HARDWARE DATA ROUTER

By

John Alaimo

Hendrik Dewald

Satyam Shah

Final Report for ECE 445, Senior Design, Fall 2012

TA: Justine Fortier

12 December 2012

Project No. 9

Abstract

We designed and created a hardware data router for interfacing with multiple independent devices from a single computer using TCP/IP. The router allows for analog and digital signal sampling, digital output, and PWM. Although we were not able to successfully implement TCP/IP using the microcontroller, the current construction supports the physical interface itself and shows successful use of the peripherals, namely the digital and analog inputs and outputs.

A LabVIEW interface and instruction protocol for managing the data routers from a computer were developed and all functionality separate from the TCP/IP control and demonstrable was tested. Some improvements to our design are possible for implementation with more inputs and outputs and smaller device size.

Contents

1. Introduction

1.1. Purpose
The design of the hardware data router is to enable devices, which on their own do not possess a reliable computer interface, with a way of receiving and sending signals from a computer. To allow for robustness in the connection, as well as future expansion, an Ethernet interface was developed.

1.2. Functionality
In order to handle a large range of devices, the hardware data router was made capable of handling multiple forms of input and output on multiple channels. It was fitted with two PWM outputs, three digital outputs, two digital inputs, and two analog inputs. And for the purpose of providing power to any connected device, the router also controls two voltage supplies, the 5V logic source, and a higher external input, which is rated up to 28V at 15A with the current choice of relays.

1.3. Design Subdivisions
The design approach was generally divided into two major categories, the software and the hardware. Major hardware selection was completed early, allowing software to be developed while a hardware layout was specified. Additionally, the hardware was further associated into groups for the microcontroller, Ethernet functionality, and power handling. The software portion of the device lied in two categories, namely the user and device end. The user end involved the computer command interface responsible for providing instruction to the hardware router and receiving returned data, while the device end is composed of the programming of the microcontroller for peripheral usage and instruction and data handling.

2. Design

2.1. Design Alternatives

Multiple microcontroller architectures were considered for use here, though ultimately a PIC was selected, because of the assumption that course staff familiarity with PICs and an abundance of resources available online would allow for relatively easy program development. Other, already built packages were considered, like an Arduino or other Atmel chips, but the design avoided those options on the grounds of novelty. It was thought that, if the design is already using an Atmel chip, why not just use an Arduino, and if the design uses an Arduino, why not just use a USB interface instead of an Ethernet interface, since it was already included in the hardware.

USB was an interface consideration, though only briefly. Previous experience with hardware interface boards, like servo controllers, have shown limitations in the software provided by the manufacturer, as well as restrictions placed on the host device, since most interface boards are either RS-232, USB, or some proprietary interface, such as NI PCI cards. In order to use the hardware on a number of platforms without having to develop multiple software interfaces, or try to make a universal software solution, Ethernet was chosen to be the most viable option, specifically TCP/IP v4. Ethernet controllers are a common add-on to many projects, so support for development was thought to be relatively easy, as opposed to developing drivers for various operating systems.

2.2. Circuit Layout

The concept development of the hardware router began with identifying the interfaces it would have. Initially, the focus was some external device, with analog and digital signals, as well as a computer. However, there were no specifications in mind for the computer, so the use of Ethernet allowed for the rest of the design to be independent of the computer used. With the two data interfaces identified, the only other required device was the external logic power, as well as a consideration for handling of some external voltage source.

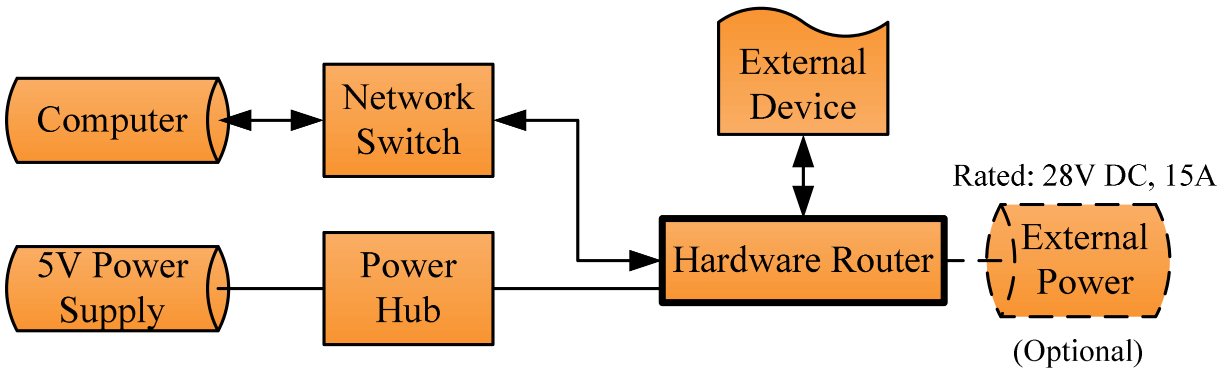

Figure 1. Network Level Block Diagram

Figure 1 shows the network level view of the hardware router, interfacing with other devices, such as the network switch, which is required for network operation, especially when multiple routers are used.

In order to facilitate testing and upgrades, through-hole package chips were used, instead of smaller surface mount packages, since the assembly would initially be much easier, and in the event of a part failure, the chips could be swapped out and easily replaced. One major challenge faced with the entire layout was routing traces on the PCBs, since the easy-to-use packages affected the trace layout on both sides of the board, instead of only a single side. The complications of layout were resolved, in part, by the use of multiple stacked boards, where each board represented an aspect of the router's functionality: main board, with the microcontroller and inputs/outputs, Ethernet board, and power board.

2.2.1. Microcontroller

Microcontroller selection was the driving factor behind most of the other design components. After research, a dsPIC33JF64MC802 was deemed to be the best chip for the application, though it ran on 3.3V instead of 5V, and as such, a regulator was also needed. The dsPIC33F family offered a fast speed of up to 80MHz with a sizable amount of program space (16kB), Direct Memory Access, which assured the microcontroller wouldn't be slowed by data transfer from peripherals to memory, and reprogrammable pins. In order to communicate with the Ethernet controller, SPI was chosen as the interface type, since it was common to both the microcontroller, and an alternative Ethernet controller and jack solution that had been considered. Due to cost reasons, however, the ready-made SPI-Ethernet solution was broken apart into a separate Ethernet controller and network jack.

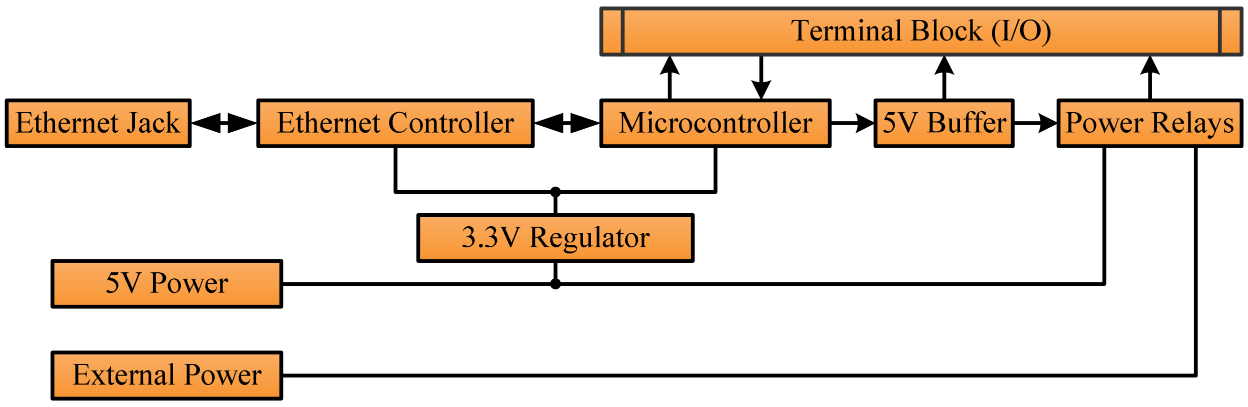

Figure 2. Hardware Router Internal Block Diagram

The selection of a logic-level translator, or voltage buffer, was based on the fact that the microcontroller outputs a 3.3V signal level, maximum, but the design was for the router to use 5V logic signals. As such, the buffer translated the voltage levels, and also provided extra current sourcing and sinking ability, but will be detailed further.

The microcontroller required a number of small discrete components to enable its functionality, as specified in the datasheet [1]. These parts are shown in Figure 7, and were mostly suggested by the manufacturer, except for the inductance and capacitance values for the analog input pins, which is calculated below.

According to the dsPIC33FJ datasheet [1], an inductor can be placed, from AV_{dd} to AV_{ss}, to eliminate noise and needs to be designed with a capacitor from the master clear pin (MCLEAR#) to ground. Generally, the frequency is half of the ADC sampling rate, such that

$$f = {}^{1}/_{2}\,(ADC\ rate) \tag{1}$$

The datasheet also specifies that the impedance of the inductor should be less than one Ohm

$$2\pi f L < 1\Omega \tag{2}$$

$$L < \frac{1\Omega}{2\pi f} \tag{3}$$

The selected microcontroller has a maximum sampling rate of 1.1 million samples per second,

$$L < \frac{1\Omega}{2\pi(1.1x10^{6}\ \text{Hz})} \tag{4}$$

$$L < 289\ \text{nH} \tag{5}$$

resulting in an inductance maximum value of 289 nH. To find the capacitance value, the datasheet provides

$$L = \left(\frac{1}{2\pi f\sqrt{C}}\right)^{2} \tag{6}$$

Rearranging and applying the above inequality, (9), yields

$$C > \left(\frac{1}{2\pi f\sqrt{L}}\right)^{2} \tag{7}$$

$$C > \left(\frac{1}{2\pi f\sqrt{289nH}}\right)^{2} \tag{8}$$

$$C > 72\,nF \tag{9}$$

That imposes a lower limit on the capacitance, and includes both types of capacitors used on the microcontroller, 10 µF and 100 nF. The lower capacitance is relatively close, almost within 25%, so the 10 µF capacitor value is selected, in order to keep parts standardized. Substituting back into (6) yields

$$L = \left(\frac{1}{2\pi f\sqrt{10\mu F}}\right)^{2} \tag{10}$$

$$L = 2.093\,nH \tag{11}$$

Given the result of (15), an inductor value of 2 nH was selected.

The Ethernet controller sinks, according to the datasheet [2], I_{DD} = 180 mA. The LED pins, clock out, data out, and interrupt pins can also sink or source up to 24 mA, resulting in a total power use of

$$P = I * V \tag{12}$$

$$P = (180 + 24)mA * 3.3V \tag{13}$$

$$P = 673\ mW \tag{14}$$

The microcontroller itself uses 300 mA max, for a maximum power consumption of

$$P = 300mA * 3.3V \tag{15}$$

$$P = 990\ mW \tag{16}$$

Thus, these two controllers consume approximately 1.66 W. Allowing for a safety factor of 25% for temperature effects, the total power being provided by the linear regulator should be approximately 2.08 W. The total power into the regulator is

$$P_{in} = \frac{V_{in}}{V_{out}} P_{out} \tag{17}$$

$$P_{in} = \frac{5V}{3.3V} 2.08W \tag{18}$$

$$P_{in} = 3.15W \tag{19}$$

which results in 630 mA on the regulator. The power hub should be able to supply at least that much to each router, but a budget of double, 1260 mA, should be allocated, to include all other internal logic and support circuitry. Additionally, a limit of 500 mA is imposed on the external logic, which is consistent with the USB specifications, creating a router maximum current of 1.76 A. Again, this is at full load, and happens briefly as the transistors change states. For two routers, the power hub should be able to support approximately 3.5 A. Considering the doubled current assumption, 3.5 A is an absolute maximum, and would not be continuous. Each device would use a maximum of 1.76 A, but the best solution for overcurrent protection is a fuse, and the nearest common fuse value is 2 A, which will be used on each line in the power hub.

2.2.2. Ethernet Interface

The design of the Ethernet controller layout, as well as Ethernet jack, was almost entirely suggested by the manufacturer in its datasheet [2]. The only modification was the choice to use a full-duplex instead of half-duplex indicator.

2.2.3. Power Switching

The power switching section, as it was referred to, handled the incoming power as well as control to the hardware of logic and external voltages. The choice of the regulator LD1086D2M33TR was dictated by the current calculations above [3]. As discussed mentioned, the buffer, TC74ACT244, was able to sink and source more current than the microcontroller could by itself, but also at a full 5V logic level [4]. For the design to accommodate voltage control to the external hardware, much consideration was given to using solid-state switches or mechanical relays, but it was ultimately decided that, for safest operation of an end user, to design with mechanical relays. The relays allow for full Galvanic isolation of the external hardware from the internal logic voltage, until the voltage is turned on by the microcontroller. Additionally, use of relays for the external voltage protects the switching logic from reverse polarity without more complicated isolation circuitry, such as optocouplers. The relays isolate the input negative voltage from the circuit ground, eliminating the possibility that a single bad connection with a piece of hardware will cause a ground fault and short out the internal circuitry.

2.3. Software Design

2.3.1. Microcontroller Programming

Programming the Microcontroller was a multiple stage process, often involving several different implementations to verify different peripheral functionality independent of Ethernet. But for all different programs we tested, proper initialization of microcontroller parameters was always the first step

Upon setting up a program, the first step towards correct implementation involved setting the configuration bits of our microcontroller. The configuration bits control the initial boot setup, and for our purposes they were chosen to set the oscillator to run from internal settings with a phase locked loop, set some clock out pins to digital input/output instead, and disable some unnecessary watchdog utilities. After the configuration bits were set, the oscillator was configured to run at a set frequency with the phase locked loop. Timers are initialized immediately after the oscillator, as well as any of the peripherals in use by the current code. For our demo, this included PWM, digital out, and digital in.

The microcontroller PWM module allowed, in our project, management of two PWM outputs, with independent duty cycles and a shared frequency. PWM functionality was accomplished through use of the microcontroller documentation [9] and some explanation on forums [5]. Initialization of the module consisted of setting related pins to outputs, activating independent running mode for the high pins that we used, setting the low pins to normal input/output mode, setting the prescaler, and enabling the module. Actual run-time implementation then just consisted of setting the frequency, which was dependent on the desired frequency, the prescaler, and the microcontroller operating frequency, and the duty cycle, which depended on the PWM frequency and the desired duty cycle. We could test our PWM functionality using an oscilloscope and LEDs, as per our verifications section.

The digital out functionality of our project was the simplest portion of functionality required from our microcontroller. Initialization required setting the related pins to outputs and, during run time, the output would be set simply by setting the matching pin latch to low or high. A timer was utilized to implement changing output signals separate from Ethernet instruction protocol commands. Testing was performed as per our verifications section, and the digital outputs were involved in debugging processes as well, providing us visual feedback of program status when connected to LEDs.

Digital in was closely tied to digital out in terms of implementation. Initialization simply required setting the related pins to outputs, and run time simply included setting the related pins’ port to low or high as needed. Digital in provided us a means to properly test our other modules more effectively in this case where we were not able to get TCP/IP functionality and thus had no complex external control. Digital in itself was also tested as per verification procedures as much as manageable without Ethernet functionality.

Analog in was programmed following the outline and formatting of example code provided on the Microchip website [8]. The sampling demo code consisted of initializing the ADC module and then sampling the incoming data according to a timer interrupt. The data was then pushed to DMA and recorded under SRAM. However, our original requirements and verifications did not outline many methods for testing analog in functionality in a case without working Ethernet.

Microchip, the company behind our microcontroller [1] and Ethernet controller [2], provides an extensive library for use with their products along with their documentation. In said library are the means for implementing a TCP/IP stack for a variety of controller and board combinations. By referencing their provided documentation and demo code, as well as a guide [5]and demo code from an open source project we found online [6], we constructed a generic TCP/IP stack handling microcontroller program. We included all the necessary libraries and header files for our project requirements, as indicated by the provided TCPIP Stack Help manual [7], and altered the two configuration sections of the code to activate the relevant portions of the included source files. We then implemented the stack management in the main loop of the program and included some debug lines intended to send high outputs to the digital outputs and voltage switches when certain sections of the code were reached, with the hope of being able to see at which point in the main program the microcontroller is at any time.

2.3.2. Data Protocol

The data protocol for this project is defined in such a way that it is not application limited. It can be used to transmit data through any program that supports communication over Ethernet Protocol. We will use string data type to represent our message[1]. The basic idea behind the form is to concatenate every portion of the message together along with the string length of each part, which will separate one portion of the message from another, as shown in Table 1. The concatenated string is ended with four termination characters, each of which conveys a special message to the receiver. This is the structure of Payload that should be followed every time the message is sent over Ethernet.

Table 1: General Form of Data Payload Portion of Every Ethernet Frame or TCP/IP Packet

4 bytes	(bytes equal to number represented by String Length 1) **n bytes**	**4 bytes**	(bytes equal to number represented by String Length 2) **m bytes**	…….	**4 bytes**	(bytes equal to number represented by String Length k) **n bytes**	**4 bytes**
String Length 1	Relevant Data 1	String Length 2	Relevant Data 2		String Length k	Relevant Data k	Termination? (n/h,A-Z,0-9,&)

Comments on the General Format of the Data Protocol

- Each string length portion represents the length of the follow-up piece of information in bytes. For example, in Table 1, if the Relevant Data 1 is 2 bytes long, then the first string length would be _ _ _ 2. This way of including the string length makes it easier for the receiver, a microcontroller in this case, to decode the message, and extract the information.
- m, n represent an arbitrary number of bytes.

[1] The data does not have to be represented in string format, but could be any arbitrary data type supported by the application used to communicate over Ethernet. The user can and should select the data type that they are most comfortable to work with in defining their message. For our purposes, we have chosen to use string format due to LabVIEW's limitation in only transmitting strings over Ethernet, in their ASCII binary representation.

- The four termination characters act as an indicator for the receiver. They will inform the receiver that the sent message with all the useful information has been read and it can stop reading any further. The 4 characters mean the following each:

  First character: n/h

  - If this character is an h, it informs the receiver that the received message is a portion of more data to come.
  - If this character is an n, it means that this data is either independent of all the received data and/or it is the last sequence of the message sent in multiple packets. The second and third characters will inform exactly which of these two interpretations apply in a given situation.

  Second/Third characters: A-Z/0-9 comfort

  - If the message is sent in multiple packets, then the receiver needs to know which message sent previously does the current message correspond to. That's what these 2 characters represent. This is useful to have especially when the computer receives the sampled data of an analog signal or a digital signal data from the microcontroller in multiple sequences.
  - The 2nd character is a capital alphabetical letter A-Z and the 3rd character is a number 0-9. The letter acts as a unique identifier and the number is a sequence number of the message. Concatenating them together allows the receiver to know the message this data belongs to. For example, A1 would mean that this is a part of the message A with sequence number 1.

  Fourth character: &

  - This informs the receiver needs not to read anymore.

3. Design Verification

3.1. Ethernet Integrated Circuit (EIC)

EIC.1: Receive data packets from the computer successfully.

EIC.2: Communicates with the microcontroller using an SPI interface.

3.2. Microcontroller (MC)

MC.1: Sends, receives, and interprets information from the Ethernet IC, through the use of a set instruction protocol and the SPI interface.

a) Will only send data to the computer, as the microcontroller will save the MAC address of the first command it receives which will be sent from the computer after start up
b) Will only accept commands from the computer, again, identified by the MAC address set at start up

MC.2: Can perform multiple tasks in response to computer instruction and according to supplied parameters.

a) Samples analog inputs
b) Samples digital inputs
c) Produces a PWM output
d) Produces a digital output
e) Switch on or off both the 5V logic and the higher voltage external power supply to the device
f) Can be reset to start conditions
g) Can send a status update on all pin's current settings and functionality

MC.3: Logic output voltages shall be 5V signals

a) $V_{OH} > 75\%$ Vcc
b) $V_{OL} < 25\%$ Vcc

3.3. Power Switching (PS)

PS.1: The relays and fuses of the 5V logic and the higher voltage external power supply ensure the safety of the device use.

3.4. Power Hub (PH)

PH.1: Nominally supplies 5V to each of the hardware routers, with a minimum of 4.891V for proper operation of the linear regulator.

3.5. Computer (CO)

CO.1: LabVIEW interface provides a means to access and utilize the microcontroller output, input, power routing, and status functions.

CO.2: LabVIEW presents acquired data in table, graph, or scope form.

3.6. Testing

3.6.1. Microcontroller

Concerning our Ethernet program, none of the outputs were returning high when the code was loaded onto the microcontroller, showing that we had not reached even the earliest stages of our program successfully. The microcontroller always stayed at a constant state where all pins, save the active low reset and chip select pins for controlling the Ethernet controller, were low. Attempted simulations using the MPLAB X IDE microcontroller programming software provided no further insight, instead showing no compilation errors or crashes throughout implementation of our code. Unfortunately, we had to continue working primarily on direct peripheral controls as opposed to Ethernet functionality. And as we were unable to achieve a functioning TCP/IP interface with the microcontroller, our testing remained primarily in verifying peripheral outputs to match set values loaded in the program. Following our requirement and verification procedures, we observed functioning PWM and digital input/output both with the oscilloscope and LED circuits.

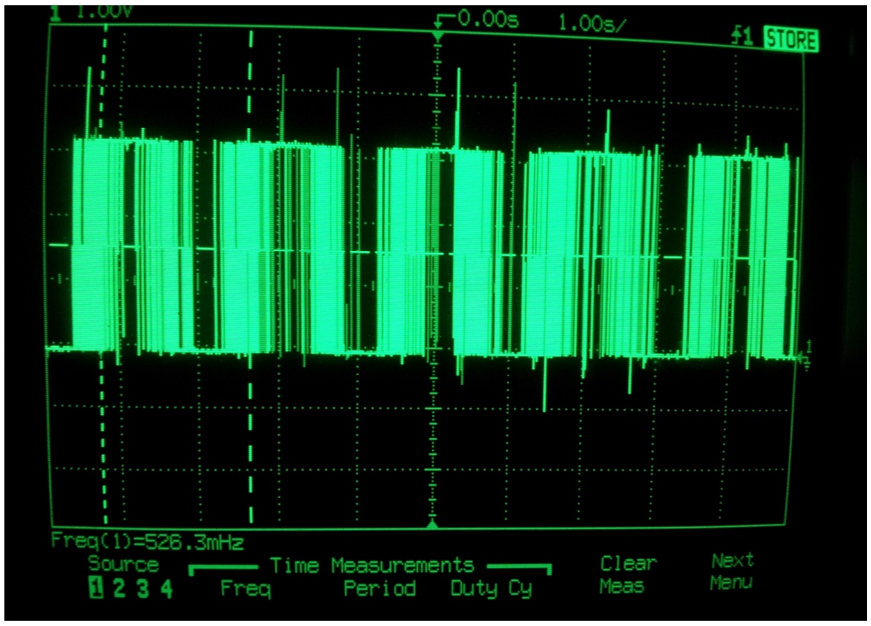

Figure 3: Sinusoidal PWM Output

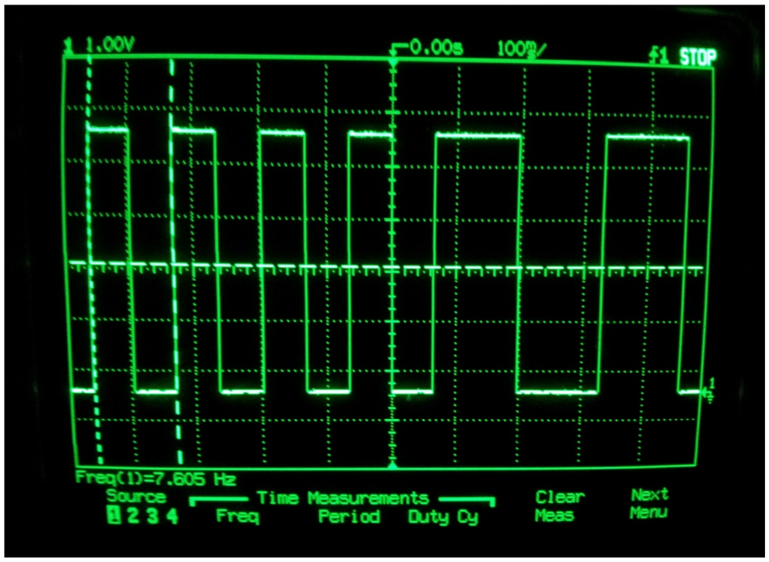

Figure 4: Switching Digital Output with Frequency Determined by Digital Input

3.6.2. LabVIEW

To test the functionality of the project, we wrote a LabVIEW program that sends and receives data over Ethernet based on the predefined instructions in Table 1. To generate the instructions, the program uses the following format defined in Table 1. The original plan was to use the MAC address of each Ethernet controller to communicate with the devices. But due to limitation of LabVIEW in communicating over Ethernet using just MAC address, it required setting up a DHCP server to assign an IP address to each device.

Table 2: Instruction Assignments to Unique Identifiers

Code	Category
A0	Microcontroller Status
A1	Analog Read
A2	Analog Write
A3	Digital Read
A4	Digital Write

After running the program, the general outline of this program is as follows: First, the user enters the IP address and a port number, which is an arbitrary 4-digit number, of the particular Ethernet controller with which he/she wishes to communicate. Then, the user selects from the options the instruction and relevant data. LabVIEW uses these inputs to generate the instruction data in string format. The generated data can be seen in the top-left of the front panel in “Sent Data” box. Next, when the user hits the SEND button, the program opens a connection with the Ethernet controller, converts the data to ASCII binary format and sends the data as a TCP/IP packet. As long as the program is still running, the connection is open. If the Ethernet controller sends instruction/data to the computer, the LabVIEW program receives it. The program extracts the information and then performs tasks relevant to the nature of the data. For example, if the received information represents the sampled signal of an analog input, then the program regenerates the signal using the samples and plots the wave in the front panel. When the program is stopped, the connection with the Ethernet controller is closed. To communicate with a different device while the program is still running, the user just needs to change the corresponding IP address and he/she starts communicating with a different Ethernet controller.

With this program, the user can command the microcontroller do multiple tasks. The program can be used to output a PWM waveform with particular properties, such as frequency, amplitude, duty cycle, etc or it can be used to output a digital data with specified sting of bits. It can also reset the microcontroller pins to its initial state and/or command the microcontroller to power the device with either Vlogic or Vexternal. The program can also command the microcontroller to forward the computer the sampled data of an analog or digital input and it can use the received samples to regenerate the original waveform.

4. Costs

4.1. Parts

Table 3: Budget

Part	Part No	Supplier	Price	Quantity	SubTotal
Microcontroller	dsPIC33FJ64GP802-DIP	Digikey	$ 6.24	2	$ 12.48
Microcontroller	dsPIC33FJ64MC802-DIP	Digikey	$ 6.38	2	$ 12.76
Ethernet controller	ENC28J60	Digikey	$ 3.86	2	$ 7.72
2 nH inductor (smd)	MLG1005S2N0S	Digikey	$ 0.06	2	$ 0.12
Ferrite bead	MH2029-070Y	Digikey	$ 0.04	2	$ 0.08
25 MHz xtal osc.	ABLS-25.000MHZ-B4-F-T	Digikey	$ 0.41	2	$ 0.82
Linear regulator	LD1086D2M33TR	Digikey	$ 1.02	2	$ 2.04
100nF Capacitor	C1608X7R1E104K	Digikey	$ 0.10	8	$ 0.80
10 µF Capacitor	EMK212BJ106KG-T	Digikey	$ 0.25	7	$ 1.75
18 pF Capacitor	C0603C0G1E180G	Digikey	$ 0.10	4	$ 0.40
Fuse (2A)	5MF 2-R	Digikey	$ 0.19	4	$ 0.76
Fuse holder	64900001039	Digikey	$ 0.39	2	$ 0.78
Switch (SPDT slide)	GF-624-6014	Digikey	$ 0.77	1	$ 0.77
PCB		Parts shop	$10.00	3	$ 30.00
7 Pin Terminal Block	20020316-H071B01LF	Digikey	$ 1.73	4	$ 6.92
470Ω Resistor	RMCF0402JT470R	Digikey	$ 0.02	2	$ 0.04
10KΩ Resistor	CR1206-FX-1002ELF	Digikey	$ 0.02	2	$ 0.04
49.9Ω Resistor	RMCF1206FT49R9	Digikey	$ 0.04	10	$ 0.40
49.9KΩ Resistor	RMCF0603FT49K9	Digikey	$ 0.04	8	$ 0.32
2.32KΩ Resistor	RNCP1206FTD2K32	Digikey	$ 0.09	7	$ 0.63
Banana Jack (red)	108-0902-001	Digikey	$ 0.70	2	$ 1.40
Banana Jack (black)	108-0903-001	Digikey	$ 0.70	2	$ 1.40
28 pin DIP socket	4828-3004-CP	Digikey	$ 0.42	4	$ 1.68
20 pin DIP socket	A20-LC-TT-R	Digikey	$ 0.29	2	$ 0.58
Power Relay (SPST)	ORWH-SH-105HM3F,000	Digikey	$ 1.63	4	$ 6.52
Signal Relay (DPST)	ALA2F05	Digikey	$ 1.96	2	$ 3.92
Power Jack	PJ-037A	Digikey	$ 0.91	5	$ 4.55
Power Plug	PP3-002A	Digikey	$ 1.36	6	$ 8.16
Dual octal buffer	TC74ACT244P(F)	Digikey	$ 0.73	4	$ 2.92
Ethernet breakout	BOB-00716	Sparkfun	$ 1.95	1	$ 1.95
Ethernet Jack	J00-0045NL	Digikey	$ 4.73	2	$ 9.46
Shipping + Handling			*20%*		*$ 39.65*
Total					**$ 171.82**

4.2. Labor

Table 4: Labor Cost

Worker	Hourly Rate	Hours/Week	Weeks	Salary
John Alaimo	$ 60.00	12	12	$ 8,640
Hendrik Dewald	$ 40.00	18	12	$ 8,640
Satyam Shah	$ 30.00	24	12	$ 8,640
Subtotal				*$ 25,920*
Total	**2.5x overhead**			**$ 64,800**

Estimated project grand total: $64,971.82

5. Conclusion

5.1. Accomplishments

In terms of the functionality of the project, we successfully had the hardware router to function properly, which was very essential for the overall functionality of the project. Also, we were able to get to compile and load all the programs in the microcontroller. The microcontroller was able to use control the devices such as an LED using PWM mode to output analog and digital signal of various types, such as sine wave, square wave, etc. We were able to control the duty cycle and frequency of the PWM outputs with frequencies as large as 41.49 KHz.

5.2. Uncertainties

Though we created a program for sampling analog inputs, the sampled data is stored by DMA and, without Ethernet, is not very accessible. MPLAB X IDE allows for reading memory from a Microcontroller, but read data primarily consists of the program memory. Thus we have not yet directly viewed sampled Analog data, and are uncertain as to the current state of analog sampling functionality. Also, we have not had a chance to fully test the functionality of the LabVIEW program due to lack of functionality of the Ethernet. We are still in the process of verifying that the program works by showing the output in a network analyzer.

5.3. Ethical considerations

We adhered to all of the IEEE Code of Ethics, especially those that most pertained to the project, such as:

1. To accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment

To the best of our knowledge, we have taken into account all the safety measures possible to make sure the end product is safe to use; especially, any safety concerns related to the control of the external power, where our specified device ratings will have at least a 25% safety factor included with respect to the manufacturer's listed ratings.

3. To be honest and realistic in stating claims or estimates based on available data

We have not fabricated any of our results or attempted to hide any shortcomings of our final product. All results and subsequent conclusions are supported by collected data and simulations.

7. To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others

We have tried to make a cost-effective and the best possible product design using all the available resources. We are open to any constructive criticisms made that would help us in improving the product design, performance, and its applicability in the future. Since the product was made using parts and programs already available in the market, we tried our best to give full credit to their manufacturers in any means possible.

5.4. Future work

Future work is ongoing, in both design and testing stages. Current effort focuses on additional functionality testing, in order to validate proper payload formation in LabVIEW in order to send data to the Ethernet controller, as well as additional testing and debugging of the software loaded onto the microcontroller.

Immediately, redesign work will be happening with a testing goal for the end of January, since the hardware data router has been promised for certain applications by that time. Improvements that will be implemented will make the board easier to assemble and interface with other systems. The first update will be increasing the line voltage level to 12 volts, for use with standard 12 volt battery systems. A higher voltage will also allow for an amplifier to boost the 3.3 volt analog output to a 5 volt signal, since that had not been possible with only a 5 volt source because any amplifier circuit, such as an op-amp, would be limited and unable to get a full 5 volt swing. In terms of actual production quality improvements, surface mount devices will be used more, to include the buffer, microcontroller, and Ethernet controller. Both the microcontroller and Ethernet controller feature multiple surface mount package options, and the only redesign will be in which pins are used, as well as the selection of the voltage buffer. On the note of surface mount parts, the resistors and capacitors selected were all of the correct values, but however the part sizes selected during the original design were of varying sizes, including some, like package M0201, were extremely difficult to use. The redesign will standardize the package size, likely using a package size M1206, since that is a relatively large package that still has readable text. Standardizing the package size will allow for easier milling, as well as designing in drafting software.

Another additional update would likely be a more effective device name, where currently the term "network hardware platform" seems to be a better fit, though it was felt to maintain the current name so all documentation of the project is clear for the first hardware revision. Network hardware platform would not use the ambiguous term "router", which is already a commonly known piece of network hardware. The idea of a platform suggests that future versions of the product will be able to full support devices on a network, instead of just routing data for it; long term usage goals would be to have the network hardware platform running a distributed network of devices, for example lighting arrays in a building, which could be able to run for extended amounts of time without user interaction, but could sample data and send it back to a central building server.

Another future change that can be implemented involves, presumably integrated for a surface mount microcontroller upgrade, more PWM functionality. Currently, our PWM outputs are both from the same module and thus share the same frequency. This is helpful in situations like running a three-phase motor. But in future implementations, we can make use of a second PWM module present to provide two sets of PWM outputs with independent frequencies. And with a surface mount version of our microcontroller and it's increased number of pins, this means we could perhaps operate both a three-phase motor using three PWM signals from one module and another motor or servo using a PWM signal from the other module.

References

[1] Microchip Technology, "dsPIC33FJ64MC802," Microchip Technology, [Online]. Available: http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en532314#documentation. [Accessed September-December 2012].

[2] Microchip Technology, "Stand-Alone Ethernet Controller with SPI Interface," July 2006. [Online]. [Accessed October 2012].

[3] Toshiba Corporation, "TC74ACT244P/F/FW/FS," April 1996. [Online].

[4] STMicroelectronics, "1.5 A Adjustable and fixed low drop out psitive voltage regulator," May 2006. [Online].

[5] Dangerous Prototypes, "Introduction to dsPIC33 programming," Dangerous Prototypes, [Online]. Available: http://dangerousprototypes.com/docs/Introduction_to_dsPIC33_programming. [Accessed September-December 2012].

[6] Dangerous Prototypes, "Web platform hardware design," Dangerous Prototypes, [Online]. Available: http://dangerousprototypes.com/docs/Web_platform_hardware_design. [Accessed December 2012].

[7] Microchip Technology, "Microchip TCP/IP Stack," [Online]. Available: http://www.egr.msu.edu/classes/ece480/capstone/fall11/group03/TCPIP%20Stack%20Help.pdf. [Accessed September-December 2012].

[8] ilco, "Microchip," Microchip Technology, [Online]. Available: http://www.microchip.com/forums/m511200.aspx. [Accessed November 2012].

[9] National Instruments, "What is VISA?," [Online]. Available: http://www.ni.com/visa.

[10] National Instruments, "Basic TCP/IP Communication in LabVIEW," National Instruments, [Online]. Available: http://www.ni.com/white-paper/2710/en. [Accessed October 2012].

[11] Microchip Technology, "16-bit Digital Signal Controllers (up to 128 KB Flash and 16K," [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/70291G.pdf. [Accessed December 2012].

[12] Microchip Technology, "Section 14. Motor Control PWM," [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/70187E.pdf. [Accessed December 2012].

[13] Microchip Technology, "Section 16. Analog-to-Digital Converter (ADC)," [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/70183D.pdf. [Accessed December 2012].