Lab4_a- BRAM ECC

In this lab we are going to generate a Block Memory (BRAM) with a built-in Error Correction Code (ECC). The ECC detects and corrects a single bit error in the BRAM. It can also detect a multi bit error but it can't correct it.

Block RAM (BRAM) is a type of random access memory embedded throughout an FPGA for data storage.

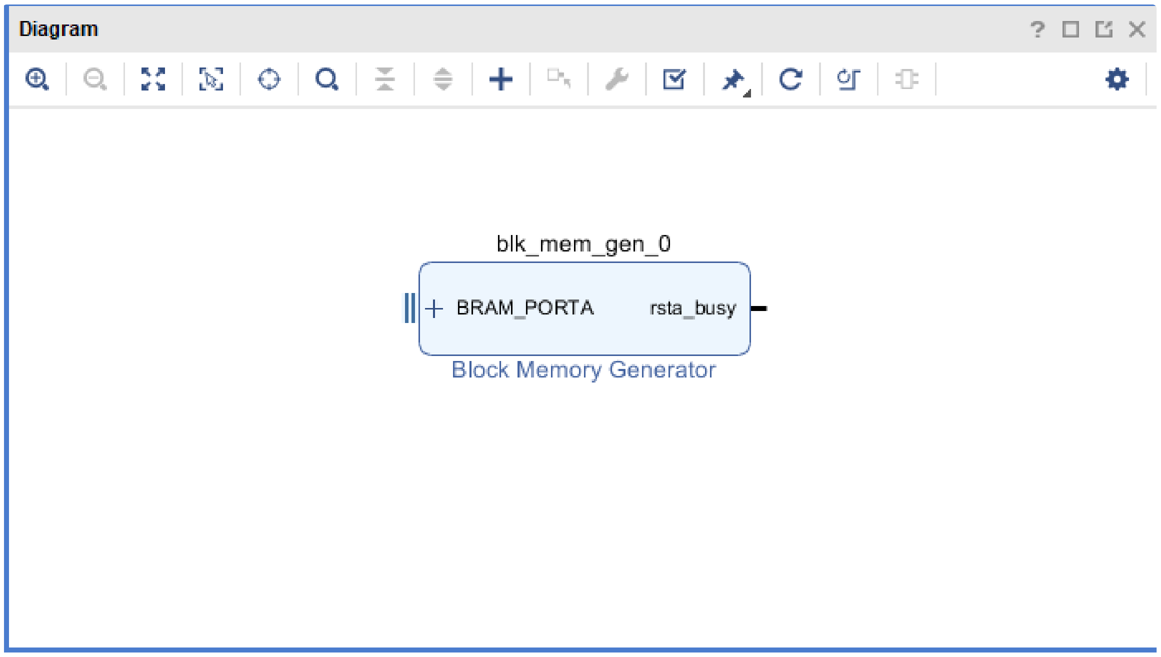
You can use BRAM to accomplish the following tasks:

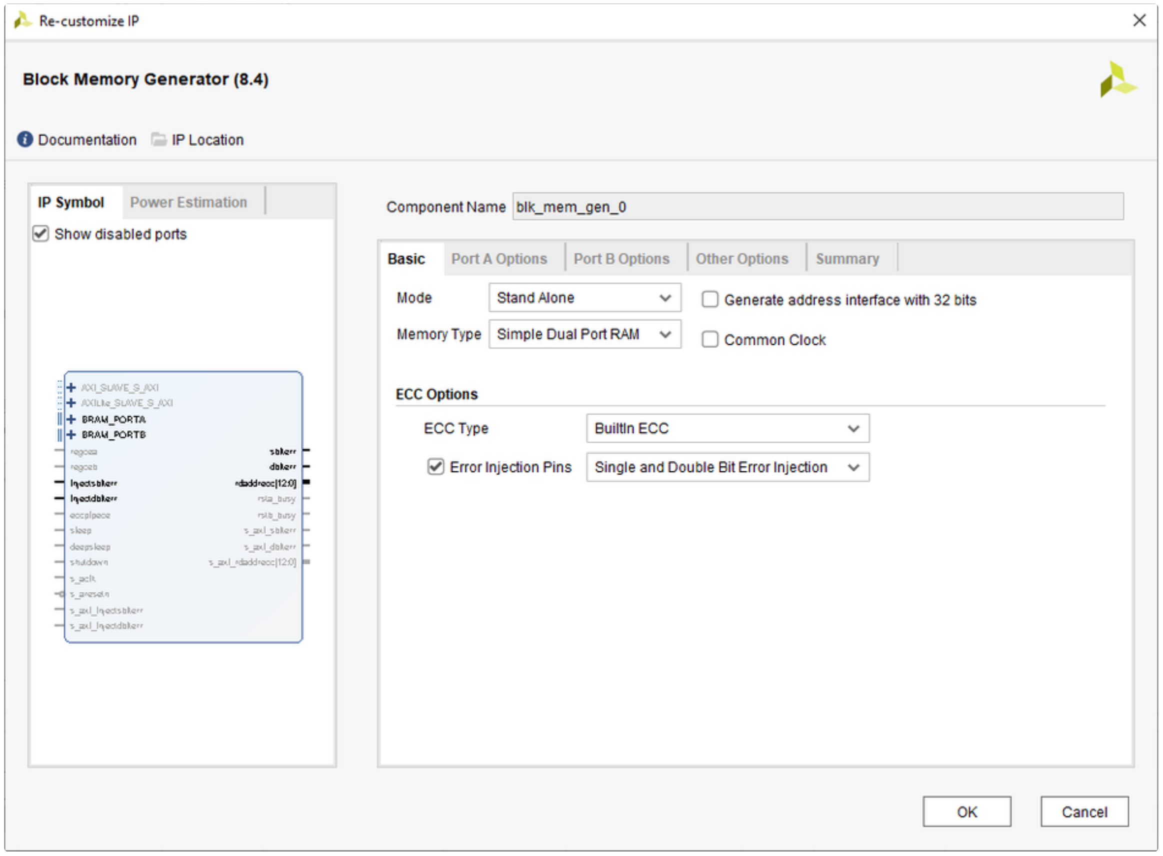
- Transfer data between multiple clock domains by using local FIFOs
- Transfer data between an FPGA target and a host processor by using a DMA FIFO
- Transfer data between FPGA targets by using a peer-to-peer FIFO
- Store large data sets on an FPGA target more efficiently than RAM built from look-up tables

It is important for critical applications to use a form of a correction code in order to ensure smooth functioning. For this reason we will use the Block Memory Generator IP, given by Xilinx, which has a built-in ECC and it also has injection bit signals to simulate a single/double bit error.

Steps:

1. Launch Vivado
2. Click Create Project
3. Click Next
4. Specify Project name and project Location and click Next
5. Choose the first bullet (RTL Project) and check the box under (**Do not specify sources at this time**) and click Next
6. Choose from the TAB->Boards the board **ZYBO Z7-10** and click Next
7. Click Finish
8. From the Flow Navigator Tab on the left Click on **Create Block Design** and then OK
9. Click on the Add IP icon
   +
10. Search Block Memory Generator and double click it
11. The initial IP should look like this



12. Now double click on the IP to enter its settings
13. Fill the settings based on the images below.
    a. 

Re-customize IP

Block Memory Generator (8.4)

Documentation IP Location

IP Symbol | Power Estimation

☑ Show disabled ports

Component Name: blk_mem_gen_0

Basic | Port A Options | Port B Options | Other Options | Summary

Mode: Stand Alone

☐ Generate address interface with 32 bits

Memory Type: Simple Dual Port RAM

☐ Common Clock

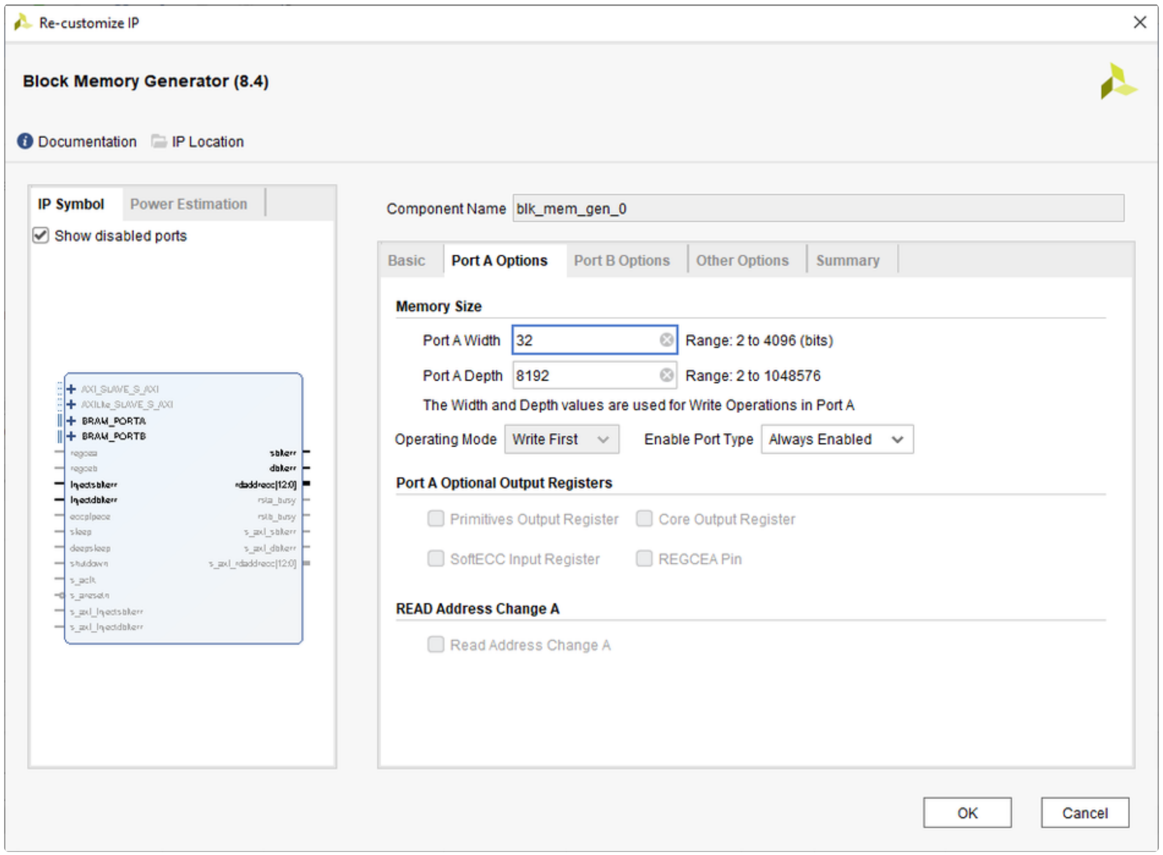
ECC Options

ECC Type: BuiltIn ECC

☑ Error Injection Pins: Single and Double Bit Error Injection

OK Cancel

b.



Re-customize IP

Block Memory Generator (8.4)

Documentation IP Location

IP Symbol | Power Estimation

☑ Show disabled ports

Component Name: blk_mem_gen_0

Basic | **Port A Options** | Port B Options | Other Options | Summary

Memory Size

Port A Width: 32 Range: 2 to 4096 (bits)

Port A Depth: 8192 Range: 2 to 1048576

The Width and Depth values are used for Write Operations in Port A

Operating Mode: Write First Enable Port Type: Always Enabled

Port A Optional Output Registers

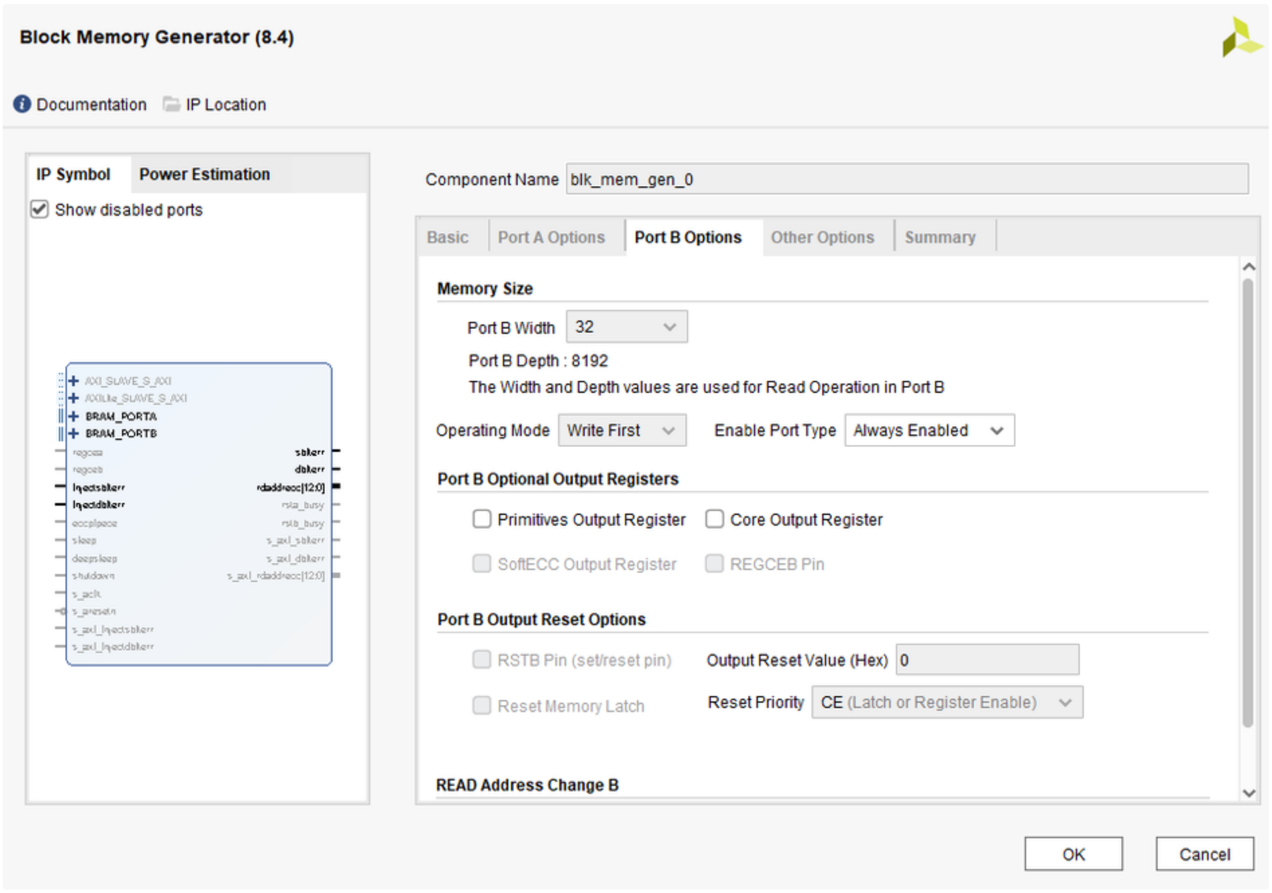
☐ Primitives Output Register ☐ Core Output Register

☐ SoftECC Input Register ☐ REGCEA Pin

READ Address Change A

☐ Read Address Change A

OK Cancel



Block Memory Generator (8.4)

Documentation IP Location

IP Symbol | Power Estimation

☑ Show disabled ports

Component Name: blk_mem_gen_0

Basic | Port A Options | **Port B Options** | Other Options | Summary

Memory Size

Port B Width: 32

Port B Depth : 8192

The Width and Depth values are used for Read Operation in Port B

Operating Mode: Write First Enable Port Type: Always Enabled

Port B Optional Output Registers

☐ Primitives Output Register ☐ Core Output Register

☐ SoftECC Output Register ☐ REGCEB Pin

Port B Output Reset Options

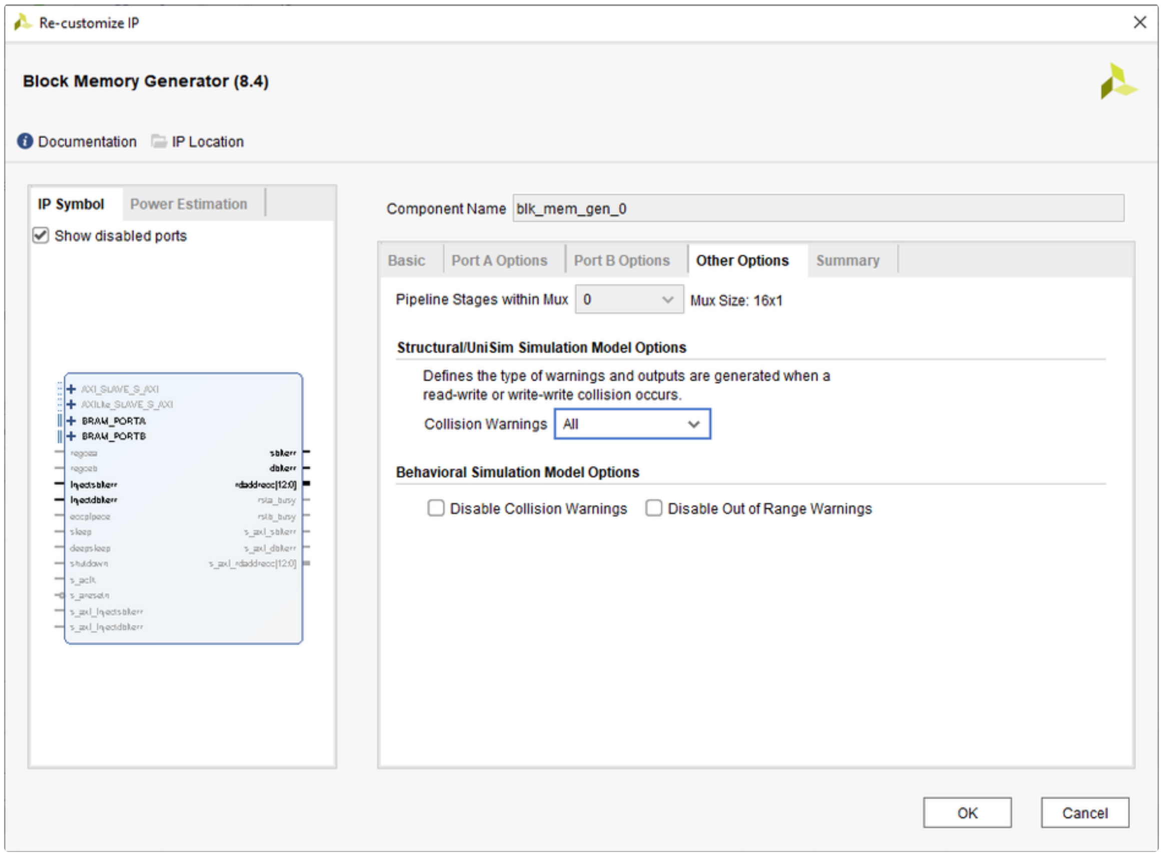
☐ RSTB Pin (set/reset pin) Output Reset Value (Hex): 0

☐ Reset Memory Latch Reset Priority: CE (Latch or Register Enable)

READ Address Change B

OK Cancel

c.

d. 
Re-customize IP

Block Memory Generator (8.4)

Documentation   IP Location

IP Symbol | Power Estimation

☑ Show disabled ports

Component Name: blk_mem_gen_0

Basic | Port A Options | Port B Options | Other Options | Summary

Pipeline Stages within Mux: 0   Mux Size: 16x1

Structural/UniSim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

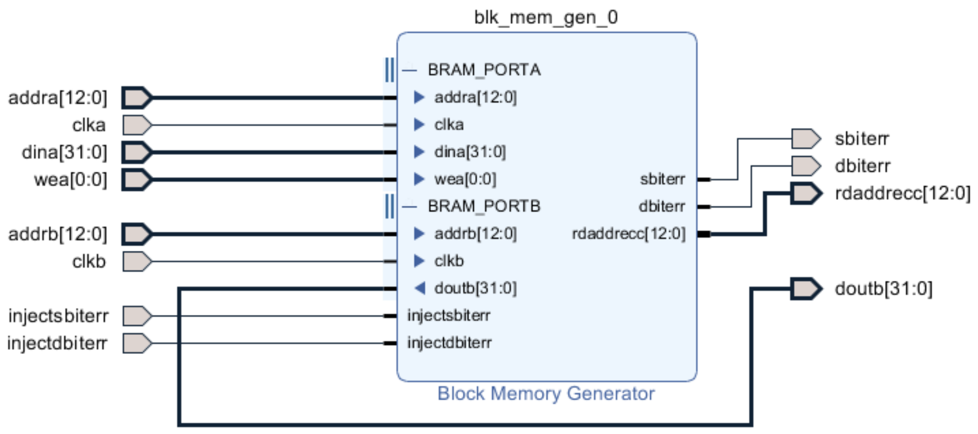
Collision Warnings: All

Behavioral Simulation Model Options

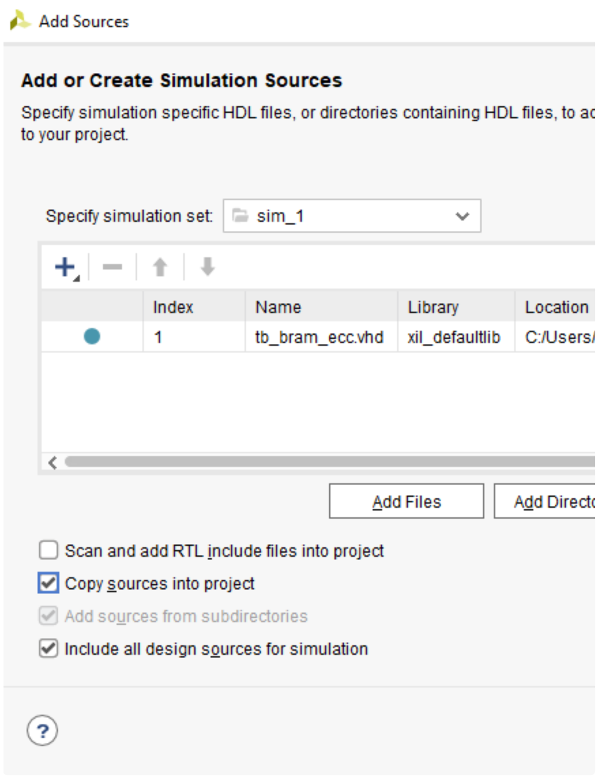
☐ Disable Collision Warnings   ☐ Disable Out of Range Warnings

OK   Cancel

14. Click OK
15. Select every signal of the IP and make it external
16. Remove the suffix _0 in every external signal
17. The IP now should look like the image below



18. Save the Design
19. In the Design Sources tab, right click on the design_1 block and Click on **Generate Output Products**
20. In the Design Sources tab now, again right click on the design_1 block and Click on the **Create HDL Wrapper**
21. Now in order to test the ECC we will perform a **simulation** where we will inject a single bit error and a double bit error and we will observe the behavior of the IP.
22. On the Flow Navigation panel on the left click **Add Sources → Add or Create Simulation Sources.**
23. Click on the **Add Files** and navigate to the **testbench** file that we provided you

24. 

Add Sources

Add or Create Simulation Sources

Specify simulation specific HDL files, or directories containing HDL files, to ac
to your project.

Specify simulation set: sim_1

	Index	Name	Library	Location
●	1	tb_bram_ecc.vhd	xil_defaultlib	C:/Users/

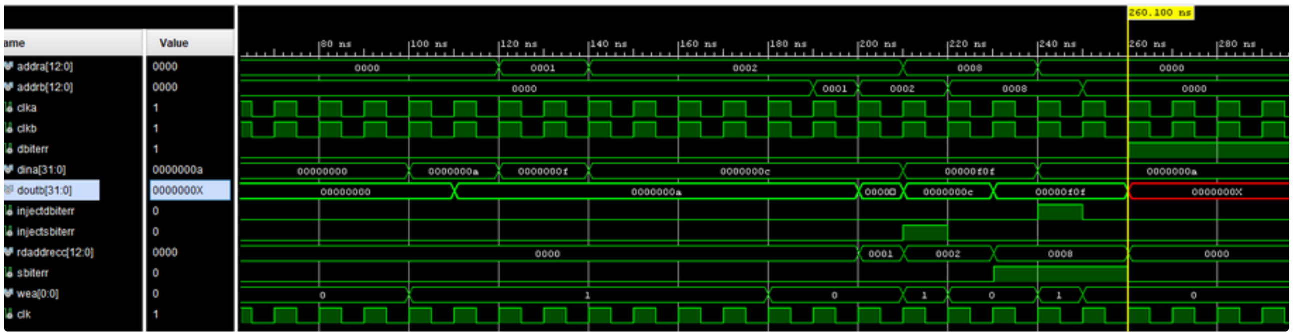
- [ ] Scan and add RTL include files into project
- [x] Copy sources into project
- [x] Add sources from subdirectories
- [x] Include all design sources for simulation

25. Click Finish

26. Open and Read the testbench and the comments to understand the simulation that we want to analyze.

27. Now run the Simulation by clicking the Run Simulation option in the Flow Navigator panel.

28. The waveform of the simulation should look like the image below.



ame	Value
addra[12:0]	0000
addrb[12:0]	0000
clka	1
clkb	1
dbiterr	1
dina[31:0]	0000000a
doutb[31:0]	0000000X
injectdbiterr	0
injectsbiterr	0
rdaddrecc[12:0]	0000
sbiterr	0
wea[0:0]	0
clk	1

Lab4_b- BRAM ECC on the FPGA

Now you will extend your project in order to test its behavior in the FPGA.

In order to download the BRAM ECC project to the FPGA, you should make some changes.

1. You need to downgrade the size of the BRAM to 4*8
2. You need to fill the Bram addresses with values when the FPGA is powered on or after reset
3. You need to implement a logic that checks if the user wants to inject fault/fault and pass the fault to the BRAM
4. You need to have alarm LEDs that inform the user that a single/double error is detected

FPGA operation:

When the FPGA is programmed it should work like this:

1. During power on the BRAM should be filled with values that equal their values (for example address 0x1 should have the value 0x1, and address 0x7 should have the value 0x7)

2. After this LED3 of the board should be '1', indicating that the Bram is filled up with the correct values
3. The user can see the values of the Bram by changing the values of the switches. The leds on top of the switches show the corresponding values of the address given.
4. Now the user can inject Fault/Faults in order to Test the behavior
5. Use BTN2 for single-fault injection and BTN3 for double-fault injection
6. By pressing one of these buttons we inject fault/faults in the BRAM and the alarm led should be turned to one
7. The RGB LD6 is used as the alarm led. The Red indication informs the user that there is a single fault detected in the current address (given by the switches) and the Green indication informs the user of a double fault detection.
8. The BTN1 is used as a select signal of a multiplexer that chooses the values of the LEDs
   a. The LEDs will either show the value of the current address or the Address of the ECC detected
9. The BTN0 is the Reset signal for the FPGA

Steps :

1. Create a new project for the ZYBO-Z7-10 FPGA (close the previous project)
2. Create a block design
3. Add the Block Memory Generator IP
   a. Basic Settings
      i. Mode: Stand Alone
      ii. Memory Type: Simple Dual Port RAM
      iii. ECC Type: Soft ECC
      iv. Check the Error injection Pins box
      v. Select from the dropdown the Single and double Bit Error Injection
   b. Port A Settings:
      i. Port A width: 4
      ii. Port A depth: 8
      iii. Enable Port Type: Always Enabled
   c. Port B Settings:
      i. Port B width: 4
      ii. Enable Port Type: Always Enabled
      iii. Uncheck the Primitives Output Register box
   d. Click OK
4. Now you need to create two VHDL files
   a. The FSM machine will have three states
      i. idle
      ii. fill bram
      iii. waiting for injection
   b. A simple multiplexer which will pass to the LEDs of the FPGA either the values of the BRAM or the address of the ECC detection :
      i. It will have two inputs
         1. The doutb[3:0]
         2. The rdaddrecc[2:0]
      ii. An external select signal (A button)
      iii. The LEDs as an output
      iv. In our case, the possible values of the BRAM will be from 0 to 7 so you only need the LEDS[2:0]. The LED[3] will be used for another signal
   c. **The FSM:**
      i. Create a clocked process with an asynchronous reset. The reset should set the FSM to state IDLE.

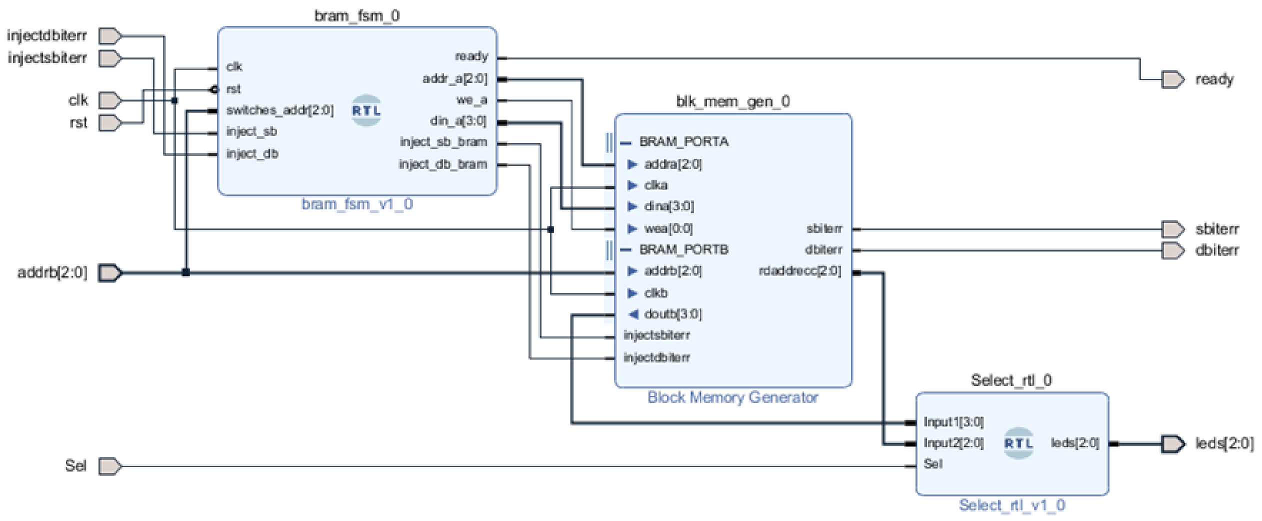
ii. The FSM should
  1. Initialize the process values
  2. Fill the BRAM
  3. Indicate the BRAM is filled and wait for fault injections

iii. During the IDLE state
  1. The FSM should reset the following signals:
     a. ready
     b. we_a
     c. addr_a
     d. din_a
     e. counter
  2. After the initialization, the FSM should move to the fill_bram STATE

iv. During the fill_bram STATE:
  1. Indicate ready='0'
  2. Use the counter to fill the Bram with the corresponding values
     a. example:
        i. addr_a<=std_logic_vector(counter);
        ii. din_a<='0' & std_logic_vector(counter);
        iii. we_a<='1';
        iv. counter<=counter+'1';
  3. **Be careful:**
     a. The counter should be unsigned in order to perform addition
  4. If the current value of the counter is “111” and you add +1 the next value will be set to “000”
  5. After every address of the Bram is filled up moved to the next state of Waiting_injection

v. During the Waiting_injection STATE:
  1. The ready signal (LED[3]) should be '1'
  2. The FSM will check if the BTN2 or BTN3 (Single or Double Fault INJECTION) buttons are pressed and if this case is true:
     a. The FSM will write the address indicated by the switches with the corresponding value the same way as it did during the fill_bram state
     b. It will enable the single/double injection pins of the BRAM for one cycle.
  3. The FSM stays in this State until reset or until the FPGA is powered off.

5. The Block Desing of your project should look like the following image



bram_fsm_0
injectdbiterr
injectsbiterr
clk
rst
clk
rst
switches_addr[2:0]
inject_sb
inject_db
ready
addr_a[2:0]
we_a
din_a[3:0]
inject_sb_bram
inject_db_bram
RTL
bram_fsm_v1_0
blk_mem_gen_0
BRAM_PORTA
addra[2:0]
clka
dina[3:0]
wea[0:0]
BRAM_PORTB
addrb[2:0]
clkb
doutb[3:0]
injectsbiterr
injectdbiterr
sbiterr
dbiterr
rdaddrecc[2:0]
Block Memory Generator
addrb[2:0]
ready
sbiterr
dbiterr
Select_rtl_0
Input1[3:0]
Input2[2:0]
Sel
RTL
leds[2:0]
Select_rtl_v1_0
Sel
leds[2:0]