Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Co-design for an SoC embedded network controller

ZOU Lian-ying†, ZOU Xue-cheng

(*Department of Electronic Science & Technology, Huazhong University of Science & Technology, Wuhan 430074, China*)

†E-mail: zou_ly@126.com

Received May 21, 2005; revision accepted Nov. 23, 2005

Abstract: With the development of Ethernet systems and the growing capacity of modern silicon technology, embedded communication networks are playing an increasingly important role in embedded and safety critical systems. Hardware/software co-design is a methodology for solving design problems in processor based embedded systems. In this work, we implemented a new 1-cycle pipeline microprocessor and a fast Ethernet transceiver and established a low cost, high performance embedded network controller, and designed a TCP/IP stack to access the Internet. We discussed the hardware/software architecture in the forepart, and then the whole system-on-a-chip on Altera Stratix EP1S25F780C6 device. Using the FPGA environment and SmartBit tester, we tested the system's throughput. Our simulation results showed that the maximum throughput of Ethernet packets is up to 7 Mbps, that of UDP packets is up to 5.8 Mbps, and that of TCP packets is up to 3.4 Mbps, which showed that this embedded system can easily transmit basic voice and video signals through Ethernet, and that using only one chip can realize that many electronic devices access to the Internet directly and get high performance.

Key words: System-on-Chip (SoC), Embedded, Microprocessor, Network controller, TCP/IP, Co-design
doi:10.1631/jzus.2006.A0591 **Document code:** A **CLC number:** TN4

INTRODUCTION

With the development of Ethernet systems and the growing capacity of modern silicon technology, embedded communication networks are playing an increasingly important role in embedded and safety critical systems. Advances in VLSI technology have also pushed integration to the point where it is now possible to design and implement a microprocessor and network controller on a single chip, known as System-on-Chip (SoC). In a network of embedded systems, each system can communicate with the other systems in the network, sharing information and sending and responding to requests as needed.

Embedded devices need to be designed to solve specific problems. It is a challenge to find the right balance between power and cost. This becomes even more complicated when adding network capability to a device. One route is to try to extend an original 8-bit micro-controller into the networking world. While it is feasible, it will inevitably be slow. Another approach is to use an embedded processor device (e.g. TMS320VC33) and a network transceiver (e.g. RTL8019AS) to buildup a multi-chip system (Pan, 2004) which will be fast and responsive, but costly.

The advent of Field Programmable Gate Arrays (FPGAs) with thousands of logic gates has made it possible to verify specific software functions on specific hardware (Thomas *et al*., 2000). This reduces the design cycle and hence the execution cycle time to make the embedded system respond faster in real-time. We used FPGA technique to design a new 1-cycle pipeline microprocessor and a fast Ethernet MAC to build a low cost, high performance embedded network controller.

STATE OF HARDWARE/SOFTWARE CO-DESIGN

Hardware/software co-design is a methodology for solving design problems in processor based embedded systems and allows the concurrent design of both hardware and software, thereby reducing the design time and also meets the performance goals.

Conventional design lengthens the design time and significantly increases the amount of redesign when trade-offs between hardware and software implementations become clear late in the design process. A significant part of the design problem consists of deciding the software and hardware architecture for the system and also the parts that should be implemented in software running on programmable components and which should be implemented in specialized hardware. The flexibility and high density of logic packing in FPGA open new applications for digital circuits and lead to new hardware/software co-design issues.

Attention must be given in embedded systems to co-design techniques that can accelerate software implementation. So the partitioning of the system into hardware and software is of critical importance as it has major impact on the system level cost/performance in embedded systems (de Micheli and Gupta, 1997). For an embedded network controller, the most common procedure is to deal with the Ethernet packet. In order to accelerate the processing procedure of Ethernet MAC, we put it in hardware part. The software mainly deals with TCP/IP packet flow. The calculating procedure of TCP/IP checksum can be implemented more rapidly in hardware circuit than in software code. It also can offload the embedded microprocessor. So we implement TCP/IP checksum arithmetic in special hardware circuit.

The detailed partitioning of the embedded network controller system is described in the following sections. The last section of this paper discusses the FPGA verification of the SoC system.

HARDWARE ARCHITECTURE

Embedded network controller offers embedded devices to access Internet. The main parts of the hardware architecture are embedded microprocessor and network access controller. A powerful memory control unit is implemented to buildup the chip's expansibility. Fig.1 shows the whole hardware part of this embedded network controller.

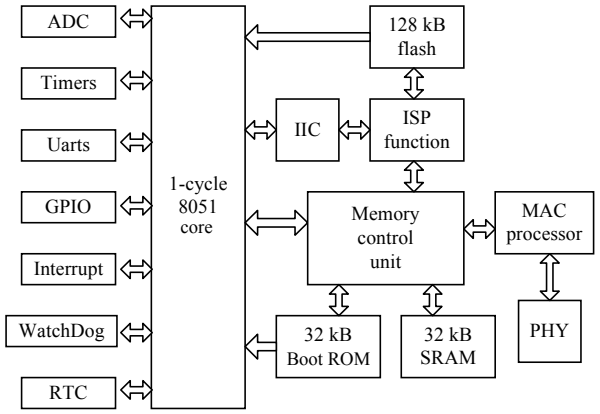

Fig.1 Hardware architecture

One-cycle microprocessor core

8051 is one of the most popular 8-bit microprocessor architecture (Salcic, 1997). Its instruction set is so simple to understand, so most embedded system designers like to use it. Many instructions can access I/O pins directly. So the internal MCU core can control peripherals rapidly. On the other hand, users can easily get integrated development environment to get started with their application development.

However, the bottleneck of embedded system is the original 8051 core. Although the crystal oscillator's frequency is up to 40 MHz, one machine cycle still needs 12 clock cycles, and each instruction's execution time needs from 1 machine cycle to 4 machine cycles. It means that running an instruction needs from 12 to 48 clock cycles. So the throughput is limited with 3 MIPS (million instructions per second) even only running one series of NOP instructions.

In order to improve the embedded system's performance, it is necessary to redesign the 8051 core. By analyzing the original 8051-core design, we can find that there is hugely waste in the 12 clock cycles. Most instructions are forced to do nothing in one machine cycle. The DS80C400 is a fast 8051-compatible micro-controller that implements instructions three times faster than an original 8051 at the same crystal speed. Its maximum system-clock frequency of 75 MHz results in minimum instruction cycle time of 54 ns. But in the DS80C400, a machine cycle requires 4 clocks. There is still waste in the 4 clocks. So we present a new high performance 8051 core. We first separate the data bus and address bus, using concurrent processing to control data bus and address bus. Then we split instruction implementation time into two steps: fetch decode step and execute step, which means that we use a two level pipeline to

implement more instructions in parallel. The fundamental innovation of the high-speed 8051 core is the use of only one clock cycle per instruction cycle compared with 12 for the original 8051. This results in up to ten times improvement in performance.

In addition, the high-speed microprocessor is updated with several new peripherals and features while providing all of the standard features of an 80C32. These include 256 bytes of on-chip RAM for variables and stack, 32 I/O ports, three 16-bit timer/counters, and an on-chip UART.

The 1-cycle high-speed microprocessor studied in a previous section is an 8051-compatible device that provides improved performance and power consumption compared to the original version and retains instruction-set and object-code compatibility with the 8051, yet performs the same operations in fewer clock cycles. Consequently, more throughput is possible for the same crystal speed. The high-speed microprocessor's more efficient design allows faster crystal speed to get high performance than an original 8051, while using much less power.

Memory control unit

The high-speed microprocessor uses a memory-addressing scheme that separates program memory from data memory. The program and data segments can overlap since they are accessed in different ways. Instructions are fetched by the MCU core automatically. So the program addresses are never written by software. The data memory area is accessed explicitly using the MOVX instruction with multiple ways to specify the target address. Memory control unit allows software to dynamically increase program and data memory beyond the 64 kB-limit by using bank-switching techniques.

The program code in an embedded system may be referred to as firmware, which usually means that the code is stored in flash memory or other nonvolatile memory, and tends to be an integral, seldom-changing part of the chip. Users may have the ability to load new firmware into a device, with the new firmware being typically an update or upgrade of the existing code, rather than an entirely different application type. There is therefore a 32 kB ROM that allows the device to act as a bootstrap loader for the internal flash through Ethernet or serial port.

Security is very important for software programmers. We add a security control circuit for this embedded system. After programmers ensure the firmware in the flash, a special write to the flash's information block will startup the safety circuit. Any illegal read or modify firmware operations will cause mass erase of the flash. However normal update operation will be better supported through ISP mode.

Ethernet access control

The Ethernet control part has high performance and is lightly integrated with the local MCU bus. The controller implements both 10 Mbps and 100 Mbps Ethernet function based on IEEE802.3/IEEE802.3u LAN standard, and can auto-negotiate with the link partners to achieve better link configurations and also provides an extra IEEE802.3u compliant media-independent interface (MII) to support other media applications. Fig.2 shows the details of the Ethernet control.

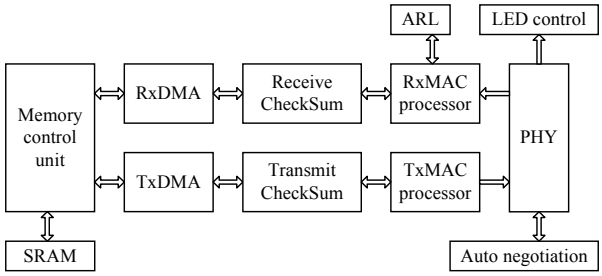

Fig.2 Ethernet access architecture

The Ethernet controller has a local memory to hold packets. Both the packets to be transmitted and the packets received from the network are first buffered in the local memory.

When the host has a packet to send, the host first constructs the packet in its memory, then transfers it byte by byte to the local memory. After the packet is completely transferred from the host's memory to the local memory, the host then issues a command to transmit the packet to the network. To reduce data latency, there is another 8-byte buffer between the local memory and the Ethernet twisted-pair. The DMA controller transfers the packet byte by byte from the local memory to the FIFO. Each byte in the FIFO is then serialized (change from byte to bits), encoded (e.g. change to Manchester encoding), and transmitted to the Ethernet twisted-pair.

When a packet is received from the network, each byte is first decoded, de-serialized, and stored in the FIFO, which can be programmed with a threshold value. When the number of bytes in the FIFO reaches a threshold, the bytes are transferred from the FIFO to the local memory with a local DMA operation. When the complete packet is received in the local memory, the controller informs the host with an interrupt. The driver's Interrupt Service Routine (ISR) then copies the data from the local memory to the host memory.

It is well known that in typical Web-server workloads, a major portion of CPU utilization is attributable to the TCP/IP stack (Bentham, 2002). Much of the TCP overhead is actually attributable to its implementation error handling. So we add TCP/IP checksum to offload the embedded MCU.

SOFTWARE ARCHITECTURE

This firmware facilitates building "Internet-enabled" devices that communicate with other computers or intelligent devices on the Internet using the TCP/IP protocol stack (Comer and Stevens, 2001). Fig.3 summarizes the implemented protocol stack.

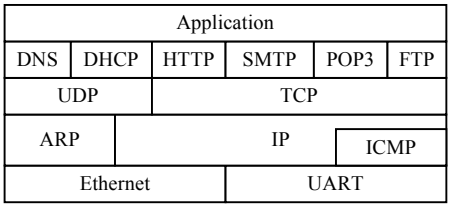

Fig.3 TCP/IP protocol stack

Data memory considerations

The hardware contains on-chip 32 kB data memory. Use of this limited amount of SRAM must be carefully planned, as one may need to make design trade-offs between the memory requirement and the overall system performance. Fig.4 summarizes the journey of individual packets as they pass through the various protocol layers and buffers.

The minimum data memory requirements for the firmware stack are listed below:

(1) Each socket needs a transmit buffer, the minimum size is 2 kB.

(2) Each socket can have an optional receive buffer. If it does, the minimum size is 2 kB.

(3) Data link layer buffer: 1.5 kB.

(4) Physical layer buffer: minimum 4 kB for Ethernet receive and 1.5 kB for Ethernet transmit.

Assuming that the application has to support 5 sockets, and that each socket has no receive buffer, the minimum memory requirement is then 10 kB+1.5 kB+1.5 kB+4 kB=17 kB. So 15 kB are left for the application buffer and other program variables.

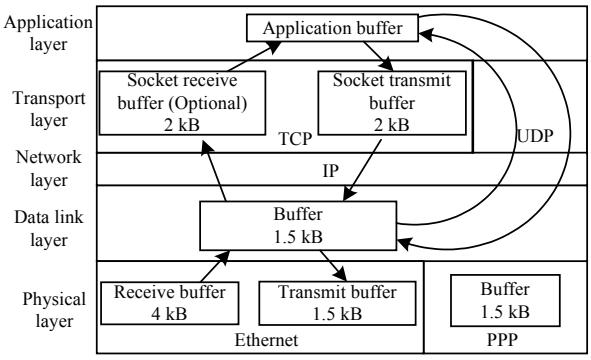

Fig.4 Packet flow diagram

Firmware architecture

The firmware includes two parts: firmware in ROM and firmware in flash. The firmware in ROM implements FTP protocol for updating firmware in flash. The steps are as follows:

(1) Initialize the Ethernet controller;

(2) Initialize the serial port;

(3) Initialize the timer function;

(4) Implement FTP client function.

The firmware in flash starts up by initializing various data structures and I/O devices. After initialization the firmware enters the main superloop, in which the Ethernet and serial port interfaces are polled constantly for incoming frames. If a frame is received it is passed up the protocol stack for processing. The main super loop also checks for any data queued for transmission, and a physical frame is created and transmitted on the Ethernet or the serial port.

When a physical frame is received over the Ethernet, the firmware checks the "type" field within the Ethernet frame. If the type is ARP, an ARP response is constructed and transmitted across the Ethernet interface. If the type is IP, the firmware searches through the protocol type: ICMP, TCP or

UDP. If it is an ICMP echo request, an echo response is generated. If the type is TCP or UDP, the main loop calls the TCP or UDP routines to process the datagram. If there is application level data, TCP or UDP then delivers the data to the application.

FPGA VERIFICATION AND LAYOUT DESIGN

After completing hardware and software implementation, the whole embedded system can be verified using FPGA techniques. We select Altera Stratix EP1S25F780C6 to verify this network controller. After the top level of hardware RTL code to FPGA top code is first identified, we use Quartus II to create 32 kB ROM and 32 kB RAM to replace the ASIC modules. Then we use Keil to compile the firmware for Boot and create hex file for ROM, and finally use Quartus II to synthesize and create sof file, and FPGA downloads it to the main FPGA chip of EP1S25F780C6 through the JTAG interface. The environment is shown in Fig.5.

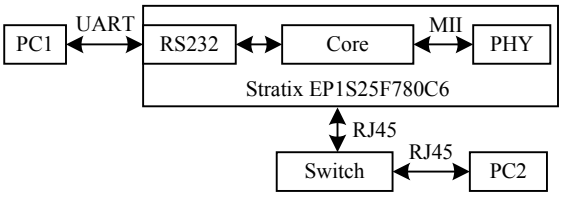

Fig.5 FPGA environment

A UART debug tool in PC1 can observe the detailed information on the embedded network controller. An FTP server can be installed in PC2. New firmware can be downloaded to internal flash through Ethernet and UART.

Throughput is one of the most important parameters of an Internet device. By connecting the FPGA board to a SmartBit tester, we can test Ethernet packets, UDP packets and TCP packets. Fig.6 shows distinctly that the maximum throughput of Ethernet packets is up to 7 Mbps, that of UDP packets is up to 5.8 Mbps, and that of TCP packets is up to 3.4 Mbps. So this embedded system can be used to facilitate transmission of basic voice and video signals.

After the functions of the hardware/software system were verified, we use 1st Silicon 0.25 μm process to make the layout. Fig.7 is the pictorial view

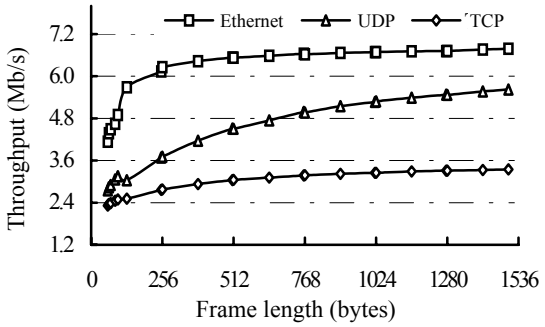

Fig.6 Throughput performance diagram

of the last version layout. The whole chip area is 4800 μm×4578 μm. The digital logic cells are about 10^5 gates. The overall cost of one chip is about one dollar. It is obvious that this single chip is smaller and cheaper than that multi-chip system described in the preceding parts of this section.

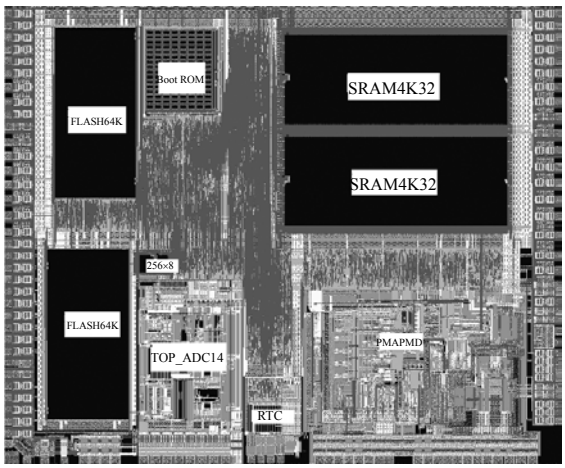

Fig.7 Layout pictorial view

CONCLUSION

This paper starts with an introduction to methodologies for designing embedded systems, focusing on hardware/software co-design embedded network controller. First, the hardware architecture and key technologies are discussed. We redesigned a new 1-cycle 8051 core, using low cost to implement a high performance MCU. Then, we describe the firmware used for TCP/IP stack. After completing the hardware part and software part design, the whole system on chip

was verified on Altera Stratix EP1S25F780C6 device. Using the FPGA environment and SmartBit tester, we tested the throughput of this SoC system. The performance diagram showed that we could use this chip to realize that many electronic devices access to the Internet directly and get high performance.

References

Bentham, J., 2002. TCP/IP LEAN-Web Server for Embedded Systems. CMP Books.

Comer, D.E., Stevens, D.L., 2001. Internetworking with TCP/IP Vol. II: Design, Implementation, and Internals (3th Ed.). Publishing House of Electronics Industry, Beijing (in Chinese).

de Micheli, G., Gupta, R.K., 1997. Hardware/Software co-design. *Proceedings of IEEE*, **85**(3):349-365. [doi:10.1109/5.558708]

Pan, Z.J., 2004. 8-bit MCU based embedded Internet technology. *World of Electronic Product*, **2**:36-38 (in Chinese).

Salcic, Z., 1997. A Micro-controller/FPGA-based prototyping system for embedded applications. *Microprocessors and Microsystems*, **21**(4):249-256. [doi:10.1016/S0141-9331(97)00041-0]

Thomas, F., Nayak, M.M., Udupa, S., Kishore, J.K., Agrawal, V.K., 2000. A hardware/software co-design for improved data acquisition in a processor based embedded system. *Microprocessors and Microsystems*, **24**(3):129-134. [doi:10.1016/S0141-9331(00)00065-X]