ADXL343

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/i2c.h"

// I2C address
static const uint8_t ADXL343_ADDR = 0x53;

// Registers
static const uint8_t REG_DEVID = 0x00;
static const uint8_t REG_POWER_CTL = 0x2D;
static const uint8_t REG_DATAX0 = 0x32;

// Other constants
static const uint8_t DEVID = 0xE5;
static const float SENSITIVITY_2G = 1.0 / 256;  // (g/LSB)
static const float EARTH_GRAVITY = 9.80665;     // Earth's gravity in [m/s^2]


int reg_write(  i2c_inst_t *i2c,
                const uint addr,
                const uint8_t reg,
                uint8_t *buf,
                const uint8_t nbytes);

int reg_read(   i2c_inst_t *i2c,
                const uint addr,
                const uint8_t reg,
                uint8_t *buf,
                const uint8_t nbytes);


// Write 1 byte to the specified register
int reg_write(  i2c_inst_t *i2c,
                const uint addr,
                const uint8_t reg,
                uint8_t *buf,
                const uint8_t nbytes) {

    int num_bytes_read = 0;
    uint8_t msg[nbytes + 1];

    // Check to make sure caller is sending 1 or more bytes
    if (nbytes < 1) {
        return 0;
    }

    // Append register address to front of data packet
    msg[0] = reg;
    for (int i = 0; i < nbytes; i++) {
        msg[i + 1] = buf[i];
    }

    // Write data to register(s) over I2C
    i2c_write_blocking(i2c, addr, msg, (nbytes + 1), false);

    return num_bytes_read;
}

// Read byte(s) from specified register. If nbytes > 1, read from consecutive
// registers.
int reg_read(   i2c_inst_t *i2c,
                const uint addr,
                const uint8_t reg,
                uint8_t *buf,
                const uint8_t nbytes) {

    int num_bytes_read = 0;

    // Check to make sure caller is asking for 1 or more bytes
    if (nbytes < 1) {
        return 0;
    }

    // Read data from register(s) over I2C
    i2c_write_blocking(i2c, addr, &reg, 1, true);
    num_bytes_read = i2c_read_blocking(i2c, addr, buf, nbytes, false);

    return num_bytes_read;
}


int main() {
    int16_t acc_x;
    int16_t acc_y;
    int16_t acc_z;
    float acc_x_f;
    float acc_y_f;
    float acc_z_f;
```

```
    // Pins
    const uint sda_pin = 4;
    const uint scl_pin = 5;

    // Ports
    i2c_inst_t *i2c = i2c0;

    // Buffer to store raw reads
    uint8_t data[6];

    // Initialize chosen serial port
    stdio_init_all();

    //Initialize I2C port at 400 kHz
    i2c_init(i2c, 400 * 1000);

    // Initialize I2C pins
    gpio_set_function(sda_pin, GPIO_FUNC_I2C);
    gpio_set_function(scl_pin, GPIO_FUNC_I2C);

    // Read device ID to make sure that we can communicate with the ADXL343
    reg_read(i2c, ADXL343_ADDR, REG_DEVID, data, 1);
    if (data[0] != DEVID) {
        printf("ERROR: Could not communicate with ADXL343\r\n");
        while (true);
    }

    // Read Power Control register
    reg_read(i2c, ADXL343_ADDR, REG_POWER_CTL, data, 1);
    printf("0x%02X\r\n", data[0]);

    // Tell ADXL343 to start taking measurements by setting Measure bit to high
    data[0] |= (1 << 3);
    reg_write(i2c, ADXL343_ADDR, REG_POWER_CTL, &data[0], 1);

    // Test: read Power Control register back to make sure Measure bit was set
    reg_read(i2c, ADXL343_ADDR, REG_POWER_CTL, data, 1);
    printf("0x%02X\r\n", data[0]);

    // Wait before taking measurements
    sleep_ms(2000);

    // Loop forever
    while (true) {

        // Read X, Y, and Z values from registers (16 bits each)
        reg_read(i2c, ADXL343_ADDR, REG_DATAX0, data, 6);

        // Convert 2 bytes (little-endian) into 16-bit integer (signed)
        acc_x = (int16_t)((data[1] << 8) | data[0]);
        acc_y = (int16_t)((data[3] << 8) | data[2]);
        acc_z = (int16_t)((data[5] << 8) | data[4]);

        // Convert measurements to [m/s^2]
        acc_x_f = acc_x * SENSITIVITY_2G * EARTH_GRAVITY;
        acc_y_f = acc_y * SENSITIVITY_2G * EARTH_GRAVITY;
        acc_z_f = acc_z * SENSITIVITY_2G * EARTH_GRAVITY;

        // Print results
        printf("X: %.2f | Y: %.2f | Z: %.2f\r\n", acc_x_f, acc_y_f, acc_z_f);

        sleep_ms(100);
    }
}
```

MPU6050

```
#include <stdio.h>
#include <string.h>
#include "pico/stdlib.h"
#include "pico/binary_info.h"
#include "hardware/i2c.h"

/* Example code to talk to a MPU6050 MEMS accelerometer and gyroscope

   This is taking to simple approach of simply reading registers. It's perfectly
   possible to link up an interrupt line and set things up to read from the
   inbuilt FIFO to make it more useful.

   NOTE: Ensure the device is capable of being driven at 3.3v NOT 5v. The Pico
   GPIO (and therefor I2C) cannot be used at 5v.

   You will need to use a level shifter on the I2C lines if you want to run the
   board at 5v.

   Connections on Raspberry Pi Pico board, other boards may vary.

   GPIO PICO_DEFAULT_I2C_SDA_PIN (On Pico this is GP4 (pin 6)) -> SDA on MPU6050 board
   GPIO PICO_DEFAULT_I2C_SCL_PIN (On Pico this is GP5 (pin 7)) -> SCL on MPU6050 board
   3.3v (pin 36) -> VCC on MPU6050 board
   GND (pin 38)  -> GND on MPU6050 board
*/

// By default these devices  are on bus address 0x68
static int addr = 0x68;

#ifdef i2c_default
static void mpu6050_reset() {
    // Two byte reset. First byte register, second byte data
    // There are a load more options to set up the device in different ways that could be added here
    uint8_t buf[] = {0x6B, 0x00};
    i2c_write_blocking(i2c_default, addr, buf, 2, false);
}

static void mpu6050_read_raw(int16_t accel[3], int16_t gyro[3], int16_t *temp) {
    // For this particular device, we send the device the register we want to read
    // first, then subsequently read from the device. The register is auto incrementing
    // so we don't need to keep sending the register we want, just the first.

    uint8_t buffer[6];

    // Start reading acceleration registers from register 0x3B for 6 bytes
    uint8_t val = 0x3B;
    i2c_write_blocking(i2c_default, addr, &val, 1, true); // true to keep master control of bus
    i2c_read_blocking(i2c_default, addr, buffer, 6, false);

    for (int i = 0; i < 3; i++) {
        accel[i] = (buffer[i * 2] << 8 | buffer[(i * 2) + 1]);
    }

    // Now gyro data from reg 0x43 for 6 bytes
    // The register is auto incrementing on each read
    val = 0x43;
    i2c_write_blocking(i2c_default, addr, &val, 1, true);
    i2c_read_blocking(i2c_default, addr, buffer, 6, false);  // False - finished with bus

    for (int i = 0; i < 3; i++) {
        gyro[i] = (buffer[i * 2] << 8 | buffer[(i * 2) + 1]);;
    }

    // Now temperature from reg 0x41 for 2 bytes
    // The register is auto incrementing on each read
    val = 0x41;
    i2c_write_blocking(i2c_default, addr, &val, 1, true);
    i2c_read_blocking(i2c_default, addr, buffer, 2, false);  // False - finished with bus

    *temp = buffer[0] << 8 | buffer[1];
}
#endif

int main() {
    stdio_init_all();
#if !defined(i2c_default) || !defined(PICO_DEFAULT_I2C_SDA_PIN) || !defined(PICO_DEFAULT_I2C_SCL_PIN)
    #warning i2c/mpu6050_i2c example requires a board with I2C pins
    puts("Default I2C pins were not defined");
#else
    printf("Hello, MPU6050! Reading raw data from registers...\n");

    // This example will use I2C0 on the default SDA and SCL pins (4, 5 on a Pico)
    i2c_init(i2c_default, 400 * 1000);
    gpio_set_function(PICO_DEFAULT_I2C_SDA_PIN, GPIO_FUNC_I2C);
    gpio_set_function(PICO_DEFAULT_I2C_SCL_PIN, GPIO_FUNC_I2C);
    gpio_pull_up(PICO_DEFAULT_I2C_SDA_PIN);
    gpio_pull_up(PICO_DEFAULT_I2C_SCL_PIN);
    // Make the I2C pins available to picotool
    bi_decl(bi_2pins_with_func(PICO_DEFAULT_I2C_SDA_PIN, PICO_DEFAULT_I2C_SCL_PIN, GPIO_FUNC_I2C));
```

```
    mpu6050_reset();

    int16_t acceleration[3], gyro[3], temp;

    while (1) {
        mpu6050_read_raw(acceleration, gyro, &temp);

        // These are the raw numbers from the chip, so will need tweaking to be really useful.
        // See the datasheet for more information
        printf("Acc. X = %d, Y = %d, Z = %d\n", acceleration[0], acceleration[1], acceleration[2]);
        printf("Gyro. X = %d, Y = %d, Z = %d\n", gyro[0], gyro[1], gyro[2]);
        // Temperature is simple so use the datasheet calculation to get deg C.
        // Note this is chip temperature.
        printf("Temp. = %f\n", (temp / 340.0) + 36.53);

        sleep_ms(100);
    }

#endif
    return 0;
}
```