Interfacing MPU6050 with nRF52840DK

Kindly visit the *MPU6050* Gyro-Accelerometer interfacing program and the related documentation for more information.

The *nRF52840DK* is a versatile single board development kit for Bluetooth Low Energy, Bluetooth mesh, Thread, Zigbee, 802.15.4, ANT and 2.4*GHz* proprietary applications. Underneath sits the ARM Cortex - *M*4 processor with support of floating point. It comes with 4 user-programmable LEDs and 4 buttons and an on-board SEGGER J-Link debugger. It is usually powered by USB but there are other ways as well.

MPU6050 is a Micro Electro-mechanical system (MEMS) consisting of three-axis accelerometer and three-axis gyroscope. It helps us to measure velocity, orientation, acceleration, displacement and other motion like features.

For clarity, the connections are given below:

MPU6050	*nRF52840DK*
VCC	5*V*
GND	GND
SCL	*P*0.27
SDA	*P*0.26
INT	*P*1.11

Table 1: Pin connections

It is particularly specified in the **zephyr/boards/arm/nrf52840dk_nrf52811/nrf52840dk_nrf52811.dts** file about the pin connections to be used. The **.dts** is the *devicetree source* file, in other words, it specifies all the details of the peripherals incorporated by the development kit.

Here is the image of the **.dts** file where the i2c0 peripheral is described as a node of the devicetree.

```
10 &i2c0 {
11     compatible = "nordic,nrf-twi";
12     status = "okay";
13     /* Arduino compatible PINs */
14     sda-pin = <26>;
15     scl-pin = <27>;
16 };
17
18 &pwm0 {
19     status = "okay";
20     ch0-pin = <13>;
21     ch0-inverted;
22 };
```

Figure 10: Devicetree source for I2C peripheral

From the figure, it is evident that that the SDA and SCL pins have their default pin numbers. The source code for interfacing MPU6050 also specifies that logging the data from the sensor is interrupt driven. Here is the code snippet from **zephyr/samples/sensor/mpu6050/src/main.c**

```
#ifdef CONFIG_MPU6050_TRIGGER
static struct sensor_trigger trigger;
static void handle_mpu6050_drdy(const struct device *dev, struct sensor_trigger *trig)
{
        int rc = process_mpu6050(dev);
        if (rc != 0)
        {
                printf("cancelling trigger due to failure: %d\n", rc);
                (void)sensor_trigger_set(dev, trig, NULL);
                return;
        }
}
#endif /* CONFIG_MPU6050_TRIGGER */
void main(void)
{
        const char *const label = DT_LABEL(DT_INST(0, invensense_mpu6050));
        const struct device *mpu6050 = device_get_binding(label);
        if (!mpu6050)
        {
                printf("Failed to find sensor %s\n", label);
                return;
        }
        #ifdef CONFIG_MPU6050_TRIGGER trigger = (struct sensor_trigger)
        {
                .type = SENSOR_TRIG_DATA_READY,
                .chan = SENSOR_CHAN_ALL,
        };
        if (sensor_trigger_set(mpu6050, &trigger, handle_mpu6050_drdy) < 0)
        {
                printf("Cannot configure trigger\n");
                return;
        }
        printk("Configured for triggered sampling.\n");
        #endif
        while (!IS_ENABLED(CONFIG_MPU6050_TRIGGER))
        {
                int rc = process_mpu6050(mpu6050);
                if (rc != 0)
                        break;
```

```
                k_sleep(K_SECONDS(1));
        }
}
```

Earlier, it was specified that the interrupt pin of the sensor should be connected to the pin 11 of *GPIO*1 i.e. *P*1.11, which is described in the **zephyr/samples/sensor/mpu6050/boards/nrf52840dk_nrf529811** file. Here is the image which shows the same:

```
6  &i2c0 {
7      mpu6050@68 {
8          compatible = "invensense,mpu6050";
9          reg = <0x68>;
10         status = "okay";
11         label = "MPU6050";
12         int-gpios = <&gpio0 11 GPIO_ACTIVE_HIGH>;
13     };
14 };
```

Figure 11: Overlay file

It must be noted that **&gpioo** does not mean port 0 on the board. The **&gpioo** refers in general, to a gpio node on the board's device-tree structure. The figure shown below depicts the hardware connections of *MPU*6050 interfaced with *nRF*52840*DK*:

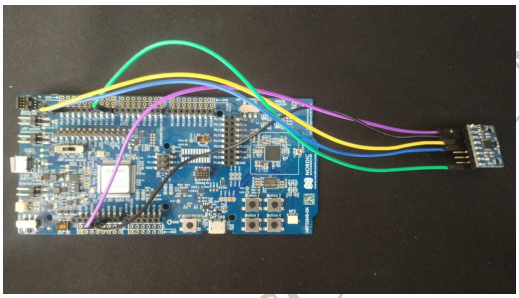

Figure 12: Pin Connections

Also, **west flash** or **west flash –erase** flashes the program onto the flash memory of the board. It need not display the values recorded on the screen. To observe the values, we need a serial monitor. **Minicom** is a good choice for the same. Note that settings of **minicom** need to be changed.

Run **minicom -s** in the terminal to change the default settings.

The COM port at which the board is detected has to be observed (visit /dev on local machine). The board is either reflected as **/dev/ttyACM*** or **/dev/ttyUSB***. Since the board consists of on board memory, /dev/sd**x** also appears in the directory **/dev**. That must not be chosen.

To verify that if the board has also been reflected as **/dev/sdx**, run **lsblk** on the terminal.

Some issues that may arise are:

- FATAL ERROR: build.ninja
  FATAL ERROR: command exited with status 1
  This can happen if the overlay file is incorrectly named. The overlay file has to be named as **<BOARD_NAME>.overlay**

- *** Booting Zephyr OS build zephyr-v2.6.0-692-gfadc5d15a3b5 ***
  Failed to find sensor *MPU*6050
  This is most likely due to incorrect hardware connections.

If built and flashed successfully, the output log should be:

Figure 13: Flashed on board

After verifying the above requirements, run
minicom -D /dev/ttyACM* -b 115200
to to observe the output from serial port onto the monitor. Here, 115200 is one of the standard baud rates at which the serial communication takes place. The output looks like:

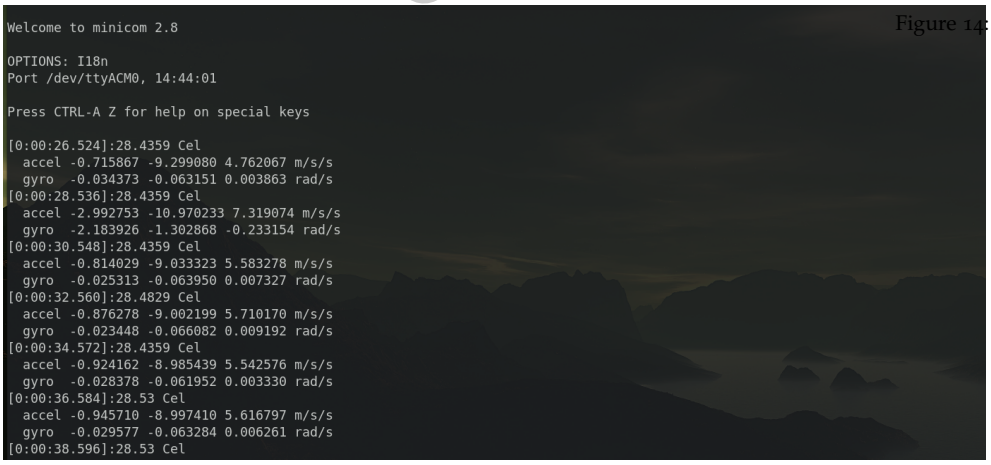

Figure 14: MPU6050 Output