PulseView

Saturday, December 23, 2023 2:36 PM

https://sigrok.org/wiki/Downloads

Open source tool for analyzing various protocols including modbus

https://learn.sparkfun.com/tutorials/using-the-usb-logic-analyzer-with-sigrok-pulseview/exploring-the-capabilities

Here's a fun sparkfun walkthrough of the tool and a link for a usb c version of the logic analyzer.

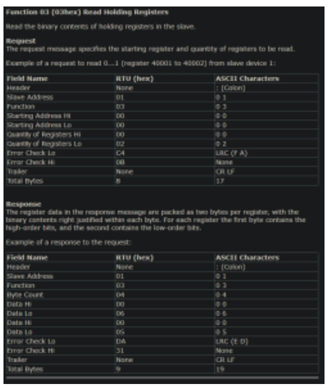
After a lot of troubleshooting a discovered that even at around double the sampling rate of 9600 baud, I was still getting inaccurate packets. So far it has been working at 25ks/s. Take the readings with a grain of salt but if sending a basic check packet to read the temperature

https://www.modbustools.com/modbus.html

This is the best reference I found to look at how the modbus proto is constructed for simple reg reads

https://minimalmodbus.readthedocs.io/en/stable/
https://minimalmodbus.readthedocs.io/en/stable/_modules/minimalmodbus.html

Docs for minimal modbus because vscode intellisense seems to only work 10% of the time… so that's a bummer



Function 03 (03hex) Read Holding Registers

Read the binary contents of holding registers in the slave.

Request
The request message specifies the starting register and quantity of registers to be read.

Example of a request to read 0…1 (register 40001 to 40002) from slave device 1:

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	01	0 1
Function	03	0 3
Starting Address Hi	00	0 0
Starting Address Lo	00	0 0
Quantity of Registers Hi	00	0 0
Quantity of Registers Lo	02	0 2
Error Check Lo	C4	LRC (F A)
Error Check Hi	0B	None
Trailer	None	CR LF
Total Bytes	8	17

Response
The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register the first byte contains the high-order bits, and the second contains the low-order bits.

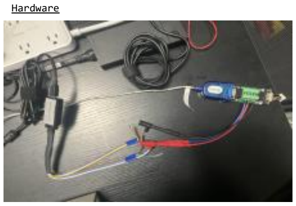
Example of a response to the request:

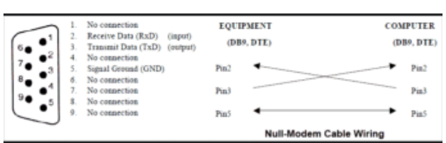
Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	01	0 1
Function	03	0 3
Byte Count	04	0 4
Data Hi	00	0 0
Data Lo	06	0 6
Data Hi	00	0 0
Data Lo	05	0 5
Error Check Lo	DA	LRC (E D)
Error Check Hi	31	None
Trailer	None	CR LF
Total Bytes	9	19

SUCCESS we have a perfectly captured send and receive…

This took like 3 weeks to figure out..

Hardware



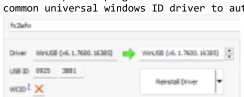


So just put the TX from pin 3 on DB9 to A0 of logic analyzer and pin 2 RX to A2.

PulseView Setup

run setup executable

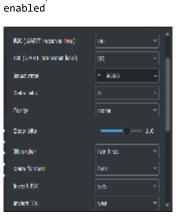
after setup this program is also installed which sets up the WCID, driver which is a common universal windows ID driver to automatically ID usb devices



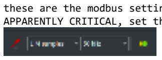
Select list devices to install the specific device that the logic analyzer firmware works with
the device refers to the chip set installed (cypress fx2)

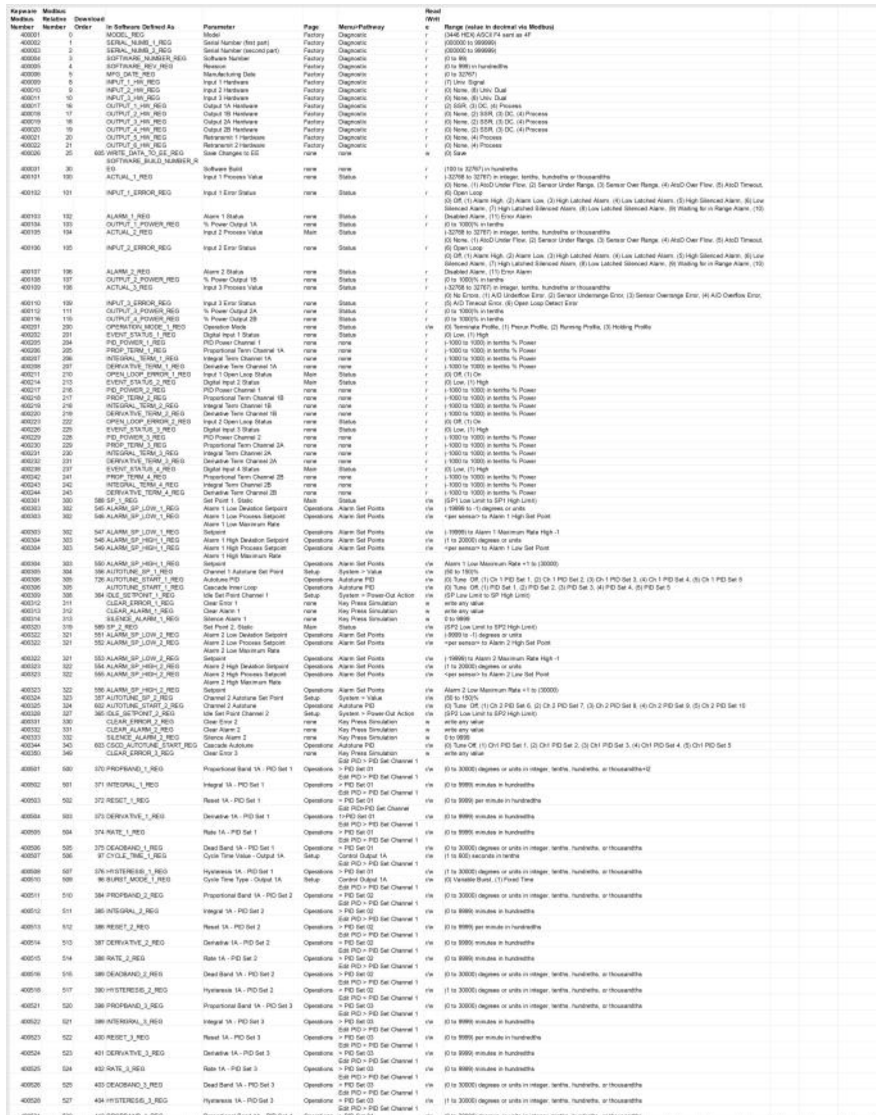
Launching pulseview should automatically populate the device with Salae hardware, which is the compatible version

choose protocol decoders at the top and load the modbus rtu protocol with UART RX/TX mode enabled



these are the modbus settings
APPARENTLY CRITICAL, set the sample rate to be > 20kHz!!





Keypad Modbus Number	Modbus Relative Number	Download Order	In Software Defined As	Parameter	Page	Menu>Pathway	Read/Write	Range (value in decimal via Modbus)
400001	0		MODEL_REG	Model	Factory	Diagnostic	r	(3446 HEX) ASCII F4 sent as 4F
400002	1		SERIAL_NUMB_1_REG	Serial Number (first part)	Factory	Diagnostic	r	(000000 to 999999)
400003	2		SERIAL_NUMB_2_REG	Serial Number (second part)	Factory	Diagnostic	r	(000000 to 999999)
400004	3		SOFTWARE_NUMBER_REG	Software Number	Factory	Diagnostic	r	(0 to 99)
400005	4		SOFTWARE_REV_REG	Revision	Factory	Diagnostic	r	(0 to 999) in hundredths
400006	5		MFG_DATE_REG	Manufacturing Date	Factory	Diagnostic	r	(0 to 32767)
400009	8		INPUT_1_HW_REG	Input 1 Hardware	Factory	Diagnostic	r	(7) Univ. Signal
400010	9		INPUT_2_HW_REG	Input 2 Hardware	Factory	Diagnostic	r	(0) None, (6) Univ. Dual
400011	10		INPUT_3_HW_REG	Input 3 Hardware	Factory	Diagnostic	r	(0) None, (6) Univ. Dual
400017	16		OUTPUT_1_HW_REG	Output 1A Hardware	Factory	Diagnostic	r	(2) SSR, (3) DC, (4) Process
400018	17		OUTPUT_2_HW_REG	Output 1B Hardware	Factory	Diagnostic	r	(0) None, (2) SSR, (3) DC, (4) Process
400019	18		OUTPUT_3_HW_REG	Output 2A Hardware	Factory	Diagnostic	r	(0) None, (2) SSR, (3) DC, (4) Process
400020	19		OUTPUT_4_HW_REG	Output 2B Hardware	Factory	Diagnostic	r	(0) None, (2) SSR, (3) DC, (4) Process
400021	20		OUTPUT_5_HW_REG	Retransmit 1 Hardware	Factory	Diagnostic	r	(0) None, (4) Process
400022	21		OUTPUT_6_HW_REG	Retransmit 2 Hardware	Factory	Diagnostic	r	(0) None, (4) Process
400026	25	605	WRITE_DATA_TO_EE_REG	Save Changes to EE	none	none	w	(0) Save
400031	30		SOFTWARE_BUILD_NUMBER_REG	Software Build	none	none	r	(100 to 32767) in hundredths
400101	100		ACTUAL_1_REG	Input 1 Process Value	none	Status	r	(-32768 to 32767) in integer, tenths, hundredths or thousandths
400102	101		INPUT_1_ERROR_REG	Input 1 Error Status	none	Status	r	(0) None, (1) AtoD Under Flow, (2) Sensor Under Range, (3) Sensor Over Range, (4) AtoD Over Flow, (5) AtoD Timeout, (6) Open Loop
400103	102		ALARM_1_REG	Alarm 1 Status	none	Status	r	(0) Off, (1) Alarm High, (2) Alarm Low, (3) High Latched Alarm, (4) Low Latched Alarm, (5) High Silenced Alarm, (6) Low Silenced Alarm, (7) High Latched Silenced Alarm, (8) Low Latched Silenced Alarm, (9) Waiting for in Range Alarm, (10) Disabled Alarm, (11) Error Alarm
400104	103		OUTPUT_1_POWER_REG	% Power Output 1A	none	Status	r	(0 to 1000)% in tenths
400105	104		ACTUAL_2_REG	Input 2 Process Value	Main	Status		(-32768 to 32767) in integer, tenths, hundredths or thousandths
400106	105		INPUT_2_ERROR_REG	Input 2 Error Status	none	Status	r	(0) None, (1) AtoD Under Flow, (2) Sensor Under Range, (3) Sensor Over Range, (4) AtoD Over Flow, (5) AtoD Timeout, (6) Open Loop
400107	106		ALARM_2_REG	Alarm 2 Status	none	Status	r	(0) Off, (1) Alarm High, (2) Alarm Low, (3) High Latched Alarm, (4) Low Latched Alarm, (5) High Silenced Alarm, (6) Low Silenced Alarm, (7) High Latched Silenced Alarm, (8) Low Latched Silenced Alarm, (9) Waiting for in Range Alarm, (10) Disabled Alarm, (11) Error Alarm
400108	107		OUTPUT_2_POWER_REG	% Power Output 1B	none	Status	r	(0 to 1000)% in tenths
400109	108		ACTUAL_3_REG	Input 3 Process Value	none	Status	r	(-32768 to 32767) in integer, tenths, hundredths or thousandths
400110	109		INPUT_3_ERROR_REG	Input 3 Error Status	none	Status	r	(0) No Errors, (1) A/D Underflow Error, (2) Sensor Underrange Error, (3) Sensor Overrange Error, (4) A/D Overflow Error, (5) A/D Timeout Error, (6) Open Loop Detect Error
400112	111		OUTPUT_3_POWER_REG	% Power Output 2A	none	Status	r	(0 to 1000)% in tenths
400116	115		OUTPUT_4_POWER_REG	% Power Output 2B	none	Status	r	(0 to 1000)% in tenths
400201	200		OPERATION_MODE_1_REG	Operation Mode	none	Status	r/w	(0) Terminate Profile, (1) Pause Profile, (2) Running Profile, (3) Holding Profile
400202	201		EVENT_STATUS_1_REG	Digital Input 1 Status	none	Status	r	(0) Low, (1) High
400205	204		PID_POWER_1_REG	PID Power Channel 1	none	none	r	(-1000 to 1000) in tenths % Power
400206	205		PROP_TERM_1_REG	Proportional Term Channel 1A	none	none	r	(-1000 to 1000) in tenths % Power
400207	206		INTEGRAL_TERM_1_REG	Integral Term Channel 1A	none	none	r	(-1000 to 1000) in tenths % Power
400208	207		DERIVATIVE_TERM_1_REG	Derivative Term Channel 1A	none	none	r	(-1000 to 1000) in tenths % Power
400211	210		OPEN_LOOP_ERROR_1_REG	Input 1 Open Loop Status	Main	Status	r	(0) Off, (1) On
400214	213		EVENT_STATUS_2_REG	Digital Input 2 Status	Main	Status	r	(0) Low, (1) High
400217	216		PID_POWER_2_REG	PID Power Channel 1	none	none	r	(-1000 to 1000) in tenths % Power
400218	217		PROP_TERM_2_REG	Proportional Term Channel 1B	none	none	r	(-1000 to 1000) in tenths % Power
400219	218		INTEGRAL_TERM_2_REG	Integral Term Channel 1B	none	none	r	(-1000 to 1000) in tenths % Power
400220	219		DERIVATIVE_TERM_2_REG	Derivative Term Channel 1B	none	none	r	(-1000 to 1000) in tenths % Power
400223	222		OPEN_LOOP_ERROR_2_REG	Input 2 Open Loop Status	none	Status	r	(0) Off, (1) On
400226	225		EVENT_STATUS_3_REG	Digital Input 3 Status	none	Status	r	(0) Low, (1) High
400229	228		PID_POWER_3_REG	PID Power Channel 2	none	none	r	(-1000 to 1000) in tenths % Power
400230	229		PROP_TERM_3_REG	Proportional Term Channel 2A	none	none	r	(-1000 to 1000) in tenths % Power
400231	230		INTEGRAL_TERM_3_REG	Integral Term Channel 2A	none	none	r	(-1000 to 1000) in tenths % Power
400232	231		DERIVATIVE_TERM_3_REG	Derivative Term Channel 2A	none	none	r	(-1000 to 1000) in tenths % Power
400238	237		EVENT_STATUS_4_REG	Digital Input 4 Status	Main	Status	r	(0) Low, (1) High
400242	241		PROP_TERM_4_REG	Proportional Term Channel 2B	none	none	r	(-1000 to 1000) in tenths % Power
400243	242		INTEGRAL_TERM_4_REG	Integral Term Channel 2B	none	none	r	(-1000 to 1000) in tenths % Power
400244	243		DERIVATIVE_TERM_4_REG	Derivative Term Channel 2B	none	none	r	(-1000 to 1000) in tenths % Power
400301	300	588	SP_1_REG	Set Point 1, Static	Main	Status	r/w	(SP1 Low Limit to SP1 High Limit)
400303	302	545	ALARM_SP_LOW_1_REG	Alarm 1 Low Deviation Setpoint	Operations	Alarm Set Points	r/w	(-19999 to -1) degrees or units
400303	302	546	ALARM_SP_LOW_1_REG	Alarm 1 Low Process Setpoint	Operations	Alarm Set Points	r/w	<per sensor> to Alarm 1 High Set Point
400303	302	547	ALARM_SP_LOW_1_REG	Alarm 1 Low Maximum Rate Setpoint	Operations	Alarm Set Points	r/w	(-19999) to Alarm 1 Maximum Rate High -1
400304	303	548	ALARM_SP_HIGH_1_REG	Alarm 1 High Deviation Setpoint	Operations	Alarm Set Points	r/w	(1 to 20000) degrees or units
400304	303	549	ALARM_SP_HIGH_1_REG	Alarm 1 High Process Setpoint	Operations	Alarm Set Points	r/w	<per sensor> to Alarm 1 Low Set Point
400304	303	550	ALARM_SP_HIGH_1_REG	Alarm 1 High Maximum Rate Setpoint	Operations	Alarm Set Points	r/w	Alarm 1 Low Maximum Rate +1 to (30000)
400305	304	356	AUTOTUNE_SP_1_REG	Channel 1 Autotune Set Point	Setup	System > Value	r/w	(50 to 150)%
400306	305	726	AUTOTUNE_START_1_REG	Autotune PID	Operations	Autotune PID	r/w	(0) Tune Off, (1) Ch 1 PID Set 1, (2) Ch 1 PID Set 2, (3) Ch 1 PID Set 3, (4) Ch 1 PID Set 4, (5) Ch 1 PID Set 5
400306	305		AUTOTUNE_START_1_REG	Cascade Inner Loop	Operations	Autotune PID	r/w	(0) Tune Off, (1) PID Set 1, (2) PID Set 2, (3) PID Set 3, (4) PID Set 4, (5) PID Set 5
400309	308	364	IDLE_SETPOINT_1_REG	Idle Set Point Channel 1	Setup	System > Power-Out Action	r/w	(SP Low Limit to SP High Limit)
400312	311		CLEAR_ERROR_1_REG	Clear Error 1	none	Key Press Simulation	w	write any value
400313	312		CLEAR_ALARM_1_REG	Clear Alarm 1	none	Key Press Simulation	w	write any value
400314	313		SILENCE_ALARM_1_REG	Silence Alarm 1	none	Key Press Simulation	w	0 to 9999
400320	319	589	SP_2_REG	Set Point 2, Static	Main	Status	r/w	(SP2 Low Limit to SP2 High Limit)
400322	321	551	ALARM_SP_LOW_2_REG	Alarm 2 Low Deviation Setpoint	Operations	Alarm Set Points	r/w	(-9999 to -1) degrees or units
400322	321	552	ALARM_SP_LOW_2_REG	Alarm 2 Low Process Setpoint	Operations	Alarm Set Points	r/w	<per sensor> to Alarm 2 High Set Point
400322	321	553	ALARM_SP_LOW_2_REG	Alarm 2 Low Maximum Rate Setpoint	Operations	Alarm Set Points	r/w	(-19999) to Alarm 2 Maximum Rate High -1
400323	322	554	ALARM_SP_HIGH_2_REG	Alarm 2 High Deviation Setpoint	Operations	Alarm Set Points	r/w	(1 to 20000) degrees or units
400323	322	555	ALARM_SP_HIGH_2_REG	Alarm 2 High Process Setpoint	Operations	Alarm Set Points	r/w	<per sensor> to Alarm 2 Low Set Point
400323	322	556	ALARM_SP_HIGH_2_REG	Alarm 2 High Maximum Rate Setpoint	Operations	Alarm Set Points	r/w	Alarm 2 Low Maximum Rate +1 to (30000)
400324	323	357	AUTOTUNE_SP_2_REG	Channel 2 Autotune Set Point	Setup	System > Value	r/w	(50 to 150)%
400325	324	602	AUTOTUNE_START_2_REG	Channel 2 Autotune	Operations	Autotune PID	r/w	(0) Tune Off, (1) Ch 2 PID Set 6, (2) Ch 2 PID Set 7, (3) Ch 2 PID Set 8, (4) Ch 2 PID Set 9, (5) Ch 2 PID Set 10
400328	327	365	IDLE_SETPOINT_2_REG	Idle Set Point Channel 2	Setup	System > Power-Out Action	r/w	(SP2 Low Limit to SP2 High Limit)
400331	330		CLEAR_ERROR_2_REG	Clear Error 2	none	Key Press Simulation	w	write any value
400332	331		CLEAR_ALARM_2_REG	Clear Alarm 2	none	Key Press Simulation	w	write any value
400333	332		SILENCE_ALARM_2_REG	Silence Alarm 2	none	Key Press Simulation	w	0 to 9999
400344	343	603	CSCD_AUTOTUNE_START_REG	Cascade Autotune	Operations	Autotune PID	r/w	(0) Tune Off, (1) Ch1 PID Set 1, (2) Ch1 PID Set 2, (3) Ch1 PID Set 3, (4) Ch1 PID Set 4, (5) Ch1 PID Set 5
400350	349		CLEAR_ERROR_3_REG	Clear Error 3	none	Key Press Simulation	w	write any value
400501	500	370	PROPBAND_1_REG	Proportional Band 1A - PID Set 1	Operations	Edit PID > PID Set Channel 1 > PID Set 01	r/w	(0 to 30000) degrees or units in integer, tenths, hundreths, or thousandths+I2
400502	501	371	INTEGRAL_1_REG	Integral 1A - PID Set 1	Operations	Edit PID > PID Set Channel 1 > PID Set 01	r/w	(0 to 9999) minutes in hundredths
400503	502	372	RESET_1_REG	Reset 1A - PID Set 1	Operations	Edit PID > PID Set Channel 1 > PID Set 01	r/w	(0 to 9999) per minute in hundredths
400504	503	373	DERIVATIVE_1_REG	Derivative 1A - PID Set 1	Operations	Edit PID>PID Set Channel 1>PID Set 01	r/w	(0 to 9999) minutes in hundredths
400505	504	374	RATE_1_REG	Rate 1A - PID Set 1	Operations	Edit PID > PID Set Channel 1 > PID Set 01	r/w	(0 to 9999) minutes in hundredths
400506	505	375	DEADBAND_1_REG	Dead Band 1A - PID Set 1	Operations	Edit PID > PID Set Channel 1 > PID Set 01	r/w	(0 to 30000) degrees or units in integer, tenths, hundreths, or thousandths
400507	506	97	CYCLE_TIME_1_REG	Cycle Time Value - Output 1A	Setup	Control Output 1A	r/w	(1 to 600) seconds in tenths
400508	507	376	HYSTERESIS_1_REG	Hysteresis 1A - PID Set 1	Operations	Edit PID > PID Set Channel 1 > PID Set 01	r/w	(1 to 30000) degrees or units in integer, tenths, hundreths, or thousandths
400510	509	98	BURST_MODE_1_REG	Cycle Time Type - Output 1A	Setup	Control Output 1A	r/w	(0) Variable Burst, (1) Fixed Time
400511	510	384	PROPBAND_2_REG	Proportional Band 1A - PID Set 2	Operations	Edit PID > PID Set Channel 1 > PID Set 02	r/w	(0 to 30000) degrees or units in integer, tenths, hundreths, or thousandths
400512	511	385	INTEGRAL_2_REG	Integral 1A - PID Set 2	Operations	Edit PID > PID Set Channel 1 > PID Set 02	r/w	(0 to 9999) minutes in hundredths
400513	512	386	RESET_2_REG	Reset 1A - PID Set 2	Operations	Edit PID > PID Set Channel 1 > PID Set 02	r/w	(0 to 9999) per minute in hundredths
400514	513	387	DERIVATIVE_2_REG	Derivative 1A - PID Set 2	Operations	Edit PID > PID Set Channel 1 > PID Set 02	r/w	(0 to 9999) minutes in hundredths
400515	514	388	RATE_2_REG	Rate 1A - PID Set 2	Operations	Edit PID > PID Set Channel 1 > PID Set 02	r/w	(0 to 9999) minutes in hundredths
400516	515	389	DEADBAND_2_REG	Dead Band 1A - PID Set 2	Operations	Edit PID > PID Set Channel 1 > PID Set 02	r/w	(0 to 30000) degrees or units in integer, tenths, hundreths, or thousandths
400518	517	390	HYSTERESIS_2_REG	Hysteresis 1A - PID Set 2	Operations	Edit PID > PID Set Channel 1 > PID Set 02	r/w	(1 to 30000) degrees or units in integer, tenths, hundreths, or thousandths
400521	520	398	PROPBAND_3_REG	Proportional Band 1A - PID Set 3	Operations	Edit PID > PID Set Channel 1 > PID Set 03	r/w	(0 to 30000) degrees or units in integer, tenths, hundreths, or thousandths
400522	521	399	INTERGRAL_3_REG	Integral 1A - PID Set 3	Operations	Edit PID > PID Set Channel 1 > PID Set 03	r/w	(0 to 9999) minutes in hundredths
400523	522	400	RESET_3_REG	Reset 1A - PID Set 3	Operations	Edit PID > PID Set Channel 1 > PID Set 03	r/w	(0 to 9999) per minute in hundredths
400524	523	401	DERIVATIVE_3_REG	Derivative 1A - PID Set 3	Operations	Edit PID > PID Set Channel 1 > PID Set 03	r/w	(0 to 9999) minutes in hundredths
400525	524	402	RATE_3_REG	Rate 1A - PID Set 3	Operations	Edit PID > PID Set Channel 1 > PID Set 03	r/w	(0 to 9999) minutes in hundredths
400526	525	403	DEADBAND_3_REG	Dead Band 1A - PID Set 3	Operations	Edit PID > PID Set Channel 1 > PID Set 03	r/w	(0 to 30000) degrees or units in integer, tenths, hundreths, or thousandths
400528	527	404	HYSTERESIS_3_REG	Hysteresis 1A - PID Set 3	Operations	Edit PID > PID Set Channel 1 > PID Set 03	r/w	(1 to 30000) degrees or units in integer, tenths, hundreths, or thousandths
400531	530	412	PROPBAND_4_REG	Proportional Band 1A - PID Set 4	Operations	Edit PID > PID Set Channel 1 > PID Set 04	r/w	(0 to 30000) degrees or units in integer, tenths, hundreths, or thousandths

that's's it! Can press go and then send rtu packets

The script itself is straightforward,

Using one of the serial decoders (B&B electronics or one of the Pluggable ones)

pip3 install minimalmodbus

```python
import minimalmodbus

# Set up the instrument
instrument = minimalmodbus.Instrument('COM10', 1)  # find the com device in device manager, 1: the slave address
instrument.serial.baudrate = 9600                  # Baud F4 uses either 9600 or 19200
instrument.serial.bytesize = 8                     # data bit size, refer to modbus manual chart, each pair of hex digits is 1 byte:8 bits
instrument.serial.stopbits = 1                     # 1 stop bit is default but we state it explicitly
instrument.serial.timeout  = 0.2                   # seconds.  increase if you are having problems


print (instrument)                                 # prints out the settings for instrument
print(instrument.read_register(100))               # reads the current chamber temp
```

minimalmodbus.Instrument<id=0x26988f14490, address=1, mode=rtu, close_port_after_each_call=False, precalculate_read_size=True, clear_buffers_before_each_transaction=True, handle_local_echo=False, debug=False, serial=Serial<id=0x26988f14460, open=True>(port='COM10', baudrate=9600, bytesize=8, parity='N', stopbits=1, timeout=0.2, xonxoff=False, rtscts=False, dsrdtr=False)>

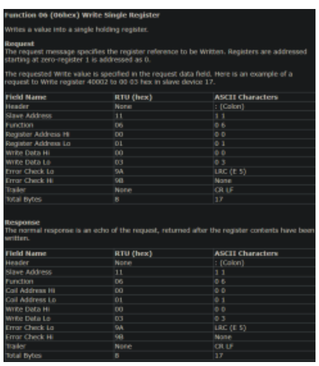
These chambers seem to not like a parity bit because of the 8 data bit format, but you could add it and play around using this line

an EVEN parity bit asserted (meaning 1), would mean that the data should have an even number of bits, so if the data returns an odd number the parity bit would not be asserted and would result in some sort of checksum error.
I struggle with parity but thankfully its not needed

Set Temp

Set point register is 300

```python
print(instrument.write_register(300, 1000, functioncode=6, signed=True))
```



Function 06 (06hex) Write Single Register

Writes a value into a single holding register.

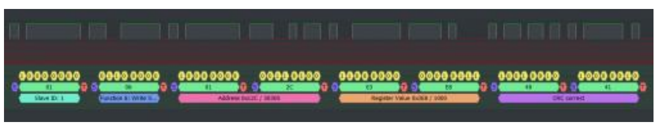
Request
The request message specifies the register reference to be Written. Registers are addressed starting at zero-register 1 is addressed as 0.

The requested Write value is specified in the request data field. Here is an example of a request to Write register 40002 to 00 03 hex in slave device 17.

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	11	1 1
Function	06	0 6
Register Address Hi	00	0 0
Register Address Lo	01	0 1
Write Data Hi	00	0 0
Write Data Lo	03	0 3
Error Check Lo	9A	LRC (E 5)
Error Check Hi	9B	None
Trailer	None	CR LF
Total Bytes	8	17

Response
The normal response is an echo of the request, returned after the register contents have been written.

Field Name	RTU (hex)	ASCII Characters
Header	None	: (Colon)
Slave Address	11	1 1
Function	06	0 6
Coil Address Hi	00	0 0
Coil Address Lo	01	0 1
Write Data Hi	00	0 0
Write Data Lo	03	0 3
Error Check Lo	9A	LRC (E 5)
Error Check Hi	9B	None
Trailer	None	CR LF
Total Bytes	8	17



01	Slave 1
06	Write single reg
01	Reg High byte
2c	reg Low byte
03	Data high byte
e8	data low byte
49	crc high
41	crc low

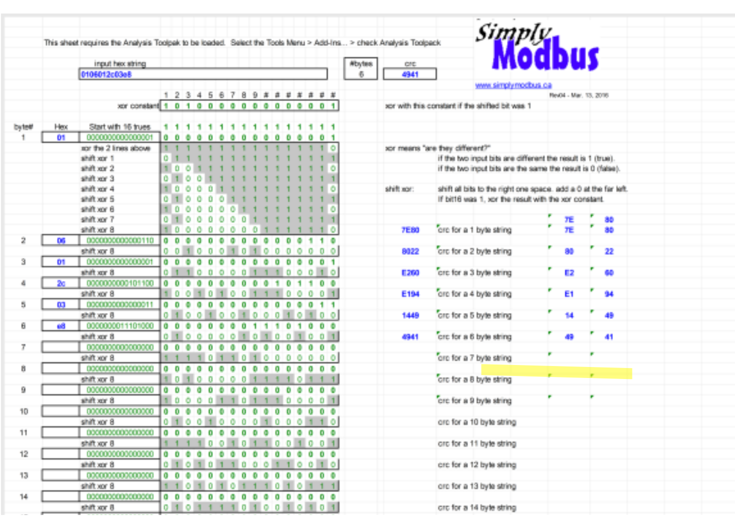
012c=300

03e8=1000
(implied decimal 0:100.0 Celcius)

01	slave id
06	write single reg
01	address high
2c	address low
03	data high
e8	data low
49	crc high
41	crc low

K cool, it works yo

The last thing I want to do is caluclate the crc. it should be straight forward



Python example, but keep in mind in outputs LOW BYTE first
so switch the two

```python
def modbus_crc(data):
    crc = 0xFFFF
    for byte in data:
        crc ^= byte
        for _ in range(8):
            lsb = crc & 1
            crc >>= 1
            if lsb:
                crc ^= 0xA001
    return crc
# Example usage
data = [0x01, 0x06, 0x01, 0x2c, 0x03, 0xe8]  # Sample Modbus data
crc_result = modbus_crc(data)
print(f"CRC: {crc_result:04X}")
```

> initialze crc to 1111 1111 1111 1111
> first byte (sign extend 0's)
  - 0000 0000 0000 0001
  - crc ^= byte
    - ^= means XOR, TRUE if different

1111 1111 1111 1111	
0000 0000 0000 0001	
1111 1111 1111 1110	NEW CRC

> for each bit in byte
  - lsb -least significant bit

1111 1111 1111 1110	AND
0000 0000 0000 0001	Must be the same to eval to true
0000 0000 0000 0000	therefore lsb is 0

  - this is basically a way to "mask" the results to only get the lsb
  - you can ge the lsb but "anding" the crc with a 1

> LSR - logical shift right the crc (which is still 1111 1111 1111 1110)

1111 1111 1111 1110	LSR
0111 1111 1111 1111	shifted right, fill in 16th bit with a zero

	shift xor 8	0	1	0	0	1	0	0	0	0	1	0	0	0	1	1	0		crc for a 10 byte string
11	0000000000000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	shift xor 8	1	1	1	1	0	0	1	0	1	1	0	0	1	0	0	1		crc for a 11 byte string
12	0000000000000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	shift xor 8	0	1	0	1	0	1	1	0	0	0	1	1	0	0	1	0		crc for a 12 byte string
13	0000000000000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	shift xor 8	1	1	0	1	0	1	0	1	1	1	0	1	0	1	1	1		crc for a 13 byte string
14	0000000000000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	shift xor 8	0	1	0	1	1	1	1	0	1	0	0	1	0	1	0	1		crc for a 14 byte string
15	0000000000000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	shift xor 8	0	1	1	0	1	1	1	1	1	0	0	1	1	1	1	0		crc for a 15 byte string
16	0000000000000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
	shift xor 8	1	0	1	0	1	0	0	0	1	1	1	0	1	1	1	0		crc for a 16 byte string
																		#	**16 bytes max**

- this is basically a way to "mask" the results to only get the lsb
- you can ge the lsb but "anding" the crc with a 1

\> LSR - logical shift right the crc (which is still 1111 1111 1111 1110)

1111 1111 1111 1110	LSR
0111 1111 1111 1111	shifted right, fill in 16th bit with a zero

\> if lsb (meaning lsb = 1)
- XOR new CRC with the polynomial
- the polynomial is hex A001 ; decimal 40,961
- I tried researching this to see why it was this number but from what i've found its just JFM

SO ALL OF THAT FIRST JUST THE FIRST BIT OF A 6 BYTE PACKET!!! It would take million years to do the whole thing by hand

so with some print-foo