

roZetta™

Home Automation Gateway

User Manual

(Linux Version)

roZetta™ is a trademark of Dave Houston.
BX-24 and BasicX are trademarks of NetMedia, Inc.
ZX-24a, ZX-40a and ZBasic are trademarks of Elba Corp.
EM202 and Tibbo Basic are trademarks of Tibbo Technology, Inc.
FRAM is a trademark of Ramtron International Corporation.
PureBasic is a trademark of Fantaisie Software.
mikroBasic is a trademark of mikroElektronika.
UPB is a trademark of Powerline Control Systems, Inc.
Insteon is a trademark of SmartLabs™ Inc.
Stargate is a trademark of JDS Technologies.
All other trademarks are the property of their respective owners.

roZetta™ is not to be used in life support devices or systems, if a failure of roZetta™ can reasonably be expected to cause the failure of that life support device or system, or to affect the safety or effectiveness of that device or system.

Chapter 1: Introduction and Overview

Introduction

roZetta[™] is a major upgrade of an earlier DIY Home Automation project, the BX24-AHT, that I designed a few years ago. That project, which used the BX-24 module from NetMedia, quickly used all of its capacity for expansion, in part, because the BX-24 only had a 32KB EEPROM for program and user data and, in part, because of the way I made use of that EEPROM.

roZetta[™] uses the ZX-40a chip from Elba Corp. which, while similar in concept to NetMedia's chips and modules, uses more advanced versions of Atmel AVR chips, more advanced virtual machine firmware, and a faster, more advanced dialect of the Basic programming language (ZBasic). By using the ZX-40a instead of the ZX-24a, the design can use a larger, external 64KB EEPROM. In addition, ZBasic is more efficient than NetMedia's BasicX dialect so the same program usually takes less EEPROM for program storage, leaving more room for user data, and **roZetta**[™] uses the EEPROM more efficiently, allowing for even more user data or feature expansion. Having found the 32KB of the BX-24 too small, I wanted to be sure that **roZetta**[™] had ample EEPROM and even an option to add still more. Development was delayed for 8-9 months because my original source for the 64KB EEPROM changed their MOQ from 1 to 2500 units. While I had already purchased 100 units, I did not want to introduce **roZetta**[™] only to find I could not continue to supply it with adequate EEPROM once the initial inventory was gone. It was only after finding another reliable source for the 64KB EEPROM with an MOQ of 1 unit that development was resumed.

While the BX24-AHT appeared to have four serial ports to interface with Home Automation devices, that was an illusion created by assigning a single software UART to different pins on the fly – none were truly full-duplex. The ZX-40a has four independent, low-speed, buffered, full-duplex software UARTs plus two TTL ports that can interface with X-10 OEM modules to send/receive X-10 PLC protocol in the background. Or, **roZetta**[™] can emulate an X-10 OEM module and send/receive X-10 PLC protocol with legacy controllers designed to operate with such modules. Plus, there is a high-speed hardware UART (**S0**) that is used to interface with a PC. Optional hardware provides a 10/100 network interface (Tibbo EM202), a battery-backed RTC and an 8KB FRAM (EEPROM) or up to 64KB of standard I²C EEPROM.

In addition to making sure there was ample EEPROM, my other design goals were to maximize flexibility and to use hardware supported by free or low cost Basic language compilers so users who wish to do so can create their own programs. (I think Basic is the most accessible programming language.) Where the BX24-AHT only handled the X-10 RF and PLC protocols, **roZetta**[™] handles X-10 RF, IR, and PLC protocols plus subsets of the UPB and Insteon PLC protocols. I have designed PIC based devices to interface serially or via RS485 that receive and report RF codes from various security devices. It can also interface with undefined devices that use ASCII or simple Binary protocols. In addition to ZBasic (free), the EM202 can be programmed in Tibbo Basic (free). The Windows and Linux applications are written in PureBasic (only \$99 and that gets you all platforms). The PIC based serial and RS485 devices I have designed to interface with **roZetta**[™] are programmed in mikroBasic which requires no license for applications smaller than 2KB (which is adequate for nearly all of my designs). Once everything looks fairly stable, plans are to publish all source code (where license allows) along with full details of the communications protocol.

Overview

The simplest way to understand **roZetta**[™] is to view it as an intelligent switchboard that allows inputs on any port (or pin) to trigger outputs on any other port (or pin), translating between protocols as necessary. For example, an X-10 PLC input on TTL port T2 from a legacy controller (e.g. JDS Stargate) can trigger an output (or outputs) on one or more of the other ports. There might be a TW523 on TTL port T1 which can send X-10 PLC codes and/or a UPB PIM or Smarthome 2412S on the serial ports S1-S4 which can send UPB and Insteon codes. By defining the actions to be taken in response to specific inputs (on a specific port) the user effects any needed translation between protocols. **roZetta**[™] *understands* and, with limitations, can interpret the communications protocols used by defined devices and reports all traffic in human readable form, triggering default actions for some protocols if the user enables this feature. Traffic for undefined devices such as those with ASCII or simple Binary protocols is reported in the same manner and while **roZetta**[™] cannot *understand* or interpret them, they can be used to trigger responses in the same manner as for defined devices but can only trigger responses specifically defined by the user, there are no default actions.

roZetta[™] has two (optionally three) EEPROMs for storing non-volatile data. 2KB of EEPROM is internal to the ZX-40a chip. This is too small for program data or much user data and has the added disadvantage that it is only guaranteed for 100,000 write/erase cycles. A small part of it is used by the ZBasic virtual machine for storing its configuration data. **roZetta**[™] uses another small part for storing its configuration data which changes infrequently. The ZBasic virtual machine firmware is stored in the ZX-40a flash memory. The main 64KB EEPROM (ST M95512) is external to the ZX-40a and communicates with the ZX-40a via an 7.37MHz SPI bus. It can withstand 1,000,000 write/erase cycles. It is used for storing the compiled **roZetta**[™] firmware (starting at address 00000) and user data (stored above the program firmware). ZBasic has built-in methods for rapid access to this main EEPROM. A third, optional EEPROM is on a plug-in daughterboard that also contains a battery-backed RTC. The third EEPROM can be either 8KB of FRAM (a special type of memory that can withstand billions of write/erase cycles) or 8KB FRAM plus 64KB of standard EEPROM (1,000,000 write/erase cycles). This optional daughterboard communicates with the ZX-40a over a slower 400kHz I²C bus. The FRAM can be treated as if it were RAM so it can be used for things like a state table to track the status of all controlled Home Automation devices or for daily randomization of selected scheduled events. All data stored in any EEPROM survives power cycles. Should the 64KB SPI EEPROM again become difficult to source, the main EEPROM can be 32KB (readily available) which should be adequate for program storage and the optional 64KB I²C EEPROM can be used for user data, offering some degree of future proofing.

The ZBasic virtual machine includes a software RTC. This is used for normal scheduling but it resets to 00:00 on January 1, 1999 whenever the ZX-40a is reset. If **roZetta**[™] is always connected to a PC or network, it can be configured to automatically request a synchronizing time-stamp to reset the software RTC after each reset. For standalone operation, the optional battery-backed RTC is needed to reset the software RTC. The RTC board can also supply a 64Hz squarewave which can be used to simulate ZC for cases where a legacy controller is used but there is no need for a TW523. (Some legacy controllers may not like 64Hz.)

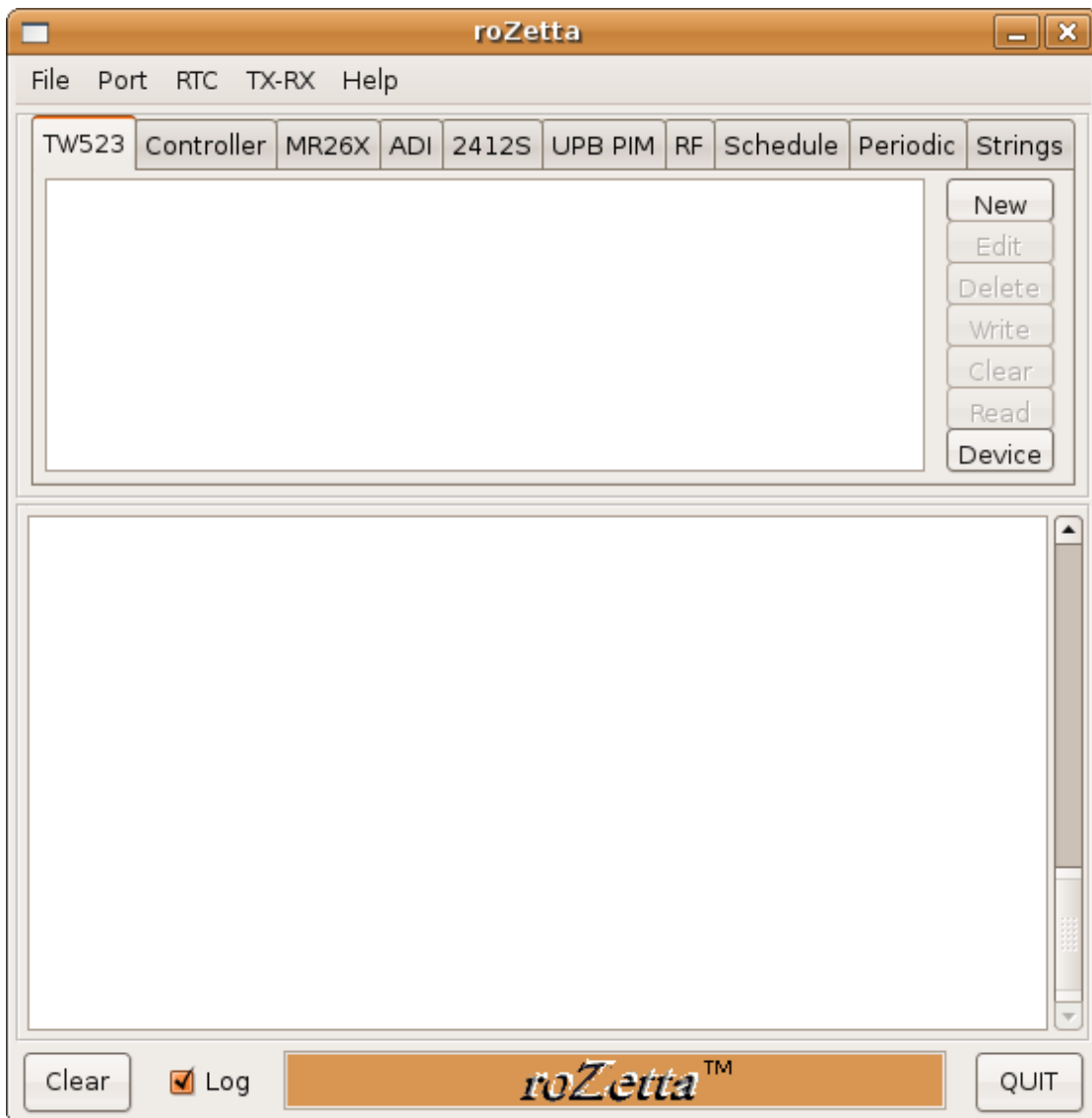
Figure 1, on the following page, shows the main screen for the interface interface to **roZetta**[™]. The top section contains a panel with ten tabs. The first seven tabs are associated with the two TTL ports (T1, T2), the four low-speed serial ports (S1, S2, S3, S4) and with the onboard wireless port (W0) which can be optionally configured for IR or RF.

Each port is associated with a contiguous block of EEPROM (at a user selected address) where the input triggers, output responses and other data associated with each entry are stored. The entries are displayed independently for each tab. The data for the trigger/response tables are entered and managed using the column of buttons to the right of the trigger/response display windows. Because the different protocols have different command lengths, the user must organize both the base address and record size for each tab.

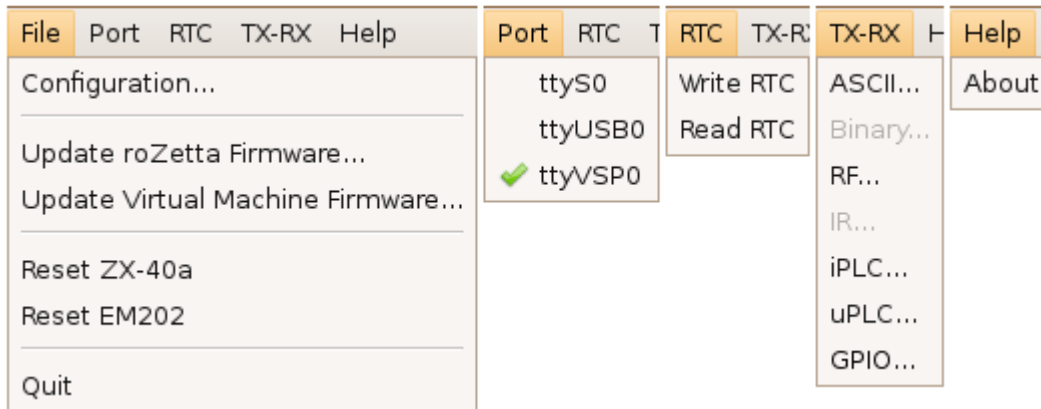
In addition to the seven ports, there are tabs for events scheduled by time of day, for events that repeat periodically (e.g. querying a sensor), and for user defined ASCII strings which can be referenced by address and can contain embedded variables which are updated for output.

All of the input/output traffic and configuration details are sent to the PC via the high-speed serial port (S0) or optional network interface and is displayed in the Input/Output window that occupies the lower portion of the main screen. This data can be automatically logged to a file on the HDD with the filename automatically changing monthly (e.g. Jan2008.log).

FIGURE 1: roZetta™ Main Screen



roZetta™ Menu Items



Most of the menu items under **File** and **TX-RX**, shown in **Figure 2**, above, launch secondary data entry windows and will be covered in detail on the pages that follow. The others are covered below.

The **Reset ZX-40a** and **Reset EM202** items under the **File** menu send hardware reset commands to the appropriate device. (The EM202 reset command is actually sent from the ZX-40a, acting on a command from the PC.) If there is a need to send reset commands, reset the ZX-40a first and then reset the EM202. The EM202 command may fail if the ZX-40a is in need of a reset. Both the ZX-40a and EM202 reset when power is cycled.

The **Port** menu displays the available COM ports, including any VCP (Virtual Com Port) or VSP (Virtual Serial Port) ports associated with USB-Serial and Ethernet-Serial adapters. **roZetta™** only supports 100 COM ports (which can still have names as high as COM256) but this can be increased, if needed. At startup, the interface application reads the **ports =** key in the **user.prf** preference file and presents the ports list in the menu. *Users must edit the file to list the ports.* Once a port is selected, it will be written to the preference file and **roZetta™** will automatically connect to it at subsequent startups.

The **RTC** menu **Write RTC** item synchronizes the **roZetta™** RTC with the PC's current date and time. If **roZetta™** has been configured for the optional battery-backed RTC, it will automatically be synchronized in the same operation. The **Read RTC** item will display the data reported by **roZetta™** for its software RTC and, if present, the battery-backed RTC. The battery-backed RTC uses the ST M41T81 chip which includes calibration registers that can keep the RTC accurate to ± 3 seconds per month once calibrated. The M41T81 also records a power-down time-stamp whenever power is lost and it switches to battery power. When power is restored, the time-stamps for power loss and power restore are saved to internal EEPROM and can be accessed as desired. If the optional EM202 ethernet interface is installed, **roZetta™** can be programmed to periodically connect to a Network Time Server to keep the RTC accurate. (This may require reprogramming the EM202.)

At this time, the Help menu only has an About box. Once *roZetta*™ is stable, I will try to create a compiled help file. In the meantime, this manual and the online [forum](#) will have to suffice.

Figure 2 : The About Box



I hope to make this an open source project but there may be areas where license terms prevent releasing detailed source code. Where possible, I will handle these areas with compiled library files that hide restricted details but allow publishing the higher level Basic language source. Unfortunately, library files are not yet possible in ZBasic so there may be some areas which cannot be open.

Chapter 2: Getting Started

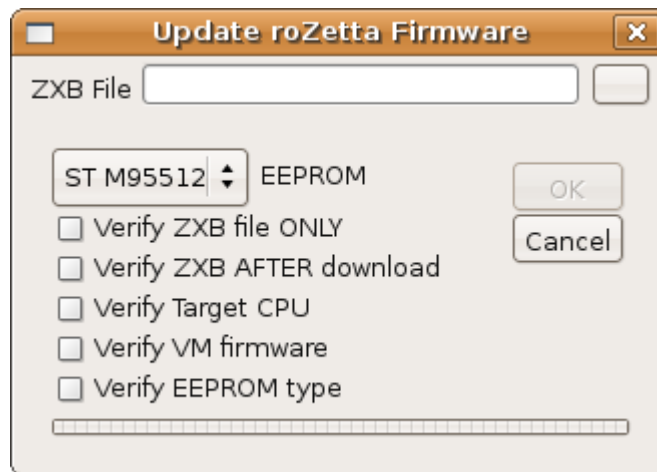
Once the assembly is finished, the interface application needs to be installed in order to communicate with the hardware via the high-speed hardware UART on S0 (DB9F). The distribution file, roZetta.zip includes...

roZetta.exe	interface application
roZetta.prf	preferences file (configuration data)
user.prf	user preferences
holiday.txt	USA Federal holiday list
roZetta User Manual.pdf	this user manual
roZetta.zxb	compiled application firmware file
zx40a_X-X-X.zvm	latest ZBasic ZX-40a virtual machine firmware

All files should be extracted to the folder designated for **roZetta™** (e.g. C:\roZetta). Once the files are extracted, launch roZetta.exe and select the COM port to which the hardware is connected.

The ZX-40a chip ships from Elba Corp. preloaded with the latest version of the ZBasic virtual machine firmware. However, the ST M95512 EEPROM chip cannot be preloaded with the latest **roZetta™** firmware so the first task after finishing assembly and installing the interface application is to download the roZetta.zxb file to the EEPROM. Clicking the **Update roZetta Firmware...** item in the **File** menu will launch the update window shown in **Figure 3**, below.

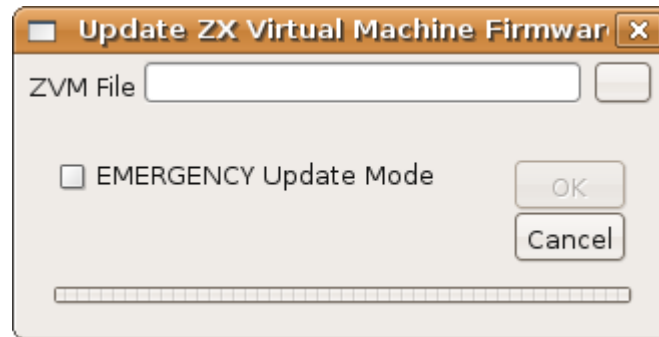
Figure 3: Update roZetta Firmware



Click on the **Browse** button (...) to launch a file browser and select roZetta.zxb. The **OK** button is disabled until a file is selected. Make sure the ST M95512 EEPROM is selected. The first option, **Verify ZXB file ONLY**, will scan the file to assure it is not corrupt without actually downloading it. If it is selected, the other options are disabled. Any or all of the other options can be applied when (or just before) the file is downloaded. Selecting any of **Verify ZXB AFTER download**, **Verify Target CPU**, **Verify VM firmware**, or **Verify EEPROM type** disables the first option. **Verify ZXB AFTER download** checks the records in the EEPROM itself after the download completes to assure there were no errors during the download.

When a ZBasic program is compiled, the CPU and EEPROM type must be specified. The compiler also notes the methods and functions actually used by the program code and determines the minimum Virtual Machine firmware version required. These data are embedded in the file. If any of the final three options is selected, the download procedure checks the configuration data stored in the internal EEPROM of the ZX-40a to make sure the CPU, EEPROM or VM version match the requirements of the .zxb file, blocking the download if a mismatch is detected. During any download or file verify operation, the progress bar indicates the percent completed.

Figure 4: Update Virtual Machine Firmware



Elba Corp. is very customer friendly, responding to feature requests whenever adding the requested feature is beneficial and is actually possible without impinging on or slowing down any fundamental procedures. Several current features were added at my request. New features require new Virtual Machine Firmware to be downloaded to the flash memory of the ZX-40a chip. Click the **Browse** button (...) and select the .zvm file. The **OK** button is disabled until a file is selected. Click **OK** to download. The progress bar will indicate the percent completed.

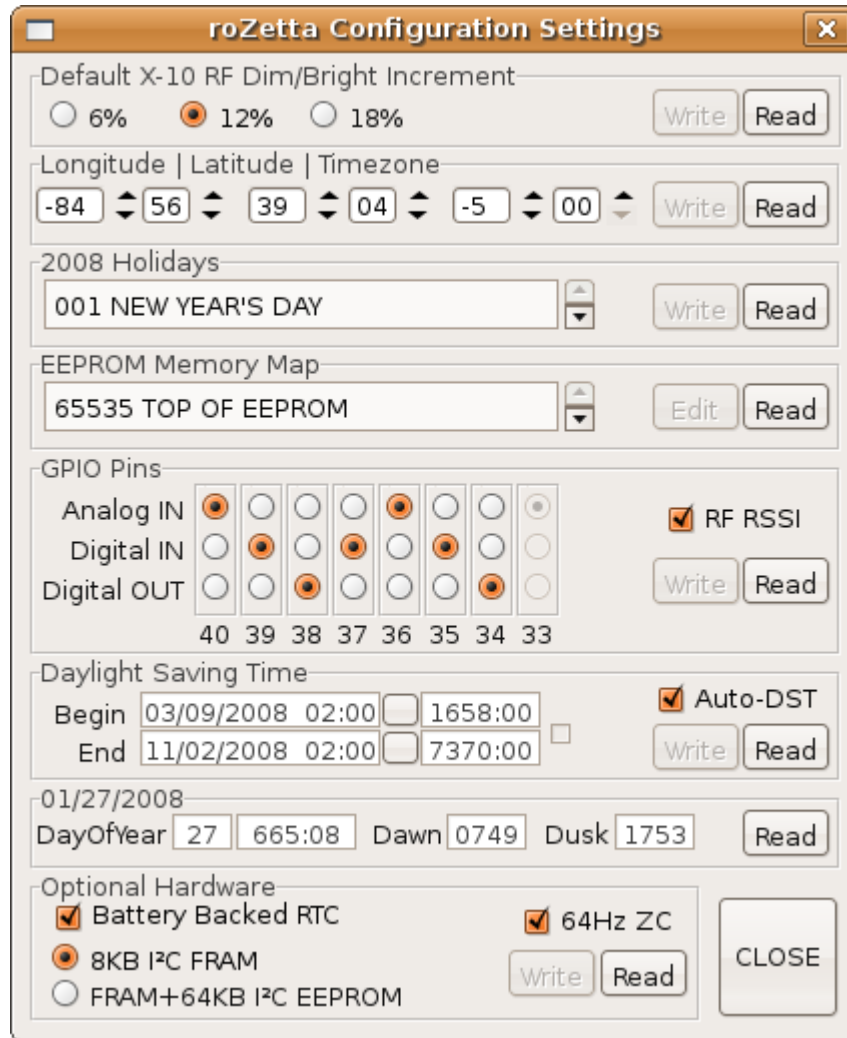
Emergency Update Mode

Because the timing of certain handshake signals may vary from one PC to another, there is an emergency procedure that can be used to download the Virtual Machine firmware if the standard method fails. This requires powering down the roZetta board and installing jumpers on the pins 1-2 and 3-4 of the 1x4 **EUM** header located adjacent to pins 4-6 of the ZX-40a chip. Select the **EMERGENCY Update Mode** option. When power is again applied to the hardware, the ZX-40a will be in Emergency Update Mode, ready to receive the Virtual Machine File. After downloading, remove power, remove all jumpers from the EUM header and restore power for normal operation. Refer to the picture of the top surface of the [roZetta™](#) printed circuit board in **Appendix A**. For **EMERGENCY Update Mode**, install the jumpers as shown in the small graphic just left of the header labeled **EUM**. For normal operation, remove the jumpers as shown in the small graphic just right of the header labeled **OK**.

Chapter 3: Configuration Details

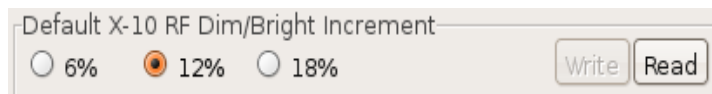
Clicking **Configuration...** in the **File** menu will launch the **Configuration** window shown in **Figure 5**, below.

Figure 5: Configuration Window



All of the configuration data is stored in two places, in the EEPROM and in a preferences file on the PC's HDD. When the Configuration window is launched, the data it displays is taken from the preferences file. As changes are made, the data displayed in the Configuration window may not agree with either until the associated **Write** button is clicked and the new data is written to both locations. The **Write** button is disabled until the data is changed. The data is first written to the EEPROM and only if this operation is successful is it written to the preferences file. Once the new data has been written to EEPROM and the preferences file, the **Write** button is again disabled. The **Read** buttons are always enabled. Clicking them causes **roZetta**™ to output the associated data which will be displayed in the **Input/Output** window. If any **Write** button is still enabled when the **Close** button is clicked, the user will be prompted to save or abandon the changes.

Default X-10 RF Dim/Bright Increment



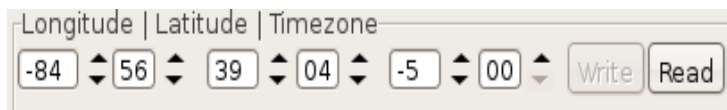
Default X-10 RF Dim/Bright Increment

6% 12% 18%

Write Read

Standard X-10 lamp modules and dimmer switches will go from full dim to full bright (or vice versa) with 21 contiguous power-line dim or bright codes. How many dim or bright codes are sent to the power-line varies with both the RF transceiver and with the RF remote used. Some RF remotes can send single copies of the RF code while others always send a minimum of five copies (six unless the button is pressed and released very quickly). The TM751 will respond to 1-5 RF copies by sending enough contiguous power-line codes to effect a 12% change. The RR501 will effect an 18% change with the same input. Transceivers from other manufacturers may operate differently. **roZetta**[™] can send 6%, 12% or 18% increments in response to 1-6 RF copies received, whether from its internal wireless port (**W0**) or from any supported X-10 RF receiver attached to any of the serial ports.

Longitude | Latitude | Timezone



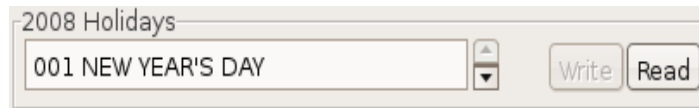
Longitude | Latitude | Timezone

-84 56 39 04 -5 00

Write Read

This data is used by **roZetta**[™] to calculate sunrise and sunset as each day of the year begins at 0000. The signs of latitude and timezone should be the same.

Holidays



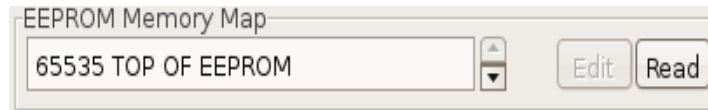
The interface application looks for a text file (holiday.txt) that contains a list of datecodes for all of the observed holidays and calculates the day of the year for each, adjusting as needed when a holiday falls on a weekend. This list needs to be stored in EEPROM and needs to be updated each year on the first day of January when the list will automatically be recalculated for the new year when the **Configuration** window launches. The update will occur automatically if **roZetta™** is connected to a PC (RS232 or network), which is running the interface application, at 0000 on 1 January. The list details will depend on the country. The datecodes in the holiday.txt file for USA Federal holidays are explained below. Users in other countries can create their own holiday.txt file.

New Year's Day=1/1,1;	0=static, 1=Fri<-Sat,Sun->Mon
MLK Birthday=3,1,1;	Third Monday in January
Washington's Birthday=3,1,2;	Third Monday in February
Memorial Day=L,1,5;	Last Monday in May
Independence Day=7/4,1;	0=static, 1=Fri<-Sat,Sun->Mon
Labor Day=1,1,9;	First Monday in September
Columbus Day=2,1,10;	Second Monday in October
Veterans Day=11/11,1;	0=static, 1=Fri<-Sat,Sun->Mon
Thanksgiving=4,4,11;	Fourth Thursday in November
Christmas=12/25,1;	0=static, 1=Fri<-Sat,Sun->Mon

The name of the holiday is given first followed by an equals sign (=). For holidays that occur on a date certain, such as New Year's Day, the month and day are separated by a slash (/). The day is followed by a comma and then a 0 or 1 depending on whether it should automatically be adjusted for weekends. For holidays that do not fall on a date certain, the first digit in the datecode indicates the ordinal rank (1=1st, 2=2nd, 3=3rd, 4=4th, N=next to last, L=last), the second digit is the day of the week (0-6) and the third digit is the month (1-12). The datecodes are terminated by the semicolon (;) with anything following constituting a comment.

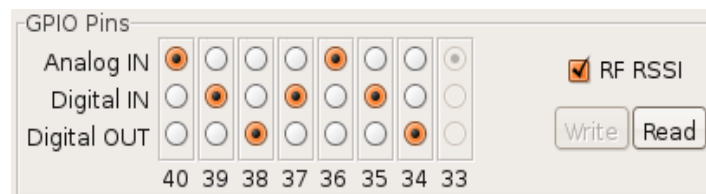
When defining the responses to input triggers, the days of the week, holidays and time-frame are potential qualifiers for each entry. **Schedule** and **Periodic** entries have no time-frame qualifier; **String** entries have no qualifiers.

EEPROM Memory Map



The EEPROM is divided into 12 memory blocks. The **roZetta**™ firmware is automatically loaded into the bottom of EEPROM starting at address 00000 and an area above it is reserved for future changes to the firmware in order to avoid major rewriting of the higher blocks as features are added. A list of holidays (saved as the days of the year, taking two bytes per entry) is automatically loaded at the top of EEPROM, building down from address 65535. The region between these two fixed, non-editable blocks is divided among the 10 tabs of the main screen. Since each user is likely to connect a different set of devices to **roZetta**™, it is left to the user to organize each memory block, specifying its base address and record size. Initially, each block is allocated 4,000 bytes but the user can rearrange this as needed. Selecting an editable memory block enables the **Edit** button. Clicking it launches another window where the details can be edited and saved. See **Managing EEPROM Memory Blocks** on page 16 for further details.

GPIO Pins



Pins 33-40 of the ZX-40a can be configured as 10-bit ADC, as digital inputs or as digital outputs. Optionally, pin 33 can instead be used (in ADC mode) as a Received Signal Strength Indicator (RSSI) when **WO** is configured for an RF receiver. This requires shorting the RSSI header and opening the /RSSI header. RSSI is only needed when tuning the RF receiver and/or positioning/orienting the antenna. The RSSI & /RSSI header pair allow for switching between RSSI mode and ADC or DIO mode without disconnecting any sensor or indicator on pin 33.

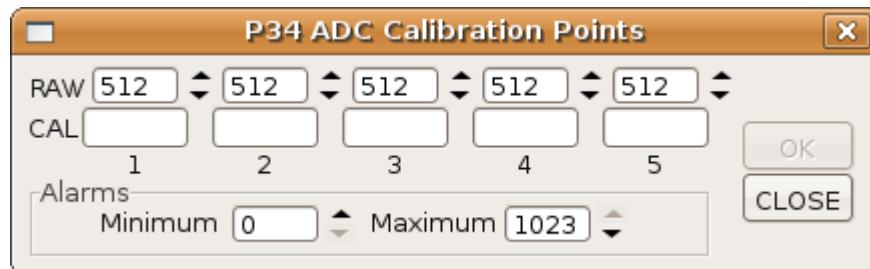
When configured as digital outputs, the pins can be used to send pulse-trains to Infrared or RF transmitters for control of AV gear, camera mounts, ceiling fans, etc. Pulse-trains require the optional RTC board with the extra 64KB I²C EEPROM. The data is stored in a compressed format in variable length entries referenced by the address similar to how strings are stored. Sending a pulse-train requires designating the output pin and address of the pulse-train.

#35<address>

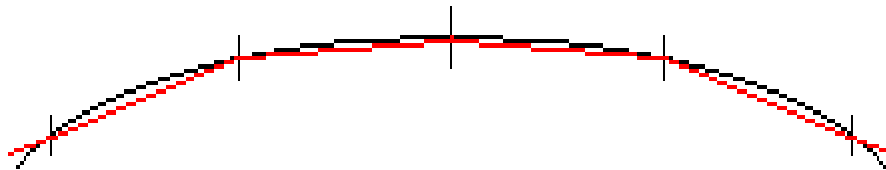
Once **roZetta**™ is stable and I have some time, I will rewrite some software I wrote a few years ago that can capture, decode and generate most IR and RF codes. It will have an option to output codes in the format needed by **roZetta**™.

Data for up to 5 calibration points is stored for each pin configured as an Analog input. **roZetta™** will use linear interpolation for readings that fall between the calibration points or beyond the minimum and maximum points. The **ADC Calibration** window, below, will launch whenever **Analog** is selected an **I/O** pin in the **Configuration** window.

Figure 6: ADC Calibration Window



Leave all five **CAL** values blank, initially. Record the **RAW** values for the pin and note the calibration value for each. Once all the data is collected, set the raw values (0-1023) and enter the corresponding calibration value. If fewer than five points are needed, leave the higher, unused values as shown above but do not leave uncalibrated gaps. The data can be determined empirically or derived from charts or formulas accompanying the sensor. Choose calibration points that span the range of interest. Choosing points near critical values will increase accuracy near those points.



The pin value can be embedded in **Strings** using the notation **~P nn d** where **nn** is the pin number and **d** is the number of decimal digits. Some examples follow.

- Temperature: **~P401!F** (e.g. Temperature: 98.6F)
- Barometer: **~P392!"Hg** (e.g. Barometer: 29.74"Hg)
- Humidity: **~P381!%** (e.g. Humidity: 66.4%)
- Pin 37: **~P37** (e.g. P37=622 or P37=1)

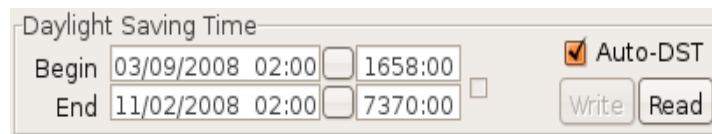
There are two methods for reading the pins and accessing the values. One is the embedded **String** method shown above. The second is to define a **Schedule** or **Periodic** event to read the pin using one of two variations in syntax. **~P40** will record the **RAW** value of pin 40 to FRAM (if installed) while, in an ASCII **String**, **~P40** will use the **RAW** value from FRAM and **~P40!** will use the current pin value.

A Periodic event can be used to output the values to the **Input/Output** window to gather empirical data needed for calibration.

Sending **?P40** from a PC will cause **roZetta™** to output the current RAW value of pin 40 while **?P40d** will cause **roZetta™** to output the adjusted value with **d** decimal places. If **d** is 0, as in **?P400**, **roZetta™** outputs the adjusted value with no decimal digits.

The same syntax works for pins configured as digital outputs but the output will always be **0** or **1** depending on the pin state.

Daylight Savings Time



Daylight Saving Time

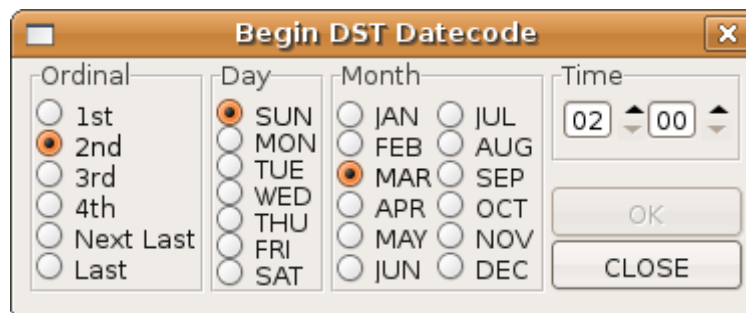
Begin 03/09/2008 02:00 1658:00 Auto-DST

End 11/02/2008 02:00 7370:00

Write Read

The interface application calculates when DST begins and ends as hours and minutes from 0000 on 1 January, based on datecodes stored in the *user.prf* file. The results are stored in EEPROM and must be updated whenever another year starts. The update will occur automatically if *roZetta*™ is connected to a PC (RS232 or network), which is running the interface application, at 0000 on 1 January. If the Auto-DST option is checked, *roZetta*™ will automatically adjust its RTC(s) as appropriate. The initial datecodes are those for the USA. New datecodes for other countries can be entered by clicking the buttons to the right of the date/time display.

Figure 7: DST Datecode Window



Begin DST Datecode

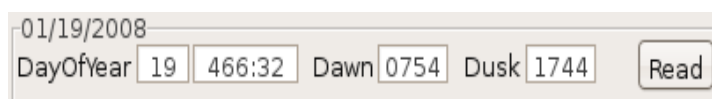
Ordinal	Day	Month	Time
<input type="radio"/> 1st	<input checked="" type="radio"/> SUN	<input type="radio"/> JAN <input type="radio"/> JUL	02 00
<input checked="" type="radio"/> 2nd	<input type="radio"/> MON	<input type="radio"/> FEB <input type="radio"/> AUG	
<input type="radio"/> 3rd	<input type="radio"/> TUE	<input checked="" type="radio"/> MAR <input type="radio"/> SEP	
<input type="radio"/> 4th	<input type="radio"/> WED	<input type="radio"/> APR <input type="radio"/> OCT	
<input type="radio"/> Next Last	<input type="radio"/> THU	<input type="radio"/> MAY <input type="radio"/> NOV	
<input type="radio"/> Last	<input type="radio"/> FRI	<input type="radio"/> JUN <input type="radio"/> DEC	
	<input type="radio"/> SAT		

OK

CLOSE

Clicking the **OK** button will update the DST data in the **Configuration** window but the data will only be written to EEPROM and the preferences file with the **Configuration** window **Write** button.

Astronomical Clock



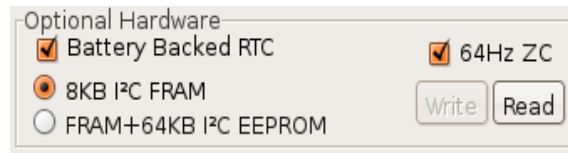
01/19/2008

DayOfYear 19 466:32 Dawn 0754 Dusk 1744

Read

roZetta™ calculates sunrise and sunset each day based on longitude, latitude, timezone and DST status. The data displayed in the **Configuration** window is calculated by the interface application. It does not need to be downloaded but clicking the **Read** button will output the data calculated by *roZetta*™ to the **Input/Output** window for comparison.

Optional Hardware



If the optional plug-in battery-backed RTC is installed, check the **Battery Backed RTC** option and select the type of EEPROM used. If the **Battery Backed RTC** option is unchecked, the EEPROM radio buttons and the **64Hz ZC** option for a **Controller** on **T2** are disabled.

Managing EEPROM Memory Blocks

Figure 8: EEPROM Memory Blocks Window



The **EEPROM Memory Blocks** window, shown in two configurations in Figure 8 above, can only be accessed from the **Configuration** window by selecting a specific memory block and clicking the **Edit** button. Changes are saved to EEPROM and the preferences file by clicking the **Write** button associated with the specific memory block. Saved changes will appear in the **Configuration** window when the **EEPROM Memory Blocks** window is closed.

If the optional RTC board with 64KB of additional EEPROM is installed, checking the box adjacent to the **Write** button, only enabled when the optional board is present and blocks above have been moved, will move a block to that EEPROM. This must be done in top down order, starting with **Strings**. Moved blocks will retain the same top down order. Base addresses (00000-65535) are increased by 100,000 for the blocks residing in the optional EEPROM. **Holidays** remain in the main EEPROM.

Each of the ten tabs in the top portion of the main screen has an associated block of EEPROM for storing user defined trigger/response tables, schedules, periodic event tables and ASCII strings. The mix of data types and protocols, both defined and undefined, makes it impossible to predefine a memory footprint that will work for all so it is left to the user to allocate the base address and record size (10-25 bytes) for each block (except for **Schedule** with 5-25 since no time-frame is needed). Of course, there are some constraints. The lower portion of the EEPROM is occupied by the *roZetta*[™] firmware plus a block reserved for future growth and the top portion of the EEPROM is occupied by the optional list of annual holidays.

All EEPROM between these limits is available for user data. Initially, each tab is allocated 4000 bytes and the record sizes are all set to 10 bytes. Strings are stored contiguously with the first byte indicating the length of the string and are accessed by referencing their address. The maximum string size is 255 characters although anything greater than 25-30 characters is impractical for processing due to limited buffer sizes for all serial ports.

The order of the blocks is fixed. Blocks for any unused tabs cannot be deleted but can shrink down to a 20 byte placeholder. Each base address can be increased or decreased until it bumps up against the base address above or bumps into the final record or placeholder of the block below.

Each trigger/response entry includes 2-5 bytes for qualifying flag bits plus a continuation flag. The length of the triggers and responses will vary from one device to another (and can even vary for the same device). Record sizes should be adjusted to accommodate the majority of entries for each tab. Shorter entries will waste the balance of the record. Longer entries will spill over to the following record or records, automatically setting continuation flags, and will waste any unused balance of the final record. There can be multiple entries with the same trigger to allow multiple responses to the same trigger or allow entries where both trigger and response are the same but qualifiers like day of the week, holiday and/or time-frame differ.

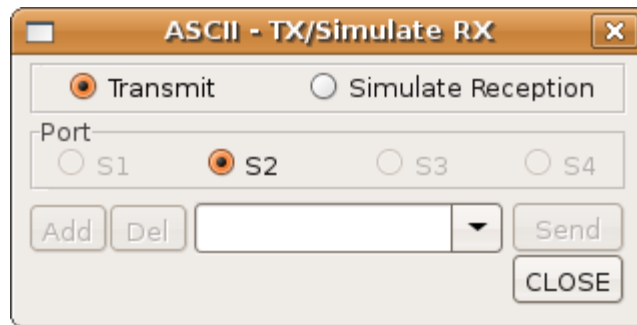
When new entries are inserted or when a base address or record size are changed, the entire occupied portion of the memory block must be rewritten. While the EEPROM can withstand 1,000,000 write/erase cycles, it is good practice to minimize rewrites by good organization and planning, making all edits to a block before writing to EEPROM.

Chapter 4: TX-RX Menu Details

The **TX-RX** menu items launch dialogs which allow for testing of configuration details by sending codes, using appropriate protocols, to the hardware connected to the serial ports and to simulate the reception of codes from the hardware for testing of user created trigger/response tables. The menu items enabled and the options available will reflect the current configuration of the ports and capabilities of attached hardware. Choosing the **Transmit** option will cause the data entered to be sent to the hardware on the selected port. This is useful for testing port and hardware configurations. Choosing the **Simulate Reception** option will simulate the event and allow testing of trigger/response entries.

ASCII

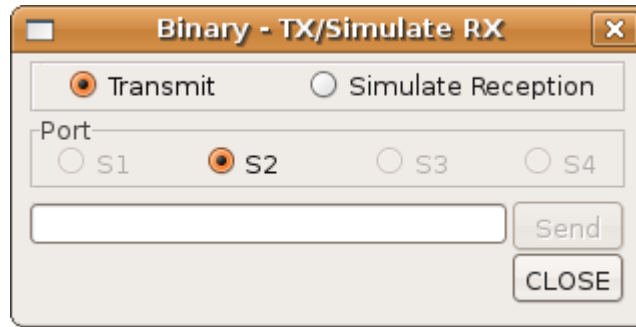
Figure 9: ASCII Dialog



ASCII codes should work with any devices that use ASCII protocols. Occasional binary codes can be inserted as hexadecimal bytes enclosed in square brackets like **[02F3]**. This allows for use with serial LCDs that may need binary codes to position the cursor, etc. that precede ASCII data for display. ASCII codes can include embedded variables which will be replaced by the value of the variable taken either from optional FRAM tables (this requires a periodic event definition) or by forcing a fresh reading. The notation **~P36d** will insert the most recent value recorded for pin 36 from FRAM (with *n decimal places*) while **?P36d!** will insert the current state of pin 36. For example, if pin 36 is set for ADC and is attached to a temperature sensor, the ASCII string **Temperature = ~P36d** will replace **~P36.n** with the latest value in FRAM and send the string **Temperature = <value>** out port S2 (if it has been configured as ASCII). If the string **Temperature =** is stored in EEPROM, **<address>?P36d** (where **<address>** is the EEPROM address of **Temperature =**) will accomplish the same thing. The latter form minimizes EEPROM use when defining trigger/response events and allows the string **Temperature =** to be used with other embedded variables. For pins defined as digital inputs, possible values are 0 for logic low and 1 for logic high.

Binary

Figure 10: Binary Dialog

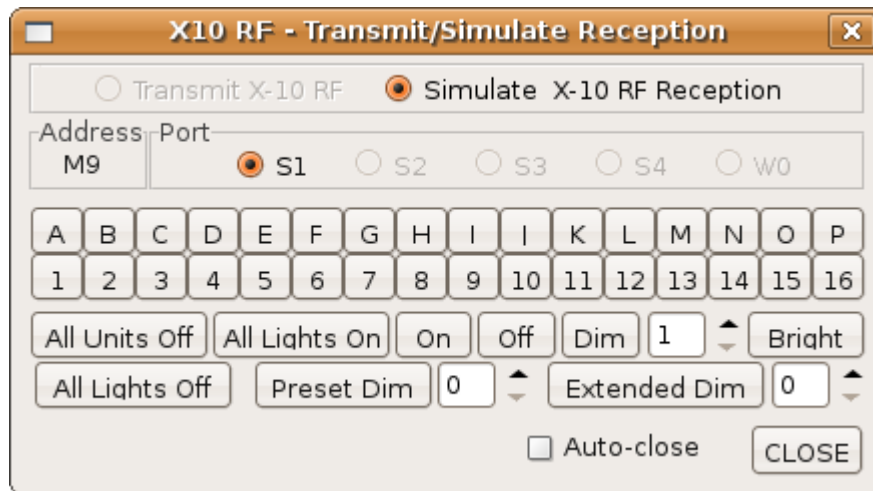


Simple binary protocols are supported using hexadecimal strings without brackets. Very simple handshaking is supported through the use of the WAIT command which can take the forms **WAIT SUM8** or **WAIT value** where the first will wait for the device to return an 8-bit checksum for the previous hexadecimal string it was sent and the second will wait for it to return a specific hexadecimal value (e.g. ACK).

02F1558C WAIT SUM8 02D21F2407 WAIT 06

RF

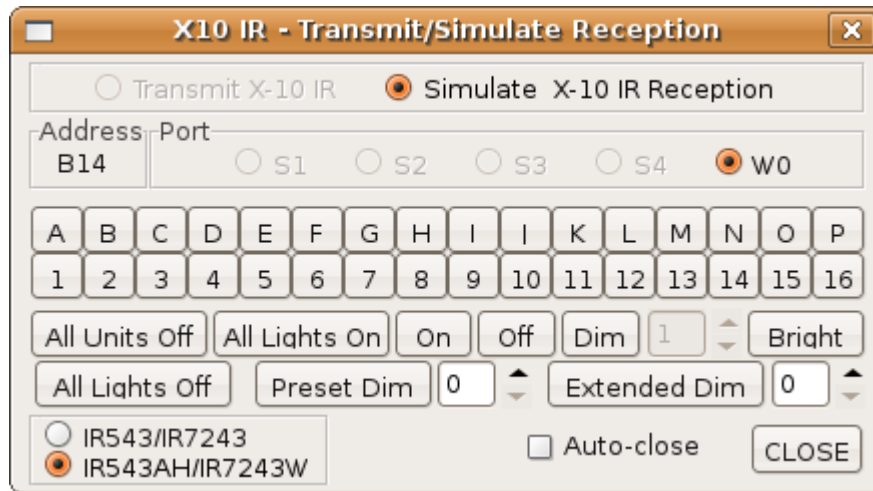
Figure 11: X10 RF Dialog



Available functions will vary with the capabilities of the device connected to the selected port. Preset and Extended Dim codes can only be sent by programmable RF remotes as these are extensions to the X-10 RF protocol created by DLH. Not all transceivers respond to all codes.

IR

Figure 12: X10 IR Dialog

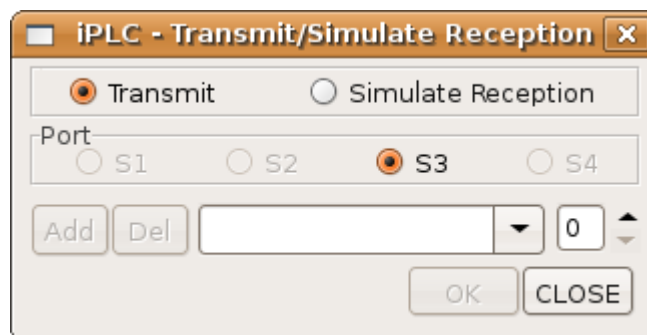


Available functions will vary with the capabilities of the device connected to the selected port. In IR543AH mode, codes can only be sent by programmable IR remotes as these are extensions to the X-10 IR protocol created by Laser Systems and later copied by X-10 Europe for the IR7243W.

In IR543 mode, the housecode is read from the user.prf file. It can be changed on-the-fly by clicking the current housecode key. This will enable the other keys and clicking one will reset the housecode and then again disable the other keys.

iPLC

Figure 13: iPLC Dialog

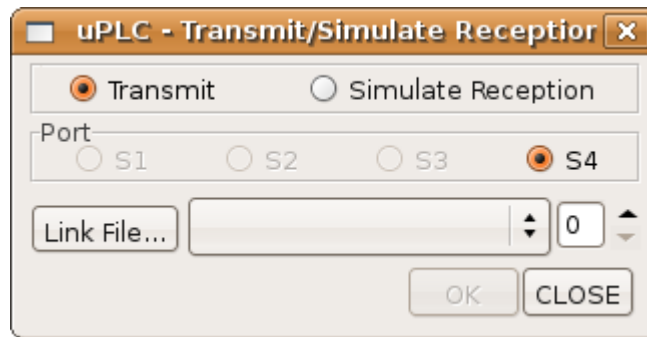


To send commands, select a device from the drop-down list, set the desired level (0-100%) and click the **OK** button. Devices can be added to the drop-down list by entering the ID (24-bit number assigned at the factory for each Insteon device) followed by a bracketed X-10 address (.. if none is assigned) plus an identifying description and clicking the **Add** button. Clicking the **Del** button will delete the selected device. *roZetta*™ does not link the devices with the serial interface. That must be done manually or by using some other PC software.

When adding Insteon devices to the database, they can each be assigned an alias X-10 address which will be used to translate **Extended Dim** or **Preset Dim** to Insteon addresses and levels when this is defined as the default action. Descriptive labels can also be appended to simplify keeping track of aliases.

uPLC

Figure 14: uPLC Dialog

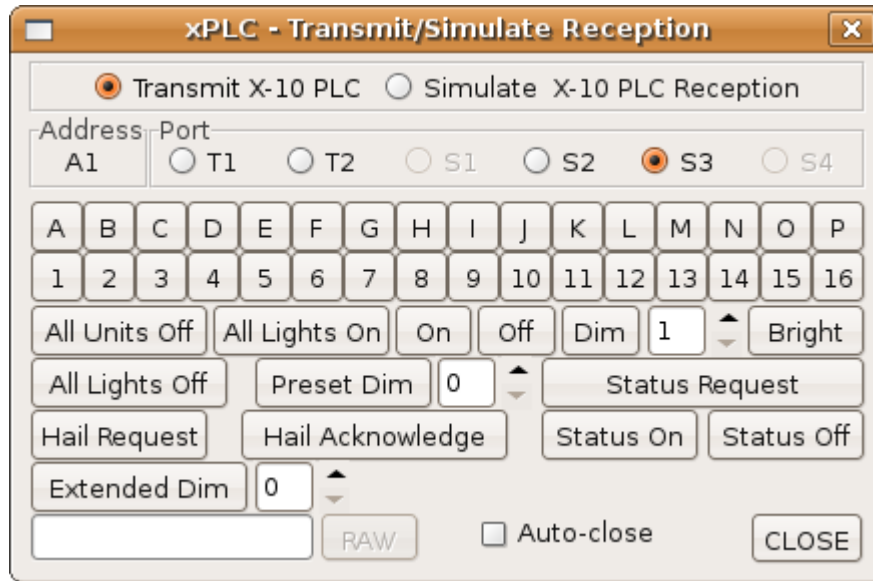


To send a command, select a device from the drop-down list, set a level (0-100%) and click the **OK** button. Creating a UPB network and linking all the devices is a complex and time-consuming task for which the ZX-40a and limited code space are not well suited. **roZetta™** can import a list of devices from the export file created by **UPStart Configuration Software** from SimplyAutomated.com which can be used to enumerate and organize the UPB network. Click the **Link File...** button to locate the UPStart export file.

UPB devices in the database can each be assigned alias X-10 addresses which will be used to translate **Extended Dim** or **Preset Dim** to UPB addresses and levels when this is defined as the default action. Descriptive labels can also be appended to simplify keeping track of aliases.

xPLC

Figure 15: xPLC Dialog



Available functions will vary with the capabilities of the device connected to the selected port. Raw X-10 PLC codes can only be sent using TTL interfaces on T1 or T2 or the CM15X-S on S1-S4.

Raw mode requires three fields separated by spaces. The first field is a decimal number (0-63) representing the number of times to repeat the sequence. The second field is a 2 digit binary number defining the initial ZC state. (00=falling edge, 01=rising edge, 10,11= next edge) The final field is a binary bit-stream (MSB first) with each digit representing one half cycle of the power-line. (1=120kHz burst, 0=idle)

For example, **1 11 1110011010010110100101** will send a standard **A1** starting with the next ZC, repeated once.

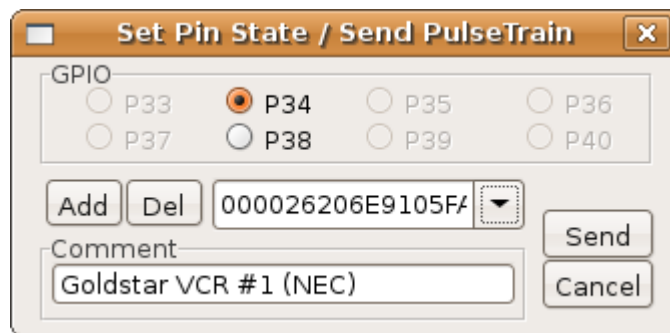
GPIO

If any of the GPIO pins 33-40 are configured as digital outputs, individual pin states can be set high or low under program control. *If the optional RTC daughterboard with the extra 64KB EEPROM is installed*, digital outputs can be used to send pulse-trains to amplified IR emitters, IR transmitters or RF transmitters for control of AV gear, camera mounts, ceiling fans, etc.

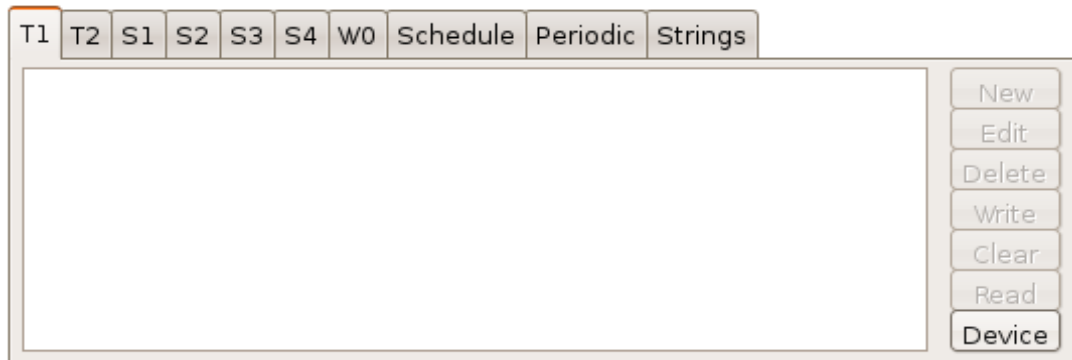
The pins can sink or source only a small amount of current so loads which require more than a few milliamps will require additional electronics to limit the current. LEDs can be driven directly as long as there is a 330 ohm (or higher) resistor in series with the LED to limit the current. In addition, the total current for all of the GPIO pins 33-40 is limited to 100mA.

The **roZetta™** +5V bus can be used to supply power to transmitters as long as the current is modest. To avoid damage to the **roZetta™** printed circuit board, all power leads to peripheral modules should be fused, inline and close to the **roZetta™** board. Most RF transmitters use about 5mA but IR transmitters can draw far more. Anything needing more than 20-25mA should have its own power supply.

Figure 16: GPIO Dialog



Chapter 5: Triggers and Responses



Each of the ten tabs on the main screen is associated with a block of EEPROM for storing lists of triggers and responses or user defined ASCII strings. The first seven of the tabs represent two TTL ports (**T1,T2**), four low-speed serial ports (**S1-S4**) and the internal wireless port (**W0**) to which various devices can be assigned. In these cases, the triggers are signals input to the ports and the responses can be signals sent out on the other ports. Scheduled events are triggered by time of day. Periodic events are ones scheduled to repeat at certain intervals (in minutes), such as querying a sensor on one of the eight I/O pins. Strings do not have associated triggers but are for use in ASCII responses to other triggers. The lists are managed through use of the buttons to the right of the list box for each tab.

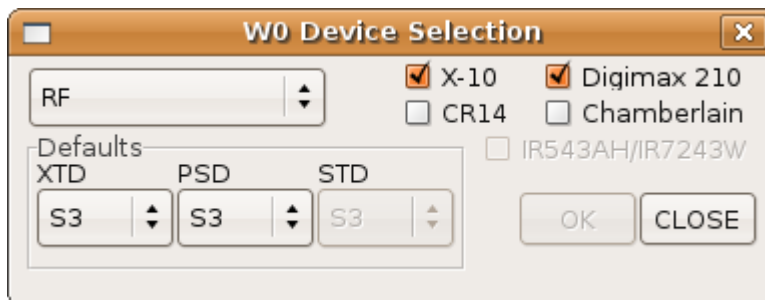
Assigning Devices To Ports

When no device has been assigned to a port, the tab name defaults to that of the port itself as shown above. Clicking the **Device** button launches the **Device Selection** dialog. The devices and options available will depend on the port selected. Only **T2** has the **Controller** option and only **S4** has the **RS485** option. Otherwise, **T1** has the same device list as **T2** and **S1-S3** have the same device list as **S4**. **W0** has the **IR** and **RF** options.

If **T2** is used for a **Controller**, there are options for selecting the type of connection (opto-coupled or direct) and setting the extended dim range (0-31 or 0-63) for the specific controller used.

If **S4** is used for 0-5V **TTL RS232**, there are options to select the polarity (inverted or non-inverted) of the signal. In all cases, select **NONE** if the port is unused.

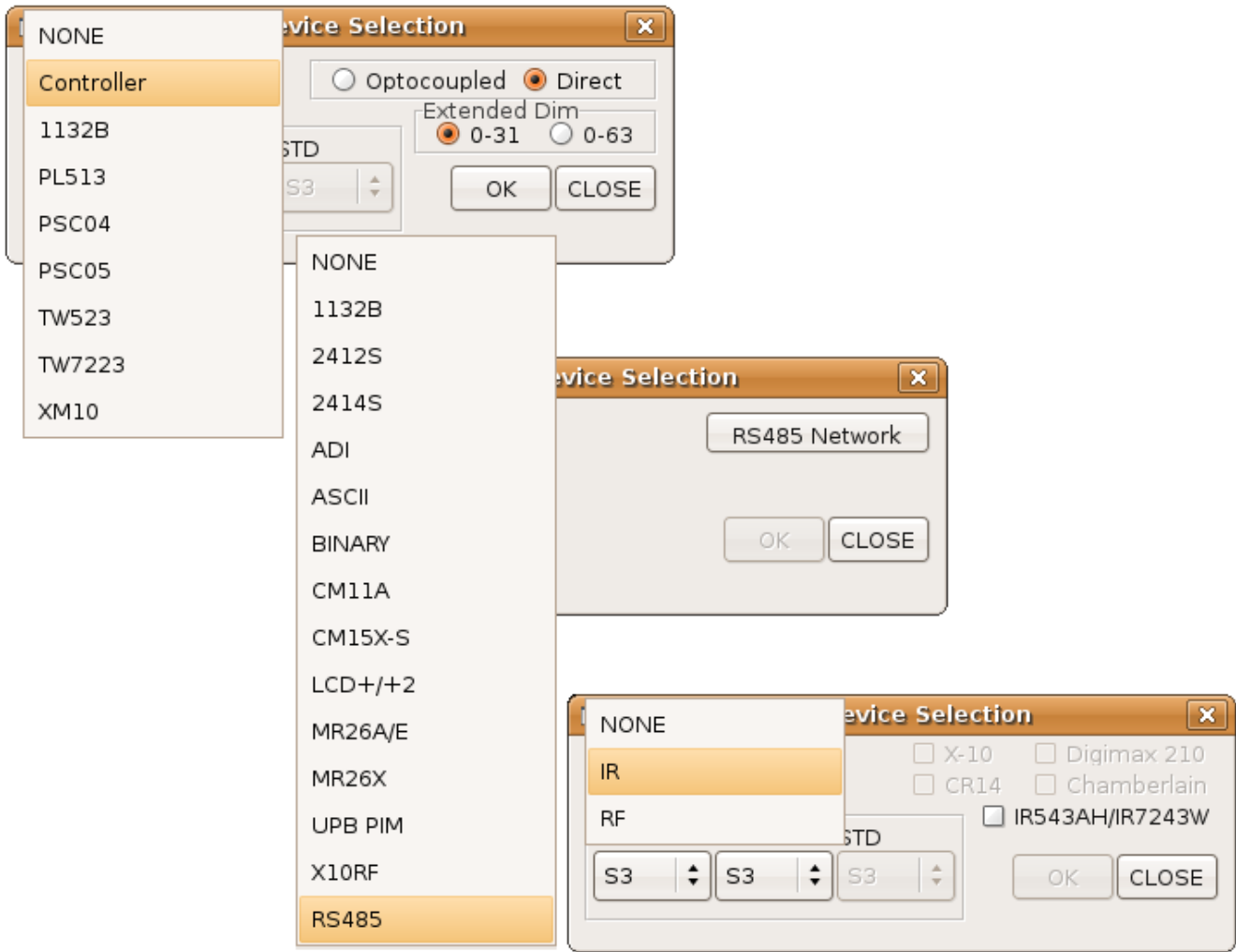
Figure 17: Device Selection Dialog



For most X-10 devices, depending on specific device capabilities for both input and output channels, separate default ports can be selected for **Extended Dim** codes (**XTD**), **Preset Dim** codes (**PSD**) and **Standard X-10** codes (**STD**) and *roZetta*™ will automatically translate the received code to the proper protocol for transmission. Custom responses can be defined if there is no default or for individual codes even if there is a default selection. There are no similar defaults for **ASCII** or **Binary** devices but their received codes can be assigned responses individually. **ASCII** or **Binary** devices, however, can be assigned to **S0** as the default and they will be treated *as if* they were connected to **S0**. This allows *roZetta*™ to interface with devices like SimpleLAN (on **S4** only) or a Global Cachè GC-100 (**S1-S4**) to transfer inputs and outputs over a network while still using **S0** for another serial connection.

One use for defaults might be to assign Extended Dim or Preset Dim codes input from a legacy Controller to a port with a 2412S or 2414S for sending codes to Insteon devices (while still sending X-10 codes to X-10 devices) or to a port with a UPB interface. When Extended Dim or Preset Dim codes are assigned by default to Insteon or UPB interfaces, they will automatically be translated to address and level codes in the appropriate protocol based on the assignment of alias X-10 addresses to the Insteon and UPB devices. If no matching alias is found, the code will be sent as X-10 extended or preset dim, as appropriate.

Figure 18: Device Selection Dialogs



Entering and Editing Trigger/Response Data

Because it supports so many different devices, each with its own protocol, **roZetta**[™] uses its own universal protocol for defining the various X-10 inputs and outputs and translates to/from it when communicating with specific devices. The **roZetta**[™] universal protocol is loosely based on X-10's PLC protocol. It defines all X-10 commands in three hex bytes. The first two bytes define House and Key codes (HKHK). The first byte (HK) is always an address. If BIT0 of the third byte is non-zero, it means the preceding byte is a function. For **Extended Dim (XTD)** and **Preset Dim (PSD)**, BIT1-BIT7 of the third byte define the level (0-63 or 0-31) and for standard Dim/Bright commands it defines the number of contiguous PLC Dim/Bright codes (1-21). The protocol always uses X-10 function 7 (hex 05) to represent an extended dim. If any of the other extended functions are received, they are reported as raw hex (longer than 3 bytes). For example, **D6D53F** defines **H1 XTD31** which is a 50% extended dim level for X-10 address H1.

When a code is received, the Input/Output window will display a timestamp followed by the input port designation, a < to denote it's an input, and the hexcode (which is underlined below). This will be followed by a human readable description in parenthesis, For example:

```
15:55:11 T2<D6D53F (xPLC H1 50%)
```

If the user has defined an alias for H1 (in *user.prf* file), it will be inserted in place of the address in the description.

```
15:55:11 T2<D6D53F (xPLC MBR 50%)
```

Insteon (and UPB) inputs are reported as four hex bytes. The first three represent the address and the fourth represents the level. For example:

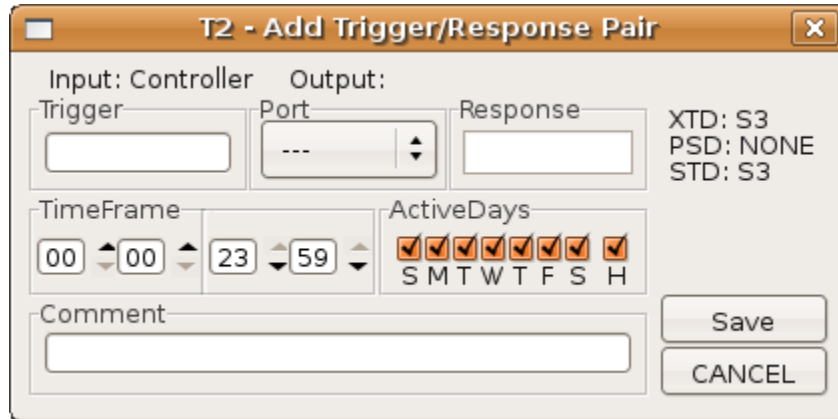
```
15:56:42 S3<0144DF7F (iPLC 01.44.DF 50%)
```

Again, if the user has defined an alias for the address, it will be inserted instead.

```
15:56:42 S3<0144DF7F (iPLC MBR 50%)
```

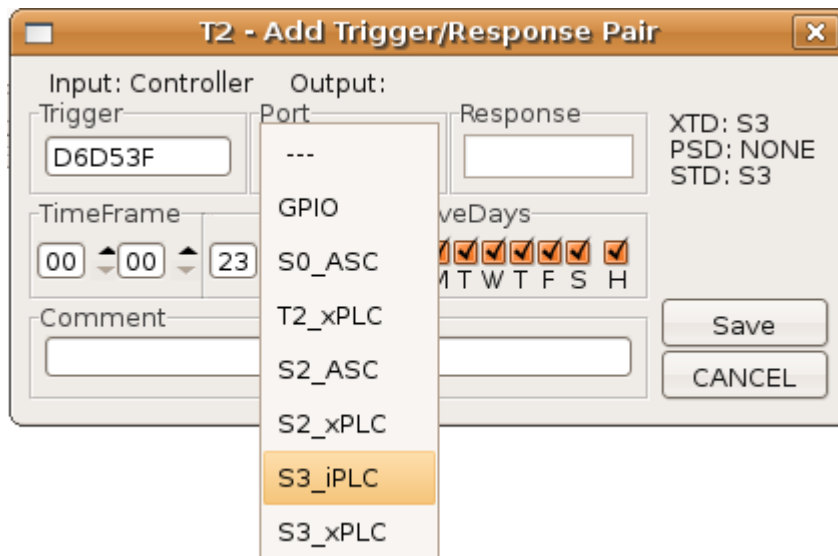
To add a **Trigger/Response** entry, first highlight the code to be used for the Trigger by double-clicking it in the **Input/Output** window (simulate, if necessary), then right-click the highlighted text and copy it to the clipboard. Select the tab associated with the input, click the **New** button to launch the **Add Trigger/Response Pair** dialog (shown on the following page) and paste the code into the **Trigger** box.

Figure 19: Trigger/Response Dialog #1



Entering a valid **Trigger** code will enable the **Output Port** drop-down list-box as shown in Figure 17 below. What constitutes a valid **Trigger** code depends on the device connected to the currently selected input port. **roZetta™** will validate *most* entries, only allowing selection of an **Output Port** for a valid entry, but **roZetta™** cannot validate **ASCII** or **Binary** entries as these are undefined. (ADI devices may use either.) The simplest way to assure an entry is valid is to copy/paste it from the **Input/Output** window, either from an actual or simulated input.

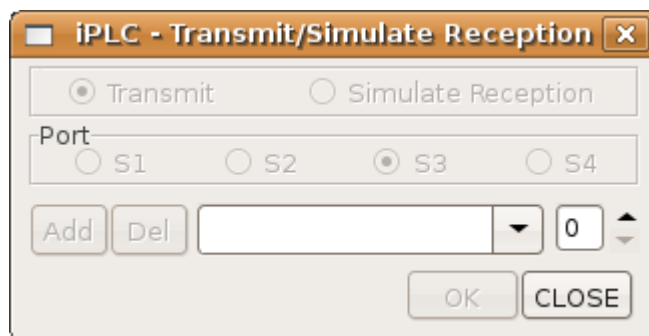
Figure 20: Trigger/Response Dialog #2



The ports and protocols available will depend on the current hardware configuration. Selecting an **Output Port** and protocol will launch the appropriate **TX-RX** dialog where the **Response** code can be generated and automatically pasted into the **Response** box which is *read only* to assure that all entries are valid.

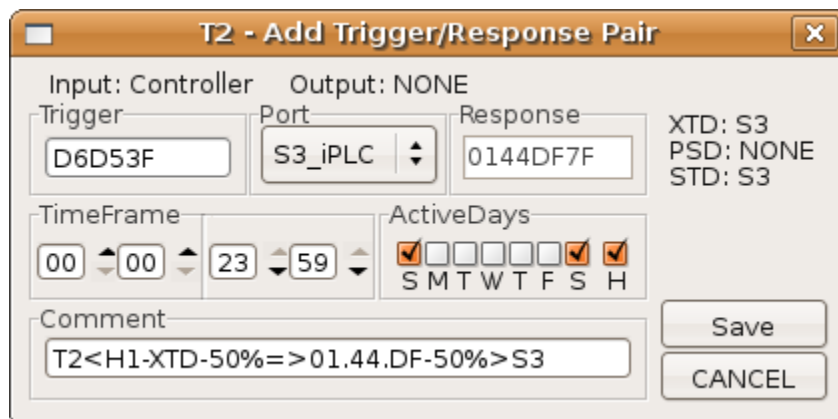
Select --- to create a **Null Response** in order to override an automatic **Response** for the specific **Trigger** code. A frequent use for this is to prevent secondary RF codes from X-10 motion detectors from being transmitted to the power-line. The input port is an implied part of the **Trigger**. This allows different **Responses** to the same **Trigger**, depending on the port.

Figure 21: Generating a Response

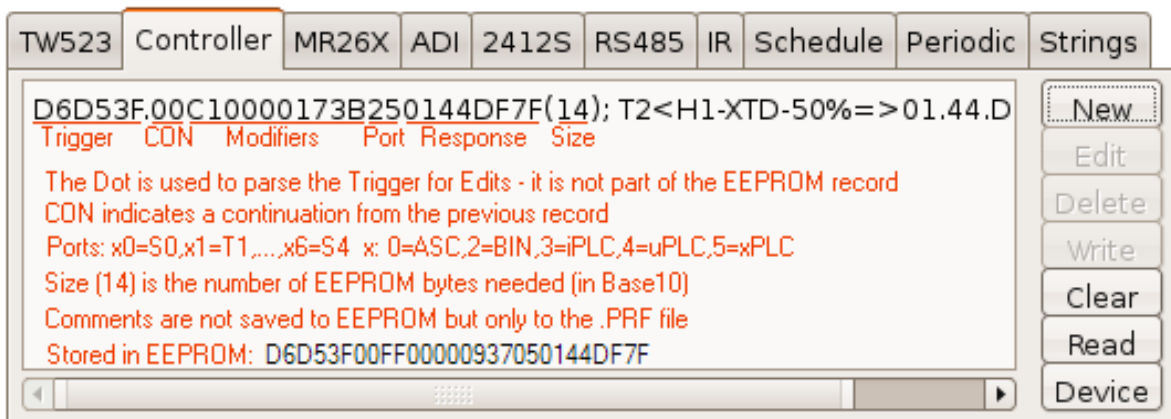


In the example, select a device from the drop-down list-box, set a level (for non-dimming devices use 0 or 100), and click the **OK** button to paste the code to the **Response** box in the **Trigger/Response** dialog. Details will vary with the port and device to which it connects.

Figure 22: Trigger/Response Dialog #3



Each **Trigger/Response** entry has modifiers for the timeframe and days of the week (plus holidays) which determine when it is enabled. Users can add comments (which are not stored in EEPROM) to help understand the activity defined. Clicking the **Save** button adds the newly defined **Trigger/Response** to the list for the currently selected device.



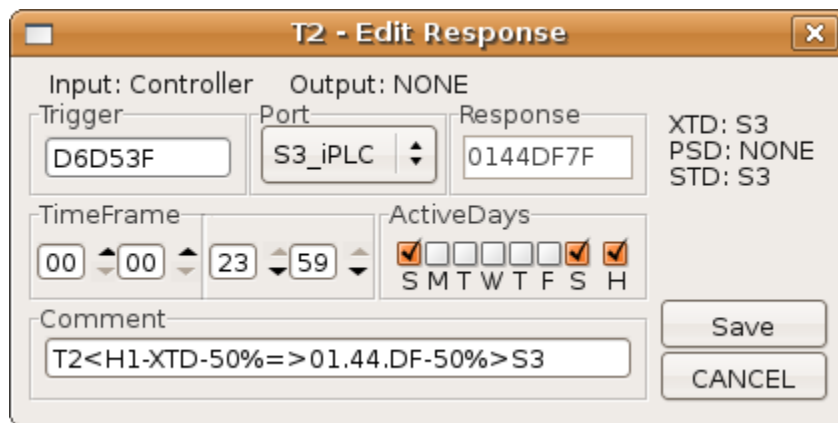
Rather than manually entering the **Trigger** code, it can be copied from the **Input/Output** window and pasted (right-click for menu) into the **Trigger** box. This works whether the code in the **Input/Output** window was input or was simulated with a **TX-RX** dialog. If the same **Trigger** code is used more than once, either with different **Response** codes or different modifiers, the sequence is controlled by entering the items in the desired order of execution. If the new entry has the same data as an existing entry, a pop-up dialog will indicate this and the new entry will not be added to the list. In this case, change the data as needed and click the **Save** button again.

The same routine handles all of the input ports so it must be able to deal with **Triggers** of different lengths, perhaps even from the same device since **roZetta™** supports **ASCII** and **Binary** devices which may have variable length codes.

As simple as the preceding example is, it can be even simpler. **XTD**, **PSD** and **STD**, above the **Save** button, show the current port's default assignments for **Extended Dim**, **Preset Dim** and **Standard X-10** codes. When any of these codes are received on the port they are automatically sent to the designated output port where they will be translated into the protocol for that port. In the case of **Extended Dim** and **Preset Dim codes**, if the code's X-10 address has been assigned to an Insteon address, it will be sent as an Insteon command. If no assignment has been made, it will be sent as an X-10 command. It works the same way for UPB except that the code will be ignored if there has been no address assignment as the **UPB PIM** cannot send/receive X-10.

Editing of an existing **Trigger/Response** entry is identical except that the **Trigger** code cannot be changed.

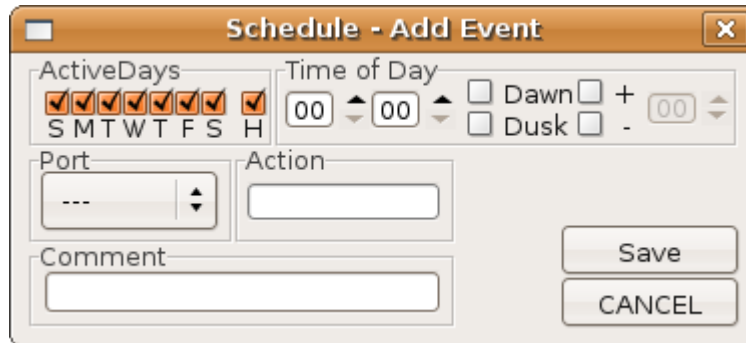
Figure 23: Edit Response Dialog



After editing, click the **Save** button and the data will overwrite the original.

Entering and Editing Scheduled Events

Figure 24: Add Scheduled Event Dialog #1



Data for **Scheduled Events** are entered in the same manner as for **Trigger/Response** pairs. Some differences being that the trigger is **Time of Day** and that an output **Port** is not required for all actions. **Dawn** and **Dusk** can be offset by plus or minus 1-30 minutes. Both **Dawn**, **Dusk** and **Time Of Day** can be randomized by $\pm 1-30$ minutes if both **+** and **-** are checked. The offsets and randomizations require the optional RTC board.

Figure 25: Add Scheduled Event Dialog #2

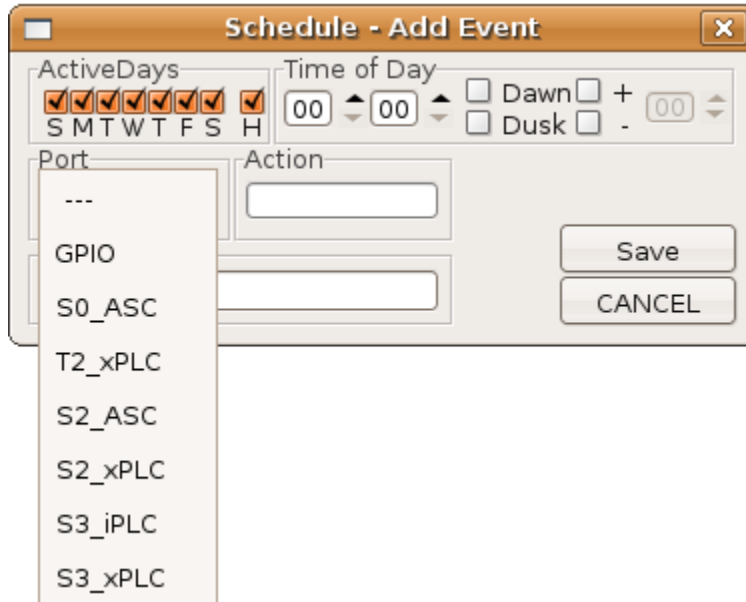


Figure 26: Edit Scheduled Event Dialog

Schedule - Edit Event

ActiveDays: S M T W T F S H

Time of Day: Dawn + Dusk -

Port: S3_xPLC

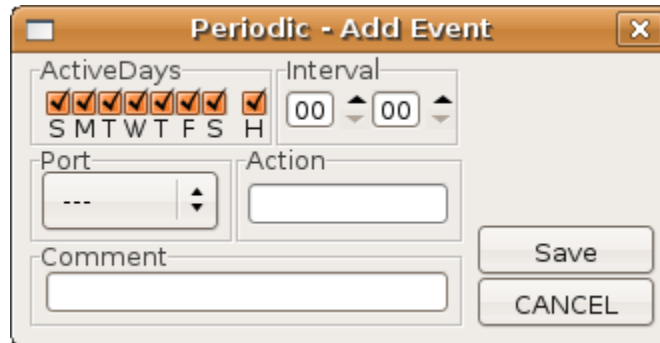
Action: D6D301

Comment: H1 Off @DAWN S3

Save CANCEL

Entering and Editing Periodic Events

Figure 27: Add Periodic Event Dialog #1



Data for **Periodic Events** are entered in the same manner as for **Scheduled Events**. The only difference being that the trigger is an **Interval** of time. Intervals are calculated from 00:00 for each day so intervals greater than 11:59 will only execute once daily. In effect, they will be like **Scheduled Events**.

Figure 28: Add Periodic Event Dialog #2

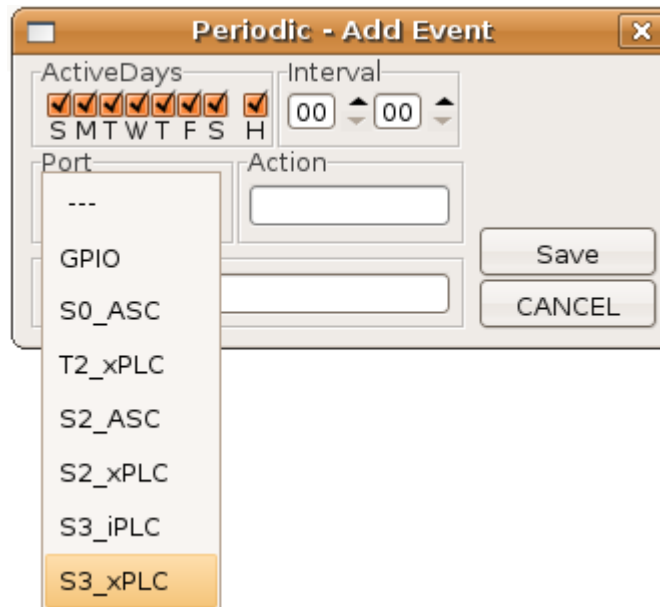
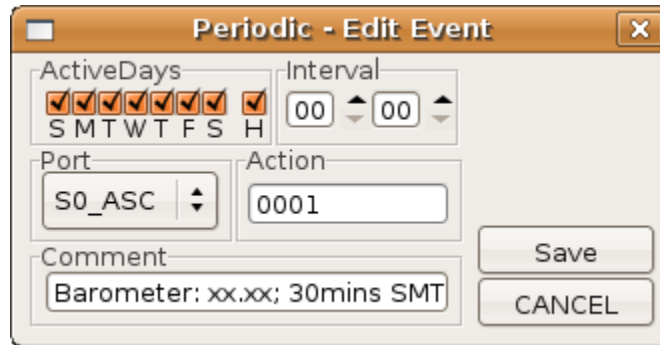
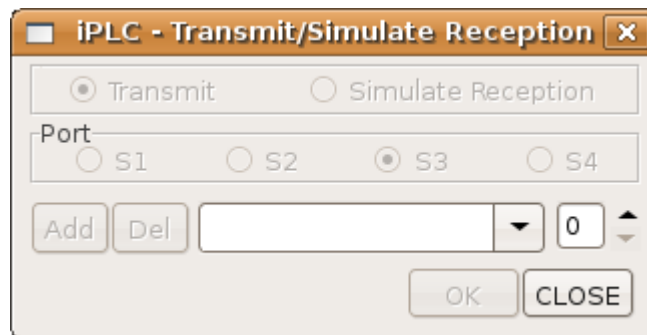


Figure 29: Edit Periodic Event Dialog



With both **Scheduled** and **Periodic** events, certain actions, such as reading a sensor or outputting a pulse-train do not require an output **Port**. I am still working on the details of how to allow users to enter this data while still validating it. In the meantime, the pop-up dialog below in Figure 24 allows for *unvalidated* data entry.

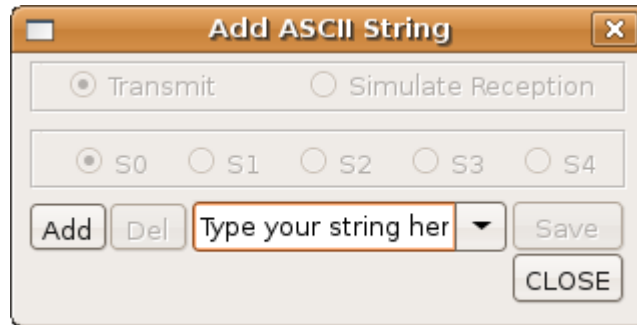
Figure 30: Action Data Dialog



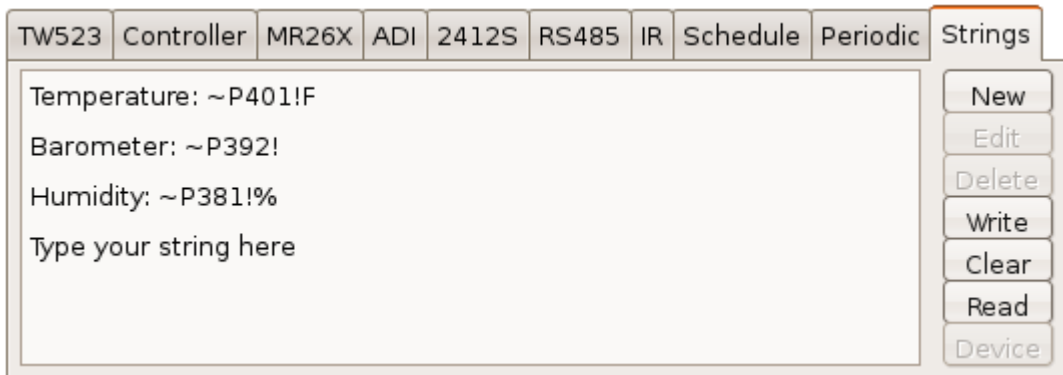
Entering and Editing ASCII Strings

Select the **Strings** tab and click the **New** button to launch the **Add ASCII String** dialog shown in Figure 30, below.

Figure 31: Adding/Editing ASCII Strings



Type in the new string and click the **Add** button to add it to the **Strings** list.



To edit an existing string, select it and click the **Edit** button which will launch the same window shown in Figure 30. Make the desired changes and click the **Save** button to overwrite the original with the edited string.

To delete an existing string, select it and click the **Delete** button. It will be removed from the list.

To delete all **Strings**, click the **Clear** button. If you answer the *Are You Sure?* prompt affirmatively, the list will be cleared.

To conserve EEPROM, strings are stored contiguously with the first byte of each indicating its length in bytes. Strings are accessed using the address of the first byte. This requires an index table to track the addresses which has to be rewritten whenever there are additions, deletions or edits that alter the length of any string.

The maximum number of strings is 4096 but, as each string requires a two byte address in the index table, the actual number of strings will be much lower unless the optional RTC board with the 64KB EEPROM is installed.

Entering and Editing PulseTrains

If any of the GPIO pins 38-40 are configured as digital outputs and if the optional RTC daughterboard with the 64KB EEPROM is installed, pulse-trains representing IR and RF codes for controlling A/V gear, camera mounts, ceiling fans, etc. can be stored in the EEPROM for playback as responses to triggers or use in scheduled events. The codes are compressed to minimize space and are stored in similar fashion to strings with an index that includes the address of the code itself. These are stored at the bottom of the optional EEPROM. Further details will be forthcoming when the RTC board is released.

Figure 31: PulseTrain Dialog #1

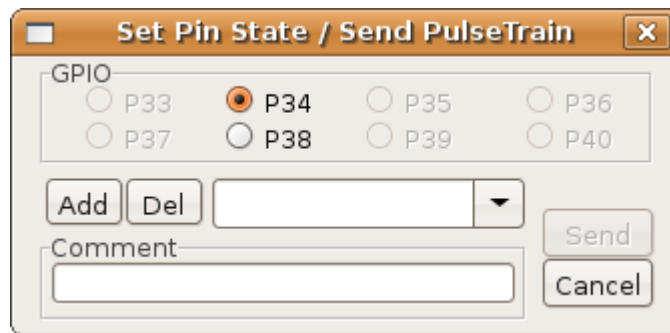


Figure 32: PulseTrain Dialog #2

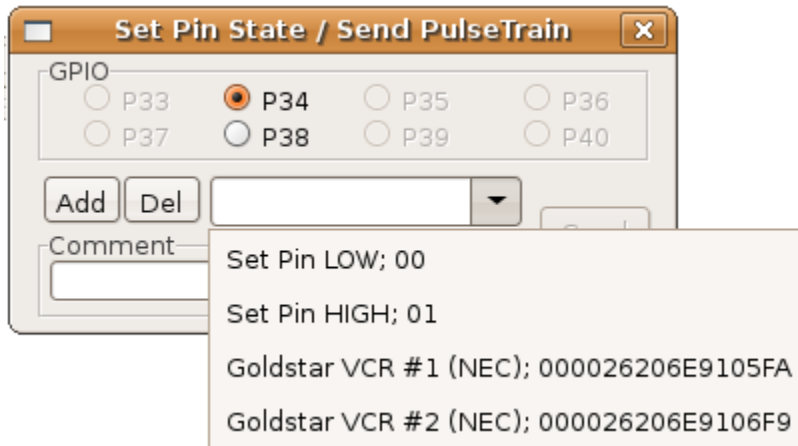


Figure 33: PulseTrain Dialog #3

Set Pin State / Send PulseTrain [X]

GPIO

P33 P34 P35 P36
 P37 P38 P39 P40

Add Del 000026206E9105F/ [v]

Comment

Goldstar VCR #1 (NEC)

Send Cancel

Preference Files

Each of the ten tabs is associated with its own preference file (*.prf) stored in the same folder as the **roZetta**[™] executable. These are ascii text files organized along the same principles as Windows .ini files. They are created and updated automatically as entries/changes are made to the associated tab list boxes. These files have a single section [items] and one key, named *count*, which tracks the number of entries in the specific file. The keys for the individual entries begin at 0 and increase linearly to *count-1*. The numeric keys represent line numbers in the list boxes and determine the order of display as well as the order in EEPROM storage which, in turn, determines execution order.

If there is a need to edit any of these files, they should be backed up first. One reason for such edits might be to alter execution sequences for a group of entries having the same trigger which can be done by swapping their keys. Any manual edits will also need to be saved to EEPROM using the associated **Write** buttons. Editing any entry using the **Edit** button, even merely adding and deleting a space at the end of a comment, will enable the **Write** button.

In addition, the user.prf file stores data that will usually vary from user to user. An example is shown in **Appendix D**.

Chapter 6: Supported Devices

Defined Devices

T1: TTL Port

1132B – in TW523 mode

PL513, PSC04 – transmit only

TW523, PSC05, TW7223, XM10 – transmit/receive

T2: TTL Port

Legacy Controllers – anything designed to send/receive X-10 using one of the OEM interfaces (1132B,PL513,TW523,PSC04,PSC05,TW7223,XM10) should work. **roZetta**[™] emulates the OEM interface. Controllers can only connect to T2 and require 1132B, PL513, TW523, PSC04, PSC05, TW7223 or XM10 on T1 to supply ZC or the optional plug-in RTC daughterboard with switch **SW4** set to supply a 64Hz ZC proxy.

1132B – in TW523 mode

PL513,PSC04 – transmit only

TW523,PSC05,TW7223,XM10 – transmit/receive

S1-S4: Low-Speed (9600 max) RS232 Ports

1132B – in RS232 mode

2412S, 2414S

ADI (CPU-XA, Ocelot, Leopard)

CM11A and European equivalents

CM15X-S – CM15A converted to a PIC16F88 and RS232

LCD+/LCD+2 – from NetMedia – has eight 10-bit ADC or DIO channels. Any serial LCD should work when defined as an ASCII device. I plan an adapter to use them via RS485 and will explore adding a PIC to CircuitEd's small touchscreen to make it a serial device.

MR26A/E – standard American (310MHz) and European (433.92MHz) devices. SW1-SW3 supply +5V power to S1-S3 for powering these. S4 also has +5V available.

MR26X – MR26 with replacement PIC – handles standard & security X-10, CR14A/E, Digimax 210 (European wireless thermostat) protocols. SW1-SW3 supply +5V power to S1-S3 for powering these. S4 also has +5V available.

UPB PIM - all UPB serial interfaces use the same protocol.

X10RF – RS232/RS485 equivalent to MR26X. SW1-SW3 supply +5V power to S1-S3 for powering these. S4 also has +5V available.

Other – several PIC-based DIY RF receiver designs of various frequencies for various security devices, report via RS232 or RS485

W0: Wireless Port

User installed RF receiver (310, 418, 433MHz) for various X-10 RF protocols or user installed wideband IR receiver (TSOP1100 32-67kHz) which, initially, will support X-10 IR protocol and extended IR protocol developed by Laser Systems in the UK - may later add support for other popular IR protocols.

X-10 uses 310MHz in N. America and 433.92MHz elsewhere. Most programmable RF-capable remotes like the Pronto and MX3000 use 418MHz in N. America and 433.92MHz elsewhere. Inexpensive superregenerative RF receiver modules are readily available worldwide in frequencies of 315, 418 and 433.92MHz (and other frequencies) and all are tunable over a fairly wide range so it's easy to tune the 315MHz version to 310MHz or to tune it or others to the frequencies used by security systems in N. America. By mixing and matching frequencies of the onboard wireless port and various RS232/RS485 modules, it is possible to input X-10 and other RF codes from a variety of remote devices.

Undefined Devices

ASCII – with [embedded] binary codes

BINARY – simple protocols with limited handshaking

ADC and DIO Pins

When configured for ADC, pins 33-40 can be used with a variety of sensors for measurement of temperature, pressure, humidity, etc. There are some example circuits and links to sensor datasheets on my website.

When configured as digital inputs, pins 33-40 can be used with devices that provide 0V and +5V to denote two possible states.

When configured as digital outputs, pins 33-40 can be used to drive LEDs, IR emitters and RF transmitters. **roZetta**[™] can send several popular IR protocols as well as numerous RF protocols for controlling ceiling fans, etc. There are some example circuits on my website. The structure of most IR and RF protocols can be defined in 20 or fewer bytes. Once the structure is defined, the codes themselves can be defined in 2-4 bytes. At this time, no decision has been made about where to store structures and codes but an EEPROM block between **Strings** and **Holidays** is most likely. This will allow users to define only those protocols they need in order to minimize storage requirements.

Since the pins can be configured individually for any of the three modes, it is possible to connect a mix of sensors, indicators and transmitters. If more ADC or DIO channels are desired, the NetMedia LCD+ and LCD+2 have 8 channels which can be defined in a similar manner to pins 33-40. If either TTL port is not used for X-10, some of those pins can be used for DIO. There are various serial devices that aggregate multiple ADC or DIO channels.

Take care not to overload **roZetta**[™]. Each GPIO pin has a maximum source/sink current limit of 40ma but the 8-pin port defined by pins 33-40 has a total current limit of 100mA. The +5V bus can supply power to external devices as long as the current required for any one device does not exceed about 25mA and the total is less than 100mA. **RoZetta**[™] should be protected by an inline fuse (close to the **roZetta**[™] connection) in the line supplying the external device(s). There is a resettable fuse in the +5V line to the GPIO pins that will pass 80mA and trip at ~320mA.

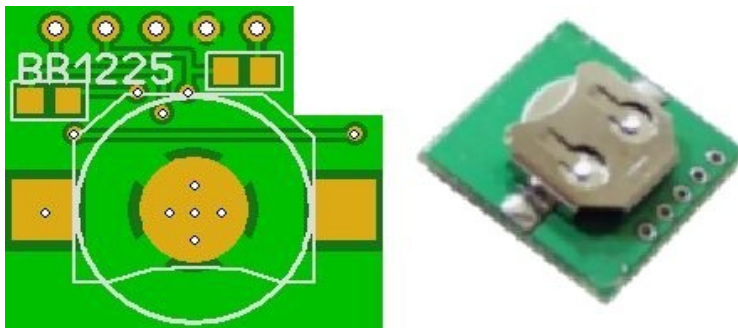
Chapter 6: Hardware Options

Network Interface: Tibbo EM202 Serial-to-Ethernet Converter



If the Tibbo EM202 is used, S0 is disabled. Initially, the Tibbo module will just be used as a serial server and will employ Tibbo's virtual serial port drivers. Later, I'll try to write code to enhance its role, doing e-mail, etc. The EM202 firmware can be upgraded using the ethernet or serial link. In the event of problems, there are jumpers that can be used to make a direct serial connection to the module. Referring to the picture of the printed circuit board in **Appendix A**, removing the three jumpers from the **Network/RS232** 2x6 header and installing them on the 2x3 header as shown in the graphic labeled **/MD** connects the module to S0 in direct serial mode. For normal network operation, remove all three jumpers as shown in the **MD** graphic and install them as shown in the **Network** graphic.

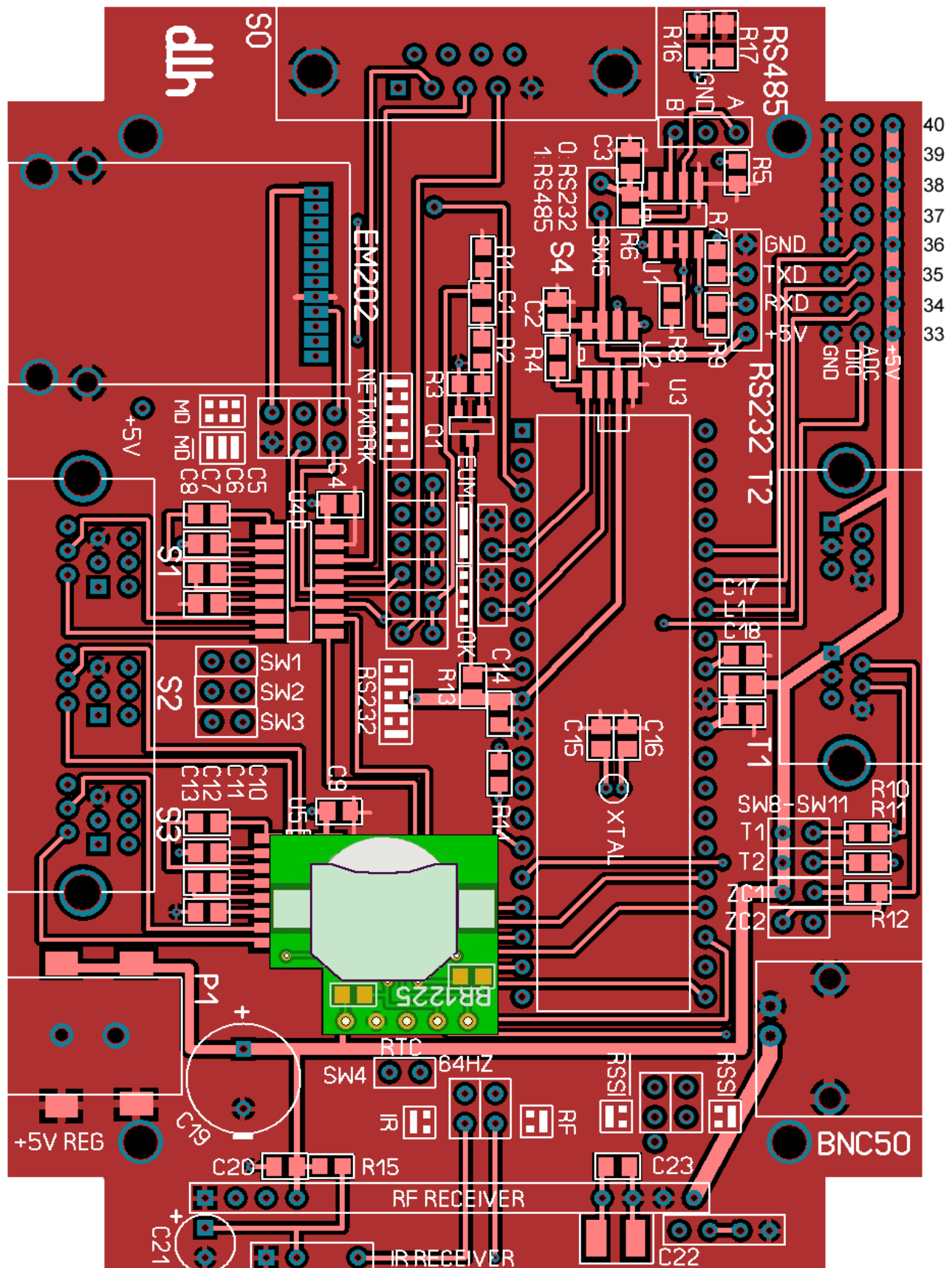
Battery-backed RTC



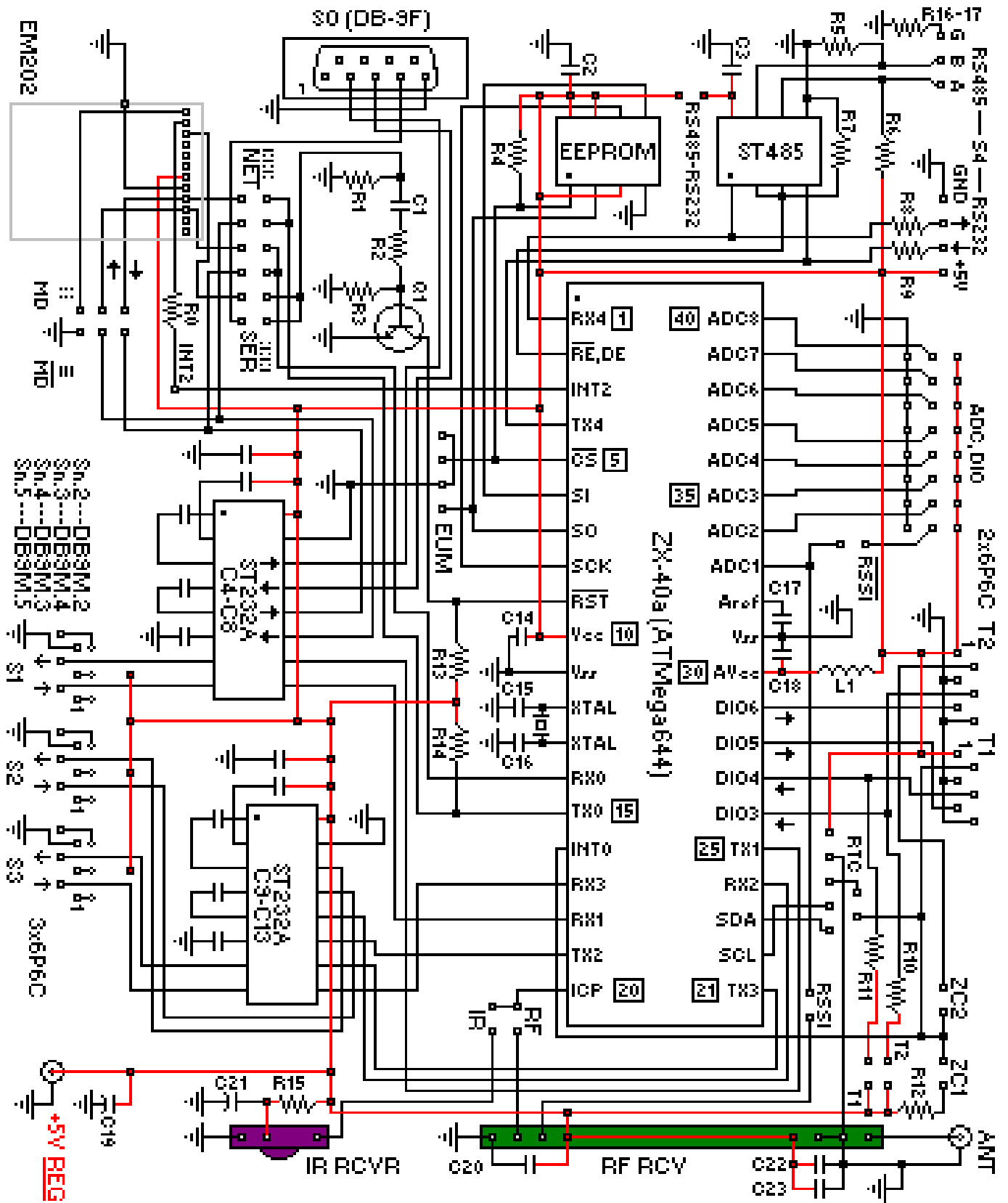
The plug-in RTC daughterboard uses the ST M41T81 chip with options for either 8KB FRAM or 64KB EEPROM. The BR1225 battery has more than a ten year life even if called on to power the M41T81 full time. I²C bus speed is only 400kHz while the SPI bus used for the main EEPROM runs at 7.37MHz. FRAM handles billions of write/erase cycles so can be treated almost like RAM and can be used to track status of all controlled devices, randomize events, etc.

NOTE: The picture above shows the prototype on the right. The final design is as shown in the PCB layout on the left.

Appendix B: roZetta™ Printed Circuit Board + RTC Board



Appendix C: Schematic Diagram



Appendix D: Example *user.prf* File

```
[user]
months = Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
xtd = iPLC
psd = uPLC
;wait (ms) for ACK/NAK when downloading firmware
wait = 250
;ports: list of serial ports in system - find ports with "$ dmesg | fgrep tty"
ports = ttyS0, ttyUSB0, ttyVSP0
;
[xPLC_addr2name]
H1 = MBR
H2 = BATH-MBR
H3 = OFFICE
H4 = BATH-MAIN
H5 = KITCHEN
H6 = DR
H7 = PANTRY
H8 = LR
;
[xPLC_name2addr]
MBR = H1
BATH-MBR = H2
OFFICE = H3
BATH-MAIN = H4
KITCHEN = H5
PANTRY = H6
LR = H7
DR = H8
;
[iPLC]
DEVICES = 6
0 = 0144DF[H1]MBR
1 = 0498A7[H3]OFFICE
2 = 05999F[H4]BATH-MAIN
3 = 05A8B9[H2]BATH-MBR
4 = 05A90B[H5]KITCHEN
5 = 07ACD2[H7]LR
;
[iPLC_addr2Name]
0144DF = MBR
05A8B9 = BATH-MBR
0498A7 = OFFICE
05999F = BATH-MAIN
05A90B = KITCHEN
07ACD2 = LR
;
[iPLC_name2addr]
MBR = 0144DF
BATH-MBR = 05A8B9
OFFICE = 0498A7
BATH-MAIN = 05999F
KITCHEN = 05A90B
LR = 07ACD2
DR =
PANTRY =
;
[uPLC]
DEVICES =
;
[uPLC_addr2name]
;
[uPLC_name2addr]
```

Appendix E: Example Log File (at Boot)

At boot, **roZetta**™ outputs information about its current ports configurations as well as information about the current CPU, Virtual Machine version, **roZetta**™ firmware size, **roZetta**™ firmware compile date, EEPROM size and reason for reset. If the optional RTC board is present, the time power was lost and restored is logged along with information about the RTC board's EEPROM. If the RTC board is not present, **roZetta**™ asks the PC for a time-stamp to reset its software RTC. An example is shown below.

```
01/11/2007 09:15:14 roZetta (tm)
01/11/2007 09:15:14 Copyright © 2006, 2007 D.L.Houston
01/11/2007 09:15:14 All Rights Reserved
01/11/2007 09:15:14 Compile Date: Jan 04 2007 11:51:45
01/11/2007 09:15:14 EEPROM:65536 – CODE:9827
01/11/2007 09:15:14 CPU: ZX40a - VM 1.5.4
01/11/2007 09:15:14 Power-On + Brownout Reset
01/11/2007 09:15:15 S0 OPEN @ 19200 8N1
01/11/2007 09:15:15 S1 OPEN @ 4800 8N1 [2414S]
01/11/2007 09:15:15 S2 OPEN @ 4800 8N1 [UPB-PIM]
01/11/2007 09:15:15 S3 OPEN @ 9600 8N1 [MR26A]
01/11/2007 09:15:15 S4 OPEN @ 9600 8N1 [RS485] Inverted
01/11/2007 09:15:15 T1: TW523
01/11/2007 09:15:15 T2: CTL No pull-up
01/11/2007 09:15:15 W0: IR243
01/11/2007 09:15:15 ?RTC
01/11/2007 09:15:15 S0<-1052D707010B090F0F920600E21C00010A
01/11/2007 09:15:15 RTC: 11 JAN 2007 09:15:15 DST: 01,1682:00,7394:00,273:15
01/11/2007 09:15:15 NOW:0915 DAY:11 SUNRISE:0757 SUNSET:1736 HOLIDAY:False
```

