Computer Organization

Direct Memory Access

Jinha Jeong (arch23-ta@hpcs.snu.ac.kr)
High Performance Computer System (HPCS) Lab
May 22, 2023

Goals

- Understand DMA and its impact on performance
- Understand how devices communicate with the CPU and the memory
- Implement a simplified DMA logic on your pipelined CPU with cache

Peripheral Devices

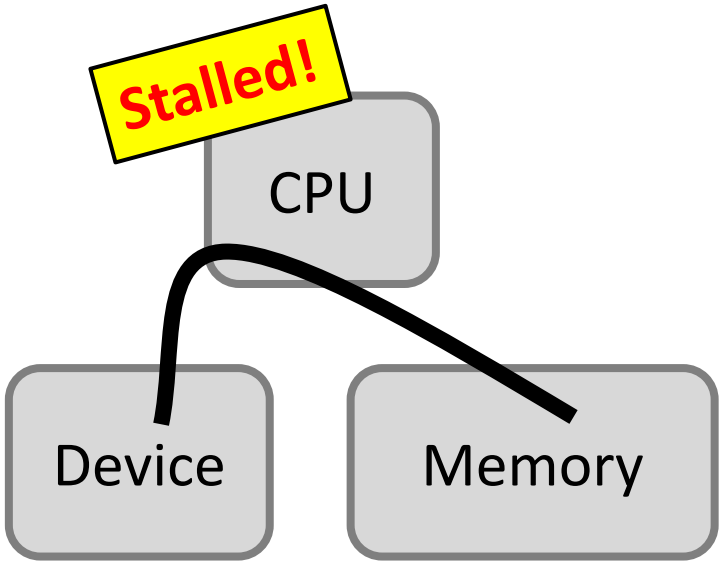
- A computer has lots of peripheral devices (e.g., HDD, GPU, network interface card (NIC)) for various purpose (e.g., transfer data to the memory)

- Communication mechanisms between CPU
  - A CPU needs to communicate with a device to use its functions (e.g., save the data in an SSD)
  - Either the CPU or the device can initiate the communication (e.g., Data save in an SSD by a CPU, data receive notification from a network card)

Communication Details

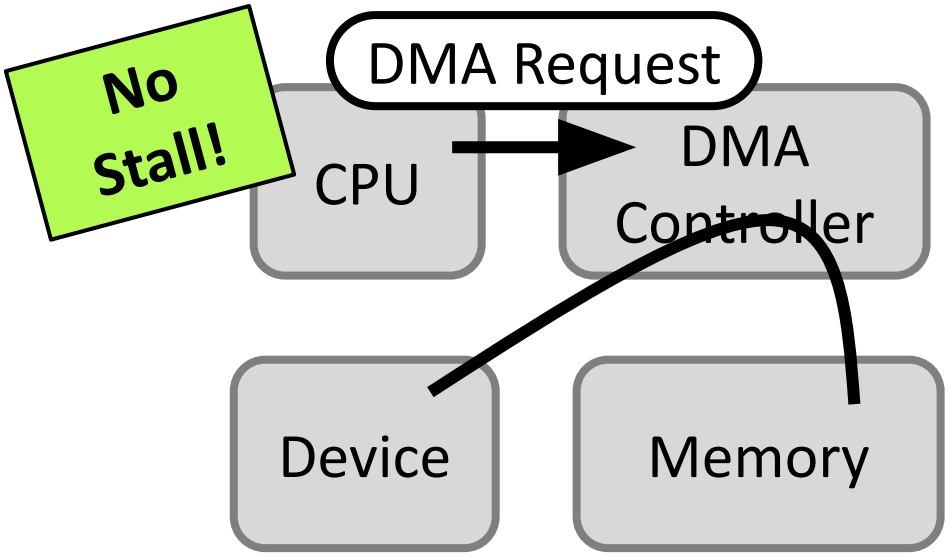
- Initiated by the CPU
  - The device driver running on the CPU writes communication information via memory mapped I/O
  - The device **decodes the information** and **fetches the data**
  - The device notifies the end of the communication through “interrupt”
- Initiated by the device
  - The device notifies that it needs to communicate with the CPU through “interrupt”
  - The CPU catches the interrupt and the interrupt handler calls the device driver
  - The **device driver does what needs to be done** (e.g., copy the data from the network card to the memory)

Naïve Data Transfer



- The CPU must **perform load/store instructions for every word** to transfer data from the I/O device to the memory.
  - The CPU reads data from the I/O device and write them to the memory.
  - As a result, the CPU is busy to perform the transfer and **cannot proceed its own work**.

Direct Memory Access (DMA)



- DMA decouples I/O-to-memory data transfers from other instructions.
  - The **CPU requests** I/O-to-memory data transfers to the DMA controller.
  - The **DMA controller** is responsible for **transferring data** from the device to the memory.
  - The **CPU executes other instructions** while the DMA controller transfers data.

Simple DMA Design

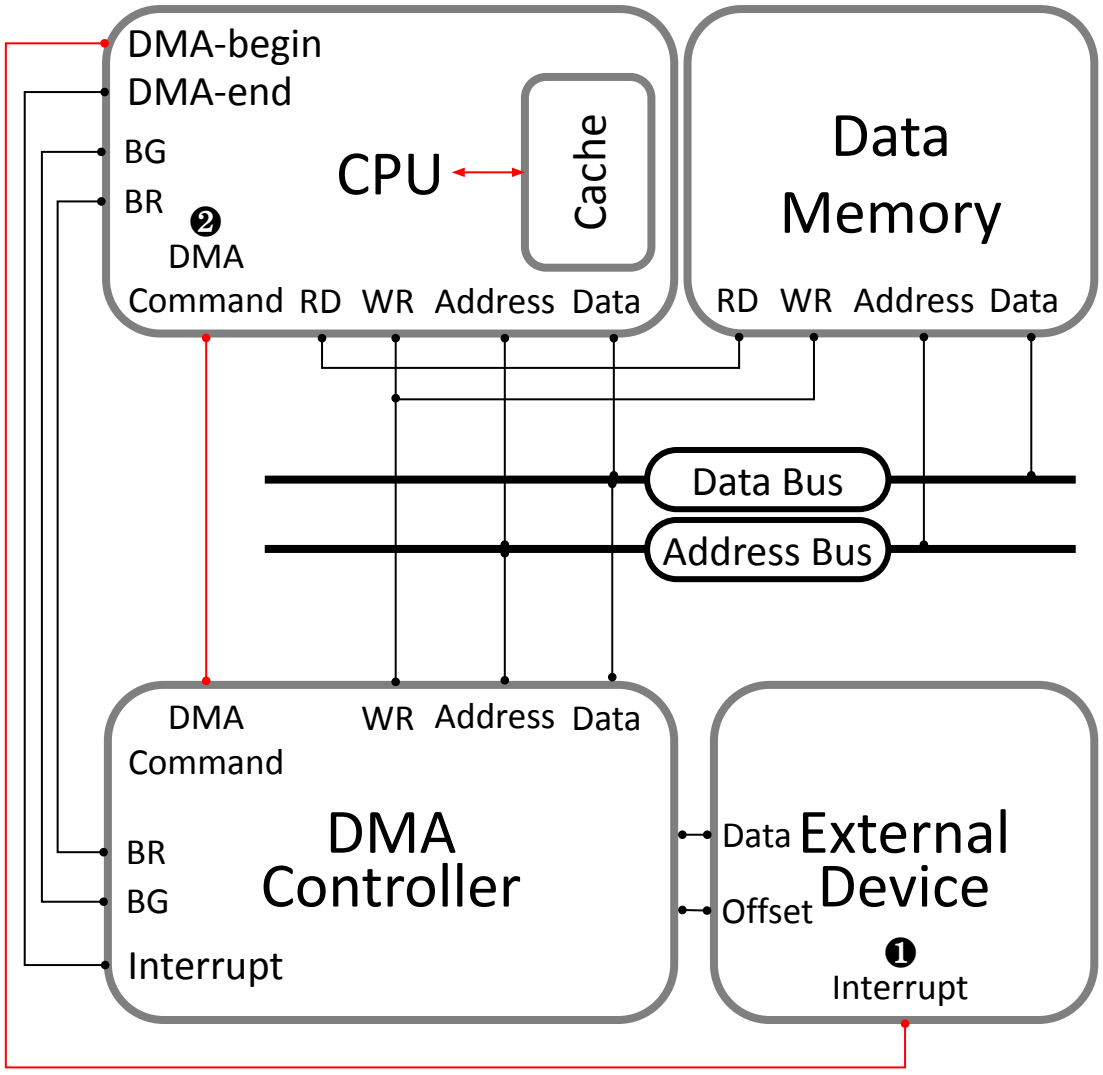
- We will use a simple DMA design for education
  - **Fixed address** and **fixed size data** transfer
  - Only **the device initiates** the data transfer
  - Support **memory write** only (not read!)

- Actual DMA is very complicated
  - Device drivers running on CPU generate device-specific commands for variable address and length
  - A device can read/write from/to the memory.
  - Multiple devices can be involved.
  - The device interrupts happen when the data transfer is done, and does not when the initiation.
  - The device writes the data and knows where to write.
  - …

Simulation Environment

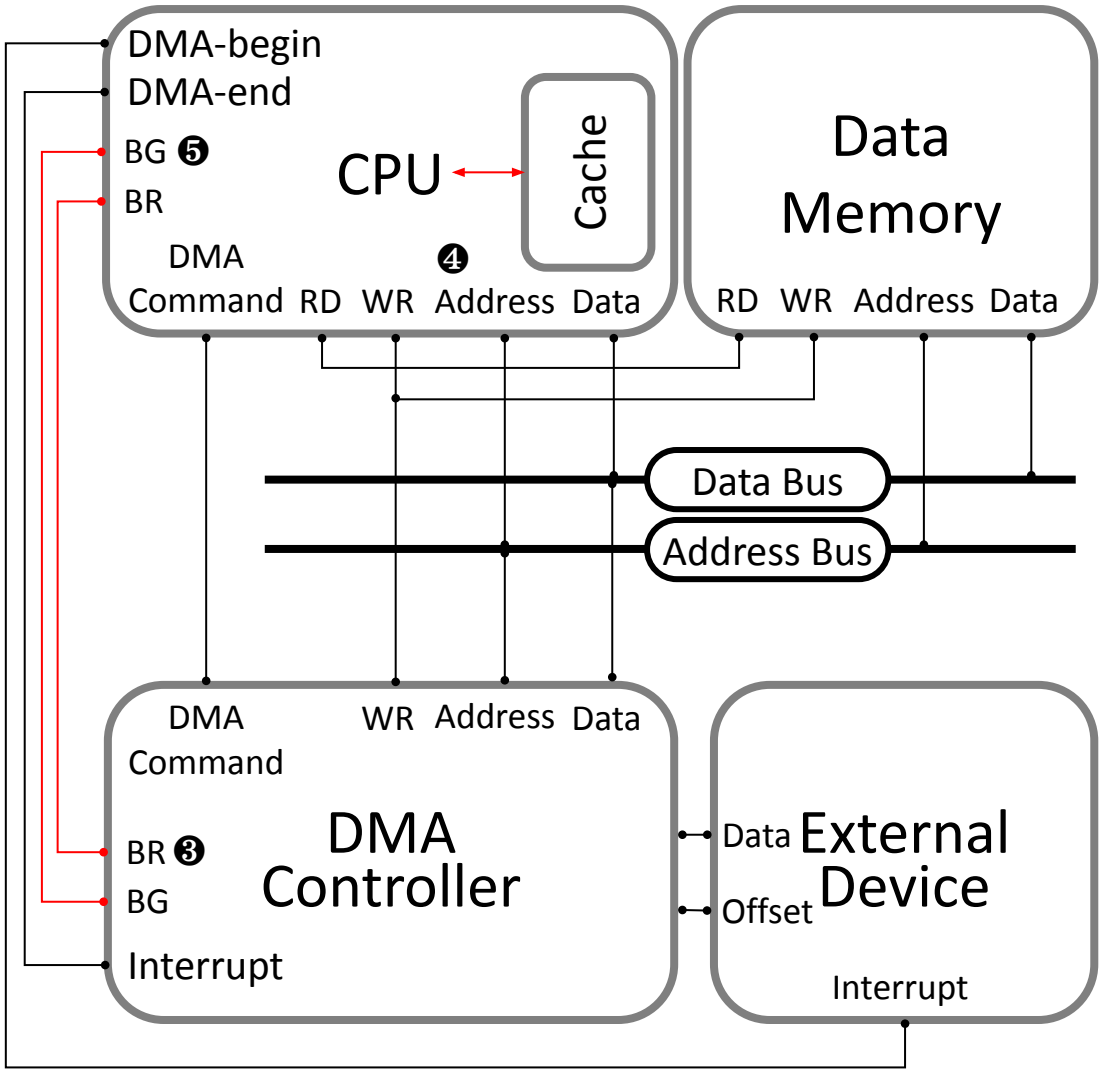
- A virtual external device (external_device.v)
  - An external device has **randomly-generated 12-words data** (i.e., total 24 bytes).
  - When the device wants to write the data in the memory, it interrupts the CPU.
- Four-words bandwidth memory
  - Three memory transactions are required (i.e., 4 word write x 3 times).
- The DMA controller arbitrates data transfer between memory and the device.

DMA Read/Write Transaction



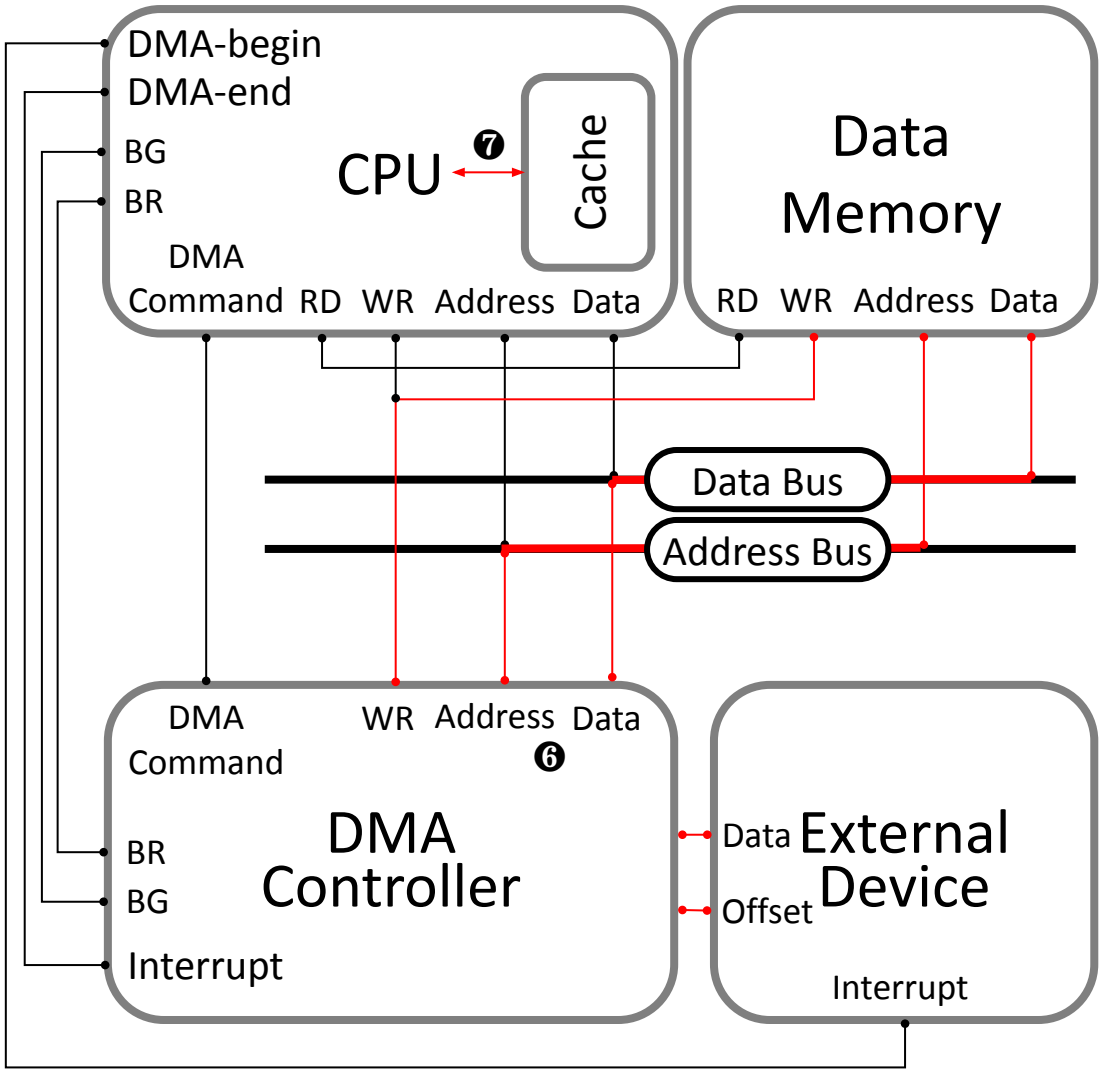
1. The **external device** generates an interrupt. The **CPU** must be ready for handling DMA-begin and DMA-end interrupts.
2. The **CPU** sends a command (address, length) to the **DMA controller**.
   - **Address**: the target memory address of DMA transactions, **0x1F4 (fixed address in this project)**
   - **Length**: the length of data, **12 words (fixed length in this project)**

DMA Read/Write Transaction



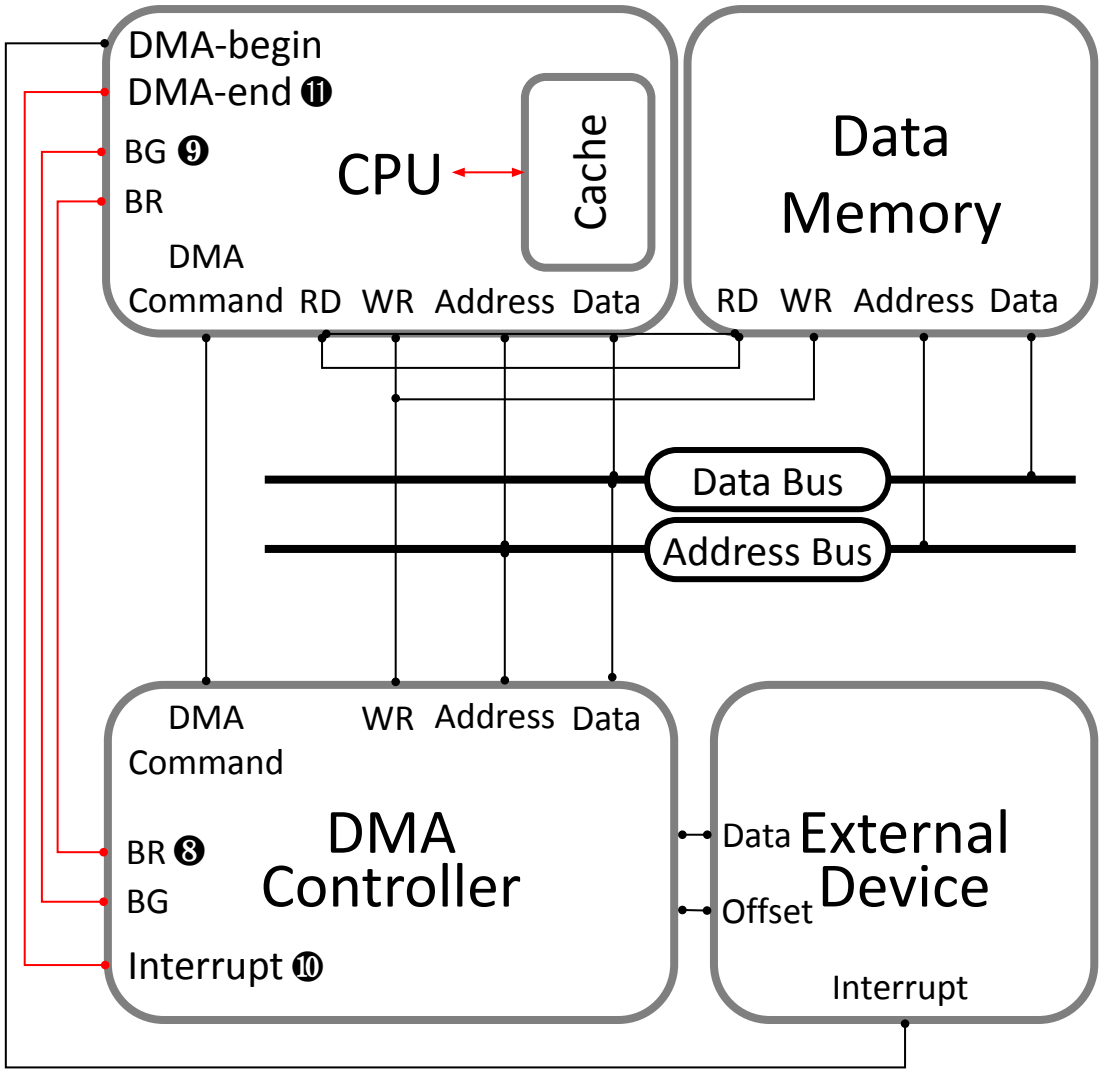
3. The **DMA controller** raises the Bus Request (BR) signal.
4. When it receives the BR signal, the **CPU** must stop using address/data buses and RD/WR signals of **data memory or ports (i.e., d_readM, d_writeM, d_address, d_data)**.
5. After that, the **CPU** raises the Bus Granted (BG) signal.

DMA Read/Write Transaction



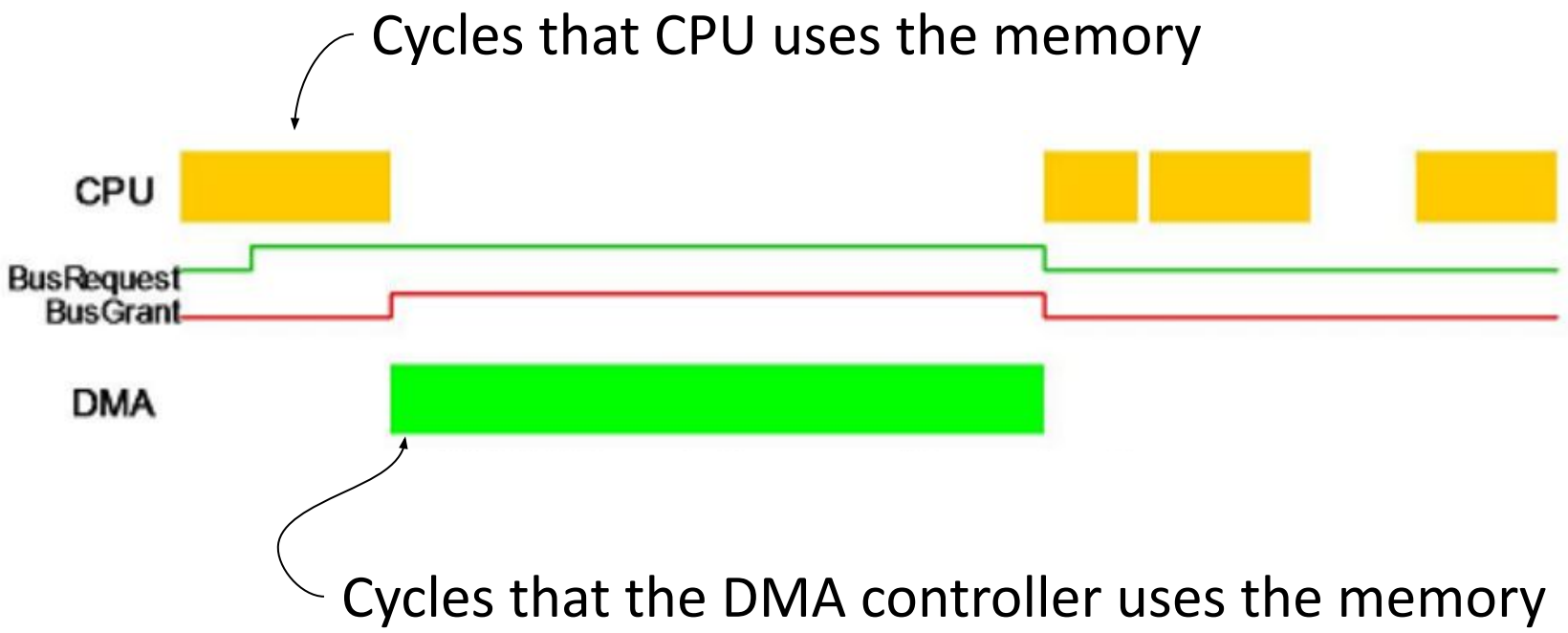
6. After receiving the memory bus grant, the **DMA controller** performs DMA write transactions.
   - DMA write: the DMA controller read 12-word data from the external device and writes data to the target address.
7. At that time, a **CPU** should run with the data on its cache. It should be blocked when a cache miss occurs.

DMA Read/Write Transaction



8. The **DMA controller** clears the BR signals when it finishes data transfers between the device and memory.
9. The **CPU** clears the BG signal and enables using memory buses.
10. The **DMA controller** generates the DMA-end interrupt
11. The **CPU** handle the interrupt.

Example Simulation Results



Required Implementation - CPU

- **Interrupt handler**
  - Two interrupt pins: DMA-begin, DMA-end
  - The CPU must stop and handle the interrupts immediately
- **DMA command generation**
  - After receiving the DMA-begin interrupt, the CPU generates a DMA command and sends it to the DMA controller
- **Data memory bus arbitration**
  - The DMA controller should be granted the data memory bus when it raises the Bus Request (BR) signal
  - The CPU cannot use data memory when the DMA controller is using the memory buses. However, the CPU can access both instruction and data caches
  - After the DMA controller releases the memory bus, the CPU processes stalled memory operations immediately
  - *Instruction memory or instruction ports are not involved in DMA in this project*

Required Implementation - DMA

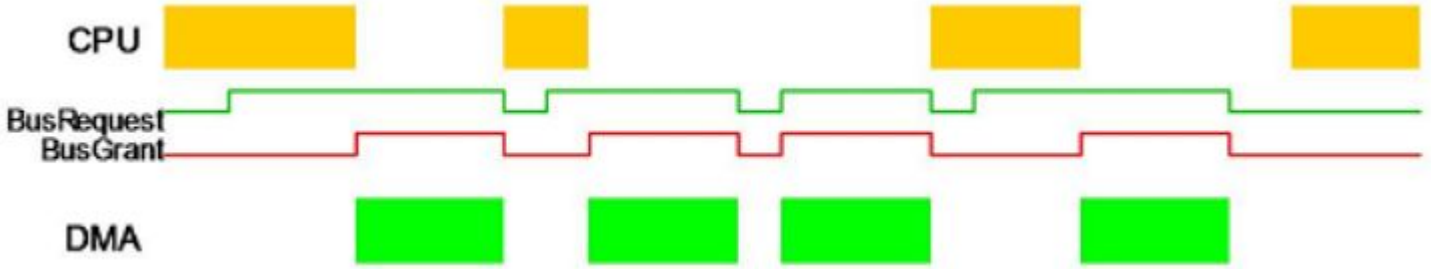
- **DMA Controller**
  - The DMA controller should receive and parse a DMA command from the CPU, and get the memory buses exclusively using Bus Request and Bus Granted signals
  - After that, the DMA controller reads the data from the external device and writes the data to the memory
  - The DMA controller releases the data memory buses after all DMA transactions are done
  - *DMA target address (0x1F4) and length (12 words) are fixed in this project*

Grading

- **Be still functional**!
  - Your CPU should pass every test in the previous testbench.
  - Its functional correctness must not be affected by the DMA.
- Handle external signals and interrupts properly at the right time
- The CPU should be working while the DMA is on-going unless the cache miss happens
- After the DMA-end interrupt, the device's data should be read from the designated memory address
- **YOU SHOULD PROVE THE CORRECTNESS BY PROVIDING THE PROPER WAVEFORMS TO GET THE PERFECT SCORE**

Extra Credit: Cycle Stealing

- After send 4 words, the DMA engine releases the bus.
- Then, the stalled-CPU can execute instruction.
- If the CPU does not use the memory bus, the DMA controller should retake the bus as soon as possible
- Compare the performance to the case without cycle stealing



Summary

- Understand what DMA is
- Implement the **simplified DMA logics** in Verilog
- Be careful about the **required timing constraints**
- (Optional) Cycle stealing