IO (Input Output)

- Synchronous IO: CPU is waiting for IO to complete.
  - Hw technique: Polling
- Asynchronous IO: CPU is not waiting for IO to complete (doing some other task)
  - Hw technique: Interrupts

Interrupt Processing

- IO event is sensed by IO device controllers.
- It will be conveyed to CPU as a special signal - Interrupt.
- CPU pause current execution
- get address of "ISR" (from IVT) and execute ISR.
- When ISR is completed, execution resumes where it was paused.

Hardware vs Software interrupt

- Hardware -- interrupts from hardware peripherals.
- Software interrupt
  - Special instructions (Assembly/Machine level) when executed, current execution is paused, interrupt handler is executed and then the paused execution resumes.
  - Arch specific:
    - 8085/86: INT
    - ARM 7: SWI
    - ARM Cortex: SVC
  - Also called as "Trap" in few architecture.

Interrupt Controller

- Convey the interrupts from various peripherals to the CPU.
- Also manage priority of the interrupt (when multiple interrupts arrives at same time).
- e.g. 8085/86 <-- 8259, Modern x86 processors (apic), ARM-7 (VIC), ARM-CM3 (NVIC), ...

Interrupt delivery in ARM processors

- It involves an Interrupt Controller that receives interrupt requests from peripherals, resolves priorities, and signals the CPU core.
- The core, after receiving a signal, saves its current state, identifies the interrupt source by reading an interrupt ID from the CPU interface, and executes a corresponding interrupt handler from the vector table.
- After servicing the interrupt, the handler signals completion to the controller, allowing for the processing of the next interrupt.

The Interrupt Delivery Process

1. Interrupt Assertion:

- A peripheral device generates an interrupt request signal, which is then sent to the interrupt controller.

2. Interrupt Controller Actions:

- Configuration: The interrupt controller is configured to manage interrupts.
- Priority Resolution: The controller determines the highest-priority pending interrupt.
- Signaling the CPU: The highest priority interrupt is forwarded to the core's CPU interface.

3. Core Interaction:

- Interrupt Signal: The CPU interface signals the core about the pending interrupt.
- Entering Exception Mode: The core temporarily suspends its current task, saves its context (like registers), and enters an interrupt exception mode.
- Vector Table Lookup: The core jumps to an entry in the interrupt vector table to find the starting address of the appropriate interrupt handler.

4. Interrupt Handler Execution:

- Interrupt Identification: The top-level interrupt handler reads the interrupt ID from the CPU Interface register to identify the specific source of the interrupt.
- Dispatching the Handler: The top-level handler then calls the specific device-specific interrupt service routine (ISR).
- Servicing the Interrupt: The device-specific handler performs the necessary actions to service the interrupt.

5. Interrupt Completion:

- Signaling Completion: Once the ISR finishes, it writes the interrupt ID to the End of Interrupt (EOI) register in the CPU interface.
- Status Update: The interrupt controller marks the interrupt as inactive in the distributor.
- Resuming Operation: The core restores its previous context and resumes its interrupted task or starts processing the next pending interrupt.

Bit-banding

- Bit-banding is a hardware-level feature on select ARM Cortex-M microcontrollers (M3, M4) that allows atomic, single-bit access to memory and peripheral registers.
- It maps a region of memory (the "bit-band region") into a larger "alias region," where each byte in the alias region corresponds to a single bit in the bit-band region.
- This enables direct, single-instruction operations to set or clear individual bits, bypassing inefficient read-modify-write sequences required by traditional methods and improving response times for critical applications.

How it Works

1. Memory Mapping:

- The processor's memory map includes two regions for bit-banding:
  - A bit-band region (e.g., 1MB of SRAM or peripheral memory) where the actual bits reside.
  - A corresponding bit-band alias region (e.g., 32MB of memory) that is mapped to the bit-band region.

2. Atomic Access:

- When a program accesses an address in the bit-band alias region, the bus interface logic intercepts it:
    - Reads: For a read operation, the logic extracts the specified bit from the bit-band region and returns it in the Least Significant Bit (LSB) of the read data.
    - Writes: For a write operation, the logic converts the operation into an atomic read-modify-write sequence for that specific bit, ensuring the operation completes without interruption.

Why It's Useful

- Atomicity:
    - Unlike standard memory access, which works on bytes, bit-banding provides atomic operations for single bits. This is crucial for preventing data corruption or missed events in multi-threaded environments (RTOS) or interrupt-driven systems.
- Efficiency:
    - It eliminates the need for multi-step read-modify-write operations when only a single bit needs to be changed, resulting in faster code execution and smaller compiled code.
- Direct Control:
    - It offers direct and simple access to control individual flags or bits within peripheral registers, such as setting an output pin high or low.

Disadvantages

- Lack of portability
- Manual Address calculations
- Increased code complexity

Examples of Use Cases

- Critical Systems:
    - In applications like anti-lock braking systems (ABS), airbags, or medical devices, fast and predictable bit-level control is essential for safety.
- Real-time Applications:
    - For tasks requiring immediate responses to hardware events, bit-banding helps in controlling and reacting to events more quickly than conventional methods.
- Embedded Systems:
    - It simplifies the control of complex devices with many digital inputs and outputs by allowing precise manipulation of individual flags or status bits.

Address calculations in bit banding

- To calculate a bit-banding address in an Arm processor, use the

```
bit_word_addr = bit_band_base + (byte_offset × 32) + (bit_number × 4)
```

    - The bit_band_base is the starting address of the bit-band alias region,
    - byte_offset is the byte in the bit-band region containing the target bit

    - bit_number is the specific bit (0-7) within that byte.
- This calculation maps a bit-band address in the "alias region" to the corresponding bit in the "bit-band region".

Understanding the Formula

- bit_band_base:
    - This is the base address of the bit-band alias region, which is typically a larger area of memory (e.g., 32MB for SRAM) that maps to the smaller bit-band region.
- byte_offset:
    - This represents the byte position of the target bit within the original bit-band region. For example, if you want to modify a bit in the first byte of a variable, byte_offset would be 0.
- bit_number:
    - This is the specific bit within the target byte that you want to access, ranging from 0 (the least significant bit) to 7 (the most significant bit of the byte).

How it Works

1. Mapping:

- Each bit in the bit-band region has a 32-byte (8-word) area in the bit-band alias region dedicated to it.

2. Address Calculation:
    - byte_offset × 32: This part calculates the starting address of the 32-byte block associated with the specific byte containing your target bit.
    - bit_number × 4: This adds the offset within that 32-byte block to point to the specific word (4-byte address) that maps to your chosen bit.
3. Alias Word:

- The resulting bit_word_addr is the address of the word in the alias region that, when read, will read the bit's status, and when written to, will directly set or clear that specific bit in the original bit-band region.

Example

- If bit_band_base is 0x22000000 and you want to change bit of the byte at 0x200FFFFF, the calculation would be:

```
0x22000000 + (0xFFFFF × 32) + (7 × 4).
```

- The result is 0x23FFFFFC, the alias word address that corresponds to bit of that specific byte.