National Textile University, Faisalabad

Department of Computer Science

Name	Nimra Fatima
Reg no	23-NTU-CS-1081
Section	BSCS 5th -B
Subject	Embedded & IoT Systems
Assignment	1
Submitted to	Sir Nasir
Submission date	19-10-25

Assignment#1

Question#1

Why is volatile used for variables shared with ISRs?

We use **volatile** for variables shared with ISR because value changes when an interrupt generates so the compiler does not take value which the ISR gives the value during its interrupt because value has stored in the register for better performance. Therefore, the reason we use volatile is because when an interrupt generates volatile saves the value of **interruption** and volatile helps to reserve the changing value during an **interrupt**

Question#2

Compare hardware-timer ISR debouncing vs. delay()-based debouncing.

Debouncing:

When we press the button, the state(**on/off**) still bounces and it takes time so the factor that we use is debouncing it means that it prevents the system from confusion and error.

Hardware-timer ISR debouncing	delay()-based debouncing.
It is fast and efficient	It takes little bit of time.
it is continuously working	It stops getting the value of interrupt
It gives accurate result	It gives less accurate result
It does not wait because a hardware sets (**ISR Timer**)	It waits because it does not contain any hardware, after finding value then it executes

---

Question#3

What does IRAM_ATTR do, and why is it needed?

The function of **IRAM_ATTR is** that store the interrupt function in **RAM** instead of flash memory. We need this function because the program and its instructions usually stored in the flash memory of the **ESPR32** that causes the execution of program slowly so this **IRAM_ATTR** helps us to store the program in RAM that causes the fast execution.

Question#4

Define LEDC channels, timers, and duty cycle.

LEDC Channels:

A channel is basically an output line from which the PWM signal is generated.
Each channel has its own output pin.
In the ESP32, there are multiple LEDC channels (for example, up to 16).

LEDC Timers:

A **timer** controls the **speed or frequency** of the PWM signal — meaning how fast it turns ON and OFF.
One timer can work with **one or more channels**.

Duty Cycle:

How long the signal stays ON and how long the signal stays OFF that is call ed duty cycle.

duty cycle=timer on/timer on timer off *100

Question#5

Why should you avoid Serial prints or long code paths inside ISRs?

Long code means it takes instructions like loops and instructions and for execution of these instructions it takes a lot of time.

When these instructions run, microcontroller does not take and run any other interruptions

and miss these interruptions.

Serial prints means that it takes character through serial ports into the computer, and it takes time.

The response of ISR becomes slow and it causes delays and crashes.

ISR should be fast, and they are supposed to be run very fast, and the long code and serial points take time very much, so it is better to set a flag or store a value in ISR.

Question#6

What are the advantages of timer-based task scheduling?

A hardware timer based in microcontrollers that run to perform specific task automatically after a fixed time and regular interval.

- **Advantages:**
- It allows tasks to run very accurately with automatically fixed intervals without stopping the main program.
- Provides smother performance
- Better multitasking.

Question#7

Describe I^2C signals SDA and SCL.

I^2C is a communication system that connect multiple devices through two wires that are

SDA (Serial Data Line): It carries data between master and slave devices.

SCL (Serial Clock Line): it provides a clock signal that controls when the data is sent or received.

Both these wires help data to communicate in a synchronized way.

Question#8

What is the difference between polling and interrupt-driven input?

Polling: it means the microcontroller continuously checks the interruption and signal whether the event happens or not.

It is time consuming.

Wasting CPU time.

interrupt-driven input:

It allows the processor to continue other tasks and only respond when an event happens, through an interrupt signal.

It does not consume time and does not consume CPU.

Question#9

What is contact bounce, and why must it be handled?

Contact bounce happens when a mechanical switch or button is pressed, and its metal contacts rapidly make and break the connection several times before settling.

It must be handled because these fast, repeated signals can be mistaken as multiple presses, leading to wrong or unstable readings in the system.

Question#10

How does the LEDC peripheral improve PWM precision?

The **LEDC (LED Controller)** peripheral improves PWM precision by using hardware timers and high-resolution counters so that it can generate stable and accurate **PWM** signals.

It allows control over both the **frequency** and **duty cycle** with fine resolution, ensuring smoother brightness changes and more precise motor control.

Question#11

How many hardware timers are available on the ESP32?

ESP32 has four hardware timers

Each hardware timer works independently to precise timing and interrupt tasks.

Question#12

What is a timer prescaler, and why is it used?

A timer prescaler is a divider that slows down the input clock before it reaches the timer.

It is **used** to adjust the timer's counting speed, allowing it to measure longer time intervals or generate slower timing events accurately.

Question#13

Define duty cycle and frequency in PWM.

Duty cycle:

It means how long the signal **ON** and how long the signal **OFF.**

When the signal becomes off then no duty cycle is occurring because it mainly depends upon the factor of both on and off and only on.

Frequency:

The number of times on and off in one second is called frequency of the **PWM**.

It is measured in hertz (Hz).

Question#14

How do you compute duty for a given brightness level?

The duty cycle for the brightness level mainly depends upon the brightness and we can determine the duty cycle for the brightness level is determined by brightness level divided by a maximum brightness level and multiplied by 100.

Duty Cycle=Maximum Brightness/Brightness Level×100

Question#15

Contrast non-blocking vs. blocking timing.

Blocking timing stops the program and waits for a certain time before continuing (e.g., using delay (), which prevents other tasks from running.
non-blocking timing uses time checks (e.g., using millis()) so the program can continue running other tasks while waiting.
non-blocking timing makes the program more efficient and responsive.

Blocking Timing	**Non-blocking time**
The program stops and waits for delay to finish before moving on.	The program does not stop ,it keeps running other tasks while checking the time.
Uses functions like delay()	Uses function like millis()

Blocking time is less efficient

Non-Blocking time is more efficient.

Question#16

What resolution (bits) does LEDC support?

The LEDC (LED Controller) on the ESP32 supports PWM resolutions from 1 bit up to 20 bits. Higher resolution allows finer control over the duty cycle and smoother brightness or speed changes.

Question#17

Compare general-purpose hardware timers and LEDC (PWM) timers.

Both are used in **ESP32,** and they are used for timing and PWM control.

Hardware Timers	**LEDC Timers**
Used for delays, periodic interrupts and time measurements.	Used to generate **PWM signals** for controlling LED'S and brightness with vision.
High precision for time measurement.	High precision for duty cycle and frequency control.
Ideal for scheduling and triggering tasks.	Ideal for brightness and speed control applications.

Question#18

What is the difference between Adafruit_SSD1306 and Adafruit_GFX?

Both these libraries are used for display, and they are used when we are working on OLED for display.

Adafruit_SSD1306	**Adafruit_GFX**
This library is basically used for the OLED display, and it takes care of sending the display data on the screen.	This is a graphics library and it provides basic drawing functions (like drawing lines, circles and other different shapes)
Handles low level communication between the microcontroller and the OLED screen.	Does not control the screen directly,but gives the tool to draw shapes.

Question#19

How can you optimize text rendering performance on an OLED?

Text rendering performance on an OLED can be optimized by:

- Updating only the parts of the screen that change, instead of redrawing the whole display.
- Using smaller fonts and calling display. Display() only when needed.
- Avoiding frequent clear Display() commands and unnecessary graphics.

These methods reduce data transfer and make the display refresh faster and smoother.

Question#20

Give short specifications of your selected ESP32 board (NodeMCU-32S).

NodeMCU-32S (ESP32) Specifications:

- **Processor:** Dual-core 32-bit Xtensa LX6 CPU (up to 240 MHz)
- **Memory**: 520 KB SRAM, 4 MB Flash
- **Wi-Fi**: 802.11 b/g/n
- **Bluetooth**: v4.2 + BLE
- **GPIO Pins**: 30 (with PWM, ADC, DAC, I^2C, SPI, UART support)
- **Operating Voltage:** 3.3V
- **USB Interface**: Micro-USB for programming and power

Question#2

Logical Questions

Question#1

A 10 kHz signal has an ON time of 10 ms. What is the duty cycle? Justify with the formula

Duty cycle:

it means how long the signal **is on** and how long the signal **off.**

Frequency (f) = 10 kHz

ON time (TON) = 10 ms

Formula:

$$\textbf{Duty Cycle (\%)} = \frac{\textbf{ON Time}}{\textbf{ON +OFF time period}} \times 100$$

$$T = \frac{1}{10,000} = 0.0001 \text{ seconds} = 0.1 \text{ ms}$$

$$\textbf{Duty Cycle} = \frac{\textbf{ON Time}}{\textbf{ON+OFF time period}} \times 100$$

$$\textbf{Duty Cycle} = \frac{10 \text{ ms}}{0.1 \text{ ms}} \times 100 = 10,000\%$$

This result is not valid because the ON time (10 ms) is much longer than the total period (0.1 ms).
Hence, for a 10 kHz signal, ON time must be ≤ 0.1 ms for a valid duty cycle (0–100%).

Question#2

How many hardware interrupts and timers can be used concurrently? Justify.

The ESP32 provides **4 general-purpose hardware timers**, divided into two groups (Timer0–Timer3). All four timers can be used **concurrently**, as each operates independently.
Each timer can generate its own **hardware interrupt**, allowing **up to 4 timer interrupts** to work at the same time.
In addition, **multiple GPIO interrupts** can also run concurrently, since they are managed by separate interrupt sources.

Question#3

How many PWM-driven devices can run at distinct frequencies at the same time on ESP32? Explain constraints.

The ESP32's hardware PWM (the **LEDC peripheral**) provides:

16 channels (PWM outputs) total, and

8 timers (each timer can be configured with its own frequency and resolution).

Each timer defines a single PWM frequency (and resolution), and **one or more channels** are mapped to a timer.

Therefore the number of **distinct frequencies** you can run simultaneously equals the number of timers: **8**.
(If two channels use the same timer they must share the same frequency.)

The ESP32 supports **16 LEDC channels** and **8 LEDC timers**. Each timer defines one PWM frequency, so you can run **up to 8 devices at different frequencies** simultaneously (more devices can run, but channels sharing a timer must use the same frequency). Constraints include the 16-channel limit, the 8-timer limit, resolution vs. frequency trade-offs, prescaler/clock limits, and pin-muxing restriction

---

Question#4

Compare a 30% duty cycle at 8-bit resolution and 1 kHz to a 30% duty cycle at 10-bit resolution (all else equal).

Duty cycle = (ON time / total period) × 100%.
A **30% duty cycle** means the signal is ON for 30% of each cycle and OFF for 70%.

Resolution (in bits) means how finely the duty cycle can be divided into steps.

A 30% duty cycle at 8-bit resolution (1 kHz) and 10-bit resolution both produce the same ON/OFF ratio, but the 10-bit version offers **four times finer precision** (1024 vs 256 steps). The 10-bit PWM allows smoother changes in brightness or speed because each step represents a smaller change in duty cycle (~0.098% vs 0.39%).
Therefore, while the average power output is the same, the **10-bit PWM gives higher accuracy and smoother control**.

Parameter	8-bit, 1 kHz	10-bit, 1 kHz
Frequency	1khz	1khz
Duty cycle	30%	30%
Resolution	8 bits	10 bits(1024 levels)
ON value	0.30*256=77 counts	0.30*1024=307 counts

Step size	~0.39%	~0.098%
Brightness smoothness	Less smooth	Much smoother

Question#5

How many characters can be displayed on a 128*64 OLED at once with the minimum font size vs. the maximum font size? State assumptions.

Display resolution = **128 * 64 pixels** (width * height).

Minimum font = the common small built-in font (glyph 5*7 pixels) with **1 pixel horizontal spacing** and **1 pixel vertical spacing** == effective character cell **6 * 8 pixels**.

Maximum font (extreme case) = a single character uses the full display height (character height = 64 px). For simplicity assume character **width = character height** (square glyph). This gives the largest readable single-line character size.

Formula

- Characters per row = floor (display width / char width)
- Rows of text = floor (display height / char height)
- Total characters = characters_per_row × rows

Minimum font (6 * 8 px per character)

- Characters per row = floor (128 / 6) = 21
- Rows = floor (64 / 8) = 8
- **Total characters = 21 * 8 = 168 characters**

Conclusion (min font): 168 characters can be shown at once.

Maximum font (char size = 64 * 64 px, square)

- Characters per row = floor (128 / 64) = 2
- Rows = floor (64 / 64) = 1
- **Total characters = 2 * 1 = 2 characters**

Conclusion (max font, extreme): 2 characters can be shown at once (one 64-pixel high line with two wide characters).

On 128*64 **OLED**, with the **smallest font (6×8 pixels)**, about **168 characters (21*8)** can be displayed at once.
With the **largest font (64*64 pixels)**, only **2 characters** fit on the screen.
