

Aquila Microarchitecture



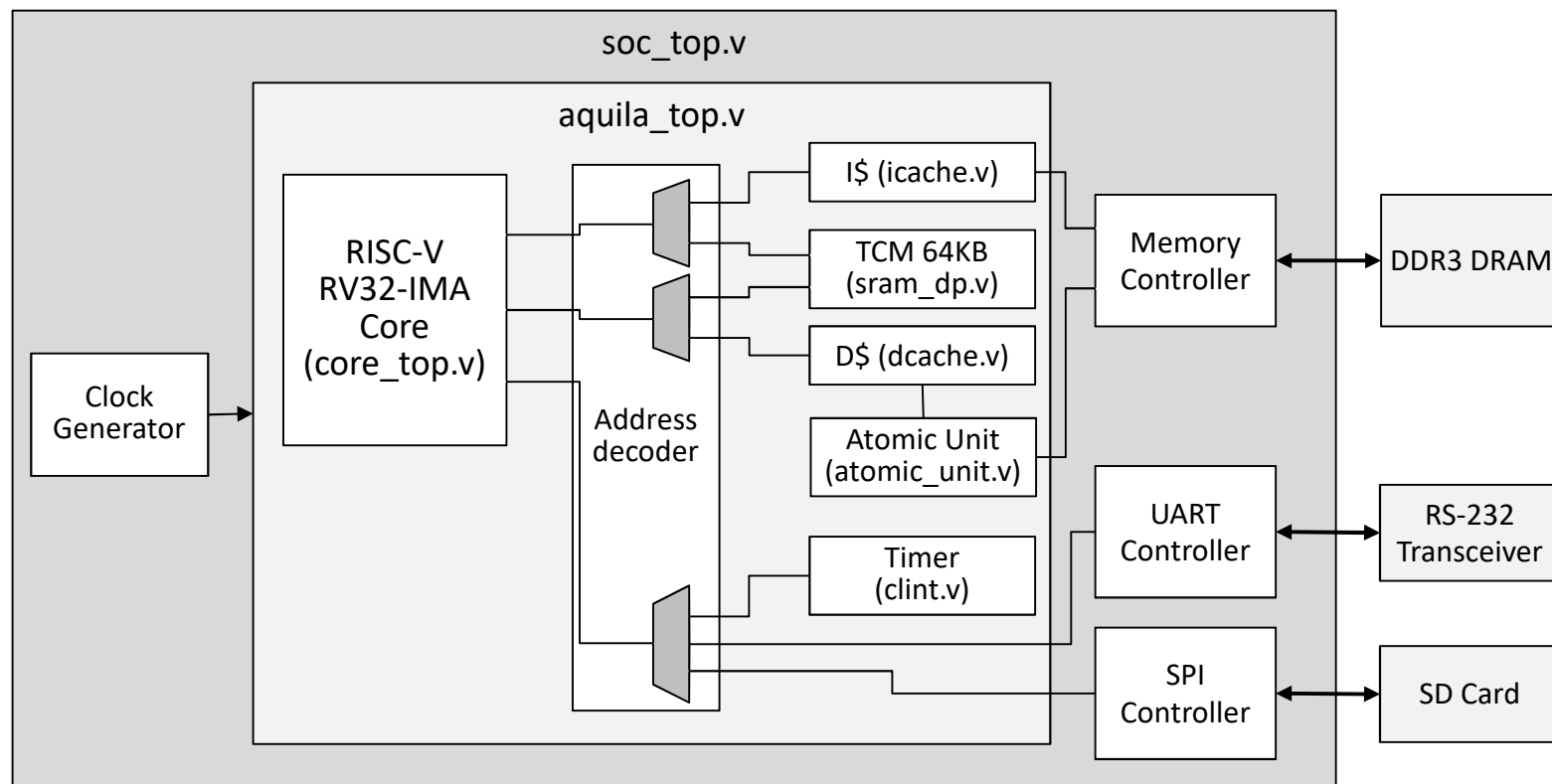
Chun-Jen Tsai
NYCU
10/29/2021

Aquila – An Open Source RISC-V

- ❑ The latest version of Aquila SoC has the specification:
 - RV32IMA ISA-compliant
 - Embedded 64KB tightly-coupled on-chip memory (TCM)
 - L1 4-way set associative data and instruction caches
 - CLINT for standard timer interrupts
 - CSRs & related instructions for M-, U-, and S-mode support
 - Memory Management Unit for SV32 paging support
 - Multi-core support with coherent data cache controller
 - SD card I/O support
 - The RTL model written in Verilog

SoC Interfaces (1/2)

- ❑ Aquila core can be used as:
 - A reusable IP (using AXI4 bus protocol)
 - A proprietary SoC (using proprietary bus protocol)

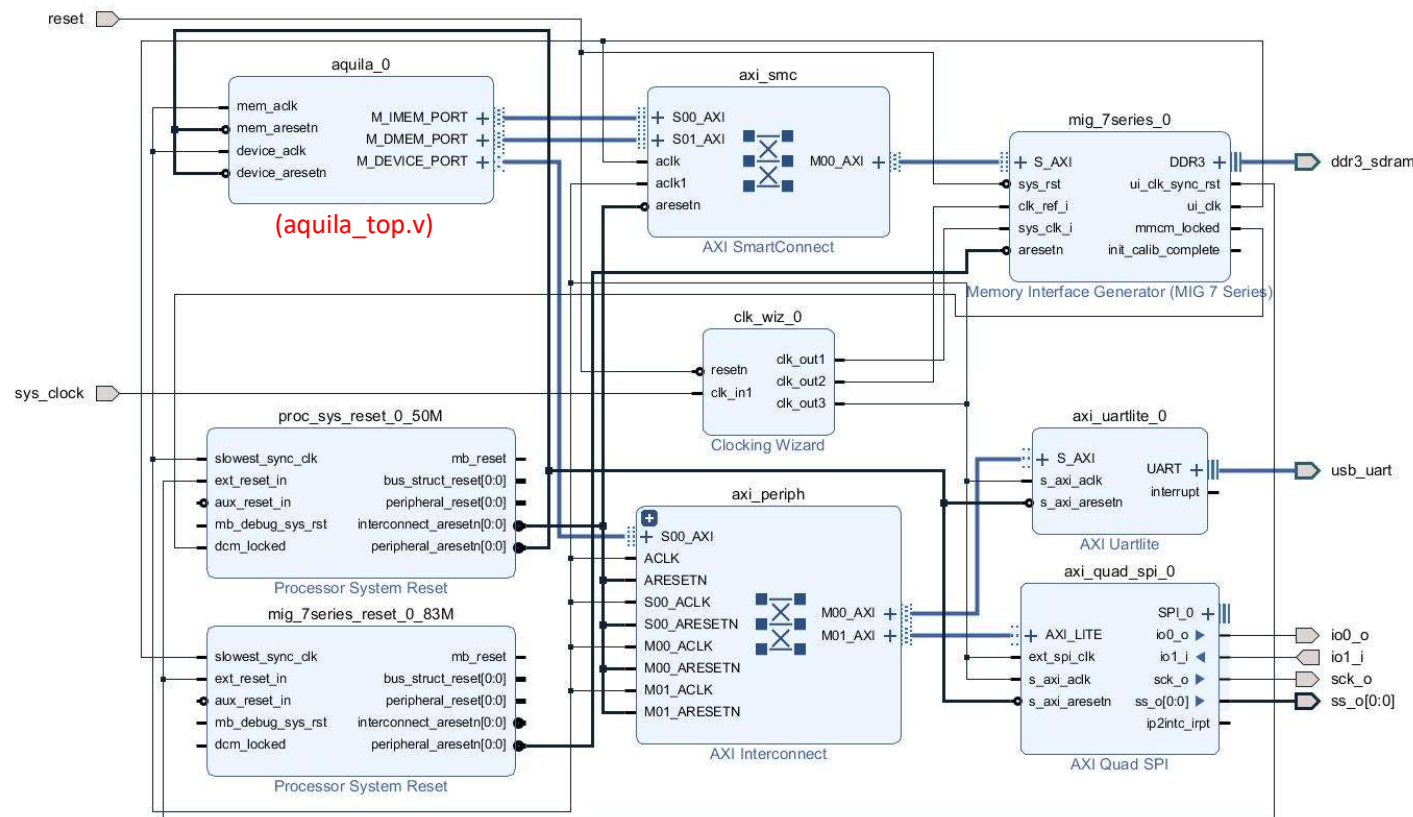


SoC Circuit Block Interfaces (2/2)

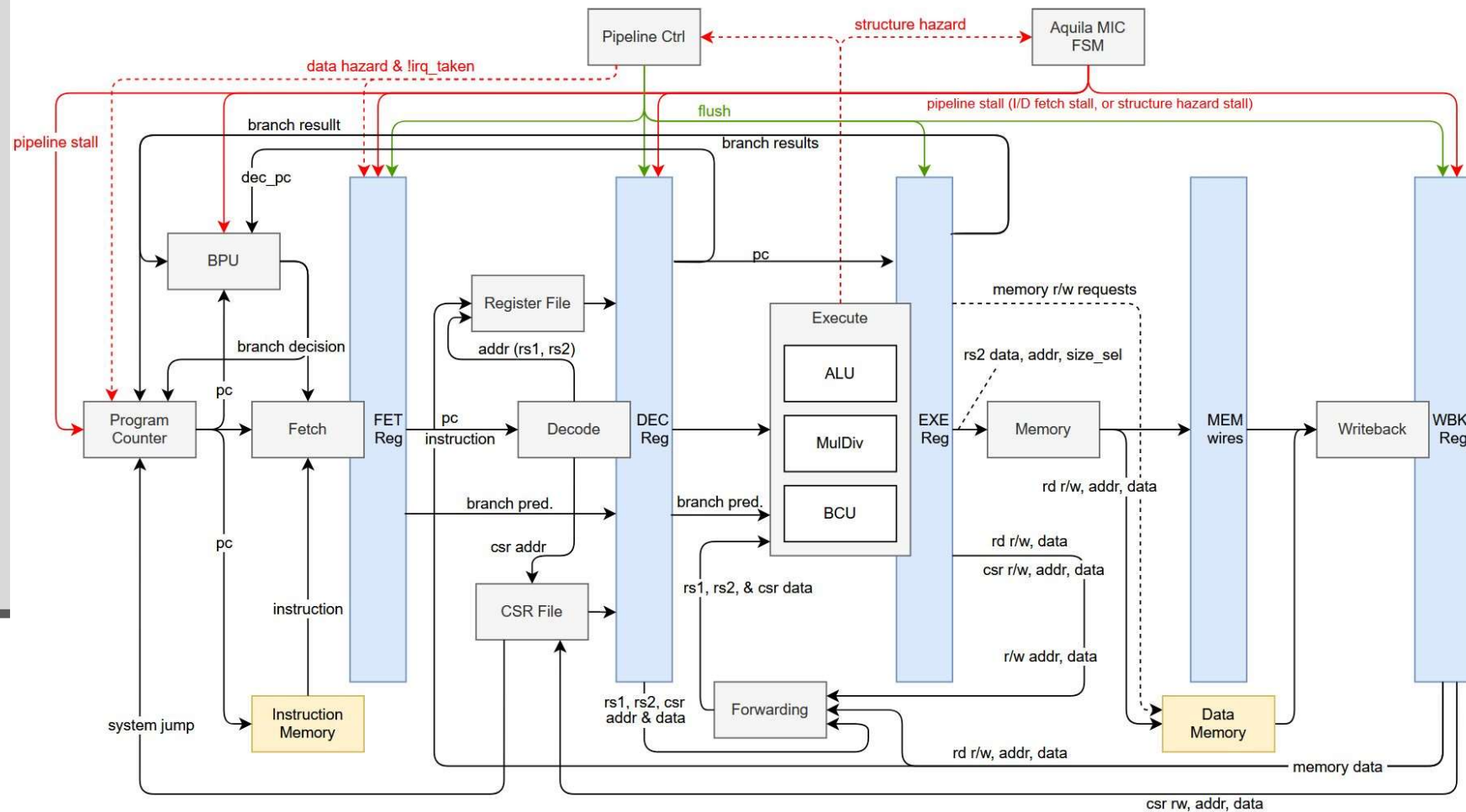
- ❑ Aquila SoC adopts several interfaces for circuit module connections
 - Low-level HW-specific interfaces:
 - memory controller interface
 - cache controller interface
 - General-purpose bus interface
 - Proprietary (Aquila on-chip bus): Timer, UART controller
 - Reusable across vendors (ARM AXI on-chip bus): SPI controller for SD card

Aquila as an AXI Reusable IP

- ❑ Aquila supports AXI bus infrastructure
 - AXI bus is designed by ARM to ease complex SoC integration
 - Vivado has an IP Integrator tool for this purpose:

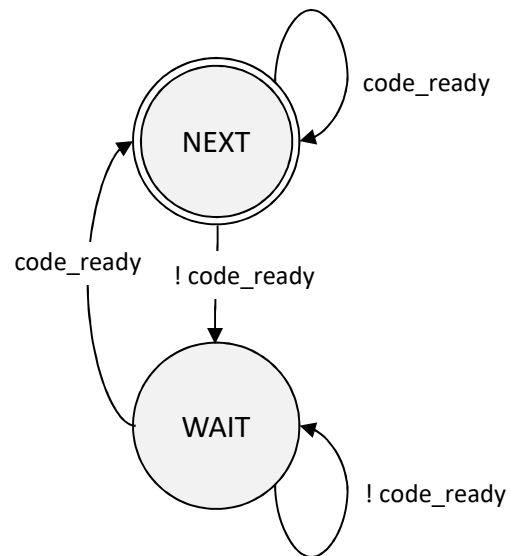


Aquila Pipeline Organization



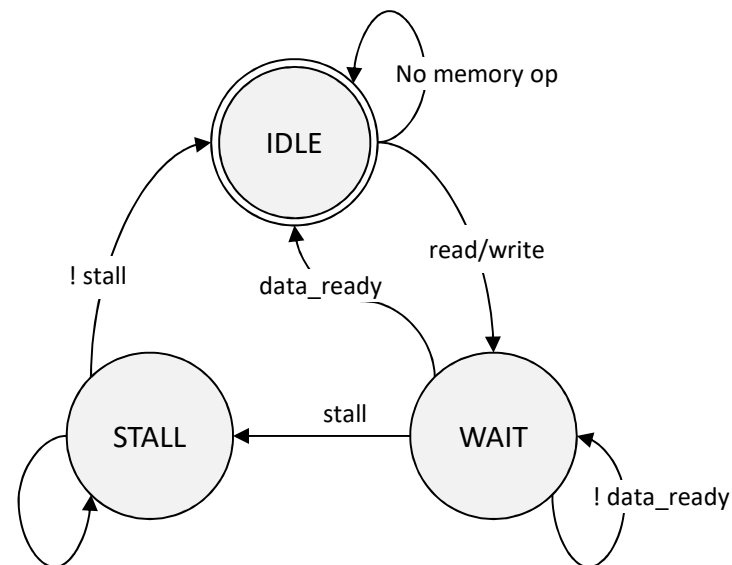
Aquila Memory Interface Controllers

- ❑ Memory interface controllers (MIC) in `core_top.v` controls the instruction & data fetch states:



Instruction fetch controller

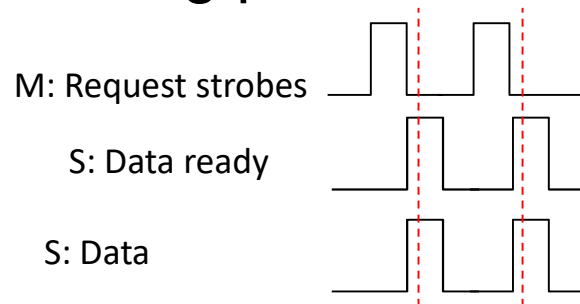
`stall = stall_instr_fetch || stall_from_exe`



Data access controller

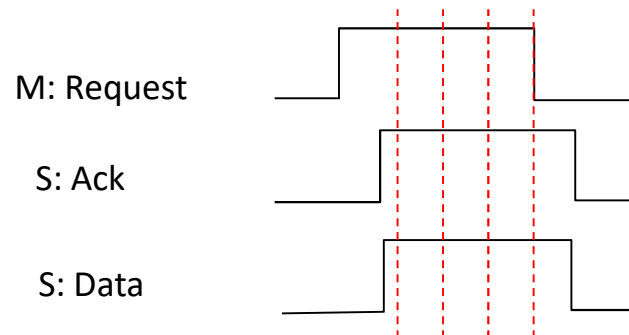
Memory Access Signaling

- ❑ Aquila MIC uses the strobing protocol
- ❑ Strobing protocol: single-cycle request (read)



M means master, S means slave.

- ❑ Handshaking protocol: multi-cycle handshaking (read)



M means master, S means slave.

Pipeline Controller

- ❑ Pipeline Controller sends out flush signals and data hazard signals to different pipeline stages
- ❑ Different stage are flushed differently due to different instructions
 - Fetch – branch, sys_jump, or fence_i instructions
 - Decode – branch, load hazard, illegal, or fence_i instructions
 - Execute – sys_jump or fence_i instructions
 - Writeback – sys_jump instructions

Fetch Stage

- ❑ Receives instructions from the instruction memory, and pass it to the Decode unit

- ❑ Functions
 - Delay the instruction upon stall
 - Send “NOP” instruction upon flush, invalid instruction
 - Branch prediction is often considered as part of the Fetch stage tasks
 - Detect page fault exceptions when we have MMU
 - Pass exceptions to later stages

Decode Stage

- ❑ Extracts different fields from the instruction and sets control and immediate value signals accordingly
 - Very tedious but straightforward to implement

- ❑ Functions:
 - Decode instructions
 - Detect data hazard
 - Happens the 'rd' of the last instruction is the same as either the 'rs1' or 'rs2' of the current instruction
 - Pass exceptions to later stages

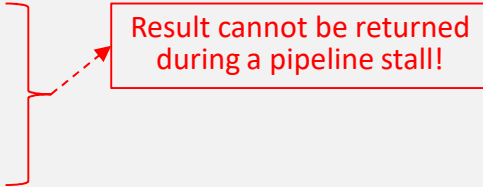
Execute Stage

- ❑ The Execute carries out data manipulation. The critical path of a processor usually involves this stage
 - On FPGAs, the critical path may get a little bit tricky
- ❑ Functions:
 - Single-cycle ALU operations
 - Multi-cycle ALU operations
 - Branch comparison calculations (using exclusive comparators)
 - Branch address computation (using exclusive adder)
 - Sent out memory read/write requests
 - May generate ALU exceptions
 - Pass exceptions to later stages

Multi-Cycle Instructions

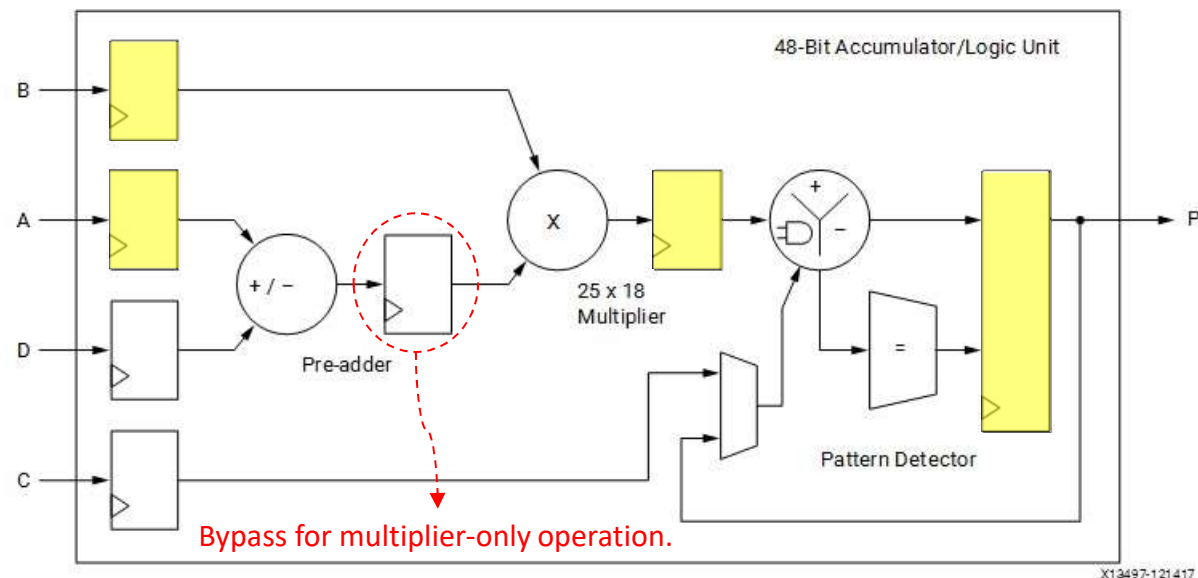
- ❑ RISC-V Ext. M is in the MulDiv module
 - This module is more of a template for multi-cycle instructions
 - The FSM for multi-cycle multiply/division:

```
always @(*)
begin
  case (S)
    S_IDLE:
      S_nxt = (req_i)? (is_a_zero | is_b_zero)? S_DONE : S_CALC : S_IDLE;
    S_CALC:
      S_nxt = (is_calc_done)? S_SIGN_ADJUST : S_CALC;
    S_SIGN_ADJUST:
      S_nxt = S_DONE;
    S_DONE:
      S_nxt = (stall_i)? S_STALL : S_IDLE;
    S_STALL:
      S_nxt = (stall_i)? S_STALL : S_IDLE;
    default:
      S_nxt = S_IDLE;
  endcase
end
```



Multiplier on FPGA (1/2)

- ❑ On Xilinx FPGAs, multiplier can be implemented using LUTs or DSP48 slices
 - Aquila has both implementations
- ❑ For DSP48 multiplier, you can specify the #stages
 - There are three sets of registers along the combinational path
 - Each set can be bypassed for a long combinational path



Multiplier on FPGA (2/2)

- ❑ To use DSP slices for 32×32 multiplication, four slices will be used:

$$((a \ll 16) + b) \times ((c \ll 16) + d) = (a \times c) \ll 32 + (a \times d) \ll 16 + (b \times c) \ll 16 + b \times d.$$

- ❑ The number of pipeline stages can be inferred using the following code patterns:

```
always @(posedge clk)
begin
    m_r <= v1*v2;
end
```

one-stage

```
always @(posedge clk)
Begin
    v1_r <= v1;
    v2_r <= v2;
    m_r <= v1_r*v2_r;
end
```

two-stage

```
always @(posedge clk)
Begin
    v1_r <= v1;
    v2_r <= v2;
    m0_r <= v1_r * v2_r;
    m_r <= m0_r
end
```

three-stage

Memory Stage

- ❑ The Memory unit of Aquila is a combinational circuit
 - It does not take the clock & reset inputs
 - The pipeline stage “state” is stored in the data memory
- ❑ Tasks:
 - Prepare 8-bit or 16-bit data for 32-bit buses
 - Setup the byte-selection signals
 - Detect memory alignment exceptions
 - Pass exceptions to later stages
- ❑ Aquila only supports aligned data accesses for 16- or 32-bit data since non-aligned data access may take multiple cycles
 - This is an constraint from the bus & memory device

Writeback Stage

- ❑ The Writeback unit performs physical write back to registers
 - The source may come from the Execute unit (through the Memory unit) or the data memory
 - The destination may be the register file, the CSR file, or the Forwarding unit

- ❑ Functions:
 - Data write back to registers
 - Perform sign-extension if necessary
 - Handles exception

Atomic Unit

- ❑ Atomic unit sits between the data cache and the DDRx memory controller, intercepting all the memory requests from the atomic instructions

- ❑ Functions:
 - If the load/store instruction is not atomic instructions, bypass
 - If it is a lock-based AMO instruction, perform the atomic read-modify-write operations
 - If it is a lock-free LR / SC, manage the reservation table:
 - Register the reservation for LR by its thread ID
 - Check the reservation to allow an SC to cancel all other ID's reservations

Multi-Core Atomic Unit Configuration

- ❑ Aquila is multi-core capable
 - Atomic unit is not really critical for a single-core processor
 - For multiple cores, we need **arbitration** and **coherence**

