synthiam.com

Reading Values From Imu Mpu6050 6dof

Hello everyone,
I had been trying for some time now and have finally come up with a code for taking values from the IMU. @DJ here it goes; I would also like you to include it in the next update of the SDK if possible.
So, to understand it step by step here is how it goes:

1. The latest SDK has some issues in reading method from the I2C port, so I searched over and used one of the early EZ_B dll's....

Last Updated: 1/31/2014

Step 1

Hello everyone,
I had been trying for some time now and have finally come up with a code for taking values from the IMU. @DJ here it goes; I would also like you to include it in the next update of the SDK if possible.
So, to understand it step by step here is how it goes:

1. The latest SDK has some issues in reading method from the I2C port, so I searched over and used one of the early EZ_B dll's. The main issue is that, in the recent sdk, the read method only takes the device address and not the register address. Now, those of you wondering what is the difference between the two, is that in the datasheet of an IMU the device address is always mentioned and the device address points to the starting register of the device. While on the other hand the register address is the register which you want to use. So, initially i declare some and initialize some variables:

```
byte default_addr = 104; //imu device address 0x68
```

```
int GYRO_XOUT_OFFSET;
    int GYRO_YOUT_OFFSET;
    int GYRO_ZOUT_OFFSET;

    long GYRO_XOUT_OFFSET_1000SUM = 0;
    long GYRO_YOUT_OFFSET_1000SUM = 0;
    long GYRO_ZOUT_OFFSET_1000SUM = 0;

    byte GYRO_XOUT_L;
    byte GYRO_XOUT_H;
    byte GYRO_YOUT_L;
    byte GYRO_YOUT_H;
    byte GYRO_ZOUT_L;
    byte GYRO_ZOUT_H;

    int ACCEL_XOUT;
    int ACCEL_YOUT;
    int ACCEL_ZOUT;

    byte ACCEL_XOUT_L;
    byte ACCEL_XOUT_H;
    byte ACCEL_YOUT_L;
    byte ACCEL_YOUT_H;
    byte ACCEL_ZOUT_L;
    byte ACCEL_ZOUT_H;

    float ACCEL_XANGLE;
    float ACCEL_YANGLE;
    float ACCEL_ZANGLE;

    float gyro_xsensitivity = 66.5f; //66.5 Dead on at last check
    float gyro_ysensitivity = 66.5f; //72.7 Dead on at last check
    float gyro_zsensitivity = 65.5f;

    int GYRO_XOUT;
    int GYRO_YOUT;
    int GYRO_ZOUT;

    float GYRO_XRATE;
    float GYRO_YRATE;
    float GYRO_ZRATE;

    float dt = 0.05f;

    float GYRO_XANGLE;
    float GYRO_YANGLE;
```

```
    float GYRO_ZANGLE;

    float COMPLEMENTARY_XANGLE;
    float COMPLEMENTARY_XANGLEPREV;
    float COMPLEMENTARY_YANGLE;
    float COMPLEMENTARY_YANGLEPREV;
```

1. So, enough with the talking; moving onto taking the actual values from the imu. The first part is to check whether the I2C link is functioning or not. Now how is that done? Here is the part when you open your IMU's datasheet/Register address sheet and search around for an "WHO_AM_I" named register address. For my IMU it is 0x75. This register contains the chips I2C address/device address, which in my case is 0x68.

```
byte ret = ezB_Connect1.EZB.I2C.Read(device_addr, 0x75);
```

If this communication is successful with an output of 0x68 in the "ret" variable, then you know your chip functions partially, congratulations!

1. The next step is to write to all the configuration registers:

```
public void setup()
{
//Sets sample rate to 8000/1+7 = 1000Hz
ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x19, 0x07 });
//Disable FSync, 256Hz DLPF
ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x1A, 0x00 });
//Disable gyro self tests, scale of 500 degrees/s
ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x1B, 0x08 });
//Disable accel self tests, scale of +-2g, no DHPF
ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x1C, 0x00 });
//Freefall threshold of |0mg|
ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x1D, 0x00 });
//Freefall duration limit of 0
ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x1E, 0x00 });
//Motion threshold of 0mg
ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x1F, 0x00 });
//Motion duration of 0s
ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x20, 0x00 });
//Zero motion threshold
ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x21, 0x00 });
//Zero motion duration threshold
ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x22, 0x00 });
//Disable sensor output to FIFO buffer
ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x23, 0x00 });
```

```
//AUX I2C setup
        //Sets AUX I2C to single master control, plus other config
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x24, 0x00 });
        //Setup AUX I2C slaves
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x25, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x26, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x27, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x28, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x29, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x2A, 0x00 });
```

```
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x2B, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x2C, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x2D, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x2E, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x2F, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x30, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x31, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x32, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x33, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x34, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x35, 0x00 });

        //MPU6050_RA_I2C_MST_STATUS //Read-only
        //Setup INT pin and AUX I2C pass through
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x37, 0x00 });
        //Enable data ready interrupt
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x38, 0x00 });

        //Slave out
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x63, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x64, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x65, 0x00 });
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x66, 0x00 });

        //More slave config
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x67, 0x00 });
        //Reset sensor signal paths
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x68, 0x00 });
        //Motion detection control
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x69, 0x00 });
        //Disables FIFO, AUX I2C, FIFO and I2C reset bits to 0
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x6A, 0x00 });
        //Sets clock source to gyro reference w/ PLL
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x6B, 0x02 });
        //Controls frequency of wakeups in accel low power mode plus the sensor standby modes
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x6C, 0x00 });

        //Data transfer to and from the FIFO buffer
        ezB_Connect1.EZB.I2C.Write(default_addr, new byte[] { 0x74, 0x00 });
  }
```

1. The setting up is done! I have also tried to outline what configuration changes I am making with every register write, in the comments above each line. The major part is done! Next is to read and convert the sensor data:

```
public void Calibrate_Gyros()
{
for (int i = 0; i < 50; i++)
{
GYRO_XOUT_H = ezB_Connect1.EZB.I2C.Read(default_addr, 0x43);
GYRO_XOUT_L = ezB_Connect1.EZB.I2C.Read(default_addr, 0x44);
GYRO_YOUT_H = ezB_Connect1.EZB.I2C.Read(default_addr, 0x45);
GYRO_YOUT_L = ezB_Connect1.EZB.I2C.Read(default_addr, 0x46);
GYRO_ZOUT_H = ezB_Connect1.EZB.I2C.Read(default_addr, 0x47);
GYRO_ZOUT_L = ezB_Connect1.EZB.I2C.Read(default_addr, 0x48);
```

```
GYRO_XOUT_OFFSET_1000SUM += ((GYRO_XOUT_H &amp;lt;&amp;lt; 8) | GYRO_XOUT_L);
            GYRO_YOUT_OFFSET_1000SUM += ((GYRO_YOUT_H &amp;lt;&amp;lt; 8) | GYRO_YOUT_L);
            GYRO_ZOUT_OFFSET_1000SUM += ((GYRO_ZOUT_H &amp;lt;&amp;lt; 8) | GYRO_ZOUT_L);

            //System.Threading.Thread.Sleep(1);
        }

        GYRO_XOUT_OFFSET = Convert.ToInt32(GYRO_XOUT_OFFSET_1000SUM / 50);
```

```
        GYRO_YOUT_OFFSET = Convert.ToInt32(GYRO_YOUT_OFFSET_1000SUM / 50);
        GYRO_ZOUT_OFFSET = Convert.ToInt32(GYRO_ZOUT_OFFSET_1000SUM / 50);
    }
```

```
public void Get_Accel_Values()
{
ACCEL_XOUT_H = ezB_Connect1.EZB.I2C.Read(default_addr, 0x3B);
ACCEL_XOUT_L = ezB_Connect1.EZB.I2C.Read(default_addr, 0x3C);
ACCEL_YOUT_H = ezB_Connect1.EZB.I2C.Read(default_addr, 0x3E);
ACCEL_YOUT_L = ezB_Connect1.EZB.I2C.Read(default_addr, 0x3D);
ACCEL_ZOUT_H = ezB_Connect1.EZB.I2C.Read(default_addr, 0x3F);
ACCEL_ZOUT_L = ezB_Connect1.EZB.I2C.Read(default_addr, 0x40);
```

```
ACCEL_XOUT = ((ACCEL_XOUT_H &amp;lt;&amp;lt; 8) | ACCEL_XOUT_L);
        ACCEL_YOUT = ((ACCEL_YOUT_H &amp;lt;&amp;lt; 8) | ACCEL_YOUT_L);
        ACCEL_ZOUT = ((ACCEL_ZOUT_H &amp;lt;&amp;lt; 8) | ACCEL_ZOUT_L);
    }
```

```
public void Get_Accel_Angles()
{
ACCEL_XANGLE = (float)(57.295 * Math.Atan((float)ACCEL_YOUT /
Math.Sqrt(Math.Pow((float)ACCEL_ZOUT, 2) + Math.Pow((float)ACCEL_XOUT, 2))));
ACCEL_YANGLE = (float)(57.295 * Math.Atan((float)-ACCEL_XOUT /
Math.Sqrt(Math.Pow((float)ACCEL_ZOUT, 2) + Math.Pow((float)ACCEL_YOUT, 2))));
}
```

```
public void Get_Gyro_Rates()
{
GYRO_XOUT_H = ezB_Connect1.EZB.I2C.Read(default_addr, 0x43);
GYRO_XOUT_L = ezB_Connect1.EZB.I2C.Read(default_addr, 0x44);
GYRO_YOUT_H = ezB_Connect1.EZB.I2C.Read(default_addr, 0x45);
GYRO_YOUT_L = ezB_Connect1.EZB.I2C.Read(default_addr, 0x46);
GYRO_ZOUT_H = ezB_Connect1.EZB.I2C.Read(default_addr, 0x47);
GYRO_ZOUT_L = ezB_Connect1.EZB.I2C.Read(default_addr, 0x48);
```

```
GYRO_XOUT = ((GYRO_XOUT_H &amp;lt;&amp;lt; 8) | GYRO_XOUT_L) - GYRO_XOUT_OFFSET;
        GYRO_YOUT = ((GYRO_YOUT_H &amp;lt;&amp;lt; 8) | GYRO_YOUT_L) - GYRO_YOUT_OFFSET;
        GYRO_ZOUT = ((GYRO_ZOUT_H &amp;lt;&amp;lt; 8) | GYRO_ZOUT_L) - GYRO_ZOUT_OFFSET;

        GYRO_XRATE = (float)GYRO_XOUT / gyro_xsensitivity;
        GYRO_YRATE = (float)GYRO_YOUT / gyro_ysensitivity;
        GYRO_ZRATE = (float)GYRO_ZOUT / gyro_zsensitivity;

        GYRO_XANGLE += GYRO_XRATE * dt;
        GYRO_YANGLE += GYRO_YRATE * dt;
        GYRO_ZANGLE += GYRO_ZRATE * dt;
    }
```

```
public void Zero_Sensors()
```

{
float BUFFER_XANGLE = 0;
float BUFFER_YANGLE = 0;
int x = 0;

```
for (x = 0; x &lt; 100; x++)
        {
            Get_Accel_Values();
            Get_Accel_Angles();
            BUFFER_XANGLE += ACCEL_XANGLE;
            BUFFER_YANGLE += ACCEL_YANGLE;
            //System.Threading.Thread.Sleep(1);
        }

        COMPLEMENTARY_XANGLE = (float)(BUFFER_XANGLE / 100.0);
        COMPLEMENTARY_YANGLE = (float)(BUFFER_YANGLE / 100.0);
        GYRO_XANGLE = (float)(BUFFER_XANGLE / 100.0);
        GYRO_YANGLE = (float)(BUFFER_YANGLE / 100.0);
    }
```

1. Finally call the methods:

```
  while (true)
{
if (i == 0)     //only do this once
{
byte ret = ezB_Connect1.EZB.I2C.Read(default_addr, 0x75);
listBox1.Items.Add(ret);
```

```
i = 1;
                setup();
                Calibrate_Gyros();
            }

            Get_Gyro_Rates();
            Zero_Sensors();

        }
```

The code is still under some fine refinements such as adding the complementary or kalman filter to it to reduce the error. Once I write those too, I will include it as well. I am attaching the dll i have used along with the register description file of the IMU.
EZ_B(dll).zip
RegisterMapandDescriptions.pdf
Enjoy!