```
/*
 * brd_init.c
 *
 * Todd Hiers
 *
 * Initializes all DSPs
 * Loads mpi.i0 program data file into all DSPs
 * Starts all DSPs
 *
 */

#include "c6205_lib.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

typedef struct
{
    WD_INTERRUPT Int;
    HANDLE hThread;
    WD_TRANSFER Trans[1];
    C6205_INT_HANDLER funcIntHandler;
} C6205_INT_INTERRUPT;

typedef struct
{
    DWORD index;
    DWORD dwMask;
    BOOL  fIsMemory;
    BOOL  fActive;
} C6205_ADDR_DESC;

typedef struct C6205_STRUCT
{
    HANDLE hWD;
    BOOL   fUseInt;
    C6205_INT_INTERRUPT Int;
    WD_PCI_SLOT pciSlot;
    C6205_ADDR_DESC addrDesc[C6205_ITEMS];
    WD_CARD_REGISTER cardReg;
} C6205_STRUCT;


    void RegisterWinDriver()
    {
        HANDLE hWD;
        WD_LICENSE lic;

        hWD = WD_Open();
        if (hWD!=INVALID_HANDLE_VALUE)
        {
            strcpy(lic.cLicense, "77f43e95087ec1cd118164d71689d5.Texas
Instruments");
            WD_License(hWD, &lic);
            WD_Close(hWD);
        }
    }



int asciitobyte (char * data) {
   int i = 0;

   i += 0x10 * (data[0] > 57 ? (data[0] - 55) : (data[0] - 48));
   i += (data[1] > 57 ? (data[1] - 55) : (data[1] - 48));

   return i;
```

```
}

unsigned int asciitoword (char * data) {
   unsigned int i = 0;

   i += 0x10000000 * (data[0] > 57 ? (data[0] - 55) : (data[0] - 48));
   i += 0x1000000 * (data[1] > 57 ? (data[1] - 55) : (data[1] - 48));
   i += 0x100000 * (data[2] > 57 ? (data[2] - 55) : (data[2] - 48));
   i += 0x10000 * (data[3] > 57 ? (data[3] - 55) : (data[3] - 48));
   i += 0x1000 * (data[4] > 57 ? (data[4] - 55) : (data[4] - 48));
   i += 0x100 * (data[5] > 57 ? (data[5] - 55) : (data[5] - 48));
   i += 0x10 * (data[6] > 57 ? (data[6] - 55) : (data[6] - 48));
   i += (data[7] > 57 ? (data[7] - 55) : (data[7] - 48));

   return i;
}

unsigned int asciitohword (char * data) {
   unsigned int i = 0;

   i += 0x1000 * (data[0] > 57 ? (data[0] - 55) : (data[0] - 48));
   i += 0x100 * (data[1] > 57 ? (data[1] - 55) : (data[1] - 48));
   i += 0x10 * (data[2] > 57 ? (data[2] - 55) : (data[2] - 48));
   i += 0x1 * (data[3] > 57 ? (data[3] - 55) : (data[3] - 48));

   return i;
}

int main(int argc, char *argv[])
{

   unsigned int i =0,j, k;
   BOOL fail = FALSE;
    C6205_HANDLE hC6205;
    HANDLE hWD;

   unsigned int size, type, page;
   unsigned int data, addr;
   FILE * fData;
   char line[255];


   unsigned int PCI_BAR0[64];
   unsigned int PCI_BAR1[64];

   unsigned int num_dsps;

    printf ("C6205 Multiprocesor board initialization utility.\n");
    printf ("Application accesses hardware using WinDriver.\n");

    // make sure WinDriver is loaded
   RegisterWinDriver();
    if (!PCI_Get_WD_handle(&hWD)) return 0;
    WD_Close (hWD);

   while (C6205_Open (&hC6205, C6205_DEFAULT_VENDOR_ID,
C6205_DEFAULT_DEVICE_ID, i, 0)) {

      printf("6205 #%i found and opened\n",i);

        C6205_WriteEMIF_SCTRL0(hC6205, 0x00000030);
      C6205_WriteEMIF_SDTR( hC6205, 0x0000800);
      C6205_WriteEMIF_SDCTRL( hC6205, 0x07116000);
      printf("\tInitilized EMIF registers\n");


        PCI_BAR0[i]= PCI_ReadBytes(hC6205->hWD, hC6205->pciSlot, 0x10, 4);
```

```
        PCI_BAR1[i]= PCI_ReadBytes(hC6205->hWD, hC6205->pciSlot, 0x14, 4);

      C6205_Close(hC6205);
      i++;
   }

   num_dsps = i;


   for (i=0;i<num_dsps;i++) {

      if (! C6205_Open (&hC6205, C6205_DEFAULT_VENDOR_ID,
C6205_DEFAULT_DEVICE_ID, i, 0))
         printf("open error %i\n",i);

      fData = fopen( "d:\\thesis\\messaging2\\mpi.i0", "r");
      if (fData == NULL)
         printf("Error: cannot open data file\n");

      addr = 0;
      data = 0;
      page = 0;

      while (fgets( line, 255, fData) != NULL){

         size= asciitobyte(line + 1);
         addr = (addr & 0xffff0000) | asciitohword(line + 3);
         type = asciitobyte(line + 7);

         if (type == 0) {

            for (k=0; k < size; k+=4){
               data = asciitoword(line + k * 2 + 9);
               //printf("%x : %x\n", addr + k, data);

               if ((addr + k) & 0xffc00000 != page) {
                  page = ((addr + k) & 0xffc00000) >> 22;
                  C6205_WriteDSPP(hC6205, page);
               }

               C6205_WriteDword(hC6205, 0, (addr + k) & 0x3fffff, data);

               if (C6205_ReadDword(hC6205, 0, (addr + k) & 0x3fffff) !=
data){
                  printf("ERROR on %i @ %x\n", i, (addr + k) & 0x3fffff);

               }

            }

         } else if (type == 4) {

            addr = 0x10000 * asciitohword(line+9);

         }

      }

      fclose(fData);
      C6205_Close(hC6205);
   }

   getch();

   for (i=0;i<num_dsps;i++) {
```

```
      if (! C6205_Open (&hC6205, C6205_DEFAULT_VENDOR_ID,
C6205_DEFAULT_DEVICE_ID, i, 0))
         printf("open error %i\n",i);

      C6205_WriteDword(hC6205, 0, 0x20, i); // MSG_LocalPID
      C6205_WriteDword(hC6205, 0, 0x24, num_dsps); // MSG_NumProcesses
      C6205_WriteDword(hC6205, 0, 0x30, 0); // MSG_LinkAddr
      C6205_WriteDword(hC6205, 0, 0x34, 0); // MSG_LinkCtrlAddr

      for (j=0;j<num_dsps; j++)
         C6205_WriteDword(hC6205, 0, 0x40 + (4*j), PCI_BAR0[j] & 0xfffffff0);
// MSG_RemoteMailbox
      for (j=0;j<num_dsps; j++)
         C6205_WriteDword(hC6205, 0, 0x80 + (4*j), (PCI_BAR1[j] & 0xfffffff0)
+0x27fff4); // MSG_RemoteInt (HDCR)
      for (j=0;j<num_dsps; j++)
         C6205_WriteByte(hC6205, 0, 0xc0 + j, 'b'); // MSG_RemoteType (All
on-board)
      C6205_WriteDSPP(hC6205, 1);
      printf("\tSet up MPI info\n");

      C6205_WriteHDCR(hC6205, 2);
      printf("\tBrought core out of reset\n");


      C6205_Close(hC6205);
   }


    return 0;
}
```

```
////////////////////////////////////////////////////////////////////
// File - c6205_lib.c
//
// Library for accessing the C6205 card.
// Code was generated by Driver Wizard.
// It accesses the hardware via WinDriver functions.
//
////////////////////////////////////////////////////////////////////

#include "c6205_lib.h"
#include "d:/thesis/windriver/include/windrvr_int_thread.h"
#ifndef __KERNEL__
   #include <stdio.h>
#endif

// this string is set to an error message, if one occurs
CHAR C6205_ErrorString[1024];

// internal data structures
typedef struct
{
    WD_INTERRUPT Int;
    HANDLE hThread;
    WD_TRANSFER Trans[1];
    C6205_INT_HANDLER funcIntHandler;
} C6205_INT_INTERRUPT;

typedef struct
{
    DWORD index;
    DWORD dwMask;
    BOOL  fIsMemory;
    BOOL  fActive;
} C6205_ADDR_DESC;

typedef struct C6205_STRUCT
{
    HANDLE hWD;
    BOOL   fUseInt;
    C6205_INT_INTERRUPT Int;
    WD_PCI_SLOT pciSlot;
    C6205_ADDR_DESC addrDesc[C6205_ITEMS];
    WD_CARD_REGISTER cardReg;
} C6205_STRUCT;

// internal function used by C6205_Open()
BOOL C6205_DetectCardElements(C6205_HANDLE hC6205);

DWORD C6205_CountCards (DWORD dwVendorID, DWORD dwDeviceID)
{
    WD_VERSION ver;
    WD_PCI_SCAN_CARDS pciScan;
    HANDLE hWD;

    C6205_ErrorString[0] = '\0';
    hWD = WD_Open();

    // check if handle valid & version OK
    if (hWD==INVALID_HANDLE_VALUE)
    {
        sprintf (C6205_ErrorString, "Failed opening WinDriver device\n");
        return 0;
    }

    BZERO(ver);
    WD_Version(hWD,&ver);
    if (ver.dwVer<WD_VER)
```

```
    {
        sprintf (C6205_ErrorString, "Incorrect WinDriver version\n");
        WD_Close (hWD);
        return 0;
    }

    BZERO(pciScan);
    pciScan.searchId.dwVendorId = dwVendorID;
    pciScan.searchId.dwDeviceId = dwDeviceID;
    WD_PciScanCards (hWD, &pciScan);
    WD_Close (hWD);
    if (pciScan.dwCards==0)
        sprintf (C6205_ErrorString, "no cards found\n");
    return pciScan.dwCards;
}

BOOL C6205_Open (C6205_HANDLE *phC6205, DWORD dwVendorID, DWORD dwDeviceID,
DWORD nCardNum, DWORD dwOptions)
{
    C6205_HANDLE hC6205 = (C6205_HANDLE) malloc (sizeof (C6205_STRUCT));

    WD_VERSION ver;
    WD_PCI_SCAN_CARDS pciScan;
    WD_PCI_CARD_INFO pciCardInfo;

    *phC6205 = NULL;
    C6205_ErrorString[0] = '\0';
    BZERO(*hC6205);

    hC6205->cardReg.hCard = 0;
    hC6205->hWD = WD_Open();

    // check if handle valid & version OK
    if (hC6205->hWD==INVALID_HANDLE_VALUE)
    {
        sprintf (C6205_ErrorString, "Failed opening WinDriver device\n");
        goto Exit;
    }

    BZERO(ver);
    WD_Version(hC6205->hWD,&ver);
    if (ver.dwVer<WD_VER)
    {
        sprintf (C6205_ErrorString, "Incorrect WinDriver version\n");
        goto Exit;
    }

    BZERO(pciScan);
    pciScan.searchId.dwVendorId = dwVendorID;
    pciScan.searchId.dwDeviceId = dwDeviceID;
    WD_PciScanCards (hC6205->hWD, &pciScan);
    if (pciScan.dwCards==0) // Found at least one card
    {
        sprintf (C6205_ErrorString, "Could not find PCI card\n");
        goto Exit;
    }
    if (pciScan.dwCards<=nCardNum)
    {
        sprintf (C6205_ErrorString, "Card out of range of available cards\n");
        goto Exit;
    }

    BZERO(pciCardInfo);
    pciCardInfo.pciSlot = pciScan.cardSlot[nCardNum];
    WD_PciGetCardInfo (hC6205->hWD, &pciCardInfo);
    hC6205->pciSlot = pciCardInfo.pciSlot;
    hC6205->cardReg.Card = pciCardInfo.Card;
```

```
    hC6205->fUseInt = (dwOptions & C6205_OPEN_USE_INT) ? TRUE : FALSE;
    if (!hC6205->fUseInt)
    {
        DWORD i;
        // Remove interrupt item if not needed
        for (i=0; i<hC6205->cardReg.Card.dwItems; i++)
        {
            WD_ITEMS *pItem = &hC6205->cardReg.Card.Item[i];
            if (pItem->item==ITEM_INTERRUPT)
                pItem->item = ITEM_NONE;
        }
    }
    else
    {
        DWORD i;
        // make interrupt resource sharable
        for (i=0; i<hC6205->cardReg.Card.dwItems; i++)
        {
            WD_ITEMS *pItem = &hC6205->cardReg.Card.Item[i];
            if (pItem->item==ITEM_INTERRUPT)
                pItem->fNotSharable = FALSE;
        }
    }

    hC6205->cardReg.fCheckLockOnly = FALSE;
    WD_CardRegister (hC6205->hWD, &hC6205->cardReg);
    if (hC6205->cardReg.hCard==0)
    {
        sprintf ( C6205_ErrorString, "Failed locking device.\n");
        goto Exit;
    }

    if (!C6205_DetectCardElements(hC6205))
    {
        sprintf ( C6205_ErrorString, "Card does not have all items expected
for C6205\n");
        goto Exit;
    }

    // Open finished OK
    *phC6205 = hC6205;
    return TRUE;

Exit:
    // Error during Open
    if (hC6205->cardReg.hCard)
        WD_CardUnregister(hC6205->hWD, &hC6205->cardReg);
    if (hC6205->hWD!=INVALID_HANDLE_VALUE)
        WD_Close(hC6205->hWD);
    free (hC6205);
    return FALSE;
}

void C6205_Close(C6205_HANDLE hC6205)
{
    // disable interrupts
    if (C6205_IntIsEnabled(hC6205))
        C6205_IntDisable(hC6205);

    // unregister card
    if (hC6205->cardReg.hCard)
        WD_CardUnregister(hC6205->hWD, &hC6205->cardReg);

    // close WinDriver
    WD_Close(hC6205->hWD);
```

```
    free (hC6205);
}

void C6205_WritePCIReg(C6205_HANDLE hC6205, DWORD dwReg, DWORD dwData)
{
    WD_PCI_CONFIG_DUMP pciCnf;

    BZERO(pciCnf);
    pciCnf.pciSlot = hC6205->pciSlot;
    pciCnf.pBuffer = &dwData;
    pciCnf.dwOffset = dwReg;
    pciCnf.dwBytes = 4;
    pciCnf.fIsRead = FALSE;
    WD_PciConfigDump(hC6205->hWD,&pciCnf);
}

DWORD C6205_ReadPCIReg(C6205_HANDLE hC6205, DWORD dwReg)
{
    WD_PCI_CONFIG_DUMP pciCnf;
    DWORD dwVal;

    BZERO(pciCnf);
    pciCnf.pciSlot = hC6205->pciSlot;
    pciCnf.pBuffer = &dwVal;
    pciCnf.dwOffset = dwReg;
    pciCnf.dwBytes = 4;
    pciCnf.fIsRead = TRUE;
    WD_PciConfigDump(hC6205->hWD,&pciCnf);
    return dwVal;
}

BOOL C6205_DetectCardElements(C6205_HANDLE hC6205)
{
    DWORD i;
    DWORD ad_sp;

    BZERO(hC6205->Int);
    BZERO(hC6205->addrDesc);

    for (i=0; i<hC6205->cardReg.Card.dwItems; i++)
    {
        WD_ITEMS *pItem = &hC6205->cardReg.Card.Item[i];

        switch (pItem->item)
        {
        case ITEM_MEMORY:
        case ITEM_IO:
            {
                DWORD dwBytes;
                DWORD dwPhysAddr;
                BOOL fIsMemory;
                if (pItem->item==ITEM_MEMORY)
                {
                    dwBytes = pItem->I.Mem.dwBytes;
                    dwPhysAddr = pItem->I.Mem.dwPhysicalAddr;
                    fIsMemory = TRUE;
                }
                else
                {
                    dwBytes = pItem->I.IO.dwBytes;
                    dwPhysAddr = pItem->I.IO.dwAddr;
                    fIsMemory = FALSE;
                }

                for (ad_sp=0; ad_sp<C6205_ITEMS; ad_sp++)
                {
                    DWORD dwPCIAddr;
```

```
                    DWORD dwPCIReg;

                    if (C6205_IsAddrSpaceActive(hC6205, (C6205_ADDR) ad_sp))
continue;
                    if (ad_sp<C6205_AD_EPROM) dwPCIReg = PCI_BAR0 + 4*ad_sp;
                    else dwPCIReg = PCI_ERBAR;
                    dwPCIAddr = C6205_ReadPCIReg(hC6205, dwPCIReg);
                    if (dwPCIAddr & 1)
                    {
                        if (fIsMemory) continue;
                        dwPCIAddr &= ~0x3;
                    }
                    else
                    {
                        if (!fIsMemory) continue;
                        dwPCIAddr &= ~0xf;
                    }
                    if (dwPCIAddr==dwPhysAddr)
                        break;
                }
                if (ad_sp<C6205_ITEMS)
                {
                    DWORD j;
                    hC6205->addrDesc[ad_sp].fActive = TRUE;
                    hC6205->addrDesc[ad_sp].index = i;
                    hC6205->addrDesc[ad_sp].fIsMemory = fIsMemory;
                    hC6205->addrDesc[ad_sp].dwMask = 0;
                    for (j=1; j<dwBytes && j!=0x80000000; j *= 2)
                    {
                        hC6205->addrDesc[ad_sp].dwMask =
                            (hC6205->addrDesc[ad_sp].dwMask << 1) | 1;
                    }
                }
            }
            break;
        case ITEM_INTERRUPT:
            if (hC6205->Int.Int.hInterrupt) return FALSE;
            hC6205->Int.Int.hInterrupt = pItem->I.Int.hInterrupt;
            break;
        }
    }

    // check that all the items needed were found
    // check if interrupt found
    if (hC6205->fUseInt && !hC6205->Int.Int.hInterrupt)
    {
        return FALSE;
    }

    // check that at least one memory space was found
    for (i = 0; i<C6205_ITEMS; i++)
        if (C6205_IsAddrSpaceActive(hC6205, (C6205_ADDR) i)) break;
    if (i==C6205_ITEMS) return FALSE;

    return TRUE;
}

BOOL C6205_IsAddrSpaceActive(C6205_HANDLE hC6205, C6205_ADDR addrSpace)
{
    return hC6205->addrDesc[addrSpace].fActive;
}

void C6205_GetPciSlot(C6205_HANDLE hC6205, WD_PCI_SLOT *pPciSlot)
{
    memcpy((PVOID)pPciSlot, &(hC6205->pciSlot), sizeof(WD_PCI_SLOT));
}
```

```
// General read/write function
void C6205_ReadWriteBlock(C6205_HANDLE hC6205, C6205_ADDR addrSpace, DWORD
dwOffset, BOOL fRead, PVOID buf, DWORD dwBytes, C6205_MODE mode)
{
    WD_TRANSFER trans;
    BOOL fMem = hC6205->addrDesc[addrSpace].fIsMemory;
    // safty check: is the address range active
    if (!C6205_IsAddrSpaceActive(hC6205, addrSpace)) return;
    BZERO(trans);
    if (fRead)
    {
        if (mode==C6205_MODE_BYTE) trans.cmdTrans = fMem ? RM_SBYTE :
RP_SBYTE;
        else if (mode==C6205_MODE_WORD) trans.cmdTrans = fMem ? RM_SWORD :
RP_SWORD;
        else if (mode==C6205_MODE_DWORD) trans.cmdTrans = fMem ? RM_SDWORD :
RP_SDWORD;
    }
    else
    {
        if (mode==C6205_MODE_BYTE) trans.cmdTrans = fMem ? WM_SBYTE :
WP_SBYTE;
        else if (mode==C6205_MODE_WORD) trans.cmdTrans = fMem ? WM_SWORD :
WP_SWORD;
        else if (mode==C6205_MODE_DWORD) trans.cmdTrans = fMem ? WM_SDWORD :
WP_SDWORD;
    }
    if (fMem)
        trans.dwPort =
hC6205->cardReg.Card.Item[hC6205->addrDesc[addrSpace].index].I.Mem.dwTransAddr
;
    else trans.dwPort =
hC6205->cardReg.Card.Item[hC6205->addrDesc[addrSpace].index].I.IO.dwAddr;
    trans.dwPort += dwOffset;

    trans.fAutoinc = TRUE;
    trans.dwBytes = dwBytes;
    trans.dwOptions = 0;
    trans.Data.pBuffer = buf;
    WD_Transfer (hC6205->hWD, &trans);
}

BYTE C6205_ReadByte (C6205_HANDLE hC6205, C6205_ADDR addrSpace, DWORD
dwOffset)
{
    BYTE data;
    if (hC6205->addrDesc[addrSpace].fIsMemory)
    {
        PBYTE pData = (PBYTE)
(hC6205->cardReg.Card.Item[hC6205->addrDesc[addrSpace].index].I.Mem.dwUserDire
ctAddr + dwOffset);
        data = *pData; // read from the memory mapped range directly
    }
    else C6205_ReadWriteBlock( hC6205, addrSpace, dwOffset, TRUE, &data,
sizeof (BYTE), C6205_MODE_BYTE);
    return data;
}

WORD C6205_ReadWord (C6205_HANDLE hC6205, C6205_ADDR addrSpace, DWORD
dwOffset)
{
    WORD data;
    if (hC6205->addrDesc[addrSpace].fIsMemory)
    {
        PWORD pData = (PWORD)
(hC6205->cardReg.Card.Item[hC6205->addrDesc[addrSpace].index].I.Mem.dwUserDire
ctAddr + dwOffset);
```

```
        data = *pData; // read from the memory mapped range directly
    }
    else C6205_ReadWriteBlock( hC6205, addrSpace, dwOffset, TRUE, &data,
sizeof (WORD), C6205_MODE_WORD);
    return data;
}

DWORD C6205_ReadDword (C6205_HANDLE hC6205, C6205_ADDR addrSpace, DWORD
dwOffset)
{
    DWORD data;
    if (hC6205->addrDesc[addrSpace].fIsMemory)
    {
        PDWORD pData = (PDWORD)
(hC6205->cardReg.Card.Item[hC6205->addrDesc[addrSpace].index].I.Mem.dwUserDire
ctAddr + dwOffset);
        data = *pData; // read from the memory mapped range directly
    }
    else C6205_ReadWriteBlock( hC6205, addrSpace, dwOffset, TRUE, &data,
sizeof (DWORD), C6205_MODE_DWORD);
    return data;
}

void C6205_WriteByte (C6205_HANDLE hC6205, C6205_ADDR addrSpace, DWORD
dwOffset, BYTE data)
{
    if (hC6205->addrDesc[addrSpace].fIsMemory)
    {
        PBYTE pData = (PBYTE)
(hC6205->cardReg.Card.Item[hC6205->addrDesc[addrSpace].index].I.Mem.dwUserDire
ctAddr + dwOffset);
        *pData = data; // write to the memory mapped range directly
    }
    else C6205_ReadWriteBlock( hC6205, addrSpace, dwOffset, FALSE, &data,
sizeof (BYTE), C6205_MODE_BYTE);
}

void C6205_WriteWord (C6205_HANDLE hC6205, C6205_ADDR addrSpace, DWORD
dwOffset, WORD data)
{
    if (hC6205->addrDesc[addrSpace].fIsMemory)
    {
        PWORD pData = (PWORD)
(hC6205->cardReg.Card.Item[hC6205->addrDesc[addrSpace].index].I.Mem.dwUserDire
ctAddr + dwOffset);
        *pData = data; // write to the memory mapped range directly
    }
    else C6205_ReadWriteBlock( hC6205, addrSpace, dwOffset, FALSE, &data,
sizeof (WORD), C6205_MODE_WORD);
}

void C6205_WriteDword (C6205_HANDLE hC6205, C6205_ADDR addrSpace, DWORD
dwOffset, DWORD data)
{
    if (hC6205->addrDesc[addrSpace].fIsMemory)
    {
        PDWORD pData = (PDWORD)
(hC6205->cardReg.Card.Item[hC6205->addrDesc[addrSpace].index].I.Mem.dwUserDire
ctAddr + dwOffset);
        *pData = data; // write to the memory mapped range directly
    }
    else C6205_ReadWriteBlock( hC6205, addrSpace, dwOffset, FALSE, &data,
sizeof (DWORD), C6205_MODE_DWORD);
}

DWORD C6205_ReadDSPP (C6205_HANDLE hC6205)
{
```

```
    return C6205_ReadDword(hC6205, (C6205_ADDR) C6205_DSPP_SPACE,
C6205_DSPP_OFFSET);
}

void C6205_WriteDSPP (C6205_HANDLE hC6205, DWORD data)
{
    C6205_WriteDword(hC6205, (C6205_ADDR) C6205_DSPP_SPACE, C6205_DSPP_OFFSET,
data);
}

DWORD C6205_ReadEMIF_SCTRL0 (C6205_HANDLE hC6205)
{
    return C6205_ReadDword(hC6205, (C6205_ADDR) C6205_EMIF_SCTRL0_SPACE,
C6205_EMIF_SCTRL0_OFFSET);
}

void C6205_WriteEMIF_SCTRL0 (C6205_HANDLE hC6205, DWORD data)
{
    C6205_WriteDword(hC6205, (C6205_ADDR) C6205_EMIF_SCTRL0_SPACE,
C6205_EMIF_SCTRL0_OFFSET, data);
}

DWORD C6205_ReadEMIF_GCTRL (C6205_HANDLE hC6205)
{
    return C6205_ReadDword(hC6205, (C6205_ADDR) C6205_EMIF_GCTRL_SPACE,
C6205_EMIF_GCTRL_OFFSET);
}

void C6205_WriteEMIF_GCTRL (C6205_HANDLE hC6205, DWORD data)
{
    C6205_WriteDword(hC6205, (C6205_ADDR) C6205_EMIF_GCTRL_SPACE,
C6205_EMIF_GCTRL_OFFSET, data);
}

DWORD C6205_ReadEMIF_SDTR (C6205_HANDLE hC6205)
{
    return C6205_ReadDword(hC6205, (C6205_ADDR) C6205_EMIF_SDTR_SPACE,
C6205_EMIF_SDTR_OFFSET);
}

void C6205_WriteEMIF_SDTR (C6205_HANDLE hC6205, DWORD data)
{
    C6205_WriteDword(hC6205, (C6205_ADDR) C6205_EMIF_SDTR_SPACE,
C6205_EMIF_SDTR_OFFSET, data);
}

DWORD C6205_ReadEMIF_SDCTRL (C6205_HANDLE hC6205)
{
    return C6205_ReadDword(hC6205, (C6205_ADDR) C6205_EMIF_SDCTRL_SPACE,
C6205_EMIF_SDCTRL_OFFSET);
}

void C6205_WriteEMIF_SDCTRL (C6205_HANDLE hC6205, DWORD data)
{
    C6205_WriteDword(hC6205, (C6205_ADDR) C6205_EMIF_SDCTRL_SPACE,
C6205_EMIF_SDCTRL_OFFSET, data);
}

DWORD C6205_ReadHDCR (C6205_HANDLE hC6205)
{
    return C6205_ReadDword(hC6205, (C6205_ADDR) C6205_HDCR_SPACE,
C6205_HDCR_OFFSET);
}

void C6205_WriteHDCR (C6205_HANDLE hC6205, DWORD data)
{
```

```
    C6205_WriteDword(hC6205, (C6205_ADDR) C6205_HDCR_SPACE, C6205_HDCR_OFFSET,
data);
}

DWORD C6205_ReadHSR (C6205_HANDLE hC6205)
{
    return C6205_ReadDword(hC6205, (C6205_ADDR) C6205_HSR_SPACE,
C6205_HSR_OFFSET);
}

void C6205_WriteHSR (C6205_HANDLE hC6205, DWORD data)
{
    C6205_WriteDword(hC6205, (C6205_ADDR) C6205_HSR_SPACE, C6205_HSR_OFFSET,
data);
}

BOOL C6205_IntIsEnabled (C6205_HANDLE hC6205)
{
    if (!hC6205->fUseInt) return FALSE;
    if (!hC6205->Int.hThread) return FALSE;
    return TRUE;
}

void C6205_IntHandler (PVOID pData)
{
    C6205_HANDLE hC6205 = (C6205_HANDLE) pData;
    C6205_INT_RESULT intResult;
    intResult.dwCounter = hC6205->Int.Int.dwCounter;
    intResult.dwLost = hC6205->Int.Int.dwLost;
    intResult.fStopped = hC6205->Int.Int.fStopped;
    hC6205->Int.funcIntHandler(hC6205, &intResult);
}

BOOL C6205_IntEnable (C6205_HANDLE hC6205, C6205_INT_HANDLER funcIntHandler)
{
    C6205_ADDR addrSpace;

    if (!hC6205->fUseInt) return FALSE;
    // check if interrupt is already enabled
    if (hC6205->Int.hThread) return FALSE;

    BZERO(hC6205->Int.Trans);
    // This is a sample of handling interrupts:
    // One transfer command is issued to CANCEL the source of the interrupt,
    // otherwise, the PC will hang when an interrupt occurs!
    // You will need to modify this code to fit your specific hardware.
    addrSpace = C6205_AD_BAR0; // put the address space of the register here
    if (hC6205->addrDesc[addrSpace].fIsMemory)
    {
        hC6205->Int.Trans[0].dwPort =
hC6205->cardReg.Card.Item[hC6205->addrDesc[addrSpace].index].I.Mem.dwTransAddr
;
        hC6205->Int.Trans[0].cmdTrans = WM_DWORD;
    }
    else
    {
        hC6205->Int.Trans[0].dwPort =
hC6205->cardReg.Card.Item[hC6205->addrDesc[addrSpace].index].I.IO.dwAddr;
        hC6205->Int.Trans[0].cmdTrans = WP_DWORD;
    }
    hC6205->Int.Trans[0].dwPort += 0; // put the offset of the register from
the beginning of the address space here
    hC6205->Int.Trans[0].Data.Dword = 0x0;
    hC6205->Int.Int.dwCmds = 1;
    hC6205->Int.Int.Cmd = hC6205->Int.Trans;
    hC6205->Int.Int.dwOptions |= INTERRUPT_CMD_COPY;
```

```
    // this calls WD_IntEnable() and creates an interrupt handler thread
    hC6205->Int.funcIntHandler = funcIntHandler;
    if (!InterruptThreadEnable(&hC6205->Int.hThread, hC6205->hWD,
&hC6205->Int.Int, C6205_IntHandler, (PVOID) hC6205))
        return FALSE;

    return TRUE;
}

void C6205_IntDisable (C6205_HANDLE hC6205)
{
    if (!hC6205->fUseInt) return;
    if (!hC6205->Int.hThread) return;

    // this calls WD_IntDisable()
    InterruptThreadDisable(hC6205->Int.hThread);

    hC6205->Int.hThread = NULL;
}
```