Volume 5, Issue 1, January 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Implementation of TCP/IP Ethernet Web Services on ARM7 LPC2148 for Embedded Ethernet Applications

R. Pallavi, C. Veeranna
Department of ECE & JNTU,
Anantapur, India

Abstract— As computer networks and embedded Internet technology rapid development of embedded systems in industrial production and daily life have been widely used. Embedded real-time operating system and dedicated hardware structure of Internet users around as long as you can at any time, any place using the system remote monitoring and control of embedded devices. Embedded system is an intelligent system that has the capability of processing, monitoring and controlling. It may comprise of Sensors, Microcontrollers, etc. The TCP/IP protocol is a widely used standard for modern digital communication. Increasing amounts of integration of microcontrollers in electrical goods in today's world and the ever growing adoption of cheap 'smart home' solutions. Consider if we can remotely monitor the status of our embedded system using a web browser, or if send an alert when it needs a service or is sold out of specific items. These things are all made possible with Embedded Ethernet. The key benefits of Embedded Ethernet Connectivity are described in the following paragraphs. Embedded connectivity stands at the forefront of today's embedded systems. So a solution need be found to realize the communication between industrial control devices and Ethernet. As the embedded system itself has the performance of network and human-computer interaction, it is possible that the embedded system replaces the previous control method based on microcontroller. In this project we are implementing an embedded web sever using ARM Microcontroller and Ethernet device. ENC28J60 is a Microchip Technology has introduced a 28-pin stand-alone Ethernet controller and also greatly simplifies the related design, reducing the space. ARM microcontroller can communicate with serial data acquisition equipments at the terminal through SPI interface and can transmit data to remote host computer through Ethernet ENC28J60 interface. ARM acting as a standalone web server, with controls for various input and output transducers. The web page(s) will allow monitoring of traducers and status of the different devices.

Key words --- Ethernet, embedded systems, web server, microcontroller and TCP/IP.

I. INTRODUCTION

1.1 INTRODUCTION TO ARM7 MICROPROCESSOR

A microprocessor system consists of a microprocessor with memory, input ports and output ports connected to it externally. A microcontroller is a single chip containing a microprocessor, memory, input ports and output ports. Since all four blocks reside on one chip, a microcontroller is much faster than a microprocessor system. We have several other basic microcontroller families such as PIC, M68HCXX, AVR etc .All these basic microcontrollers are useful for implementing basic interfacing and control mechanism for simple applications. There are several applications which require lot of computation and high speed data processing. In such applications advanced microcontrollers and microprocessors are used. One such advanced architecture is ARM. The ARM7TDMI core is a 32-bit embedded RISC processor delivered as a hard macro cell optimized to provide the best combination of performance, power and area characteristics. The ARM7TDMI core enables system designers to build embedded devices requiring small size, low power and high performance.

The ARM7 family also includes the ARM7TDMI processor, the ARM7TDMI-S processor, the ARM720T processor and the ARM7EJ-S processors, each of which has been developed to address different market requirements.

The market for microprocessors continues to diversify, based on the evolving demands of applications including wireless, home entertainment, automotive and microcontrollers. ARM core families sharing the ARMv7 architecture will cover the widening spectrum of embedded processing. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles. The RISC instruction set and related decode mechanism are much simpler than those of Complex Instruction Set Computer (CISC) design.

This simplicity gives:

· A high instruction throughput
· An excellence real-time interrupts response
· A small, cost-effective, processor macro cell

The ARM7TDMI core is the industry's cost widely used 32-bit embedded RISC microprocessor solution. Optimized for cost and power-sensitive application, the ARM7TDMI solution provides low power consumption, small size, and high performance needed in portable, embedded application.

The ARM7DMI-S is synthesizable version of ARM7TDMI core. The ARM720T hard macro cell contain the ARM7DMI core, 8KB unified cache and MMU (Memory Management Unit) that allows the use of protected execution space and virtual memory.

The ARM7EJ-S processor is synthesizable core that provides all the benefit of ARM7DMI, while also incorporating ARM's latest DSP extensions and jazelle technology, enabling acceleration of Java-based applications.

1.2 ARM ARCHITECTURE

The ARM core uses a RISC architecture aimed at delivering simple but powerful instructions that execute within a single cycle at a high clock speed. The RISC philosophy concentrates on reducing the complexity of instructions performed by the hardware because it is easier to provide greater flexibility and intelligence in software rather than hardware. As a result RISC design plays greater demands on the compiler. In contrast, the traditional complex instruction set computer (CISC) relies more on the hardware for instruction functionality and consequently the CISC instructions are more complicated.

The ARM7 core is based on the von Neumann architecture with 32-bit data bus that carries both instruction and data. Data can be of 8 bits, 16 bits, 32 bits. It has following features:

1) One cycle execution time
2) Instruction pipeline
3) Memory format
4) Operation modes
4) Coprocessor
5) Debugging feature

LARGE NUMBER OF REGISTERS:

The RISC design philosophy generally incorporates a larger number of registers to prevent in large amounts of interactions with memory. Any register can contain either data or an address. Registers act as the fast local memory store for all data processing operation.

LOAD-STORE ARCHITECTURE:

The processor operates on data held in registers. Separate load and store instructions transfer data between the register bank and external memory.These design rules allow a RISC processor to be simpler, and thus the core can operate at higher clock frequencies.

ARM PROCESSOR CORE:

Similar to most RISC machines ARM works on load-store architecture, so only load and store instructions perform memory operations and all other arithmetic and logical operations are only performed on processor registers. The figure shows the ARM core data flow model. In which the ARM core as functional units connected by data buses. And the arrows represent the flow of data, the lines represent the buses, and boxes represent either an operation unit or a storage area. The figure shows not only the flow of data but also the abstract components that make up an ARM core.

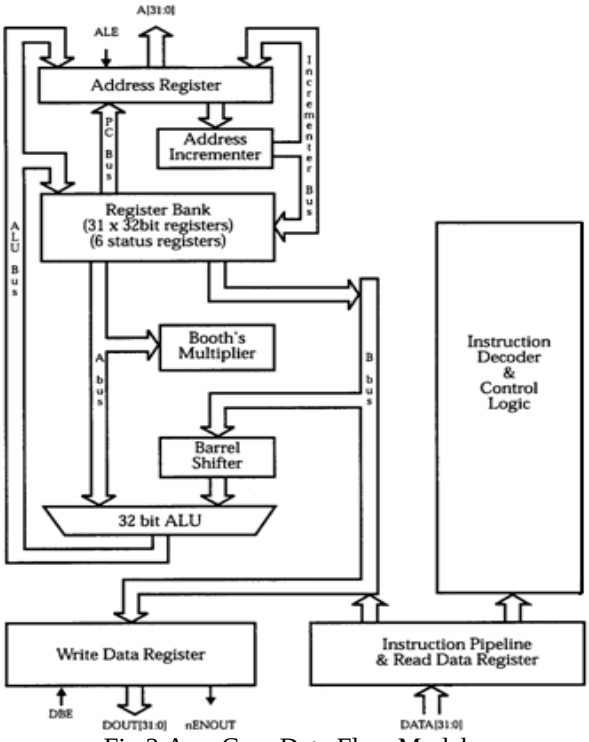

Fig 2 Arm Core Data Flow Model

In the above figure the Data enters the processor core through the Data bus. The data may be an instruction to execute or a data item. This ARM core represents the Von Neumann implementation of the ARM data items and instructions share the same bus. In contrast, Harvard implementations of the ARM use two different buses.

The instruction decoder translates instructions before they are executed. Each instruction executed belongs to a particular instruction set.

The ARM processor like all RISC processors, use a load-store architecture. This means it has two instruction types for transferring data in and out of the processor: load instructions copy data from memory to registers in the core, and conversely the store instructions copy data from registers to memory. There are no data processing instructions that directly manipulate data in memory. Thus, data processing is carried out solely in registers.

Data items are placed in the register file – a storage bank made up of 32-bit registers. Since the ARM core is a 32- bit processor, most instructions treat the registers as holding signed or unsigned 32-bit values.

The sign extend hardware converts signed 8-bit and 16-bit numbers to 32-bit values as they are read from memory and placed in a register.

The ALU (Arithmetic logic unit) or MAC (multiply – accumulate unit) takes the register values Rn and Rm from the A and B buses and computes a result. Data processing instructions write the result in Rd directly to the register file. Load and store instructions use the ALU to generate an address to be held in the address register and broadcast on the Address bus.

One important feature of the ARM is that register Rm alternatively can be preprocessed in the barrel shifter before it enters the ALU. Together the barrel shifter and ALU can calculate a wide range of expressions and addresses.

After passing through the functional units, the result in Rd is written back to the register file using the result bus. For load and store instructions the incrementer updates the address register before the core reads or writes the next register value from or to the next sequential memory location. The processor continues executing instructions until an exception or interrupt changes the normal execution flow.

II. PROPOSED WORK

Our stack is a collection of different modules. Some modules (such as IP, TCP, UDP and ICMP) must be called when a corresponding packet is received. Any application utilizing our stack must perform certain steps to ensure that modules are called at the appropriate times. This task of managing stack modules remains
the same, regardless of main application logic.

In order to relieve the main application from the burden of managing the individual modules, our TCP/IP Stack uses a special application layer module known as "StackTask", or the Stack Manager. This module is implemented by the source file "StackTsk.c". StackTask is implemented as a cooperative task; when given processing time, it polls the MAC layer for valid data packets. When one is received, it decodes it and routes it to the appropriate module for further processing.

It is important to note that Stack Manager is not an integral part of our TCP/IP Stack. It is written so that the main application does not have to manage stack modules, in addition to its own work such as RS-232, LCD, and Sensor interfacing. Before the Stack Manager task can be put to work, it must be initialized by calling the StackInit( ) function. This function initializes the Stack Manager variables and individual modules in the correct order. Once StackInit( ) is called, the main application must call the StackTask( ) function periodically, to ensure that all incoming packets are handled on time, along with any time-out and error condition handling.

The exact steps used by StackTask are shown in the algorithm below.

```
If a data packet received then
        Get data packet protocol type
        If packet type is IP then
                Fetch IP header of packet
                Get IP packet type
                If IP packet type is ICMP then
                        Call ICMP module
                Else if IP packet type is TCP then
                        Call TCP module
                Else if IP packet type is UDP then
                        Call UDP module
                Else
                        Handle not supported protocol
                End If
        End If
Else if packet type is ARP then
        Call ARP module
End If
```

Thus the protocols supported by our project are MAC, ARP, IP, TCP, UDP, ICMP, and HTTP. The FTP, DHCP and other protocols such as SLIP are not supported. The absence of DHCP implementation in our project causes the IP address of the microcontroller to be fixed and unable to be changed by the network server.

REFERENCES

[1] Li Min; Liu Ying; Su Zhihua, "An Embedded Remote Temperature Measurement System," Digital Manufacturing and Automation (ICDMA), 2013 Fourth International Conference on , vol., no., pp.148,151, 29-30 June 2013

[2] I. Talzi, A. Hasler, S. Gruber, and C. Tschudin, "Permasense: Investigating permafrost with a WSN in the Swiss Alps," in *Proc. 4th Workshop Embedded Netw. Sensors*, New York, 2007, pp. 8–12.

[3] P. Harrop and R. Das,Wireless sensor networks 2010–2020, IDTechEx Ltd, Cambridge, U.K., 2010.

[4] N. Burri, P. von Rickenbach, and R. Wattenhofer, "Dozer: Ultra-low power data gathering in sensor networks," in *Inf. Process. Sensor Netw.*, Apr. 2007, pp. 450–459.

[5] I. Dietrich and F. Dressler, "On the lifetime of wireless sensor networks," *ACM Trans. Senor Netw.*, vol. 5, no. 1, pp. 5:1–5:39, Feb. 2009.

[6] B. Yahya and J. Ben-Othman, "Towards a classification of energy aware MAC protocols for wireless sensor networks," *Wireless Commun. Mobile Comput.*, vol. 9, no. 12, pp. 1572–1607, 2009.

[7] J. Yang and X. Li, "Design and implementation of low-power wireless sensor networks for environmental monitoring," *Wireless Commun., Netw. Inf. Security*, pp. 593–597, Jun. 2010.

[8] K. Martinez, P. Padhy, A. Elsaify, G. Zou, A. Riddoch, J. Hart, and H. Ong, "Deploying a sensor network in an extreme environment," *Sensor Netw., Ubiquitous, Trustworthy Comput.*, vol. 1, pp. 8–8, Jun. 2006.

[9] A. Hasler, I. Talzi, C. Tschudin, and S. Gruber, "Wireless sensor networks in permafrost research—Concept, requirements, implementation and challenges," in *Proc. 9th Int. Conf. Permafrost*, Jun. 2008, vol.1, pp. 669–674.

[10] J. Beutel, S. Gruber, A. Hasler, R. Lim, A. Meier, C. Plessl, I. Talzi, L. Thiele, C. Tschudin, M. Woehrle, and M. Yuecel, "PermaDAQ: A scientific instrument for precision sensing and data recovery in environmental extremes," in *Inf. Process. Sensor Netw.*, Apr. 2009, pp. 265–276.

[11] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *Proc. 7th Symp. Operat. Syst. Design Implement.*, Berkeley, CA, 2006, pp. 381–396.

[12] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli, "The hitchhiker's guide to successful wireless sensor network deployments," in *Proc. 6th ACM Conf. Embedded Netw. Sensor Syst.*, New York, 2008, pp. 43–56.

[13] R. Szewczyk, J. Polastre, A.Mainwaring, and D. Culler, "Lessons from a sensor network expedition," *Wireless Sensor Netw.*, pp. 307–322,2004.

