```
/**
*
*  BTICard PowerPC Linux Driver  Version 1.3.4  (09/25/2006)
*  Copyright (c) 2003-2006
*  Ballard Technology, Inc.
*  3229A Pine Street
*  Everett WA 98201
*  (800) 829-1553
*  (425) 339-0281
*  (425) 339-0915 FAX
*  ALL RIGHTS RESERVED
*
*  NAME:   BTICard.H -- PowerPC Linux
*                       BTICard Driver Include Header File.
*
**/

/**
*
*  This file defines the procedures provided by the PowerPC Linux
*  Driver Library for Ballard Technology interface cards.
*  Applications using the BTICard Driver Library must
*  incorporate this include file using the preprocessor directive
*  #include. If this file is in the current working directory,
*  the form would be:
*
*  #include "BTICard.H"
*
**/

/**
*
*  Conditional block to prevent multiple defines.
*
**/

#ifndef __BTICard_H
#define __BTICard_H

/**
*
*  Typedefs used by the BTICard Driver.
*
**/

#ifndef BTICardAPI
#define BTICardAPI
#endif

#ifndef VOID
#define VOID void
#endif

#ifndef LPVOID
#define LPVOID void *
#endif

#ifndef INT
#define INT int
#endif

#ifndef LPINT
#define LPINT int *
#endif

#ifndef BYTE
#define BYTE unsigned char
#endif
```

```
#ifndef LPBYTE
#define LPBYTE unsigned char *
#endif

#ifndef USHORT
#define USHORT unsigned short
#endif

#ifndef LPUSHORT
#define LPUSHORT unsigned short *
#endif

#ifndef ULONG
#define ULONG unsigned long
#endif

#ifndef LPULONG
#define LPULONG unsigned long *
#endif

#ifndef MSGADDR
#define MSGADDR unsigned long
#endif

#ifndef BASEADDR
#define BASEADDR unsigned long
#endif

#ifndef LISTADDR
#define LISTADDR unsigned long
#endif

#ifndef LPMSGADDR
#define LPMSGADDR unsigned long *
#endif

#ifndef LPCSTR
#define LPCSTR const char *
#endif

#ifndef LPSTR
#define LPSTR char *
#endif

#ifndef CHAR
#define CHAR char
#endif

#ifndef BOOL
#define BOOL int
#endif

#ifndef HCARD
#define HCARD int
#endif

#ifndef LPHCARD
#define LPHCARD int *
#endif

#ifndef HCORE
#define HCORE int
#endif

#ifndef LPHCORE
#define LPHCORE int *
#endif

#ifndef HRPC
```

```
#define HRPC int
#endif

#ifndef LPHRPC
#define LPHRPC int *
#endif

#ifndef ERRVAL
#define ERRVAL int
#endif

#ifndef SCHNDX
#define SCHNDX int
#endif

/**
*
*  Structs used by the BTICard Driver.
*
**/

#ifndef SEQRECORD1553
typedef struct
{
        USHORT  type;               //Valid in all versions
        USHORT  count;              //Valid in all versions
        ULONG   timestamp;          //Valid in all versions
        USHORT  activity;           //Valid in all versions
        USHORT  error;              //Valid in all versions
        USHORT  cwd1;               //Valid in all versions
        USHORT  cwd2;               //Valid in all versions
        USHORT  swd1;               //Valid in all versions
        USHORT  swd2;               //Valid in all versions
        USHORT  datacount;          //Valid in all versions
        USHORT  data[40];           //Variable length (don't exceed data[datacount-1])
} SEQRECORD1553;
#endif

#ifndef LPSEQRECORD1553
typedef SEQRECORD1553 * LPSEQRECORD1553;
#endif

#ifndef SEQRECORDMORE1553
typedef struct
{
        ULONG   timestamph;         //Valid if version of base record (SEQRECORD1553) >= 1
        USHORT  resptime1;          //Valid if version of base record (SEQRECORD1553) >= 1
        USHORT  resptime2;          //Valid if version of base record (SEQRECORD1553) >= 1
} SEQRECORDMORE1553;
#endif

#ifndef LPSEQRECORDMORE1553
typedef SEQRECORDMORE1553 * LPSEQRECORDMORE1553;
#endif

#ifndef SEQRECORD429
typedef struct
{
        USHORT  type;               //Valid in all versions
        USHORT  count;              //Valid in all versions
        ULONG   timestamp;          //Valid in all versions
        USHORT  activity;           //Valid in all versions
        USHORT  decgap;             //Valid if version >= 1
        ULONG   data;               //Valid in all versions
        ULONG   timestamph;         //Valid if version >= 1
} SEQRECORD429;
#endif

#ifndef LPSEQRECORD429
```

```
typedef SEQRECORD429 * LPSEQRECORD429;
#endif

#ifndef SEQRECORD717
typedef struct
{
        USHORT  type;               //Valid in all versions
        USHORT  count;              //Valid in all versions
        ULONG   timestamp;          //Valid in all versions
        USHORT  activity;           //Valid in all versions
        USHORT  wordnum;            //Valid in all versions
        USHORT  subframe;           //Valid in all versions
        USHORT  superframe;         //Valid in all versions
        USHORT  data;               //Valid in all versions
        USHORT  rsvd9;              //Valid if version >= 1
        ULONG   timestamph;         //Valid if version >= 1
} SEQRECORD717;
#endif

#ifndef LPSEQRECORD717
typedef SEQRECORD717 * LPSEQRECORD717;
#endif

#ifndef SEQRECORD708
typedef struct
{
        USHORT  type;               //Valid in all versions
        USHORT  count;              //Valid in all versions
        ULONG   timestamp;          //Valid in all versions
        USHORT  activity;           //Valid in all versions
        USHORT  datacount;          //Valid in all versions
        USHORT  data[100];          //Valid in all versions
        USHORT  extra[16];          //Valid if version >= 1
        USHORT  bitcount;           //Valid if version >= 1
        USHORT  rsvd123;            //Valid if version >= 1
        ULONG   timestamph;         //Valid if version >= 1
} SEQRECORD708;
#endif

#ifndef LPSEQRECORD708
typedef SEQRECORD708 * LPSEQRECORD708;
#endif

#ifndef SEQRECORDCSDB
typedef struct
{
        USHORT  type;               //Valid in all versions
        USHORT  count;              //Valid in all versions
        ULONG   timestamp;          //Valid in all versions
        ULONG   timestamph;         //Valid in all versions
        USHORT  activity;           //Valid in all versions
        USHORT  datacount;          //Valid in all versions
        USHORT  data[32];           //Valid in all versions
} SEQRECORDCSDB;
#endif

#ifndef LPSEQRECORDCSDB
typedef SEQRECORDCSDB * LPSEQRECORDCSDB;
#endif

#ifndef SEQRECORDDIO
typedef struct
{
        USHORT  type;               //Valid in all versions
        USHORT  count;              //Valid in all versions
        USHORT  bank;               //Valid in all versions
        USHORT  state;              //Valid in all versions
        ULONG   timestamp;          //Valid in all versions
        ULONG   timestamph;         //Valid in all versions
```

```
} SEQRECORDDIO;
#endif

#ifndef LPSEQRECORDDIO
typedef SEQRECORDDIO * LPSEQRECORDDIO;
#endif

#ifndef SEQFINDINFO
typedef struct
{
        LPUSHORT pRecFirst;
        LPUSHORT pRecNext;
        LPUSHORT pRecLast;
} SEQFINDINFO;
#endif

#ifndef LPSEQFINDINFO
typedef SEQFINDINFO * LPSEQFINDINFO;
#endif

#ifndef BTIIRIGTIME
typedef struct
{
        USHORT  days;
        USHORT  hours;
        USHORT  min;
        USHORT  sec;
        USHORT  msec;
        USHORT  usec;
} BTIIRIGTIME;
#endif

#ifndef LPBTIIRIGTIME
typedef BTIIRIGTIME * LPBTIIRIGTIME;
#endif

#ifndef BTIIDENTIFY
typedef struct
{
        CHAR mac_address[20];
        CHAR serial_number[8];
        CHAR card_string[32];
        CHAR type_string[32];
        CHAR user_string[64];
} BTIIDENTIFY;
#endif

#ifndef LPBTIIDENTIFY
typedef BTIIDENTIFY * LPBTIIDENTIFY;
#endif

/**
*
*  "C" block if compiling a C++ file.
*
**/

#ifdef __cplusplus
extern "C" {
#endif

/**
*
*  BTICard Driver functions.
*
**/

BTICardAPI ULONG BTICard_AddrDSP(ULONG addr,HCARD handleval);
BTICardAPI ULONG BTICard_AddrHost(ULONG addr,HCARD handleval);
BTICardAPI ERRVAL BTICard_AsciiToMant(LPCSTR str,LPULONG mant,LPINT exp);
BTICardAPI ULONG BTICard_BCDToBin(ULONG bcdval,INT msb,INT lsb);
BTICardAPI ULONG BTICard_BinToBCD(ULONG oldbcdval,ULONG binval,INT msb,INT lsb);
BTICardAPI ERRVAL BTICard_CardClose(HCARD handleval);
BTICardAPI ERRVAL BTICard_CardCloseAll(VOID);
BTICardAPI ULONG BTICard_CardGetInfo(USHORT infotype,INT channum,HCARD handleval);
BTICardAPI ERRVAL BTICard_CardGetInfoEx(LPUSHORT bufmodel,USHORT bufmodelcount,LPUSHORT
buffeature,USHORT buffeaturecount,HCARD handleval);
BTICardAPI BOOL BTICard_CardIsRunning(HCARD handleval);
BTICardAPI ERRVAL BTICard_CardNop(HCARD handleval);
BTICardAPI ERRVAL BTICard_CardOpen(LPHCARD lphCard,INT cardnum);
BTICardAPI LPCSTR BTICard_CardProductStr(HCARD handleval);
BTICardAPI VOID BTICard_CardReset(HCARD handleval);
BTICardAPI VOID BTICard_CardResetEx(HCARD handleval);
BTICardAPI ERRVAL BTICard_CardResume(HCARD handleval);
BTICardAPI ERRVAL BTICard_CardStart(HCARD handleval);
BTICardAPI BOOL BTICard_CardStop(HCARD handleval);
BTICardAPI VOID BTICard_CardSyncEnable(BOOL enableflag,USHORT syncmask,USHORT pinpolarity,HCARD
handleval);
BTICardAPI USHORT BTICard_CardSyncValid(HCARD handleval);
BTICardAPI ERRVAL BTICard_CardTest(USHORT level,HCARD handleval);
BTICardAPI ERRVAL BTICard_CardTest0(HCARD handleval);
BTICardAPI ERRVAL BTICard_CardTest1(HCARD handleval);
BTICardAPI ERRVAL BTICard_CardTest2(HCARD handleval);
BTICardAPI ERRVAL BTICard_CardTest3(HCARD handleval);
BTICardAPI VOID BTICard_CardTrigger(HCARD handleval);
BTICardAPI VOID BTICard_CardTriggerEnable(BOOL enableflag,HCARD handleval);
BTICardAPI VOID BTICard_CardTriggerEnableEx(BOOL enableflag,USHORT trigmask,USHORT
pinpolarity,HCARD handleval);
BTICardAPI VOID BTICard_CardTriggerEx(USHORT trigmask,HCARD handleval);
BTICardAPI USHORT BTICard_CardTriggerValid(HCARD handleval);
BTICardAPI LPCSTR BTICard_CardTypeStr(HCARD handleval);
BTICardAPI VOID BTICard_ChDARClr(USHORT maskval,USHORT addrval,INT channum,HCARD handleval);
BTICardAPI BOOL BTICard_ChDARGet(USHORT maskval,USHORT addrval,INT channum,HCARD handleval);
BTICardAPI ULONG BTICard_ChDARRdL(USHORT addrval,INT channum,HCARD handleval);
BTICardAPI VOID BTICard_ChDARRdsW(LPUSHORT valueptr,USHORT addrval,INT countval,INT channum,HCARD
handleval);
BTICardAPI USHORT BTICard_ChDARRdW(USHORT addrval,INT channum,HCARD handleval);
BTICardAPI VOID BTICard_ChDARSet(USHORT maskval,USHORT addrval,INT channum,HCARD handleval);
BTICardAPI VOID BTICard_ChDARWrL(ULONG value,USHORT addrval,INT channum,HCARD handleval);
BTICardAPI VOID BTICard_ChDARWrsW(LPUSHORT valueptr,USHORT addrval,INT countval,INT channum,HCARD
handleval);
BTICardAPI VOID BTICard_ChDARWrW(USHORT value,USHORT addrval,INT channum,HCARD handleval);
BTICardAPI ERRVAL BTICard_CISRd(LPUSHORT buf,USHORT bufcount,INT cistype,HCARD handleval);
BTICardAPI ERRVAL BTICard_CISWr(LPUSHORT buf,USHORT bufcount,INT cistype,HCARD handleval);
BTICardAPI USHORT BTICard_CommBufRd(USHORT offset,HCARD handleval);
BTICardAPI VOID BTICard_CommBufWr(USHORT value,USHORT offset,HCARD handleval);
BTICardAPI ERRVAL BTICard_CommCall(ULONG addrval,HCARD handleval);
BTICardAPI ERRVAL BTICard_CommChannelReconfig(ULONG chmask,HCARD handleval);
BTICardAPI ERRVAL BTICard_CommCheck(HCARD handleval);
BTICardAPI ERRVAL BTICard_CommDisable(USHORT command,HCARD handleval);
BTICardAPI ERRVAL BTICard_CommDisableEx(USHORT command,HCARD handleval);
BTICardAPI ERRVAL BTICard_CommEnable(HCARD handleval);
BTICardAPI ERRVAL BTICard_CommExternSRQ(ULONG chmask,HCARD handleval);
BTICardAPI ERRVAL BTICard_CommFillW(USHORT value,ULONG addrval,USHORT count,HCARD handleval);
BTICardAPI BOOL BTICard_CommProtocolFunc(USHORT opcode,USHORT argcount,LPUSHORT argbuf,HCARD
handleval);
BTICardAPI ULONG BTICard_CommRdL(ULONG addrval,HCARD handleval);
BTICardAPI ERRVAL BTICard_CommRdsW(LPUSHORT valueptr,ULONG addrval,USHORT count,HCARD handleval);
BTICardAPI USHORT BTICard_CommRdW(ULONG addrval,HCARD handleval);
BTICardAPI ERRVAL BTICard_CommRun(LPUSHORT dataptr,USHORT datacount,LPUSHORT codeptr,USHORT
codecount,HCARD handleval);
BTICardAPI VOID BTICard_CommWrL(ULONG value,ULONG addrval,HCARD handleval);
BTICardAPI ERRVAL BTICard_CommWrsW(LPUSHORT valueptr,ULONG addrval,USHORT count,HCARD handleval);
BTICardAPI VOID BTICard_CommWrW(USHORT value,ULONG addrval,HCARD handleval);
BTICardAPI ERRVAL BTICard_CoProcGetInfo(LPULONG valueptr,USHORT infotype,HCARD handleval);
BTICardAPI ERRVAL BTICard_CoreOpen(LPHCORE lphCore,INT corenum,HCARD hCard);
BTICardAPI VOID BTICard_DARClr(USHORT maskval,USHORT addrval,HCARD handleval);
BTICardAPI BOOL BTICard_DARGet(USHORT maskval,USHORT addrval,HCARD handleval);
BTICardAPI ULONG BTICard_DARRdL(USHORT addrval,HCARD handleval);
BTICardAPI USHORT BTICard_DARRdW(USHORT addrval,HCARD handleval);
BTICardAPI VOID BTICard_DARSet(USHORT maskval,USHORT addrval,HCARD handleval);
BTICardAPI VOID BTICard_DARWrL(ULONG value,USHORT addrval,HCARD handleval);
BTICardAPI VOID BTICard_DARWrW(USHORT value,USHORT addrval,HCARD handleval);
BTICardAPI ULONG BTICard_Div(ULONG diva,ULONG divb);
BTICardAPI ERRVAL BTICard_DspBioClear(HCARD handleval);
BTICardAPI BOOL BTICard_DspBioRd(HCARD handleval);
BTICardAPI ERRVAL BTICard_DspBioSet(HCARD handleval);
BTICardAPI ERRVAL BTICard_DspIntfClear(USHORT intmask,HCARD handleval);
BTICardAPI BOOL BTICard_DspIntfRd(USHORT intmask,HCARD handleval);
BTICardAPI ERRVAL BTICard_DspIntmClear(HCARD handleval);
BTICardAPI ERRVAL BTICard_DspIntmSet(HCARD handleval);
BTICardAPI ERRVAL BTICard_DspXfClear(HCARD handleval);
BTICardAPI BOOL BTICard_DspXfRd(HCARD handleval);
BTICardAPI ERRVAL BTICard_DspXfSet(HCARD handleval);
BTICardAPI LPCSTR BTICard_ErrDesc(ERRVAL errval,HCARD handleval);
BTICardAPI LPCSTR BTICard_ErrDescStr(ERRVAL errval,HCARD handleval);
BTICardAPI LPCSTR BTICard_ErrName(ERRVAL errval,HCARD handleval);
BTICardAPI ERRVAL BTICard_EventLogClear(HCARD handleval);
BTICardAPI ERRVAL BTICard_EventLogConfig(USHORT configval,USHORT count,HCARD handleval);
BTICardAPI ULONG BTICard_EventLogRd(LPUSHORT typeval,LPULONG infoval,LPINT channel,HCARD
handleval);
BTICardAPI INT BTICard_EventLogStatus(HCARD handleval);
BTICardAPI VOID BTICard_ExpandMant(LPULONG mant,LPINT exp);
BTICardAPI BOOL BTICard_ExtDinRd(HCARD handleval);
BTICardAPI VOID BTICard_ExtDinWr(BOOL dinval,HCARD handleval);
BTICardAPI VOID BTICard_ExtDIODirSet(INT dionum,BOOL dirval,HCARD handleval);
BTICardAPI BOOL BTICard_ExtDIORd(INT dionum,HCARD handleval);
BTICardAPI VOID BTICard_ExtDIOWr(INT dionum,BOOL dioval,HCARD handleval);
BTICardAPI BOOL BTICard_ExtLEDRd(HCARD handleval);
BTICardAPI VOID BTICard_ExtLEDWr(BOOL ledval,HCARD handleval);
BTICardAPI VOID BTICard_ExtStatusLEDRd(LPINT ledon,LPINT ledcolor,HCARD handleval);
BTICardAPI VOID BTICard_ExtStatusLEDWr(BOOL ledon,BOOL ledcolor,HCARD handleval);
BTICardAPI USHORT BTICard_GetHigh(ULONG val);
BTICardAPI USHORT BTICard_GetLow(ULONG val);
BTICardAPI USHORT BTICard_GlobalRdW(USHORT addrval,HCARD handleval);
BTICardAPI VOID BTICard_GlobalWrW(USHORT value,USHORT addrval,HCARD handleval);
BTICardAPI ERRVAL BTICard_HandleIgnoreSet(BOOL flag,HCARD handleval);
BTICardAPI ERRVAL BTICard_HandleInfo(LPSTR cardstr,LPINT cardnum,LPULONG sizval,LPVOID
*vxdptr,HCARD handleval);
BTICardAPI ERRVAL BTICard_HandleInfoEx(LPULONG valueptr,ULONG type,HCARD handleval);
BTICardAPI BOOL BTICard_HandleIsCard(HCARD handleval);
BTICardAPI BOOL BTICard_HandleIsCore(HCORE handleval);
BTICardAPI ERRVAL BTICard_HandleMakeCard(LPHCARD lphCard,LPINT lpcorenum,HCORE hCore);
BTICardAPI ERRVAL BTICard_HandleMakeCore(LPHCORE lphCore,INT corenum,HCARD hCard);
BTICardAPI ERRVAL BTICard_HandleMakeRPC(LPHCARD hCard_Remote,LPHRPC lphRPC,HCARD handleval);
BTICardAPI BOOL BTICard_HandleOkay(HCARD handleval);
BTICardAPI ULONG BTICard_HeapAlloc(INT section,ULONG wordcount,HCARD handleval);
BTICardAPI ULONG BTICard_HeapAllocAll(INT section,LPULONG wordcount,HCARD handleval);
BTICardAPI ULONG BTICard_HeapAllocEx(USHORT configval,INT section,ULONG wordcount,HCARD
handleval);
BTICardAPI ULONG BTICard_HeapWipe(INT section,HCARD handleval);
BTICardAPI VOID BTICard_HookProtocolFunc(INT index,ERRVAL (* ptr)(INT msgval,LPVOID lpParam,HCARD
handleval));
BTICardAPI VOID BTICard_HPIFill(USHORT value,USHORT addrval,INT countval,HCARD handleval);
BTICardAPI ULONG BTICard_HPIRdL(USHORT addrval,HCARD handleval);
BTICardAPI VOID BTICard_HPIRdsL(LPULONG valueptr,USHORT addrval,INT countval,HCARD handleval);
BTICardAPI VOID BTICard_HPIRdsW(LPUSHORT valueptr,USHORT addrval,INT countval,HCARD handleval);
BTICardAPI USHORT BTICard_HPIRdW(USHORT addrval,HCARD handleval);
BTICardAPI VOID BTICard_HPIWrL(ULONG value,USHORT addrval,HCARD handleval);
BTICardAPI VOID BTICard_HPIWrsL(LPULONG valueptr,USHORT addrval,INT countval,HCARD handleval);
BTICardAPI VOID BTICard_HPIWrsW(LPUSHORT valueptr,USHORT addrval,INT countval,HCARD handleval);
BTICardAPI VOID BTICard_HPIWrW(USHORT value,USHORT addrval,HCARD handleval);
BTICardAPI ERRVAL BTICard_HWMonFault(HCARD handleval);
BTICardAPI ERRVAL BTICard_HWMonStatus(LPINT status,HCARD handleval);
BTICardAPI ERRVAL BTICard_Identify(LPBTIIDENTIFY info,HCARD handleval);
BTICardAPI USHORT BTICard_IDRegRd(INT gate_array_num,HCARD handleval);
BTICardAPI VOID BTICard_IDRegWr(USHORT value,INT gate_array_num,HCARD handleval);
BTICardAPI VOID BTICard_IntClear(HCARD handleval);
BTICardAPI ERRVAL BTICard_IntDisable(HCARD handleval);
BTICardAPI ERRVAL BTICard_IntEnable(HCARD handleval);
BTICardAPI ERRVAL BTICard_IntEnableCond(HCARD handleval);
BTICardAPI LPVOID BTICard_IntGet(HCARD handleval);
BTICardAPI ERRVAL BTICard_IntInstall(LPVOID hEvent,HCARD handleval);
BTICardAPI VOID BTICard_IntReset(HCARD handleval);
BTICardAPI ERRVAL BTICard_IntUninstall(HCARD handleval);
BTICardAPI ULONG BTICard_IORdL(INT addrval,HCARD handleval);
BTICardAPI USHORT BTICard_IORdW(INT addrval,HCARD handleval);
BTICardAPI USHORT BTICard_IOWINRdW(INT addrval,HCARD handleval);
BTICardAPI VOID BTICard_IOWINWrW(USHORT value,INT addrval,HCARD handleval);
BTICardAPI VOID BTICard_IOWrL(ULONG value,INT addrval,HCARD handleval);
BTICardAPI VOID BTICard_IOWrW(USHORT value,INT addrval,HCARD handleval);
BTICardAPI ERRVAL BTICard_IRIGConfig(ULONG configval,HCARD handleval);
BTICardAPI ULONG BTICard_IRIGFieldGetDays(ULONG irigvalh,ULONG irigvall);
BTICardAPI ULONG BTICard_IRIGFieldGetHours(ULONG irigvalh,ULONG irigvall);
BTICardAPI ULONG BTICard_IRIGFieldGetMicrosec(ULONG irigvalh,ULONG irigvall);
BTICardAPI ULONG BTICard_IRIGFieldGetMillisec(ULONG irigvalh,ULONG irigvall);
BTICardAPI ULONG BTICard_IRIGFieldGetMin(ULONG irigvalh,ULONG irigvall);
BTICardAPI ULONG BTICard_IRIGFieldGetSec(ULONG irigvalh,ULONG irigvall);
BTICardAPI VOID BTICard_IRIGFieldPutDays(ULONG value,LPULONG irigvalh,LPULONG irigvall);
BTICardAPI VOID BTICard_IRIGFieldPutHours(ULONG value,LPULONG irigvalh,LPULONG irigvall);
BTICardAPI VOID BTICard_IRIGFieldPutMicrosec(ULONG value,LPULONG irigvalh,LPULONG irigvall);
BTICardAPI VOID BTICard_IRIGFieldPutMillisec(ULONG value,LPULONG irigvalh,LPULONG irigvall);
BTICardAPI VOID BTICard_IRIGFieldPutMin(ULONG value,LPULONG irigvalh,LPULONG irigvall);
BTICardAPI VOID BTICard_IRIGFieldPutSec(ULONG value,LPULONG irigvalh,LPULONG irigvall);
BTICardAPI ERRVAL BTICard_IRIGRd(LPBTIIRIGTIME irigtime,HCARD handleval);
BTICardAPI ERRVAL BTICard_IRIGRdEx(LPUSHORT timebuf,HCARD handleval);
BTICardAPI BOOL BTICard_IRIGSyncStatus(HCARD handleval);
BTICardAPI ERRVAL BTICard_IRIGWr(LPBTIIRIGTIME irigtime,HCARD handleval);
BTICardAPI ERRVAL BTICard_IRIGWrEx(LPUSHORT timebuf,HCARD handleval);
BTICardAPI ULONG BTICard_MakeLong(USHORT valh,USHORT vall);
BTICardAPI USHORT BTICard_MakeWord(BYTE valh,BYTE vall);
BTICardAPI LPSTR BTICard_MantToAscii(LPSTR buf,long mant,int exp);
BTICardAPI ULONG BTICard_Mask(ULONG dataval,USHORT cntval);
BTICardAPI VOID BTICard_MaxMant(LPULONG mant,LPINT exp);
BTICardAPI ULONG BTICard_Mod(ULONG moda,ULONG modb);
BTICardAPI ULONG BTICard_Mul(ULONG mula,ULONG mulb);
BTICardAPI VOID BTICard_NormalMant(LPULONG mant,LPINT exp);
BTICardAPI USHORT BTICard_PortRd(INT addrval,HCARD handleval);
BTICardAPI VOID BTICard_PortWr(USHORT value,INT addrval,HCARD handleval);
BTICardAPI USHORT BTICard_ProgRdW(ULONG addrval,HCARD handleval);
BTICardAPI VOID BTICard_ProgWrW(USHORT value,ULONG addrval,HCARD handleval);
BTICardAPI VOID BTICard_RAMFill(USHORT value,ULONG addrval,ULONG countval,HCARD handleval);
BTICardAPI USHORT BTICard_RAMRdB(ULONG addrval,HCARD handleval);
BTICardAPI ULONG BTICard_RAMRdL(ULONG addrval,HCARD handleval);
BTICardAPI VOID BTICard_RAMRdmL(LPULONG valueptr,LPULONG addrptr,INT countval,HCARD handleval);
BTICardAPI VOID BTICard_RAMRdmW(LPUSHORT valueptr,LPULONG addrptr,INT countval,HCARD handleval);
BTICardAPI VOID BTICard_RAMRdsL(LPULONG valueptr,ULONG addrval,INT countval,HCARD handleval);
BTICardAPI VOID BTICard_RAMRdsW(LPUSHORT valueptr,ULONG addrval,INT countval,HCARD handleval);
BTICardAPI USHORT BTICard_RAMRdW(ULONG addrval,HCARD handleval);
BTICardAPI VOID BTICard_RAMWipe(HCARD handleval);
BTICardAPI VOID BTICard_RAMWipeEx(USHORT value,HCARD handleval);
BTICardAPI VOID BTICard_RAMWrB(USHORT value,ULONG addrval,HCARD handleval);
BTICardAPI VOID BTICard_RAMWrL(ULONG value,ULONG addrval,HCARD handleval);
BTICardAPI VOID BTICard_RAMWrmL(LPULONG valueptr,LPULONG addrptr,INT countval,HCARD handleval);
BTICardAPI VOID BTICard_RAMWrmW(LPUSHORT valueptr,LPULONG addrptr,INT countval,HCARD handleval);
BTICardAPI VOID BTICard_RAMWrsL(LPULONG valueptr,ULONG addrval,INT countval,HCARD handleval);
BTICardAPI VOID BTICard_RAMWrsW(LPUSHORT valueptr,ULONG addrval,INT countval,HCARD handleval);
BTICardAPI VOID BTICard_RAMWrW(USHORT value,ULONG addrval,HCARD handleval);
BTICardAPI ERRVAL BTICard_ROMProg(USHORT enableflag,LPUSHORT valueptr,USHORT count,ULONG
addrval,HCARD handleval);
BTICardAPI ULONG BTICard_ReverseLong(ULONG value);
BTICardAPI USHORT BTICard_ReverseWord(USHORT value);
BTICardAPI ULONG BTICard_SeqBlkRd(LPUSHORT buf,ULONG bufcount,LPULONG blkcnt,HCARD handleval);
BTICardAPI ULONG BTICard_SeqBlkRdEx(LPUSHORT buf,ULONG bufcount,ULONG maxblkcnt,LPULONG
blkcnt,HCARD handleval);
BTICardAPI ERRVAL BTICard_SeqClear(HCARD handleval);
BTICardAPI USHORT BTICard_SeqCommRd(LPUSHORT buf,USHORT bufcount,HCARD handleval);
BTICardAPI ERRVAL BTICard_SeqConfig(ULONG configval,HCARD handleval);
BTICardAPI ERRVAL BTICard_SeqConfigEx(ULONG configval,ULONG seqcount,USHORT cardnum,HCARD
handleval);
BTICardAPI ERRVAL BTICard_SeqConfigExx(ULONG configval,ULONG seqaddr,ULONG seqcount,USHORT
cardnum,HCARD handleval);
BTICardAPI BOOL BTICard_SeqFindCheckVersion(LPUSHORT pRecord,USHORT version);
BTICardAPI ERRVAL BTICard_SeqFindInit(LPUSHORT seqbuf,INT seqbufsize,LPSEQFINDINFO sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindMore1553(LPSEQRECORDMORE1553 *pRecMore,LPSEQRECORD1553 pRecBase);
BTICardAPI ERRVAL BTICard_SeqFindMore1553Ex(LPSEQRECORDMORE1553 pRecMore,USHORT
recordsize,LPSEQRECORD1553 pRecBase);
BTICardAPI ERRVAL BTICard_SeqFindNext(LPUSHORT *pRecord,LPUSHORT seqtype,LPSEQFINDINFO sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNext1553(LPSEQRECORD1553 *pRecord,LPSEQFINDINFO sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNext1553Ex(LPSEQRECORD1553 pRecord,USHORT
recordsize,LPSEQFINDINFO sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNext429(LPSEQRECORD429 *pRecord,LPSEQFINDINFO sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNext429Ex(LPSEQRECORD429 pRecord,USHORT recordsize,LPSEQFINDINFO
sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNext708(LPSEQRECORD708 *pRecord,LPSEQFINDINFO sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNext708Ex(LPSEQRECORD708 pRecord,USHORT recordsize,LPSEQFINDINFO
sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNext717(LPSEQRECORD717 *pRecord,LPSEQFINDINFO sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNext717Ex(LPSEQRECORD717 pRecord,USHORT recordsize,LPSEQFINDINFO
sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNextCSDB(LPSEQRECORDCSDB *pRecord,LPSEQFINDINFO sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNextCSDBEx(LPSEQRECORDCSDB pRecord,USHORT
recordsize,LPSEQFINDINFO sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNextDIO(LPSEQRECORDDIO *pRecord,LPSEQFINDINFO sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNextDIOEx(LPSEQRECORDDIO pRecord,USHORT recordsize,LPSEQFINDINFO
sfinfo);
BTICardAPI ERRVAL BTICard_SeqFindNextEx(LPUSHORT pRecord,USHORT recordcount,LPUSHORT
seqtype,LPSEQFINDINFO sfinfo);
BTICardAPI INT BTICard_SeqInterval(INT interval,INT mode,HCARD handleval);
BTICardAPI USHORT BTICard_SeqIntervalEx(USHORT shiftval,HCARD handleval);
BTICardAPI BOOL BTICard_SeqIsRunning(HCARD handleval);
BTICardAPI USHORT BTICard_SeqLogFrequency(USHORT logfreq,HCARD handleval);
BTICardAPI USHORT BTICard_SeqRd(LPUSHORT buf,HCARD handleval);
BTICardAPI USHORT BTICard_SeqRdEx(LPUSHORT buf,USHORT bufcount,HCARD handleval);
BTICardAPI BOOL BTICard_SeqResume(HCARD handleval);
BTICardAPI BOOL BTICard_SeqStart(HCARD handleval);
BTICardAPI BOOL BTICard_SeqStatus(HCARD handleval);
BTICardAPI BOOL BTICard_SeqStop(HCARD handleval);
BTICardAPI ULONG BTICard_Shl(ULONG dataval,USHORT cntval);
BTICardAPI ULONG BTICard_Shr(ULONG dataval,USHORT cntval);
BTICardAPI USHORT BTICard_SignMant(LPCSTR str);
BTICardAPI ERRVAL BTICard_SwapEndianL(LPULONG value);
BTICardAPI INT BTICard_TickTimerStart(INT milliseconds);
BTICardAPI BOOL BTICard_TickTimerValid(INT timer);
BTICardAPI VOID BTICard_TimerClear(HCARD handleval);
BTICardAPI ULONG BTICard_TimerRd(HCARD handleval);
BTICardAPI INT BTICard_TimerResolution(INT timerresol,HCARD handleval);
BTICardAPI USHORT BTICard_TimerResolutionEx(USHORT timershift,HCARD handleval);
BTICardAPI VOID BTICard_TimerWr(ULONG value,HCARD handleval);
BTICardAPI LPSTR BTICard_ValAsciiCat(LPSTR strdest,LPCSTR strsrc);
BTICardAPI INT BTICard_ValAsciiCmpi(LPSTR str1,LPSTR str2);
BTICardAPI LPSTR BTICard_ValAsciiCpy(LPSTR strdest,LPCSTR strsrc,INT count);
BTICardAPI VOID BTICard_ValAsciiTrimLead(LPSTR buf);
BTICardAPI VOID BTICard_ValAsciiTrimTrail(LPSTR buf);
BTICardAPI VOID BTICard_ValAsciiTrimZero(LPSTR buf);
BTICardAPI ULONG BTICard_ValFromAscii(LPCSTR asciistr,INT radixval);
BTICardAPI ULONG BTICard_ValGetBits(ULONG oldvalue,INT startbit,INT endbit);
BTICardAPI LPSTR BTICard_ValIncAscii(LPSTR asciistr);
BTICardAPI LPSTR BTICard_ValInccAscii(LPSTR asciistr);
BTICardAPI BOOL BTICard_ValIsLower(INT value);
BTICardAPI INT BTICard_ValLenAscii(INT numbits,INT radixval);
BTICardAPI ULONG BTICard_ValPutBits(ULONG oldvalue,ULONG newfld,INT startbit,INT endbit);
BTICardAPI LPSTR BTICard_ValToAscii(ULONG value,LPSTR asciistr,INT numbits,INT radixval);
BTICardAPI INT BTICard_ValToUpper(INT value);
BTICardAPI ULONG BTICard_VARRdL(USHORT addrval,HCARD handleval);
BTICardAPI VOID BTICard_VARRdsW(LPUSHORT valueptr,USHORT addrval,INT countval,HCARD handleval);
BTICardAPI USHORT BTICard_VARRdW(USHORT addrval,HCARD handleval);
BTICardAPI VOID BTICard_VARWrL(ULONG value,USHORT addrval,HCARD handleval);
BTICardAPI VOID BTICard_VARWrsW(LPUSHORT valueptr,USHORT addrval,INT countval,HCARD handleval);
BTICardAPI VOID BTICard_VARWrW(USHORT value,USHORT addrval,HCARD handleval);

#ifdef __cplusplus
}
#endif

/**
*
*  Core number constants
*
**/

enum {  COREA              = 0x0000,              //Selects Core A
                COREB              = 0x0001,              //Selects Core B
                COREC              = 0x0002,              //Selects Core C
                CORED              = 0x0003              //Selects Core D
};

/**
*
*  Sequential Record configuration options
*
**/

enum {  SEQCFG_DEFAULT     = 0x00000000L,          //Select all default settings
                SEQCFG_FILLHALT    = 0x00000000L,          //Enable sequential record in fill and
halt mode (default)
                SEQCFG_DISABLE     = 0x00000001L,          //Disable sequential record
                SEQCFG_CONTINUOUS  = 0x00000002L,          //Enable sequential record in
continuous mode
                SEQCFG_DMA         = 0x00000004L,          //Enable monitor in dma mode
                SEQCFG_FREE        = 0x00000008L,          //Enable sequential record in free mode
                SEQCFG_DELTA       = 0x00000010L,          //Enable sequential record in delta
mode
                SEQCFG_INTERVAL    = 0x00000020L,          //Enable sequential record in interval
mode
                SEQCFG_NOLOGFULL   = 0x00000000L,          //Do not generate event log when
sequential record is full (default)
                SEQCFG_LOGFULL     = 0x00001000L,          //Generate event log when sequential
record is full
                SEQCFG_NOLOGFREQ   = 0x00000000L,          //Do not generate event logs at a user
specified frequency (default)
                SEQCFG_LOGFREQ     = 0x00002000L,          //Generate event logs at user specified
frequency
                SEQCFG_16K         = 0x00000000L,          //Allocate a 16K sequential record
buffer (default)
                SEQCFG_ALLAVAIL    = 0x01000000L,          //Allocate all available memory to a
sequential record buffer
                SEQCFG_32K         = 0x02000000L,          //Allocate a 32K sequential record
buffer
                SEQCFG_64K         = 0x04000000L,          //Allocate a 64K sequential record
buffer
                SEQCFG_128K        = 0x08000000L           //Allocate a 128K sequential record
buffer
};

/**
*
*  Sequential Record type fields
*
**/

enum {  SEQTYPE_MASK       = 0x00FF,              //Sequential record type mask value
                SEQTYPE_429        = 0x0001,              //Sequential record type is ARINC 429
                SEQTYPE_717        = 0x0002,              //Sequential record type is ARINC 717
                SEQTYPE_1553       = 0x0003,              //Sequential record type is MIL-STD-
1553
                SEQTYPE_708        = 0x0004,              //Sequential record type is ARINC 708
                SEQTYPE_USER       = 0x0005,              //Sequential record type is User
Defined
                SEQTYPE_CSDB       = 0x0006,              //Sequential record type is CSDB
                SEQTYPE_DIO                = 0x0007               //Sequential record type is DIO
};

/**
*
*  Sequential Record version fields
*
**/

enum {  SEQVER_MASK        = 0xFF00,              //Sequential record version mask value
                SEQVER_0           = 0x0000,              //Sequential record version number is 0
                SEQVER_1           = 0x0100               //Sequential record version number is 1
};

/**
*
*  Event log list configuration options
*
**/

enum {  LOGCFG_DEFAULT     = 0x00000000L,         //Select all default settings
                LOGCFG_ENABLE      = 0x00000000L,         //Enable event log list (default)
                LOGCFG_DISABLE     = 0x00000001L          //Disable event log list
};

/**
*
*  IRIG timer configuration options
*
**/

enum {  IRIGCFG_DEFAULT    = 0x00000000L,         //Select all default settings
                IRIGCFG_ENABLE     = 0x00000000L,         //Enable IRIG timer (default)
                IRIGCFG_DISABLE    = 0x00000001L,         //Disable IRIG timer
                IRIGCFG_SPEEDB     = 0x00000000L,         //Bit rate is IRIGB (default)
                IRIGCFG_SPEEDA     = 0x00000002L,         //Bit rate is IRIGA
                IRIGCFG_INTERNAL   = 0x00000000L,         //IRIG timer operates internally
(default)
                IRIGCFG_EXTERNAL   = 0x00000004L,         //IRIG timer operates externally
                IRIGCFG_SLAVE      = 0x00000000L,         //IRIG timer is a slave  / receiver
(default)
                IRIGCFG_MASTER     = 0x00000008L          //IRIG timer is a master / transmitter
};

/**
*
*  IRIG timer field definitions
*
**/

enum {  IRIGFIELD_USECLSB  = 0x0000,              //Microseconds LSB in timestamp field
                IRIGFIELD_USECMSB  = 0x000B,              //Microseconds MSB in timestamp field
                IRIGFIELD_MSECLSB  = 0x000C,              //Milliseconds LSB in timestamp field
                IRIGFIELD_MSECMSB  = 0x0017,              //Milliseconds MSB in timestamp field
                IRIGFIELD_SECLSB   = 0x0018,              //Seconds LSB in timestamp field
                IRIGFIELD_SECMSB   = 0x001F,              //Seconds MSB in timestamp field

                IRIGFIELD_MINLSB   = 0x0000,              //Minutes LSB in timestamph field
                IRIGFIELD_MINMSB   = 0x0007,              //Minutes MSB in timestamph field
                IRIGFIELD_HRSLSB   = 0x0008,              //Hours LSB in timestamph field
                IRIGFIELD_HRSMSB   = 0x000F,              //Hours MSB in timestamph field
                IRIGFIELD_DAYLSB   = 0x0010,              //Days LSB in timestamph field
                IRIGFIELD_DAYMSB   = 0x001B               //Days MSB in timestamph field
```

```
};

/**
*
*  C54x Global Registers
*
**/

enum {  C54_GREG_IMR       = 0x0000,               //Interrupt mask register
                C54_GREG_IFR       = 0x0001,               //Interrupt flag register
                C54_GREG_ST0       = 0x0006,               //Status register 0
                C54_GREG_ST1       = 0x0007,               //Status register 1
                C54_GREG_AL        = 0x0008,               //Accumulator A low word (bits 15-00)
                C54_GREG_AH        = 0x0009,               //Accumulator A high word (bits 31-16)
                C54_GREG_AG        = 0x000A,               //Accumulator A guars bits (bits 39-32)
                C54_GREG_BL        = 0x000B,               //Accumulator B low word (bits 15-00)
                C54_GREG_BH        = 0x000C,               //Accumulator B high word (bits 31-16)
                C54_GREG_BG        = 0x000D,               //Accumulator B guard bits (bits 39-32
                C54_GREG_T         = 0x000E,               //Temporary register
                C54_GREG_TRN       = 0x000F,               //Transition register
                C54_GREG_AR0       = 0x0010,               //Auxilliary register 0
                C54_GREG_AR1       = 0x0011,               //Auxilliary register 1
                C54_GREG_AR2       = 0x0012,               //Auxilliary register 2
                C54_GREG_AR3       = 0x0013,               //Auxilliary register 3
                C54_GREG_AR4       = 0x0014,               //Auxilliary register 4
                C54_GREG_AR5       = 0x0015,               //Auxilliary register 5
                C54_GREG_AR6       = 0x0016,               //Auxilliary register 6
                C54_GREG_AR7       = 0x0017,               //Auxilliary register 7
                C54_GREG_SP        = 0x0018,               //Stack pointer
                C54_GREG_BK        = 0x0019,               //Circular-buffer size register
                C54_GREG_BRC       = 0x001A,               //Block-repeat counter
                C54_GREG_RSA       = 0x001B,               //Block-repeat start address
                C54_GREG_REA       = 0x001C,               //Block-repeat end address
                C54_GREG_PMST      = 0x001D,               //Processor mode status register
                C54_GREG_XPC       = 0x001E,               //Program counter extension register
                C54_GREG_DRR20     = 0x0020,               //McBSP0 data receive register high
                C54_GREG_DRR10     = 0x0021,               //McBSP0 data receive register low
                C54_GREG_DXR20     = 0x0022,               //McBSP0 data transmit register high
                C54_GREG_DXR10     = 0x0023,               //McBSP0 data transmit register low
                C54_GREG_TIM       = 0x0024,               //Timer count register
                C54_GREG_PRD       = 0x0025,               //Timer period register
                C54_GREG_TCR       = 0x0026,               //Timer control register
                C54_GREG_SWWSR     = 0x0028,               //External interface software wait-
state register
                C54_GREG_BSCR      = 0x0029,               //External interface bank-switching
control register
                C54_GREG_SWCR      = 0x002B,               //Software wait-state control register
                C54_GREG_HPIC      = 0x002C,               //Host port interface control register
                C54_GREG_DRR22     = 0x0030,               //McBSP2 data receive register high
                C54_GREG_DRR12     = 0x0031,               //McBSP2 data receive register low
                C54_GREG_DXR22     = 0x0032,               //McBSP2 data transmit register high
                C54_GREG_DXR12     = 0x0033,               //McBSP2 data transmit register low
                C54_GREG_SPSA2     = 0x0034,               //McBSP2 sub-address register
                C54_GREG_SPDR2     = 0x0035,               //McBSP2 sub-address data register
                C54_GREG_SPSA0     = 0x0038,               //McBSP0 sub-address register
                C54_GREG_SPDR0     = 0x0039,               //McBSP0 sub-address data register
                C54_GREG_DRR21     = 0x0040,               //McBSP1 data receive register high
                C54_GREG_DRR11     = 0x0041,               //McBSP1 data receive register low
                C54_GREG_DXR21     = 0x0042,               //McBSP1 data transmit register high
                C54_GREG_DXR11     = 0x0043,               //McBSP1 data transmit register low
                C54_GREG_SPSA1     = 0x0048,               //McBSP1 sub-address register
                C54_GREG_SPDR1     = 0x0049,               //McBSP1 sub-address data register
                C54_GREG_DMPREC    = 0x0054,               //DMA channel priority and enable
control
                C54_GREG_DMSBAR    = 0x0055,               //DMA channel sub-address register
                C54_GREG_DMADI     = 0x0056,               //DMA channel sub-address data with
increment
                C54_GREG_DMADN     = 0x0057,               //DMA channel sub-address data without
increment
```

```
                C54_GREG_CLKMD     = 0x0058                //Clock-mode register
};

/**
*
*  Event types.
*
**/

enum {  EVENTTYPE_1553MSG    = 0x0001,             //MIL-STD-1553 message
                EVENTTYPE_1553OPCODE = 0x0002,             //MIL-STD-1553 event log opcode
                EVENTTYPE_1553HALT   = 0x0003,             //MIL-STD-1553 schedule halt
                EVENTTYPE_1553PAUSE  = 0x0004,             //MIL-STD-1553 schedule pause
                EVENTTYPE_1553LIST   = 0x0005,             //MIL-STD-1553 list buffer empty/full
                EVENTTYPE_1553SERIAL = 0x0006,             //MIL-STD-1553 serial empty

                EVENTTYPE_429MSG     = 0x0011,             //ARINC 429 message
                EVENTTYPE_429OPCODE  = 0x0012,             //ARINC 429 event log opcode
                EVENTTYPE_429HALT    = 0x0013,             //ARINC 429 schedule halt
                EVENTTYPE_429PAUSE   = 0x0014,             //ARINC 429 schedule pause
                EVENTTYPE_429LIST    = 0x0015,             //ARINC 429 list buffer empty/full
                EVENTTYPE_429ERR     = 0x0016,             //ARINC 429 decoder error detected

                EVENTTYPE_717WORD    = 0x0021,             //ARINC 717 word received
                EVENTTYPE_717SUBFRM  = 0x0022,             //ARINC 717 sub frame completed
                EVENTTYPE_717SYNCERR = 0x0023,             //ARINC 717 receive channel lost
synchronization

                EVENTTYPE_708MSG     = 0x0031,             //ARINC 708 message

                EVENTTYPE_SEQFULL    = 0x0041,             //Sequential record full
                EVENTTYPE_SEQFREQ    = 0x0042,             //Sequential record frequency

                EVENTTYPE_422TXTHRESHOLD    = 0x0051,      //RS-422 TX under threshold
                EVENTTYPE_422TXFIFO         = 0x0052,      //RS-422 TX underflow
                EVENTTYPE_422RXTHRESHOLD    = 0x0053,      //RS-422 RX over threshold
                EVENTTYPE_422RXFIFO         = 0x0054,      //RS-422 RX overflow
                EVENTTYPE_422RXERROR        = 0x0055,      //RS-422 RX error

                EVENTTYPE_CSDBMSG    = 0x0058,             //CSDB message
                EVENTTYPE_CSDBOPCODE = 0x0059,                 //CSDB event log opcode
                EVENTTYPE_CSDBHALT   = 0x005A,             //CSDB schedule halt
                EVENTTYPE_CSDBPAUSE  = 0x005B,             //CSDB schedule pause
                EVENTTYPE_CSDBLIST   = 0x005C,             //CSDB list buffer empty/full
                EVENTTYPE_CSDBERR    = 0x005D,             //CSDB decoder error detected
                EVENTTYPE_CSDBSYNCERR= 0x005E,             //CSDB receive channel lost
synchronization

                EVENTTYPE_DIOEDGE    = 0x0060,             //DIO edge event
                EVENTTYPE_DIOFAULT   = 0x0061              //DIO fault event
};


/**
*
*  Card Info types
*
**/

enum {  INFOTYPE_PLAT      = 0x0001,               //Returns the platform type
                INFOTYPE_PROD      = 0x0002,               //Returns the product type
                INFOTYPE_GEN       = 0x0003,               //Returns the generation number
                INFOTYPE_1553COUNT = 0x0004,               //Returns the 1553 channel count
                INFOTYPE_1553SIZE  = 0x0005,               //Returns the 1553 channel size
                INFOTYPE_429COUNT  = 0x0006,               //Returns the 429 channel count
                INFOTYPE_429SIZE   = 0x0007,               //Returns the 429 channel size
                INFOTYPE_717COUNT  = 0x0008,               //Returns the 717 channel count
                INFOTYPE_717SIZE   = 0x0009,               //Returns the 717 channel size
                INFOTYPE_708COUNT  = 0x000A,               //Returns the 708 channel count
                INFOTYPE_708SIZE   = 0x000B,               //Returns the 708 channel size
                INFOTYPE_VERSION   = 0x000C,               //Returns the version number
                INFOTYPE_DATE      = 0x000D,               //Returns the version date
                INFOTYPE_CHINFO    = 0x000E,               //Returns the channel info
                INFOTYPE_422COUNT  = 0x000F,               //Returns the 422 port count
                INFOTYPE_422SIZE   = 0x0010,               //Returns the 422 port size
                INFOTYPE_CSDBCOUNT = 0x0011,               //Returns the CSDB channel count
                INFOTYPE_CSDBSIZE  = 0x0012,               //Returns the CSDB channel size
                INFOTYPE_DIOCOUNT  = 0x0013,               //Returns the DIO bank count
                INFOTYPE_DIOSIZE   = 0x0014                //Returns the DIO bank size
};

/**
*
*  Co-Processor Info types
*
**/

enum {  COPROCINFO_PLAT      = 0x0001,             //Returns the platform type
                COPROCINFO_PROD      = 0x0002,             //Returns the product type
                COPROCINFO_GEN       = 0x0003,             //Returns the generation number
                COPROCINFO_VERSION   = 0x0004,             //Returns the version number
(major.minor)
                COPROCINFO_DATE      = 0x0005,             //Returns the version date
                COPROCINFO_DMA       = 0x0007,             //Returns whether or not CoPorc
supports DMA mode
                COPROCINFO_VERSIONEX = 0x0009              //Returns the version number
(major.minor.minorminor)
};

/**
*
*  CIS types
*
**/

enum {  CISTYPE_CARD       = 0x0001,             //Select card CIS
                CISTYPE_IOMODULE   = 0x0002                //Select I/O module CIS
};

/**
*
*  Handle Info types
*
**/

enum {  HANDINFO_CORENUM   = 0x0001              //Returns the Handle Core number
};

/**
*
*  Trigger flags
*
**/

enum {  TRIGMASK_TRIGA     = 0x0001,             //Selects trigger line A
                TRIGMASK_TRIGB     = 0x0002,               //Selects trigger line B
                TRIGMASK_TRIGC     = 0x0004,               //Selects trigger line C

                TRIGVAL_TRIGAOFF   = 0x0000,               //Tests for trigger line A inactive
                TRIGVAL_TRIGAON    = 0x0001,               //Tests for trigger line A active
                TRIGVAL_TRIGBOFF   = 0x0000,               //Tests for trigger line B inactive
                TRIGVAL_TRIGBON    = 0x0002,               //Tests for trigger line B active
                TRIGVAL_TRIGCOFF   = 0x0000,               //Tests for trigger line C inactive
                TRIGVAL_TRIGCON    = 0x0004,               //Tests for trigger line C active

                TRIGPOL_TRIGAL     = 0x0000,               //Sets active low polarity for trigger
line A
                TRIGPOL_TRIGAH     = 0x0001,               //Sets active high polarity for trigger
```

```
line A
                TRIGPOL_TRIGBL     = 0x0000,               //Sets active low polarity for trigger
line B
                TRIGPOL_TRIGBH     = 0x0002,               //Sets active high polarity for trigger
line B
                TRIGPOL_TRIGCL     = 0x0000,               //Sets active low polarity for trigger
line C
                TRIGPOL_TRIGCH     = 0x0004                //Sets active high polarity for trigger
line C
};

/**
*
*  Sync flags
*
**/

enum {  SYNCMASK_SYNCA     = 0x0001,               //Selects sync line A
                SYNCMASK_SYNCB     = 0x0002,               //Selects sync line B
                SYNCMASK_SYNCC     = 0x0004,               //Selects sync line C

                SYNCPOL_SYNCAL     = 0x0000,               //Sets active low polarity for sync line
A
                SYNCPOL_SYNCAH     = 0x0001,               //Sets active high polarity for sync
line A
                SYNCPOL_SYNCBL     = 0x0000,               //Sets active low polarity for sync line
B
                SYNCPOL_SYNCBH     = 0x0002,               //Sets active high polarity for sync
line B
                SYNCPOL_SYNCCL     = 0x0000,               //Sets active low polarity for sync line
C
                SYNCPOL_SYNCCH     = 0x0004                //Sets active high polarity for sync
line C
};

/**
*
*  Heap flags.
*
**/

enum {  HEAPCFG_DEFAULT    = 0x0000,               //Block is allocated within page
                HEAPCFG_TEST       = 0x0001,               //Returns address but doesn't make it
permanent
                HEAPCFG_NULL       = 0x0002,               //Returns address but doesn't mark it
as in use
                HEAPCFG_SPAN       = 0x0004,               //Block can span pages
                HEAPCFG_QUICK      = 0x0008,               //Does a quick allocation

                HEAPSECT_SRAM      = 0x0000                //Heap section for primary SRAM
};

/**
*
*  Timer resolutions.
*
**/

enum {  TIMERRESOL_1US     = 1,                    //1us timer resolution, 1:11:34 range
                TIMERRESOL_16US    = 2,                    //16us timer resolution, 19:05:19 range
                TIMERRESOL_1024US  = 3                     //1024us timer resolution, 50 day range
};

/**
*
*  Interval calculation modes.
*
**/
```

```
enum {  INTERVALMODE_CLOSEST  = 1,                 //Finds interval closest to value specified
                INTERVALMODE_LESS     = 2,                 //Finds closest interval less than
value specified
                INTERVALMODE_GREATER  = 3                  //Finds closest interval more than
value specified
};

/**
*
*  Test flags.
*
**/

enum {  TEST_LEVEL_0       = 0,                    //Test I/O interface
                TEST_LEVEL_1       = 1,                    //Test memory interface
                TEST_LEVEL_2       = 2,                    //Test communication process
                TEST_LEVEL_3       = 3                     //Test bus transceiver
};

/**
*
*  Status flags.
*
**/

enum {  STAT_EMPTY         = 0,                   //Buffer is empty
                STAT_PARTIAL       = 1,                   //Buffer is partially filled
                STAT_FULL          = 2,                   //Buffer is full
                STAT_OFF           = 3                    //Buffer is off
};

/**
*
*  Other flags.
*
**/

enum {  RCV             = 0,
                XMT             = 1
};

/**
*
*  Error types.
*
**/

enum {  ERR_NONE          =  0,                   //No error
                ERR_UNKNOWN       = -1,                   //An unexpected error occurred
                ERR_BADVER        = -2,                   //A bad version was encountered
                ERR_BADPTR        = -3,                   //A bad pointer was passed
                ERR_NOCORE        = -4,                   //The specified core number doesn't
exist
                ERR_BADPARAMS     = -11,                  //CardOpen() called with bad parameters
                ERR_NOHANDLES     = -12,                  //CardOpen() already has allocated too
many handles
                ERR_NOCARD        = -13,                  //CardOpen() could not find a L43 card
at the specified address
                ERR_NOIO          = -14,                  //CardOpen() could not find the I/O
ports
                ERR_NOMEM         = -15,                  //CardOpen() could not find the memory
                ERR_BAD16BIT      = -16,                  //Card is conflicting with another 16-
bit card
                ERR_WRONGMODEL    = -17,                  //Card does not support this feature
                ERR_NOSEL         = -18,                  //CardOpen() could not allocate a memory
selector
                ERR_LOCK          = -19,                  //The communication process is locked up
                ERR_TOOMANY       = -20,                  //Too many channels have been configured
                ERR_BADHANDLE     = -21,                  //A bad handle was specified
                ERR_GOODHANDLE    = -22,                  //The handle is still valid and should
not be destroyed
                ERR_NOTCHAN           = -23,                  //Not a valid channel
                ERR_NOTXMT        = -24,                  //The Transmitter has not been
configured
                ERR_NOTRCV        = -25,                  //The Receiver has not been configured
                ERR_NOTSEQ        = -26,                  //The Sequential Record has not been
configured
                ERR_ALLOC         = -27,                  //There is not enough memory to allocate
                ERR_VXD           = -28,                  //An error occurred in the VXD
                ERR_BADLABEL      = -29,                  //The specified label value is not valid
                ERR_BADSDI            = -30,                  //The specified sdi value is not
valid
                ERR_BADMSG        = -31,                  //The specified command block is not a
message block
                ERR_BADSCHNDX     = -32,                  //Specified command index is out of
range
                ERR_BUFSIZE       = -33,                  //Insufficient space in user buffer
                ERR_NOCONFIG      = -34,                  //The card has not been properly
configured
                ERR_CONFLICTS     = -35,                  //Unable to resolve conflicts
                ERR_RANGE         = -36,                  //Schedule is out of range
                ERR_FACTOR        = -37,                  //A bad factor value was specified
                ERR_NOIOBOOT      = -40,                  //Could not talk to IO Boot port of DSP
                ERR_BOOTFULL      = -41,                  //No space to add boot code
                ERR_BOOTNUM       = -42,                  //There is no boot code with the
specified number
                ERR_ACCESS        = -43,                  //Unable to write to access register
                ERR_ROMVERIFY     = -44,                  //Unable to verify the value written to
the ROM
                ERR_COUNT         = -45,                  //An invalid count was specified
                ERR_CRC           = -46,                  //There was a bad checksum in the HEX
file
                ERR_FNAME         = -47,                  //Bad filenames were specified
                ERR_FRDWR         = -48,                  //There was an error reading or writing
the HEX file
                ERR_HEX           = -49,                  //There was a bad hex character in the
HEX file
                ERR_INDEX         = -51,                  //The command block index was invalid or
the schedule is full
                ERR_NOMSGS        = -52,                  //No messages specified
                ERR_TYPE          = -54,                  //There was a bad type value in the HEX
file
                ERR_ZEROLEN       = -55,                  //Zero length was specified
                ERR_BADADDRESS    = -56,                  //A bad address was specified
                ERR_MSGNDX        = -57,                  //A bad message index was specified
                ERR_BADTA         = -60,                  //A bad terminal address was specified
                ERR_BADFRAME      = -61,                  //A bad frame time was specified
                ERR_NOTBC         = -62,                  //The BC has not been configured
                ERR_NOTRT         = -63,                  //The RT has not been configured
                ERR_NOTMON        = -64,                  //The monitor has not been configured
                ERR_SELFIOFAIL    = -71,                  //I/O selftest failed
                ERR_SELFMEMFAIL   = -72,                  //Memory selftest failed
                ERR_SELFCOMMFAIL  = -73,                  //Communication selftest failed
                ERR_SELFXMTFAIL   = -74,                  //Transmit selftest failed
                ERR_PLXBUG        = -75,                  //PLX bug is causing problems
                ERR_NOTSUPPORTED  = -76,                  //Base class does not support feature
                ERR_DLL           = -77,                  //A required DLL is missing
                ERR_SEQTYPE       = -80,                  //Invlaid sequential record type value
                ERR_SEQNEXT       = -81,                  //Next sequential record does not exist
                ERR_SEQFINDINFO   = -82,                  //The SEQFINDINFO structure is not valid
                ERR_SEQBASEPTR    = -83,                  //The base pointer passed is invalid
                ERR_SEQMORE       = -84,                  //More (extended) record data does not
exist
                ERR_TIMEOUT       = -90,                  //Function timed out waiting for data
                ERR_SUBFRMNUM     = -101,                 //Invalid SubFrame number was specified
                ERR_WORDNUM       = -102,                 //Invalid Word number was specified
                ERR_NOTINSYNC     = -103,                 //Not Synchronized to databus
                ERR_SUPERFRM      = -104,                 //SuperFrame not configured
                ERR_SUPERFRMNUM   = -105                  //Invalid SuperFrame number was
specified
};

#endif
```