

Assignment - 1:

Date _____
Page No. _____

1. Define an embedded system. Explain its main characteristics with suitable examples.

⇒ An embedded system is a special purpose computer system designed to perform a specific task or a small set of tasks.

Embedded systems are built into larger devices or machines and work as part of the overall system. They usually consist of hardware (microcontroller or microprocessor) and software (firmware).

Main characteristics of Embedded systems.

1. Single functioned:

An embedded system is designed to perform one specific task repeatedly. It does not act as a general purpose computer.

Example:

✓ A washing machine controller only manages washing operations like motor control, timing, and water level sensing.

2. Tightly Constrained:

Embedded systems operate under strict constraints on

- Processing power /
- memory /
- Energy consumption
- size /
- cost /

They must be efficient to work within these limitations.

Example:

A digital thermometer has limited memory and battery, so the firmware must be optimized.

3. Reactive / real time operation:

An embedded system continually monitors its environment and responds to input from sensors or user.

Example:

A temperature control system reads temperature changes and reacts by switching the heater/cooler on or off.

4. Real time operation:

Many embedded systems must provide output within strict deadlines. Missing these deadlines can cause system failure.

Example:

Airbag deployment system must respond within milliseconds during a collision.

Embedded systems are dedicated, constrained, reactive and real time computing systems, widely used in devices like microwaves, mobile phones, medical equipment and automobiles.

2] Explain the constraints (performance, memory, power, reliability, cost) faced in embedded system design.

⇒ Embedded systems face several design constraints because they must be small, efficient, low-cost and highly reliable. The main constraints include:

(i) Performance constraints:

Embedded system must perform specific tasks quickly and efficiently, often in real time.

Performance is limited due to:

- low processing speed
- limited instruction set
- need for fast response.

Example:

An anti lock braking system (ABS) must react immediately to avoid accidents.

(ii) Memory Constraints:

most embedded systems have very limited RAM and ROM because,

- more memory increases cost
- Devices must remain small and low power.

Developers must optimize code to fit into small memory.

Example:

Microcontrollers in home appliances may have only a few kilobytes of RAM.

(iii) Power Constraints:-

many embedded devices run on batteries or low power sources, so they must:

- Use low power processors.
- Reduce energy consumption
- Enter Sleep/ low power modes.

Example:

Wearable devices like smartwatches require efficient power usage to last long.

(iv) Reliability Constraints:

Embedded systems often operate in critical situations where failure is unacceptable. They must be,

- Error free
- fail safe
- Able to run continuously for years.

Example:

pacemakers must work reliably 24/7 without failure

(v) Cost Constraints:

most embedded systems are produced in large quantities, so cost must be minimized.

Designers must choose,

- low cost microcontroller
- Minimal hardware components
- Efficient manufacturing.

Example:

A TV remote is made at very low cost because millions are produced.

3.] Describe the different types of embedded system with examples.

⇒ Embedded systems can be classified based on performance, functionality and complexity.

* Based on Performance :-

(i) Small Scale Embedded system :

Built using low cost 8-bit or 16-bit micro-controllers with limited memory and processing power.
Examples:

Toys, TV remote controls, simple temperature sensors used for simple tasks.

(ii) Medium Scale Embedded system :

Use 16-bit or 32-bit microcontrollers or DSPs (Digital Signal Processors). They have more memory and support real time operations.

Examples:

Smart washing machines, Automotive engine control units, printers, routers, etc.

(iii) Large Scale Embedded system :

Complex systems designed using powerful 32-bit or 64-bit processors. They support large software

OS, and networking.

Examples:

Smart TVs, advanced medical equipment, etc.
Industrial automation systems, Aircraft control systems.

* Based on functionality :

(i) Standalone Embedded system :

These systems work independently without relying on a host system (like a computer).

Examples:

Calculator, Digital camera, microwave oven, MP3 player.

These devices take input, process it and produce output on their own.

(ii) Real Time Embedded system :

These systems must perform operations within strict time limits (deadlines). They are divided into

a) Hard Real Time systems :

Missing a deadline can cause catastrophic failure.

Examples:

Airbag Control System, Anti lock Braking System (ABS), pacemakers.

b) Soft Real Time Systems :-

Missing deadlines reduces performances but is not dangerous.

Examples:

Online video streaming, Computer games, multimedia systems.

iii) Networked Embedded Systems :

These systems are connected to a network (LAN, WAN or Internet) and communicate with other devices.

Examples:

Smart home devices (IoT), ATM machines, security alarm systems, smart meters.

Used in IoT applications and remote monitoring.

(iv). Mobile Embedded Systems :

These systems are portable and battery powered, installed inside mobile devices.

Examples:

Smartphones, Tablets, Digital watches, fitness bands.

Designed for mobility and low power consumption.

4.] Compare embedded system with general purpose computer systems based on architecture, functionality, performance, and constraints.

⇒ Comparison between embedded systems with general purpose computer systems.

Based on	Embedded systems	General purpose computers
→ Architecture	→ Application specific with microcontroller, limited memory and I/O	→ Complex architecture with powerful CPU, large memory and many I/O devices
→ Functionality	→ Designed for a specific dedicated task	→ Designed to perform many different tasks and applications
→ Performance	→ Optimized for real time and task specific performance.	→ High performance for multitasking, large applications and heavy workloads.
→ Constraints	→ Strict constraints on memory, power, cost and size.	→ Fewer constraints, more flexible with larger memory, power usage and cost.

→ User Interface	Simple or no user interface.	→ Rich GUI with keyboard, mouse, display etc.
→ flexibility	→ low flexibility, hard to reprogram or upgrade.	→ High flexible, easy to install, update and customize.

Q.N.5] Explain advantages and limitations of embedded systems compared to general purpose systems.

⇒ Embedded systems have advantages and some disadvantages in compare to general purpose computers. Some are mentioned below.

Advantages of embedded systems.

→ highly efficient: Designed for a specific task, so they work faster and more efficient way.

→ low cost: Use minimal hardware, so the overall system cheaper.

→ Small size: Compact and easy to integrate into devices.

→ low Power Consumption: Suitable for battery-powered and

portable devices.

→ high reliability: perform consistently and require the less maintenance.

Limitations of embedded systems:

→ limited functionality: Can perform only some specific tasks.

→ Difficult to upgrade: Hardware and software updates are restricted or impossible.

→ Resource constraints: limited memory, processing power and storage.

→ less flexible: Cannot run multiple application like general system.

→ Harder to debug: Errors are difficult to trace due to tight hardware software integration.

→ Real time constraints.

→ Cost and performance trade off.

→ Reliability, safety and maintenance.

Q. No. 1] Explain real time constraints in embedded systems and why such constraints are not typical in general purpose systems?

⇒ Real time constraints in embedded systems:

Real time constraints refer to the requirement that an embedded system must respond to an event within a strict and guaranteed time limit. The correctness of the system depends not only on the output but also on how fast the output is produced.

Eg. Airway systems, medical monitors and industrial controllers must react in milliseconds.

General purpose systems (like PCs or laptops) do not require strict timing because their primary goal is high performance and multitasking, not guaranteed response time. Delays of a few milliseconds or seconds do not cause failure in general computer applications like browsing, gaming or editing documents.

Q. No. 2] List the major applications domains of embedded systems with example for each domain and its most critical real-time constraints.

⇒ The application domains of an embedded system are very large. Some are given below:

→ Automobile Applications: Embedded systems are widely used to improve safety, performance and reliability in vehicles.

→ Engine control unit (ECU) for fuel management.

→ Air bag control systems.

→ Anti-lock Braking Systems (ABS).

→ Power windows and mirrors.

→ Health care Applications:

Embedded systems are used for monitoring and life saving medical functions.

Eg. → Ventilators for heart and respiration.

→ Blood glucose monitors.

→ MRI/CT scanners.

→ Infusion pumps for controlled medication delivery.

→ Industrial Control Applications:

Used for automation, monitoring and controlling industrial processes to increase efficiency and

Safety.

- eg. → programmable logic controllers (PLCs).
- Robotics arms in manufacturing.
- Conveyor Control Systems.
- Temperature and pressure monitoring systems.

Q.N.8] Explain the role of embedded system in IoT.
Discuss any three real world IoT applications.

⇒ Roles of Embedded system in IoT:
Embedded systems act as the core hardware and software units in IoT devices. They collect data using sensors, process it using microcontrollers and communicate with other devices or cloud services through wireless network. They enable automation, remote monitoring, and intelligent decision making in IoT applications.

Some Real World IoT applications:

→ Smart home systems:

Devices like smart lights, smart thermostats, and security cameras use embedded sensors and wifi modules to allow remote control and automation through smartphones.

→ Wearable Health Devices:

Fitness bands and smart watches use embedded microcontrollers to track heart pulse, rate steps, sleep patterns and send data to mobile apps for health monitoring.

→ Smart Agriculture:

IoT sensors, embedded in soil, weather stations and irrigation systems monitor moisture temperature and crop conditions enabling automated watering and improved farm productivity.

Q.N.9] Write short notes on embedded systems in the automotive industry.

⇒ Embedded systems in the automotive industry.

Embedded systems are extensively used in modern vehicles to improve safety, performance, comfort and automation. They are integrated into various subsystems to monitor and control operations in real time. Key application includes,

→ Engine Control Unit (ECU): optimize fuel injection ignition timing and emissions.

→ Safety systems: AntiLock Braking System (ABS)

Airbag Control, traction Control and stability systems.

→ Driver Assistance: parking sensors, lane departure warning, adaptive cruise control.

→ Entertainment Systems: Touchscreen displays, GPS navigation and multimedia control.

→ Electric and hybrid vehicles: Battery management, motor control and regenerative braking system.

Embedded systems in vehicles enhance reliability, efficiency, and safety while enabling features like autonomous driving.

Q.N.10] Discuss the importance of embedded systems in the healthcare sector with suitable examples.

⇒ Importance of embedded systems in the health-care sectors.

Embedded systems play a crucial role in health-care by enabling accurate monitoring, diagnosis, treatment, and life support functions. They improve patient safety, reduce human

error and allow remote healthcare services.

Key examples include:

→ Pacemakers: Regulate heartbeat automatically.

→ Blood glucose monitors: Measure and track blood sugar levels.

→ Infusion pumps: Deliver precise doses of medications.

→ MRI/CT scanners: Control imaging hardware and process data for diagnosis.

→ Wearable Health Devices: Track vital signs such as heart rate, oxygen level, and sleep patterns.

These systems enhance efficiency, enable real-time monitoring and support critical medical interventions.

Q.N. 11] Explain industrial automation using embedded systems. Describe with its benefits and challenges.

⇒ Industrial Automation using embedded system:

Industrial automation involves using embedded system to control, monitor and optimize manufacturing and production processes with minimal human intervention. Embedded controllers, sensors, actuators and communication modules are integrated to automate tasks like assembly, material handling, quality control and process monitoring.

* Benefits :-

- Increased efficiency : machine operates faster and continuously.
- Improved Accuracy : Reduce human errors in repetitive tasks.
- Cost Reduction : lowers costs and material waste.
- Enhanced Safety : Dangerous operations are handled by machines.
- Real-Time monitoring : Quick detection of faults or process deviations.

* Challenges :-

- High Initial Cost : Installing automated systems requires significant investment.
- Complexity : Designing, programming and maintaining embedded systems is technically challenging.
- flexibility limitations : Hard to change once it was built.
- Reliability requirements : System failure can halt production or cause safety hazards.

Q.N. 12] Explain the architecture and features of typical microcontroller used in embedded systems.

⇒ The microcontroller that is used in embedded system has a different architecture and features than general one.

* Architecture of a typical microcontroller :

A microcontroller is compact integrated circuit design for embedded system. It combines CPU, memory and peripherals on a single chip to control specific tasks. The typical architecture includes :-

- Central processing unit (CPU): For processing
- Memory:
 - ROM/Flash: Stores firmware / program code
 - RAM: Temporary storage for data during execution.
 - EEPROM (optional): Non volatile data storage.
- Input/output (I/O) ports: to interface external devices.
- Timers/Counters: Generate precise timing signals for tasks.
- Serial Communication Modules: UART, I2C, SPI
Interfere for device communication.
- Interrupt Controller: Allow the microcontroller to respond immediately to external events.
- Analog peripherals (optional): ADC / DAC for interfacing analog signals.

* Features of typical microcontroller:

- Compact and low power
- Dedicated task handling
- Real time processing
- Cost effective
- Versatile I/O operations

Examples: AVR, Cortex-M series, 8051 (ATmega), PIC microcontroller.

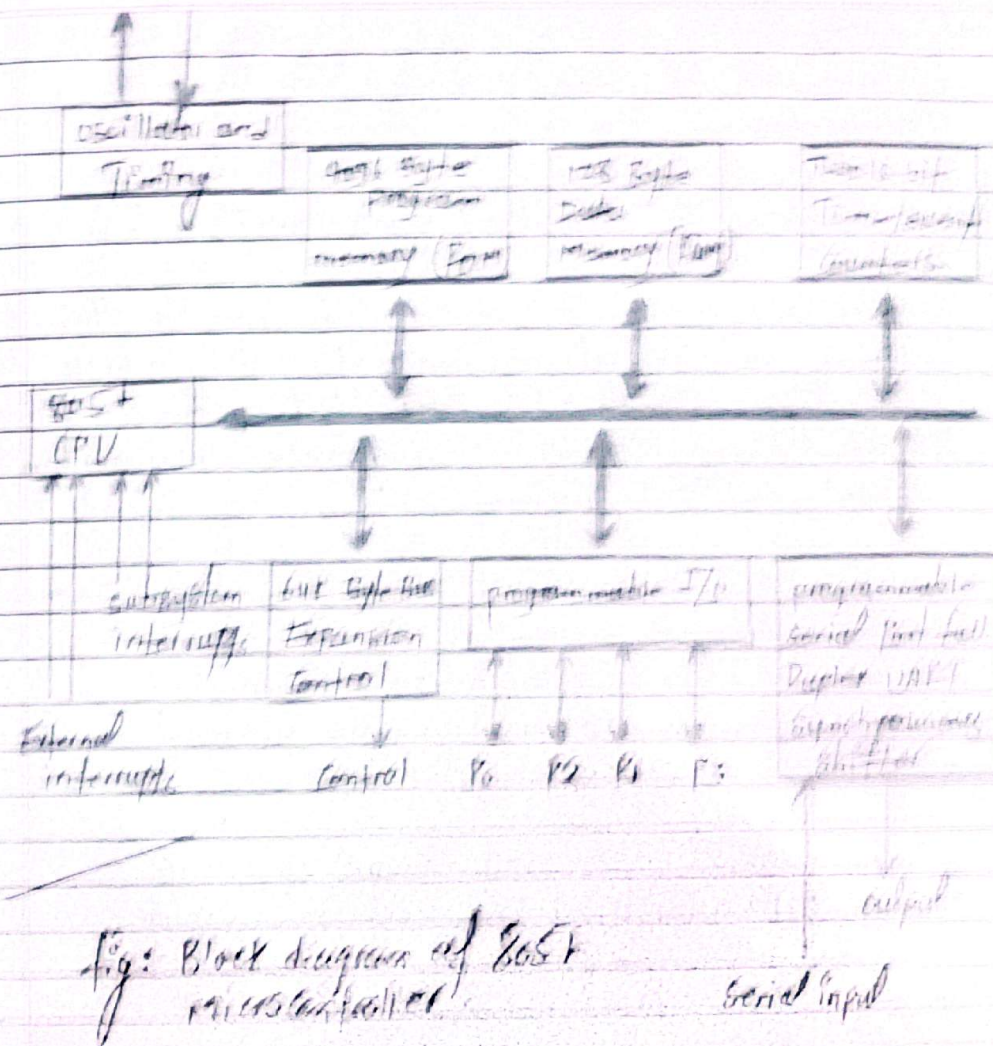


Fig: Block diagram of 8051 microcontroller

Q.13] Describe different types of sensors and their role in embedded systems. Give examples.

⇒ Sensors in Embedded systems :-

Sensors are devices that detect changes in the physical environment and convert them into electrical signals that embedded systems can process. They play a crucial role in monitoring, control and automation.

Types of Sensors :-

1. Temperature sensors: Measure heat changes.
Eg. LM35, Thermistor, Used in thermostats, climate control systems.
2. Pressure sensors: Measure change in pressure.
Eg. MPX5010, used in industrial automation etc.
3. Proximity / Distance sensors: Detect presence or distance of object. Eg. Ultrasonic sensors.
4. Light sensors: Detect light intensity or changes.
Eg. LDR (light dependent resistor), photodiodes.

5. Motion sensors: Detect movement or acceleration.
Eg. PIR sensors, accelerometers.

6. Gas sensors: Detect the presence of gases like CO, LPG, or smoke.

7. Humidity sensors: Measure moisture levels in the air. Eg. DHT11, DHT22.

* Roles of sensors in embedded systems :-

Sensors provide real time data to the microcontroller, enabling the system to monitor conditions, make decisions and control actuators automatically. For example, a temperature sensor is a thermostat triggers the heater or cooler based on the room temperature.

Q.214] Discuss the importance of peripherals (I/O, Ports, timers, ADC, Communication Interfaces) in embedded system design.

⇒ Importance of peripherals in embedded system design :-
peripherals are essential components that extend the functionality of an embedded system, allowing it to interact with the external world and perform precise tasks.

1. Input/output (I/O) ports:

- Enable the microcontroller to interface with sensors, switches, LEDs and other external devices.
- With out I/O ports, the system cannot receive inputs or send outputs.
- Eg. Reading a push button or controlling a motor.

2. Timer and Counters:

- Generate precise time delays, measures time intervals and count event.
- Crucial for real time operations, PWM control and scheduling tasks.
- Eg. Generating PWM signals for motor speed control or controlling LED blinking patterns.

3. Analog to Digital Converters (ADC):

- Converts analog signals from sensors into digital signals.
- Enables embedded systems to interact with real world analog signals.
- Eg. Reading light intensity from an LDR.

4. Communication Interfaces:

- Enable data exchange between microcontroller, computers and other device using protocol like UART, I2C etc.
- Essential for networking, remote monitoring and IoT application.
- Eg. Sending sensor data to a smartphone app via I2C.

Assignment - 2:

Q. No. 1] Explain the AVR microcontroller architecture with a neat block diagram. Describe each component in detail.

⇒ The AVR microcontroller architecture feature a RISC CPU at its core, connected via a data bus to key components like flash memory (Program Storage), SRAM (data variable), EEPROM (non-volatile data), a register file, ALU, program counter, status register, I/O ports, Timer, ADC and communication peripherals (UART, SPI, I2C).

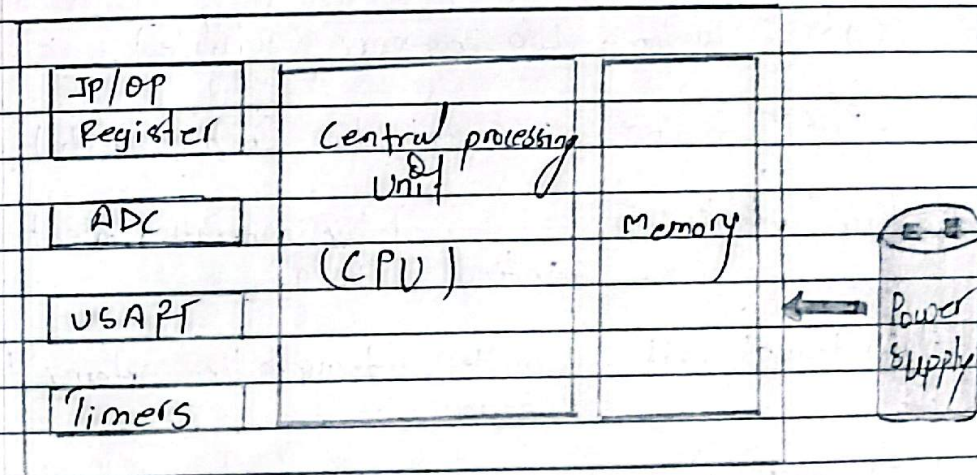


fig: Block diagram showing Architecture of AVR microcontroller.

X. Components:

(i) CPU Core: Executes instructions, performs arithmetic/logic operations.

(ii) ALU: handles additions, subtraction, shifting, logical operations.

(iii) Register file (32 Registers): Used for fast data access, reduce memory access time.

(iv) Flash memory: store program code (non-volatile).

(v) SRAM: Runtime data memory (volatile).

(vi) EEPROM: store permanent data (non-volatile).

(vii) Timer/Counters: used for delay generation, event counting, PWM.

(viii) Interrupt unit: handles interrupts like external, timer, serial.

(ix) I/O ports: used to connect LEDs, motors, sensors, displays.

x) Communication Interface: UART, SPI, I2C.

xi) Clock unit: provides timing for CPU operations.

Q.2] Describe the register structure of AVR microcontroller. Explain the role of general purpose registers and special function registers.

⇒ Register structure of AVR microcontroller.

(i) General purpose Register (GPRs):

R ₀	R ₁	R ₂	R ₃	R ₄	R ₅	
R ₆	R ₇	R ₈	R ₉	R ₁₀	R ₁₁	

→ AVR has 32 register (R₀ - R₃₁).

→ Directly connected to ALU

→ single cycle execution.

→ Registers R₂₆ - R₃₁ act as index registers.

→ X = R₂₆ : R₂₇

→ Y = R₂₈ : R₂₉

→ Z = R₃₀ : R₃₁.

(ii) Special Function Registers (SFRs):

SFR			TCR0 / TCCR1
SPE	TIMSK	TIMSK	ADCSRA
			DDRx

Control hardware peripherals :-

→ DDRx: Data Direction Register (I/O Configuration).

→ PORTx: output latch.

→ PSNx: Reads external pin state.

→ TCCR0, TCNT0: Timer Control and Counter.

→ UCSRA, UDF: USART registers.

→ ADCSRA, ADMUX: ADC register.

→ SREG: Status register (C, Z, N, V flags etc).

fig: register file in AVR microcontroller.

	Address
R0	0X00
R1	0X01
.....	
R13	0X0D
R14	0X0E
R15	0X0F
R16	0X10
.....	
R26	0X1A X-register low byte
R27	0X1B X-register high byte
R28	0X1C Y- " low byte
R29	0X1D Y- " high byte
R30	0X1E Z-register low byte
R31	0X1F Z- " high byte

Q. N3] Explain the memory organization of AVR (Flash, SRAM, EEPROM) with suitable diagram.

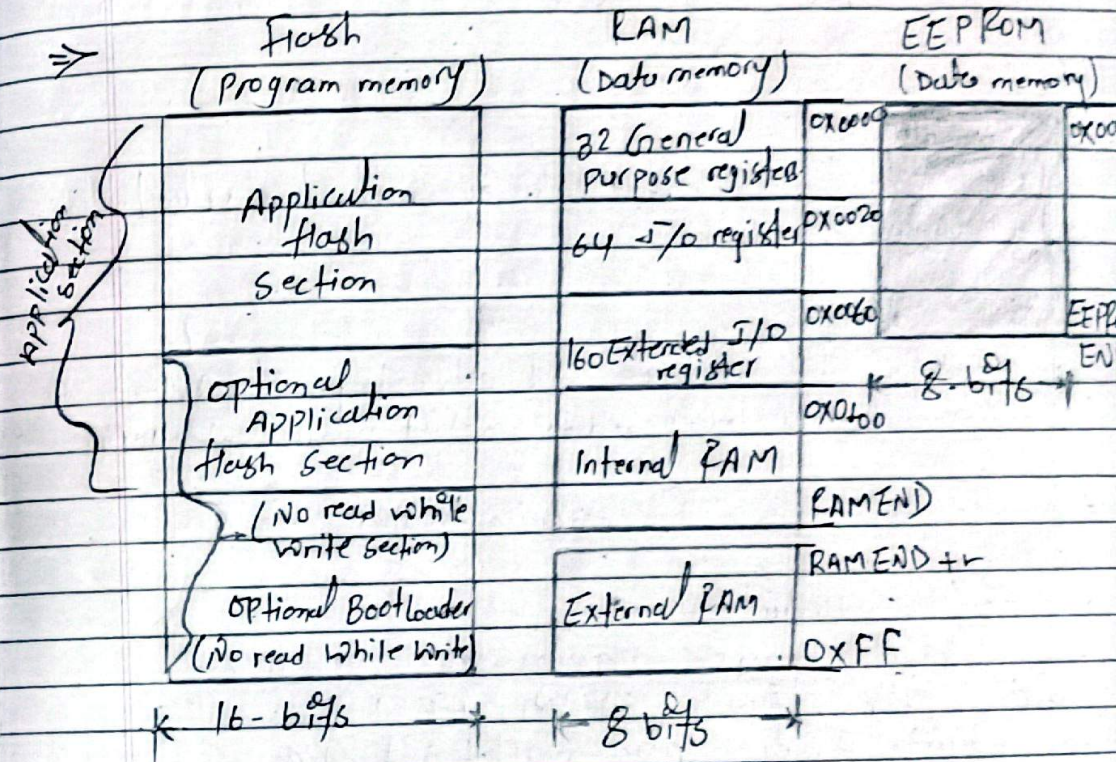


fig: above all three are the memory organization of AVR (Flash, SRAM, EEPROM) microcontrollers

* Flash :

→ Store program code.

→ non-volatile.

→ it is 16-bit.

* SRAM :

- Stores temporary variable
- Stack grow downward
- it is 8-bits.

* EEPROM :

- Used for permanent data (e.g. calibration value).
- it is also 8-bit.

Q.4] Explain the features and advantage of AVR microcontroller over other 8-bit microcontrollers.

⇒ Features of AVR microcontroller :

- RISC Architecture.
- 32 general purpose registers.
- on chip flash, SRAM, EEPROM.
- Inbuilt timer, ADC, USART, SPI, I2C.
- up to 20 MIPS at 20 MHz.
- low power consumption.

* Advantages of AVR microcontroller.

- faster execution (single cycle instructions).
- Easy to program (Embedded C).
- Excellent peripherals support.
- cost effective.

Q.2.5] Explain the structure of an embedded C program with a suitable example.

⇒ Here Example :

Program: Blink an LED connected to PBD of AVR microcontroller.

```
#include <avr/io.h>
```

```
#include <util/delay.h>
```

```
int main(void) {
```

```
    DD PB1 = (1 << PBD);
```

```
    while (1) {
```

```
        PORTB |= (1 << PBD); // Turn LED ON
```

```
        delay_ms(500);
```

```
        PORTB &= ~(1 << PBD); // Turn LED off
```

```
        delay_ms(500);
```

```
    }
```

```
}
```

* Explanation :-

(i) Header file inclusion :

```
#include <avr/io.h>
```


- Obtain definitions of hardware registers (DPFR, PDRFR, PSNR etc).
- Allows to the program to access microcontroller peripherals.

(ii) Global variable declaration (optional):
used for counters, buffers, flags.

(iii) Function prototypes (optional):
used when multiple functions are defined.

(iv) main function:

```
int main(void)
{
    // ---
}
```

- The main function executes on reset startup.
- After initialization, embedded programs usually run forever so while(1) infinite loop is used.

(v) Peripheral initialization:

This involves:-

- setting pin direction
- configuration timer
- setting up ADC, DAC
- Enabling interrupt

Example: `DPFR = (1 << PDR); // set PDR as`

(vi) Infinite loop (super loop):

Embedded system run continuously, they should not terminate.

Example

```
while(1) {
```

```
    // --- main program
}
```

(vii) Interrupt service routines (optional)

Defined when interrupts are used.

Ex: P

Qp 6] Discuss bitwise operation in Embedded C and their importance in hardware level programming. provide example.

⇒ Bitwise operation in Embedded C can used to manipulate individual bits of a number or hardware register.

* Types of Bitwise operators

(i) Bitwise AND (&): used to clear a bit

→ ANDing with 0 clear a bit

→ ANDing with 1 keep the bit unchanged

Example: `PORTB &= 2 (1 << PDR);`

- (ii) Bitwise OR ($|$): used to set a bit.
 → OR with 1 set the bit
 → OR with 0 keep the bit unchanged

Example: $PORTB |= (1 << PB0);$

- (iii) Bitwise XOR ($\wedge$): used to toggle a bit.
 → XOR with 1 toggles the bit
 → XOR with 0 leaves it unchanged.

Example: $PORTB \wedge= (1 << PB0);$

- (iv) Bitwise NOT ($\sim$): inverts all bits.
 → used to flip every bit in a register.

Example: $PORTB = \sim PORTB;$

- (v) Left Shift ($<<$): move bits to left
 → used to create bit masks.

Example: $(1 << PB3);$

- (vi) Right Shift ($>>$): move bits to right.
 → used to read specific bit value:

Example: $value = (PIND >> PD2) \& 1;$

* Bitwise Operators Are Important in Embedded C.

- setting or clearing ports pins
- Configuration registers
- Handling interrupt
- Reading sensor inputs
- creating masks for hardware control
- Efficient memory usage.

Example:

```
#include <avr/io.h>
int main(void){
```

```
DDRB |= (1 << PB0);
```

```
while(1){
```

```
PORTB \wedge= (1 << PB0);
```

```
for( volatile long i=0; i<2000; i++);
```

```
}
}
```


S.m.f

Date
Page No

Q.7] Explain various data type used in Embedded C. How do they affect memory usage and performance in microcontrollers?

⇒ Common Data type used in Embedded C.

(i) Char (8-bit):

→ size 1 byte.

→ Range:

→ signed char: -128 to +127

→ unsigned char: 0 to 255

* used for:

→ storing small values.

→ Accessing 8-bit registers.

→ ASCII character.

(ii) int (16-bit in AVR):

→ size 2 bytes.

→ Range:

→ signed int: -32768 to +32767

→ unsigned int: 0 to 65535

* Used for:

→ counters

→ loops

→ medium sized integer operations.

(iii) long and unsigned long (32-bit)

→ size 4 bytes.

* used for:

→ large numbers.

→ Time delay calculation

→ sensor reading

(iv) float (32 bit)

→ size 4-byte.

* used for:

→ Decimal values

→ Scientific calculations.

(v). double:

→ same as float in AVR (32-bit)

→ used rarely due to heavy processing.

(6) volatile Data type:

special keyword used for variables that can change unexpectedly.

* used for:

→ Interrupt variables

→ I/O registers

→ sensor values.

Date
Page No

* Data types Important in Embedded System.

→ Memory is very limited:

→ Typical AVR SRAM is only 1 KB or 2 KB.
→ used correct types saves memory.

→ Speed optimization:

smaller types and faster execution.

→ Prevent overflow:

Correct type selection avoids unexpected behavior.

→ Hardware register access:

8-bit data types match microcontroller register size.

→ Efficient handling of sensors, timers and interrupts.

Imp

Q.08] Explain Control Structures (if else, switch, loop) used in Embedded C with example relevant to embedded application.

⇒ Types of Control Structures:

Control Structures in embedded C are divided into three categories.

1) Decision making (selection) statements:

a) if statement:

Executes a block only when the condition is true.

eg.

```
if (PIND & (1 << PDI)) {  
    PORTB |= (1 << PBO);  
}
```

b) if else statement:

used when two possibilities exist.

eg.

```
if (temp > 50) {  
    PORTB |= (1 << PBO);  
}  
else {  
    PORTB &= ~(1 << PBO);  
}
```


c. Switch case Statement

used when multiple condition need to be checked.

Example:

```
switch (mode) {  
    case 1: PORTB = 0x01; break;  
    case 2: PORTB = 0x02; break;  
    case 3: PORTB = 0x03; break;  
    default: PORTB = 0x00;  
}
```

2. Looping (Iteration) statements.

a) for loop:

used for fixed number of repetitions.

Example:

```
for (int i = 0; i < 10; i++) {  
    PORTB ^= (1 << PBO);  
}
```

b) while loop:

Runs as long as the condition remains true.

Example:

```
while (PIND & (1 << PDL)) {  
    PORTB |= (1 << PB2);  
}
```

c) do while loop:

Executes the block at least once

Example: do {

Count ++;

}

while (count < 1000);

3. Jump statements

a) break:

used to exit loops or switch case

Example:

```
while (1) {  
    if (flag == 1)  
        break;  
}
```

b) continue:

skips the remaining statement in a loop.

Example:

```
for (i = 0; i < 10; i++) {  
    if (i == 5)  
        continue; // skip when i = 5.  
}
```

c) return:

used to exit a function.

- # Control Structure are important in Embedded C
- To perform decision based tasks.
 - To create repetitive operations (LEDs blinking, delays).
 - To implement state machines.
 - To improve program structure and readability.
 - To manage complex embedded applications.

~~S.M.~~

Q.N.9] Explain how pointers are useful in accessing hardware registers. Give suitable code example.

⇒ In embedded systems (like AVR microcontrollers), hardware registers such as PORTx, PINx, DDRx, TCCR0, etc are stored at fixed memory addresses.

* Here some reason are why pointers are useful :-

i) Direct access to hardware:

Hardware registers are mapped to specific memory addresses. A pointer can be assigned to these address to read/write them directly.

ii) Faster execution:

Pointer based register manipulation avoids functions overhead and gives low level control.

iii) flexibility :-

Same pointer based technique works for different registers by changing only the address.

iv) Mandatory in embedded C.

In many microcontrollers, registers are not variables they exist at fixed memory locations, so pointers are the easiest way to access them.

Example: setting PORTB as output using pointer

```
#define DDRB_ADDR 0x37
#define PORTB_ADDR 0x38
```

```
int main() {
```

```
    volatile unsigned char *ddrb = (unsigned char *) DDRB_ADDR;
```

```
    volatile unsigned char *portb = (unsigned char *) PORTB_ADDR;
```

```
    *ddrb |= (1 << 0);
```

```
    *portb |= (1 << 0);
```

```
    while (1) {
```

```
    }
```


S.m.f

Date
Page No.

10.] Discuss memory constraints in microcontrollers and techniques to optimize memory usage.

⇒ Microcontrollers (like AVR) have very limited compared to PCs. Typical AVR chips (eg. ATmega-16, ATmega328) contain:

- Flash (Program memory): stores code (8KB - 64KB).
- SRAM (Data memory): stores variables, stack (1KB - 8KB)
- EEPROM: stores permanent data (512B - 4KB)

Because the available memory is very small, the programmer must write code that uses memory efficiently.

Memory constraints in microcontroller:

1. Limited SRAM:

- RAM is only a few KB (kilobyte).
- used for global variables, local variables, stack, and heap.
- Risk of stack overflow or running out of memory.

2. Limited flash memory:

- Program size must fit into flash.
- Large libraries or heavy code cannot be used.

3. No external memory (in most cases):

- unlike PCs, microcontrollers rarely support external RAM or flash.

4. Stack limitations:

- Deep recursion can cause stack overflow.
- Large local variables are risky.

5. No Dynamic Memory Management.

- malloc() and free() are not recommended due to fragmentation.

6. Interrupt usage affects memory.

- Every interrupt consumes stack space.
- Too many interrupts increase RAM usage.

Techniques to optimize memory usage.

(A) optimizing SRAM usage:

1. use const to store data in flash instead of SRAM.
 - stay in flash, not in RAM.
 - saves SRAM significantly.

2. Used global variable wisely.

- Globals stay in memory throughout the program.

→ Reduce their usage when possible.

3. Avoid larger local variables.

→ local arrays inside functions consume stack memory.

4. Avoid recursion.

→ uses stack heavily.

→ Rewrite recursion as iterative logic.

5. Use smaller data type.

(B) Optimizing Flash Memory Usage:

1. Avoid unnecessary library functions.

→ Large functions (like printf) increase code size.

2. Reduce code duplication.

→ Use functions instead of repeating code.

3. Use macros only when needed.

→ Macros increase code size if used excessively.

(C) EEPROM usage optimization.

→ Store only essential permanent data.

→ Avoid excessive write operations (EEPROM has limited write cycle ~100,000).

J.m.f

Q.P.11] Explain the Interrupt System of AVR microcontroller with a diagram of the vector table.

⇒ Interrupt System of AVR microcontroller: Key features:

1) Interrupt Vector table

→ A table stored at the beginning of program memory.

→ Each entry contains the address of the ISR for a specific interrupt.

→ The CPU jumps to this address when the interrupt occurs.

2) Global Interrupt Enable Bit (I-bit):

→ located in status register (SREG, bit 7)

→ must be ^{set} to allow interrupts.

3) Individual Interrupt Enable bits:

→ Each interrupt has its own enable bit in specific control registers.

4) Interrupt flags:

→ Set by hardware when an interrupt event occurs.

→ found in registers like GIFR, TIFR.

5) Execution flow:

→ Execute ISR

→ return using ret.

Interrupt Vector Table (Diagram)

(Based on ATmega16/32: similar for many AVR controllers):

Address	Interrupt Source
0x0000	→ Reset Vector (Reset ISR)
0x0002	→ INTO External Interrupt Request 0
0x0004	→ INT1 External Interrupt Request 1
0x0006	→ Timer2 Compare Match
0x0008	→ TIMER2 overflow
0x000A	→ TIMER1 Capture event
0x000C	→ TIMER1 Capture Match A
0x000E	→ TIMER1 capture Match B
0x0010	→ TIMER1 overflow
0x0012	→ TIMERO Compare Match
0x0014	→ TIMERO overflow
0x0016	→ SPI Serial Transfer Complete
0x0018	→ USART, Rx Complete
0x001A	→ USART, UDR Empty
0x001C	→ USART, Tx Complete
0x001E	→ ADC Conversion Complete
0x0020	→ EEPROM Ready
0x0022	→ Analog Comparator
0x0024	→ Two wire serial Interface (TWI)
0x0026	→ Store program Memory ready

Q.12] Describe the steps involved in Configuring and handling interrupts in AVR using Embedded C.

⇒ Interrupts in AVR allow the CPU to respond automatically to external or internal events.

Steps to Configure and handle Interrupt in AVR.

Step-1] Include Required Header files

#include <avr/io.h>

#include <avr/interrupt.h>

→ <avr/io.h> provides register definitions.

→ <avr/interrupt.h> provide ISR function.

Step-2] Configure the Interrupt Source

Depending on the Interrupt type, we need to configure related registers.

Example: Configure External Interrupt INTO
MCUCR |= (1 << ISC01);

For timers:

TIMSK |= (1 << TOIE0);

For ADC:

ADCSRA |= (1 << ADIF);

Step-3] Enable the specific Interrupt
Enable interrupt in its control register.

Example For INT0:

```
GICR |= (1 << INT0);
```

For Timer1:

```
TIMSK |= (1 << TOIE0);
```

Step-4] Enable Global Interrupts

The 1-bit (Global Interrupt Enable) in

SREG must be set:

```
sei();
```

To disable all interrupts:

```
cli();
```

Step-5] Write the ISR (Interrupt Service Routine)

→ This is the function that runs automatically
when the interrupt occurs.

→ The ISR name corresponds to the vector in
the interrupt table.

Example: ISR for INT0 interrupt:

```
ISR(INT0_vect) {
```

```
// ----
```

```
}
```

for ISR for Timer1 overflow:

```
ISR(TIMER1_OVF_vect) {
```

```
// ----
```

```
}
```

Step-6] main program runs normally

```
int main() {
```

```
MULCR |= (1 << ISC0);
```

```
GICR |= (1 << INT0);
```

```
sei();
```

```
while(1) {
```

```
// main program loop
```

```
}
```

```
}
```

The CPU runs the main loop.

Q.2.
13.] Discuss different types of interrupts in AVR (external, timer, serial). Give examples.

⇒ AVR supports several types of interrupts. Below is a clear discussion of the three main types you asked for, with examples.

1. External Interrupts:

External interrupts are generated by external pins when a specific signal change occurs.

In most AVR MCUs (like ATmega328P):

→ INT0 → PD2

→ INT1 → PD3

They can be triggered by:-

→ low level

→ Any logical change

→ falling edge

→ rising edge

Example use

→ Push button press

→ motion sensor output

→ External hardware signal

Example: `#include <avr/io.h>`

`#include <avr/interrupt.h>`

`ISR (INT0_vect) {`

`PORTB ^= (1 << PB0);`

`}`

`int main (void) {`

`DDRB |= (1 << PB0);`

`PORTD |= (1 << PD2);`

`EIFR |= (1 << INT0);`

`EIFR |= (1 << ISC0);`

`sei();`

`while(1);`

3

2) Timer Interrupts:

Timer interrupts are generated when a timer/counter reaches a specific condition:

→ Overflow

→ Compare Match

→ Input capture

AVR has multiple timers (Timer0, Timer1, Timer2).

Example Use:

→ Delay generation

→ PWM signal generation

→ periodic task execution (eg. blinking LED every 1s)

Example: `#include <avr/io.h>`

`#include <avr/interrupt.h>`

`ISR (TIMER0_OVF_vect) {`

`PORTB ^= (1 << PB0);`


```

int main(void) {
    DDPRB = (1 << PRB0);
    TCCR0B = (1 << CS01);
    TIMSK0 = (1 << TOIE0);
    sei();
    while(1);
}

```

3. Serial (USART) Interrupts:

Serial interrupts are generated by the USART module during serial communication.

Common USART interrupts:

- RX Complete: Data received.
- TX Complete: Data transmitted.
- Data Register Empty.

Example use:

- Receiving commands via USART
- Communication with PC, GSM, Bluetooth modules.

Example: #include <avr/io.h>
#include <avr/interrupt.h>

```

ISR(USART_RX_vect) {
    char data = UDR0;
    PORTB = data;
}

```

```

int main(void) {
    DDPRB = 0xFF;
    UBRR0 = 103;
    UCSRB = (1 << RXEN0) | (1 << RXCIE0);
    UCSPO = (1 << UCSZ0H) | (1 << UCSZ0L);
    sei();
    while(1);
}

```

Q.2] Explain how ISR (Interrupt Service Routine) works
[A] Write an example Embedded C program for external interrupt.

⇒ An interrupt service routine (ISR) is a special function that is automatically executed when an interrupt occurs.

Working steps:-

1. Normal program execution:

The AVR CPU executes instructions in the main() loop.

2. Interrupt Occurs:

An interrupt request is generated (eg. external pin INT0 goes low).

3. CPU finishes current instruction:
AVR completes the instruction it is executing.
4. Program Counter (PC) is saved:
The address of the next instruction is pushed onto the stack.
5. Global interrupts disabled automatically:
prevents another interrupt during ISR execution.
- 6) ISR is executed:
CPU jumps to the ISR address (interrupt vector).
- 7) ISR completes:
RETI (return from interrupt) instruction is executed.
8. Program resumes:
PC is restored from the stack and normal program continues.

Example: `#include <avr/io.h>`
`#include <avr/interrupt.h>`

```
ISR (INT0_vect) {
    PORTB ^= (1 << PB0);
}
```

```
int main(void) {
```

```
    DD PB 1 = (1 << PB0);
```

```
    PORTD 1 = (1 << PD2);
```

```
    EIMSK 1 = (1 << INT0);
```

```
    EICRA 1 = (1 << ISC0L);
```

```
    sei();
```

```
    while(1);
```

```
}
```

Q.15] Explain the configuration and operation of I/O ports in AVR. Discuss DD_{ix}, PORT_{ix} and PIN_{ix} registers.

⇒ AVR I/O Port Overview:

In AVR microcontrollers (like ATmega 16/32, ATmega 328), digital I/O pins are organized into ports such as PORTA, PORTB, PORTC, PORTD etc.

→ Each port usually has 8 pins (PA0-PA7, PB0-PB7, ...)

→ Every I/O port is controlled using three main registers.

→ DD_{ix}: Data direction register

→ PORT_{ix}: Data register.

→ $PSINx$: Input pins Address register.
Where ($x = A, B, C, D, \dots$).

1. $DDRx$: Data direction register.
 $DDRx$ decides whether a pin acts as Input or output.

Bit value.

$DDRx \text{ bit} = 1$; pin is output.

$DDRx \text{ bit} = 0$; pin is input.

Example: $DDRB = 0xFF$;
 $DDRC = 0x00$;

2. $PORTx$: Data Register

$PORTx$ has two different roles, depending on pin direction.

a) When Pin is Output:

$PORTx$ is used to write data to the pin.

→ $PORTx = 1$; output high (5V).

→ $PORTx = 0$; output low (0V).

Example:
 $PORTB = 0xFF$;
 $PORTB = 0x00$;

b) When pin is Input:

$PORTx$ controls the internal pull up register.

→ $PORTx \text{ bit} = 1$; pull up register ENABLED.

→ $PORTx \text{ bit} = 0$; pull up register DISABLED.

Example: $DDRC = 0x00$;
 $PORTC = 0xFF$;

3) $PSINx$: Input Pins Register

$pinx$ is used to read the actual logic level present on the pin.

→ $PSINx \text{ bit} = 1$; pin is high

→ $PSINx \text{ bit} = 0$; pin is low

Example:

```
if (PSIND & (1 << PDO))
```

```
{
```

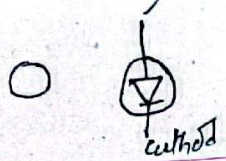
```
    // PDO is high
```

```
}
```

```
else {
```

```
    // PDO is low
```

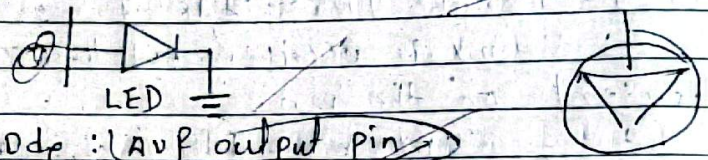
```
}
```

Q16] Explain interfacing of LEDs and Switches with AVR microcontroller using proper diagram and embedded C code.

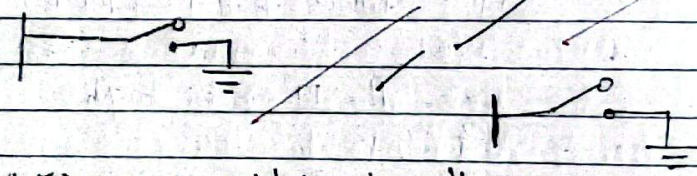
⇒ Interfacing of LEDs and Switches with AVR microcontroller.

a) Interfacing LED with AVR microcontroller.



- LED anode : AVR output pin
- LED cathode : ground

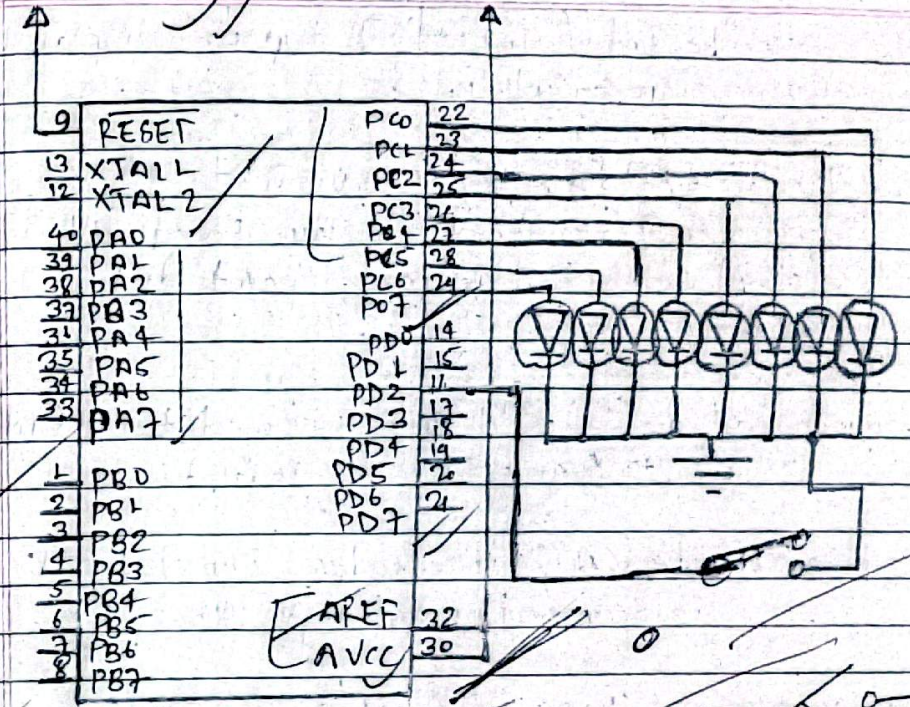
b) Interfacing Switch with AVR microcontroller.



c) Interfacing LED and switch together

- When switch is pressed : LED ON
- When switch released : LED OFF.

Now, Diagram with AVR microcontroller.



ATMEGA 32

```
Code: #include <avr/io.h>
int main(void) {
    DDRC |= 0xFF;
    PORTC ^= ~0xFF;
    DDRD |= (1 << PD2);
    PORTD |= (1 << PD2);
    while(1) {
        if (! (PIND & (1 << PD2))) {
            PORTC |= 0xFF;
        } else {
            PORTC ^= ~0xFF;
        }
    }
}
```


Q.17] Describe Interfacing of 7 segment display with AVR microcontroller.

⇒ 7-Segment Display Overview:-
A 7-segment display consists of 7 LED (a-g) and one decimal point (dp).

Type:-

1. Common Cathode (CC): All cathodes connected to GND.
→ Segment on: logic 1.
2. Common Anode (CA): All anodes connected to Vcc.
→ Segment on: logic 0.

- * Advantages of using AVR with 7-segment.
- Simple port control
 - Fast response.
 - minimal external components
 - Easy multiplexing for multiple digits.

Code:

```
#define F_CPU 8000000UL  
#include <avr/io.h>  
#define LED_direction DDRA  
#define LED_PORT PORTA
```

```
int main(void){
```

```
LED_direction |= 0xFF;  
LED_PORT = 0xFF;
```

```
char  
array[] = {
```

```
0x60,  
0xF9,  
0xA4,  
0xB0,  
0x99,  
0x92,  
0x82,  
0x80,  
0x90
```

```
while(1){
```

```
int i;  
for(i=0; i<10; i++){  
LED_PORT = array[i];  
delay_ms(400);
```

```
}
```


Q.N. 18] Explain digital input and output techniques with an example program for port manipulation.

⇒ AVR microcontrollers support digital I/O through programmable ports (PORTA, PORTB, PORTC, PORTD).

Each port use three registers

- DD Px : data direction registers
- PORT x : Data register
- PIN x : Input pins register.

a) Digital Output technique

- Configure pin as Output using DD Px
- Write logic high or low using PORTx.

b) Digital Input technique.

- Configure pin as Input
- Enable pull up register
- Read input using PINx.

c) Port Manipulation technique.

(i) Entire port operation:

```
DDRC = 0xFF;
PORTC = 0xAA;
```

(ii) Single bit operation (bit masking)

```
PORTC |= (1 << PC2); // Set PC2
```

```
PORTC &= ~ (1 << PC2); // Clear PC2
```

```
PORTC ^= (1 << PC2); // Toggle PC2.
```

(iii) Read Modify write operation.

```
PORTB = (PORTB & 0xF0) | 0x05;
```

Example program:

```
#include <avr/io.h>
```

```
int main(void) {
```

```
    DDRB |= (1 << PB0);
```

```
    DDRD &= ~ (1 << PD2);
```

```
    PORTD |= (1 << PD2);
```

```
    while (1) {
```

```
        if (1 (PIND & (1 << PD2)))
```

```
            PORTB |= (1 << PB0);
```

```
        else
```

```
            PORTB &= ~ (1 << PB0);
```

```
    }
```

```
}
```


Q.19] Write an Embedded C program to generate a delay using Timer (). Explain your code.

⇒ Embedded C Program:
LED Blinking Using Timer ()

```
#include <avr/io.h>
void timer0_delay(void);
int main(void)
{
    DDPB |= (1 << PBD);
    while(1)
    {
        PORTB ^= (1 << PBD);
        timer0_delay();
    }
}

void timer0_delay(void) {
    unsigned int i;
    for(i=0; i < 488; i++) {
        TCNT0 = 0x00;
        TCCR0 = (1 << CS0F) | (1 << CS00);
        while(!(TIFR & (1 << TOV0)));
        TIFR |= (1 << TOV0);
        TCCR0 = 0x00;
    }
}
```

Explanation:-

- DDRB sets PBD as output.
- LED toggles every delay.
- TCNT0 starts from 0.
- Timer counts up to 255.
- Overflow sets TOV0 flag.
- program waits until overflow.
- Overflow repeated 488 times ≈ 1 second.
- Timer stopped after delay.

1) a) Define embedded system? Justify, how ATM machine is a good example of embedded system.

⇒ An embedded system is a microprocessor or microcontroller-based system designed to perform a specific dedicated function within a larger system. It consists of both hardware and software, optimized for real time operations, reliability and efficiency.

Q₂

ATM as an Embedded system.

An ATM (Automated Teller machine) is good example of an embedded system because:

- It is designed to perform specific banking operation only.
- It uses embedded components like CPU, keypad, card reader, display, printer and cash dispenser, etc.
- It runs dedicated software for secure transactions.
- It operates in real time, responding immediately to user inputs.
- It provides high reliability and security.

→ users cannot install or modify applications.

So, ATM satisfies all the characteristics of an embedded system.

(5)

Embedded System	General purpose system
→ Designed for a specific task	→ Designed for multiple task.
→ Uses dedicated software	→ Uses general purpose software
→ Works in real time	→ Usually not real time
→ Limited memory and CPU	→ High memory and CPU
→ Low power consumption.	→ High power consumption.
→ Minimal user interface.	→ Rich user interface.
→ High reliability.	→ Less reliable
→ Ex. ATM, Microwave, etc.	→ Ex. laptop, PC, etc.

done unit-3	2024-21	G.m.p.	Date Page No.
3a] FreeRTOS	VxWorks		
→ Open Source RTOS	→ Commercial RTOS		
→ lightweight and small memory footprint	→ large and feature rich		
→ Use mainly in small embedded systems	→ used in mission critical systems		
→ limited middleware support	→ Extensive middleware support		
→ No built in file system	→ Supports advanced file systems		
→ less secure	→ highly secure		
→ Simple scheduler	→ Advanced priority based scheduler		
→ Ex. IoT devices, etc	→ Example: Aerospace, defense, etc		

2024-21	G.m.p.	unit-3	Date Page No.
3.b] General purpose OS	RTOS		
→ Designed for user convenience	→ Designed for time critical tasks		
→ No strict timing constraints	→ strict timing constraints		
→ Response time not guaranteed	→ Deterministic response		
→ used in PCs and laptops	→ used in embedded system		
→ Ex. windows, linux	→ Ex. FreeRTOS, VxWorks		
→ high throughput	→ predictable behavior		
→ Task deadlines may be missed	→ Deadlines must be met		

4(a) AVR Interrupt Handling:

→ AVR microcontrollers support interrupt driven execution to handle asynchronous events. When an interrupt occurs:-

→ The current instruction is completed.

→ The program jumps to the Interrupt Service Routine (ISR).

→ The return address is stored on the stack.

→ After execution, control returns to the main program using RETI.

AVR ~~improve~~ interrupts improve response time and efficiency.

or

4(b) Task scheduling / Context Switching:

→ The process of deciding which task should run next.

→ performed by the scheduler.

→ Based on priority, time slice or Algorithm.

→ Faster operation.

→ logical decision making process.

→ Determines task order.

→ The process of saving and restoring task state.

→ performed by the kernel/os.

→ Based on switching CPU from one task to another.

→ Take more time (overhead).

→ physical execution process.

→ Enables task change.

done

unit-3

* features of FIOS:-

- It is perform deterministic behavior.
- It support multitasking.
- It has fast context switching.
- It follow synchronization mechanisms.
- Interrupt handling.

Assignment-3

Unit-1 Introduction to Embedded System:

1. What is an embedded system. Explain its main characteristics with suitable example.

⇒ An embedded system is a microprocessor or microcontroller-based system designed to perform a specific dedicated function within a larger system.

It consists of both hardware and software optimized for real time operation, reliability and efficiency.

→ Main characteristics of an Embedded system.

i.) Single function:

An embedded system is designed to perform one specific task repeatedly. It does not act as a general purpose computer.

Example: A Washing Machine.

ii.) Tightly Constrained:

Embedded system operate under strict constraints on

→ processing power

→ memory

→ Energy Consumption

→ size

→ Cost

They must be efficient to work within these limitations.

Example: A digital thermometer.

iii.) Reactive:

An embedded system continually monitors its environments and responds to input from sensors or user.

Example: A temperature control system.

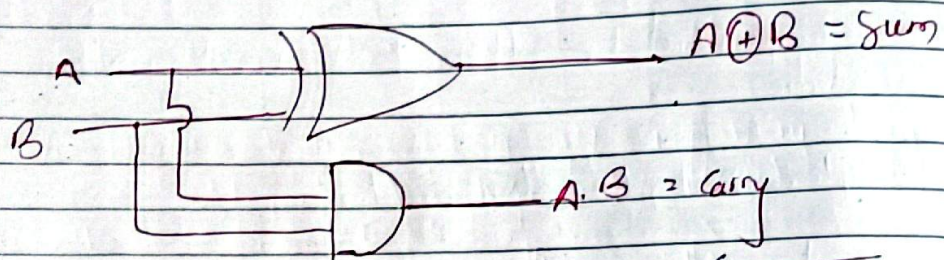
iv.) Real time operation:

Many embedded system must provide output within strict deadlines. If missing, these deadlines can cause system failure.

Example: Airbag deployment system.

Half adder:

Circuit



library IEEE;

USE IEEE.Std_logic_1164.all;

Entity half_adder IS

PORT (A, B : in Std_logic;

Sum, Carry : out Std_logic

);

END half_adder;

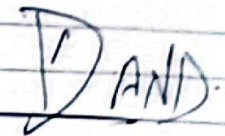
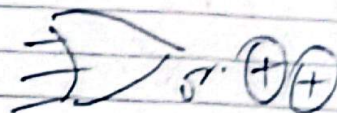
Architecture halfadder arch.
OF halfadder is

Begin

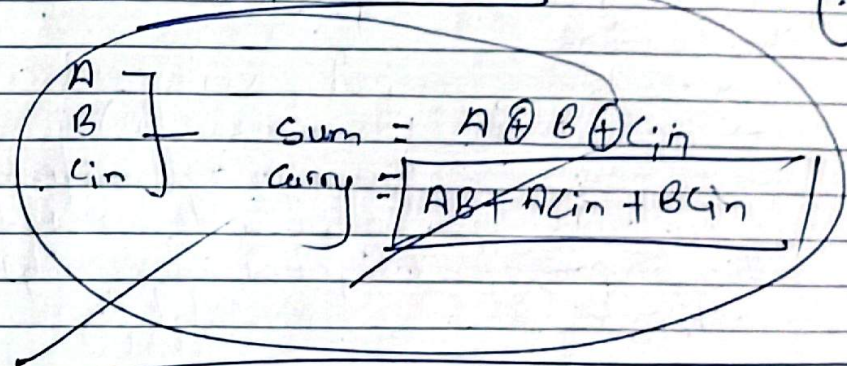
Sum <= A XOR B;

Carry <= A and B;

END halfadder arch;



full adder:



half. Subtractor

Diff. = A ⊕ B

Borrow = $\bar{A} \cdot B$

full

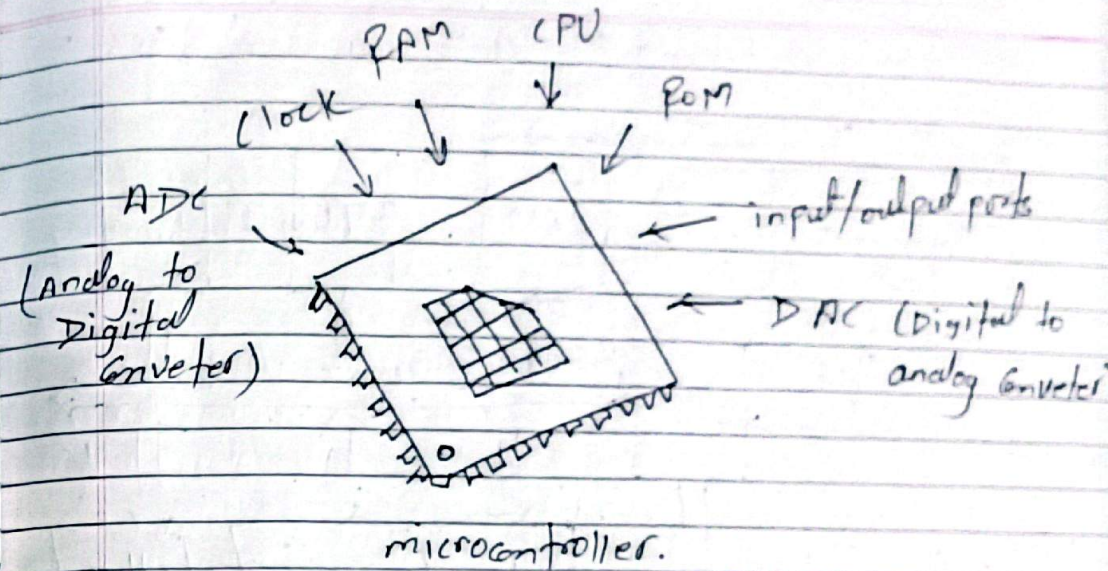
Diff

B = $\bar{A}B + A'Cin + BCin$

done / Unit-2
~~Aut Architecture:-~~

Microcontroller:

- A microcontroller is a VLSI (Very large Scale integration) integrated Circuit (IC) that contains
 - electronics computing units
 - logic unit (known as CPU)
 - memory (Data memory and program memory)
 - I/O ports (input/output ports) and few other components integrated in a single chip.
- Sometimes microcontroller is called computer on a chip or single chip computer.
- It is also called embedded controller. Since it control the embedded devices.



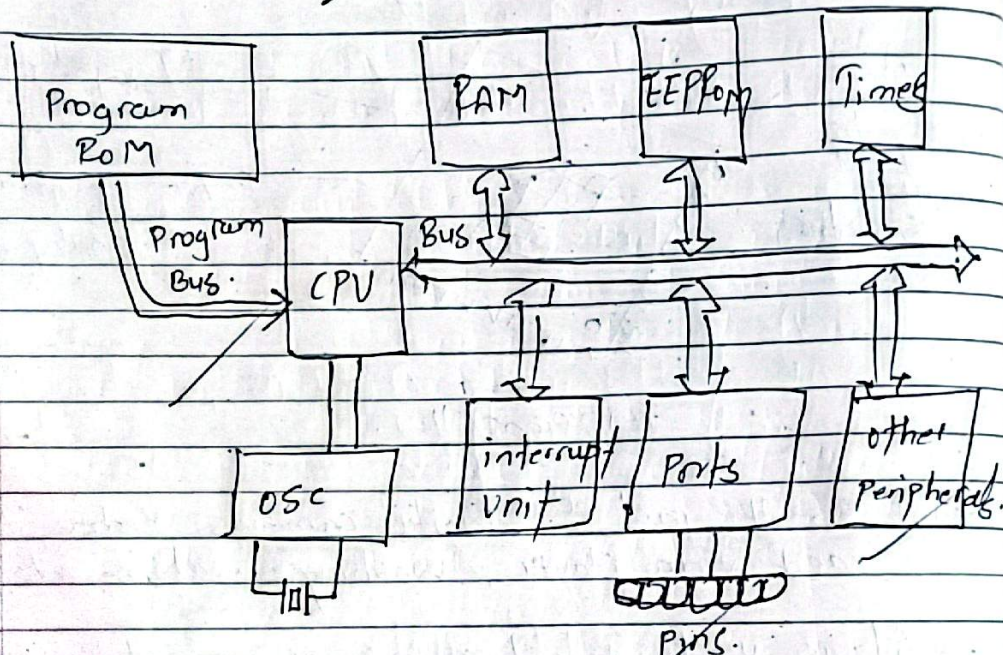
- It is the part of Digital cameras, video, CD and video player, washing machine etc.
- It is used in test and measurement equipment like multimeter, function generators etc.
- It also used in desktop computers like printers, routers, keyboard etc.
- The most common microcontrollers are 8051, AVR and PIC.

S.m.f

done

Date: _____
Page No. _____

Architecture of AVR microcontroller:



fig

Pin Diagram:

It is based on advanced RISC architecture.

ATMEGA 32

9	RESET	PC0	22
13	XTAL1	PC1	23
12	XTAL2	PC2	24
40	PA0/AD0	PC3	25
39	PA1	PC4	26
38	PA2	PC5	27
37	PA3	PC6	28
36	PA4	PC7	29
35	PA5		
34	PA6	PD0	14
33	PA7/AD7	PD1	15
		PD2	16
1	PB0	PD3	17
2	PB1	PD4	18
3	PB2	PD5	19
4	PB3	PD6	20
5	PB4	PD7	21
6	PB5		
7	PB6	AREF	32
8	PB7	AVCC	30

Program :- L7 All LEDs ON/off using switch.

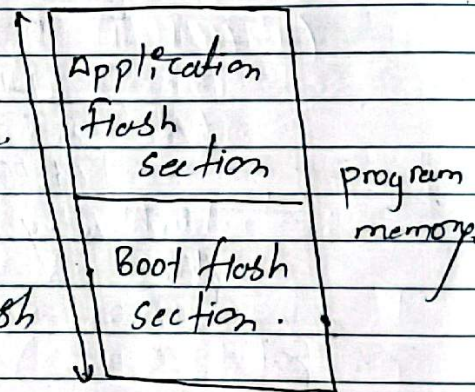
Memory Management:

- The AVR microcontroller's memory is divided into ~~two~~ Program memory and data memory.
- Program memory (PROM) is used for permanent saving the program being executed while Data memory (RAM) is used for temporarily storing and keeping intermediate results and variables.

→ In program memory,

→ it's divided into two types/section.

- Boot flash section
- Application flash section.



→ The AVR executes programs stored in program memory only.

→ In Data memory,

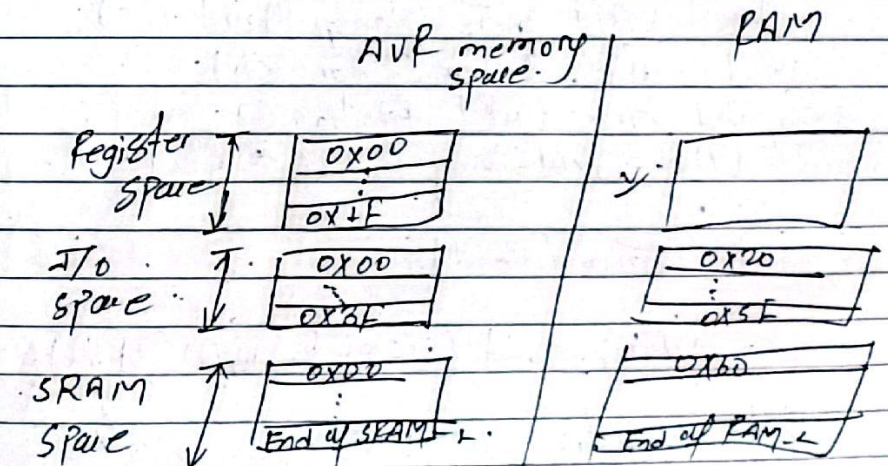
→ It consists of,

- Rx space
- I/O memory
- Extended I/O memory
- Internal SRAM

→ Rx space consists of 32 general purpose working 8-bit registers (R0-R31).

→ it provides shortest (fastest) access time, which allows single cycle Arithmetic logic unit (ALU) operation.

Internal SRAM used for temporarily storing and keeping intermediate result and variable.



Q.7 Write a C program for AVR ATmega 16/32 to initialize UART Communication and perform serial data transmission and reception.

```

=> #define F_CPU 8000000UL // Define CPU frequency as 8MHz
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include <stdio.h> // define desired baud rate.

```

```

#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU/USART_BAUDRATE * 16UL)) - 1)

```

// Function Prototype :-

```

void UART_init(long baudrate);
void UART_Txchar(char ch);
unsigned char UART_Rxchar();
void UART_SendString(char *str);

```

```
int main(void) {
```

```
    UART_init(USART_BAUDRATE);
```

```
    // initialize the UART Communication.
```

```
// send a starting message to the PC.
```

```
UART_SendString("System is ready. Echo Mode active....\r\n");
```

```
while(1) {
```

```
    // receive the character from PC
```

```
    char data = UART_Rxchar();
```

```
    UART_Txchar(data);
    return 0;
```

```
}
```

```
void UART_init(long baudrate) {
```

```
    UCSRB |= (1 << RXEN) | (1 << UCSZ1)
```

```
    UBRRF = BAUD_PRESCALE;
```

```
    UBRRH = (BAUD_PRESCALE >> 8);
```

```
}
```


unsigned char UART_Pxchar() {
 while ((UCSRA & (1 << PXC)) == 0);
 return(UDR);
 }

void UART_Txchar(char ch) {
 while ((UCSRA & (1 << UDRE)) == 0);
 UDR = ch;
 }

void UART_sendString(char *str) {
 unsigned char j = 0;
 while (str[j] != '\0') {
 UART_Txchar(str[j]);
 j++;
 }

DDRCB1 = (1 << DDB2);
 TCCR0B = 1 << WGM02;
 TCCR0B1 = (1 << COM0A0);
 OCR0A = OCR0A_VALUE;

2.b

#define F_CPU 16000000UL
 #include <avr/io.h>
 #include <avr/interrupt.h>

#define OCR0A_VALUE 124
 #define OCR1A_VALUE 999

void ~~Setup~~ Setup_Timer0(void) {

DDRCB1 = (1 << DDB2);
 TCCR0A1 = (1 << WGM01);
 TCCR0A1 = (1 << COM0A0);
 OCR0A = OCR0A_VALUE;

TCCR0B1 = (1 << CS02);

}

void Setup_Timer1(void) {

DDRCB1 = (1 << DDB4);
 TCCR1B1 = (1 << WGM12);
 TCCR1A1 = (1 << COM1A0);
 OCR1A = OCR1A_VALUE;

TCCR1B1 = (1 << CS11);

}


```
int main (void) {
```

```
    Setup_Timer_D();  
    Setup_Timer_K();
```

```
    while(1) {
```

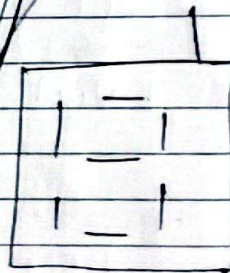
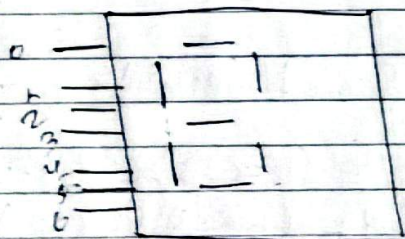
```
    }
```

```
}
```

Common - cathode.

7-Segment display.

Anode



2024-1F

6.a]

```
#include <avr/io.h>
```

```
#include <util/delay.h>
```

```
unsigned char Segment_codes[] = {
```

```
    0x3F, // 0
```

```
    0x06, // 1
```

```
    0x5B, // 2
```

```
    0x4F, // 3
```

```
    0x66, // 4
```

```
    0x6D, // 5
```

```
    0x7D, // 6
```

```
    0x07, // 7
```

```
    0x7F, // 8
```

```
    0x6F // 9
```

```
};
```

```
int main (void) {
```

```
    DDLED = 0xFF;
```

```
    while(1) {
```

```
        for (int i=0; i<10; i++)
```

```
    }
```


PORTD = segment Codes[i];
delay_ms(2);

}
}
}

$$\text{Duty Cycle(\%)} = \frac{\text{on Time}}{\text{total period}} \times 100\%$$

Unit - VII

Role of Embedded System in IoT, (Internet of things).

- An Embedded System is a combination of both software and hardware designed to perform a specific task. In IoT (Internet of things), embedded system act as brain of smart devices.

Main roles:-

- Data Collection:
 - Reads the data from sensors.
- Data processing:
 - The processes the data using a microcontrollers. (like Arduino, ESP32 etc).
- Decision making:
 - Take action based on programmed logic.
- Communication:-
 - Sends and receives data over the internet.
- Control:- Controls actuators like motors LEDs etc.

Integration of Sensors, Actuators & Communication modules.

1. → Sensors (Input devices):

- Convert the physical parameters into electrical signals.

→ Example:

- temperature sensor
- humidity sensor
- motion sensor.

→ Working: Sensors → Sends signal → Embedded system.

2. → Actuator (Output devices):

- Perform the physical actions based on command.
- Example:
 - motors, LEDs, Buzzers etc.

Working: Embedded system → Sends signal → Actuator perform action.

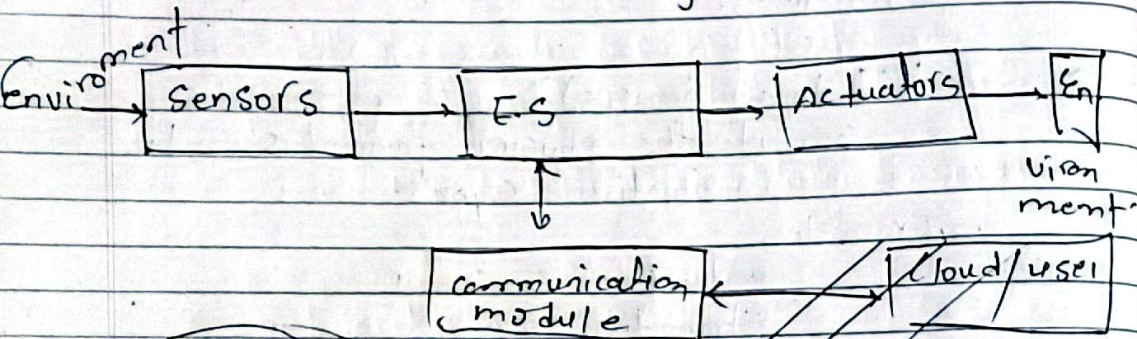
3. → Communication modules:

- Enable device to connect with other devices or cloud.

Example: wifi, Bluetooth, ZigBee etc.

→ Working: ES → Sends data → Internet/cloud/mobile app.

Simple/Basic Diagram:-



2024-1f
6.6]

MQTT (message Queuing Telemetry Transport)

→ MQTT (message Queuing Telemetry Transport) is a lightweight communication protocol used in IoT to send data between devices over the internet.

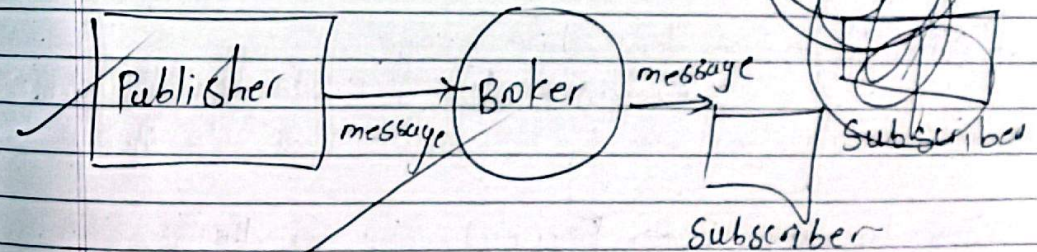
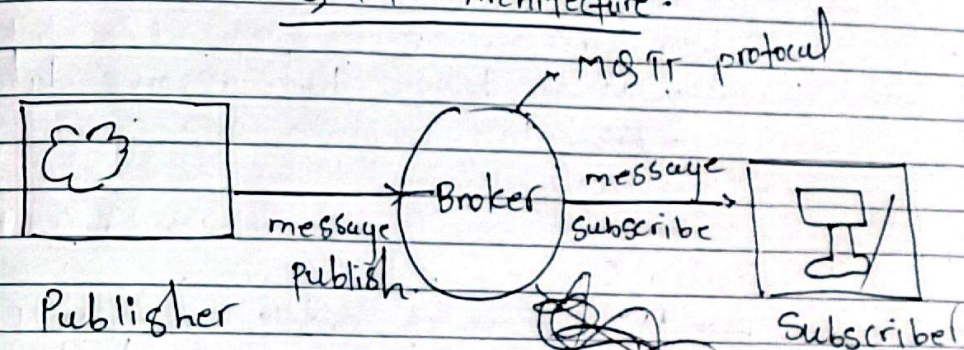
→ It is defined for:

- low bandwidth networks
- low power devices.
- unreliable connection.

→ It's features:

- It is lightweight
- fast communication.
- Works on publish/subscribe model
- requires less power and memory.

MQTT Architecture:



→ It's has three main Components:

1. Publisher:

→ It is devices that sends data.

→ Example: temperature sensors.

2. Subscriber:

→ It is devices that receive data.

→ Example: mobile App

3. Broker:

→ It acts as a middleman/server.

→ It receives data from publisher and sends it's to subscriber.

→ advantage: → low data usage.

→ Efficient for IoT devices.

→ Disadvantage: → Depends on Broker.

→ less secure without encryption.

1. ~~X~~ Arduino:

→ A Microcontroller based development board.

→ It is used for simple embedded and IoT projects.

~~It has~~

→ It's features:-

→ It is easy to use and beginner friendly.

→ It has Digital and analog input/output pins.

→ It has low cost.

→ It requires external modules for internet (Wifi, Bluetooth).

→ Uses: It is used in

→ sensor based projects.

→ It is used in LED control, motor control etc.

→ It is used in basic automation systems.

→ Example: smart temperature monitoring system.

2. ESP32 :

- It is a powerful microcontroller with built in wifi and bluetooth.
- It's key features:-
 - Inbuilt wifi and bluetooth.
 - Faster than arduino.
 - Low Power Consumption.
- Uses:-
 - Smart home systems.
 - It is used in wireless sensors networks.
 - It is used in IoT devices with cloud connectivity.
- Example: Home automation using mobile App.

Hedged.

3. Raspberry Pi :-

- It is a full fledged mini computer.
- It is also called single board computer.
- it's key features:-
 - It runs operating system (Linux).
 - It is high processing power.
 - Supports keyboard, mouse, display easily.
- Uses:-
 - It is used in advanced IoT application.
 - It is used in AI and IoT projects.
- Example: Smart Surveillance system with cameras.

2024-1-1

6.6

i)

~~Simple~~ organize data using hierarchical topic structure to ensure scalability and easy filtering by the cloud.

→ Topic Schema: Use a format like parking/[location]/[lot_id]/[slot_id] / status.

→ Payload: Send lightweight JSON objects containing status (occupied/vacant) and timestamp.

ii)

Use built in MQTT features to prevent data loss when the server is offline.

→ QoS Level 1 or 2: Ensures messages are delivered "at least one" or "exactly one", handling re-transmission automatically.

→ Retained Messages: The broker stores the last known status of each slot, providing immediate updates to new subscribers.

Date
Page No

→ Persistent Sessions: Setting `cleanSession = false` allows the broker to store messages for clients while they are disconnected.

iii)

The system follows a simple loop for detection, processing and communication.

→ Sensing: Ultrasonic or IR sensors detect a vehicle by measuring distance.

→ Processing: The Arduino Uno reads the sensor data and compares it against a threshold distance to determine occupancy.

→ Action: It updates a local LCD screen and controls servo motors to operate entry/exit gates.

→ Cloud Sync: A wifi module publishes the status to the cloud via MQTT.

iv)

→ MQTT: ideal for real time, low-bandwidth status updates.

Date
Page No

→ HTTP/HTTPS : Standard for web-based interaction and RESTful APIs.

→ LoRaWAN : Best for long range, low power communication in large outdoor lots.

V.

The system be enhanced for real time monitoring using :-

→ Heartbeat Signal :

→ Mobile App Integration.

→ Image processing.

→ ^{Data} Analytics.

Program :-

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

#define LCD_Data_Port PORTD
    " " Dir DDLD
    " " Port PORTC
    " " Dir DDPC
```

```
#define RS PC0
#define RW PL1
#define EN PC2
```

```
void LCD_Command(unsigned char cmd){
```

```
    LCD_Data_Port = cmd;
    LCD_Ctrl_Port &= ~(1 << RS);
    " " " &= ~(1 << RW);
    " " " &= (1 << EN);
    _delay_us(1);
```

```
    LCD_Ctrl_Port &= ~(1 << EN);
    _delay_ms(3);
```

```
}
```

```
void LCD_Char(unsigned char data){
```

```
    LCD_Data_Port = data;
    LCD_Ctrl_Port |= (1 << RS);
    LCD_Ctrl_Port &= ~(1 << RW);
    " " " |= (1 << EN);
```

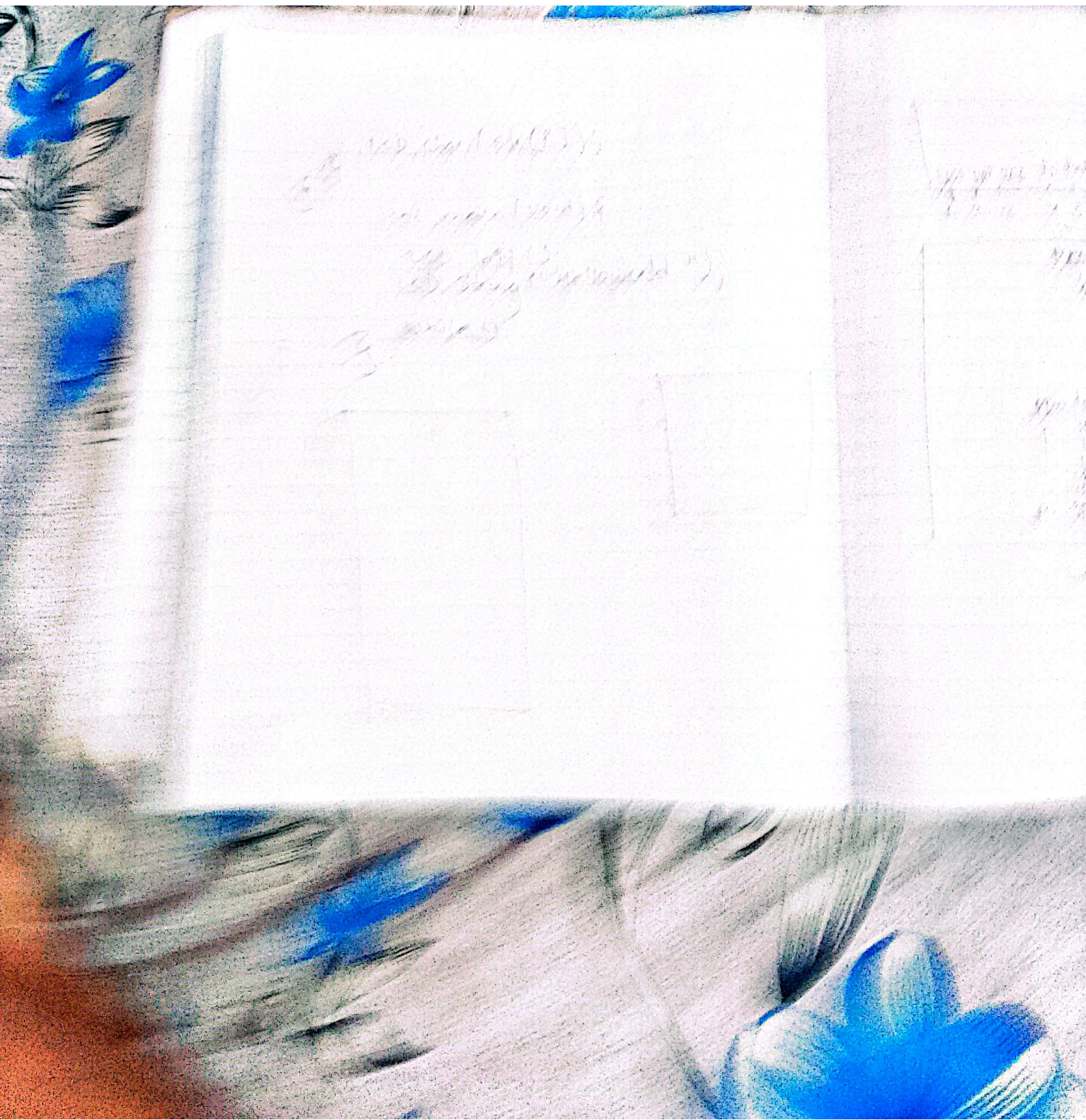
```
    _delay_us(1);
    LCD_Ctrl_Port &= ~(1 << EN);
    _delay_ms(1);
```

```
void LCD_init(void){
```

```
    LCD_Data_Dir = 0xFF;
    LCD_Ctrl_Dir |= (1 << RS) | (1 << RW) | (1 << EN);
    _delay_ms(20);
```

```
    LCD_Command(0x38);
    LCD " (0x01);
    " " (0x06);
    LCD " (0x01);
    _delay_ms(2);
}
```

```
void LCD_String(char *str){
    int i;
    for(i=0; str[i] != 0; i++){
```

PWM Controlling :-

```
#include <avr/io.h>
#include <util/delay.h>

void InitPWM() {
    TCCR0 = (1 << WGM00) | (1 << WGM01) |
             (1 << CS00);

    DDRB |= (1 << PB3);
}

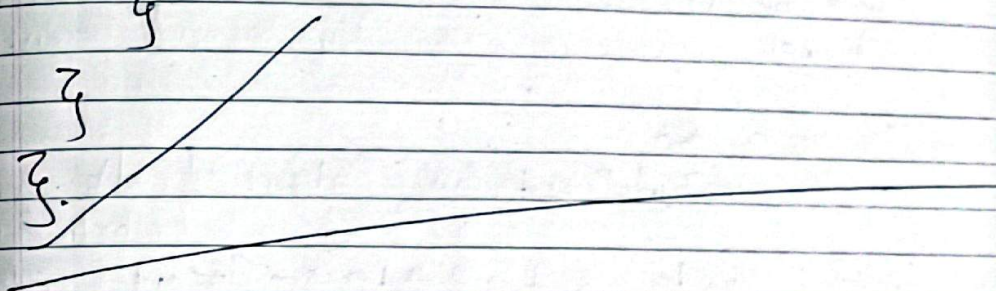
void SetPWMOutput(uint8_t duty) {
    OCR0 = duty;
}

int main(void) {
    uint8_t brightness = 0;

    InitPWM();

    while(1) {
        for (brightness = 0; brightness < 255;
             brightness++) {
            SetPWMOutput(brightness);
            delay_ms(20);
        }
    }
}
```

```
}
for (brightness = 255; brightness > 0; brightness--)
{
    SetPWMOutput(brightness);
    delay_ms(20);
}
}
}
}
```



* Serial Communication.

Unit - 5:

- Application
- peripheral interfacing
 - chip to chip communication
 - Industrial control

Serial communication is a method of transferring data one bit at a time over a communication channel.

→ It works as:
Data is sent in a stream of bits.

like: Sender → 1 → 0 → 1 → 0 → Receiver.

→ It uses.

TX (transmit): sends data
RX (receive) → receive data

→ Types: UART, I²C, SPI etc.

→ Advantages: → It is simple wiring.
→ low cost
→ less noise interference.

→ Disadvantages: → slower than parallel communication
→ it bit by bit transfer.

- Application
- In use, IoT
 - Personal area network
 - Industrial Automation
 - Automotive.

* Wireless Communication:

→ Wireless communication transfers data without physical wires, using electromagnetic waves like radio signals.

→ It works as:

- Data is converted into signal
- Transmitted by antenna.
- Received and decoded by another devices.

devices A → 11111 → Device B:

→ Types: Wi-Fi, Bluetooth, ZigBee, LoRa etc.

→ Advantage: → It has no wiring needed.
→ It is flexible and scalable.

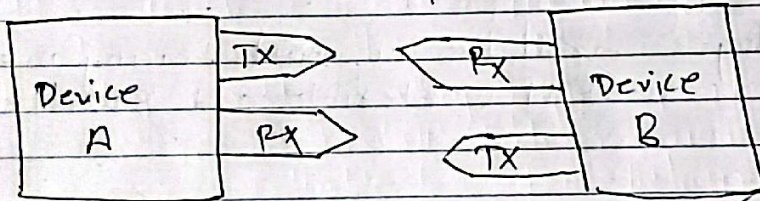
→ Disadvantages: → Security risks.

- Power Consumption
- limited range.

* UART:

- It is types of Serial Communication protocol used in embedded system to transfer data one bit at a time without a clock signal.
- It stands for Universal Asynchronous Receiver Transmitter.
- It is one of the simplest and most widely used communication methods between devices like microcontrollers, sensors and computers.

UART



Where, Tx (transmit): Data Sends.
 Rx (Receiver): Receives Data.

- Error detection is possible.
- Advantage: → It is simple to use.
- It has only two wires needed.
- It is widely support.
- It is not clock line required.

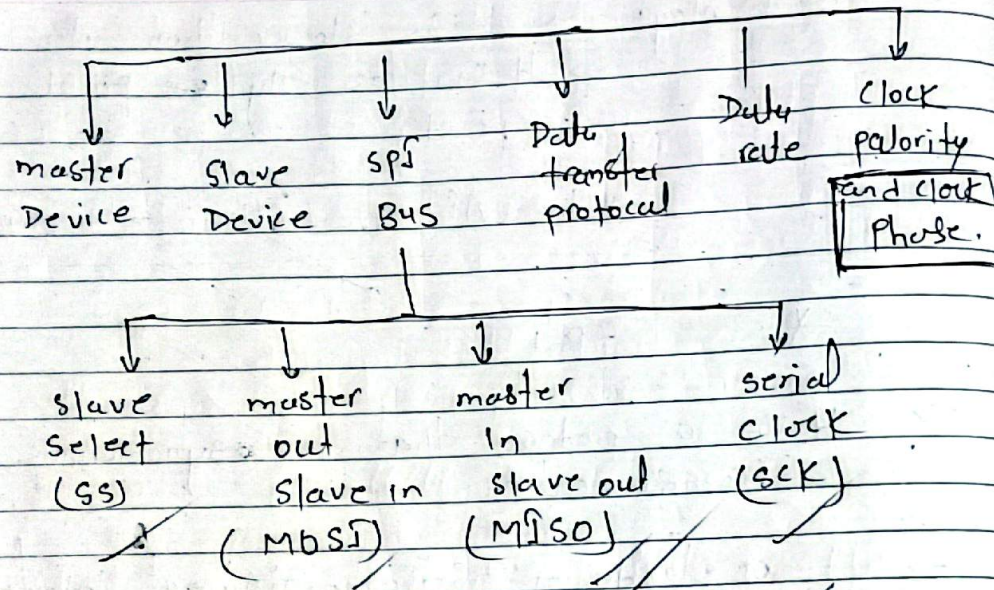
- Disadvantage: → It is slower than SPI
- It work as only one point to point

→ In UART, no built in addressing system

* SPI

- It is a protocol that is asynchronous serial communication.
- It stands for Serial peripheral Interface.
- It is used to communicate between the peripheral devices i.e input and output devices and microcontrollers.
- It is allowed to transfer high speed data.
- It can transfer the data and receive data from one device to and another device at a time.

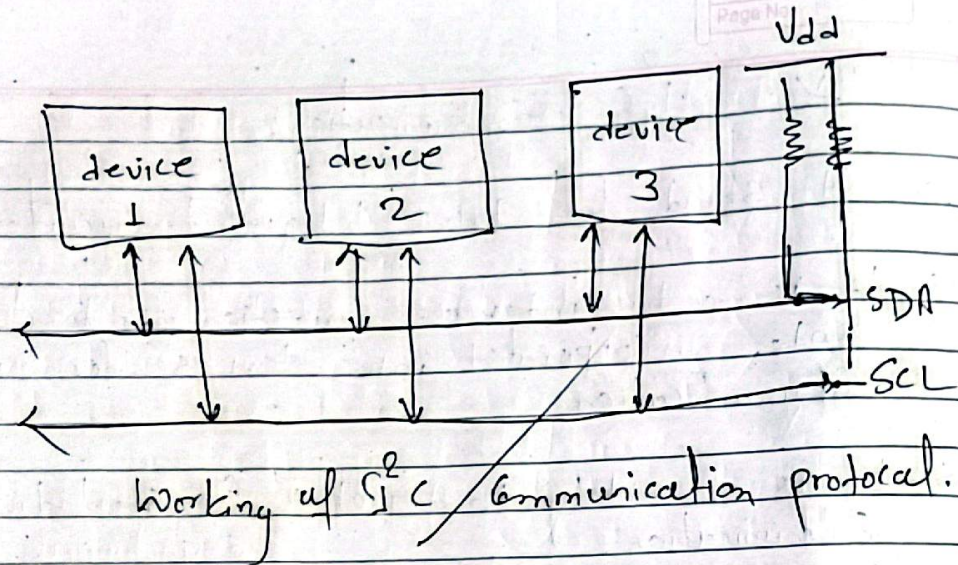
→ Component of SPI



- It is used for IoT application.
- Advantage:
 - It is very high speed.
 - It is full duplex communication.
 - Reliable for short distances.
- Disadvantage:
 - It requires more wires.
 - Not suitable for long distances.

X. I²C Communication:

- It stands for Inter-Integrated Circuit.
- It is a bus interface connection protocol incorporated into devices for serial communication.
- It is a widely used protocol for short distance communication.
- It is also called Two wire interface (TWI).
- It need two wire i.e.
 - SDA (Serial Data line): carries data.
 - SCL (Serial Clock line): carries clock signal.
- It is half duplex communication protocol.
- Advantage:
 - It need only two wire.
 - It supports multiple devices.
 - Saves microcontroller pins.
- Disadvantage:
 - It is slower than SPI.
 - It is more complex than UART.
 - It has limited distance.



Bluetooth:

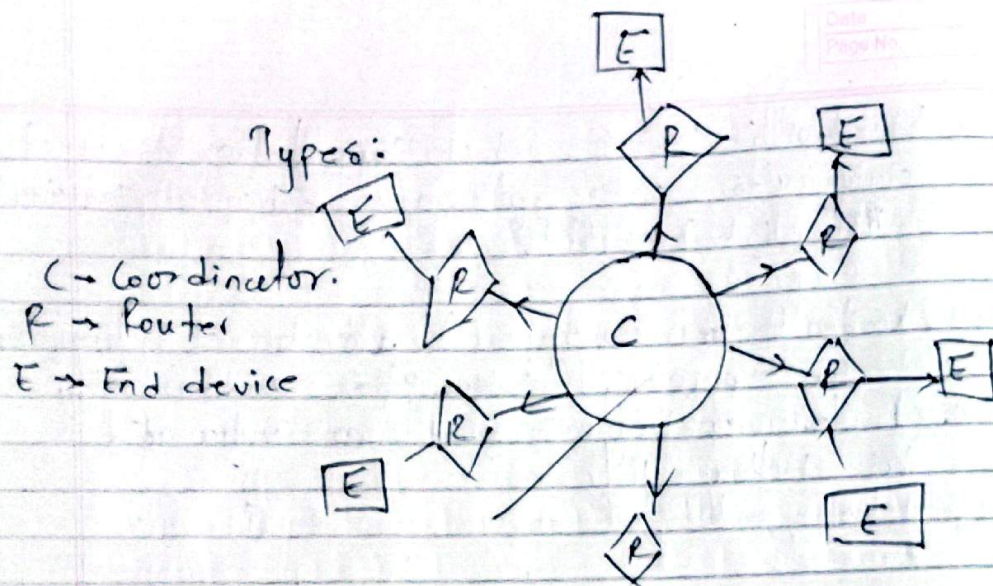
- It is technology allows devices to communicate with each other without cables or wire.
- It is types of wireless communication.
- It relies on short-range radio frequency, and any device that incorporates the technology can communicate as long as it is within the required distance.
- The two Bluetooth standards currently in use today are:

→ Bluetooth Classic, which support two distinct data rates, Basic rate and Enhanced Data rate (BR/EDR)

→ Bluetooth low energy (LE), which is optimized for low power consumption and mainly used for application that are constrained by battery life.

ZigBee

- ZigBee is an open, global, packet based protocol designed to provide an easy to use architecture for secure, reliable, low power wireless networks.
- flow or process control equipment can be placed anywhere and still communicate with rest of the system.
- It is also be moved, since the network doesn't care about the physical location of a sensor, pump or valve.



1. ZigBee Coordinator Device: It communicates with routers.

2. ZigBee Router: It is used for passing the data between devices.

3. ZigBee End Device: It is device that is going to be controlled.

→ Features of ZigBee

- Stochastic addressing
- Link management
- Frequency Agility
- Power management

→ Advantage:

- Designed for low power consumption.
- Easy to implementation
- low cost.
- mesh networking.

Disadvantage: → It has limited range
→ It has limited data rate
→ Some security problems.

Application:

- Home automation
- Medical Data collection.
- Light Control system.
- Commercial use.
- Home Networking etc.

LoRa :-

- LoRa (long range) wireless technology is a low power wide area network technology for communication for remote devices.
- It is designed to achieve wide-area coverage and low power wireless communication especially for Internet of things (IoT) applications.
- LoRa uses a technology called Spread Spectrum modulation.
- It frequency Band in 868 MHz to 915 MHz in general.
- It data rate usually between 0.3 Kbps and 27 kbps.
- LoRa networks usually adopt a star topology.
- Advantages:
 - long range coverage.
 - low power consumption.
 - high penetration.
 - low cost.

- Disadvantage :
 - low data rate.
 - some security issues.
 - limited network capacity.

→ Application:

- In IoT, environmental monitoring system.
- Smart agriculture.
- Smart cities.

GSM

- It stands for Global System for mobile communication.
- It is 2G
- It's network type is circuit-switched.
- It's main purpose voice communication.
- It's frequency band is 900 MHz and 1800 MHz
- Billing Basis of Duration of call
- Internet access is not supported
- It's popular services is SMS
- It has single time slot per user.

GPRS

- General Packet Radio service.
- It is 2.5G (Generation)
- It's network type is packet switched.
- It main purpose. Data communication.
- It's frequency band is 850, 900, 1800 and 1900 MHz
- Billing Basis of Amount of data transferred.
- Internet access is supported.
- MMS
- It has multiple time slot per user possible.

Unit-3

Task scheduling:

- The process of deciding which task should run next is called task scheduling.
- It is logical decision making process.
- It determines the task order.
- It performs fast operation.
- It is performed operation by scheduler.

Types of task scheduling Algorithm in RTOS.

(A) Rate monotonic scheduling (RMS):

- It is static priority algorithm.
- In shortest period give high priority.
- It works for periodic tasks.
- It is preemptive.

Advantage: → It is simple and deterministic
→ widely used in RTOS.

Limitations/

Disadvantage: → only for periodic tasks.
→ priority depends on task.

B. Earliest Deadline first (EDF):

- It is a dynamic priority algorithm.
- optimal for both periodic and aperiodic tasks.
- CPU can fully utilized up to 100%.

Advantage: → Better utilization than RMS.

Disadvantage: → It is more complex to implement.

C. Priority Based preemptive scheduling:

- Each task is assigned a fixed priority.

→ A higher priority task always preempts a lower priority task.

→ Advantage: It is very fast context switching.

→ Disadvantage: - It follows priority inversion.

Context Switching:

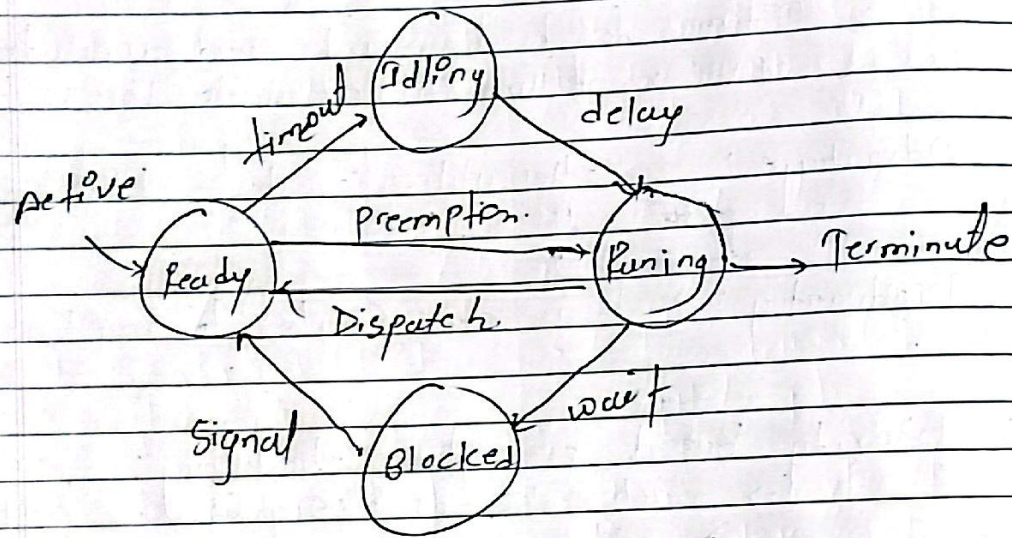


fig: task states in context switching.

Advantages :-

- It is efficient for multitasking.
- Better CPU utilization.

Disadvantage :-

- It causes overhead.
- It reduces performance.

Task Synchronization

→ It is a technique used to coordinate the execution of multiple tasks so that they work together without conflicts especially when sharing common resources.

→ It helps in

- Data Corruption.
- Race Conditions.
- Deadlocks.
- priority inversion.

→ Common Synchronization mechanism:-

1. Semaphores: → Binary semaphores
→ Counting semaphores.

2. Mutex (mutual Exclusion).

3. Message Queues.

4. Event flags / Event Groups.

5. Monitors.

Multi tasking

→ The process of ^{manages} multiple tasks in CPU is called

→ Each task has its own memory.

→ It has higher (task switching)

→ It is a independent function.

→ Strong isolation betⁿ tasks

→ It is slower as a compared to thread

Multi threading :-

→ The process of multiple threads inside a single task is called.

→ Threads share memory

→ It has lower (thread switching).

→ It is a parts of the same complex function.

→ It is less isolation.

→ It is faster.

Deadlock :-

Deadlock is a state in a OS where two or ~~one~~ more processes are stuck forever because each is waiting for a resource held by another.

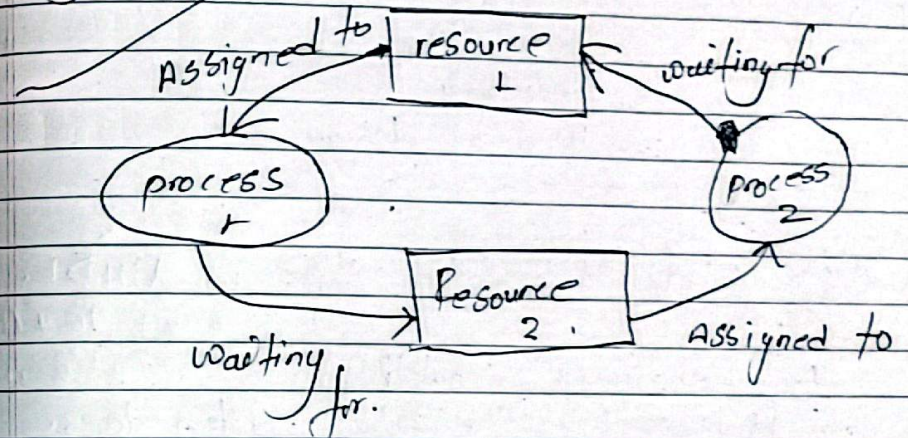


fig: Condition for deadlock.

→ Deadlock can be handled using prevention, avoidance (Banker's Algorithm) or, detection and recovery.

done

Unit - 4

VHDL

→ VHDL is a hardware description language used to design digital circuits.

→ It is used in:

→ simulation.

→ modeling

→ Hardware design.

4(b)

2. Different modeling style in VHDL.

1. Behavioral Modeling:

→ It describes the what system does.

→ It uses processes and sequential statement variable, if, case, control statement.

→ Example:

Process (A, B)

begin

if (A = '1' and 'B' = '1') then

Y <= '1';

else Y <= '0';

end if;
end process;

2. Data flow modeling:

→ It describes the how data flows using concurrent assignments.

→ It uses operators like and, or, xor, etc.

→ Example:

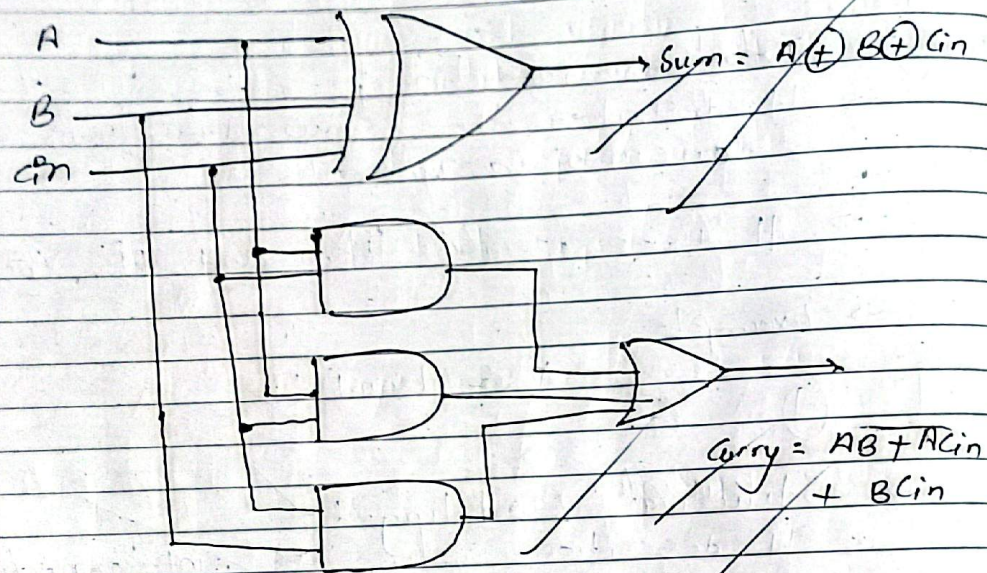
Y <= A and B.

3. Structural Modeling:

→ It describes system as interconnection of components.

→ It is used in circuit diagram.

full adder using structure modeling.



→ inputs are A, B and cin

→ outputs are :-

→ Sum = $A \oplus B \oplus cin$

→ Carry = $AB + ACin + BCin$

Code:

LIBRARY IEEE;
USE IEEE.Std_logic_1164.all;

ENTITY Fulladder IS

PORT (A, B, Cin : in Std_logic;
Sum, Carry : out Std_logic

);

END Fulladder;

ARCHITECTURE adder_arch OF Fulladder IS
BEGIN

$$\text{Sum} \leq A \text{ XOR } B \text{ XOR } cin;$$

$$\text{Carry} \leq (A \text{ and } B) + (A \text{ and } cin) \text{ or } (B \text{ and } cin);$$

END adder_arch;

7.07

A test bench is an VHDL program used to simulate and verify the functionality of a design (like mux, adder etc.).

→ It does not have input/output

→ It generates test signals.

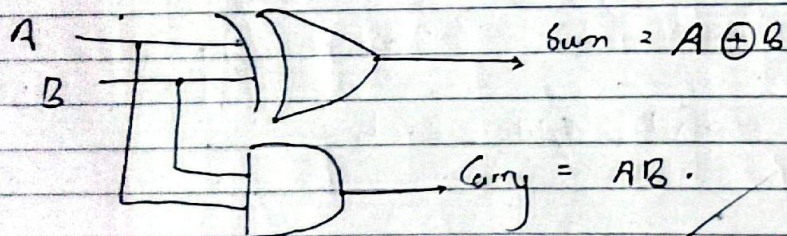
Code and figure in note:

VHDL :-

Half adder:

(4XL)

↓ : 4 demux



Code:

Library ieee;
Use ieee.std_logic_1164.all;

Entity halfadder is

Port

A, B : in std_logic;

Sum, Carry : out std_logic;

);

End halfadder;

Architecture halfadder_arch of halfadder is
Begin

Sum <= A ^{Xor} B

Carry <= A and B;

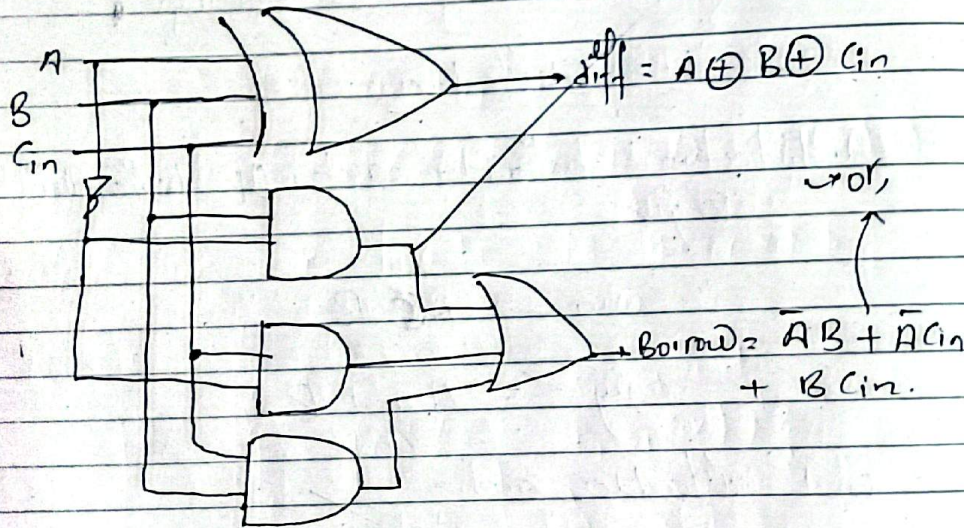
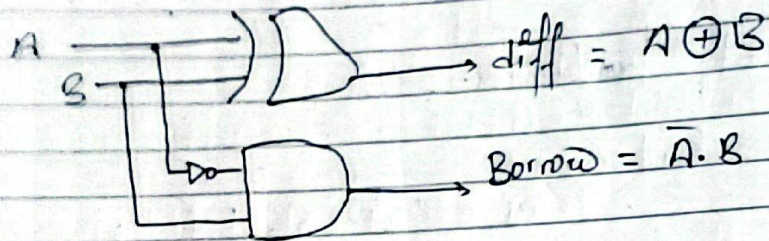
End halfadder_arch;

0 and + 0.

⊕ Xor.

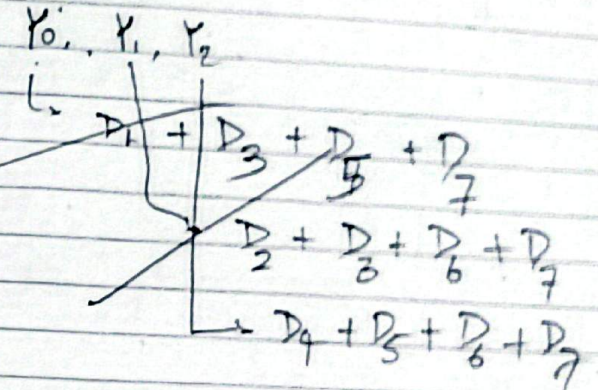
Half Subtractor:

- not.
• And
• XOR.



Octal to binary decoder

input are $D_0, D_1, \dots, D_7$ (0-7)
output are.



3-to-8 decoder

inputs are, A, B and C

output are $Y_0, \dots, Y_7$.

$$Y_0 \rightarrow \bar{A} \cdot \bar{B} \cdot \bar{C} \quad Y_1 = \bar{A} \bar{B} C \quad Y_2 = \bar{A} B \bar{C}$$

$$Y_3 = A \bar{B} \bar{C} \quad Y_4 = A \bar{B} C \quad Y_5 = A B \bar{C}$$

$$Y_7 = A B C$$

truth table.

A	B	C	output
0	0	0	0
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

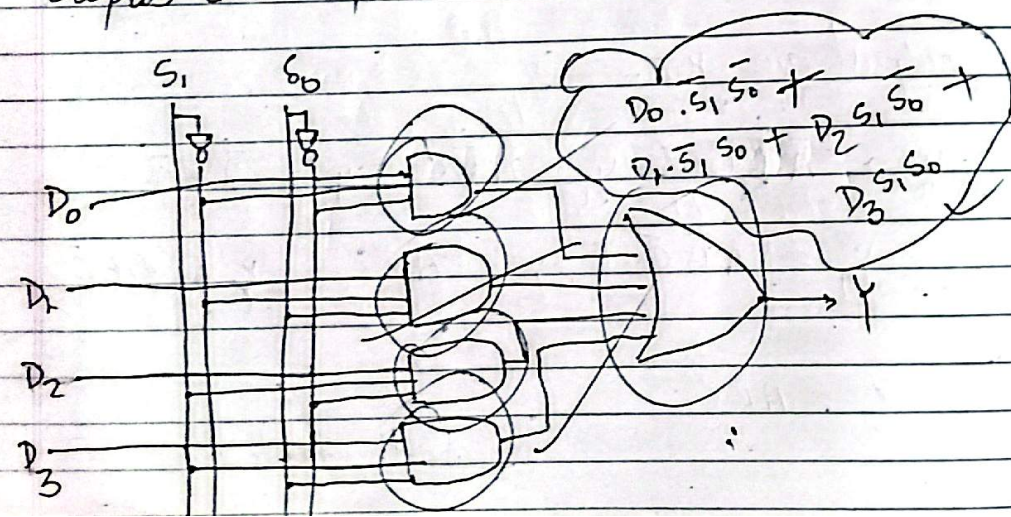
$\rightarrow Y_0 \rightarrow \bar{A}\bar{B}\bar{C}$

$\bar{A}\bar{B}\bar{C}$

4x1 multiplexer

inputs are $S_1, S_0, D_0, D_1, D_2, D_3$

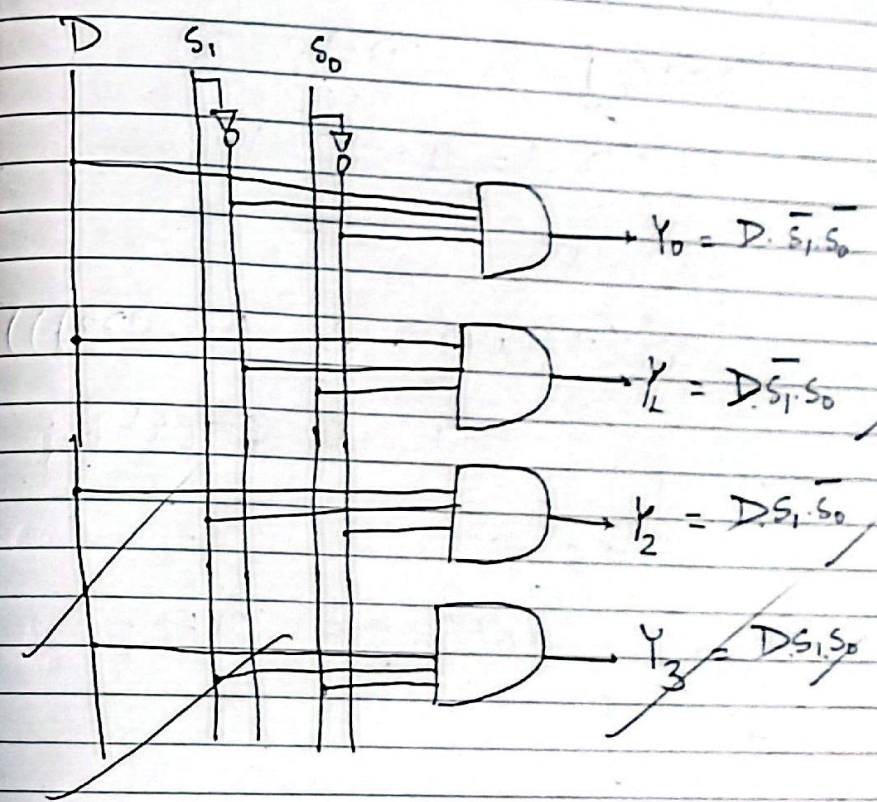
outputs are Y



$$D_0 \cdot \bar{S}_1 \bar{S}_0 + D_1 \cdot \bar{S}_1 S_0 + D_2 \cdot S_1 \bar{S}_0 + D_3 \cdot S_1 S_0$$

1x4 demultiplexer

input are D, S_1, S_0
output are Y_0, Y_1, Y_2, Y_3



$$Y_0 = D \cdot \bar{S}_1 \bar{S}_0$$

$$Y_1 = D \cdot \bar{S}_1 S_0$$

$$Y_2 = D \cdot S_1 \bar{S}_0$$

$$Y_3 = D \cdot S_1 S_0$$