Bridge of Life Education

Soc Design

FIR Workbook (lab_3)

Revision

Revision 1-20-2025

- Add four steps (datapath, interface, testbench, optimization)

Revise: 7-8-2024

- Elaborate more on testbench function
- Write one to clear ap_done

Revise: 10-14-2024

- Add Multiplication and Addition are run in separate pipeline cycle
- Add # of tap
- Tap parameters start at 0x80

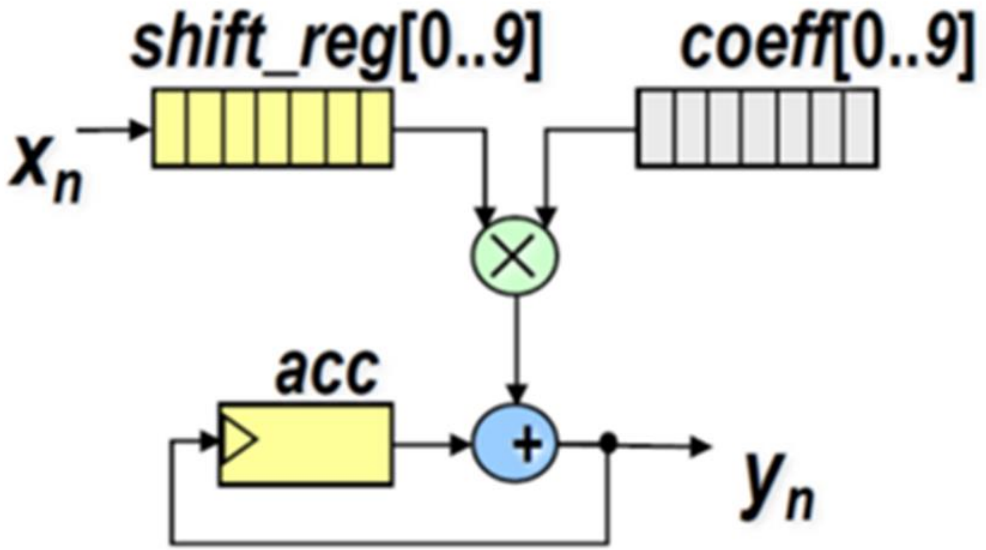
Function specification

- Same as course-lab_2(FIRN11stream)
  - https://github.com/bol-edu/course-lab_2
- y[t] = Σ (h[i] * x[t - i])

- Reference C-code
  - https://github.com/bol-edu/course-lab_2/blob/2022.1/hls_FIRN11Stream/FIR.cpp

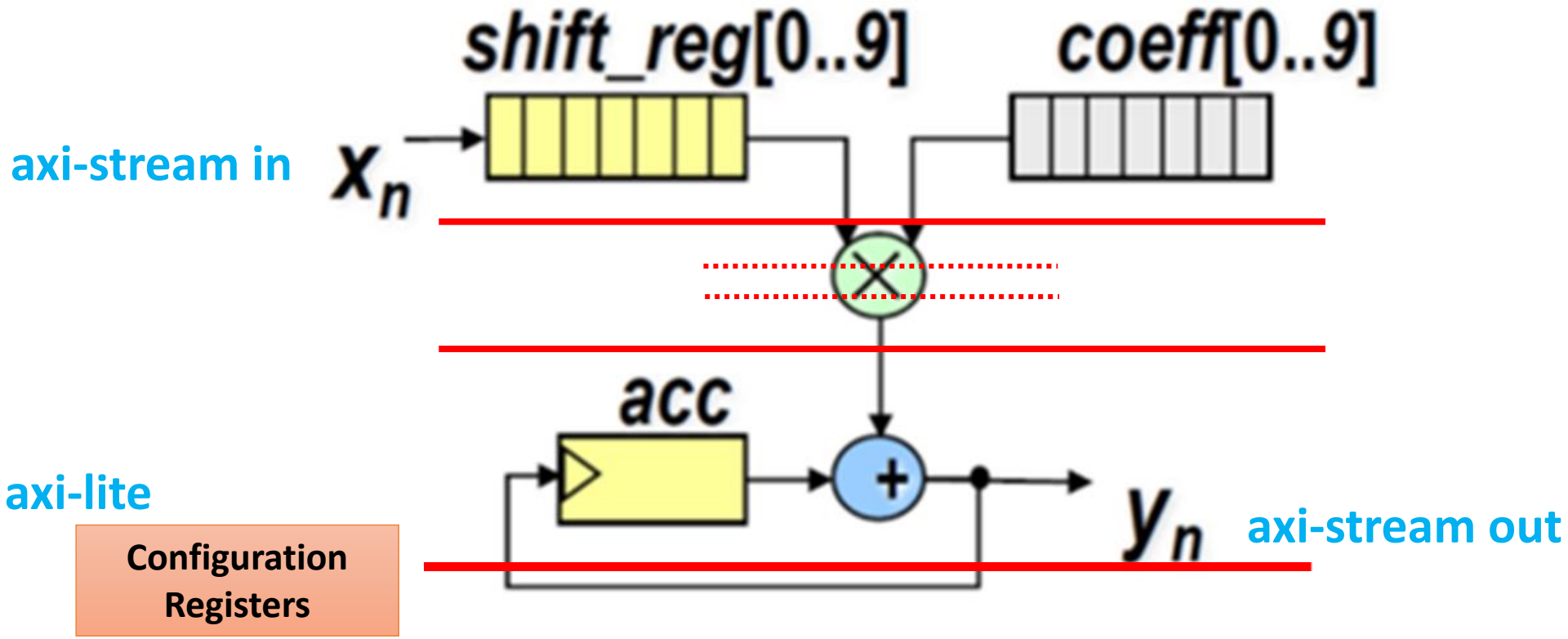
```
void fir_n11_strm(stream_t* pstrmInput, stream_t* pstrmOutput,
     int32_t an32Coef[MAP_ALIGN_4INT], reg32_t regXferLeng)
{
#pragma HLS INTERFACE s_axilite port=regXferLeng
#pragma HLS INTERFACE s_axilite port=an32Coef
#pragma HLS INTERFACE axis register both port=pstrmOutput
#pragma HLS INTERFACE axis register both port=pstrmInput
#pragma HLS INTERFACE s_axilite port=return
                    static int32_t an32ShiftReg[N];

   n32NumXfer4B = (regXferLeng + (sizeof(int32_t) - 1)) / sizeof(int32_t);
XFER_LOOP:
  for (n32XferCnt = 0; n32XferCnt < n32NumXfer4B; n32XferCnt++) {
      n32Acc = 0;
      value_t valTemp = pstrmInput->read();
      n32Temp = valTemp.data;
SHIFT_ACC_LOOP:
      for (n32Loop = N - 1; n32Loop >= 0; n32Loop--) {
         if (n32Loop == 0) {
               an32ShiftReg[0] = n32Temp;
               n32Data = n32Temp;
          } else {
                an32ShiftReg[n32Loop] = an32ShiftReg[n32Loop - 1];
                n32Data = an32ShiftReg[n32Loop];
          }
          n32Acc += n32Data * an32Coef[n32Loop];
      }
      valTemp.data = n32Acc;
      pstrmOutput->write(valTemp);
      if (valTemp.last) break;
  }
  return;
}
```

Design specification



- Data_Width 32bit (integer)
- Tap_Num 11
- Data_Num TBD – Based on the size of the data file
- Interface
  - **axilite**: configuration data_in:
  - **axi-stream**: stream (Xn), stream (Yn)
- **Using one Multiplier and one Adder:**
  - **Note: Multiplication and Addition are run in separate pipeline cycle**
  - **Note: Don’t use DSP, use Xilinx directive (* use_dsp = "no" *)**
- **Shift registers implemented with SRAM** (Shift_RAM, size = 10 DW)
- **Tap coefficient implemented with SRAM** (Tap_RAM = 11 DW), initialized by axilite write
- Operation (Refer to: Configuration Register Address map)
  - Program ap_start to initiate FIR engine
  - Stream in Xn. The rate is varied by testbench. axi-stream tvalid/tready for flow control.
  - Stream out Yn, the rate of output transfer is varied by testbench. axi-stream tvalid/tready for flow control.

Block Diagram

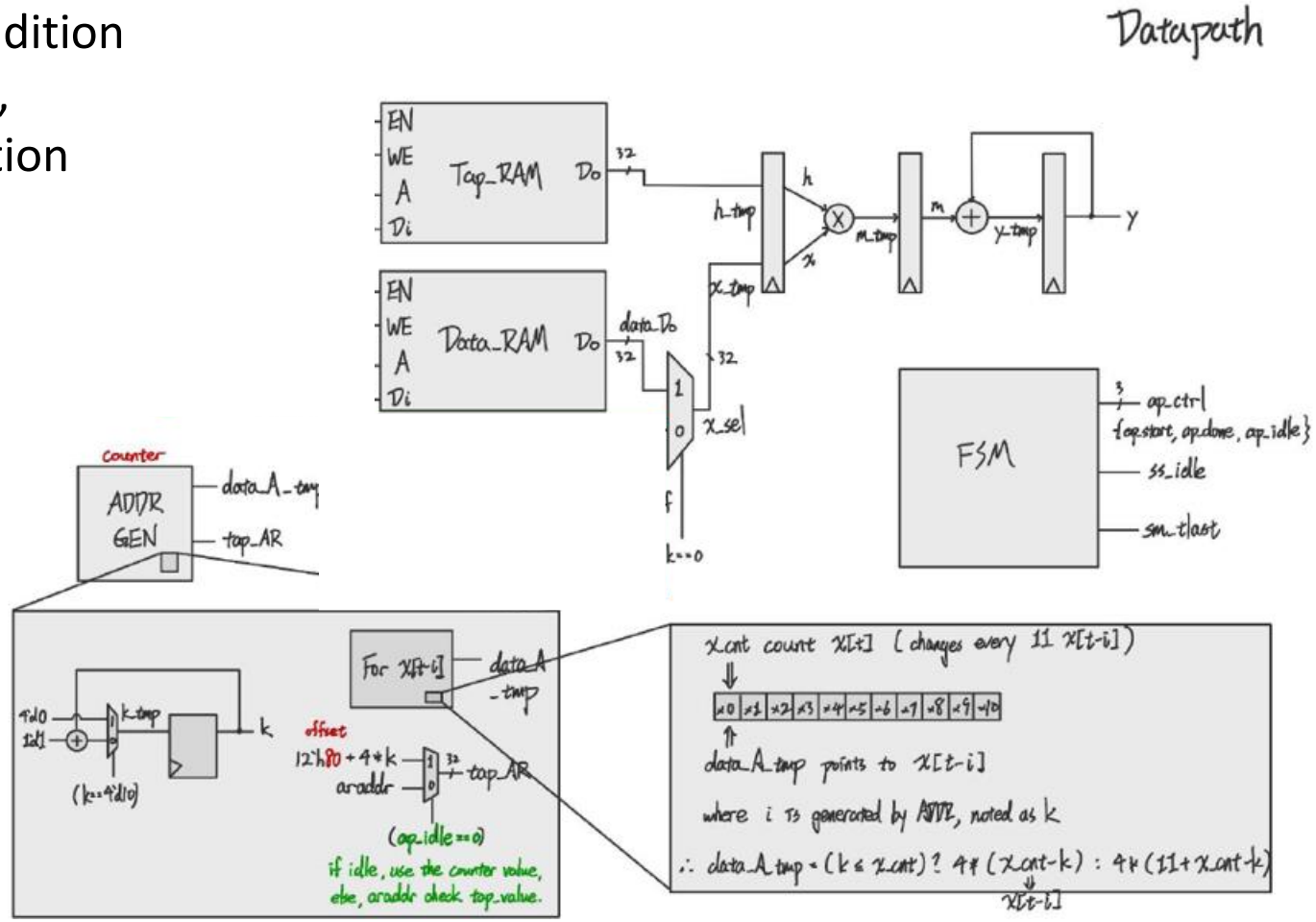


Step#1 – Implement the FIR compute Engine

Implement the FIR core Compute Engine

1. Datapath: RAM, MUX, Multiplication, Addition
2. FSM to control the datapath component,
3. RAM (FIFO) control and address generation

```
    n32NumXfer4B = (regXferLeng + (sizeof(int32_t) - 1)) / sizeof(int32_t);
XFER_LOOP:
        for (n32XferCnt = 0; n32XferCnt < n32NumXfer4B; n32XferCnt++) {
                n32Acc = 0;
                value_t valTemp = pstrmInput->read();
                n32Temp = valTemp.data;
SHIFT_ACC_LOOP:
                for (n32Loop = N - 1; n32Loop >= 0; n32Loop--) {
                        if (n32Loop == 0) {
                                an32ShiftReg[0] = n32Temp;
                                n32Data = n32Temp;
                        } else {
                                an32ShiftReg[n32Loop] = an32ShiftReg[n32Loop - 1];
                                n32Data = an32ShiftReg[n32Loop];
                        }
                        n32Acc += n32Data * an32Coef[n32Loop];
                }
                valTemp.data = n32Acc;
                pstrmOutput->write(valTemp);
                if (valTemp.last) break;
        }
```



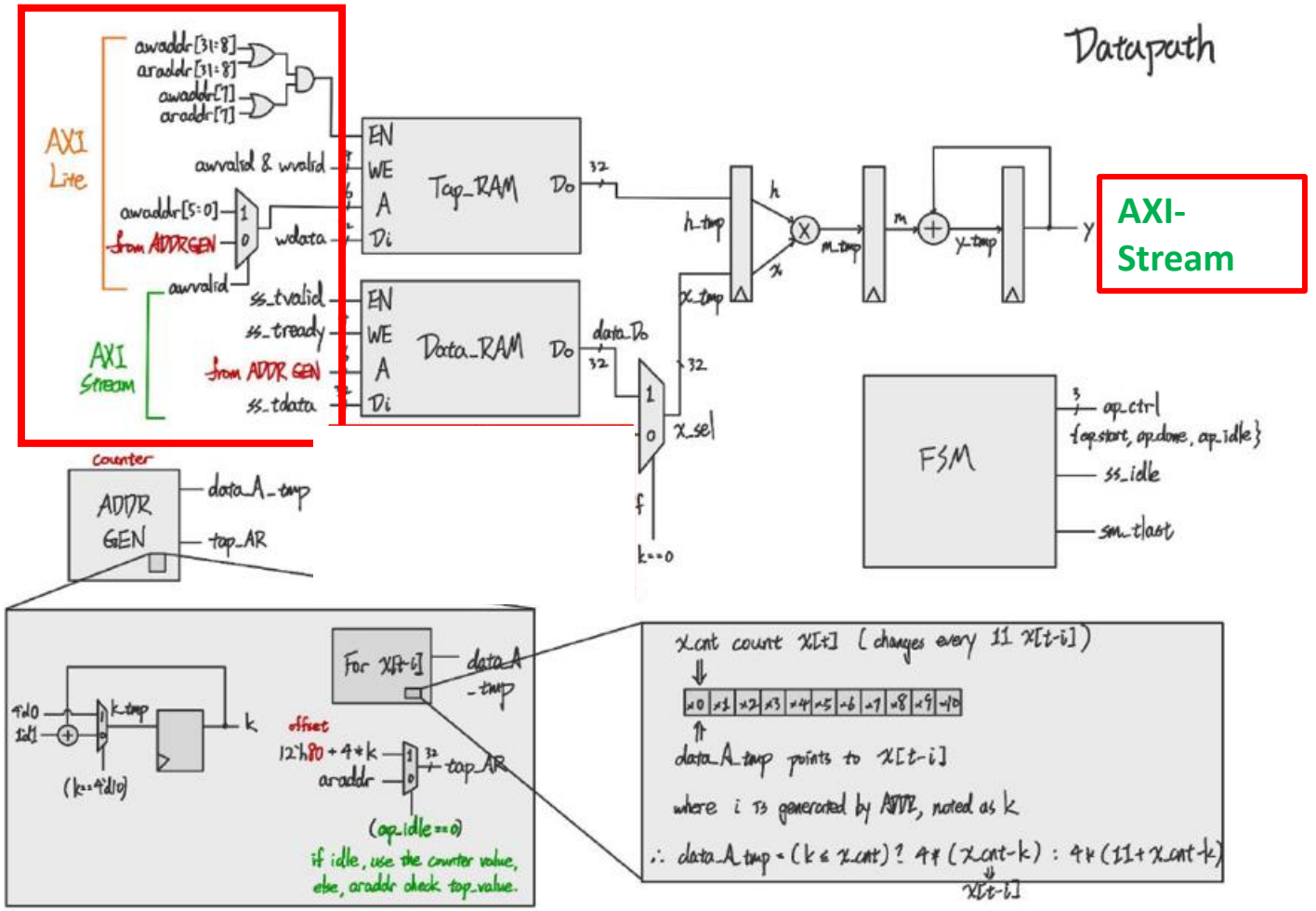
https://github.com/bol-edu/course-lab_2/blob/2022.1/hls_FIRN11Stream/FIR.cpp

Step#2 – Add the AXI Bus interface

Add AXI Interface

1. AXI-Lite for configuration registers, and TapRAM
2. AXI-Stream in for X[n]
3. AXI-Stream out for Y[n]

```
void fir_n11_strm(stream_t* pstrmInput, stream_t* pstrmOutput, i
{
#pragma HLS INTERFACE s_axilite port=regXferLeng
#pragma HLS INTERFACE s_axilite port=an32Coef
#pragma HLS INTERFACE axis register both port=pstrmOutput
#pragma HLS INTERFACE axis register both port=pstrmInput
#pragma HLS INTERFACE s_axilite port=return
```



Step#3 – Develop Testbench

- Programming sequence. Refer to Configuration Register Access Protocol
- Refer to TestBench Specification

Step#4 : Enhance Design Robustness and Performance

- With the testbench validation, the varied input/output latency will reveal the design weakness and the need for performance enhancement.
- After synthesizing the design and analyzing the timing, you will find a critical path and try to improve its critical timing.
- You will develop your idea to improve the above two areas.

- Those improvements will help in Lab#4 (HW-SW codesign)

Step#4 : Possible Improvement

- You will consider adding some buffer on the X-input and Y-output.
- To improve speed, you will consider adding a pipeline on Multiplier.
  - Using Vivado IP generator, and incorporate the IP into your design

You will implement
- fir.v (design)
- fir_tb.v (testbench)

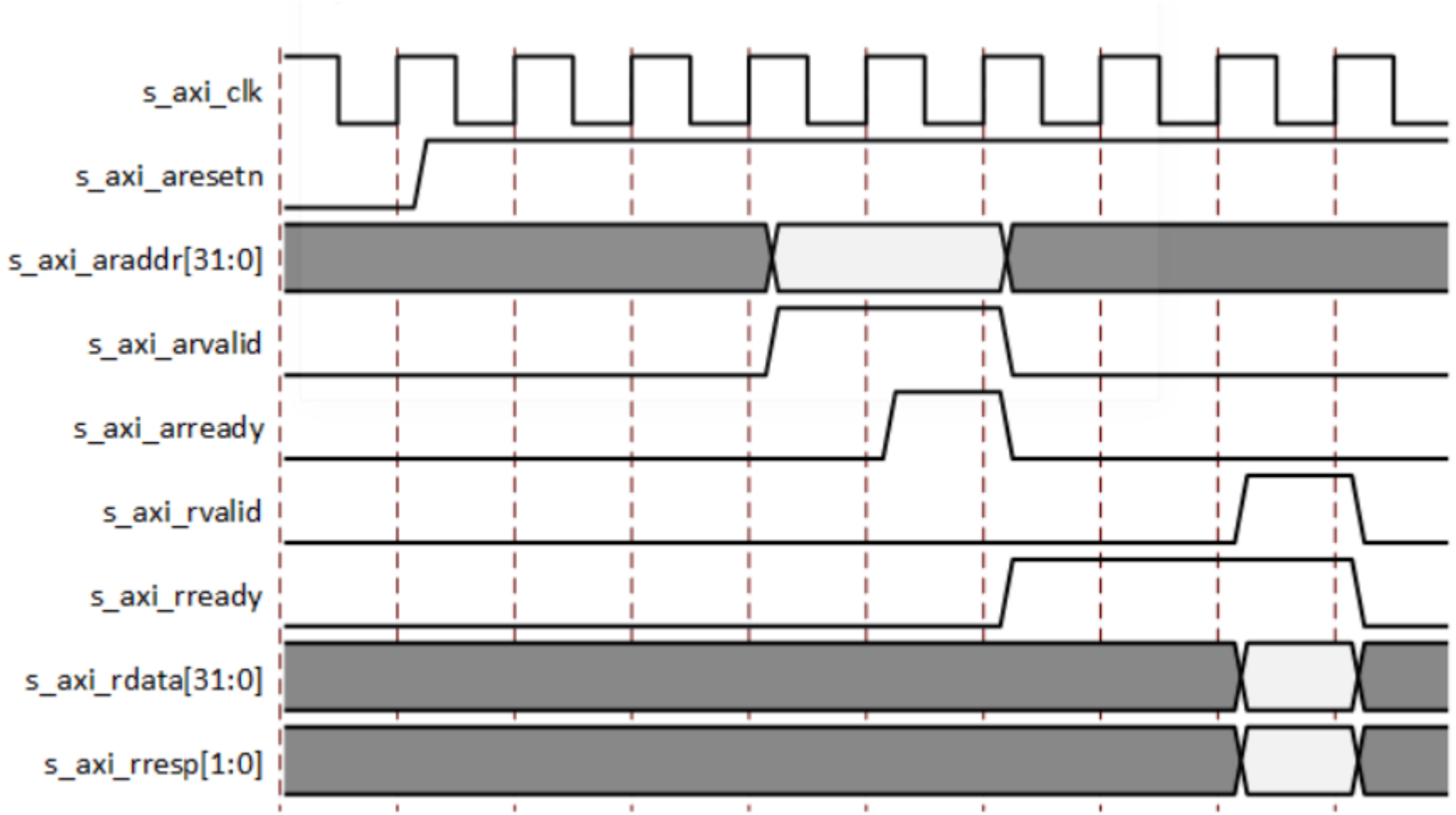
Lab Github:
https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/lab-fir

Background (AXI)
Refer to lecture: Interconnect - axi

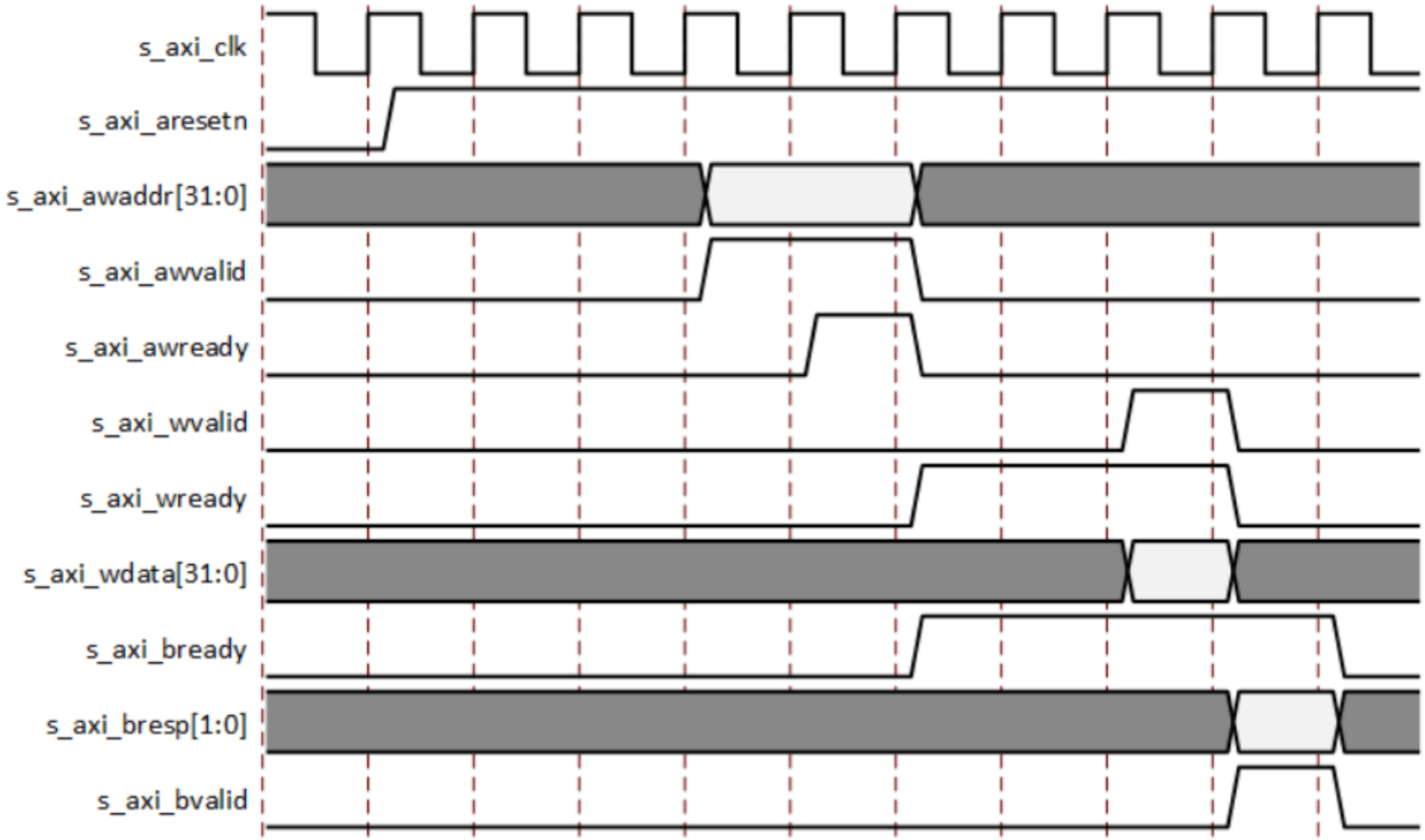
FIR module interface (AXI-Lite, AXI-Stream)

- AXI-lite:
  - https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081
  - https://docs.xilinx.com/r/en-US/pg202-mipi-dphy/AXI4-Lite-Interface
- AXI-stream:
  - https://developer.arm.com/documentation/ihi0051/latest/
  - https://docs.xilinx.com/r/en-US/pg256-sdfec-integrated-block/AXI4-Stream-Interface
- BRAM Interface: Synchronous read/write

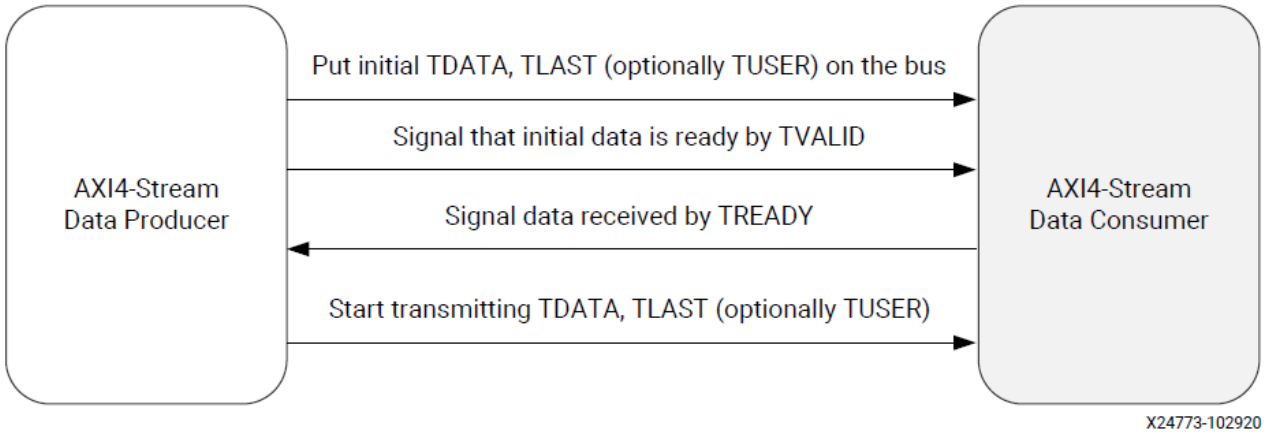
AXI4-Lite Read Transaction

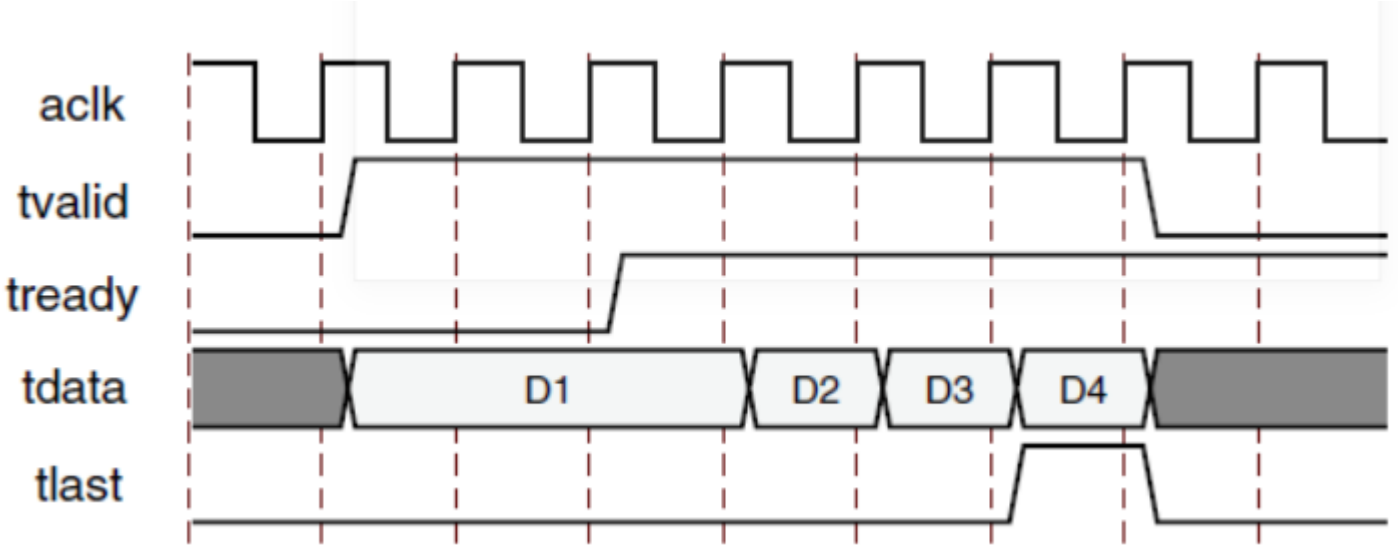


AXI4-Lite Write Transaction



AXI4-Stream Transfer Protocol

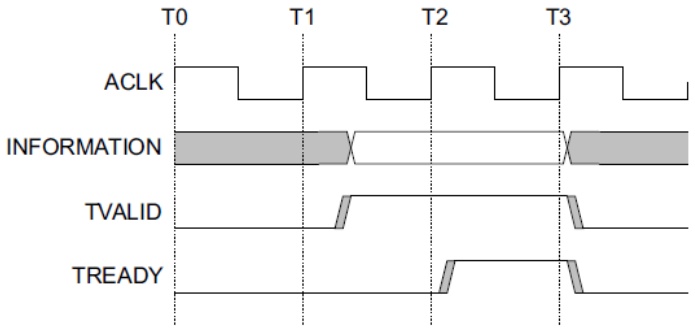




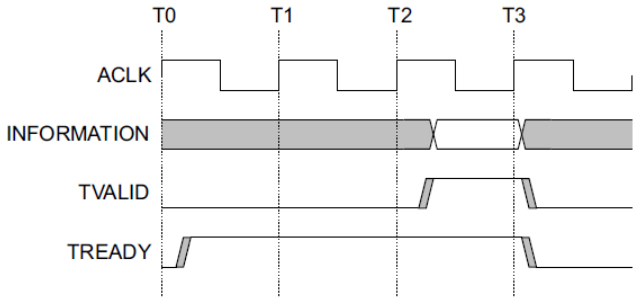
Data Transfer Handshake : TVALID, TREADY

- For a transfer to occur, both **TVALID** and **TREADY** must be asserted
- A Transmitter is not permitted to wait until **TREADY** is asserted before asserting **TVALID**
- Once **TVALID** is asserted, it must remain asserted until the handshake occurs
- A Receiver is permitted to wait for **TVALID** to be asserted before asserting **TREADY**

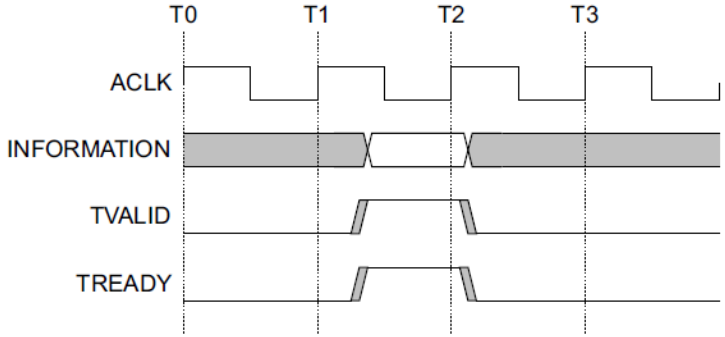
TVALID asserted before TREADY



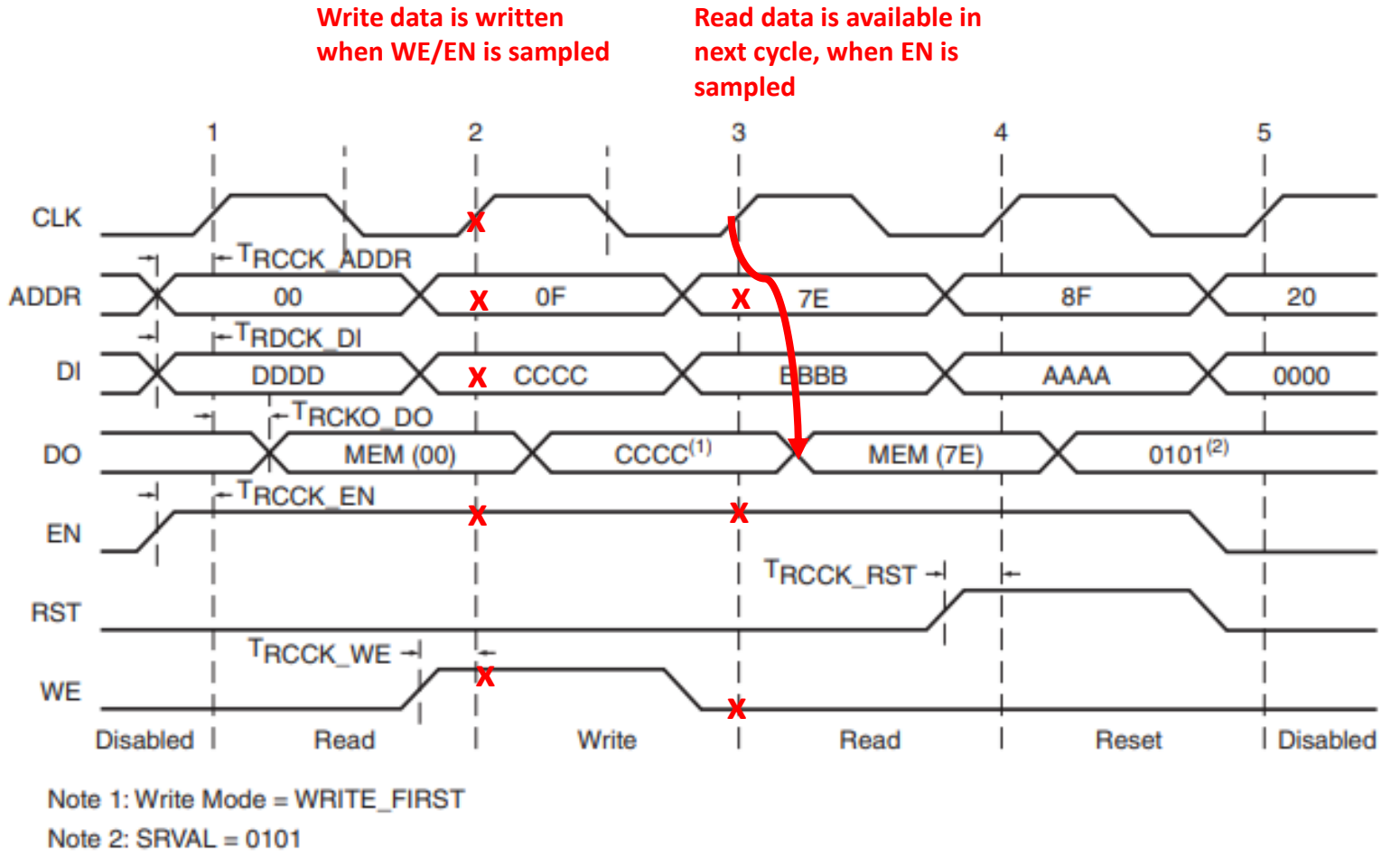
TREADY asserted before TVALID



TVALID and TREADY asserted simultaneously



SRAM Access Timing



SRAM access has different modes, refer to https://docs.xilinx.com/r/en-US/am007-versal-memory/Read-Operation?tocId=VRYu0HURA1U147fufYDMNQ

You are given the module header

- Use simplified AXI-lite and AXI-stream protocol, interface defined in the module.
- Use parameter for resource allocation.
- bram model

```
module fir
#(  parameter pADDR_WIDTH = 12,
    parameter pDATA_WIDTH = 32,
    parameter Tape_Num    = 11 )
)

(
begin

   // write your code here!


end
```

```
// axi-lite for configuration, tap-RAM
output  wire                     awready,
output  wire                     wready,
input   wire                     awvalid,
input   wire [(pADDR_WIDTH-1):0] awaddr,
input   wire                     wvalid,
input   wire [(pDATA_WIDTH-1):0] wdata,
output  wire                     arready,
input   wire                     rready,
input   wire                     arvalid,
input   wire [(pADDR_WIDTH-1):0] araddr,
output  wire                     rvalid,
output  wire [(pDATA_WIDTH-1):0] rdata,
```

```
// axi-stream slave for X[n] input – SS bus
input   wire                     ss_tvalid,
input   wire [(pDATA_WIDTH-1):0] ss_tdata,
input   wire                     ss_tlast,
output  wire                     ss_tready,
```

```
// axi-stream master for Y[n] output – SM bus
input   wire                     sm_tready,
output  wire                     sm_tvalid,
output  wire [(pDATA_WIDTH-1):0] sm_tdata,
output  wire                     sm_tlast,
```

```
// bram for tap RAM
output  wire [3:0]               tap_WE,
output  wire                     tap_EN,
output  wire [(pDATA_WIDTH-1):0] tap_Di,
output  wire [(pADDR_WIDTH-1):0] tap_A,
input   wire [(pDATA_WIDTH-1):0] tap_Do,
```

```
// bram for data RAM
output  wire [3:0]               data_WE,
output  wire                     data_EN,
output  wire [(pDATA_WIDTH-1):0] data_Di,
output  wire [(pADDR_WIDTH-1):0] data_A,
input   wire [(pDATA_WIDTH-1):0] data_Do,
```

https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/lab-fir/fir/rtl/fir.v

https://github.com/bol-edu/caravel-soc_fpga-lab/blob/main/lab-fir/bram/bram11.v

Configuration Register Access Protocol

Configuration Register Address map

0x00 –

[0] - **ap_start** (r/w) command

When ap_start is programmed one, the FIR engine starts.

[1] – **ap_done (rwc)** status

1: indicate FIR has processed all the dataset, i.e. receive last data X, and the last Y is transferred.

[2] – **ap_idle** (ro) status

1: indicate FIR is idle. 0: FIR is actively processing data

0x10-14 - data-length

0x14-18 - number of taps (Max: 32)

0x40-7F – Tap parameters, (e.g., 0x40-43 Tap0, 0x44-0x47 Tap1 .. in sequence …)

Note: Tap parameters set at 0x40 can directly use a lower address bit for the SRAM address.

0x80-7F

ap_start protocol and implementation

1. ap_start is a read/write registers
2. When ap_start is programmed one, the FIR engine starts.
3. Host Software or testbench can program ap_start
    1. When ap_idle is one.
    2. After data-length, tap parameters are programmed
    3. If ap_start is programmed one when ap_idle is zero (i.e. engine is running), the programming the ap_start is not effective, i.e. ignored
4. ap_start is set by software/testbench, and reset by engine
5. Engine resets ap_start when engine is not idle, i.e. engine starts processing data

    set by host program, when axilite write to 0x00[0]

    reset, when starting data transfer, i.e. 1st axi-stream data come in

ap_done protocol and implementation

1. It is a read/write-one-clear register (rwc)
2. ap_done is asserted when the engine completes the last data processing, and data is transferred
3. ap_done is reset in the following condition
    1. Reset signal is asserted
    2. After a task is complete, the ap_done is cleared by
        1. When ap_done is read, i.e. address 0 is read
        2. Write one to ap_done register bit to clear

        => Choose one implementation

ap_idle protocol and implementation

1. ap_idle is set to 1 when reset
2. ap_idle is set to 0 when ap_start is sampled, and the first data is sampled
3. ap_idle is set to 1 when the FIR engine processes the last data and last data is transferred

Handle Configuration read/write while the engine is active

- It is an illegal operation, but we must handle it.
- Configuration read:
  - AccessTapRAM read, you have two options
    1. Return ‘hffffffff (invalid value, software can check). Note: TapRAM is accessed by the engine when it is active.
    2. Return valid value. The design is more complicated. You can latch the request and access the TapRAM at the appropriate time.
  - Other address, return a valid value (it does not interfere the engine operation)
- Configuration write: Ignore it, i.e., drop the write transaction. It is illegal.

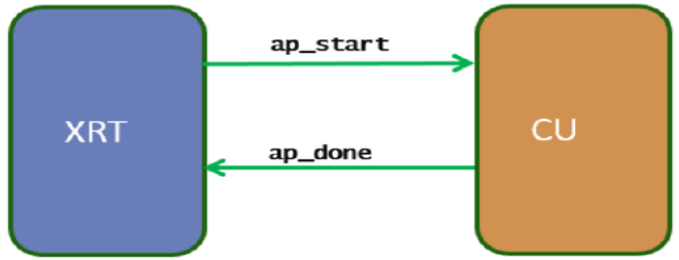
Testbench Specification

Testbench

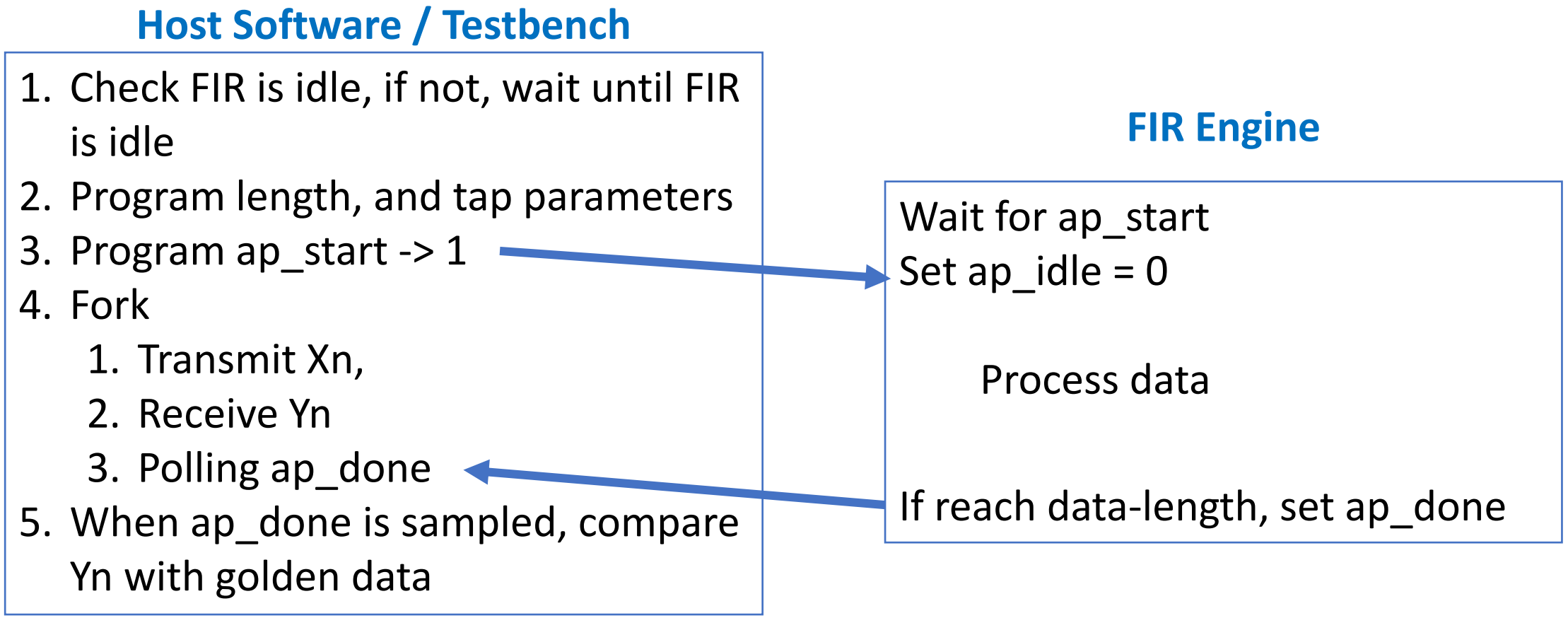
- fir_tb(Please name your top module same as **fir** for simulation)
- The testbench should keep in the same directory with Makefile
- Block-level Protocol (ap_start, ap_done):

AP_CTRL_HS (Sequential Executed Kernel)

- Host and Kernel Synchronization by
  - ap_start
  - ap_done
- Kernel can only be restarted (ap_start), after it completes the current execution (ap_done)
- Serving one execution request a time



Host software / Testbench Programming Sequence



Note: Transmit Xn (stream-in), Receive Yn (stream-out) and Polling ap_done (axilite) are running concurrently. They are using different interface and do not interfere each other

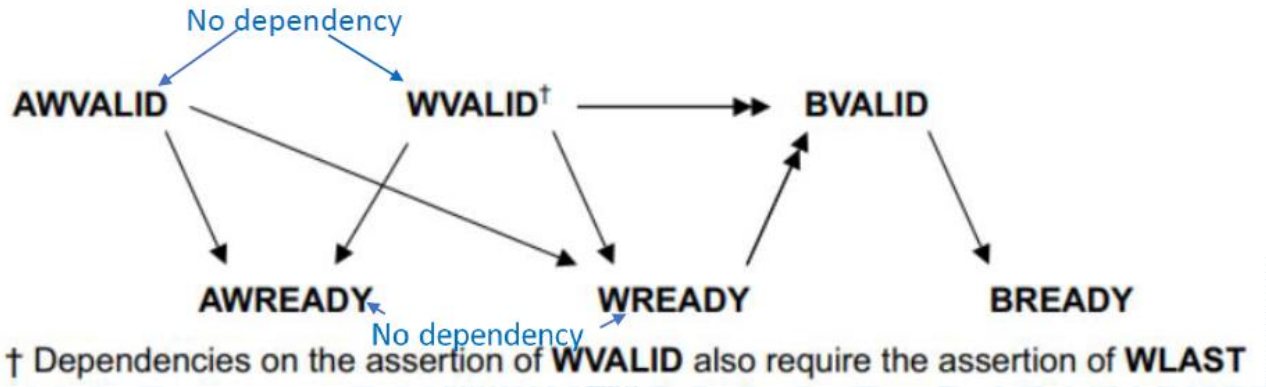
Testbench – Develop your own testbench

1. **Setup phase**
    1. Load datafile, and count # of data = data_length
    2. Program tap_parameters and data_length, read back and check it is correctly programmed
    3. Compute Yn expected value, or load golden data into Yn buffer
    4. Read and check ap_start, ap_idle, ap_done are in proper state
2. **Execution phase**
    1. Program ap_start
    2. Start latency timer
    3. Fork the following operations, run concurrently
        1. Task1(axis-in): Stream_in_Xn
        2. Task2(axis_out): Stream_out_Yn and save into Yn buffer
        3. Task3(axilite):
            1. Polling ap_done, when ap_done is sampled, disable tasks (stream_in_Xn, stream_out_Yn, and Polling)
            2. Read/write tap_parameters: make sure it does not corrupt fir computation
                1. Read return invalid value, e..g. 'hffffffff
                2. Write ignored
3. **Checking Phase**
    1. Report latency
    2. Compare Yn buffer with golden data
4. **Repeat 2 – 3 for three times**

Note: You may print message to assist debugging

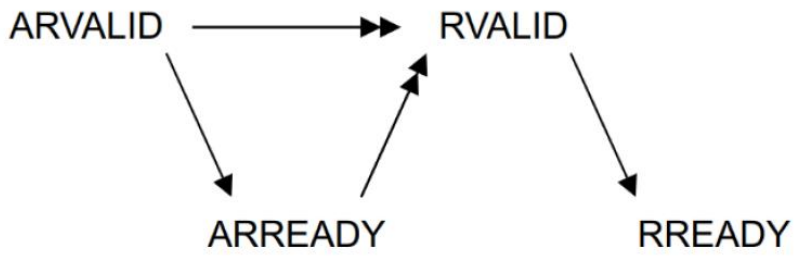
TestBench – Axilite write

- Axilite write
  - Spawn two processes: aw_proc (address) , w_proc (data)
  - Each process has random initial delay 0 – 5T (so address / data can be out-of-order)
    - Asserts awaddr (wdata) when awvalid (wvalid) asserts
    - Wait for awready (wready)
    - Once awready (wready) is sampled active**, invalidate awaddr (wdata)**
    - Optionally wait for bvalid (in our case, no bvalid – not implement)



TestBench – Axilite read

- Axilite read
  - Spawn two processes: ar_proc (address), r_proc (data)
  - Each process has a random initial delay 0 - 5T
    - Asserts araddr, arvalid and rready
    - Once arready is sampled active, **invalidate araddr**
  - Check rvalid is only asserted after arvalid and arready are asserted



TestBench – axi-stream X input

Protocol

- For a transfer to occur, both TVALID and TREADY must be asserted
- A Transmitter is not permitted to wait until TREADY is asserted before asserting TVALID
- Once TVALID is asserted, it must remain asserted until the handshake occurs
- A Receiver is permitted to wait for TVALID to be asserted before asserting TREADY

Testbench

- Randomize tvalid initial delay
  - Case1: short latency [0-5]
  - Case2: long latency [0-2 x Filter latency]
- Once sampled tready, deasserts tdata

Testbench: axi-stream Y output

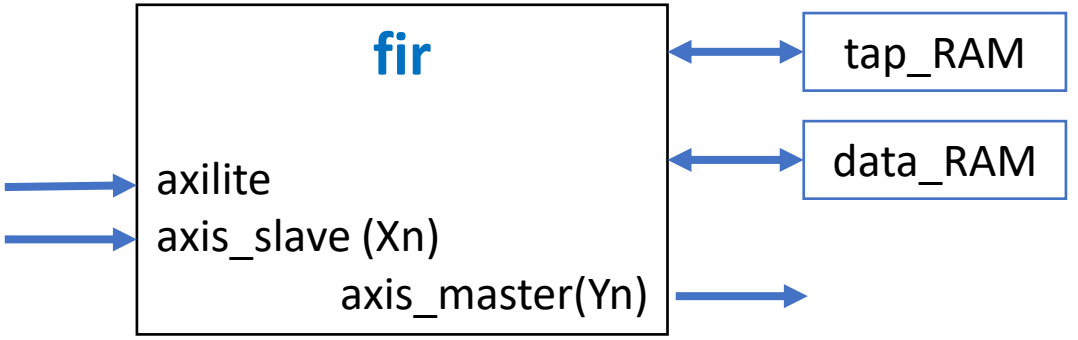
- spawn a task ->task_sm_tready
  - randomize sm_tready delay
    - Short latency: delay [0-5]
    - Long latency: delay [0- 2* filter latency]
  - when both sm_tready and sm_tavlid sampled asserted,
    - Check data
    - finish the task
- Note: sm_tready could be asserted before sm_tvalid asserted

Test dataset

- Samples_triangular_wave.dat
- out_gold.dat

SRAM Interface Implementation

- Refer to verilog-sram.pdf - Use memory for ASIC flow
- Github updated - https://github.com/bol-edu/caravel-soc_fpga-lab/tree/main/lab-fir
- Implement SRAM without .db/.lib
- Use external SRAM (bram.v). fir.v provides ports to interface with the external SRAM. So, the fir.v can be synthesized with BRAM
- Two size of bram.v (you choose either one to fit your design)
  - bram11.v (11 x 32) – depth 11
  - bram12.v (12 x 32) – depth 12



Peer Evaluation

Designer#A – Testbench fir_tb.v designer

Designer#B – Design fir.v designer

Evaluations

1. Coding Style evaluation by Designer#A
2. Design (fir.v) evaluation by Designer#A
3. Testbench (fir_tb) evaluation by TA

- Designer#A runs a simulation against fir.v from designer#B
- Designer#A writes an evaluation form
  - For every bug found or coding style violation, designer#A gains a point, and designer#B loses a point

Submission (1/2) – Github - TBD by TA

- Github should contain at the following
  - fir.v （the fir design）
  - fir_tb.v (the testbench)
  - Log files including : synthesis, simulation, static timing report
  - Synthesis report – area usage, Including FF, LUT (Note: there should be no BRAM because BRAM is an external model, not in the RTL design)
  - Timing Report, including slack, and max delay path
  - Waveform – show
    - Configuration write
    - ap_start, ap_done (measure # of clock cycles from ap_start to ap_done)
    - Xn stream-in, and Yn stream-out
  - Report in HackMD
- README.md

Submission (2/2) – Report (HackMD) - TBD

- Block Diagram
  - Datapath – dataflow
  - Control signals
- Describe the operation, e.g.
  - How to receive data-in and tap parameters and place into SRAM
  - How to access DataRAM and tapRAM to do the computation
  - How ap_done is generated.
  - …..
- Resource usage: including FF, LUT, BRAM
- Performance report: latency & throughput (# of clock to generate a Y output)
- Timing Report
  - Synthesize the design with a maximum frequency
  - Report timing on the longest path, slack
- Simulation Waveform, show
  - Coefficient program, and read back
  - Data-in stream-in
  - Data-out stream-out
  - RAM access control