UTILIZATION OF HMI AND MODBUS RTU FOR APPLICATION IN ROBOTICS AND CONTROL SYSTEMS USING ARDUINO AS A DEVELOPMENT PLATFORM

By Voltaire B. Dupo, ECE

Abstract

This manual takes you thru the process of creating an automate system. It builds on current knowledge in electronics and electromagnetic machines and add the concepts of process of conceptualization, documentation and test and evaluation. The material uses examples as loose basis for decision making and process creation. It covers Human Machine Interface using two software based HMI.

Voltaire Dupo
Vdupo79@gmail.com

Contents

Writers Notes

This booklet was initially a set of written exercises for the students of DBCS written on the free time of the author not on the time extended and paid for by DBCS. The work presented is from the writer / engineer who had to come up with laboratory materials from a system that was without one for the class.

Attribution-NoDerivatives 4.0 International

This is not a Free Culture License.

Consideration on Time and Effort

There are of course similar modules online and you probably need around 4 modules or so to complete the same type of course. If each module where priced at around 20 USD per module without taxes. That would set you back around 4000 pesos and spend around 5-8 days figuring out how to connect each module to each other and write the materials.

Future Work

Future work will be done to document using the OMRON CP1E and CP1H to do the some same sort of integration to complete a series that starts with Arduino MODBUS based Automation and Systems Integration and continues on to OMRON Based Automation and Systems Integration. Of course another separate Attribution-NoDerivates 4.0 License will be extended for that work but work will focus more documenting the process of how to use CX-Programmer to do the job.

Preparation for the Course

1. Download Arduino IDE https://www.arduino.cc/en/software
2. Download the MODBUS Libraries
    a. Download the MODBUS Master – Slave Library https://drive.google.com/file/d/1H2xZYBFeB_kDleod_ihiV7Wb-HykK0JH/view?usp=sharing
    b. Download the MODBUS Master Library https://drive.google.com/file/d/1kJ15zbcLAbcHGY9FJT9Uc9IM6M25zngR/view?usp=sharing
3. Install the MODBUS Libraries to your Arduino IDE by adding the zip file to the library

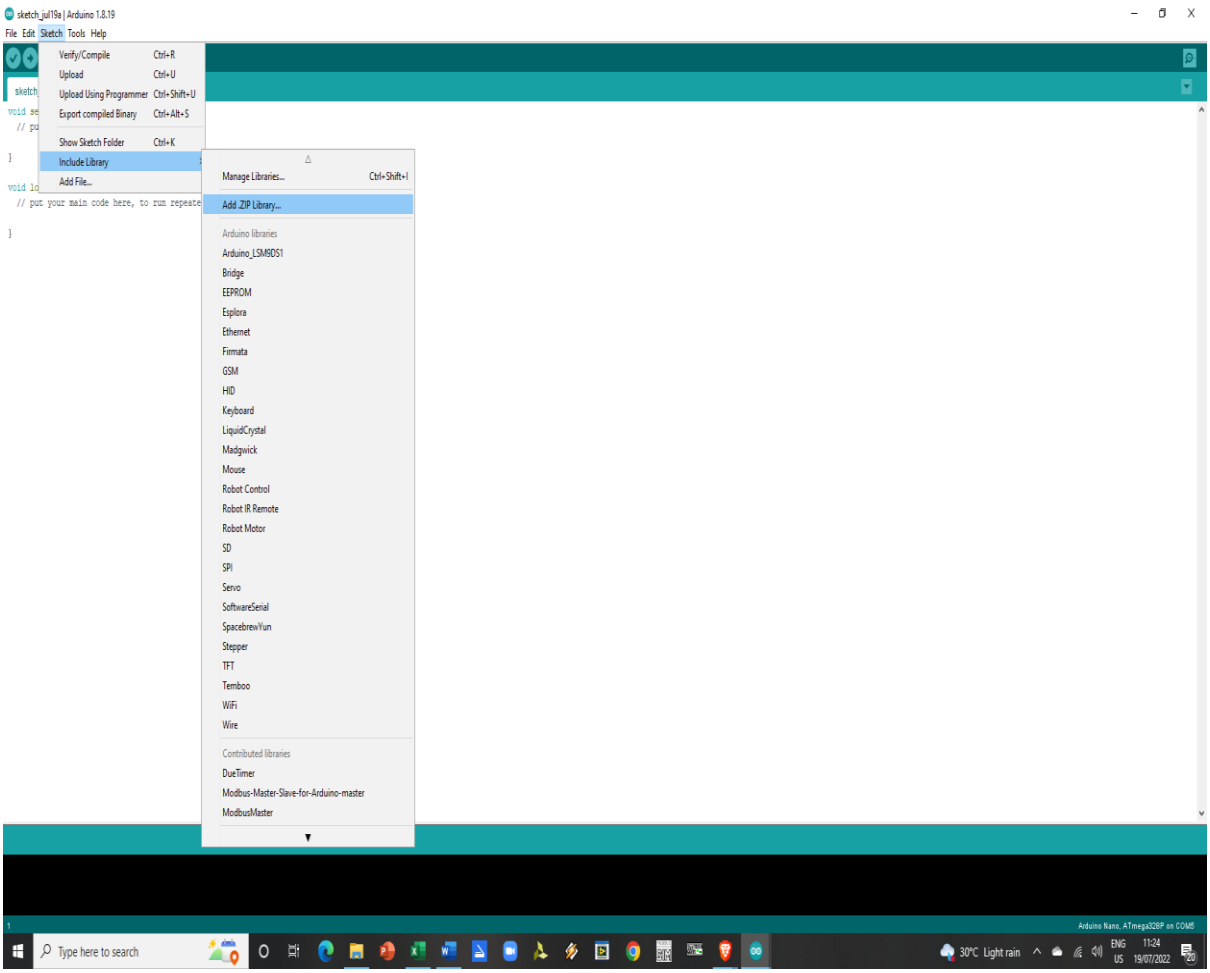


Figure A. How to add the zip file to install MODBUS Library

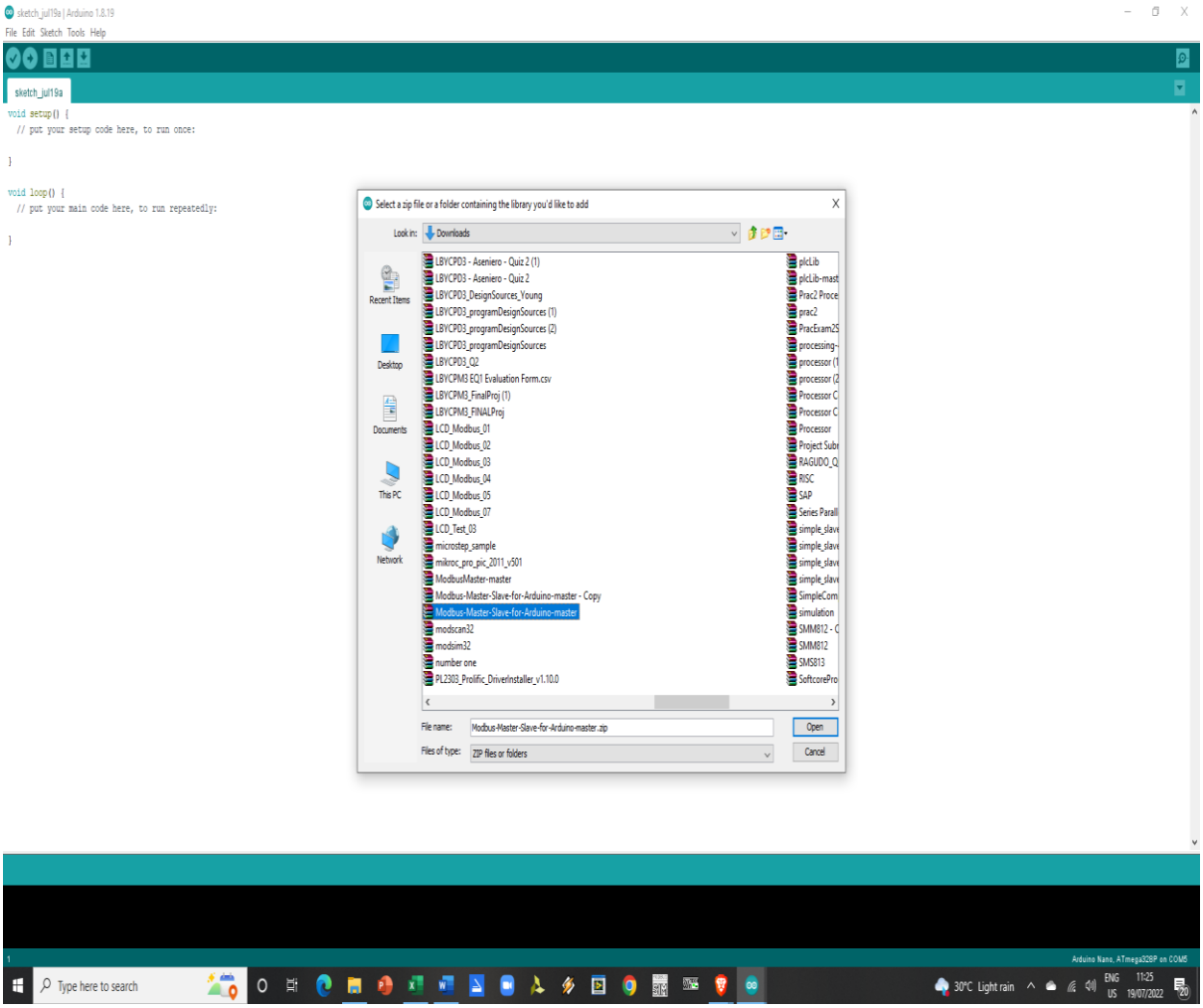


Figure B. Selecting the Library

4. Do step 3 for both the Master Slave and Master Libraries
5. Download files using this link https://drive.google.com/drive/folders/1tOXJwciSB7_lcpMgATGnggSyML6vbXKo?usp=sharing
    a. MODSCAN32
    b. MODSIM32
6. Unpack the files using zip archive tool.
7. Install the VisualStudio on your unit make sure you use community and include .NET functionality.
8. Download and install the Virtual Serial Port kit Software

Systems Design 101

Basic Considerations for a general Systems Designs

In any task being automated there are always three sections to consider. These are:

1. Inputs – Signals / Materials feed into the system. Sometime the signals are taken from sensors that are either analog devices and sometimes digital ones that run on IIC or Modbus standards. For signals its how to obtain them and how to interpret them that matters. Materials on the other hand will have to be accounted for and this creates a signal input that is electrical in nature.
2. Process – is a algorithm or simply a set of tasks done one after the other in most cases the way that the task is administered is dependent on the input signals feed into the system.
3. Outputs – These are signals that are converted into physical movement or energy applied to a certain object.

Now that the items have been briefly discussed lets go into the consideration of what these are. Remember the example is simply a specific set of inputs, process and outputs for one particular purpose. It is not the same for all situations hence an understanding of the process you want to automate is very important when considering automating a system.

Table 1. Example 1 Egg Incubator

Input	Process	Action
Temperature	Application of Heat depending on the set point and the existing temperature	Heater On/Off
Egg		Detection if there is a load of eggs inside
Air	Introduction of Fresh Air / Exhausting Warm Air if temperature overshoots	Fan Operation

Once you can see the input, process and action outline this way you can now ask the necessary questions on specifics.

List of Questions pertaining to the Example 1 Process, Inputs and Outputs outlined.

Like the set temperature of incubating and egg?

What can I use to detect if an egg tray is present?

How do I know if the fan is doing its work?

How will I turn a fan on/off using what particular device?

For each and everyone of the following questions you need to recall your subject on sensors well for this set of questions, how to interface a fan to a controller and what controller to use for the application.

Table 2. Example 1 Egg Incubator more detailed considering the four questions raised

Input	Process	Action
Set Point Temperature = 37-39 Temperature Sensor TMP31A/Thermistor?	OMRON CP1E Arduino Nano / Uno	Use an L293D to route current to the heater
LDR Circuit / IR Detector Circuit		
Temperature of Air measured by same TMP35A but you may need a second sensor for this		Use an L293D to turn off and turn on a Fan Motor

Evaluation of answers

Not in all instances what you think of can be applicable or readily used. It comes down to specification and the ability to source or purchase and obtain the parts. This is a necessary thing to consider in planning a project. You can do a self evaluation simply by lining out the parts and considering the requirements against the specifications, supply and cost of parts

Table 3. Example 1 Evaluation of proposed parts

Part	Function	Pros	Cons	Price in PhP	Quantity
TMP31A	Temp Sensor	Easy to use	Expensive	165	1
Thermistor	Temp Sensor	Cheap	Difficult to use	20	1
LDR	Tray Detector	Cheap and Easy to Use	Values vary depending on Model Type	30	2
Roller Switch	Tray Detector	Cheap and Easy to Find	Bulky you need to consider where to place this	15	2
L293D	Actuator Driver	Can be used until 48v has 6 units in one module	Moderately difficult to source TTL	250	1
Dual Mechanical Relay Driver	Actuator	Can be used for even 220v loads	Low Signal Triggered TTL / CMOS	100	1
Heater Pad	Heating Element	Steady output response	Expensive and Difficult to Source to Specification	1000	1
Signal Lamp	Heating Element	Cheap and somewhat available in market	Slow Response uneven heating	200	1
PLC CP1E	Processor	Easy to program Carries MODBUS	Expensive CP1E has only one Analog input	65000	1
ATMEL 328A Arduino UNO Dev. Platform	Processor	3 Analog Inputs 10 Digital I/O Cheap Can be interfaced to MODBUS	More Complex Programming If MODBUS is used requires added hardware	2500	1

Solution Development

After doing the considerations in Table 3. You can choose either ease of development or low price or a balance of both in your solution development. At this point based on your decision the solution schematic can be laid out. The example schematic is shown in Figure 1.

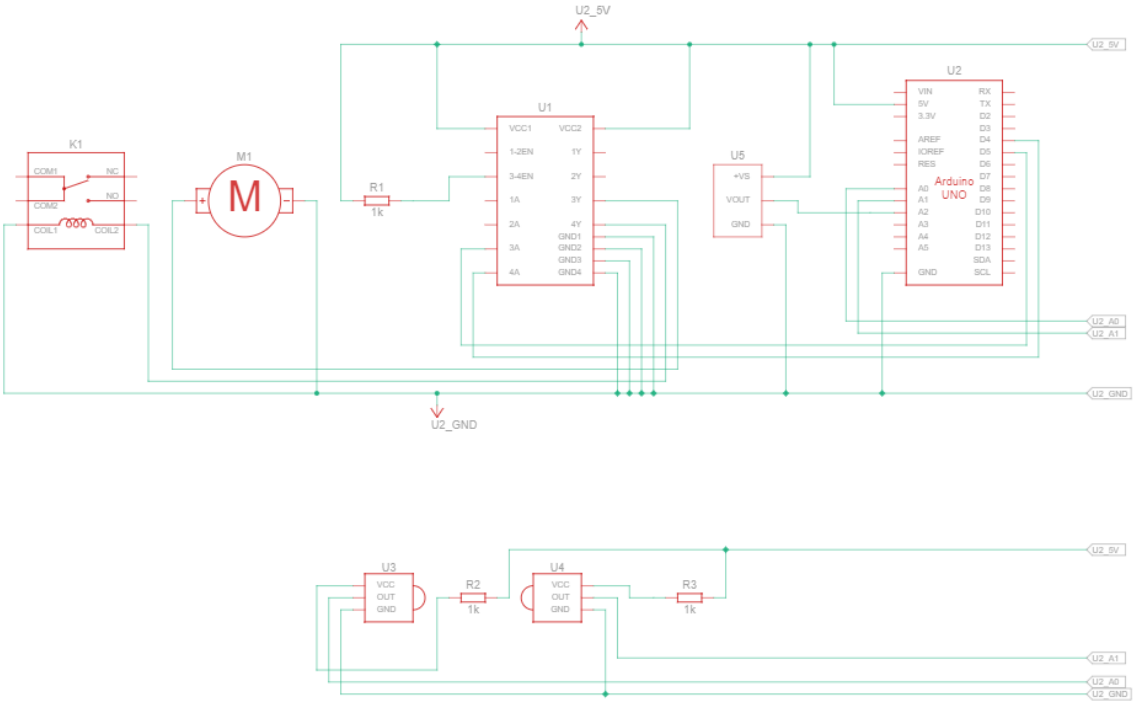


Figure 1. One possible Schematic for the solution

Systems Test and Evaluation

To test your system you need to two kinds of test

1. Technical Test – this is done to see if your equipment / devices work properly. Technical tests are commonly also include timing tests for pumps and sensors performance. For a temperature sensor this would entail determining monitoring values and cut off values for actions in the program like 37 degrees Celsius = 48k ohms for a thermistor. These include
    a. Calibration points
    b. Range of measurable values
    c. Timing Duration
2. Performance Tests – this test evaluates not how specific machines work but how your system actually functions and performs its task. The aim is to determine if it is doing what it is supposed to do and if it needs adjustments. These tests if they go bad will affect production lines and outputs so be cautious on how you conduct these kind of tests.

In each area a checklist is typically generated with a recording of the values associated with desired measurements

Table 4. Technical Tests for Example 1 Project

Device	Function	Testing Parameters	Measured Values at this desired parameter
TMP31A	Temp Sensor	Room Temp Measure 37 Celsius Measure 39 Celsius Measure 36 Celsius Measure 40 Celsius	_____ _____ _____ _____ _____
Thermistor	Temp Sensor	Room Temp Measure 37 Celsius Measure 39 Celsius Measure 36 Celsius Measure 40 Celsius	_____ _____ _____ _____ _____
LDR	Tray Detector	No Tray on top With Tray on top	_____ _____
Roller Switch	Tray Detector	No Tray on top With Tray on top	_____ _____
L293D	Actuator Driver	On Output Voltage Off Output Voltage	_____ _____
Dual Mechanical Relay Driver	Actuator	On Output Voltage Off Output Voltage	_____ _____
Heater Pad	Heating Element	PWM to maintain 38 degrees Celsius or Duration of Time On to Maintain 38 Celsius	
Signal Lamp	Heating Element	PWM to maintain 38 degrees Celsius or Duration of Time On to Maintain 38 Celsius	
Fan Motor	Cooling Element	Duration of Time On to Move from 40 Celsius overshoot to 38	

Table 5. System Test

Date and Time	Temperature Outside in C	Temperature Inside in C	Recorded by	Passed (Temp In) 37-39 C

You can see that Table 4 is very specific to a parameter like the description of a technical test. Table 5 focuses on the action of maintain values within the Range of 37-39. For most cases Table 5 or the system test table can be history based to see if the hardware can match up the desired function.

Standard Documentation for Projects

In projects the designer must create specific documentation in order to record how the system was created and how it improved this is called a design record.

Design Record

The design record has the date of work done and what the form of the project is from physical design, hardware circuits, firmware design and software design. This is typically a single document compiled.

User Manual

The user manual is a document that is created to allow users to learn how to operate the device. It starts with the interface and steps to do to use it. For projects that are both hardware and software both are usually covered by it. For technical jargon the user manual is also known as level 1 access where the machine is operated.

User Manuals have the following parts

I. Machine Name and Description
II. Assembly
III. Operating Specifications
IV. Operating Procedures
V. Maintenance Details

Technical Manuals

The technical manual contains items not needed by a user but needed by a repair personnel or maintenance personnel. The intention is to help him troubleshoot and maintain the device. The usual parts of a technical manual are

I. User Manual
II. Technical Block Diagram
III. Schematic Diagram
IV. Interconnection Diagram
V. Bill of Materials
VI. Replacement Procedure for each part
VII. Inspection Checklist

Activity 0: Plan for the creation of an Automated Water Dispensing Machine.

1. It should be calibrated to dispense water for between x number of hours with the amount of water dependent on the container it is dispensing to.
2. There should be a mechanism that can evaluate moisture level of the water and determine if additional watering is needed between each delivery period.
3. An indicator lamp to state that the device is active and on is needed.
4. A on/off value if mains water line powered or a motorized pump if bucket reservoir water source is used will be the possible means to dispense water to the target.
5. A Block Diagram should be provided.
6. Input – Process – Output Table should be created and documented to evaluate the design.
7. A Wiring Diagram should be created.
8. Program / Ladder Diagram should be documented for the fully functional circuit.

Activity 1A. Build the Machine listed here and create the necessary User and Technical Manual.

Activity 1B: Build an Automated Watering Delivery System (AWDS) for plants. The test period should be around not less than 14 days. This device should have a measurement of

1. Soil Moisture
2. Ambient Temperature
3. Measure the number of times water was delivered at the prescribed rate
4. Be able to show these values on an LCD

The device should deliver a specific amount of water every pre set period of time.

MODBUS Basics

History

Modbus is a transmission format system published in 1979 in Modicon® mainly intended for PLC.
Modus uses the format of one Master and multiple slaves up to 247 units
Data transmission is formatted into the following

Table 1. MODBUS Transmission Frame

Start	Address	Function	Data	LRC	End
:	2 Chars	2 Chars	N Chars	2 Chars	CR LF

Address pertains to the following

Table 2. MODBUS Address Names and Types of Action

Coil/Register Numbers	Data Addresses	Type	Table Name
1-9999	0000 to 270E	Read-Write	Discrete Output Coils
10001-19999	0000 to 270E	Read-Only	Discrete Input Contacts
30001-39999	0000 to 270E	Read-Only	Analog Input Registers
40001-49999	0000 to 270E	Read-Write	Analog Output Holding Registers

Function Pertains to the following

Table 3. Function Codes in MODBUS

Function Code	Action	Table Name
01 (01 hex)	Read	Discrete Output Coils
05 (05 hex)	Write single	Discrete Output Coil
15 (0F hex)	Write multiple	Discrete Output Coils
02 (02 hex)	Read	Discrete Input Contacts
04 (04 hex)	Read	Analog Input Registers
03 (03 hex)	Read	Analog Output Holding Registers
06 (06 hex)	Write single	Analog Output Holding Register
16 (10 hex)	Write multiple	Analog Output Holding Registers

Longtidunal Redundancy Check (LRC) refers to an error checking mechanism.

For Full Systems this is not LRC but CRC of Cyclic Redundancy Checks

To get a better understanding think of MODBUS as a communications and filing system. Each processor / controller in an Operational Technology Environment has memory and how that memory is arranged and can be accessed is something that falls in line with MODBUS.

Example in Omron PLCs the inputs are designated as 000.0 – 000.16 depending on the model of the PLC used convention can fall into an understanding that input pins actually fall into the category that this is accessible as Register Number 000 with bits 0 -16 being available for individual checking. The same goes for outputs designated as 100.0-100.16 in Omron PLC's this refers to Register Address 100 with the individual bits being represented by the second number after the decimal point.

For this book we will use the following assignments

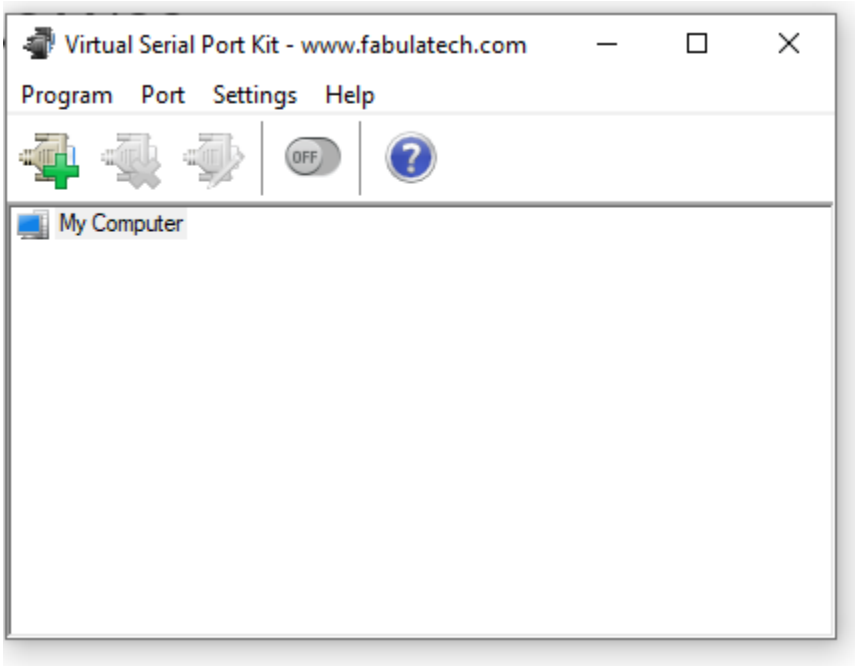
Table 4. How declared addresses and functions combine into PLC Address Values.

Function Code	Register Number	Type	Name	PLC Address Values
4:	0001	Read	Input Bits Status	40001
4:	0002	Write	Trigger to Change Output Bits	40002
4:	0003	Read	Status of Output Bits	40003
4:	0004	Read	Analog 0 Value	40004
4:	0005	Read	Analog 3 Values	40005
4:	0006	Write	MODBUS Triggering for Inputs	40006

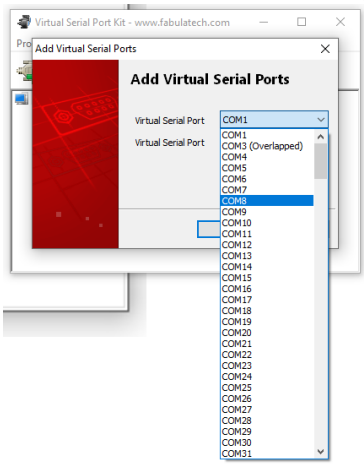
In order to understand this better secure from the download link the tools needed to implement a MODBUS Slave and Master Simulation using MODSIM32 and MODSCAN32

Virtual MODBUS Exercises:

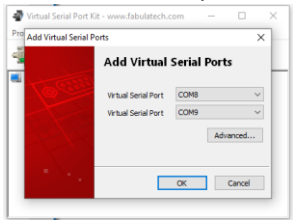
1. Open the Virtual Serial Port



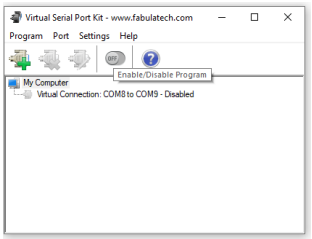
2. Click the Add Com Port Pair (+) icon
3. Use Serial Ports that are free (not overlapped)



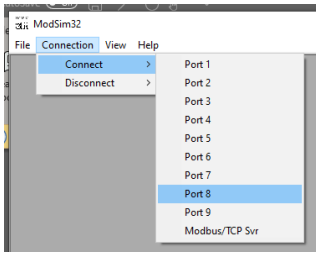
4. Do this for both ports and press OK to set the settings into record.



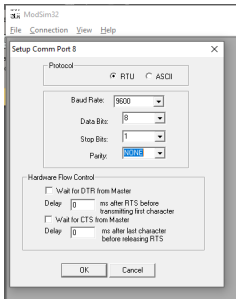
5. Click on the Enable Switch to activate the virtual port.



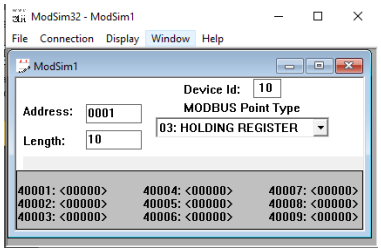
6. Turn on MODSIM32 and use one of the COMPORTS you enabled on the Virtual Serial port you can do this by



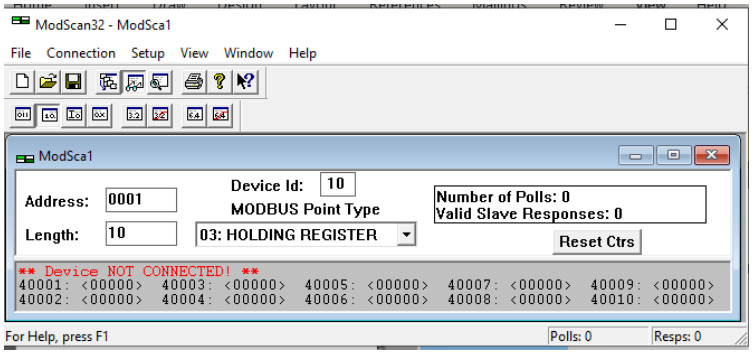
7. A pop up window will come out make sure to set the Baud Rates, Parity, LRC, Stop Bits and Data Size to the values shown below this step



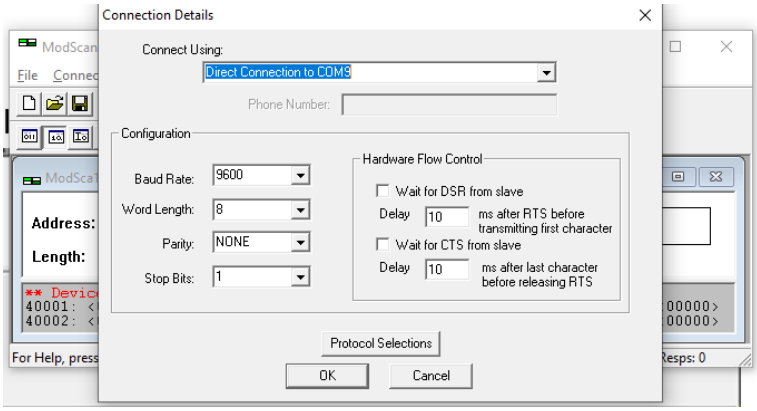
8. Create a new session using File -> New
9. Set up the values with the following values



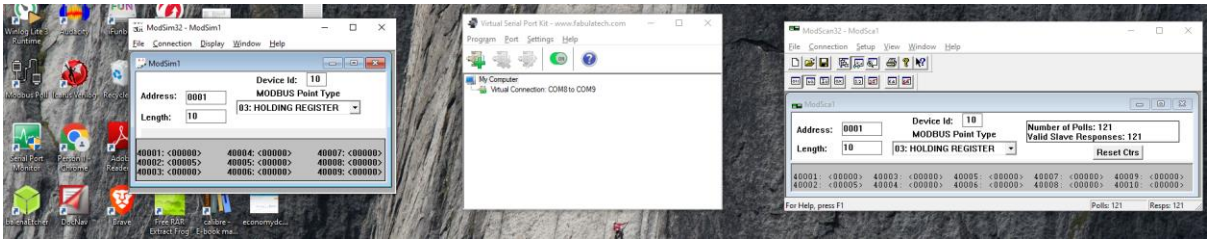
10. Open MODSCAN32
11. Setup a Session using File -> New
12. Make sure the values are set to the Values listed below



13. Click on the connection option on the menu bar set the parameter to those listed below



14. Arrange the Windows of the three applications to make it easy to see what is taking place between the three



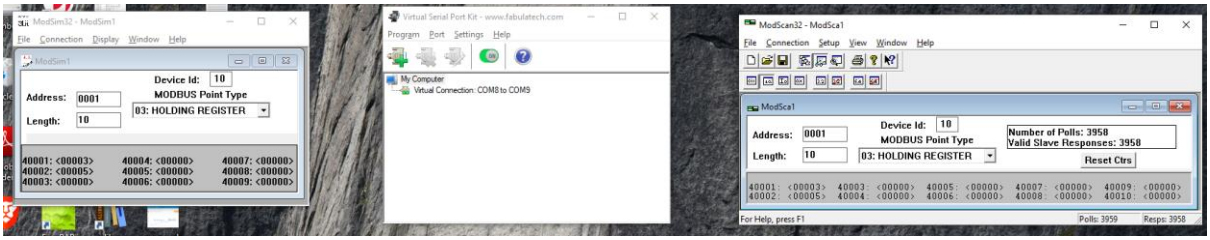
15. Click on 40001: <0000> a window popup should appear that shows how you can change the values on register 0001 with a function of 4 or read input register.
16. Type 3 and press Update Button.

Explanation of Virtual MODBUS Exercise

MODBUS is a both a storage system of analogue data and digital I/O values. The virtual exercise shows the reader how changes on a single MODBUS Slave the MODSim32 application can cause this changes to the same registers inside the MODBUS Master played by MODScan32 application. The transmission is direct and if we can only see the memory registers we don't really see that the master is requesting data from the slaves and slaves answering.

For the Virtual Exercise we selected the MODSIM32 to be Device 10 along Channel 1. Remember the MODBUS Master must communicate directly to the intended slave device by specify the correct Device ID. If you change the Device ID of the MODSCAN32 to you can see that its does to no reactions if you change the values. It even lists a MODBUS TimeOUT Error

17. Watch as the value on 40001:<0000> on the ModScan changes to 3 as well.



18. Try to do the same for the other holding registers.

Activity 1. Modbus RTU Slave Implemented in Arduino

Write the program listed below this paragraph and use MODSCAN32 to obtain the values using MODBUS remember the slave id is 10 and the Baud Rate is 9600. These values should be configured on the MODSCAN32 Side along with the COM PORT that is associated with the MODBUS RTU Arduino Unit we programmed you can see that on the detected COMPORT you used to program the device unless you are using a Due or NANO 33 BLE where the programming port is not the transceiver port that is readily used.

Program 1. Basic MODBUSRTU Arduino Slave

```
#include <ModbusRtu.h>

// registers in the slave and their contents
uint16_t au16data[16] = { 3, 1415, 9265, 4, 2, 7182, 28182, 8, 0, 0, 0, 0, 0, 0, 1, -1 };

Modbus slave(10,0,0); // this is slave @10 and RS-232 or USB-FTDI
```

```
void setup()
{
  slave.begin( 9600 ); // baud-rate at 9600
}

void loop()
{
  slave.poll( au16data, 16 );
}
```

Authors Notes on Activity 0 and 1.

Take note the command slave.poll (data container name, number of elements) is the way that the program updates the MODBUS Scanner of any changes to the values listed within each register and obtains as well any changes to the values listed inside the scanner.

This implies that if you intend to use MODBUS RTU on your Arduino you should not use block delays instead use non-blocking (millis()) based delays to ensure that your data values updates are not delayed or your HMI / MODBUS Scanner freezes up.

Try adding a 3000 mS delay to the loop section of Activity 1. Program you will see that your Modscan32 will hang each time the processor does the blocking delay of 3000 mS even a 2000 mS cause some lag time.

In Activity 0 where we test our ModSIM32 Modbus slave while in Activity 1 we test our Arduino MODBUS implementation. Why are we using both?

During the Arduino Modbus – PC Modbus activities we mainly use the Modscan32 application but once we start to build HMI's we need to use ModSIM32 first to test the interface before we plug in our Arduino Modbus Hardware.

Activity 2: Pre Discussion Capitalizing and using MODBUS RTU on Arduino

Now that we have used and ran a basic application on MODBUS using the simulator and an actual hardware device lets start to utilize the Hardware side to display some values that we need on actual design. These are commonly either

1. Input Status Bits – the state of inputs configured inside the ATMEL 328A processor operating the Arduino Development Board
2. Output Status Bits - the state of outputs configured inside the ATMEL 328A processor operating the Arduino Development Board
3. Analog Input Values - the state of analog to digital conversions configured and connected to sensors on the Arduino Development Board
4. Output Bit MODBUS Triggers – this forces the outputs to be set or reset according to the MODSCAN32 set values.

Now items 1 to 3 on Page 18 are Holding Register Values and should not be written on only Read. While item 4 on Page 18 is meant to be written on and read. In any application you develop using this type of format you as the programmer need to set what is read only and what are read and write holding registers for your MODBUS RTU Arduino Hardware.

Example Table of Specifications Design

We need to be clear what holding registers are Read Only and Read and Write to make this into a constructive task let use the table below:

Table 5. Holding Register Specifications

PLC MODBUS Address	Use of Register	Read Only	Write – Read	audata[] index number
40001	Input Pins State	X		0
40002	Output Pins Trigger		X	1
40003	Output Pins State	X		2
40004	Analog A0 Value	X		3
40005	Analog A3 Value	X		4
40006	Inputs Trigger		x	5

Having such a table in the program and design stage of the project will make sure that declared changes to the data values done with the program are uniform and no tainting of the data is done accidentally.

This would mean that for table 5 having statements such as:

Example to encode Polling of Bits

```
bitWrite( au16data[0], 0, digitalRead( 2 )); //
bitWrite( au16data[0], 1, digitalRead( 3 ));
```

where the bit values of au16data[0] changed by the actual value of the two input push button switches is okay since the Modscan / HMI will only read those values and not to write to them.

We can see a similar setup where the processor is the only one allowed to write into a register that is used by the modbus transmission with the code that obtains Analog Data from A0 and A3 pins.

```
au16data[2]=analogRead(A0);
au16data[3]=analogRead(A3);
```

That having the code

Example of setting outputs on / off using MODBUS / processor coding by use of the logic function or in the conditions

```
void updateouts(int x) // Checks status of au16data[1] bits and assigned them to bits 4-6 and 13
{                              // Updates au16data[2] values
  if(bitRead(au16data[1],0)==HIGH || x==0) { digitalWrite(4,HIGH); bitWrite(au16data[2]=HIGH,0,1); }
  if(bitRead(au16data[1],1)==HIGH || x==1) { digitalWrite(5,HIGH); bitWrite(au16data[2]=HIGH,1,1);}
  if(bitRead(au16data[1],2)==HIGH || x==2) { digitalWrite(6,HIGH); bitWrite(au16data[2]=HIGH,2,1);}
 if(bitRead(au16data[1],3)==HIGH || x==3) { digitalWrite(13,HIGH); bitWrite(au16data[2]=HIGH,3,1);}
  if(bitRead(au16data[1],0)==LOW || x==0) { digitalWrite(4,LOW); bitWrite(au16data[2]=HIGH,0,0); }
  if(bitRead(au16data[1],1)==LOW || x==1) { digitalWrite(5,LOW); bitWrite(au16data[2]=HIGH,1,0); }
  if(bitRead(au16data[1],2)==LOW || x==2) { digitalWrite(6,LOW); bitWrite(au16data[2]=HIGH,2,0); }
 if(bitRead(au16data[1],3)==LOW || x==3) { digitalWrite(13,LOW); bitWrite(au16data[2]=HIGH,3,0);}
}
```

Is also okay since au16data[1] is meant to be something that the HMI/Modscan can read or write to while au16data[2] is only written to by the processor and read only the HMI/Modscan application.

Activity 2 provides a working example of a formed set of holding registers used in the manner stated by Table 5.

Activity 2. Modbus RTU with Defined Memory Mapped for Holding Registers

1. Encode the program 2
2. Run a syntax check.
3. Load it into your Arduino.
4. Use the ModSCAN configured to the Arduino associated COMPORT, Slave ID 10

Program 2. MODBUS RTU Slave with Read Only and Read-Write Registers from the MODSCAN / HMI prespective.

```
#include <ModbusRtu.h>

// registers in the slave
uint16_t au16data[16] = { 3, 0, 0, 4, 2, 7182, 28182, 8, 0, 0, 0, 0, 0, 0, 1, -1 };
// { input pins state, output pin trigger, output pin state, Analog A0, Analog A3, input pins trigger }
//    40001              40001              40003              40004        40005      40006
// au16data7-au16data15 are free for people to use

Modbus slave(10,0,0); // this is slave @1 and RS-232 or USB-FTDI

void setup() {
  pinMode(12, OUTPUT); // rs
  pinMode(11, OUTPUT);  // e
  pinMode(7, OUTPUT); // d7
  pinMode(8, OUTPUT);  //d6
```

```
  pinMode(9, OUTPUT);  // d5
  pinMode(10, OUTPUT);  //d4

  pinMode(13, OUTPUT); // Output use
  pinMode(6, OUTPUT);  // Output use
  pinMode(5, OUTPUT);  // Output use
  pinMode(4, OUTPUT);  // Output use
  pinMode(2,INPUT_PULLUP); // 2 Input use
  pinMode(3,INPUT_PULLUP); // 3 Input use

// A1, A2 A4 and A5 are unused for

  slave.begin( 9600 ); // baud-rate at 9600

 lcd.begin(16, 2); // Setup of LCD
  lcd.home(); // Housekeeping
  lcd.clear();
  lcd.write("MODBUS RTU ");
  lcd.setCursor(0,1);
  lcd.write("Template");
  delay(2000);
  lcd.clear();
  lcd.write("V.B. Dupo, ECE ");
  lcd.setCursor(0,1);
  lcd.write("19072022");
  delay(4000);
  lcd.clear();
  lcd.write("CH 1 DEV 10");
  lcd.setCursor(0,1);
  lcd.write("0001-0016");
  delay(4000);
  lcd.write("IN:2,3 AIN:A0,A3");
  lcd.setCursor(0,1);
  lcd.write("OUT:4,5,6,13");
  delay(4000);
}

void updateouts(int x) // Checks status of au16data[1] bits and assigned them to bits 4-6 and 13
{                              // Updates au16data[2] values
  if(bitRead(au16data[1],0)==HIGH || x==0) { digitalWrite(4,HIGH); bitWrite(au16data[2]=HIGH,0,1); }
  if(bitRead(au16data[1],1)==HIGH || x==1) { digitalWrite(5,HIGH); bitWrite(au16data[2]=HIGH,1,1) ;}
  if(bitRead(au16data[1],2)==HIGH || x==2) { digitalWrite(6,HIGH); bitWrite(au16data[2]=HIGH,2,1) ;}
 if(bitRead(au16data[1],3)==HIGH || x==3) { digitalWrite(13,HIGH); bitWrite(au16data[2]=HIGH,3,1);}
  if(bitRead(au16data[1],0)==LOW || x==0) { digitalWrite(4,LOW); bitWrite(au16data[2]=HIGH,0,0); }
  if(bitRead(au16data[1],1)==LOW || x==1) { digitalWrite(5,LOW); bitWrite(au16data[2]=HIGH,1,0); }
  if(bitRead(au16data[1],2)==LOW || x==2) { digitalWrite(6,LOW); bitWrite(au16data[2]=HIGH,2,0); }
 if(bitRead(au16data[1],3)==LOW || x==3) { digitalWrite(13,LOW); bitWrite(au16data[2]=HIGH,3,0);}
}

void loop() {
  slave.poll( au16data, 16 );
```

```
  //Poll Bits
  bitWrite( au16data[0], 0, digitalRead( 2 )); //
  bitWrite( au16data[0], 1, digitalRead( 3 ));

  // Check for bits to be set
  updateouts(6);

  // Obtain Analog Data
  au16data[2]=analogRead(A0);
  au16data[3]=analogRead(A3);

  delay(1000);

}
```

5. Disconnect your Arduino from the Programming USB
6. Connect pins 2 and 3 to Pushbutton section of your Arduino E-Gizmo Trainer,
7. While Pins 12 – 7 are connected to the RS, E, D4,D5,D6 and D7 terminals.
8. Connect the 5v pin and GND of the Arduino to the 5v and the GND connectors of the Board these are terminals the upper left of the bread board on the E-Gizmo Trainer.
9. Connect pins 13, 6, 5 and 4 to the LED Bank connector on the upper right corner of the Trainer Board.
10. Plug in your USB to power the circuit
11. Observe if the LCD comes alive and displays the following information

    MODBUS RTU → V.B.DUPO, ECE → CH 1 DEV 10 → IN:2,3 AIN:A0,A3
    Template 19072022 0001-0016 OUT:4,5,6,13

12. Open your MODBUS Scan 32 Application Connect to the MODBUS COMPORT associated with the ARDUINO you are using. Remember to set the Baud to 9600 and Slave ID to 10.
13. Check the data status of 4001, 4003, 40004 and 40005 as these are read only registers.
14. Change the value of register 40002 from 0 to 1 and watch the value of 40003 change from 0 to 1. And you should see one of the LED's light up.
15. If you write 7 then the value register 40003 also changes to 7 meaning all three outputs should be set to high and the LEDS connected to pins 4,5 and 6 all lit on.

Activity 3: Pre-discussion

Now that we can see the values are being sent and received by the ModScan32 application we can proceed to create an LCD Interface that intended to show status values to the user in front of the machine not on a remote terminal or MODBUS Scanner Terminal. To accomplish this we need to use another technique observed when creating applications that use modbus on a processor this is called the processor equivalent register.

Any MODBUS implementation done on Arduino has to have pairing of MODBUS Used Registers and Processor Registers particularly for the registers that are used for read and Write Functions. This was advised to help keep the MODBUS registers free from erroneous writing and another reason is to process any display updates by comparing the pair with each other.

I commonly execute this by using:

```
uint16_t au16data[16] = {3, 0, 0, 4, 2, 7182, 28182, 8, 0, 0, 0, 0, 0, 0, 1, -1 };
int ARdata[16] = {3, 0, 0, 4, 2, 71, 2, 8, 0, 0, 0, 0, 0, 0, 1, -1 };
```

Comparing is done with:

```
if(ARdata[x]==au16data[x]) { what to do if condition is true }
```

The to do part is divided into three tasks:

1. Updating the value of ARdata[x] by using the command:
   ARdata[x]=(au16data);

2. Setting a update latch to force the LCD to make an update after all data points checked for updates are covered and primed. This is done by the code below:

   ```
   refreshkita=HIGH;
   ```

   Take note refreshkita is a Boolean value declared as a global variable it is set to low during startup and set to low again once the lcd finishes writing the new values into the display memory.

3. Any other tasks associated with the update. This could be starting a pump or resetting it. It really depends on the data value listed and how it affects your program.

Activity 3 Focuses on the implementation of such a code change take note program 3 in Activity 3 is a modification of Activity 2s program 2.

Activity 3. MODBUS and a Functional LCD Display with Values

1. Encode the code listed in program 3. You may obtain your copy of program 2 from your computer and just add the missing sections found on program 3 on to it.

2. Run your Code check first before you encode the program into the hardware.

Program 3. LCD Update for MODBUS Input values

```
#include <ModbusRtu.h>
#include <LiquidCrystal.h>

// registers in the slave Do not remove this unless you want the Modbus to not work
uint16_t au16data[16] = {3, 0, 0, 4, 2, 7182, 28182, 8, 0, 0, 0, 0, 0, 0, 1, -1 };
int ARdata[16] = {3, 0, 0, 4, 2, 71, 2, 8, 0, 0, 0, 0, 0, 0, 1, -1 };

bool refreshkita;

LiquidCrystal lcd (12, 11, 10, 9, 8, 7); // rs enable d4, d5, d6, d7
Modbus slave(10,0,0); // this is slave @1 and RS-232 or USB-FTDI

void setup() {
  pinMode(13, OUTPUT); // Output for Motor Control
  pinMode(12, OUTPUT); // rs
  pinMode(11, OUTPUT);  // e
  pinMode(7, OUTPUT); // d7
  pinMode(8, OUTPUT);  //d6
  pinMode(9, OUTPUT);  // d5
  pinMode(10, OUTPUT);  //d4
  pinMode(6, OUTPUT);  // LED
  pinMode(5, OUTPUT);  // LED
  pinMode(4, OUTPUT);  // LED
  pinMode(2,INPUT_PULLUP); // 2 Input
  pinMode(3,INPUT_PULLUP); // 3 Input
   lcd.begin(16, 2); // Setup of LCD
  lcd.home(); // Housekeeping
  lcd.clear();
  lcd.write("MODBUS RTU ");
  lcd.setCursor(0,1);
  lcd.write("Template");
  delay(2000);
  lcd.clear();
  lcd.write("V.B. Dupo, ECE ");
  lcd.setCursor(0,1);
  lcd.write("19072022");
  delay(4000);
  lcd.home();
  lcd.clear();
  lcd.write("IN:2,3 AIN:A0,A3");
  lcd.setCursor(0,1);
  lcd.write("OUT:4,5,6,13");
  delay(4000);
  lcd.clear();
  lcd.write("CH 1 DEV 10");
  lcd.setCursor(0,1);
  lcd.write("0001-0016");
  delay(4000);
```

```
  slave.begin( 9600 ); // baud-rate at 9600
  refreshkita=LOW;
  updateprompts();
}

void updateouts(int x)
{
  if(bitRead(au16data[1],0)==HIGH || x==0) { digitalWrite(4,HIGH); bitWrite(au16data[2],0,1);}
  if(bitRead(au16data[1],1)==HIGH || x==1) { digitalWrite(5,HIGH); bitWrite(au16data[2],1,1);}
  if(bitRead(au16data[1],2)==HIGH || x==2) { digitalWrite(6,HIGH); bitWrite(au16data[2],2,1);}
  if(bitRead(au16data[1],3)==HIGH || x==3) { digitalWrite(13,HIGH); bitWrite(au16data[2],3,1);}
  if(bitRead(au16data[1],0)==LOW || x==4) { digitalWrite(4,LOW); bitWrite(au16data[2],0,0);}
  if(bitRead(au16data[1],1)==LOW || x==5) { digitalWrite(5,LOW); bitWrite(au16data[2],1,0);}
  if(bitRead(au16data[1],2)==LOW || x==6) { digitalWrite(6,LOW); bitWrite(au16data[2],2,0);}
  if(bitRead(au16data[1],3)==LOW || x==7) { digitalWrite(13,LOW); bitWrite(au16data[2],3,0);}
}

void updateprompts()
{
  lcd.home();
  lcd.clear();
  lcd.write("IN:");
  lcd.print(ARdata[0]);
  lcd.write(" A0:");
  lcd.print(ARdata[3]);
  lcd.setCursor(0,1);
  lcd.write("OUT:");
  lcd.print(ARdata[2]);
  lcd.write(" A3:");
  lcd.print(ARdata[4]);
  refreshkita=LOW; // Turns off refreshkita trigger
}

void loop() {
  slave.poll( au16data, 16 );

  //Poll Bits
  bitWrite( au16data[0], 0, digitalRead( 2 )); //
  bitWrite( au16data[0], 1, digitalRead( 3 ));

  // Check for bits to be set
  updateouts(6);

  // Obtain Analog Data
  au16data[3]=analogRead(A0);
  au16data[4]=analogRead(A3);

// Program Mode

  if(au16data[0]==1) { updateouts(0); }
  if(au16data[0]==2) { updateouts(1); }
```

```
 if(au16data[0]==0) { updateouts(0); updateouts(1); }
 if(au16data[0]==3) { updateouts(4); updateouts(5); }

// checks for updates to value to refresh lcd

if(ARdata[0]!=au16data[0]) { ARdata[0]=au16data[0]; // Refresh values for Input
                    refreshkita=HIGH;  // prompt to refresh
                   }
if(ARdata[2]!=au16data[2]) { ARdata[2]=au16data[2]; // Refresh values for Output
                    refreshkita=HIGH;  // prompt to refresh
                   }
if(ARdata[3]!=au16data[3]) { ARdata[3]=au16data[3]; // Refresh values for Analog A0
                    refreshkita=HIGH;  // prompt to refresh
                   }
if(ARdata[4]!=au16data[4]) { ARdata[4]=au16data[4]; // Refresh values for Analog A3
                    refreshkita=HIGH;  // prompt to refresh
                    }
if(refreshkita==HIGH) { updateprompts(); } // Updates values if refreshkita is set to high

delay(500);

}
```

3. Load it into your Arduino.
4. Disconnect your Arduino from the Programming USB
5. Use the ModSCAN configured to the Arduino associated COMPORT, Slave ID 10
6. Plug in your USB to power the circuit
7. Power up the Modscan32 by connecting into the associated comport that you used to program your Arduino Uno / Nano.
8. Observe if the LCD comes alive and displays the following information

   MODBUS RTU → V.B.DUPO, ECE → CH 1 DEV 10 → IN:2,3 AIN:A0,A3
   Template 19072022 0001-0016 OUT:4,5,6,13

   → IN: 3 A0: 1024
   OUT: 0 A3: 1024

9. Open your MODBUS Scan 32 Application Connect to the MODBUS COMPORT associated with the ARDUINO you are using. Remember to set the Baud to 9600 and Slave ID to 10.
10. Check the data status of 4001, 4003, 40004 and 40005 as these are read only registers.
11. Change the value of register 40002 from 0 to 1 and watch the value of 40003 change from 0 to 1. And you should see one of the LED's light up and the value at the LCD OUT: change from 0 to 1:
12. If you write 7 then the value register 40003 also changes to 7 meaning all three outputs should be set to high and the LEDS connected to pins 4,5 and 6 all lit on while the value at the LCD OUT changes to 7.

This program demonstrates the ability to change the LED's and reflect the changes to the LCD. But do note that the LCD changes its value if and only if any of the values inside au16data[0],

au16data[2], au16data[3] and au16data[4] changes this code is seen on the // Checks for updates to value comment area seen in the last page 26.

Activity 1B: MODBUS into the Automated Watering Delivery System (AWDS) for plants.

You're improvement should have the ability to:

1. Change Settings without reprogramming (MODBUS)
    a. Duration of waiting period
    b. Delivery duration for motor on function
    c. Ability to clear the counter
2. Include a refresh routine for your LCD each time the counter value increases.

Internal EEPROM of ATMEL328A using it to your advantage for non-volatile data storage in projects.

Memory there are two types of memory for computers

1. RAM – Random Access Memory This is made from registers they store data but when the power is turned off that data is lost.
2. ROM – Read Only Memory This is made from basically hot wired lines of tin that would stay conducting or shorted to represent 0 and open to represent 1.
    a. The original ROM was only programmable by factory / forges.
    b. The current technology allows that the ROM can be Electrically Erasable and Programmable. This is best seen in the Arduino Program memory which is an EEPROM (Electrically Erasable Programmable Read Only Memory).

Structure of Memory

The table below represents the structure of basic memory be it a ROM / RAM Device. There is always the data bus and the address.

Address	Data Bus bits							
	7	6	5	4	3	2	1	0
00								
01								
02								
03								
….								
999								

The address as the name suggest is used to point to the memory location where the data in this case (8 bits in Length) will be deposited to.

For devices such as the Arduino Uno or Nano this address can go from 0 to 999 making it 1000 addresses.

Writing to RAM

In writing to the RAM we often use the command for declaring variables such as

int <variable name>;

The variable name is the address while the size of the container is determined by the declared variable type.

int - 2 bytes long

long – 4 bytes long

bool – 1 bit long

byte – 8 bits long or 1 byte

Storing values into them requires that you use the concurrent declarations such as :

```
int var;
var=82;
```

In the above code implies that a variable with the address name of var was created and it was assigned the number 82.

The utilization of C-like code in the implementation of Arduino caused this easy nature of accessing and using RAM in ATMEL 328A utilized inside the Arduino development platform.

EEPROM on the other hand requires the utilization of three commands.

```
#include<EEPROM.h>  // Standard Library to utilize the EEPROM
EEPROM.read(address); // Reads the contents of the address stated
EEPROM.write(address,data); // Write to the EEPROM address stated the value listed inside the data
                            //place holder
```

Activity 1C: Integrating Non-Volatile Memory Storage in Arduino MODBUS RTU into the Automated Watering Delivery System (AWDS) for plants

This improvement should use the program developed for Activity 1B and it should be able to:

1. Store the present counter value to EEPROM memory 0.
2. Store the Duration in Minutes to EEPROM memory 20.
3. Store the Interval Duration of Motor Activation in Seconds inside EEPROM memory 50.
4. Upon Restart the values listed inside Memory Locations 0, 20 and 50 are loaded into au16data[0], au16data[3] and au16data[4] respectively.

Human Machine Interface

Human Machine Interfaces or HMI's for short are just graphical presentations of a process / monitoring screen to show that a current device status is and what it is doing or has done. These devices are created to allow visualization of the process from a remote location from where the process is located at or to understand what is happening to a machine that is currently active.

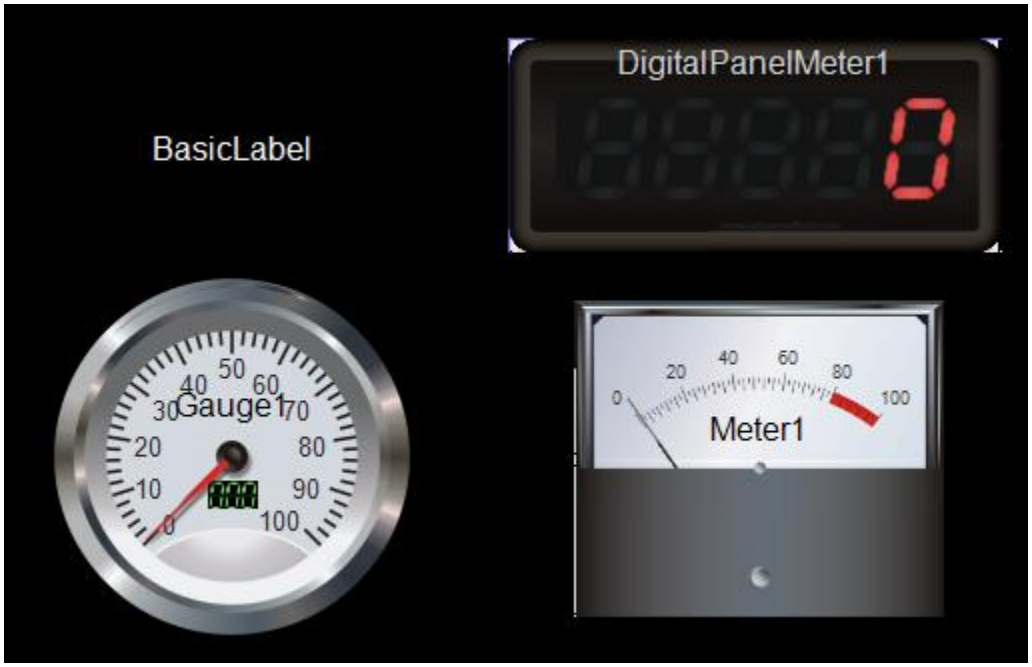
HMI's created for industrial applications typically use MODBUS as their Network and Data Link layer while the physical is handled by either RS232 or USB or RS485 or IEEE 802.1 depending on what you want to use as your transmission system. For this book we shall just use USB as our Physical Layer this is very similar to RS232 Transmission systems.

The beauty of having to create and field an HMI is that your process monitor will not need any added training to oversee a process he/she simply interacts with the HMI to force a system to adjust its settings or to see its current status.

Rules on Creating HMI

Before you start with any HMI Creation work make sure to figure out and state what the parameters you want to see, manipulate and how you want to see these parameters.

The appearance can be any of the following ones from Advanced HMI



These best to use for Analog Values but for discrete inputs and outputs LED type indicators are best us to use but our implementation of MODBUS uses Holding (4) and Input (3) Registers not discrete holding coils (0) or input coils (1) so we are mostly stuck with these type of indicators unless you assign one Modbus RTU Register to each one of your output and input pins.

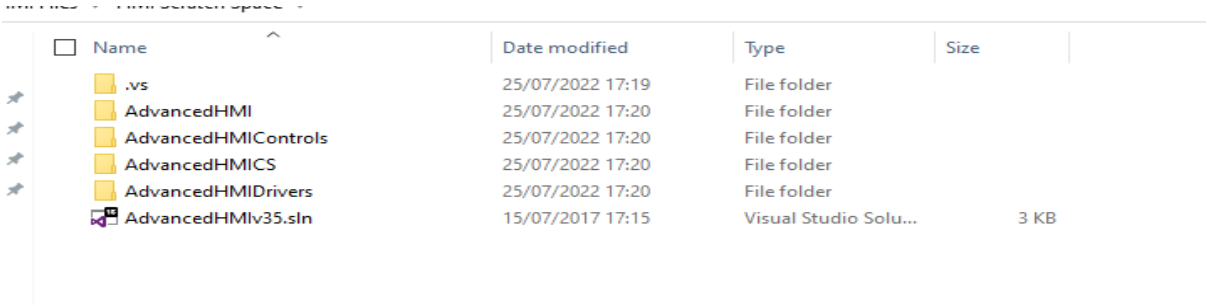
For this you may use the table below:

Table 6.Planning your HMI layout based on table 5.

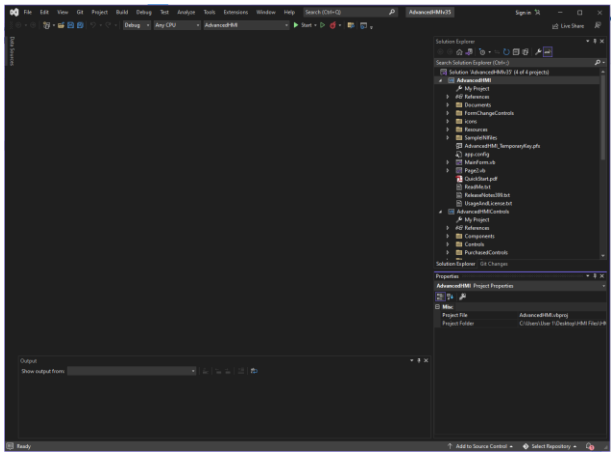
PLC MODBUS Address	Use of Register	Read Only	Write – Read	Desired Appearance
40001	Input Pins State	X		Digital Panel Meter
40002	Output Pins Trigger		X	None at this time (Not Implemented)
40003	Output Pins State	X		Digital Panel Meter
40004	Analog A0 Value	X		Meter Type
40005	Analog A3 Value	X		Gauge Type
40006	Inputs Trigger		x	None at this time (Not Implemented)

Advanced HMI steps on how to use it

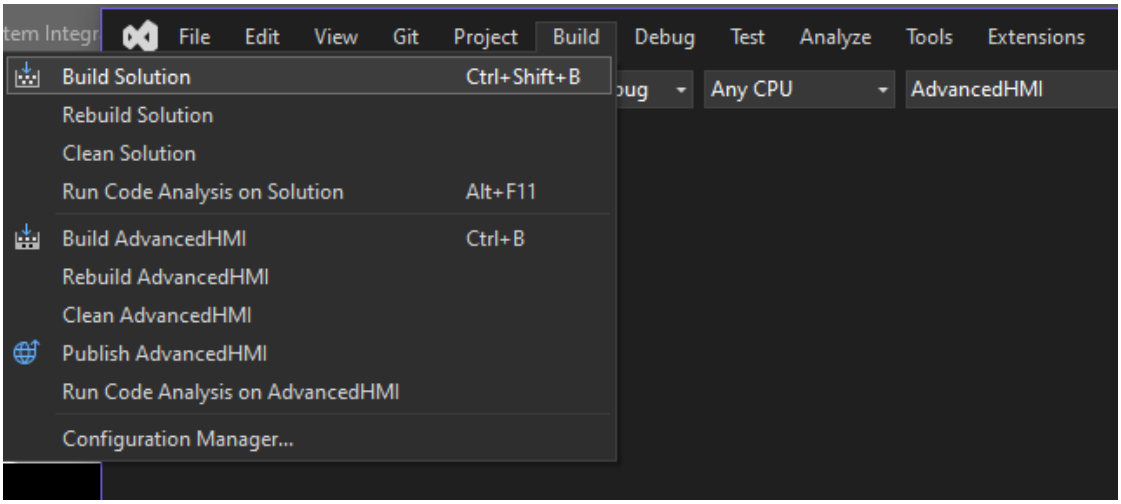
1. Download the Advanced HMI solution from their website advancedhmi.com
2. Install Visual Studio 2022 Community on your PC.
3. Unpack AdvancedHMI and open the sln file.



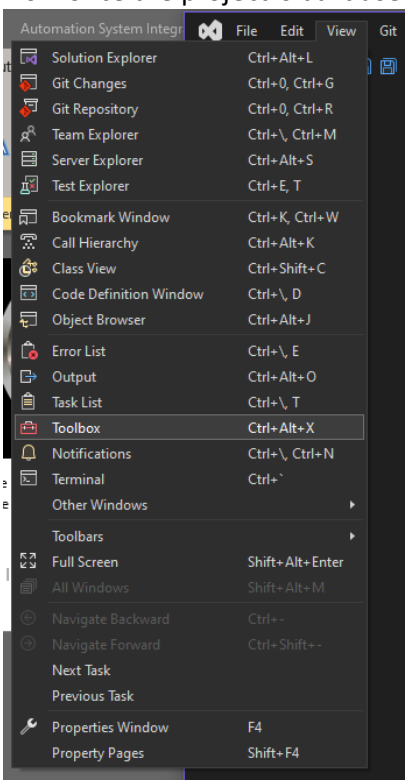
4. This will load the file into visual Basic.
5. You will be greeted by the screen shown below



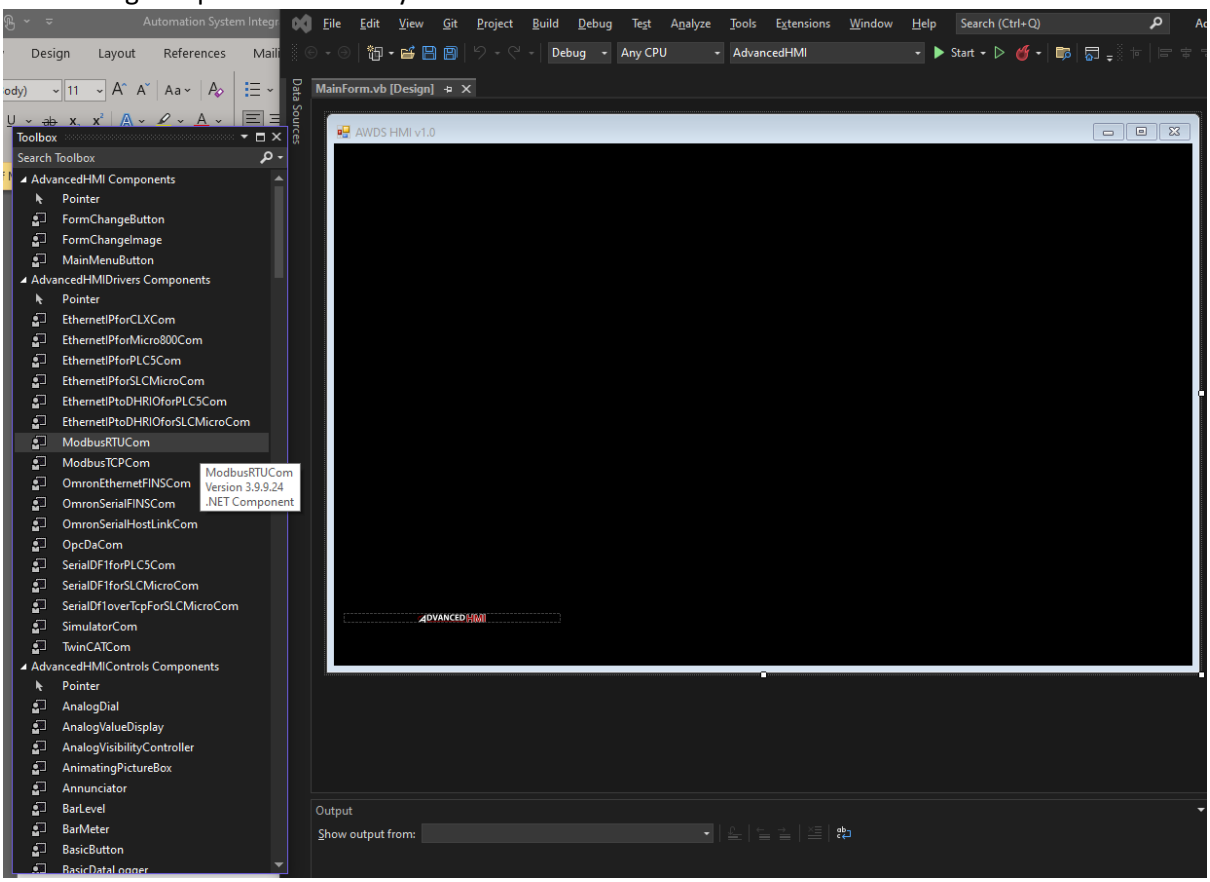
6. You need to build the solution first do this by using Build→ Build Solution



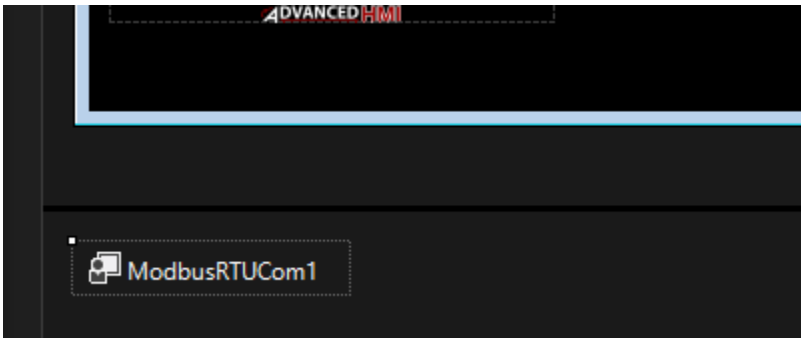
7. Once built you can launch your GUI Screen Builder by going inside the Solution Explorer and into the Advanced HMI Solution locating and clicking on the file MainForm.vb
8. Now once the project is built use View→ Toolbox to show your toolboxes



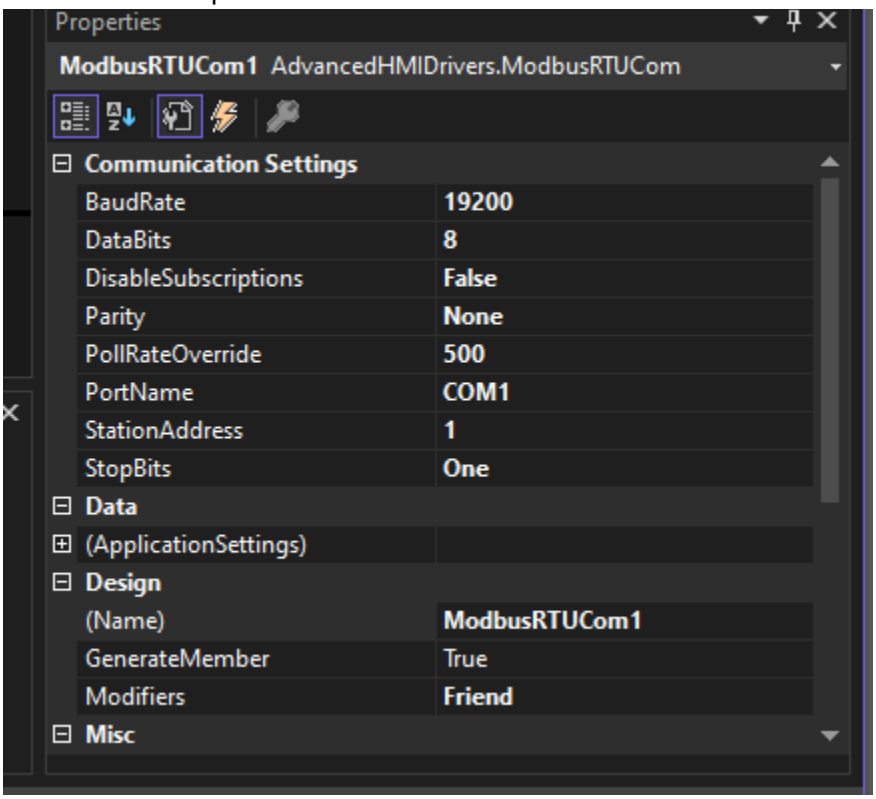
9. On the AdvancedHMI Drivers Components section of the tool box click on MODBUS RTU. Select Drag and pull it into the Layout screen area.



10. Once its active in the GUI Area you should see at the bottom left of the gui screen the words MODBUSRTUCOM1.

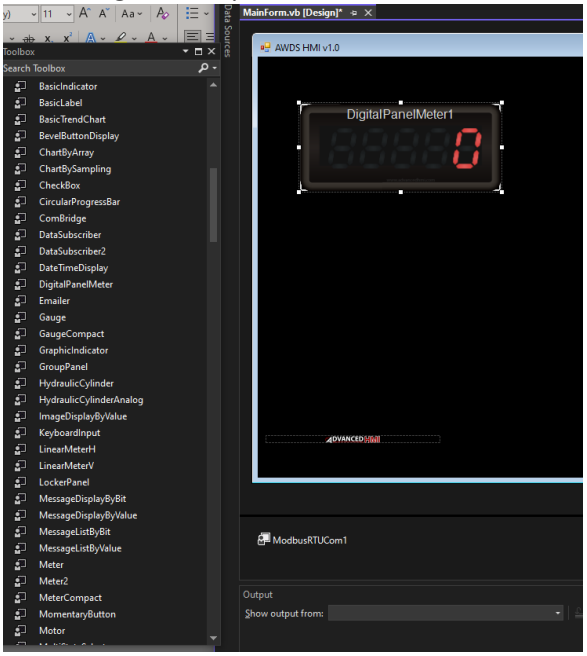


11. Click on this and on the bottom right area of your Visual Studio Workscreen a properties window should open.



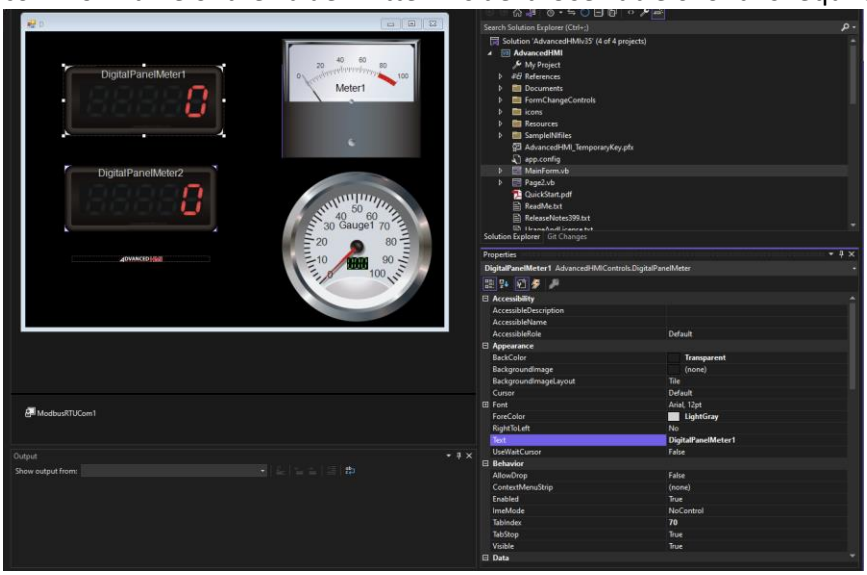
12. Change the Station address to match your Baud rates, Slave ID of choice and the COMPORT to one that you intend your hardware to or your MODSIM32 to using Visual Serial Port.
13. On the Toolbox go to the AdvanceHMIControls Components and browse thru the title names until you find the one listed in Table 6.

14. Click, Drag and Drop it into the GUI Work Screen area

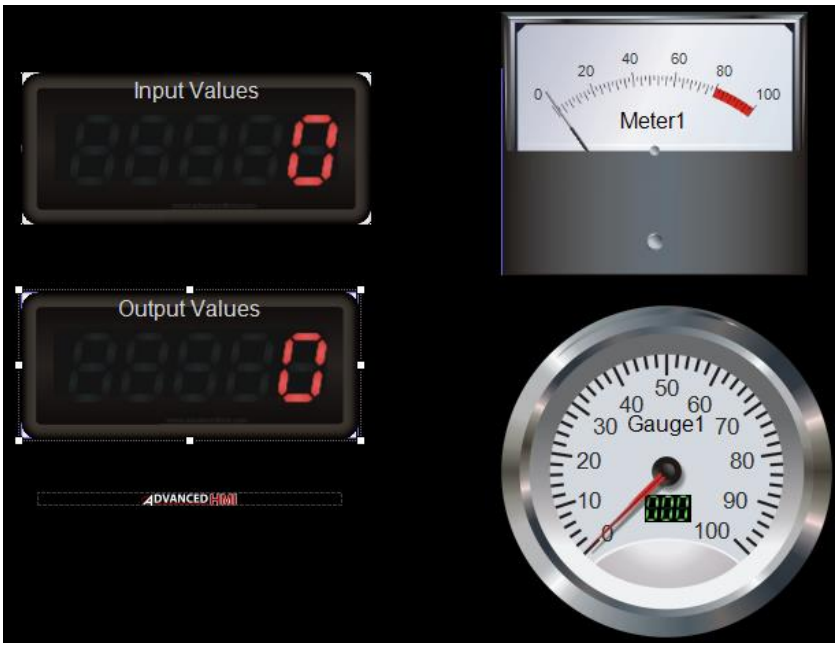


15. Do the same for the other required items until you end up with this layout

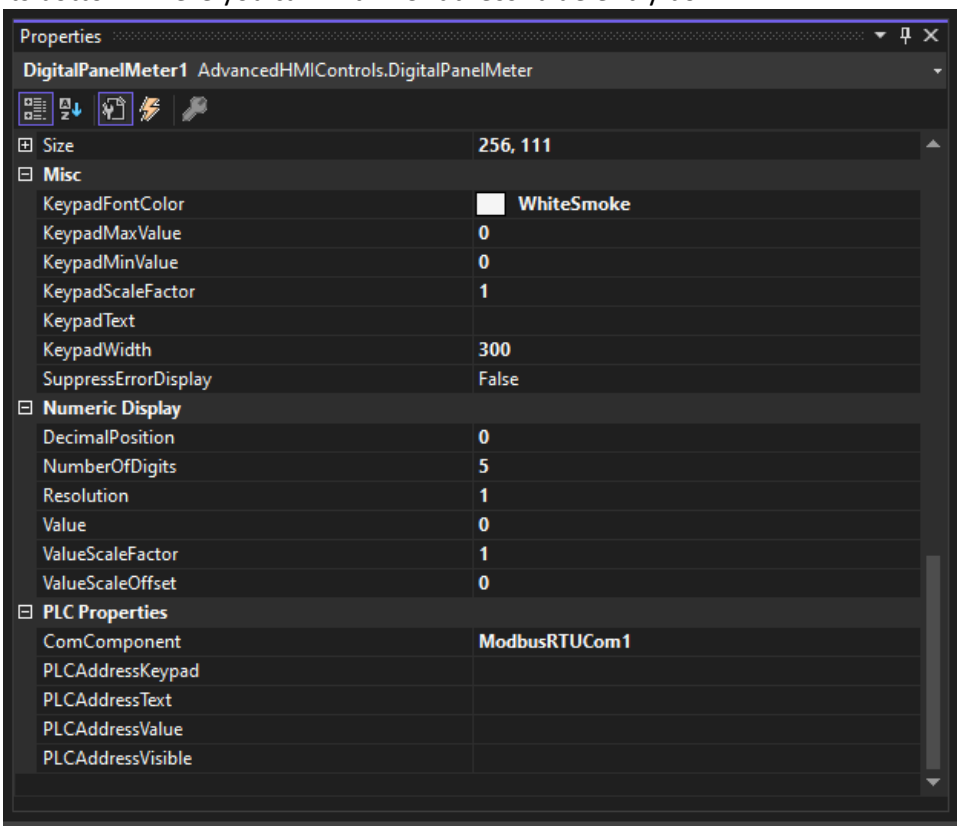
16. Click on the first digital meter and change the name of the panel meter to your desired common name of the value written inside it. Use Table 6 for this requirement.



17. Do the same for the second digital meter

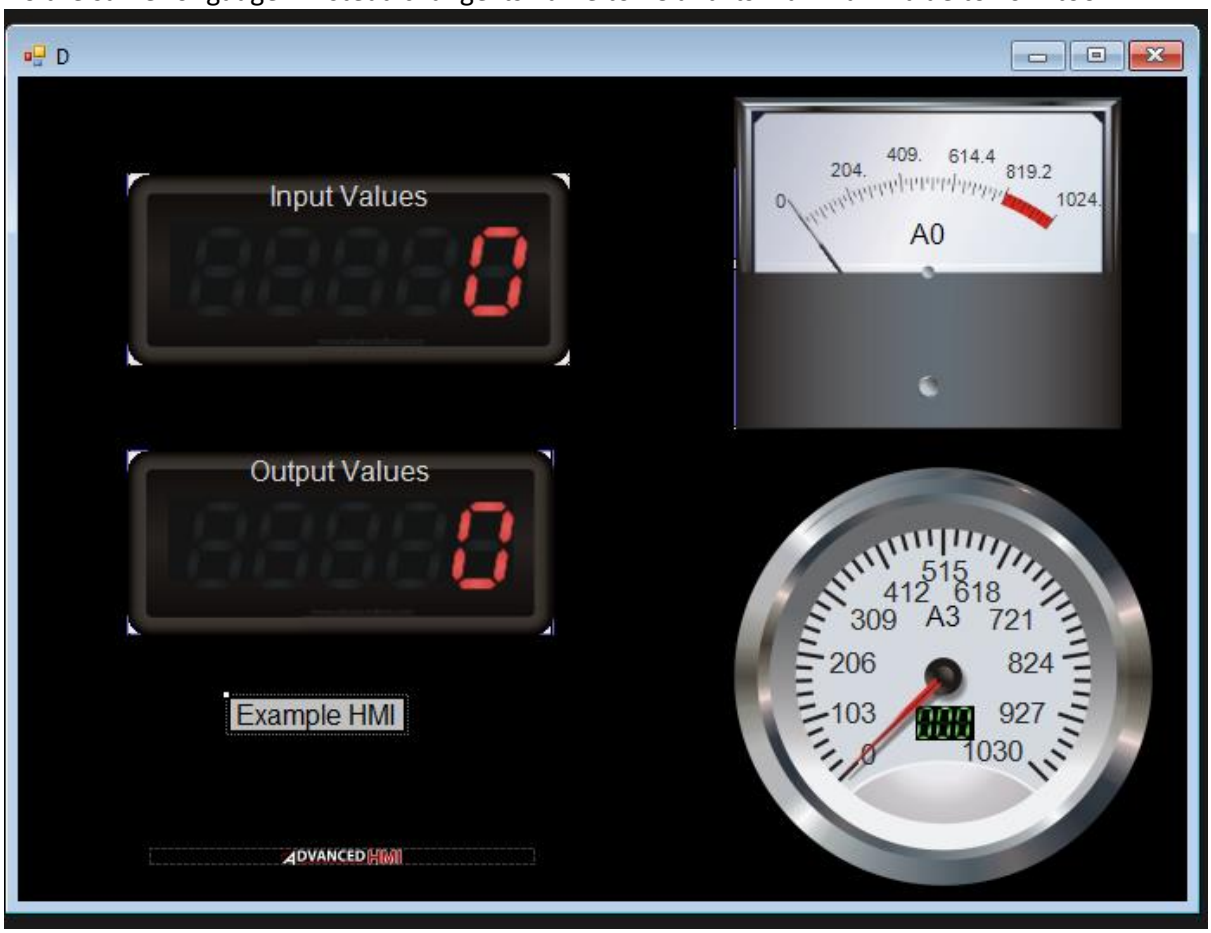


18. Click back to the Input Values Digital Meter Face and on the properties area scroll down to its bottom where you can find PLCAddressValue entry box



19. Use Table 6 to type in the register used for this which is 40001.
20. Do the same for the Output Values and Meters 1 and Gauge 1 assigning them to 40003, 40004 and 40005.
21. Click on Meter 1. Change its name to A0
22. On the properties tab scroll down to Misc Label and change the maximum value to 1024 and the minimum to 0.
23. This will change the meter face to this

24. Do the same for gauge 1 instead change its name to A3 and its maximum value to 1024 too.

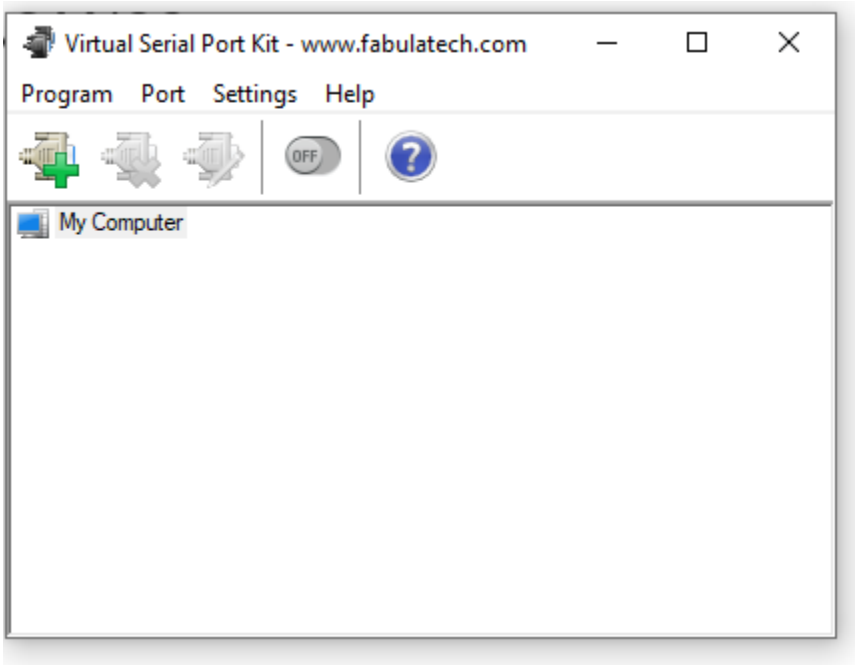


25. When you reach this point you are ready to test your HMI solution.

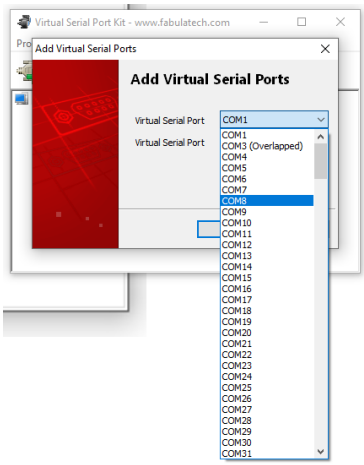
Software Bench Testing your HMI

This section contains steps that are needed to evaluate your HMI before you connect it to the Arduino.

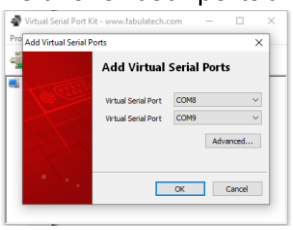
1. Open the Virtual Serial Port



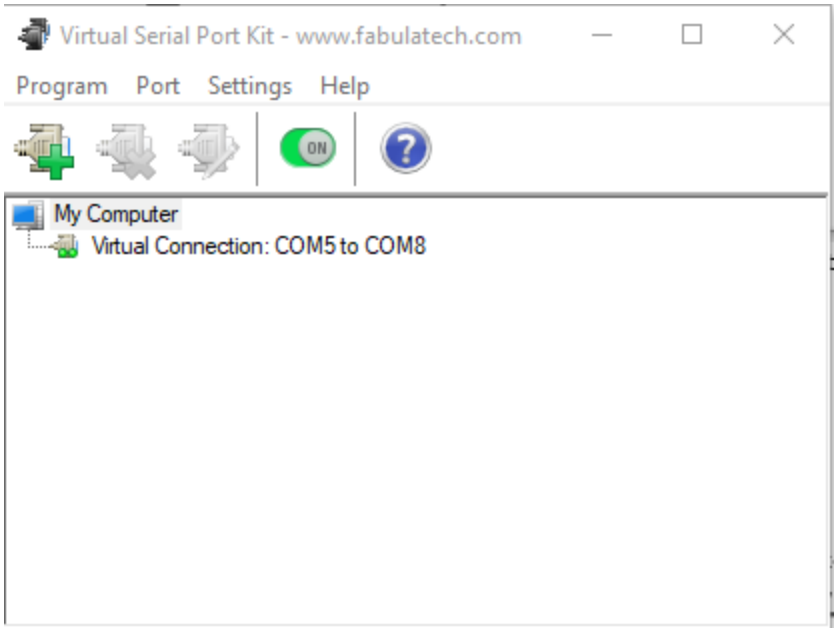
2. Click the Add Com Port Pair (+) icon
3. Use Serial Ports that are free (not overlapped)



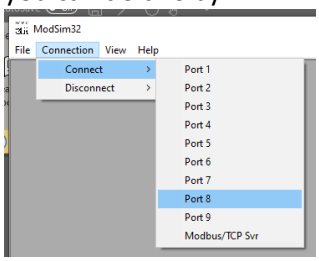
4. Do this for both ports and press OK to set the settings into record.



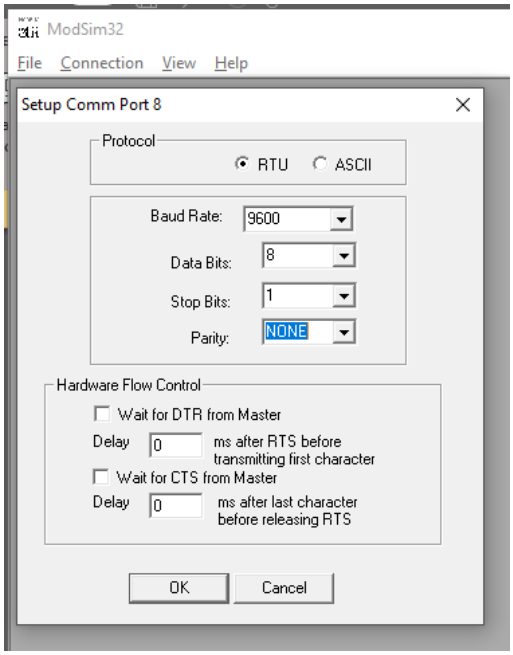
5. Click on the Enable Switch to activate the virtual port.



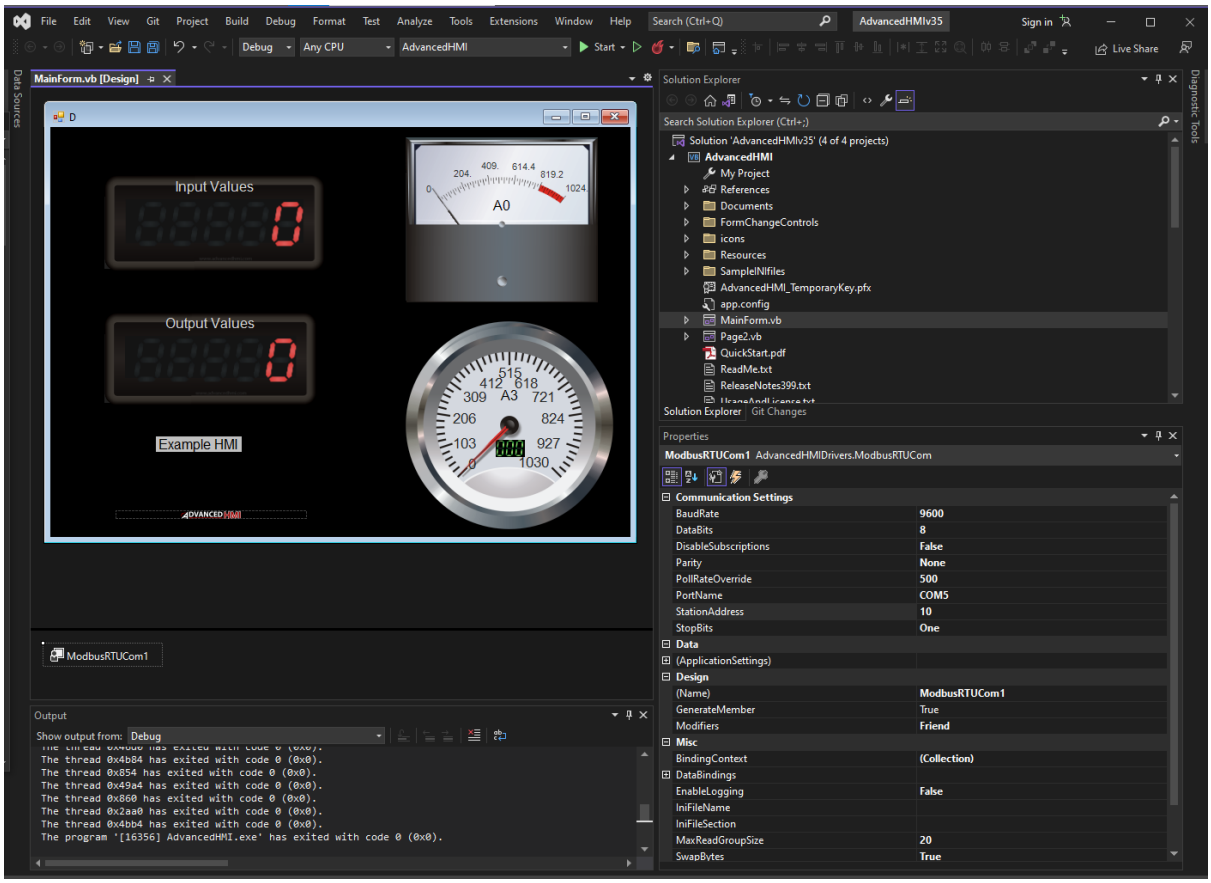
6. Turn on MODSIM32 and use one of the COMPORTS you enabled on the Virtual Serial port you can do this by



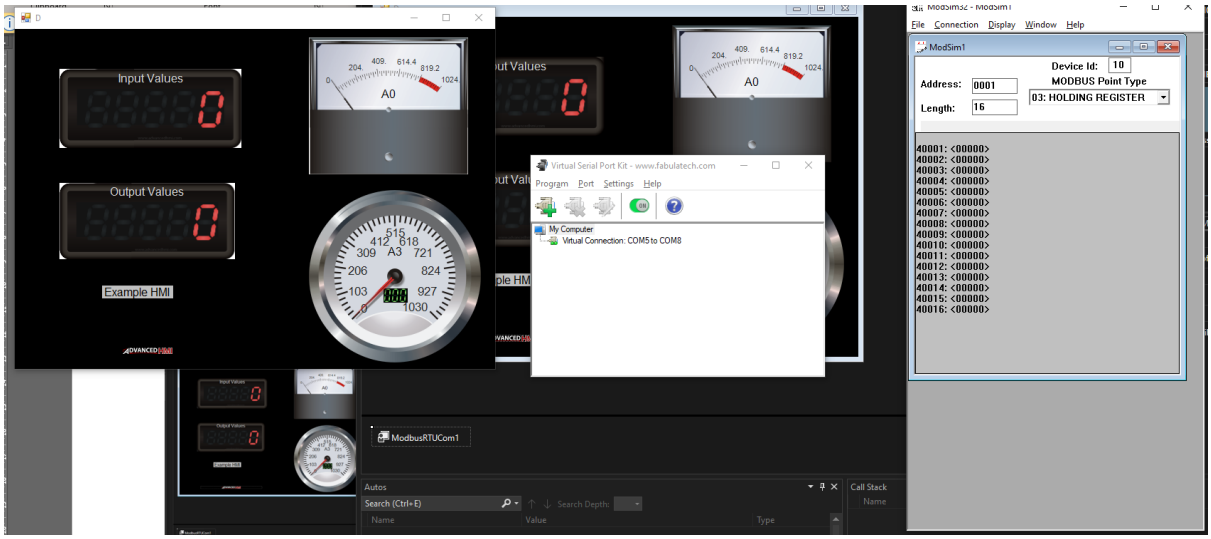
7. A pop up window will come out make sure to set the Baud Rates, Parity, LRC, Stop Bits and Data Size to the values shown below this step



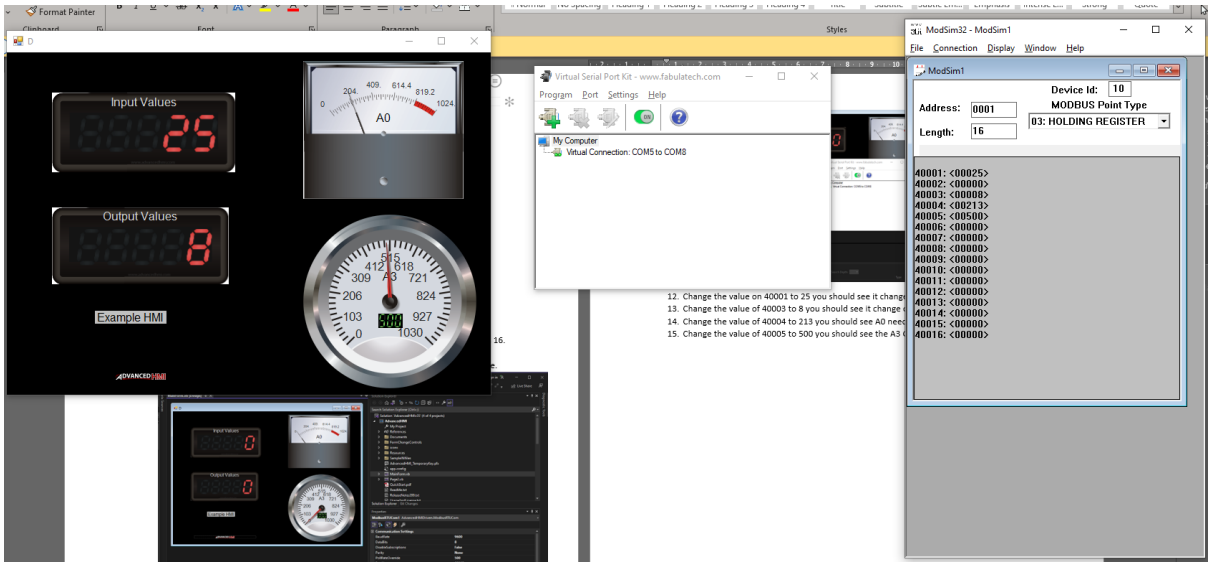
8. Create a new session using File -> New
9. Set it up for the Session ID to be 10; Type - Holding Registers and Number of Registers 16. Starting Address is 0001
10. Make sure to adjust the HMI Modbus RTU to COM5 and Station 10 and 9600 Baud Rate.



11. Once set press the start button to run the solution and test the codes. You setup should be consisting of the HMI, Virtual Serial Port and Modbus SIM32



12. Change the value on 40001 to 25 you should see it change on Input Values
13. Change the value of 40003 to 8 you should see it change on the Output Value
14. Change the value of 40004 to 213 you should see A0 needle move to the right
15. Change the value of 40005 to 500 you should see the A3 Gauge move the center



16. Disconnect your ModSim32 using Connection → Disconnect → Port 8
17. Turn off your Virtual Serial Port
18. At this point your HMI is working and can be connected to the Arduino

Migrating your Work from AdvancedHMI Solutions Workarea to an executable

This is a simple task:

1. Go to the folder that contains the advancedHMI.sln file you used to launch the application.
2. Select the AdvancedHMI folder
3. Inside the folder you will see another set of folders select and go into the bin folder
4. Inside the bin folder select the Debug Folder you should see your solution file in .exe form there.
5. Note that this executable form is soft coded to the Comport you set the Modbus RTU at so be careful to make sure that your device is detected on that computer thru this comport. Otherwise you may need a virtual serial port kit software to the link the hardware port to the incorrect connection of the HMI.

Activity 1D: Connecting your AWDS Modbus, EEPROM, LCD to the HMI.

1. Prepare an HMI for your Modbus AWDS. Use the Advanced HMI steps and how to use it instructions to make your own layout.
2. Test it using a Modbus Simulator first before testing it on your device.

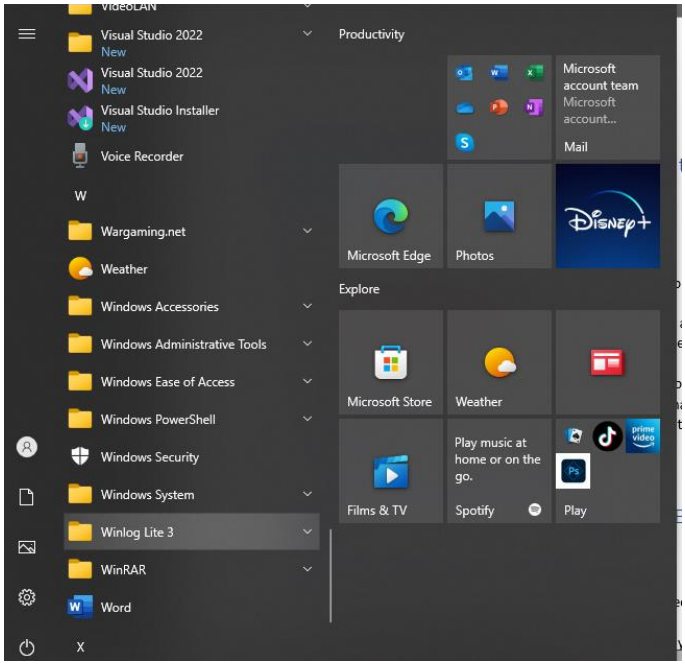
Using Winlog Lite 3.0

The application we developed in the previous section used Visual Studio and Advanced HMI. Understanding how to use the solution package based on Visual Studio is a good starting point for HMI development but it only exposes you the learner to only one type of software. So now I want to introduce the commercial software called Winlog in this case its lite edition. This was a recommended software by my instructor since its PC based but the end product if you have the pro edition can be ported to a java applications. You may download this software thru

https://www.sielcosistemi.com/en/download/public/winlog_lite.html#:~:text=Winlog%20Lite%20is%20the%20free,60%20minutes%20of%20full%20operation.

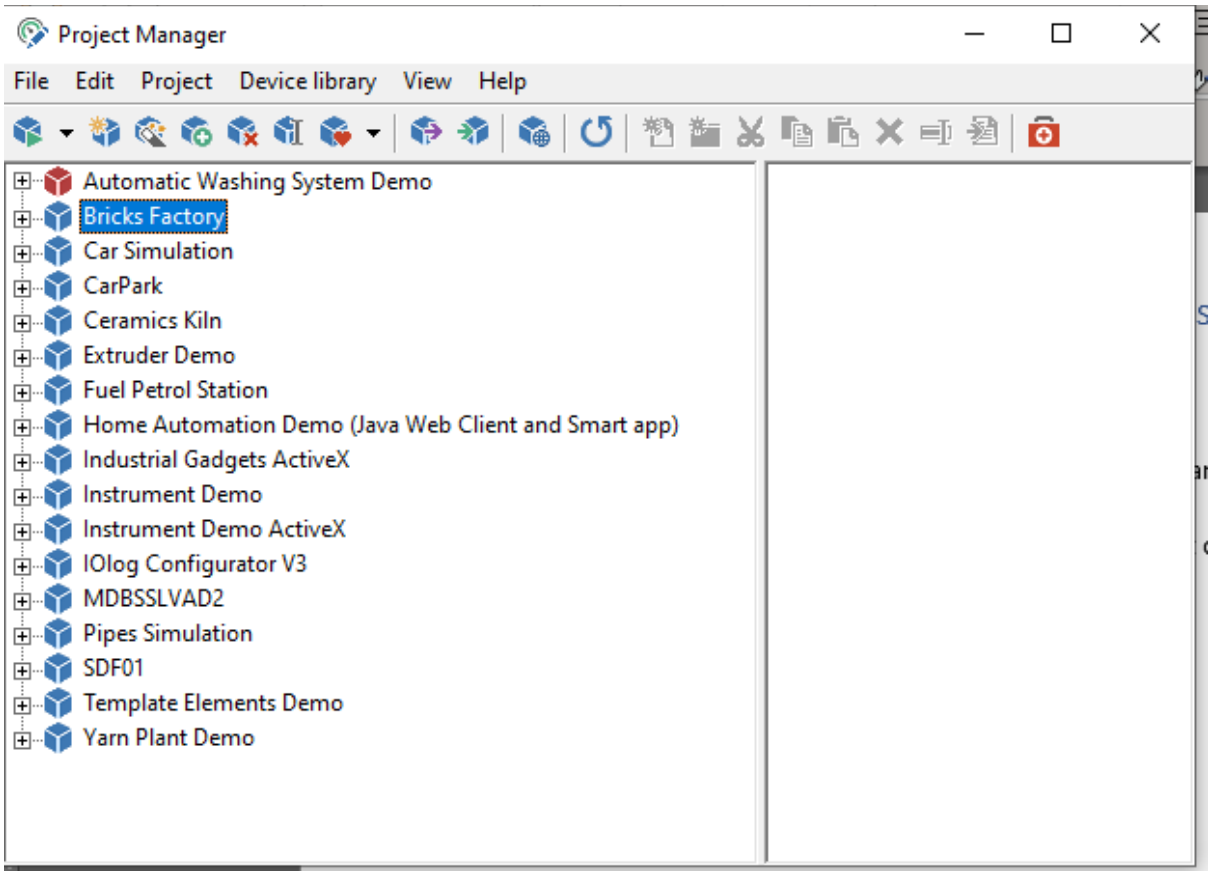
Preparation for Use and Opening the Program

Sielco Sistemi emailed me that Winlog Lite 3.0 as this July 2022 is now not covered by a paid edition which is a shame instead they sell the Winlog Pro edition. This program (Winlog Lite) needs to be installed. After installation you can find it on the W section of the applications menu area.



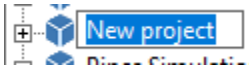
Click on the Winlog Lite 3 folder and find and click on the Project Manager Application

This will open up a new window that looks like



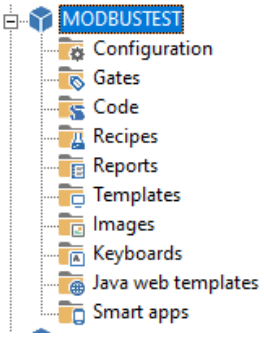
To create a new file click on the box icon with the sun.

This should lead to a new project being declared.

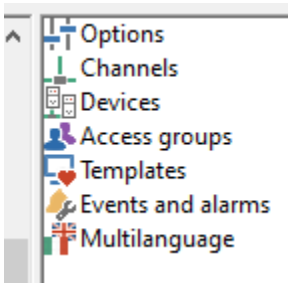


You can rename this to your liking.

Expand the project to show its contents

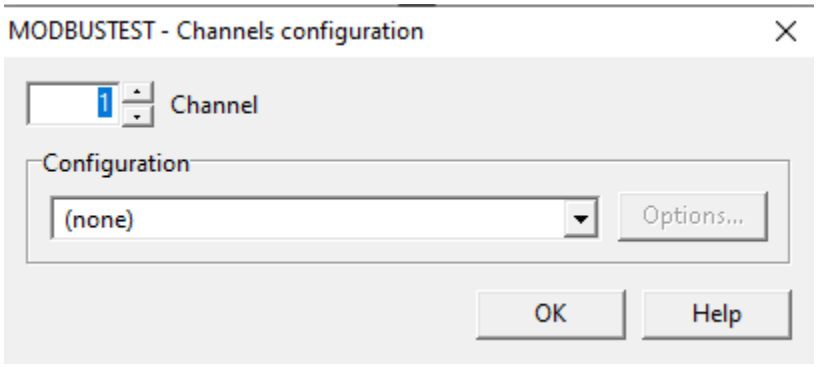


Click on the configuration folder it should show you the following

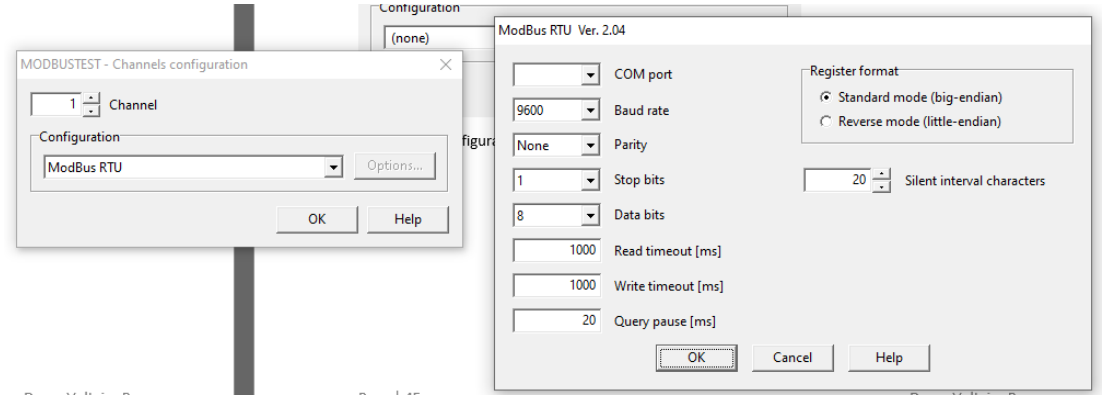


Setting up Modbus communication

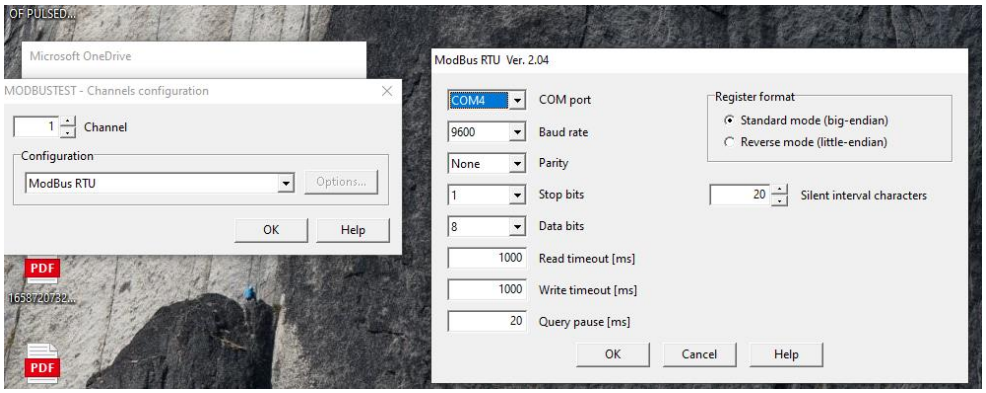
Double Click on the Channels. It should open up a new popup window



Click on the configuration, find MODBUS RTU and select it from the menu



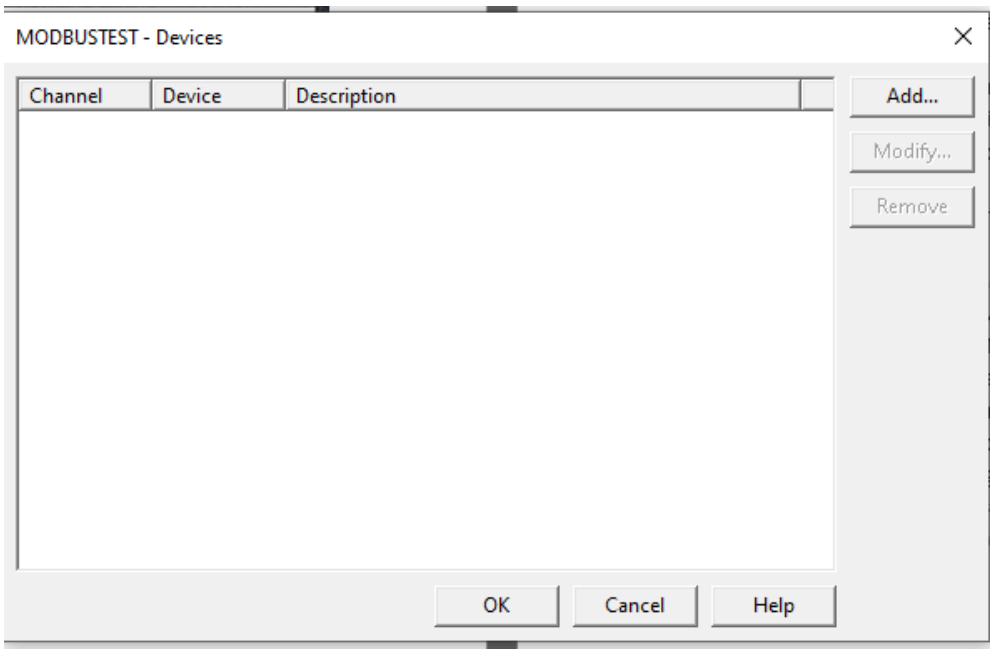
Set the COMPORT, Parity, Stop Bits and Data Bits



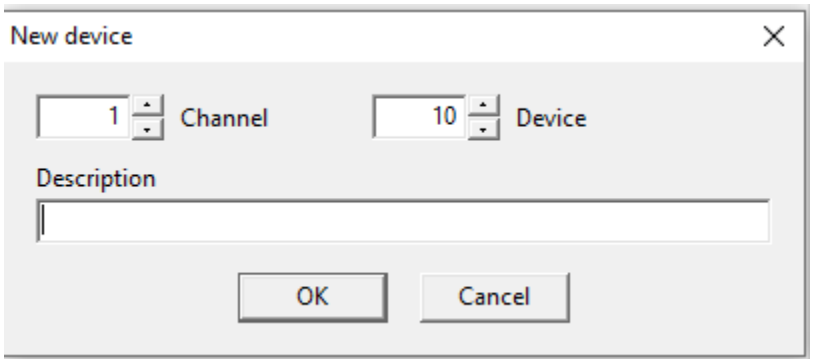
Press OK on the MODBUS RTU Configuration Pop – up

Press OK on the Channels Configuration this will bring you back to the Project Manager Window

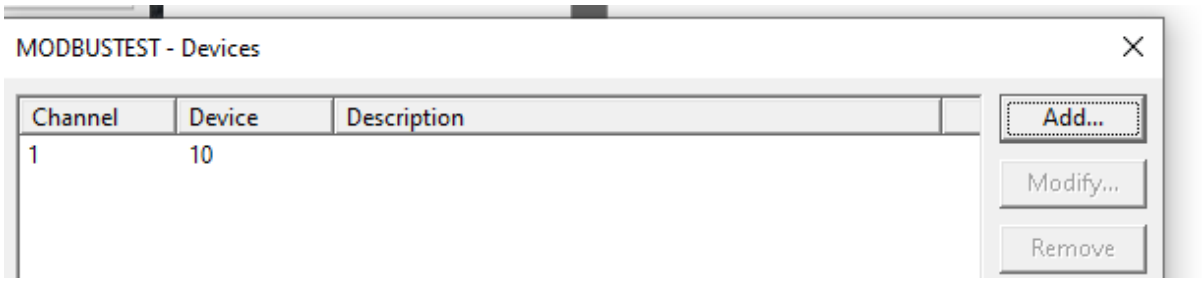
Proceed to double click on the device.



Click on the Add Button and set your Slave Device ID here

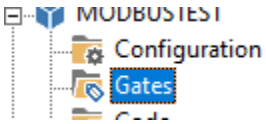


You can be able to verify the configuration settings will be shown to you after you confirm the new device.

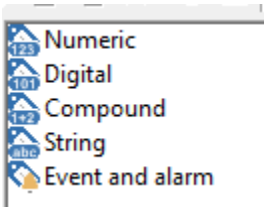


Press OK here this brings you back to the project manager.

Proceed to click on the Gates folder inside the project your editing.



This will show you the following data types



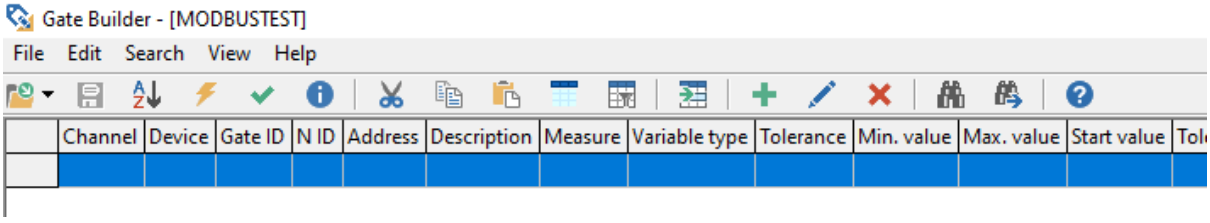
Gates are also known as tags in other general language used in PLC

At this point open up your program or au16data table to know what is the nature of the inputs for this example its actually

Table 7. au16data Containers

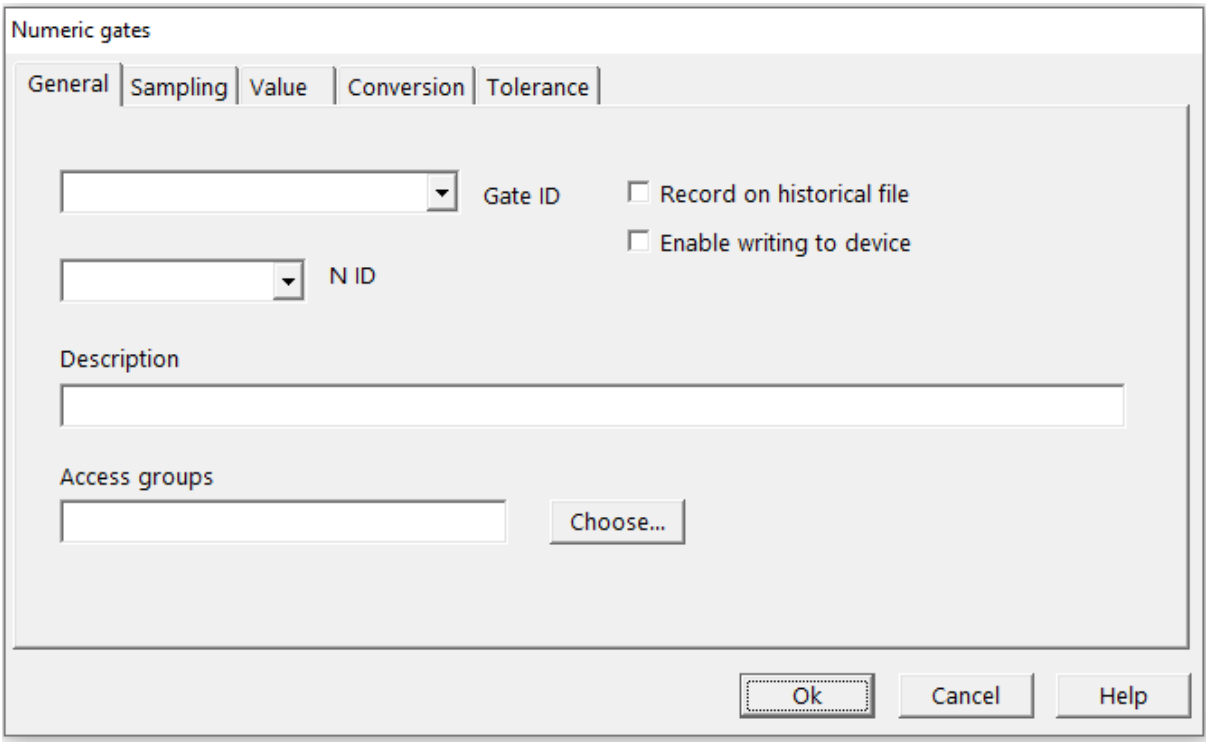
0	1	2	3	4	5	6	7	8
Prime	Command	Value	referencetime	count	Elapsed time		Motor On / Off	SML Value

Click on the Numeric this should open up the Gate Builder Screen



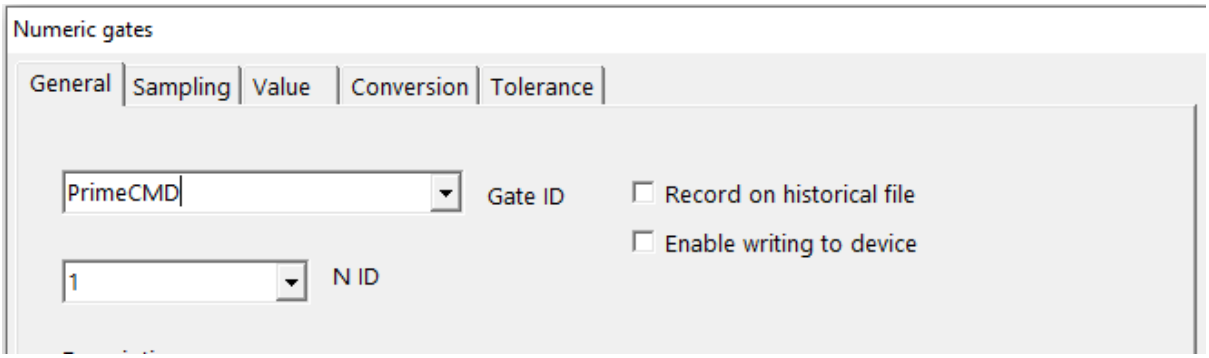
Press the green plus sign to open listing of new tags.

Use your version of Table 7 to create the table.

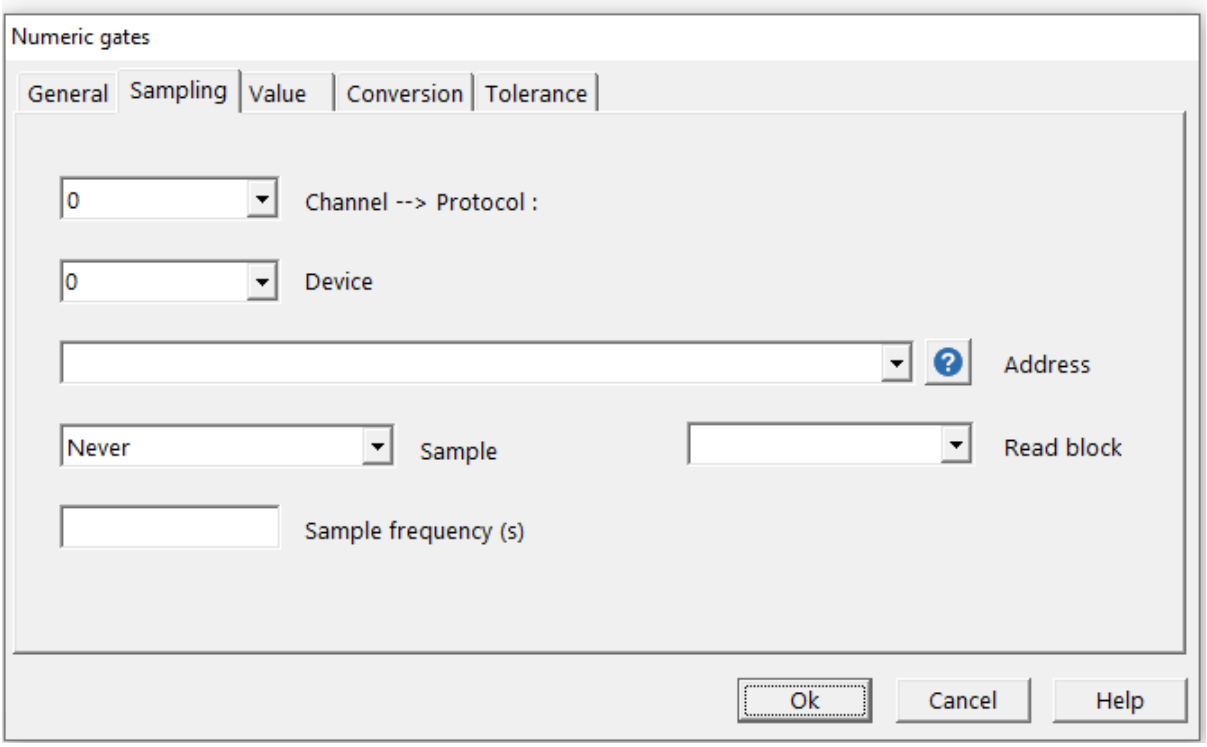


Add in the Gate ID this is a text value at assigns a more easier to remember name for the type of data.

Add in the N ID which is just a number that denotes number in a series of similar numerical tags there are you should start at 1.

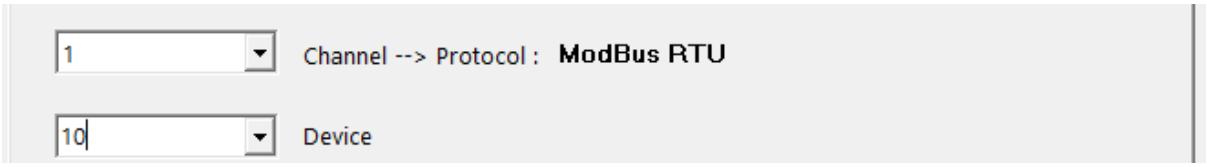


Click on the Sampling Tab these open configuration items should be shown.



Select Channel you configured previously

Select the device ID number of your Arduino.



You will notice upon selecting 1 Ch the name of the protocol is shown associated with this channel.

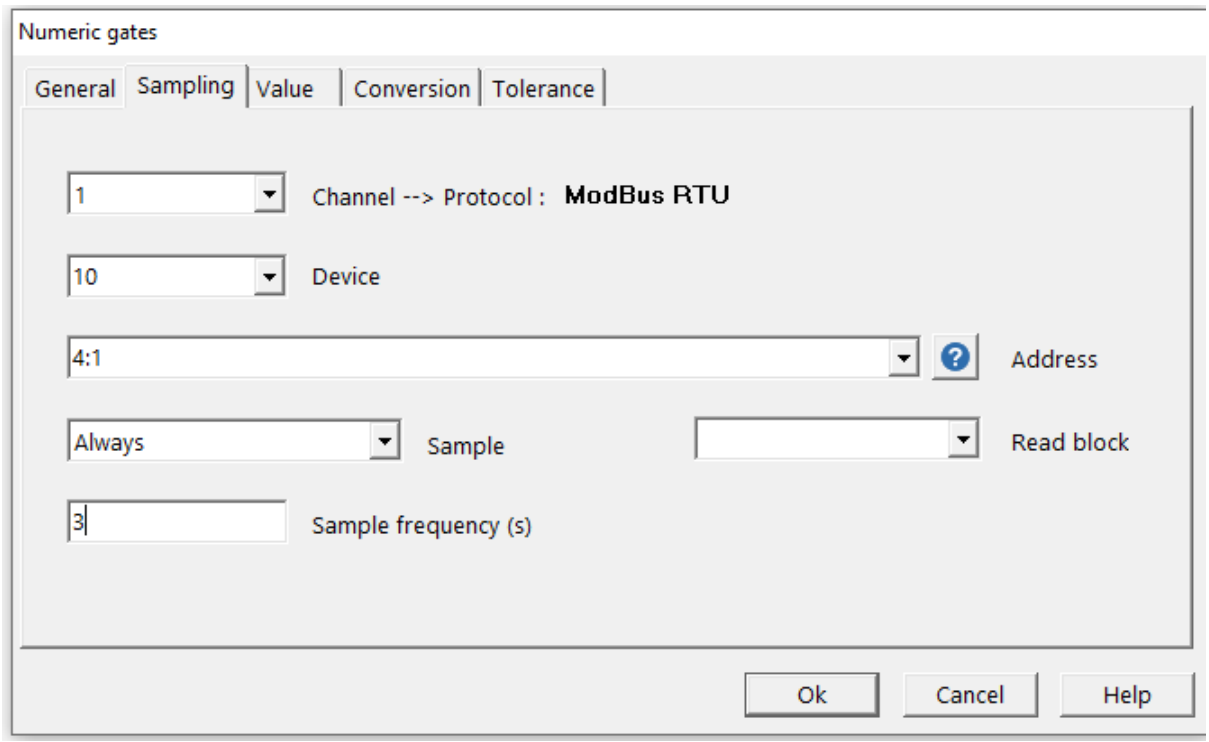
The address is bound by a prefix number system table 8 shows it.

Table 8. Address Prefixes for Numerical Input and Holding Register

Decription	Function	Address	Gate read	Gate write	Block read
3 (obsolete)	HOLDING REGISTER 16 bit	XXXX (0...9999 decimal)	Yes	Yes	Yes
3:	HOLDING REGISTER 16 bit	XXXXX (0...65535 decimal)	Yes	Yes	Yes
3h:	HOLDING REGISTER 16 bit	XXXXh (0...FFFF Hexadecimal)	Yes	Yes	Yes
3:16:	HOLDING REGISTER 16 bit	XXXXX (0...65535 decimal)	Yes	Yes	Yes
3h:10h:	HOLDING REGISTER 16 bit	XXXXh (0...FFFF Hexadecimal)	Yes	Yes	Yes
4 (obsolete)	INPUT REGISTER 16 bit	XXXX (0...9999 decimal)	Yes	No	Yes
4:	INPUT REGISTER 16 bit	XXXXX (0...65535 decimal)	Yes	No	Yes
4h:	INPUT REGISTER 16 bit	XXXXh (0...FFFF Hexadecimal)	Yes	No	Yes

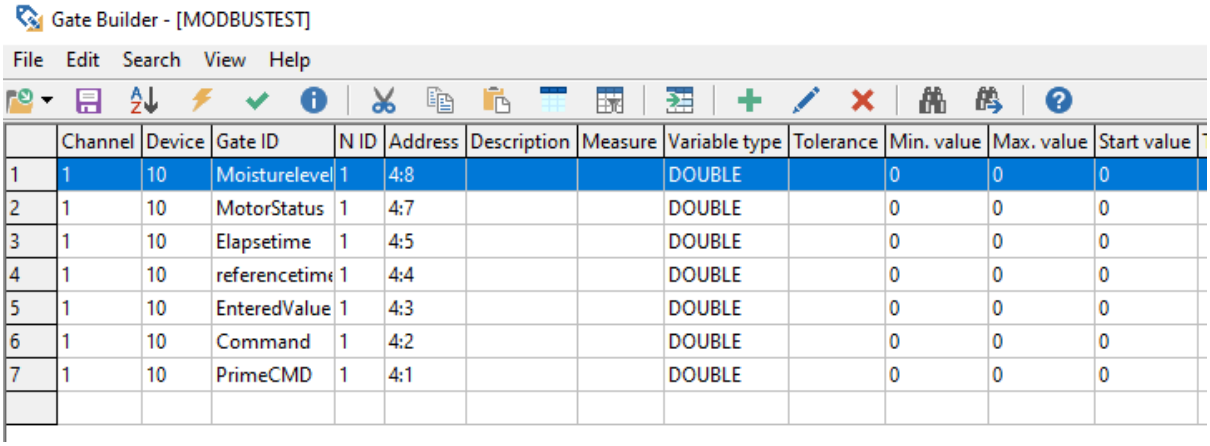
Type in 4: and followed by the address using table 7 add one to this since Winlog lite is base 1 software configuration not like Arduino which is base 0.

Let the program sample the register with the setting Always



The sample frequency is up to you to determine and press ok

Populate the Gate Builder with more numeric tags based on Table 7.



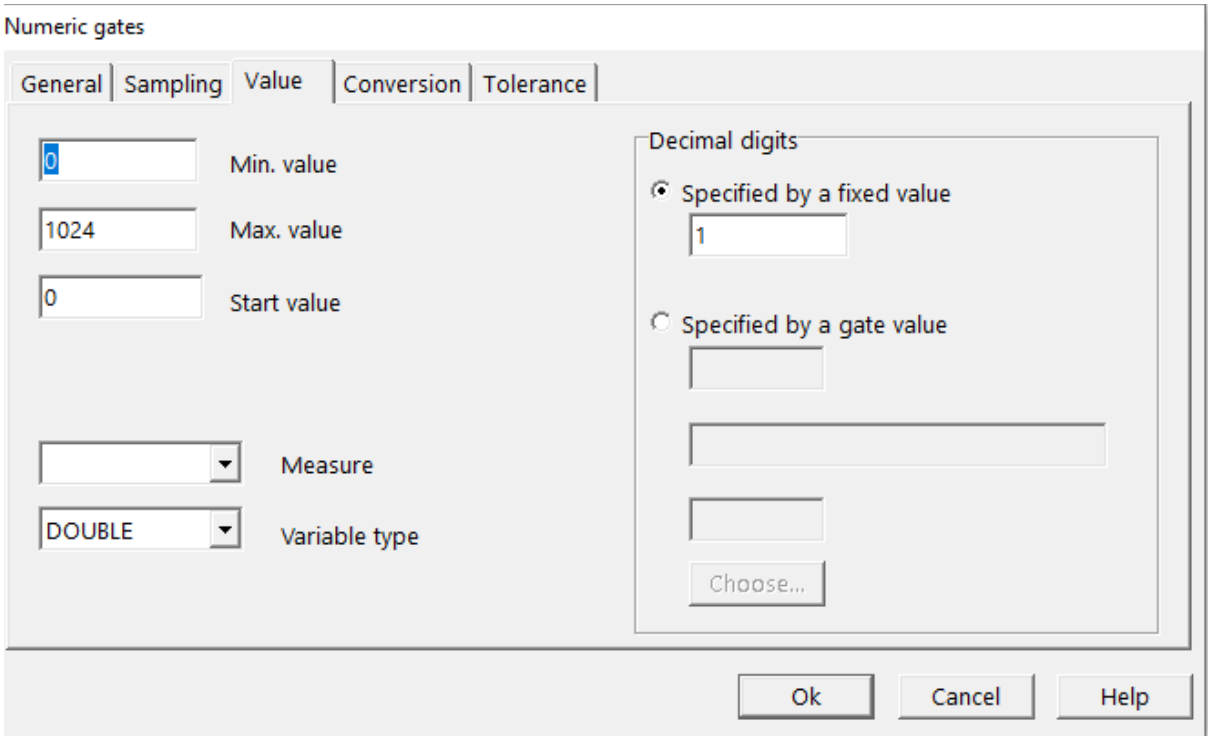
	Channel	Device	Gate ID	N ID	Address	Description	Measure	Variable type	Tolerance	Min. value	Max. value	Start value
1	1	10	Moisturelevel	1	4:8			DOUBLE		0	0	0
2	1	10	MotorStatus	1	4:7			DOUBLE		0	0	0
3	1	10	Elapsetime	1	4:5			DOUBLE		0	0	0
4	1	10	referencetime	1	4:4			DOUBLE		0	0	0
5	1	10	EnteredValue	1	4:3			DOUBLE		0	0	0
6	1	10	Command	1	4:2			DOUBLE		0	0	0
7	1	10	PrimeCMD	1	4:1			DOUBLE		0	0	0

Now we need to edit some values like the Maximum Values

Double Click the tag you want to edit

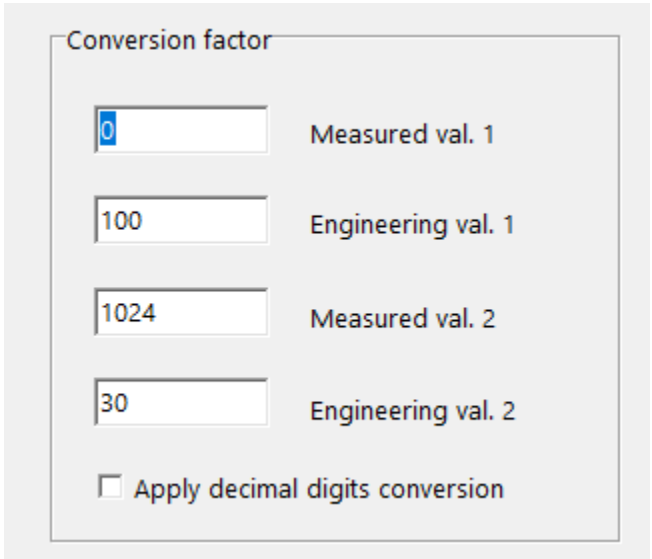
Click on the value tab

Write the minimum and maximum data values

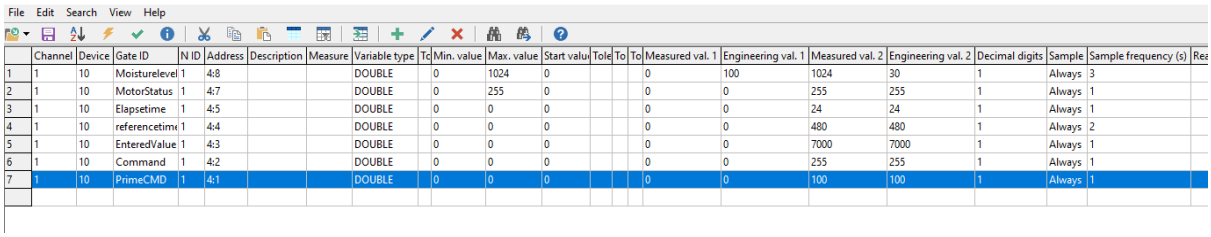


Click on the Conversion Tab. Use this feature if you want to map out values using the raw data that your machine provides.

For the Soil Moisture Meter this is



This once you edited all of the values you should end up with something like this.

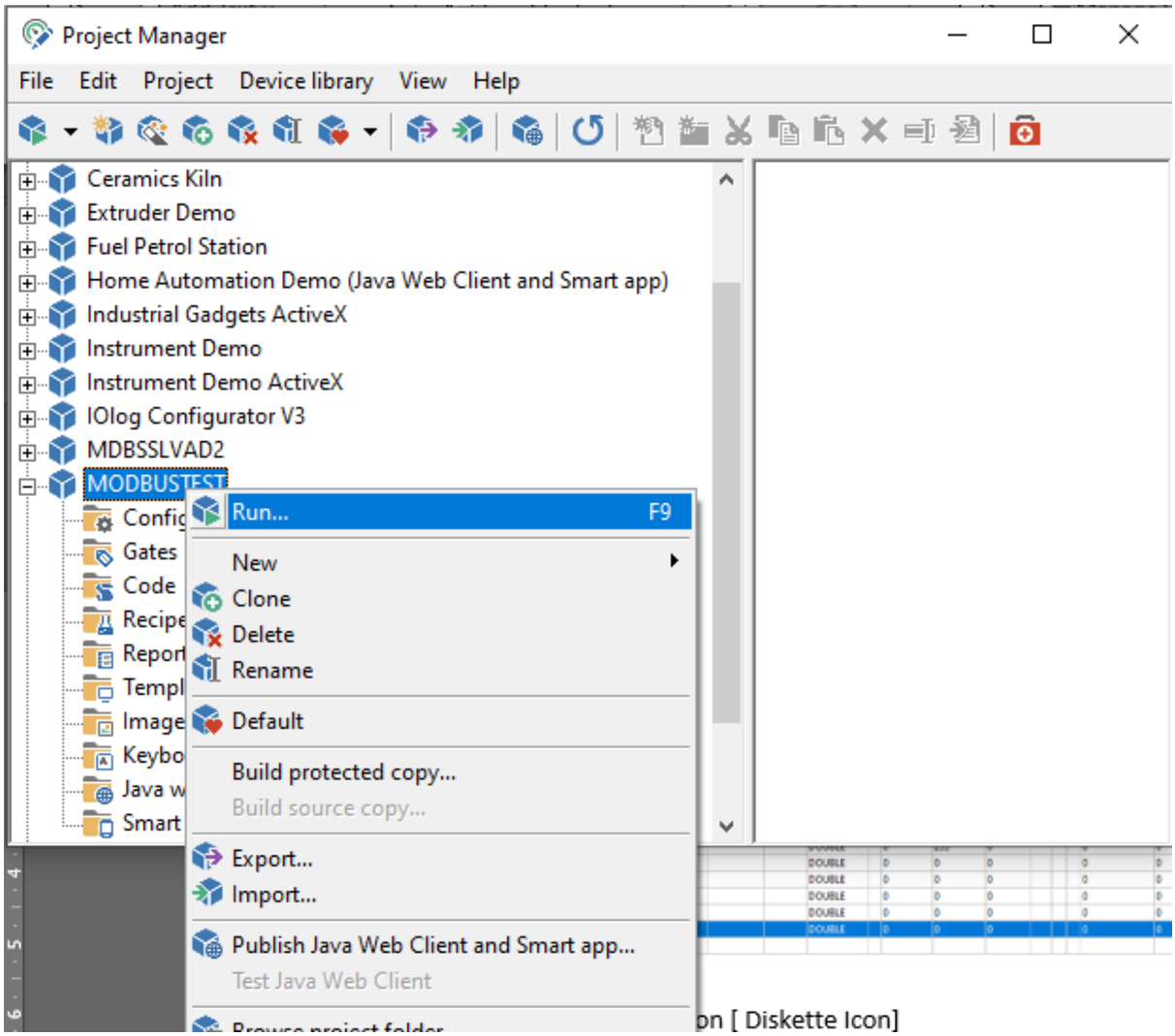


	Channel	Device	Gate ID	N ID	Address	Description	Measure	Variable type	Tc	Min. value	Max. value	Start valu	Tole	To	To	Measured val. 1	Engineering val. 1	Measured val. 2	Engineering val. 2	Decimal digits	Sample	Sample frequency (s)	Rea
1	1	10	Moisturelevel	1	4:8			DOUBLE		0	1024	0				0	100	1024	30	1	Always	3	
2	1	10	MotorStatus	1	4:7			DOUBLE		0	255	0				0	0	255	255	1	Always	1	
3	1	10	Elapsetime	1	4:5			DOUBLE		0	0	0				0	0	24	24	1	Always	1	
4	1	10	referencetime	1	4:4			DOUBLE		0	0	0				0	0	480	480	1	Always	2	
5	1	10	EnteredValue	1	4:3			DOUBLE		0	0	0				0	0	7000	7000	1	Always	1	
6	1	10	Command	1	4:2			DOUBLE		0	0	0				0	0	255	255	1	Always	1	
7	1	10	PrimeCMD	1	4:1			DOUBLE		0	0	0				0	0	100	100	1	Always	1	

Press the Save Button [Diskette Icon]

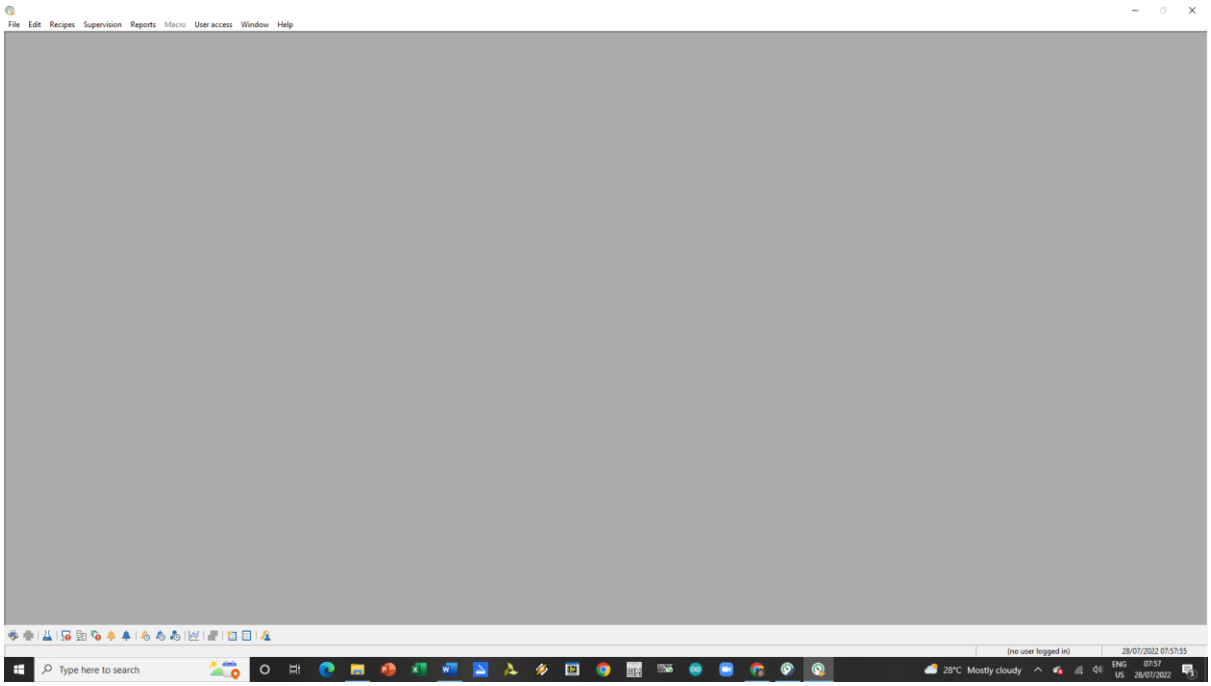
Leave the Gate Builder Window.

Test your interface first by running the software using the Run command.

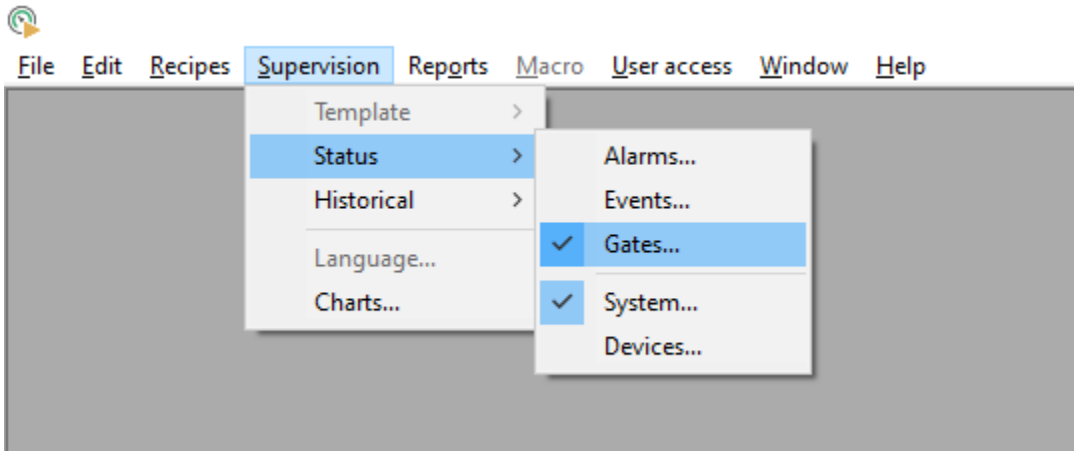


on [Diskette Icon]

Once it starts to run the screen will change to this.

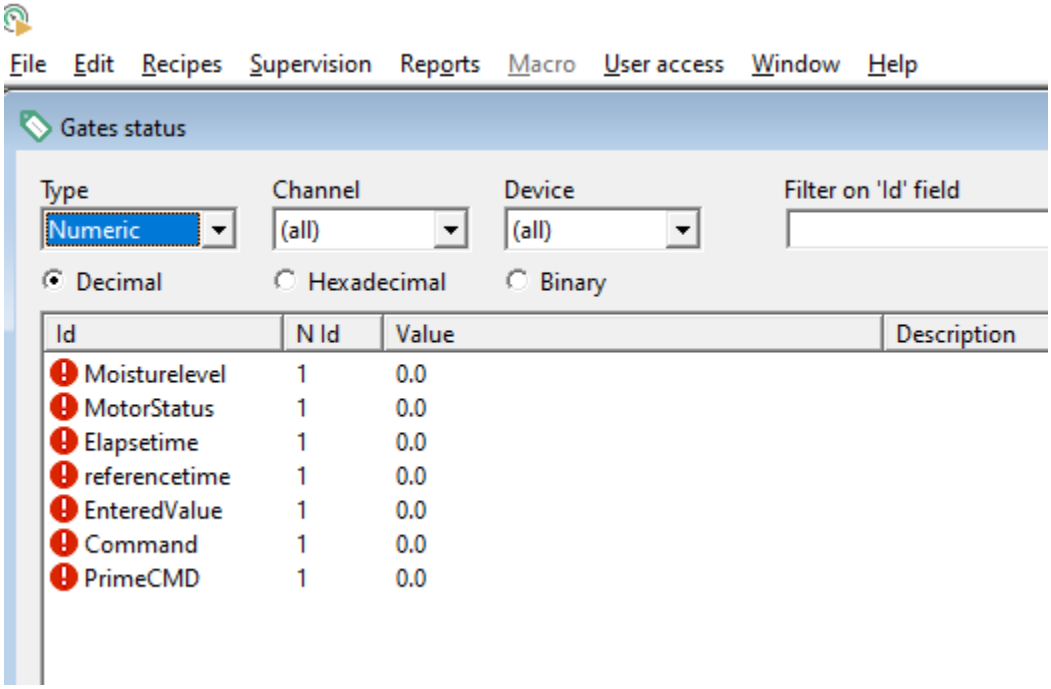


Simply use the Supervision Tab to access the gates



A new window will come out to show the input status

Select Numeric under the Type of Gate



If there is a problem in terms of connection this should show up. You can see its all red.

Project 2. Build your assigned project

1. Draw Lots on which group does what particular project.
2. Write down the description of what your project does.
3. Create a Block Diagram based on the Description.
4. Create an Input-Process-Output Table to define your device I/O needs and the internal process.
5. Create a proposed Wiring Diagram of how you intended to connect your modules.
6. Create the schematic diagram after you finalize your device wiring. Make sure to update the wiring diagram and block diagram for any additional devices that you may need to do your implementation.
7. Create the Bill of Materials for your project.
8. Create and finalize any mechanical designs your machine needs to operate. This should be on a separate design sheet under mechanism designs. It should have its own discussion on how it operates in line with the machine you want to build.
9. Work on the Arduino MODBUS, EEPROM and LCD program you need to run your project.
10. Work on a User HMI that can show some details not available on the LCD display about your device condition like remaining charges, hours used, cumulative tally and other stuff that you can see as a technician for the device and not an end user.
11. Test your Circuitry using actual people.
12. Test your HMI on the Hardware Unit you used. Let Students grade the ease of use and the ability to understand the data presented on your HMI thru an after use survey.