# MP2 Report

By: Nomaan Dossaji (ndossa2)
And
Jay Patel (jdpatel4)

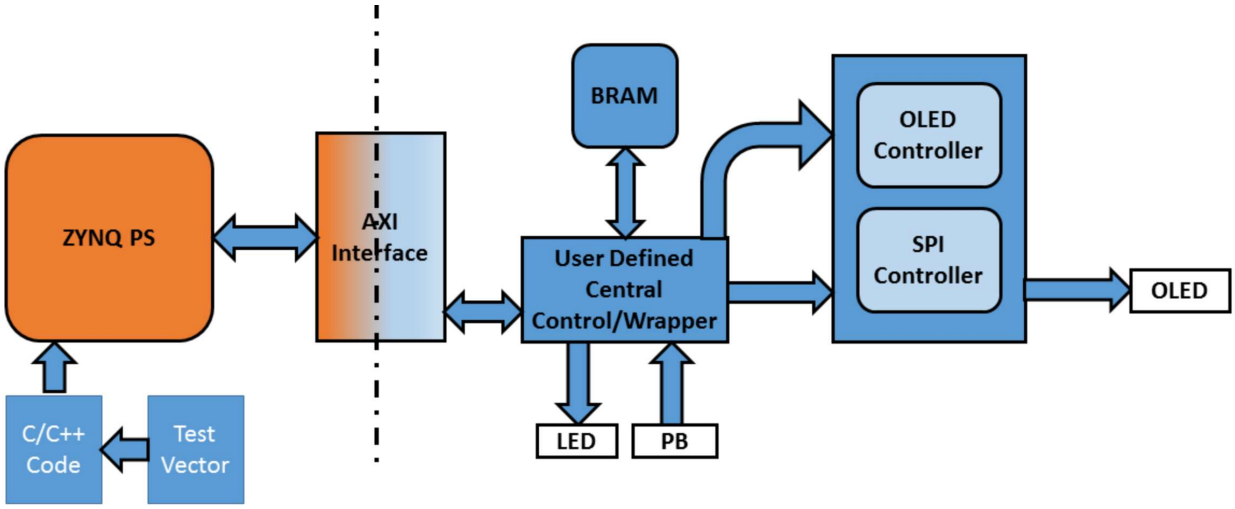
## Part 1

### Assumptions

We assume the test vector file would be the same format as the sample file. We also assume that we need to use up to 8 LEDs and not all LEDs need to be used for the status.

### Block Diagram

Our implementation was the exact same as the block diagram given, so we assumed it is safe to reuse this block diagram.




### Entities/Modules

PMOD OLED: Outputs data onto PMOD.

ZYNQ Processing System: The processor for the FPGA that contains the functionality.

Control Wrapper: Controls how the values in the display are changed.

BRAM Generator: Stores the value being displayed on the PMOD.

AXI Lite Interface: Controls the communication between the wrapper and the processor.

### Design Process

We did not use a state machine since we were able to access all 512 bits of data at the same time. The only state machine was given to us with the PMOD project. We are also using Digilents states to display the LEDs.

PS and PL communicate via AXI Lite. They interconnect transports the data between the two interfaces.

The central controller transfers the 512 bits from BRAM to OLED when you press reset through a simple connection bus.

Writing to the AXI Lite registers were implemented in software.

One test vector is sent during initialization and every new test vector is sent when the user presses the center button.

We chose the BRAM to be 512 bits wide to hold all the data that is being sent. The BRAM has a depth of 2 because that is the minimum.

We created an AXI peripheral that was our communication protocol between PS-PL-side.

We did not implement any control or status signals between the PS and PL besides the center button being reset. The signals that are used are from the AXI peripheral.

The status with the LEDs will display the states. When the board is initializing, the LEDs will show one LED. The second LED will light up once the board is ready for the user.

Latency Calculations

a. 512 bits/32 bits per cycle = 16 clock cycles to transfer the initialize vector. The initialize state machine shows 20 states. Assuming these are done in parallel, the design has a 20-cycle latency to initialize.
b. The time to display all 64 characters on the OLED after the user presses the button is described like this. 64 cycles for the alphabet screen. 64 cycles for the clear screen. 64 cycles for the Digilent screen. There are also an extra 3 cycles for wait states. There is also another cycle to read from BRAM and send it to PMOD. 64*3 + 4 = 196 cycle latency.

Post Implementation Results

| Resource | Utilization |
|---|---|
| LUT | 1555 |
| LUT RAM | 62 |
| FF | 2699 |
| IO | 14 |
| BRAM | 8 |

| | **Power** |
|---|---|
| Clocks | .005W |
| Signals | .005W |
| Logic | .004W |
| BRAM | .002W |
| PS7 | 1.529W |
| IO | .003W |

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| WNS | 3.15ns | WHS | 0.45ns | WPWS | 4.02ns |
| TNS | 0 ns | THS | 0 ns | TPWS | 0 |
| Failing Endpoints | 0 | Failing Endpoints | 0 | Failing Endpoints | 0 |

| | | | | | |
|---|---|---|---|---|---|
| Number of Endpoints | 3622 | Number of Endpoints | 3622 | Number of Endpoints | 1543 |

Difficulties/Bugs

A difficulty we encountered was creating the AXI interface. We did not think there was enough documentation to know what needed to be done so we were stuck on this part for majority of the time.

What We Learned

We learned to be able to use the AXI interface to communicate from the PS to the PL. We also learned to use the BRAM to hold the data we needed to transfer to the PMOD OLED.

Environment

We used our personal Windows computers with Vivado 2018.2.

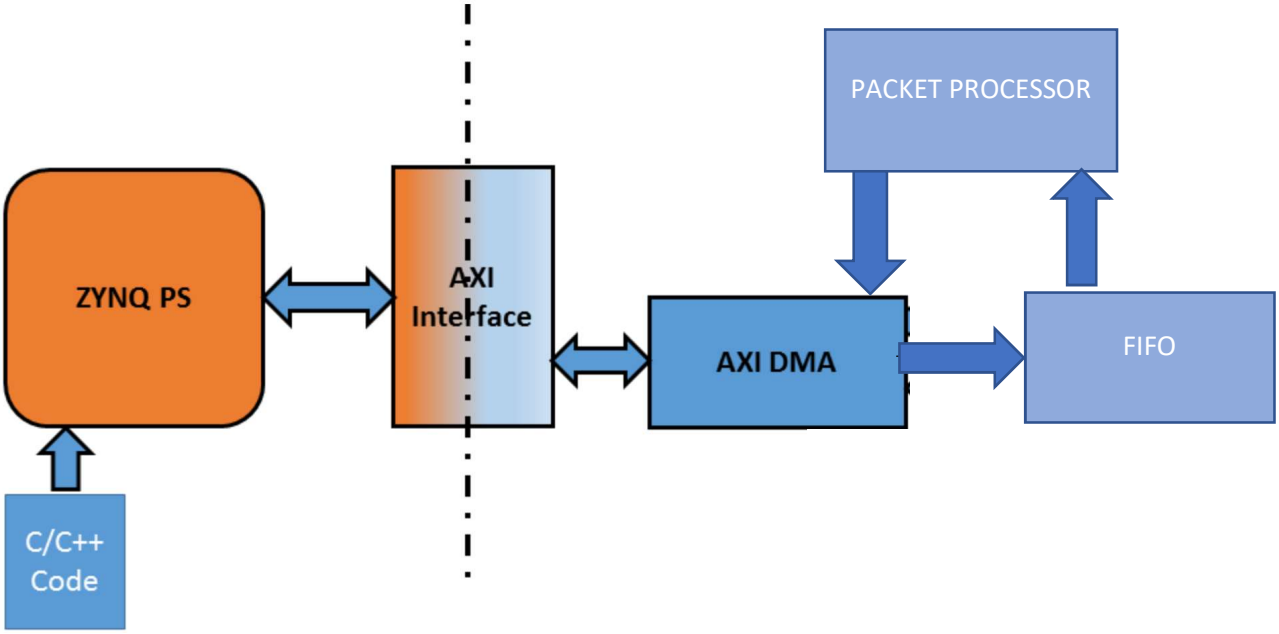
## Part 2

### Assumptions

We need to add 10 to the entire packet that is received by the FIFO. We also assume that the DMA is located within the PL sector.

Block Diagram

We used part of the diagram given by the instructors. We used the part that was still applicable to us and added additional blocks to indicate what was missing from the original diagram. The packet processing unit needed to include the FIFO.




Entities/Modules

ZYNQ Processing System: The processor for the FPGA that contains the functionality.

DMA: Direct Memory Access stored the data that we were to send to the FIFO. It is also the location where we would write the processed packets to.

FIFO: A queue that holds the data sent from the DMA

Packet Processer: Increments the data sent from the FIFO by 10.

AXI Lite Interface: Controls the communication between the DMA and the processor.

Design Process

For this part, the processor gives instructions to the DMA to send packets to the FIFO. The FIFO sends these packets to the packet processor, which adds 10 to each packet. After this, the packets are sent back to the DMA to be written in its memory. In the end, the processor verifies to make sure the DMA received the correct values.

We did not need to implement any state machine states for this part.

PS and PL communicates via the AXI interconnect. The DMA is located within the PL and its communication with the FIFO is via AXI Stream.

Controlling the DMA and verifying the DMA has the correct values is controlled through software.

The only control signals between the PS and PL is when the DMA sends interrupts to the processor.

Post Implementation Results

| Resource | Utilization |
|---|---|
| LUT | 5486 |
| LUT RAM | 838 |
| FF | 8024 |
| IO | 138 |
| BRAM | 3.5 |

| | **Power** |
|---|---|
| Clocks | .026W |
| Signals | .011W |
| Logic | .01W |
| BRAM | .001W |
| PS7 | 1.533W |

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| WNS | 2.978ns | WHS | .02ns | WPWS | 3.75ns |
| TNS | 0 ns | THS | 0 ns | TPWS | 0 |
| Failing Endpoints | 0 | Failing Endpoints | 0 | Failing Endpoints | 0 |
| Number of Endpoints | 28021 | Number of Endpoints | 28021 | Number of Endpoints | 9576 |

Difficulties/Bugs

A major difficulty we encountered was not understanding the documentation. We assumed that we needed to create a peripheral to interface the DMA and the packet processing, but we were wrong. We simply needed to connect the packet processing in between the FIFO and DMA.

What We Learned

Through this part of the MP, we learned to communicate with the DMA. We were able to communicate to the DMA from the PS as well as access the DMA from programmable logic.

Environment

We used our personal Windows computers with Vivado 2018.2.