Keywords: MAXQ2000, uIP, network, TCP, IP, remote, DS2502-E48 Feb 29, 2008

APPLICATION NOTE 4205

Using the uIP Stack to Network a MAXQ Microcontroller

By: Zach Metzinger

Abstract: This application note describes how to network a MAXQ® microcontroller using the uIP TCP/IP network stack. A popular SPI™-to-Ethernet IC is used as the MAC/PHY for this application. The MAXQ2000 serves as the example microcontroller.

Introduction

Remote monitoring and control of a system is, perhaps, one of the most valuable capabilities of any microcontroller application. Imagine a vast lawn sprinkler system for a golf course, in which the system is expected to report faults to a master computer. Such a system would nearly eliminate the need for manual observation of proper system operation.

Many communications methods, such as RS-232 serial or infrared, could be used to implement the control and monitoring of remote devices within this system. However, all of these methods are tied to specific interfaces which involve incompatible transport mediums and protocols, and are distance limited.

Internet Protocol (IP) over Ethernet

The Internet Protocol provides a solution to the above challenges. All modern operating systems implement an IP stack. While IP can be run on a variety of transport mediums, Ethernet is, by far, the most ubiquitous. In addition, thanks to switched and routed network topologies, Ethernet is not limited by distance.

With the advent of advanced MAC/PHY integrated circuits with on-board buffers, we can use a microcontroller and IP over Ethernet to control and gather data from nearly any remote system. In this application note, we demonstrate how to network a MAXQ2000 microcontroller using the free uIP stack and a SPI-to-Ethernet IC.

Introducing uIP

uIP (pronounced "micro IP") provides a minimal IP stack which includes TCP, UDP, and ICMP protocols. uIP was developed by Adam Dunkels and released under a BSD-style license. Full source code is available on the Internet at http://www.sics.se/~adam/uip/index.php/Main_Page.

Naturally, a full implementation of TCP/IP is not necessary, nor prudent, for most applications. uIP provides a set of features which meets the minimum requirements for a fully-functional host, and does not require "special case" exceptions like some other lightweight IP stacks.

Compiler Environment

The Rowley CrossWorks 1.1 build 1 compiler for C was used to compile, assemble, and link this project. To facilitate debugging, all code optimization features were disabled. The final code size was only a small fraction of the available program flash memory on the MAXQ2000.

The MAXQ2000 has 2kB of available data memory. We must be judicious in the use of this data memory, as buffers must be allocated for the processing of IP packet contents. To optimize the use of data memory, all constant strings are stored in code space and copied to a fixed-size RAM buffer on demand.

Using the uIP Stack

The uIP stack can be viewed as an event loop with timeouts based on architecture-specific clock code. In the main loop, any packets received by the MAC/PHY will be processed by `uip_arp_ipin()` and `uip_input()`. These calls may generate an output packet, which must be transmitted by the MAC/PHY driver code.

A timer is then checked to clean up closed connections and ARP table entries which have not been seen for some time, and to invoke the application callback for retransmissions. The main event loop is boilerplate code, and need not be altered for most applications.

It is important to note that uIP implements a very small TCP window size, so that only one outstanding (un-ACKed) packet is a candidate for processing and, potentially, retransmission at a time.

The real application processing occurs in the function defined for `UIP_APPCALL`. Upon invocation, the application callback may check several functions which return uIP's current state. The most important of these states are: `uip_connected()`, `uip_closed()`, `uip_aborted()`, `uip_timedout()`, `uip_newdata()`, and `uip_rexmit()`. The first four functions manage the opening and closing of incoming connections; the latter two functions deal with incoming and outgoing data.

When new data arrives on the socket for the application to process, `uip_newdata()` will return a nonzero result. The application should then process the data, which is held in the buffer pointed to by `uip_appdata`, and, optionally, return a response. All data, including Ethernet link-layer and IP headers, is stored in `uip_appdata`.

If the network drops any part of the data returned to the peer, the application callback will be invoked through a timer timeout and `uip_rexmit()` will return a nonzero result.

It is at this point that uIP diverges from other IP stacks. Typically, TCP packet retransmission is handled by the IP stack. uIP saves memory by requiring the application to retransmit the missing data when presented with the `uip_rexmit()` flag. This retransmission can be accomplished by regenerating the data, or by keeping the previously generated data in a buffer for retransmission.

The MAC/PHY Driver

To transmit and receive IP datagrams over Ethernet, we must provide the MAC/PHY driver code to uIP. The API is fairly simple: the driver must signal that an incoming Ethernet frame is ready for processing; there must be calls to receive packets and transmit packets on the wire. These calls are implemented in `macphy.c`, and are called from the main event loop.

For this application, we chose the Microchip ENC28J60 SPI-to-Ethernet chip. This integrated MAC/PHY is accessed through SPI and features 8kB of packet buffer memory. **Figure 1** illustrates the connection between the MAXQ2000 and the ENC28J60.

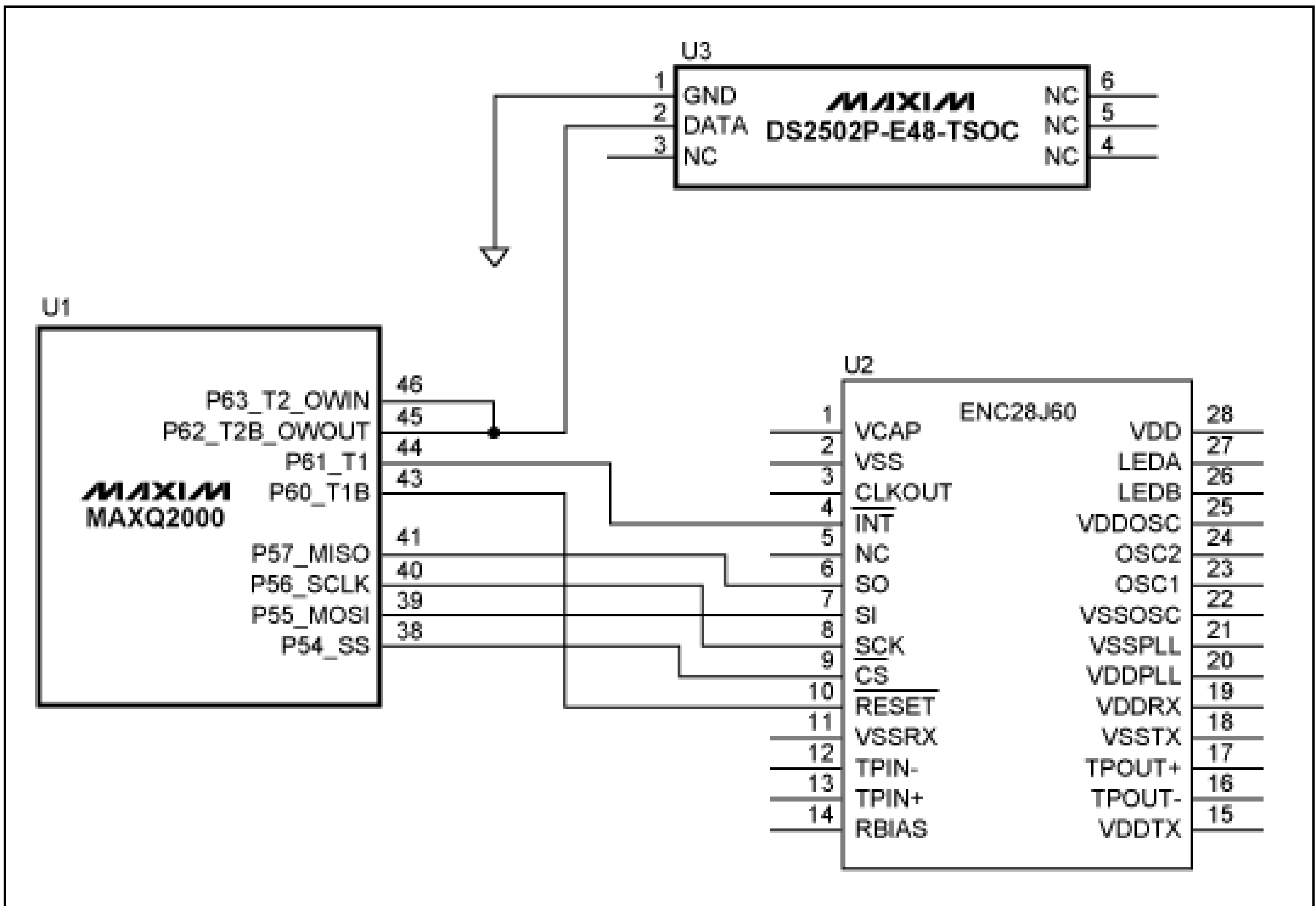

Figure 1. Schematic illustrates the connection between the MAXQ2000 microcontroller and the SPI-to-Ethernet IC.

The uIP stack, as currently implemented, assumes that the entire IP datagram resides in memory. The MAXQ2000 has 2kB of data RAM, which may need to be used for other tasks, so copying a packet from the Ethernet up to the MTU of 1500 bytes is unwise. Instead, we use the TCP feature of Maximum Segment Size to specify that we will not accept segments over 500 bytes, including the Ethernet link-layer and IP headers. This segment size provides adequate throughput for most applications. Additionally, we instruct the MAC/PHY to silently discard any packet over 500 bytes.

The example source code for this project includes the MAC/PHY driver code in the file `macphy.c` and the associated include files `macphy.h` and `macphy_priv.h`.

A Sample Application

Our sample application implements a modified echo server listening on TCP port 23. First, we start the MAXQ2000's real-time clock (RTC) for event timing. Then, we initialize the SPI hardware and load the MAC/PHY with default settings.

The application then retrieves a unique Ethernet hardware address from the 1-Wire® DS2502-E48 located on the PCB. Both the MAC/PHY driver and the uIP stack must be informed of this hardware address. Two calls, `macphy_init()` and `uip_setethaddr()`, are made to set this information.

The uIP stack is initialized with a call to `uip_init()`, and the compiled-in default IP address, netmask, and gateway are set.

When an incoming connection is completed, a greeting is printed out to the peer. Any text sent by the client is sent back in an "echo" fashion, except that all words are reversed letter-for-letter.

Ample computational resources remain to perform the primary functions of our microcontroller system. In our golf-course sprinkler example, we could use the interrupt system available on the RTC to open and close sprinkler valves at the proper times.

There are no real-time constraints on the network interface, as TCP/IP handles timeouts and retransmissions seamlessly. A TCP connection will normally take minutes to hours before dropping the connection due to timeout. Thus, the microcontroller code can manage the network connection as a background task.

Conclusion

Networking the MAXQ2000 is easily accomplished using the free uIP stack. The event-driven nature of uIP lends itself to remote monitoring and control of microcontroller systems.

This application note describes the implementation of a simple application, which can be expanded to create more complex systems. The source code for this sample application, including the MAC/PHY driver, is available for download (ZIP, 112kB).

1-Wire is a registered trademark of Dallas Semiconductor Corp.
MAXQ is a registered trademark of Maxim Integrated Products, Inc.
SPI is a trademark of Motorola, Inc.

Dallas Semiconductor is a wholly owned subsidiary of Maxim Integrated Products, Inc.

Application Note 4205: www.maxim-ic.com/an4205

More Information
For technical support: www.maxim-ic.com/support
For samples: www.maxim-ic.com/samples
Other questions and comments: www.maxim-ic.com/contact

Keep Me Informed
Preview new application notes in your areas of interest as soon as they are published. Subscribe to EE-Mail - Application Notes for weekly updates.

Related Parts

DS2502-E48:	QuickView -- Full (PDF) Data Sheet -- Free Samples
MAXQ2000:	QuickView -- Full (PDF) Data Sheet -- Free Samples

AN4205, AN 4205, APP4205, Appnote4205, Appnote 4205
Copyright © by Maxim Integrated Products
Additional legal notices: www.maxim-ic.com/legal