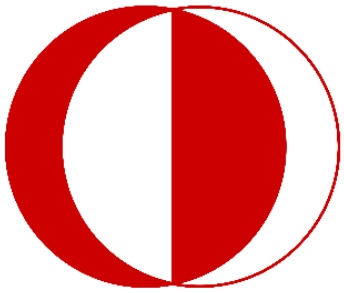

MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF ELECTRICAL- ELECTRONICS ENGINEERING

EE447 Introduction to Microprocessors
Term Project

Temperature-Initiated Object Detection

Final Report

Student Name **:** Birkan GENÇ & Zülal ULUDOĞAN

Student ID **:** 2443059 & 2444057

Date **:** 13.01.2025

1. Introduction

The term project for EE447 focuses on designing and implementing a temperature-initiated object detection system using the microcontroller. This project integrates knowledge of embedded systems, including interfacing multiple sensors, utilizing serial communication protocols (I2C, SPI), and handling complex tasks through efficient software design. The objective is to create a multi-functional setup capable of transitioning between deep sleep and active modes based on temperature thresholds. Key features include heat sensing, object detection via an ultrasonic distance sensor and stepper motor, and a user interface displayed on a LCD. The system utilizes some components including the TM4C123G microcontroller, LM35 analog temperature sensor, BMP280 digital pressure and temperature sensor, HC-SR04 ultrasonic distance sensor, Nokia 5110 LCD display, stepper motor, 4x4 keypad, multiturn trimpot, onboard RGB LEDs, an external 1W power LED, speaker, and supporting transistors. This project emphasizes hardware-software co-design, modularity, and the practical application of interrupt-driven programming in embedded systems.

2. System Design

2.1. Mechanical Design

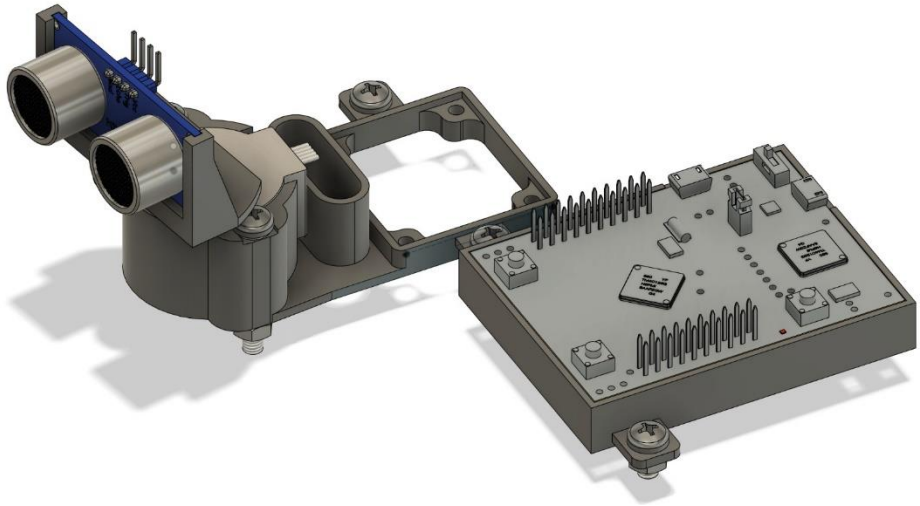

Figure 1: System mechanical design

The distance sensor is securely mounted on the shaft of the stepper motor using a custom-designed 3D-printed mount, enabling precise and stable movement during operation. This ensures accurate scanning of the environment, critical for detecting objects at various angles. Additionally, other 3D-printed components, as shown in Figure 1, enhance the system's compactness and organization, resulting in a functional and visually appealing design.

2.2. System States

The temperature-initiated object detection system is designed to transition between multiple states based on environmental triggers and user inputs. The system integrates multiple sensors, actuators, and peripherals to achieve a comprehensive solution for detecting objects when a temperature threshold is exceeded. The main states of the system are Deep Sleep, Object Scanning, and Active Operation, with each state playing a specific role in the overall workflow. The state diagram of the system is given in Figure 2.

- **State 1: Deep Sleep Mode**

  In the Deep Sleep Mode, the system operates in a low-power state to conserve energy by continuously monitoring the temperature using the LM35 analog temperature sensor, displaying the digital and analog temperature thresholds on the Nokia 5110 LCD, and entering deep sleep with the enterDeepSleep() function to minimize power usage. If the analog comparator detects a temperature above the threshold, the system wakes up and transitions to the Active Operation State.

- **State 2: Object Scanning**

  In this state, the system uses an ultrasonic distance sensor mounted on a stepper motor to scan an angular range of -90° to 90°, measuring distances at 10° intervals. The measured distances are stored in an array, and the LCD is updated with the scan data.

  The RGB LEDs indicate the detected object's distance:

- **Red LED:** Lights up for distances less than 50 cm.
- **Blue LED:** Activates for distances between 50 cm and 75 cm.
- **Green LED:** Turns on for distances between 75 cm and 100 cm.

During scanning, the RGB LED color changes dynamically based on the distance measurement at each interval. Once the scan completes, the LED indicates the minimum distance measured.

Throughout the scanning process, the system monitors the Deep Sleep button. If pressed, the sensor returns to its starting position, and the system transitions to Deep Sleep Mode. If not, it moves to the Active Operation State.

- **State 3: Active Operation**

The Active Operation State handles critical functions such as continuously updating the temperature using the BMP280 digital temperature sensor every second, displaying the detected object minimum distance, angle, digital temperature thresholds, and temperature on the LCD screen depending on the selected page, and activating the speaker when the digital temperature exceeds its threshold. If the digital temperature surpasses the threshold and the scanned_once variable is not set to one, the system transitions to the Object Scanning State to evaluate the environment. Otherwise, this state remains active until the user manually transitions the system back to Deep Sleep Mode using the on board SW1 button.

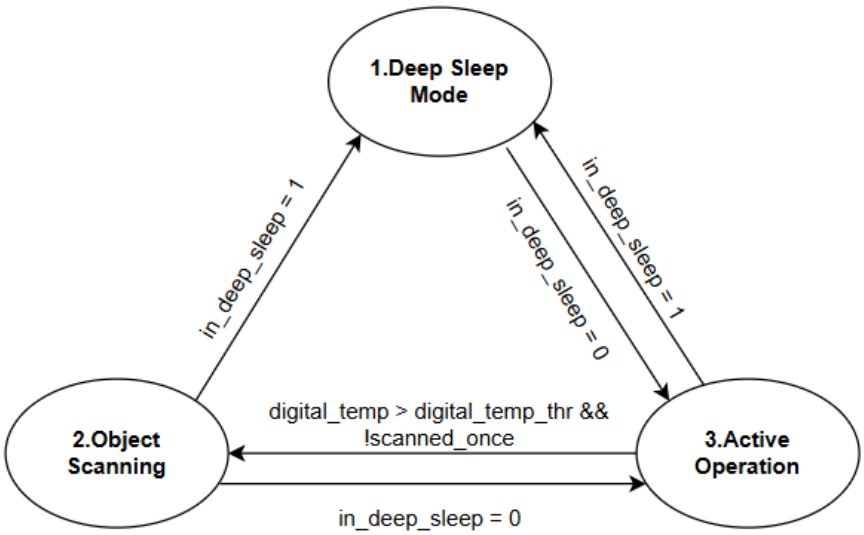

Figure 2: System state diagram

3. Sub-modules

3.1. HC-SR04 Ultrasonic Distance Sensor

The HC-SR04 ultrasonic distance sensor is a key component for detecting the presence and distance of objects in this system. It operates using ultrasonic sound waves to measure distances within a specified range.

- **Trigger and Echo Mechanism:** The HC-SR04 requires two signals for operation: a trigger signal to initiate the ultrasonic wave and an echo signal to measure the time of flight (TOF) of the wave. The TRIG pin is configured as an output, and a HIGH pulse of at least 10 microseconds is sent to initiate the transmission of an ultrasonic wave. This HIGH pulse is generated in the send_trigger_pulse() function by toggling the GPIO pin (PB5) connected to TRIG. The ECHO pin, on the other hand, is configured as an input with an alternate function, enabling it to capture the time of flight using the Timer1 capture module. The sensor sends a reflected ultrasonic wave back to the ECHO pin, where the timer measures the duration of the pulse.

- **Calculating Distance:** The distance is calculated based on the duration of the pulse measured on the ECHO pin. The dist_sense() function, given in Figure 3, accomplishes this by sending the trigger pulse through the send_trigger_pulse() function, then capturing the time of the rising edge (start of the echo pulse) and the falling edge (end of the echo pulse) using the Timer1 module. The pulse width, which represents the time taken for the ultrasonic wave to travel to the object and back, is computed using the formula,

$$Pulse\ Width = \frac{(Fall\ time - Rise\ Time)}{16}$$

where the division by 16 accounts for the clock cycles per microsecond. The pulse width is subsequently converted into distance in cm using the speed of sound with the formula,

$$distance = \frac{(pulseWidth\ * 0.34)}{20}$$

where the division by 2 adjusts for the round-trip time of the ultrasonic wave.

```
int dist_sense(int* dist_arr) {
    uint32_t rise_time, fall_time, pulse_width;
    // Send trigger pulse
    send_trigger_pulse();

    // Wait for rising edge (CAERIS flag)
    while ((TIMER1->RIS >> 2 &1) == 0);
    rise_time = TIMER1->TAR;  // Capture rising edge time
    TIMER1->ICR = 0x04;       // Clear capture flag or  TIMER1->ICR |= (1 << 2);

    // Wait for falling edge (CAERIS flag)
    while ((TIMER1->RIS >> 2 &1) == 0);
    fall_time = TIMER1->TAR;  // Capture falling edge time
    TIMER1->ICR = 0x04;       // Clear capture flag

    // Measure pulse width
    pulse_width = (fall_time - rise_time) / 16; // Convert to microseconds 1/16000000

    // Calculate distance and write to array
    *dist_arr = (pulse_width * 0.34) /2 /10;     // in cm record to array

    return 1;
}
```

Figure 3: dist_sense function definition

- **Module Implementation:** The implementation involves the GPIO module to configure PB5 (TRIG) as output and PB4 (ECHO) as input with alternate functions. The Timer1 module is used to capture the rising and falling edges of the echo signal, providing precise timing for the pulse width calculation. The TRIG pin sends a 10-microsecond HIGH pulse to emit an ultrasonic wave, which reflects off an object and returns as an echo signal to the ECHO pin. Timer1 measures the duration of the echo signal by capturing the rising and falling edges, and the time is converted into distance using the speed of sound formula. The result is stored in an array.

3.2. Stepper Motor

A stepper motor is a highly accurate electromechanical device used for precise angular positioning. It operates by sequentially energizing its internal coils, which results in controlled rotation of the motor shaft. In this project, the stepper motor is essential for scanning the environment, as it rotates an ultrasonic distance sensor within a specified angular range.

To achieve better precision and smoother motion, the motor is controlled using the half-step drive method. When the system transitions to the "Object Scanning" state, the stepper motor begins its operation by rotating the ultrasonic sensor incrementally through the defined angular range. At each 10-degree interval, the motor briefly pauses, allowing the distance sensor to capture measurements. Once the scanning process is complete, the motor returns the sensor to its initial starting position, ready for the next operation.

- **Driving the Stepper Motor:** The stepper motor is driven using four GPIO pins of Port B (PB0 to PB3) on the TM4C123G microcontroller, with each pin controlling one coil of the motor. The driving sequence is defined by a step table that outlines the activation pattern for each step, consisting of eight steps to ensure smooth and precise rotation. The set_step function energizes the appropriate coils by calculating the step number modulo 8 to stay within range and updating the GPIO pins (PB3 to PB0) according to the step table. The motor's movement is controlled by the move_motor function, which calculates the target step based on the desired angular movement and adjusts the motor's position step-by-step. A delay introduced by the low_Delay function ensures smooth operation and synchronization of the motor. In this project, the stepper motor is used to rotate the ultrasonic distance sensor through an angular range of −90 to +90 degrees with 10 degree intervals for object scanning. The move_motor function, given in Figure 4, receives the desired angle of rotation, converts it into the corresponding number of steps based on the motor's resolution of 4096 steps per revolution, and drives the motor step by step toward the target position. Key aspects of its operation include precision rotation enabled by the motor's step size and sequence, accurate angle-to-step conversion, and smooth movement facilitated by delays between steps to prevent abrupt motions that might affect sensor readings.

```
void move_motor(float degree) {
    step_dest += degree * 2*2048 / 360;
    while ((int)step_dest > step){
      step += 1;
      set_step(step);
      low_Delay(2000);
    }
    while ((int)step_dest < step){
      step -= 1;
      set_step(step);
      low_Delay(2000);
    }
}
```

Figure 4: move_motor function definition

3.3. LM35 Analog Temperature Sensor and Trimpot

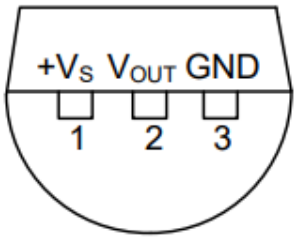

Figure 5: LM35 pin configuration

The LM35 is a precision integrated-circuit temperature sensor that outputs a voltage linearly proportional to temperature, increasing by 10 mV per degree Celsius. Its linear characteristic eliminates the need for external calibration. In this setup, the LM35's analog output is connected to the microcontroller's analog comparator module. The comparator continuously monitors the sensor's output against a dynamically adjustable reference voltage, set using a multi-turn trimpot. Acting as a variable resistor, the trimpot allows the user to modify the reference voltage as required, setting the desired temperature threshold.

- **Operation of the Analog Comparator:** When the LM35's output exceeds the threshold voltage, the comparator triggers an interrupt. This is handled by the COMP0_Handler function, which is given in Figure 6. It clears the interrupt flag to acknowledge the event. To ensure stable operation, a delay loop debounces the signal. The comparator's output status is then checked to confirm the condition persists. If true, the comparator interrupt is disabled, and the system transitions to an active state by setting the state variable (state) to 3. This mechanism ensures efficient temperature monitoring while enabling the system to wake from a low-power mode when necessary.

```
void COMP0_Handler(void) {
    COMP->ACMIS = 0x01;                     // Clear interrupt flag
    for (int i = 0; i < 100000; i++);       // Delay loop
    if ((COMP->ACSTAT0 & 0x02) == 0x02) {   // Check comparator output
        COMP->ACINTEN &= ~0x1;              // Disable comparator interrupt
        in_deep_sleep = 0;
        state = 3;
    }
}
```

Figure 6: COMP0_Handler function definition

- **ADC Configuration:** To read the trimpot's threshold value, the ADC1 module is used. The ADC1_Init function configures the ADC to operate on sequencer 3, sampling the analog signal from PE1 (AIN0). It enables the necessary clocks, sets the pin's alternate function, and configures the ADC for single sample operation. The ADC1_Read function triggers the sampling process, retrieves the result, and clears the interrupt flag. This digital result is stored in the analog_temp_thr_adc_value variable. Then, this value is converted to temperature value to represent the user-defined temperature threshold using Equation 1.

$$\text{analog\_temp\_thr} = \text{analog\_temp\_thr\_adc\_value} * \frac{3.3}{4096} * 100 \quad (1)$$

3.4. BMP280 Pressure and Temperature Sensor

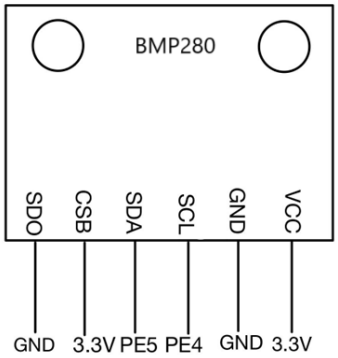

Figure 7: BMP280 pin configuration

The BMP280 is a high-precision, low-power sensor used for measuring atmospheric pressure and temperature. Communication with the BMP280 can be established through I2C or SPI interfaces. In this setup, the BMP280 is connected to the microcontroller using the I2C2 module on pins PE4 (SCL) and PE5 (SDA). The BMP280's pin configuration and its connections are given in Figure 7.

- **I2C Communication Overview:** The Inter-Integrated Circuit (I2C) is a synchronous, multi-master, multi-slave communication protocol commonly used for interfacing peripherals in embedded systems. It operates on two wires: the Serial Clock Line (SCL) and the Serial Data Line (SDA), both of which are open-drain and require pull-up resistors. Data transmission begins with a start condition, followed by the transmission of the 7-bit slave address and a read/write bit. Data bytes are then exchanged between the master and slave. The communication ends with a stop condition.

- **Initializing the I2C2 Module:** The I2C2_Init function sets up the I2C2 module for communication. This involves enabling the clock for the I2C2 peripheral and Port E, configuring the alternate functions for the PE4 and PE5 pins, and enabling the digital functions for these pins. Pull-up resistors are also activated to ensure reliable communication on the I2C bus. The module is configured as a master, with a clock speed of 100 kHz suitable for standard mode I2C communication. This initialization ensures a stable and ready interface for data exchange with the BMP280.

- **Writing and Reading Data via I2C:** The I2C_Write function allows the microcontroller to send data to the BMP280. It first sends the device's I2C address in write mode (RW = '0'). The BMP280's I2C address is determined by the state of the SDO pin: when SDO is connected to ground (low), the device address is 0x76. If SDO is connected to VCC (high), the device address becomes 0x77. Multiple byte writes can be achieved by sending pairs of register addresses and corresponding data, ending with a stop condition. In the case of reading data, the I2C_Read_Multiple function is used. To read registers, the master first sends the slave address in write mode along with the desired register address. Following this, either a stop or repeated start condition is issued, and the slave is readdressed in read mode (RW = '1'). The BMP280 then sends out data from auto-incremented registers until a NOACK and stop condition is generated. This two-step read process, involving separate write and read operations, is a characteristic feature of the BMP280 compared to other sensors that may allow direct sequential reads.

- **BMP280 Initialization and Calibration:** The BMP280_Init function configures the BMP280 to operate in normal mode with oversampling enabled for temperature measurement. The control and configuration registers (0xF4 and 0xF5) are set to define the operating mode, oversampling rates, and standby duration. Calibration data stored in the BMP280's non-volatile memory are read using the BMP280_ReadCalibration function, which retrieves six bytes from the calibration registers (starting at address 0x88). These calibration parameters are crucial for converting raw sensor data into accurate temperature values.

- **Reading Temperature from the BMP280:** The temperature is read using the BMP280_ReadTemperature function. This function reads three bytes of raw temperature data from the BMP280's temperature registers (starting at 0xFA) and applies the sensor-specific compensation formula. The formula uses the calibration coefficients to compute a fine-tuned temperature value, which is returned in hundredths of a degree Celsius for high precision. This ensures that temperature measurements are accurate and ready for application-specific use.

- **Averaging Temperature Data:** To improve reliability and reduce noise, the BMP280_GetAverageTemperature function computes the average temperature over 128 readings. This involves repeatedly calling the BMP280_ReadTemperature function, summing the results, and dividing by the number of samples. The final value is converted to degrees Celsius, offering a stable and accurate representation of the ambient temperature.

3.5. NOKIA 5110 LCD Screen

To drive and initialize the Nokia 5110 LCD, Prof. Valvano's functions are used [2]. The LCD screen is organized into a grid of 12x6 blocks, where the coordinate (0,0) corresponds to the top-left corner of the screen and (11,5) represents the bottom-right corner. This structured layout enables precise positioning of printed strings and graphical elements on the display.

The function *void Nokia5110_SetCursor(uint8_t x, uint8_t y)* sets the cursor's current position on the LCD, enabling text or graphics to be placed at a specific location. Once the cursor is

positioned, the function *void Nokia5110_OutString(char ptr)* is used to print a string starting from the cursor's position. Each character in the string occupies one block of the display grid, ensuring neatly aligned and organized text output.

Additionally, two custom functions were implemented to print numerical values on the LCD:

- **void print_number_int(uint8_t x, uint8_t y, int number)**: Prints an integer at the specified location.
- **void print_number_float(uint8_t x, uint8_t y, float number)**: Prints a floating-point number at the specified location.

These functions convert the input numbers into strings before using the *Nokia5110_OutString* function internally to display them on the screen. This design provides flexibility for handling and displaying both integer and floating-point values efficiently.

Figure 8 illustrates the LCD screen display in Sleep mode, which includes the Digital Threshold and Analog Threshold values set by the user. Whenever the Digital Threshold is changed, the screen automatically refreshes to reflect the new value. However, to update the Analog Threshold value displayed on the screen, the user must press Button 1, which triggers a manual refresh of the screen.

Figure 8: Sleep Mode LCD Screen

In the Awake operation state, there are three distinct pages:

- **Page 1 (Figure 9)**

  This page displays the current Digital Threshold and the measured Digital Temperature value.

- **Page 2 (Figure 10)**

  This page provides a graphical representation of scanned objects. The function *void PlotScanData(int distances, int numPoints)* is used to plot the data on the screen. At the bottom of the display, the initial position of the sensor is marked with a triangle. The screen contains 19 2x2 pixel points, each representing the measured distance and angle for intervals ranging from -90° to 90° in 10° steps. If a measured distance exceeds 100 cm, the corresponding pixel is placed at the maximum possible distance for that angle.

- **Page 3 (Figure 11)**

  This page shows the minimum measured distance of the scanned object and its corresponding angle. During the Object Scanning State (State 2), this page is also displayed by default and is continuously updated to reflect the current minimum distance measurement. During scanning state, if the measured distance is more than 100cm it will keep displaying “No Object Detected!” until it measures a distance less than 100cm.

Figure 9: LCD screen page 1 output

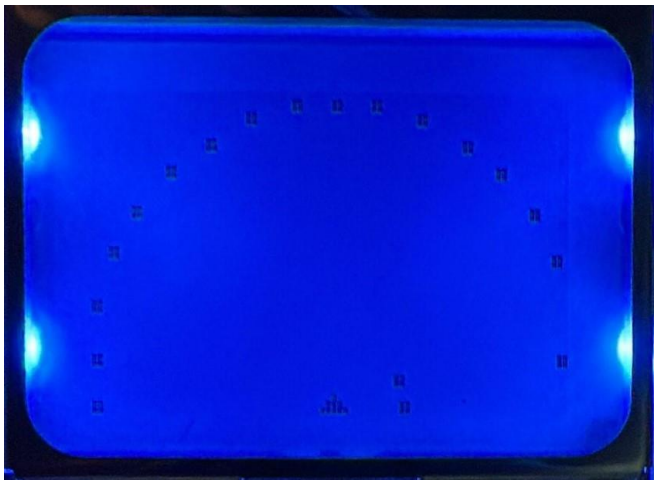

Figure 10: LCD screen page 2 output

Figure 11: LCD screen page 3 output

3.6. 4x4 Keypad

Figure 12: 4x4 Keypad

The 4x4 keypad module serves as an essential user interface for the system, seamlessly integrating hardware control with user interactions. This module is implemented in ARM Assembly to fulfill the project requirement of developing at least one submodule or function using assembly language.

The keypad is connected to the **GPIOD pins**, with the **GPIOD_Handler** function handling interrupts triggered by button presses. These interrupts ensure the system can process user inputs smoothly and responsively. Each button on the keypad, as illustrated in Figure 12, is assigned to a specific function to enhance system control.

- **Button 1 (PD3):** Updates the analog threshold displayed on the LCD screen. When pressed, the system retrieves the current analog temperature threshold using the ADC1 module, converts it into a voltage, and scales it to a temperature value. This value is then updated on the LCD for user reference.

- **Button 2 (PD2):** This button is used to change the page displayed on the LCD screen, enabling users to navigate through various system parameters or configurations. Each press of the button increments the page variable, cycling through the available pages within the system. The functionality of this button is exclusive to the Active Operation State (State 3), as it is the only state in which multiple pages are available for user interaction.

- **Button 3 (PD0):** Decreases the digital temperature threshold by 1 degree. Before making any changes, the system ensures that the current threshold value is above the minimum allowable limit (**min_digital_temp_thr**). This verification prevents the threshold from falling below its defined minimum, allowing users to make precise adjustments while maintaining system integrity.

- **Button 4 (PD1):** Increases the digital temperature threshold by 1 degree. Similarly, the system ensures that the threshold is within a valid range before incrementing it by 1, giving users the ability to fine-tune the settings.

3.7. Speaker

The speaker is an 8-ohm device that converts electrical signals into sound waves. Since speakers require more current than a GPIO pin on a microcontroller can supply, an external 9V power supply is used to drive the speaker effectively. The BC547 transistor acts as a switch, enabling the microcontroller to control the speaker's operation. The transistor amplifies the current from the GPIO pin to a level sufficient to drive the speaker. Resistor R1 (10kΩ) given in Figure 13 limits the base current to the transistor, preventing damage and ensuring proper operation.

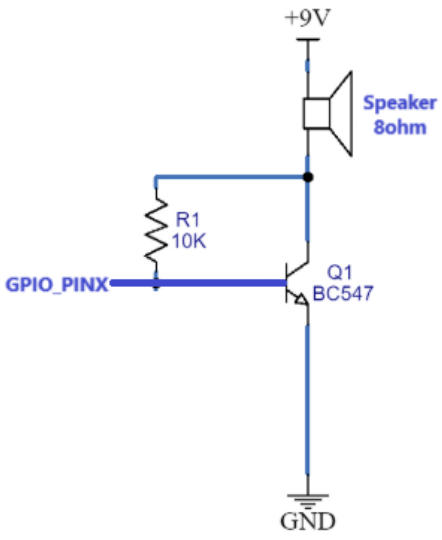

Figure 13: Speaker circuit

- **Use of Timers and Interrupts:** The GPIO pin connected to the base of the transistor is configured to output a Pulse Width Modulation (PWM) signal. Timers and interrupts play a crucial role in the precise generation of PWM signals. Timer0 operates in a periodic mode and triggers an interrupt at specified intervals. Within the interrupt

handler (TIMER0A_Handler), the GPIO pin's state toggles to produce the desired PWM signal. The timer's interval is adjusted dynamically based on the current state (high or low) of the GPIO pin, allowing seamless control over the signal's duty cycle and frequency. Interrupts ensure non-blocking operation by allowing the microcontroller to handle other tasks while the timer manages the PWM generation.

The speaker_activate() function is called to enable the timer and generate the sound signal through the speaker. This function enables the interrupt and timer for TIMER0. The sound is played for a duration of 2 seconds, controlled by the Delay(500) function, which introduces a delay. After the specified duration, the Timer0 interrupt (TIMER0->IMR) is disabled to stop the PWM signal generation, and the timer (TIMER0->CTL) itself is also disabled to conserve resources and stop further toggling of the GPIO pin. Lastly, the timer interval load register (TIMER0->TAILR) is reset to a default value of 10,000 to prepare for future use. This ensures that the speaker stops producing sound after the desired duration, maintaining precise control over the sound's timing and preventing unintended operation. The code can be seen in Figure 14.

```
speaker_activate();     // play sound for 2 seconds and disable timer0 for speaker
Delay(500);
TIMER0->IMR &= ~0x01;   // Disable Timer0A interrupt
TIMER0->CTL &= ~0x01;   // Disable timer during setup
TIMER0->TAILR = 10000;
```

Figure 14: Timer0 enable and disable code

The TIMER0A_Handler function, given in Figure 15, dynamically adjusts the frequency of the sound produced by the speaker based on the temperature reading obtained from the BMP280 sensor. The temperature data, represented by the digital_temp_thr variable, is compared against a baseline or threshold value (min_digital_temp_thr). This comparison determines the overall duration of the PWM signal cycle, which in turn dictates the frequency of the sound emitted by the speaker. The duration of one complete PWM cycle (comprising high and low phases) is inversely proportional to the temperature. When the temperature increases, the duration of the cycle decreases, resulting in a higher frequency and a correspondingly higher-pitched sound.

```
void TIMER0A_Handler(void) {
    TIMER0->ICR = 0x01;          // Clear the interrupt flag

    duration = 10000/((digital_temp_thr - min_digital_temp_thr) +1);
    high_time = duration*0.6;
    low_time =  duration*0.4;

    GPIOC->DATA ^= 0x10;             // Toggle PC4

    // Adjust Timer Interval Load (TAILR) for the next cycle
    if (GPIOC->DATA & 0x10) {        // If PA2 is HIGH
        TIMER0->TAILR = high_time; // Set HIGH duration

    } else {
        TIMER0->TAILR = low_time;  // Set LOW duration
    }
}
```

Figure 15: TIMER0A_Handler function definition

3.8. 1W Power LED

The system exits deep sleep mode when the measured temperature, detected by an analog sensor, exceeds the predefined analog threshold. Upon this condition, a GPIO pin output is set high, turning on the BC547 transistor by allowing current to flow through its base via the resistor. This action completes the circuit for the 1W power LED, allowing it to turn on and indicating that the system has activated in response to the temperature threshold being exceeded. The constructed power LED circuit is shown in Figure 16.

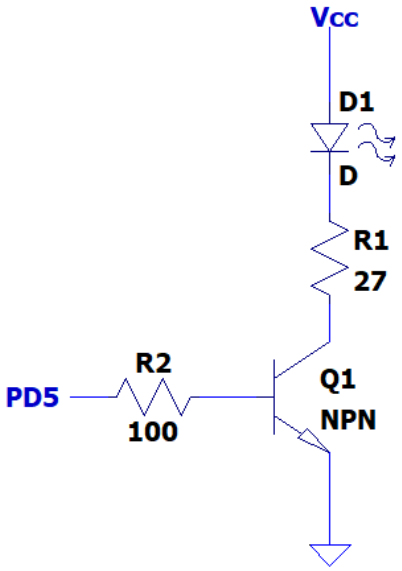

Figure 16: 1W Power LED circuit

4. Conclusion

By leveraging the TM4C123G microcontroller and a combination of analog and digital sensors, the system demonstrates the practical application of key embedded systems concepts, including interrupt-driven programming, low-power design, and effective user interaction through a keypad and LCD. Key achievements of this project include the seamless transition between operational states, accurate temperature monitoring, and reliable object detection using ultrasonic sensors and stepper motor integration. The dynamic use of visual and auditory indicators enhances user experience, while careful hardware-software co-design ensures robustness and energy efficiency.

This project shows the importance of modularity in embedded design, as well as the value of combining mechanical, electrical, and software engineering principles to address real-world challenges. The experience also provided an opportunity to implement and understand I2C and SPI communication protocols, LCD screen integration, timers, interrupts, and the analog comparator module. These components and techniques further enriched the system's functionality and contributed to a deeper understanding of embedded system design, laying a strong foundation for future endeavors in the field.

5. References

[1] TI, “Tiva™ tm4c123gh6pm microcontroller data sheet.”

Tiva™ C Series TM4C123GH6PM Microcontroller Data Sheet datasheet (Rev. E)

[2] J. Valvano, “Starter files for embedded systems.”

Starter files for embedded systems

[3] TI, “LM35 Precision Centigrade Temperature Sensors data sheet”

https://www.ti.com/lit/ds/symlink/lm35.pdf

[4] Bosch, “BMP280: Data sheet”

https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf

APPENDIX

Table 1. Microcontroller Pin Connections.

	Destination	Microcontroller PIN
1W Power LED Drive	Transistor Base	PC5
Speaker Drive	Transistor Base	PC4
4x4 Keypad	S1	PD3
	S2	PD2
	S3	PD0
	S4	PD1
HC-SR04 Ultrasonic Distance Sensor	Echo	PB4
	Trig	PB5
Step Motor	IN1	PB0
	IN2	PB1
	IN3	PB2
	IN4	PB3
BMP280 Digital Temperature Sensor	SCL	PE4
	SDA	PE5
	CSB	3.3V
	SDO	GND

NOKIA 5110 LCD Screen	**RST (Pin 1)**	**PA7**
	CE (Pin 2)	**PA3**
	DC (Pin 3)	**PA6**
	Din (Pin 4)	**PA5**
	Clk (Pin 5)	**PA2**
	Vcc (Pin6)	**3.3V**
	BL (Pin 7)	**GND**
	Gnd (Pin 8)	**GND**
Analog Comparator	**Trimpot**	**PC7**
	LM35 Analog Temperature Sensor (V_{out})	**PC6**
ADC	**Trimpot**	**PE3**