1. EVM and FSM

An volling machine contain mainly 5 mode of operation:
1. Poll vote to a candidate
2. Check a candidate's vote
3. Check Total votes polled (all candidates)
4. Check Winner (and his vote)
5. Reset machine

This requires an input module a finite state machine (FSM) to process input a memory and output.
My part included both the FSM and Memory module (code segments: e_fsm_logic.vhdl and e_memory.vhdl).

In memory we can store at max 65,536 votes (16bit) and we have a RAM / Memory of 8 so we can store votes of 8 candidates at max.
Our RAM has 8 16bit memory cells / registers each corresponding to one candidate.

We will observe mode of operation are of two kinds:
1. Single memory access : We just need one memory access to poll a vote for example or Check one candidate's vote.
2. Multiple memory access : We need multiple (8 in our case) memory access to Check winner , Reset all votes or Check total polled votes.

States (1,2) Can be completed in 1 clock cycle
States (3,4,5) Need 8 clock cycles in our case

Representing our FSM states as:
1. Poll vote to a candidate (POLL)
2. Check a candidate's vote (CHK)
3. Check Total votes polled (all candidates) (TOT)
4. Check Winner (and his vote) (WIN)
5. Reset machine (RST)

We have 8 action pairs {0,1,2,3,4,5,6,7} as inputs NOT provided either by user BUT generated using logic. This is done to enable a loop from Memory cell 1 to cell 8 (Candidate1 to Candidate8).
And there are 5 more user provided input as action in our FSM {p,c,w,t,r}.

Hence for states (1,2) we can return to Q0 at any action. For the other states they return to Q0 only at action 7 but stays in same state for other Actions to ensure we lopp through all 8 memory cells, reading the memory cell N is counted as action N where N = {0,1,2,3,4,5,6,7} .

Additionally Q0 is our default start state. Our EVM (electronic voting machine) entity is a Mealy machine as represented next:

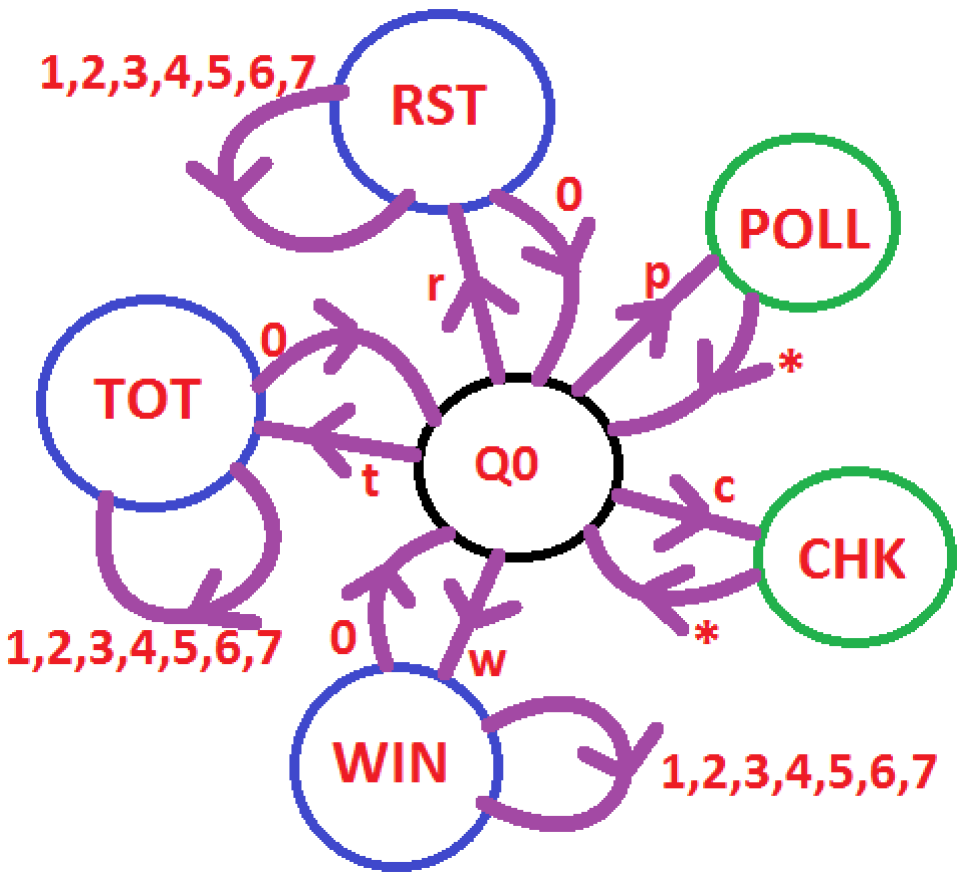

Fig: Finite state machine corresponding to our EVM

The Transition State Action pair can also be representeed in following table:

Initial State	**Action**	**Final State**
Q0	W	WIN
Q0	P	POLL
Q0	T	TOT
Q0	C	CHK
Q0	R	RST
WIN	0	Q0
WIN	1,2,3,4,5,6,7	WIN
POLL	ANY ACTION (*)	Q0
CHK	ANY ACTION (*)	Q0
RST	0	Q0
RST	1,2,3,4,5,6,7	RST
WIN	0	Q0
WIN	1,2,3,4,5,6,7	WIN

2.Modules and Parts

The whole circuit consists of the following parts (also represented in diagram below):
1. FSM Logic
2. Memory Logic / RAM
3. Input
4. Output

Of this currently FSM and Memory design has been contributed submitted by me.
The FSM and Memory module are entity named e_fsm_logic and e_memory. The input output modules are not complete.

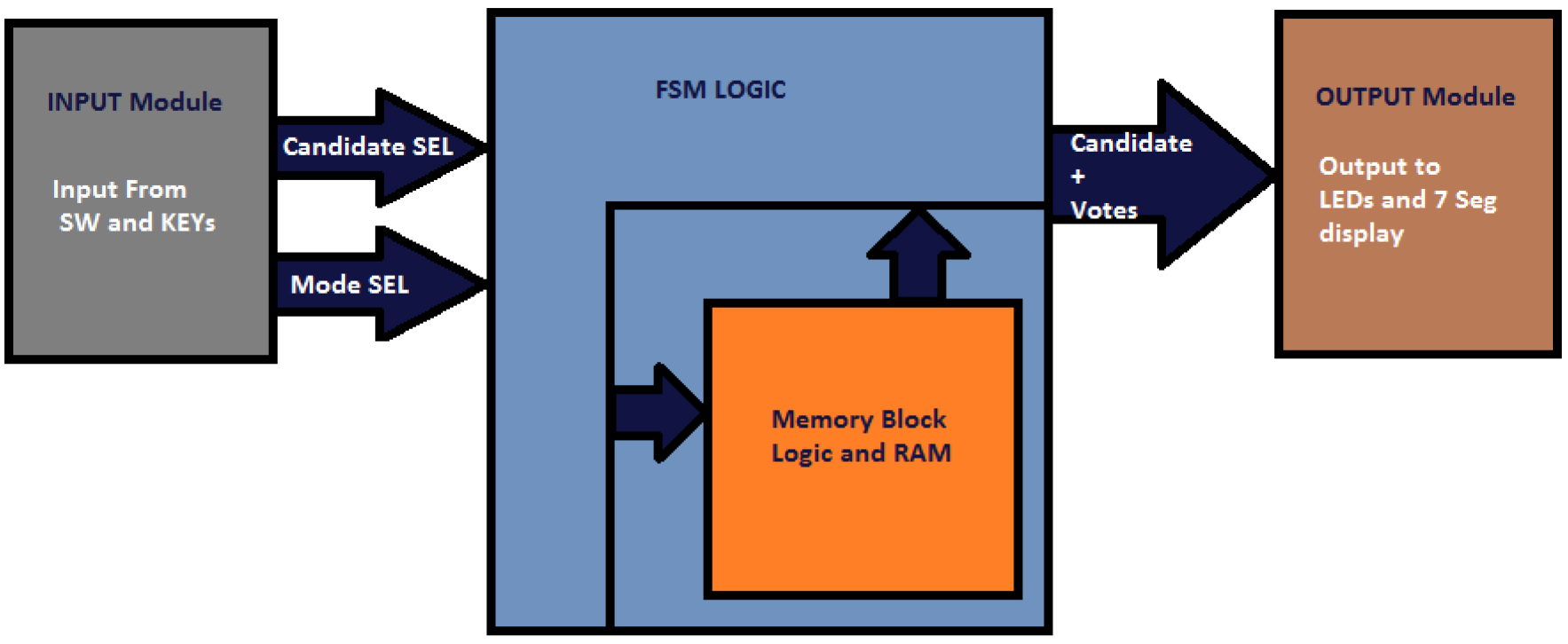

Fig: Modules to used and their Interaction in Full Design

There are 8 candidates in our case hence the design uses following signal logic vector in the interfaces:
1. Candidate Select : 3 bit [SLV_CND_SL]
2. Operation Mode Select: 3 bit [SLV_OPTION]
3. Output Candidate + Votes : 16 bits + 3 bits = 19 bits [SLV_OP]

There is a Output ready bit in SLV_OP for Output device which is useful inmode of operations taking more than 1 clock cycles. For example we will know reset of votes for all registers / vote memory locations are complete only iff this bit becomes one for RST mode of operation

The binary values corresponding to various Inputs and Outputs to our evm module are:

SL_CLK	SLV_OPTION (3bit)		SLV_CND_SL (3bit)		SLV_OP (20 bit)		
	Mode of Operation (3bit)		Candidate No. (3bit)		Output Ready (1 Bit)	Candidate No. (3 bit)	Vote count (16 bit)
1	S_INIT	000	1	000	ready - 1	1 000	1 (as 16 bit binary)
1	S_CHK	001	2	001	not ready - 0	2 001	2 (as 16 bit binary)
1	S_RST	010	3	010		3 010	3 (as 16 bit binary)
1	S_COUNT	011	4	011		4 011	4 (as 16 bit binary)
1	S_POLL	100	5	100		5 100	...
1	S_WIN	101	6	101		6 101	...
			7	110		7 110	...
			8	111		8 111	continued till 65536 = 2^16

3. Modes and Test

In This section we test two modes of operation of our voting machine entity: Check winner candidate and Poll Vote.
We use continous waves in our test bench files to check and test the correctness of our circuit in the modes of operation. This will also enable use to progressively check the changes in output for particular inputs over elapsed unit clock time.
Also the RAM / Memory is pre populated with vote count for 8 candidates to make testing easier.

We pre-populate the Vote data of our candidates in memory block with hexadecimal values as:
(0 => x"0FFF", 1 => x"1AFF", 2 => x"0FF5", 3 => x"0FF7", 4 => x"0FF1", 5 => x"1FAF", {6,7} => x"00F0")
Candidate numbering from 1 but memory block starts from 000 .

We do the following steps:

Steps
we add 2 vote to Candidate7
we check Candidate7's vote
we check the Winner
we reset and check values in memory locations
we count total votes

The waveforms simulation results for above tests are show below, these coresponds to the expected output:

1. Polling two more Votes for Candidate7 [110]

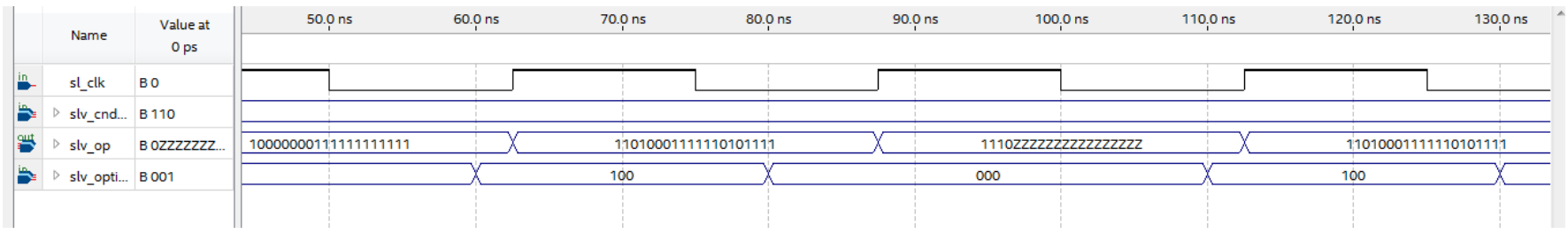

Fig:Simulation waveform of vote poll

Two votes are polled at 62.5ns and 112.5ns we also observe that after vote is polled the output shows:
slv_op: 1110ZZZZZZZZZZZZZZZZ at time 90nS

Output Ready	Candidate Selected	16 bit vote count
1	110	ZZZ..Z

which is expected (we dont show votes in POLL mode of operation) .
Since votes of Candidate7 was initiated in RAM as HEX(00F0) = 240 . After polling twice, the number of votes should be 242 = BIN(11110010).

2. Reading the Votes for Candidate7 in Memory location 110.

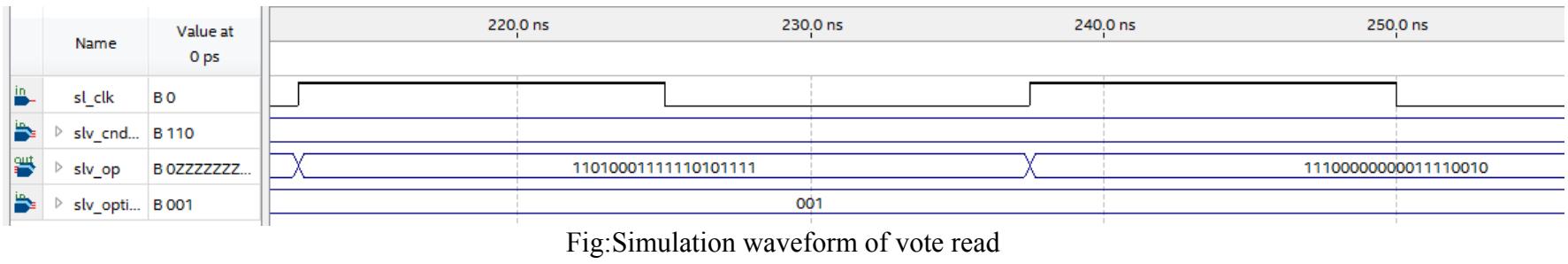

Fig:Simulation waveform of vote read

slv_op: 1110000000011110010 at time 240nS .

Output Ready	Candidate Selected	16 bit vote count
1	110	0000 0000 1111 0010

This is as per expected output, the corresponding decimal value is 242 .

3. Checking Winner and Number of Votes

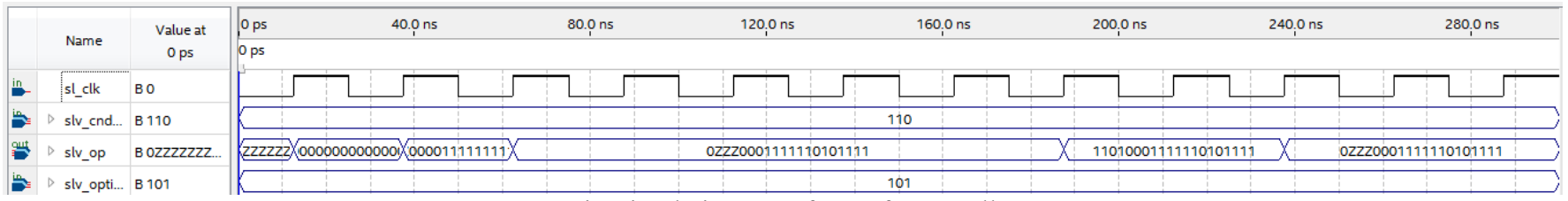

Fig:Simulation waveform of vote poll

slv_op: 11010001111110101111 at time 200nS (roughly 8 clock pulses after slv_option was 101).

Whwn Output is Ready	Candidate Selected	16 bit vote count
1	101	0001 1111 1010 1111

Thus we see after8 clock pulses our finite machine sends the signal Output Ready (=1) and shows Candidate6 whose Vote is stored in Memory location 5 has the highest number of votes. This is expected as we have already populated value (or votes stored) of Memory location 5 as HEX(1FAF) = BIN(0001 1111 1010 1111) = 8111 .

4. Count total Votes

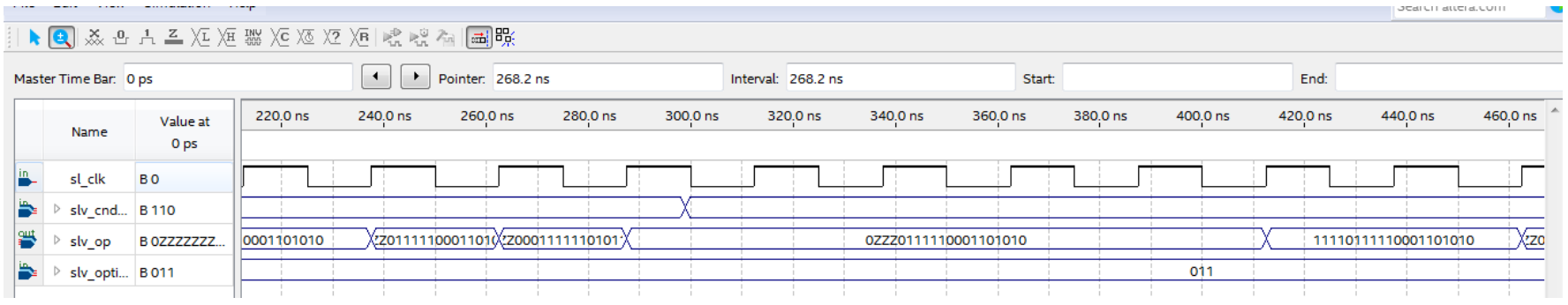

Fig:Simulation waveform to count total Votes

slv_op: 11110111110001101010 at time 420nS (roughly 8 clock pulses after slv_option was 011).

When Output is Ready	Candidate Selected	16 bit vote count
1	111	0111 1100 0110 1010

After 8 clock pulses once output is ready we can see the Candidate number 8 is selected indicating that the last memory location ’111’ has already been read. The Total vote count = HEX(0FFF + 1AFF + 0FF5 + 0FF7 + 0FF1 + 1FAF + 00F0 + 00F0) = 31850 = BIN(0111 1100 0110 1010)

4. Timing constraint

After compiling we found that the circuit had a negetive slack of -20.098nS for a typical 1nS clock pulse. Thus our circuit should have a time constraint of 21.98 nS for proper working and the slack must atleast be positive. We rounded off the time period from 21nS to 25nS and frequency 45 MHz to 40 MHz and compiled the circuit design. The slack was subsequently positive after this.

Slow 1130mV 85C Model: 43.49 MHz, 32.68 MHz, sl_clk

Slow 1130mV 0C Model: 43.82 MHz ,28.86 MHz, sl_clk

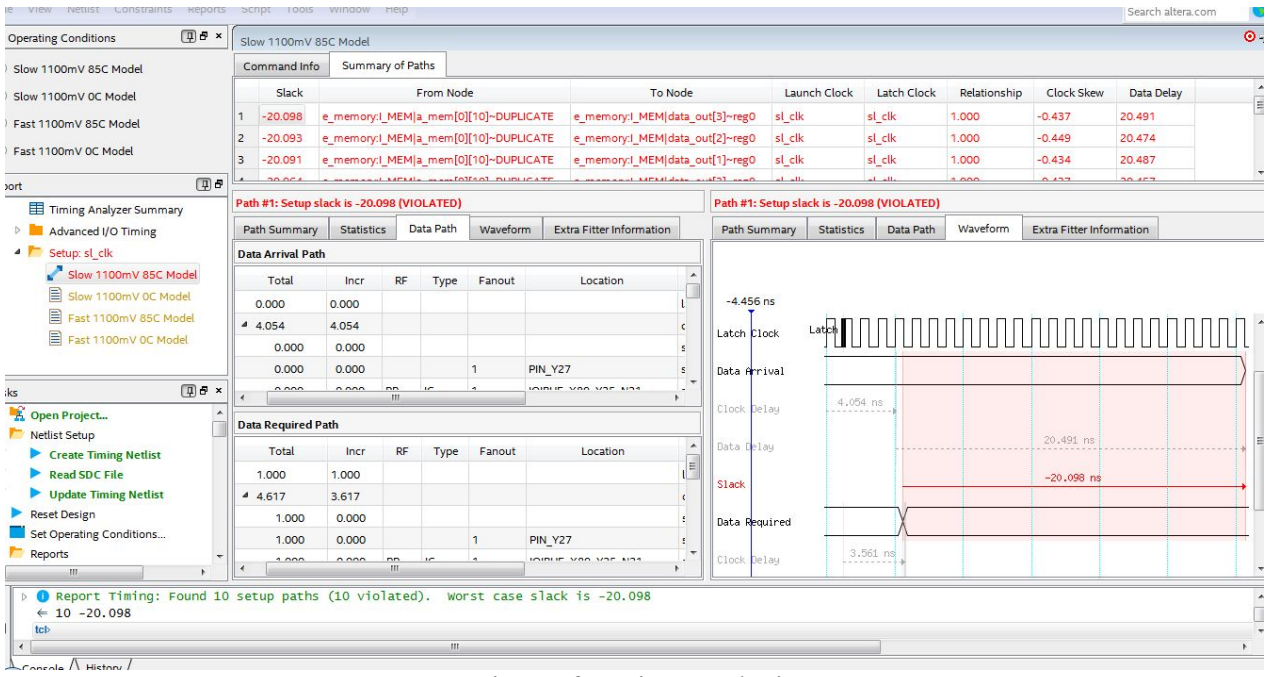

Fig: Before time analysis

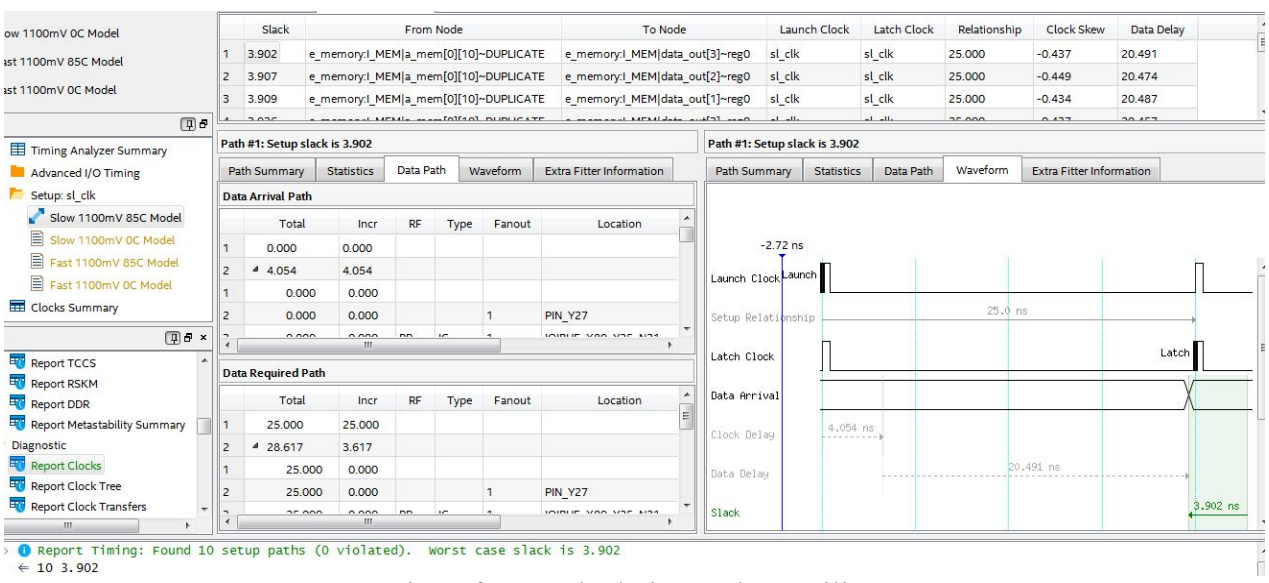

Fig: After recalculating and compiling

We also use the same time period of 25nS for simulations. And since we have to read all memory registers for some states their output appears after 25x8 nS (Signalled by Output Ready bit).