			68.00	(0.92×)	27,999.00	(0.83×)	2.56	(0.92×)	2.49	(8.89×)	0.16	(0.57×)	1.80	(0.57×)	1.91	(0.60×)	8.24	(0.80×)	18.84	(0.80×)	0.10	(0.79×)
ex1PM16_fir_28	4,294.00	(0.86×)	1.00	1.00	(1.00×)	13.00	(1.00×)	147.00	(0.83×)			64.00	(1.00×)	19,709.00	(0.88×)	2.64	(1.05×)	0.53	(4.08×)	0.10	(0.71×)	1.46	(0.94×)	1.09	(0.71×)	3.70	(0.81×)	9.71	(0.85×)	0.07	(0.90×)
ex1BT16_fir_20	2,708.00	(0.84×)	1.00	1.00	(1.00×)	13.00	(1.00×)	96.00	(0.80×)			62.00	(0.97×)	11,554.00	(0.81×)	2.41	(0.91×)	0.47	(5.22×)	0.07	(0.78×)	0.90	(0.71×)	0.68	(0.76×)	2.33	(0.65×)	6.29	(0.77×)	0.05	(0.89×)
ex4EP16_fir_10	1,659.00	(0.85×)	1.00	1.00	(1.00×)	13.00	(1.00×)	59.00	(0.83×)			66.00	(1.27×)	6,584.00	(0.77×)	2.03	(0.95×)	0.22	(4.40×)	0.03	(0.60×)	0.56	(0.93×)	0.41	(0.82×)	1.62	(0.90×)	4.00	(0.88×)	0.04	(0.93×)
ex2EP16_fir_13	1,351.00	(0.93×)	1.00	1.00	(1.00×)	13.00	(1.00×)	49.00	(0.96×)			50.00	(0.96×)	6,506.00	(1.02×)	2.17	(0.98×)	0.23	(5.75×)	0.03	(0.75×)	0.42	(0.88×)	0.29	(0.81×)	1.20	(0.78×)	3.21	(0.89×)	0.04	(0.97×)
ex1EP16_fir_6	955.00	(0.87×)	1.00	1.00	(1.00×)	13.00	(1.00×)	34.00	(0.89×)			52.00	(1.08×)	4,023.00	(0.88×)	2.28	(1.07×)	0.15	(5.00×)	0.03	(1.50×)	0.28	(0.88×)	0.20	(0.83×)	0.81	(0.71×)	2.21	(0.82×)	0.04	(0.94×)
GEOMEAN	5,502.12	(0.83×)		1.00	(1.00×)	13.00	(1.00×)	200.13	(0.82×)			67.34	(1.02×)	25,525.90	(0.85×)	2.44	(0.91×)	0.85	(4.31×)	0.14	(0.72×)	1.74	(0.81×)	1.59	(0.72×)	6.01	(0.77×)	14.33	(0.81×)	0.10	(0.86×)
GEOMEAN (> 1K)	19,082.07	(0.82×)		1.00	(1.00×)	13.00	(1.00×)	706.41	(0.78×)			81.25	(1.01×)	97,537.33	(0.83×)	2.63	(0.86×)	2.62	(3.23×)	0.58	(0.65×)	5.86	(0.81×)	7.23	(0.68×)	27.44	(0.75×)	56.93	(0.83×)	0.24	(0.80×)
% TOTAL > 1K																	5.31%		1.03%		10.21%		12.58%		47.95%		100.00%				

Runtime in Seconds, Memory in GiB, WL in grid tiles, and CPD in ns.

Table B.1: FIR Benchmarks using Yosys+Odin-II Front-end in the VTR Flow (Relative to the VTR Baseline)

Benchmark	Netlist Primitives		Clocks	Logic Depth		I/Os		CLBs		DSPs	RAMs	W_{min}		Routed Wirelength (1.3 W_{min})		Routed CPD (1.3 W_{min})		Synth Time (Yosys+Odin-II))		ABC Time		Pack Time		Place Time		Route Time (Find W_{min})		VTR Flow Elapsed Time		Peak Memory	
ex4LS16_fir	32,121.00	(0.76×)	1.00	5.00	(5.00×)	13.00	(1.00×)	1,257.00	(0.75×)			100.00	(1.11×)	182,365.00	(0.82×)	3.25	(0.98×)	7.38	(4.19×)	3.37	(1.89×)	8.54	(0.73×)	16.44	(0.67×)	53.95	(0.68×)	115.19	(0.71×)	0.39	(0.73×)
ex4PM16_fir_152	22,226.00	(0.81×)	1.00	5.00	(5.00×)	13.00	(1.00×)	857.00	(0.81×)			100.00	(1.16×)	119,916.00	(0.87×)	3.38	(1.06×)	4.50	(4.50×)	1.77	(1.55×)	7.96	(0.95×)	11.04	(0.83×)	36.24	(0.82×)	79.90	(0.91×)	0.27	(0.82×)
ex2PM16_fir_119	17,471.00	(0.81×)	1.00	6.00	(6.00×)	13.00	(1.00×)	662.00	(0.80×)			98.00	(1.26×)	88,553.00	(0.80×)	3.13	(1.11×)	3.85	(5.35×)	1.29	(1.47×)	5.80	(0.82×)	8.19	(0.86×)	26.37	(0.79×)	59.66	(0.88×)	0.23	(0.86×)
ex3PM16_fir_61	9,598.00	(0.82×)	1.00	5.00	(5.00×)	13.00	(1.00×)	367.00	(0.81×)			78.00	(1.11×)	43,712.00	(0.79×)	2.73	(0.95×)	1.85	(5.44×)	0.49	(1.36×)	3.40	(0.85×)	3.43	(0.82×)	13.46	(0.87×)	29.67	(0.90×)	0.13	(0.86×)
ex2BT16_fir_71	7,757.00	(0.87×)	1.00	1.00	(1.00×)	13.00	(1.00×)	274.00	(0.83×)			80.00	(1.08×)	30,475.00	(0.90×)	2.57	(0.92×)	1.65	(5.89×)	0.28	(1.00×)	2.59	(0.82×)	2.60	(0.82×)	10.41	(1.00×)	23.30	(0.99×)	0.12	(0.94×)
ex1LS16_fir_41	6,410.00	(0.82×)	1.00	6.00	(6.00×)	13.00	(1.00×)	244.00	(0.82×)			98.00	(1.53×)	27,915.00	(0.79×)	2.96	(0.97×)	1.30	(5.91×)	0.30	(1.30×)	2.38	(0.84×)	2.35	(0.91×)	8.82	(1.32×)	20.19	(1.12×)	0.11	(0.98×)
ex1PM16_fir_28	4,154.00	(0.83×)	1.00	4.00	(4.00×)	13.00	(1.00×)	161.00	(0.91×)			72.00	(1.13×)	21,018.00	(0.94×)	2.66	(1.05×)	0.88	(6.77×)	0.16	(1.14×)	1.81	(1.16×)	1.54	(1.01×)	4.95	(1.08×)	12.56	(1.10×)	0.07	(0.94×)
ex1BT16_fir_20	2,706.00	(0.84×)	1.00	1.00	(1.00×)	13.00	(1.00×)	98.00	(0.82×)			72.00	(1.13×)	13,702.00	(0.96×)	2.56	(0.96×)	0.73	(8.11×)	0.08	(0.89×)	0.97	(0.76×)	0.90	(1.01×)	3.82	(1.07×)	8.66	(1.06×)	0.06	(0.94×)
ex4EP16_fir_10	1,611.00	(0.83×)	1.00	1.00	(1.00×)	13.00	(1.00×)	58.00	(0.82×)			64.00	(1.23×)	7,023.00	(0.82×)	2.37	(1.11×)	0.39	(7.80×)	0.04	(0.80×)	0.66	(1.10×)	0.48	(0.96×)	1.62	(0.90×)	4.46	(0.99×)	0.04	(0.92×)
ex2EP16_fir_13	1,381.00	(0.96×)	1.00	1.00	(1.00×)	13.00	(1.00×)	49.00	(0.96×)			64.00	(1.23×)	5,654.00	(0.89×)	2.32	(1.05×)	0.43	(10.75×)	0.04	(1.00×)	0.50	(1.04×)	0.40	(1.11×)	1.58	(1.03×)	4.13	(1.15×)	0.04	(0.99×)
ex1EP16_fir_6	916.00	(0.83×)	1.00	1.00	(1.00×)	13.00	(1.00×)	32.00	(0.84×)			64.00	(1.33×)	3,320.00	(0.73×)	2.32	(1.09×)	0.28	(9.33×)	0.02	(1.00×)	0.29	(0.91×)	0.23	(0.96×)	1.08	(0.95×)	2.84	(1.06×)	0.04	(0.94×)
GEOMEAN	5,489.29	(0.83×)		2.44	(2.44×)	13.00	(1.00×)	203.74	(0.83×)			79.61	(1.20×)	25,313.08	(0.84×)	2.73	(1.02×)	1.28	(6.46×)	0.24	(1.18×)	1.93	(0.90×)	1.99	(0.90×)	7.34	(0.94×)	17.38	(0.98×)	0.10	(0.90×)
GEOMEAN (> 1K)	18,601.04	(0.80×)		5.23	(5.23×)	13.00	(1.00×)	715.25	(0.79×)			93.50	(1.16×)	95,919.22	(0.82×)	3.11	(1.02×)	3.92	(4.84×)	1.39	(1.56×)	6.05	(0.84×)	8.45	(0.79×)	28.86	(0.79×)	63.53	(0.82×)	0.24	(0.82×)
% TOTAL > 1K																	6.37%		2.20%		9.65%		13.28%		45.15%		100.00%				

Runtime in Seconds, Memory in GiB, WL in grid tiles, and CPD in ns.

Table B.2: FIR Benchmarks using Yosys Front-end in the VTR Flow (Relative to the VTR Baseline)

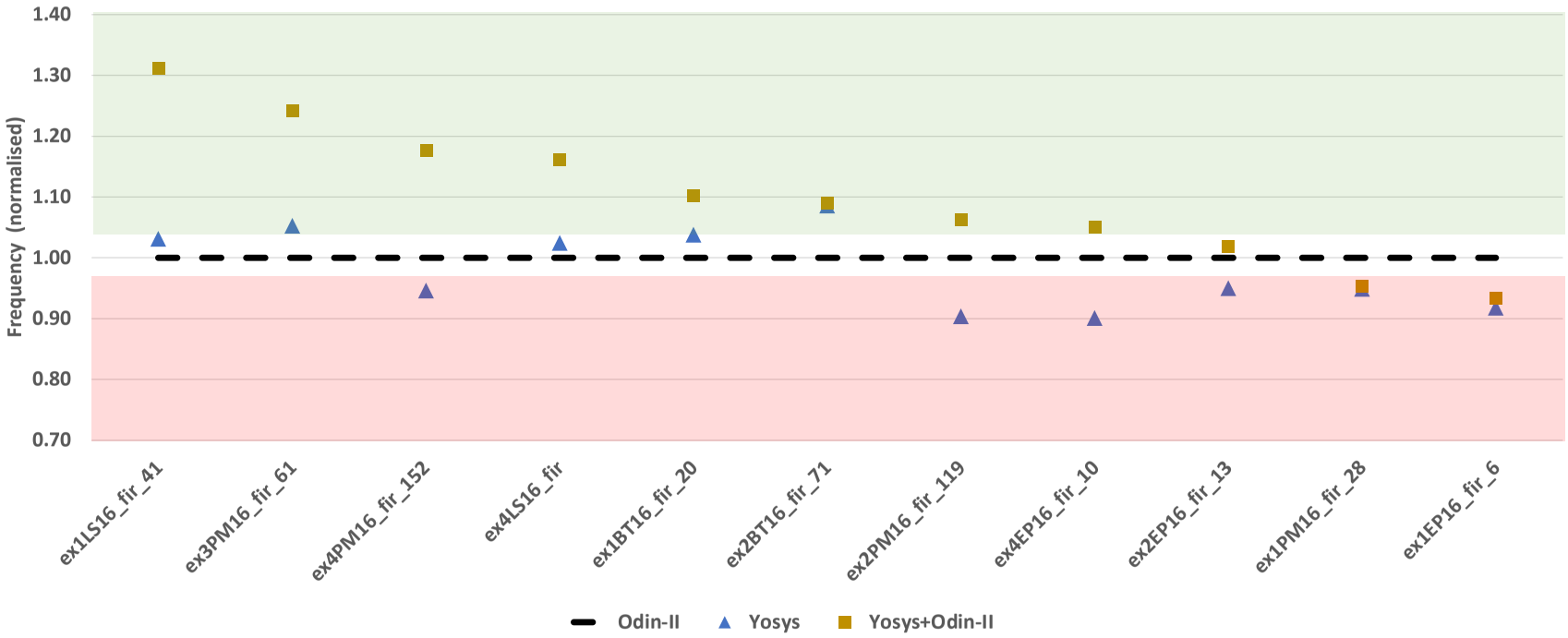


Figure B.1: The FIR Benchmarks Frequency Comparison. (Normalised to the VTR Baseline).

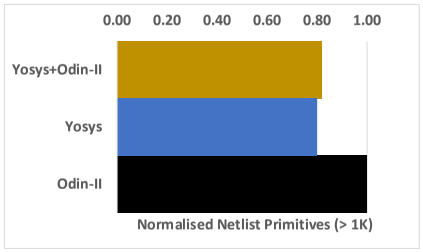


Figure B.2: Netlist Primitives for FIR Benchmarks with all Front-ends.

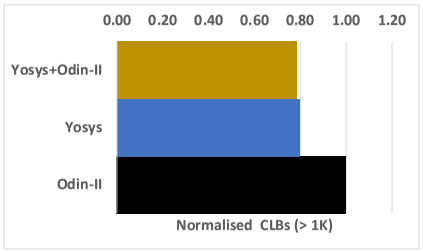


Figure B.3: CLBs for FIR Benchmarks with all Front-ends.

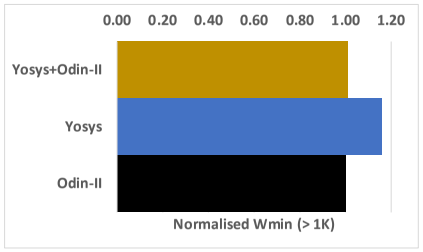


Figure B.4: W_*min* for FIR Benchmarks with all Front-ends.

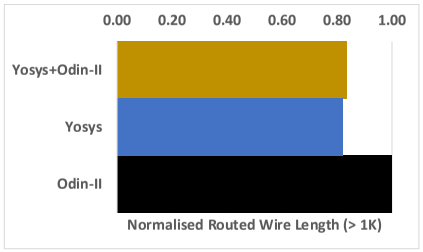


Figure B.5: Routed Wire Lengths for FIR Benchmarks with all Front-ends.

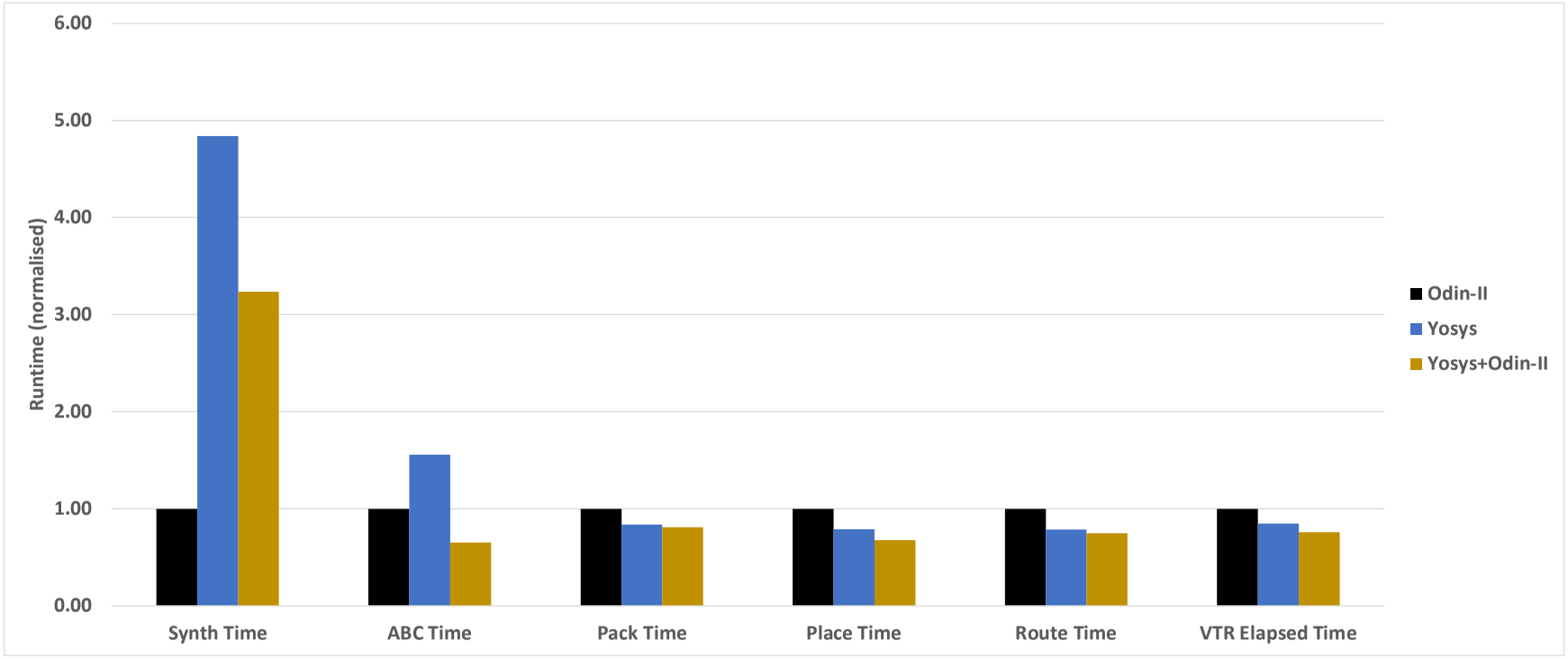


Figure B.6: VTR Sectional Runtime for FIR Benchmarks with all Front-ends.

Appendix C

Yosys+VTR Synthesis Script

```
# XXX (input circuit) is replaced with filename by the run_vtr_flow script
read_verilog -nolatches XXX
# These commands follow the generic 'synth'
# command script inside Yosys
# The -libdir argument allows Yosys to search the current
# directory for any definitions to modules it doesn't know
# about, such as hand-instantiated (not inferred) memories
hierarchy -check -auto-top -libdir .
proc
# Check that there are no combinational loops
scc -select
select -assert-none %
select -clear
# Optimization, FSM extraction and memory inference
opt_expr; opt_clean; check; opt -nodffe -nosdff;
fsm; opt; wreduce; peepopt; opt_clean; share;
opt; memory -nomap; opt -full;
# Transform all async FFs into synchronous ones
# (TTT) will be replaced with ODIN_TECHLIB by the python script
techmap -map +/adff2dff.v
techmap -map TTT/adffe2dff.v
# Map multipliers, DSPs, and add/subtracts according to yosys_models.v
# (YYY) will be replaced with yosys_models.v by the python script
techmap -map YYY */t:$mul */t:$mem */t:$sub */t:$add
opt -fast -full
memory_map
```

```
# Taking care to remove any undefined muxes that
# are introduced to promote resource sharing
opt -full
# Then techmap all other 'complex' blocks into basic
# (lookup table) logic
techmap; opt -fast;
# We read the definitions for all the VTR primitives
# as blackboxes
read_verilog -lib TTT/adder.v
read_verilog -lib TTT/multiply.v
#( SSS) and (DDD) will be replaced with single_port_ram.v
# and dual_port_ram.v by the python script, respectively
read_verilog -lib SSS
read_verilog -lib DDD
# Rename singlePortRam to single_port_ram
# Rename dualPortRam to dual_port_ram
# rename function of Yosys not work here
# since it may result in a hierarchy error
#(SSR) and (DDR) will be replaced with spram_rename.v
# and dpram_rename.v by the python script, respectively
read_verilog SSR
read_verilog DDR
# Flatten the netlist
flatten
# Turn all DFFs into simple latches
dffunmap
opt -fast -noff
# Lastly, check the hierarchy for any unknown modules,
# and purge all modules (including blackboxes) that
# aren't used
hierarchy -check -purge_lib
tee -o /dev/stdout stat
autoname
# Then write it out as a blif file, remembering to call
# the internal '$true'/' $false' signals vcc/gnd, but
# switch '-impltf' doesn't output them
# ZZZ will be replaced by run_vtr_flow.pl
write_blif -true + vcc -false + gnd -undef + unconn -blackbox ZZZ
```

Listing C.1: Yosys+VTR Synthesis Script

Vita

Candidate's full name
Seyed Alireza Damghani

University attended (with dates and degrees obtained)
Bachelor with Honours of Computer Science, University of Tehran, 2018

Publications

- Seyed Alireza Damghani, Jean-Philippe Legault, and Kenneth B. Kent. **Desired Footprint by Technology Mapping Modification using a Genetic Algorithm in Odin II**. In International Workshop on Rapid System Prototyping, RSP 2020, Hamburg, Germany, September 24-25, 2020, pages 1-7. IEEE, 2020. doi: 10.1109/RSP51120.2020.9244862

- Aman Arora, Andrew Boutros, Daniel Rauch, Aishwarya Rajen, Aatman Borda, Seyed Alireza Damghani, Samidh Mehta, Sangram Kate, Pragnesh Patel, Kenneth B. Kent, Vaughn Betz, and Lizy K. John. **Koios: A deep learning benchmark suite for FPGA architecture and CAD research**. CoRR, abs/2106.07087, 2021. arXiv:2106.07087

- Harpreet Kaur, Georgiy Krylov, Seyed Alireza Damghani, and Kenneth B. Kent. **Heterogeneous Logic Implementation for Adders in VTR**. In International Workshop on Rapid System Prototyping, RSP 2021, Virtual Event, October 10-15, 2021. IEEE, 2021.

Dissertations

- Seyed Alireza Damghani and Mostafa Ersali Salehinasab. **Development and Optimization of Reconfigurable Hardware to Compensate Computationally-Intensive Sections of Image Processing Algorithms**. In Honours Thesis Defended at University of Tehran. University of Tehran, Tehran, Tehran, Iran, September 1, 2018.

Talks

- Seyed Alireza Damghani, Jean-Philippe Legault, and Kenneth B. Kent. **Desired Footprint by Technology Mapping Modification using a Genetic Algorithm in Odin II**. In Embedded Systems Week (ESWEEK) 2020, https://whova.com/embedded/speaker/eswe_202009/12014668/