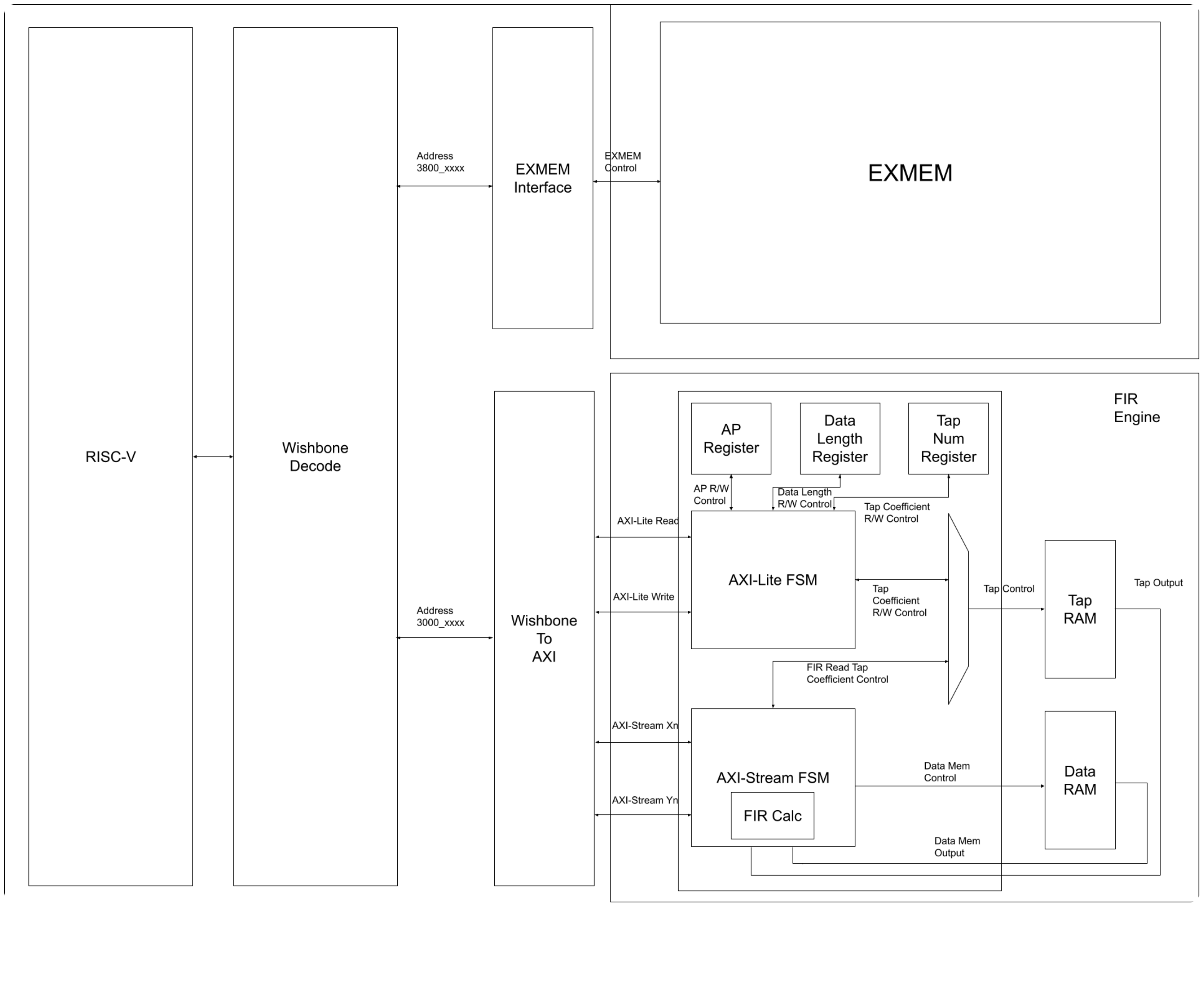
Lab 4-2 Caravel FIR

Student ID: 20726557

This lab involves integrating Lab 3-FIR and Lab 4-1 exmem-FIR into the Caravel user project area. A Wishbone to AXI interface is designed to communicate with FIR engine from Lab 3.

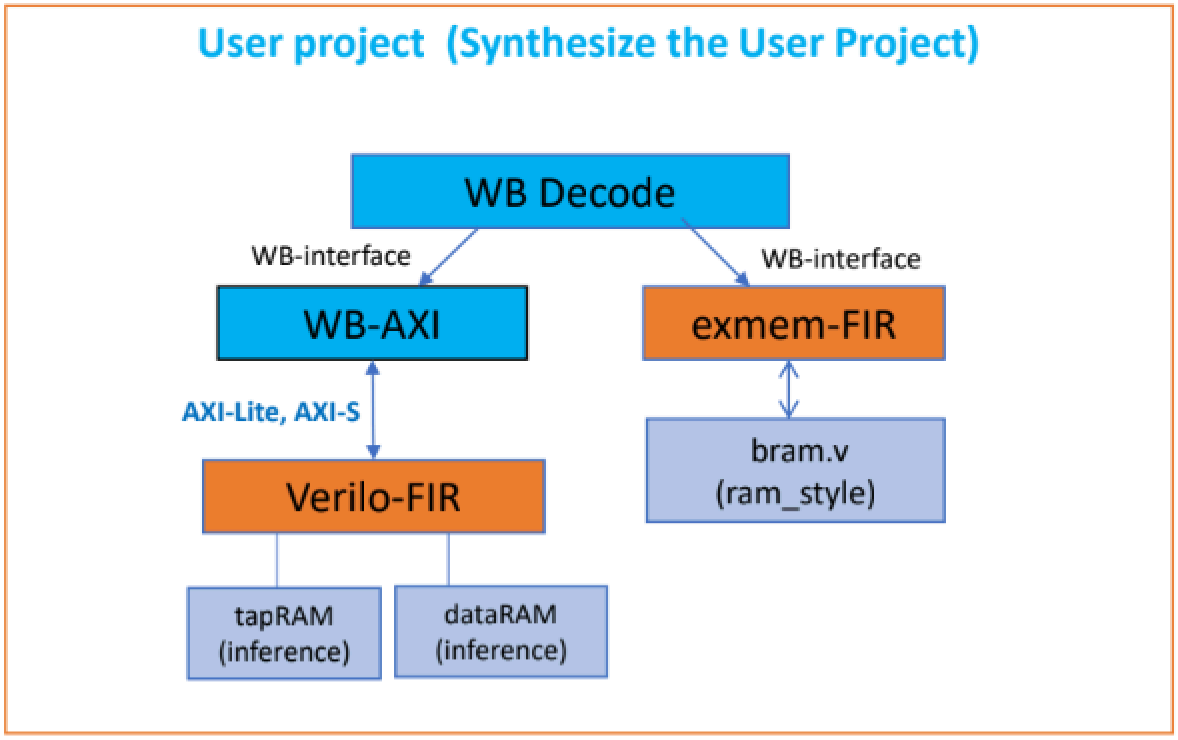
Design block diagram – datapath, control-path



The interface protocol between firmware, user project, and testbench

Firmware and User Project

The interface protocol between the firmware and the user project uses the Wishbone protocol to transmit data. The RISC-V CPU will execute the instructions in the firmware to transmit data through the Wishbone interface to the FIR engine and EXMEM. A Wishbone decoder will detect the access address of the transaction. The FIR engine is accessed by through the address of 0x3000_xxxx, while the EXMEM is accessed through the address of 0x3800_xxxx.



The firmware accesses the FIR engine in the User Project through a Wishbone-to-AXI conversion layer. The Wishbone-to-AXI layer detects the Wishbone request to access the FIR engine and converts the access into AXI-Lite or AXI-Stream format.

This lab uses the following addresses to access the FIR engine registers

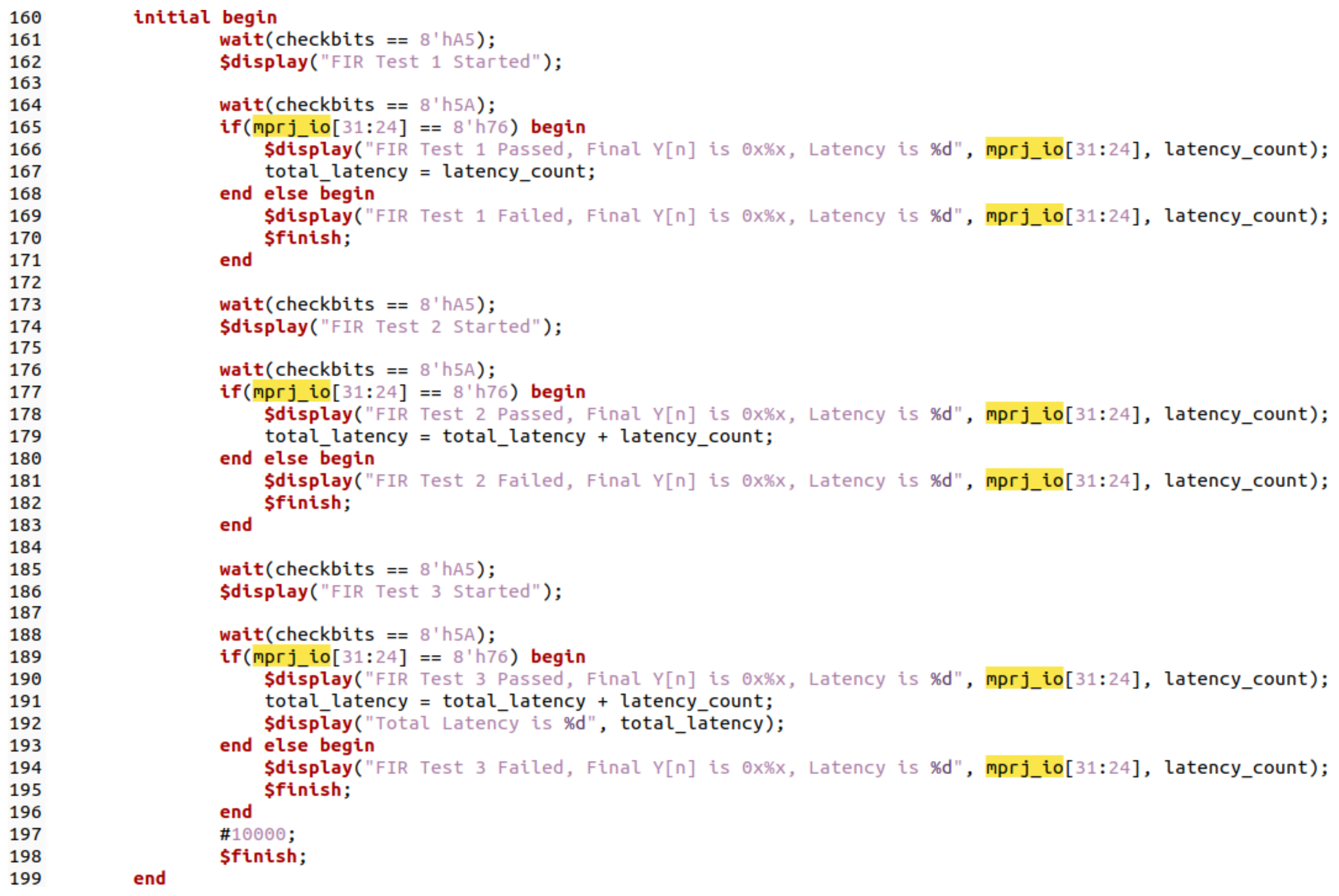
- 0x00: AP Configuration Register (AXI-Lite)
- 0x10-13: Data Length Register (AXI-Lite)
- 0x14-18: Number of Taps (AXI-Lite)
- 0x40-43: X[n] input (AXI-Stream)

- 0x44-47: Y[n] input (AXI-Stream)
- 0x80-100: Tap parameters

The access of EXMEM is done directly through the Wishbone decoder to store and load the firmware code of the FIR engine. The Wishbone signals are converted into the bram control signals. The Wishbone receives an acknowledgement after a certain period of delay set by a parameter.

Firmware and Testbench

The Firmware and Testbench communicates through the mprj_io signal. The testbench sets mprj_io[23:16] as the checkbits to determine the state of the project. When the firmware loads the FIR engine, the checkbits will be set to 8'A5 and to notify the testbench to start checking for the FIR output and calculate the latency. After the FIR process has completed, the firmware sets the CPU to assert 8'h5A to the checkbits. The testbench will check the final Y[n] output with the a reference output and record the latency. This is repeated 3 times to and the total latency is recorded.

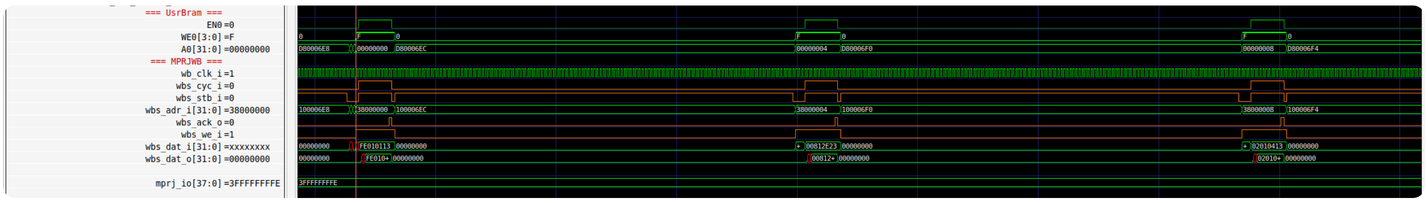


```
initial begin
        wait(checkbits == 8'hA5);
        $display("FIR Test 1 Started");

        wait(checkbits == 8'h5A);
        if(mprj_io[31:24] == 8'h76) begin
            $display("FIR Test 1 Passed, Final Y[n] is 0x%x, Latency is %d", mprj_io[31:24], latency_count);
            total_latency = latency_count;
        end else begin
            $display("FIR Test 1 Failed, Final Y[n] is 0x%x, Latency is %d", mprj_io[31:24], latency_count);
            $finish;
        end

        wait(checkbits == 8'hA5);
        $display("FIR Test 2 Started");

        wait(checkbits == 8'h5A);
        if(mprj_io[31:24] == 8'h76) begin
            $display("FIR Test 2 Passed, Final Y[n] is 0x%x, Latency is %d", mprj_io[31:24], latency_count);
            total_latency = total_latency + latency_count;
        end else begin
            $display("FIR Test 2 Failed, Final Y[n] is 0x%x, Latency is %d", mprj_io[31:24], latency_count);
            $finish;
        end

        wait(checkbits == 8'hA5);
        $display("FIR Test 3 Started");

        wait(checkbits == 8'h5A);
        if(mprj_io[31:24] == 8'h76) begin
            $display("FIR Test 3 Passed, Final Y[n] is 0x%x, Latency is %d", mprj_io[31:24], latency_count);
            total_latency = total_latency + latency_count;
            $display("Total Latency is %d", total_latency);
        end else begin
            $display("FIR Test 3 Failed, Final Y[n] is 0x%x, Latency is %d", mprj_io[31:24], latency_count);
            $finish;
        end
        #10000;
        $finish;
end
```

Waveform and analysis of the hardware/software behavior.

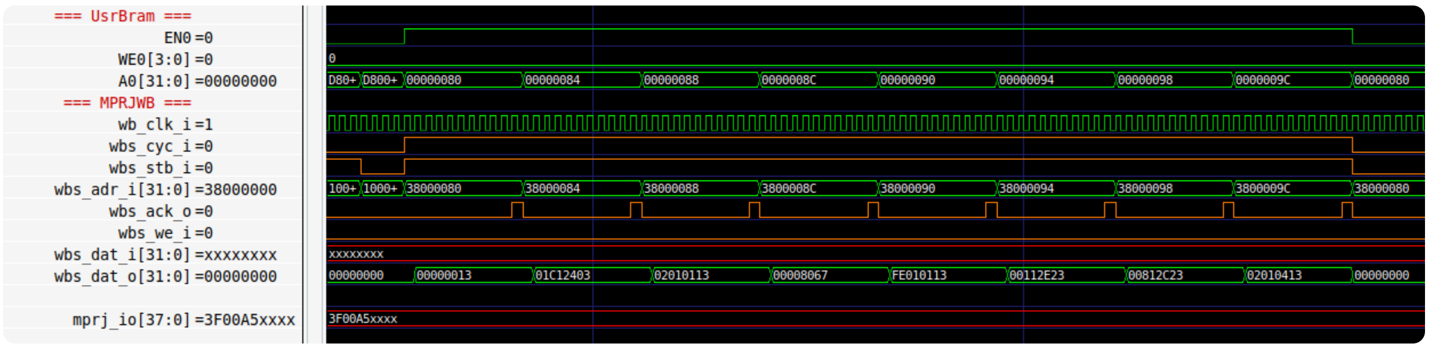
Software Behavior

Moving code from SPIFlash to EXMEM



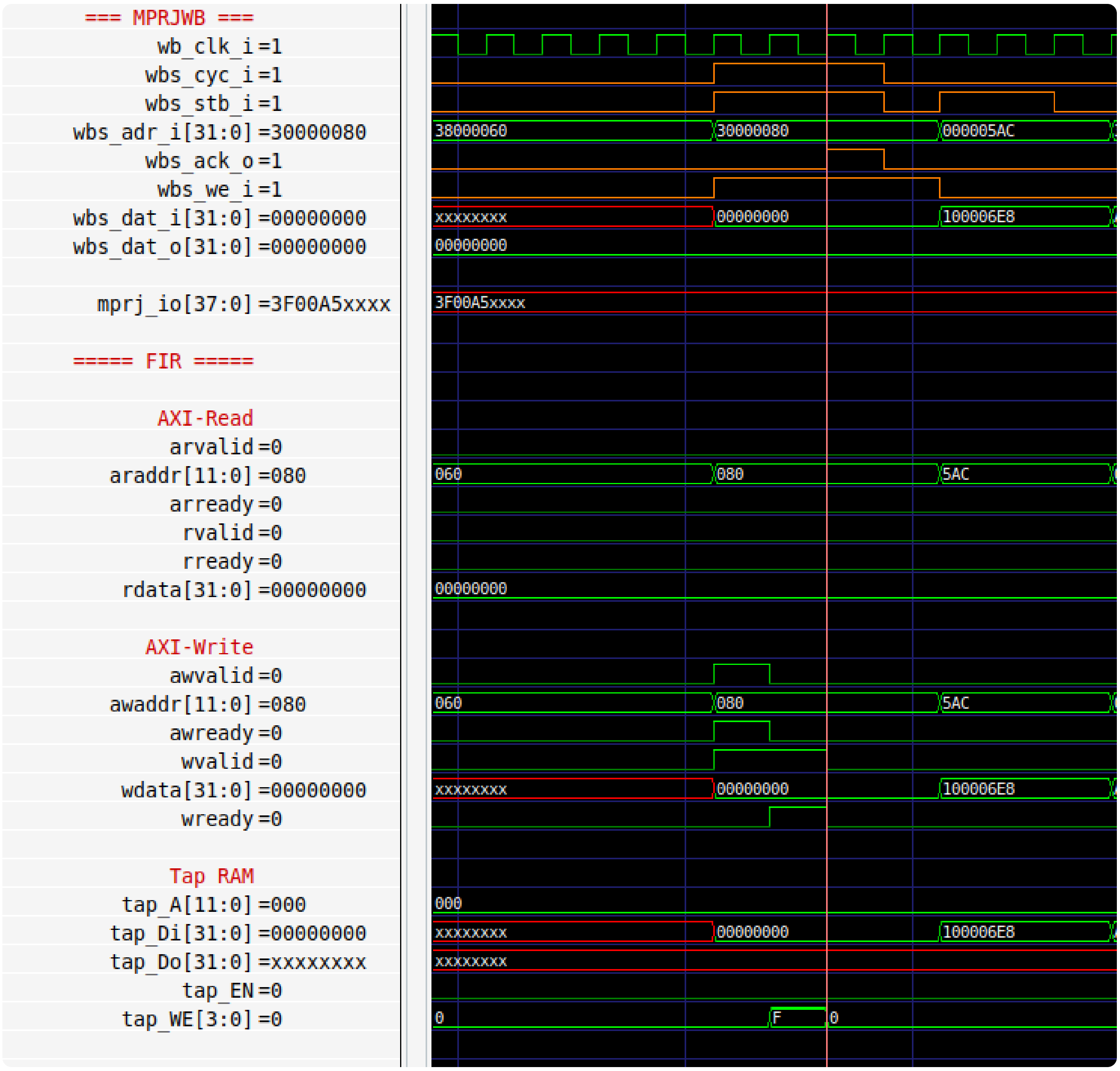
The firmware writes the code to access the FIR engine from SPIFlash to the EXMEM. This is done through the Wishbone interface. After a certain delay set by a parameter, an acknowledge is sent back through the Wishbone interface to complete the write operation.

Reading code from EXMEM



When the FIR engine is enabled, the code is retrieved from the EXMEM. This is also done through the Wishbone interface and also follows the same delay to return the acknowledgement when the read is completeted.

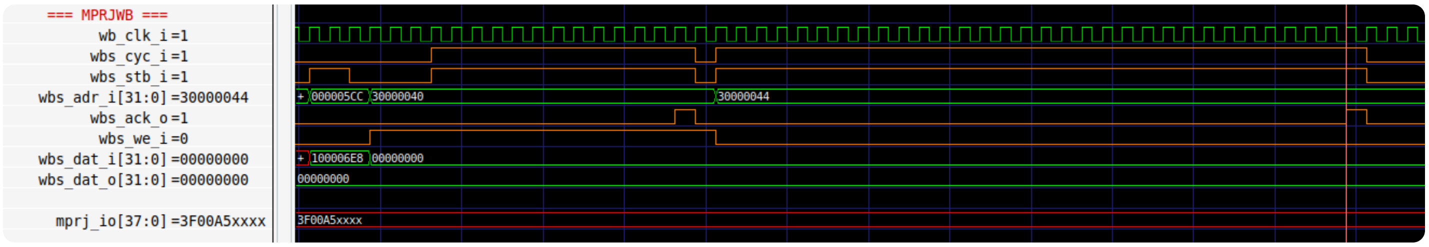
Initfir()



The FIR engine is initiated by passing the data length, number of taps, and the tap coefficients to the corresponding addresses. This is done in software by passing the data through using the Wishbone interface.

```c
void __attribute__ ( ( section ( ".mprjram" ) ) ) initfir() {
        //initial your fir
        reg_fir_data_length = data_length;
        reg_fir_tap_num = N;
        for (int i = 0; i < N; i++) {
            reg_fir_coeff(i) = taps[i];
        }
}
```

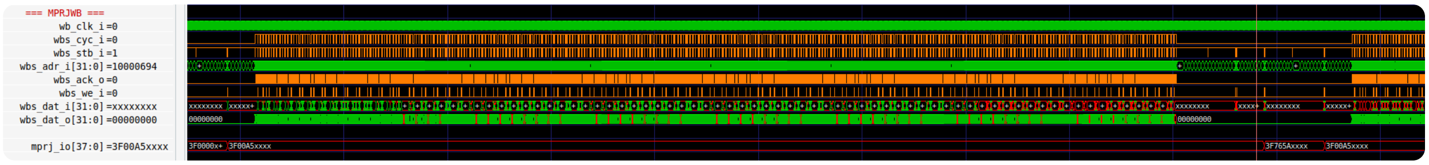
fir()



The software starts the FIR engine by setting AP configuration register to 0x01. The software transmits starts the X[n] and receives Y[n] through the Wishbone interface. The data length is 64 sets.

```
int* __attribute__ ( ( section ( ".mprjram" ) ) ) fir(){
        initfir();
        //write down your fir
        reg_fir_control = 0x00000001;
        for (int i = 0; i < data_length; i++) {
            reg_fir_x = i;
            outputsignal[i] = reg_fir_y;
        }
        return outputsignal;
}
```

Communication with Testbench

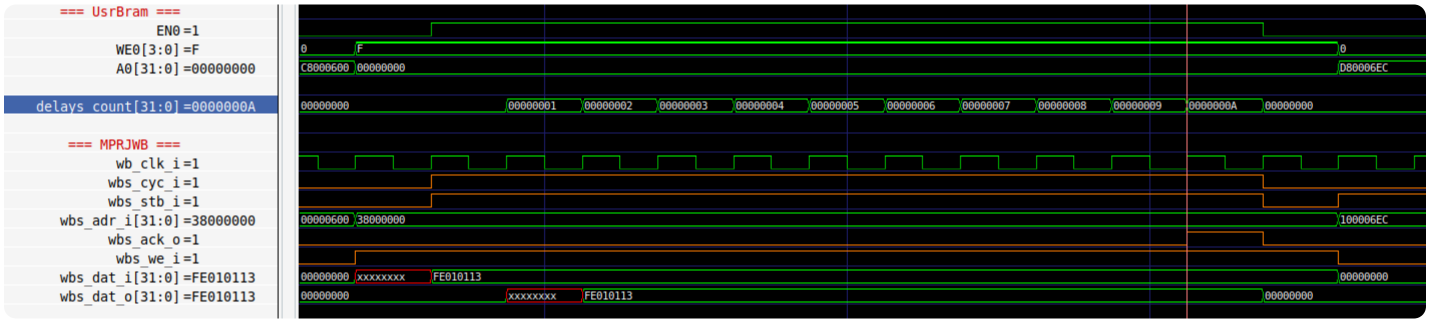


The software communicates with the verilog testbench through the mprj_io. When the checkbits is set to 8'hA5, the testbench will start the latency counter. When the FIR engine completes, the software sets the checkbits to 8'5A, allowing the testbench to determine the total latency.

```
        for (int i = 0; i < 3; i++) {
            // Flag start of the test
            reg_mprj_datal = 0x00A50000;

            int* tmp = fir();
            reg_mprj_datal = (*(tmp + 63) << 24) + 0x005A0000;

            //print("\n");
            //print("Monitor: Test 1 Passed\n\n");    // Makes simulation very long!
            //reg_mprj_datal = 0x005A0000;
        }
}
```

Hardware Behavior

Wishbone-to-EXMEM Delay Interface

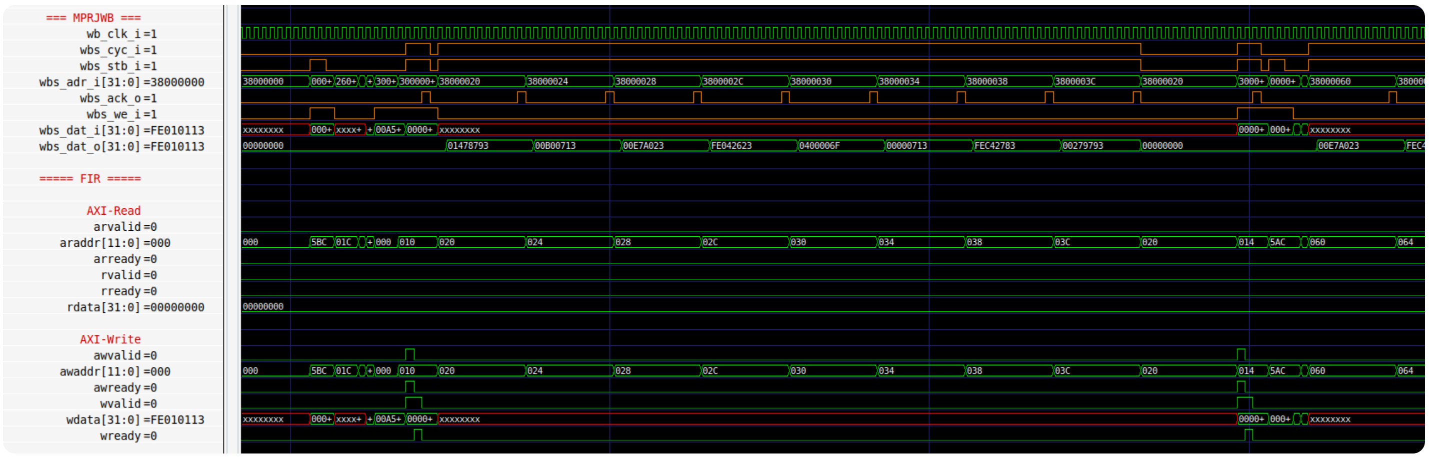


The Wishbone-to-EXMEM Delay Interface is in charge of the read/write operation between the Firmware and the EXMEM. The interface converts the Wishbone signals into BRAM control signals such that the data can be read/written. The wbs_ack_o ackwoledgement signal is asserted after a certain delay, determined using the DELAYS parameter, through the use of a delay counter.

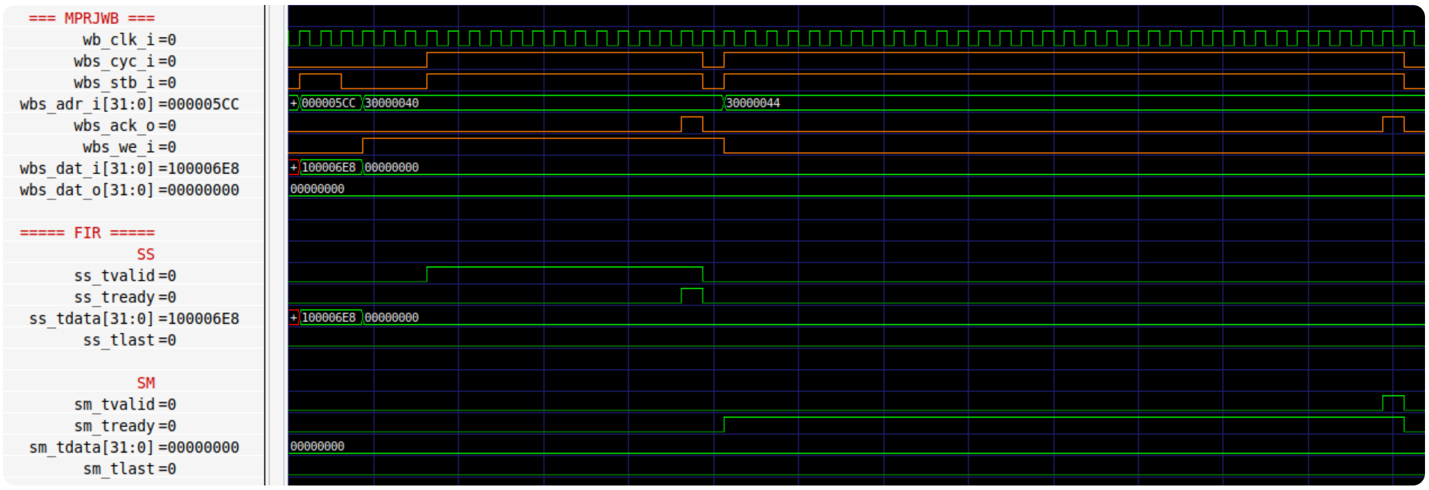
```
    // EXMEM
    assign exmem_en         = wbs_stb_i & wbs_cyc_i & (wbs_adr_i[31:24] == 8'h38);
    assign exmem_we[3:0]    = {4{wbs_we_i}} & wbs_sel_i[3:0]                      ;
    assign exmem_adr[31:0] = wbs_adr_i[31:0] - 32'h38000000                       ;

    always @(posedge wb_clk_i) begin
        if (wb_rst_i) begin
            delays_count[31:0] <= 32'b0;
        end else begin
            if (exmem_en) begin
                if (delays_count[31:0] != DELAYS) begin
                    delays_count[31:0] <= delays_count[31:0] + 32'b1;
                end else begin
                    delays_count[31:0] <= 32'b0;
                end
            end
        end
    end

    bram user_bram (
        .CLK(wb_clk_i         ),
        .WE0(exmem_we[3:0]    ),
        .EN0(exmem_en         ),
        .Di0(wbs_dat_i[31:0]  ),
        .Do0(exmem_dat_o[31:0]),
        .A0 (exmem_adr[31:0]  )
    );
```

Wishbone-to-AXI Interface

AXI-Lite



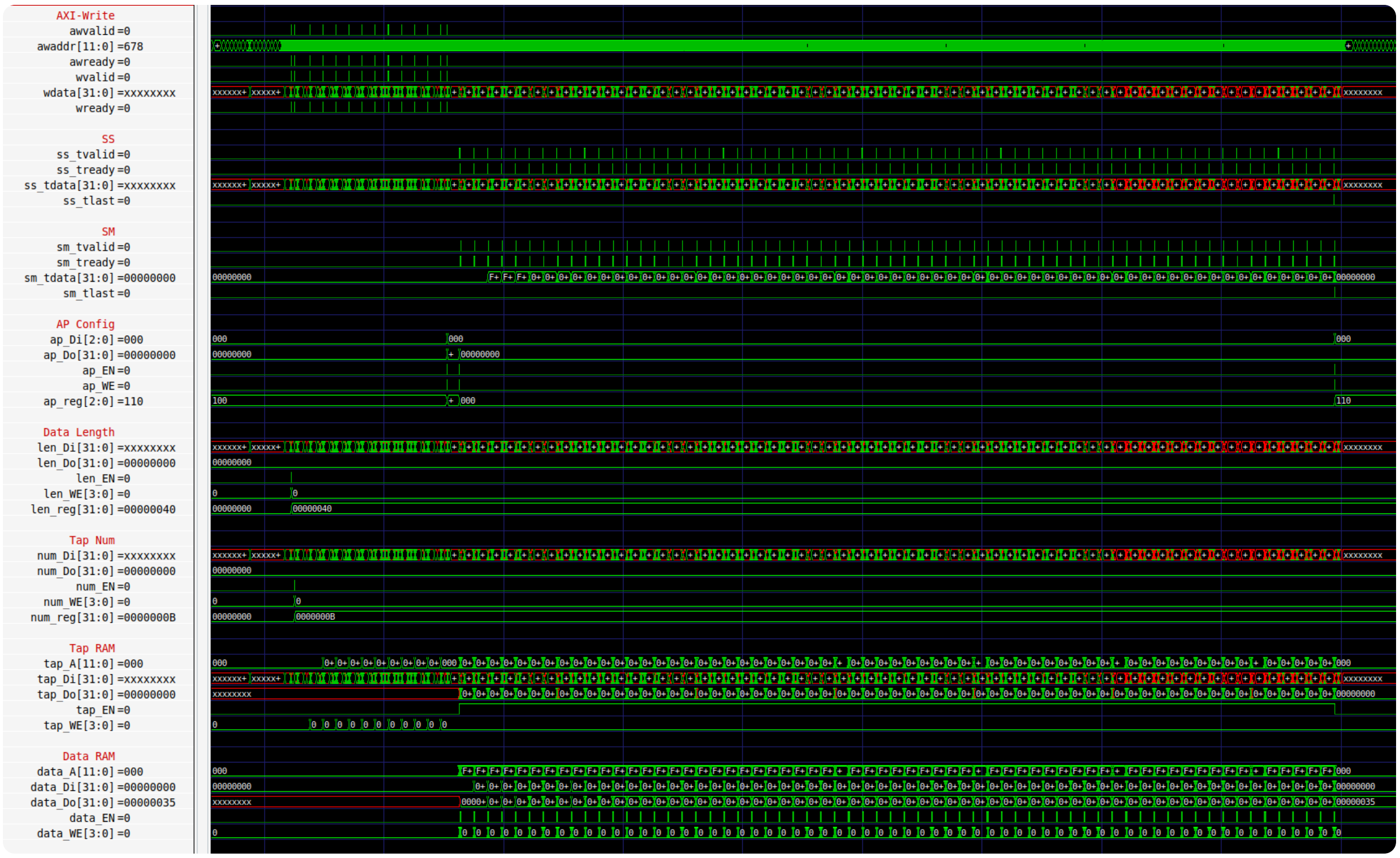
AXI-Stream



To communicate with the FIR engine, the Wishbone signals from the CPU is converted to the AXI protocol. To access the AP Configuration Register, Data Length, Tap Number registers, and Tap Coefficient RAM, the Wishbone signals are converted in AXI-Lite Protocol. The Read and Write transactions of the FIR engine is converted from Wishbone to AXI-Stream Protocol.

```verilog
assign fir_en = wbs_stb_i & wbs_cyc_i & (wbs_adr_i[31:24] == 8'h30);

// AXI-Lite
assign axilite_en        = fir_en & ~((wbs_adr_i[6:0] >= 7'h40) & (wbs_adr_i[6:0] <= 7'h47));
assign axilite_adr[31:0] = wbs_adr_i[31:0] - 32'h30000000                                   ;

always @(posedge wb_clk_i or posedge wb_rst_i) begin
    if (wb_rst_i) begin
        axilite_awready <= 1'b0;
        axilite_wready  <= 1'b0;
        axilite_arready <= 1'b0;
    end else begin
        if (wbs_ack_o) begin
            axilite_awready <= 1'b0;
            axilite_wready  <= 1'b0;
            axilite_arready <= 1'b0;
        end else begin
            axilite_awready <= awready | axilite_awready;
            axilite_wready  <= wready  | axilite_wready ;
            axilite_arready <= arready | axilite_arready;
        end
    end
end

assign awvalid                   = axilite_en &  wbs_we_i & ~axilite_awready;
assign awaddr[(pADDR_WIDTH-1):0] = axilite_adr[(pADDR_WIDTH-1):0]           ;

assign wvalid                    = axilite_en &  wbs_we_i & ~axilite_wready ;
assign wdata[(pDATA_WIDTH-1):0]  = wbs_dat_i[(pDATA_WIDTH-1):0]             ;

assign arvalid                   = axilite_en & ~wbs_we_i & ~axilite_arready;
assign araddr[(pADDR_WIDTH-1):0] = axilite_adr[(pADDR_WIDTH-1):0]           ;

assign rready                    = axilite_en & ~wbs_we_i                   ;

// AXI-Stream
assign axistream_x = fir_en & (wbs_adr_i[6:0] == 7'h40);
assign axistream_y = fir_en & (wbs_adr_i[6:0] == 7'h44);

always @(posedge wb_clk_i or posedge wb_rst_i) begin
    if (wb_rst_i) begin
        axistream_count[31:0] <= 32'b0;
    end else begin
        if (axistream_x & wbs_ack_o) begin
            axistream_count[31:0] <= axistream_count[31:0] + 32'b1;
        end else begin
            axistream_count[31:0] <= axistream_count[31:0];
        end
    end
end

assign ss_tvalid                   = axistream_x &  wbs_we_i                                ;
assign ss_tdata[(pDATA_WIDTH-1):0] = wbs_dat_i[(pDATA_WIDTH-1):0]                           ;
assign ss_tlast                    = ss_tvalid & (axistream_count[31:0] == pDATA_LENGTH - 1);

assign sm_tready                   = axistream_y & ~wbs_we_i                                ;
```

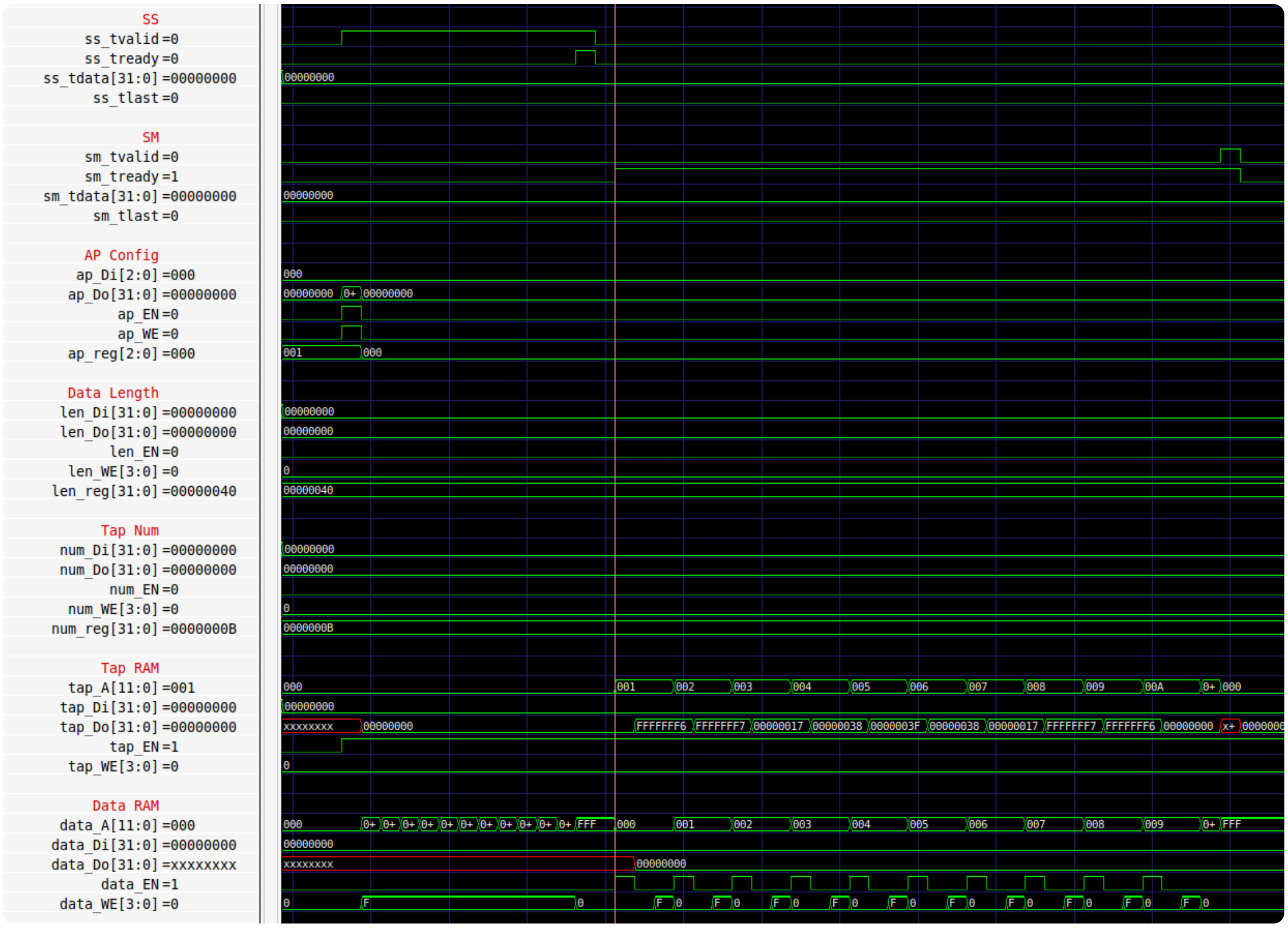
FIR Engine



```
AXI-Write
awvalid =0
awaddr[11:0] =678
awready =0
wvalid =0
wdata[31:0] =xxxxxxxx
wready =0

SS
ss_tvalid =0
ss_tready =0
ss_tdata[31:0] =xxxxxxxx
ss_tlast =0

SM
sm_tvalid =0
sm_tready =0
sm_tdata[31:0] =00000000
sm_tlast =0

AP Config
ap_Di[2:0] =000
ap_Do[31:0] =00000000
ap_EN =0
ap_WE =0
ap_reg[2:0] =110

Data Length
len_Di[31:0] =xxxxxxxx
len_Do[31:0] =00000000
len_EN =0
len_WE[3:0] =0
len_reg[31:0] =00000040

Tap Num
num_Di[31:0] =xxxxxxxx
num_Do[31:0] =00000000
num_EN =0
num_WE[3:0] =0
num_reg[31:0] =0000000B

Tap RAM
tap_A[11:0] =000
tap_Di[31:0] =xxxxxxxx
tap_Do[31:0] =00000000
tap_EN =0
tap_WE[3:0] =0

Data RAM
data_A[11:0] =000
data_Di[31:0] =00000000
data_Do[31:0] =00000035
data_EN =0
data_WE[3:0] =0
```

The FIR Engine receives the AXI-Lite and AXI-Stream transactions to compute the FIR outputs. When ap_start is set to 1, the FIR engine starts the computation by reading X[n] and output Y[n] through the AXI-Stream.

What is the FIR engine theoretical throughput, i.e. data rate? Measured throughput?

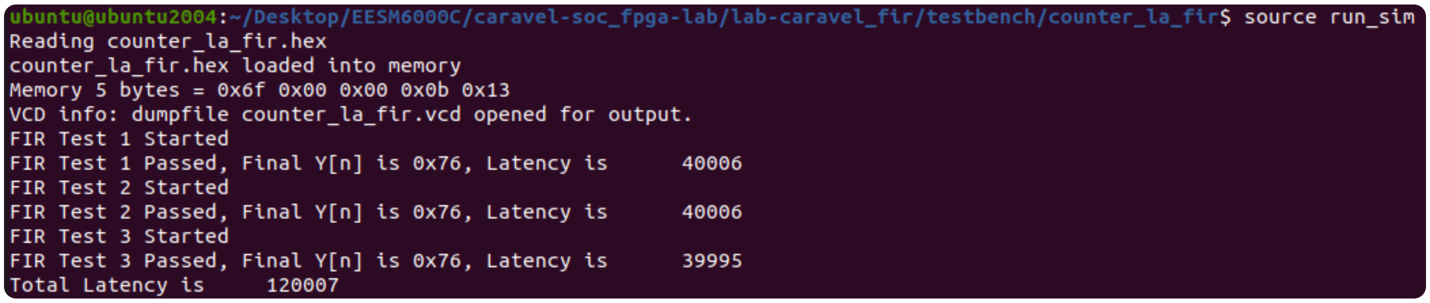


The theoretical delay per Y[n] output is 33 cycles. With a data length of 64, the theoretical delay is 2112 cycles.

Therefore, the thoretical throughput per 64 data sets is the following:

```
Theoretical throughput: 1/2112 = 0.00047348484 data per cycle
```

However, the actual measured delay for a data length of 64 is larger than expected.



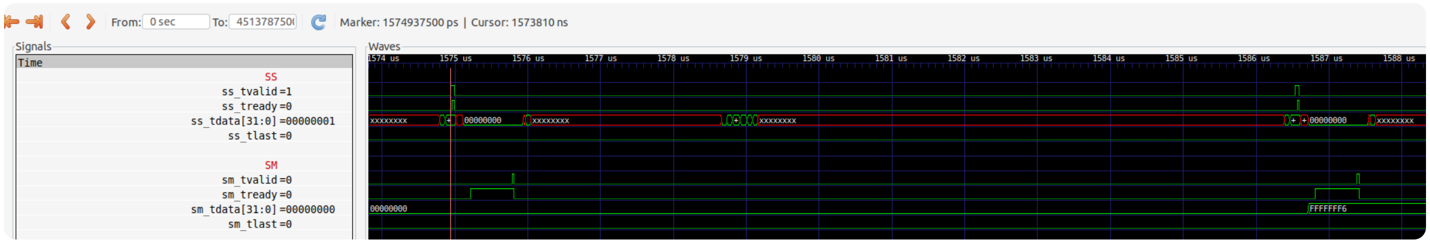
```
ubuntu@ubuntu2004:~/Desktop/EESM6000C/caravel-soc_fpga-lab/lab-caravel_fir/testbench/counter_la_fir$ source run_sim
Reading counter_la_fir.hex
counter_la_fir.hex loaded into memory
Memory 5 bytes = 0x6f 0x00 0x00 0x0b 0x13
VCD info: dumpfile counter_la_fir.vcd opened for output.
FIR Test 1 Started
FIR Test 1 Passed, Final Y[n] is 0x76, Latency is      40006
FIR Test 2 Started
FIR Test 2 Passed, Final Y[n] is 0x76, Latency is      40006
FIR Test 3 Started
FIR Test 3 Passed, Final Y[n] is 0x76, Latency is      39995
Total Latency is      120007
```

```
Measured throughput: 1/40006 = 0.00002499625 data per cycle
```

The cause of the increase in delay from theoretical is due to the time between each X[n] input. The FIR engine stalls and waits until the next X[n] input is received through the AXI-Stream.

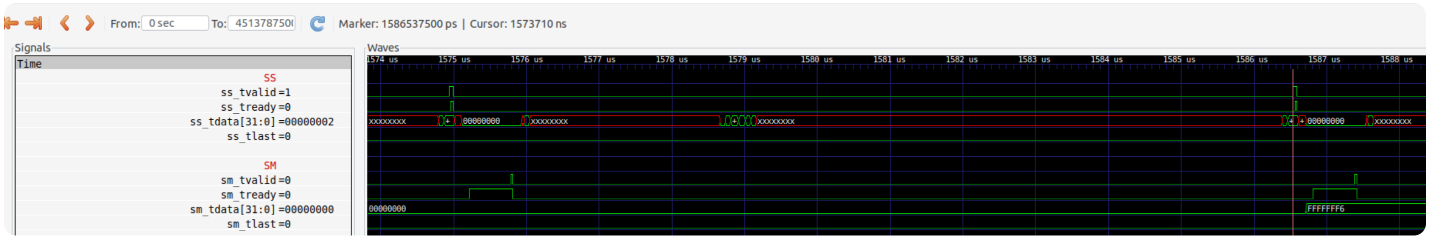
What is the latency for firmware to feed data?

The latency for the firmware to feed data can be determined through the cycles between each X[n] input from AXI-Stream.



From: 0 sec To: 451378750 Marker: 1574937500 ps | Cursor: 1573810 ns

```
Time
SS
ss_tvalid=1
ss_tready=0
ss_tdata[31:0]=00000001
ss_tlast=0
SM
sm_tvalid=0
sm_tready=0
sm_tdata[31:0]=00000000
sm_tlast=0
```



From: 0 sec To: 451378750 Marker: 1586537500 ps | Cursor: 1573710 ns

```
Time
SS
ss_tvalid=1
ss_tready=0
ss_tdata[31:0]=00000002
ss_tlast=0
SM
sm_tvalid=0
sm_tready=0
sm_tdata[31:0]=00000000
sm_tlast=0
```

```
Latency: (1586537500 - 1574937500) ps / (25000) ps = 464 cycles
```

What techniques are used to improve the throughput?

There are several techniques that can be done to improve the throughput. In terms of hardware, the FIR engine can be optimized.

The FIR engine can only receive X[n] and send Y[n] one at a time. If there is no data sent from the RISC-V CPU to the FIR engine, the FIR engine will stall, affecting the throughput. To improve this, a buffer can be added to store X[n] inputs and Y[n] output, and will not stall the FIR engine even if the SS/SM Bus tvalid is not asserted.

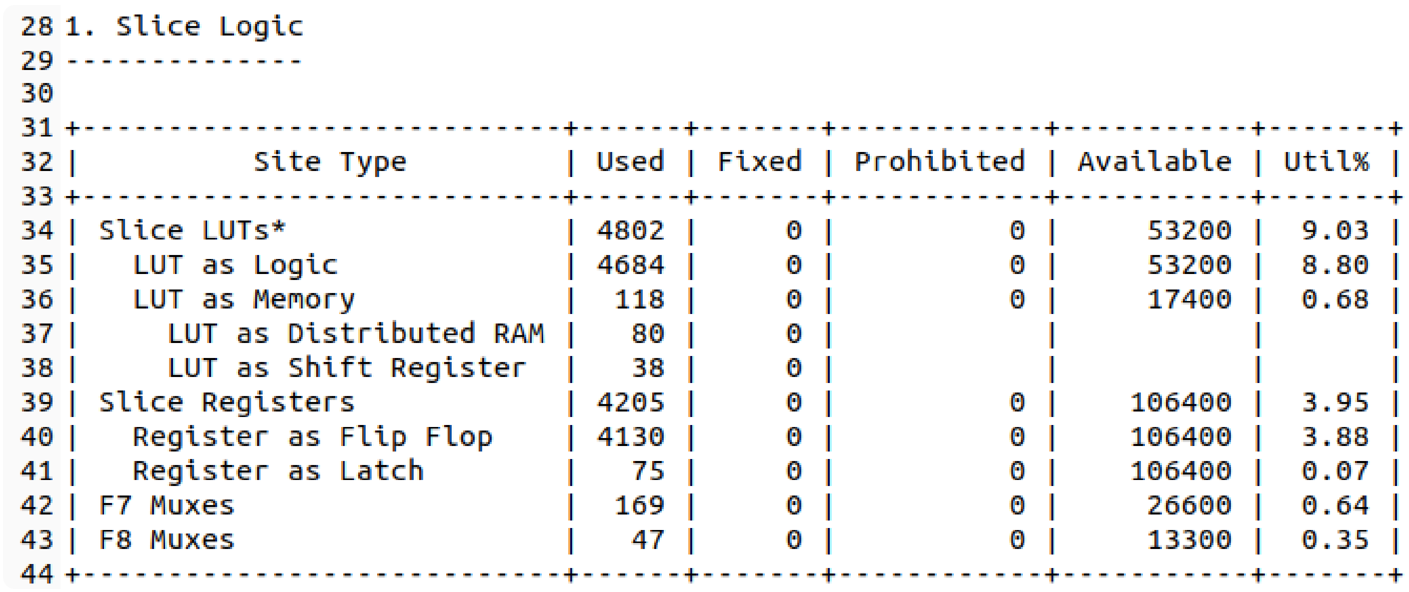
Another optimization done in the FIR engine is parallelism. In the current design, multiply and sum actions are done in separate cycles. The FIR engine can take advantage of the parallelism by using the Multiplier and Adder at the same cycles. When the previous dataset is using the Adder, the next dataset can use the Multiplier. This will reduce the FIR calculation delay and improve the throughput.

In terms of software, the throughput can be improved as well. There is a large latency for the firmware to feed the data. To improve this, the X[n] input can be decoupled from the Y[n] output, and allow X[n] to input into the user project area before receiving the Y[n] output. This will improve the delay between each data input and improve the throughput.

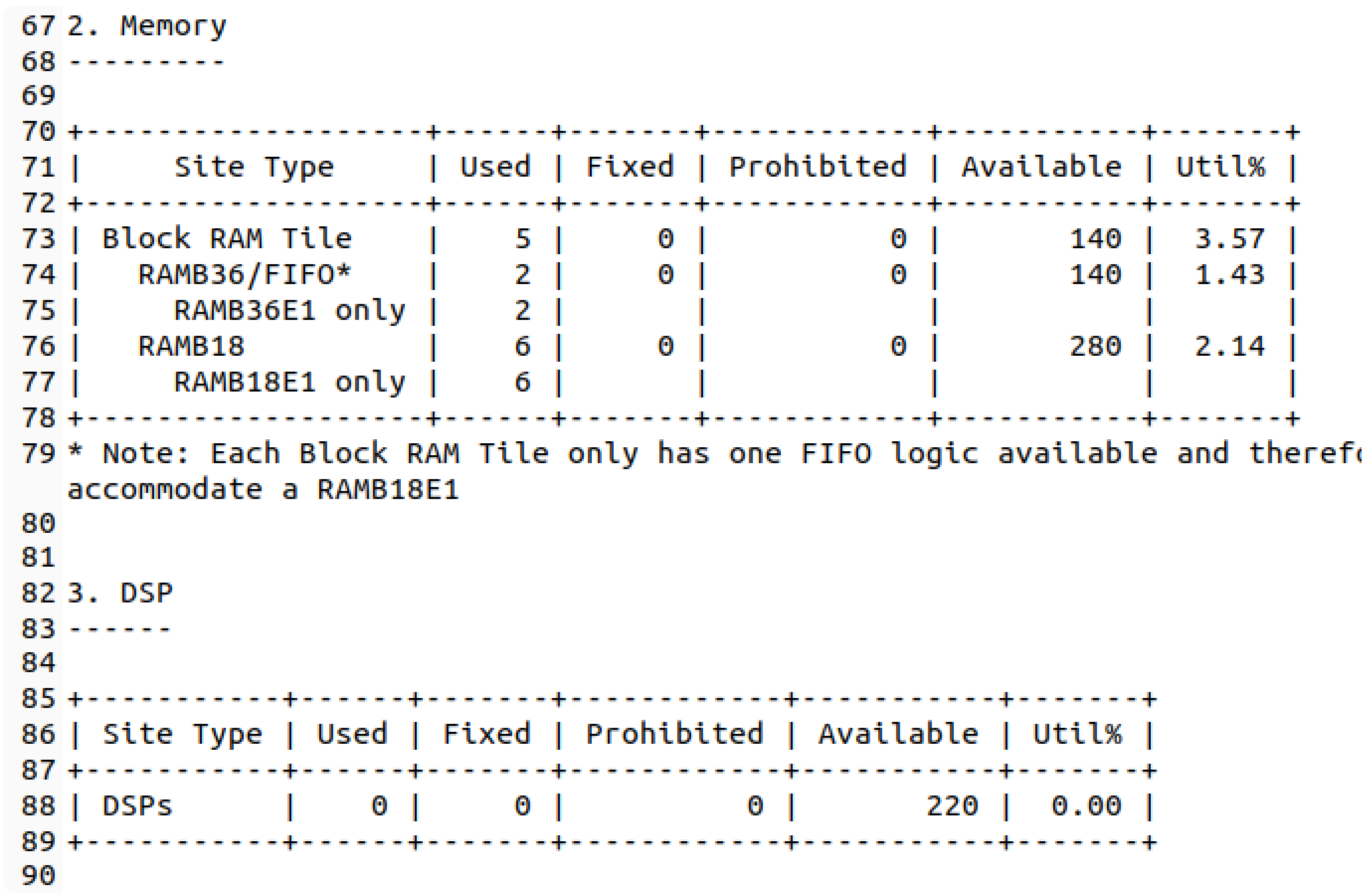
Any other insights?

Synthesis

Utilization Report



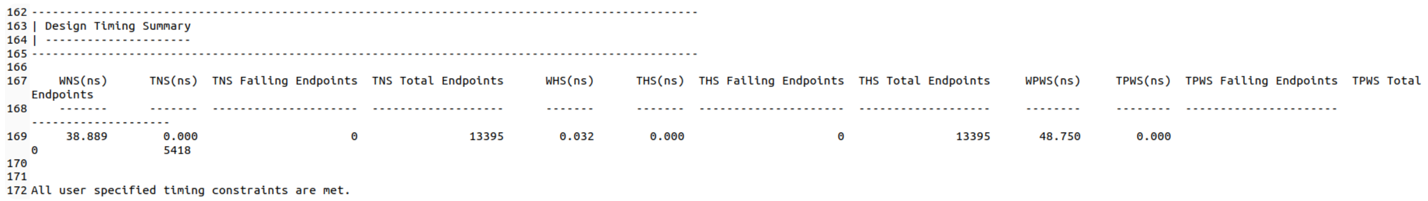
```
1. Slice Logic
--------------

+----------------------------+------+-------+------------+-----------+-------+
|          Site Type         | Used | Fixed | Prohibited | Available | Util% |
+----------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*                | 4802 |     0 |          0 |     53200 |  9.03 |
|   LUT as Logic             | 4684 |     0 |          0 |     53200 |  8.80 |
|   LUT as Memory            |  118 |     0 |          0 |     17400 |  0.68 |
|     LUT as Distributed RAM |   80 |     0 |            |           |       |
|     LUT as Shift Register  |   38 |     0 |            |           |       |
| Slice Registers            | 4205 |     0 |          0 |    106400 |  3.95 |
|   Register as Flip Flop    | 4130 |     0 |          0 |    106400 |  3.88 |
|   Register as Latch        |   75 |     0 |          0 |    106400 |  0.07 |
| F7 Muxes                   |  169 |     0 |          0 |     26600 |  0.64 |
| F8 Muxes                   |   47 |     0 |          0 |     13300 |  0.35 |
+----------------------------+------+-------+------------+-----------+-------+
```



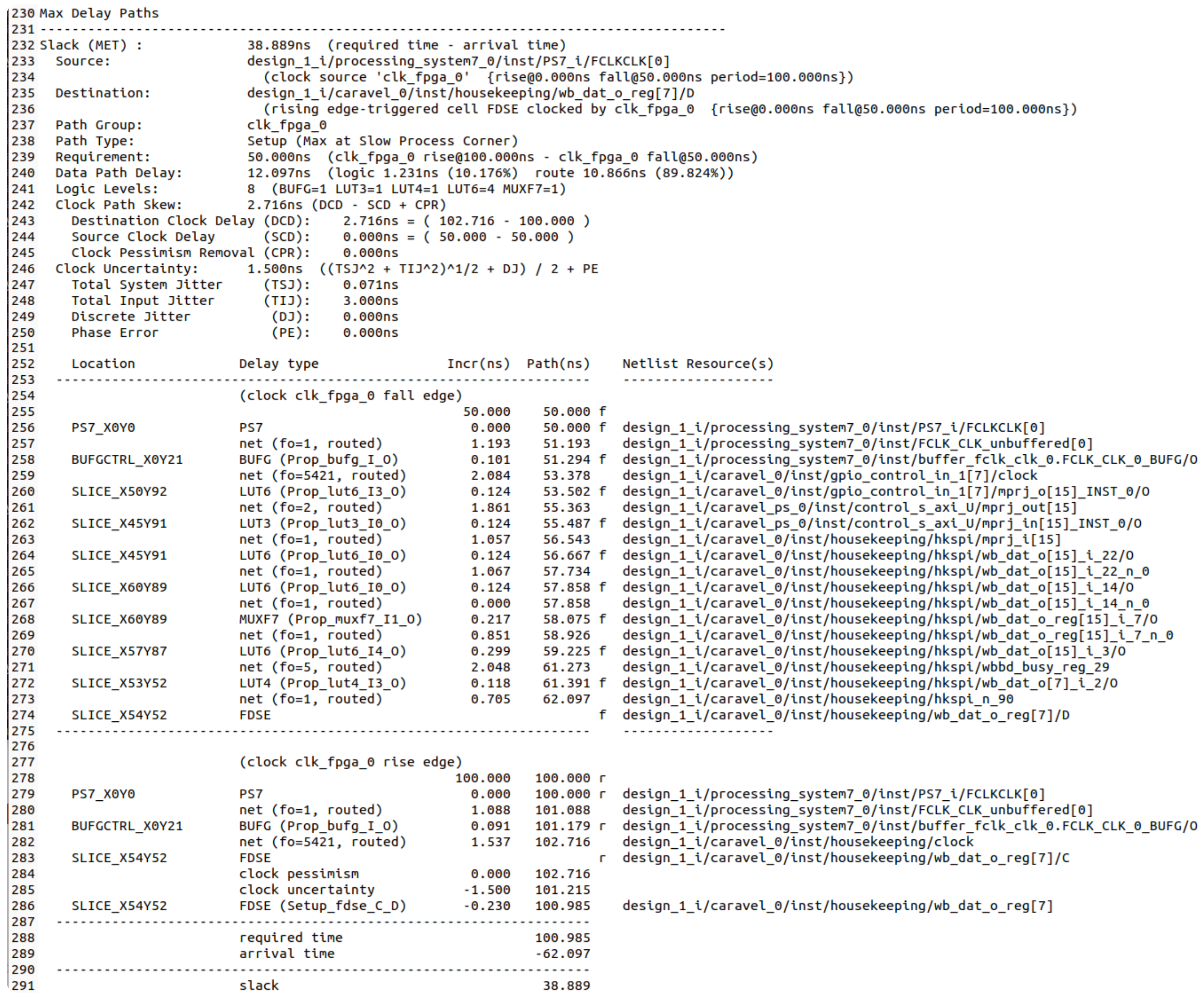
```
2. Memory
---------

+-------------------+------+-------+------------+-----------+-------+
|     Site Type     | Used | Fixed | Prohibited | Available | Util% |
+-------------------+------+-------+------------+-----------+-------+
| Block RAM Tile    |    5 |     0 |          0 |       140 |  3.57 |
|   RAMB36/FIFO*    |    2 |     0 |          0 |       140 |  1.43 |
|     RAMB36E1 only |    2 |       |            |           |       |
|   RAMB18          |    6 |     0 |          0 |       280 |  2.14 |
|     RAMB18E1 only |    6 |       |            |           |       |
+-------------------+------+-------+------------+-----------+-------+
* Note: Each Block RAM Tile only has one FIFO logic available and theref
accommodate a RAMB18E1


3. DSP
------

+-----------+------+-------+------------+-----------+-------+
| Site Type | Used | Fixed | Prohibited | Available | Util% |
+-----------+------+-------+------------+-----------+-------+
| DSPs      |    0 |     0 |          0 |       220 |  0.00 |
+-----------+------+-------+------------+-----------+-------+

```

Timing Report

Using the 10 MHz FPGA synthesis



```
------------------------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)  TPWS Failing Endpoints  TPWS Total
Endpoints
    -------      -------  --------------------  -------------------      -------      -------  --------------------  -------------------     --------     --------  ----------------------
--------------------
     38.889        0.000                     0                13395        0.032        0.000                     0                13395       48.750        0.000
0                 5418


All user specified timing constraints are met.
```



```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             38.889ns  (required time - arrival time)
  Source:                 design_1_i/processing_system7_0/inst/PS7_i/FCLKCLK[0]
                            (clock source 'clk_fpga_0'  {rise@0.000ns fall@50.000ns period=100.000ns})
  Destination:            design_1_i/caravel_0/inst/housekeeping/wb_dat_o_reg[7]/D
                            (rising edge-triggered cell FDSE clocked by clk_fpga_0  {rise@0.000ns fall@50.000ns period=100.000ns})
  Path Group:             clk_fpga_0
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            50.000ns  (clk_fpga_0 rise@100.000ns - clk_fpga_0 fall@50.000ns)
  Data Path Delay:        12.097ns  (logic 1.231ns (10.176%)  route 10.866ns (89.824%))
  Logic Levels:           8  (BUFG=1 LUT3=1 LUT4=1 LUT6=4 MUXF7=1)
  Clock Path Skew:        2.716ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.716ns = ( 102.716 - 100.000 )
    Source Clock Delay      (SCD):    0.000ns = ( 50.000 - 50.000 )
    Clock Pessimism Removal (CPR):    0.000ns
  Clock Uncertainty:      1.500ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    3.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         (clock clk_fpga_0 fall edge)
                                                     50.000    50.000 f
    PS7_X0Y0             PS7                          0.000    50.000 f  design_1_i/processing_system7_0/inst/PS7_i/FCLKCLK[0]
                         net (fo=1, routed)           1.193    51.193    design_1_i/processing_system7_0/inst/FCLK_CLK_unbuffered[0]
    BUFGCTRL_X0Y21       BUFG (Prop_bufg_I_O)         0.101    51.294 f  design_1_i/processing_system7_0/inst/buffer_fclk_clk_0.FCLK_CLK_0_BUFG/O
                         net (fo=5421, routed)        2.084    53.378    design_1_i/caravel_0/inst/gpio_control_in_1[7]/clock
    SLICE_X50Y92         LUT6 (Prop_lut6_I3_O)        0.124    53.502 f  design_1_i/caravel_0/inst/gpio_control_in_1[7]/mprj_o[15]_INST_0/O
                         net (fo=2, routed)           1.861    55.363    design_1_i/caravel_ps_0/inst/control_s_axi_U/mprj_out[15]
    SLICE_X45Y91         LUT3 (Prop_lut3_I0_O)        0.124    55.487 f  design_1_i/caravel_ps_0/inst/control_s_axi_U/mprj_in[15]_INST_0/O
                         net (fo=1, routed)           1.057    56.543    design_1_i/caravel_0/inst/housekeeping/hkspi/mprj_i[15]
    SLICE_X45Y91         LUT6 (Prop_lut6_I0_O)        0.124    56.667 f  design_1_i/caravel_0/inst/housekeeping/hkspi/wb_dat_o[15]_i_22/O
                         net (fo=1, routed)           1.067    57.734    design_1_i/caravel_0/inst/housekeeping/hkspi/wb_dat_o[15]_i_22_n_0
    SLICE_X60Y89         LUT6 (Prop_lut6_I0_O)        0.124    57.858 f  design_1_i/caravel_0/inst/housekeeping/hkspi/wb_dat_o[15]_i_14/O
                         net (fo=1, routed)           0.000    57.858    design_1_i/caravel_0/inst/housekeeping/hkspi/wb_dat_o[15]_i_14_n_0
    SLICE_X60Y89         MUXF7 (Prop_muxf7_I1_O)      0.217    58.075 f  design_1_i/caravel_0/inst/housekeeping/hkspi/wb_dat_o_reg[15]_i_7/O
                         net (fo=1, routed)           0.851    58.926    design_1_i/caravel_0/inst/housekeeping/hkspi/wb_dat_o_reg[15]_i_7_n_0
    SLICE_X57Y87         LUT6 (Prop_lut6_I4_O)        0.299    59.225 f  design_1_i/caravel_0/inst/housekeeping/hkspi/wb_dat_o[15]_i_3/O
                         net (fo=5, routed)           2.048    61.273    design_1_i/caravel_0/inst/housekeeping/hkspi/wbbd_busy_reg_29
    SLICE_X53Y52         LUT4 (Prop_lut4_I3_O)        0.118    61.391 f  design_1_i/caravel_0/inst/housekeeping/hkspi/wb_dat_o[7]_i_2/O
                         net (fo=1, routed)           0.705    62.097    design_1_i/caravel_0/inst/housekeeping/hkspi_n_90
    SLICE_X54Y52         FDSE                                           f  design_1_i/caravel_0/inst/housekeeping/wb_dat_o_reg[7]/D
  -------------------------------------------------------------------    -------------------

                         (clock clk_fpga_0 rise edge)
                                                    100.000   100.000 r
    PS7_X0Y0             PS7                          0.000   100.000 r  design_1_i/processing_system7_0/inst/PS7_i/FCLKCLK[0]
                         net (fo=1, routed)           1.088   101.088    design_1_i/processing_system7_0/inst/FCLK_CLK_unbuffered[0]
    BUFGCTRL_X0Y21       BUFG (Prop_bufg_I_O)         0.091   101.179 r  design_1_i/processing_system7_0/inst/buffer_fclk_clk_0.FCLK_CLK_0_BUFG/O
                         net (fo=5421, routed)        1.537   102.716    design_1_i/caravel_0/inst/housekeeping/clock
    SLICE_X54Y52         FDSE                                         r  design_1_i/caravel_0/inst/housekeeping/wb_dat_o_reg[7]/C
                         clock pessimism              0.000   102.716
                         clock uncertainty           -1.500   101.215
    SLICE_X54Y52         FDSE (Setup_fdse_C_D)       -0.230   100.985    design_1_i/caravel_0/inst/housekeeping/wb_dat_o_reg[7]
  -------------------------------------------------------------------
                         required time                        100.985
                         arrival time                         -62.097
  -------------------------------------------------------------------
                         slack                                 38.889
```

GitHub

https://github.com/AnthonyGithub/EESM6000C-Lab-4/tree/main/Lab 4-2

(https://github.com/AnthonyGithub/EESM6000C-Lab-4/tree/main/Lab%204-2)