INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, KURNOOL

Project Report

Design and Implementation of AMBA AHB-Lite Memory Controller using Semi-Custom VLSI Design Flow

K. Jyoshika
123EC0041

Aim:

To design, simulate, and implement an AMBA AHB-Lite Memory Controller using the semi-custom VLSI design flow, covering stages from RTL design to GDSII generation.

Apparatus Required:

- Cadence Virtuoso
- Verilog HDL for RTL design
- Genus for synthesis
- Innovus for layout and DRC verification

Theory:

Overview:

The AMBA AHB-Lite Memory Controller is designed to interface between an AHB-Lite bus master (such as a CPU) and an on-chip memory block. It supports single-master communication and performs both read and write transactions according to the AHB-Lite protocol. The design ensures correct data transfer synchronization, pipelining, and response handling.

AHB-Lite Bus Features:

The AMBA AHB-Lite protocol provides high-performance, pipelined communication for SoC interconnects. It uses a single clock edge–triggered mechanism and supports burst transfers, sequential and non-sequential accesses. Key signals used in this design include:

- **HCLK:** Main system clock controlling all bus operations.
- **HRESETn:** Active-low reset signal used to initialize the controller.
- **HADDR:** 32-bit address bus that specifies memory locations.
- **HTRANS:** 2-bit signal indicating the type of transfer (Idle, Busy, Non-Sequential, Sequential).
- **HWRITE:** Control signal that defines the direction of the transfer (1 for write, 0 for read).
- **HSIZE:** Specifies the size of the data transfer (byte, halfword, or word).
- **HWDATA:** 32-bit data bus carrying data from master to slave during write operations.
- **HRDATA:** 32-bit data bus carrying data from slave to master during read operations.

- **HREADY:** Indicates when the slave is ready for the next transfer.
- **HRESP:** 2-bit response signal indicating the status of the transfer (OKAY, ERROR).

Internal Structure and Registers:

Inside the controller, the following key elements are implemented:

- A 256-word memory array:

  ```
  reg [31:0] mem [0:255];
  ```

  which acts as the internal RAM.
- An address register `addr_reg` to store the current address during transactions.

Operation of the Controlle:r

The controller operates synchronously with the rising edge of the clock (`HCLK`) and supports both read and write cycles.

Reset Operation:

When `HRESETn` is asserted low, all outputs and internal registers are cleared:

$$HRDATA \leftarrow 0; \quad HREADYOUT \leftarrow 1; \quad addr\_reg \leftarrow 0;$$

This ensures the memory controller starts from a known state before any transfer begins.

Write Operation:

When the bus master initiates a write transfer (`HWRITE = 1`) and the `HTRANS[1]` bit indicates a valid transfer, the address on `HADDR` and data on `HWDATA` are captured. The controller writes the data into memory as:

$$mem[HADDR[9:2]] \leftarrow HWDATA;$$

The address bits `[9:2]` are used to access word-aligned memory locations. Once the write operation completes, the slave signals readiness by asserting:

$$HREADYOUT \leftarrow 1;$$

Read Operation:

For read transfers (`HWRITE = 0`), the controller retrieves the data stored in the memory array at the address specified by `HADDR`:

$$HRDATA \leftarrow mem[HADDR[9:2]];$$

The data is then placed on the `HRDATA` bus for the master to capture. As in the write operation, the slave sets:

$$HREADYOUT \leftarrow 1;$$

Response Handling:

The signal `HRESP` is assigned a constant value:

$$HRESP = 2'b00;$$

indicating that the slave always returns an "OKAY" response, meaning no error condition occurs during transfers.

Timing Behaviour:

Each transfer consists of two main phases:

- **Address Phase:** The master provides the address and control information.
- **Data Phase:** The data is either written to or read from memory.

These phases overlap in consecutive transfers, forming a pipeline that enhances bus throughput. The effective transfer time can be represented as:

$$T_{transfer} = N_{cycles} \times T_{clk}$$

where T_{clk} is the clock period and N_{cycles} is the number of cycles per transaction (typically one cycle for valid operations).

Design Characteristics:

- Single-cycle read and write access.
- Pipelined structure compatible with AHB-Lite protocol.
- Always "OKAY" response simplifies error handling.
- Fully synchronous design with respect to `HCLK`.

Applications:

This AHB-Lite memory controller can be used in:

- Embedded microcontroller subsystems.
- On-chip SRAM interface for ARM-based processors.
- SoC test benches for bus protocol verification.
- Educational and prototype ASIC design experiments.

Procedure:

Overview:

The semi-custom VLSI design flow bridges the gap between pure RTL design and full-custom layout creation. It allows efficient implementation of digital designs using standard cell libraries and automated backend tools. The following steps describe the complete design flow followed for implementing the AMBA AHB-Lite Memory Controller.

Step 1: RTL Design and Simulation:

(a) The Verilog RTL code for the AMBA AHB-Lite Memory Controller is written using behavioral modeling techniques.

(b) The design includes all AHB interface signals such as `HCLK`, `HRESETn`, `HADDR`, `HWRITE`, `HWDATA`, and `HRDATA`.

(c) Functional verification is performed using a Verilog testbench in simulation tools like **Vivado Simulator**.

(d) The testbench applies various read and write operations, and waveform results are verified to ensure correct operation.

Step 2: Synthesis:

(a) RTL code is synthesized using **Cadence Genus**.

(b) The target technology library is specified during synthesis.

(c) The synthesis tool converts RTL into a gate-level netlist using technology-specific standard cells.

(d) Reports for area, timing, and power are generated and analyzed.

(e) Timing constraints are applied using an SDC (Synopsys Design Constraints) file to ensure correct clock setup and hold conditions.

Step 3: Floorplanning:

(a) The synthesized netlist is imported into a physical design tool such as **Cadence Innovus**.

(b) Core and die areas are defined based on utilization and aspect ratio.

(c) I/O pins for AHB-Lite signals (HADDR, HWRITE, HREADYOUT, HRDATA, etc.) are placed around the periphery.

(d) Power (VDD, VSS) rings and straps are planned for efficient current distribution.

Step 4: Placement:

(a) Standard cells from the netlist are automatically placed inside the core area.

(b) Placement optimization is performed to reduce total wirelength and improve timing.

(c) Clock buffers and repeaters are strategically placed to minimize skew.

Step 5: Clock Tree Synthesis (CTS):

(a) A balanced clock tree is generated for the main clock signal `HCLK`.

(b) The tool inserts buffers and inverters to minimize clock skew and insertion delay.

(c) Post-CTS timing analysis ensures that setup and hold times are satisfied for all sequential elements.

Step 6: Routing:

(a) After CTS, the detailed routing phase connects all signal nets and power lines according to the design rules.

(b) The router uses multiple metal layers to connect cells while adhering to spacing and width constraints.

(c) Routing quality checks ensure that all nets are connected and no shorts or opens exist.

Step 7: Physical Verification:

(a) Design Rule Check (**DRC**) and connectivity check is performed using **Assura** to ensure the layout adheres to fabrication design rules.

Step 8: GDSII Generation

(a) The final verified layout is exported as a **GDSII** file, which represents the mask layout data for fabrication.

(b) The GDS file is ready for tape-out or for viewing in layout tools like Cadence Virtuoso.

Codes:

Code 1: AHB Memory Controller (RTL)

```
`timescale 1ns/ 1ps
module ahb_lite_mem_ctrl(
    input  wire         HCLK,
    input  wire         HRESETn,
    input  wire         HSEL,
    input  wire [31:0]  HADDR,
    input  wire [1:0]   HTRANS,
    input  wire         HWRITE,
    input  wire [2:0]   HSIZE,
    input  wire [31:0]  HWDATA,
    output reg  [31:0]  HRDATA,
    input  wire         HREADY,
    output reg          HREADYOUT,
    output wire [1:0]   HRESP
);

    assign HRESP = 2'b00; // Always OKAY
```

```
    reg [31:0] mem [0:255];

    reg [31:0] addr_reg;

    always @(posedge HCLK or negedge HRESETn) begin
        if (!HRESETn) begin
            HRDATA    <= 32'b0;
            HREADYOUT <= 1'b1;
            addr_reg  <= 32'b0;
        end else begin
            if(HREADY && HSEL && HTRANS[1]) begin
                addr_reg <= HADDR;
                if(HWRITE) begin
                    mem[HADDR[9:2]] <= HWDATA;
                    HREADYOUT <= 1'b1;
                end else begin
                    HRDATA <= mem[HADDR[9:2]];
                    HREADYOUT <= 1'b1;
                end
            end else begin
                // Remain ready
                HREADYOUT <= 1'b1;
            end
        end
    end

endmodule
```

Code 2: Testbench

```
`timescale 1ns/ 1ps
module tb_ahb_lite_mem_ctrl;
    reg HCLK, HRESETn, HSEL, HWRITE, HREADY;
    reg [1:0] HTRANS;
    reg [31:0] HADDR, HWDATA;
    wire [31:0] HRDATA;
    wire HREADYOUT;
    wire [1:0] HRESP;

    ahb_lite_mem_ctrl dut (
        .HCLK(HCLK), .HRESETn(HRESETn),
        .HSEL(HSEL), .HADDR(HADDR),
        .HTRANS(HTRANS), .HWRITE(HWRITE),
        .HSIZE(3'b010), .HWDATA(HWDATA),
        .HRDATA(HRDATA), .HREADY(HREADY),
        .HREADYOUT(HREADYOUT), .HRESP(HRESP)
    );

    initial begin
        HCLK = 0;
```

```
        forever #5 HCLK = ~HCLK;
    end

    initial begin
        HRESETn = 0;
        HSEL = 0; HWRITE = 0; HREADY = 1; HTRANS = 2'b00;
        HADDR = 32'h0; HWDATA = 0;

        #12 HRESETn = 1; // Assert then release reset

        // Write
        #10; HSEL = 1; HWRITE = 1; HTRANS = 2'b10; HADDR = 32'h8;
            HWDATA = 32'hA5A5A5A5;
        #10; HSEL = 0; HWRITE = 0; HTRANS = 2'b00;

        // Read
        #10; HSEL = 1; HWRITE = 0; HTRANS = 2'b10; HADDR = 32'h8;
        #10; HSEL = 0; HTRANS = 2'b00;

        #20;
        $display("Read Data = %h", HRDATA);

        #20 $finish;
    end
endmodule
```

Code 3: Synthesis Script (TCL)

```
# ================================================================
# Genus Synthesis Script for AHB-Lite Memory Controller (
   digital_jyo)
# ================================================================

# Set library search path to your working directory
set_db init_lib_search_path /home/iiitdmk/digital_jyo/

set_db init_lib_search_path /home/install/FOUNDRY/digital/90nm/
   dig/lib/

# Specify your standard cell library (replace with actual
   filename)
set_db library slow.lib

# Read RTL
read_hdl {/home/iiitdmk/digital_jyo/ahb_lite_mem_ctrl.v}

# Elaborate design
elaborate

# Read constraints if present
```

```
read_sdc /home/iiitdmk/digital_jyo/constraints_input.sdc

# Set synthesis effort
set_db syn_generic_effort medium
set_db syn_map_effort     medium
set_db syn_opt_effort     medium
‘
# Synthesis steps
syn_generic
syn_map
syn_opt

# Write netlist and constraints
write_hdl > /home/iiitdmk/digital_jyo/ahb_lite_mem_ctrl_netlist.v
write_sdc > /home/iiitdmk/digital_jyo/
   ahb_lite_mem_ctrl_output.sdc

# Generate reports
report timing > /home/iiitdmk/digital_jyo/
   ahb_lite_mem_ctrl_timing.rpt
report power  > /home/iiitdmk/digital_jyo/
   ahb_lite_mem_ctrl_power.rpt
report area   > /home/iiitdmk/digital_jyo/
   ahb_lite_mem_ctrl_area.rpt
report gates  > /home/iiitdmk/digital_jyo/
   ahb_lite_mem_ctrl_gates.rpt

# Optionally open GUI for schematic view
gui_show
```

Code 4:Constraints.sdc

```
# ====================================================
# Timing and Design Constraints for AHB-Lite Memory Controller
# ====================================================

# Create main clock (100 MHz = 10 ns period)
create_clock -name HCLK -period 10 [get_ports HCLK]

# Input and output delay assumptions
set_input_delay 1 -clock HCLK [get_ports {HRESETn HSEL HWRITE
   HREADY HTRANS[] HADDR[] HWDATA[*]}]
set_output_delay 2 -clock HCLK [get_ports {HRDATA[] HREADYOUT
   HRESP[]}]

# Mark asynchronous reset as false path
set_false_path -from [get_ports HRESETn]

# Protect memory from optimization (behavioral array)
set_dont_touch [get_cells mem*]
```

```
# Set typical analysis conditions
set_operating_conditions -analysis_type on_chip_variation
```

Figures:

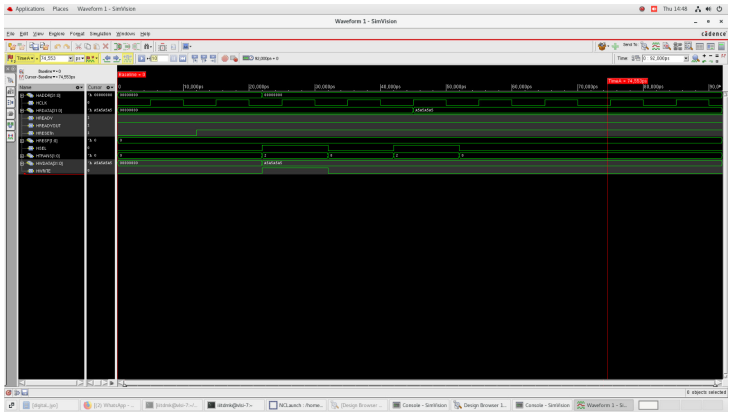

Figure 1: Simulation waveform observed in ncLaunch showing correct AHB read and write operations. The waveform verifies functional behavior of the memory controller RTL before synthesis.

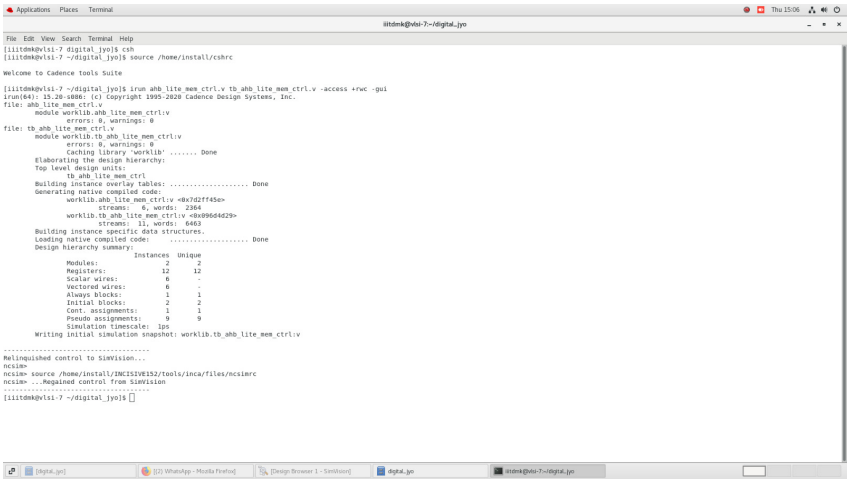

Figure 2: Command-line execution of the `irun` simulation environment. It compiles and simulates Verilog files using the NC-Verilog compiler and generates functional verification results.

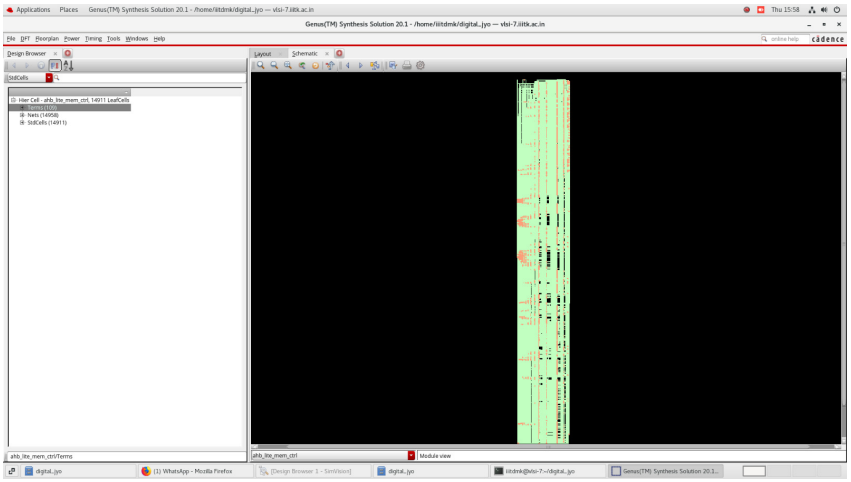

Figure 3: Full schematic view generated after synthesis in Cadence Genus. It represents the technology-mapped gate-level netlist for the AHB-Lite Memory Controller using standard cells.

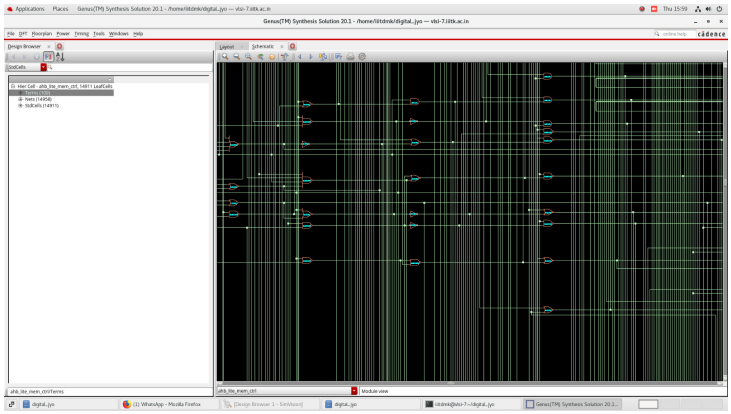

Figure 4: Zoomed-in schematic view from Genus showing logic details of address decoding and data path. Each gate corresponds to a standard cell instance in the synthesized design.

```
File  Edit  View  Search  Terminal  Help
ifile(run.tcl) 39: report area   > /home/iiitdmk/digital_jyo/ahb_lite_mem_ctrl_a
ea.rpt
ifile(run.tcl) 40: report gates  > /home/iiitdmk/digital_jyo/ahb_lite_mem_ctrl_g
tes.rpt
ifile(run.tcl) 43: gui_show
@ End verbose source ./run.tcl
genus:root: 3> report_area
============================================================
  Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
  Generated on:           Oct 30 2025  04:22:14 pm
  Module:                 ahb_lite_mem_ctrl
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

    Instance     Module  Cell Count  Cell Area  Net Area   Total Area   Wireloa
------------------------------------------------------------------------------
--
hb_lite_mem_ctrl             14911 210548.387     0.000   210548.387 <none> (D

  (D) = wireload is default in technology library
genus:root: 4>
```

Figure 5: Area report generated by Genus showing total cell area, number of instances, and hierarchy utilization.

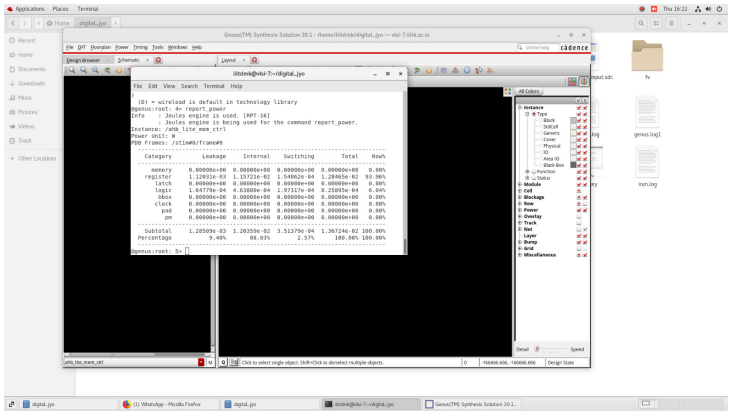

Figure 6: Power report produced after synthesis, indicating dynamic and leakage power contributions. This helps identify critical modules consuming higher power for potential optimization.

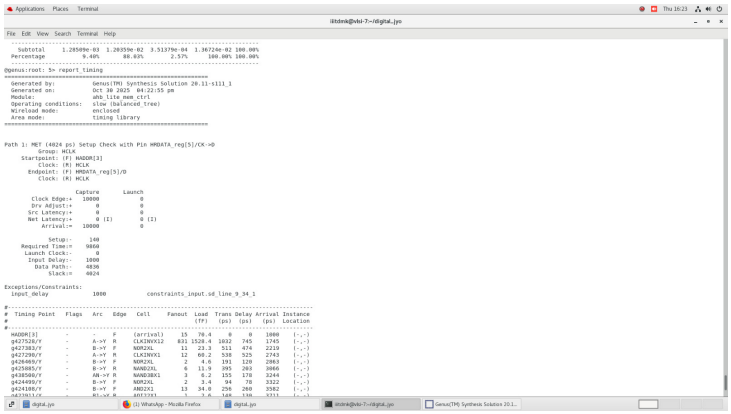

Figure 7: Timing report from Genus highlighting setup and hold slack across critical paths. Ensuring all paths meet timing constraints is essential before backend placement and routing.

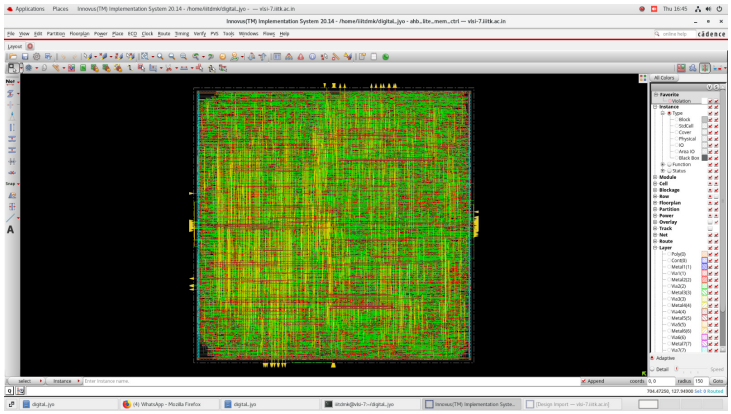

Figure 8: 2D layout view of the placed and routed AHB-Lite Memory Controller in Cadence Innovus.
The view shows standard cell rows, routing layers, and I/O pin locations after physical design.

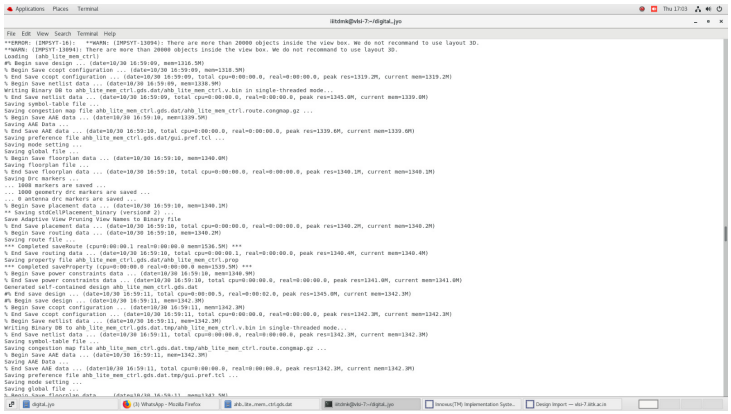

Figure 9: 3D layout rendering of the same design in Innovus. Although over 2000 routing objects exist, the 3D view provides an intuitive visualization of metal layers and interconnect density.

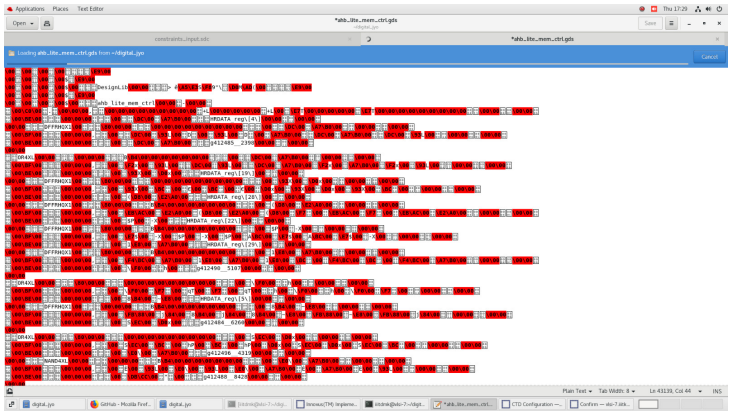

Figure 10: Final GDSII file visualization showing the complete mask layout of the AMBA AHB-Lite Memory Controller. This GDS file is the final output of the semi-custom flow, ready for fabrication.

Conclusion:

The design and implementation of the AMBA AHB-Lite Memory Controller using the semi-custom VLSI design flow were successfully carried out. The layout passed DRC and connectivity checks, verifying its correctness. This project demonstrates a complete ASIC design process from RTL to GDSII generation.