4.3 PARALLEL BUS ARCHITECTURES

Single bus architectures connect multiple processors with their own cache memory using shared bus. This is a simple architecture but it suffers from latency and bandwidth issues. This naturally led to deploying parallel or multiple bus architectures. Multiple bus multiprocessor systems use several parallel buses to interconnect multiple processors with multiple memory modules. The following are the connection schemes in multi bus architectures:

1. **Multiple-bus with full bus–memory connection (MBFBMC)**

   This has all memory modules connected to all buses. The multiple-bus with single bus memory connection has each memory module connected to a specific bus. For N processors with M memory modules and B buses, the number of connections requires are: B(N+M) and the load on each bus will ne N+M.

2. **Multiple bus with partial bus–memory connection (MBPBMC)**

   The multiple-bus with partial bus–memory connection, has each memory module connected to a subset of buses.

3. **Multiple bus with class-based memory connection (MBCBMC)**

   The multiple-bus with class-based memory connection (MBCBMC), has memory modules grouped into classes whereby each class is connected to a specific subset of buses. A class is just an arbitrary collection of memory modules.

4. **Multiple bus with single bus memory connection (MBSBMC)**

   Here, only single bus will be connected to single memory, but the processor can access all the buses. The numbers of connections:

   $$BN + \sum_{j=1}^{k} M_j(j + B - k)$$

   And load on each bus is given by

   $$N + \sum_{j=\max(i+k-B,1)}^{k} M_j,\ 1 \le i \le B$$

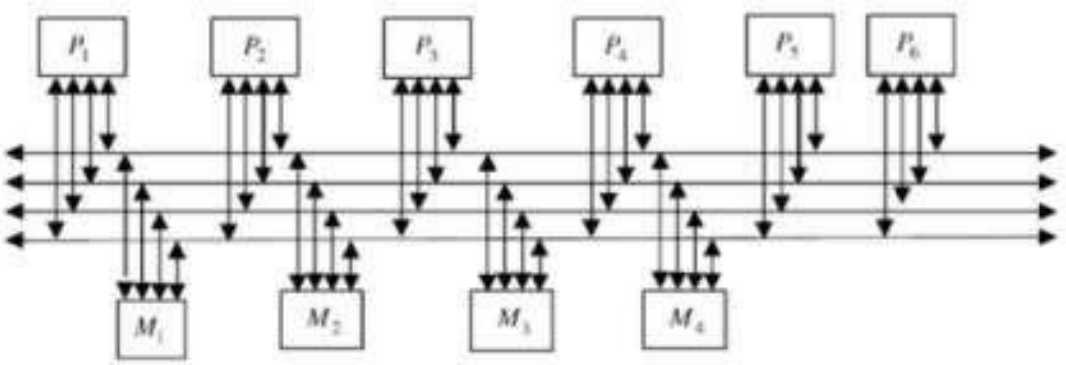

Fig 1: a) Multiple-bus with full bus–memory connection (MBFBMC)

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach.

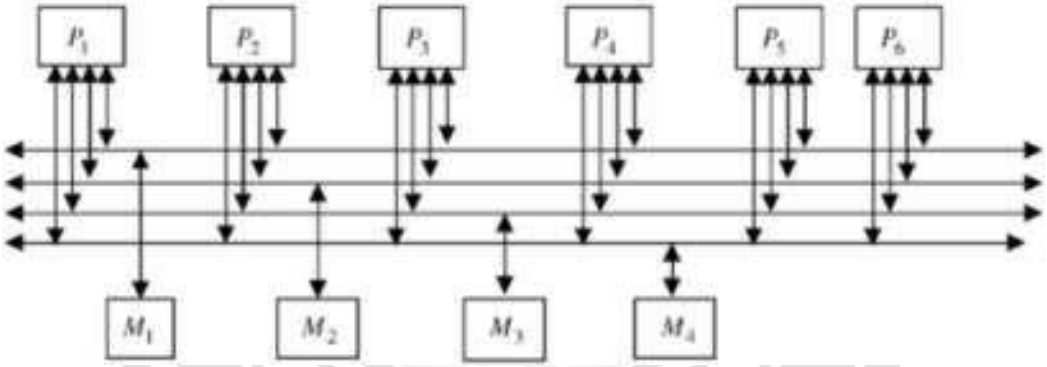

Fig 1b) Multiple bus with single bus memory connection (MBSBMC)

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach.

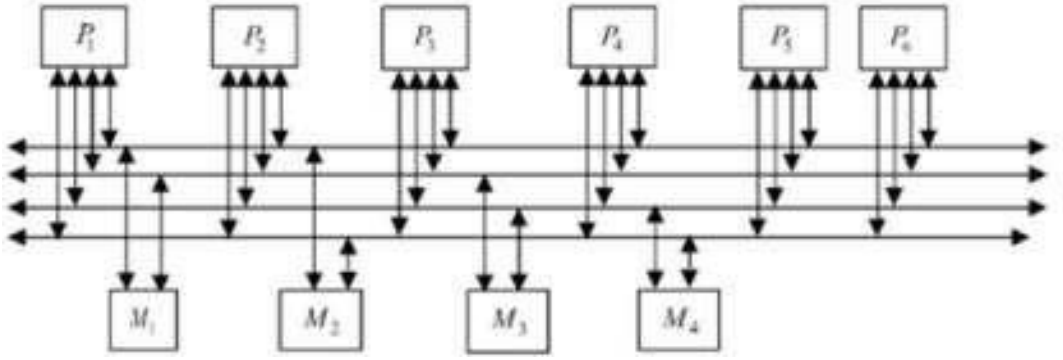

Fig 1 c) Multiple bus with partial bus–memory connection (MBPBMC)

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach.

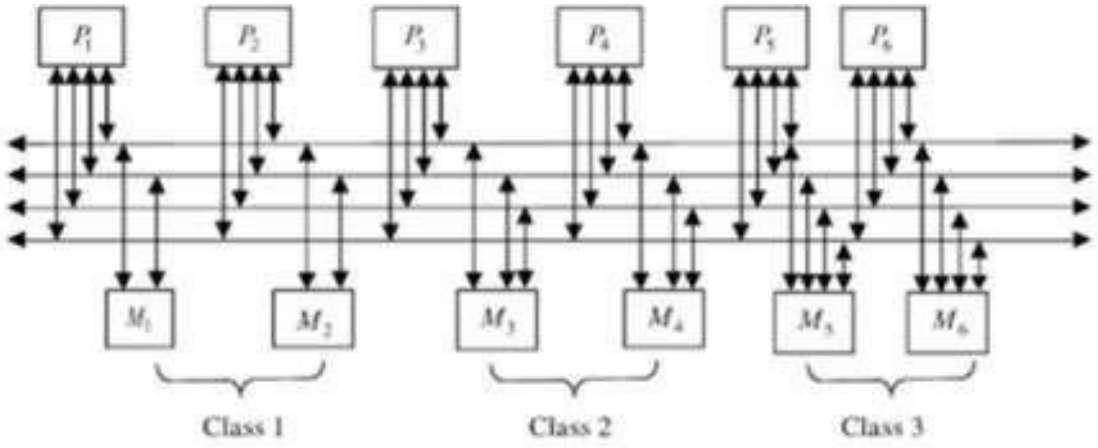

Fig 1d) Multiple bus with class-based memory connection (MBCBMC)

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach.

Bus Synchronization:

- In a single bus multiprocessor system, bus arbitration is required in order to resolve the bus contention that takes place when more than one processor competes to access the bus.
- A bus can be classified as synchronous or asynchronous. The time for any transaction over a synchronous bus is known in advance. Asynchronous bus depends on the availability of data and the readiness of devices to initiate bus transactions.
- The processors that want to use the bus submit their requests to bus arbitration logic. The latter decides, using a certain priority scheme, which processor will be granted access to the bus during a certain time interval (bus master).
- The process of passing bus mastership from one processor to another is called **handshaking,** which requires the use of two control signals: bus request and bus grant.
- Bus request indicates that a given processor is requesting mastership of the bus.
- Bus grant: indicates that bus mastership is granted.
- Bus busy: is usually used to indicate whether or not the bus is currently being used.
- In deciding which processor gains control of the bus, the bus arbitration logic uses a predefined priority scheme.

- Among the priority schemes used are random priority, simple rotating priority, equal priority, and least recently used (LRU) priority.
- After each arbitration cycle, in simple rotating priority, all priority levels are reduced one place, with the lowest priority processor taking the highest priority. In equal priority, when two or more requests are made, there is equal chance of any one request being processed.

In the LRU algorithm, the highest priority is given to the processor that has not used the bus for the longest time.

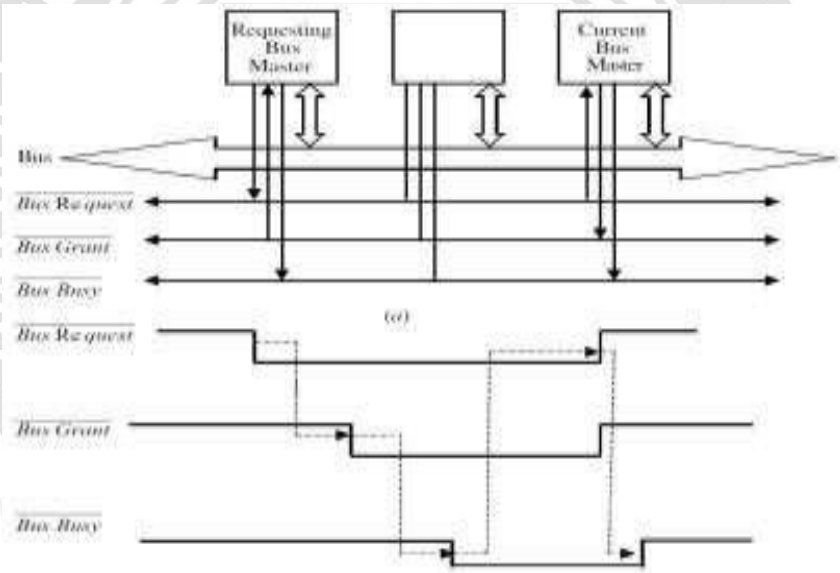

Fig 2: Bus synchronization

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach.

4.3.1 INTERNAL COMMUNICATION METHODOLOGIES

CPU of the computer system communicates with the memory and the I/O devices in order to transfer data between them. The method of communication of the CPU with memory and I/O devices is different. The CPU may communicate with the memory either directly or through the Cache memory. However, the communication between the CPU and I/O devices is usually implemented with the

help of interface. There are three types of internal communications:

- Programmed I/O
- Interrupt driven I/O
- Direct Memory Access (DMA)

Programmed I/O

- Programmed I/O is implicated to data transfers that are initiated by a CPU, under driver software control to access Registers or Memory on a device.

  With programmed I/O, data are exchanged between the processor and the I/O module.
- The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data.
- When the processor issues a command to the I/O module, it must wait until the I/O operation is complete.
- If the processor is faster than the I/O module, this is wasteful of processor time. With interrupt-driven I/O, the processor issues I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work.
- With both programmed and interrupt I/O, the processor is responsible for extracting data from main memory for output and storing data in main memory for input.
- The alternative is known as direct memory access. In this mode, the I/O module and main memory exchange data directly, without processor involvement.
- With programmed I/O, the I/O module will perform the requested action and then set the appropriate bits in the I/O status register.
- The I/O module takes no further action to alert the processor.
- When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate

I/O module. In particular, it does not interrupt the processor.

- It is the responsibility of the processor periodically to check the status of the I/O module. Then if the device is ready for the transfer (read/write).
- The processor transfers the data to or from the I/O device as required. As the CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data.
- The CPU, while waiting, must repeatedly check the status of the I/O module, and this process is known as **Polling.**

The level of the performance of the entire system is severely degraded.

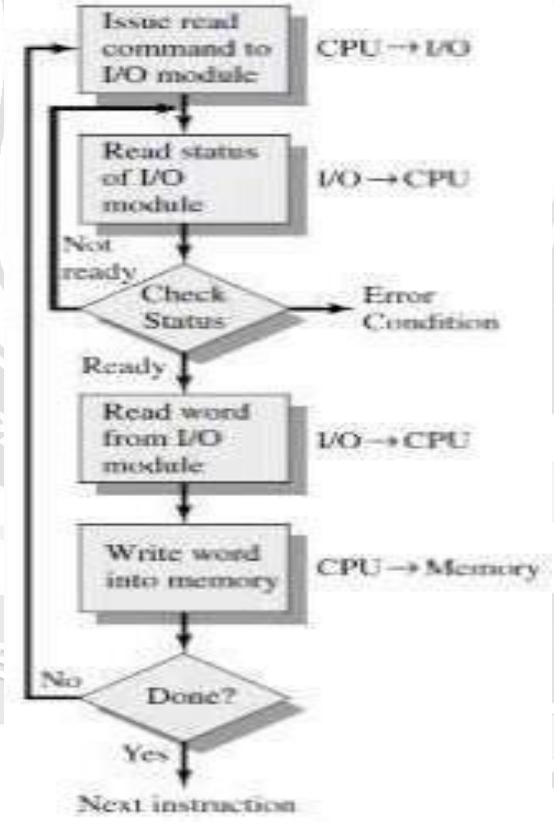

Fig 3: Workflow in programmed I/O

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach.

Interrupt Driven I/O

- The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.
- For input, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.
- For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.
- Although Interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory.

  Below are the basic operations of Interrupt:

  1. CPU issues read command
  2. I/O module gets data from peripheral whilst CPU does other work
  3. I/O module interrupts CPU
  4. CPU requests data
  5. I/O module transfers data

Direct Memory Access (DMA)

- Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement.
- DMA module controls exchange of data between main memory and the I/O device.
- Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus.
- CPU is only involved at the beginning and end of the transfer and interrupted

only after entire block has been transferred.

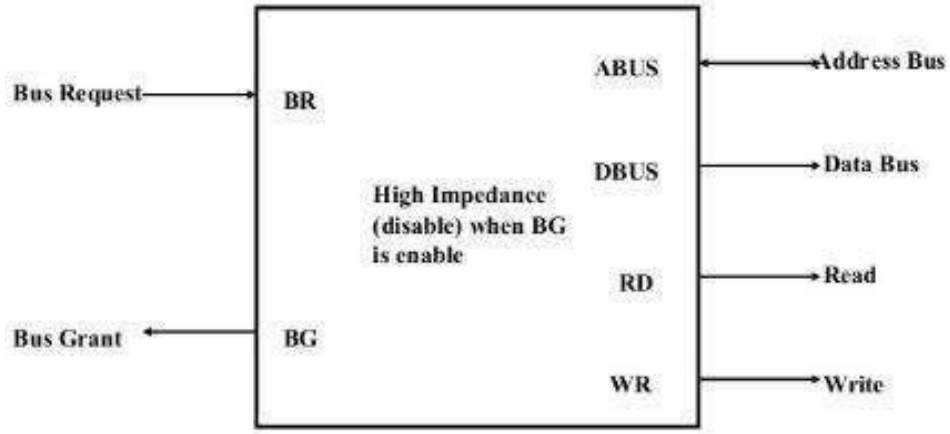

Fig 4: CPU bus signals for DMA transfer

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach.

- The CPU programs the DMA controller by setting its registers so it knows what to transfer where.
- It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum.
- When valid data are in the disk controller┬s buffer, DMA can begin. The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller.
- This read request looks like any other read request, and the disk controller does not know whether it came from the CPU or from a DMA controller.
- The memory address to write to is on the bus address lines, so when the disk controller fetches the next word from its internal buffer, it knows where to write it.
- The write to memory is another standard bus cycle.
- When the write is complete, the disk controller sends an acknowledgement

signal to the DMA controller, also over the bus.

- The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, steps 2 through 4 are repeated until the count reaches 0.
- At that time, the DMA controller interrupts the CPU to let it know that the transfer is now complete.
- When the operating system starts up, it does not have to copy the disk block to memory; it is already there.
- The DMA controller requests the disk controller to transfer data from the disk controller's buffer to the main memory. In the first step, the CPU issues a command to the disk controller telling it to read data from the disk into its internal buffer.

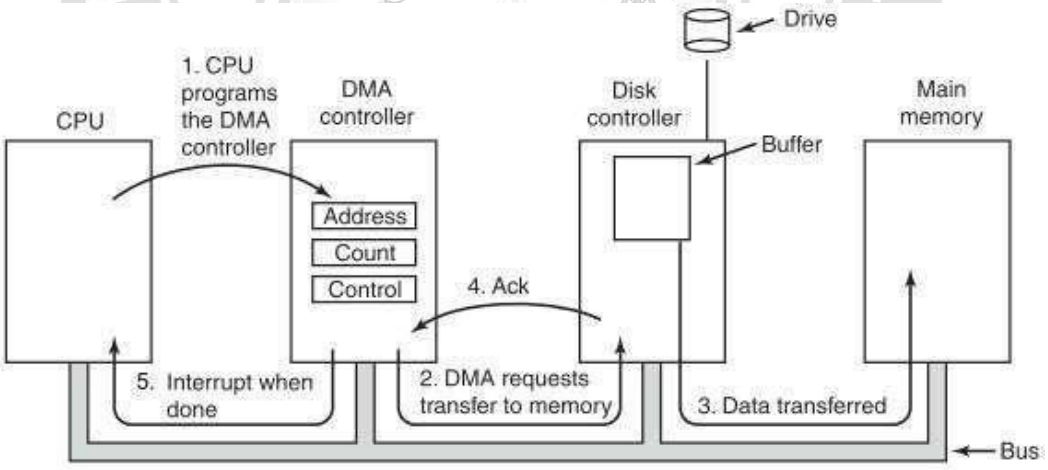

Fig 5: Operations in DMA

Source: Miles J. Murdocca and Vincent P. Heuring, —Computer Architecture and Organization: An Integrated approach.