Chapter 1: STM32 Programming

2023

SWV to print and plot variables

SWV stands for **Serial Wire Viewer**. This tool allows printing and plotting variables. Steps to configure SWV:

1. Configure SWO properly
2. Implement _write function:
   ```
   int _write(int file, char *ptr, int len)
   { int DataIdx;
   for(DataIdx = 0; DataIdx < len; DataIdx++)
   ITM_SendChar(*ptr++);
   return len;
   }
   ```
3. Test printf function. You can write the following lines inside of the while loop:
   ```
   printf("Hello world \n");
   HAL_Delay(1000);
   ```
4. Before debugging, enable SWV and set proper Clock Frequency
5. Open SWV ITM data console
6. Enable Port 0
7. Start "trace" and resume code

ICM-20948 communication

One can communicate with ICM-20948 sensor using SPI Interface.

SPI - **Serial Peripheral Interface**

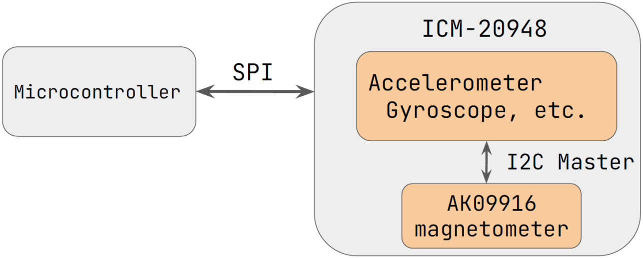

Figure: ICM-20948 communication illustration

SPI Interface

SPI - **Serial Peripheral Interface**

1. SCLK - Serial Clock
2. MISO - Master-in-slave-out
3. MOSI - Master-out-slave-in
4. CS - Chip select

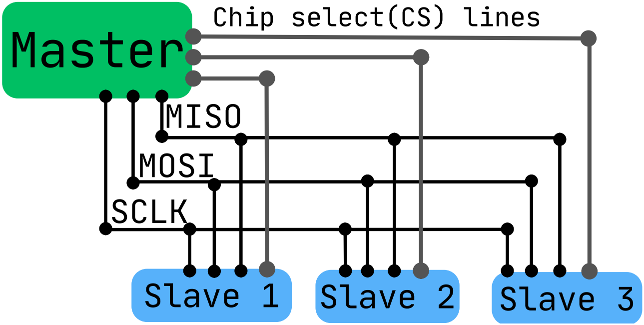

Figure: SPI illustration

SPI Interface Example: communicate/activate slave 2

CS line 1&3: High
CS line 2: Low

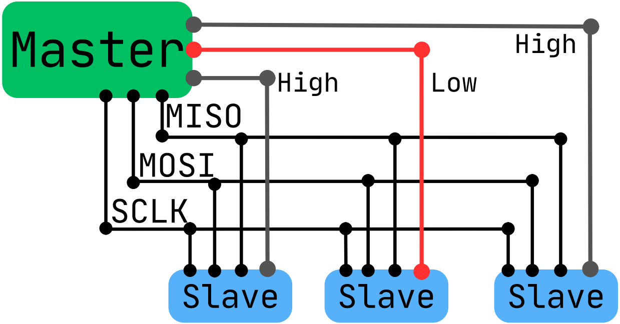

Figure: SPI illustration: activating slave 2

ICM-20948 Block Diagram

The IMU sensor contains multiple registers. The size of each register is one byte (8 bits).

Each register has its own unique address. By **writing** to certain registers, we can control how the sensor operates.

By **reading** registers, we can extract information from the sensor.

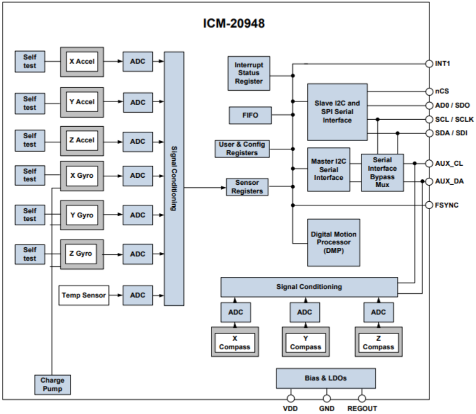

Figure: ICM-20948 block diagram

SPI: master sending data

Example: send 0x57, then 0x34:
The value of register 0x57 is 0x34.

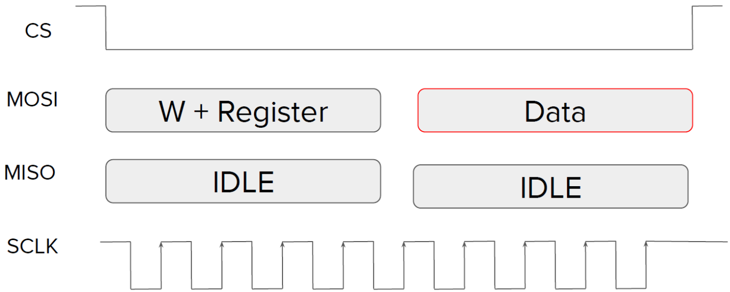

Figure: Master sending data

SPI: master receiving data

Example: send 0x80|0x57 and 0xff:
'Data' is the value of the register 0x57

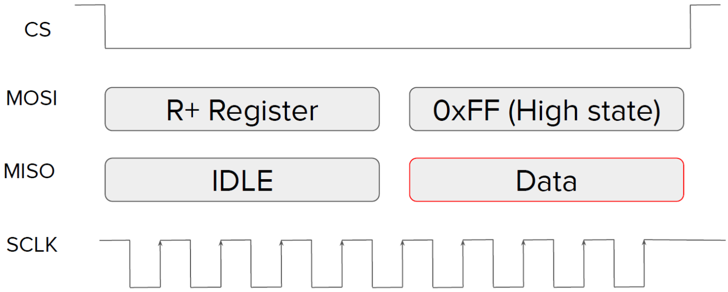

Figure: Master receiving data

SPI: multiple-byte read

Example: send 0x80|0x57, 0xff, 0xff, 0xff:
We will get the values of registers 0x57, 0x58, and 0x59

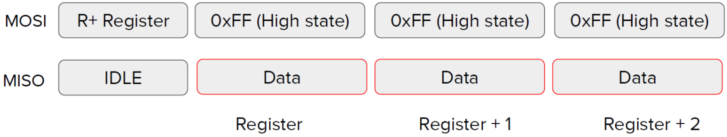

Figure: Multiple-byte read

SPI: multiple-byte write

Example: send 0x57, 0x03, 0x45, 0x35:
Register 0x57 = 0x03
Register 0x58 = 0x45
Register 0x59 = 0x35

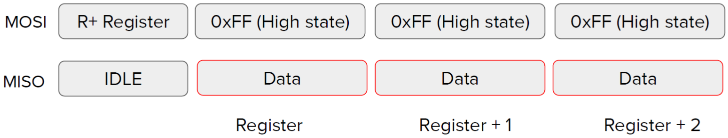

Figure: Multiple-byte read

SPI: Clock polarity (CPOL) and Clock Phase (CPHA)

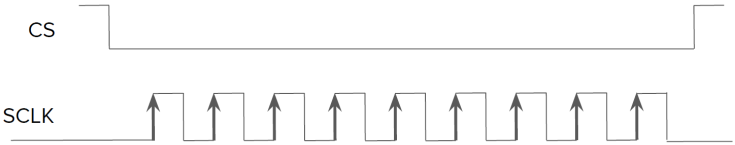

Figure: CPOL = 0 and CPHA = 0: clock is low in idle state and sampling on the leading clock edge

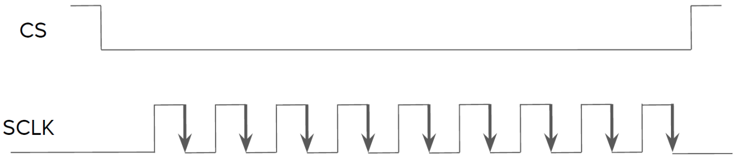

Figure: CPOL = 0 and CPHA = 1: clock is low in idle state and sampling on the second clock edge

SPI: Clock polarity (CPOL) and Clock Phase (CPHA)

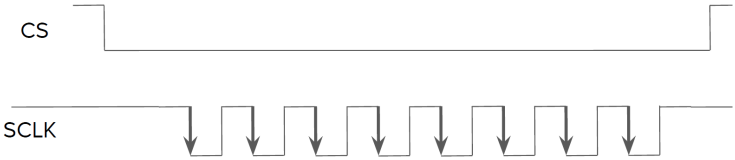

Figure: CPOL = 1 and CPHA = 0: clock is high in idle state and sampling on the leading clock edge

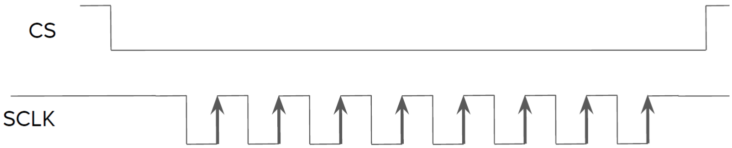

Figure: CPOL = 2 and CPHA = 1: clock is high in idle state and sampling on the second clock edge

Gyroscope

The gyroscope measures an angular velocity. The measurements are 16-bit signed integer numbers, which has the following value range: [-32767, 32767]

Gyroscope

Example: 250 dps full-scale, value 345:

$$32757 \longleftrightarrow 250\ deg/sec$$
$$345 \longleftrightarrow x\ deg/sec$$

$x = 345 * 250/32767 = 2.63\ deg/sec$

Example: 500 dps full-scale, value -400:

$$32757 \longleftrightarrow 500\ deg/sec$$
$$-400 \longleftrightarrow x\ deg/sec$$

$x = -400 * 500/32767 = -6.10\ deg/sec$

Accelerometer

The accelerometer measures acceleration. The measurements are 16-bit signed integer numbers, which has the following value range: [-32767, 32767]

Accelerometer

Example: 4g full scale, value 345

$$32757 \longleftrightarrow 4\ g$$
$$345 \longleftrightarrow x\ g$$

$$x = 345 * 4/32767\ g = 0.042\ g$$

Example: 2g full scale, value -900:

$$32757 \longleftrightarrow 2\ g$$
$$-900 \longleftrightarrow x\ g$$

$$x = -900 * 2/32767\ g = -0.0549\ g$$

Direct Memory Access (DMA)

Problem Statement

Blocking mode:
HAL_SPI_Receive(&IMU_SPI, data_rx, 20, 1000);

for (int i = 0; i < buffer_size; i++)
{ when(data_available)
{ Buffer[i] = peripheral_rx; } }

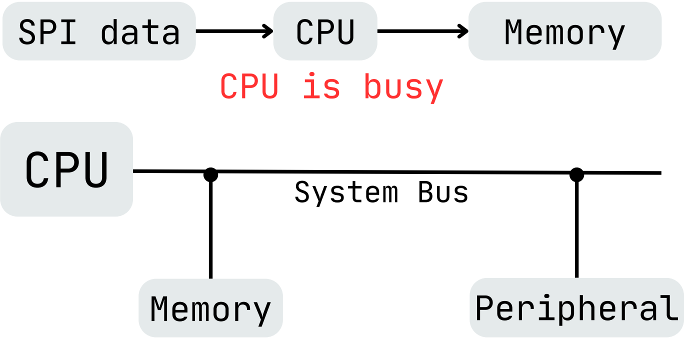

Figure: Problem statement

Direct Memory Access (DMA)

Solution

DMA transfers data from location A to B without the CPU intervention:

1. Memory to peripheral (M2P)
2. Peripheral to memory (P2M)
3. Memory to memory (M2M)

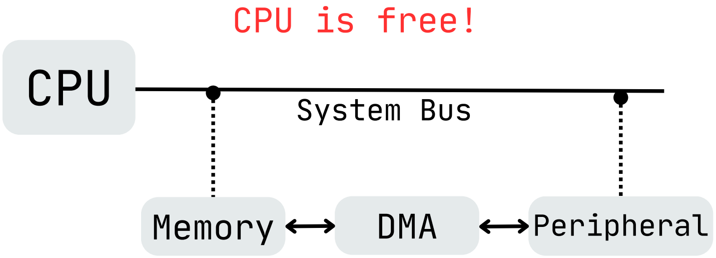

Figure: Solution

Direct Memory Access (DMA) Configuration

1. Peripheral address (I2C1_RX, I2C1_TX, et cetera)
2. Memory address (address of an array or variable)
3. Channel priority
4. Increment mode
5. The peripheral and memory data size (1 byte, 2 bytes, 4 bytes, ..)
6. Circular and non-circular mode

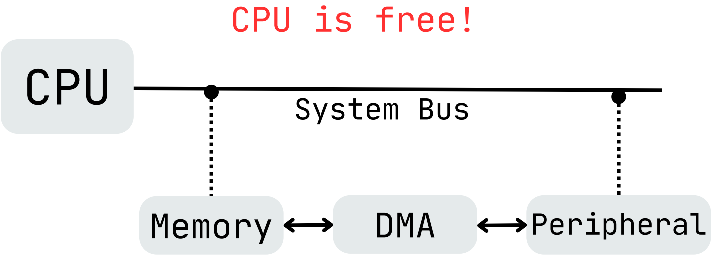

Figure: Solution

Accelerometer & Magnetometer normalization

ACCELEROMETER

Accelerometer Magnitude computation:

$accel\_mag = \sqrt{accel\_x^2 + accel\_y^2 + accel\_z^2}$

Accelerometer Normalization:

$$\begin{bmatrix} accel\_x \\ accel\_y \\ accel\_z \end{bmatrix} \rightarrow \begin{bmatrix} accel\_x/accel\_mag \\ accel\_y/accel\_mag \\ accel\_z/accel\_mag \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az \end{bmatrix} \tag{1}$$

Outcome: $\sqrt{ax^2 + ay^2 + az^2} = 1$

MAGNETOMETER

Magnetometer Magnitude computation:

$compass\_mag = \sqrt{mag\_x^2 + mag\_y^2 + mag\_z^2}$

Accelerometer Normalization:

$$\begin{bmatrix} mag\_x \\ mag\_y \\ mag\_z \end{bmatrix} \rightarrow \begin{bmatrix} mag\_x/compass\_mag \\ mag\_y/compass\_mag \\ mag\_z/compass\_mag \end{bmatrix} = \begin{bmatrix} mx \\ my \\ mz \end{bmatrix} \tag{2}$$

Gyroscope scaling

It is necessary to scale the gyroscope readings to radians per sec (rad/sec) unit.

Scaling Factor:
$(pi/180) \times (full\_scale/32767) = full\_scale \times 5.32648 \times 10^{-7}$

Example: 250 dps
$Scale\_factor = 250 \times 5.32648 \times 10^{-7} = 0.00013162$

$$\begin{bmatrix} gyro\_x \\ gyro\_y \\ gyro\_z \end{bmatrix} \rightarrow \begin{bmatrix} gyro\_x \times Scale\_factor \\ gyro\_y \times Scale\_factor \\ gyro\_z \times Scale\_factor \end{bmatrix} = \begin{bmatrix} gx \\ gy \\ gz \end{bmatrix} \tag{3}$$