

**HIGHER COLLEGES OF TECHNOLOGY**

**DUBAI MEN’S COLLEGE**

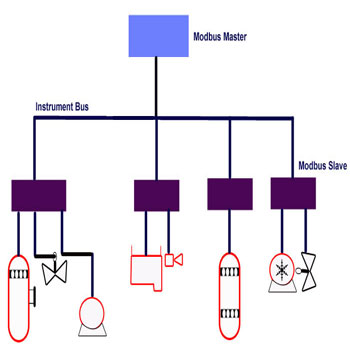
**COURSE CODE:** Industrial Control Systems- EMT 4013

Assessed Task

**Project Report**

**Modbus® RTU**

**This is a Graded Task**

[](http://www.google.ae/url?sa=i&rct=j&q=&esrc=s&frm=1&source=images&cd=&cad=rja&uact=8&ved=0CAcQjRw&url=http://www.abtechteam.com/fieldbus_Modbus_Serial_Protocol.html&ei=HuRiVcu7FIXuUIvggdgI&bvm=bv.93990622,d.bGQ&psig=AFQjCNFsS6NGfZFzloxyWZZOyXg0ydvTBg&ust=1432630648392695)

**Submitted by:** Saleh AlZaabi

**ID:** H00214600

**Submitted to:** Dr. Basim Al-Lami

**Submitted date:** 26-05-2015

Contents

[Purpose:- 3](#_Toc372555471)

[Introduction:- 3](#_Toc372555472)-4

[Features:](#_Toc372555473) 4-6

[requred componants:](#_Toc372555474) 7

[modbus function:](#_Toc372555475) 7-11

[Intelligent sensor HMC5883L:](#_Toc372555476) 12-14

[apparatus:](#_Toc372555477) 15-16

[program:](#_Toc372555477) 17-20

[how project works:](#_Toc372555478) 21

[conclusion:](#_Toc372555478) 21

# 

# purpose:

Build a Modbus® RTU to control 5 slaves like: stepper motor, DC motor, Servo, intelligent sensor, and distance sensor.

# Introduction:

Modbus is a serial communications protocol originally published by Schneider Electric in 1979 for use with their programmable logic controllers (PLCs). Modbus uses a master/slave (or client-server) architecture. Simple and robust, it has since become a de facto standard communication protocol, and it is now a commonly available means of connecting industrial electronic devices. Modbus communication protocol presents the follow features:

* It has been developed with industrial applications in mind.
* Openly published and royalty-free.
* Easy development and maintenance.
* The Modbus communication interface is built around messages. The format of these Modbus messages is independent of the type of physical interface used. On plain old RS232 are the same messages used as on Modbus/TCP over Ethernet. This gives the Modbus interface definition a very long lifetime. The same protocol can be used regardless of the connection type.

Modbus allows communication between many devices connected to the same network, for example a system that measures temperature and humidity and communicates the results to a computer.

Many of the data types are named from its use in driving relays:

* A single-bit physical output is called a coil.
* A single-bit physical input is called a discrete input or a contact.

This list includes some of the most common uses of the standard:

* Multiple master-slave applications
* Sensors and instruments
* Industrial networking
* Building and infrastructure
* Transportation and energy applications

### Modbus RTU Implementation

This implementation is designed to provide a popular data exchange format connecting these instruments to both Honeywell and foreign master devices. The Modbus RTU allows the instrument to be a citizen on a data link shared with other devices that subscribe to the Modbus RTU RS-485 specification.

These instruments DO NOT emulate any MODICON type device. The Modbus RTU specification is respected in the physical and data link layers. The message structure of the Modbus RTU function codes is employed and standard IEEE 32-bit floating point and integer formats are used. Data register mapping is unique to these instruments. The definition in Table 2-1 is the register mapping for many Honeywell instruments and the corresponding parameter value within those instruments.

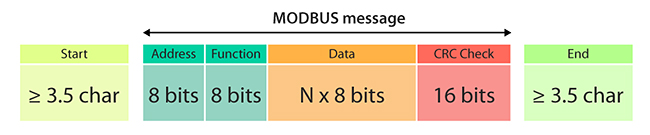
Features:

* Modbus for Arduino / RaspberryPi / Galileo uses RTU mode of transmission.
* This mode is used in serial communication and makes use of a compact, binary representation of the data for protocol communication.
* The RTU format Receives the commands/data with a cyclic redundancy check checksum as an error check mechanism to ensure the reliability of data.
* A Modbus RTU message must be transmitted continuously without inter-character hesitations.
* Modbus messages are framed by idle periods.
* The Modbus RTU transmission mode usually includes a parity bit to detect transmission errors. You can choose to transmit data with or without parity checking, but always make sure that all equipment connected to Modbus has the same configuration mode, otherwise communication will not be possible.

### Modbus message structure:

Modbus RTU frame:

An RTU frame includes the following information:



### Address:

* Valid slave device addresses are in the range of 0 ... 247 (decimal).
* Value 0 is reserved for broadcast messages (no response).
* When the slave sends its Answer, it places its own address in this address field of the response frame to let the master know which slave is responding.
* Within a Modbus device, the holding registers, inputs and outputs are assigned a number between 1 and 10000.
* One would expect, that the same addresses are used in the Modbus messages to read or set values. Unfortunately this is not the case. In the Modbus messages addresses are used with a value between 0 and 9999.
* If you want to read the value of output (coil) 18 for example, you have to specify the value 17 in the Modbus query message. More confusing is even, that for input and holding registers an offset must be subtracted from the device address to get the proper address to put in the Modbus message structure.

|  |  |  |
| --- | --- | --- |
| **Device and Modbus address ranges** | | |
| **Device address** | **Modbus address** | **Description** |
| **1**...**10000**\* | address - 1 | Coils (outputs) |
| **10001**...**20000**\* | address - 10001 | Inputs |
| **40001**...**50000**\* | address - 40001 | Holding registers |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Access** | **Visual** |
| Discrete Input | single bit | read-only | Discrete Input |
| Discrete Output (Coils) | single bit | read-write | Discrete output/Coil |
| Input Registers | 16-bit word | read-only | Input Register |
| Holding Registers (Registers) | 16-bit word | read-write | (Holding) Register |

### Function Code:

* Valid codes are in the range of 1... 255 (decimal).
* The function code field tells the slave what kind of action to perform. For a normal response, the slave simply echoes the original function code.
* For an exception response, the slave returns a code that is equivalent to the original function code with its most significant bit set to a logic 1.
* All Modbus devices recognize the same set of function codes.

### Data:

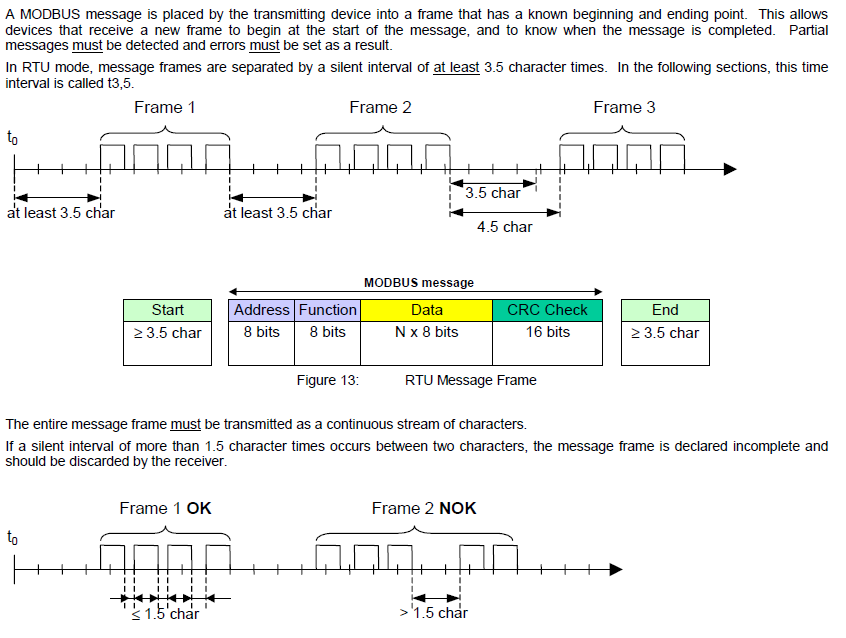
* The data field contains additional information related to the I/O bits or Memory. This can include items like register addresses, number of items to be handled, etc.
* If no error occurs, the data field contains the data requested.
* If an error occurs, the field contains an exception code that the master application can use to determine the next action to be taken.

### CRC:

* The checksum is calculated by the master and sent to the slave.
* The checksum is re-calculated by the slave and compared to the value sent by the master.
* If a difference is detected, the slave will not construct a response to the master.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Characters** | ASCII **0**...**9** and **A**….**F** | | Binary **0**...**255** | |
| **Error check** | **LRC** Longitudinal Redundancy Check | | **CRC** Cyclic Redundancy Check | |
| **Frame start** | character '**:**' | | 3.5 chars silence | |
| **Frame end** | characters **CR**/**LF** | | 3.5 chars silence | |
| **Gaps in message** | 1 sec | | 1.5 times char length | |
|  |  |  |  |  |
| **Start bit** | **1** | | **1** | |
| **Data bits** | **7** | | **8** | |
| **Parity** | even/odd | none | even/odd | none |
| **Stop bits** | **1** | **2** | **1** | **2** |

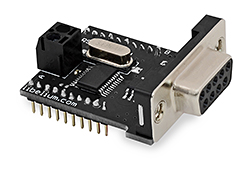
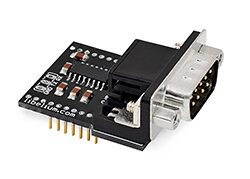
### Message Timing:



# Required Components:

### RS-485 and RS-232 Interfaces:

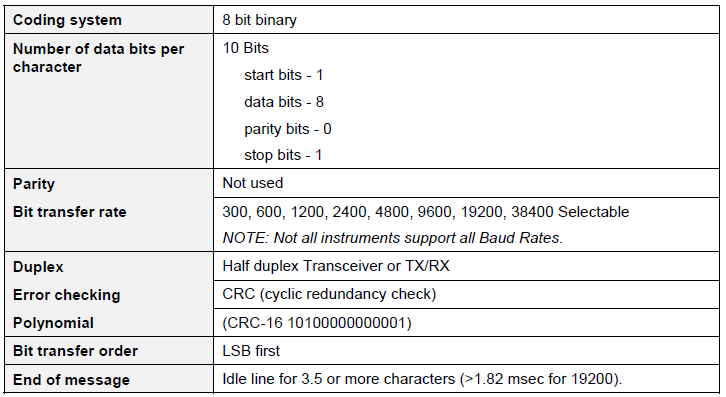
* The Modbus protocol can be implemented over RS-485 and RS-232 physical layers.
* Cooking-Hacks provides the necessary hardware and software for working with both protocols.
* The name and use of the functions are the same for RS-232 and RS-485, and the only changes are the library to include and the instantiation of the object.
* The differences between the two standards are explained in the corresponding tutorials.



**modbus function**:

The function code field of the message frame will contain two characters (in ASCII mode), or 8 binary bits (in RTU Mode) that tell the slave what kind of action to take. Valid function codes are from 1-255, but not all codes will apply to a module and some codes are reserved for future use.

### Modbus RTU Message Format:



### Modbus RTU Link Layer:

The link layer includes the following properties/behaviors:

* Slave address recognition,
* Start / End of Frame detection,
* CRC-16 generation / checking,
* Transmit / receive message time-out,
* Buffer overflow detection,
* Framing error detection,
* Idle line detection.

Errors detected by the physical layer in messages received by the slave are ignored and the physical layer automatically restarts by initiating a new receive on the next idle line detection.

The format for each byte in RTU mode is:

**Coding System:**

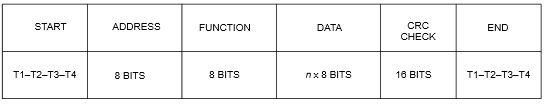
* 8–bit binary, hexadecimal 0–9, A–F
* Two hexadecimal characters contained in each
* 8–bit field of the message

**Bits per Byte:**

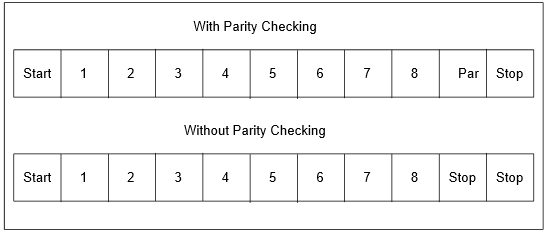
* 1 start bit
* 8 data bits, least significant bit sent first
* 1 bit for even/odd parity; no bit for no parity
* 1 stop bit if parity is used; 2 bits if no parity

**Error Check Field:**

* Cyclical Redundancy Check (CRC)



**RTU character framing, the bit sequence is:**

****

### General Modbus RTU message format:

**Query message format:**

**The Query:** The function code in the query tells the addressed slave device what kind of action to perform. The data bytes contain any additional information that the slave will need to perform the function

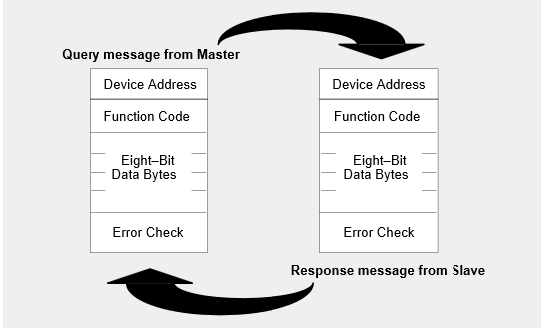
[Slave Address, Function Code, Function code dependent data, CRC 16]

**Response message format:** If the slave makes a normal response, the function code in the response is an echo of the function code in the query. The data bytes contain the data collected by the slave, such as register values or status. If an error occurs, the function code is modified to indicate that the response is an error response, and the data bytes contain a code that describes the error. The error check field allows the master to confirm that the message contents are valid.

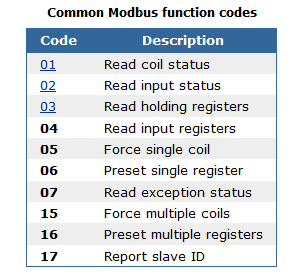
[Slave Address, Function Code\*, Function code dependent data, CRC 16]

\* If an error is detected in a valid message the response function code is modified by adding 80 (hex) and the function code dependent data is replaced by an exception response code as described in *Section* 4 - Modbus RTU Exception Codes.

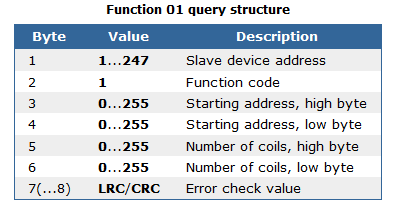
Between messages, the RS-485 link is in a high impedance state. During this time receiving devices aremore susceptible to noise generated false start of messages. Although noise-generated messages arerejected due to address, framing, and CRC checking, they can cause the loss of a good message when theyare included in the message stream. In the slave, the transmitting device enables its transmitter line driverand forces an idle line state onto the link for three character time slots prior to transmitting. This actionforces termination of any noise generated messages and improves message frame synchronization.

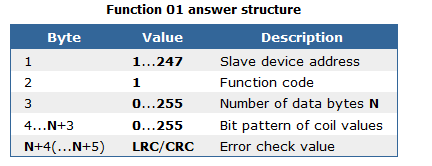


## Modbus function codes:

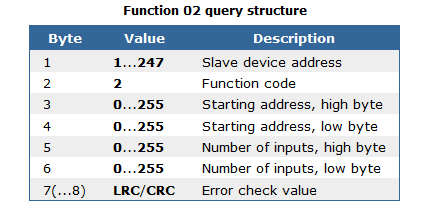


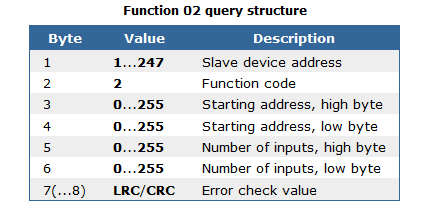
### Function 01: Read coil status



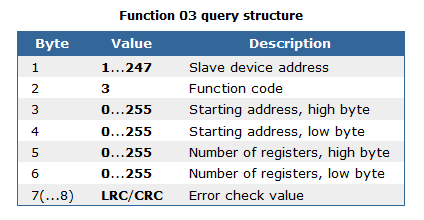


### Function 02: Read input status





### Function 03: Read holding registers



### MODBUS ERROR CHECKING:

MODBUS networks employ two methods of error checking: parity checking:

* Parity checking of the data character frame (even, odd, or no parity)
* Frame checking within the message frame (Cyclical Redundancy Check in RTU Mode)

### Modbus RTU Data Layer:

The data layer includes:

* Diagnostic loopback,
* Function code recognition / rejection,
* Busy / repoll,
* Data error code generation

Intelligent Sensor (hmc5883l) my slave:

The HMC5883L (figure#1) is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as low-cost compassing and magnetometer. The HMC5883L includes our state-of-the-art, high-resolution HMC118X series magneto-resistive sensors plus an ASIC containing amplification, automatic degaussing strap drivers, offset cancellation, and a 12-bit ADC that enables 1° to 2° compass heading accuracy. The I2C serial bus allows for easy interface. The HMC5883L is a 3.0x3.0x0.9mm surface mount 16-pin leadless chip carrier (LCC). Applications for the HMC5883L include Mobile Phones, Netbooks, Consumer Electronics, Auto Navigation Systems, and Personal Navigation Devices.

The HMC5883L utilizes Anisotropic Magneto resistive (AMR) technology that provides advantages over other magnetic sensor technologies. These anisotropic, directional sensors feature precision in-axis sensitivity and linearity. These sensors’ solid-state construction with very low cross-axis sensitivity is designed to measure both the direction and the magnitude of Earth’s magnetic fields, from mille-gauss to 8 gauss. Honeywell’s Magnetic Sensors are among the most sensitive and reliable low-field sensors in the industry.

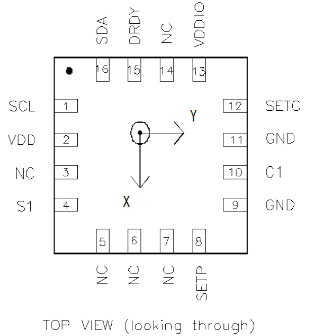
 

Figure #1

### 

### FEATURES:

* 3-Axis Magneto resistive Sensors and ASIC in a 3.0 x 3.0 x0.9mm LCC Surface Mount package.
* 12-bit ADC Coupled with low noise AMR sensor Achieves 2 mille-gauss field resolution in ±8 Gauss Fields.
* Built-in self-test.
* Low voltage operates (2.16v to 3.6v)
* Low power consumption ( 100 μA)
* Built- in strap drive circuits.
* I2C Digital Interface.
* Lead Free Package Construction.
* Wide Magnetic Field Range (+/-8 Oe).
* Software and Algorithm Support Available.
* Fast 160 Hz Maximum Output Rate.

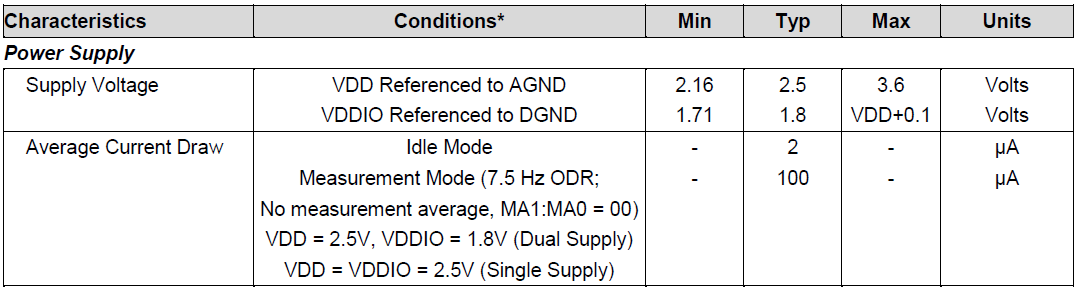
### BENEFITS:

* Small Size for Highly Integrated Products. Just Add a Micro-Controller Interface, Plus Two External SMT Capacitors Designed for High Volume, Cost Sensitive OEM Designs Easy to Assemble & Compatible with High Speed SMT Assembly.
* Enables 1° to 2° Degree Compass Heading Accuracy.
* Enables Low-Cost Functionality Test after Assembly in Production.
* Compatible for Battery Powered Applications.
* Set/Reset and Offset Strap Drivers for Degaussing, Self-Test, and Offset Compensation.
* Popular Two-Wire Serial Data Interface for Consumer Electronics.
* RoHS Compliance
* Sensors Can Be Used in Strong Magnetic Field Environments with a 1° to 2° Degree Compass Heading Accuracy.
* Compassing Heading, Hard Iron, Soft Iron, and Auto Calibration Libraries Available.
* Enables Pedestrian Navigation and LBS Applications.

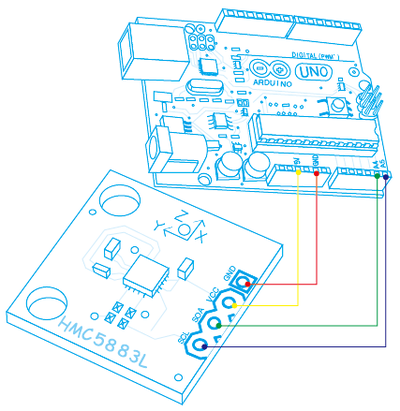
### Application:

* GPS enabled wireless phones.
* Portable GPS Receivers (PNDs)
* Wireless Mice/Pointers.
* Sport Watches/Drive Computer Gaming.

### SPECIFICATIONS:

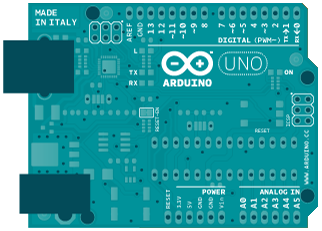


### The connection between the hmc5883l and Arduino:

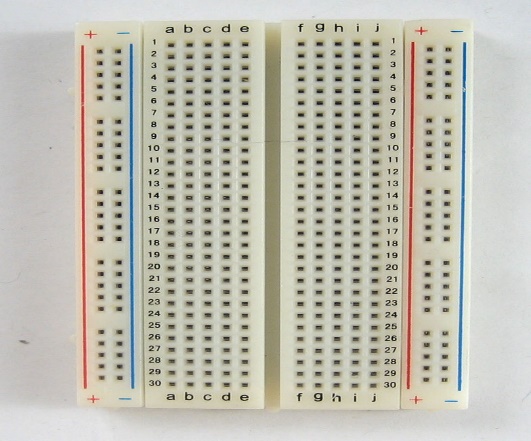


# Apparatus:

* Arduino:

[](http://www.google.ae/url?sa=i&rct=j&q=&esrc=s&frm=1&source=images&cd=&cad=rja&uact=8&ved=0CAcQjRw&url=http://www.sunbedded.nl/en/arduino-uno.html&ei=DKJdVZKsK4XtUoX_geAI&bvm=bv.93756505,d.bGQ&psig=AFQjCNFEVnMcrEC5cttDyQ--8mpgjI3GEA&ust=1432286082104606)

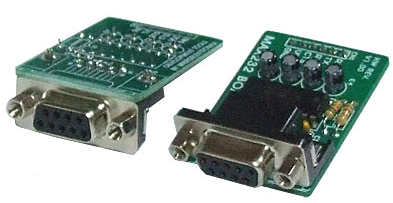
* Breadboard:



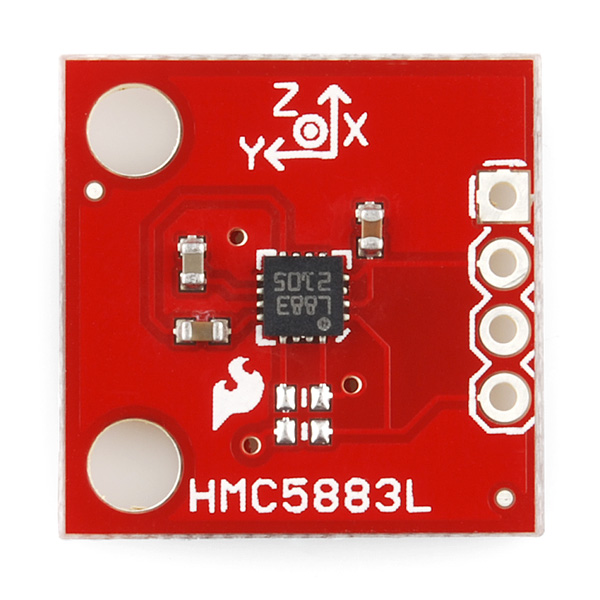
* RS485 – RS232:



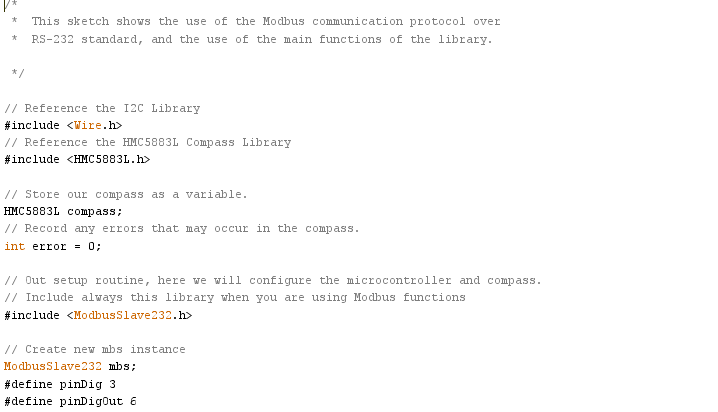
* MAX232:

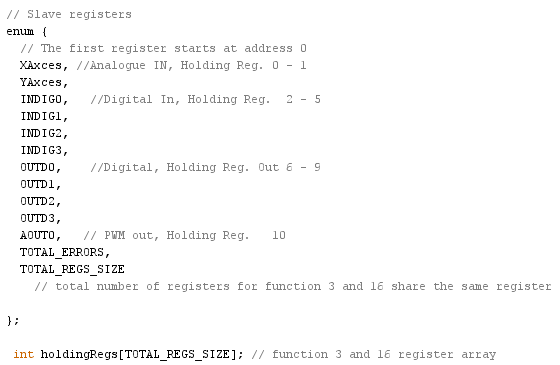


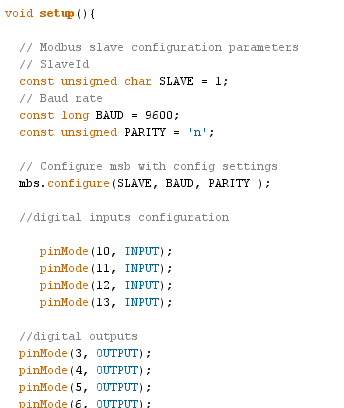
* Intelligent Sensor (hmc5883l):

[](https://www.google.ae/url?sa=i&rct=j&q=&esrc=s&frm=1&source=images&cd=&cad=rja&uact=8&ved=0CAcQjRw&url=https://www.sparkfun.com/products/10530&ei=siJiVfezDeK6ygPqioGgBA&bvm=bv.93990622,d.bGQ&psig=AFQjCNFxoLMHzZ-AIzxIF_ahYLks_C2mnw&ust=1432581157131929)

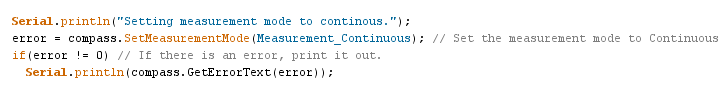
# Program:



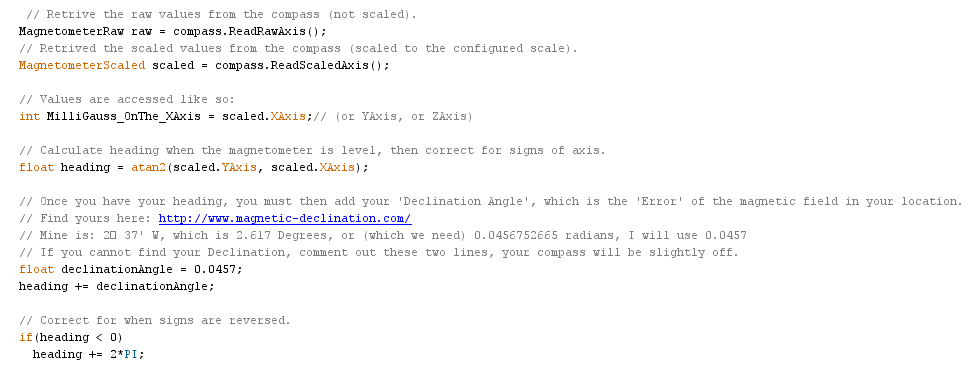


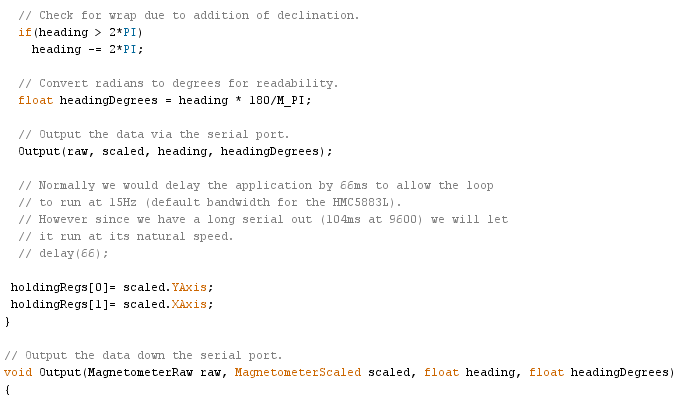


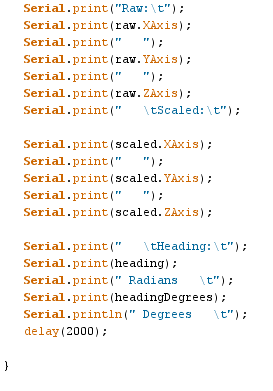




****

****





# how the project works:

The hmc5883l takes the reading of X-axis and Y-axis and save it in the register zero and one. The master read the X-axis and Y axis from the holding registers. The master send the message to the slave and read form it or let it do something like check the angle or position of servo, Dc motor and stepper motor.

The messages pass all slaves and the selected slave will send or do the massage and send a feedback.

My slave hmc5883l will take the reading and save it and the master takes the reading and display it in the screen.

Conclusion:

* In conclusion, this project was helpful and I have learned how the Modbus is a serial communications protocol is done and how it works, and I am able to do hall system.
* This project help me to understand how to control hall system.
* This project helps me to know the job of master and Slave. Also I have learned how I can find if the master or slave is not working.
* This project helps me to understand the program and the communications between master and slave.
* I have learned that the Modbus is Easy development and maintenance, and Modbus allows communication between many devices connected to the same network.