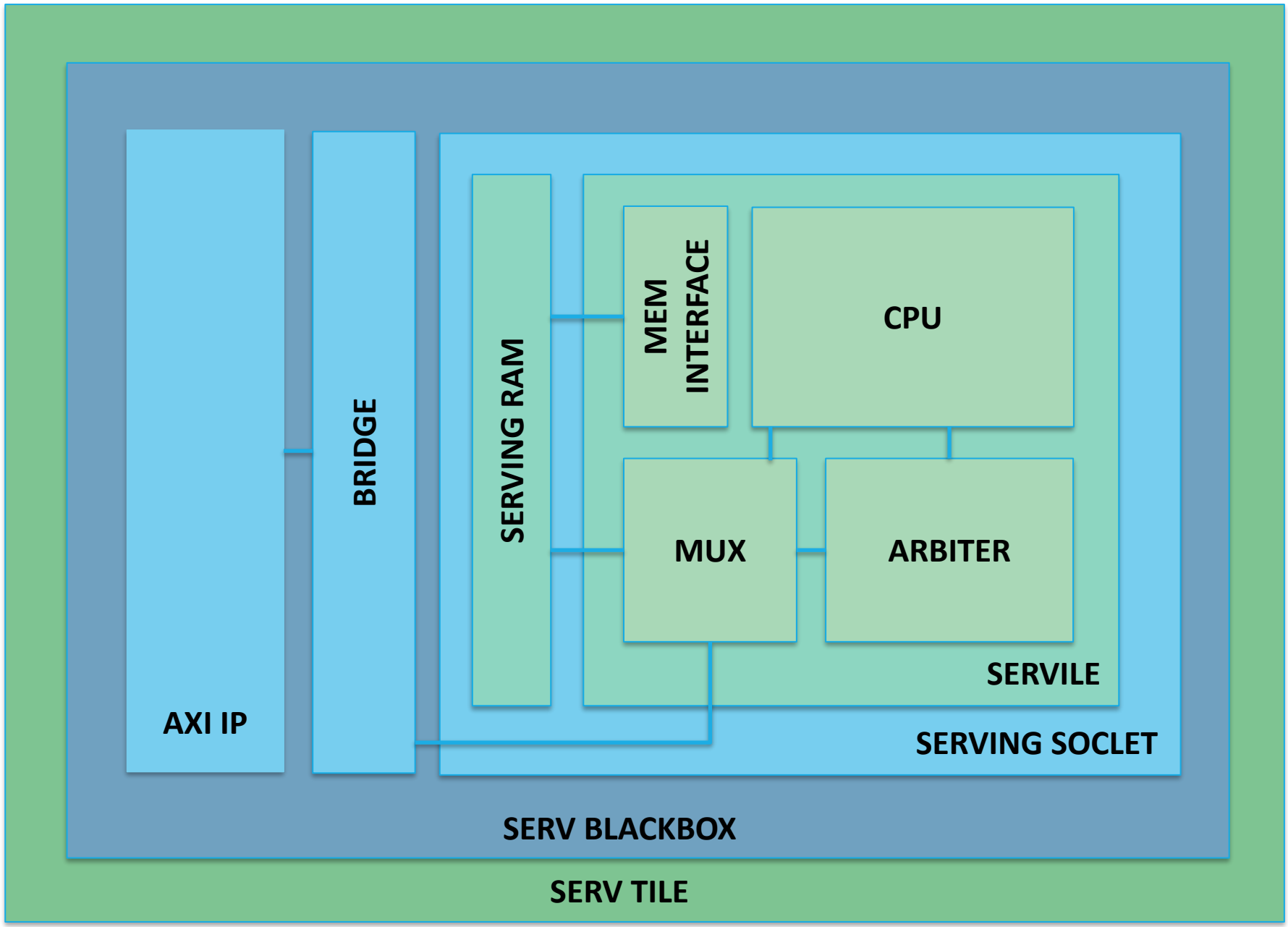
Integration of SERV Core in Chipyard

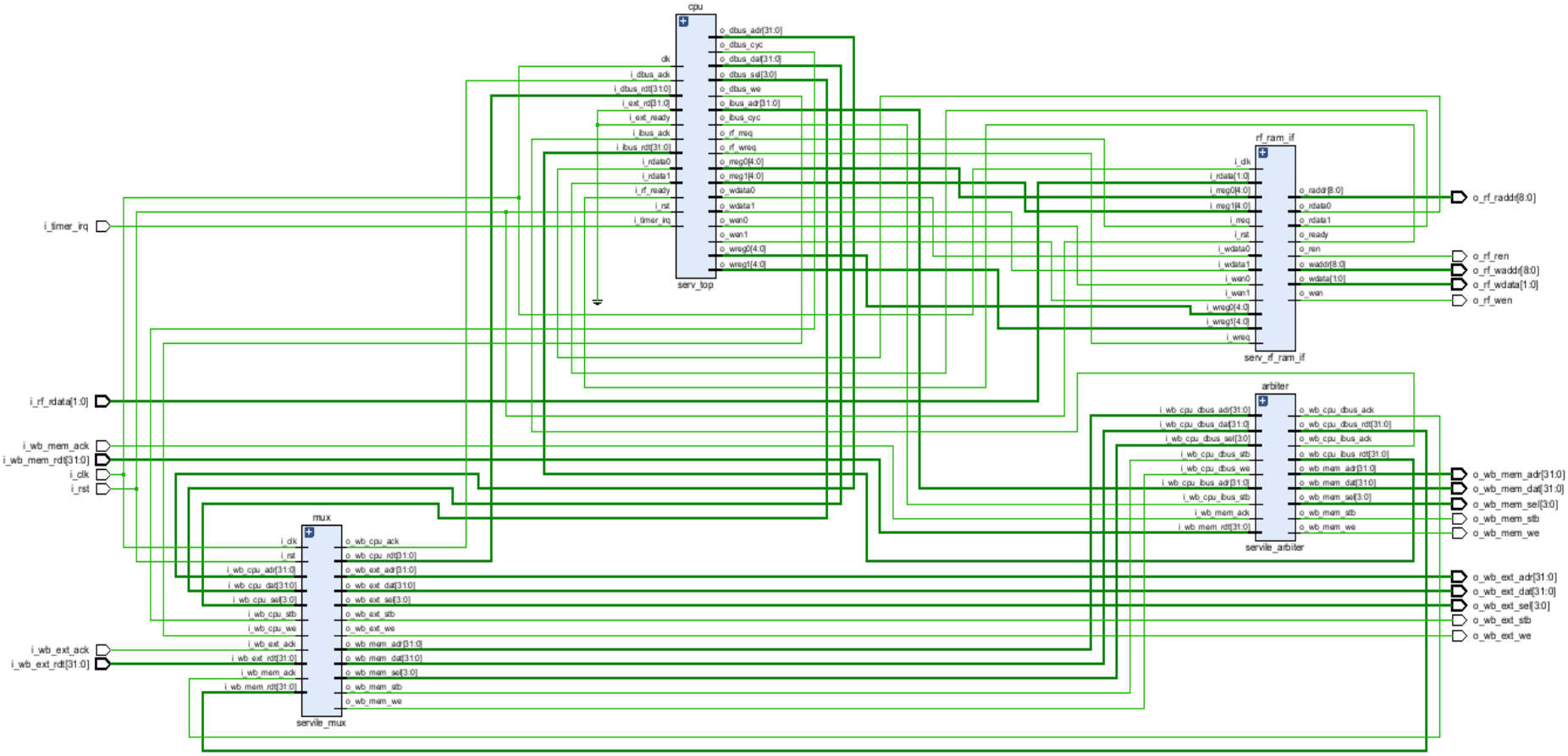
Progress Presentation

Presented By: Ayesha Qazi
Humail Nawaz

Project Supervisor: Engr. Mehmoona Gul
Co-Supervisor: Dr. Aneesullah


SERV TILE
SERV BLACKBOX
AXI IP
BRIDGE
SERVING RAM
MEM INTERFACE
CPU
MUX
ARBITER
SERVILE
SERVING SOCLET

Servile: Convenience Wrapper



servile_mux

Wishbone address-based multiplexer that routes CPU Wishbone requests to either memory or external peripherals.
Distinguishes transactions by checking the top two bits of the address **(i_wb_cpu_adr[31:30])** and routes accordingly, enabling **separation of address regions** for internal memory vs. external devices.

Routes Transactions to:
`wb_mem_*`
`wb_ext_*`

servile_arbiter

Wishbone bus arbiter that merges SERV's instruction bus (**ibus**) and data bus (**dbus**) into a single Wishbone master interface.
Handles arbitration by prioritizing instruction bus when active, otherwise allows data bus transactions.
Critical for enabling both code fetch and data access over a **shared Wishbone memory bus**.

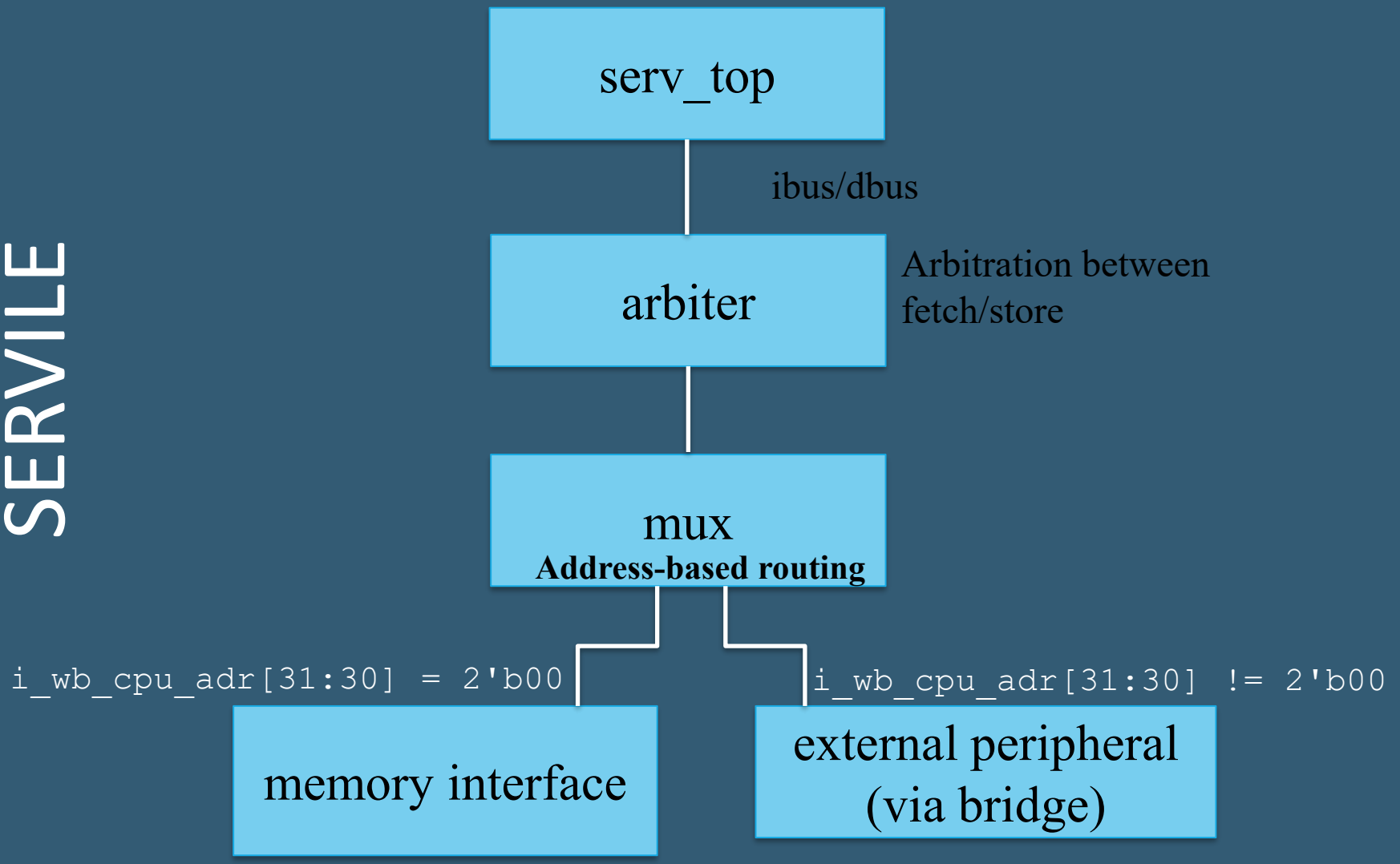
serv_top

Central compute engine of the wrapper system, generating bus transactions that must be arbitrated and routed via the arbiter and mux.
Highly parameterizable however does not expose standard Wishbone signals.

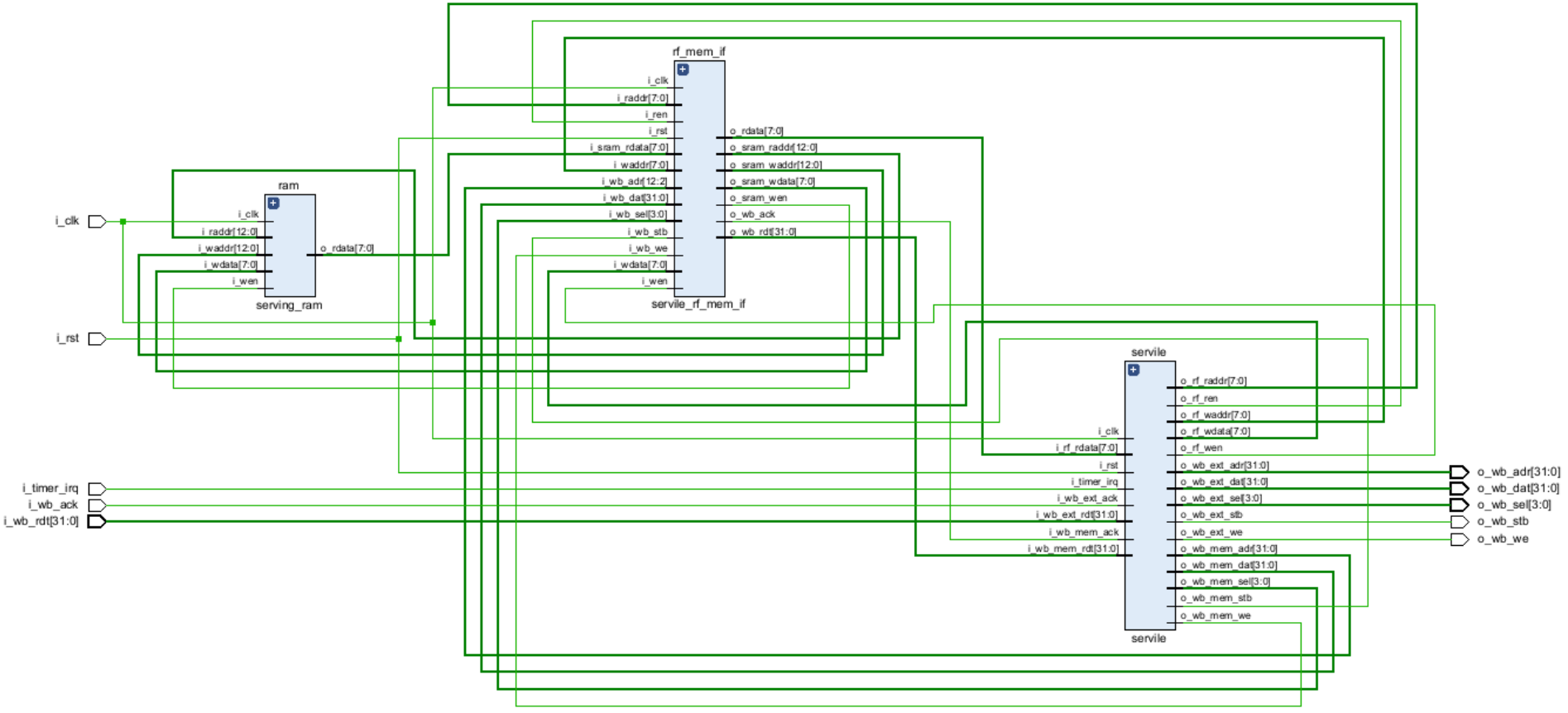
serv_rf_ram_if

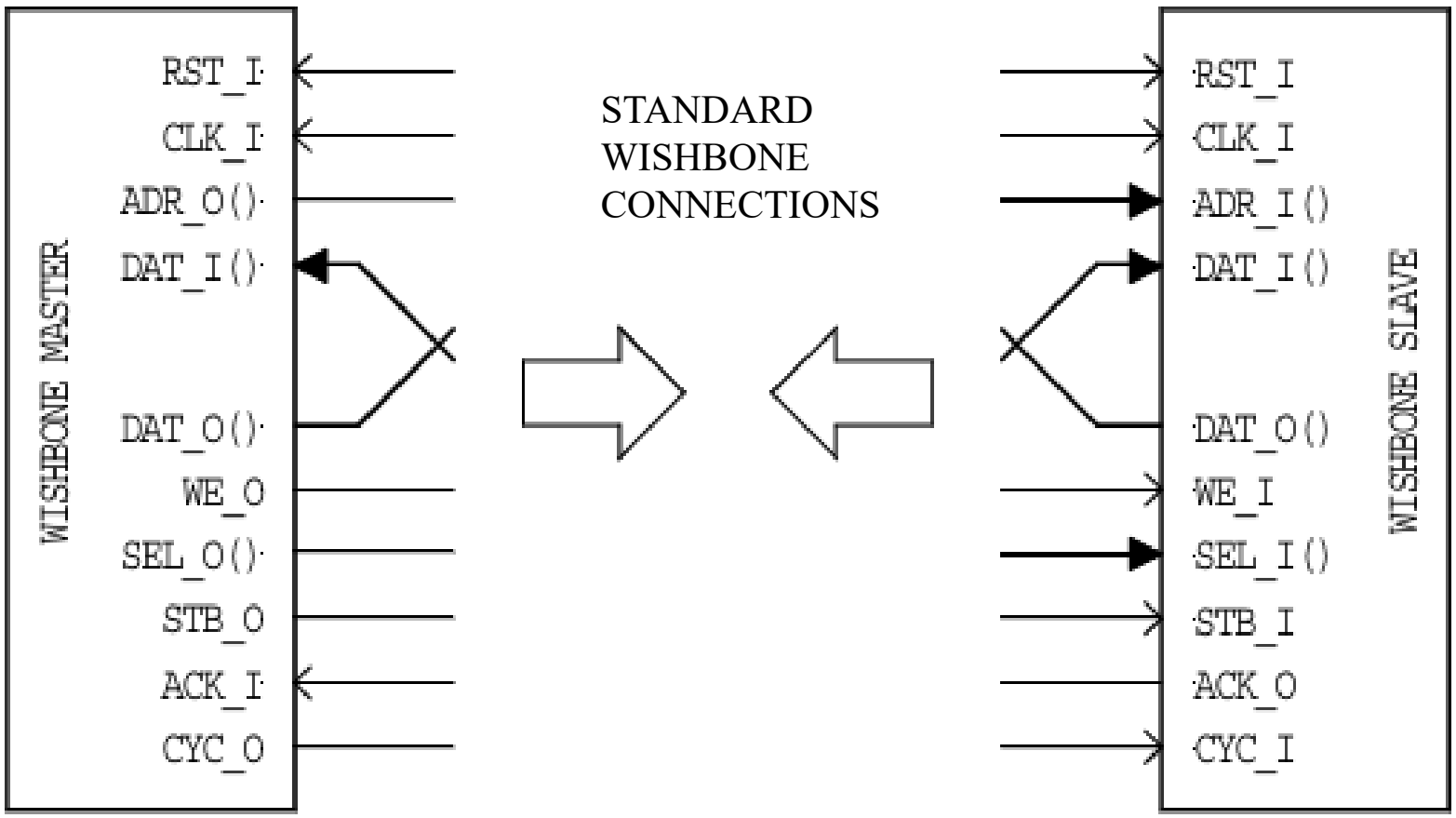
Provides a simple read/write interface between the SERV CPU's register file and an external SRAM or memory block.
Handles partial-width operations, pipelining, and address packing for CSR + GPR registers in a unified RAM.

SERVILE


serv_top
ibus/dbus
arbiter
Arbitration between fetch/store
mux
Address-based routing
i_wb_cpu_adr[31:30] = 2'b00
i_wb_cpu_adr[31:30] != 2'b00
memory interface
external peripheral (via bridge)

Serving SoClet

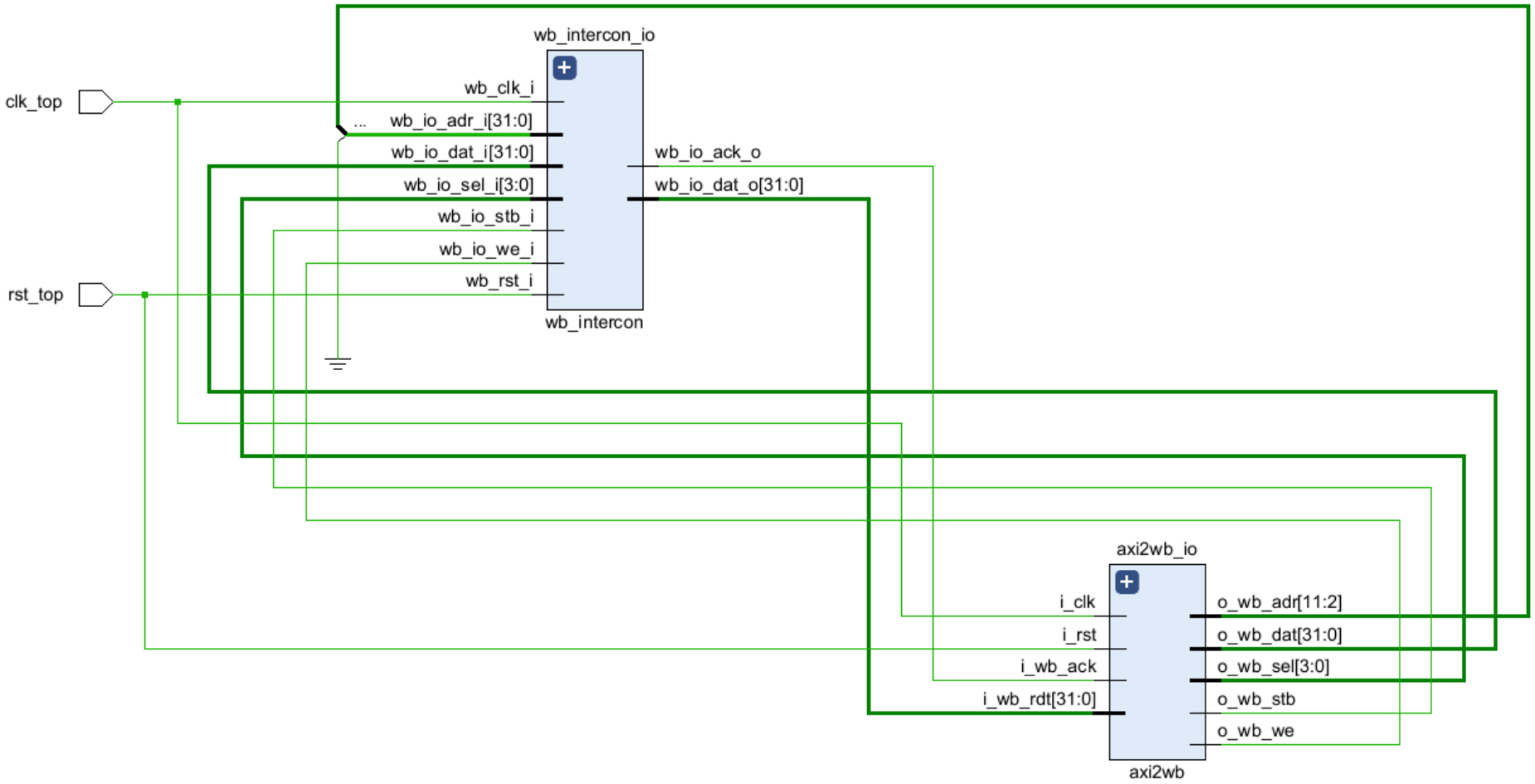




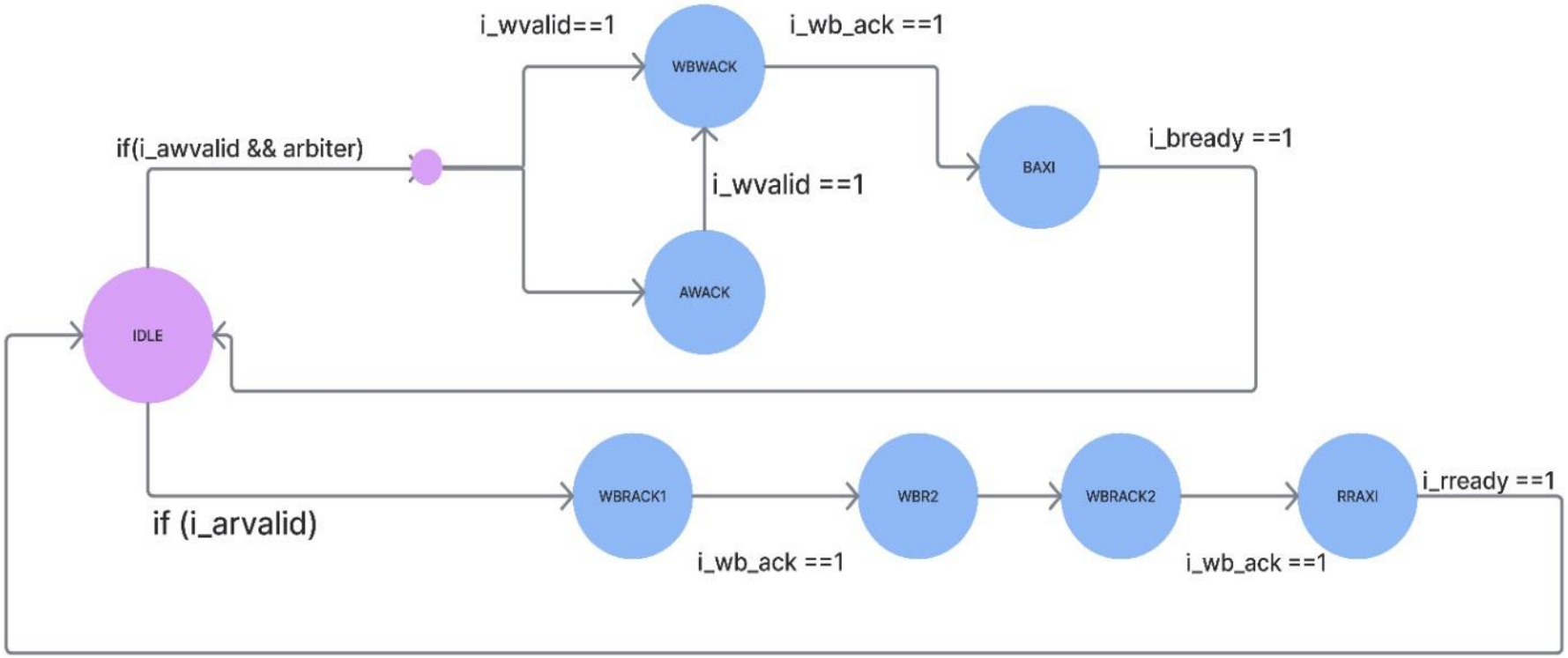
AXI-TO-WISHBONE BRIDGE

SERV SIDE (SERVING SOCLET)

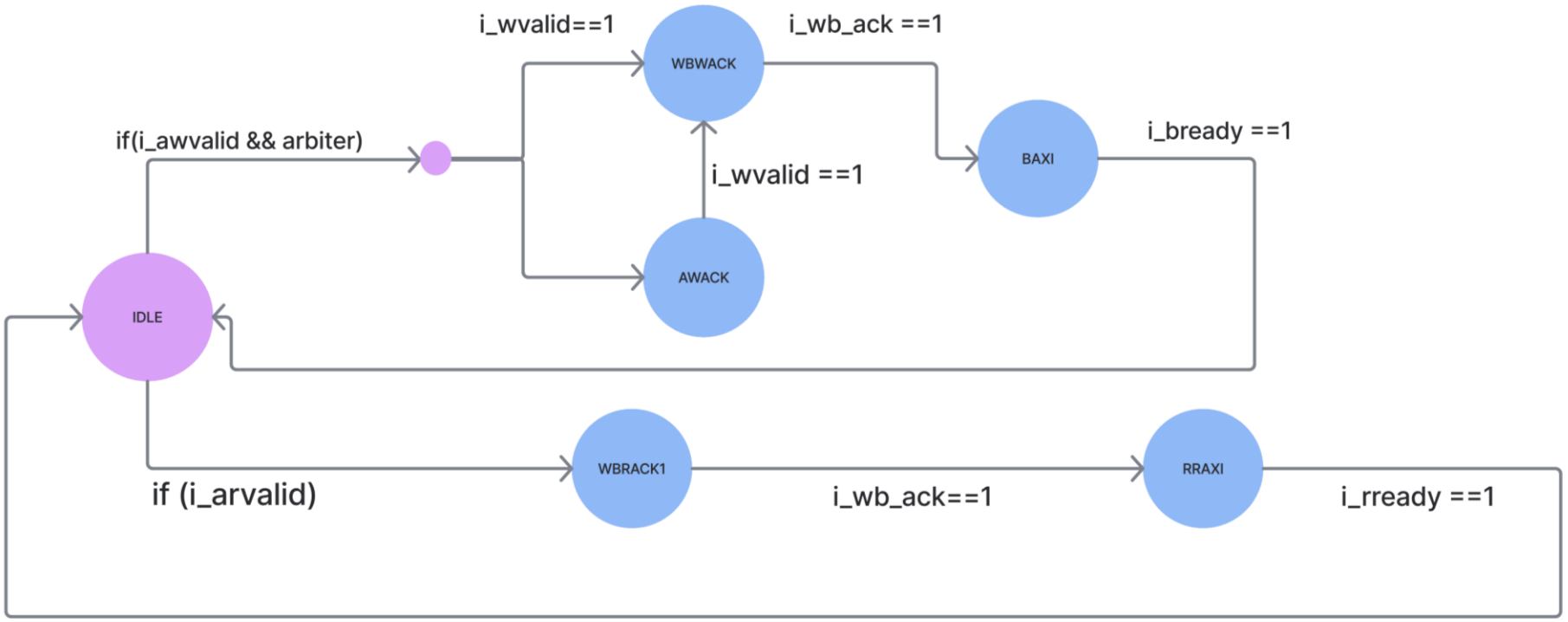
SoClet Connected to Bridge



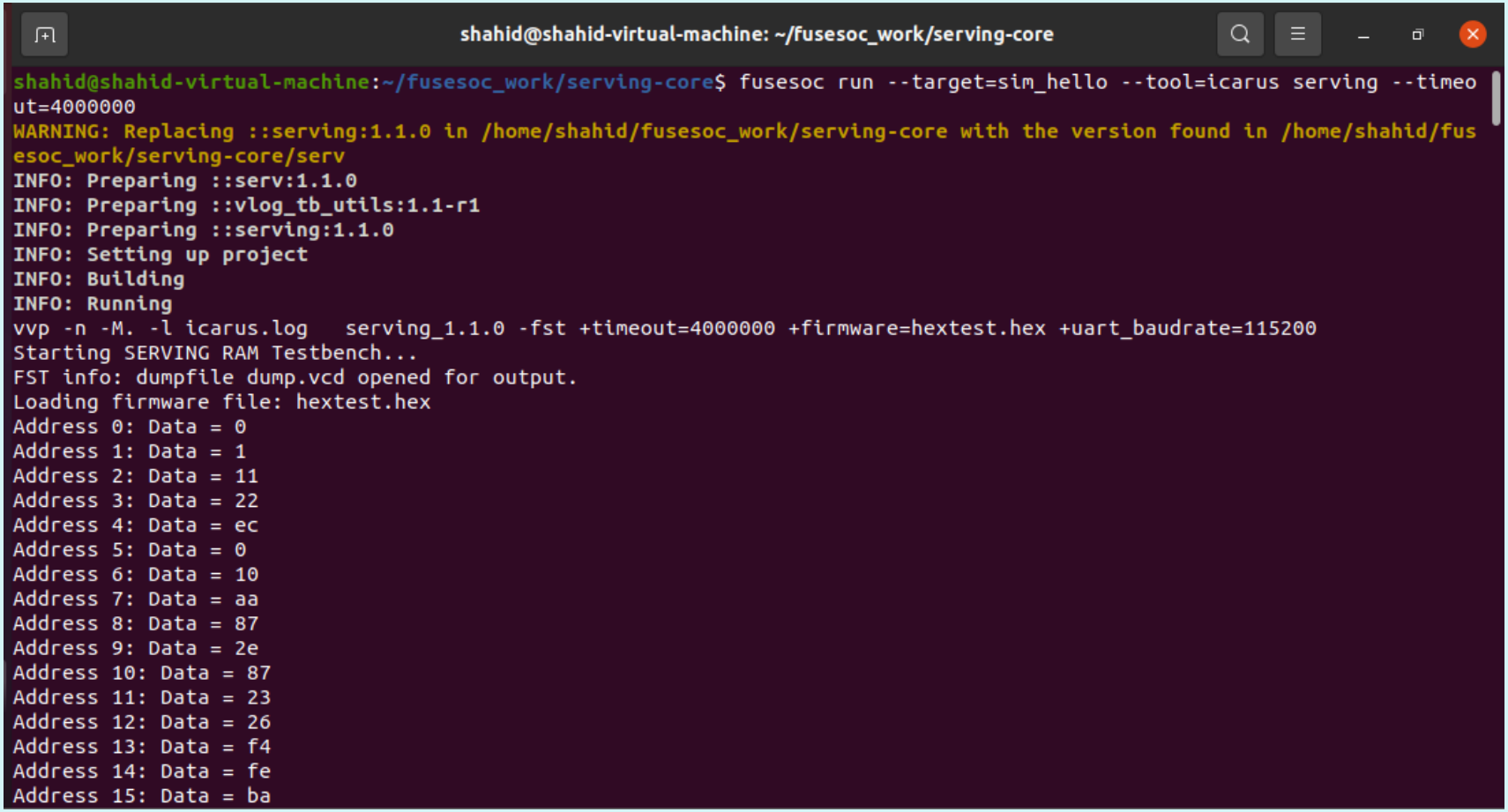
AXI-to-Wishbone Bridge



Modification for 32-bit System



Writing Hex to Serving RAM – via FuseSoC



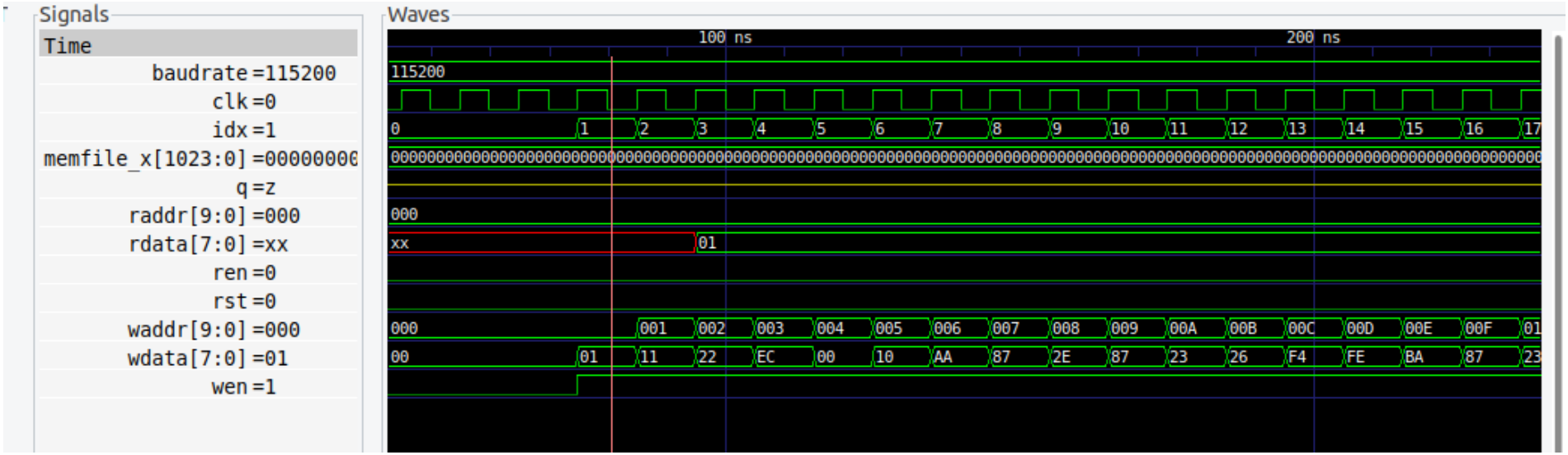
The Ubuntu terminal window displays the successful execution of a simulation using **Icarus Verilog** through the **FuseSoC** flow. The process involves initializing the simulation environment, compiling the **serving.core** file containing instance of the testbench that writes a hexadecimal memory file (.hex) into the **serving_ram** memory module.

Results

Text Editor — 23:01 11 ابريل

Open — hextest.hex — ~/HEXTEST — Save

```
1 @00000000
2 01 11 22 EC 00 10 AA 87 2E 87 23 26 F4 FE BA 87
3 23 24 F4 FE 03 27 C4 FE 83 27 84 FE B9 9F 81 27
4 3E 85 62 64 05 61 82 80 01 11 06 EC 22 E8 00 10
5 91 45 0D 45 EF F0 DF FC AA 87 23 26 F4 FE 83 27
6 C4 FE 3E 85 E2 60 42 64 05 61 82 80
```



AXI IP Integration

As the next step, a custom-designed **Adder IP** was instantiated in the top-level module and connected to the **AXI-to-Wishbone bridge**. The Adder IP was implemented with a 64-bit AXI interface and used standard AXI4-Lite signals for read and write operations.

In the top module `(wb_intercon2bridge)`:

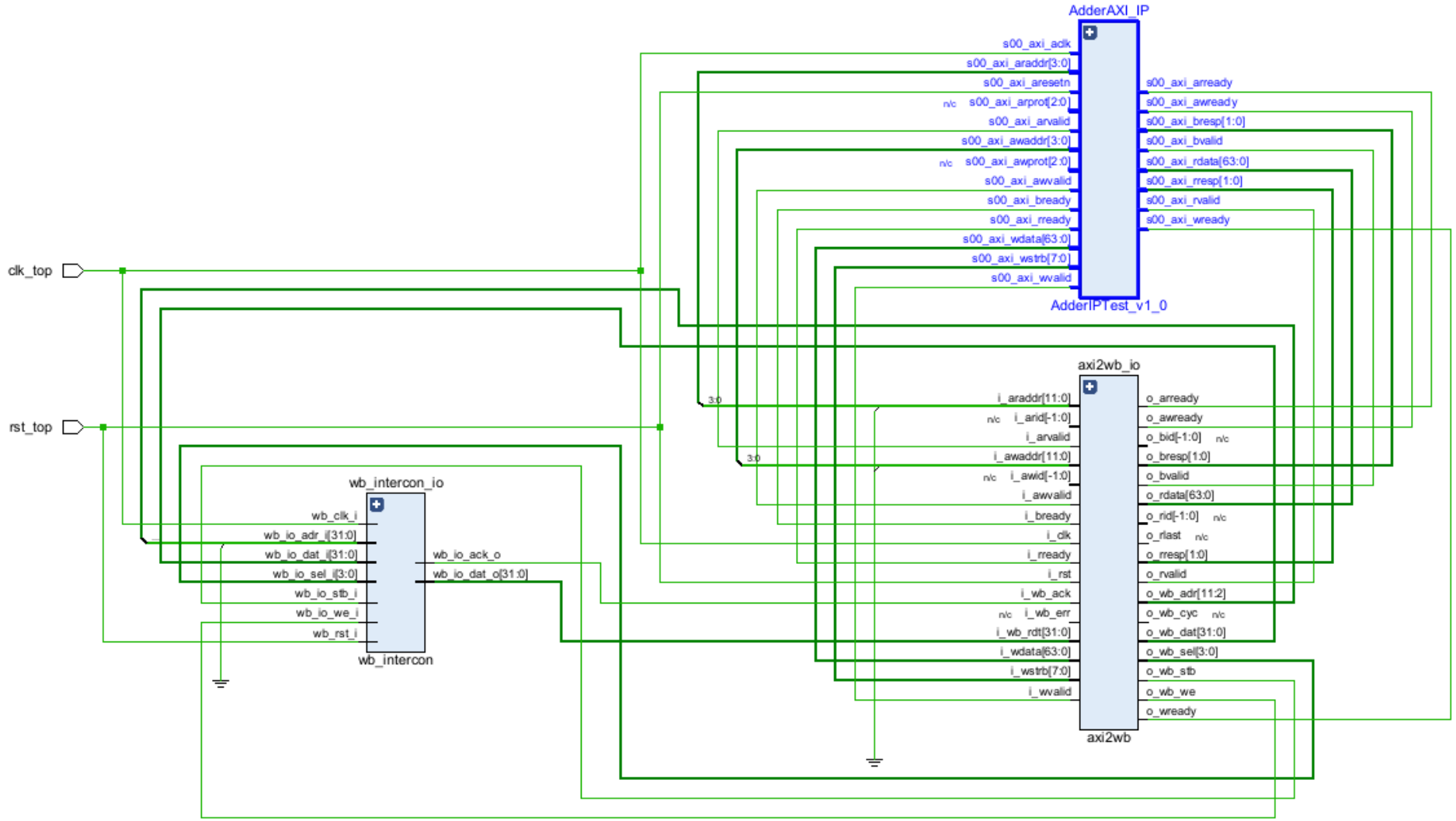
The **Adder AXI IP** receives AXI signals such as address, data, valid, ready, and response.

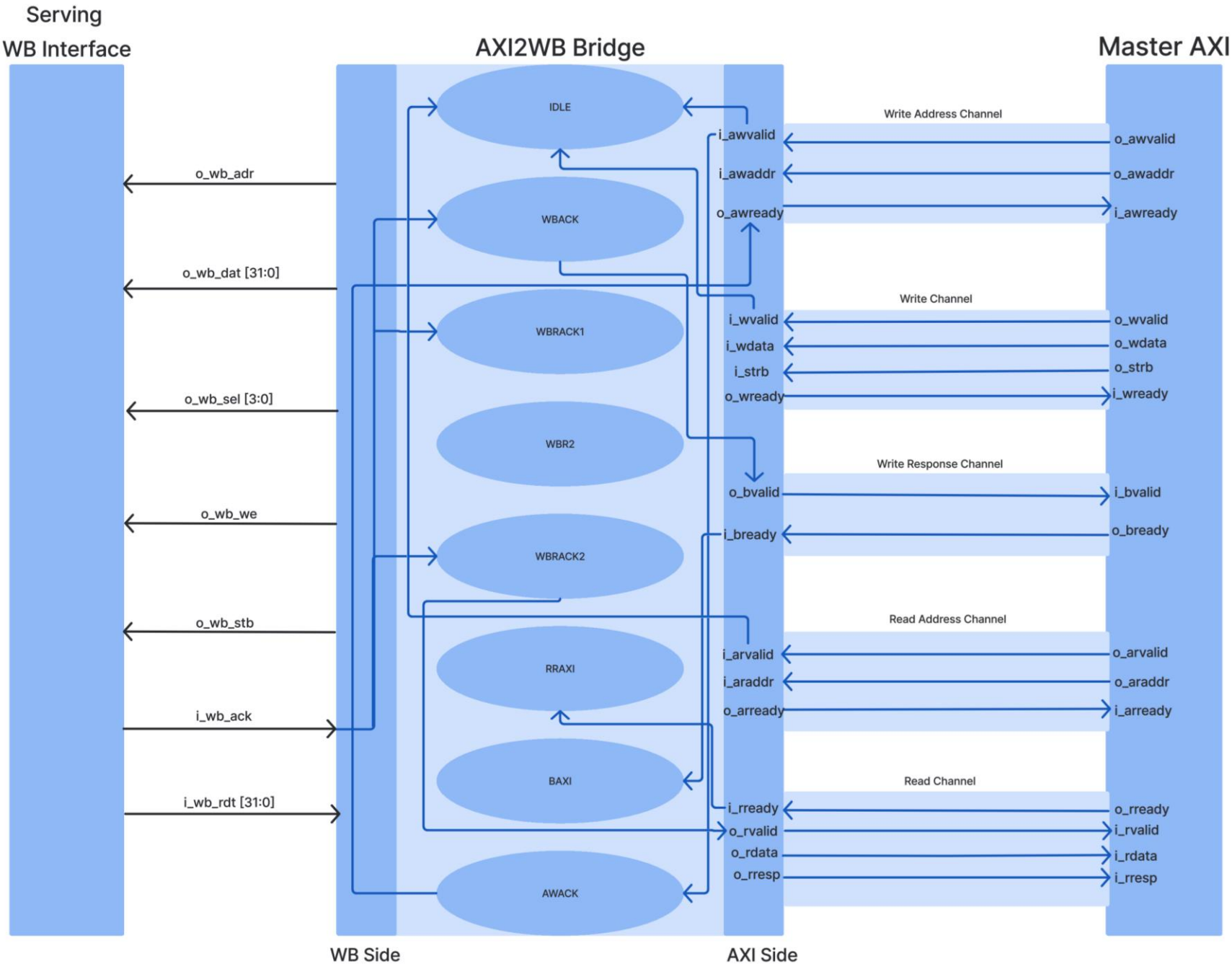
These signals are routed through the **AXI-to-Wishbone bridge**, which converts AXI transactions into equivalent Wishbone protocol signals.

The Wishbone signals are then handled by a **Wishbone interconnect module**, allowing the system to test correct read/write operations via the Adder IP.

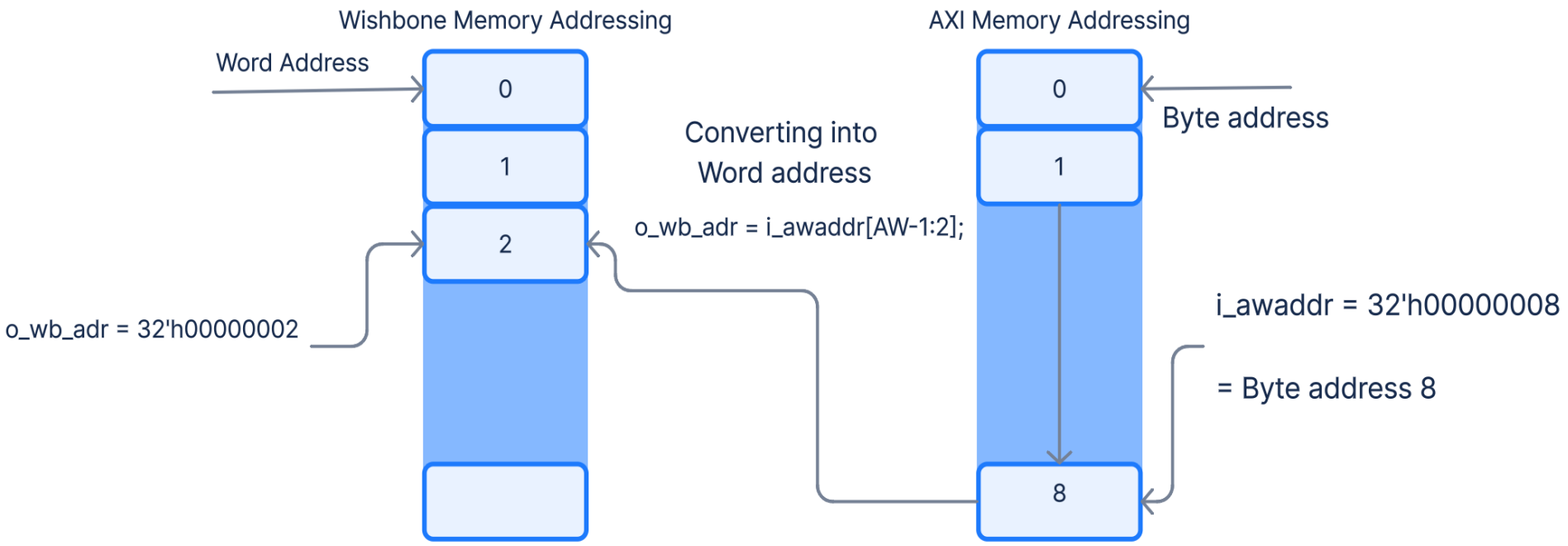
*In future iterations, the Adder AXI IP will be replaced with a **formal AXI Verification IP** to enable thorough protocol checking and stress testing under various conditions.*

AXI IP connected to Bridge

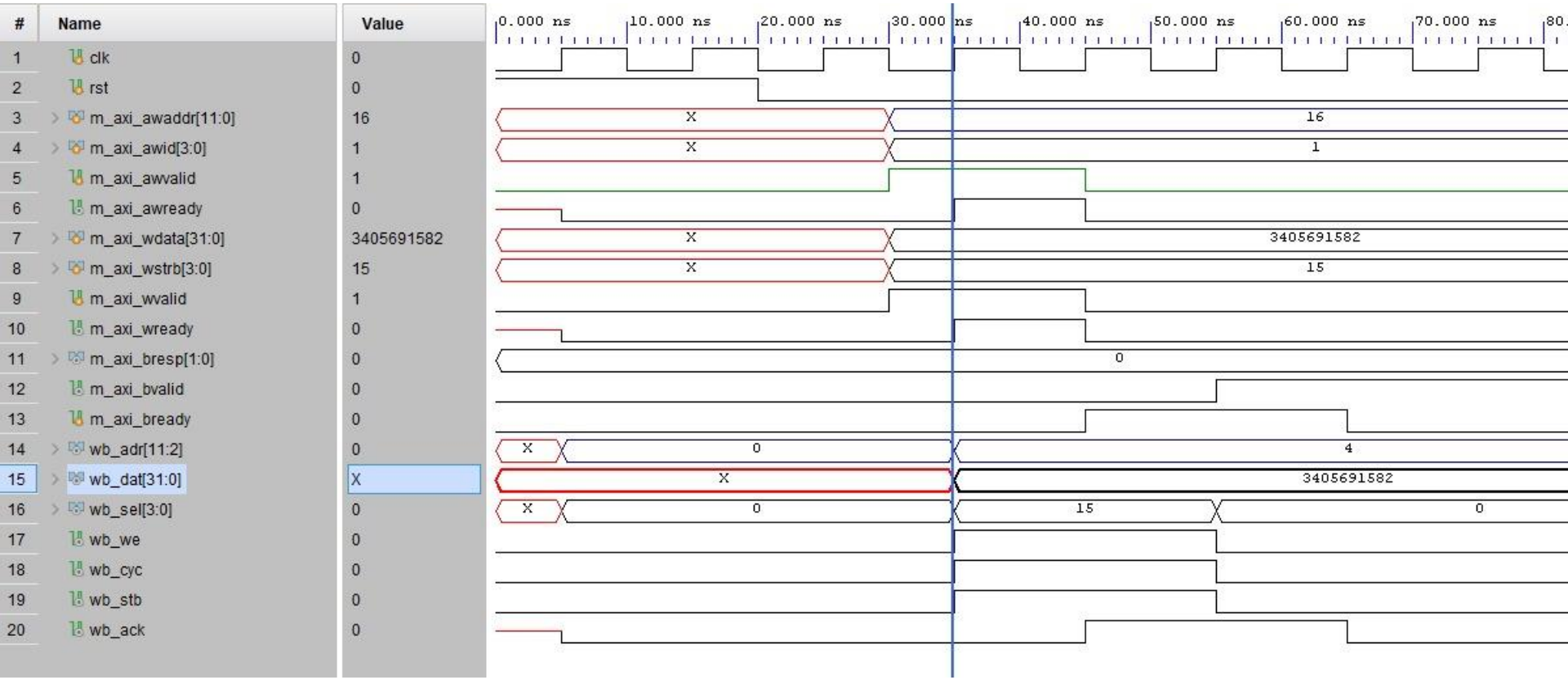


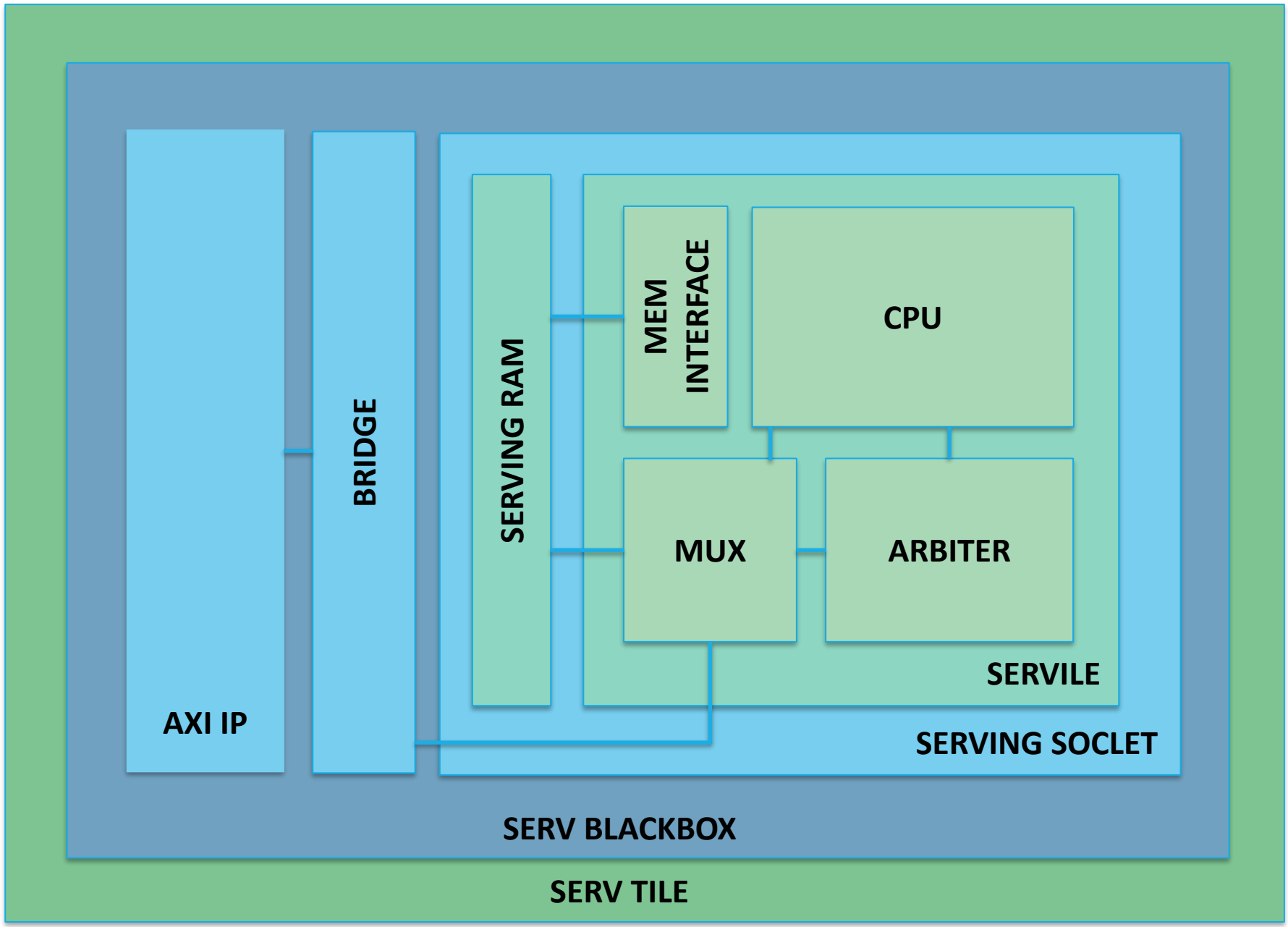

Serving
WB Interface
AXI2WB Bridge
Master AXI
IDLE
WBACK
WBRACK1
WBR2
WBRACK2
RRAXI
BAXI
AWACK
o_wb_adr
o_wb_dat [31:0]
o_wb_sel [3:0]
o_wb_we
o_wb_stb
i_wb_ack
i_wb_rdt [31:0]
Write Address Channel
i_awvalid
i_awaddr
o_awready
o_awvalid
o_awaddr
i_awready
Write Channel
i_wvalid
i_wdata
i_strb
o_wready
o_wvalid
o_wdata
o_strb
i_wready
Write Response Channel
o_bvalid
i_bready
i_bvalid
o_bready
Read Address Channel
i_arvalid
i_araddr
o_arready
o_arvalid
o_araddr
i_arready
Read Channel
i_rready
o_rvalid
o_rdata
o_rresp
o_rready
i_rvalid
i_rdata
i_rresp
WB Side
AXI Side

Address Translation



Bridge Verification




SERV TILE
SERV BLACKBOX
AXI IP
BRIDGE
SERVING RAM
MEM INTERFACE
CPU
MUX
ARBITER
SERVILE
SERVING SOCLET

Integration Flow – (Tentative)