Hae Chan Park
June 11, 2025

COMMAND PROCESSOR MODULE

I. PURPOSE

The objective of this lab project is to design and implement a simple command processor, also known as the CPM. The Command Processor Module (CPM) is an amalgamation of various topics that was covered in the 194BB course, hosting a slew of peripherals, requiring FSM design, and studying results through ILA's (Integrated Logic Analyzer). To describe the architecture specifically, the CPM would take commands from the user using a JTAG AXI master interface and would then use those commands and data to compute arithmetic or logical operations. The data flow is centralized around the block RAM (BRAM), which is the main memory of the program. The project results should demonstrate proper data through TCL commands that we will use.

II. METHODOLOGY

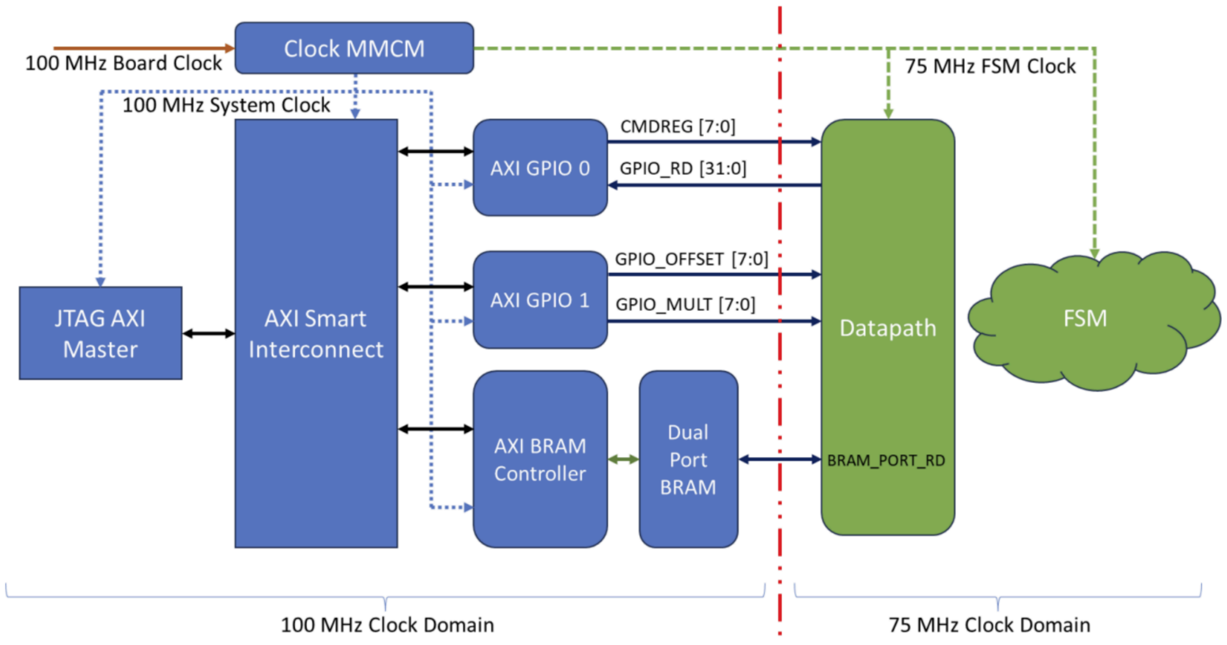
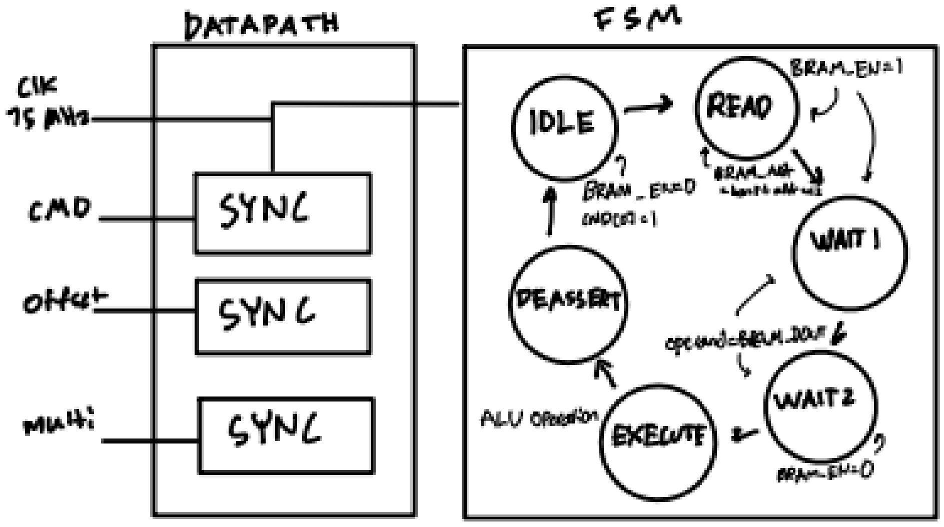


Figure 1: CPM Architecture

The design of the finite state machine (FSM) must be precise. The processor must somehow incorporate a 75 MHz and 100 MHz clock together, which can be tricky. In the final result of the project, this was resolved by using a synchronization block of Verilog in the FSM that would 'synchronize' the two clocks by using 2 flip-flops and various placeholder registers. Doing this would match their domains successfully. Another aspect of the design was the FSM itself, which consisted of various states: IDLE, READ, WAIT (1-2), EXECUTE, and DEASSERT, all of which are needed in carrying out the operations. The operations were handled with basic hard-coded expressions that would execute the right operation by utilizing the most significant three bits of the command register input.

III. RESULTS

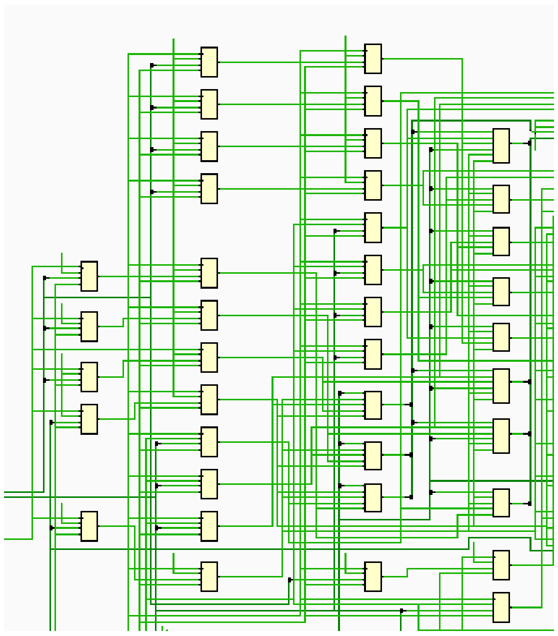
A. Data Path and FSM



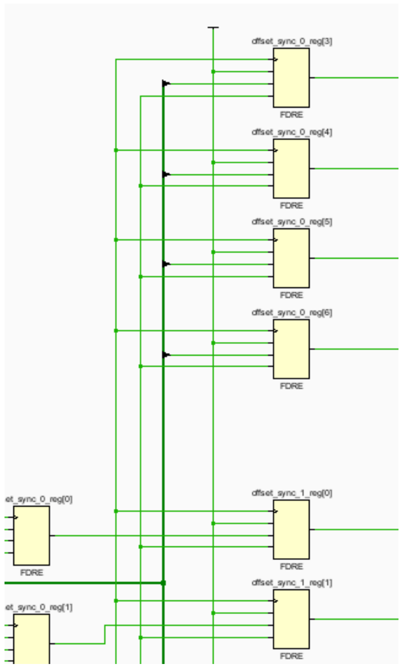
The FSM and datapath are essential to the operation of the CPM. The synchronization blocks in the datapath are constructed with dual Flip Flops to synchronize the two clock signals to the right domain. The notable signals are CMD, offset, and multi, all of which are from the 100 MHz domain. Registers were used to store intermittent values for the synchronization. The FSM shown to the right shows the design implementation of the project.

B. Clock Domain Crossing

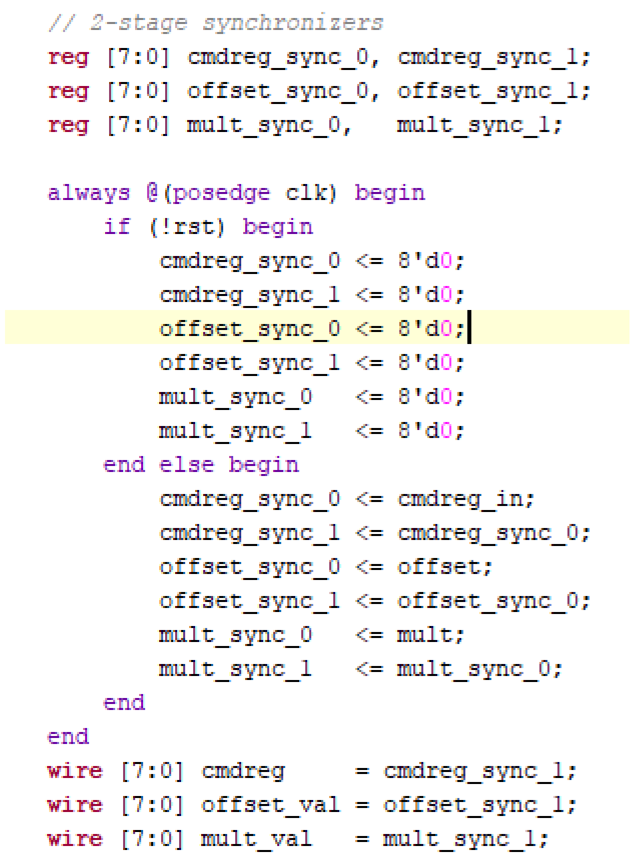
The CDC was resolved with synchronizers, specifically a dual flip flop sync. Any signal that needed to cross the clock domain must go through the synchronizers to ensure stability of the design. This proved successful, the project was able to produce results despite this challenge with this resolution. Shown below are the RTL schematics.



Zoomed out view



Zoomed in view on some of the sync registers.

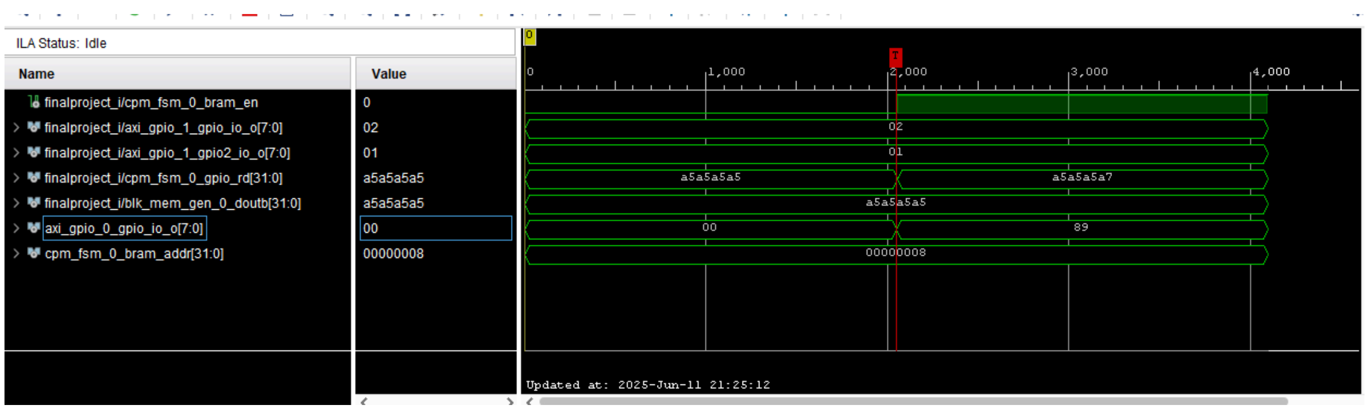


```verilog
// 2-stage synchronizers
reg [7:0] cmdreg_sync_0, cmdreg_sync_1;
reg [7:0] offset_sync_0, offset_sync_1;
reg [7:0] mult_sync_0,   mult_sync_1;

always @(posedge clk) begin
    if (!rst) begin
        cmdreg_sync_0 <= 8'd0;
        cmdreg_sync_1 <= 8'd0;
        offset_sync_0 <= 8'd0;
        offset_sync_1 <= 8'd0;
        mult_sync_0   <= 8'd0;
        mult_sync_1   <= 8'd0;
    end else begin
        cmdreg_sync_0 <= cmdreg_in;
        cmdreg_sync_1 <= cmdreg_sync_0;
        offset_sync_0 <= offset;
        offset_sync_1 <= offset_sync_0;
        mult_sync_0   <= mult;
        mult_sync_1   <= mult_sync_0;
    end
end
wire [7:0] cmdreg     = cmdreg_sync_1;
wire [7:0] offset_val = offset_sync_1;
wire [7:0] mult_val   = mult_sync_1;
```

Verilog HDL

C. ILA Capture of 'ADD' Command



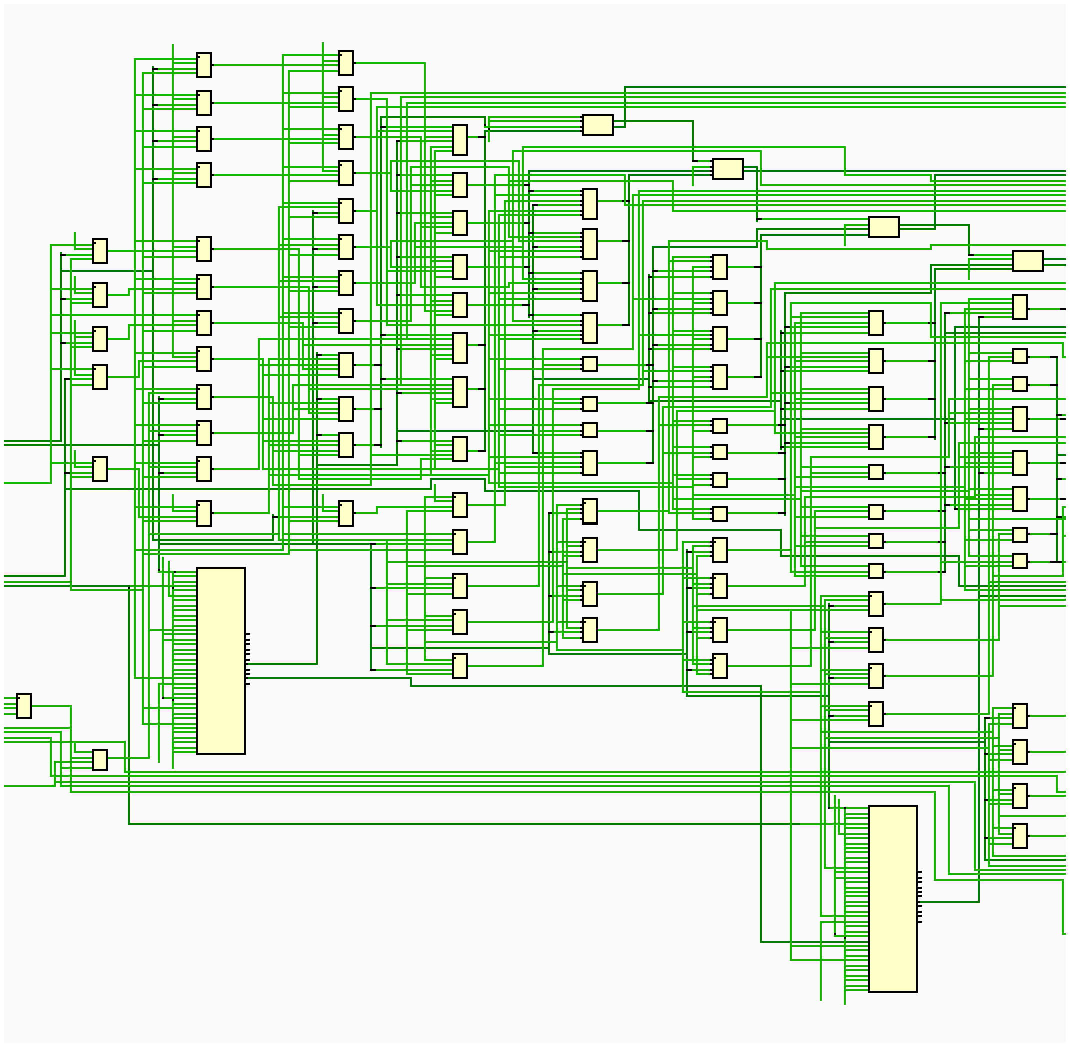
ILA Status: Idle

Name	Value
finalproject_i/cpm_fsm_0_bram_en	0
finalproject_i/axi_gpio_1_gpio_io_o[7:0]	02
finalproject_i/axi_gpio_1_gpio2_io_o[7:0]	01
finalproject_i/cpm_fsm_0_gpio_rd[31:0]	a5a5a5a5
finalproject_i/blk_mem_gen_0_doutb[31:0]	a5a5a5a5
axi_gpio_0_gpio_io_o[7:0]	00
cpm_fsm_0_bram_addr[31:0]	00000008

```
Updated at: 2025-Jun-11 21:25:12
```

Above is the ILA for the addition of two to the hexadecimal value listed (a5a5a5a5). The bottom signal is the command register, the enable is the top signal, the second to the top signal is the added word (offset), and the middle signal is the GPIO_RD. The signal successfully produces an a5a5a5a7 signal, which is expected.

D. RTL Schematic of Arithmetic Block

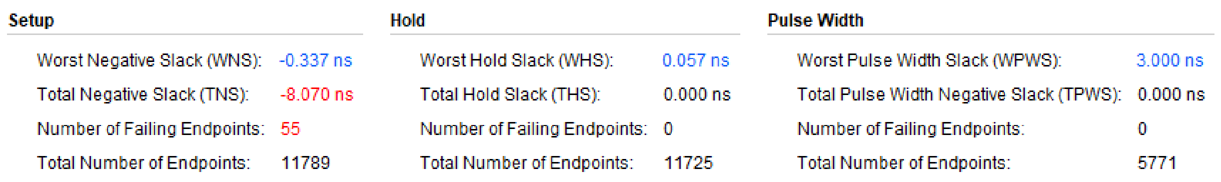


Zoomed out view of Arithmetic Blocks

There are two notable blocks used for the multiplication operation. This logically makes sense as well as the operand value is 32 bits and the mult_val is 8 bits. Thus it only needs two of these blocks.

E. Static Timing Report

The following is the static timing report. We notably had some total negative slack, which is odd, however, it makes sense logically as two separate clocks were employed through the clock MMCM.

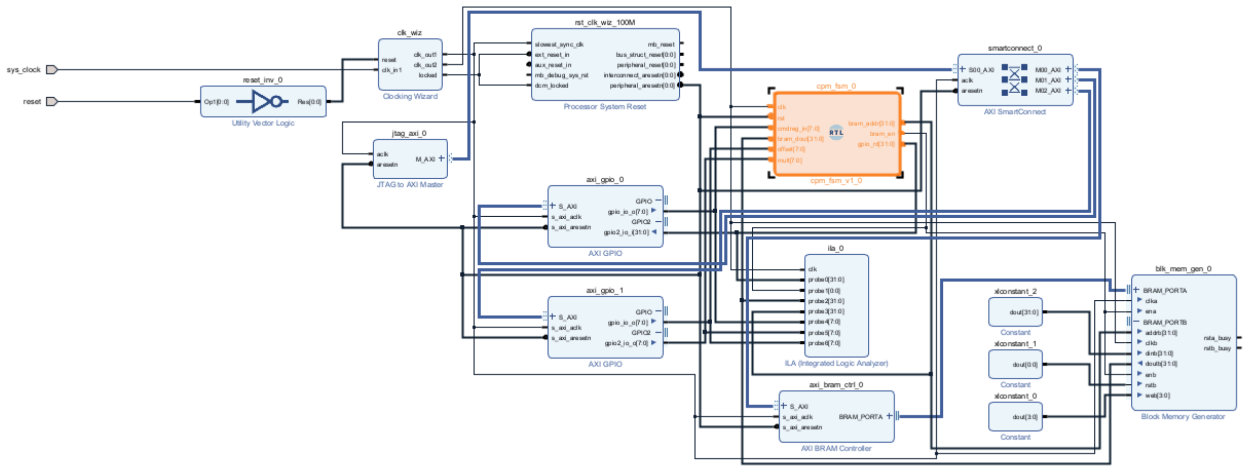


Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-0.337 ns	Worst Hold Slack (WHS):	0.057 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	-8.070 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	55	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	11789	Total Number of Endpoints:	11725	Total Number of Endpoints:	5771

Timing Report

IV. OVERVIEW

Overall, the lab was generally successful. The final design managed to produce some results after two cycles of entering the same commands over. This is likely attributed to the FSM not properly waiting for the CMD bit to go low, which can cause overlapping data between different commands. As such, errors occurred that involved previous data that was not to be read or used for that specific command. However, the FSM ultimately would produce the right results. The problem is mitigated when commands are conducted on the same BRAM address. Despite this design flaw, the FSM performed as expected when working properly and the final implementation fulfilled most of the requirements.


sys_clock
reset
clk_wiz
Clocking Wizard
reset_inv_0
Utility Vector Logic
rst_clk_wiz_100M
Processor System Reset
jtag_axi_0
JTAG to AXI Master
smartconnect_0
AXI SmartConnect
cpm_fsm_0
cpm_fsm_v1_0
axi_gpio_0
AXI GPIO
axi_gpio_1
AXI GPIO
ila_0
ILA (Integrated Logic Analyzer)
xlconstant_2
xlconstant_1
xlconstant_0
Constant
blk_mem_gen_0
Block Memory Generator
axi_bram_ctrl_0
AXI BRAM Controller