```
/*
 * messaging.h
 *
 * Todd Hiers
 *
 * DSP/BIOS messaging subsystem header file
 *
 * THIS VERSION ASSUMES ONE PROCESS PER PROCESSSOR
 *
 */

#include <std.h>
#include <log.h>
#include <lck.h>
#include <mbx.h>
#include <tsk.h>
#include <mem.h>

extern far LOG_Obj MSG_DEBUG_LOG;
extern far LOG_Obj MSG_DEBUG_LOG2;
extern far LCK_Obj MSG_PCI_LCK;
extern far MBX_Obj MSG_HWI_PCI_MBX;
extern far MBX_Obj MSG_RECV_MBX;
extern far TSK_Obj MSG_PROCESS_MBXTABLE;
extern far int MSG_MEM_User0;
extern far int MSG_MEM_User1;
extern far SEM_Obj MSG_PCI_SEM;



#define hMSG_DEBUG_LOG (& MSG_DEBUG_LOG)
#define hMSG_PCI_LCK (& MSG_PCI_LCK)
#define hMSG_HWI_PCI_MBX (& MSG_HWI_PCI_MBX)
#define hMSG_RECV_MBX (& MSG_RECV_MBX)
#define hMSG_PROCESS_MBXTABLE (& MSG_PROCESS_MBXTABLE)
#define hMSG_BUFFER_MEM (& MSG_BUFFER_MEM)
#define hMSG_PCI_SEM (& MSG_PCI_SEM)

#define MSG_ERROR 0
#define MSG_OK 1

#define MSG_MAX_PROCESSES 32
#define MSG_MAX_SIMUL_RECV 4

#define MSG_CONST_RTS 0x00100000
#define MSG_CONST_DONE 0x00200000
#define MSG_CONST_INIT 0x01000000

#define MSG_TSK_SEND_PRI 5
#define MSG_TSK_RECV_PRI 5

#define MSG_PROC_TYP_ONBRD 'b'
#define MSG_PROC_TYP_ONCHP 'c'
#define MSG_PROC_TYP_ONNET 'n'



#define PCI_CMD_MasterWrite 0x00000001;

#define PCI_DSPMA (* (volatile unsigned int **) 0x01A40010)
#define PCI_PCIMA (* (volatile unsigned int *) 0x01A40014)
#define PCI_PCIMC (* (volatile unsigned int *) 0x01A40018)
#define PCI_PCIIS (* (volatile unsigned int *) 0x01A40008)
#define PCI_PCIIENA (* (volatile unsigned int *) 0x01A4000C)

struct MSG_HWI_PCI_MBX_MSG {
   MBX_Handle hMbx;
```

```
    int pid;
};

typedef struct MSG_Thread {
    TSK_Handle hTsk;
    SEM_Handle hSem;
} MSG_Thread, *MSG_Thread_Handle;


/*
struct MSG_RECV_TSK_MSG {
    int src_pid;
    int dst_pid;
    unsigned int msg;
};
*/

void MSG_Send(int dst_pid, void * data, int wordcount,
                MSG_Thread_Handle hThread);
void MSG_Recv (int src_pid, void * data, int wordcount,
                MSG_Thread_Handle hThread);
void MSG_Init(void);
int MSG_IsThreadDone(MSG_Thread_Handle);
void MSG_ThreadFinalize(MSG_Thread_Handle);
void MSG_ThreadWait(MSG_Thread_Handle);
```

```
/*
 * BIOS_messaging.c
 *
 * Todd Hiers
 *
 * Implements messaging subsystem for c6205 multiprocessor board
 *
 * THIS VERSION ASSUMES ONE PROCESS PER PROCESSSOR
 *
 */

#include <std.h>
#include <log.h>
#include <tsk.h>
#include <mbx.h>
#include <sys.h>
#include <lck.h>
#include <mem.h>
#include <hwi.h>
#include <c6x.h>
#include "messaging.h"

unsigned int MSG_Interrupt = 0x00000002;

/* vairbles that must be defined by the host at boot time */

const unsigned int * MSG_RemoteMailbox    = (unsigned int *) 0x40;
const unsigned int * MSG_RemoteInt        = (unsigned int *) 0x80;
const char * MSG_RemoteProcessType        = (char *) 0xC0;
const unsigned int * MSG_LocalMailboxTable= (unsigned int *) 0x400000;
const unsigned int * MSG_pLocalPID        = (unsigned int *) 0x20;
const unsigned int * MSG_pNumProcesses    = (unsigned int *) 0x24;
const unsigned int * MSG_pLinkAddr        = (unsigned int *) 0x30;
const unsigned int * MSG_pLinkCtrlAddr    = (unsigned int *) 0x34;

#define MSG_LocalPID (* MSG_pLocalPID)
#define MSG_LinkAddr (* MSG_pLinkAddr)
#define MSG_LinkCtrlAddr (* MSG_pLinkCtrlAddr)
#define MSG_NumProcesses (* MSG_pNumProcesses)


/* local globals */
MBX_Handle MSG_RecvMBX[MSG_MAX_PROCESSES];
volatile int MSG_PROCESS_MBX;

#pragma FUNC_EXT_CALLED (MSG_Send);
#pragma FUNC_EXT_CALLED (MSG_Recv);
#pragma FUNC_EXT_CALLED (MSG_Init);
#pragma FUNC_EXT_CALLED (MSG_IsThreadDone);
#pragma FUNC_EXT_CALLED (MSG_ThreadFinalize);


void MSG_TSK_SEND (int src_pid, int dst_pid, unsigned int * data, int
wordcount,
                   SEM_Handle sem) {

   /* thread to send a message - invoked for each message*/
   /* ASSUMES DEST'N PROCESS IS OFF-CHIP */
   /* NOT THREADSAFE FOR SAME SRC,DST - NEED TO FIX */

   MBX_Handle hthreadMBX;
   MBX_Attrs threadMBX_attrs;
   struct MSG_HWI_PCI_MBX_MSG pcimsg;
   unsigned int mbx_data;
   unsigned int msg;
   unsigned int addr;
```

```
    /* Create a mailbox to communicate with the PCI INT thread */
    threadMBX_attrs = MBX_ATTRS;
    hthreadMBX = MBX_create(4, 1, &threadMBX_attrs);

    /* tell the PCI INT thread about our mailbox */
    pcimsg.hMbx = hthreadMBX;
    pcimsg.pid = dst_pid;
    MBX_post(hMSG_HWI_PCI_MBX, & pcimsg, SYS_FOREVER);

    /* Send the RTS */
    /* new data flag | RTS message constant | size */
    mbx_data = 0x80000000 + MSG_CONST_RTS + wordcount;
    if (! MSG_SendToMailbox(src_pid, dst_pid, mbx_data, FALSE))
       SYS_abort("Could not send RTS. Help!");

    /* wait for the CTS reply */
    MBX_pend(hthreadMBX, & msg, SYS_FOREVER);

    /* Are we CTS? */
    if ((msg & 0xC0000000) != 0xC0000000)
       SYS_abort("Targert replied, but refused CTS. Help!");
       /* NEED TO ALLOW FOR TRY_AGAIN_LATER (bsend) */

    addr = (msg & 0x3fffffff) << 2;
    /* empty, then destroy the mailbox */
    while (MBX_pend(hthreadMBX, & msg, 0));
    MBX_delete(hthreadMBX);

    /* send the actual data */
    if (! MSG_SendToBuffer(dst_pid, data, wordcount, addr))
       SYS_abort("Could not send data. Help!");

    /* Send the Done msg */
    /* new data flag | Done message constant | size */
    mbx_data = 0x80000000 + MSG_CONST_DONE + wordcount;
    if (! MSG_SendToMailbox(src_pid, dst_pid, mbx_data, FALSE))
       SYS_abort("Could not send DONE. Help!");

    /* Done - alert waiting semaphore, if necessary */
    /* boost our priority to allow to finish */
    if (sem != NULL) {
       TSK_setpri(TSK_self(),TSK_MAXPRI);
       SEM_ipost(sem);
    }


}

void MSG_TSK_RECV (int src_pid, int dst_pid, unsigned int * buf, int
wordcount,
                   SEM_Handle sem) {
    /* thread to receive a message */
    /* requires atomic sends from w/in src,dst pair */

    unsigned int addr;
    unsigned int msg;

    /* find msg info */
    /* RecvMBX array takes up space, but allows for cleaner
       receive thread blocking semantics */

    if (MSG_LocalPID != dst_pid)
       SYS_abort("dst_pid in MSG_RECV_TSK is not local");

    /* Atomic test & Set */
    HWI_disable();
    if (MSG_RecvMBX[src_pid] == NULL)
```

```
        MSG_RecvMBX[src_pid] = MBX_create(4, MSG_MAX_SIMUL_RECV,& MBX_ATTRS);
    HWI_enable();

    MBX_pend(MSG_RecvMBX[src_pid], & msg, SYS_FOREVER);
    /* value sent back is offset from Mailbox place */
    /* >>2 implicit in (u i) type (div by 4) */
    addr = (buf - MSG_LocalMailboxTable);

    /* send the CTS */
    MSG_SendToMailbox(dst_pid, src_pid, ((addr & 0x3FFFFFFF) | 0xC0000000) ,
                      TRUE);

    /* Wait for the done msg */
    MBX_pend(MSG_RecvMBX[src_pid], & msg, SYS_FOREVER);

    /* if our ack was a new send request, keep for next receive thread */
    /* otherwise delte the MBX */
    if ((msg & 0x7FF00000) == MSG_CONST_RTS)
        MBX_post(MSG_RecvMBX[src_pid], & msg, SYS_FOREVER);
    else {
        HWI_disable();
        MBX_delete(MSG_RecvMBX[src_pid]);
        MSG_RecvMBX[src_pid] = NULL;
        HWI_enable();
    }


    /* Done - alert waiting semaphore, if necessary */
    /* boost our priority to allow to finish */
    if (sem != NULL) {
        TSK_setpri(TSK_self(), TSK_MAXPRI);
        SEM_post(sem);
    }

}

void MSG_HWI_PCI_DISPATCH (void) {

 /* figure out action neeed to deal with PCI interrupt and do it */
    if (PCI_PCIIS & 0x00000008) {
        /* Host interrupt */
        /* clear interrupt source register */
        PCI_PCIIS = 0x00000008;

        /* schedule the mailbox checker task */
        MSG_PROCESS_MBX = 1;
        TSK_setpri(hMSG_PROCESS_MBXTABLE, 6);

    }
    if (PCI_PCIIS & 0x00000040) {
        /* Master Transfer Successfully Completed */
        /* clear interrupt source register */
        PCI_PCIIS = 0x00000040;

        if (SEM_count(hMSG_PCI_SEM))
            SYS_abort("MSG_PCI_SEM already 1");

        /* Signal the PCI semaphore to wake the waiting task */
        SEM_post(hMSG_PCI_SEM);

    }

    if (PCI_PCIIS & 0xffffffb7)
        SYS_abort("unexpected PCIIS reason. Help!");

}
```

```
void MSG_TSK_PROCESS_MBXTABLE (void) {

    struct MSG_HWI_PCI_MBX_MSG pcimsg;
    unsigned int msg;
    int i;


    /* loop repeatedly and control execution via priority */
    /* static obj is more efficient */
    /* must check everything because we don't know who interrupted */
    while(1){

        MSG_PROCESS_MBX=0;

        /* check mailbox table */
        /* i effectively index - * (u i *) implicitly multiples offet by 4 */
        for (i=0; i < MSG_NumProcesses; i++){
            /* check CTS responses */
            if ((* (unsigned int *) (MSG_LocalMailboxTable + 2*i + 1))
                 & 0x80000000) {
                /* we have a new response */

                /* get the communicator mailbox */
                /* other way to do this is a static array of MBX on pid */
                /* this is better for sparse comunications */
                MBX_pend(hMSG_HWI_PCI_MBX, & pcimsg, SYS_FOREVER);
                while (pcimsg.pid != i) {
                    MBX_post(hMSG_HWI_PCI_MBX, & pcimsg, SYS_FOREVER);
                    MBX_pend(hMSG_HWI_PCI_MBX, & pcimsg, SYS_FOREVER);
                }

                /* notify waiting send thread */
                MBX_post(pcimsg.hMbx, (Ptr) (MSG_LocalMailboxTable + 2*i + 1),
                           SYS_FOREVER);

                /* clear the new flag */
                * (unsigned int *)  (MSG_LocalMailboxTable + 2*i + 1)
                    &= 0x7FFFFFFF;
            }



            msg = *  (MSG_LocalMailboxTable + 2*i);

            /* Check RTS messages */
            if (msg & 0x80000000) {
                /* we have a new RTS */
                /* THIS BEHAVIOR CHANGES FOR BUFFERED VS NON-BUFFERED SENDS */
                /* notify receive thread */
                if (MSG_RecvMBX[i] == NULL)
                    MSG_RecvMBX[i] = MBX_create(4, MSG_MAX_SIMUL_RECV,
                                                & MBX_ATTRS);

                MBX_post(MSG_RecvMBX[i], & msg, SYS_FOREVER);
                    /* clear the new flag */
                * (unsigned int *) (MSG_LocalMailboxTable + 2*i) =
                     msg & 0x7FFFFFFF;

            }
        }

        /* we've scanned the entire table - goto sleep if no interrupt*/
        /* turn off interrupts & task scheduler to do this atomically */
        TSK_disable();
        HWI_disable();
        if (! MSG_PROCESS_MBX)
            TSK_setpri(TSK_self(), -1);
```

```
        HWI_enable();
        TSK_enable();

    }
}



MSG_SetLinkAlias(unsigned int base){

    /* set addresses & start link alias transfer */
    PCI_DSPMA = & base;
    PCI_PCIMA = MSG_LinkCtrlAddr;
    PCI_PCIMC = (4 << 16) | PCI_CMD_MasterWrite;

    /* wait for the transfer to complete */
    while (PCI_PCIMC & 0x00000003 != 0)
        TSK_yield();
}



int MSG_SendToMailbox(int src_pid, int dst_pid, unsigned int data, int
response) {

/* send data word data to mailbox mbx and interrupt receiveing processor */

    unsigned int baseaddr;

    /* sanity check */
    if ((dst_pid < 0) || (dst_pid > MSG_NumProcesses))
        return MSG_ERROR;

    /* acquire PCI port lock */
    LCK_pend(hMSG_PCI_LCK, SYS_FOREVER);

    if (MSG_RemoteProcessType[dst_pid] == MSG_PROC_TYP_ONNET) {
        /* route thru serial link */
        /* set serial link addressing alias */
        MSG_SetLinkAlias(MSG_RemoteMailbox[dst_pid]);
        baseaddr = MSG_LinkAddr;
    } else
        baseaddr = MSG_RemoteMailbox[dst_pid];



    /* set addresses & start data transfer */
    PCI_DSPMA = & data;
    if (response)
        PCI_PCIMA = baseaddr + (src_pid*8) + 4;
    else
        PCI_PCIMA = baseaddr + (src_pid*8);
    PCI_PCIMC = (4 << 16) | PCI_CMD_MasterWrite;

    /* wait for the transfer to complete */
    /* TSK_yield sucks - better to use interrupt + SEM */
    /*while (PCI_PCIMC & 0x00000003 != 0)
        TSK_yield();
     */
     SEM_pend(hMSG_PCI_SEM, SYS_FOREVER);

    if (MSG_RemoteProcessType[dst_pid] == MSG_PROC_TYP_ONNET) {
        /* route thru serial link */
        /* set serial link addressing alias */
        MSG_SetLinkAlias(MSG_RemoteInt[dst_pid]);
        baseaddr = MSG_LinkAddr;
    } else
```

```
      baseaddr = MSG_RemoteInt[dst_pid];

   /* set addresses & start interrupt transfer */
   PCI_DSPMA = & MSG_Interrupt;
   PCI_PCIMA = baseaddr;
   PCI_PCIMC = (4 << 16) | PCI_CMD_MasterWrite;

   /* wait for the transfer to complete */
   /* TSK_yield sucks - better to use interrupt + SEM */
   /*while (PCI_PCIMC & 0x00000003 != 0)
      TSK_yield();
    */
    SEM_pend(hMSG_PCI_SEM, SYS_FOREVER);

    /* release PCI port lock */
    LCK_post(hMSG_PCI_LCK);

   return MSG_OK;
}


int MSG_SendToBuffer(int dst_pid, unsigned int * data, int wordcount, unsigned
int addr) {

/* send data to remote buffer */

   unsigned int baseaddr;

   /* sanity check */
   if ((dst_pid < 0) || (dst_pid > MSG_NumProcesses))
      return MSG_ERROR;

   /* acquire PCI port lock */
   LCK_pend(hMSG_PCI_LCK, SYS_FOREVER);

   if (MSG_RemoteProcessType[dst_pid] == MSG_PROC_TYP_ONNET) {
      /* route thru serial link */
      /* set serial link addressing alias */
      MSG_SetLinkAlias(MSG_RemoteMailbox[dst_pid]);
      baseaddr = MSG_LinkAddr;
   } else
      baseaddr = MSG_RemoteMailbox[dst_pid];


   /* set addresses & start data transfer */
   PCI_DSPMA = data;
   PCI_PCIMA = baseaddr + addr;
   PCI_PCIMC = (wordcount << 18) | PCI_CMD_MasterWrite;

   /* wait for the transfer to complete */
   /* TSK_yield sucks - better to use interrupt + SEM */
   /*while (PCI_PCIMC & 0x00000003 != 0)
      TSK_yield();
    */
    SEM_pend(hMSG_PCI_SEM, SYS_FOREVER);


    /* release PCI port lock */
    LCK_post(hMSG_PCI_LCK);

   return MSG_OK;

}

void MSG_Send(int dst_pid, void * data, int wordcount, MSG_Thread_Handle
hThread){
```

```
   /* decode dest to determine where it's going */
   /* spawn any send tasks */

   TSK_Attrs SendTskAttrs;

   /* can't send to ourselves */
   if (dst_pid == MSG_LocalPID){
      SYS_abort("Bad dst_pid in Send");
      return;
   }

   if (hThread != NULL) {
       SendTskAttrs = TSK_ATTRS;
       SendTskAttrs.priority = MSG_TSK_SEND_PRI;
       hThread->hSem = SEM_create(0,& SEM_ATTRS);
       hThread->hTsk = TSK_create((Fxn) MSG_TSK_SEND, & SendTskAttrs,
                        MSG_LocalPID, dst_pid, data, wordcount,
hThread->hSem);
   } else
      MSG_TSK_SEND(MSG_LocalPID, dst_pid, data, wordcount, NULL);

}

void MSG_Recv (int src_pid, void * data, int wordcount, MSG_Thread_Handle
hThread){

 /* spawn any RECV tasks */

   TSK_Attrs RecvTskAttrs;

   /* can't receive from ourselves */
   if (src_pid == MSG_LocalPID){
      SYS_abort("Bad src_pid in Recv");
      return;
   }

   if (data < (void *) 0x400000 || data >= (void *) 0x800000) {
      SYS_abort("Bad Address in Recv");
      return;
   }

   if (hThread != NULL) {
       RecvTskAttrs = TSK_ATTRS;
       RecvTskAttrs.priority = MSG_TSK_RECV_PRI;
       hThread->hSem = SEM_create(0,& SEM_ATTRS);
       hThread->hTsk = TSK_create((Fxn) MSG_TSK_RECV, & RecvTskAttrs,
src_pid,
                              MSG_LocalPID, data, wordcount, hThread->hSem);
   } else
      MSG_TSK_RECV(src_pid, MSG_LocalPID, data, wordcount, NULL);

}

int MSG_IsThreadDone(MSG_Thread_Handle hThread){

   TSK_Stat stat;

   TSK_stat(hThread->hTsk,& stat);
   return (stat.mode == TSK_TERMINATED);
}

void MSG_ThreadFinalize(MSG_Thread_Handle hThread){
   /* should be terminated - otherwise possible mem leak */

   SEM_delete(hThread->hSem);
   TSK_delete(hThread->hTsk);
}
```

```
void MSG_ThreadWait(MSG_Thread_Handle hThread){

    SEM_pend(hThread->hSem, SYS_FOREVER);
}


void MSG_Init(void){

    unsigned int msg;
    int i;

    /* Clear interrupt flags in PCIIS and turn on int 13 */
    /* Turn on MasterOK, HostSW, and MasterAbort Interrupts */
    /* reset PCI master */
    PCI_PCIMC = 0xffff0000;
    PCI_PCIIS = 0xffffffff;
    PCI_PCIIENA = 0x4c;
    IER = IER | 0x00002000;
    HWI_enable();


    /* clear static arrays */
    for (i=0;i<MSG_NumProcesses;i++) {
        MSG_RecvMBX[i]= NULL;
        * (unsigned int *) (MSG_LocalMailboxTable + 2*i)=0;
        * (unsigned int *) (MSG_LocalMailboxTable + 2*i +1)=0;
    }

    if (MSG_LocalPID == 0) {

        /* by convention, last to startup */
        /* send go ahead to all others */

        msg =  0x80000000 + MSG_CONST_INIT;
        for (i=2;i<MSG_NumProcesses;i++)
            if (! MSG_SendToMailbox(0, i, msg, FALSE))
                SYS_abort("Could not send INIT. Help!");
    } else {

        /* wait for INIT from PID 0 */
        /* Atomic test & Set */
        HWI_disable();
        if (MSG_RecvMBX[0] == NULL)
            MSG_RecvMBX[0] = MBX_create(4, MSG_MAX_SIMUL_RECV,& MBX_ATTRS);
        HWI_enable();

        MBX_pend(MSG_RecvMBX[0], & msg, SYS_FOREVER);

    }
}
```