Freescale Semiconductor
Application Note

Document Number: AN3507
Rev. 0, 09/2007

Small Footprint ColdFire TCP/IP Stack

by: Eric Gregori
Product Specialist Embedded Software
Chicago

1 Introduction

The ColdFire® TCP/IP stack is a public source stack provided for use with the ColdFire line of processors. The stack is very robust and highly configurable. It supports most of the commonly used protocols, and includes many sample applications.

This application note discusses how to configure the TCP/IP stack for minimum flash and RAM usage. For complete details on the ColdFire TCP/IP stack, refer to application note AN3470. For details on the Freescale Web Server refer to Application Note AN3455.

ColdFire TCP/IP stack features:

- Hyper text transport protocol (HTTP) server
- HTTP client
- RSS/XML client
- TCP/UDP client and servers
- Serial-to-Ethernet client and servers
- Trivial file transfer (TFTP)

Contents

- Dynamic host configuration protocol (DHCP) or manual IP configuration
- Domain name server client (DNS)
- Transmission control protocol (TCP)
- User datagram protocol (UDP)
- Internet control messaging protocol (ICMP)
- BOOTstrap protocol (BOOTP)
- Address resolution protocol (ARP)
- Internet protocol (IP)

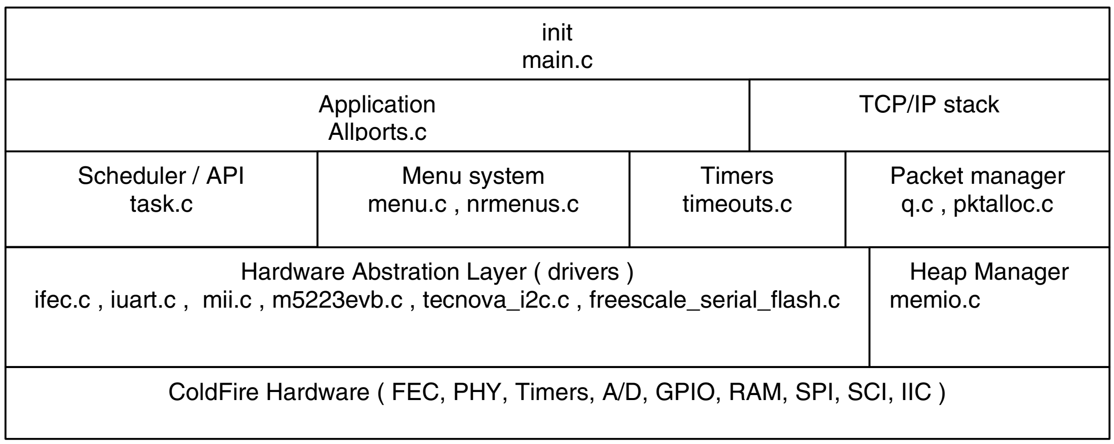

Figure 1. ColdFire TCP/IP Stack

2 ColdFire TCP/IP Flash and RAM Requirements

These targets are discussed fully in application note AN3470.

Table 1. Flash and RAM Requirements for Various ColdFire TCPIP Builds

Target	Flash (bytes)	BSS+DATA (bytes)	Stack (bytes)	Heap (bytes)	Total RAM (bytes)
Stack Only	33744	2820	1024	7852	11696
UDP Client	34368	2856	1024	8870	12750
TCP Client	35344	2938	1024	8870	12832
UDP Server	34176	2856	1024	8870	12730
TCP Server	35520	3202	1024	8870	13096
TCP Serial Server	36176	3198	1024	8870	13092
TCP Serial Client	36256	3198	1024	8870	13092
Web Server	45264	4660	1024	9894	15578

2.1 ColdFire TCP/IP Configuration for Above Results

2.1.1 ipport.h - Component Disabling to Save Flash

Comment out (disable)	In_Menus
	Net_ Stats
	DNS_ Client
	DHCP_ Client
	NPDEBUG

#define INCLUDE_ARP	1	/* use Ethernet ARP */
#define FULL_ICMP	1	/* use all ICMP || ping only */
#define OMIT_IPV4	1	/* not IPV4, use with MINI_IP */
#define MINI_IP	1	/* Use Nichelite mini-IP layer */
#define MINI_TCP	1	/* Use Nichelite mini-TCP layer */
#define MINI_PING	1	/* Build Light Weight Ping App for Niche Lite */
#define BSDISH_RECV	1	/* Include a BSD recv()-like routine with mini_tcp */
#define BSDISH_SEND	1	/* Include a BSD send()-like routine with mini_tcp */
#define NB_CONNECT	1	/* support Non-Blocking connects (TCP, PPP, et al) */
#define MUTE_WARNS	1	/* gen extra code to suppress compiler warnings */
//#define IN_MENUS	1	/* support for InterNiche menu system */
//#define NET_STATS	1	/* include statistics printfs */
#define QUEUE_CHECKING	1	/* include code to check critical queues */
#define INICHE_TASKS	1	/* InterNiche multitasking system */
#define MEM_BLOCKS	1	/* list memory heap stats */
#ifdef TFTP_PROJECT	1	
#define TFTP_CLIENT	1	/* include TFTP client code */
#define VFS_FILES	1	/* include Virtual File System */
#endif	1	
// EMG #define TFTP_SERVER	1	/* include TFTP server code */
// #define DNS_CLIENT	1	/* include DNS client code */
#define INICHE_TIMERS	1	/* Provide Interval timers */
// EMG - To enable DHCP, uncomment the line below		
//#define DHCP_CLIENT	1	/* include DHCP client code */
// EMG #define INCLUDE_NVPARMS	1	/* non-volatile (NV) parameters logic */

```
//#define NPDEBUG              1    /* turn on debugging dprintf()s */
// EMG #define
USE_MEMDEV                     1    /* Psuedo VFS files mem and null */
#define NATIVE_PRINTF          1    /* use target build environment's printf function */
#define NATIVE_SPRINTF         1    /* use target build environment's printf function */
#define PRINTF_STDARG          1    /* build ...printf() using stdarg.h */
//#define TK_STDIN_DEVICE      1    /* Include stdin (uart) console code */
#define BLOCKING_APPS          1    /* applications block rather than poll */
#define INCLUDE_TCP            1    /* this link will include NetPort TCP w/MIB */
```

2.1.2 ipport.h – Decrease the Number of Network Buffers to Save RAM

```
/* define number and sizes of free buffers */
#define NUMBIGBUFS   4                    //4 * 1552 bytes = 6208 bytes
#define NUMLILBUFS   3                    // 3 * 200 bytes = 600 bytes

/* FEC buffer descriptors */
#define NUM_RXBDS    1                    // # of bigbufs - 3
#define NUM_TXBDS    2
```

2.1.3 main.c – Set the Network Buffer Sizes

```
bigbufsiz = 1536 + 16;                    // 1552 bytes / buffer
lilbufsiz = 200;                          // 200 bytes / buffer
```

2.1.4 *lcf File (Link Command File) – Set System Stack Size

```
___SP_END      = .;
.            = . + (0x400);// 1024 bytes  // 1024 bytes
___SP_INIT     = .;
```

2.1.5 osport.h – Set Task Stack Size

```
/* task stack sizes */
#define  NET_STACK_SIZE        2048     // Not used
#define  APP_STACK_SIZE        1024     //  Application task's stack size
#define  CLOCK_STACK_SIZE      512      // Clock task's stack size

#define  IO_STACK_SIZE         1024     // STDIN task's stack size – TK_STDIN_TASK
```

2.1.6 Specific Applications Task Size

```
//****************************************************************************
// Declare Task Object
//****************************************************************************
TK_OBJECT(to_emgtcpsrv);
TK_ENTRY(tk_emgtcpsrv);
struct inet_taskinfo emg_tcp_task = {
        &to_emgtcpsrv,
        "EMG TCP server",
        tk_emgtcpsrv,
        NET_PRIORITY,
        0x400
        };
```

2.2 Small Footprint Results

Using the configuration from Figure 1, the TCP/IP stack was instrumented to show HEAP usage.

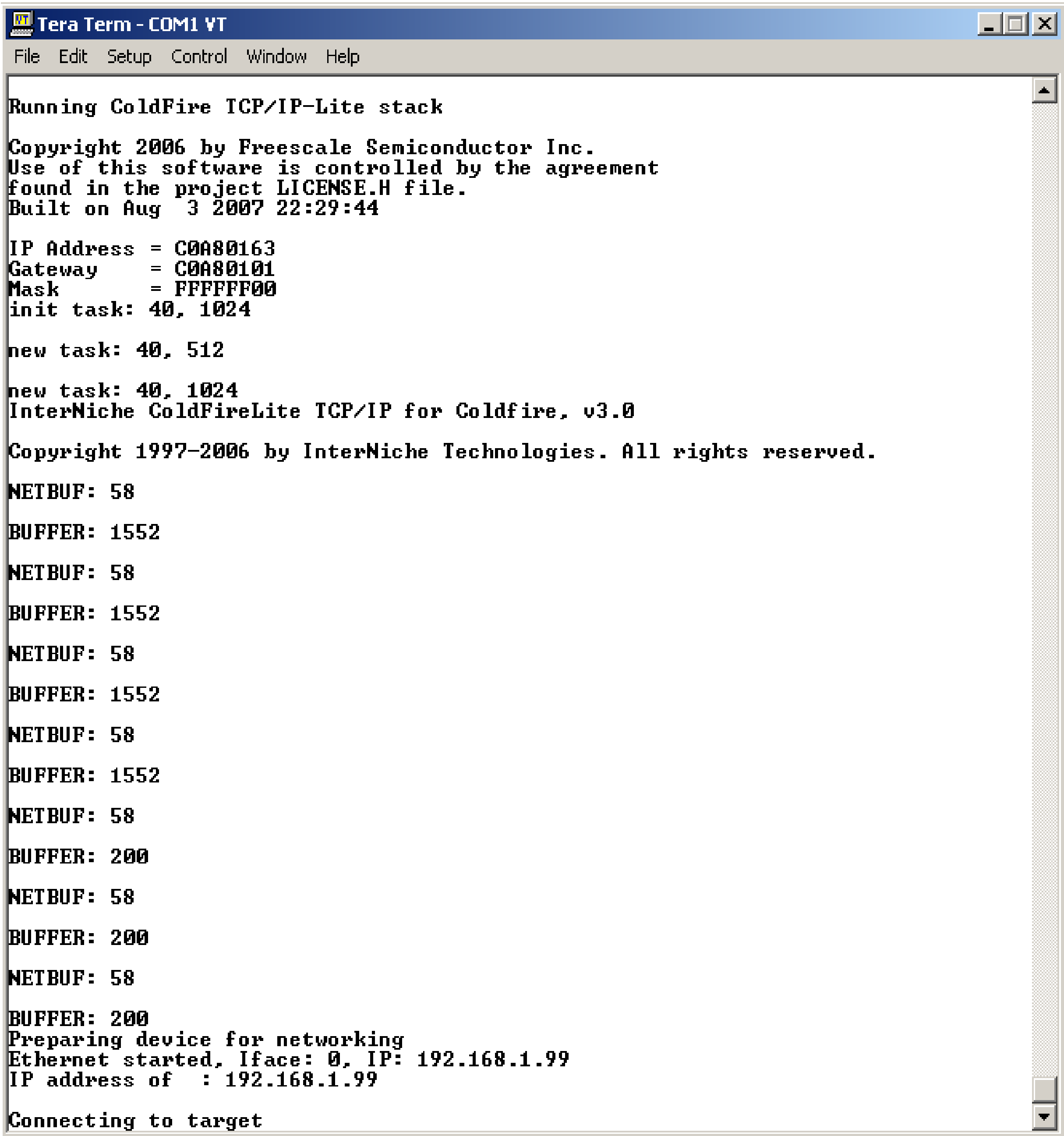

Figure 2. Instrumented TCP Client Boot

init task: 40, 1024

— The init task function creates the first task, (the network task) and assigns that task the system stack. This function allocates 40 bytes from the HEAP for the task control block. The 1024 bytes is the size of the system task and is not allocated from the HEAP.

new task: 40, 512

— The clock task is the first task to be started. It has a 512 byte call stack. The 40 byte TCB and the 512 bytes for the call stack are allocated out of the HEAP.

new task: 40, 1024

— The application task allocates 1024 byte for its call stack from the HEAP. It also requires a TCB.

NETBUF: 58

BUFFER: 1552

— At this point, all the tasks are up and running. The TCP/IP stack starts allocating its network buffers from the HEAP. NETBUF is a structure used to manage the network buffers. The NETBUF structure is allocated from the HEAP. The buffer in this case is a big buffer. There are four big buffers. Each big buffer requires 1610 bytes of HEAP space.

NETBUF: 58

BUFFER: 200

— The little buffers are only 200 bytes. Each little buffer also requires a network buffer to be allocated. Each little buffer requires 258 bytes of HEAP space.

Results:

- Call Stacks + overhead = (3 * 40) + 1024 + 512 = 1656 bytes
- Buffers + overhead = (7 * 58) + (4 * 1552) + (3 * 200) = 7214 bytes

Total heap required (1024 byte application stack) = 8870.

Total heap required (2048 byte application stack) = 9894.

3 ColdFire TCP/IP Components

The flash and RAM sizes in Figure 1 are for only TCP or UDP projects. The ColdFire TCP/IP stack includes additional protocols or features to enable embedded network devices. These additional protocols or features (DHCP, DNS, and the console) are discussed fully in the application note AN3470. These additional protocols or features are enabled or disabled by a set of defines in the file ipport.h.

Table 2. Stack Component, Features FLASH and RAM Adders

Component	Description	Flash Adder (Bytes)	RAM Adder (Bytes)
NET_STATS	statistics printfs	15200	442
DNS_CLIENT	DNS client	2715	51
DHCP_CLIENT	DHCP client	6907	117
NPDEBUG	debugging dprintf()s	3408	32
TK_STDIN_DEVICE IN_MENUS	Stdin (uart) console and menu system	8272	600

The base stack (no applications) uses: 33744 bytes of flash and 2820 bytes of BSS RAM. Adding features like the ones specified above, increases the flash and RAM footprint. Most of the base stacks RAM requirements come from the HEAP.

3.1 Example: Calculating Flash and RAM Requirements

TCP client with DHCP and DNS enabled:

Base FLASH = 35344 (from table in section 1)

DHCP adder = 6907

DNS adder = 2715

Total Flash Required = 35344 + 6907 + 2715 = 44966 bytes

Base RAM = 12832 (from table in section 1)

DHCP adder = 117

DNS adder = 51

Total RAM Required = 12832 + 117 + 51 = 13000 bytes

4 ColdFire TCP/IP Memory Model

DATA_RAM (Init data) BSS (Static vars)	RAM Start-_DATA_RAM
System stack (size set in*.lef)	Stack start
HEAP (manage by TCP/IP stack) Task stacks Network buffers Misc.structures	HEAP start RAM end

Figure 3. ColdFire TCP/IP RAM Memory Model

A ColdFire TCP/IP project uses three types of RAM. The BSS or DATA_RAM is assigned by the linker. This RAM contains the global variables, and is considered static. The system stack is defined by the linker, explicitly specified in the projects *.lcf file. The system stack size is hard-coded in the *.lcf file. The system stack is the call stack used by the ColdFire out of reset, and used by the ColdFire TCP/IP stack for its network task.

The HEAP is managed by the ColdFire TCP/IP stack. The HEAP is also used by the TCP/IP stack for network buffers, task stacks, and the temporary storage of misc.structures.

4.1 HEAP – Task Stacks

If the RTOS is enabled, it provides a dynamic mode of operation for the stack. Tasks can be created and destroyed at runtime. As a task is created a new stack is created for the task by allocating RAM from the HEAP. If the task is destroyed, the RAM allocated for the tasks stack is returned to the heap. This way your application is dynamic. Using the RTOS may be more RAM efficient.

If a task is created, memory is allocated from the RAM (via the HEAP) for the new tasks stack. The size of each stack is static, determined at compile time. The stack size must be chosen so that it's big enough to accommodate not only the task needs, but any interrupts used by the system.

With the RTOS enabled each task has a task control block (TCB). The TCBs are linked together in a linked list. The TCB structure is declared in task.h. This simple RTOS does not support task priorities. The scheduler simply increments to the next TCB in the list executing the task pointed to by the TCB if the task is ready to execute and not sleeping. The RTOS is also non-preemptive, task switching occurs only when a task gives up control. A task gives up control by calling tk_block() or goes to sleep (tk_sleep()).

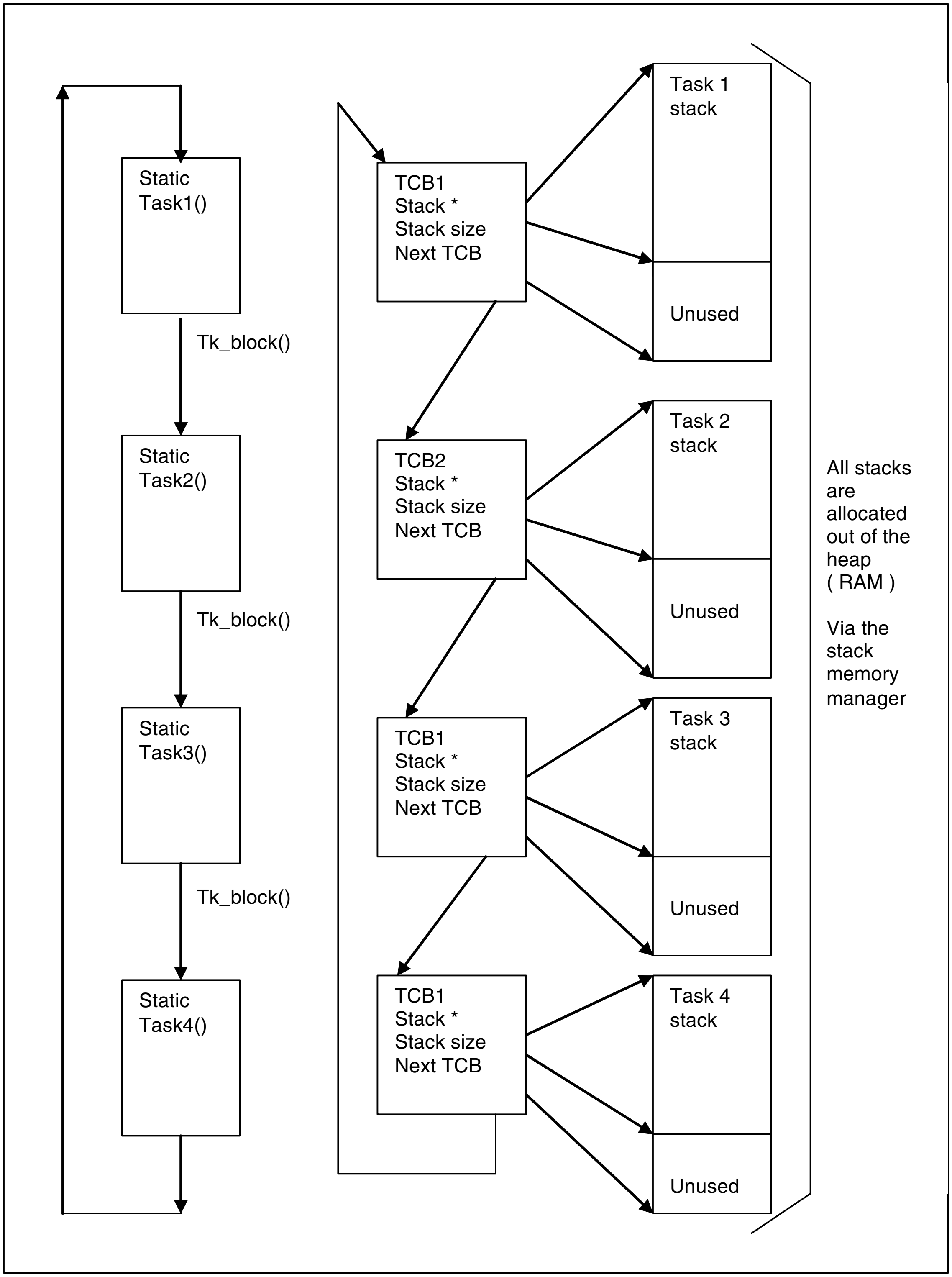

Figure 4. ColdFire TCP/IP Memory Allocation

4.2 HEAP – Network Buffer

The TCP/IP stack uses RAM for packet storage. Packets are stored in buffers managed by a dedicated buffer queue, not the HEAP manager. The module pktalloc.c contains the packet buffer memory manager.

There are two categories of packet buffers, based on the size of the buffer:

- bigbufsiz is declared in pktalloc.c. It determines the size of the big buffers.
- lilbufsiz is declared in pktalloc.c. It determines the size of the small buffers.

Ethernet receive operations use only the big buffers. Transmit operations use either big buffers or small buffers depending on how big the packet is. The receive operation must use only big buffers, because the size of the packet received is not known until the whole packet is received. Bigbufsiz is set to be larger than any packet that is to be received.

When the stack wants to send a packet, it calls pk_alloc (length). This length equals the desired packet length. Pk_alloc() uses either a big buffer or small buffer depending on the desired packet size. This is done for RAM efficiency. Make sure for TCP applications that the lilbufsiz is greater then the size of a ACK packet (60 bytes with ether header).

RX buffers must be large enough to accept a full size ethernet packet. The macro MAX_ETH_PKT defined in fecport.h sets the maximum packet size that can be received or transmitted by the fast Ethernet controller (FEC). Bigbufsiz must be larger then MAX_ETH_PKT. The FEC requires that the packet buffer be on a 16 byte boundary, so we add 16 bytes to the buff size to accommodate alignment.

bigbufsiz >= MAX_ETH_PKT + 16

lilbufsiz must be greater then TCP ACK packet size (60)

The number of buffers is a trade off between performance and available RAM. The stack allocates packet buffer RAM from its HEAP during initialization. The packets buffer RAM is never returned to the HEAP. Packet buffers are managed by a separate and independent memory manager (pktalloc.c). The primary things to consider while determining the number of buffers are network performance requirements and traffic, even if the embedded application does not require heavy Ethernet traffic. If there is heavy traffic on the network connection more buffers are required. Although the FEC filters Ethernet addresses, the broadcast addresses from ARP requests are passed to the stack, using packet buffers. A small number of packet buffers could affect broadcast packets on stack performance. It is important that any changes to the number of big buffers be tested in a network environment similar to the environment where the final device will be used.

TCPTV_MSL defines the amount of time a TCP connection waits in the CLOSE-WAIT state. If a socket is closed, it does not close immediately. It waits TCPTV_MSL * 2 seconds before actually closing and releasing the packet buffers. In an environment where the connection is often opened and closed, and waiting too long to free up, a packet buffer from a previously closed connection can result, while all the packet buffers being locked up waiting for TCPTV_MSL timeout.

TCP_MSS or TCP maximum segment size sets the maximum number of data bytes a TCP segment can hold. This value must be smaller then bigbufsiz. In fact, the MSS must be less then the bigbufsiz – TCP header – IP header – Ethernet header. If sending large amounts of data, the higher the TCP_MSS the better, within the limits mentioned above. The total data sent is broken up into (total data size) /TCP_MSS TCP segments. The more segments, the more overhead to ACK the segments, resulting in a decrease in performance.

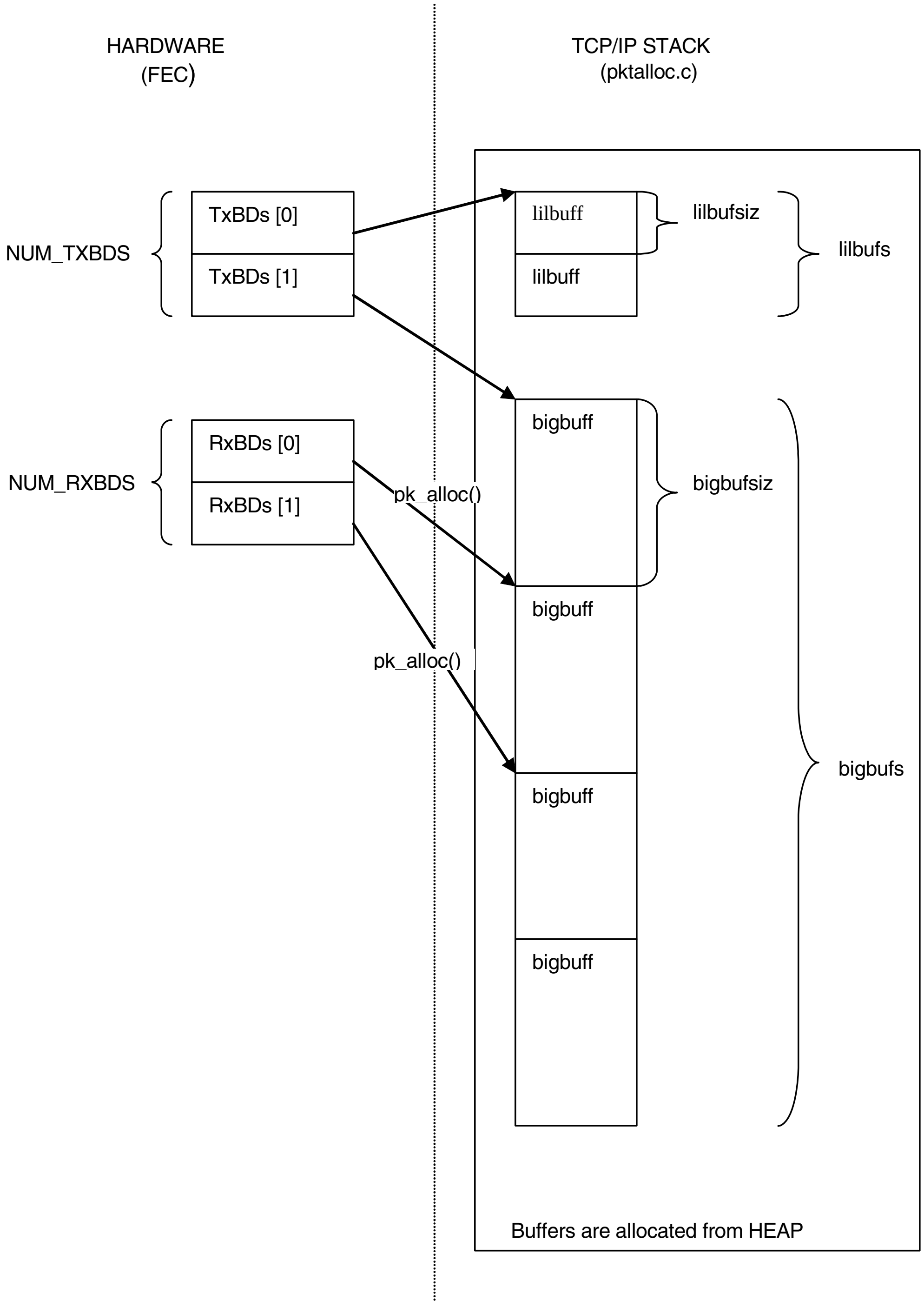

Figure 5. Packet Buffer Usage

5 Conclusion

Using the information in this application note the user can significantly reduce the flash and RAM requirements for the ColdFire TCP/IP stack.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN3507
Rev. 0
09/2007

freescale™
semiconductor