IOT - SUM24
Faculty of Computers and Data Science
02-24-03205 - Field Training
Module 2 Assignment Report

Ahmed Mohamed Gaber
2203173

Hazem Ahmed Abdelfatah
2203175

Moataz Ali Ramadan
2203177

Momen Wagdy Hamed
2206134

Youssef Salah Mostafa
22010442

July 2024

Table of Contents

1 Tensilica Xtensa LX6

1.1 Introduction

Xtensa LX6 processor is a processor developed by Tensilica in May 2004. The main benefit of using Xtensa processors is their customizable instruction set. Cadence, a main contributor to the project, calls the processors DPUs (Data-plane Processing Units) due to their customizable nature. This allows SoCs (System on Chip) designers to alter and modify the chip according to its specific use case, enabling systems where the processor is modified and tailor-made for the specific task it's performing, cutting out any overhead that might otherwise be generated.[1]

The Xtensa LX6 processor is the one used in ESP32. It boasts a 32-bit architecture, a clock frequency of 240MHz on 2 cores, and scores 600 on the DMIPS benchmark.[2]

1.2 Features

The Xtensa LX6 has many features that make it a strong candidate for SoC designers. These features are crucial and often indispensable for designers, including:

- Efficient, small, and low power [1]
- IEEE 754-compliant single-/double-precision scalar floating-point unit.[1]
- High I/O bandwidth with multiple, wide, designer-defined FIFO, GPIO, and lookup interfaces.[1]
- Local memory configurable up to 8MB, which can be used for memory parity or ECC.[1]
- Automatic generation of matching software development tool-chains for each core.[1]

These are just a few features that differentiate the Xtensa LX6 from other microprocessors such as the ATmega series and other processors in the Xtensa series. Its use in ESP devices makes it a popular choice for SoC designers.

1.3 Why Tensilica Xtensa

Most processors are designed without a specific use in mind, which results in a rigid instruction set built to be produced many times with the same features for general use. This rigidity can lead to less efficiency and higher costs. The need for processors that can be tailored for specific use cases led to the development of Tensilica processors, which can be customized based on the intended application with an automated design process to save time.[3] For IoT applications, specific parts of the processor are needed more than

others, making it logical to remove unused parts to save memory and reduce power consumption.

2 Micro-processors and Micro-controllers

2.1 Micro-processors

A micro-processors is a single integrated circuit containing the logical units needed for computing, it is often referred to as the CPU, it functions on its own without any external device[4], it has a couple of benefits over micro-controllers such as:

- They have small foot prints compared to micro-controllers.[4]
- They have low costs, and can be customized easily since you decide how many I/O and peripherals you use.[4]
- Used in general purpose compute as small compute reserves.[4]

Micro-processors, however, still have some demerits where micro-controllers shine more, such as:

- They lack on-chip memory.[4]
- They do not have any built in peripheral functions.[4]
- They do not directly interface with I/O devices.[4]

For these reasons, and depending on the use case, some opt out and use micro-controllers in their systems; Understanding your system and its requirements is crucial in making the decision between the two, it also depends on technical expertise and complexity.

2.2 Micro-controllers

Micro-controllers are micro-processors that include memory, I/O interfaces, and built in functions. They are widely used in today's world and have become a crucial player in everyday life, especially in IoT systems.[4], they have a few benefits over micro-processors such as:

- They come with on-chip memory integrated into the board.[4]
- They do have many built in peripheral functions and protocols for communication.[4]
- They can directly interface with I/O devices and most modern micro-controllers can even read/write digital and analog data.[4]

However, like micro-processors,they have demerits that might stop you from using them in your systems, these include:

- They have relatively higher footprints.[4]
- Having higher costs compared to micro-processors.[4]

- Having too much overhead to be used as efficient compute reserves.[4]

Both the merits and demerits of micro-controllers and micro-processors were compared using the same points as to draw parallels to help in the decision making process.

3 Digital and Analog Signals

In the previous section, we talked about how micro-controllers can even read/write digital and analog signals, in this section we will be discussing the differences between both signals.

3.1 Digital Signals

Digital signals are a way of representing information through binary sequences, they carry discrete values, of HIGH and LOW, or 1 and 0.[5]

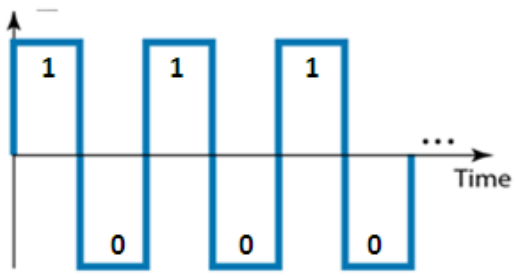


Figure 1: Digital Signals.[5]

A digital signal can consist of many levels, each level carries a bit sequence, the length of that sequence is defined by the relation: $B = log_2 L$, Where B is the number of bits in each level, and L is the number of levels.[5]

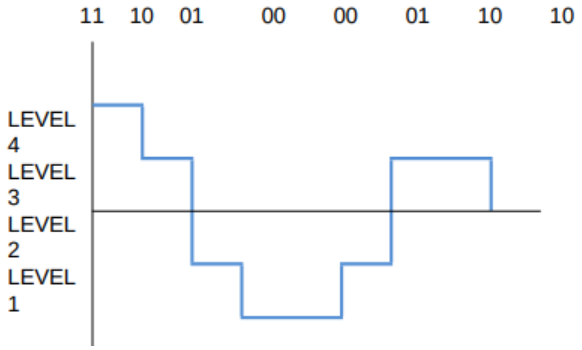


Figure 2: Multilevel digital signal.[5]

In digital signals we have a few terms that define a digital signal, these terms are used to differentiate each signal from the other, where 2 signals are equal

if and only if they are equal in all of their properties and the data carried.[5] These properties include:

- Bit length, it is the time needed to send one bit in seconds.[5]
- Bit rate, it is the number of bits sent in one second, measured in bits per second.[5]
- Baud speed, the rate at which the signal changes, measured in baud per second.[5]

To extract valuable information from digital signals, these signals can just be decoded back into their original form, where that is a flag, data, or identifier. They can also be broken down into an infinite series of sine waves or harmonics through Fourier analysis, more on that later.[5]

3.2 Analog Signals

Analog signals are a way of representing continuous data, such as audio, resonance frequencies, and magnetic effect. They have an infinite number of possible values unlike digital signals where they only have a set number of values. They are widely used in electronics, they are also used in modern control systems such as with fuzzy inference systems.[5]

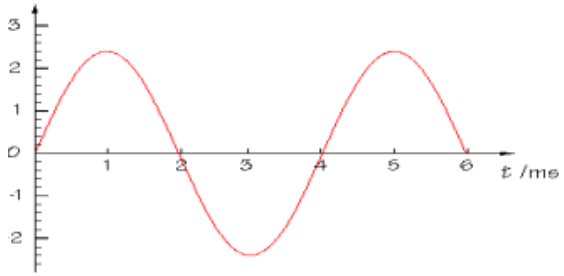


Figure 3: Sine wave.[5]

Analog signals are used in many electronic applications, so much so that there exists a branch of electronics for analog electronics. There are a few values that define an analog signal; Two analog signals are considered equal if and only if they have the exact same values for them.[5] These values are:

- Peak amplitude, it is the absolute value of the maximum intensity throughout the signal.[5]
- Frequency, it is the number of complete repetitions of a wave in a second.[5]
- Phase shift, it is the position of the wave at time 0.[5]

These values are, however, not enough to define or extract data from a signal. In many modern control systems, the device needs to track values of many functions to be able to detect and deal with faults and abnormalities. That is why for simple analog signals, Fourier analysis is used to predict the value of

a signal at time period t. The generating function for wave S can be defined as follow[6], where it can be thought of as a Fourier series of one wave:

$$S(t) = A\sin(2\pi ft + \phi)$$

Where A is the peak amplitude, f is the frequency, and ϕ is the phase shift.[6]

More complex analog signals, such as audio or video data, would need a bigger Fourier series for analysis, where many sine and cosine waves overlap to form the given wave.[6]

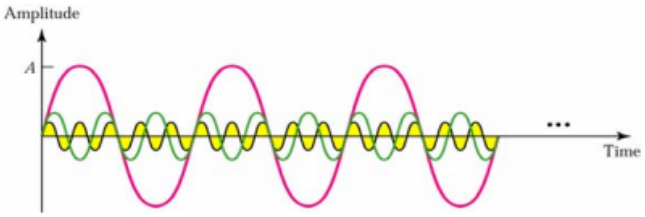


Figure 4: Three harmonics (sine waves) overlapping each other.[6]

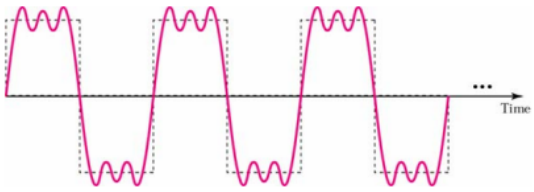


Figure 5: Three harmonics after adding.[6]

The general formula for the Fourier series can be defined as follows:

$$S(t) = A_0 + \sum_{i=1}^{\infty}[A_\mathrm{i}\cos(i\omega_0 t) + B_\mathrm{i}\sin(i\omega_0 t)]$$

Where A_0, A_i, B_i are variables that can be found through least square regression or integration, ω is the angular frequency equal to: $2\pi f$, and t is the time period.[7]

As we can see from *Figure 4* and *Figure 5*, if more terms, harmonics, were to be added in the series, the square wave would be formed more properly, however this acts as a good analyzer for modern control systems.

4 Interrupts and Timers on ESP32

4.1 Interrupts

Interrupts are events that do not normally occur in the normal flow of a program, they happen on the spur, so repeatedly checking for them would

be a waste of resources, instead board like the ESP32 define interrupts, 32 for each core on the ESP32, to aid engineers.[8] Normally if we want to check for a change in a value, we have to wait for the execution of the preceding instructions, however with interrupts, the controller stops the execution of the main routine and lends central control to the Interrupt Service Routine, ISR, where the interrupt is detected and dealt with accordingly.[8]

There are two main types of interrupts, Hardware interrupts, and Software Interrupts. Hardware interrupts are called upon a change in voltage across a pin, this change can vary across different interrupts, these changes include:

- LOW, triggered when the interrupt pin is LOW.[8]
- HIGH, triggered when the interrupt pin is HIGH.[8]
- CHANGE, triggered when any change in voltage is detected.[8]
- FALLING, triggered when the interrupt pin changes from HIGH to LOW.[8]
- RISING, triggered when the interrupt pin changes from LOW to HIGH.[8]

Software interrupts are called upon triggering an event in the program such as timer overflow.[8] Interrupts can be used in heavy machines when possibility for injury is detected, the sensor would send an interrupt signal and the execution of the machine would be halted at once.

4.2 Timers

Timers are hardware units on the ESP32 responsible for measuring accurate time intervals that aren't affected by ISRs, they are the perfect choice for accurate time sensitive tasks. The ESP32 has 4 general purpose timers, each 2 in one group, they are based on 64-bit counters, and 16-bit pre-scale divider that divide the base frequency, usually 80MHz, into smaller fractions so it can measure time flexibly.[9]

Timers are often paired with ISR, they function hand to hand to create time sensitive and precise devices. Timers with ISR can be used in alarm clocks, and high precision clocks.[9]

5 LCDs with ESP32

5.1 LCDs

Liquid Crystal Displays, LCDs, were developed by Austrian Chemist Friedrich Reiniter in 1888, they are made of special crystals that enter a "nematic" state when heated where they act as rods aligning with the polarity, when they are cooled they enter a "semectic" phase where they cannot pass other layers.[10]

By using filters that polarize light by only allowing light that is orthogonal to the filter, the LCD is lit up by a common light, and by twisting of the liquid

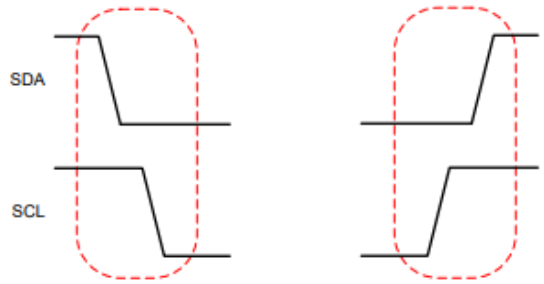
crystal, certain light is passed through showing color.[10] LCDs interface with micro-controllers such as ESP32 either using the 16 pins on the LCD, or through I2C, Inter Integrated-Circuit protocol, pronounced I Squared C, where I2C compliant boards are connected to the LCDs.

5.2 I2C Protocol

The I^{I}C or I2C protocol is a well known and popularly used protocol, from being used in serial communication between devices, and micro-controllers, to being used in more intensive units like thrusters.[11]

First off, the I2C protocol works over 2 buses, the SDA, Data bus, for transmitting data, and the SCL, Clock bus, for synchronization. When a controller device wants to connect using the I2C protocol, if the bus is open, it pulls the SDA to a low signal, followed by pulling the SCL to a low signal, indicating to all other devices that it is claiming communication, this is known as I2C START.[11]

If the device is done with its communication, it first releases SCL to high then releases SDA to high, indicating end of communication, this knows as I2C STOP.[11]



(a) I2C START. (b) I2C STOP.

Figure 6: SDA and SCL states in I2C START and I2C STOP.[11]

When sending data over I2C, the controller sends 2 types of frames. First one sent is the device address, consisting of 8-bits, then multiple data frames, each holding a byte of data. At the start of each address-frame pair, I2C START is sent, followed by the each frame and stopping with an ACK, Acknowledge, bit to verify the communication was successful, at the end of the interaction I2C STOP is sent to terminate the communication.[11] The address frame is made up of 8-bits, but only 7 are used for the address, the last bit is called R/W-bit, if it is 1, then the controller is asking to read data from the target device, if it is 0 then the controller is asking to write data to the target device.[11]

Due to its easy and versatile setup, I2C is chosen by a lot of engineers as a communication protocol in their systems and workloads, it can also

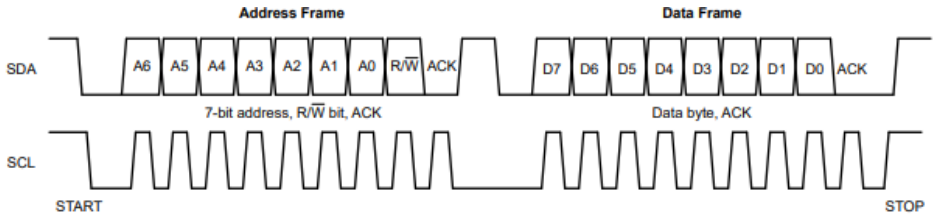


Figure 7: I2C Address and Data frames.[11]

communicate easily with other protocols through the used of decoders and converters.

6 IR Object Detection

6.1 IR Operations

IR, Infra-Red, sensors, work by emitting an IR signal using an emitter in the sensor, then detecting it using a photo-transistor that converts the incoming photons to electrons into the base of the transistor, which in turn allows current through across the transistor depending on the generated current.[12] Object detection occurs depending on the IR rays reflected back into the photo-transistor.

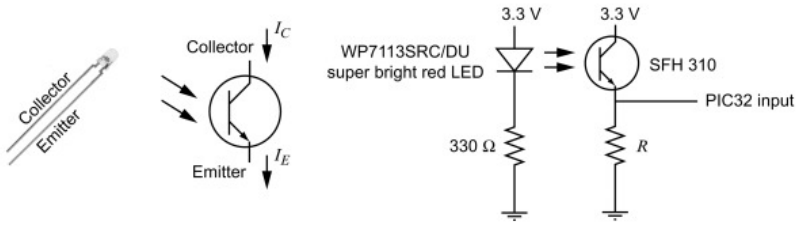


Figure 8: Photo-transistor schematic.[12]

6.2 Measuring Distance

Some engineers use a mapping function to detect the distance of an object from the IR sensor, but that method is far from accurate, to get an accurate result, the process is a bit more convoluted. First to we need to use Phong's equation of light intensity:

$$I = C_0(\vec{\mu}_\mathrm{s}\vec{\mu}_\mathrm{n}) + C_1(\vec{\mu}_\mathrm{r}\vec{\mu}_\mathrm{v})^\mathrm{n} + C_2 \quad (1)$$

Where C_0 is the Ambient and Diffuse reflection, C_1 is the Specular reflection, C_2 is the additional light contribution, and n is the shininess coefficient.[13] $\vec{\mu}_\mathrm{s}, \vec{\mu}_\mathrm{n}, \vec{\mu}_\mathrm{r}, \vec{\mu}_\mathrm{v}$ are the source vector, normal vector, reflected vector, and the viewing vector respectively.[13]

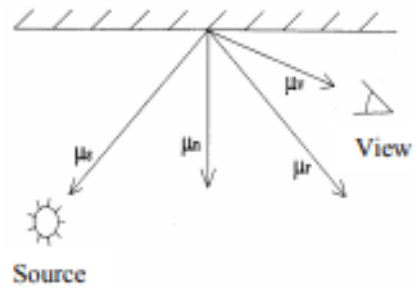


Figure 9: Phong's Model.[13]

The energy absorbed by photo-transistors is given by the equation:

$$E = \frac{IA}{(2l)^2} \quad (2)$$

Where I is the light intensity, A is the area of the photo-transistor, l is the distance travelled, and 2 is a scaling factor than can be omitted.[13]

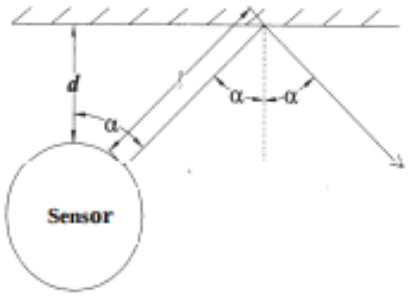


Figure 10: Phong's Model in IR sensor's case.[13]

In the case of the IR sensor, the IR source is the same as the IR view, so we can use geometric rules to figure out the angles between the vectors as α and 2α. From *Figure 10* we can update (1) by normalizing the vectors[13], the new equation yields:

$$I = C_0(\cos(\alpha)) + C_1(\cos(2\alpha))^{\mathrm{n}} + C_2 \quad (3)$$

We can also define an equation for l using trigonometric properties, the equation is:

$$l = \frac{d}{\cos\alpha} + r\left(\frac{1}{\cos\alpha} - 1\right) \quad (4)$$

Where d is the distance between the sensor and the object and r is the radius of the sensor[13].

We can set C_2 as 0, and n as 1, and A as a constant whose value will get distributed over the equation, by substituting (3) and (4) in (2) we get:

$$E = \frac{C_0 \cos(\alpha) + C_1(\cos(2\alpha))^n + C_2}{\left(\frac{d}{\cos(\alpha)} + r\left(\frac{1}{\cos(\alpha)} - 1\right)\right)^2}$$

To obtain the values of C_0 and C_1, we can simply move the device at a known distances and angles and solve the system of equations.[13]

Then after some operations we get the final formula for the distance:

$$d = r(\cos(\alpha) - 1) + \cos(\alpha)\sqrt{\frac{C_0 \cos(\alpha) + C_1 \cos(2\alpha)}{E}}$$

Through this formula we can detect the distance of any object, of different materials and textures, from the device with high precision, this is used in the vision systems of mobile robots.[13]

7 PWM on ESP32

7.1 Pulse Width Modulation

Pulse Width Modulation, PWM, is a technique to generate controllable waveforms of digital signals, it works using a timer extracted from the main clock signal, the timer counts up as its value is being compared, if it reaches duty cyle then the pin resets to LOW, then the timer continues until it reached the set period, then it is set as HIGH, and the timer resets to 0 and the process is repeated.[14]

The PWM frequency is $\frac{1}{T}$, where T is the period of each cycle. The PWM resolution is how many discrete levels of duty cycles we control, it is equal to $\log_2 levels$, so the number of levels we have in a PWM signal can be defined as $2^{\text{PWM Res}}$.[14] The higher the resolution the finer the generated signal is, and as PWM Resolution $\to \infty$, the generate wave becomes finer and more continuous.

To generate a PWM signal with an ESP32, all you need to do is check the data-sheet for a PWM pin, all pins on the ESP32 are PWM except for 4 input only pins, and the Vcc and Ground pins. Then in the code you define the values for the frequency, resolution and channel, and setup the PWD controller using them, then simply attach the pin to the channel.[14]

When using PWM, there are a few factors to consider, often times on micro-controllers, such as the ESP32, PWM controller has 16 channels, 8 high speed ones and 8 low speed ones, each group has 4 timers / 8 channels, so each 2 channels share a timer, which means they share the same frequency, so when dealing with multiple PWM pins, this is important to keep in mind.[14] One should also consider the needed frequency and resolution for each use case, since improper values for them might produce expected results.[14]

8 Matrix Keypad

8.1 Using with ESP32

Matrix keypads are used in many electronic system, the principle behind them is quite simple, they consist of 16 buttons and 8 pins, 4 row pins, and 4 column pins, each button is connected to its corresponding row and column pin depending on its place in the matrix, when a button is pressed, the circuit is closed and current is allowed to flow between its respective pins.[15]

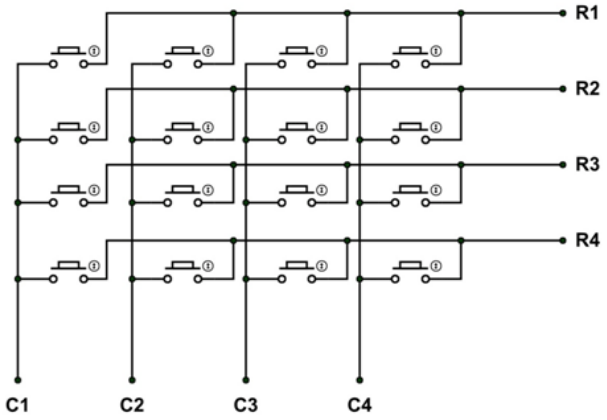


Figure 11: Keypad Schematic.[15]

To interface with the keypad using an ESP32, several libraries can be used, they all however function under the same principle. First we set all the row pins as OUTPUT, and set them to HIGH, and set the column pins as INPUT, we can then determine which column the pressed button is in, next we set all the column pins as OUTPUT, and set them to HIGH, and set the row pins as INPUT, we can then determine which row the pressed button is in. This way we get the indices of the pressed button, and using a defined array in the code, you can access the array using these indices to get the value of the button.[15]

9 Programming Problems

9.1 5s-Array Sum

In this question, we were asked to print summation of an integer array of multiples of 5. The summation is printed cumulatively while looping over the array.

```
#include <stdio.h>
#include <stdlib.h>
int *create_array(int size){
```

```
    int *arr = (int*) malloc(size*sizeof(int));
    for (int i= 0; i<size;i++){
        arr[i] = (i+1)*5;
    }

    return arr;
}


void main(){
    int *arr = create_array(5);
    int sum = 0;
    for (int i=0; i<5;i++){
            printf("Current element: %d\n",arr[i]);
            sum += arr[i];
            printf("Cumulative sum: %d\n",sum);
    }
    printf("The sum is %d\n",sum);


}
```

Figure 12: C Code for Problem 1.

This code contains one user-defined function, `create_array`, to create an array of multiples of 5 given the max array size, the array size is allocate using `malloc` to ensure it is dynamic and doesn't get allocated only in the function's scope. The cumulative sum is then printed.

9.2 Array Average

In this question, we were tasked to take an array input from the user of size n, and to calculate its average.

```
#include <stdio.h>

float calculateAverage(int array[], int size) {
    int sum = 0;

    for (int i = 0; i < size; i++) {
        sum += array[i];
    }

    return (float)sum / size;
}
```

```
int main() {
    int size, i;

    printf("Enter the number of elements
    in the array: ");
    scanf("%d", &size);

    int array[size];

    printf("Enter %d integers: ", size);
    for (i = 0; i < size; i++) {
        scanf("%d", &array[i]);
    }

    float average = calculateAverage(array, size);
    printf("The average of the entered
    numbers is: %.2f\n", average);

    return 0;
}
```

Figure 13: C Code for Problem 2.

This code contains one user-defined function, `calculateAverage`, which takes an array and its size as input, it then loops over the array and calculates its sum, then divides it by its size. First the user is prompted to enter the size of the array, then the user is tasked with filling it with numbers, then at last the average is printed.

9.3 Maximum Product

In this question, we were asked to find the maximum product of 3 integers in an array, this can be done by sorting the array an comparing the value of the smallest 2 multiplied by the greatest element and the value of the product of the top 3 greatest elements, this is done to account for negative values, the sorting algorithm used will be mentioned in 9.7.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int size, i;

    printf("Enter the number of elements in
    the array: ");
    scanf("%d", &size);

    int arr[size];
```

```
    printf("Enter %d integers: ", size);
    for (i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }


    printf("Given array is \n");
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");

    mergeSort(arr, 0, size - 1);
    int neg_max = 0;
    int pos_max = 0;
    neg_max = arr[0]*arr[1]*arr[size-1];
    pos_max = arr[size-1]*arr[size-2]*arr[size-3];


    int max = pos_max > neg_max? pos_max:neg_max;
    printf("The maximum product of 3 number is
    : %d\n", max);
    return 0;
}
}
```

Figure 14: C Code for Problem 3.

9.4 Vowel Count

In this question, we asked to count the number of vowels in a given vowels, this question refers to pure vowels such as a,e,i,o and u, and not diphthongs or glides such as h,y,w, and j in some words.

```
#include <iostream>
#include <cmath>
#include <string>
using namespace std;


int countVowels(const char*ptr){
  int vowelCount=0;
    while (*ptr != '\0') {
        if (tolower(*ptr)=='a' ||
        tolower(*ptr)=='e'||
        tolower(*ptr)=='i'||
        tolower(*ptr)=='o'||
        tolower(*ptr)=='u') {
```

```
                vowelCount++;
            }
            ptr++;
        }
        return vowelCount;
    };

int main() {
    cout << "Enter a word: ";
    string word;
    getline(cin, word);


    int numVowels = countVowels(word.c_str());
    cout << "The number of vowels in the string
    is: " << numVowels << endl;
  return 0;
}
```

Figure 15: C Code for Problem 4.

This code contains one user-defined function, `countVowels`, which takes a char pointer as input and loops over it till termination, if the pointer points to a vowel character the count is incremented, at the end the vowel count is printed.

9.5 Input-Array Sum

In this question we were asked to take the sum of an integer user-defined array. It is similar to the first problem, only difference is we take the array as input from the user.

```
#include <stdio.h>
int main() {
    int size, i,sum;
    printf("Enter the number of elements in the
    array: ");
    scanf("%d", &size);
    int array[size];
    printf("Enter %d integers: ", size);
    for (i = 0; i < size; i++)
        scanf("%d", &array[i]);
    for(int i = 0; i < size; i++) sum+=array[i];
    printf("The sum is %d ",sum);
    return 0;
}
```

Figure 16: C Code for Problem 5.

9.6 Factorial

In this question, we were asked to calculate the factorial of a given number n, where $f(n) = \prod_{i=1}^{n} i$.

```c
#include <stdio.h>

int main() {
    int num1, factorial = 1;

    printf("Enter an integer: ");
    scanf("%d", &num1);

    for(int i = num1; i >= 1; i--) {
        factorial *= i;
    }

    printf("Factorial of %d is %d\n", num1,
    factorial);

    return 0;
}
```

Figure 17: C Code for Problem 6.

In this question we take an integer input from the user and we loop until we reach zero, each step multiplying the `factorial` value with the loop iterator.

9.7 Merge Sort

In this question we were asked to implement the merge sort algorithm in C, with random array of size n, where n is chosen by the user, then we have to sort the array in ascending order and calculate the median for it.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void merge(int arr[], int left, int mid,
    int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int *L = (int *)malloc(n1 * sizeof(int));
    int *R = (int *)malloc(n2 * sizeof(int));

    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
```

```
    int i = 0;
    int j = 0;
    int k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
            } else {
            arr[k] = R[j];
            j++;
            }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }

    free(L);
    free(R);
}

void mergeSort(int arr[], int left,
    int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}


int main() {
    srand(time(NULL));
    int size;
    printf("Enter size of the array: ");
    scanf("%d",&size);
```

```
    int *arr = (int *)malloc(size *
    sizeof(int));
    for (int i = 0; i<size;i++) arr[i] =
    rand()% 101;
    printf("Given array is \n");
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");

    mergeSort(arr, 0, size - 1);

    printf("\nSorted array is \n");
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
    if (size % 2 ==0) {
        printf("The median is: %f\n",
        (float)(arr[(int)size/2]
        + arr[(int)size/2 + 1])/2);
    }
    else{
        printf("The median is: %d\n",
        arr[(size+1)/2]);
}
    free(arr);
    return 0;
}
```

Figure 18: C Code for Problem 7.

In this code we use the function `mergeSort` to repeatedly divide an array into 2 parts, then merge those 2 parts, the `merge` function works by copying the data from both given sides, and storing them in 2 arrays, then to update the original array with the smaller arrays, looping over each smaller array and comparing them both to append the smaller item first, then if there are any numbers that haven't been appended, we simply just append them. Then we calculate the median for the array, the median for the array and return it to the user, the median is :

For odd n:

$$\left(\frac{n+1}{2}\right)^{\text{th}} \text{ element when } n \text{ is odd}$$

For even n:

$$\frac{\left(\frac{n}{2}\right)^{\text{th}} + \left(\frac{n}{2}+1\right)^{\text{th}}}{2} \text{ for even values of } n$$

10 Schematics

10.1 ESP System

In this task, we were asked to implement an ESP32 system that has 3 main functions, first function is implementing a bar graph of LEDs that increase sequentially in series depending on the readings of a sensor, the second function is a push button that lights up an LED and can withstand glitches from the connection, the third function is an LDR that measures light intensity and triggers a buzzer and an LED when the light level is below a certain intensity.

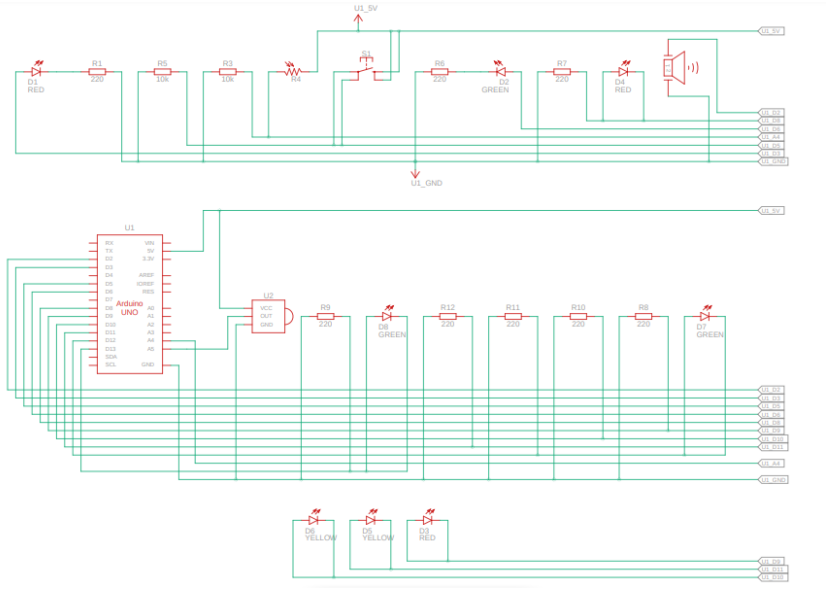


Figure 19: Question 1 Schematic.

```
int IR_sensor = 13;
const int LEDs_number = 9;
int LEDs[LEDs_number] = {23, 1, 3, 19, 18,
5, 16, 4, 15};
int LED1 = 12;
int button = 14;
int buzzer = 27;
int LED2 = 26;
int LDR = 25;

void setup() {
Serial.begin(115200);
pinMode(IR_sensor, INPUT);
for (int i = 0; i < LEDs_number; i++) {
  pinMode(LEDs[i], OUTPUT);
}
pinMode(LED1, OUTPUT);
pinMode(button, INPUT);
```

```
pinMode(buzzer, OUTPUT);
pinMode(LED2, OUTPUT);
pinMode(LDR, INPUT);
}

void loop() {
int bar_progress = map(analogRead(IR_sensor),
0, 4095, 0 , LEDs_number);
Serial.println(bar_progress);
for(int i = 0; i < LEDs_number; i++) {
  if (i < bar_progress) {
    digitalWrite(LEDs[i], HIGH);
  } else {
    digitalWrite(LEDs[i], LOW);
  }
}

int button_state = digitalRead(button);
digitalWrite(LED1, button_state);

int resistance = analogRead(LDR);
Serial.println(resistance);
if (resistance < 2000) {
  digitalWrite(buzzer, HIGH);
  digitalWrite(LED2, HIGH);
} else {
  digitalWrite(buzzer, LOW);
  digitalWrite(LED2, LOW);
}

delay(50);
}
```

Figure 20: Question 1 Code.

In this code, we first define the pins for the sensors, LEDs, button, buzzer and LDR, we then set the pin mode for all the define pins, we read the IR sensor and map it from 0 to the number of LEDs, then we light up the LEDs if their index is less than or equal to the mapped value, then we check the state of the button, then write it to the LED, by calling `delay()` later, we implement fault tolerance in the button, finally we check for the LDR reading and enable the buzzer and LED if the reading is less than 2000. The simulation for this circuit can be found here.

10.2 Smart door

In this task we were asked to implement a smart door, where if the IR sensor detects a person, the servo turns to 180 degrees, and waits for a set amount of time, then it closes if no person is present.

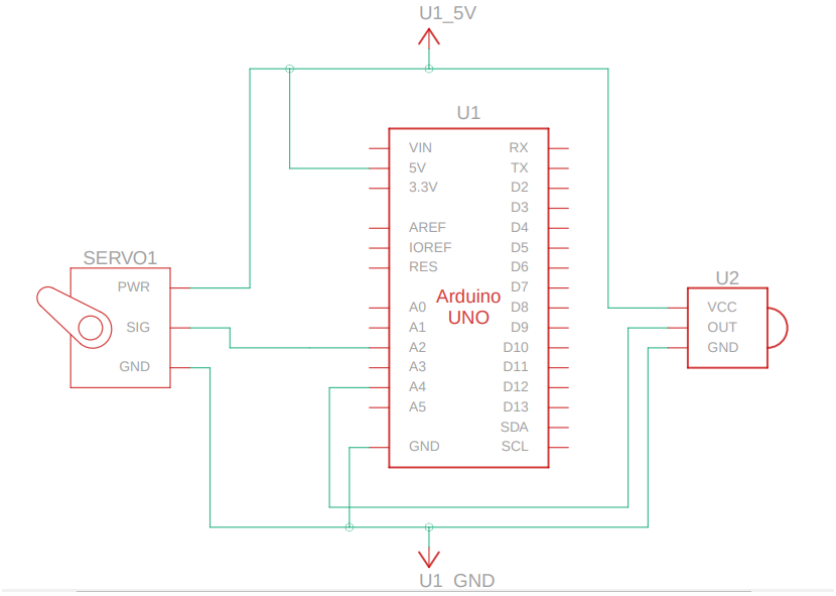


Figure 21: Question 2 Schematic

```
#include <Servo.h>

Servo myServo;
bool doorOpen = false;

void setup() {
  // Initialize serial communication
  Serial.begin(115200);

  // Attach the servo to pin 2
  myServo.attach(2);
  myServo.write(0);
}

void loop() {
  // Read the value from the IR sensor
  int irValue = analogRead(4);

  int outputValue = map(irValue, 0, 4095,
  0, 20);
  Serial.println(outputValue);

  // If a person is detected (outputValue == 0)
  and the door is not open, open the door
```

```
    if (outputValue == 0 && !doorOpen) {
      openDoor();
    } else if (outputValue >= 1 && doorOpen) {
      // If no person is detected and the door
      is open, close the door after a delay
      delay(10000);
      // Keep the door open for 10 seconds
      closeDoor();
    }

    // Small delay to avoid bouncing
    delay(100);
}

void openDoor() {
    //  open the door
    myServo.write(180);
    doorOpen = true;
    Serial.println("Door opened.");
}

void closeDoor() {
    //  close the door
    myServo.write(0);
    doorOpen = false;
    Serial.println("Door closed.");
}
```

Figure 22: Question 2 Code.

In this code, we first create a servo object, then attach the servo pin to it, and write 0, in the main loop we read the IR value and map it into a range of 0 to 20, given a certain value, we open the door, if the door is open but no person is detected we wait for 10 seconds and close the door. The simulation for this circuit can be found here.

10.3 Servo Control

In this task we were asked to implement a system using a servo motor, and a matrix keypad, where each key was mapped to a certain angle and you could reset, stop, and enter custom angles, the schematic for it can be found below.

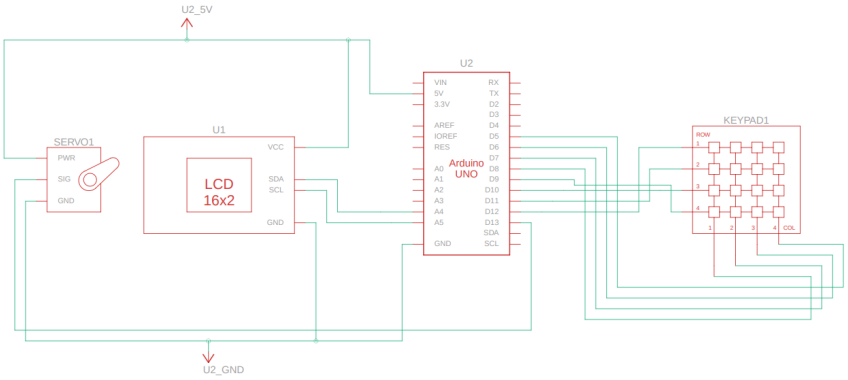


Figure 23: Question 3 Schematic

```
#include <Keypad.h>
#include <Servo.h>
#include <LiquidCrystal_I2C.h>

int pos = 0;
int servoPin=13;
int ind = 0;
int intr_in = 8;
int customArray[] ={0,0,0};
char buttons[4][4] ={{'1','2','3','A'},
                     {'4','5','6','B'},
                     {'7','8','9','C'},
                     {'*','0','#','D'}};
byte rowPins[4] = {12,11,10,9};
byte colPins[4] = {8,7,6,5};
bool customAngle = false;

Servo myservo;
LiquidCrystal_I2C lcd(0x27,16,2);
Keypad myKeypad =
Keypad(makeKeymap(buttons),rowPins,colPins,4,4);

int formAngle(int *arr){
int final = arr[0]*100+arr[1]*10+arr[2];

return final;
}
void interupts(){
   char customKey = myKeypad.getKey();
  if (customKey == '*')
  {myservo.detach();myservo.attach(servoPin);}
}


void setup() {
```

```
  myservo.attach(servoPin);
  lcd.init();
  lcd.backlight();
  attachInterrupt(digitalPinToInterrupt
  (intr_in), interupts, CHANGE);
}



void loop() {
 char customKey = myKeypad.getKey();
  if (customKey){
    int angle = 0;
    if (!customAngle){
      switch (customKey){
        case '1':
          angle = 20;break;
        case '2':
          angle = 40;break;
        case '3':
          angle = 60;break;
        case '4':
          angle = 80;break;
        case '5':
          angle = 100;break;
        case '6':
          angle = 120;break;
        case '7':
          angle = 140;break;
        case '8':
          angle = 160;break;
        case '9':
          angle = 180;break;
        case '0':
          angle = 0;break;
        case 'C':
          angle= pos; customAngle = true;break;
        default:
          angle = pos; break;

      }
      pos = angle;
      lcd.clear();
      lcd.home();
      lcd.print("Angle is: ");
      lcd.setCursor(11,0);
      lcd.print(angle);
      myservo.write(pos);
```

```
}
else{
  switch (customKey){
    case '1':
      customArray[ind] = 1;break;
    case '2':
      customArray[ind] = 2;break;
    case '3':
      customArray[ind] = 3;break;
    case '4':
      customArray[ind] = 4;break;
    case '5':
      customArray[ind] = 5;break;
    case '6':
      customArray[ind] = 6;break;
    case '7':
      customArray[ind] = 7;break;
    case '8':
      customArray[ind] = 8;break;
    case '9':
      customArray[ind] = 9;break;
    case '0':
      customArray[ind] = 0;break;
    case 'C':
      customAngle = false;break;
    case 'A':
      myservo.write(formAngle(customArray))
      ;ind=-1;break;
    default:
      ind -=1; break;

  }
  ind = (ind+1) % 3;
  lcd.clear();
  lcd.home();
  lcd.print("Custom Angle: ");
  lcd.setCursor(0,1);
  int newAngle = formAngle(customArray);
  if (newAngle < 100)
  {
    lcd.print("0");
    lcd.setCursor(1,1);
  }
  if (newAngle<10){
    lcd.print("0");
    lcd.setCursor(2,1);
  }
  lcd.print(formAngle(customArray));
```

```
        }
    }

}
```

Figure 24: Question 3 Code.

In the code, first we define the pins, the LCD, and the matrix for the matrix keypad, then create the interrupt for the stop function, and then we attach the servo motor, initialize the LCD, and attach the interrupt. In the main loop we fetch the pressed key, then go through a switch statement to decide the angle, you can input a custom angle by pressing C, then input the values through a switch statement, we use a 3 digit array for finding the value of the number, and convert it to int and write it to the servo by pressing A, on pressing $*$ the servo will halt its movement. The simulation for this circuit can be found here.

10.4 IR Reading

In this task, we were asked to read signals from an IR sensor and print them to an LCD display, and have error handling for when the sensor is not properly connected.

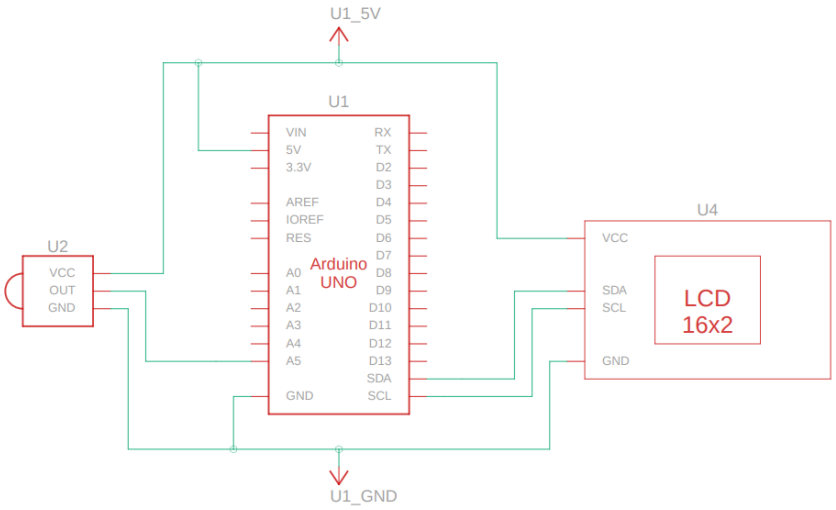


Figure 25: Question 4 Schematic.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

const int irSensorPin = A5;

LiquidCrystal_I2C lcd(0x27, 16, 2);
```

```
void setup() {
  Serial.begin(115200);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("IR Sensor Readings");
  delay(2000);
  lcd.clear();
}

void loop() {
  int irSensorValue = analogRead(irSensorPin);
  float distance = map(irSensorValue, 0, 4095,
  1, 25);

  if (irSensorValue == 0 ||
  irSensorValue == 4095) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("No input detected");
    lcd.setCursor(0, 1);
    lcd.print("Please connect");
    Serial.println("No input detected,
    please connect the sensor.");
  } else {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("IR Value: ");
    lcd.print(irSensorValue);
    lcd.setCursor(0, 1);
    lcd.print("Distance: ");
    lcd.print(distance);
    lcd.print(" mm");

    Serial.print("IR Sensor Value: ");
    Serial.print(irSensorValue);
    Serial.print(" | Distance: ");
    Serial.print(distance);
    Serial.println(" mm");
  }

  delay(500);
}
```

Figure 26: Question 4 Code.

In this code, we first define the IR pin and the LCD, then we initialize it and

print a starting statement, in the main loop we read the IR value, then map it to a value from 1mm to 25mm, if the sensor value is either 0, or 4095, depending on whether the pin is pulled HIGH or LOW, we alert the user about improper for faulty connection, other wise we print the sensor data and the distance data. The simulation for this circuit can be found here.

References

[1] Cadence. *Xtensa LX6 Customizable DPU*. https://mirrobo.ru/wp-content/uploads/2016/11/Cadence_Tensillica_Xtensa_LX6_ds.pdf.

[2] Gertech. *ESP32 Specification*. https://gertech.se/gertech/files/ESP32_Specification.pdf.

[3] Tensilica Inc. *Xtensa®Instruction Set Architecture (ISA)*. Tensilica Technical Publications, 2010.

[4] Ph.D. Alyssa J. Pasquale. *Microcontrollers*. College of DuPage, 2021.

[5] Sathyabama. *Signals errors*. https://www.sathyabama.ac.in/sites/default/files/course-material/2020-10/Unit2_6.pdf.

[6] N. Vlajic. *Analog and Digital Signals, Analog and Digital Signals, Time and Frequency Time and Frequency Representation of Representation of Signals*. https://www.eecs.yorku.ca/course_archive/2010-11/F/3213/CSE3213_04_AnalogDigitalSignals_F2010.pdf.

[7] Steven C Chapra. *NUMERICAL METHODS FOR ENGINEERS, SEVENTH EDITION*. McGraw-Hill Education, 2015.

[8] Jobit Joseph. *ESP32 Interrupt Tutorial*. https://circuitdigest.com/microcontroller-projects/esp32-interrupt.

[9] ElectronicWings. *ESP32 Timer Interrupts*. https://www.electronicwings.com/esp32/esp32-timer-interrupts.

[10] Marshall Williams. *LCD Displays ECE 480: Design Team 3*. https://www.egr.msu.edu/classes/ece480/capstone/spring15/group03/docs/ECE480ApplicationNote.pdf.

[11] Joseph Wu. *A Basic Guide to I2C*. Texas Instruments Incorporated, 2022.

[12] Dr. Thomas Kenny. *Sensor Technology Handbook*. Elsevier Inc., 2005.

[13] Tarek Mohammad. "Using Ultrasonic and Infrared Sensors for Distance Measurement". In: *World Academy of Science, Engineering and Technology* (2009).

[14] Khaled Magdy. *ESP32 PWM Tutorial Examples (AnalogWrite) – Arduino*. https://deepbluembedded.com/esp32-pwm-tutorial-examples-analogwrite-arduino/.

[15] Components101. *4x4 Keypad Module*. https://components101.com/misc/4x4-keypad-module-pinout-configuration-features-datasheet.