HW#3 Cache Optimization

Chun-Jen Tsai
National Chiao Tung University
11/9/2021

Homework Goal

- Cache is crucial for the memory performance of a microprocessor. In this HW, we use a π computing program as a benchmark to study the optimization of the data cache

- Your tasks:
  - The program compute π to 1000 digits
  - Analyze the cache behavior and try to improve the cache
  - The data cache size should be set to 2KB. You can only change cache design, not cache size

- Upload your code & report to E3 by 11/29, 17:00.

Aquila SoC with DRAM Support

- For this homework, download the SW `pi.tar.gz` and the HW `aquila_dram_build.zip` from E3:

```
aquila_dram_build -+- src -+- core_rtl/
                   |       +- mem/            Configuration file(s) for
                   |       +- mig/            Xilinx memory compiler
                   |       +- sim/
                   |       +- xdc/            Clock-domain crossing FIFO
                   |       +- cdc_sync.v
                   |       +- mem_arbiter.v   Instruction/memory access arbiter
                   |       +- soc_top.v       for single-port DDR3 main memory
                   |       +- uart.v
                   |
                   +- build.tcl               Again, run this script to create the workspace
```

Some Simple Statistics

- Aquila has a pair of 4-way set associative I/D-caches. When cache sizes are set to 2KB, the logic usage is
  - 70% usage of LUT
  - 35% usage of FF
  - 64% usage of BRAM

- Using the linker script to put code/data/heap/stack in either TCM or DRAM, the π computing time are:
  - Running on TCM: 804 msec
  - Running on DRAM, 4KB caches: 866 msec
  - Running on DRAM, 2KB caches: 1082 msec
  - Running on DRAM, 1KB caches: 1473 msec

About Compiling the π Program

- Note that to support DRAM, we changed the clock rate of Aquila to 41.666 MHz, so we have to redefine the frequency macro in `time.h`:

```
#define CPU_FREQ_MHZ   42      /* 42 MHz */
```

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

```
#define NDIGITS      1000
```

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

Linker Scripts of the π Program

- There are two linker scripts of the `pi.c` program, `pi_tcm.ld` and `pi_ddr.ld`.
- To build the DRAM version of `pi.ebf`, use the following commands to create a link of `pi.ld`:

```
$ ln -s pi_ddr.ld pi.ld
$ make
```

- To build the TCM version, type:

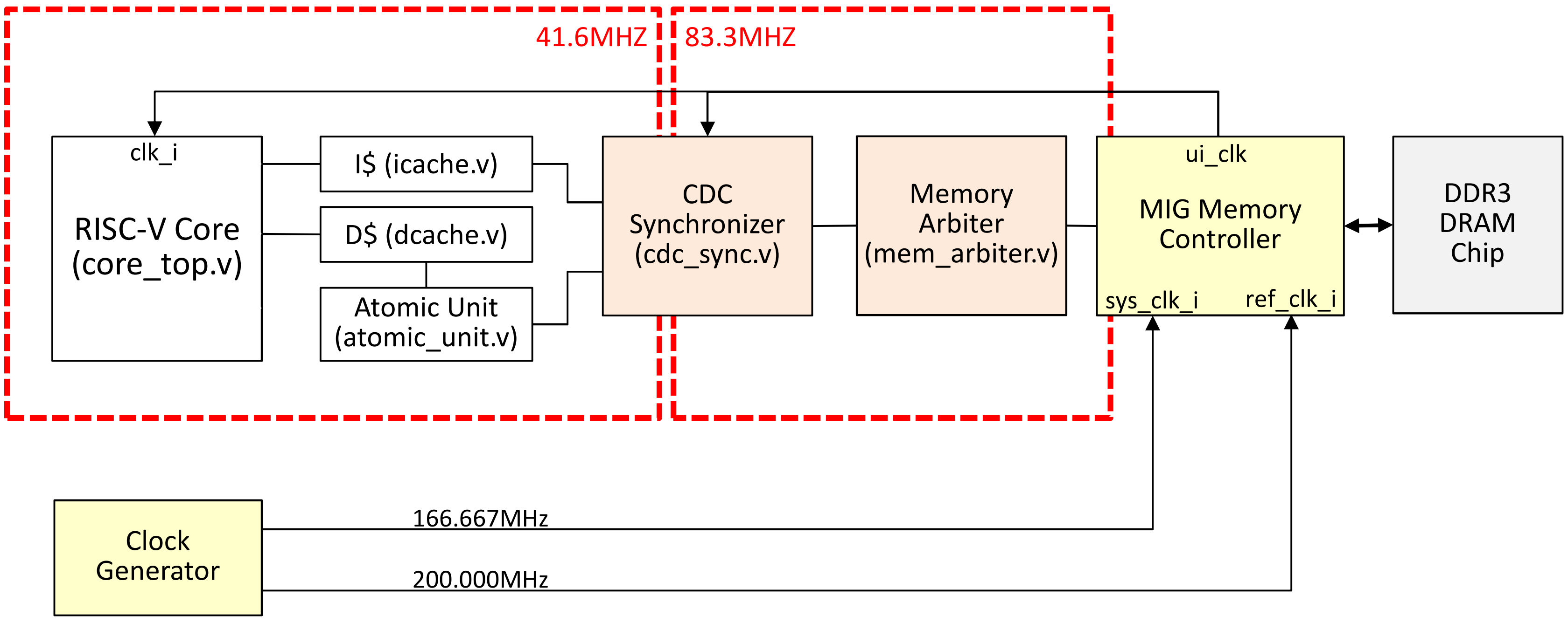
```
$ ln -s pi_tcm.ld pi.ld
$ make
```

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 under-clocked at 333MHz (equivalent to a DDR3-667)
  - The memory controller we use is developed by Xilinx, called MIG controller
  - To support 333Mhz DRAM clock, the MIG must run at 333/4 = 83.333 MHz or 333/2 = 166.667 MHz
- On the other hand, Aquila on Arty cannot be clocked at higher than 50MHz
  - We choose to use 83.333/2 = 41.666 MHz to simplify the clock-domain crossing (CDC) issue

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:



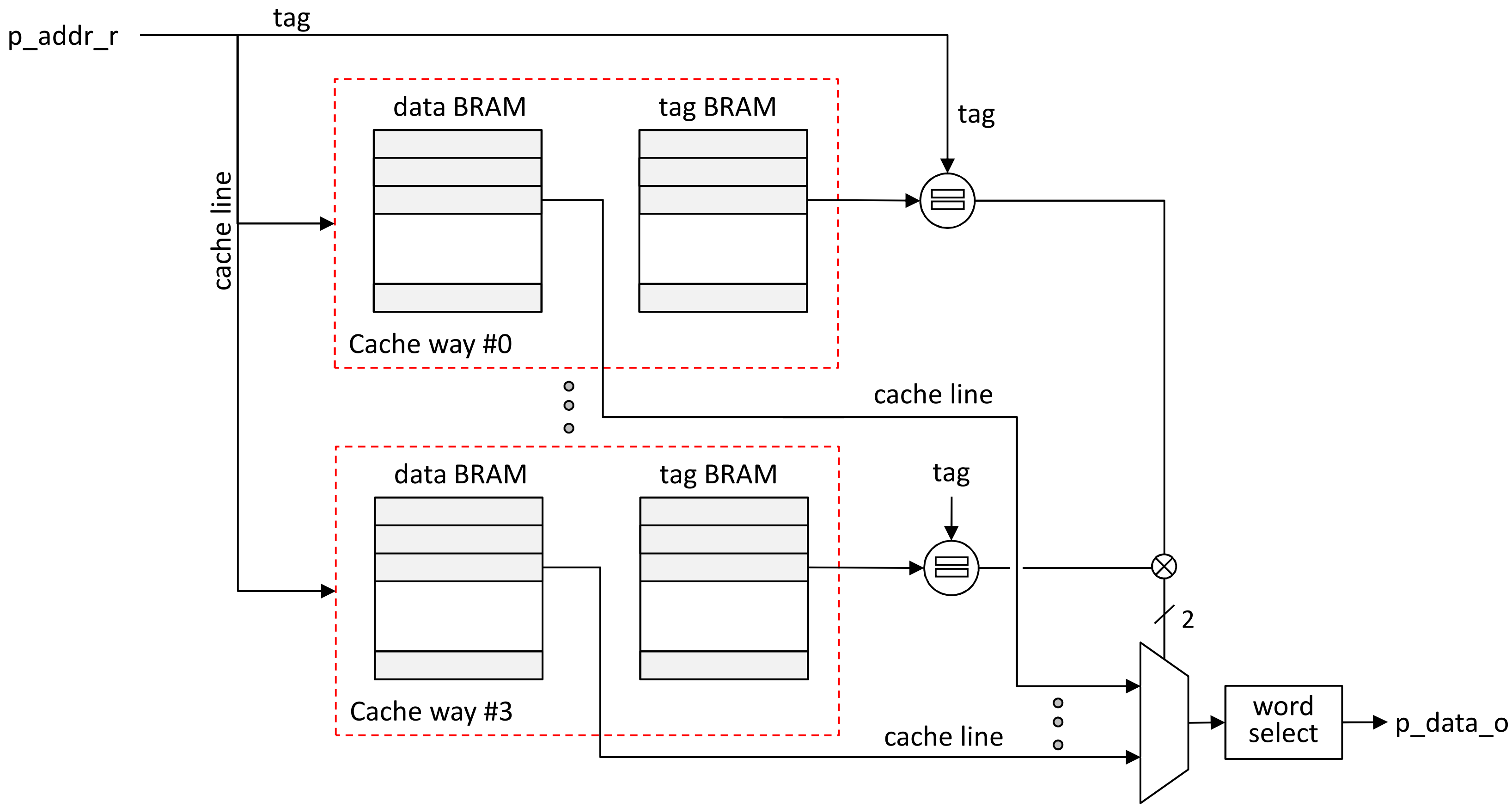
Handling Aquila Memory Requests

- Notes on D-Cache:
  - Unlike TCM, a request to data cache can take multiple cycles
  - Aquila uses single-cycle strobe signals for memory requests
  - Therefore, `p_addr_i` must be registered by `p_addr_r` at the strobe cycle inside the data cache (`dcache.v`) for multi-cycle processing

- Notes on I-cache:
  - An instruction could have been returned by the I-cache in the same clock cycle of strobe upon cache hit, however, we don’t do it this way
  - The returned instruction is intentionally delayed till next clock edge to match the behavior of TCM

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];
```

- 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):

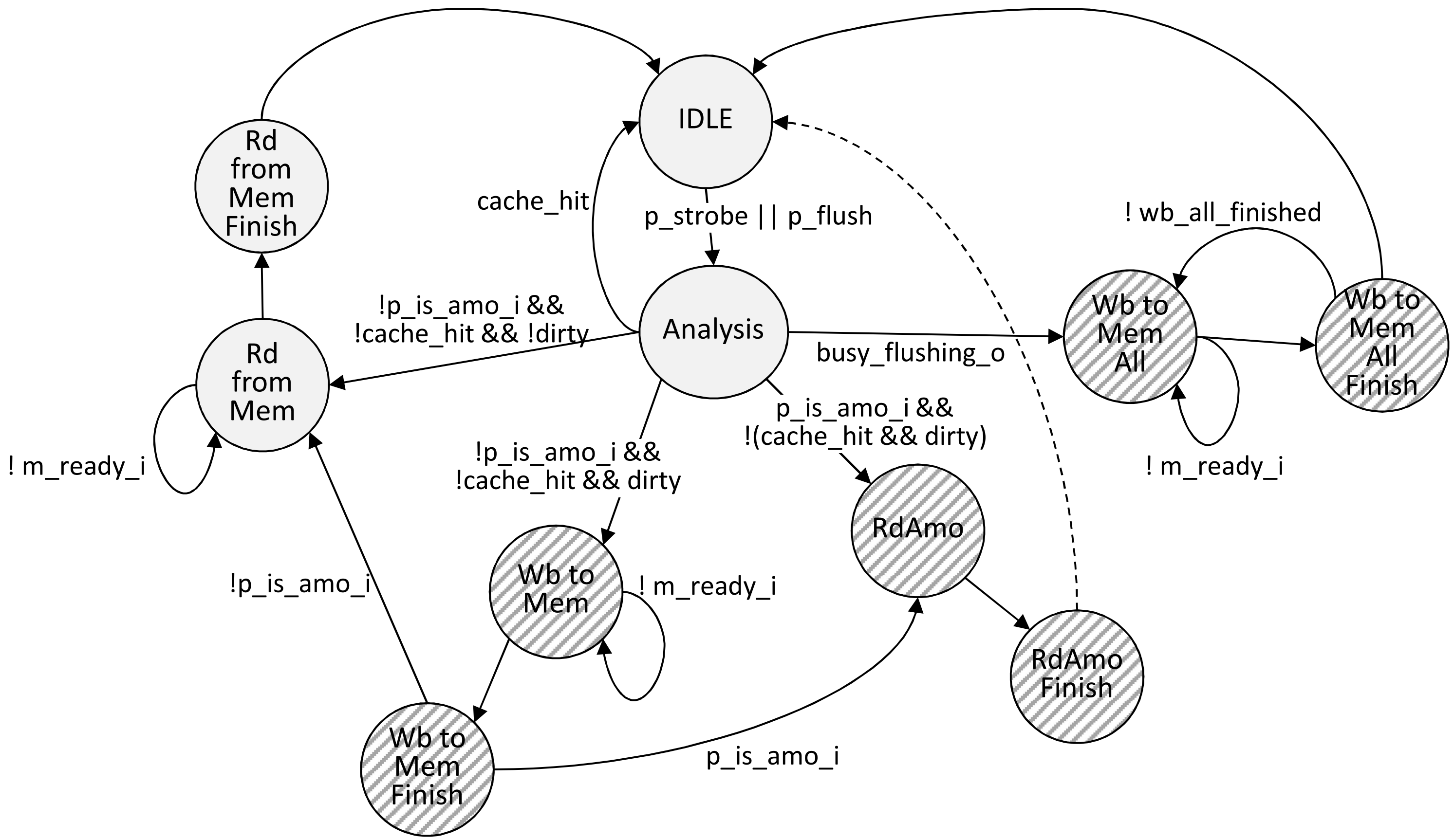
```
(* ram_style = ”block” *) reg [0:31] my_ram [127:0];
```

    - 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`.

Things You Can Try

- There are a few things you can try to improve the D-cache performance of Aquila:
  - Change cache ways (8-way cache are worth trying)
  - Change the cache replacement policy
  - Applying good pre-fetching algorithm
  - Redesign the cache controller based on the statistics you have collected

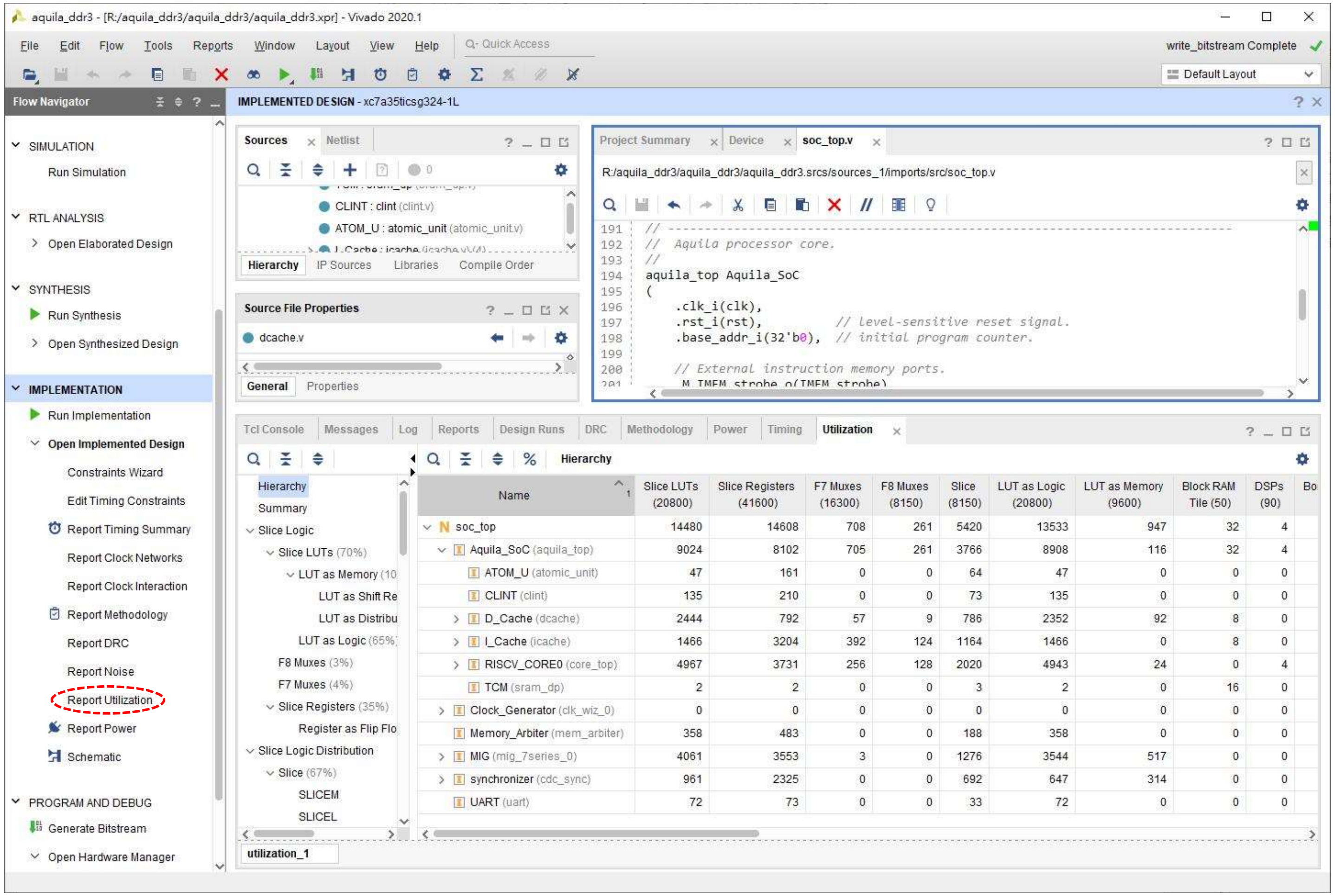
Warning on Using ILA

- For this homework, you should use ILA for debugging because it is difficult to simulate DDR memory at system level

- Note that ILA uses on-chip memory to capture data. If you need to capture a lot of data, you may have to reduce the cache sizes of TCM to save more BRAMs for ILA

Resource Usage on the FPGA

- Checking FPGA utilization after implementation:



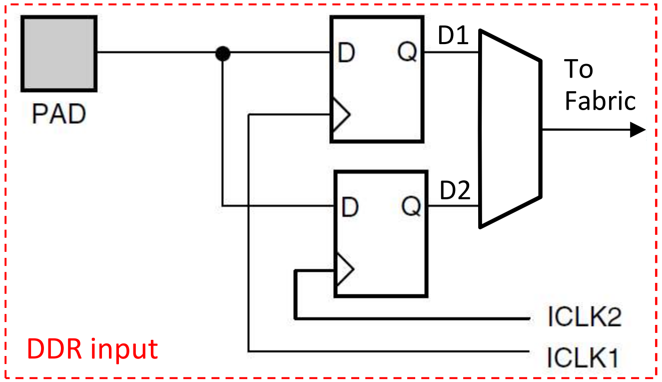
Memory Controller

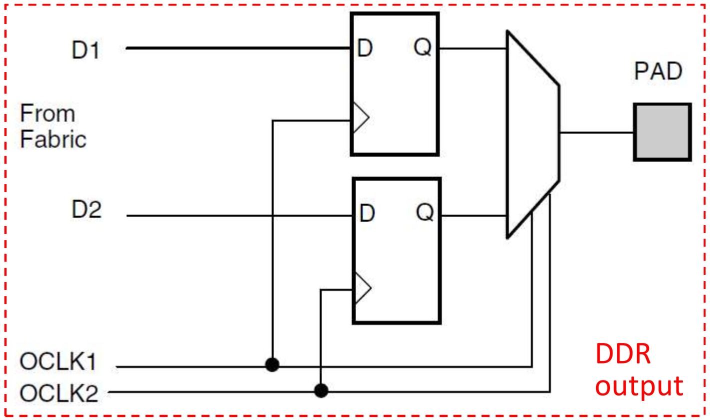
- Memory controller connects the processing cores to the main memory
  - Typical main memory is composed of DRAM chips
  - Crucial to data-intensive applications

- 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

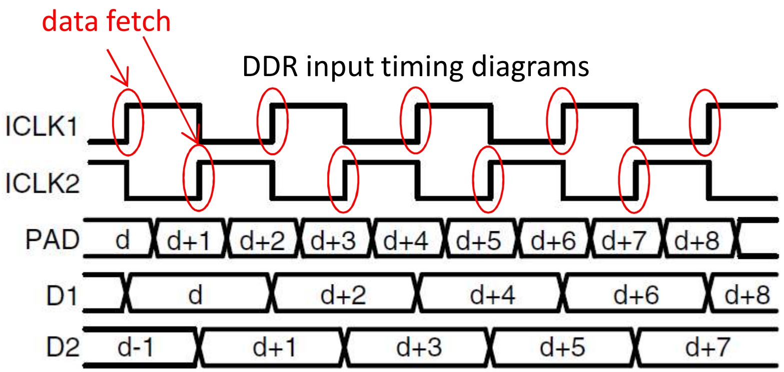
DDRx Memory Controller (1/2)

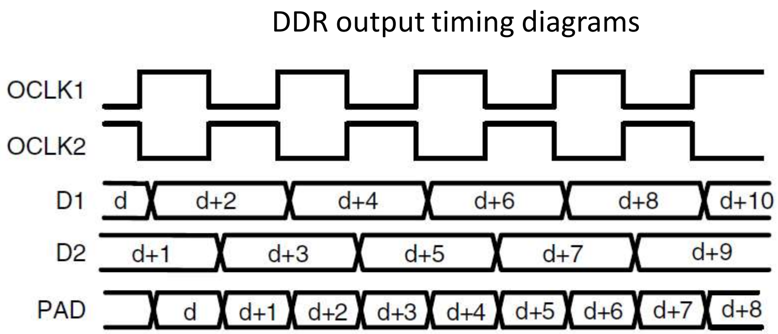
- We can design a low-speed DRAM controller and connect it to a DRAM chip with generic FPGA user pins





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

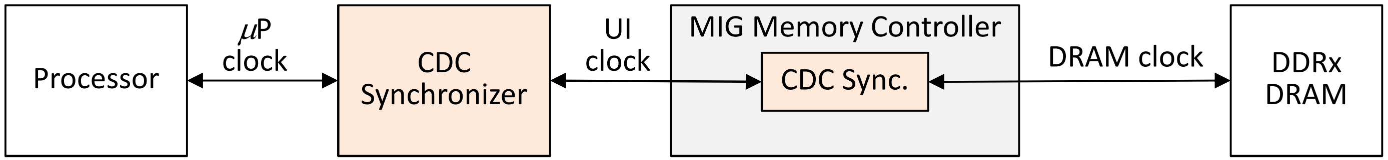
MIG Interface on Processor Side

- MIG support two types of processor side interface:
  - 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, 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

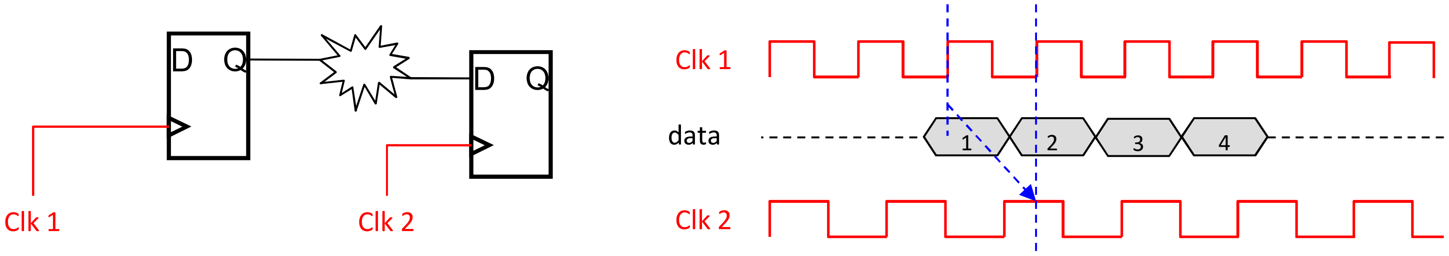
Clock Domain Crossing (CDC) Design

- 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:



CDC Issues

- When two flip-flops are driven by different clocks, data exchange can cause problems (a.k.a. meta-stability)



- Solutions:
  - Dual flip-flops or hand-shaking – for single-beat data transfer
  - Asynchronous FIFOs† – for burst data transfer

- We use Xilinx async. FIFOs in our CDC synchronizer

† C. E. Cummings and P. Alfke, "Simulation and Synthesis Techniques for Asynchronous FIFO design," Synopsys Users Group Meetings, San Jose, CA, 2002.

DRAM Native Interface

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

Connections to user logic:

```
output          ui_clk
output          ui_clk_sync_rst

input   [27:0]  app_addr
input   [2:0]   app_cmd
input           app_en
input   [127:0] app_wdf_data
input           app_wdf_end
input   [15:0]  app_wdf_mask
input           app_wdf_wren
output  [127:0] app_rd_data
output          app_rd_data_end
output          app_rd_data_valid
output          app_rdy
output          app_wdf_rdy
input           app_sr_req
input           app_ref_req
input           app_zq_req
output          app_sr_active
output          app_ref_ack
output          app_zq_ack
```

Connections to memory clock & reset generator:

```
input sys_clk_i
input sys_rst
input clk_ref_i
```

MIG Memory Controller

Connections to DRAM chips:

```
output  [13:0]  ddr3_addr
output  [2:0]   ddr3_ba
output          ddr3_cas_n
output  [0:0]   ddr3_ck_n
output  [0:0]   ddr3_ck_p
output  [0:0]   ddr3_cke
output          ddr3_ras_n
output          ddr3_reset_n
output          ddr3_we_n
inout   [15:0]  ddr3_dq
inout   [1:0]   ddr3_dqs_n
inout   [1:0]   ddr3_dqs_p

output  init_calib_complete

output  [0:0]   ddr3_cs_n
output  [1:0]   ddr3_dm
output  [0:0]   ddr3_odt
```

DRAM clocks

128-bit data block on Arty,
512-bit data block on KC-705

16-bit word address on Arty,
64-bit word address on KC-705

Block-based I/O of Memory Controller

- DRAM chips typically operates on a row basis, each read/write operation will be on 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

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 fetch has higher priority over data access

Your Homework

- The goal of this homework is to improve the memory subsystem by modifying the D-caches
- Note that you shall keep the data cache size to 2KB.

- Write a report:
  - Describe how you collect the memory operation statistics and analyze the results
  - Describe your improvements to the data cache