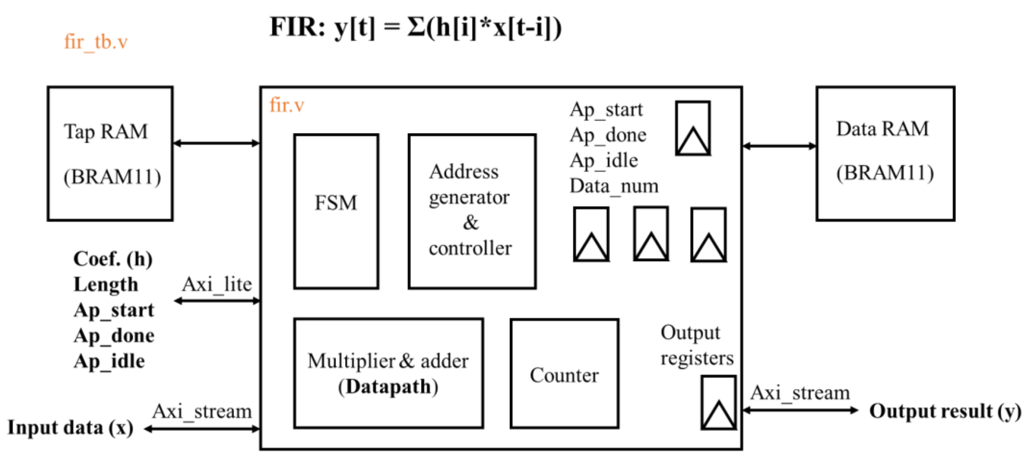
112-1 SoC Design Laboratory

Lab3

112061573 王語卉

I. Block diagram

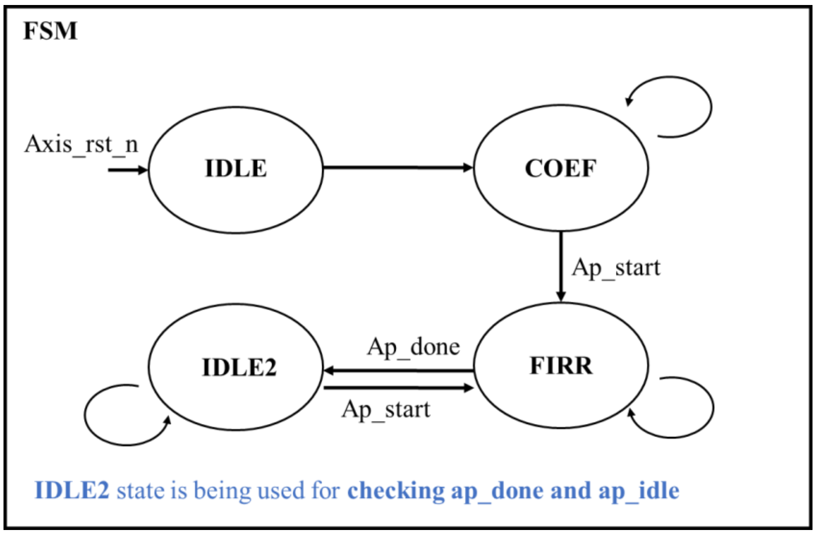
In this lab, we're going to use Verilog code to implement the FIR system. The following is the block diagram of the whole FIR system:



The blocks are for readability. There're actually no submodules in fir.v

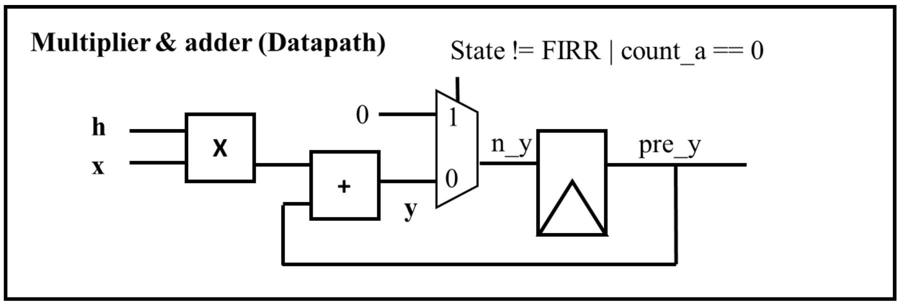
The input and output port have been defined. I use **bram11** for tap RAM and data RAM. There're an FSM, a multiplier and an adder, address generator and some registers in my design.

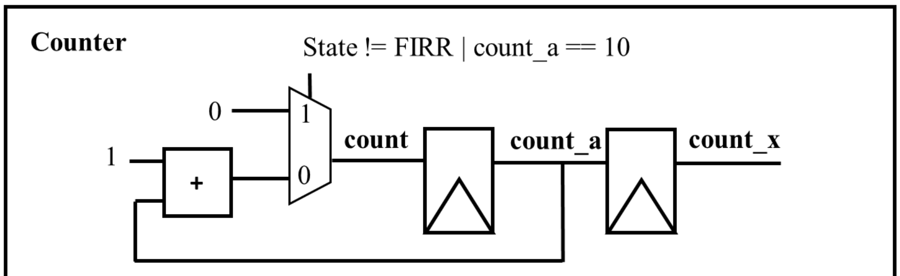
- The FSM:



I use four states in the FSM. The **COEF** state is used for the tap RAM to store the coefficient and to read out the coefficient. The **FIRR** state is used to read data in, to store data, to read the data and coefficient out and to perform multiply and add operation. The **IDLE2** state is used to check the ap_done and ap_idle.

- The datapath and the control counter:





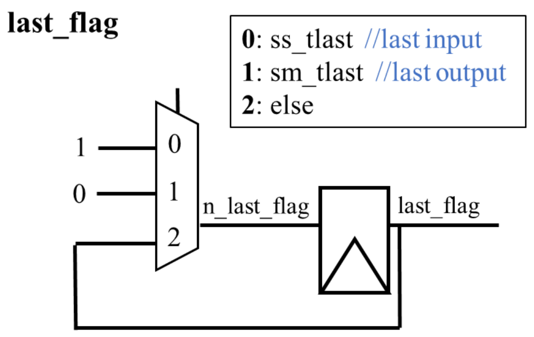
count_a for counting **a**ddress (since the data will be received in the next cycle)
count_x for counting y accumulation

state	COEF	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR
count	0	1	2	3	4	5	6	7	8	9	10	0	1	2	3	4
count_a	0	0	1	2	3	4	5	6	7	8	9	10	0	1	2	3
count_x	0	0	0	1	2	3	4	5	6	7	8	9	10	0	1	2
y	X	X	d0	d0~d1	d0~d2	d0~d3	d0~d4	d0~d5	d0~d6	d0~d7	d0~d8	d0~d9	d0~d10	d0	d0~d1	d0~d2
n_y	0	0	d0	d0~d1	d0~d2	d0~d3	d0~d4	d0~d5	d0~d6	d0~d7	d0~d8	d0~d9	0	d0	d0~d1	d0~d2
pre_y	X	0	0	d0	d0~d1	d0~d2	d0~d3	d0~d4	d0~d5	d0~d6	d0~d7	d0~d8	d0~d9	0	d0	d0~d1

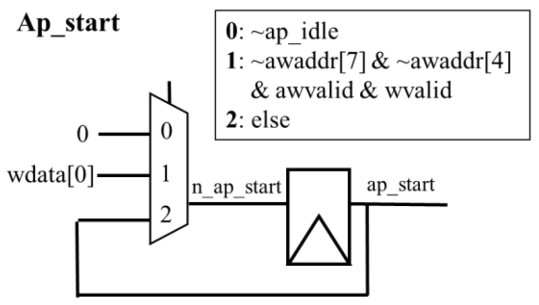
I use **count**, **count_a** (for counting RAM address) and **count_x** (for counting accumulations) to control the datapath, the address and the output control.

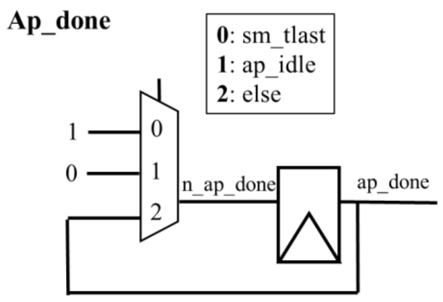
out_sm_tvalid = count_x == 10
out_ sm_tlast = (last_flag == 1) & (count_x == 10)
out_ sm_tdata = y
out_ ss_tready = count == 10 //load next data

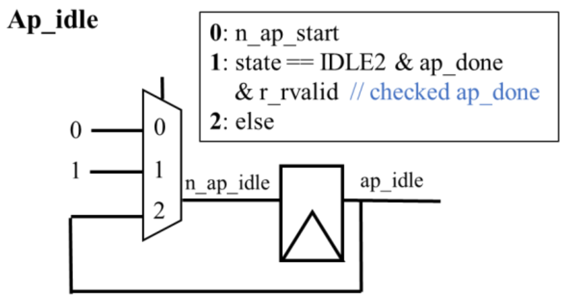
(The output port are all blocked by the register, therefore the actual output will delay one cycle)

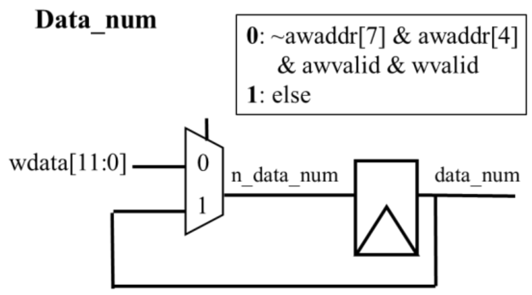


- <u>The registers:</u>



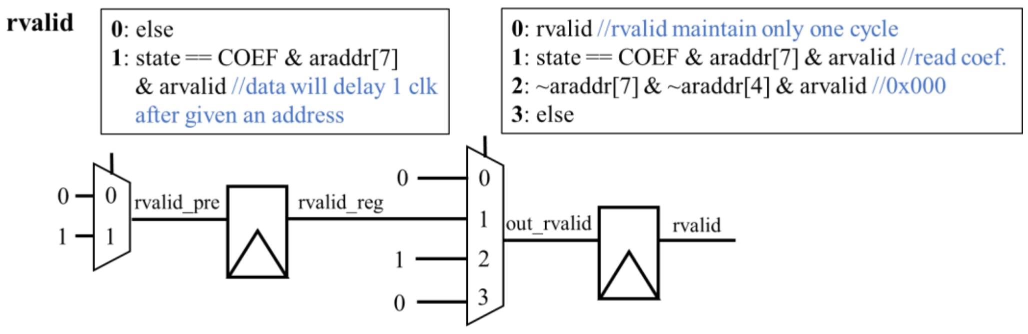


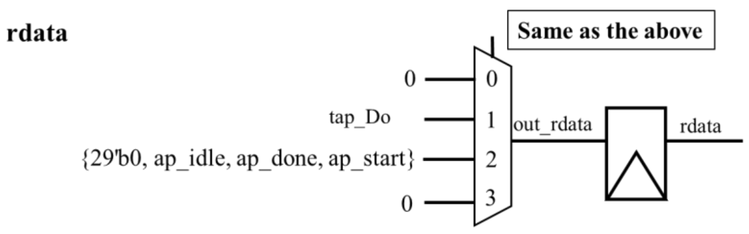




The registers are used to store the engine's current situation. Ap_done will last until ap_idle become 1. Ap_idle will become 1 after sending the ap_done signal out.

- <u>The control signals:</u>


rvalid
0: else
1: state == COEF & araddr[7] & arvalid //data will delay 1 clk after given an address
0: rvalid //rvalid maintain only one cycle
1: state == COEF & araddr[7] & arvalid //read coef.
2: ~araddr[7] & ~araddr[4] & arvalid //0x000
3: else
0
1
0
1
rvalid_pre
rvalid_reg
0
1
0
0
1
2
3
out_rvalid
rvalid



Arready = 1
Awready = wready = 1

Since the bram will output the data in the next cycle after given an address, I use one register to delay rvalid signal. Except for the COEF state that rvalid and rdata are being used to send the coefficient, they're sending ap_start/ap_done/ap_idle in other time.

The tap RAM and data RAM operations will be described in the next section.

II. Operations

The overall flow of the FIR engine is like below:

Control	State	Flow	Detail operations
	IDLE	none	
	COEF	Step1. Write	Write coefficient into **tap RAM** Write 0 into **data RAM**
		Step2. Read	Read coefficient from **tap RAM**
ap_start	FIRR	Every cycle	Read coefficient from **tap RAM** Read data from **data RAM**
		Every 11 cycle	Load new data **x** Write new data into **data RAM** Send the result **y** to the output port Update the address flag
ap_done	IDLE2	Step1. send ap_done	
		Step2. send ap_idle	

In COEF:

- In COEF state, **tap RAM** have two conditions: (1) awvalid & waddr[7] (write) (2) arvalid & raddr[7] (read).

  I use **waddr[6:0]** and **raddr[6:0]** for the tap RAM address in (1) and (2) respectively.

  In (1), the tap_EN and tap_WE are **wready** which means the data exchange (wready & wvalid). In (2), tap_EN is **out_rvalid** which means the output data is prepared.

- Since the **data RAM** is not initialized, I use the same address, EN and WE as tap RAM to initialize data RAM in this state.

In FIRR:

- We always need to access the RAM to get the coefficient and the data. Therefore, both of the EN are always on. The address of the **tap RAM** is the **count_a**. The WE of the tap RAM is 0.
- In FIR system, the data order changes every 11 cycles. Therefore, I use an additional register as **address_flag** to store the starting address of the data RAM for every 11 cycles.
- The newest data will store in the address that "**address_flag** – 1" points to. The table represents the change of the flag and the red word means the newest data.

flag	1	2	3	4	5	6	7	8	9	10	0	1	2	3	4	5
cycle	0~10	11~21	22~32	33~43	44~54	55~65	66~76	77~87	88~98	99~109	110~120	121~131	132~142	143~153	154~164	165~175
address	x(data)															
0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D0	D11	D11	D11	D11	D11
1	0	D1	D1	D1	D1	D1	D1	D1	D1	D1	D1	D1	D12	D12	D12	D12
2	0	0	D2	D2	D2	D2	D2	D2	D2	D2	D2	D2	D2	D13	D13	D13
3	0	0	0	D3	D3	D3	D3	D3	D3	D3	D3	D3	D3	D3	D14	D14
4	0	0	0	0	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D4	D15
5	0	0	0	0	0	D5	D5	D5	D5	D5	D5	D5	D5	D5	D5	D5
6	0	0	0	0	0	0	D6	D6	D6	D6	D6	D6	D6	D6	D6	D6
7	0	0	0	0	0	0	0	D7	D7	D7	D7	D7	D7	D7	D7	D7
8	0	0	0	0	0	0	0	0	D8	D8	D8	D8	D8	D8	D8	D8
9	0	0	0	0	0	0	0	0	0	D9	D9	D9	D9	D9	D9	D9
10	0	0	0	0	0	0	0	0	0	0	D10	D10	D10	D10	D10	D10

- The data order in every 11 cycles is represented in the next table. I use count_a to calculate the address.

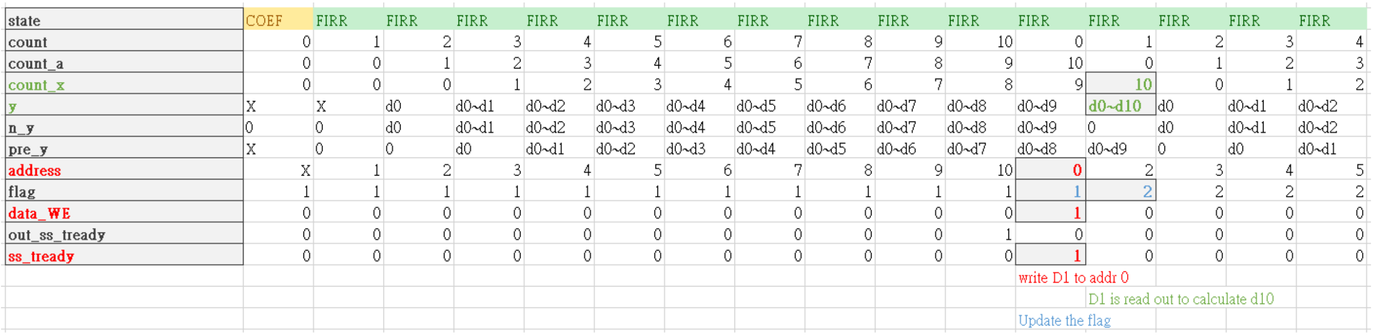
Count_a

flag	1	2	3	4	5	6	7	8	9	10	0	1	2	3	4	5
cycle	0~10	11~21	22~32	33~43	44~54	55~65	66~76	77~87	88~98	99~109	110~120	121~131	132~142	143~153	154~164	165~175
address	x(data)															
0	D0 10	D0 9	D0 8	D0 7	D0 6	D0 5	D0 4	D0 3	D0	D0	D0	D11	D11	D11	D11	D11
1	0 0	D1 10	D1 9	D1 8	D1 7	D1 6	D1 5	D1 4	D1	D1	D1	D1	D12	D12	D12	D12
2	1 0	0 0	D2 10	D2 9	D2 8	D2 7	D2 6	D2 5	D2	D2	D2	D2	D2	D13	D13	D13
3	2 0	1 0	0 0	D3 10	D3 9	D3 8	D3 7	D3 6	D3	D3	D3	D3	D3	D3	D14	D14
4	3 0	2 0	1 0	0 0	D4 10	D4 9	D4 8	D4 7	D4	D4	D4	D4	D4	D4	D4	D15
5	4 0	3 0	2 0	1 0	0 0	D5 10	D5 9	D5 8	D5	D5	D5	D5	D5	D5	D5	D5
6	5 0	4 0	3 0	2 0	1 0	0 0	D6 10	D6 9	D6	D6	D6	D6	D6	D6	D6	D6
7	6 0	5 0	4 0	3 0	2 0	1 0	0 0	D7 10	D7	D7	D7	D7	D7	D7	D7	D7
8	7 0	6 0	5 0	4 0	3 0	2 0	1 0	0 0	D8	D8	D8	D8	D8	D8	D8	D8
9	8 0	7 0	6 0	5 0	4 0	3 0	2 0	1 0	0	D9	D9	D9	D9	D9	D9	D9
10	9 0	8 0	7 0	6 0	5 0	4 0	3 0	2 0	0	0	D10	D10	D10	D10	D10	D10

If "**count_a + flag <= 10**", then address is **"count_a + flag"**.

If "**count_a + flag > 10**", then address is **"count_a + flag – 11"**.

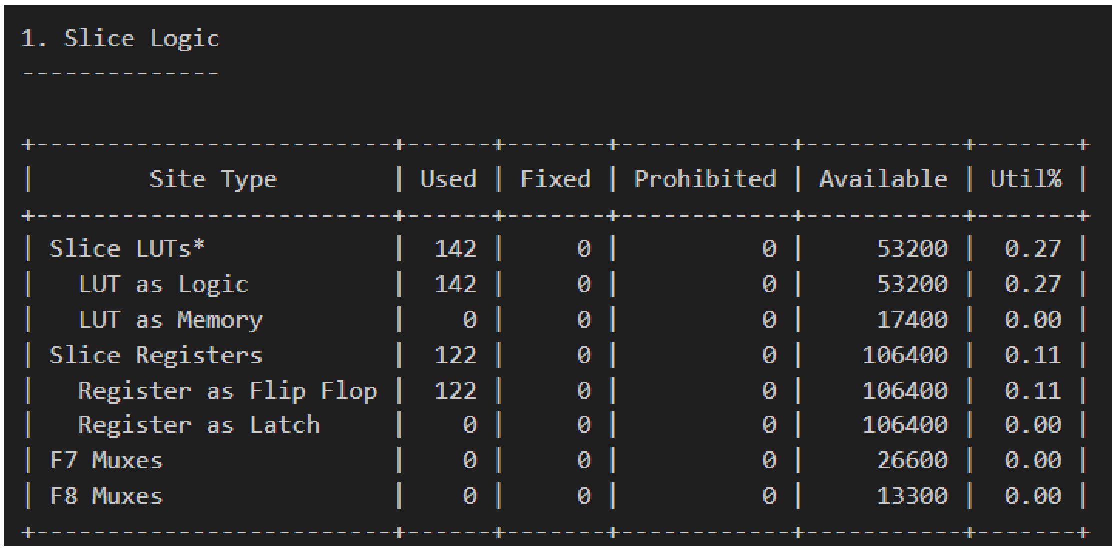
- When the address comes to "**address_flag** – 1", i.e., the last address in every 11 cycles, I set **ss_tready** to 1 to receive the new data. At the same time, **data_WE** is also set to 1 to write the new data into data RAM. Therefore, the new data will be read out in the next cycle to perform the last accumulation in these 11 cycles. Moreover, since the next cycle will start the new 11 cycles, the **address_flag,** i.e., the starting address, will be updated in this cycle, too.



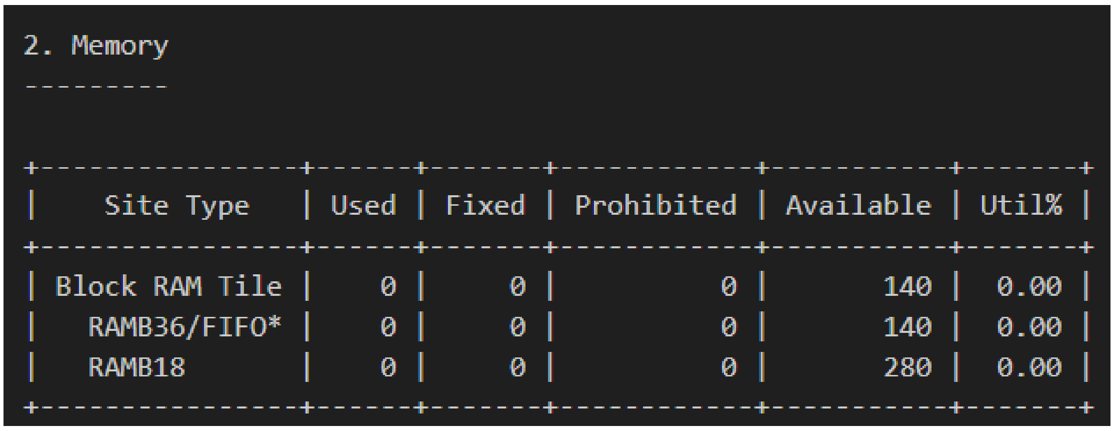
state	COEF	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR	FIRR
count	0	1	2	3	4	5	6	7	8	9	10	0	1	2	3	4
count_a	0	0	1	2	3	4	5	6	7	8	9	10	0	1	2	3
count_x	0	0	0	1	2	3	4	5	6	7	8	9	10	0	1	2
y	X	X	d0	d0~d1	d0~d2	d0~d3	d0~d4	d0~d5	d0~d6	d0~d7	d0~d8	d0~d9	d0~d10	d0	d0~d1	d0~d2
n_y	0	0	d0	d0~d1	d0~d2	d0~d3	d0~d4	d0~d5	d0~d6	d0~d7	d0~d8	d0~d9	0	d0	d0~d1	d0~d2
pre_y	X	0	0	d0	d0~d1	d0~d2	d0~d3	d0~d4	d0~d5	d0~d6	d0~d7	d0~d8	d0~d9	0	d0	d0~d1
address	X	1	2	3	4	5	6	7	8	9	10	0	2	3	4	5
flag	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2
data_WE	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
out_ss_tready	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
ss_tready	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
												write D1 to addr 0				
													D1 is read out to calculate d10			
												Update the flag				

- If count_x == 10, then we output the result **y**, and **out_sm_tvalid** will be 1. If the **last_flag** is 1, **out_sm_tlast** will be 1. When **sm_tlast** become 1, then **ap_done** will become 1.

III. Resource usage



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


+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |  142 |     0 |          0 |     53200 |  0.27 |
|   LUT as Logic          |  142 |     0 |          0 |     53200 |  0.27 |
|   LUT as Memory         |    0 |     0 |          0 |     17400 |  0.00 |
| Slice Registers         |  122 |     0 |          0 |    106400 |  0.11 |
|   Register as Flip Flop |  122 |     0 |          0 |    106400 |  0.11 |
|   Register as Latch     |    0 |     0 |          0 |    106400 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |     26600 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |     13300 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```



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


+----------------+------+-------+------------+-----------+-------+
|    Site Type   | Used | Fixed | Prohibited | Available | Util% |
+----------------+------+-------+------------+-----------+-------+
| Block RAM Tile |    0 |     0 |          0 |       140 |  0.00 |
|   RAMB36/FIFO* |    0 |     0 |          0 |       140 |  0.00 |
|   RAMB18       |    0 |     0 |          0 |       280 |  0.00 |
+----------------+------+-------+------------+-----------+-------+
```

There're no bram in my design.

IV. Timing report

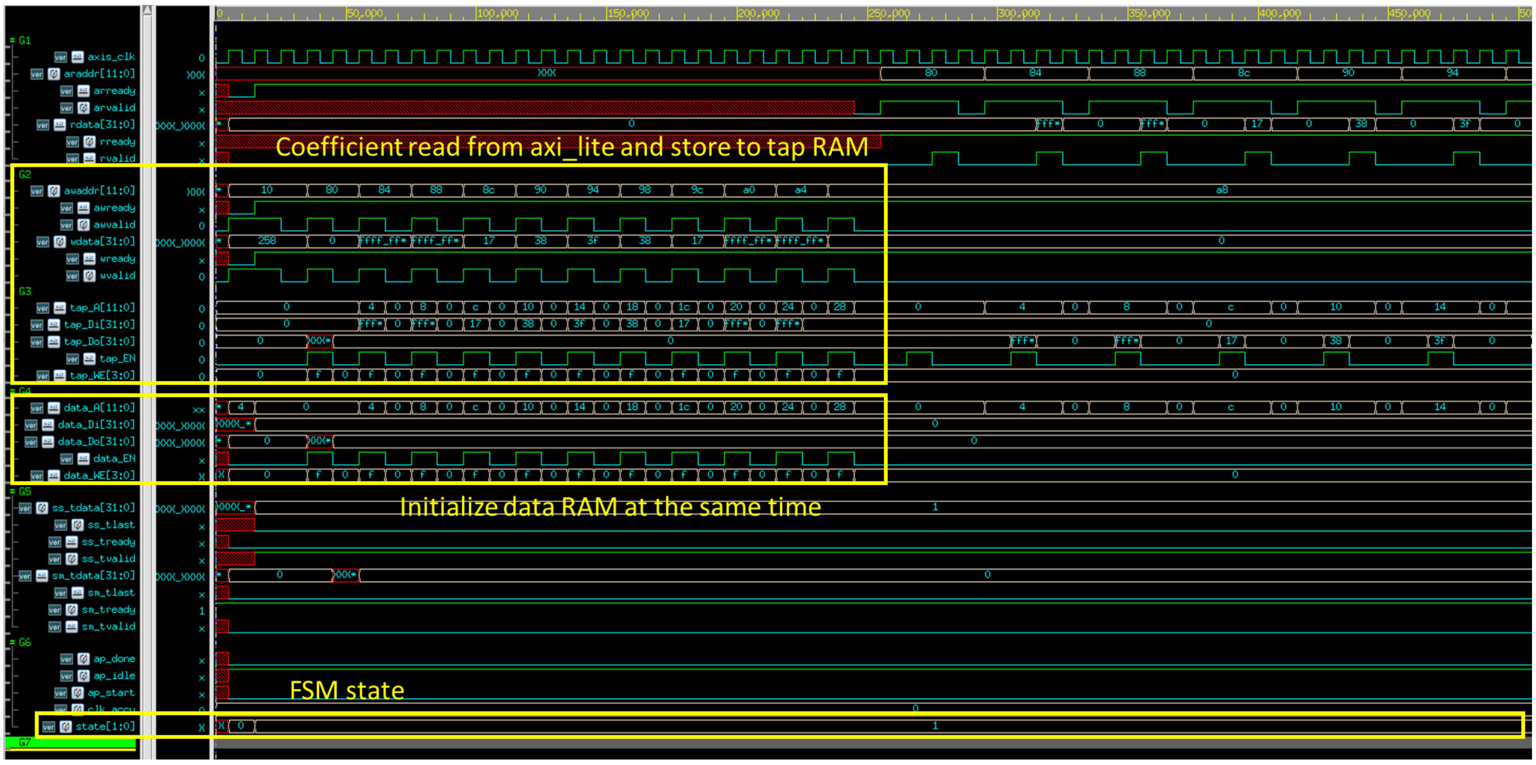
```
-------------------------------------------------------------------------------------------------
| Clock Summary
| -------------
-------------------------------------------------------------------------------------------------

Clock     Waveform(ns)       Period(ns)      Frequency(MHz)
-----     ------------       ----------      --------------
axis_clk  {0.000 7.500}      15.000          66.667
```

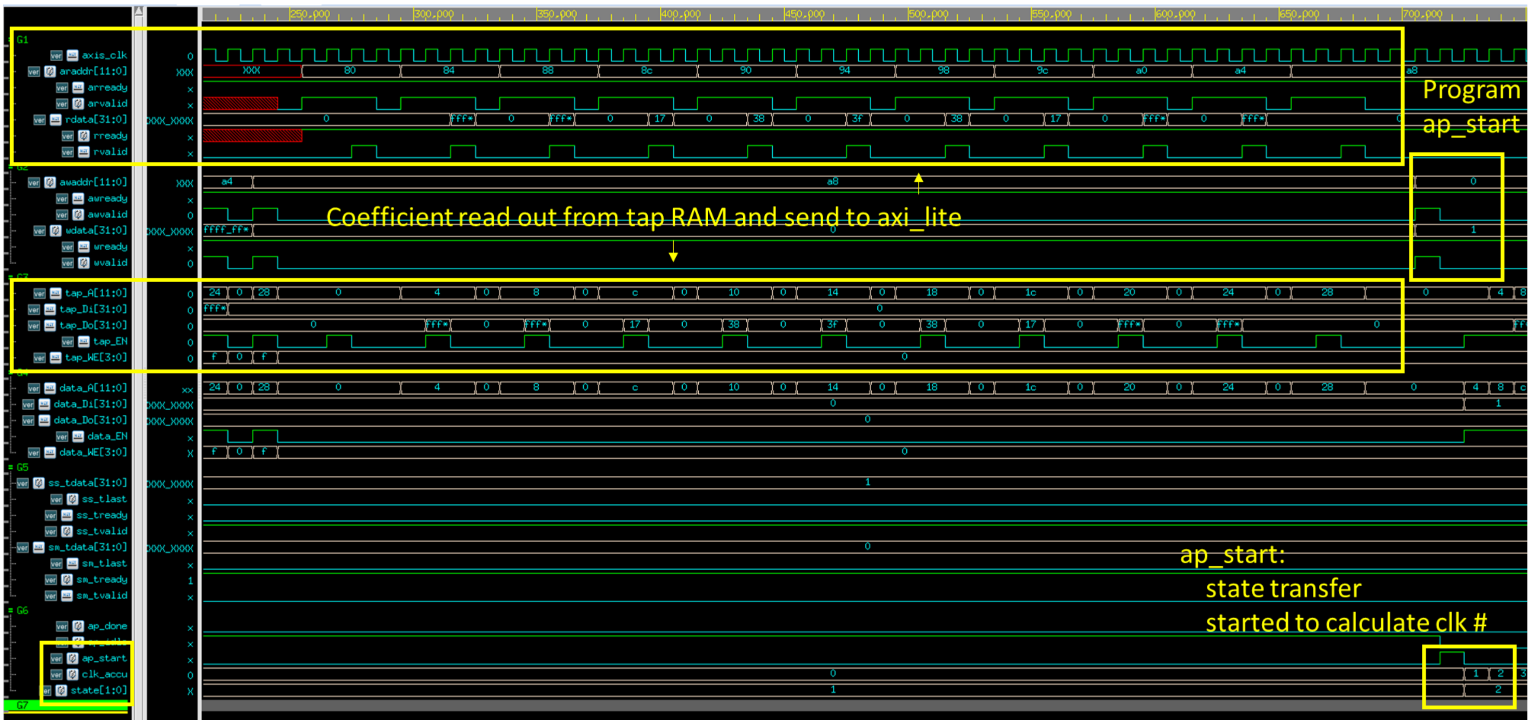
```
----------------------------------------------------------------------------------------------
From Clock:  axis_clk
  To Clock:  axis_clk

Setup :            0  Failing Endpoints,  Worst Slack        0.477ns,  Total Violation        0.000ns
Hold  :            0  Failing Endpoints,  Worst Slack        0.140ns,  Total Violation        0.000ns
PW    :            0  Failing Endpoints,  Worst Slack        7.000ns,  Total Violation        0.000ns
----------------------------------------------------------------------------------------------
```

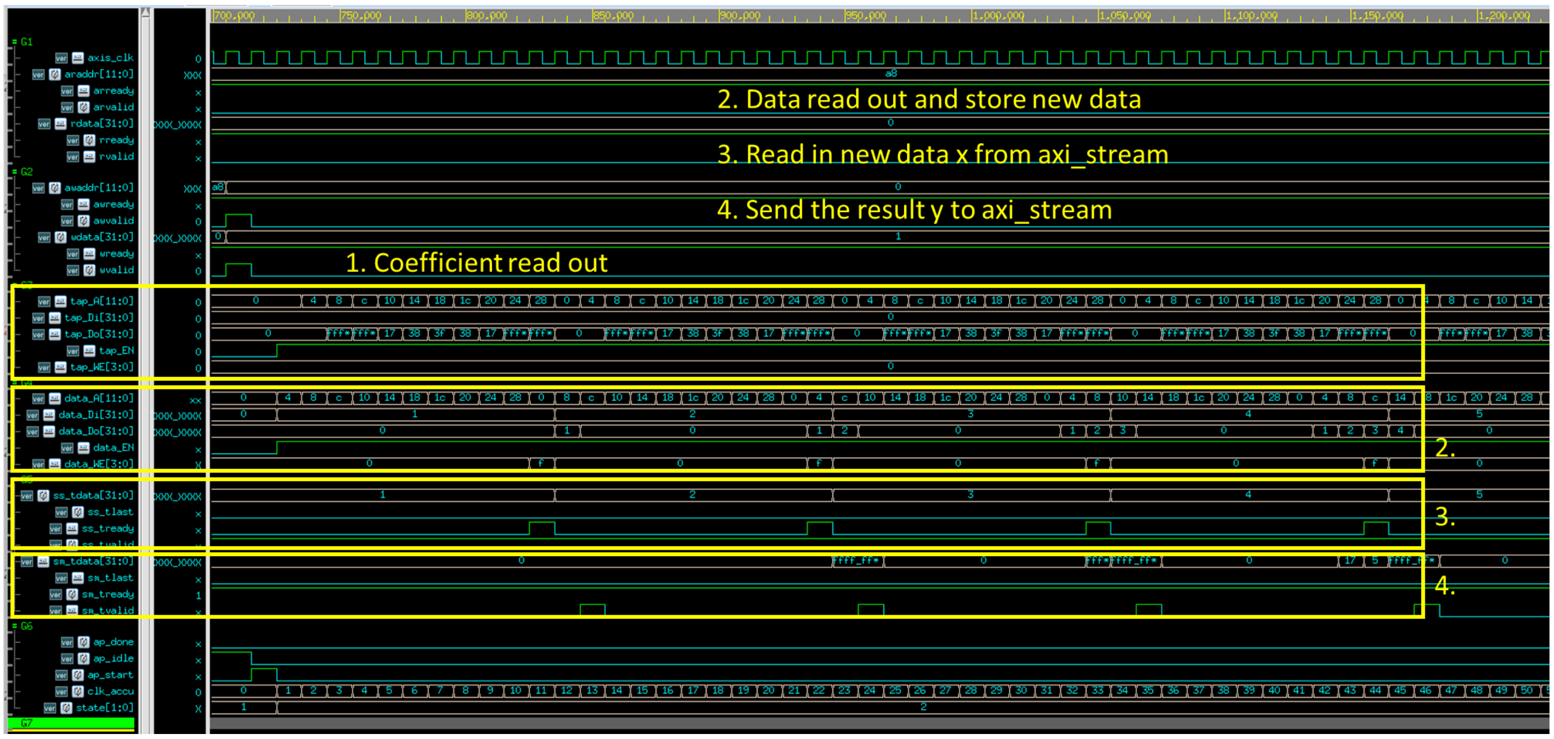
```
Max Delay Paths
--------------------------------------------------------------------------------------
Slack (MET) :             0.477ns  (required time - arrival time)
  Source:                 araddr[7]
                            (input port clocked by axis_clk  {rise@0.000ns fall@7.500ns period=15.000ns})
  Destination:            data_A[5]
                            (output port clocked by axis_clk  {rise@0.000ns fall@7.500ns period=15.000ns})
  Path Group:             axis_clk
  Path Type:              Max at Slow Process Corner
  Requirement:            15.000ns  (axis_clk rise@15.000ns - axis_clk rise@0.000ns)
  Data Path Delay:        6.988ns  (logic 3.978ns (56.932%)  route 3.009ns (43.068%))
  Logic Levels:           5  (IBUF=1 LUT3=1 LUT6=2 OBUF=1)
  Input Delay:            3.750ns
  Output Delay:           3.750ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

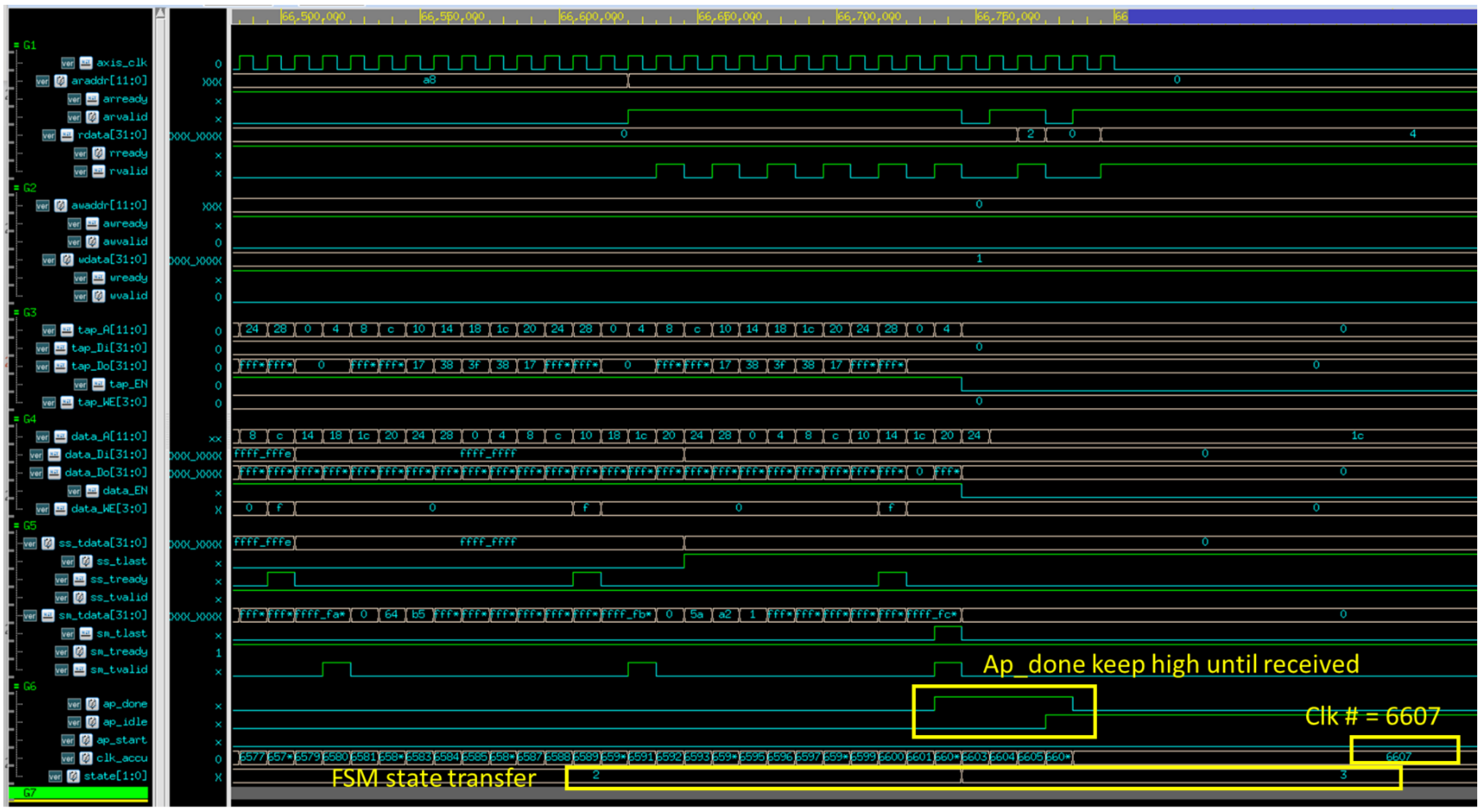
```
    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         (clock axis_clk rise edge)
                                                      0.000     0.000 r
                         input delay                  3.750     3.750
                                                      0.000     3.750 r  araddr[7] (IN)
                         net (fo=0)                   0.000     3.750    araddr[7]
                                                                      r  araddr_IBUF[7]_inst/I
                         IBUF (Prop_ibuf_I_O)         0.972     4.722 r  araddr_IBUF[7]_inst/O
                         net (fo=34, unplaced)        0.800     5.521    araddr_IBUF[7]
                                                                      r  out_rvalid_reg_i_1/I2
                         LUT3 (Prop_lut3_I2_O)        0.124     5.645 r  out_rvalid_reg_i_1/O
                         net (fo=12, unplaced)        0.497     6.142    out_rvalid_pre
                                                                      r  tap_A_OBUF[5]_inst_i_1/I3
                         LUT6 (Prop_lut6_I3_O)        0.124     6.266 r  tap_A_OBUF[5]_inst_i_1/O
                         net (fo=2, unplaced)         0.913     7.179    tap_A_OBUF[5]
                                                                      r  data_A_OBUF[5]_inst_i_1/I0
                         LUT6 (Prop_lut6_I0_O)        0.124     7.303 r  data_A_OBUF[5]_inst_i_1/O
                         net (fo=1, unplaced)         0.800     8.103    data_A_OBUF[5]
                                                                      r  data_A_OBUF[5]_inst/I
                         OBUF (Prop_obuf_I_O)         2.634    10.738 r  data_A_OBUF[5]_inst/O
                         net (fo=0)                   0.000    10.738    data_A[5]
                                                                      r  data_A[5] (OUT)
  -------------------------------------------------------------------    -------------------

                         (clock axis_clk rise edge)
                                                     15.000    15.000 r
                         clock pessimism              0.000    15.000
                         clock uncertainty           -0.035    14.965
                         output delay                -3.750    11.215
  -------------------------------------------------------------------
                         required time                         11.215
                         arrival time                         -10.738
  -------------------------------------------------------------------
                         slack                                  0.477
```

V. Simulation waveform




Program
ap_start
Coefficient read out from tap RAM and send to axi_lite
ap_start:
state transfer
started to calculate clk #


2. Data read out and store new data
3. Read in new data x from axi_stream
4. Send the result y to axi_stream
1. Coefficient read out
2.
3.
4.
axis_clk
araddr[11:0]
arready
arvalid
rdata[31:0]
rready
rvalid
awaddr[11:0]
awready
awvalid
wdata[31:0]
wready
wvalid
tap_A[11:0]
tap_Di[31:0]
tap_Do[31:0]
tap_EN
tap_WE[3:0]
data_A[11:0]
data_Di[31:0]
data_Do[31:0]
data_EN
data_WE[3:0]
ss_tdata[31:0]
ss_tlast
ss_tready
ss_tvalid
sm_tdata[31:0]
sm_tlast
sm_tready
sm_tvalid
ap_done
ap_idle
ap_start
clk_accu
state[1:0]


Ap_done keep high until received
Clk # = 6607
FSM state transfer
6607
2
3