Bridge of Life Education

SOC Design

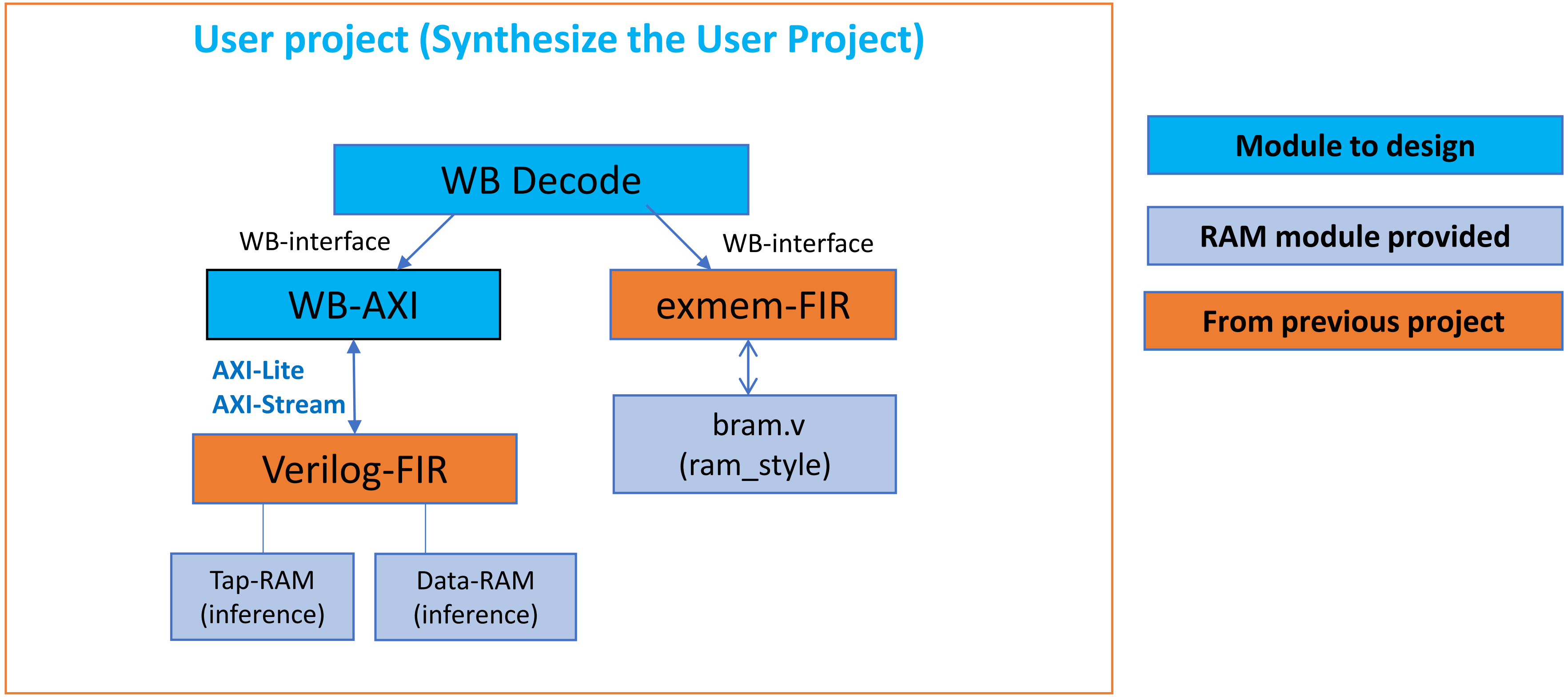
Lab4-2 Caravel FIR

Jiin Lai

Background

- In previous lab, you have designed
  - Lab3 – Verilog FIR
    - Design an FIR HW accelerator with host interface protocol
  - Lab4-0, Lab 4-1 - Caravel SOC simulation & Execute code in user project memory (exmem-fir)
    - Design firmware run in Caravel RISC-V core
    - Integrate an RAM in user project to interact with firmware and testbench
- In this lab, you will
  - Integrate Lab3-FIR & exmem-FIR (Lab4-1) into Caravel user project area (add WB interface)
  - Execute RISC-V firmware (FIR) from user project memory
  - Firmware to move data in/out FIR
  - You will be challenged to optimize the performance by software/hardware co-design

Design Scope & Hierarchy



The designs are included in user project wrapper

RISC-V / FIR – Handshake Protocol

- RISC-V use MMIO (Wishbone) to Interface with FIR
  - Design your own MMIO configuration address space
    - Define your version of mmio address and bit configuration, make sure the protocol has no chance to fail under any condition.
  - Design Wishbone to Axilite/Axi-stream interface (For Verilog-FIR)
  - Use Lab4-1 exmem-FIR for code storage/access
  - RISC-V moves X[n](input)/Y[n](output) FIR engine

RISCV-FIR Firmware/Hardware Handshake Specification

- RISCV-FIR Firmware/Hardware Handshake Specification
  0. Firmware code loaded into exmem and execute from it (refer Lab4-1)
  1. RISC-V Program coeff, len
  2. RISC-V outputs a Start Mark (‘hA5) on mprj[23:16] to notify Testbench to start latency-timer (in testbench)
  3. RISC-V sends X[n] to FIR (note: make sure FIR is readily to accept X[n])
  4. RISC-V receives Y[n] from FIR (note: make sure Y[n] is ready)
  5. Repeat 3, 4, until len of Y[n] is received
  6. When finish, write final Y (Y[7:0] output to mprj[31:24]), EndMark (‘h5A – mprj[23:16]), record the latency-timer
  7. Testbench check correctness by checking mprj[31:24], and print out the latency-timer.
  8. Repeat 2 – 7 for three times, and record and add up the latency timer

Test Data

- Due to Caravel SOC has only limited data memory, and there is no file system for the data file.
- The tap parameters is defined in the program code (Global data)
  - int[10:0] tap = {0, -10, -9, 23, 56, 63, 56, 23, -9, -10, 0};
- The data set is generated by RISC-V program. Using the following loop to generate X[n] - data_length = 64

```
  // Design your own sequence – area for optimization
for(n = 0; n < data_length; n++)  {
    x[n] = n;
    // send x[n] to FIR
   // receive y[n] from FIR
}
```

Configuration Register Address map (Suggested)

User Project Memory Starting: 3800_0000

User Project FIR Base Address : 3000_0000

0x00 – [0] - ap_start (r/w)

set, when ap_start signal assert

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

[1] – ap_done (ro) -> when FIR process all the dataset, i.e. receive tlast and last Y generated/transferred

[2] – ap_idle (ro) -> indicate FIR is actively processing data

[3] – Reserved (ro) -> read zero

[4] – X[n]_ready to accept input (ro) -> X[n] is ready to accept input.

[5] - Y[n] is ready to read -> set when Y[n] is ready, reset when 0x00 is read

0x10-13 - data-length

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

0x80-83 – X[n] input (r/w)

0x84-87 – Y[n] output (ro)

Simulation & Synthesis

- Integrate user-project into Caravel SOC
- Perform Caravel SOC simulation (RTL) with FIR firmware code
  - Get Performance report (latency count)
  - We only need to run behavioral simulation
- Synthesize User Project
  - Get area/timing report

Metrics to measure the FIR system

- You may use bram11.v or bram12.v （if one extra data buffer helps）
  - We will provide bram11.v and bram12.v (lab-caravel_fir/rtl/user/bram11.v)
- Metrics : **#-of-clock** (latency-timer) * **clock_period** * **gate-resource**
  - **#-of-clock** - the latency-timer in testbench
  - **clock_period** – after synthesis, the longest path in static timing report (worst-case timing)
  - **gate-resource** - # of (LUT + FF) – assuming there is no distributed RAM, nor Block RAM in RTL use.
  - The lower the better

In this lab, you need to design

- **Firmware code**
  - fir.c
  - fir.h
- **RTL Design**
  - Wishbone decoder (refer to lab4-1 **user_proj_example.counter.v**)
  - Wishbone to AXI-interface (refer to lab4-1 **user_proj_example.counter.v**)

- You don’t need to implement on FPGA board and post-synthesis simulation, but still need to do synthesis.

Submission (1/2)

- Hierarchy:
  - StudentID_lab4-2/
    - Waveform
    - Simulation.log
    - Report.pdf
    - Synthesis report
    - Github link

Submission (2/2)

- Github
  - README.md – introduce the content of the work, and how to replicate the work. (replicate to run simulation)
  - Design sources, user_project design, including Exmem, FIR RTL, Firmware code, Testbench
  - Synthesis area report, timing report, and log files generated in the process.
- Report
  - Design block diagram – datapath, control-path
  - The interface protocol between firmware, user project and testbench
  - Waveform and analysis of the hardware/software behavior.
  - What is the FIR engine theoretical throughput, i.e. data rate? Actually measured throughput?
  - What is latency for firmware to feed data?
  - What techniques used to improve the throughput?
    - Does bram12 give better performance, in what way?
    - Can you suggest other method to improve the performance?
  - Any other insights ?