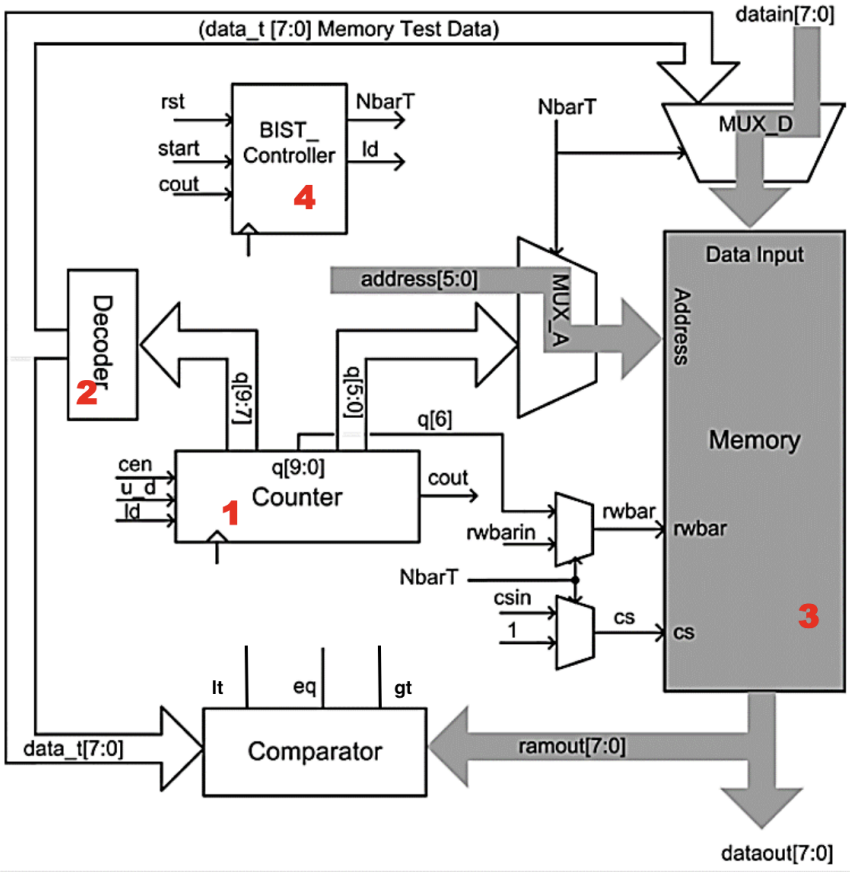
MBIST PROJECT

Objective:

In this project, you will design and verify a complete MBIST (Memory Built-In-Self-Test) system for a 64 x 8-bit single port SRAM using SystemVerilog. The project involves implementing individual submodules such as the controller, decoder, counter, multiplexer, and memory. Detailed descriptions of each submodule are provided and their integration is illustrated in the architecture.

Architecture:



Task:

Design each submodule as per the description provided below. Write a testbench for each submodule and verify functionality through simulation. Finally, integrate all submodules into the BIST top module and simulate the complete system.

Comparator:

1. The comparator module is used within the BIST (Built-In-Self-Test) system to compare two 8-bit values:
    a. The expected data pattern (data_t) generated by the decoder
    b. The actual data read from the SRAM (ramout).
2. It outputs comparison flags indicating whether the actual memory data matches, is greater than, or less than the expected value. This comparison result is used to detect memory faults during test algorithms.

Port	Direction	Width	Description
data_t	Input	[7:0]	Expected data value (from decoder)
ramout	Input	[7:0]	Data read from SRAM
gt	Output	1-bit	Set to 1 if data_t > ramout
eq	Output	1-bit	Set to 1 if data_t == ramout
lt	Output	1-bit	Set to 1 if data < ramout

Counter:

1. The Counter module is a configurable, general purpose synchronous counter used in the BIST system to generate:
    a. Sequential memory addresses
    b. Timing control for BIST sequences
    c. Pattern selection bits for the decoder module
    d. Supports up-counting, down-counting, load functionality, and carry out detection.
2. Parameterize the module using a parameter named **length** with a default value of 10.
3. On each positive edge check if **cen** is high then:
    a. If **ld** is high -> counter is loaded with **d_in**
    b. Else if **u_d** = 1 -> counter increments else it will decrement.

Port	Direction	Width	Description
d_in	Input	[length-1:0]	Value to load into the counter
clk	Input	1-bit	Clock signal
ld	Input	1-bit	Load enable (loads d_in into counter)
u_d	Input	1-bit	Direction control (1 = up, 0 = down)
cen	Input	1-bit	Count enable; must be high to count
q	Output	[length-1:0]	Current counter value

cout	Output	1-bit	Carry-out bit (overflow/underflow)

Decoder:

1. The decoder module is a simple combinational logic block that translates a 3-bit input selector into one of several predefined 8-bit data patterns. These patterns are used as test data during memory Built-in-Self-Test produces to detect faults such as stuck-at, pattern sensitivity, and coupling faults in SRAM.
2. If the selector (q) does not match any defined case then data_t is assigned as unknown, indicating an invalid pattern.

Port	Direction	Width	Description
q	Input	3-bit	Pattern selector code
data_t	Output	8-bit	Corresponding pattern

Selector(q)	Pattern (data_t)
3'b000	8'b10101010
3'b001	8'b01010101
3'b010	8'b11110000
3'b011	8'b00001111
3'b100	8'b00000000
3'b101	8'b11111111

Controller:

1. Implements a simple 2-state FSM to control the BIST process. It switches between **reset** and **test** states based on the **start** and **cout** signals.
2. FSM starts in **reset** state and moves to **test** when **start** is high; otherwise it will remain in **reset**.
3. While in the **test** state, if **cout** becomes high, it transitions back to **reset**; otherwise, it stays in **test**.
4. Outputs:
    a. **NbarT** = 1 during test
    b. **Id** = 1 during reset

Port	Dir	width	Desc
start	Input	1-bit	Triggers the BIST process
rst	Input	1-bit	Resets the FSM to reset state
clk	Input	1-bit	Clock Signal
cout	Input	1-bit	Signals test completion
NbarT	Output	1-bit	High during test phase
ld	Output	1-bit	High during reset phase (loads counter)

Multiplexer:

1. Selects between normal operation inputs and BIST mode inputs for memory address or data, based on the **NbarT** control signal.
2. If **NbarT** = 0: select normal Input
3. If **NbarT** = 1: selects BIST-generated input
4. The module is parameterized with a generic **WIDTH**, allowing it to be reused for both address (6-bit) and data (8-bit) switching.
5. A single multiplexer module was reused instead of having two separate hard-codes.

Port	Direction	Width	Description
normal_in	Input	[WIDTH-1:0]	Input from normal operation
bist_in	Input	[WIDTH-1:0]	Input from BIST logic
NbarT	Input	1-bit	Select signal (0:normal, 1: BIST)
out	Output	[WIDTH-1:0]	Output to SRAM(address or data)

SRAM:

1. Implements a 64 x 8-bit single-port synchronous SRAM used as the memory under test in the BIST system. It supports write and read operations on the rising edge of the clock. The module emulates typical single-port memory behavior found in FPGAs and ASICs.
2. On every positive clock edge check if **cs** (chip select) is high:
    a. If **rwbar** is low (0 = write), the input data **ramin** is written to the memory location specified by **ramaddr**.
    b. Always store the address **ramaddr** into a temporary register (assume addr_reg) for subsequent access.
3. The output **ramout** is continuously assigned the value at **ram[addr_reg]** only when **cs** is high and **rwbar** is high. If either **cs** is not high or **rwbar** is low, **ramout** defaults to ‘0.

Port	Direction	Width	Description
ramaddr	Input	6-bit	Address for memory access
ramin	Input	8-bit	Data to be written into memory
rwbar	Input	1-bit	(0= write, 1 = read)
clk	Input	1-bit	Clock signal (rising edge triggered).
cs	input	1-bit	Active high chip select
ramout	Output	8-bit	Data read from memory

BIST (TOP module):

1. The BIST module integrates all submodules required to perform memory self-testing on a 64 x 8-bit single-port SRAM. It operates in both normal mode and BIST mode, controlled by external signals.
2. Parameterized the BIST module using **size** and **length** parameters to define address and data widths , respectively.
3. All submodules must be instantiated and connected according to the provided architecture diagram.
4. The memory address and data signals are selected via two instances of a parameterized multiplexer, which chooses between normal and BIST inputs based on the **NbarT** signal.
5. The write enable (**we**) signal for SRAM is driven by either **rwbarin** (in normal mode) or by bit **q[6]** from the counter output(in BIST mode), depending on the value of **NbarT** which comes for the Controller module.
6. The chip select signal **cs** should be connected to **csin** in normal mode and set to 1 in test mode(**NbarT = 1**).
7. Determine the **fail** output at every positive edge of the clock.
8. A **fail** condition is triggered during a BIST read operation (**NbarT** = 1, **opr** = 1, and **rwbarin** (during normal mode) or **q[6]** (in test mode))and if the comparator detects a data mismatch (**eq = 0**). **fail** condition should be set to 0 otherwise. **Tip**: create a temp variable to select one of the variable **rwbarin** and **q[6]** based on **NbarT**
    a. For Normal mode: **(NbarT && rwbarin && ~eq && opr)**
    b. For Bist mode: **(NbarT && q[6] && ~eq && opr)**
9. The output from memory (**ramout**) is continuously made available at the module's dataout port for observation in both normal and bist mode.

Parameter	Description

size	Address width for SRAM (default: 6-bits)
length	Data width for SRAM (default: 8 bits)

Port	Direction	Width	Description
start	Input	1-bit	Sent to controller to change from normal to test
rst	Input	1-bit	Resets the internal FSM and counter
clk	Input	1-bit	Clock signal
csin	Input	1-bit	Chip Select input (used in normal mode)
rwbarin	Input	1-bit	Read/Write control for normal mode
opr	input	1-bit	Check for fail condition only when opr is 1
address	input	[size-1:0]	Address input for normal mode
datain	input	[length-1:0]	Data input for normal write operation
dataout	output	[length-1:0]	Data output from SRAM
fail	Output	1-bit	Indicates memory test failure during BIST

Timing Constraints:

Apply the following SDC constraints to your synthesis:

```
create_clock -period 6 -name clk [get_ports clk]
set_input_delay 0.1 -clock clk [all_inputs]
set_output_delay 0.15 -clock clk [all_outputs]
set_load 0.1 [all_outputs]
set_max_fanout 1 [all_inputs]
set_fanout_load 8 [all_outputs]
set_clock_uncertainty 0.01 [all_clocks]
set_clock_latency 0.01 -source [get_ports clk]
```

Simulation & Synthesis:

1. Use **Synopsys VCS** to perform functional simulation of your design. Verify the correctness of your implementation using a waveform viewer.
2. For synthesis, use **Synopsys Design Compiler**. Ensure your design meets the provided timing constraints. Address any setup timing violations either by optimizing your RTL design or by adjusting the clock cycle time accordingly.

Deliverables:

1. Submit your design and testbench files in autograder (Project Part 1, Project Part 2) for simulation
2. Submit the design, script file and sdc file for synthesis in autograder(Project Part 3)
3. Make sure you label the design and tb files as follows:
    a. comparator.sv, controller.sv, decoder.sv, counter.sv, multiplexer.sv, sram.sv, bist.sv
    b. tb_comparator.sv, tb_controller.sv, tb_decoder.sv, tb_counter.sv, tb_multiplexer,sv, tb_sram.sv, tb_bist.sv
    c. controller.sdc for constraints file name.