**ST. ANNE'S COLLEGE OF ENGINEERING AND TECHNOLOGY**
Approved by AICTE, New Delhi. Affiliated to Anna University, Chennai
Accredited by NAAC
ANGUCHETTYPALAYAM, PANRUTI – 607 106.

| | | |
|---|---|---|
| **Subject Name** | : | **Embedded Processors** |
| **Subject Code** | : | **EE3018** |
| **Regulation** | : | **2021** |
| **Department** | : | **Electrical and Electronics Engineering** |
| **Year / Semester** | : | **II/IV** |

# UNIT-III - PERIPHERALS OF ARM

**ARM : I/O Memory – EEPROM – I/O Ports – SRAM – Timer – UART – Serial Communication with PC**

## Memory Mapped I/O

- ✓ I/O Port addressing is not the only way the processor can communicate with external devices
- ✓ Another commonly used technique is called memory mapped I/O
- ✓ Asserting the I/O pin and addressing a data port, the processor just accesses a memory address.
- ✓ Directly. The external device can have a small amount of RAM or ROM that the processor.
- ✓ Reads or writes as needed Common Peripherals in ARM-based Systems
- ✓ Memory-mapped I/O (MMIO) is a method used by computers to control hardware devices.
- ✓ device registers are mapped into the same address space as program memory and/or user memory.
- ✓ This allows the CPU to control devices by reading and writing specific memory locations.

## MMIO Works in ARM Architecture

**Address Mapping**:

- ✓ Each Peripheral Device Is Assigned A Unique Range Of Memory Addresses.
- ✓ Addresses Are Mapped To The Device Registers, Which Control The Hardware.

**Accessing Devices**:

- ✓ To interact with a device, the CPU reads from or writes to specific addresses within the device's memory range.
- ✓ Special instructions or regular load/store instructions can be used to perform these operations.

**Advantages**:

- ✓ Simplifies the CPU design as the same instructions used for regular memory operations are used for I/O.
- ✓ Allows for efficient access to device registers, as memory operations are typically fast.

**Disadvantages**:

- ✓ Requires careful management
- ✓ The programmer needs to know the specific memory addresses

**Memory mapped I/O and Isolated I/O**

As a CPU needs to communicate with the various memory and input-output devices (I/O)

. There are three ways in which system bus can be allotted to them :

1. Separate set of address, control and data bus to I/O and memory.
2. Have common bus (data and address) for I/O and memory but separate control lines.
3. Have common bus (data, address, and control) for I/O and memory.

**Isolated I/O**

- ✓ Then we have Isolated I/O in which we Have common bus(data and address) for I/O and memory but separate read and write control lines for I/O.
- ✓ when CPU decode instruction then if data is for I/O then it places the address on
- ✓ the address line and set I/O read or write control line on due to which data transfer occurs between CPU and I/O.

As the address space of memory and I/O is isolated and the name is so.

The address for I/O here is called ports. Here we have different read instruction

**Memory Protection and MMIO**

- ✓ Controls MMIO regions, ensuring
- ✓ Authorized code can interact with hardware devices.

**Memory-Mapped I/O in ARM**

- In ARM-based systems, peripherals are accessed via memory-mapped I/O, where each peripheral is assigned a specific range of memory addresses.
- Accessing these addresses allows the CPU to read from or write to peripheral registers,
- configuring their behaviour and exchanging data.
- For example, configuring a GPIO pin as an output and setting its state (high or low) involves writing to specific bits in the GPIO registers.

**Programming Considerations**

When programming with ARM peripherals:

- **Read and Understand Datasheets**: Each microcontroller family has specific registers and configurations for its peripherals
- . Consult the datasheet and reference manual for detailed information.
- **Use Peripheral Libraries**: ARM often provides CMSIS (Cortex Microcontroller Software Interface Standard)
- libraries that include standardized functions and macros for accessing and configuring peripherals.
- **Interrupt Handling**: Configure interrupts to handle asynchronous events from peripherals efficiently, ensuring timely response to external stimuli.

```
4GB +------------------+   <- 32-bit
    | DRAM             |
    |                  |
2GB +------------------+
    | Mapped I/O       |
1GB +------------------+
    | ROM & RAM & I/O  |
0GB +------------------+   0
```

- **Power Management**: Utilize low-power modes and peripheral clock gating to optimize power consumption in battery-operated devices.
- **Debugging Tools**: Use debuggers and simulation tools to verify peripheral configurations and troubleshoot issues during development.

## I/O PORTS OF ARM

- ✓ The series of PIC16 consists of five ports such as Port A, Port B, Port C, Port D & Port E.
- ✓ Port A is an 16-bit port that can be used as input or output port based on the status of the TRISA (Tradoc Intelligence Support Activity) register.
- ✓ Port B is an 8- bit port that can be used as both input and output port.
- ✓ Port C is an 8-bit and the input of output operation is decided by the status of the TRISC register.
- ✓ Port D is an 8-bit port acts as a slave port for connection to the microprocessor BUS.
- ✓ Port E is a 3-bit port which serves the additional function of the control signals to the analog to digital converter
- ✓ I/O ports refer to the general-purpose input/output pins that can be
- ✓ configured to interface with external devices, sensors, actuators, and other components.
- ✓ These ports are typically part of the GPIO (General-Purpose Input/Output)
- ✓ peripheral and are crucial for connecting and controlling various external circuits in embedded systems.

### Structure and Configuration of I/O Ports

**GPIO Registers**:

- ✓ Each GPIO port is associated with a set of control registers
- ✓ that configure its behavior, including direction
- ✓ (input or output), mode (push-pull or open-drain),
- ✓ speed, and pull-up/pull-down configuration.

**Purpose**:

- ✓ GPIO pins are versatile and can be configured as inputs or outputs to

- ✓ connect external devices such as LEDs, buttons, sensors, and actuators.

- **Registers**: Typically controlled via registers like
  - GPIOx_MODER (Mode Register),
  - GPIOx_OTYPER (Output Type Register)
  - GPIOx_OSPEEDR (Output Speed Register),
  - GPIOx_PUPDR (Pull-up/Pull-down Register), and
  - GPIOx_ODR (Output Data Register).

- ✓ **Example Usage**: Controlling LED blinking, reading button states, interfacing with simple digital sensors

**Pin Configuration**:

- ✓ GPIO pins are typically numbered sequentially within a port (e.g., GPIOA, GPIOB, etc.).
- ✓ Each pin can be individually configured and controlled through its respective register bits.

**Modes of Operation**:

- ✓ **Input Mode**: Pins configured as inputs can read external signals.
- ✓ **Output Mode**: Pins configured as outputs can drive external loads
- ✓ Output pins can be configured for various drive strengths and can operate in push-pull or open-drain configurations.

**Example of GPIO Usage in ARM**

**Programming Considerations**

**Portability**:

- ✓ Use CMSIS (Cortex Microcontroller Software Interface Standard)
- ✓ headers and peripheral libraries provided by the microcontroller
- ✓ manufacturer to ensure code portability across different ARM Cortex-based microcontrollers.

**Interrupt Handling**:

- ✓ Configure GPIO interrupts to respond to external events (e.g., button press
- ✓ efficiently using NVIC (Nested Vectored Interrupt Controller).

- **Power and Speed Control**:

  - ✓ Optimize GPIO configurations for power consumption
  - ✓ signal integrity by adjusting output drive strength and speed settings.

**Debugging Tools**:

- ✓ Use debugging tools (e.g., debugger probes, logic analyzers) to verify GPIO pin states
- ✓ troubleshoot connectivity issues during development.

**EEPROM IN ARM**

- ✓ EEPROM (Electrically Erasable Programmable Read-Only Memory).
- ✓ EEPROM is a non-volatile memory technology used in ARM microcontrollers
- ✓ processors to store small amounts of data that need to be retained even when the power is turned off.
- ✓ employed for storing configuration settings, calibration data, and other critical information in embedded systems:

**Integration of EEPROM in ARM Systems**

1. **Hardware Interface**:

- ✓ **I2C Interface**: EEPROMs often use I2C (Inter-Integrated Circuit) interface for communication with the ARM microcontroller.
- ✓ straightforward way to connect multiple devices using only two wires

2. **Memory Organization**:

- ✓ EEPROMs are organized in pages or blocks, typically ranging from a few kilobits to several megabits in size.
- ✓ Each page/block can be individually read from, written to, or erased.

3. **Memory Access Methods**:

**Sequential Access**:

- ✓ Data in EEPROM is accessed sequentially by specifying the memory address to read from or write to.

**Random Access**:

- ✓ Some EEPROMs support random access,
- ✓ allowing direct access to any memory location using specific commands and addressing schemes.

**Key Characteristics of EEPROM**

1. **Non-Volatile Memory**:

- ✓ Data stored in EEPROM is preserved even when the device is powered off.

2. **Byte-Level Erase and Write**:

- ✓ EEPROM can be erased and written at the byte level,
- ✓ allowing for more flexible data storage compared to other types of memory like Flash.

3. **Limited Write Cycles**:

- ✓ EEPROM has a limited number of write cycles (typically in the range of 100,000 to 1,000,000),
- ✓ important to minimize write operations to extend its lifespan.

**How EEPROM is Accessed in ARM-Based Systems**

- ✓ In ARM-based systems, EEPROM can be either internal (integrated within the microcontroller) or external (connected via I2C, SPI, or other communication interfaces).

**1. Internal EEPROM**

- Some ARM microcontrollers, especially those used in embedded and automotive applications, have built-in EEPROM.
- Access to internal EEPROM is often through dedicated registers and may involve special sequences to initiate read/write operations.

**2. External EEPROM**

- ✓ External EEPROMs are often connected via communication interfaces like I2C or SPI.
- ✓ Accessing commands to the EEPROM chip via the appropriate protocol (I2C/SPI) to read or write data.

## APPLICATIONS

1. **Storing Configuration Data**:

- ✓ ARM microcontrollers use EEPROM to store configuration settings such as device parameters,
- ✓ calibration data for sensors, network settings, and user preferences. This data is retained across power cycles.

2. **Data Logging and Event Storage**:

- ✓ In logging applications, EEPROMs can store data logs, event records, and error codes,
- ✓ providing a reliable storage medium that doesn't require constant power supply like RAM.

3. **Security and Key Storage**:

- ✓ Critical security information such as encryption keys, certificates, and security tokens

## Considerations for Using EEPROM

**Write Endurance**:

- ✓ EEPROMs have limited write endurance(number of write cycles per memory cell) compared to other non-volatile memories like Flash.

**Page Write Operations**:

- ✓ To optimize write operations, many EEPROMs support page write operations
- ✓ where multiple bytes can be written at once, reducing overhead and improving efficiency.

**Data Retention**:

- ✓ Ensure that data written to EEPROM is periodically refreshed

- ✓ rewritten to maintain data integrity, especially in applications with stringent reliability requirements.

**Operating Conditions**:

Follow manufacturer guidelines for voltage levels, timing specifications, and temperature

- Ranges to ensure reliable operation of EEPROM in various environmental .conditions.

**SRAM (Static Random-Access Memory)**

- ✓ Static random access memory (SRAM) is faster than the more traditional.
- ✓ DRAM, requires more silicon area.
- ✓ SRAM is *static*—the RAM does not require refreshing.
- ✓ The access time for SRAM is considerably shorter than the equivalent DRAM. because SRAM does not require a pause between data accesses Because of its higher cost, it is used mostly.
- ✓ Smaller high-speed tasks, such as fast memory and caches.

**Purpose**: SRAM is a type of volatile memory used for storing data that the CPU needs to access quickly and frequently during program execution.

- **Characteristics**:
  - **Speed**: SRAM offers fast access times compared to other types of memory like DRAM (Dynamic RAM).
  - **Volatility**: SRAM requires continuous power to retain stored data.
- **Usage**:
  - **Execution Memory**: SRAM is often used as a cache memory or directly as working memory for the CPU.
  - **Data Storage**: It can also be used for storing temporary variables, stack space, and other runtime data..

**SRAM in ARM Architecture**

In ARM microcontrollers, SRAM is used for storing:

- ✓ **Program Variables**: Local variables, global variables, and static variables during program execution.

- ✓ **Stacks**: Stack memory used for function calls, local variables, and return addresses.
- ✓ **Buffers**: Data buffers for I/O operations, such as UART, SPI, or I2C communications.

```
#include <stdint.h>

uint8_t buffer[256];  // This buffer is stored in SRAM

void main() {
   uint8_t counter = 0;  // This variable is also stored in SRAM

   for (counter = 0; counter < 256; counter++) {
      buffer[counter] = counter;  // Writing to the SRAM buffer
   }

   while (1) {
      // Main loop
   }
}
```

## Accessing SRAM in ARM Assembly

In ARM assembly, you can directly manipulate SRAM by working with registers and memory addresses. Below is a simple example.

### Example in ARM Assembly

```
   AREA MyCode, CODE, READONLY
   ENTRY


   LDR R0, =0x20000000  ; Load base address of SRAM into R0
   MOV R1, #0xFF        ; Load value 0xFF into R1
   STR R1, [R0]         ; Store value from R1 into SRAM at address
0x20000000
   LDR R2, [R0]         ; Load value from SRAM into R2
   B   .                ; Infinite loop
```

## Using SRAM Efficiently

**Variable Scope**:

- ✓ Use local variables where possible to minimize SRAM usage since global

- ✓ static variables are retained throughout the program's execution.

**Stack Management**:

- ✓ Be cautious with stack usage, especially in systems with limited SRAM, to avoid stack overflow.

**Buffers**:

- ✓ Allocate buffers dynamically if supported, to manage SRAM efficiently, especially in resource-constrained systems.

**Timer/Counter Module**

- ✓ ATtiny328P has two internal timers.
- ✓ We can use these timers to make counters and to generate pulses.
- ✓ Both of these timers are dependent on an oscillator.
- ✓ Both timers can use the internal and external clock to operate,
- ✓ they also have an internal pin which can be used to count according to the external pulses.
- ✓ All of these pins in microcontroller ATmega328P are given below:

◎ T0 – GPIO6

◎ T1 – GPIO11

◎ TOSC1 – GPIO9

◎ TOSC2 – GPIO10

◎ ICP1 – GPIO

ICP1 is an input capture pin which can be used to capture the external pulse at a specific interval of time.

When an input pulse will occur on this pin then it will generate a timestamp which can tell when the external signal was received.

**Purpose**: Timers are essential peripherals that provide accurate timing and counting capabilities,

enabling precise control over time-based events and synchronization.

- **Types**:

**General-Purpose Timers**:

- ✓ Used for generating periodic interrupts, measuring time intervals, or PWM (Pulse Width Modulation) generation.

**Watchdog Timers**:

- ✓ Monitors system health by resetting the microcontroller if certain conditions (e.g., no timer reset within a specified period) are met.

**Features**:

**Clock Sources**:

- ✓ Timers typically operate from internal or external clock source
- ✓ providing flexibility in timing accuracy and power consumption.

**Capture/Compare Units**:

- ✓ Enable advanced features like input capture
- ✓ for measuring external events and output compare (for generating PWM signals).

**UART (Universal Asynchronous Receiver/Transmitter)**

- ✓ Provides serial communication capability for transmitting and receiving data
- ✓ asynchronously between the microcontroller and external devices like GPS modules, Bluetooth modules, and serial consoles.

**Registers**:

- ✓ Configuration registers for baud rate, data format (e.g., number of bits per frame, parity, stop bits).
- ✓ status registers for transmission and reception.

**Example Usage**:

- ✓ Serial communication for debugging, data logging, wireless communication.

**Purpose**: UART provides serial communication capability for transmitting and receiving data asynchronously between the microcontroller and external devices.

- ✓ **Characteristics**:
  - o **Simplex Communication**: Supports one-way (transmit or receive) or duplex (transmit and receive) serial data transmission.

**Configurable Parameters**:

- ✓ Baud rate, data bits, parity, and stop bits can be configured to match communication requirements with external peripherals.

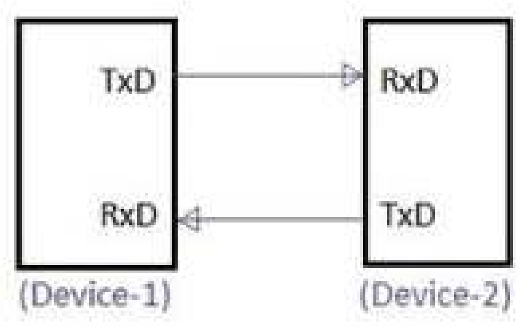


Fig.1.1 UART inerface

**Features of UART interface**

- ✓ **UART** supports lower data rate.
- ✓ Receiver need to know baud rate of the transmitter before initiation reception i.e. before communication to be established.
- ✓ UART is simple protocol, it uses start bit (before data word), stop bits (one or two, after data word),
- ✓ parity bit (even or odd) in its base format for data formatting. Parity bit helps in one bit error detection.
- ✓ UART Packet = 1 start bit(low level), 8 data bits including parity bit, 1 or 2 stop bit(high level).
- ✓ Data is transmitted byte by byte.
- ✓ UART generates clock internally and synchronizes it with data stream with the help of transition of start bit.
- ✓ It is also referred as RS232 .

- ✓ For long distance communication, 5V UART is converted to higher voltages viz. +12V for logic 0 and -12V for logic 1.

**Usage**:

- ✓ Serial Communication Used for debugging, data logging, interfacing with sensors
- ✓ communication with other microcontrollers or peripherals (e.g., GPS modules, Bluetooth modules).
    - **Boot loader Interface**: Often used during firmware development and updates,
    - allowing communication with a host computer for programming and debugging.

**SERIAL COMMINICATION WITH PC OF ARM**

- Serial communication between an ARM-based microcontroller and a PC is commonly achieved using UART (Universal Asynchronous Receiver/Transmitter).
- This method allows for bidirectional data exchange over a simple serial interface

**SPI Protocol:** The term SPI stands for Serial Peripheral Interface.

- This protocol is used to send data between PIC microcontroller and other peripherals such as SD cards, sensors and shift registers.
- PIC microcontroller support three wire SPI communications between two devices on a common clock source.
- The data rate of SPI protocol is more than that of the USART.

**SERIAL CABLE**

- A serial cable is a type of cable used to transfer information between two devices using serial communication.
- In serial communication, data is sent one bit at a time, sequentially, over a communication channel or computer bus.
- Serial cables are commonly used for connecting various devices such as computers, modems, and other peripherals.

**Key Characteristics of Serial Cables:**

1. **Connectors**: Serial cables often use connectors like DB9 (9-pin) or DB25 (25-pin).
2. **Standards**: There are several serial communication standards, with RS-232 being one of the most common.
3. Other standards include RS-485 and RS-422, which offer different performance characteristics and are used in various applications.
4. **Transmission Speed**: depending on the standard and the quality of the cable. RS-232, for example, typically supports speeds up to 115.2 kbps.
5. **Distance**: The effective communication distance of serial cables can also vary.
6. RS-232 cables are generally used for shorter distances (up to 50 feet), while standards like RS-485 can support longer distances (up to 4,000 feet).

**Common Uses:**

- **Connecting Computers and Peripherals**: Serial cables can connect computers to devices like printers, modems, and mice.
- **Networking**: They can be used in certain types of networking setups, particularly in industrial environments.
- **Data Transfer**: Serial cables facilitate data transfer between devices, such as between a computer and an embedded system for programming and debugging purposes.

**Advantages and Disadvantages:**

**Advantages**:

- Simple and cost-effective for short-distance communication.
- Widely used and supported by many devices.

**Disadvantages**:

- Limited data transfer speed compared to modern alternatives like USB and Ethernet.
- Limited distance for certain standards (e.g., RS-232)

**I2C Protocol:**

**6. I2C (Inter-Integrated Circuit)**

**Purpose**:

- ✓ Provides multi-master, multi-slave,
- ✓ serial communication interface for connecting low-speed peripherals such as EEPROMs, sensors (e.g., temperature sensors, accelerometers), and real-time clocks.
- ✓ **Registers**: Configuration registers for setting clock speed, addressing mode, data registers for transmitting and receiving data.
- ✓ **Example Usage**: Sensor networks, real-time clock synchronization, EEPROM data storage.

- ✓ The term I2C stands for Inter Integrated Circuit , and it is a serial protocol .
- ✓ which is used to connect low speed devices such as EEPROMS, microcontrollers, A/D converters, etc.
- ✓ PIC microcontroller support two wire Interface or I2C communication between two devices which can work as both Master and Slave device.

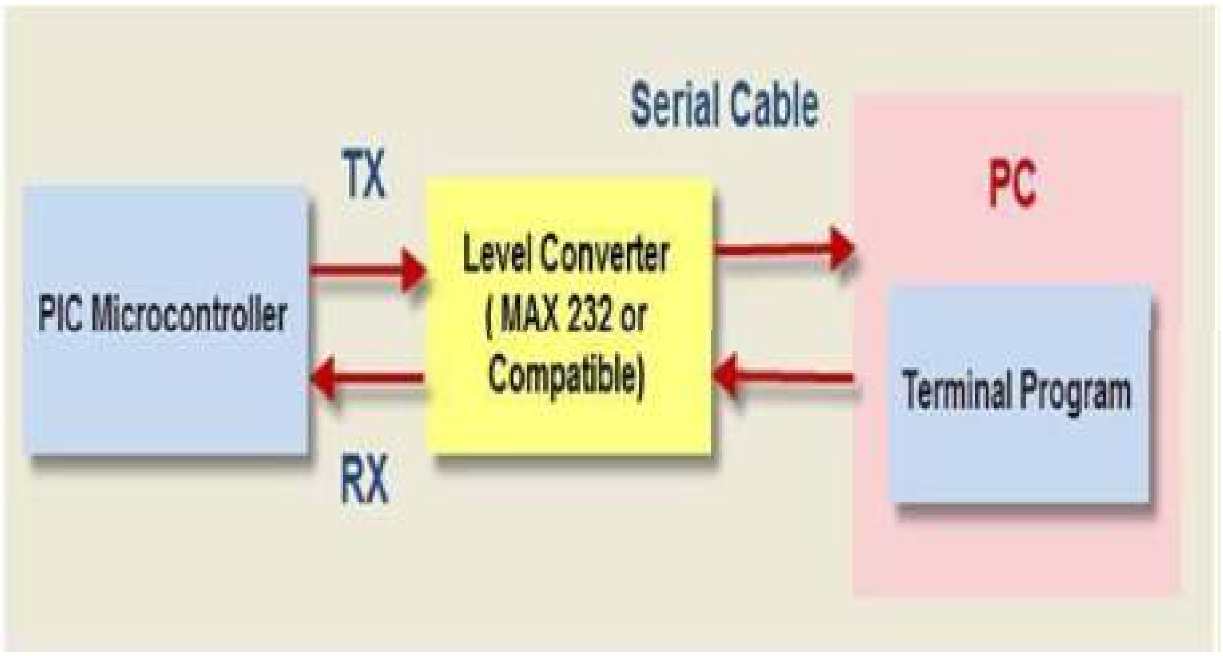


**Fig. 1.2 12c interface**

**PWM (Pulse Width Modulation)**

- **Purpose**: Generates digital pulses of variable width to control the power delivered to devices like mo
- tors, LEDs, and servo motors.
- **Registers**: Configuration registers for duty cycle, period, and output enable.
- **Example Usage**: Motor speed control, LED brightness control, servo motor position control.

**USART:**

- The name USART stands for Universal synchronous and Asynchronous Receiver and Transmitter which is a serial communication for two protocols.

- It is used for transmitting and receiving the data bit by bit over a single wire with respect to clock pulses

- The PIC microcontroller has two pins TXD and RXD. These pins are used for transmitting and receiving the data serially.

**Hardware Setup**:

  - ARM-based microcontroller development board with UART interface.
  - USB-to-serial adapter or a development board with USB-to-UART bridge (optional if the microcontroller has USB support).

2. **Software Tools**:

- ✓ Integrated Development Environment (IDE) for ARM development (e.g., Keil µVision, STM32CubeIDE, or any IDE supporting ARM Cortex-M development).
- ✓ Terminal software on the PC  for serial communication monitoring and testing.

**Steps to Implement Serial Communication**

**1. Configure UART on ARM Microcontroller**

- **Initialize UART Peripheral**: Use manufacturer-provided libraries (e.g., CMSIS, HAL libraries for STM32) to configure UART communication.

- **Set UART Parameters**: Configure baud rate, data bits, parity, and stop bits according to the UART settings of your PC terminal software.
- **Enable UART Interrupts** (Optional): Configure UART interrupts for handling incoming data or transmission events asynchronously.

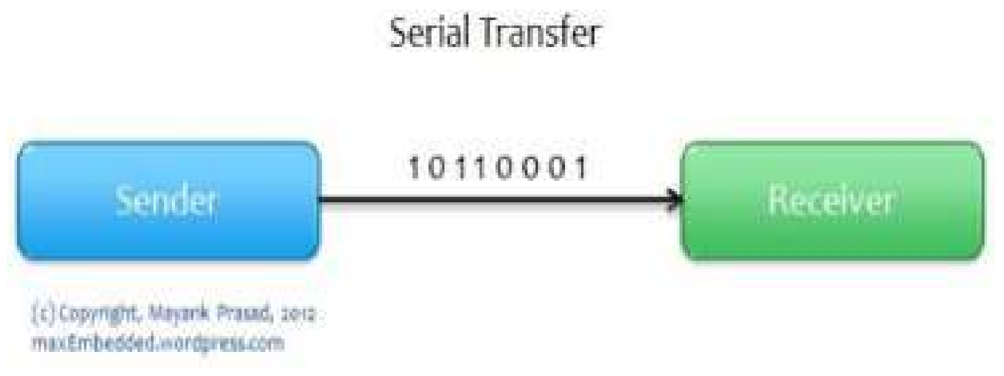


**Fig 1.3.serial communication**

**Transmit Data from ARM Microcontroller**

- Use UART transmit functions to send data from the microcontroller to the PC.

**Receive Data on ARM Microcontroller**

- Implement UART receive functions to capture data sent from the PC to the microcontroller.

**PC Terminal Setup**

- **Connect PC and Microcontroller**: Use a USB-to-serial adapter to connect the UART pins (TX, RX, GND) of the microcontroller to the USB port of the PC.
- **Terminal Software Configuration**: Open the terminal software on the PC and configure the COM port (typically COM1, COM2, etc.) and settings (baud rate, data bits, parity, stop bits) to match the settings configured in the microcontroller code.
- **Testing Communication**: Send and receive data between the PC and microcontroller using the terminal software.
- Verify that transmitted data from the microcontroller is correctly received by the PC, and vice versa.

**Considerations**

- **Buffering**: Implement FIFO (First In, First Out) buffers for UART data to handle incoming and outgoing data efficiently, especially in interrupt-driven applications.
- **Error Handling**: Implement error checks (e.g., parity error, framing error) in UART interrupt handlers to ensure robust communication.
- **Power and Grounding**: Ensure proper grounding between the microcontroller and PC to prevent electrical noise and signal integrity issues.

**ADC/DAC INTERFACING OF ARM**

- ✓ Techniques to convert analog signals to digital data, generate analog signals from digital data, and control stepper motor movements.:

**ADC (Analog-to-Digital Converter) Interfacing**

**2. ADC (Analog-to-Digital Converter)**

- ✓ Converts analog signals from sensors (e.g., temperature sensors, light sensors, analog joysticks) into digital values for processing by the microcontroller.
- ✓ **Registers**: Configuration registers include ADC control, channel selection, sampling rate, and result registers for storing converted digital values.
- ✓ **Example Usage**: Reading analog sensor values for temperature monitoring, light intensity measurement.

**Purpose**: ADCs convert analog signals (e.g., from sensors, potentiometers) into digital values that can be processed by the ARM microcontroller.

- **Hardware Setup**:

- ✓ Connect the analog signal source (e.g., sensor output) to the input pins of the ADC module on the ARM microcontroller.
- ✓ Ensure proper power and ground connections for accurate analog signal measurement.

- **Configuration**:
    - Initialize the ADC module using manufacturer-provided libraries or CMSIS drivers.
    - Configure ADC parameters such as resolution (bits), sampling rate, and input channel selection.
- **Reading ADC Values**:
    - Trigger ADC conversions and read digital values from ADC data registers.
    - Implement scaling or calibration factors to convert digital ADC values back to meaningful physical quantities (e.g., temperature, voltage).

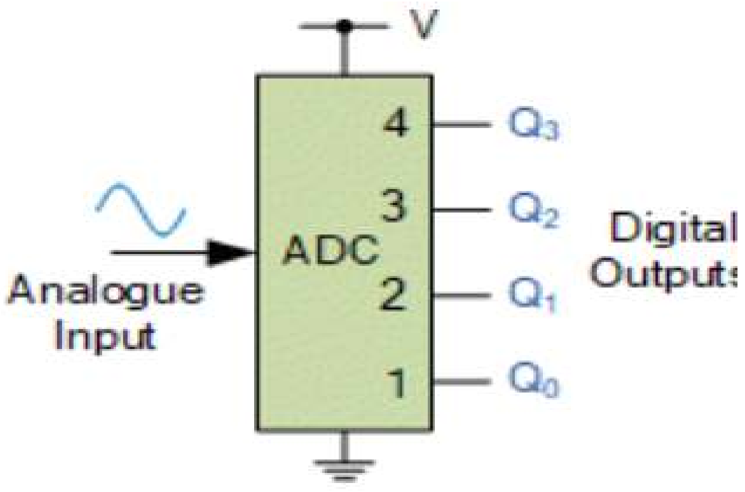


**Fig 1.4 ADC interfacing**

- The resolution of the ADC is the number of bits it uses to digitize the input samples.
- For an n bit ADC the number of discrete digital levels that can be produced is 2n.
- Thus, a 12 bit digitizer can resolve 212 or 4096 levels. The least significant bit (lsb) represents the smallest interval that can be detected and in the case of a 12 bit digitizer is 1/4096 or 2.4 x 10-4.
- To convert the lsb into a voltage we take the input range of the digitizer and divide by two raised to the resolution of the digitizer.

- Table 1 shows the lsb for a one Volt (±500 mV) input range for digitizers with resolutions of 8 to 16 bits.

| Resolution | Ideal Dynamic range | Minimum Voltage Increment |
|---|---|---|
| 8 Bit | 256:1 | 3.92 mV |
| 10 Bit | 1024:1 | 0.98 mV |
| 12 Bit | 4096:1 | 0.244 mV |
| 14 Bit | 16384:1 | 61 μV |
| 16 Bit | 65536:1 | 15 μV |

- The main intention of this analog to digital converter is to convert analog voltage values to digital voltage values.
- A/D module of PIC microcontroller consists of 5 inputs for 28 pin devices and 8 inputs for 40 pin devices.
- The operation of the analog to digital converter is controlled by ADCON0 and ADCON1 special registers.
- The upper bits of the converter are stored in register ADRESH and lower bits of the converter are stored in register ADRESL. For this operation, it requires 5V of an analog reference voltage.

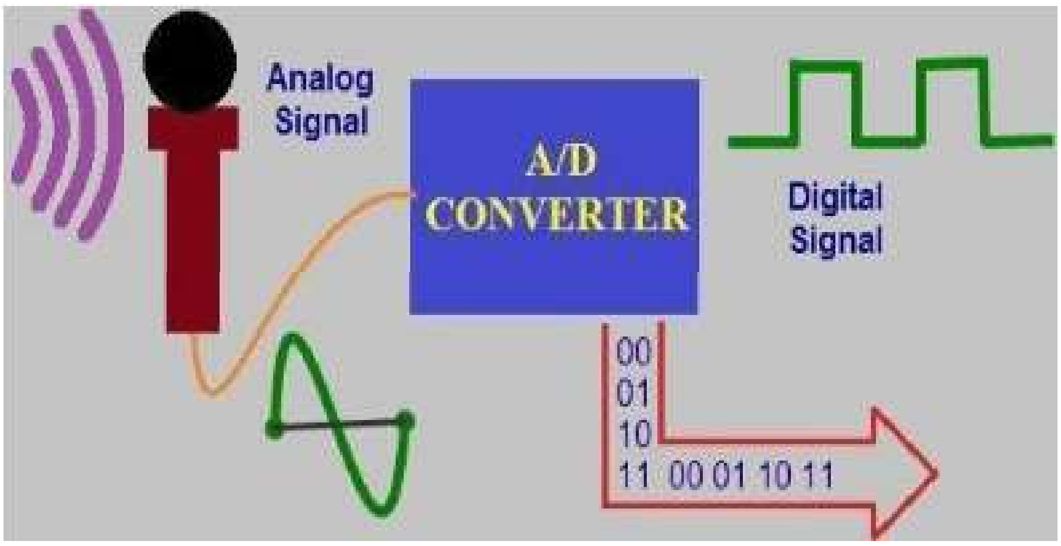


**FIG1.5A/D CONVERTER**

## DAC (Digital-to-Analog Converter) Interfacing

- ✓ Converts digital data into analog signals used for applications such as audio output, voltage control, and waveform generation.
- ✓ **Registers**: Configuration registers control output voltage range, buffer enable, and data registers for setting analog output levels.
- ✓ **Example Usage**: Generating audio signals, controlling analog devices such as motor speed controllers.
- ✓ **Purpose**: DACs convert digital values from the ARM microcontroller into corresponding analog voltages or currents for driving analog devices or actuators.
- ✓ A **Digital to Analog Converter (DAC)** converts a digital input signal into an analog output signal.
- ✓ The digital signal is represented with a binary code,
- ✓ which is a combination of bits 0 and 1.
- ✓ A Digital to Analog Converter (DAC) consists of a number of binary inputs and a single output.
- ✓ In general, the **number of binary inputs** of a DAC will be a power of two.

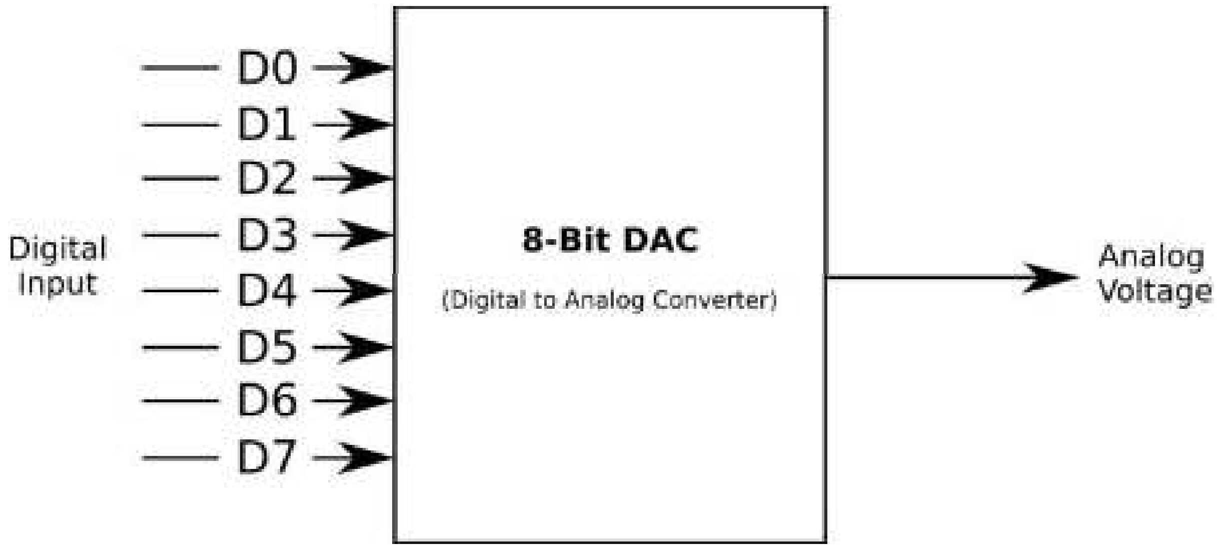


**Fig 1.6 DAC interfacing**

- **Hardware Setup**:
  - Connect DAC output pins to external circuitry (e.g., analog voltage amplifier, motor driver) to control analog outputs.

  - Ensure proper grounding and power supply for stable DAC operation.
- **Configuration**:
  - Initialize the DAC module using CMSIS or manufacturer-provided libraries.
  - Set DAC parameters such as resolution, output voltage range, and output buffer status.
- **Output Generation**:
  - Write digital values (typically 12-bit for most DACs) to the DAC data register.
  - The DAC converts digital values to corresponding analog voltages or currents at the output pins.

**Stepper Motor Interfacing**

- ✓ Stepper motors are precise actuators that move in discrete steps.
- ✓ making them suitable for applications requiring accurate positioning or motion control.
- ✓ A **stepper motor** is made up of a rotor, which is normally a permanent magnet and it is, as the name suggests the rotating component of the motor.
- ✓ A stator is another part which is in the form of winding.
- ✓ In the diagram below, the center is the rotor which is surrounded by the stator winding. This is called as four phase winding.

**Stepper Motor Interface with ARM Processor**

- ✓ A stepper motor is a type of brushless DC motor that moves in discrete steps, making it ideal for precise control of position and speed.
- ✓ When interfacing a stepper motor with an ARM processor, the processor controls the motor by sending pulses to a motor driver circuit.
- ✓ The motor driver then translates these pulses into specific movements of the motor shaft.

**Basic Components:**

1. **ARM Processor:**
   - Acts as the brain of the system.
   - Sends control signals to the motor driver based on the desired movement.

2. **Stepper Motor:**
   - Converts electrical pulses into precise mechanical movements.
   - Moves in fixed step angles, allowing for precise positioning.
3. **Motor Driver:**
   - Interfaces between the ARM processor and the stepper motor.
   - Amplifies the control signals and supplies the required current to the motor windings.
4. **Power Supply:**
   - Provides the necessary power to both the motor driver and the stepper motor.
5. **Driver Selection**: Choose a stepper motor driver IC (e.g., A4988, DRV8825) compatible with the stepper motor's specifications (e.g., current rating, voltage, steps per revolution).
6. **Connections**:
   - Connect stepper motor windings to the driver outputs (usually labeled as A, A', B, B') and provide power and ground connections to the driver module.
   - Interface the driver control inputs (e.g., step, direction) with GPIO pins of the ARM microcontroller.

**Block Diagram:**

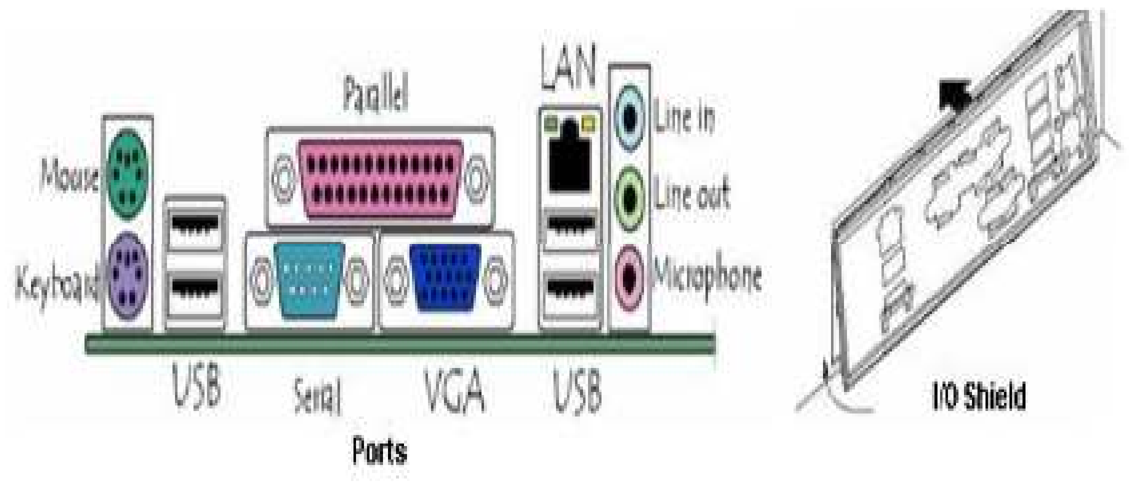


**Fig 1.6 Stepper motor interfacing**

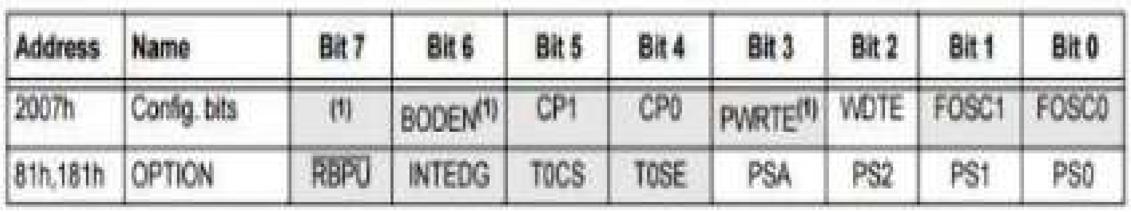
- **Control Logic**:
  - Implement control algorithms (e.g., full-step, half-step, micro stepping) to drive the stepper motor using GPIO outputs to the driver module.
  - Use timers or software delays to control the speed and direction of stepper motor rotation.

**2-MARKS**

**1. What are the I/O memory**

```
4GB +-----------------+   <- 32-bit
    | DRAM            |
    |                 |
2GB +-----------------+
    | Mapped I/O      |
1GB +-----------------+
    | ROM & RAM & I/O |
0GB +-----------------+   0
```

**2.What is watchdog timer**



| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 2007h | Config. bits | (1) | BODEN(1) | CP1 | CP0 | PWRTE(1) | WDTE | FOSC1 | FOSC0 |
| 81h,181h | OPTION | RBPU | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 |

**3.What is memory mapping.**

- ✓ I/O Port addressing is not the only way the processor can communicate with external devices
- ✓ Another commonly used technique is called memory mapped I/O
- ✓ Asserting the I/O pin and addressing a data port, the processor just accesses a memory address.
- ✓ Directly. The external device can have a small amount of RAM or ROM that the processor.

**4. what is SPI protocol**

The term SPI stands for Serial Peripheral Interface.

- This protocol is used to send data between PIC microcontroller and other peripherals such as SD cards, sensors and shift registers.
- PIC microcontroller support three wire SPI communications between two devices on a common clock source.

**5 .What is the TIMER 1?**

Timer1 can operate in one of two modes:

- The clock source can be either the internal system clock (Fosc/4), an external clock, or an external crystal.
- Timer1 can operate as either a timer or a counter.
- When operating as a counter , the counter can either operate synchronized to the device or asynchronously to the device.
- TImer1 also has a presale option which allows TMR1 to increment at the following rates: 1:1, 1:2, 1:4, and 1:8.

**6.What is TIMER 0 ?**

- The Timer0 module is a simple 8-bit overflow counter.
- The clock source can be either the internal system clock or an external clock When the clock source is an external clock, the Timer0 module can be selected to increment on either the rising or falling edge.
- The Timer0 module also has a programmable prescaler option. This prescaler can be assigned to either the Timer0 module or the Watchdog Timer.

**7. What is the function of capture mode?**

The Basic function of capture mode by CCP1 or CCP2 module is that the exact point of time of occurrence of that input edge change can be detected. For this purpose, timer1 must be running in timer mode

**8.Define EEPROM.**

EEPROM (also called E2PROM) stands for electrically erasable programmable read-only memory and is a type of non-volatile memory used in computers,

usually integrated in microcontrollers such as smart cards and remote keyless systems, or as a separate chip device to store relatively small amounts of data by allowing individual bytes to be erased and reprogrammed.

**9.What is SRAM?**

SRAM (static RAM) is a type of random access memory (RAM) that retains data bits in its memory as long as power is being supplied.

Unlike dynamic RAM (DRAM), which must be continuously refreshed, SRAM does not have this requirement, resulting in better performance and lower power usage.

However, SRAM is also more expensive than DRAM, and it requires a lot more space.

**10.Define UART?**

UART stands for Universal Asynchronous Receiver/Transmitter. It's not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone IC. A UART's main purpose is to transmit and receive serial data.

**11.What ADC and DAC interfacing?**

- ✓ The Analog to Digital Conversion is a quantizing process. Here the analog signal is represented by equivalent binary states.
- ✓ The A/D converters can be classified into two groups based on their conversion techniques.
- ✓ In the first technique it compares given analog signal with the initially generated equivalent signal. In this technique, it includes successive approximation, counter and flash type converters.

**12. List some features of ADC 0808/0809:**

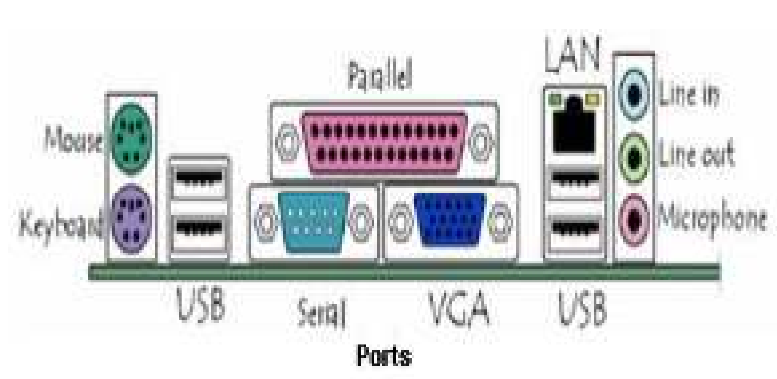
The conversion speed is much higher

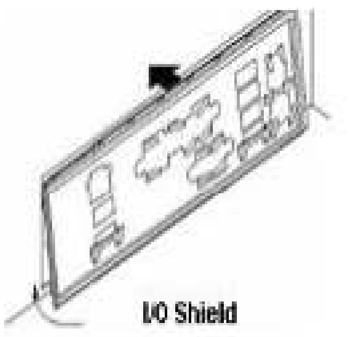
- The accuracy is also high
- It has minimal temperature dependence
- Excellent long term accuracy and repeatability
- Less power consumption

**13. What is stepper motor interfacing?**

A **stepper motor** is made up of a rotor, which is normally a permanent magnet and it is, as the name suggests the rotating component of the motor.

A stator is another part which is in the form of winding. In the diagram below, the centre is the rotor which is surrounded by the stator winding. This is called as four phase winding.





**14.What is timer?**

A timer is **a specialized type of clock used for measuring specific time intervals.** Timers can be categorized into two main types. The word "timer" is usually reserved for devices that count down from a specified time interval called a countdown timer

**15.What is ADC?**

Analog to digital converter is a circuit that converts a continuous voltage value (analog) to a binary value (digital) that can be understood by a digital device which could then be used for digital computation.

These ADC circuits can be found as an individual **ADC ICs** by themselves or embedded into a **microcontroller**. They’re called ADCs for short.