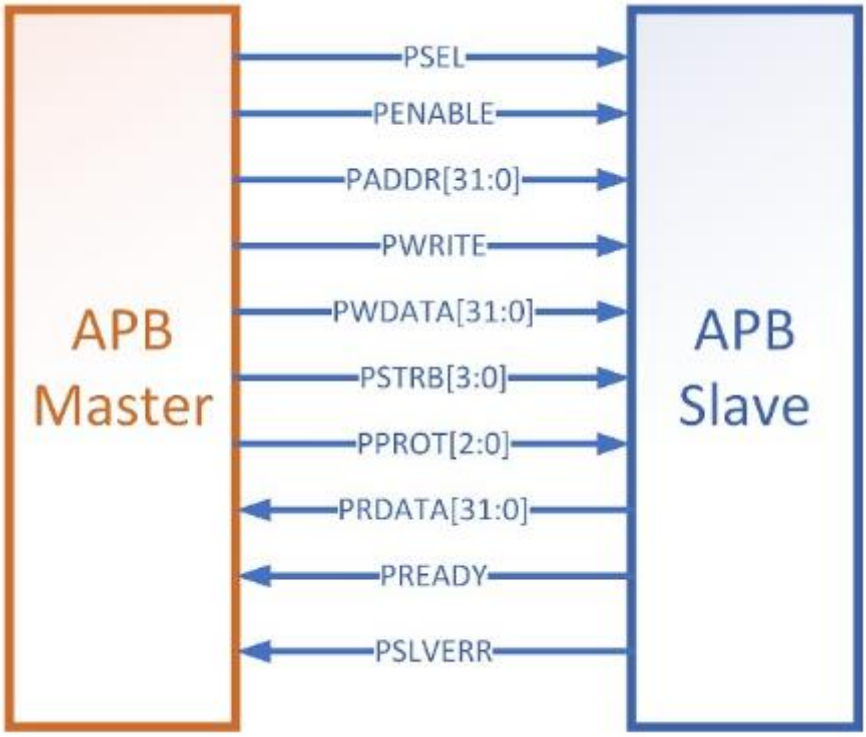
# AMBA APB Protocol

# (APB4)




## Created By:

**Mohamed Ahmed Mohamed Hussein** **25/8/2024**

# Introduction to AMBA APB Protocol:

The AMBA (Advanced Microcontroller Bus Architecture) APB (Advanced Peripheral Bus) protocol is a low-complexity, low-power interface used in modern System-on-Chip (SoC) designs. As part of the AMBA family of protocols, the APB is specifically optimized for communication with peripherals that do not demand the high performance or complexity of buses like AHB (Advanced High-performance Bus) or AXI (Advanced eXtensible Interface).

# Importance of the AMBA APB Protocol:

The AMBA APB protocol is integral to efficient system design in embedded systems and microcontroller architectures due to its simplicity and power efficiency. By providing a straightforward, non-pipelined communication method, APB allows designers to connect and manage various low-speed peripherals such as timers, UARTs, GPIOs, and more without the overhead associated with more complex bus protocols.

One of the key advantages of the APB protocol is its ability to minimize power consumption, which is critical in battery-operated devices and applications where power efficiency is a priority. The protocol's non-pipelined nature and single-clock edge operation simplify timing requirements, making it easier to integrate with other system components.

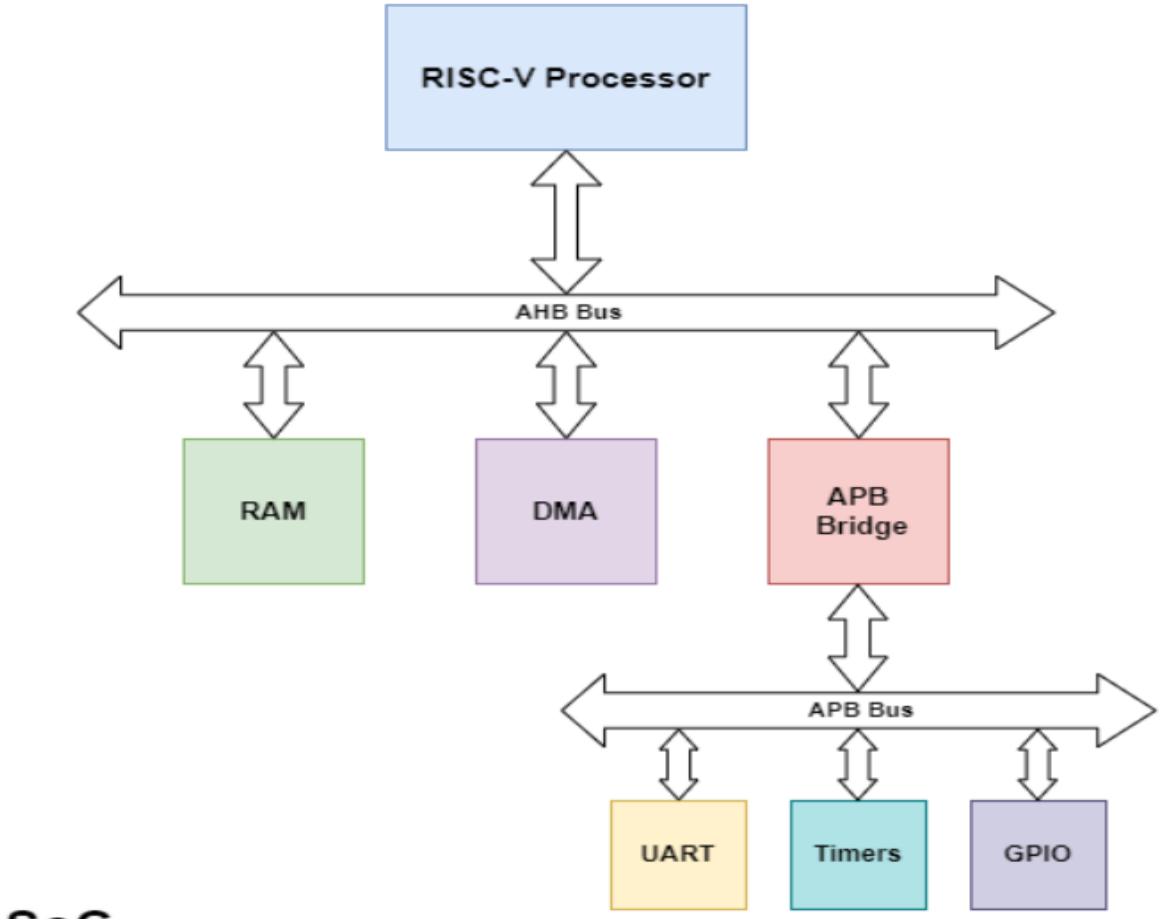
# Applications of the AMBA APB Protocol:

The AMBA APB protocol is widely used in various applications within SoCs, particularly in areas where simplicity and low power are essential. Common applications include:

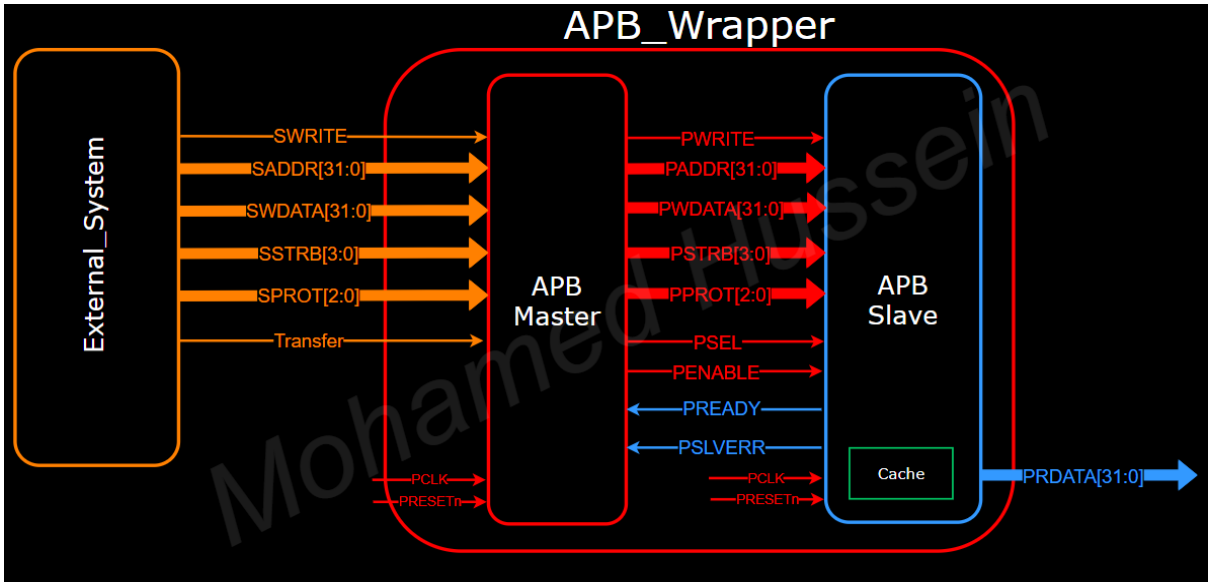
- **Peripheral Connectivity:** APB is ideal for connecting low-speed peripherals like GPIO controllers, UARTs, and timers to the main processor. These peripherals typically do not require high-speed data transfers, making APB a perfect fit.
- **Configuration Registers:** Many SoCs use APB for accessing configuration registers that control various aspects of the chip's operation. The low-bandwidth nature of these operations suits the APB's characteristics.
- **Power Management Modules:** APB is often used in power management and clock control modules within SoCs. Its low power consumption aligns with the requirements of these subsystems, ensuring efficient operation.
- **Debug and Control Interfaces**: APB is also used for debug interfaces, control interfaces, and other low-speed data paths within an SoC. Its simplicity makes it easy to implement and integrate into complex designs.

By providing a low-complexity, low-power bus interface, the AMBA APB protocol plays a crucial role in the efficient design and operation of modern embedded systems and SoCs. Its widespread adoption and applicability in various peripheral communication scenarios underscore its importance in the industry.

# SoC Application using AMBA protocols:




# Design Architecture:



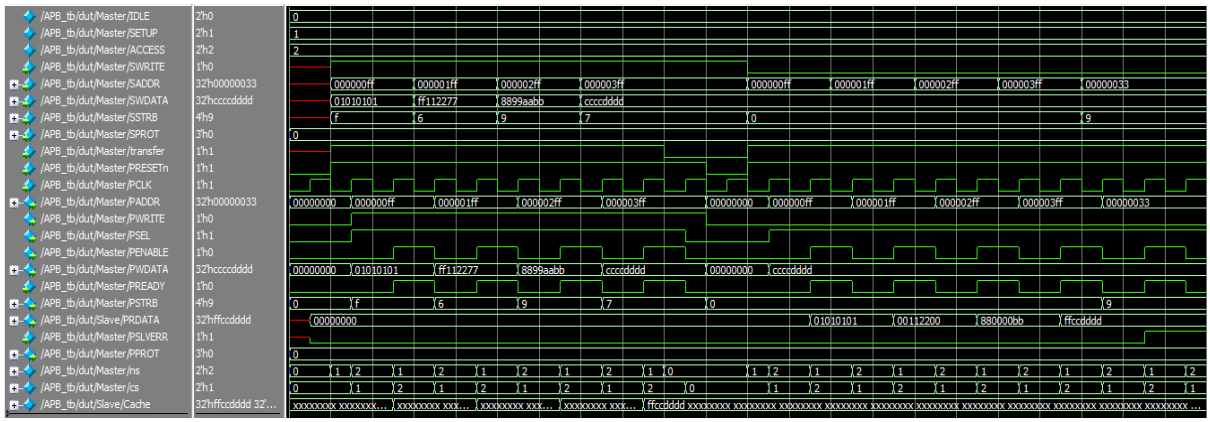
# Verifying Functionality:

In the test bench we are pretending to be the external system and we are sending the address, write data and the transfer signal to the APB master and then the master will the commands to the APB slave.

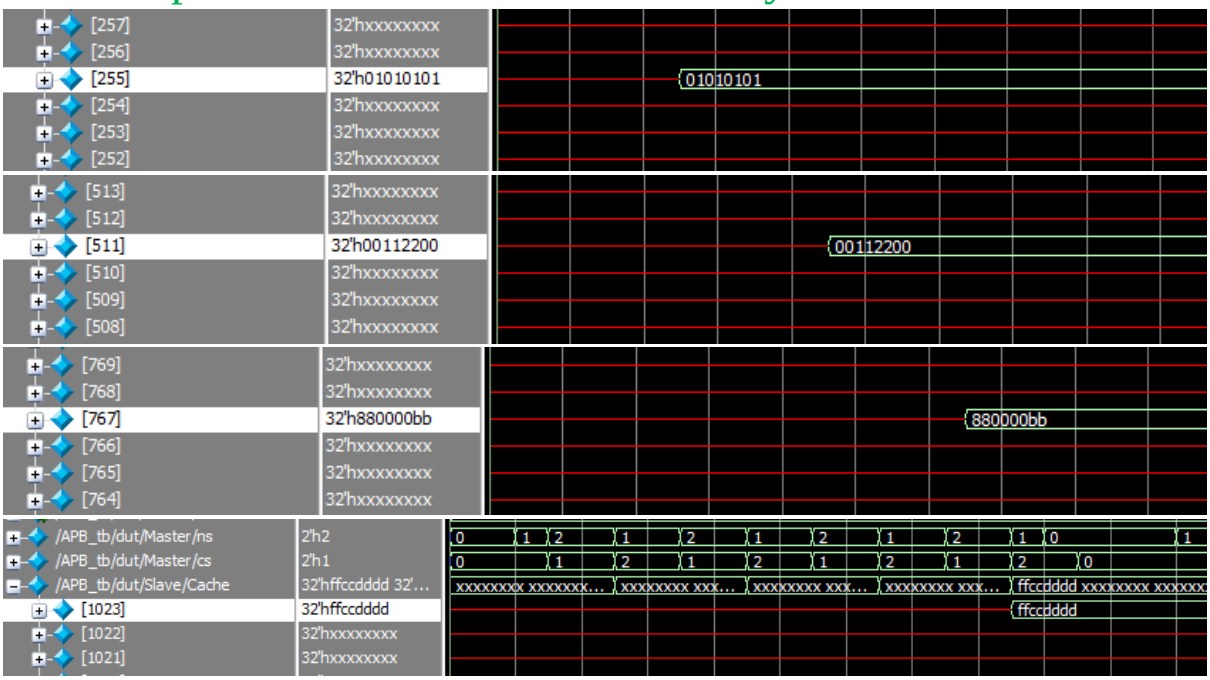
- **Scenario:**
  - **Writing**: We will write random data in several addresses and each address will have a different data to be written and also each address will have a different write strobe signal to verify its functionality.
  - **Reading**: After writing process comes the reading process in which we will read from the same addresses we wrote data into, and at the end there is a test to verify slave error signal functionality, so we are expecting the following:
    - PSEL, PENABLE signals are low in IDLE state
    - PSEL is high and PENABLE is low in SETUP state and SETUP state is only entered for one clock cycle
    - PSEL, PENABLE signals are high in ACCESS state
    - PREADY signal when high it lasts for one clock cycle and then turns back low
    - Note that the above specs can be verified through the **whole snippet** and the **comparison** in the next pages
    - The data values in each writing process to be written in cache memory in its specified address and the data will be written by their format specified by the PSTRB signal
    - The values written in memory will show in the PRDATA signal
    - Finally, the SLVERR signal will be high at the end due to our scenario

**Note:** we will see each one of the above observations in order in the **snippets** section

- **Snippets:**
  - Snapshot of whole wave form:

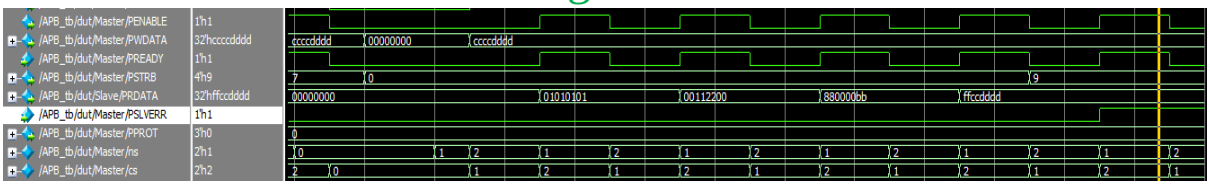



  - Snapshot of data written in memory in order:




    - Note that the values written in memory are different from the input values due to write strobe signal and last element stored has sign extension
  - Snapshot of the values written in memory showing up in **PRDATA** signal in order:

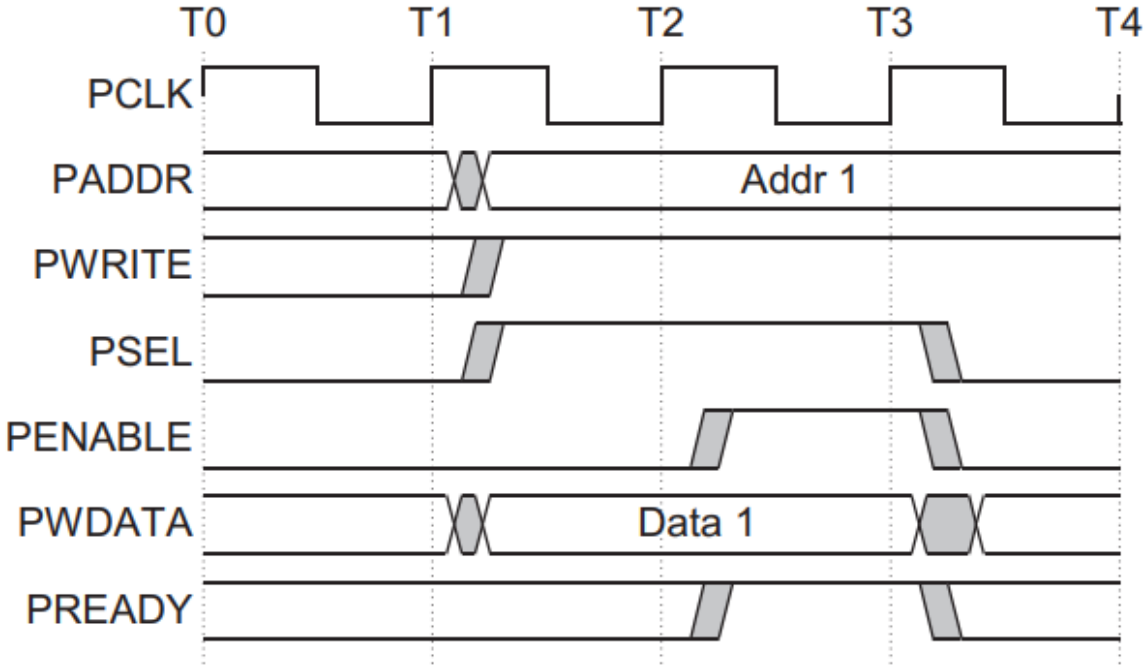
- Snapshot of **PSLVERR** high as **PSTRB** signal has some value while reading:




# • Comparison:

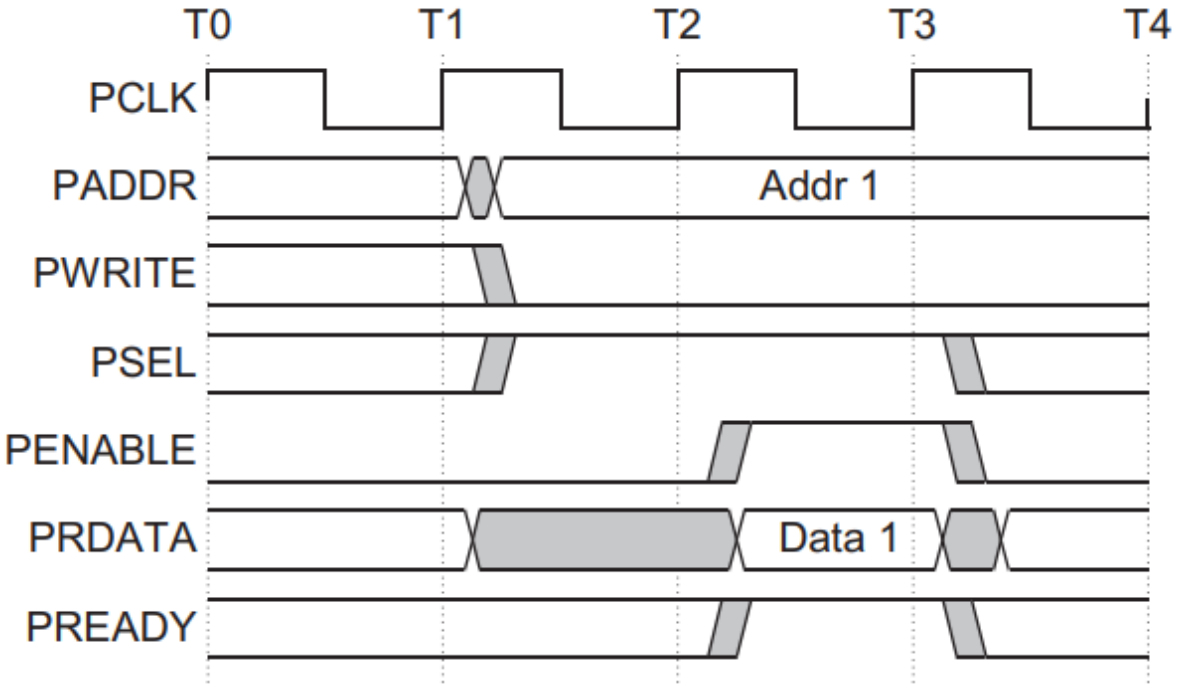
## ○ Writing Process:

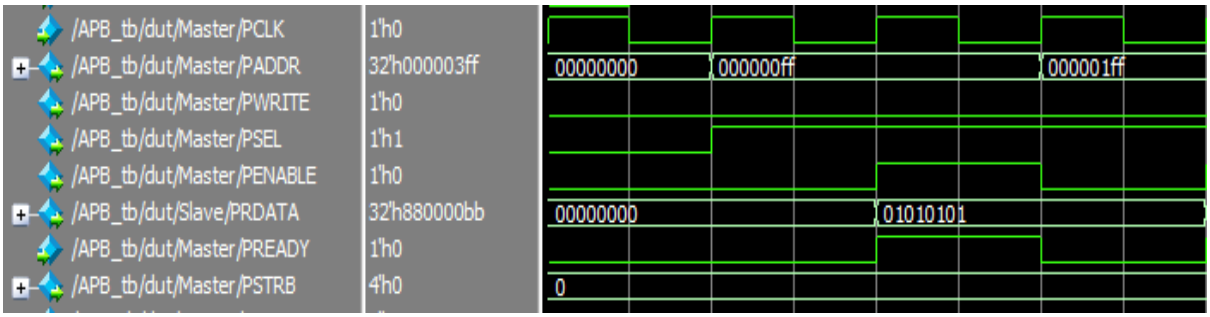
Comparing between the APB documentation wave form and my design wave form in writing process




- ## Reading Process:

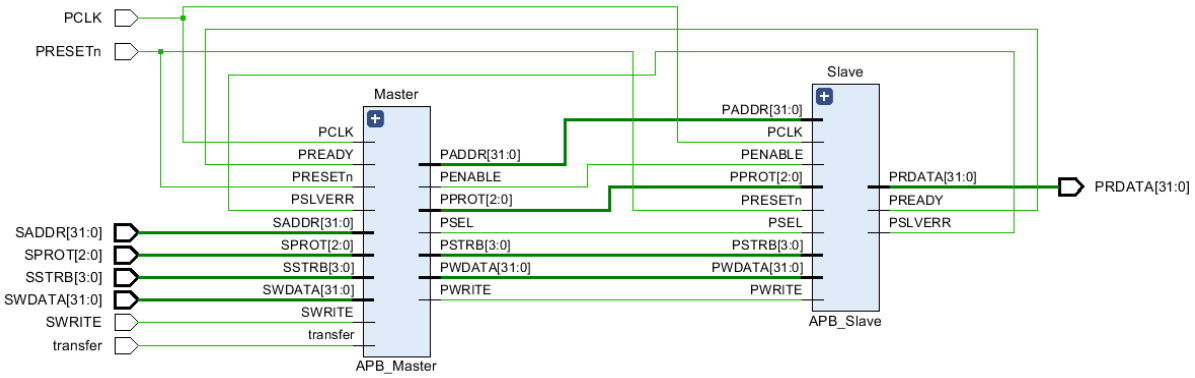
  Comparing between the APB documentation wave form and my design wave form in reading process



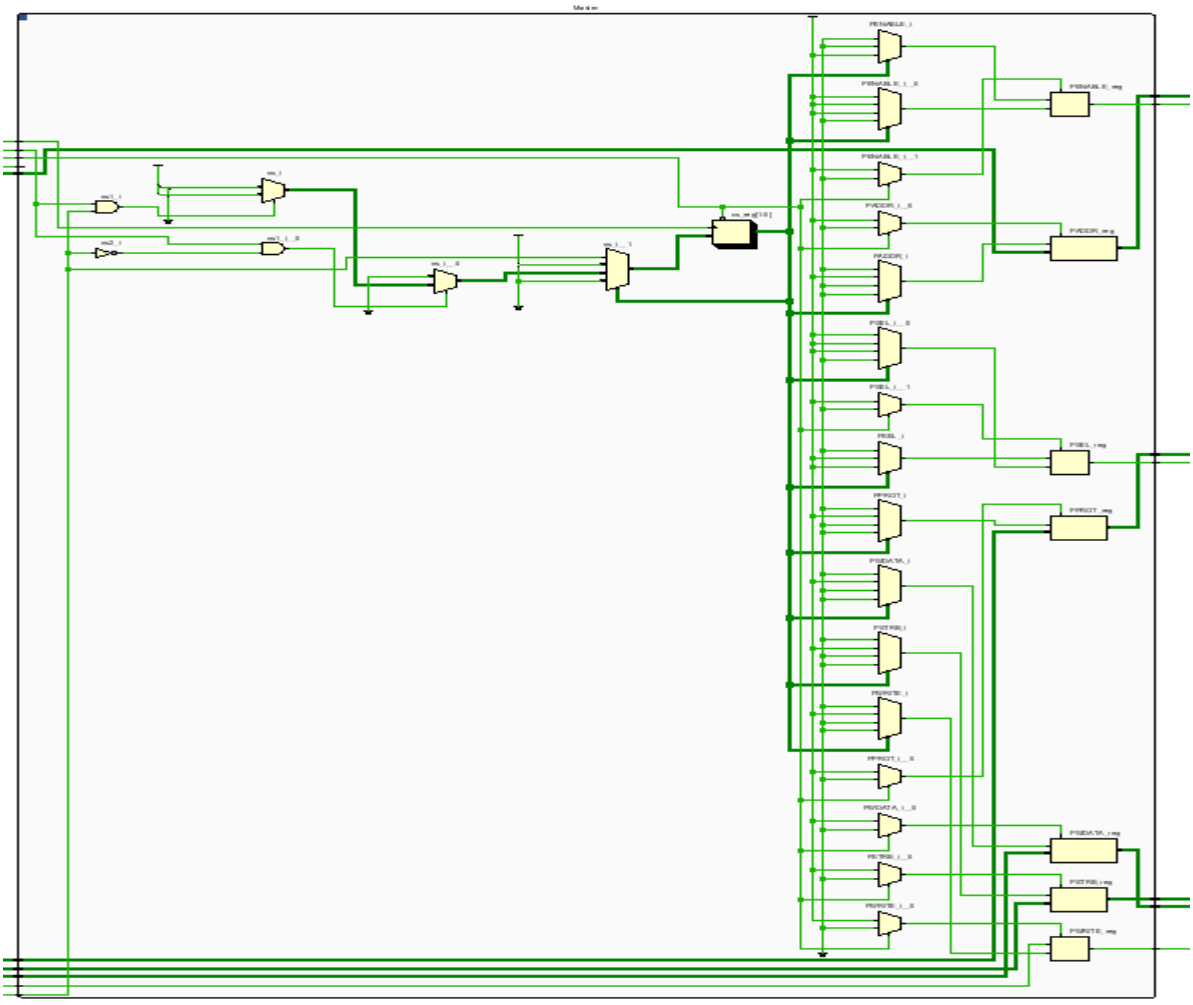



# Vivado:

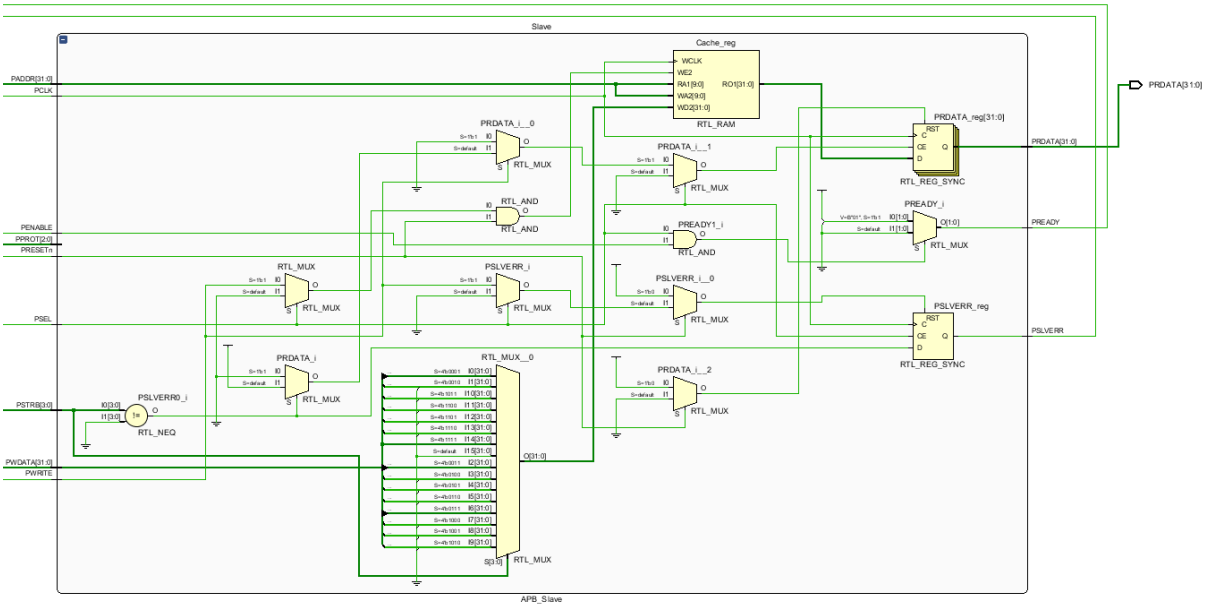
- Elaboration:
  - System:



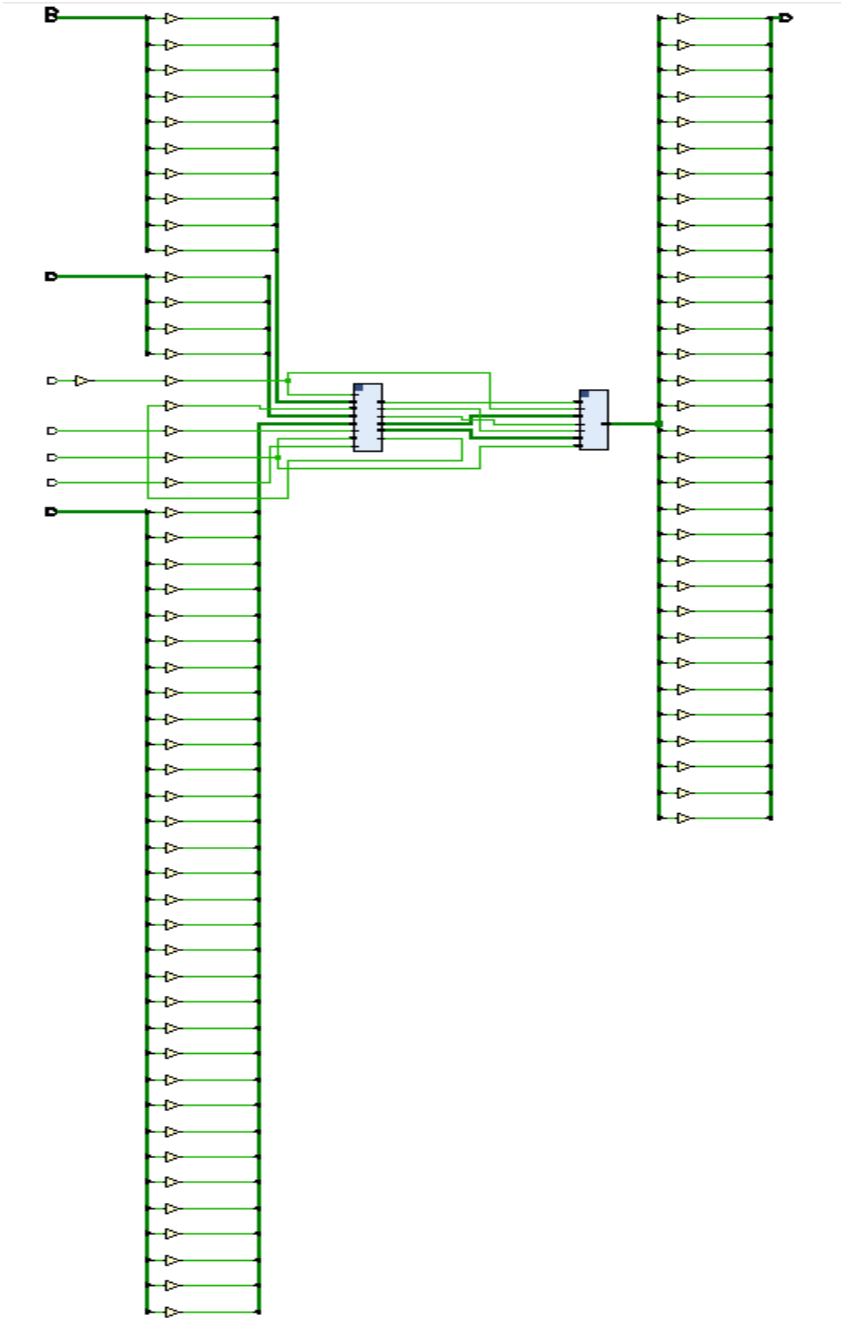

  - Master:



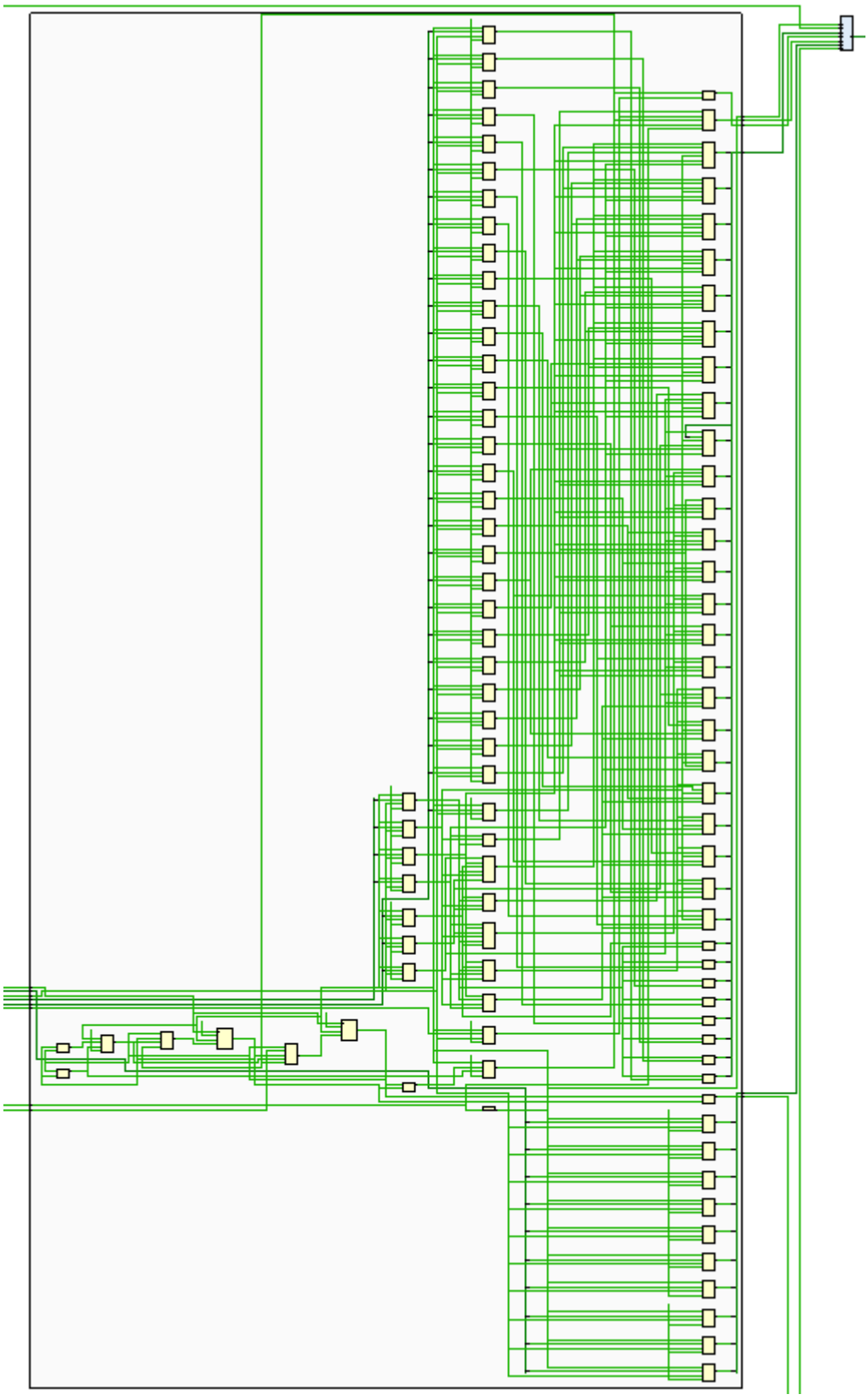
- Slave:


Slave
Cache_reg
WCLK
WE2
RA1[9:0]
RO1[31:0]
WA2[9:0]
WD2[31:0]
RTL_RAM
PADDR[31:0]
PCLK
PENABLE
PPROT[2:0]
PRESETn
PSEL
PSTRB[3:0]
PWDATA[31:0]
PWRITE
PRDATA_i__0
PRDATA_i__1
PRDATA_i__2
PRDATA_i
RTL_MUX
RTL_AND
PREADY1_i
PSLVERR_i
PSLVERR_i__0
RTL_MUX__0
PSLVERR0_i
RTL_NEQ
PRDATA_reg[31:0]
RTL_REG_SYNC
PREADY_i
PSLVERR_reg
PRDATA[31:0]
PREADY
PSLVERR
APB_Slave

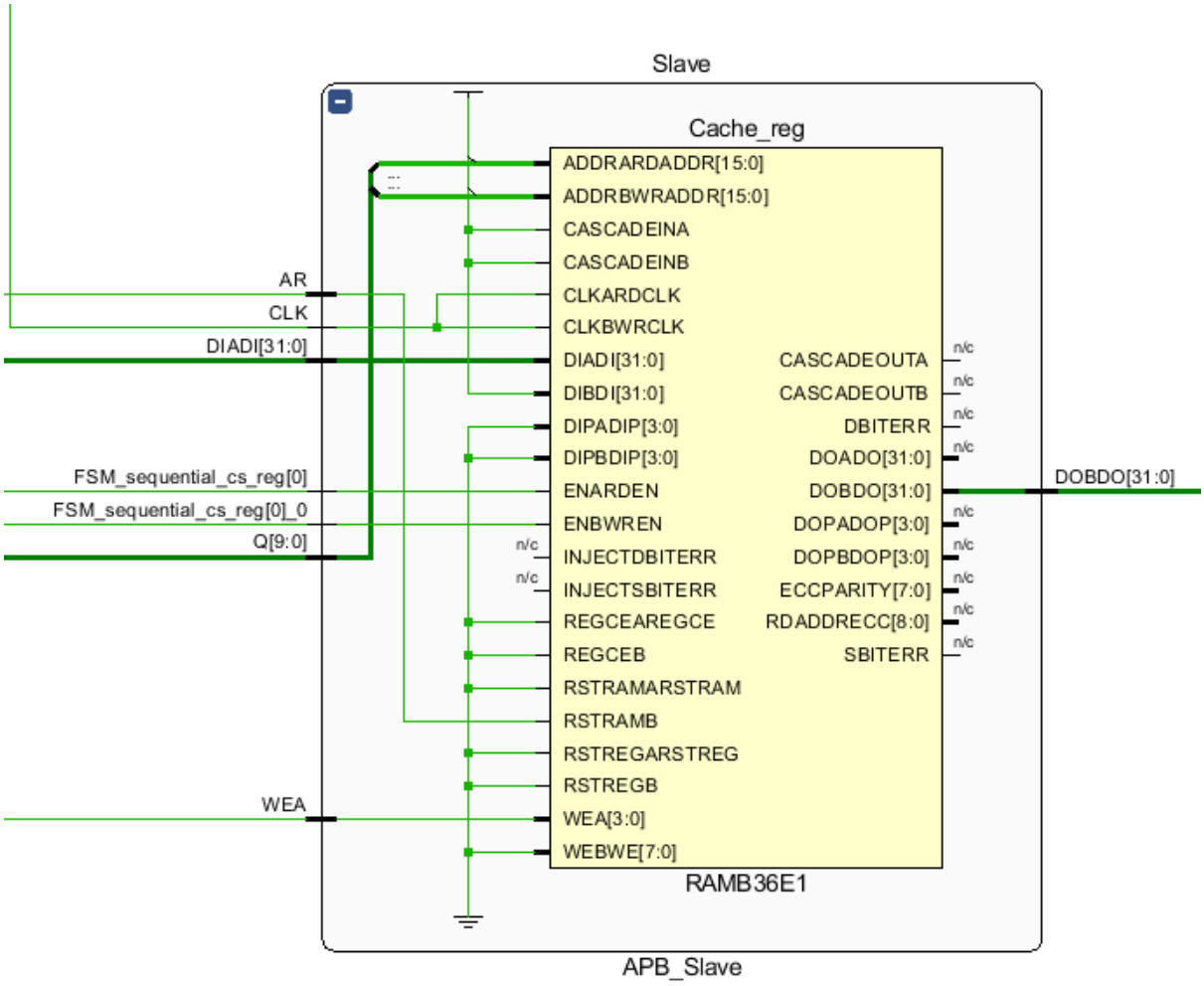
- **Synthesis:**
  - **Seq:**
    - Schematic:
      - System:



- Master:



- Slave:

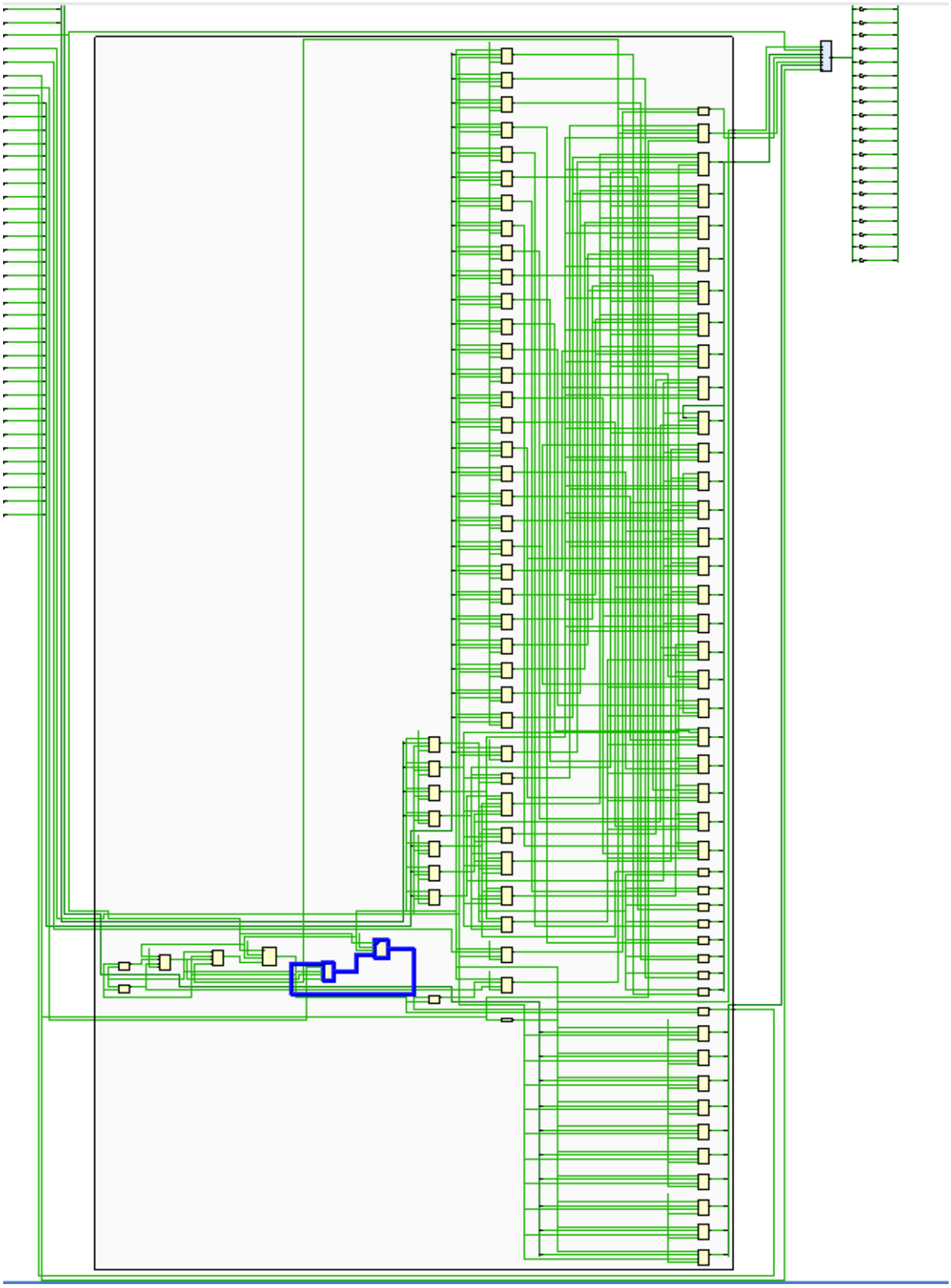



- Encoding Report:

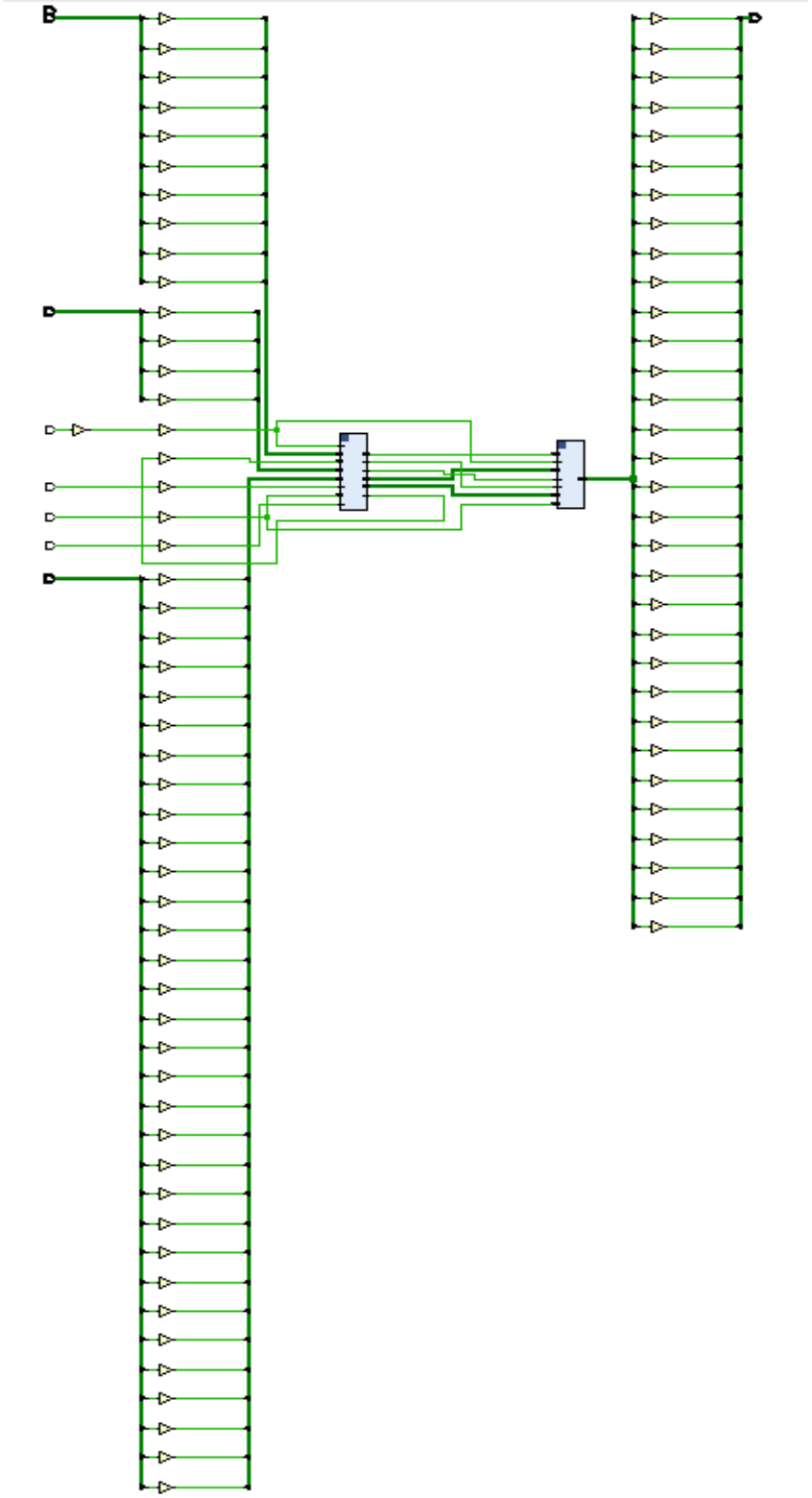
| State | New Encoding | Previous Encoding |
|---|---|---|
| IDLE | 00 | 00 |
| SETUP | 01 | 01 |
| ACCESS | 10 | 10 |

- Timing Summary on 10 ns clock period:

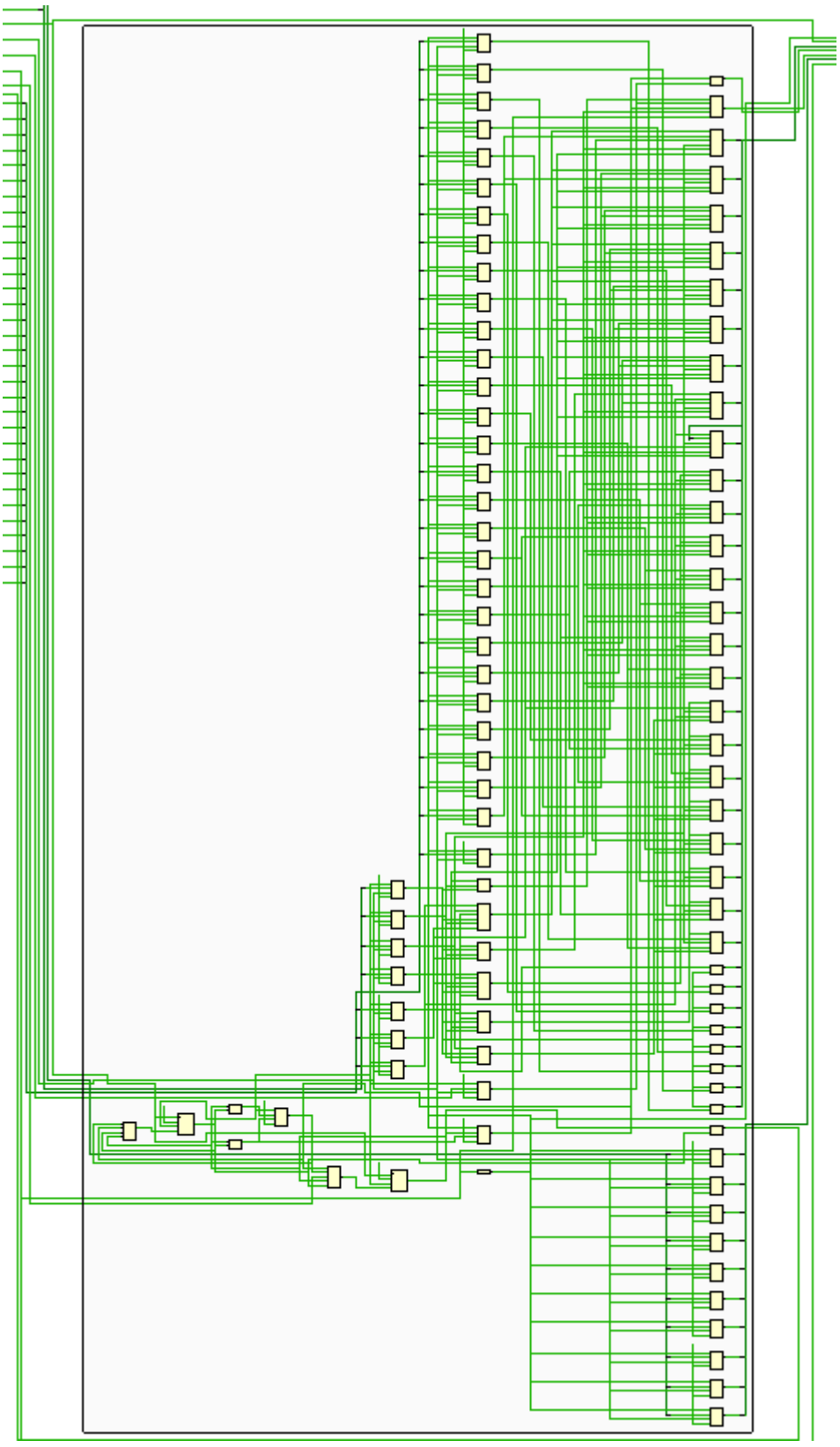
- Critical Path:



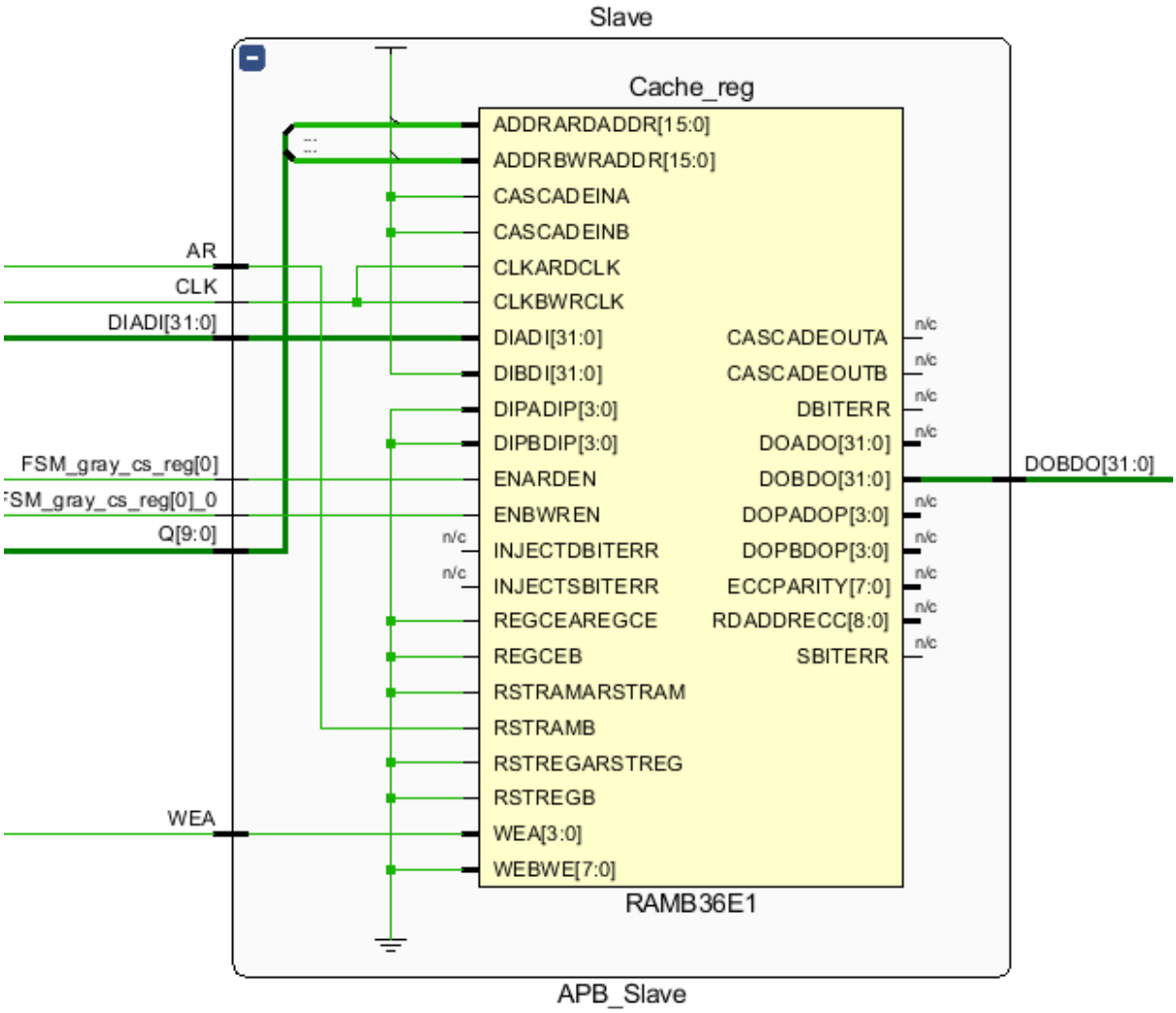
- **Gray:**
  - Schematic:
    - System:



- Master:



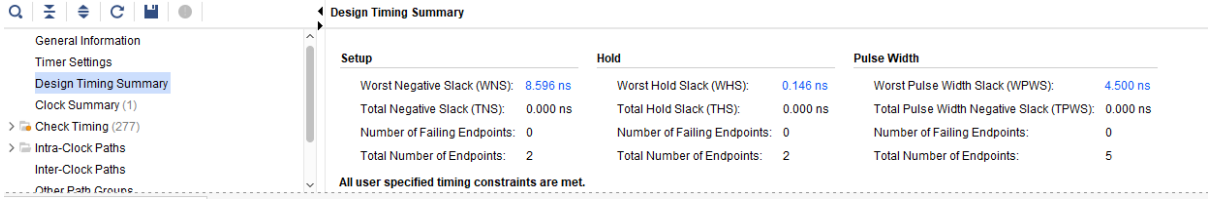
- Slave:



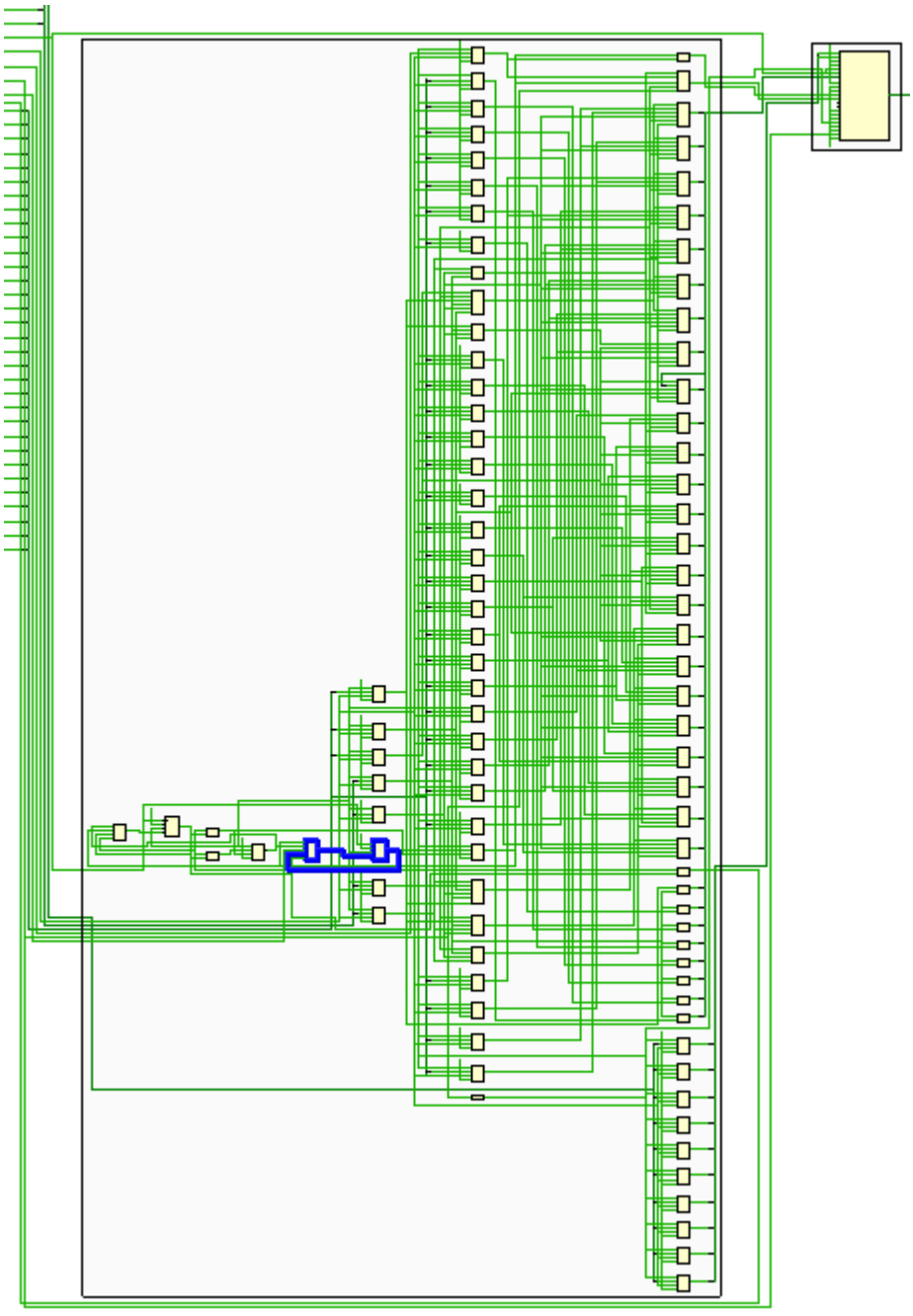

- Encoding Report:

| State | New Encoding | Previous Encoding |
|---|---|---|
| IDLE | 00 | 00 |
| SETUP | 01 | 01 |
| ACCESS | 11 | 10 |

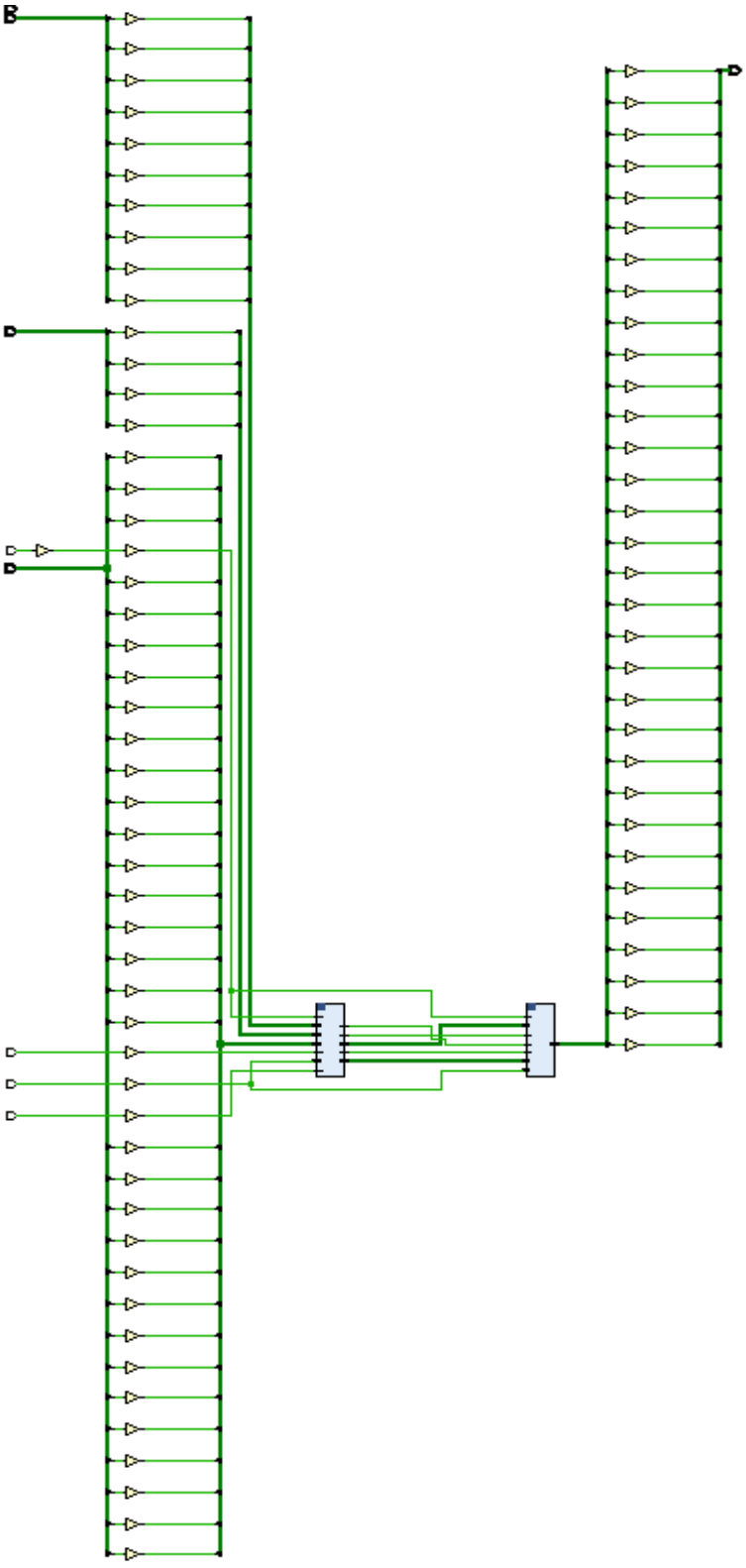
- Timing Summary on 10 ns clock period:



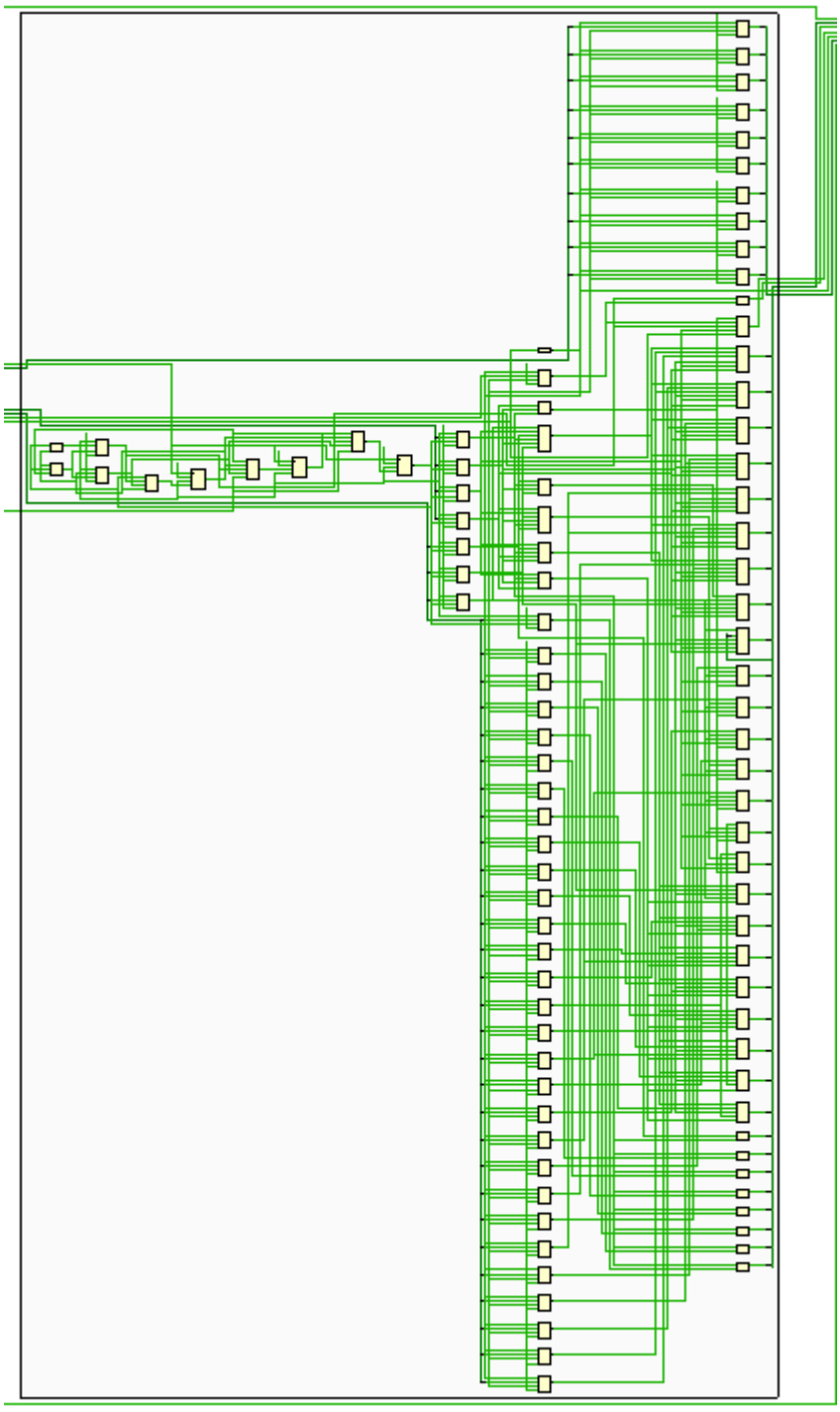
- Critical Path:



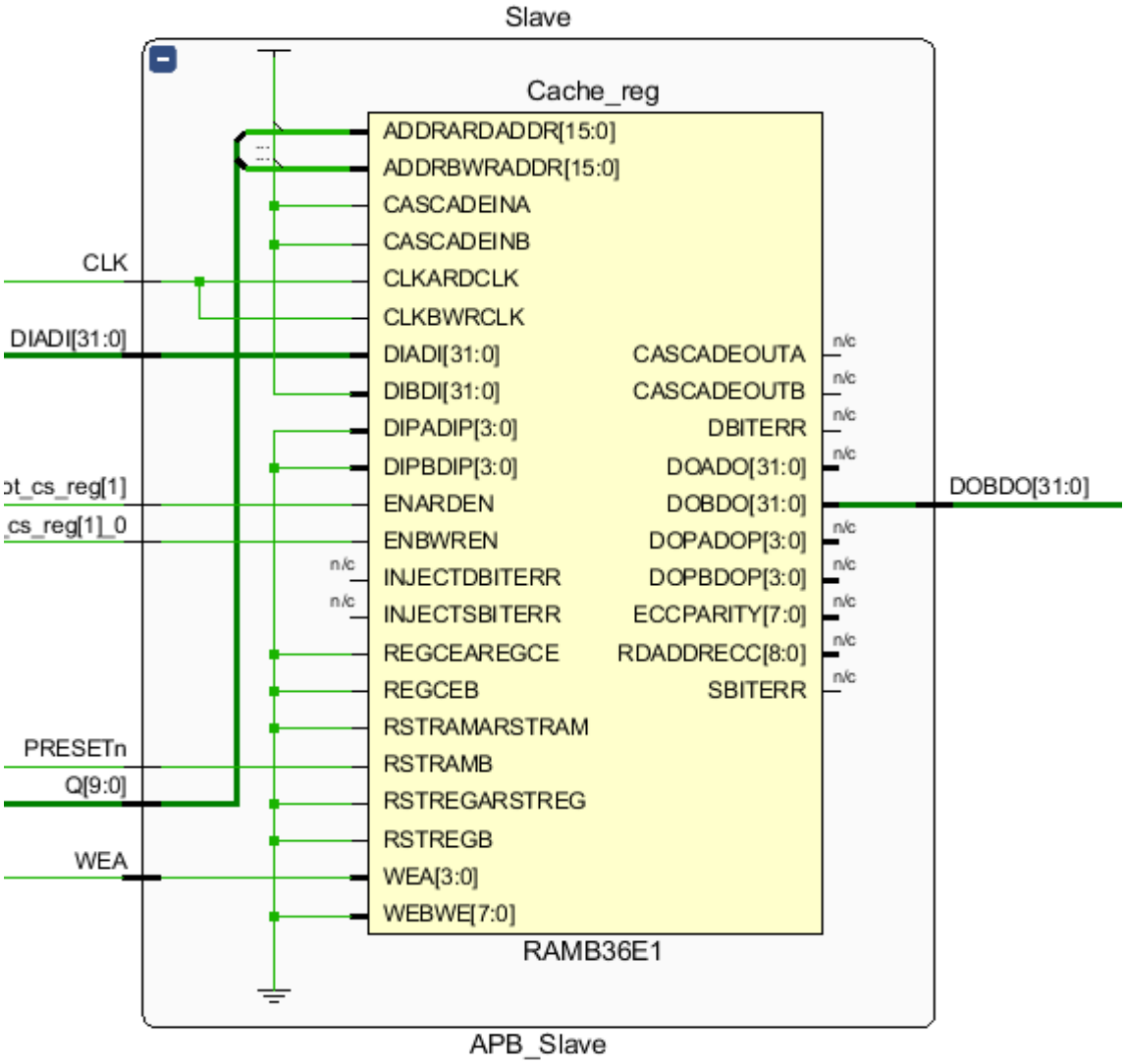
- **One_Hot:**
  - Schematic:
    - System:



- Master:



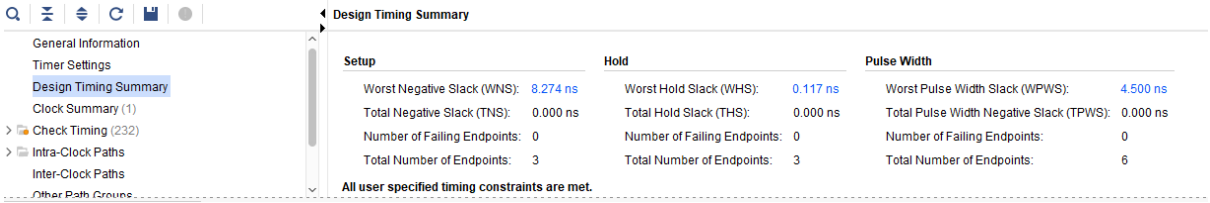
- Slave:



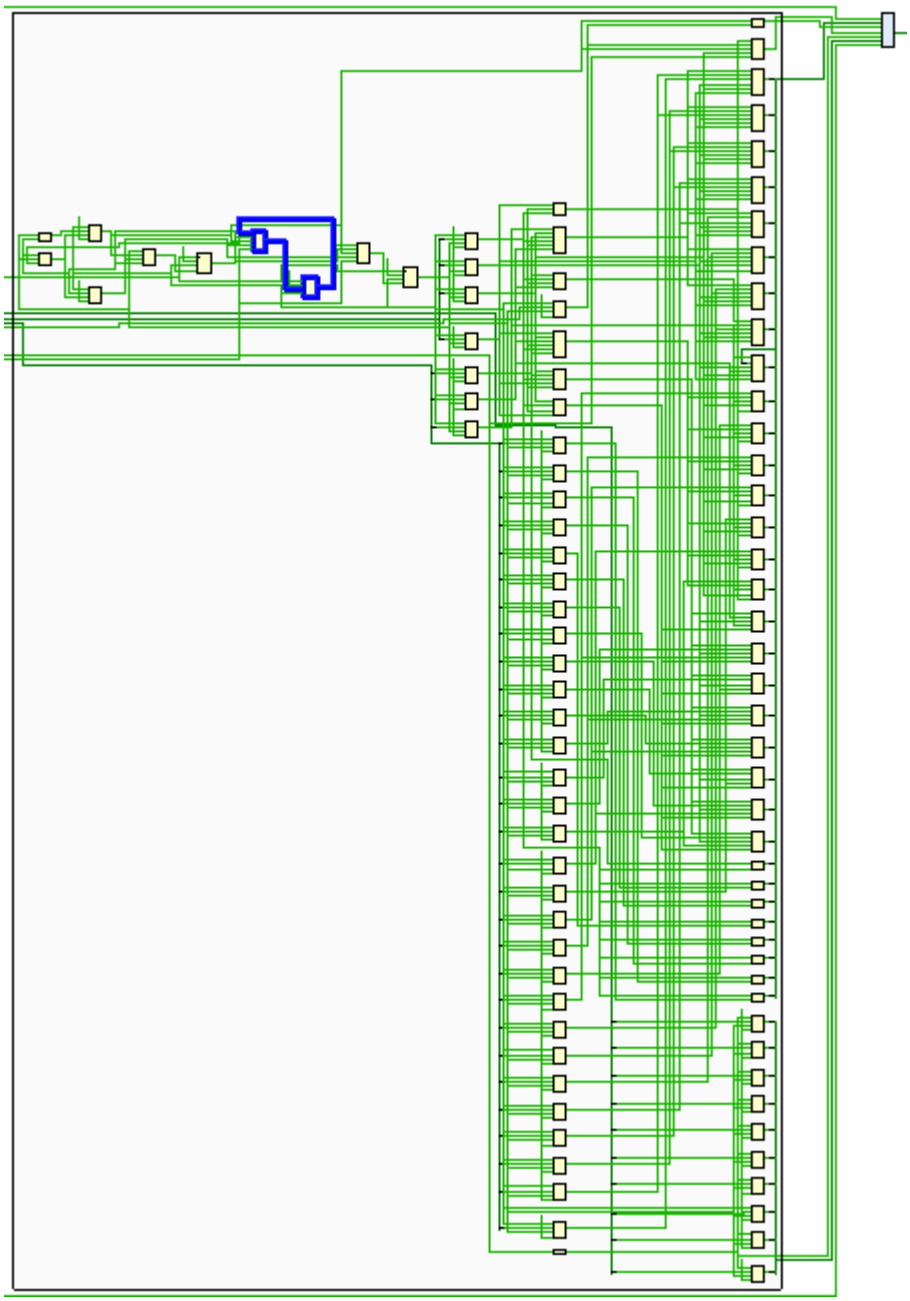

- Encoding Report:

| State | New Encoding | Previous Encoding |
|---|---|---|
| IDLE | 001 | 00 |
| SETUP | 010 | 01 |
| ACCESS | 100 | 10 |

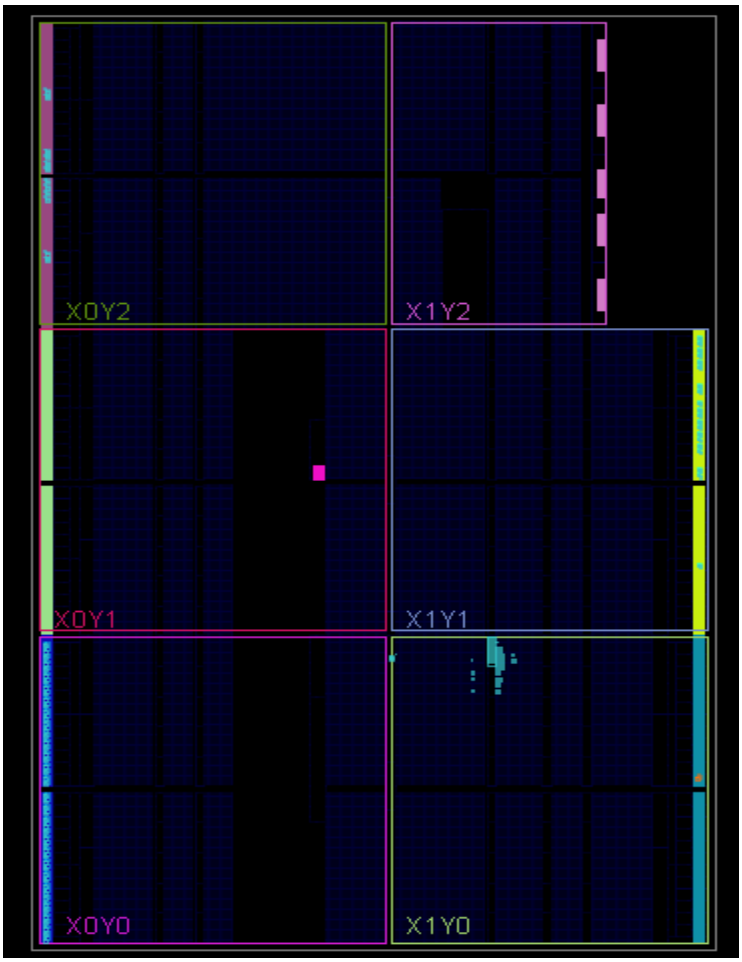
- Timing Summary on 10 ns clock period:



- Critical Path:



- # Implementation:
  - ## Seq:
    - ### Schematic:



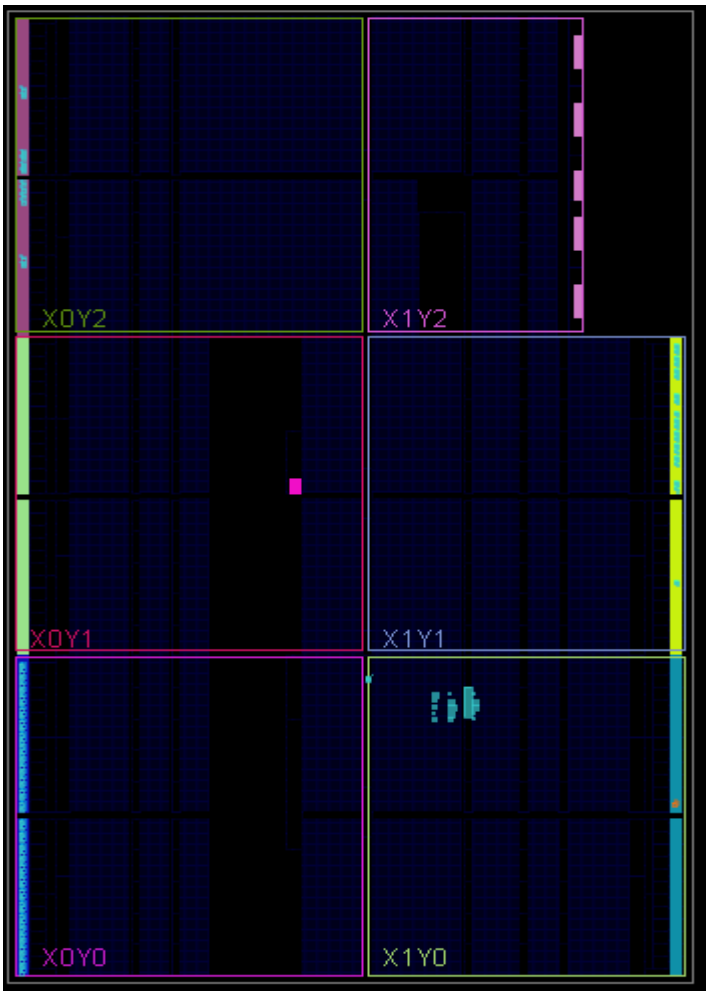

    - ### Timing on 10 ns clock period:


    - ### Utilization:

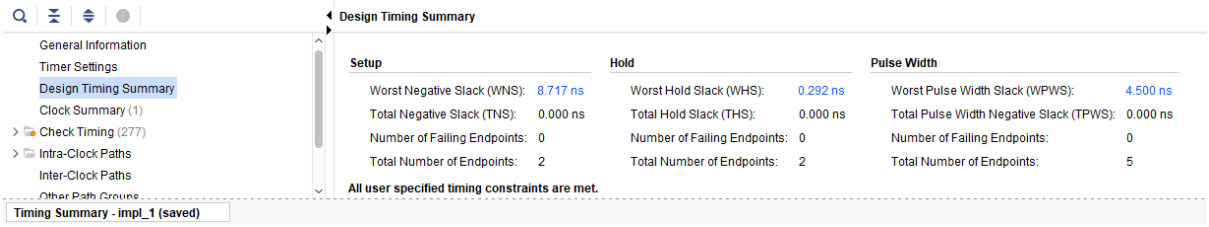
| Name | Slice LUTs (20800) | Slice Registers (41600) | Slice (8150) | LUT as Logic (20800) | LUT Flip Flop Pairs (20800) | Block RAM Tile (50) | Bonded IOB (106) | BUFGCTRL (32) |
|---|---|---|---|---|---|---|---|---|
| N APB_Wrapper | 44 | 51 | 22 | 44 | 4 | 1 | 82 | 2 |
| Master (APB_Master) | 44 | 51 | 22 | 44 | 4 | 0 | 0 | 0 |
| Slave (APB_Slave) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

## ○ Gray:

### ▪ Schematic:




### ▪ Timing on 10 ns clock period:



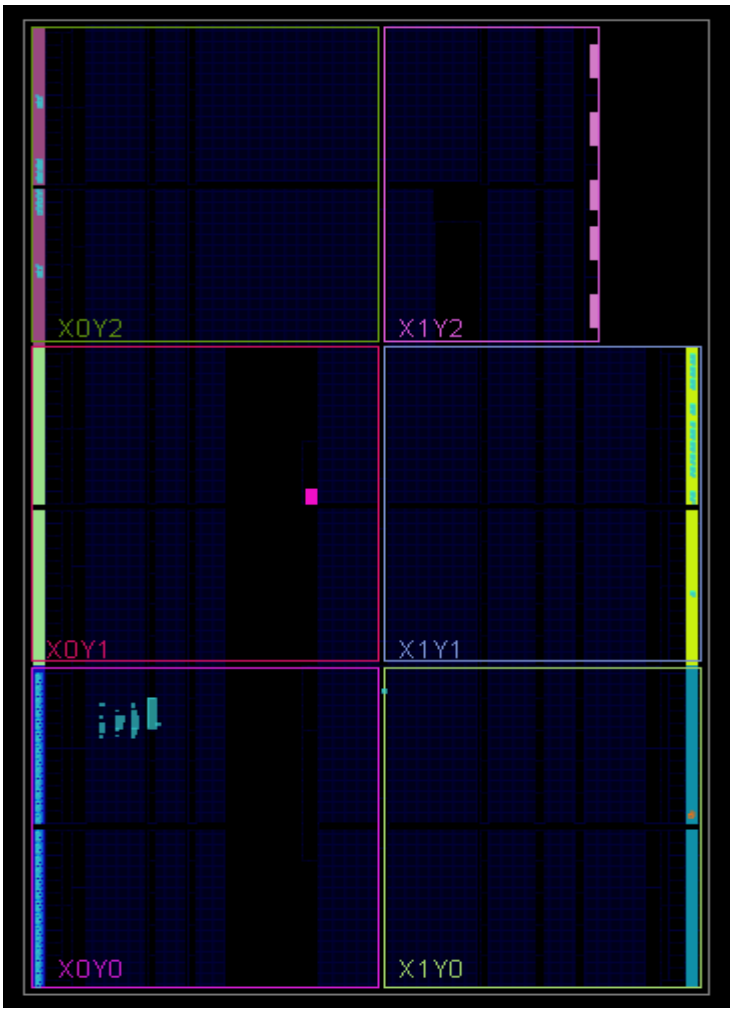

### ▪ Utilization:

Hierarchy

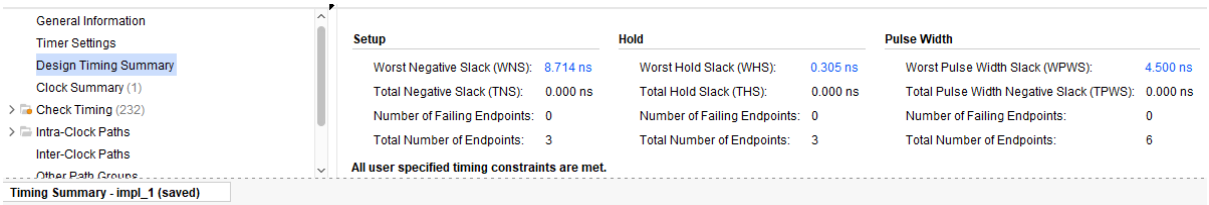
| Name | Slice LUTs (20800) | Slice Registers (41600) | Slice (8150) | LUT as Logic (20800) | LUT Flip Flop Pairs (20800) | Block RAM Tile (50) | Bonded IOB (106) | BUFGCTRL (32) |
|---|---|---|---|---|---|---|---|---|
| N APB_Wrapper | 42 | 51 | 26 | 42 | 2 | 1 | 82 | 2 |
| Master (APB_Master) | 42 | 51 | 26 | 42 | 2 | 0 | 0 | 0 |
| Slave (APB_Slave) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

## ○ One_Hot:

### ▪ Schematic:




### ▪ Timing on 10 ns clock period:




### ▪ Utilization:

Hierarchy

| Name | Slice LUTs (20800) | Slice Registers (41600) | Slice (8150) | LUT as Logic (20800) | LUT Flip Flop Pairs (20800) | Block RAM Tile (50) | Bonded IOB (106) | BUFGCTRL (32) |
|---|---|---|---|---|---|---|---|---|
| N APB_Wrapper | 44 | 52 | 26 | 44 | 3 | 1 | 82 | 1 |
| I Master (APB_Master) | 44 | 52 | 26 | 44 | 3 | 0 | 0 | 0 |
| I Slave (APB_Slave) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

**From results we found that the best encoding is seq.**

# References:

AMBA APB Protocol Specification