Sensor Shield vs Breakout Boards

Sensors may be interfaced to a Lapis microcontroller via an independent connection (breakout board), or through a shared interface/connection (shield). The main difference lies in the jumper selection:

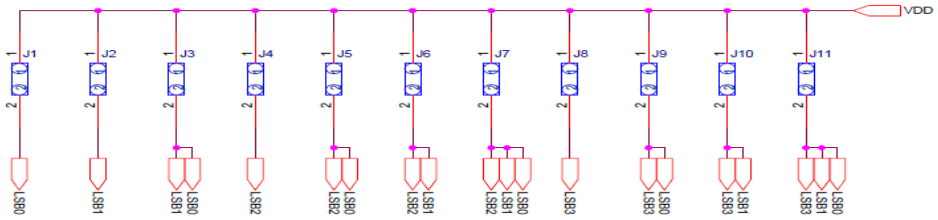


Figure 1 - Sensor selection by jumpers

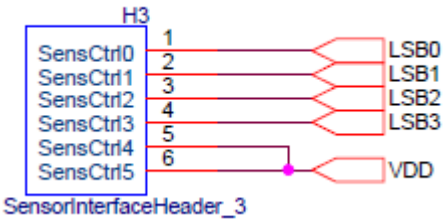


Figure 3 - Sensor selection in shield (notice pins 5 and 6)

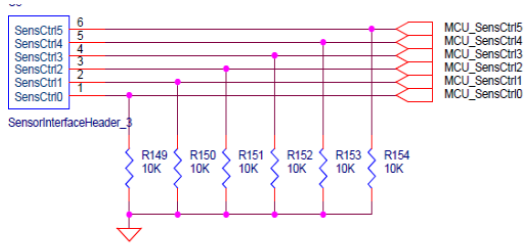


Figure 2 - Sensor selection in break out board

The main difference to notice is that the shield, given the forced connection to VDD on LSB4 and LSB 5, only has the option of defining 16 sensors 110000(0x30) to 111111(0x3F), while the break boards are not limited to this range (see picture). Since the firmware needs to be prepared for both cases, it is desired to define all the breakout options in the mutually exclusive range, 000000 (0x00) to 101111(0x2F), which gives room to ID 48 break out boards.

The current ID case statement is presented below. Any extra sensors, whether a breakout board or part of a shield, should be added to the ranges stated before:

Current ID List

1:	Hall_Effect_Sensors_1()		10:	UV_Sensor_10()
2:	Hall_Effect_Sensors_2()		15:	KMX122() Accel MEM Sensors
5:	Ambient_Light_Sensor_5()		16:	KMX062() Accel + Mag MEM Sensors
6:	Ambient_Light_Sensor_6()		20:	Temperature_Sensor_20();
7:	Ambient_Light_Sensor_7()		21:	Temperature_Sensor_21();
8:	Ambient_Light_Sensor_8()		22:	Temperature_Sensor_22()
9:	Ambient_Light_Sensor_9()		23:	Temperature_Sensor_23()

Skipped: 3(pressure),4(color),11(ALS/Prox),12(gyro),13,14,17,18,19

The main task at hand is to incorporate 4 more sensors to the platform. For quick prototyping purposes, these will be added to the firmware as breakout boards, though any adjustment would just require initializing a relevant case statement in the right range. The sensors to be added are the following:

1. Pressure Sensor (BM1383GLV)
2. Gyro (KXG03)
3. ALS/Proximity Sensor (RPR-0521)
4. Color Sensor (BH1745NUC)

Pressure Sensor (BM1383GLV)

This is an I2C device that will leverage many of the existing functions for the protocol used by other sensors. Some of the variables and methods of interest are:

Initialization and Operation routines:

```
void MainOp_Pressure_Sensor_3();
void Init_Pressure_Sensor_3();
```

I2C functions to configure and retrieve data:

I2C_Read, I2C_Write

I2C device address of BM1383GLV

```
const unsigned char BM1383GLV_I2C_ADDR                  = 0x5du;
```

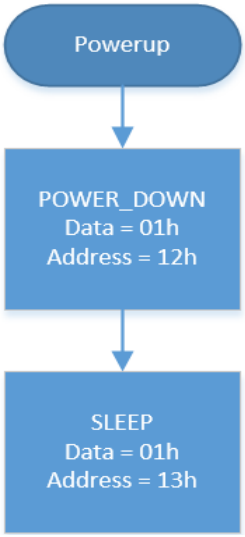
Register addresses of BM1383GLV

```
const unsigned char BM1383GLV_ID                        = 0x10u;
const unsigned char BM1383GLV_RESET_CONTROL             = 0x11u;
const unsigned char BM1383GLV_POWER_DOWN                = 0x12u;
const unsigned char BM1383GLV_SLEEP                     = 0x13u
const unsigned char BM1383GLV_MODE_CONTROL              = 0x14u
const unsigned char BM1383GLV_TEMPERATURE_MSB           = 0x1au;    //data read starts here
const unsigned char BM1383GLV_TEMPERATURE_LSB           = 0x1bu;
const unsigned char BM1383GLV_PRESSURE_MSB              = 0x1cu;
const unsigned char BM1383GLV_PRESSURE_LSB              = 0x1du;
const unsigned char BM1383GLV_PRESSURE_LSB              = 0x1eu;
```

Configuration data registers

```
const unsigned char BM1383GLV_POWER_ON                  = 0x01u;
const unsigned char BM1383GLV_POWER_OFF                 = 0x00u;
const unsigned char BM1383GLV_SLEEP_ACT                 = 0x01u;
const unsigned char BM1383GLV_SLEEP_SLP                 = 0x00u;
const unsigned char BM1383GLV_MODE_CONTROL_AVGN         = 0x02u;    //12 ms delay
const unsigned char BM1383GLV_MODE_CONTROL_MODE         = 0x04u;    //sampling 200ms
const unsigned char BM1383GLV_MODE_CONTROL_REG  = (BM1383GLV_MODE_CONTROL_AVGN <<
5)+ BM1383GLV_MODE_CONTROL_MODE;
```

Initialization: The sensor follows a very simple power up routine summarized by the writing of the registers described in the flow chart. Also, since the speed of the data acquisition is not so important (for our demo application), we are initializing the 'MODE_CONTROL' register at this stage, and simply read the relevant registers when the information needs to be retrieved.

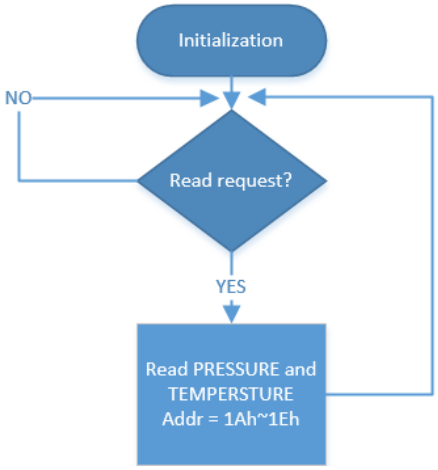


The statement to do this is:

MODE_CONTROL 14h = 01h (refer to pg 8 of data sheet for other modes)

```
I2C_Write(BM138GLV_ADDR,  &BM1383GLV_MODE_CONTROL,  1,
&BM1383GLV_MODE_CONTROL_REG, 1);
```

Operation:



The reading of the relevant data is done via a burst I2C read of 5 relevant registers. All bytes need to be interpreted according to the data sheet to make sure the information makes sense. The parsed info is printed via UART.

Color Sensor (BH1745NUC)

Initialization and Operation routines:
void MainOp_Color_Sensor_4();
void Init_Color_Sensor_4();

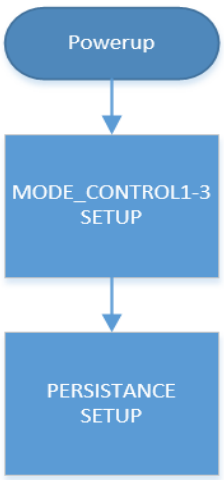
I2C functions to configure and retrieve data:
I2C_Read, I2C_Write

I2C device address of BH1745
const unsigned char BH1745 = 0x38u;

Configuration register addresses of BM1383GLV
const unsigned char BH1745_SYSTEM_CONTROL = 0x40u;
const unsigned char BH1745_MODE_CONTROL1 = 0x41u;
const unsigned char BH1745_MODE_CONTROL2 = 0x42u;
const unsigned char BH1745_MODE_CONTROL3 = 0x43u;
const unsigned char BH1745_PERSISTENCE = 0x61u;
const unsigned char BH1745_INTERRUPT = 0x60u;

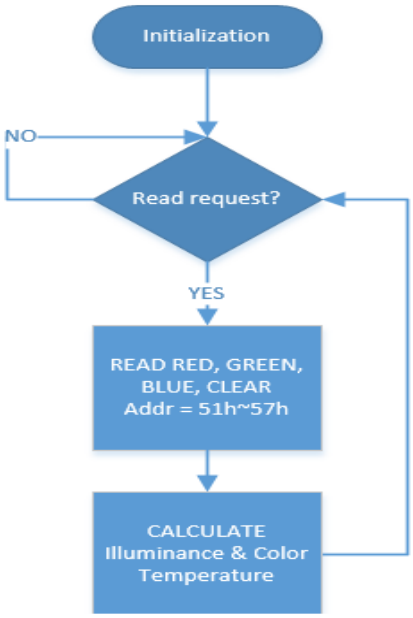
Register addresses of BM1383GLV
const unsigned char BH1745_MANUFACTURER_ID = 0x92u;
const unsigned char BH1745_RED_DATA_LSBs = 0x50u;
const unsigned char BH1745_RED_DATA_MSBs = 0x51u;
const unsigned char BH1745_GREEN_DATA_LSBs = 0x52u;
const unsigned char BH1745_GREEN_DATA_MSBs = 0x53u;
const unsigned char BH1745_BLUE_DATA_LSBs = 0x54u;
const unsigned char BH1745_BLUE_DATA_MSBs = 0x55u;

Initialization: The sensor is initialized according to the parameters below. The purpose is to setup a reading to calculate Illuminance and Color temperature.



```
I2C_Write(0x38u, & BH1745_PERSISTENCE, 1, &0x03, 1);        // see data sheet pg. 9
I2C_Write(0x38u, & BH1745_MODE_CONTROL1, 1, 0x00, 1);       //000 : 160msec
I2C_Write(0x38u, & BH1745_MODE_CONTROL2, 1, 0x92, 1);       //16x gain, RGBC_EN
I2C_Write(0x38u, & BH1745_MODE_CONTROL3, 1, 0x02, 1);       //default 0x02
```

Operation: The reading of the relevant data is done via a burst I2C read of 8 relevant registers. All bytes need to be interpreted according to the data sheet to make sure the information makes sense. This info is later processed by the algorithm presented in the"BH1745 Reference Calculation formula" pdf, which will provide Illuminance and Temperature values. The values are printed via UART.



Below is an example of the reading and interpretation of the bits corresponding to the red color:

```
I2C_Read(sensor_addr[i], &RED_DATA_LSBs, 1, uniRawSensorOut._ucharArr, 8);

rgb_s1_R[sumIndex] += ((int)uniRawSensorOut._ucharArr[0] |
                       ((int)uniRawSensorOut._ucharArr[1])<<8);
```

ALS/Proximity Sensor (RPR-0521)

RPR-0521RS is a module which integrates optical proximity, digital ambient light sensor IC, and infrared LED (IrLED). Proximity sensor (PS) part detects the human or object approaching by the reflection of IrLED light. Ambient light sensor (ALS) part detects the wide range of illumination; from the dark environment to the direct sun light. The illuminant intensity of LCD display and keypad can be adjusted by using RPR-0521RS. It enables lowering current consumption and/or improving the visibility under the bright environment.

Initialization and Operation routines:

```
void MainOp_Prox_Sensor_11();
void Init_Prox_Sensor_11();
```

I2C functions to configure and retrieve data:

I2C_Read, I2C_Write

I2C device address of RPR0521

```
const unsigned char RPR0521                 = 0x38u;
```

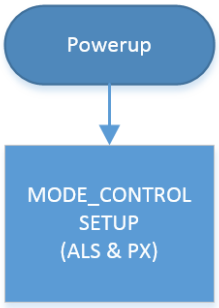
Configuration register addresses of RPR0521

```
const unsigned char Prox_ModeCTR            = 0x41u;
```

Register addresses of RPR0521

```
const unsigned char RPR0521_Manufact_ID         = 0x92u;
const unsigned char RPR0521_Prox_PS_LSB         = 0x44u;
const unsigned char RPR0521_Prox_PS_MSB         = 0x45u;
const unsigned char RPR0521_ALS0_LSB            = 0x46u;
const unsigned char RPR0521_ALS0_MSB            = 0x47u;
const unsigned char RPR0521_ALS1_LSB            = 0x48u;
const unsigned char RPR0521_ALS1_MSB            = 0x49u;
```

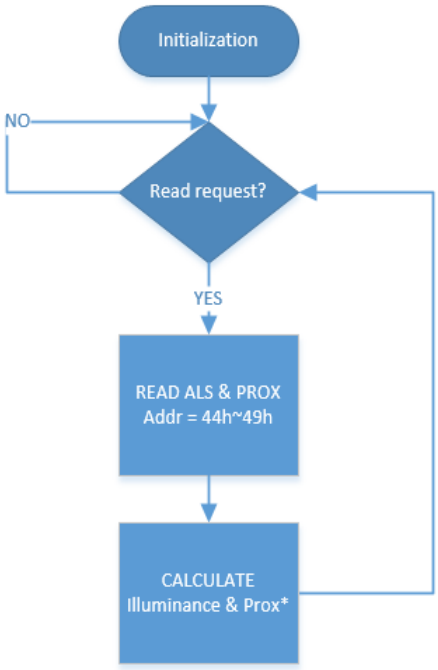
Initialization



This step sets necessary parameters for detection are set: enable (ALS & PX), operating mode, and measurement time. See datasheet pg 16 for more details. The statement below summarizes the initialization step.

```
I2C_Write(0x38, & RPR0521_Prox_ModeCTR, 1, 0xC6, 1);        //0xC6: ALS=PS= enabled & 100ms
```

Operation



The operation step reads the relevant registers holding ALS and Prox information. Appropriate calculations follow to make sure the bytes are interpreted correctly. The statements below summarize the operation step.

```
I2C_Read(RPR0521_I2C_ADDR, & RPR0521_Prox_PS_LSB, 1, uniRawSensorOut._ucharArr, 4);      //burst read
//I2C_Read(0x38u, & RPR0521_Prox_PS_MSB, 1, uniRawSensorOut._ucharArr, 1);
//I2C_Read(0x38u, & RPR0521_ALS0_LSB, 1, uniRawSensorOut._ucharArr, 1);
//I2C_Read(0x38u, & RPR0521_ALS0_MSB, 1, uniRawSensorOut._ucharArr, 1);

IR Data….not relevant
//I2C_Read(0x38u, & RPR0521_ALS1_LSB, 1, uniRawSensorOut._ucharArr, 1);
//I2C_Read(0x38u, & RPR0521_ALS1_MSB, 1, uniRawSensorOut._ucharArr, 1);
```

Gyro (KXG03)

The KXG03 sensor consists of a tri-axial micro machined gyroscope plus a tri-axial accelerometer. It also features flexible user programmable gyroscope full scale ranges of ±256, ±512, ±1024, and ±2048º/sec and user-programmable ±2g/±4g/±8g/±16g full scale range for the accelerometer.

Initialization and Operation routines:
void Init_KXG03_Sensor_12();
void MainOp_KXG03_Sensor_12()

I2C functions to configure and retrieve data:
I2C_Read, I2C_Write

I2C device address of KXG03 = KXG03_I2C_ADDR = 0x4E or 0x4F (depending on LSB pin)

//Configuration Registers

```
const unsigned char KXG03_STBY_REG          =       0x29;
const unsigned char KXG03_CTRL_REG1         =       0x2A;
const unsigned char KXG03_CTRL_REG2         =       0x2B;
const unsigned char KXG03_ODCTRL            =       0x2C;
```

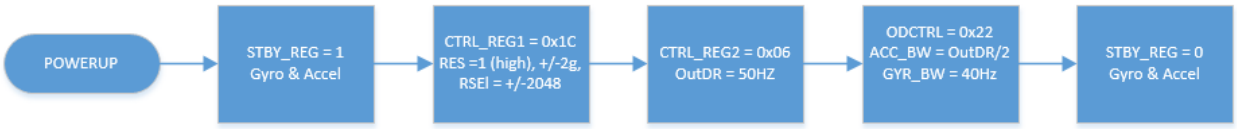
//Configuration Data

```
const unsigned char KXG03_STBY_SET                  =       0x7f;
const unsigned char KXG03_CTRL_REG1_CFDAT           =       0x1c;
const unsigned char KXG03_CTRL_REG2_CFDAT           =       0x06;   //OutDR = 50HZ
const unsigned char KXG03_ODCTRL_CFDAT              =       0x22;   //GYRBW = 40Hz
```

//Relevant register Addresses

```
const unsigned char KXG03_GYRO_XOUT_L       =       0x00;
const unsigned char KXG03_GYRO_XOUT_H       =       0x01;
const unsigned char KXG03_GYRO_YOUT_L       =       0x02;
const unsigned char KXG03_GYRO_YOUT_H       =       0x03;
const unsigned char KXG03_GYRO_ZOUT_L       =       0x04;
const unsigned char KXG03_GYRO_ZOUT_H       =       0x05;
const unsigned char KXG03_ACC_XOUT_L        =       0x06;
const unsigned char KXG03_ACC_XOUT_H        =       0x07;
const unsigned char KXG03_ACC_YOUT_L        =       0x08;
const unsigned char KXG03_ACC_YOUT_H        =       0x09;
const unsigned char KXG03_ACC_ZOUT_L        =       0x0A;
const unsigned char KXG03_ACC_ZOUT_H        =       0x0B;
```

Initialization:

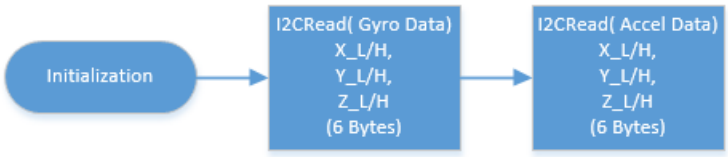


The flow presented above takes place within the Init_KXG03_Sensor_12() function:

```
I2C_Write(KXG03_I2C_ADDR, &KXG03_STBY_REG, 1, &KXG03_STBY_SET, 1);
I2C_Write(KX022_I2C_ADDR, &KXG03_CTRL_REG1, 1, &KXG03_CTRL_REG1_CFDAT, 1);
I2C_Write(KX022_I2C_ADDR, &KXG03_CTRL_REG2, 1, &KXG03_CTRL_REG2_CFDAT, 1);
I2C_Write(KX022_I2C_ADDR, &KXG03_ODCTRL, 1, &KXG03_ODCTRL_CFDAT, 1);

// Set accelerometer to operating mode (PC1=1)
uniRawSensorOut._uchar = (unsigned char)(KXG03_STBY_SET&0xfc);
I2C_Write(KX022_I2C_ADDR, &KX022_CNTL1, 1, &uniRawSensorOut._uchar, 1);
```

Operation:



The operation flow takes place in the MainOp_KXG03_Sensor_12() function:

```
I2C_Read(KXG03_I2C_ADDR, &KXG03_GYRO_XOUT_L, 1, uniRawSensorOut._ucharArr, 6);
I2C_Read(KXG03_I2C_ADDR, &KXG03_ACC_XOUT_L, 1, uniRawSensorOut._ucharArr, 6);
```