HW#3 Cache Optimization

Chun-Jen Tsai
NYCU
11/18/2022

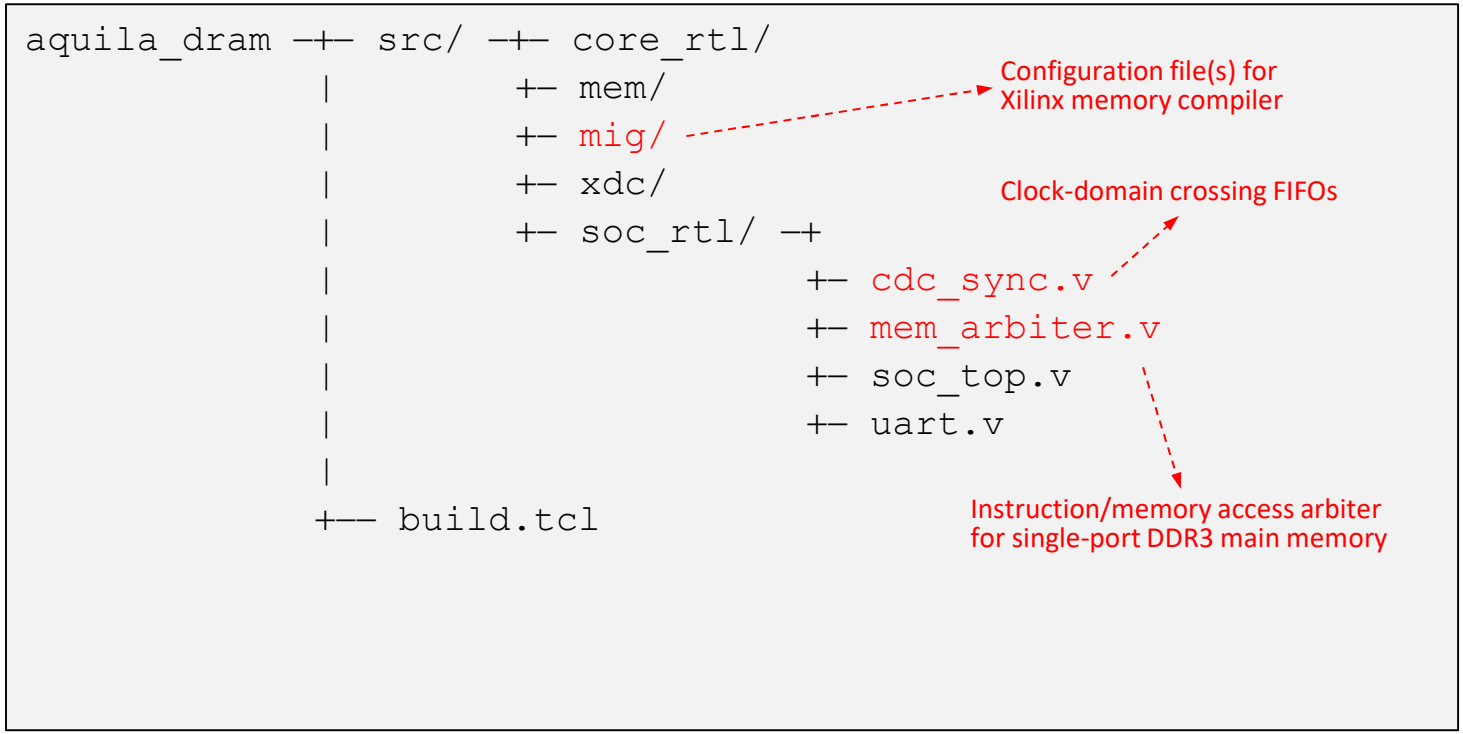
Homework Goal

- Cache is crucial for the memory performance of a microprocessor. In this HW, we must analyze and optimize the data cache for the CoreMark benchmark
- Your tasks:
  - Analyze the cache behavior over CoreMark execution
  - Optimize the data cache to improve CoreMark
- Upload your code & report to E3 by 12/04, 23:55.

analyze the data cache system.

Aquila SoC with DRAM Support

- For this homework, download the new HW workspace `aquila_dram_build.zip` from E3:



Some Simple Statistics

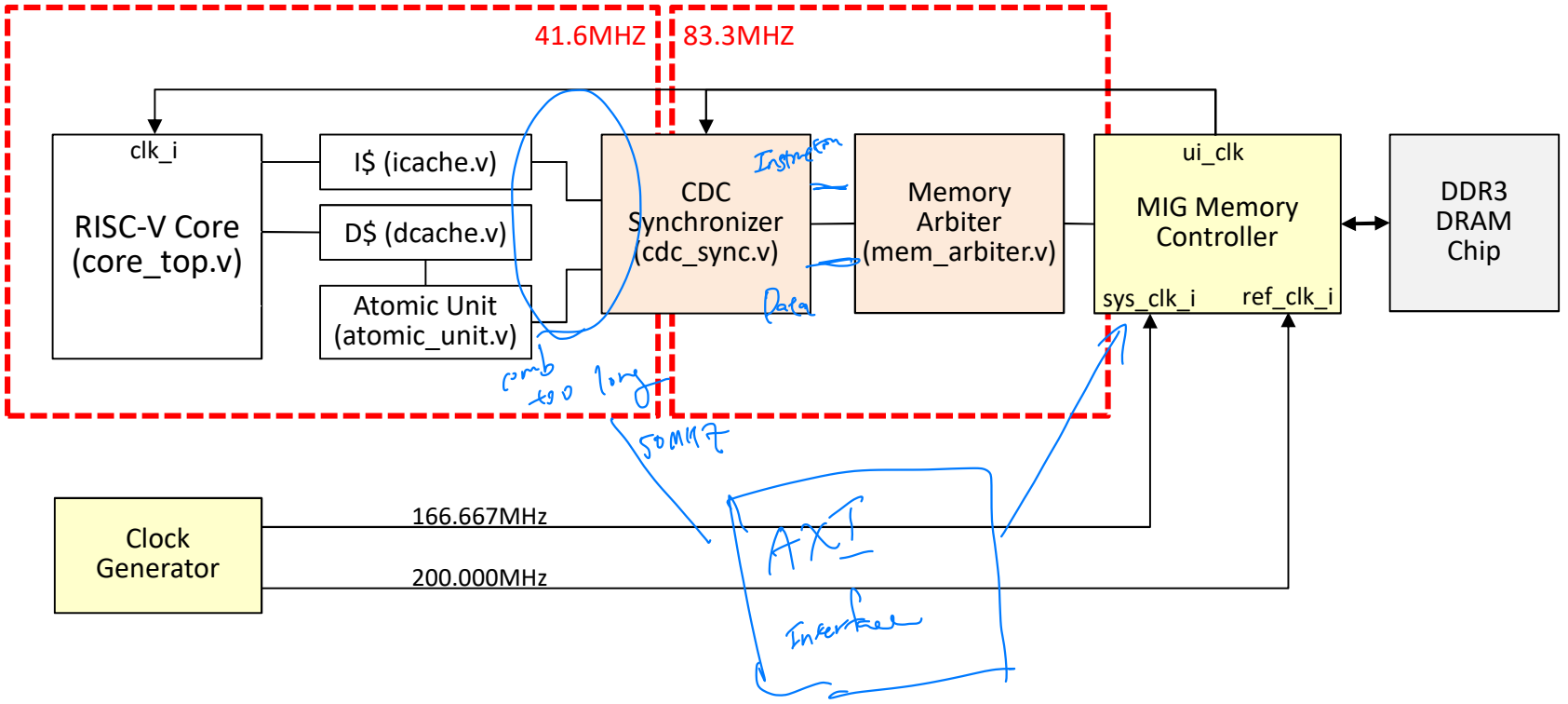
- Aquila has a pair of 4-way set associative I/D-caches. Default cache sizes is set to 8KB, the logic usage is
  - 33% usage of LUT
  - 20% usage of FF
  - 25% usage of BRAM
- Using the linker script to put code/data/heap/stack in either TCM or DRAM, the CoreMark/MHz are:
  - Running on TCM: 2.07
  - Running on DRAM, 8KB caches: 1.71
  - Running on DRAM, 4KB caches: 1.69
  - Running on DRAM, 2KB caches: 1.63

DRAM Interface of Aquila (1/2)

- The DRAM Chip on Arty is a Micron *MT41K128M16JT-125*
  - The chip is a 16-bit DDR3-1600 component, but is under-clocked at 333.333 MHz (equivalent to a DDR3-667)
  - The memory controller we use is from Xilinx, called MIG
  - To support 333MHz DRAM clock, the MIG must run at 333/4 = 83.333 MHz or 333/2 = 166.667 MHz
- 83.333 for Aquila on Arty is too high, thus, we choose to use 83.333/2 = 41.6667 MHz for Aquila

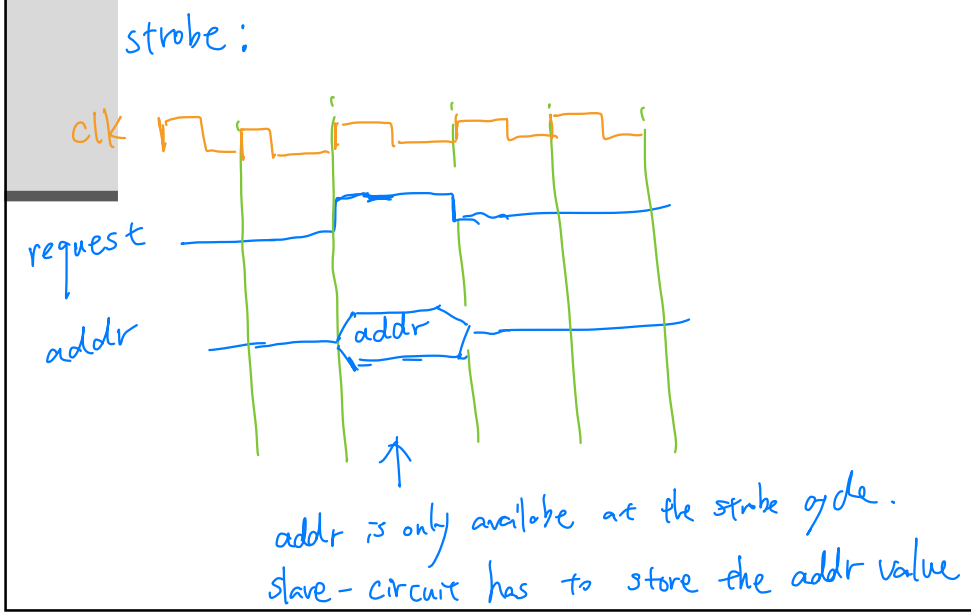
DRAM Interface of Aquila (2/2)

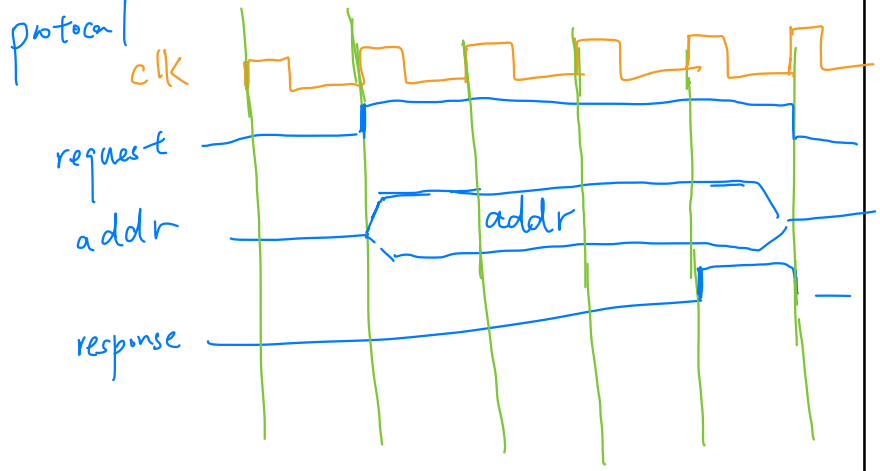
- Both instruction and data memory shares the same DRAM, so we must add an arbiter and a CDC synchronizer to the system:



Handling Aquila Memory Requests

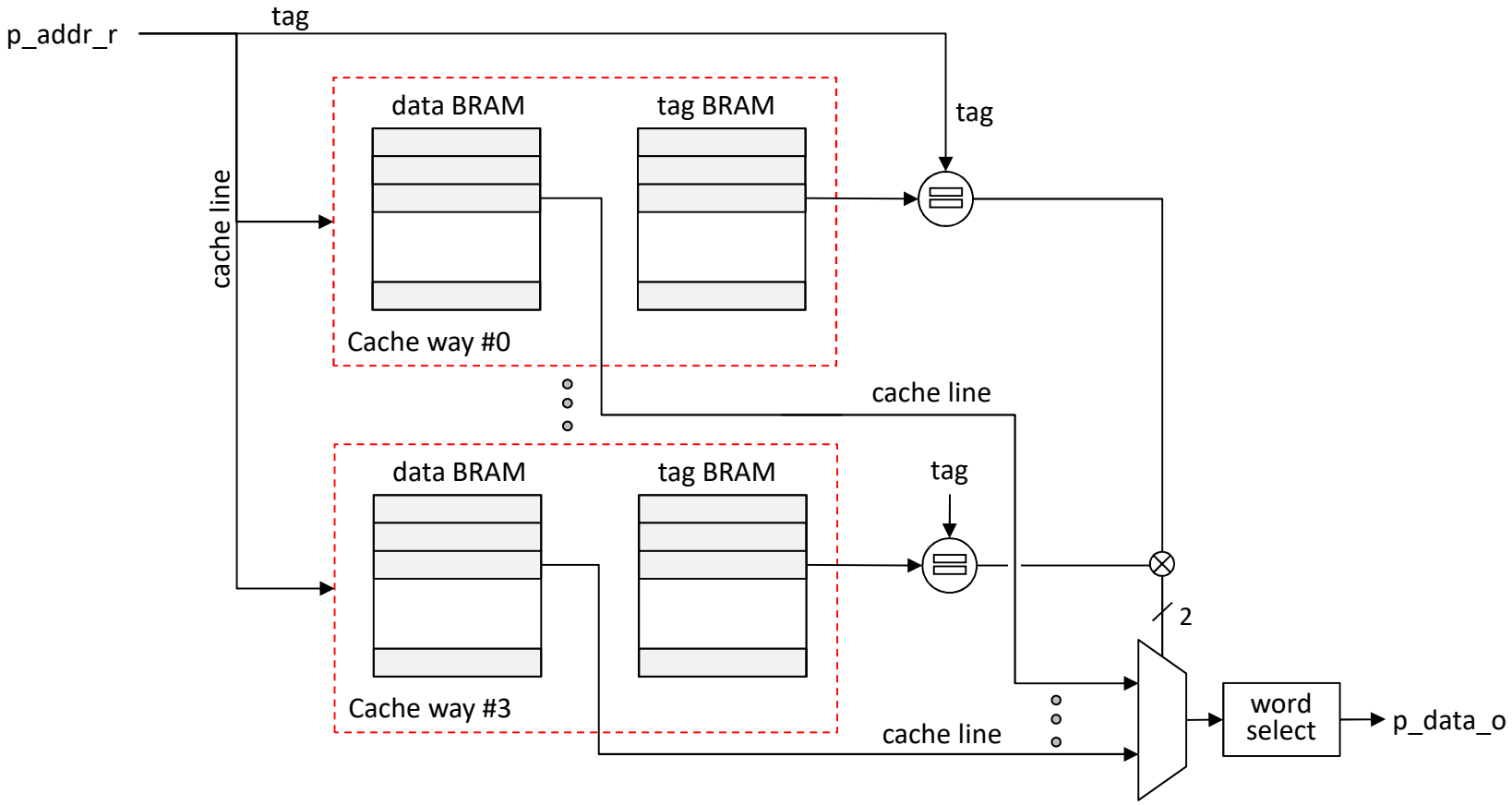
- Unlike TCM, a request to data cache may take multiple cycles to complete
- Aquila uses single-cycle strobe signals for memory requests
- We must register `p_addr_i` during the strobe cycle inside the cache controller for multi-cycle processing





Cache Organization of Aquila SoC

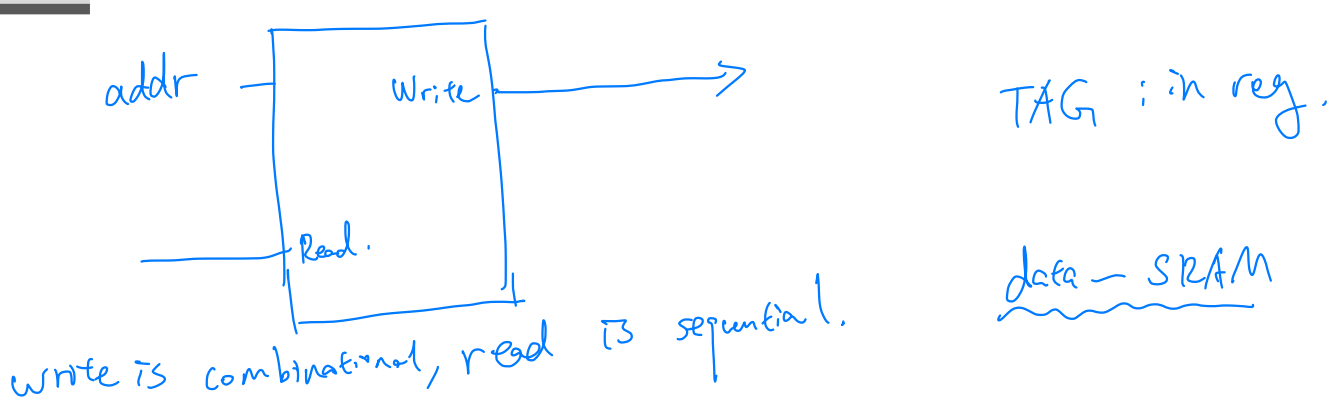
- Data flow on cache hit:



Cache Memory Coding Issue

- Each cache line should have a pair of “valid” and “dirty” flags that stores the state of the cache line
  - Valid – the cache line contains valid data
  - Dirty – the data in the cache line have been modified

- These flags can be synthesized using LUTs or BRAMs
  - For a small cache, using BRAM may be wasteful since BRAMs are allocated in 18-kbit unit
  - Aquila uses LUTs for cache block flags



Implementing 1- or 2-Port Memory

- A memory with up to 2 ports can be implemented using LUTs, Flip-Flops, or BRAMs of FPGA:

```
reg VALID_ [0 : N_LINES-1][0 : N_WAYS-1];
reg DIRTY_ [0 : N_LINES-1][0 : N_WAYS-1];
```

LUT / FF

- How do you control the implementation methods?
  - By proper inference coding style (see `sram.v` or `sram_dp.v`) or by a pragma in your code (may not be honored):

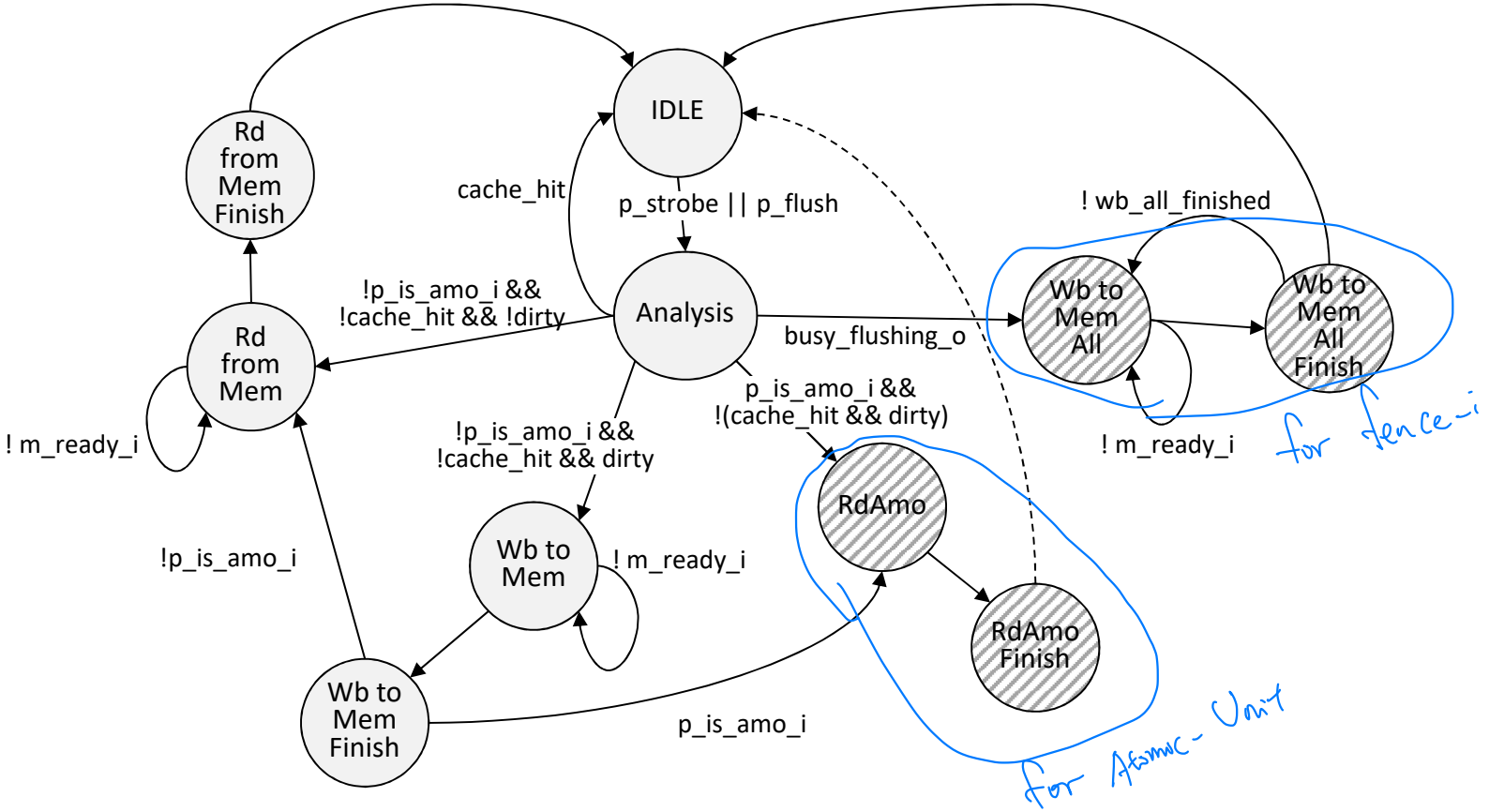
```
(* ram_style = "block" *) reg [0:31] my_ram [127:0];
```

    - Type of RAM styles are: `block`, `distributed`, or `ultra`

  - BRAMs can only be used to synthesize a memory array with at most two clocked ports, each port must be controlled by an enable and a read/write signals

Cache Controller

- The FSM of the D-Cache controller is as follows:
  - You can ignore the shaded states for this homework!



Measuring Performance Hotspot

- You should add some counters in the cache controller to find the hotspot by collecting the following statistics
  - Average cache latency for each memory request
    - Read/write latency should be separated
    - Miss/hit latency should be separated
  - Cache hit/miss rates

- By latency, we mean the #cycles between the `p_strobe_i` and `p_ready_o`.

For Performance Improvements

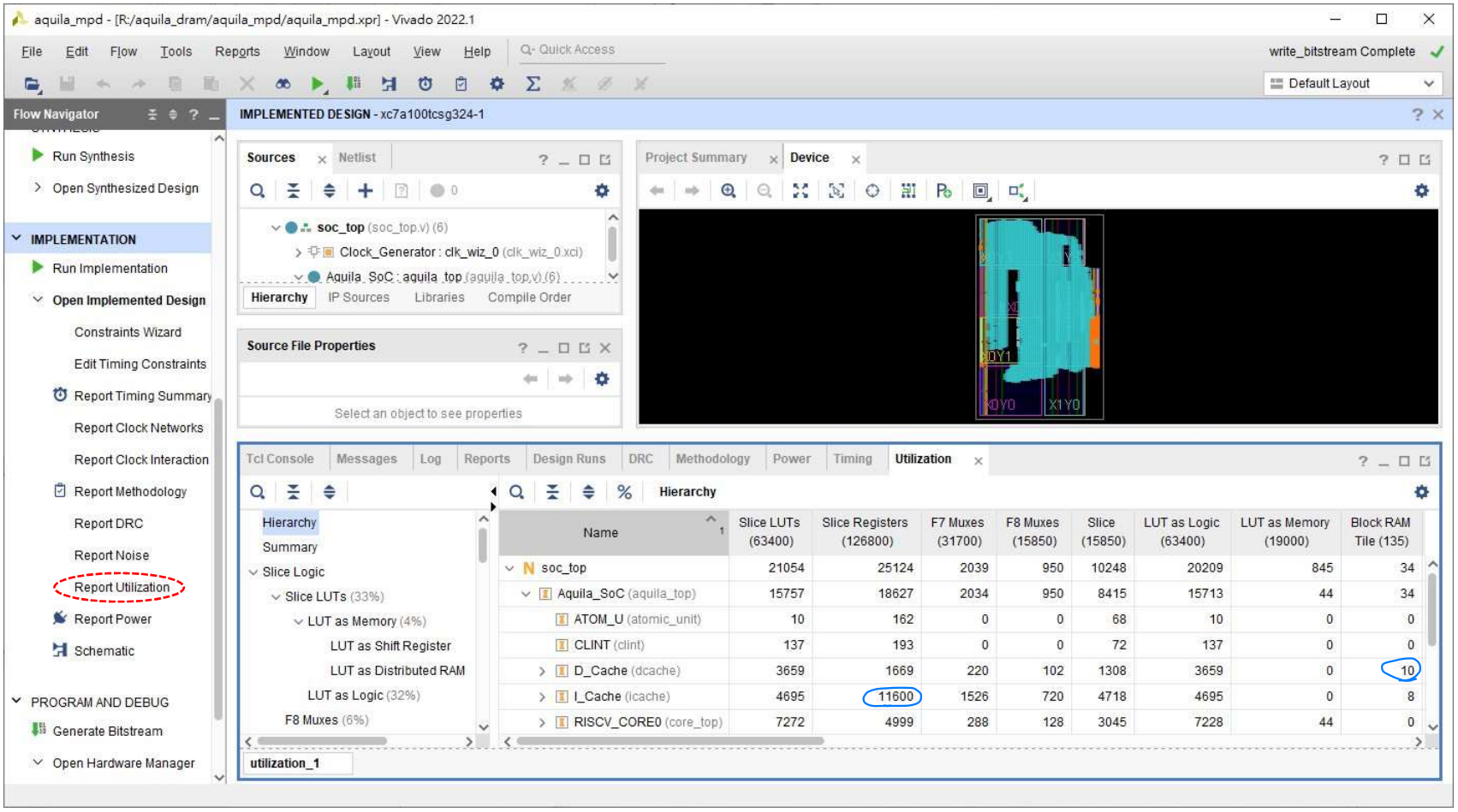
- There are a few things you can try to improve the D-Cache performance of Aquila:
  - Change cache ways (2- and 8-way caches are worth trying)
    - Changing the local parameter N_WAYS is not enough. Several places in `dcache.v` must be modified for different cache ways.
  - Change the cache replacement policy
  - Applying a good pre-fetching algorithm
  - Redesign the cache controller based on the statistics you have collected

Warning on Using ILA

- For this homework, you probably wants to use ILA for debugging because it is difficult to simulate DDR memory at the system level
- ILA uses on-chip memory to capture data. If you need to capture a lot of data, you may have to reduce the sizes of TCM to save more BRAMs for ILA
  - The boot code size is less than 4KB, the TCM size is 64KB

Resource Usage on the FPGA

- Checking FPGA utilization after implementation:



Memory Controller

- Memory controller connects the processing cores to the main memory
  - Typical main memory is composed of DRAM chips
  - Crucial to data-intensive applications
- Types of DRAM chips
  - SDR – old DRAM that handles one transaction per DRAM clock cycle, up to 133 Mhz
  - DDR – modern DRAM that handles two transactions per DRAM clock cycle, DDR4 can be up to 1.6 GHz

SDR : Single Data Rate

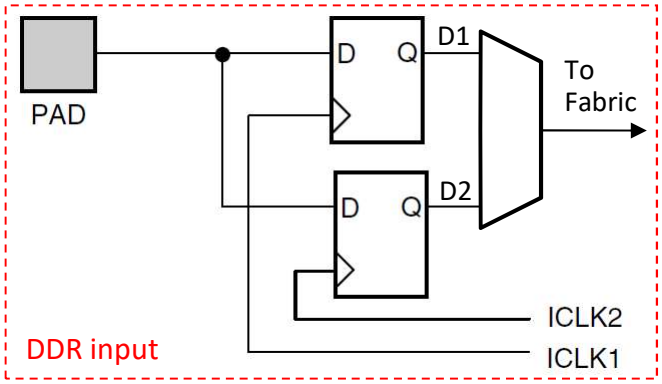
DDR : Double Data Rate

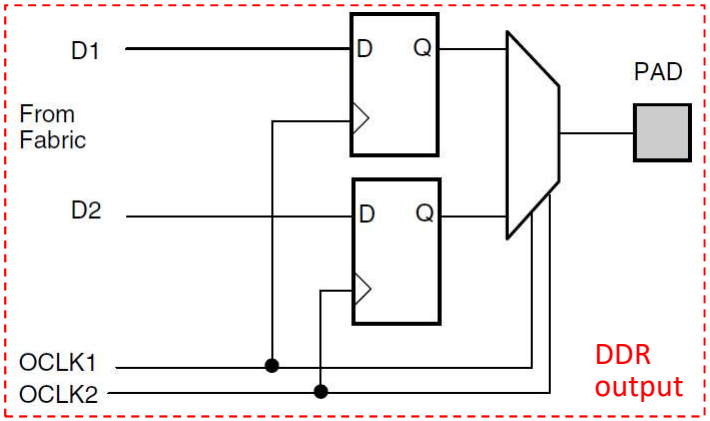
DDR4-2666

over-clocked (DDR4-3200 / DDR4-3800)

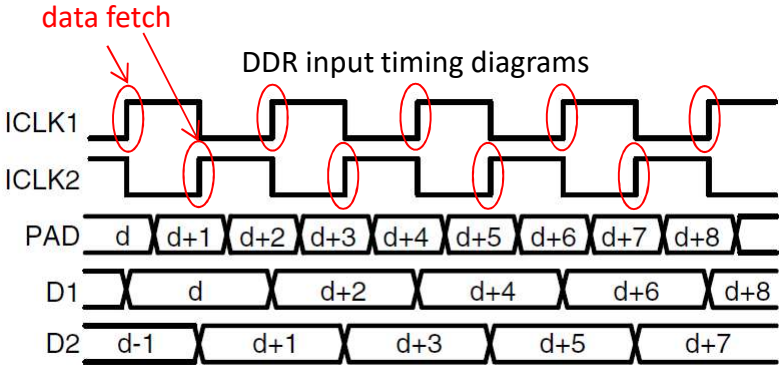
DDRx Memory Controller (1/2)

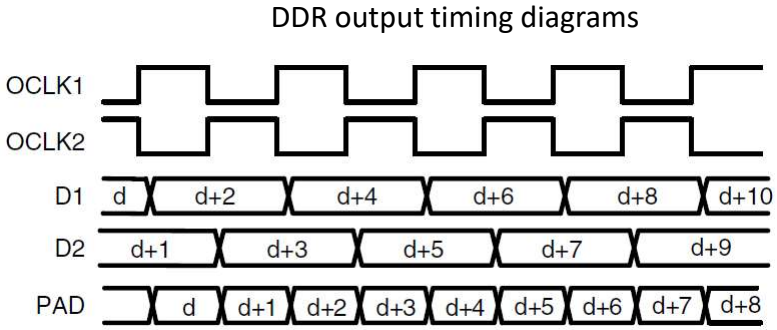
- We can design a low-speed DRAM controller and connect it to a DRAM chip with generic FPGA user pins





CLK1 and CLK2 are 180° phase shifted





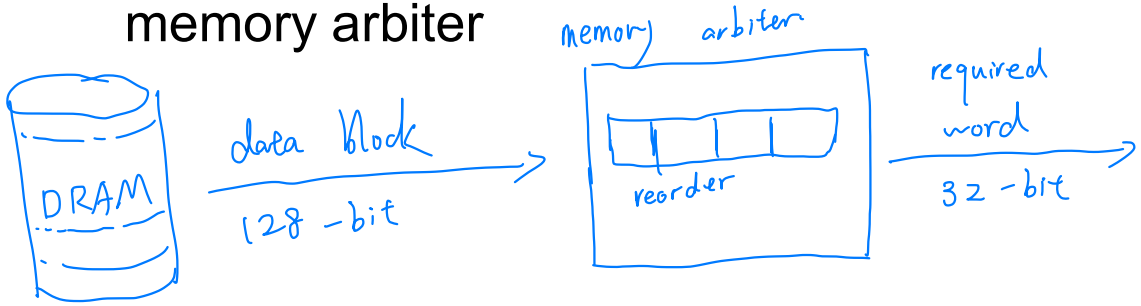
DDRx Memory Controller (2/2)

- High-speed DDRx memory controller IPs are complicated and requires some dedicated I/O logic
  - Only certain FPGA I/O banks can be used to connect to the high-speed DRAM chips
  - The controller need custom I/O pins to talk to the DRAM chips
- Xilinx solution for memory controllers
  - Xilinx provides a configurable Memory Interface Generator (MIG) that can be used to generate a memory controller
  - The available DRAM parameters depends on the FPGA family
    - On Kintex devices, DRAM clock up to 800MHz (DDR3-1600)
    - On Artix devices, DRAM clock up to 400MHz (DDR3-800)
    - UltraScale+ devices supports DDR4 chips

MIG Interface on the Processor Side

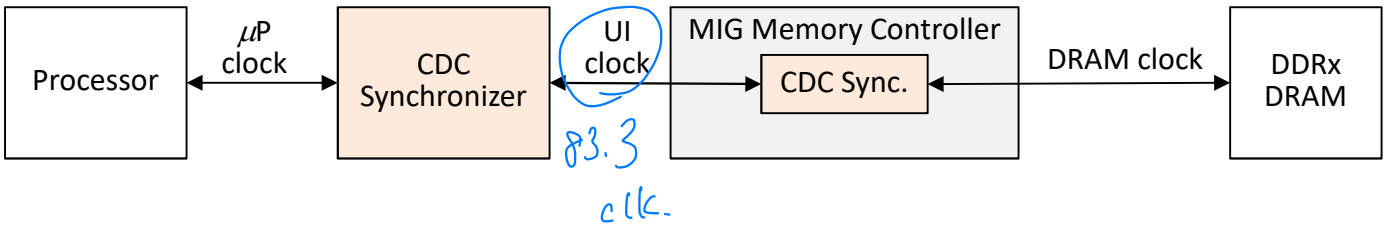
- MIG supports two types of processor side interfaces:
  - AXI interface – easier to use if your processor has AXI-compatible memory ports (Has more resource on the Net, AXI is popular)
  - Native interface – close to the real DRAM chip interface, more efficient to use, but your logic must handle the DRAM burst re-ordering and the large access block issues.

- For this HW, we choose to use the native MIG interface
  - Aquila has I-Cache and D-Cache so we always access the DRAM on a block basis (128-bit at a time)
  - Burst ordering issue is not hard to handle, we do that in the memory arbiter



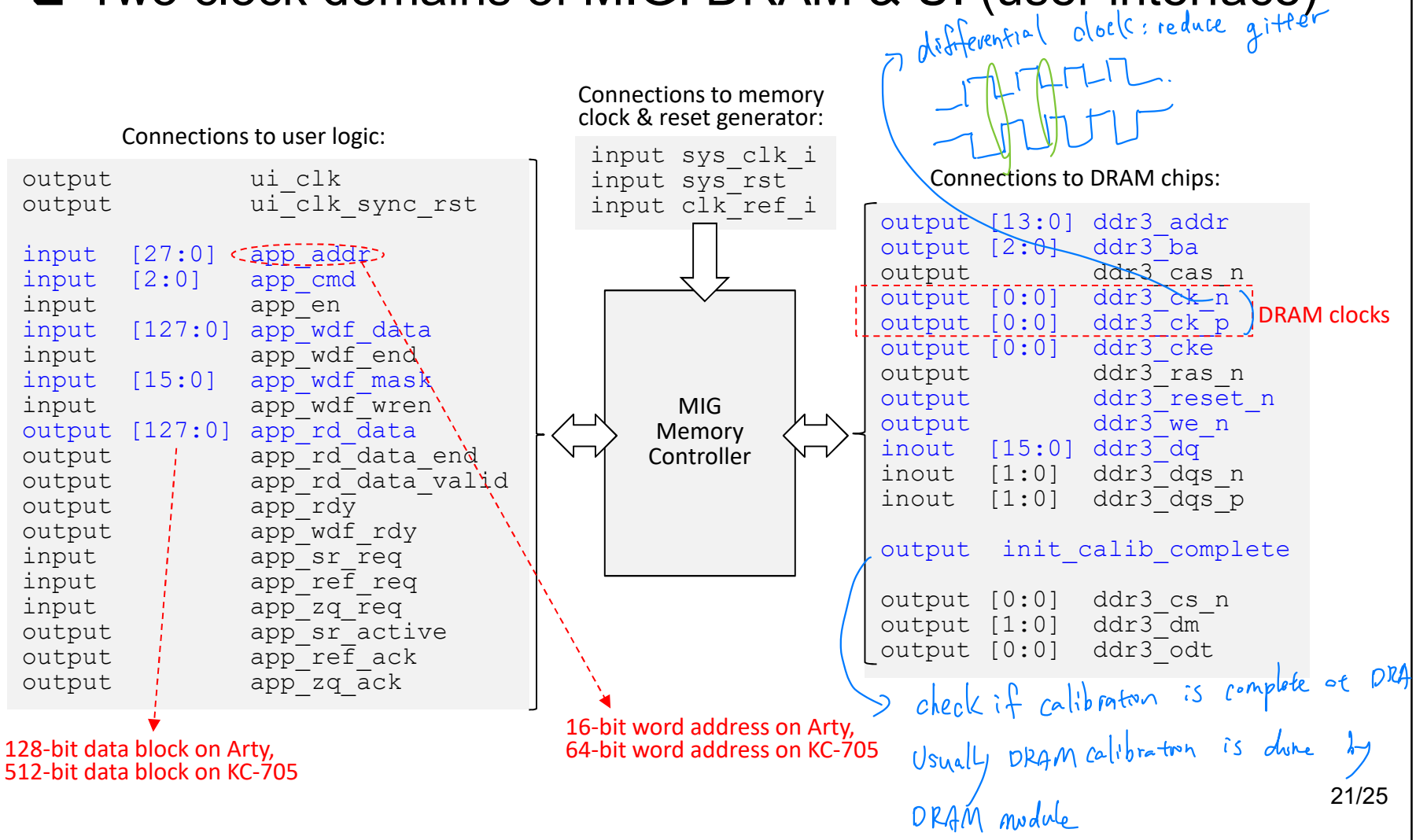
Clock Domain Crossing of MIG

- The memory controller generated by MIG is a cross-clock domain IP
  - On DRAM side, it runs at `sys_clk` rate (166.667MHz on Arty)
  - On processor side, it runs at `ui_clk` rate (83.333MHz on Arty)
- If `ui_clk` is too high for the processor, we must produce a slower clock for the processor core
  - In this case, a CDC synchronizer module must be used to connect the processor to the memory controller:



DRAM Native Interface

- Two clock domains of MIG: DRAM & UI (user interface)



Block-based I/O of Memory Controller

- DRAM chips typically operates on a row at one time, each read/write operation is for a row of memory cells
  - The memory controller will read/write a large block at one time
- On Arty, MIG read/write 128-bit data at a time
  - You specify the 16-bit starting word addresses, the memory controller will read 128-bit data that contains the data in the same row of DRAM cells
  - For writing, a mask can be used to specify the words you want to modify

Data Reordering of Transaction Data

- MIG is hardwired to read/write 8-word burst each time
  - However, DRAM chips output 4-word wrapping burst each time
  - The least significant word contains the `[app_addr]` data
  - For efficiency, a read burst returns data out-of-order
- On Arty, the following logic is used to re-order the data back to normal order (not really necessary for Aquila):

```
always @(posedge clk_i) begin
    if (rst_i) read_data  <= {128{1'b0}};
    else if (read_data_valid_i)
        case(addr_o[2:0])
        3'h0: read_data  <= {word7, word6, word5, word4, word3, word2, word1, word0};
        3'h1: read_data  <= {word6, word5, word4, word7, word2, word1, word0, word3};
        3'h2: read_data  <= {word5, word4, word7, word6, word1, word0, word3, word2};
        3'h3: read_data  <= {word4, word7, word6, word5, word0, word3, word2, word1};
        3'h4: read_data  <= {word3, word2, word1, word0, word7, word6, word5, word4};
        3'h5: read_data  <= {word2, word1, word0, word3, word6, word5, word4, word7};
        3'h6: read_data  <= {word1, word0, word3, word2, word5, word4, word7, word6};
        3'h7: read_data  <= {word0, word3, word2, word1, word4, word7, word6, word5};
        endcase
end
```

2nd 4-word wrapping burst

1st 4-word wrapping burst

un-wrap

Watch the Micron DRAM spec

2-to-1 Memory Arbitration

- Since Aquila has two memory ports (I-Mem & D-Mem) that accesses the DRAM, a 2-to-1 multiplexor must be used to share the single memory controller port
- For Aquila, instruction fetches have higher priority over data accesses

Your Homework

- The goal of this homework is to analyze and improve the data cache
  - Note that you shall keep the data cache size to 8KB.
- Write a report:
  - Analyze the data cache behavior for the CoreMark program
  - Describe the improvements you have done to the data cache
- Note: it is possible that your work turns out to degrade the performance. You can still discuss why your idea does not work and get good grade