HW#3 Cache Optimization

Chun-Jen Tsai
NYCU
11/1/2024

Homework Goal

- Caches are crucial for computer performance. In this HW, we analyze and optimize the data cache for a program that computes π to 5,000 digits

- Your tasks:
  - Analyze the cache behavior of the π program
  - Optimize the data cache @ 4KB to improve the performance

- Upload your code & a report to E3 by 11/19, 17:00.

Aquila SoC with DRAM Support

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

```
aquila_dram/
    |
    +- src/ -+- core_rtl/
    |        +- mem/
    |        +- mig/
    |        +- xdc/
    |        +- soc_rtl/ -+
    |                     +- cdc_sync.v
    |                     +- mem_arbiter.v
    |                     +- mig_7series_sim.v
    |                     +- soc_tb.v
    |                     +- soc_top.v
    |                     +- uart.v
    |
    +-- build.tcl
```

Configuration file(s) for Xilinx memory compiler

Clock-domain crossing FIFOs

I$/D$ accesses arbiter for the single-port DDR3 memory

MIG and DRAM Simulation Model

Top-level simulation model with MIG & DRAM

Some HW/SW Statistics

- Aquila has a pair of 4-way set associative I- and D-caches. The default cache sizes is set to 4KB each, the logic usage of the SoC is
  - 20% usage of LUT
  - 5% usage of LUTRAM
  - 9% usage of FF
  - 26% usage of BRAM

- The computing time of π to 5,000 digits are:
  - Running on DRAM, 4KB D$: 30,517 msec
  - Running on DRAM, 8KB D$: 29,092 msec
  - Running on DRAM, 16KB D$: 16,904 msec
  - Running on TCM: 16,876 msec

The Accuracy Parameter of π

- The program `pi.c` uses Machin's formula to compute π to any # digits (constrained by the main memory size)
  - You can change the macro to change the # digits

```
#define NDIGITS     5000
```

  - However, when NDIGITS > 10,000,000 you have to rewrite the function `termno()`; please see the comments in the code.

Cache Memory Coding Issue

- In addition to tag and data of cache blocks, each block should record “valid” and “dirty” flags:
  - Valid – the cache line contains valid data
  - Dirty – the data in the cache line have been modified

- Memory blocks can be synthesized using LUTRAMs (aka distributed memory) or BRAMs
  - Aquila uses LUTRAMs to store valid bits and dirty bits, and BRAM to store TAGs and cache blocks
  - BRAMs are allocated in 36-kbit or 18-kbit units, so they are less efficient for synthesizing small blocks of memory

Implementing Memory on FPGA

- A memory block can be implemented using LUTRAMs, Flip-Flops, or BRAMs cells of the FPGA

- How do you control the implementation methods?
  1. By proper coding styles (see next slide)
  2. Or, use a pragma declaration (may not be honored):

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

  - Type of RAM styles are: `block`, `distributed`, or `ultra`
  - Whether the pragma can be satisfied or not depends on:
    - How many accesses per clock cycle?
    - How many clocks do you use to synchronize the accesses?

Inferencing Memory Blocks in FPGA

- The write port of a memory block should always be synchronous (i.e. updated at clock edge); the read port can be either
  - Asynchronous read port → synthesize into LUTRAM
  - Synchronous read port → synthesize into BRAM

```
reg [DATA_WIDTH-1:0] RAM [1024:0];

assign data_o = RAM[read_addr_i];

always @(posedge clk_i)
begin
    if (we_i)
    begin
        RAM[write_addr_i] <= data_i;
    end
end
```

LUTRAM

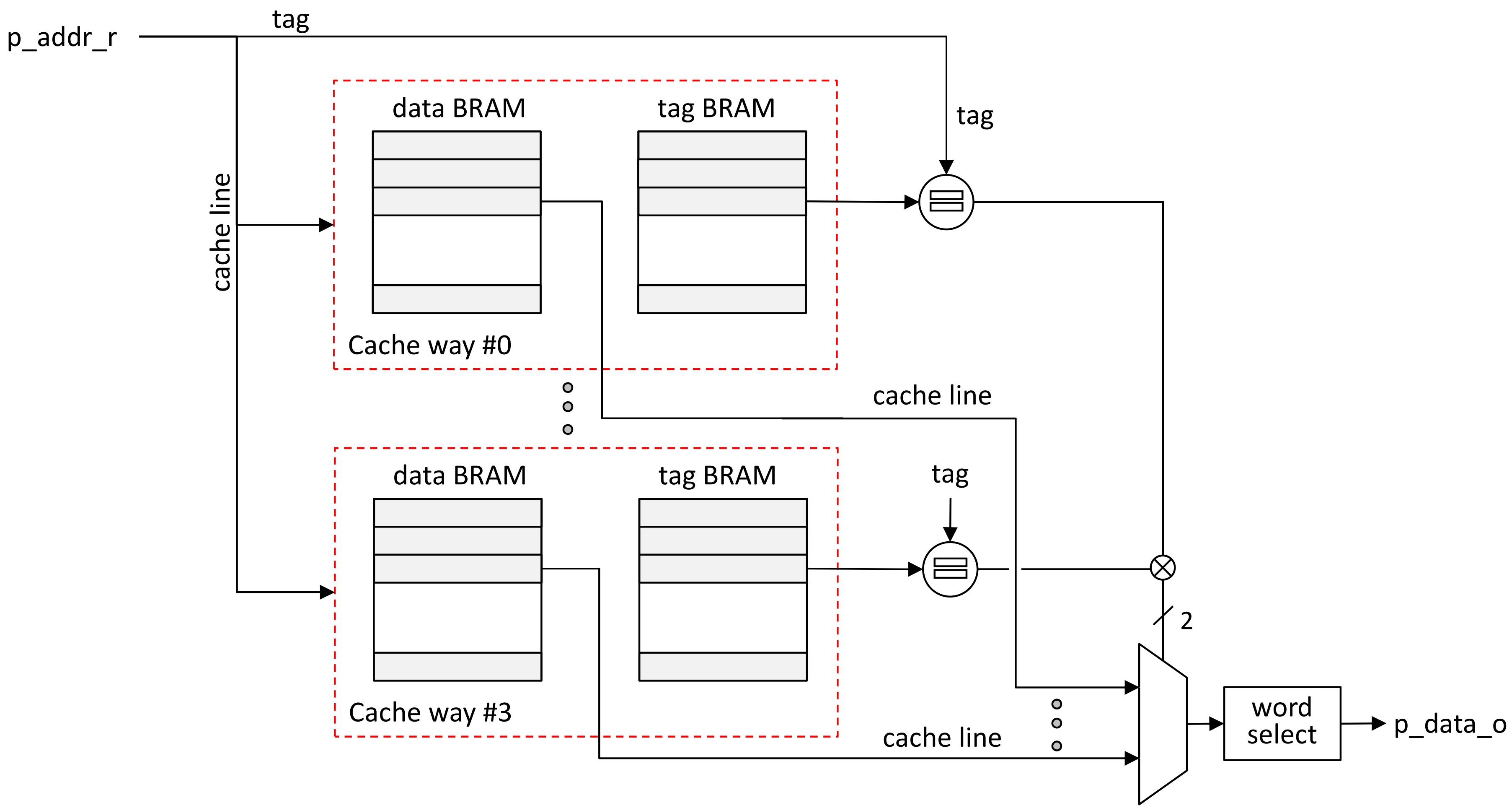
```
reg [DATA_WIDTH-1:0] RAM [1024:0];

always @(posedge clk_i)
begin
    if (en_i)
    begin
        if (we_i)
        begin
            RAM[addr_i] <= data_i;
            data_o <= data_i;
        end
        else
            data_o <= RAM[addr_i];
    end
end
```

BRAM

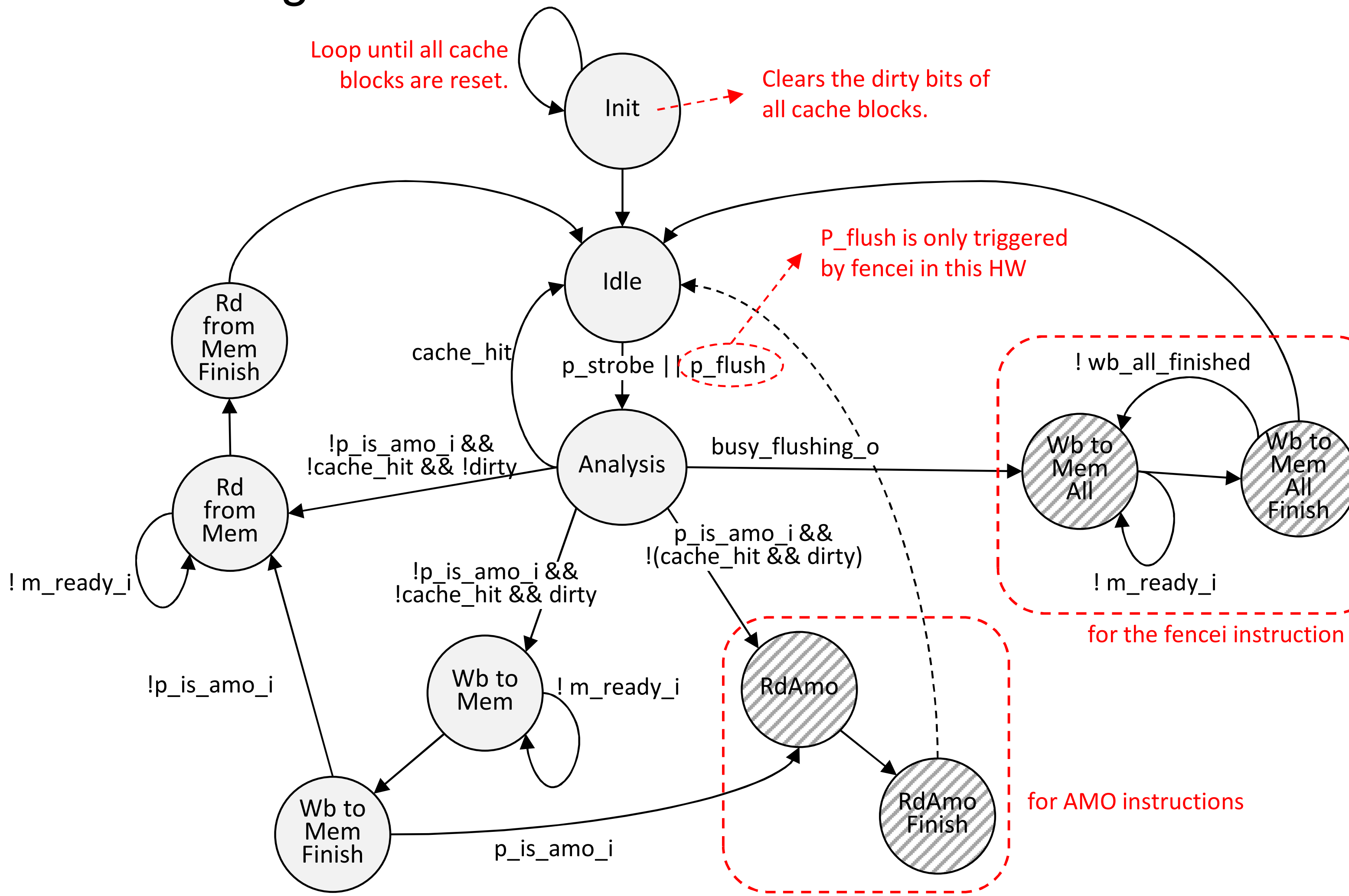
Cache Organization of Aquila SoC

- Data flow on cache hit:



Data Cache Controller

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



Measuring D$ Performance

- You should add counters in the D$ controller to collect 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

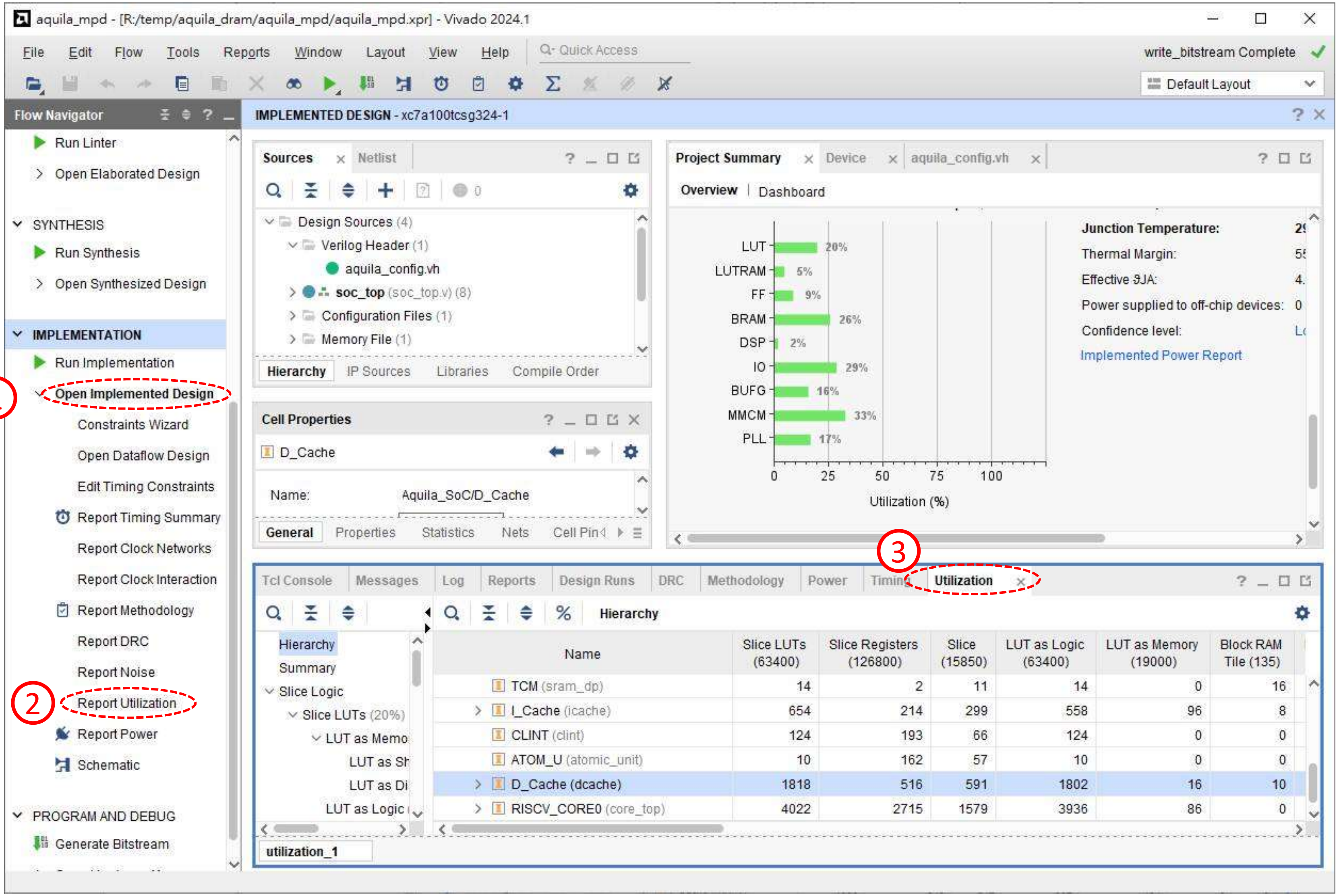
- By latency, we mean the #cycles between the `p_strobe_i` and `p_ready_o` signals of the D$ ports

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.
    - For certain applications, 2-way actually gives better results.
  - Change the cache replacement policy
  - Applying a good pre-fetching algorithm
  - Redesign the cache controller to reduce the stall cycles
  - . . .

Resource Usage on the FPGA

- ❑ Always check FPGA utilization after implementation:



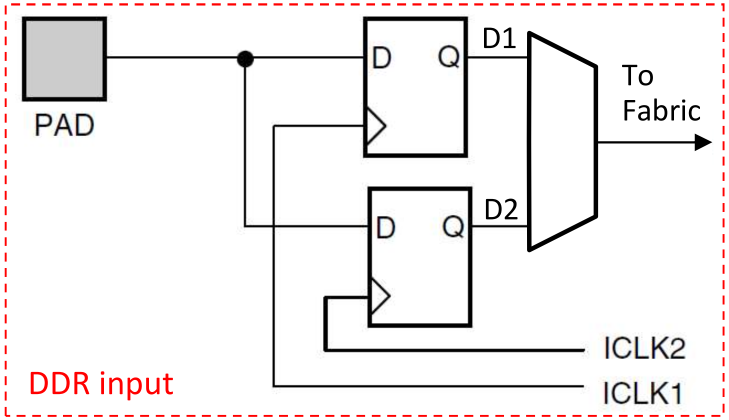
Memory Controller

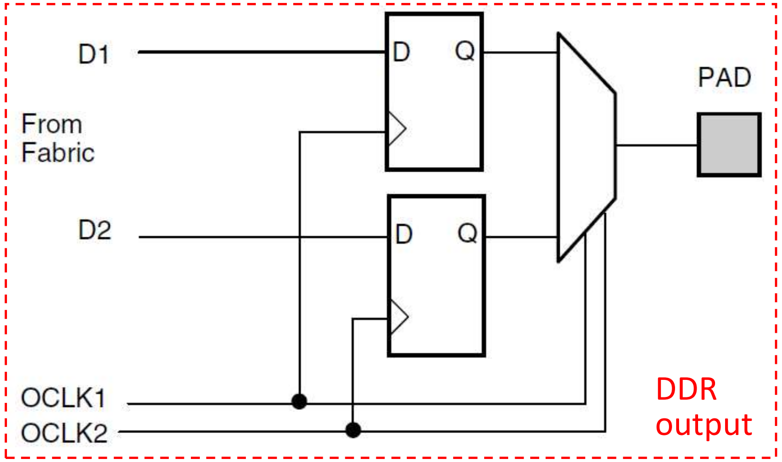
- Memory controller connects the processing cores to the main memory
  - Typical main memory is composed of DRAM chips
  - Crucial to data-intensive applications

- Types of DRAM chips
  - Single Data Rate (SDR) – old DRAM that handles one transaction per DRAM clock cycle, up to 133 Mhz
  - Double Date Rate (DDR) – modern DRAM that handles two transactions per DRAM clock cycle, DDR4 goes up to 1.6GHz
    - Note that DDR4-3200 is clocked at 1.6GHz, with 2 transactions per clock cycle, hence the rate is “3200”

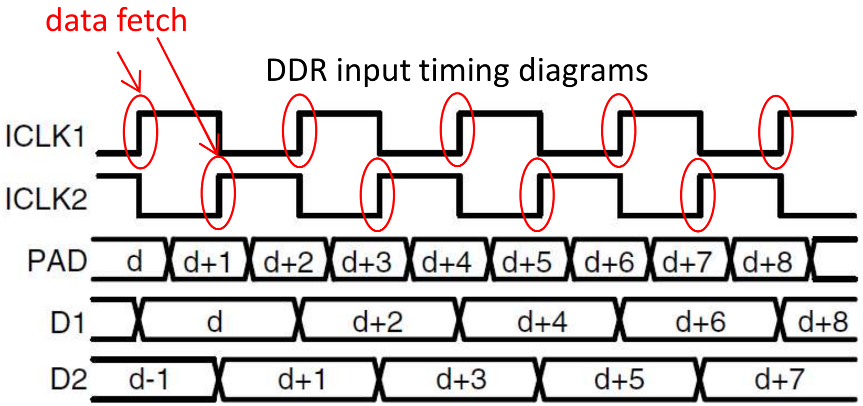
DDRx Memory Controller (1/2)

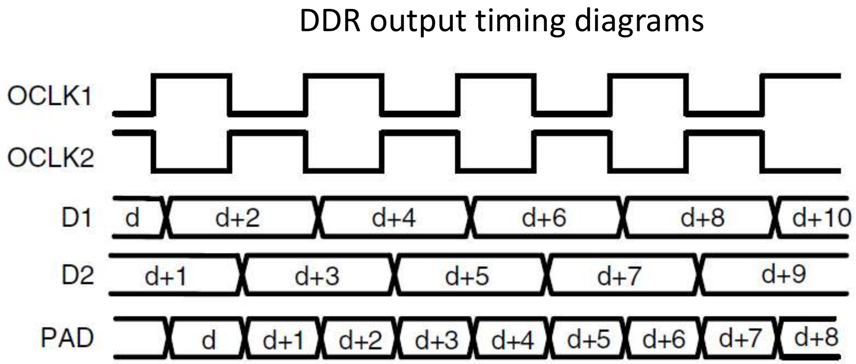
- We can design a low-speed DRAM controller and connect it to a DRAM chip with generic FPGA user pins





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 depend 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 up to DDR4-3200

DRAM Clocks on Arty

- The DRAM Chip on Arty is *Micron MT41K128M16JT*
  - A 16-bit DDR3-1600 IC
  - Under-clocked at 333.333 MHz (equivalent to a DDR3-667)
  - The memory controller is the Xilinx MIG IP
  - The MIG clock to the user-logic interface (UI) can be the DRAM clock divided by 2 or 4

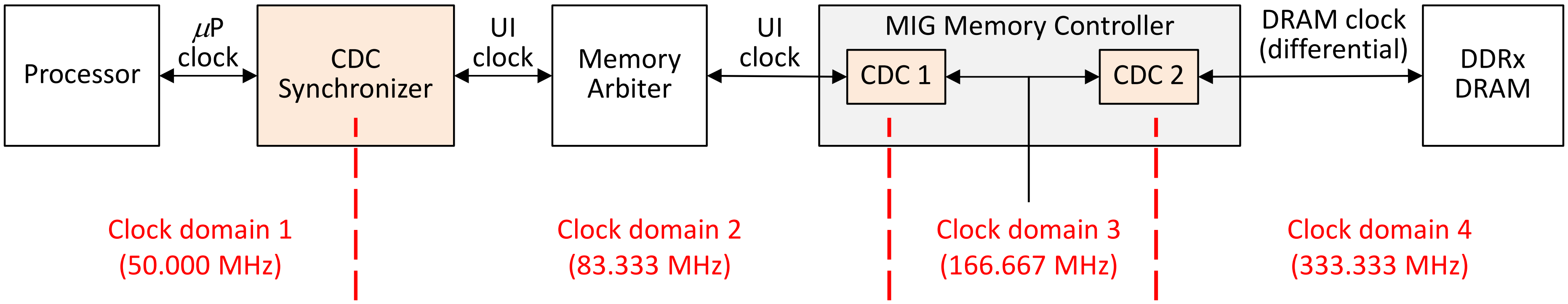
- 333.333/4 = 83.333 is still too high for Aquila on Arty!
  - We choose to use 50.0 MHz for the Aquila core → Need clock-domain crossing (CDC) to bridge the two domains

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
  - Native interface:
    - Close to the real DRAM chip interface
    - More efficient to use
    - Must handle block-based access and DDR3 DRAM re-ordering
- 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 (cache line size is 128-bit)
  - DRAM re-ordering is not necessary on Arty since the cache line size matches the DRAM block size

Clock Domain Crossing of MIG

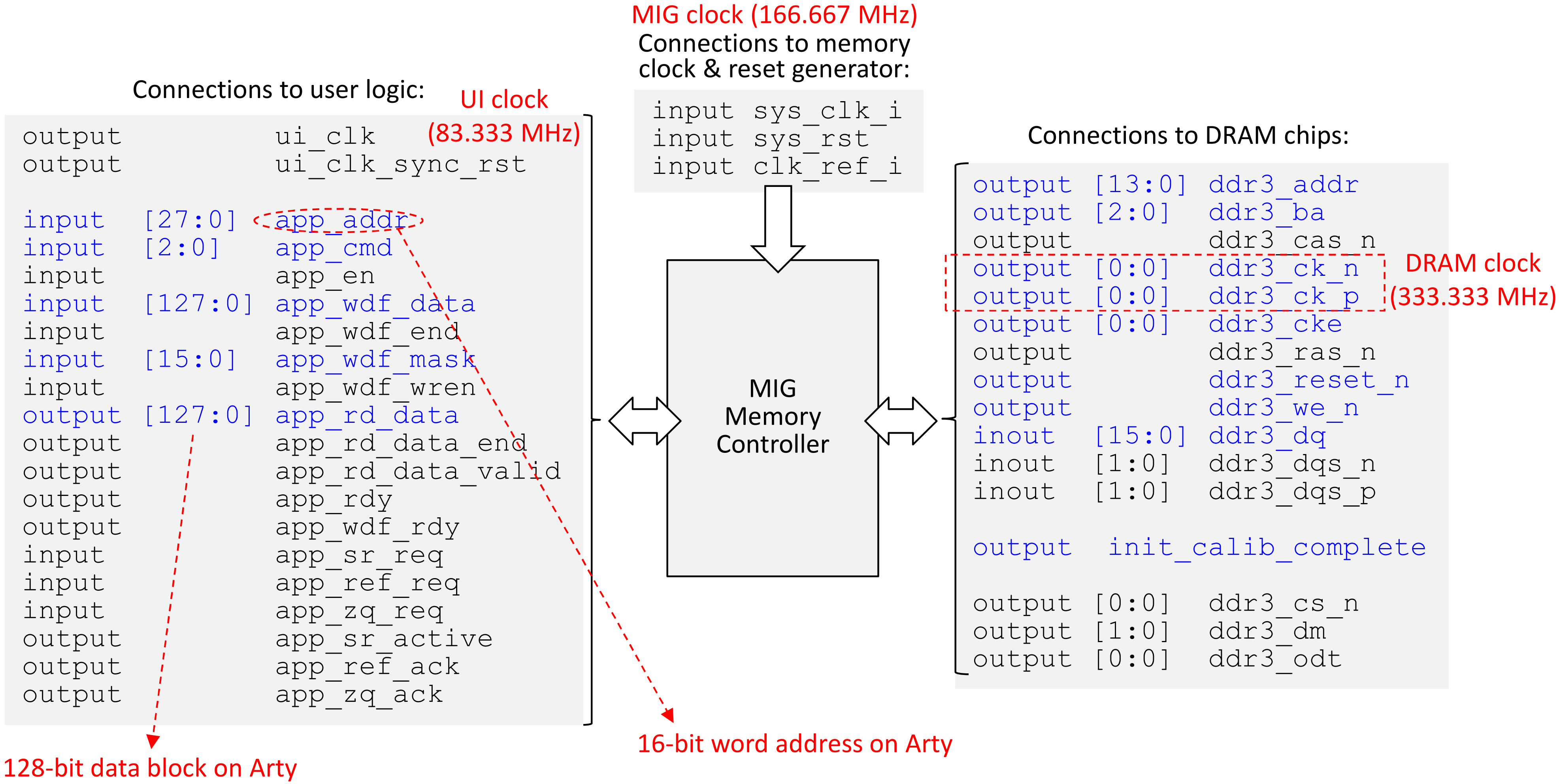
- The MIG controller itself contains 3 clock domains



- 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 one row at a time, each read/write operation is for a entire row of bit 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 DDR3 Read 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 on Arty):

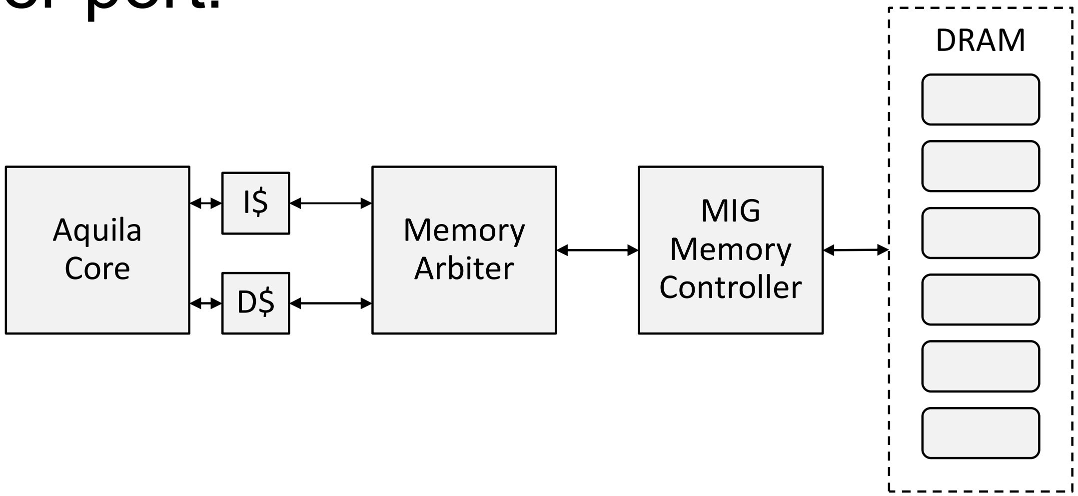
```
always @(posedge clk_i) begin
    if (rst_i) read_data  <= {128{1'b0}};
    else if (read_data_valid_i)
        case(addr_o[2:0])
        3'h0: read_data  <= {word7, word6, word5, word4, word3, word2, word1, word0};
        3'h1: read_data  <= {word6, word5, word4, word7, word2, word1, word0, word3};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

2-to-1 Memory Arbitration

- Since Aquila has two memory ports (I-Mem & D-Mem) but MIG has only one-port user-logic interface, 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 for specifically the π program
  - Use the unmodified 4KB D$ or 8KB D$ as the baseline, and see how you can improve the speed by changing the design
- Write a report (3 pages at most):
  - Analyze the data cache behavior for the π program
  - Describe the improvements you have tried
- 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 a good grade!