MY PROJECTS

ADHIRAJ MANDAL

ELECTRONICS AND TELECOMMUNICATION ENGINEERING STUDENT

TABLE OF CONTENTS

AHB2APB BRIDGE RTL DESIGN

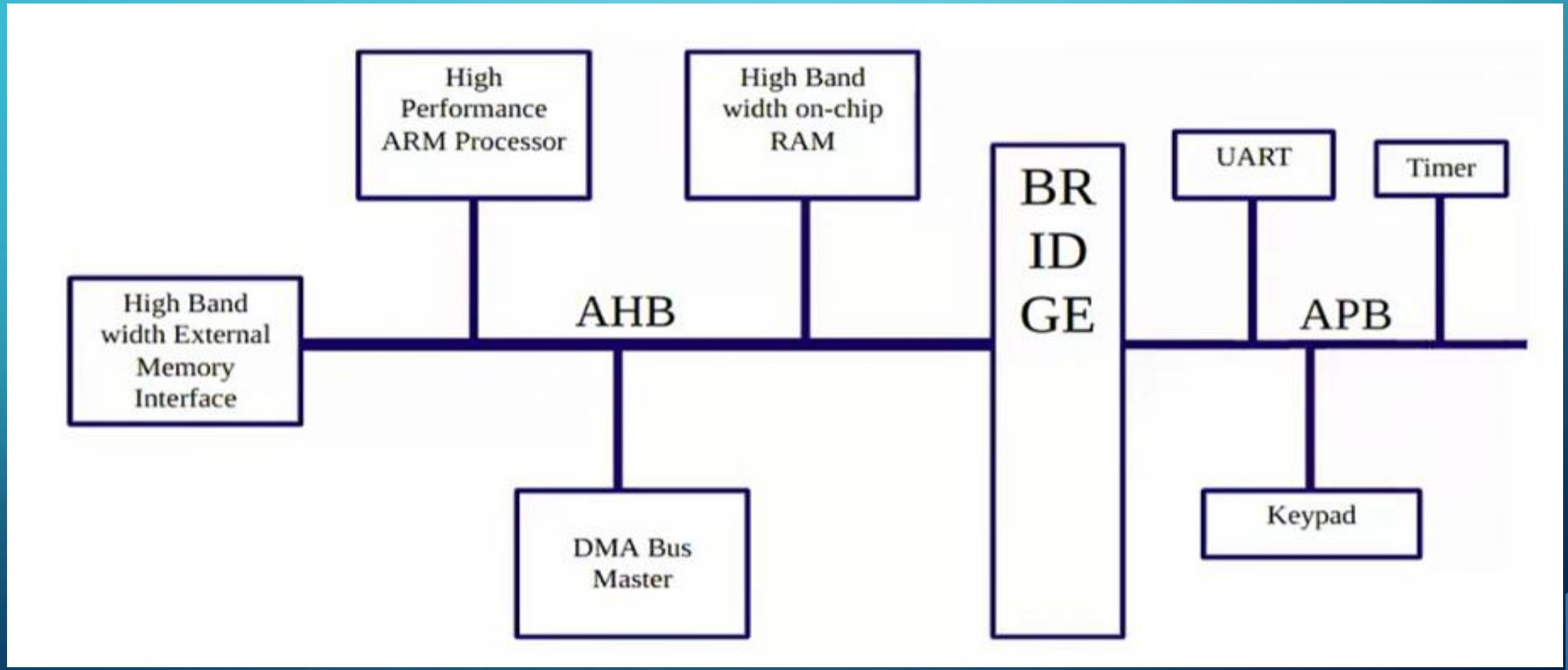
AHB2APB BRIDGE RTL DESIGN

- **Objective:**
  To design and implement an **AHB to APB Bridge** that connects high-speed AHB peripherals with low-speed APB devices in SoC architectures.
- The AHB2APB Bridge is a crucial design for efficient communication between the high-performance Advanced High-performance Bus (AHB) and the low-power Advanced Peripheral Bus (APB). This project implements the bridge to enable protocol conversion, ensuring seamless data transfer between the two buses. The design includes key modules like the AHB Slave Interface and APB Controller, tested and verified for accuracy, timing, and functionality.
- **Key Focus:**
- Protocol conversion (AHB → APB)
- Efficient data and control transfer
- Verification and synthesis of RTL design

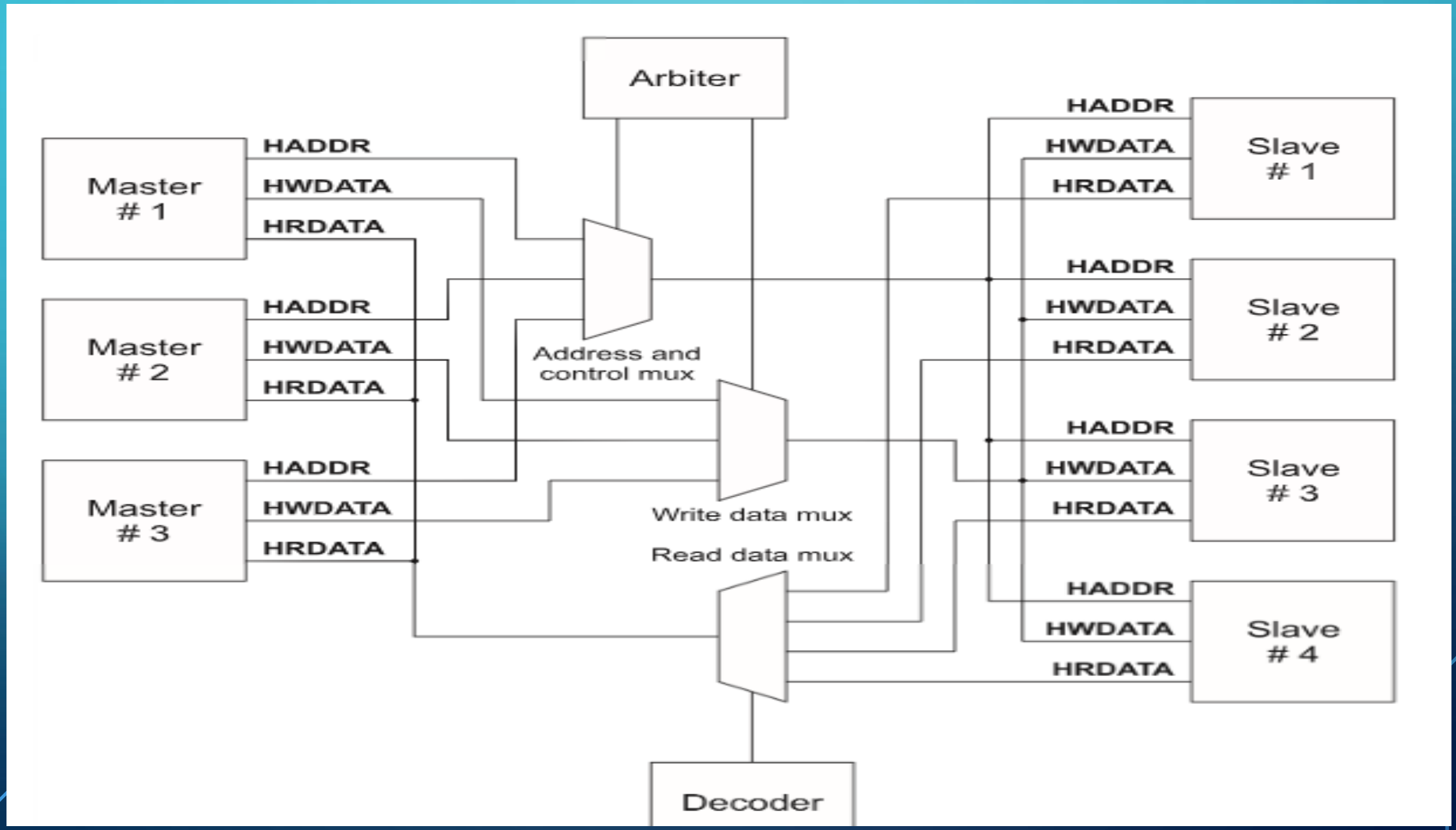
ARCHITECTURE

- **Overview:**
  The bridge acts as an interface between the AHB master and APB slaves.
- Receives AHB transactions
- Translates them into APB-compatible signals
- Ensures data integrity and synchronization
- **Main Blocks:**
- AHB Slave Interface
- APB Controller
- Bridge Integration Module

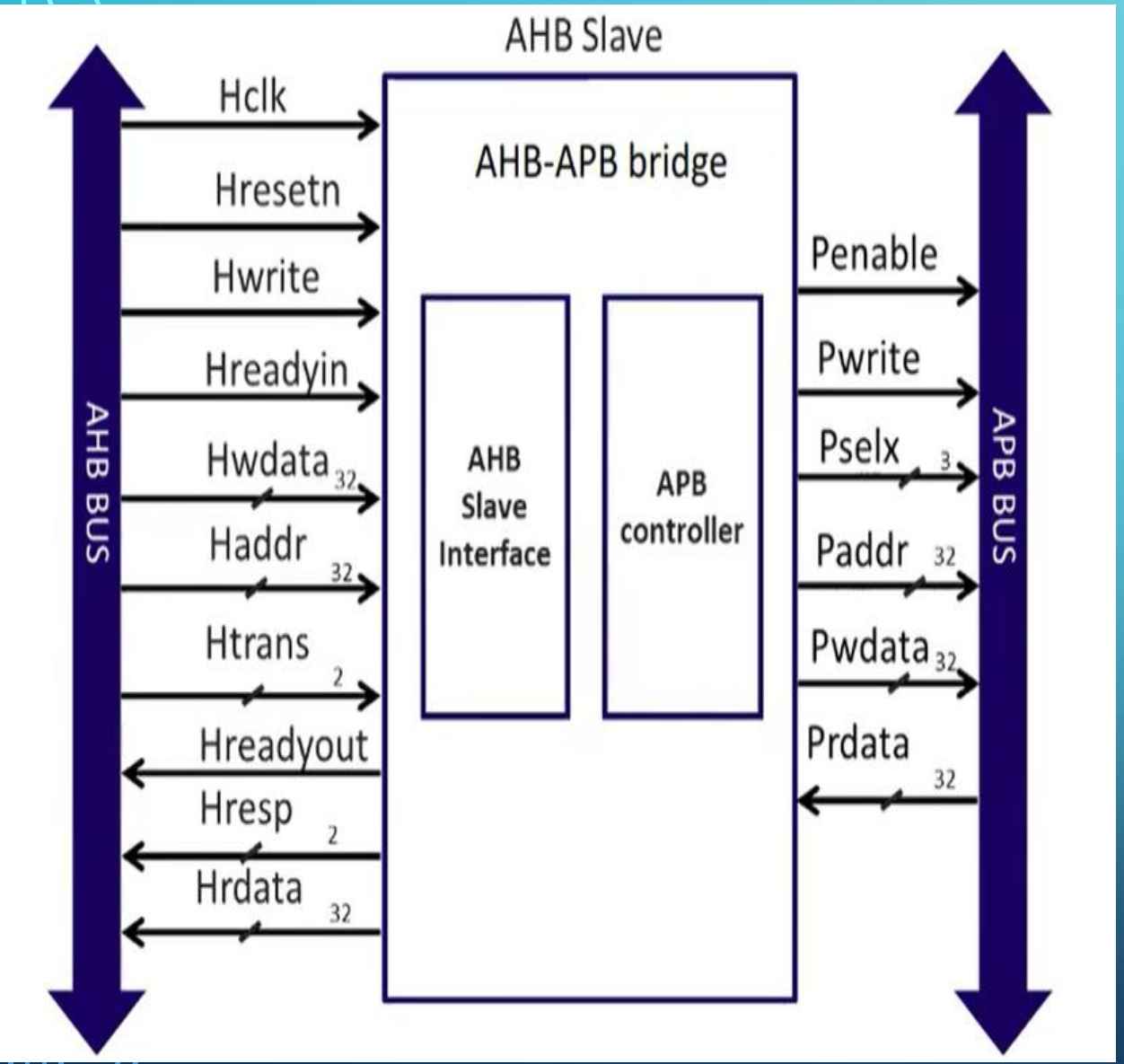
ARCHITECTURE

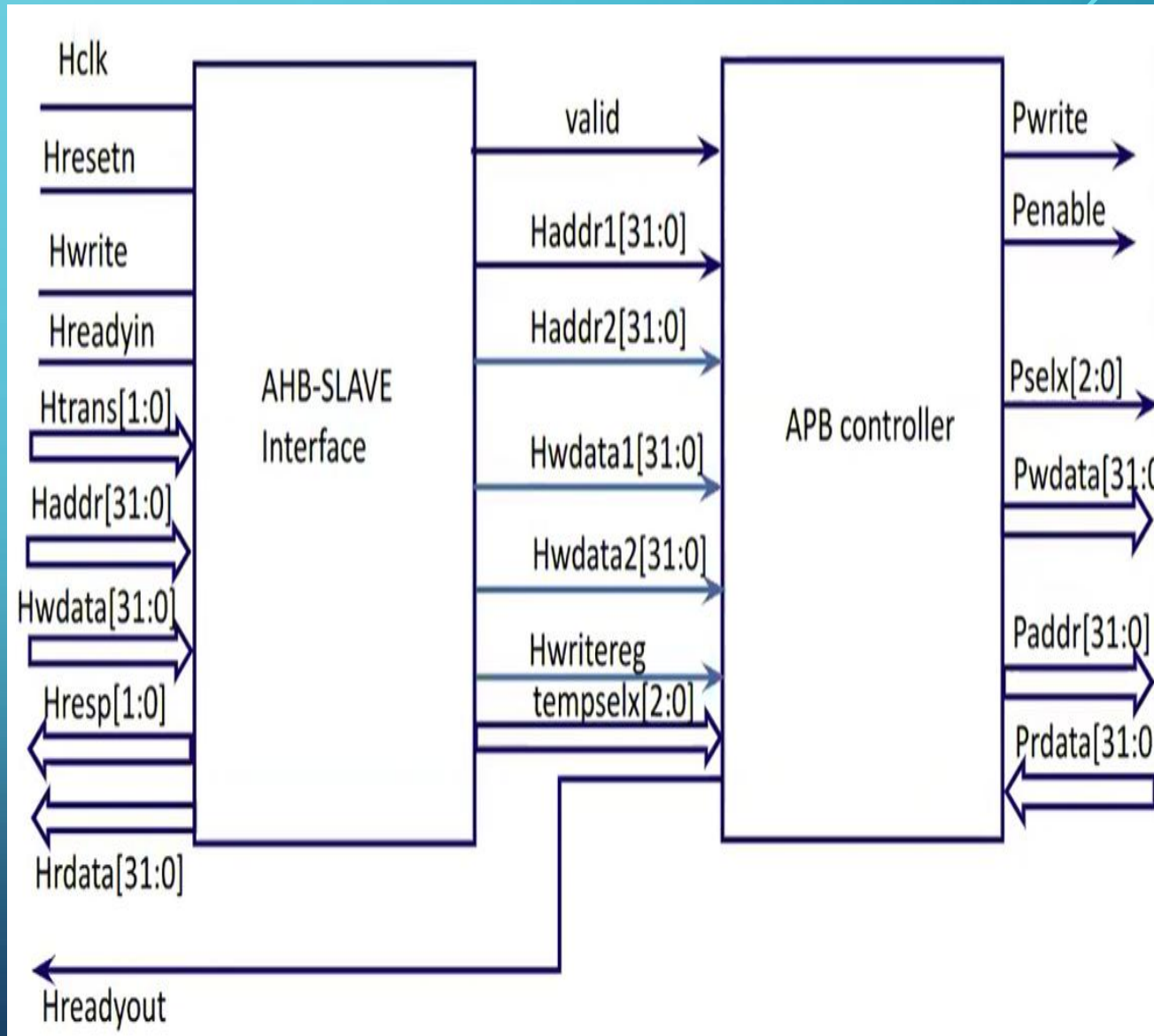


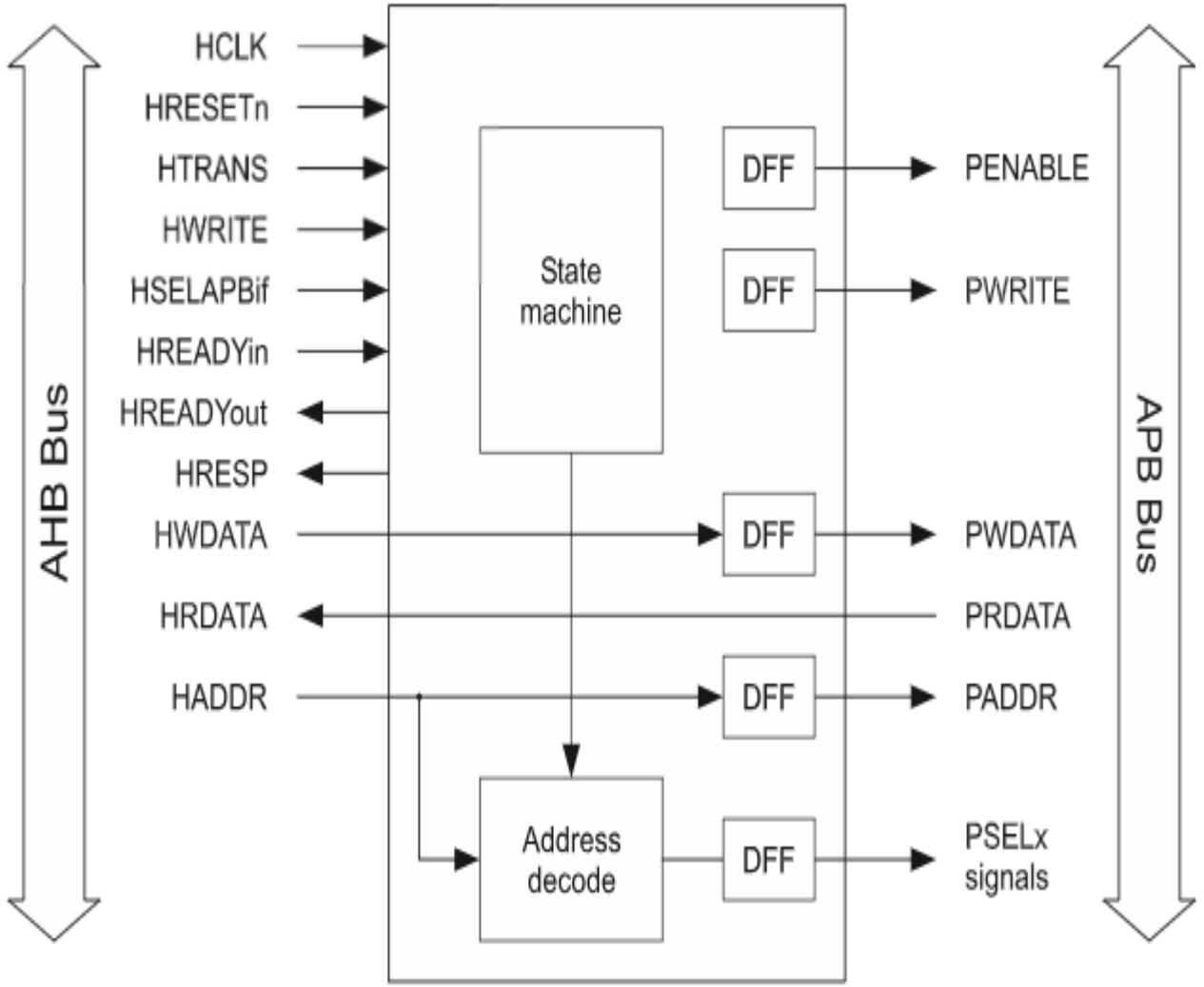
MULTIPLEXOR INTERCONNECTION

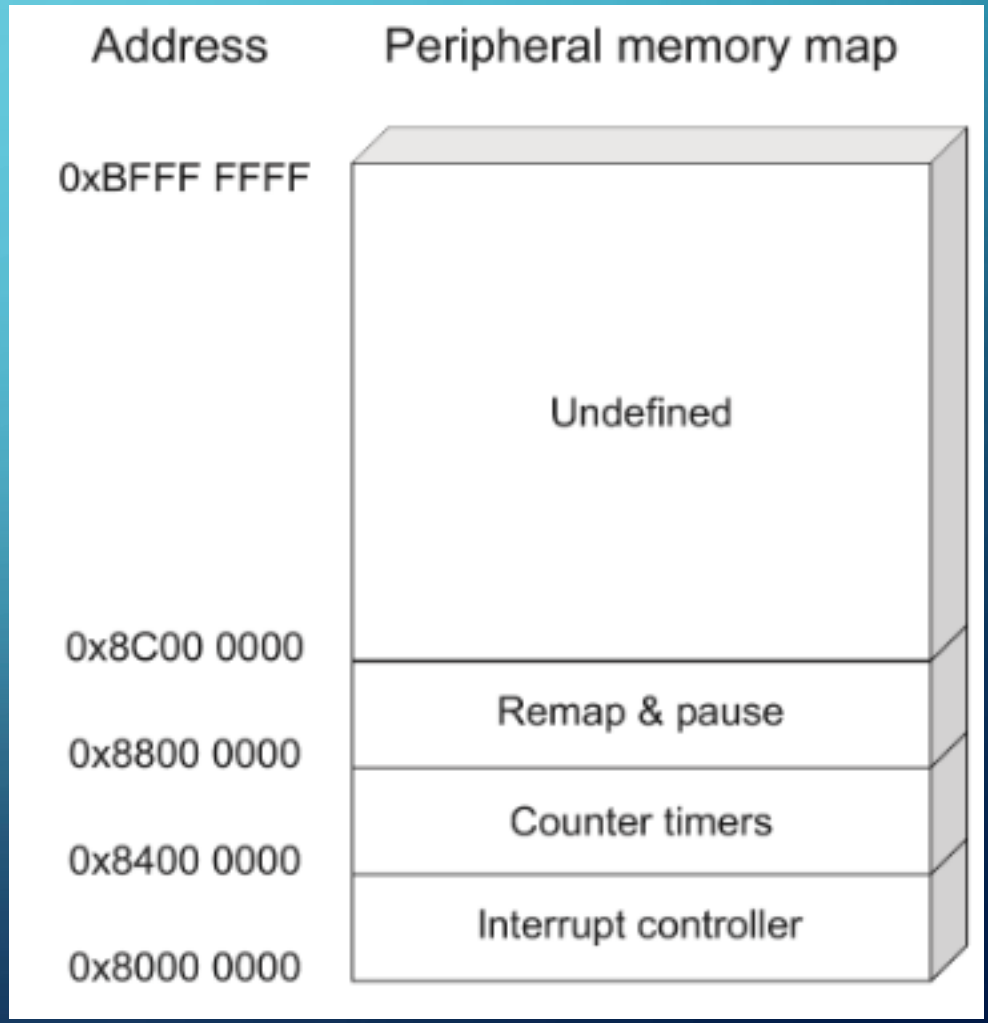


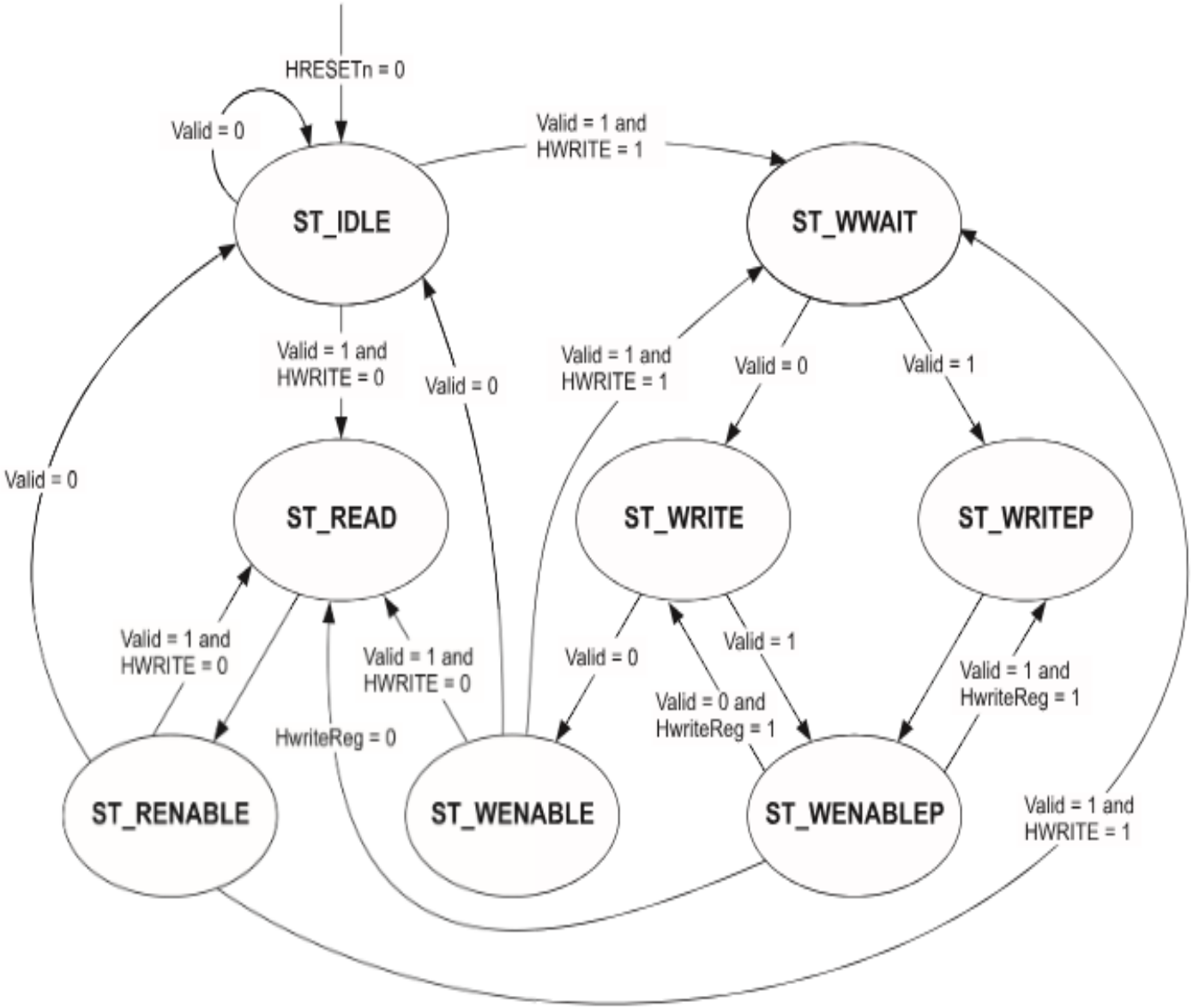
BLOCK DIAGRAM

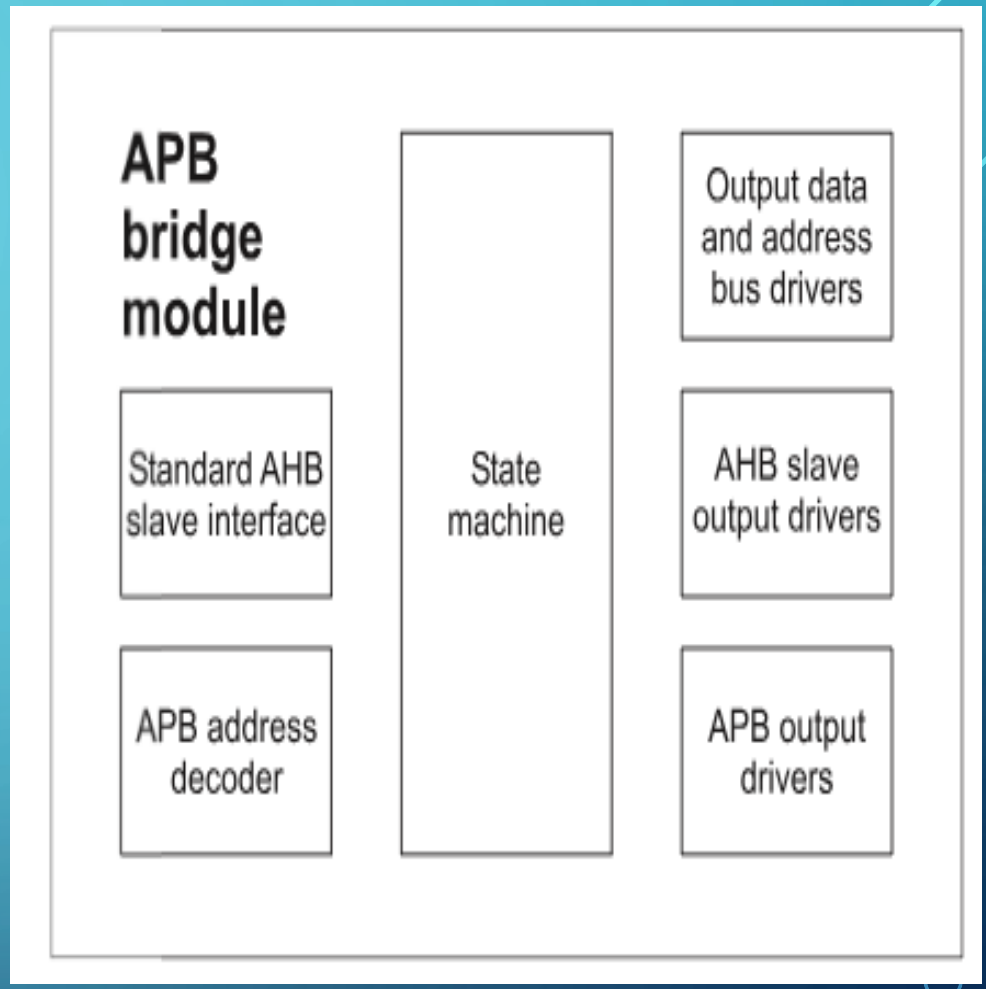





AHB Bus
HCLK
HRESETn
HTRANS
HWRITE
HSELAPBif
HREADYin
HREADYout
HRESP
HWDATA
HRDATA
HADDR
State machine
Address decode
DFF
DFF
DFF
DFF
DFF
PENABLE
PWRITE
PWDATA
PRDATA
PADDR
PSELx signals
APB Bus


Address
Peripheral memory map
0xBFFF FFFF
Undefined
0x8C00 0000
Remap & pause
0x8800 0000
Counter timers
0x8400 0000
Interrupt controller
0x8000 0000

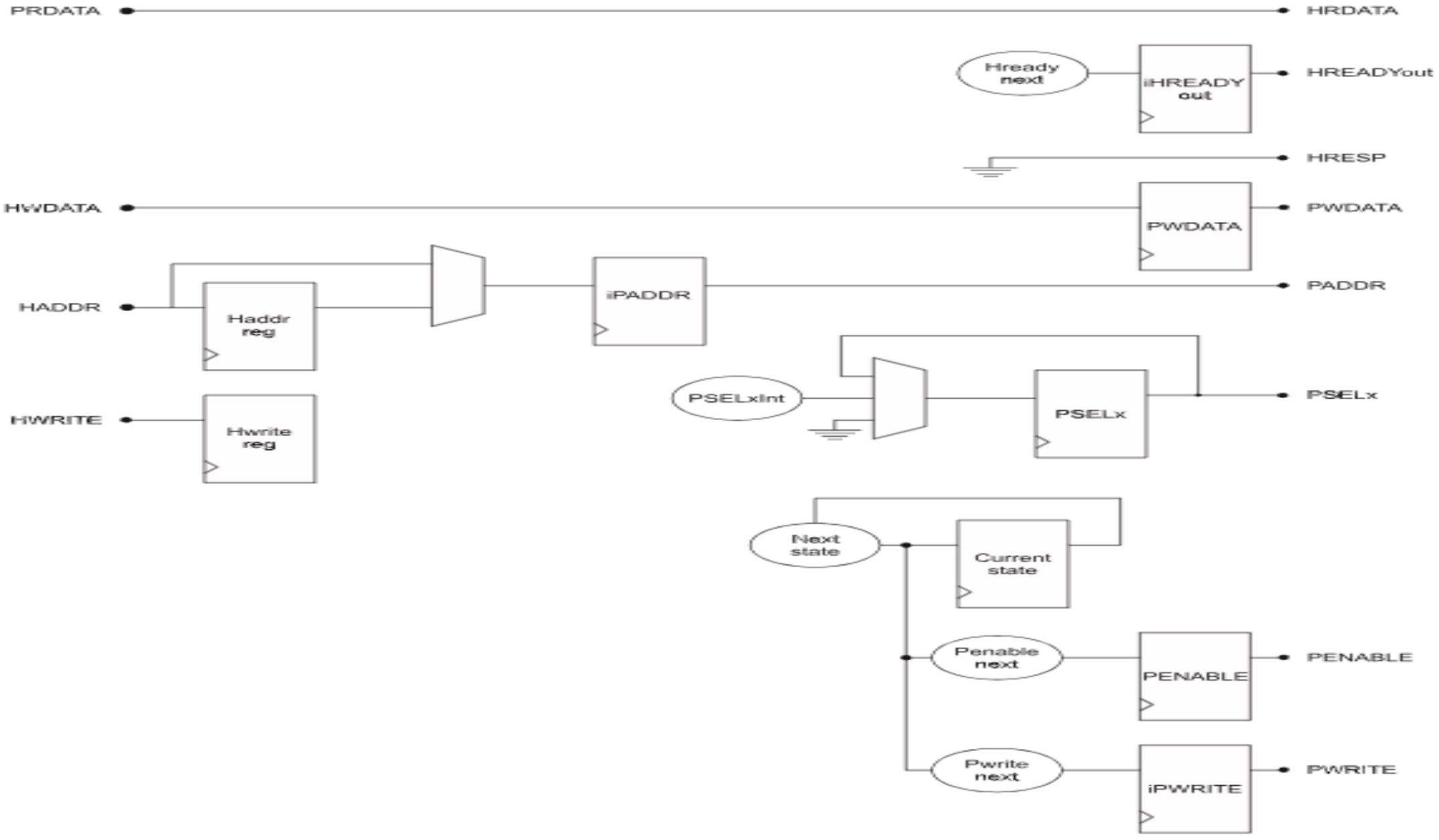

HRESETn = 0
Valid = 0
Valid = 1 and HWRITE = 1
ST_IDLE
ST_WWAIT
Valid = 1 and HWRITE = 0
Valid = 0
Valid = 1 and HWRITE = 1
Valid = 0
Valid = 1
Valid = 0
ST_READ
ST_WRITE
ST_WRITEP
Valid = 1 and HWRITE = 0
Valid = 1 and HWRITE = 0
Valid = 0
Valid = 1
Valid = 1 and HwriteReg = 1
HwriteReg = 0
Valid = 0 and HwriteReg = 1
ST_RENABLE
ST_WENABLE
ST_WENABLEP
Valid = 1 and HWRITE = 1


APB bridge module
Standard AHB slave interface
APB address decoder
State machine
Output data and address bus drivers
AHB slave output drivers
APB output drivers

KEY FUNCTIONAL BLOCKS OF THE AHB2APB BRIDGE

- AHB Master Interface: This block is responsible for receiving data and control signals from the AHB master, which initiates transactions to communicate with devices on the APB bus.
- Bridge Controller: The Bridge Controller is the core logic that manages the conversion of the AHB protocol to the APB protocol. It handles arbitration, protocol conversion, and synchronization between the two buses.
- AHB to APB Address Decoder: This block is responsible for translating the AHB address space to the APB address space.
- APB Slave Interface: The APB slave interface is responsible for receiving and transmitting data to and from the APB peripherals.
- FIFO Buffers (Optional, for Burst Handling): FIFO buffers temporarily hold data between the AHB and APB interfaces to smooth out the difference in transaction rates between the two buses.
- Control and Timing Logic: This block is responsible for managing the timing and synchronization of data transfers
- Error Handling and Status Signals: The error handling and status block is designed to handle any issues during data transfers, such as invalid addresses or write-read conflicts.

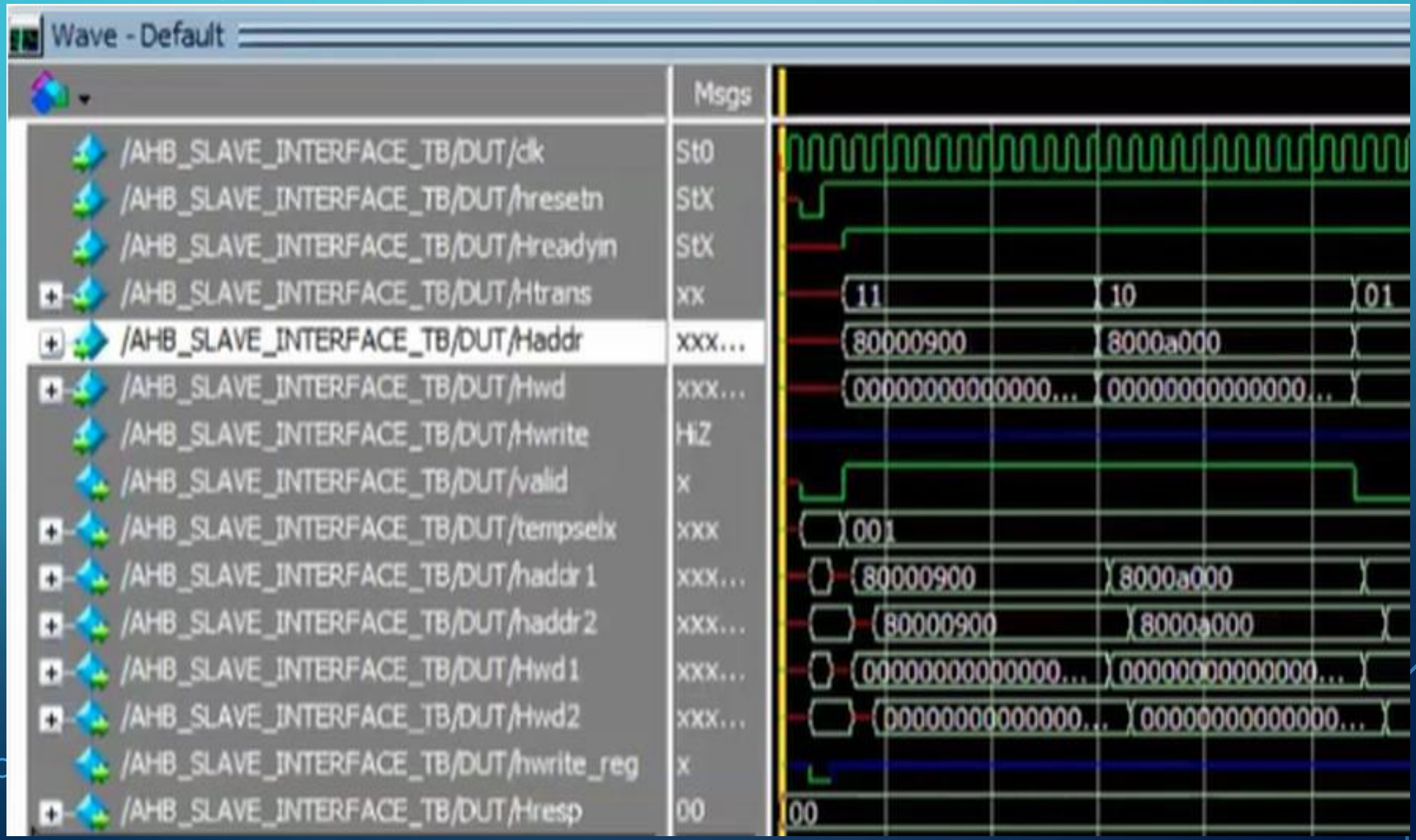
APB BRIDGE MODULE SYSTEM DIAGRAM



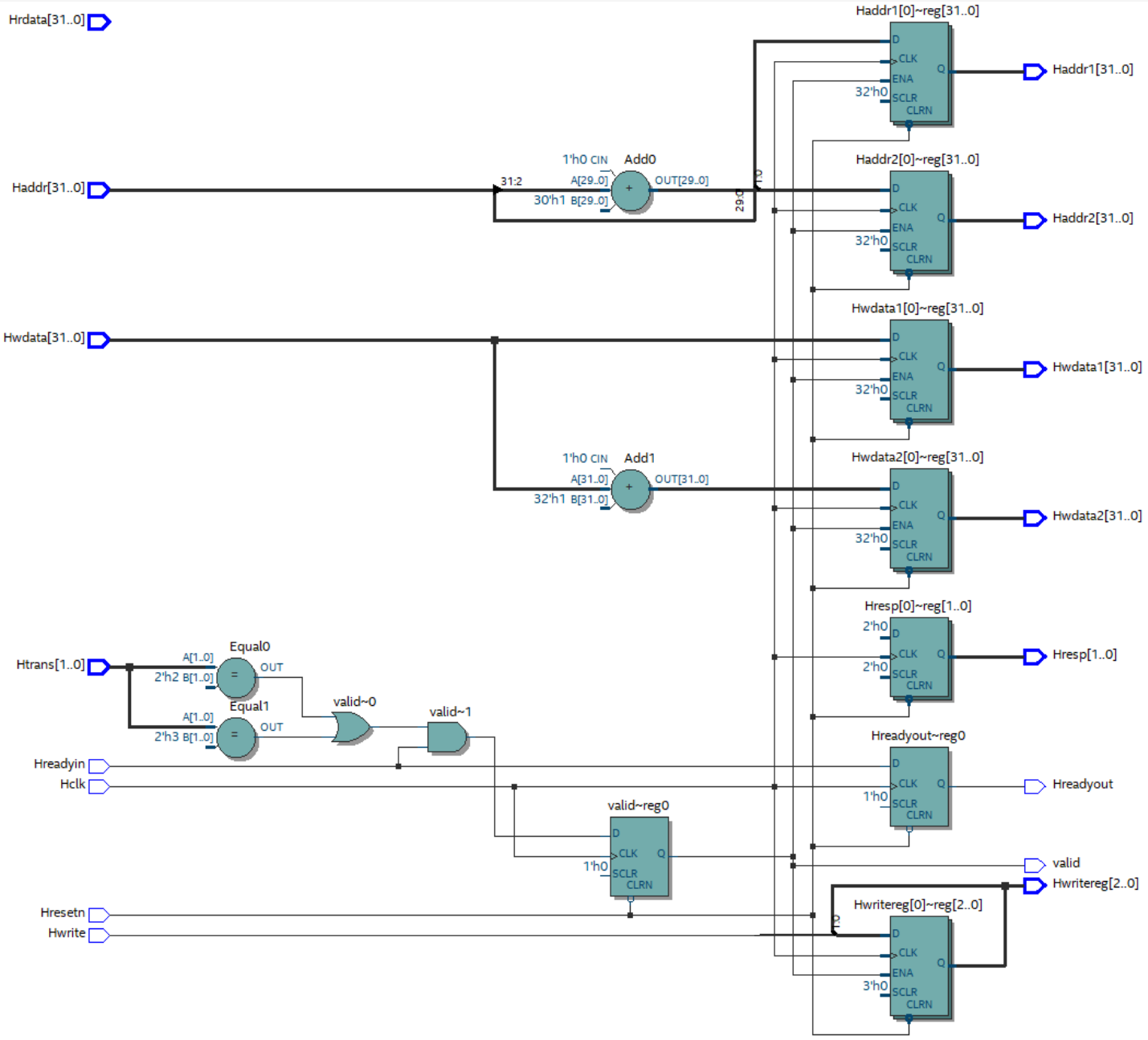
AHB SLAVE INTERFACE

- Accepts and validates AHB transactions
- Extracts address, data, and control information
- Generates valid signals for APB controller
- The AHB Slave Interface is responsible for receiving transactions from the AHB master (typically a CPU or DMA). It decodes and processes the control signals from the AHB bus (e.g., Hwrite, Htrans, Haddr, Hwdata) and prepares them for transfer to the APB controller.

WAVEFORM


Wave - Default
Msgs
/AHB_SLAVE_INTERFACE_TB/DUT/clk
/AHB_SLAVE_INTERFACE_TB/DUT/hresetn
/AHB_SLAVE_INTERFACE_TB/DUT/Hreadyin
/AHB_SLAVE_INTERFACE_TB/DUT/Htrans
/AHB_SLAVE_INTERFACE_TB/DUT/Haddr
/AHB_SLAVE_INTERFACE_TB/DUT/Hwd
/AHB_SLAVE_INTERFACE_TB/DUT/Hwrite
/AHB_SLAVE_INTERFACE_TB/DUT/valid
/AHB_SLAVE_INTERFACE_TB/DUT/tempselx
/AHB_SLAVE_INTERFACE_TB/DUT/haddr1
/AHB_SLAVE_INTERFACE_TB/DUT/haddr2
/AHB_SLAVE_INTERFACE_TB/DUT/Hwd1
/AHB_SLAVE_INTERFACE_TB/DUT/Hwd2
/AHB_SLAVE_INTERFACE_TB/DUT/hwrite_reg
/AHB_SLAVE_INTERFACE_TB/DUT/Hresp
St0
StX
StX
xx
xxx...
xxx...
HiZ
x
xxx
xxx...
xxx...
xxx...
xxx...
x
00
11
10
01
80000900
8000a000
001
00

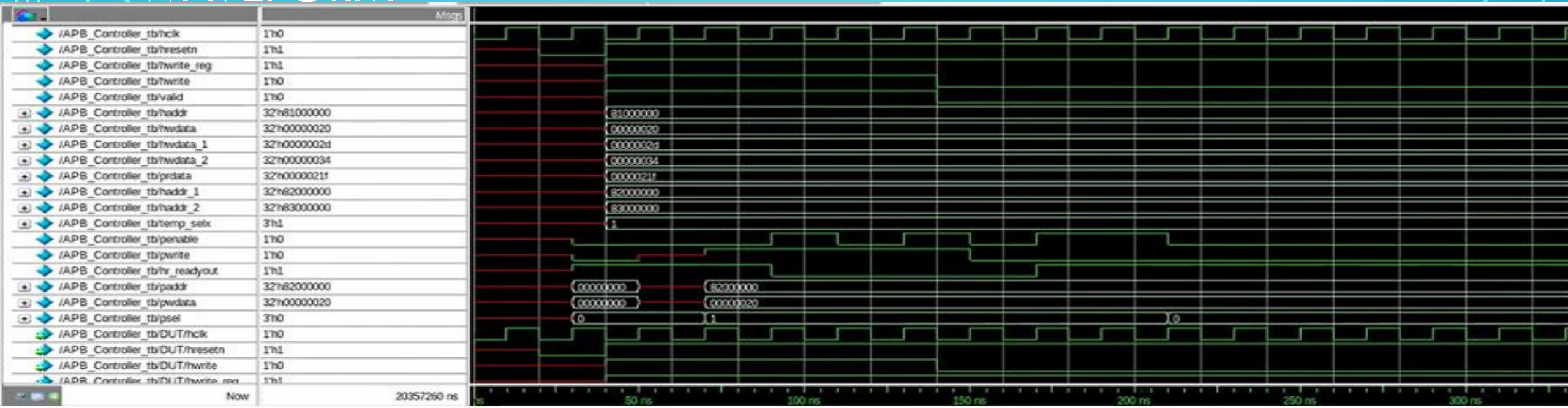
SYNTHESIS

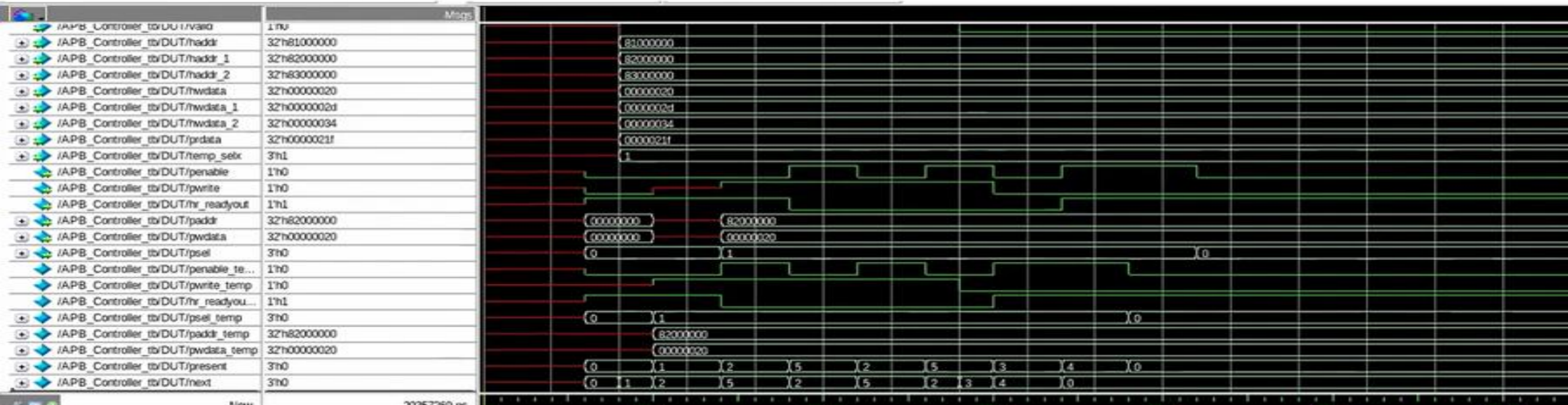

Hrdata[31..0]
Haddr1[0]~reg[31..0]
Haddr1[31..0]
Haddr[31..0]
1'h0 CIN
Add0
31:2
A[29..0]
OUT[29..0]
30'h1 B[29..0]
29:0
Haddr2[0]~reg[31..0]
Haddr2[31..0]
Hwdata[31..0]
Hwdata1[0]~reg[31..0]
Hwdata1[31..0]
1'h0 CIN
Add1
A[31..0]
OUT[31..0]
32'h1 B[31..0]
Hwdata2[0]~reg[31..0]
Hwdata2[31..0]
Hresp[0]~reg[1..0]
Hresp[1..0]
Htrans[1..0]
Equal0
A[1..0]
2'h2 B[1..0]
OUT
Equal1
2'h3 B[1..0]
valid~0
valid~1
Hreadyin
Hclk
Hreadyout~reg0
Hreadyout
valid~reg0
valid
Hwritereg[2..0]
Hresetn
Hwrite
Hwritereg[0]~reg[2..0]
D
CLK
ENA
SCLR
CLRN
Q
32'h0
2'h0
1'h0
3'h0

APB CONTROLLER

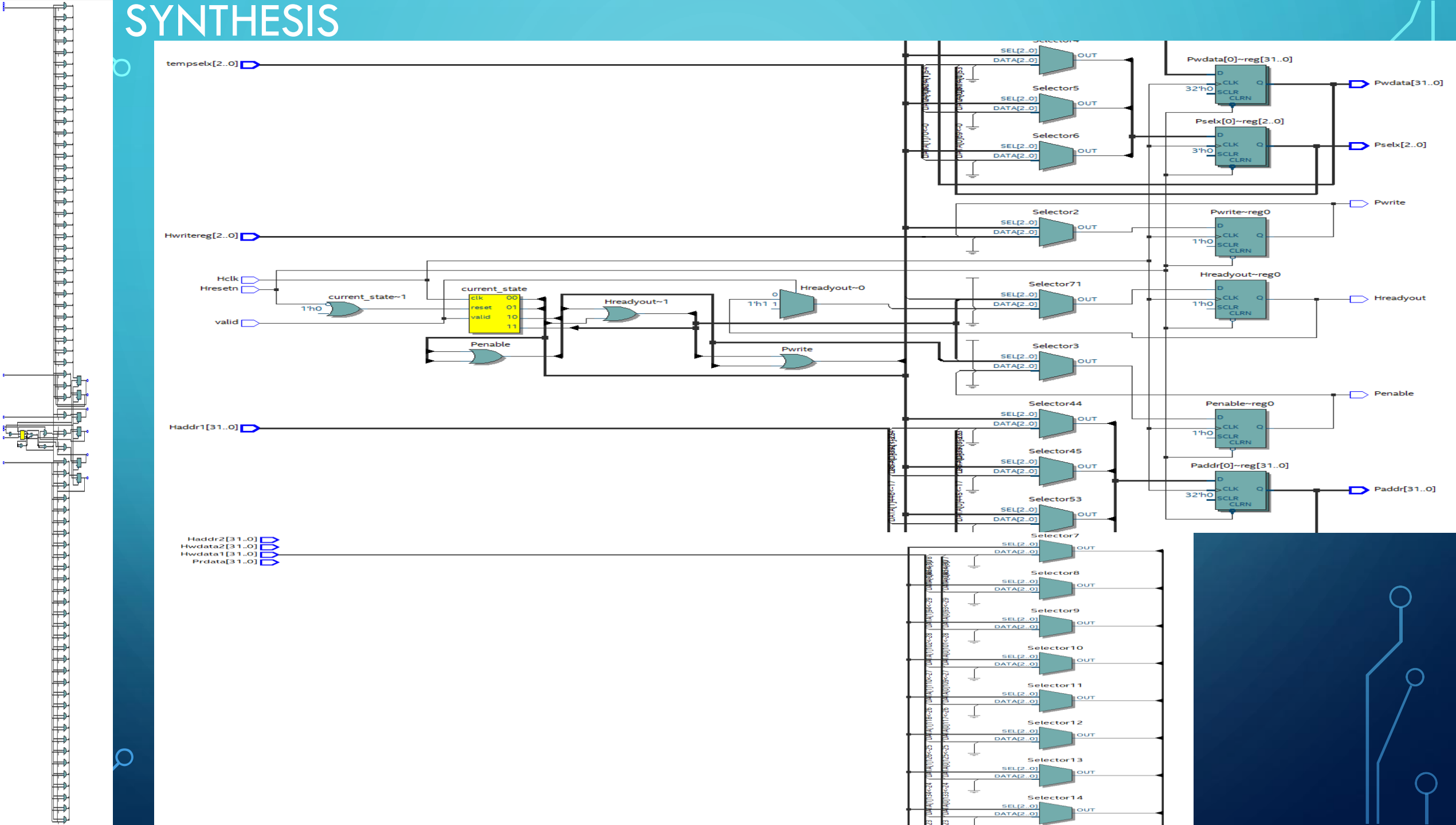
- Converts AHB transactions into APB protocol
- Generates PADDR, PWRITE, PENABLE, PSELx, and PWDATA
- Handles read/write transfers to APB peripherals
- The APB Controller is responsible for converting AHB transactions (reads and writes) into APB protocol operations. It generates the appropriate APB control signals, including Penable, Pwrite, Pselx, Paddr, and Pwdata

WAVEFORM


/APB_Controller_tb/hclk
/APB_Controller_tb/hresetn
/APB_Controller_tb/hwrite_reg
/APB_Controller_tb/hwrite
/APB_Controller_tb/valid
/APB_Controller_tb/haddr
/APB_Controller_tb/hwdata
/APB_Controller_tb/hwdata_1
/APB_Controller_tb/hwdata_2
/APB_Controller_tb/prdata
/APB_Controller_tb/haddr_1
/APB_Controller_tb/haddr_2
/APB_Controller_tb/temp_selx
/APB_Controller_tb/penable
/APB_Controller_tb/pwrite
/APB_Controller_tb/hr_readyout
/APB_Controller_tb/paddr
/APB_Controller_tb/pwdata
/APB_Controller_tb/psel
/APB_Controller_tb/DUT/hclk
/APB_Controller_tb/DUT/hresetn
/APB_Controller_tb/DUT/hwrite
1'h0
1'h1
1'h1
1'h0
1'h0
32'h81000000
32'h00000020
32'h0000002d
32'h00000034
32'h0000021f
32'h82000000
32'h83000000
3'h1
1'h0
1'h0
1'h1
32'h82000000
32'h00000020
3'h0
1'h0
1'h1
1'h0
Now
20357260 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns


/APB_Controller_tb/DUT/haddr
/APB_Controller_tb/DUT/haddr_1
/APB_Controller_tb/DUT/haddr_2
/APB_Controller_tb/DUT/hwdata
/APB_Controller_tb/DUT/hwdata_1
/APB_Controller_tb/DUT/hwdata_2
/APB_Controller_tb/DUT/prdata
/APB_Controller_tb/DUT/temp_selx
/APB_Controller_tb/DUT/penable
/APB_Controller_tb/DUT/pwrite
/APB_Controller_tb/DUT/hr_readyout
/APB_Controller_tb/DUT/paddr
/APB_Controller_tb/DUT/pwdata
/APB_Controller_tb/DUT/psel
/APB_Controller_tb/DUT/penable_te...
/APB_Controller_tb/DUT/pwrite_temp
/APB_Controller_tb/DUT/hr_readyou...
/APB_Controller_tb/DUT/psel_temp
/APB_Controller_tb/DUT/paddr_temp
/APB_Controller_tb/DUT/pwdata_temp
/APB_Controller_tb/DUT/present
/APB_Controller_tb/DUT/next
32'h81000000
32'h82000000
32'h83000000
32'h00000020
32'h0000002d
32'h00000034
32'h0000021f
3'h1
1'h0
1'h0
1'h1
32'h82000000
32'h00000020
3'h0
1'h0
1'h0
1'h1
3'h0
32'h82000000
32'h00000020
3'h0
3'h0

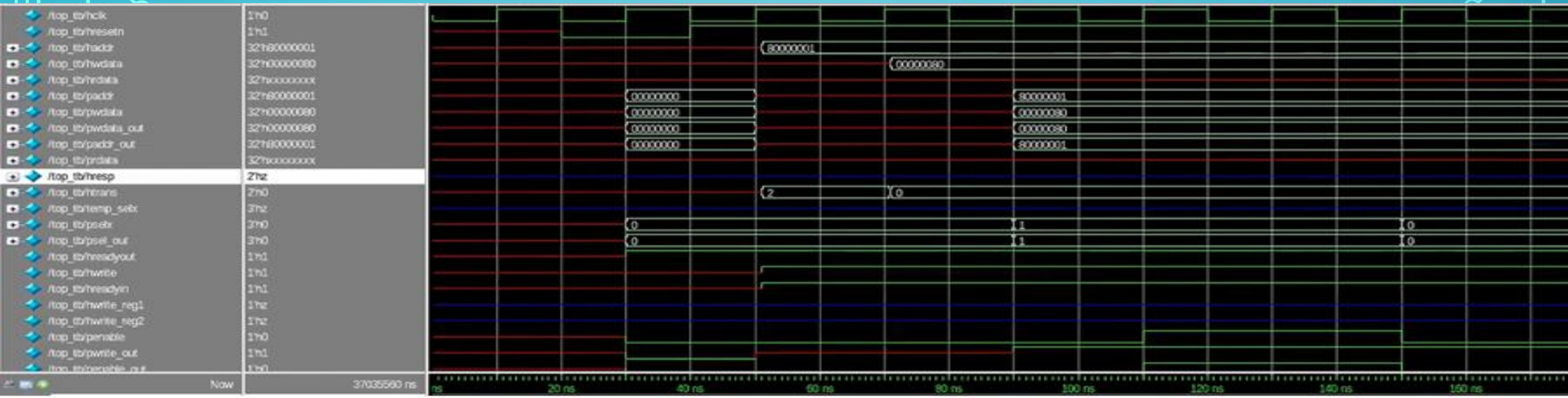
SYNTHESIS

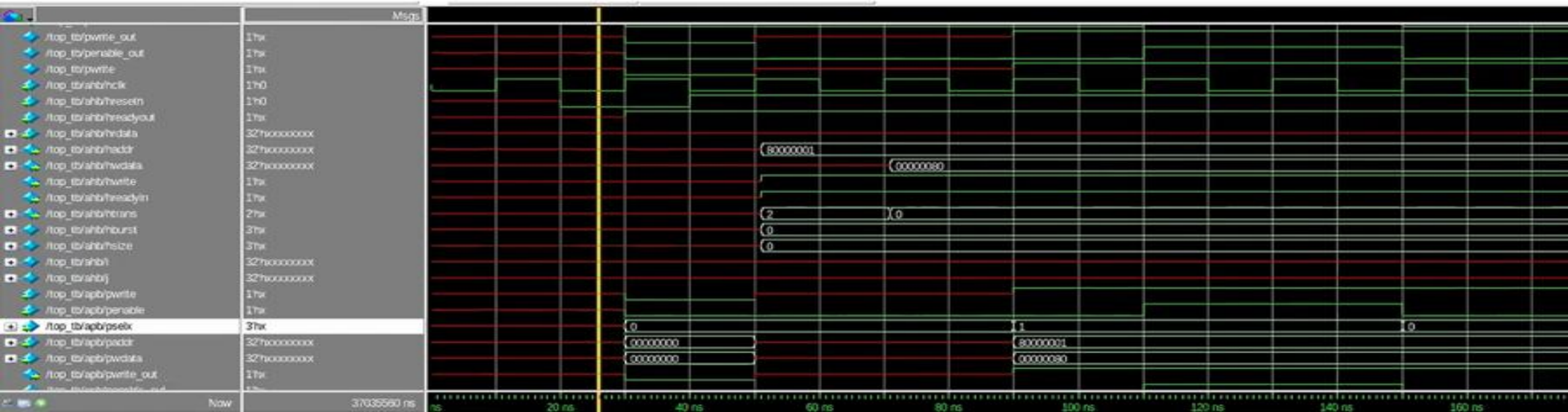

tempselx[2..0]
Hwritereg[2..0]
Hclk
Hresetn
valid
current_state~1
1'h0
current_state
clk
reset
valid
00
01
10
11
Penable
Hreadyout~1
Hreadyout~0
0
1'h1
1
Pwrite
Haddr1[31..0]
Haddr2[31..0]
Hwdata2[31..0]
Hwdata1[31..0]
Prdata[31..0]
Selector0
Selector5
Selector6
Selector2
Selector71
Selector3
Selector44
Selector45
Selector53
Selector7
Selector8
Selector9
Selector10
Selector11
Selector12
Selector13
Selector14
SEL[2..0]
DATA[2..0]
OUT
Pwdata[0]~reg[31..0]
Pwdata[31..0]
Pselx[0]~reg[2..0]
Pselx[2..0]
Pwrite~reg0
Hreadyout~reg0
Hreadyout
Penable~reg0
Penable
Paddr[0]~reg[31..0]
Paddr[31..0]
32'h0
3'h0
1'h0
D
CLK
Q
SCLR
CLRN

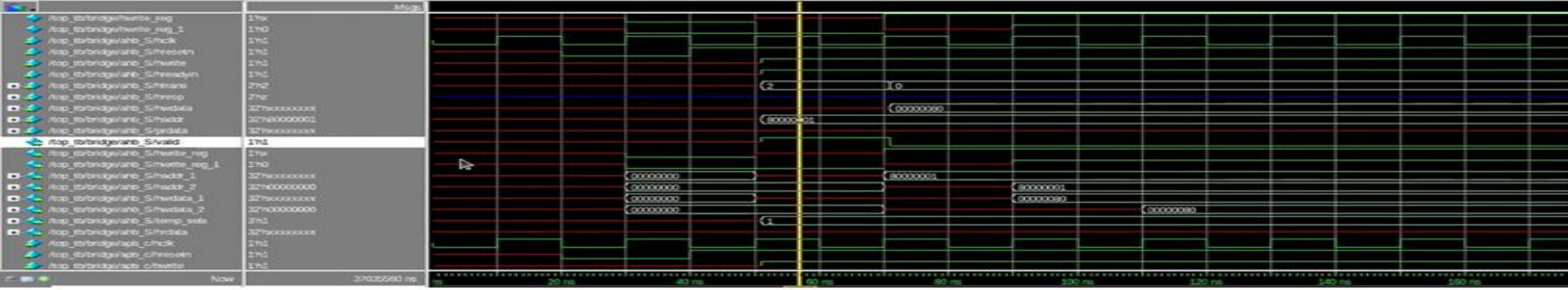
AHB2APB BRIDGE

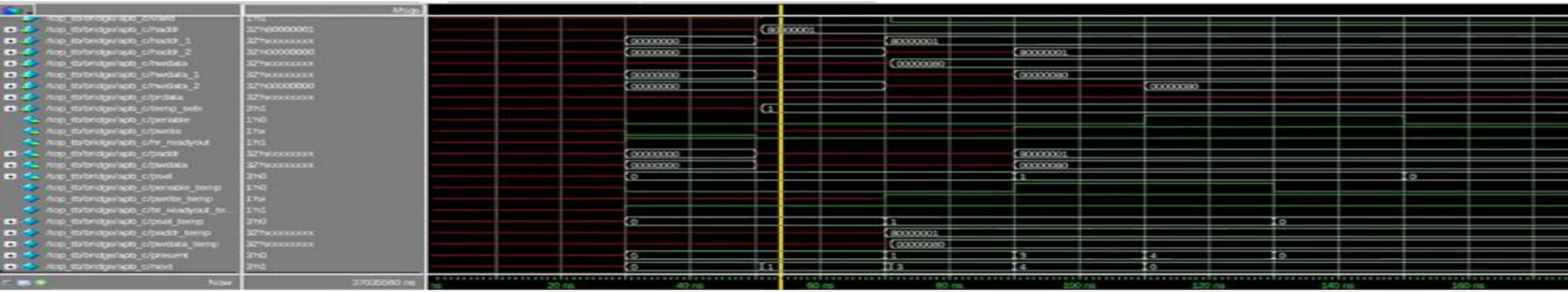
- Connects AHB Slave Interface and APB Controller
- Ensures end-to-end transaction flow between buses
- Generates final outputs and response signals
- The AHB2APB Bridge integrates the AHB Slave Interface and APB Controller, converting AHB transactions to APB transactions and vice versa. It manages the protocol conversion and the synchronization between the two different bus systems.

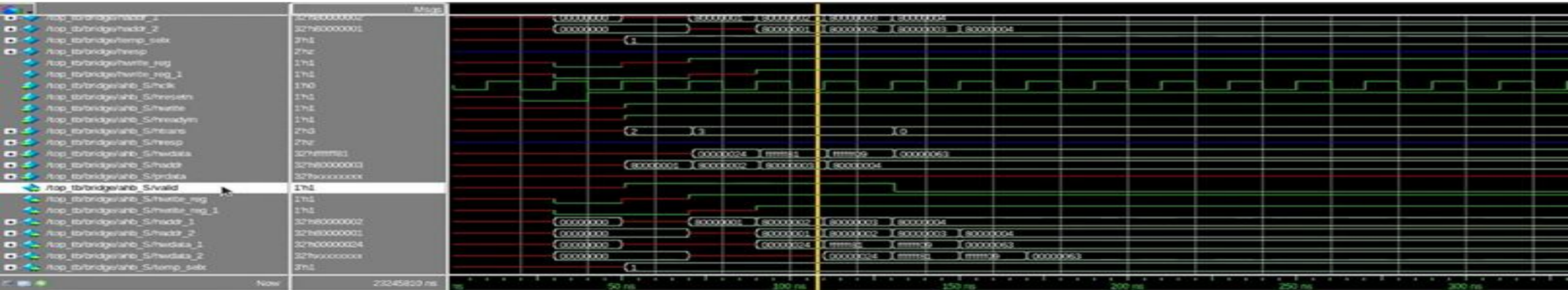
WAVEFORM


/top_tb/hclk
/top_tb/hresetn
/top_tb/haddr
/top_tb/hwdata
/top_tb/hrdata
/top_tb/paddr
/top_tb/pwdata
/top_tb/pwdata_out
/top_tb/paddr_out
/top_tb/prdata
/top_tb/hresp
/top_tb/htrans
/top_tb/temp_sebi
/top_tb/psebi
/top_tb/psel_out
/top_tb/hreadyout
/top_tb/hwrite
/top_tb/hreadyin
/top_tb/hwrite_reg1
/top_tb/hwrite_reg2
/top_tb/penable
/top_tb/pwrite_out
Now
37035560 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns

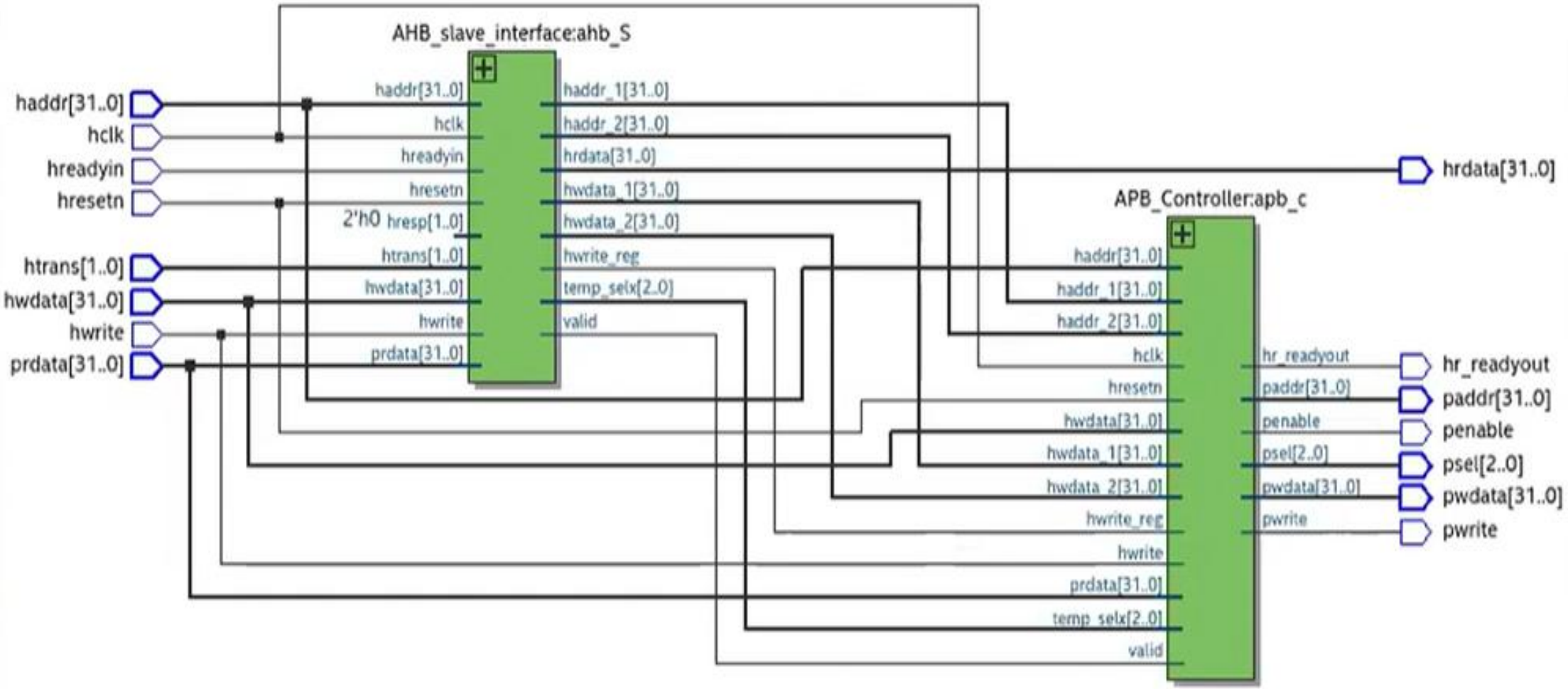

Msgs
/top_tb/pwrite_out
/top_tb/penable_out
/top_tb/pwrite
/top_tb/ahb/hclk
/top_tb/ahb/hresetn
/top_tb/ahb/hreadyout
/top_tb/ahb/hrdata
/top_tb/ahb/haddr
/top_tb/ahb/hwdata
/top_tb/ahb/hwrite
/top_tb/ahb/hreadyin
/top_tb/ahb/htrans
/top_tb/ahb/hburst
/top_tb/ahb/hsize
/top_tb/ahbi
/top_tb/ahbj
/top_tb/apb/pwrite
/top_tb/apb/penable
/top_tb/apb/pselx
/top_tb/apb/paddr
/top_tb/apb/pwdata
/top_tb/apb/pwrite_out
Now
37035560 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns


/top_tb/bridge/ahb_S/valid
/top_tb/bridge/ahb_S/hwrite_reg
/top_tb/bridge/ahb_S/hwrite_reg_1
/top_tb/bridge/ahb_S/haddr_1
/top_tb/bridge/ahb_S/haddr_2
/top_tb/bridge/ahb_S/hwdata_1
/top_tb/bridge/ahb_S/hwdata_2
/top_tb/bridge/ahb_S/temp_sel
/top_tb/bridge/ahb_S/hrdata
/top_tb/bridge/apb_c/hclk
/top_tb/bridge/apb_c/hresetn
/top_tb/bridge/apb_c/hwrite
Now
37035560 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns


/top_tb/bridge/apb_c/valid
/top_tb/bridge/apb_c/haddr
/top_tb/bridge/apb_c/haddr_1
/top_tb/bridge/apb_c/haddr_2
/top_tb/bridge/apb_c/hwdata
/top_tb/bridge/apb_c/hwdata_1
/top_tb/bridge/apb_c/hwdata_2
/top_tb/bridge/apb_c/prdata
/top_tb/bridge/apb_c/temp_sel
/top_tb/bridge/apb_c/penable
/top_tb/bridge/apb_c/pwrite
/top_tb/bridge/apb_c/hr_readyout
/top_tb/bridge/apb_c/paddr
/top_tb/bridge/apb_c/pwdata
/top_tb/bridge/apb_c/psel
/top_tb/bridge/apb_c/penable_temp
/top_tb/bridge/apb_c/pwrite_temp
/top_tb/bridge/apb_c/hr_readyout_te...
/top_tb/bridge/apb_c/psel_temp
/top_tb/bridge/apb_c/paddr_temp
/top_tb/bridge/apb_c/pwdata_temp
/top_tb/bridge/apb_c/present
/top_tb/bridge/apb_c/next
Now
37035560 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns


/top_tb/bridge/haddr_2
/top_tb/bridge/temp_sel
/top_tb/bridge/hresp
/top_tb/bridge/hwrite_reg
/top_tb/bridge/hwrite_reg_1
/top_tb/bridge/ahb_S/hclk
/top_tb/bridge/ahb_S/hresetn
/top_tb/bridge/ahb_S/hwrite
/top_tb/bridge/ahb_S/hreadyin
/top_tb/bridge/ahb_S/htrans
/top_tb/bridge/ahb_S/hresp
/top_tb/bridge/ahb_S/hwdata
/top_tb/bridge/ahb_S/haddr
/top_tb/bridge/ahb_S/prdata
/top_tb/bridge/ahb_S/valid
/top_tb/bridge/ahb_S/hwrite_reg
/top_tb/bridge/ahb_S/hwrite_reg_1
/top_tb/bridge/ahb_S/haddr_1
/top_tb/bridge/ahb_S/haddr_2
/top_tb/bridge/ahb_S/hwdata_1
/top_tb/bridge/ahb_S/hwdata_2
/top_tb/bridge/ahb_S/temp_sel
Now
23245819 ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns

AHB2APB BRIDGE



CONCLUSION AND LEARNING OUTCOMES

- Successfully designed and implemented an **AHB2APB Bridge**.
- Verified design functionality through **simulation and waveform analysis**.
- Gained hands-on experience in **AMBA protocols, RTL design, and verification**.
- The bridge efficiently connects high-speed and low-speed peripherals in SoC systems.
- Tools: Verilog (RTL Design), ModelSim (Simulation and Verification), Quartus Prime (Design Synthesis)
- The AHB2APB bridge design project has successfully achieved its primary objective of designing and implementing a bridge that interfaces between two popular on-chip communication buses: AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus). The project involved understanding the nuances of both bus protocols, designing the necessary interface components, synthesizing the design, and performing various optimizations to meet the performance and resource requirements.
- Achievements: Designed an efficient top-level AHB2APB bridge module that successfully manages the data, address, and control signal translation between AHB and APB. Implemented key features such as FIFO buffers to handle burst transfers, ensuring smooth data flow and efficient buffering during communication. Developed and verified AHB and APB interfaces, both of which were tested through simulation and waveform analysis to validate functionality.
- Challenges Faced: Understanding Protocols, Signal Synchronization, Timing and Performance Optimization, Debugging and Verification
- Optimizations Performed: FIFO Buffering, Clock Domain Crossing, Timing Optimization, Resource Optimization
- Real-World Applications of the AHB2APB Bridge: Embedded Systems, Consumer Electronics, Automotive Systems, Industrial Control Systems
- Key Learnings from the Project: In-depth Knowledge of Bus Protocols, Complexity of Bus Interfacing, Optimizing Designs for Resource and Timing, Simulation and Debugging Skills

ACKNOWLEDGMENT

I would like to express my gratitude to **Maven Silicon** for providing this learning opportunity and valuable guidance throughout the project.

aceic™ Accelerate Collaborate Excel | MAVEN SILICON Centre of Excellence in Semicon

CERTIFICATE OF COMPLETION

This is to certify that Mr./Ms. **Adhiraj Mandal** a student of **B.TECH** from **Indian Institute of Engineering Science and Technology Shibpur, West Bengal** & has successfully completed VLSI Design Internship Program from **04-12-2024** to **18-01-2025**

During the internship program with us, they worked on **AHBtoAPB Bridge RTL Design** project.

We wish them all the best for their future endeavours.

Date: **30-01-2025**

Place: **Bengaluru**

MSUID: **MS/23-24/5794**

MAVEN SILICON SOFTECH PVT. LTD. BANGALORE

SIVAKUMAR P R

FOUNDER & CEO, MAVEN SILICON

LOW POWER ML ACCELERATOR (RTL DESIGN) IN VERILOG

LOW POWER ML ACCELERATOR (RTL DESIGN) IN VERILOG

- **Objective:**
  To design a parameterized **machine learning accelerator datapath** in Verilog with **low-power techniques** (clock gating and operand gating) and verify its functionality and efficiency through RTL simulation and synthesis.
- Designed and implemented a parameterized low-power ML accelerator datapath in Verilog, featuring a pipelined MAC array, SRAM buffers, and an FSM-based controller
- Implemented low-power techniques such as clock and operand gating to minimize switching activity and dynamic power

SYSTEM ARCHITECTURE

- **MAC Array:** Performs multiply-accumulate operations for neural computations.
- **SRAM Buffer:** Temporary data storage for input/output tensors.
- **Controller (FSM):** Manages data flow, compute scheduling, and power states.
- **Top Module:** Integrates all submodules and handles start/done signaling

MAC (MULTIPLY–ACCUMULATE) UNIT

Purpose:

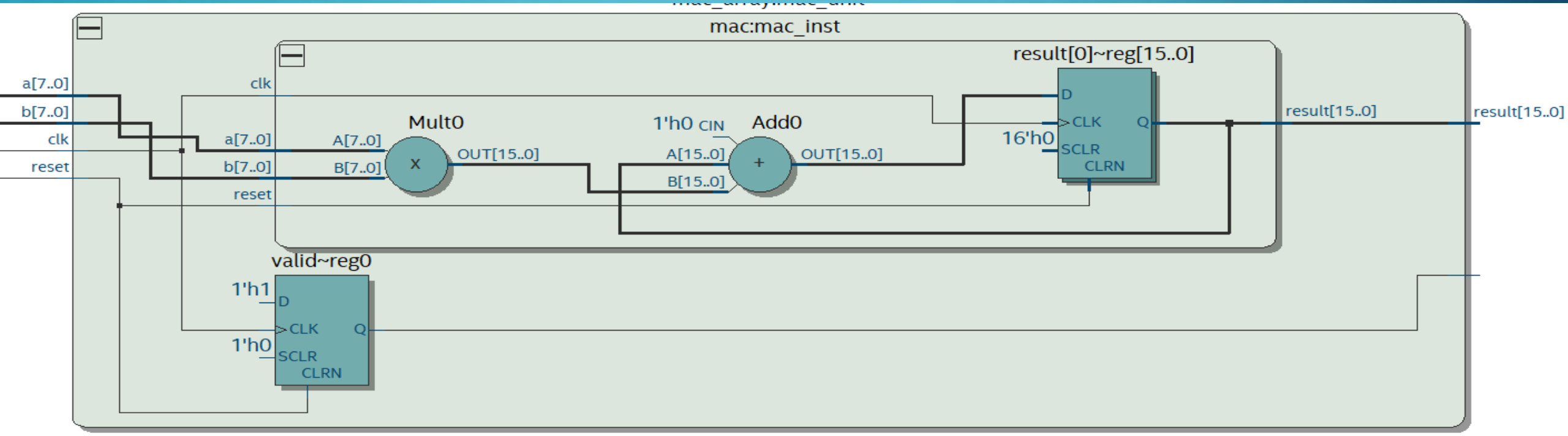
Implements the core arithmetic operation for neural computations.

Key Features:

- Parameterized bit-width for flexible precision (e.g., 8/16 bits).
- Pipelined operation for higher throughput.
- Low-power operand gating — computation disabled when inputs are zero.

Verilog Modules:

- mac.v — single MAC block
- mac_array.v — 2D array of MACs working in parallel



SRAM BUFFER

Purpose:

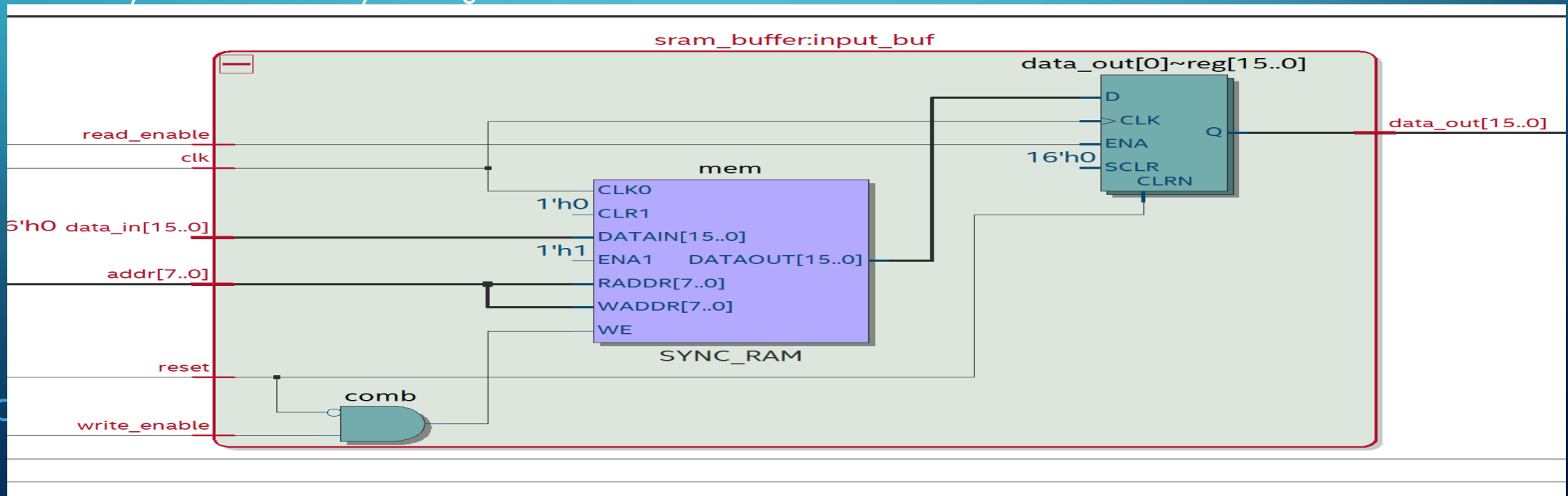
Provides local storage for inputs, weights, and intermediate outputs.

Design Details:

- Dual-port memory (read/write) interface.
- Parameterized address and data widths.
- Sequential access controlled by FSM.

Low-Power Aspects:

- Read/write only when enabled by controller.
- Idle cycles automatically clock-gated.



CONTROLLER (FSM)

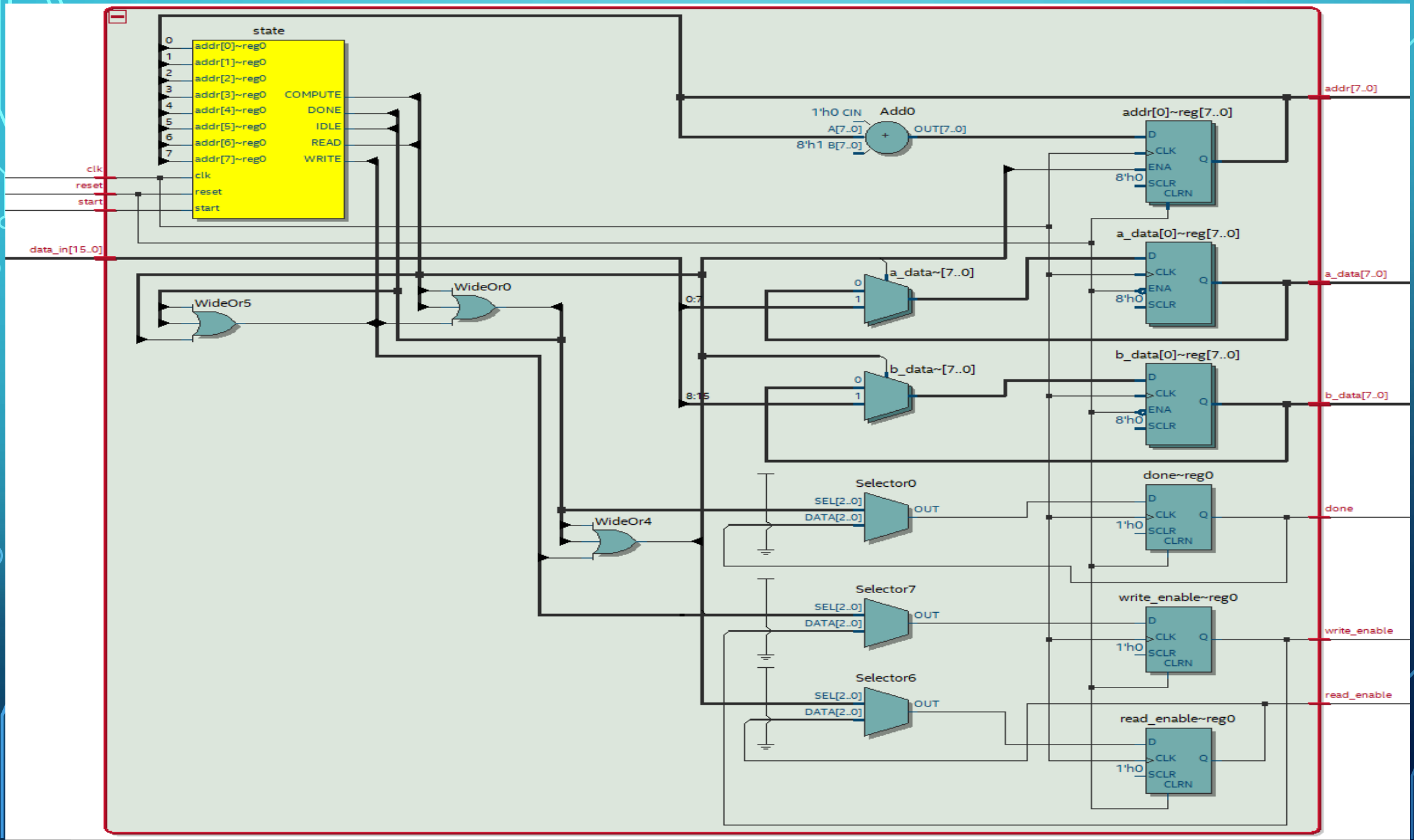
Purpose:
Coordinates all modules — manages data movement and operation phases.

FSM States:

- IDLE – Wait for start signal
- LOAD – Fetch input and weight data
- COMPUTE – Trigger MAC array computation
- WRITEBACK – Store results to SRAM

Implemented In: controller.v

- Parameterized FSM for flexibility
- Sequential control ensures proper timing between memory and MAC array
- Designed in pure Verilog for synthesis compatibility


state
addr[0]~reg0
addr[1]~reg0
addr[2]~reg0
addr[3]~reg0
addr[4]~reg0
addr[5]~reg0
addr[6]~reg0
addr[7]~reg0
clk
reset
start
COMPUTE
DONE
IDLE
READ
WRITE
clk
reset
start
data_in[15..0]
WideOr5
WideOr0
WideOr4
Add0
1'h0 CIN
A[7..0]
8'h1 B[7..0]
OUT[7..0]
addr[0]~reg[7..0]
addr[7..0]
a_data~[7..0]
a_data[0]~reg[7..0]
a_data[7..0]
b_data~[7..0]
b_data[0]~reg[7..0]
b_data[7..0]
0:7
8:15
Selector0
Selector7
Selector6
SEL[2..0]
DATA[2..0]
OUT
done~reg0
done
write_enable~reg0
write_enable
read_enable~reg0
read_enable
D
CLK
ENA
SCLR
CLRN
Q
8'h0
1'h0

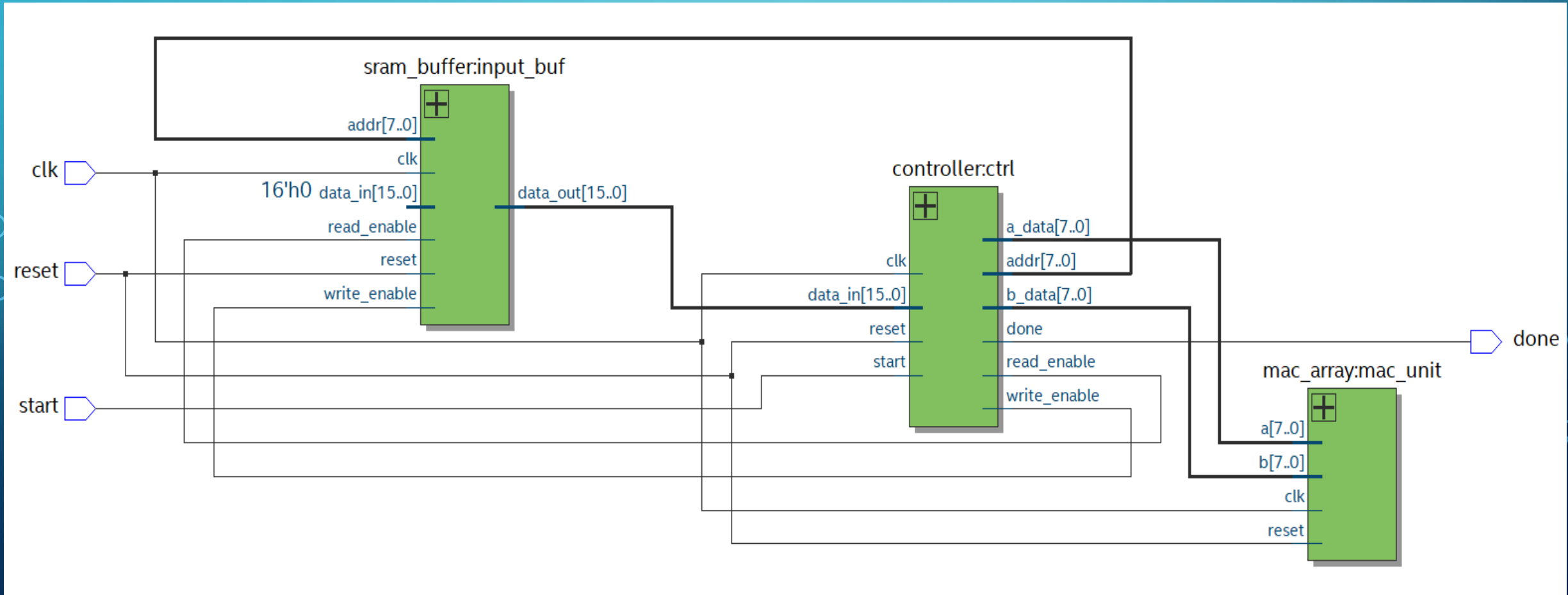
TOP MODULE INTEGRATION

Purpose:

Combines all functional blocks into a single synthesizable design.

Responsibilities:

- Instantiate and connect controller, mac_array, and sram_buffer.
- Manage clk, reset, start, and done signals.
- Provide a clean top-level interface for synthesis and testbench simulation.



LOW-POWER DESIGN TECHNIQUES

- **Clock Gating:** Disabled inactive modules' clock signals to reduce dynamic power.
- **Operand Gating:** Prevented unnecessary switching when operands are zero or idle.
- **Parameterization:** Allowed scaling of MAC size and buffer depth for trade-off analysis

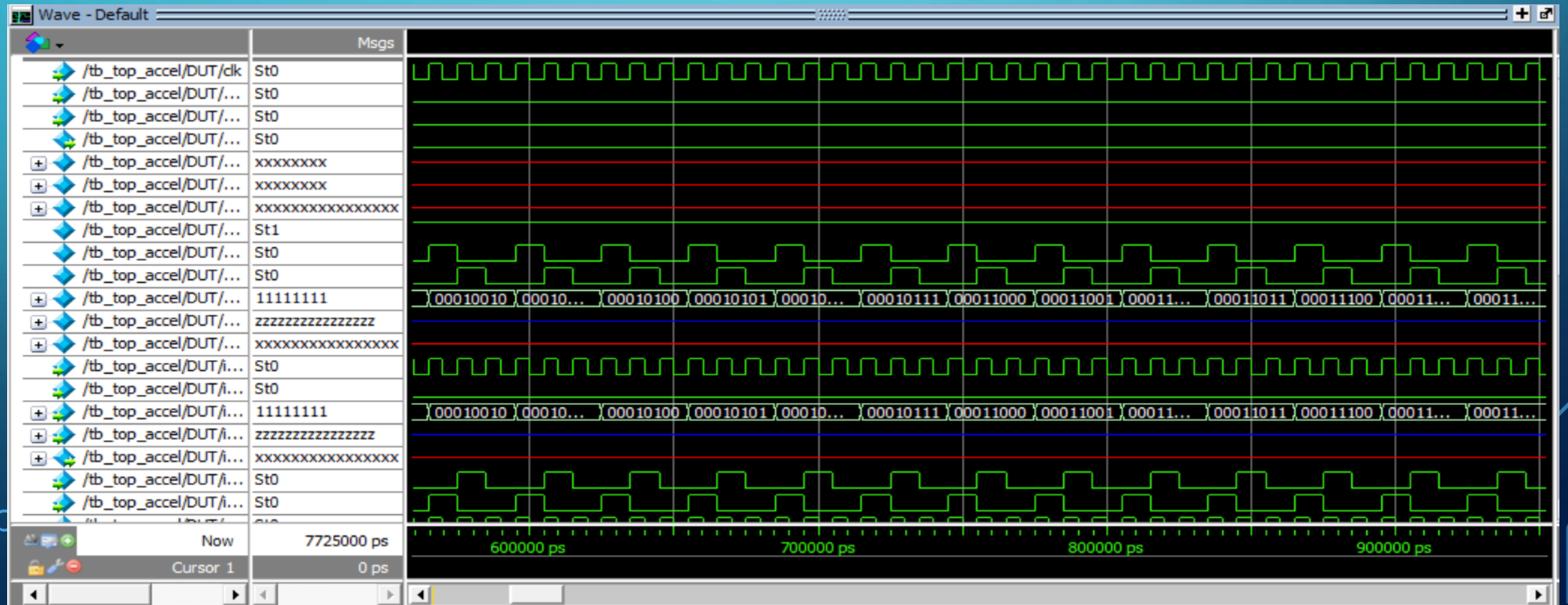
VERIFICATION & RESULTS

Verification:

- Developed a **self-checking testbench** in ModelSim (tb_top_accel.v).
- Verified correctness of dataflow and FSM sequencing.
- Monitored start/done signals and validated memory transactions.

Results:

- Design passed RTL simulation successfully.
- Synthesis reports confirmed optimized timing and reduced power through gating.



RESULTS & KEY LEARNINGS

Results:

- Functional accelerator verified successfully at RTL level.
- Achieved power reduction via gating techniques (observed through synthesis metrics).
- Implemented RTL using Verilog HDL and simulated with ModelSim for correctness.
- Designed a self-checking testbench to automate validation of start/done and output behavior.
- Synthesized design in Quartus Prime to obtain timing, resource, and power utilization.
- Performed design synthesis using Quartus Prime, and simulation/verification using a self checking testbench in ModelSim

Key Learnings:

- RTL design and FSM-based control for ML accelerators.
- Integrating low-power concepts at architectural level.
- Practical flow of simulation → synthesis → analysis.

LOW POWER UART AND TIMER IP BLOCK FOR SOC INTEGRATION IN VERILOG

LOW POWER UART AND TIMER IP BLOCK FOR SOC INTEGRATION IN VERILOG

Objective:
To design and verify low-power UART and Timer IP cores with SoC integration support, focusing on efficient communication and timing modules.

Key Features:

- Synthesizable Verilog implementation of UART and Timer
- Memory-mapped registers for software access
- Interrupt (IRQ) generation and clock gating for low power
- SoC wrapper for seamless integration
- Verified using self-checking testbenches in ModelSim
- Designed and verified a low-power UART (baud generator, TX/RX) and a programmable Timer IP in synthesizable Verilog for SoC integration
- Implemented memory-mapped control registers, interrupt (IRQ) lines, clock-gating, a SoC wrapper, and comprehensive self-checking testbenches

UART IP DESIGN

Modules Implemented:

- **Baud Rate Generator:** Generates accurate clock pulses for UART communication.
- **Transmitter (TX):** Serializes parallel data for transmission.
- **Receiver (RX):** Deserializes incoming serial data.

Key Highlights:

- Parameterizable baud rates
- Start/stop bit and parity support
- Low-power operation with clock gating

UART BAUD GENERATOR

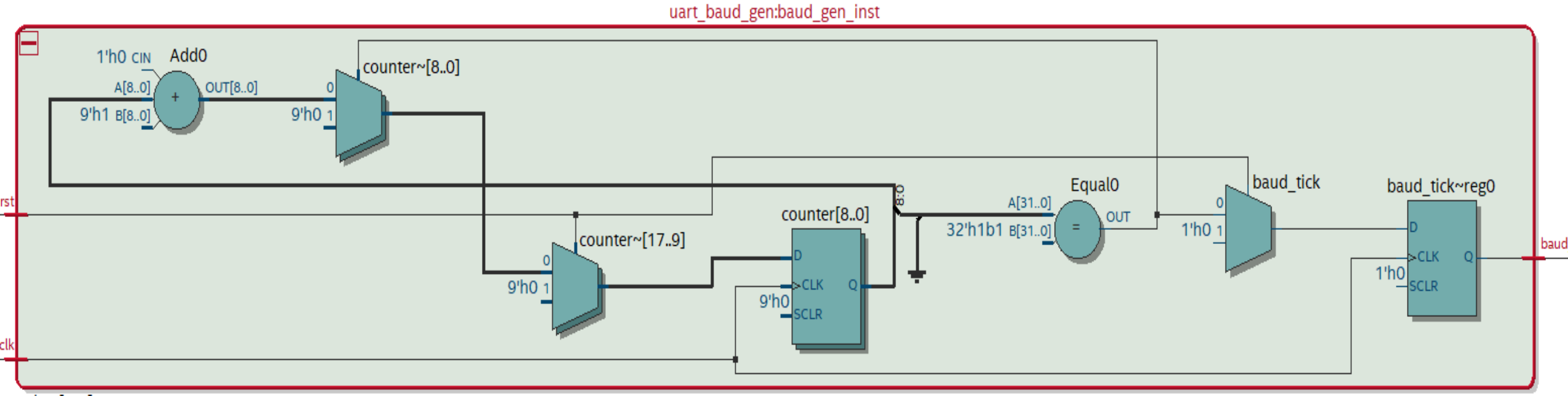
Function:
Generates baud rate clock pulses derived from the system clock to synchronize UART TX and RX.

Features:

- Configurable baud divisor
- Low power using clock enable logic
- Stable timing for data transmission

Verification:

- Simulated at different baud rates
- Verified pulse timing accuracy



UART TRANSMITTER (TX)

Function:

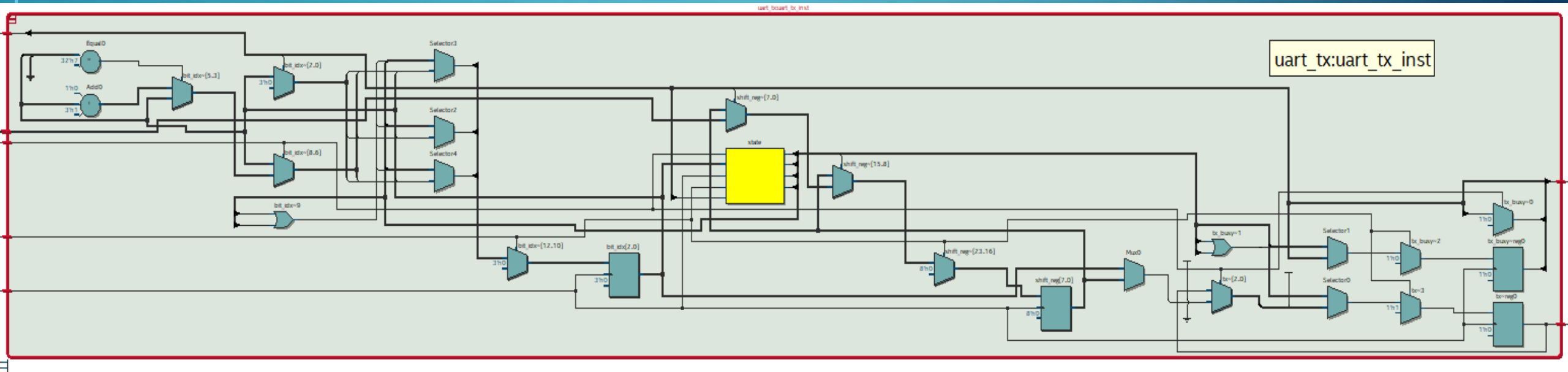
Converts parallel data into serial bitstream for transmission.

Key Features:

- Adds start and stop bits
- Shifts data bits out at baud clock edges
- TX Busy signal for status monitoring

Verification:

- Checked for correct bit framing
- Verified data sequence and timing



UART RECEIVER (RX)

Function:

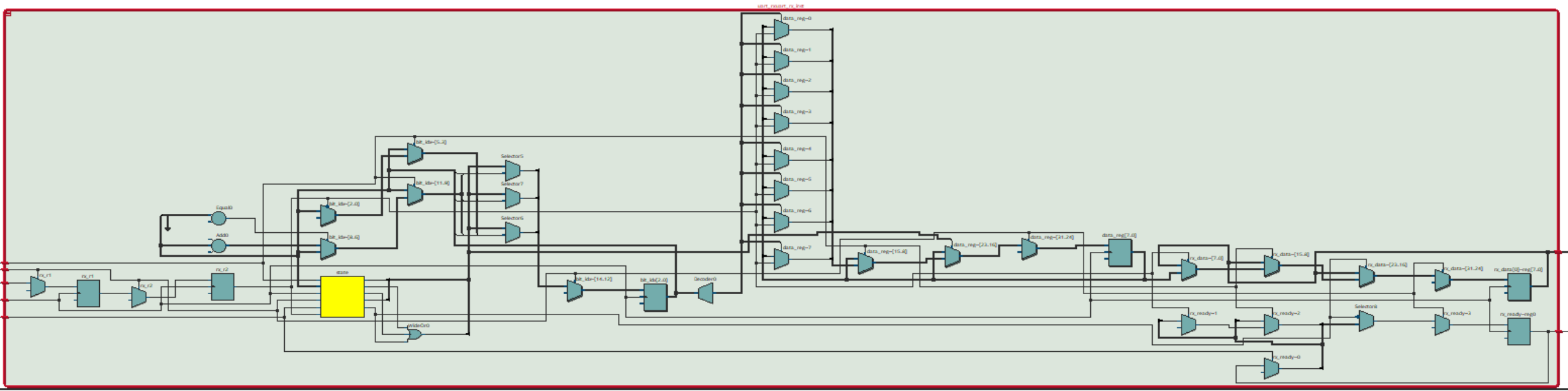
Converts incoming serial data into parallel form for CPU read.

Key Features:

- Detects start bit automatically
- Samples data bits at baud intervals
- Sets RX Done interrupt flag

Verification:

- Tested under varying baud mismatch
- Validated stop-bit detection accuracy



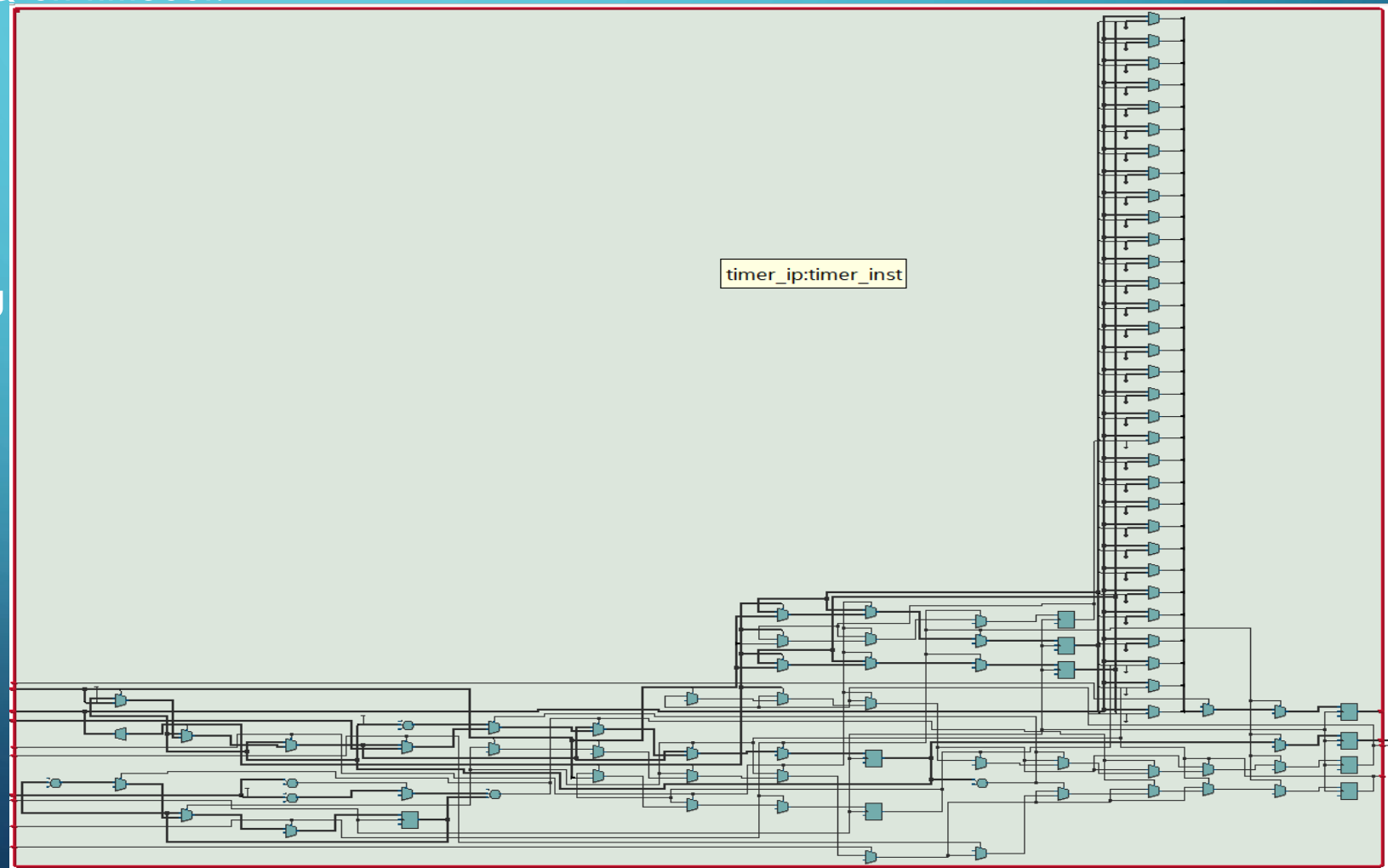
TIMER IP DESIGN

Modules Implemented:

- **Programmable Counter:** Configurable for up/down counting.
- **Control and Status Registers:** Memory-mapped interface.
- **Interrupt Generation:** Triggers IRQ on timeout.

Key Highlights:

- Variable timer period
- Clock enable for power efficiency
- Clean reset and interrupt handling



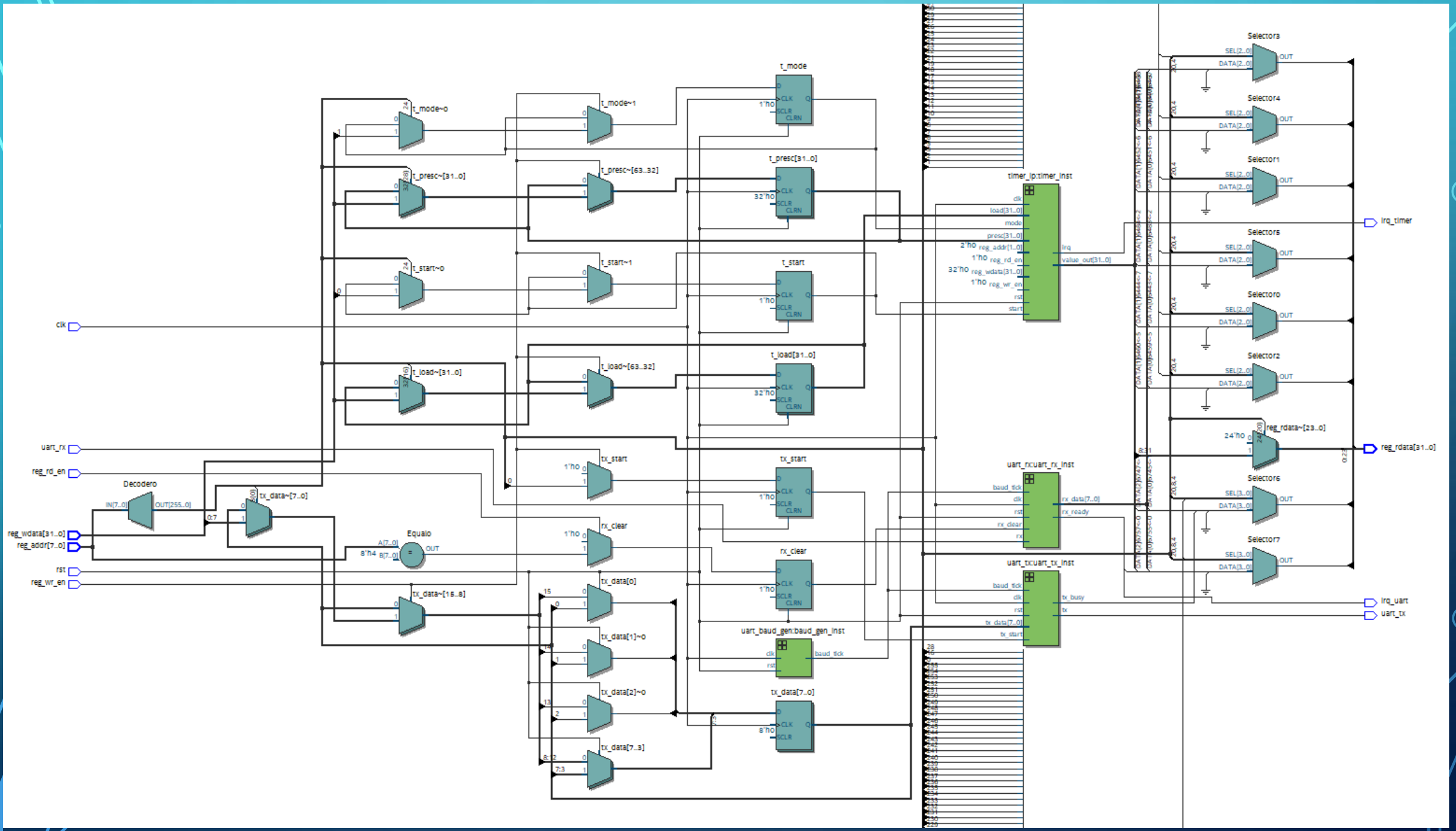
SOC INTEGRATION

SoC Wrapper Components:

- Integrated UART and Timer modules
- Memory-mapped register interface
- Shared system clock and reset
- Interrupt lines routed to CPU

Integration Goals:

✅ Easy software access through address mapping
✅ Reusable IPs for larger SoC designs
✅ Low-power operation


clk
uart_rx
reg_rd_en
reg_wdata[31..0]
reg_addr[7..0]
rst
reg_wr_en
Decodero
IN[7..0]
OUT[255..0]
Equalo
A[7..0]
B[7..0]
8'h4
OUT
t_mode~0
t_mode~1
t_mode
t_presc~[31..0]
t_presc~[63..32]
t_presc[31..0]
t_start~0
t_start~1
t_start
t_load~[31..0]
t_load~[63..32]
t_load[31..0]
tx_start
tx_data~[7..0]
tx_data~[15..8]
rx_clear
tx_data[0]
tx_data[1]~0
tx_data[2]~0
tx_data[7..3]
tx_data[7..0]
uart_baud_gen:baud_gen_inst
timer_ip:timer_inst
uart_rx:uart_rx_inst
uart_tx:uart_tx_inst
Selector3
Selector4
Selector1
Selector5
Selector0
Selector2
Selector6
Selector7
irq_timer
irq_uart
uart_tx
reg_rdata~[23..0]
reg_rdata[31..0]

VERIFICATION AND RESULTS

- **Verification Approach:**
- Self-checking testbenches
- Functional and timing simulation using **ModelSim**
- RTL synthesis using **Quartus Prime**
- **Results:**
- 100% functional coverage for UART and Timer test cases
- Verified interrupt and memory-mapped communication
- Successfully synthesized with no timing violations


Wave - Default
Msgs
/tb_soc_uart_timer/clk
/tb_soc_uart_timer/...
Now
1901470000 ps
Cursor 1
0 ps
100000000 ps
200000000 ps
300000000 ps

TOOLS & TECHNOLOGIES

- **HDL:** Verilog
- **Simulation:** ModelSim
- **Synthesis:** Quartus Prime
- **Design Methodology:** RTL → Testbench → Synthesis → Verification
- Performed design synthesis using Quartus Prime, and simulation/verification using ModelSim

KEY LEARNINGS

- RTL design flow and modular IP development
- Memory-mapped I/O and SoC integration
- Low-power design techniques (clock gating)
- Writing self-checking testbenches
- Verification workflow in industry tools

CONCLUSION

Outcome:

Designed, verified, and synthesized two SoC-compatible IP cores (UART + Timer) with low-power and interrupt capabilities.

Future Work:

- Add APB/AHB bus interface for advanced SoC integration
- Extend testbench for constrained-random verification (SystemVerilog/UVM)

AI-POWERED HDL GENERATION TOOL FOR ASIC/FPGA RTL DESIGN & TESTBENCH

AI-POWERED HDL GENERATION TOOL FOR ASIC/FPGA RTL DESIGN & TESTBENCH

- Developed an AI-powered Verilog generation tool that converts natural-language specs or JSON specs into synthesizable, parameterized RTL and self-checking testbenches using LLMs, prompt engineering, and heuristic post-processing to produce download-ready Verilog files
- The tool accelerates front-end design workflows for ASIC/FPGA development.

MOTIVATION / PROBLEM STATEMENT

Problem:

- RTL and testbench creation is time-consuming and repetitive.
- Engineers often manually translate written specs into Verilog modules.
- Need for automation to speed up design prototyping and reduce human error.

Goal:

Automate Verilog RTL and testbench generation directly from textual or file-based specifications.

PROPOSED SOLUTION

- Used an AI-based code generation pipeline leveraging large language models.
- Input: Natural-language or JSON specification.
- Output: Synthesizable Verilog RTL and its corresponding testbench.
- Integrated heuristic post-processing to ensure clean, synthesizable HDL output.
- Built an interactive front-end using Streamlit for user-friendly prompt/file-based input and code download.

SYSTEM ARCHITECTURE

- User Prompt / Spec File → LLM Backend → RTL & TB Generation → Output Display & Download

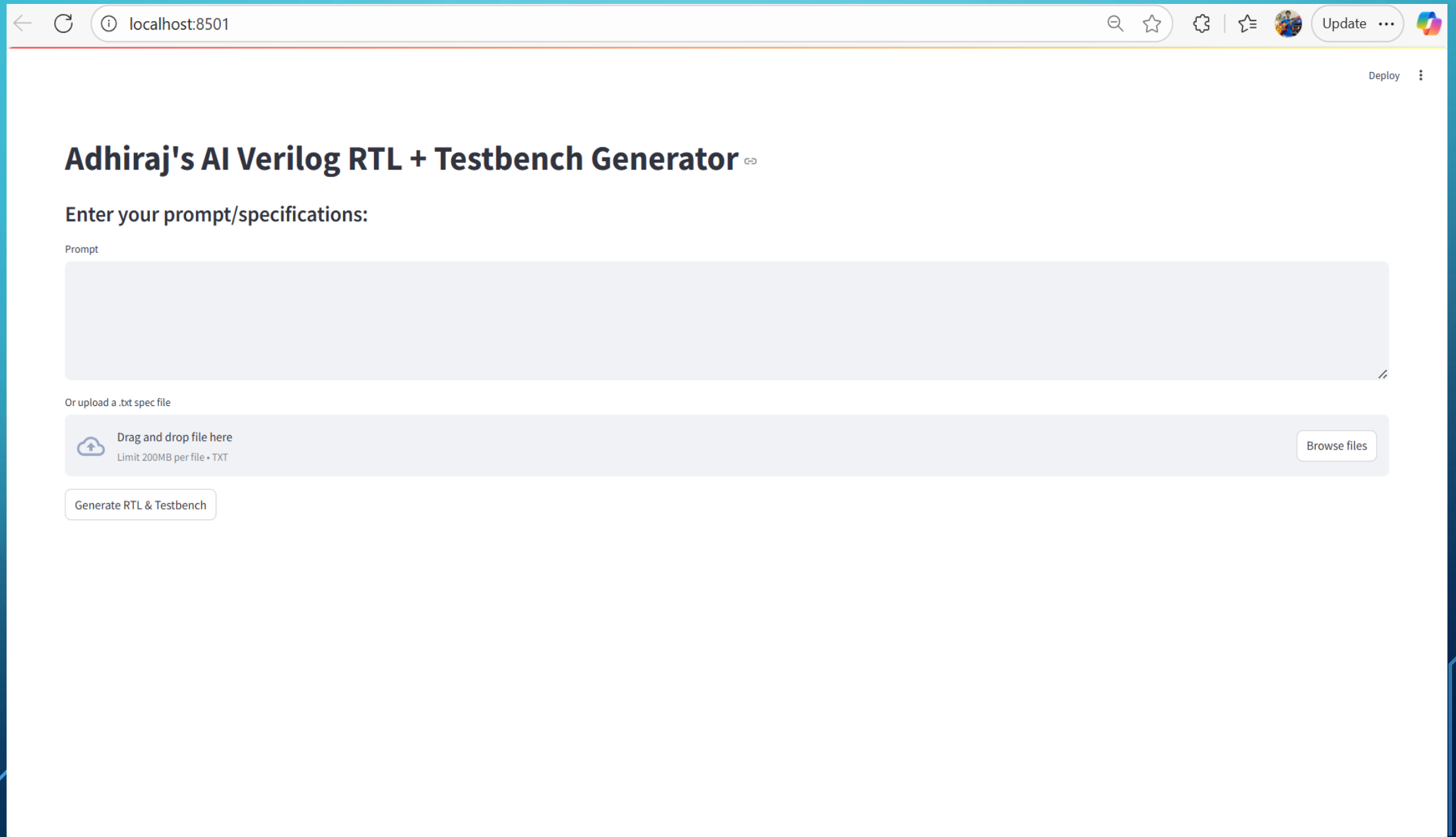
Key Components:

- **Frontend:** Streamlit-based interface for input & output visualization
- **Backend:** Python module integrating OpenRouter/OpenAI LLM API
- **Code Modules:** RTL generator, testbench generator, file manager

DEMONSTRATION AND RESULTS / OUTPUT EXAMPLES

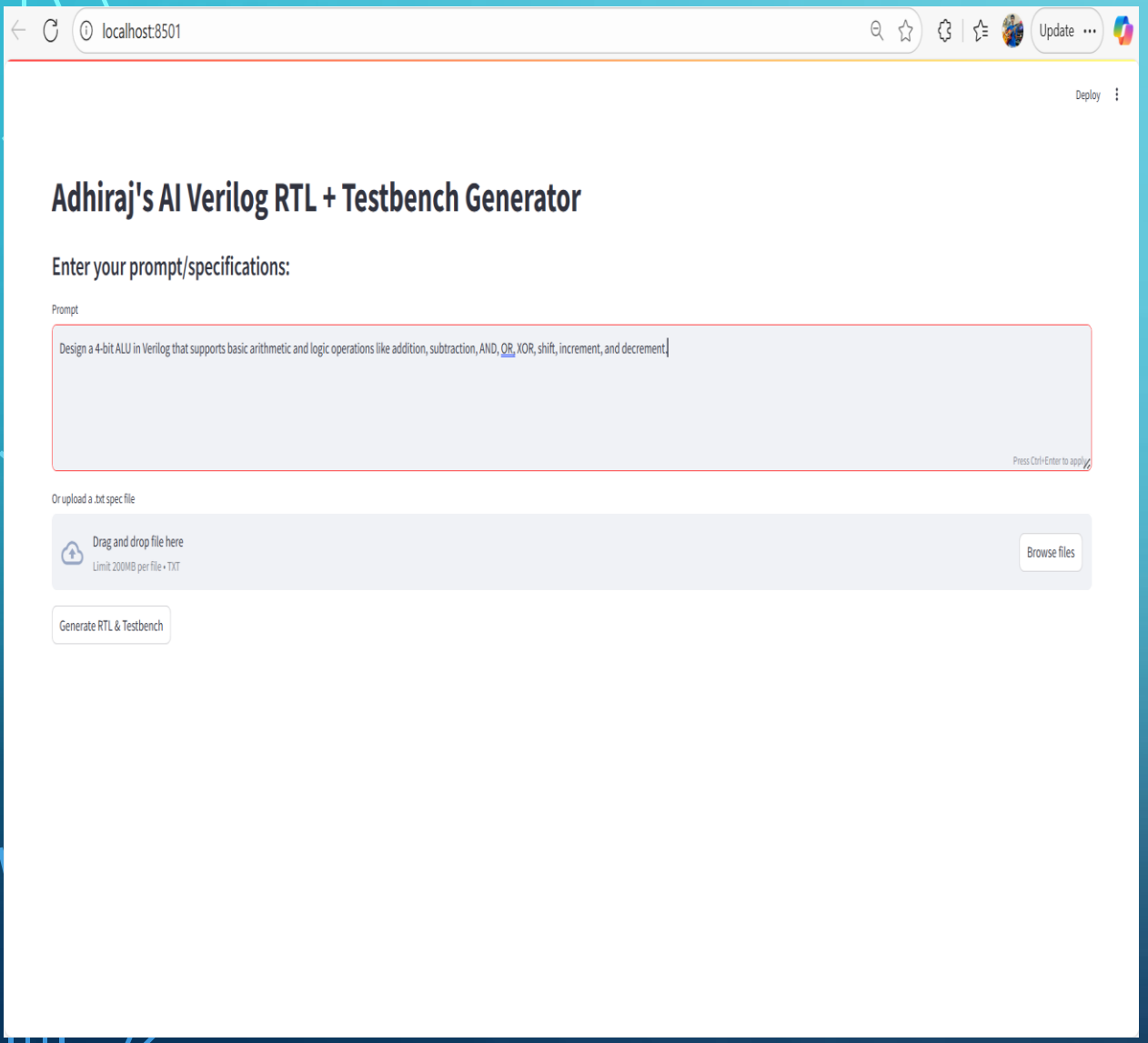
- **Examples:**
- 4-bit ALU
- D Flip-Flop
- Counter (8 bit up/down Counter)
- FIFO (8x8 FIFO)
- AXI4-Lite Memory Mapped Timer Peripheral (using spec file)
- **Outcome:**
  Generated Verilog modules and testbenches that are synthesizable and simulate correctly

USER INTERFACE

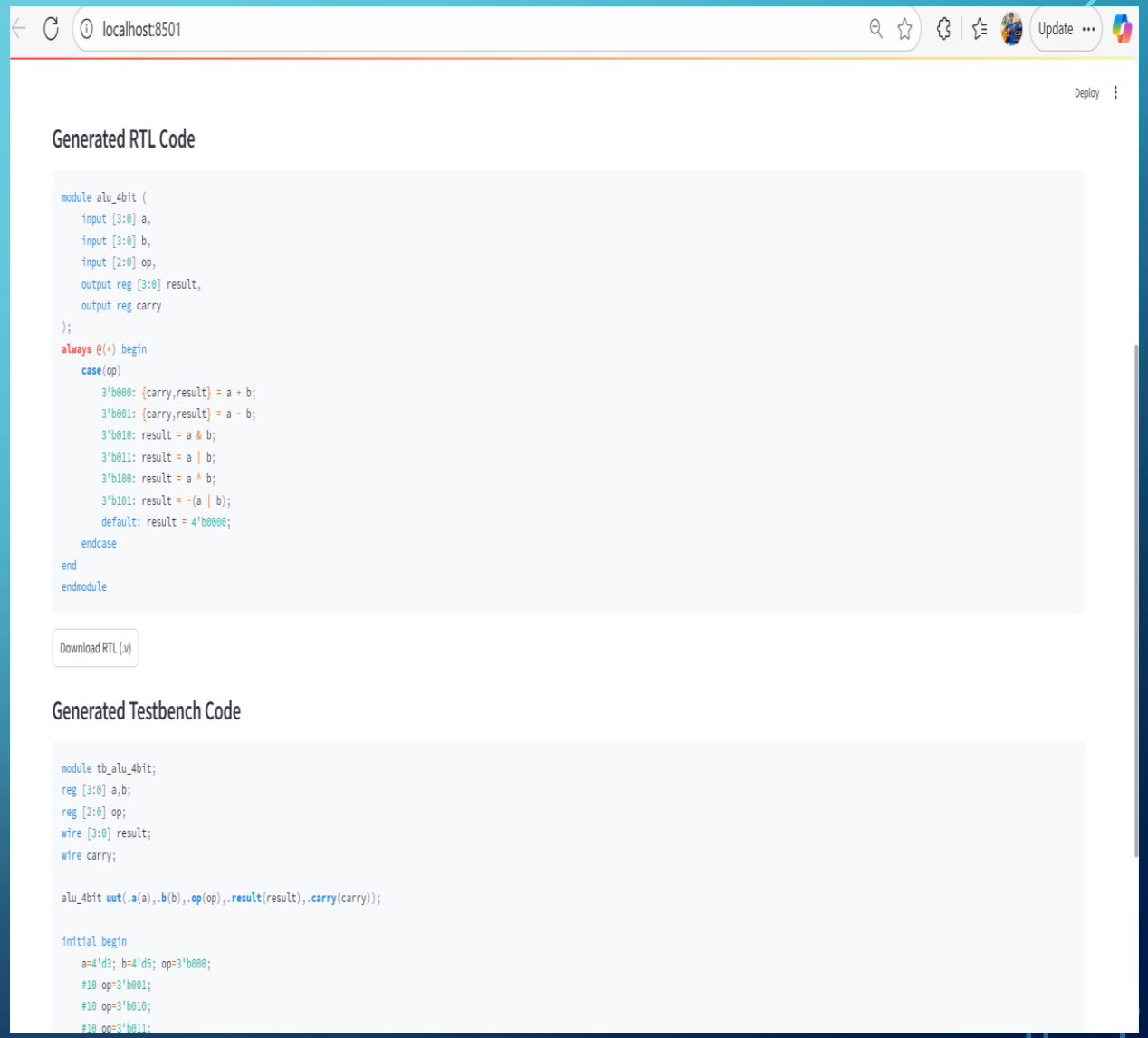


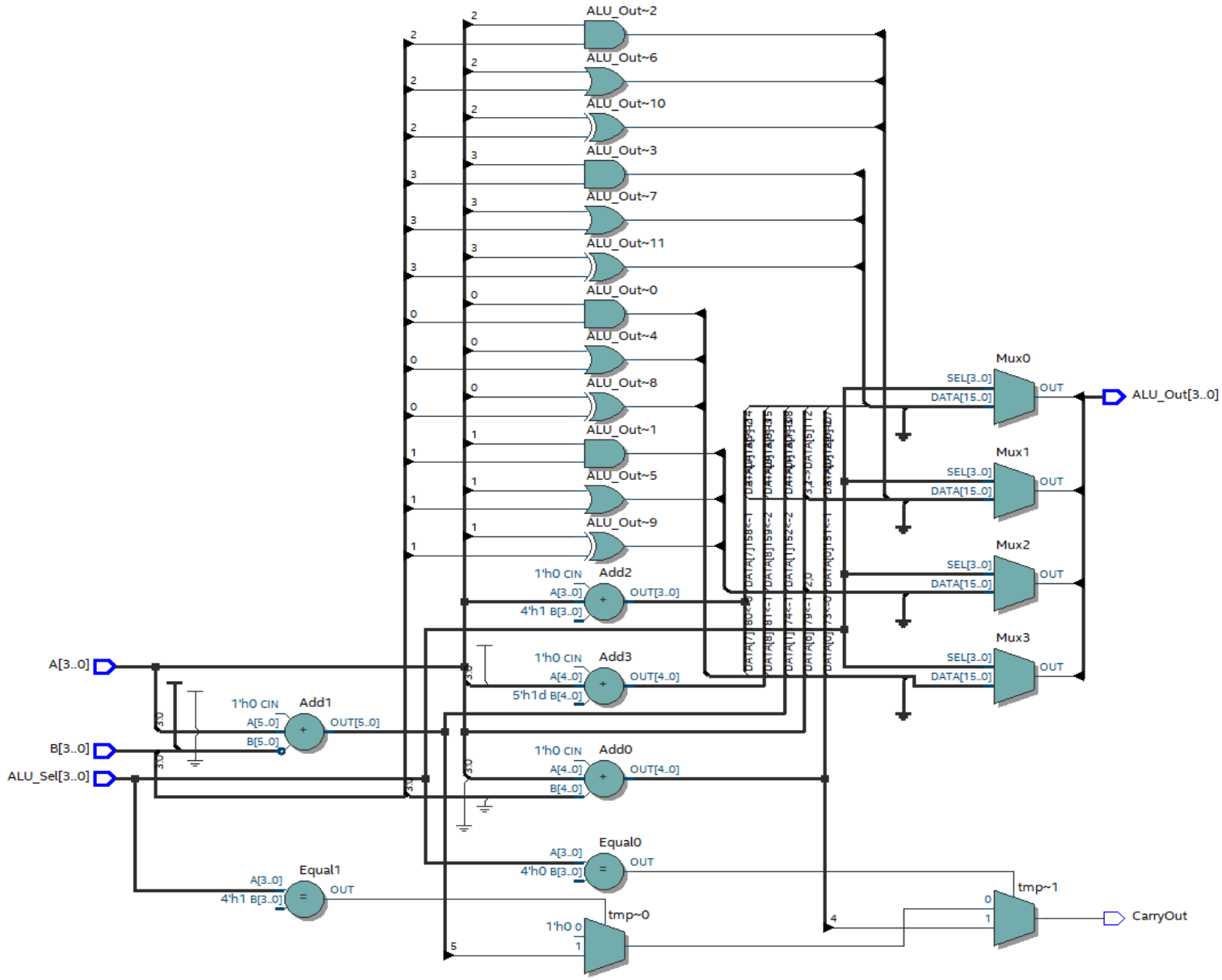
ALU

INPUT



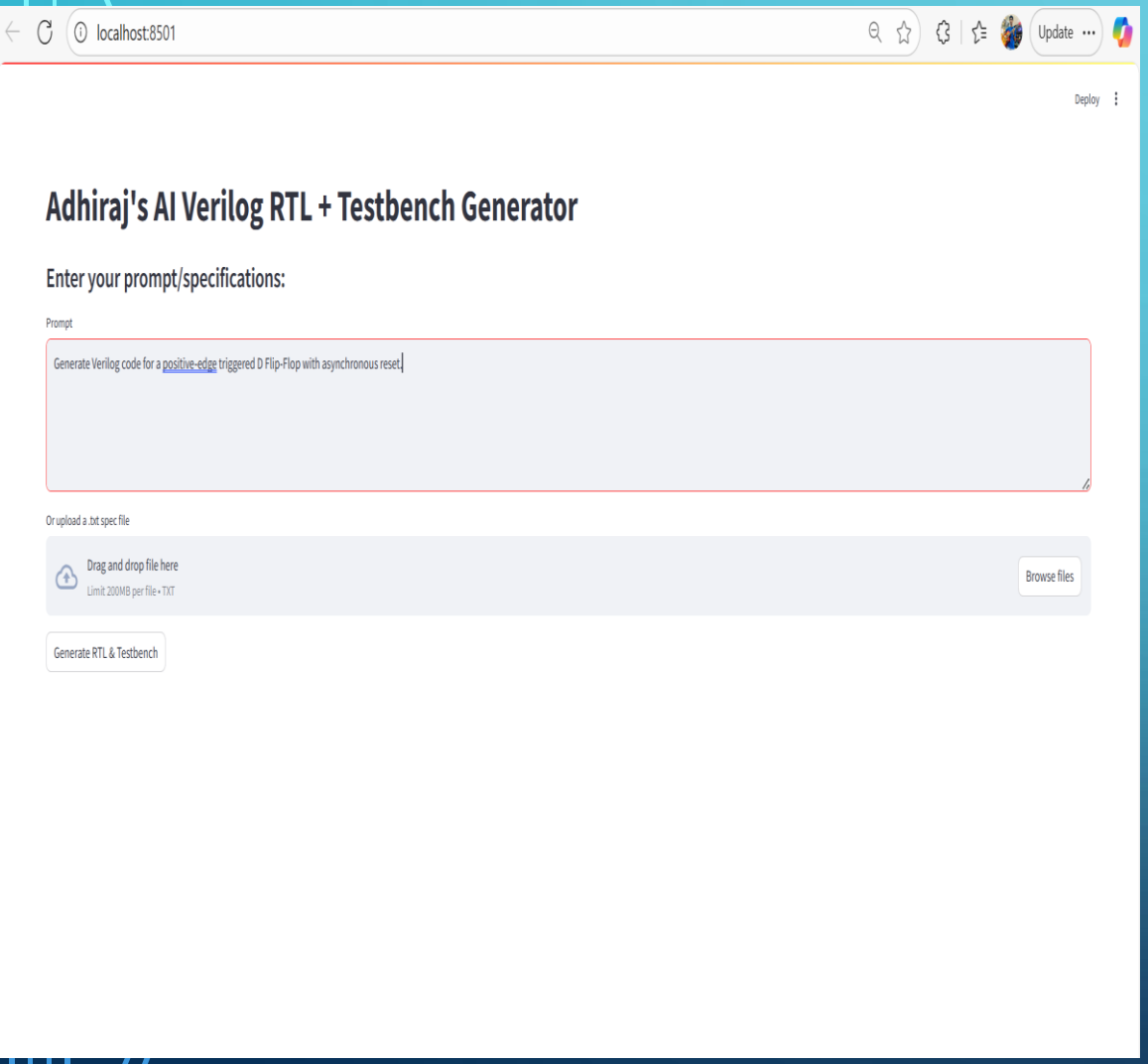
OUTPUT



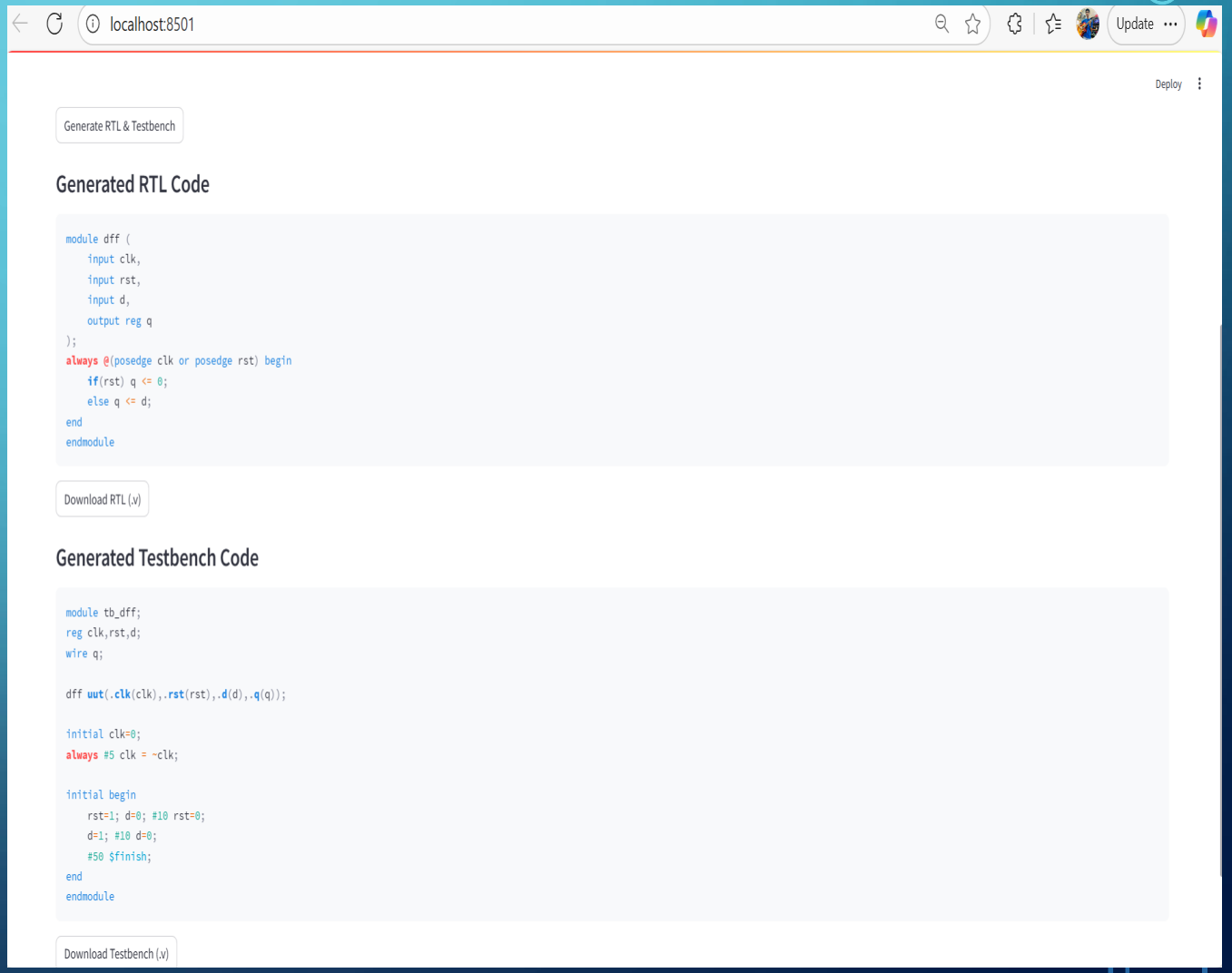

ALU_Out~2
ALU_Out~6
ALU_Out~10
ALU_Out~3
ALU_Out~7
ALU_Out~11
ALU_Out~0
ALU_Out~4
ALU_Out~8
ALU_Out~1
ALU_Out~5
ALU_Out~9
Add2
1'h0 CIN
A[3..0]
4'h1 B[3..0]
OUT[3..0]
Add3
A[4..0]
5'h1d B[4..0]
OUT[4..0]
Add1
A[5..0]
B[5..0]
OUT[5..0]
Add0
B[4..0]
Equal0
4'h0 B[3..0]
OUT
Equal1
4'h1 B[3..0]
tmp~0
tmp~1
Mux0
Mux1
Mux2
Mux3
SEL[3..0]
DATA[15..0]
A[3..0]
B[3..0]
ALU_Sel[3..0]
ALU_Out[3..0]
CarryOut

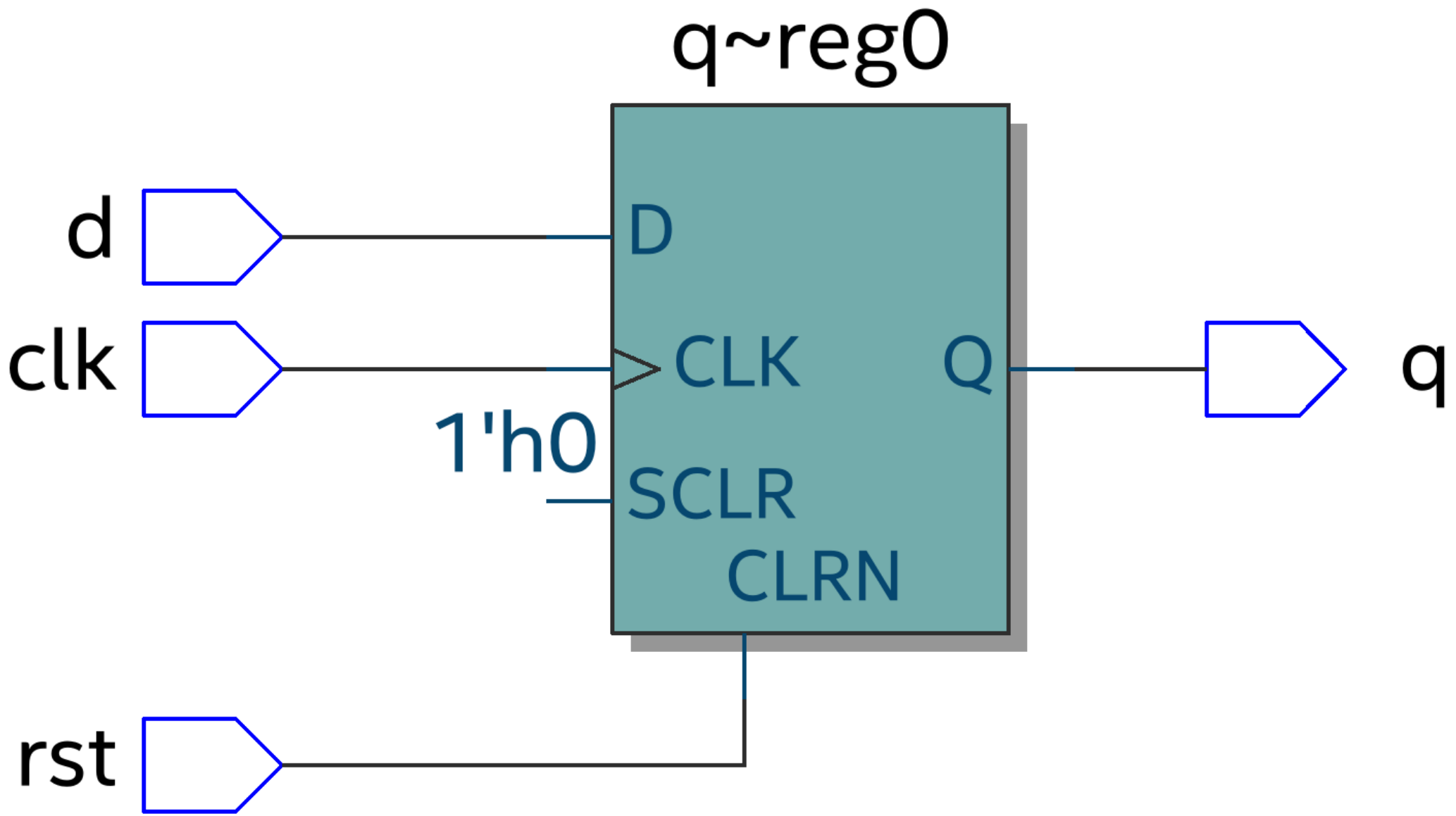
D FLIP FLOP

INPUT



OUTPUT

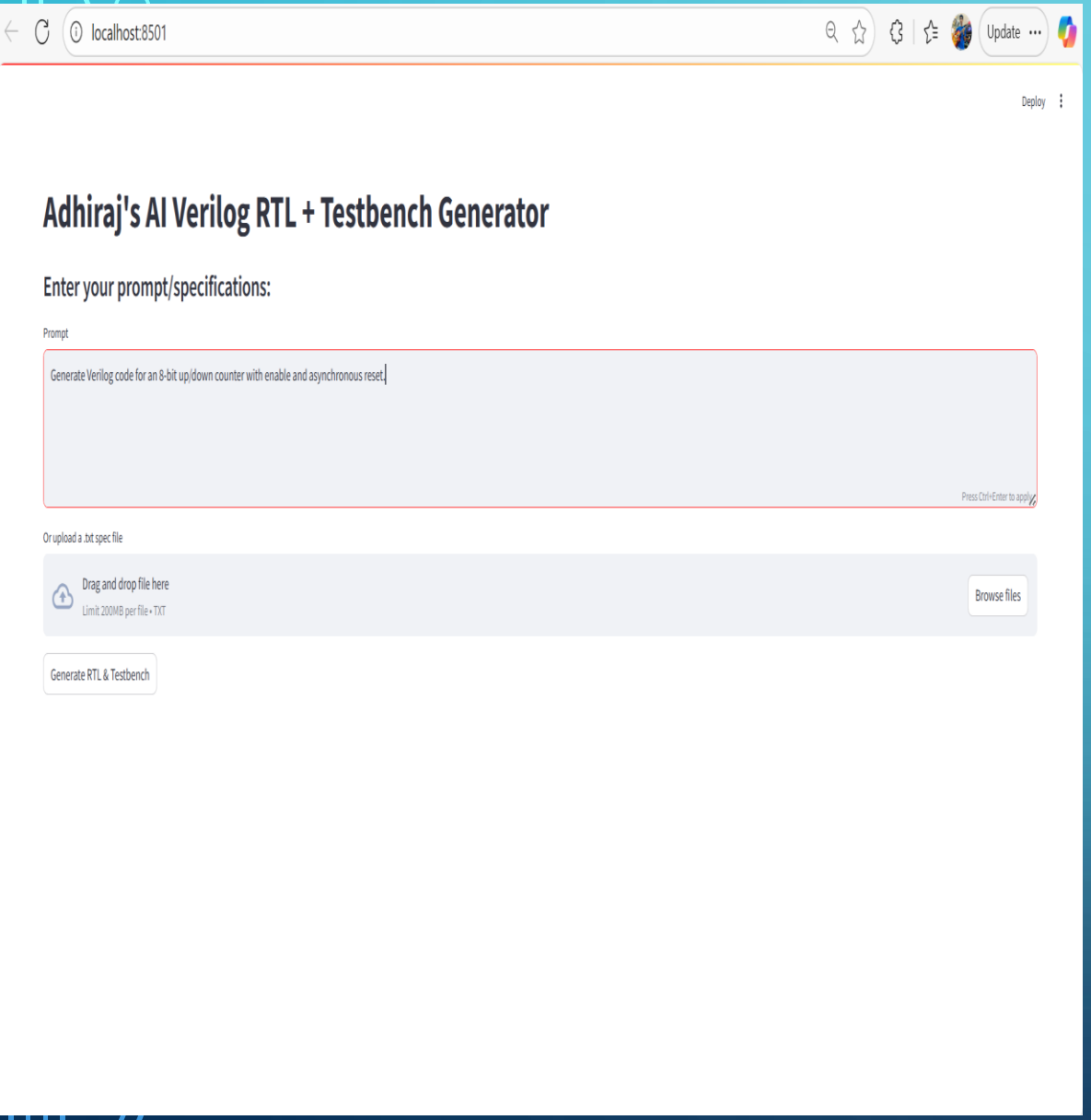


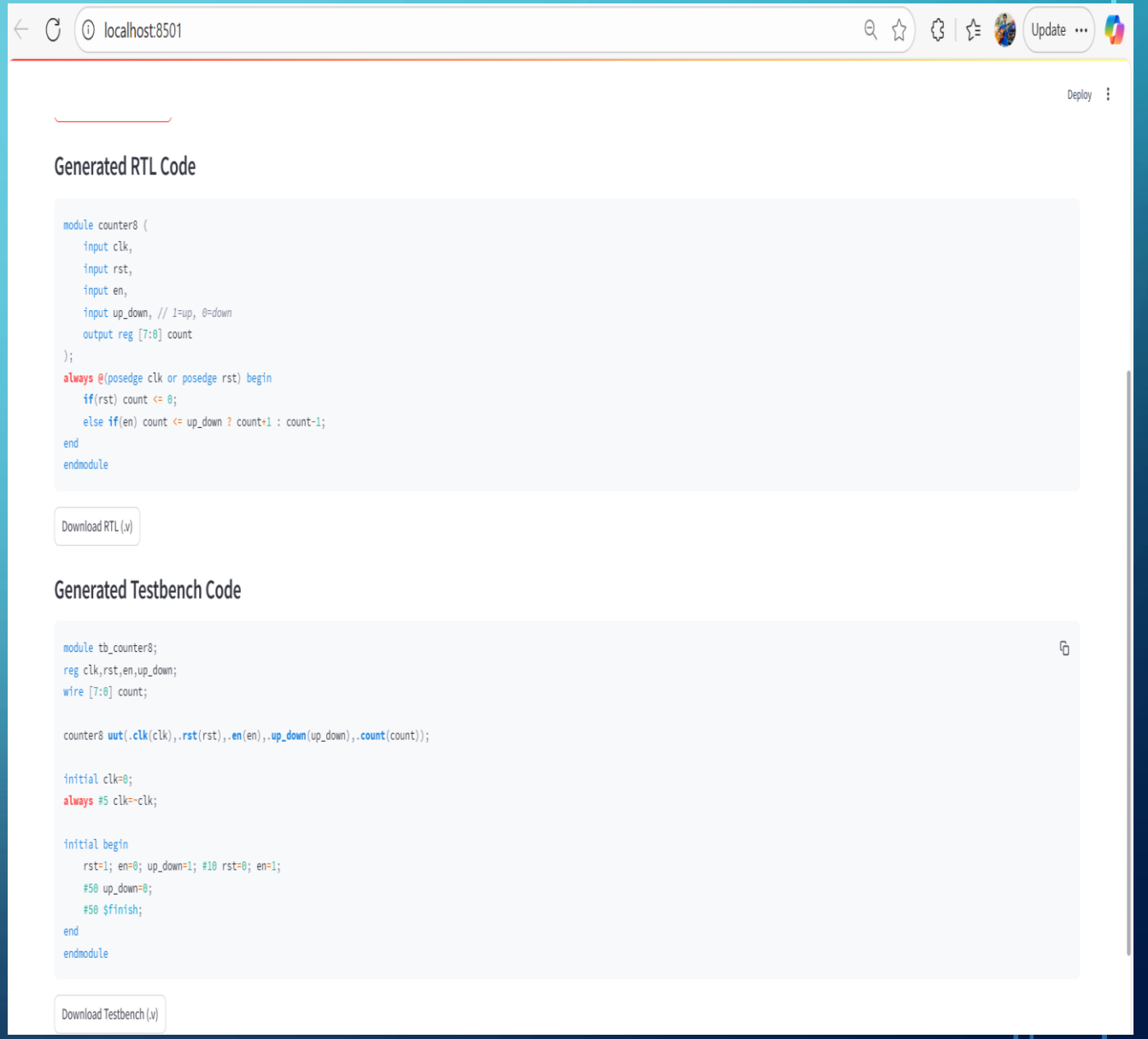

q~reg0
d
clk
1'h0
rst
D
CLK
SCLR
CLRN
Q
q

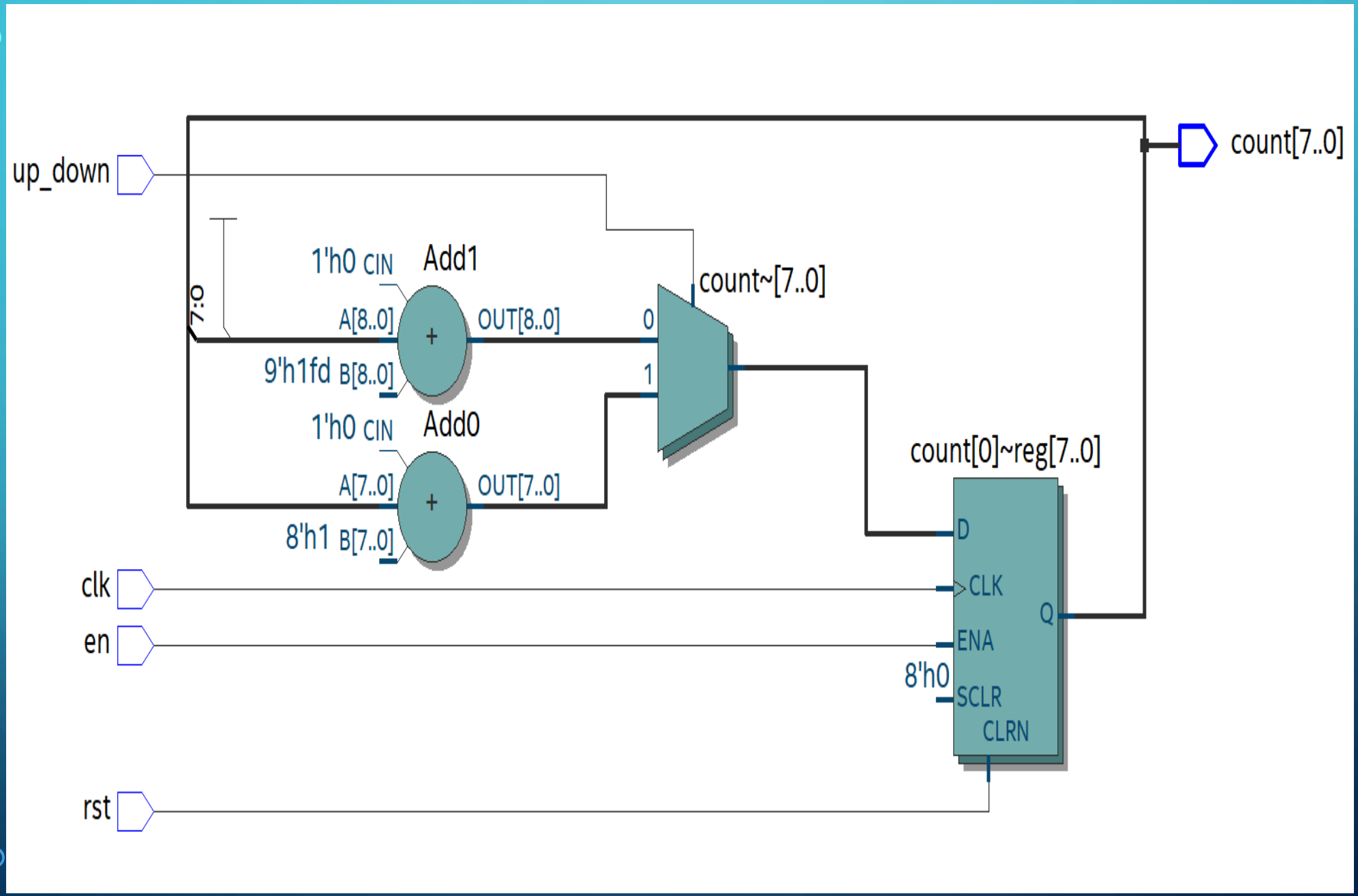
8 BIT UP/DOWN COUNTER

INPUT

OUTPUT

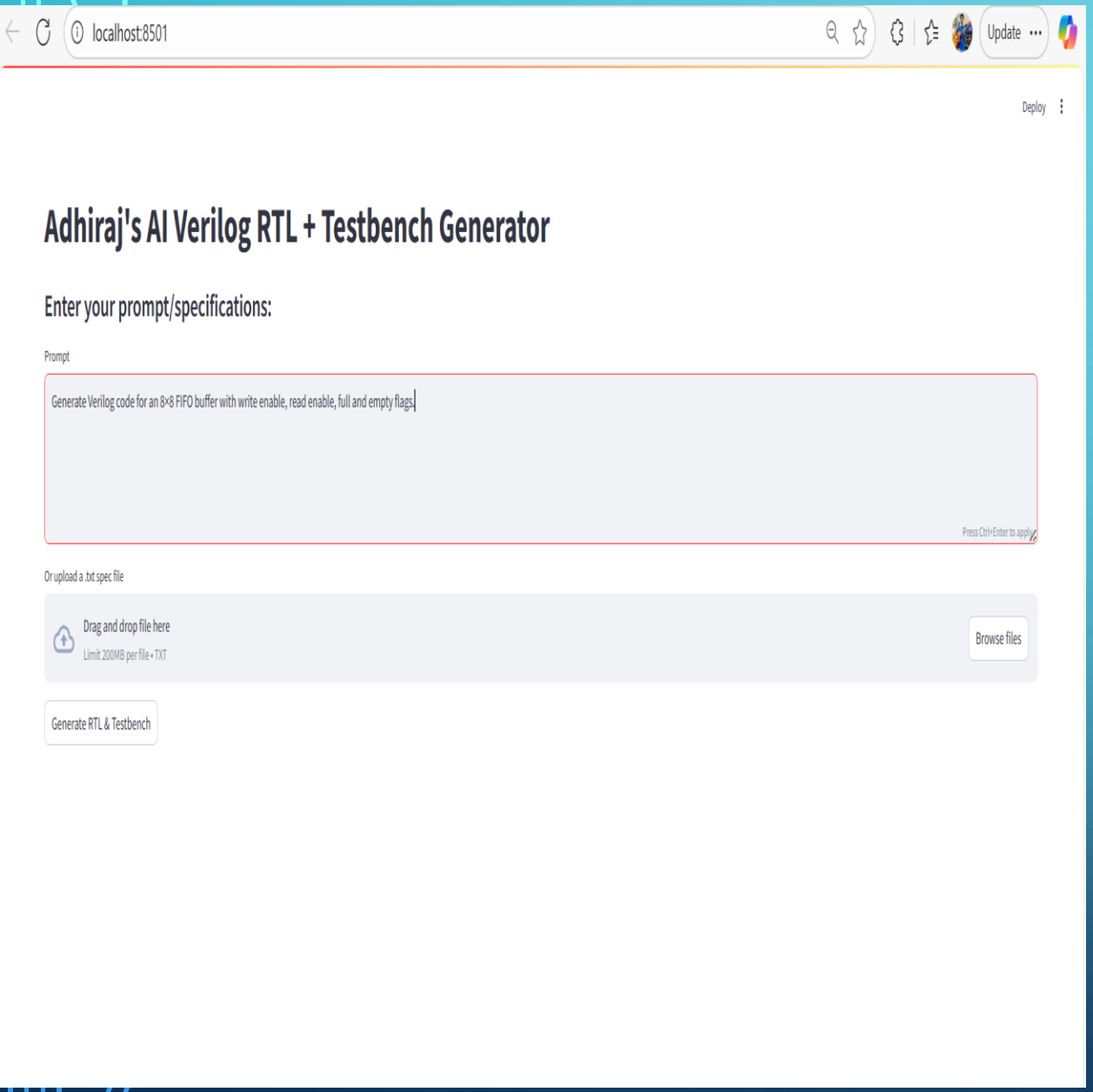




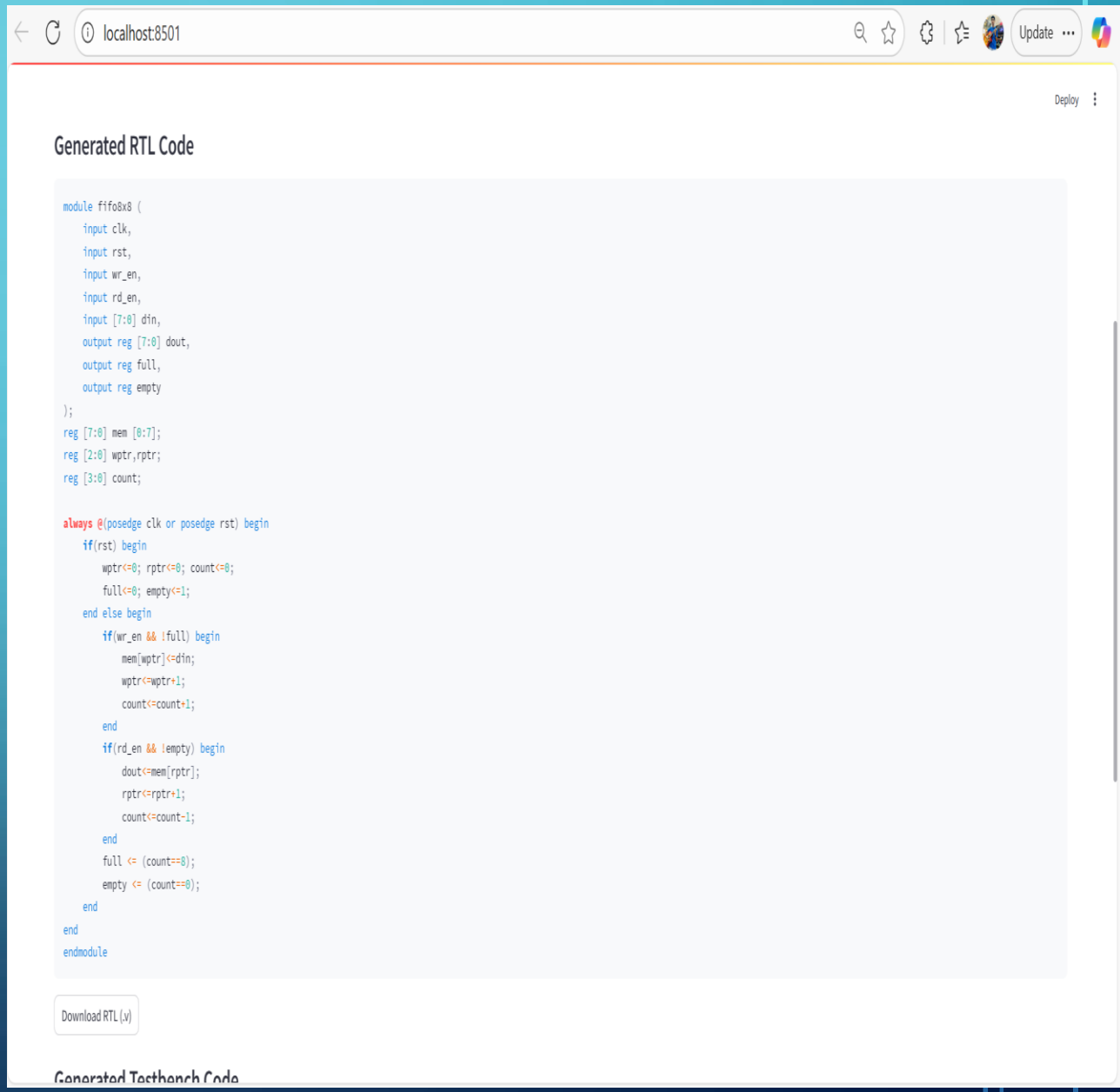

up_down
count[7..0]
1'h0 CIN
Add1
7:0
A[8..0]
OUT[8..0]
9'h1fd B[8..0]
count~[7..0]
0
1
1'h0 CIN
Add0
A[7..0]
OUT[7..0]
8'h1 B[7..0]
count[0]~reg[7..0]
D
CLK
Q
ENA
8'h0
SCLR
CLRN
clk
en
rst

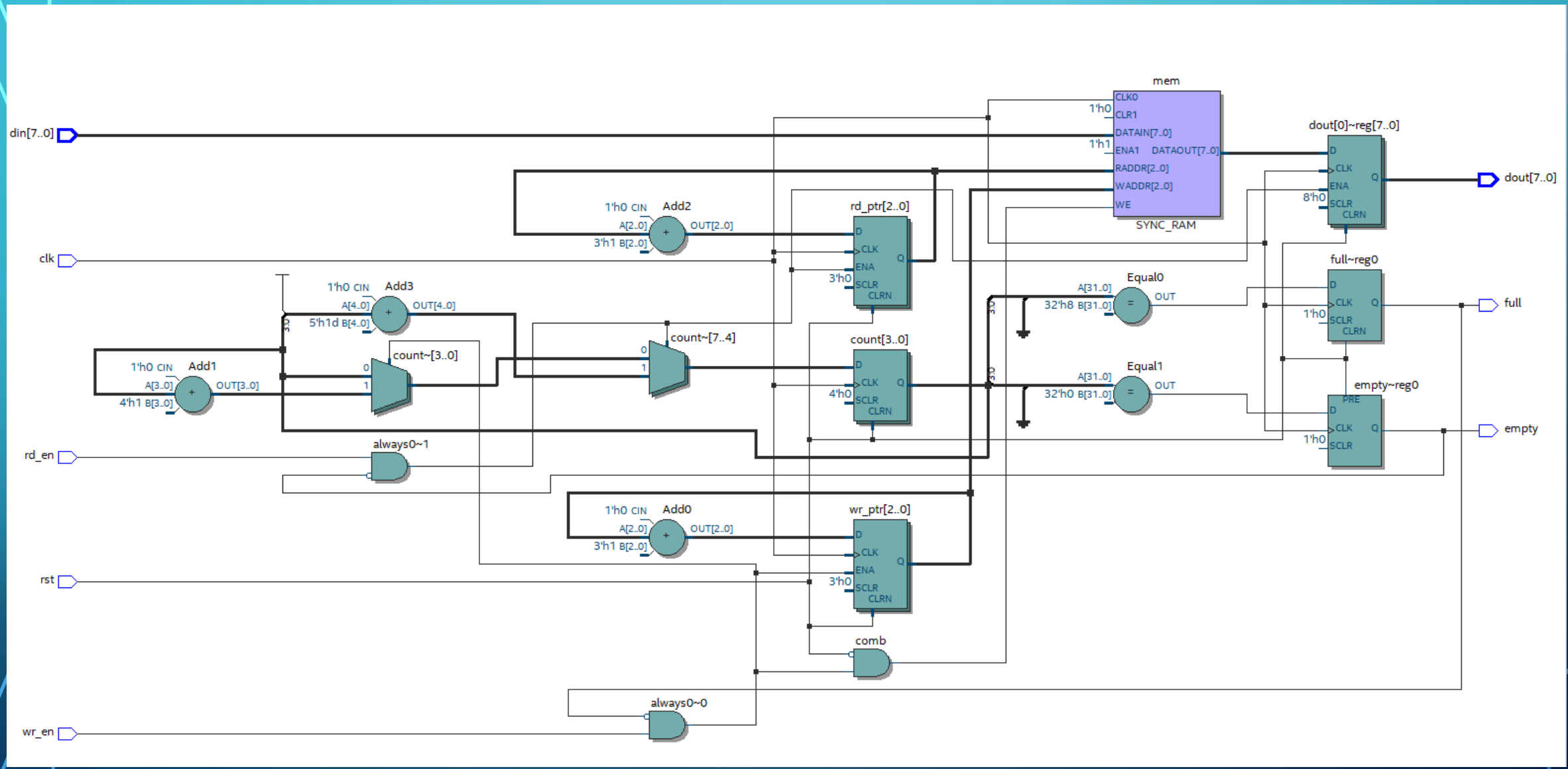
8X8 FIFO

INPUT



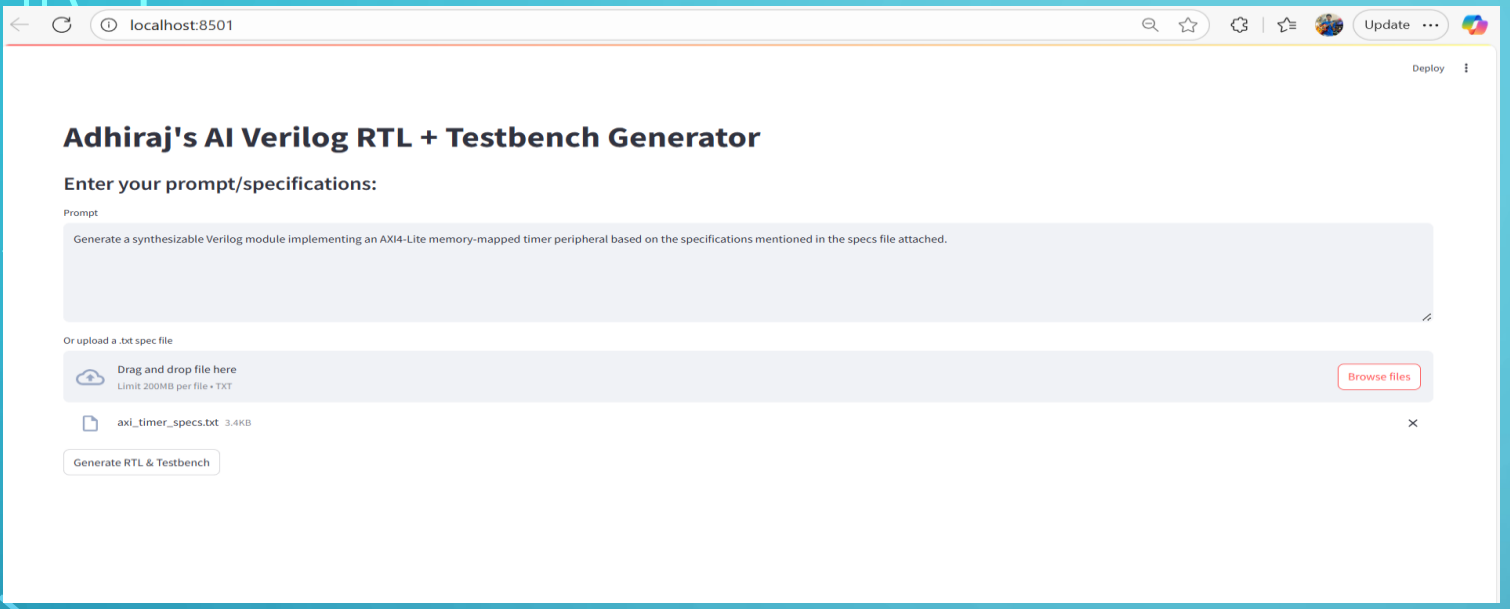
OUTPUT




din[7..0]
clk
rd_en
rst
wr_en
mem
CLK0
CLR1
DATAIN[7..0]
ENA1
DATAOUT[7..0]
RADDR[2..0]
WADDR[2..0]
WE
SYNC_RAM
1'h0
1'h1
dout[0]~reg[7..0]
dout[7..0]
8'h0
full~reg0
full
empty~reg0
empty
Add0
Add1
Add2
Add3
1'h0 CIN
A[2..0]
3'h1 B[2..0]
OUT[2..0]
A[3..0]
4'h1 B[3..0]
OUT[3..0]
A[4..0]
5'h1d B[4..0]
OUT[4..0]
rd_ptr[2..0]
wr_ptr[2..0]
count[3..0]
count~[3..0]
count~[7..4]
3'h0
4'h0
Equal0
Equal1
A[31..0]
32'h8 B[31..0]
32'h0 B[31..0]
OUT
always0~1
always0~0
comb
D
CLK
ENA
SCLR
CLRN
PRE
Q
30
0
1

AXI4-LITE MEMORY MAPPED TIMER PERIPHERAL

INPUT



SPEC FILE

```
{
  "module_name": "axi_timer",
  "description": "AXI4-Lite memory-mapped timer peripheral with programmable prescaler, auto-reload, compare interrupt, and status flags.",
  "parameters": {
    "ADDR_WIDTH": 4,
    "DATA_WIDTH": 32,
    "PRESCALER_WIDTH": 16,
    "COUNT_WIDTH": 32
  },
  "interfaces": {
    "clock_reset": {
      "clk": "System clock input",
      "rst_n": "Active-low asynchronous reset"
    },
    "axi4_lite_slave": {
      "description": "Standard AXI4-Lite slave interface for register read/write access",
      "signals": [
        "axi_awaddr", "axi_awvalid", "axi_awready",
        "axi_wdata", "axi_wstrb", "axi_wvalid", "axi_wready",
        "axi_bresp", "axi_bvalid", "axi_bready",
        "axi_araddr", "axi_arvalid", "axi_arready",
        "axi_rdata", "axi_rresp", "axi_rvalid", "axi_rready"
      ]
    },
    "interrupt": {
      "irq": "Interrupt output signal asserted when compare match occurs and IRQ enabled"
    }
  },
  "register_map": [
    {
      "name": "CTRL",
      "address_offset": "0x00",
      "access": "read/write",
      "description": "Control register",
      "fields": [
        { "name": "ENABLE", "bit_offset": 0, "bit_width": 1, "description": "Enable timer counting" },
        { "name": "AUTO_RELOAD", "bit_offset": 1, "bit_width": 1, "description": "Enable automatic reload from LOAD register" },
        { "name": "IRQ_ENABLE", "bit_offset": 2, "bit_width": 1, "description": "Enable interrupt generation on compare match" },
```

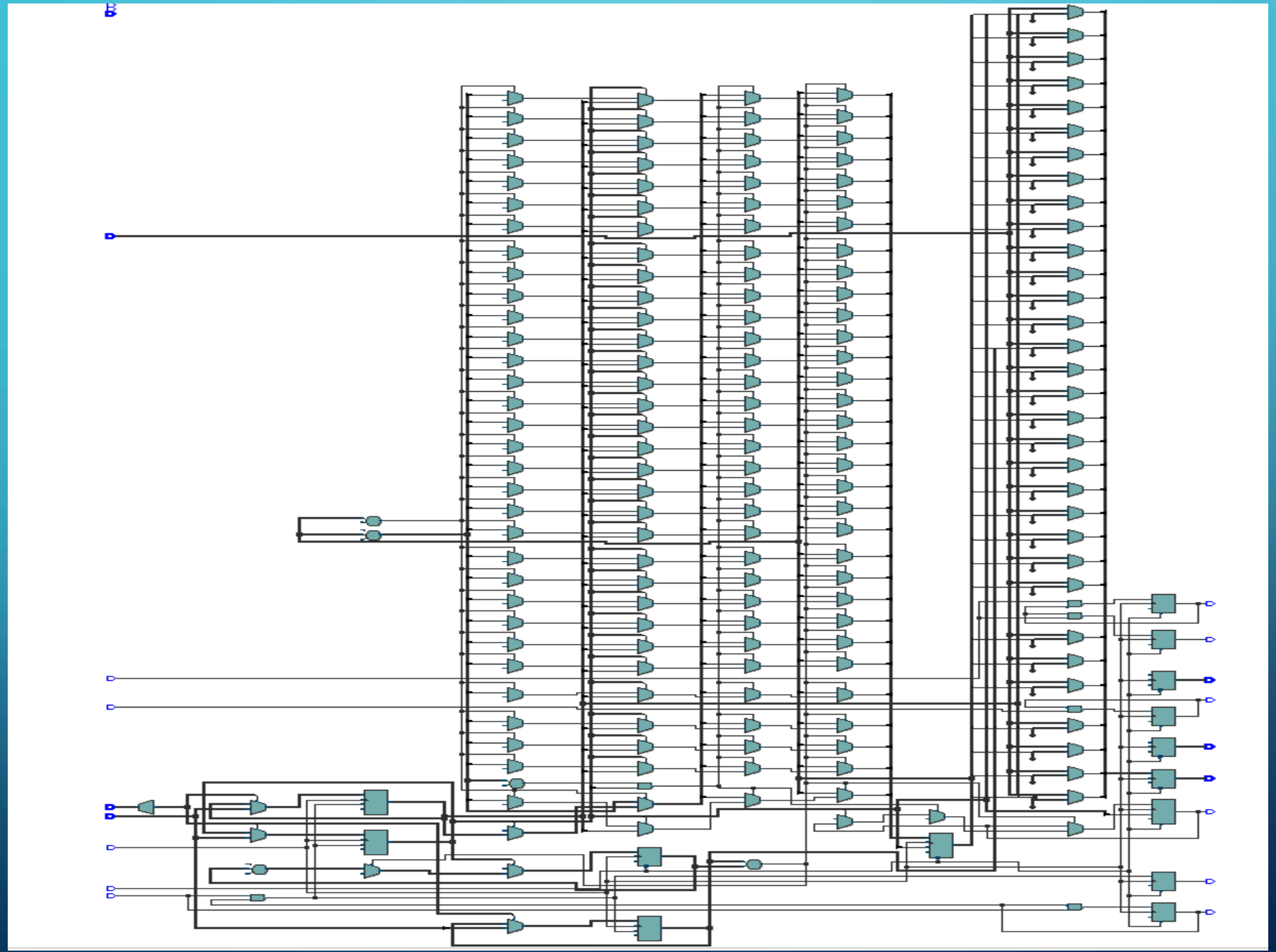
OUTPUT

Deploy

Generated RTL Code

```
module axi_timer #(
    parameter ADDR_WIDTH=4,
    parameter DATA_WIDTH=32,
    parameter PRESCALER_WIDTH=16,
    parameter COUNT_WIDTH=32
)(
    input clk,
    input rst_n,
    output reg irq,
    input [ADDR_WIDTH-1:0] axi_awaddr,
    input axi_awvalid,
    output reg axi_awready,
    input [DATA_WIDTH-1:0] axi_wdata,
    input [3:0] axi_wstrb,
    input axi_wvalid,
    output reg axi_wready,
    output reg [1:0] axi_bresp,
    output reg axi_bvalid,
    input axi_bready,
    input [ADDR_WIDTH-1:0] axi_araddr,
    input axi_arvalid,
    output reg axi_arready,
    output reg [DATA_WIDTH-1:0] axi_rdata,
    output reg [1:0] axi_rresp,
    output reg axi_rvalid,
    input axi_rready
);
    // Simplified timer logic (pre-determined)
    reg [PRESCALER_WIDTH-1:0] prescaler;
    reg [COUNT_WIDTH-1:0] load,count,cmp;
    reg enable, auto_reload, irq_enable;
    reg irq_pending;

    always @(posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            count<=0; prescaler<=0; irq<=0; irq_pending<=0;
        end else begin
            // simple tick increment
```



TOOLS & TECHNOLOGIES

Tools & Tech Used:

- Python
- Streamlit
- OpenRouter / OpenAI (LLMs)
- Verilog (RTL & TB)
- Spyder

IMPACT / FUTURE SCOPE

Impact:

- Reduced manual RTL design time.
- Accelerated design-space exploration and verification.
- Potential for integration into EDA workflows.

Future Scope:

- Add automated simulation feedback loops.
- Integrate coverage analysis and synthesis validation.
- Extend to VHDL and SystemVerilog.

DISEASE PREDICTOR AND TREATMENT RECOMMENDER

DISEASE PREDICTOR AND TREATMENT RECOMMENDER

- To predict the most likely disease based on user-input symptoms using machine learning, and recommend suitable treatments.
- **Technologies Used:**

  Python · Pandas · Scikit-learn · Random Forest · CountVectorizer · Joblib

- **How It Works:**
- **Input:** User enters symptoms as comma-separated text
- **Preprocessing:** Symptoms are vectorized using CountVectorizer
- **Prediction:** A trained Random Forest classifier predicts the disease
- **Output:** Disease name and its recommended treatments are displayed

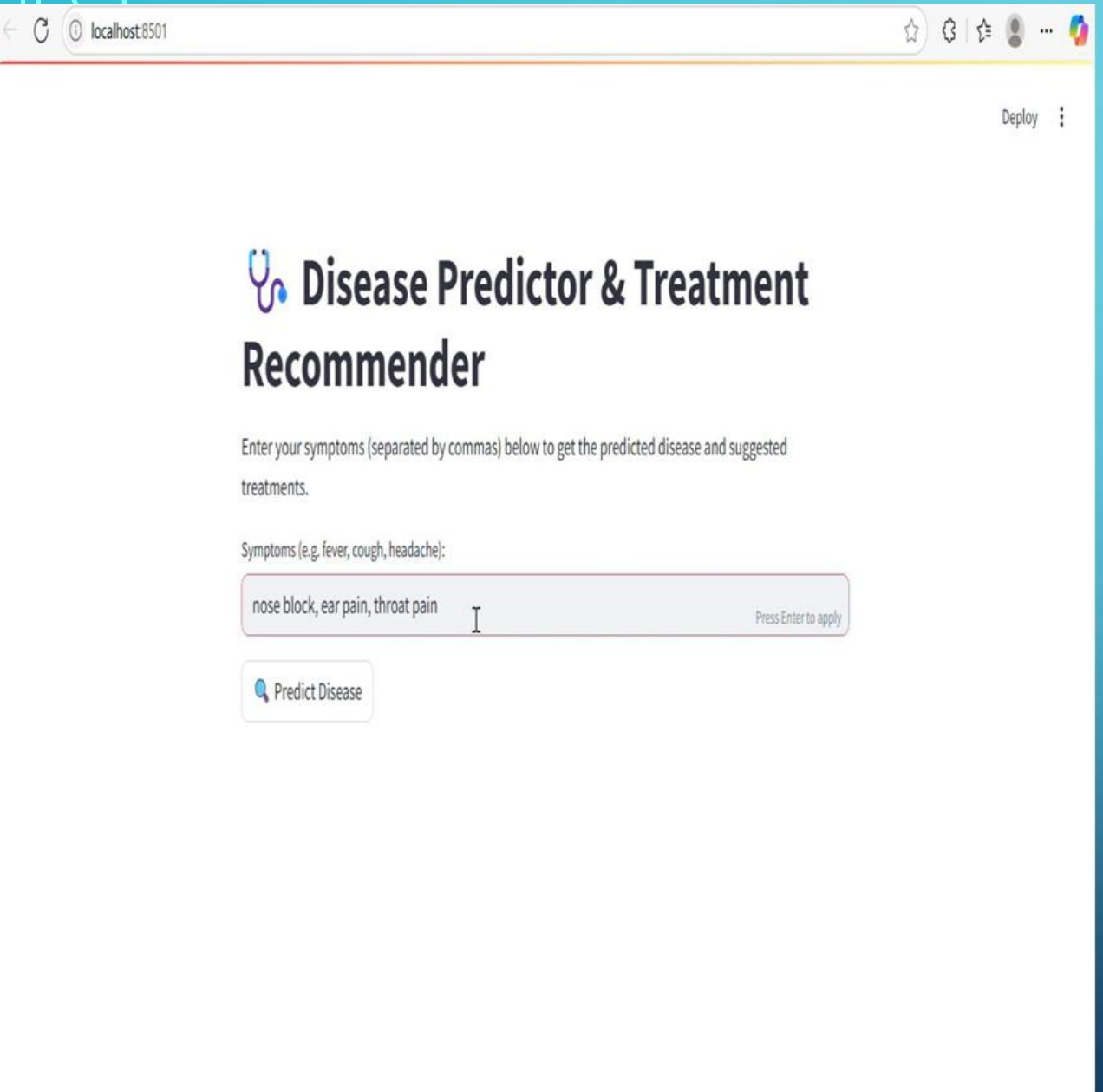
Training Pipeline Includes:

- Text Vectorization
- Label Encoding
- Random Forest Classifier
- Model + Encoder + Treatment Map saved using joblib

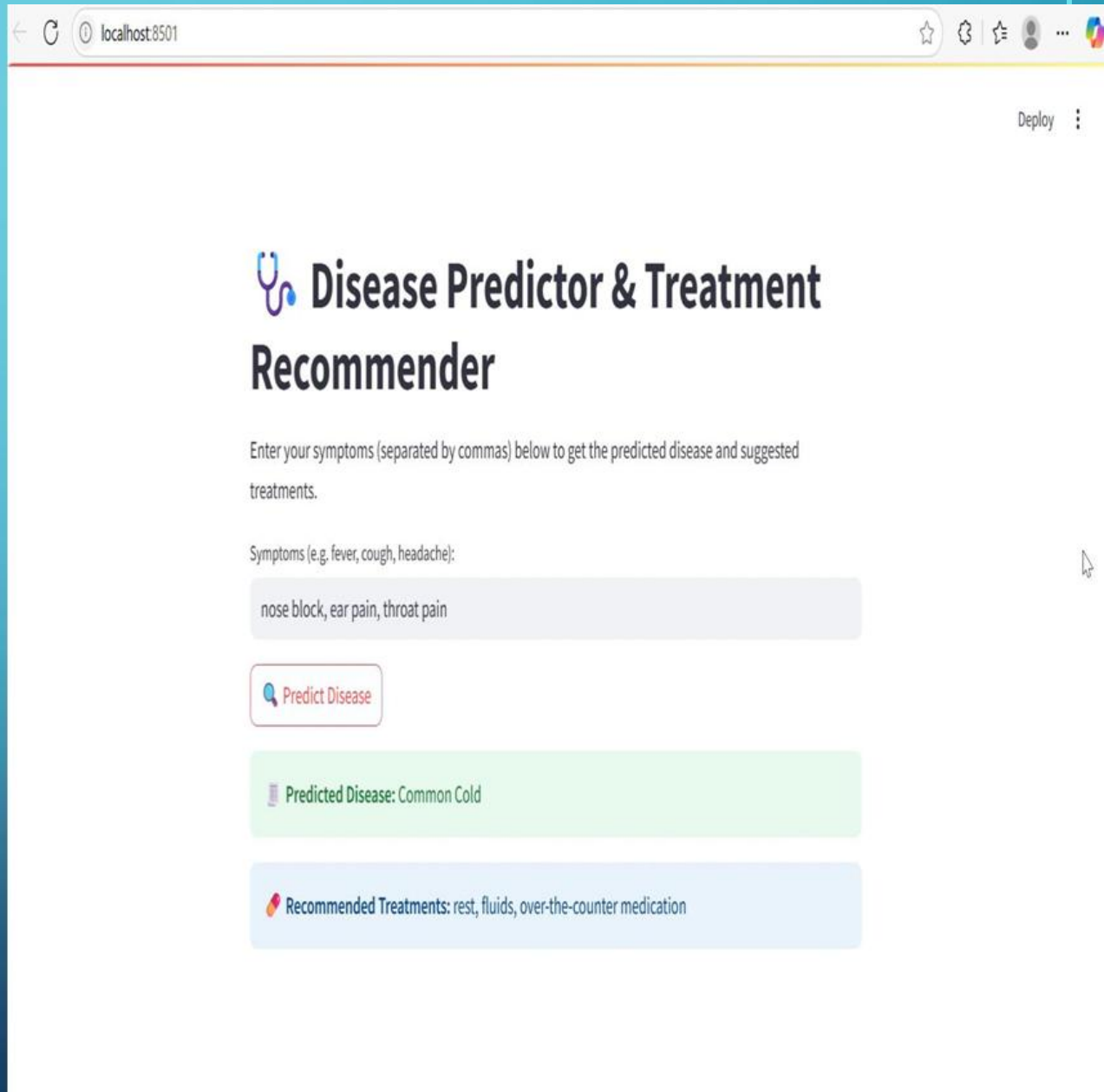
Takeaway: This project shows how real-world medical data can be turned into a smart, interactive prediction system using machine learning.

PPTOJECT SCREENSHOTS

INPUT



OUTPUT



TRAINING CODE

```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
import joblib
import os

#  Step 1: Load dataset from Kaggle path
file_path = "C:\Adhiraj\AI_ML_upflairs_internship\python\Diseases_Symptoms.csv"
df = pd.read_csv(file_path)

#  Step 2: Preprocessing
df.dropna(subset=['Symptoms', 'Name', 'Treatments'], inplace=True)
df['Symptoms'] = df['Symptoms'].str.lower().str.replace(", ", ",").str.replace(" ,", ",")  # standardize

#  Step 3: Define features and target
X_raw = df['Symptoms']
y_raw = df['Name']

#  Step 4: Encode target variable
le = LabelEncoder()
y = le.fit_transform(y_raw)
joblib.dump(le, 'label_encoder.joblib')

#  Step 5: Model pipeline
model_pipeline = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('classifier', RandomForestClassifier(n_estimators=200, random_state=42))
])
model_pipeline.fit(X_raw, y)

#  Step 6: Save model
joblib.dump(model_pipeline, 'disease_prediction_model.joblib')

#  Step 7: Save treatment map
treatment_map = dict(zip(df['Name'], df['Treatments']))
joblib.dump(treatment_map, 'treatment_map.joblib')
```

PREDICTION CODE

```python
import joblib

# Load model, label encoder, and treatment map
model = joblib.load('disease_prediction_model.joblib')
le = joblib.load('label_encoder.joblib')
treatment_map = joblib.load('treatment_map.joblib')

# Take input symptoms
input_symptoms = input("Enter symptoms separated by commas: ").lower().replace(", ", ",").replace(" ,", ",")

# Predict
predicted_label = model.predict([input_symptoms])[0]
predicted_disease = le.inverse_transform([predicted_label])[0]
treatments = treatment_map.get(predicted_disease, "No treatment info available")

print("\n🩺 Predicted Disease:", predicted_disease)
print("💊 Recommended Treatments:", treatments)
```

CHALLENGES AND SOLUTIONS

During the project, I faced several real-world challenges and overcame them using practical techniques.

•Handling inconsistent symptom text

The dataset had varied symptom formats. I solved this by applying .lower() and .replace() methods to standardize user input and dataset values.

•Mapping predictions back to disease names

Since the model used encoded labels, I used LabelEncoder and its inverse_transform() method to convert predicted labels back to readable disease names.

•Model not understanding similar symptoms

Free-text symptoms made matching difficult. I used CountVectorizer to convert symptom text into numerical vectors that the model could learn from.

•Organizing code and making it reusable

To keep the project clean, I separated the training and prediction code and used a pipeline to manage preprocessing and modeling in one step.

•Managing saved files and reusability

I used joblib to save the model, encoder, and treatment map, which made reusing and deploying the project much easier and reliable.

Takeaway: Each challenge helped me learn practical machine learning skills like data cleaning, model persistence, and user-focused program design

IMPACT AND METRICS

My project demonstrated how machine learning can be applied to real-world healthcare problems like symptom-based disease prediction. Here's what I achieved:

- **Model Accuracy & Performance**

Trained a Random Forest Classifier on symptom text Achieved accurate disease predictions in testing with real user inputs Delivered relevant treatment suggestions for each predicted disease

- **Efficiency & Reusability**

Used a full ML pipeline with CountVectorizer and RandomForestClassifier Saved the model, label encoder, and treatment map using joblib for reuse Created a clean, console-based user interface for real-time symptom input

- **Technical Deliverables**

  - Trained ML model saved as disease_prediction_model.joblib
  - Label mapping with label_encoder.joblib
  - Treatment dictionary with treatment_map.joblib
  - Interactive Python-based prediction system

- **Tools & Libraries Used**

  Python · Pandas · Scikit-learn · Joblib · VS Code / Spyder

Takeaway: This project not only improved my understanding of machine learning workflows but also taught me how to turn a dataset into a functional, user-facing solution.

NEXT STEPS AND LEARNINGS

• Key Learnings from the Internship:

• Gained hands-on experience in Python programming, data preprocessing, and machine learning

• Understood how to use libraries like Pandas, NumPy, and Scikit-learn to build real ML models

• Learned to structure projects, save models, and build user-friendly interfaces

• Developed critical problem-solving skills through real challenges • What I Plan to Do Next:

• Explore more advanced ML algorithms like XGBoost and LightGBM

• Learn deep learning topics like CNNs and RNNs using TensorFlow/Keras • Try deploying models on web platforms using Flask or Streamlit

• Work on more datasets from domains like healthcare, finance, and NLP

• Start contributing to GitHub and participating in Kaggle competitions to improve skills further

• Takeaway: This internship was a major milestone in my AI/ML journey and gave me the confidence to build real, impactful projects

ACKNOWLEDGMENTS

I would like to express my gratitude to **Upflairs** for providing this learning opportunity and valuable guidance throughout the project.

Upflairs
...because winning takes it all

CERTIFICATE
OF COMPLETION

Proudly Presented to

Adhiraj Mandal

B.Tech. IV Sem. | Department of Electronics and Communication Engineering

from **Indian Institute of Engineering Science & Technology** has successfully completed 60 Days Summer Internship & Training Program on **Data Science with Machine Learning & AI** from 1st May 2025 to 30th June 2025 organized by **Upflairs Pvt. Ltd.**

UPFLAIRS PRIVATE LIMITED

Siddharth Singh
Founder & Director
Upflairs Pvt. Ltd.

Enrollment No. :
UF/0525/6134

MSME
MICRO, SMALL & MEDIUM ENTERPRISES
OUR STRENGTH

#startupindia

Ministry of MSME, Govt. of India

IMAGE RECOGNITION AND CLASSIFICATION USING DEEP LEARNING

PROBLEM STATEMENT

- Build an image classification model using Convolutional Neural Networks (CNN).
- Train the model on the CIFAR-10 dataset containing 10 object categories.
- Enable automatic recognition of objects in input images.
- Use learned features from training data to classify new, unseen images accurately.

DATASET USED – CIFAR-10

- CIFAR-10 is a labeled image dataset widely used for machine learning research.
- Contains 60,000 images divided into 10 different classes.
- Each image is 32x32 pixels with 3 color channels (RGB).
- Classes include: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.
- Dataset is split into 50,000 training images and 10,000 testing images.

TOOLS AND TECHNOLOGIES USED

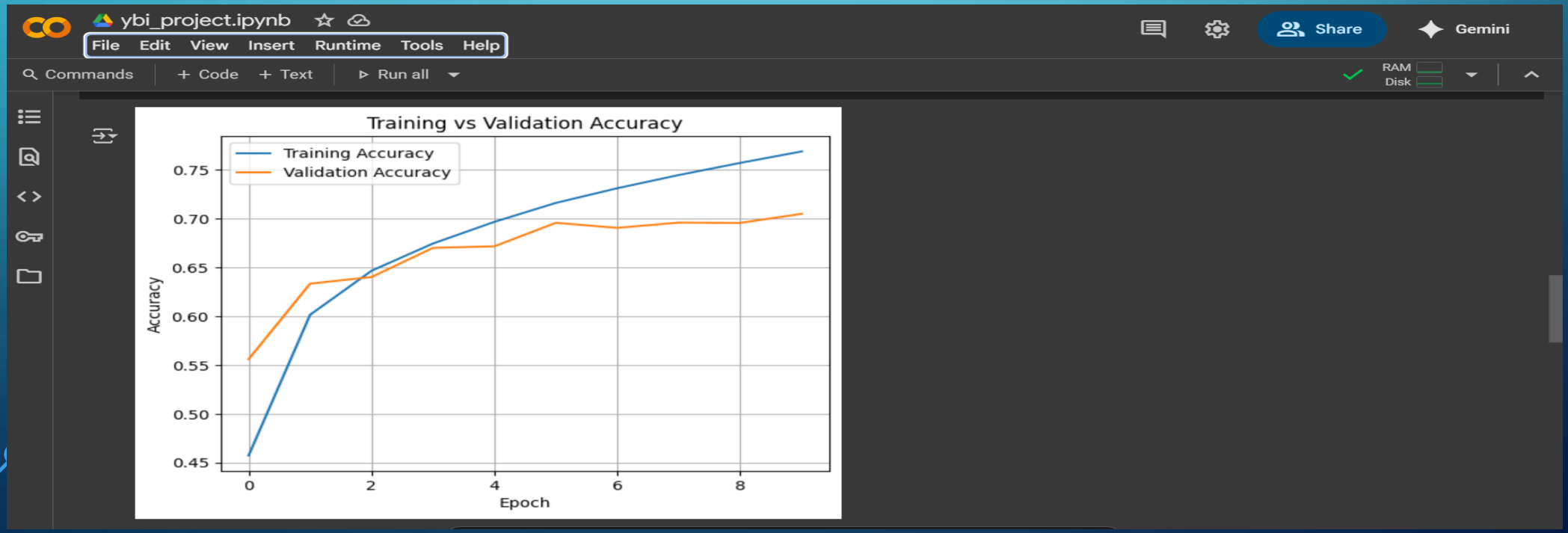
- Python– Programming language for model development
- TensorFlow & Keras– For building and training the CNN model
- Google Colab– Cloud-based platform for training the model
- Streamlit– To create the interactive web-based frontend
- NumPy & Matplotlib– For data handling and visualization
- GitHub– For version control and project submission

MODEL ARCHITECTURE- CONVOLUTIONAL NEURAL NETWORK (CNN)

- Input Layer: Accepts 32×32×3 RGB images
- Conv2D Layers: Extract spatial features using convolution filters
- MaxPooling Layers: Downsample feature maps to reduce complexity
- Flatten Layer: Converts 2D feature maps into 1D vector
- Dense (Fully Connected) Layers: Learn complex patterns and make predictions
- Output Layer: 10 neurons with softmax activation (for 10 classes)

MODEL TRAINING AND ACCURACY

- Model trained on 50,000 images from the CIFAR-10 training set
- Used categorical crossentropy as the loss function
- Adam optimizer for efficient training
- Achieved 77.36% training accuracy and 70.51% validation accuracy
- Training and validation accuracy were plotted to monitor performance
- Final model saved as cnn_model.keras for deployment



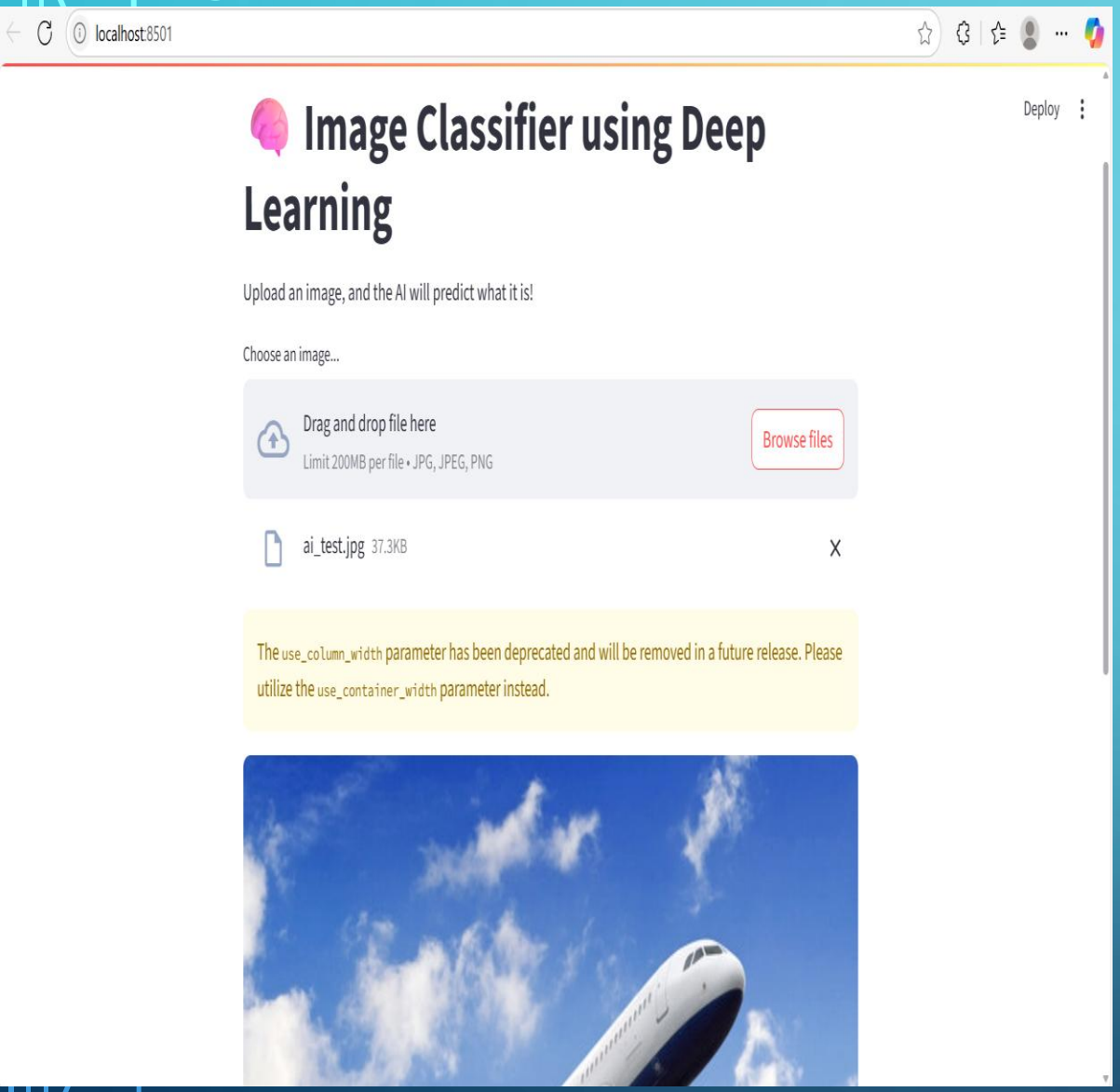
FRONT END – STREAMLIT APP

- Built a simple and interactive web interface using Streamlit
- Users can upload an image from their device
- The trained CNN model predicts the image class instantly
- Displays both the uploaded image and the predicted label
- Makes it easy to demo and test the model without writing code

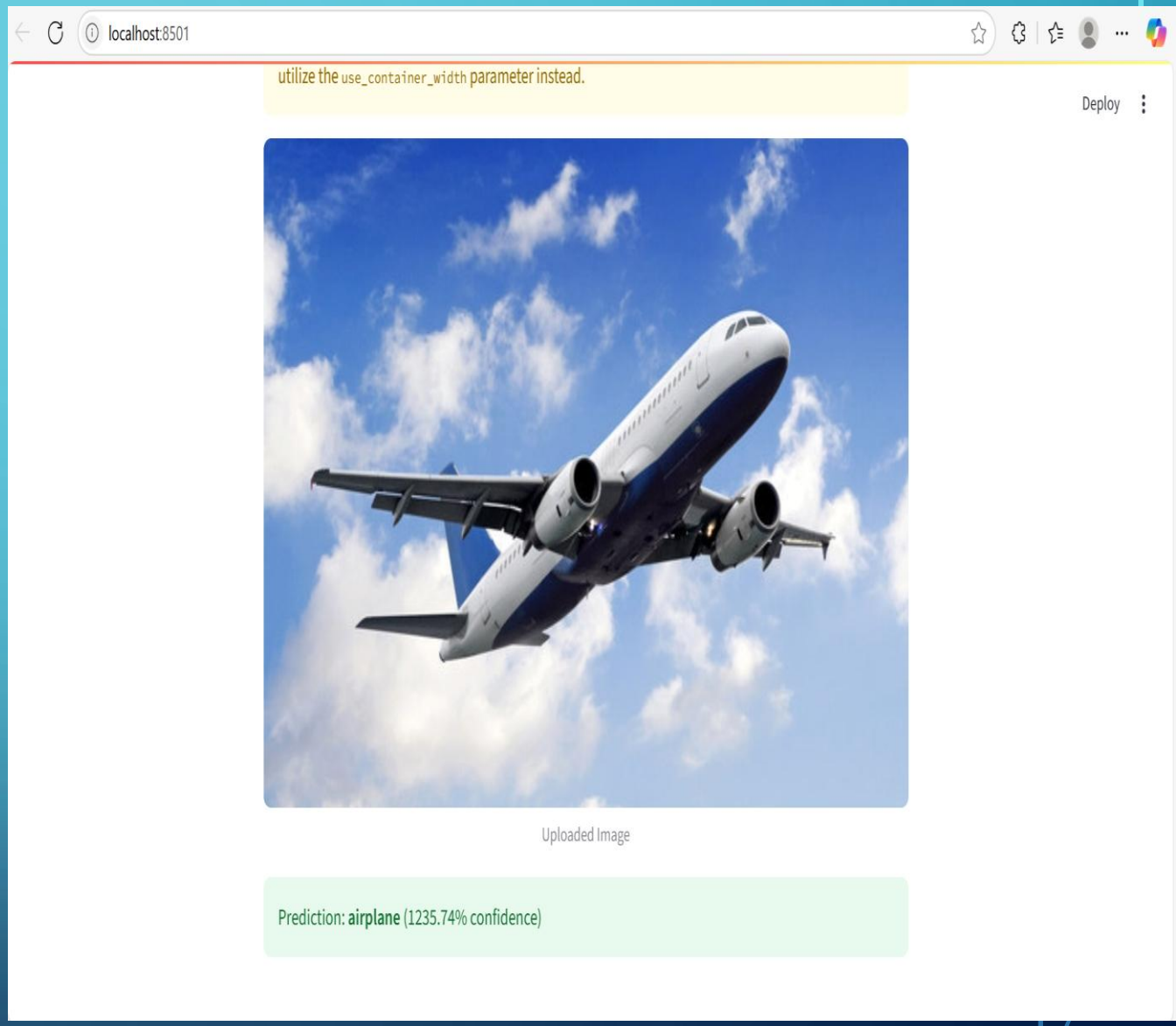
MODEL DEMO- IMAGE PREDICTIONS

- Demonstrated real-time predictions using the Streamlit app
- Uploaded test images such as airplane, dog, and ship
- The model correctly identified the uploaded images
- Showcases the model's practical usability and performance

INPUT

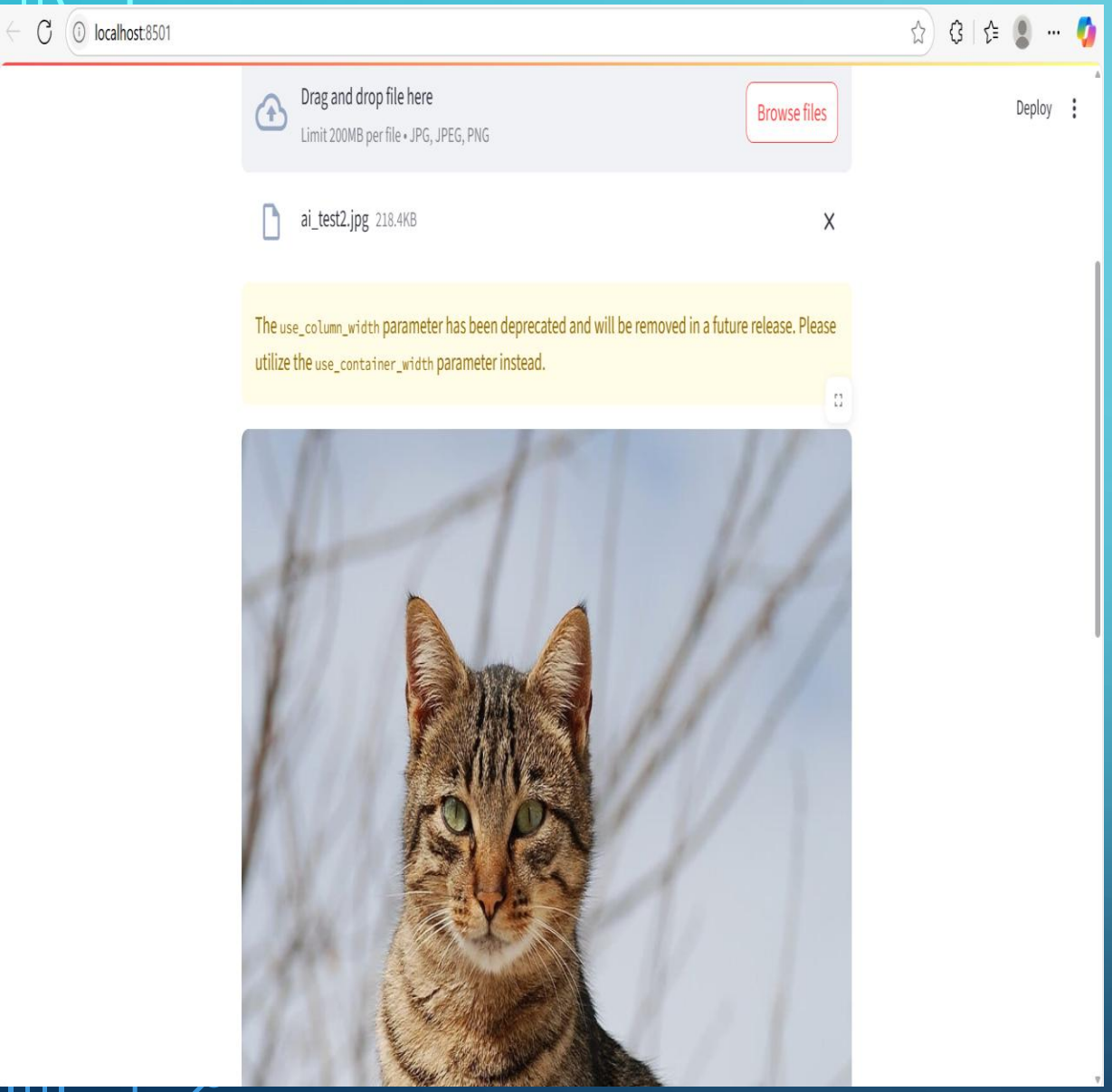


OUTPUT

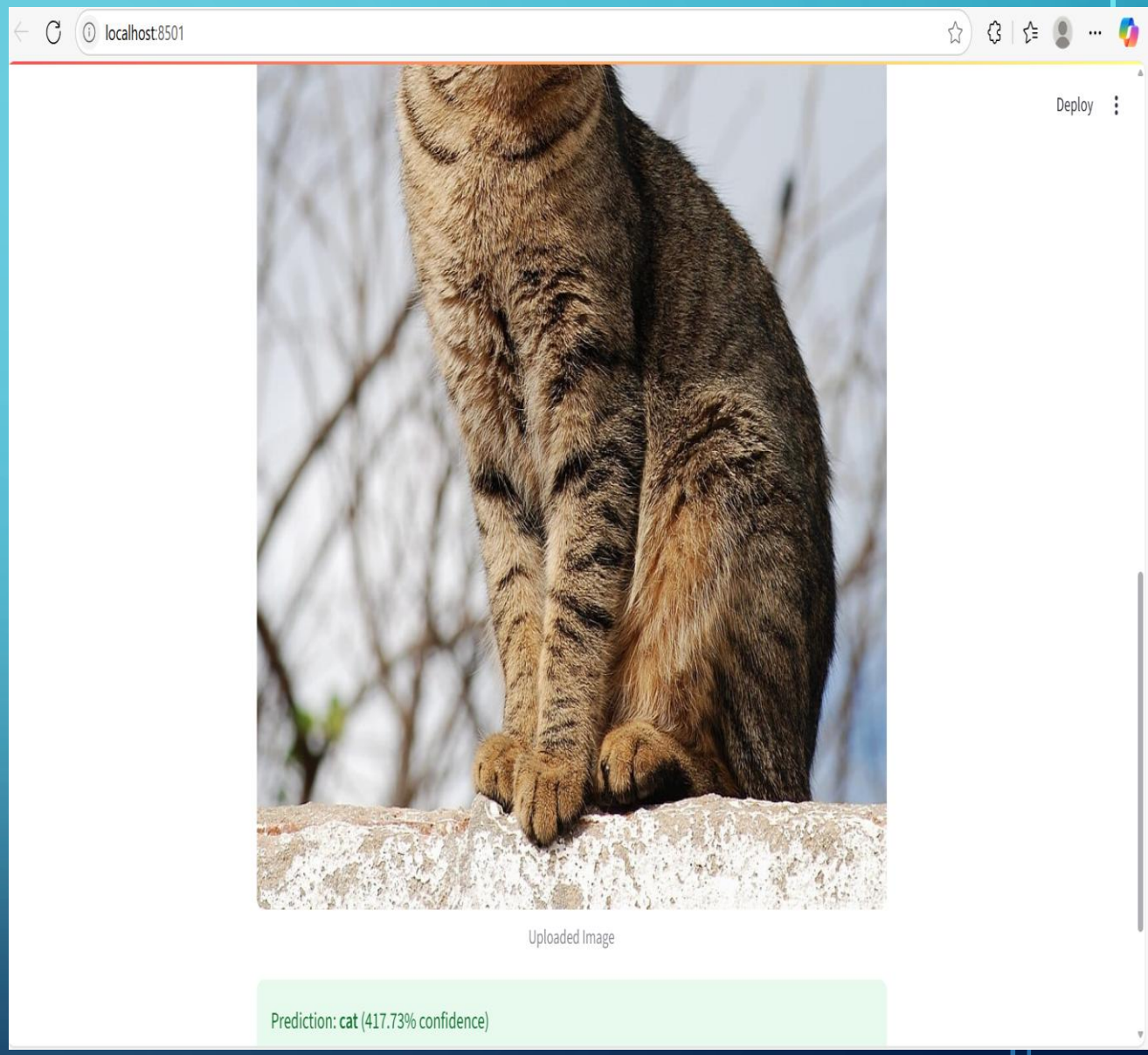


The tool has correctly predicted the airplane and classified the image accurately

INPUT

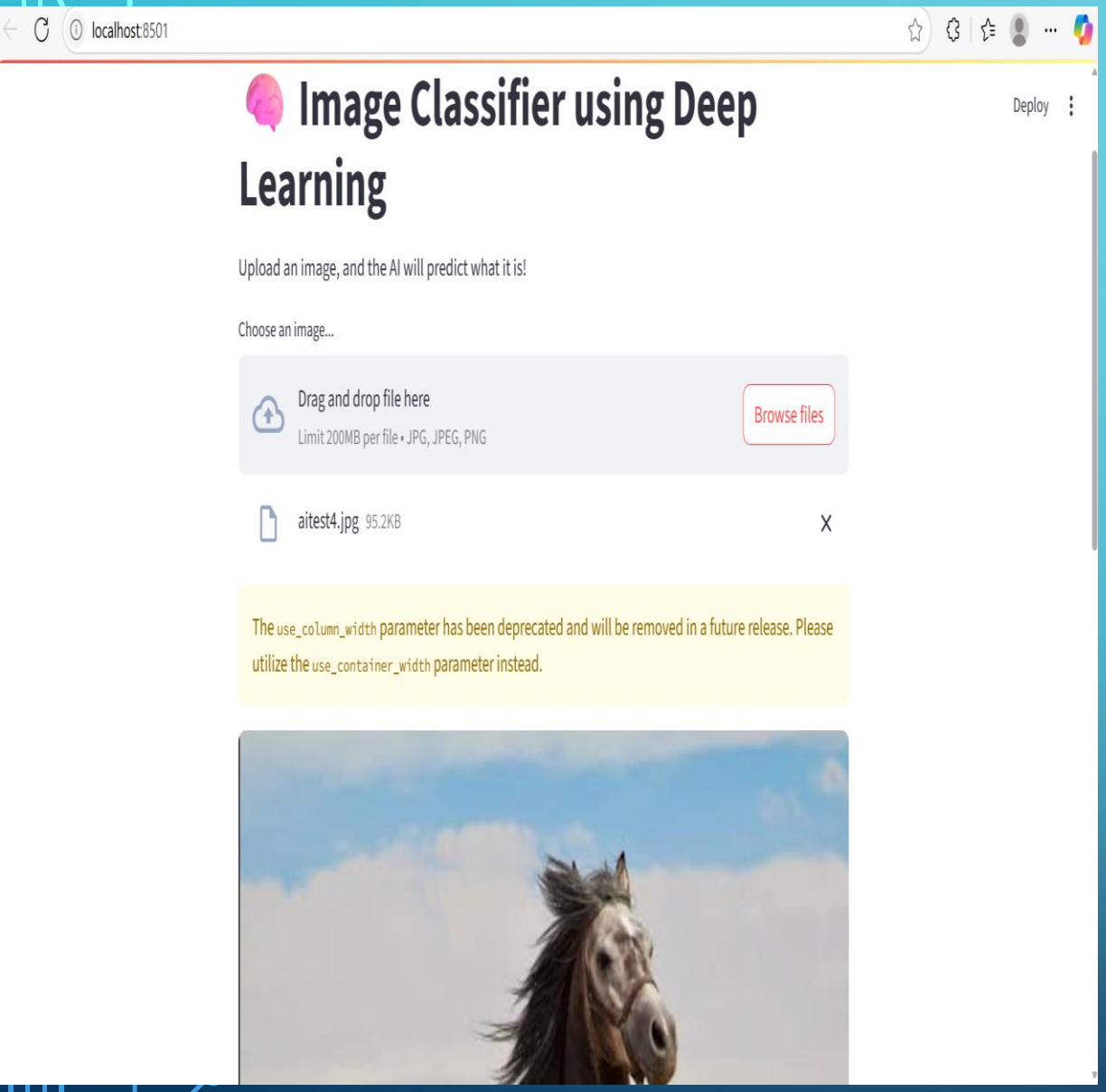


OUTPUT

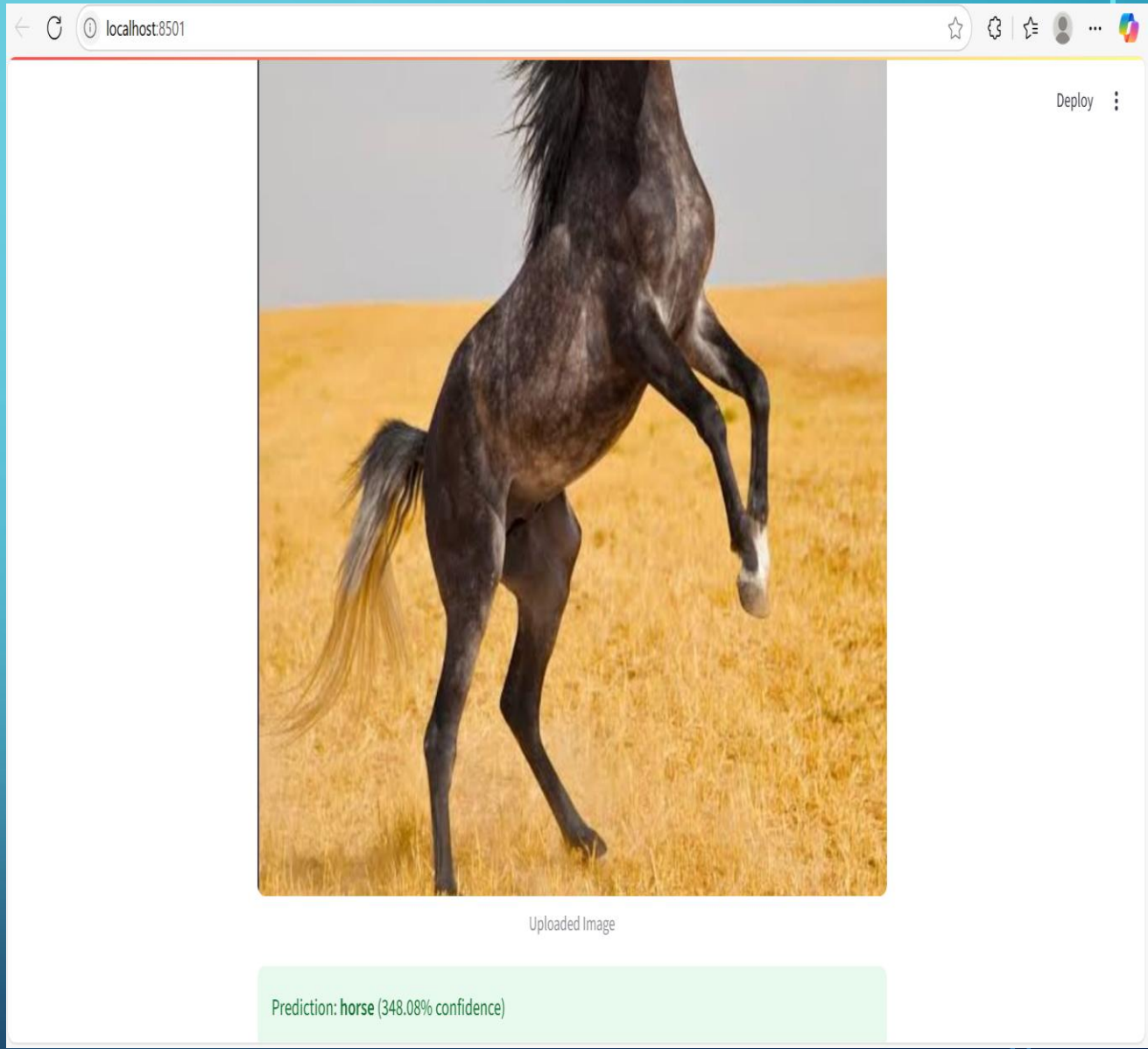


The tool has correctly predicted the cat and classified the image accurately

INPUT



OUTPUT



The tool has correctly predicted the horse and classified the image accurately

WHAT I LEARNED

- Gained hands-on experience with image classification using CNN
- Learned how to preprocess datasets and train deep learning models
- Understood the importance of model evaluation and accuracy tracking
- Explored how to create a simple frontend using Streamlit
- Learned how to save and load models for deployment
- Improved my skills in Python, TensorFlow, and project structuring

THANK YOU

I would like to express my gratitude to **YBI Foundation** for providing this learning opportunity and valuable guidance throughout the project.

THANK YOU

ADHIRAJ MANDAL

B.TECH. ELECTRONICS AND TELECOMMUNICATION ENGINEERING

INDIAN INSTITUTE OF ENGINEERING SCIENCE AND TECHNOLOGY

(IIEST SHIBPUR)