## General System Architecture

- Both myip_v1_0.vhd and matrix_multiply_vhd are mealy FSMs.
  - Figure 2 and 3 shows “Moore-like” architecture. However, some values within each state depends on external values input values which makes it “Mealy”.

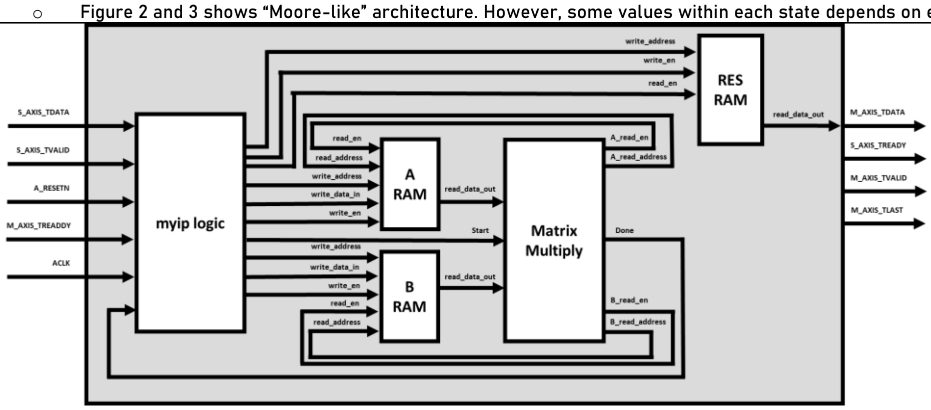


Figure 1: Generic System Architecture Derived From Synthesized Schematic

<u>Summarised Explanation of System Architecture:</u>

Our code utilises both sequential and combinational circuit, for a more detailed and elaborated look, please refer to the codes, comments are available for easier understanding.

<u>Myip states: `Idle`, `Read_Inputs`, `Compute`, `Write_Outputs`, `Write_Buffer` [Figure 2. Depicts the FSM flow when planning the processes]</u>

Idle:
- Registers are initialised as 0, when `S_AXIS_TVALID = '1'` state goes to Read_Inputs

**`Read_Inputs`**:
- myip reads data from the input memory file via S_AXIS_TDATA
- These data is then written in A and B rams for matrix A and B respectively when A/B write_en is asserted in order when writing to their respective rams, a read_counter is utilised for tracking and allocation of memory. Once read_counter indicated that all elements of matrix A and B have been read and written into their respective rams, the states goes to compute

**`Compute`**:
- `Start = '1'` is asserted matrix_multiply coprocessor will then be utilised, one matrix_multiply has completed its processes, Done = ‘1’ is asserted and state goes to Write_Output

**`Write_Outputs and Write_Buffer`**:
- Write_flag is utilised to toggle between Write_outputs and Write_buffer as 3 cycles is needed when delivering the outputs to result memory
- Address is first updated for `RES_read_address`, `M_AXIS_TDATA` will then be updated with values from `RES_read_data_out`
- Once the data is ready to be presented to result memory, `M_AXIS_TVALID` is then asserted to ‘1’ and thus updating the result memory
- write_counter is used to for address and progress tracking. Once write_counter tracks that all data has been present to the result memory, `M_AXIS_TLAST` is asserted to `'1'` and state goes back to Idle for reset.

<u>Matrix_multiply state: Idle, Buff, Add, Reset [Figure 3. Depicts the FSM flow when planning the processes]</u>

**`Idle`**:
- Registers initialised as 0, state goes to Buff

**`Buff`**:
- Buff here acts as a buffer for registers to be updated during the Counter increment and Product + Sum processes, state goes to Add

**`Add`**:
- Here (product) `P` is product of the desired value from matrix A and B, and it is Summed based on `Sum <= Sum + P`. Note: ever other state P is 0 and Sum = Sum
- Address counters for A and B are utilised to keep track and of the data being used and progress
- Once `B_address_counters` reaches the last address in B ram, state goes to reset

**`Reset`**:
- Here Sum resets to 0, `RES_address_counter` tracks whether all the Summed values for the output has been allocated to the desired address
- Once all the data has been sent to RES and the state goes back to Idle, else if not completed the cycle resets with `B_address_counter` being and state goes to Buff

## Finite State Machines

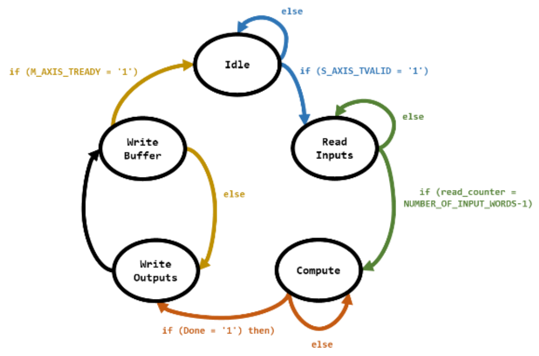
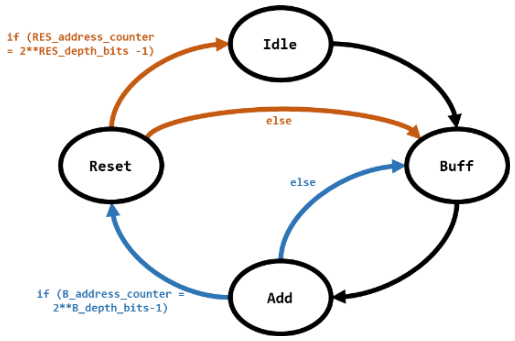


Figure 2: my_ip FSM

| Combinational Logic / signals outside of clock | |
|---|---|
| • `A_write_en` | • `RES_read_address` |
| • `B_write_en` | • `M_AXIS_TDATA(7 downto 0)` |
| • `RES_read_en` | • `M_AXIS_TLAST` |
| • `M_AXIS_TVALID` | • `B_write_address` |

Table 1i: Combinational Logic at my_ip


if (RES_address_counter = 2**RES_depth_bits -1)
Idle
else
Reset
Buff
else
if (B_address_counter = 2**B_depth_bits-1)
Add
Figure 3: matrix_multiply FSM

| Combinational Logic / signals outside of clock | |
|---|---|
| • Done | • B_read_en |
| • RES_write_en | • P |
| • A_read_en | • RES_write_data_in |

Table 2i: Combinational Logic at matrix_multiply

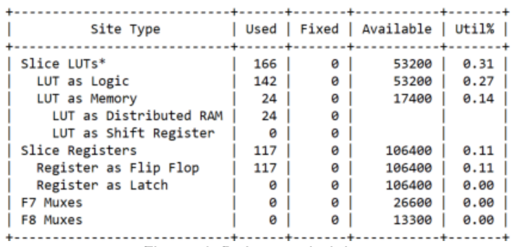
| Non Combinational Logic / Signals within clock | |
|---|---|
| • Start | • write_flag |
| • Done | • A_write_data_in |
| • read_counter | • B_write_data_in |
| • write_counter | • A_write_address |
| • S_AXIS_TREADY | |

Table 1ii: Non-Combinational Logic at my_ip

| Non Combinational Logic / Signals within clock | |
|---|---|
| • A_address_counter | • RES_address_counter |
| • B_address_counter | • sum |

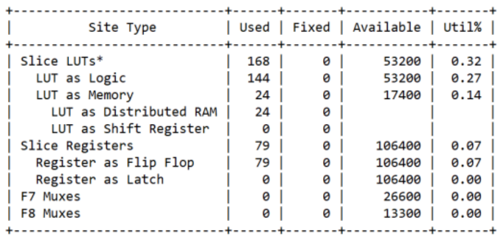
Table 2ii: Combinational Logic at matrix_multiply

## Resource Usage



```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |  166 |     0 |     53200 |  0.31 |
|   LUT as Logic          |  142 |     0 |     53200 |  0.27 |
|   LUT as Memory         |   24 |     0 |     17400 |  0.14 |
|     LUT as Distributed RAM |   24 |     0 |           |       |
|     LUT as Shift Register  |    0 |     0 |           |       |
| Slice Registers         |  117 |     0 |    106400 |  0.11 |
|   Register as Flip Flop |  117 |     0 |    106400 |  0.11 |
|   Register as Latch     |    0 |     0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |     13300 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Figure 4: Before optimising



```
+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs*                |  168 |     0 |     53200 |  0.32 |
|   LUT as Logic             |  144 |     0 |     53200 |  0.27 |
|   LUT as Memory            |   24 |     0 |     17400 |  0.14 |
|     LUT as Distributed RAM |   24 |     0 |           |       |
|     LUT as Shift Register  |    0 |     0 |           |       |
| Slice Registers            |   79 |     0 |    106400 |  0.07 |
|   Register as Flip Flop    |   79 |     0 |    106400 |  0.07 |
|   Register as Latch        |    0 |     0 |    106400 |  0.00 |
| F7 Muxes                   |    0 |     0 |     26600 |  0.00 |
| F8 Muxes                   |    0 |     0 |     13300 |  0.00 |
+----------------------------+------+-------+-----------+-------+
```

Figure 5: After optimising

Figure 4: Most of our values are within the clock process → not very efficient

Figure 5: We switch all the values we can to combinational to save as many registers as we can → gained 2 LUTs in exchange for a reduction of 38 registers.

- Our "Combinational logic" is mostly behavioural and without clock → we declare the values out at every state

## Post-Synthesis Simulation

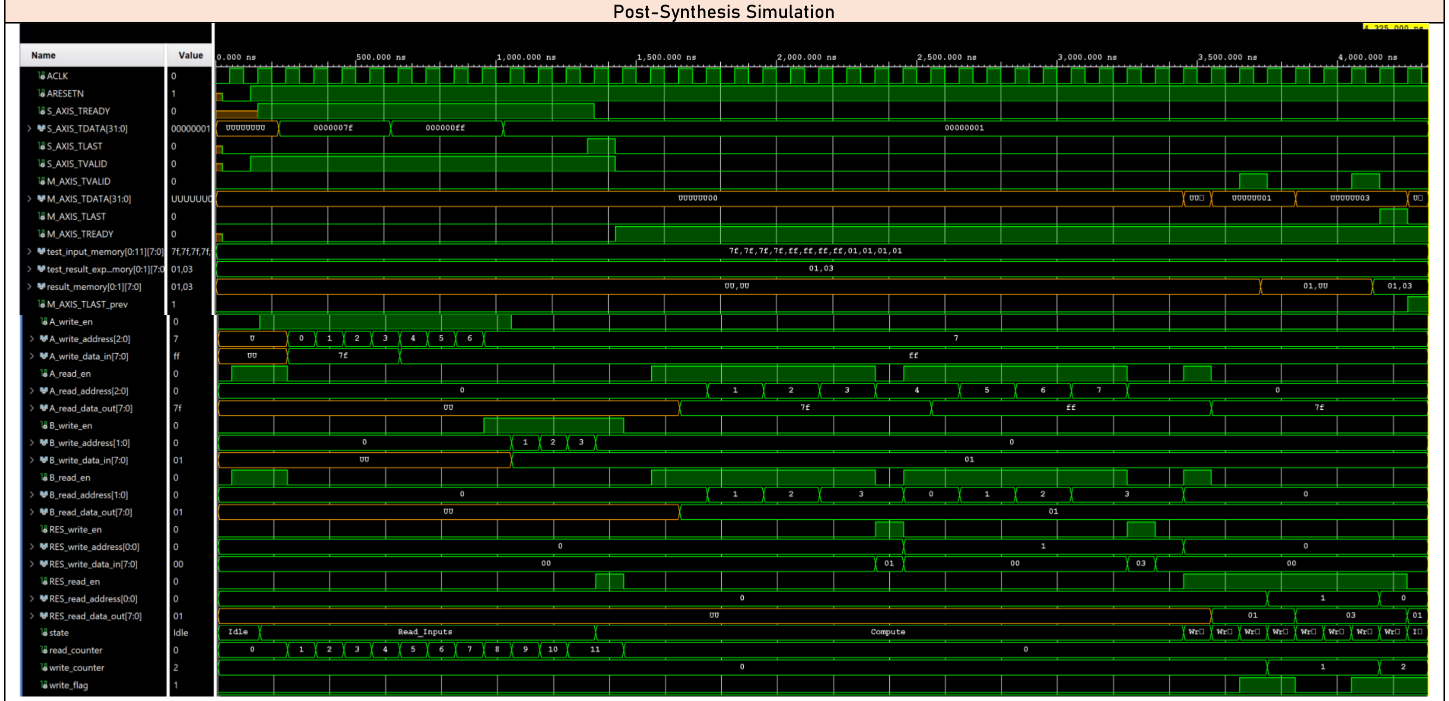


Figure 6: Post Synthesis Simulation (myip)

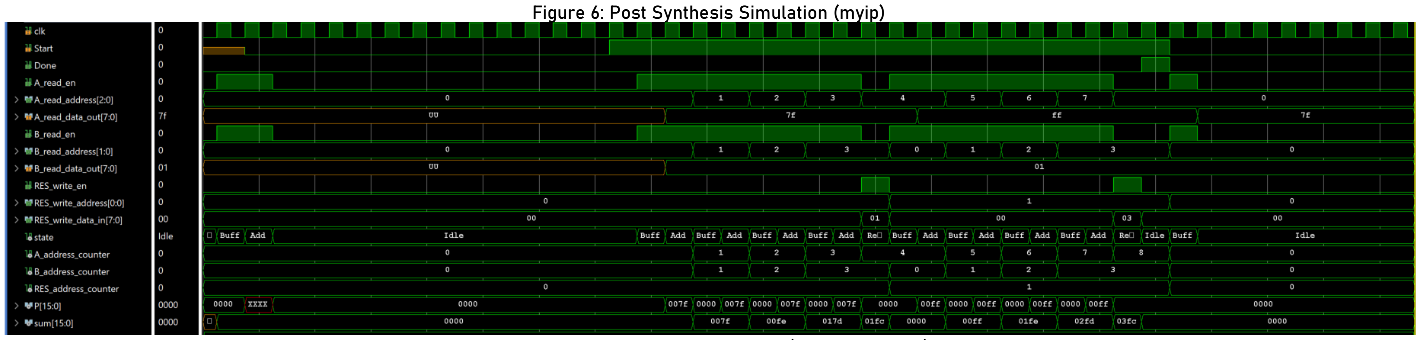


Figure 7: Post Synthesis Simulation (matrix_multiply)

Simulation @ myip:

- Idle: 1 clock cycle (cc), 100ns
- Read_Input: 13 cc, 1300ns
- Compute: 21 cc, 2100ns
- Write_Output+Write_Buffer: 8 cc, 800ns
- Total: 43 cycles, 4325ns (as per simulation result)