2005 WSEAS Int. Conf. on DYNAMICAL SYSTEMS and CONTROL, Venice, Italy, November 2-4, 2005 (pp158-161)

Ethernet Enabled Embedded Controller for Distributed Measurement and Control Applications

DORIN PETREUS, ZOLTAN JUHOS, ADRIAN GULES
Electronics and Telecommunications Faculty, Applied Electronics Department
Technical University of Cluj-Napoca
26-28, George Baritiu Street, 400027, Cluj-Napoca
ROMANIA
http://www.ael.utcluj.ro

Abstract: - Embedded systems are becoming more complex and are processing more data than ever before. Consequently they require faster means of communication to other systems, compared with traditional methods such as RS232 or RS485. One solution is TCP/IP over Ethernet, allowing communications at 10 – 100Mbps, and communication between systems spread over a large area. The “large area” is usually limited to a “local area” due to the addressing issues of the currently used IPv4 protocol. This paper proposes the implementation of an Ethernet/IPV6 enabled system that can read an array of sensors and can control an array of actuators.

Key-Words- Ethernet, IPv6, sensors, actuators

1 Introduction

In large systems, where lots of equipment has to be interconnected in an ever-changing manner a solution has to be found that can handle the communication speed, assure the data safety and support rapid reconfiguration of the connected equipment’s topology. All these conditions are intrinsically present in a twisted pair based Ethernet network, with a TCP/IP stack on top. Even if this kind of communication is considered to be too heavy for a microcontroller based system, it still can be implemented successfully in a 16-bit environment, maybe with a few compromises at the higher level protocols. Having such a secure and fast way of communication, the user application can reach new dimensions. This way secure and reliable measurement and controlling applications can be developed that use the already existent network architectures to communicate with distant systems. Usually these kinds of devices are situated in local area networks, being connected to the exterior by a server. The communication between two devices situated in different local area networks is nearly impossible because the devices aren’t using real IP addresses. This issue can be avoided by implementing an IPv6 based protocol stack. Compared to the 32 bit address space of the IPv4 protocol, the IPv6 uses 128 bits to represent the IP address, resolving once and for all the problem of real IP addresses.

2 Design of the Ethernet enabled controller

The proposed application uses a 16-bit C163 microcontroller from Infineon connected to an already existent network component that is a Realtek RTL8019 based network card, featuring an ISA bus interface.

The C163 microcontroller can use up to 5 memory zones, each of these activated by a chip select signal. Since the address space is linear, this is necessary to discriminate the program memory banks, the data memory banks and the I/O space. While the first four zones are proper for program memory or external data memory, the last zone can be used for accessing I/O mapped devices. With the exception of the first select signal, that is active when all the other select signals are inactive for each of these zones, a start address and a length can be defined. Due to the fact that the memories or peripherals selected with these signals can have different speeds, we have the possibility to introduce wait states during the bus access. Taking advantage of this fact and the classical external bus of the C163, the ISA card can be mapped in the I/O address space without the need of glue-logic.

The addressing of the RTL8019 is done in an 8 bit manner. From the ISA interface we used the address signals SA0 through SA10, SD0 through SD7, IOR, IOW and the RESET. The RESET signal must be asserted before the communication with the RTL8019 is started. To ease the decoding of the address, the most significant address line of the RTL8019 is replaced with the chip select of the microcontroller that is used to address I/O mapped devices.

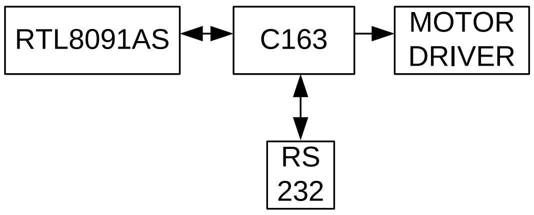


Fig.1 – The system's architecture

Usually, the network card's base address is 300h, but due to the fact that the microcontroller uses 16 bit memory accesses, the least significant bit isn't used, so the whole address space will be shifted to the left with one bit, resulting in an apparent base address of 600h. The further addressing is done ignoring the least significant address bit, to access the physical address 301h, the address 602h must be present on the address bus. The most significant address line will be replaced by the Chip Select signal. To this address we have to add the offset of the page declared to be addressed by using the CS4 signal that is 100000h.

The network controller is compatible with National Semiconductor's DP8390, as an extra, it has a set of registers for Plug and Play configuration over the ISA bus. It includes the physical layer controller and the data link layer from the OSI model. This way the data received can be presented to the network layer that is part of the TCP/IP protocol stack.

The serial port present on the application is used for debugging purposes in case that the local network is disconnected, it allows full access to the transducer parameters present in the system and memory dump commands can be issued for analyzing the state of the network protocol stack.

The present application controls a stepper motor and reads a temperature sensor. The motor driver has to be implemented and connected to the hardware. The driver is connected to a general purpose port, allowing rapid switching of the phases and eliminating the need for extra hardware to decode a certain address for this purpose. The temperature sensor is DS18S20 from Maxim Inc. This reads the temperature with a precision of up to 0.06°C. The temperature reading is done through a one wire interface that is connected to a general purpose port of the microcontroller.

From software point of view, we implemented the network interface as general as possible, so it can be used in other designs as well. The network interface is broken up into several modules, corresponding to the OSI layers. Having in mind the sharing of the resources with the other modules that will run on the microcontroller, the memory buffer for the packets arriving from the network interface card (NIC) will store only one frame. Upper layers will decode the frame from the buffer up to their representation and will return structures that point to the corresponding location in the memory buffer. Although this has the disadvantage that the packet and segment size will be limited to approximately 1400 bytes and that fragmented packets are not supported, being in an embedded environment and expecting communication with other embedded devices, this is not a problem.

The first module from the network interface is the RTL module. This implements the low level communication with the NIC, acting as a driver, taking the place of the data link layer of the OSI model. Above this module is the ETH module that will decode the frame into its structure. The IP6 [1] module uses the services of the ETH and will decode the packet by returning relevant integers or pointers. This module will also do host-to-network or network-to-host byte order conversions, due to the different endianness of the microcontroller and the Ethernet protocols. The ICMP6 [2] is a helper protocol for diagnostic, discovery, error and configuration purposes. Although it is encapsulated in an IPv6 packet, it is not considered to be a layer 4 protocol. This module will implement ICMP6 functions like auto-configuration, neighbor discovery, echo reply, etc. UDP [3] is one of the modules the application that uses the network will see. It uses the services of IP6 in order to receive and send UDP packets from/to the network. TCP [4] module is at the same level as the UDP, using the services of IP6. It is a minimal implementation of the TCP, still under development. It is used by the web server, as well as by other modules of the application that will require a TCP connection to a distant device.

The other major part of the microcontroller's software is the transducer interface. This is also intended to be a framework for transducer based applications. To be noted that *transducer* will include sensors as well as actuators. The framework supposes that each transducer will have an electronic descriptor that will describe the type and possibilities of it. Each type of transducer has a unique descriptor and a specific function list. The system that requires access to a certain transducer will first inquire the list of locally connected transducers. The connection can be done to the distant system only in certain conditions. If the transducer is an actuator, it must not be involved in a connection with another distant or local device. If the transducer is a sensor, we'll have two situations: the sensor delivers volatile data, in this case the same rules are applied as in the case of the actuators, or the sensor delivers non-volatile data, so that multiple connections can be made to it.

Getting or setting a certain transducer's values is possible only after the connection is made, according to the rules above. This is done by the means of a set of functions that will include the identifier of the connection, the parameter of interest and a pointer to the value of the parameter.

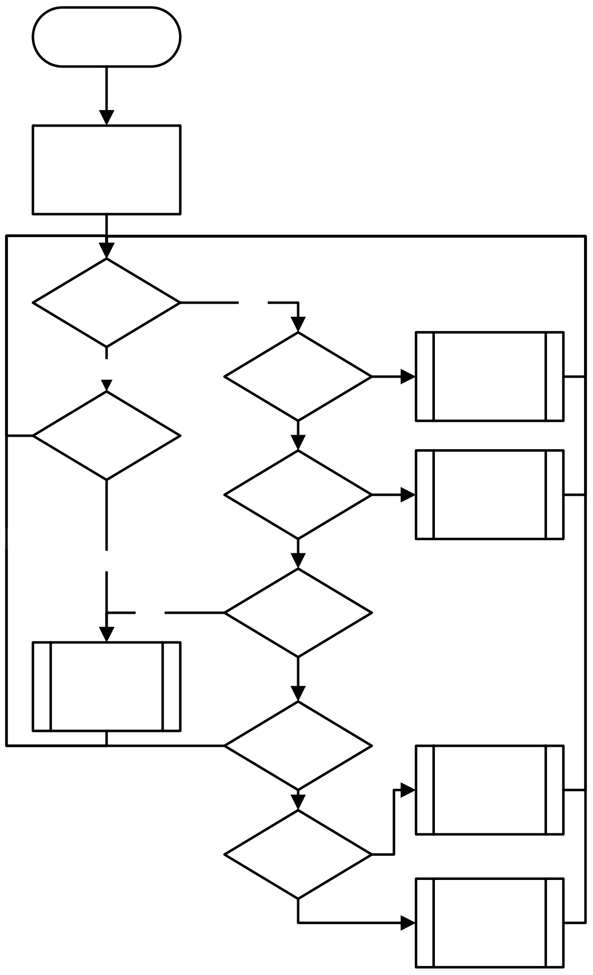


Fig. 2 – Flowchart of the application

The third major part of the application is a web server that is placed on the top of the network interface. Although the transducers can be achieved by the means of a UDP or TCP connection, the web server allows visual connection with a distant user that wants to inquire the sensors' status or control the actuators.

The rest of the application intermediates the network interface and the transducer interface, allowing access to the resources through UDP or TCP ports.

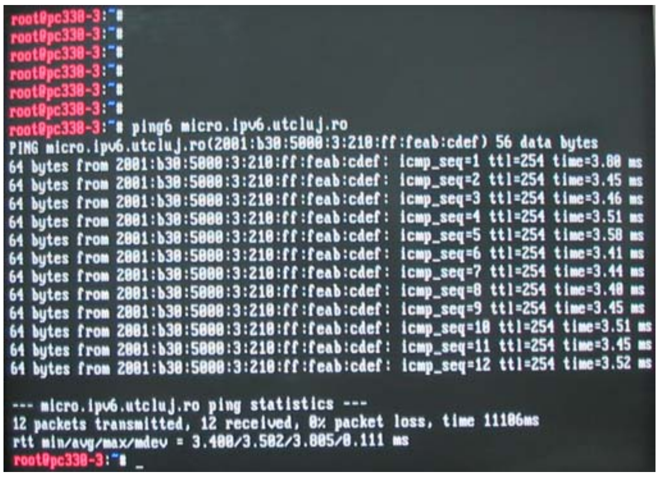
The whole application being broken up into tasks, we needed an environment to handle all the tasks in a transparent manner. Considering that a RTOS would be too complex to do the job, taking up too many of the resources, we wrote a non-preemptive scheduler, named NPTASK, based on the idea of protothreads [5]. This scheduler exports only two functions, *nptask_create* for creating a task and *nptask_yield* for ceasing the system resources to another task. The module will execute the functions in order, so each task can do its job in a sequential manner.

The transducers that we connected for testing were a stepper motor, as an actuator and a digital temperature sensor. For the motor we can set the following parameters:

- type of stepping: full step, half step
- direction: clockwise, counterclockwise
- speed: specified in milliseconds between each step (max. 100)
- number of steps: 0 up to 65535

For the temperature sensor we can set the calibration parameters and we can read the temperature.

The transducers can be reached by the means of the UDP protocol ports 2001 and 2002.



```
root@pc330-3:~#
root@pc330-3:~#
root@pc330-3:~#
root@pc330-3:~#
root@pc330-3:~#
root@pc330-3:~#
root@pc330-3:~# ping6 micro.ipv6.utcluj.ro
PING micro.ipv6.utcluj.ro(2001:b30:5000:3:210:ff:feab:cdef) 56 data bytes
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=1 ttl=254 time=3.80 ms
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=2 ttl=254 time=3.45 ms
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=3 ttl=254 time=3.46 ms
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=4 ttl=254 time=3.51 ms
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=5 ttl=254 time=3.58 ms
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=6 ttl=254 time=3.41 ms
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=7 ttl=254 time=3.44 ms
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=8 ttl=254 time=3.40 ms
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=9 ttl=254 time=3.45 ms
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=10 ttl=254 time=3.51 ms
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=11 ttl=254 time=3.45 ms
64 bytes from 2001:b30:5000:3:210:ff:feab:cdef: icmp_seq=12 ttl=254 time=3.52 ms

--- micro.ipv6.utcluj.ro ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11106ms
rtt min/avg/max/mdev = 3.400/3.502/3.805/0.111 ms
root@pc330-3:~# _
```

Fig. 3 – Echo reply from the device

The device was tested in the local network of the Technical University of Cluj-Napoca. The minimum response time that we obtained is around 3ms, with an average of 4.5-5ms.

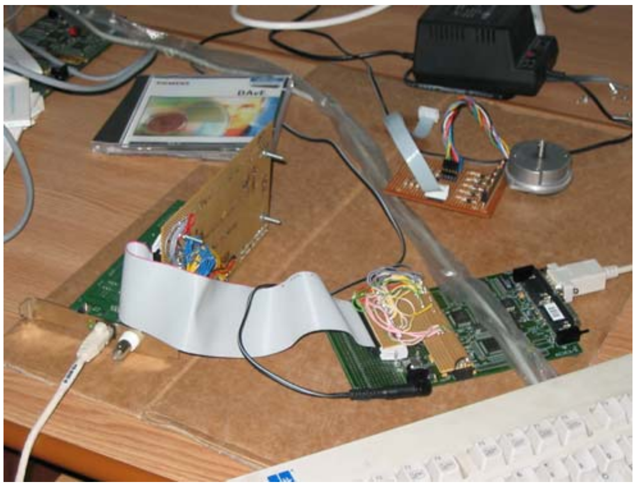


Fig. 4 – The Ethernet enabled controller

3 Conclusion

The proposed system can be used to efficiently monitor a large area of sensors or to control an area of actuators. Due to its architecture a common communication interface is achieved for communicating with other systems or a user. While from the machine's point of view it is just another communication protocol, from the user's point of view it's a great benefit, since any endpoint can be configured by the means of a web browser.

References:

[1] http://www.ietf.org/rfc/rfc1883.txt

[2] http://www.ietf.org/rfc/rfc1885.txt

[3] http://www.ietf.org/rfc/rfc768.txt

[4] http://www.ietf.org/rfc/rfc793.txt

[5]http://www.sics.se/~adam/pt/pt-1.1-refman/main.html