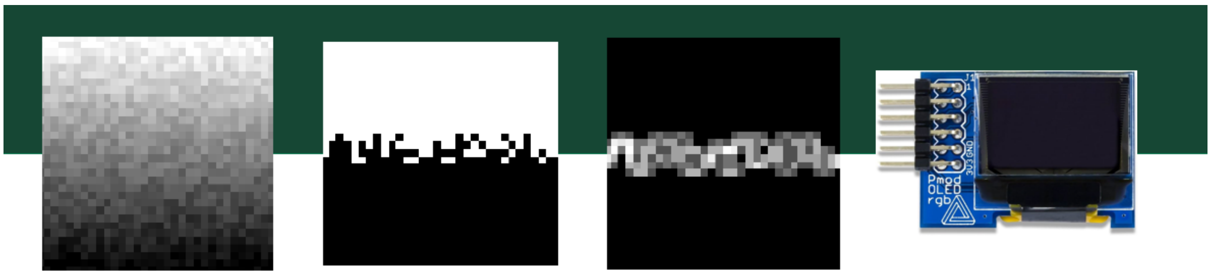
Dev Halai – FPGA Capstone Handout.

Designing a Sobel Image Processor on Basys 3 board.

High level overview of process:



Input is going to be a greyscale image

Apply a **threshold** over it.

Anything over our threshold is white, anything below is black.

Using this threshold apply our **sobel** filter (edge detection)

Output so we can see our result produced by FPGA.

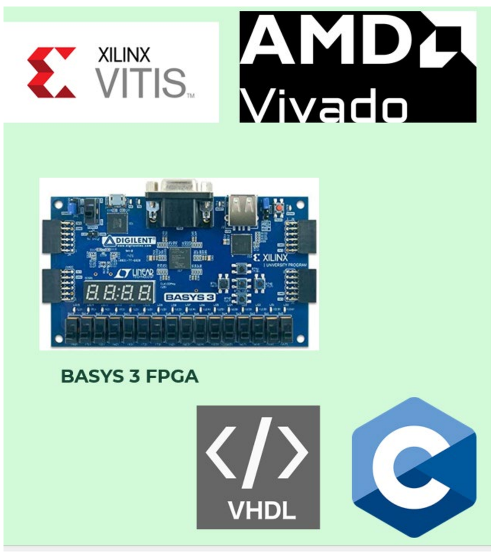
Why? Think of a car or any modern device where you might need visual input to perform other actions. Like a dashboard on a smart car. Needs cameras with data to see where the road is, detect where objects and people are in real time.

Tech Stack used:

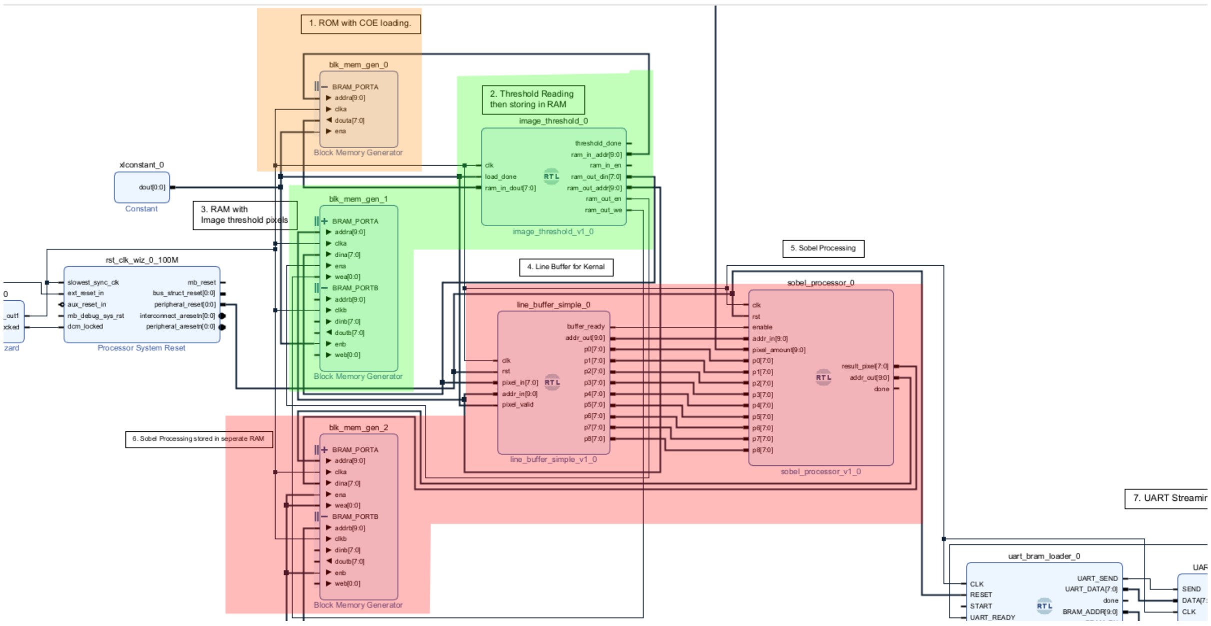
Vitis on **Microblaze** for prototyping, using **C** language.

AMD / Xilinx suite for IDE and debugging tools used whilst working on the **Basys 3** board.

FPGA language used was **VHDL**.



Block Diagram for main design.

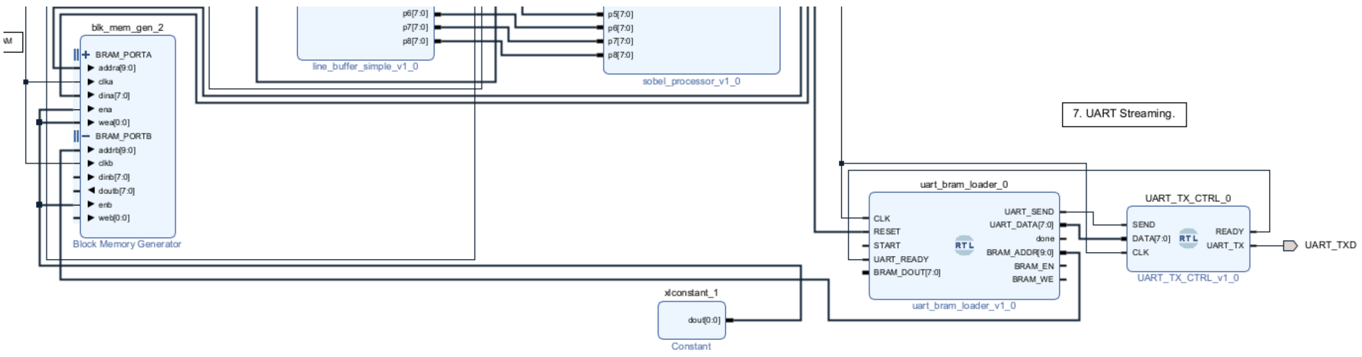


On the left we have the clock wizard to generate a 100 Mhz clock.
In Orange is the image loaded as a COE to initialize our ROM with the grey scale scale.

In Green is the threshold process which takes data from the ROM, applies a threshold filter and stores it in a dual port RAM (blk mem gen 1).

In Pink is our sobel filter. Sobel uses kernal convolution. So for this module we had to work on not just our immediate pixel, but surrounding ones too. A line / data buffer was needed to input the correct data into our sobel filter calculation. Which then the individual pixels could be stored in the BRAM at the correct address.

Uart streaming via VHDL.

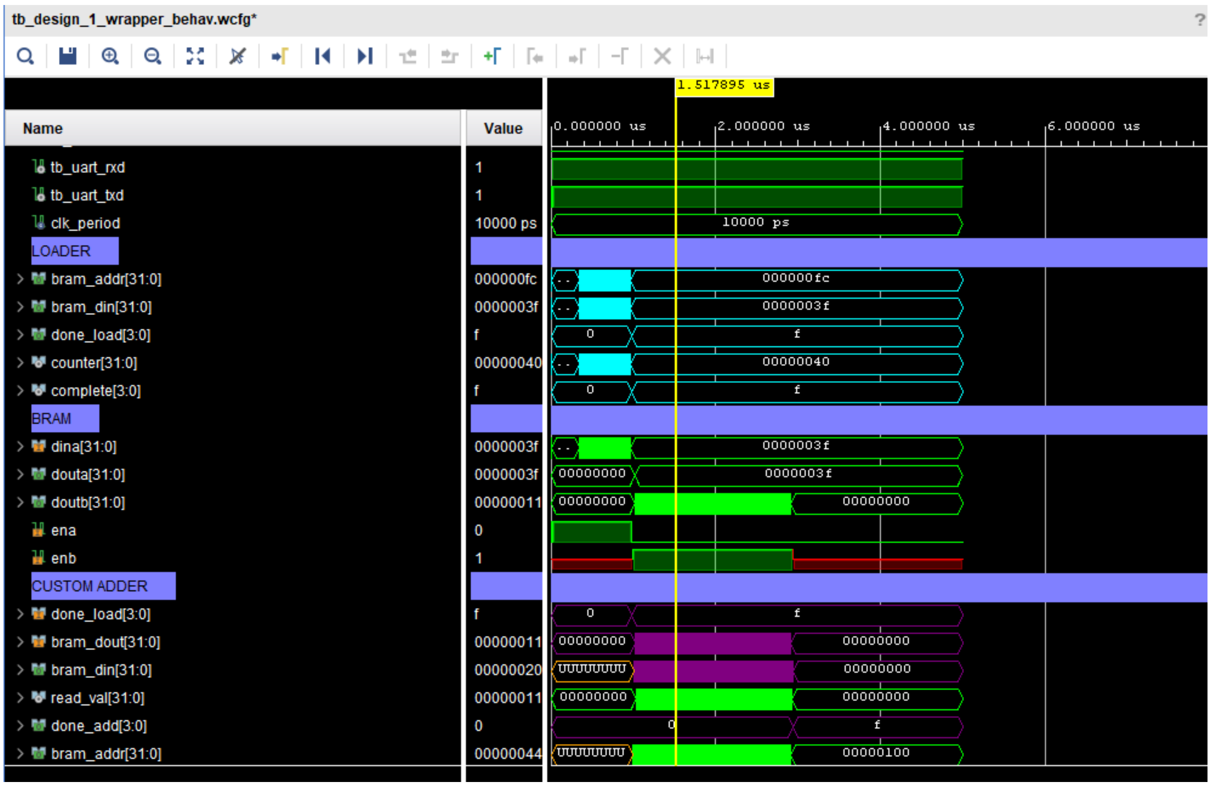


To check if the data was correct I streamed it using UART to Putty with a VHDL module.

Since Uart requires data to be sent 1 bit at a time I had to make a ‘uart bram loader’ which took data from BRAM and sent to putty.

BRAM Sample File.

Since I was going to be using BRAM a lot in my project to store data, read data, manipulate and write back. I decided to create a mini-isolated project focusing on loading and adding numbers and writing back to get comfortable with the process.


tb_design_1_wrapper_behav.wcfg*

Name	Value
tb_uart_rxd	1
tb_uart_txd	1
clk_period	10000 ps
LOADER	
bram_addr[31:0]	000000fc
bram_din[31:0]	0000003f
done_load[3:0]	f
counter[31:0]	00000040
complete[3:0]	f
BRAM	
dina[31:0]	0000003f
douta[31:0]	0000003f
doutb[31:0]	00000011
ena	0
enb	1
CUSTOM ADDER	
done_load[3:0]	f
bram_dout[31:0]	00000011
bram_din[31:0]	00000020
read_val[31:0]	00000011
done_add[3:0]	0
bram_addr[31:0]	00000044

The testbench was incredibly useful here especially with the ability to group / divide and colour data to distinguish between so many signals and off different data widths.

Data is 32 bits since original plan was to either stream or write data back to vitis to use with PMOD. Later changed to 8 bits to save space on FPGA board.

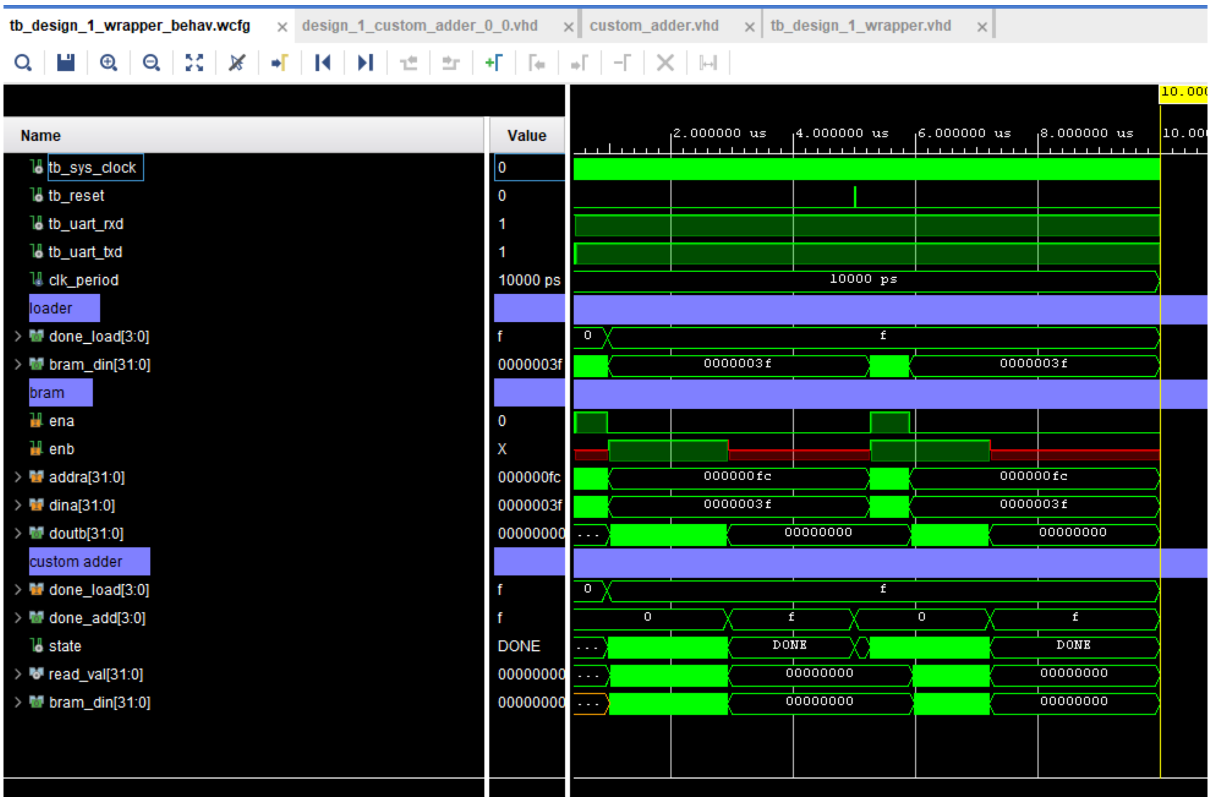
This proved invaluable.

Image Threshold TestBench

Testbench was ran of top model since I wanted to use the data from the COE BRAM loaded into module to test real conditions.

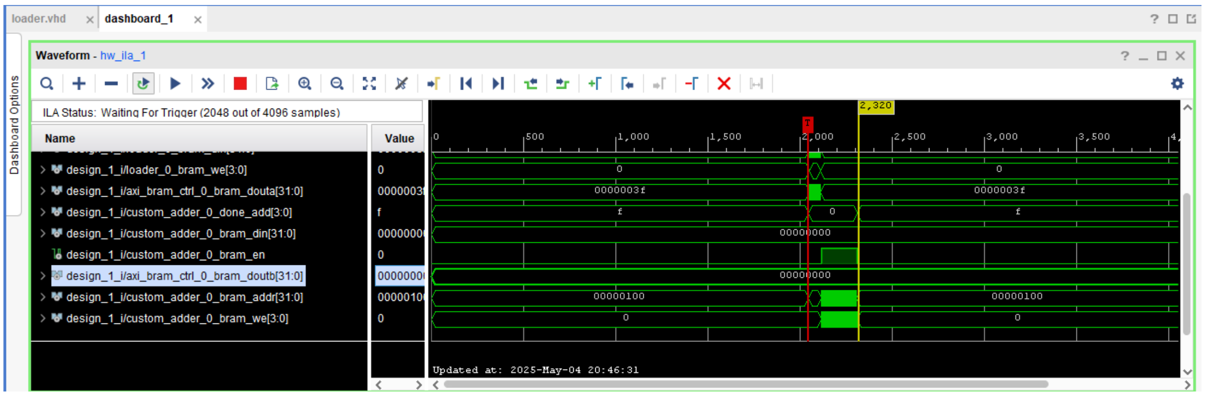
Additional tests

1. Reset model working (makes catching ILA easier). Test bench as expected.



Name	Value
tb_sys_clock	0
tb_reset	0
tb_uart_rxd	1
tb_uart_txd	1
clk_period	10000 ps
loader	
done_load[3:0]	f
bram_din[31:0]	0000003f
bram	
ena	0
enb	X
addra[31:0]	000000fc
dina[31:0]	0000003f
doutb[31:0]	00000000
custom adder	
done_load[3:0]	f
done_add[3:0]	f
state	DONE
read_val[31:0]	00000000
bram_din[31:0]	00000000

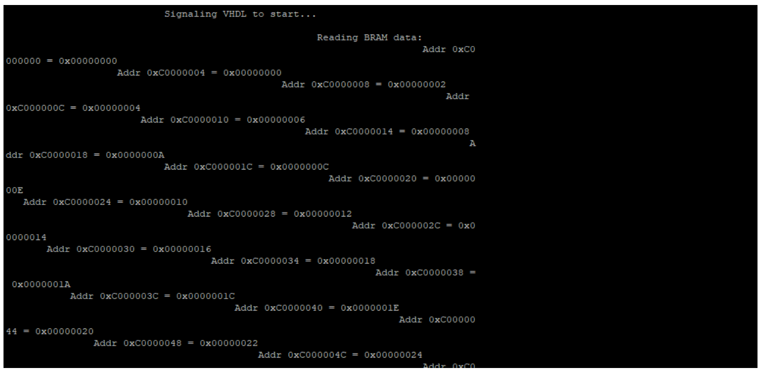
Reset captured on ILA and all data reset and started from beginning. Useful for captures.



```
ILA Status:  Waiting For Trigger (2048 out of 4096 samples)

Name                                          Value
design_1_i/loader_0_bram_we[3:0]              0
design_1_i/axi_bram_ctrl_0_bram_douta[31:0]   0000003f
design_1_i/custom_adder_0_done_add[3:0]       f
design_1_i/custom_adder_0_bram_din[31:0]      00000000
design_1_i/custom_adder_0_bram_en             0
design_1_i/axi_bram_ctrl_0_bram_doutb[31:0]   00000000
design_1_i/custom_adder_0_bram_addr[31:0]     00000100
design_1_i/custom_adder_0_bram_we[3:0]        0

Updated at: 2025-May-04 20:46:31
```

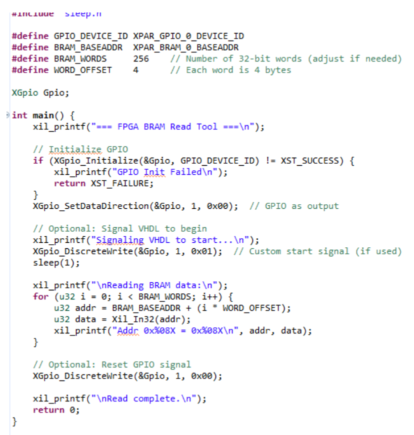
Above on ILA. Port B isn't being read as it should for BRAM. We can't read the info for some reason.

2. Able to read my AXI BRAM Data in Vitis (for Pmod if wanted to use). Basic UART showing incremented address and array of increasing numbers.



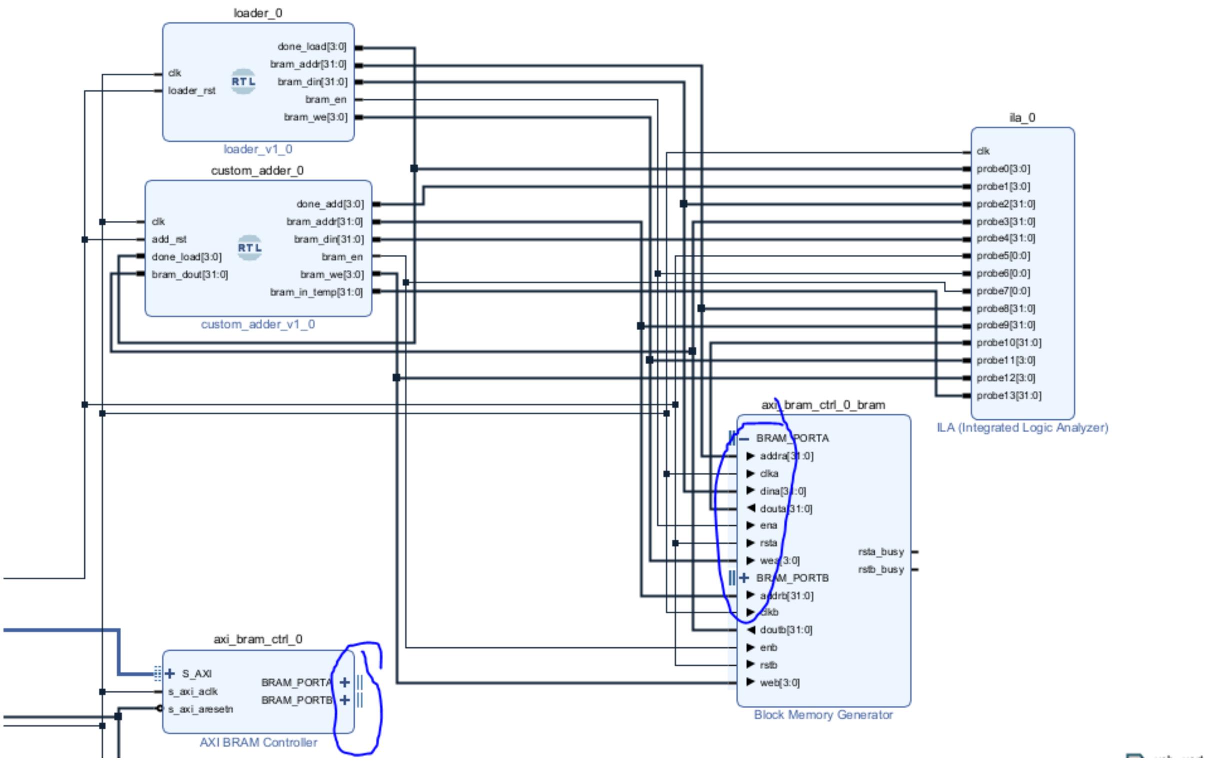
```
                           Signaling VHDL to start...

                                                          Reading BRAM data:
                                                                            Addr 0xC0
000000 = 0x00000000
                   Addr 0xC0000004 = 0x00000000
                                               Addr 0xC0000008 = 0x00000002
                                                                           Addr
0xC000000C = 0x00000004
                       Addr 0xC0000010 = 0x00000006
                                                   Addr 0xC0000014 = 0x00000008
                                                                               A
ddr 0xC0000018 = 0x0000000A
                           Addr 0xC000001C = 0x0000000C
                                                       Addr 0xC0000020 = 0x00000
00E
   Addr 0xC0000024 = 0x00000010
                               Addr 0xC0000028 = 0x00000012
                                                           Addr 0xC000002C = 0x0
0000014
       Addr 0xC0000030 = 0x00000016
                                   Addr 0xC0000034 = 0x00000018
                                                               Addr 0xC0000038 =
 0x0000001A
           Addr 0xC000003C = 0x0000001C
                                       Addr 0xC0000040 = 0x0000001E
                                                                   Addr 0xC00000
44 = 0x00000020
               Addr 0xC0000048 = 0x00000022
                                           Addr 0xC000004C = 0x00000024
                                                                       Addr 0xC0
```

Output from Putty.

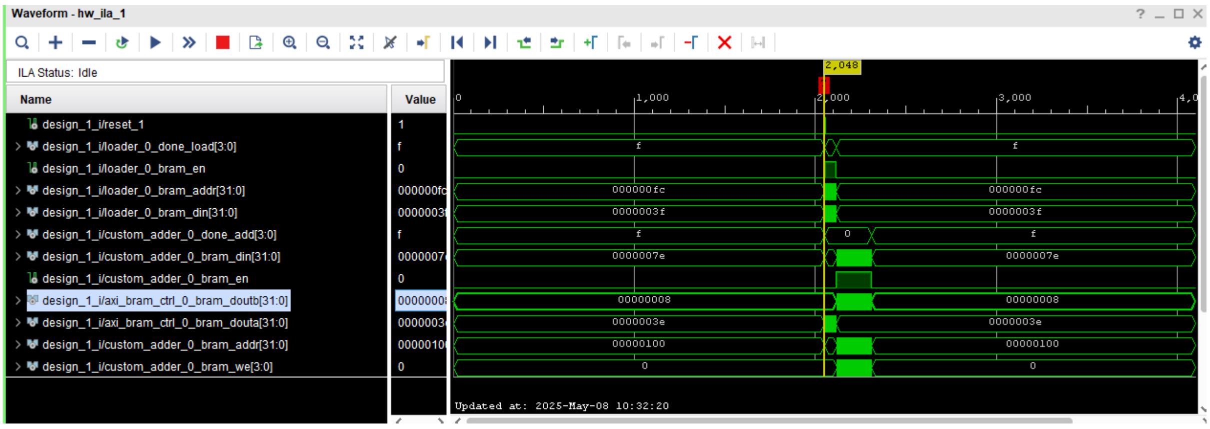


```c
#include "sleep.h"

#define GPIO_DEVICE_ID XPAR_GPIO_0_DEVICE_ID
#define BRAM_BASEADDR  XPAR_BRAM_0_BASEADDR
#define BRAM_WORDS     256    // Number of 32-bit words (adjust if needed)
#define WORD_OFFSET    4      // Each word is 4 bytes

XGpio Gpio;

int main() {
    xil_printf("=== FPGA BRAM Read Tool ===\n");

    // Initialize GPIO
    if (XGpio_Initialize(&Gpio, GPIO_DEVICE_ID) != XST_SUCCESS) {
        xil_printf("GPIO Init Failed\n");
        return XST_FAILURE;
    }
    XGpio_SetDataDirection(&Gpio, 1, 0x00);  // GPIO as output

    // Optional: Signal VHDL to begin
    xil_printf("Signaling VHDL to start...\n");
    XGpio_DiscreteWrite(&Gpio, 1, 0x01);  // Custom start signal (if used)
    sleep(1);

    xil_printf("\nReading BRAM data:\n");
    for (u32 i = 0; i < BRAM_WORDS; i++) {
        u32 addr = BRAM_BASEADDR + (i * WORD_OFFSET);
        u32 data = Xil_In32(addr);
        xil_printf("Addr 0x%08X = 0x%08X\n", addr, data);
    }

    // Optional: Reset GPIO signal
    XGpio_DiscreteWrite(&Gpio, 1, 0x00);

    xil_printf("\nRead complete.\n");
    return 0;
}
```

Code in C to read data from AXI once loaded.



AXI Bram was connected to block memory generator. I didn't know the signals beneath were being overridden by AXI. PortA worked fine, but portB was causing issues. Disconnecting made it work. I realised its best to have one port for AXI, rest for regular BRAM access without AXI.

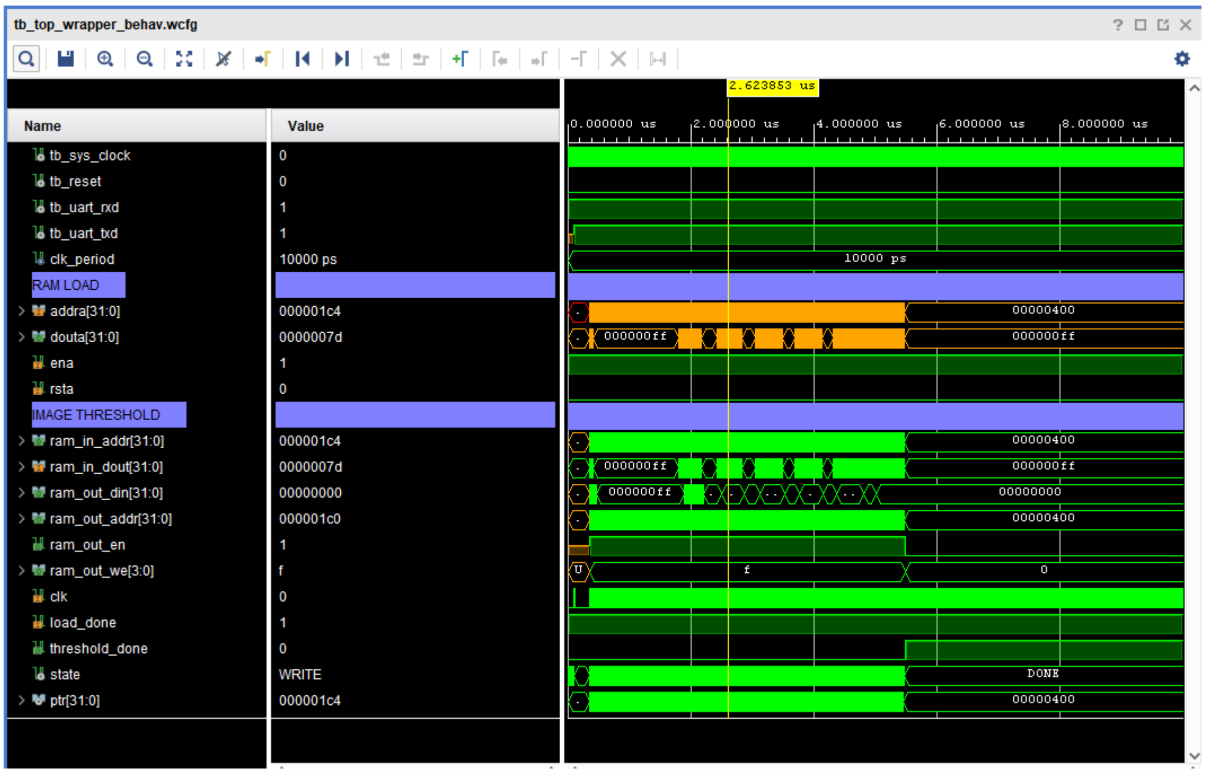

Waveform - hw_ila_1

ILA Status: Idle

Name	Value
design_1_i/reset_1	1
design_1_i/loader_0_done_load[3:0]	f
design_1_i/loader_0_bram_en	0
design_1_i/loader_0_bram_addr[31:0]	000000fc
design_1_i/loader_0_bram_din[31:0]	0000003
design_1_i/custom_adder_0_done_add[3:0]	f
design_1_i/custom_adder_0_bram_din[31:0]	0000007
design_1_i/custom_adder_0_bram_en	0
design_1_i/axi_bram_ctrl_0_bram_doutb[31:0]	0000000
design_1_i/axi_bram_ctrl_0_bram_douta[31:0]	0000003
design_1_i/custom_adder_0_bram_addr[31:0]	000010
design_1_i/custom_adder_0_bram_we[3:0]	0

```
Updated at: 2025-May-08 10:32:20
```

Data is now coming out of portb as expected, and my custom adder din (final result) has the data I expected.



tb_top_wrapper_behav.wcfg

Name	Value
tb_sys_clock	0
tb_reset	0
tb_uart_rxd	1
tb_uart_txd	1
clk_period	10000 ps
RAM LOAD	
addra[31:0]	000001c4
douta[31:0]	0000007d
ena	1
rsta	0
IMAGE THRESHOLD	
ram_in_addr[31:0]	000001c4
ram_in_dout[31:0]	0000007d
ram_out_din[31:0]	00000000
ram_out_addr[31:0]	000001c0
ram_out_en	1
ram_out_we[3:0]	f
clk	0
load_done	1
threshold_done	0
state	WRITE
ptr[31:0]	000001c4

RAM LOAD. And Image threshold.

How did I know it worked? In simulation, everything from RAM Load coming out had various values (showing it was a greyscale pixel values). Everything coming out from Image threshold was either 0 or ..00FF, meaning it was either black or white, correct for a threshold response.