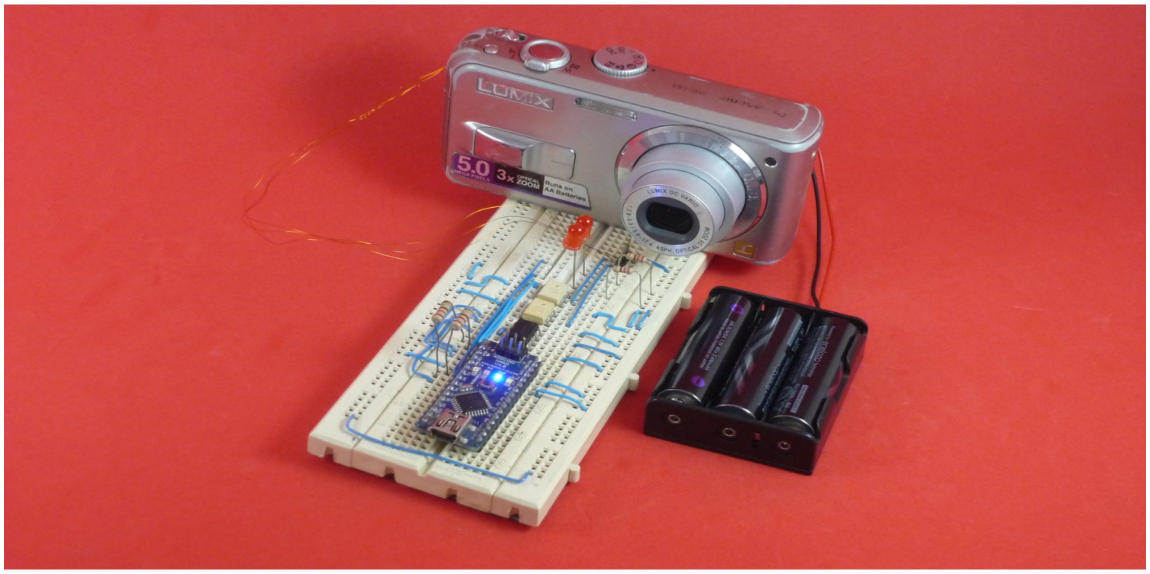
ARDUINO CONTROLLED
DIGITAL CAMERA
VIA MODBUS RTU



REMOTE CONTROLLED DIGITAL CAMERA

It's very likely that a lot of people have a digital camera in a relatively good shape, getting dust or hidden in some drawer because actual smartphones have cameras with better resolution. Those abandoned cameras could get a second life as trap cams adding some circuit with a PIR sensor, time lapse cams, or a camera to take automated pictures in a small production chain, etc.

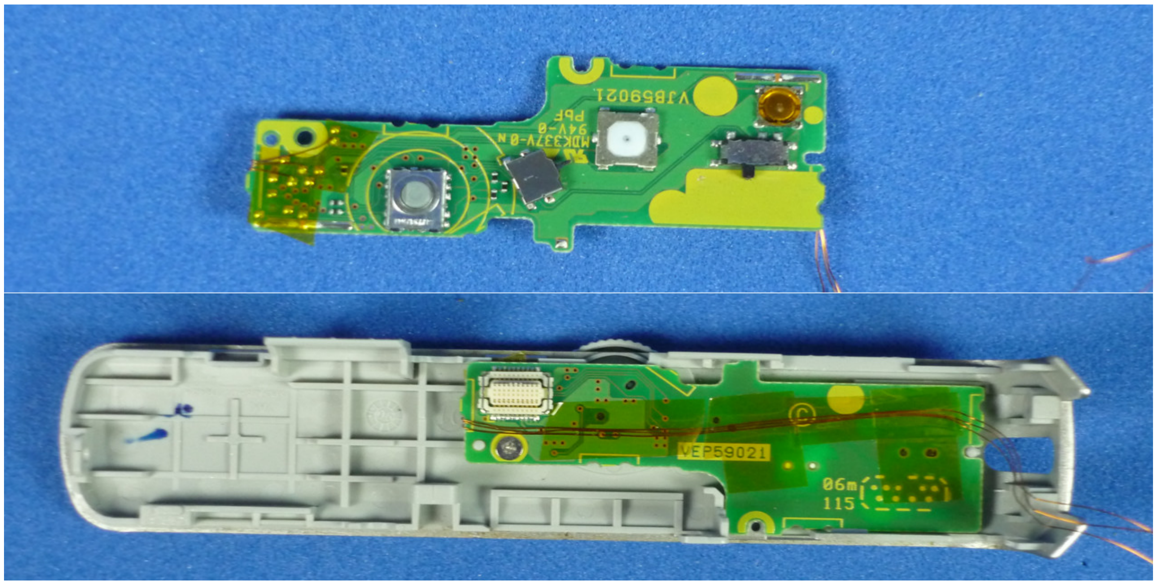
There are many ways to adapt those cameras to other purposes without much trouble, if the cam has the option for an external shutter switch, or if it is one of the lucky ones that supports CHDK (Canon Hack Development Kit)

If none of the aforementioned options are available for a particular digital camera, there is a "less elegant" alternative soldering wires to the push-buttons that triggers some functions in the cam, like focus, shutter, zoom, menu, etc.

The process requires a lot of patience, because the camera needs to be disassembled (as possible using a kind of plastic tool used to open cell phones to avoid cracks in the soft parts) to reach the push-buttons and solder very thin wires to their pads. There is a mystery than needs to be solved with every cam: That particular switch is active low or active high?. To overcome this a little bit of knowledge/experience is needed if there is no repair/service manual of the camera in sight.

In this particular case, extension wires have been soldered to the focus and shutter test pads on the PCB, if there are no test pads, wires have to be soldered directly to the push-button pins.

Because the signals are active-low, a wire to a GND pad was soldered. A little bit of Kapton tape was used to protect the thin wires.



ARDUINO AND MODBUS RTU PROTOCOL

ModBus protocol is widely used in industrial automation hardware like PLC, HMI, VFD, etc. Usually over twisted pair using RS-485 standard in distances up to 1200 meters. Because is an open protocol, well documented and easy to implement in many languages/platforms (Java, Python, Arduino, PIC, etc.) is an ideal candidate for the remote control circuit.

The goal is to use an Arduino Nano to receive commands (focus, take pictures, etc.) and execute the respective push button actions electrically simulating manual operation. Optocouplers were used like in another project to optically isolate the camera from the remote control circuitry

In this example, a Panasonic Lumix DMC-LS3 camera, has 2 position push-button. Pressing to half position, the camera focuses, and when pressed full, camera takes a picture. The push-buttons should keep hold for some amount of time, because if "pressed" too fast, the debouncer algorithm in the camera will reject the action. This delay time should be found experimentally because debounce times vary from camera to camera.

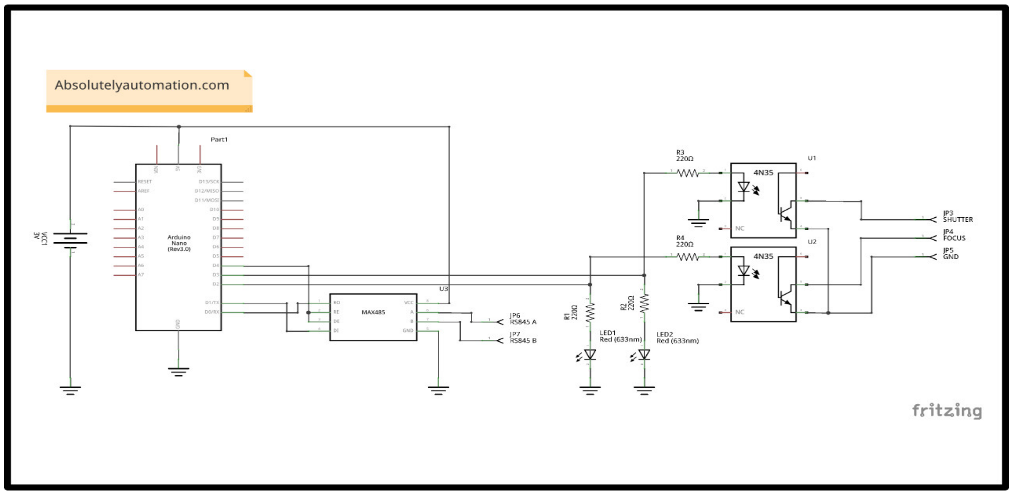
The following commands were implemented:

1. Press and hold focus.
2. Release focus
3. Press and hold focus, wait for some time, press and hold shutter, wait for some time and finally release focus and shutter.

The first and second commands are very useful to keep the camera powered on, because if the time between taking pictures is very long some cameras automatically power off to save energy. In some cameras the auto power off feature could be disabled or it's time extended, but in others isn't possible. The commands could be used to focus the camera periodically as a sort of "ping" to keep the camera on.

The third command takes a picture and emulate the human push-button sequence process.

For the execution of each command, a non-zero value should be written to their respective ModBus register
.

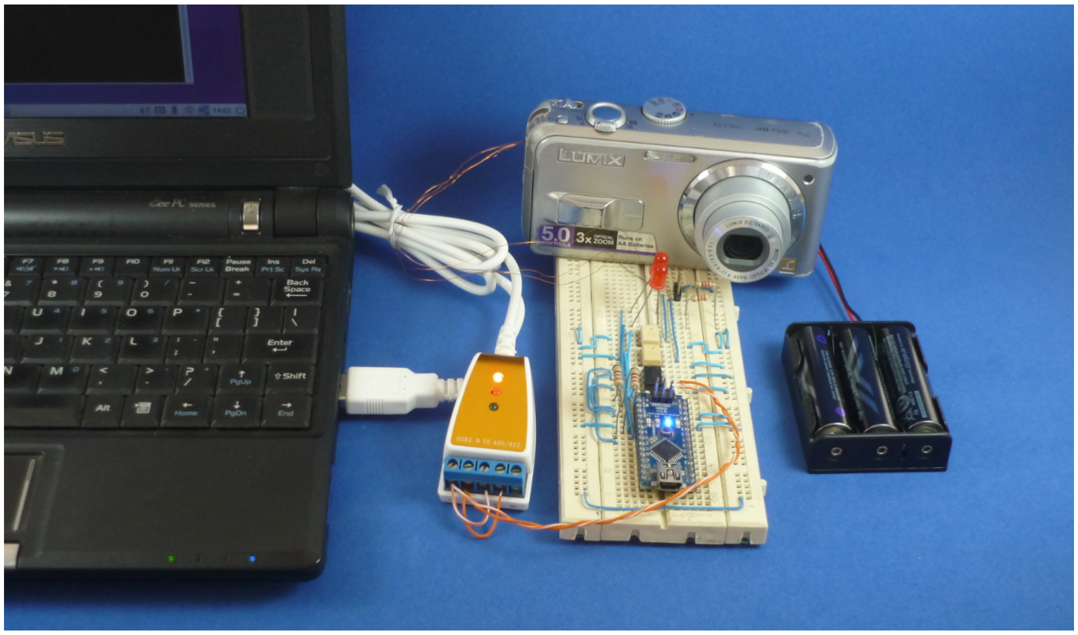


PC FIRST TEST

For this test a personal computer (PC) with a ModBus RTU Python client app were used. A USB to RS-485 converter is required to connect the Arduino to the computer. This combination makes the Modbus RTU network master.

The Arduino remote controlled digital camera circuit, and the converter were wired with a cheap twisted pair. This makes the ModBus RTU network slave

A command to write a non-zero value to holding register 10 was sent. This register triggers the picture taking command sequence. Note the red leds used to debug the delay hold time for focus and shutter buttons.



TESTING IN AN RS485 NETWORK WITH A PLC AND A RASPBERRY PI

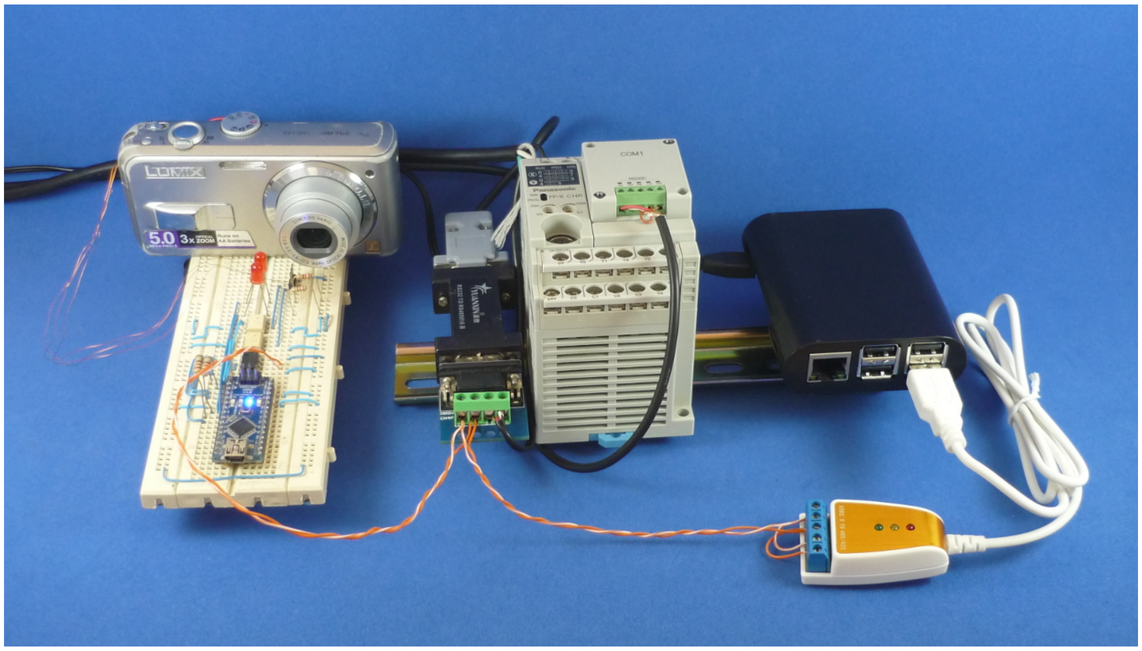
To simulate a more complex ModBus RTU network, some industrial equipment was added. The setup is as follows:

- Raspberry Pi, with a USB to RS-485 converter will be the network master
- Panasonic FPX-C14 PLC with an RS-232 to RS-485 will be a network slave
- Arduino Nano with the camera will be a network slave

In this test the master reads a value from a register in the PLC, if this value is greater than some fixed number, the command to take a picture will be sent to the Arduino Nano. The PLC was programmed in a way that the register value can be modified with an external potentiometer built into the PLC.

ModBus RTU command logic running inside Raspberry Pi, was implemented this time, with Node-RED. A ModBus RTU didactic flow was written and only requires serial port nodes and no other external library or dependency. It wasn't written in the most efficient way, but is very didactic, because all the steps in the request packet frame building are easily spotted. Also, it is very easy to add new ModBus functions.

Finally to give it a more "industrial appeal", the Raspberry Pi was placed in an enclosure with added plastic clips for DIN rail mount.



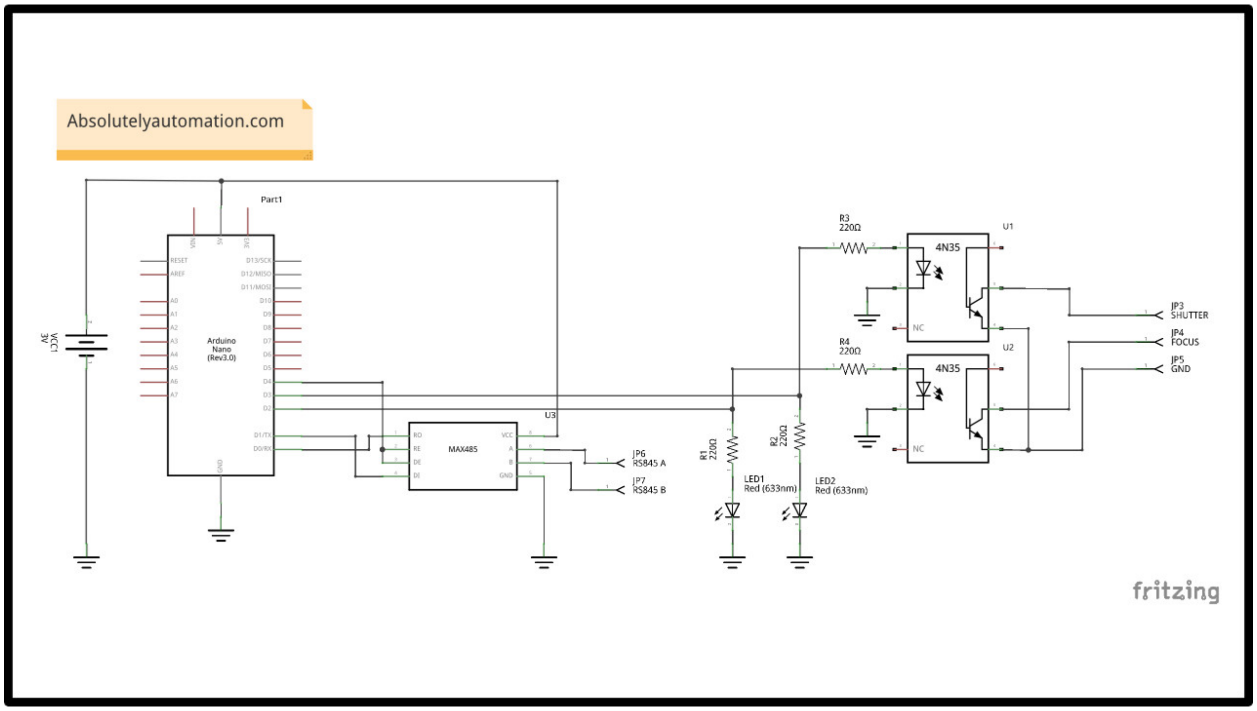
TESTS AND CONCLUSIONS

- An initial test was done with a Celeron 900 Mhz, 2B RAM PC, Lubuntu 14.04, Node-Red v0.15.2 and Node.js v0.10.25. Node-RED cyclic trigger was configured to fire reads from the PLC and writes to the Arduino every 15 seconds. No noticeable delay was observed.
- A second test was made with a Raspberry Pi (1 B+) Broadcom 700MHz, 512MB RAM, Raspbian Jessi Lite 4.4, Node-Red v0.15.2 and Node.js v0.10.29. A delay was observed (about 10 seconds +) to the 15 configured for the reading cycles. There are no big gaps between versions of Node-Red and Node.js, but there are a big difference in the hardware of the two tests and probably is the cause of the "slowness". Remember again the flow wasn't written in an optimal way!.
- In every of the two tests the results were the same: A picture is only taken if the value read from the register in the PLC is below the fixed number programmed in Node-RED logic.
- ModBus RTU implementation for Arduino works pretty well, so it can be used as a node in an RS-485 network with other industrial equipment.
- As a future upgrade, the circuit could be miniaturized to place it inside the camera.
- Explore low power options for the microcontroller, so it can be powered by the camera batteries.
- Node-RED is a very useful tool to develop communication protocols in a graphic and quick manner.

LINKS

Video that shows all the components and working tests : **youtu.be/b42Js7DofjM**
Arduino Nano Clone: **http://s.click.aliexpress.com/e/FaYvBuf**
Tool for opening plastic cases: **http://s.click.aliexpress.com/e/RfAEqnM**
USB to RS-485 converter: **http://s.click.aliexpress.com/e/EUFiuRF**
Clips for PCB DIN rail mount: **http://s.click.aliexpress.com/e/AIIuBIm**

SCHEMATIC FOR THE ARDUINO CONTROLLEDDIGITAL CAMERA VIA MODBUS RTU CIRCUIT



ARDUINO MODBUS RTU SOURCE CODE

```
// ***************************************************
// Absolutelyautomation.com
// ***************************************************

#include <SimpleModbusSlave.h>

#define  ledPin  13         // onboard led
#define  halfshutterPin  2  // connected to the optocoupler's led that control
half shutter ( focus )
#define  fullshutterPin  3  // connected to the optocoupler's led that control
full shutter ( take picture )
#define  buttonPin  7       // push button

#define  delayhalf  2500 // The value depens of the type of camera, please
make your own experiments
#define  delayfull  500  // The value depens of the type of camera, please
make your own experiments


/* This example code has 9 holding registers. 6 analogue inputs, 1 button, 1
digital output
   and 1 register to indicate errors encountered since started.
   Function 5 (write single coil) is not implemented so I'm using a whole
register
   and function 16 to set the onboard Led on the Atmega328P.

   The modbus_update() method updates the holdingRegs register array and
checks communication.

   Note:
   The Arduino serial ring buffer is 128 bytes or 64 registers.
   Most of the time you will connect the arduino to a master via serial
   using a MAX485 or similar.

   In a function 3 request the master will attempt to read from your
   slave and since 5 bytes is already used for ID, FUNCTION, NO OF BYTES
   and two BYTES CRC the master can only request 122 bytes or 61 registers.

   In a function 16 request the master will attempt to write to your
   slave and since a 9 bytes is already used for ID, FUNCTION, ADDRESS,
   NO OF REGISTERS, NO OF BYTES and two BYTES CRC the master can only write
   118 bytes or 59 registers.

   Using the FTDI USB to Serial converter the maximum bytes you can send is
limited
   to its internal buffer which is 60 bytes or 30 unsigned int registers.

   Thus:

   In a function 3 request the master will attempt to read from your
   slave and since 5 bytes is already used for ID, FUNCTION, NO OF BYTES
   and two BYTES CRC the master can only request 54 bytes or 27 registers.
   In a function 16 request the master will attempt to write to your
   slave and since a 9 bytes is already used for ID, FUNCTION, ADDRESS,
   NO OF REGISTERS, NO OF BYTES and two BYTES CRC the master can only write
   50 bytes or 25 registers.

   Since it is assumed that you will mostly use the Arduino to connect to a
   master without using a USB to Serial converter the internal buffer is set
   the same as the Arduino Serial ring buffer which is 128 bytes.
*/


// Using the enum instruction allows for an easy method for adding and
// removing registers. Doing it this way saves you #defining the size
// of your slaves register array each time you want to add more registers
// and at a glimpse informs you of your slaves register layout.

/////////////// registers of your slave ///////////////////
enum
{
  // just add or remove registers and your good to go...
  // The first register starts at address 0
  ADC0,
  ADC1,
  ADC2,
  ADC3,
  ADC4,
  ADC5,
  LED_STATE,
  BUTTON_STATE,
  HALF_SHUTTER_HOLD,
  HALF_SHUTTER_RELEASE,
  HALFFULL_SHUTTER_CLICK,
  TOTAL_ERRORS,
  // leave this one
  TOTAL_REGS_SIZE
  // total number of registers for function 3 and 16 share the same register
array
};

unsigned int holdingRegs[TOTAL_REGS_SIZE]; // function 3 and 16 register array
boolean halftoggled;
boolean fulltoggled;
unsigned long timehalf;
unsigned long timefull;

////////////////////////////////////////////////////////////

void setup()
{
  /* parameters(long baudrate,
                unsigned char ID,
                unsigned char transmit enable pin,
                unsigned int holding registers size,
                unsigned char low latency)

     The transmit enable pin is used in half duplex communication to activate
a MAX485 or similar
     to deactivate this mode use any value < 2 because 0 & 1 is reserved for
Rx & Tx.
     Low latency delays makes the implementation non-standard
     but practically it works with all major modbus master implementations.
  */

  modbus_configure(9600, 1, 4, TOTAL_REGS_SIZE, 0);
  pinMode(ledPin, OUTPUT);
  pinMode(halfshutterPin, OUTPUT);
  pinMode(fullshutterPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop()
{

  // modbus_update() is the only method used in loop(). It returns the total
error
  // count since the slave started. You don't have to use it but it's useful
  // for fault finding by the modbus master.
  holdingRegs[TOTAL_ERRORS] = modbus_update(holdingRegs);
  for (byte i = 0; i < 6; i++)
  {
    holdingRegs[i] = analogRead(i);
    delayMicroseconds(50);
  }


  // If a value > 0 is written in the respective register, a command will be
executed
  // Add more pins, registers to get more commands: zoom in/out, camera menus,
etc


  if(holdingRegs[HALF_SHUTTER_HOLD] > 0)
  {
    digitalWrite(halfshutterPin, HIGH);
    digitalWrite(ledPin, HIGH);
  }

  if(holdingRegs[HALF_SHUTTER_RELEASE] > 0)
  {
    digitalWrite(halfshutterPin, LOW);
    holdingRegs[HALF_SHUTTER_HOLD]=0;
    holdingRegs[HALF_SHUTTER_RELEASE]=0;
   digitalWrite(ledPin, LOW);
  }

  if(holdingRegs[HALFFULL_SHUTTER_CLICK] > 0 && halftoggled==0 )
  {
    digitalWrite(halfshutterPin, HIGH);
    halftoggled=1;
    timehalf = millis();
    digitalWrite(ledPin, HIGH);
  }


  if(halftoggled==1 && fulltoggled==0)
  {
    if (millis() > timehalf + delayhalf)
     {
       digitalWrite(fullshutterPin, HIGH);
       fulltoggled=1;
       timefull = millis();
     }
  }

  if( fulltoggled==1 )
  {
    if (millis() > timefull + delayfull)
     {
       digitalWrite(halfshutterPin, LOW);
       digitalWrite(fullshutterPin, LOW);

       halftoggled=0;
       fulltoggled=0;
       holdingRegs[HALFFULL_SHUTTER_CLICK] = 0;

       digitalWrite(ledPin, LOW);

     }
  }


}

// ****************************************************
// Absolutelyautomation.com
// ****************************************************
```

MODBUS RTU CONTROL LOGIC IN NODE-RED

```
[{"id":"d2d74c5a.3d845","type":"tab","label":"Flow 1"},
{"id":"804fb389.d5c04","type":"serial
out","z":"d2d74c5a.3d845","name":"SERIAL","serial":"e73c106.4acf0f","x":1706.9
640407562256,"y":919.6626811027527,"wires":[]},
{"id":"6553dc1b.45db14","type":"serial
in","z":"d2d74c5a.3d845","name":"","serial":"e73c106.4acf0f","x":483.503959655
7617,"y":1605.5829491615295,"wires":[["f6094ac1.9de378","e044f03a.cbf64"]]},
{"id":"d5360e96.f942a","type":"switch","z":"d2d74c5a.3d845","name":"F
switch","property":"payload.function","propertyType":"msg","rules":
[{"t":"eq","v":"0x01","vt":"num"},{"t":"eq","v":"0x02","vt":"num"},
{"t":"eq","v":"0x03","vt":"num"},{"t":"eq","v":"0x04","vt":"num"},
{"t":"eq","v":"0x05","vt":"num"},{"t":"eq","v":"0x06","vt":"num"},
{"t":"eq","v":"0x0F","vt":"num"},{"t":"eq","v":"0x10","vt":"num"},{"t":"else"}
],"checkall":"true","outputs":9,"x":775.6306838989258,"y":920.9960017204285,"w
ires":[["49d1563a.f68028"],["1e59f50a.c55d6b"],["e6f31c3f.866b8"],
["4bdac700.13dd98"],["48f45da2.f849f4"],["788402de.59745c"],
["7e5d4808.b4a018"],["395e4073.c184b"],["52335107.fdfb2"]]},
{"id":"952acace.7cb528","type":"function","z":"d2d74c5a.3d845","name":"RTU CRC
CHECK","func":"// Absolutelyautomation.com\n//\nvar len =
msg.payload.rxpacket.length;\nvar dataStr = msg.payload.rxpacket;\nvar crc =
0xFFFF;\n\nfor ( var pos = 0; pos < (len-2) ; pos++ ) \n{\n  crc = crc ^
dataStr[pos]; \n  \n  for (var i = 8; i !== 0; i--) \n  {   // Loop over each
bit\n      \n      if ((crc & 0x0001) !== 0) \n      { // If the LSB is set\n
crc = crc >> 1;                    // Shift right and XOR 0xA001\n        crc
= crc ^ 0xA001;\n      }\n      else                                // Else LSB is
not set\n        crc = crc >> 1;                    // Just shift right\n    }
\n  \n}\n\n// Note, this number has low and high bytes swapped, so use it
accordingly (or swap bytes)\n\nvar crclow = crc >> 8;\nvar crchi = crc &
0x00FF;\n\nif ( crchi === dataStr[len-2] && crclow === dataStr[len-1] ) \n
{ \n    //flow.set('CRCrcv',0x01);\n    msg.payload.okcrc = 0x01;\n}\nelse
\n    //flow.set('CRCrcv',0x00);\n    msg.payload.okcrc = 0x00;\n\n\nreturn
msg;
\n","outputs":1,"noerr":0,"x":1953.186538696289,"y":1617.8526859283447,"wires"
:[["da07586e.3dac18"]]},
{"id":"d4e43630.a11ee8","type":"switch","z":"d2d74c5a.3d845","name":"Min
packet
size?","property":"payload.rxpacket.length","propertyType":"msg","rules":
[{"t":"gt","v":"3","vt":"num"},{"t":"lt","v":"4","vt":"num"}
],"checkall":"true","outputs":2,"x":1744.2976989746094,"y":1624.9957427978516,
"wires":[["952acace.7cb528"],["737ab9ed.9e6698"]]},
{"id":"737ab9ed.9e6698","type":"function","z":"d2d74c5a.3d845","name":"Packet
too short error","func":"// Absolutelyautomation.com\n//\nnode.error(\"Packet
too short Error\",
msg);","outputs":1,"noerr":0,"x":1964.6150283813477,"y":1666.1067037582397,"wi
res":[[]]},
{"id":"da07586e.3dac18","type":"switch","z":"d2d74c5a.3d845","name":"OK
CRC?","property":"payload.okcrc","propertyType":"msg","rules":[{"t":"eq","v":"0","vt":"num
F15 slave mismatch\", msg);\n}\n// Function is the same?\nelse if (msg.payload.function !==
global.get('MBFunction'))\n{\n    node.status({fill:\"red\",shape:\"dot
\",text:\"error\"});    \n    node.error(\"F15 function mismatch\", msg);\n}
\nelse\n{\n    node.status({fill:\"green\",shape:\"dot\",text:\"sucess\"});
\n    return msg;\n}\n
```

\n","outputs":1,"noerr":0,"x":3010.7778329849243,"y":1770.468183517456,"wires"
:[[]]},
{"id":"278f9029.f248d","type":"function","z":"d2d74c5a.3d845","name":"F16
analyze packet","func":"// Absolutelyautomation.com\n//\n// F16: Preset
multiple registers\nvar regrx = (msg.payload.rxpacket[2] * 256) |
(msg.payload.rxpacket[3]);\nvar tmpreg = global.get('MBRegister'); \nvar qtytx
= global.get('MBQuantity');\nvar qtyrx = (msg.payload.rxpacket[4] * 256) |
(msg.payload.rxpacket[5]);\n\n// Register is the same?\nif(regrx != tmpreg-1
)\n{\n   node.status({fill:\"red\",shape:\"dot\",text:\"error\"});   \n
node.error(\"F16 Register mismatch\", msg);\n}\n// Qty is the same?\nelse if
(qtytx != qtyrx)\n{\n   node.status({fill:\"red\",shape:\"dot\",text:\"error
\"});   \n   node.error(\"F16 Qty mismatch\", msg);\n}\n// Slave is the
same?\nelse if(msg.payload.slave != global.get('MBSlaveId'))\n{\n
node.status({fill:\"red\",shape:\"dot\",text:\"error\"});   \n   node.error
(\"F16 slave mismatch\", msg);\n}\n// Function is the same?\nelse if
(msg.payload.function != global.get('MBFunction'))\n{\n   node.status({fill:
\"red\",shape:\"dot\",text:\"error\"});   \n   node.error(\"F16 function
mismatch\", msg);\n}\nelse\n{\n   node.status({fill:\"green\",shape:\"dot
\",text:\"sucess\"});   \n   return msg;\n}\n
\n","outputs":1,"noerr":0,"x":3007.7621240615845,"y":1832.531626701355,"wires"
:[[]]},
{"id":"e540204d.155af","type":"function","z":"d2d74c5a.3d845","name":"Unimplem
ented Function answer error","func":"// Absolutelyautomation.com\n//
\nnode.error(\"Unimplemented Function answer Error\",
msg);","outputs":1,"noerr":0,"x":3070.3332147598267,"y":1910.2460289001465,"wi
res":[[]]},
{"id":"72b640da.bea9b","type":"inject","z":"d2d74c5a.3d845","name":"Manual
single
test","topic":"","payload":"","payloadType":"date","repeat":"","crontab":"","o
nce":false,"x":170.69032287597656,"y":299.5555419921875,"wires":
[["11163099.4ffebf"]]},{"id":"9159661f.48b738","type":"link
out","z":"d2d74c5a.3d845","name":"","links":
["59823f74.3900f"],"x":828.0711803436279,"y":255.03174781799316,"wires":[]},
{"id":"59823f74.3900f","type":"link in","z":"d2d74c5a.3d845","name":"MB
Call","links":
["9159661f.48b738","905f9fac.b6233"],"x":221.42857551574707,"y":920.5317316055
298,"wires":[["894e09ca.c9d548"]]},
{"id":"b65cd759.d1efb8","type":"function","z":"d2d74c5a.3d845","name":"MB
Values from reading functions examples","func":"// Absolutelyautomation.com
\n//\n// ***************************************\n// Example F01: Read coil
status #000034 to 000040\n//   from slave 19\n\n//var arrtmp =
global.get('MBArray');\n//var coil34status = arrtmp[33];\n//var coil35status
= arrtmp[34];\n//var coil36status = arrtmp[35];\n//var coil37status = arrtmp
[36];\n//var coil38status = arrtmp[37];\n//var coil39status = arrtmp[38];\n//
var coil40status = arrtmp[39];\n\n// ***************************************
\n// Example F02: Read input status #100100 to 100101\n//   from
slave 28\n\n//var arrtmp = global.get('MBArray');\n//var input100status =
arrtmp[99];\n//var input101status = arrtmp[100];\n\n//
***************************************\n// Example F03: Read Holding
register #40000F\n//   from slave 3\n\nvar arrtmp = global.get
('MBArray');\nvar holding14value = arrtmp[14];\nglobal.set
('MBGlobalValue',holding14value);\n//console.log(holding14value);\n\n//
***************************************\n// Example F04: Read Input registers
#300006 to 300009\n//   from slave 30\n\n//var arrtmp = global.get

('MBArray');\n//var input6value = arrtmp[5];\n//var input7value = arrtmp[6];
\n//var input8value = arrtmp[7];\n//var input9value = arrtmp[8];\n\nreturn
msg;","outputs":1,"noerr":0,"x":1665.0238342285156,"y":325.0000123977661,"wire
s":[["605e32b0.23f23c"]]},
{"id":"eaa2d511.738688","type":"function","z":"d2d74c5a.3d845","name":"YOUR
ERROR HANDLER FUNCTION","func":"// Absolutelyautomation.com\n//\nreturn
msg;","outputs":1,"noerr":0,"x":1455.0237884521484,"y":254.99999284744263,"wir
es":[[]]},
{"id":"37214816.fbf668","type":"comment","z":"d2d74c5a.3d845","name":"Absolute
lyautomation.com","info":"Absolutelyautomation.com","x":173.02367401123047,"y"
:97.99999189376831,"wires":[]},
{"id":"f32a92cb.a9ed5","type":"inject","z":"d2d74c5a.3d845","name":"First time
auto
run","topic":"","payload":"","payloadType":"date","repeat":"","crontab":"","on
ce":true,"x":378.2500305175781,"y":769.1666884422302,"wires":
[["5feb322c.07030c"]]},
{"id":"11163099.4ffebf","type":"function","z":"d2d74c5a.3d845","name":"Read
Holding 40000F from slave 3 ","func":"// Absolutelyautomation.com\n//\n//
Change values in global variables to \n// modify ModBus behavior\n//
MBSlaveId :   Id of the Modbus Slave\n// MBFunction: ModBus function
\n// MBRegister: Register or Coil to be written or read.\n//
First register be written or read if\n//   a multi-register/
coil is called\n// MBQuantity: Quantity of registers/coils\n//
to be writter/readed. Only used\n//   in multi-register/coil
functions\n// MBArray: Array to put data to be written,\n//
or when data is stored after a \n//   read operation\n//
MBRs458Enable: Remove loopbacked chars\n//   produced by some
USB/RS232 to RS458 \n//   converters\n\n\n//
***\n// Example F03: Read Holding
register #40000F \n//   from slave 3 using RS485 converter\n
\nglobal.set('MBSlaveId',0x03);\nglobal.set('MBFunction',0x03);\nglobal.set
('MBRegister',0x000F);\nglobal.set('MBQuantity',0x0001);\nglobal.set
('MBRs458Enable',0x0001);\n\n\n\nreturn
msg;","outputs":1,"noerr":0,"x":455,"y":255,"wires":[["8411ab41.5e7fd8"]]},
{"id":"e044f03a.cbf64","type":"debug","z":"d2d74c5a.3d845","name":"","active":
true,"console":"false","complete":"false","x":684.3095245361328,"y":1782.16647
14813232,"wires":[]},
{"id":"f6094ac1.9de378","type":"switch","z":"d2d74c5a.3d845","name":"RS485
Enabled?","property":"MBRs458Enable","propertyType":"global","rules":
[{"t":"eq","v":"0","vt":"num"},{"t":"else"}
],"checkall":"true","outputs":2,"x":675.476188659668,"y":1605.571434020996,"wi
res":[["34061e18.016ed2"],["ea49d1a9.9815d"]]},
{"id":"ea49d1a9.9815d","type":"function","z":"d2d74c5a.3d845","name":"Remove
loopback chars","func":"var actualsize = msg.payload.length;\nvar txsize =
global.get('MBTxcharsCnt');\n\nif(actualsize - txsize > 0)\n{\n   var
trimmedsize = actualsize - txsize;\n   BinRxBuff = new Buffer(trimmedsize);\n
for(var i=0 ; i < trimmedsize ; i++)\n   {\n   BinRxBuff[i]=msg.payload
[i+txsize];\n   }\n   \n}\nelse\n{\n   BinRxBuff = new Buffer(0);\n}\n\n
\nmsg.payload=BinRxBuff;\nreturn
msg;","outputs":1,"noerr":0,"x":901.0714416503906,"y":1654.2856311798096,"wire
s":[["34061e18.016ed2"]]},
{"id":"6bb37b71.e294f4","type":"function","z":"d2d74c5a.3d845","name":"Write
holding register 40000B to slave 1","func":"// Absolutelyautomation.com\n//
\n// Change values in global variables to \n// modify ModBus behavior\n//

MBSlaveId :   Id of the Modbus Slave\n// MBFunction: ModBus function \n// MBRegister: Register or Coil to be written or read.\n// First register be written or read if\n//   a multi-register/ coil is called\n// MBQuantity: Quantity of registers/coils\n// to be writter/readed. Only used\n//   in multi-register/coil functions\n// MBArray: Array to put data to be written,\n// or when data is stored after a \n//   read operation\n// MBRs458Enable: Remove loopbacked chars\n//   produced by some USB/RS232 to RS458 \n//   converters\n\n\n// ***************************************\n// Example F06: Write value >0 to slave 1\n//   into output holding register 40000B\n// via RS485 converter\n\nglobal.set('MBSlaveId',0x01);\nglobal.set('MBFunction',0x06);\nglobal.set('MBRegister',0x000B);\n\nvar arrtmp = global.get('MBArray');\narrtmp[(global.get('MBRegister')-1)]=0x00FF;\nglobal.set('MBArray',arrtmp);\nglobal.set('MBRs458Enable',0x01);\n\n\nreturn msg;","outputs":1,"noerr":0,"x":2249.166675567627,"y":310.8333435058594,"wires":[["edc293af.4d199"]]},
{"id":"605e32b0.23f23c","type":"switch","z":"d2d74c5a.3d845","name":"Value > 500 ?","property":"MBGlobalValue","propertyType":"global","rules":[{"t":"gt","v":"500","vt":"num"},{"t":"else"}],"checkall":"true","outputs":2,"x":1958.7501220703125,"y":326.2500219345093,"wires":[["6bb37b71.e294f4"],["38fc5a78.865136"]]},
{"id":"38fc5a78.865136","type":"function","z":"d2d74c5a.3d845","name":"","func":"\nreturn msg;","outputs":1,"noerr":0,"x":2140.416759490967,"y":382.91666984558105,"wires":[[]]},{"id":"905f9fac.b6233","type":"link out","z":"d2d74c5a.3d845","name":"","links":["59823f74.3900f"],"x":2613.750186920166,"y":310.8333559036255,"wires":[]},
{"id":"8411ab41.5e7fd8","type":"function","z":"d2d74c5a.3d845","name":"sequence=0","func":"global.set('sequence',0x00);\nreturn msg;","outputs":1,"noerr":0,"x":705.8333206176758,"y":254.99998474121094,"wires":[["9159661f.48b738"]]},
{"id":"8326a4a1.1e97e8","type":"switch","z":"d2d74c5a.3d845","name":"sequence==0?","property":"sequence","propertyType":"global","rules":[{"t":"eq","v":"0","vt":"num"},{"t":"else"}],"checkall":"true","outputs":2,"x":1383.333309173584,"y":325.00001525878906,"wires":[["b65cd759.d1efb8"],["c1c9b27c.c4164"]]},
{"id":"c1c9b27c.c4164","type":"function","z":"d2d74c5a.3d845","name":"","func":"\nreturn msg;","outputs":1,"noerr":0,"x":1583.333333333333,"y":391.66666666666663,"wires":[[]]},
{"id":"edc293af.4d199","type":"function","z":"d2d74c5a.3d845","name":"sequence=1","func":"global.set('sequence',0x01);\nreturn msg;","outputs":1,"noerr":0,"x":2503.3331909179688,"y":309.99999237060547,"wires":[["905f9fac.b6233"]]},
{"id":"5e4db1a6.2e29e","type":"inject","z":"d2d74c5a.3d845","name":"Repetitive","topic":"","payload":"","payloadType":"date","repeat":"6","crontab":"","once":false,"x":149.99999237060547,"y":208.33332061767578,"wires":[["11163099.4ffebf"]]},{"id":"e73c106.4acf0f","type":"serial-port","z":"","serialport":"/dev/ttyUSB0","serialbaud":"9600","databits":"8","parity":"none","stopbits":"1","newline":"50","bin":"bin","out":"time","addchar":false}]

```
p[99];\n//var input101status = arrtmp[100];\n\n// ****************************************
[{"id":"2f8ccd56.fa5192","type":"tab","label":"Flow 2"},{"id":"c5484fab.af2c4","type":"serial out","z":"2f8ccd56.fa5192","name":"S
\n// Example F03: Read Holding registers #400499 to 400501\n//
from slave 150\n\nvar arrtmp = global.get('MBArray');\nvar holding499value =
arrtmp[498];\nvar holding500value = arrtmp[499];\nvar holding501value =
arrtmp[500];\n\n// ****************************************\n// Example F04:
Read Input registers #300006 to 300009\n//              from slave 30\n\n//
var arrtmp = global.get('MBArray');\n//var input6value = arrtmp[5];\n//var
input7value = arrtmp[6];\n//var input8value = arrtmp[7];\n//var input9value
= arrtmp[8];\n\nreturn
msg;","outputs":1,"noerr":0,"x":1217.3333892822266,"y":258.33336353302,"wires"
:[[]]},
{"id":"f39eda35.486a08","type":"function","z":"2f8ccd56.fa5192","name":"YOUR
ERROR HANDLER FUNCTION","func":"// Absolutelyautomation.com\n//\nreturn
msg;","outputs":1,"noerr":0,"x":1194.000099182129,"y":188.33337020874023,"wire
s":[[]]},
{"id":"e2602952.f6e6a8","type":"comment","z":"2f8ccd56.fa5192","name":"Absolut
elyautomation.com","info":"Absolutelyautomation.com","x":115,"y":42.3333587646
4844,"wires":[]},
{"id":"9226f54b.6cc228","type":"inject","z":"2f8ccd56.fa5192","name":"First
time auto
run","topic":"","payload":"","payloadType":"date","repeat":"","crontab":"","on
ce":true,"x":320.22635650634766,"y":713.5000553131104,"wires":
[["d1318291.0cc0a"]]},
{"id":"71342885.b0a0e8","type":"function","z":"2f8ccd56.fa5192","name":"Remove
loopback chars","func":"var actualsize = msg.payload.length;\nvar txsize =
global.get('MBTxcharsCnt');\n\nif( actualsize - txsize > 0)\n{\n    var
trimmedsize = actualsize - txsize;\n    BinRxBuff = new Buffer(trimmedsize);\n
for(var i=0 ; i < trimmedsize ; i++)\n    {\n        BinRxBuff[i]=msg.payload
[i+txsize];\n    }\n    \n}\nelse\n{\n    BinRxBuff = new Buffer(0);\n}\n\n
\nmsg.payload=BinRxBuff;\nreturn
msg;","outputs":1,"noerr":0,"x":692.119026184082,"y":1591.57133102417,"wires":
[["9803a1b3.adcb7"]]},
{"id":"2d4ba7a6.c766c8","type":"switch","z":"2f8ccd56.fa5192","name":"RS485
Enabled?","property":"MBRs458Enable","propertyType":"global","rules":
[{"t":"eq","v":"0","vt":"num"},{"t":"else"}
],"checkall":"true","outputs":2,"x":466.8253860473633,"y":1555.2062797546387,"
wires":[["9803a1b3.adcb7"],["71342885.b0a0e8"]]},
{"id":"e8869665.d81788","type":"serial-port","z":"","serialport":"/dev/
ttyUSB0","serialbaud":"9600","databits":"8","parity":"none","stopbits":"1","ne
wline":"25","bin":"bin","out":"time","addchar":false}]
```

PYTHON MODBUS RTU TEST CLIENT

```
#!/usr/bin/env python
#---------------------------------------------------------------------------#
# Absolutelyautomation.com
#---------------------------------------------------------------------------#

'''
Pymodbus Synchrnonous Client Examples
--------------------------------------------------------------------------

The following is an example of how to use the synchronous modbus client
implementation from pymodbus.

It should be noted that the client can also be used with
the guard construct that is available in python 2.5 and up::

    with ModbusClient('127.0.0.1') as client:
        result = client.read_coils(1,10)
        print result
'''
#---------------------------------------------------------------------------#
# import the various server implementations
#---------------------------------------------------------------------------#
#from pymodbus.client.sync import ModbusTcpClient as ModbusClient
#from pymodbus.client.sync import ModbusUdpClient as ModbusClient
from pymodbus.client.sync import ModbusSerialClient as ModbusClient

#---------------------------------------------------------------------------#
# configure the client logging
#---------------------------------------------------------------------------#
import logging
logging.basicConfig()
log = logging.getLogger()
log.setLevel(logging.DEBUG)

#---------------------------------------------------------------------------#
# choose the client you want
#---------------------------------------------------------------------------#
# make sure to start an implementation to hit against. For this
# you can use an existing device, the reference implementation in the tools
# directory, or start a pymodbus server.
#---------------------------------------------------------------------------#
#client = ModbusClient('127.0.0.1')
client = ModbusClient("rtu", port="/dev/ttyUSB0", baudrate=9600, timeout=2)


#---------------------------------------------------------------------------#
# example requests
#---------------------------------------------------------------------------#
# simply call the methods that you would like to use. An example session
# is displayed below along with some assert checks.
#---------------------------------------------------------------------------#
```

```
#rq = client.write_coil(1, True,unit= 0x01)
#rr = client.read_coils(1,1,unit= 0x01)
#assert(rq.function_code < 0x80)     # test that we are not an error
#assert(rr.bits[0] == True)          # test the expected value

#rq = client.write_coils(1, [True]*8,unit= 0x01)
#rr = client.read_coils(1,8,unit= 0x01)
#assert(rq.function_code < 0x80)     # test that we are not an error
#assert(rr.bits == [True]*8)         # test the expected value

#rq = client.write_coils(1, [False]*8,unit= 0x01)
#rr = client.read_discrete_inputs(1,8,unit= 0x01)
#assert(rq.function_code < 0x80)     # test that we are not an error
#assert(rr.bits == [False]*8)        # test the expected value

#rq = client.write_register(8, 200,unit= 0x01)
#rq = client.write_register(9, 200,unit= 0x01)
rq = client.write_register(10, 200,unit= 0x01)

#rr = client.read_holding_registers(8,200,unit= 0x01)
#assert(rq.function_code < 0x80)     # test that we are not an error
#assert(rr.registers[0] == 10)       # test the expected value

#rq = client.write_registers(1, [10]*8,unit= 0x01)
#rr = client.read_input_registers(1,8,unit= 0x01)
#assert(rq.function_code < 0x80)     # test that we are not an error
#assert(rr.registers == [10]*8)      # test the expected value

#rq = client.readwrite_registers(1, [20]*8,unit= 0x01)
#rr = client.read_input_registers(1,8,unit= 0x01)
#assert(rq.function_code < 0x80)     # test that we are not an error
#assert(rr.registers == [20]*8)      # test the expected value

#---------------------------------------------------------------------------#
# close the client
#---------------------------------------------------------------------------#
client.close()
```