AN2282
APPLICATION NOTE

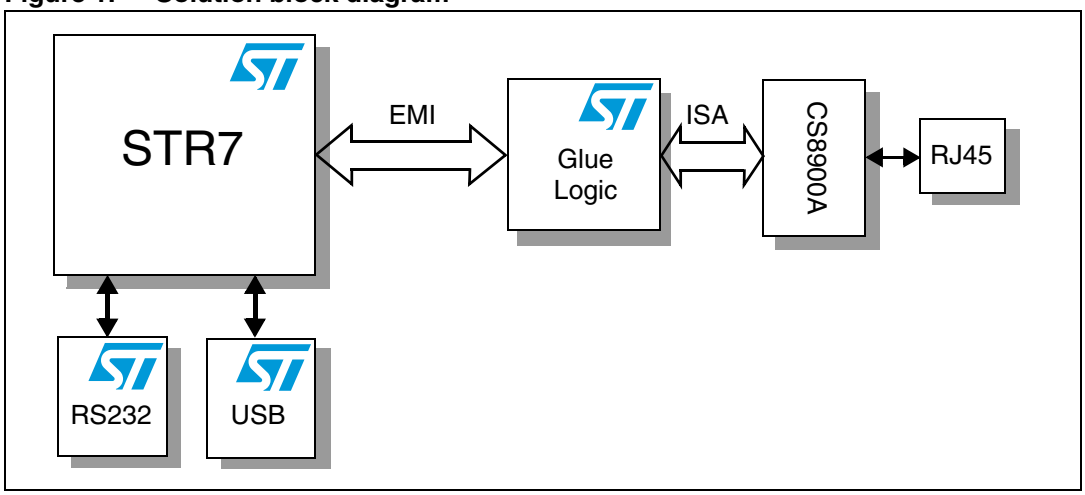
TCP/IP Over Ethernet Connectivity with the STR710

Introduction

With the ever-increasing success of the Internet, many applications with networking features are being developed making the TCP/IP protocol a global standard for communication both for the Internet and local area networks. In a normal scenario, a microcontroller inside a home appliance or process controller of an industrial distributed control system is not directly connected to the Internet but instead is a host of a home or industrial local area network using an Ethernet protocol.

This application note describes a method to provide TCP/IP over Ethernet connectivity to an STR710-based solution and, as an example, an application layer has been included in the solution firmware. The application is a server that waits for a Client request. The successful or unsuccessful processing notification is sent back to the Client. The solution block diagram is showed in *Figure 1*. The MAC layer is implemented using the third part CIRRUS LOGIC® Crystal LANTM CS8900A 10-baseT Ethernet controller because of its ISA bus interface. The STR7 EMI interface can be interfaced with an ISA bus by simple Glue Logic.

Figure 1. Solution block diagram



Contents

1 Hardware

The target microcontroller for the solution is the STR710 with TQFP144 package to have the EMI bus available for Ethernet controller interfacing. The hardware includes also a RS232 interface for SLIP implementation.

1.1 The STR710 microcontroller

The STR710 is a 16-/32-bit microcontroller based on the ARM7TDMI RISC microprocessor. It combines the high performance of the ARM® CPU with an extensive range of peripheral functions and enhanced I/O capabilities. The microcontroller has on-chip high-speed single voltage FLASH memory and high-speed RAM, since both memories are mounted on an ARM® CPU native bus the STR710 maximizes the CPU calculation speed. The STR710 has an embedded ARM® core and is therefore compatible with all ARM tools and software. The 144-pin version (used for the solution in the present AN) has a non-multiplexed 16-bit data/24-bit address bus available that supports four 16-Mbyte banks of external memory (EMI). Wait states are programmable individually for each bank allowing different memory types (Flash, EPROM, ROM, SRAM etc.) to be used to store programs or data. Later on, the way of interfacing the EMI with an ISA Ethernet controller will be shown.

The STR710 main features are as follows [1]:

- Memories:
  - Up to 272 KB (256+16) FLASH program memory
  - Up to 64 KB RAM
  - EMI for up to 4 banks with 16MB of addressing space each (64MB total)
  - Multi-boot capability
- Clock, Reset and Supply Management:
  - 3.3V supply and I/O interface
  - Embedded 1.8V voltage regulator for core supply
  - Up to 50 MHz CPU operating frequency with external clock source and internal PLL
  - Real-time clock for clock-calendar function
  - 4 power saving modes: SLOW, WAIT, STOP and STANDBY
- Nested Interrupt Controller:
  - Fast interrupt handling with multiple vectors
  - 32 vectors with 16 IRQ priority levels
  - 2 maskable FIQ sources
- I/O Ports:
  - Up to 48 I/O ports
- 5 Timers:
  - 16-bit watchdog timer
  - Four 16-bit timers each with: 2 input captures, 2 output compares, PWM and pulses counter mode
- 10 Communication Interfaces:
  - 2 I2C interfaces (1 multiplexed with SPI)
  - 4 UART
  - Smart card ISO7816-3 interface on UART1
  - 2 BSPI
  - CAN interface (2.0B Active)
  - Full speed USB (12Mbits/sec)
  - HDLC interface
- 4 Channel 12-bit A/D Converter:
  - Sampling frequency up to 1 KHz
  - Conversion range: 0 to 2.5V
- Development Tools Support:
  - JTAG with debug mode trigger request

1.2 EMI-ISA Interface

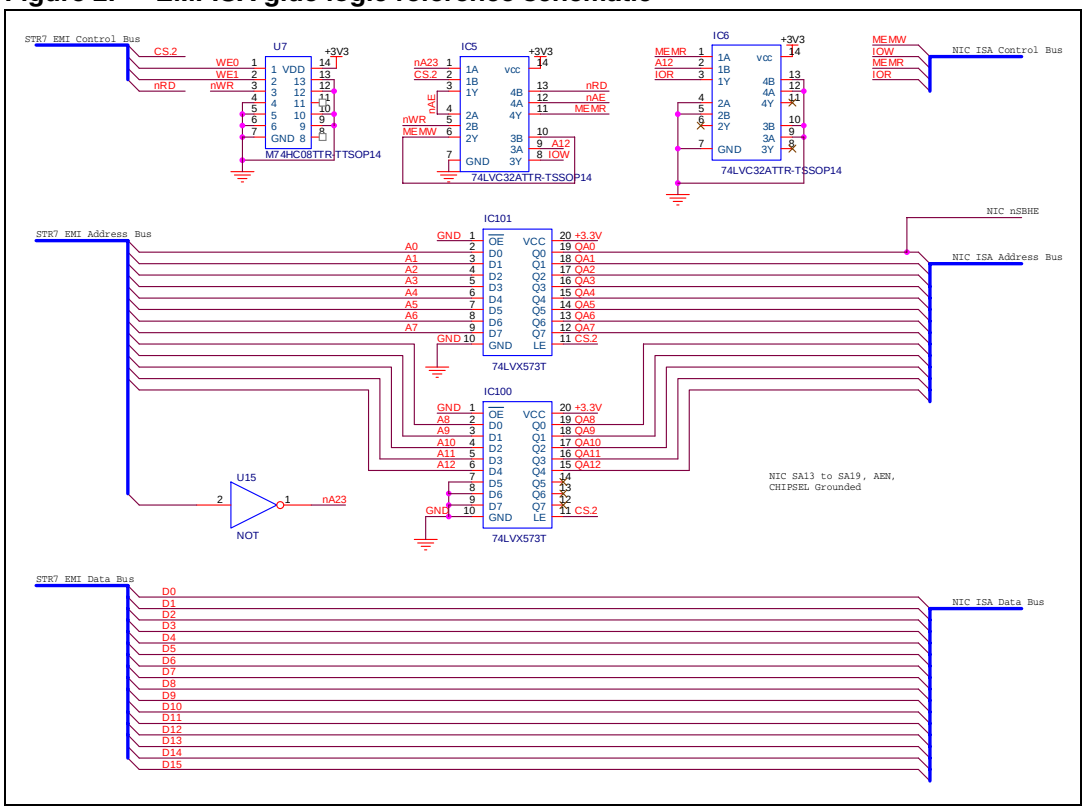
The CS8900A Ethernet controller includes a direct interface to the ISA bus. The ISA bus (Industry Standard Architecture) has been the first standard in the IBM PC architecture; detailed information and specification can be found in the following documents:

- EISA Specification, Version 3.12. This document includes specifications for ISA as well as the "Extended Industry Standard Architecture" that defines a 32-bit extension to the ISA [2].
- PS/2 Technical Reference - AT Bus Systems. This document includes signal definitions and timing diagrams for the ISA bus used in some IBM computers [3].

At the same time, the CS8900A can be used for cost-effective, full-duplex Ethernet solutions for non-ISA architectures as referenced in the application note AN83 from CIRRUS LOGIC® Crystal LANTM CS8900A ETHERNET CONTROLLER TECHNICAL REFERENCE MANUAL [4].

The reference schematic of the hardware interface between EMI and the ISA bus is shown in *Figure 2*. The Chip Select 2 of EMI is used to address the Ethernet Controller; moreover address line A23 of EMI is used together with CS.2 to enable read/write operations on CS8900A. From a firmware point of view, this means that the Ethernet Controller is mapped at base address 0x64800000; in this way the CS.2 is not committed only by the CS8900A but it also can be used to address other devices.

Figure 2. EMI-ISA glue logic reference schematic



The CS8900A can be accessed both in ISA IO mode and ISA memory mode; the glue logic is designed to allow both access modes. The Address line A12 separates IO address space and memory address space. When A12 is low the Ethernet Controller is accessed in IO

mode and when A12 is high it is accessed in memory mode, so memory mode base address register of Ethernet controller must be programmed with a value 0x1000. The IO mode default value of CS8900A (0x0300) is used as IO mode base address. The corresponding firmware addresses are 0x64800300 for IO mode operations and 0x64801000 for memory mode operations. For a detailed description of IO and memory access modes refer to the CS8900A datasheet [5].

ISA Bus timing constraints must be satisfied to guarantee correct access to Ethernet controller registers; Figures *3* and *4* show the CS8900A bus timing diagrams. Configuring the MCLK of STR7 with the value 48 MHz, the EMI CS.2 must be configured with at least 7 wait states; Figures *5* and *6* show the STR7 EMI bus timing diagrams. 8 wait states are used to guarantee a safe access.

Figure 3. CS8900A read timing diagram

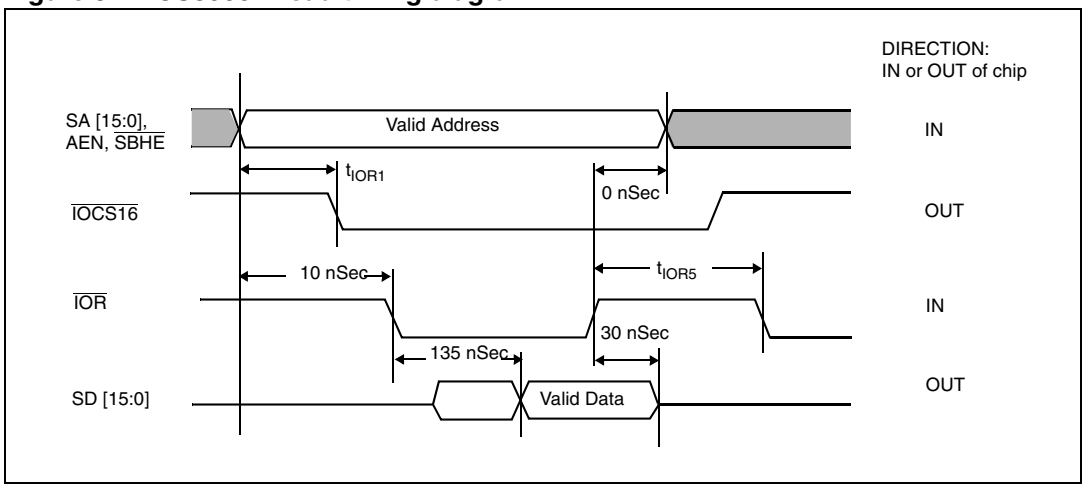


Figure 4. CS8900A write timing diagram

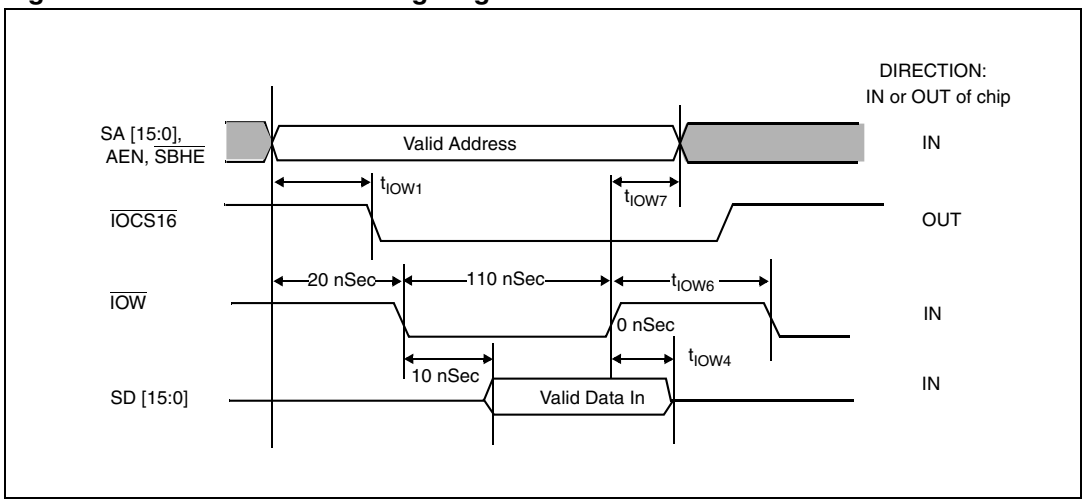


Figure 5. STR7 EMI read timing diagram

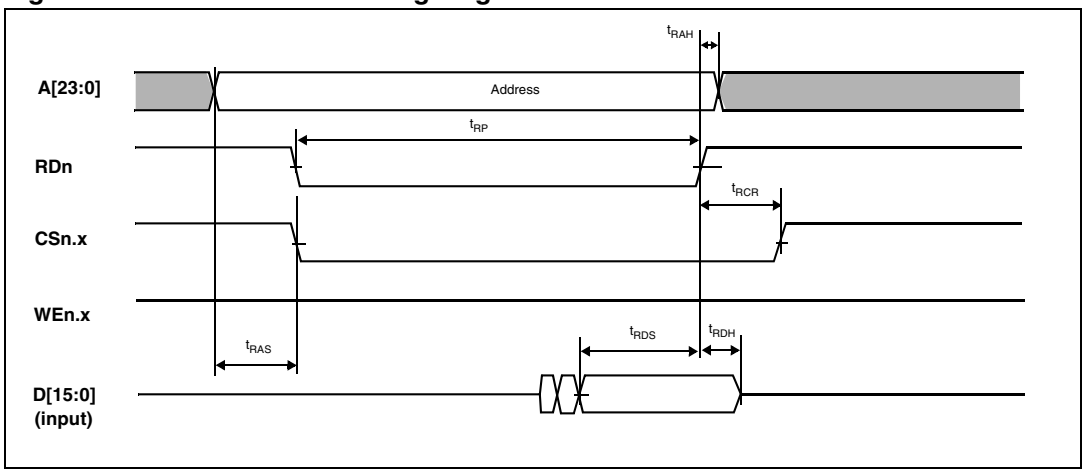
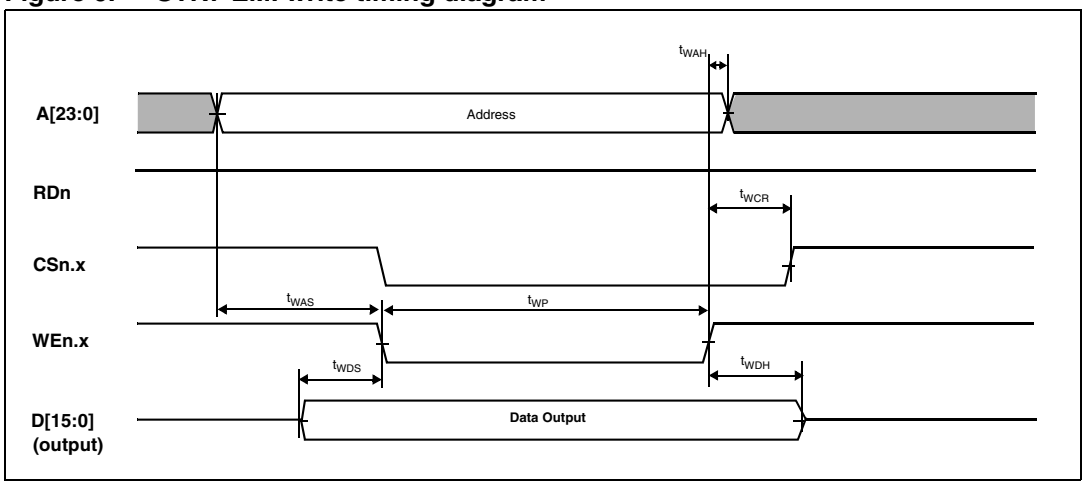


Figure 6. STR7 EMI write timing diagram



CS8900A timing diagrams for memory mode access are the same so they are not included in this document.

Considering the MCLK value is equal to 48 MHz and 7 wait states, the cycle times are as follows:

a) $t_{RAS} = t_{WAS} = 27$ nSec.
b) $t_{RP} = t_{WP} = 166$ nSec.
c) $t_{RDS} = t_{WDS} = 3$ nSec. (fixed)
d) $t_{RDH} = t_{WDH} = 3$ nSec. (fixed)
e) $t_{RAH} = t_{WAH} = 3$ nSec. (fixed)
f) $t_{RCR} = t_{WCR}$ 20 nSec.

All cycle times are compatible with Ethernet controller timing. Of course it also possible to use more than 7 wait states.

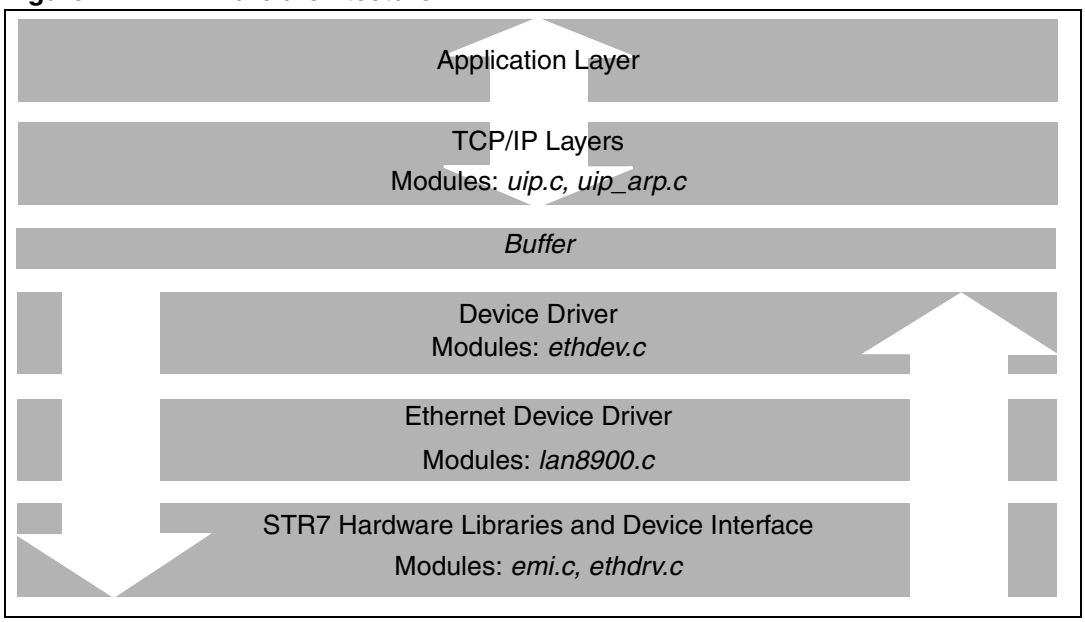
Considering the data bus connection of *Figure 2*, the STR7 reads incoming data in Little Endian byte order. For example assume that the data received from the Ethernet wire is 0x01, 0x02, 0x03, 0x04, etc. where 0x01 is the first byte, 0x02 the second and so on; STR7 reads the following data words: 0x0201, 0x0403, etc.

The hardware also includes one connection of a STR7 GPIO to the reset pin of the Ethernet Controller and a second connection from an interrupt pin of Ethernet controller to an external interrupt pin of the STR7.

2 Firmware

The firmware is developed in several layers following the standard ISO/OSI model for networking from physical layer up to application layer. A firmware architecture diagram is shown below in *Figure 7*.

Figure 7. Firmware architecture



The firmware also includes a module for the configuration of the GPIO used to manage the reset signal and interrupt signal of the Ethernet Controller.

2.1 Ethernet controller driver

The two lower layers (the *Ethernet Device Driver* layer and the *STR7 Hardware Libraries and Device Interface* layer) manage the CS8900A Ethernet controller and the physical access to the device.

2.1.1 Low Level Interface

The *emi.c* is part of the STR710 library and it defines the functions for configuration of the EMI peripheral. The *ethdrv.c* module provides the interface function to the upper layer and is useful for low level access to the Ethernet controller when it is accessed in I/O mode:

- `void outw(int address, short data);` Writes a 16-bit data word to an I/O register located at a given address.
- `char inb(int address);` Reads a byte data from an I/O register located at a given address.
- `short inw(int address);` Reads a 16-bit word from an I/O register located at a given address.

The module also provides the function for hardware reset of the Ethernet controller:

- `void reset_NIC(void);`

The inb function is used only to perform the switch of Ethernet controller from an 8-bit to a 16-bit data interface because, even if after a reset the CS8900A starts in 8-bit mode, it is recommended to be used in 16-bit mode. This switching operation is done by driving a transition on nSBHE pin of CS8900A, so connecting this pin to address line SA0 (see *Figure 2*), the transition is simply done by performing some dummy byte read operations.

2.1.2 Initialization and configuration

The *lan8900.c* module provides all the functions for CS8900A initialization, configuration and packet transmission/reception management. The Ethernet controller can be initialized either by using an external EEPROM (where all configuration words are automatically downloaded into the CS8900A internal registers at reset time) or by filling them through software. In this application note the Ethernet controller is initialized and configured by the function

- `unsigned long csInitFromBSPDefs(void);`

This function reads the configuration constant from the module *bsp.h*. The header file for this is mainly defined as follows:

a) The base address for I/O access mode:
```
#define BSP_CS8900_IO_BASE      0x300
```

b) The interrupt request number:
```
#define BSP_CS8900_IRQ          10
```
This number defines which interrupt line of the Ethernet Controller must be connected to the external interrupt line of the STR7 [5].

c) The access type (I/O or memory mode):
```
#define BSP_CS8900_MEM_MODE     YES
```

d) The base address for memory access mode:
```
#define BSP_CS8900_MEM_BASE     0x01000
```

e) The media type:
```
#define BSP_CS8900_MEDIA_TYPE   TEN_BASE_T
```

f) The individual Ethernet address:
```
#define BSP_CS8900_IND_ADDR     "00:24:20:10:FF:41"
```

The header file *lan8900.h* defines all CS8900A internal registers, for more details refer to the CS8900A datasheet. All Initialization functions (csInitFromBSPDefs included) are called by the function:

- `unsigned long csInitialize( void (*ap_addr)(mblk_t *, IA, IA) );`

The function verifies the Ethernet controller ID and resets it by calling the following functions:

- `unsigned long csVerifyChip(void);`
- `unsigned long csResetChip(void);`

Moreover, this function sets the individual and broadcast Ethernet addresses, the function for new packet notification from the passed argument and initializes the interrupt number calling the function `unsigned long csInitInterrupt(void);`

2.1.3 Ethernet packet management

The packets to transmit or to receive are exchanged with the upper layer as messages with the following structure:

```
typedef struct{
   unsigned char buff[1600];
   unsigned short dim;
   unsigned type;
} mblk_t;
```

The messages to be transmitted are managed by two linked lists of requests inside a static array of transmission requests. It is possible to manage up to 16 outstanding requests. The request structure is as follows:

```
struct txreq
{
   struct txreq  *pNext;
   PIA            pDestAddr;
   mblk_t        *pMsg;
   unsigned short Length;
   unsigned short Type;
};
typedef struct txreq TXREQ, *PTXREQ;
```

The first list is composed of the outstanding transmission requests, the second one of the unused requests. The functions for message management are:

- `unsigned long csSend( PIA pDestAddr, mblk_t *pSendParms );`
  This routine is called when a new message has to be transmitted. The Destination address and the message are passed by the arguments. The new transmit request is placed on the transmit request queue. If a transmit request is not currently in progress then the new transmit request is started, else it waits its turn in queue.
- `unsigned short csRequestTransmit(void);`
  This routine requests that a transmit be started by writing a transmit command and a frame length to the chip. The content of the BusStatus register is returned which indicates the success of the request.
- `void csCopyTxFrame(void);`
  This routine builds an Ethernet header in the chip's transmit frame buffer and then copies the frame data from the list of message blocks to the chip's transmit frame buffer. When all the data has been copied then the chip automatically starts to transmit the frame.
- `void csProcessISQ(void);`
  This routine processes the events on the Interrupt Status Queue. The events are read one at a time from the ISQ and the appropriate event handlers are called. The ISQ is read until it is empty. If the chip's interrupt request line is active, then reading a zero from the ISQ will deactivate the interrupt request line.
- `void csReceiveEvent( unsigned short RxEvent );`
  This routine is called whenever a frame has been received at the chip. This routine copies the received frame into the data buffer of the reception message and passes the message to the upper layer by calling `Announce_Packet()` entry point of the upper layer. Moreover, this routine verifies that the frame type is acceptable.
- `void csTransmitEvent(void);`
  This routine is called whenever the transmission of a frame has completed (either successfully or unsuccessfully). This routine removes the completed transmit request

from the transmit request queue. If there are more transmit requests waiting, then start the transmission of the next transmit request.

- `unsigned short csReadPacketPage( unsigned short Offset );`
  This reads the PacketPage register at the specified offset.
- `void csWritePacketPage( unsigned short Offset, unsigned short Value );`
  This writes to the PacketPage register at the specified offset.
- `unsigned long csEnqueueTxReq( PIA pDestAddr, mblk_t *pSendParms );`
  This routine places a transmit request at the end of the Transmit Request queue.
- `void csDequeueTxReq(void);`
  This routine removes a transmit request from the head of the Transmit Request queue.

Summarizing, the *lan8900.c* module moves the received packet from the Ethernet controller memory to the reception message and the packet to transmit from the transmission message to the Ethernet controller memory.

2.2 Device driver

The module *ethdev.c* is the interface between the CS8900A driver and the TCP/IP stack. The module also provides the interface to the main routine for Ethernet controller initialization. The TCP/IP stack is interfaced to the lower layer by a single buffer both for reception and transmission, so this module has to manage data movement between lower layer messages and this single buffer. The functions of the module are the following:

- `int ethdev_init(void);`
  The interface for device initialization.
- `unsigned int ethdev_read(void);`
  The interface for processing the events on the Interrupt Status Queue.
- `void ethdev_send(void);`
  The interface to send packets. It copies the TCP/IP stack buffer data into the transmission message data.
- `void ProcessMsg( mblk_t *pMsg );`
  This routine process the incoming message. It copies the received message data into TCP/IP stack buffer and returns the length of the received packet avoiding collision between incoming and outgoing data. Its Entry Point is passed to the lower layer to indicate which function must process the incoming messages (used to initialize cs_pAnnounce_Packet variable).

2.3 TCP/IP layers

The TCP/IP stack (*uip.c* and *uip_arp.c* modules) used in the solution is the uIP open source stack from www.sics.se. It provides the necessary protocols for Internet communication, with a very small code footprint and RAM requirements. The uIP implementation is designed to have only the absolute minimal set of features needed for a full TCP/IP stack. It can only handle a single network interface and contains only a rudimentary UDP implementation, but focuses on the IP, ICMP and TCP protocols. The full TCP/IP suite consists of numerous protocols, ranging from low level protocol such as ARP which translates IP addresses to MAC addresses, to application level protocols. The uIP is mostly concerned with the TCP and IP protocols and upper layer protocols will be referred to as "the application" [6].

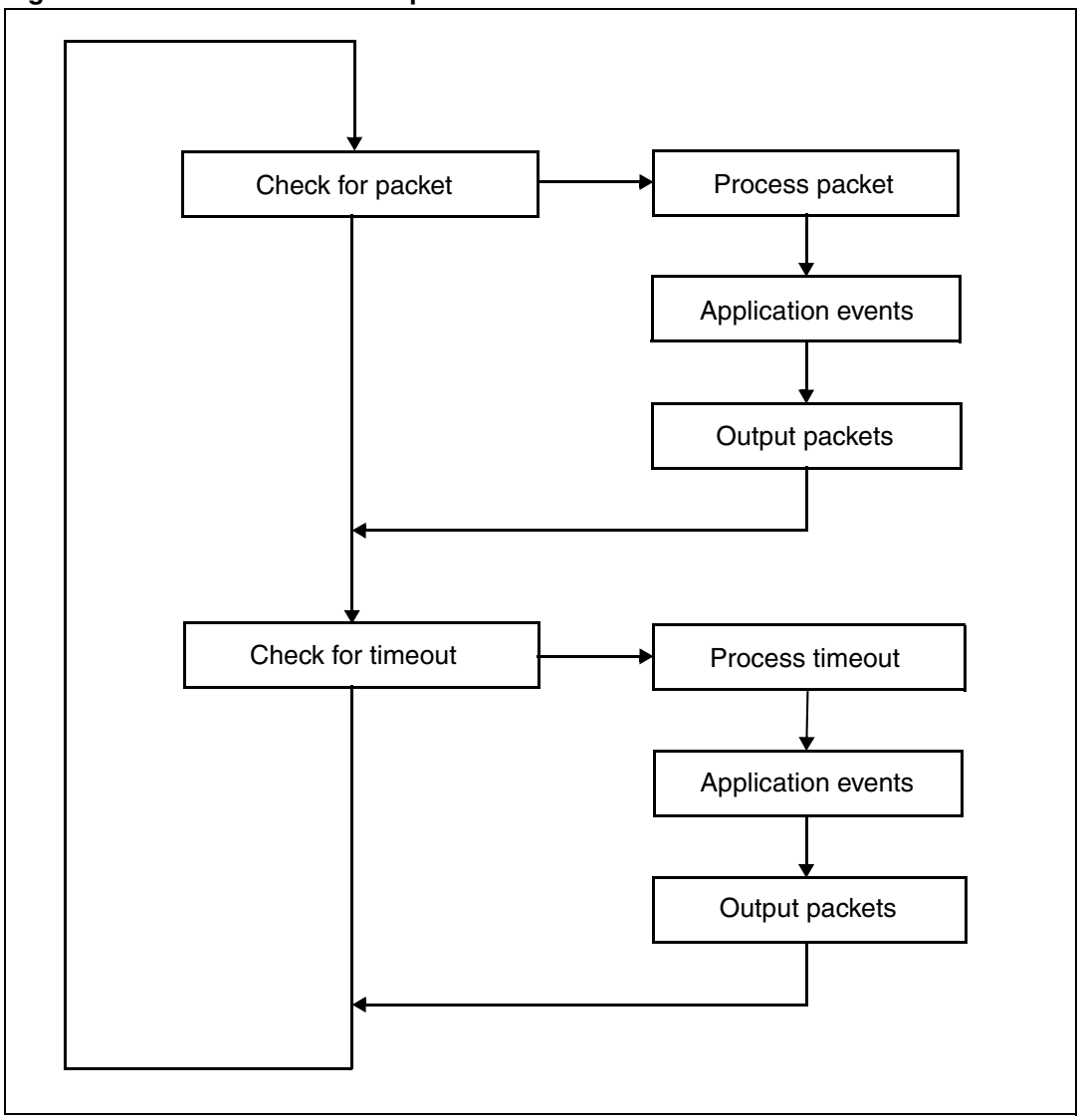
The uIP main features are as follows:

- Very small code size
- Very low RAM usage, configurable at compile time
- ARP, SLIP, IP, UDP, ICMP (ping) and TCP protocols
- Includes a set of example applications: web server, web client, e-mail sender (SMTP client), Telnet server, DNS hostname resolver
- Any number of concurrently active TCP connections, maximum amount configurable at compile time
- Any number of passively listening (server) TCP connections, maximum amount configurable at compile time
- Free for both commercial and non-commercial use
- RFC compliant TCP and IP protocol implementations, including flow control, fragment reassembly and retransmission time-out estimation

The total memory requirement (ROM+RAM) is less then 21.3 KB. The uIP stack does not use explicit dynamic memory allocation. Instead, it uses a single global buffer (uip_buf) for holding packets and has a fixed table for holding connection states. The global packet buffer is large enough to contain one packet of maximum size. When a packet is received, the device driver places it in the global buffer, updates the global variable uip_len with the value of the received packet dimension and calls the TCP/IP stack. If the packet contains data, the TCP/IP stack will notify the corresponding application. In uIP, the same global packet buffer is also used for the TCP/IP headers of outgoing data. If the application sends dynamic data, it may use the parts of the global packet buffer that are not used for headers as a temporary storage buffer.

To send the data, the application passes a pointer to the data as well as the length of the data to the stack. The data is not queued for retransmissions; instead, the application has to reproduce the data if a retransmission is necessary. uIP uses an event driven interface where the application is invoked in response to certain events. The application must be implemented as a C function, `UIP_APPCALL()`, that uIP calls whenever an event occurs. uIP calls the application when data is received, when data has been successfully delivered to the other end of the connection, when a new connection has been set up, or when data has to be retransmitted. The application is also periodically polled for new data. The application provides only one callback function; it is up to the application to deal with mapping different network services to different ports and connections. *Figure 8* shows the uIP main control loop flow diagram.

Figure 8. uIP main control loop



For more details on TCP/IP stack application program interfaces refer to the uIP Reference Manual and to the available documentation and information from www.sics.se [6].

The stack is processed in the main loop in the *main.c* module; the execution is temporized by using the alarm of the RTC. The stack execution time is one second by hooking the TCP/IP processing function `void TCP_IP_Stack( void )` to the RTC alarm interrupt handler. After initialization the software waits five seconds before starting the cyclic processing. Using this approach, the TCP/IP stack is processed independently of the rest of the whole firmware, so the stack firmware can be considered as a module of a bigger application.

2.4 Application layer

The uIP uses an event-driven interface where the application is invoked in response to certain events. An application running on top of the uIP stack is implemented as a C function that is called in response of the following connection events:

a) Aborted
b) Timeout
c) Closed
d) Connected
e) New data reception
f) Transmitted data acked
g) Polling
h) Re-transmission need

The re-transmission event is necessary because the TCP/IP layers require help from the application layer in a different way of other TCP/IP stacks. With this approach, the data has to be buffered by the application.

The application is implemented by two modules *<user_app>.c* and *<user_app>.h* . In the header file the user has to define the UIP_APPCALL() macro, an optional application state and the FS_STATISTICS macro to enable or disable the statistic related to the packet's elaboration. In the C file the user has to put the application function definition. The application options (IP and MAC addresses, etc.) are set by modifying the *uipopt.h* file. Examples on how to write an application layer can be found in the uIP Reference Manual.

A simple example application has been implemented as a demonstration of stack functionalities. The application modules are *httpd.c*, *httpd.h*. The application is a simple Web Server that manages a typical Web site: a home page and several linked pages. The html files are stored as char vectors and managed by the modules *fs.c* and *fs.h*. These modules access the files using a simple file system.

3 Conclusion

The solution provides a way to add Ethernet connectivity features to STR710 with minimum resources load. It uses a minimal part of one chip-select addressing space of the EMI and since it is hooked to the RTC alarm, it is possible to calibrate the CPU load for its processing. The stack also supports SLIP protocol; in this case it also commits the UART0 resource. The user can easily customize the application by modifying just three files. In the end, the firmware uses little of STR710's memory resources: 65.04 KB of Flash and 7.16 KB of RAM, correspondingly the 25.4% of the Flash and the 11.18% of the SRAM; of course these values depend on the files size of the web server that are variable.

Table 1. Definitions

Acronym	Definition
TCP	Transmission Control Protocol
IP	Internet Protocol
UDP	User Data Protocol
ICMP	Internet Control Message Protocol
ARP	Address Resolution Protocol
EMI	External Memory Interface
ISA	Industry Standard Architecture
SLIP	Serial Line Internet Protocol
MAC	Media Access Control
LAN	Local Area Network
ISO	International Standards Organization
OSI	Open Systems Interconnection
SMTP	Simple Mail Transfer Protocol
DNS	Domain Name System
RTC	Real Time Clock

4 References

[1] STR71x datasheet.

[2] EISA Specification, Version 3.12.

[3] PS/2 Technical Reference - AT Bus Systems.

[4] CIRRUS LOGIC® Crystal LANTM CS8900A ETHERNET CONTROLLER TECHNICAL REFERENCE MANUAL.

[5] CIRRUS LOGIC® Crystal LANTM CS8900A ETHERNET CONTROLLER CS8900A product datasheet.

[6] uIP 0.9 Reference Manual.

Full product information is available at http://www.st.com/mcu

5 Revision history

Date	Revision	Changes
23-Feb-2006	1.0	Initial release