#include <stdio.h>

#include "pico/stdlib.h"

#include "pico/binary\_info.h"

#include "hardware/spi.h"

#include <stdint.h>

// =====================

// Define Constants

// =====================

// Define the Chip Select (CS) pin for SPI

#define IMU\_CS\_PIN 5 // Change this to the actual CS pin connected to the IMU

#define READ\_BIT 0x80 // Read operation indicator

// ICM-42686 Register Addresses (Placeholders - Replace with actual addresses)

#define REG\_PWR\_MGMT\_1 0x06 // Power Management Register

#define REG\_RESET 0x07 // Reset Register

#define REG\_DEVICE\_ID 0x75 // Device ID Register

#define REG\_FIFO\_EN 0x14 // FIFO Enable Register

#define REG\_FIFO\_CONFIG 0x16 // FIFO Configuration Register

#define REG\_FIFO\_STATUS 0x18 // FIFO Status Register

#define REG\_FIFO\_DATA 0x19 // FIFO Data Register

// Expected Device ID for ICM-42686

#define DEVICE\_ID\_ICM42686 0x43 // Example Device ID (verify from datasheet)

// FIFO Configuration Settings (Placeholders - Adjust as per datasheet)

#define FIFO\_MODE\_CONTINUOUS 0x00

#define FIFO\_MODE\_STOP 0x01

#define FIFO\_SAMPLE\_RATE 0x04 // Example: Set sample rate divider

// Function Prototypes

static inline void cs\_select();

static inline void cs\_deselect();

static void imu\_reset();

static void imu\_init();

static void fifo\_init();

static uint8\_t imu\_read\_device\_id();

static void read\_register(uint8\_t add\_reg, uint8\_t \*buf, uint16\_t len);

static void write\_register(uint8\_t add\_reg, uint8\_t value);

static uint16\_t fifo\_get\_sample\_count();

static void fifo\_read\_data(uint8\_t \*buf, uint16\_t len);

static void parse\_fifo\_data(uint8\_t \*buf, uint16\_t len, int16\_t accel[3], int16\_t gyro[3]);

// Function to select the CS pin (active low)

static inline void cs\_select(){

gpio\_put(IMU\_CS\_PIN, 0);

}

// Function to deselect the CS pin

static inline void cs\_deselect(){

gpio\_put(IMU\_CS\_PIN, 1);

}

// Function to reset the IMU

static void imu\_reset(){

uint8\_t reset\_cmd[] = {REG\_RESET, 0x01}; // Writing 1 to reset register

cs\_select();

spi\_write\_blocking(spi\_default, reset\_cmd, 2);

cs\_deselect();

sleep\_ms(10); // Wait for reset to complete

}

// Function to initialize the IMU (wake up and basic settings)

static void imu\_init(){

uint8\_t buf[2];

// Wake up the device by clearing the sleep bit in the power management register

buf[0] = REG\_PWR\_MGMT\_1;

buf[1] = 0x00; // Assuming 0x00 wakes up the device (verify from datasheet)

cs\_select();

spi\_write\_blocking(spi\_default, buf, 2);

cs\_deselect();

sleep\_ms(10); // Wait for power management settings to take effect

// Additional configuration can be added here as needed

// For example, setting sample rates, enabling specific sensors, etc.

}

// Function to initialize the FIFO

static void fifo\_init(){

uint8\_t buf;

// 1. Enable FIFO for desired data streams

// Assuming bit 0: Accelerometer, bit 1: Gyroscope (verify from datasheet)

buf = 0x03; // Enable both Accel and Gyro

write\_register(REG\_FIFO\_EN, buf);

// 2. Configure FIFO settings

// Set FIFO mode and sample rate

// Assuming bits 0-1: FIFO mode, bits 2-7: Sample rate divider

buf = (FIFO\_MODE\_CONTINUOUS & 0x03) | ((FIFO\_SAMPLE\_RATE << 2) & 0xFC);

write\_register(REG\_FIFO\_CONFIG, buf);

// 3. Additional FIFO configurations can be set here as per requirements

// For example, setting watermark levels, trigger sources, etc.

sleep\_ms(10); // Wait for FIFO settings to take effect

}

// Function to read the device ID for verification

static uint8\_t imu\_read\_device\_id(){

uint8\_t device\_id;

read\_register(REG\_DEVICE\_ID, &device\_id, 1);

return device\_id;

}

// Function to read data from a specified register

static void read\_register(uint8\_t add\_reg, uint8\_t \*buf, uint16\_t len){

uint8\_t tx[len + 1];

tx[0] = add\_reg | READ\_BIT; // Set the read bit

// Initialize dummy bytes for reading

for(int i = 1; i <= len; i++) tx[i] = 0x00;

cs\_select();

spi\_write\_read\_blocking(spi\_default, tx, buf, len + 1);

cs\_deselect();

// Diagnostic print (optional, can be commented out after debugging)

/\*

printf("Read from register 0x%02X: ", add\_reg & 0x7F);

for(int i = 1; i <= len; i++) {

printf("0x%02X ", buf[i-1]);

}

printf("\n");

\*/

}

// Function to write a single byte to a register

static void write\_register(uint8\_t add\_reg, uint8\_t value){

uint8\_t tx[2];

tx[0] = add\_reg & 0x7F; // Ensure write bit is cleared

tx[1] = value;

cs\_select();

spi\_write\_blocking(spi\_default, tx, 2);

cs\_deselect();

// Optional: Add delay if required by the sensor

sleep\_ms(1);

}

// Function to get the number of samples available in FIFO

static uint16\_t fifo\_get\_sample\_count(){

uint8\_t buf[2];

read\_register(REG\_FIFO\_STATUS, buf, 2); // Assuming FIFO\_STATUS has two bytes

// Parsing FIFO sample count

// Assuming FIFO\_STATUS register provides the number of samples

// Replace with actual parsing as per datasheet

uint16\_t sample\_count = ((uint16\_t)buf[1] << 8) | buf[0];

return sample\_count;

}

// Function to read data from FIFO

static void fifo\_read\_data(uint8\_t \*buf, uint16\_t len){

// Read 'len' bytes starting from FIFO\_DATA register

read\_register(REG\_FIFO\_DATA, buf, len);

}

// Function to parse FIFO data into acceleration and gyroscope arrays

static void parse\_fifo\_data(uint8\_t \*buf, uint16\_t len, int16\_t accel[3], int16\_t gyro[3]){

// Each sample is 12 bytes: 6 bytes accel + 6 bytes gyro

// Ensure that len is a multiple of 12

if(len % 12 != 0){

printf("FIFO data length mismatch!\n");

return;

}

uint16\_t samples = len / 12;

for(uint16\_t s = 0; s < samples; s++){

uint16\_t base = s \* 12;

// Parse Acceleration Data (Assuming order: X, Y, Z)

accel[0] = ((int16\_t)(buf[base] << 8)) | buf[base + 1];

accel[1] = ((int16\_t)(buf[base + 2] << 8)) | buf[base + 3];

accel[2] = ((int16\_t)(buf[base + 4] << 8)) | buf[base + 5];

// Parse Gyroscope Data (Assuming order: X, Y, Z)

gyro[0] = ((int16\_t)(buf[base + 6] << 8)) | buf[base + 7];

gyro[1] = ((int16\_t)(buf[base + 8] << 8)) | buf[base + 9];

gyro[2] = ((int16\_t)(buf[base + 10] << 8)) | buf[base + 11];

// Print the parsed data

printf("Sample %d:\n", s + 1);

printf(" Acc. X = %d, Y = %d, Z = %d\n", accel[0], accel[1], accel[2]);

printf(" Gyro. X = %d, Y = %d, Z = %d\n", gyro[0], gyro[1], gyro[2]);

}

}

// Main function

int main() {

// Initialize standard I/O for UART communication

stdio\_init\_all();

sleep\_ms(1000); // Allow time for UART to initialize

// Initialize SPI at 500 kHz (adjust if necessary)

spi\_init(spi\_default, 500 \* 1000);

// Set SPI GPIO pins

gpio\_set\_function(PICO\_DEFAULT\_SPI\_RX\_PIN, GPIO\_FUNC\_SPI); // MISO

gpio\_set\_function(PICO\_DEFAULT\_SPI\_SCK\_PIN, GPIO\_FUNC\_SPI); // SCK

gpio\_set\_function(PICO\_DEFAULT\_SPI\_TX\_PIN, GPIO\_FUNC\_SPI); // MOSI

bi\_decl(bi\_3pins\_with\_func(

PICO\_DEFAULT\_SPI\_RX\_PIN,

PICO\_DEFAULT\_SPI\_TX\_PIN,

PICO\_DEFAULT\_SPI\_SCK\_PIN,

GPIO\_FUNC\_SPI

));

// Initialize CS pin

gpio\_init(IMU\_CS\_PIN);

gpio\_set\_dir(IMU\_CS\_PIN, GPIO\_OUT);

gpio\_put(IMU\_CS\_PIN, 1); // Deselect IMU

bi\_decl(bi\_1pin\_with\_name(IMU\_CS\_PIN, "SPI CS"));

// Configure SPI format: 8 bits, Mode 0, MSB first

spi\_set\_format(spi\_default, 8, SPI\_CPOL\_0, SPI\_CPHA\_0, SPI\_MSB\_FIRST);

// Reset and initialize the IMU

imu\_reset();

imu\_init();

fifo\_init();

// Verify device ID

uint8\_t device\_id = imu\_read\_device\_id();

if(device\_id != DEVICE\_ID\_ICM42686){

printf("Device ID mismatch! Expected 0x%02X, Got 0x%02X\n", DEVICE\_ID\_ICM42686, device\_id);

while (1) {

sleep\_ms(1000); // Halt execution

}

}

printf("IMU Initialization Successful. Device ID: 0x%02X\n", device\_id);

// Buffer to store FIFO data

// Adjust size as per maximum expected samples

#define MAX\_SAMPLES 10

uint8\_t fifo\_buffer[MAX\_SAMPLES \* 12]; // 12 bytes per sample

// Arrays to store parsed acceleration and gyroscope data

int16\_t accel[3], gyro[3];

while (true)

{

// Get the number of samples available in FIFO

uint16\_t sample\_count = fifo\_get\_sample\_count();

if(sample\_count > 0){

// Limit to MAX\_SAMPLES to prevent buffer overflow

if(sample\_count > MAX\_SAMPLES){

sample\_count = MAX\_SAMPLES;

}

// Calculate the number of bytes to read

uint16\_t bytes\_to\_read = sample\_count \* 12;

// Read FIFO data

fifo\_read\_data(fifo\_buffer, bytes\_to\_read);

// Parse and display the data

parse\_fifo\_data(fifo\_buffer, bytes\_to\_read, accel, gyro);

}

sleep\_ms(100); // Adjust delay as needed based on sample rate

}

}