THAT Home Automation Topology™ (THAT)

Final Report
2010/05/14

Project By
Nick Viera
Chris Miller

Advised By
Dr. James Irwin
Dr. Aleksander Malinowski

Bradley University
Electrical and Computer
Engineering Department

Summary

THAT Home Automation Topology™ (also known as "THAT" and "THAT System") describes a modern, Ethernet and Internet-based home automation and control system. THAT System consists of specifications, hardware, and software which work together to provide a comprehensive feature set. THAT System is designed specifically with modularity, affordability, and the open-source philosophy in mind.

The goal of home automation is to provide users with systemic, intelligent control over their home/building environment. Inexpensive, simplistic, and functionally-limited products are already available on the market as well as expensive, complex, and functionally-chaotic devices. The goal of THAT is to provide an option that combines the best of these two extremes.

Abstract

THAT Home Automation Topology (also known as "THAT") describes a modern, Ethernet and Internet-based home automation and control system. THAT consists of specifications, hardware, and software which work together to provide a comprehensive feature set. Included are a network interface card design, and a power supply design capable of using a power over Ethernet (PoE) or an auxiliary AC/DC input. Additionally, a programmable digital thermostat module and a programmable electronic access module based on 8-bit Atmel AVR microcontrollers have been designed. Schematics, PCB layouts, parts lists, firmware (AVR C), and PC control software based on Python and XML are included.

Table of Contents

1. Project Introduction

The scope of this project involved the creation and definition of THAT System, as well as the design and construction of two THAT-compatible modules. The work was completed during the 2009-2010 academic year as Chris Miller and Nick Viera's Capstone Project in Electrical Engineering at Bradley University.

Nick Viera and Chris Miller co-developed the hardware and software frameworks which define THAT System. In addition, Nick Viera developed a THAT-compatible “Digital Thermostat Module.” Chris Miller developed a THAT-compatible “Electronic Access Module.”

1.1. Significance

Home Automation can benefit users by providing convenience, safety, savings, and fun.

1.1.1. Problem Statement

The basic technology needed to facilitate advanced home automation functionality has been available for at least the last decade. However, home automation has not yet enjoyed mass adoption in the marketplace.

Inexpensive, simplistic, and functionally-limited products are already available on the market as well as expensive, complex, and functionally-chaotic devices. The goal of THAT is to provide an option that combines the best of these two extremes, without the pitfalls of either, in an open-source framework. In doing so, the developers hope to make home automation ubiquitous.

1.2. Development Tools

1.2.1. Hardware

The two main hardware devices used in the development of THAT were microcontroller programmers. One was an STK500 development board manufactured by Atmel. The other was an AVRUSB500v2, an open-source in-system programmer kit by Guido Socher of www.tuxgraphics.org.

1.2.2. Software

The EAGLE layout editor by Cadsoft (http://www.cadsoft.de/) was used by both developers to draw the schematics and corresponding Printed Circuit Board (PCB) layouts.

The Python programming language and the Python interpreter were used to develop Master Control Software for THAT System (see Appendix A).

The open-source GNU C Complier (http://gcc.gnu.org/) and AVR Libc libraries (http://www.nongnu.org/avr-libc/) were used to compile the firmware for the microcontrollers. Once compiled, the microcontrollers were programmed using the open-source AVRDUDE (http://savannah.nongnu.org/projects/avrdude/).

2. THAT Home Automation Topology

2.1. Modules and System Interaction

THAT System is defined to be used in (but not limited to) two basic system configurations: independent and integrated. The basic hardware topology of a THAT System is shown in Figure 7-1.

The independent configuration consists of one or more THAT modules operating as “stand-alone” devices. The modules may optionally communicate peer-to-peer.

The integrated configuration consists of one or more modules operating with a computer. The computer is the master arbitrator of the system in this mode. It uses information from some modules to dictate the behavior of others. The computer runs THAT Control Software (see Section 8). Should the master computer fail or otherwise disappear from the network, the integrated system would revert back to an independent system.

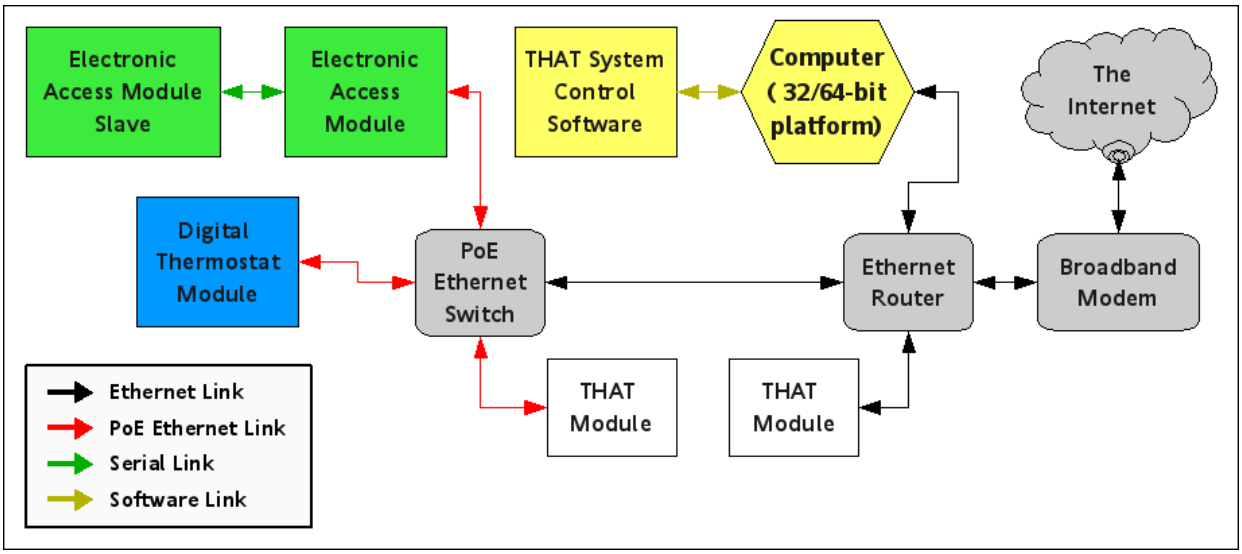

Figure 7-1 : Basic THAT hardware topology

THAT System can be expanded to support a wide variety of sensor and control functions through the creation of additional modules and through the use of custom and 3rd party software. The functionality of THAT is limited mainly by the variety of available modules and not by design exclusions. The developers encourage further construction / expansion of modules, and of THAT System as a whole, by third-parties. Some modules which the designers would ultimately like to see incorporated into THAT are shown in Figure 8-1.

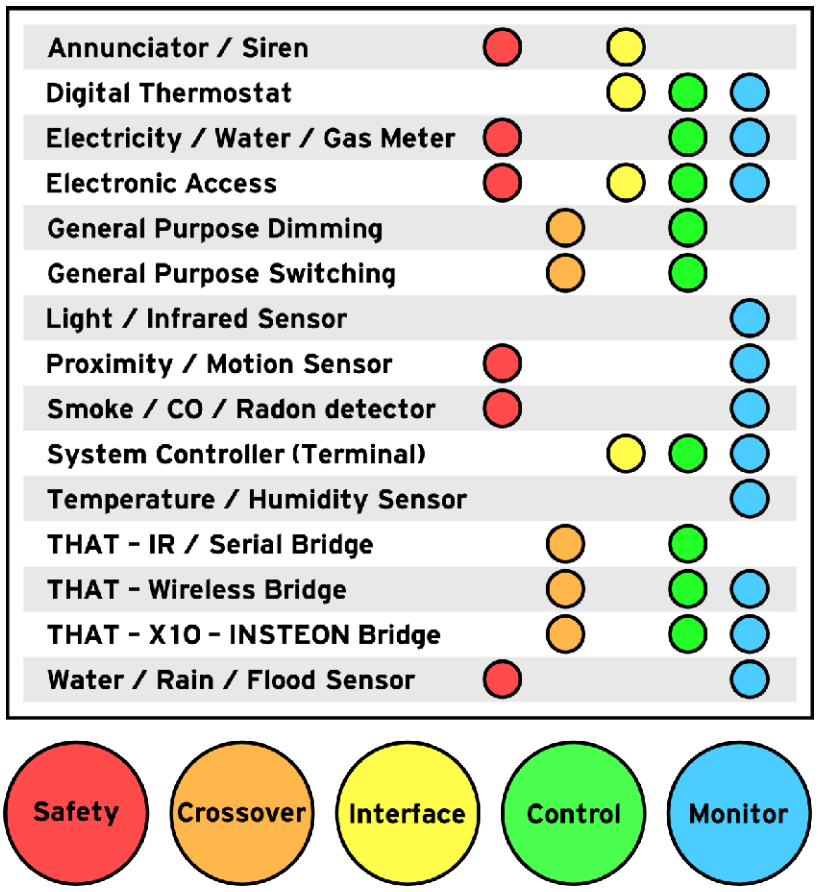

Figure 8-1: Possible Modules for THAT System

2.2. Overall Design Goals

- Modularity
- Standardization of hardware
- Standardization of communication
- Form follows function
- Design integrity takes precedence over design cost
- Open Source software and hardware for most functionality
- “Freemium*” philosophy for advanced functionality

* Freemium is a business model that works by offering basic services or products for free while charging a premium for advanced or special features. The word "freemium" is a portmanteau created by combining the two aspects of the business model: "free" and "premium". The business model has gained popularity with Web 2.0 companies. [wikipedia.org]

2.3. Hardware Requirements

2.3.1. Link Protocol

The primary link protocol specified for THAT System is 10BASE-T Ethernet per the IEEE 802.3 standard, or any backwards-compatible successor.

Ethernet was chosen as a link protocol because its components, cabling, and infrastructure are mass-deployed and technologically mature. Specifically, Ethernet is ideal for use with THAT System because it provides the following:

- A high-speed, electrically-isolated data link
- Support for reasonably-long cable runs of up to 300m
- A star-topology allowing a high degree of flexibility in the system layout
- Support for power and data transmission within the same cabling

2.3.2. Data Connection and Cabling

The data port connector specified for use with THAT modules is the standard 8P8C modular jack widely used with Ethernet, using the RJ-45 configuration. Data cabling should be twisted-pair CAT5 or better, per the IEEE 802.3 standard.

2.3.3. Primary Power Supply

Power over Ethernet (PoE) per the IEEE 802.3af standard will be used as the primary power source for all THAT modules. PoE is an ideal power distribution scheme for THAT because one power supply can efficiently provide the (relatively low) power levels needed to run many modules. In addition, it can do so while utilizing existing cabling and components. Finally, PoE allows for easy backup power solutions since all PoE devices can be powered from a single location.

2.3.4. Secondary Power Supply

All modules will be required to accept at least a 6-26 Volt AC or 9-36 Volt DC input from a “point-of-use” or secondary power supply. The secondary power connection should be made through a 5.5mm outer diameter (2.1mm inner diameter) barrel jack. This jack size was chosen because it is a widely available and commonly used power connector.

Should both primary and secondary power supplies be active, the PoE supply will take precedence over, and disable, the sourcing of power from the secondary AC/DC power supply. THAT modules functioning as safety devices or which serve critical roles should also contain their own local battery backup solutions. In these cases, the local backup power must be available to the module at all times.

2.4. Communication Requirements

2.4.1. Data Transport Protocol

The data transport protocol required for use by all THAT modules is the Transmission Control Protocol with the Internet Protocol, version 4 (TCP/IP). TCP was selected over other protocols such as UDP due to the higher reliability of its transmissions. Reliability is important because lost packets or other transmission errors could create an inconvenience for the user and more overhead for the control software and modules.

2.4.2. Data Access

All THAT modules must default to being servers on the network, listening on TCP port 8428. Modules are required to be servers because servers listen on the network at all times and can thus be contacted arbitrarily by multiple clients. Such functionality makes it possible for multiple clients to communicate with a module simultaneously.

Port 8428 was selected as the default software port for THAT because it corresponds to the letters 'T' 'H' 'A' 'T' as entered on an alpha-numeric telephone keypad. Additionally, port 8428 is currently not known to be widely used by any other devices or services.

Additionally, THAT modules are allowed to function as TCP clients and/or use other ports on the network. However, this behavior should not be the default mode of operation, and should not replace the module's ability to function as a TCP server on port 8428.

2.4.3. Standard Command Set

The THAT Command Set, as defined in Appendix B, is used for reading data from and writing data to THAT modules. The command set is used by the master control software to communicate with modules, as well as for modules to (optionally) communicate with each other. All modules are required to support the standard command set. Optionally, they may implement any additional / proprietary commands as needed.

3. THAT Power Supply

3.1. Introduction

One of the fundamental components that every THAT module will need is a PoE power supply. Although not required, it is likely that most THAT modules will make use of identical or very similar power supplies. The developers' power supply design is presented here. The power supply has two input sources (PoE and an external AC/DC input) and two outputs (5 and 3.3 Volts DC).

3.1.1. Block Diagram

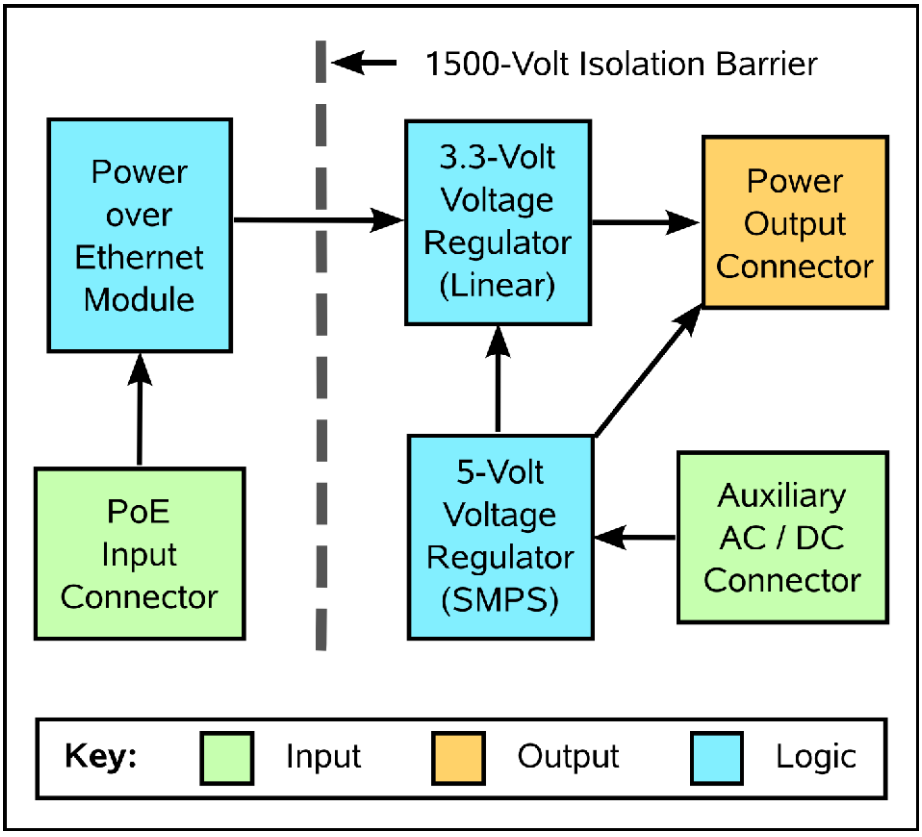

Figure 12-1: THAT Power Supply block diagram

3.1.2. Voltage Regulation

To handle the PoE functionality, as per the IEEE 802.3af standard, the developers selected the Ag9400-S PoE module from the manufacturer Silver. This module handles the "handshake" procedure for identifying and obtaining power from a PoE powering device. It outputs regulated 5 Volts DC from a nominal 48 Volt PoE source.

To generate regulated 5 Volts DC from the auxiliary AC/DC input, the LM2575 IC was selected. This device uses two external components to create a high-efficiency (75–85% efficient) buck converter, which is a type of switch-mode power supply (SMPS). The use of a SMPS was necessary to minimize power dissipation, which would have been significant had a linear regulator been used in the presence of such high input voltages (up to 42 Volts DC).

From the 5 Volt DC source listed above, the developers then use an MC33269 IC, which is a linear voltage regulator, to provide the 3.3 Volt DC output. A linear regulator was considered acceptable in this application since the regulator is only dropping 1.7 Volts, and its power dissipation in doing so is acceptably low.

Note that the components mentioned here represent the “best” choices available at the time of this writing, as determined by the developers. Other options exist.

3.2. Parts List

The parts list for THAT Power Supply is shown in Figure 14-1.

Parts List – THAT Power Supply 0.3 Nick Viera – 2009.12.28 – Updated: 2010.04.07							
			Temperature				
Part ID	Type	Attributes	Min.	Max	Package	Manufacturer Part	Qty
C1	Capacitor	0.33uF, 50V, film	-40	105	5mm grid	ECQ-V1H334JL	1
C2	Capacitor	10uF, 25V, electro.	-40	85	Round, 4mm	ECE-A1EKA100	1
C3,4	Capacitor	680uF, 10V, electro.	-40	105	8x15, 3.5 grid	EEU-FM1A681L	2
C5	Capacitor	120uF, 50V, electro.	-55	105	10mm, 5 grid	UPW1H121MPD6	1
X1,2	Connector	3-pin, 2.54mm pitch			2.54mm grid	PPPC031LFBN-RC	2
D1,7,8	Diode	60V, 2A, Schottky	-65	125	DO-41	SR206-TP	3
D2	Diode	TVS, 43V, bi, 600W	-55	175	DO-15	P6KE43CA	1
D3-6	Diode	1N4004, 400V, 1A	-55	175	DO-41	1N4004	4
D9	Diode	TVS, 68V, bi, 1500W	-55	175	DO-201AD	1.5KE68CA	1
F1	Fuse	500mA, Fast blow			5x20mm	0034.1513	1
F1	Fuse	Plastic, Black, Horiz.			5x20mm	0031.8211	1
IC1	IC	PoE supply, Isolated	-20	70	Rectangle	Ag9405-2BR	1
IC3	IC	Vreg, 4-40V, 1A, 5V	-40	125	TO-220-5	LM2575T-5G	1
IC4	IC	Vreg, 3.3V, 800mA	-40	125	TO-220	MC33269T-3.3G	1
L1	Inductor	330uH, 1.15A			SMD 12.5mm	SRR1260-331K	1
R1	Resistor	47k, 5%, carbon	-55	155	Axial	CF 1/4 47K 5% R	1
R2	Resistor	8.2k, 5%, carbon	-55	155	Axial	CF 1/4 8.2K 5% R	1

Figure 14-1: THAT Power Supply parts list

3.3. Schematic

The schematic for THAT Power Supply is shown in Figure 15-1. Some notes are listed:

- X1 is the AC/DC input jack which can accept 6-26 Volts AC or 9-36 Volts DC. R2 is used to increase the output voltage of the PoE module slightly. This is necessary since the diode D8 will drop about 0.3 Volts
- The addition of R2 boosts the output of the PoE module to approximately 5.3 Volts to compensate for the drop, resulting in a 5 Volt DC output
- Capacitors C3 and C4 should be low ESR capacitors suitable for use with a SMPS, such as high-quality electrolytic or metal film capacitors
- D2 is a bi-directional TVS diode with a peak power dissipation of 600W for 1ms
- D9 is a bi-directional TVS diode with a peak power dissipation of 1500W for 1ms
- D2 and D9 are used for surge protection, and could be omitted if surge protection is unnecessary

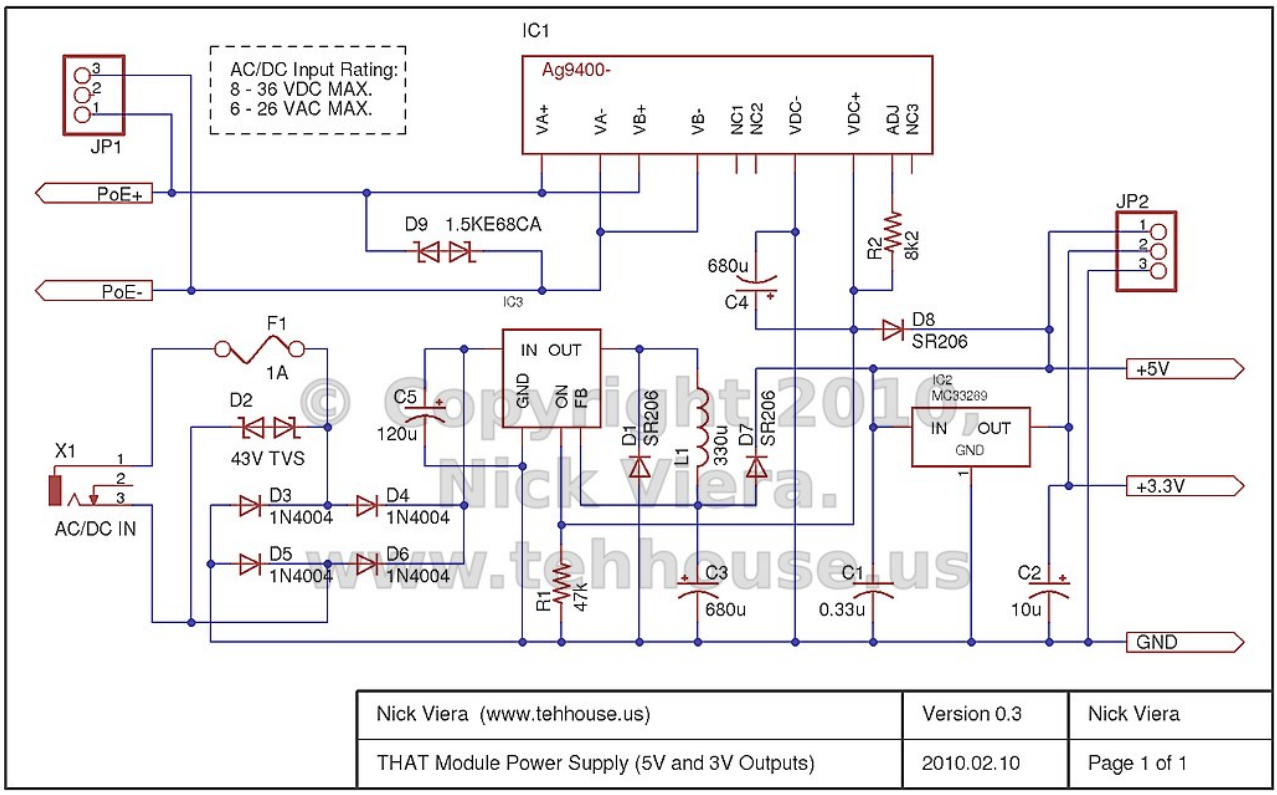

Figure 15-1: THAT Power Supply Schematic

3.4. PCB Layout

A Printed Circuit Board design for the Power Supply Board has been completed. The PCB is a standard, 2-layer board with through-hole components. It measures 83.5 mm x 52.2 mm. The PCB design layout is shown in Figure 16-1, followed by a photo of the PCB populated with components shown in Figure 16-2.

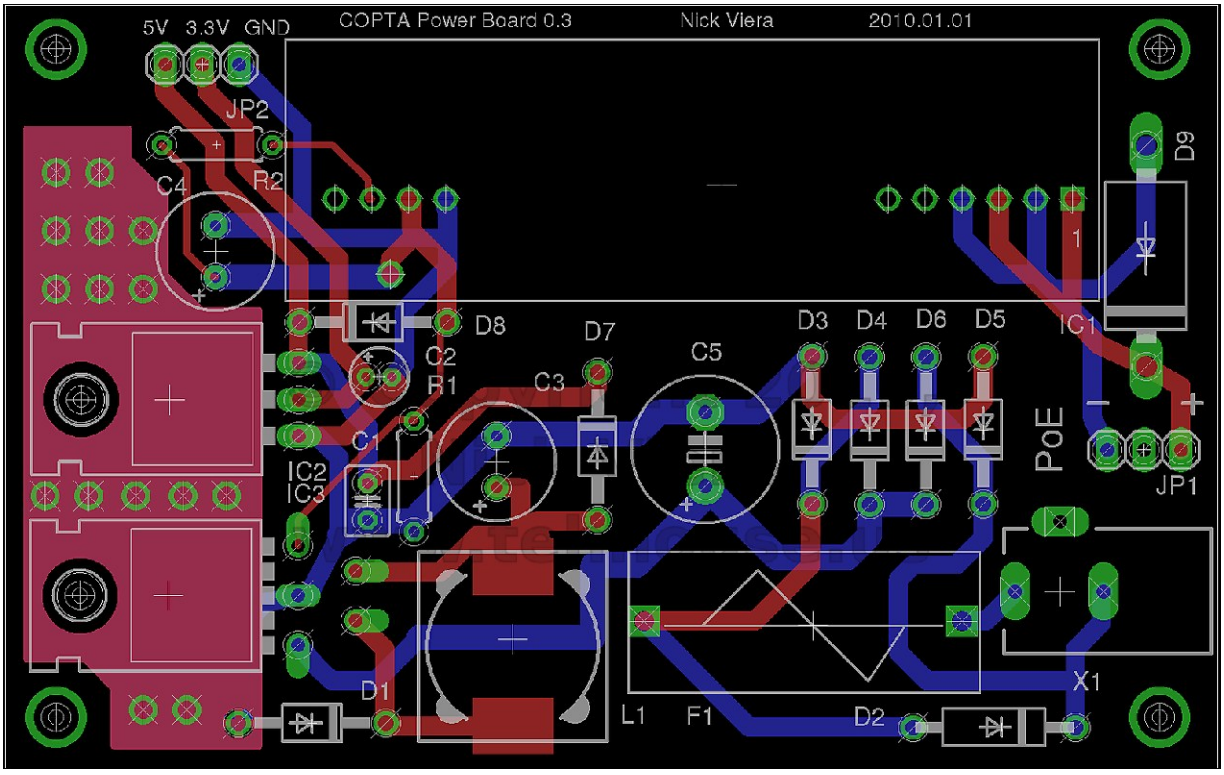

Figure 16-1: THAT Power Supply PCB Layout

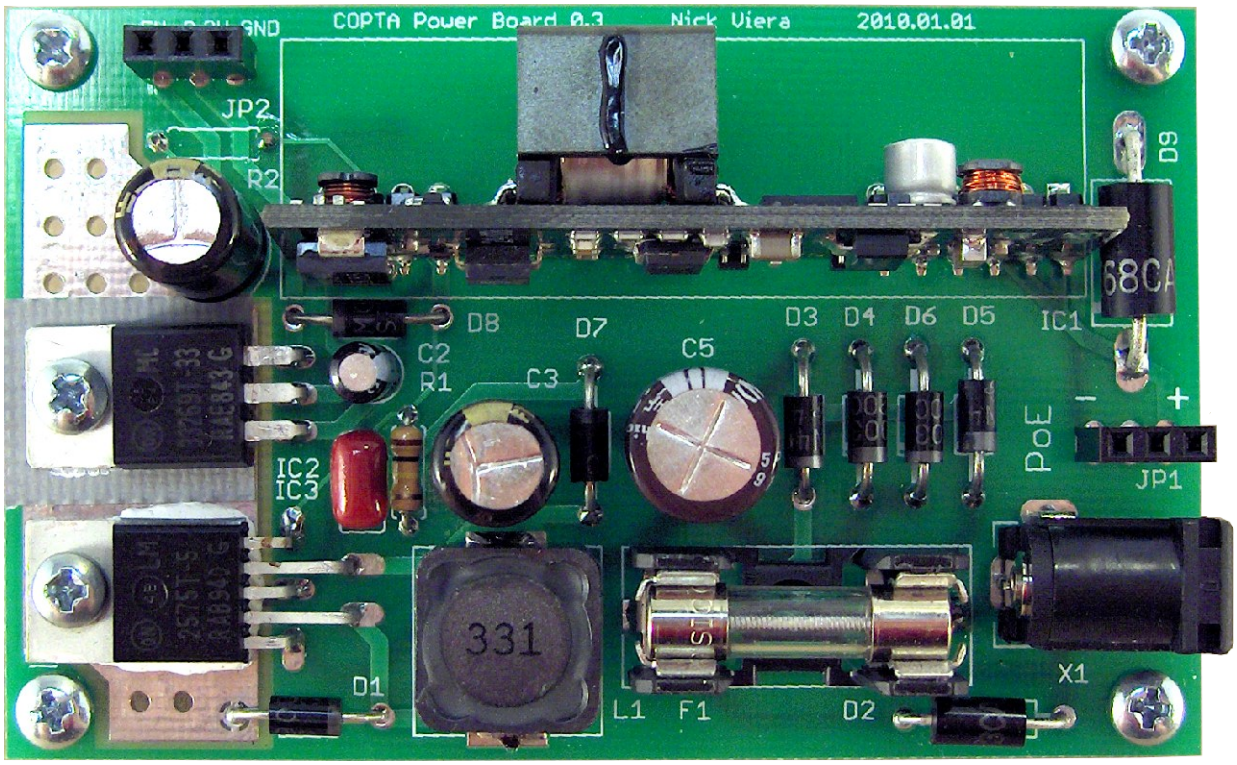

Figure 16-2: Complete THAT Power Supply

3.5. Connections

The Power Supply board contains two female pin header connectors (2.54mm pitch), in addition to a 5.5mm OD / 2.5mm ID barrel jack. The barrel jack is the AC/DC input from the secondary power supply.

One 3-pin header is for the Power Over Ethernet (PoE), and it connects to the PoE output header on the current THAT Ethernet Interface board. The other 3-pin header is the power output header, which carries 5 Volts, 3.3 Volts, and Ground. It is designed to connect to the mainboard of a THAT module.

4. THAT Network Interface

4.1. Introduction

An Ethernet controller (also known as a Network Interface or NIC) is a component which every THAT module must contain. It is likely that many THAT modules will make use of identical or very similar NIC designs. The developers' network interface design is presented here.

The main purpose of the NIC is to implement low-level network functions in hardware. Additionally, the NIC contains the physical connection port and all the magnetics (transformers and chokes) to complete the IEEE 802.3-compliant network interface.

4.1.1. Block Diagram

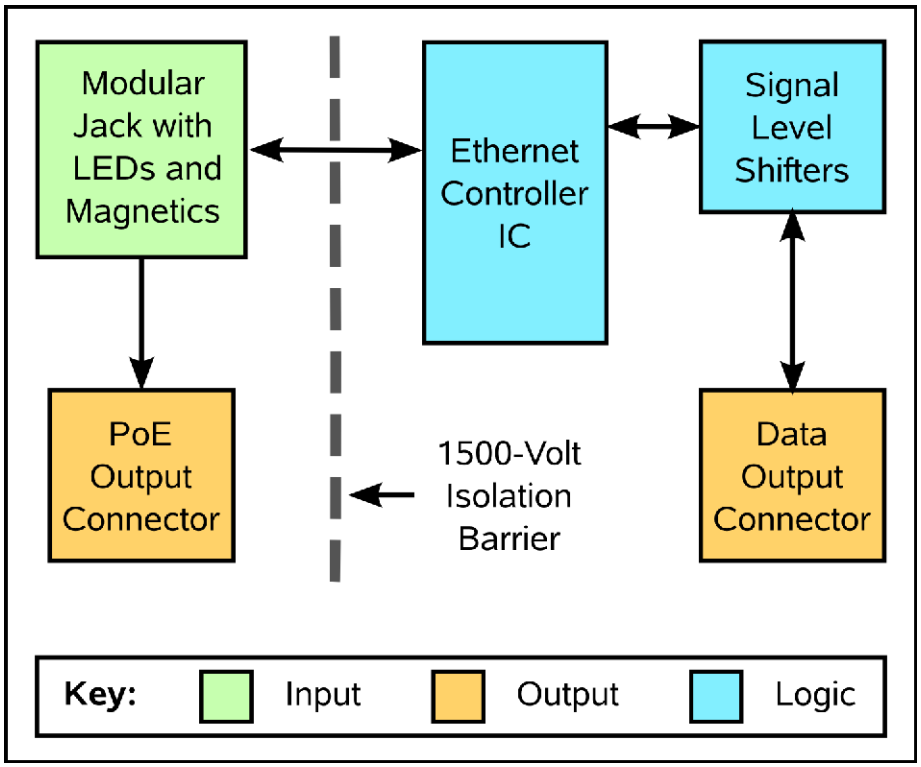

Figure 18-2: THAT Network Interface Block Diagram

4.1.2. Ethernet Controller

Since the Ethernet controller IC is the main component of the NIC, it was the first part selected. Unfortunately, there are very few Ethernet controller ICs made in through-hole packages, an attribute which was important to the developers.

One Ethernet controller that is readily available in a through-hole package is the Microchip ENC28J60 Ethernet Controller. The main benefits of the ENC28J60 are that it uses a standard 3-wire synchronous serial interface (SPI) for communication with a microcontroller, has only 28-pins, supports auto-negotiation and half or full duplex modes, and is affordable. Thus, the ENC28J60 was selected for use in THAT Network Interface.

Note that the components mentioned here represent the “best” choices available at the time of this writing, as determined by the developers. Other options exist.

4.3. Parts List

The parts list for THAT Network Interface is shown in Figure 20-1.

4.4. Schematic

The schematic for THAT Network Interface is shown in Figure 21-2. Some specific notes about this schematic are listed below:

- The 50-ohm, 1% tolerance resistors for R7-R10 are (apparently) difficult to find. Substitutions of 49.9, 51, or similar resistances have been found to work.

- The modular jack (Digikey P/N A99644-ND) contains the necessary additional components for proper IEEE 802.3 support. These include two 1:1 center-tapped transformers, matched chokes, and bridge rectifiers needed to capture PoE power. The jack also includes two LEDs which will be used to indicate network link and activity.

- The AND Gate (IC1) is being used as a buffer to convert the ENC28J60 3.3 Volt outputs to 5 Volt logic. If the microcontroller is operated at 3.3 Volts, this IC as well as D1 and D2 can be omitted.

- The ENC28J60 uses a 25 MHz crystal to drive its internal oscillator. It can then output a clock signal of various speeds using an internal divider / prescaler. In a future version this clock output may be fed into the microcontroller's XTAL1 pin to provide it with a clock (depending on final voltage and speed requirements.)

- The choke L1 is ill-defined. The ENC28J60 datasheet only suggests that the choke must be rated to carry at least 80mA. Guido Socher from tuxgraphics.org reports that a handful of turns around a 5mm ferrite bead works well. We chose to use a 10uH, 228mA inductor from API Delevan, Inc. (Digikey P/N DN42077-ND). The Ethernet controller has been found to work correctly with this inductor. With that said, it is somewhat overkill for this application, and cheaper options exist.

Parts List (version 0.3.2) – THAT Network Interface
Nick Viera – 2009.12.28 – Updated: 2010.04.06

			Temperature				
Part ID	Type	Attr.	Min.	Max	Package	Manufacturer Part	Qty
C1, C2	Capacitor	10uF, 25V, electro.	-40	85	Round, 4mm	ECE-A1EKA100	2
R1,2	Resistor	1k, 5%, carbon	-55	155	Axial	CF 1/4 1K 5% R	2
R4	Resistor	10k, 5%, carbon	-55	155	Axial	CF 1/4 10K 5% R	1
R5,6	Resistor	100, 5%, carbon	-55	155	Axial	CF 1/4 100 5% R	2
R3	Resistor	2k, 1%, film			Axial	RNF 1/4 T1 2K 1% R	1
R7-10	Resistor	50, 5%, carbon	-55	155	Axial	CFR-25JB-51R	4
C3,4	Capacitor	20pF, 50V, ceramic	-55	125	2.5mm grid	RPE5C1H200J2P1Z03B	2
C5-9	Capacitor	0.1u, 50v, ceramic	-55	125	2.54mm grid	K104K15X7RF5TL2	5
IC2	IC	Ethernet controller, SPI	-40	85	DIP-28	ENC28J60-I/SP	1
IC1	IC	Quad AND gate	-40	125	DIP-14	74HCT08N,652	1
C10,11	Capacitor	10nF, 50v, ceramic	-55	125	2.54mm grid	K103K15X7RF5TL2	2
Q6	Crystal	25MHz, 20pF, +/-50ppm	-20	70	HC49/US	FOXSLF/250F-20	1
D1,2	Diode	Schottky, 200mA, 30V	-65	125	DO34/35	BAT85	2
L1	Inductor	100uH, 165mA, 5.5MHz	-55	105	Axial	78F101J-RC	1
X2	Connector	8-pin, 2.54mm pitch			2.54mm grid	PPPC081LFBN-RC	1
X3	Connector	3-pin, 2.54mm pitch			2.54mm grid	PPPC031LFBN-RC	1
X1	Connector	8P8C, Mag, PoE, 2-LED			Rectangle	1-6605310-1	1

Figure 20-1: THAT Network Interface parts list

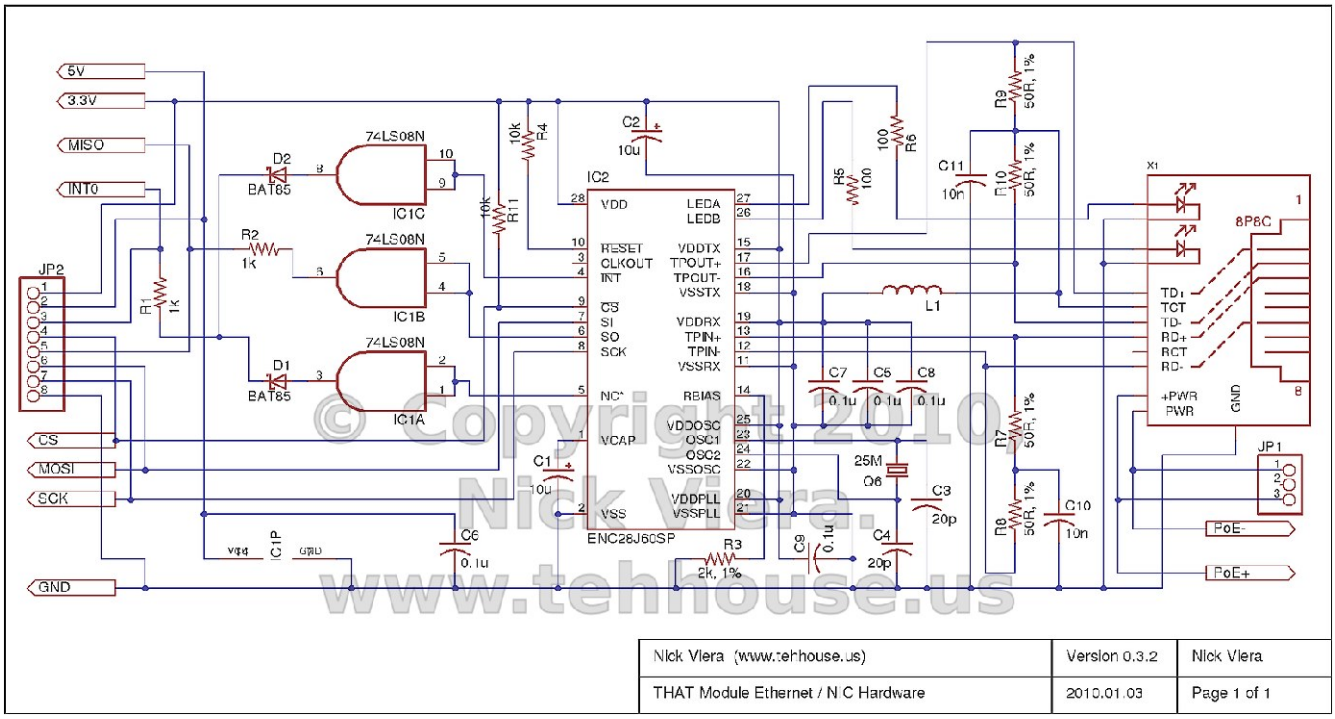

Figure 21-2: THAT Network Interface Schematic

4.4. PCB Layout

A Printed Circuit Board design for the Ethernet Controller Board has been completed. The PCB is a standard, 2-layer board with through-hole components. It measures 70.1 mm x 43.2 mm. The PCB design layout is shown in Figure 22-1, followed by a photo of the PCB populated with components in Figure 22-2.

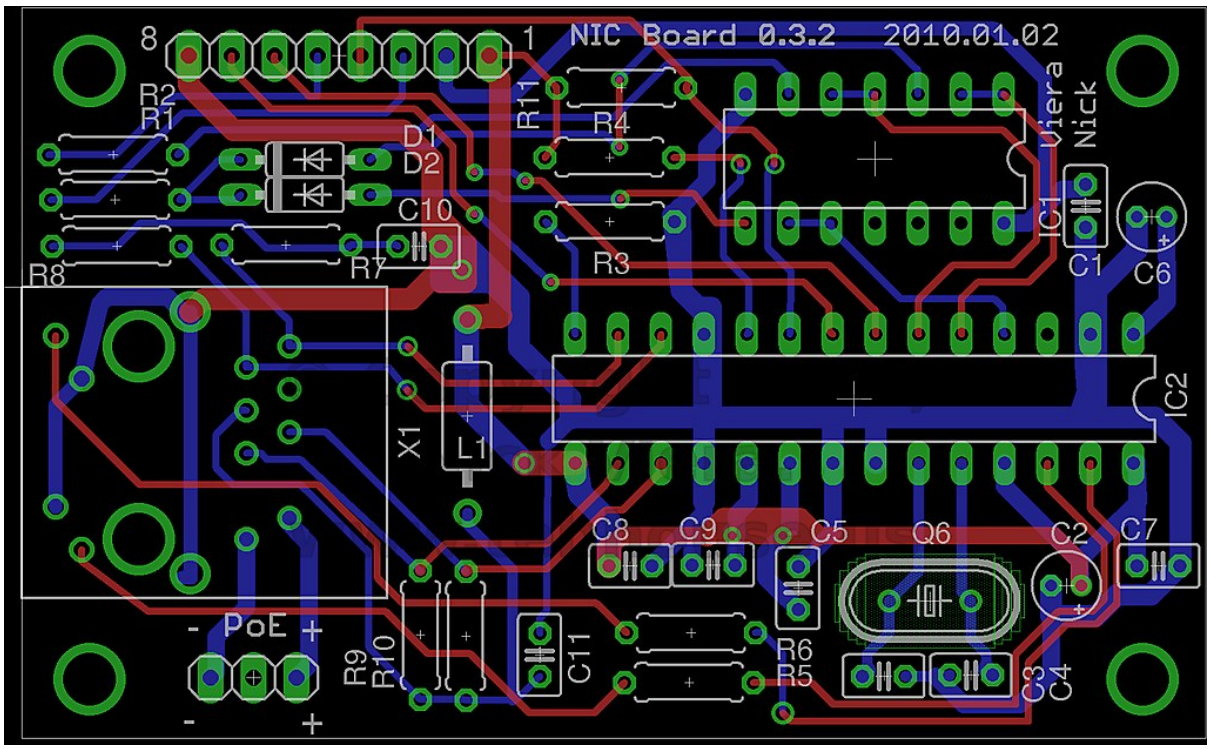

Figure 22-1: THAT Network Interface PCB Layout

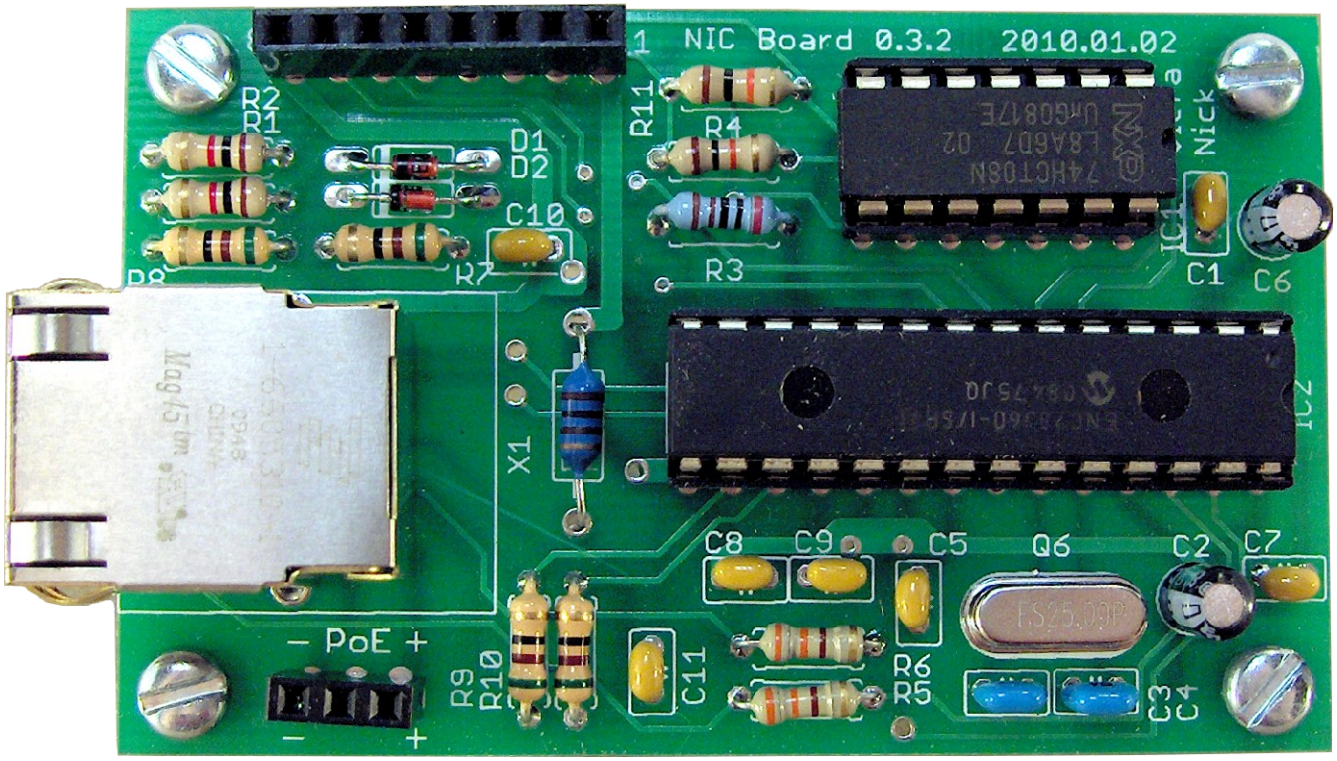

Figure 22-2: Complete THAT Network Interface

4.5. Connections

The NIC board contains two female pin header connectors (2.54mm pitch), in addition to the 8P8C modular Ethernet jack. The small, 3-pin header is for PoE, and it connects to the PoE input header on the THAT Power Supply board. The Large, 8-pin header is for data and power for the Ethernet Controller. It is designed to connect to the mainboard of a THAT module.

The 8-pin data header pinout was defined so that the constant power signals (VCC) are near one end, with ground at the opposite end. The data lines then are oriented so that the higher-frequency data lines are close to ground, while the lower-frequency lines are closer to VCC.

The pinout of the 8-pin data header is as follows:

- Pin 1: 3.3 Volts (VCC2)
- Pin 2: 5 Volts (VCC1)
- Pin 3: Interrupt Signal
- Pin 4: Ethernet Chip Select Signal (CS)
- Pin 5: Serial Peripheral Interface Slave Out (MISO)
- Pin 6: Serial Peripheral Interface Slave In (MOSI)
- Pin 7: Serial Peripheral Interface Clock (SCK)
- Pin 8: Ground

5. Digital Thermostat Module

5.1. Introduction

The Digital Thermostat Module (version 1.0), code-named COPTA, is an advanced control module for use with THAT. As such, the COPTA design shares the basic goals and requirements defined for THAT. This module has been developed by Nick Viera.

COPTA is a digital thermostat with an advanced feature set and flexible programmable control. It is designed for use with most standardized residential and light-commercial, single or split-unit HVAC systems.

5.2. Hardware Specifications

5.2.1. Power Supply and Ethernet Interface

The digital thermostat module makes use of the THAT Power Supply and THAT Network Interface. These components are described in detail previously in sections 3 and 4, respectively.

5.2.2. Microcontroller Platform

The COPTA hardware consists of an embedded microcontroller system. The microcontroller is an Atmel AVR Atmega324P.

The Atmel AVR family of microcontrollers was chosen for this design primarily because they are widely available, have excellent support for open-source software, and are easy to use with the Linux operating system.

The Atmega324P was selected for this design because it offers a large quantity of Input/Output (I/O) pins coupled with a large flash memory space. In addition, the Atmega324P is pin-compatible with 3 other AVR microcontrollers (Atmega164P, Atmega644P, and Atmega1284P) which offer options for more flash memory space. Thus, there is the possibility for easy future upgrades should more memory space be needed.

In addition to the microcontroller, a Microchip MCP23008 I/O port expander is used to provide and additional 8 I/O pins. The use of the port expanded was necessary to ensure that all peripherals and user interface devices could be implemented without running out of I/O pins.

5.2.3. Sensors

COPTA contains an on-board analog temperature sensor and a capacitance-based relative humidity sensor. Both sensors are read by the microcontroller through its integral Analog to Digital converter (ADC).
The temperature sensor is a Microchip MCP9700A IC. It reads temperatures in the range of -40 to 125 °C with an accuracy of ±2 °C across typical indoor air temperatures (0 to 70 °C).

The MCP9700A was selected for this design because it is inexpensive, physically small, and has both the accuracy and measuring ranges desired for this design. In addition, the sensor can operate at either 3.3 or 5 Volts DC and is offered in a through-hole package.

The humidity sensor is a Honeywell HCH-1000 sensor. It measures levels of 10% to 95% Relative Humidity with an accuracy of ±2 %RH. The HCH-1000 sensor was selected for this design because it is offered in a through-hole package, has an integral case for dust-protection, and is less expensive than other alternatives.

5.2.4. Real Time Clock

For programmability, the thermostat design includes a real time clock (RTC), which keeps time using its own crystal oscillator (for stability and accuracy). The RTC used is the Dallas DS1337+ IC. This IC was selected mainly due to its availability in a small, through-hole package and its I2C data bus.

5.2.5. User Interface

The user interface allows for the direct, local interaction between the user and the thermostat module. The interface components have been selected to provide enough feedback and sources of input to make the interface as intuitive as possible. The user interface consists of:

- 5 LED indicator lamps (1 each of red, amber, green, blue, and white)
- 6 momentary pushbutton switches, laid out as 3 discrete pairs
- A large, graphical LCD screen with backlight

The LCD screen is a 128x128 pixel chip-on-glass device manufactured by Newhaven Display International. It supports 16 shades of gray, and is almost exactly square in shape.

This particular LCD was selected for the thermostat module mainly due to its physical size and shape matching the desired traits. A lot of emphasis was placed on finding such an LCD, in order to stay true to the physical design concept (see section 5.4).

5.2.6. Peripheral Communication Interfaces

Two serial and one parallel communication interface are used to transfer data between the microcontroller and peripherals.

A two-wire synchronous, I2C-compatible, serial interface is used to communicate with the RTC and the I/O port expander ICs. I2C was chosen due to the wide availability of I2C components, as well as the fact that it only requires 2 signal lines for communication with the microcontroller.
A 3-wire synchronous serial interface (called the Serial Peripheral Interface or SPI) is used to communicate with the THAT Network Interface. SPI is used because the Ethernet controller IC on the THAT Network Interface is only available with an SPI bus.

Lastly, a parallel interface is used to communicate with the LCD screen due to the requirement for rapid communication (the LCD's serial interface is slow in speed and complicated to use). The parallel interface uses 8 signal lines for data and 3 signal lines for control purposes. The data arbitration method is standardized and is commonly known as the “Parallel 68” or “6800-series” interface.

5.3. Software Specifications

The microcontroller firmware for COPTA is written in C and compiled into AVR assembly code by AVR-GCC. The firmware implements both the functionality necessary for thermostat operation, and the Ethernet / TCP stack necessary for THAT communications.

Overall software design goals are listed below:

- Modularity in design.
- Configurable support for single and multi-stage A/C and heat-pump systems.
- Configurable support for controlling dynamically variable HVAC systems.
- Configurable support for reading external temperature/humidity sensors.
- “Learning” of codes from Infrared remotes.

5.4. Design Concept

The COPTA module is designed to be a stand-alone unit capable of directly replacing any “standard” wall-mount thermostat. To achieve this goal, the unit must not be too large in size and should be as aesthetically pleasing as possible. The physical design concept for COPTA is a unit approximately 115mm in diameter, and 51mm deep. An example packaging concept for the digital thermostat prototype is shown in Figure 28-1.

Figure 28-2 shows a brief overview of the user interface elements of the Digital Thermostat Module. The pushbutton switches all serve multiple functions. Primary functions are shown in black. Secondary functions, such as when in a menu or settings mode, are shown in [Blue]. Finally, functions which are executed when a button is held down for 5+ seconds are shown in (Red).

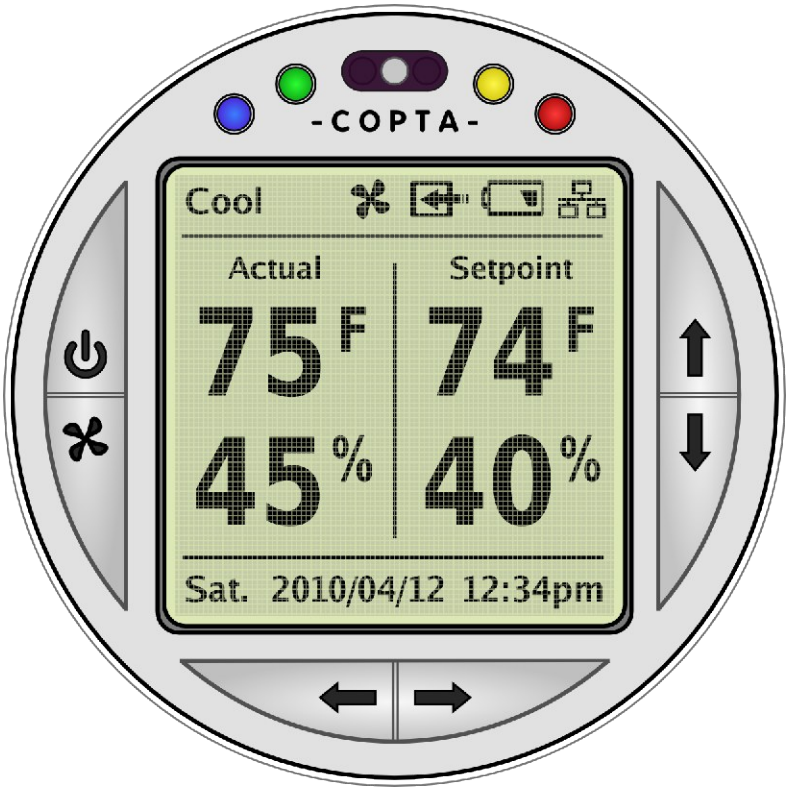

Figure 28-1: COPTA Physical Design Concept

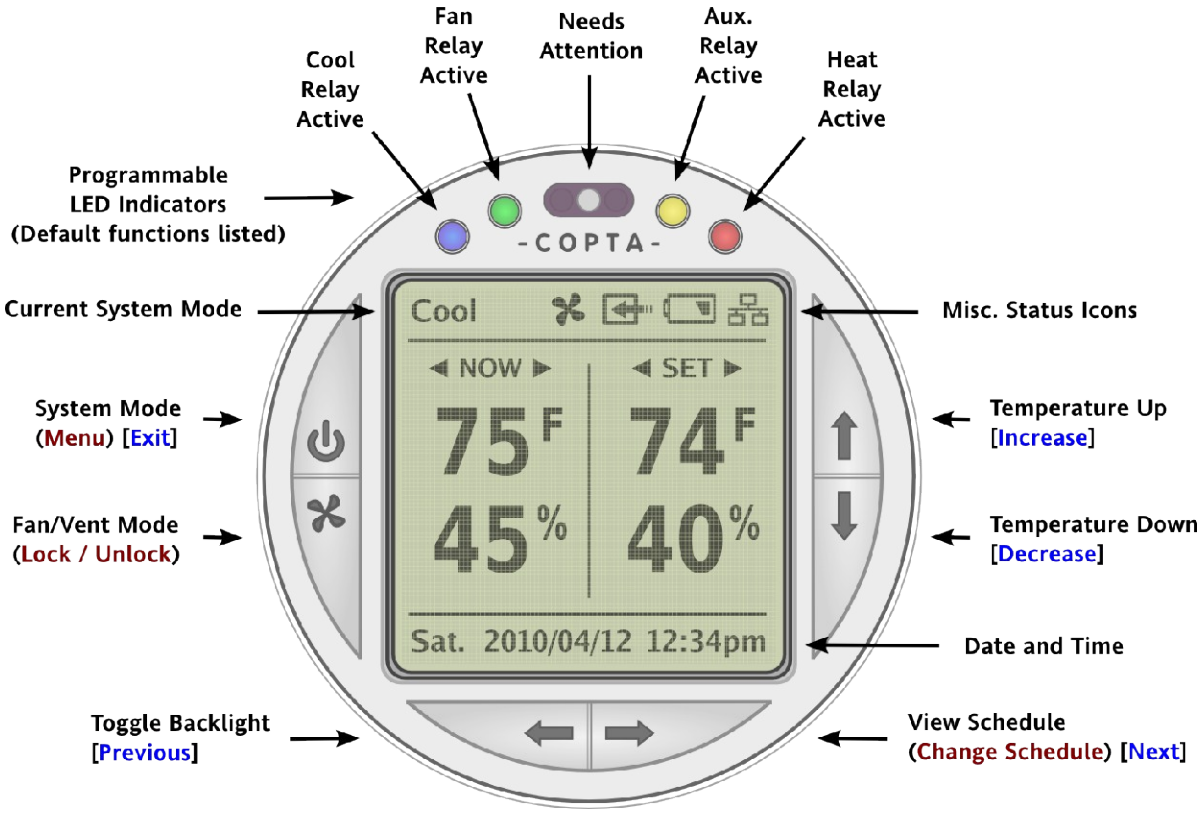

Figure 28-2: COPTA user interface key

5.5. Block Diagram

The overall hardware block diagram for the module is shown in figure 29-1.

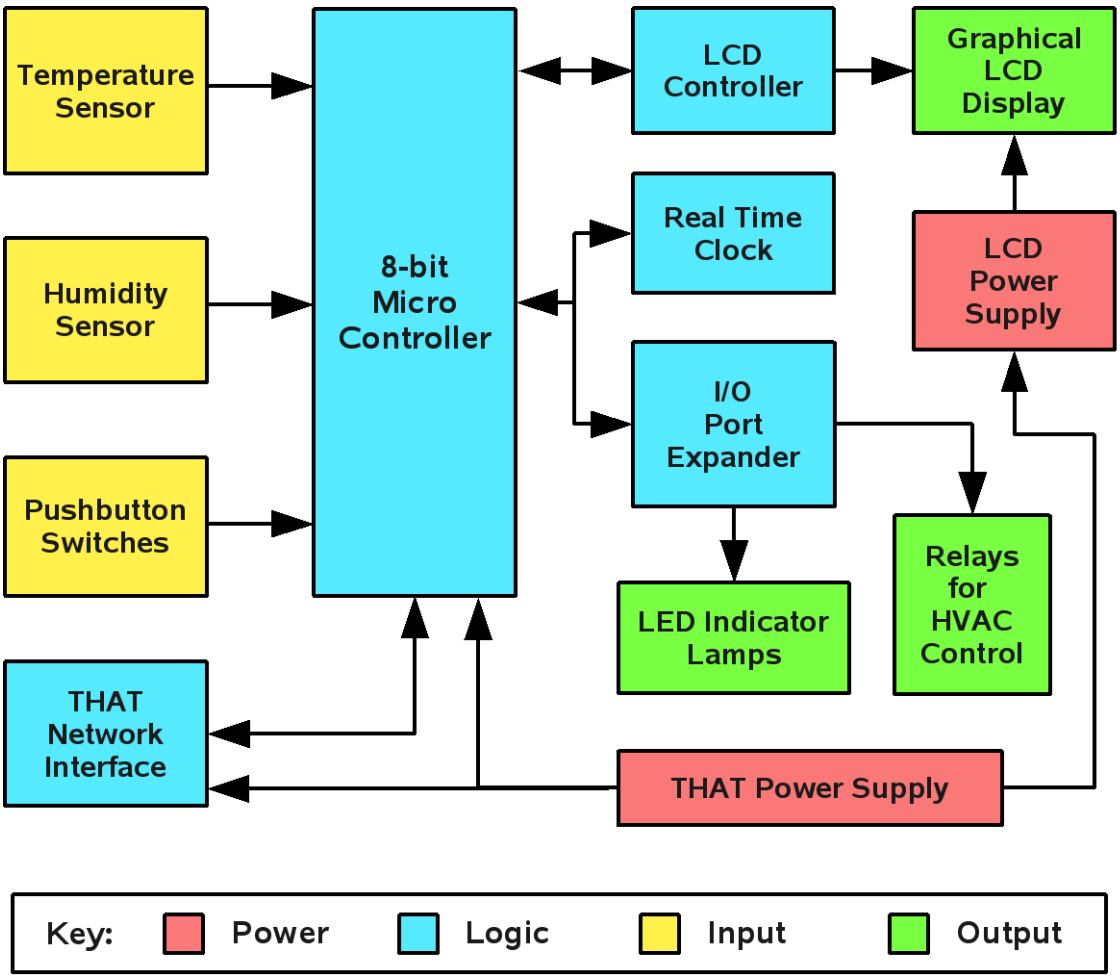

Figure 29-1: Digital Thermostat Module Block Diagram

5.6. Parts List

The parts list for the digital thermostat module is shown in Figure 30-1.

Parts List – Digital Thermostat Module (COPTA) Nick Viera – 2010.04.10							
			Temperature				
Part ID	Type	Attr.	Min.	Max	Package	Manufacturer Part	Qty
Q8	Crystal	32.768K, 6pf	-20	70	Cylinder	C-002RX 32.7680K-E	1
IC3	IC	RTC, I2C	-40	85	DIP-8	DS1337+	1
S1-5	Switch	SPST, mom, 150gf	-25	70	6mm square	B3F-1022	5
	Socket	DIP-14, 0.3" wide, open			DIP-14	1-390261-3	1
	Socket	DIP-40 wide			DIP-40	4840-6000-CP	1
	Socket	DIP-28, 0.3" wide, open			DIP-28	1-390261-9	1
X1	Connector	2x3-pin, 2.54mm grid			Rectangle	75869-131LF	1
IC1	IC	Atmega324, 20MHz	-40	85	DIP-40	ATMEGA324P-20PU	1
C7	IC	Humidity, 10-95%RH	-40	120	SIP-2	HCH-1000-002	1
IC2	IC	Temp, -40-125C, +/-2C	-40	125	TO-92	MCP9700A-E/TO	1
L1	Inductor	10uH, 210mA	-40	105	1008 (2520)	NLCV25T-100K-PF	1
C1,2	Capacitor	20pF, 50V, ceramic	-55	125	2.5mm grid	RPE5C1H200J2P1Z03B	2
Q6	Crystal	20MHz, 20pF			HC49/US	FOXSLF/200-20	1
C3-6	Capacitor	0.1u, 50v, ceramic	-55	125	2.54mm grid	K104K15X7RF5TL2	4
R1-6,17-24	Resistor	1k, 5%, carbon	-55	155	Axial	CF 1/4 1K 5% R	14
R7,9	Resistor	270, 5%, carbon	-55	155	Axial	CF 1/4 270 5% R	2
R8,10,11	Resistor	180, 5%, carbon	-55	155	Axial	CF 1/4 180 5% R	3
K1-4	Relay	SPDT, 125V/0.5A, 5V	-40	85	Rectangle	IMC03CTS	4
D1-4	Diode	Fast, 200mA, 100V	-65	175	DO35	1N4148	4
Q2-5,7	Transistor	MOSFET, 60V, 200mA	-55	150	TO-92	2N7000	5
for lcd	Capacitor	1uF, 50V	-30	85	radial	FK14Y5V1H105Z	5
for lcd	Inductor	10uH, 950mA,	-40	85	radial	11R103C	1
for lcd	IC	Vreg, 4.8-22V, 10mA	-40	85	SOT-23-5	576-2597-1-ND	1
	IC	I/O expander, 8-ch	-40	125	DIP-18	MCP23008-E/P	1
for psu	Resistor	100K, 1%, 1/4W			axial	MFR-25FBF-100K	1
	Connector	8-pin, 2.54mm pitch			2.54mm grid	68002-408HLF	2
	Connector	3-pin, 2.54mm pitch			2.54mm grid	68002-404HLF	4
	LCD		-20	70	**Newhaven**	NHD-C128128BZ-FSW-GBW	1
for NIC	Inductor	100uH, 165mA, ferrite	-55	105		78F101J-RC	1
R13-16,25	Resistor	4.7k, 5%, carbon	-55	155	Axial	CF 1/4 4.7K 5% R	5

Figure 30-1: Digital Thermostat Module Parts List

5.7. Schematics

The schematic for the digital thermostat module is split into 3 parts. The first contains the microcontroller and all general components. The second contains the components which use the I2C bus as well as the status LEDs and HVAC relays. The last contains the LCD power supply components and LCD connectors.

5.7.1. Microcontroller Schematic

The microcontroller schematic is shown in Figure 31-1.

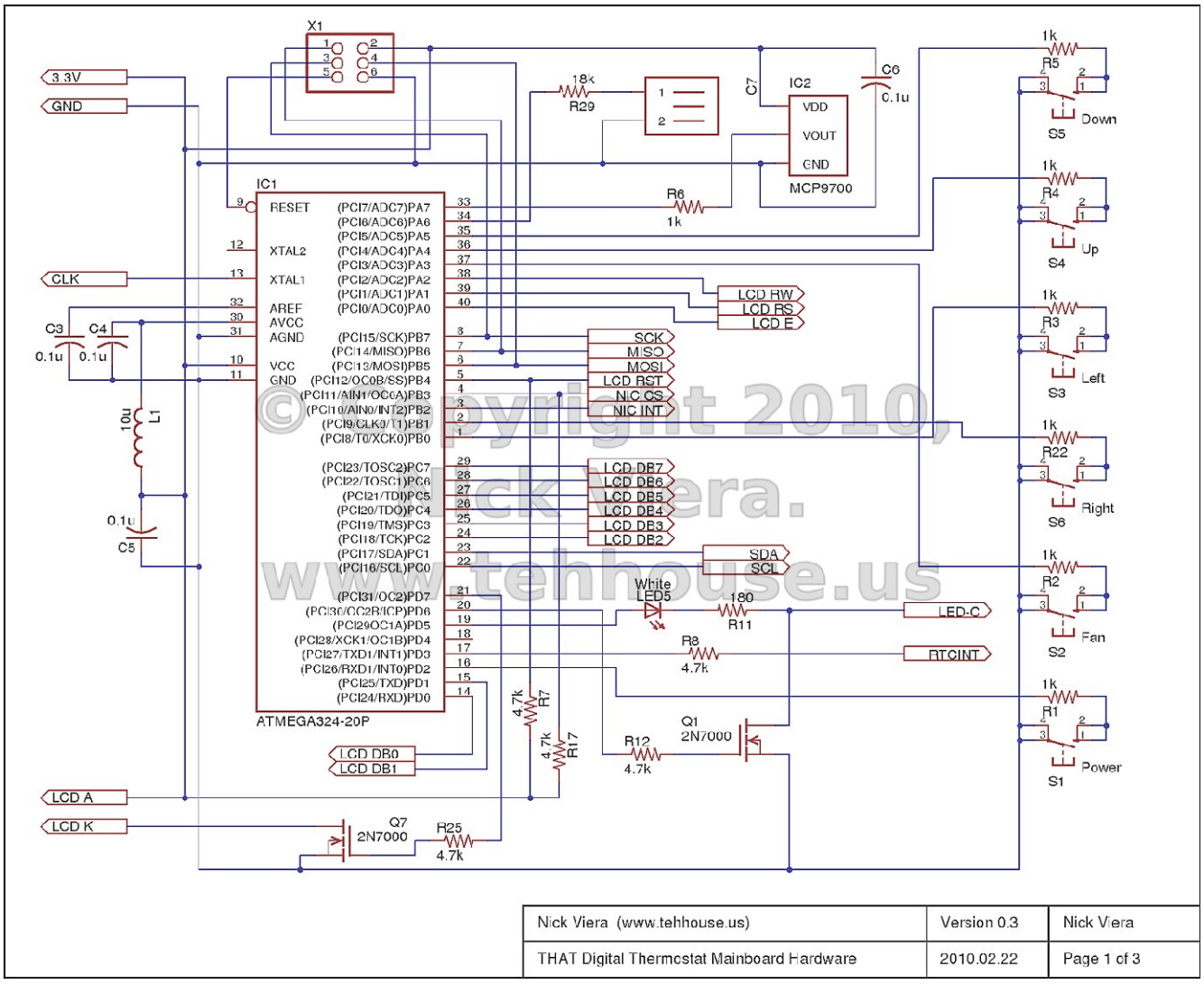

Figure 31-1: Digital Thermostat Microcontroller Schematic

5.7.2. I2C Peripherals Schematic

The I2C peripherals schematic is shown in Figure 32-1. Notes about this schematic are listed below.

- R1 and R2 are pull-up resistors for the I2C bus lines. The criteria for their selection is listed in the Atmel Atmega324 datasheet, shown below.
- MOSFETs Q1 through Q4 are used as buffers so that the relays can be driven without exceeding the current limit per I/O pin of the AVR microcontroller.
- D1 through D4 are flyback diodes to prevent large voltage spikes from appearing across the MOSFETs' drain-source junction when the relay coils are de-energized.

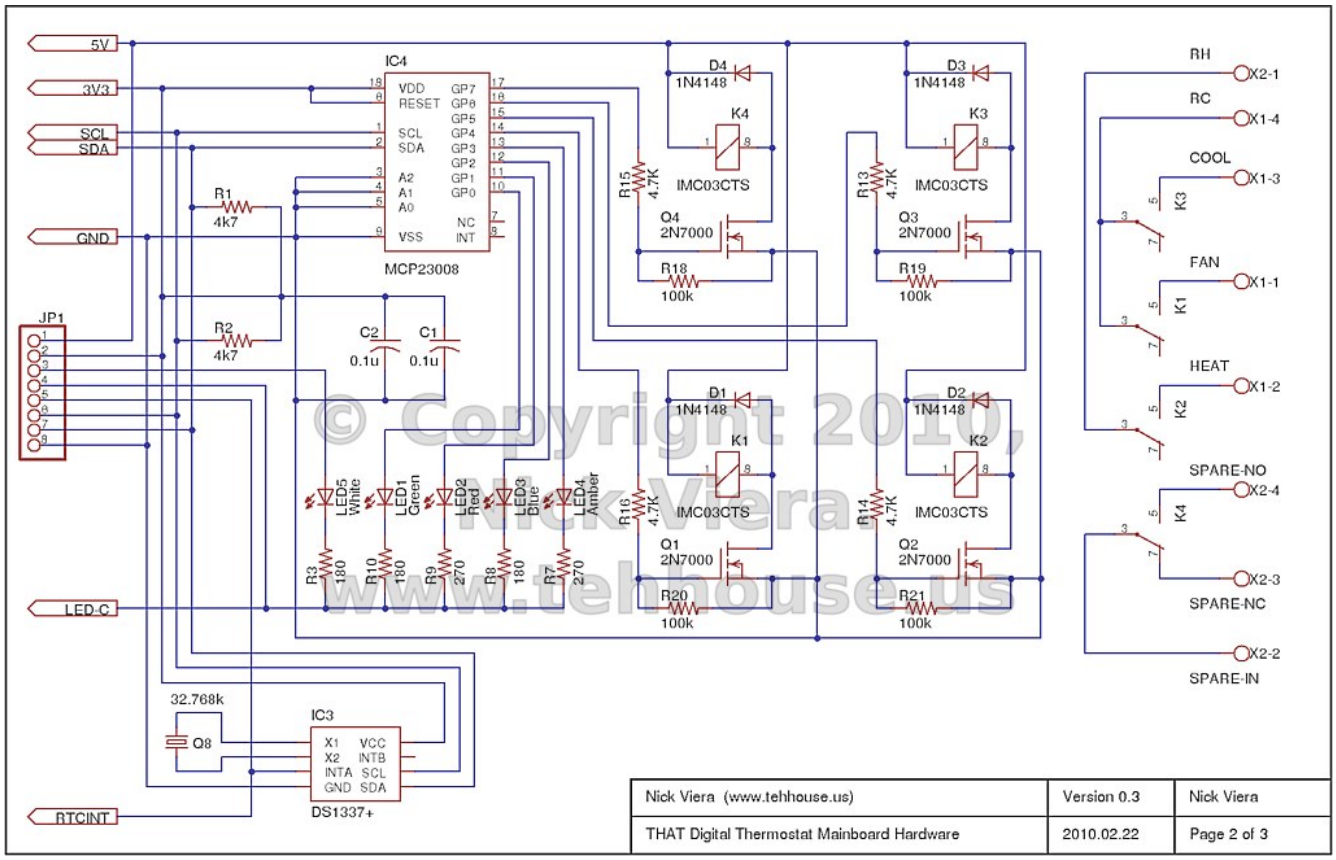

Figure 32-1: Digital Thermostat I2C Peripherals Schematic

5.7.3. LCD Components Schematic

The LCD components schematic is shown in Figure 33-1. Notes about this schematic are listed below.

- C2 through C6 are used to stabilize the LCD driving voltages which are created internally by the LCD controller IC. Each capacitor has a varying percentage of the LCD contrast voltage (15 Volts) across it. For example, if even distributed, the capacitors would be at 3,6,9,12, and 15 Volts respectively.
- IC1, L1, and D1 create a switch-mode boost converter which outputs 15 Volts from the 5 Volt input. This output is used to provide the LCD driving power.
- C7 is used as a buffer / stabilizer for the 15 Volt rail.

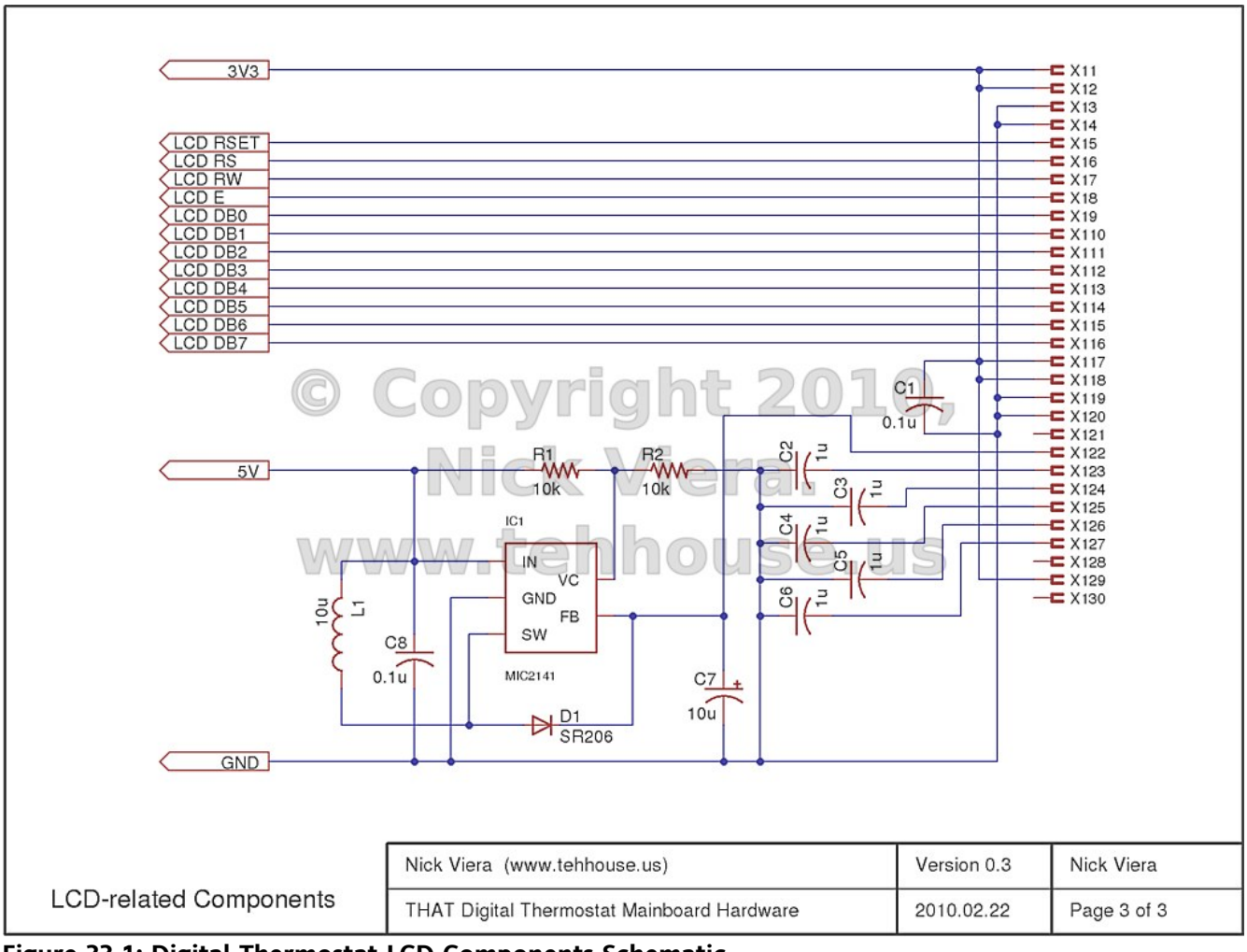

Figure 33-1: Digital Thermostat LCD Components Schematic

6. Electronic Access Module

6.1. Introduction

The Electronic Access Module (E.A.M. Version 1.0) is an advanced control module for use with THAT. It shares the basic goals and requirements defined for THAT and is designed for use with residential and light-commercial buildings. An additional submodule, called the Door Widget, extends the EAM's functionality by adding a doorbell button and a randomly-rotating passcode interface. Both devices were developed by Chris Miller.

6.2. Hardware Specifications

6.2.1. Power Supply and Ethernet Interface

The EAM makes use of the THAT Power Supply and THAT Network Interface. These components are described in detail previously in sections 3 and 4, respectively.

6.2.2. Microcontroller Platform

The EAM hardware includes an embedded 8-bit microcontroller. The microcontroller is an Atmel AVR Atmega324P.

The Atmel AVR series of microcontrollers were chosen for this design primarily because they are widely available, have excellent support for open-source software, and are easy to use with any desktop operating system (including Linux).

The Atmega324P was selected for this design because it offers a large number of Input/Output (I/O) pins coupled with a large flash memory space. In addition, the Atmega324P is pin-compatible with three other microcontrollers in the AVR series (Atmega164P, Atmega644P, and Atmega1284P) which offer options for more flash memory space. Thus, there is the possibility for easy future upgrades should more memory space be needed.

6.2.3. User Interface

The EAM's user interface allows for the direct, local interaction between the user and the EAM. The interface components were selected to provide enough feedback and sources of input to make the interface as intuitive as possible. The user interface consists of:

- 3 bi-color LED indicators
- 8 momentary pushbutton switches
- A large, graphical LCD screen with backlight

The LCD is a 128x64 pixel, 3-5/8” screen manufactured by Newhaven Display International. It is monochromatic and has a serial interface. This particular LCD was selected for the EAM mainly due to its relatively large size, increasing its visibility at larger distances.

The Door Widget has a user interface of its own that allows the user to indirectly request access from the EAM. The interface components were selected to allow for intuitive operation. The user interface consists of:

- 10 seven-segment LED displays
- 11 momentary pushbutton switches
- 1 bicolor LED indicator

Seven segment LED displays have several benefits for use in a device such as the Door Widget including their many size options, dimming ability, and flexible spacing options. The particular units used in the Door Widget were chosen due to their wide operating temperature range, high brightness and small size.

6.2.4. Peripheral Communication Interfaces

Two serial communication interfaces are used to transfer data between the microcontroller and peripherals.

A three-wire synchronous serial interface (Serial Peripheral Interface or SPI) is used to communicate with the THAT Network Interface. SPI is used because the Ethernet controller IC on the THAT Network Interface is only available with an SPI bus.

The SPI bus is also used to transmit data to the LCD. The LCD is unidirectional (only capable of receiving data) and therefore uses only two of the three SPI lines.

A two-wire synchronous, I2C-compatible, serial interface is used by the EAM and Door Widget for communication with one another. I2C was chosen for this communication for the following reasons:

- The microcontroller supports I2C in hardware
- Multiple nodes (up to 112) may exist on the same bus
- Distances of over 75ft may be reached with the aid of additional components (bus extender)
- The EAM firmware could be further developed to accept other I2C submodules (such as single doorbell buttons or door-open sensors).

6.3. Software Specifications

The base firmware for the EAM and Door Widget is written in the C programming language. The firmware currently implements the functionality necessary for electronic access through the main user interface of the EAM. Significant progress was made towards working communication between the EAM and Door Widget, as well as the master computer through the EAM's Ethernet / TCP stack. These two portions have been fully-implemented in hardware and partially-implemented software but are still non-functional.

6.4. Design Concept

The EAM was designed with utilitarianism in mind. It could easily be mounted within a metal box in an out-of-the-way location.

The Door Widget, however, was designed to be aesthetically pleasing due to being installed near residential entrances. The physical design concept drawing used during the design process is shown in Figure 38-1 with important features indicated in Figure 38-2.

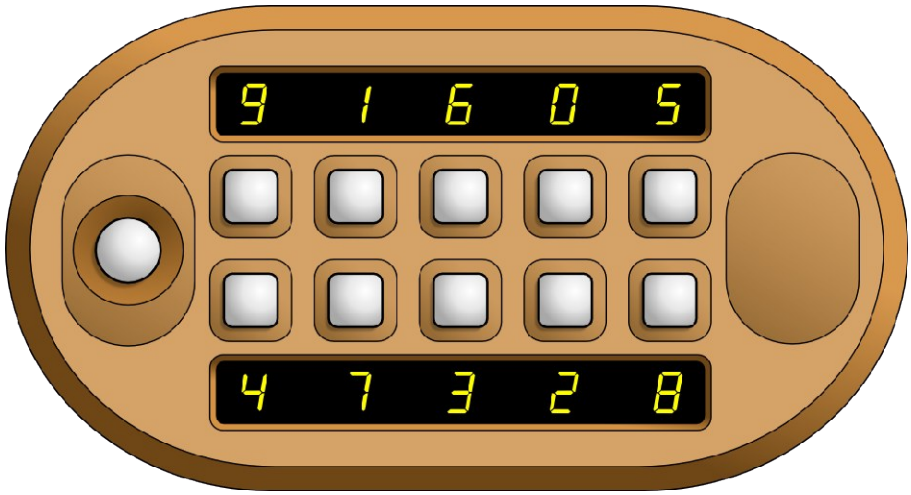

Figure 37-1: Door Widget Design Concept Drawing

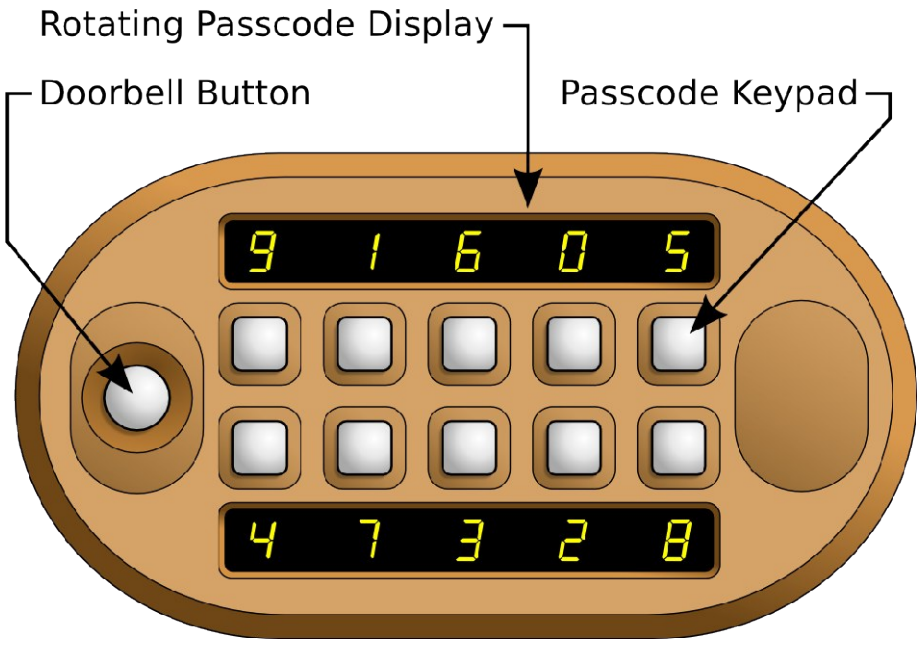

Figure 37-2: Door Widget Important Features

6.5. Block Diagram

The overall hardware system block diagram for the EAM is shown in Figure 38-1. The overall hardware system block diagram for the Door Widget is shown in Figure 38-2.

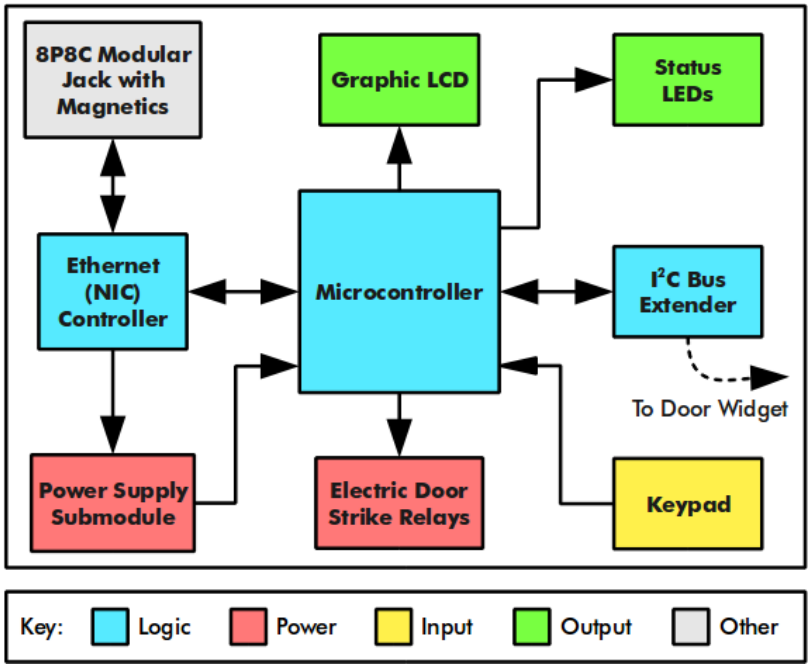

Figure 38-1: Electronic Access Module System Block Diagram

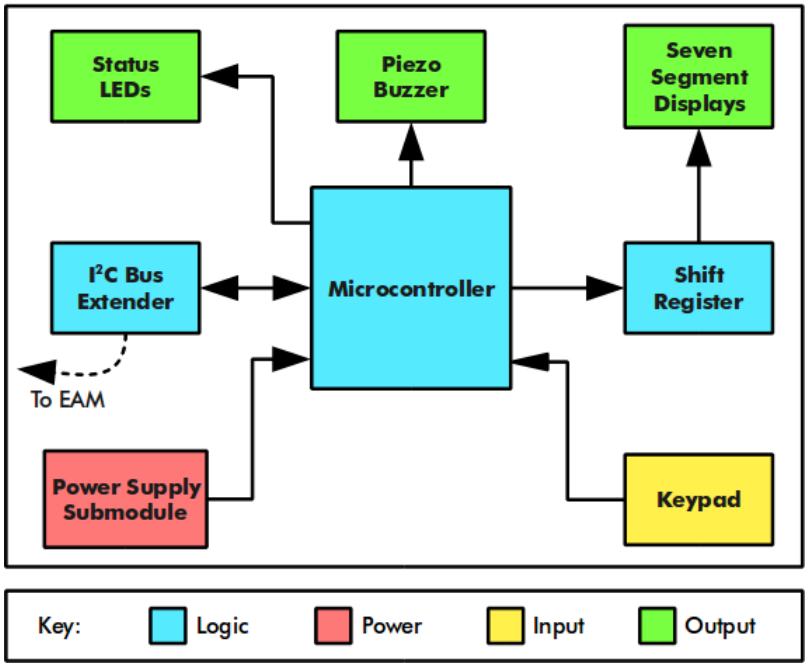

Figure 38-2: Door Widget System Block Diagram

6.6. Parts List

The parts lists for the EAM and Door Widget are shown in Figures 39-1 and 39-2 respectively.

Electronic Access Module Parts List (v1.0) Copyright Chris Miller – 2010-03-23							
			Temperature (°C)				
Part ID	Type	Attributes	Min	Max	Package	Manufacturer Part	Qty
IC1	IC	AVR Microcontroller	-50	85	DIP-40	Atmega324P	1
IC3	IC	I2C Bus Extender	-40	85	DIP-8	P82B96	1
LCD	LCD	128x64 graphical LCD	-20	70	NA	NHD-C12864EZ-FSW-FTW-P	1
Q2–4	MOSFET	N-channel MOSFET	-55	150	TO-92	2N7000_D26Z	3
K1–3	Relay	SPDT, 10A @ 120VAC	-25	85	NA	G5LE-14 DC5	3
S1–8	Switch	SPST-NO, tactile	-25	70	B3F	B3F-1022	8
Q1	Crystal	13.5MHz Crystal Oscillator	-20	70	HC-49US	HC49US-13.500MABJ-UB	1
R1–11	Resistor	5% carbon resistor	-55	155	Axial	CF ¼ 5% R	11
C1,2	Capacitor	25V, electrolytic	-40	85	Round	ECE-A1EKA	2
C3,4	Capacitor	50V, ceramic	-55	125	2.5mm grid	RPE5C1H200J2P1Z03B	2
D1-3	Diode	4A @ 1us surge current	-65	175	DO-35	1N4148TA	3

Figure 39-1: Electronic Access Module Parts List

Door Widget Parts List (v1.0) Copyright Chris Miller – 2010-03-26							
			Temperature (°C)				
Part ID	Type	Attributes	Min	Max	Package	Manufacturer Part	Qty
IC1	IC	AVR Microcontroller	-50	85	DIP-40	Atmega324P	1
IC2	IC	I2C Bus Extender	-40	85	DIP-8	P82B96	1
Q1	Crystal	13.5MHz Crystal Oscillator	-20	70	HC-49US	HC49US-13.500MABJ-UB	1
C1,2	Capacitor	25V, electrolytic	-40	85	Round	ECE-A1EKA	2
C3,4	Capacitor	50V, ceramic	-55	125	2.5mm grid	RPE5C1H200J2P1Z03B	2
R5–20	Resistor	5% carbon resistor	-55	155	Axial	CF ¼ 5% R	16
S1–11	Switch	SPST-NO, tactile	-25	70	B3F	B3F-1022	11

Figure 39-2: Door Widget Parts List

6.7. Schematics

The schematic for the EAM is a single document that can be found in the next subsection. The schematic for the Door Widget is split up into two parts. The first schematic contains the microcontroller (MCU) portion of the design. The second contains the display portion of the design.

6.7.1. EAM Schematic

The schematic for the EAM is shown in Figure 41-1. Notes about this schematic are listed below.

- IC1 is the Atmel Atmega324P microcontroller.
- IC2 is the NXP P82B96 I2C bus extender.
- K1 through K3 are relays with a maximum load rating of 10 A at 120 VAC and 8 A at 30 VCD.
- MOSFETs Q1 through Q3 are buffers that allow the relays to be driven without exceeding the current limit per I/O pin of the AVR microcontroller.
- D1 through D3 are flyback diodes to prevent large voltage spikes from appearing across the MOSFETs' drain-source junction when the relay coils are de-energized.

6.7.2. Door Widget MCU Schematic

The schematic of the Door Widget's microcontroller portion is shown in Figure 42-1. Notes about this schematic are listed below.

- IC1 is the Atmel Atmega324P microcontroller.
- IC2 is the NXP P82B96 I2C bus extender.

6.7.3. Door Widget Display Schematic

The schematic of the Door Widget's display portion is shown in Figure 43-1.

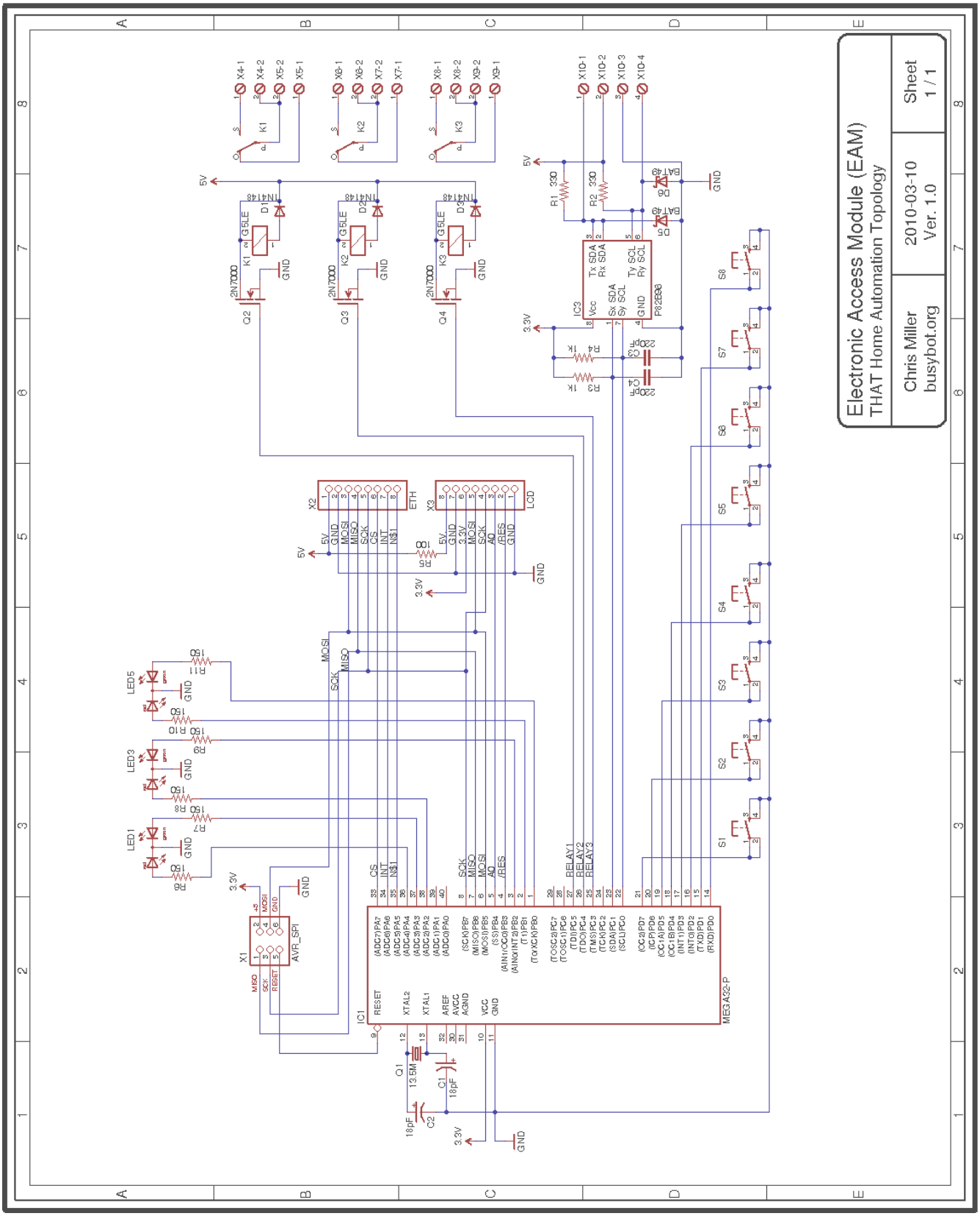

Figure 41-1: EAM Schematic

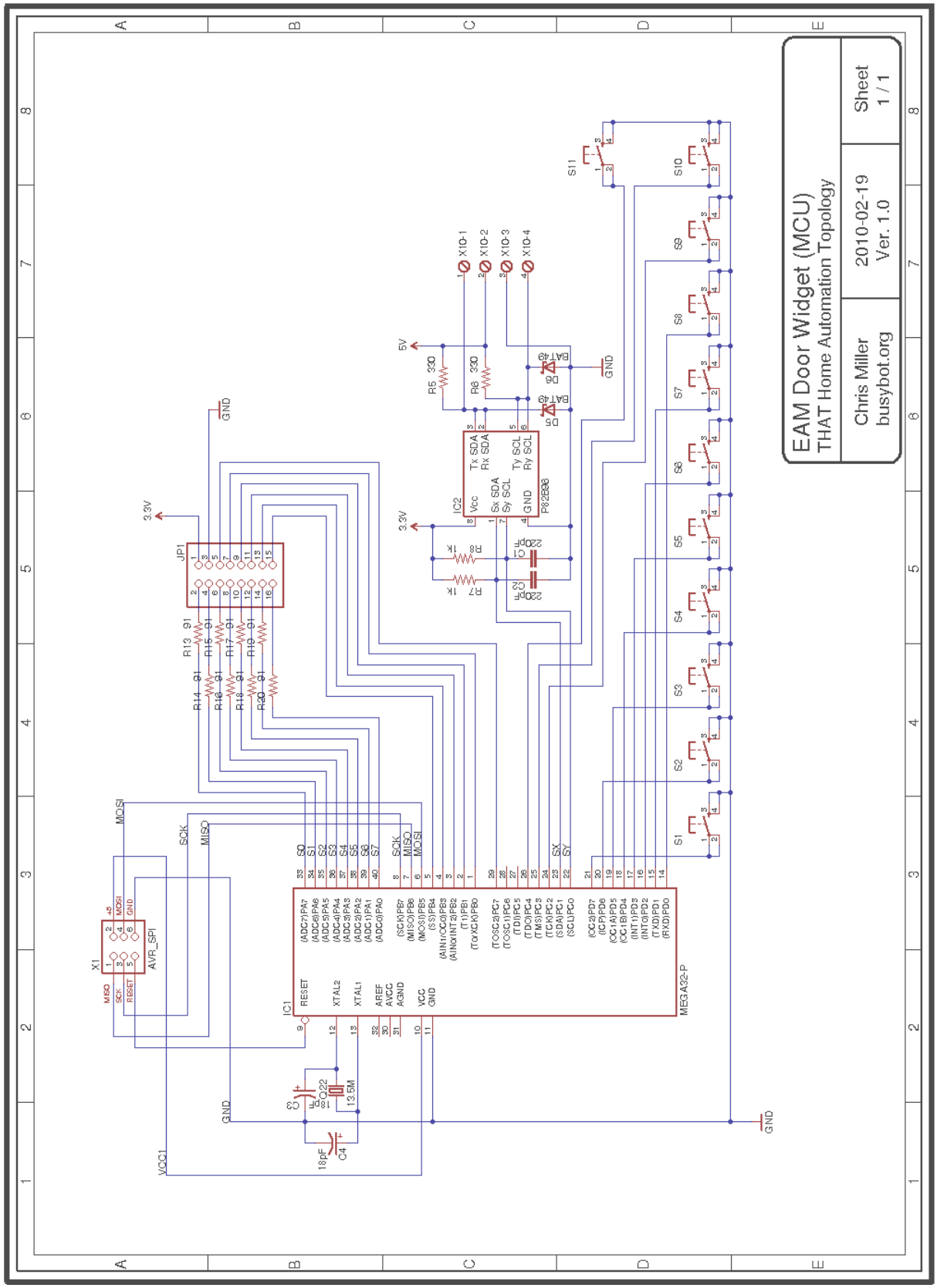

Figure 42-1: Door Widget MCU Schematic

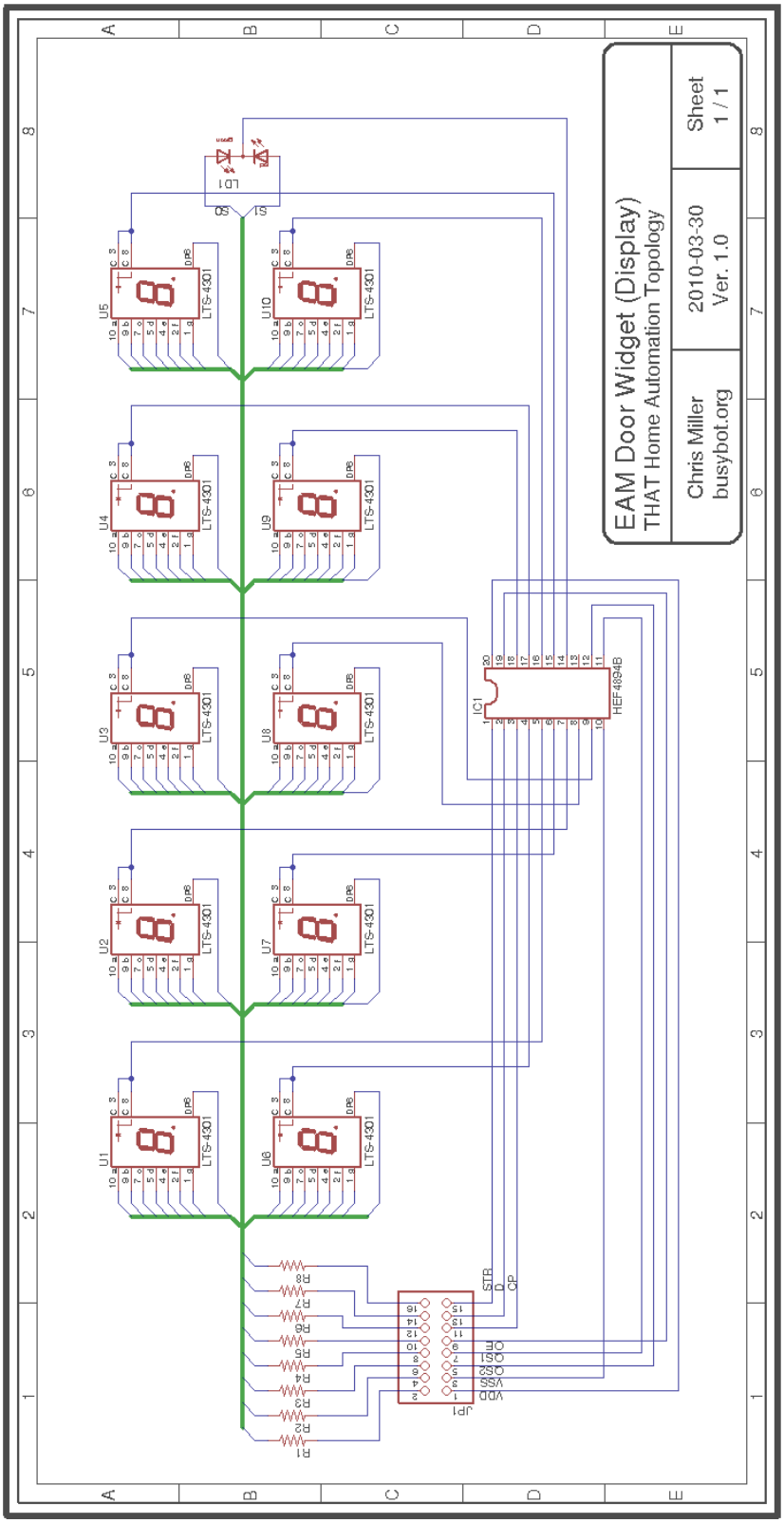

Figure 43-1: Door Widget Display Schematic

7. THAT Control Software

7.1. Introduction

Additional functionality and flexibility of a THAT System can be realized using a computer as a “master” arbitrator for the modules. A computer provides new opportunities such as the capability to remotely control modules, as well as to create advanced schedules and presets for use in controlling modules.

Additionally, a computer with Internet access provides the opportunity to more intelligently control modules based on outside information (such as weather conditions and forecasts).

In order for a computer to interact with THAT, it needs to run software which can communicate using THAT protocols. The developers have created THAT Control Software for this purpose.

7.2. Platform and Interoperability

Early on in the project, the developers made the decision to only design software which is cross-platform compatible, meaning it is capable of running on multiple computer operating systems. At the minimum, support for Linux, MacOS X, and Windows XP was a requirement.

The developers decided to utilize the Python programming language (http://www.python.org/) for this software. Python meets the cross-platform compatibility requirement, and is characterized as a powerful, high-level language that allows for rapid software development.

7.3. Block Diagram

The block diagram for THAT Control Software, showing all software sub-modules, is shown in Figure 45-1.

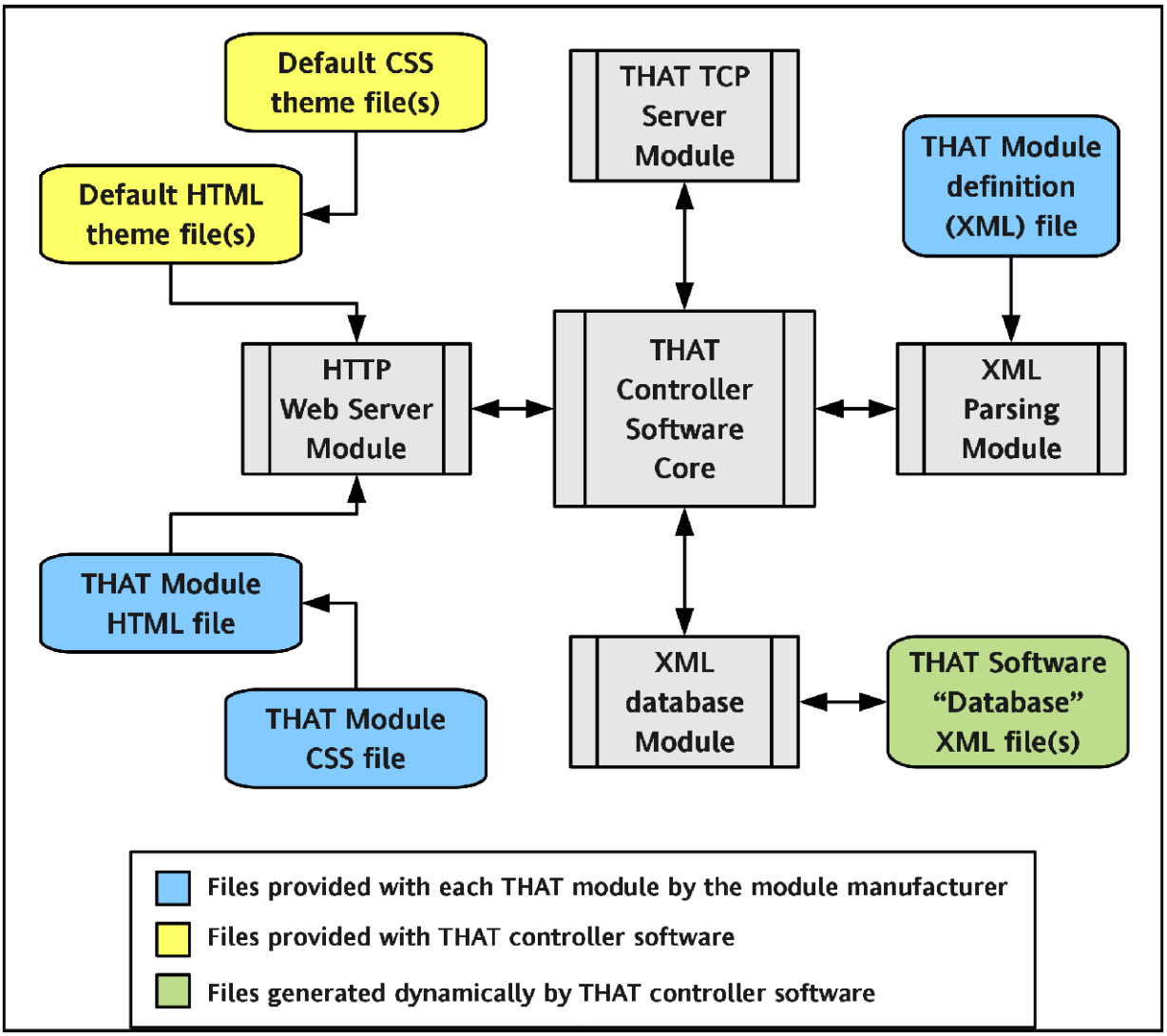

Figure 45-1: THAT Control Software Block Diagram

The gray blocks in the software block diagram represent the core parts of THAT Control Software, which consist of Python code spread across multiple files (See Appendix E for source code.)

The blue blocks represent files which would be provided to the end-user by the manufacturer of a specific THAT module. The most important manufacturer-provided file is the Module definition XML file. This file contains, in XML format, a listing of all of a THAT module's controls, settings, and identification information. Using this file, THAT Control Software can "learn" everything it needs to know about a THAT module, without having any prior knowledge of the module's existence.

The blue blocks are files containing HTML and CSS code which tell THAT Control Software how to design the web interface for a specific THAT module. This allows a manufacturer of a THAT module to customize the section of the web interface that the end-user sees when they access each module's controls and settings.

The yellow blocks represent HTML and CSS files which are provided with THAT Control Software. These files create the “default” web interface which is seen by the end-user. These files could be modified or replaced by the end-user to customize or replace the default web interface with one better suited to their needs.

Lastly, the green blocks represent XML files which are dynamically generated by THAT Control Software. These XML files represent a running “database” of information about the currently-configured THAT Modules. Additionally, these files are used by THAT Control Software to store all user preferences. Thus, the software could theoretically be upgraded/removed/installed without altering or losing any of the current settings or system configuration.

7.4. Markup Language

As mentioned in the previous section, XML, HTML, and CSS markup languages are used by THAT Control Software.

The developers chose to use XML for module definitions as well as the database for THAT Control Software for several reasons. Most importantly, XML is highly human-readable, and its syntax is easy to understand without prior programming experience. In addition, XML is stored as plain-text, so the files are very easy to move, copy, and edit with standard utilities, which is generally not the case for complex database systems such as MySQL.

The developers chose to give the user access to HTML and CSS files used by THAT Control Software. HTML and CSS are used internally by the software to create the user (website) interface. By allowing the users to create and edit certain HTML and CSS files, the look and feel of the web interface can be easily modified and expanded by end-users. The developers feel that access to “raw” HTML and CSS is acceptable because, like XML, HTML and CSS are highly readable languages which many people are already familiar with or can easily learn.

8. Licensing Information

8.1. Hardware Licensing

The hardware designs presented in this document are Copyright © 2010 by Nick Viera and Chris Miller. The hardware designs are released under the terms and conditions of the Creative Commons Attribution Share Alike license.

```
This work is licensed under the Creative Commons Attribution-Share Alike 3.0
Unported License. To view a copy of this license, visit
http://creativecommons.org/licenses/by-sa/3.0/ or send a letter to Creative
Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.
```

The full license can be viewed at: http://creativecommons.org/licenses/by-sa/3.0/

8.2. Software Licensing

The Digital Thermostat Module firmware is Copyright © 2010 by Nick Viera, and the Electronic Access Module firmware is Copyright © 2010 by Chris Miller. THAT Control Software is Copyright © 2010 by Nick Viera and Chris Miller. The Python and C source code are released as open source software under the terms and conditions of the GNU Public License (GPL), version 3.

```
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/>
```

The full license can be viewed at: http://www.gnu.org/licenses/gpl.html

8.3. Brand Licensing

THAT Home Automation Topology and the THAT "house" logo (as seen on the front page of this report) are trademarked (™) 2010 by Nick Viera and Chris Miller.

Derivative products and services created by 3rd parties must not use the logo or give any indication of being officially associated with THAT Home Automation Topology without explicit permission from Nick Viera or Chris Miller.

However, derivative works are allowed and encouraged to call themselves "THAT-compatible," "THAT-compliant," "For use with THAT Home Automation Topology," or "For use with THAT System" to advertise their interoperability with THAT Home Automation Topology.

9. Conclusion

The development of THAT System has been an in-depth project with many parallel work flows. The developers began with the idea for a new home automation system and from that idea worked to define a set of hardware, software, and communication protocols with which to implement THAT.

The developers have successfully designed and implemented both a prototype power supply and a network interface, which are the most basic building blocks of any THAT module. These two components' completion will allow for more rapid prototyping of subsequent THAT modules in the future. Future work for THAT Power Supply and THAT Network Interface should include optimization of the designs to increase efficiency and reduce physical size, amongst other things.

In addition, the developers have made substantial progress towards implementing a Digital Thermostat Module and Electronic Access Module for THAT. These two modules are perhaps the most complex of the modules currently envisioned for THAT, and their successful prototyping implies that the design and implementation of other modules will be both possible and less difficult. Future work for these modules mainly includes further development of their firmware as well as hardware optimizations.

Lastly, the developers made substantial progress towards the development of THAT Control Software. The software currently implements all the necessary core functionality as a proof of concept and is able to communicate with the Digital Thermostat Module. Future work includes development of additional features as well as optimization of the Python code to increase performance and security of the software.

9.1. Contact Information

The developers can be contacted through the current THAT Home Automation Topology website, which is located at: http://www.tehhouse.us/electronics/that/

The developers' current e-mail addresses are as follows:
Nick Viera → nick @ driveev.com
Chris Miller → sparticle @ gmail.com

10. References

[1] D. Dwelley & J. Herbold, “Banish Those Wall Warts With Power Over Ethernet,” electronic design, Oct. 2003. [Online]. Available: http://electronicdesign.com/article/power/page/page/1/banish-those-wall-warts-with-power-over-ethernet59.aspx

[2] D. Sherman, “AN452 One mile long I2C communication using the P82B715.” Datasheet & Application Note Database, May 2010. [Online]. Available: http://www.datasheetarchive.com/pdf/Datasheet-054/DSA004594.pdf

[3] G. Socher, “An AVR microcontroller based Ethernet device,” June 2006. [Online]. Available: http://www.tuxgraphics.org/electronics/200606/article06061.shtml

[4] G. Socher, “HTTP/TCP with an atmega88 microcontroller,” Nov. 2006. [Online]. Available: http://www.tuxgraphics.org/electronics/200611/embedded-webserver.shtml

Appendix A: Glossary

Cascading Style Sheet (CSS)

A style sheet language used to describe the presentation semantics (that is, the look and formatting) of a document written in a markup language. Its most common application is to style web pages written in HTML and XHTML, but the language can also be applied to any kind of XML document, including SVG and XUL.

Extensible Markup Language (XML)

A set of rules for encoding documents electronically. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications. XML's design goals emphasize simplicity, generality, and usability over the Internet. It is a textual data format, with strong support via Unicode for the languages of the world. Although XML's design focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.

Firmware

Low-level machine code (usually written in C or Assembly) which is optimized to run on a microcontroller or similar embedded hardware. This is distinct from software, which usually describes higher-level code that is interpreted and translated before being run on hardware.

HVAC

Heating, Ventilation, and Air Conditioning. Standard residential HVAC systems use a simple control scheme. In this scheme, various functions of the HVAC unit are triggered by a thermostat which applies 24 Volt AC signals to the correct control wires. For a standard, split-unit residential system, there are 4 to 6 defined wires as follow:

Red (R):	24VAC signal supply from the HVAC unit
Green (G):	Blower Fan enable
Yellow (Y):	Air Conditioner enable
White (W / W1):	Furnace enable
---	---
Blue (B) / Black (X):	24VAC return or 2nd Stage cooling (non-standard)
Orange (O):	Heat pump changeover enable

Hyper Text Markup Language (HTML)

The predominant markup language for web pages. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes, and other items. It allows images and objects to be embedded and can be used to create interactive forms. It is written in the form of HTML elements consisting of "tags" surrounded by angle brackets within the web page content.

Internet Protocol (IP)

The primary protocol in the Internet Layer of the Internet Protocol Suite and has the task of delivering distinguished protocol datagrams (packets) from the source host to the destination host solely based on their addresses. For this purpose the Internet Protocol defines addressing methods and structures for datagram encapsulation.

Power over Ethernet (PoE)

An extension to the IEEE 802.3 Ethernet standard which defines ways to safely transmit moderate amounts of power along with data over the same cabling to Ethernet devices. PoE requires category 5 cable or higher for high power levels, but can operate with category 3 cable for low power levels. Power can come from a power supply within a PoE-enabled networking device such as an Ethernet switch or from a device built for "injecting" power onto the Ethernet cabling, dubbed Midspan.

Transmission Control Protocol (TCP)

One of the core protocols of the Internet Protocol Suite. TCP operates at a high level, concerned only with the two end systems, for example a Web browser and a Web server. In particular, TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. Besides the Web, other common applications of TCP include e-mail and file transfer. Among other management tasks, TCP controls segment size, flow control, and data exchange rate.

User Datagram Protocol (UDP)

One of the core members of the Internet Protocol Suite. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths. UDP uses a simple transmission model without implicit hand-shaking dialogues for guaranteeing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice.

Appendix B: THAT Communications Command Set

@ACK - Acknowledge proper receiving of a write command

Generic form: @ACK$variablename#data
Specific form: @ACK$101#43
Expected response: NONE

@DAT - Begins a packet of data

Generic form: @DAT$variablename#data ...
Specific form: @DAT$101#43 ...
Expected response: NONE

@ALL - Read all available data from a device

Generic form: @ALL
Specific form: @ALL
Expected response: @DAT$100#1$101#34$102#56$103#0 ...

@GET - Read one piece of data from a device

Generic form: @GET$variablename
Specific form: @GET$102
Expected response: @DAT$102#56

@SET - Write one piece of data to a device

Generic form: @SET$variablename#data
Specific form: @SET$102#54
Expected response: @ACK$102#54

@IDD - Get module ID information from a device

Generic form: @IDD
Specific form: @IDD
Expected response: @DAT$NAME#Copta$ID#123d-42ab$MAN#DEV, inc.

@KEY - Setup trusted hash key for a device using the device serial number (which the device will never send via TCP/IP. The serial number must be entered into the client application.)

Generic form: @KEY$HASH#random_hash
Specific form: @KEY$HASH#12A435BE451FEC
Expected response: @DAT$HASH#345ABCD18956DE

Appendix C: Digital Thermostat Firmware Source Code

basic_adjust.c

```
// ===================================================================================
// THAT Home Automation Topology -- Digital Thermostat Module
// This file is part of Firmware version 0.1.8 -- 2010.05.01
// Copyright (C) Nick Viera  ( http://www.nickviera.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.

   You should have received a copy of the GNU General Public License
   along with this program. If not, see <http://www.gnu.org/licenses/> */
// ===================================================================================

    #include "main.h"
    #include "i2c.h"
    #include "lcd.h"
    #include "screen.h"
    #include "basic_adjust.h"

void sys_change(void)
{
    switch(system_mode)
    {
        case SYS_OFF:
            system_mode = SYS_AUTO;
            break;

        case SYS_AUTO:
            system_mode = SYS_HEAT;
            break;

        case SYS_HEAT:
            system_mode = SYS_COOL;
            break;

        case SYS_COOL:
            system_mode = SYS_OFF;
            break;

        default:
            system_mode = SYS_OFF;
            break;
    }

    redraw_screen();
    return;
}
```

```
void fan_switch_on(void)
{
    RELAY_FAN_ON;
    LED_GREEN_ON;
    return;
}

void fan_switch_off(void)
{
    RELAY_FAN_OFF;
    LED_GREEN_OFF;
    return;
}

void fan_switch_autooff(void)
{
    if (fan_mode == FAN_AUTO)
    {
        fan_switch_off();
    }
    return;
}

void fan_change(void)
{
    switch(fan_mode)
    {
        case FAN_AUTO:
            fan_mode = FAN_CIRC;
            fan_switch_off();
            break;
        case FAN_CIRC:
            fan_mode = FAN_ON;
            fan_switch_on();
            break;
        case FAN_ON:
            fan_mode = FAN_AUTO;
            fan_switch_off();
            break;
    }

    print_fan();
    return;
}

// direction:  0 = down, 1 = up
void temperature_change(unsigned char direction)
{
    unsigned char limit    = 0;
    signed int    temp_old = 0;

    temp_old = temperature_set;

    switch (temperature_lock)
    {

        case (TEMPERATURE_UNLOCKED):
            if   (direction == 0) temperature_set -= 5;
            else                  temperature_set += 5;
            break;
```

```
        case (TEMPERATURE_RESTRICT):
            if (direction == 0)
            {
                limit = temperature_base - temperature_variance;
                temperature_set -= 5;
                if (temperature_set < limit) temperature_set = limit;
            }

            else
            {
                limit = temperature_base + temperature_variance;
                temperature_set += 5;
                if (temperature_set > limit) temperature_set = limit;
            }
            break;

        case (TEMPERATURE_LOCKED):
            break;
    }

    if      (temperature_set < TEMPERATURE_MIN)
             temperature_set = TEMPERATURE_MIN;
    else if (temperature_set > TEMPERATURE_MAX)
             temperature_set = TEMPERATURE_MAX;

    if (temp_old != temperature_set)
    {
        if (pgm_mode == MANUAL)
        {
            temperature_override = 0;
            print_setpoint();
        }
        else
        {
            temperature_override = 1;
            print_setpoint();
        }
    }
    return;
}
```

basic_adjust.h

```
#ifndef BASIC_H
#define BASIC_H

void sys_change         (void);
void fan_change         (void);
void temperature_change (unsigned char direction);
void fan_switch_autooff (void);
void fan_switch_on      (void);

#endif
```

```
#include <avr/io.h>
#include <avr/eeprom.h>
#include "main.h"
#include "i2c.h"

const unsigned char EEMEM eereg10to19[7]   = {192,168,7,222,84,28,0};
const unsigned char EEMEM eereg20to29[7]   = {200,248,0,4,127,2,4};
const signed   char EEMEM eereg30to39[5]   = {0,4,0,0};
const unsigned char EEMEM eereg100to109[3] = {0,0,0};
const unsigned int  EEMEM eereg110to119[3] = {0,250,250};
const unsigned char EEMEM eetime[5]        = {6,30,4,10};

void write_settings(unsigned char group)
{
    unsigned char i=0;

    switch(group)
    {

        case 10:
            for(i=0; i<7; i++)
                eeprom_write_byte((void *)&eereg10to19[i], reg10to19[0][i]);
            break;

        case 20:
            for(i=0; i<6; i++)
                eeprom_write_byte((void *)&eereg20to29[i], reg20to29[0][i]);
            break;

        case 30:
            for(i=0; i<4; i++)
                eeprom_write_byte((void *)&eereg30to39[i], reg30to39[0][i]);
            break;
    }
    return;
}

void ee_datetime(unsigned char rw)
{
    unsigned char i=0;
    for (i=0; i<3; i++)
    {
        if (rw == 1)
            eeprom_write_byte((void *)&eetime[i], reg90to99[0][i+3]);
        else
            reg90to99[0][i+3] = eeprom_read_byte((void *)&eetime[i]);
    }
    return;
}

void read_settings(unsigned char group)
{
    unsigned char i=0;

    switch(group)
    {
        case 10:
            for(i=0; i<6; i++)
                reg10to19[0][i] = eeprom_read_byte((void *)&eereg10to19[i]);
```

```
            break;

        case 20:
            for(i=0; i<6; i++)
                reg20to29[0][i] = eeprom_read_byte((void *)&eereg20to29[i]);
            break;

        case 30:
            for(i=0; i<4; i++)
                reg30to39[0][i] = eeprom_read_byte((void *)&eereg30to39[i]);
            break;
    }
    return;
}

void read_settings_all(void)
{
    read_settings(20);
    read_settings(30);
    return;
}

void write_settings_all(void)
{
    write_settings(20);
    write_settings(30);
    return;
}
```

eemem.h

```
#ifndef EEMEM_H
#define EEMEM_H

void write_settings     (unsigned char group);
void read_settings      (unsigned char group);
void ee_datetime        (unsigned char rw);
void write_settings_all (void);
void read_settings_all  (void);

#endif
```

```
/*********************************************
 * Author: Guido Socher
 * Copyright: GPL V2
 * http://www.gnu.org/licenses/gpl.html
 *
 * Based on the enc28j60.c file from the AVRlib library by Pascal Stang.
 * For AVRlib See http://www.procyonengineering.com/
 * Used with explicit permission of Pascal Stang.
 *
 * Title: Microchip ENC28J60 Ethernet Interface Driver
 * Chip type : ATMEGA88 with ENC28J60
 *********************************************/
// Includes modifications made by Nick Viera, April 2010.

#include <avr/io.h>
#include "enc28j60.h"
#include "main.h"
#define F_CPU 12500000UL  // 12.5 MHz

#ifndef ALIBC_OLD
#include <util/delay_basic.h>
#else
#include <avr/delay.h>
#endif

static uint8_t Enc28j60Bank;
static int16_t gNextPacketPtr;

#define ENC28J60_CONTROL_PORT   PORTB
#define ENC28J60_CONTROL_DDR    DDRB

#define ENC28J60_CONTROL_SO     PORTB6
#define ENC28J60_CONTROL_SI     PORTB5
#define ENC28J60_CONTROL_SCK    PORTB7

// set CS to 0 = active, or set CS to 1 = passive
#define CSOUTPUT  bit_set(DDRB,3)
#define CSACTIVE  bit_clr(PORTB,3)
#define CSPASSIVE bit_set(PORTB,3)
#define waitspi() while(!(SPSR&(1<<SPIF)))

uint8_t enc28j60ReadOp (uint8_t op, uint8_t address)
{
        CSACTIVE;
        // issue read command
        SPDR = op | (address & ADDR_MASK);
        waitspi();
        // read data
        SPDR = 0x00;
        waitspi();
        // do dummy read if needed (for mac and mii, see datasheet page 29)
        if(address & 0x80)
        {
                SPDR = 0x00;
                waitspi();
        }
        // release CS
        CSPASSIVE;
        return(SPDR);
```

```
}

void enc28j60WriteOp(uint8_t op, uint8_t address, uint8_t data)
{
        CSACTIVE;
        SPDR = op | (address & ADDR_MASK); // issue write command
        waitspi();
        SPDR = data;  // write data
        waitspi();
        CSPASSIVE;
}

void enc28j60ReadBuffer(uint16_t len, uint8_t* data)
{
        CSACTIVE;
        // issue read command
        SPDR = ENC28J60_READ_BUF_MEM;
        waitspi();
        while (len)
        {
                len--;
                SPDR = 0x00; // read data
                waitspi();
                *data = SPDR;
                data++;
        }
        *data='\0';
        CSPASSIVE;
}

void enc28j60WriteBuffer(uint16_t len, uint8_t* data)
{
        CSACTIVE;
        // issue write command
        SPDR = ENC28J60_WRITE_BUF_MEM;
        waitspi();
        while(len)
        {
                len--;
                // write data
                SPDR = *data;
                data++;
                waitspi();
        }
        CSPASSIVE;
}

void enc28j60SetBank(uint8_t address)
{
        // set the bank (if needed)
        if((address & BANK_MASK) != Enc28j60Bank)
        {
                // set the bank
                enc28j60WriteOp(ENC28J60_BIT_FIELD_CLR, ECON1, (ECON1_BSEL1|ECON1_BSEL0));
                enc28j60WriteOp(ENC28J60_BIT_FIELD_SET, ECON1, (address & BANK_MASK)>>5);
                Enc28j60Bank = (address & BANK_MASK);
        }
}

uint8_t enc28j60Read(uint8_t address)
{
```

```
        // set the bank and then do the read
        enc28j60SetBank(address);
        return enc28j60ReadOp(ENC28J60_READ_CTRL_REG, address);
}

// read upper 8 bits
uint16_t enc28j60PhyReadH(uint8_t address)
{

       // Set the right address and start the register read operation
       enc28j60Write(MIREGADR, address);
       enc28j60Write(MICMD, MICMD_MIIRD);
        _delay_loop_1(40); // 10us

       // wait until the PHY read completes
       while(enc28j60Read(MISTAT) & MISTAT_BUSY);

       // reset reading bit
       enc28j60Write(MICMD, 0x00);

       return (enc28j60Read(MIRDH));
}

void enc28j60Write(uint8_t address, uint8_t data)
{
        // set the bank
        enc28j60SetBank(address);
        // do the write
        enc28j60WriteOp(ENC28J60_WRITE_CTRL_REG, address, data);
}

void enc28j60PhyWrite(uint8_t address, uint16_t data)
{
        // set the PHY register address
        enc28j60Write(MIREGADR, address);
        // write the PHY data
        enc28j60Write(MIWRL, data);
        enc28j60Write(MIWRH, data>>8);
        // wait until the PHY write completes
        while(enc28j60Read(MISTAT) & MISTAT_BUSY){
                _delay_loop_1(40); // 10us
        }
}

void enc28j60clkout(uint8_t clk)
{
        //setup clkout: 2 is 12.5MHz:
       enc28j60Write(ECOCON, clk & 0x7);
}

void enc28j60Init(uint8_t* macaddr)
{
    // initialize I/O
    // ss as output:
    //ENC28J60_CONTROL_DDR |= 1<<ENC28J60_CONTROL_CS;
    CSOUTPUT;
    CSPASSIVE; // ss=0

    ENC28J60_CONTROL_DDR  |= 1<<ENC28J60_CONTROL_SI | 1<<ENC28J60_CONTROL_SCK; //mosi,sckout
    ENC28J60_CONTROL_DDR  |= 1<<ENC28J60_CONTROL_SO; // MISO is input
    ENC28J60_CONTROL_PORT|= 1<<ENC28J60_CONTROL_SI; // MOSI low
```

```
ENC28J60_CONTROL_PORT|= 1<<ENC28J60_CONTROL_SCK; // SCK low

// initialize SPI interface (ATmega48/88/168/328)
// master mode and Fosc/2 clock:
//   SPCR = (1<<SPE)|(1<<MSTR);
//   SPSR |= (1<<SPI2X);

// initialize SPI interface (ATmega324/644/1284)
    SPCR = 0b01010000;  // SPI enable, master mode

// perform system reset
   enc28j60WriteOp(ENC28J60_SOFT_RESET, 0, ENC28J60_SOFT_RESET);
    _delay_loop_2(0); // 20ms

   // check CLKRDY bit to see if reset is complete
// The CLKRDY does not work. See Rev. B4 Silicon Errata point. Just wait.

   //while(!(enc28j60Read(ESTAT) & ESTAT_CLKRDY));
   // do bank 0 stuff
   // initialize receive buffer
   // 16-bit transfers, must write low byte first
   // set receive buffer start address
   gNextPacketPtr = RXSTART_INIT;
    // Rx start
   enc28j60Write(ERXSTL, RXSTART_INIT&0xFF);
   enc28j60Write(ERXSTH, RXSTART_INIT>>8);
   // set receive pointer address
   enc28j60Write(ERXRDPTL, RXSTART_INIT&0xFF);
   enc28j60Write(ERXRDPTH, RXSTART_INIT>>8);
   // RX end
   enc28j60Write(ERXNDL, RXSTOP_INIT&0xFF);
   enc28j60Write(ERXNDH, RXSTOP_INIT>>8);
   // TX start
   enc28j60Write(ETXSTL, TXSTART_INIT&0xFF);
   enc28j60Write(ETXSTH, TXSTART_INIT>>8);
   // TX end
   enc28j60Write(ETXNDL, TXSTOP_INIT&0xFF);
   enc28j60Write(ETXNDH, TXSTOP_INIT>>8);
   // do bank 1 stuff, packet filter:
    // For broadcast packets we allow only ARP packtets
    // All other packets should be unicast only for our mac (MAADR)
    //
    // The pattern to match on is therefore
    // Type     ETH.DST
    // ARP      BROADCAST
    // 06 08 -- ff ff ff ff ff ff -> ip checksum for theses bytes=f7f9
    // in binary these poitions are:11 0000 0011 1111
    // This is hex 303F->EPMM0=0x3f,EPMM1=0x30
   enc28j60Write(ERXFCON, ERXFCON_UCEN|ERXFCON_CRCEN|ERXFCON_PMEN);
   enc28j60Write(EPMM0, 0x3f);
   enc28j60Write(EPMM1, 0x30);
   enc28j60Write(EPMCSL, 0xf9);
   enc28j60Write(EPMCSH, 0xf7);
    //
    //
   // do bank 2 stuff
   // enable MAC receive
   enc28j60Write(MACON1, MACON1_MARXEN|MACON1_TXPAUS|MACON1_RXPAUS);
   // bring MAC out of reset
   enc28j60Write(MACON2, 0x00);
   // enable automatic padding to 60bytes and CRC operations
```

```
	enc28j60WriteOp(ENC28J60_BIT_FIELD_SET, MACON3, MACON3_PADCFG0|MACON3_TXCRCEN|
MACON3_FRMLNEN);
	// set inter-frame gap (non-back-to-back)
	enc28j60Write(MAIPGL, 0x12);
	enc28j60Write(MAIPGH, 0x0C);
	// set inter-frame gap (back-to-back)
	enc28j60Write(MABBIPG, 0x12);
	// Set the maximum packet size which the controller will accept
	  // Do not send packets longer than MAX_FRAMELEN:
	enc28j60Write(MAMXFLL, MAX_FRAMELEN&0xFF);
	enc28j60Write(MAMXFLH, MAX_FRAMELEN>>8);
	// do bank 3 stuff
	  // write MAC address
	  // NOTE: MAC address in ENC28J60 is byte-backward
	  enc28j60Write(MAADR5, macaddr[0]);
	  enc28j60Write(MAADR4, macaddr[1]);
	  enc28j60Write(MAADR3, macaddr[2]);
	  enc28j60Write(MAADR2, macaddr[3]);
	  enc28j60Write(MAADR1, macaddr[4]);
	  enc28j60Write(MAADR0, macaddr[5]);
	// no loopback of transmitted frames
	enc28j60PhyWrite(PHCON2, PHCON2_HDLDIS);
	// switch to bank 0
	enc28j60SetBank(ECON1);
	// enable interrutps
	enc28j60WriteOp(ENC28J60_BIT_FIELD_SET, EIE, EIE_INTIE|EIE_PKTIE);
	// enable packet reception
	enc28j60WriteOp(ENC28J60_BIT_FIELD_SET, ECON1, ECON1_RXEN);
}

// read the revision of the chip:
uint8_t enc28j60getrev(void)
{
	return(enc28j60Read(EREVID));
}

// link status
uint8_t enc28j60linkup(void)
{
	  // bit 10 (= bit 3 in upper reg)
	return(enc28j60PhyReadH(PHSTAT2) && 4);
}

void enc28j60PacketSend(uint16_t len, uint8_t* packet)
{
	  // Check no transmit in progress
	  while (enc28j60ReadOp(ENC28J60_READ_CTRL_REG, ECON1) & ECON1_TXRTS)
	  {
		  // Reset the transmit logic problem. See Rev. B4 Silicon Errata point 12.
		  if( (enc28j60Read(EIR) & EIR_TXERIF) ) {
			  enc28j60WriteOp(ENC28J60_BIT_FIELD_SET, ECON1, ECON1_TXRST);
			  enc28j60WriteOp(ENC28J60_BIT_FIELD_CLR, ECON1, ECON1_TXRST);
		  }
	  }
	// Set the write pointer to start of transmit buffer area
	enc28j60Write(EWRPTL, TXSTART_INIT&0xFF);
	enc28j60Write(EWRPTH, TXSTART_INIT>>8);
	// Set the TXND pointer to correspond to the packet size given
	enc28j60Write(ETXNDL, (TXSTART_INIT+len)&0xFF);
	enc28j60Write(ETXNDH, (TXSTART_INIT+len)>>8);
	// write per-packet control byte (0x00 means use macon3 settings)
```

```
	enc28j60WriteOp(ENC28J60_WRITE_BUF_MEM, 0, 0x00);
	// copy the packet into the transmit buffer
	enc28j60WriteBuffer(len, packet);
	// send the contents of the transmit buffer onto the network
	enc28j60WriteOp(ENC28J60_BIT_FIELD_SET, ECON1, ECON1_TXRTS);
}

// just probe if there might be a packet
uint8_t enc28j60hasRxPkt(void)
{
	if( enc28j60Read(EPKTCNT) ==0 ){
		return(0);
	 }
	 return(1);
}

// Gets a packet from the network receive buffer, if one is available.
// The packet will by headed by an ethernet header.
//      maxlen  The maximum acceptable length of a retrieved packet.
//      packet  Pointer where packet data should be stored.
// Returns: Packet length in bytes if a packet was retrieved, zero otherwise.
uint16_t enc28j60PacketReceive(uint16_t maxlen, uint8_t* packet)
{
	uint16_t rxstat;
	uint16_t len;
	// check if a packet has been received and buffered
	//if( !(enc28j60Read(EIR) & EIR_PKTIF) ){
	 // The above does not work. See Rev. B4 Silicon Errata point 6.
	if( enc28j60Read(EPKTCNT) ==0 ) return(0);

	// Set the read pointer to the start of the received packet
	enc28j60Write(ERDPTL, (gNextPacketPtr &0xFF));
	enc28j60Write(ERDPTH, (gNextPacketPtr)>>8);
	// read the next packet pointer
	gNextPacketPtr  = enc28j60ReadOp(ENC28J60_READ_BUF_MEM, 0);
	gNextPacketPtr |= enc28j60ReadOp(ENC28J60_READ_BUF_MEM, 0)<<8;
	// read the packet length (see datasheet page 43)
	len  = enc28j60ReadOp(ENC28J60_READ_BUF_MEM, 0);
	len |= enc28j60ReadOp(ENC28J60_READ_BUF_MEM, 0)<<8;
	 len-=4; //remove the CRC count
	// read the receive status (see datasheet page 43)
	rxstat  = enc28j60ReadOp(ENC28J60_READ_BUF_MEM, 0);
	rxstat |= ((uint16_t)enc28j60ReadOp(ENC28J60_READ_BUF_MEM, 0))<<8;
	// limit retrieve length
	 if (len>maxlen-1){
		len=maxlen-1;
	 }
	 // check CRC and symbol errors (see datasheet page 44, table 7-3):
	 // The ERXFCON.CRCEN is set by default. Normally we should not
	 // need to check this.
	 if ((rxstat & 0x80)==0){
		// invalid
		len=0;
	 }else{
		// copy the packet from the receive buffer
		enc28j60ReadBuffer(len, packet);
	 }
	// Move the RX read pointer to the start of the next received packet
	// This frees the memory we just read out
	enc28j60Write(ERXRDPTL, (gNextPacketPtr &0xFF));
	enc28j60Write(ERXRDPTH, (gNextPacketPtr)>>8);
```

```
        // Move the RX read pointer to the start of the next received packet
        // This frees the memory we just read out.
        // However, compensate for the errata point 13, rev B4: enver write an even address!
        if ((gNextPacketPtr - 1 < RXSTART_INIT)
                || (gNextPacketPtr -1 > RXSTOP_INIT)) {
                enc28j60Write(ERXRDPTL, (RXSTOP_INIT)&0xFF);
                enc28j60Write(ERXRDPTH, (RXSTOP_INIT)>>8);
        } else {
                enc28j60Write(ERXRDPTL, (gNextPacketPtr-1)&0xFF);
                enc28j60Write(ERXRDPTH, (gNextPacketPtr-1)>>8);
        }
       // decrement the packet counter indicate we are done with this packet
       enc28j60WriteOp(ENC28J60_BIT_FIELD_SET, ECON2, ECON2_PKTDEC);
       return(len);
}
```

enc28j60.h

```
/*****************************************************************************
* vim:sw=8:ts=8:si:et
*
* Title      : Microchip ENC28J60 Ethernet Interface Driver
* Author     : Pascal Stang
* Modified by: Guido Socher
* Copyright: GPL V2
*
*This driver provides initialization and transmit/receive
*functions for the Microchip ENC28J60 10Mb Ethernet Controller and PHY.
*This chip is novel in that it is a full MAC+PHY interface all in a 28-pin
*chip, using an SPI interface to the host processor.
*
*
*****************************************************************************/
//@{

#ifndef ENC28J60_H
#define ENC28J60_H
#include <inttypes.h>

// ENC28J60 Control Registers
// Control register definitions are a combination of address,
// bank number, and Ethernet/MAC/PHY indicator bits.
// - Register address        (bits 0-4)
// - Bank number        (bits 5-6)
// - MAC/PHY indicator        (bit 7)
#define ADDR_MASK        0x1F
#define BANK_MASK        0x60
#define SPRD_MASK        0x80
// All-bank registers
#define EIE              0x1B
#define EIR              0x1C
#define ESTAT            0x1D
#define ECON2            0x1E
#define ECON1            0x1F
// Bank 0 registers
#define ERDPTL           (0x00|0x00)
#define ERDPTH           (0x01|0x00)
#define EWRPTL           (0x02|0x00)
#define EWRPTH           (0x03|0x00)
```

```
#define ETXSTL          (0x04|0x00)
#define ETXSTH          (0x05|0x00)
#define ETXNDL          (0x06|0x00)
#define ETXNDH          (0x07|0x00)
#define ERXSTL          (0x08|0x00)
#define ERXSTH          (0x09|0x00)
#define ERXNDL          (0x0A|0x00)
#define ERXNDH          (0x0B|0x00)
#define ERXRDPTL        (0x0C|0x00)
#define ERXRDPTH        (0x0D|0x00)
#define ERXWRPTL        (0x0E|0x00)
#define ERXWRPTH        (0x0F|0x00)
#define EDMASTL         (0x10|0x00)
#define EDMASTH         (0x11|0x00)
#define EDMANDL         (0x12|0x00)
#define EDMANDH         (0x13|0x00)
#define EDMADSTL        (0x14|0x00)
#define EDMADSTH        (0x15|0x00)
#define EDMACSL         (0x16|0x00)
#define EDMACSH         (0x17|0x00)
// Bank 1 registers
#define EHT0            (0x00|0x20)
#define EHT1            (0x01|0x20)
#define EHT2            (0x02|0x20)
#define EHT3            (0x03|0x20)
#define EHT4            (0x04|0x20)
#define EHT5            (0x05|0x20)
#define EHT6            (0x06|0x20)
#define EHT7            (0x07|0x20)
#define EPMM0           (0x08|0x20)
#define EPMM1           (0x09|0x20)
#define EPMM2           (0x0A|0x20)
#define EPMM3           (0x0B|0x20)
#define EPMM4           (0x0C|0x20)
#define EPMM5           (0x0D|0x20)
#define EPMM6           (0x0E|0x20)
#define EPMM7           (0x0F|0x20)
#define EPMCSL          (0x10|0x20)
#define EPMCSH          (0x11|0x20)
#define EPMOL           (0x14|0x20)
#define EPMOH           (0x15|0x20)
#define EWOLIE          (0x16|0x20)
#define EWOLIR          (0x17|0x20)
#define ERXFCON         (0x18|0x20)
#define EPKTCNT         (0x19|0x20)
// Bank 2 registers
#define MACON1          (0x00|0x40|0x80)
#define MACON2          (0x01|0x40|0x80)
#define MACON3          (0x02|0x40|0x80)
#define MACON4          (0x03|0x40|0x80)
#define MABBIPG         (0x04|0x40|0x80)
#define MAIPGL          (0x06|0x40|0x80)
#define MAIPGH          (0x07|0x40|0x80)
#define MACLCON1        (0x08|0x40|0x80)
#define MACLCON2        (0x09|0x40|0x80)
#define MAMXFLL         (0x0A|0x40|0x80)
#define MAMXFLH         (0x0B|0x40|0x80)
#define MAPHSUP         (0x0D|0x40|0x80)
#define MICON           (0x11|0x40|0x80)
#define MICMD           (0x12|0x40|0x80)
#define MIREGADR        (0x14|0x40|0x80)
```

```
#define MIWRL            (0x16|0x40|0x80)
#define MIWRH            (0x17|0x40|0x80)
#define MIRDL            (0x18|0x40|0x80)
#define MIRDH            (0x19|0x40|0x80)
// Bank 3 registers
#define MAADR1           (0x00|0x60|0x80)
#define MAADR0           (0x01|0x60|0x80)
#define MAADR3           (0x02|0x60|0x80)
#define MAADR2           (0x03|0x60|0x80)
#define MAADR5           (0x04|0x60|0x80)
#define MAADR4           (0x05|0x60|0x80)
#define EBSTSD           (0x06|0x60)
#define EBSTCON          (0x07|0x60)
#define EBSTCSL          (0x08|0x60)
#define EBSTCSH          (0x09|0x60)
#define MISTAT           (0x0A|0x60|0x80)
#define EREVID           (0x12|0x60)
#define ECOCON           (0x15|0x60)
#define EFLOCON          (0x17|0x60)
#define EPAUSL           (0x18|0x60)
#define EPAUSH           (0x19|0x60)
// PHY registers
#define PHCON1           0x00
#define PHSTAT1          0x01
#define PHHID1           0x02
#define PHHID2           0x03
#define PHCON2           0x10
#define PHSTAT2          0x11
#define PHIE             0x12
#define PHIR             0x13
#define PHLCON           0x14

// ENC28J60 ERXFCON Register Bit Definitions
#define ERXFCON_UCEN     0x80
#define ERXFCON_ANDOR    0x40
#define ERXFCON_CRCEN    0x20
#define ERXFCON_PMEN     0x10
#define ERXFCON_MPEN     0x08
#define ERXFCON_HTEN     0x04
#define ERXFCON_MCEN     0x02
#define ERXFCON_BCEN     0x01
// ENC28J60 EIE Register Bit Definitions
#define EIE_INTIE        0x80
#define EIE_PKTIE        0x40
#define EIE_DMAIE        0x20
#define EIE_LINKIE       0x10
#define EIE_TXIE         0x08
#define EIE_WOLIE        0x04
#define EIE_TXERIE       0x02
#define EIE_RXERIE       0x01
// ENC28J60 EIR Register Bit Definitions
#define EIR_PKTIF        0x40
#define EIR_DMAIF        0x20
#define EIR_LINKIF       0x10
#define EIR_TXIF         0x08
#define EIR_WOLIF        0x04
#define EIR_TXERIF       0x02
#define EIR_RXERIF       0x01
// ENC28J60 ESTAT Register Bit Definitions
#define ESTAT_INT        0x80
#define ESTAT_LATECOL    0x10
```

```
#define ESTAT_RXBUSY     0x04
#define ESTAT_TXABRT     0x02
#define ESTAT_CLKRDY     0x01
// ENC28J60 ECON2 Register Bit Definitions
#define ECON2_AUTOINC    0x80
#define ECON2_PKTDEC     0x40
#define ECON2_PWRSV      0x20
#define ECON2_VRPS       0x08
// ENC28J60 ECON1 Register Bit Definitions
#define ECON1_TXRST      0x80
#define ECON1_RXRST      0x40
#define ECON1_DMAST      0x20
#define ECON1_CSUMEN     0x10
#define ECON1_TXRTS      0x08
#define ECON1_RXEN       0x04
#define ECON1_BSEL1      0x02
#define ECON1_BSEL0      0x01
// ENC28J60 MACON1 Register Bit Definitions
#define MACON1_LOOPBK    0x10
#define MACON1_TXPAUS    0x08
#define MACON1_RXPAUS    0x04
#define MACON1_PASSALL   0x02
#define MACON1_MARXEN    0x01
// ENC28J60 MACON2 Register Bit Definitions
#define MACON2_MARST     0x80
#define MACON2_RNDRST    0x40
#define MACON2_MARXRST   0x08
#define MACON2_RFUNRST   0x04
#define MACON2_MATXRST   0x02
#define MACON2_TFUNRST   0x01
// ENC28J60 MACON3 Register Bit Definitions
#define MACON3_PADCFG2   0x80
#define MACON3_PADCFG1   0x40
#define MACON3_PADCFG0   0x20
#define MACON3_TXCRCEN   0x10
#define MACON3_PHDRLEN   0x08
#define MACON3_HFRMLEN   0x04
#define MACON3_FRMLNEN   0x02
#define MACON3_FULDPX    0x01
// ENC28J60 MICMD Register Bit Definitions
#define MICMD_MIISCAN    0x02
#define MICMD_MIIRD      0x01
// ENC28J60 MISTAT Register Bit Definitions
#define MISTAT_NVALID    0x04
#define MISTAT_SCAN      0x02
#define MISTAT_BUSY      0x01
// ENC28J60 PHY PHCON1 Register Bit Definitions
#define PHCON1_PRST      0x8000
#define PHCON1_PLOOPBK   0x4000
#define PHCON1_PPWRSV    0x0800
#define PHCON1_PDPXMD    0x0100
// ENC28J60 PHY PHSTAT1 Register Bit Definitions
#define PHSTAT1_PFDPX    0x1000
#define PHSTAT1_PHDPX    0x0800
#define PHSTAT1_LLSTAT   0x0004
#define PHSTAT1_JBSTAT   0x0002
// ENC28J60 PHY PHCON2 Register Bit Definitions
#define PHCON2_FRCLINK   0x4000
#define PHCON2_TXDIS     0x2000
#define PHCON2_JABBER    0x0400
#define PHCON2_HDLDIS    0x0100
```

```
// ENC28J60 Packet Control Byte Bit Definitions
#define PKTCTRL_PHUGEEN  0x08
#define PKTCTRL_PPADEN   0x04
#define PKTCTRL_PCRCEN   0x02
#define PKTCTRL_POVERRIDE 0x01

// SPI operation codes
#define ENC28J60_READ_CTRL_REG       0x00
#define ENC28J60_READ_BUF_MEM        0x3A
#define ENC28J60_WRITE_CTRL_REG      0x40
#define ENC28J60_WRITE_BUF_MEM       0x7A
#define ENC28J60_BIT_FIELD_SET       0x80
#define ENC28J60_BIT_FIELD_CLR       0xA0
#define ENC28J60_SOFT_RESET          0xFF


// The RXSTART_INIT should be zero. See Rev. B4 Silicon Errata
// buffer boundaries applied to internal 8K ram
// the entire available packet buffer space is allocated
//
// start with recbuf at 0/
#define RXSTART_INIT     0x0
// receive buffer end
#define RXSTOP_INIT      (0x1FFF-0x0600-1)
// start TX buffer at 0x1FFF-0x0600, pace for one full ethernet frame (~1500 bytes)
#define TXSTART_INIT     (0x1FFF-0x0600)
// stp TX buffer at end of mem
#define TXSTOP_INIT      0x1FFF
//
// max frame length which the conroller will accept:
#define        MAX_FRAMELEN        1500        // (note: maximum ethernet frame length would
be 1518)
//#define MAX_FRAMELEN     600


// functions
extern uint8_t  enc28j60ReadOp      (uint8_t op, uint8_t address);
extern void     enc28j60WriteOp     (uint8_t op, uint8_t address, uint8_t data);
extern void     enc28j60ReadBuffer  (uint16_t len, uint8_t* data);
extern void     enc28j60WriteBuffer (uint16_t len, uint8_t* data);
extern void     enc28j60SetBank     (uint8_t address);
extern uint8_t  enc28j60Read        (uint8_t address);
extern void     enc28j60Write       (uint8_t address, uint8_t data);
extern void     enc28j60PhyWrite    (uint8_t address, uint16_t data);
extern void     enc28j60clkout      (uint8_t clk);
extern void     enc28j60Init        (uint8_t* macaddr);
extern void     enc28j60PacketSend  (uint16_t len, uint8_t* packet);
extern uint8_t  enc28j60hasRxPkt    (void);
extern uint16_t enc28j60PacketReceive(uint16_t maxlen, uint8_t* packet);
extern uint8_t  enc28j60getrev      (void);
extern uint8_t  enc28j60linkup      (void);

#endif
//@}
```

```
// =============================================================================
// THAT Home Automation Topology -- Digital Thermostat Module
// This file is part of Firmware version 0.1.8 -- 2010.05.01
// Copyright (C) Nick Viera  ( http://www.nickviera.com/ )
// License: GNU General Public License, version 3
// =============================================================================

#ifndef FONT58_H
#define FONT58_H

const prog_uint8_t font_5x7_data[] = {
                        0x00, 0x00, 0x00, 0x00, 0x00, // SPACE
                        0x00, 0x00, 0x5F, 0x00, 0x00, // !
                        0x00, 0x03, 0x00, 0x03, 0x00, // "
                        0x14, 0x3E, 0x14, 0x3E, 0x14, // #
                        0x24, 0x2A, 0x7F, 0x2A, 0x12, // $
                        0x43, 0x33, 0x08, 0x66, 0x61, // %
                        0x36, 0x49, 0x55, 0x22, 0x50, // &
                        0x00, 0x05, 0x03, 0x00, 0x00, // '
                        0x00, 0x1C, 0x22, 0x41, 0x00, // (
                        0x00, 0x41, 0x22, 0x1C, 0x00, // )
                        0x14, 0x08, 0x3E, 0x08, 0x14, // *
                        0x08, 0x08, 0x3E, 0x08, 0x08, // +
                        0x00, 0x50, 0x30, 0x00, 0x00, // ,
                        0x08, 0x08, 0x08, 0x08, 0x08, // -
                        0x00, 0x60, 0x60, 0x00, 0x00, // .
                        0x20, 0x10, 0x08, 0x04, 0x02, // /
                        0x3E, 0x51, 0x49, 0x45, 0x3E, // 0
                        0x00, 0x04, 0x02, 0x7F, 0x00, // 1
                        0x42, 0x61, 0x51, 0x49, 0x46, // 2
                        0x22, 0x41, 0x49, 0x49, 0x36, // 3
                        0x18, 0x14, 0x12, 0x7F, 0x10, // 4
                        0x27, 0x45, 0x45, 0x45, 0x39, // 5
                        0x3E, 0x49, 0x49, 0x49, 0x32, // 6
                        0x01, 0x01, 0x71, 0x09, 0x07, // 7
                        0x36, 0x49, 0x49, 0x49, 0x36, // 8
                        0x26, 0x49, 0x49, 0x49, 0x3E, // 9
                        0x00, 0x36, 0x36, 0x00, 0x00, // :
                        0x00, 0x56, 0x36, 0x00, 0x00, // ;
                        0x08, 0x14, 0x22, 0x41, 0x00, // <
                        0x14, 0x14, 0x14, 0x14, 0x14, // =
                        0x00, 0x41, 0x22, 0x14, 0x08, // >
                        0x02, 0x01, 0x51, 0x09, 0x06, // ?
                        0x3E, 0x41, 0x59, 0x55, 0x5E, // @
                        0x7E, 0x09, 0x09, 0x09, 0x7E, // A
                        0x7F, 0x49, 0x49, 0x49, 0x36, // B
                        0x3E, 0x41, 0x41, 0x41, 0x22, // C
                        0x7F, 0x41, 0x41, 0x41, 0x3E, // D
                        0x7F, 0x49, 0x49, 0x49, 0x41, // E
                        0x7F, 0x09, 0x09, 0x09, 0x01, // F
                        0x3E, 0x41, 0x41, 0x49, 0x3A, // G
                        0x7F, 0x08, 0x08, 0x08, 0x7F, // H
                        0x00, 0x41, 0x7F, 0x41, 0x00, // I
                        0x30, 0x40, 0x40, 0x40, 0x3F, // J
                        0x7F, 0x08, 0x14, 0x22, 0x41, // K
                        0x7F, 0x40, 0x40, 0x40, 0x40, // L
                        0x7F, 0x02, 0x0C, 0x02, 0x7F, // M
                        0x7F, 0x02, 0x04, 0x08, 0x7F, // N
                        0x3E, 0x41, 0x41, 0x41, 0x3E, // O
```

```
                            0x7F, 0x09, 0x09, 0x09, 0x06, // P
                            0x1E, 0x21, 0x21, 0x21, 0x5E, // Q
                            0x7F, 0x09, 0x09, 0x09, 0x76, // R
                            0x26, 0x49, 0x49, 0x49, 0x32, // S
                            0x01, 0x01, 0x7F, 0x01, 0x01, // T
                            0x3F, 0x40, 0x40, 0x40, 0x3F, // U
                            0x1F, 0x20, 0x40, 0x20, 0x1F, // V
                            0x7F, 0x20, 0x10, 0x20, 0x7F, // W
                            0x41, 0x22, 0x1C, 0x22, 0x41, // X
                            0x07, 0x08, 0x70, 0x08, 0x07, // Y
                            0x61, 0x51, 0x49, 0x45, 0x43, // Z
                            0x00, 0x7F, 0x41, 0x00, 0x00, // [
                            0x02, 0x04, 0x08, 0x10, 0x20, // slash
                            0x00, 0x00, 0x41, 0x7F, 0x00, // ]
                            0x04, 0x02, 0x01, 0x02, 0x04, // ^
                            0x40, 0x40, 0x40, 0x40, 0x40, // _
                            0x00, 0x01, 0x02, 0x04, 0x00, // `
                            0x20, 0x54, 0x54, 0x54, 0x78, // a
                            0x7F, 0x44, 0x44, 0x44, 0x38, // b
                            0x38, 0x44, 0x44, 0x44, 0x44, // c
                            0x38, 0x44, 0x44, 0x44, 0x7F, // d
                            0x38, 0x54, 0x54, 0x54, 0x18, // e
                            0x04, 0x04, 0x7E, 0x05, 0x05, // f
                            0x08, 0x54, 0x54, 0x54, 0x3C, // g
                            0x7F, 0x08, 0x04, 0x04, 0x78, // h
                            0x00, 0x44, 0x7D, 0x40, 0x00, // i
                            0x20, 0x40, 0x44, 0x3D, 0x00, // j
                            0x7F, 0x10, 0x28, 0x44, 0x00, // k
                            0x00, 0x41, 0x7F, 0x40, 0x00, // l
                            0x7C, 0x04, 0x78, 0x04, 0x78, // m
                            0x7C, 0x08, 0x04, 0x04, 0x78, // n
                            0x38, 0x44, 0x44, 0x44, 0x38, // o
                            0x7C, 0x14, 0x14, 0x14, 0x08, // p
                            0x08, 0x14, 0x14, 0x14, 0x7C, // q
                            0x00, 0x7C, 0x08, 0x04, 0x04, // r
                            0x48, 0x54, 0x54, 0x54, 0x20, // s
                            0x04, 0x04, 0x3F, 0x44, 0x44, // t
                            0x3C, 0x40, 0x40, 0x20, 0x7C, // u
                            0x1C, 0x20, 0x40, 0x20, 0x1C, // v
                            0x3C, 0x40, 0x30, 0x40, 0x3C, // w
                            0x44, 0x28, 0x10, 0x28, 0x44, // x
                            0x0C, 0x50, 0x50, 0x50, 0x3C, // y
                            0x44, 0x64, 0x54, 0x4C, 0x44, // z
                            0x00, 0x08, 0x36, 0x41, 0x41, // {
                            0x00, 0x00, 0x7F, 0x00, 0x00, // |
                            0x41, 0x41, 0x36, 0x08, 0x00, // }
                            0x02, 0x01, 0x02, 0x04, 0x02, // ~
                            // Special characters starting at 127
                            6, 9, 9, 6, 0,                // degree
                            8, 28, 62, 127, 0,            // <|
                            0, 127, 62, 28, 8             // |>
};

const prog_uint8_t big_font[] =
{
    // 0 (48)
    240, 252, 254, 30, 7, 7, 7, 7, 30, 254, 252, 240,
    15, 63, 127, 120, 224, 224, 224, 224, 120, 127, 63, 15,
    //1 (49)
    0, 0, 4, 6, 7, 255, 255, 255, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 255, 255, 255, 0, 0, 0, 0,
```

```
    //2 (50)
    0, 24, 30, 30, 15, 7, 7, 135, 207, 254, 254, 60,
    0, 192, 224, 240, 248, 252, 255, 239, 231, 227, 224, 224,
    //3 (51)
    0, 0, 12, 14, 15, 7, 199, 199, 207, 255, 126, 60,
    0, 24, 120, 120, 240, 224, 225, 225, 243, 127, 127, 30,
    //4 (52)
    0, 0, 128, 192, 240, 56, 14, 255, 255, 255, 0, 0,
    28, 30, 31, 29, 28, 28, 28, 255, 255, 255, 28, 28,
    //5 (53)
    0, 0, 254, 255, 255, 231, 231, 231, 231, 199, 199, 7,
    0, 112, 113, 224, 224, 224, 224, 224, 113, 127, 63, 31,
    //6 (54)
    0, 0, 192, 240, 248, 254, 223, 199, 195, 129, 128, 0,
    0, 31, 127, 127, 243, 225, 225, 225, 243, 127, 63, 30,
    //7 (55)
    0, 7, 7, 7, 7, 7, 199, 247, 255, 63, 15, 3,
    0, 0, 192, 240, 252, 127, 31, 7, 1, 0, 0, 0,
    //8 (56)
    0, 0, 60, 254, 255, 199, 199, 199, 255, 254, 60, 0,
    0, 62, 127, 127, 243, 225, 225, 225, 243, 127, 127, 62,
    //9 (57)
    0, 120, 252, 254, 207, 135, 135, 135, 207, 254, 252, 248,
    0, 0, 1, 129, 195, 227, 251, 127, 31, 15, 3, 0,
    //C (58)
    224, 248, 252, 62, 14, 15, 7, 7, 7, 7, 14, 14,
    7, 31, 63, 124, 112, 240, 224, 224, 224, 224, 112, 120,
    //F (59)
    0, 0, 255, 255, 255, 199, 199, 199, 199, 199, 199, 0,
    0, 0, 255, 255, 255, 1, 1, 1, 1, 1, 1, 0,
    //H (60)
    255, 255, 255, 192, 192, 192, 192, 192, 192, 255, 255, 255,
    255, 255, 255, 1, 1, 1, 1, 1, 1, 255, 255, 255,
    //R (61)
    255, 255, 255, 135, 135, 135, 135, 207, 254, 254, 56, 0,
    255, 255, 255, 3, 7, 15, 61, 121, 240, 224, 192, 128,
    //% (62)
    60, 126, 231, 231, 126, 188, 192, 96, 56, 28, 6, 3,
    192, 112, 56, 12, 6, 3, 61, 126, 231, 231, 126, 60,
    // degree (63)
    0, 0, 0, 60, 126, 231, 231, 231, 126, 60, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    // ' '
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

#endif
```

```
    #include <avr/io.h>
    #include "main.h"
    #include "i2c.h"
    #include "screen.h"
    #include "lcd.h"

volatile unsigned char i2c_port;

void i2c_init(void)
{
    // Two Wire Serial (i2c) Bus Setup -- Clocked @ ~100kHz
    TWBR = I2C_BITRATE; // bitrate setting
    TWCR = 0b00000100;  // bit 7    : i2c interrupt flag
                        // bit 6    : i2c enable ack bit
                        // bit 5    : i2c start condition bit
                        // bit 4    : i2c stop condition bit
                        // bit 3    : i2c write collision flag
                        // bit 2    : i2c enabled
                        // bit 1    : Reserved
                        // bit 0    : i2c interrupt enable bit
    TWSR = 0b00000000;  // bit 7-3  : i2c Status bits
                        // bit 2    : Reserved
                        // bit 1,0  : Bit rate prescaler (scale=0)
    // Apply prescaler
    TWSR |= (I2C_PRESCALE & 0b00000011);
    return;
}

// ================ i2c start function ================
// This function sends a START data byte over the i2c bus
void i2c_start(void)
{
    // Send an i2c START condition then wait
    TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN)|(0<<TWSTO);
    while (!(TWCR & (1<<TWINT)));
    return;
}

// ================ i2c Error function ================
// This function is passed the expected i2c status code as well
// as the actual current status code, and prints them to the
// serial terminal with an error message.
void i2c_error(unsigned char i2c_code, unsigned char exp_code)
{
    // Display the error on the LEDs ignoring the 3 LSBs
    lcd_set_column(0);
    lcd_set_rowpage(8);
    draw_string("i2c: ");
    print_number(0, i2c_code);
    draw_string(" Exp: ");
    print_number(0, exp_code);
    return;
}

// ================ i2c Check status function ================
// This function reads in the current status of the i2c bus from
// TWSR and checks it against the expected state of the bus as
// defined by "exp_code". If the code/states don't match
// i2c_error is called.
```

```
unsigned char i2c_check_status(unsigned char exp_code)
{
    unsigned char i2c_status=0x00;

    // Read the current i2c bus status and mask off bits 2-0.
    // Call the error routine if code doesn't match what
    // we were expecting.
    i2c_status  = TWSR;
    i2c_status &= 0xF8;

    // Pass the values of the expected and actual codes to
    // the error logging routine, i2c STOP, and return
    if(i2c_status != exp_code) {
        i2c_error(i2c_status, exp_code);
        I2C_DISABLE();
        return(1);
    }
    return(0);
}

unsigned char i2c_data_out(unsigned char data, unsigned char status)
{
    unsigned char fail=0;

    TWDR = data;                    // Load data into buffer
    TWCR = (1<<TWINT) | (1<<TWEN);  // Start TX of address
    while (!(TWCR & (1<<TWINT)));   // Wait for TX to finish

    // Check for errors, return 1 if error
    fail = i2c_check_status(status);
    return(fail);
}

// acknack: 1 = ack, 0 = nack
signed int i2c_data_in(unsigned char acknack)
{
    unsigned char fail=0, data=0;

    // Prepare to read in data, reply with ACK (more data to follow)
    // or with NACK (last data byte to read in)
    if (acknack) TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
    else         TWCR = (1<<TWINT) | (1<<TWEN);

    while (!(TWCR & (1<<TWINT)));
    data = TWDR;

    // Check for errors, abort i2c transfer on error
    // Status: Data was received, ACK was sent or
    //         Data was received, NACK was sent
    if (acknack) fail = i2c_check_status(0x50);
    else         fail = i2c_check_status(0x58);

    if (fail) return(-1);
    return(data);
}

// ================ i2c Transmit data byte function ================
// This function places the AVR in Master transmitter mode. It sends
// the data byte "datain" to the i2c device with bus address "addressin".
void i2c_send_byte(unsigned char addressin, unsigned char datain1)
{
```

```
    unsigned char fail=0;
    I2C_ENABLE();
    i2c_start();

    fail = i2c_check_status(0x08);
    if (fail) return;

    // Write out device address
    fail = i2c_data_out((addressin | 0x00), 0x18);
    if (fail) return;

    // Write out data
    fail = i2c_data_out(datain1, 0x18);
    if (fail) return;

    I2C_STOP();
    I2C_DISABLE();
    return;
}

void i2c_send_block(unsigned char device, unsigned char reg, unsigned char data)
{
    unsigned char fail=0;
    I2C_ENABLE();
    i2c_start();

    fail = i2c_check_status(0x08);
    if (fail) return;

    // Write out device address
    fail = i2c_data_out(device, 0x18);
    if (fail) return;

    // Write out memory register address
    fail = i2c_data_out(reg, 0x28);
    if (fail) return;

    // Write out data
    fail = i2c_data_out(data, 0x28);
    if (fail) return;

    I2C_STOP();
    I2C_DISABLE();
    return;
}

signed int i2c_read_byte(unsigned char device)
{
    unsigned char fail=0, data=0;

    I2C_ENABLE();
    i2c_start();

    fail = i2c_check_status(0x08);  // works correctly
    if (fail) return(2);

    // Write out device address (SLA+R read mode)
    fail = i2c_data_out((device | 0x01), 0x40);
    if (fail) return(6);

    data = i2c_data_in(0);
```

```
    I2C_STOP();
    I2C_DISABLE();
    return(data);
}


signed int i2c_write_then_read(unsigned char device, unsigned char reg)
{
    unsigned char fail=0, data=0;

    I2C_ENABLE();
    i2c_start();

    fail = i2c_check_status(0x08);  // works correctly
    if (fail) return(2);

    // Write out device address (SLA+W write mode)
    fail = i2c_data_out(device, 0x18);
    if (fail) return(3);

    // Write out memory register address
    fail = i2c_data_out(reg, 0x28);
    if (fail) return(4);

    // Generate a repeated start condition
    i2c_start();

    // Check for errors, abort i2c transfer on error
    // Status: Repeated Start was transmitted
    fail = i2c_check_status(0x10);
    if (fail) return(5);

    // Write out device address (SLA+R read mode)
    fail = i2c_data_out((device | 0x01), 0x40);
    if (fail) return(6);

    data = i2c_data_in(0);
    I2C_STOP();
    I2C_DISABLE();
    return(data);
}

void ioexp_port_write(unsigned char portdata)
{
    // Set GPIO = Port output setting register
    i2c_send_block(ADDR_IOEXP, 0x09, portdata);
    return;
}

void ioexp_pin_write(unsigned char bitnum, unsigned char level)
{
    // Read in existing port value
    //val = i2c_write_then_read(ADDR_IOEXP);
    if (level == 1) bit_set(i2c_port, bitnum);
    else            bit_clr(i2c_port, bitnum);

    ioexp_port_write(i2c_port);
    return;
}
```

```
void ioexp_init(void)
{
    // I2C Address for MCP23008 Port Expander is 0100nnnx
    // Where nnn = physical address, x = Read(1) or Write(0)

    // Set IODIR = I/O pin direction register
    // Set all outputs
    i2c_send_block(ADDR_IOEXP, 0x00, 0b00000000);

    // Set IPOL = Input polarity register
    // Set all normal
    i2c_send_block(ADDR_IOEXP, 0x01, 0b00000000);

    // Set GPINTEN = Interrupt on change register
    // Set all disabled
    i2c_send_block(ADDR_IOEXP, 0x02, 0b00000000);

    // Set IOCON = Configuration register
    // Set sequential op disabled, INT pin output active high
    i2c_send_block(ADDR_IOEXP, 0x05, 0b00100010);

    // Set GPIO = Port output setting register
    i2c_send_block(ADDR_IOEXP, 0x09, 0b00000000);
    return;
}

void set_time(void)
{
    i2c_send_block(ADDR_RTC, MINUTE,    0b00000000);
    i2c_send_block(ADDR_RTC, HOUR,      0b01100001);
    i2c_send_block(ADDR_RTC, DATE,      0b00110000);
    i2c_send_block(ADDR_RTC, DAYOFWEEK, 0b00000101);
    i2c_send_block(ADDR_RTC, MONTH,     0b00000100);
    return;
}

void init_rtc(void)
{
    // Set Control Register Settings
    // Bit 7   : Enable Oscillator (0)
    // Bit 4-2 : 1kHz square wave on pin INTB (000)
    // Bit 1-0 : Disable alarm interrupts
    i2c_send_block(ADDR_RTC, 0x0E, 0b00000000);

    // Set to 24-hour mode
    //i2c_send_block(ADDR_RTC, 0x02, 0b01000000);
    set_time();
    return;
}

unsigned char get_time(unsigned char mode)
{
    unsigned char data=0,data1=0;

    data  = i2c_write_then_read(ADDR_RTC, mode);
    data1 = (data & 0b00001111);

    switch (mode)
    {
        case MINUTE:
        case SECOND:  data &= 0b01110000; break;
```

```
        case HOUR:    data &= 0b00110000; break;
        case DATE:    data &= 0b00110000; break;
        case MONTH:   data &= 0b00010000; break;
        case YEAR:    data &= 0b11110000; break;
    }

    data1 += ((data >> 4) * 10);
    return(data1);
}

void get_seconds(void)
{
    unsigned char data=0, data1=0;

    data = i2c_write_then_read(ADDR_RTC, SECOND);
    data1  = (data & 0b00001111);
    data  &= 0b01110000;
    data1 += ((data >> 4) * 10);

    lcd_set_column(0);
    lcd_set_rowpage(9);
    print_number(0, data1);
}

unsigned char get_seconds2(void)
{
    unsigned char data=0, data1=0;

    data = i2c_write_then_read(ADDR_RTC, SECOND);
    data1  = (data & 0b00001111);
    data  &= 0b01110000;
    data1 += ((data >> 4) * 10);

    return(data1);
}


void print_time(void)
{
    unsigned char data=0, military=0;

    // Read in months
    data   = i2c_write_then_read(ADDR_RTC, MONTH);
    month  = (data & 0b00001111);
    data  &= 0b00010000;
    month += ((data >> 4) * 10);

    // Read in days
    data  = i2c_write_then_read(ADDR_RTC, DATE);
    date  = (data & 0b00001111);
    data &= 0b00110000;
    date += ((data >> 4) * 10);

    // Read in Hours
    data = i2c_write_then_read(ADDR_RTC, HOUR);
    hour   = (data & 0b00001111);
    if (military == 0) data  &= 0b00010000;
    else               data  &= 0b00110000;
    hour += ((data >> 4) * 10);

    // Read in Minutes
```

```
    data    = i2c_write_then_read(ADDR_RTC, MINUTE);
    minute  = (data & 0b00001111);
    data   &= 0b01110000;
    minute += ((data >> 4) * 10);

    lcd_set_column(0);
    lcd_set_rowpage(15);

    draw_string  ("Fri.  ");
    print_number(2, month);
    draw_char   ('/', 0);
    print_number(2, date);
    draw_char   (' ', 0);
    draw_char   (' ', 0);
    print_number(2, hour);
    draw_char   (':', 0);
    print_number(2, minute);
    draw_string (" PM");
    return;
}
```

```
// ============================================================================
// THAT Home Automation Topology -- Digital Thermostat Module
// This file is part of Firmware version 0.1.8 -- 2010.05.01
// Copyright (C) Nick Viera  ( http://www.nickviera.com/ )
// License: GNU General Public License, version 3
// ============================================================================

#ifndef I2C_H
#define I2C_H

// I2C Macros
#define I2C_DISABLE()   bit_clr(TWCR, TWEN)
#define I2C_ENABLE()    bit_set(TWCR, TWEN)
#define I2C_STOP()      TWCR = (1<<TWINT)|(1<<TWSTO)|(1<<TWEN)|(0<<TWSTA)

// Sets the bus speed to ~100kHz
#define I2C_PRESCALE    0
#define I2C_BITRATE     8

// Base address (LSB is the R/W flag)
#define ADDR_IOEXP      0b01000000
#define ADDR_RTC        0b11010000

// Addresses for the different time info in the RTC
#define SECOND          0x00
#define MINUTE          0x01
#define HOUR            0x02
#define DAYOFWEEK       0x03
#define DATE            0x04
#define MONTH           0x05
#define YEAR            0x06

//extern volatile unsigned char year, month, date, hour, minute;

void ioexp_pin_write    (unsigned char bitnum, unsigned char level);
void ioexp_port_write   (unsigned char portdata);
void i2c_init           (void);
void ioexp_init         (void);
void init_rtc           (void);
void get_seconds        (void);
void print_time         (void);

unsigned char get_time         (unsigned char mode);
unsigned char get_seconds2     (void);
unsigned char get_time_seconds (void);

#endif
```

```
#ifndef ICON_H
#define ICON_H

const prog_uint8_t icons[] = {
    // Network icon
    0, 0, 192, 79, 73, 121, 73, 79, 64, 192, 0, 0,
    47, 41, 41, 41, 47, 32, 32, 47, 41, 41, 41, 47,

    // Lock icon
    0, 192, 64, 126, 67, 65, 65, 67, 126, 64, 192, 0,
    32, 47, 41, 41, 41, 41, 41, 41, 41, 41, 47, 32,

    // Vent icon
    255, 1, 65, 225, 241, 249, 225, 225, 239, 224, 224, 224,
    47, 40, 40, 40, 41, 43, 40, 40, 46, 32, 32, 32,

    // Filter / Exclaim Icon
    0, 0, 0, 28, 126, 255, 255, 126, 28, 0, 0, 0,
    32, 32, 32, 32, 32, 45, 45, 32, 32, 32, 32, 32
};

#endif
```

```
    #include <avr/io.h>
    #include <avr/interrupt.h>
    #include "interrupts.h"
    #include "main.h"
    #include "lcd.h"
    #include "i2c.h"
    #include "screen.h"

volatile unsigned int   adc_result, adc_result_old;
volatile unsigned char  adc_flag, timer_int_hits;
volatile unsigned char  switch_flags, switch_hold;

ISR (ADC_vect)
{
    // Get new ADC value
    adc_result  =  ADCL;
    adc_result += (ADCH << 8);

    // Only notify of the new result if it is different from the
    // previously read result... this should save some time.
    if (adc_result != adc_result_old)
    {
        adc_result_old = adc_result;
        adc_flag       = 1;
    }
    return;
}

ISR (INT0_vect)
{
    // Power Switch
    INT_PWRSW_DISABLE;
    TIMER_INT_START;
    bit_set(switch_flags, 7);
    switch_hold++;
    return;
}

// NIC Interrupt
ISR (INT2_vect)
{
    draw_icon(1, ICON_NETWORK);
    bit_set(icon_flags, ICON_NETWORK);
    return;
}

ISR (PCINT1_vect)
{
    if (INT_LEFTSW_PIN == 0)
    {
        // Left Switch
        INT_LEFTSW_DISABLE;
        TIMER_INT_START;
        bit_set(switch_flags, INT_LEFTSW);
    }

    if (INT_RIGHTSW_PIN == 0)
    {
        // Right Switch
```

```
        INT_RIGHTSW_DISABLE;
        TIMER_INT_START;
        bit_set(switch_flags, INT_RIGHTSW);
    }
    return;
}

ISR (PCINT0_vect)
{
    if (bit_get(SW_PINS1,INT_DOWNSW) == 0)
    {
        // Down Switch
        INT_DOWNSW_DISABLE;
        TIMER_INT_START;
        bit_set(switch_flags, INT_DOWNSW);
    }
    else if (bit_get(SW_PINS1,INT_UPSW) == 0)
    {
        // Up Switch
        INT_UPSW_DISABLE;
        TIMER_INT_START;
        bit_set(switch_flags, INT_UPSW);
    }
    else if (bit_get(SW_PINS1,INT_FANSW) == 0)
    {
        // Fan Switch
        INT_FANSW_DISABLE;
        TIMER_INT_START;
        bit_set(switch_flags, INT_FANSW);
    }
    return;
}

ISR (TIMER0_OVF_vect)
{
    // Count for a much longer amount of time
    LED_WHITE_ON;
    timer_int_hits++;
    if (timer_int_hits > 6)
    {
         LED_WHITE_OFF;
        // This is the power switch we are checking
        if (bit_get(switch_flags, 7))
        {
            // Switch is now high (not pressed)
            if (INT_PWRSW_PIN)
            {
                // If held down for a while, enter the menu
                // otherwise just report a normal press.
                if (switch_hold > 24)
                {
                    pgm_mode = SETTINGS;

                }
                else                bit_set(switch_flags, INT_PWRSW);
                switch_hold = 0;
            }
            bit_clr(switch_flags, 7);
        }

        // Reset timer
```

```
        TIMER_INT_STOP;
        TIMER_INT_COUNT=0;
        timer_int_hits =0;

        // Re-enable (all) switch interrupts
        INT_LEFTSW_ENABLE;
        INT_RIGHTSW_ENABLE;
        INT_PWRSW_ENABLE;
        INT_UPSW_ENABLE;
        INT_DOWNSW_ENABLE;
        INT_FANSW_ENABLE;

        // The switch is still being held down,
        // Keep noticing it..
        if (INT_RIGHTSW_PIN == 0)
        {
            TIMER_INT_START;
            bit_set(switch_flags, INT_RIGHTSW);
        }

        else if (INT_LEFTSW_PIN == 0)
        {
            TIMER_INT_START;
            bit_set(switch_flags, INT_LEFTSW);
        }

        else if (INT_UPSW_PIN == 0)
        {
            TIMER_INT_START;
            bit_set(switch_flags, INT_UPSW);
        }

        else if (INT_DOWNSW_PIN == 0)
        {
            TIMER_INT_START;
            bit_set(switch_flags, INT_DOWNSW);
        }
    }
    return;
}
```

```
#ifndef INTS_H
#define INTS_H

// The PIN registers to which the switches are connected
#define     SW_PINS1            PINA
#define     SW_PINS2            PINB
#define     SW_PINPWR           PIND

#define     INT_DOWNSW          5
#define     INT_DOWNSW_ENABLE   bit_set(PCMSK0, 5)
#define     INT_DOWNSW_DISABLE  bit_clr(PCMSK0, 5)
#define     INT_DOWNSW_PIN      bit_get(SW_PINS1, INT_DOWNSW)

#define     INT_UPSW            4
#define     INT_UPSW_ENABLE     bit_set(PCMSK0, 4)
#define     INT_UPSW_DISABLE    bit_clr(PCMSK0, 4)
#define     INT_UPSW_PIN        bit_get(SW_PINS1, INT_UPSW)

#define     INT_FANSW           3
#define     INT_FANSW_ENABLE    bit_set(PCMSK0, 3)
#define     INT_FANSW_DISABLE   bit_clr(PCMSK0, 3)

#define     INT_PWRSW           2
#define     INT_PWRSW_ENABLE    bit_set(EIMSK, 0)
#define     INT_PWRSW_DISABLE   bit_clr(EIMSK, 0)
#define     INT_PWRSW_PIN       bit_get(SW_PINPWR, INT_PWRSW)

#define     INT_RIGHTSW         1
#define     INT_RIGHTSW_ENABLE  bit_set(PCMSK1, 1)
#define     INT_RIGHTSW_DISABLE bit_clr(PCMSK1, 1)
#define     INT_RIGHTSW_PIN     bit_get(SW_PINS2, INT_RIGHTSW)

#define     INT_LEFTSW          0
#define     INT_LEFTSW_ENABLE   bit_set(PCMSK1, 0)
#define     INT_LEFTSW_DISABLE  bit_clr(PCMSK1, 0)
#define     INT_LEFTSW_PIN      bit_get(SW_PINS2, INT_LEFTSW)

#define     TIMER_INT_START     TCCR0B |= 0b00000101
#define     TIMER_INT_STOP      TCCR0B &= 0b11111000
#define     TIMER_INT_COUNT     TCNT0

extern volatile unsigned int   adc_result, adc_result_old;
extern volatile unsigned char  adc_flag;
extern volatile unsigned char  switch_flags;

#endif
```

```
/*********************************************
 * vim:sw=8:ts=8:si:et
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 *
 * Author: Guido Socher
 * Copyright: GPL V2
 * See http://www.gnu.org/licenses/gpl.html
 *
 * IP, Arp, UDP and TCP functions.
 *
 * The TCP implementation uses some size optimisations which are valid
 * only if all data can be sent in one single packet. This is however
 * not a big limitation for a microcontroller as you will anyhow use
 * small web-pages. The web server must send the entire web page in one
 * packet. The client "web browser" as implemented here can also receive
 * large pages.
 *
 * Chip type           : ATMEGA88/168/328 with ENC28J60
 *********************************************/
// Includes modifications made by Nick Viera, April 2010.

#include <avr/io.h>
#include <avr/pgmspace.h>
#include <string.h>
#include <stdlib.h>
#include "net.h"
#include "enc28j60.h"
#include "ip_config.h"

static uint8_t  wwwport_l     = 80;  // server port
static uint8_t  wwwport_h     = 0;   // Note: never use same as TCPCLIENT_SRC_PORT_H
static uint16_t info_data_len = 0;
static uint8_t  seqnum        = 0xa; // my initial tcp sequence number
static uint8_t  macaddr[6];

// Removed using direct access
uint8_t  ipaddr [4];

static void    (*icmp_callback)(uint8_t *ip);

#define HTTP_HEADER_START ((uint16_t)TCP_SRC_PORT_H_P+(buf[TCP_HEADER_LEN_P]>>4)*4)
const char arpreqhdr[] PROGMEM ={0,1,8,0,6,4,0,1};

#if defined (WWW_client) || defined (NTP_client) ||  defined (WOL_client)
   // 0x82 is the total len on ip, 0x20 is ttl (time to live)
   const char iphdr[] PROGMEM ={0x45,0,0,0x82,0,0,0x40,0,0x20};
#endif

// The Ip checksum is calculated over the ip header only starting
// with the header length field and a total length of 20 bytes
// unitl ip.dst
// You must set the IP checksum field to zero before you start
// the calculation.
// len for ip is 20.
//
// For UDP/TCP we do not make up the required pseudo header. Instead we
// use the ip.src and ip.dst fields of the real packet:
// The udp checksum calculation starts with the ip.src field
// Ip.src=4bytes,Ip.dst=4 bytes,Udp header=8bytes + data length=16+len
```

```
// In other words the len here is 8 + length over which you actually
// want to calculate the checksum.
// You must set the checksum field to zero before you start
// the calculation.
// The same algorithm is also used for udp and tcp checksums.
// len for udp is: 8 + 8 + data length
// len for tcp is: 4+4 + 20 + option len + data length
//
// For more information on how this algorithm works see:
// http://www.netfor2.com/checksum.html
// http://www.msc.uky.edu/ken/cs471/notes/chap3.htm
// The RFC has also a C code example: http://www.faqs.org/rfcs/rfc1071.html
uint16_t checksum(uint8_t *buf, uint16_t len,uint8_t type){
        // type 0=ip / icmp; 1=udp; 2=tcp
        uint32_t sum = 0;

        //if(type==0){
        //        // do not add anything, standard IP checksum as described above
        //        // Usable for ICMP and IP header
        //}
        if(type==1)
        {
                sum+=IP_PROTO_UDP_V; // protocol udp
                // the length here is the length of udp (data+header len)
                // =length given to this function - (IP.scr+IP.dst length)
                sum+=len-8; // = real udp len
        }
        if(type==2)
        {
                sum+=IP_PROTO_TCP_V;
                // the length here is the length of tcp (data+header len)
                // =length given to this function - (IP.scr+IP.dst length)
                sum+=len-8; // = real tcp len
        }
        // build the sum of 16bit words
        while(len >1)
        {
                sum += 0xFFFF & (((uint32_t)*buf<<8)|*(buf+1));
                buf += 2;
                len -= 2;
        }
        // if there is a byte left then add it (padded with zero)
        if (len) sum += ((uint32_t)(0xFF & *buf))<<8;

        // now calculate the sum over the bytes in the sum
        // until the result is only 16bit long
        while (sum>>16) sum = (sum & 0xFFFF)+(sum >> 16);

        // build 1's complement:
        return( (uint16_t) sum ^ 0xFFFF);
}

// This initializes the web server
// you must call this function once before you use any of the other functions:
void init_ip_arp_udp_tcp(uint8_t *mymac,uint8_t *myip,uint16_t port)
{
    uint8_t i=0;
    wwwport_h = (port>>8)&0xff;
    wwwport_l = (port&0xff);

// Removed for direct access to thermostat setting
```

```
//     while(i < 4)
//     {
//         ipaddr[i]=myip[i];
//         i++;
//     }

    i = 0;
    while(i < 6)
    {
        macaddr[i]=mymac[i];
        i++;
    }
}

uint8_t check_ip_message_is_from(uint8_t *buf,uint8_t *ip)
{
    uint8_t i=0;
    while(i < 4){
        if(buf[IP_SRC_P+i]!=ip[i]) return(0);
        i++;
    }
    return(1);
}

uint8_t eth_type_is_arp_and_my_ip(uint8_t *buf,uint16_t len)
{
    uint8_t i=0;

    if (len<41) return(0);
    if (buf[ETH_TYPE_H_P] != ETHTYPE_ARP_H_V ||
        buf[ETH_TYPE_L_P] != ETHTYPE_ARP_L_V)
        return(0);

    while(i < 4)
    {
        if(buf[ETH_ARP_DST_IP_P+i] != ipaddr[i]) return(0);
        i++;
    }
    return(1);
}

uint8_t eth_type_is_ip_and_my_ip(uint8_t *buf,uint16_t len)
{
    uint8_t i=0;
    //eth+ip+udp header is 42
    if (len<42) return(0);
    if (buf[ETH_TYPE_H_P]!=ETHTYPE_IP_H_V ||
        buf[ETH_TYPE_L_P]!=ETHTYPE_IP_L_V)
        return(0);

    if (buf[IP_HEADER_LEN_VER_P]!=0x45)
    {
        // must be IP V4 and 20 byte header
        return(0);
    }
    while(i<4)
    {
        if(buf[IP_DST_P+i]!=ipaddr[i]){
                return(0);
        }
        i++;
```

```
    }
    return(1);
}

// make a return eth header from a received eth packet
void make_eth(uint8_t *buf)
{
    uint8_t i=0;
    //copy the destination mac from the source and fill my mac into src
    while(i < 6)
    {
        buf[ETH_DST_MAC +i] = buf[ETH_SRC_MAC +i];
        buf[ETH_SRC_MAC +i] = macaddr[i];
        i++;
    }
}

void fill_ip_hdr_checksum(uint8_t *buf)
{
        uint16_t ck;
        // clear the 2 byte checksum
        buf[IP_CHECKSUM_P]   = 0;
        buf[IP_CHECKSUM_P+1] = 0;
        buf[IP_FLAGS_P]      = 0x40; // don't fragment
        buf[IP_FLAGS_P+1]    = 0;    // fragement offset
        buf[IP_TTL_P]        = 64;   // ttl
        // calculate the checksum:
        ck                   = checksum(&buf[IP_P], IP_HEADER_LEN,0);
        buf[IP_CHECKSUM_P]   = ck>>8;
        buf[IP_CHECKSUM_P+1] = ck& 0xff;
}

// make a return ip header from a received ip packet
void make_ip(uint8_t *buf)
{
    uint8_t i=0;
    while(i < 4)
    {
        buf[IP_DST_P+i] = buf[IP_SRC_P+i];
        buf[IP_SRC_P+i] = ipaddr[i];
        i++;
    }
    fill_ip_hdr_checksum(buf);
}

// swap seq and ack number and count ack number up
void step_seq(uint8_t *buf,uint16_t rel_ack_num,uint8_t cp_seq)
{
        uint8_t i;
        uint8_t tseq;
        i=4;
        // sequence numbers:
        // add the rel ack num to SEQACK
        while(i>0){
                rel_ack_num=buf[TCP_SEQ_H_P+i-1]+rel_ack_num;
                tseq=buf[TCP_SEQACK_H_P+i-1];
                buf[TCP_SEQACK_H_P+i-1]=0xff&rel_ack_num;
                if (cp_seq){
                        // copy the acknum sent to us into the sequence number
                        buf[TCP_SEQ_H_P+i-1]=tseq;
                }else{
```

```
                        buf[TCP_SEQ_H_P+i-1]= 0; // some preset vallue
                }
                rel_ack_num=rel_ack_num>>8;
                i--;
        }
}

// make a return tcp header from a received tcp packet
// rel_ack_num is how much we must step the seq number received from the
// other side. We do not send more than 765 bytes of text (=data) in the tcp packet.
// No mss is included here.
//
// After calling this function you can fill in the first data byte at TCP_OPTIONS_P+4
// If cp_seq=0 then an initial sequence number is used (should be use in synack)
// otherwise it is copied from the packet we received
void make_tcphead(uint8_t *buf,uint16_t rel_ack_num,uint8_t cp_seq)
{
        uint8_t i;
        // copy ports:
        i=buf[TCP_DST_PORT_H_P];
        buf[TCP_DST_PORT_H_P]=buf[TCP_SRC_PORT_H_P];
        buf[TCP_SRC_PORT_H_P]=i;
        //
        i=buf[TCP_DST_PORT_L_P];
        buf[TCP_DST_PORT_L_P]=buf[TCP_SRC_PORT_L_P];
        buf[TCP_SRC_PORT_L_P]=i;
        step_seq(buf,rel_ack_num,cp_seq);
        // zero the checksum
        buf[TCP_CHECKSUM_H_P]=0;
        buf[TCP_CHECKSUM_L_P]=0;
        // no options:
        // 20 bytes:
        // The tcp header length is only a 4 bit field (the upper 4 bits).
        // It is calculated in units of 4 bytes.
        // E.g 20 bytes: 20/4=6 => 0x50=header len field
        buf[TCP_HEADER_LEN_P]=0x50;
}

void make_arp_answer_from_request(uint8_t *buf)
{
        uint8_t i=0;
        //
        make_eth(buf);
        buf[ETH_ARP_OPCODE_H_P]=ETH_ARP_OPCODE_REPLY_H_V;
        buf[ETH_ARP_OPCODE_L_P]=ETH_ARP_OPCODE_REPLY_L_V;
        // fill the mac addresses:
        while(i<6){
                buf[ETH_ARP_DST_MAC_P+i]=buf[ETH_ARP_SRC_MAC_P+i];
                buf[ETH_ARP_SRC_MAC_P+i]=macaddr[i];
                i++;
        }
        i=0;
        while(i<4){
                buf[ETH_ARP_DST_IP_P+i]=buf[ETH_ARP_SRC_IP_P+i];
                buf[ETH_ARP_SRC_IP_P+i]=ipaddr[i];
                i++;
        }
        // eth+arp is 42 bytes:
        enc28j60PacketSend(42,buf);
}
```

```
void make_echo_reply_from_request(uint8_t *buf,uint16_t len)
{
        make_eth(buf);
        make_ip(buf);
        buf[ICMP_TYPE_P]=ICMP_TYPE_ECHOREPLY_V;
        // we changed only the icmp.type field from request(=8) to reply(=0).
        // we can therefore easily correct the checksum:
        if (buf[ICMP_CHECKSUM_P] > (0xff-0x08)){
                buf[ICMP_CHECKSUM_P+1]++;
        }
        buf[ICMP_CHECKSUM_P]+=0x08;
        //
        enc28j60PacketSend(len,buf);
}

// you can send a max of 220 bytes of data
void make_udp_reply_from_request(uint8_t *buf,char *data,uint8_t datalen,uint16_t port)
{
        uint8_t i=0;
        uint16_t ck;
        make_eth(buf);
        if (datalen>220){
                datalen=220;
        }
        // total length field in the IP header must be set:
        buf[IP_TOTLEN_H_P]=0;
        buf[IP_TOTLEN_L_P]=IP_HEADER_LEN+UDP_HEADER_LEN+datalen;
        make_ip(buf);
        // send to port:
        //buf[UDP_DST_PORT_H_P]=port>>8;
        //buf[UDP_DST_PORT_L_P]=port & 0xff;
        // sent to port of sender and use "port" as own source:
        buf[UDP_DST_PORT_H_P]=buf[UDP_SRC_PORT_H_P];
        buf[UDP_DST_PORT_L_P]= buf[UDP_SRC_PORT_L_P];
        buf[UDP_SRC_PORT_H_P]=port>>8;
        buf[UDP_SRC_PORT_L_P]=port & 0xff;
        // calculte the udp length:
        buf[UDP_LEN_H_P]=0;
        buf[UDP_LEN_L_P]=UDP_HEADER_LEN+datalen;
        // zero the checksum
        buf[UDP_CHECKSUM_H_P]=0;
        buf[UDP_CHECKSUM_L_P]=0;
        // copy the data:
        while(i<datalen){
                buf[UDP_DATA_P+i]=data[i];
                i++;
        }
        ck=checksum(&buf[IP_SRC_P], 16 + datalen,1);
        buf[UDP_CHECKSUM_H_P]=ck>>8;
        buf[UDP_CHECKSUM_L_P]=ck& 0xff;
        enc28j60PacketSend(UDP_HEADER_LEN+IP_HEADER_LEN+ETH_HEADER_LEN+datalen,buf);
}

void make_tcp_synack_from_syn(uint8_t *buf)
{
        uint16_t ck;
        make_eth(buf);
        // total length field in the IP header must be set:
        // 20 bytes IP + 24 bytes (20tcp+4tcp options)
        buf[IP_TOTLEN_H_P]=0;
        buf[IP_TOTLEN_L_P]=IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN+4;
```

```
        make_ip(buf);
        buf[TCP_FLAGS_P]=TCP_FLAGS_SYNACK_V;
        make_tcphead(buf,1,0);
        // put an inital seq number
        buf[TCP_SEQ_H_P+0]= 0;
        buf[TCP_SEQ_H_P+1]= 0;
        // we step only the second byte, this allows us to send packts
        // with 255 bytes, 512  or 765 (step by 3) without generating
        // overlapping numbers.
        buf[TCP_SEQ_H_P+2]= seqnum;
        buf[TCP_SEQ_H_P+3]= 0;
        // step the inititial seq num by something we will not use
        // during this tcp session:
        seqnum+=3;
        // add an mss options field with MSS to 1280:
        // 1280 in hex is 0x500
        buf[TCP_OPTIONS_P]=2;
        buf[TCP_OPTIONS_P+1]=4;
        buf[TCP_OPTIONS_P+2]=0x05;
        buf[TCP_OPTIONS_P+3]=0x0;
        // The tcp header length is only a 4 bit field (the upper 4 bits).
        // It is calculated in units of 4 bytes.
        // E.g 24 bytes: 24/4=6 => 0x60=header len field
        buf[TCP_HEADER_LEN_P]=0x60;
        // here we must just be sure that the web browser contacting us
        // will send only one get packet
        buf[TCP_WIN_SIZE]=0x5; // 1400=0x578
        buf[TCP_WIN_SIZE+1]=0x78;
        // calculate the checksum, len=8 (start from ip.src) + TCP_HEADER_LEN_PLAIN + 4 (one
option: mss)
        ck=checksum(&buf[IP_SRC_P], 8+TCP_HEADER_LEN_PLAIN+4,2);
        buf[TCP_CHECKSUM_H_P]=ck>>8;
        buf[TCP_CHECKSUM_L_P]=ck& 0xff;
        // add 4 for option mss:
        enc28j60PacketSend(IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN+4+ETH_HEADER_LEN,buf);
}

// do some basic length calculations and store the result in static varibales
uint16_t get_tcp_data_len(uint8_t *buf)
{
        int16_t i;
        i=(((int16_t)buf[IP_TOTLEN_H_P])<<8)|(buf[IP_TOTLEN_L_P]&0xff);
        i-=IP_HEADER_LEN;
        i-=(buf[TCP_HEADER_LEN_P]>>4)*4; // generate len in bytes;
        if (i<=0){
                i=0;
        }
        return((uint16_t)i);
}


// fill in tcp data at position pos. pos=0 means start of
// tcp data. Returns the position at which the string after
// this string could be filled.
uint16_t fill_tcp_data_p(uint8_t *buf, uint16_t pos, const prog_char *progmem_s)
{
        char c;
        // fill in tcp data at position pos
        // with no options the data starts after the checksum + 2 more bytes (urgent ptr)
        while ((c = pgm_read_byte(progmem_s++))) {
                buf[TCP_CHECKSUM_L_P+3+pos] = c;
```

```
                pos++;
        }
        return(pos);
}

// fill in tcp data at position pos. pos=0 means start of
// tcp data. Returns the position at which the string after
// this string could be filled.
uint16_t fill_tcp_data(uint8_t *buf,uint16_t pos, const char *s)
{
        // fill in tcp data at position pos
        //
        // with no options the data starts after the checksum + 2 more bytes (urgent ptr)
        while (s && *s) {
                buf[TCP_CHECKSUM_L_P+3+pos]=*s;
                pos++;
                s++;
        }
        return(pos);
}

// Make just an ack packet with no tcp data inside
// This will modify the eth/ip/tcp header
void make_tcp_ack_from_any(uint8_t *buf,int16_t datlentoack,uint8_t addflags)
{
        uint16_t j;
        make_eth(buf);
        // fill the header:
        buf[TCP_FLAGS_P]=TCP_FLAGS_ACK_V|addflags;
        if (datlentoack==0){
                // if there is no data then we must still acknoledge one packet
                make_tcphead(buf,1,1); // no options
        }else{
                make_tcphead(buf,datlentoack,1); // no options
        }
        // total length field in the IP header must be set:
        // 20 bytes IP + 20 bytes tcp (when no options)
        j=IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN;
        buf[IP_TOTLEN_H_P]=j>>8;
        buf[IP_TOTLEN_L_P]=j& 0xff;
        make_ip(buf);
        // use a low window size otherwise we have to have
        // timers and can not just react on every packet.
        buf[TCP_WIN_SIZE]=0x4; // 1024=0x400
        buf[TCP_WIN_SIZE+1]=0x0;
        // calculate the checksum, len=8 (start from ip.src) + TCP_HEADER_LEN_PLAIN + data
len
        j=checksum(&buf[IP_SRC_P], 8+TCP_HEADER_LEN_PLAIN,2);
        buf[TCP_CHECKSUM_H_P]=j>>8;
        buf[TCP_CHECKSUM_L_P]=j& 0xff;
        enc28j60PacketSend(IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN+ETH_HEADER_LEN,buf);
}


// dlen is the amount of tcp data (http data) we send in this packet
// You can use this function only immediately after make_tcp_ack_from_any
// This is because this function will NOT modify the eth/ip/tcp header except for
// length and checksum
// You must set TCP_FLAGS before calling this
void make_tcp_ack_with_data_noflags(uint8_t *buf,uint16_t dlen)
{
```

```
        uint16_t j;
        // total length field in the IP header must be set:
        // 20 bytes IP + 20 bytes tcp (when no options) + len of data
        j=IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN+dlen;
        buf[IP_TOTLEN_H_P]=j>>8;
        buf[IP_TOTLEN_L_P]=j& 0xff;
        fill_ip_hdr_checksum(buf);
        // zero the checksum
        buf[TCP_CHECKSUM_H_P]=0;
        buf[TCP_CHECKSUM_L_P]=0;
        // calculate the checksum, len=8 (start from ip.src) + TCP_HEADER_LEN_PLAIN + data
len
        j=checksum(&buf[IP_SRC_P], 8+TCP_HEADER_LEN_PLAIN+dlen,2);
        buf[TCP_CHECKSUM_H_P]=j>>8;
        buf[TCP_CHECKSUM_L_P]=j& 0xff;
        enc28j60PacketSend(IP_HEADER_LEN+TCP_HEADER_LEN_PLAIN+dlen+ETH_HEADER_LEN,buf);
}

// you must have initialized info_data_len at some time before calling this function
//
// This info_data_len initialisation is done automatically if you call
// packetloop_icmp_tcp(buf,enc28j60PacketReceive(BUFFER_SIZE, buf));
// and test the return value for non zero.
//
// dlen is the amount of tcp data (http data) we send in this packet
// You can use this function only immediately after make_tcp_ack_from_any
// This is because this function will NOT modify the eth/ip/tcp header except for
// length and checksum
void www_server_reply(uint8_t *buf,uint16_t dlen)
{
        make_tcp_ack_from_any(buf,info_data_len,0); // send ack for http get
        // fill the header:
        // This code requires that we send only one data packet
        // because we keep no state information. We must therefore set
        // the fin here:
        buf[TCP_FLAGS_P]=TCP_FLAGS_ACK_V|TCP_FLAGS_PUSH_V|TCP_FLAGS_FIN_V;
        make_tcp_ack_with_data_noflags(buf,dlen); // send data
}

#if defined (WWW_client) || defined (NTP_client) ||  defined (WOL_client)
// fill buffer with a prog-mem string
void fill_buf_p(uint8_t *buf,uint16_t len, const prog_char *progmem_s)
{
        while (len){
                *buf= pgm_read_byte(progmem_s);
                buf++;
                progmem_s++;
                len--;
        }
}
#endif

#ifdef PING_client
// icmp echo, matchpat is a pattern that has to be sent back by the
// host answering the ping.
// The ping is sent to destip  and mac gwmacaddr
void client_icmp_request(uint8_t *buf,uint8_t *destip)
{
        uint8_t i=0;
        uint16_t ck;
        //
```

```
        while(i<6){
                buf[ETH_DST_MAC +i]=gwmacaddr[i]; // gw mac in local lan or host mac
                buf[ETH_SRC_MAC +i]=macaddr[i];
                i++;
        }
        buf[ETH_TYPE_H_P] = ETHTYPE_IP_H_V;
        buf[ETH_TYPE_L_P] = ETHTYPE_IP_L_V;
        fill_buf_p(&buf[IP_P],9,iphdr);
        buf[IP_TOTLEN_L_P]=0x82;
        buf[IP_PROTO_P]=IP_PROTO_UDP_V;
        i=0;
        while(i<4){
                buf[IP_DST_P+i]=destip[i];
                buf[IP_SRC_P+i]=ipaddr[i];
                i++;
        }
        fill_ip_hdr_checksum(buf);
        buf[ICMP_TYPE_P]=ICMP_TYPE_ECHOREQUEST_V;
        buf[ICMP_TYPE_P+1]=0; // code
        // zero the checksum
        buf[ICMP_CHECKSUM_H_P]=0;
        buf[ICMP_CHECKSUM_L_P]=0;
        // a possibly unique id of this host:
        buf[ICMP_IDENT_H_P]=5; // some number
        buf[ICMP_IDENT_L_P]=ipaddr[3]; // last byte of my IP
        //
        buf[ICMP_IDENT_L_P+1]=0; // seq number, high byte
        buf[ICMP_IDENT_L_P+2]=1; // seq number, low byte, we send only 1 ping at a time
        // copy the data:
        i=0;
        while(i<56){
                buf[ICMP_DATA_P+i]=PINGPATTERN;
                i++;
        }
        //
        ck=checksum(&buf[ICMP_TYPE_P], 56+8,0);
        buf[ICMP_CHECKSUM_H_P]=ck>>8;
        buf[ICMP_CHECKSUM_L_P]=ck& 0xff;
        enc28j60PacketSend(98,buf);
}
#endif // PING_client

#ifdef WOL_client
// -------------------- special code to make a WOL packet

// A WOL (Wake on Lan) packet is a UDP packet to the broadcast
// address and UDP port 9. The data part contains 6x FF followed by
// 16 times the mac address of the host to wake-up
//
void send_wol(uint8_t *buf,uint8_t *wolmac)
{
        uint8_t i=0;
        uint8_t m=0;
        uint8_t pos=0;
        uint16_t ck;
        //
        while(i<6){
                buf[ETH_DST_MAC +i]=0xff;
                buf[ETH_SRC_MAC +i]=macaddr[i];
                i++;
        }
```

```
        buf[ETH_TYPE_H_P] = ETHTYPE_IP_H_V;
        buf[ETH_TYPE_L_P] = ETHTYPE_IP_L_V;
        fill_buf_p(&buf[IP_P],9,iphdr);
        buf[IP_TOTLEN_L_P]=0x54;
        buf[IP_PROTO_P]=IP_PROTO_ICMP_V;
        i=0;
        while(i<4){
                buf[IP_SRC_P+i]=ipaddr[i];
                buf[IP_DST_P+i]=0xff;
                i++;
        }
        fill_ip_hdr_checksum(buf);
        buf[UDP_DST_PORT_H_P]=0;
        buf[UDP_DST_PORT_L_P]=0x9; // wol=normally 9
        buf[UDP_SRC_PORT_H_P]=10;
        buf[UDP_SRC_PORT_L_P]=0x42; // source port does not matter
        buf[UDP_LEN_H_P]=0;
        buf[UDP_LEN_L_P]=110; // fixed len
        // zero the checksum
        buf[UDP_CHECKSUM_H_P]=0;
        buf[UDP_CHECKSUM_L_P]=0;
        // copy the data (102 bytes):
        i=0;
        while(i<6){
                buf[UDP_DATA_P+i]=0xff;
                i++;
        }
        m=0;
        pos=UDP_DATA_P+i;
        while (m<16){
                i=0;
                while(i<6){
                        buf[pos]=wolmac[i];
                        i++;
                        pos++;
                }
                m++;
        }
        //
        ck=checksum(&buf[IP_SRC_P], 16+ 102,1);
        buf[UDP_CHECKSUM_H_P]=ck>>8;
        buf[UDP_CHECKSUM_L_P]=ck& 0xff;

        enc28j60PacketSend(pos,buf);
}
#endif // WOL_client

void register_ping_rec_callback(void (*callback)(uint8_t *srcip))
{
        icmp_callback=callback;
}

#ifdef PING_client
// loop over this to check if we get a ping reply:
uint8_t packetloop_icmp_checkreply(uint8_t *buf,uint8_t *ip_monitoredhost)
{
        if(buf[IP_PROTO_P]==IP_PROTO_ICMP_V && buf[ICMP_TYPE_P]==ICMP_TYPE_ECHOREPLY_V){
                if (buf[ICMP_DATA_P]== PINGPATTERN){
                        if (check_ip_message_is_from(buf,ip_monitoredhost)){
                                return(1);
                                // ping reply is from monitored host and ping was from us
```

```
                        }
                }
        }
        return(0);
}
#endif // PING_client

// return 0 to just continue in the packet loop and return the position
// of the tcp data if there is tcp data part
//       *buf = pointer to start of packet
//       plen = packet length
uint16_t packetloop_icmp_tcp(uint8_t *buf, uint16_t plen)
{
        uint16_t len;

        //plen will be unequal to zero if there is a valid
        // packet (without crc error):

        // arp is broadcast if unknown but a host may also
        // verify the mac address by sending it to
        // a unicast address.
        if(eth_type_is_arp_and_my_ip(buf,plen))
        {
                if (buf[ETH_ARP_OPCODE_L_P]==ETH_ARP_OPCODE_REQ_L_V)
                {
                        // is it an arp request
                        make_arp_answer_from_request(buf);
                }

                return(0);

        }
        // check if ip packets are for us:
        if(eth_type_is_ip_and_my_ip(buf,plen)==0) return(0);

        if(buf[IP_PROTO_P]==IP_PROTO_ICMP_V && buf[ICMP_TYPE_P]==ICMP_TYPE_ECHOREQUEST_V)
        {
                if (icmp_callback){ (*icmp_callback)(&(buf[IP_SRC_P])); }
                // a ping packet, let's send pong
                make_echo_reply_from_request(buf,plen);
                return(0);
        }
        if (plen<54 && buf[IP_PROTO_P]!=IP_PROTO_TCP_V )
        {
                // smaller than the smallest TCP packet and not tcp port
                return(0);
        }

        //
        // tcp port web server start
        if (buf[TCP_DST_PORT_H_P]==wwwport_h && buf[TCP_DST_PORT_L_P]==wwwport_l){
                if (buf[TCP_FLAGS_P] & TCP_FLAGS_SYN_V){
                        make_tcp_synack_from_syn(buf);
                        // make_tcp_synack_from_syn does already send the syn,ack
                        return(0);
                }
                if (buf[TCP_FLAGS_P] & TCP_FLAGS_ACK_V){
                        info_data_len=get_tcp_data_len(buf);
                        // we can possibly have no data, just ack:
                        // Here we misuse plen for something else to save a variable.
                        // plen is now the position of start of the tcp user data.
```

```
                        len = HTTP_HEADER_START;
                        if (info_data_len == 0)
                        {
                                if (buf[TCP_FLAGS_P] & TCP_FLAGS_FIN_V){
                                        // finack, answer with ack
                                        make_tcp_ack_from_any(buf,0,0);
                                }
                                // just an ack with no data, wait for next packet
                                return(0);
                        }
                        // Here we misuse plen for something else to save a variable
                        return(len);
                }
        }
        return(0);
}
```

```
/*********************************************
 * vim:sw=8:ts=8:si:et
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 * Author: Guido Socher
 * Copyright: GPL V2
 *
 * IP/ARP/UDP/TCP functions
 *
 * Chip type           : ATMEGA88 with ENC28J60
 *********************************************/
//@{
#ifndef IP_ARP_UDP_TCP_H
#define IP_ARP_UDP_TCP_H

#include "ip_config.h"
#include <avr/pgmspace.h>

// Added for thermostat direct reading of register
extern uint8_t  ipaddr [4];

// -- web server functions --
// you must call this function once before you use any of the other server functions:
extern void init_ip_arp_udp_tcp(uint8_t *mymac,uint8_t *myip,uint16_t port);
// for a UDP server:
extern uint8_t eth_type_is_ip_and_my_ip(uint8_t *buf,uint16_t len);
extern void make_udp_reply_from_request(uint8_t *buf,char *data,uint8_t datalen,uint16_t
port);
// return 0 to just continue in the packet loop and return the position
// of the tcp data if there is tcp data part
extern uint16_t packetloop_icmp_tcp(uint8_t *buf,uint16_t plen);
// functions to fill the web pages with data:
extern uint16_t fill_tcp_data_p(uint8_t *buf,uint16_t pos, const prog_char *progmem_s);
extern uint16_t fill_tcp_data(uint8_t *buf,uint16_t pos, const char *s);
// send data from the web server to the client:
extern void www_server_reply(uint8_t *buf,uint16_t dlen);

// -- client functions --
#if defined (WWW_client) || defined (NTP_client)
extern void client_set_gwip(uint8_t *gwipaddr);
extern void client_set_wwwip(uint8_t *wwwipaddr);
#endif

#define HTTP_HEADER_START ((uint16_t)TCP_SRC_PORT_H_P+(buf[TCP_HEADER_LEN_P]>>4)*4)
#ifdef WWW_client
// ----- http get
extern void client_browse_url(prog_char *urlbuf, char *urlbuf_varpart, prog_char
*hoststr,void (*callback)(uint8_t,uint16_t));
// The callback is a reference to a function which must look like this:
// void browserresult_callback(uint8_t statuscode,uint16_t datapos)
// statuscode=0 means a good webpage was received, with http code 200 OK
// statuscode=1 an http error was received
// statuscode=2 means the other side in not a web server and in this case datapos is also
zero
// ----- http post
// client web browser using http POST operation:
// additionalheaderline must be set to NULL if not used.
// postval is a string buffer which can only be de-allocated by the caller
// when the post operation was really done (e.g when callback was executed).
// postval must be urlencoded.
```

```
extern void client_http_post(prog_char *urlbuf, prog_char *hoststr, prog_char
*additionalheaderline,char *postval,void (*callback)(uint8_t,uint16_t));
// The callback is a reference to a function which must look like this:
// void browserresult_callback(uint8_t statuscode,uint16_t datapos)
// statuscode=0 means a good webpage was received, with http code 200 OK
// statuscode=1 an http error was received
// statuscode=2 means the other side in not a web server and in this case datapos is also
zero
#endif

#ifdef NTP_client
extern void client_ntp_request(uint8_t *buf,uint8_t *ntpip,uint8_t srcport);
extern uint8_t client_ntp_process_answer(uint8_t *buf,uint32_t *time,uint8_t dstport_l);
#endif

// you can find out who ping-ed you if you want:
extern void register_ping_rec_callback(void (*callback)(uint8_t *srcip));

#ifdef PING_client
extern void client_icmp_request(uint8_t *buf,uint8_t *destip);
// you must loop over this function to check if there was a ping reply:
extern uint8_t packetloop_icmp_checkreply(uint8_t *buf,uint8_t *ip_monitoredhost);
#endif // PING_client

#ifdef WOL_client
extern void send_wol(uint8_t *buf,uint8_t *wolmac);
#endif // WOL_client

//
#endif /* IP_ARP_UDP_TCP_H */
//@}
```

ip_config.h

```
/*********************************************
 * vim:sw=8:ts=8:si:et
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 * Author: Guido Socher
 * Copyright: GPL V2
 *
 * This file can be used to decide which functionallity of the
 * TCP/IP stack shall be available.
 *
 *********************************************/
//@{
#ifndef IP_CONFIG_H
#define IP_CONFIG_H

//------------- functions in ip_arp_udp_tcp.c --------------
// an NTP client (ntp clock):
#undef NTP_client

// to send out a ping:
#undef PING_client
#define PINGPATTERN 0x42

// a UDP wake on lan sender:
#undef WOL_client

// a "web browser". This can be use to upload data
// to a web server on the internet by encoding the data
// into the url (like a Form action of type GET):
#define WWW_client
// if you do not need a browser and just a server:
//#undef WWW_client
//
//------------- functions in websrv_help_functions.c --------------
//
// functions to decode cgi-form data:
#define FROMDECODE_websrv_help

// function to encode a URL (mostly needed for a web client)
#define URLENCODE_websrv_help

#endif /* IP_CONFIG_H */
//@}
```

```
// ============================================================================
// Low-level code and routines for ST7528 LCD use
// Parts of this code are from the GPL code:
//     Program for writing to Newhaven Display graphic LCD.
//     (c)2008 Curt Lagerstam - Newhaven Display International, LLC.
// ============================================================================

#include <avr/io.h>
#include <avr/pgmspace.h>
#include "lcd.h"
#include "main.h"
#include "screen.h"

volatile unsigned char lcd_coarse;
         unsigned char lcd_column, lcd_rowpage;

void lcd_out_data(unsigned char data)
{
    unsigned char upper_bits=0, lower_bits=0;

    // Prepare and write upper 5 bits (DB7-DB2)
    upper_bits = (data & 0b11111100);
    // Prepare and write lower 2 bits (DB1,DB0)
    lower_bits = (data & 0b00000011);

    PORTC &= 0b00000011;
    PORTC |= upper_bits;

    PORTD &= 0b11111100;
    PORTD |= lower_bits;
    return;
}

unsigned char lcd_in_data(void)
{
    unsigned char upper_bits=0, lower_bits=0, data=0;

    // Read upper 5 and lower 2 bits (DB7-DB0)
    upper_bits = (PINC & 0b11111100);
    lower_bits = (PIND & 0b00000011);
    data = (upper_bits | lower_bits);
    return(data);
}

void lcd_e_strobe(void)
{
    LCD_DELAY1;
    LCD_E_CLR;
    LCD_DELAY2;
    LCD_E_SET;
    LCD_DELAY1;
    return;
}

unsigned char lcd_data_read(void)
{
    unsigned char data=0;

    LCD_DB_INPUTS;
```

```
    LCD_RS_SET;
    LCD_RW_SET;
    LCD_E_SET;

    lcd_e_strobe();
    data = lcd_in_data();
    return(data);
}


unsigned char lcd_comm_read(void)
{
    unsigned char data=0;

    LCD_DB_INPUTS;
    LCD_RS_CLR;
    LCD_RW_SET;
    LCD_E_SET;

    lcd_e_strobe();
    data = lcd_in_data();
    return(data);
}

void lcd_check_busy(void)
{
    unsigned char input=0;
    input = lcd_comm_read();
    // Don't proceed until LCD becomes ready
    // Command register, bit 7 = busy flag
    while (bit_get(input,7) != 0)
        asm volatile ("nop\n");
}

void lcd_data_write(unsigned char data)
{
    LCD_DB_OUTPUTS;
    LCD_RS_SET;
    LCD_RW_CLR;
    LCD_E_SET;

    lcd_out_data(data);
    lcd_e_strobe();
    lcd_check_busy();
    return;
}

void lcd_comm_write(unsigned char data)
{
    LCD_DB_OUTPUTS;
    LCD_RS_CLR;
    LCD_RW_CLR;
    LCD_E_SET;

    lcd_out_data(data);
    lcd_e_strobe();
    lcd_check_busy();
    return;
}

// Column: 0-127
```

```
void lcd_set_column(unsigned char column)
{
    // Byte H and L
    lcd_comm_write(0b00010000 | (column >> 4));
    lcd_comm_write(column & 0b00001111);
    // Keep track of the last column we were on
    lcd_column=column;
    return;
}

// Row (Actually, PAGE of 8 rows): 0-16
void lcd_set_rowpage(unsigned char row)
{
    lcd_comm_write(0b10110000 | row);
    // Keep track of the last row we were on
    lcd_rowpage = row;
    return;
}

void lcd_set_contrast(unsigned char dir)
{
    // Decrease Value
    if (dir == 0)
    {
        //if (lcd_contrast >= 1) lcd_contrast -= 1;
        if (lcd_contrast >= 2) lcd_contrast -= 2;

        // If fine=0, decrement coarse and start fine back at 63
        else
        {
            if (lcd_coarse > 32)
            {
                lcd_coarse--;
                lcd_contrast = 63;
            }
        }
    }

    // Increase Value
    else if (dir == 1)
    {
        //if (lcd_contrast <= 62) lcd_contrast += 1;
        if (lcd_contrast < 62) lcd_contrast += 2;

        // If fine=63, increment coarse and start fine back at 0
        else
        {
            if (lcd_coarse < 39 )
            {
                lcd_coarse++;
                lcd_contrast = 0;
            }
        }

    }
    lcd_comm_write(lcd_coarse);

    // Set Electronic Volume Register (2 bytes to set!)
    lcd_comm_write(0b10000001);     // 1st Byte
    lcd_comm_write(lcd_contrast);   // 2nd Byte n=0~3f (6 LSB)
    return;
```

```
}

void lcd_light_fade(unsigned char dir)
{
    while(1)
    {
        if (dir == 0)
        {
            if (BKLIGHT_PWM > lcd_light_min) BKLIGHT_PWM--;
            else
            {
                BKLIGHT_PWM = lcd_light_min;
                break;
            }
        }
        else if (dir == 1)
        {
            if (BKLIGHT_PWM < lcd_light_level) BKLIGHT_PWM++;
            else
            {
                BKLIGHT_PWM = lcd_light_level;
                break;
            }
        }
        delay(6000);
    }
    return;
}

void lcd_init(void)
{
    LCD_DB_OUTPUTS;
    LCD_E_SET;
    LCD_RESET_SET;
    delay(1000);

    // ==== Begin electronic CONTRAST settings ====

    // Set the resistor divider value for LCD Contrast
    // Control (the 3 Least Significant Bits)
    lcd_coarse = 39;               // min 32, max 39
    lcd_comm_write(lcd_coarse);   //111 works when force display is = 1

    // Set Electronic Volume Register (2 bytes to set!)
    lcd_contrast = 50;
    lcd_comm_write(0b10000001); // 1st Byte
    lcd_comm_write(lcd_contrast); // 2nd Byte n=0~3f (6 LSB)

    lcd_comm_write(0xA2);   // ICON OFF;
    lcd_comm_write(0xAE);   // Display OFF
    lcd_comm_write(0x48);   // Set Duty ratio
    lcd_comm_write(0x80);   // No operation
    lcd_comm_write(0xa0);   // Set scan direction
    lcd_comm_write(0xc8);   // SHL select
    lcd_comm_write(0x40);   // Set STARTLINE
    lcd_comm_write(0x00);
    lcd_comm_write(0xab);   // Start internal Oscillator

    lcd_comm_write(0x64);//3x
    delay(2000);
    lcd_comm_write(0x65);//4x
```

```
delay(2000);
lcd_comm_write(0x66);//5x
delay(2000);
lcd_comm_write(0x67);//6x
delay(2000);

// ==== Begin electronic CONTRAST settings ====

// Set the resistor divider value for LCD Contrast
// Control (the 3 Least Significant Bits)
lcd_coarse = 39;               // min 32, max 39
lcd_comm_write(lcd_coarse);   //111 works when force display is = 1

// Set Electronic Volume Register (2 bytes to set!)
lcd_contrast = 45;
lcd_comm_write(0b10000001); // 1st Byte
lcd_comm_write(lcd_contrast); // 2nd Byte n=0~3f (6 LSB)

// ==== End electronic CONTRAST settings ====

lcd_comm_write(0x57);//1/12bias
lcd_comm_write(0x92);//FRCandpwm

// Power Control VC,VR,VF
lcd_comm_write(0x2C);
delay(2000);//200ms
lcd_comm_write(0x2E);
delay(2000);//200ms
lcd_comm_write(0x2F);
delay(2000);//200ms

lcd_comm_write(0x92); //frcandpwm

// Mode, Frame Frequency, and Booster Efficiency control
lcd_comm_write(0b00111000);   // 1st Byte
lcd_comm_write(0b01110101);   // 2nd Byte   Ext register=1 (to set gray modes)

// startsettingsfor16-levelgrayscale
lcd_comm_write(0x97);//3frc,45pwm

lcd_comm_write(0x80);
lcd_comm_write(0x00);
lcd_comm_write(0x81);
lcd_comm_write(0x00);
lcd_comm_write(0x82);
lcd_comm_write(0x00);
lcd_comm_write(0x83);
lcd_comm_write(0x00);

lcd_comm_write(0x84);
lcd_comm_write(0x06);
lcd_comm_write(0x85);
lcd_comm_write(0x06);
lcd_comm_write(0x86);
lcd_comm_write(0x06);
lcd_comm_write(0x87);
lcd_comm_write(0x06);

lcd_comm_write(0x88);
lcd_comm_write(0x0b);
lcd_comm_write(0x89);
```

```
lcd_comm_write(0x0b);
lcd_comm_write(0x8a);
lcd_comm_write(0x0b);
lcd_comm_write(0x8b);
lcd_comm_write(0x0b);

lcd_comm_write(0x8c);
lcd_comm_write(0x10);
lcd_comm_write(0x8d);
lcd_comm_write(0x10);
lcd_comm_write(0x8e);
lcd_comm_write(0x10);
lcd_comm_write(0x8f);
lcd_comm_write(0x10);

lcd_comm_write(0x90);
lcd_comm_write(0x15);
lcd_comm_write(0x91);
lcd_comm_write(0x15);
lcd_comm_write(0x92);
lcd_comm_write(0x15);
lcd_comm_write(0x93);
lcd_comm_write(0x15);

lcd_comm_write(0x94);
lcd_comm_write(0x1a);
lcd_comm_write(0x95);
lcd_comm_write(0x1a);
lcd_comm_write(0x96);
lcd_comm_write(0x1a);
lcd_comm_write(0x97);
lcd_comm_write(0x1a);

lcd_comm_write(0x98);
lcd_comm_write(0x1e);
lcd_comm_write(0x99);
lcd_comm_write(0x1e);
lcd_comm_write(0x9a);
lcd_comm_write(0x1e);
lcd_comm_write(0x9b);
lcd_comm_write(0x1e);

lcd_comm_write(0x9c);
lcd_comm_write(0x23);
lcd_comm_write(0x9d);
lcd_comm_write(0x23);
lcd_comm_write(0x9e);
lcd_comm_write(0x23);
lcd_comm_write(0x9f);
lcd_comm_write(0x23);

lcd_comm_write(0xa0);
lcd_comm_write(0x27);
lcd_comm_write(0xa1);
lcd_comm_write(0x27);
lcd_comm_write(0xa2);
lcd_comm_write(0x27);
lcd_comm_write(0xa3);
lcd_comm_write(0x27);

lcd_comm_write(0xa4);
```

```
lcd_comm_write(0x2b);
lcd_comm_write(0xa5);
lcd_comm_write(0x2b);
lcd_comm_write(0xa6);
lcd_comm_write(0x2b);
lcd_comm_write(0xa7);
lcd_comm_write(0x2b);

lcd_comm_write(0xa8);
lcd_comm_write(0x2f);
lcd_comm_write(0xa9);
lcd_comm_write(0x2f);
lcd_comm_write(0xaa);
lcd_comm_write(0x2f);
lcd_comm_write(0xab);
lcd_comm_write(0x2f);

lcd_comm_write(0xac);
lcd_comm_write(0x32);
lcd_comm_write(0xad);
lcd_comm_write(0x32);
lcd_comm_write(0xae);
lcd_comm_write(0x32);
lcd_comm_write(0xaf);
lcd_comm_write(0x32);

lcd_comm_write(0xb0);
lcd_comm_write(0x35);
lcd_comm_write(0xb1);
lcd_comm_write(0x35);
lcd_comm_write(0xb2);
lcd_comm_write(0x35);
lcd_comm_write(0xb3);
lcd_comm_write(0x35);

lcd_comm_write(0xb4);
lcd_comm_write(0x38);
lcd_comm_write(0xb5);
lcd_comm_write(0x38);
lcd_comm_write(0xb6);
lcd_comm_write(0x38);
lcd_comm_write(0xb7);
lcd_comm_write(0x38);

lcd_comm_write(0xb8);
lcd_comm_write(0x3a);
lcd_comm_write(0xb9);
lcd_comm_write(0x3a);
lcd_comm_write(0xba);
lcd_comm_write(0x3a);
lcd_comm_write(0xbb);
lcd_comm_write(0x3a);

lcd_comm_write(0xbc);
lcd_comm_write(0x3c);
lcd_comm_write(0xbd);
lcd_comm_write(0x3c);
lcd_comm_write(0xbe);
lcd_comm_write(0x3c);
lcd_comm_write(0xbf);
lcd_comm_write(0x3c);
```

```
    //endsettingsfor16levelgrayscale

    // Mode, Frame Frequency, and Booster Efficiency control
    lcd_comm_write(0b00111000);   // 1st Byte
    lcd_comm_write(0b01110100);   // 2nd Byte   Ext register=0 (to set all other modes)

    display_clear();
    lcd_comm_write(0b10101111);   // Display ON
    return;
}
```

lcd.h

```
#ifndef LCD_H
#define LCD_H

#include <stdint.h>

unsigned char lcd_dl;

// Definitions for the Newhaven Display Model NHD-C128128BZ-FSW-GBW
#define LCD_DELAY1  for(lcd_dl=0; lcd_dl<10; lcd_dl++){ asm volatile("nop\n"); }
#define LCD_DELAY2  for(lcd_dl=0; lcd_dl<16; lcd_dl++){ asm volatile("nop\n"); }

// LCD definitions
#define LCD_E_SET           bit_set(PORTA, 0)
#define LCD_E_CLR           bit_clr(PORTA, 0)
#define LCD_RS_SET          bit_set(PORTA, 1)
#define LCD_RS_CLR          bit_clr(PORTA, 1)
#define LCD_RW_SET          bit_set(PORTA, 2)
#define LCD_RW_CLR          bit_clr(PORTA, 2)
#define LCD_RESET_SET       bit_set(PORTB, 4)
#define LCD_RESET_CLR       bit_clr(PORTB, 4)

#define LCD_DB_INPUTS       DDRD  &= 0b11111100, \
                            PORTD &= 0b11111100, \
                            DDRC  &= 0b00000011, \
                            PORTC &= 0b00000011
#define LCD_DB_OUTPUTS      DDRD  |= 0b00000011, \
                            PORTD &= 0b11111100, \
                            DDRC  |= 0b11111100, \
                            PORTC &= 0b00000011

// Warning: This sets MAX LED CURRENT!!!
#define BKLIGHT_PWM       OCR2A
#define BKLIGHT_PWM_MAX   245
#define BKLIGHT_PWM_MIN   0

extern unsigned char lcd_column, lcd_rowpage;
unsigned char   lcd_data_read    (void);
unsigned char   lcd_comm_read    (void);
void            lcd_data_write   (unsigned char data);
void            lcd_comm_write   (unsigned char data);
void            lcd_set_column   (unsigned char column);
void            lcd_set_rowpage  (unsigned char row);
void            lcd_set_contrast (unsigned char dir);
void            lcd_init         (void);
void            lcd_light_fade   (unsigned char dir);

#endif
```

```
    #include <avr/io.h>
    #include <stdlib.h>
    #include <stdio.h>
    #include <avr/portpins.h>
    #include <avr/interrupt.h>
    #include <avr/pgmspace.h>
    #include "main.h"
    #include "lcd.h"
    #include "screen.h"
    #include "interrupts.h"
    #include "i2c.h"
    #include "basic_adjust.h"
    #include "menu_settings.h"
    #include "eemem.h"
    #include "text.h"

    // Includes for Ethernet functionaility
    #include <string.h>
    #include "ip_arp_udp_tcp.h"
    #include "enc28j60.h"
    #include "timeout.h"
    #include "net.h"

#define BUFFER_SIZE  200
static unsigned char mymac[6] = {0x54,0x55,0x58,0x10,0x00,0x29};
static unsigned char buf [BUFFER_SIZE+1];

// All temperatures and settings are in degrees C and are multiplied by 10
// i.e.  100 = 10.0 degrees C
unsigned char temperature_lock=0, temperature_variance=30, temperature_override;
unsigned int  humidity_now, humidity_set;
volatile unsigned char
        pgm_mode_last, latch,

        swing_heat_low=10, swing_heat_high=10,
        swing_cool_low=10, swing_cool_high=10,

        heat_on_time,   heat_min_on_time=1,
        heat_off_time,  heat_min_off_time=1,
        cool_on_time,   cool_min_on_time=1,
        cool_off_time,  cool_min_off_time=1;

// bit 0 = fan, bit 1 = heat, bit 2=cool
         unsigned char system_mode_now=0;
volatile unsigned char icon_flags;

// Array for register / settings
// Column 0 = current value
// Column 1 = minimum value
// Column 2 = maximum value
volatile void *regs[12];
unsigned char reg10to19[3][7] =
                       { {192, 168,  7,222, 84, 28,  0},
                         {  0,   0,  0,  0,  0,  0,  0},
                         {255, 255,255,255, 99, 99,  1} };
unsigned char reg20to29[3][7] =
                       { {200,248,  0,  4,127,  2,  4},
                         {  0,  0,  0,  1,  0,  0,  1},
                         {255,248,  2, 15,255,  2, 15} };
```

```
unsigned char reg30to39[3][5] =
                        { {  0,  4,   0,   0},
                          {  0,  2,   0,   0},
                          {  1, 15,  50,  30} };
unsigned char reg90to99[3][8] =
                        { {  0,  0, 13,  6, 30,  4, 10},
                          {  0,  0,  0,  1,  1,  1,  0},
                          { 59, 59, 24,  7, 31, 12, 99} };
unsigned char reg100to109[3][3] =
                        { {  0,  0,  0},
                          {  0,  0,  0},
                          {  3,  2,  3} };
unsigned int reg110to119[3][3] =
                        { {  0,250,250},
                          {  0,  0,  0},
                          {400,400,400} };

void delay(unsigned long time)
{
    unsigned long i=0;
    for(i=0; i<=time; i++) asm volatile("nop");
    return;
}

unsigned char subtract_with_roll(unsigned char n2, unsigned char n1)
{
    signed char temp=0;
    unsigned char ret=0;

    // Base is 60 for seconds, minutes
    #define rollpoint 60

    temp = (n2 - n1);
    if (temp < 0) temp = (temp + rollpoint);
    ret = (unsigned char)temp;
    return(ret);
}

unsigned int get_humidity(void)
{
    unsigned int humidity=0;

    // Stop and Preset ADC to channel 6
    bit_set(DDRA,6);
    bit_clr(PORTA,6);
    asm ("nop");
    asm ("nop");
    asm ("nop");
    asm ("nop");
    bit_set(PORTA,6);
    asm ("nop");
    asm ("nop");
    bit_clr(DDRA,6);
    bit_clr(PORTA,6);
    ADMUX &= 0b11110000;
    ADMUX |= 0b00000110;
    ADC_START;

    while(adc_flag != 1) asm("nop");
    bit_set(DDRA,6);
    bit_clr(PORTA,6);
```

```
    humidity = adc_result;
    adc_flag=0;
    return(humidity);
}

signed int get_temperature(void)
{
    unsigned int tmpx100 = 0;

    // Vin = (1.1*ADC)/1024 = 0.0010742 * ADC
    // Multiply by 1000 to reduce float error
    tmpx100  = (float)(1.0742 * adc_result);
    tmpx100 = (tmpx100 - 500);
    return(tmpx100);
}

void adc_conversion(unsigned char channel)
{
    unsigned char temp=0;

    temp  = ADMUX;
    temp &= 0b11110000;
    temp |= channel;
    ADMUX = temp;
    ADC_START;
    return;
}

void leds_off(void)
{
    LED_WHITE_OFF;
    LED_RED_OFF;
    LED_GREEN_OFF;
    LED_BLUE_OFF;
    LED_AMBER_OFF;
    return;
}

void relays_off(void)
{
    RELAY_AUX_OFF;
    RELAY_COOL_OFF;
    RELAY_HEAT_OFF;
    RELAY_FAN_OFF;
    return;
}

void mcu_init(void)
{
    unsigned int tcp_port;

    // Initial Port Setup
    DDRA  = 0b01000111;
    DDRB  = 0b11111000;
    DDRC  = 0b11111111;
    DDRD  = 0b11111011;

    PORTA = 0b00111000;
    PORTB = 0b00000011;
    PORTC = 0b00000000;
    PORTD = 0b00000100;
```

```
// I2C Serial Bus and peripherals Setup
i2c_init();

// Analog to Digital Setup
ADMUX = 0b10000111;     // Bits 7,6 : Internal 1.1V reference
                        // Bit  5   : Right adjust result
                        // Bits 4-0 : Start on ADC channel 7
ADCSRA = 0b10001101;    // Bit  7   : ADC enable
                        // Bit  6   : Start conversion
                        // Bit  3   : Interrupt Enable
                        // Bit  2-0 : Prescaler clk/32

TCCR0A = 0b00000000;    // No Waveform Generation
TCCR0B = 0b00000000;    // Start with timer stopped
TIMSK0 = 0b00000001;    // Bit 0 : Enable overflow interrupt


TCCR2A       = 0b10100011;    // Enable OC2*, clear on match, Fast PWM mode
TCCR2B       = 0b00000001;    // Clock / 1
BKLIGHT_PWM  = 0;
LED_PWM      = 0;             // Start with PWM at 0% duty cycle

// Pin Change Interrupts Setup
EICRA  = 0b00110000;     // INT0 low level interrupt, INT2 rising edge
EIMSK  = 0b00000101;     // Enable INT0, INT2

PCMSK0 = 0b00111000;     // Enable PCI5,4,3  (Down, Up, Fan Switch)
PCMSK1 = 0b00000011;     // Enable PCI8,9     (Left, Right Switch)
PCICR  = 0b00000011;     // Enable Pin change interrupt masks 0,1

// SPI Setup
SPCR = 0b01010000;      // Enable SPI, enable master mode

// Ethernet Controller Initialization
// Set the clock speed to "no pre-scaler" (8MHz with internal osc or
// full external speed). Set the clock prescaler. First write CLKPCE
// to enable setting of clock the next four instructions.
CLKPR = (1<<CLKPCE);    // change enable
CLKPR = 0;              // "no pre-scaler"
_delay_loop_1(0);       // 60us

// Initialize the ethernet controller (enc28j60)
enc28j60Init(mymac);
enc28j60clkout(2);      // clkout from 6.25MHz to 12.5MHz
_delay_loop_1(0);       // 60us

// Ethernet Jack LED configuration
// LEDB = yellow, LEDA = green
// 0x476 is PHLCON LEDA = link status, LEDB = RX/TX Act
enc28j60PhyWrite(PHLCON,0x476);

// initialize the web server ethernet/ip layer:
// NOTE: This could be done way more efficiently by modifying the tcp stack
// to accept the parameters correctly.

// High to Low byte order
ipaddr[0] = tcp_ip_0;
ipaddr[1] = tcp_ip_1;
ipaddr[2] = tcp_ip_2;
ipaddr[3] = tcp_ip_3;
```

```
    tcp_port  = tcp_port_l;
    tcp_port += (tcp_port_h * 100);
    init_ip_arp_udp_tcp (mymac,0,tcp_port);

    sei();
    return;
}

// Gets the length of data from the pointer origin to the delimiter
unsigned int get_data_length(unsigned char *str, unsigned char delim)
{
    unsigned int location=0;

    while(1)
    {
        if((str[location] == 0x00) || (str[location] == delim)) break;
        else location++;
    }
    return(location);
}

// Takes a pointer denoting the beginning of a substring
// that contains numerical data and return the integer
// representation of that data. i.e. "123" -> 123
unsigned int parse_data(unsigned char *data, unsigned char end_delim)
{
    unsigned char *tmp;
    unsigned char length=0, actual_number;

    // Find the end of the section marked by end_delim and get length
    length = get_data_length(data, end_delim);

    // Allocate memory, copy the data substring to a new
    // pointer location and convert the string num to an int
    // Note that we strip off the leading delimited since we are
    // starting at &tmp[1] instead of &tmp[0]
    tmp = (unsigned char*)malloc(length+1);
    tmp = memmove(tmp, data, length);
    actual_number = atoi(&tmp[1]);
    return(actual_number);
}

void ether_send_ack(unsigned char regnum, signed int regdata)
{
    unsigned char temp=0;
    unsigned char outstring[25];
    unsigned char plen=0;
    unsigned char *ptr;

    const unsigned char str_ack[] = "@ACK";

    // Set pointer to first array element
    ptr = &(outstring[0]);

    // Copy strings and increment pointer
    ptr = strncpy(ptr, &(str_ack[0]), 4);
    ptr = ptr + 4;

    // Append '$', register number and increment pointer
    *ptr = '$';
```

```
    ptr++;
    temp = sprintf(ptr, "%d", regnum);
    ptr = ptr + temp;

    // Append '#', register value and increment pointer
    *ptr = '#';
    ptr ++;
    temp = sprintf(ptr, "%d", regdata);
    ptr = ptr + temp;

    // Reset pointer to beginning of output string
    ptr = &(outstring[0]);

    plen = fill_tcp_data(buf,plen,(char*)ptr);
    www_server_reply(buf,plen); // send data
}

// rw   : 0=read , 1=write
// data : pointer to beginning of data string
void ether_rw_register(unsigned char rw, unsigned char *data)
{
    unsigned char regnum=0, datanum=0;

    while(1)
    {

        // Find the start of reg marked by '$', abort on NULL
        data  = strchr(data,'$');
        if (data == 0x00) return;

        regnum = parse_data(data, '#');

        if ((regnum < 1) || (regnum > 200)) return;

        // We want to write the sent data to the register!
        if (rw == 1)
        {
            // Find the start of data block marked by '#',
            // abort if no data block was found
            data  = strchr(data,'#');
            if (data == 0x00) return;

            // Find the end of the data block
            datanum = parse_data(data, '$');

            // Attempt to write data to the register
            datanum = register_rw(2, datanum, regnum);
        }
        else if (rw == 0)
        {
            // Just read the current data from the register
            datanum = register_rw(0, 0, regnum);
        }

        ether_send_ack(regnum, datanum);  // Send reply

    }
    return;
}

void ether_check(void)
```

```
{
    unsigned int  packet_length=0, dat_p=0, new=0;
    //unsigned char *data;

    // Ethernet Stuff
    // Respond to PING and wait for an incomming TCP packet
    packet_length    = enc28j60PacketReceive(BUFFER_SIZE, buf);
    buf[BUFFER_SIZE] = '\0';
    dat_p            = packetloop_icmp_tcp(buf,packet_length);

    // Ignore the packet if it is NOT for us...
    if (dat_p == 0) return;
    if (strncmp("@SET", &(buf[dat_p]), 4) == 0)
    {
        lcd_set_column  (1);
        lcd_set_rowpage (13);
        draw_string(">@SET");
        ether_rw_register(1, &(buf[dat_p+4]));
    }
    else if (strncmp("@GET", &(buf[dat_p]), 4) == 0)
    {
        lcd_set_column  (1);
        lcd_set_rowpage (13);
        draw_string(">@GET");
        ether_rw_register(0, &(buf[dat_p+4]));
    }
}

void led_fadeup(void)
{
    unsigned char pwm=0;
    for (pwm=0; pwm < led_level; pwm++)
    {
        LED_PWM = pwm;
        delay(5000);
    }
    return;
}

int main(void)
{
    unsigned long i=0, j=0;
    unsigned char temp1=0, temp2=0, temp3=0;
    unsigned char time_switch=0, time_elap=0;

    mcu_init();
    ioexp_init();
    init_rtc();
    leds_off();
    relays_off();

    LED_WHITE_ON;

    delay(300000);
    delay(300000);
    delay(300000);

    //LED_WHITE_ON;
    //led_fadeup();

    lcd_init();
```

```
LED_WHITE_OFF;

// FIXME - cleanup
lcd_set_rowpage(6);
lcd_set_column(25);
draw_string_P(str_reading);
lcd_set_rowpage(7);
lcd_set_column(25);
draw_string_P(str_memory);

lcd_light_fade(1);

read_settings_all();
registers_update();
ee_datetime(0);

draw_off_screen();
adc_conversion(7);

draw_icon(1, ICON_NETWORK);
draw_icon(1, ICON_LOCK);
draw_icon(1, ICON_VENT);
draw_icon(1, ICON_WARN);

while(1)
{
    // In the settings menu
    if (pgm_mode == SETTINGS) { menu_settings(); }

    // In the normal, non-menu mode
    else
    {
        if (bit_get(switch_flags, INT_PWRSW))
        {
            sys_change();
            bit_clr(switch_flags, INT_PWRSW);
        }

        if (bit_get(switch_flags, INT_FANSW))
        {
            fan_change();
            bit_clr(switch_flags, INT_FANSW);
        }

        if (bit_get(switch_flags, INT_LEFTSW))
        {
            //fan_change();
            bit_clr(switch_flags, INT_LEFTSW);

        }

        if (bit_get(switch_flags, INT_RIGHTSW))
        {
            //fan_change();
            bit_clr(switch_flags, INT_RIGHTSW);

        }

        if (bit_get(switch_flags, INT_UPSW))
        {
            bit_clr(switch_flags, INT_UPSW);
```

```
        if (system_mode != SYS_OFF) temperature_change(1);
    }

    if (bit_get(switch_flags, INT_DOWNSW))
    {
        bit_clr(switch_flags, INT_DOWNSW);
        if (system_mode != SYS_OFF) temperature_change(0);
    }

    if (system_mode != SYS_OFF)
    {

      j++;
      if (j > 1500)
      {
        j=0;
        if (adc_flag == 1)
        {
            adc_flag = 0;

            temperature_now = get_temperature();
            print_temperature(temperature_now);
            humidity_now = get_humidity();
            print_humidity(humidity_now);
            adc_conversion(7);

            print_time();
        }
      }
    }
}

// Different functions for different operating modes
// Things which occur in ALL normal and menu screens
switch(system_mode)
{
    //=====================================================
    case(SYS_OFF):
        print_time();

        LED_RED_OFF;
        RELAY_HEAT_OFF;
        LED_BLUE_OFF;
        RELAY_COOL_OFF;
        LED_AMBER_OFF;
        RELAY_AUX_OFF;

        fan_switch_autooff();
        system_mode_now &= 0b11111001;
    break;

    //=====================================================
    case(SYS_HEAT):

        if (bit_get(system_mode_now, 1)) fan_switch_on();
        else                             fan_switch_autooff();

        time_switch = get_time(MINUTE);

        if (temperature_now < (temperature_set - swing_heat_low))
        {
```

```
        time_elap = subtract_with_roll(time_switch, heat_off_time);
        if (time_elap >= heat_min_off_time)
        {
            LED_RED_ON;
            RELAY_HEAT_ON;
            fan_switch_on();
            bit_set(system_mode_now, 1);
            heat_on_time = get_time(MINUTE);
        }
    }

    else if (temperature_now > (temperature_set + swing_heat_high))
    {
        time_elap = subtract_with_roll(time_switch, heat_on_time);
        if (time_elap >= heat_min_on_time)
        {
            LED_RED_OFF;
            RELAY_HEAT_OFF;
            fan_switch_autooff();
            bit_clr(system_mode_now, 1);
            heat_off_time = get_time(MINUTE);
        }
    }

break;

//=================================================
case(SYS_COOL):

    if (bit_get(system_mode_now, 2)) fan_switch_on();
    else                             fan_switch_autooff();

    time_elap = get_time(MINUTE);

    if (temperature_now < (temperature_set - swing_cool_low))
    {
        time_elap = subtract_with_roll(time_switch, cool_on_time);
        if (time_elap >= cool_min_on_time)
        {
            LED_BLUE_OFF;
            RELAY_COOL_OFF;
            fan_switch_autooff();
            bit_clr(system_mode_now, 2);
            cool_off_time = get_time(MINUTE);
        }
    }

    else if (temperature_now > (temperature_set + swing_cool_high))
    {
        time_elap = subtract_with_roll(time_switch, cool_off_time);
        if (time_elap >= cool_min_off_time)
        {
            LED_BLUE_ON;
            RELAY_COOL_ON;
            fan_switch_on();
            bit_set(system_mode_now, 2);
            cool_on_time = get_time(MINUTE);
        }
    }

break;
```

```
    //=================================================
    case(SYS_AUTO):

        time_elap = get_time(MINUTE);
        temp2     = get_time(SECOND);
        temp3     = subtract_with_roll(temp2, temp1);
        if (temp3 >= 3)
        {
            temp1 = temp2;
            temp3 = 0;
            LED_WHITE_TOG;
        }

        if (temperature_now < (temperature_set - swing_heat_low))
        {
            time_elap = subtract_with_roll(time_switch, heat_off_time);
            if (time_elap >= heat_min_off_time)
            {
                LED_RED_ON;
                RELAY_HEAT_ON;
                fan_switch_on();
                bit_set(system_mode_now, 1);
                heat_on_time = get_time(MINUTE);
            }
        }

        else if (temperature_now > (temperature_set + swing_cool_high))
        {
            time_elap = subtract_with_roll(time_switch, cool_off_time);
            if (time_elap >= cool_min_off_time)
            {
                LED_BLUE_ON;
                RELAY_COOL_ON;
                fan_switch_on();
                bit_set(system_mode_now, 2);
                cool_on_time = get_time(MINUTE);
            }
        }

        else
        {
                LED_BLUE_OFF;
                RELAY_COOL_OFF;
                fan_switch_autooff();
                bit_clr(system_mode_now, 2);
                cool_off_time = get_time(MINUTE);

                LED_RED_OFF;
                RELAY_HEAT_OFF;
                fan_switch_autooff();
                bit_clr(system_mode_now, 1);
                heat_off_time = get_time(MINUTE);

        }

    break;

    //=================================================
}
```

```
        i++;
        if (i > 250)
        {
            if (bit_get(icon_flags, ICON_NETWORK))
            {
                draw_icon(0, ICON_NETWORK);
                bit_clr(icon_flags, ICON_NETWORK);
            }
            i=0;
        }

        ether_check();
        registers_update();
    }
}
```

```
#ifndef MAIN_H
#define MAIN_H

    // Bitwise operation macros
    #define BIT(x)              (0x01 << (x))
    #define bit_get(p,m)        ((p) &  _BV(m))
    #define bit_set(p,m)        ((p) |=  _BV(m))
    #define bit_clr(p,m)        ((p) &=~ _BV(m))
    #define bit_tog(p,m)        ((p) ^=  _BV(m))

    // Relay-LED macros
    #define RELAY_FAN_ON        ioexp_pin_write(4,1)
    #define RELAY_FAN_OFF       ioexp_pin_write(4,0)
    #define RELAY_HEAT_ON       ioexp_pin_write(5,1)
    #define RELAY_HEAT_OFF      ioexp_pin_write(5,0)
    #define RELAY_COOL_ON       ioexp_pin_write(6,1)
    #define RELAY_COOL_OFF      ioexp_pin_write(6,0)
    #define RELAY_AUX_ON        ioexp_pin_write(7,1)
    #define RELAY_AUX_OFF       ioexp_pin_write(7,0)

    #define LED_GREEN_ON        ioexp_pin_write(0,1)
    #define LED_GREEN_OFF       ioexp_pin_write(0,0)
    #define LED_RED_ON          ioexp_pin_write(1,1)
    #define LED_RED_OFF         ioexp_pin_write(1,0)
    #define LED_BLUE_ON         ioexp_pin_write(2,1)
    #define LED_BLUE_OFF        ioexp_pin_write(2,0)
    #define LED_AMBER_ON        ioexp_pin_write(3,1)
    #define LED_AMBER_OFF       ioexp_pin_write(3,0)
    #define LED_WHITE_ON        bit_set(PORTD,5)
    #define LED_WHITE_OFF       bit_clr(PORTD,5)
    #define LED_WHITE_TOG       bit_tog(PORTD,5)

    #define LED_PWM             OCR2B
    #define LED_PWM_MAX         255
    #define LED_PWM_MIN         0

    #define CHAR    8
    #define INT     16

    // Modes that the system and fan can be in
    #define SYS_OFF     0
    #define SYS_HEAT    1
    #define SYS_COOL    2
    #define SYS_AUTO    3

    #define FAN_AUTO    0
    #define FAN_ON      1
    #define FAN_CIRC    2

    #define MANUAL      0
    #define PROGRAM1    1
    #define PROGRAM2    2
    #define OVERRIDE    3

    #define SETTINGS    6

    #define TEMPERATURE_MAX         400
    #define TEMPERATURE_MIN         40
    #define TEMPERATURE_UNLOCKED    0
```

```
    #define TEMPERATURE_RESTRICT    1
    #define TEMPERATURE_LOCKED      2

    // ADC macros
    #define ADC_ENABLE          bit_set(ADCSRA, 7)
    #define ADC_DISABLE         bit_clr(ADCSRA, 7)
    #define ADC_START           bit_set(ADCSRA, 6)

    #define tcp_ip_0            reg10to19[0][0]
    #define tcp_ip_1            reg10to19[0][1]
    #define tcp_ip_2            reg10to19[0][2]
    #define tcp_ip_3            reg10to19[0][3]
    #define tcp_port_h          reg10to19[0][4]
    #define tcp_port_l          reg10to19[0][5]

    #define lcd_contrast        reg20to29[0][0]

    #define lcd_light_level     reg20to29[0][1]
    #define lcd_light_min       reg20to29[1][1]
    #define lcd_light_max       reg20to29[2][1]

    #define lcd_light_mode      reg20to29[0][2]
    #define lcd_light_time      reg20to29[0][3]
    #define led_level           reg20to29[0][4]
    #define led_mode            reg20to29[0][5]
    #define led_time            reg20to29[0][6]

    #define temperature_units   reg30to39[0][0]
    #define temperature_gap     reg30to39[0][1]
    #define temperature_offset  reg30to39[0][2]
    #define humidity_offset     reg30to39[0][3]

    #define second              reg90to99[0][0]
    #define minute              reg90to99[0][1]
    #define hour                reg90to99[0][2]
    #define day                 reg90to99[0][3]
    #define date                reg90to99[0][4]
    #define month               reg90to99[0][5]
    #define year                reg90to99[0][6]

    #define pgm_mode            reg100to109[0][0]
    #define system_mode         reg100to109[0][1]
    #define fan_mode            reg100to109[0][2]

    #define temperature_now     reg110to119[0][0]
    #define temperature_set     reg110to119[0][1]
    #define temperature_base    reg110to119[0][2]

extern volatile void *regs[12];
extern unsigned char  temperature_lock, temperature_variance, temperature_override;

extern unsigned int   humidity_now, humidity_set;
extern unsigned int   MYWWWPORT;
extern volatile unsigned char
        pgm_mode_last, latch,

        swing_heat_low, swing_heat_high,
        heat_on_time,   heat_min_on_time,
        heat_off_time,  heat_min_off_time,

        swing_cool_low, swing_cool_high,
```

```
        cool_on_time,   cool_min_on_time,
        cool_off_time,  cool_min_off_time;

extern unsigned char reg10to19[3][7];
extern unsigned char reg20to29[3][7];
extern unsigned char reg30to39[3][5];
extern unsigned char reg100to109[3][3];
extern unsigned int  reg110to119[3][3];
extern unsigned char reg90to99[3][8];

extern volatile unsigned char icon_flags;
void delay (unsigned long time);

#endif
```

```
// ============================================================================
// THAT Home Automation Topology -- Digital Thermostat Module
// This file is part of Firmware version 0.1.8 -- 2010.05.01
// Copyright (C) Nick Viera  ( http://www.nickviera.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.

   You should have received a copy of the GNU General Public License
   along with this program. If not, see <http://www.gnu.org/licenses/> */
// ============================================================================

    #include <avr/io.h>
    #include "menu_settings.h"
    #include "main.h"
    #include "interrupts.h"
    #include "ip_arp_udp_tcp.h"
    #include "lcd.h"
    #include "screen.h"
    #include "eemem.h"
    #include "text.h"

volatile unsigned char reg, regdata;

// dir: 0 = --, 1 = ++, 2 = just check value
unsigned int change_integer
   (unsigned char dir, unsigned int regvalue,
    unsigned int min, unsigned int max)
{
    signed int tmp=0;

    if (dir == 0)
    {
        tmp = (regvalue - 1);
        if (tmp >= min) regvalue--;
    }
    else if (dir == 1)
    {
        tmp = (regvalue + 1);
        if (tmp <= max) regvalue++;
    }
    else if (dir == 2)
    {
        if ((regvalue < min) || (regvalue > max))
            return(32000);
    }

    return(regvalue);
}

void registers_update(void)
{
```

```
    LED_PWM     = led_level;
    BKLIGHT_PWM = lcd_light_level;
    return;
}

signed char register_rw
    (unsigned char rw, signed int dir, unsigned char regnum)
{
    unsigned int  i=0;
    unsigned char plength=CHAR;
    unsigned char *pval    , *pmin    , *pmax;
    unsigned int  *pval_int, *pmin_int, *pmax_int;

    registers_update();

    if ((regnum >= 20) && (regnum <= 26))
    {
        i = regnum - 20;
        pval = &(reg20to29[0][0]);
        pmin = &(reg20to29[1][0]);
        pmax = &(reg20to29[2][0]);
    }

    else if ((regnum >= 30) && (regnum <= 38))
    {
        i = regnum - 30;
        pval = &(reg30to39[0][0]);
        pmin = &(reg30to39[1][0]);
        pmax = &(reg30to39[2][0]);
    }

    else if ((regnum >= 90) && (regnum <= 96))
    {
        i = regnum - 90;
        pval = &(reg90to99[0][0]);
        pmin = &(reg90to99[1][0]);
        pmax = &(reg90to99[2][0]);
    }

    else if ((regnum >= 100) && (regnum <= 109))
    {
        i = regnum - 100;
        pval = &(reg100to109[0][0]);
        pmin = &(reg100to109[1][0]);
        pmax = &(reg100to109[2][0]);
    }


    else if ((regnum >= 110) && (regnum <= 112))
    {
        plength = INT;
        i = regnum - 110;
        pval_int = &(reg110to119[0][0]);
        pmin_int = &(reg110to119[1][0]);
        pmax_int = &(reg110to119[2][0]);
    }

    else return(0);

    if (plength == CHAR)
    {
```

```
        // Adjust the pointers to reference the correct element of the
        // array for the given register number.
        pval = pval + i;
        pmin = pmin + i;
        pmax = pmax + i;

        if      (rw == 1) *pval = change_integer(dir, *pval, *pmin, *pmax);
        else if (rw == 2)
        {
            // Now dir is being used to send the new data!
            // if i == 32000, the requested setting was too high or low
            i = change_integer(2, dir, *pmin, *pmax);
            if (i < 32000) *pval = i;
        }
        return(*pval);
    }

    else if (plength == INT)
    {
        // Adjust the pointers to reference the correct element of the
        // array for the given register number.
        pval_int = pval_int + i;
        pmin_int = pmin_int + i;
        pmax_int = pmax_int + i;
        if      (rw == 1) *pval_int = change_integer(dir, *pval_int, *pmin_int, *pmax_int);
        else if (rw == 2)
        {
            // Now dir is being used to send the new data!
            // if i == 32000, the requested setting was too high or low
            i = change_integer(2, dir, *pmin_int, *pmax_int);
            if (i < 32000) *pval_int = i;
        }
        return(*pval_int);
    }

    return(0);
}

void menu_settings(void)
{
    if (latch == 0)
    {
        bit_clr(switch_flags, INT_PWRSW);
        draw_menu_common();
        regdata = register_rw(0, 0, reg);
    }

    if (bit_get(switch_flags, INT_PWRSW))
    {
        //Exit menu and go back to previous mode
        latch = 0;

        // FIXME - cleanup
        display_clear();
        draw_menu_common();
        lcd_set_rowpage(6);
        lcd_set_column(25);
        draw_string_P(str_writing);
        lcd_set_rowpage(7);
        lcd_set_column(25);
        draw_string_P(str_memory);
```

```
        write_settings_all();
        delay(400);

        pgm_mode = pgm_mode_last;
        display_clear();
        draw_main_screen();
        bit_clr(switch_flags, INT_PWRSW);
        return;
    }

    else if (bit_get(switch_flags, INT_LEFTSW))
    {
        if (reg > 1) reg--;
        regdata = register_rw(0, 0, reg);
        bit_clr(switch_flags, INT_LEFTSW);
    }

    else if (bit_get(switch_flags, INT_RIGHTSW))
    {
        if (reg < 99) reg++;
        regdata = register_rw(0, 0, reg);
        bit_clr(switch_flags, INT_RIGHTSW);
    }

    else if (bit_get(switch_flags, INT_UPSW))
    {
        regdata = register_rw(1, 1, reg);
        bit_clr(switch_flags, INT_UPSW);
    }

    else if (bit_get(switch_flags, INT_DOWNSW))
    {
        regdata = register_rw(1, 0, reg);
        bit_clr(switch_flags, INT_DOWNSW);
    }

    print_register(reg);
    print_regvalue(regdata);
    latch = 1;
    return;
}
```

menu_settings.h

```
#ifndef SETTT_H
#define SETTT_H

signed char   register_rw      (unsigned char rw, signed int dir, unsigned char regnum);
void          menu_settings    (void);
void          registers_update (void);

#endif
```

```
/*********************************************
 * vim:sw=8:ts=8:si:et
 * To use the above modeline in vim you must have "set modeline" in your .vimrc
 * Author: Guido Socher
 * Copyright: GPL V2
 *
 * Based on the net.h file from the AVRlib library by Pascal Stang.
 * For AVRlib See http://www.procyonengineering.com/
 * Used with explicit permission of Pascal Stang.
 *
 * Chip type           : ATMEGA88 with ENC28J60
 *********************************************/

// notation: _P = position of a field
//           _V = value of a field

//@{

#ifndef NET_H
#define NET_H

// ******* ETH *******
#define ETH_HEADER_LEN    14
// values of certain bytes:
#define ETHTYPE_ARP_H_V 0x08
#define ETHTYPE_ARP_L_V 0x06
#define ETHTYPE_IP_H_V  0x08
#define ETHTYPE_IP_L_V  0x00
// byte positions in the ethernet frame:
//
// Ethernet type field (2bytes):
#define ETH_TYPE_H_P 12
#define ETH_TYPE_L_P 13
//
#define ETH_DST_MAC 0
#define ETH_SRC_MAC 6


// ******* ARP *******
#define ETH_ARP_OPCODE_REPLY_H_V 0x0
#define ETH_ARP_OPCODE_REPLY_L_V 0x02
#define ETH_ARP_OPCODE_REQ_H_V 0x0
#define ETH_ARP_OPCODE_REQ_L_V 0x01
// start of arp header:
#define ETH_ARP_P 0xe
//
#define ETHTYPE_ARP_L_V 0x06
// arp.dst.ip
#define ETH_ARP_DST_IP_P 0x26
// arp.opcode
#define ETH_ARP_OPCODE_H_P 0x14
#define ETH_ARP_OPCODE_L_P 0x15
// arp.src.mac
#define ETH_ARP_SRC_MAC_P 0x16
#define ETH_ARP_SRC_IP_P 0x1c
#define ETH_ARP_DST_MAC_P 0x20
#define ETH_ARP_DST_IP_P 0x26

// ******* IP *******
```

```
#define IP_HEADER_LEN     20
// ip.src
#define IP_SRC_P 0x1a
#define IP_DST_P 0x1e
#define IP_HEADER_LEN_VER_P 0xe
#define IP_CHECKSUM_P 0x18
#define IP_TTL_P 0x16
#define IP_FLAGS_P 0x14
#define IP_P 0xe
#define IP_TOTLEN_H_P 0x10
#define IP_TOTLEN_L_P 0x11

#define IP_PROTO_P 0x17

#define IP_PROTO_ICMP_V 1
#define IP_PROTO_TCP_V 6
// 17=0x11
#define IP_PROTO_UDP_V 17
// ******* ICMP *******
#define ICMP_TYPE_ECHOREPLY_V 0
#define ICMP_TYPE_ECHOREQUEST_V 8
//
#define ICMP_TYPE_P 0x22
#define ICMP_CHECKSUM_P 0x24
#define ICMP_CHECKSUM_H_P 0x24
#define ICMP_CHECKSUM_L_P 0x25
#define ICMP_IDENT_H_P 0x26
#define ICMP_IDENT_L_P 0x27
#define ICMP_DATA_P 0x2a

// ******* UDP *******
#define UDP_HEADER_LEN    8
//
#define UDP_SRC_PORT_H_P 0x22
#define UDP_SRC_PORT_L_P 0x23
#define UDP_DST_PORT_H_P 0x24
#define UDP_DST_PORT_L_P 0x25
//
#define UDP_LEN_H_P 0x26
#define UDP_LEN_L_P 0x27
#define UDP_CHECKSUM_H_P 0x28
#define UDP_CHECKSUM_L_P 0x29
#define UDP_DATA_P 0x2a

// ******* TCP *******
#define TCP_SRC_PORT_H_P 0x22
#define TCP_SRC_PORT_L_P 0x23
#define TCP_DST_PORT_H_P 0x24
#define TCP_DST_PORT_L_P 0x25
// the tcp seq number is 4 bytes 0x26-0x29
#define TCP_SEQ_H_P 0x26
#define TCP_SEQACK_H_P 0x2a
// flags: SYN=2
#define TCP_FLAGS_P 0x2f
#define TCP_FLAGS_SYN_V 2
#define TCP_FLAGS_FIN_V 1
#define TCP_FLAGS_RST_V 4
#define TCP_FLAGS_PUSH_V 8
#define TCP_FLAGS_SYNACK_V 0x12
#define TCP_FLAGS_ACK_V 0x10
#define TCP_FLAGS_PSHACK_V 0x18
```

```
//  plain len without the options:
#define TCP_HEADER_LEN_PLAIN 20
#define TCP_HEADER_LEN_P 0x2e
#define TCP_WIN_SIZE 0x30
#define TCP_CHECKSUM_H_P 0x32
#define TCP_CHECKSUM_L_P 0x33
#define TCP_OPTIONS_P 0x36
//
#endif
//@}
```

```
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "lcd.h"
#include "main.h"
#include "screen.h"
#include "font.h"
#include "text.h"
#include "icons.h"

void display_clear(void)
{
    unsigned char page;
    unsigned char col;

    for (page=0; page<16; page++)     // Write to page 0 then go to mext page.
    {                                 // 128pixels / 8 per page = 16 pages
        lcd_set_rowpage(page);        // Set page address
        lcd_comm_write(0x10);         // Set column address MSB
        lcd_comm_write(0x00);         // Set column address LSB
        for(col=0; col<128; col++)    // each page has 128 pixel columns
        {
            lcd_data_write(0x00);
            lcd_data_write(0x00);
            lcd_data_write(0x00);
            lcd_data_write(0x00);
        }
    }
    lcd_rowpage = 0;
    lcd_column  = 0;
    return;
}

// Column: 0-127 column# , Value: 0-255 8-pixel data, level: 0-15 gray level
void write_column(unsigned char value, unsigned char level)
{
    // 16 level grayscale; must write each byte 4 times
    if (bit_get(level,3) != 0) lcd_data_write(value);
    else                       lcd_data_write(0x00);
    if (bit_get(level,2) != 0) lcd_data_write(value);
    else                       lcd_data_write(0x00);
    if (bit_get(level,1) != 0) lcd_data_write(value);
    else                       lcd_data_write(0x00);
    if (bit_get(level,0) != 0) lcd_data_write(value);
    else                       lcd_data_write(0x00);
    lcd_column++;
    return;
}

// Column: 0-127 column# , Value: 0-255 8-pixel data, level: 0-15 gray level
void draw_hr(unsigned char row)
{
    unsigned char col=0, temp=0, ddram[4]={0,0,0,0};

    lcd_set_column  (0);
    lcd_set_rowpage (row / 8);
    temp =          (row % 8);
    temp =          (1 << temp);

    ddram[3] = lcd_data_read();     // dummy read!
```

```
    for(col=0;col<128;col++)
    {
        // Read in current column's data
        ddram[3] = lcd_data_read();
        ddram[2] = lcd_data_read();
        ddram[1] = lcd_data_read();
        ddram[0] = lcd_data_read();

        // Go back one column to write to the same column
        // that was just read from. Also do a dummy read.
        lcd_set_column(col);
        ddram[3] = lcd_data_read();

        // Append new data w/o clobbering existing data
        lcd_data_write(ddram[3] | temp);
        lcd_data_write(ddram[2] | temp);
        lcd_data_write(ddram[1] | temp);
        lcd_data_write(ddram[0] | temp);
    }
    return;
}

// x: column to set, yi: initial row page, yf: final row page
void draw_vr(unsigned char x, unsigned char yi, unsigned char yf, unsigned char level)
{
    unsigned char page=0;

    for(page=yi; page<=yf; page++)
    {
        lcd_set_column(x);
        lcd_set_rowpage(page);
        write_column(0xff, level);
    }
    return;
}


void draw_char(unsigned char ch, unsigned char font)
{
    unsigned char x, b;
    unsigned char tmpcol, tmprow;
    const prog_uint8_t* chp;

    if (ch < 32)  ch = 32;

    // Use 5x7 font
    if (font == 0)
    {
        chp = font_5x7_data + 5 * (ch-32);
        for(x=0; x<5; x++)
        {
            b = pgm_read_byte(chp + x);
            write_column(b, 15);
        }
    }

    // Use large font
    else if (font == 1)
    {
        tmpcol = lcd_column;
```

```
        tmprow = lcd_rowpage;

        chp = big_font + 24 * (ch-48);

        for(x=0; x<24; x++)
        {
            if (x == 12)
            {
                lcd_set_rowpage((tmprow + 1));
                lcd_set_column((tmpcol));
            }

            b = pgm_read_byte(chp + x);
            write_column(b, 15);
        }
        // Make sure to restore back to the row we began on
        lcd_set_rowpage(tmprow);
    }
    return;
}

void draw_string(unsigned char *data)
{
    while (*data)
    {
        draw_char(*data, 0);    // The character
        write_column(0, 15); // Trailing blank space
        data++;
    }
    return;
}

void draw_string_P(const unsigned char *data)
{
    while (pgm_read_byte(data))
    {
        draw_char(pgm_read_byte(data), 0);    // The character
        write_column(0x00, 15); // Trailing blank space
        data++;
    }
    return;
}

// Type: 0 = integer, 1 = 1 decimal place, 2 = for Menu (0.0 -> 9.9)
//       3 = integer but no blank spaces, 2-digits (for date-time)
void print_number(unsigned char type, signed int innum)
{
    unsigned char i=0, j=0;
    unsigned int  outnum[6];

    if (innum < 0)
    {
        innum *= (-1);
        draw_char('-',0);
    }

    if (type != 2)
    {
        if (type != 3) outnum[5] = 0;
        outnum[4] = (innum / 1000);
        outnum[3] = (innum / 100);
```

```
        j=5;
    }
    else j=3;

    outnum[2] = (innum / 10);
    outnum[1] = '.';
    outnum[0] = (innum % 10);

    for (i=j; i>0; i--)
    {
        if ((i-1) > 1)
        {
            outnum[i-1] %= 10;
            // For type 0,1 draw ' ' instead of leading 0
            if ((type != 2) || (type != 3))
            {
                if ((outnum[i] == 0) && (outnum[i-1] == 0))
                    draw_char(' ',0);
                else  draw_char(outnum[i-1] + 0x30,0);
            }
            // For type 2, always draw the leading 0
            else draw_char(outnum[i-1] + 0x30,0);
        }

        if      ((i-1) == 1) { if (type == 1) draw_char('.',0); }
        else if ((i-1) == 0) { draw_char(outnum[i-1] + 0x30,0); }
        write_column(0x00, 15); // Trailing blank space
    }
}

void print_big_number(unsigned char type, signed int innum)
{
    unsigned char i=0, tmprow=0;
    unsigned int  outnum[6];

    tmprow = lcd_rowpage;

    if (innum < 0)
    {
        innum *= (-1);
        draw_char('-',0);
    }

    // type 0 = temperature(0.0 - 100.0)
    // type 1 = humidity (10-95)
    outnum[5] = 0;
    outnum[4] = (innum / 1000);
    outnum[3] = (innum / 100);
    outnum[2] = (innum / 10);
    outnum[1] = '.';
    outnum[0] = (innum % 10);

    for (i=5; i>0; i--)
    {
        if ((i-1) > 1)
        {
            outnum[i-1] %= 10;
            // draw leading 0 before .
            if (i != 2)
            {
                if ((type == 1) && (i ==3)) {}
```

```
                else {

                    if ((outnum[i] == 0) && (outnum[i-1] == 0))
                            draw_char(64,1);
                    else  draw_char(outnum[i-1] + 0x30,1);

                }
            }
            else draw_char(outnum[i-1] + 0x30,1);
        }

        if      ((i-1) == 1) { lcd_set_column(lcd_column-2); }//if (type == 1)
draw_char('.',0); }
        else if ((i-1) == 0)
        {
            // Big or small digit at the end depending on type
            if (type == 1) draw_char(outnum[i-1] + 0x30,1);
            else
            {
                lcd_set_rowpage(tmprow + 1);
                draw_char('.',0);
                draw_char(outnum[i-1] + 0x30,0);
                lcd_set_rowpage(tmprow);
                lcd_set_column(lcd_column - 8);
            }
        }
        write_column(0x00, 15); // Trailing blank space
    }
}

void print_mode(void)
{
    lcd_set_column  (1);
    lcd_set_rowpage (0);

    switch(system_mode)
    {
        case SYS_OFF:
            draw_string_P (str_off);
            break;
        case SYS_AUTO:
            draw_string_P (str_auto);
            break;
        case SYS_HEAT:
            draw_string_P (str_heat);
            break;
        case SYS_COOL:
            draw_string_P (str_cool);
            break;
    }
    return;
}

void print_fan(void)
{
    lcd_set_column  (38);
    lcd_set_rowpage (0);

    switch(fan_mode)
    {
        case FAN_AUTO:
```

```
            draw_string_P (str_auto);
            break;
        case FAN_ON:
            draw_string_P (str_on);
            break;
        case FAN_CIRC:
            draw_string_P (str_circ);
            break;
    }
    return;
}

void print_date_time(void)
{

    lcd_set_column  (1);
    lcd_set_rowpage (15);

    draw_string_P (str_mon);
    draw_string   (". ");

    return;
}

void print_temperature(signed int temperature)
{
    signed int tmp=0;

    lcd_set_column  (1);
    lcd_set_rowpage (5);

    if (temperature_units == 1) tmp = ((1.8 * temperature) + 320);
    else                        tmp = temperature;

    print_big_number(0, tmp);

    draw_char(CH_DEGREE,0);
    if      (temperature_units == 0) draw_char('C',0);
    else if (temperature_units == 1) draw_char('F',0);

    return;
}

void print_setpoint(void)
{
    signed int tmp=0;

    lcd_set_column  (68);
    lcd_set_rowpage  (5);

    if (temperature_units == 1) tmp = ((1.8 * temperature_set) + 320);
    else                        tmp = temperature_set;

    print_big_number(0, tmp);

    draw_char(CH_DEGREE,0);
    if      (temperature_units == 0) draw_char('C',0);
    else if (temperature_units == 1) draw_char('F',0);

    lcd_set_column  (68);
    lcd_set_rowpage  (7);
```

```
    if (temperature_override == 1) draw_string("Override");
    else                           draw_string("        ");

    return;
}

void print_humidity(unsigned int humidity)
{
    lcd_set_column(1);
    lcd_set_rowpage(9);
    print_big_number(1, humidity);
    write_column(0x00, 15);
    draw_char('%',0);
    return;
}

// Mode: 0 = horizontal lines, 1 = vertical lines, 2 = all lines
void draw_main_grid(unsigned char mode)
{

    if ((mode == 0) || (mode == 2))
    {
        draw_hr(13);
        draw_hr(113);
    }

    if ((mode == 1) || (mode == 2))
    {
        draw_vr(62,3,12,6);
        draw_vr(63,3,12,15);
        draw_vr(64,3,12,15);
        draw_vr(65,3,12,6);
    }
    return;
}


void draw_menu_common(void)
{
    display_clear();
    draw_main_grid(0);

    lcd_set_column(25);
    lcd_set_rowpage(0);
    draw_string_P(str_settings);

    lcd_set_column(10);
    lcd_set_rowpage(4);
    draw_string_P(str_register);
    draw_string_P(str_sep);

    lcd_set_column(28);
    lcd_set_rowpage(6);
    draw_string_P(str_value);
    draw_string_P(str_sep);

    lcd_set_column(1);
    lcd_set_rowpage(15);
    draw_char(128, 0);
    draw_string(" Prev");
    lcd_set_column(91);
```

```
    draw_string("Next ");
    draw_char(129, 0);
    return;
}

void print_register(unsigned char reg)
{
    lcd_set_column(68);
    lcd_set_rowpage(4);

    // Print the number as 25 -> 2.5
    print_number(1, reg);
    return;
}

void print_regvalue(unsigned char value)
{
    lcd_set_column(68);
    lcd_set_rowpage(6);

    // Print the number as 25 -> 2.5
    print_number(0, value);
    return;
}

void draw_off_screen(void)
{
    display_clear();
    draw_main_grid(0);
    print_fan();
    print_mode();
    return;
}

void draw_icon(unsigned char drawerase, unsigned char icon)
{
    unsigned char startcol=0, i=0, data=0;
    const         prog_uint8_t* chp;

    lcd_set_rowpage(0);

    switch(icon)
    {
        case ICON_NETWORK:
            startcol = 115;
            break;
        case ICON_LOCK:
            startcol = 100;
            break;
        case ICON_VENT:
            startcol = 85;
            break;
        case ICON_WARN:
            startcol = 70;
            break;
    }

    chp = icons + (24 * icon);
    lcd_set_column(startcol);

    for (i=0; i<24; i++)
```

```
    {
        if (i == 12)
        {
            lcd_set_rowpage(1);
            lcd_set_column(startcol);
        }
        if (drawerase == 1)
        {
            data = pgm_read_byte(chp + i);
            write_column(data, 15);
        }
        else write_column(0x00, 15);
    }
    return;
}

void print_program(void)
{
    lcd_set_rowpage(12);
    lcd_set_column(73);

    switch (pgm_mode)
    {
        case MANUAL:
            draw_string_P(str_manual);
            break;
        case PROGRAM1:
            draw_string_P(str_prog1);
            break;
        case PROGRAM2:
            draw_string_P(str_prog2);
            break;
        case OVERRIDE:
            draw_string_P(str_override);
            break;
    }
    return;
}

void draw_titles(void)
{
    lcd_set_rowpage(3);
    lcd_set_column(11);

    draw_char(128, 0); //<|
    draw_char(' ', 0);
    draw_string_P(str_act);
    draw_char(' ', 0);
    draw_char(129, 0); //|>

    lcd_set_column(79);
    draw_char(128, 0); //<|
    draw_char(' ', 0);
    draw_string_P(str_set);
    draw_char(' ', 0);
    draw_char(129, 0); //|>

    return;
}

void draw_main_screen(void)
```

```
{
    draw_main_grid(2);
    print_fan();
    print_setpoint();
    print_mode();
    print_program();
    draw_titles();
    return;
}

void redraw_screen(void)
{
    if (system_mode == SYS_OFF) draw_off_screen();
    else                        draw_main_screen();
    return;
}
```

screen.h

```
#ifndef SCREEN_H
#define SCREEN_H

#define CH_DEGREE     127
#define ICON_NETWORK  0
#define ICON_LOCK     1
#define ICON_VENT     2
#define ICON_WARN     3

void display_clear     (void);
void draw_hr           (unsigned char row);
void draw_vr           (unsigned char x, unsigned char yi, unsigned char yf, unsigned char
level);
void draw_char         (unsigned char ch, unsigned char font);
void draw_main_grid    (unsigned char mode);
void draw_menu_common  (void);
void draw_string       (unsigned char *data);
void draw_string_P     (const unsigned char *data);
void print_number      (unsigned char type, signed int innum);
void print_date_time   (void);
void print_temperature (signed int temperature);
void print_setpoint    (void);
void print_mode        (void);
void print_fan         (void);
void print_humidity    (unsigned int humidity);
void draw_off_screen   (void);
void draw_main_screen  (void);
void draw_icon         (unsigned char drawerase, unsigned char icon);
void redraw_screen     (void);
void print_register    (unsigned char reg);
void print_regvalue    (unsigned char value);

#endif
```

```
#include <avr/pgmspace.h>

    const unsigned char str_mon[] PROGMEM   = "Mon";
    const unsigned char str_tue[] PROGMEM   = "Tue";
    const unsigned char str_wed[] PROGMEM   = "Wed";
    const unsigned char str_thu[] PROGMEM   = "Thu";
    const unsigned char str_fri[] PROGMEM   = "Fri";
    const unsigned char str_sat[] PROGMEM   = "Sat";
    const unsigned char str_sun[] PROGMEM   = "Sun";

    const unsigned char str_heat[] PROGMEM   = "Heat";
    const unsigned char str_off[]  PROGMEM   = "Off ";
    const unsigned char str_cool[] PROGMEM   = "Cool";
    const unsigned char str_auto[] PROGMEM   = "Auto";

    const unsigned char str_manual[] PROGMEM   = " Manual ";
    const unsigned char str_prog1[]  PROGMEM   = "Prog. #1";
    const unsigned char str_prog2[] PROGMEM    = "Prog. #2";
    const unsigned char str_override[] PROGMEM = "Override";

    const unsigned char str_a1[]   PROGMEM   = "- ";
    const unsigned char str_act[]  PROGMEM   = "NOW";
    const unsigned char str_set[]  PROGMEM   = "SET";
    const unsigned char str_a2[]   PROGMEM   = " -";

    const unsigned char str_on[]   PROGMEM   = "On  ";
    const unsigned char str_circ[] PROGMEM   = "Circ";

    const unsigned char str_settings[] PROGMEM = "Settings Menu";
    const unsigned char str_register[] PROGMEM = "Register";
    const unsigned char str_value[] PROGMEM    = "Value";
    const unsigned char str_sep[] PROGMEM      = " :";

    const unsigned char str_reading[] PROGMEM     = "Reading from";
    const unsigned char str_writing[] PROGMEM     = " Writing to ";
    const unsigned char str_memory[] PROGMEM      = "the memory...";
```

text.h

```
#ifndef TEXT_H
#define TEXT_H

    extern const unsigned char str_mon[];
    extern const unsigned char str_tue[];
    extern const unsigned char str_wed[];
    extern const unsigned char str_thu[];
    extern const unsigned char str_fri[];
    extern const unsigned char str_sat[];
    extern const unsigned char str_sun[];

    extern const unsigned char str_heat[];
    extern const unsigned char str_off[];
    extern const unsigned char str_cool[];
    extern const unsigned char str_auto[];

    extern const unsigned char str_manual[];
    extern const unsigned char str_prog1[];
    extern const unsigned char str_prog2[];
    extern const unsigned char str_override[];

    extern const unsigned char str_a1[];
    extern const unsigned char str_act[];
    extern const unsigned char str_set[];
    extern const unsigned char str_a2[];

    extern const unsigned char str_on[];
    extern const unsigned char str_circ[];

    extern const unsigned char str_settings[];
    extern const unsigned char str_register[];
    extern const unsigned char str_value[];
    extern const unsigned char str_sep[];

    extern const unsigned char str_reading[];
    extern const unsigned char str_writing[];
    extern const unsigned char str_memory[];

#endif
```

timeout.h

```
/* vim: set sw=8 ts=8 si et: */
#define F_CPU 12500000UL  // 12.5 MHz
#ifndef ALIBC_OLD
#include <util/delay.h>
#else
#include <avr/delay.h>
#endif
```

Appendix D: Electronic Access Module Firmware Source Code

main.h

```
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// THAT Home Automation Topology * Electronic Access Module
// Microcontroller Firmware version 1.0 * 2010.04.08
// Chris Miller * busybot.org
//
/* 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/> */
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#ifndef MAIN_H
#define MAIN_H

    void delay(unsigned long time);

    // Bitwise operation macros
    #define BIT(x)              (0x01 << (x))
    #define bit_get(p,m)        ((p) &  _BV(m))
    #define bit_grab(p,m)       (((p) &  _BV(m)) >> m)
    #define bit_set(p,m)        ((p) |=  _BV(m))
    #define bit_clr(p,m)        ((p) &=~ _BV(m))
    #define bit_tog(p,m)        ((p) ^=  _BV(m))

    #define bit_write(c,p,m)    (c ? bit_set(p,m) : bit_clr(p,m))
    #define LONGBIT(x)          ((unsigned long)0x00000001 << (x))

    #define setbit(addr,bit)    (addr |= (1<<bit))
    #define clearbit(addr,bit)  (addr &= ~(1<<bit))
    #define flipbit(addr,bit)   (addr ^= (1<<bit))
    #define checkbit(addr,bit)  (addr & (1<<bit))

    #define setbitmask(x,y)     (x |= (y))
    #define clearbitmask(x,y)   (x &= (~y))
    #define flipbitmask(x,y)    (x ^= (y))
    #define checkbitmask(x,y)   (x & (y))

    #define GREEN_ON            bit_set(PORTA, 0)
    #define GREEN_OFF           bit_clr(PORTA, 0)
    #define GREEN_TOG           bit_tog(PORTA, 0)
    #define RED_ON              bit_set(PORTA, 2)
    #define RED_OFF             bit_clr(PORTA, 2)
    #define RED_TOG             bit_tog(PORTA, 2)
    #define LOCKED          0
    #define UNLOCKED        4
    #define DOOR_CLOSED     8
    #define DOOR_OPENED     12
    #define KEY_OK          16
    #define KEY_BAD         20
```

```
    #define BELL_OFF    140
    #define BELL_ON     142
    #define NET_OFF     144
    #define NET_ON      146
    #define DW_OFF      148
    #define DW_ON       150
    #define ARROW       131
    #define SCOL1               0x01
    #define SCOL2               0x02
    #define SCOL3               0x04
    #define SCOL4               0x08
    #define SROW1               0x10
    #define SROW2               0x20
    #define SROW3               0x40
    #define SROW4               0x80
    #define RELAY1_ON           bit_set(PORTC, 2)
    #define RELAY1_OFF          bit_clr(PORTC, 2)
    #define RELAY1_TOG          bit_tog(PORTC, 2)
    #define RELAY2_ON           bit_set(PORTC, 1)
    #define RELAY2_OFF          bit_clr(PORTC, 1)
    #define RELAY2_TOG          bit_tog(PORTC, 1)
    #define RELAY3_ON           bit_set(PORTC, 0)
    #define RELAY3_OFF          bit_clr(PORTC, 0)
    #define RELAY3_TOG          bit_tog(PORTC, 0)
    #define LED1_G_ON           bit_set(PORTA, 0)
    #define LED1_G_OFF          bit_clr(PORTA, 0)
    #define LED1_G_TOG          bit_tog(PORTA, 0)
    #define LED1_R_ON           bit_set(PORTA, 1)
    #define LED1_R_OFF          bit_clr(PORTA, 1)
    #define LED1_R_TOG          bit_tog(PORTA, 1)
    #define LED2_G_ON           bit_set(PORTA, 2)
    #define LED2_G_OFF          bit_clr(PORTA, 2)
    #define LED2_G_TOG          bit_tog(PORTA, 2)
    #define LED2_R_ON           bit_set(PORTA, 3)
    #define LED2_R_OFF          bit_clr(PORTA, 3)
    #define LED2_R_TOG          bit_tog(PORTA, 3)
    #define LED3_G_ON           bit_set(PORTA, 4)
    #define LED3_G_OFF          bit_clr(PORTA, 4)
    #define LED3_G_TOG          bit_tog(PORTA, 4)
    #define LED3_R_ON           bit_set(PORTA, 5)
    #define LED3_R_OFF          bit_clr(PORTA, 5)
    #define LED3_R_TOG          bit_tog(PORTA, 5)
#endif
```

```
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// THAT Home Automation Topology * Electronic Access Module
// Microcontroller Firmware version 1.0 * 2010.04.08
// Chris Miller * busybot.org
//
/* 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/> */
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#include <avr/io.h>
#include <stdlib.h>
//#include <avr/portpins.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/eeprom.h>

#include "lcd.h"
#include "main.h"
#include "screen.h"
#include "interrupts.h"

extern volatile unsigned char switch_flags;
extern volatile unsigned char key_content;
volatile unsigned char relay_status, door_status, relay_status_flag;
volatile unsigned char mode = 1;
volatile unsigned char mode_flag;
volatile unsigned char relay_setting, relay_setting_flag;
volatile unsigned char bell = BELL_ON;
volatile unsigned char wait_flag;

static unsigned char name1[] PROGMEM = "Front   ";
static unsigned char name2[] PROGMEM = "Back    ";
static unsigned char name3[] PROGMEM = "Garage  ";
static unsigned char no[] PROGMEM = "NO";
static unsigned char nc[] PROGMEM = "NC";

void init(void)
{
    // Initial Port Setup (ATmega324p)
    bit_set(DDRA, 0);   // Pin40  Output  [LED1 G]
    bit_set(DDRA, 1);   // Pin39  Output  [LED1 R]
    bit_set(DDRA, 2);   // Pin38  Output  [LED2 G]
    bit_set(DDRA, 3);   // Pin37  Output  [LED2 R]
    bit_set(DDRA, 4);   // Pin36  Output  [LED3 G]
    bit_set(DDRA, 5);   // Pin35  Output  [LED3 R]
    bit_set(DDRA, 6);   // Pin34  Output  []
    bit_set(DDRA, 7);   // Pin33  Output  []

    bit_set(DDRB, 0);   // Pin01  Output  [piezo speaker]
```

```
bit_set(DDRB, 1);   // Pin02  Output  [piezo speaker]
bit_set(DDRB, 2);   // Pin03  Output  []
bit_set(DDRB, 3);   // Pin04  Output  []
bit_set(DDRB, 4);   // Pin05  Output  []
bit_set(DDRB, 5);   // Pin06  Output  []
bit_set(DDRB, 6);   // Pin07  Output  []
bit_set(DDRB, 7);   // Pin08  Output  []

bit_set(DDRC, 0);   // Pin22  Output  []
bit_set(DDRC, 1);   // Pin23  Output  []
bit_set(DDRC, 2);   // Pin24  Output  []
bit_set(DDRC, 3);   // Pin25  Output  []
bit_set(DDRC, 4);   // Pin26  Output  []
bit_set(DDRC, 5);   // Pin27  Output  [RELAY1]
bit_set(DDRC, 6);   // Pin28  Output  [RELAY2]
bit_set(DDRC, 7);   // Pin29  Output  [RELAY3]

bit_clr(DDRD, 0);   // Pin14  Input   [Switch Col 1]
bit_clr(DDRD, 1);   // Pin15  Input   [Switch Col 2]
bit_clr(DDRD, 2);   // Pin16  Input   [Switch Col 3]
bit_clr(DDRD, 3);   // Pin17  Input   [Switch Col 4]
bit_clr(DDRD, 4);   // Pin18  Input   [Switch Row 1]
bit_clr(DDRD, 5);   // Pin19  Input   [Switch Row 2]
bit_clr(DDRD, 6);   // Pin20  Input   [Switch Row 3]
bit_clr(DDRD, 7);   // Pin21  Input   [Switch Row 4]

bit_clr(PORTA, 0);  // Pin40  Low  [LED1 G]
bit_clr(PORTA, 1);  // Pin39  Low  [LED1 R]
bit_clr(PORTA, 2);  // Pin38  Low  [LED2 G]
bit_clr(PORTA, 3);  // Pin37  Low  [LED2 R]
bit_clr(PORTA, 4);  // Pin36  Low  [LED3 G]
bit_clr(PORTA, 5);  // Pin35  Low  [LED3 R]
bit_clr(PORTA, 6);  // Pin34  Low  []
bit_clr(PORTA, 7);  // Pin33  Low  []

bit_set(PORTB, 0);  // Pin01  High [piezo speaker]
bit_clr(PORTB, 1);  // Pin02  Low  [piezo speaker]
bit_clr(PORTB, 2);  // Pin03  Low  []
bit_clr(PORTB, 3);  // Pin04  Low  []
bit_clr(PORTB, 4);  // Pin05  Low  []
bit_clr(PORTB, 5);  // Pin06  Low  []
bit_clr(PORTB, 6);  // Pin07  Low  []
bit_clr(PORTB, 7);  // Pin08  Low  []

bit_clr(PORTC, 0);  // Pin29  Low  []
bit_clr(PORTC, 1);  // Pin28  Low  []
bit_clr(PORTC, 2);  // Pin27  Low  []
bit_clr(PORTC, 3);  // Pin26  Low  []
bit_clr(PORTC, 4);  // Pin25  Low  []
bit_clr(PORTC, 5);  // Pin24  Low  [RELAY1]
bit_clr(PORTC, 6);  // Pin23  Low  [RELAY2]
bit_clr(PORTC, 7);  // Pin22  Low  [RELAY3]

bit_set(PORTD, 0);  // Pin14  Low  [Switch Col 1]
bit_set(PORTD, 1);  // Pin15  Low  [Switch Col 2]
bit_set(PORTD, 2);  // Pin16  Low  [Switch Col 3]
bit_set(PORTD, 3);  // Pin17  Low  [Switch Col 4]
bit_set(PORTD, 4);  // Pin18  High [Switch Row 1]
bit_set(PORTD, 5);  // Pin19  High [Switch Row 2]
bit_set(PORTD, 6);  // Pin20  High [Switch Row 3]
bit_set(PORTD, 7);  // Pin21  High [Switch Row 4]
```

```
    //// Initialize Timer0
    TCCR0A = 0b00000000;    // No Waveform Generation
    TCCR0B = 0b00000000;    // Start with timer stopped
    TIMSK0 = 0b00000001;    // Bit 0 : Enable Overflow interrupt
    //Initialize Counter
    TCNT0=0;

    //// Initialize Pin Change Interrupts
    //PCMSK0 = 0b00000001;     // Enable PCINT0 (Pin40)
    //PCICR  = 0b00000001;     // Enable Pin change interrupt (for PCINT0-PCINT7)
    KEYPAD_ENABLE;

    PCICR  = 0b00001000;    // Enable Pin change interrupt (for PCINT24-PCINT31)

    //Enable Global Interrupts
    sei();

}

void delay(unsigned long time)
{
    unsigned long i=0;
    for(i=0; i<=time; i++) asm volatile("nop");
}

void chirp(void)
{
    if (bell == BELL_ON)
    {
        unsigned char i = 0;
        for (i=0;i<=20;i++)
        {
            bit_set(PORTB, 0);  // Pin40  High
            bit_clr(PORTB, 1);  // Pin39  Low
            delay(100);
            bit_clr(PORTB, 0);  // Pin40  Low
            bit_set(PORTB, 1);  // Pin39  High
            delay(100);
        }
    }
}

void draw_main(void)
{
    screen_clear();

    //draw_char_c(ARROW,0,0);
    draw_string_c("Main",0,0);
    draw_byte_stretch(1,0,127,2);

    draw_dual_p(bell, 0, 96);
    draw_dual_p(NET_ON, 0, 107);
    draw_dual_p(DW_ON, 0, 118);

    draw_string_pc(name1,3,4);
    if (!(bit_get(relay_status,0)))
        draw_icon(LOCKED,2,17);
    else
        draw_icon(UNLOCKED,2,17);
```

```
    if (!(bit_get(door_status,0)))
        draw_icon(DOOR_CLOSED,2,20);
    else
        draw_icon(DOOR_OPENED,2,20);

    draw_string_pc(name2,5,4);
    if (!(bit_get(relay_status,1)))
        draw_icon(LOCKED,4,17);
    else
        draw_icon(UNLOCKED,4,17);
    if (!(bit_get(door_status,1)))
        draw_icon(DOOR_CLOSED,4,20);
    else
        draw_icon(DOOR_OPENED,4,20);

    draw_string_pc(name3,7,4);
    //draw_icon(KEY_OK,6,14);
    if (!(bit_get(relay_status,2)))
        draw_icon(LOCKED,6,17);
    else
        draw_icon(UNLOCKED,6,17);
    if (!(bit_get(door_status,2)))
        draw_icon(DOOR_CLOSED,6,20);
    else
        draw_icon(DOOR_OPENED,6,20);
}

void draw_settings(void)
{
    screen_clear();

    //draw_char_c(ARROW,0,0);
    draw_string_c("Settings",0,0);
    draw_byte_stretch(1,0,127,2);

    draw_dual_p(bell, 0, 96);
    draw_dual_p(NET_ON, 0, 107);
    draw_dual_p(DW_ON, 0, 118);

    draw_string_c("NO/NC",2,16);

    draw_string_c("Front",3,4);
    if ((bit_get(relay_setting,0)))
    {
        //draw_string_c("NC",3,17);
        draw_char_c(' ',3,17);
        draw_char_c(132,3,20);
        LED1_G_OFF;
        LED1_R_ON;
    }
    else
    {
        //draw_string_c("NC",3,17);
        draw_char_c(' ',3,20);
        draw_char_c(132,3,17);
        LED1_G_ON;
        LED1_R_OFF;
    }

    draw_string_c("Back",5,4);
    if ((bit_get(relay_setting,1)))
```

```
    {
        //draw_string_c("NC",3,17);
        draw_char_c(' ',5,17);
        draw_char_c(132,5,20);
        LED2_G_OFF;
        LED2_R_ON;
    }
    else
    {
        //draw_string_c("NC",3,17);
        draw_char_c(' ',5,20);
        draw_char_c(132,5,17);
        LED2_G_ON;
        LED2_R_OFF;
    }

    draw_string_c("Garage",7,4);
    if ((bit_get(relay_setting,2)))
    {
        //draw_string_c("NC",3,17);
        draw_char_c(' ',7,17);
        draw_char_c(132,7,20);
        LED3_G_OFF;
        LED3_R_ON;
    }
    else
    {
        //draw_string_c("NC",3,17);
        draw_char_c(' ',7,20);
        draw_char_c(132,7,17);
        LED3_G_ON;
        LED3_R_OFF;
    }
}

int main(void)
{
    delay(80000);
    init();
    delay(50000);
    lcd_init();
    //keypad_init();

    mode_flag = 1;
    mode = 1;

    LED1_G_ON;
    LED2_G_ON;
    LED3_G_ON;
    LED1_R_OFF;
    LED2_R_OFF;
    LED3_R_OFF;

    chirp();
    delay(1000);
    chirp();

    unsigned int i;
    unsigned char x = 0;
    unsigned char y = 0;
    unsigned char z = 0;
```

```
unsigned char key = 0;

while(1)
{
    switch (switch_flags)
    {
        case 0x00:  // None
                break;
        case 0x01:  // Mode
                key = 'M';
                switch_flags = 0;
                mode_flag = 1;
                chirp();
                break;
        case 0x02:  // Quick Open 1
                key = '1';
                switch_flags = 0;
                //chirp();
                break;
        case 0x04:  // Quick Open 2
                key = '2';
                switch_flags = 0;
                //chirp();
                break;
        case 0x08: // Quick Open 3
                key = '3';
                switch_flags = 0;
                //chirp();
                break;
        case 0x10:  // Bell
                key = 'B';
                switch_flags = 0;
                if (bell == BELL_ON)
                    bell = BELL_OFF;
                else
                    bell = BELL_ON;
                draw_dual_p(bell, 0, 96);
                chirp();
                break;
        case 0x20:  // Toggle 1
                key = 'X';
                switch_flags = 0;
                //chirp();
                break;
        case 0x40:  // Toggle 2
                key = 'Y';
                switch_flags = 0;
                //chirp();
                break;
        case 0x80:  // Toggle 3
                key = 'Z';
                switch_flags = 0;
                //chirp();
                break;
        default:    // Error
                key = '?';
                switch_flags = 0;
                //chirp();
                break;
    }
```

```
if (mode_flag)
{
    switch (mode)
    {
        case 1:
            mode = 2;   mode_flag = 0;
            draw_main();
            break;
        case 2:
            mode = 1;   mode_flag = 0;
            draw_settings();
            break;
        default:
            break;
    }
}

switch (key)
{
    case '1':
        if (mode == 2) // Main
        {
            bit_tog(relay_setting,0);
            relay_status_flag = 1;
            chirp();
        }
        key = 0;
        break;
    case '2':
        if (mode == 2) // Main
        {
            bit_tog(relay_setting,1);
            relay_status_flag = 2;
            chirp();
        }
        key = 0;
        break;
    case '3':
        if (mode == 2) // Main
        {
            bit_tog(relay_setting,2);
            relay_status_flag = 3;
            chirp();
        }
        key = 0;
        break;
    case 'X':
        if (mode == 1) // Settings
        {

            relay_setting_flag = 1;
            chirp();
        }
        key = 0;
        break;
    case 'Y':
        if (mode == 1) // Settings
        {

            relay_setting_flag = 2;
            chirp();
```

```
        }
        key = 0;
        break;
    case 'Z':
        if (mode == 1) // Settings
        {

            relay_setting_flag = 3;
            chirp();
        }
        key = 0;
        break;
    default:
        break;
}

switch (relay_setting_flag)
{
    case 1:
        //LED1_R_TOG;
        //RELAY1_TOG;
        bit_tog(relay_setting,0);
        relay_setting_flag = 0;
        if ((bit_get(relay_setting,0)))
        {
            RELAY1_TOG;
            draw_char_c(' ',3,17);
            draw_char_c(132,3,20);
            LED1_G_OFF;
            LED1_R_ON;
        }
        else
        {
            RELAY1_TOG;
            draw_char_c(' ',3,20);
            draw_char_c(132,3,17);
            LED1_G_ON;
            LED1_R_OFF;
        }
        break;
    case 2:
        //LED2_R_TOG;
        //RELAY2_TOG;
        bit_tog(relay_setting,1);
        relay_setting_flag = 0;
        if ((bit_get(relay_setting,1)))
        {
            RELAY2_TOG;
            draw_char_c(' ',5,17);
            draw_char_c(132,5,20);
            LED2_G_OFF;
            LED2_R_ON;
        }
        else
        {
            RELAY2_TOG;
            draw_char_c(' ',5,20);
            draw_char_c(132,5,17);
            LED2_G_ON;
            LED2_R_OFF;
        }
```

```
        break;
    case 3:
        //LED3_R_TOG;
        //RELAY3_TOG;
        bit_tog(relay_setting,2);
        relay_setting_flag = 0;
        if ((bit_get(relay_setting,2)))
        {
            RELAY3_TOG;
            draw_char_c(' ',7,17);
            draw_char_c(132,7,20);
            LED3_G_OFF;
            LED3_R_ON;
        }
        else
        {
            RELAY3_TOG;
            draw_char_c(' ',7,20);
            draw_char_c(132,7,17);
            LED3_G_ON;
            LED3_R_OFF;
        }
        break;
    default:    break;
}

switch (relay_status_flag)
{
    case 1:
        RELAY1_TOG;
        bit_tog(relay_status,0);
        relay_status_flag = 0;
        if (!(bit_get(relay_status,0)))
            draw_icon(LOCKED,2,17);
        else
            draw_icon(UNLOCKED,2,17);
        //LED1_R_ON;
        //LED1_G_ON;
        //wait_flag = 1;
        break;
    case 2:
        //LED2_G_TOG;
        RELAY2_TOG;
        bit_tog(relay_status,1);
        relay_status_flag = 0;
        if (!(bit_get(relay_status,1)))
            draw_icon(LOCKED,4,17);
        else
            draw_icon(UNLOCKED,4,17);
        //LED2_R_ON;
        //LED2_G_ON;
        //wait_flag = 1;
        break;
    case 3:
        //LED3_G_TOG;
        RELAY3_TOG;
        bit_tog(relay_status,2);
        relay_status_flag = 0;
        if (!(bit_get(relay_status,2)))
            draw_icon(LOCKED,6,17);
        else
```

```
                    draw_icon(UNLOCKED,6,17);
                //LED3_R_ON;
                //LED3_G_ON;
                //wait_flag = 1;
                break;
            default:    break;
        }
    }
    return 0;
}
```

interrupts.h

```
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// THAT Home Automation Topology * Electronic Access Module
// Microcontroller Firmware version 1.0 * 2010.04.08
// Copyright (C) Chris Miller * busybot.org
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#ifndef INTERRUPTS_H
#define INTERRUPTS_H

#define     TIMER_INT_START     TCCR0B |= 0b00000011
#define     TIMER_INT_STOP      TCCR0B &= 0b11111000
#define     TIMER_INT_COUNT     TCNT0

#define INT_TEST_ENABLE         bit_set(PCMSK0, 0)
#define INT_TEST_DISABLE        bit_clr(PCMSK0, 0)

#define KEYPAD_ENABLE         PCMSK3 = 0b11111111
#define KEYPAD_DISABLE        PCMSK3 = 0b00000000


#endif
```

interrupts.c

```
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// THAT Home Automation Topology * Electronic Access Module
// Microcontroller Firmware version 1.0 * 2010.04.08
// Copyright (C) Chris Miller * busybot.org
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#include <avr/io.h>
#include <avr/interrupt.h>

#include "main.h"
#include "interrupts.h"

// Global variables
volatile unsigned char  t0_count;
volatile unsigned char  switch_flags;
volatile unsigned char  key_content, key_char;


// ISR (PCINT0_vect)
// {
//     // Action Switch
//     INT_TEST_DISABLE;     // Disable PCINT0 (Pin40)
//     switch_flags = 250;
//     TIMER_INT_START;
//     return;
// }

ISR (PCINT3_vect)  // Switch Row 1
{
    KEYPAD_DISABLE;

    switch_flags = ~PIND;

    TIMER_INT_START;

    return;
}

//Timer0 overflow interrupt service routine.
//Timer0 overflow interrupt service routine.
ISR (TIMER0_OVF_vect)
{
    t0_count++;
    if(t0_count >= 15)
    {
        // Reset timer
        TIMER_INT_STOP;
        TIMER_INT_COUNT=0;
        t0_count = 0;

        //Reenable switch interrupts
        //INT_TEST_ENABLE;     // Disable PCINT0 (Pin40)
        KEYPAD_ENABLE;

    }
    return;
}
```

lcd.h

```
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// THAT Home Automation Topology * Electronic Access Module
// Microcontroller Firmware version 1.0 * 2010.04.08
// Copyright (C) Chris Miller * busybot.org
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#ifndef LCD_H
#define LCD_H
    #define LCD_LISTEN_ON       bit_clr(PORTB, 2)
    #define LCD_LISTEN_OFF      bit_set(PORTB, 2)
    #define LCD_RESET_SET       bit_set(PORTB, 3)
    #define LCD_RESET_CLR       bit_clr(PORTB, 3)
    #define LCD_DATA_SEL        bit_set(PORTB, 4)
    #define LCD_COMM_SEL        bit_clr(PORTB, 4)
    void lcd_comm(unsigned char j);
    void lcd_data(unsigned char j);
    void lcd_col(unsigned char column);
    void lcd_row(unsigned char row);
    void lcd_init(void);
#endif
```

lcd.c

```
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// THAT Home Automation Topology * Electronic Access Module
// Microcontroller Firmware version 1.0 * 2010.04.08
// Copyright (C) Chris Miller * busybot.org
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#include <avr/io.h>
#include <stdlib.h>
//#include <avr/portpins.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

#include "lcd.h"
#include "main.h"
#include "screen.h"


void lcd_comm(unsigned char j)
{
    LCD_LISTEN_ON;
    LCD_COMM_SEL;
    bit_clr(SPSR, 7); // Clear SPI interrupt flag
    SPDR = j; // Write byte out
    while(!(bit_get(SPSR,7))) { } // Wait to finish
    LCD_LISTEN_OFF;
    return;
}

void lcd_data(unsigned char j)
{
    LCD_LISTEN_ON;
```

```
    LCD_DATA_SEL;
    bit_clr(SPSR, 7); // Clear SPI interrupt flag
    SPDR = j; // Write byte out
    while(!(bit_get(SPSR,7))) { } // Wait to finish
    LCD_LISTEN_OFF;
    return;
}

// Column: 0-127
void lcd_col(unsigned char column)
{
    // Byte H and L
    lcd_comm(0b00010000 | (column >> 4));
    lcd_comm(column & 0b00001111);
    return;
}

// Row (Actually, PAGE of 8 rows): 0-16
void lcd_row(unsigned char row)
{
    // Byte H and L
    lcd_comm(0b10110000 | row);
    return;
}

void lcd_init(void)
{
    // Set SPI control register appropriately (SPI interrupt enable, Master select)
    //SPCR = 0b01010011;
    SPCR = 0b01010000;

    LCD_RESET_SET;
    // LCD initialization commands
    lcd_comm(0xA0); // ACD select (set RAM address: 0 normal)
    lcd_comm(0xAE); // Display OFF
    lcd_comm(0xC0); // Common Output Mode (COM) select (normal)
    lcd_comm(0xA2); // LCD bias voltage set (1/9 bias)
    lcd_comm(0x2F); // Power control set (internal power supply operating mode)
    lcd_comm(0x26); // V5 voltage regulator resistor ratio set
    lcd_comm(0x81); // Electronic volume mode set
    lcd_comm(0x22); // V5 voltage regulator resistor ratio set
    screen_clear();  //Clear LCD RAM
    lcd_comm(0xAF);  // Display ON
}
```

screen.h

```
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// THAT Home Automation Topology * Electronic Access Module
// Microcontroller Firmware version 1.0 * 2010.04.08
// Copyright (C) Chris Miller * busybot.org
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


#ifndef SCREEN_H
#define SCREEN_H

    void screen_clear(void);
    void draw_char(unsigned char ch);
    void screen_char(unsigned char ch, unsigned char x, unsigned char y);
    void draw_string_pc(unsigned char *data, unsigned char row, unsigned char col);
    void draw_string_p(unsigned char *data);
    //void draw_icon(unsigned char ch, unsigned char row, unsigne char col);

#endif
```

screen.h

```
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// THAT Home Automation Topology * Electronic Access Module
// Microcontroller Firmware version 1.0 * 2010.04.08
// Copyright (C) Chris Miller * busybot.org
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#include "lcd.h"
#include "main.h"
#include "screen.h"
#include <avr/pgmspace.h>
#include "font.h"

extern volatile unsigned char name1[];

void screen_clear(void)
{
    unsigned char page = 0;
    unsigned char col = 0;

    for (page=0; page<8; page++)     // Write to page 0 then go to mext page.
    {                                 // 128pixels / 8 per page = 16 pages
        lcd_comm(page | 0b10110000);        // Set page address
        lcd_comm(0x10);         // Set column address MSB
        lcd_comm(0x00);         // Set column address LSB
        for(col=0; col<128; col++)    // each page has 128 pixel columns
        {
            lcd_data(0x00);
        }
    }
    return;
}

void draw_char(unsigned char ch)
{
    unsigned char x, b;
    const         prog_uint8_t* chp;
```

```
    if      (ch < FONT_5X7_ASCII_MIN)  ch = 32;
    //else if (ch > FONT_5X7_ASCII_MAX) ch = 136;

    chp = font_5x7_data + 5 * (ch-32);
    for(x=0; x<5; x++)
    {
        b = pgm_read_byte(chp + x);
        lcd_data(b);
    }

    return;
}

void draw_icon(unsigned char ch, unsigned char row, unsigned char col)
{
    unsigned char x, y, b;
    const         prog_uint8_t* chp;




    for(y=0; y<=1; y++)
    {
        chp = icons_2x2_data + 5 * (ch + 2*y);
        lcd_row(row+y);
        lcd_col(col*5);
        for(x=0; x<10; x++)
        {
            b = pgm_read_byte(chp + x);
            lcd_data(b);
        }
    }

    return;
}

void draw_char_c(unsigned char ch, unsigned char row, unsigned char col)
{
    lcd_row(row);
    lcd_col(col*5);
    draw_char(ch);

    return;
}

void draw_dual_p(unsigned char ch, unsigned char row, unsigned char col)
{
    lcd_row(row);
    lcd_col(col);
    draw_char(ch);
    draw_char(ch+1);
    return;
}

void draw_string(unsigned char *data)
{
    while (*data)
    {
        draw_char(*data);    // The character
        lcd_data(0);            // Trailing blank space
```

```
        data++;
    }
    return;
}

void draw_string_p(unsigned char *data)
{
    while (pgm_read_byte(data))
    {
        draw_char(pgm_read_byte(data));    // The character
        lcd_data(0);            // Trailing blank space
        data++;
    }
    return;
}

void draw_string_c(unsigned char *data, unsigned char row, unsigned char col)
{
    lcd_row(row);
    lcd_col(col*5+1);
    while (*data)
    {
        draw_char(*data);    // The character
        lcd_data(0);            // Trailing blank space
        data++;
    }
    return;
}

void draw_string_pc(unsigned char *data, unsigned char row, unsigned char col)
{
    lcd_row(row);
    lcd_col(col*5+1);
    while (pgm_read_byte(data))
    {
        draw_char(pgm_read_byte(data));    // The character
        lcd_data(0);            // Trailing blank space
        data++;
    }
    return;
}

void draw_byte_stretch(unsigned char y, unsigned char x, unsigned char l, unsigned char b)
{
    lcd_row(y);
    unsigned char i=x;
    for(i; i<=x+l; i++)
    {
        lcd_col(i);
        lcd_data(b);
    }
}
```

```
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// THAT Home Automation Topology * Electronic Access Module
// Microcontroller Firmware version 1.0 * 2010.04.08
// Copyright (C) Chris Miller * busybot.org
// %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#ifndef FONT58_H
#define FONT58_H

    #define FONT_5X7_ASCII_MIN      32
    #define FONT_5X7_ASCII_MAX      136
    #define ICONS_2X2_DATA_MIN      0
    #define ICONS_2X2_DATA_MAX      0

const prog_uint8_t font_5x7_data[] = {
    0x00, 0x00, 0x00, 0x00, 0x00, // SPACE
    0x00, 0x00, 0x5F, 0x00, 0x00, // !
    0x00, 0x03, 0x00, 0x03, 0x00, // "
    0x14, 0x3E, 0x14, 0x3E, 0x14, // #
    0x24, 0x2A, 0x7F, 0x2A, 0x12, // $
    0x43, 0x33, 0x08, 0x66, 0x61, // %
    0x36, 0x49, 0x55, 0x22, 0x50, // &
    0x00, 0x05, 0x03, 0x00, 0x00, // '
    0x00, 0x1C, 0x22, 0x41, 0x00, // (
    0x00, 0x41, 0x22, 0x1C, 0x00, // )
    0x14, 0x08, 0x3E, 0x08, 0x14, // *
    0x08, 0x08, 0x3E, 0x08, 0x08, // +
    0x00, 0x50, 0x30, 0x00, 0x00, // ,
    0x08, 0x08, 0x08, 0x08, 0x08, // -
    0x00, 0x60, 0x60, 0x00, 0x00, // .
    0x20, 0x10, 0x08, 0x04, 0x02, // /
    0x3E, 0x51, 0x49, 0x45, 0x3E, // 0
    0x00, 0x04, 0x02, 0x7F, 0x00, // 1
    0x42, 0x61, 0x51, 0x49, 0x46, // 2
    0x22, 0x41, 0x49, 0x49, 0x36, // 3
    0x18, 0x14, 0x12, 0x7F, 0x10, // 4
    0x27, 0x45, 0x45, 0x45, 0x39, // 5
    0x3E, 0x49, 0x49, 0x49, 0x32, // 6
    0x01, 0x01, 0x71, 0x09, 0x07, // 7
    0x36, 0x49, 0x49, 0x49, 0x36, // 8
    0x26, 0x49, 0x49, 0x49, 0x3E, // 9
    0x00, 0x36, 0x36, 0x00, 0x00, // :
    0x00, 0x56, 0x36, 0x00, 0x00, // ;
    0x08, 0x14, 0x22, 0x41, 0x00, // <
    0x14, 0x14, 0x14, 0x14, 0x14, // =
    0x00, 0x41, 0x22, 0x14, 0x08, // >
    0x02, 0x01, 0x51, 0x09, 0x06, // ?
    0x3E, 0x41, 0x59, 0x55, 0x5E, // @
    0x7E, 0x09, 0x09, 0x09, 0x7E, // A
    0x7F, 0x49, 0x49, 0x49, 0x36, // B
    0x3E, 0x41, 0x41, 0x41, 0x22, // C
    0x7F, 0x41, 0x41, 0x41, 0x3E, // D
    0x7F, 0x49, 0x49, 0x49, 0x41, // E
    0x7F, 0x09, 0x09, 0x09, 0x01, // F
    0x3E, 0x41, 0x49, 0x49, 0x3A, // G
    0x7F, 0x08, 0x08, 0x08, 0x7F, // H
    0x00, 0x41, 0x7F, 0x41, 0x00, // I
    0x30, 0x40, 0x41, 0x3F, 0x01, // J
    0x7F, 0x08, 0x14, 0x22, 0x41, // K
```

```
0x7F, 0x40, 0x40, 0x40, 0x40, // L
0x7F, 0x02, 0x0C, 0x02, 0x7F, // M
0x7F, 0x02, 0x04, 0x08, 0x7F, // N
0x3E, 0x41, 0x41, 0x41, 0x3E, // O
0x7F, 0x09, 0x09, 0x09, 0x06, // P
0x3E, 0x41, 0x51, 0x21, 0x5E, // Q
0x7F, 0x09, 0x19, 0x29, 0x46, // R
0x26, 0x49, 0x49, 0x49, 0x32, // S
0x01, 0x01, 0x7F, 0x01, 0x01, // T
0x3F, 0x40, 0x40, 0x40, 0x3F, // U
0x1F, 0x20, 0x40, 0x20, 0x1F, // V
0x3F, 0x40, 0x30, 0x40, 0x3F, // W
0x63, 0x14, 0x08, 0x14, 0x63, // X
0x07, 0x08, 0x70, 0x08, 0x07, // Y
0x61, 0x51, 0x49, 0x45, 0x43, // Z
0x00, 0x7F, 0x41, 0x00, 0x00, // [
0x02, 0x04, 0x08, 0x10, 0x20, // slash
0x00, 0x00, 0x41, 0x7F, 0x00, // ]
0x04, 0x02, 0x01, 0x02, 0x04, // ^
0x40, 0x40, 0x40, 0x40, 0x40, // _
0x00, 0x01, 0x02, 0x04, 0x00, // `
0x20, 0x54, 0x54, 0x54, 0x78, // a
0x7F, 0x44, 0x44, 0x44, 0x38, // b
0x38, 0x44, 0x44, 0x44, 0x44, // c
0x38, 0x44, 0x44, 0x44, 0x7F, // d
0x38, 0x54, 0x54, 0x54, 0x18, // e
0x04, 0x04, 0x7E, 0x05, 0x05, // f
0x08, 0x54, 0x54, 0x54, 0x3C, // g
0x7F, 0x08, 0x04, 0x04, 0x78, // h
0x00, 0x44, 0x7D, 0x40, 0x00, // i
0x20, 0x40, 0x44, 0x3D, 0x00, // j
0x7F, 0x10, 0x28, 0x44, 0x00, // k
0x00, 0x41, 0x7F, 0x40, 0x00, // l
0x7C, 0x04, 0x78, 0x04, 0x78, // m
0x7C, 0x08, 0x04, 0x04, 0x78, // n
0x38, 0x44, 0x44, 0x44, 0x38, // o
0x7C, 0x14, 0x14, 0x14, 0x08, // p
0x08, 0x14, 0x14, 0x14, 0x7C, // q
0x00, 0x7C, 0x08, 0x04, 0x04, // r
0x48, 0x54, 0x54, 0x54, 0x20, // s
0x04, 0x04, 0x3F, 0x44, 0x44, // t
0x3C, 0x40, 0x40, 0x20, 0x7C, // u
0x1C, 0x20, 0x40, 0x20, 0x1C, // v
0x3C, 0x40, 0x30, 0x40, 0x3C, // w
0x44, 0x28, 0x10, 0x28, 0x44, // x
0x0C, 0x50, 0x50, 0x50, 0x3C, // y
0x44, 0x64, 0x54, 0x4C, 0x44, // z
0x00, 0x08, 0x36, 0x41, 0x41, // {
0x00, 0x00, 0x7F, 0x00, 0x00, // |
0x41, 0x41, 0x36, 0x08, 0x00, // }
0x02, 0x01, 0x02, 0x04, 0x02, // ~
// Extended Characters
0x0C, 0x1E, 0x3C, 0x1E, 0x0C, // heart
0x30, 0x27, 0x20, 0x27, 0x30, // =]
0x30, 0x17, 0x10, 0x17, 0x30, // =[
0x10, 0x26, 0x20, 0x26, 0x10, // =)
0x00, 0x7F, 0x3E, 0x1C, 0x08, // arrow
0x18, 0x30, 0x18, 0x0C, 0x06, // check mark
0x7F, 0x7F, 0x7F, 0x7F, 0x7F, // block
0x10, 0x0A, 0x14, 0x28, 0x04, // spin 1
0x04, 0x34, 0x00, 0x16, 0x10, // spin 2
```

```
    0x08, 0x04, 0x3E, 0x04, 0x08, // up arrow
    0x08, 0x10, 0x3E, 0x10, 0x08, // down arrow
    0x08, 0x1C, 0x2A, 0x08, 0x08, // left arrow
    0x08, 0x08, 0x2A, 0x1C, 0x08, // right arrow
    0x60, 0xDC, 0x42, 0x61, 0xD1, // bell off left
    0xC9, 0x45, 0x42, 0x5D, 0x60, // bell off right
    0x60, 0x5C, 0x42, 0x41, 0xC1, // bell on left
    0xC1, 0x41, 0x42, 0x5C, 0x60, // bell on right
    0x00, 0x00, 0x00, 0x0F, 0x49, // net off left
    0x79, 0x49, 0x0F, 0x00, 0x00, // net off right
    0x00, 0xE0, 0xE0, 0xEF, 0x49, // network on left
    0x79, 0x49, 0xEF, 0xE0, 0xE0, // network on right
    0x00, 0x0F, 0x09, 0x19, 0x09, // DW offline left
    0x4F, 0x40, 0xE0, 0xA0, 0xE0, // DW offline right
    0x00, 0x0F, 0x09, 0x79, 0x49, // DW offline left
    0x4F, 0x40, 0xE0, 0xA0, 0xE0, // DW offline right

};

const prog_uint8_t icons_2x2_data[] = {
    // Lock closed
    0x00, 0xF0, 0x08, 0xE8, 0x28,   // Lock closed upper left
    0x28, 0xE8, 0x08, 0xF0, 0x00,   // Lock closed upper right
    0x7F, 0x41, 0x41, 0x41, 0x41,   // Lock closed lower left
    0x41, 0x41, 0x41, 0x41, 0x7F,   // Lock closed lower right
    // Lock opened
    0x00, 0x3C, 0x22, 0x3A, 0x0A,   // Lock opened upper left
    0x0A, 0xFA, 0x02, 0xFC, 0x00,   // Lock opened upper right
    0x7F, 0x41, 0x41, 0x41, 0x41,   // Lock opened lower left
    0x41, 0x41, 0x41, 0x41, 0x7F,   // Lock opened lower right
    // Door closed
    0xFE, 0x02, 0x3A, 0x2A, 0x2A,   // Door closed upper left
    0x3A, 0x02, 0xFE, 0x00, 0x00,   // Door closed upper right
    0x7F, 0x40, 0x42, 0x40, 0x40,   // Door closed lower left
    0x40, 0x40, 0x7F, 0x00, 0x00,   // Door closed lower right
    // Door opened
    0xFE, 0x02, 0x02, 0x02, 0x02,   // Door opened upper left
    0x02, 0xFF, 0x01, 0xFF, 0x00,   // Door opened upper right
    0x7F, 0x40, 0x40, 0x40, 0x40,   // Door opened lower left
    0x41, 0x7F, 0x40, 0x7F, 0x01,   // Door opened lower left

    0x00, 0x00, 0x00, 0x10, 0x30,
    0x60, 0x30, 0x18, 0x0C, 0x04,
    0x0E, 0x11, 0x15, 0x11, 0x0A,
    0x06, 0x0A, 0x06, 0x0A, 0x04
};
#endif
```

Appendix E – THAT Control Software Source Code

xml_parser.py

```
# -*- coding: utf-8 -*-
from xml.dom.minidom import parse, parseString
import sys, string

class xml_parser:

    def __init__(self, xml_file, verbose):
        # 0: no logging
        # 1: errors only
        # 2: errors + warnings only
        # 3: errors + warnings + info
        self.verbose = verbose

        self.mod_shortname    = ""
        self.mod_fullname     = ""
        self.mod_class        = ""
        self.mod_description  = ""
        self.mod_version      = ""
        self.mod_id           = ""

        self.manu_name     = ""
        self.manu_location = ""
        self.manu_date     = ""
        self.manu_phone    = ""
        self.manu_fax      = ""
        self.manu_website  = ""
        self.manu_email    = ""

        # A group of lists will hold the information corresponding to
        # every control available for read/write on a THAT module, and
        # the possible states, min/max value, current value, etc.
        self.control  = []  # Textual Name of a control
        self.register = []  # Integer or low-level name of a control
        self.states   = []  # Possible states (or "--")
        self.type     = []  # Read/write setting
        self.min      = []  # Minimum value for this control (or "--")
        self.max      = []  # Maximum value for this control (or "--")
        self.defaultv = []  # Default value
        self.defaults = []  # Default State
        self.current  = []  # The current value of this control

        try:
            xml_dom = parse(xml_file)
            self.print_info("XML file " + xml_file + " loaded")
            self.parse_module(xml_dom)
        except IOError:
            self.print_error("XML file \"" + xml_file + "\" could not be loaded!")
        except Exception, error:
            self.print_error("XML error in file: \"" + xml_file + "\" : " + str(error))

    def print_info(self, text):
        if (self.verbose >= 3):
            print "[ INFO ] " + text
    def print_error(self, text):
        if (self.verbose >= 1):
            print "[ ERROR] " + text
```

```
def print_warning(self, text):
    if (self.verbose >= 2):
        print "[ WARN ] " + text

def getText(self, nodelist):
    rc = ""
    for node in nodelist:
        if node.nodeType == node.TEXT_NODE:
            rc = rc + node.data
    return rc

def parse_module(self, module):
    # Check XML file for a <module> element
    mod = self.get_element(module, "module")
    if (mod == "---"): return

    # Check XML file for a name="" attribute
    self.mod_shortname = self.get_attribute(mod, "name")
    if (self.mod_shortname == "---"): return

    # Handle the <identification> element, if it exists
    stat = self.parse_identification(module)
    if (stat == "---"): return

    # Handle the <manufacturer> element, if it exists
    self.parse_manufacturer(module)

    # Handle the <controls> element, if it exists
    stat = self.parse_controls(module)
    if (stat == "---"): return

def parse_identification(self, module):
    # Check XML file for a <identification> element
    ident = self.get_element(module, "identification")
    if (ident == "---"): return("---")

    self.mod_id = self.get_attribute(ident, "id")
    if (self.mod_id == "---"): return("---")

    self.mod_fullname    = self.get_element_data(ident, "fullname")
    self.mod_description = self.get_element_data(ident, "description")
    self.mod_version     = self.get_element_data(ident, "version")
    self.mod_class       = self.get_element_data(ident, "class")

def parse_manufacturer(self, module):
    # Check XML file for a <manufacturer> element, if it exists
    manu = self.get_element(module, "manufacturer")
    if (manu == "---"): return

    self.manu_name     = self.get_element_data(manu, "name")
    self.manu_location = self.get_element_data(manu, "location")
    self.manu_date     = self.get_element_data(manu, "date")
    self.manu_phone    = self.get_element_data(manu, "phone")
    self.manu_fax      = self.get_element_data(manu, "fax")
    self.manu_website  = self.get_element_data(manu, "website")
    self.manu_email    = self.get_element_data(manu, "email")

def parse_controls(self, module):
    # Read <controls> section of XML file
    # Read <controls-item> sections from <controls> section
    controls     = self.get_element(module, "controls")
```

```
        if (controls == "---"): return
        controlitems = controls.getElementsByTagName("controls-item")
        # Parse each <controls-item> separately
        for item in controlitems:
            self.parse_control(item)

    def parse_control(self, control):
        print("")
        name     = control.getAttribute("name")
        self.control.append(name)
        self.print_info("Found new control  : " + name)

        register = self.get_element_data(control, "register")
        self.register.append(register)
        self.print_info("---> Register      : " + register)

        type     = self.get_element_data(control, "type")
        self.type.append(type)
        self.print_info("---> Type          : " + type)

        values   = self.get_element(control, "values")
        if (values != "---"):
            default = values.getAttribute("default")
            min      = self.get_element_data(control, "min")
            max      = self.get_element_data(control, "max")
            self.current.append(default)
        else:
            default = "---"
            min     = "---"
            max     = "---"

        self.defaultv.append(default)
        self.min.append(min)
        self.max.append(max)

        self.print_info("---> Default Value : " + default)
        self.print_info("---> Minimum       : " + min)
        self.print_info("---> Maximum       : " + max)

        statelist = []
        states   = self.get_element(control, "states")
        if (states != "---"):
            try:
                default = states.getAttribute("default")
                if (default == ""): raise Exception
            except:
                self.print_error("XML <control> element has no/invalid default attribute.
Parsing Stopped.")
                return

            statesitems = states.getElementsByTagName("states-item")
            for state in statesitems:
                try:
                    state_name  = state.getAttribute("name")
                    if (state_name == ""): raise Exception
                except:
                    self.print_warning("XML <states-item> element has no/invalid name.")

                state_value = self.get_element_data(state, "value")
```

```
            #self.print_info("---> State        : " + state_name)
            #self.print_info("------> Value     : " + state_value)

            statelist.append([state_name, state_value])
            # If the current state's name matches the default...
            if (state_name == default):
                self.defaults.append(state_value)
                self.current.append(state_value)

        self.print_info("---> Default state : " + default)
        self.states.append(statelist)

    # This control is NOT state-based, so return --- for all state values
    else:
        statelist = "---"
        self.states.append("---")
        self.defaults.append("---")

    self.print_info("---> States        : " + str(statelist))

    print("")

def get_attribute(self, element, attribute):
    try:
        data = element.getAttribute(attribute)
        if (data == ""): raise Exception
    except:
        self.print_error("XML <" + str(element.nodeName) + \
        "> element has invalid \"" + attribute + \
        "\" attribute. Parsing Stopped.")
        return("---")
    return(data)

def get_element_data(self, parent, tagname):
    try:
        object = parent.getElementsByTagName(tagname)[0]
        data   = self.getText(object.childNodes)
        return (data)
    except:
        #self.print_warning("Couldn't find tag: " + tagname)
        return ("---")

def get_element(self, parent, tagname):
    try:
        object = parent.getElementsByTagName(tagname)[0]
        return (object)
    except:
        self.print_error("XML <" + tagname + \
        "> element Not Found.")
        return ("---")
```

```
# -*- coding: utf-8 -*-
import string,time,sys
import os
import datetime
import threading
import that_module
from web_server  import *

VERBOSE    = 1
VERSION    = "0.6.3"
BUILD      = "2010/04/02"
OSNAME     = os.name
server     = ""

def print_info(text):
    print "[ INFO ] " + text
def print_error(text):
    print "[ ERROR] " + text
def print_warning(text):
    print "[ WARN ] " + text

def get_time():
    timenow = time.strftime("%H:%M:%S", time.localtime())
    return(timenow)

def get_date():
    datenow = time.strftime("%Y/%m/%d", time.localtime())
    return(datenow)

def modules_load(path):
    dir_list = os.listdir(path)
    for file_name in dir_list:
        print file_name

def main(arguments):
    global SERVER_HOST
    global SERVER_PORT
    global VERBOSE
    global server

    def print_splash():
        print "\r\nTHAT Home Automation Control Software"
        print "Version 0.2 - 2010/03/09"
        print "Copyright (c) 2010 Nick Viera"
        print "http://www.tehhouse.us/electronics/that/"

    def print_menu():
        print_splash()
        print "\r\nValid command-line options are:"
        print "-h,  --help     Print this menu and then exit"
        print "     --port=N   Start the web server on port N"
        print "     --verbose  Run with verbose output\r\n"

    # Handle each command-line argument
    for i in range(0, len(arguments)):
        # Handle a blank argument
        if (arguments[i] == "" or arguments[i] == " "):
            pass
```

```
        # Handle the port number argument
        elif (arguments[i].startswith("--port=")):
            part1, part2, part3 = arguments[i].partition("=")
            SERVER_PORT = int(part3)

        # Handle help or invalid argument
        elif (arguments[i] == "--help" or arguments[i] == "-h"):
            print_menu()
            sys.exit(0)

    copta = that_module.that_mod("Main Thermostat")
    that_module.mod_names.append("copta")
    that_module.mod_objs.append(copta)

    print_info("Starting HTTP server on port " + str(SERVER_PORT))

    try:
        server = HTTPServer((SERVER_HOST, SERVER_PORT), http_handler)
        print_info("HTTP server started")
        server.serve_forever()

    except KeyboardInterrupt:
        sys.stdout.write("\b\b")
        print_info("Received keyboard interrupt")
        print_info("Stopping HTTP server")
        server.socket.close()
        print_info("Exiting program")
        sys.exit(0)

    except:
        print_error("The HTTP server died unexpectedly")


if __name__ == '__main__':
    # Get any command-line arguments passed from the terminal
    arguments = []
    for arg in sys.argv:
        arguments.append(arg)
    main(arguments)
```

```
# -*- coding: utf-8 -*-
import socket
from   xml_parser import *
from   main       import *

mod_names = []
mod_objs  = []

################################################################################
class that_mod:

    def __init__(self, alias):
        self.socket   = ""
        self.alias    = alias

        self.mod_id           = ""
        self.mod_shortname    = ""
        self.mod_fullname     = ""
        self.mod_class        = ""
        self.mod_description  = ""
        self.mod_version      = ""
        self.addr             = 0
        self.port             = 0

        # A group of lists will hold the information corresponding to
        # every control available for read/write on a THAT module, and
        # the possible states, min/max value, current value, etc.
        self.control  = []  # Textual Name of a control
        self.register = []  # Integer or low-level name of a control
        self.states   = []  # Possible states (or "--")
        self.type     = []  # Read/write setting
        self.min      = []  # Minimum value for this control (or "--")
        self.max      = []  # Maximum value for this control (or "--")
        self.defaultv = []  # Default value
        self.defaults = []  # Default State
        self.current  = []  # The current value of this control

        print_info("Module " + self.alias + " loaded")
        xmlnow = xml_parser("modules/copta-module.xml", 3)
        self.load_xml_data(xmlnow)

    def load_xml_data(self, xml):
        self.mod_shortname    = xml.mod_shortname
        self.mod_fullname     = xml.mod_fullname
        self.mod_class        = xml.mod_class
        self.mod_description  = xml.mod_description
        self.mod_version      = xml.mod_version
        self.mod_id           = xml.mod_id

        self.manu_name        = xml.manu_name
        self.manu_location    = xml.manu_location
        self.manu_date        = xml.manu_date
        self.manu_phone       = xml.manu_phone
        self.manu_fax         = xml.manu_fax
        self.manu_website     = xml.manu_website
        self.manu_email       = xml.manu_email

        self.control  = xml.control
        self.register = xml.register
```

```
        self.states   = xml.states
        self.type     = xml.type
        self.min      = xml.min
        self.max      = xml.max
        self.defaultv = xml.defaultv
        self.defaults = xml.defaults
        self.current  = xml.current

        print_info(self.mod_shortname)
        print_info(self.mod_fullname)
        print_info(self.mod_class)
        print_info(self.mod_description)
        print_info(self.mod_version)

        print_info(self.manu_name)
        print_info(self.manu_location)
        print_info(self.manu_date)
        print_info(self.manu_phone)
        print_info(self.manu_fax)
        print_info(self.manu_website)
        print_info(self.manu_email)

        print_info(str(self.control))
        print_info(str(self.register))
        print_info(str(self.states))
        print_info(str(self.type))
        print_info(str(self.min))
        print_info(str(self.max))
        print_info(str(self.defaultv))
        print_info(str(self.defaults))
        print_info(str(self.current))

    def get_value_by_name(self, name):
        index = ""

        # Find which list element is THIS control
        for i in range (0, len(self.control)):
            val = self.control[i]
            if (val == name):
                index = i
                break

        if (index == ""): return("???")    # Error if control not found

        # Now, we should know if it is a VALUE or STATE control
        val = self.states[index]
        if (val != "---"):      # It is a STATE-based control
            val       = self.current[index] # Get current state element by number
            val, null = self.states[index][int(val)]    # Get name of that state
        else: val = self.current[index]
        return val

    def set_value_by_name(self, name, value):
        if    name == "???": return
        elif value == "???": return
        index = ""

        # Find which list element is THIS control
        for i in range (0, len(self.control)):
            val = self.control[i]
            if (val == name):
```

```
            index = i
            break

    if (index == ""): return # Error if control not found

    # Now, we should know if it is a VALUE or STATE control
    val = self.states[index]
    if (val != "---"):      # It is a STATE-based control
        pass
        # FIXME FIXME
        #val       = self.current[index] # Get current state element by number
        #val, null = self.states[index][int(val)]    # Get name of that state
    else: self.current[index] = value
    print_info("Saved " + value + " to control " + name)

    # Get the register number for this control
    regnum = self.register[index]
    self.connect("192.168.7.222", 8428)
    self.SET(regnum, value)

#-------------------------------------------------------------
# Opens a new connection to a THAT module
# addr = The IP address of the module
# port = The network port to connect through
def connect(self, addr, port):
    addr_from   = ''
    port_from   = 0
    self.addr   = addr
    self.port   = port

    # Create a new TCP socket object
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        # Setup the source port for the Kernel to use
        self.socket.bind((addr_from, port_from))
    except socket.error, msg:
        tmp = "Module " + self.alias + " socket error:" + str(msg)
        print_error(tmp)

    try:
        # Setup the destination port to connect to
        self.socket.connect((self.addr, self.port))
        tmp = "Module " + self.alias + " connected at " + addr
        print_info(tmp)
    except socket.error, msg:
        self.socket.close()
        tmp = "Module " + self.alias + " connection error:" + str(msg)
        print_error(tmp)

#-------------------------------------------------------------
# Disconnects from a THAT module
def disconnect(self):
    self.socket.close()
    print_info("Module " + self.alias + " Disconnected")

#-------------------------------------------------------------
# Sends a @IDD command to a THAT module to get its identification
def IDD(self):
    data = self.tcp_sendrecv("@IDD")
    n1   = 4
```

```
    n2   = data.find("#")

    if (data[0:n1] == "@DAT"):
        tmp = data[n1:n2]

        if (tmp == "$NAME"):
            data         = data[n2:len(data)]
            n1           = data.find("#")
            n2           = data.find("$")
            self.modname = data[n1+1:n2]
            print_info("Module Name: " + self.modname)

            data         = data[n2:len(data)]
            n1           = data.find("$")
            n2           = data.find("#")
            tmp          = data[n1:n2]

            if (tmp == "$ID"):
                data       = data[n2:len(data)]
                n1         = data.find("#")
                n2         = data.find("$")
                self.modid = data[n1+1:n2]
                print_info("Module ID: " + self.modid)

                data         = data[n2:len(data)]
                n1           = data.find("$")
                n2           = data.find("#")
                tmp          = data[n1:n2]

                if (tmp == "$MAN"):
                    data        = data[n2:len(data)]
                    n1          = data.find("#")
                    n2          = len(data)
                    self.modman = data[n1+1:n2]
                    print_info("Module Manufacturer: " + self.modman)

                else:
                    print_warning("Invalid Manufacturer data")
            else:
                print_warning("Invalid ID data")
        else:
            print_warning("Invalid NAME data")
    else:
        print_warning("Module provided unexpected response to IDD")

#-------------------------------------------------------------
# Sends a @SET command to a THAT module
# register = Register / Port / memory location to write to
# setdata  = The actual data or value to set
def SET(self, register, setdata):
    cmd  = "@SET$" + str(register) + "#" + str(setdata)
    data = self.tcp_sendrecv(cmd)
    n1   = 4
    n2   = data.find("#")

    if (data[0:n1] == "@ACK"):
        tmp = data[n1:n2]

        if (tmp == "$" + str(register)):
            tmp = tmp[1:len(tmp)]
            print_info("Register Name: " + tmp)
```

```
            data = data[n2:len(data)]

            if (data == "#" + str(setdata)):
                data = data[1:len(data)]
                print_info("Register set to: " + data)
            else:
                print_error("Invalid response, data was NOT set!")
        else:
            print_warning("Invalid register returned")
    else:
        print_warning("Module ACK not received in response to SET")

#-------------------------------------------------------------
# Sends a @GET command to a THAT module
# register = Register / Port / memory location to read from
def GET(self, register):
    cmd  = "@GET$" + str(register)
    data = self.tcp_sendrecv(cmd)
    n1   = 4
    n2   = data.find("#")

    if (data[0:n1] == "@DAT"):
        tmp = data[n1:n2]

        if (tmp == "$" + str(register)):
            tmp = tmp[1:len(tmp)]
            print_info("Register Name: " + tmp)

            data = data[n2:len(data)]

            if (data[0] == "#"):
                data = data[1:len(data)]
                print_info("Register data is: " + data)
            else:
                print_error("Invalid response, data was NOT received")
        else:
            print_warning("Invalid register returned")
    else:
        print_warning("Module DAT not received in response to GET")

#-------------------------------------------------------------
# Send and receive a set of information to/from a THAT module
# command = The complete command string to send
def tcp_sendrecv(self, command):
    tcp_buffer = 1024

    try:
        # Send a message
        self.socket.send(command)
        tmp = "Sent command:" + command
        print_info(tmp)
    except socket.error, msg:
        tmp = "TCP tx error:" + str(msg)
        print_error(tmp)
        return("xxx")

    try:
        # Receive a reply
        indata = self.socket.recv(tcp_buffer)
    except socket.error, msg:
```

```
            tmp = "TCP rx error:" + str(msg)
            print_error(tmp)
            return("xxx")

        return(indata)
#############################################################################
```

```
# -*- coding: utf-8 -*-
import string,cgi,time,sys
import os
import datetime
import threading

from socket import gethostname
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer

import that_module
import main

SERVER_PORT = 8080
SERVER_HOST = ''

PATH_WWW      = "www/"
PATH_BASE     = "builtin/"
PATH_MODULES  = "modules/"
HTML_WRAPPER  = "wrapper.html"
HTML_HEADER   = "header.html"
HTML_MENUMAIN = "menu_main.html"

def print_info(text):
    print "[ INFO ] " + text
def print_error(text):
    print "[ ERROR] " + text
def print_warning(text):
    print "[ WARN ] " + text

class page_generator:

    def __init__(self):
        self.html_output = []       # A new list of strings for the output
        self.load_wrapper()

    def load_wrapper(self):
        content = self.load_file(PATH_BASE + HTML_WRAPPER)
        if (content == -1): return

        head, foot = self.split_html(content, "[[Content]]")

        if (head == "" or foot == ""):
            head = "<html>"
            foot = "</html>"
            print_warning("HTML template file was parsed incorrectly")

        # Append the html header and footer code into the
        # list of strings to be output later...
        self.html_output.append(head)
        self.html_output.append(foot)

    def load_template(self, header, menu):
        if (header == 1):
            content = self.load_file(PATH_BASE + HTML_HEADER)
            if (content == -1): return
            self.html_output.append(content)

        if (menu == 1):
            content = self.load_file(PATH_BASE + HTML_MENUMAIN)
```

```
        if (content == -1): return
        self.html_output.append(content)

def load_copta(self):
    self.load_template(1,1)
    content = self.load_file("modules/copta-module.html")
    if (content == -1):
        self.html_output.append("ERROR reading index.html")
        return (-1)

    obj = that_module.mod_objs[0]

    content = self.replace_tag(content, "[[ALIAS]]", obj.alias)
    content = self.replace_tag(content, "[[FULLNAME]]", obj.mod_fullname)
    content = self.replace_tag(content, "[[ALIAS]]", obj.alias)

    content = self.parse_control_tags(obj, content)
    self.html_output.append(content)

def parse_control_tags(self, modobj, content):
    # Search for and handle all [[x]] tags in html file
    while(1):
        first, delim, last = content.partition("[[")
        if (delim == "" or last == ""): break

        tag, delim, last  = last.partition("]]")
        if (delim == "" or last == ""): break

        tagbr = "[[" + tag + "]]"
        content = self.replace_tag(content, tagbr, modobj.get_value_by_name(tag))
    return content

def load_index(self):
    self.load_template(1,1)
    content = self.load_file("index.html")
    if (content == -1):
        self.html_output.append("ERROR reading index.html")
        return (-1)

    content = self.replace_tag(content, "[[TIME]]", main.get_time())
    content = self.replace_tag(content, "[[DATE]]", main.get_date())
    self.html_output.append(content)

def load_status(self):
    self.load_template(1,1)
    content = self.load_file("status.html")
    if (content == -1):
        self.html_output.append("ERROR reading status.html")
        return (-1)
    content = self.replace_tag(content, "[[TIME]]",     main.get_time())
    content = self.replace_tag(content, "[[DATE]]",     main.get_date())
    content = self.replace_tag(content, "[[HOSTNAME]]", gethostname())
    content = self.replace_tag(content, "[[OSNAME]]",   main.OSNAME)
    content = self.replace_tag(content, "[[VERSION]]",  main.VERSION)
    content = self.replace_tag(content, "[[BUILDDATE]]",main.BUILD)
    self.html_output.append(content)


def load_file(self, filename):
    try:
        print_info("Opening HTML file: " + PATH_WWW + filename)
```

```
            fileobject = open(PATH_WWW + filename)
            htmlinput = fileobject.read()
            fileobject.close()
            print_info("HTML file read successfully")
            return (htmlinput)
        except:
            print_error("Unable to load HTML file")
            return (-1)

    def load_content(self, content):
        self.html_output.append(content)

    def split_html(self, content, delimiter):
        beginning = ""
        delim = ""
        end = ""

        # Split the string up into 3 parts
        beginning, delim, end = content.partition(delimiter)
        return beginning, end

    def replace_tag(self, content, delimiter, replacement):
        temp_content = content
        beginning = ""
        delim = ""
        end = ""

        while(True):
            # Split the string up into 3 parts
            beginning, delim, end = temp_content.partition(delimiter)

            # Delimiter tag was not found, return unaltered content
            if (delim == "" or end == ""): return (temp_content)
            temp_content = beginning + str(replacement) + end

        return (temp_content)

    def send(self):
        output = ""
        output = self.html_output[0]              # Send beginnning
        for i in range(2, len(self.html_output)):
            output = output + self.html_output[i] # Send content
        output = output + (self.html_output[1])   # Send ending
        return output


class http_handler (BaseHTTPRequestHandler):
    # HTTP GET request handler
    def do_GET(self):
        print_info("HTTP GET request for " + self.path)

        try:
            if (self.path == "/") or (self.path == "/index.html"):
                newpage = page_generator()
                result  = newpage.load_index()

                if (result == -1): raise IOError;

                self.send_response(200)
                self.send_header('Content-type','text/html')
                self.end_headers()
```

```
            self.wfile.write(newpage.send())

        elif self.path.endswith(".css"):
            path = self.path[1:len(self.path)]
            try:
                print_info("Opening CSS file: " + PATH_WWW + path)
                fileobject = open(PATH_WWW + path)
                cssinput = fileobject.read()
                fileobject.close()
            except:
                print_error("Reading of CSS file failed.")
                return

            self.send_response(200)
            self.send_header('Content-type','text/css')
            self.end_headers()
            self.wfile.write(cssinput)

        elif self.path.endswith(".esp"):   #our dynamic content
            self.send_response(200)
            self.send_header('Content-type','text/html')
            self.end_headers()
            self.wfile.write("hey, today is the" + str(time.localtime()[7]))
            self.wfile.write(" day in the year " + str(time.localtime()[0]))
            return

        else:
            newpage = page_generator()

            if (self.path == "/status.html"):
                result = newpage.load_status()
            elif (self.path == "/modules/copta-module.html"):
                result = newpage.load_copta()
            elif (self.path == "/modules/main-thermostat"):
                result = newpage.load_copta()
            else:
                result = newpage.load_file(self.path[1:len(self.path)])

            if (result == -1): raise IOError;

            try:
                if (self.send_headers == 1):
                    pass
            except:
                self.send_headers = 0

            if (self.send_headers == 1):
                    self.send_headers = 0
            else:
                self.send_response(200)
                self.send_header('Content-type','text/html')
                self.end_headers()
            self.wfile.write(newpage.send())

        return

    except IOError:
        self.send_error(404,'File Not Found: %s' % self.path)

# HTTP POST request handler
def do_POST(self):
```

```
    global rootnode

    print_info("HTTP POST received")

    try:
        ctype, pdict = cgi.parse_header(self.headers.getheader('content-type'))
        if ctype == 'multipart/form-data':
            query=cgi.parse_multipart(self.rfile, pdict)
        self.send_response(301)
        self.end_headers()

        # Get alias value from the dictionary, which returns the value
        # as a list element, so then get the first list element
        modulealias = query['alias']
        modulealias = modulealias[0]

        # Remove the alias dictionary element
        del query['alias']

        # Convert the remaining dictionary elements into an array
        # of strings representing the key and value pairs like:
        # query[0] = ('key',['value'])
        query    = dict.items(query)
        length   = len(query)

        # Handle each element one-by-one...
        for i in range(0, length):

            controlname = str(query[i])
            #sep1, sep2, controlname = controlname.partition(",")

            # Get just the control name now
            controlname, sep, controldata = controlname.partition(",")
            sep1  = controlname.find ("'")
            sep2  = controlname.rfind("'")
            controlname = controlname[sep1+1 : sep2]

            # Get just the control value now
            sep1 = controldata.find("['")
            sep2 = controldata.find("']")
            controldata = controldata[(sep1+2): sep2]

            print_info("Control: \"" + controlname + "\" to be set to: " + controldata)

            modobj = that_module.mod_objs[0]
            modobj.set_value_by_name(controlname, controldata)

        self.path = "/modules/copta-module.html"
        self.send_headers = 1
        self.do_GET()

    except :
        print_error("Error in POST data")

def send_error(self, number, text):
    self.send_response(number)
    self.send_header('Content-type','text/html')
    self.end_headers()

    if (number == 404): self.wfile.write("<h2>404 Error</h2>")
    self.wfile.write(text)
```