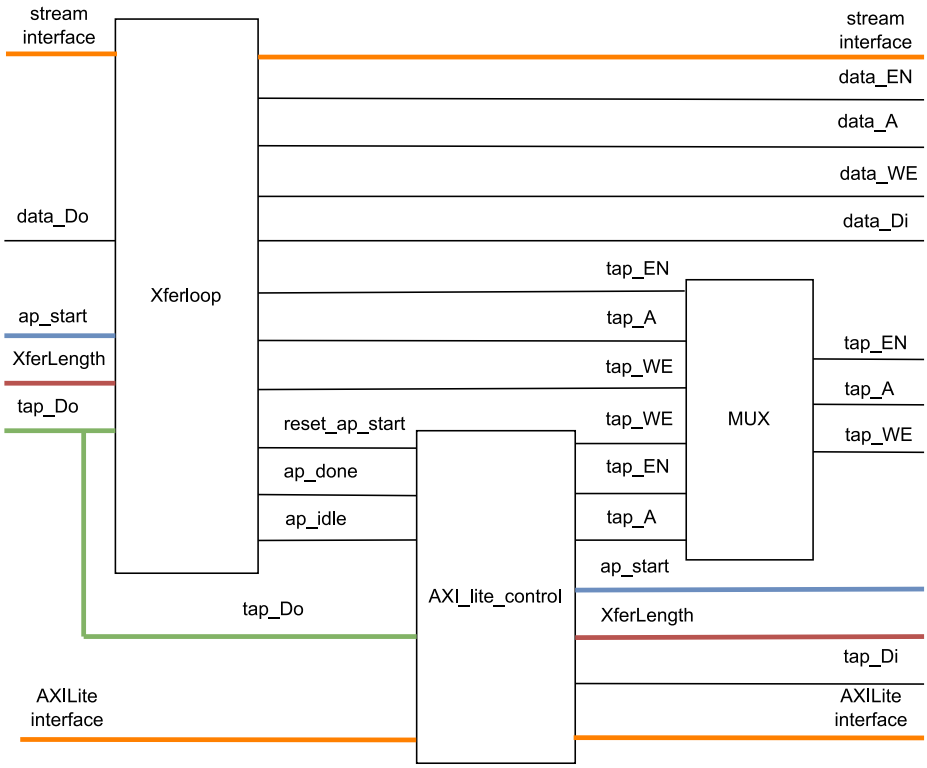
SOC Design Lab Lab3 Report

R11943025
謝郁楷

1. Block Diagram.

- **Overview of the design:**

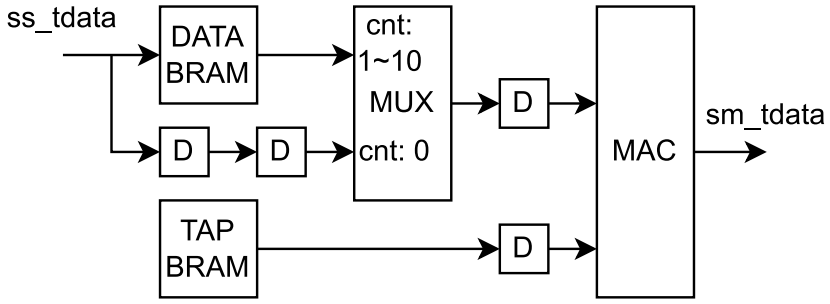


My design can be divided into two parts: Xferloop module and axi_lite_control module. Xferloop module includes the stream interface and the MAC computation unit, while axi_lite_control module contains all the control signals for AXI Lite and access to the control registers.

Control registers that store the tap values is implemented using BRAM this time. Both axi_lite_control module and Xferloop module will need to access this range of control registers, with one performing write operations and the other performing read operations. Therefore, you can see in the block diagram that there is a mux used to make the selection between the two.

In the diagram, you can see the output 'ap_start' of the axi_lite_control module. This signal is used to notify the Xferloop module to start the computation. The Xferloop module's output 'reset_ap_start' is used to reset 'ap_start' when the first data handshake occurs in the stream. Besides, the output ‘XferLength’ of axi_lite_control module is used to tell Xferloop module the length of the input stream. Xferloop also has two outputs, 'ap_done' and 'ap_idle,' which are used to instruct 'axi_lite_control' to set these two states in control register.

- **Xferloop:**



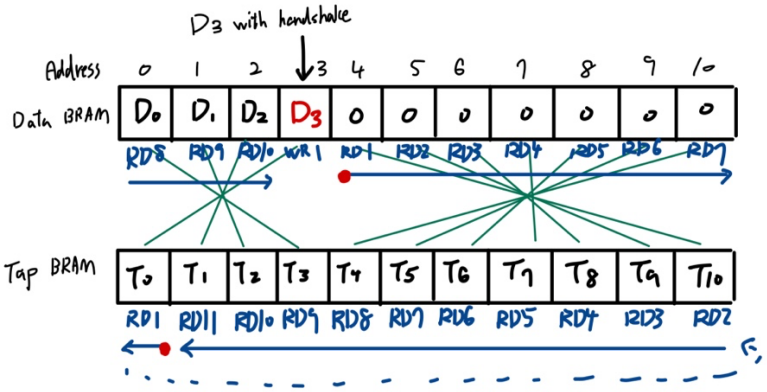
The data path in Xferloop, as shown in the above diagram, has two branches for the 'ss_tdata' input. One branch enters the BRAM for storage, during which the 'wen' (write enable) of the BRAM is set to 1. The other branch requires a delay of two cycles. This is because in my design, it takes two cycles from setting 'wen' to 0 until the corresponding data is read out. This is done to ensure that data access for MAC operation is continuous, and the BRAM needs to maintain to do a “Write” then do 10 “Read”. When there is no stuck due to handshake, it needs 11 cycles per output calculation.

The calculation of the read address for the BRAMs is determined by a counter that counts from one to eleven. This counter halts when there is no handshake on the output, ensuring that the circuit does not continue to calculate the address. Similarly, the MAC also incorporates this mechanism, where the accumulator register's value will not be updated when the output is not being sent out.

2. Describe operation.

- **How to access shift BRAM and tap BRAM to do computation.**

The diagram below illustrates the read and write operations of two BRAMs during computation, with both BRAMs having a size of 11.



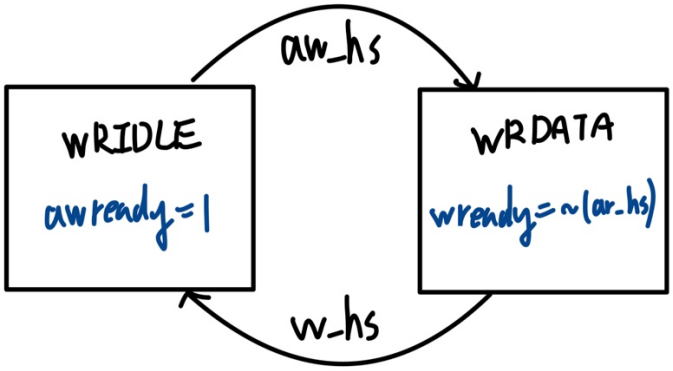
a. Data BRAM.

In the data BRAM, you can observe that three data points, D0, D1, and D2, are already stored. Currently, the fourth data point, D3, has just initiated a handshake for storage and is prepared to be written to address 3. After this write operation, it will be followed by reading from addresses 4 to 10 and 0 to 2, with address 3 as the starting point. Additionally, to address the padding issue that arises with the first ten data points, I write 0 to the BRAM for the initial 11 cycles before starting the MAC operations.

b. Tap Bram.

In the tap BRAM, the operation follows a fixed pattern, reading values in a circular back-and-forth manner from T0 to T10 and T9 to T1 and repeating this each output generation. The green-colored connections in the diagram represent data points that will undergo multiplication.

- **How to receive and store tap parameters and control signals.**

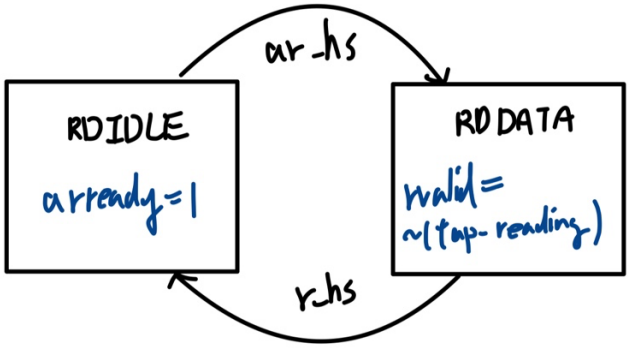


The diagram above is the finite state machine in axi_lite_control for the write operation. Initially, the state is in 'WRIDLE,' and as long as it remains in this state, 'awready' is directly set to 1. When the 'aw' handshake occurs, it transitions to the 'WRDATA' state. In most cases, 'wready' is immediately set to 1 in this state. However, when an 'ar' handshake also occurs simultaneously, to avoid simultaneous read and write access to the BRAM, priority is given to the read operation.

In the 'WRDATA' state, we can determine whether data should be sent to BRAM based on the 'aw_address.' For control signals' positions like 'ap_start' and 'XferLength,' since they are both stored using registers, there's no need to access BRAM. So, they are directly written to a register for storage. However, if the data is intended to be written to BRAM, e.g. Tap parameters, in this case, 'aw_address' is subtracted by a base address and then sent to BRAM with corresponding wdata for storage.

- **How to read tap Bram.**

The situation for reading from the tap BRAM can be divided into two scenarios. One scenario is when an external request is made through the AXI Lite interface to retrieve the values of tap parameters. The other scenario is when the Xferloop module needs to read during its computations. Let me first explain how the implementation works for the first scenario.



The diagram above is the finite state machine in axi_lite_control for the read operation. The implementation of the FSM for reading from the AXI Lite interface is quite similar to the write part and can also be divided into two states: RDIDLE and RDDATA. In the RDIDLE state, 'arready' is set to 1, indicating readiness to receive the address for reading at any time. Upon 'ar' handshake, it transitions to the RDDATA state.

Within RDDATA, there are two situations. One is for reading control registers like ap_start, ap_done, or XferLength, and the other is for reading tap parameters. The second scenario involves accessing the BRAM and therefore requires more cycles. As a result, during tap parameter reading, 'valid' is not raised immediately. It's only raised once the data has send back from the tap BRAM, indicating the completion of a read operation.

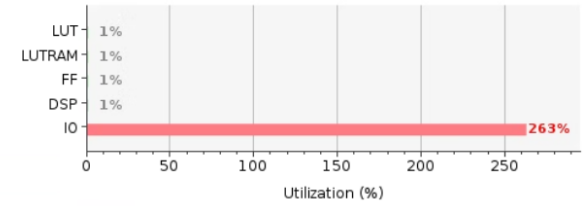
As for the part that Xferloop reads from the tap BRAM, it's relatively straightforward because the addresses to be read follow a simple order. It reads addresses in the order of 0, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1. The reason for this specific order can be referenced in the earlier explanation of how Xferloop computation operates. In fact, my IP does not handle situations where both Xferloop and axi_lite_control access to the BRAM simultaneously. Instead, it prioritizes granting access rights to the Xferloop module during output computations.

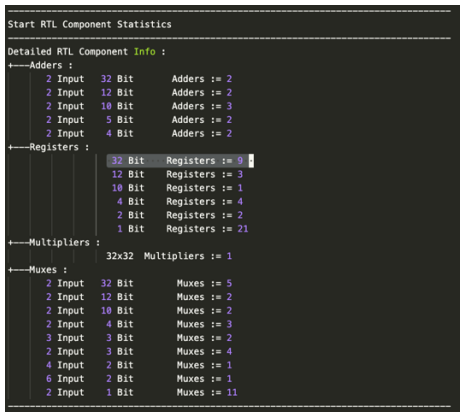
- **How ap_done and ap_idle is generated.**

Because the Xferoop module includes an input, XferLength, I use a counter to directly count the number of output handshakes until it reaches XferLength. When XferLength is reached, 'ap_done' is asserted. 'ap_done' is subsequently deasserted during the next read operation of control register address 0, ensuring that the testbench has detected 'ap_done' correctly. As for 'ap_idle,' it is asserted together with 'ap_done' but is not deasserted subsequently.

3. Resource Usage.

Resource	Utilization	Available	Utilization %
LUT	274	53200	0.52
LUTRAM	1	17400	0.01
FF	330	106400	0.31
DSP	3	220	1.36
IO	329	125	263.20





The results presented here were obtained with a clock period of 10ns and input-output delay of 1.2ns. The top diagram shows a summary of utilization. Regarding the number of Flip-Flops (FF) used, I noticed that my classmate's utilization is around 130, whereas mine is 330. We later compared and found that the difference is primarily due to the fact that I used 32-bit registers in the data path for tap parameters and input data (especially BRAM data IO). In essence, these 32-bit registers added substantial usage. When we consider the utilization without these 32-bit registers, the overall usage is similar between us. I also benefitted from using these registers. Under the same synthesis conditions, my classmate's design could only achieve a maximum clock period of 17ns, while mine was able to run at 10ns

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

The figure above show the BRAM usage is 0, since we use external behavior model of BRAM which is not in the RTL design.

4. Timing Report.

- The design is synthesis with clock period 10ns, input delay 1.2ns, output delay 1.2ns. The timing summary is attached below.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.541 ns	Worst Hold Slack (WHS):	0.078 ns	Worst Pulse Width Slack (WPWS):	4.020 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	929	Total Number of Endpoints:	929	Total Number of Endpoints:	332

All user specified timing constraints are met.

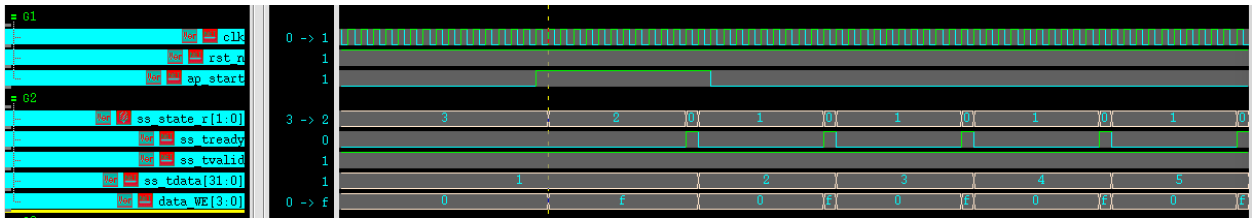
- The max delay path with slack=0.541.

```
Max Delay Paths
--------------------------------------------------------------------------------
Slack (MET) :             0.541ns  (required time - arrival time)
  Source:                 genblk1.Xferloop_U0/mul__0/CLK
                            (rising edge-triggered cell DSP48E1 clocked by axis_clk  {rise@0.000ns fall@5.000ns period=10.000ns})
  Destination:            genblk1.Xferloop_U0/acc_reg[29]/D
                            (rising edge-triggered cell FDRE clocked by axis_clk  {rise@0.000ns fall@5.000ns period=10.000ns})
  Path Group:             axis_clk
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            10.000ns  (axis_clk rise@10.000ns - axis_clk rise@0.000ns)
  Data Path Delay:        9.355ns  (logic 7.849ns (83.904%)  route 1.506ns (16.096%))
  Logic Levels:           8  (CARRY4=5 DSP48E1=1 LUT2=1 LUT3=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.128ns = ( 12.128 - 10.000 )
    Source Clock Delay      (SCD):    2.456ns
    Clock Pessimism Removal (CPR):    0.184ns
  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
                                                    0.000     0.000 r  axis_clk (IN)
                         net (fo=0)                 0.000     0.000    axis_clk
                                                                     r  axis_clk_IBUF_inst/I
                         IBUF (Prop_ibuf_I_O)       0.972     0.972 r  axis_clk_IBUF_inst/O
                         net (fo=1, unplaced)       0.800     1.771    axis_clk_IBUF
                                                                     r  axis_clk_IBUF_BUFG_inst/I
                         BUFG (Prop_bufg_I_O)       0.101     1.872 r  axis_clk_IBUF_BUFG_inst/O
                         net (fo=334, unplaced)     0.584     2.456    genblk1.Xferloop_U0/axis_clk_IBUF_BUFG
                         DSP48E1                                     r  genblk1.Xferloop_U0/mul__0/CLK
  -------------------------------------------------------------------    -------------------
```

```
                         DSP48E1 (Prop_dsp48e1_CLK_PCOUT[47])
                                                      4.206     6.662 r  genblk1.Xferloop_U0/mul__0/PCOUT[47]
                         net (fo=1, unplaced)         0.055     6.717    genblk1.Xferloop_U0/mul__0_n_106
                                                                 r  genblk1.Xferloop_U0/mul__1/PCIN[47]
                         DSP48E1 (Prop_dsp48e1_PCIN[47]_P[0])
                                                      1.518     8.235 r  genblk1.Xferloop_U0/mul__1/P[0]
                         net (fo=2, unplaced)         0.800     9.035    genblk1.Xferloop_U0/mul__1_n_105
                                                                 r  genblk1.Xferloop_U0/mul_carry_i_3/I0
                         LUT2 (Prop_lut2_I0_O)        0.124     9.159 r  genblk1.Xferloop_U0/mul_carry_i_3/O
                         net (fo=1, unplaced)         0.000     9.159    genblk1.Xferloop_U0/mul_carry_i_3_n_0
                                                                 r  genblk1.Xferloop_U0/mul_carry/S[1]
                         CARRY4 (Prop_carry4_S[1]_CO[3])
                                                      0.533     9.692 r  genblk1.Xferloop_U0/mul_carry/CO[3]
                         net (fo=1, unplaced)         0.009     9.701    genblk1.Xferloop_U0/mul_carry_n_0
                                                                 r  genblk1.Xferloop_U0/mul_carry__0/CI
                         CARRY4 (Prop_carry4_CI_CO[3])
                                                      0.117     9.818 r  genblk1.Xferloop_U0/mul_carry__0/CO[3]
                         net (fo=1, unplaced)         0.000     9.818    genblk1.Xferloop_U0/mul_carry__0_n_0
                                                                 r  genblk1.Xferloop_U0/mul_carry__1/CI
                         CARRY4 (Prop_carry4_CI_O[3])
                                                      0.331    10.149 r  genblk1.Xferloop_U0/mul_carry__1/O[3]
                         net (fo=4, unplaced)         0.642    10.791    genblk1.Xferloop_U0/mul__3[27]
                                                                 r  genblk1.Xferloop_U0/acc[24]_i_2/I0
                         LUT3 (Prop_lut3_I0_O)        0.307    11.098 r  genblk1.Xferloop_U0/acc[24]_i_2/O
                         net (fo=1, unplaced)         0.000    11.098    genblk1.Xferloop_U0/acc[24]_i_2_n_0
                                                                 r  genblk1.Xferloop_U0/acc_reg[24]_i_1/S[3]
                         CARRY4 (Prop_carry4_S[3]_CO[3])
                                                      0.376    11.474 r  genblk1.Xferloop_U0/acc_reg[24]_i_1/CO[3]
                         net (fo=1, unplaced)         0.000    11.474    genblk1.Xferloop_U0/acc_reg[24]_i_1_n_0
                                                                 r  genblk1.Xferloop_U0/acc_reg[28]_i_1/CI
                         CARRY4 (Prop_carry4_CI_O[1])
                                                      0.337    11.811 r  genblk1.Xferloop_U0/acc_reg[28]_i_1/O[1]
                         net (fo=1, unplaced)         0.000    11.811    genblk1.Xferloop_U0/acc_reg[28]_i_1_n_6
                         FDRE                                    r  genblk1.Xferloop_U0/acc_reg[29]/D
  ---------------------------------------------------------------    -------------------

                         (clock axis_clk rise edge)
                                                     10.000    10.000 r
                                                      0.000    10.000 r  axis_clk (IN)
                         net (fo=0)                   0.000    10.000    axis_clk
                                                                 r  axis_clk_IBUF_inst/I
                         IBUF (Prop_ibuf_I_O)         0.838    10.838 r  axis_clk_IBUF_inst/O
                         net (fo=1, unplaced)         0.760    11.598    axis_clk_IBUF
                                                                 r  axis_clk_IBUF_BUFG_inst/I
                         BUFG (Prop_bufg_I_O)         0.091    11.689 r  axis_clk_IBUF_BUFG_inst/O
                         net (fo=334, unplaced)       0.439    12.128    genblk1.Xferloop_U0/axis_clk_IBUF_BUFG
                         FDRE                                    r  genblk1.Xferloop_U0/acc_reg[29]/C
                         clock pessimism              0.184    12.311
                         clock uncertainty           -0.035    12.276
                         FDRE (Setup_fdre_C_D)        0.076    12.352    genblk1.Xferloop_U0/acc_reg[29]
  -------------------------------------------------------------------
                         required time                         12.352
                         arrival time                         -11.811
  -------------------------------------------------------------------
                         slack                                 0.541
```

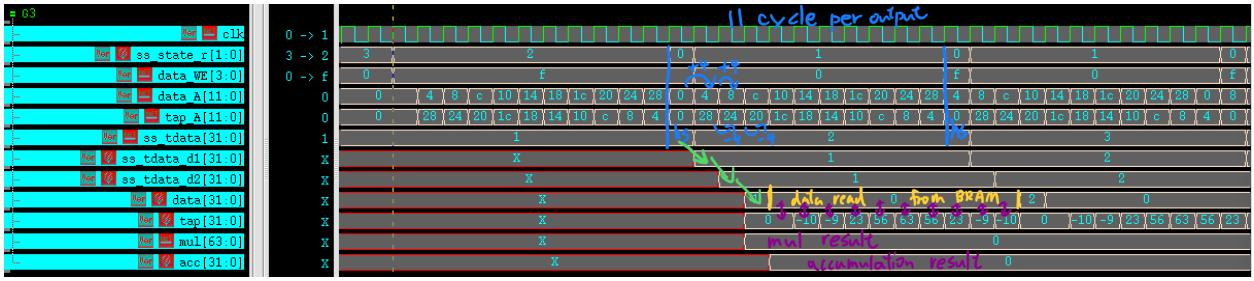
5. Simulation Waveform

- **Xferloop data-in stream waveform.**



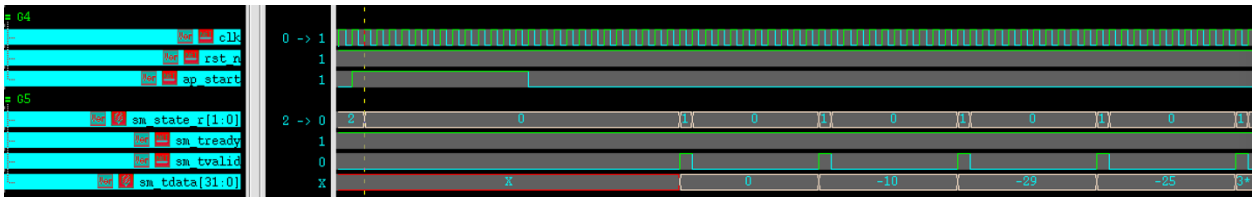
This chart illustrates that 'ap_start' is deasserted after the first handshake. Additionally, you can observe the read FSM (signal ss_state_r). State 3 is the reset state, and upon encountering 'ap_start,' it transitions to state 2. In state 2, the data BRAM is reset to all zeros for zero padding, and then it alternates between states 0 and 1. In state 0, the 'ss_tdata' is stored in the BRAM upon handshake, and in state 1, the output computational takes place.

- **Xferloop calculation waveform.**



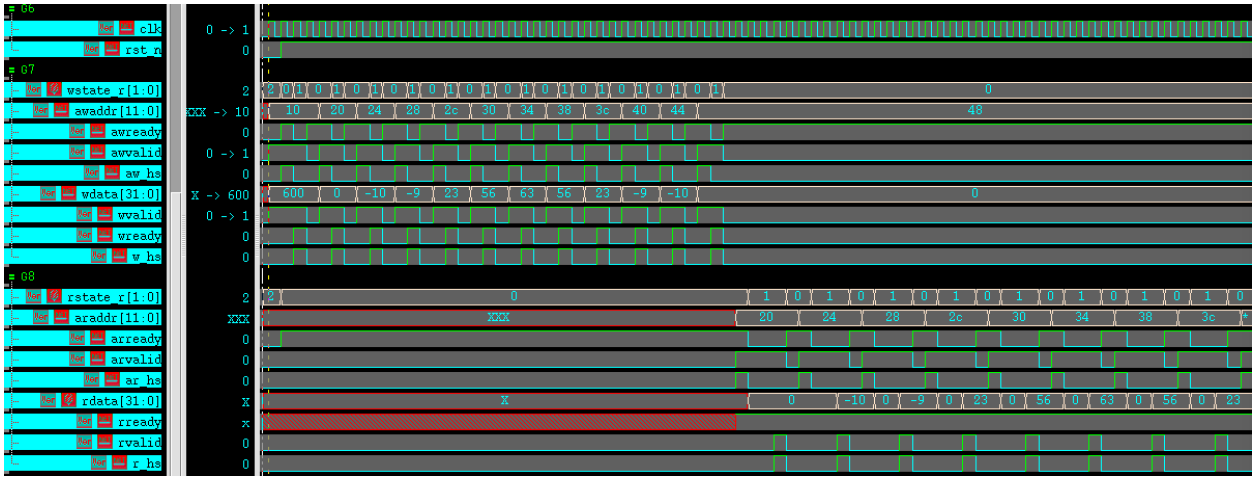
Based on the figure above, it can be observed that for each output, it requires at least 11 cycles. Additionally, the data address settings align with the description in Part 2. Furthermore, 'ss_tdata' is delayed by two cycles to align with the data read from the data DRAM, and then multiplication and accumulation are performed.

- **Xferloop data-out stream waveform.**



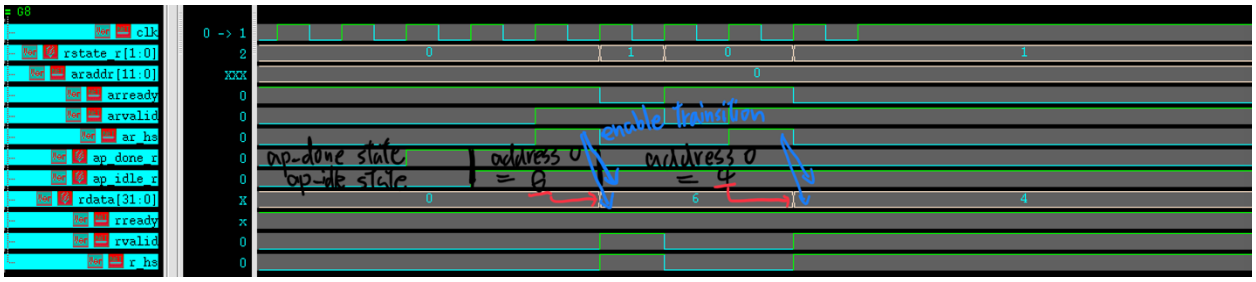
It can be seen that the write FSM alternates between states 0 and 1. In state 0, data is being processed, and in state 1, 'valid' is asserted, and the results are sent out.

- **axi_lite_control read write waveform:**



First, looking at the write part, the FSM here alternates between states 0 and 1. In state 0, 'awready' is asserted and it waits for 'waddress.' In state 1, 'wready' is asserted, and the write operation is performed. The read part operates similarly, where in state 0, 'arready' is asserted to wait for 'read address,' and in state 1, 'rvalid' is asserted only after the data has been successfully retrieved from the BRAM.

- **axi_lite_control ap_done and ap_idle signals.**



The 'ap_done_r' and 'ap_idle_r' in the waveform represent the states stored in control register address 0. After the computation is completed, both of these states are set to 1. When a read operation occurs at address 0, 'ap_done_r' is deasserted, causing the value at address 0 to be updated from 6 to 4. At this point, 'r_data' is also updated to 6 and is awaiting the read handshake signal for transmission. Similarly, when a second read operation is performed at address 0, 'r_data' will have value 4.

The above figure shows the cycles between ap_start and ap_done. The cycle between them is around 11*600 cycles.