MPU6050 Over I^{2}C on PSoC 5

Ian Lacy

June 2023

1 Overview

This tutorial shows how to configure the Cypress PSoC 5 to communicate with an MPU6050 Module using the I^{2}C (Inter-Integrated Circuit, said as "I squared C") protocol. This tutorial will cover the basics of I^{2}C and is meant to serve as a quick-start guide to I^{2}C on the PSoC 5.

Accompanying this tutorial is a Cypress Workspace containing three example projects ranging from bare-bones I^{2}C communication, to a deeper look into the MPU6050's functionality.

2 Hardware

2.1 PSoC 5LP

The projects "Example 1- I2C Simple Use" and "Example 2- I2C Two-axis" use only the PSoC 5LP "Stick." The more thorough example, "Example 3- I2C Kovid Konsole" uses the Kovid Konsole as built in class.

In Figure 1 below, pins boxed in RED are used in all projects. Pins boxed in BLUE are used only in Example 3.

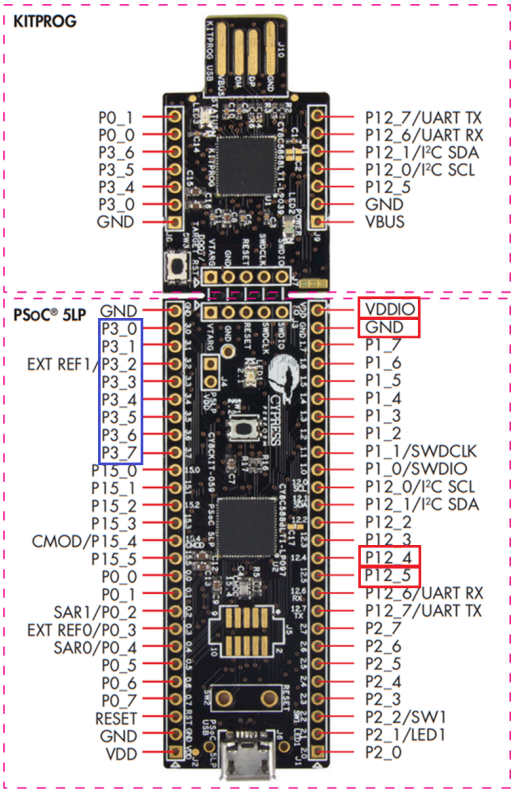

Figure 1: The Pins used in this tutorial.

2.2 MPU6050 Module

The MPU6060 is a very popular inertial measurement unit (IMU) and is commonly sold on breakout boards for use with prototyping platforms such as Arduino and Raspberry Pi. These breakout boards are often referred to as MPU6050s themselves, as is done here. This IMU features three gyroscope axes to measure angular velocity and three accelerometer axes to measure linear acceleration. It also features an infrequently-used temperature sensor as well as an auxiliary interface for use with 3rd-party magnetometer expansion chips. It features an I^2C interface, an interrupt pin for data control and frame synchronization, a digital low-pass filter for the measured data, and an integrated sleep mode. Thanks to this chip's ubiquity, ease of use, and simple interface, it will be used as the foundation for these I^2C examples.

Figure 3 below shows the pinout of the MPU6050 Module. Table 1 gives a description of the pins and their use.

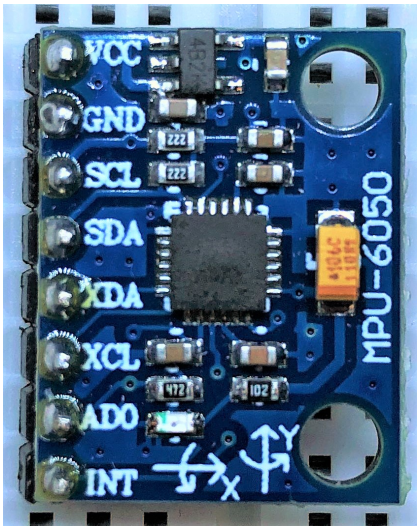

Figure 2: The MPU6050 Module Used in This Example

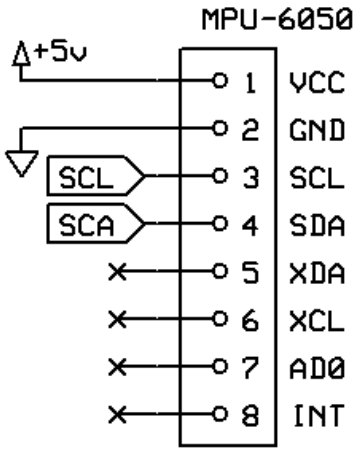

Figure 3: The Pinout/Wiring Diagram for the MPU6050 Board.

Since this module features an I^2C interface, let's first go over the I^2C protocol, its implementation on PSoC5, and then cover how to use the MPU6050 in particular.

Pin	Description
VCC	+5 V
GND	Ground
SCL	I^2C clock line
SDA	I^2C data line
XDA	Aux. data line
XCL	Aux. clock line
AD0	LSB of I^2C address
INT	Interrupt

Table 1: Pin Descriptions of the MPU6050 Module

3 The I^2C Protocol

The I^2C protocol, invented in 1982, is intended as a shared-bus protocol for serial communications. The interface consists of only two wires:

- SCL- Serial Clock. This line controls the timing of the bus.
- SDA- Serial Data. This line carries the data and control signals of the bus.

Typically, these lines are driven as open-drain with a resistive pull-up. In an inactive state, both lines are high.

All I^2C chips on a bus will connect to the same two wires. Chips are divided into two categories- *Masters* (also known as *Controllers*), and *Slaves* (also known as *Targets*). Controllers, as the name implies, control the operation of the I^2C bus. They start and end transactions, and send read/write commands to targets.

3.1 Signalling

Figure 4 shows an example of the timing of I^2C signals.

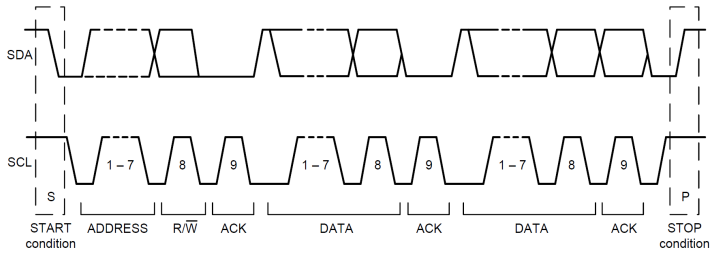

Figure 4: An Example of I^2C Timing. See the MPU6050 datasheet for more details.

There are six signals that are used in I^2C. They are as follows:

- START Condition- This signals the start of a transaction. It is signalled by SDA transitioning from high to low while SCL is high.
- STOP Condition - This signals the end of a transaction. It is signalled by SDA transitioning from low to high while SCL is high.
- RESTART Condition- This is identical to a START signal, but is only signalled after a START signal has been issued without being followed by a STOP signal
- DATA- This signal is eight clock cycles long, signalling 1 byte of data. The SDA line is sampled when SCL transitions from low to high. DATA bytes are sent with the **least significant bit first.** DATA should always be followed by an ACK or a NAK from the receiving chip.

- ACK- Acknowledgement of data received. This is sent by whichever chip is receiving data on the clock cycle after data is sent onto the bus. SDA is held low while SCL transitions from low to high.
- NAK- Non-acknowledgement of data received. Also happens on the cycle after data is sent, but signals that no more data will be requested. SDA is held high while SCL transitions from low to high.
- W/R+AD- A special DATA signal that follows a START or RESTART signal. It is 1 byte- a write/read (W/R) bit concatenated with the 7-bit target address (AD). A 0 indicates a write and a 1 indicates a read. The target address ranges from 0 to 127 (in decimal) and is typically given in manufacturer datasheets. Sometimes, it can be modified in hardware to support multiple copies of a certain chip or to prevent conflicts. As with regular DATA signals, W/R+AD is sent with the **least significant bit first.**

3.2 Transactions

There are two broad types of transactions: read and write. The exact steps for a transaction (and types of transaction) vary by chip, so be sure to consult the datasheet.

An example of a simple 1-byte write transaction follows this procedure:

1. The controller sends a START signal
2. The controller sends W+AD
3. The target at address AD sends ACK
4. The controller sends DATA
5. The target sends ACK
6. The controller sends a STOP signal

An example of a simple 1-byte read transaction follows this procedure:

1. The controller sends a START signal
2. The controller sends R+AD
3. The target sends ACK
4. The target sends DATA
5. The controller sends NAK
6. The controller sends a STOP signal

There are more types of transactions. For example, a project may require that a chip writes or reads multiple bytes in a row from a chip, or that it writes a register address to a chip to then begin reading from that location. The procedures for these types of transactions may vary by chip. **Be sure to consult the datasheet.**

4 I^2C on PSoC

PSoC Creator 3.3 includes an I^2C Master schematic macro, found in the component catalog under **Cypress > Communications > I2C**.

The **I2C Master (Fixed Function)** component, shown in Figure 4 is used for the example projects. It has two terminals: **scl** and **sda**. These should be routed to pins 12.4 and 12.5 respectively. You'll notice these pins are marked as "I2C[0].scl" and "I2C[0].sda" in the design-wide resources file, and they're intended to be used for this special function. The pins these connect to should be set as bidirectional pins, with the Drive mode set to "Resistive pull up".

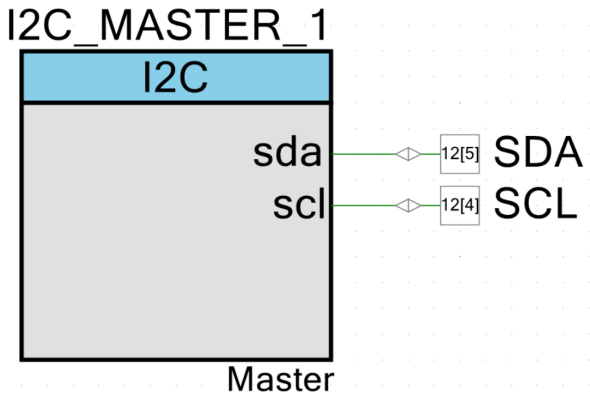

Figure 5: The I2C Master Schematic Macro Connected to Two Pins Named 'SCL' and 'SDA'

The settings for the I^{2}C Master component have options for data rate and Fixed Function/UDB implementations. For these projects, the data rate is set to 100 kbps and it is left as a **Fixed Function** I^{2}C module in Master mode.

Be sure to enable the I^{2}C master with the I2C_MASTER_Start() function before attempting to use it.

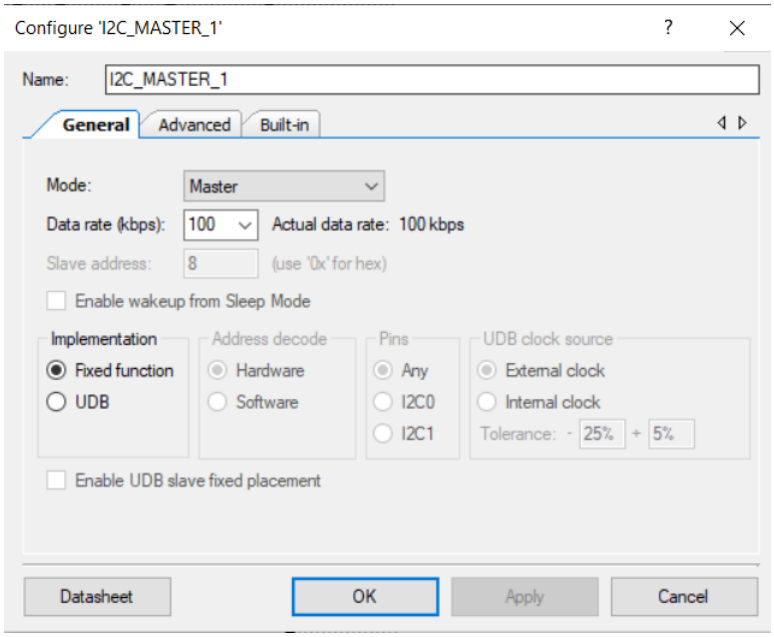

Figure 6: The Settings of the I^{2}C Master Schematic Component

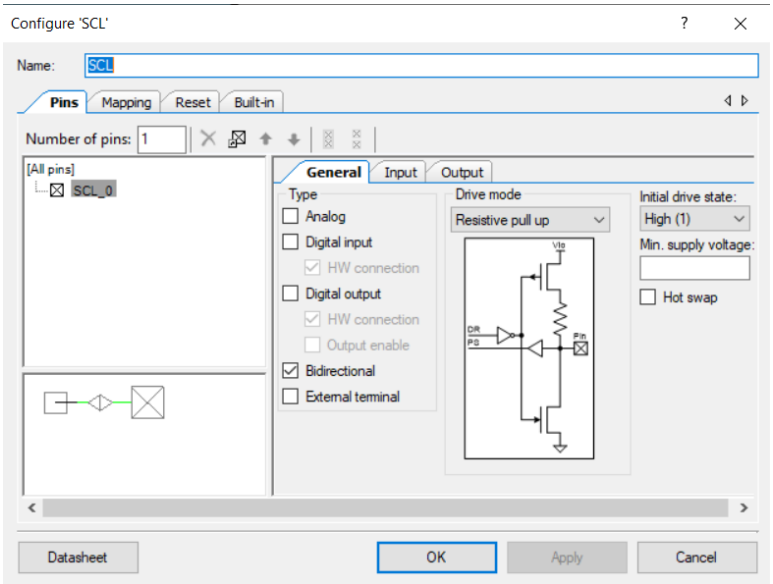

Figure 7: The Settings of the SDA and SCL Pins

4.1 I2C_MASTER API

There are four basic commands used in this project. For more information on the API functions, consult the component's datasheet.

The commands are:

- I2C_MASTER_MasterSendStart(uint8 slaveAddress, uint8 R_nW) - This function sends a START signal to the I^2C bus, followed by R/W + AD. slaveAddress is the target address, and R_nW is 0 for writing and 1 for reading
- I2C_MASTER_MasterSendRestart(uint8 slaveAddress, uint8 R_nW) - This function sends a RESTART signal to the I^2C bus, followed by R/W + AD. slaveAddress is the target address, and R_nW is 0 for writing and 1 for reading.
- I2C_MASTER_MasterSendStop(void) - This functions sends a STOP signal to the I^2C bus.
- I2C_MASTER_MasterWriteByte(uint8 theByte) - This function sends DATA over the I^2C bus and waits to receive ACK or NAK from the target.
- I2C_MASTER_MasterReadByte(uint8 acknNak) - This function receives DATA over the I^2C bus and sends ACK or NAK in response. The parameter should be the I2C_MASTER_ACK_DATA or I2C_MASTER_NAK_DATA macros for ACK or NAK, respectively.

5 The MPU6050 as an I^2C example

The MPU6050's interface is relatively simple. It has a series of control registers and readout registers, and reads/writes are done in the same way for all of them. Sequential writes/reads can be done easily, as the register address pointed to increments after each read or write command. The following subsections cover the basics of using the interface, the control registers, and the readout registers.

5.1 Communicating with the MPU6050

The MPU6050 has four types of transaction:

- Single Write
- Single Read

- Burst Write
- Burst Read

A single write is done as follows:

1. The controller sends a START signal
2. The controller sends W + AD
3. The target sends ACK
4. The controller sends the register address for the write (DATA)
5. The target sends ACK
6. The controller sends DATA to be written
7. The target sends ACK
8. The controller sends a STOP signal

If it's necessary to write t multiple sequential regsiters, use what's known as a "burst write". To do this, simply repeat steps 6 and 7 for as many registers as need to be written.

A single read is done as follows:

1. The controller sends a START signal
2. the controller sends W + AD
3. The target sends ACK
4. The controller sends the register address for the read (DATA)
5. The target sends ACK
6. The controller sends a RESTART signal
7. The controller sends R + AD
8. The target sends ACK
9. The target sends DATA from the register being read from
10. The controller sends NAK
11. The controller sends a STOP signal

If it's necessary to read from multiple sequential regsiters, use what's known as a "burst read". To do this, the controller will instead send ACK at step 10 and receive another byte from the target. To keep receiving bytes, the controller will keep responding with ACK. To signal that no more bytes will be requested, a NAK is sent by the controller after receiving data, and it is then followed by a STOP signal.

Charts depicting these transactions can be found in the MPU6050 manufacturer datasheet.

5.2 MPU6050 Control Registers

For more information on the MPU6050 control registers, **consult the manufacturer's Register Map.**

For these example projects, there are 4 relevant control registers on the MPU6050. Their addresses and purposes are shown in Table 2.

All four of these registers are written to 0x00 for the simple examples. Register 0x1A is written to 0x06 for the showcase example. This enables the digital low-pass filter, setting it to a bandwidth of 5 Hz.

Address	Description
0x1A	CONFIG, controls frame sync and the DLPF
0x1B	GYRO_CONFIG, controls the gyro configuration, including measurement range
0x1C	ACCEL_CONFIG, controls the accelerometer configuration, including measurement range
0x23	FIFO_EN, controls the FIFO buffer settings
0x6B	PWR_MGMT_1, controls power settings and sleep mode

Table 2: Descriptions of the Relevant Control Registers of the MPU6050

5.3 MPU6050 Readout Registers

For more information on the MPU6050 readout registers, **consult the manufacturer's Register Map.**

The MPU6050's readout registers are broken into the 3 accelerometer axes (starting at address 0x3B), the temperature measurement (starting at address 0x41), and the 3 gyro axes (starting at address 0x43).

All measurements are stored as 16-bit signed integers. To read the full measurement, both registers for a given value must be read, and then combined in software. Table 3 covers the readout measurements.

To read multiple axes at once, as done in Examples 2 and 3, simply do a burst read of the axes needed.

Address	Description
0x3B	x-accel [15:8]
0x3C	x-accel [7:0]
0x3D	y-accel [15:8]
0x3E	y-accel [7:0]
0x3F	z-accel [15:8]
0x40	z-accel [7:0]
0x41	temperature [15:8]
0x42	temperature [7:0]
0x43	x-gyro [15:8]
0x44	x-gyro [7:0]
0x45	y-gyro [15:8]
0x46	y-gyro [7:0]
0x47	z-gyro [15:8]
0x48	z-gyro [7:0]

Table 3: The Readout Registers of the MPU6050

6 Example Projects

This section covers the operation of the example projects. Figure 8 shows the build for "Example 3- I2C Kovid Konsole." The wiring for Examples 1 and 2 are largely the same, simply remove the LED bank and switch bank.

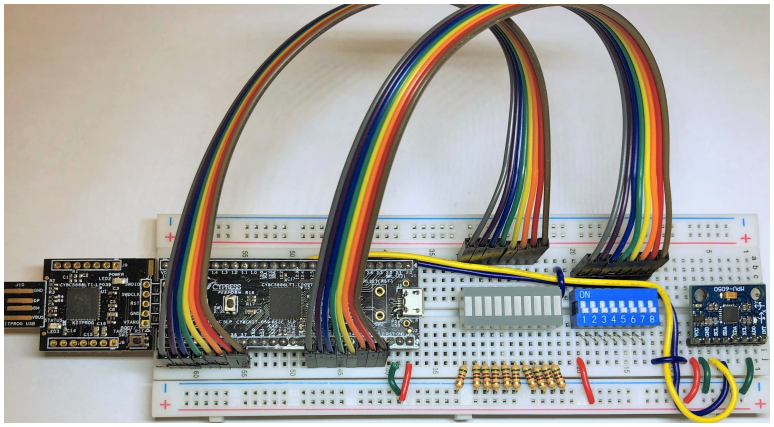

Figure 8: The Kovid Konsole I^{2}C build.

These example projects are designed to be done in order, and each builds off of the previous project. To switch between projects in the workspace, right-click a project name and click "Set As Active Project".

Double-check your wiring. Ensure that pin 12.4 on the PSoC is connected to SCL on the MPU6050 and that pin 12.5 on the PSoC is connected to SDA on the MPU6050. Ensure that the MPU6050 is hooked up to 5V and ground. Refer back to Figure 3 for the wiring diagram of the MPU6050.

Also to note: In Figures 3 and 8, the AD0 pin on the MPU6050 is tied to ground. This is not strictly necessary, as in theory AD0 should be resistively pulled to ground on the module, but it has been done here in case it is not done on the module.

6.1 Example 1: 1-axis Accelerometer

This example is a bare-bones showcase and uses only the x-axis accelerometer. It uses the MPU6050 and the PSoC 5LP's on-board LED. When the reading given by the MPU is above .25 gee, the on-board LED (pin 2.1) is on. Otherwise, the LED is off.

To toggle the light, tilt the board with and against the x-axis marked on the board.

6.2 Example 2: 2-axis Accelerometer

This example shows the use of multiple axes of the accelerometer. It uses the MPU6050 and the PSoC 5LP's on-board LED. When the reading from the x-axis is above .25 gee and is higher than the reading from the y-axis, the light turns on. When the reading from the y axis is above .25 gee and is higher than the reading from the x-axis, the light blinks. Otherwise, the light turns off.

To change the light, tilt the board along and against the x and y axes.

6.3 Example 3: Kovid Konsole MPU6050 Showcase

This example uses the Kovid Konsole as built in class, and the MPU6050. It also makes use of the on-board LED (pin 2.1) and the on-board button (pin 2.2)

For this example, there are three modes:

- **Mode 1- Tilt-** In this mode, the acceleration of each axis is read and represented on the LED bank. From left to right, the LEDS represent acceleration in the +x, -x, +y, -y, +z, -z directions,

respectively. Leaving the Kovid Konsole flat on the table should light up the fifth LED from the left, indicating an acceleration in the +z direction (up)[1].

- **Mode 2- Spin-** In this mode, the angular velocity of each axis is read and represented on the LED bank. From left to right, the LEDS represent an angular velocity on the +x, -x, +y, -y, +z, -z directions, respectively. Note that angular velocity points along the axis of the rotation in accordance with the right-hand rule. Leaving the Kovid Konsole flat on the table should light up no LEDS.
- **Mode 3- Acceleration-** In this mode, the acceleration of each axis is read, and the magnitude of the sum of the vectors is represented on the LED bank. This magnitude is represented as a 6-bit value (0-63), with 32 representing 1 gee. The LSB is the left-most LED in the bank, and the MSB is the sixth LED from the left. Leaving the Kovid Konsole flat on the table should light up the sixth LED from the left, but may also cause the first LED to flicker due to measurement and computational inaccuracy.

On power-up, the Kovid Konsole will be in Mode 1- Tilt. Push the on-board button to cycle through these modes in order. The on-board LED represents the status of the on-board button, for the purpose of troubleshooting. The LED should remain on while the button is not depressed, and should turn off when the button is depressed.

Additionally, there are a couple things of note in this example. The enumeration of state variables, the mode_t, as well as the MPU data struct, MPU_DATA_t, help to keep the code readable and clean. For more complicated uses of readings from things like this, consider using structs and enumerations to keep things clear.

7 References/Resources

1. MPU6050 Chip Datasheet- I^2C Timing, Transaction charts- https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf
2. MPU6050 Register Map- Registers and config- https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

[1] "But wait! Gravity accelerates us down!" you might say. That's true in Newtonian mechanics, but General Relativity says that our geodesic points toward the center of the earth, and that the normal force from the earth causes us to deviate from that geodesic and not fall.