Temperature-Initiated Object Detection

Mert EKREN 2575173
Sefer İDACI 2575421

Introduction

This project aims to teach us how to implement an energy-efficient temperature monitoring system using the Tiva C Series TM4C123GH6PM microcontroller. The system uses analog sensors, including an LM35 temperature sensor and a trimpot for threshold adjustment, to trigger actions based on temperature readings. It uses the microcontroller's analog comparators and ADC modules to process sensor data and determine when the system should wake from deep sleep mode indicated with the opening of a power LED. Additionally, there is also an additional temperature sensor (BMP280) and its threshold is adjusted using a 4X4 Keypad. It is used to confirm the temperature threshold is indeed crossed after that a timer is configured to activate a buzzer to indicate the crossing. LEDs are used to provide real-time visual feedback on distance levels, indicating whether there is an object within the 1 meter radius in the arc which will be checked by HC-SR04 and a stepper motor then printed on an LCD screen as distance and angle with digital threshold and temperature if such an object exists.

General Solution

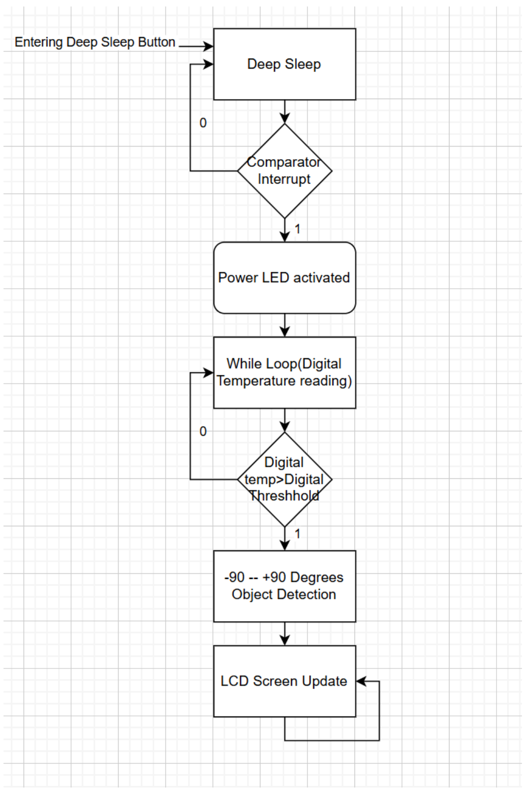
Entering Deep Sleep Button
Deep Sleep
0
Comparator Interrupt
1
Power LED activated
While Loop(Digital Temperature reading)
0
Digital temp>Digital Threshhold
1
-90 -- +90 Degrees Object Detection
LCD Screen Update

Figure 1. Flowchart of the System

Initially, GPIO ports, ADC modules, comparators, and communication protocols (I2C and SPI) are configured to interface with sensors (LM35, BMP280, HC-SR04) and output devices (RGB LED, buzzer, Nokia 5110 LCD). The system enters deep sleep mode to minimize power consumption and wakes up when the LM35 sensor detects a temperature above a threshold set by a trimpot, triggering an interrupt. Upon waking, the BMP280 confirms the temperature using I2C, and if it exceeds a user-defined digital threshold entered via a 4x4 keypad, further actions are initiated. The stepper motor scans an arc from -90° to +90°, while the HC-SR04 sensor detects object distance, displaying results on the LCD via SPI communication. Visual feedback is provided by an RGB LED, and a buzzer alerts the user if critical conditions are met.

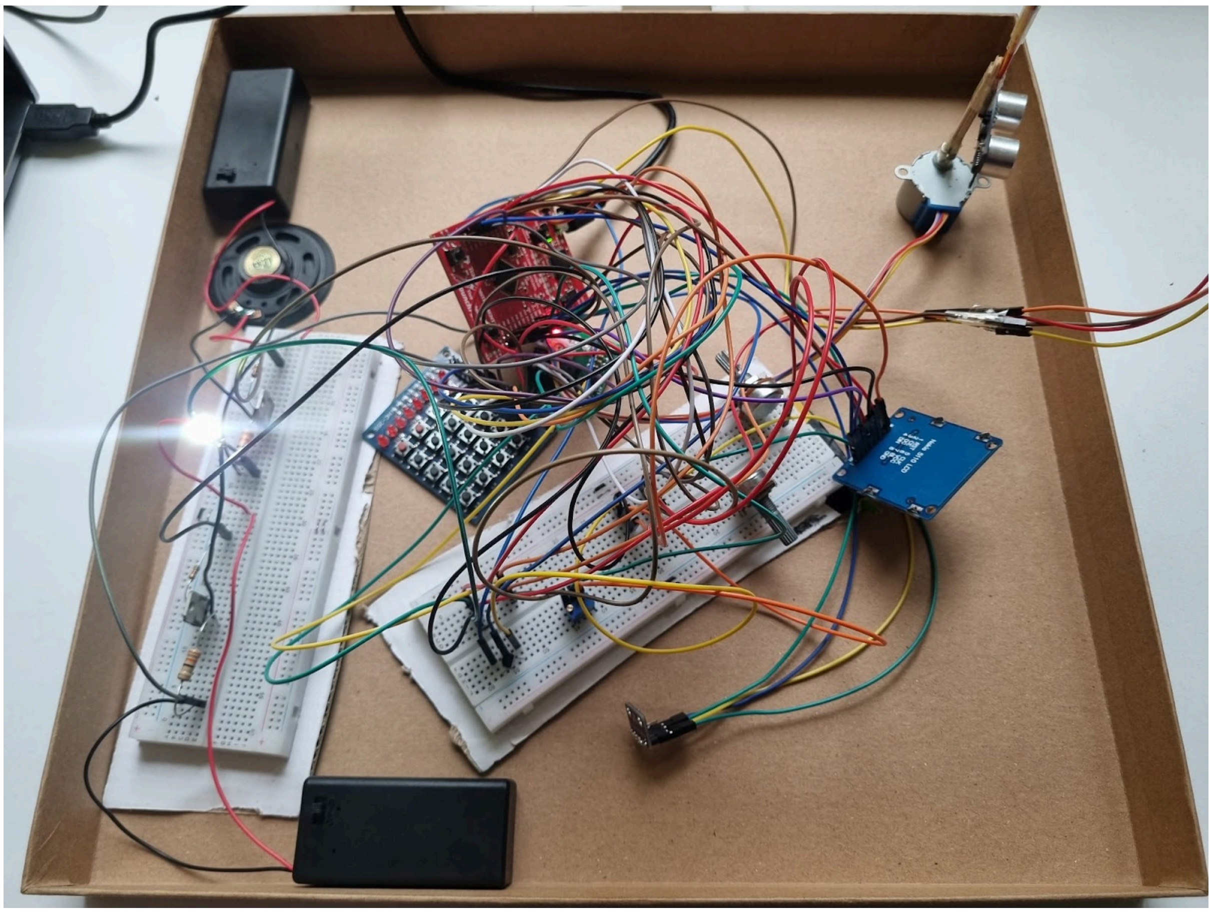

Figure 2. Setup For the Project

General Pinout

- **PE2**: R1 of the Keypad
- **PE3**: R2 of the Keypad
- **PE4**: R3 of the Keypad
- **PE5**: R4 of the Keypad
- **PD0**: L1 of the Keypad
- **PD1**: L2 of the Keypad
- **PD2**: L3 of the Keypad
- **PD3**: L4 of the Keypad

- **PB0**: Input 1 for ULN2003 A
- **PB1**: Input 2 for ULN2003 A
- **PC0**: Input 3 for ULN2003 A
- **PC1**: Input 4 for ULN2003 A
- **VBUS**: 5v Input for Step Motor Driver and the Ultrasonic Distance Sensor
- **GND**: General Ground
- **PB4**: Echo pin for the Ultrasonic Distance Sensor
- **PB5**: Trigger pin for the Ultrasonic Distance Sensor
- **3.3V**: 3.3v Input for Step Motor Driver and the Ultrasonic Distance Sensor
- **GND**: General Ground
- **PB2**: SCL Pin for BMP280
- **PB3**: SDA Pin for BMP280
- **GND**: General Ground 8
- **3.3V**: 3.3v Input for THE LCD Screen
- **PA7**: Reset pin for the LCD
- **PA3**: CE pin for the LCD
- **PA6**: DC pin for the LCD
- **PA5**: Din pin for the LCD
- **PA2**: CLK pin for the LCD
- **PC6**: C0+ for the comparator LM35
- **PC7**: C0- for the comparator Trimpot input
- **PD6**: Output for driving the Speaker
- **PD7**: Output for Power LED
- **PB6**: Deep-Sleep Re entry Button

Deep Sleep and Exiting

Deep sleep is a low-power mode in microcontrollers designed to significantly reduce power consumption by disabling most system clocks and peripherals while retaining essential functions. Unlike regular sleep modes, deep sleep shuts down the main clock and only keeps specific low-power clock sources, such as a 32.768 kHz external oscillator or an internal precision oscillator, active. This mode is particularly useful in energy-critical applications where the system needs to remain idle for long periods and only wake up when triggered by an event, such as a timer or an interrupt from a sensor.

During deep sleep, power consumption is minimized by turning off non-essential peripherals while preserving the system's state, including register values and memory contents. Wake-up sources, such as timers, analog comparators, or GPIO interrupts, can bring the microcontroller out of deep sleep, allowing it to quickly resume operation without losing its context. Configuring the microcontroller for deep sleep involves setting the Sleep bit in the system control block, selecting an appropriate clock source for peripherals during sleep, and using the `wfi` in assembly to enter the low-power state.

In this project we employed the above knowledge inorder to create an interrupt when lm35 temperature exceeds that of the trimpot initiating a comparator interrupt and awakening the system using PC6 and PC7 pins as + and - pins respectively to check for the correct environment settings.

Keypad Digital Threshold

The keypad has 4 rows (R1–R4) and 4 columns (C1–C4), where each key press connects a specific row-column pair. By alternating between reading rows and columns, the program determines the pressed key and maps it to an integer value. This value is then used to calculate a threshold temperature by combining the two digits entered by the user.

	COLUMN1	*COLUMN2*	*COLUMN3*	*COLUMN4*
ROW1	F	E	D	C
ROW2	B	A	9	8
ROW3	7	6	5	4
ROW4	3	2	1	0

The program begins with the initialization of GPIO ports. The function port_Init1 configures the rows (PE2–PE5) as inputs and columns (PD0–PD3) as outputs. Pull-up resistors are enabled for the inputs to ensure stable signal detection. The columns are initially set to a low state, allowing the program to detect key presses in the rows. When a key is pressed, the function read_PortC reads the row signals and updates register R7 to reflect the active row. Then port_Init1 reinitialises the ports reversing the roles of rows and columns. In this configuration, columns are set as inputs with pull-up resistors, and rows are set as outputs and store the data in register R1. The program processes these inputs in a sequence. storing the result in register R0. Then, it waits for the user to release the key and returns the integer value. We then Use the DELAY200() in order to prevent debouncing. And do the same for each of the digits required for our precision. As a final note this subroutine is done using Assembly

Re-Entering Deep Sleep

In this project After waking up from deep sleep and performing the necessary steps the program can re enter sleep if given an outside signal to do so by the user this is done using PB6 pin and the GPIO Buttons on our keypad

	S1	*S2*	*S3*	*S4*
GPIO BUTTONS	For Re-entering Deep Sleep Pb6	Unused	Unused	Unused

The program begins with the initialization of the PB6 port and its configuration as an input than it polls continuously and will be checked in a while loop if it is zero ie pressed it will break the for and while blocks and then it will retrace its steps to comeback to its original position before the program will do the necessary steps needed to reconfigure the systems and re-enter deep sleep.

Power LED and Speaker Drivers

After waking from Deep Sleep we activate the Power led to indicate that we have woken up by toggling it and using a power BJT namely TIP 122 to drive it as seen in the Figure 3 and toggling it back when re-entering Deep Sleep.

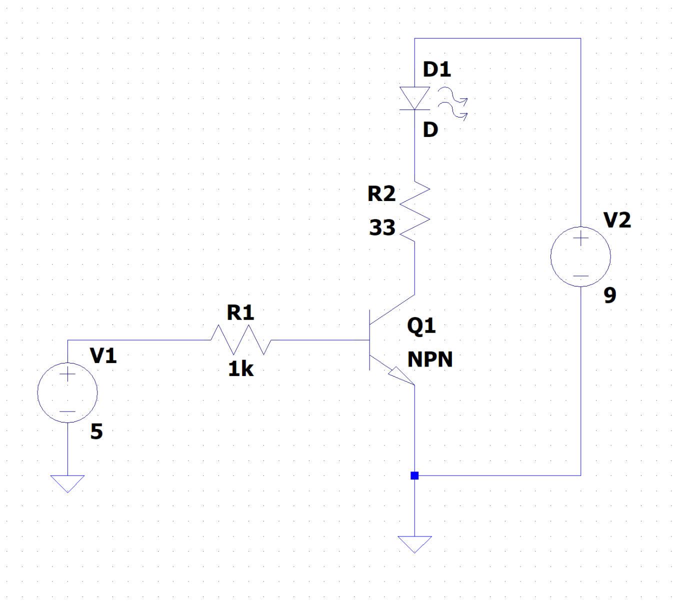

Figure 3. Driving of the Power LED

After confirming with taking the temperature of the environment 128 times taking its average and confirming that it is above the threshold temperature it will start to xor pin unknown which will periodically activate and deactivate the speaker for 2 seconds.. As seen in Figure 4.

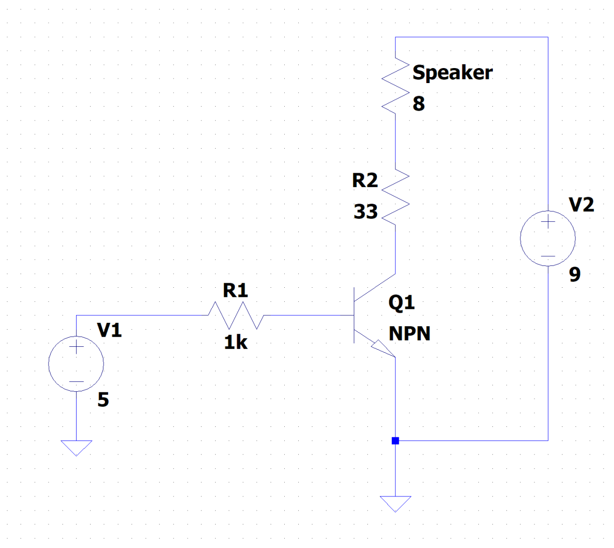

Figure 4. Driving of the Speaker (Buzzer)

-90 to +90 Stepping and distance Measurement

When the system awakens from deep sleep, it initiates a scanning process across an arc ranging from -90° to +90°. This is accomplished using a stepper motor that moves in precise increments of 34 half steps this number is calculated as it takes 4096 half steps for a full 360 degrees rotation, the HC-SR04 ultrasonic distance sensor is used to detect objects within a range of 1 meter, collecting data as it progresses.

During the scan, the system compiles the detected distances into an array. It calculates the total number of objects within the 1-meter range and determines the average distance. Similarly, the angles of the detected objects are recorded, and the system calculates the average angle. This data is processed in real-time to provide meaningful insights.

Once the scan is complete, the results, including the total object count, average distance, and average angle, are displayed on an LCD screen. Simultaneously, RGB LEDs on Tiva are activated for varying distance ranges to provide visual feedback based on the collected data. After displaying the information, the system returns the stepper motor to its original starting position, completing the scanning cycle. If the system receives a command to re-enter deep sleep , it halts all operations and reconfigures itself to transition into a low-power state.

I2C Temperature Reading

Microcontroller Configuration

- **General I2C Knowledge**

I2C (Inter-Integrated Circuit) is a communication protocol that uses two lines—**SCL** (clock) and **SDA** (data)—to enable communication between a master device

(TM4C123GH6PM in this case) and one or more slave devices (BMP280). It supports multiple devices on the same bus by assigning unique 7-bit addresses to each device. The protocol features start and stop conditions, as well as acknowledgment bits to ensure proper communication.

- **Pin and Peripheral Configuration**

To interface with the BMP280, the I2C module on TM4C123GH6PM is configured using the following initialization code:

Enable the clock for the I2C0 module and GPIO Port B.
Wait for Port B to be ready by checking its peripheral ready status.
Configure GPIO pins PB2 and PB3 for I2C:

- Enable alternate functionality for PB2 and PB3.
- Set PB3 (SDA) to open-drain mode.
- Enable digital I/O for PB2 and PB3.
- Update the pin control (PCTL) register to assign I2C functionality to PB2 and PB3.
- Disable analog functionality for PB2 and PB3.

Enable I2C master mode by setting the appropriate bit in the master control register.
Set the I2C clock speed to 100 kHz by configuring the timer period register.

This function prepares the I2C0 module by configuring PB2 and PB3 for I2C communication, enabling open-drain functionality on SDA, and setting the clock speed for the desired I2C frequency (100 kHz).

BMP280 Reading and Configuration

- **Overview of BMP280**

The BMP280 sensor provides temperature and pressure readings with high accuracy. Configuration involves setting the oversampling rates and operational modes to optimize performance. Communication is performed using I2C, with calibration coefficients retrieved from the sensor's internal memory for temperature compensation.

- **BMP280 Configuration**

The BMP280 is configured using the configure_bmp280 function. This sets the desired standby time, filtering, and oversampling settings by writing to the control registers:

```
void configure_bmp280(void) {
   I2C_Send2(0x76, 0xF5, (0x03 << 5));  // Configure t_sb = 250ms, filter = 0
   I2C_Send2(0x76, 0xF4, ((0x02 << 4) | (0x02 << 2) | 0x03));  // Set oversampling and
mode
}
```

This function writes to registers 0xF5 and 0xF4 of the BMP280 to set standby time and oversampling for temperature and pressure measurements.

Calibration data is read from the BMP280 using the following pseudo code of the subroutine:

```
        Define variables msb and lsb to store the received bytes.
        Wait until the I2C module is not busy.
        Set the slave address and enable read mode in the I2C address register.
        Start the I2C communication by sending the start condition and enabling run mode.
        Wait until the I2C module is not busy.
        Read the MSB from the data register.
        Send the stop condition and enable run mode to stop the communication.
        Wait until the stop operation completes.
        Read the LSB from the data register.
        Combine the MSB and LSB into a 16-bit value and return it.
```

This is used in let_there_be_temp to retrieve calibration coefficients dig_T1, dig_T2, and dig_T3.

The temperature compensation is performed using:

```
        Define variables var1 and var2 for intermediate calculations.
        Calculate var1 as the adjusted difference of adc_T and dig_T1, multiplied by dig_T2,
        and shifted right by 11.
        Calculate var2 as the squared adjusted difference of adc_T and dig_T1, multiplied by
        dig_T3, and shifted right by 14.
        Add var1 and var2 to compute the final compensated temperature.
        Return the computed temperature.
```

This formula is derived from the BMP280 datasheet, ensuring high precision in the calculated temperature.

The main function let_there_be_temp handles temperature measurement, compensation, and averaging:

```
Define variables for calibration coefficients (dig_T1, dig_T2, dig_T3), raw data (adc_T), and
temperature calculations.
Initialize the I2C module.
Configure the BMP280 sensor.
Set the pointer to register 0x88 and read dig_T1.
Set the pointer to register 0x8A and read dig_T2.
Set the pointer to register 0x8C and read dig_T3.
Loop through the number of readings (NUM_READINGS):
```

- Set the pointer to register 0xFA and read the MSB of the raw temperature.
- Set the pointer to register 0xFB and read the LSB of the raw temperature.
- Set the pointer to register 0xFC and read the XLSB of the raw temperature.
- Combine the MSB, LSB, and XLSB to form the 20-bit raw temperature (adc_T).
- Compensate the temperature using adc_T and the calibration coefficients.
- Store the compensated temperature in an array.

Sum all temperature readings in the array.
Calculate the average temperature by dividing the sum by NUM_READINGS.
Return the average temperature.

This function handles raw data acquisition and compensation using the BMP280's calibration coefficients. The averaging ensures stability in temperature readings.

LCD Screen Printing

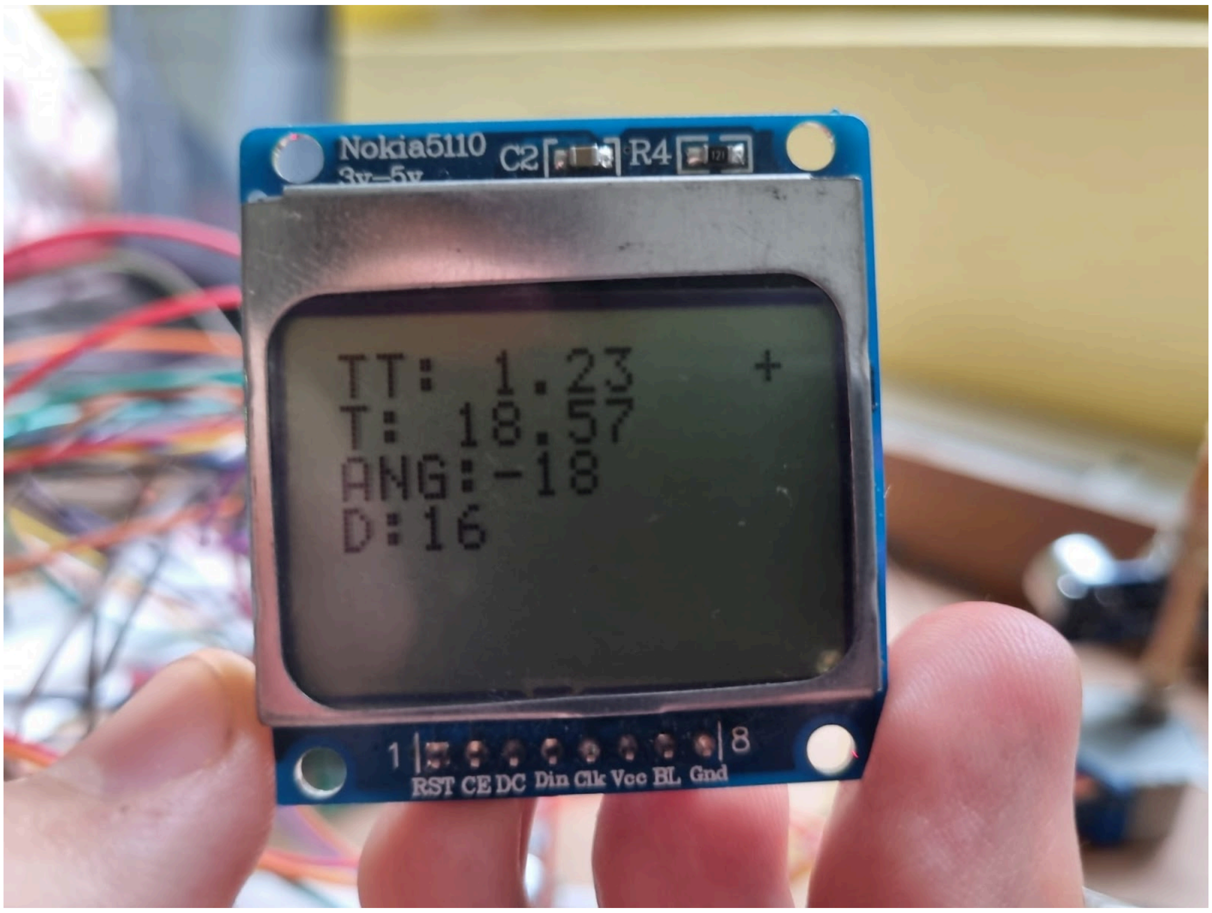

Figure 5. LCD Screen

General SPI Knowledge

The Nokia 5110 LCD uses the SPI protocol for communication. SPI (Serial Peripheral Interface) is a full-duplex synchronous communication protocol that uses four main signals:

- **SCLK** (Serial Clock): Provides the clock signal.
- **MOSI** (Master Out Slave In): Sends data from the master (TM4C123GH6PM) to the slave (LCD).
- **CS/SS** (Chip Select): Selects the active slave device.
- **DC** (Data/Command): Differentiates between commands and data being sent to the LCD.

SPI is preferred for high-speed communication with devices like displays.

Pin Configuration

The microcontroller pins are configured for digital I/O and alternate functions to support SPI. Below is the initialization pseudo code for SPI communication:

```
Enable the SSI0 module by setting the relevant bit in the SSI clock control register.
Enable Port A by setting the corresponding bit in the GPIO clock control register.
Wait for the system to stabilize by reading the GPIO clock control register.
Set PA6 and PA7 as output pins in the GPIO direction register.
Enable alternate functions on PA2, PA3, and PA5 by configuring the GPIO alternate function select register.
Enable digital I/O on PA2, PA3, PA5, PA6, and PA7 by configuring the GPIO digital enable register.
Configure the GPIO port control register to set PA2, PA3, and PA5 for SSI functionality.
Disable analog functionality on PA2, PA3, PA5, PA6, and PA7 by clearing bits in the GPIO analog mode select register.
Disable the SSI module by clearing the SSI enable bit in the SSI control register.
Set the SSI module to master mode by clearing the master/slave bit in the SSI control register.
Configure the SSI clock source to use the system clock by writing to the SSI clock configuration register.
Set the SSI clock prescale divisor to 24 by configuring the SSI clock prescale register.
Configure the SSI mode for 8-bit data, Freescale SPI frame format, and specific clock phase settings by writing to the SSI control register.
Enable the SSI module by setting the SSI enable bit in the SSI control register.
Set the RESET pin to low to reset the LCD.
Wait for a short delay by looping a few iterations.
Set the RESET pin to high to release the LCD reset.
Send initialization commands to the LCD:

    ● Enable the extended instruction set.
    ● Set the contrast level.
    ● Configure the temperature coefficient.
    ● Set the bias system.
    ● Switch back to the basic instruction set.
    ● Enable normal display mode.
```

Function: let_there_be_tt

Purpose:
This function displays the temperature value TT (as a double) on the Nokia 5110 LCD. It formats the double value into its integer and fractional components, then prints them as a string on the screen.

Pseudo Code:

```
Initialize the system clock to 50 MHz using the PLL.
Initialize the Nokia 5110 LCD.
Extract the integer part of TT by casting it to an integer.
Extract the fractional part of TT by subtracting the integer part from TT, multiplying by 100, and casting the result to an integer.
Set the cursor to position (0, 0) on the LCD.
Display the label "TT:" on the LCD.
Move the cursor to position (4, 0) to prepare for the value.
Display the integer part of TT on the LCD.
Display a decimal point on the LCD.
Display the fractional part of TT on the LCD.
```

Key Points:

1. **Integer and Fractional Separation**:
   - The integer and fractional parts of TT are separated using type casting and arithmetic.
2. **LCD Display**:
   - The cursor positions are set precisely using Nokia5110_SetCursor.
   - Nokia5110_OutString and Nokia5110_OutUDec are used to print text and numbers.

Function: let_there_be_screen

Purpose:
This function displays various parameters on the Nokia 5110 LCD, including:

- TT: Total temperature (double)
- T: Current temperature (double)
- ANG: Angle (uint16_t)
- D: Distance (uint16_t)
- A sign indicator (+ or -) depending on the value of o_exs.

The function handles formatting and conditional display logic for angle adjustments.

Pseudo Code:

Initialize the system clock to 50 MHz using the PLL.
Initialize the Nokia 5110 LCD.
Extract the integer and fractional parts of TT.
Extract the integer and fractional parts of T.
Check if o_exs is set:

- If o_exs is 1:
    - Adjust ANG by subtracting or calculating the absolute difference with 90.
    - Clear the LCD and display a "+" sign at cursor position (11, 0).
    - Set the cursor to (0, 0) and display "TT:", followed by the integer and fractional parts of TT.
    - Set the cursor to (0, 1) and display "T:", followed by the integer and fractional parts of T.
    - Set the cursor to (0, 2) and display "ANG:", followed by the adjusted ANG.
    - Set the cursor to (0, 3) and display "D:", followed by the value of D.
- If o_exs is not set:
    - Clear the LCD and display a "-" sign at cursor position (11, 0).
    - Set the cursor to (0, 0) and display "TT:", followed by the integer and fractional parts of TT.
    - Set the cursor to (0, 1) and display "T:", followed by the integer and fractional parts of T.
    - Set the cursor to (0, 2) and display "ANG:", followed by "?".
    - Set the cursor to (0, 3) and display "D:", followed by "?".

Key Points:

1. **Conditional Display**:
    - Based on o_exs, the function chooses different displays.
    - Adjustments to ANG are made conditionally.
2. **Modular Display Logic**:
    - Similar to let_there_be_tt, integer and fractional components are separated and displayed.
    - Nokia5110_Clear ensures the screen is refreshed before displaying new data.

Sub Functions Used

Nokia 5110 SetCursor

Positions the cursor on the LCD at (x, y):

> Check if x is greater than 11 or y is greater than 5; if true, exit the function.
> Calculate the X position by multiplying x by 7 and writing it to the LCD as a command with 0x80 ORed with the result.
> Write the Y position to the LCD as a command with 0x40 ORed with y.

Nokia5110_OutString

Prints a string of characters:
While the character pointed to by ptr is not null:

- Output the current character using Nokia5110_OutChar.
- Move to the next character by incrementing ptr.

Nokia5110_OutUDec

Prints an unsigned decimal number:

If n is less than 10, output n as a single character by adding '0' and passing it to Nokia5110_OutChar.
If n is less than 100:

- Output the tens digit by dividing n by 10, adding '0', and passing it to Nokia5110_OutChar.
- Output the ones digit by taking n % 10, adding '0', and passing it to Nokia5110_OutChar.

If n is less than 1000:

- Output the hundreds digit by dividing n by 100, adding '0', and passing it to Nokia5110_OutChar.
- Output the tens digit by taking (n % 100) / 10, adding '0', and passing it to Nokia5110_OutChar.
- Output the ones digit by taking n % 10, adding '0', and passing it to Nokia5110_OutChar.

Otherwise (for n >= 1000):

- Output the thousands digit by dividing n by 1000, adding '0', and passing it to Nokia5110_OutChar.
- Output the hundreds digit by taking (n % 1000) / 100, adding '0', and passing it to Nokia5110_OutChar.
- Output the tens digit by taking (n % 100) / 10, adding '0', and passing it to Nokia5110_OutChar.
- Output the ones digit by taking n % 10, adding '0', and passing it to Nokia5110_OutChar.

These functions provide a versatile framework for displaying dynamic data on the Nokia 5110 LCD, with modularity and clarity for easy customization.

Conclusion

This project effectively showcases a power-efficient and responsive temperature monitoring system using the TM4C123GH6PM microcontroller. By integrating analog sensors with ADC modules and leveraging deep sleep capabilities, the system achieves minimal energy consumption while ensuring reliable performance. The use of interrupts allows for timely responses to temperature fluctuations, enhancing overall system efficiency.

The implementation of I2C communication for interfacing with the BMP280 sensor enabled accurate temperature measurement, highlighting the utility of integrating precise digital sensors in embedded applications. The BMP280's configuration and data acquisition processes demonstrated the seamless integration of real-world sensors through I2C, reinforcing concepts of communication protocols and calibration techniques.

Additionally, the integration of the Nokia 5110 LCD using SPI communication added a dynamic output interface, allowing real-time display of critical parameters such as temperature and angle. The process of designing and implementing functions for data formatting and display strengthened the understanding of SPI and practical LCD usage.

This project also revisited GPIO and Timer configurations, deepening the understanding of their practical applications. By combining efficient communication protocols, power management strategies, and modular code development, this project highlights the versatility of low-power embedded systems in applications involving multiple input and output devices. The hands-on experience serves as a comprehensive exercise in embedded system design, bridging theoretical concepts with real-world implementations.

References

[1]TI, “Tiva™ tm4c123gh6pm microcontroller data sheet.” http://www.ti.com/lit/ds/spms376e/spms376e.pdf.
[2]J.Valvano,“Starter_files_for_embedded_systems.”https://users.ece.utexas.edu/%7Evalvano/arm/.