Integrating Bit-Serial SERV Core with Chipyard for Scalable, Ultra-Compact RISC-V SoCs

Project Supervisor: Engr. Mehmoona Gul

Co-Supervisor: Dr. Aneesullah

Presented by: Humail Nawaz and Ayesha Qazi

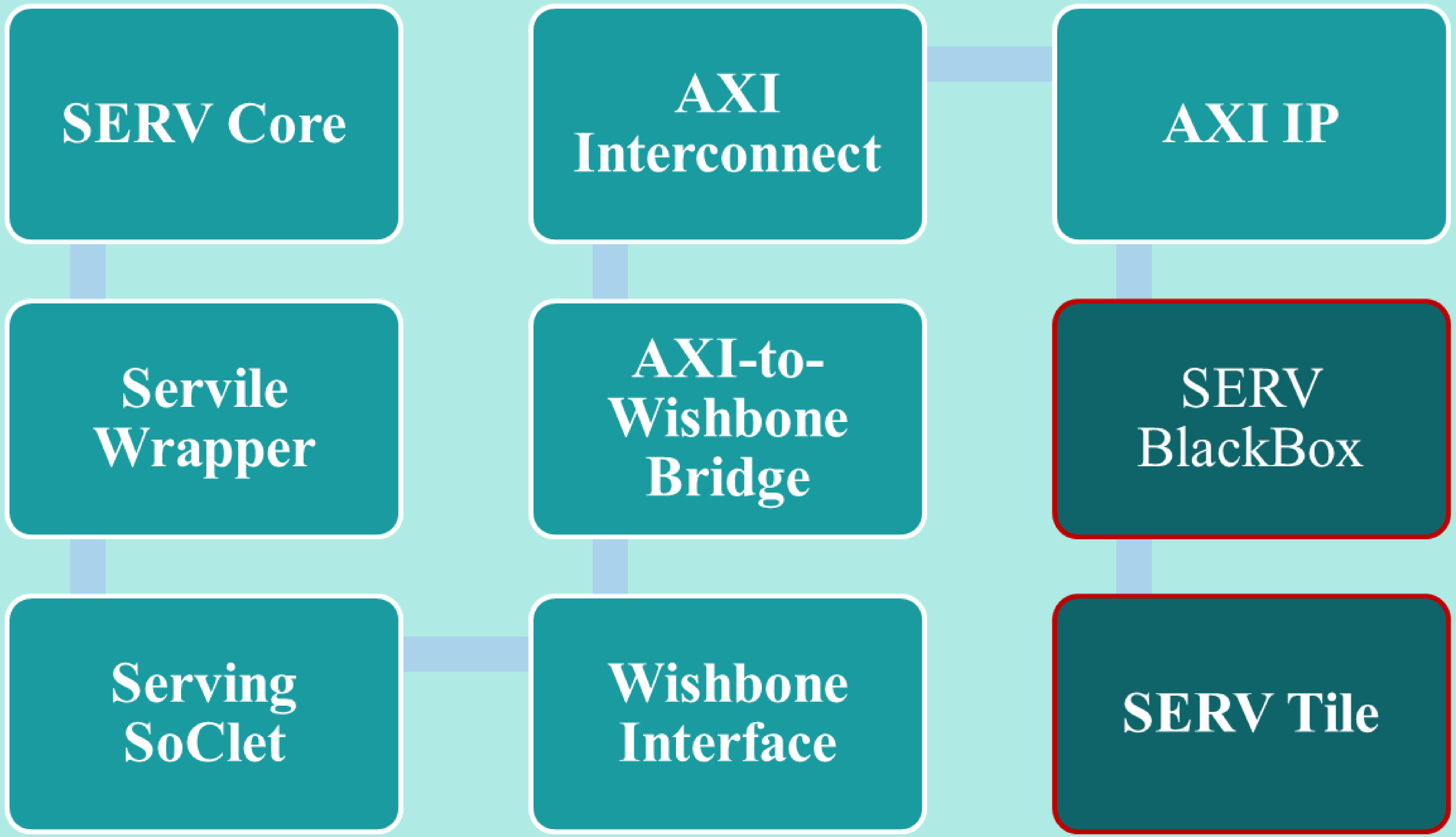
June 30, 2025

CONTENTS

PROBLEM STATEMENT

Chipyard is a powerful SoC design framework, but it lacks compact, low-power cores, limiting its use in minimalist or energy-efficient systems. SERV, a bit-serial RISC-V core and the world’s smallest, offers extreme area and power efficiency but uses the Wishbone protocol, which is incompatible with Chipyard’s TileLink infrastructure. This project bridges that gap by adapting SERV through protocol translation and integration, enabling its use as a functional tile in Chipyard and expanding the framework’s capability to support ultra-lightweight SoC designs.

RECAP


SERV Core
Servile Wrapper
Serving SoClet
AXI Interconnect
AXI-to-Wishbone Bridge
Wishbone Interface
AXI IP
SERV BlackBox
SERV Tile

COMPATIBILITY CHALLENGES

Integrating SERV into Chipyard involved addressing multiple compatibility issues across communication protocols, design languages, and system architecture.

01

Being minimal, SERV does not expose interfaces. Wrappers were need to expose Wishbone signals and enable peripheral connections

02

SERV uses the Wishbone protocol, while Chipyard relies on TileLink. The two protocols are structurally different, requiring a translation bridge.

03

SERV is implemented in Verilog, whereas Chipyard is built in Chisel/Scala. This required using BlackBoxes to connect the Verilog core to the Chisel-based SoC.

SERV

Bit-serial RISC-V CPU core. Executes instructions one per cycle. Communicates via internal control and Wishbone signals.

SERVILE

Wrapper for exposing SERV’s interface. Converts internal signals to standard Wishbone bus, enabling external communication.

SERVING

Modified SoC wrapper around SERV + Servile. Now supports either internal RAM or an external peripheral (the bridge). Includes select logic for routing based on access type.

BRIDGE

Custom 2-way Wishbone ↔ AXI bridge. FSM-based module that successfully translates Wishbone transactions to AXI and vice versa.

ADAPTER

AXI to TileLink adapter. Present as Chipyard’s diplomacy widgets to convert AXI signals into TileLink protocol.

TILE

Custom tile wrapping the whole system. Interfaces with the rest of the SoC using TileLink. Now behaves like a standard core tile in Chipyard.

Protocol Conversion

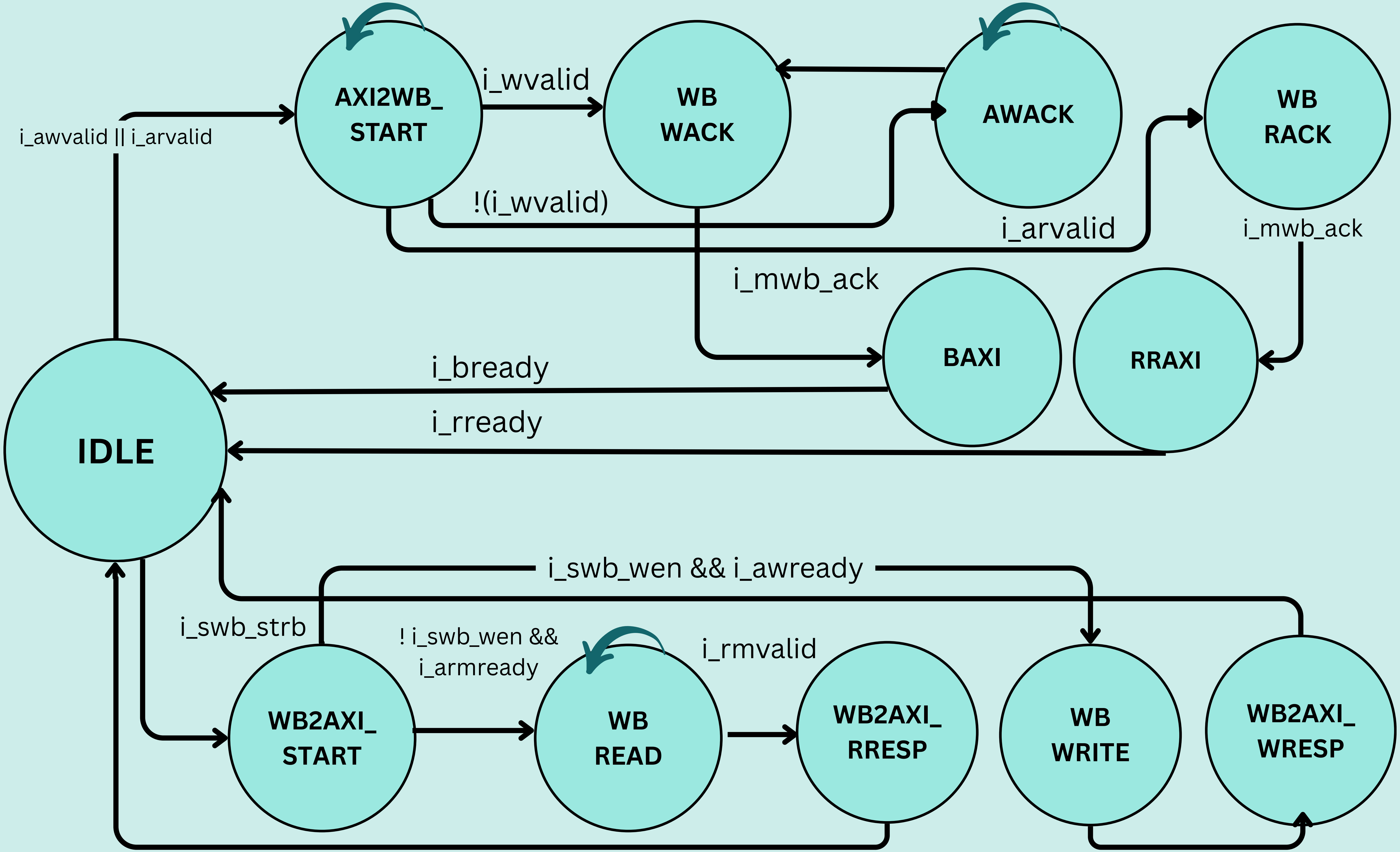
To enable communication between the Wishbone-based SERV core and Chipyard's TileLink ecosystem, a custom bidirectional protocol bridge was developed. This bridge translates between Wishbone and AXI4, and then Chipyard's standard adapters handle conversion between AXI4 and TileLink.

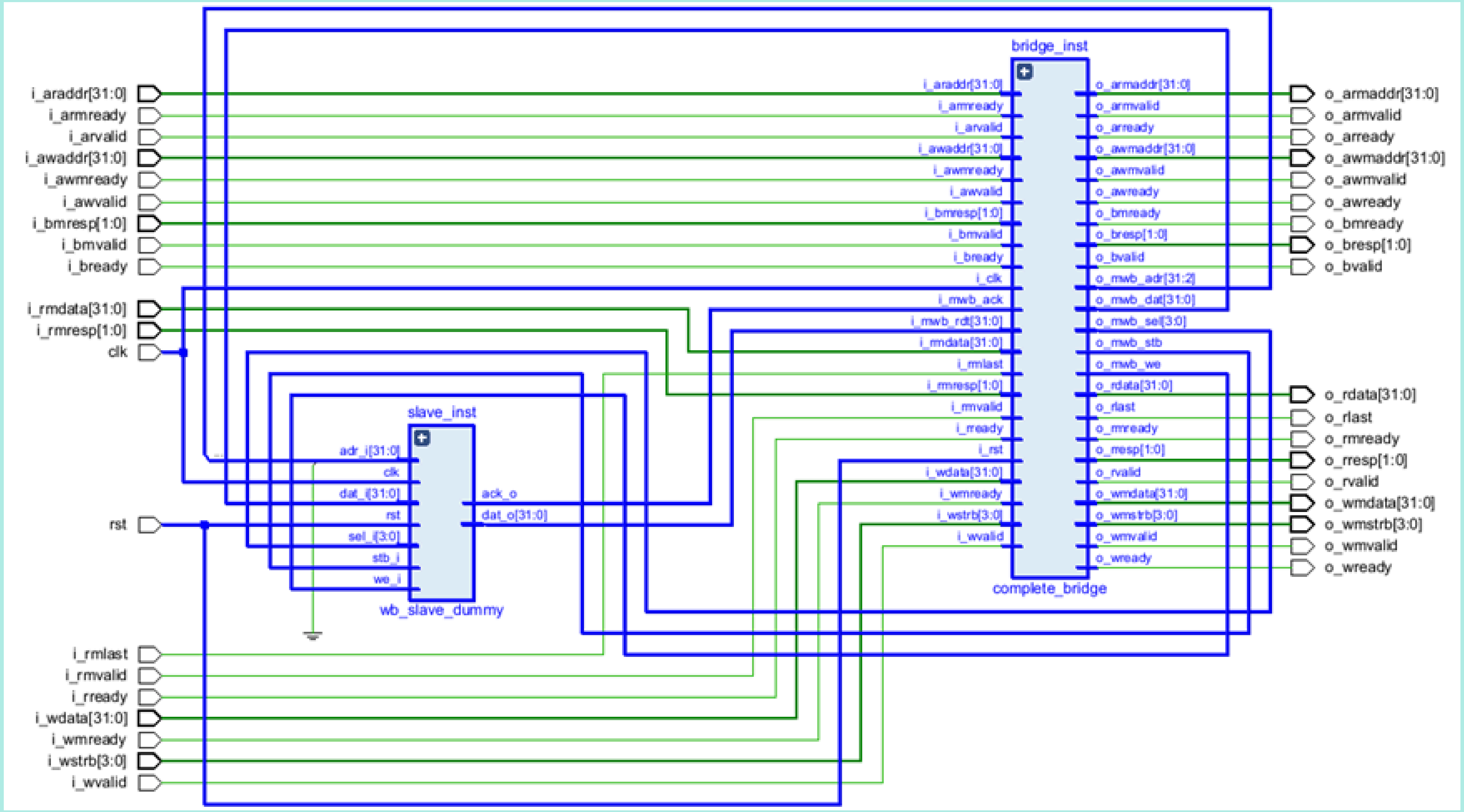
01 The bridge supports both Wishbone-to-AXI and AXI-to-Wishbone communication, enabling reads and writes from either side.

02 The conversion logic is implemented using finite state machines that track handshakes, strobes, and response signals for both protocols.

03 Signals from SERV are translated into AXI read/write transactions. AXI accesses from peripherals or the system bus are converted into Wishbone-compatible signals.

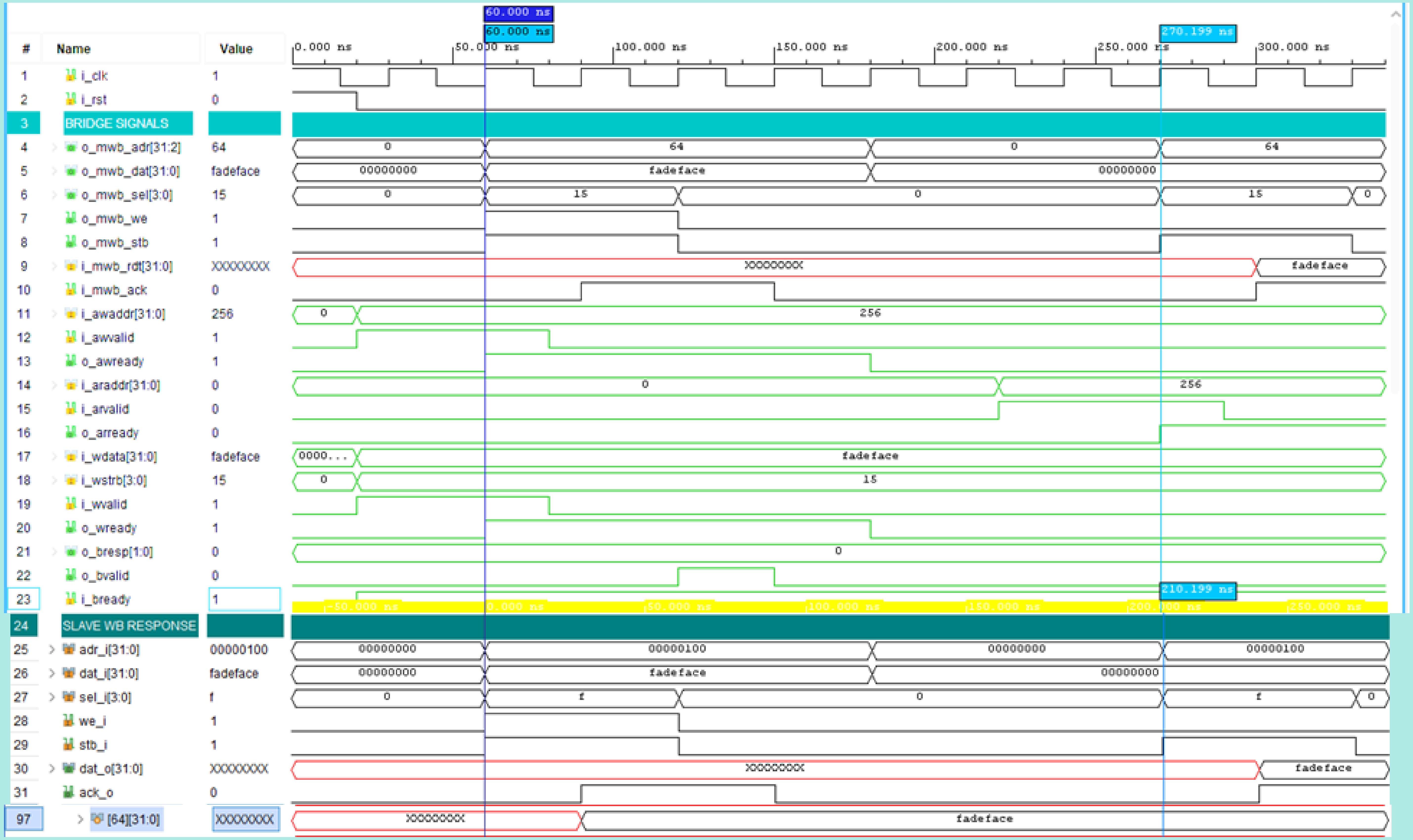
04 Designed to be modular and reusable, the bridge can support other Wishbone-based components with minimal modifications.


i_awvalid || i_arvalid
AXI2WB_ START
i_wvalid
WB WACK
AWACK
WB RACK
!(i_wvalid)
i_arvalid
i_mwb_ack
i_mwb_ack
BAXI
RRAXI
i_bready
i_rready
IDLE
i_swb_wen && i_awready
i_swb_strb
! i_swb_wen && i_armready
i_rmvalid
WB2AXI_ START
WB READ
WB2AXI_ RRESP
WB WRITE
WB2AXI_ WRESP

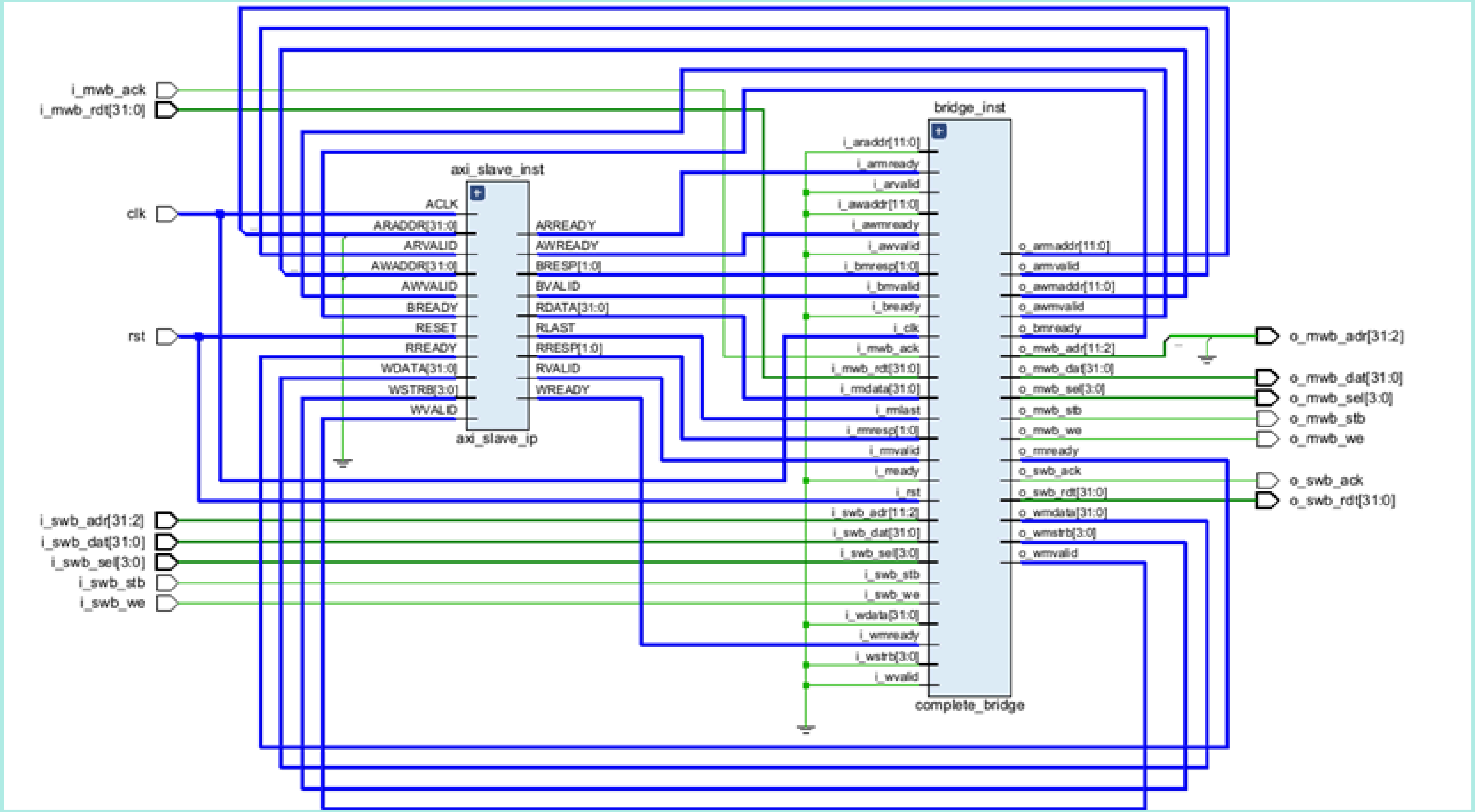

i_araddr[31:0]
i_armready
i_arvalid
i_awaddr[31:0]
i_awmready
i_awvalid
i_bmresp[1:0]
i_bmvalid
i_bready
i_rmdata[31:0]
i_rmresp[1:0]
clk
rst
i_rmlast
i_rmvalid
i_rready
i_wdata[31:0]
i_wmready
i_wstrb[3:0]
i_wvalid
slave_inst
adr_i[31:0]
clk
dat_i[31:0]
rst
sel_i[3:0]
stb_i
we_i
ack_o
dat_o[31:0]
wb_slave_dummy
bridge_inst
i_araddr[31:0]
i_armready
i_arvalid
i_awaddr[31:0]
i_awmready
i_awvalid
i_bmresp[1:0]
i_bmvalid
i_bready
i_clk
i_mwb_ack
i_mwb_rdt[31:0]
i_rmdata[31:0]
i_rmlast
i_rmresp[1:0]
i_rmvalid
i_rready
i_rst
i_wdata[31:0]
i_wmready
i_wstrb[3:0]
i_wvalid
o_armaddr[31:0]
o_armvalid
o_arready
o_awmaddr[31:0]
o_awmvalid
o_awready
o_bmready
o_bresp[1:0]
o_bvalid
o_mwb_adr[31:2]
o_mwb_dat[31:0]
o_mwb_sel[3:0]
o_mwb_stb
o_mwb_we
o_rdata[31:0]
o_rlast
o_rmready
o_rresp[1:0]
o_rvalid
o_wmdata[31:0]
o_wmstrb[3:0]
o_wmvalid
o_wready
complete_bridge
o_armaddr[31:0]
o_armvalid
o_arready
o_awmaddr[31:0]
o_awmvalid
o_awready
o_bmready
o_bresp[1:0]
o_bvalid
o_rdata[31:0]
o_rlast
o_rmready
o_rresp[1:0]
o_rvalid
o_wmdata[31:0]
o_wmstrb[3:0]
o_wmvalid
o_wready

AXI - WB

The AXI-to-Wishbone side of the bridge was verified by issuing AXI read and write transactions and observing correct translation into Wishbone signals.

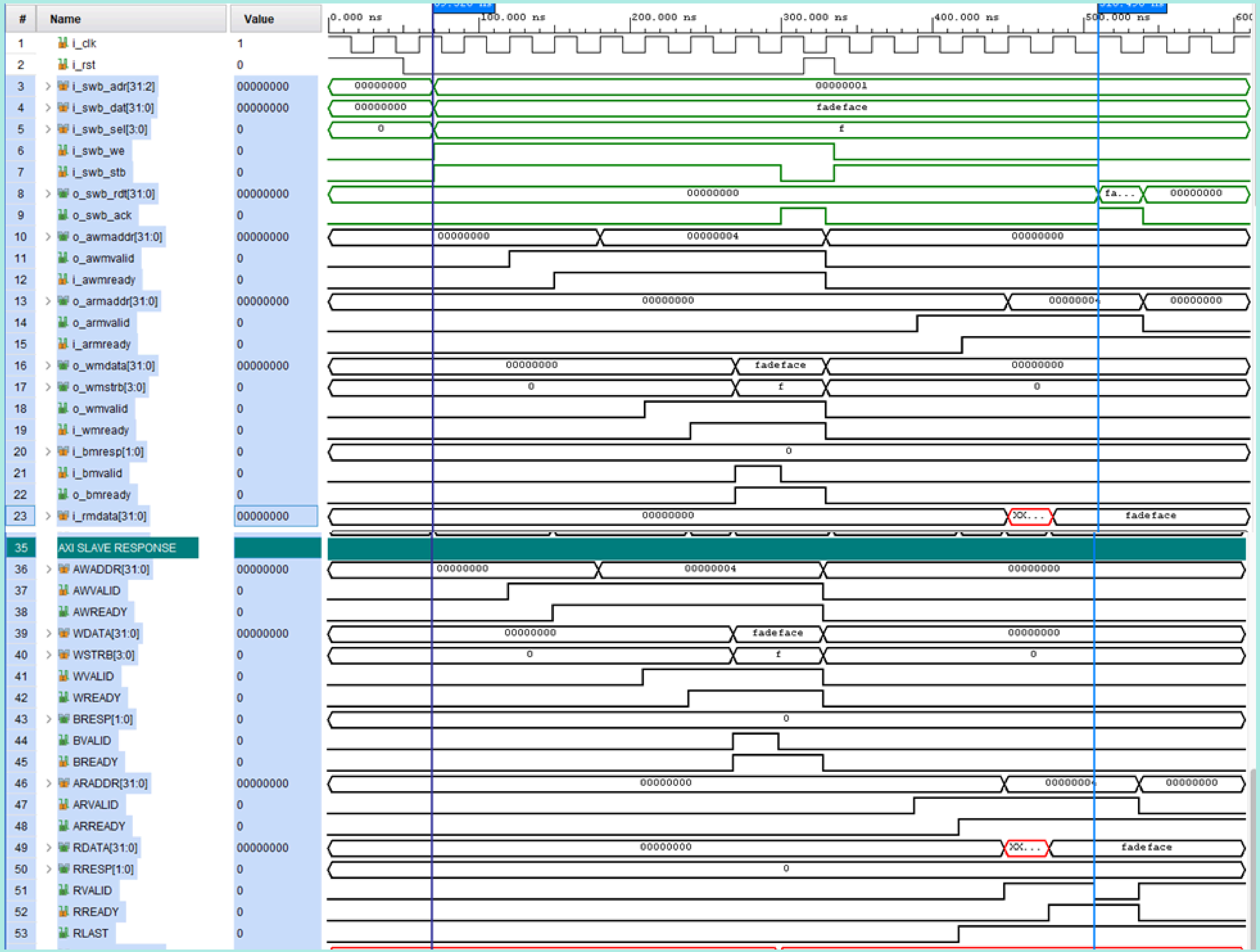


#	Name	Value
1	i_clk	1
2	i_rst	0
3	BRIDGE SIGNALS	
4	o_mwb_adr[31:2]	64
5	o_mwb_dat[31:0]	fadeface
6	o_mwb_sel[3:0]	15
7	o_mwb_we	1
8	o_mwb_stb	1
9	i_mwb_rdt[31:0]	XXXXXXXX
10	i_mwb_ack	0
11	i_awaddr[31:0]	256
12	i_awvalid	1
13	o_awready	1
14	i_araddr[31:0]	0
15	i_arvalid	0
16	o_arready	0
17	i_wdata[31:0]	fadeface
18	i_wstrb[3:0]	15
19	i_wvalid	1
20	o_wready	1
21	o_bresp[1:0]	0
22	o_bvalid	0
23	i_bready	1
24	SLAVE WB RESPONSE	
25	adr_i[31:0]	00000100
26	dat_i[31:0]	fadeface
27	sel_i[3:0]	f
28	we_i	1
29	stb_i	1
30	dat_o[31:0]	XXXXXXXX
31	ack_o	0
97	[64][31:0]	XXXXXXXX


i_mwb_ack
i_mwb_rdt[31:0]
clk
rst
i_swb_adr[31:2]
i_swb_dat[31:0]
i_swb_sel[3:0]
i_swb_stb
i_swb_we
axi_slave_inst
ACLK
ARADDR[31:0]
ARVALID
AWADDR[31:0]
AWVALID
BREADY
RESET
RREADY
WDATA[31:0]
WSTRB[3:0]
WVALID
ARREADY
AWREADY
BRESP[1:0]
BVALID
RDATA[31:0]
RLAST
RRESP[1:0]
RVALID
WREADY
axi_slave_ip
bridge_inst
i_araddr[11:0]
i_armready
i_arvalid
i_awaddr[11:0]
i_awmready
i_awvalid
i_bmresp[1:0]
i_bmvalid
i_bready
i_clk
i_mwb_ack
i_mwb_rdt[31:0]
i_rmdata[31:0]
i_rmlast
i_rmresp[1:0]
i_rmvalid
i_rready
i_rst
i_swb_adr[11:2]
i_swb_dat[31:0]
i_swb_sel[3:0]
i_swb_stb
i_swb_we
i_wdata[31:0]
i_wmready
i_wstrb[3:0]
i_wvalid
o_armaddr[11:0]
o_armvalid
o_awmaddr[11:0]
o_awmvalid
o_bmready
o_mwb_adr[11:2]
o_mwb_dat[31:0]
o_mwb_sel[3:0]
o_mwb_stb
o_mwb_we
o_rmready
o_swb_ack
o_swb_rdt[31:0]
o_wmdata[31:0]
o_wmstrb[3:0]
o_wmvalid
complete_bridge
o_mwb_adr[31:2]
o_mwb_dat[31:0]
o_mwb_sel[3:0]
o_mwb_stb
o_mwb_we
o_swb_ack
o_swb_rdt[31:0]

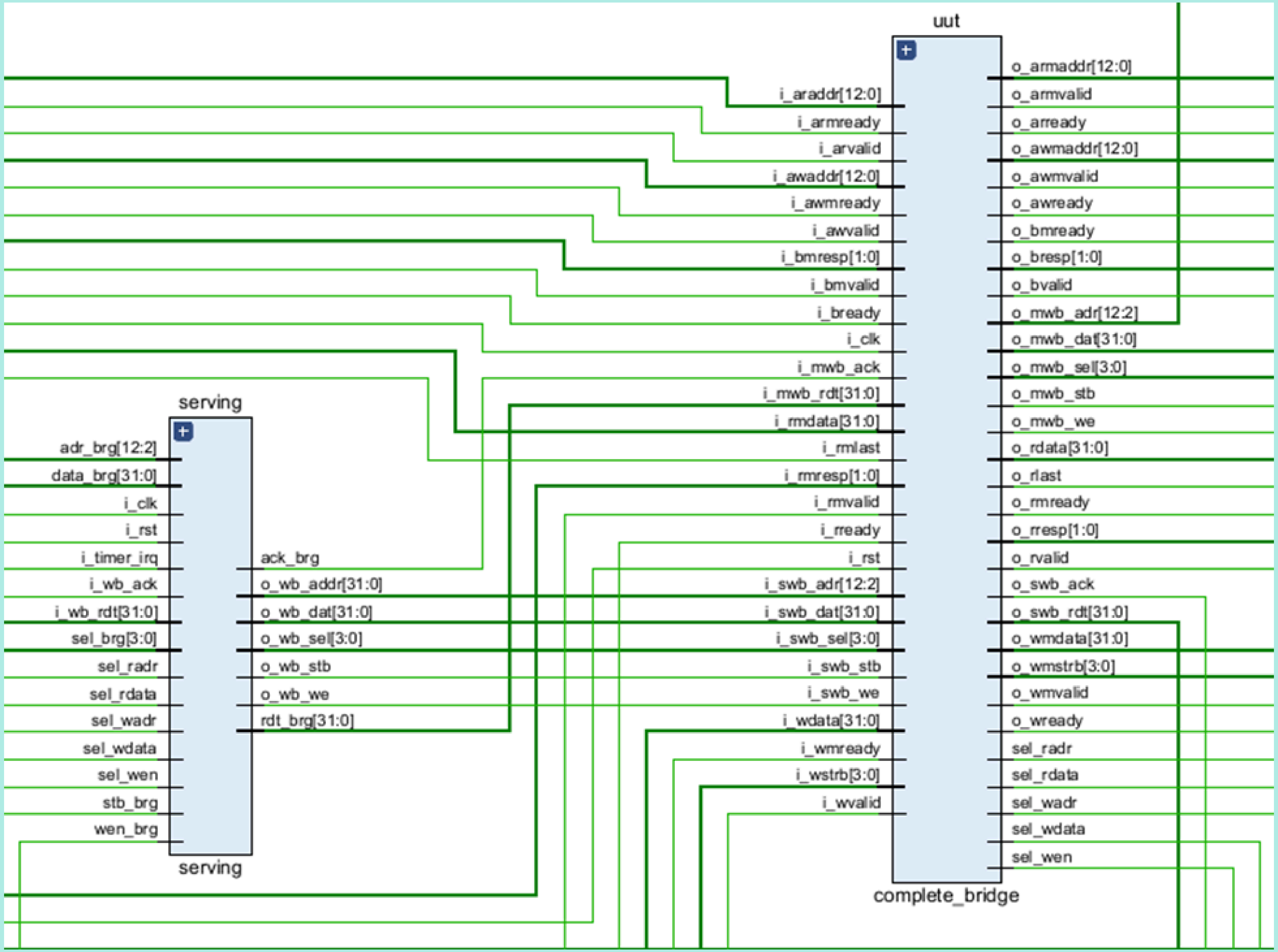
WB - AXI

Write verification from Wishbone to AXI was limited due to SERV design constraints, not the bridge itself; read operations and signal behavior confirm correct bridge functionality.



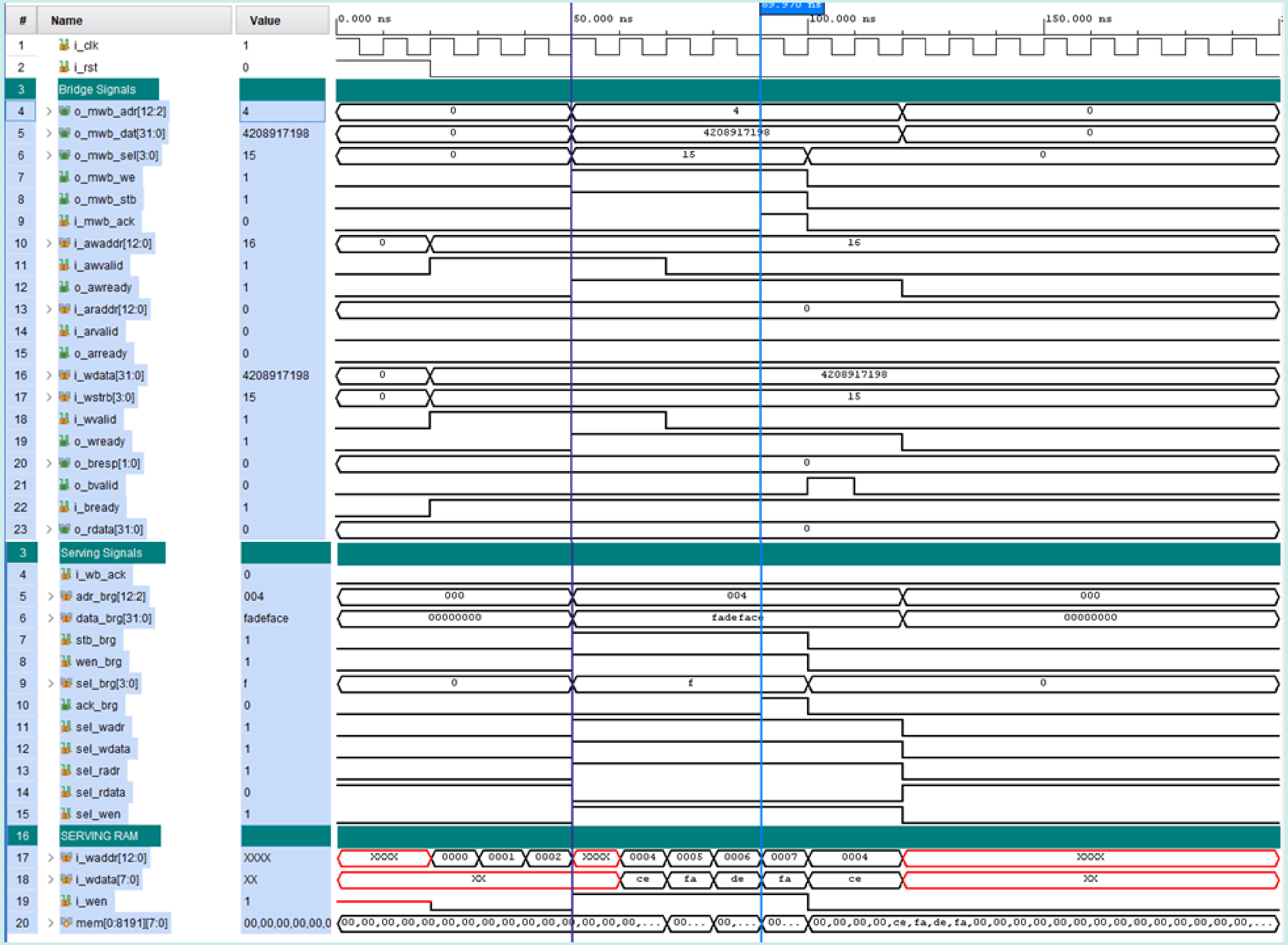
BRIDGE CONNECTED TO SERVING

This diagram illustrates the bidirectional communication path between the bridge and the Serving. The bridge directly interfaces with the SERV core's Wishbone bus, enabling seamless AXI-to-Wishbone transactions for both read and write operations.



AXI-SERVING WRITE

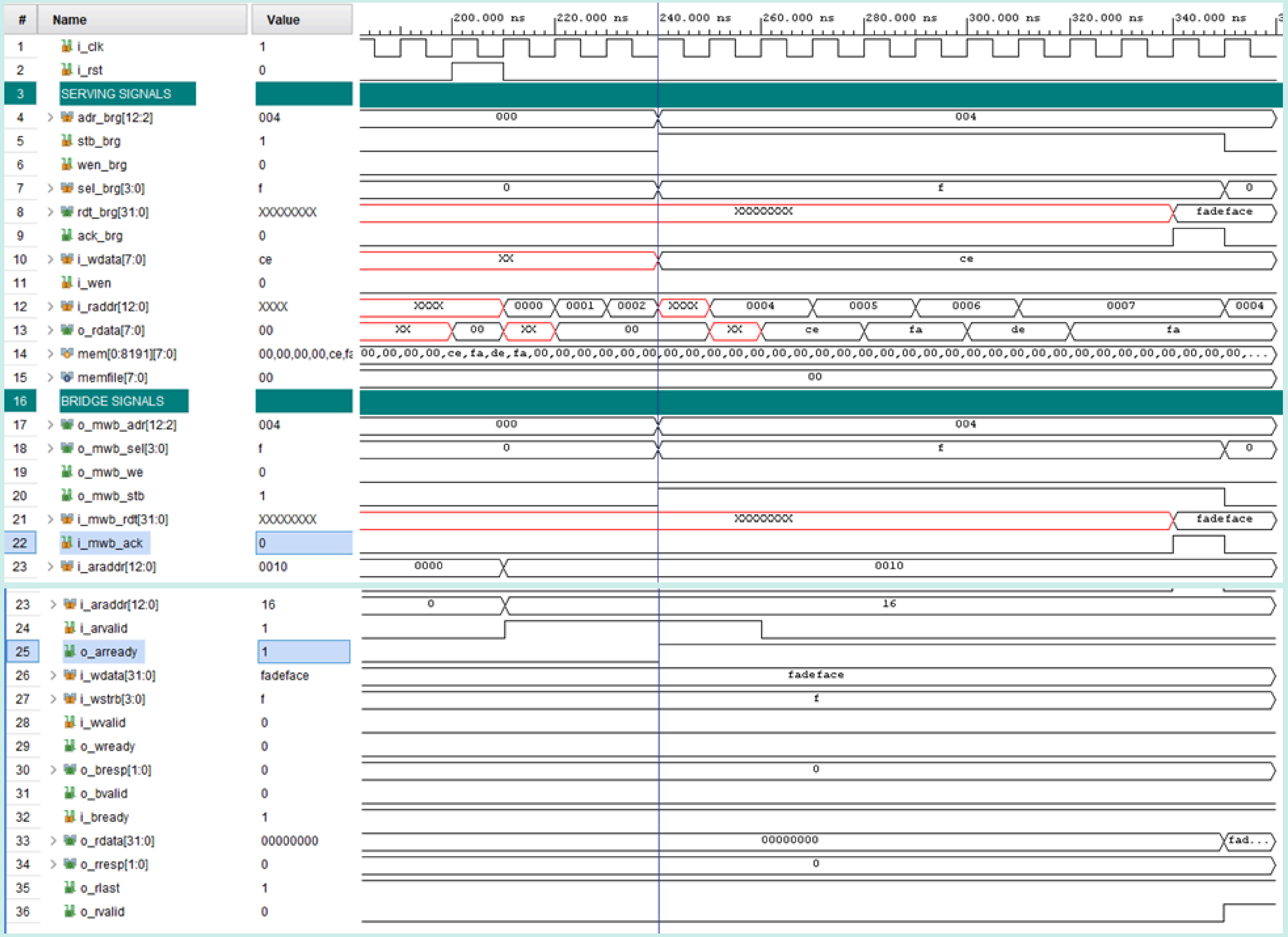
Write Transaction:
This diagram illustrates a successful AXI write sequence routed through the AXI2Serving bridge. It shows the address (AW channel) and data (W channel) phases, followed by the write response (B channel), indicating a properly acknowledged write to the SERV core.



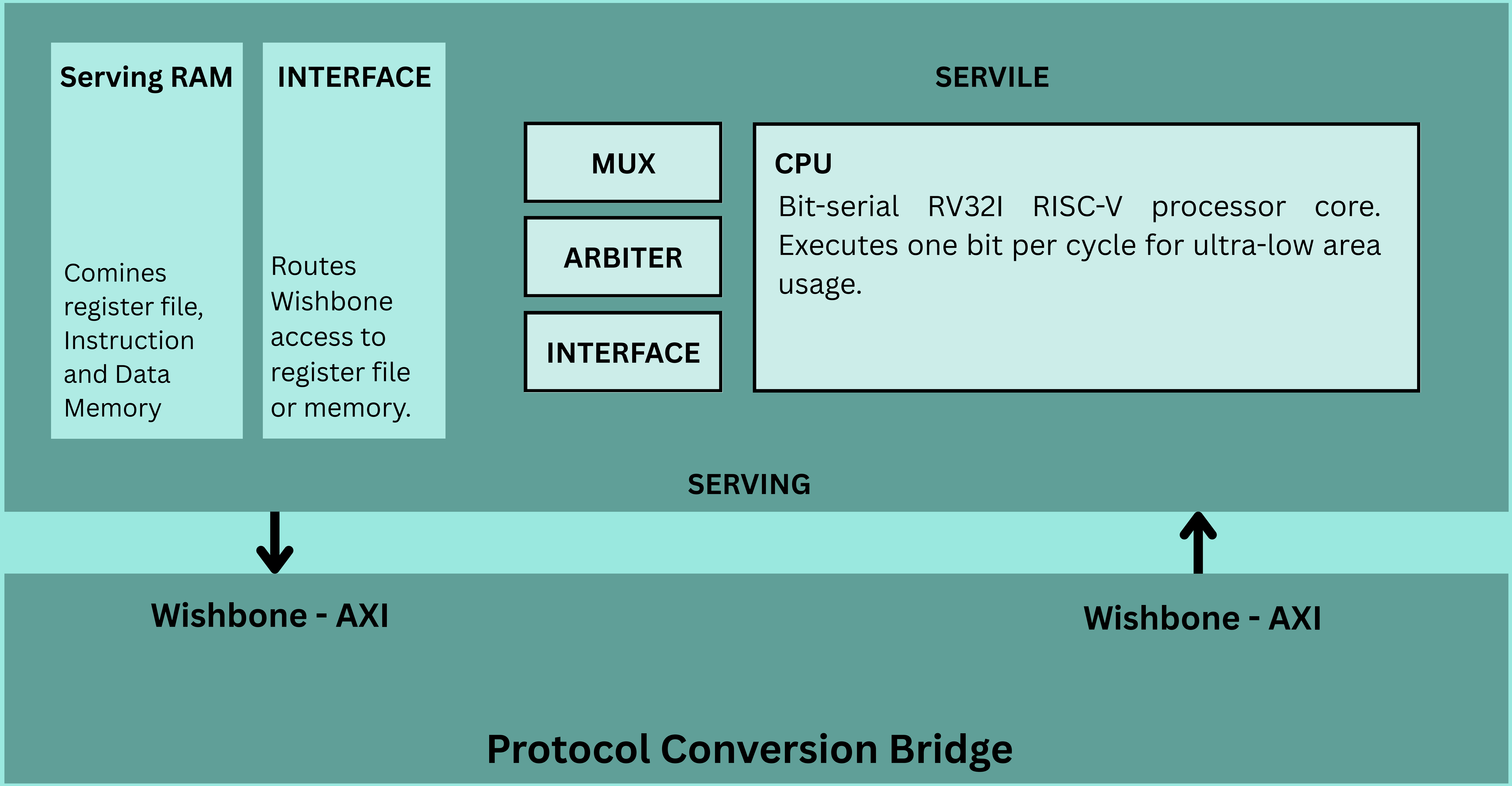
#	Name	Value
1	i_clk	1
2	i_rst	0
3	Bridge Signals	
4	o_mwb_adr[12:2]	4
5	o_mwb_dat[31:0]	4208917198
6	o_mwb_sel[3:0]	15
7	o_mwb_we	1
8	o_mwb_stb	1
9	i_mwb_ack	0
10	i_awaddr[12:0]	16
11	i_awvalid	1
12	o_awready	1
13	i_araddr[12:0]	0
14	i_arvalid	0
15	o_arready	0
16	i_wdata[31:0]	4208917198
17	i_wstrb[3:0]	15
18	i_wvalid	1
19	o_wready	1
20	o_bresp[1:0]	0
21	o_bvalid	0
22	i_bready	1
23	o_rdata[31:0]	0
3	Serving Signals	
4	i_wb_ack	0
5	adr_brg[12:2]	004
6	data_brg[31:0]	fadeface
7	stb_brg	1
8	wen_brg	1
9	sel_brg[3:0]	f
10	ack_brg	0
11	sel_wadr	1
12	sel_wdata	1
13	sel_radr	1
14	sel_rdata	0
15	sel_wen	1
16	SERVING RAM	
17	i_waddr[12:0]	XXXX
18	i_wdata[7:0]	XX
19	i_wen	1
20	mem[0:8191][7:0]	00,00,00,00,00,0

AXI-SERVING READ

Read Transaction:
This waveform demonstrates a successful AXI read operation to the SERV core via the AXI2Serving bridge. It highlights the address handshake (AR channel), the valid data response (R channel), and the coordination between AXI signals and SERV's Wishbone interface.



#	Name	Value
1	i_clk	1
2	i_rst	0
3	SERVING SIGNALS	
4	adr_brg[12:2]	004
5	stb_brg	1
6	wen_brg	0
7	sel_brg[3:0]	f
8	rdt_brg[31:0]	XXXXXXXX
9	ack_brg	0
10	i_wdata[7:0]	ce
11	i_wen	0
12	i_raddr[12:0]	XXXX
13	o_rdata[7:0]	00
14	mem[0:8191][7:0]	00,00,00,00,ce,fa
15	memfile[7:0]	00
16	BRIDGE SIGNALS	
17	o_mwb_adr[12:2]	004
18	o_mwb_sel[3:0]	f
19	o_mwb_we	0
20	o_mwb_stb	1
21	i_mwb_rdt[31:0]	XXXXXXXX
22	i_mwb_ack	0
23	i_araddr[12:0]	0010
23	i_araddr[12:0]	16
24	i_arvalid	1
25	o_arready	1
26	i_wdata[31:0]	fadeface
27	i_wstrb[3:0]	f
28	i_wvalid	0
29	o_wready	0
30	o_bresp[1:0]	0
31	o_bvalid	0
32	i_bready	1
33	o_rdata[31:0]	00000000
34	o_rresp[1:0]	0
35	o_rlast	1
36	o_rvalid	0


Serving RAM
Comines register file, Instruction and Data Memory
INTERFACE
Routes Wishbone access to register file or memory.
SERVILE
MUX
ARBITER
INTERFACE
CPU
Bit-serial RV32I RISC-V processor core. Executes one bit per cycle for ultra-low area usage.
SERVING
Wishbone - AXI
Wishbone - AXI
Protocol Conversion Bridge

INTEGRATION PROCESS

01

BlackBox Wrapping

The complete SERV+Bridge design was wrapped in a Verilog BlackBox, allowing Chipyard’s Chisel/Scala flow to reference the external Verilog module.

02

Defining the Core & Tile

Created ServTile.scala, following Chipyard's structure. This defines the core's interface and its encapsulation in a tile.

03

AXI-TileLink Conversion

Used Chipyard’s AXI4ToTL and TLToAXI4 adapters to connect the AXI interface of the bridge to the TileLink system bus.

04

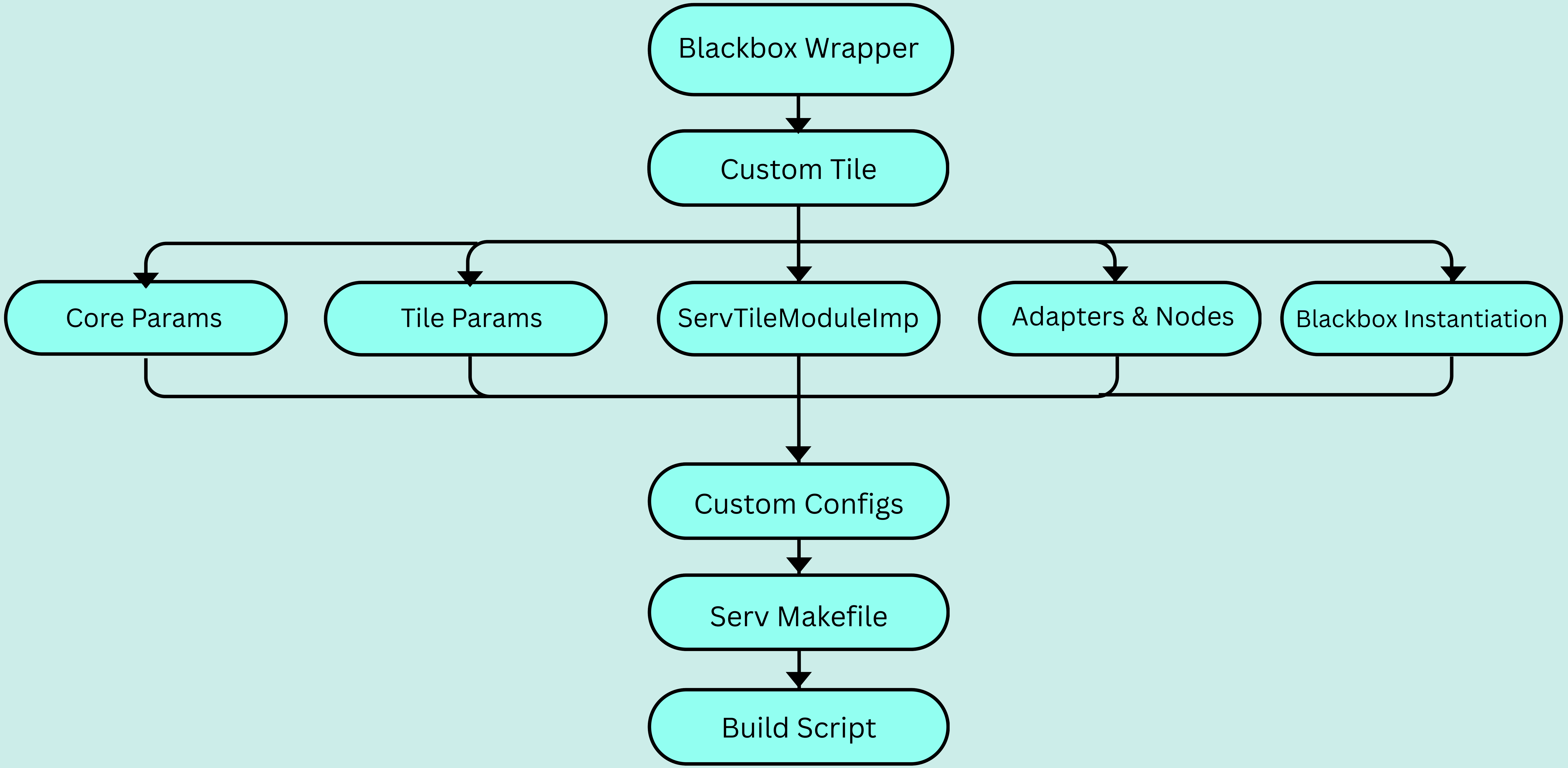
Parameter & Config Setup

Added a ServConfigs and ConfigMixins to make the tile selectable in Chipyard’s build system.

05

Build Hookup

Linked the tile, configs, & blackbox in build.sbt and Makefiles to enable successful build with make CONFIG=ServConfig.


Blackbox Wrapper
Custom Tile
Core Params
Tile Params
ServTileModuleImp
Adapters & Nodes
Blackbox Instantiation
Custom Configs
Serv Makefile
Build Script

BlackBoxes

01	BlackBox Wrapper in Scala	A BlackBox class was created in ServBlackBox.scala, mapping ports to the Verilog module and specifying the file.
02	Verilog Source Linking	All Verilog files (serving.v, servile.v, bridge files, etc.) were placed under vsrc/ and referenced.
03	Build Configuration	build.sbt was updated to include these Verilog files so they're compiled and passed correctly during elaboration and Verilator simulation.
04	Result	The entire SERV+Bridge unit was successfully treated as a Verilog core within a Chisel-based SoC design.

SERV TILE

CORE CONFIGURATION

SERV is a minimal 32-bit RISC-V core optimized for area and power. It disables complex features like debug, FPU, and atomics. Its parameters include memory size, bit-width, and CSR support, focusing on ultra-lightweight embedded use.

TILE SETUP

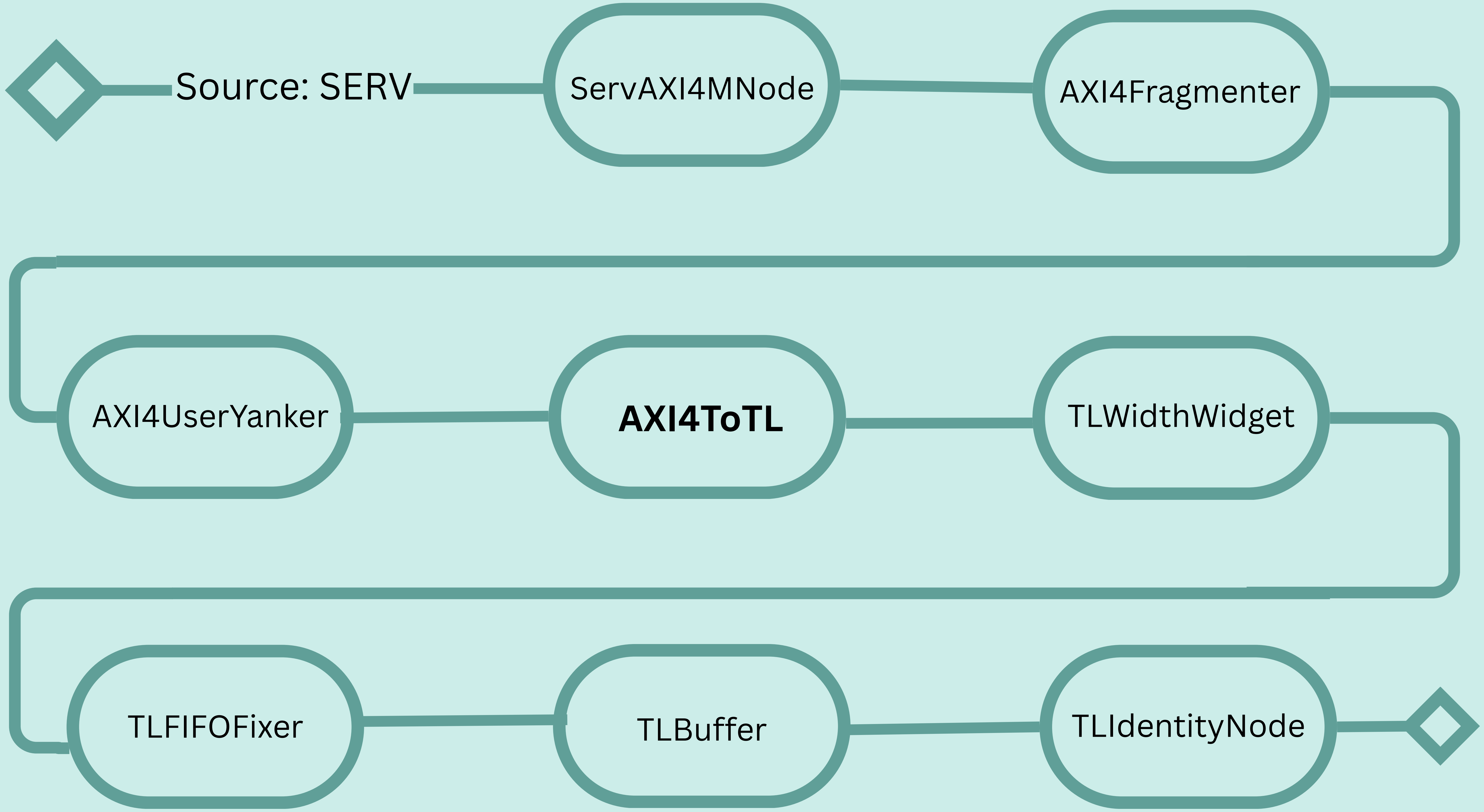
The SERV tile sets the core's position in the system, disabling caches and choosing simple clock crossings. It includes basic identifiers and resource declarations to integrate cleanly into RocketChip.

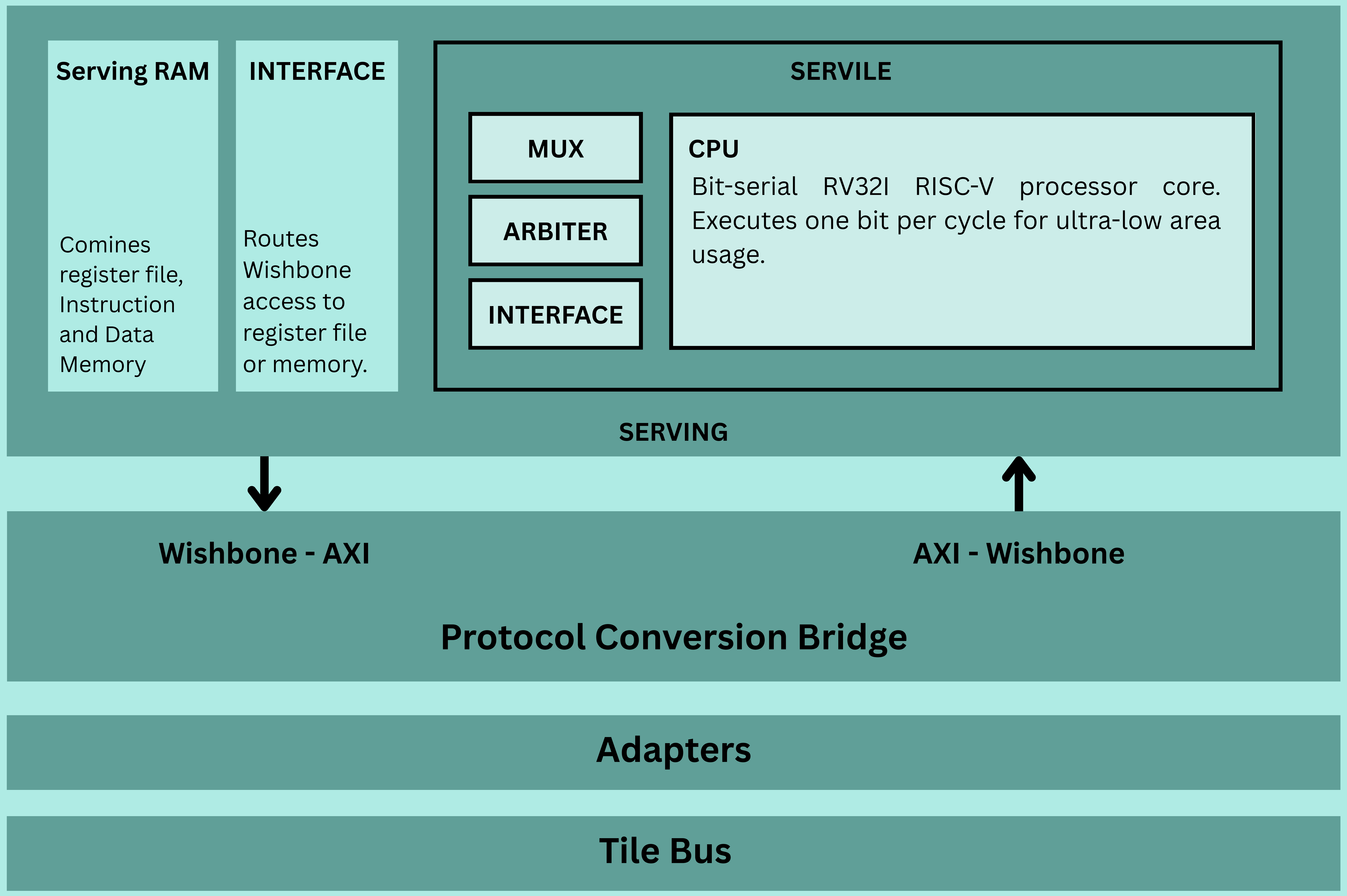
CORE INTEGRATION

ServTile instantiates the core, connects clock/reset/IRQs, and wraps it for TileLink compatibility. It uses ServTileModuleImp to handle interfacing and sets up device entries for system-level recognition.

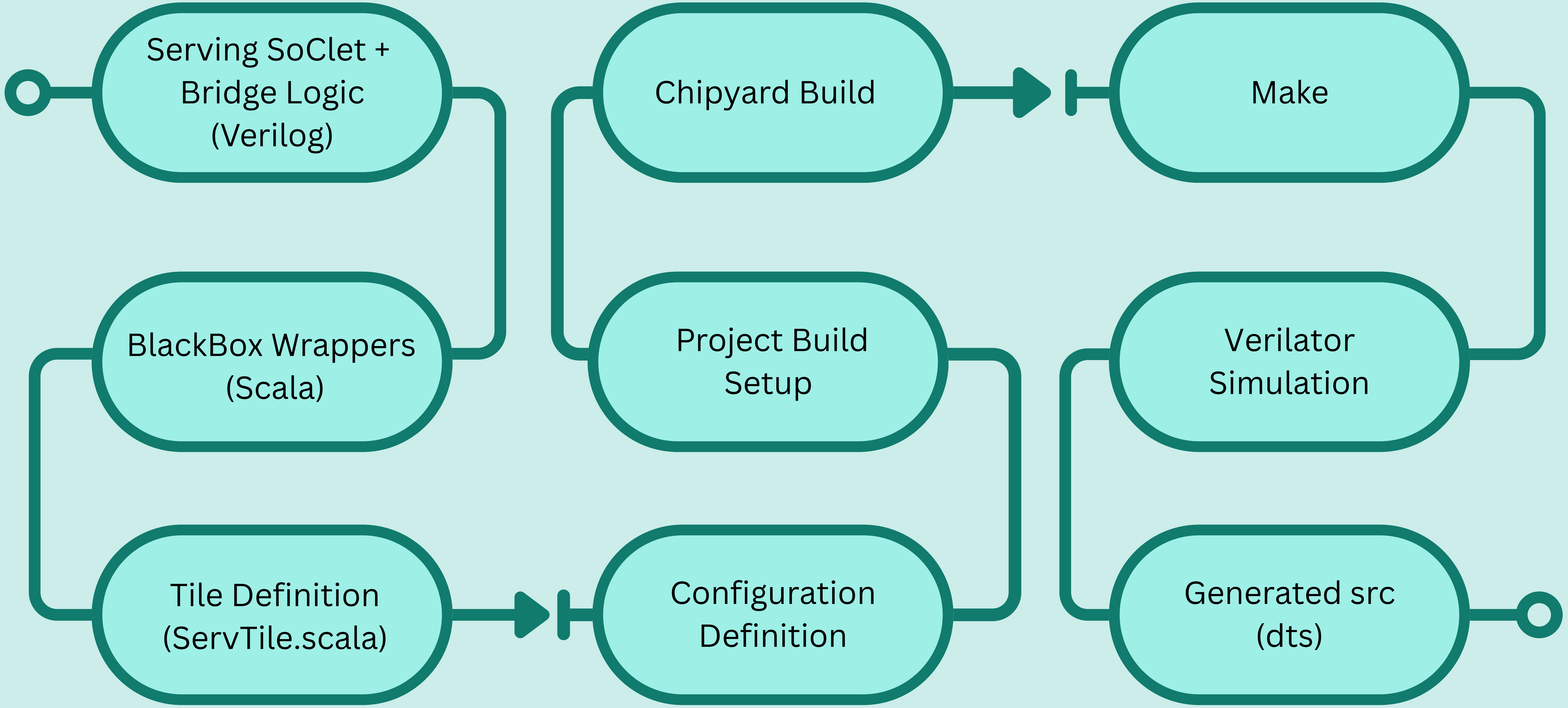
INTERCONNECTS

SERV uses an AXI4 master interface, converted to TileLink via adapters like AXI4ToTL. Though a slave port exists, it's unused. This setup lets the simple SERV core work within the full Chipyard fabric.


Source: SERV
ServAXI4MNode
AXI4Fragmenter
AXI4UserYanker
AXI4ToTL
TLWidthWidget
TLFIFOFixer
TLBuffer
TLIdentityNode

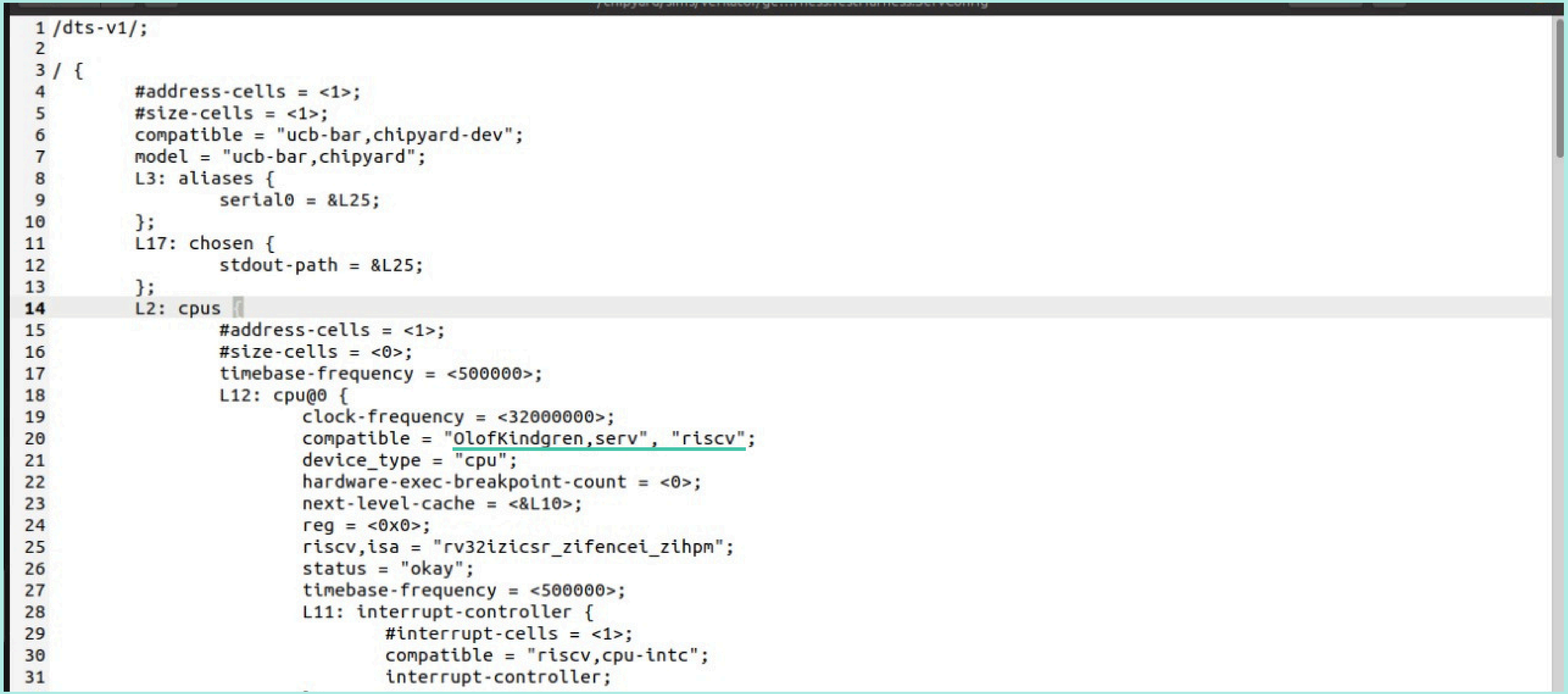

Serving RAM
Comines register file, Instruction and Data Memory
INTERFACE
Routes Wishbone access to register file or memory.
SERVILE
MUX
ARBITER
INTERFACE
CPU
Bit-serial RV32I RISC-V processor core. Executes one bit per cycle for ultra-low area usage.
SERVING
Wishbone - AXI
AXI - Wishbone
Protocol Conversion Bridge
Adapters
Tile Bus

BUILD FLOW


Serving SoClet + Bridge Logic (Verilog)
BlackBox Wrappers (Scala)
Tile Definition (ServTile.scala)
Configuration Definition
Project Build Setup
Chipyard Build
Make
Verilator Simulation
Generated src (dts)

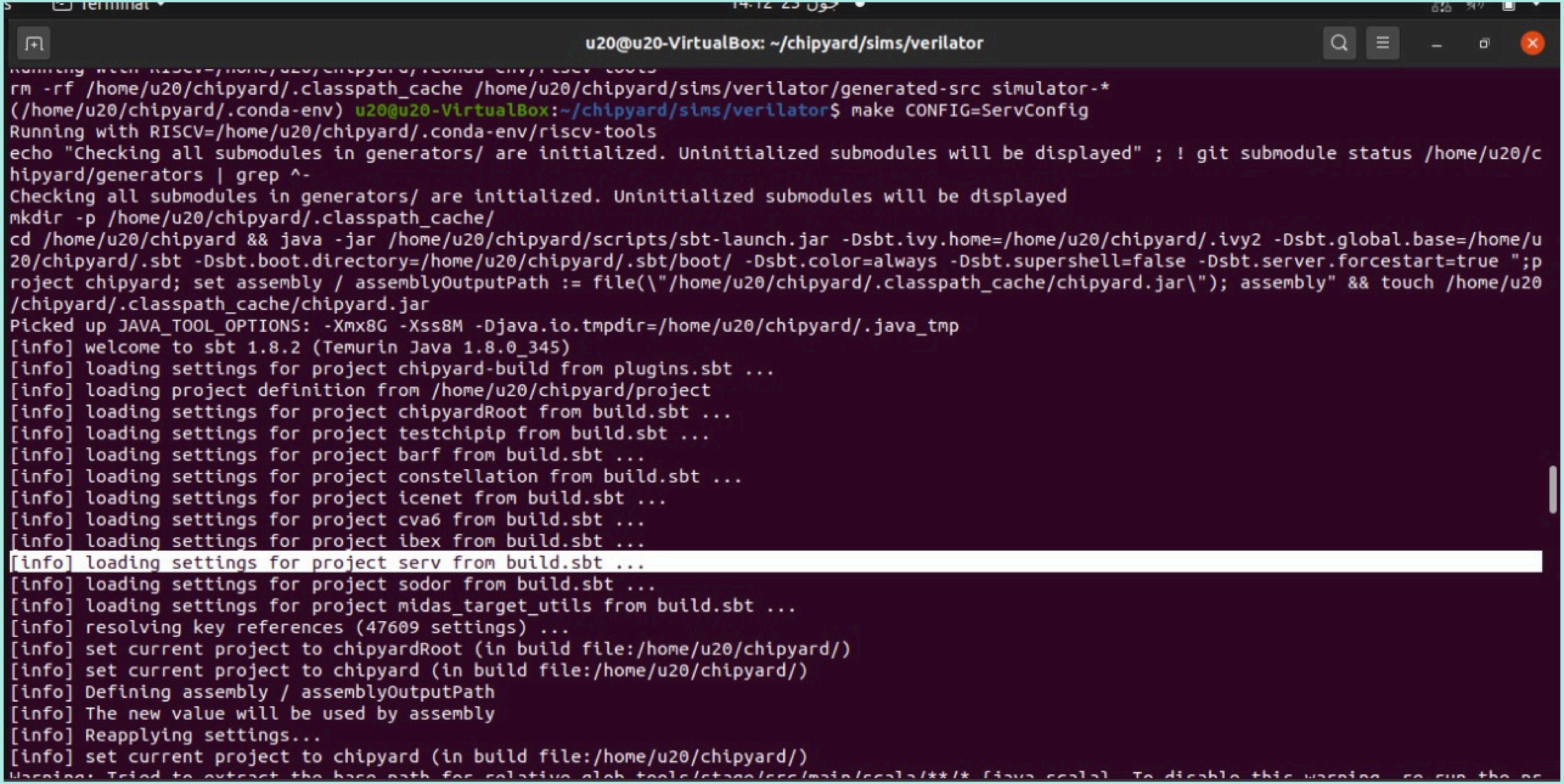
RESULTS AND ACHIEVEMENTS

SERV Device Tree inside Chipyard:



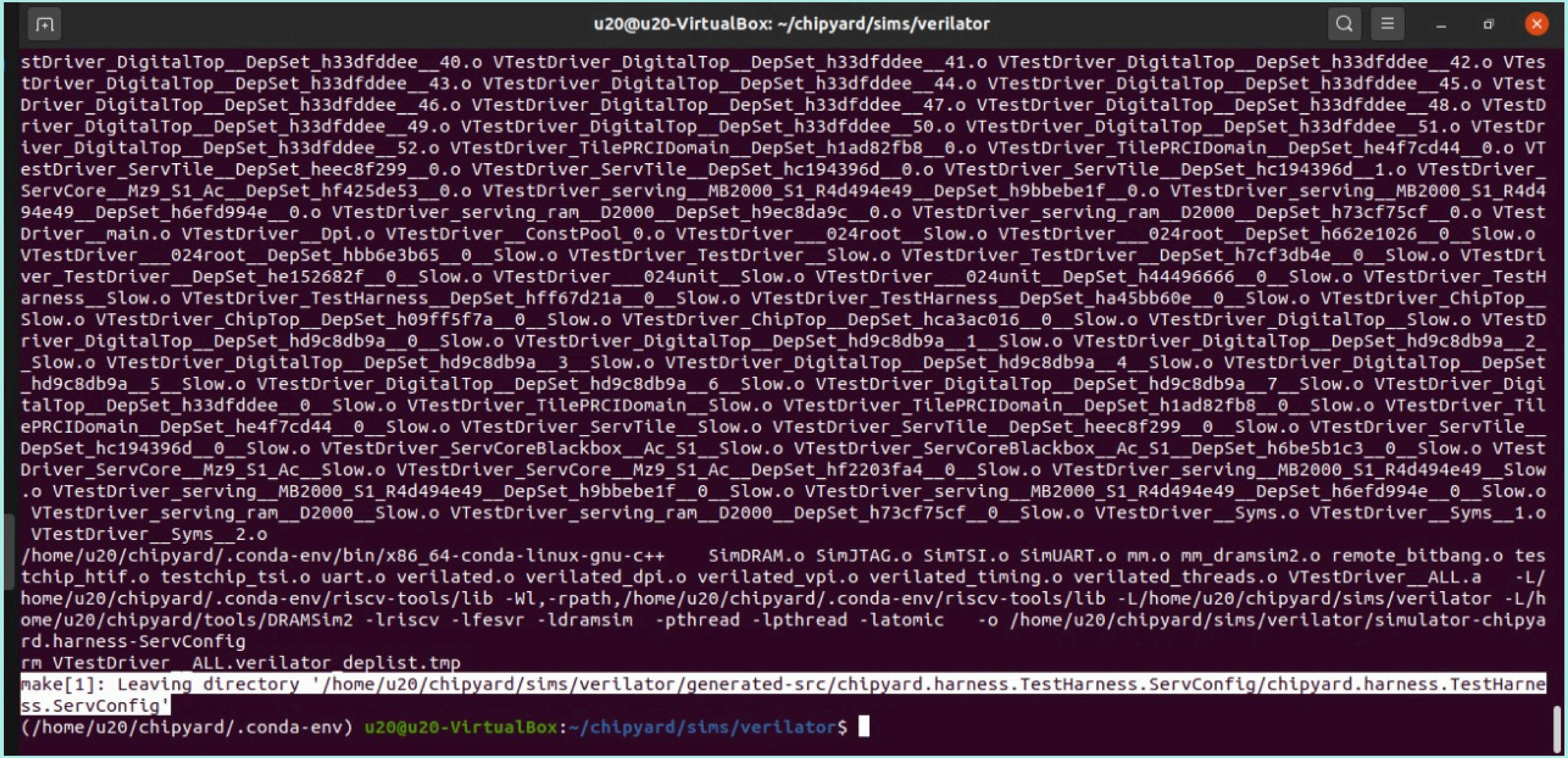
```
/dts-v1/;

/ {
        #address-cells = <1>;
        #size-cells = <1>;
        compatible = "ucb-bar,chipyard-dev";
        model = "ucb-bar,chipyard";
        L3: aliases {
                serial0 = &L25;
        };
        L17: chosen {
                stdout-path = &L25;
        };
        L2: cpus {
                #address-cells = <1>;
                #size-cells = <0>;
                timebase-frequency = <500000>;
                L12: cpu@0 {
                        clock-frequency = <32000000>;
                        compatible = "OlofKindgren,serv", "riscv";
                        device_type = "cpu";
                        hardware-exec-breakpoint-count = <0>;
                        next-level-cache = <&L10>;
                        reg = <0x0>;
                        riscv,isa = "rv32izicsr_zifencei_zihpm";
                        status = "okay";
                        timebase-frequency = <500000>;
                        L11: interrupt-controller {
                                #interrupt-cells = <1>;
                                compatible = "riscv,cpu-intc";
                                interrupt-controller;
```

SERV Core Successfully Registered in Chipyard's Core List


u20@u20-VirtualBox: ~/chipyard/sims/verilator

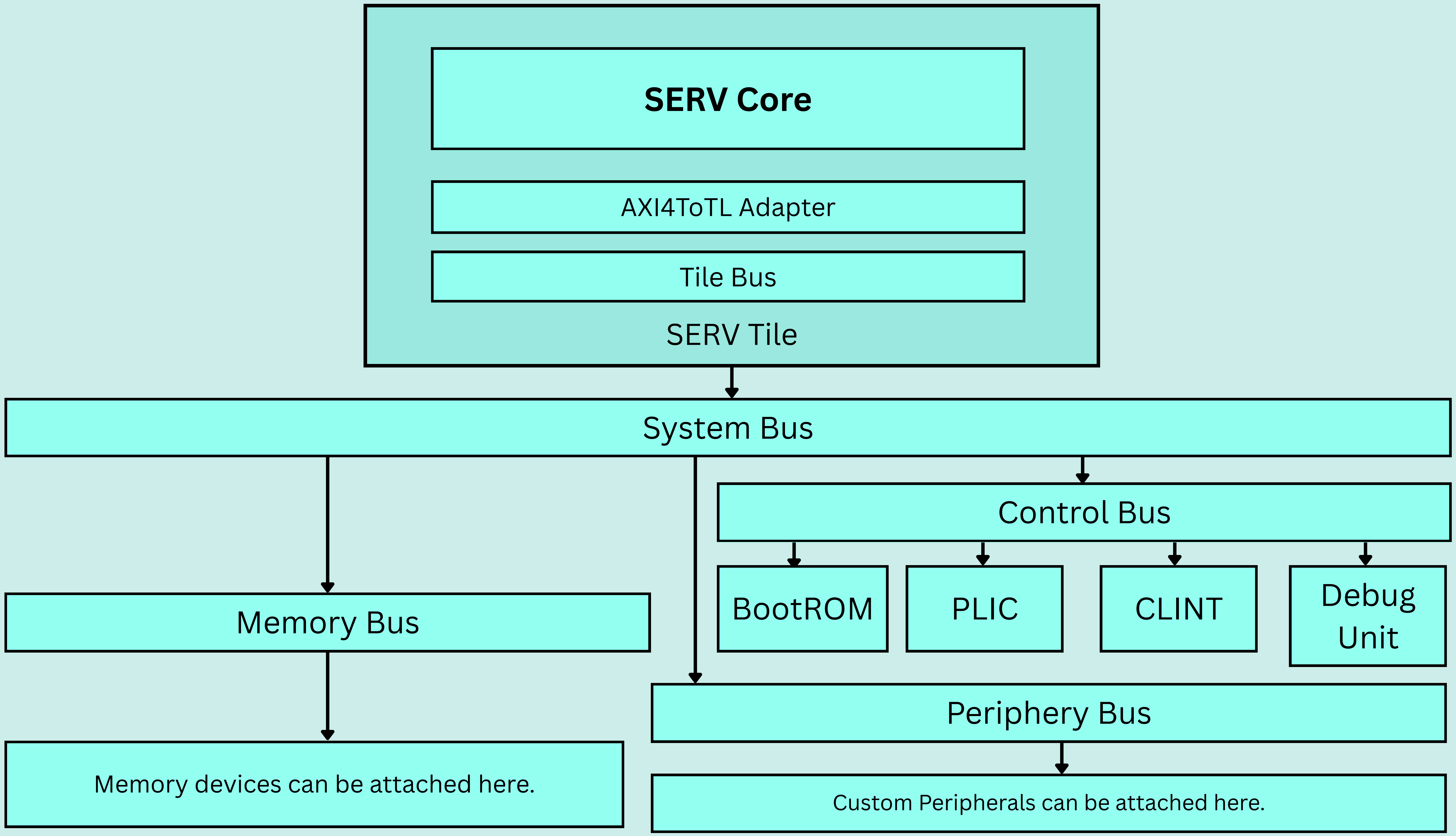
```
rm -rf /home/u20/chipyard/.classpath_cache /home/u20/chipyard/sims/verilator/generated-src simulator-*
(/home/u20/chipyard/.conda-env) u20@u20-VirtualBox:~/chipyard/sims/verilator$ make CONFIG=ServConfig
Running with RISCV=/home/u20/chipyard/.conda-env/riscv-tools
echo "Checking all submodules in generators/ are initialized. Uninitialized submodules will be displayed" ; ! git submodule status /home/u20/c
hipyard/generators | grep ^-
Checking all submodules in generators/ are initialized. Uninitialized submodules will be displayed
mkdir -p /home/u20/chipyard/.classpath_cache/
cd /home/u20/chipyard && java -jar /home/u20/chipyard/scripts/sbt-launch.jar -Dsbt.ivy.home=/home/u20/chipyard/.ivy2 -Dsbt.global.base=/home/u
20/chipyard/.sbt -Dsbt.boot.directory=/home/u20/chipyard/.sbt/boot/ -Dsbt.color=always -Dsbt.supershell=false -Dsbt.server.forcestart=true ";p
roject chipyard; set assembly / assemblyOutputPath := file(\"/home/u20/chipyard/.classpath_cache/chipyard.jar\"); assembly" && touch /home/u20
/chipyard/.classpath_cache/chipyard.jar
Picked up JAVA_TOOL_OPTIONS: -Xmx8G -Xss8M -Djava.io.tmpdir=/home/u20/chipyard/.java_tmp
[info] welcome to sbt 1.8.2 (Temurin Java 1.8.0_345)
[info] loading settings for project chipyard-build from plugins.sbt ...
[info] loading project definition from /home/u20/chipyard/project
[info] loading settings for project chipyardRoot from build.sbt ...
[info] loading settings for project testchipip from build.sbt ...
[info] loading settings for project barf from build.sbt ...
[info] loading settings for project constellation from build.sbt ...
[info] loading settings for project icenet from build.sbt ...
[info] loading settings for project cva6 from build.sbt ...
[info] loading settings for project ibex from build.sbt ...
[info] loading settings for project serv from build.sbt ...
[info] loading settings for project sodor from build.sbt ...
[info] loading settings for project midas_target_utils from build.sbt ...
[info] resolving key references (47609 settings) ...
[info] set current project to chipyardRoot (in build file:/home/u20/chipyard/)
[info] set current project to chipyard (in build file:/home/u20/chipyard/)
[info] Defining assembly / assemblyOutputPath
[info] The new value will be used by assembly
[info] Reapplying settings...
[info] set current project to chipyard (in build file:/home/u20/chipyard/)
```

Make successful resulting in SERV SoC Generation



u20@u20-VirtualBox: ~/chipyard/sims/verilator

```
stDriver_DigitalTop__DepSet_h33dfddee__40.o VTestDriver_DigitalTop__DepSet_h33dfddee__41.o VTestDriver_DigitalTop__DepSet_h33dfddee__42.o VTes
tDriver_DigitalTop__DepSet_h33dfddee__43.o VTestDriver_DigitalTop__DepSet_h33dfddee__44.o VTestDriver_DigitalTop__DepSet_h33dfddee__45.o VTest
Driver_DigitalTop__DepSet_h33dfddee__46.o VTestDriver_DigitalTop__DepSet_h33dfddee__47.o VTestDriver_DigitalTop__DepSet_h33dfddee__48.o VTestD
river_DigitalTop__DepSet_h33dfddee__49.o VTestDriver_DigitalTop__DepSet_h33dfddee__50.o VTestDriver_DigitalTop__DepSet_h33dfddee__51.o VTestDr
iver_DigitalTop__DepSet_h33dfddee__52.o VTestDriver_TilePRCIDomain__DepSet_h1ad82fb8__0.o VTestDriver_TilePRCIDomain__DepSet_he4f7cd44__0.o VT
estDriver_ServTile__DepSet_heec8f299__0.o VTestDriver_ServTile__DepSet_hc194396d__0.o VTestDriver_ServTile__DepSet_hc194396d__1.o VTestDriver_
ServCore__Mz9_S1_Ac__DepSet_hf425de53__0.o VTestDriver_serving__MB2000_S1_R4d494e49__DepSet_h9bbebe1f__0.o VTestDriver_serving__MB2000_S1_R4d4
94e49__DepSet_h6efd994e__0.o VTestDriver_serving_ram__D2000__DepSet_h9ec8da9c__0.o VTestDriver_serving_ram__D2000__DepSet_h73cf75cf__0.o VTest
Driver__main.o VTestDriver__Dpi.o VTestDriver__ConstPool_0.o VTestDriver___024root__Slow.o VTestDriver___024root__DepSet_h662e1026__0__Slow.o
VTestDriver___024root__DepSet_hbb6e3b65__0__Slow.o VTestDriver_TestDriver__Slow.o VTestDriver_TestDriver__DepSet_h7cf3db4e__0__Slow.o VTestDri
ver_TestDriver__DepSet_he152682f__0__Slow.o VTestDriver___024unit__Slow.o VTestDriver___024unit__DepSet_h44496666__0__Slow.o VTestDriver_TestH
arness__Slow.o VTestDriver_TestHarness__DepSet_hff67d21a__0__Slow.o VTestDriver_TestHarness__DepSet_ha45bb60e__0__Slow.o VTestDriver_ChipTop__
Slow.o VTestDriver_ChipTop__DepSet_h09ff5f7a__0__Slow.o VTestDriver_ChipTop__DepSet_hca3ac016__0__Slow.o VTestDriver_DigitalTop__Slow.o VTestD
river_DigitalTop__DepSet_hd9c8db9a__0__Slow.o VTestDriver_DigitalTop__DepSet_hd9c8db9a__1__Slow.o VTestDriver_DigitalTop__DepSet_hd9c8db9a__2_
_Slow.o VTestDriver_DigitalTop__DepSet_hd9c8db9a__3__Slow.o VTestDriver_DigitalTop__DepSet_hd9c8db9a__4__Slow.o VTestDriver_DigitalTop__DepSet
_hd9c8db9a__5__Slow.o VTestDriver_DigitalTop__DepSet_hd9c8db9a__6__Slow.o VTestDriver_DigitalTop__DepSet_hd9c8db9a__7__Slow.o VTestDriver_Digi
talTop__DepSet_h33dfddee__0__Slow.o VTestDriver_TilePRCIDomain__Slow.o VTestDriver_TilePRCIDomain__DepSet_h1ad82fb8__0__Slow.o VTestDriver_Til
ePRCIDomain__DepSet_he4f7cd44__0__Slow.o VTestDriver_ServTile__Slow.o VTestDriver_ServTile__DepSet_heec8f299__0__Slow.o VTestDriver_ServTile__
DepSet_hc194396d__0__Slow.o VTestDriver_ServCoreBlackbox__Ac_S1__Slow.o VTestDriver_ServCoreBlackbox__Ac_S1__DepSet_h6be5b1c3__0__Slow.o VTest
Driver_ServCore__Mz9_S1_Ac__Slow.o VTestDriver_ServCore__Mz9_S1_Ac__DepSet_hf2203fa4__0__Slow.o VTestDriver_serving__MB2000_S1_R4d494e49__Slow
.o VTestDriver_serving__MB2000_S1_R4d494e49__DepSet_h9bbebe1f__0__Slow.o VTestDriver_serving__MB2000_S1_R4d494e49__DepSet_h6efd994e__0__Slow.o
 VTestDriver_serving_ram__D2000__Slow.o VTestDriver_serving_ram__D2000__DepSet_h73cf75cf__0__Slow.o VTestDriver__Syms.o VTestDriver__Syms__1.o
 VTestDriver__Syms__2.o
/home/u20/chipyard/.conda-env/bin/x86_64-conda-linux-gnu-c++    SimDRAM.o SimJTAG.o SimTSI.o SimUART.o mm.o mm_dramsim2.o remote_bitbang.o tes
tchip_htif.o testchip_tsi.o uart.o verilated.o verilated_dpi.o verilated_vpi.o verilated_timing.o verilated_threads.o VTestDriver__ALL.a   -L/
home/u20/chipyard/.conda-env/riscv-tools/lib -Wl,-rpath,/home/u20/chipyard/.conda-env/riscv-tools/lib -L/home/u20/chipyard/sims/verilator -L/h
ome/u20/chipyard/tools/DRAMSim2 -lriscv -lfesvr -ldramsim  -pthread -lpthread -latomic   -o /home/u20/chipyard/sims/verilator/simulator-chipya
rd.harness-ServConfig
rm VTestDriver__ALL.verilator_deplist.tmp
make[1]: Leaving directory '/home/u20/chipyard/sims/verilator/generated-src/chipyard.harness.TestHarness.ServConfig/chipyard.harness.TestHarne
ss.ServConfig'
(/home/u20/chipyard/.conda-env) u20@u20-VirtualBox:~/chipyard/sims/verilator$
```


SERV Core
AXI4ToTL Adapter
Tile Bus
SERV Tile
System Bus
Control Bus
BootROM
PLIC
CLINT
Debug Unit
Memory Bus
Periphery Bus
Memory devices can be attached here.
Custom Peripherals can be attached here.

IMPROVEMENT AND
FUTURE WORK

All tasks are targeted for completion before the FYP Exhibition. However, even without these additions, the integration of SERV as a functioning tile in Chipyard marks the successful completion of the core objective.

01

We aim to connect SERV as a memory-mapped peripheral using an AXI4 slave interface, enabling external masters to communicate directly with the core.

02

Plan to support ELF binary loading into SERV's internal memory through Verilator simulation using Front-End Server (FESVR), as in Chipyard's communication mechanisms.

03

Explore performance benchmarking or RTL-level power analysis of the SERV core inside Chipyard.

CONCLUSION

This project marks the successful hybridization of the SERV bit-serial RISC-V core with the Chipyard SoC ecosystem, showcasing that even the most compact cores can be brought into advanced design environments.

Verified core functionality through Verilator simulation and waveform inspection, confirming correct instruction execution. Demonstrated that minimalist processors like SERV can coexist with robust SoC ecosystems, paving the way for ultra-low power applications. This work sets a foundation for future expansion, including memory-mapped peripherals, ELF-based program loading, and custom system designs.