Need of DMA:

1. DMA acts as coprocessor to main memory for time consuming tasks.
2. Processor configures the DMA to perform the transfer of data from src to dest.
3. DMA informs the processor about status of transfer via polling or interrupt mode.
4. DMA do not process the data. It only transfers the data from one place to another.

DMA acting as Co-processor:

• One way of reducing the burden on the processor is to use Direct Memory Access (DMA) to perform memory transfers.
• Using this approach, the processor issues a memory transfer request to the DMA controller, which will then perform the memory transaction.
• This allows the processor to perform other tasks while the DMA controller performs the transfer.
• In this situation, the DMA controller acts as both a bus master and a bus slave.
• As a master, the DMA controller communicates with the memory controller while arbitrating for the bus.
• While acting as a slave, the DMA controller sets up memory transfers by responding to requests from bus masters (the processor, in most cases).

How DMA controller acts as both a bus master and a bus slave?

• As a master, the DMA controller communicates with the memory controller while arbitrating for the bus.
• While acting as a slave, the DMA controller sets up memory transfers by responding to requests from bus masters (the processor, in most cases).

Two Bus cycles of DMA:

• Each DMA cycle will typically result in at least two bus cycles: either a peripheral read followed by a memory write or a memory read followed by a peripheral write, depending on the transfer base Addresses.

DMA for efficient use of interrupts:

• DMA permits the peripheral, such as a UART, ethernet to transfer data directly to or from memory without having each byte (or word) handled by the processor.
• Thus DMA enables more efficient use of interrupts, increases data throughput.
• It may lead to reduction in hardware costs by eliminating the need for peripheral-specific FIFO buffers.

Snoop Control Unit:

1. Responsible for data coherency between L1 cache, L2 cache and DDR memory.
2. Access and priority to L2 cache by the two cores of processor is controlled by SCU.
3. ACP accesses L1,L2,DDR and OCM via SCU.

ACP→ Memory mapped
HP→ Memory mapped
GP→ Lite

Question 2. Explain the term simple transfer in the context of the DMA. Explain the importance of order of executions of two DMA transactions given below.

```
// Simple DMA Transfers
status=XAxiDma_SimpleTransfer(&AxiDMA,(UINTPTR)FFT_output_hw,(sizeof(float complex)*N),XAXIDMA_DEVICE_TO_DMA);
status=XAxiDma_SimpleTransfer(&AxiDMA,(UINTPTR)FFT_input,(sizeof(float complex)*N),XAXIDMA_DMA_TO_DEVICE);
```

5 Marks

DMA has two modes of operation: 1) Simple or single cycle mode and 2) Scatter Gather mode. Simple transfer in DMA is used when CPU wants to transfer single burst of data located on contiguous location.

1 Mark

The following code corresponds to two simple transfers performed by AXI DMA. The order is important due to backpressure issue. In the first transfer, CPU is configuring the DMA to receive the data from FFT. In the second transfer, CPU is configuring the DMA to send the data to FFT. If we initiate second transfer before the first transfer, FFT may not accept the input data from DMA since the DMA is not yet ready to accept the FFT output. Thus, we configure the DMA first to make it ready to accept the FFT output before configuring the DMA to send the FFT input.

Zynq Boot Process:

1. BootROM (Boot Read-Only Memory) - A part of the system that starts the boot process.
2. OCM (On-Chip Memory) - A small memory inside the chip used during the early stages of booting.
3. FSBL (First Stage Boot Loader) - A program that takes over after the BootROM to further set up the system.

You also need to discuss how the processors and programmable logic (PL) are configured during this process.

1. Processors Boot First

    - In Zynq SoC, the processing system (PS) (like the ARM cores) starts first, not the programmable logic (PL) (like the FPGA part).
    - This makes the boot process software-focused.
2. Multi-Stage Boot Process
    - The boot process happens in steps, starting with BootROM (stage 0), followed by the FSBL (stage 1).
3. Role of BootROM (1 Mark)
    - What BootROM Does:
        - It runs automatically when the system powers on.
        - Sets up one ARM core.
        - Reads the mode pins to know which device to boot from (like SD card or flash memory).
        - Copies the FSBL (First Stage Boot Loader) to OCM (On-Chip Memory).
        - Starts executing the FSBL.
    - What BootROM Doesn't Do:
        - It doesn't configure the PL (programmable logic).
4. Role of FSBL (2 Marks)
    - The FSBL takes over after BootROM.
    - What FSBL Does:
        - It initializes important parts of the PS, like memory and I/O (using data generated by Vivado tools).
        - Configures the PL if a bitstream is provided (this is optional).
        - Loads the next part of the software (e.g., second-stage bootloader or an application) into the main DDR memory.
        - Passes control to the next stage (either the second-stage bootloader or application).
    - Where FSBL is Stored:
        - It is usually stored in flash memory or downloaded via JTAG.

Explain the Zynq Boot process specifically highlighting the difference between the boot process using JTAG and SD/Flash memory-based approaches:

Key Differences Between JTAG and SD/Flash Boot

Feature	JTAG Boot	SD/Flash Memory Boot
Usage	Development/debugging	Production or standalone systems
BootROM Behavior	Skips boot source selection	Reads mode pins for boot source
Boot Image Storage	Loaded directly from JTAG tools	Stored persistently in SD/Flash
Configuration Speed	Faster for testing new builds	Slightly slower due to storage access
PL Configuration	Optional, typically handled manually	Automated via FSBL if configured
Persistent Boot	No	Yes

When to Use Each Method

1. **JTAG Boot**:
   - Ideal for testing and debugging software or hardware designs.
   - Example: Loading a bare-metal application directly for testing.
2. **SD/Flash Boot**:
   - Essential for deploying a fully functional standalone system.
   - Example: Booting a production-ready embedded Linux system or standalone application.

DDR and Flash Memory:

Comparison

Feature	DDR (RAM)	Flash Memory
Volatility	Volatile (data lost on power off)	Non-volatile (data retained)
Speed	Very fast for temporary storage	Slower for read/write
Usage	Runs programs and handles data	Stores programs and firmware
Example	Application execution memory	Bootloader and OS storage

In simpler terms:

- **DDR** is like a notepad you use while working—it's fast and temporary.

- **Flash memory** is like a notebook where you store things permanently for future reference.

Who (ARM or FPGA) has high priority memory access to On-chip and DDR memories? Explain.

On-Chip Memory (OCM) Access Priority

- OCM is small, fast memory located within the Processing System (PS).
- Priority: **ARM (PS) has higher priority access to OCM** by default because it is tightly coupled to the PS.
    - This ensures the processors can access critical boot data (like the FSBL) quickly and reliably during initialization.
    - The FPGA (PL) can access OCM only if explicitly configured to do so through the AXI (Advanced eXtensible Interface) interconnect.

DDR Memory Access Priority

- DDR is external memory shared by both the PS (ARM) and PL (FPGA).
- Priority:
    - By default, the ARM (PS) cores have higher priority for DDR access, as the PS manages critical system tasks, including running the operating system and applications.
    - The FPGA (PL) access to DDR is via AXI, and its priority can be configured based on system requirements.

Example:

- For real-time, high-speed data processing, you might configure the FPGA with higher priority for DDR access.
- For general-purpose computation or software execution, the ARM cores retain higher priority.

Differences Between AXI HP and ACP Ports:

Feature	AXI HP (High Performance)	AXI ACP (Accelerator Coherency Port)
Number of Ports	Up to **4 ports** are available.	Only **1 port** is available.
Data Bus Size	Wider bus: **64 or 128 bits** for large transfers.	Narrower bus: **64 bits** only.
FIFO Availability	Includes **FIFO buffers** to handle burst transfers.	No FIFO buffers, suited for small data requests.
Access to OCM	Direct access to **OCM** for large data transfers.	Indirect access to OCM through CPU's cache.
Access to DDR	Can directly access **DDR memory**.	Indirect access via CPU's cache.
Data Coherency	No cache coherency; must manage coherency manually.	Maintains cache coherency automatically.

When to Use AXI HP Ports:

AXI HP ports are preferred when:

1. **Large Data Transfers**:
   - Suitable for moving large amounts of data directly from DDR or OCM.
   - Example: Streaming video frames or processing large datasets.
2. **CPU Access to OCM**:
   - When the CPU is already using OCM, HP ports allow the FPGA to directly transfer data without interrupting the CPU.
3. **Multiple FPGA Masters**:
   - When more than one FPGA master needs simultaneous access to OCM or DDR, HP ports handle multiple high-performance data paths.

When to Use AXI ACP Port:

ACP is optimized for small, coherent data transactions rather than bulk memory operations.
AXI ACP ports are preferred when:

1. **Data Coherency is Critical**:
   - Example: When the FPGA needs access to the latest version of data being used by the CPU.
   - The ACP ensures the FPGA sees the same data as the CPU's cache without extra synchronization.
2. **Peripheral Access to OCM**:
   - If other peripherals are accessing OCM via the interconnect, ACP avoids conflicts and ensures the CPU doesn't lose access to critical data.

Use **HP ports for high-speed, large data transfers** and **ACP ports for cache-coherent operations involving small, shared data with the CPU**.

When to Use AXI GP (General Purpose) Ports:

Provides general access to PS resources, including DDR and OCM, but with **lower bandwidth.**

The **AXI GP ports** in the Zynq SoC are designed for **low-bandwidth, general-purpose data transfers** between the Programmable Logic (PL) and the Processing System (PS).

We use AXI GP ports for low-bandwidth tasks like **control, status reporting, debugging, or small data transfers or FPGA configure**.

Three diff ways of PL accessing OCM:

1. **Via High-Performance AXI Interfaces (AXI_HP):**
    - The PL can access the OCM through the AXI_HP ports, which connect to the DDR interconnect and memory controller.
    - Latency: Moderate latency due to the buffering and FIFO stages in the AXI_HP pipeline.
    - Priority: High priority when large data transfers are required, as the AXI_HP interface is designed for high-bandwidth operations.
2. **Via General-Purpose AXI Interfaces (AXI_GP):**
    - The PL accesses the OCM using AXI_GP, which is suitable for control data and low-bandwidth operations.
    - Latency: Higher latency compared to AXI_HP since AXI_GP is optimized for general-purpose tasks.
    - Priority: Lower priority as it is not intended for heavy data processing or time-sensitive tasks.
3. **Direct Access Through the OCM Interconnect:**
    - The PL can directly access the on-chip RAM (OCM) through the OCM interconnect, bypassing the central interconnect.
    - Latency: Lowest latency because it avoids additional routing through other interconnects.
    - Priority: Highest priority as it ensures immediate access to on-chip resources.

Roles of Central Interconnect:

1. **Data routing**
2. **Bandwidth Optimization**
3. **Error detection and handling**
4. **Protocol conversion**
5. **It arbitrates access when multiple masters (e.g., CPU cores, DMA, PL) request access to shared resources.**

What is Cache Flush?

A cache flush is an operation that ensures all the data stored in a CPU's cache is written back to the main memory (e.g., DDR or OCM). It is used to maintain data consistency between the cache and the main memory.

We do it in AXI HP port as ACP maintains cache coherency automatically.

Why Cache Flush is Needed?

Caches are temporary storage areas that store frequently used data to speed up processing. However:

1. If data in the cache changes but is not written back to memory, other parts of the system (like an FPGA accelerator) may work with outdated or incorrect data.
2. Cache flush ensures that all changes in the cache are committed to memory, making the memory up-to-date.