Integration of Bit-Serial SERV RISC-V Core into Chipyard: Expanding Open-Source SoC Ecosystems for Compact, Low-Power Designs

Submitted By

Group Number: 01

Name of Student	**Registration Number**
Ayesha Qazi	21ABELT0895
Humail Nawaz	21ABELT0892

Supervisor
Engr. Mehmoona Gul
Lecturer - Department of Electronic Engineering,
UET Peshawar - Abbottabad Campus

DEPARTMENT OF ELECTRONIC ENGINEERING
UNIVERSITY OF ENGINEERING AND TECHNOLOGY
PESHAWAR

August 2025

Integration of Bit-Serial SERV RISC-V Core into Chipyard: Expanding Open-Source SoC Ecosystems for Compact, Low-Power Designs

Submitted By

Name of Student	Registration Numbers
Humail Nawaz	21ABELT0892
Ayesha Qazi	21ABELT0895

A report submitted in partial fulfillment of the requirements for the degree of B.Sc. Electronic Engineering

Supervisor
Engr. Mehmoona Gul
Lecturer - Department of Electronic Engineering,
UET Peshawar - Abbottabad Campus

Co-Supervisor
Dr. Anees Ullah
Assistant Professor - Department of Electronic Engineering,
UET Peshawar - Abbottabad Campus

Head of Department Signature: ________________________________

External Examiner Signature: ________________________________

Thesis Supervisor Signature: ________________________________

DEPARTMENT OF ELECTRONIC ENGINEERING
UNIVERSITY OF ENGINEERING AND TECHNOLOGY
PESHAWAR - ABBOTTABAD CAMPUS

August 2025

Abstract

The call for energy-efficient processing, driven by the slowdown of Moore's Law demands system-on-chip architectures designed to optimize power and area utilization, yet deliver optimal performance metrics. Such systems-on-chip are the need of every device one comes across today. This effort addresses the limitations of the Chipyard System-on-Chip Framework to support and facilitate ultra-compact low-power SoCs by means of integrating bit-serial [3] RISC-V processor with Chipyard. This not only expands Chipyard's core ecosystem but also allows creation of resource-constrained lightweight SoCs. The integration has been carried out in incremental stages. This report comprehensively covers the integration approach, procedure, design modifications, challenges, iterative debugging, and most suitable solutions. The design, when completed, successfully enables reliable communication between the SERV core and Chipyard's SoC environment that included buses, interrupt controllers, BootROM and custom peripherals, validated through functional tests and simulation. This thesis not only delivers a functional low-power RISC-V SoC design but also provides a detailed methodology, enabling future researchers to extend and adapt the framework with minimal rework.

Keywords: RISC-V, System-on-Chip Design，Minimal Bit-Serial Processors，Chipyard，Wishbone，AXI, Tilelink, Chisel HDL, Verilog Blackbox, Finite State Machines, Tiles, Adapters

Innovation Points

First Ever Bit-Serial Processor in Chipyard

Ultra Minimalist Tile

Custom FSM-Based Bidirectional Protocol Conversion Bridge

Ecosystem Enhancement of Chipyard for Resource-Constrained Devices

List of Abbreviations

The following table describes the significance of various abbreviations and acronyms used throughout the thesis.

Abbreviation	Acronym
RISC-V	Reduced Instruction Set Computer (Fifth Generation)
ISA	Instruction Set Architecture
SoC	System-on-Chip
HDL	Hardware Description Language
HCL	Hardware Construction Language
SERV	Serial Risc-V
BOOM	Berkely Out-of-Order Machine
FESVR	Front-End Server
AXI	Advanced eXtensible Interface
AMBA	Advanced Microcontroller Bus Architecture
TL	Tilelink
DTS	Device Tree Source
PLIC	Platform Level Interrupt Controller
CLINT	Core Local Interrupt
FPGA	Field Programmable Gate Array
IP	Intellectual Property
CHISEL	Constructing Hardware in Scala Embedded Language
MDU	Multiply/Divide Unit
GPR	General Purpose Registers
CSRs	Control and Status Registers
DMA	Direct Memory Access
LUT	Look-up Table

List of Figures

Figure 1-1: Project Roadmap I
Figure 2-1: First pipeline stage of Rocket Core 10
Figure 2-2: Rocket Core's complete pipeline 11
Figure 2-3: BOOM architecture and pipeline 12
Figure 2-4: CVA6 System level diagram 13
Figure 2-5: Tilelink Connectivity 15
Figure 2-6: TL-UL Messages 16
Figure 2-7: TL-UH Messages 18
Figure 2-8: TL-C Messages 20
Figure 2-9: Rocket Tile 21
Figure 3-1: Adder.scala File 27
Figure 3-2: Adder example in DigitalTop.scala 31
Figure 3-3: Adder Configs in MMIOAcceleratorConfigs 32
Figure 3-4: SERV SoC Make 33
Figure 3-5: SERV SoC Make successful 33
Figure 3-6: AdderTLRocketConfig 34
Figure 3-7: Adder binary and .riscv files generated 34
Figure 3-8: Adder peripheral verified 35
Figure 3-9: Request transaction from SoC to Adder 36
Figure 3-10: Response from Adder to SoC 36
Figure 3-11: Rocket Core's DTS includes Adder Peripheral 37
Figure 4-1: Integration Flow 38
Figure 4-2: Blackbox Block representation 39
Figure 4-3: AXI-Tilelink Adapter Chain 49
Figure 4-4: Tilelink-AXI Adapter Chain 50
Figure 4-5: Filesystem for Integration 51
Figure 5-1: SERV Top Block Diagram 52
Figure 5-2: SERV Schematic 55
Figure 5-3: SERV ALU Block Diagram 55
Figure 5-4: Addition/Subtraction Unit 56
Figure 5-5: Serv ALU Schematic 58
Figure 5-6: ALU Timing Diagram 58

Figure 5-7: SERV Bufreg Block Diagram 59
Figure 5-8: SERV Bufreg Schematic 59
Figure 5-9: SERV Bufreg Schematic 60
Figure 5-10: Bufreg2 Block Diagram 60
Figure 5-11: Bufreg2 Timing Diagram 61
Figure 5-12: Bufreg2 Schematic 61
Figure 5-13: SERV CSR Block Diagram 62
Figure 5-14: CSR Schematic 63
Figure 5-15: CSR Timing Diagram 64
Figure 5-16: Control Block Diagram 64
Figure 5-17: SERV Control Unit Logic 65
Figure 5-18: SERV Control Unit Schematic 66
Figure 5-19: Timing Diagram of serv_ctrl 66
Figure 5-20: Decode Unit Block Diagram 67
Figure 5-21: Decode logic representation 67
Figure 5-22: Decode Unit Timing Diagram 69
Figure 5-23: Immdec Block Diagram 70
Figure 5-24: Decode Logic 70
Figure 5-25: Immdec Unit Schematic 71
Figure 5-26: Immdec Unit Timing Diagram 71
Figure 5-27: SERV Memory Interface 72
Figure 5-28: Memory Interface Schematic 72
Figure 5-29: Logic Representation 73
Figure 5-30: Serv Memory Interface Timing Diagram 73
Figure 5-31: RF Interface Block Diagram 74
Figure 5-32: RF Interface Logic Representation 74
Figure 5-33: RF Interface Schematic 75
Figure 5-34: RF Interface Timing Diagram 75
Figure 5-35: SERV State Unit Timing Diagram 77
Figure 5-36: Instruction Fetch 78
Figure 5-37: Instruction Decode Timing Diagram 79
Figure 5-38: Handshake between wreq and rf_ready 79
Figure 5-39: Timing Diagram for single-stage execution 80

Figure 5-40: Trap asserted for external interrupts and ecall/ebreak operations 80
Figure 5-41: Two-stage SERV operations representation 81
Figure 5-42: Servile Schematic 82
Figure 5-43: Servile Timing Diagram 83
Figure 5-44: Servile Mux Schematic 84
Figure 5-45: Servile Mux Timing Diagram 85
Figure 5-46: Servile Arbiter Schematic 86
Figure 5-47: Servile Arbiter Timing Diagram 86
Figure 5-48: RF RAM Interface Schematic 87
Figure 5-49: RF RAM Interface Timing Diagram 88
Figure 5-50: Serving Schematic 89
Figure 5-51: Serving RAM 91
Figure 5-52: Servile Mem Interface Schematic 91
Figure 5-53: Servile Mem Interface Timing Diagram 92
Figure 5-54: Modified Serving 94
Figure 5-55: Serving Timing Diagram 94
Figure 7-1: Bridge State Diagram 104
Figure 7-2: Bridge Testing with dummy Wishbone IP 105
Figure 7-3: AXI2WB Transaction verification 105
Figure 7-4: AXI Dummy IP for Testing WB2AXI Transactions 106
Figure 7-5: WB2AXI Verification 106
Figure 7-6: Serving connected to Bridge 107
Figure 7-7: AXI-Serving Write Transaction 108
Figure 7-8: AXI-Serving Read Transaction 109
Figure 7-9: Transactions propagated from Serving 110
Figure 7-10: Transactions translated into AXI 110
Figure 8-1: ConfigMixins.scala File 114
Figure 8-2: ServConfigs.scala File 114
Figure 8-3: Build.sbt in SERV 115
Figure 8-4: Addition of SERV to Chipyard's core list 115
Figure 8-5: SERV in Chipyard's main Build.sbt 115
Figure 8-6: SERV SoC Make 116
Figure 8-7: SERV SoC Make successful 116

Figure 8-8: SERV SoC Make successful 117
Figure 8-9: SERV DTS in Chipyard 117
Figure 8-10: SERV SoC Block Diagram 118
Figure 9-1: ServConfigs added to FPGA Configuration File 119
Figure 9-2: Make command for Bitstream Generation 120
Figure 9-3: Make flow in Vivado 120
Figure 9-4: Synthesis started 121
Figure 9-5: RTL Optimization Step 121
Figure 9-6: Synthesis Progress 122
Figure 9-7: Synthesis step completed 123
Figure 9-8: DRC step begins 123
Figure 9-9: Power Optimizations in Implementation 124
Figure 9-10: Design Rule Check completed 124
Figure 9-11: Placer Initialization 124
Figure 9-12: Global Placement 125
Figure 9-13: Detail Placement 125
Figure 9-14: Post Placement Optimization and Cleanup 125
Figure 9-15: Design Placement Successful 126
Figure 9-16: Routing Design 126
Figure 9-17: Design Routed successfully 126
Figure 9-18: Bitstream Generation completed 127
Figure 9-19: SERV SoC running on Arty A7 35T 127

List of Tables

Table 2-1: Tilelink Channels 15
Table 2-2: TL-UL Signals 17
Table 2-3: TL-UH Signals 18
Table 2-4: TL-C Signals 20
Table 5-1: Servile Parameters 82
Table 9-1: SERV SoC LUT Utilization 128

Table of Contents

Abstract III

Innovation Points IV

List of Abbreviations V

Table of Contents XI

1 Introduction and Background 1

1.1 Overview of the Project 1

1.1.1 Project Motivation 1

1.1.2 Problem Statement 1

1.1.2.1 Research Questions 2

1.1.2.2 Scope of the Project 2

1.2 Background 3

1.3 Objectives of Thesis 3

1.4 Project Roadmap 4

1.4.1 Features of the Project 4

1.5 Goals and Milestones Achieved 5

1.5.1 Key Goals Set 5

1.5.2 Milestones Achieved 5

1.6 Difficulties 6

1.7 Outline of Dissertation 6

2 Chipyard 9

2.1 Introduction 9

2.1.1 What is Chipyard 9

2.1.2 The Need for Chipyard 9

2.2 Chipyard Ecosystem 9

2.2.1 Cores 9
2.2.1.1 Rocket Core: 9
2.2.1.2 BOOM: 11
2.2.1.3 CVA6: 13
2.2.1.4 Ibex: 13
2.2.2 Accelerators 13
2.2.3 Toolchains 14
2.2.4 Sims 14
2.3 SoC Architecture in Chipyard 14
2.3.1 TileLink Interconnect Overview [2] 14
2.3.1.1 Channels 15
2.3.1.2 Conformance Levels and Messages 16
2.3.2 Tile Structure 21
2.3.3 Memory Hierarchy and Buses 22
2.3.4 Diplomatic Widgets and Adapters 23
2.3.4.1 TLBuffer 23
2.3.4.2 TLFragmenter 23
2.3.4.3 TLSourceShrinker 23
2.3.4.4 TLWidthWidget 23
2.3.4.5 TLFIFOFixer 23
2.3.4.6 AXI4Fragmenter 24
2.3.4.7 AXI4Buffer 24
2.3.4.8 AXI4UserYanker 24
2.3.4.9 AXI4Deinterleaver 24
2.3.4.10 AXI4IdIndexer 24
2.3.4.11 TLToAXI4 and AXI4ToTL 24

2.4 CHISEL: Chipyard’s Primary Hardware Construction Language 25
2.4.1 Overview of CHISEL 25
2.4.2 Rationale for Using Chisel in Chipyard 25
2.4.3 Key Benefits for SoC Design 25
2.4.4 Chisel in the Chipyard Ecosystem 25
2.5 Simulation and Prototyping Flows 26
2.5.1 Software RTL Simulation with Verilator 26
2.5.2 FPGA Prototyping using FPGA Shells 26
2.6 Chapter Summary 26
3 Peripheral Integration with Chipyard 27
3.1 Introduction 27
3.2 Designing the Peripheral 27
3.2.1 Overview of the Adder Peripheral 27
3.2.2 Chisel Implementation 27
3.3 Configuring Chipyard for the Peripheral 31
3.3.1 Inclusion in Digital Top: 31
3.3.2 Modifying MMIO Accelerator Configs 32
3.4 Adder BlackBox 32
3.5 Simulation Workflow 33
3.5.1 Building the SoC with the Peripheral 33
3.5.2 Preparing the Test Program in C 34
3.5.3 Compiling and Linking for RISC-V 34
3.5.4 Running the Binary in Verilator 35
3.6 Results and Observations 35
3.6.1 Addition Verification: 35
3.6.2 Communication Path: Rocket Core to Adder Peripheral 35

3.6.3 Adder Peripheral in the Device Tree Source (DTS) 37
3.7 Chapter Summary 37
4 Core Integration Overview 38
4.1 Introduction 38
4.1.1 Purpose and importance of integrating a custom core into Chipyard. 38
4.1.2 Overview of integration steps 38
4.2 BlackBox Wrapping for Core Integration 39
4.2.1 Verilog BlackBox Wrapping 39
4.2.2 Scala BlackBox Definition 39
4.2.3 Practical Considerations and Limitations 41
4.3 Defining Core and Tile Parameters 41
4.3.1 Crafting CoreParams and TileParams case classes 41
4.3.2 InstantiableTileParams 42
4.3.3 Documentation and handling of “Rocket-specific” parameters 43
4.4 The Tile Class 43
4.4.1 Custom Class 43
4.4.2 Connecting TileLink master/slave nodes to diplomatic fabric 44
4.4.3 Handling device description and resource registration 45
4.5 Tile Module Implementation 45
4.5.1 TileModuleImp 45
4.5.2 Connecting core blackbox to TileLink or AXI interfaces 46
4.5.3 Interrupt Handling 47
4.6 Configuring Core in Chipyard Build Flow 47
4.6.1 Writing Config Fragments to modify TilesLocated. 48
4.6.2 CanAttachTile and TileAttachParams 48
4.7 Protocol Bridging (if Non–TileLink Core) 49

4.7.1 Identifying interface mismatches 49
4.7.2 Inserting diplomatic adapters 49
4.8 Integration Validation 50
4.9 Chapter Summary 51
5 SERV Core 52
5.1 Introduction 52
5.2 SERV Architecture 52
5.3 SERV Instruction Cycle: 78
5.3.1 Instruction Fetch: 78
5.3.2 Instruction Decode: 78
5.3.3 Execution: 79
5.3.3.1 One-stage instructions 80
5.3.3.2 Interrupts and ecall/ebreak 80
5.3.3.3 Two-stage operations 81
5.4 Servile: The Convinience Wrapper 81
5.4.1 Components of SERVILE: 84
5.4.1.1 Multiplexer 84
5.4.1.2 Arbiter 85
5.4.1.3 RF RAM Interface 87
5.4.1.4 SERV Top (CPU): 89
5.5 Serving SoClet 89
5.5.1 Components of The Serving SoClet: 90
5.5.1.1 Serving RAM: 90
5.5.1.2 Servile Memory Interface 91
5.5.1.3 Servile 92
5.5.2 Modifications in Serving 93

5.6 Chapter Summary 95

6 Integration Challenges 96

6.1 Introduction 96

6.2 Language Mismatch 96

6.3 Communication Protocol 96

6.4 Bridge Integration 97

6.5 Chapter Summary 97

7 Protocol Conversion 98

7.1 Introduction 98

7.2 Protocols 98

7.2.1 AXI 98

7.2.2 Wishbone 98

7.3 Conversion Bridge 99

7.3.1 Architecture 99

7.3.2 Finite State Machine (FSM) 99

7.3.3 AXI-Wishbone 105

7.3.3.1 Design 105

7.3.3.2 Verification 105

7.3.4 Wishbone-AXI 106

7.3.4.1 Design 106

7.3.4.2 Verification 106

7.4 System Top 107

7.5 Chapter Summary 110

8 SERV Inside Chipyard 111

8.1 Introduction 111

8.2 Blackboxes 111

8.2.1 Verilog Blackbox 111
8.2.2 Scala Blackbox 112
8.3 SERV Tile 113
8.3.1 Core Params 113
8.3.2 Tile Params 113
8.4 Custom Configurations 114
8.5 Makefile 114
8.6 Build Script 115
8.7 Results 116
8.8 Chpater Summary 118
9 FPGA Implementation 119
9.1 Introduction 119
9.2 Environment 119
9.3 SoC Configuration for FPGA 119
9.4 Bitstream Generation Flow 120
9.5 FPGA Programming and Verification 127
9.6 Results and Observations 128
10 Conclusions and Future Work 129
10.1 Conclusion 129
10.2 Future Work: 130
References 131