UofG Masters DSP Project

Ben Frazer*

December 29, 2021

Contents

1 Foreword **1**

2 Reference Material **2**

3 ESP 32 Reference **2**
3.1 Pin Layout 2

4 SYSEX Command Syntax (I2C ect) **3**
4.1 Aditional Notes 3
4.1.1 Pyfirmata expected Baudrate 3
4.2 Frimata protocol **SYSEX** Commands 3
4.3 SYSEX I2C_REQUEST (I2C request) 4
4.3.1 Frimata protocol **SYSEX I2C_Request** structure 4
4.3.2 Firmware implementation 5
4.3.3 **I2CFirmata.cpp** implementation: 5
4.4 SYSEX I2C_REPLY Returned Data structure 7
4.4.1 Returned Data Structure 7
4.4.2 Converting back from 7 bit bytes 7

5 MPU 6050 **8**
5.1 MPU-6050 I2C 8
5.1.1 I2C Address 8
5.1.2 Sensor Data Registers 8

1 Foreword

This is the source code for the masters year DSP project on real time IIR filtering at the university of Glasgow.
This read-me is currenly acting as a notebook for this project and should not be seen as documentation as such.

Note: Primarily these notes relate to the implementation of I2C in Firmata as well as the relevant registers of the MPU-6050 however it seems like this sensor will not be used in favour of the ADXL335 since getting I2C working with firmata is proving to be a nightmare. I will however leave my notes on the matter in here for future reference.

*2704250F@student.gla.ac.uk

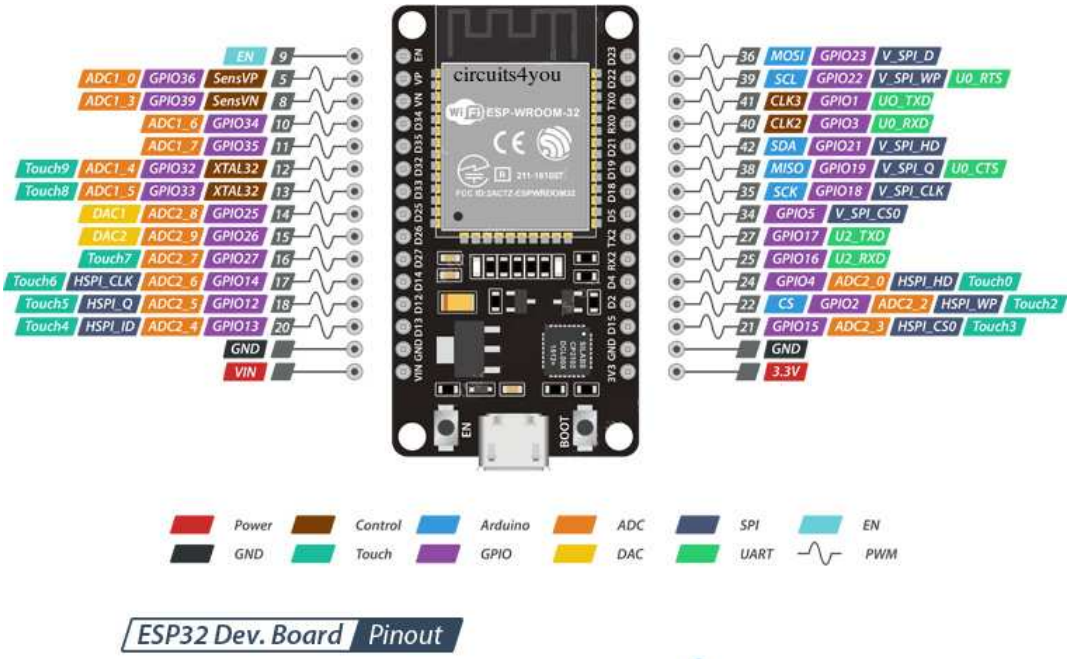
2 Reference Material

- MPU-6000/6050 datasheet
- MPU-6050/6050 Registers
- Assessment Sheet

3 ESP 32 Reference

3.1 Pin Layout

From: PinLayout



4 SYSEX Command Syntax (I2C ect)

4.1 Aditional Notes

4.1.1 Pyfirmata expected Baudrate

57600

4.2 Frimata protocol SYSEX Commands

- I2C_REPLY -> 119
- I2C_REQUEST -> 118

- REPORT_ANALOG -> 192
- REPORT_Digital -> 208
- QUERY_FIRMWARE -> 118
- START_SYSEX -> 240
- END-SYSEX -> 247
- EXTEND_ANALOG-> 111
- STRING_DATA -> 113
- CAPABILITY_RESPONSE -> 108
- SYSEX_NON_REALTIME -> 126
- SYSEX_REALTIME -> 127
- REPORT_FIRMWARE -> 121
- SAMPLING_INTERVAL -> 122
- REPORT_VERSION -> 249

4.3 SYSEX I2C_REQUEST (I2C request)

Any **SYSEX** command is first handled by the **handleSysex** method which then conditionally routes **I2C_REQUEST** commands to the **handleI2CRequest** method of **I2CFirmata**

4.3.1 Frimata protocol SYSEX I2C_Request structure

From:`https://github.com/firmata/protocol/blob/master/i2c.md`

1. START_SYSEX (0xF0)
2. I2C_REQUEST (0x76)

3. slave address (LSB)

4. slave address (MSB) + read/write and address mode bits

    - {bit 7: always 0}
    - {bit 6: auto restart transmission, 0 = stop (default), 1 = restart}
    - {bit 5: address mode, 1 = 10-bit mode}
    - {bits 4-3: read/write, 00 = write, 01 = read once, 10 = read continuously, 11 = stop reading}
    - {bits 2-0: slave address MSB in 10-bit mode, not used in 7-bit mode}

5. data 0 (Target slave register LSB)

6. data 0 (Target slave register MSB)

7. data 1 (Number of bytes to extract LSB)

8. data 1 (Number of bytes to extract MSB)

4.3.2 Firmware implementation

As far as I can tell, all I2C sensor data requested via **SYSEX** command is handled in the same way by the same method on the firmware side (**readAndReportData**). This means, that there is no way to tell which I2C reply is which unless you have only just sent a single request.

Note: all data buffers handled by this and other functions prior to sending are handled as simple byte arrays conversion to 7 bit bytes is handled in the **sendSysex** method

I2CFirmata.cpp implementation:

```
void I2CFirmata::report(bool elapsed)
{
// report i2c data for all device with read continuous mode enabled
if (queryIndex > -1) {
    for (byte i = 0; i < queryIndex + 1; i++) {
    readAndReportData(query[i].addr, query[i].reg, query[i].bytes,
    ↪ query[i].stopTX);
    }
}
}
```

4.3.3 I2CFirmata.cpp implementation:

```
void I2CFirmata::handleI2CRequest(byte argc, byte* argv)
{
  byte mode;
  byte stopTX;
  byte slaveAddress;
  byte data;
  int slaveRegister;
  mode = argv[1] & I2C_READ_WRITE_MODE_MASK;
  if (argv[1] & I2C_10BIT_ADDRESS_MODE_MASK) {
    Firmata.sendString(F("10-bit addressing not supported"));
    return;
  }
  else {
    slaveAddress = argv[0];
  }

  // need to invert the logic here since 0 will be default for client
  // libraries that have not updated to add support for restart tx
  if (argv[1] & I2C_END_TX_MASK) {
    stopTX = I2C_RESTART_TX;
  }
  else {
    stopTX = I2C_STOP_TX; // default
  }

  switch (mode) {
      ... // more here
  case I2C_READ:
    if (argc == 6) {
      // a slave register is specified
      slaveRegister = argv[2] + (argv[3] << 7);
      data = argv[4] + (argv[5] << 7);  // bytes to read
    }
    else {
      // a slave register is NOT specified
      slaveRegister = I2C_REGISTER_NOT_SPECIFIED;
      data = argv[2] + (argv[3] << 7);  // bytes to read
    }
    readAndReportData(slaveAddress, (int)slaveRegister, data, stopTX);
    break;
  case I2C_READ_CONTINUOUSLY:
    if ((queryIndex + 1) >= I2C_MAX_QUERIES) {
      // too many queries, just ignore
      Firmata.sendString(F("too many queries"));
      break;
    }
    if (argc == 6) {
```

```
      // a slave register is specified
      slaveRegister = argv[2] + (argv[3] << 7);
      data = argv[4] + (argv[5] << 7);  // bytes to read
    }
    else {
      // a slave register is NOT specified
      slaveRegister = (int)I2C_REGISTER_NOT_SPECIFIED;
      data = argv[2] + (argv[3] << 7);  // bytes to read
    }
    queryIndex++;
    query[queryIndex].addr = slaveAddress;
    query[queryIndex].reg = slaveRegister;
    query[queryIndex].bytes = data;
    query[queryIndex].stopTX = stopTX;
    break;
    ...// more here
```

4.4 SYSEX I2C_REPLY Returned Data structure

Data returned from an I2C request is always Prefixed with the **SYSEX** command: **SYSEX_I2C_REPLY**

4.4.1 Returned Data Structure

From:

1. START_SYSEX (0xF0)
2. I2C_REPLY (0x77)
3. slave address (LSB)
4. slave address (MSB)
5. register (LSB)
6. register (MSB)
7. data 0 (LSB) - presumably the contents of the register
8. data 0 (MSB)

4.4.2 Converting back from 7 bit bytes

The payload is always preceded with a **SYSEX** command header. This is NOT in a 7 bit form. The subsequent message payload however is.
The **sendSysex** method of the Firmata firmware packages each byte of payload data as two 7 bit bytes.

5 MPU 6050

5.1 MPU-6050 I2C

5.1.1 I2C Address

This depends on how the AD0 of the MPU 6050 is set.

AD0 = 0	AD0 = 1
1101000	1101001

I believe on the sparkfun breakout board AD0 = 0

5.1.2 Sensor Data Registers

From:MPU-6050 Register Datasheet

1. Accelerometer
   - MPU 60X0 Registers Pg 29.
   - 16 bit twos complement

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2g	16384 LSB/g
1	±4g	8192 LSB/g
2	±8g	4096 LSB/g
3	±16g	2048 LSB/g

2. Gyroscope

- MPU 60X0 Registers Pg 31.
- 16 bit twos complement

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT[7:0]							

3. Temp sensor
   - MPU 60X0 Registers Pg 31.
   - 16 bit signed value

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
41	65	TEMP_OUT[15:8]							
42	66	TEMP_OUT[7:0]							

Description:

Temp Formula:
$T_{degC} = (T_{Reg})/340 + 36.53$