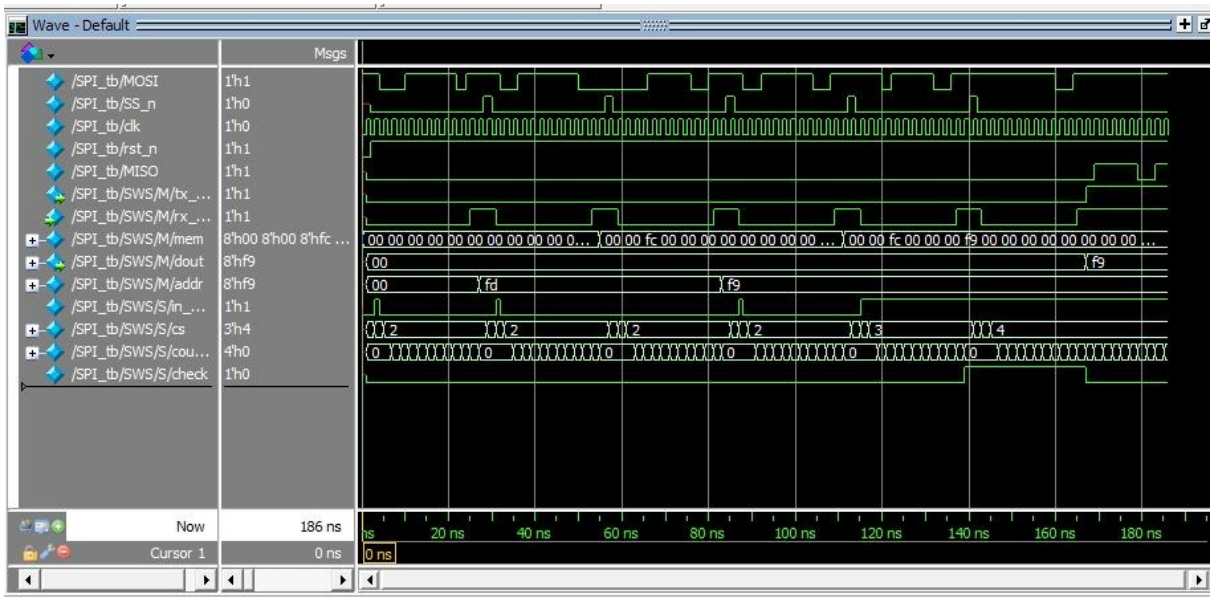
SPI Slave With single Port RAM

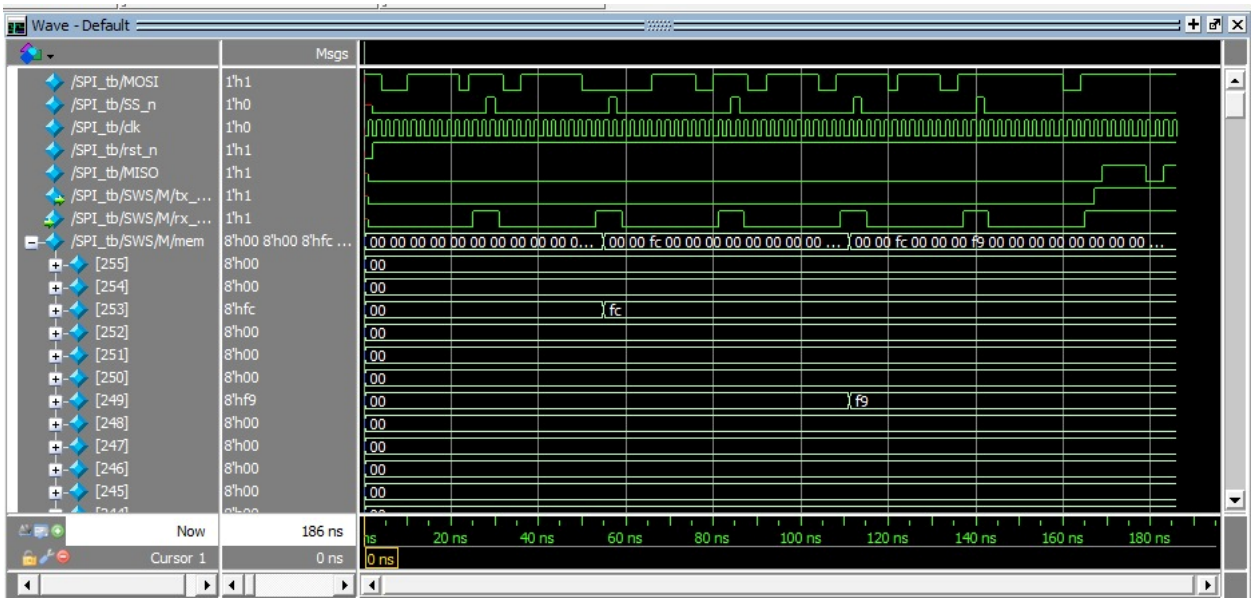
FPI Team

Supervised by: Eng Karim Waseem

Team members
Abdelrahman Mohamed Hassan Ali
Sherif Shawky Gaafar Mahmoud
Khaled Ali Elsayed Ali

1-Waveform snippets:





2-QuestaLint snippets:

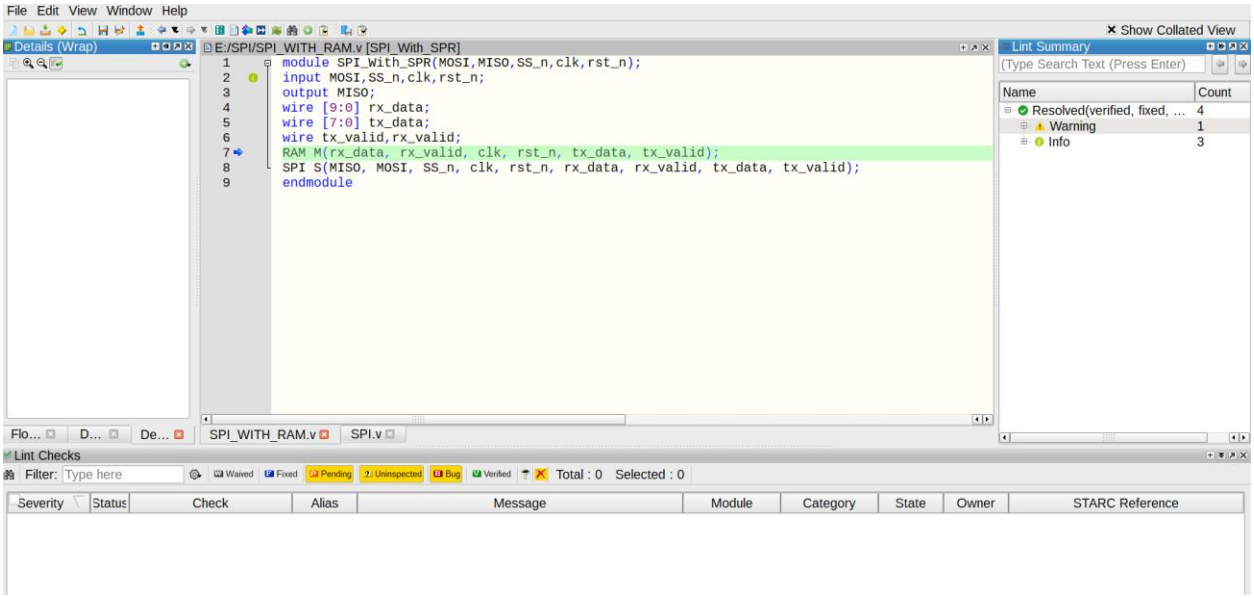
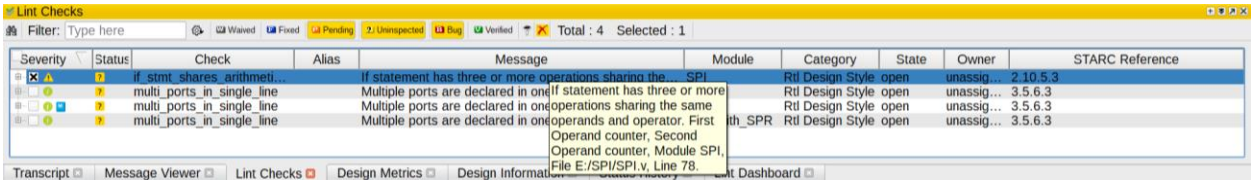


Figure 1: snippet from QuestaLint showing no linting Errors

Lint checks had only one warning as shown below:




```
70          end
71          else if(SS_n==0 && cs==WRITE)begin
                                          2
72              if(counter==10)begin
73              rx_valid<=1;
74              counter<=0;
75          end
76              else begin
77              rx_data<={rx_data[8:0],MOSI};
78              counter<=counter+1;
79          end
```

There was no problem with this warning so we just waived it and the design now has no Errors as shown in figure 1.

3-Synthesis snippets for each encoding:

1- Sequential :

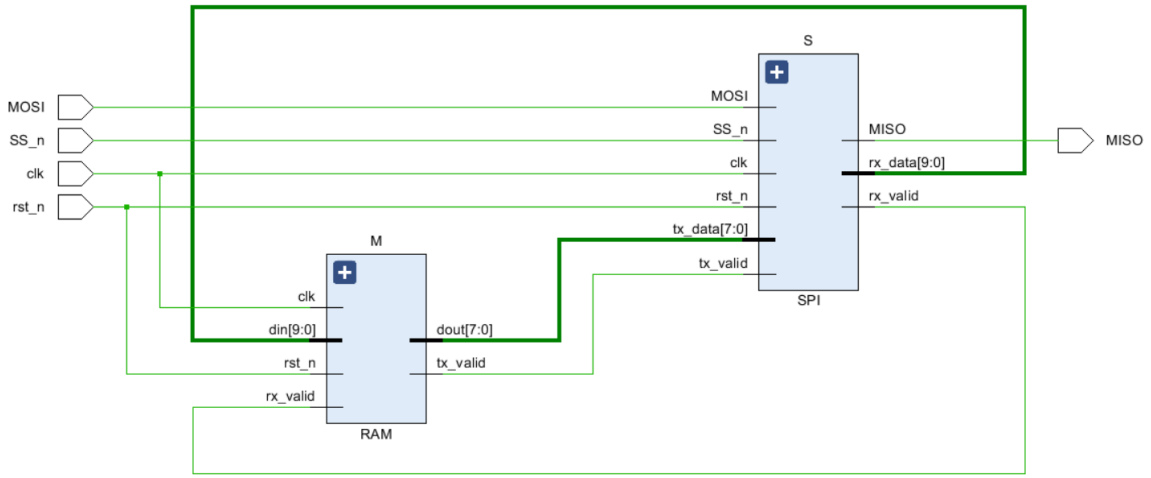


Figure 2: Schematic after the elaboration

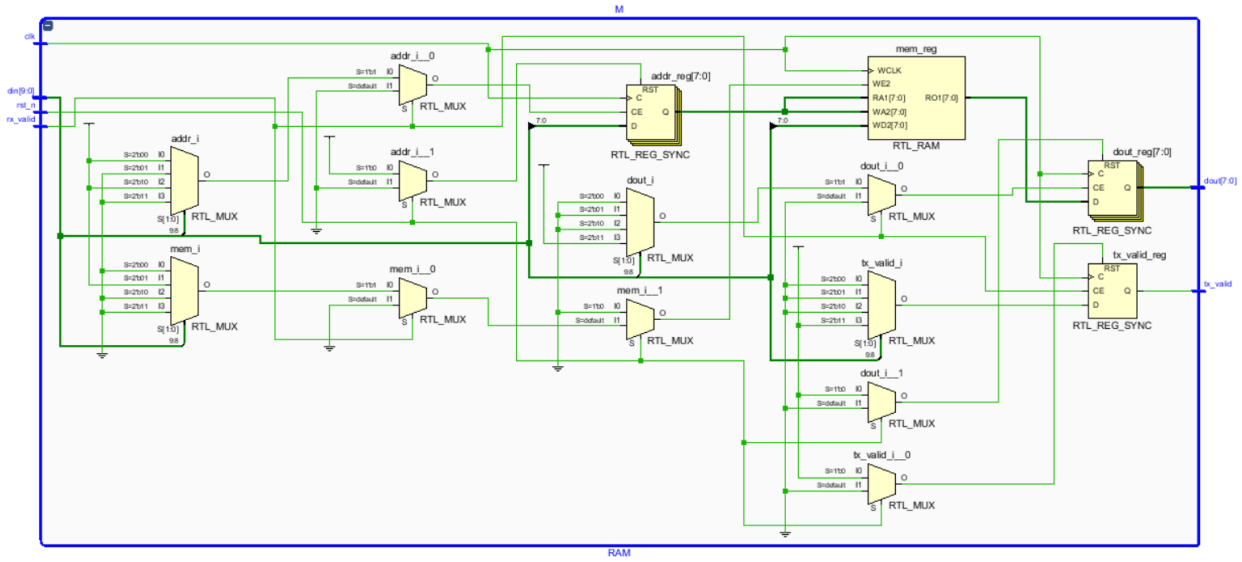


Figure 3: RAM

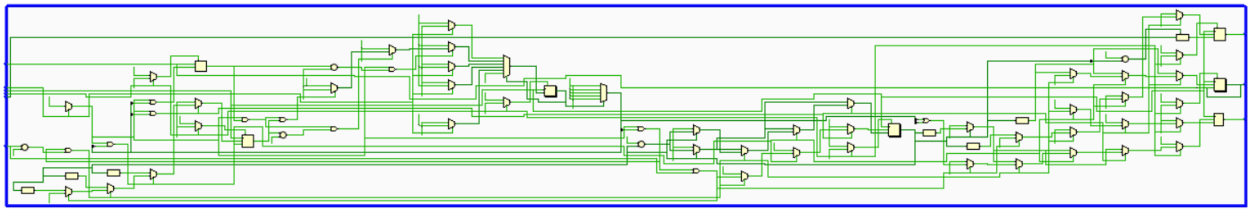


Figure 4: SPI

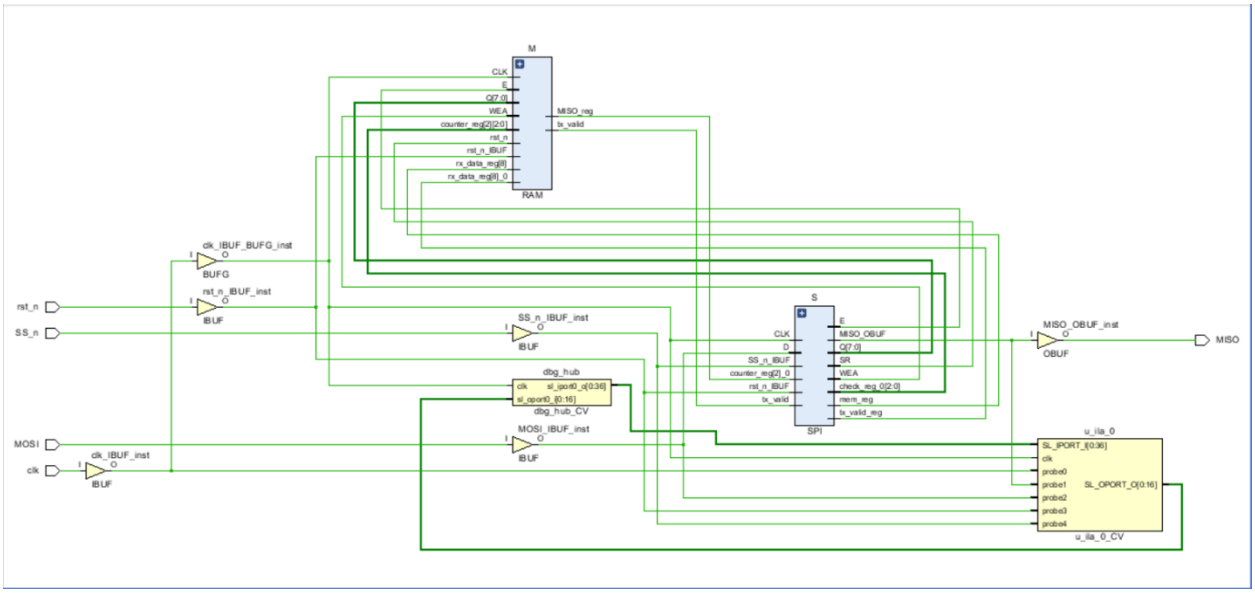


Figure 5: Schematic after synthesis

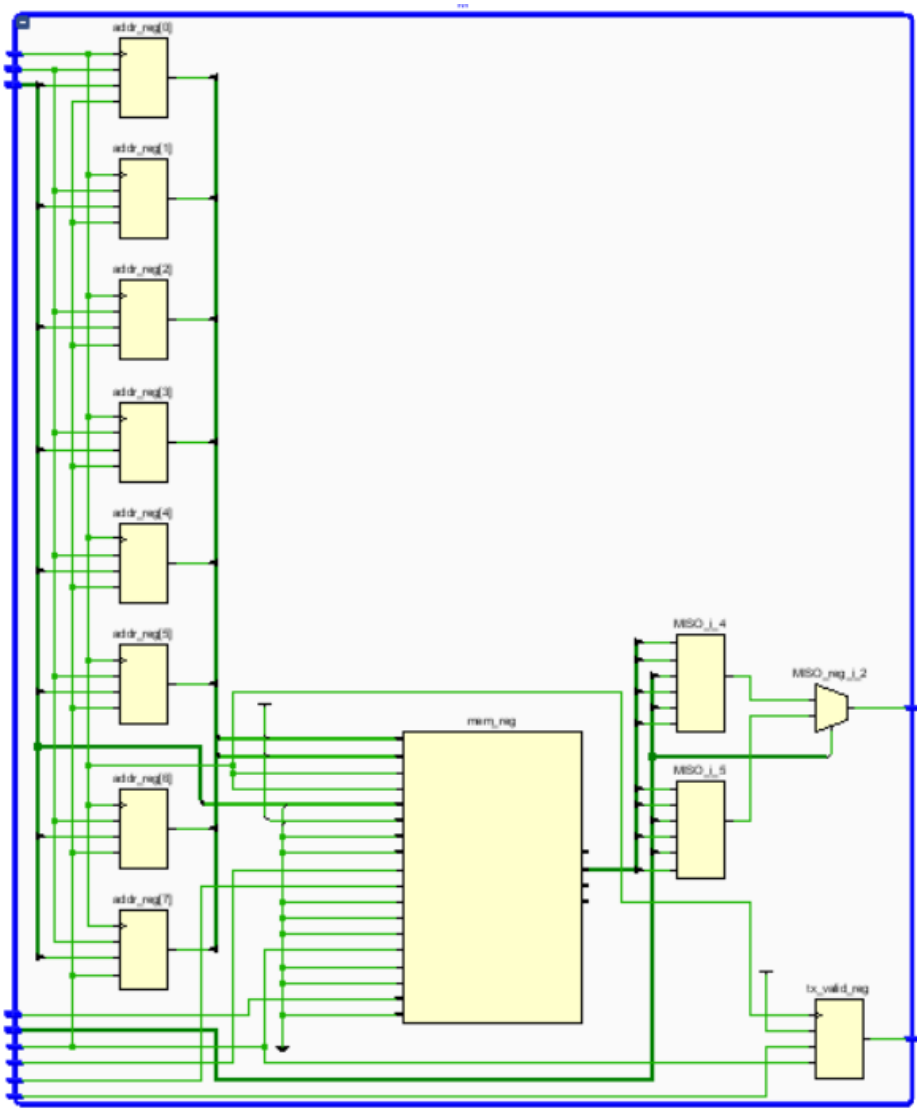


Figure 6: RAM after synthesis

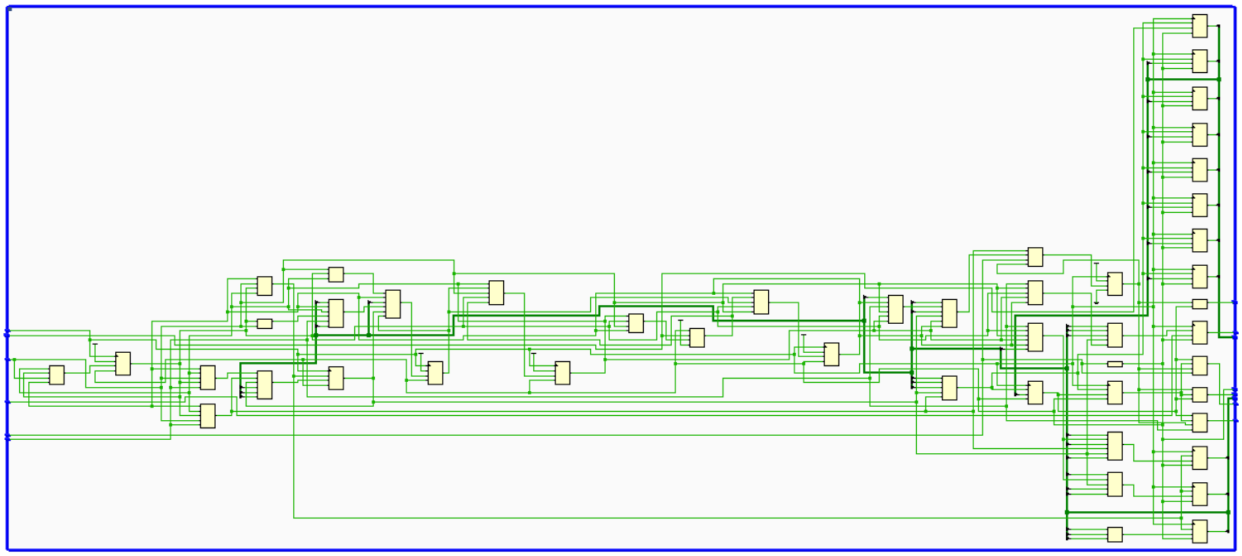


Figure 7: SPI after synthesis

```
98  | ----------------------------------------------------------------------------------------------------
99  |                State |               New Encoding |                Previous Encoding
100 | ----------------------------------------------------------------------------------------------------
101 |                 IDLE |                        000 |                             000
102 |              CHK_CMD |                        100 |                             001
103 |                WRITE |                        011 |                             010
104 |             READ_ADD |                        010 |                             011
105 |            READ_DATA |                        001 |                             100
106 | ----------------------------------------------------------------------------------------------------
107 | INFO: [Synth 8-3354] encoded FSM with state register 'cs_reg' using encoding 'sequential' in module 'SPI'
```

Figure 8: Report showing type of encoding used (sequential)

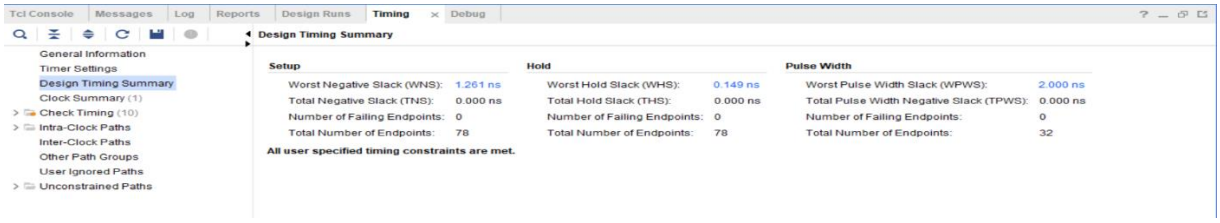


Figure 9: Timing Report

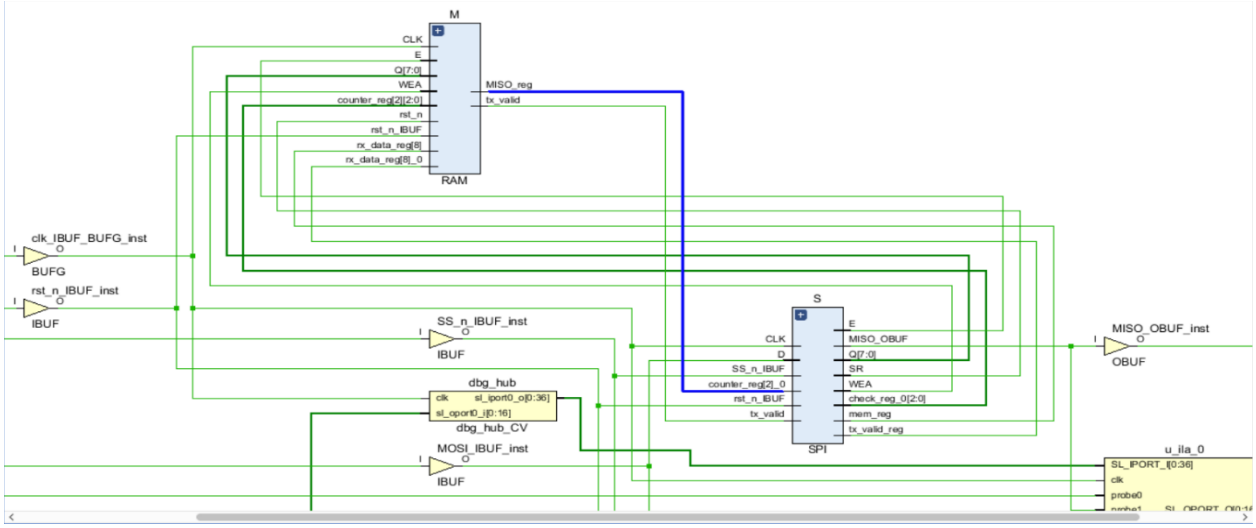


Figure 10: Snippet of the critical path highlighted in the schematic

2- One_hot:

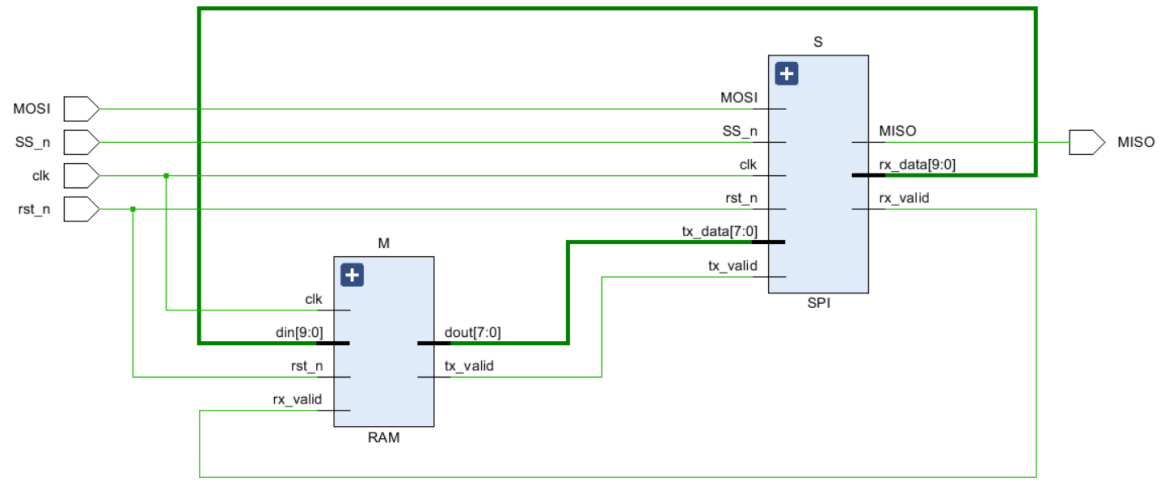


Figure 11:Schematic after the elaboration

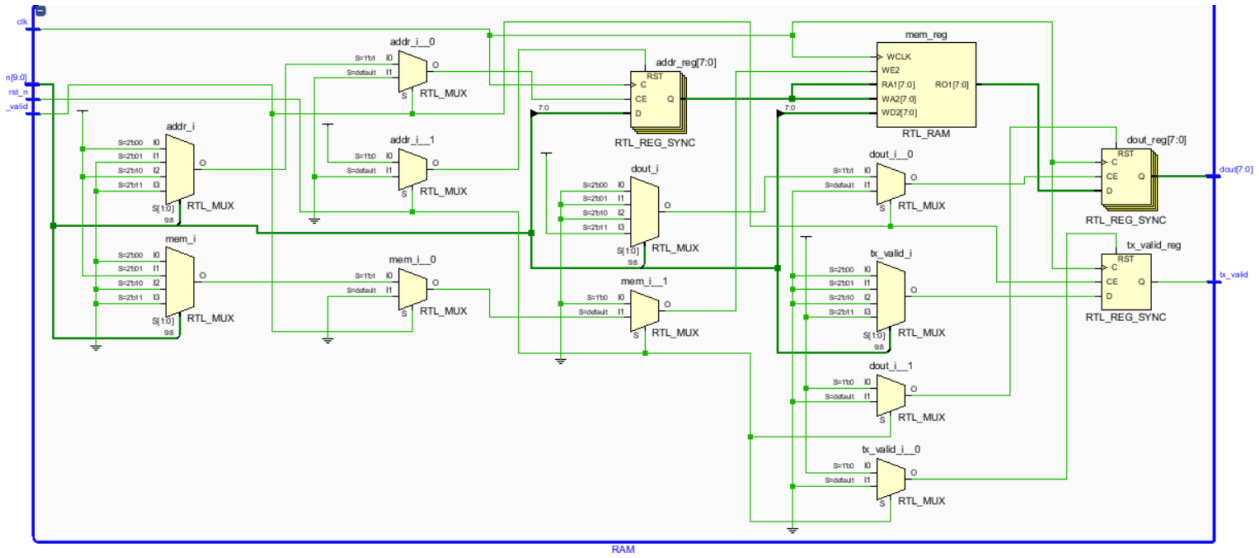


Figure 12: RAM

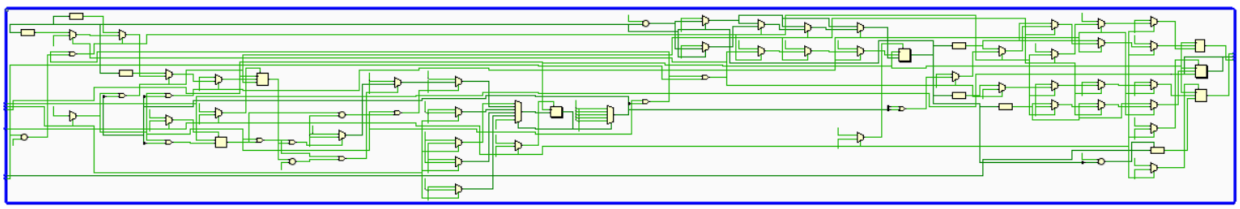


Figure 13: SPI

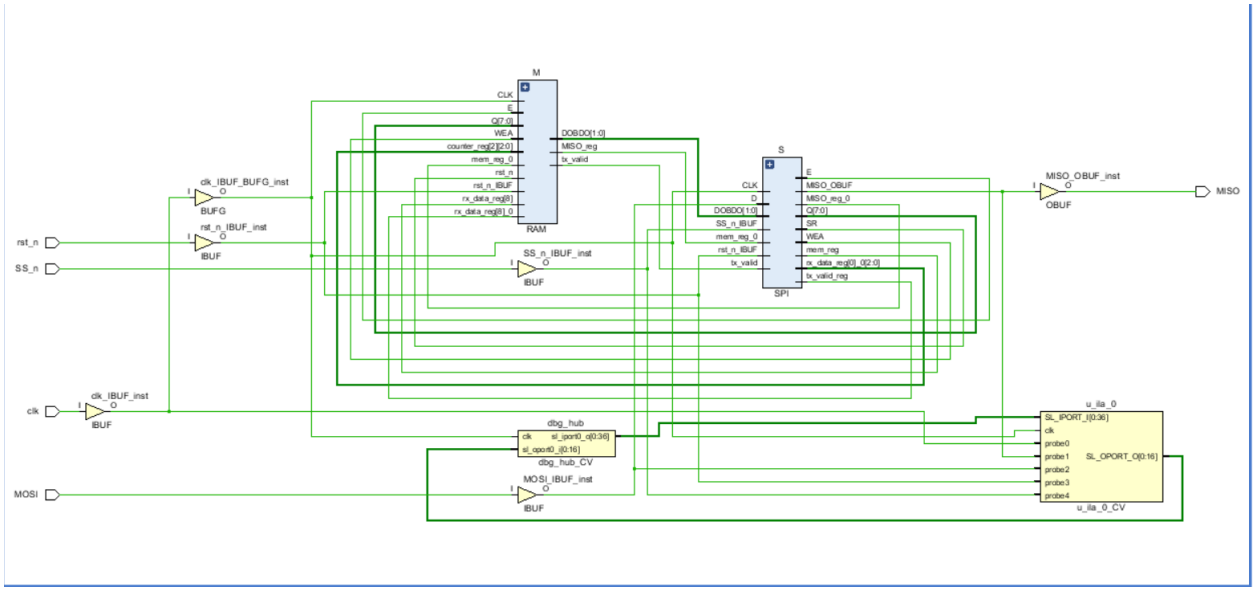


Figure 14:Schematic after synthesis

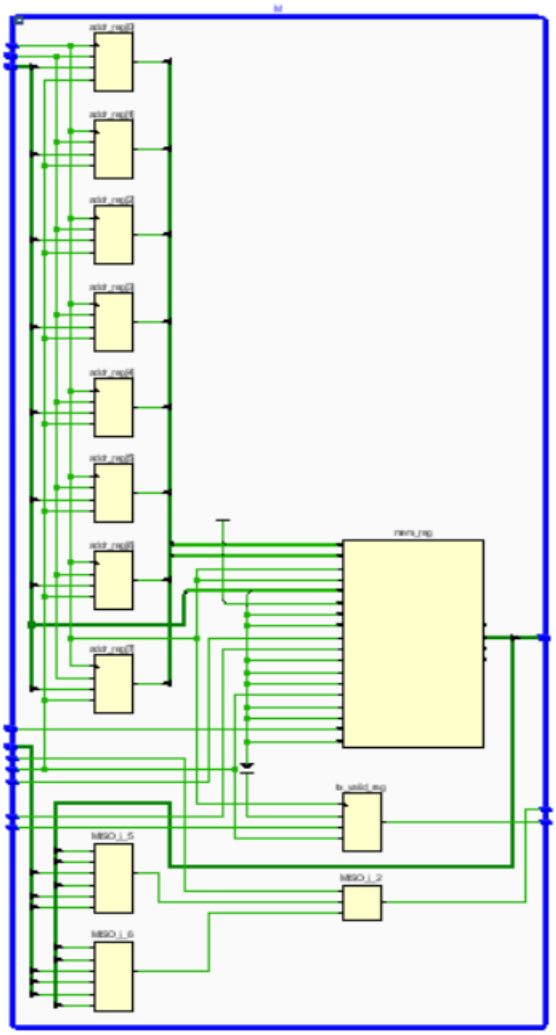


Figure 15: RAM after synthesis

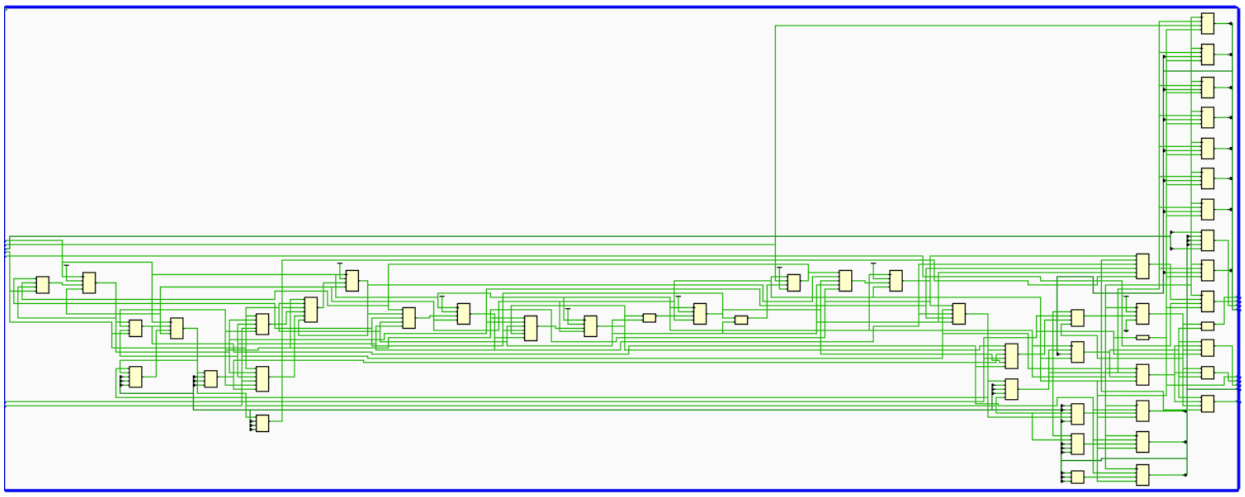


Figure 16: SPI after synthesis

State	New Encoding	Previous Encoding
IDLE	00001	000
CHK_CMD	10000	001
WRITE	01000	010
READ_ADD	00100	011
READ_DATA	00010	100

```
INFO: [Synth 8-3354] encoded FSM with state register 'cs_reg' using encoding 'one-hot' in module 'SPI'
WARNING: [Synth 8-327] inferring latch for variable 'in_MOSI_reg' [E:/digital design projects/Team_project_
```

Figure 17:Report showing type of encoding used (one_hot)

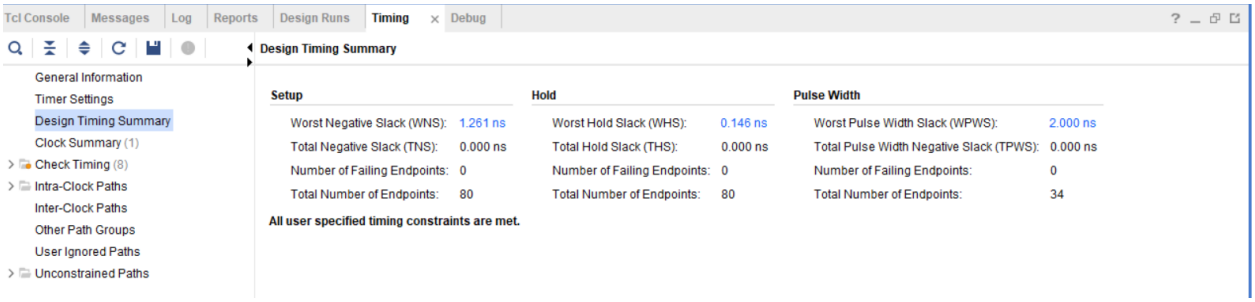


Figure 18: Timing Report

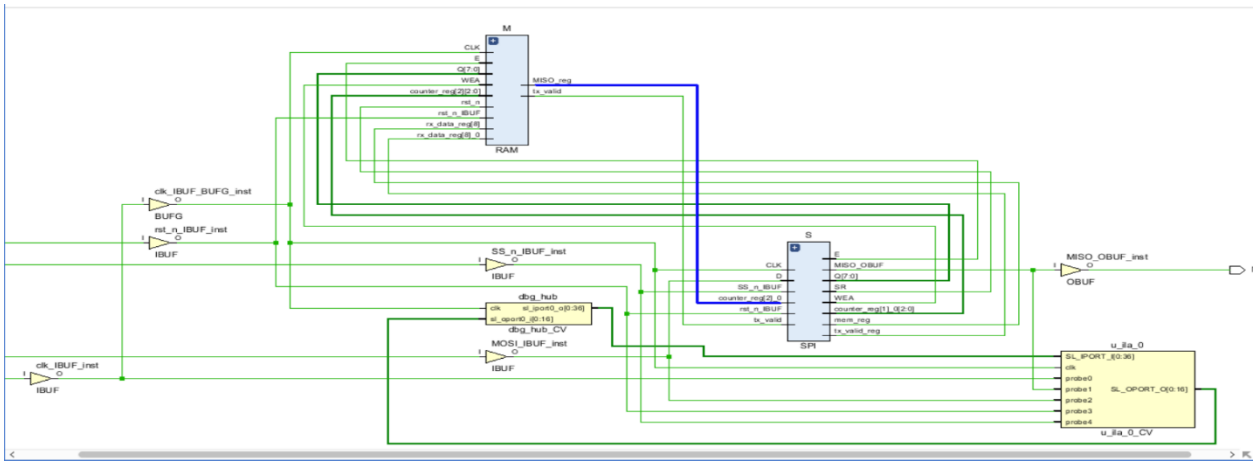


Figure 19: Critical Path

3- gray:

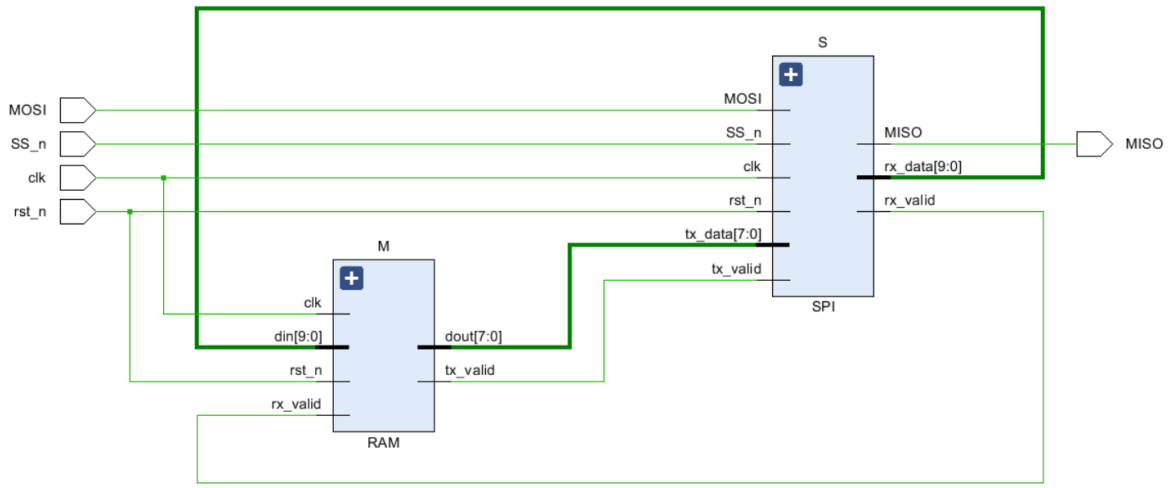


Figure 20: Schematic after the elaboration

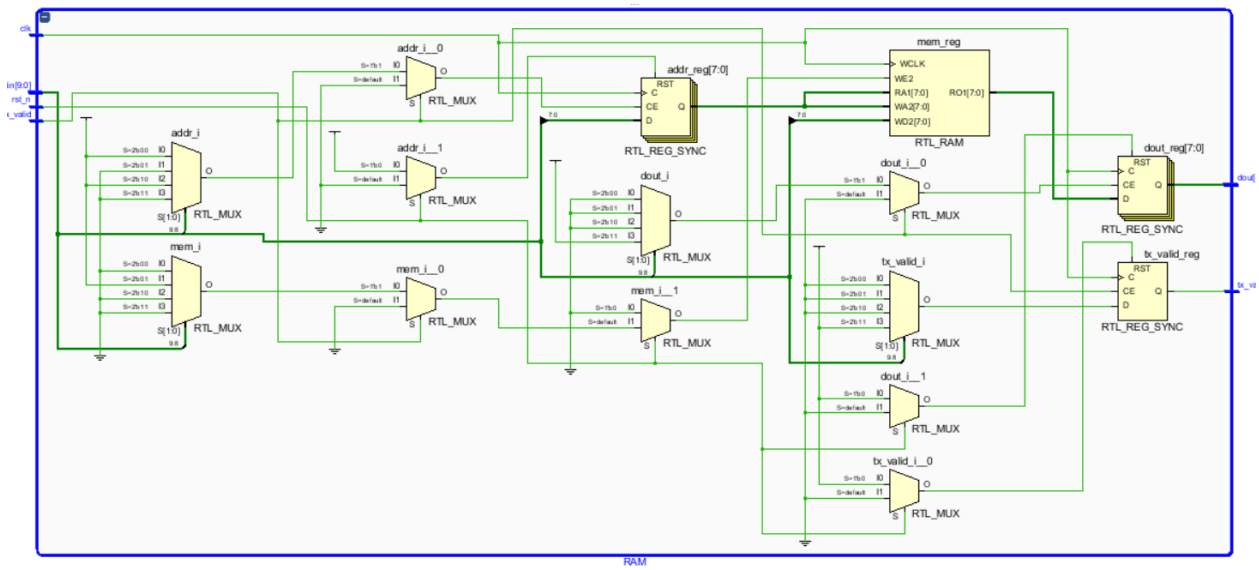


Figure 21: RAM

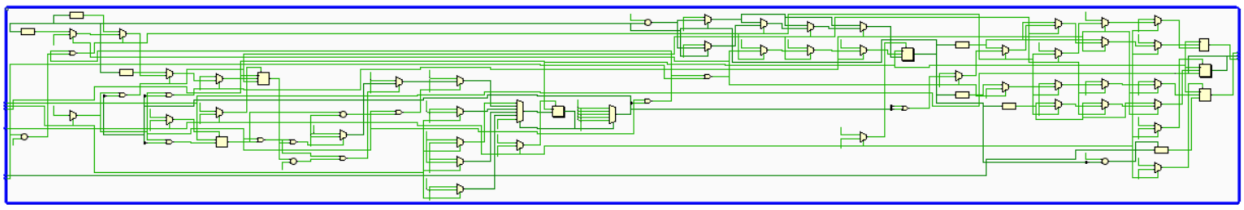


Figure 22: SPI

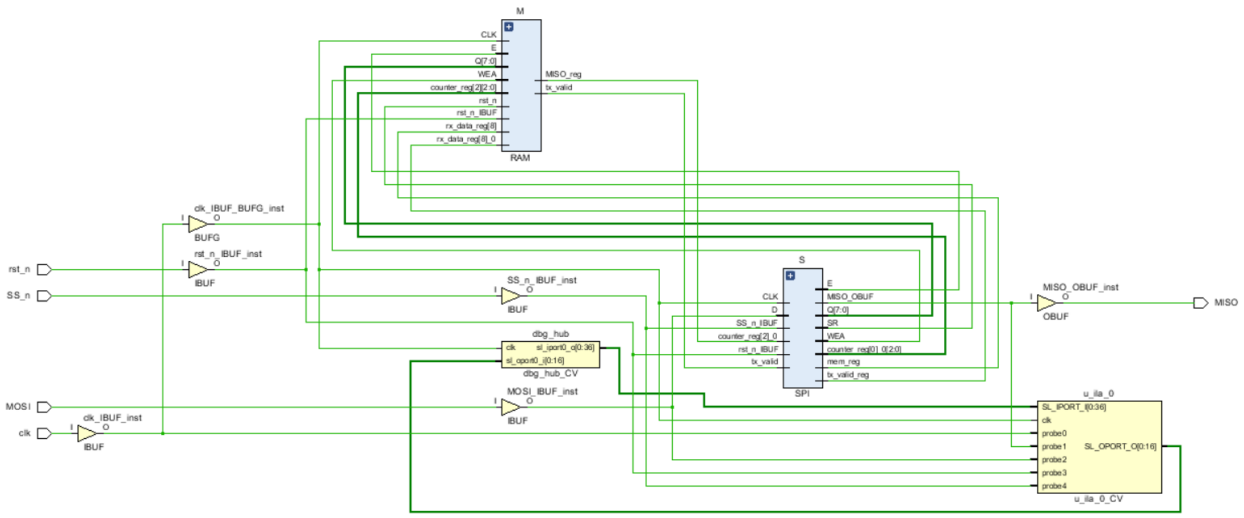


Figure 23: Schematic after synthesis

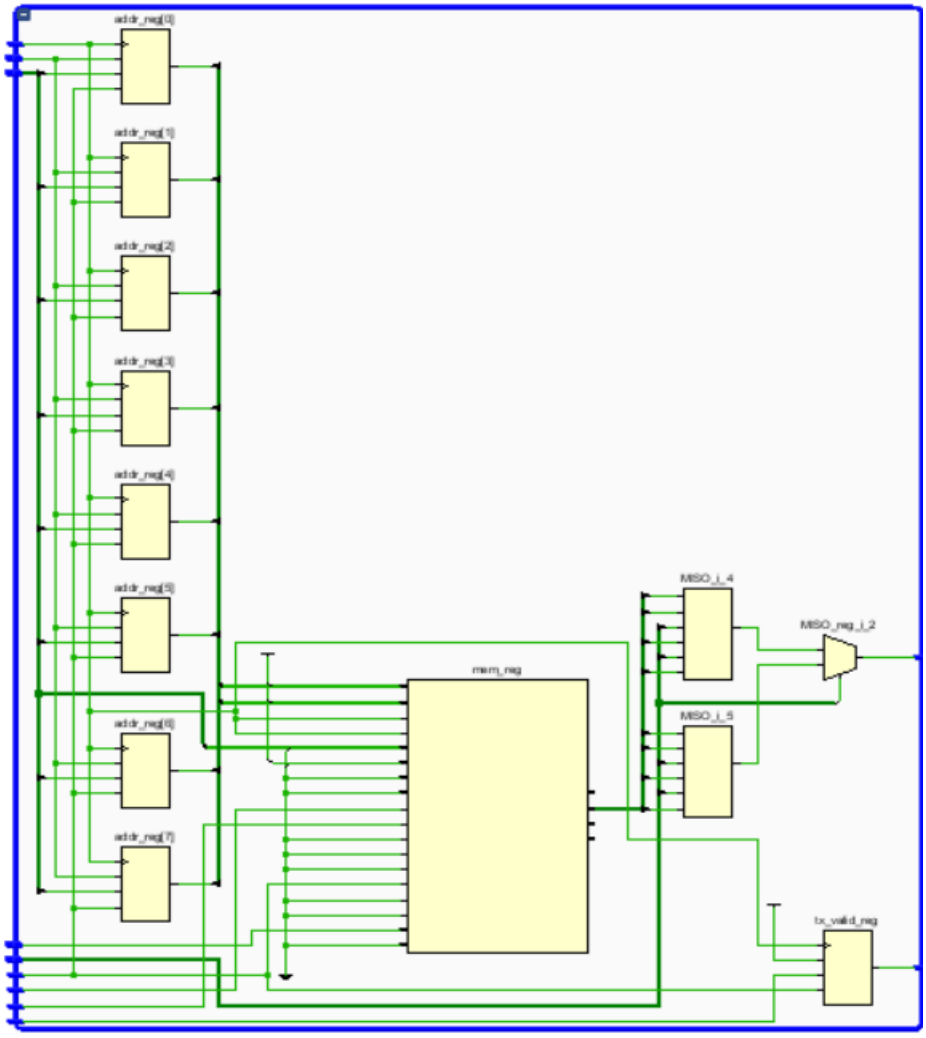


Figure 24: RAM after synthesis

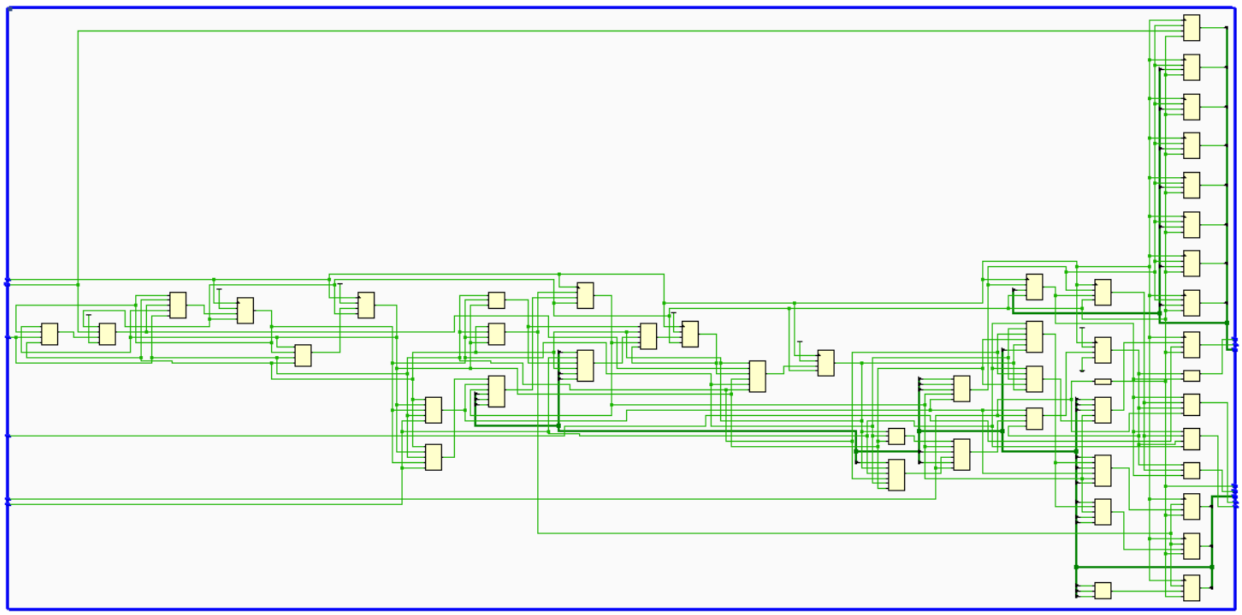


Figure 25: SPI after synthesis

State	New Encoding	Previous Encoding
IDLE	000	000
CHK_CMD	111	001
WRITE	010	010
READ_ADD	011	011
READ_DATA	001	100

```
INFO: [Synth 8-3354] encoded FSM with state register 'cs_reg' using encoding 'gray' in module 'SPI'
```

Figure 26: Report showing type of encoding used (gray)

Note: we wrote the attribute (* fsm_encoding = "gray" *)

Before cs and ns declaration and tried to open new

Project and run again but the tool keeps the gray

Encoding as shown in figure (the first two states are

Wrong then it becomes right)

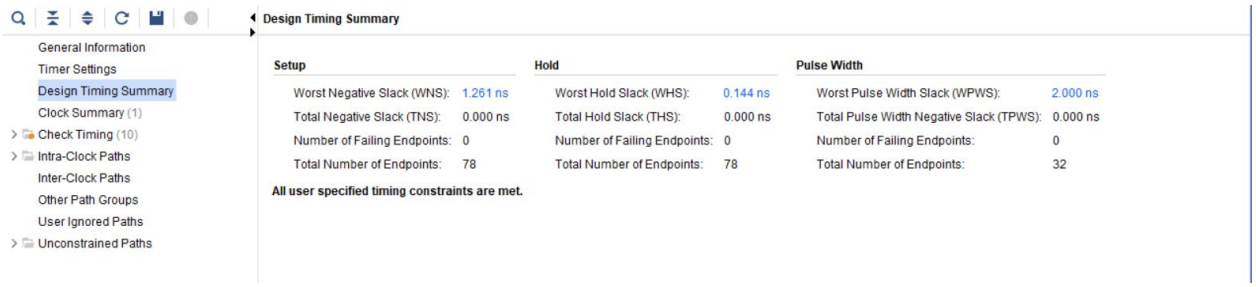


Figure 27: Timing Report

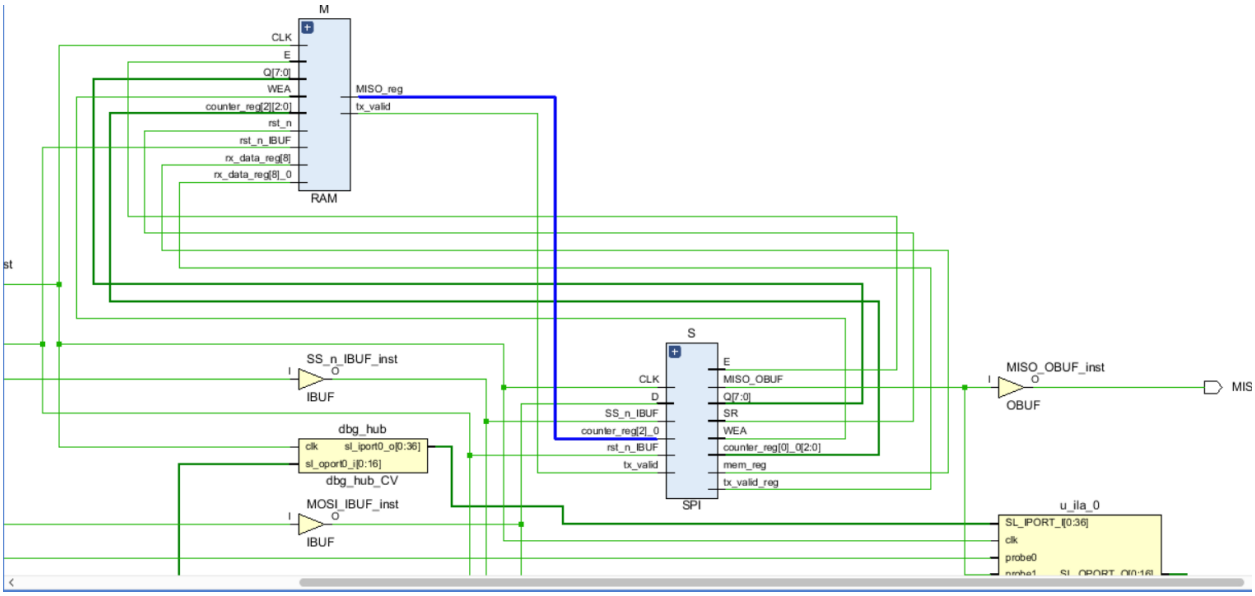


Figure 28: Critical path

4-Implementation snippets for each encoding:

1- Sequential:

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	BSCANE2 (4)
SPI_With_SPR	1274	1946	11	605	1166	108	743	1	5	2	1
dbg_hub (dbg_hub)	481	727	0	230	457	24	309	0	0	1	1
M (RAM)	2	9	1	3	2	0	0	0.5	0	0	0
S (SPI)	25	21	0	10	25	0	9	0	0	0	0
u_ila_0 (u_ila_0)	766	1189	10	370	682	84	422	0.5	0	0	0

Figure 29: Utilization report

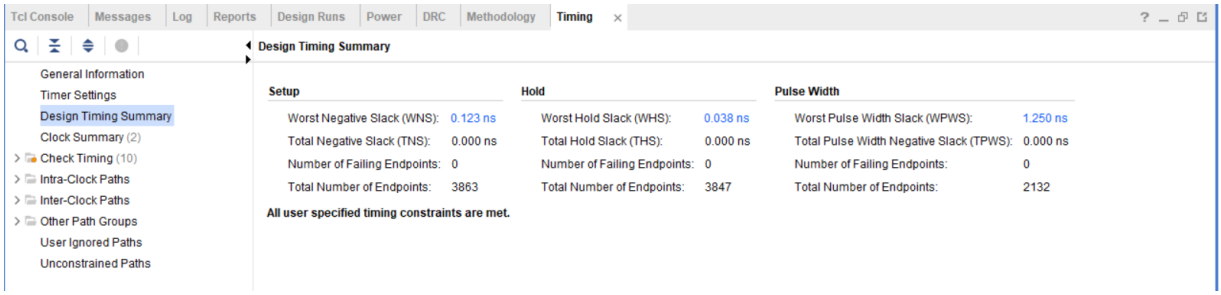


Figure 30: Timing Report

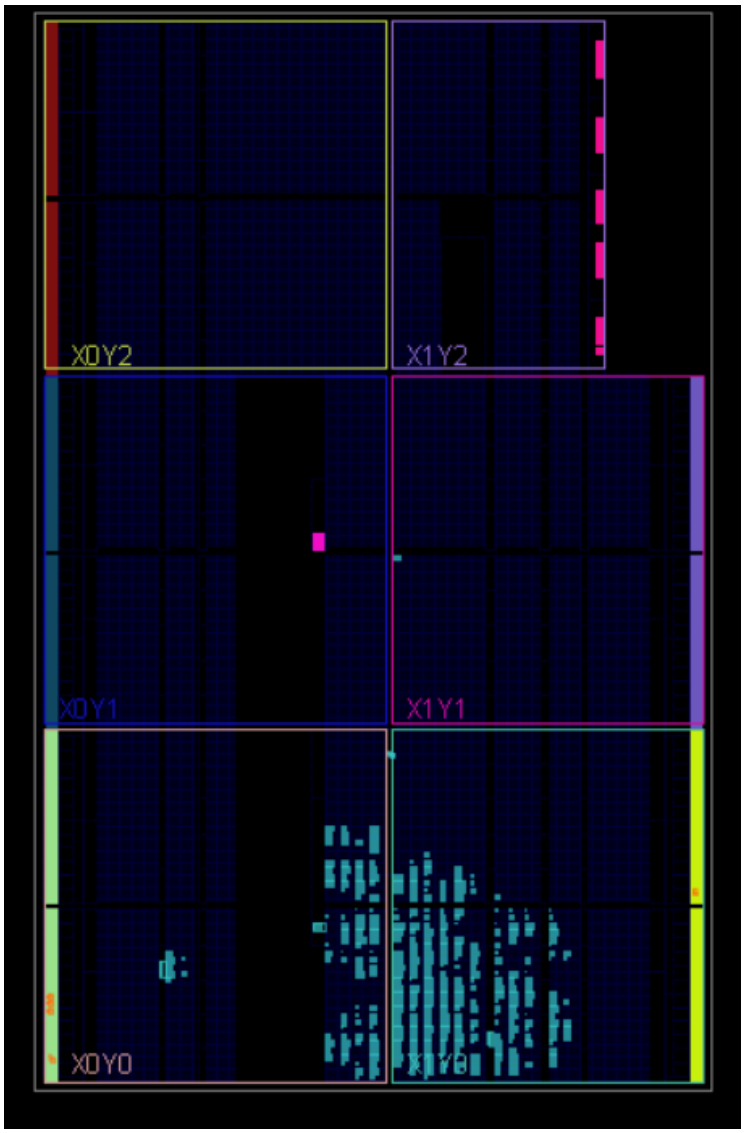


Figure 31: device

2- One_hot:

Tcl Console | Messages | Log | Reports | Design Runs | Power | DRC | Methodology | Timing | Utilization

Hierarchy

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	BSCANE2 (4)
SPI_With_SPR	1270	1948	11	653	1162	108	742	1	5	2	1
dbg_hub (dbg_hub)	480	727	0	245	456	24	313	0	0	1	1
M (RAM)	2	9	1	3	2	0	0	0.5	0	0	0
S (SPI)	24	23	0	13	24	0	10	0	0	0	0
u_ila_0 (u_ila_0)	764	1189	10	402	680	84	417	0.5	0	0	0

Figure 32: Utilization report

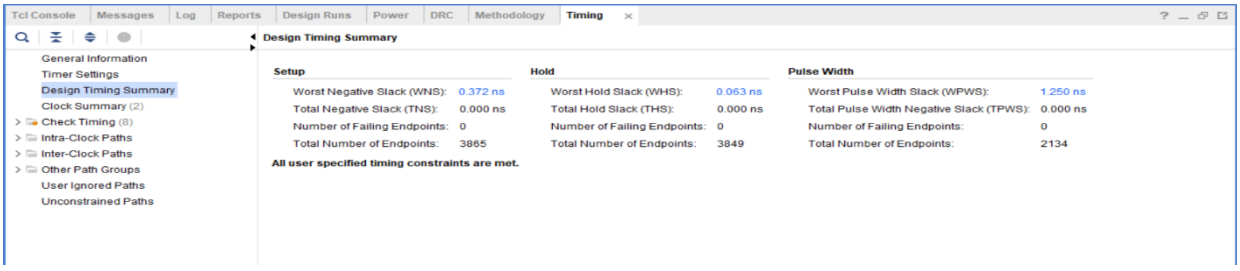


Figure 33: Timing Report

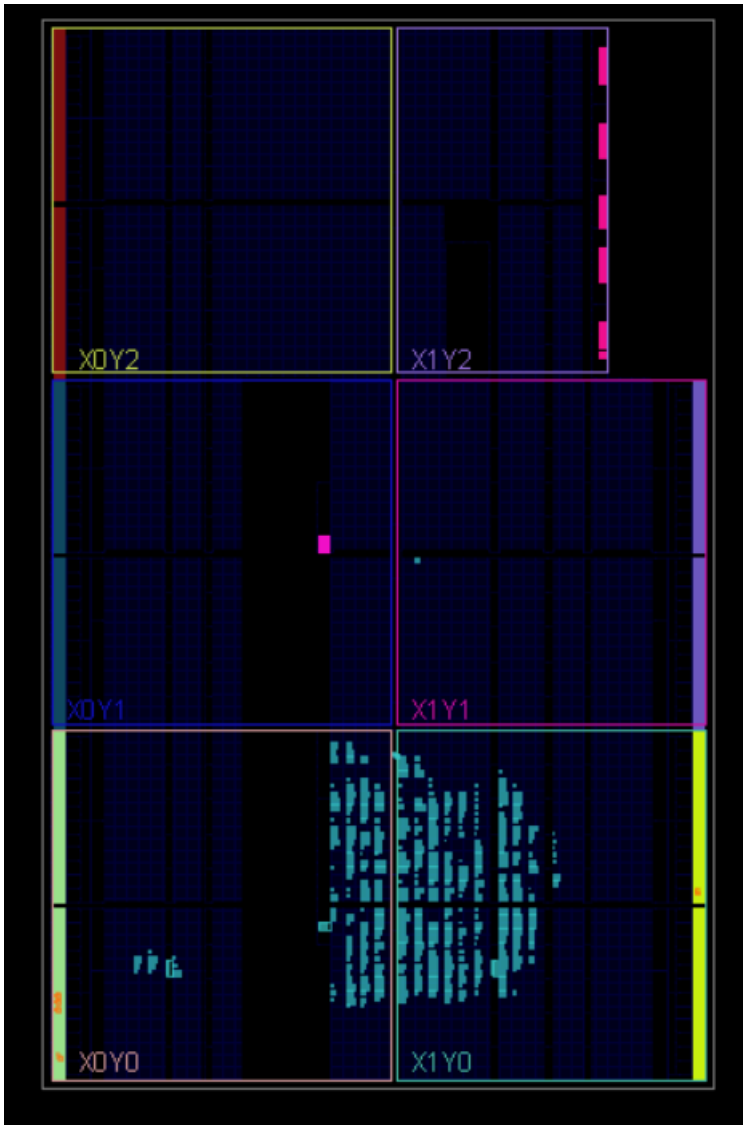


Figure 34: device

3- gray:

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	LUT Flip Flop Pairs (20800)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)	BSCANE2 (4)
SPI_With_SPR	1267	1946	11	611	1159	108	744	1	5	2	1
dbg_hub (dbg_hub)	480	727	0	234	456	24	309	0	0	1	1
M (RAM)	2	9	1	3	2	0	0	0.5	0	0	0
S (SPI)	22	21	0	10	22	0	8	0	0	0	0
u_ila_0 (u_ila_0)	763	1189	10	369	679	84	423	0.5	0	0	0

Figure 35: Utilization report

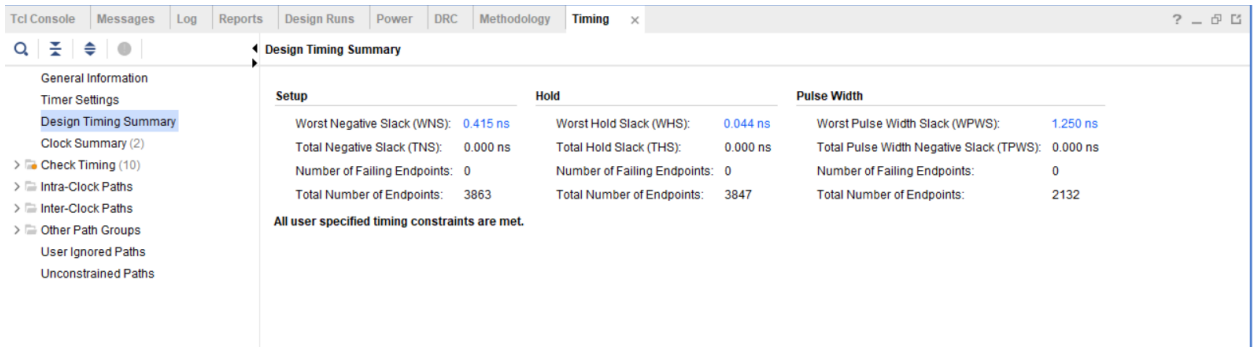


Figure 36: Timing report

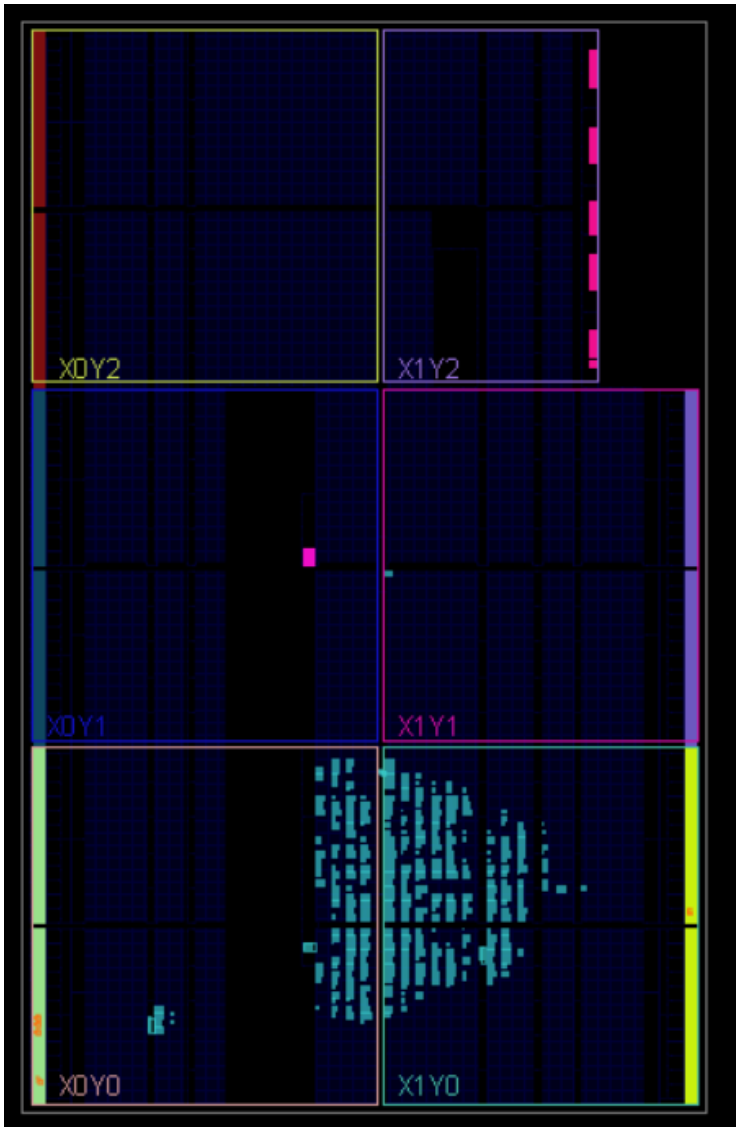


Figure 37: device

- Since we wish to operate at the highest frequency possible so we will choose the encoding based on the encoding type that gives us highest setup time slack
    - **Sequential = 0.123 ns**
    - **One_hot =0.372 ns**
    - **gray = 0.415 ns**
- So, we will choose gray encoding and now we will add a debug core such that all internals can be analyzed and generate bitstream file.

5- Snippet of messages tab:

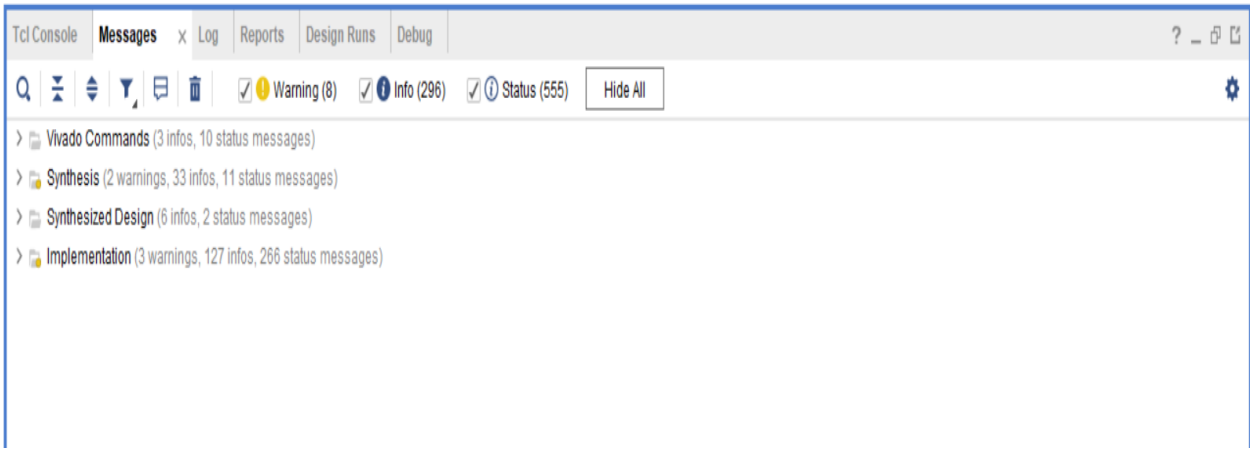


Figure 38: Snippets of messages tab showing no errors after running (elaboration, synthesis and implementation)

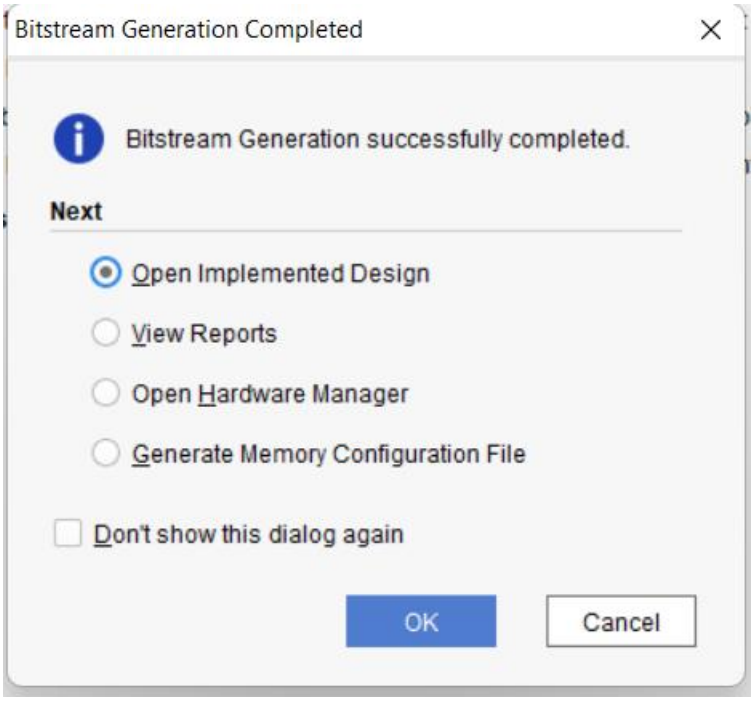


Figure 39: successful bitstream