EL6443 VLSI-II Project: 256x4b MBIST

Vidyut Singh (vps4038), Abrar Fahim (af4175)

***Abstract*—MBIST, or Memory Built-in Self-Test, is an industry-standard for testing embedded memories of a system. The embedded memory of a system is tested using various sets of algorithms to detect different types of faults that may reside inside of memory, such as Stuck At Fault (SAF), Transitional Delay Fault (TDF), Coupling Fault (CF), Neighbourhood Pattern Sensitive Faults (NPSF). Since the method is self-testing, a single start signal is provided and the test is carried out on its own using stored patterns.**

I. Project Overview

This project aims to design an MBIST for testing a 256 x 4 bit SRAM cell using the checkerboard and blanket of 0 and 1 pattern where both the BIST and the SRAM cell operate on the same clock.

The project's Behavioral description (RTL Design) is written in Verilog, and the RTL design functionality is tested using the HDL compiler and visualization tools – VCS and Verdi. Synthesis and optimization of the design is done using Genus.

II. Initial Design Principle and Operation

A. Design Components

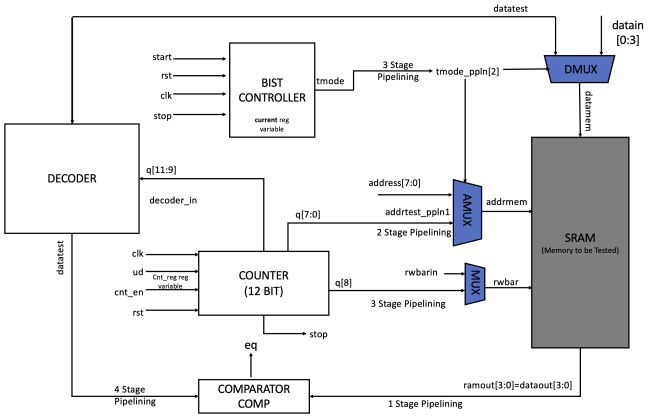


Fig. 1: Schematic of MBIST Dataflow.

The MBIST {Fig. 1} uses the following basic elements in its design to produce a blanket of 0s and 1s, checkerboard and reverse checkerboard patterns:

Memory – 256 word x 4 bit SRAM that is going to be tested by the MBIST.

Counter – 12-bit counter to generate addresses and decoder inputs. The counter can be considered the heart of the MBIST. It generates the address for the memory and generates the decoder inputs. Based on the input, the decoder generates the patterns, which will be either stored (write function) in the memory module or sent to the comparator to compare data retrieved (read function) from the memory module. When the counter generates a stop bit, the counter value again resets to zero.

Decoder – 4-bit wide decoder to generate 3 types of patterns – '0000', '0101' and '0011' – and their inverse. All decoder patterns are shown in Fig. 2

din[2:0], din[3]	out_ppln	d_out
00 00	0000	1111
00 01		0000
00 10		0000
00 11		1111
01 00	0101	1010
01 01		0101
01 10		0101
01 11		1010
10 00	0011	1100
10 01		0011
10 10		0011
10 11		1100
11 xx	xxxx	xxxx

Fig. 2: Patterns generated by decoder.

BIST Controller – Two-state FSM that alternates between functional and test mode states. The BIST controller can be considered the brain of the MBIST. The BIST controller switches to test mode on receiving the start signal and waits for the stop bit to be generated to send the reset signal to the counter.

Multiplexer – These choose between the functional memory address and data or the MBIST generated values based on the test mode signal.

Comparator – 2-input comparator to check that decoder and memory outputs are equal.

B. Design Parameters

The size of the SRAM is 256 x 4 bit. This implies that there are 256 words, and each word has a data length of 4 bits. Therefore, the address required to access each word in the memory is $log_2(256)$ = 8 bits in length.

A single bit is used to indicate the Read or Write operation to the system.

The pattern generation decoder uses a 4 bit input. Among the 4 bits, 2 bits are directly used for generating the pattern. Two additional bits are used for generating the checkerboard pattern and the alternation of the checkerboard patterns. Here, the 0th bit of the counter gets reused to allow for alternating patterns to be applied at each memory address.

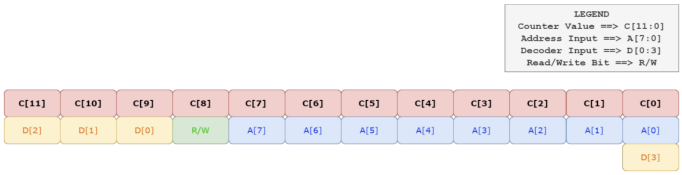


Fig. 3: Translation of counter values to decoder input, address input, and read/write bit.

Therefore, the total length of the counter = 3 bits + 1 bits + 8 bits = 12 bits. Fig. 3 shows the mapping of counter bits.

C. Parametrization

The design is parameterized {Fig. 4} – allowing for varying sizes of SRAM modules to be tested. The design supports a minimum word length of 4 bits as that is the smallest output length the decoder can generate. The two parameters used are **wcount** for the word count and **wlength** for the word length.

```
//TOP LEVEL MODULE DESIGN:

//Module Definition:
module TOP #(parameter wcount=256, wlength=4)(
  input wire start, rst, clk, rwbarin,
  input wire [wlength-1:0] datain,
  input wire [$clog2(wcount)-1:0] address,
  output wire [wlength-1:0] dataout,
  output reg fail
);
```

Fig. 4: Parametrizing the MBIST design.

III. Design Improvements

The initial design provided was extensively pipelined and modified to support the required clock period of **0.6 ns**. Over the courses of the following 4 stages we were not only able to achieve the required clock speed, but also ensure our design operates at 2x the speed as well.

A. Stage I - RTL Improvements

Registering module outputs – The intial design was updated so that combinational elements such as the decoder, multiplexers and comparator have registered outputs to break the a single into smaller paths with lower propagation delay – shown in Fig. 5.

Pipelining address decode – The 256 word SRAM was split into smaller 4x64 word SRAM cells and one additional stage of pipelining was added internally for address decode and memory read. Hence, a memory read/write takes a total of 2 clock cycles, shown in Fig. 6.

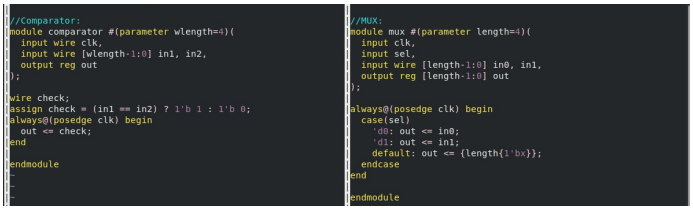


Fig. 5: Registering module outputs.

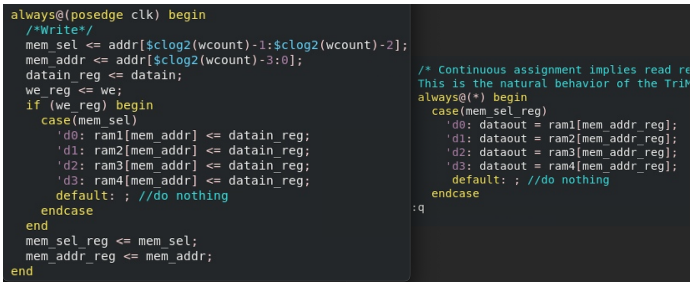


Fig. 6: Pipelining address decode.

Pipelining pattern decode – Decoder logic is also split into two combinational paths with a pipeline, where the first stage decides the pattern to send and the second stage decides whether the pattern needs to alternate. The decoder takes a total of 2 clock cycles to generate the required pattern, as shown in Fig. 7.

```
//CB/RCB DECODER:
module decoder #(parameter wlength=4)(
  input wire clk,
  input wire [3:0] in,
  output reg [wlength-1:0] out
);

reg [wlength-1:0] out_temp_reg;
reg negate;

always@(posedge clk) begin
  case(in[2:1])
    'd0: out_temp_reg <= {wlength/4{4'b0000}};
    'd1: out_temp_reg <= {wlength/4{4'b0101}};
    'd2: out_temp_reg <= {wlength/4{4'b0011}};
    default: out_temp_reg <= {wlength{1'bx}};
  endcase
  negate <= in[3]~^in[0];
  out <= negate ? ~out_temp_reg : out_temp_reg;
end

endmodule
```

Fig. 7: Pipelining pattern decode.

Additional pipelining – Additional pipelining was done to ensure that signals arrive at their destination at the desired clock cycle. Fig. 8 shows the cycle-level information of each pipeline stage in the design.

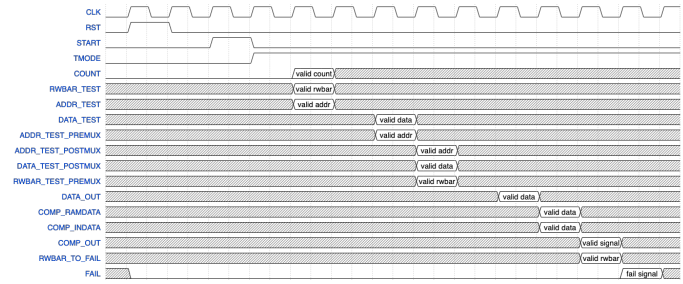
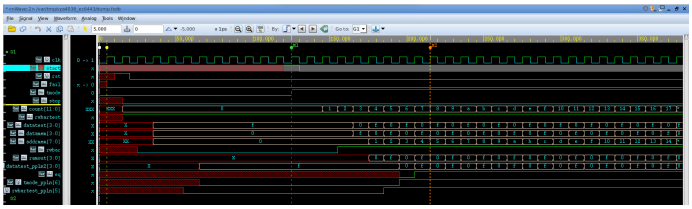
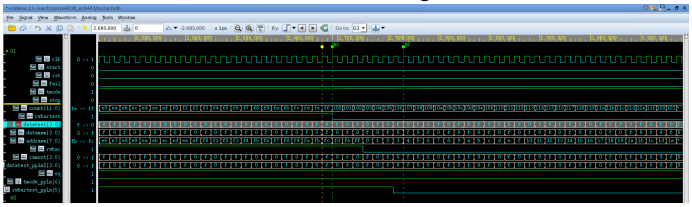


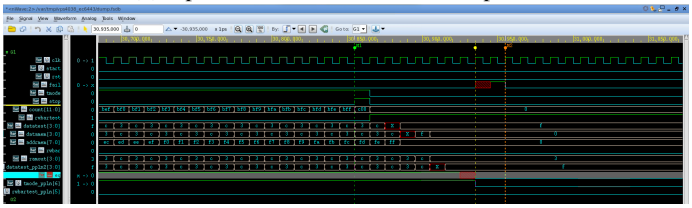
Fig. 8: Final Timing Waveform.



(a) Initialization – sending start bit.



(b) Operation – write and read comparison.



(c) Stop – exiting test mode.

Fig. 9: Simulation Waveforms.

B. Stage II - Simulation

This design was simulated using VCS/Verdi to verify functionality. Fig. 9 show three waveforms from the initialization, operation and stop stages of the MBIST.

C. Stage III - Fixing Genus Issues

Genus threw two errors during synthesis – "TUI-61: A required object parameter could not be found" and "Unable to map design without a tristate buffer or inverter. [MAP-1] [map_library_sanity_check]".

Fixing the TUI-61 error required providing the parameter values during elaboration in Genus using the following command – `elaborate -parameters {256 4}`

The MAP-1 error was solved by using 'x' (unknown or indeterminate logic value) instead of 'z' (high impedance or tri-state signal) in the default case statements inside the decoder, multiplexer and comparator module.

D. Stage IV - Fixing Timing Violations

After running synthesis, the Genus output log showed a timing violation of 181ps {Fig. 10} from the memory address register to the comparator.

```
Path 1: VIOLATED (-181 ps) Setup Check with Pin COMP_out_reg/CLK->D
        Group: clk
    Startpoint: (R) RAM_MEM_mem_addr_reg_reg[0]/CLK
         Clock: (R) clk
      Endpoint: (F) COMP_out_reg/D
         Clock: (R) clk

                  Capture       Launch
     Clock Edge:+       1            0
     Src Latency:+      0            0
     Net Latency:+      0 (I)        0 (I)
         Arrival:=      1            0

           Setup:-      4
   Required Time:=     -3
    Launch Clock:-      0
       Data Path:-    178
           Slack:=   -181

#-------------------------------------------------------------------------------------------------------------
#          Timing Point           Flags    Arc    Edge          Cell           Fanout Load Trans Delay Arrival Instance
#                                                                               (FF)  (ps)  (ps)   (ps)  Location
#-------------------------------------------------------------------------------------------------------------
  RAM_MEM_mem_addr_reg_reg[0]/CLK -       -      R     (arrival)              337     -     0     0       0  (-,-)
  RAM_MEM_mem_addr_reg_reg[0]/QN  -       CLK->QN R    DFFHQNx2_ASAP7_75t_SL    3   3.8    20    33      33  (-,-)
  g50448/Y                        -       A->Y   F     INVx2_ASAP7_75t_SL       2   2.9    11     6      39  (-,-)
  g50445/Y                        -       B->Y   F     AND2x4_ASAP7_75t_SL      8  10.4    14    16      55  (-,-)
  g49925/Y                        -       B->Y   F     AND2x4_ASAP7_75t_SL      8   6.8    11    14      69  (-,-)
  g48634__1881/Y                  -       B->Y   F     AND2x2_ASAP7_75t_SL      4   3.6    12    14      84  (-,-)
  g48505__3680/Y                  -       C1->Y  F     AO332x1_ASAP7_75t_SL     1   0.9    11    21     104  (-,-)
  g191/Y                          -       B->Y   R     NOR3xp33_ASAP7_75t_SL    1   2.1    35    17     122  (-,-)
  g190/Y                          -       D->Y   F     NAND4xp75_ASAP7_75t_SL   1   1.9    22    10     131  (-,-)
  g55066/Y                        -       A->Y   R     NOR3x1_ASAP7_75t_SL      1   2.1    20    10     141  (-,-)
  g55068/Y                        -       A->Y   F     NAND4xp75_ASAP7_75t_SL   2   1.6    21     9     151  (-,-)
  g2/Y                            -       B->Y   R     XNOR2x2_ASAP7_75t_SL     1   2.1    10    19     169  (-,-)
  g18657/Y                        -       D->Y   F     NAND4xp75_ASAP7_75t_SL   1   1.0    10     8     178  (-,-)
  COMP_out_reg/D                  <<<     -      F     DFFHQNx2_ASAP7_75t_SL    1     -     -     0     178  (-,-)
#-------------------------------------------------------------------------------------------------------------
```

Fig. 10: Timing Violation 1.

We further granularized the memory module. to 8x32 word memories as opposed to 4x64 word memories, and also added a pipeline stage between the memory output and the comparator input to improve timing. The negative slack was improved by 20ps with this update, however the timing violation persists. {Fig. 11}

```
Path 1: VIOLATED (-161 ps) Setup Check with Pin ramout_ppln0_reg[2]/CLK->D
        Group: clk
    Startpoint: (R) RAM_MEM_mem_addr_reg_reg[3]/CLK
         Clock: (R) clk
      Endpoint: (R) ramout_ppln0_reg[2]/D
         Clock: (R) clk

                  Capture       Launch
     Clock Edge:+       1            0
     Src Latency:+      0            0
     Net Latency:+      0 (I)        0 (I)
         Arrival:=      1            0

           Setup:-      6
   Required Time:=     -5
    Launch Clock:-      0
       Data Path:-    155
           Slack:=   -161

#-------------------------------------------------------------------------------------------------------------
#          Timing Point           Flags    Arc    Edge          Cell           Fanout Load Trans Delay Arrival Instance
#                                                                               (FF)  (ps)  (ps)   (ps)  Location
#-------------------------------------------------------------------------------------------------------------
  RAM_MEM_mem_addr_reg_reg[3]/CLK -       -      R     (arrival)              347     -     0     0       0  (-,-)
  RAM_MEM_mem_addr_reg_reg[3]/QN  -       CLK->QN R    DFFHQNx2_ASAP7_75t_SL    3   5.1    23    34      34  (-,-)
  g19/Y                           -       A->Y   F     INVx3_ASAP7_75t_SL       2   2.9    10     5      40  (-,-)
  g53920/Y                        -       B->Y   F     AND2x4_ASAP7_75t_SL      8   9.1    13    15      54  (-,-)
  g53918/Y                        -       B->Y   F     AND2x2_ASAP7_75t_SL      8   6.2    16    17      72  (-,-)
  g49942__1666/Y                  -       B->Y   F     AND2x2_ASAP7_75t_SL      4   3.9    12    16      87  (-,-)
  g53431/Y                        -       B1->Y  R     AOI22x1_ASAP7_75t_SL     1   1.0    15     8      96  (-,-)
  g53430/Y                        -       D->Y   R     AND4x1_ASAP7_75t_SL      1   1.0    11    16     112  (-,-)
  g939/Y                          -       A->Y   R     AND3x1_ASAP7_75t_SL      1   1.5    12    15     127  (-,-)
  g53287/Y                        -       A->Y   F     NAND2x1_ASAP7_75t_SL     1   2.1    12     6     133  (-,-)
  g53305/Y                        -       A->Y   R     NOR4xp75_ASAP7_75t_SL    1   1.8    25    11     144  (-,-)
  g53587/Y                        -       C->Y   F     NAND3x1_ASAP7_75t_SL     2   1.3    14     5     149  (-,-)
  g930/Y                          -       A->Y   R     INVx1_ASAP7_75t_SL       1   1.0     9     6     155  (-,-)
  ramout_ppln0_reg[2]/D           <<<     -      R     DFFHQNx1_ASAP7_75t_SL    1     -     -     0     155  (-,-)
#-------------------------------------------------------------------------------------------------------------
```

Fig. 11: Timing Violation 2.

We modified the memory better divide the address decode for memory read. In the first cycle, data from each memory block is stored in a data buffer and the required data buffer is picked in the next cycle. This can be loosely considered as a hierarchical memory. Even with this improvement, we saw negative slack reduced by 25ps, but the timing violation was not resolved. {Fig. 12}

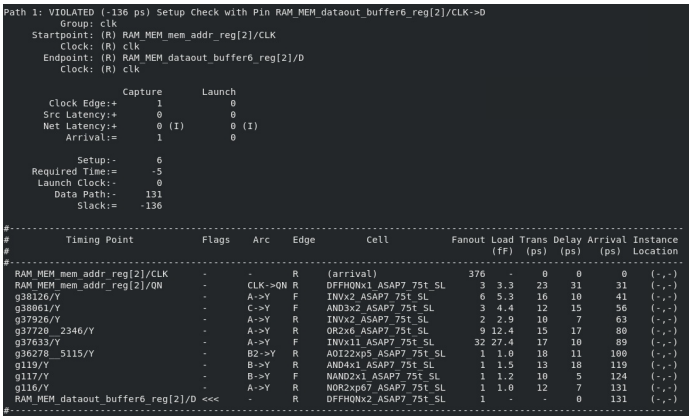


Fig. 12: Timing Violation 3.

So, we updated the timing constraint from 0.6 to 1.0 in the SDC file, and even used the parameters to test a significantly smaller 64 word memory. But it still showed timing violation; this made us question the time scale unit of the SDC, which was not mentioned in the SDC file provided in the project description. Therefore, we updated the SDC file by setting the time unit in ns using the following command – `set_units -time ns`.

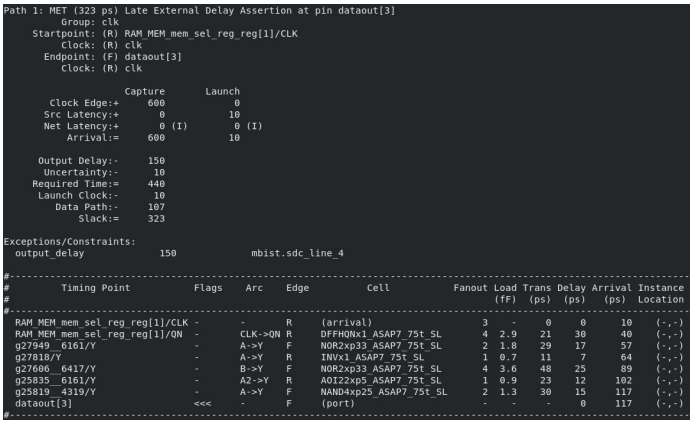


Fig. 13: Timing met with clock period = 0.6ns

With this update, we were able to met the timing for the MBIST with a clock period = 0.6ns. {Fig. 13}

E. Stage V - Improving Clock Speed

Finally, since we were seeing a large positive slack of 323ps with the SDC update, we decided to increase the clock speed – reducing the clock period to 0.3ns.

Hence, we were able to meet timing even at 2x the required clock speed. {Fig. 14}

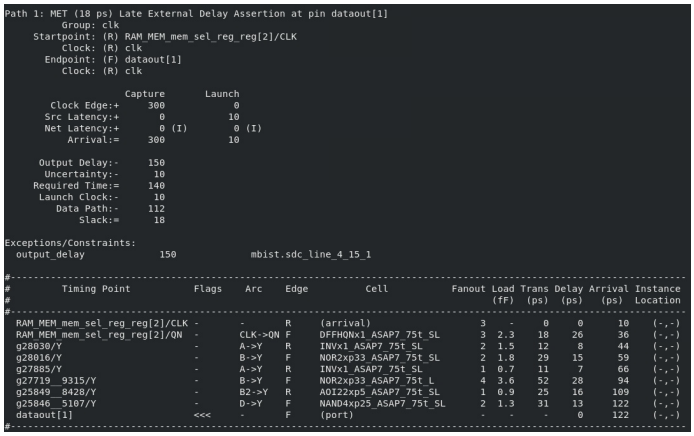


Fig. 14: Timing met with 2x faster clock (period = 0.3ns)

IV. Conclusion

The MBIST designed in this project successfully generated the test pattern required for testing a 256 x 4 bit SRAM cell. The RTL design met all the timing constraints mentioned in the SDC and can even perform at half the time-period mentioned in the SDC.

The entire RTL code, genus timing reports as well as run scripts and input files have been uploaded to GitHub.[1]

V. Acknowledgement

This project work would not have been possible without the tremendous efforts provided by the course instructor Prof. Azeez Bhavnagarwala and the Teaching Assistant, Aishwarya Raj, in developing the foundational background knowledge upon which this project stands.

[1] https://github.com/catchyviddy/MBIST-256x4b