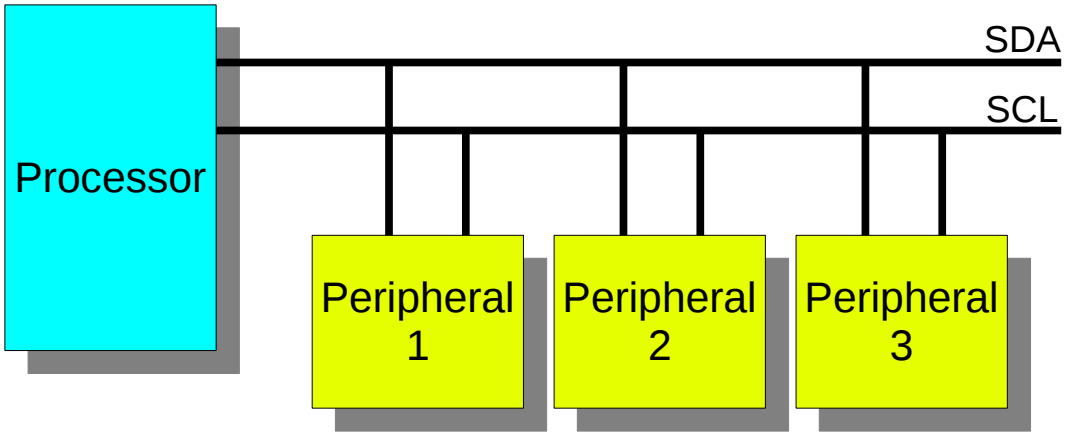
Magnetometer and I2C Bus

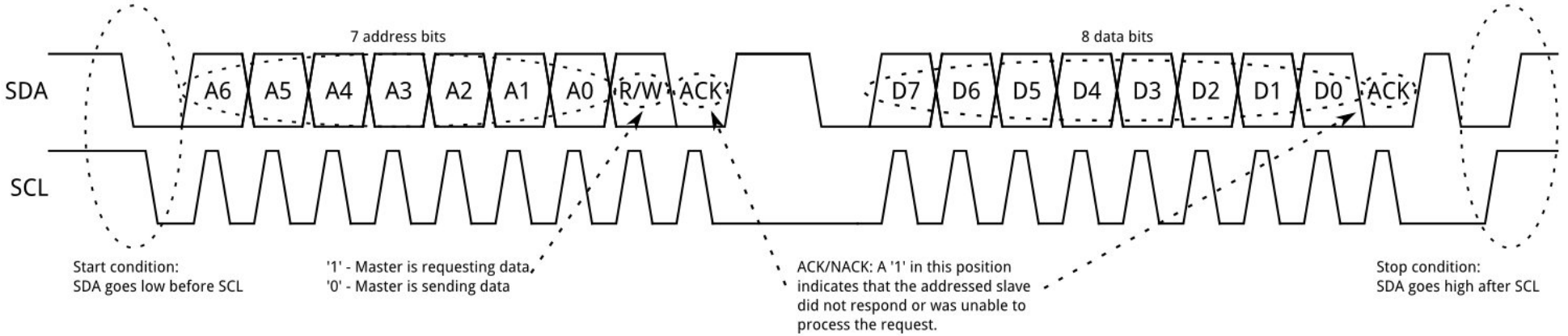
I2C Bus

- I2C stands for Inter-Integrated Circuit. It is a serial type interface requiring only two signals, a clock signal and a data signal.
- The I2C bus is typically used to interface with sensors and peripheral devices not needing to communicate at high speeds. The standard data rate is 100 Kilobits per second.
- Multiple devices can be connected to a single I2C bus. The processor is the controller and all the connected devices are peripherals. The processor is also called the master and the peripherals are slaves. Each slave has a unique address.



I2C Bus

- The clock signal is labeled SCL. This signal is used to control the flow of the data bits.
- The data signal is called SDA. This carries the data serially.
- The diagram below shows how a data transfer occurs.
- The data transfer protocol is for the master to first send out a device address. This is a 7 bit number followed by a bit indicating if the next byte is to be written to a slave or read from a slave.
- The SCL signal toggles for each bit sent.

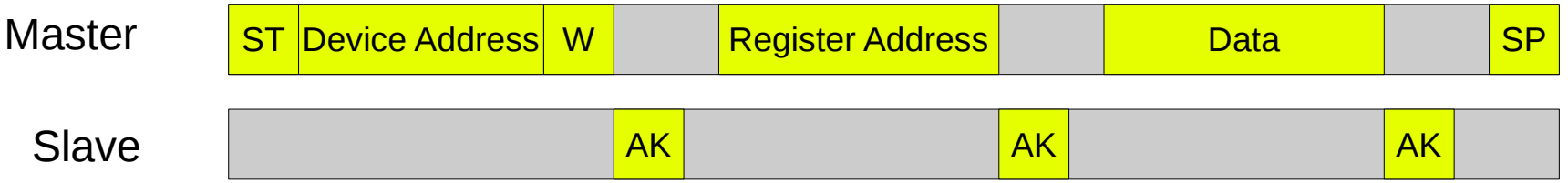


I2C Sequence

- Every device on the I2C bus has a unique 7-bit address. The magnetometer address is 0x0d.
- The I2C operation for writing to a register is:

    1. Send Start sequence by keeping SCL high and changing SDA from high to low (ST)
    2. Send the device address
    3. Send the register address
    4. Send the stop sequence by changing SDA from low to high while SCL is high first. (SP)

    - All Bytes sent are acknowledge by the slave.



I2C Sequence

- The I2C operation for reading a register is:

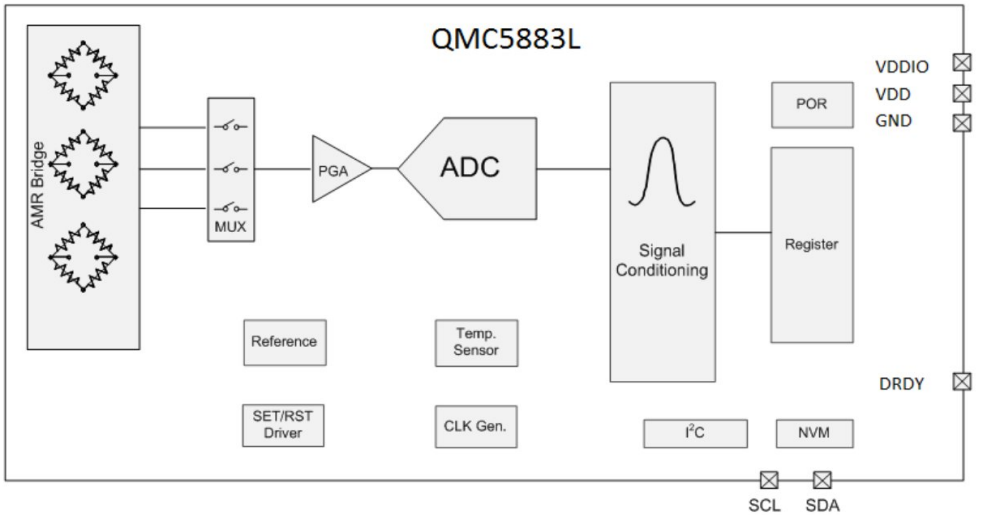
  1. Send Start sequence by keeping SCL high and changing SDA from high to low (ST)
  2. Send the device address
  3. Send the register address
  4. Send a repeat start sequence with a read indication (SR)
  5. Read in 6 bytes
  6. Send a stop sequence (SP)

Magnetometer Sensor

- The QMC5883L is a 3-axis magnetic sensor called a magnetometer or compass. It can measure up to +/- 8 gauss or +/-2 gauss.
- It uses an internal 16-bit ADC and can provide heading accuracy of 1 to 2 degress.
- It uses the I2C bus for communications.
- It is sensitive enough to detect earth's magnetic field so it can be used as a compass. Being this sensitive, it will also be sensitive to other magnetic fields in the area which can distort its measurement of earth's magnetic field.

Magnetometer Sensor

- The sensor uses magneto-resistive devices to detect magnetic fields. The devices use a property of material that changes its resistance in the presence of a magnetic field. Lord Kelvin discovered this when he noticed a slight resistance change in iron when exposed to a magnetic field.
- The diagram to the right shows the three sensors in what is called a wheatstone bridge circuit. This circuit is used widely in sensors.



Magnetometer Configuration

- The magnetometer has a group of registers which is special memory for holding data and configuration information. Each register has a unique address. The I2C transaction always starts with specifying the register address and then the data transaction.
- The magnetometer has 13 registers. The following registers shown are what will be used.

Register Name	Address
X LSB	0x00
X MSB	0x01
Y LSB	0x02
Y MSB	0x03
Z LSB	0x04
Z MSB	0x05
Control Reg 1	0x09
Control Reg 2	0x0a
Set/Reset Period	0x0b

Register Description

- The registers with the data start at address 0x00. They are formatted as shown below. Since the data for each axis is 16 bits, two registers are required to hold the data. The upper half called the most significant byte or MSB contains the 8 most significant bits of the data.
- The second register holds the least significant byte or LSB and contains the lower 8 bits.
- The data range is 0xFFFF 0x7FFF. It is two compliment so when the most significant bit is set, the value is negative.

Address	Reg	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00	LSB	D7	D6	D5	D4	D3	D2	D1	D0
0x01	MSB	D15	D14	D13	D12	D11	D10	D9	D8

Control Register

- Configuration register 1 is used to set the number of samples averaged, the data rate and the measurement range.
- The tables below show the settings for each bit of the register.

Bits 5,4	Full Scale
00	2 gauss
01	8 gauss

Bits 1,0	Mode
00	Standby
01	Continuous

Bits 7,6	Over Sample Ratio
00	512
01	256
10	12
11	64

Bits 3,2	Output Data Rate
00	10 Hz
01	50 Hz
10	100 Hz
11	200 Hz

Reg	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x09	OSR[1]	SDR[0]	RNG[1]	RNG[0]	ODR[1]	ODR[0]	MODE[1]	MODE[0]

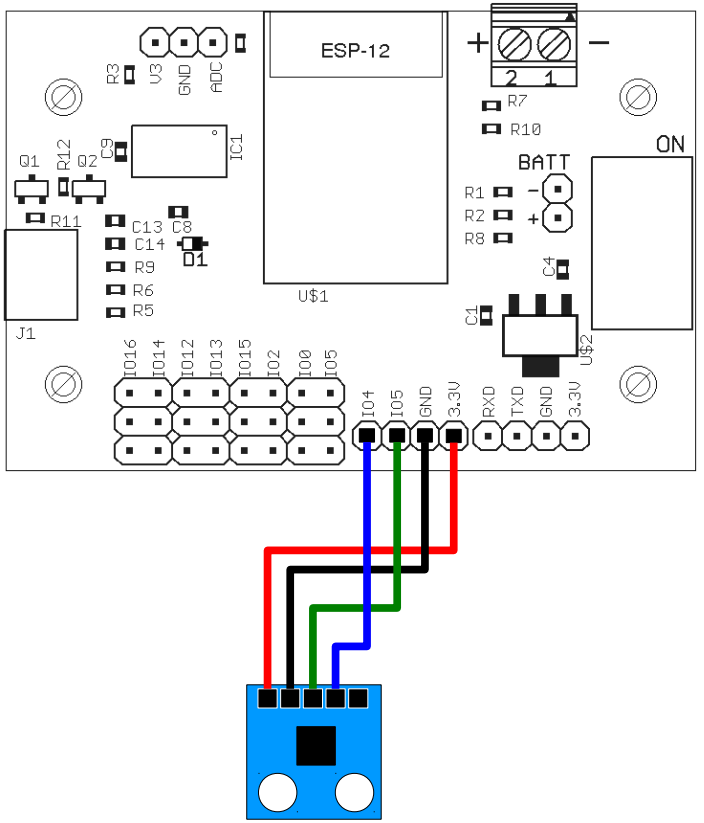
Control Register

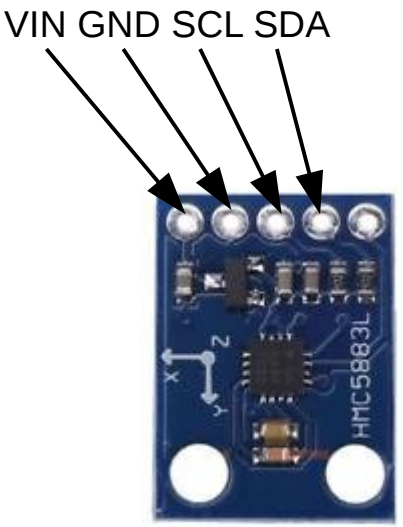
- Configuration register 2 has only three bits used.
- Bit 0 is used to enable an interrupt pin. It is not available.
- Bit 6 is a pointer rollover bit. If set, code can keep reading the data registers without having to send an address each time.
- Bit 7 is a soft reset. When set to 1, the device will reset to default settings.

Reg	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x0a	SOFT RESET	ROL PNT						INT ENABLE

Connecting the Accelerometer

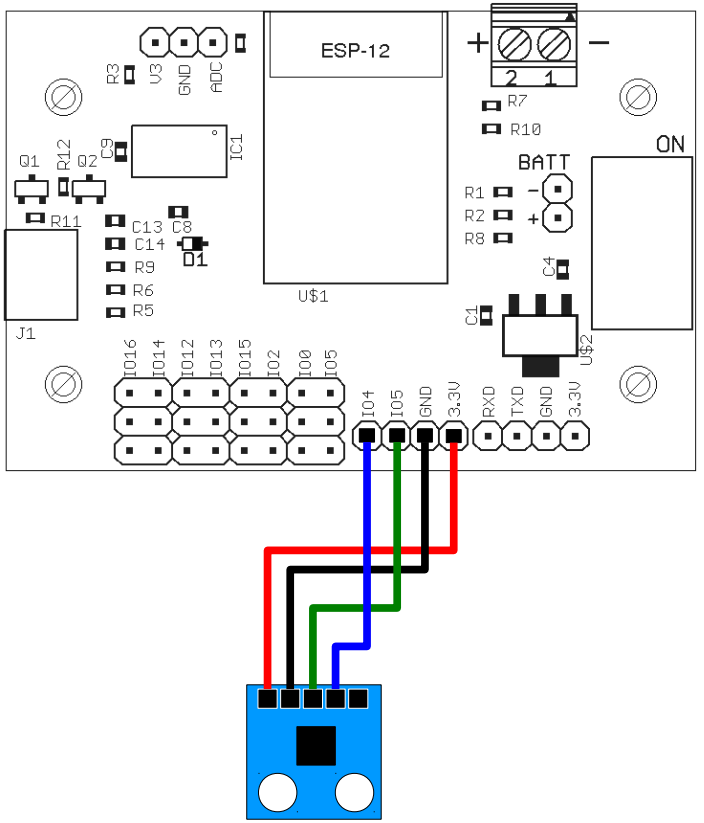
- The processor uses pins 4 and 5 for the I2C bus. Pin 4 is SCL and pin 5 is SDA.
- The pins on the accelerometer are identified on the back side. The picture shown is from the top side of the accelerometer.
- The sensor can be inserted into the solderless bread board.
- Use the male to female jumpers.

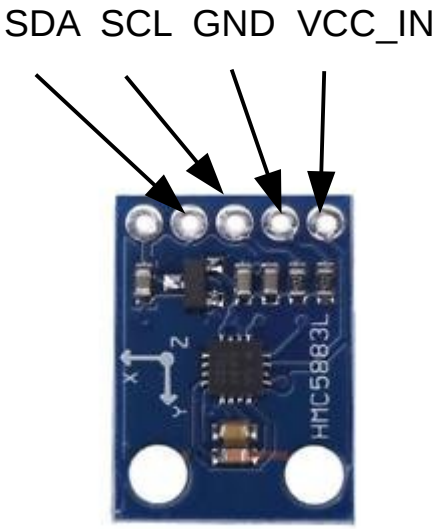




Connecting the Accelerometer

- Connect 3.3V to VCC_IN.
- Skip a pin.
- Connect GND to GND.
- Connect D4 to the next pin SCL.
- Connect D5 to the next pin SDA.





I2C Bus

- There is a library called **Wire** that handles all the I2C data transfer details. Using this library simplifies using the I2C bus.
- In a new program, add the library **Wire**.
- To add a library, click on the **Sketch** menu, select **Include Library**, and scan down to locate **Wire**.
- The include file will be added to the top of the program.

```
#include <Wire.h>

void setup() {
  Wire.begin(4,5);
  Serial.begin(115200);
```

I2C Bus

- In **setup()**, the I2C interface is configued. Digital Pin 4 is set to SCL and pin 5 is set to SDA.
- The serial interface is configured so data from the accelerometer to be connected can be viewed.
- Noticed the data rate has been increased to 115200. This allows the data to be displayed faster.

```
#include <Wire.h>

void setup() {
  Wire.begin(4,5);
  Serial.begin(115200);
```

I2C Bus

- Next, the magnetometer needs to be configured.
- The magnetometer address is 0x0d. The first I2C command is **Wire.beginTransmission(0x0d);** This command initiates the I2C transfer and sends out the device address.
- The **Wire.write(0x09);** command sends the first byte on the I2C bus. This byte is used to tell the magnetometer that the next byte is to be written into register 0x09. Register 0x09 is **CTRL_REG1**.
- The next byte **0x05** is the data to be written in the register.
- **Wire.endTransmission()** completes the data transfer.

```
#include <Wire.h>

void setup() {
  Wire.begin(4,5);
  Serial.begin(115200);
  Wire.beginTransmission(0x0d);
  Wire.write(0x09);
  Wire.write(0x05);
  Wire.endTransmission();
}
```

I2C Bus

- Moving on to the loop() section. The X,Y,Z accelerometer data will be read from the slave. This is a two step process.
- int reg[6] declares an array of variables. They all have the same name but are differentiated by the number between the brackets. This array will be used to hold the accelerometer data.

```
void loop() {
  int reg[6];
  int i;
  Wire.beginTransmission(0x0d);
  Wire.write(0x00);
  Wire.endTransmission(false);
```

I2C Bus

- First step is to send the address where the data is located to be read. The I2C data transfer is started as usual, the register address is written with **Wire.write(0x00)**. The I2C transfer is stopped with an argument **false**. This is needed to indicate a read is to occur.
- The data registers start at address 0x00. Each axis uses two registers.

```
void loop() {
  int reg[6];
  int i;
  Wire.beginTransmission(0x0d);
  Wire.write(0x00);
  Wire.endTransmission(false);
```

I2C Bus

- The second step is to request reading a specific number of bytes.
- **Wire.requestFrom(0x0d,6)** starts the reading of the data registers. The slave address is specified and the number of bytes to read is specified.

```
void loop() {
  int reg[6];
  int i;
  Wire.beginTransmission(0x0d);
  Wire.write(0x00);
  Wire.endTransmission(false);
  Wire.requestFrom(0x0d,6);
```

I2C Bus

- After the request is made, a while loop waits for 6 bytes to be transferred from the accelerometer. Once 6 bytes are transferred, the code can read in the bytes.

```
void loop() {
  int reg[6];
  int i;
  Wire.beginTransmission(0x0d);
  Wire.write(0x00);
  Wire.endTransmission(false);
  Wire.requestFrom(0x0d,6);
  while(Wire.available() < 6) {
    delay(1);
  }
  for(i=0;i<6;i++) reg[i] = Wire.read();
```

I2C Bus

- The bytes read are combined into complete 16-bit numbers.
- reg[0] holds the least significant byte of the X data. reg[1] holds the most significant byte of the X data. To combine them, reg[1] is shifted left 8 bits to a 16 bit variable and then OR'd with the least significant byte. This combines the two registers into a single 16-bit number.
- The same is done with the Y and Z data.

```
void loop() {
  int reg[6];
  int i;
  Wire.beginTransmission(0x0d);
  Wire.write(0x00);
  Wire.endTransmission(false);
  Wire.requestFrom(0x0d,6);
  while(Wire.available() < 6) {
    delay(1);
  }
  for(i=0;i<6;i++) reg[i] = Wire.read();
  short x = (reg[1] << 8) | reg[0];
  short y = (reg[3] << 8) | reg[2];
  short z = (reg[5] << 8) | reg[4];
```

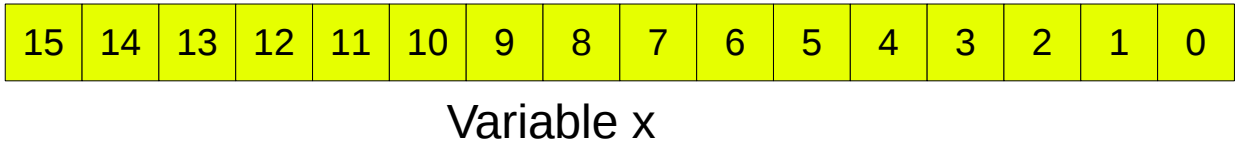
I2C Bus

- Lastly, the data is displayed in the serial monitor.
- Upload the program and run it. Change the orientation of the sensor board and observe how the values change.

```
void loop() {
  int reg[6];
  int i;
  Wire.beginTransmission(0x0d);
  Wire.write(0x00);
  Wire.endTransmission(false);
  Wire.requestFrom(0x0d,6);
  while(Wire.available() < 6) {
    delay(1);
  }
  for(i=0;i<6;i++) reg[i] = Wire.read();
  short x = (reg[1] << 8) | reg[0];
  short y = (reg[3] << 8) | reg[2];
  short z = (reg[5] << 8) | reg[4];
  Serial.print(x); Serial.print(" ");
  Serial.print(y); Serial.print(" ");
  Serial.println(z);
  delay(100);
}
```

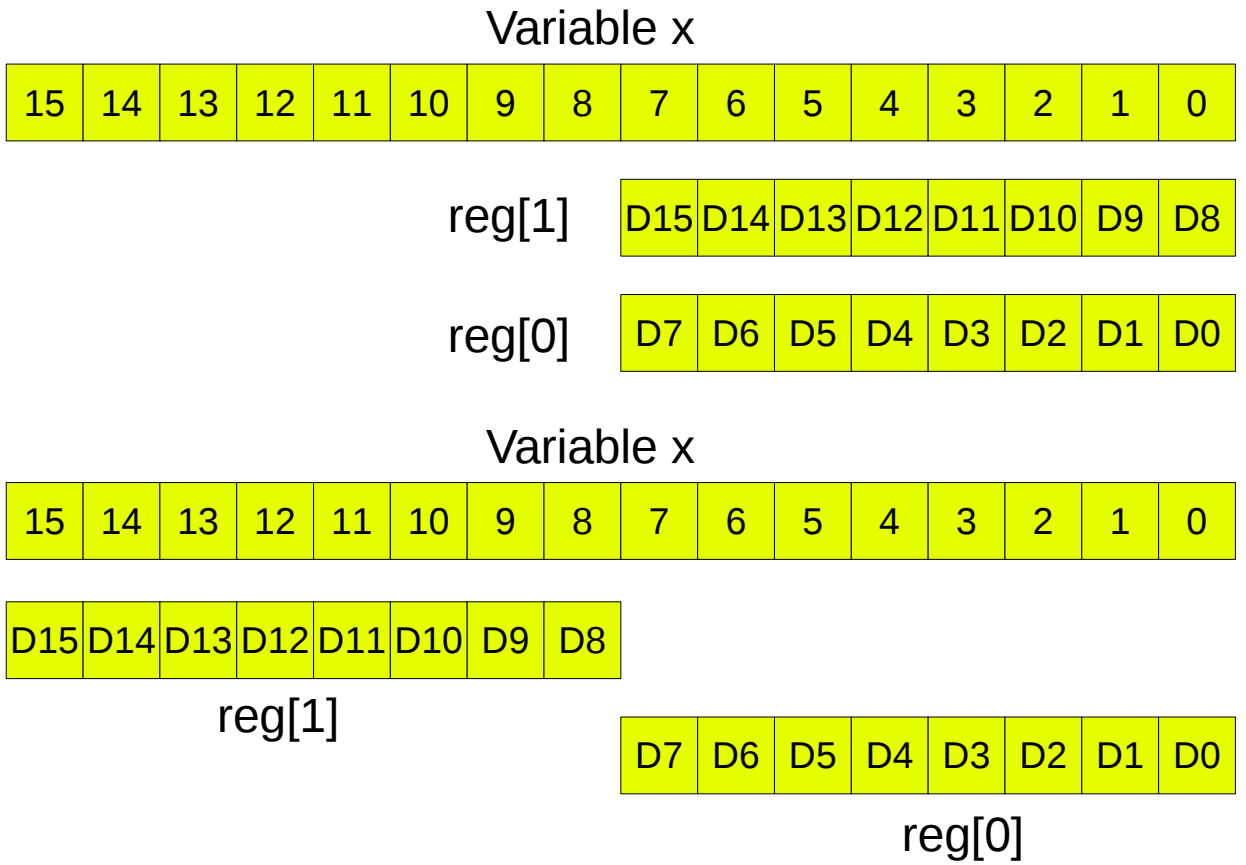
Logical Operations

- There are some logical operations used in the code. Let's start with the shift. **reg[1]** contains the most significant bits and **reg[0]** contains the least significant bits from the accelerometer for the X axis.
- Variable **x** is declared a short which means **x** is a 16 bit variable.



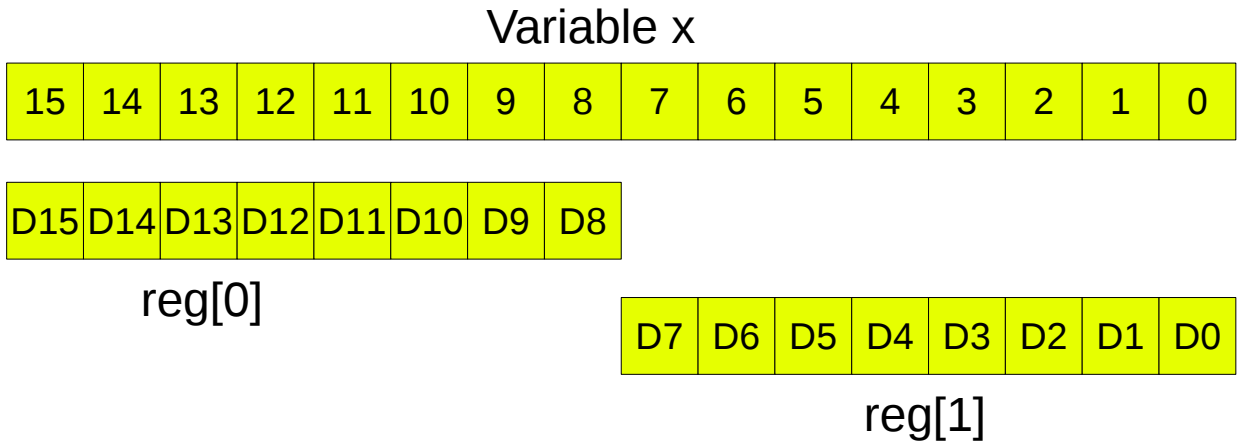
Logical Operations

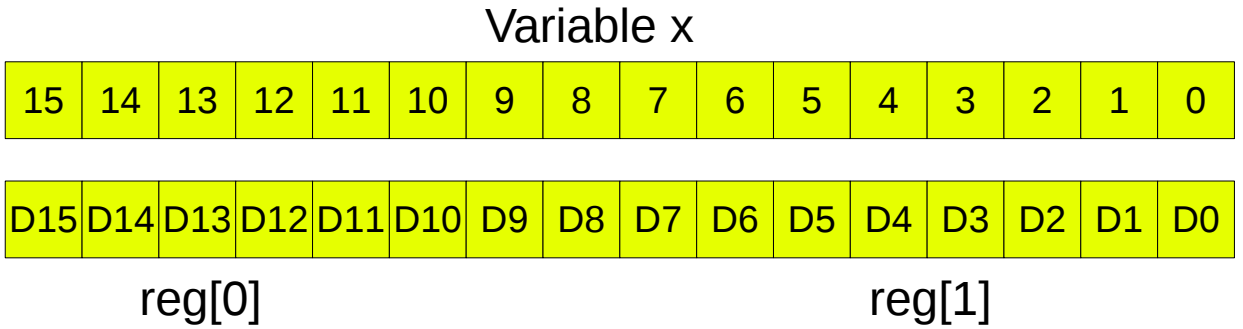
- Top right shows the variables. **reg[0]** and **reg[1]** are 8 bits and x is 16 bits.
- The equation **reg[1] <<8** moves the bits of **reg[1]** to the left 8 bits as shown in the lower right. **reg[1]** becomes the top half of variable x



Logical Operations

- Next, the variable **reg[0]** is OR'd with variable **x**. The vertical line in the equation is the OR symbol.
- This combines **reg[1]** into variable **x**.





OR symbol

```
short x = (reg[0] << 8) | reg[1];
```

Shift Symbol

Logical Operators

- What is OR? OR is a logical operation. At it's simplest with two inputs, if any of the inputs are a logic level 1, the result is 1. If both inputs are 0, the output is 0.
- When doing a logic OR between two variables, each bit from one variable is OR'd with the corresponding bit of the other variable.

Variable A	0	1	0	0	1	1	0	0
Variable B	1	0	0	0	0	1	0	1
Result	1	1	0	0	1	1	0	1

Calibrating the Magnetometer

- The magnetometer can have a bias meaning that as the sensor is rotated, the values do not go around the center point of 0. One axis can have all negative values in all orientations. External magnetic fields can cause this or it was caused during manufacturing.
- A simple way to determine the bias and remove it is to assume the values generated as the sensor is rotated will make a circle if plotted. The offset of the center of the circle can be determined and subtracted from the sensor data to move the circle of data so the center is at 0.

Calibrating the Magnetometer

- This code will calibrate.
- At the top of the code, global variables are declared. minX, maxX and the rest will hold the largest and smallest values detected in the X axis. The other axis have their variables.
- offX, offY, offZ will hold the offsets calculated.
- In setup, the variables are initialized. maxX is set to -9999 so it is guaranteed a larger value will occur. Same with minX. It is set to a value that is larger than expected.

```
#include <Wire.h>
int maxX, maxY, maxZ;
int minX, minY, minZ;
int offX, offY, offZ;
int fx, fy, fz;

void setup() {
  Serial.begin(115200);
  delay(500);
  Wire.begin(4, 5);
  Wire.setClock(100000);
  delay(100);
  Wire.beginTransmission(0x0d);
  Wire.write(0x09);
  Wire.write(0x05);
  Wire.endTransmission();
  Serial.println("\n\nConfigured");
  maxX = -9999; maxY = -9999; maxZ = -9999;
  minX = 9999; minY = 9999; minZ = 9999;
  offX = 0; offY = 0; offZ = 0;
}
```

Calibrating the Magnetometer

- In the loop, the sensor will be sampled and the calibration will be performed.
- The data is compared to the min and max variables and the min and max variable are updated when the data exceeds the min and max values.
- The offsets are calculated by determining the average between the min and max.
- The final data fx,fy, and fz are calculated by subtracting the offset from the sensor data.

```
void loop() {
  // put your main code here, to run repeatedly:
  int reg[6];
  //  Serial.println("Getting data");
  Wire.beginTransmission(0x0d);
  Wire.write(0x0);
  Wire.endTransmission(false);
  delay(10);
  Wire.requestFrom(0x0d, 6);
  while (Wire.available() < 6) delay(1);
  for (int i = 0; i < 6; i++) reg[i] = Wire.read();
  short x = (reg[1] << 8) | reg[0];
  short y = (reg[3] << 8) | reg[2];
  short z = (reg[4] << 8) | reg[4];
  if (x > maxX) maxX = (int)x;
  if (x < minX) minX = (int)x;
  if (y > maxY) maxY = (int)y;
  if (y < minY) minY = (int)y;
  if (z > maxZ) maxZ = (int)z;
  if (z < minZ) minZ = (int)z;
  offX = (maxX + minX) / 2;
  offY = (maxY + minY) / 2;
  offZ = (maxZ + minZ) / 2;
  fx = (int)x - offX;
  fy = (int)y - offY;
  fz = (int)z - offZ;
```

Calibrating the Magnetometer

- Finally, the results are displayed.
- Upload the code and try it out.
- Once it starts running, rotate the sensor in all directions. The values should start centering around zero.
- The more you rotate slowly and more often, the better the offset calculation will get.
- You could plot the data in 3 dimensions to see if a sphere is created.
- Calibration should not be continuous. It should be done for a period of time then the offsets used.

```
  fx = (int)x - offX;
  fy = (int)y - offY;
  fz = (int)z - offZ;

  Serial.print(fx);
  Serial.print(" ");
  Serial.print(fy);
  Serial.print(" ");
  Serial.println(fz);
  delay(100);
}
```

End