



Optimized memory copy routine for TCP/IP on the STR91x

Introduction

This application note describes an optimized memory copy for TCP/IP. It can also be used for any memory copy.

This routine is written in ARM assembler, and can be called by any C program which conforms to the ARM procedure call standard.

Using this optimized routine will accelerate the memory copy up to 20 times as compared to a standard implementation.

This optimized buffer alignment routine has been integrated in the HAL Ethernet driver distributed by ST.

This program has been fully tested and run inside a TCP/IP stack. However, STMicroelectronics cannot guarantee that this driver is flawless in all applications.

Contents

1 Background 3

 1.1 The problem 3

 1.2 The solution 3

2 The optimized routine 4

 2.1 Interface 4

 2.2 Algorithm description 4

 2.3 Code description 4

 2.4 Performance results 5

 2.5 Portability 6

 2.6 Extension 6

Appendix A Code (IAR Version) 7

Appendix B Example of C call 10

3 Revision history 11

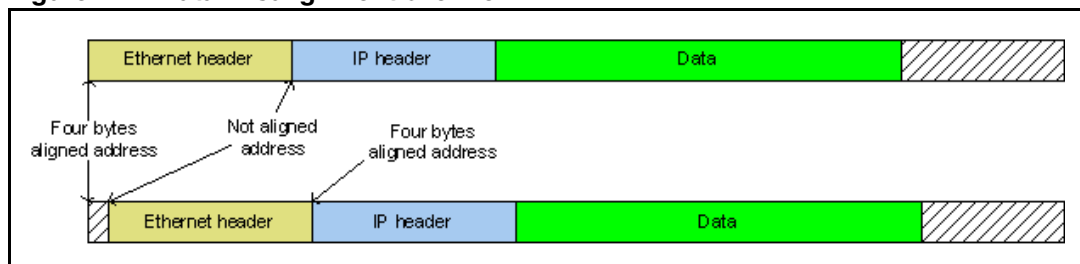
1 Background

1.1 The problem

In a TCP/IP application, the packets are transferred from the STR91x Ethernet MAC unit to STR91x on-chip memory using one of the STR91x DMA channels.

Inside this packet will be a 14-byte header, which is the source of the misalignment. All following data will be misaligned by 2 bytes, making the processing of the packet by the CPU much slower.

Figure 1. Data misalignment overview



1.2 The solution

The solution is to have an optimized routine which will read the source buffer and write in the destination buffer with an offset of 2 bytes. Hence the data will now be fully aligned.

Table 1. Memory copy example

	Before	After
Data starts at address	0x040001A8E (misaligned)	0x040001A8C (aligned)

2 The optimized routine

2.1 Interface

```
void MEMCOPY_L2S_BY4(void *pDEST, void *pSRC, ushort NbrByte)
```

Note about pointers:

In this description above, the pointers are pointing to a void.

A way of looking at it is that the pointer is pointing to a buffer of long with the first 2 bytes filled with zeros.

Another way of looking at it is that the pointer is pointing to a buffer of chars.

While it is beyond this application note to explain the problem of mapping bytes, short and long in the same memory space in C, it is worth noting that the routine could be declared differently.

In fact, when running the program to test the routine on its own, it was declared as

```
void MEMCOPY_L2S_BY4(short *pDEST, long *pSRC, long NbrByte)
```

See [Appendix B on page 11](#) for an example of the C call.

2.2 Algorithm description

The algorithm works as follows:

- First level - granularity 16: The CPU transfer 16 bytes per iteration until the number of bytes is less than 16. Go to second level.
- Second level: Granularity 4: Depending on the remaining number of bytes, the CPU transfer either 4, 8 or 12 bytes. Go to third level.
- Third level: Granularity1: Depending on the remaining number of bytes, the CPU transfer either 1, 2 or 3 bytes.

2.3 Code description

There are 5 blocks in this routine.

Table 2. Overview of code blocks

Name	Function	Implementation
save_context	save all registers used by the routine	
d_main	main loop; transfer 16 bytes per iteration	single loop
d_swit	second level switch: transfer 4, 8 or 12 bytes	linear routine
d_switch	third level switch: transfer 1, 2 or 3 bytes	switch case
restore_context	restore all registers	

The code listing is given in appendix A.

2.4 Performance results

The following table shows the number of cycles for different number of bytes all divisible by 16.

From this table the following formula can be used:

$$\text{Cycles} = 84 + \text{NBytes} \times 1.5$$

84 cycles represent the total overhead of: Caller (C) transmit the 3 parameters, branch to routine, routine context save, routine context restore, return to caller.

Table 3. Performance results (no. of bytes divisible by 16)

Number of bytes	Number of cycles
32	126
128	276
512	852
1024	1620

When the number of bytes is not divisible by 16, then the number of cycles is not linear. The following formula must be used:

$$\text{Cycles} = 84 + \text{NBytes} \times 1.5 + \text{Additional Cycles for remaining bytes}$$

Table 4. Performance results (no. of bytes not divisible by 16)

Remaining bytes	Additional cycles
0	0
1	27
2	27

Table 4. Performance results (no. of bytes not divisible by 16)

Remaining bytes	Additional cycles
3	17
4	17
5	44
6	44
7	34
8	34
9	61
10	61
11	49
12	38
13	65
14	65
15	55

All these results were obtained on a STR910-EVAL best case configuration for speed. The clock is 96 MHz, the number of wait states is minimum for Flash and for SRAM.

The code is run from Flash.

Using the former table and adding a line for a 342-byte TCP-IP packet gives the following numbers.

Table 5. Performance results summary

Number of bytes	Number of cycles	Number of microseconds
32	126	1.38
128	276	2.88
342	1620	6.58
512	852	8.88
1024	1620	16.88

2.5 Portability

Since the code is written in assembler, in principle it can be used with any other toolchain. However it contains a keyword DC32 that is a directive specific to the IAR toolchain.

This directive is used to declare a 32-bit constant.

This can be easily ported by using the Toolchain respective directives.

GNU tool chain .word

ARM ADS Toolchain DCD

Obviously, all the declarative directives such as `MODULE`, `PUBLIC`, etc... must also be changed.

2.6 Extension

From this routine, other similar optimized memory copy routines can be derived. This was done in a form of a small library for GNU.

- 16-bit aligned to 32-bit aligned
- 32-bit aligned to 16-bit aligned
- 32-bit aligned to 32-bit aligned

The file is given as an additional file (`memcpy.s`).

Appendix A Code (IAR Version)

```

MODULE MEMORYP

PUBLIC MEMCOPY_L2S_BY4

RSEG ICODE:CODE(4)

CODE32

MEMCOPY_L2S_BY4

    STMFD        SP!, {R4-R10}

d_main

    LDMIA        R1!, {R3,R5,R7,R9}
    MOV          R4, R3, LSR #16
    MOV          R6, R5, LSR #16
    MOV          R8, R7, LSR #16
    MOV          R10, R9, LSR #16
    STRH         R3, [R0], #2
    STRH         R4, [R0], #2
    STRH         R5, [R0], #2
    STRH         R6, [R0], #2
    STRH         R7, [R0], #2
    STRH         R8, [R0], #2
    STRH         R9, [R0], #2
    STRH         R10, [R0], #2
    SUB          R2, R2, #16
    CMP          R2,#16      ; if ( remaining_bytes >= 16)
    BHS          d_main      ; keep on looping
                                ; else switch(remaining_bytes)

d_swit

d_0123

    CMP          R2,#3      ; remaining <= 3 bytes?
    BLS          d_switch

d_4567

    LDR          R3, [R1], #4
    MOV          R4, R3, LSR #16
    STRH         R3, [R0], #2
    STRH         R4, [R0], #2

```

```

        SUB        R2, R2, #4
        CMP        R2,#3    ; remaining <= 3 bytes?
        BLS        d_switch    ;    switch(remaining_bytes 0,1,2,3)
d_891011
        LDR        R3, [R1], #4
        MOV        R4, R3, LSR #16
        STRH       R3, [R0], #2
        STRH       R4, [R0], #2
        SUB        R2, R2, #4
        CMP        R2,#3    ; remaining <= 3 bytes?
        BLS        d_switch    ;    switch(remaining_bytes 0,1,2,3)
d_12131415
        LDR        R3, [R1], #4
        MOV        R4, R3, LSR #16
        STRH       R3, [R0], #2
        STRH       R4, [R0], #2
        SUB        R2, R2, #4
d_switch
        LDR PC,[PC, R2, LSL#2]
        B d_END
DC32    d_END
DC32    d_1B
DC32    d_2B
DC32    d_3B
d_1B
        LDRB       R3, [R1]
        STRB       R3, [R0]
        B          d_END
d_2B
        LDRH       R3, [R1]
        STRH       R3, [R0]
        B          d_END
d_3B
        LDRH       R3, [R1], #2
        LDRB       R4, [R1]

```

```
        STRH      R3, [R0], #2
        STRB      R4, [R0]
        B         d_END
d_END
        LDMFD     SP!, {R4-R10}
        MOV       PC, LR
END
```

Appendix B Example of C call

```
ulong DMACHandleRxPkt(void* ppkt, ushort size)
{
    __DMA_DSCR_Base    *pDescr;
    dmacEthernetif     *ptDMACIf;
    ulong              value;
    ptDMACIf = GetDMACContext();
    /* Check if the descriptor is owned by the DMA or CPU */
    value = (ulong) (ptDMACIf->pDmaRxDscrBase->dmaPackStatus &
DMA_DSCR_RX_STATUS_VALID_MSK);
    /* Check for valid mask */
    if (value == DMA_DSCR_RX_STATUS_VALID_MSK)
    {
        return (DMAC_NOK);
    }
    // Get the Rx Descriptor base
    pDescr = ptDMACIf->pDmaRxDscrBase;
    // Check for validity
    if ( (pDescr != NULL)  && (pDescr->dmaAddr) )
    {
        size = ((pDescr->dmaPackStatus & 0x7ff) - 4);
        MEMCOPY_L2S_BY4((char *)ppkt, tempRxAry, size);
    }
    else return DMAC_NOK;
    /* Give the buffer back to DMAC */
    pDescr->dmaPackStatus = DMA_DSCR_RX_STATUS_VALID_MSK;
    /* Start the receive operation */
    DMAC_REG_BIT_SET( ptDMACIf->pRfDMA->dmaRxStart,
DMA_RX_START_FETCH);
    // Return no error
    return DMAC_OK;
}
```

3 Revision history

Table 6. Document revision history

Date	Revision	Changes
30-May-2006	1	Initial release.
02-Jun-2008	2	Removed references to obsolete products

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST's terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST'S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED REPRESENTATIVE OF ST, ST PRODUCTS ARE NOT DESIGNED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS, WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2008 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com