EE 671 – VLSI DESIGN

Course Project - 2

Design and Synthesis of AXI to APB bridge

Group Members:

Sahil Ashish Mehta – 24M1124
Naman Agarwal – 24M1121
Ratnesh Upadhyay – 24M1119
Abhiman Kumar – 24M1129

Overview

Here we have designed a **32 Bit AXI to APB bridge**, writing its Verilog code creating a module to translate high-speed AXI transactions to low-speed APB operations. The design includes address decoding, data conversion, and control logic to manage communication between the two protocols. Verification is carried out using a testbench, which simulates AXI transactions and checks if the bridge correctly handles read/write operations and synchronization. Once verified, we have done **RTL-to-GDS** flow, including Synthesis, Floorplan, Placement, Clock Tree Synthesis (CTS), Routing, Parasitics STA and DRC LVS checks to generate a GDSII file.

Details of Design:

AXI to APB bridge plays a vital role in the design of modern digital systems, particularly in systems that integrate different types of bus protocols, **Advanced Extensible Interface (AXI)** and **Advanced Peripheral Bus (APB)**, commonly used in System-on-Chip (SoC) designs. This bridge facilitates communication between these two distinct interfaces, enabling seamless data transfer across components using different protocols.

Key Features of AXI and APB:

AXI (Advanced Extensible Interface): AXI is a high-performance, high-bandwidth interface typically used for connecting high-speed components such as processors, memory controllers, and high-end peripherals. It supports advanced features like multiple data lanes, and burst transfers.

APB (Advanced Peripheral Bus): APB is a simpler, low-speed bus designed for interfacing with low-performance peripherals like Timers, and UARTs. It offers a simpler interface with lower power consumption as compared to AXI.

AXI to APB Bridge: Functionality and Importance

1. Protocol Translation:

The AXI to APB bridge provides a mechanism to translate between the high-speed, complex AXI protocol and the simpler, slower APB protocol. It allows components designed for AXI to communicate with peripherals and modules that are designed to interface with the APB, ensuring compatibility in heterogeneous systems.

2. Data Transfer and Clock Domain Crossing:

The bridge ensures efficient data transfer by adapting the timing and data width between AXI and APB. AXI typically operates with higher data widths and faster clock speeds compared to APB. The bridge handles this difference.

3. Cost-Effective Integration:

Many SoCs include a mix of high-performance components and lower-speed peripherals. Instead of designing separate high-performance buses for every component, the AXI to APB bridge enables the integration of both types of components on the same chip. This reduces the need for multiple interconnects and hence minimizes design complexity.

4. Flexible System Design:

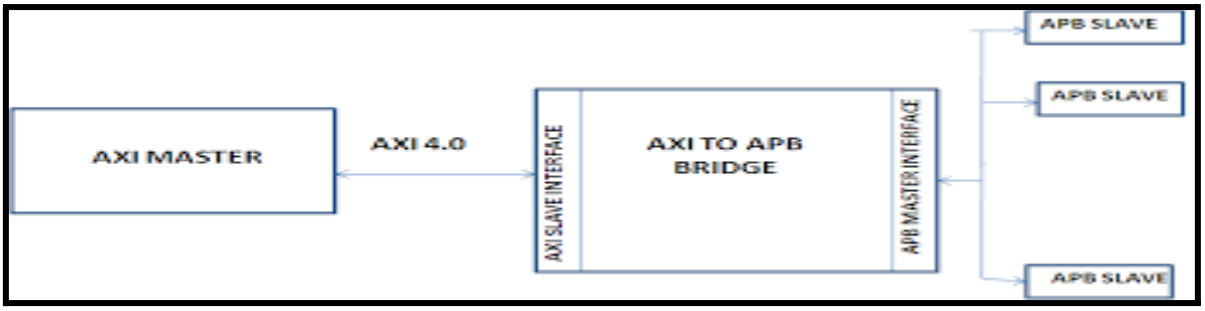
The AXI to APB bridge offers flexibility in SoC design. Designers can implement a wide range of peripherals that use the APB without needing to redesign the high-performance parts of the system that use AXI.

5. Performance Isolation:

By isolating high-performance components from low-speed peripherals, the bridge ensures that performance-critical operations on the AXI side are not slowed down by lower-speed transactions on the APB side. This isolation is important for ensuring overall system performance remains optimal.

Conclusion

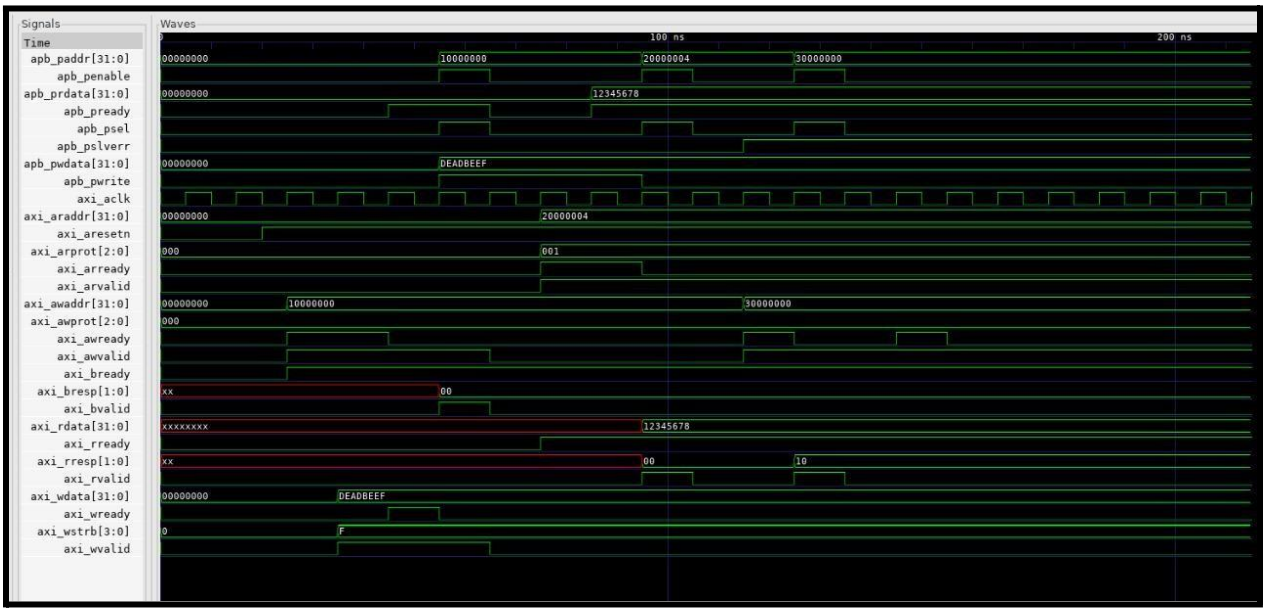
The AXI to APB bridge is an essential component in modern SoC designs, providing a crucial interface between high-performance components and low-speed peripherals. It ensures seamless communication, improves system efficiency, reduces power consumption, and simplifies design complexity, making it a cornerstone in digital communication within embedded systems and SoCs.



Simulations Output (pre synthesis):

```
VCD info: dumpfile axi_apb_bridge_tb.vcd opened for output.
Write Address Transaction: ADDR = 10000000
Write Address Transaction: ADDR = 10000000
Read Address Transaction: ADDR = 20000004
Read Address Transaction: ADDR = 20000004
Read Data Transaction: DATA = 12345678, RESP = 00
Write Address Transaction: ADDR = 30000000
Read Data Transaction: DATA = 12345678, RESP = 10
Write Address Transaction: ADDR = 30000000
```

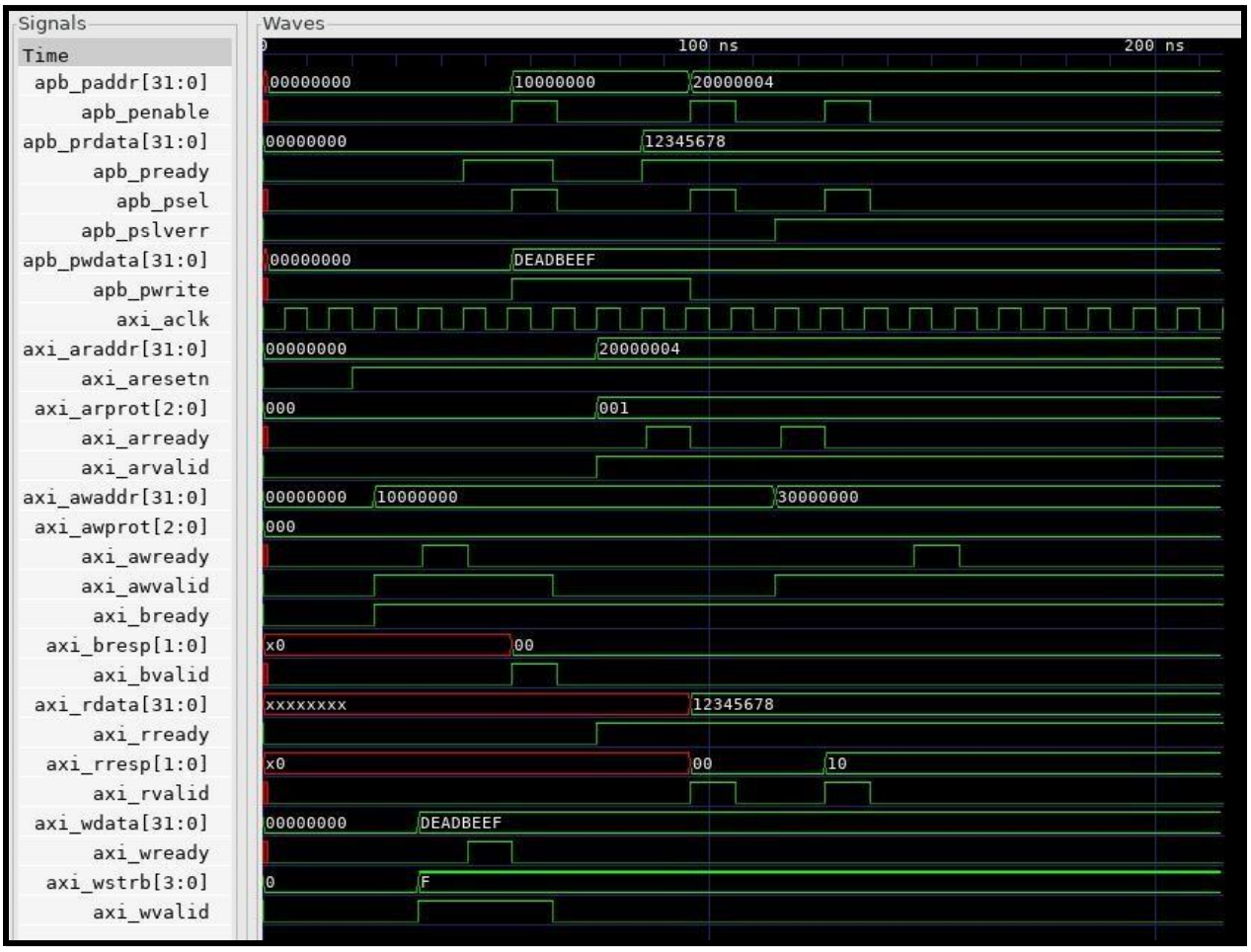
Screenshot of the gtk waveforms (pre-synthesis):



Simulations Output (post synthesis):

```
VCD info: dumpfile axi_apb_bridge_tb.vcd opened for output.
Write Address Transaction: ADDR = 10000000
Read Address Transaction: ADDR = 20000004
Read Data Transaction: DATA = 12345678, RESP = 00
Read Address Transaction: ADDR = 20000004
Read Data Transaction: DATA = 12345678, RESP = 10
Write Address Transaction: ADDR = 30000000
```

Screenshot of the gtk waveforms (post-synthesis):



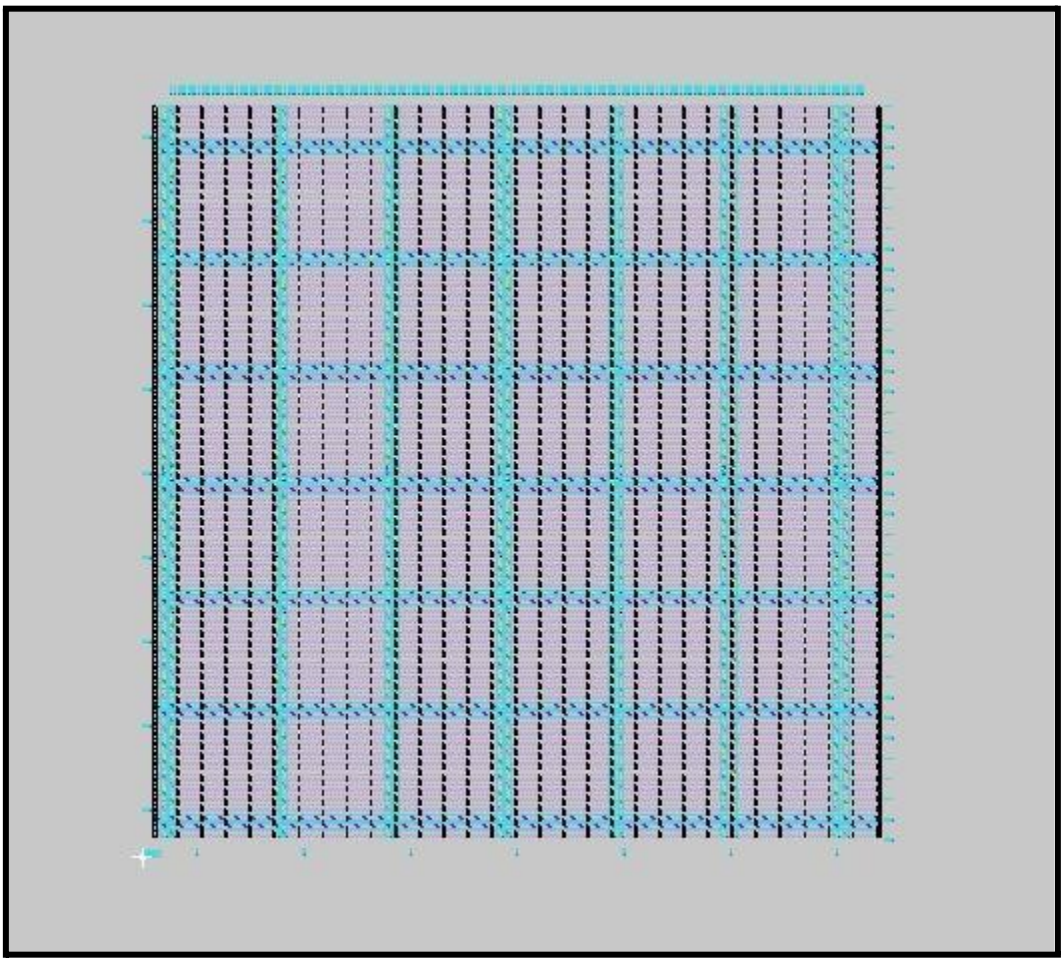
The following table was generated from the RTL file:

Number of combinational cells	**593**
Number of Sequential cells (Flip flops)	**139**
Total number of cells	**732**

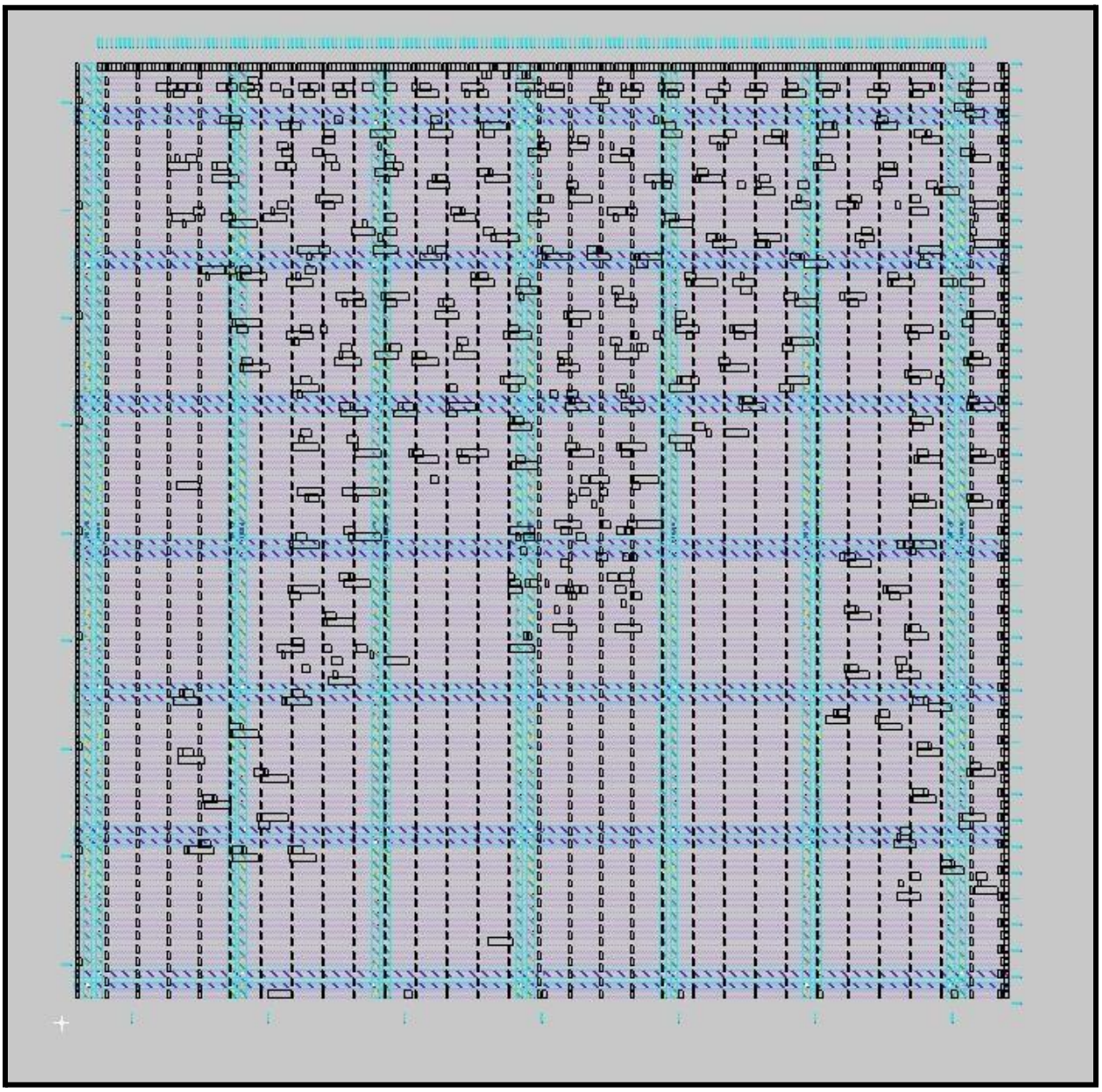
Screenshot of calculation of Combinational and Sequential blocks :

```
[EE671_3@vlsi73 synthesis]$ cat axi_apb_bridge.v | grep "sky130_fd_sc_hd__" | wc -l
593
[EE671_3@vlsi73 synthesis]$ cat axi_apb_bridge.v | grep "sky130_fd_sc_hd__dfrtp" | wc -l
139
```

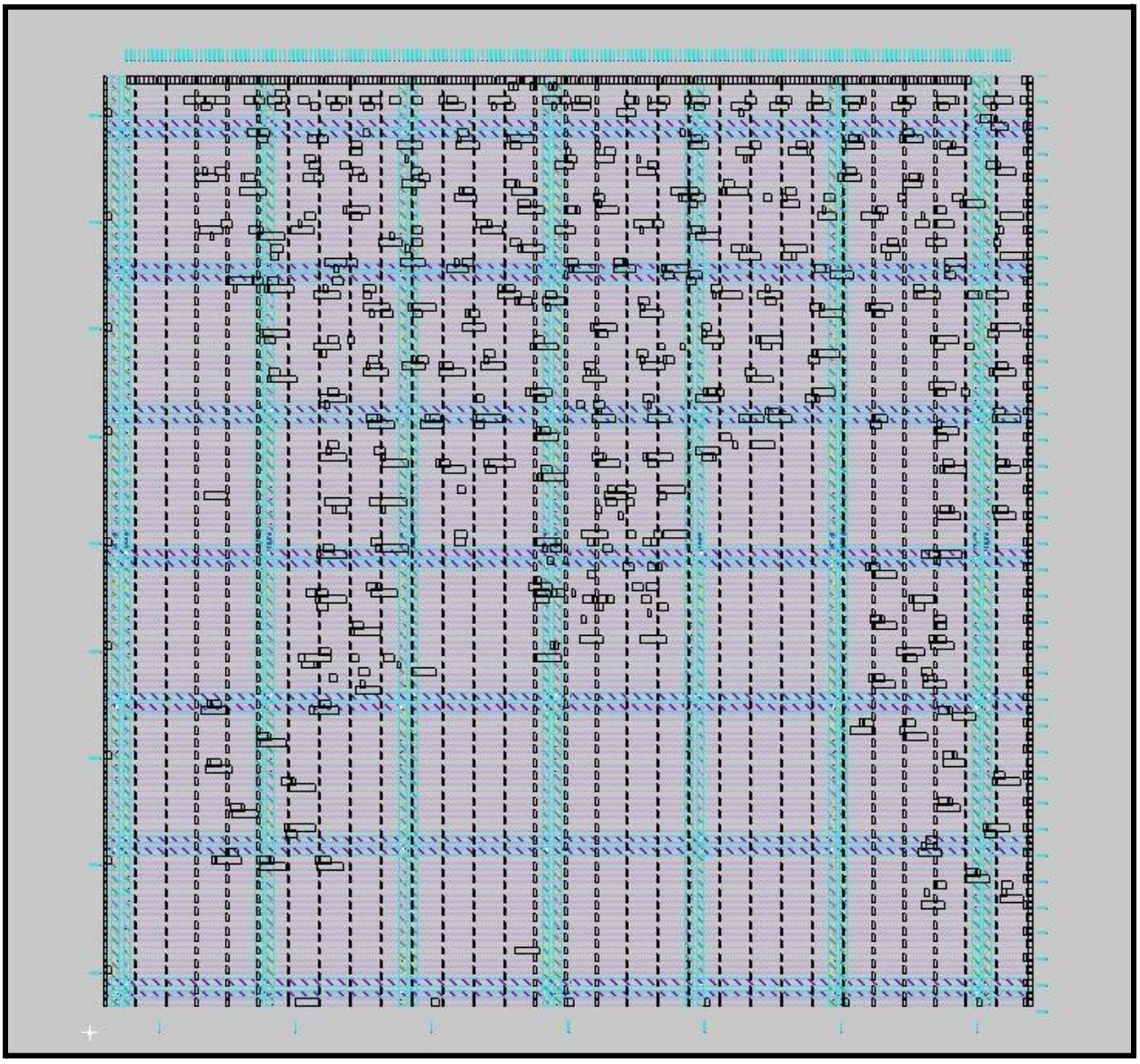
DEF file after Floorplan:



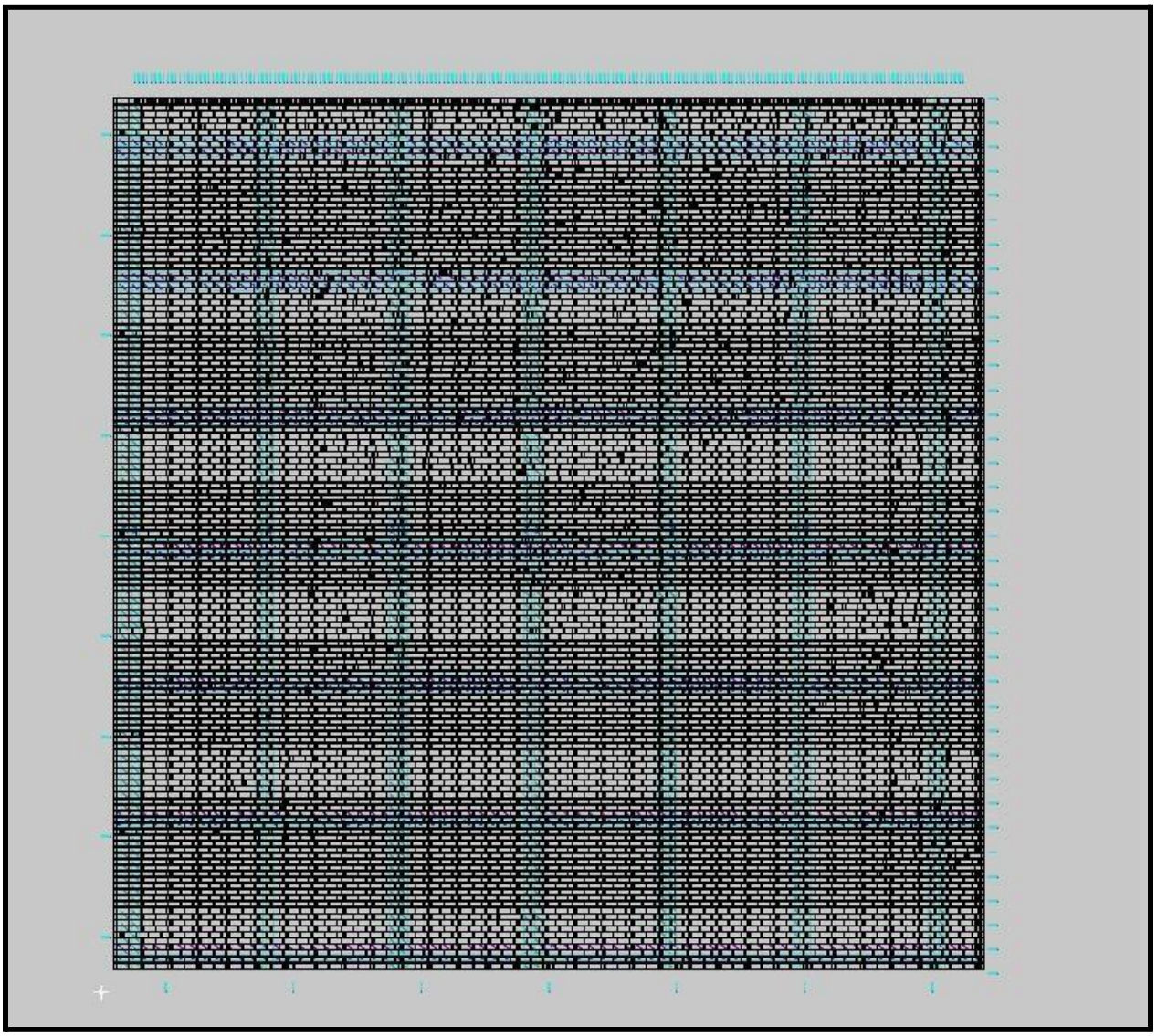
DEF file after Placement:



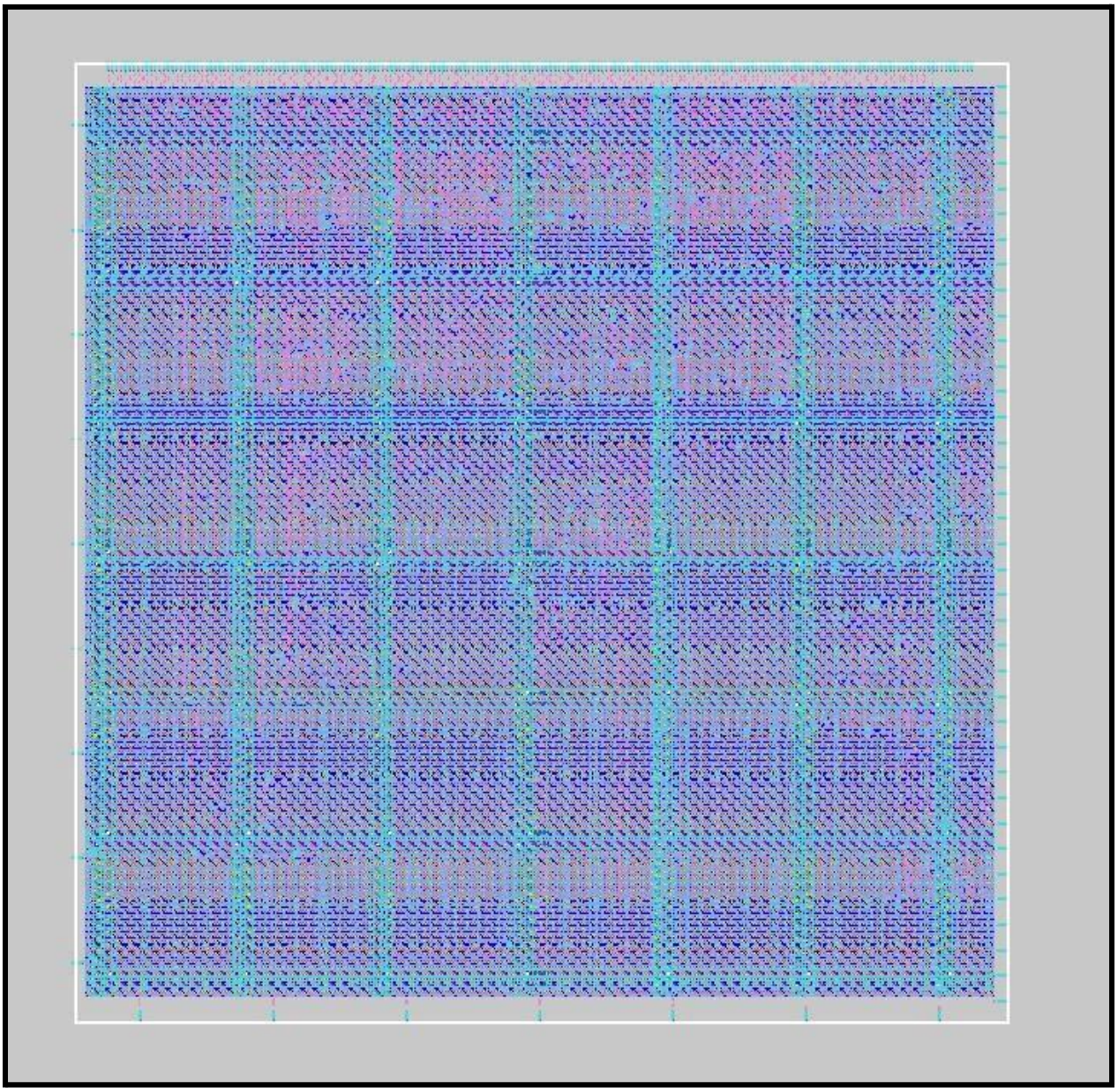
DEF file after CTS:



DEF file after Routing:

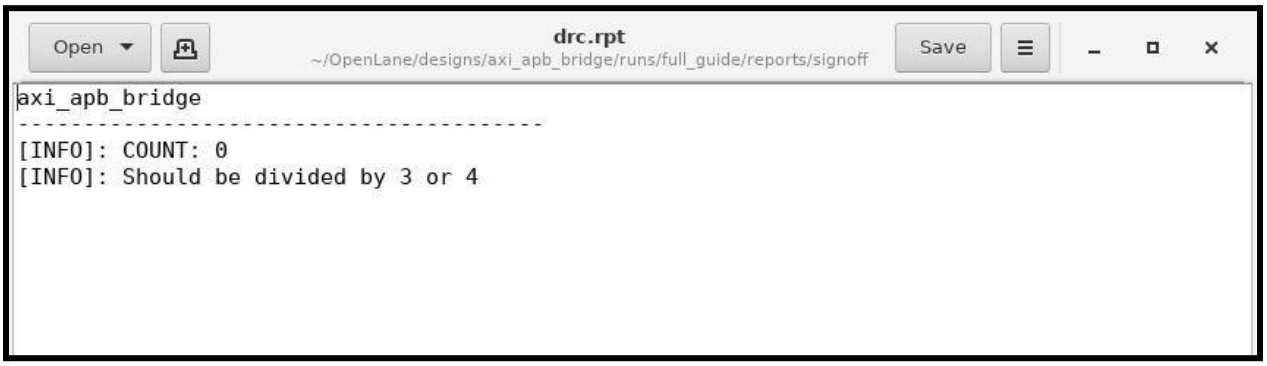


Final gds:



Screenshot of lvs.rpt:

Screenshot of drc.rpt:



Screenshot from grt_sta.log:

```
======================= Typical Corner ===================================

Group                  Internal  Switching    Leakage      Total
                          Power      Power      Power      Power (Watts)
----------------------------------------------------------------
Sequential             1.13e-03   1.13e-04   1.98e-09   1.25e-03  48.5%
Combinational          6.30e-04   2.63e-04   1.75e-09   8.94e-04  34.7%
Clock                  2.43e-04   1.91e-04   1.14e-09   4.33e-04  16.8%
Macro                  0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
Pad                    0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
----------------------------------------------------------------
Total                  2.01e-03   5.67e-04   4.87e-09   2.57e-03 100.0%
                          78.0%      22.0%       0.0%

power_report_end
skew_report

============================================================================
report_clock_skew
============================================================================
Clock core_clock
   0.31 source latency _0966_/CLK ^
  -0.31 target latency _0899_/CLK ^
   0.00 CRPR
--------------
   0.00 setup skew

skew_report_end
summary_report

============================================================================
report_tns
============================================================================
tns 0.00

============================================================================
report_wns
============================================================================
wns 0.00

============================================================================
report_worst_slack -max (Setup)
============================================================================
worst slack 7.92

============================================================================
report_worst_slack -min (Hold)
============================================================================
worst slack 0.52
summary_report_end
area_report

============================================================================
report_design_area
============================================================================
Design area 11845 u^2 8% utilization.
area_report_end
check_nonpropagated_clocks
check_nonpropagated_clocks_end
[WARNING] Did not save OpenROAD database!
```

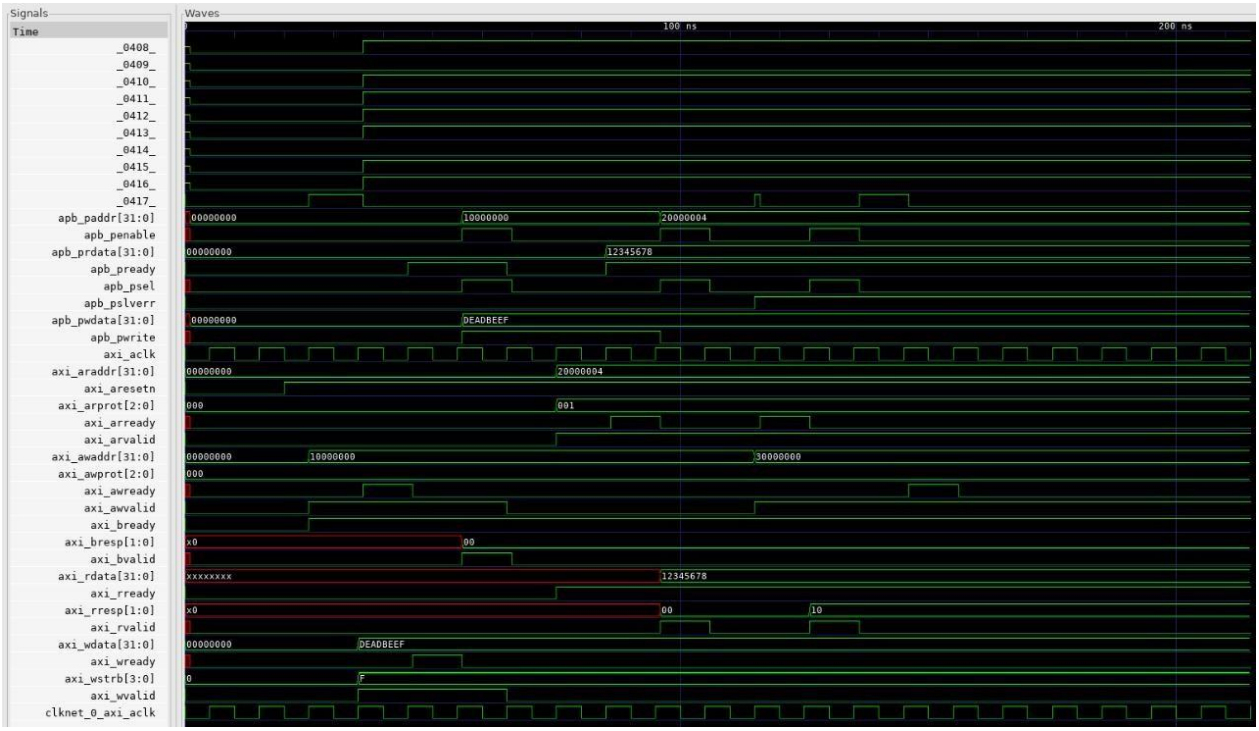
Table from grt_sta.log :

Clock Frequency (MHz)	**83.33**
Worst case setup slack (ns)	**7.92**
Worst case hold slack (ns)	**0.52**
Design area (um2)	**11845**
Power consumption (sequential) (uW)	**1250**
Power consumption (combinational) (uW)	**894**
Power consumption (clock) (uW)	**433**
Total power consumption (uW)	**2577**

Output for Testbench:

```
VCD info: dumpfile axi_apb_bridge_tb.vcd opened for output.
Write Address Transaction: ADDR = 10000000
Read Address Transaction: ADDR = 20000004
Read Data Transaction: DATA = 12345678, RESP = 00
Read Address Transaction: ADDR = 20000004
Read Data Transaction: DATA = 12345678, RESP = 10
Write Address Transaction: ADDR = 30000000
```

Screenshot of gtk waveform of dump:



Contribution Table:

Task	Naman	Sahil	Abhiman	Ratnesh
Design of AXI toAPB Bridge in Verilog	Architecture design, AXI toAPB mapping	AXI write andread handling	APB interface design, address decoding	Control logic and synchronization
Module Implementation	Implemented AXI read/write handlers	Implemented APB write/read commands	Implemente daddress mapping	Data conversionand final integration
Testbench Design& Verification	Created AXI master model, stimulus generation	Created APB slave model andverified	Wrote test cases	Verified test casesand timing