We are IntechOpen,
the world's leading publisher of
Open Access books
Built by scientists, for scientists

5,300
Open access books available

130,000
International authors and editors

155M
Downloads

154
Countries delivered to

Our authors are among the
TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™
Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

1

Networking Applications for Embedded Systems

Sorin Zoican
Politehnica University of Bucharest
Romania

1. Introduction

The chapter is organized as follows: a briefly introduction that motivate the necessity of networking embedded systems, a background section which illustrates the basic resources used to accomplish the networking embedded systems (Blackfin microcomputer, Visual DSP Kernel and LWIP suite), two sections that discuss the frameworks for networking application development and performance evaluation, and conclusion. A VoIP system based on AMR codec is illustrated, as a complex networking application example.

Embedded networking applications are now more and more used to send out multimedia content (audio and image) over wired or wireless networks. Control applications and sensor networks are additional areas where adding network means is desirable. More or fewer all systems connected to the Internet communicate using the IP protocol stack (Deborah Estrin, 2001, Gregory Pottie and William Kaiser, 2005). Owing to the supremacy of IP networks, many networked embedded systems are connected to such networks and therefore must be able to communicate using such protocols. However, the IP protocol suite is often apparent to be "heavyweight" in that an achievement of the protocols requires many memory resources and processing power (Adam Dunkels, 2005). The understanding that the IP protocols would require large memory leads to using large microprocessors. If the IP protocol stack is carried out using lesser amount of memory, then smaller microprocessors could be chosen. This would not only make the resulting systems less expensive to manufacture, but would also enable a whole class of smaller embedded systems to communicate using the IP protocol. On the other hand, if the microprocessor is large, and the IP protocol uses less memory then more complex applications may be accomplished.

For the embedded systems, the cost is a limiting element that constrains the resources such as memory and processor capabilities. As a result, many embedded systems do not have more than a few tens kilobytes of memory that make impractical to run the TCP/IP from Linux or Windows. The main difficulty is the memory constraint. The processing power is not quite a difficult problem owing to current technology improvements. The solution to the dilemma of running TCP/IP inside constrained memory limits is developing a very small TCP/IP implementation capable to run on a system with very little memory, called "lightweight" IP (LWIP). The LWIP was carried out for a powerful microcomputer, Blackfin BF5xx (Analog Devices Inc., ADI) using the EZ-KIT LITE BF5xx evaluation boards.

Making an embedded system for networking applications require detailed knowledge of hardware and software resources needed to develop it. Architectural elements such as computational units, address units, control units and memory management are presented for a correct evaluation of the computational power of Blackfin family processors used in complex networking applications that include digital signal processing like as voice-over IP system. The LWIP protocol stack benefits of many operating system primitives. Well understanding of the operating system kernel functioning, such as scheduling, semaphore, memory management, is necessary for writing the software needed in networking applications. The LWIP implementation for Blackfin family must be detailed while the networking application uses the protocol stack resources (functions, memory allocation).

2. Background

This section shows the elements that help achieve an embedded system with LWIP capabilities: the Blackfin microcomputer, as hardware support, the Visual DSP Kernel (VDK) real-time operating system kernel, as software support and LWIP protocol stack.

2.1 Blackfin microcomputer

The Blackfin microcomputer is a 16-bit fixed-point processor based on the micro-signal architecture (MSA) core, developed in cooperation by Analog Devices and Intel. Low-cost and high performance features make Blackfin suitable for computationally applications including video equipment and third-generation cell phones (Sorin Zoican, 2008). The Blackfin microcomputer may have embedded Ethernet and controller area network. The Figure 1 illustrated the Blackfin general architecture (Analog Devices, 2006).

The MSA is designed to attain high-speed digital signal performance and top power efficiency. This architecture combines the best capabilities of microcontroller and DSP processor into a single programming model. A dynamic power management circuit continually monitors the running software and dynamically adjusts the voltage and the frequency at which the core runs. As a result, power consumption and performance for real-time applications, such node in sensor networks, are optimized.. The Blackfin core combines dual multiply-accumulate (MAC) units, an orthogonal reduced instruction-set computer (RISC) instruction set, single instruction, multiple data (SIMD) programming capabilities, and multimedia processing features into a unified architecture. As shown in Figure 1, the Blackfin BF5xx processor includes system peripherals such as parallel peripheral interface (PPI), serial peripheral interface (SPI), serial ports (SPORTs), general-purpose timers, universal asynchronous receiver transmitter (UART), real-time clock (RTC), watchdog timer, and general-purpose input/output (I/O) ports. The Blackfin processor has a direct memory access (DMA) controller that efficiently transfers data between external devices/memories and the internal memories without processor involvement. Blackfin processors offer L1 cache memory for fast accessing of both data and instructions.

Blackfin processors have well-to-do peripheral supports, memory management unit (MMU), and RISC-like instructions, usually found in many high-end microcontrollers. These processors have high-speed buses and highly developed computational units that support variable-length arithmetic operations in hardware. These features make the Blackfin processors appropriate to replace other high-end DSPs and MCUs. The Blackfin processor uses a modified Harvard architecture, which allows multiple memory accesses per clock cycle. The Blackfin processor instruction set is optimized so 16-bit operation codes are the frequently used instructions. Complex DSP instructions are encoded into 32-bit operation codes like multifunction instructions. Blackfin microcomputers bear a limited multi-issue facility, where a 32-bit instruction can be issued in parallel with two 16-bit instructions. The programmer can use several core resources in a single instruction cycle. The Blackfin architecture supports instructions that control vector operations. We take advantage of these instructions to carry out concurrent operations on multiple 16-bit values (add, subtract, multiply, shift, negate, pack, and search instructions).

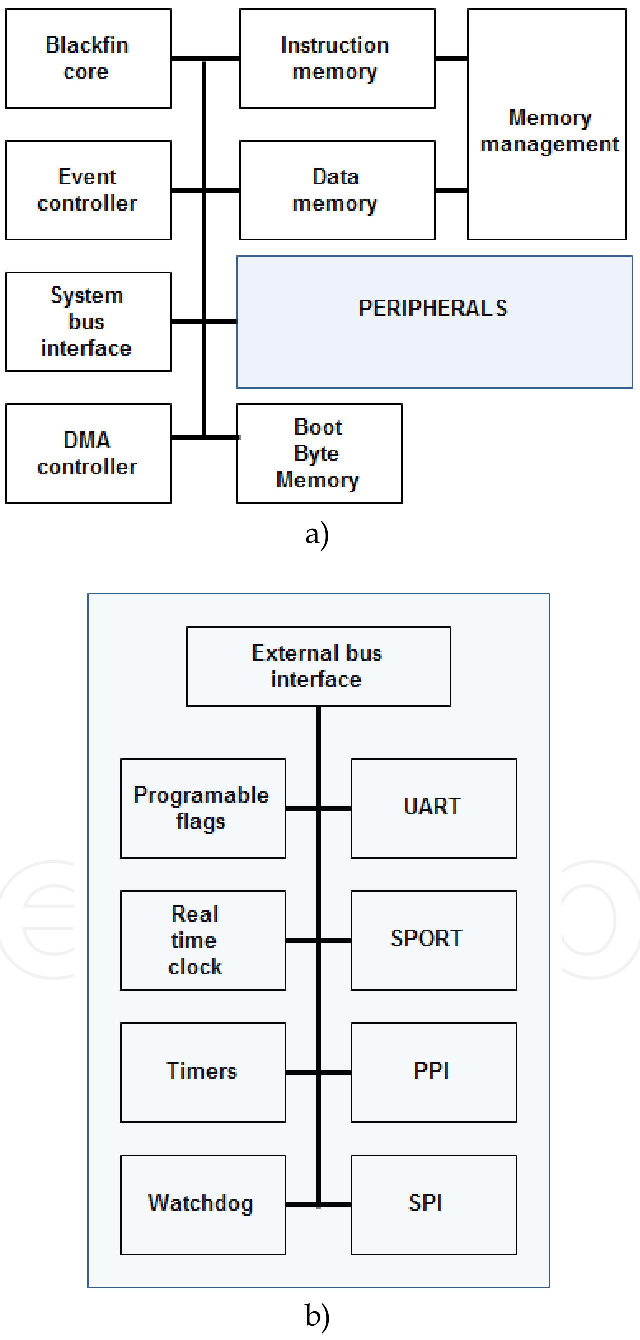


Fig. 1. a) Overall Blackfin architecture; b) The Blackfin peripherals

The Blackfin microcomputers family includes dual-core processors, such as the ADSP-BF561 microcomputer. Besides to other features, dual-core processors append a new facet to application development. Each dual-core Blackfin processor has two Blackfin cores, A and B, each with internal L1 memory. The two cores have a common internal memory shared between them. The cores share access to external memory. Each core functions autonomously: they have their reset address, Event Vector Table, instruction and data caches. On reset, core A starts running from its reset address, whereas core B is disabled. Core B starts running when it is enabled by core A. When core B starts running, it starts running its application, from its reset address. Having one application per core the complete potential of the dual-core Blackfin processor is exploiting. Effectively, two single core applications are building autonomously, and run in parallel on the processor. The common memory areas, both internal and external, are each subdivided into three areas: a section dedicated to core A, a section dedicated to core B, and a shared section.

Figure 2 shows that the core architecture consists of three main units: the address arithmetic unit, the data arithmetic unit, and the control unit.

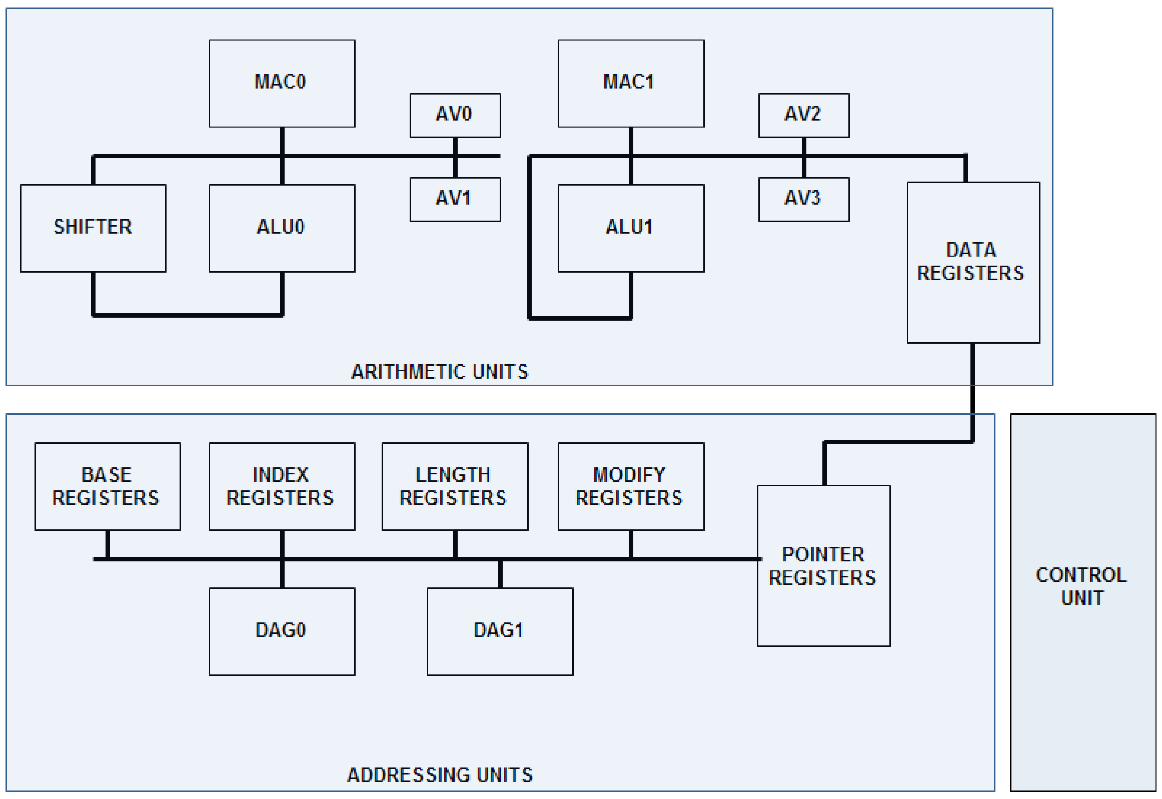
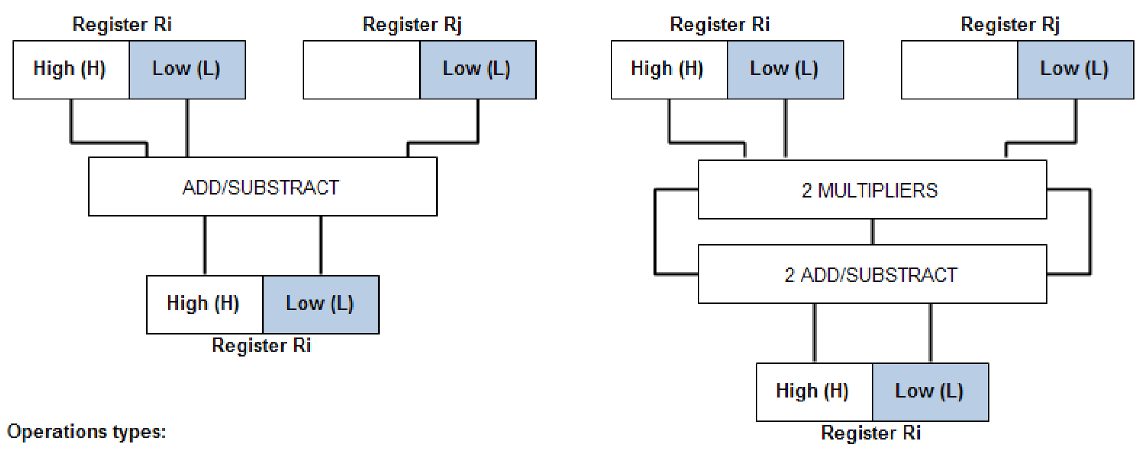


Fig. 2. The Blackfin core

The arithmetic unit supports SIMD operation and it has Load/Store architecture. The assembler instruction syntax is algebraic; it is insightful and makes it simple to understand what the instruction does. Figure 3 illustrates several of arithmetic instruction efficiently executed in a single cycle. The video ALUs offer parallel computational power for video operations — quad 8-bit add/subtract, quad 8-bit average, SAA (Subtract-Absolute-Accumulate). Quad 8-bit ALU instruction takes one cycle to complete.



Operations types:

- single 16 bits addition (Rk.H=Ri.L+Rj.H)
- dual 16 bits add and subtract (Rk=Ri+|-Rj)
- quad 16 bits addition and subtract (Rk=Ri+|-Rj, Rp=Ri-|+Rj) where p=k+1
- single 32 bits addition (Rk=Ri+Rj)
- dual 32 bits add and subtract (Rk=Ri+Rj, Rp=Ri-Rj) where p=k+1
- dual 16 bits multiplications (A1=Ri.H*Rj.H, A0=Ri.L*Rj.L)
- dual 16 bits multiply/accumulate (Rk.H=A1-=Ri.H*Rj.H, Rk.L=A0+=Ri.L*Rj.L)

Fig. 3. Blackfin arithmetic instructions

A program sequencer controls the instruction execution flow, which includes instruction alignment and instruction decoding. The Blackfin processor supports two loops with two sets of loop counters, loop top and loop bottom registers to handle looping. Hardware counters calculate the loop condition.

Blackfin sequencer manages events that include: interrupts (hardware and software), exceptions (error situation or service related). Despite the Blackfin processor has a Harvard architecture, it has a single memory map shared between data and instruction memory. Instead of using a single large memory, the Blackfin processor supports a hierarchical memory model as shown in Figure 4. The L1 data and instruction memory are placed on the chip and are generally smaller but faster than the L2 external memory, which has a superior capacity. As a result, transmission data from memory to registers in the Blackfin processor is set in a hierarchy from the slowest memory (L2) to the fastest memory (L1).

IntechOpen

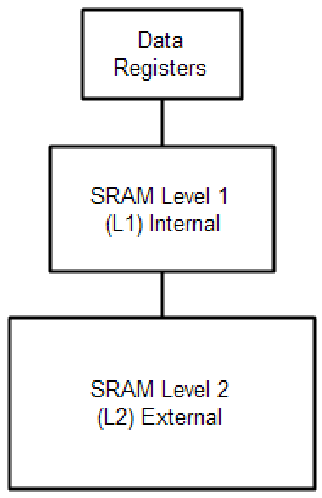


Fig. 4. Blackfin memory model

The justification following hierarchy memory is based on three principles: (1) the principle of building the common case fast, where code and data that have to be accessed repeatedly are stored in the fastest memory; (2) the principle of locality, where the program reuse instructions and data that have been used in recent times; and (3) the principle of smaller is faster, where smaller memory has faster access time.

All the shown features, make the Blackfin microcomputers family an ideal candidate to develop the embedded real-time system, which can run complex applications including networking applications with LWIP support.

2.2 The real-time operating system Visual DSP Kernel (VDK)

Other problem that must be solved, for networking applications, is to develop an operating system that knows how to run on the embedded system.

The resource restrictions and application characteristics of embedded systems put particular requirements on the operating systems running on the embedded system. The applications are usually event-based: the application performs nearly all of its work in reply to external events. Early researches into operating systems for sensor networks identified the requirements and proposed a system, called TinyOS, which solved many problems. Yet, in the TinyOS, a set of features normally found in larger operating systems, such as multithreading and run-time module loading is not accomplished. The multithreading and run-time loading of modules is wanted features of an operating system for embedded systems with networking applications.

The VDK includes the features above and runs in the resource limitations of a sensor node, and thus show that these features are feasible for sensor node operating systems. This section illustrates the main characteristics of a real-time kernel for DSP processors. The real-time kernel Visual DSP Kernel (VDK), produced by Analog Devices, is illustrated (Analog Devices 2, 2007).

The Visual DSP kernel is a strong real-time operating system kernel. It provides critical kernel features. Features include a completely preemptive scheduler (time slicing and cooperative scheduling are as well supported), thread creation, semaphores, interrupt management, interthread messaging, events, and memory administration (memory pools and multiple heaps). In multiprocessors environments, messaging is provided.

Processes are created by a primitive operating system call, which makes memory allocation for process execution.

Real-time operating system for DSP applications manage signal processor resources and have the following functions:

1. Process scheduling according to the priority given to each. The processes having all resources available less CPU is placed in a ready queue.
2. Communication between interdependent processes.
3. Synchronization process with the external environment and other processes.
4. Exclusive use of shared resources.

The real-time kernel state diagram is illustrated in Figure 5.

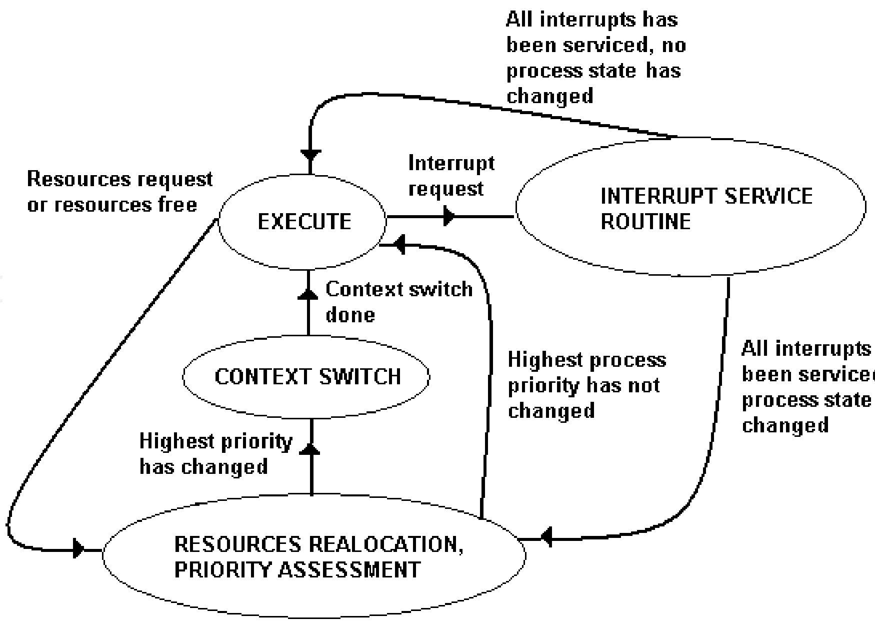

All interrupts has been serviced, no process state has changed
Resources request or resources free
EXECUTE
Interrupt request
INTERRUPT SERVICE ROUTINE
Context switch done
CONTEXT SWITCH
Highest process priority has not changed
All interrupts has been serviced process state has changed
Highest priority has changed
RESOURCES REALOCATION, PRIORITY ASSESSMENT

Fig. 5. The real-time kernel state diagram

Execution of the processes will be based on the priority (dynamically allocated) associated with each process. In VDK, three methods of scheduling can be defined: cooperative scheduling, uniform time division (round robin) and preemptive scheduling. Cooperative scheduling is involved for processes with the same priority. Each process takes the DSP processor and passes it, on own initiative, to the next process. The running process will yield the processor by calling a primitive system; the suspended process will be placed in the ready queue in the last position. The round robin scheduling assigns to processes with equal priority equal length time quantum for execution.

Semaphores, events and I/O flags are used to achieve communication and synchronization between processes. A process waiting a signal may continue execution (whether the signal is available) or can enter the blocking state whether the signal is not available. The process will unlock when the signal becomes available or when a timer expires. Semaphores are global variables in the system; therefore, they are available to any process. A semaphore is a data structure that can be used to:

- Control access to shared system resources
- Allow synchronization of processes
- Periodic executions of processes

The semaphore may be posted or it may be pending. Pending the semaphore means testing its value with zero. If the semaphore is zero then the process increases the semaphore value and access the shared resource, otherwise the process waits. When the process has finished using the shared resource it posts the semaphore. Posting a semaphore means that its value is decremented and the resource is make available. Processes interact with semaphores by real-time kernel routines for both semaphores pending and post.

The VDK kernel has all the necessary features to support the LWIP protocol stack development (memory management, semaphores and process scheduling).

2.3 The LWIP implementation on Blackfin microcontrollers

In embedded system architecture, RAM is the most demanding resource. With only a little RAM available for the TCP/IP stack to use, mechanisms used in conventional TCP/IP cannot be straight applied (Adam Dunkels, 2007).

Two memory management solutions may be chosen: dynamic buffers or single global buffer. For first memory management solution, the memory for storing connection state and packets is dynamically allocated from a global group of available memory blocks.

The second memory management solution does not use dynamic memory allocation. As an alternative, it uses a single global buffer for storing packets and has an unchanging table for holding connection state.

The total memory used for LWIP implementations depends deeply on the applications of the particular device in which the implementations are to be run. The memory arrangement determines both the amount of traffic the system should be capable to handle and the maximum of concurrent connections. A device that will be transported large e-mails while running a web server with extremely dynamic web pages and multiple concurrent clients need more RAM than an undemanding Telnet server. TCP/IP implementation with as small as 200 bytes of RAM is achievable, but such a pattern will provide very low throughput and will allow few of simultaneous connections.

The LWIP stack, (Analog Devices 2, 2010), uses the dynamic memory allocations, with VDK support. A general framework for network applications development, based on LWIP implementation, will be defined in the next section. The LWIP stack package on Analog Devices Blackfin family of processors uses a standardized driver interface to permit it to be used with different Ethernet controllers. The drivers are each accomplished as part of the Analog Devices Inc. (ADI) driver model and use the System Services Libraries (SSL). The stack provides the standard BSD socket API to the application and has been planned to decouple it from both the working environment and the particular network interface being used. The stack is connected to the nearby environment by two standardized interfaces (TCP wrapper and LWIP library) accomplished as a different library for separate environment shown in Figure 6.

The kernel API abstracts the operating system services. The kernel abstraction simplifies the movement of LWIP stack to different operating system environments. The fundamental services contain synchronization services, interrupt services, and timer based callback services (Analog Devices 1, 2007).

Blackfin processors can take benefit of the system service library (SSL), which provides reliable, easy C language access to Blackfin features: the interrupt manager, direct memory access (DMA), and power management units. Clock frequency and voltage can be changed without difficulty at run time through a set of simple APIs. Interrupt handling is fired at the time of the event, or delayed to a time of the application's choosing. A device manager integrates device drivers for on-chip and off-chip peripherals. The SSL is operating system neutral and can be run as a separate or with a real-time operating system (RTOS).

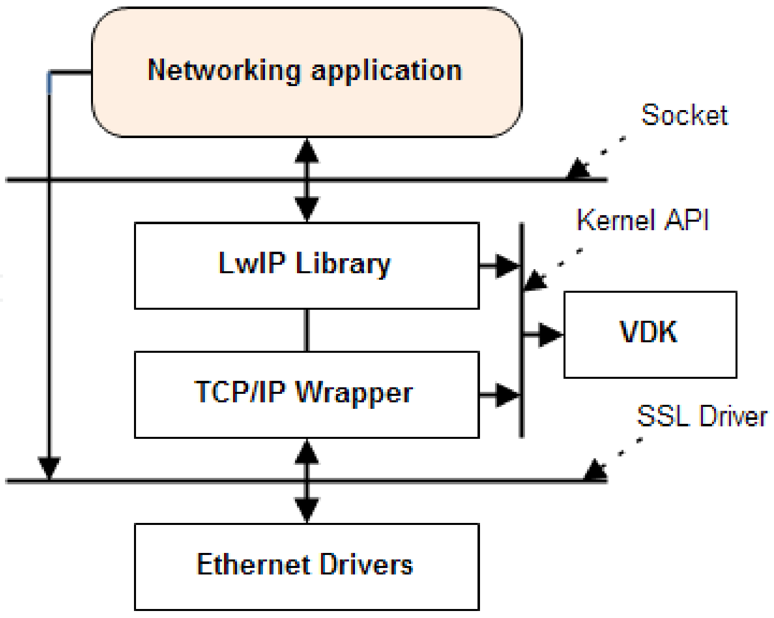


Fig. 6. The Analog Devices LWIP architecture

3. The framework of networking applications for telecommunications development using LWIP and VDK

This section presents a framework for networking applications for telecommunications. Such applications involve complex computation (for example, digital signal processing).

In the proposed framework, the programmer must be aware of all capabilities of Blackfin family (arithmetic instructions, addressing mode, hardware loops, interrupts and memory management) presented in section 3.

The LWIP stack can be run also as a task in a multitasking system or as the main program in a single tasking system. In both cases, the main control loop, illustrated in Figure 7, performs two operations repeatedly: check whether a packet has arrived from the network and check whether a periodic timeout has occurred. This pattern will be used in the framework for application development and performance evaluation.

An application that requirements to use the LWIP stack with VDK is responsible for creating a VDK thread type that the LWIP stack will use to create any threads that it requires during operation. The application also has to initialize several system service library components besides to creating an instance of the driver for the appropriate Ethernet controller.

The steps involved in creating an application that uses the LWIP stack can be summarized as (Analog Devices 2, 2010):

1. Specifying the header file for the socket API
2. Guarantee that enough VDK semaphores are configured
3. Initialize the SSL interrupt and device driver managers
4. Initialize and set up the kernel API library
5. Open the device driver for the Ethernet MAC controller and pass the handle for the driver to the LWIP stack

6. Configure external bus interface unit (EBIU) controller to provide DMA priority over processor
7. Provide the MAC address that will be used by the device driver
8. Give memory to the device driver to enable it to bear the appropriate number of concurrent reads and writes
9. Inform the device driver library that the Ethernet driver will use the dataflow method
10. Initialize and build up the LWIP stack supplying memory for the stack to use as its internal heap
11. Tell the Ethernet driver that it should now start to run
12. Wait for the physical link to be established

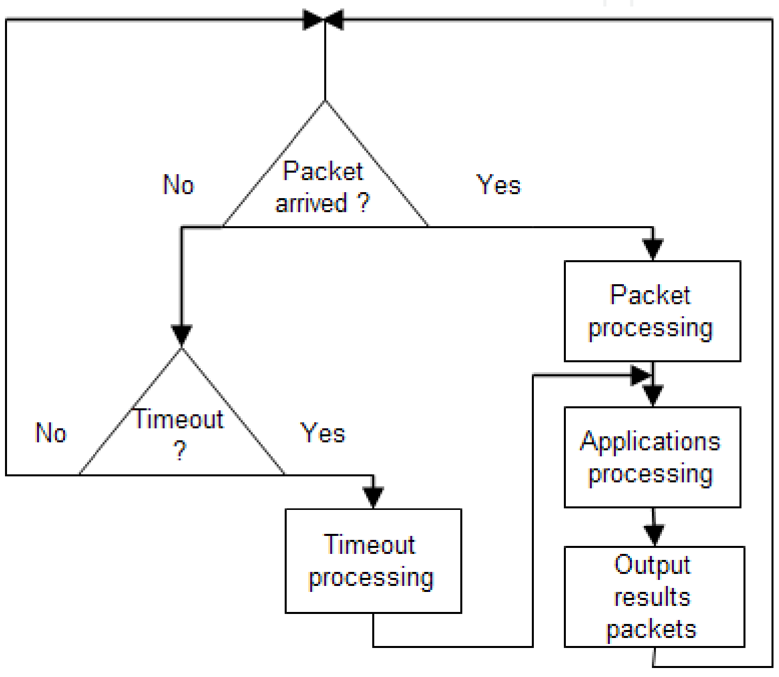


Fig. 7. The main control loop for LWIP stack

The framework for networking applications for telecommunications is based on the following assumptions (Sorin Zoican, 2011):

- there are two groups of tasks: the first manage digital processing of analog signals (such as audio or video samples) and the second deals with packets processing among the network (receive and transmit packets)
- the first group has complex tasks and the second group has less complex tasks
- block processing may be necessarily
- the system should work in real-time
- there are two type of information: signals, constituted by samples of analog signals, and data, constituted by binary results which will be transferred over network

These tasks will be scheduled using the VDK primitives (illustrated in section 3). All the scheduling methods are involved: cooperative scheduling for collaborative tasks, round robin scheduling for equal priority tasks, and preemptive scheduling for higher priority tasks. Several system tasks will be created to interact with network card using system service library, as it is stated in section 3.

If it is necessarily, two independent applications may be carried out using a dual core microcontroller such as Blackfin BF561. The application in the fist core implements the tasks of first group and the application in the second core may implements some simple tasks in the first group and the networking tasks (these tasks are less complex). This strategy was validated by implementing a voice-over IP system based on an adaptive multirate (AMR) codec (Redwan Salami et al., 2002). In the first group (tasks A) consider an AMR encoder and in the second group (task B) consider an AMR decoder, the setting of the codec rate and the packet processing that consists of receiving and transmitting packets over network using UDP protocol (Johan Sjöberg et al., 2002).

The AMR codec has high computational demands. Many specific signal processing operations must be completed in real time. The Blackfin powerful instruction set (as can observe in Figure 3), memory management, events control and the peripherals included in its architecture make realizable a real-time implementation of the both AMR codec and networking processing in a VoIP embedded system.

A technique called *switching buffers* was involved, to ensure a real-time functioning of the communication system (Woon-Seng Gan. and Sen M. Kuo, 2007). In this technique, there are pairs of input and output buffers that will be switched periodically. One pair of buffers is used for processing and the other pair is used for receiving the inputs and sending previous results.

For the particular case of AMR-VoIP system, the following buffers are defined:

- Signal_RX[0] and Signal_RX[1] – for input frames
- Signal_TX[0] and Signal_TX[1] – for output frames
- Data_RX[0] and Data_RX[1] – for coder results
- Data_TX[0] and Data_TX[1] – for decoder inputs

The input frames have N sample length each and the coder/decoder results buffer have length of M results each. Two flags, *flag_A* and *flag_B*, are defined to control the program flow in the core A and core B. A variable *counter* is defined to manage the sample frames acquisition. An interrupt is generated by the audio analog to digital converter (ADC) every input speech sample. The interrupt routine service acquires the new sample, store it in the current signal input buffer and transmit to the digital to analog converter (DAC) the signal sample from current output buffer. After N signal samples have been acquired, the signal and data buffers will be switched. A necessary condition for real-time operating is that the acquisition time for input current frame must be greater than the processing time of this frame. The buffers, flags and counter, above defined, are shared resources of the two cores in the Blackfin microcomputers. Figures 8 to 12 illustrated the switching buffer technique, flowcharts of core A, core B and interrupt service routine, respectively. The variables *i* and *j* are the acquisition buffer and the processing buffer indexes. The networking processing is illustrated in Figure 13.

The application code, written in C, may be optimized for speed (Analog Devices 1, 2010). Several techniques for C optimizations were used:

- build-in functions for fractional data
- using circular buffers and hardware controlled loops

- interprocedural analysis (IPA) - the compiler sees the entire source files used in a final link at compilation time and to use that information while optimizing

The most computational expensive block in the voice-over IP, based on AMR codec, is the AMR encoder, carried out in core A. The rest of computations, (AMR decoder, setting AMR rate at each frame and the networking processing) are less computational expensive.

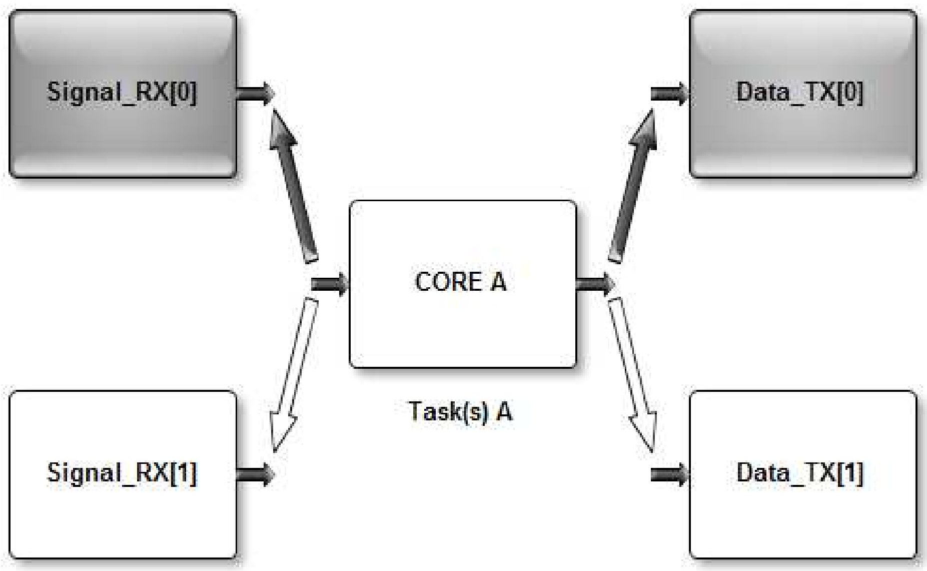


Fig. 8. Switching buffers technique in core A

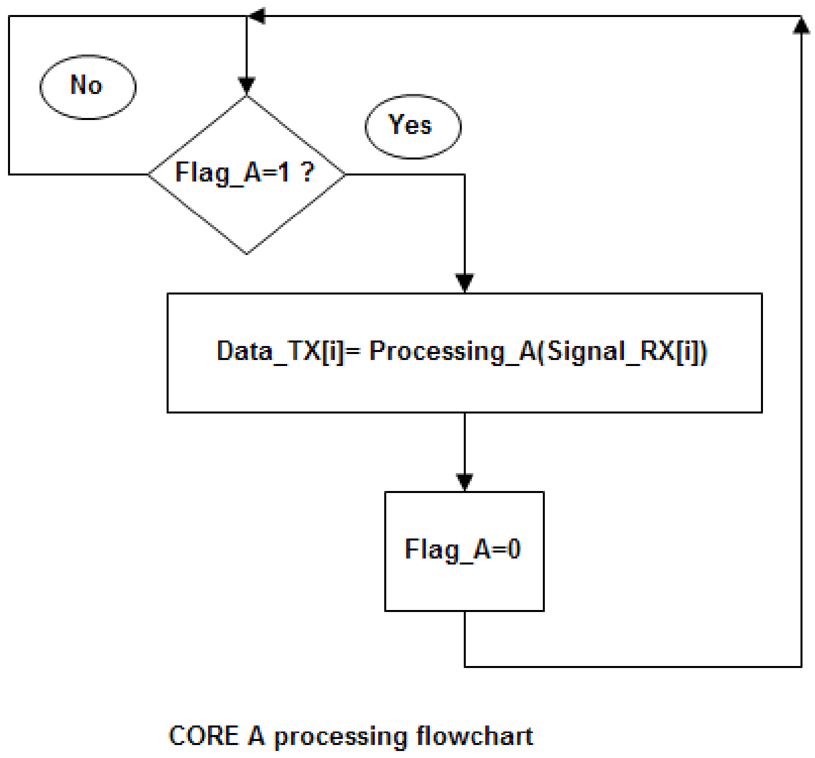

No
Yes
Flag_A=1 ?
Data_TX[i]= Processing_A(Signal_RX[i])
Flag_A=0
CORE A processing flowchart

Fig. 9. Program flowchart in core A

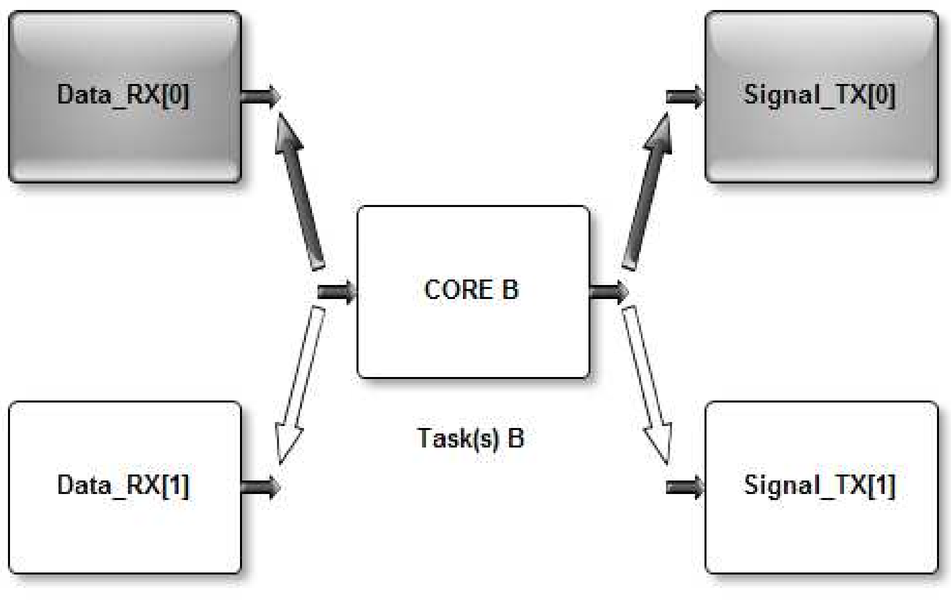


Fig. 10. Switching buffers technique in core B

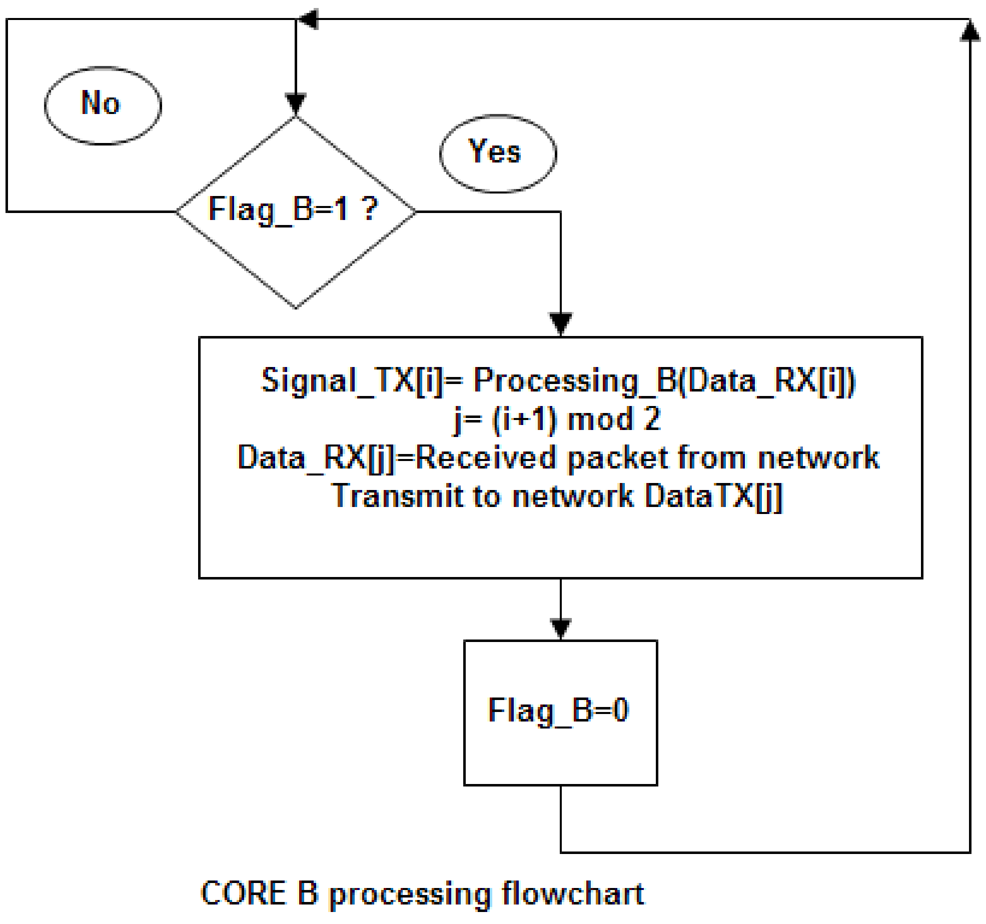

No
Yes
Flag_B=1 ?
Signal_TX[i]= Processing_B(Data_RX[i])
j= (i+1) mod 2
Data_RX[j]=Received packet from network
Transmit to network DataTX[j]
Flag_B=0
CORE B processing flowchart

Fig. 11. Program flowchart in core B

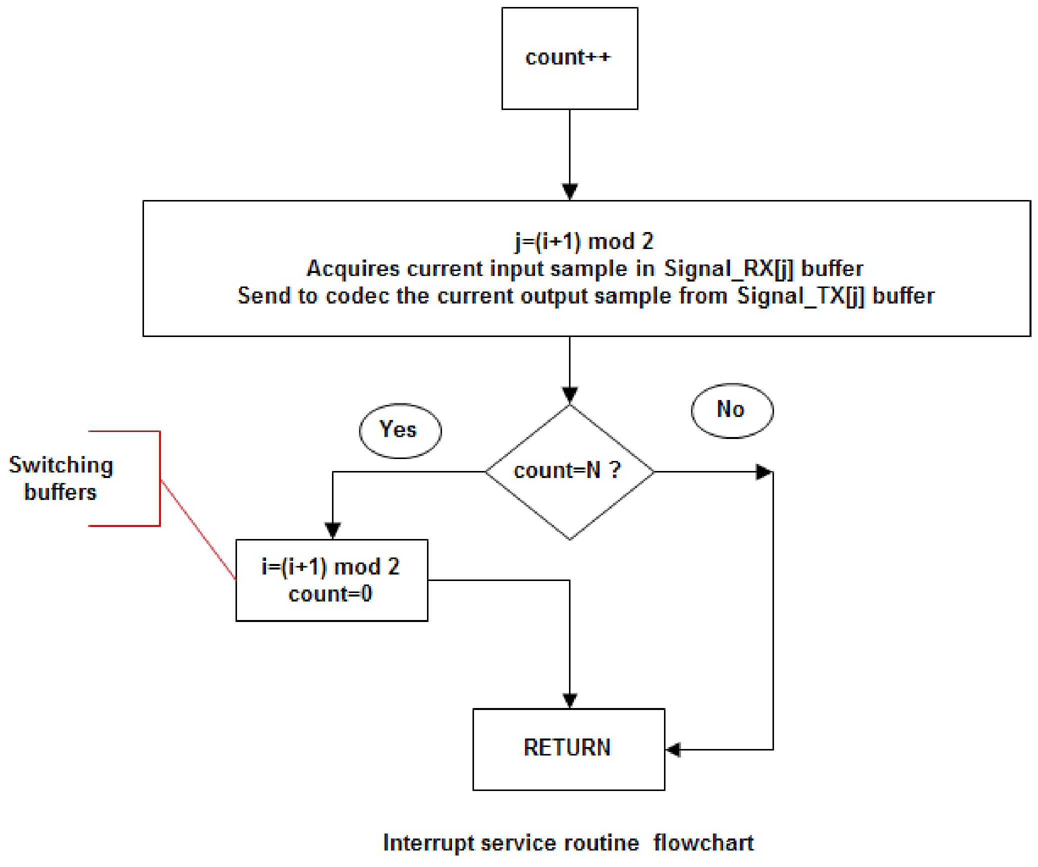

count++
j=(i+1) mod 2
Acquires current input sample in Signal_RX[j] buffer
Send to codec the current output sample from Signal_TX[j] buffer
count=N ?
Yes
No
Switching buffers
i=(i+1) mod 2
count=0
RETURN
Interrupt service routine flowchart

Fig. 12. ISR flowchart

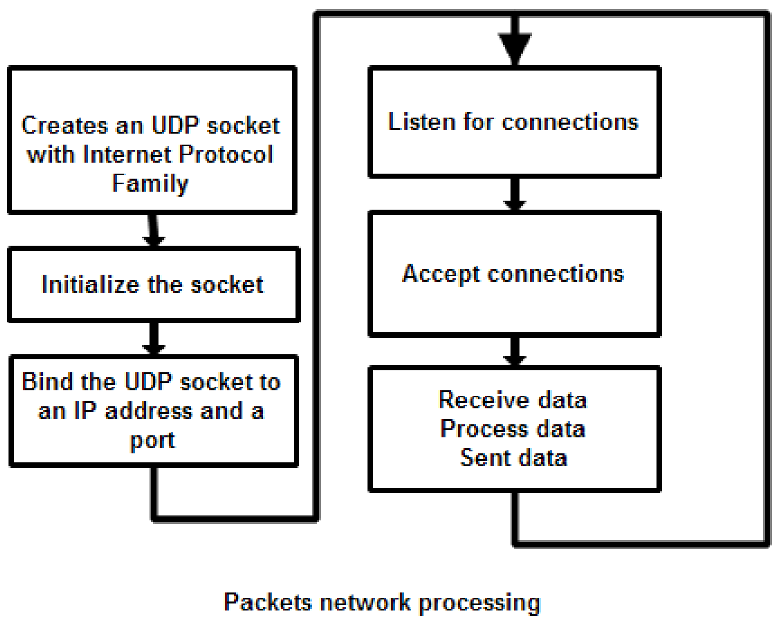

Creates an UDP socket with Internet Protocol Family
Initialize the socket
Bind the UDP socket to an IP address and a port
Listen for connections
Accept connections
Receive data
Process data
Sent data
Packets network processing

Fig. 13. Network processing flowchart

The optimized execution time can be seen in Table 1. In this table one can observe that the processing time is less than frame acquisition time of 20 milliseconds and therefore the VoIP system works in real-time.

MODE	AMR Codec--Execution time (Dual core Blackfin Processors)			
	CORE A		CORE B	
	Cycles	Milliseconds	Cycles	Milliseconds
MR475 4.75 Kbit/s	13593292	18.12	1783754	2.38
MR515 5.15 Kbit/s	10673898	14.23	1756316	2.34
MR59 5.90 Kbit/s	12141920	16.19	1743907	2.33
MR67 6.70 Kbit/s	14376904	19.17	1745566	2.33
MR74 7.40 Kbit/s	13584752	18.11	1752102	2.34
MR795 7.95 Kbit/s	14025496	18.70	1830429	2.44
MR102 10.20 Kbit/s	14200042	18.93	1813014	2.42
MR122 12.20 Kbit/s	14685916	19.58	1887733	2.52

Table 1. Execution time for AMR codec

The proposed strategy may be used as a framework for various applications, especially for applications in sensor networks, that requires fast computation, lower power consumption and network (fixed or mobile) connectivity.

4. LWIP performance evaluation

This section presents the framework in which performance evaluation was performed. Two test programs were developed: a client program and a server program. The client connect to the server and send continually data requests, while the server listen for connections, accept it and send a replay message to the clients. More than ten client instances were started to evaluate the performance of the LWIP connection with high load.

The client and server flowcharts are illustrated in Figures 14 and 15, respectively.

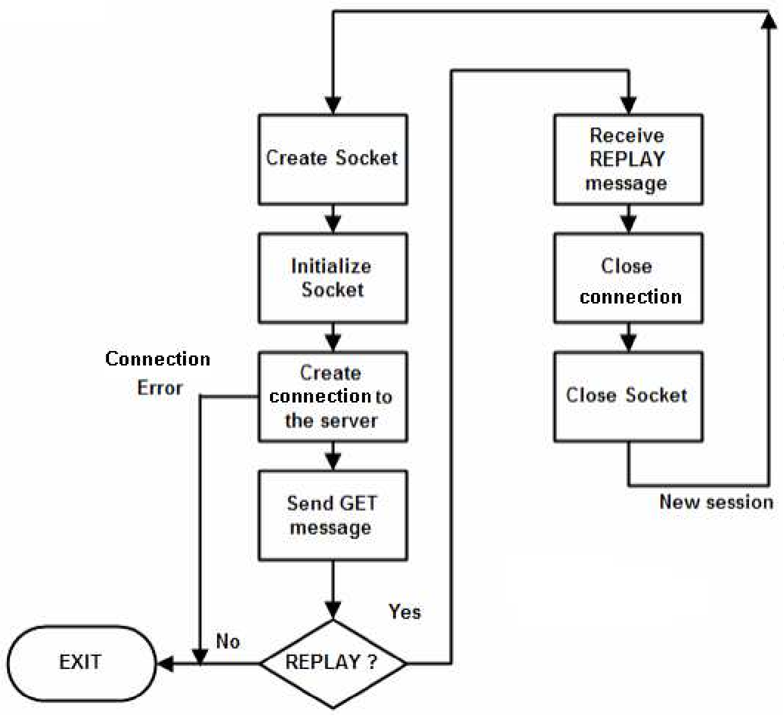


Fig. 14. Client flowchart

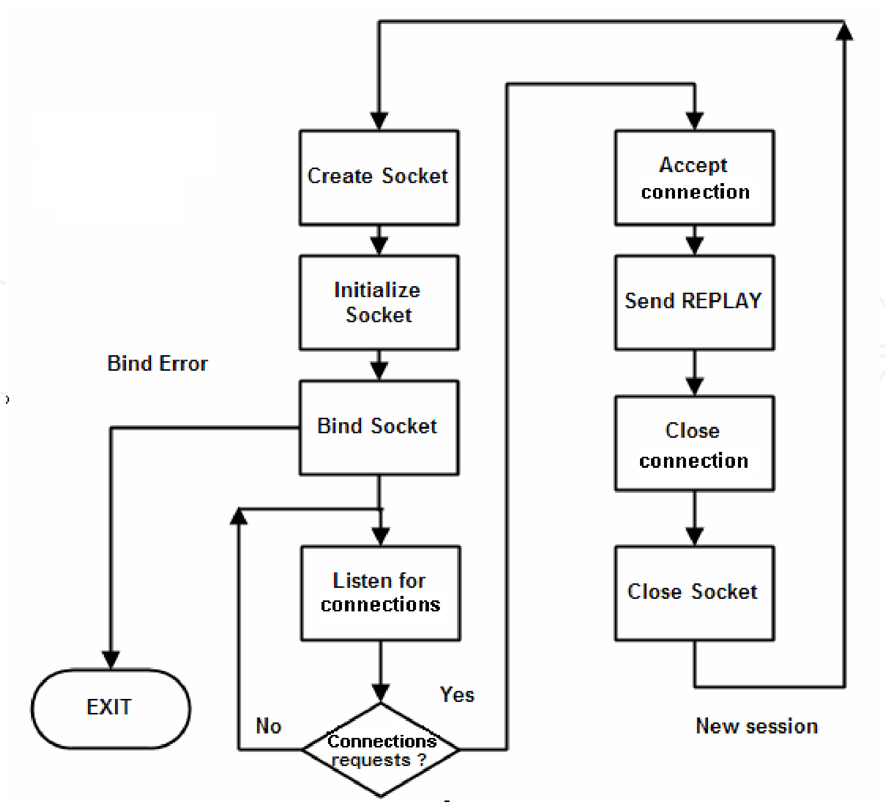


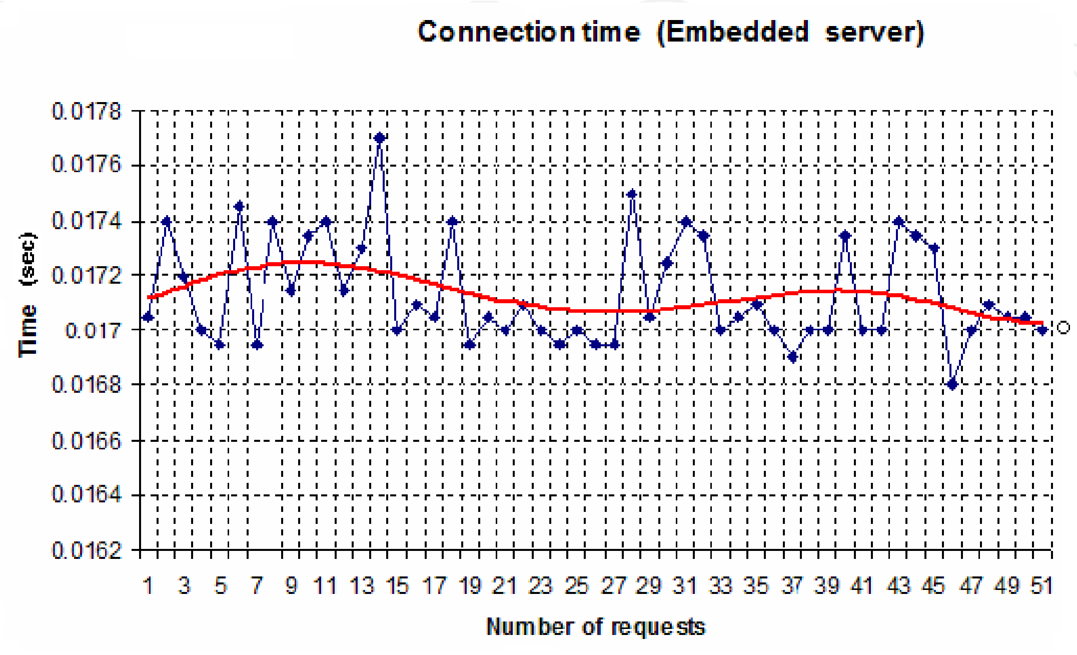
Fig. 15. Server flowchart

The clients were run on a personal computer (PC) with Windows operating system and they were written in C language using Visual 2008 Studio. The WinSock2.0 library was used for manage the network connections, sending and receiving the packets over network. The server has two versions: one developed under Windows operating system (as the clients) and the second developed using the LWIP stack protocol under the VDK operating system kernel, presented in the section 3. The embedded server was written using the LWIP API functions (Analog Devices 2, 2010) and benefits of VDK support as it specify above.

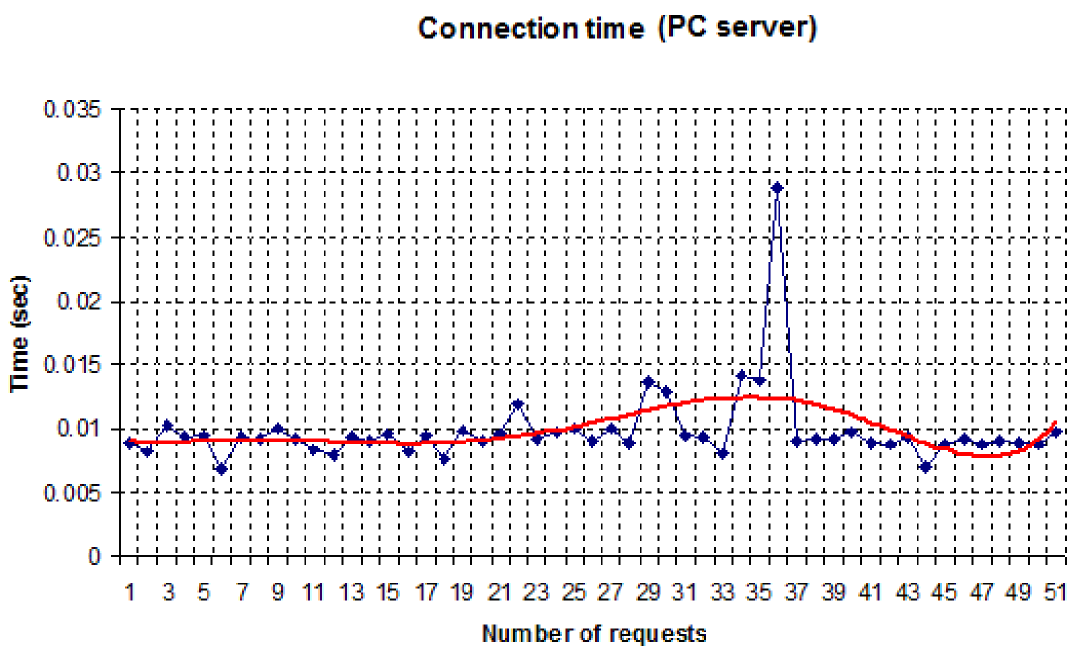
The clients and server programs were run, both on PC and Blackfin BF537 evaluation board and the packets transferred over the network was captured using the network analyzer Wireshark. The following sequence that shows the flow of a TCP connection was used to measure the connection time and the response time between clients and server:

1. The server creates the listener socket waiting for remote clients to connect.
2. The client call *connect ()* socket function to start the TCP handshake (SYN, SYN/ACK, ACK).
3. The server call *accept ()* socket function to accept the connection request.
4. The client and server issue the *read ()* and *write ()* socket functions to exchange data over the socket.
5. Either the server or the client decides to close the socket that causes the occurrence of the sequences FIN and ACK.
6. The server either closes the listener socket or repeats beginning with step 2 to accept another connection from a remote client.

The packets, transferred between clients and server, were analyzed for SYN, SYN/ACK ACK and FIN flags. The time between TCP flags SYN and SYN/ACK was measured to determine the connection time of the clients to the server. The time between the GET and REPLAY messages was measured to determine the response time. The results, both for embedded server and PC server, are illustrated in Figures 16 and 17.

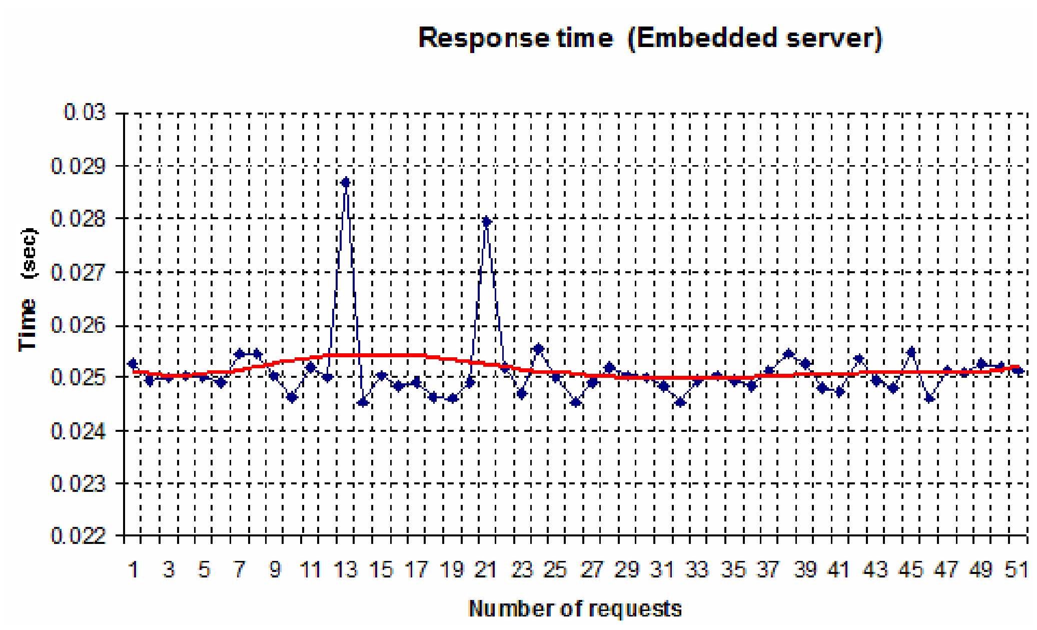


a)

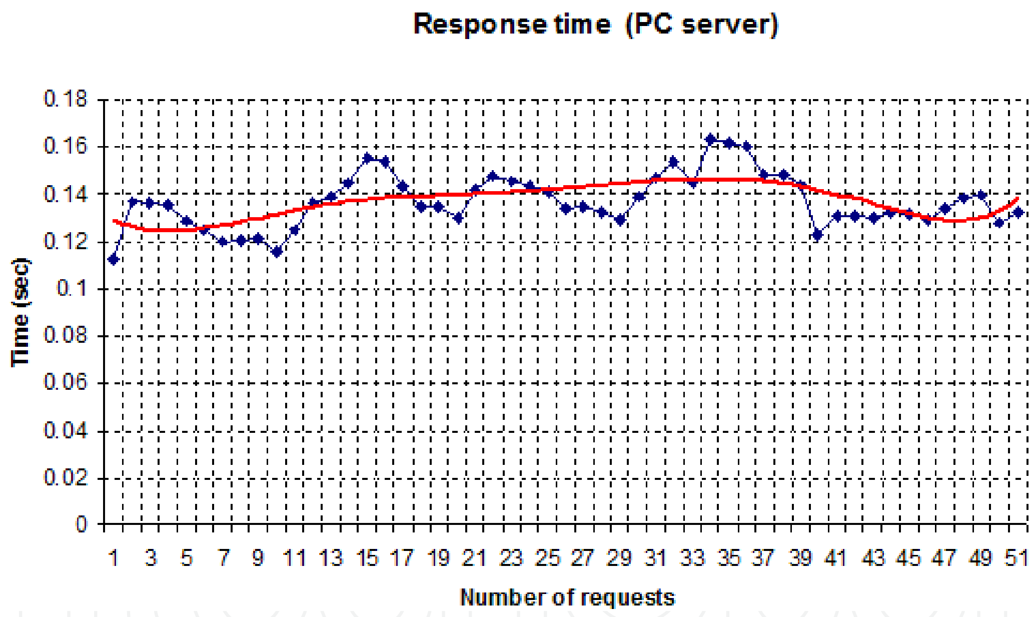


b)

Fig. 16. The connection time: a) Embedded server; b) PC server (red line is a trend line)



a)



b)

Fig. 17. The response time: a) Embedded server; b) PC server (red line is a trend line)

Accordingly with these figures, the embedded server has a slower connection time. That is happened due the LWIP has a limited memory buffers and involves a slow memory management mechanism. The clients and the PC server were run on the same PCs (Windows XP 32 bits, 100Mbps network interface). In both situations, up to ten clients were run without connection error.

Comparing the connection and the response time, one can observe that the differences between the TCP and LWIP implementation are not critical, for absolute values. The values obtained with LWIP implementation are very good; the connection time is similar for

Blackfin implementation and PC implementation. The response time depends of the traffic load on the PC, but it is good for both implementations.

Due the limited memory and the memory management used in LWIP implementation, the connection time and response time are slower than in personal computers, but they remain still acceptable.

Several system threads are created to interface the network card with the Blackfin core using the SSL library. These threads were created using the VDK primitives. This approach allows the clients to run concurrently and therefore the connection time and response time will be decreased. The processor load is about 30% (considering up to ten clients that require connections). Figure 18 illustrates the system threads and processor load.

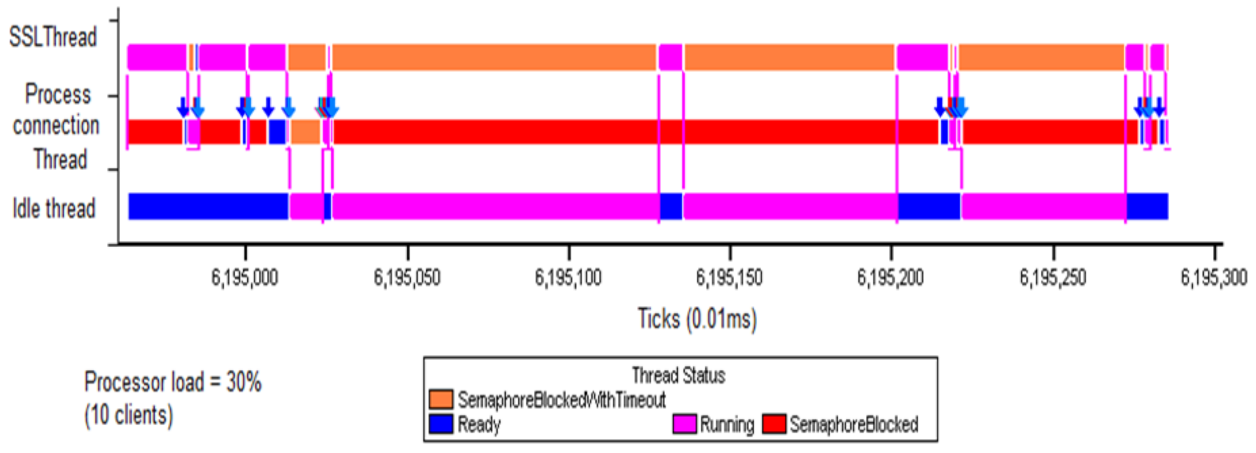


Fig. 18. Threads and processor load

5. Conclusion

The chapter presents a strategy for networking real-time applications development--based on Blackfin microcomputers family, VDK operating kernel and LWIP stack protocol--and evaluates the performance of the lightweight TCP/IP protocol stack for embedded systems. Its applications reside in sensor networks in which the sensors may be connected directly to the Internet. This strategy may be used for various complex applications such digital signal processing in sensor network. The overall performance is similar to the performance of the implementation of the TCP/IP protocol stack in PCs. The connection time and response time are slower in the LWIP implementation, comparing with typical TCP/IP implementation, but they are acceptable. A real-time implementation of network applications, such as voice--over IP system, is exemplified, using a dual core microcomputer and a practical approach to achieve a real-time functioning is provided. Future work will investigate the real-time functionality of presented strategy in multimedia applications.

6. Acknowledgement

This work has benefited the support of FP7 project ALICANTE—FP7—ICT—2009—4 no. 248652.

7. References

Deborah Estrin (2001). Embedded Everywhere: A Research Agenda for Networked Systems of Embedded Computers, National Academy Press, ISBN 0-309-07568-8

Redwan Salami et al. (2002). The Adaptive Multi-Rate Codec: History and Performance, IEEE Speech Coding Workshop, Tsukuba, Japan, pp. 144–146

Johan Sjöberg et al. (2002). Real-Time Transport Protocol (RTP) Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs, IETF RFC 3267

Gregory Pottie and William Kaiser (2005). Principles of Embedded Networked Systems Design, Cambridge University Press, ISBN 9780521840125

Adam Dunkels (2005). Towards TCP/IP for Wireless Sensor Networks, Ed. Arkitektkopia, Vasteras, Sweden, ISBN 91-88834-96-4

Analog Devices (2006). ADSP-BF533: Blackfin Embedded Processor DataSheet, Rev. C., www.analog.com

Analog Devices 1 (2007). VisualDSP++ 5.0 Device Drivers and System Services Manual for Blackfin Processors, www.analog.com

Analog Devices 2 (2007). VisualDSP 5.0 Kernel (VDK) Users Guide, www.analog.com

Adam Dunkels (2007). Programming Memory-Constrained Networked Embedded Systems, SICS Dissertation Series 47, Ed. Arkitektkopia, Vasteras, Sweden, ISSN 1101-1335

Woon-Seng Gan. and Sen M. Kuo, (2007). Embedded Signal Processing with the Micro Signal Architecture, Wiley-Interscience, ISBN 978-0471738411

Sorin Zoican (2008). The Role of Programmable Digital Signal Processors (DSP) for 3G Mobile Communication Systems, Acta Tehnica Napocensis, Electronic and Telecommunications, vol. 49, no. 3, 2008, pp.49-56

Analog Devices 1 (2010). VisualDSP++ 5.0 C/C++ Compiler and Library Manual for Blackfin Processors, www.analog.com

Analog Devices 2 (2010). LWIP user guide, www.analog.com

Sorin Zoican (2011). The Adaptive Multirate Speech Codec: Deployment Strategy Using the Blackfin Microcomputer", Sped 2011 - The proceedings of 6th Conference on Speech Technology and Human - Computer Dialogue, Brasov, Romania, pp. 81-84.

Additional readings

John Proakis, Charles Rader, and Fuyun Ling (1992). Advanced Topics in Digital Signal Processing, Prentice Hall, ISBN: 0-02-396841-9

Arnold Berger (2001), Embedded Systems Design: An Introduction to Processes, Tools and Techniques, CPM Books, ISBN 1-800-788-3123

John Catsoulis and O'Reilly (2005). Designing Embedded Hardware, O'Reilly Media, ISBN 0-596-00755-8

Douglas Comer (1993). Internetworking with TCP/IP - Principles, Protocols and Architecture, Prentice Hall, 1993, ISBN 86-7991-142-9

REAL-TIME SYSTEMS, ARCHITECTURE, SCHEDULING, AND APPLICATION

Edited by Seyed Morteza Babamir

INTECH

Real-Time Systems, Architecture, Scheduling, and Application
Edited by Dr. Seyed Morteza Babamir

ISBN 978-953-51-0510-7
Hard cover, 334 pages
Publisher InTech
Published online 11, April, 2012
Published in print edition April, 2012

This book is a rich text for introducing diverse aspects of real-time systems including architecture, specification and verification, scheduling and real world applications. It is useful for advanced graduate students and researchers in a wide range of disciplines impacted by embedded computing and software. Since the book covers the most recent advances in real-time systems and communications networks, it serves as a vehicle for technology transition within the real-time systems community of systems architects, designers, technologists, and system analysts. Real-time applications are used in daily operations, such as engine and break mechanisms in cars, traffic light and air-traffic control and heart beat and blood pressure monitoring. This book includes 15 chapters arranged in 4 sections, Architecture (chapters 1-4), Specification and Verification (chapters 5-6), Scheduling (chapters 7-9) and Real word applications (chapters 10-15).

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Sorin Zoican (2012). Networking Applications for Embedded Systems, Real-Time Systems, Architecture, Scheduling, and Application, Dr. Seyed Morteza Babamir (Ed.), ISBN: 978-953-51-0510-7, InTech, Available from: http://www.intechopen.com/books/real-time-systems-architecture-scheduling-and-application/networking-applications-for-embedded-systems-

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

© 2012 The Author(s). Licensee IntechOpen. This is an open access article distributed under the terms of the Creative Commons Attribution 3.0 License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.