NETWORKING WITH EMBEDDED SYSTEMS

Antonio Alexander, Farrokh Attarzadeh
Engineering Technology Department
University of Houston
alalexander2@uh.edu, FAttarzadeh@uh.edu

Abstract

The fundamental building blocks for an embedded system are central processing, interfacing, and communication; these blocks make up the embedded system triangle. One of the most complex components is communication and as such Internet Protocol version 4 (IPv4) [1] was chosen for this research because it is the standard on which the Internet functions. Furthermore, the relative ease with which any embedded system can readily be connected to the network, and obtain additional functionality with remote querying and access will be described and explained in later sections. An introduction to socket programming and the creation of client/server programs will be shown whose functionality can be easily enhanced and further developed. For demonstration purposes, a simple embedded system was created to exhibit the depth of functionality available through the research and later expanded upon.

1. Introduction

In Embedded Systems, there are always obstacles to overcome in areas dealing with communications. Developing one's own communications protocol is costly and may require external expertise. The following sections will describe how to operate a communication stack used on Ethernet-based IPv4 networks. The technique will also demonstrate simple interfacing that can easily be applied to a more complex system such as process automation or computer aided manufacturing. Only small portions of this research were synthesized and that most of the research entailed learning different systems and making them work together. Three major objectives had to be overcome in order to achieve the objective of the abstract: (1) how to connect Embedded systems to IPv4 based networks, (2) use of socket programming with text based communication, and (3) configuring and using communication's stack.

2. How to Connect Embedded Systems to IPv4 Networks

To overcome the first obstacle, the Atmel Atmega168 [2, 3] connected to a Microchip ENC28J60 [4] via SPI (Serial Peripheral Interconnect) was used. Software-wise, Adam Dunkel's Micro Internet Protocol (uIP) TCP/IP [5, 6] stack was used. The second problem was solved using the Perl programming language and the sockets library. Prior to the present prototype, shown in Fig. 1, the Tuxgraphics [7] communication stack was used. In the end, its use was decided against because it was not a fully functional communications stack and was replaced with the uIP stack rewritten for the AVR [8].

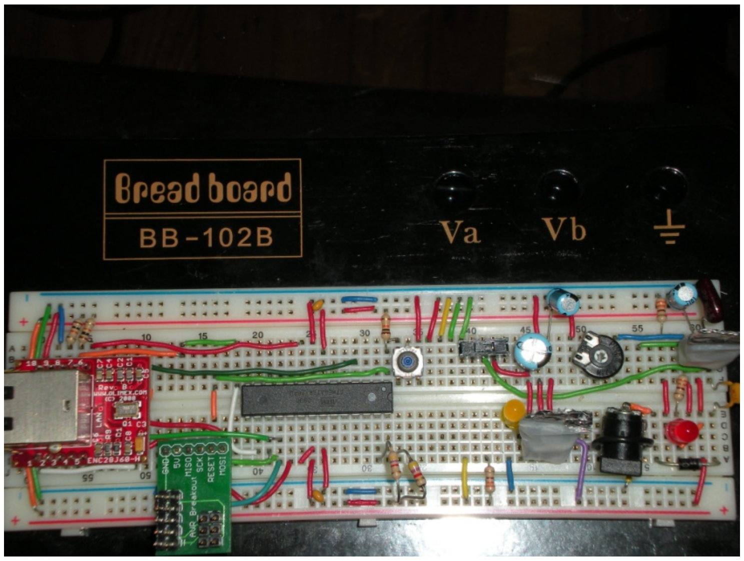

Figure 1. System Prototype

The implementation of the aforementioned objectives is based on the premise that most network communication is text based. Text based communication is very simple because it is human readable, as well as the fact that it is not binary therefore much easier to implement. The Atmega168 has a wide community support, a Parallel Dual Inline Package (PDIP) applicable for bread boarding and specialized internal hardware for communication using Serial Peripheral Interface (SPI), Inter-integrated Circuit (I^2C) or Two-Wire Interface (TWI) and Universal Asynchronous Receive/Transmit (UART). The ENC28J60 Ethernet controller is useful because it is available on an Olimex breakout board specifically made for breadboards. Perl was chosen because of its powerful ability to handle text as well as its use of libraries for special functions such as socket programming.

3. Communication

The communications stack consisted of two parts: a main driver program and an application. The driver program was responsible for initializing everything from the Ethernet controller, to the addresses used in the stack for communication over the network. Once the controller was initialized and access to the network was established, applications were used to add extra functionality to the communications stack such as a web server, or an interfacing script that used I^2C to communicate with other devices. For example, if you had an air conditioning system that was controlled using the network, a temperature sensor network could be monitored via an application.

The other side of the communication was accomplished with a client. A client was an end-user program that allowed communication with a server. For example, when accessing a webpage, a web browser is used to communicate with the web server. Socket programming was used to create a client to match the specifications of the embedded system. A socket is defined as connection to a system that consists of an IP address and port number. The general sequence of events when the client communicated with the communication stack was as follows: (1) the device listens on a specific port and IP address, (2) the client connects to the device on the specific port and IP address, and (3) the device runs its application and performs a certain function. This process is illustrated in Fig. 2. There are two primary ways of writing simple echo based clients that receive and transmit text over the network: using a forking client or a non-forking client. A non-forking client uses a static sequence of events for communication while a forking client separates the sending and receiving processes so the client can easily communicate with both passive and active servers.

4. Application: Network Enabled Displays

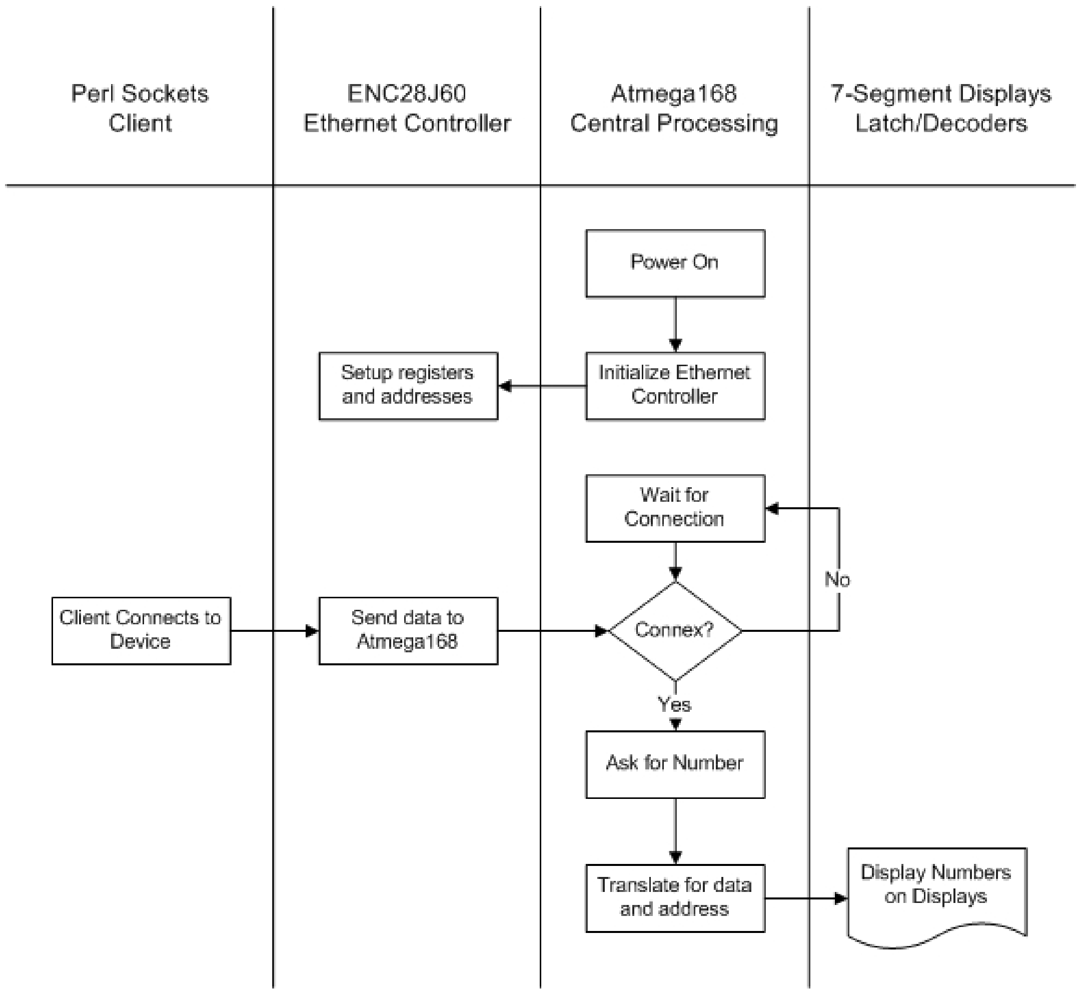
Perl Sockets Client
ENC28J60 Ethernet Controller
Atmega168 Central Processing
7-Segment Displays Latch/Decoders
Power On
Setup registers and addresses
Initialize Ethernet Controller
Wait for Connection
No
Client Connects to Device
Send data to Atmega168
Connex?
Yes
Ask for Number
Translate for data and address
Display Numbers on Displays

Figure 2. Flowchart

An example embedded system was devised to show the level of functionality attainable through the use of interfacing, socket programming, and embedded systems programming. The example interfaces four seven-segment displays using a seven-segment to BCD encoder with latch [9] a 3 to 8 multiplexer [10] and an I^{2}C based 8-bit I/O Expander [11]. Figure 3 shows a simple block diagram of each of the important system components. There are two important busses used in the system to transmit data: a four bit data bus, and a three bit address bus. The data bus is used to provide the input for the BCD to seven-segment encoders, and the address bus is used to control multiplexer, which activates the latches.

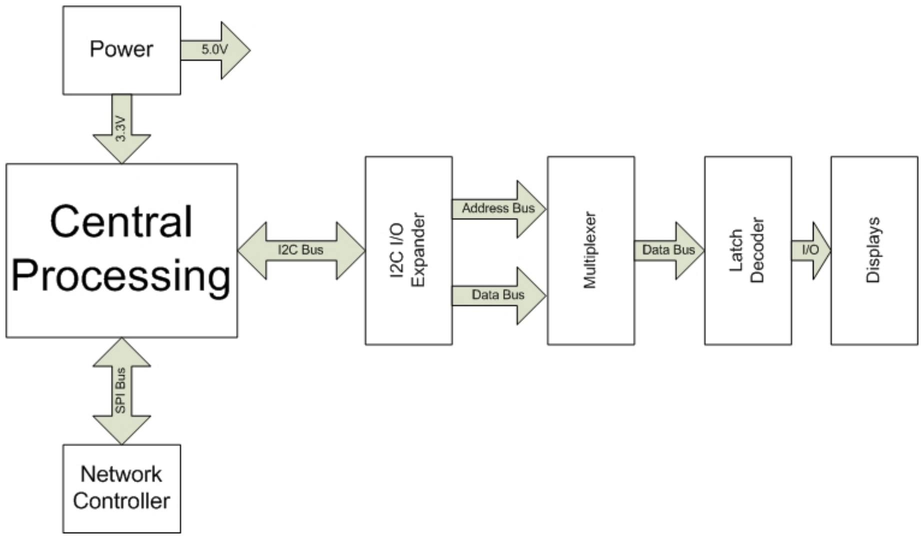

Figure 3. Block Diagram

The lower four bits of the I/O expander is connected to the data bus, while the upper four bits is connected to the input of the multiplexer. If the microcontroller had to display a 9 on the first seven-segment display, it would write 0x19 to the I/O expander and the digit 9 would be displayed on the first display. Use of the multiplexer would allow continuous control of up to seven seven-segment displays and hypothetically many more through use of the enable pins on additional multiplexers as extra bits for the address bus.

5. Teaching within an Embedded Systems Course

All embedded systems have three main components: interfacing, central processing, and communication. When teaching an embedded systems course to undergraduate students, the communication aspect is often the most difficult to teach because it is often very time consuming and the programming required has a steep learning curve. Thus ready-made code should be used instead of the students developing their own code. One of the first obstacles to using this research in a teaching module is that there is no formal curriculum to teach from. During the progress of this research, it was found that documentation provided with the uIP stack [5] was sufficient for programming the microcontroller and configuration of the stack while the books “Network programming with Perl” [12] and “Learning Perl” [13] were invaluable resources when designing the client while “Embedded C programming and the Atmel AVR” [14] was a good source for programming the microcontroller.

Also online resources such as the uIP mailing list and tutorials at AVRFreaks.com [15-17] were very helpful in the learning process. The ATmega series of microcontrollers are community driven, and aside from their informative datasheets and notes from Atmel, they describe the functions of the microcontrollers themselves. Also with the inception of the Arduino [18], the ATmega168 has such a large following that its available online resources are abundant. The only prerequisite knowledge required for this module is basics of programming taught during or before the course. Although the module involves networking, knowledge of its inner workings is unnecessary to understand its practical usage.

Although the example developed during the research used the AVR Atmel microcontroller and Perl, but different microcontrollers and languages can be used instead. Applicable microcontrollers are the Microchip PIC or the Atmel 8051 and alternate programming languages include Java, C or C++. Even another Ethernet controller such as the Micrel KSZ8851 can be used in lieu of the Microchip ENC28J60 if drivers are available for the chosen network communication stack. Using the Embedded Systems course taught at the College of Technology in the University of Houston Main campus as an example, the module could easily be taught in two classes and a lab session totaling seven hours easily during the last weeks of the class. Use of the research for developing a teaching module would be extremely useful for teaching the communication aspect of embedded systems.

6. Conclusions

This research combines knowledge from many sources such as networking, programming, and embedded system interfacing. The research has a number of applications spanning from instant global communication over the Internet with any embedded system to teaching undergraduate students how to use communication stacks. The premise that most Internet exchanges are text based opens the door to an infinite number of possibilities and applications for embedded systems.

References

[1] Wikipedia, "Internet Protocol." h*ttp://en.wikipedia.org/wiki/Internet_Protocol* Web. 18 Dec. 2009.

[2] Atmel Atmega168 Datasheet.

[3] Atmel, "Atmel: Atmega168 Product Card." *http://www.atmel.com/dyn/products/Product_card.asp?part_id=3303* Web. 18 Dec. 2009.

[4] ENC28J60 Datasheet – Microchip Ethernet Controller.

[5] A. Dunkel, "uIP Main Page." *http://www.sics.se/~adam/uip/index.php/Main_Page*. Web. 21. Dec. 2009.

[6] A. Dunkel, "uIP Documentation." *http://www.sics.se/~adam/uip/uip-1.0-refman/*. Web. 16 Nov. 2009.

[7] Tuxgraphics, “An AVR microcontroller based Ethernet device.” *http://tuxgraphics.org/electronics/200606/article06061.shtml* Web. 18 Dec. 2009.
[8] J. Derehag, “avr-uip - Port of uIP tcp/ip stack from Adam Dunkels to use with AVR microcontrollers.” *http://code.google.com/p/avr-uip/*. Web. 12 Dec. 2009.
[9] CD74HC4511E Datasheet - BCD-7 segment encoder with latch.
[10] DM74138 Datasheet - 3-8 Decoder/Multiplexer.
[11] PCF8574P Datasheet - 8-bit I^2C I/O port Expander.
[12] L. Stein, *Network Programming with Perl*. Addison-Wesley, 2004.
[13] T. Bryan, and R. Schwartz, *Learning Perl*. California: O’Reilly Media, 2005.
[14] S. Larry, R. H. Barnett, *Embedded C programming and the Atmel AVR*. Delmar Cengage Learning, 2006.
[15] Camera, Dean, “AVR Tutorials - *[TUT] [C] Newbie's Guide to AVR Timers.”* *http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=50106*. Web. 09 Sep. 2009.
[16] Weddington, Eric, “AVR Tu*torials - [TUT] [C] Bit manipulation (AKA "Programming 101").”* *http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=37871*. Web. 20 Sep. 2009.
[17] Worster, Ken, “AVR Tutorials - [TUT] [C] Newbie's Guide to the AVR ADC.” *http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=56429*. Web. 20 Sep. 2009.
[18] Arduino, “Arduino – HomePage.” *http://www.arduino.cc*

Acknowledgment

The author would like to thank Dr. Farrokh Attarzadeh for the opportunity to pursue this research and the Undergraduate Faculty for Computer Engineering Technology especially Dr. Bernard McIntyre and Dr. Driss Benhaddou for generously imparting their knowledge inside the classroom and out. The author would also like to thank the College of Technology and The Provost's Undergraduate Research Scholarship Program (PURS) for funding this research.

Biographies

ANTONIO ALEXANDER is a recent graduate of the College of Technology with a Bachelor’s in Computer Engineering Technology. He has plans to start his career in the Information Technology field as well as spend time working on embedded systems related projects. He is currently employed by the University of Houston.

FARROKH ATTARZADEH earned his PhD in Electrical Engineering from the University of Houston in 1983. He is an associate professor in the Engineering Technology Department, College of Technology at the University of Houston. He teaches software programming, digital logic, and is in charge of the senior project course in the Computer Engineering Technology Program. He has developed a concept referred to as EMFA (Electromechanical Folk Art) as a vehicle to attract young students to the STEM fields. He is the Associated Editor for student papers at *the Technology Interface Journal* (http://engr.nmsu.edu/~etti/), and Chair, Conference/Organization Member Affairs for IAJC (http://www.iajc.org/). He is a member of ASEE and has been with the University of Houston since 1983. Dr. Attarzadeh may be reached at FAttarzadeh@central.uh.edu