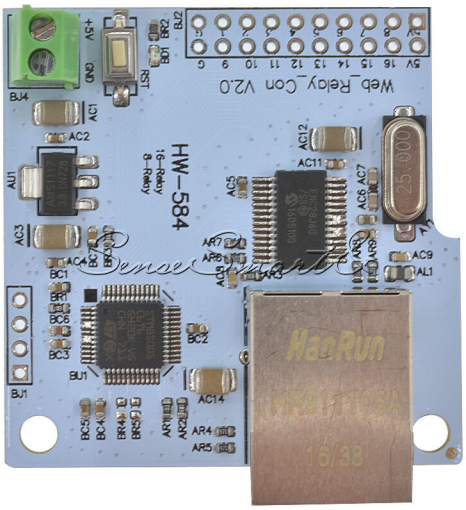
Network Module Manual

Replacement Firmware for the
Web_Relay_Con V2.0 HW-584
August 17, 2022

Author: Michael Nielson
nielsonm.projects@gmail.com

Introduction



Did you buy one (or more) of these Network Modules and then find disappointment in the software on the board?

- All of the modules have the same MAC address. That's a problem if you want more than one on your network. And the supplier does not give you a way to change the MAC.
- If you change the IP Address the device returns to its default IP Address when it power cycles. That makes it pretty much useless even if you only put one on your network - unless you're OK with it always having IP Address 192.168.1.4.

I decided to write my own firmware for the device to provide a web server interface that let's you change the IP Address, Gateway (Default Router) Address, Netmask, Port number (a REAL port number), and MAC Address. I also added the ability for the device to remember all these settings through a power cycle. Any Output settings you make are also optionally saved through a power cycle (Outputs typically being Relay controls).

NOTE 1: The software provided in this project only works with the “Web_Relays_Con V2.0 HW-584” which is based on the STM8S-005 or STM8S-105 processor and ENC28J60 ethernet controller. I haven't tried it with any other version of the hardware. I think the V.1 FC-160 is based on a Nuvoton processor and this code and the tools are incompatible.

NOTE 2: I am not in any way associated with the manufacturer of this device. I only wrote code to run on it for my own hobby purposes, and I am making it available for other hobbyists.

NOTE 3: If you’re looking to buy these modules the best source I’ve found is eBay. Best search term is “ENC28J60 Network Module”. I’ve also seen them on Amazon, Banggood, and AliExpress. In some cases they show photos of both the V.1 and V2.0 versions, but don’t provide a way of specifying which one you want. You may need to communicate with the seller to be sure you’ll get the V2.0 device.

Thank You!

Many thanks to Carlos Ladeira for his help with user interface ideas, many hours of testing, and his patience during development of the MQTT version of this code. And many thanks to Jevdeni Kiski for his guidance and code contributions in developing the Home Assistant interface, new Configuration interface, and JavaScript. We’ve ended up with a much better project as a result.

Document License

Table of Contents

Change Log

June 20, 2020 – Initial Release

August 2, 2020 – Added descriptions of "8 Output / 8 Input" and "16 Input" configurations.

August 6, 2020 – Added a note to make sure people know they don't need to set up a development environment unless they need to change the code.

August 20, 2020 – Added a note on editing the .stp file

November 16, 2020 – Major change: Added MQTT support

Other changes:

- Changed the GUI to have a Configuration page in place of the former Address Settings page
- Moved the device Name input field to the new Configuration page
- Moved the Relay Invert control to the new Configuration page
- Added an Input Invert control
- Added the ability to select whether all relays retain their state, are all forced off, or are all forced on when a power cycle occurs. This is in the "Config" settings added to the Configuration page.
- Added the ability to select whether the Ethernet interface operates in Half or Full Duplex.
- Added the ability to configure settings for MQTT Broker IP Address and Port.
- Added the ability to provide options MQTT ID and Password information.
- Added MQTT connection status indicators in the Configuration page.
- Fixed several corner case bugs in the web server code that were causing anomalous behavior in browsers.

November 19, 2020 – Minor edits to this document:

- Corrected the screen shot for the MQTT IO Control page
- Changed text to indicate that Full Duplex worked with some unmanaged switches, but not all.
- Added that you can use "all" to turn all relays on or off with MQTT.

November 21, 2020 – Minor change to code and this document:

- Fix to the "short form" Input pin state information – it was not showing correctly if the Invert function was on.
- Added tables showing how the displayed pin states relate to the physical pin voltages.
- Added an error count statistics display to the MQTT version of the code.

November 26, 2020 – Document changes only:

- Rearranged some sections in the document.
- Added EEPROM bit field definitions.
- Added tables showing relationship of browser and MQTT fields to pin logic levels.

November 30, 2020

Code changes:

- Added the /98 REST command
- Corrected typo in the HELP pages that mis-stated the REST command numbers.

Document changes:

- Added section describing all the REST commands.

December 2, 2020

Code changes:

- Fixed issue regarding browsers on multiple IP addresses.
- Fixed issue regarding browser interference on page changes.

Document changes:

- Added section to describe functional limitations (like number of browser sessions, browser interference).
- Added section on an alternative method for entering the initial IP Address.

December 4, 2020

Code changes:

- Code change made to reduce or eliminate relay state changes during reboot.

Document changes:

- Added section on alternative hardware design methods for maintaining relays states during power loss.

December 18, 2020 Code Revision 20201218 2202

Code changes:

- Fixed bug in RXERIF diagnostic counter (MQTT builds only).
- Significant rewrite of timing functions around the MQTT code to improve the rate at which MQTT commands can be executed.
- Added Independent Watchdog (hardware watchdog) to restart the module should it hang.
- Removed the Error Statistics button from the MQTT builds.
- Added EEPROM lock-out except when intentionally making EEPROM changes.

Document changes:

- Relocated the MQTT Error Statistic description

December 21, 2020

MAJOR CHANGE: Changed all MQTT commands to match the Home Assistant standard.

Code changes:

- Fixed timing issue in MQTT command processing
- Changed all MQTT commands to match the Home Assistant standard.
- Simplified the appearance of the browser interface to make code space available for continued MQTT development.

Document changes:

- Updated MQTT command list

December 30, 2020 Code Revision 20201230 0411

Code changes:

- Added window sizing to improve appearance on small devices (iPhone etc).
- Added Home Assistant Auto Discovery including a Config setting to enable Auto Discovery.
- Changed reset button routine to prevent hardware watchdog from firing while button is pressed.

Document changes:

- Added description of Auto Discovery and associated Config setting

January 23, 2021 Code Revision 20210123 1257

MAJOR CODE UPDATE. This code update will retain the majority of your settings from previous releases like your IP addresses, Port numbers, and MAC address. However, you need to re-enter

settings associated with the IO pins (Input/Output, Invert, after Boot State, etc). This will be readily apparent in the Browser GUI. Note that the default is for all pins to be Input pins.

Code changes:

- Updated GUI to improve the way IP Addresses and MAC numbers are entered.
- Updated GUI and code to provide a single build for Browser and MQTT configurations.
- Updated GUI and code to allow each pin to be individually configured as an input or output, and to allow each pin to be individually configured for Invert and Power On state.
- "Help" no longer fits in the Flash on the module, so a link was added to get to documentation on the GitHub site.
- Added a checkbox based "Features" field to replace the former "Config" bytes.
- Improved the Home Assistant Auto Discovery feature to better control population of the Home Assistant device management screen when Pin configurations are changed.

Document changes:

- Describe above code changes
- Changed Screen Shots
- Replaced Config Settings section with Feature Settings section
- Added section on Individual IO Settings
- Updated REST commands
- Eliminated section "Notes on Compiling Different Configurations"
- Updated section "Location of EEPROM Variables"
- Added section "Alternative Way to Force Defaults"
- Updated section "Programming the Module" to reflect use of a single release for all configurations.

January 26, 2021 Code Revision 20210126 0355

Code changes:

- Repaired Debug Statistics functionality, HOWEVER that functionality is still not enabled due to lack of space and continuing work on other features and bug fixes.

Document changes:

- None.

January 26, 2021 Code Revision 20210126 0527

Code changes:

- Fixed two bugs where a integer-to-hex conversion was being made that should have been integer-to-decimal.

Document changes:

- None.

January 27, 2021 Code Revision 20210127 1112

Code changes:

- Fixed bug that was allowing blank fields in the IP Address, Gateway, Netmask, and Port fields.
- Fixed bug that was preventing operation of the REST 00-31 commands.
- Fixed bug that was causing the REST 98 and 99 commands to output the wrong values.

Document changes:

- None.

February 8, 2021 Code Revision 20210208 0523

Code changes:

- Added DS18B20 Temperature Sensor interface and display functions

Document changes:

- Documented the DS18B20 interface

February 20, 2021 Code Revision 20210220 2350

Code changes:

- Added support for UART debug output messages on IO 11 in Developer builds
- Re-enabled the former MQTT Error Statistics page renamed as Link Error Statistics

Document changes:

- Documented support for UART debug output messages on IO 11
- Documented Link Error Statistics
- Added description of Debug bytes for developers
- Changed some section titles to clarify which ones are for Developers, and reorganized the document to collocate Developer sections.
- Moved the Change Log after the Table of Contents
- Updated information on where to find .stp and .sx files on GitHub

February 21, 2021 Code Revision 20210221 1826

Code changes:

- Bug fix in POST parsing routine

Document changes:

- None

April 12, 2021 Code Revision 20210412 1333

Code changes:

- Code size reductions
- Added separate build for "Browser Only" users (no MQTT) to free up memory for Browser Only features:
  - Browser Only build includes IO Names, IO Timers, and the Network Statistics page.
- Added DeviceName to the Browser tab
- Added degrees F to browser temperature sensor display

Document changes:

- Added section with Developer information on Stack Overflow detection
- Added section with Developer information on mqtt sendbuf sizing
- Added section with Developer information on Flash programming from the application.
- Added section with Developer information on Flash wear
- Added section on the the Browser Only features
- Added notes on how to reinstall the Browser Only version
- Updated Screen Shots for degrees F display and Browser Only version
- Added hardware design information regarding 16 Channel relay boards

April 13, 2021 Code Revision 20210413 1254

Code changes:

- Fixed issue where all temperature sensors were not appearing

May 9, 2021 Code Revision 20210509 2031

Code changes:

- Fixed Issue #55: Home Assistant; Auto Discovery; MQTT; Temp Sensors do not get deleted and redefined properly when added/removed
- Fixed Issue #56: Temperature Sensor accuracy improvement
- Fixed Issue #57: Temperature sensor order in GUI and Home Assistant. Fixed by changing Temperature Sensor ID's from "1, 2, 3, 4, 5" to a 12 character ID per sensor that is based on the sensor serial number
- Fixed Issue #58: Home Assistant; Auto Discovery; MQTT; Should a temperature sensor be deleted in HA if it fails during runtime?

- Fixed Issue #62: Browser Only; Timers don't work with Invert setting. Included in fix is a correction for "Timer values not saved correctly" and "Timer value being set to 256 in error".
- Fixed Issue #63: REST commands preventing IO Control page from making pin state changes.
- Fixed Issue #66: MQTT doesn't report temperature sensors unless at least 1 IO is enabled.

Document changes:

- Updated description of Temperature Sensor ID's and run time add/delete functionality.
- Added additional explanations on how Temperature Sensor ID's are sorted for display.

May 11, 2021 Code Revision 20210511 1317

Code changes:

- Fixed Issue #67: Change port state using SAVE button not working on IO Control page.

August 21, 2021 Code Revision 20210821 1541

Code changes:

- Addressed Issue #71: Extensive code additions to support Issue #71 "Firmware Update via webinterface", aka "Upgradeable" firmware.
- Addressed Issue #77: Fixed a potential math error in how pin timers are calculated. The error only affects the "hours" timeout value in the Browser Only code.
- Addressed Issue #78: Fixed a bug in calculation of the Configuration page size. This only affects MQTT builds. The symptom is the Configuration page may not completely paint or you may not be able to save changes.
- Addressed Issue #79: Fixed a problem where rapid use of REST relay commands (/00 /01 etc) may interfere with Browser updates. This fix required changing the operating characteristics in that use of the REST relay commands, if executed from a Browser, will no longer repaint the GUI, instead only showing a blank screen.
- Addressed Issue #80: Fixed a problem where the REST /98 or /99 command would interfere with painting the IOControl and Configuration pages.
- Addressed Issue #81: Significant improvement to "multi-browser" operation.
- Addressed Issue #82: Fixed bug in DS18B20 CRC check.
- Addressed Issue #83: /98 /99 command does not return the correct size for the page in the HTML header.

Document changes:

- Added new section on using "Upgradeable" firmware
- Added REST command notes
- Added Developers section: Notes on Proto-Sockets
- Added Developers section: #pragma, Sections, Segments and the .lkf File. Includes information on Editing the .lkf file, Flash Update and Copy-RAM-to-Flash Code, _fctcpy, #pragma Sections, and Content of the .lkf File
- Added Developers section: STM8 Address Map
- Added Developers section: Flash Memory Map
- Added Developers section: I2C EEPROM Memory Map, Includes information on I2C EEPROM Regions for Upgradeable Builds and Addressing the I2C EEPROM Regions
- Added Developers section: Strings File Generation

February 5, 2022 Code Revision 20220205 1645

Code changes:

- Addressed Issue #91: Major problem: Clicking Save in IOControl causes loss of Configuration settings

- Fixed bug where the check for pin state changes would run even when there were no changes. No functionality change but should free up some processor bandwidth.
- Added a "code type" identifier to the Revision text on the Configuration page.

Document changes:

- Added Developers section: "How Connections Open and Close And Relationship to the "current_page" Variable"
- Added Developers section: "How Proto-Sockets are Implemented and Connections Established"
- Addressed Issue #88: REST Command IO State Output. Corrected the document to reflect actual order of the pin states displayed in the short form display.
- Added a Functional Limitation note with regard to multiple Browser tabs related to Issue #87.
- Added section "16 Channel Relay Moduless" describing hardware modifications to make 16 Channel Relay boards compatible with the Network Module hardware.
- Added additional information to the "Developers: Using the UART" section.

August 17, 2022 Code Revision 20220817 1508

Code changes:

- Addressed Issue #95: "MQTT state-req returning all zeroes"

Document changes:

- Updated Change Log.

August 31, 2022 Code Revision 20220831 2011

Code changes:

- Addressed Issue #101 enhancement request: "Make ON/OFF labels clickable"

Document changes:

- Updated Change Log.

MQTT Firmware vs Browser Only Firmware

Two firmware types are available for the Network Module: “MQTT” and “Browser Only”.

The intent of having the two firmware builds is to address two different communities of users.

The **MQTT** firmware is directed at users that have the network infrastructure to support a MQTT Broker and device management tools like Home Assistant, NodeRed, and similar tools. Those added network tools provide a very diverse set of device management capabilities including user friendly interfaces and timing control. The MQTT firmware CAN also be used by someone that only wants a Browser GUI.

The **Browser Only** firmware is directed at users that want simpler, self-contained functionality without the need for an MQTT broker and other “always on” network tools. The Browser Only firmware eliminates the MQTT code and provides a GUI that includes a few extra user interface features like “IO Naming” and “IO Timers”. Of course “Browser Only” significantly limits automation relative to what you can do with MQTT and the device management tools associated with MQTT.

Feature Comparison:

Feature	MQTT Build	Browser Only Build	MQTT Upgradeable Build	Browser Only Upgradeable Build
MQTT Support	X		X	
Home Assistant Support	X		X	
Full/Half Duplex	X	X	X	X
Link Error Statistics	X	X	X	X
DS18B20 Temp Sensor	X	X	X	X
IO Naming		X		X
IO Timers		X		X
Link Error Statistics	X	X	X	X
Network Statistics		X		X
I2C Support			X	X
Upgradeable Over Ethernet			X	X

Upgradeable Firmware

REQUIRES ADDITIONAL HARDWARE

"Regular" firmware can only be installed on the Network Module using an ST-Link V2 and the SWIM interface (see section "Programming the Module"). Then, for upgrades you must attach the ST-Link V2 again and reprogram the device. However, by adding an external EEPROM you can take advantage of "Upgradeable" firmware that enables you to update the firmware on the Network Module via Ethernet. You still must program the Network Module at least one time via the SWIM interface, but thereafter you can do all code upgrades via Ethernet. Both the MQTT and Browser Only versions of the firmware are available in Upgradeable form.

Advantages:

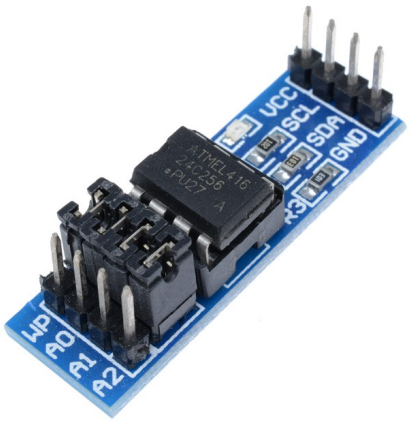
- No need to physically access the Network Module to update the firmware. Reprogramming is fairly fast, taking only about 1 minute.
- You can switch between MQTT and Browser Only versions as needed.
- More Flash space is made available for future features.
- An I2C interface is implemented to support the feature, enabling addition of other I2C devices.

Disadvantages:

- IO pins 14 and 15 are required for use as an I2C interface
- **You must add an external 1MBit (128Kbyte) I2C EEPROM.**

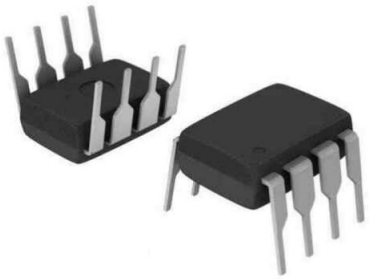
What hardware is needed?

You can design your own hardware to add to the Network Module, or you can buy the following:



Search eBay or other sites with the term “Serial I2C Interface EEPROM”. These are typically sold WITH a 32KByte (256Kbit) EEPROM already on them – but we need a larger capacity EEPROM. So, you can get the “board only” without an EEPROM for as little as $1 USD.

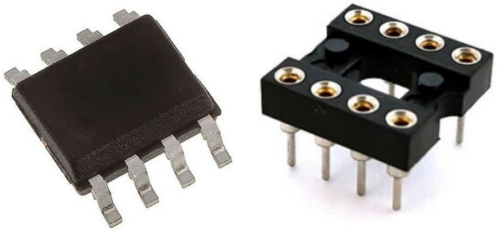
The EEPROM needed is a 24AA1025 or 24LC1025 or 24FC1025 in DIP form.



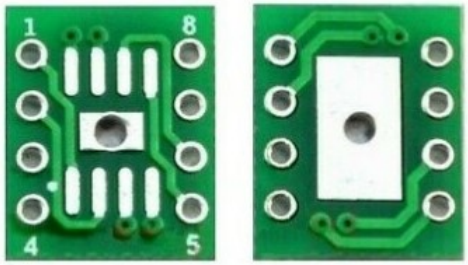
The 24xx1025 is a 128Kbyte (1Mbit) device. It is plug compatible with the board shown above.

The 24xx1025 can be hard to find in DIP form. Single devices are typically around $7 USD. If multiples are bought the price is much lower per device. I bought 10 for $1.25 each. Search on all three types (AA, LC, FC) as sometimes one of the alternatives is cheaper.

One user bought the TSSOP/MSOP SMT/SMD style of package, soldered it to a machined DIP socket, then plugged that into the socket on the board. I don’t recommend this approach due to its fragility, but it will work in a pinch if you are good with kluging.



You can also consider searching eBay or other sites for SO to 8 pin DIP converter boards. I found these under the heading “SO/SOP/SOIC 8 to DIP8 Adapter Converter”.



Connecting to the Network Module:

The “VCC” pin on the EEPROM board must be attached to 3.3V on the Network Module. 3.3V is accessible on the SWIM connector, or you can add a wire.

The “SCL” pin on the EEPROM board must be attached to IO 14 on the Network Module.

The “SDA” pin on the EEPROM board must be attached to IO 15 on the Network Module.

The “GND” pin on the EEPROM board must be attached to GND on the Network Module.

IMPORTANT: The A2 jumper on the EEPROM board needs to be moved to the outer position (tied to 3.3V) . The WP, A0, and A1 jumpers remain as shown in the photo above (all tied to GND).

How does it work?

Due to the small amount of code space on the Network Module the “upgradeable firmware process” works as follows:

a) A “Code Uploader” is copied from the added “Off-Board EEPROM” into the STM8 Flash, replacing the program that was in the Flash.
b) The Code Uploader is then used to install and start new firmware.

I left out some details in the above for simplification. So let me expand on the process. The first question you may ask is “How did the Code Uploader get into the Off-Board EEPROM”? Well, whenever the Code Uploader runs the first thing it does is copy itself to a special region in the Off-Board EEPROM. So, the first time this process is ever used the user must use the SWIM interface to install the Code Uploader in the Network Module Flash, and then boot the Network Module. The Code Uploader copies itself to the EEPROM, and it is then available for future use.

Another detail: The “upgradeable” versions of the MQTT and Browser Only firmware needed more space for runtime code, so the “strings” used to define the IOControl and Configuration pages are also stored in the Off-Board EEPROM. The Code Uploader is used to copy those strings to the Off-Board EEPROM.

So, knowing all this, here is the procedure the FIRST time you install the upgradeable firmware:

a) Use the SWIM interface to install the Code Uploader in the Network Module.
b) Boot the Network Module to start the Code Uploader and access the Network Module with your Browser
c) Use the Code Uploader GUI to select the “Strings” file and upload it.

d) Use the Code Uploader GUI to select the upgradeable version of the MQTT or Browser Only runtime code and install it.

You are up and running.

Future upgrades:

After initial install as described above here's how you upgrade firmware in the future:

a) A release will always contain these files:
   - Code Uploader (example name: NetworkModule-CodeUploader.sx)
   - Strings file (example name: NetworkModule-Strings.sx)
   - Runtime files: (example names: NetworkModule-MQTT-UPG.sx and NetworkModule-Browser-UPG.sx)

   When using "upgradeable" code it is important to ONLY load the MQTT and Browser Only >>upgradeable<< versions or you will break the upgrade loop and have to go back to the "first time" steps.

b) Since you are upgrading you already have a "runtime" version installed. Enter the URL command /72. This will install the Code Uploader from the Off-Board EEPROM.

c) Use the Code Uploader to install the new Code Uploader. Why do this? Often there won't be any change to the Code Uploader, but just to keep the process "familiar" I will always release all files at the same time.

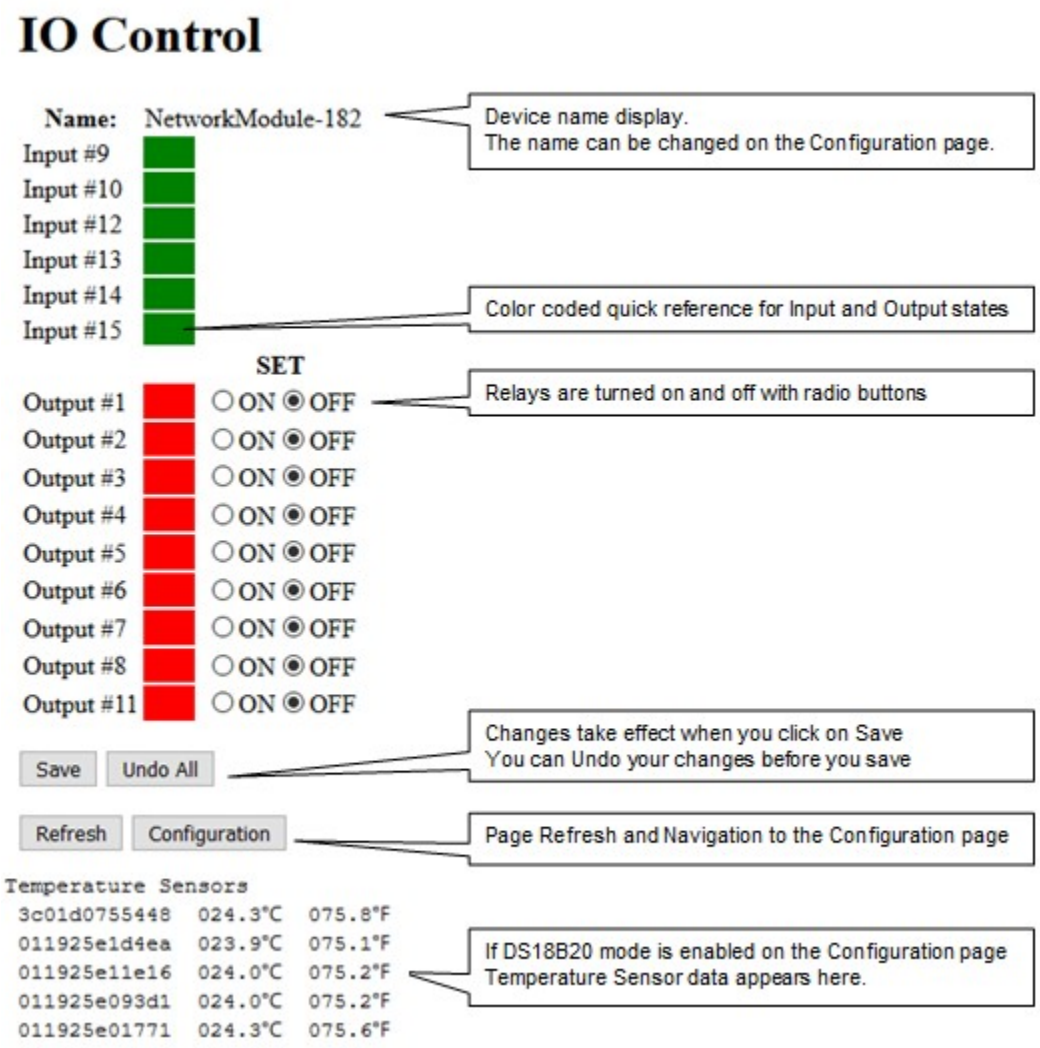
d) Use the Code Uploader to install the new Strings file. Again this file usually doesn't change, but installing it every time prevents procedural mistakes.

e) Use the Code Uploader to install the new Runtime file of your choice.

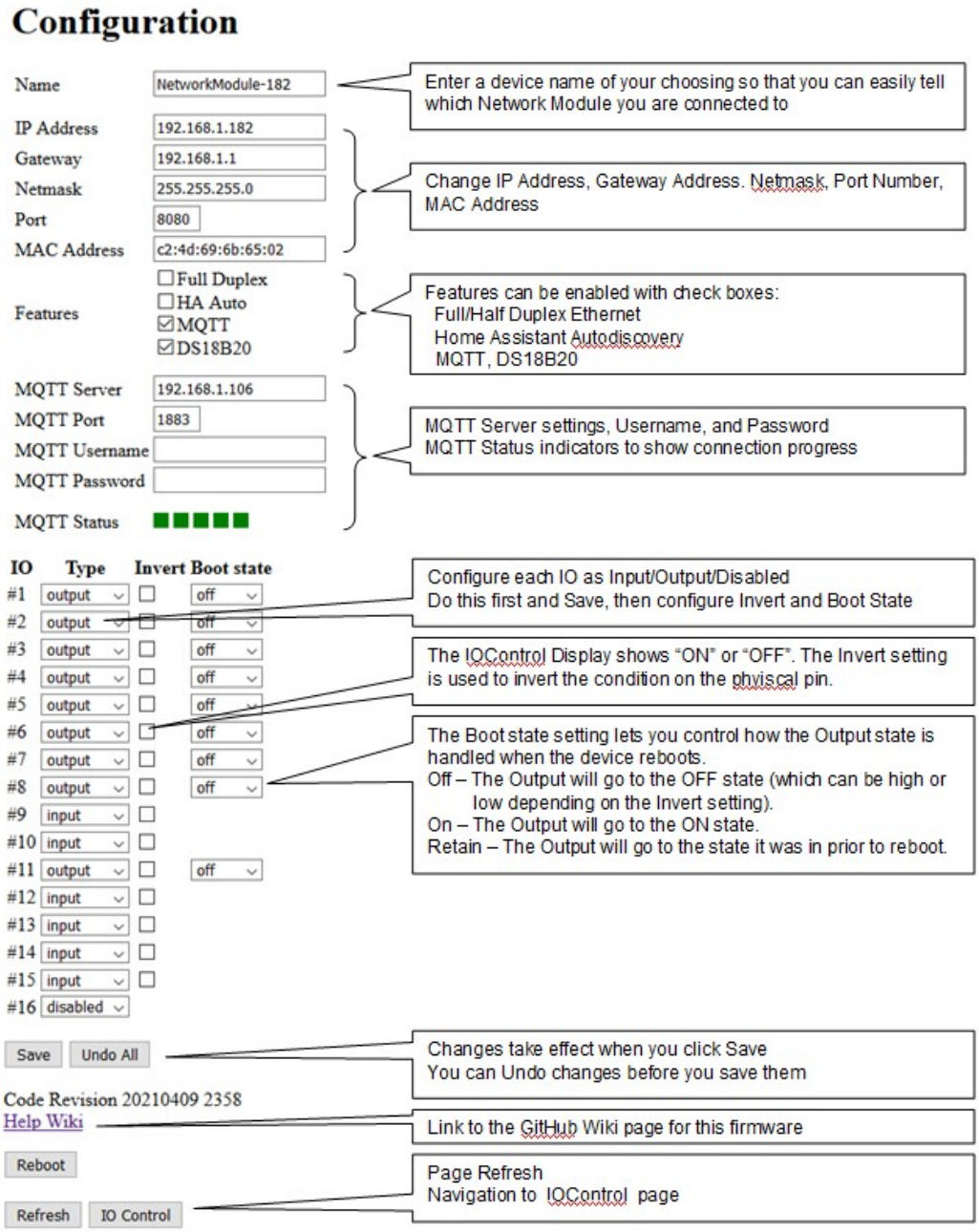
Note: If you are certain you already installed the new Code Uploader and String files you can use the Code Uploader (via the /72 URL command) to install the MQTT and Browser versions as needed. This lets you move back and forth between the code types for development work.

That's pretty much it. The Code Upload process includes some pop-up screens to help you with progress and timing of the steps. But if you are new to the Network Module it may be best to just use the non-upgradeable code and the ST-Link V2 / SWIM interface programming process until you get familiar.

Screen Shots and Usage - MQTT Build



Screen Shots and Usage
MQTT Build

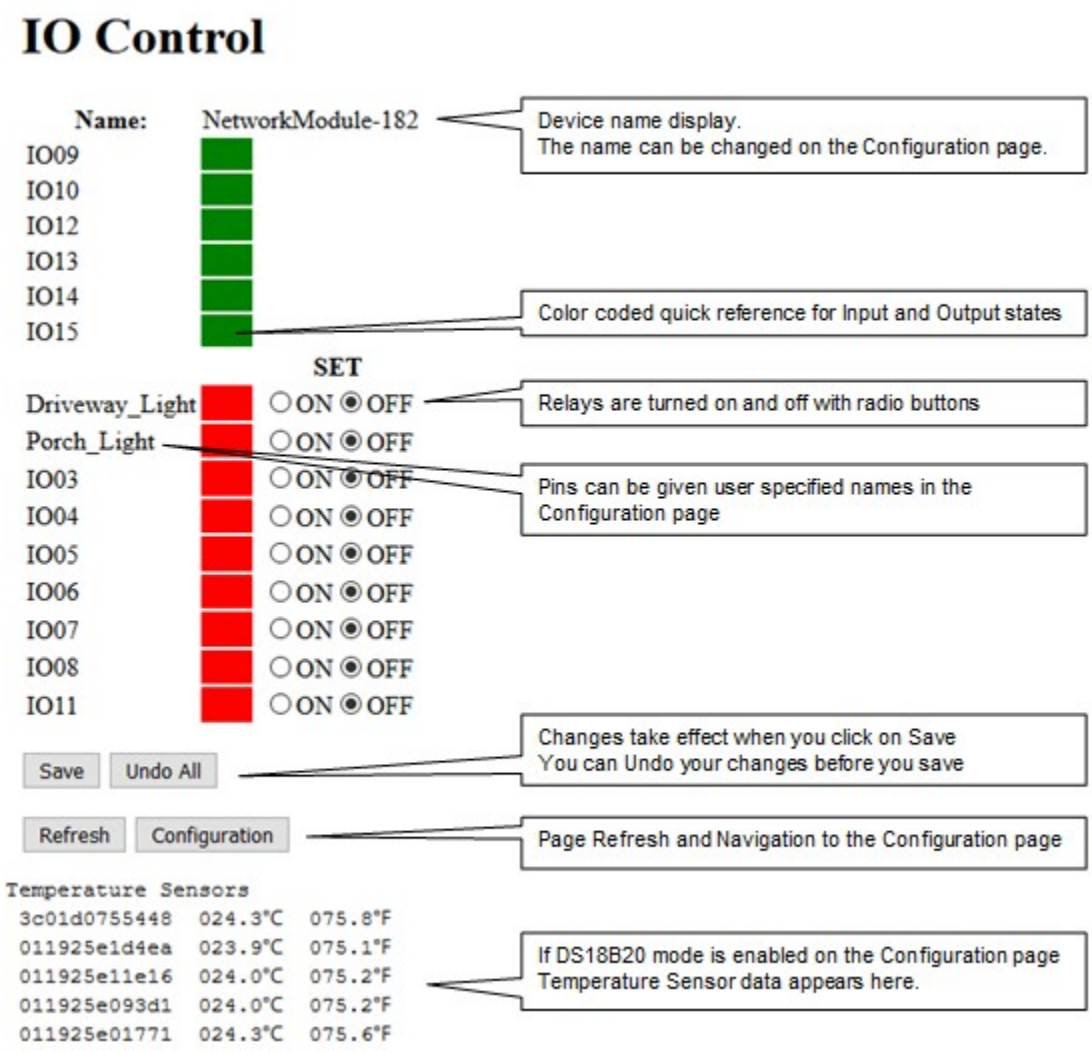


Use caution when changing Configuration settings. If you make a mistake and can no longer access the device you may have to restore factory defaults by holding down the reset button for 10 seconds.

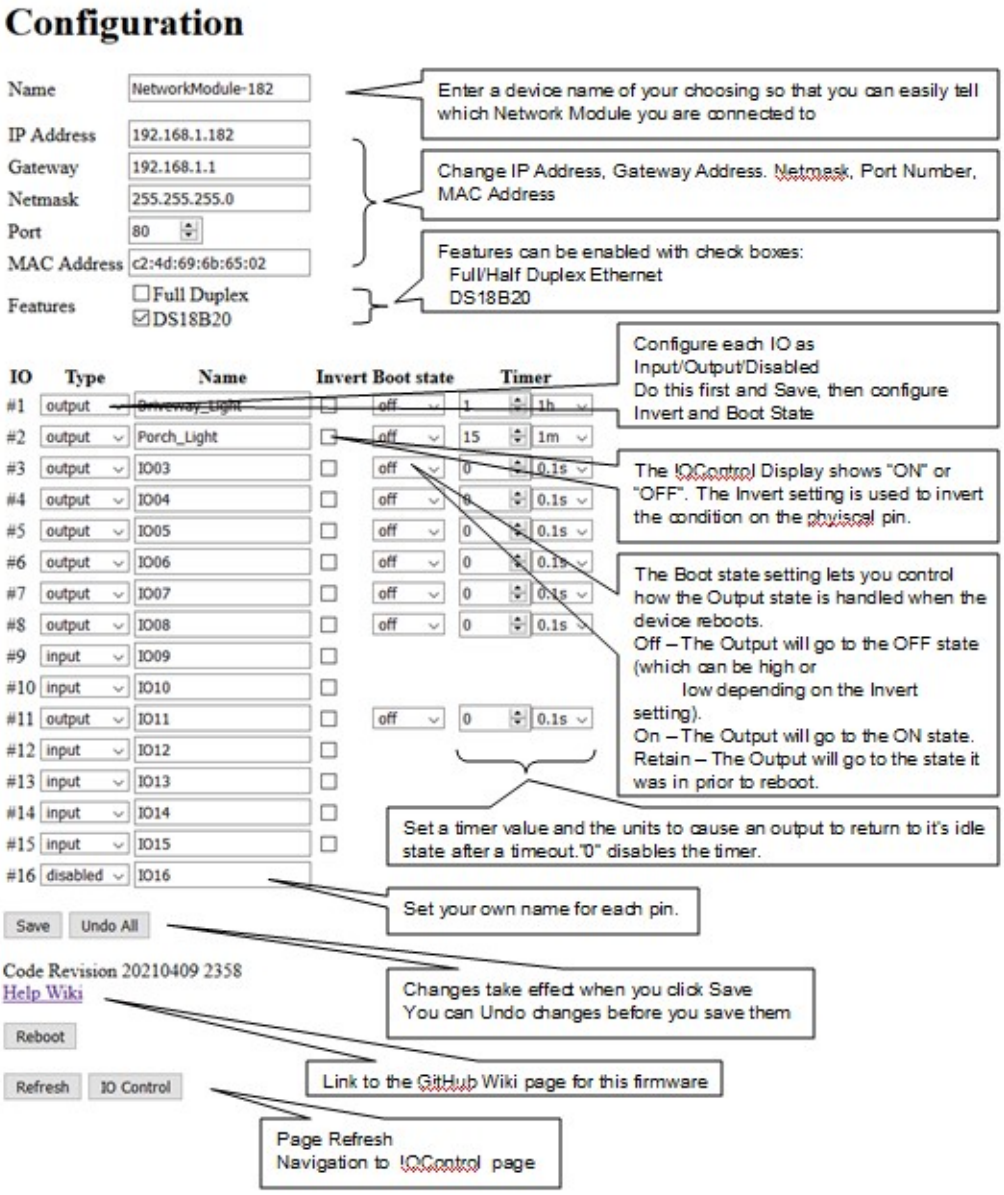
Make sure the MAC you assign is unique to your local network. Recommended is that you just increment the lowest octet and then label your devices for future reference.

If you change the highest octet of the MAC you MUST use an even number to form a unicast address. 00, 02, ... fc, fe, etc work fine. 01, 03 ... fd, ff are for multicast and will not work.

Screen Shots and Usage - Browser Only Build



Screen Shots and Usage
Browser Only Build



Use caution when changing Configuration settings. If you make a mistake and can no longer access the device you may have to restore factory defaults by holding down the reset button for 10 seconds.

Make sure the MAC you assign is unique to your local network. Recommended is that you just increment the lowest octet and then label your devices for future reference.

If you change the highest octet of the MAC you MUST use an even number to form a unicast address. 00, 02, ... fc, fe, etc work fine. 01, 03 ... fd, ff are for multicast and will not work.

Screen Shots and Usage
Browser Only Build

To access the Network Statistics Page enter the http command “http://IP:Port/68”.

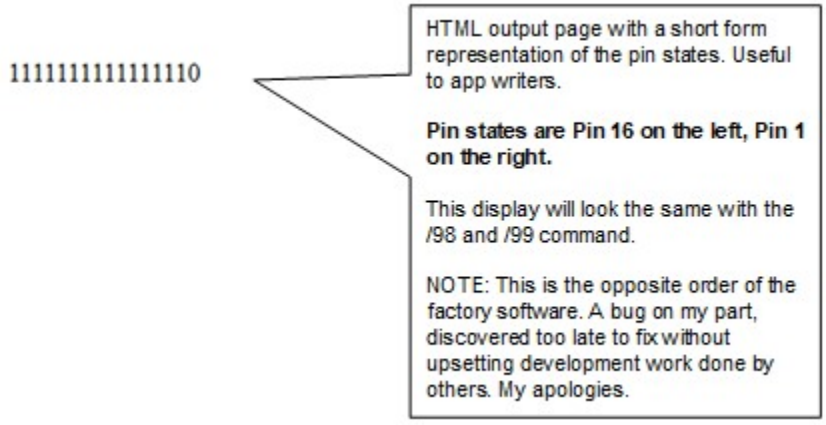
Network Statistics

Values shown are since last power on or reset

0000282271	Dropped packets at the IP layer
0000282784	Received packets at the IP layer
0000000476	Sent packets at the IP layer
0000000000	Packets dropped due to wrong IP version or header length
0000000000	Packets dropped due to wrong IP length, high byte
0000000000	Packets dropped due to wrong IP length, low byte
0000000000	Packets dropped since they were IP fragments
0000000000	Packets dropped due to IP checksum errors
0000000000	Packets dropped since they were not ICMP or TCP
0000000000	Dropped ICMP packets
0000000000	Received ICMP packets
0000000000	Sent ICMP packets
0000000000	ICMP packets with a wrong type
0000000000	Dropped TCP segments
0000000518	Received TCP segments
0000000483	Sent TCP segments
0000000000	TCP segments with a bad checksum
0000000000	TCP segments with a bad ACK number
0000000000	Received TCP RST (reset) segments
0000000012	Retransmitted TCP segments
0000000000	Dropped SYNs due to too few connections avaliable
0000000000	SYNs for closed ports, triggering a RST

Configuration | Refresh | Clear

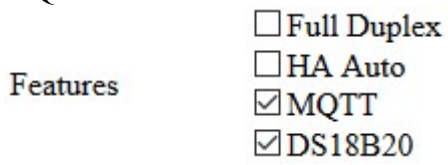
Screen Shots and Usage - All Builds



Notes on Feature Settings

The Features checkboxes on the Configuration page let you modify operation of the code as follows:

MQTT build:



Browser Only build:

Full Duplex Setting:

The Full Duplex checkbox determines the Half / Full Duplex Ethernet communication method. **The default setting is Half Duplex because that is the most reliable setting for the ENC28J60 Ethernet chip.** Since all the Ethernet transactions that will occur with the module are small and infrequent there is no real performance advantage to using Full Duplex.

During test it was found that Cisco business level switches exhibited Half Duplex timing that the ENC28J60 cannot handle, the symptom being a device disconnect (and automatic recovery) every few hours. This isn't the fault of the Cisco switch, rather it appears to be the fault of errata in the ENC28J60. During test it was determined that we could get around this issue when connected to the Cisco switch by enabling Full Duplex mode in the ENC28J60. While there was concern that this would not work (due to chip spec notes and online discussion over the years), it seemed to run error free. There may be other switches which show the same issue. Again, the reason Full Duplex works may be the very low messaging rate used with the Network Module which eliminates the need for flow control.

Note 1: The spec for the chip indicates that Full/Half Duplex auto-negotiation DOES NOT work. However, experimentation showed that both Full and Half Duplex worked with some unmanaged switches, but not with others. Problems were always running Half Duplex only with the Cisco 1G managed switch. No problem was seen running Half Duplex with a Cisco 10/100 managed switch.

Note 2: If you choose to use the Full Duplex setting note that the spec says the Switch port the device is connected to MUST be manually configured for Full Duplex operation … even though our testing did not always show that to be the case. Of course we had a limited number of switches and this might be an issue on some other switch.

Note3: Feel free to experiment with this setting at your own risk to see what works best in your network configuration. I recommend you use Half Duplex and only try Full Duplex if you have issues.

HA Auto:

The Home Assistant Auto Discovery setting enables the Network Module to send MQTT Auto Discovery Publish messages to your Home Assistant server. Checking this setting will automatically enable MQTT (and the MQTT checkbox will automatically be set). Do not enable this setting unless you are operating in an MQTT environment with Home Assistant.

MQTT:

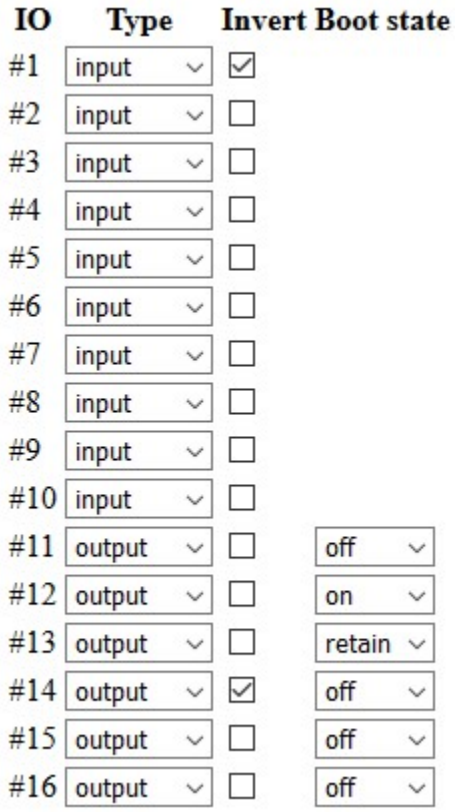
Checking this box will enable the MQTT interface. HA Auto DOES NOT need to be enabled with MQTT. This will allow you to operate with MQTT servers without using Home Assistant, OR it will enable you to use Home Assistant without Auto Discovery.

DS18B20:

Checking this box will cause IO 16 (Pin 16) to be disabled for use as an Input / Output pin and will enable operation of the DS18B20 Temperature Sensor interface on IO 16 (Pin 16). You can attach up to 5 DS18B20 temperature sensors to pin 16, and the temperatures sensed by those devices will be displayed on the IOControl page. See the section “Adding DS18B20 Temperature Sensors”.

Notes on Individual IO Settings – MQTT build

The Individual IO Settings drop down and check boxes on the Configuration page lets you modify functionality of each IO as follows:



Type Drop Down Settings:

Each IO has a "Type" drop down box that lets you configure each IO as follows:

Input – Self explanatory. Sets the IO as an input.
Output – Self explanatory. Sets the IO as an output.
Disabled – In Browser applications a Disabled Input or Output will not appear in the IO Control page. REST commands will also not affect a Disabled Output. In Home Assistant applications a Disabled Input or Output will result in a Config message with an empty payload, resulting in that IO being deleted from the Home Assistant configuration.

IMPORTANT: Set the Input/Output/Disabled setting, THEN Save, THEN make other setting changes.

VERY IMPORTANT: Be sure you understand your hardware design. You must avoid setting an IO as an Output if the associated pin is tied to VCC or Ground, as that is likely to damage the output driver on the processor. If your hardware design can provide high levels of input current on a pin make sure that pin is defined as an Input.

Invert Settings:

Each IO has an "Invert" checkbox.

Effect on Inputs:
If not checked, a low voltage on the Input pin will display as OFF in the IOControl page and will be reported as OFF to MQTT clients.
If checked, a low voltage in the Input pin will display as ON in the IOControl page and will be reported as ON to MQTT clients.

Effect on Outputs:
If not checked, an OFF indication in the IOControl page or MQTT Client will result in a low voltage on the Output pin.
If checked, an OFF indication in the IOControl page or MQTT Client will result in a high voltage on the Output pin.

Since some devices connected to Output pins may be in an ON state with a low voltage, and others may be in an OFF state with a low voltage you will need to figure out what the Invert setting should be for your specific design. It is really a simple matter of connecting your peripheral device, setting ON or OFF in the Browser, then checking the Invert box as needed so that an ON state in the Browser matches an ON condition in your peripheral.

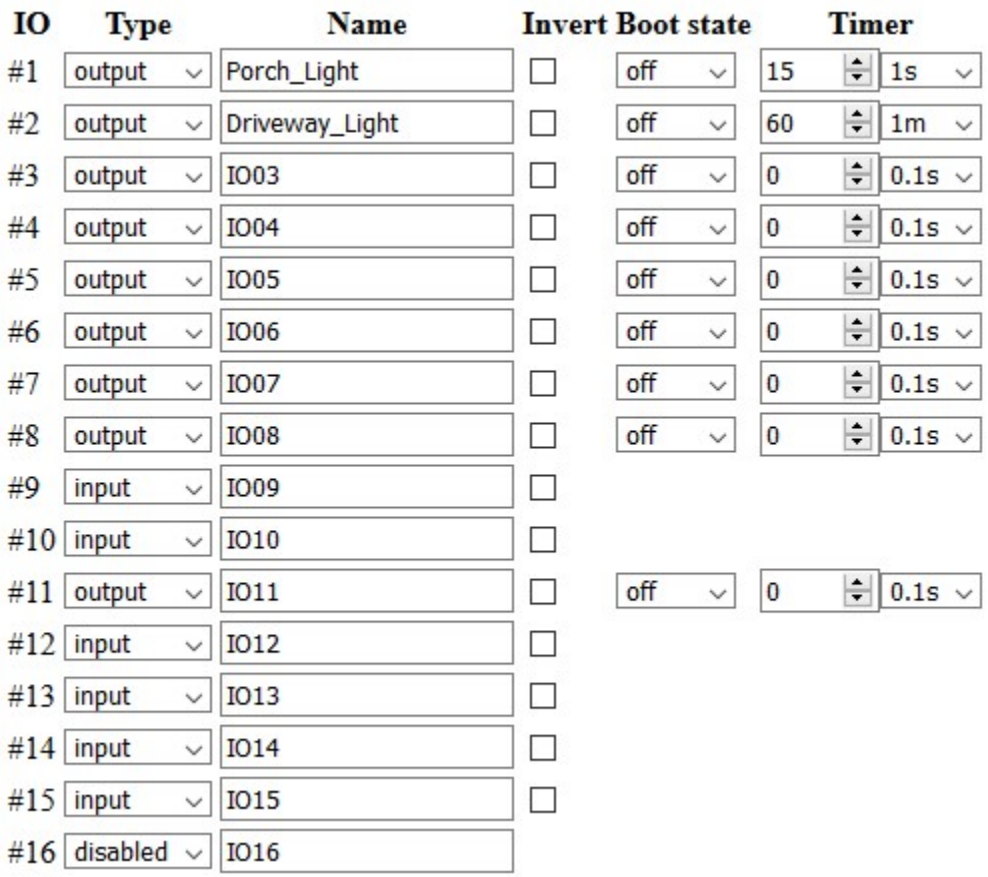
Boot State Settings:

Each IO has a "Boot State" dropdown box. The effect of each setting is as follows:

Off: After boot the state of the Output is OFF.
On: After boot the state of the Output is ON.
Retain: After boot the state of the Output is the same as it was before boot.
IMPORTANT: Only use "Retain" if the output changes infrequently.

Notes on Individual IO Settings – Browser Only build

The Individual IO Settings drop down and check boxes on the Configuration page lets you modify functionality of each IO as follows:



Type Drop Down Settings: Same as MQTT build.

Invert Settings: Same as MQTT build.

Boot State Settings: Same as MQTT build.

Name Settings:

Enter a name for the IO pin. This name will appear on the IOControl page for the pin. Use any alphanumeric character plus -*_ and . (no spaces).

Timer Settings:

Two fields are provided:

"Value" field:

0 disables the Timer.

1 to 16383 can be entered for the of "ticks" to operate the Timer. For example, 1 second or 1 minute or 1 hour.

"Units" field:

Select the time unit to use: 0.1 second, seconds, minutes, or hours.

How the Timers work:

Timers only work on Output pins.

If you enter "0" in the Value field the Timer will not affect the Output pin.

The Timer assumes that the "Boot State" is the idle state of the output, ie, the "normal" state the output is in. If you then change the output to its non-idle state the Timer will start and will return the output to the idle state when the Timer expires. Example:

Output #1 is named "Porch Light". The Boot State is "off". So, normally this output is "off". I have the Timer value and units set to 15 seconds. If I change the output to "on" the Timer will turn the output off after 15 seconds.

"Boot State" must be "on" or "off". "retain" will disable the Timer.

If a Reboot or power cycle occurs while a Timer is running the Output will be set to its "Boot State".

If you change the IO Timer value while the Timer is running:

- The Timer is reloaded with the new IO Timer value. The Timer will continue running and will expire at the new value entered.
- If you entered "0" as the new value the Timer will expire immediately.

NOTE: "Name" and "IO Timer" features are not available in the MQTT build due to lack of Flash space. Typically this type of feature isn't needed in MQTT environments as MQTT management tools provide similar functionality.

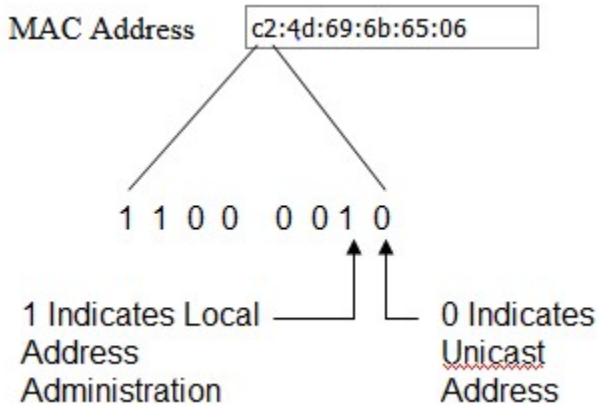
IMPORTANT: Don't change the IO Names and IO Timers settings frequently. These values are stored in the Flash (not in the EEPROM). Flash has a much more limited number of write cycles than EEPROM, so try to keep the number of times you change IO Name or IO Timer settings to a few hundred per pin. This should be more than adequate for typical usage.

Notes on the MAC Address

When delivered from the factory the Network Modules all have the same MAC address. This obviously doesn't work when you try to put more than one on a network.

A MAC address is only used within your network. Your router(s) and switch(es) use the MAC address as the means of uniquely addressing all the hardware in your network. The MAC address does not appear outside your network so it only needs to be unique to YOUR network, not to the entire world. This being the case, you only need to make sure that any MAC address you put in the Network Module does not conflict with any other hardware in your local network.

The default MAC address value in the code provided is just a random value with the exception that it has the two least significant bits of the most significant octet arranged to make it a "Unicast" and "Locally Administered Address (LAA)" as illustrated here. You MUST make sure you use a LAA and Unicast address.



All other bits and octets in the MAC address (including those in the most significant octet) can be anything you want as long as you set the two bits above as shown.

Despite this being a LAA MAC address there is still some very remote possibility the MAC you pick will conflict with some other hardware you have on your network. You can search on Google to find methods of finding all MAC addresses on your network – the method you choose will depend on your level of expertise. Generally this is not required, and if you suspect a conflict you may just find it easier to try a different MAC address on the Network Module. Maybe make the middle fours octets something you fancy.

A good reference for MAC address explanations is here:
https://en.wikipedia.org/wiki/MAC_address

If you are installing multiple devices on your network I suggest that you just change the values in the least significant octet and leave the others as-is. I advise you add a label to your Network Module with the MAC you programmed into it.

Notes on REST Commands

A REST (Representational State Transfer) type of interface has been implemented to enable access to Input/Output states and other functions without the use of the browser. This is to enable development of external programs to operate the Network Module without use of the full GUI. A complete list of the REST commands is provided here.

For all commands enter an http request as follows:

http://IP:Port/xx

where

- IP = the Network Module IP Address, for example 192.168.1.4
- Port = the Network Module Port number, for example 8080 (Port number may be omitted if the device is set to Port 80)
- xx = one of the codes below

00 = IO 1 OFF	08 = IO 5 OFF	16 = IO 9 OFF	24 = IO 13 OFF
01 = IO 1 ON	09 = IO 5 ON	17 = IO 9 ON	25 = IO 13 ON
02 = IO 2 OFF	10 = IO 6 OFF	18 = IO 10 OFF	26 = IO 14 OFF
03 = IO 2 ON	11 = IO 6 ON	19 = IO 10 ON	27 = IO 14 ON
04 = IO 3 OFF	12 = IO 7 OFF	20 = IO 11 OFF	28 = IO 15 OFF
05 = IO 3 ON	13 = IO 7 ON	21 = IO 11 ON	29 = IO 15 ON
06 = IO 4 OFF	14 = IO 8 OFF	22 = IO 12 OFF	30 = IO 16 OFF
07 = IO 4 ON	15 = IO 8 ON	23 = IO 12 ON	31 = IO 16 ON

55 = All Outputs ON
56 = All Outputs OFF

60 = Show IOControl Page
61 = Show Configuration Page
65 = Flash LED
66 = Show Link Error Statistics page
67 = Clear Link Error Statistics
68 = Show Network Statistics page (Browser Only builds)
69 = Clear Network Statistics and refresh page (Browser Only builds)
70 = Clear the "Reset Status Register" counters
71 = Display the Temperature Sensor Serial Numbers (Only in special debug builds)
72 = Load Code Uploader (Only in Upgradeable builds)
73 = Reload firmware existing image (Only in the Code Uploader)
74 = Erase entire Off-Board EEPROM (Only in the Code Uploader)
91 = Reboot
98 = Show Short Form IO Settings (without HTML formatting)
99 = Same as /98

Note1: Output control commands (00 to 31) only work for IO defined as an Output. If the IO referenced in the command is an Input no action is taken.

Note 2: Commands 98 and 99 provide the IO states as 16 alphanumeric characters WITHOUT any HTML formatting. Both commands produce the same result for backward compatibility. This command may be useful to some external applications that automate interaction with the Network Module using URL style commands.

Note 3: Releases prior to August 2021 would repaint the GUI with the "current webpage" when /00 to /31, /55, or /56 commands were executed. This was found to interfere with operation of other Browsers logged into the Network Module. Beginning with August 2021 releases these commands will return a "blank" webpage to eliminate this interference.

Note 4: VERY IMPORTANT - Since it is common to implement some automation with REST commands while still wanting to use a Browser to make configuration changes the following should be noted:

REST automation should only use these commands:

/00 to /31 Output state change commands
/55 All outputs ON
/56 All outputs OFF
/91 Reboot
/98 Show Short Form IO State page
/99 Show Short Form IO State page

Use of any other commands will likely interfere with Browser operation in the IOControl and Configuration pages.

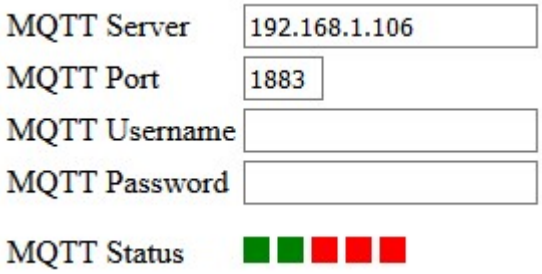
If you choose to use some of the other commands you may need to suspend REST automation while a Browser is used to access the IOControl or Configuration pages.

Notes on MQTT

This is not a tutorial on MQTT as there are a lot of great resources online to bring you up to speed if you are just getting started with this protocol. Here I am only including notes on the tools and methods used in test of the MQTT functionality on the Network Module.

Configuration Settings:

First a discussion of the Configuration page parameters associated with MQTT



At this date the MQTT Broker Server must be specified in the form of an IP Address (as opposed to a URL). In future versions I may be able to allow use of a URL for the Broker Server.

The MQTT Port is self explanatory. The MQTT default value of 1883 automatically appears, but you can enter any port number you have assigned to MQTT on your Broker Server.

Security:

The MQTT Username and MQTT Password are optional and only required if you've set up your Broker to require them.

IMPORTANT: SSL/TSL are NOT implemented due to memory restrictions. I haven't found any implementations that are small enough to fit in the flash space available on the Network Module, so I don't expect this to ever be possible.

If security is really important due to the need to access MQTT on the device from the internet I suggest setting up access to your internal MQTT Broker Server in a secure way, then letting the Broker Server pass all messages on your internal network. You don't have to do it this way, but this is a suggestion for improving security rather than just exposing the device to the internet.

Tools and test methods:

Tools used in development and test of the MQTT functionality:

- The Mosquitto Broker was used on a Windows 10 laptop and worked very well with the Network Module.
- Chrome with MQTTLens was used at various points to provide a manual MQTT subscribe and publish interface.
- NodeRed was used to drive automated MQTT messages.
- Carlos Ladeira used Home Assistant in concert with NodeRed and the Mosquitto Broker in a Linux environment to perform extensive long run testing.

You aren't restricted to the above. Any tools and interfaces that are MQTT compliant should work just as well. But if you are just getting started I can recommend the above as a good place to start.

MQTT Status Indicators:

The MQTT Status indicators show connection progress with the MQTT Server and Broker:
Box 1 - Indicates that the MQTT Connection process has started.
Box 2 - Indicates a successful ARP reply from the Server.
Box 3 - Indicates a successful TCP connection with the Server.
Box 4 - Indicates the MQTT Broker has responded (Connect phase)
Box 5 - Indicates initial communication with the Broker has completed successfully (initial subscribe and publish messages completed).

Once all 5 boxes are green the Network Module is connected to the Broker and normal MQTT communications can proceed.

Reminder regarding the Output "Boot State" setting:

A reminder regarding the Output Boot State setting for the Outputs on the Configuration page: If you select 'Retain' the Output states are written to the EEPROM every time an Output changes state. If you anticipate a lot of Output state changes you may wear out the EEPROM with too many changes to the Output states.

Client Publish and Subscribe messaging:

The Client **Publishes** the following messages to control outputs:

NetworkModule/<devicename>/output/xx/set Payload: "ON" or "OFF"
NetworkModule/<devicename>/output/all/set Payload: "ON" or "OFF"
NetworkModule/<devicename>/state-req Payload: none
Where “xx” is the IO number of the Output

The Client **Subscribes** to the following:

NetworkModule/<devicename>/availability
This Subscribe enables the Client to receive the Network Module online / offline messages.
NetworkModule/<devicename>/input/+
NetworkModule/<devicename>/output/+
These two Subscribes enable the Client to receive changes in Input and Output states. The "+" causes the broker to send the client the IO state messages without reflecting the client's own "set" commands back to the client (reduces traffic).
NetworkModule/<devicename>/temp/+
This allows the client to receive the Temperature Sensor data produced by the Network Module if DS18B20 mode is enabled and Temperature Sensors are attached to the Network Module.
NetworkModule/<devicename>/state
This allows the client to receive the responses to the state-req Publish commands that the Client sends.

The Network Module **Publishes** the following when an IO state change occurs:

NetworkModule/<devicename>/input/xx Payload: "ON" or "OFF"
NetworkModule/<devicename>/output/xx Payload: "ON" or "OFF"
Where “xx” is the IO number of the Input or Output

The Network Module **Publishes** the following every 30 seconds if DS18B20 mode is enabled and Temperature Sensors are attached to the Network Module::

NetworkModule/<devicename>/temp/xxxxxxxxxxxx Payload: Temperature in degrees Celsius in the format " 000.0" or “-000.0”
Where “xxxxxxxxxxxx” is the Temperature Sensor ID in 12 hex encoded characters.

The Network Module **Publishes** the following in response to receiving a state-req Publish message:

NetworkModule/<devicename>/state Payload: see below

The payload consists of two bytes with the bits organized as follows:

First byte:

Bit 7 = IO 16 state (1 = ON, 0 = OFF)
Bit 6 = IO 15 state
Bit 5 = IO 14 state
Bit 4 = IO 13 state
Bit 3 = IO 12 state
Bit 2 = IO 11 state
Bit 1 = IO 10 state
Bit 0 = IO 9 state

Second byte:

Bit 7 = IO 8 state
Bit 6 = IO 7 state
Bit 5 = IO 6 state
Bit 4 = IO 5 state
Bit 3 = IO 4 state
Bit 2 = IO 3 state
Bit 1 = IO 2 state
Bit 0 = IO 1 state

Note that Disabled IO will still have either an ON or OFF state – dependent on the last state seen for that IO. The user application is responsible for knowing which IO are Enabled and Disabled when using the above response.

The Network Module **Subscribes** to the following topics when it connects to the Broker:

NetworkModule/<devicename>/output/+/set

This Subscribe enables the Network Module to receive output Publish commands from Clients. The + as used above causes the broker to send the module the "set" commands, but won't reflect the modules own IO state messages back to the module (reduces traffic).

NetworkModule/<devicename>/state-req

This Subscribe enables the Network Module to receive the state-req Publish command from Clients

When the Network Module connects to the Broker it will establish a "last will" message of "offline" with the will topic “NetworkModule/<devicename>/availability”

When the Network Module connects to the Broker it will Publish to the following topics:

NetworkModule/<devicename>/availability Payload: online
NetworkModule/<devicename>/input/xx Payload: "ON" or "OFF"
NetworkModule/<devicename>/output/xx Payload: "ON" or "OFF"

Where “xx” is the IO number for the Input or Output.
No Publish occurs for a Disabled IO.

NetworkModule/<devicename>/temp/xxxxxxxxxxxx Payload: Temperature in degrees Celsius in the format " 000.0" or “-000.0”
Where “xxxxxxxxxxxx” is the Temperature Sensor ID in 12 hex characters

Home Assistant Auto Discovery Publish messaging:

If the HA Auto checkbox is set to enable the Network Module **Publishes** the following messages at boot time:

For each Output:
homeassistant/switch/<macaddress>/xx/config Payload: see below
homeassistant/binary_sensor/<macaddress>/xx/config Payload: empty
The “binary_sensor” message with an empty payload makes sure that Home Assistant will delete any prior Input configuration on this IO.
homeassistant/sensor/<macaddress>/yy/config Payload: empty
This only occurs if IO 16 is defined as an Output. The “sensor” message with an empty payload makes sure that Home Assistant will delete any prior sensor configuration on IO 16 where IO 16 was previously defined for use as a Temperature Sensor connection.

For each Input:
homeassistant/binary_sensor /<macaddress>/xx/config Payload: see below
homeassistant/switch/<macaddress>/xx/config Payload: empty
The “switch” message with an empty payload makes sure that Home Assistant will delete any prior Output configuration on this IO.
homeassistant/sensor/<macaddress>/yy/config Payload: empty
This only occurs if IO 16 is defined as an Input. The “sensor” message with an empty payload makes sure that Home Assistant will delete any prior Temperature Sensor configuration on IO 16.

For each Disabled IO:
homeassistant/binary_sensor /<macaddress>/xx/config Payload: empty
homeassistant/switch/<macaddress>/xx/config Payload: empty
These “empty payload” messages sure that Home Assistant will delete any prior Input and Output configuration on this IO.
homeassistant/sensor/<macaddress>/yy/config Payload: empty
This only occurs if DS18B20 mode is Disabled. The “sensor” message with an empty payload makes sure that Home Assistant will delete any prior Temperature Sensor configuration on IO 16.

For each Temperature Sensor:
homeassistant/sensor/<macaddress>/yyyy/config Payload: see below
This only occurs if DS18B20 mode is Enabled.

In the above topics <macaddress> is the MAC address of the Network Module. The “xx” is the IO number. The “yyyy” is the Temperature Sensor ID. Outputs are defined as “switch” topics, Inputs are defined as “binary_sensor” topics, and Temperature Sensors are defined as “sensor” topics.

Where a Home Assistant Auto Discovery Payload is not empty it takes this form for the “switch” topics:

```
{
"uniq_id":"<macaddress>_output_01",
"name":"<devicename> output 01",
"~":"NetworkModule/<devicename>",
"avty_t":"~/availability",
"stat_t":"~/output/01",
"cmd_t":"~/output/01/set",
"dev":{
"ids":["NetworkModule_<macaddress>"],
"mdl":"HW-584",
"mf":"NetworkModule",
"name":"<devicename>",
"sw":"<code_revision>"
}
}
```

The above example is for an Output on IO 01.
<macaddress> is replaced with the MAC address of the Network Module as entered on the Configuration page..
<devicename> is replaced with the Name of the Network Module as entered on the Configuration page.
<code_revision> is replaced with the code revision for the firmware programmed into the Network Module.

Where a Home Assistant Auto Discovery Payload is not empty it takes this form for the “binary-sensor” topics:

```
{
"uniq_id":"<macaddress>_input_01",
```

"name":"<devicename> input 01",
"~":"NetworkModule/<devicename>",
"avty_t":"~/availability",
"stat_t":"~/input/01",
"dev":{
"ids":["NetworkModule_<macaddress>"],
"mdl":"HW-584",
"mf":"NetworkModule",
"name":"<devicename>",
"sw":"<code_revision>"
}
}

The above example is for Input on IO 01.
<macaddress> is replaced with the MAC address of the Network Module as entered on the Configuration page..
<devicename> is replaced with the Name of the Network Module as entered on the Configuration page.
<code_revision> is replaced with the code revision for the firmware programmed into the Network Module.

Where a Home Assistant Auto Discovery Payload is not empty it takes this form for the "sensor" topics:

{
"uniq_id":"<macaddress>_temp_012356abcdef",
"name":"<devicename> temp 012356abcdef ",
"~":"NetworkModule/<devicename>",
"avty_t":"~/availability",
"stat_t":"~/temp/012356abcdef ",
"unit_of_meas":"\xc2\xb0\x43",
"dev":{
"ids":["NetworkModule_<macaddress>"],
"mdl":"HW-584",
"mf":"NetworkModule",
"name":"<devicename>",
"sw":"<code_revision>"
}
}

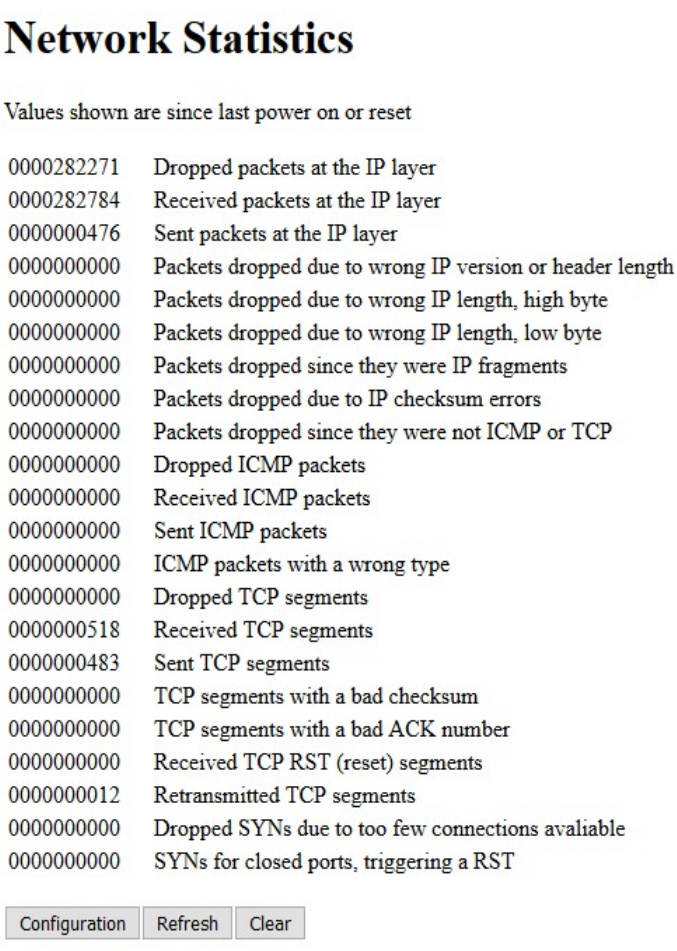
The above example is for Temperature Sensor ID "012356abcdef". The ID is a 12 character hex encoded field representing the 6 digits (48 bits) of the Temperature Sensor serial number.
<macaddress> is replaced with the MAC address of the Network Module as entered on the Configuration page..

<devicename> is replaced with the Name of the Network Module as entered on the Configuration page.
<code_revision> is replaced with the code revision for the firmware programmed into the Network Module.

Notes on Network Statistics – Browser Only

Network Statistics are accessible only via the http command “http://IP:Port/68”. This page is available in the Browser Only build (there is not enough memory in the MQTT build to include it). To be honest I was reluctant to add this page as it has only been minimally useful – but it is kind of cool so here it is. The information may be useful if you are debugging your network or developing applications to interface to the Network Module.

Note: Seeing large numbers of “Dropped packets at the IP layer” is not unusual. At least not unusual in my network. I traced this to Smart Home devices that seem to attempt some form of connection maintenance with everything on the network many times per hour. And I have lots of Smart Home devices on my network … so … lots of connection attempts. The Network Module drops these requests. The difference between “Dropped” and “Received” is the number of packets actually destined for the Network Module.



Network Statistics

Values shown are since last power on or reset

0000282271	Dropped packets at the IP layer
0000282784	Received packets at the IP layer
0000000476	Sent packets at the IP layer
0000000000	Packets dropped due to wrong IP version or header length
0000000000	Packets dropped due to wrong IP length, high byte
0000000000	Packets dropped due to wrong IP length, low byte
0000000000	Packets dropped since they were IP fragments
0000000000	Packets dropped due to IP checksum errors
0000000000	Packets dropped since they were not ICMP or TCP
0000000000	Dropped ICMP packets
0000000000	Received ICMP packets
0000000000	Sent ICMP packets
0000000000	ICMP packets with a wrong type
0000000000	Dropped TCP segments
0000000518	Received TCP segments
0000000483	Sent TCP segments
0000000000	TCP segments with a bad checksum
0000000000	TCP segments with a bad ACK number
0000000000	Received TCP RST (reset) segments
0000000012	Retransmitted TCP segments
0000000000	Dropped SYNs due to too few connections avaliable
0000000000	SYNs for closed ports, triggering a RST

Configuration Refresh Clear

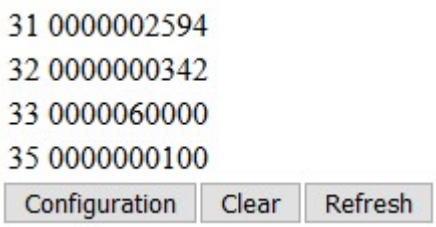
Notes on Link Error Statistics

Link Error Statistics are accessible only via the http command “http://IP:Port/66”. The statistics may be useful to you for determining if Full Duplex works better than Half Duplex in your particular network configuration.

As noted in other parts of the manual the normal mode of operation is “Half Duplex”, and you should not need to change that. However, during development it was noticed that the Network Module works better with some Cisco 1 Gbit business class switches if the Network Module and the Cisco switch are manually set to Full Duplex. So, this statistics page can let you compare error statistics over several hours or days to help you decide which configuration works better for you.

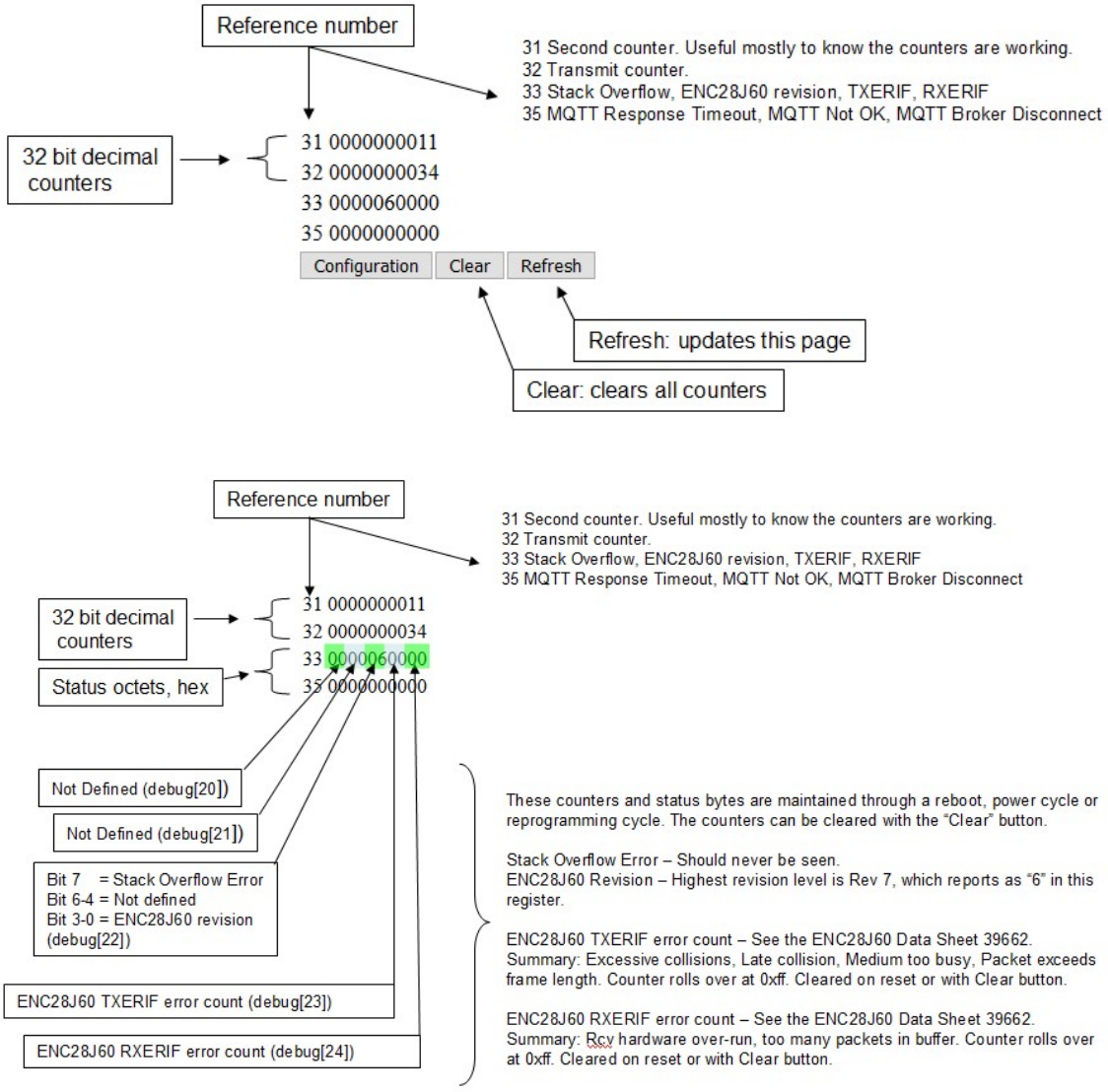
IMPORTANT: If you use “Full Duplex” the ENC28J60 specifications state that you must manually set your switch to “Full Duplex”. This is because the ENC28J60 cannot auto-negotiate Full/Half Duplex. If your switch does not allow you to manually set Full Duplex you should probably just leave the Network Module at Half Duplex. Having said that, experimentation suggests that some switches will work just fine regardless of the Full/Half duplex setting in the Network Module. So … do your own experiments and use the “/66” command to observe results. But in general I suggest you just leave the module at its default “Half Duplex” mode.

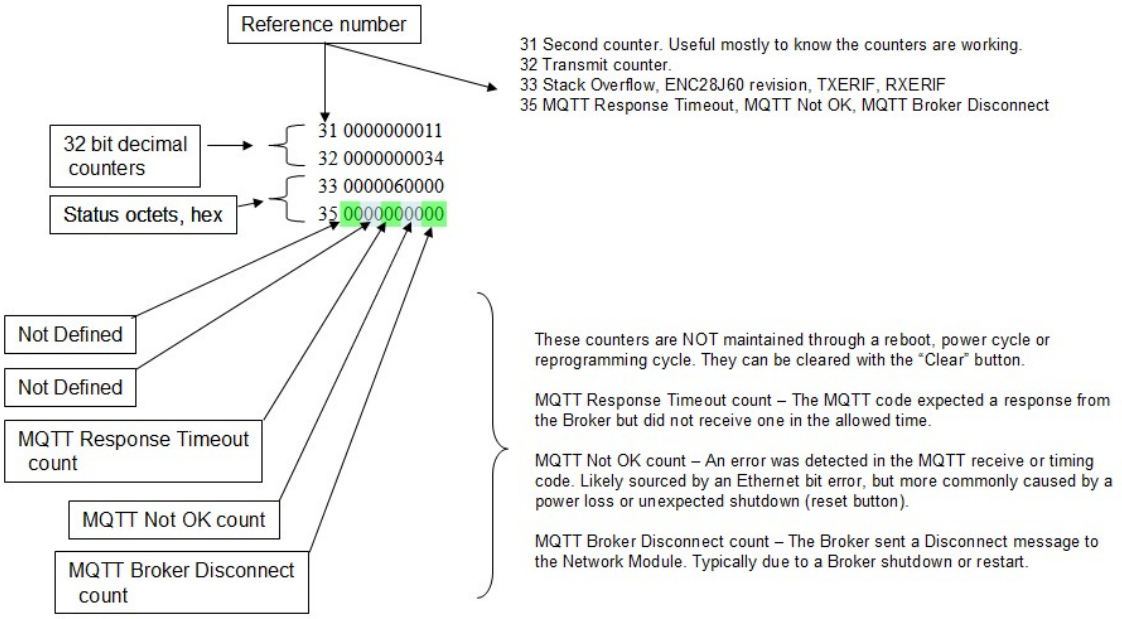
If you enter the “/66” command you’ll get a display similar to this:



The following explains the above fields:

Debug Statistics Display Schema





Seconds counter: # of seconds since boot
Transmit counter: # of transmits from the ENC28J60
Stack error: Not a counter, "1" will indicate detection of a stack overflow. That's bad … drop me a note if you see that happen.
ENC28J60 revision: Chip revision. This should indicate "06", which indicates chip B7.
TXERIF: count of TRXERIF errors (see ENC28J60 documentation)
RXERIF: count of RXERIF errors (see ENC28J60 documentation)
MQTT Response Timeout count: Count of response timeouts in the MQTT code
MQTT Not OK: Count of MQTT Not OK events in the MQTT code
MQTT Broker Disconnect count: Count of MQTT Broker disconnects as detected in the MQTT code

The Link Error Statistics page is "semi-hidden" and enabled only by the "/66" command because it can be very confusing to the typical user.

The content of the Link Error Stats page is likely to change as the code matures. It was added to assist with testing of MQTT performance using a variety of switches. One critical finding is that the statistics are not completely consistent, likely due to the number of variables affecting the values. Still, they can be useful for relative measurements when determining if you might have unusually high error rates on your network, and/or if you might need to experiment with the Half / Full Duplex setting.

The RXERIF error indicates that the ENC28J60 experienced a receive buffer overflow condition. This likely indicates extremely high network traffic. The impact is that packets received at the Network Module may be dropped. It is not unusual to have a few of these errors over a long period of time.

The TXERIF error indicates that a transmit abort has occurred. This error can occur as a result of the following;

1) Excessive Ethernet packet collisions
2) Late collisions
3) Transmission was unable to occur because the medium was occupied for too long.

It is normal to see transmit or receive errors, just not "too many", which is somewhat arbitrary. On a LAN with well behaving clients and good cabling you may only see one error every few months. Communication over WiFi, WAN, or Internet will see much higher error rates. And the error rate depends on how much traffic is occurring. Some users may never see an error indication, which is great. Others may see lots of errors, which may be 'normal' or may indicate a problem depending on their specific environment.

These counters are included because they were useful during development to help determine that Full Duplex worked much better than Half Duplex on the Cisco 1Gb “business level” managed switches. With Half Duplex a TXERIF error was occurring several times per day, accompanied by an MQTT disconnect (and automatic reconnect). Once Full Duplex was enabled zero errors were seen for several weeks in the Cisco 1Gb configuration.

On the other hand, TXERIF errors were seen when Full Duplex was used with some unmanaged switches, and no errors when those same switches were used with Half Duplex.

Your specific configuration and conditions may require some experimentation to get the best result.

Note: The “Seconds since boot” counter is approximate. The Network Module does not have a highly accurate clock, and there will be an accumulating deviation from real time, particularly if the counter is run for a long period. But, it is close enough for this purpose.

Functional Limitations

The code space and RAM in the processor on the Network Module is extremely limited, so there are many functional limitations that you would not expect on a device without these constraints. Some of the limitations to be aware of:

Maximum number of TCP Connections:

Maximum number of TCP Connections: 4. Implications: Each browser session and MQTT connection requires a TCP Connection. If you are running a non-MQTT build of the code you could connect up to 4 browsers to the device at one time. If you are running an MQTT build of the code you could also have up to 3 browser sessions connected.

Multiple browsers connected at the same time:

Multiple browsers connected at the same time CAN interfere with each other. For instance, multiple browsers attempting to make Configuration changes at the same time can cause unexpected results, particularly if Save is clicked on both browsers at the same time. I recommend you select ONE browser to make configuration changes, and the other browsers should be used for monitoring. Or at least make sure you only make configuration changes on one browser at a time.

Multiple browser tabs connected at the same time:

Using multiple browser tabs (say, one connected to the Configuration Page and another connected to the IOControl Page) may cause problems. I've tried it and it seems to work (at least with firmware version 20220120), but I know from the firmware design it would be easy to run out of resources and perhaps lose configuration or pin state information. I recommend instead using a single browser and a single tab in that browser, then use the buttons on the IOControl and Configuration page to move between pages. It is OK to use separate tabs or browsers to "monitor" the device – just don't use them to make configuration or pin state changes.

REST command rate:

Using the REST commands with a high repetition rate may slow the response time of the Network Module to the point that the browser interface becomes unusable. A suggestion for high repetition rates is to use the MQTT interface instead as it is more efficient than the HTML interface used by REST commands. Even so, you can push enough MQTT commands that the browsers might be unusable.

MQTT SSL/TSL:

MQTT does not support SSL/TSL. There is insufficient code space to implement this functionality.

Overall Command rate:

Processing speed is very limited given the functions implemented, so I imagine it will be easy to over-run the Network Module with state change requests. The code is single threaded, so whatever function has been requested must be completed before the next can be addressed. More testing needs to be done to determine if packets are simply dropped (if too many received) or if there are cases where the module may stop functioning. So far I haven't seen a "stopped functioning" scenario. If that were to occur a power cycle may be the only recovery option.

Configuration Errors:

There are very few "warnings" in the code to keep the user from creating bad configurations. The most concerning is that if you enable "Retain" for the power cycle output states AND you subject the device to rapid output state changes you run the risk of wearing out the EEPROM. Other situations likely only cause the device to lose contact with browsers or MQTT brokers (like mis-configuring IP addresses or Port numbers).

- IMPORTANT: "Retain" was ON by default in the early code releases. The default "Boot State" is now "OFF" to aid in preventing a user from inadvertently wearing out EEPROM. The original usage scenario was anticipated to be one in which output state changes would occur only via human interaction with a browser (therefore "infrequent changes"). Subsequent users have started implementing increasing automation, with some saying they may create many output changes per hour forever. In those "frequent output change' scenarios I strongly recommend using the OFF or ON "Boot State" setting instead of "Retain" to eliminate the EEPROM wearout concern.

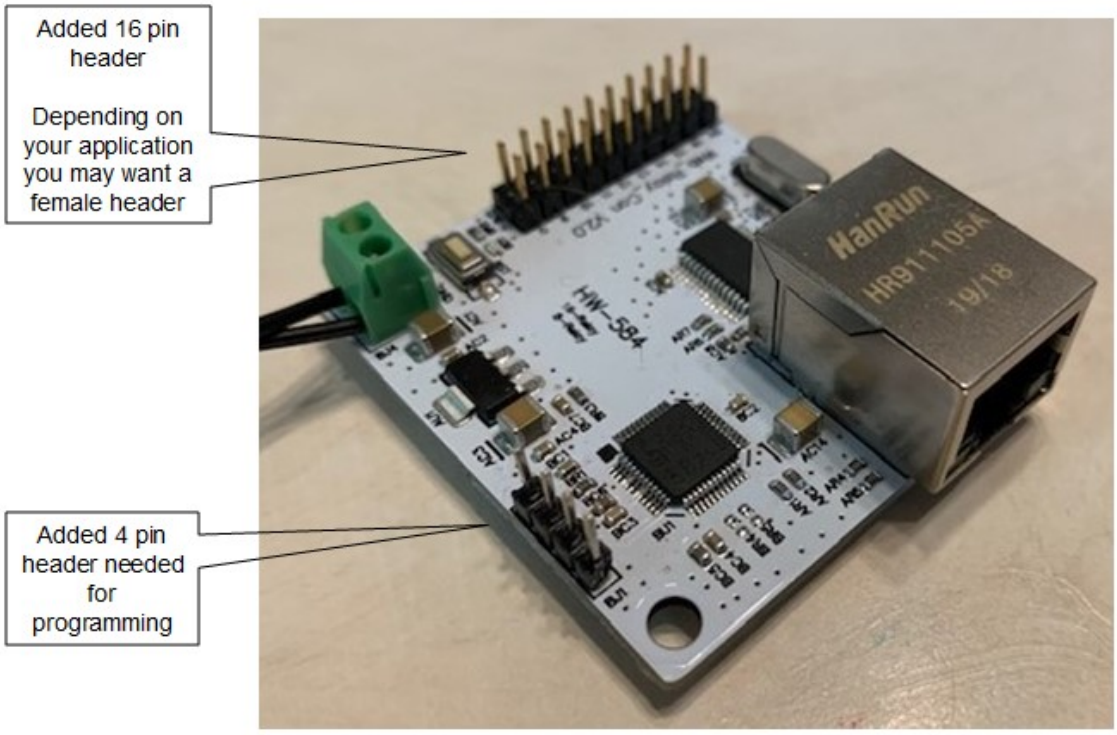
Programming the Module

Assuming you have the Web_Relays_Con V2.0 HW-584 and all you want to do is apply this firmware the following describes the process.

> **IMPORTANT NOTE: In the steps below you'll turn off the Read Out Protection bit on the Network Module. This will ERASE the program currently in the device. It will only work again after you successfully reprogram it. DO THIS AT YOUR OWN RISK.**

Note that as of January 2021 I've gone to one release of the firmware that covers the functionality of the four previous parallel releases.

1) Prepare your Network Module: Install a 4 pin header on the board (see photo)



2) Buy the Programmer: Purchase a ST-Link V2 (see photo). If you are patient you can get one from China in about a month for about $3.50. Or in less than a week from within the US for about $6.00 (assuming you are in North America). Price estimates are as of June 2020. Search on Google, Amazon, eBay, etc.

The ST-Link V2 is required to reprogram the Network Module. It is a USB to SWIM interface module supported by free software from STMicroelectronics. You'll need a four wire Dupont cable if you don't already have one. Some sellers ship the module with a cable. The Dupont cable is just a simple four wire cable with female push connectors on each end (as shown in the photo below).

The ST-Link V2 modules come in several colors so pick the color you like.

3) Obtain and Install Free Software: All of my development work was on the Windows 10 OS. If you are using Linux you will have a little more homework to do on your own, but I don't think there is much difference. For Windows you'll need to download and install the following files:

en.stsw-link009.zip
You'll find the above at https://www.st.com/en/development-tools/stsw-link009.html

en.stvp-stm8.zip
You'll find the above at https://www.st.com/en/development-tools/stvp-stm8.html

You'll need to create an account at st.com to get the above software. It's free but they want an email address to contact you. When you try to download the software you'll be asked for your account credentials and given the option to create an account. By providing my email address I've gotten some invitations to online programming seminars but otherwise no spam. Not much hassle.

The stsw-link009 software is the driver to operate the ST-Link V2.

The stvp-stm8 software is a development utility and the programmer specific to the STM8 processor. When you install en.stvp-stm8 you'll get two programs:

1) ST Visual Develop
2) ST Visual Programmer (STVP)

I only used STVP even when developing the code. And if you are only reprogramming your devices STVP is the only tool you'll need.

4) Copy the Program: Now that you've installed the necessary software you need to copy the STVP Project file and the Binary file from GitHub that will be programmed into the Network Module.

NOTE: Beginning with the February 20, 2021 release you should obtain source code and executable files from the "Releases" part of the GitHub web page (along the right margin of the page).

On my Windows 10 machine the project was located in the following directory:

C:/Users/Mike/Documents/COSMIC/FSE_Compilers/CXSTM8/NetworkModule

If you locate your copy of the project files in a similar Documents file location this should minimize the tinkering you have to do. And should you decide to modify the program you'll already have an appropriate directory set up.

The STVP programmer needs a ".stp" and ".sx" file pair to program the Network Module. Now that we have one code set to cover all the previous functionality you'll only need to copy the following files into the Documents directory you created above:

For the MQTT plus limited Browser Version:

NetworkModule.stp - The STVP project file
NetworkModule.sx - The NetworkModule binary file

For the Browser Only Version:

NetworkModule-Browser.stp - The STVP project file
NetworkModule-Browser.sx - The NetworkModule binary file

You will find these files in the "Releases" section of the GitHub web page (along the right margin of the page). The above are the only files you need to copy from the GitHub project account if you only want to program your module and you are not jumping right into code modifications.

IMPORTANT1: Since the path to your "Documents" directory will be different than mine (if for no other reason than your user ID is different than "Mike), you may need to **edit the .stp file to match your directory path**. Open the .stp file with NotePad or NotePad++ and look for the following. Edit it to match the path to your .sz file.

```
30  File0=C:\Users\Mike\Documents\COSMIC\FSE_Compilers\CXSTM8\NetworkModule\NetworkModule.sxCRLF
```

IMPORTANT2: Later releases of the code have already modified the .stp file so that you should not need to edit it. If you find the following in the .stp file you only need to make sure that the .stp file and the .sx file are in the same directory:

```
30  File0=NetworkModule.sxCRLF
```

I use NotePad++ and have it set to show the CR/LF at the end of the line. If you use regular NotePad as your text editor you won't see that.

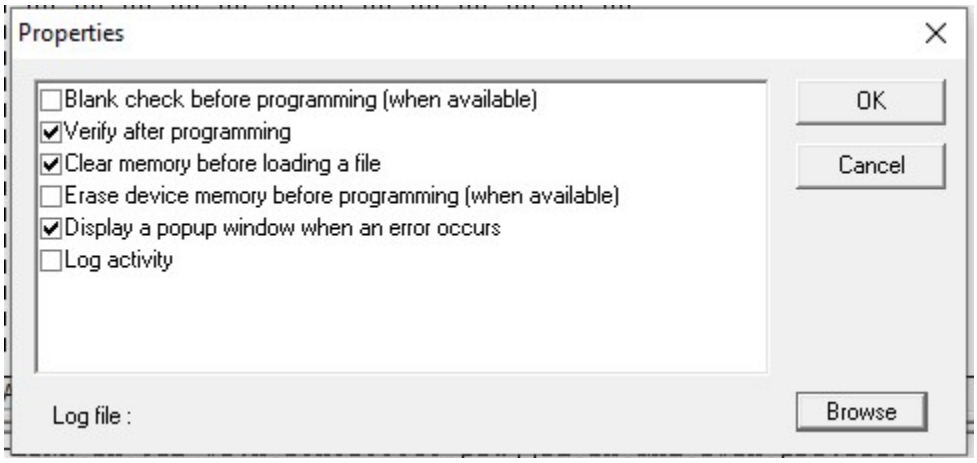
Telling STVP where your files are:

Since your User name on your Windows machine is probably not "Mike" you'll need to start STVP, click on "**Project/Open**", and browse for the .stp file that you copied to your **Documents/…** directory. Once you open the project file STVP should automatically load the .sx file from that same directory.

Setting up ST-Link Communication:

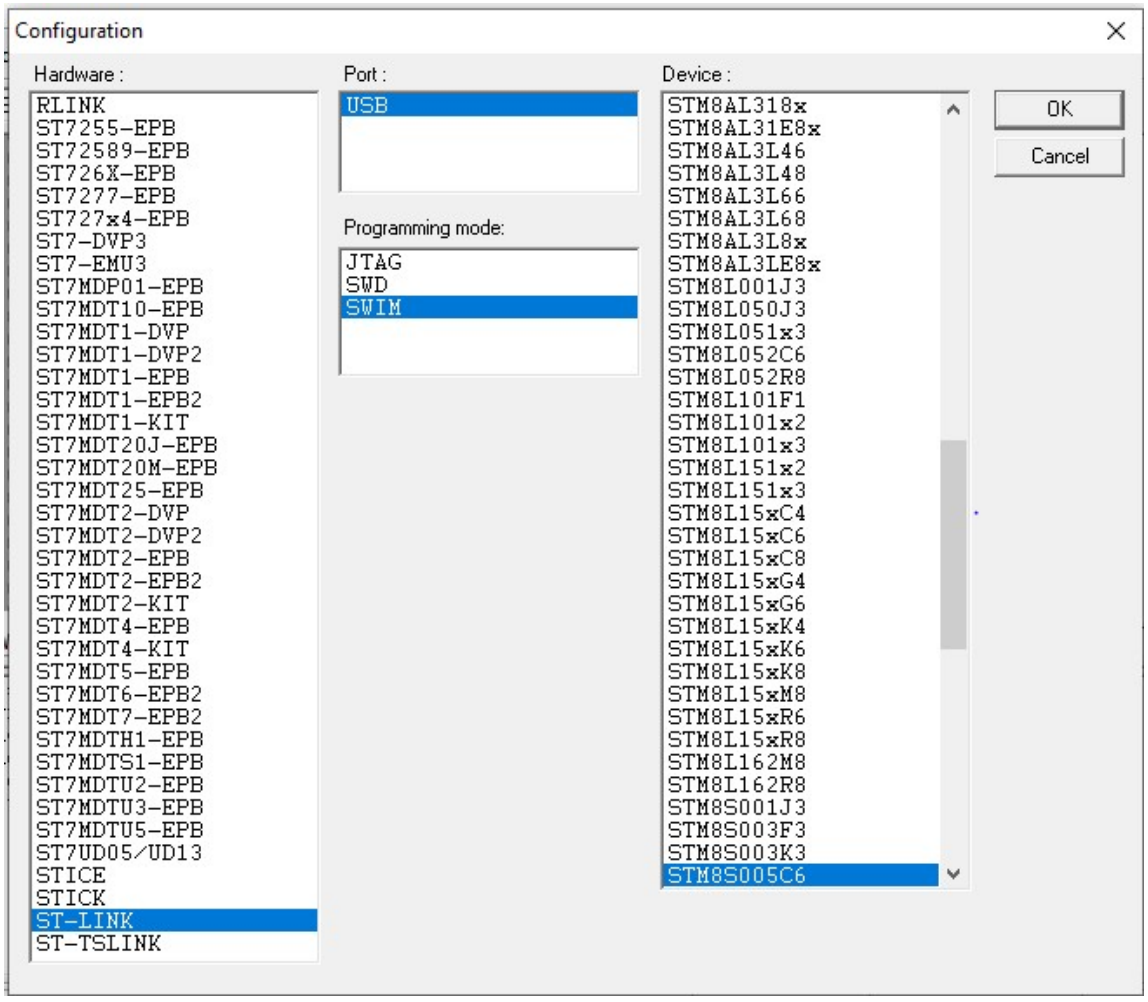
The project file contains various settings that enable the ST-Link V2 to communicate with your target board. They should already be set for you, but just in case the following is how I had them set:

Under "**Edit/Preferences**":



(Continued)

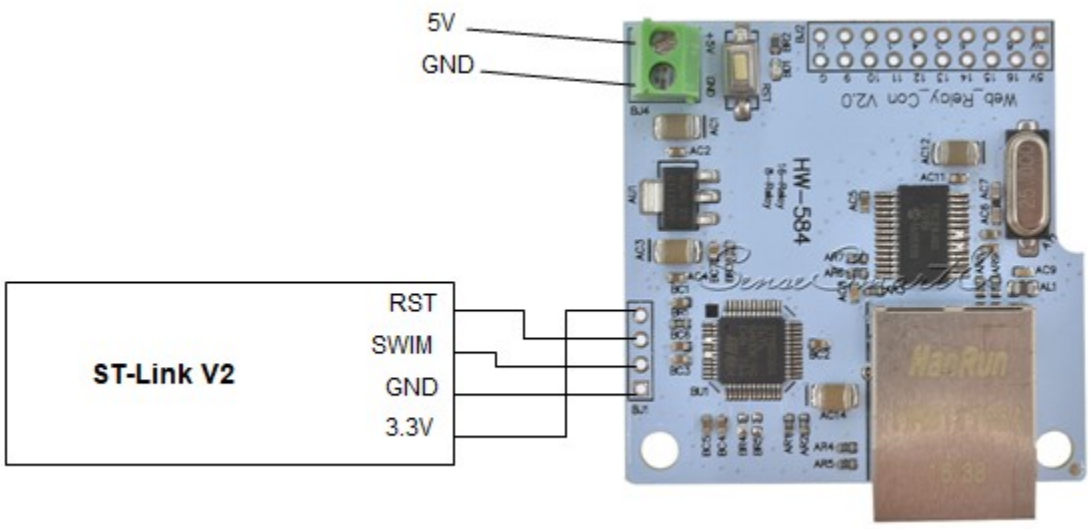
Under “**Configure/Configure ST Visual Programmer**”



If the above looks OK you are ready to program the Network Module.

Setting up the Hardware to allow programming:

First, attach the ST-Link V2 to your Network Module as follows:



Apply power to your Network Module. You should be using a 5V power supply connected to the power pins on the Network Module.

Plug the ST-Link V2 into your PC USB port.

If STVP is not already running, start it now.

If the NetworkModule.stp project is not already loaded, load it now (click on **"Project/Open"**, and browse for the .stp file that you copied to your **Documents/…** directory). Give it 10 or 20 seconds to load the .sx file.

If you see “out of range” messages like the following this is NOT an error. It would have been nice if the messages were more informative, but they are just telling you that the indicated addresses are in non-programmable areas of the chip during program load. The addresses shown are typically in EEPROM and RAM.

```
> Loading file C:\Users\Mike\Documents\COSMIC\FSE_Compilers\CXSTM8\MQTT-Network
FILE : line 961: Address 0x4000 is out of range and is ignored!
FILE : line 961: Address 0x4001 is out of range and is ignored!
FILE : line 961: Address 0x4002 is out of range and is ignored!
FILE : line 961: Address 0x4003 is out of range and is ignored!
FILE : line 961: Address 0x4004 is out of range and is ignored!
FILE : line 961: Address 0x4005 is out of range and is ignored!
FILE : line 961: Address 0x4006 is out of range and is ignored!
```

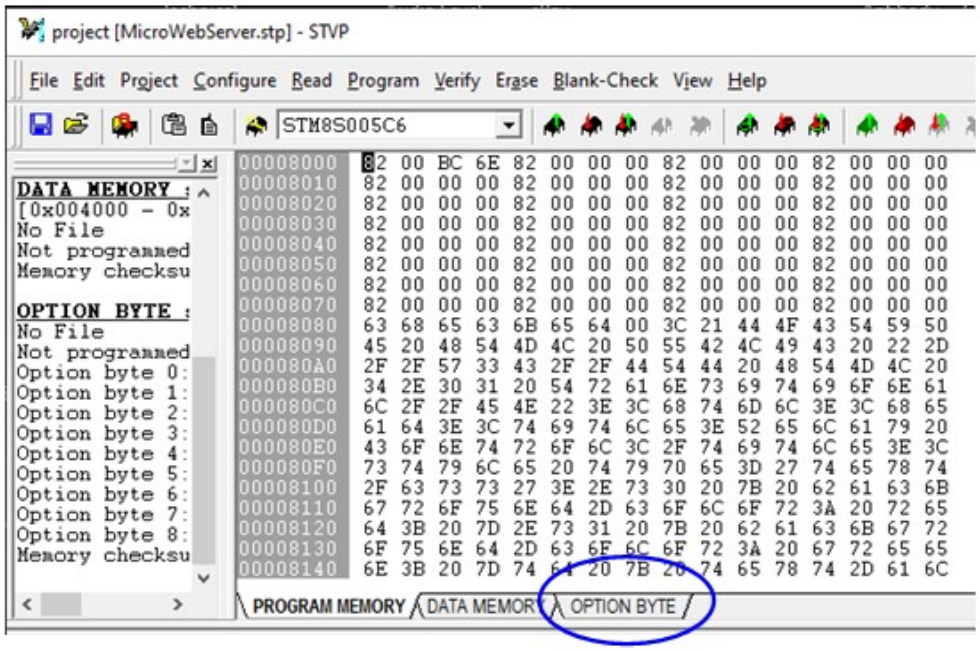
Once the program is successfully loaded in the programmer you will see a message like this (although the checksum will likely be different than what you see here).

```
FILE : line 964: Address 0xD is out of range and is ignored!
FILE : line 964: Address 0xE is out of range and is ignored!
FILE : line 964: Address 0xF is out of range and is ignored!
< File successfully loaded. File Checksum 0x297EE9
```

Clear the ROP Bit:

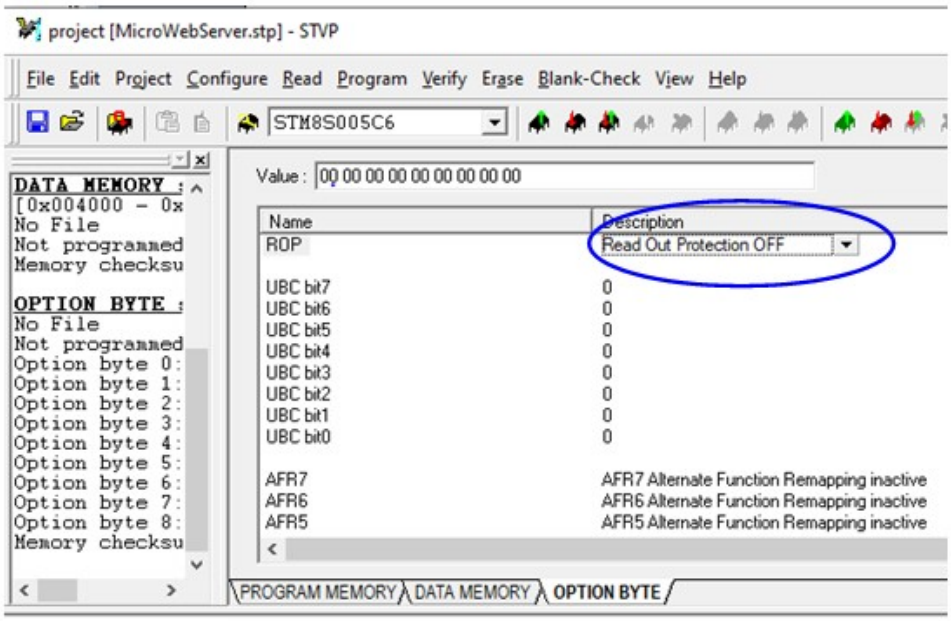
If this is the first time you are programming your Network Module you will need to clear the Read Out Protection (ROP) bit. If you don't clear the ROP any attempt to program the Network Module will give you a "This device is protected" message. How to clear the ROP bit:

In the STVP main window click on the "**Option Byte**" tab



(Continued)

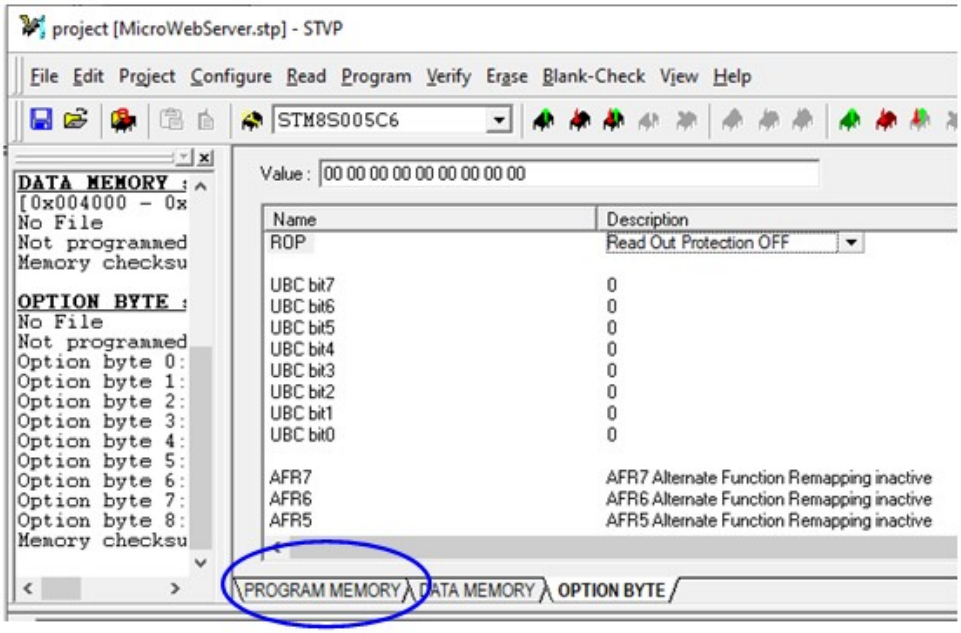
Make sure "**Read Out Protection OFF**" is selected in this drop down.



Next click on **"Program / Current Tab"**. This will clear the ROP bit and allow you to reprogram the device. **IMPORTANT: CLEARING THE ROP BIT ERASES THE CODE IN THE NETWORK MODULE. After you clear the ROP bit you MUST reprogram the Network Module to make it useful again.**

Programming the Device:

Select the Program Memory tab



Next select **"Program / Current Tab"**

If you got an error message while attempting to program the Network Module:

a) Make sure the RST connection is in place.
b) Make sure the power supply connected to the Network Module is providing 5V.
c) Make sure you have good connections from the ST-Link V2 to the Network Module.
d) You might have to unplug the ST-Link V2 from the USB port on your PC and plug it back in again.
e) You might have to stop the STVP program, unplug and replug the ST-Link V2, then restart the STVP program.
f) If you have 16 relays connected to your Network Module I suggest disconnecting them while reprogramming. If you have a very robust power supply it may be possible to leave them connected. The Network Module will be reset a couple of times during programming, and this may cause the relays to simultaneously turn on and off. Whether this interferes with programming depends on whether your power supply can handle the surge caused by the relay coils.

   Generally I haven't had to do any of the above as I seldom saw an error. But on occasion I saw an error message that the link was not working, and the above tinkering got it working again.

(Continued)

If you see a message indicating programming success you are ready to attempt to connect to the Network Module via the Ethernet connector.

a) Disconnect the RST wire between the ST-Link V2 and the Network Module. You can also disconnect the other wires, or leave them connected for the time being.
b) Connect the Ethernet cable. I suggest you do this the first time without using your network. Make a direct Ethernet cable connection from the Network Module to your PC and attempt to access it at 192.168.1.4:8080. If the connection does not work check your IPV4 Ethernet settings on the PC and set it to use IP address 192.168.1.100 (not DHCP). If you don't know how to do this Google it. Here's a helpful link: https://stevessmarthomeguide.com/setting-up-static-ip-address-windows-10/

While the device is directly connected to your PC you can use your browser to make address setting changes on the Network Module that are appropriate to your network. Then you can connect the device to your network, return your PC to its original Ethernet settings, and attempt to access the device.

> Comment: If your network is based on 191.168.1.xxx addresses you may be able to avoid step "b" above and just connect the Network Module to your network to contact it. You need to be sure that 192.168.1.4 and MAC c2:4d:69:6b:65:00 do not conflict with any other device on your network.

Note: See the section "Alternative Way to Set Initial IP Address".

IMPORTANT: WHEN REPROGRAMMING THE BROWSER ONLY VERSION

If you need to re-install or upgrade the Browser Only version remember that the IO Names and IO Timer values are stored in Flash. For this reason you will want to make sure you don't overwrite those values, otherwise you will have to manually re-enter them. You don't have to worry about this the first time you install the Browser Only version. But on subsequent installs do the following:

INSTEAD of programming with **"Program / Current Tab"**
use **"Program / Address Range"**
and enter the range 8000 to FEBF.
The above step will prevent overwriting your IO Names and IO Timer values.

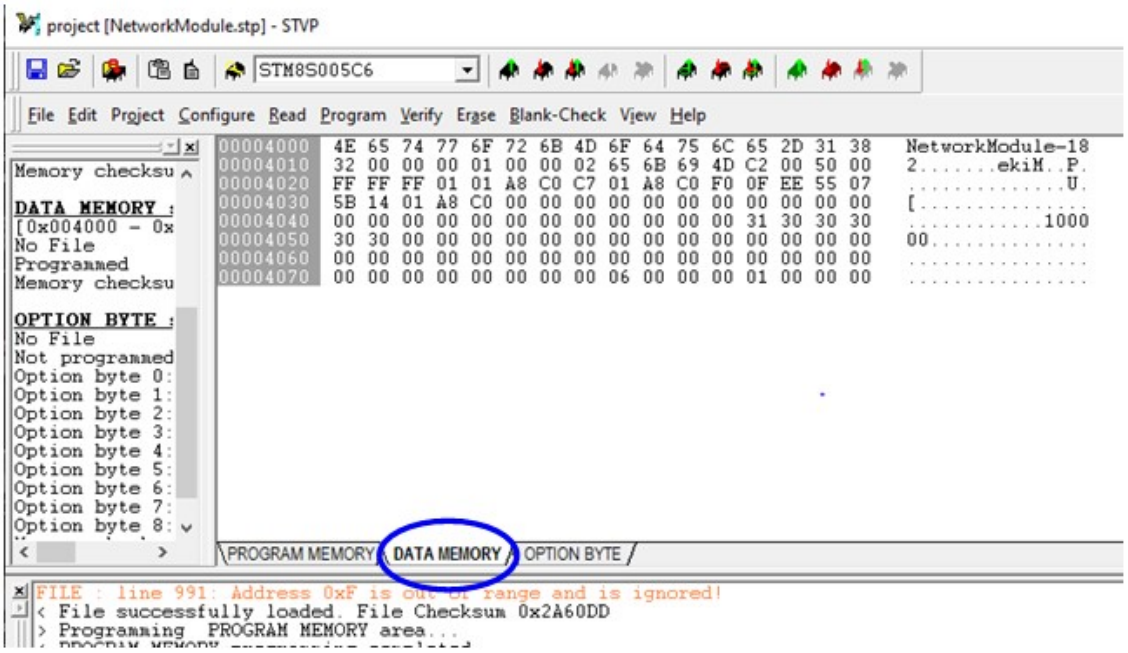
If you forget and use **"Program / Current Tab"** you'll see an error message – but the programming will complete and the IO Names and IO Timer values will return to defaults.

Alternative Way to Force Defaults or Downgrade Firmware

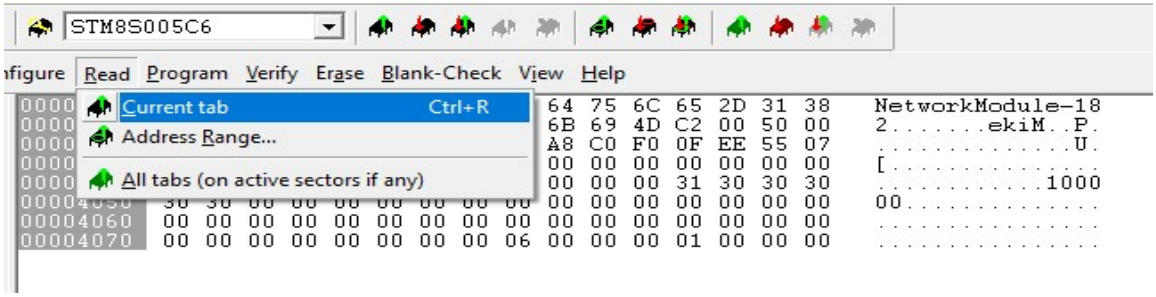
Normally all you have to do to return to “factory defaults” is press the Reset button on the Network Module board for 10 seconds. However, during development there were a couple of times where pressing the Reset button did not revive the Network Module to a point that it would operate. Admittedly this was due to “in development” code errors and may not be needed, but I’m providing it just in case.

This information is only useful if you are going to do your own development from this code. I DO NOT RECOMMEND THIS METHOD UNLESS ALL ELSE HAS FAILED TO REVIVE THE DEVICE.

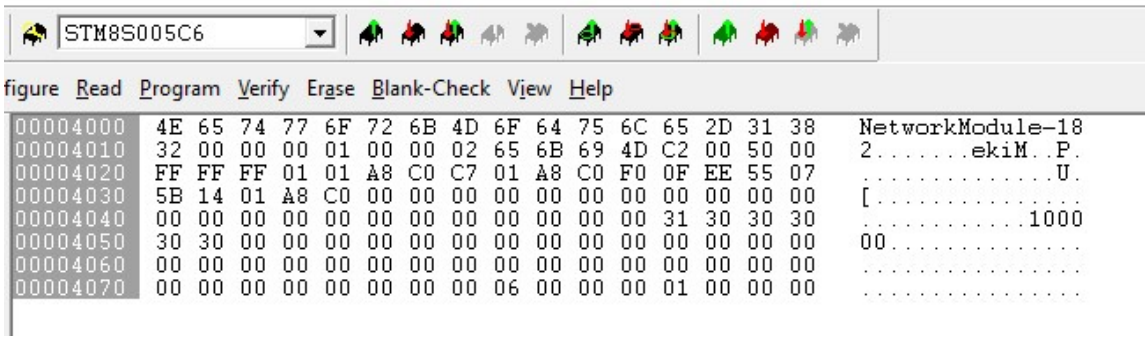
1) When the Network Module is connected to the STLink (including the Reset wire) you can access the EEPROM content with the “Data Memory” tab.



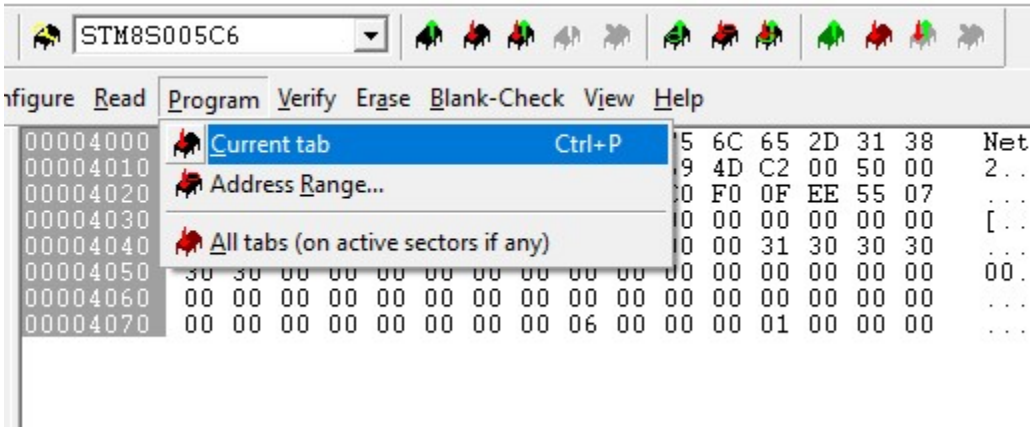
2a) After clicking on the Data Memory tab click on Read / Current tab to read the EEPROM contents.



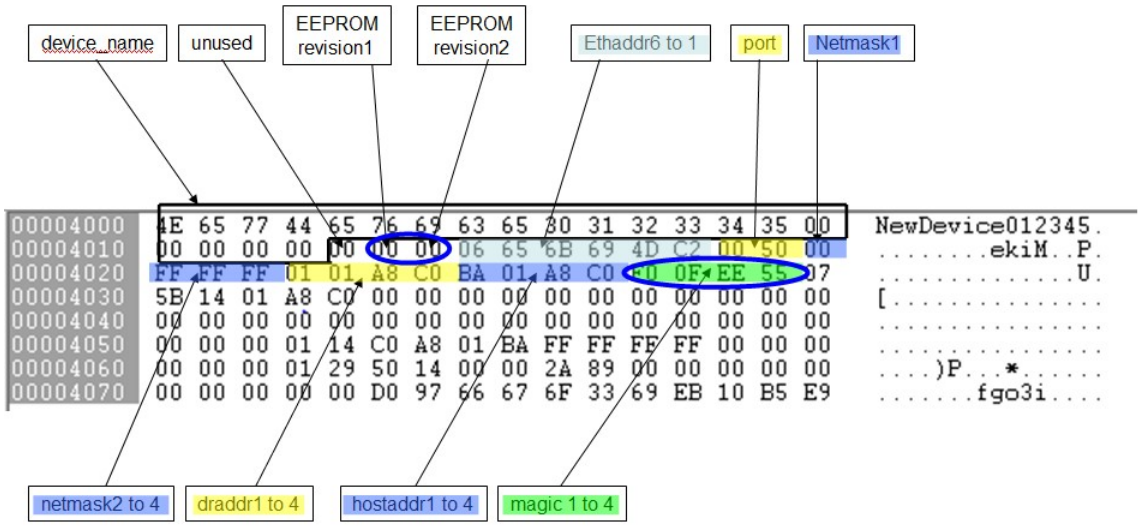
You'll get a display that looks something like this:



2b) You can change any value in the Data Memory one character at a time, then you can write the result to the EEPROM with the Program / Current tab selection.



2c) To force the EEPROM to a state where firmware must start over again set the bytes circled in blue to 0.



After committing to EEPROM reboot the device (power cycle or reconnect the reset wire, wait 2 seconds, and disconnect the reset wire).

2) You can now disconnect the STLink and connect the Network Module to your network. Once connected you should be able to use a browser to connect to the Network Module via the factory default address 192.168.1.4:8080.

FIRMWARE DOWNGRADE

Once you've cleared the EEPROM content as shown above you should be able to load a prior release of the firmware should you need to do so. If your intention is to downgrade the firmware DO NOT reboot the device until you do so.

Alternative Way to Set Initial IP Address

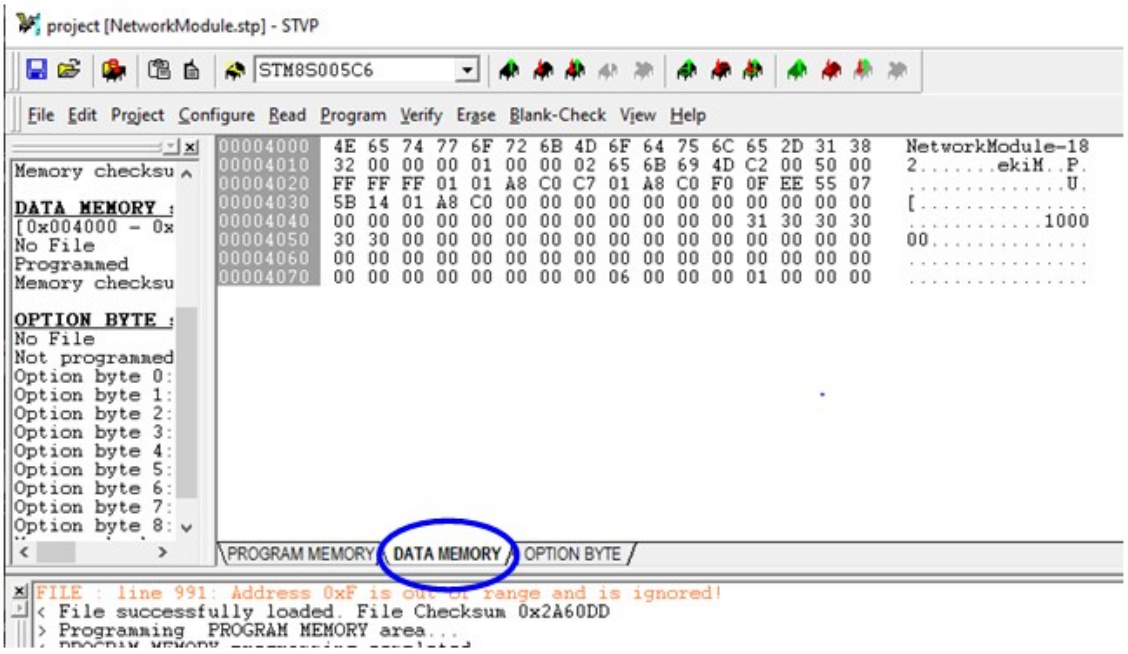
A user commented that it was cumbersome to have to set up a laptop with a fixed IP Address to program the Network Module with its first “network compatible address”. Here’s an alternative that may be useful to you.

Let’s say your network already uses 192.168.1.4, so you can’t attach the device directly to your network. Or perhaps your network uses some other variant of the 192.168.xxx.xxx address range, or even the 10.0.0.x address range. A way to work around this without needing to set up a laptop or PC for the initial Ethernet connection to the Network module as follows;

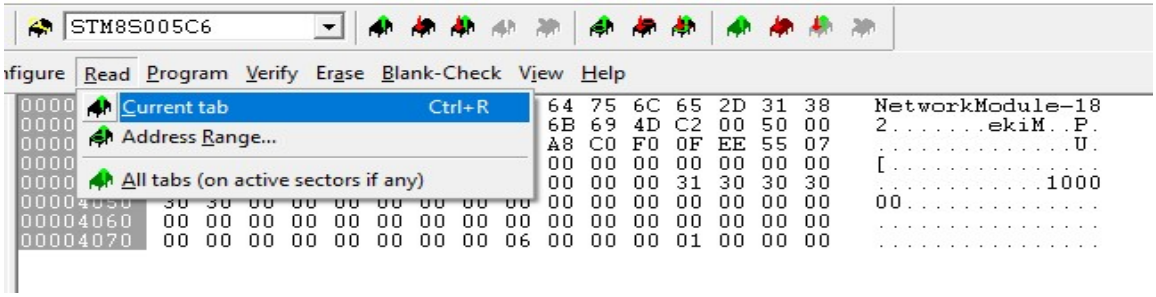
1) Assumption is that you successfully programmed the flash with the Network Module code. Reboot the Network Module (usually just be releasing the reset wire). Then reconnect the reset wire.
2) Using the STLink change the IP address in the EEPROM. Reboot the Network Module.
3) Attach the Network Module to your network, access the Network Module with a browser, and finish changing any settings via the Configuration menu.

Here’s an illustrated version of the above with greater detail;

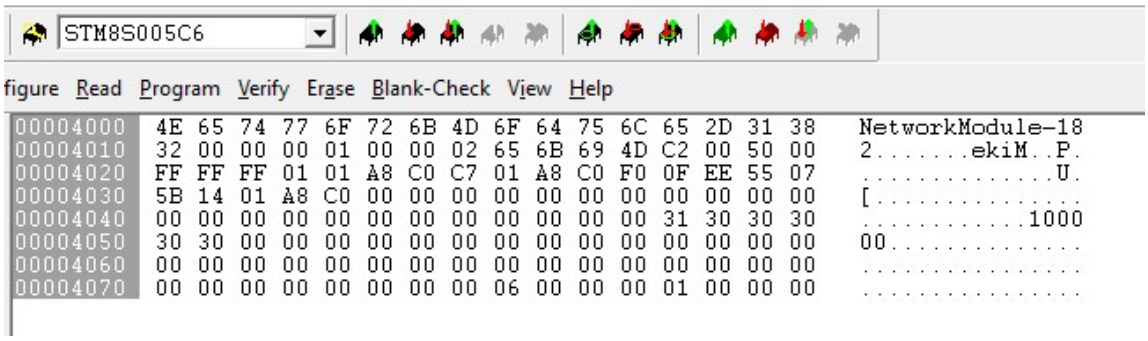
3) Step 1 is covered in the section “Programming the Module”
4) When the Network Module is connected to the STLink (including the Reset wire) you can access the EEPROM content with the “Data Memory” tab.



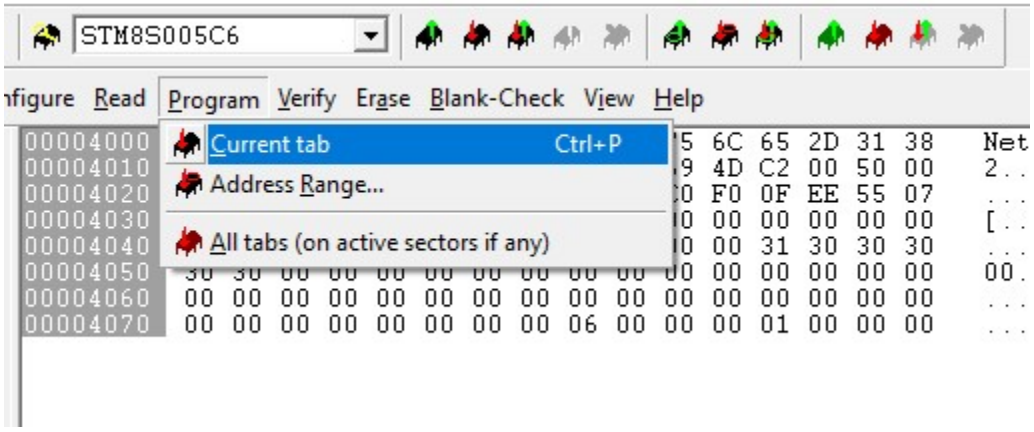
2a) After clicking on the Data Memory tab click on Read / Current tab to read the EEPROM contents.



You'll get a display that looks something like this:

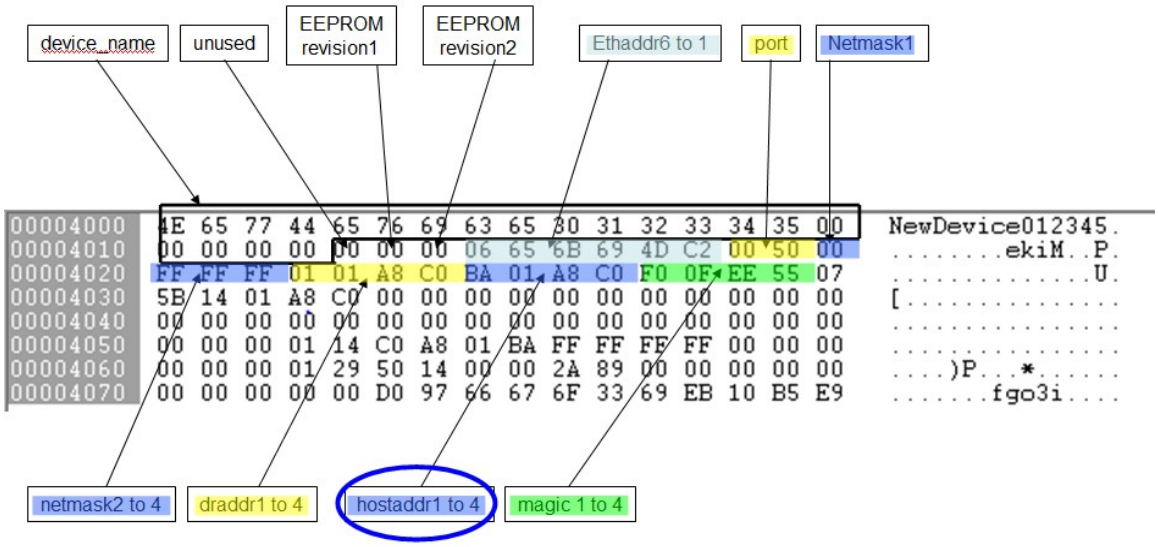


2b) You can change any value in the Data Memory one character at a time, then you can write the result to the EEPROM with the Program / Current tab selection.



2c) SO …. Which value do you want to change? You can change any of the values, but typically you only need to change the IP Address of the module to get it to appear on

your network. Looking at the EEPROM map the below shows where the IP Address is located.



"hostaddr1 to 4" is the Network Module IP Address. Note that it is in hex, and it is in reverse order (MSB on the right, LSB on the left). In decimal format the address shown is 192.168.1.186. Click on any character to change it, then be sure to use Program / Current tab to commit the changes to EEPROM.

After committing to EEPROM reboot the device (power cycle or reconnect the reset wire, wait 2 seconds, and disconnect the reset wire).

5) You can now disconnect the STLink and connect the Network Module to your network. Once connected you should be able to use a browser to connect to the Network Module and make any further changes you need in the Configuration page.

IMPORTANT NOTES:

a) You should not use the "direct access to the EEPROM" method for anything other than the minimum needed to gain access via a browser. Usually you only need to change the Network Module IP Address (hostaddr). It's too easy to make a mistake … so don't forget about the reset button if you mess it up.
b) If you press the reset button on the Network Module it will return to the hard-coded defaults, NOT to the changes you manually put in the EEPROM. You'll have to go through this process again to get back to a network compatible IP address.
c) If you change the "Magic Number" it will cause a return to factory defaults on reboot.

Display Values vs Pin Logic Levels

This information may be useful to you for understanding how the values displayed in the browser or contained in the MQTT fields correspond to output and input pin voltage levels.

16 Outputs

				State Values Reported in Browser	State Values reported in Short Form (/99)	Pin numbers as marked on board: Outputs Pin16	Pin15	Pin14	Pin13	Pin12	Pin11	Pin10	Pin9	Outputs Pin8	Pin7	Pin6	Pin5	Pin4	Pin3	Pin2	Pin1
			Invert Setting																		
Outputs	All On	All high (H)	No Inv	All Green	1111111111111111	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
	All Off	All low (L)	No Inv	All Red	0000000000000000	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
	1 On	Out1 high (H)	No Inv	#1 Green	1000000000000000	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	H
	8 On	Out8 high (H)	No Inv	#8 Green	0000000100000000	L	L	L	L	L	L	L	L	H	L	L	L	L	L	L	L
	All Off	All high (H)	Inv	All Red	0000000000000000	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
	All On	All low (L)	Inv	All Green	1111111100000000	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
	1 On	Out1 low (L)	Inv	#1 Green	1000000000000000	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
	8 On	Out8 low (L)	Inv	#8 Green	0000000100000000	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
		H = +3V logic level			Note that Pin 1 is the							H = +3V logic level									
		L = 0V logic level										L = 0V logic level									

8 Out / 8 In

				State Values Reported in Browser	State Values reported in Short Form (/99)	Pin numbers as marked on board: Inputs Pin16	Pin15	Pin14	Pin13	Pin12	Pin11	Pin10	Pin9	Outputs Pin8	Pin7	Pin6	Pin5	Pin4	Pin3	Pin2	Pin1
			Invert Setting																		
Inputs	All On	All high (H)	No Inv	All Green	xxxxxxxx11111111	H	H	H	H	H	H	H	H								
	All Off	All low (L)	No Inv	All Red	xxxxxxxx00000000	L	L	L	L	L	L	L	L								
	1 On	In1 high (H)	No Inv	#1 Green	xxxxxxxx10000000	L	L	L	L	L	L	L	H								
	8 On	In8 high (H)	No Inv	#8 Green	xxxxxxxx00000001	H	L	L	L	L	L	L	L								
	All Off	All high (H)	Inv	All Red	xxxxxxxx00000000	H	H	H	H	H	H	H	H								
	All On	All low (L)	Inv	All Green	xxxxxxxx11111111	L	L	L	L	L	L	L	L								
	1 On	In1 low (L)	Inv	#1 Green	xxxxxxxx10000000	H	H	H	H	H	H	H	L								
	8 On	In8 low (L)	Inv	#8 Green	xxxxxxxx00000001	L	H	H	H	H	H	H	H								
Outputs	All On	All high (H)	No Inv	All Green	11111111xxxxxxxx									H	H	H	H	H	H	H	H
	All Off	All low (L)	No Inv	All Red	00000000xxxxxxxx									L	L	L	L	L	L	L	L
	1 On	Out1 high (H)	No Inv	#1 Green	10000000xxxxxxxx									L	L	L	L	L	L	L	H
	8 On	Out8 high (H)	No Inv	#8 Green	00000001xxxxxxxx									H	L	L	L	L	L	L	L
	All Off	All high (H)	Inv	All Red	00000000xxxxxxxx									H	H	H	H	H	H	H	H
	All On	All low (L)	Inv	All Green	11111111xxxxxxxx									L	L	L	L	L	L	L	L
	1 On	Out1 low (L)	Inv	#1 Green	10000000xxxxxxxx									H	H	H	H	H	H	H	L
	8 On	Out8 low (L)	Inv	#8 Green	00000001xxxxxxxx									L	H	H	H	H	H	H	H
		H = +3V logic level			Note that Pin 1 is the							H = +3V logic level									
		L = 0V logic level			left most bit, Pin 16 is							L = 0V logic level									
					the right most bit																

	16 Inputs																					
				State Values Reported in Browser	State Values reported in Short Form (/99)	Pin numbers as marked on board																
						Inputs									Inputs							
			Invert Setting			Pin16	Pin15	Pin14	Pin13	Pin12	Pin11	Pin10	Pin9		Pin8	Pin7	Pin6	Pin5	Pin4	Pin3	Pin2	Pin1
Inputs	All On	All high (H)	No Inv	All Green	1111111111111111	H	H	H	H	H	H	H	H		H	H	H	H	H	H	H	H
	All Off	All low (L)	No Inv	All Red	0000000000000000	L	L	L	L	L	L	L	L		L	L	L	L	L	L	L	L
	1 On	In1 high (H)	No Inv	#1 Green	1000000000000000	L	L	L	L	L	L	L	L		L	L	L	L	L	L	L	H
	8 On	In8 high (H)	No Inv	#8 Green	0000000100000000	L	L	L	L	L	L	L	L		H	L	L	L	L	L	L	L
	All Off	All high (H)	Inv	All Red	0000000000000000	H	H	H	H	H	H	H	H		H	H	H	H	H	H	H	H
	All On	All low (L)	Inv	All Green	1111111111111111	L	L	L	L	L	L	L	L		L	L	L	L	L	L	L	L
	1 On	In1 low (L)	Inv	#1 Green	1000000000000000	H	H	H	H	H	H	H	H		H	H	H	H	H	H	H	L
	8 On	In8 low (L)	Inv	#8 Green	0000000100000000	H	H	H	H	H	H	H	H		L	H	H	H	H	H	H	H
		H = +3V logic level			Note that Pin 1 is the							H = +3V logic level										
		L = 0V logic level			left most bit, Pin 16 is							L = 0V logic level										
					the right most bit																	

	MQTT 8Out / 8In																					
				State Values Reported in MQTT		State Values Reported in Browser	State Values reported in Short Form (/99)	Pin numbers as marked on board															
								Inputs								Outputs							
			Invert Setting	Byte 0 Inputs	Byte 1 Relays			Pin16	Pin15	Pin14	Pin13	Pin12	Pin11	Pin10	Pin9	Pin8	Pin7	Pin6	Pin5	Pin4	Pin3	Pin2	Pin1
Inputs	All On	All high (H)	No Inv	FF		All Green	xxxxxxxx11111111	H	H	H	H	H	H	H	H								
	All Off	All low (L)	No Inv	00		All Red	xxxxxxxx00000000	L	L	L	L	L	L	L	L								
	1 On	In1 high (H)	No Inv	01		#1 Green	xxxxxxxx10000000	L	L	L	L	L	L	L	H								
	8 On	In8 high (H)	No Inv	80		#8 Green	xxxxxxxx00000001	H	L	L	L	L	L	L	L								
	All Off	All high (H)	Inv	00		All Red	xxxxxxxx00000000	H	H	H	H	H	H	H	H								
	All On	All low (L)	Inv	FF		All Green	xxxxxxxx11111111	L	L	L	L	L	L	L	L								
	1 On	In1 low (L)	Inv	01		#1 Green	xxxxxxxx10000000	H	H	H	H	H	H	H	L								
	8 On	In8 low (L)	Inv	80		#8 Green	xxxxxxxx00000001	L	H	H	H	H	H	H	H								
Outputs	All On	All high (H)	No Inv		FF	All Green	11111111xxxxxxxx									H	H	H	H	H	H	H	H
	All Off	All low (L)	No Inv		00	All Red	00000000xxxxxxxx									L	L	L	L	L	L	L	L
	1 On	Out1 high (H)	No Inv		01	#1 Green	10000000xxxxxxxx									L	L	L	L	L	L	L	H
	8 On	Out8 high (H)	No Inv		80	#8 Green	00000001xxxxxxxx									H	L	L	L	L	L	L	L
	All Off	All high (H)	Inv		00	All Red	00000000xxxxxxxx									H	H	H	H	H	H	H	H
	All On	All low (L)	Inv		FF	All Green	11111111xxxxxxxx									L	L	L	L	L	L	L	L
	1 On	Out1 low (L)	Inv		01	#1 Green	10000000xxxxxxxx									H	H	H	H	H	H	H	L
	8 On	Out8 low (L)	Inv		80	#8 Green	00000001xxxxxxxx									L	H	H	H	H	H	H	H
		H = +3V logic level					Note that Pin 1 is the							H = +3V logic level									
		L = 0V logic level					left most bit, Pin 16 is							L = 0V logic level									
							the right most bit																

Network Module Schematic

I traced out the parts of the Network Module that are pertinent to developing the new software. I did not trace ALL connections as my intention was not to reverse engineer the hardware design. My intention was only to fix the inadequate function of the software. The schematic may be useful should you decide to improve on the software I've provided. Some notes:

- There are a number of capacitors connecting power and ground. These are left out of the schematic.

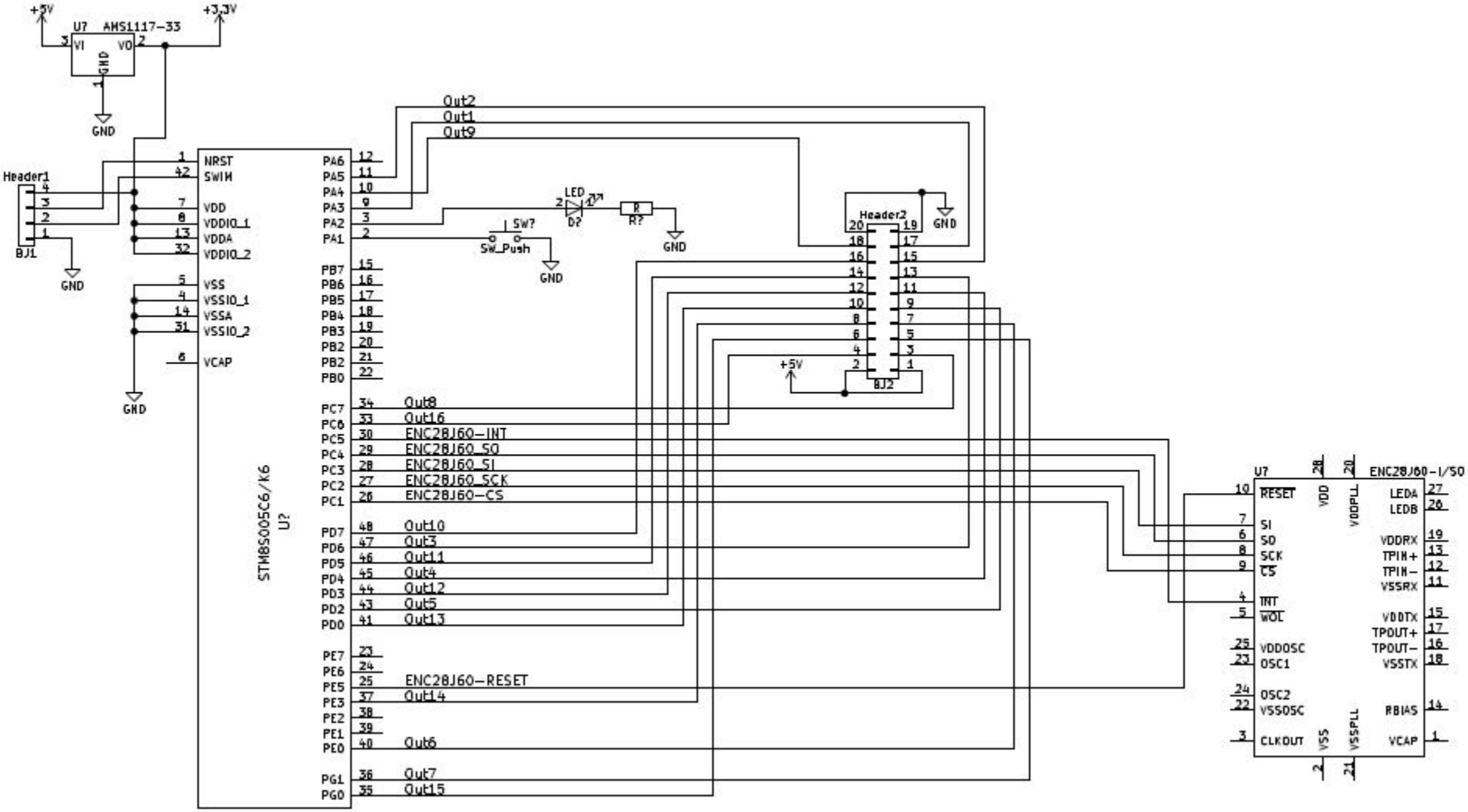
- The VCAP pin on the processor was not traced.

- Unused pins or pins that did not appear to be a necessary part of the functionality were programmed to be inputs with pull-ups. These are shown as disconnected on the schematic even if there was a component attached.
    - There are some components connected to the Port B pins. I suspect the original code used these to identify if the board was "8 port" or "16 port".

- I didn't trace out most of the pins on the ENC28J60, as I knew the design worked and did not need to do any modifications. Some notes:
    - The SPI interface on the ENC28J60 is not connected to the SPI interface on the STM8S005. Ordinary port pins on the STM8S005 are used to "bit bang" the SPI interface. Not very fast, but this is not an Ethernet performance design so it works just fine.
    - The –WOL pin does not appear to be connected.
    - The CLKOUT pin is not connected.

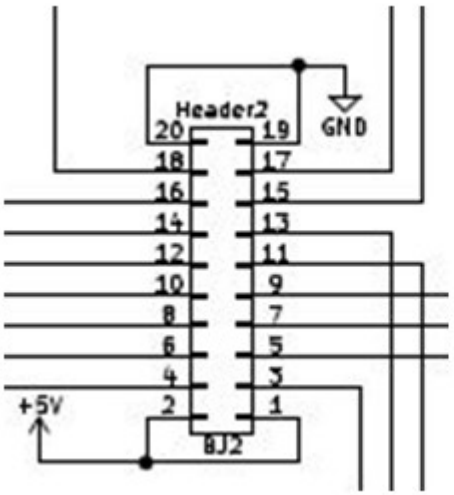
- If you dig into the STM8S005 specification you'll find that most pins that I show simply as "port pins" can be defined for other uses. I didn't include all that information in the component drawing as it just creates confusion in this context. The Network Module uses all the pins as "port pins", so that is all I show.

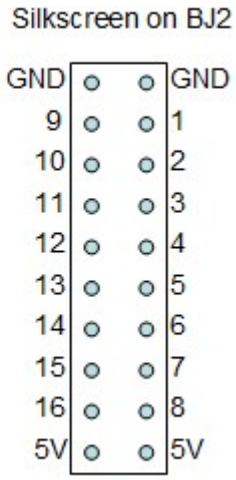
- The STM8S005 operates on its internal 16MHz clock. It does not have an external crystal or clock source.


+5V
U7 AMS1117-33
VI
VO
GND
+3.3V
GND
Header1
BJ1
GND
NRST
SWIM
VDD
VDDIO_1
VDDA
VDDIO_2
VSS
VSSIO_1
VSSA
VSSIO_2
VCAP
GND
STM8S005C6/K6
U?
PA6
PA5
PA4
PA3
PA2
PA1
PB7
PB6
PB5
PB4
PB3
PB2
PB2
PB0
PC7
PC6
PC5
PC4
PC3
PC2
PC1
PD7
PD6
PD5
PD4
PD3
PD2
PD0
PE7
PE6
PE5
PE3
PE2
PE1
PE0
PG1
PG0
Out2
Out1
Out9
LED
D?
R
R?
SW?
SW_Push
GND
GND
Header2
BJ2
GND
+5V
Out8
Out16
ENC28J60-INT
ENC28J60_SO
ENC28J60_SI
ENC28J60_SCK
ENC28J60-CS
Out10
Out3
Out11
Out4
Out12
Out5
Out13
ENC28J60-RESET
Out14
Out6
Out7
Out15
U7
ENC28J60-I/SO
RESET
SI
SO
SCK
CS
INT
WOL
VDDOSC
OSC1
OSC2
VSSOSC
CLKOUT
VDD
VDDPLL
VSS
VSSPLL
LEDA
LEDB
VDDRX
TPIN+
TPIN-
VSSRX
VDDTX
TPOUT+
TPOUT-
VSSTX
RBIAS
VCAP

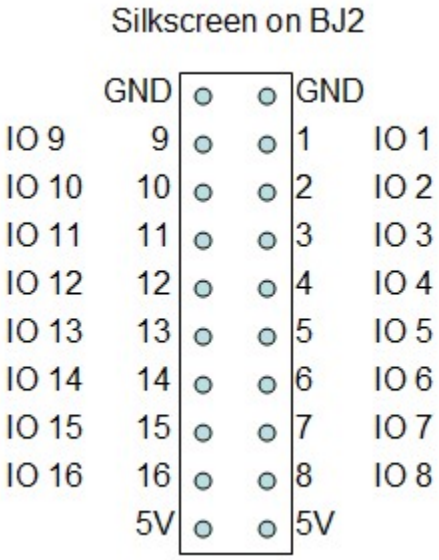
Pinouts

Schematic representation of the connection header vs silkscreen on the board;





Following are the pin definitions for the firmware configurations.

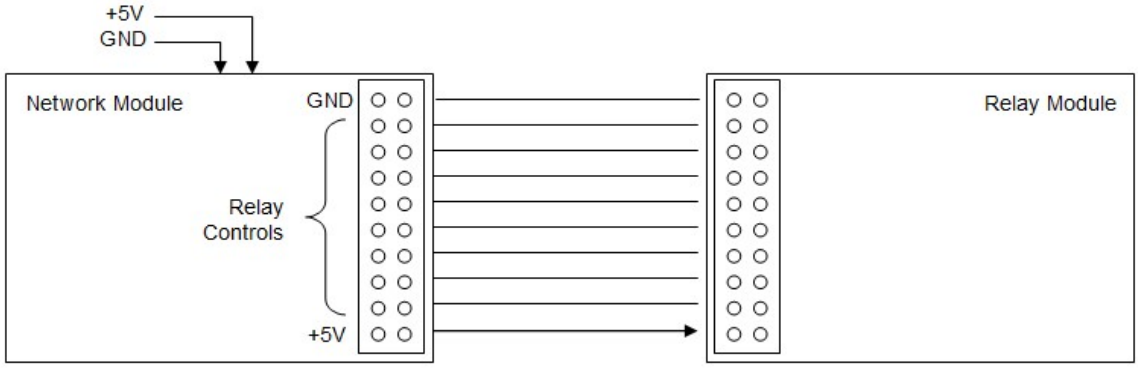


Notes on Interfacing to Relay Modules

There are two things to be cautious of when attaching relay modules to the Network Module: Power Distribution and Type of Relay Module.

Power Distribution

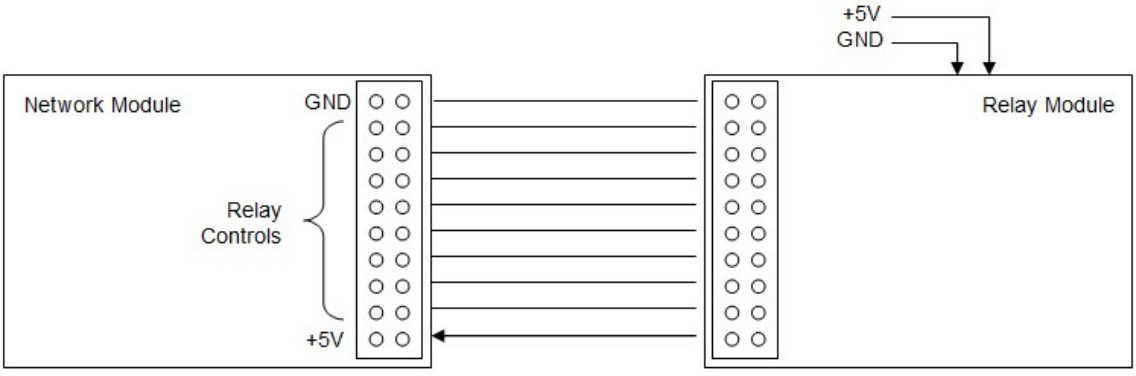
The first thing to consider is supplying power to the relay modules. The basic design of the Network Module is intended to provide +5V power to the relay modules via the pin header that also provides the relay control signals. This works well for just a few relays (up to 3 or 4). This connection method is illustrated here:



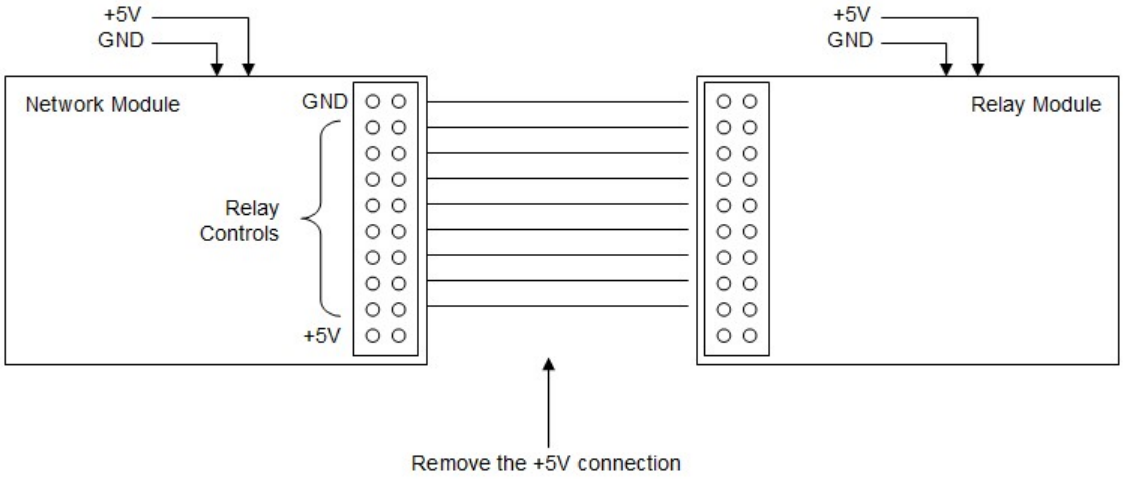
If you attach more relays you need to make sure that there is sufficient current supplied by your +5V power supply attached to the Network Module AND you need to make sure the method used to send power to the relay modules is adequate. This is particularly important if you are transferring power via a ribbon cable.

If you don't think you can provide adequate power to the relay modules via the Network Module relay header you can consider a couple of options:

1) Connect +5V power only at the Relays, and let the power/signal header send +5V back to the Network module.

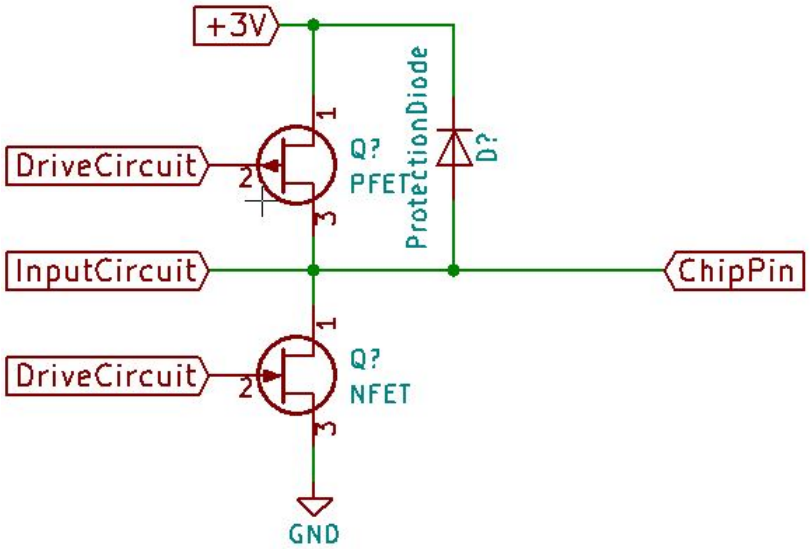


2) Use separate +5V power supplies on the Network Module and Relay Modules. If you do this you'll need to disconnect the +5V power connection between the headers.



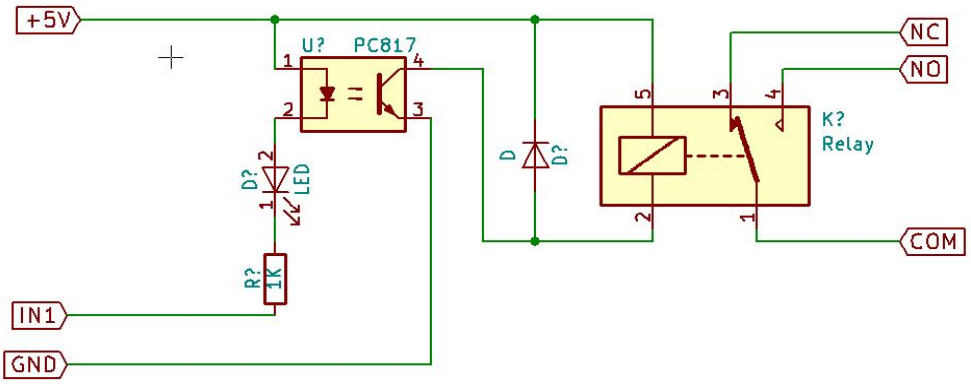
Type of Relay Module

The second consideration is the type of relay modules you attach. The SM8S processor on the Network Module operates at 3V and its outputs are connected directly to the relay control header. So, you need to avoid inadvertently causing +5V feedback from the relay modules to the 3V output pins of the processor that exceed the processor specifications (check the spec, but the short version is: Max 3.3V and/or limit to 4mA per pin, AND limit to 20mA across all pins). The reason this is a concern is because the SM8S output pins have overvoltage protection diodes that can provide a current path if a voltage higher than 3.3V appears on the pin when it is not in an active pull-down state. To visualize this here is a drawing illustrating the output pin:



Focus on the Protection Diode. There is also a protection diode to ground, but it is not a concern in this discussion so I left it out. If any of the relay modules can provide a current path from a higher voltage through the chip pin (when the pin is not pulling down) then there is the potential for damage. Knowing this let's look at typical relay module designs.

a) **Opto-isolated relay boards:** If you use opto-isolated relay boards there should not be a concern as long as the relay boards are designed to operate at a voltage no higher than 5V. The typical design of these relay modules looks like this:



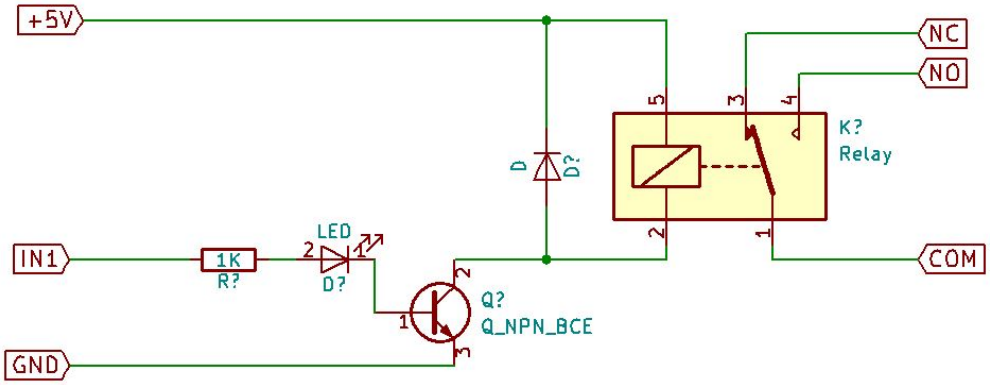
Note that in fact this relay module can provide a current path from +5V, through the photo emitter diode of the opto-isolator, through the visible LED, through the 1K resistor, then to the SM8S output pin via the "IN1" connection. But this will still work and here is why:

- The difference in voltage from the 5V supply to the SM8S output pin is 5V – 3V. But about 0.7V is dropped across the photo emitter diode. Then another 0.7V is dropped across the LED. And about 0.3V is dropped across the protection diode in the SM8S. The result is that there is only 5 – 3 – 0.7 - 0.7 - 0.3 = 0.3V potential across the 1K resistor. This will result in about 300 uA of current flowing through the path. This is not enough current to damage the SM8S and not enough current to cause the relay module to operate. So while not ideal it works.

- If your relay module does not have the LED in the trigger signal path as shown in the drawing above it might still work, but you'll have to test it to verify. The difference is that the 0.7 volt drop across the LED is missing from the equation so about 1mA will flow into the output pin of the SM8S. That won't hurt the SM8S, but it might cause the opto-isolator to operate in turn preventing the relay from releasing or causing the relay to release intermittently.

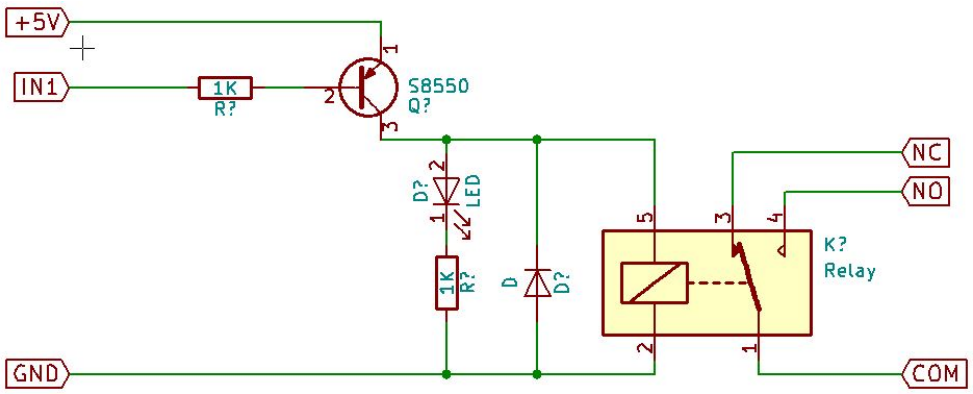
b) **Non-isolated relay module, Active HIGH trigger signal:** Some relay modules do not have opto-isolators. If they are of a design that has an active high trigger signal then the typical design has a 1K ohm resistor feeding the base of a NPN transistor. This type of relay module should operate just fine when connected directly to the Network Module, although you'll find that the logic seems reversed and you may have to set or clear the "Invert" function in the Relay Control page of the GUI.

A typical active-high relay module circuit design:



The reason this module works with the Network Module is because it has no path from +5V back to the SM8S output pin..

c) **Non-isolated relay module, Active LOW trigger signal:** This is another relay module design that does not have opto-isolators. This design typically has an active low trigger signal, and the typical design has a 1K ohm resistor feeding the base of a PNP transistor. A typical relay module design looks like this:



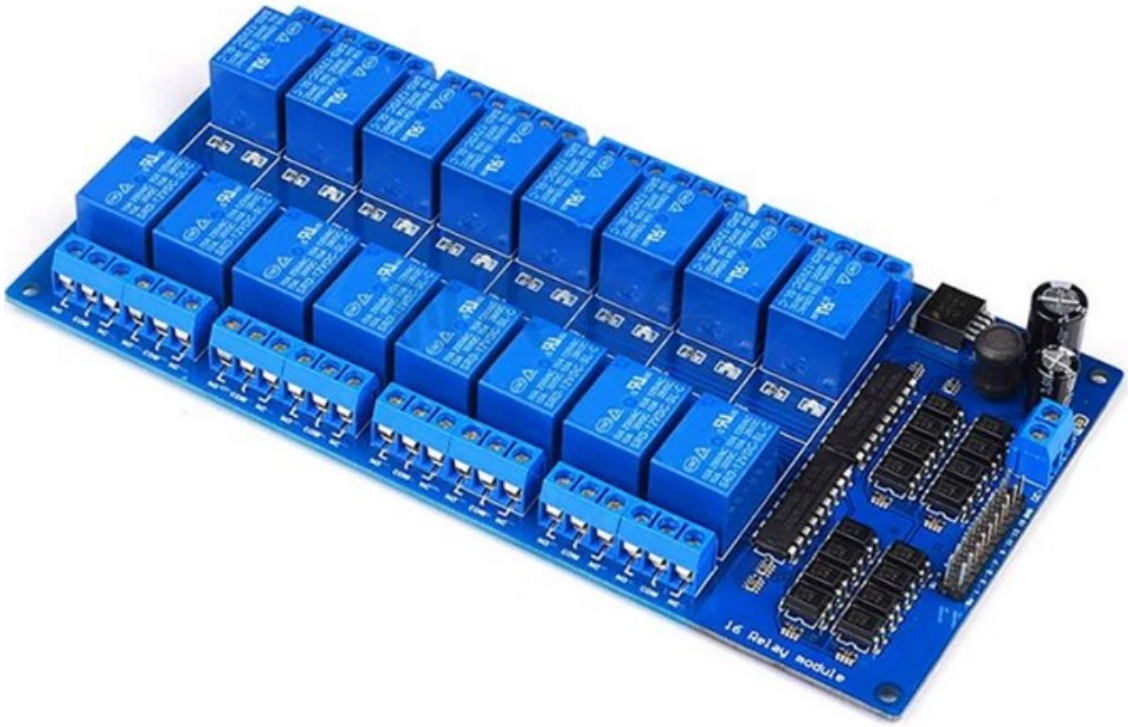
This design is problematic in that the PNP transistor is connected to 5V, and when the Network Module control signal goes to a high state a reverse current flow (also known as an injected current flow) will travel from +5V through the PNP transistor, through the 1K resistor, and into the SM8S output pin. Analyzing this path:

- The difference in voltage from the 5V power supply to the SM8S output pin is 5V – 3V. About 0.7V is dropped across the PNP transistor, and about 0.3V is dropped across the protection diode in the SM8S. The result is that there is 5 – 3 – 0.7 - 0.3 = 1V potential across the 1K resistor. This will result in about 1mA of current flowing through the path. This is not enough current to damage the SM8S, but it is in the active region of the PNP transistor. This may not allow the relay to turn off – or the relay may operate intermittently. If this is the case and you are unable to get a more compatible relay module you will need to provide a voltage shifting buffer between the Network Module and the Relay Module.

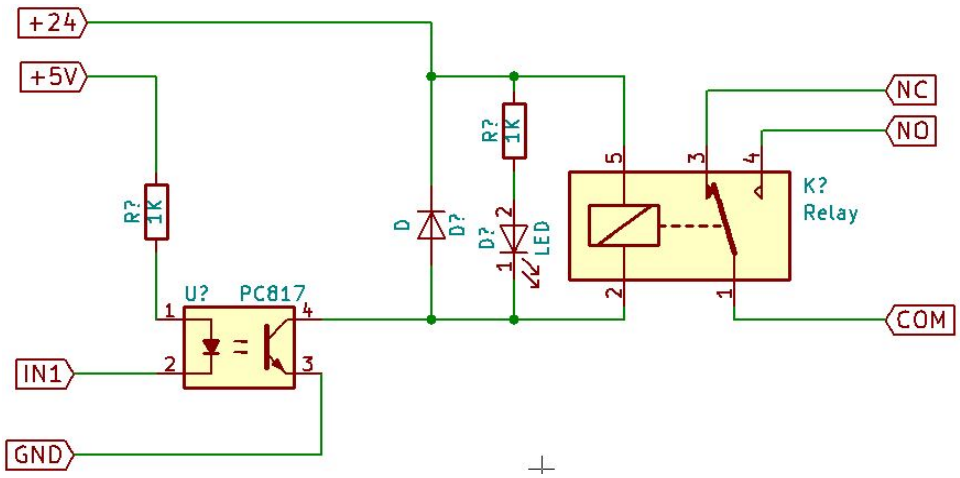
- If the relay module you have places the LED in series with the PNP transistor the module may work better due to the voltage drop across the LED. However, there may still be enough current to cause the PNP transistor and the relay to operate intermittently. All you can do is give it a try.

16 Channel Relay Modules

A user that bought 16 Channel opto-isolated boards reported problems, and on investigation it was found that the input circuit did not match what had been seen on 1, 2, 4, and 8 channel boards (even from the same supplier!). On further investigation we found that all the 16 Channel boards we could find online match this unique implementation. The type of board is shown here:

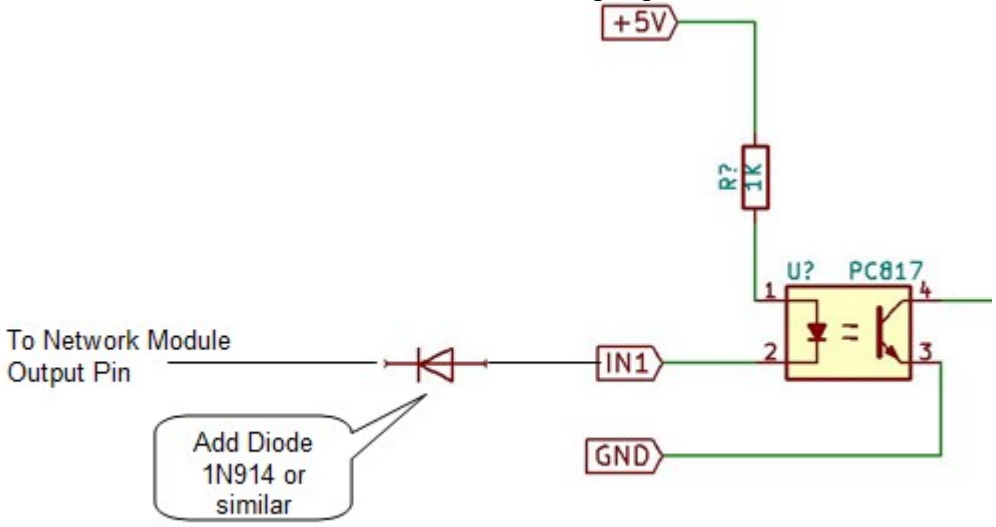


With some circuit tracing the circuit design on each channel appears to be as follows:



While the particular board this user had utilized 24V relays it appears that the board also comes in 5V and 12V relay versions. But the important thing to note is that the opto-isolator input circuit does not include an LED like the boards discussed in the previous section. So, less voltage is dropped between 5V and the STM8 output pin, and the current path through the output pin of the STM8 (via its internal protection diode) continues to conduct enough current to operate the opto-isolator even when the ouput pin is set to a logic 1 condition. This is a major problem in that the opto-isolator will remain activated regardless of the state of the output pin on the Network Module.

A possible solution to this problem is to add some additional resistance in series with the input pin, or better is to add a diode in series with the input pin as shown here:



The diode replaces the voltage drop that the series LED provides on most other relay module designs and prevents the current path through the STM8S pin protection diode

from keeping the PC817 in its active region (re-read (a) in the section above for more information). I am not sure if all 16 channel relays boards are designed this way, but all the ones found in a search on April 9 2021 appear to be as discussed here.

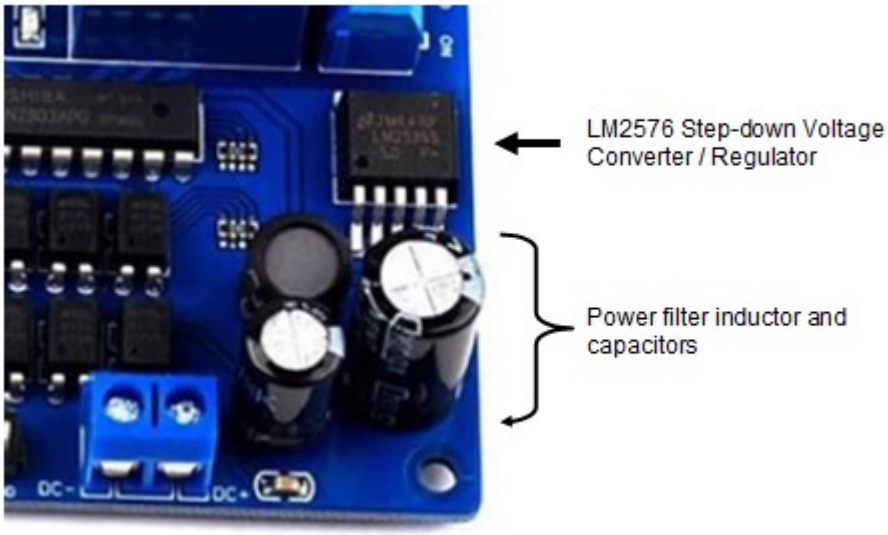
A Better Fix #1

Adding a diode in series with every input looks fine in theory but is very messy in practice because you may need to add up to 16 diodes. Here's a better idea: Put a single diode in series with the +5V path from the relay module +5V to the resistors that power all the opto-isolator devices. This requires a trace cut and insertion of the diode.

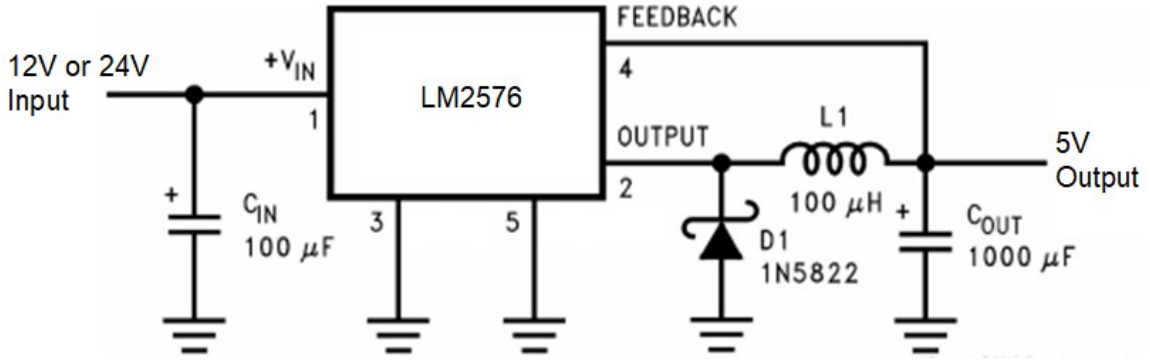
Detailed Modification Instructions

Instructions for modifying a 16 Channel relay board to be compatible with the Network Module. These instructions add a diode to the 5V regulator path that supplies power to all the opto-isolators.

The relay board with 12V and 24V relays will have a down-converter power supply on the board. This converts 12V or 24V to 5V for use by the PC817 devices and to optionally provide power to an off-board controller that requires 5V. The circuit components on 12V and 24V boards look like this:

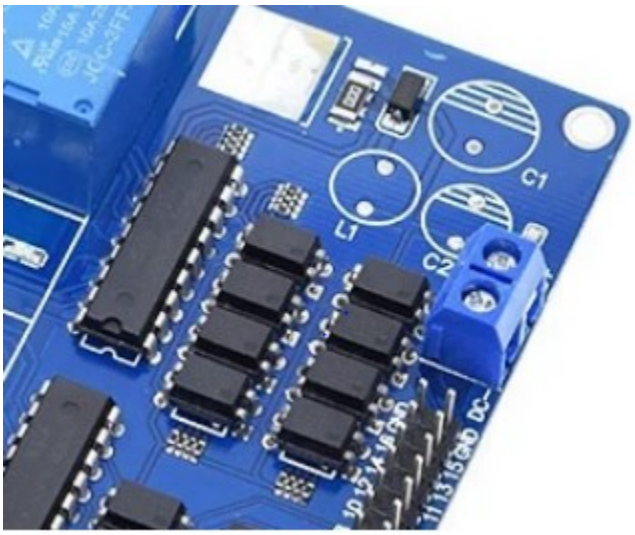


The circuit traces and component orientation on each board is a little different, but they all appear to be based on this circuit schematic:



Note that the actual component values for Cin, L1, and Cout may be different from your board. The way to tell the difference between Cin and Cout is that Cin will have a higher voltage rating marked on the part, and Cin will be physically larger than Cout.

The 16 Channel circuit boards with 5V relays do not have the down-converter components.



Referencing the circuit diagram and the above photo:
Diagram Cin is C1 in the photo
Diagram Cout is C2 in the photo
Diagram L1 is L1 in the photo

Looking at online photos from various vendors I can see that there are differences in component orientation and trace layout.

Before beginning modifications it is important to locate the 5V output side of L1 and C2.

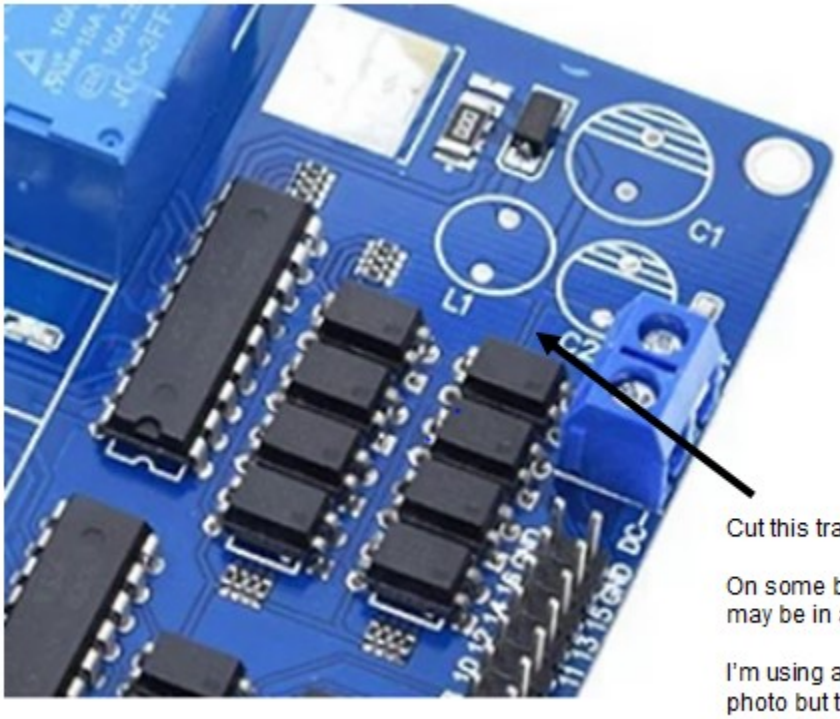
In the above photo the 5V connection to L1 is on the component pin closest to the PC817 devices. But I've noticed 5V is on the component pin furthest from the PC817 in at least one layout, and in another case L1 and C1 are in a completely different location.

Do your best to look at the circuit traces to find which side of L1 is connected to the positive side of C1. For reference, the negative side of C1 is marked like this:

So, find C1, find the positive side of C1, look at the board traces to find the positive side of L1 (connected to the positive side of C1). Put a very small mark on the bottom side of the board marking the positive side of both L1 and C1 (you'll need this marking later).

Now for the modifications:

1) Cut the 5V trace that runs under the row of PC817 devices at the location in this photo. In some layouts the trace runs under the other row of PC817 devices. If in doubt scrape some of the coating off the trace and measure the ohms from the trace to the 5V pins you identified on L1 or C2.



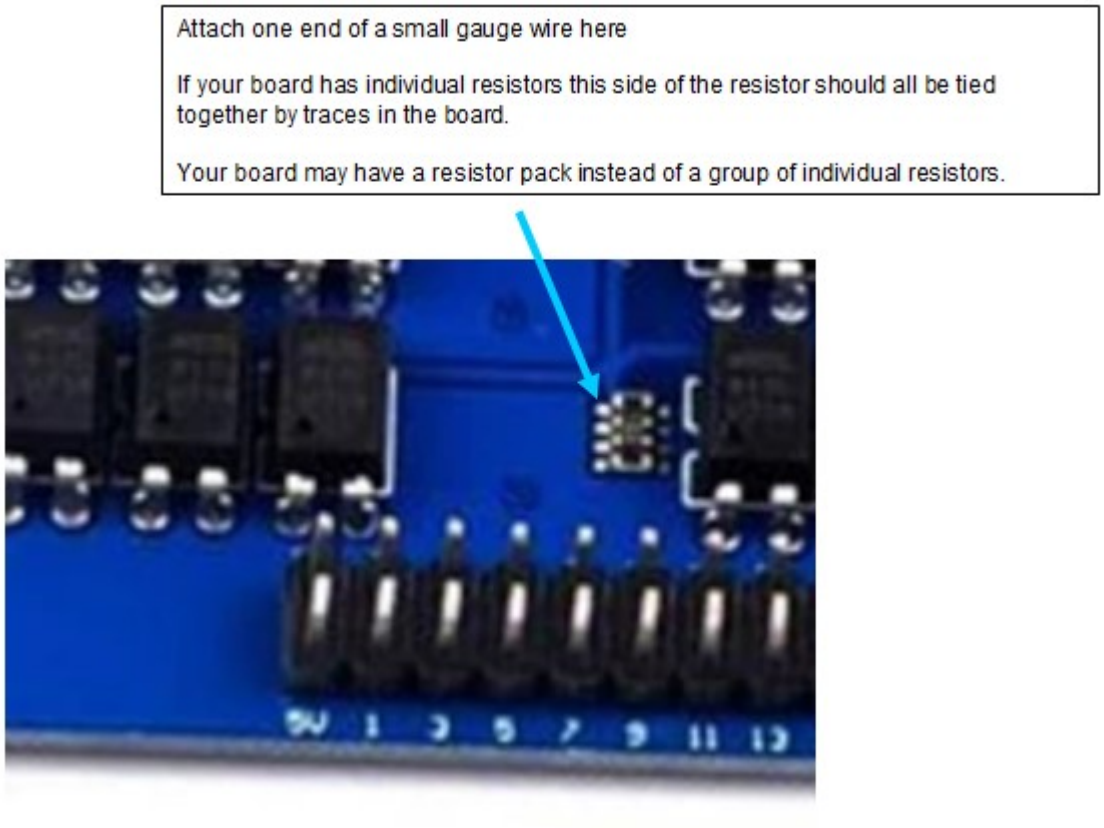
Cut this trace here.

On some boards the trace may be in a different location.

I'm using a 5V board in the photo but the same trace needs to be located and cut regardless of 5V, 12V, or 24V.

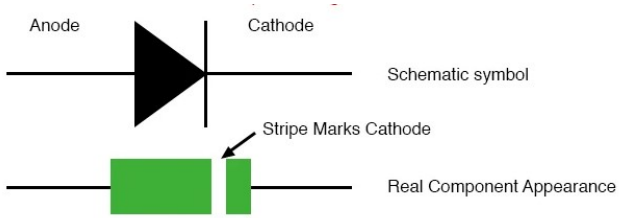
2) Cut the 5V trace to the 5V pins on the header as shown here. I haven't identified where the trace is because it is in a different place in each of several layouts I looked at. The 5V trace will be wider than the signal traces. In some layouts the trace runs up from the pins toward the PC817 devices, in another layout I see it running out to the left of the pins, and in one layout it is on the bottom of the board.

3) On the top side of the board attach a 4 inch small gauge wire to the 5V side of one of the resistor packs or resistor groups that provide pull ups to pin 1 of the PC817 devices. The wire will carry a max of 60ma so 30 gauge wire is adequate. Some boards have 4 surface mount resistors, some have a resistor pack. For example:



The wire can be attached to any one of the groups of resistors. Be careful not to heat up the resistor(s) to the point that it falls off the board.

4) On the bottom of the board solder the anode end of a 1N914 (or similar) diode to the side of L1 that you identified as the 5V output side. Keep the cathode lead fairly short (between 1/4 and 1/2 inch).



5) Solder the loose end of the wire that you installed in step 3 to the cathode end of the diode. Shorten the anode lead of the diode to about 1/4 inch. Shorten the wire to minimize excess. Once soldered use glue to secure the wire and diode to the board surface. Do not use RTV as it may be conductive. Find a non-conductive glue like airplane cement or similar. Make sure the bare connections on the diode are not able to make contact with any conductor on the board.

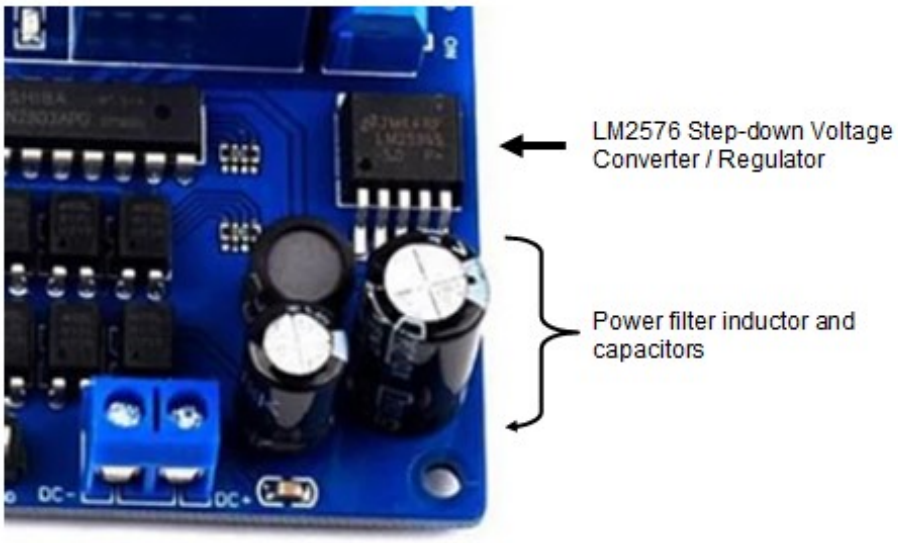
6) On the bottom of the board attach a wire from one of the 5V pins on the header to the C1 pin that you previously determined to be the 5V output. 30 gauge wire is adequate if you are only powering a Network Module. If you are powering greater loads consider a slightly larger wire. If you are not using the 5V pin on the header you do not need to add this wire. Here is the current rating of various gauge wire:
   AWG #30 0.5A
   AWG #28 0.7A
   AWT #26 1.0A
   A smaller wire will be easier for you to attach to the board.

That's all you need to do. Reminders:

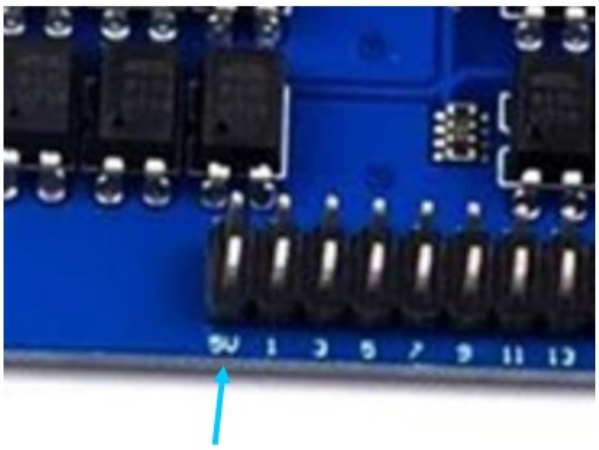
a) Determine which pins on L1 and C1 are the 5V output
b) Cut the 5V trace that feeds power to the PC817 pull-up resistors
c) Cut the 5V trace to the 5V pin on the header
d) Install diode
e) Connect 5V to header

A Better Fix #2

Another user took the approach of replacing the LM2576 5V regulator with a 3.3 volt version. This requires some careful surface mount soldering, but the leads are relatively large so it is doable if you are careful.



VERY IMPORTANT: This method will put 3.3V on the “5V” pins of the Relay Module header.



You need to make sure the 3.3V pins (marked 5V) on this header DO NOT get connected to the 5V pins on the Network Module header.

a) If you are using ribbon cable connections do not connect these pins on the headers.
b) If you are directly connecting the Network Module to the Relay Module with the headers you will need to:

i. Cut the trace on the Relay Module that goes to the pins marked 5V on the header
ii. OR cut the pins marked 5V off of the header on the Relay Module
iii. OR use some other method to make sure you don't connect these pins between the Relay Module and the Network Module.

Notes on Inputs

If you configure the Network Module to include digital inputs you'll need to be careful about the voltage you put on the input pin. The pins are directly connected to the SM8S processor. The processor operates at 3V, so you'll need to limit the high level voltage applied to the pin to 3V, or limit the current to no more than 1 mA.

Each input pin has a weak pull-up applied internal to the SM8S processor. The pull-up has a typical resistance equivalent of 60Kohm, but can range from 30Kohm to 80Kohm.

Some recommendations:

1) If you are using 3V logic to drive the input pin you should be able to directly connect it.
2) If your driver circuitry might place more than 3V on the input pin you can do one of the following:

    a) Use open collector devices or level translators to prevent putting more than 3V on the input pin.
    b) Use relay contacts to ground the input pin, relying on the SM8S pull-up to take the pin high. This might not be adequate if the wiring to the input pin is long or is subject to electrical interference.
    c) Put a 1Kohm resistor between the driver logic and the input pin, but be sure the driver cannot exceed 5V. This isn't ideal, but should limit any current driven into the SM8S to an acceptable level and still achieve adequate logic levels at the SM8S input.

Hardware Design to Maintain Relay States Through a Power Loss or Reboot

A user wanted to know how to prevent relays from changing state during a power loss on the Network Module. This question is very dependent on the whether the relays remain powered up during the power loss on the Network Module, AND it is dependent on whether the control input to the relay is active low or active high. So let's explore why things happen and what you can do about it.

First of all, be aware that when the Network Module loses power its outputs go to a low level signal. I'm sure this makes sense to you: no power, no signal output. But there is the additional consideration that the overvoltage protection on the device pins (effectively a diode to VCC) will look like a pull-down when VCC on the STM8S processor goes to zero. Also be aware that when the SM8S processor powers up it defaults all IO pins to a "floating input" state. This is a function of the chip design, so it can't be changed.

Regardless of what you define as ON or OFF, this discussion here is purely from the perspective of the output signal levels on the Network Module and the input type of the relay being driven.

Scenarios and what happens:

Scenario 1:

a) Assume relays remain powered up while the Network Module is powered down.
b) Relays are active high inputs (a high level signal activates the relay).

What will happen:

- When the Network Module loses power and then reboots any relay that is inactive will remain inactive if Retain was enabled in Configuration.
- When the Network Module loses power and then reboots any relay that is active will go to an inactive state during Network Module power loss, then the relay will return to an active state if Retain was enabled in Configuration.
- Note: If the relays also lose power they will of course go inactive during power loss.

Scenario 2:

a) Assume relays remain powered up while the Network Module is powered down.
b) Relays are active low inputs (a low level signal activates the relay).

What will happen:

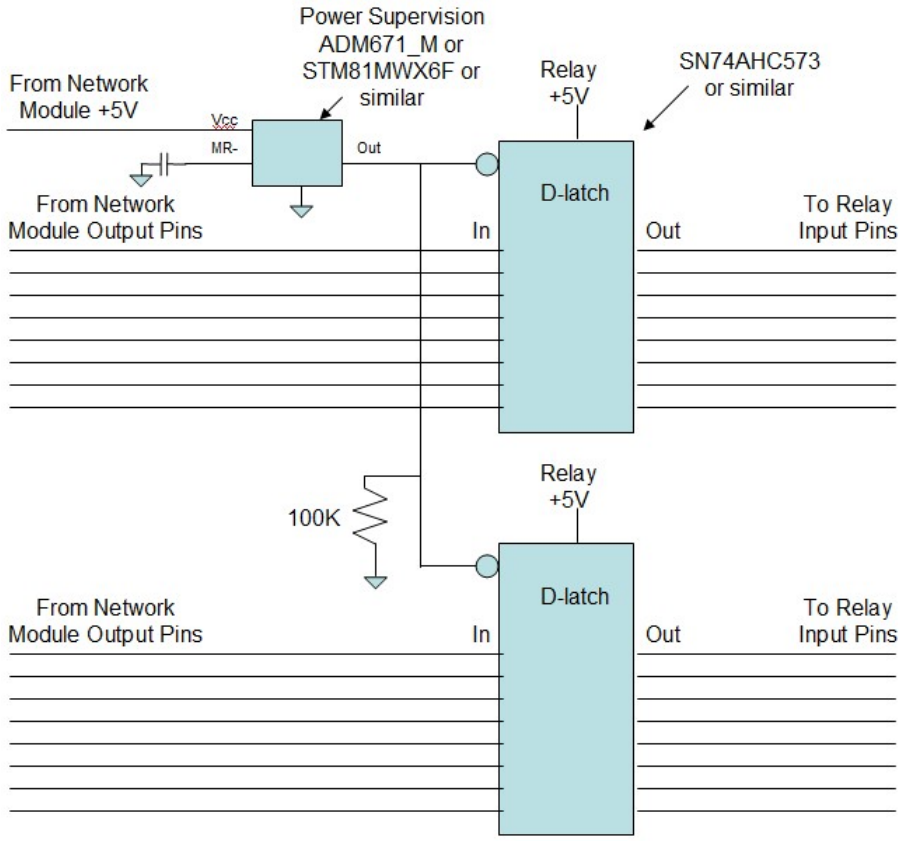
- When the Network Module loses power and then reboots any relay that is inactive will go to an active state because the Network Module outputs go to a low state

during power loss, then the relay will return to an inactive state if Retain was enabled in Configuration.
- When the Network Module loses power and then reboots any relay that is active will remain in an active state if Retain was enabled in Configuration.
- Note: If the relays also lose power they will of course go inactive during power loss.

So let's say you don't like the above scenarios and you want the relays to stay just the way you set them through a power loss, regardless of whether they were active or not. Well, this can only be done with hardware external to the Network Module. Here are some suggested solutions:

Option A:
- Assume the relays remain powered up while only the Network Module loses power.
- The relays can be either active low or active high inputs.

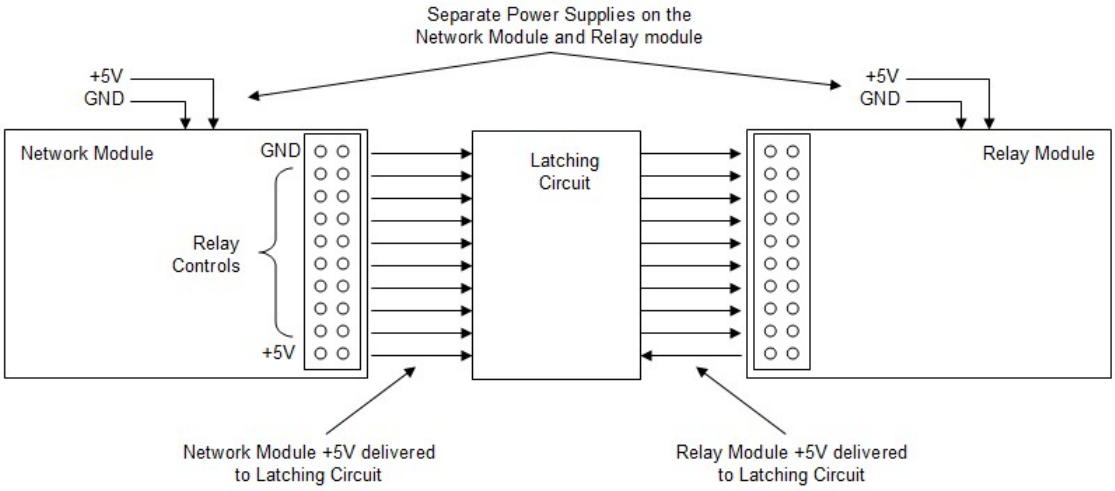
Since we know the Network Module outputs will go low during power loss we need to build hardware that will maintain the state of the relay control input during the outage. An example circuit is shown here for 16 Relays:



In the above circuit the Power Supervision device will detect when power is falling on the Network Module and will cause the D-latch devices to capture the output states of the Network Module. As long as power is maintained on the Relays and the D-latches the relays will maintain their state.

Once power returns on the Network Module the Power Supervision device will continue to keep the D-latch devices in a hold state until the STM8 processor is operating. If additional time is needed a capacitor can be added to the MR- input.

The latching circuit described above is placed between the Network Module and the Relay Module as shown here:



The downside to the above: The devices in the Latching Circuit may not all be available in DIP form – some may be surface mount. This implies that you may have to design a circuit board for this solution. That is not as onerous as you might think. There are cheap manufacturing sources in China that will make up to 10 circuit boards for almost nothing (less than $10 USD). But you must design the board and generate gerber files.

Option B:

- Use “Latching Relays”

This option almost sounds like nirvana until you look at the details.

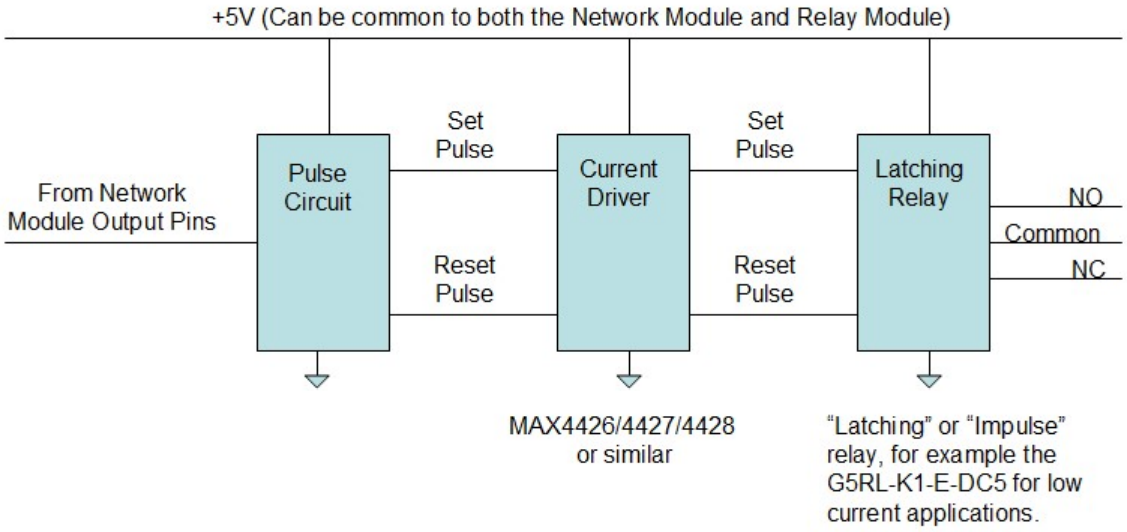
First let me caution you about “bi-stable self latching” relays that you may find on eBay. Those are meant to be provided with a pulse and they will switch and hold their state. BUT, you can’t tell what state they are in remotely, so not very useful for remote

applications. It might be possible to tap into their circuitry and find a sense point that could be fed back to one of the Network Module sense inputs. But remember these bi-stable relays are not actual latching relays as they require that power be maintained to the relay for it to retain it's state (just like Option A above)..

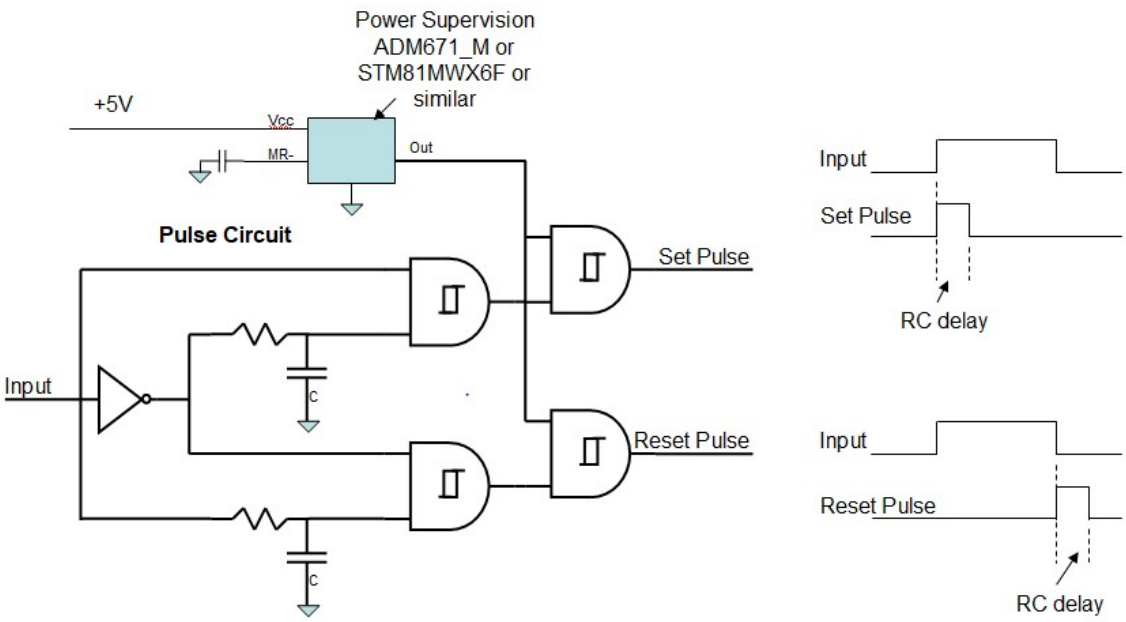
So, what about REAL latching relays? They will maintain their state even if ALL power is lost to the Network Module and the Relay Module. These relays are sometimes called "Impulse" relays because they require a pulse on a Set pin to put them in one state, and a pulse on a Reset pin to go to the other state. An example of this type of relay is the G5RL-K1-E-DC5 for low current applications (Google it).

Since the Network module only has a single state output a circuit is required to convert that signal to the pulses needed by the latching relays. And a driver is needed to supply the necessary pulse current to the relays. AND the circuit still needs to detect power loss to prevent inadvertent pulse generation when power is lost and restored.

A block diagram looks like this:



The pulse circuit looks like this:



The above needs to be repeated for each relay (except the Power Supervision can be common).

The resistor and capacitor values are determined as follows:

Need minimum pulse width of 30ms
When input is rising the output will switch at about 2.6v
When input is falling the output will switch at about 1.8v
Need about 60K and 1uF for rising signal. This will give an RC of about 60ms, with a pulse width of about 40ms.
Using 60K and 1uF for the falling signal will have a pulse width of about 60ms.

As in Option A a capacitor can be applied to the MR- input of the Power Supervision if a longer delay time is needed for the STM8 to stabilize.

Suggested part number for the Schmitt NAND gate is the SN74HC7001 and for the Inverter is the SN74LVC1G04.

After considering the Pulse Circuit, Power Supervision, and Current Drivers once again the circuit is complicated enough to require a circuit board.

Option C:

You may want to consider using a UPS to prevent power loss. That will solve most of the problems discussed here. If power loss is nearly non-existent, then the only concern is brief relay chatter in event of a reboot, which should also be nearly non-existent once the Network Module is set up.

IMPORTANT: I have not implemented any of the above power loss circuits, so there may be errors in what I've described. If you plan to go this route analyze the design carefully.

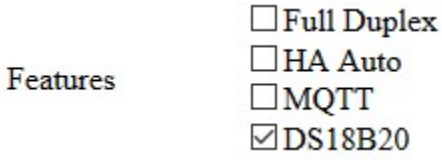
Summary:

In the end you need to consider carefully if fully retaining the relay states all the way through a power loss is really necessary. You can use the Retain setting in the Network Module to be sure the relays return to their pre-power loss state, but they may still "chatter" once or twice during reboot or power loss and recovery.

Adding DS18B20 Temperature Sensors

Code was added to allow you to use IO 16 (pin 16) for DS18B20 Temperature Sensors. You can attach up to 5 DS18B20 sensors to this pin.

In the Features section of the Configuration page you'll see a checkbox for DS18B20.



If you check this box IO 16 will show as "Disabled", and in fact the pin IS disabled for use as an Input/Output pin. But you can now attach up to 5 DS18B20 temperature sensors to the pin and have the temperature seen at these sensors displayed in the IOControl page. If you also check MQTT the temperature is Published on MQTT. And, if you also check HA Auto the temperature sensor will be Auto Discovered in Home Assistant.

Diagram for attaching one sensor:

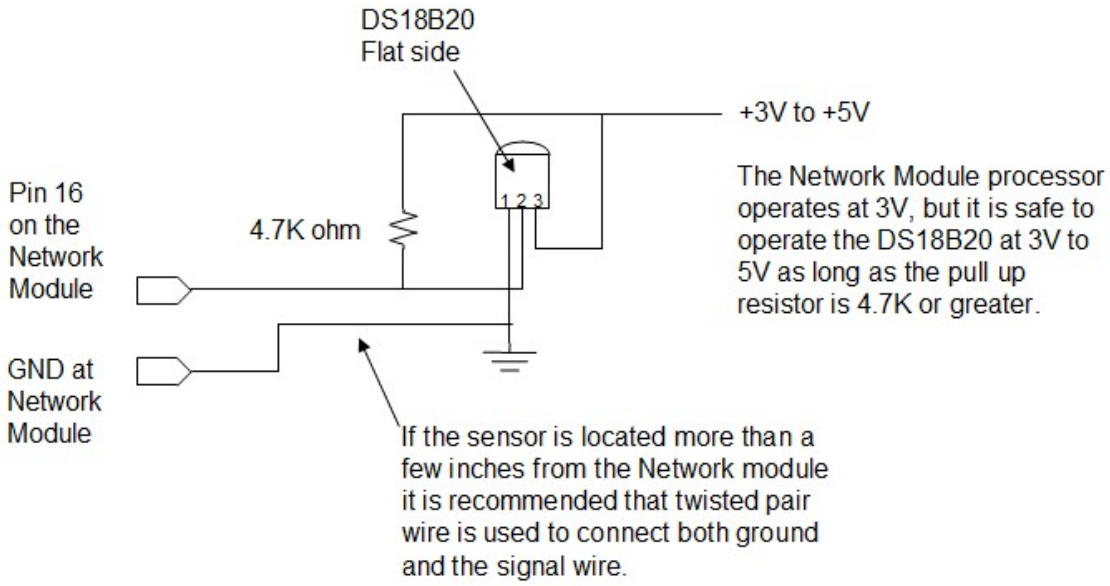
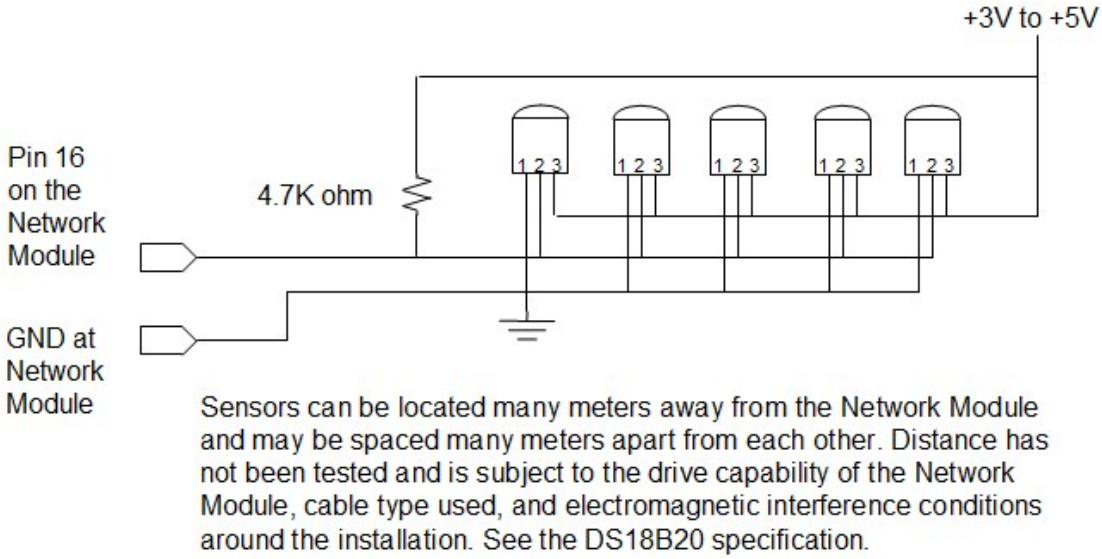


Diagram for attaching multiple sensors:



On power up the temperature display may show a “filler” value of ------°C degrees for each sensor. Likewise, if one of the 5 sensors is missing it will show the filler value. As soon as communication with the sensor is established the value will be as reported by the sensor. If the sensor fails or becomes disconnected after initial communication is established the reported temperature will be -000.1°C until the firmware discovers that the sensor is missing or unresponsive. Once the sensor is replaced or re-connected the temperature reported will require 1 to 2 minutes for the DS18B20 to begin reporting correct values. In the meantime the reported values may be indeterminate.

The sensors appear in the display in the order of the unique 48-bit Serial Number contained within each DS18B20. All 48 bits are displayed as the Temperature sensor ID (6 bytes shown as 12 hex encoded characters). But there is a little bit of a catch: The serial numbers are read from the devices LSbit first, so the device discovery and sorting algorithm is based on LSbit to MSbit, one bit at a time. However, the display of the values is shown MSByte on the left to LSByte on the right. So, they might not appear to be in some sort of ascending order, but in fact they are.

Since the serial numbers are sorted as described above they will always maintain their order in the Browser display. Thus if a sensor fails and is replaced the new sensor might be inserted in the list at any point (again, based on the LSbit to MSbit sorting). But a given sensor’s ID will always remain the same.

Note that Home Assistant sorts the sensors in ascending order based on the MSByte to LSByte order. So, Home Assistant will display the sensor ID’s in a different order than the Browser.

Developers: Setting Up a Development Environment

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code. You don't need to do this if you are going to use the binaries (.stp and .sx files) already provided.

If you want to change the code for your own use I assume you have some experience with coding and the tools typically involved. I used the tools described in the previous sections for actual programming of the device, and used the Cosmic tools for the development environment. To duplicate this you'll want the following:

1) **Download and install the Cosmic Compiler:** Use the one that is specifically for the STM8 devices. Start at this website
   https://www.st.com/en/development-tools/cxstm8.html#product-details
   Click on Product Details and follow the link to the "partner website". From there you can download the compiler. The compiler is free. They will send you a 1-year license, but I think you can renew over and over. Note that the license is specific to the machine you install it on.
   As an FYI, even though my PC is x64, the tools installed in this directory:
   C:/Program Files (x86)/COSMIC/FSE_Compilers/

2) **Download and install the following library from st.com:**
   en.stsw-stm8069.zip You'll find it at
   https://www.st.com/en/embedded-software/stsw-stm8069.html
   NOTE: I included this library in the files included with the project so you may not need this step if you copy all the files from GitHub. This is the STM8S_StdPeriph_Driver directory.

3) **Copy the Program:** With the above installed the next step is to copy the entire project from GitHub into your Documents directory. On my Windows 10 machine the project was located in the following directory:
   C:/Users/Mike/Documents/COSMIC/FSE_Compilers/CXSTM8/NetworkModule
   Of course you will likely have a different user ID.

Start the Cosmic tools by double clicking on the NetworkModule.prjsm8 file. You should be on your way.

A note about my coding style: My coding is not particularly esoteric or convoluted. I try to keep it simple to read and understand even if that is less efficient. And I put a lot of

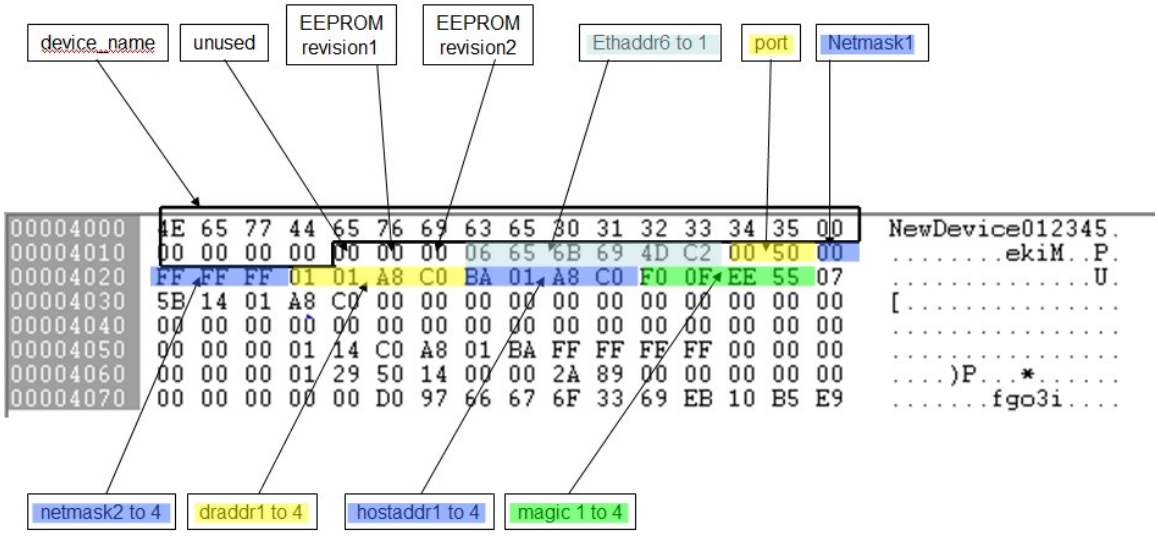
comments in, particularly if I had to do things to make the code work that didn't fully make sense to me. Sometimes that stuff happens and my intention is to come back and look at it again later. So, feel free to modify and "do it your way". I'm not proud as long as it works.

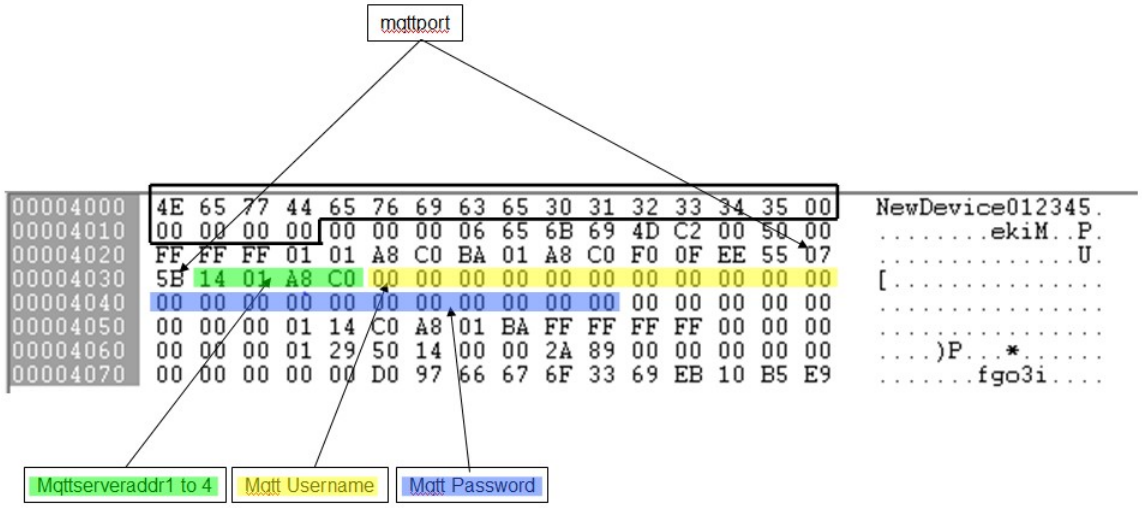
Developers: Location of EEPROM Variables

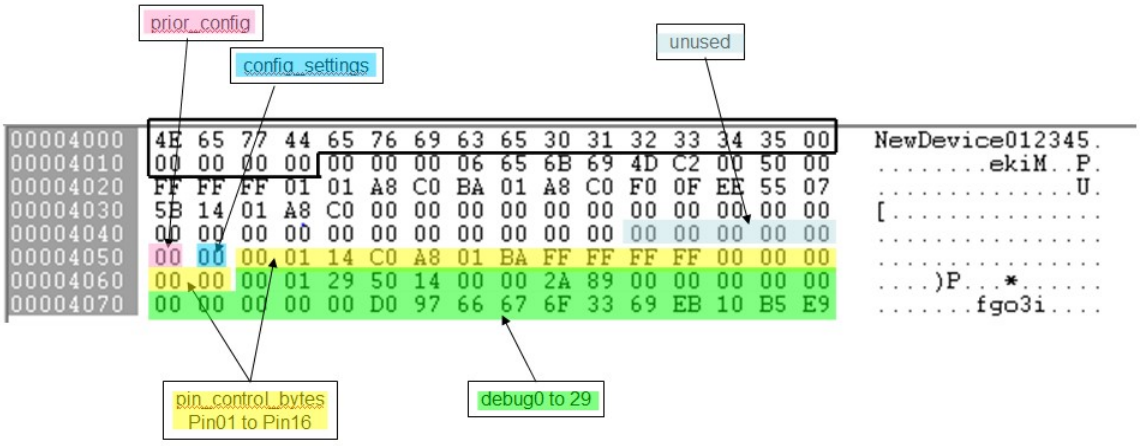
NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

You can view the contents of the EEPROM with the STVP programmer by selecting the "Data Memory" tab and using the "Read / Current Tab" function. The displayed information has the following definitions.

Note that the data in some fields is "left to right", a more human readable direction. For instance the device_name field. Some fields are "right to left", for instance the hostaddr field.







The debug bytes are defined by the developer to provide non-volatile storage of any information the developer needs to debug code function. Some routines are already present in the source files to help with capture of debug information.

Developers: Notes on Debug Bytes

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

THIS INFORMATION MAY NOT BE ACCURATE if the number of debug[] bytes changes due to code changes. In particular the number of debug[] bytes may be reduced if more EEPROM is needed to support added functionality.

The code contains compile options for implementing various debug modes. This section describes the debug modes and how debug information is captured in EEPROM.

The general idea:

DEBUG_SUPPORT compile time options are determined by the #define DEBUG_SUPPORT statement in the uipopt.h file.

When code is compiled with any DEBUG_SUPPORT option debug data is stored in RAM locations named debug[].

Each debug[] byte has a corresponding storage location in EEPROM named stored_debug[].

The data in the debug[] RAM locations is copied to EEPROM when the function update_debug_storage1() is called, or when a function requires specific debug[] bytes to be stored in EEPROM. In some functions any change in a specific debug[] byte will cause it to be stored in EEPROM.

Note that ANY time a debug[] byte is stored in EEPROM it is first compared to the byte already stored in EEPROM, and the EEPROM write only occurs if the bytes do not compare. This is to prevent excessive writes to EEPROM. Example code:

```
for (i = 0; i < NUM_DEBUG_BYTES; i++) {
  if (stored_debug[i] != debug[i]) stored_debug[i] = debug[i];
 }
```

IMPORTANT: This “compare before write to EEPROM” concept should be used any time a write to EEPROM of any kind is to occur. This will help prevent excessive wear on the EEPROM, particularly when a coding error is made that might cause a frequent write to EEPROM.

ALSO IMPORTANT: The EEPROM must be unlocked prior to any write (see function unlock_eeprom()). After the write the EEPROM should be locked again (see function lock_eeprom()). This helps prevent coding errors from inadverently writing EEPROM.

When code is compiled with DEBUG_SUPPORT == 1

All debug[] bytes are available to the developer to be used as needed, and they are copied to EEPROM when the function update_debug_storage1() is called.

When code is compiled with DEBUG_SUPPORT == 7, 11, OR 15

The last 10 bytes of debug[] (and stored_debug[]) are reserved for some specific debug information. The last 10 debug[] bytes are stored in EEPROM whenever the debug[] byte content changes. The debug[] bytes prior to the last 10 can be used just like the debug[] bytes are used when DEBUG_SUPPORT == 1.

!!!!! Leave DEBUG_SUPPORT 11 enabled for Production Code. While this seems inconsistent with the normal use of "debug", DEBUG_SUPPORT 11 enables the "Link Error Stats" web page. The Link Error Stats web page can be very useful to regular users that may need to diagnose whether Full Duplex is needed with their network switch(es).

In addition to the "update_debug_storage1()" function there are some specialized "copy debug[] to stored_debug[]" routines as follows:

update_debug_storage()

This function will copy the debug[] bytes to EEPROM only if debug[0] == 0x01. If the function finds that debug[0] contains 0x01 it will write 0x02 into debug[0]. This creates a "single snapshot" capability.

capture_uip_buf_transmit()

This function will capture a portion of the uip_buf when transmit data is present. The function can be modified to perform the capture when various triggers are present. There are some sample triggers in the code that are commented out.

capture_uip_buf_receive()

This function is similar to capture_uip_buf_transmit but contains triggers for capturing a portion of the uip_buf when receive data is present. There are some sample triggers in the code that are commented out.

capture_mqtt_sendbuf()

This function is similar to the other capture functions above except that it is designed to capture a portion of the data in the MQTT sendbuf.

#define DEBUG_SUPPORT table

Bit 0: Enable write to EEPROM for debug[] bytes
Bit 1: Last 10 bytes of debug[] allocated to:
Reset Status Register counters
TXERIF counter
RXERIF counter
Stack Overflow bit
ENC28J60 revision level
Bit 2: UART enabled
Will display the above plus other developer implemented information
Bit 3: Browser page enabled for Link Error Stats. Will display:
- Seconds counter (since last boot)
- Transmit counter (since last boot)
- Octets showing Stack Overflow, ENC28J60 revision, TXERIF counter, RXERIF counter
- Octets showing MQTT Response Timeout counter, MQTT Not OK counter, MQTT Broker Disconnect counter

Bits									
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Dec**	**Function**
0	0	0	0	0	0	0	0	0	No debug No UART No Link Error Stats browser page
0	0	0	0	0	0	0	1	1	debug[] bytes enabled - visible only via STVP No UART No Link Error Stats browser page
0	0	0	0	0	1	1	1	7	debug[] bytes enabled - visible only via STVP Last 10 bytes of debug[] allocated to specific debug data UART enabled No Link Error Stats browser page
0	0	0	0	1	0	1	1	11	debug[] bytes enabled – visible only via STVP Last 10 bytes of debug[] allocated to specific debug data No UART Link Error Stats browser page enabled **Set DEBUG_SUPPORT 11 for Production Code**
0	0	0	0	1	1	1	1	15	debug[] bytes enabled – visible only via STVP Last 10 bytes of debug[] allocated specific debug data UART enabled Link Error Stats browser page enabled

“visible only via STVP” means that the STVP programmer software should be used to display the “Data Memory” in order to view these bytes. See section “Location of EEPROM Variables”.

The table below defines the debug[] bytes dependent on the DEBUG_SUPPORT setting. Note that NUM_DEBUG_BYTES defined in main.h sets the total number of debug[] bytes available. If you make code changes that use up more EEPROM you will have to adjust NUM_DEBUG_BYTES in main.h,

debug[] Bytes Defined

DEBUG_SUPPORT == 1	DEBUG_SUPPORT == 7, 11, 15
debug[00] *	debug[00] *
debug[01] *	debug[01] *
debug[02] *	debug[02] *
debug[03] *	debug[03] *
debug[04] *	debug[04] *
debug[05] *	debug[05] *
debug[06] *	debug[06] *
debug[07] *	debug[07] *
debug[08] *	debug[08] *
debug[09] *	debug[09] *
debug[10] *	debug[10] *
debug[11] *	debug[11] *
debug[12] *	debug[12] *
debug[13] *	debug[13] *
debug[14] *	debug[14] *
debug[15] *	debug[15] *
debug[16] *	debug[16] *
debug[17] *	debug[17] *
debug[18] *	debug[18] *
debug[19] *	debug[19] *
debug[20] *	debug[20] **
debug[21] *	debug[21] **
debug[22] *	debug[22] **
debug[23] *	debug[23] **
debug[24] *	debug[24] **
debug[25] *	debug[25] **
debug[26] *	debug[26] **
debug[27] *	debug[27] **
debug[28] *	debug[28] **
debug[29] *	debug[29] **

debug[20] Not used; Available
debug[21] Not used; Available
debug[22] Bit 7 = Stack Overflow Error
Bit 6-4 = Not defined
Bit 3-0 = ENC28J60 revision
debug[23] ENC28J60 TXERIF counter
debug[24] ENC28J60 RXERIF counter

debug[25] EMCF: EMC reset flag count
debug[26] SWIMF: SWIM reset flag count
debug[27] ILLOPF: Illegal opcode reset flag count
debug[28] IWDGF: Independent Watchdog reset flag count
debug[29] WWDGF: Window Watchdog reset flag count

* Stored in EEPROM when update_eeprom_storage1() is called

** Stored in EEPROM each time a change occurs

Developers: Notes on Configuration Debug and pin_control Bytes

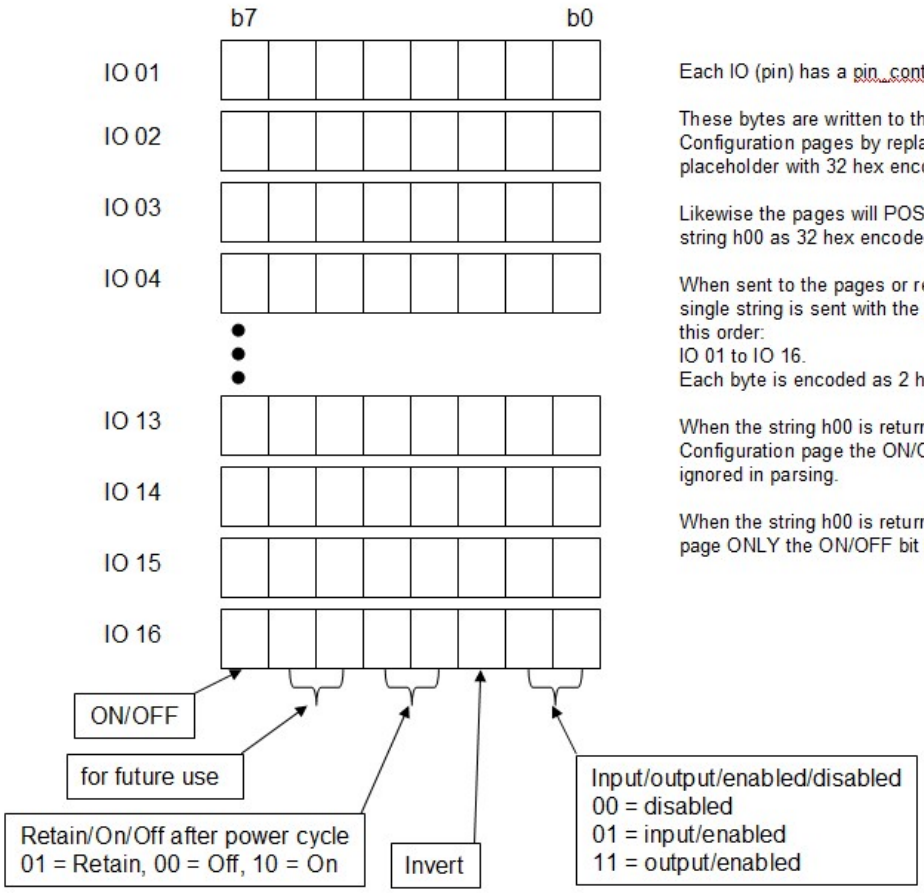
NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

The Configuration page contains a debug feature useful for development. If you go to the Configuration page, add #d to the URL, then click Refresh a column of numbers will appear to the right of the settings as illustrated here:

IO	Type	Invert	Boot state	
#1	input	☑		5
#2	input	☐		129
#3	input	☐		129
#4	input	☐		129
#5	input	☐		129
#6	input	☐		129
#7	input	☐		129
#8	input	☐		129
#9	input	☐		129
#10	input	☐		129
#11	output	☐	off	3
#12	output	☐	on	147
#13	output	☐	retain	11
#14	output	☑	off	7
#15	output	☐	off	3
#16	disabled			128

These values are the decimal equivalents of the content of the pin_control bytes in the code. The pin_control bytes are defined as shown here:

pin_control Bytes



Each IO (pin) has a pin_control byte.

These bytes are written to the IOControl and Configuration pages by replacing the %h00 placeholder with 32 hex encoded characters.

Likewise the pages will POST all 16 bytes in the string h00 as 32 hex encoded characters.

When sent to the pages or received in a POST a single string is sent with the pin_control bytes in this order:
IO 01 to IO 16.
Each byte is encoded as 2 hex characters.

When the string h00 is returned by the Configuration page the ON/OFF bit will be ignored in parsing.

When the string h00 is returned by the IOControl page ONLY the ON/OFF bit is examined.

Developers: Notes on Proto-Sockets

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

Most developers that build web server code are familiar with Windows or Linux systems which provide a “socket” interface. The “socket” basically maintains all pointers and parameters associated with a given network connection.

In a “bare metal” application like the Network Module there is no operating system providing the socket interface, so a bare-bones implementation is used and is typically called a “proto-socket” interface. The interface maintains only the very basic and essential pointers and parameters associated with network connections in order to minimize memory associated with the connections.

I didn’t write the original code used for proto sockets in this application, but I did modify it as needed. Here are some notes I took along the way in case they are useful to other developers and for future debug efforts.

struct uip_conn is a single structure identifying all the paramters of a single connection, and each uip_conn is connected by a union to a struct tHttpD, thus defining a "Proto-Socket". A Proto-Socket is really just the set of parameters that define the variables, pointers, and status of a given connection.

uip_conns[] is an array of the uip_conn structures, and the number of elements in the uip_conns[] array is defined by UIP_CONNS in the uipopt.h file. The uip_conns[] array of structures is statically allocated in the uip.c file with this statement:

```
struct uip_conn uip_conns[UIP_CONNS];
```

So, the structures are "pre-existing" and remain in RAM at all times. Note that anything added to the structure will occupy memory on a per uip_conn basis, so if I add say 10 bytes to the structure is will be multiplied by the UIP_CONNS value. UIP_CONNS is defined as 4 in this application, so adding 10 bytes to the structure would add 40 bytes of memory consumption.

Where is the content of the structure initialized? In uip.c, where the only real initialization is to set each connection to UIP_CLOSED as follows:

```
void uip_init(void)
{
  for (c = 0; c < UIP_LISTENPORTS; ++c) uip_listenports[c] = 0;
  for (c = 0; c < UIP_CONNS; ++c) uip_conns[c].tcpstateflags = UIP_CLOSED;
  /* IPv4 initialization. */
}
```

A connection is set up when a packet arrives over ethernet with the appropriate MAC, IP, and Port. If the Port matches one of the "listening ports" then the uip_process() code finds its way to the "found_listen:" code, an unused connection is located in the connection table, and the necessary connection parameters are initialized in one of the uip_conn structures contained in the uip_conns[] array.

For each connection I maintain all necessary parameters in the uip_conn structures with the exception of variables used in parsing POSTs. Of these variables the parse_tail[] array is the largest. parse_tail[] is used when parsing POST data from a Browser. parse_tail[] is very large and must be maintained across packet fragments in a POST. parse_tail[] is too large have a separate one for each connection ... so it is a usage requirement that the user must never POST from more than one Browser at a time. If they do the parse_tail[] will be corrupted and results will be indeterminate.

The other variables used in POST parsing are the “Pending_” variables. These provide temporary storage of user entered data POST parsing completes. There are many of these and there is not enough memory for them to be maintained in the uip_conn structures, so again it is a usage requirement that the user never POST from more than one Browser at a time.

In similar fashion the Code Uploader must only be run from one Browser as it is a "POST" mechanism.

Developers: #pragma, Sections, Segments and the .lkf File

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

I've included this section mostly so I don't forget how all this works.

There are several places where I've had to define additional "segments" where code is segregated for specific purposes. Those areas are:

a) Stack Overflow detection
b) Flash Update code
c) Copy RAM to Flash code

In general the reason that these segments must be defined in this application is to 1) enable the linker to provide warnings if segment sizes interfere with each other, 2) specify the location of specific code so that Flash updates will work without overwriting code that is being used, 3) enable special functionality to copy RAM content to Flash.

Editing the .lkf file

Defining and using code segments requires that you be able to edit the .lkf file. To enable editing the Linker .lkf file in IdeaSTM8 you must first edit the .prjsm8 file. Make sure the following line is in the .prjsm8 file:

LinkFileAutomatic=NO

When you first install IdeaSTM8 the default is for the program to auto generate the .lkf file each time a build is done. This is useful because it creates all the linker commands you typically need. However, when you reach the point that you need to make your own changes to the .lkf file (such as described below) edit access is disabled until the LinkFileAutomatic line is modified to "NO".

Stack Overflow Detection

Stack Overflow Detection is a part of the released code. In the main.c file the following code is used to declare two constants at the top of the RAM area. Regular variable assignments start at memory address 0x0000 and grow upwards to 0x5ff. Stack starts at 0x7ff and grows downward to 0x0600. Two constants are placed at 0x5fe and 0x5ff and are monitored to make sure they never change. If they do change it implies that the Stack

has grown into the variable storage RAM, or that a "wild pointer" may have caused writes to RAM to exceed the space allocated to RAM.

One way to locate RAM variables in a specific location is to simply use Absolute Addressing. For example:

```
uint8_t stack_limit1 @0x05fe;
uint8_t stack_limit2 @0x05ff;
```

Those two declarations would place the variables at addresses 0x05fe and 0x05ff. However, using that method does not let the compiler/linker validate whether its own variable placements also utilize those two memory addresses. So, using the #pragma and linker directives is the safer method to use.

The pragma code creates the two variables in a special section named ".iconst". In the main.c file a pragma section is created that looks like this:

```
#pragma section @near [iconst]
uint8_t stack_limit1;
uint8_t stack_limit2;
#pragma section @near []
```

The linker needs to be told where to place this section in memory. The following directive needs to be placed in the .lkf file. This directive will place the two stack limit variables at 0x5fe and 0x5ff. In the .lkf file specify the location of the pragma section like this:

```
+seg .iconst -b 0x5fe -n .iconst
```

Now the linker will be able to warn you if allocation of variables in RAM has enough space.

Flash Update and Copy-RAM-to-Flash Code

The “upgradeable” versions of the code provide the ability to upload new firmware to the Network Module and write it to the Flash on the STM8 device. This requires adding hardware in the form of an I2C EEPROM. Assuming that is done, the following process is followed by the firmware:

c) New firmware is received from the Browser and written to the I2C EEPROM.
d) The new firmware in the I2C EEPROM is copied to Flash.

In order for a) and b) to work the I2C driver must remain functional throughout the write to Flash. This requires that the I2C driver be located in a part of Flash that will remain untouched until the final parts of the Flash update.

Further, because the I2C driver itself might be updated there needs to be a means of over-writing that driver with code running from some location other than Flash. In this application that code (the copy_ram_to_flash() function) will run from RAM.

And one last requirement: To make the Flash writes occur as fast as possible they must occur as 128 byte block writes. This is accomplished in the Flash update code by copying four 128 byte blocks from I2C EEPROM to RAM, then calling the copy_ram_to_flash() function to copy those blocks from RAM to Flash.

To keep the I2C driver segment separate from the rest of the Flash code it is located in upper Flash as follows:

In the I2C.c file a #pragma flash_update code segment is defined containing the I2C driver and functions which call the copy_ram_to_flash() code.

In the linker .lkf file the following directive is added to specify the location of the pragma section:

```
+seg .flash_update -b 0xfc80
```

The copy_ram_to_flash() code also requires special handling as the STM8 hardware will only allow copies from RAM to Flash if the code performing the copy is running in RAM. Since RAM is very limited the copy_ram_to_flash code is designed to have a minimal code space footprint. The steps to implement this part of the code are as follows:

In the I2C.c file a #pragma memcpy_update code segment is defined containing the copy_ram_to_flash code.

In the linker .lkf file the following directives are added to specify the location of the pragma section:

```
+seg .memcpy_update -a .data -n memcpy_update –ic
+seg .bss -a memcpy_update -n .bss
```

In the stm8s-005.h file find the following:

```
/* Uncomment the line below to enable the FLASH functions execution
from RAM */
#if !defined (RAM_EXECUTION)
/* #define RAM_EXECUTION  (1) */
#endif /* RAM_EXECUTION */
```

… And uncomment the line

```
/* #define RAM_EXECUTION  (1) */
```

In the main.c file the following code causes the copy_ram_to_flash() function to be relocated to RAM. Once relocated, the function can be called like any other function.

```
_fctcpy ('m');
```

Given the above: To complete a copy of new firmware from I2C EEPROM to Flash the process is as follows:

a) The _fctcpy ('m') function is called from main.c
b) The eeprom_copy_to_flash() function is called from main.c. This function will copy all of the I2C EEPROM content to Flash except for the segment containing the I2C driver and copy_ram_to_flash() code. As mentioned this is done by copying 512 bytes to RAM then copying the 512 bytes to Flash and repeating as needed.
c) The last step in the eeprom_copy_to_flash() code is to copy the I2C Driver and copy_ram_to_flash() function to RAM. These functions fit within 512 bytes. Next the code initiates a copy of those functions from RAM to Flash. Remember, at this point the actual copy function is being run from the existing copy_ram_to_flash() function in RAM, so it doesn't matter that the functions are being over-written in Flash. Once the copy completes, the code waits for a device reset using the Window watchdog function. At boot the new firmware is run from Flash.

_fctcpy

For reference the following is from the CXSTM8_UserGuide.pdf:

_fctcpy

Description

Copy a moveable code segment in RAM

Syntax

```
int _fctcpy(char name);
```

Function

_fctcpy copies a moveable code segment in RAM from its storage location in ROM. *_fctcpy* scans the descriptor built by the linker and looks for a moveable segment whose flag byte matches the given argument. If such a segment is found, it is entirely copied in RAM. Any function defined in that segment may then be called directly. For more information, see "*Moveable Code*" in **Chapter 6**.

Return Value

_fctcpy returns a non zero value if a segment has been found and copied. It returns 0 otherwise.

Example

```
if (_fctcpy('b'))
      flash();
```

Notes

_fctcpy is packaged in the machine library.

#pragma Sections

For reference the following describes how to create #pragma sections.

https://cosmic-software.com/faq/faq18.php

Can I create my own segment names other than the reserved segment names used by the linker?

The compiler allows you to redefine a reserved section name for a block of code or data using #pragmas. Redefining of section names is a popular technique that gives the user more control in locating code and data images at specific memory addresses. It is possible to redirect any memory type to a user defined section by using the following pragma definition: See the section "Redefining Sections" in the user's manual for more information.

```
#pragma section <attribute> <qualified_name>
```

where <attribute> is either empty or one of the following sequences:

```
const, @dir, @eeprom, @far
```

and <qualified_name> is a section name enclosed as follows:

```
(name) - parenthesis indicating a code section
[name] - square brackets indicating uninitialized data
{name} - curly braces indicating initialized data
```

The following example redefines the .text section to .mycode where the parenthesis indicate that it's a code section. This causes all code under the first #pragma to be allocated to the .mycode section instead of the .text section. The second #pragma ends the redefinition so any subsequent code would be allocated to the .text section.

```
#pragma section (mycode)

    Any code or data

#pragma section ()
```

Example Linker segment using the .code section

```
+seg .mycode -b0xC000
```

Content of the .lkf File

For reference the +seg part of the .lkf file for this project looks like this:

```
+seg .vector -b 0x8000 -m 0x8000 -n .vector          # vectors start address
-k
+seg .const -a .vector -n .const                     # constants follow vectors
+seg .text -a .const -n .text                        # code follow constants
+seg .eeprom -b 0x4000 -m 128                        # internal eeprom
+seg .bsct -b 0 -m 0x100 -n .bsct                    # initialized RAM (256 bytes
                                                     #   max)  in page 0
+seg .ubsct -a .bsct -n .ubsct                       # uninitialized RAM follows
                                                     #   initialized RAM in page 0
+seg .bit -a .ubsct -n .bit –id                      # uninitialized bit variables
                                                     #   follows uninitialized RAM
                                                     #   in page 0
+seg .data -a .bit -m 0x800 -n .data                 # initialized RAM outside of
                                                     #   page 0 (2048 bytes max for
                                                     #   all RAM in .bsct, .ubsct, .bit,
                                                     #   and .data)
+seg .memcpy_update -a .data -n memcpy_update –ic   # specifies that the memcpy_
                                                     #   update segment in Flash is
                                                     #   relocatable and that the
                                                     #   target location is in RAM
                                                     #   following .data.
+seg .bss -a memcpy_update -n .bss                   # .bss is the uninitialized data
                                                     # segment. This specifies that
                                                     # the data is to be placed after
                                                     # the memcpy_update segment,
                                                     # effectively placing memcpy_
                                                     # update first in RAM.
+seg .flash_update -b 0xfc80                         # locates the flash_update
                                                     # segment at 0xfc80
+seg .iconst -b 0x5fe -n .iconst                     # locates the stack_overflow
                                                     # variables at 0x05fe
```

Paraphrased from the CXSTM8_Userguide.pdf

Segment Control Options Usage

-a* make the current segment follow the segment *, where * refers to a segment name given explicitly by a **-n** option. Options **-b** and **-e** cannot be specified if **-a** has been specified. Option **-o** can be specified only with value **0** to reset the logical address to the same value than the physical address.

-b* set the physical start address of the segment to *. Option **–e** or **-a** cannot be specified if **-b** has been specified.

-id initialize this segment
-ic mark this segment as moveable segment

-m* set the maximum size of the segment to * bytes. If not specified, there is no checking on any segment size. If a segment is declared with the **-a** option as following a segment which is marked with the **-m** option, then set the maximum available space for all the possible consecutive segments. If a **-m** is specified on a **-a** segment, the actual maximum size checked is equal to the given value minus the size of all the segments already allocated from the first segment of the **–a** list. So the new maximum size is computed from the start address of the list and not from the start address of that segment.

-n* set the output name of the segment to *. Segment output names have at most **15** characters; longer names are truncated. If no name is given with a **-n** option, the segment inherits a default name equal to its assembler section name.

Developers: STM8 Address Map

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

Following is the mapping of memory, peripherals, and registers in the STM8.

RAM	0x0000 to 0x05ff	1536 bytes
Stack	0x0600 to 0x07ff	512 bytes
EEPROM	0x4000 to 0x407f	128 bytes
Option ROM	0x4800 to 0x487f	128 bytes
Peripheral	0x5000 to 0x57ff	2048 bytes
Boot Rom	0x6000 to 0x67ff	2048 bytes
Registers	0x7f00 to 0x7fff	256 bytes
Int Vectors	0x8000 to 0x807f	128 bytes
Flash Prog	0x8080 to 0xffff	32640 bytes

Developers: Flash Memory Map

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

STM8 Flash Map for Non-Upgradeable builds

Block Number (128 Bytes per Block)	Block Start Address	Content
000	0x8000	Reset and Interrupt Vectors
001	0x8080	Constants and Code
…		
252	0xfe00	Constants and Code
253	0xfe80	Browser Only IO Timers and IO Names Note: The first 64 bytes are not used so these variables really start at 0xfec0
254	0xff00	Browser Only IO Timers and IO Names
255	0xff80	Browser Only IO Timers and IO Names

STM8 Flash Map for Upgradeable builds

Block Number (128 Bytes per Block)	Block Start Address	Content	Equivalent Address in I2C EEPROM Region 0
000	0x8000	Reset and Interrupt Vectors	0x0000
001	0x8080	Constants and Code	0x0080
…			
248	0xfc00	Constants and Code	0x7c00
249	0xfc80	I2C Driver and copy_ram_to_flash() function	0x7c80
250	0xfd00	I2C Driver and copy_ram_to_flash() function	0x7d00
251	0xfd80	I2C Driver and copy_ram_to_flash() function	0x7d80
252	0xfe00	I2C Driver and copy_ram_to_flash() function	0x7e00
253	0xfe80	Browser Only IO Timers and IO Names Note: The first 64 bytes are not used so these variables really start at 0xfec0	Browser Only IO Timers and IO Names Note: The first 64 bytes are not used so these variables really start at 0xfec0
254	0xff00	Browser Only IO Timers and IO Names	Browser Only IO Timers and IO Names
255	0xff80	Browser Only IO Timers and IO Names	Browser Only IO Timers and IO Names

Developers: I2C EEPROM Memory Map

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

I2C EEPROM Regions for Upgradeable Builds

The I2C EEPROM used with Upgradeable Builds is divided into four 32KB regions as follows:

Region 0: Multiple use

a) Stores Runtime Firmware or Strings as they are uploaded from the Browser
b) Stores a copy of the Runtime firmware. At boot time the Runtime Firmware will compare itself to Region 0 and will copy itself to Region 0 if there is a difference. This provides a backup copy of the Runtime Firmware for use in case the Uploader is started and then canceled, and covers the case where the STLink-V2/SWIM interface was used to flash the STM8 with Upgradeable Runtime firmware.

Region 1: Code Uploader

Used to store the Code Uploader firmware.

Region 2: Strings

Used to store the “Strings”, ie, the IO Control and Configuration webpage content.

Region 3: Reserved for future use.

Addressing the I2C EEPROM Regions

This gets a little confusing so some explanation is required.

The 24xx1025 is addressed as if it were a pair of 64KB devices. So even though it is used as four 32KB regions, the addressing of the device requires some translation.

When Region 0 or Region 1 are addressed the “I2C Command Byte” used on the I2C bus to perform a Read is 0xa1, and to perform a Write is 0xa0. Then Region 0 is addressed as bytes 0x0000 to 0x7fff, and Region 1 is addressed as bytes 0x8000 to 0xffff.

When Region 2 or Region 3 are addressed the “I2C Command Byte” used on the I2C bus to perform a Read is 0xa9, and to perform a Write is 0xa8. Then Region 2 is addressed as bytes 0x0000 to 0x7fff, and Region 3 is addressed as bytes 0x8000 to 0xffff.

The following defines are used to help translate this into something more “readable” in code:

```
#define I2C_EEPROM0_READ        0xa1   // 1010 0001
#define I2C_EEPROM0_WRITE       0xa0   // 1010 0000
#define I2C_EEPROM1_READ        0xa1   // 1010 0001
#define I2C_EEPROM1_WRITE       0xa0   // 1010 0000
#define I2C_EEPROM2_READ        0xa9   // 1010 1001
#define I2C_EEPROM2_WRITE       0xa8   // 1010 1000
#define I2C_EEPROM3_READ        0xa9   // 1010 1001
#define I2C_EEPROM3_WRITE       0xa8   // 1010 1000

#define I2C_EEPROM0_BASE        0x0000 // Base address of EEPROM0 region
#define I2C_EEPROM1_BASE        0x8000 // Base address of EEPROM1 region
#define I2C_EEPROM2_BASE        0x0000 // Base address of EEPROM2 region
#define I2C_EEPROM3_BASE        0x8000 // Base address of EEPROM3 region
```

Developers: Strings File Generation

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

THIS INFORMATION IS ONLY PERTINENT TO THE UPGRADABLE BUILDS.

When using Upgradeable Builds a “Strings File” must be generated and stored in the I2C EEPROM. The “Strings File” contains the HTML and Javascript that defines the IO Control and Configuration web pages. That HTML and Javascript resides in the httpd.c file, but it must be extracted and placed in a separate .sx file for upload to the Network Module.

I’ve written a program that runs in the Windows environment that parses the httpd.c file, locates the webpage HTML of interest, and generates the required .sx file. Along with the HTML text the .sx file also includes pointers to the start of the HTML in the .sx file and the size of the HTML text. The resulting .sx file can be directly uploaded to the Network Module using the Code Uploader.

I NEED TO MAKE THIS PROGRAM AVAILABLE TO OTHER DEVELOPERS. IN THE MEANTIME I WILL GENERATE THE STRINGS FILE AND RELEASE IT WITH THE NETWORK MODULE CODE.

Developers: Using the UART

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

The STM8S processor includes a UART. The code can be compiled with the UART enabled so that you can add debug statements to the code. The compile option is available in the uipopt.h file (see #define DEBUG_SUPPORT).

When the UART is enabled it will take over IO Pin 11 and use that as the UART transmit pin. No UART receive pin is implemented. The Configuration page will show the pin 11 as an Output, and you will not be able to change that pin configuration when the UART option is compiled. Likewise, the IOControl page will show the pin as an Output, and any attempt to change the pin state will not be successful. But the pin will continue to function as the UART output until the code is recompiled with the Debug UART option turned off.

The UART transmitter is set up as follows:

Baud rate: 115200
Data bits: 8
Parity: None
Stop bits: 1

To connect the Network Module UART Tx pin (IO 11) to a terminal on your PC you'll need a TTL-RS232 or TTL-USB converter. There are several ways to do this, but I will describe the method I used:

a) Use a TTL-USB converter to connect the Tx pin to the USB port on a laptop. Here's the device I used:

FT232RL 3.3V 5.5V FTDI USB to TTL Serial Adapter F
Arduino Mini Port N137

Condition: New

Bulk savings: Buy 1 $3.99/ea | Buy 2 $3.79/ea | Buy 3 $3.67/ea

Quantity: 1 4 or more for $3.59/ea

More than 10 available
164 sold / See feedback

Price: US $3.99/ea

Buy another

Add to cart

Add to Watchlist

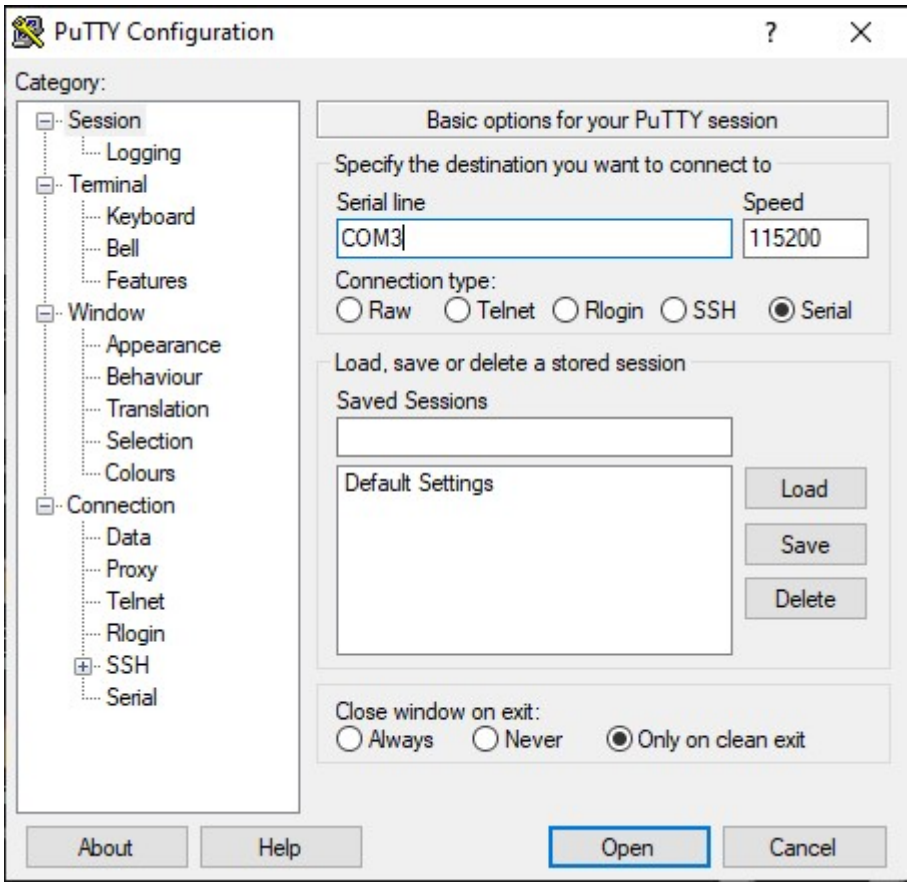
Keep in mind that the Network Module operates at 3V, so the adapter needs to be set to 3V IO with the jumper located near the connection pins.

Pin 11 on the Network Module is the “TX” output from the Network Module. It must be connected to the “RX” pin on the Serial Adapter shown above. The only other connections to the Network Module are +3.3V and GND.

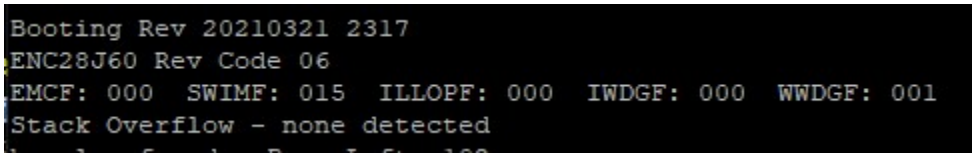
DTR, CTS and TX on the adapter are not used.

b) My laptop was a Windows 10 OS, so I had to download the drivers for the TTL-USB interface here:
https://ftdichip.com/drivers/

c) I used PuTTY on my laptop to communicate with the TTL-USB device. In my case the device showed up as COM3. PuTTY setup is simple:



d) If everything is set up correctly you should see something like this in the PuTTY display when the Network Module boots:



Note: Use the http command “http://IP:Port/70” to clear the “Reset Status Register" counters (the counts for EMCF, SWIMF, ILLOPF, IWDGF, WWDGF). See the STM8S documentation for definitions of the Reset Status Register content.

Developers: Analysis of MQTT sendbuf sizing

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

Analysis of the size required for the mqtt_sendbuf:

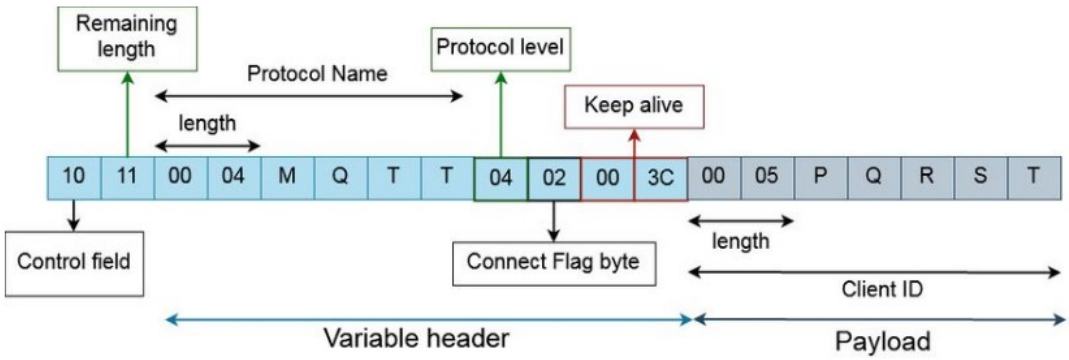
During MQTT startup we do not queue any MQTT messages. They are sent one at a time, and in the case of CONNECT or SUBSCRIBE messages the code will wait for an ACK from the broker.

After startup there can be queuing of messages, but that queue will only grow to a maximum of 2 messages. This occurs when a PINGREQ is issued, and that request immediately precedes or immediately follows some other PUBLISH message. The reason the queue is needed is because we need to watch for a PINGRESP (an ACK) to the PINGREQ. Thus, the PINGREQ is retained in the queue until the PINGRESP is received. PUBLISH does not wait for an ACK since we are operating as QOS0.

Why is all the above mentioned? Because we want the mqtt_sendbuf to be as small as possible. The way the MQTT code works is that it places a message in the mqtt_sendbuf, and that message is always accompanied by a queue management structure of about 11 bytes. So to determine the mimimum size required for mqtt_sendbuf we need to know that maximum size MQTT message that will go in the buffer.

It turns out the largest message is the CONNECT message, which is sent during MQTT startup. At that time no PINGREQ messages can be issued, so the CONNECT message will occupy the mqtt_sendbuf by itself. An analysis of the size of the CONNECT message:

CONNECT message



The above is 12 bytes (excluding the payload) plus the following:

Payload:
The payload of the CONNECT Packet contains one or more length-prefixed fields, whose presence is determined by the flags in the variable header. These fields, if present, MUST appear in the order Client Identifier, Will Topic, Will Message, User Name, Password [MQTT-3.1.3-1].

Client Identifier:
NetworkModuleaabbccddeeff
2 length bytes + 25 bytes data = 27

Will Topic:
NetworkModule/NetworkModule456789/availability
2 length bytes + 46 bytes data = 48

Will Message:
Offline
2 length bytes + 7 bytes data

UserName:
2 length bytes + 10 bytes data

Password:
2 length bytes + 10 bytes data

Queue management structure 11 bytes

Max CONNECT message when placed in the mqtt_sendbuf is
120 + 11 = 131 bytes

Largest messages AFTER MQTT startup:

The only messages that must be retained in the mqtt_sendbuf are the CONNECT, SUBSCRIBE, and PINGREQ messages. Since the CONNECT and SUBSCRIBE messages are only sent by the mqtt_startup state machine we can carefully manage that state machine to make sure the ACK for each of those messages is received before continuing the state machine. This means that the only message that will be retained in the mqtt_sendbuf during normal operation is the PINGREQ message. That message is two bytes long. So, we can minimize the size of the mqtt_sendbuf to the point that only that 2 byte message and the longest of any other MQTT transmit message will fit.

Output PUBLISH Msg (in the mqtt_sendbuf)
Fixed Header 2 bytes
Variable Header 47 bytes
Topic NetworkModule/DeviceName012345678/output/xx 43 bytes
Topic Length 2 bytes
Packet ID 2 bytes
Total: 49 bytes

Temperature PUBLISH Msg (in the mqtt_sendbuf)
Fixed Header 2 bytes
Variable Header 51 bytes
Topic NetworkModule/DeviceName012345678/temp/xxxxxxxxxxxx-000.0 49 bytes
Topic Length 2 bytes
Packet ID 2 bytes
Total: 63 bytes

Status PUBLISH Msg (in the mqtt_sendbuf)
Fixed Header 2 bytes
Variable Header 57 bytes
Topic NetworkModule/DeviceName123456789/availabilityoffline 53 bytes
Topic Length 2 bytes
Packet ID 2 bytes
Total: 59 bytes

The implication of the above is that the mqtt_sendbuf needs to be as large as one PINGREQ message (2 bytes) plus the longest Publish message (59 bytes), or 61 bytes. I

think the structure added to the mqtt_sendbuf for each message in the queue is 11 bytes (search on struct mqtt_queued_message).
This would make the requirement for the mqtt_sendbuf AFTER MQTT STARTUP = 61 + 22 = 83.

Since we know the CONNECT message far exceeds this value it will set the size of the mqtt_sendbuf.

Developers: Storing Variables in Flash via the Application

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

Flash Programming

Beginning in April 2021 the ability to have user entered names for each IO pin was implemented. This required storing those names in Flash memory (not EEPROM). Following are notes on how this is done.

The following excerpts from the RM0016 and PM0051 clearly indicate that performing byte and word writes to Flash will stop application execution until the write completes. If word or byte writes are implemented it is not necessary to have code execution in RAM. However, **block** writes to Flash do require code execution in RAM.

It appears that when byte writes are performed they may cause as many as 4 erase/write cycles on all 4 bytes around a memory location. This is because Flash is actually written 4 bytes at a time even if a single byte is being written. So if you write 4 bytes in a row, you actually write all 4 bytes 4 times.

For this reason the Word method (4-bytes at a time) will cause less wear on memory. If 4 bytes are going to be written sequentially anyway then writing them as a 4-byte Word will result in one erase/write cycle on that Word instead of 4 cycles. PM0051 section 4.3 describes how to do this. A logical presumption (because I can't find it in manuals): The first byte of a Word must be at an address ending in 0, 4, 8, or C.

Implementing FLASH writes from code
If a **byte** needs to be written do a read and compare FIRST to make sure a write is necessary, then

```
unlock_flash();
memcpy(&flash, &value, 1);
lock_flash();
```

Since writes always occur 4 bytes at a time, code should be written to write 4 byte values as much as possible. So, consideration needs to be taken to accumulate writes in RAM first, then write them as 4 byte values to Flash. The 4 byte values are on 4 byte boundaries starting at address 0x0000. Example of a 4 byte write:

```
unlock_flash();
for (i=0; i<8; i+=2) {
  // IO_TIMER values are 2 byte values. They are accumulated in RAM first, then
```

```
   // the sixteen 2 byte values are written to Flash as eight 4 byte values. First
   // the values in RAM are compared to what is already in Flash to make sure no
   // unnecessary writes are performed.
   // Compare 4 bytes at a time. If any miscompare write to Flash 4 bytes at a time.
   if (IO_TIMER[i] != Pending_IO_TIMER[i] || IO_TIMER[i+1] !=
Pending_IO_TIMER[i+1]) {
     FLASH_CR2 = 0x40;
     FLASH_NCR2 = 0xBF;
     memcpy(&IO_TIMER[i], &Pending_IO_TIMER[i], 4);
   }
  }
  lock_flash();
```

In this application (Browser Only build):

Flash Program memory is from 0x8080 to 0xffff
The upper 256 bytes could are reserved for IO Names
The next upper 32 bytes are reserved for IO Timers
An additional 32 bytes are reserved for future use
So, a total of 320 bytes are reserved in Flash for the user values.

There is no protection against code being written to the "reserved" area, so care needs to be taken that code size doesn't grow too large and over-lap the “reserved” area.

The specification states that there is 32K of Flash (32768 bytes). But the first 128 bytes (starting at 0x8000) is used for interrupt vectors. So if we use 320 bytes for user data the maximum code size is 32768 - 128 - 320 = 32320.

The real problem with using Flash for user data is that it has very limited write life. So frequent changes should be avoided. Caution the user to set IO names and IO timers only a few times during the life of the device. NEVER use an automated mechanism that might inadvertantly change the names or timer values a lot of times.

The 16 IO Names are limited to 16 characters each (including the terminator).
The 16 IO Timers are limited to 2 bytes each.

Note that single byte writes should be avoided. A single byte write actually results in a 4 byte write of the targeted byte plus the 3 other bytes in the 4 byte "word". So if you want to write all 4 bytes in a word and you do that one byte at a time, you will actually write every byte of the word 4 times. This creates a lot of wear on the Flash, so in this application data that will be written to the Flash is first accumulated in RAM, then written to Flash as a series of 4 byte "word" writes.

Note that program execution is suspended by hardware for the 6ms that a Flash write is being performed. From the PM0047 manual:

> "Byte programming is done by executing any write instruction (ld, mov...) to a Flash memory address when the memory is unlocked. The write instruction

initiates the erase/programming cycle and any core access to the memory is blocked until the cycle is over. This means that program execution from Flash is stopped until the end of the erase/programming cycle. At the end of the programming the EOP bit in the FLASH_IAPSR is set and program execution restarts from the instruction following the write/erase instruction."

A write to Flash requires about 6ms regardless of whether it is a 1 byte or 4 byte write. Just for reference if the entire 320 byte reserved area of Flash were written at one time it would take about (320 / 4) * 6ms = 480ms. Since this only occurs when the user makes changes it is not significant to operation of the device.

From CXSTM8_UsersGuide:

Referencing Absolute Addresses

References to absolute addresses have the general form @<address>, where <address> is a valid memory location in your environment. For example, to associate an I/O port at address 0x20 with the identifier name MISCR1, write a definition of the form:

```
char MISCR1 @0x20;
```

where @0x20 indicates an absolute address specification and not a data initializer. Since input/output on the STM8 architecture is memory mapped, performing I/O in this way is equivalent to writing in any given location in memory. Such a declaration does not reserve any space in memory. The compiler still creates a label, using an equate definition, in order to reference the C object symbolically. This symbol is made public to allow external usage from any other file.

Flash Unlock/Lock

Unlock: Write 56h then AEh in FLASH_PUKR (00 5062h)
Lock: Reset bit 1 (PUL) in FLASH_IAPSR (00 505Fh)

Flashing Browser Only Firmware

When programming a new code load program address range 0x8000 to 0xfebf. This will retain prior user entered IO Names and Timers. Otherwise there will be a program compare error and the user entered IO Names and Timers will be over-written with zero.

RM0016

4.6.2 Byte programming

The main program memory and the DATA area can be programmed at byte level. To program one byte, the application writes directly to the target address.

- In the main program memory:

  The application stops for the duration of the byte program operation.

RM0016

4.6.3 Word programming

A word write operation allows an entire 4-byte word to be programmed in one shot, thus minimizing the programming time.

As for byte programming, word operation is available both for the main program memory and data EEPROM. On some devices, the read-while-write (RWW) capability is also available when a word programming operation is performed on the data EEPROM. Refer to the datasheets for additional information.

- In the main program memory:

  The application stops for the duration of the byte program operation.

PM0051

4.4 Byte programming

Both main program memory and data EEPROM can be programmed and erased at byte level.

Byte programming is performed by executing a write instruction (ld, mov...) to an address in main program memory when the memory is unlocked. The write instruction initiates the erase/program cycle and any core access to the memory is blocked until the cycle has completed. This means that program execution from the Flash program memory is stopped until the end of the erase/program cycle.

When a new byte program operation starts, EOP and WR_PG_DIS bits of FLASH_IAPSR register are automatically cleared. At the end of the program operation, the EOP bit in the FLASH_IAPSR register is set and the program execution restarts from the instruction following the write/erase instruction.

PM0051

4.3 Word programming

Both main program memory and data EEPROM can be programmed and erased at word level. Word operations are performed in the same way as block operations. They can be executed either from program memory or from RAM.

When a new word program operation starts, EOP and WR_PG_DIS bits of FLASH_IAPSR register are automatically cleared.

Contrary to word programming of the Flash program memory, the word programming of data EEPROM with RWW feature (when available) does not stop program execution. The EOP bit can then be used to know if the previous operation has completed. This bit is automatically reset when reading FLASH_IAPSR.

The following sequence is required to perform a word program operation:

1. Unlock the memory if not already done.

   The UBC option byte can be read to check if the word you want to program is not in the UBC area. If necessary, reprogram it to allow programming the targeted word.
2. Write 0x40 in FLASH_CR2 (WP bit active), and 0xBF in FLASH_NCR2 (NWP bit active).
3. Write the 4 data bytes to the memory starting with the very first address of the word to be programmed.

   The programming cycle starts automatically when the 4 bytes have been written.
4. Check the WR_PG_DIS bit in FLASH_IAPSR to verify if the word you attempted to program was not write-protected (optional).
5. To check if the program operation is complete, poll the EOP bit in FLASH_IAPSR register for the end of operations. EOP is set to '1' when the word program operation has completed. To avoid polling the EOP bit, an interrupt can be generated when EOP is set.

Caution: FLASH_CR2 and FLASH_NCR2 must be written consecutively to be taken into account. If only one register is set, both are forced to their reset values, causing the program operation to be performed at byte level.

Caution: EOP and WR_PG_DIS bits are cleared by reading the FLASH_IAPSR register. It is consequently strongly recommended to perform one single read operation to the FLASH_IAPSR register to check the values of these bits.

Developers: Flash and EEPROM Wear Notes

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

Notes for Developers regarding Flash and EEPROM wear:

- Make sure anything stored in Flash is very infrequently written. The Flash spec is 100 cycles, whereas the EEPROM spec is 100,000 cycles.
- Having said that, from a practical perspective the Flash and EEPROM write/erase cycles are far greater than those numbers.
    - The spec assumes write/erase at an ambient temperature of 85C, followed by 20 years retention at 55C.
    - I think it is safe to say that most applications will not experience anything close to those temperatures for extended time frames.
- As a point of reference: During development and test of this application I've written Flash at least 1000 times on a single device and have experienced no trouble. Does this mean my test devices will retain the program for 20 years? Probably not, but since my temperatures are much lower than the spec the degradation is likely not severe. Still, this anecdotal evidence suggests that regular users should have no concerns about Flash degradation as long as they don't implement some automated method that causes a very high number of Flash writes. For example, don't implement automation to write the user entered IO Name and IO Timer values. Those values are stored in Flash and should only be entered manually by a user.
- With regard to EEPROM I know that as a result of a few development tests some EEPROM locations were unintentionally written well in excess of 300,000 cycles. Again, those devices are still operating normally. But I have marked them as potentially unreliable for long term data retention. Again it suggests that regular users likely have no concerns about EEPROM wear out as long as no automated methods are used that would cause a very high number of EEPROM writes. For example, users should avoid the use of the "Retain" setting for relay states if automation is changing relay states more than a few times a day.

Here is the pertinent device specification for the STM8S005C6 from data sheet DS8638 Rev 5 dated September 2018.

Table 35. Flash program memory and data EEPROM

Symbol	Parameter	Conditions	Min(1)	Typ	Max	Unit
V_{DD}	Operating voltage (all modes, execution/write/erase)	$f_{CPU} \leq 16$ MHz	2.95	-	5.5	V
t_{prog}	Standard programming time (including erase) for byte/word/block (1 byte/4 bytes/128 bytes)	-	-	6.0	6.6	ms
	Fast programming time for 1 block (128 bytes)	-	-	3.0	3.3	ms
t_{erase}	Erase time for 1 block (128 bytes)	-	-	3.0	3.3	ms
N_{RW}	Erase/write cycles(2) (program memory)	$T_A = 85$ °C	100	-	-	cycles
	Erase/write cycles(2) (data memory)		100 k	-	-	
t_{RET}	Data retention (program memory) after 100 erase/write cycles at $T_A = 85$ °C	$T_{RET} = 55$° C	20	-	-	years
	Data retention (data memory) after 10 k erase/write cycles at $T_A = 85$ °C		20	-	-	
	Data retention (data memory) after 100 k erase/write cycles at $T_A = 85$ °C	$T_{RET} = 85$° C	1.0	-	-	
I_{DD}	Supply current (Flash programming or erasing for 1 to 128 bytes)	-	-	2.0	-	mA

1. Guaranteed by characterization results.
2. The physical granularity of the memory is 4 bytes, so cycling is performed on 4 bytes even when a write/erase operation addresses a single byte.

Developers: How Connections Open and Close And Relationship to the "current_webpage" Variable

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

Since this issue has flummoxed me several times causing many hours of debug here is a brief writeup on how connections open and close, and how this relates to the variable "current_webpage" in the code.

Basics:
When a webpage is sent from the webserver to a browser a connection is opened, the page sent (even if several packets are required), then the connection is closed. The variable "current_webpage" is used only while that connection is open so that the firmware remembers which page is being sent in that connection.

When a client sends a POST or GET request a connection is opened, the POST or GET is received and processed in the webserver (even if several packets are required), then the connection is closed.

- When a POST is received in this implementation the connection is closed after sending a 204 header. That might be an incorrect method – an investigation for another time. But, the appropriate page repaint occurs, so perhaps I have it right.
- When a GET is received the connection is closed after sending a 200 header with the requrested information (usually the IOControl or Configuration page but it could be other actions). In some cases a 204 header is sent with no data when appropriate.

Important to note is that the variable "current_webpage" is no longer available once a page is transmitted. So, I no longer know what page is displayed on a given Client when it comes time to process the POST from that Client. So …

More detail:
When a POST is received the webserver needs to determine if that POST is coming from a IOControl page or from a Configuration page. There might be an easier way to do this, but I use the content of the POST to make this determination as follows:

If the POST is coming from a Configuration page it will look similar to this:

a00=NewDevice000&b00=c0a801c7&b04=c0a80101&b08=ffffff00&c00=0050&d00=c24d696b0199&j00=IO01&i00=0000&j01=IO02&i01=0000&j02=IO03&i02=0000&j03=IO04&j04=IO05&j05=IO06&j06=IO07&j07=IO08&j08=IO09&j09=IO10&j10=IO11&i10=0000&j11=IO12&j12=I

O13&j13=IO14&j14=IO15&j15=IO16&h00=030303010000000000000300000000000&i03=0000&i04=0000&i05=0000&i06=0000&i07=0000&i08=0000&i09=0000&i11=0000&i12=0000&i13=0000&i14=0000&i15=0000&g00=00&z00=0

If the POST is coming from a IOControl page it will look similar to this:

h00=8080000000000000000000000000000&z00=0

The thing to note is that while both POSTs contain 'h00' data, the Configuration POST always begins with 'a00' data. Thus when parsing it is easy to determine the source of the data.

Why do we even need to know the source of the POST? It comes down to properly parsing the 'h00' data. That data contains the configuration and state for the IO pins. The 'h00' data that comes from the Configuration page contains information like Input/Output/Disabled and Invert for the pins. The 'h00' data that comes from the IOControl page contains the ON/OFF state of the pin. The code needs to make sure it properly masks the bits in the returned bytes so that the Configuration page data does not change the ON/OFF bit, and so that the IOControl data does not change the configuration bits.

The code examines the incoming POST data and if it starts with the 'a00' data we know it is coming from a Configuration page (and current_webpage is set appropriately), otherwise it is from a IOControl page. Problem solved.

nParseLeft and z00=0:

When I originally pulled together the various public code and added my own code I knew that parsing and validating a POST should be based on the length of the POST. To that end the variable "pSocket->nParseLeft" is used to track the data being parsed. As code development progressed I kept running into problems with that method, so I added a dummy POST value "z00=0" (you can see it in the POST examples above). The dummy value was a secondary method for me to determine that an entire POST was received. I'm going to leave that in the code for now as it works just fine, but the redundancy is probably not necessary.

Developers: How Proto-Sockets are Implemented and Connections Established

NOTE: This information is only needed if you plan to set up your own development environment to modify and compile code.

Since this code is a collection of efforts from several code sources there are many cases where layers of obfuscation occur. An important issue is “How are proto-sockets implemented?” … and unfortunately this is hard to trace through the code. Some notes since I recently dug into this. I admit ahead of time I may describe some of this with inaccurate coding terminology as structs and unions always seem to make my head hurt.

In the uip.c and uip.h files you’ll see that data associated with connections is maintained in an array called “uip_conns[]”. The uip_conns[] array is actually an array of structs as defined in uip.h:

```
/* The array containing all uIP connections. */
extern struct uip_conn uip_conns[UIP_CONNS];
```

and

```
/**
 * Representation of a uIP TCP connection.
 * The uip_conn structure is used for identifying a connection. All but one
 * field in the structure are to be considered read-only by an application.
 * The only exception is the appstate field whos purpose is to let the
 * application store application-specific state (e.g., file pointers) for the
 * connection. The type of this field is configured in the "uipopt.h" header
 * file.
 */
struct uip_conn {
  uip_ipaddr_t ripaddr;  // The IP address of the remote host.
  uint16_t lport;        // The local TCP port, in network byte order.
  uint16_t rport;        // The local remote TCP port, in network byte order.
  uint8_t rcv_nxt[4];    // The sequence number that we expect to receive next.
  uint8_t snd_nxt[4];    // The sequence number that was last sent by us.
  uint16_t len;          // Length of the data that was previously sent.
  uint16_t mss;          // Current maximum segment size for the connection.
  uint16_t initialmss;   // Initial maximum segment size for the connection.
  uint8_t sa;            // Retransmission time-out calculation state variable.
  uint8_t sv;            // Retransmission time-out calculation state variable.
  uint8_t rto;           // Retransmission time-out.
  uint8_t tcpstateflags; // TCP state and flags.
  uint8_t timer;         // The retransmission timer.
  uint8_t nrtx;          // The number of retransmissions for the last segment sent.
```

```
  /** The application state. */
  uip_tcp_appstate_t appstate;
};
```

Note the last part “uip_tcp_appstate_t appstate;”.
The above contains the basic information for a given connection. But we need to associate that connection with more information that is specific to our webserver operation. That’s where “uip_tcp_appstate_t appstate;” comes in. Over in the uip_tcpapphub.h file you’ll see this union:

```
typedef union
{
  struct tHttpD HttpDSocket;
} uip_tcp_appstate_t;
```

In httpd.h you’ll see this struct definition:

```
struct tHttpD
{
  uint8_t nState;
  const uint8_t* pData;
  uint16_t nDataLeft;
  uint8_t nNewlines;
  uint16_t nParseLeft;
  uint8_t ParseCmd;
  uint8_t ParseNum;
  uint8_t ParseState;
  uint16_t nPrevBytes;
  uint8_t current_webpage;
  uint8_t insertion_index;
  int structID;
};
```

And in uip_tcpapphub.c you’ll see this call

```
HttpDCall(uip_appdata, uip_datalen(), &uip_conn->appstate.HttpDSocket);
```

What’s it all mean?

“HttpDCall(uip_appdata, uip_datalen(), &uip_conn->appstate.HttpDSocket);” calls the HttpDCall() function and passes the pointer “&uip_conn->appstate.HttpDSocket” to it. That pointer points to the structure union that contains all the information associated with a given connection (both the uip_conn and tHttpD structures). That information defines the “proto-socket” or connection data for this connection. The HttpDCall() function is the code that receives requests from Clients and transmits pages to Clients. Remember: This “proto-socket” only exists while a connection is open. When the connection is closed the proto-socket structure is made available for use by a new connection.

Now for some more obfuscation: How are connections established? I've had to track the following down so many times I thought I would document it for my own sanity if nothing else.

In main.c you'll see in the main loop this code:

```
  if (uip_len > 0) {
    if (((struct uip_eth_hdr *) & uip_buf[0])->type == htons(UIP_ETHTYPE_IP)) {
      uip_input();
      …
```

In the above "uip_input()" is what starts the connection process. But it doesn't do it directly. In uip.h you'll see this define:

```
#define uip_input()        uip_process(UIP_DATA)
```

In uip.c you'll see this code:

```
void uip_process(uint8_t flag)
{
 register struct uip_conn *uip_connr = uip_conn;
 if (flag == UIP_POLL_REQUEST) {
 …
 }
 else if (flag == UIP_TIMER) {
 …
 }
 // This is where the input processing starts. We fall through to this point
 // if the call was uip_process(UIP_DATA)
 …
```

At this point the uip.c code will do some incoming packet processing and validation (like checking the header) then will call the application (the webserver) to process the incoming data. Further down in uip.c you'll find this statement:

```
UIP_APPCALL();
```

The above call appears in several places depending on what the uip.c code has determined is appropriate. But how does UIP_APPCALL() actually invoke the webserver process?

In uip_tcpapphub.h you'll find:

```
#define UIP_APPCALL    uip_TcpAppHubCall
```

Then in uip_tcpapphub.c you'll find:

```
void uip_TcpAppHubCall(void)
{
 if(uip_conn->lport == htons(Port_Httpd)) {
   HttpDCall(uip_appdata, uip_datalen(), &uip_conn->appstate.HttpDSocket);
 }
```

```
}
```

And, as you'll recall from the earlier text, HttpDCall() is the webserver processing code.

Code Credits

This project borrows heavily from the work of Simon Kueppers "MicroWebServer" project available on GitHub. Extract of Simon Kueppers' code sharing statement:

> * Author: Simon Kueppers
> * Email: simon.kueppers@web.de
> * Homepage: http://klinkerstein.m-faq.de
> *
> This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.
>
> Copyright 2008 Simon Kueppers

Simon Kueppers work in turn borrows heavily from the work of Adam Dunkels uIP project, also available on GitHub and other locations. Extract of Adam Dunkels' code sharing statement:

```
/**
 * \file
 * The uIP TCP/IP stack code.
 * \author Adam Dunkels <adam@dunkels.com>
 */

/*
 * Copyright (c) 2001-2003, Adam Dunkels.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
```

```
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN
 * NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
 * OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH
 * DAMAGE.
*/
```

With the addition of the MQTT Interface the following statement is applicable:

> MIT License
>
> Copyright(c) 2018 Liam Bindle
>
> Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
>
> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
>
> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,

DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

I am likewise making my code available with this statement:

- Author: Michael Nielson
- Email: nielsonm.projects@gmail.com

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

See the GNU General Public License at <http://www.gnu.org/licenses/>.

Copyright 2020 Michael Nielson

And, finally, I want to credit Carlos Ladeira and Jevgeni Kiski for their very significant contributions to code and test.

Documentation License Note

And now for the GNU Free Documentation License (follows on the next page). If you read the short preamble the typical user will know all you really need to know.

GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not

explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that

translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take

reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These

may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a

translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/licenses/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be

used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.