HMC5883L to TM4C123G

Oct 31, 2023

Andrew Lockwood

Anton Salah

The development process of interfacing the HMC5883L 3-Axis Digital Compass with the Tiva TM4C123G LaunchPad involved several steps, both in understanding the underlying hardware protocols and in implementing the appropriate software routines. Below is a concise technical outline of the process: Understanding I2C Communication

- I2C Protocol: A communication protocol that uses two lines (SDA for data, SCL for clock) to transfer data between devices.
- Addressing: Each I2C device has a unique address; for the HMC5883L, it's typically `0x1E`.

Compass Configuration and Operation

- Magnetic Measurement: The HMC5883L measures the Earth's magnetic field in three axes (X, Y, and Z), which can be used to determine the heading.
- Registers: Configuration and measurement data are read from/written to the compass's internal registers.

Tiva TM4C123G Setup

- Clocks: The system clock is set to run from the crystal oscillator with PLL enabled for precise timing required for I2C operation.
- GPIO Configuration: The appropriate GPIO ports are initialized for I2C communication with alternate function configuration.

I2C Initialization and Configuration

- Peripheral Enable: The I2C1 and GPIOA peripherals are enabled.
- Pin Configuration: GPIO pins are configured for I2C SCL and SDA functionalities.
- Master Initialization: The I2C master module is initialized with the system clock and standard mode (100kbps).

Compass Initialization

• Slave Address: The I2C slave address for the compass is set before any communication. • Register Configuration: Configuration registers in the compass are set up for continuous measurement mode.

Data Reading Routine

- Register Selection: The data output register's address is sent before reading the measurement data.

- Burst Read: A burst read sequence is initiated to read multiple bytes from the compass's data registers.
- Error Handling: Errors during I2C communication are checked and reported.

Main Loop

- Calibration: An offset calibration is applied to the raw measurements to account for any systematic bias.
- Heading Calculation: The heading is calculated from the X and Y measurements using arctan function, adjusted to a 0-359 degree range.
- Display: The heading and raw data are printed to the UART for debugging and monitoring.

Code Snippet for I2C and Compass Setup

Code:

```
// Code to initialize I2C peripherals and configure pins
void InitializeI2C(void){
 UARTprintf("\nInitializing I2C...\n");
 // Enable the peripherals for I2C1 and GPIOA
 SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);
 SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

 // Wait for the peripherals to be ready
 while(!SysCtlPeripheralReady(SYSCTL_PERIPH_I2C1) || !SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA));
UARTprintf("Peripherals ready...\n");

 // Corrected pin configuration for I2C1
 GPIOPinConfigure(GPIO_PA6_I2C1SCL);
 GPIOPinConfigure(GPIO_PA7_I2C1SDA);
 GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_7);
 GPIOPinTypeI2CSCL(GPIO_PORTA_BASE, GPIO_PIN_6);

 // Initialize and configure the I2C1 master module
 I2CMasterInitExpClk(I2C1_BASE, SysCtlClockGet(), false); // false = standard mode (100kbps)
UARTprintf("I2C Initialization Complete...\n");
}

// Code to configure compass registers for measurement
void ConfigureCompass(void){

 // Set the slave address and specify that the next operation is a write
 I2CMasterSlaveAddrSet(I2C1_BASE, COMPASS_I2C_ADDRESS, false);

 // Write to Configuration Register A
 I2CMasterDataPut(I2C1_BASE, 0x00); // Address of Configuration Register A
 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);
 while(I2CMasterBusy(I2C1_BASE));
 I2CMasterDataPut(I2C1_BASE, 0x70); // Configuration data
 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
 while(I2CMasterBusy(I2C1_BASE));

 // Write to Configuration Register B
 I2CMasterDataPut(I2C1_BASE, 0x01); // Address of Configuration Register B
 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);
 while(I2CMasterBusy(I2C1_BASE));
```

```
 I2CMasterDataPut(I2C1_BASE, 0x20); // Configuration data
```



```
 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
while(I2CMasterBusy(I2C1_BASE));

 // Write to Mode Register
 I2CMasterDataPut(I2C1_BASE, 0x02); // Address of Mode Register
I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);
while(I2CMasterBusy(I2C1_BASE));
 I2CMasterDataPut(I2C1_BASE, 0x00); // Configuration data
I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
while(I2CMasterBusy(I2C1_BASE));
}

// Code to read compass data from the compass registers
void ReadCompass(int16_t *x, int16_t *y, int16_t *z){
 uint8_t data[TOTAL_DATA_BYTES];
 int index = 0;

 // Set the slave address and specify that the next operation is a write
I2CMasterSlaveAddrSet(I2C1_BASE, COMPASS_I2C_ADDRESS, false);

 // Place the register to be read into the data register
I2CMasterDataPut(I2C1_BASE, COMPASS_REGISTER);

 // Perform a single send operation to the slave
 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);

 // Wait for the send operation to complete
 while(I2CMasterBusy(I2C1_BASE));

 // Check for errors after writing to the register
 if(I2CMasterErr(I2C1_BASE) != I2C_MASTER_ERR_NONE){
 UARTprintf("I2C Error: %u\n", I2CMasterErr(I2C1_BASE));
return;
 }

 // Set the slave address and specify that the next operation is a read
I2CMasterSlaveAddrSet(I2C1_BASE, COMPASS_I2C_ADDRESS, true);

 // Perform a burst receive, starting with the first byte
I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);

 // Wait for the receive operation to complete
 while(I2CMasterBusy(I2C1_BASE));

 // Read the first byte
 data[index++] = I2CMasterDataGet(I2C1_BASE);

 // Read the middle bytes
 for(; index < TOTAL_DATA_BYTES - 1; ++index){
 // Continue the burst receive
 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT);
while(I2CMasterBusy(I2C1_BASE));
 data[index] = I2CMasterDataGet(I2C1_BASE);
 }
```

```
 // Finish the burst receive
 I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH); 3



 while(I2CMasterBusy(I2C1_BASE));
 data[index] = I2CMasterDataGet(I2C1_BASE);

 // Check for errors after reading the data
 if(I2CMasterErr(I2C1_BASE) != I2C_MASTER_ERR_NONE){
 UARTprintf("I2C Error: %u\n", I2CMasterErr(I2C1_BASE));
return;
 }

 // Assume the data is in the format X MSB, X LSB, Z MSB, Z LSB, Y MSB, Y LSB
*x = (data[0] << 8) | data[1];
 *z = (data[2] << 8) | data[3];
 *y = (data[4] << 8) | data[5];

}

// main function
int main(void){


 SetClocks();
 ConfigureUART();
 InitializeI2C();
 ConfigureCompass();
 InitializePins();
 MAP_FPULazyStackingEnable();

 UARTprintf("\nTesting the HMC5883L Compass response...\n");

 //IMPORTANT OFFSET WILL CALIBRATE THE COMPASS. LEADING TO BETTER ACCURACY
const int x_offset = 0;
 const int y_offset = 0;

 while(1) {
 ProofOfLifeLEDsToggle();

 int16_t x, y, z;
 ReadCompass(&x, &y, &z); // Read data from the compass

//adjust the x and y values

 UARTprintf("Raw Compass Data: X: %d, Y: %d \n", x,y);

 x+= x_offset;
 y+= y_offset;

 // Calculate heading in degrees
 int heading = i_atan2(x, y);

 // Normalize the heading to 0-359 degrees
 if (heading < 0) {
 heading += 360;
 }
```

```
 PrintDirection(heading);

 // Print the heading to UART
```



```
 UARTprintf("Heading: %d degrees\n", heading);

 // Print the received data to UART
 UARTprintf("Compass Data: X: %d, Y: %d, Z: %d\n", x, y, z);


 SysCtlDelay(100 * one_milli_sec); // delay 109 ms

 }
}// fin
 ///////////////////////////////////////////////////////////////////////////////////////////
```

Calibration

To ensure accurate and consistent compass readings from the HMC5883L module, calibration is essential. Each compass module has unique intrinsic characteristics, and external disturbances in its environment can introduce systematic biases in its measurements. Calibration identifies and compensates for these biases to provide accurate heading data.

Performing the Calibration Test

1. Run the provided TM4C123G microcontroller code with ‘**x_offset**’ and ‘**y_offset**’ variables set to zero.
2. Using UART, print data to the terminal
3. Slowly rotate the compass module 360 degrees over a period ranging from ten seconds to a minute. This will print the Raw Data to the terminal.
4. Use the printed Raw Data to calculate the offsets for x and y.

Calculating the X and Y offsets

To calculate the offsets:

1. Average the values for x and y respectively and divide by the number of data points.
2. Subtract the averaged number from zero to get the offset.

Mathematically this is represented as:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \qquad \bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$$

$$offset_x = 0 - round(\bar{x}) \qquad offset_y = 0 - round(\bar{y})$$

Implementation

After determining the offsets, update the ‘**x_offset**’ and ‘**y_offset**’ variables in the main code.

Code Snippet For Visualizing and Calculating the offset values

Use the Python script below. It reads values from “data.txt” and displays the data graphically. In “raw” mode, it calculates x and y offsets. In “calibrated” mode, it displays data with color-coded directions

```
////////////////////////////////////////////////////////////////////////////////////////////
import re
import matplotlib.pyplot as plt

# Lists to store the extracted data
headings = []
x_values = []
y_values = []
colors = []

# Color map for headings
color_map = {
    'North (N)': 'red',
    'Northeast (NE)': 'orange',
    'East (E)': 'yellow',
    'Southeast (SE)': 'lime',
    'South (S)': 'green',
    'Southwest (SW)': 'cyan',
    'West (W)': 'blue',
    'Northwest (NW)': 'purple',
    'Unknown': 'black'
}

mode = input("Enter mode (raw or calibrated): ").strip().lower()

# Define regex patterns based on mode
if mode == "raw":
    pattern = r"Raw Compass Data: X: (-?\d+), Y: (-?\d+)"
else:
```

```python
    pattern = r"Heading: (\d+) degrees\nCompass Data: X: (-?\d+), Y: (-?\d+), Z: (-?\d+)"

# Read from the data.txt file
with open("data.txt", "r") as f:
    data = f.read()

    # Extract values using regex
    for match in re.finditer(pattern, data):
        if mode == "raw":
            x_values.append(int(match.group(1)))
            y_values.append(int(match.group(2)))
            colors.append('gray')  # Default color for raw data modec
        else:
            headings.append(int(match.group(1)))
            x_values.append(int(match.group(2)))
            y_values.append(int(match.group(3)))

            # Determine direction based on heading and append the color to the colors list
            heading_val = int(match.group(1))
            if 338 <= heading_val or heading_val < 23:
                colors.append(color_map['North (N)'])
            elif 23 <= heading_val < 68:
                colors.append(color_map['Northeast (NE)'])
            elif 68 <= heading_val < 113:
                colors.append(color_map['East (E)'])
            elif 113 <= heading_val < 158:
                colors.append(color_map['Southeast (SE)'])
            elif 158 <= heading_val < 203:
                colors.append(color_map['South (S)'])
            elif 203 <= heading_val < 248:
                colors.append(color_map['Southwest (SW)'])
            elif 248 <= heading_val < 293:
                colors.append(color_map['West (W)'])
            elif 293 <= heading_val < 338:
                colors.append(color_map['Northwest (NW)'])
            else:
                colors.append(color_map['Unknown'])

# Calculate and display suggested offsets for raw mode
if mode == "raw":
    average_x = sum(x_values) / len(x_values)
    average_y = sum(y_values) / len(y_values)

    offset_x = 0 - round(average_x)
    offset_y = 0 - round(average_y)

    print(f"Suggested X offset: {offset_x}")
    print(f"Suggested Y offset: {offset_y}")

# Plotting
plt.figure(figsize=(10,10))
plt.scatter(x_values, y_values, c=colors, s=10)  # s is size of point
plt.title('Compass Data Visualization - ' + mode.capitalize() + ' Mode')
```

```
plt.xlabel('X values')
plt.ylabel('Y values')
plt.grid(True)
plt.axhline(0, color='black',linewidth=0.5)
plt.axvline(0, color='black',linewidth=0.5)
if mode == "calibrated":
    plt.legend(handles=[plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=color, markersize=10) for color in color_map.values()], labels=color_map.keys())
plt.tight_layout()
plt.show()
```

//

Conclusion

Implementation of the compass interfacing required an understanding of the I2C protocol, precise timing and clock management, and careful register configuration. The provided code integrates these elements to create a reliable system for obtaining heading information from the HMC5883L compass module.

Additional Notes

- Calibration: To achieve accurate heading information, calibration of the compass against known references is crucial.
- Timing: I2C communication and compass measurement cycle times must be considered to ensure data integrity and synchronization.
- Debugging: The use of UART for debugging allowed for real-time monitoring and troubleshooting during development.

This summary encapsulates the setup and configuration process for the HMC5883L compass module interfaced with the Tiva TM4C123G microcontroller, highlighting important elements in both hardware and software that are essential for communication with the HMC5883L.