Interfacing MPU with Pico for I2C in C SDK

Aryamman Bhatia-IITBSSP

September 2024

1 Intro, Data from Datasheets

Default I2C communication address of MPU - 0x68 - I2C works on master(pico) and slave(MPU). Therefore pico sends commands to the mpis mpu requesting to read and write on/from certain registers.

PWR_MGMT_1 - 0x6B register - Writing 0x00 on this wakes up the MPU from sleep mode.

ACCEL_XOUT_H - 0x3B register - pico reads Accelerometer data from here onwards

GYRO_XOUT_H - 0x43 register - pico reads gyro data from here onwards

TEMP_OUT_H - 0x41 register - pico reads temperature data from here onwards

Temperature sensitivity - 340

Temperature offset - 35

2 Code Explanation

2.1 Function: `mpu6050_init`

The function `mpu6050_init` initializes the MPU6050 sensor by configuring its power management register. This ensures that the MPU6050 is active and ready to provide sensor data.

Function Definition:

```
void mpu6050_init() {
    uint8_t data = 0x00;
    // Wake up the MPU6050
    i2c_write_blocking(i2c0, MPU6050_ADDR,
    (uint8_t[]){PWR_MGMT_1, data}, 2, false);
}
```

Parameters:

- `i2c0`: The I2C peripheral instance used for communication. For the Raspberry Pi Pico, this is the first I2C hardware interface.

- `MPU6050_ADDR`: The I2C address of the MPU6050 sensor, used to identify the device on the I2C bus.
- `PWR_MGMT_1`: The register address for power management settings in the MPU6050.
- `data`: A variable holding the value to be written to the `PWR_MGMT_1` register. In this function, `data` is set to `0x00`.

Operation:

1. The line `uint8_t data = 0x00;` initializes a variable `data` with the hexadecimal value `0x00`.
2. The line `i2c_write_blocking(i2c0, MPU6050_ADDR, (uint8_t[])PWR_MGMT_1, data, 2, false);` is a function in the hardware library of pico that performs an I2C write operation. It sends a 2-byte command to the address of the MPU6050 to write to register:
   - The first byte, `PWR_MGMT_1`, is the register address that specifies where the data is to be written.
   - The second byte, `data`, is set to `0x00` to configure the `PWR_MGMT_1` register. Writing `0x00` wakes the MPU6050 from sleep mode and sets it to use the internal 8 MHz clock.
3. The `false` parameter specifies that the I2C operation should not use a repeated start condition, indicating that the transaction is complete.

This function is crucial for ensuring that the MPU6050 sensor is properly initialized and ready to be used for further operations.

2.2 Function Definition: read_mp6050_data

```
void read_mpu6050_data(int16_t *ax, int16_t *ay, int16_t *az,
int16_t *gx,int16_t *gy, int16_t *gz, float *temp) {
    uint8_t buf[14];

    // Read accelerometer, gyroscope, and temperature data
    i2c_write_blocking(i2c0, MPU6050_ADDR,
    (uint8_t[]){ACCEL_XOUT_H}, 1, true);
    i2c_read_blocking(i2c0, MPU6050_ADDR, buf, 14, false);

    *ax = (int16_t)((buf[0] << 8) | buf[1]);
    *ay = (int16_t)((buf[2] << 8) | buf[3]);
    *az = (int16_t)((buf[4] << 8) | buf[5]);
    *gx = (int16_t)((buf[8] << 8) | buf[9]);
    *gy = (int16_t)((buf[10] << 8) | buf[11]);
    *gz = (int16_t)((buf[12] << 8) | buf[13]);
    // Temperature in degrees Celsius
    int16_t raw_temp = (int16_t)((buf[6] << 8) | buf[7]);
    *temp = (raw_temp / 340.0f) + 35.0f;
}
```

Explanation:

- **I2C Write Operation:** The call `i2c_write_blocking(i2c0, MPU6050_ADDR, (uint8_t[])ACCEL_XOUT_H, 1, true)` writes the register address `ACCEL_XOUT_H` to the MPU6050 sensor. The parameters are:
  - `i2c0`: I2C instance used for communication.
  - `MPU6050_ADDR`: I2C address of the MPU6050 sensor.
  - `(uint8_t[])ACCEL_XOUT_H`: Register address to start reading from.
  - `1`: Number of bytes to write (only the register address).
  - `true`: Indicates a repeated start condition, allowing an immediate read operation.
- **I2C Read Operation:** The call `i2c_read_blocking(i2c0, MPU6050_ADDR, buf, 14, false)` reads 14 bytes from the MPU6050 sensor into the buffer `buf`. The parameters are:
  - `i2c0`: I2C instance used for communication.
  - `MPU6050_ADDR`: I2C address of the MPU6050 sensor.
  - `buf`: Buffer to store the read bytes.
  - `14`: Number of bytes to read (encompassing accelerometer, gyroscope, and temperature data).
  - `false`: Indicates that no repeated start condition is used, ending the I2C transaction.
- **Data Extraction and Interpretation:** The 14 bytes read are organized as follows(can be verified by addresses of registers keeping in mind hexadecimal system.:
  - `buf[0] and buf[1]`: High and low bytes of the accelerometer X-axis data.
  - `buf[2] and buf[3]`: High and low bytes of the accelerometer Y-axis data.
  - `buf[4] and buf[5]`: High and low bytes of the accelerometer Z-axis data.
  - `buf[6] and buf[7]`: High and low bytes of the temperature data.
  - `buf[8] and buf[9]`: High and low bytes of the gyroscope X-axis data.

  - `buf[10]` and `buf[11]`: High and low bytes of the gyroscope Y-axis data.
  - `buf[12]` and `buf[13]`: High and low bytes of the gyroscope Z-axis data.

- **Combining Bytes into 16-bit Values:** The operation `<< 8` shifts the high byte to the left by 8 bits, placing it in the upper byte of a 16-bit integer:

  ```
  *ax = (int16_t)((buf[0] << 8) | buf[1]);
  ```

  This combines the high byte (shifted left) with the low byte to form the full 16-bit value.

- **Temperature Calculation:** The temperature is calculated using:

  ```
  *temp = (raw_temp / 340.0f) + 35.0f;
  ```

  - `raw_temp / 340.0f`: Converts the raw temperature data to degrees Celsius.
  - `+ 35.0f`: Adjusts the temperature based on the MPU6050's calibration.

2.3 Function: main

```
int main() {
    stdio_init_all();
```

Initialization:

- `stdio_init_all()` initializes the standard I/O for the program.
- `i2c_init(i2c0, 100000);` initializes the I2C0 peripheral with a clock speed of 100 kHz.
- `gpio_set_function(4, GPIO_FUNC_I2C);` and `gpio_set_function(5, GPIO_FUNC_I2C);` configure GPIO pins 4 and 5 for I2C SDA and SCL lines, respectively.
- `gpio_pull_up(4);` and `gpio_pull_up(5);` enable pull-up resistors on the I2C lines to ensure proper signal levels.

Sensor Initialization:

```
    mpu6050_init();
```

Data Reading and Processing Loop:

```
while (true) {
    int16_t ax, ay, az;
    int16_t gx, gy, gz;
    float temp;

    read_mpu6050_data(&ax, &ay, &az, &gx, &gy, &gz, &temp);
```

- The `read_mpu6050_data` function reads raw data from the MPU6050 sensor for accelerometer (`ax`, `ay`, `az`), gyroscope (`gx`, `gy`, `gz`), and temperature (`temp`).

Normalization:

```
    float ax_g = ax / ACCEL_SENSITIVITY;
    float ay_g = ay / ACCEL_SENSITIVITY;
    float az_g = az / ACCEL_SENSITIVITY;
    float gx_dps = gx / GYRO_SENSITIVITY_250;
    float gy_dps = gy / GYRO_SENSITIVITY_250;
    float gz_dps = gz / GYRO_SENSITIVITY_250;
```

- The raw accelerometer values (`ax`, `ay`, `az`) are divided by `ACCEL_SENSITIVITY` to convert them to units of g (gravitational force).
- The raw gyroscope values (`gx`, `gy`, `gz`) are divided by `GYRO_SENSITIVITY_250` to convert them to units of degrees per second (°/s).

Output:

```
    printf("Accel: X=%.2f g, Y=%.2f g, Z=%.2f
    g\n", ax_g, ay_g, az_g);
    printf("Gyro: X=%.2f deg/s, Y=%.2f deg/s,
    Z=%.2f deg/s\n", gx_dps, gy_dps, gz_dps);
    printf("Temperature: %.2f °C\n", temp);
```

- The accelerometer data is printed in g (gravitational force).
- The gyroscope data is printed in degrees per second (°/s).
- The temperature data is printed in degrees Celsius (°C).

Delay:

```
    sleep_ms(1000);
}
```

- The program pauses for 1 second before repeating the loop, allowing time for each data read to be processed and displayed.

2.4 How to get acceleration and gyroscope sensitivities?

To properly interpret the raw data from the MPU6050 sensor, it is essential to understand and calculate the sensitivity settings for both the accelerometer and gyroscope. This is achieved by reading specific configuration registers of the sensor. The following sections explain how to obtain and calculate these sensitivity values.

Accelerometer Sensitivity

The sensitivity of the accelerometer is determined by the setting in the `ACCEL_CONFIG` register (address `0x1C`). This register configures the full-scale range of the accelerometer, which affects the sensitivity.

- **Register Address:** `0x1C`
- **Register Bits:** Bits 1-0 (FS_ACCEL)

The accelerometer's sensitivity is calculated based on the full-scale range set in the `ACCEL_CONFIG` register:

- `00` : ±2g, sensitivity is **16384 LSB/g**
- `01` : ±4g, sensitivity is **8192 LSB/g**
- `10` : ±8g, sensitivity is **4096 LSB/g**
- `11` : ±16g, sensitivity is **2048 LSB/g**

To calculate the sensitivity, you first read the `ACCEL_CONFIG` register and extract the relevant bits. For example:

```
uint8_t accel_config = read_register(0x1C);
// Read the ACCEL_CONFIG register
uint8_t fs_accel = accel_config & 0x03;
// Extract the FS_ACCEL bits

float accel_sensitivity;
switch (fs_accel) {
    case 0x00: accel_sensitivity = 16384.0; break;
    case 0x01: accel_sensitivity = 8192.0; break;
    case 0x02: accel_sensitivity = 4096.0; break;
    case 0x03: accel_sensitivity = 2048.0; break;
}
```

Gyroscope Sensitivity

The sensitivity of the gyroscope is determined by the setting in the `GYRO_CONFIG` register (address `0x1B`). This register configures the full-scale range of the gyroscope, which affects its sensitivity.

- **Register Address:** `0x1B`

- **Register Bits:** Bits 1-0 (FS_GYRO)

The gyroscope's sensitivity is calculated based on the full-scale range set in the `GYRO_CONFIG` register:

- `00` : $\pm 250°/s$, sensitivity is **131 LSB/°/s**
- `01` : $\pm 500°/s$, sensitivity is **65.5 LSB/°/s**
- `10` : $\pm 1000°/s$, sensitivity is **32.8 LSB/°/s**
- `11` : $\pm 2000°/s$, sensitivity is **16.4 LSB/°/s**

To calculate the sensitivity, you first read the `GYRO_CONFIG` register and extract the relevant bits. For example:

```
uint8_t gyro_config = read_register(0x1B);
// Read the GYRO_CONFIG register
uint8_t fs_gyro = (gyro_config >> 3) & 0x03;
// Extract the FS_GYRO bits

float gyro_sensitivity;
switch (fs_gyro) {
    case 0x00: gyro_sensitivity = 131.0; break;
    case 0x01: gyro_sensitivity = 65.5; break;
    case 0x02: gyro_sensitivity = 32.8; break;
    case 0x03: gyro_sensitivity = 16.4; break;
}
```

Summary
To determine the sensitivity settings for the accelerometer and gyroscope:

- Read the `ACCEL_CONFIG` register to determine the accelerometer's full-scale range and its corresponding sensitivity.
- Read the `GYRO_CONFIG` register to determine the gyroscope's full-scale range and its corresponding sensitivity.
- Use the sensitivity values to normalize the raw sensor data.

Properly calculating these sensitivities allows for accurate conversion of raw sensor readings into meaningful units.

3 Entire code

```
#include "pico/stdlib.h"
#include "hardware/i2c.h"
```

```
// MPU6050 I2C Address
#define MPU6050_ADDR 0x68

// MPU6050 Registers
#define PWR_MGMT_1 0x6B
#define ACCEL_XOUT_H 0x3B
#define GYRO_XOUT_H 0x43
#define TEMP_OUT_H 0x41

// Sensitivity factors
#define ACCEL_SENSITIVITY 16384.0f
#define GYRO_SENSITIVITY_250 131.0f

// Function to initialize MPU6050
void mpu6050_init() {
    uint8_t data = 0x00;
    // Wake up the MPU6050
    i2c_write_blocking(i2c0, MPU6050_ADDR, (uint8_t[]){PWR_MGMT_1, data}, 2, false);
}

// Function to read raw data from MPU6050
void read_mpu6050_data(int16_t *ax, int16_t *ay, int16_t *az, int16_t *gx, int16_t *gy, int1
    uint8_t buf[14];

    // Read accelerometer, gyroscope, and temperature data
    i2c_write_blocking(i2c0, MPU6050_ADDR, (uint8_t[]){ACCEL_XOUT_H}, 1, true);
    i2c_read_blocking(i2c0, MPU6050_ADDR, buf, 14, false);

    *ax = (int16_t)((buf[0] << 8) | buf[1]);
    *ay = (int16_t)((buf[2] << 8) | buf[3]);
    *az = (int16_t)((buf[4] << 8) | buf[5]);
    *gx = (int16_t)((buf[8] << 8) | buf[9]);
    *gy = (int16_t)((buf[10] << 8) | buf[11]);
    *gz = (int16_t)((buf[12] << 8) | buf[13]);

    // Temperature in degrees Celsius
    int16_t raw_temp = (int16_t)((buf[6] << 8) | buf[7]);
    *temp = (raw_temp / 340.0f) + 35.0f;
}

int main() {
    stdio_init_all();

    // Initialize I2C with 100kHz
    i2c_init(i2c0, 100000);
    gpio_set_function(4, GPIO_FUNC_I2C); // SDA
    gpio_set_function(5, GPIO_FUNC_I2C); // SCL
    gpio_pull_up(4);
    gpio_pull_up(5);

    mpu6050_init();

    while (true) {
        int16_t ax, ay, az;
        int16_t gx, gy, gz;
        float temp;

        read_mpu6050_data(&ax, &ay, &az, &gx, &gy, &gz, &temp);

        float ax_g = ax / ACCEL_SENSITIVITY;
        float ay_g = ay / ACCEL_SENSITIVITY;
        float az_g = az / ACCEL_SENSITIVITY;
        float gx_dps = gx / GYRO_SENSITIVITY_250;
        float gy_dps = gy / GYRO_SENSITIVITY_250;
        float gz_dps = gz / GYRO_SENSITIVITY_250;

        printf("Accel: X=%.2f g, Y=%.2f g, Z=%.2f g\n", ax_g, ay_g, az_g);
        printf("Gyro: X=%.2f deg/s, Y=%.2f deg/s, Z=%.2f deg/s\n", gx_dps, gy_dps, gz_dps);
        printf("Temperature: %.2f °C\n", temp);

        sleep_ms(1000);
    }
}
```

4 References

MPU 6050 Datasheet- https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

MPU 6050 Register Map - https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

Lots of ChatGPT