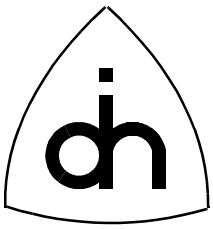
Odin TeleSystems Inc.



Application Programming Interface (API) Reference Guide for THOR DOS driver, Rev 1.24

Doc. No. 1211-1-SDA-1001

Doc. Rev. 1.11 (Released)

November 7, 1997

Changes

The material in this document is for information only and is subject to change without notice. While reasonable efforts have been made in the preparation of this document to assure its accuracy, Odin TeleSystems Inc., assumes no liability resulting from errors or omissions in this document, or from the use of the information contained herein.

Odin TeleSystems Inc. reserves the right to make changes in the product design without reservation and notification to its users.

Warranties

THE SOFTWARE AND ITS DOCUMENTATION ARE PROVIDED “AS IS” AND WITHOUT WARRANTY OF ANY KIND. ODIN TELESYSTEMS EXPRESSLY DISCLAIMS ALL THE WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE. ODIN TELESYSTEMS DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET ANY REQUIREMENTS, OR THAT THE OPERATIONS OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS WILL BE CORRECTED. FURTHERMORE, ODIN TELESYSTEMS DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE SOFTWARE OR ITS DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVISE GIVEN BY ODIN TELESYSTEMS OR ODIN TELESYSTEMS’ AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY.

UNDER NO CIRCUMSTANCE SHALL ODIN TELESYSTEMS INC., ITS OFFICERS, EMPLOYEES, OR AGENTS BE LIABLE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS, PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE AND ITS DOCUMENTATION, EVEN IF ODIN TELESYSTEMS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT WILL ODIN TELESYSTEMS’ LIABILITY FOR ANY REASON EXCEED THE ACTUAL PRICE PAID FOR THE SOFTWARE AND ITS DOCUMENTATION. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL AND CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY.

Odin TeleSystems Inc.

This document is published by:
Odin TeleSystems Inc.
P. O. Box 59686
Dallas, Texas 75229
U. S. A.

Printed in U. S. A.

1. Table of Contents

2. Abstract

The Thor-2 driver provides the software needed by the PC host processor (CPU) to communicate with the board. The driver also contains boot-up software and the operating system for the on-board processor (LPU). The CPU Software is implemented in C language and it is provided in the form of C libraries (*.lib) and header files (*.h). The libraries can be linked to a higher level C or C++ application to provide the functionality needed to communicate with the Thor-2 board and with the on-board processor. The communication between the application software and the Thor-2 driver is performed through an Application Programming Interface (API). This document provides the reference guide for the Thor API.

3. Applicable Products

The Thor drivers covered by this document are listed in Table 1 on page 9.

TABLE 1. THOR Drivers covered by this document.

Product Number	Description
SDA-1001-1	Thor DOS Driver, Compiled with Borland C++ 4.53
SDA-1001-2	Thor DOS Driver, Compiled with Microsoft C 5.1
SDA-1001-3	Thor DOS Driver, Compiled with Microsoft Visual C++ 1.52

4. Distribution

The Thor driver is distributed on a 3.5” diskette. A separate diskette exists for each driver version. The Thor driver distribution diskette contains the following files and directories.

TABLE 2. Thor driver distribution content

Directory	Files	Description
LIB	Thor.lib	Thor API Library compiled with the applicable compiler.
INC	*.h	Thor API header files

TABLE 2. Thor driver distribution content

Directory	Files	Description
UTILS	T2boot.exe	Utility program for loading the on-board Field Programmable Gate Arrays (FPGAs) and for booting the on-board processor (LPU). For a complete description please refer to "User Guide for Thor-2", Doc No. 1412-1-HAA-1004-1.
	T2config.exe	Utility program for reading and writing the line interface configuration data to the on-board flash memory. For a complete description please refer to "User Guide for Thor-2", Doc No. 1412-1-HAA-1004.
	T2.cfg	The default configuration data file. For a complete description please refer to "User Guide for Thor-2", Doc No. 1412-1-HAA-1004-1.
DEMOS	Lapiapp.exe Lapiapp.c Makefile	Low-Level API demo program. Sending and receiving of HDLC frames through the looped Line Interfaces.
	Hapiapp.exe Hapiapp.c Makefile	High-Level API demo program. Sending and receiving of HDLC frames through the looped Line Interfaces.
	Dataapp.exe Dataapp.c Makefile	Demo program using Transparent Mode Pipes. Transfers a binary file between two Line interfaces.
	T1app.exe T1app.c Makefile	Example program demonstrating Thor-2's T1 specific funtions; E.g.. the use Bit-robbed signalling.
	E1app.exe E1app.c Makefile	Example program demonstrating Thor-2's E1 specific function; E.g. access to the Si and Sa bits.
	Phoneapp.exe Phoneapp.c Makefile	Example program demonstrating Thor-2's phone and DTMF tone capabilities.
	Perfapp.exe Perfapp.c Makefile	Example program demonstrateing Thor-2's capability to send and receive HDLC frames simulatenously on multiple channels.
	Patapp.exe Patapp.c Makefile	Example program using Transparent Mode Pipes. Processing of full 2,048 Mbit/s stream. Continuously sends and receives data on 32 time-slots.
	Audioapp.exe Audioapp.c Makefile	Example program demonstrating Thor-2's audio capabilities and the use of hdlcMemoryXXXXX() API functions.
	SS7app.exe SS7app.c Makefile	Example program demonstrating Thor-2 driver's SS#7 support and the use of hdlcSS7XXXX() API functions.

TABLE 2. Thor driver distribution content

Directory	Files	Description
/	Readme	Release note and changes made after printing of this documentation.
	revision.h	A header file contaning driver revision information

5. Driver Overview

5.1 Application Programming Interface (API)

The Thor-2 driver is accessed from the application software through a specified Application Programming Interface (API). The Thor-2 API is a C function-call interface. The Application Programmer can request actions from the driver with high-level function calls without intimate knowledge of the hardware and of the low-level hardware interface. The Thor-2 driver then converts the application requests into appropriate I/O and memory access operations towards the Thor-2 hardware. The Thor-2 Driver also services the interrupts generated by the Thor-2 board. The relationship between the Application Program, the Thor-2 driver, and the Thor-2 Board is illustrated in Figure 1 on page 13.

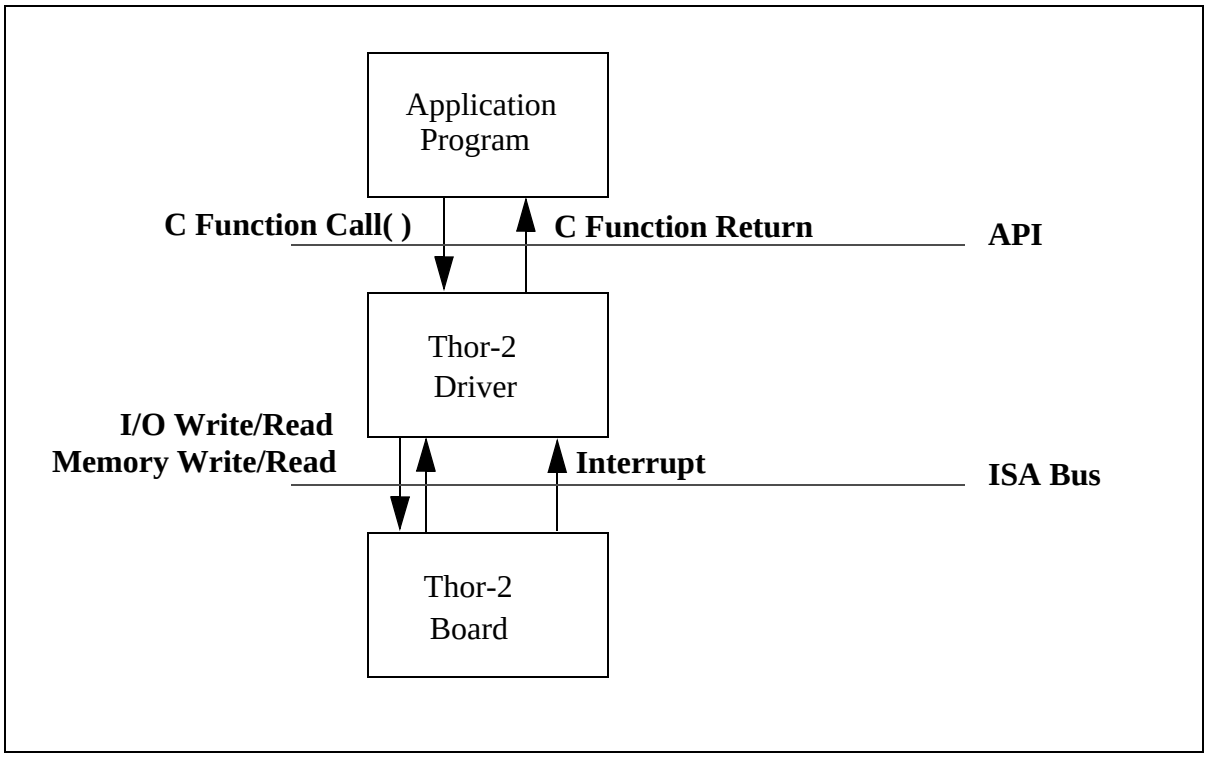


Figure 1. Thor-2 Driver Interfaces

The Thor-2 driver contains two different APIs: The High-level API and the Low-level API.

The High-level API consists of easy-to-use, “pre-packaged” functions. The High-level API functions have been designed to implement “standard applications”; e.g. setting up of a CEPT ISDN primary rate link. These functions provide a minimal set of options to the user but they are designed to be easy and quick to use.

The Low-level API, on the other hand, provides more flexibility to the user. The full functionality of the Thor-2 board can be utilized through the low-level API. However, the functions at this level contain more parameters and require more involvement from the application programmer. The High-level API is implemented using the Low-level API, as illustrated in Figure 2 on page 14.

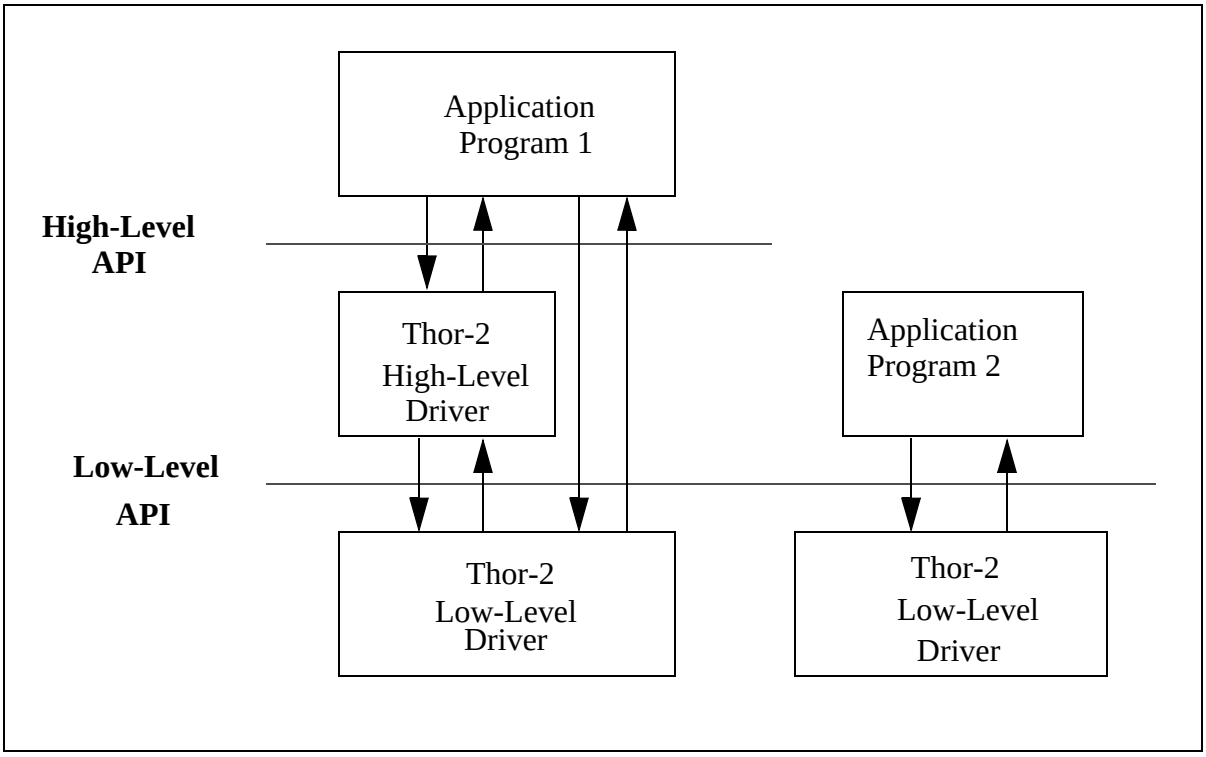


Figure 2. Thor-2 Driver's Two APIs

5.2 Files

The Thor-2 Driver package consists of header (*.h) and library (*.lib) files. The files included in the Thor-2 Driver are shown in Figure 3 on page 15 (Solid black frame) and listed in Table 3 on page 15.

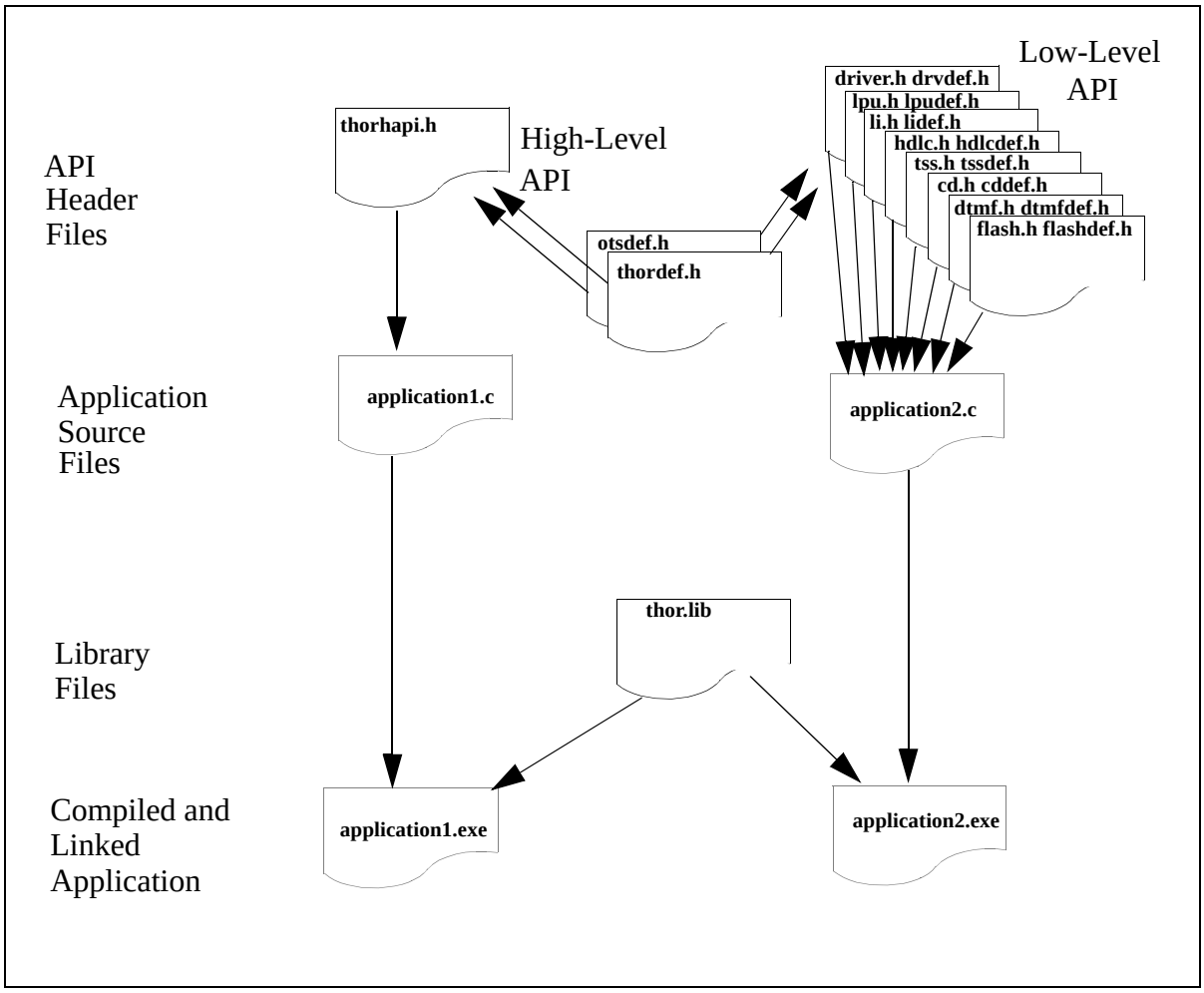


Figure 3. The overall Thor-2 Driver File Structure

TABLE 3. API Definition Header Files

API Level	Header file name	Description
High-level API	thorhapi.h	Macro definitions, type definitions, and function declarations for the high-level API.

TABLE 3. API Definition Header Files

API Level	Header file name	Description
Low-level API	driver.h	Driver level definitions and functions
	li.h	Functions for using the T1/E1 Line Interfaces (LIs).
	hdlc.h	Functions for utilizing the 32-channel HDLC controller.
	tss.h	Functions for using the 384x384 time-space switch.
	cd.h	Functions for the two Codecs.
	dtmf.h	Functions for using the two DTMF Transceivers.
	flash.h	Functions for utilizing the 512 kbyte Flash memory.
	lpu.h	Functions for controlling the Local Processing Unit (LPU)
Common	otsdef.h	Odin TeleSystems general macro and type definitions
	thordef.h	Thor API macro and type definitions
	drvdef.h	Macros, Constants and Type Definition for the Driver
	lidef.h	Macros, Constants and Type Definition for the Line Interfaces
	hdlcdef.h	Macros, Constants and Type Definition for the HDLC Controller
	tssdef.h	Macros, Constants and Type Definition for the Time-Space Switch
	cddef.h	Macros, Constants and Type Definition for the Codecs
	dtmfdef.h	Macros, Constants and Type Definition for the DTMF transceiver
	flashdef.h	Macros, Constants and Type Definition for the Flash Memory
	lpudef.h	Macros, Constants and Type Definition for the LPU

5.3 Naming Conventions

The functions and data types used within the API follow a naming convention. All the names have a prefix (E.f. Thor in the high-level API). The case of the prefix for functions, data types, and macros is different, as shown in Table 3 on page 15.

TABLE 4. Thor API Prefixes

Element	Prefix Case	Prefix Example
Function	All Small Letters	thor
Type	Start with Uppercase	Thor
Macro	All Capital Letters	THOR_

The function names consist of three parts: the prefix, the operation, and the qualifier. The operation word is a verb describing the action to be taken, e.g., construct, read. The qualifier either be a target or a result of the operation. The target describes the entity which will be impacted by the operation: E.g., ‘Driver’ or ‘Li’. The result describes the wanted end state of the operation: E.g. ‘On’, or ‘Off’.

.

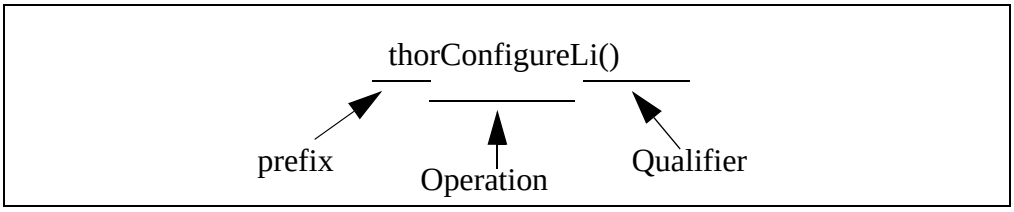


Figure 4. An Example of the Naming of the Functions.

The used prefixes are listed in Table 3 on page 15.

TABLE 5. Thor-2 prefixes

API Level	File name	Prefix and its meaning
High-level API	thorhapi.h	**thor** for the THOR-2 board
Low-level API	driver.h	**drv** for Driver.
	li.h	**li** for a Line Interface.
	hdlc.h	**hdlc** for the High-level Data Link Control.
	tss.h	**tss** for a Time-Space Switch
	cd.h	**cd** for a CoDecs.
	dtmf.h	**dtmf** for a Dual Tone Multi Frequency
	flash.h	**flash** for the FLASH memory
	lpu.h	**lpu** for Local Processing Unit
	winutils.h	Generic utility functions, no common prefix used.

5.3.1 Exception Handling

Exception handling is provided via function return codes. The return codes have defined as an enumerated type ThorRc. Each function returns 1 (“THOR_SUCCESS”) if the operation was successful (no errors). In a case of an error the function returns a non-zero value (other than 1). the value 0 is an “undefined” return code (“THOR_UNDEFINED”). The return codes can be translated into strings containing the error message with the “*thorGetErrMsg( )*” function.

5.3.2 Standard Parameters

The driver supports up to four Thor-2 boards. Most of the functions in the API apply to one board at the time. Those functions always have the board number as the first parameters. The valid board numbers are 0 - 3. Certain resources are duplicated on the Thor-2 board; E.g., the Line Interfaces, the Codecs, and the DTMF Transceivers. When a function applies to a specific circuit on a board, the second parameter to the function will be the chip number (0 or 1). Figure 5 on page 18 illustrates a typical Thor-2 API function declaration.

.

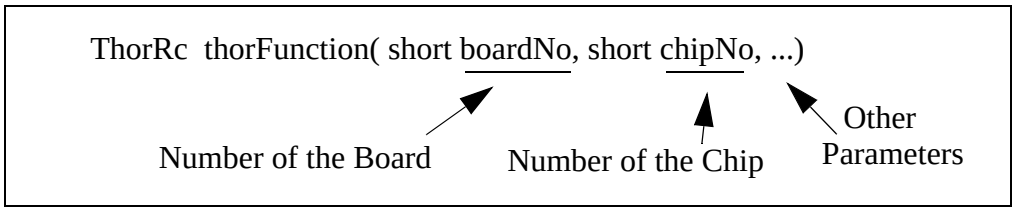


Figure 5. A typical function declaration

6. Configuration using *T2Config.exe*

The Thor-2 board contains 512 KBytes Flash memory. The Flash is used by the Thor-2 driver to persistently store the T1/E1 link configuration parameters. Thor-2 is delivered with a utility program *T2Config.exe* which can be used to store configuration information into the flash. An application using the Thor-2 driver can then retrieve the information from flash and configure the driver and the board accordingly. For more information on the Thor-2 Configuration file and the *T2Config.exe* utility program, please refer to the *Thor-2 User Guide* (Odin TeleSystems Inc. Doc. No. 1412-1-HAA-1004-1).

The overall process of how the configuration parameters are set is described in the following:

1. The user sets the wanted configuration parameters by editing the Thor-2 configuration file.

2. The user runs *T2Config.exe* providing the configuration file to be used as the command line parameter.

    2.1. *T2Config.exe* parses and syntax checks the configuration file.

    2.2. If the configuration parameters are different than the ones stored in the flash already, *T2Config.exe* stores the new parameters into the Flash. If the parameters have not changed, T2Config.exe does nothing.

3. The application program can call the *drvReadConfigData()* function to read the stored configuration parameters from the flash memory.

4. The application program can modify the parameters or pass them as is to *liConfigure()* function. The application program can also set the clock source with *drvSetClkSrc()* function.

5. If the application program wants to store different values into the flash, it can do so with *drvWriteConfigData()* function.

7. Low-Level API Function Calling Sequence Example

The recommended calling sequence for initialization of the driver using the Low-Level API with the DOS driver is described in the following.

Note: with the DOS driver the application needs to provide the communications parameters to the Driver. With Windows 95 and Windows NT drivers this information is available in the Windows registry and certain function calls are not needed.

```
// Thor-2 Application using the Low-Level API: Calling Sequence example


void main()
{
  ThorRc t2Rc;


   //  Tell the DOS driver the common parameteres
   // -------------------------------------------
  t2Rc = drvSetup(..)
  if (t2Rc != THOR_SUCCESS) {
    // ERROR:
  }

  // Init board and driver structs:
  // ------------------------------
  t2Rc = drvInit(..);
  if (t2Rc != THOR_SUCCESS) {
    // Error
  }
  // Reset all devices:
  // ------------------
  t2Rc = drvResetDevices(...);

  // Board Existence Check:
  // ----------------------
  t2Rc = drvBoardExistence(...);

  // Check FPGA Configuration:
  // -------------------------
  t2Rc = drvFpgaStatus(...);

  // Setup the memory window:
  // ------------------------
  t2Rc = drvSetupMemWin(...);

  // Setup the I/O Window:
```

```
// ---------------------
t2Rc = drvSetupIoWin(...);

// Boot up the LPU:
// ----------------
t2Rc = lpuBoot(...);

// Install Interrupt Service Routine if needed:
// -------------------------------------------
if (aIrq != -1) {
  t2Rc = drvInstallIsr(...);
}

// Initialize Devices:
// ---------------------
tssInit(bNo);
t2Rc = drvInitHdlc(...);

// Read Config Data from Flash:
// ----------------------------
t2Rc = drvReadConfigData(...);

// Set clock source:
// -----------------
t2Rc = drvSetClkSrc(...);

// Configure Line Interfaces:
// --------------------------
t2Rc = liConfigure(...);
t2Rc = liConfigure(...);

// Configure the pipes:
// --------------------
for (short pipe = 0; pipe <= 1; pipe++) {
  t2Rc = hdlcInitPipe(...);
}

// Time-Space Switch connections:
// ------------------------------
tssXConnect(...);
tssXConnect(...);

// Main Loop:
// ---------
```


```
  while (..) {
    idleFunction();
    :
    // Send messages:
    t2Rc = hdlcSendData(...);
  }  // while

  // Cleanup the driver:
  // -------------------
  t2Rc = drvUnistallIsr(...);
} // main
void idleFunction()
{
  // Check for messages:
  if ( (t2Rc = drvRead(..)) != THOR_NO_FRAMES) {
    // A frame (hdlc message, status message, or DTMF digit
    // Received, process
  }
     :
}
```

8. High-Level API Function Calling Sequence Example

While most of the Thor-2 functions can be called at any time and in any order, a recommended calling sequence for the initialization of the driver exists. An example of a working calling sequence for the High-Level API functions is described in the following:

```
// Thor-2 Application using the High-Level API: Calling Sequence example

void main()
{
  ThorRc thorRc;

  // Initialize the driver:
  // ----------------------
  thorRc = thorConstructDriver(...);
  if ( thorRc != THOR_SUCCESS) {
    // Error
  }
   :

  // Configure the Line Interfaces:
  // ------------------------------
  thorRc = thorConfigureLi(..);
  thorRc = thorConfigureLi(..);

  :

  // Configure the HDLC pipes:
  // -------------------------
  thorRc = thorConfigurePipe(..);
  :

  // Create Time-Space Switch Connections:
  // -------------------------------------
  thorRc = thorConnectChannel(..);
  :

  // Main Loop:
  // ---------
  while (..) {
    idleFunction();
    :
    // Send messages:
    thorRc = thorWritePipe(..);
```

```
  }  // while

  // Cleanup the driver:
  // -------------------
  thorRc = thorDestructDriver();
} // main


void idleFunction()
{
  // Check for messages:
  if ( (thorRc = thorRead(..)) != THOR_NO_FRAMES) {
    // A frame (hdlc message, status message, or DTMF digit
    // Received, process
  }
     :
}
```

9. Thor Driver Example Applications

9.1 Low-Level API Application: *LAPIAPP.EXE*

The *LAPIAPP.EXE* example program demonstrates the use of the Low Level API in an application for sending and receiving HDLC frames on one time-slot. The Application sets up (constructs) the driver, configures the two Line interfaces, and defines one pipe for each Line Interface (LI). After the configuration has been completed, the application enters in to the main loop where it polls for keystrokes or received messages. If key 't' is pressed, the application will send a message on Pipe 1. If the LIs are looped together with the Thor-2 Li Loop Cable, the message sent on Pipe 1 (Li 0) will be received on Pipe 2 (on Li1). The idleFunction() will pick up the received message and display it to the screen. Help can be displayed by pressing '?'.

9.2 High-Level API Application: *HAPIAPP.EXE*

The *HAPIAPP.EXE* implements the same functionality as the LAPIAPP.EXE but uses the Thor High-Level API.

9.3 Application for Sending and Receiving Raw Data: *DATAAPP.EXE*

The *DATAAPP.EXE* example program demonstrates the Thor board's capabilities to send and receive data transparently (without any HDLC framing). The data from an input file is sent out on Pipe 1 (which is connected to Li0) in four channels configured as a 256 kbit/s data stream (4x64kbit/s timeslots). If the Li0 and Li1 are looped together with the Thor-2 Li Loop cable, the Pipe 2 (which is connected to Li1) will consequently receive the same 256 kbit/s data stream and store the data in a temporary file.

Since these pipes are configured in the transparent mode TMA (for more information on the transparent modes, please refer to the Thor-2 Technical Description) they continuously receive data if the receiver is active. The idle pattern is 0xFF which is transmitted and received in the absence of actual data. To separate the actual data from the idle data. the data to be send are enclosed with signatures. The function removeSignature() reads the temporary file that contains the entire data stream, extracts the actual data between the signatures, and copies it to the final output file.

9.4 Application for Sending and Receiving Data Patterns: PATAPP.EXE

The idea behind the PATAPP.EXE demo application is to demonstrate the ability to continuously send data on maximum number of time-slots. This application sends and receives data on 31 time-slots to achieve a full E1 utilization. Difference piece of data is copied to the on-board memory for each 31 time-slots. Once the data is in-the memory, it will be send continuously over and over again. If the access 0 is looped to access 1 with the Li loop cable, the data will be received and downloaded to the Host

PC. The Host CPU will also compare on real-time that the received data matches the sent data. If any discrepancies are found, the program will stop and report the error.

9.5 Demo of T1 Functions: T1APP.EXE

The T1APP.EXE demo implements the functionality of LAPIAPP and SAPIAPP. In addition it demonstrates how the Thor-2 card can be used to handle Bit-Robbed signaling in the T1 mode.

9.6 Demo of E1 Functions: E1APP.EXE

The E1APP.EXE demo implements the functionality of LAPIAPP and SAPIAPP. In addition it demonstrates how the Thor-2 card can be used to access the S_i and S_a bits in the E1 mode.

9.7 Demo of Phone Functions: PHONEAPP.EXE

The PHONEAPP.EXE demo illustrates how the Thor-2 Codecs can be used to create a speech connection between the two Line Interfaces (LIs). It also demonstrates how DTMF tones can be send out from one LI and received and detected on the other.

9.8 Sending and Receiving of HDLC Frames on Multiple Pipes: PERFAPP.EXE

The PERFAPP.EXE demo shows the Thor-2 boards capability to send and receive HDLC frames on multiple channels (pipes) simultaneously.

9.9 Demo of Thor-2's Audio functions: AUDIOAPP.EXE

The AUDIOAPP.EXE demonstrates Thor-2's audio capabilities and the use of hdlcMemoryXXXXX() API functions.

9.10 Demo of Thor-2's SS#7 Support: SS7APP.EXE

The SS7APP.EXE demonstrates Thor-2 drivers SS#7 support and the use of hdlcSS7XXXXX() API functions.

10. Common API Macro, Constant, and Type Definitions

10.1 Odin TeleSystems' Standard Definitions - *otsdef.h*

10.1.1 Standard Type Names

Synopsis

Odin TeleSystems utilizes standardized names for basic data types: Byte is 8 bits, Word is 16 bits, and Double-Word is 32 bits.

Definition

```
typedef unsigned char  Uchar;
typedef unsigned char  Byte;
typedef unsigned int   Uint;
typedef unsigned short Ushort;
typedef unsigned short Word;
typedef unsigned long  Dword;
typedef unsigned long  Ulong;
```

10.2 Thor Specific Definitions - *thordef.h*

10.2.1 Board Definitions

Synopsis

The Thor-2 board has a unique identification code THOR_BOARD_TYPE. The driver supports upto BOARD_PER_PC number of Thor-2 boards withing one PC. One Thor-2 board contains THOR_LI_PER_BOARD Line Interface transceivers, THOR_CD_PER_BOARD Codecs, and THOR_DTMF_PER_BOARD Dtmf Transceivers.

Definition

```
#define THOR_BOARD_TYPE  106   // A unique code for Thor-2
#ifdef THOR_LITE
  #define BOARD_PER_PC  1       // Per unit (PC) (1 counting)
#else
  #define BOARD_PER_PC  4       // Per unit (PC) (1 counting)
#endif
#define THOR_LI_PER_BOARD   2   // Line Interfaces per board (1 counting)
#define THOR_CD_PER_BOARD   2   // Codecs per board (1 counting)
#define THOR_DTMF_PER_BOARD 2   // DTMF Transceivers per board (1 counting)
```

10.2.2 Highway Definitions

Synopsis

All the Thor-2 internal highways are 2.048 MBits/s consisting of 32 64KBit/s time-slots. On the AUX highway the Codec #0 is connected to time-slot #2, and Codec #1 to time-slot #3.

Definition

```
#define THOR_NO_OF_HIGHWAY_CHANNELS 32   // Number of channels in an
                                         // internal Thor highway
#define THOR_AUX_CD0_CH 2           // The channel for Codec 0
                                    // on the Auxiliary PCM highway
#define THOR_AUX_CD1_CH 3           // The channel for Codec 1
                                    // on the Auxiliary PCM highway
```

10.2.3 Internal Data Highways - *ThorPhwType*

Synopsis

The highway type is used to identify the Thor-2 internal highway. On Thor-2 a highway is always a 2.048 Mbit/s bi-directional bit stream. One highway consists of 32 channels (8-bit time-slots), each having a data rate of 64 Kbit/s. All together, the Thor-2 board contains 12 separate internal highways and 384 channels.

Definition

```
typedef enum {
  THOR_PHW_MVIP0 = 0,      // MVIP0 PCM highway
  THOR_PHW_MVIP1 = 1,      // MVIP1 PCM highway
  THOR_PHW_MVIP2 = 2,      // MVIP2 PCM highway
  THOR_PHW_MVIP3 = 3,      // MVIP3 PCM highway
  THOR_PHW_MVIP4 = 4,      // MVIP4 PCM highway
  THOR_PHW_MVIP5 = 5,      // MVIP5 PCM highway
  THOR_PHW_MVIP6 = 6,      // MVIP6 PCM highway
  THOR_PHW_MVIP7 = 7,      // MVIP7 PCM highway
  THOR_PHW_LI0   = 8,      // Li0 PCM highway
  THOR_PHW_LI1   = 9,      // Li1 PCM highway
  THOR_PHW_CTRL  = 10,     // HDLC Controller PCM highway
  THOR_PHW_AUX   = 11,     // Codec/DTMF PCM highway
  THOR_PHW_UNDEF
} ThorPhwType;
```

See Also

`tssXConnect()`
`IOCTL_TSS_XCONNECT`
`tssConstByte()`
`IOCTL_TSS_CONST_BYTE`

10.2.4 Clock Source - *ThorClkSrcType*

Synopsis

All the Thor-2 internal highways are clocked with one single clock source. Clocking for the Thor-2 board can be derived from the sources defined by ThorClkSrcType.

Definition

```
typedef enum {
  THOR_CLK_INTERNAL,         // Internal oscillator on the board
  THOR_CLK_MVIP_MASTER_CLK,  // Synchronized to MVIP signal C4\
```

```
  THOR_CLK_MVIP_SEC_CLK,     // Synchronized to MVIP secondary clock
  THOR_CLK_FA0,              // Synchronized to Falc 0 incoming span
  THOR_CLK_FA1,              // Synchronized to Falc 1 incoming span
  THOR_CLK_EXTERNAL_8K,      // Synchronized to external 8kHz
  THOR_CLK_EXTERNAL_2M       // Synchronized to external 2MHz
} ThorClkSrcType;
```

See Also

```
drvSetClkSrc()
IOCTL_DRV_SET_CLK_SRC
```

10.2.5 CPU Interrupt Mask - *THOR_CIM_XXX*

Synopsis

The CPU Interrupt Mask (CIM) register can be used to mask interrupts from the Thor-2 devices towards the Host CPU. Clearing the register masks all the device interrupts and setting the register enables them. The macros for the CIM register bits are defined as follows:

Definition

```
#define THOR_CIM_MUN      0x01     // Mask interrupts from HDLC Controller
#define THOR_CIM_FA0      0x02     // Mask interrupts from Line Interface #0
#define THOR_CIM_FA1      0x04     // Mask interrupts from Line Interface #1
#define THOR_CIM_FMI      0x08     // Mask interrupts from Time-Space Switch
#define THOR_CIM_DT0      0x10     // Mask interrupts from DTMF Transceiver #0
#define THOR_CIM_DT1      0x20     // Mask interrupts from DTMF Transceiver #1
#define THOR_CIM_LPU      0x40     // Mask interrupts from On-board Processor
```

See Also

```
drvEnableCpuIntr()
drvDisableCpuIntr()
THOR_LIM_XXX   // LPU Interrupt Mask
```

10.2.6 LPU Interrupt Mask - *THOR_LIM_XXX*

Synopsis

The LPU Interrupt Mask (LIM) register can be used to mask interrupts from the Thor-2 devices towards the LPU. Clearing the register masks all the device interrupts and setting the register enables them. The macros for the LIM register bits are defined as follows:

Definition

```
#define THOR_LIM_CPU      0x01     // Mask interrupts from CPU
#define THOR_LIM_MUN      0x02     // Mask interrupts from HDLC Controller
#define THOR_LIM_FA0      0x04     // Mask interrupts from Line Interface #0
#define THOR_LIM_FA1      0x08     // Mask interrupts from Line Interface #1
#define THOR_LIM_FMI      0x10     // Mask interrupts from Time-Space Switch
#define THOR_LIM_DT0      0x20     // Mask interrupts from DTMF Transceiver #0
#define THOR_LIM_DT1      0x40     // Mask interrupts from DTMF Transceiver #1
```

See Also

```
drvEnableLpuIntr()
drvDisableLpuIntr()
THOR_CIM_XXX   // CPU Interrupt Mask
```

10.2.7 Frame Header - *ThorFrameHeader*

Synopsis

The Thor driver adds a header to each received message. The header includes information like the status of the message, the length of the message, the time when the message was received, and the source of the message. Note that a frame can be any type of message, e.g. an HDLC frame, a status code, or a DTMF digit. The type of message is determined by *fmType* and the source is determined by *fmSrc*.

Definition

```
typedef struct {
  short          boardNo; // Board number of the received message
  ThorSrcType    fmSrc;   // Defines which device or pipe that is
                          // the source of the message.
  ThorFrameType  fmType;  // Type of the received message
  Byte  fmSeqNo;          // Sequence number. All the received
                          // messages are assigned a sequence
                          // number by the driver.
                          // Note: Wraps after 256 messages
  Word fmLength;     // Length of the received message..
  Byte  fmStatus;    // Status of the received message.
  Byte  hour;        // Time of reception: Hour.
  Byte  min;         // Time of reception: Minute.
  Byte  sec;         // Time of reception: Second.
  Byte  csec;        // Time of reception: Hundreds of second.
} ThorFrameHeader;
```

See Also

drvRead()
IOCTL_DRV_READ
thorRead()

10.2.8 Message Source - *ThorSrcType*

Synopsis

The Message Source Type indicates the source of the message reported to the Driver. A message can be initiated from one of the HDLC Pipes (typically an incoming HDLC frame) or from one of the physical devices on the board (typically a status message). See also *ThorFrameType.*

Definition

```
typedef enum {
  THOR_SRC_PIPE0 = 0,   // Pipe 0
  THOR_SRC_PIPE1 = 1,   // Pipe 1
  // .
  // .
  // .
  THOR_SRC_PIPE31 = 31, // Pipe 31
  THOR_SRC_CTL = 64,    // HDLC Controller
  THOR_SRC_LI0 = 65,    // Line Interface 0
  THOR_SRC_LI1 = 66,    // Line Interface 1
  THOR_SRC_DT0 = 67,    // DTMF transceiver 0
  THOR_SRC_DT1 = 68,    // DTMF transceiver 1
  THOR_SRC_CD0 = 69,    // Codec 0
  THOR_SRC_CD1 = 70,    // Codec 1
  THOR_SRC_LPU = 71,    // Local Processing Unit (on-board processor)
  THOR_SRC_TSS = 72,    // Time-Space Switch
  THOR_SRC_DRV = 128    // Driver (SW)
} ThorSrcType;
```

See Also

ThorFrameHeader
ThorFrameType

10.2.9 Frame Type - *ThorFrameType*

Synopsis

Indicates the type of the received message. The Thor-2 board passes information to the driver in the form of messages. The messages can be one of three types: an hdlc frame from a hdlc pipe, a status message from a device on the board, or a message indicating a received DTMF tone.

Definition

```
typedef enum {
  THOR_FM_HDLC     =  0,  // HDLC frame from a pipe
                          // (pipeNo is valid)
  THOR_FM_STAT     =  1,  // Status message (1 octet)
                          // from any device on the board
  THOR_FM_DTMF     =  2,  // DTMF tone detected at DTMF chip;
                          // 1 ASCII character
  THOR_FM_BRS      =  3,  // Bit-Robbed Signalling information; 4 bytes; The
                          // 3 Least significant bytes contains bit robbing
                          // information for 24 DS0 channels. In ESF format
                          // this will be data from a complete multiframe
                          // (24 frames). In F12 and F72 formats the data is
                          // extracted every 12 frames. The THOR_FM_BRS is
                          // only being reported if any of the bit robbing
                          // signalling channels have changed since it was
                          // last reported. The most significant byte
                          // (fmBuf[3]) specifies the bit robbing signalling
                          // channel (0=A, 1=B, and so on).
 THOR_FM_SS7_FISU = 4     // An FISU change has been detected on a SS7 receive
                          // pipe 4 Bytes: 3 least significant octets contain
                          // the new FISU
} ThorFrameType;
```

See Also

ThorFrameHeader

ThorSrcType

10.2.10 Frame End Codes

Synopsis

Frame End Codes (FECs) are used to indicate the status of the received messages in *ThorFrameHeader.*

Definition

```
#define THOR_FEC_OK    0x00  // Good message (no errors)
#define THOR_FEC_ROF   0x01  // An overflow of the internal buffer
                             // in the HDLC controller has occurred
                             // and data has been lost.
#define THOR_FEC_RA    0x02  // The received frame was aborted by an
                             // flag 0x7F, or by a Receive Abort cmd,
                             // or by a Fast Receive Channel cmd.
#define THOR_FEC_LFD   0x04  // Set if message is longer than the
                             // buffer supplied by the application
                             // program, or if a message longer
                             // than MFL (Maximum Frame Length)
                             // was received.
#define THOR_FEC_NOB    0x08 // Set if the bit content of the received
                             // frame was not divisible by 8.
#define THOR_FEC_CRCO  0x10  // Set if a frame with a CRC error was
                             // detected.
#define THOR_FEC_LOSS   0x20 // Three contiguous frames with errors
                             // in the synchronization pattern
                             // were detected.
#define THOR_FEC_SF      0x40 // A frame of less than 32 bits between
                             // start flag and end flag (or abort
                             // flag) was received.
#define THOR_FEC_OVERFLOW  0x80 // Set if the application is too slow
                             // reading messages from the driver.
                             // Messages are lost.
```

See Also

`ThorFrameHeader`

10.2.11 Frame Fill Type - *ThorFrameFillType*

Synopsis

The Frame Fill Type is used to indicate what type a fillers are send (all ones of HDLC Flags) between the HDLC frames.

Definition

```
typedef enum {
  THOR_FFT_FLAGS,       // Send HDLC flags between the frames.
  THOR_FFT_ALL_ONES,    // Send ones between frames.
  THOR_NO_FFT
} ThorFrameFillType;
```

10.2.12 Driver Parameters - ThorDriverT

Synopsis

The Thor-2 driver needs certain information to communicate with the Thor-2 board. this information is found in the Windows registry in Windows 95 and Windows NT drivers. For the DOS driver, this information must be provided by the application.

Definition

```
typedef struct {
  short   irq;                          // IRQ used
  short   ioBaseAddr;                   // I/O Base Address used
  short   noOfBoards;                   // Number of Boards Installed
  Uint    infoBufferSize;               // Size of the message FIFO
  Ulong   memWinOffset[BOARD_PER_PC];   // Memory Window Offset for Board X
  Ulong   memWinSize[BOARD_PER_PC];     // Memory Window Offset for Board X
  Word    ioWinOffset[BOARD_PER_PC];    // I/O Window Offset for Board X
  Word    ioWinSize[BOARD_PER_PC];      // I/O Window Size for Board X
  Ulong   brdDramSize[BOARD_PER_PC];    // DRAM memory installed on board
} ThorDriverT;
```

10.2.13 Return Codes - *ThorRc*

Synopsis

Return Codes are used to return the execution result from the API functions. The Return code can be converted into a corresponding Error message with the *drvThorRc2Str()* or *thorGetErrMsg()* functions.

Definition

```
#define THOR_RC_MAX  89          // Largest possible Return Code
typedef enum {
  THOR_UNDEFINED          =  0,  // The driver does not know what to return
                                 // for this request
  THOR_SUCCESS            =  1,  // All OK, no errors.
  THOR_INVALID_BOARD_TYPE =  3,  // Supplied Board type is not supported by
                                 // by this driver.
  THOR_INVALID_BOARD_NO   =  4,  // Driver was supplied a Board number which
                                 // is not allowed (should be 0<=NO<=3).
  THOR_INVALID_LI_NO      =  5,  // Driver was supplied a Li number which
                                 // is not allowed (should be 0<=NO<=1).
  THOR_FLASH_CONFIG_ERR   =  6,  // The Flash memory did not configure correctly
  THOR_INVALID_IRQ_NO     =  7,  // Driver was supplied an IRQ number which
                                 // is not allowed (Not supported by
                                 // the board)
  THOR_INVALID_ADDR       =  8,  // Driver was supplied an I/O-address which
                                 // is not allowed (Not supported by the
                                 // Board)
  THOR_NOT_SETUP          =  9,  // Functions has been called without a proper
                                 // configuration of the driver.
  THOR_CORRUPT_FPGA_FILE  = 10,  // The FPGA configuration is corrupt
  THOR_NO_FRAMES          = 11,  // No full frames received and ready for
                                 // reading.
  THOR_TX_BUSY            = 12,  // Transmitter not ready. Transmission of the
                                 // previous frame has not been completed.
  THOR_TIMEOUT            = 13,  // Function timed out before the expected
                                 // response was obtained from ISR
  THOR_L1_OK              = 14,  // Physical Layer (L1) is up (synchronous
                                 // state).
  THOR_L1_DOWN            = 15,  // Physical Layer (L1) is down,
                                 // no connection.
  THOR_BAD_CHIP           = 16,  // Line Interface (FALC54 chip) did not
                                 // respond correctly to a command
  THOR_NO_BOARD           = 17,  // No board, or IO base address mismatch
                                 // between board and config file
```


```
 THOR_DTMF_NO_TONE       = 18,  // No tone has been detected by the DTMF chip
 THOR_LI_INIT_FAILURE     = 19, // A device on the board failed to configure
 THOR_FPGA_FILE_NOT_FOUND = 20, // Could not open the FPGA configuration file
 THOR_WRONG_CONTEXT      = 21,  // Requested action is not valid in current
                                // context. Maybe the function call sequence
                                // was wrong
 THOR_OUT_OF_MEMORY      = 22,  // Not enough memory to start the driver
 THOR_FPGA_NOT_LOADED    = 23,  // FPGAs were not loaded prior to
                                // constructing the driver
 THOR_HDLC_AR_BUSY       = 24,  // The HDLC controller is already busy with
                                // an Action Request.
 THOR_TSS_INVALID_PCM_HW = 25,  // The Highway number is not one of the 12 PCM
                                // highways.
 THOR_TSS_INVALID_CHANNEL = 26, // The channel number is out of range. Accepted
                                // range is 0 <= channel <= 31.
 THOR_TSS_INVALID_TIMING_MODE = 27, // The selected timing mode is not valid
                                // in the current Thor-2 driver.
 THOR_MSG_TOO_LONG       = 28,  // The message (frame) is too long for the
                                // current driver configuration.
 THOR_FLASH_ERASE_ERR    = 29,  // The Flash did not erase correctly
 THOR_FLASH_WRITE_ERR    = 30,  // The Flash did not write correctly
 THOR_WRONG_CONFIG_VER  = 31,   // Thor-2 was configured using an
                                // incompatible version of T2CONFIG
 THOR_FILE_NOT_FOUND     = 32, // Could not open the specified file
 THOR_NON_DEFAULT_LI     = 33, // Li reports non default register values
 THOR_HDLC_INIT_FAILURE  = 34, // HDLC controller failed to initialize
 THOR_INVALID_CODEC_NO   = 35, // Invalid Codec number
 THOR_CD_COMM_FAILURE    = 36, // Codec communication failure
 THOR_FLASH_BAD_ADDR     = 37, // Invalid Flash Address
 THOR_FLASH_BAD_FILE     = 38, // Could not open file
THOR_FLASH_INVALID_SECT_NO = 39, // Invalid Flash sector number
 THOR_FLASH_PRG_FAIL     = 40, // Flash programming error
 THOR_INVALID_STATE_TRANS= 41, // Invalid state transition requested
 THOR_NO_CASE_INTR       = 42, // Timeout waiting for CASE interrupt
 THOR_TOO_MANY_PIPES     = 43, // Too many configured pipes
 THOR_HDLC_INVALID_TX_STATE = 44, // The pipe is not in a valid transmit state
 THOR_HDLC_INVALID_RX_STATE = 45, // The pipe is not in a valid receive state
 THOR_DEVICE_OPEN_FAILED = 46, // Failed to open a driver device
 THOR_DEVICE_READ_FAILED = 47, // Failed to read driver device
 THOR_FLASH_TOO_LARGE    = 48, // Data was too large to be loaded into flash
 THOR_PIPE_NOT_CONFIGURED= 49, // Referenced pipe has not been configured
 THOR_ASSERT_FAILED      = 50, // Internal error. Assert failed
 THOR_DRV_CALL_FAILED    = 51, // Call to driver failed
 THOR_DRV_NOT_INITIALIZED = 52, // The driver has not been intialized
```

```
THOR_DATA_TOO_LARGE      = 53, // Data size too large for internal buffer
THOR_NO_MAINT_AUTHORIZATION = 54, // No authorization to perform
                                  // maintenance functions
THOR_LPU_BOOT_FAILED        = 55, // Local processor failed to boot
THOR_NO_IO_WIN              = 56, // The host I/O window has not been setup
THOR_NO_MEM_WIN            = 57, // The host Memory window has not been setup
THOR_BAD_BOOT_VECTOR        = 58, // Incorrect LPU Boot Vector.
                                  // Possibly Corrupt Flash memory
THOR_DTMF_BUSY              = 59, // DTMF tone sending in progress
THOR_INVALID_RETURN_CODE    = 60, // The ThorRc return code itself
                                  // is invalid
THOR_INVALID_HOST_IO_OFFSET = 61, // The host I/O offset value is invalid
THOR_INVALID_IO_WIN_SIZE    = 62, // The I/O window size is invalid
THOR_INVALID_HOST_MEM_OFFSET= 63, // The host mem offset value is invalid
THOR_INVALID_MEM_WIN_SIZE   = 64, // The memory Window size is invalid
THOR_LI_INVALID_ALARM_TYPE  = 65, // The Provided LI Alarm type is not valid
THOR_LI_INVALID_CLOCK_MODE  = 66, // The provided LI clock mode is not a
                                  // valid mode
THOR_LI_INVALID_RESYNC_OPTION = 67, // The provided LI Auto Resynchronization
                                 // configuration option is not a valid option
THOR_LI_INVALID_TRANSMIT_LINE_CODE = 68, // The provided LI Transmit line
                                         // code is not a valid code
THOR_LI_INVALID_RECEIVE_LINE_CODE = 69,  // The provided LI Receive line code
                                         // is not a valid code
THOR_LI_INVALID_AIS_DETECTION_OPTION = 70, // The provided LI AIS detection
                                           // option is not a valid option
THOR_LI_INVALID_TRANSMIT_FRAME_FORMAT = 71, // The provided LI Transmit Frame
                                            // Format is not a valid format
THOR_LI_INVALID_RECEIVE_FRAME_FORMAT = 72,  // The provided LI Receive Frame
                                            // Format is not a valid format
THOR_LI_INVALID_HDB3_ERROR_OPTION = 73, // The provided LI HDB3 Error
                                      // Detection option is not a valid option
THOR_LI_INVALID_REGAIN_MULTI_FRAME_OPTION = 74, // The provided LI Regain
                                     // Multi Frame option is not a valid option
THOR_LI_INVALID_REMOTE_ALARM_OPTION = 75, // The provided LI Remote Alarm
                                          // option is not a valid option
THOR_LI_INVALID_TRANSMIT_POWER_OPTION = 76, // The provided LI Transmit
                                         // Power option is not a valid option
THOR_LI_INVALID_RECEIVE_EQUALIZER_OPTION  = 77, // The provided LI Receive
                                      // Equalizer option is not a valid option
THOR_LI_INVALID_SIGNALING_MODE = 78, // The provided LI Signaling mode is
                                     // not a valid mode
THOR_LI_INVALID_FRAME_FORMAT = 79, // The provided LI Frame Format is
                                   // not a valid format
```


```
  THOR_LI_INVALID_TRANSMIT_REMOTE_ALARM_FORMAT = 80, // The provided LI
                 // Transmit Remote Alarm Format is not a valid format
  THOR_LI_INVALID_RECEIVE_REMOTE_ALARM_FORMAT  = 81, // The provided LI
                 // Receive Remote Alarm Format is not a valid format
  THOR_LI_INVALID_LOOP_TYPE    = 82,  // The provided loop type is not a
                                      // valid type
  THOR_HDLC_INVALID_PIPE_NO    = 83,  // Driver was supplied a Pipe number which
                                      // is not allowed (should be 0<=NO<=31).
  THOR_LI_INVALID_MODE         = 84,  // The provided LI mode is invalid
  THOR_CD_INVALID_TX_GAIN      = 85,  // The provided Codec Transmit (TX) gain
                                      //is invalid
  THOR_CD_INVALID_RX_GAIN      = 86,  // The provided Codec Receive (RX) gain
                                      // is invalid
  THOR_CD_INVALID_LAW         = 87, // The provided Codec Coding Law is invalid
  THOR_CD_INVALID_CODE         = 88, // The provided Codec Coding code
                                    // is invalid
  THOR_INVALID_DTMF_NO         = 89,  // Invalid DTMF Number
  THOR_SIZE_TOO_LARGE          = 90,  // Size too large for internal buffers
  THOR_EXTMEM_MOVE_FAILED      = 91,  // Extended Memory Move failed
  THOR_SETUP_INCOMPLETE        = 92,  // Setup was not successfully completed
  THOR_PIPE_NO_MEM           = 93,  // This pipe can only be controlled from
                                    //the host
  THOR_INVALID_CALLBACK_FUNCTION = 94,  // The supplied callback function
                                        // is not a valid function pointer
  THOR_CALLBACK_ALREADY_SET    = 95,  // Callback function has already been set
  THOR_UNABLE_TO_CREATE_CALLBACK_THREAD = 96,  // Creation of a thread for
                                               // the callback failed
  THOR_INVALID_DIGIT           = 97,  // Unrecognized Multifrequency tone
  THOR_HDLC_NO_DATA            = 98,  // No Allocated Data
  THOR_TX_IDLE                 = 99,  // The transmitter is idle. User data
                                      // can be sent
  THOR_DATA_ID_FREE            = 100, // The data ID is free
  THOR_DATA_ID_IN_USE          = 101, // The data ID is currently in use
  THOR_WRONG_PIPE_MODE         = 102, // The pipe is configured to mode not
                                      // compatible with the requested action

} ThorRc;
```

See Also

drvThorRc2Str()

IOCTL_DRV_THOR_RC_2_STR

thorGetErrMsg()

10.2.14 Status Message - *ThorStatusType*

Synopsis

Identification for the Thor status messages. Note that for line interfaces these messages can have a different meaning depending on whether the Line Interfaces (LIs) are configured to E1 or T1 mode. The Status code can be converted into a corresponding status message with the *drvStatus2Str()* function

Definition

```
typedef enum {
  LIS_T8MS_ISF    = 0x00, // E1: receive Timeout 8 MSec
                          // T1: Incorrect Sync Format
  LIS_CASC_RSC    = 0x01, // E1: received CAS information Changed
                          // T1: Received Signalling information Changed
  LIS_CRC4_CRC6   = 0x02, // E1: receive CRC4 error
                          // T1: receive CRC6 error
  LIS__CASE       = 0x03, // E1: -
                          // T1: transmit CAS register Empty
  LIS_RDO         = 0x04, // E1: Receive Data Overflow
                          // T1: Receive Data Overflow
  LIS_XDU         = 0x05, // E1: Transmit Data Underrun
                          // T1: Transmit Data Underrun
  LIS_XLSC        = 0x06, // E1: Transmit Line Status Change
                          // T1: Transmit Line Status Change
  LIS_FAR         = 0x07, // E1: Frame Alignment Recovery
                          // T1: Frame Alignment Recovery
  LIS_LFA         = 0x08, // E1: Loss of Frame Alignment
                          // T1: Loss of Frame Alignment
  LIS_MFAR        = 0x09, // E1: MultiFrame Alignment Recovery
                          // T1: MultiFrame Alignment Recovery
  LIS_T400MS_LMFA = 0x0A, // E1: receive Timeout 400 MSec
                          // T1: Loss of MultiFrame Alignment
  LIS_AIS         = 0x0B, // E1: Alarm Indication Signal
                          // T1: Alarm Indication Signal
  LIS_LOS         = 0x0C, // E1: Loss Of Signal
                          // T1: Loss Of Signal
  LIS_RAR         = 0x0D, // E1: Remote Alarm Recovery
                          // T1: Remote Alarm Recovery
  LIS_RA          = 0x0E, // E1: Remote Alarm
                          // T1: Remote Alarm
  LIS_LMFA16_XSLP = 0x0F, // E1: Loss of MultiFrame Alignment ts16
                          // T1: Transmit SLiP indication
  LIS_AIS16_      = 0x10, // E1: Alarm Indication Signal ts16 status change
```


```
                        // T1: -
LIS_RA16_LLBSC  = 0x11, // E1: Remote Alarm ts16 status change
                        // T1: Line Loop Back Status Change
LIS_API_        = 0x12, // E1: Auxiliary Pattern Indication
                        // T1: -
LIS_SLN         = 0x13, // E1: Slip Negative
                        // T1: Slip Negative
LIS_SLP         = 0x14, // E1: Slip Positive
                        // T1: Slip Positive
LIS_ACTIVE      = 0x15, // E1: physical link ACTIVE indication
                        // T1: physical link ACTIVE indication
LIS_DEACTIVE    = 0x16, // E1: physical link DEACTIVE indication
                        // T1: physical link DEACTIVE indication
LIS_XLS         = 0x17, // E1: Transmit Line Short
                        // T1: Transmit Line Short
LIS_XLO         = 0x18, // E1: Transmit Line Open
                        // T1: Transmit Line Open
LIS_XPR         = 0x19, // Transmit Pool Ready
HDLC_UNKNOWN_INTR = 0x40,   // HDLC Unknown interrupt (should not happen)
HDLC_ARF_INTR     = 0x42,   // HDLC Action Request Failed
HDLC_ERR_RX_INTR  = 0x43,   // HDLC Protocol error; Receive direction
HDLC_ERR_TX_INTR  = 0x44,   // HDLC Host is too slow filling descriptors.
HDLC_FO_INTR      = 0x45,   // HDLC Underflow/Overflow. Internal
                            // buffer not available
HDLC_OVERFLOW_INTR = 0x46,  // HDLC Interrupt Queue overflow
HDLC_ITF_INTR     = 0x47,   // HDLC Idle/Flag change. Changed Interframe
                            // time-fill state
HDLC_SF1_INTR     = 0x48,   // HDLC Short Frame Interrupt (7E00 0000 007E
                            // (for CRC32) was detected)
HDLC_SF2_INTR     = 0x49,   // HDLC Short Frame
HDLC_IFC_INTR     = 0x4A,   // HDLC Interframe timefill character change
HDLC_SF_ERR_INTR  = 0x4B,   // HDLC Short Frame and Error Indication
HDLC_INVALID_LEN  = 0x4C,   // HDLC Invalid Message
HDLC_HI_RX_INTR   = 0x4D,   // HDLC Host Initiated Interrupt -
                            // Receive direction
HDLC_HI_TX_INTR   = 0x4E,   // HDLC Host Initiated Interrupt -
                            // Transmit direction
HDLC_INVALID_PIPE_INTR = 0x4F,  // HDLC Interrupt from an  unconfigured
                                //  pipe
HDLC_HOLD_FAILED  = 0x50,   // HDLC Failed to put a Tx pipe in hold state
DRV_FIFO_OVERFLOW = 0x80,   // Driver Internal FIFO overflow
DRV_END_OF_DRV_STATUS = 0xAF,
LPU_HW_WATCHDOG   = 0xB0,   // LPU HW not functioning
LPU_DOS_WATCHDOG  = 0xB1,   // LPU ROM-DOS crash
```

```
  LPU_APP_WATCHDOG  = 0xB2,   // LPU Application crash

} ThorStatusType;
```

See Also

drvStatus2Str()

IOCTL_DRV_STATUS_2_STR

10.3 Driver Specific Definitions - *drvdef.h*

10.3.1 Board Configuration Data - *ThorConfigT*

Synopsis

The Board configuration data is stored persistantly in the flash. The information stored is defined with *ThorConfigT.*

Definition

```
typedef struct {
  Word                structVer;  // Should be incremented with every change of
                                  // this struct
  Word                serialNo;   // Serial number of the board. Should
                                  // match the data in the GDS.
  short               clkSrc;     // Clock Source
  LiConfigOptionsT    li[2];      // Options for Li's (E1 and T1 options)
} ThorConfigT;
```

See Also

drvReadConfigData()
IOCTL_DRV_READ_CONFIG_DATA
drvWriteConfigData()
IOCTL_DRV_WRITE_CONFIG_DATA

10.3.2 Driver Mode - *DrvModeT*

Synopsis

Specifies the mode the driver is operating. Only Stand-alone (DM_STANDALONE) supported in this driver version.

Definition

```
typedef enum {
  DM_STANDALONE, // This driver is handling everything. There is no
                 //other Thor-2 driver running. Can be used on either LPU or CPU
  DM_LOW_LAYER_LPU, // (LPU only) The LPU is handling the lower protocol
                    // layers (implies that the CPU driver must be
                    // in DM_HIGH_LAYER_CPU mode)
  DM_HIGH_LAYER_CPU, // (CPU only) The CPU is handling the upper
                     //  protocol layers (implies that the LPU driver
```

```
                        // must be in DM_LOW_LAYER_LPU mode)
    DM_PARALLEL_LPU,    // (LPU only) The LPU driver is handling some
                        // pipes. The CPU driver is handling some other
                           // pipes (the tasks are totally independent).
    DM_PARALLEL_CPU        // (CPU only) The CPU driver is handling some
                           // pipes. The LPU driver is handling some other pipes
                           // (the tasks are totally independant).
} DrvModeT;
```

10.4 Line Interface (LI) Specific Definitions - *lidef.h*

10.4.1 Operation Modes - *LiMode*

Synopsis

The operation mode enumerated type is used to indicate whether the Line Interface is configured in T1 or E1 mode.

Definition

```
typedef enum {
  THOR_T1,         // The line interface is configured to operate in T1 mode.
  THOR_E1,         // The line interface is configured to operate in E1 mode.
  THOR_NO_IMODE
} LiMode;
```

10.4.2 Alarm Type - *LiAlarmType*

Synopsis

The Alarm type is used to specify the type of the alarm. Thor-2 can send 3 different alarms towards the remote end: Auxiliary Pattern (AUXP), Alarm Indication Signal (AIS), and Remote Alarm Indication (RAI or Yellow Alarm).

Definition

```
typedef enum {
  LI_AUXP = 0,              // Auxiliary Pattern (AUXP)
  LI_AIS  = 1,              // Alarm Indication Signal (AIS)
  LI_RAI  = 2,              // Remote Alarm Indication (RAI) (E1 only)
  LI_ALARM_SIMULATION = 3,  // Initiates internal error simulation of AIS,
                            // loss of signal, loss of synchronization,
                            // auxiliary pattern indication, slip, framing
                            // errors, CRC errors, and code violations.
} LiAlarmType;
```

See Also

10.4.3 Clock mode - LiCLkMode

Synopsis

The LiClkMode Type is used to indicate whether the Line Interface is configured to be a clock master or a clock slave.

Definition

```
typedef enum {
  LI_MASTER = 0,
  LI_SLAVE = 1
} LiClkMode;
```

10.4.4 Bit Robbing Data - LiBrData

Synopsis

The LiBrData Type is a parameter to *liSetBitRobData()* which specifies the signaling data (24 bits per signaling channel) to be transmitted next. This is used in CAS_BR (Channel Associated Signaling - Bit Robbing) signaling mode in T1.

The A and B channels are used in frame formats F12, ESF, and F72. The C and D channels are only used in the ESF frame format. The struct holds 24 bits for each channel; The least significant bit is transmitted first (in channel 1, frame 1). This 24 bit value will be repeated unless liSetBitRobData() is called again with different values in this struct.

Definition

```
typedef struct {
  Ulong chA;
  Ulong chB;
  Ulong chC;  // Only used in ESF format
  Ulong chD;  // Only used in ESF format
} LiBrData;
```

See Also

```
liAlarmOn()
IOCTL_LI_ALARM_ON
liAlarmOff()
IOCTL_LI_ALARM_OFF
```

10.4.5 Li Configuration Options - *LiConfigOptionsT*

Synopsis

The Line Interface (LI) configuration options are passed to the *liConfigure()* function to configure the Line Interface for either T1 or E1.

Definition

```
typedef struct {                // Used as Flash stored config options
  short   defaultMode;          // E1, T1
  LiE1ConfigOptionsT e1;
  LiT1ConfigOptionsT t1;
} LiConfigOptionsT;
```

See Also

liConfigure()
IOCTL_LI_CONFIGURE
drvReadConfigData()
IOCTL_DRV_READ_CONFIG_DATA
drvWriteConfigData()
IOCTL_DRV_WRITE_CONFIG_DATA

10.4.6 T1 Specific Configuration Options - *LiT1ConfigOptionsT*

Synopsis

The T1 Configuration options are used to configure a line interface (LI) for an T1 link.

Definition

```
// Signalling Modes
typedef enum {
  LI_CCS,    // Common Channel Signalling
  LI_CAS_CC, // Channel Associated Signalling (Common channel)
  LI_CAS_BR  // Channel Associated Signalling (Bit Robbing)
} LiT1SignallingMode;

// Line Code options for T1
typedef enum {
  LI_T1_AMI = 0, // Must be 0
  LI_T1_B8ZS
} LiT1LineCode;

// Framing options for T1
```

```
typedef enum {
  LI_F12 = 0,  // 12-frame multiframe format (F12, D3/4)
  LI_F4  = 1,  //  4-frame multiframe format (F4)
  LI_ESF = 2,  // 24-frame multiframe format (ESF)
  LI_F72 = 3,  // 72-frame multiframe format (F72, remote switch mode)
} LiT1Framing;

// Loss of Frame Alignment sensitivity
typedef enum {
  LI_2_OUT_OF_4 = 0x00, // Values match the Falc register SSC1 bits
                        // and SSC0 bits
  LI_2_OUT_OF_5 = 0x08,
  LI_2_OUT_OF_6 = 0x10,
} LiT1LfaSensitivity;

// Yellow alarm format for T1
typedef enum {
  LI_YELLOW_A = 0,  // F12: bit2 = 0  in every channel
                    // ESF: pattern '1111 1111 0000 0000' in data link channel
  LI_YELLOW_B = 1,  // F12: FS bit of frame 12
                    // ESF: bit2 = 0  in every channel
} LiT1RemoteAlarmT;

// Li Configuration Options for T1
typedef struct {
  short              signalingMode;
  short              transmitLineCode;
  short              receiveLineCode;
  short              frameFormat;
  short              enableCRC6;
  short              transmitRemoteAlarmFormat;
  short              receiveRemoteAlarmFormat;
  short              autoResynchronization;
  short              lfaSensitivity;
  short              automaticRemoteAlarm;
  Byte               losSensitivity;
  Byte               losRecovery;
  Word               lineLength;
  short              transmitPower;
  short              receiveEqualizer;
  Ulong              clearChannels;
} LiT1ConfigOptionsT;
```

10.4.7 E1 Specific Configuration Options - *LiE1ConfigOptionsT*

Synopsis

The E1 Configuration options are used to configure a line interface (LI) for an E1 link.

Definition

```
// Line code options for E1:
typedef enum {
  LI_E1_AMI = 0,              // Must be 0
  LI_E1_HDB3 = 1
} LiE1LineCode;

// AIS Alarm Detection Mode for E1:
typedef enum {
  LI_AIS_ETS300233 = 0,   // AIS alarm will be detected according to ETS300233
  LI_AIS_G775 = 1        // AIS alarm will be detected according to CCITT G.775
} LiE1AisDetectModeT;

// Framing options for E1
typedef enum {
  LI_DOUBLE_FRAME = 0,
  LI_CRC4_MULTIFRAME = 1,
  LI_CRC4_MULTIFRAME_G706 = 2 // CRC4 Multiframe format with modified CRC4
                              // Multiframe alignment algorithm (Interworking
                              // according to CCITT G.706 Annex B)
} LiE1Framing;

// Li Configuration Options for E1
typedef struct {
  short          transmitLineCode;
  short        receiveLineCode;
  short        transmitFrameFormat;
  short        receiveFrameFormat;
  short        aisDetection;
  Word         siBits;
  Word         saBits;
  short        extendedHDB3errorDetection;
  short        automaticRegainMultiframe;
  short        autoResynchronization;
  short        automaticRemoteAlarm;
  Byte         losSensitivity;
  Byte         losRecovery;
  Word         lineLength;
```

```
    short        transmitPower;
    short        receiveEqualizer;
} LiE1ConfigOptionsT;
```

10.5 HDLC Specific definitions - *hdlcdef.h*

10.5.1 HDLC definitions

```
#define HDLC_TIMESLOTS_MAX              32   // Number of time-slots supported
                                             // by the HDLC Controller
#define HDLC_CHANNELS_MAX               32   // Number of channels (pipes)
                                            // supported by the HDLC Controller
#define HDLC_NO_PATTERNS_MAX            32   // Number of Patterns supported by
                                             // hdlcSendPattern() function
#define HDLC_FRAME_LENGTH_MAX         8191   // Maximum HDLC Frame Length
                                            // supported by the HDLC Controller
```

10.5.2 Pipe Configuration Options - *HdlcPipeOpts*

Synopsis

The HdlcPipeOpts is used to specify the options for the pipes to be configured (see hdlcInitPipe()).

Definition

```
// Pipe configuration options:
typedef struct {
  Byte txFillMask[MUNICH32_TIMESLOT_MAX]; // Array 32 bytes long indicating
                                          // which Tx timeslots (and which bits)
                                         // that should be included in the channel
  Byte rxFillMask[MUNICH32_TIMESLOT_MAX]; // Array 32 bytes long indicating
                                           // which Rx timeslots (and which bits)
                                         // that should be included in the channel
  HdlcPipeMode mode;                 // TMA, TMB, TMR, V110_X30 or HDLC
  ThorFrameFillType frameFillType;   // HDLC mode only
                                     // Character to send between HDLC frames
                                     // THOR_FFT_FLAGS (0x7E) or
                                     // THOR_FFT_ALL_ONES (0xFF)
                                     // All other modes => Use THOR_NO_FFT
  Byte tflag;                        // Transparent Mode Flag (only in TMA mode)
                                    // These 8 bits constitute the fill code for
                                     // flag stuffing and flag filtering.  These
                                     // must be set to 0x00 if subchanneling is
                                     // used in TMA.
                                     // Note:
                                     // All other modes => set to 0x00
  HdlcTxRate txRate;                 // Transmission Rate (V.110 and X.30 modes)
                                     // All other modes => use the value TR_NA.
  short interFrameTimeFillNum;       // Minimum No of interframe time-fill chars
```

```
                                    // (0x7E) between 2 HDLC frames
                                    // (1=shared flags, 2=non-shared flags)
                                    // Min value is 1. Max is 1024.
                                    // Valid for all modes
  Bool bitInversion;                // When TRUE all bits are inverted
                                    // Valid for all modes
  Bool useCrc32;                   // HDLC mode only => TRUE:CRC32, FALSE:CRC16
                                    // All other modes => Don't care
  Bool suppressCrcGen;              // HDLC mode only => When TRUE the
                                    // CRC generation in the
                                    // Transmit direction is suppressed, and
                                    // the CRC bits are appended to the
                                    // messages in the receive direction
                                    // All other modes => set to FALSE
  Bool flagAdjustment;              // HDLC mode: When TRUE the number of
                                    // interframe time-fill characters is
                                    // interFrameTimeFillNum - 1 - 1/8*X
                                    // where X is the number of zero
                                    // insertions in the frame preceding
                                    // the interframe time-fill.
                                    // TMA mode: When TRUE the value tflag is
                                    // treated a FLAG and it is removed from
                                    // the rx and tx data streams . When FALSE
                                    // the pipe runs fully transparent (without
                                    // framing and without flags)
                                    // All other modes
} HdlcPipeOpts;
```

TMR mode:

- Transparent transmission/reception with GSM 08.60 frame structure
- Automatic 0x0000 flag generation/detection
- Support for 40, 39.5, and 40.5 octet frames
- Error detection (non octet frame content, short frame, long frame)

For the TMR mode, use the following Pipe options:

```
  pipeOpts.mode                  = PM_TMR;
  pipeOpts.frameFillType         = THOR_NO_FFT; // 0 required for TMR
  pipeOpts.tflag                 = 0x00;  // N/A for TMR mode; Set to 0x00
  pipeOpts.txRate                = TR_NA; // Required for TM mode
  pipeOpts.interFrameTimeFillNum = 10;    // Any number 1>=x>=1024 is selectable
  pipeOpts.bitInversion          = FALSE; // Could be either TRUE of FALSE
  pipeOpts.useCrc32              = FALSE; // N/A for TMR mode; Set to FALSE
  pipeOpts.suppressCrcGen        = FALSE; // N/A for TMR mode; Set to FALSE
```


```
pipeOpts.flagAdjustment        = FALSE; // FALSE is required for TMR
```

TMB mode:

- Transparent transmission/reception in frames delimited by 0x00 flags
- Can use shared opening and closing flag if selected
- Flag stuffing, flag detection, flag generation in the abort case
- Error detection (non octet frame content, short frame, long frame)

For the TMB mode, use the following Pipe options:

```
pipeOpts.mode                  = PM_TMB;
pipeOpts.frameFillType         = THOR_NO_FFT; // Required for TMB
pipeOpts.tflag                 = 0x00;  // N/A for TMB mode; Set to 0x00
pipeOpts.txRate                = TR_NA; // Required for TM mode
pipeOpts.interFrameTimeFillNum = 10;    // Any number 1>=x>=1024 is selectable
pipeOpts.bitInversion          = FALSE; // Could be either TRUE of FALSE
pipeOpts.useCrc32              = FALSE; // N/A for TMB mode; Set to FALSE
pipeOpts.suppressCrcGen        = FALSE; // N/A for TMB mode; Set to FALSE
pipeOpts.flagAdjustment        = FALSE; // FALSE is required for TMB
```

TMA mode:

- Slot synchronous transparent transmission/reception without frame structure
- Bit overwrite with fill/mask flags
- Flag stuffing, flag detection, flag generation in the abort case with programmable flag

For the TMA mode, use the following Pipe options:

```
pipeOpts.mode                  = PM_TMA;
pipeOpts.frameFillType         = THOR_NO_FFT; // Required for TMA
pipeOpts.tflag                 = 0x00;  // Since we flagAdjustment==FALSE,
                                        // tflag must be 0x00
pipeOpts.txRate                = TR_NA; // Not used in TMA mode
pipeOpts.interFrameTimeFillNum = 4;
pipeOpts.bitInversion          = FALSE; // Could be either TRUE of FALSE
pipeOpts.useCrc32              = FALSE; // N/A for TMB mode; Set to FALSE
pipeOpts.suppressCrcGen        = FALSE; // N/A for TMB mode; Set to FALSE
pipeOpts.flagAdjustment        = FALSE; // Could be either TRUE of FALSE
```

HDLC mode:

- Automatic flag detection and transmission
- Can use shared opening and closing flag if selected

- Detection of interframe-time-fill change, generation of interframe-time-fill 1's or flags.
- Zero bit insertion
- Flag stuffing and flag adjustment for rate adaptation
- CRC generation and checking (16 or 32 bits)
- Error detection (abort, long frame, CRC error, 2 categories of short frames, non-octet frame content)

For the HDLC mode, use the following Pipe options:

```
pipeOpts.mode                  = PM_HDLC;
pipeOpts.frameFillType         = THOR_FFT_ALL_ONES; // or THOR_FFT_FLAGS
pipeOpts.tflag                 = 0x00;  // N/A for HDLC mode; Set to 0x00
pipeOpts.txRate                = TR_NA; // Not used in HDLC mode
pipeOpts.interFrameTimeFillNum = 2;     // Any number 1>=x>=1024 is selectable
pipeOpts.bitInversion          = FALSE; // Could be either TRUE of FALSE
pipeOpts.useCrc32              = FALSE; // Could be either TRUE of FALSE
pipeOpts.suppressCrcGen        = FALSE; // Could be either TRUE of FALSE
pipeOpts.flagAdjustment       = FALSE; // Could be either TRUE of FALSE
```

10.5.3 Memory Allocation for Channels - *HdlcBufAllocT*

Synopsis

The *HdlcBufAllocT* specifies how much buffer memory (in Thor-2 on-board DRAM) is allocated for each pipe.

Definition

```
typedef struct {
  Ulong    totalBufSize;  // Total receive or transmit buffer size for
                          // this channel.
  Word bytesPerDescr;     // Must be a multiple of 4 and in the
                          // range 4<=x<=8188.
} HdlcBufAllocT;
```

10.5.4 Data Patterns to be sent with hdlcSendPattern() - *HdlcDataPatternT*

Synopsis

The *HdlcDataPatternT* is used a parameter to *hdlcSendPattern()*. It specifies what type of pattern that *hdlcSendPattern()* should transmit.

Definition

```
typedef struct {
  Byte *data;         // Pattern (frame) to be sent
  short dataLen;      // Length of the pattern (frame)
  Bool  dataEndFlag;  // TRUE if the pattern (frame) should end with a frame end
                      // character. Normally FALSE for TMA mode and TRUE
                      //otherwise.
  short interFrameTimeFillNum; // No of interframe time-fill characters
                               // (0x7E) after this 2 pattern (frame)
                               // (1=shared flags, 2=non-shared flags)
                              // Range: 1 <= x <= 4096
} HdlcDataPatternT;
```

10.6 Flash Specific Defines - *flashdef.h*

10.6.1 Flash Addresses

Definition

```
// Flash Top Address (64M):
#define FLASH_TOP_ADDR   0x3FFFFFFUL

// Flash Start Address (Top of local memory - 512 KBytes):
#define FLASH_BASE_ADDR  (FLASH_TOP_ADDR - 0x80000UL + 1)

// Location for the boot routine (must be within last 64KBytes of memory)
#define FLASH_BOOT_ADDR  (FLASH_TOP_ADDR - 0x10000UL + 1)

// Location where he LPU starts executing after reset:
#define FLASH_RESET_ADDR   0x3FFFFF0UL

// Flash Sector offsets from FLASH_BASE_ADDRESS
#define FLASH_SA0      0x00000UL     //  0th sector (64KB)
#define FLASH_SA1      0x10000UL     //             (64KB)
#define FLASH_SA2      0x20000UL     //             (64KB)
#define FLASH_SA3      0x30000UL     //             (64KB)
#define FLASH_SA4      0x40000UL     //             (64KB)
#define FLASH_SA5      0x50000UL     //             (64KB)
#define FLASH_SA6      0x60000UL     //             (64KB)
#define FLASH_SA7      0x70000UL     //  7th sector (32KB @ 0x3FF0000)
                                     //  LPU Boot strap
#define FLASH_SA8      0x78000UL     //  8th sector (8KB)
#define FLASH_SA9      0x7a000UL     //  9th sector (8KB @ 0x3FFA000)
                                     //  Thor configuration data
#define FLASH_SA10     0x7c000UL     //  10th sector (16KB)
                                     //  Boot strap addresses @ 0x3FFFFF0

#define FLSH_MAX_USR_SECT_NO  6      // Highest sector number that the user
                                     // can erase.
#define FLSH_MAX_SECT_NO    10       // Max sector number (0 count)
```

10.6.2 Thor-2 Maintenance Data - *MaintDataT*

Synopsis

The Thor-2 maintenance data contains revision information on the various programmable devices on the board. The Maintenance data can be read by an

application with *flshReadMaintData()* or *IOCTL_FLSH_READ_MAINT_DATA* functions. The data stored in the flash maintenance sector is specfied by *MaintDataT.*

Definition

```
typedef struct {
  // Add more struct members here and change FLASH_MAINT_START_ADDR
  // Add 4 bytes at a time
  //
  // Start maint address 0x3FFFFD4
  Byte  barRev[4];       // Revision of the Bus Arbiter EPLD
  Byte  madRev[4];       // Revision of the Memory Access Device EPLD
  Byte  sadRev[4];       // Revision of the Startup Access Device EPLD
  Byte  ladRev[4];       // Revision of the Local Access Device EPLD
  Byte  rscRev[4];       // Revision of the Reset Controller EPLD
  Byte  lpuBootRev[4];   // Revision of LPUBOOT.ASM
  Ulong comClkFix;       // WDTOUT#->COMCLK airwire
  // Address 0x3FFFFF0
  Byte  lpuResVect[12];    // First LPU instruction after a reset
  // Address 0x3FFFFFC
  Ulong muResVect;         // Address to the Munich CCS area
} MaintDataT;
```

See Also

```
flshReadMaintData()
IOCTL_FLSH_READ_MAINT_DATA
```

10.7 Codec Specific Definitions - *cddef.h*

10.7.1 Coding Law - *CdLawT*

Synopsis

The *CdLawT* type specifies the different available PCM coding laws.

Definition

```
typedef enum {
  CD_U_LAW,                // North American u-law
  CD_A_LAW                 // International A-law
} CdLawT;
```

10.7.2 Code Assignment - *CdCodeT*

Synopsis

The *CdCodeT* specifies the different available code assignment schemes.

Definition

```
typedef enum {
  CD_CCITT_CODE,           // u-law: true sign, inverted magnitude,
                           // a-law: true sign, alternate digit inversion
  CD_SIGN_MAGNITUDE_CODE   // sign-magnitude code assignment (independent
                           // of CdLawT)
} CdCodeT;
```

10.7.3 Digital Gain - *CdDigitalGainT*

Synopsis

The CdDigitalGainT type is used to specify the amount of gain.

Definition

```
typedef enum {
  CD_MINUS_24_DB = 0,      // -24 dB
  CD_MINUS_21_DB = 1,
  CD_MINUS_18_DB = 2,
  CD_MINUS_15_DB = 3,
  CD_MINUS_12_DB = 4,
  CD_MINUS_9_DB = 5,
```


```
  CD_MINUS_6_DB = 6,
  CD_MINUS_3_DB = 7,
  CD_0_DB = 8,
  CD_PLUS_3_DB = 9,
  CD_PLUS_6_DB = 10,
  CD_PLUS_9_DB = 11,
  CD_PLUS_12_DB = 12,
  CD_PLUS_15_DB = 13,
  CD_PLUS_18_DB = 14,
  CD_PLUS_21_DB = 15
} CdDigitalGainT;
```

10.8 DTMF Specific Definitions - *dtmfdef.h*

10.8.1 DTMF Tone Storage Options - *DtmpOptT*

Synopsis

The *DtmfOptT* type is used specify how the detected DTMF tones will be stored. The options are: Ignore the digits, store only the latest one detected, store all the detected digits in a FIFO.

Definition

```
typedef enum {
  DT_DISABLE,       // No DTMF detection
  DT_DETECT_LAST,   // Detect DTMF tones but store only the last detected digit
  DT_DETECT_STORE   // Detect DTMF tones, and store all detected digits
                    // in the FIFO
} DtmfOptT;
```

See Also

```
dtmfEnable()
IOCTL_DTMF_ENABLE
```

10.8.2 Sending Duration - *DtmfDurationT*

Synopsis

The *DtmfDurationT* type is used to specify whether the transmitted tones will be send indefinitely or in bursts.

Definition

```
typedef enum {
  DT_DURATION_INFINITY = 0,  // Send DTMF tone forever
  DT_DURATION_BURST          // 51 ms Burst and Pause duration
} DtDurationT;
```

11. Low-Level API

11.1 Driver functions - *driver.h*

11.1.1 drvBoardExistence()

Synopsis

Checks if a Thor-2 board exists at a specified I/O-address.

Definition

```
ThorRc drvBoardExistence(
  short boardNo,            // Board number.
  short aIoBaseAddr         // I/0-base address configured with the Dip-switch
);
```

Returns

```
THOR_SUCCESS            // a THOR-2 board was found
THOR_NO_BOARD           // no THOR-2 board was found
THOR_INVALID_BOARD_NO   // Supplied board number is not valid
THOR_INVALID_ADDR       // Supplied Address is not valid
```

See Also

```
thorBoardExistence()
```

11.1.2 drvCmpMemBlock()

Synopsis

Compares the contents of a block of on-board memory with a block of host memory.

Definition

```
short drvCmpMemBlock(
  short boardNo,    // Board number
  Ulong lmbAddr,    // On-board Memory Starting Address (flat model) for the
                    // Comparison
  Byte *buf,        // The buffer to be compared with the on-board memory
  Uint  count       // Number of bytes to be compared
);
```

Returns

```
== 0      // Buffers are equal
!= 0   // Buffers are different
```

See Also

```
drvReadMemBlock()
drvWriteMemFill()
drvWriteMemBlock()
```

11.1.3 drvDisableCpuIntr()

Synopsis

Disables Thor-2 device interrupts towards the CPU according to the cpuIntrMask. The mask has one bit per device (see Thor-2 Technical Description). A '1' in a mask bit will disable the specific device from interrupting the CPU. A '0' will keep the current value of the mask bit. To enable the interrupts, use *drvEnableCpuIntr()* function.

Definition

```
ThorRc drvDisableCpuIntr(
  short boardNo,        // Board number.
  short cpuIntrMask     // Interrupt Mask
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_INVALID_BOARD_NO   // Supplied board number is not valid
```

See Also

```
drvEnableCpuIntr()
drvEnableLpuIntr()
drvDisableLpuIntr()
```

11.1.4 drvDisableLpuIntr()

Synopsis

Disables interrupts from on-board devices towards the LPU. The mask has one bit per device (see Thor-2 Technical Description). A '1' in a mask bit will disable the specific device from interrupting the LPU. A '0' will keep the current value of the mask bit. To enable the interrupts, use *drvEnableLpuIntr()* function.

Definition

```
ThorRc drvDisableLpuIntr(
  short boardNo,        // Board number.
  short lpuIntrMask     // A '1' in the mask will enable the device
                        // interrupt to the CPU. A '0' will keep the current
                        // value of the bit.
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_INVALID_BOARD_NO   // Supplied board number is not valid
```

See Also

```
drvEnableLpuIntr()
drvEnableCpuIntr()
drvDisableCpuIntr()
```

11.1.5 drvEnableCpuIntr()

Synopsis

Enables (unmasks) interrupts from on-board devices towards the CPU. The mask has one bit per device (see Thor-2 Technical Description). A '1' in a mask bit will enable the specific device to interrupt the CPU. A '0' will keep the current value of the mask bit. To disable the interrupts, use the *drvDisableCpuIntr()* function.

Definition

```
ThorRc drvEnableCpuIntr(
  short boardNo,        // Board number.
  short cpuIntrMask     // Interrupt Mask
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_INVALID_BOARD_NO   // Supplied board number is not valid
```

See Also

```
drvDisableCpuIntr()
drvEnableLpuIntr()
drvDisableLpuIntr()
```

11.1.6 drvEnableLpuIntr()

Synopsis

Enables (unmasks) interrupts from on-board devices towards the LPU. The mask has one bit per device (see Thor-2 Technical Description). A '1' in a mask bit will enable the specific device to interrupt the LPU. A '0' will keep the current value of the mask bit. To disable the interrupts, use the *drvDisableLpuIntr()* function.

Definition

```
ThorRc drvEnableLpuIntr(
  short boardNo,        // Board number.
  short lpuIntrMask     // Interrupt Mask
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_INVALID_BOARD_NO   // Supplied board number is not valid
```

See Also

```
drvDisableLpuIntr()
drvEnableCpuIntr()
drvDisableCpuIntr()
```

11.1.7 drvFillMem()

Synopsis

Fills a block of the on-board memory (DRAM) with a constant value.

Caveat: Only works for block sizes smaller than the memory window size.

Definition

```
ThorRc drvFillMem(
  short boardNo,    // Board number
  Ulong lmbAddr,    // Local Memory Bus Address (flat model)
  Word value,       // value will be written (Byte)
  Word count        // Number of bytes to be written
);
```

Returns

```
THOR_SUCCESS
THOR_DATA_TOO_LARGE         // Attempted to fill a memory block that is larger
```

```
                                // than the memory window size
THOR_INVALID_BOARD_NO           // Supplied board number is not valid
```

See Also

```
drvWriteMemBlock()
```

11.1.8 drvFpgaStatus()

Synopsis

Checks if the Field Programmable Gate Arrays (FPGAs) on the Thor-2 board have been configured successfully.

Definition

```
ThorRc drvFpgaStatus(
  short boardNo             // Board number.
);
```

Returns

```
THOR_SUCCESS                // the FPGAs are configured
THOR_FPGA_NOT_LOADED        // the FPGAs are not configured
THOR_INVALID_BOARD_NO       // Supplied board number is not valid
```

11.1.9 drvGetBoardStatus()

Synopsis

Return the status of the specified board.

Definition

```
ThorRc drvGetBoardStatus(
  short boardNo   // Number of the Thor-2 Board
);
```

Returns

```
== THOR_SUCCESS  // OK. The board is configured and running.
!= THOR_SUCCESS  // The board is not OK.
```

See Also

```
drvGetStatus()
```

11.1.10 drvGetStatus()

Synopsis

Return the status of the driver in use: DOS Lib, Windows 95 VxD, or Windows NT sys.

Definition

```
ThorRc drvGetStatus(
  void
);
```

Returns

```
== THOR_SUCCESS  // OK. The driver is configured and running.
!= THOR_SUCCESS  // The driver is not OK.
```

See Also

```
drvGetBoardStatus()
```

11.1.11 drvIdent()

Synopsis

Returns the identification string of the Thor-2 driver. The identification number contains the Odin TeleSystems' product number, the driver revision, and the date the driver was compiled.

Definition

```
char *drvIdent(
  void
);
```

Returns

Pointer to a static string containing the driver identification. The string is owned by the LAPI function.

See Also

```
thorIdentDriver()
```

11.1.12 drvInit()

Synopsis

Connects to the Driver and initializes the Driver internal data structures. In Windows 95 and Windows NT this function needs to be called before any other function is called. In DOS this function needs to be called after drvSetup().

Definition

```
ThorRc drvInit(
  DrvModeT drvMode // Specifies in which mode (and on which target) this
                   // driver is running
);
```

Returns

```
THOR_SUCCESS               // OK
THOR_OUT_OF_MEMORY         // Not enough memory to create the message FIFO
THOR_DEVICE_OPEN_FAILED    // Unable to connect to the driver
THOR_NO_BOARD              // No Thor-2 Board found
```

Platforms

ALL

See Also

```
thorConstructDriver()
drvSetup()
```

11.1.13 drvInitHdlc()

Synopsis

The *drvInitHdlc()* function performs a hardware reset of the HDLC controller. Initializes the HDLC memory structure.

Definition

```
ThorRc drvInitHdlc(
  short boardNo,                // Board number
  Ulong memSize,                // Total memory on the board (in bytes)
  HdlcBufAllocT txBufAlloc[],   // Array of the memory allocation for each
                                // 32 channels (transmit direction)
  HdlcBufAllocT rxBufAlloc[],   // Array of the memory allocation for each
                                // 32 channels (receive direction)
```

```
  short maxFrameLen         // Max allowed HDLC frame length (only used in HDLC,
                            // TMB and TMR modes)
  Ulong *memoryUsed         // Total amount of memory used by the HDLC
                            // controller (output from this function).
);
```

Returns

```
THOR_SUCCESS              // OK
THOR_OUT_OF_MEMORY        // Not enough on-board memory to setup
                          // the HDLC receive and transmit data structures
THOR_NO_MEM_WIN           // Memory Window has not been setup (DOS).
                          // Call drvSetupMemWin() first.
THOR_HDLC_INIT_FAILURE    // HDLC controller initialization failed
THOR_INVALID_BOARD_NO     // Supplied board number is not valid
```

See Also

```
thorResetHdlc()
```

11.1.14 drvInstallIsr()

Synopsis

Installs the Thor-2 interrupt service routine for the specified Interrupt (IRQ).

Definition

```
ThorRc drvInstallIsr(
  short aIrq
);
```

Returns

```
THOR_SUCCESS           // OK
THOR_INVALID_IRQ_NO    // The supplied IRQ number is not valid
```

Platforms

DOS

See Also

```
drvUnInstallIsr()
thorConstructDriver()
```

11.1.15 drvRead()

Synopsis

Retrieves the next received frame (an HDLC message, a device status message, or a dtmf tone) from the driver receive FIFO. Checks all the boards and all the pipes and devices for available messages.

Note: This function cannot be used with transparent pipes. Use the *drvReadTma()* function instead.

Definition

```
ThorRc drvRead(
  Byte   fmBuf[],             // Buffer into which the received message will
                              // be written. NOTE: The buffer must be
                              // allocated by the application.
  short           fmBufSize,  // Size of fmBuf[].
  ThorFrameHeader *fmHeader   // Pointer to header structure that will be
                              // filled in by the function.
                              // NOTE: The struct must be allocated by
                              // the application.
);
```

Returns

```
THOR_SUCCESS           // A received frame was retrieved successfully
THOR_NO_FRAMES         // No complete frames have been received and
                       // ready to be read.
```

See Also

```
thorRead()
drvReadTma()
hdlcWritePipe()
```

11.1.16 drvReadTma()

Synopsis

The function *drvReadTma()* fetches data from any transparent pipe if any data is available. The function checks all the boards and all the transparent pipes for received data.

Definition

```
ThorRc drvReadTma(
```

```
  Byte fmBuf[],              // Buffer into which the received message will
                             // be written. NOTE: The buffer must be
                             // allocated by the application.
  short fmBufSize,           // Size of fmBuf[].
  ThorFrameHeader *fmHeader  // Pointer to header structure that will be
                             // filled in by the function. NOTE: The struct
                             // must be allocated by the application.
);
```

Returns

```
THOR_SUCCESS          // A received frame was retrieved successfully
THOR_NO_FRAMES        // No data has been received and
                      // is ready to be read.
```

See Also

```
thorRead()
drvRead()
hdlcWritePipe()
```

11.1.17 drvReadConfigData()

Synopsis

Reads the Thor2 T1/E1 configuration data from the flash.

Note: The calling application must allocate the *ThorConfigT* data structure and pass a pointer to the *drvReadConfigData()* function. The drvReadConfigData() function copies the information from the flash memory into the provided the data structure.

Definition

```
ThorRc drvReadConfigData(
  short boardNo,        // Board number
  ThorConfigT *cfgData  // Configuration Data read (Output from this function)
                        // NOTE: ThorConfigT struct must be allocated by the
                        // application
);
```

Returns

```
THOR_SUCCESS                // OK
THOR_WRONG_CONFIG_VER       // Configuration data has a different revision
                            // than supported by this driver
THOR_INVALID_BOARD_NO       // Supplied board number is not valid
```

See Also

`drvWriteConfigData()`

11.1.18 drvReadDriverData()

Synopsis

Read the driver communications parameters, which are either read from the registry (Windows 95 and Windows NT drivers) or set earlier by the application and stored internally in the driver (DOS driver).

Note: The calling application must allocate the *ThorDriverT* data structure and pass a pointer to the *drvReadDriverData()* function. The drvReadDriverData() function copies the information into the provided the data structure.

Definition

```
ThorRc drvReadDriverData(
  ThorDriverT *drvData      // Driver Data read (Output from this function)
);
```

Returns

```
THOR_SUCCESS                  // OK
```

11.1.19 drvReadIo()

Synopsis

Reads a byte (8 bits) from an I/O-port in an on-board device.

Definition

```
short drvReadIo(
  short boardNo,   // Board number
  Uint  portId     // Local I/O-bus port address to be read
);
```

Returns

```
The 8-bit (Byte) value of the I/O port.
```

See Also

`drvWriteIo()`

11.1.20 drvReadMem()

Synopsis

Reads a Word (16 bits) from an on-board memory (DRAM) location.

Definition

```
Word drvReadMem(
  short boardNo,      // Board number
  Ulong lmbAddr       // On-board Memory Address (flat model)
);
```

Returns

The 16-bit (Word) value of the memory location.

See Also

```
drvReadMem32()
drvWriteMem()
```

11.1.21 drvReadMem32()

Synopsis

Reads a Double-Word (32 bits) from an on-board memory (DRAM) location.

Definition

```
Ulong drvReadMem32(
  short boardNo,      // Board number
  Ulong lmbAddr       // Local Memory Bus Address (flat model)
);
```

Returns

The 32-bit (Double-Word) value of the memory location.

See Also

```
drvReadMem()
drvWriteMem()
drvWriteMem32()
```

11.1.22 drvReadMemBlock()

Synopsis

Copies a block of data from the on-board memory to a buffer in the host memory.

NOTE: This implementation only allows the number of bytes copied (count) to be less than the memory window size.

Definition

```
ThorRc drvReadMemBlock(
  short boardNo,  // Board number
  Ulong lmbAddr,  // On-board Memory Starting Address (flat model)
  Byte *dest,     // Buffer in the host where the block is going to be copied to
                  // NOTE: This buffer must be allocated by the calling
                  //       application.
  Word count      // Number of bytes to be copied
);
```

Returns

```
THOR_SUCCESS          // OK
THOR_DATA_TOO_LARGE   // The buffer is larger than the memory window
                      // size
```

See Also

```
drvWriteMemBlock()
```

11.1.23 drvRegisterCallback()

Synopsis

Register a callback function (implemented by the application) which will be called by the driver upon reception of a message or a hardware/line status change. The use of a callback function allows implementation of event driven applications.

NOTE: Only available with Windows 95 and Windows NT drivers. DOS applications must poll the driver.

Definition

```
ThorRc drvRegisterCallback(
  void (*lapiCallBack)(void)    // function pointer to the callback function in
                                // the application to be called from the driver
);
```

Returns

```
THOR_SUCCESS          // OK
THOR_INVALID_CALLBACK_FUNCTION        // Invalid function pointer
THOR_CALLBACK_ALREADY_SET             // Callback function can be set only once
THOR_UNABLE_TO_CREATE_CALLBACK_THREAD // Creation of a new thread failed
```

See Also

```
IOCTL_START_EVENT_NOTIFICATIONS
thorRegisterCallback()
```

11.1.24 drvResetDevices()

Synopsis

Resets (provides a pulse on the reset pins) the following Thor-2 devices:
- Line Interfaces (LI0 and LI1)
- Time-Space Switch (TSS)
- HDLC Controller (HDLC)
- Codecs (CD0 and CD1)

Definition

```
ThorRc drvResetDevices(
  short boardNo        // Number of the board whose devices to reset
);
```

Returns

```
THOR_SUCCESS                  // OK
THOR_INVALID_BOARD_NO         // Supplied board number is not valid
```

See Also

```
drvResetDriver()
thorResetDriver()
```

11.1.25 drvResetDriver()

Synopsis

Resets the Driver software. Clears and re-initializes the internal data structures.

Definition

```
ThorRc drvResetDriver(
  void
```

```
);
```

Returns

```
THOR_SUCCESS              // OK
```

See Also

```
thorResetDriver()
drvResetDevices()
```

11.1.26 drvSetClkSrc()

Synopsis

Sets the clock source for the Thor-2 board. All the internal data highways are synchronized and run from the same master clock. The possible clock sources are defined by *ThorClkSrcType.*

Definition

```
ThorRc drvSetClkSrc(
  short boardNo,                  // Board number.
  short clkSrc                    // Clock source to be used  (ThorClkSrcT)
);
```

Returns

```
THOR_SUCCESS                  // OK
THOR_TSS_INVALID_TIMING_MODE  // The provided Clock Source is not valid
                              // See ThorClkSrcType for valid values
THOR_INVALID_BOARD_NO         // Supplied board number is not valid
```

11.1.27 drvSetup()

Synopsis

Provides the driver the communications parameters to be used with the Thor-2 board(s). This function only needs to be called in DOS. In Windows 95 and Windows NT the information is available to the driver from the Windows Registry.

Note: The DOS driver can work in polling mode, i.e. without an IRQ. In polling mode the maximum number of pipes that can be configured is 6. To run in polling mode pass aIrq=-1 to this function.

Definition

```
ThorRc drvSetup(
```

```
  short aIoBaseAddr,          // I/0-base address
  short aNoOfBoards,          // Number of Boards installed (up to 4).
  short aIrq,                 // IRQ for all the installed THOR boards (-1 for
                              // polling mode)
  Word infoBufferSize         // Size of driver fifo
);
```

Returns

```
THOR_SUCCESS                  // OK
THOR_INVALID_BOARD_NO         // The supplied board number is not valid
THOR_INVALID_IRQ_NO           // The supplied IRQ number is not valid
THOR_INVALID_ADDR             // The supplied I/O-Base address is not valid
```

Platforms

DOS

See Also

```
thorConstructDriver()
drvInit()
```

11.1.28 drvSetupIoWin()

Synopsis

Initializes the sliding I/O window by setting the Host I/O Offset (HIO register) and the host I/O Window Size (IWS register).

With DOS driver this function must be called before any other I/O window functions are called. With Windows 95 and Windows NT drivers, the use of this function is not necessary as the information is available to the driver from the Windows Registry. However, with Windows drivers this function can be used to overwrite the information stored in the registry.

Definition

```
ThorRc drvSetupIoWin(
  short boardNo,               // Board number.
  short aHostIoOffset,         // Absolute starting address for the I/O window
                               // in the host I/O-address space (e.g. 0x290).
  short aHostIoWindowSize      // Size (no of bytes) of the I/O-window in the
                              // host I/O address space (4 <= size <= 1kByte).
                               // Typical value 16
);
```

Returns

```
THOR_SUCCESS                  // OK
THOR_INVALID_HOST_IO_OFFSET   // The provided host I/O offset value is invalid
THOR_INVALID_IO_WINDOW_SIZE   // The provided I/O window size is invalid
THOR_INVALID_BOARD_NO         // Supplied board number is not valid
```

Platforms

DOS

See Also

```
drvSetupMemWin()
thorConstructDriver()
```

11.1.29 drvSetupMemWin()

Synopsis

Initializes the memory window by setting the host memory offset (HMO register) and the host memory window size (MWS register).

With DOS driver this function must be called before any memory window functions are called. In Windows 95 and Windows NT use of this function is not necessary as the information is available to the driver from the Windows Registry. However, with Windows drivers this function can be used to overwrite the information stored in the registry.

Definition

```
ThorRc drvSetupMemWin(
  short boardNo,                // Board number.
  Ulong aHostMemoryOffset,      // Absolute starting addresses of the memory
                                // in the host memory address space
                                // (e.g. 0x0D0000).
  Ulong aHostMemoryWindowSize   // Size (no of bytes) of the memory window
                                // in the host memory address space
                                // (256 <= size <= 16MByte). Typical values:
                                // 16kBytes, 32kBytes, or 64kBytes.
);
```

Returns

```
THOR_SUCCESS                  // OK
THOR_INVALID_HOST_MEM_OFFSET  // The provided host memory offset
                              // value is invalid
```

```
THOR_INVALID_MEM_WINDOW_SIZE   // The provided memory window size is invalid
THOR_INVALID_BOARD_NO         // Supplied board number is not valid
```

Platforms

DOS

See Also

```
drvSetupIoWin()
thorConstructDriver()
```

11.1.30 drvStatus2Str()

Synopsis

Converts a status code to a string. Returns a pointer to a string describing the status code in a general fashion. Can be used for “quick and dirty” solutions when the status code is not analyzed properly by the application, but at least something needs to displayed to the user.

Definition

```
char *drvStatus2Str(
  short boardNo,                  // Number of the Thor-2 Board
  short liNo,                     // Number of the LI reporting the status
  ThorStatusType statusCode       // The status code
);
```

Returns

Pointer to a static string owned by the function.

See Also

```
drvThorRc2Str()
thorGetErrMsg()
```

11.1.31 drvThorRc2Str()

Synopsis

Converts a *ThorRc* return code to a string. Returns a pointer to a string describing the error code in a general fashion. Can be used for “quick and dirty” solutions when the return code is not analyzed properly by the application, but at least something needs to be displayed to the user.

Definition

```
char *drvThorRc2Str(
  ThorRc errCode         // ThorRc return code to be converted.
);
```

Returns

Pointer to a static string owned by the function.

See Also

`thorGetErrMsg()`
`drvStatus2Str()`

11.1.32 drvUnInstallIsr()

Synopsis

Uninstalls the Thor-2 interrupt service routine.

Definition

```
ThorRc drvUnInstallIsr(
  void
);
```

Returns

`THOR_SUCCESS`

Platforms

DOS

See Also

`drvInstallIsr()`
`thorDestructDriver()`

11.1.33 drvWriteConfigData()

Synopsis

Stores the Thor-2 T1/E1 configuration data persistently into the on-board flash

memory.

Definition

```
ThorRc drvWriteConfigData(
  short boardNo,         // Board number
  ThorConfigT *cfgData   // Configuration data to be written
);
```

Returns

```
THOR_SUCCESS           // OK
THOR_FLASH_ERASE_ERR   // Failed to erase the Flash Sector
THOR_FLASH_WRITE_ERR   // Write to the flash failed
THOR_FLASH_CONFIG_ERR  // Configuration did not complete successfully
THOR_INVALID_BOARD_NO   // Supplied board number is not valid
```

See Also

```
drvReadConfigData()
```

11.1.34 drvWriteIo()

Synopsis

Writes a byte (8 bits) to an I/O-port in an on-board device.

Definition

```
ThorRc drvWriteIo(
  short boardNo,     // Board number
  Uint portId,       // On-board I/O-port address
  Uint value         // Byte (8 bit) value to be written
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
drvReadIo()
```

11.1.35 drvWriteMem()

Synopsis

Writes a Word (16 bits) to the on-board memory (DRAM) location.

Definition

```
ThorRc drvWriteMem(
  short boardNo,      // Board number
  Ulong lmbAddr,      // On-board Memory Address (flat model)
  Word value          // Word (2 bytes) value to be written
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
drvWriteMem8()
drvWriteMem32()
drvReadMem()
```

11.1.36 drvWriteMem8()

Synopsis

Writes a Byte (8 bits) to the on-board memory (DRAM) location.

Definition

```
ThorRc drvWriteMem8(
  short boardNo,      // Board number
  Ulong lmbAddr,      // On-board Memory Address (flat model)
  Word value          // Byte (8-bit) value to be written
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
drvWriteMem()
drvWriteMem32()
```

11.1.37 drvWriteMem32()

Synopsis

Writes a Double-Word (32 bits) to the on-board memory (DRAM) location.

Definition

```
ThorRc drvWriteMem32(
  short boardNo,    // Board number
  Ulong lmbAddr,    // Local Memory Bus Address (flat model)
  Ulong value       // Dword (4 bytes) of value will be written
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
drvWriteMem()
drvReadMem32()
```

11.1.38 drvWriteMemBlock()

Synopsis

Copies a block of data from a buffer in the host memory to the on-board memory.

Definition

```
ThorRc drvWriteMemBlock(
  short boardNo,      // Board number
  Ulong lmbAddr,      // On-Board Memory Starting Address (flat model)
  Byte *src,          // Buffer in the host from where the block
                      // is going to be copied from
  Uint  count         // Number of bytes to be copied
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
drvReadMemBlock()
drvCmpMemBlock()
```

11.2 Line Interface Functions - *li.h*

11.2.1 liAlarmOff()

Synopsis

Stops sending a previously initiated alarm towards the remote end.

Definition

```
ThorRc liAlarmOff(
  short boardNo,          // Number of the Thor-2 board hosting the LI
  short liNo,             // Number of the Line Interface to clear the
                          // alarm from.
  LiAlarmType alarmType   // Alarm type to clear
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_INVALID_ALARM_TYPE // The provided alarm type is not a valid type
THOR_WRONG_CONTEXT      // The provided alarm type is not available in the
                        // current Li mode
THOR_INVALID_BOARD_NO   // Function was supplied an invalid board number
THOR_INVALID_LI_NO      // Function was supplied an invalid LI number
```

See Also

```
liAlarmOm()
thorAlarmOff()
```

11.2.2 liAlarmOn()

Synopsis

Initiates the sending of an alarm towards the remote end. The sending of an alarm will continue until turned off with *liAlarmOff()*.

Definition

```
ThorRc liAlarmOn(
  short boardNo,        // Number of the Thor-2 board hosting the LI
  short liNo,           // Number of the Line Interface to send the alarm from
  LiAlarmType alarmType // Alarm type to send
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_INVALID_ALARM_TYPE // The provided alarm type is not a valid type
THOR_WRONG_CONTEXT      // The provided alarm type is not available in the
                        // current Li mode
THOR_INVALID_BOARD_NO   // Function was supplied an invalid board number
THOR_INVALID_LI_NO      // Function was supplied an invalid LI number
```

See Also

```
liAlarmOff()
thorAlarmOn()
```

11.2.3 liBitRobAccessDisable()

Synopsis

Disables the sending and receiving of bit-robbed signalling data.

NOTE: Only meaningful in T1 mode.

Definition

```
ThorRc liBitRobAccessDisable(
  short        boardNo,          // Number of the Thor-2 board hosting the LI
  short        liNo              // Number of the Line Interface
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_WRONG_CONTEXT      // Bit Rob Data is not available in E1 Mode
THOR_INVALID_BOARD_NO   // Function was supplied an invalid board number
THOR_INVALID_LI_NO      // Function was supplied an invalid LI number
```

See Also

```
liBitRobAccessEnable()
liSetBitRobData()
```

11.2.4 liBitRobAccessEnable()

Synopsis

Enables the user to send and receive bit-robbed signalling data. Bit Robbing can be used in F12, ESF, and F72 frame formats (T1 only). In F12 and F72 there are two signaling channels called A and B. In ESF format there are four signaling channels: A,

B, C, and D. The received signalling data is passed to the user via the *drvRead()* function. See the *ThorFrameType THOR_FM_BRS*. To transmit bit-robbed signaling data, use the function *liSetBitRobData()*.

If a certain time-slot is used for data traffic, it cannot be overwritten with bit-robbing data, and those time slots should be defined as "Clear Channels" (see the Clear-Channel parameter in the T2config configuration file). If a time-slot (channel) is defined as a Clear Channel it will not be overwritten by bit robbing or Zero Code Suppression (ZCS, B7 stuffing).

NOTE: Only meaningful in T1 mode, and in F12, ESF, and F72 frame formats.

Definition

```
ThorRc liBitRobAccessEnable(
  short        boardNo,          // Number of the Thor-2 board hosting the LI
  short        liNo              // Number of the Line Interface
);
```

Returns

```
THOR_SUCCESS         // OK
THOR_WRONG_CONTEXT   // Bit Rob Data is not available in E1 Mode
THOR_NOT_SETUP       // The LI has not been configured with for a frame
                     // format that supports Bit Robbing
THOR_INVALID_BOARD_NO   // Function was supplied an invalid board number
THOR_INVALID_LI_NO      // Function was supplied an invalid LI number
```

See Also

```
liBitRobAccessDisable()
liSetBitRobData()
```

11.2.5 liConfigure()

Synopsis

Initializes one Line Interface Transceiver for E1 or T1 mode. The configuration parameters to be used are passed to the function with the *liConfigOptions* argument. The configuration options to be used can first be read from the flash with the *drvReadConfigData()* function, or can later be stored to the flash memory with the *drvWriteConfigData()* function.

Definition

```
ThorRc liConfigure(
  short boardNo,            // Number of the Thor-2 board hosting the LI
  short liNo,               // Number of the Li chip to be configured.
```

```
  LiMode liMode,            // Mode to be configured to: THOR_T1 or THOR_E1.
  LiConfigOptionsT *liConfigOptions   // Configuration parameters to be used
);
```

Returns

```
THOR_SUCCESS  // OK
THOR_LI_INVALID_MODE    // The Provide Li mode is not T1 or E1
THOR_LI_INVALID_CLOCK_MODE     // The provided LI clock mode is not a valid mode
THOR_LI_INVALID_RESYNC_OPTION  // The provided LI Auto Resynchronization
                               // configuration option is not a valid option
THOR_LI_INVALID_TRANSMIT_LINE_CODE  // The provided LI Transmit line code is
                                    // not a valid code
THOR_LI_INVALID_RECEIVE_LINE_CODE  // The provided LI Receive line code is not
                                   // a valid code
THOR_LI_INVALID_AIS_DETECTION_OPTION  // The provided LI AIS detection option
                                      // is not a valid option
THOR_LI_INVALID_TRANSMIT_FRAME_FORMAT  // The provided LI Transmit Frame Format
                                       // is not a valid format
THOR_LI_INVALID_RECEIVE_FRAME_FORMAT  // The provided LI Receive Frame Format
                                      // is not a valid format
THOR_LI_INVALID_HDB3_ERROR_OPTION  // The provided LI HDB3 Error Detection
                                   // option is not a valid option
THOR_LI_INVALID_REGAIN_MULTI_FRAME_OPTION  // The provided LI Regain Multi
                                           // Frame option is not a valid option
THOR_LI_INVALID_REMOTE_ALARM_OPTION  // The provided LI Remote Alarm option is
                                     // not a valid option
THOR_LI_INVALID_TRANSMIT_POWER_OPTION  // The provided LI Transmit Power option
                                       // is not a valid option
THOR_LI_INVALID_RECEIVE_EQUALIZER_OPTION  // The provided LI Receive Equalizer
                                          // option is not a valid option
THOR_LI_INVALID_SIGNALING_MODE  // The provided LI Signaling mode is not a
                                // valid mode
THOR_LI_INVALID_FRAME_FORMAT  // The provided LI Frame Format is not a valid
                              // format
THOR_LI_INVALID_TRANSMIT_REMOTE_ALARM_FORMAT  // The provided LI Transmit
                             // Remote Alarm Format is not a valid format
THOR_LI_INVALID_RECEIVE_REMOTE_ALARM_FORMAT  // The provided LI Receive Remote
                                             // Alarm Format is not a valid format
THOR_INVALID_BOARD_NO   // Function was supplied an invalid board number
THOR_INVALID_LI_NO      // Function was supplied an invalid LI number
```

See Also

```
drvReadConfigData()
drvWriteConfigData()
```

```
thorConfigureLi()
```

11.2.6 liExistenceChk()

Synopsis

Attempts to reads certain registers in the line interfaces and verifies that they contain the default values. The registers should contain the default values after reset. This function can be used to test is a board is present of to test that the Line Interface transceiver circuits are functional.

Definition

```
ThorRc liExistenceChk (
  short boardNo,      // Board number
  short liNo          // LI number
);
```

Returns

```
THOR_SUCCESS             // The registers contain the default values
THOR_NON_DEFAULT_LI      // Non default values read. Either there is
                         // no Thor-2 board present or the LIs are
                         // not functioning properly.
THOR_INVALID_BOARD_NO    // Function was supplied an invalid board number
THOR_INVALID_LI_NO       // Function was supplied an invalid LI number
```

See Also

```
thorBoardExistence()
drvBoardExistence()
```

11.2.7 liForceResynch()

Synopsis

Initiates the resynchronization procedure of the pulse frame and the CRC-multiframe starting directly after the old framing candidate.

Definition

```
ThorRc liForceResynch(
  short boardNo,          // Number of the Thor-2 board hosting the LI
  short liNo              // Number of the Line Interface
);
```

Returns

```
THOR_SUCCESS
```

11.2.8 liGetSaBitValue()

Synopsis

Retrieves the value to of the received SaX bits. Returns a byte (8-bits) received during the last CRC-Multiframe in parameter **saVal*.

NOTE: Only meaningful in E1 mode

Definition

```
ThorRc liGetSaBitValue(
  short      boardNo,      // Board number.
  short      liNo,         // Number of the Line Interface
  LiSaBit    saBit,        // Sa bit to use
  Byte      *saVal         // Returns last received 8-bits (output from this
                           //  function)
);
```

Returns

```
THOR_SUCCESS                // OK
THOR_WRONG_CONTEXT          // The operation is not available in T1 mode
THOR_INVALID_BOARD_NO       // Function was supplied an invalid board number
THOR_INVALID_LI_NO          // Function was supplied an invalid LI number
```

See Also

```
liSetSaBitValue()
liSaBitAccessEnable()
liSaBitAccessDisable()
```

11.2.9 liGetSiBitValue()

Synopsis

Retrieves the value of the Si bits received during the last frame. In Doubleframe format, these are the first bits of each frame. In CRC-Multiframe format, the Si bit are the first bits of frames 13 and15. In CRC-Multiframe format these bits are also known as the E-bits of spare bits for international use.

NOTE: Only meaningful in E1 mode

Definition

```
ThorRc liGetSiBitValue(
  short  boardNo,          // Number of the Thor-2 board hosting the LI
  short  liNo,             // Number of the Line Interface
  Byte *si1Val,            // Value of the FAS Si-bit in doubleframe format
                          // or Si (E) bit in frame 13 in CRC-multiframe format
  Byte *si2Val             // Value of the service word Si bit in DoubleFrame
                           // format or Si (E) bit in frame 15 in
                           // CRC-multiframe format
);
```

Returns

```
THOR_SUCCESS               // OK
THOR_WRONG_CONTEXT         // The operation is not available in T1 mode
THOR_INVALID_BOARD_NO      // Function was supplied an invalid board number
THOR_INVALID_LI_NO         // Function was supplied an invalid LI number
```

See Also

`liSetSiBitValue()`

11.2.10 liGetStatus()

Synopsis

Checks and returns the physical line status of the T1/E1 line interface.

Definition

```
ThorRc liGetStatus(
  short boardNo,               // Board number.
  short liNo                   // Number of the Line Interface to be read.
);
```

Returns

```
THOR_L1_OK                 // Physical Layer is up
THOR_L1_DOWN               // Physical Layer is down
```

See Also

`thorGetStatusLi()`

11.2.11 liLoop()

Synopsis

Loops the Line Interface receive and transmit lines; I.e. received E1/T1 data will be transmitted back on the transmit pairs.

Definition

```
ThorRc liLoop(
  short  boardNo,          // Number of the Thor-2 board hosting the LI
  short  liNo,             // Number of the Line Interface
  LiLoopT loopType         // Loop type: Line loop or Remote loop
);
```

Returns

```
THOR_SUCCESS               // OK
THOR_LI_INVALID_LOOP_TYPE  // The provided loop type is not valid
THOR_INVALID_BOARD_NO      // Function was supplied an invalid board number
THOR_INVALID_LI_NO         // Function was supplied an invalid LI number
```

See Also

```
thorLoopLi()
```

11.2.12 liSaBitAccessDisable()

Synopsis

Disables the sending of the Sa-bit values specified with the *liSetSaBitValue*() function.

NOTE: Only meaningful in E1 mode

Definition

```
ThorRc liSaBitAccessDisable(
  short        boardNo,          // Number of the Thor-2 board hosting the LI
  short        liNo              // Number of the Line Interface
);
```

Returns

```
THOR_SUCCESS                // OK
THOR_WRONG_CONTEXT          // The operation is not available in T1 mode
```

See Also

```
liSaBitAccessEnable()
liSetSaBitValue()
liGetSaBitValue()
```

11.2.13 liSaBitAccessEnable()

Synopsis

Enables the sending the Sa-bit values specified with the *liSetSaBitValue()* function.

NOTE: Only meaningful in E1 mode

Definition

```
ThorRc liSaBitAccessEnable(
  short        boardNo,          // Number of the Thor-2 board hosting the LI
  short        liNo              // Number of the Line Interface
);
```

Returns

```
THOR_SUCCESS                // OK
THOR_WRONG_CONTEXT          // The operation is not available in T1 mode
THOR_INVALID_BOARD_NO       // Function was supplied an invalid board number
THOR_INVALID_LI_NO          // Function was supplied an invalid LI number
```

See Also

```
liSetSaBitValue()
liGetSaBitValue()
liSaBitAccessDisable()
```

11.2.14 liSetBitRobData()

Synopsis

If Bit robbing has been enabled with a successful call to *liBitRobAccessEnable()*, then this function will transmit the bit robbed signaling data passed to this function. The same data will be sent continuously until this function has been called again to change the data. However, when called repeatedly, every data will be sent in at least one frame. If the function is called before the transmitting of the previous data has been transmitted at least once, the function will return THOR_TX_BUSY.

NOTE: Only meaningful in T1 mode.

Definition

```
ThorRc liSetBitRobData(
  short        boardNo,          // Number of the Thor-2 board hosting the LI
  short        liNo              // Number of the Line Interface
  LiBrData    *brData            // Bit-robbed signalling data to be transmitted
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_WRONG_CONTEXT      // Bit Rob Signalling is only available in T1 mode
THOR_NOT_SETUP          // The LI has not been configured for a frame
                        // format that supports Bit Robbing
THOR_TX_BUSY            // The previously specified data has not yet been
                        // transmitted. Try again later
THOR_INVALID_BOARD_NO   // Function was supplied an invalid board number
THOR_INVALID_LI_NO      // Function was supplied an invalid LI number
```

See Also

```
liBitRobAccessEnable()
liBitRobAccessDisable()
```

11.2.15 liSetClkMode()

Synopsis

Sets a Line Interface Transceiver as a Clock Master or a Clock Slave.

Definition

```
ThorRc liSetClkMode (
  short boardNo,        // Number of the Thor-2 board hosting the LI
  short liNo,           // Number of the Line Interface
  LiClkMode clkMode     // Clock Mode
);
```

Returns

```
THOR_SUCCESS                // OK
THOR_LI_INVALID_CLOCK_MODE  // The provided clock mode is not a valid mode
```

11.2.16 liSetSaBitValue()

Synopsis

Sets the value to be sent at the SaX bits. Eight bits to be sent can be specified per Sa-bit. In CRC-Multiframe format, one bit is sent in the corresponding Sa-bit location of time-slot 0 in every other frame (in frames that do not contain frame alignment information). The least significant bit of the saVal byte is sent first in the frame number 1 of the multiframe and the most significant bit of the Byte is sent last in the frame number 15 of the multiframe.

In Doubleframe format one bit of the saVal word is sent in the every other frame starting from the least significant bit.

NOTE: Only meaningful in E1 mode

Definition

```
ThorRc liSetSaBitValue(
  short      boardNo,          // Number of the Thor-2 board hosting the LI
  short      liNo,             // Number of the Line Interface
  LiSaBit    saBit,            // Sa bit to use
  Byte       saVal             // word to be sent. LSB will be sent first.
);
```

Returns

```
THOR_SUCCESS                // OK
THOR_WRONG_CONTEXT          // The operation is not available in T1 mode
THOR_INVALID_BOARD_NO   // Function was supplied an invalid board number
THOR_INVALID_LI_NO      // Function was supplied an invalid LI number
```

See Also

```
liGetSaBitValue()
liSaBitAccessEnable()
liSaBitAccessDisable()
```

11.2.17 liSetSiBitValue()

Synopsis

Sets the value to be sent at the Si bit positions. In Doubleframe format, these are the first bits of each frame. In CRC-Multiframe format, the Si bits are the first bits of frames 13 and 15. In CRC-Multiframe format these bits are also known as the E-bits or Spare bits for International use.

NOTE: Only meaningful in E1 mode

Definition

```
ThorRc liSetSiBitValue(
  short  boardNo,         // Number of the Thor-2 board hosting the LI
  short  liNo,            // Number of the Line Interface
  Byte si1Val,            // Value of the FAS Si-bit in doubleframe format
                          // or Si (E) bit in frame 13 in CRC-multiframe format
  Byte si2Val             // Value of the service word Si bit in DoubleFrame
                          // format
                          // or Si (E) bit in frame 15 in CRC-multiframe format
);
```

Returns

```
THOR_SUCCESS                // OK
THOR_WRONG_CONTEXT          // The operation is not available in T1 mode
THOR_INVALID_BOARD_NO   // Function was supplied an invalid board number
THOR_INVALID_LI_NO      // Function was supplied an invalid LI number
```

See Also

```
liGetSiBitValue()
```

11.3 High-Level Data Control Functions - *hdlc.h*

11.3.1 hdlcInitPipe()

Synopsis

Initializes the HDLC Controller. A pipe can contain one time-slot, only certain bits of a time-slot (sub-channel), or several time-slots (super-channel). Each bit that is to be included in the pipe is passed as a bit rate mask (see HdlcPipeOpts). The bit rate mask is an array of 32 bytes, where index 0 is time-slot 0, etc. A '1' in a bit position indicates that the corresponding bit in the time-slot if included in the pipe. A pipe needs to be configured before data or HDLC frames can be received or sent. The initialization of the time-slot assignment and the selected channel is performed in both TX and RX directions.

Definition

```
ThorRc hdlcInitPipe(
  short boardNo,      // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,       // Number of the Pipe (channel)
  HdlcPipeOpts *pipeOpts // Configuration options for the pipe
);
```

Returns

```
THOR_SUCCESS               // OK
THOR_TOO_MANY_PIPES        // The user has attempted to initialize more pipes
                           // than supported in this version
THOR_HDLC_AR_BUSY          // HDLC Controller is BUSY
THOR_INVALID_BOARD_NO      // Function was supplied an invalid Board number
THOR_HDLC_INVALID_PIPE_NO  // Function was supplied an invalid Pipe number
```

See Also

```
thorConfigurePipe()
```

11.3.2 hdlcSendAbort()

Synopsis

Aborts the currently transmitted frame (if there is one being transmitted). The frame is aborted by:

```
0x7F for HDLC mode
0x00 for TMB mode
0x0000 for TMR mode
pipeOpts.tflag for TMA (pipeOpts.flagAdjustment==TRUE)
```

```
  0xFF for TMA (pipeOpts.flagAdjustment==FALSE)
```

To resume sending of frames, use the *hdlcSendData()* or *hdlcSendPattern()* functions.

Definition

```
ThorRc hdlcSendAbort(
  short boardNo,    // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo      // Number of the Pipe (channel)
);
```

Returns

```
THOR_SUCCESS                   // OK
THOR_HDLC_AR_BUSY              // HDLC Controller is BUSY
THOR_HDLC_INVALID_STATE_TRANS  // The pipe is not in a valid state
                                   // to perform this action
THOR_INVALID_BOARD_NO          // Function was supplied an invalid Board number
THOR_HDLC_INVALID_PIPE_NO      // Function was supplied an invalid Pipe number
```

See Also

```
thorWritePipe()
hdlcSendData()
hdlcSendPattern()
```

11.3.3 hdlcSendData()

Synopsis

Transmits transparent data or HDLC frames over the specified pipe.

Definition

```
ThorRc hdlcSendData(
  short boardNo,    // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,     // Number of the Pipe (channel)
  Byte    *data,    // Data to be sent
  Word dataLen,     // Length of the data
  Bool endOfData    // TRUE if a Frame End should be sent after the data
);
```

Returns

```
THOR_SUCCESS                // OK, message has been successfully sent
THOR_TX_BUSY                // Transmitter not ready. Transmission of the
                            // previous frame has not been completed.
                            // Try again later.
```

```
THOR_HDLC_MSG_TOO_LONG        // The message (frame) is too long for the
                              // current driver configuration.
THOR_HDLC_INVALID_TX_STATE    // The pipe is not in a valid transmit state
THOR_HDLC_INVALID_STATE_TRANS // Function Internal error
THOR_INVALID_BOARD_NO         // Function was supplied an invalid Board number
THOR_HDLC_INVALID_PIPE_NO     // Function was supplied an invalid Pipe number
```

See Also

```
thorWritePipe()
thorRead()
drvRead()
```

11.3.4 hdlcSendPattern()

Synopsis

Transmits data patterns (which can be HDLC frames or transparent data) continuously over a number of pipes. An array of patterns is passed as a parameter. This function will send each pattern in the array, starting with the first pattern in the array. When the last pattern in the array is sent it will wrap and send the first pattern again. Between each pattern a number of inter-frame time fill characters can be sent (as specified in HdlcDataPatternT). To stop sending the pattern, call either the *hdlcSendAbort()* or the *hdlcSendData()* functions.

Definition

```
ThorRc hdlcSendPattern(
  short boardNo,    // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,     // Number of the Pipe (channel)
  HdlcDataPatternT patterns[], // Patterns (frames) to be sent
  short noOfPatterns           // Number of elements in the patterns[] array
);
```

Returns

```
THOR_SUCCESS                   // Message accepted and is being sent
THOR_TX_BUSY                   // Previous message not yet sent completely
                               // Try again later
THOR_HDLC_MSG_TOO_LONG         // Message is too long to be sent
THOR_HDLC_INVALID_STATE_TRANS  // HDLC Invalid State Transition,
                               // Driver Internal error
THOR_HDLC_AR_BUSY              // HDLC Controller is BUSY,
                               // Driver internal error
THOR_INVALID_BOARD_NO          // Function was supplied an invalid Board number
THOR_HDLC_INVALID_PIPE_NO      // Function was supplied an invalid Pipe number
```

See Also

```
thorWritePipe()
thorRead()
drvRead()
hdlcSendData()
hdlcSendAbort()
```

11.3.5 hdlcReceiveOff()

Synopsis

Sets the receiver in the off condition for a configured pipe. When the receiver is turned off the HDLC controller can still receive frames, but they are discarded and not stored in the receive fifo.

Definition

```
ThorRc hdlcReceiveOn(
  short boardNo,    // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,     // Number of the Pipe (channel)
);
```

Returns

```
THOR_SUCCESS                  // OK
THOR_HDLC_INVALID_RX_STATE    // The Pipe is not in a valid receive state
THOR_HDLC_INVALID_STATE_TRANS  // HDLC Invalid State Trsnsition,
                               // Driver Internal error
THOR_HDLC_AR_BUSY          // HDLC Controller is BUSY, Driver internal error
THOR_INVALID_BOARD_NO       // Function was supplied an invalid Board number
THOR_HDLC_INVALID_PIPE_NO    // Function was supplied an invalid Pipe number
```

See Also

```
hdlcReceiveOn()
```

11.3.6 hdlcReceiveOn()

Synopsis

Turns on the receiver for a configured pipe. The received frames will be stored in the receive fifo.

Definition

```
ThorRc hdlcReceiveOn(
```

```
  short boardNo,    // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,     // Number of the Pipe (channel)
);
```

Returns

```
THOR_SUCCESS                  // OK
THOR_HDLC_INVALID_RX_STATE    // The Pipe is not in a valid receive state
THOR_HDLC_INVALID_STATE_TRANS  // HDLC Invalid State Trsnsition,
                               // Driver Internal error
THOR_HDLC_AR_BUSY          // HDLC Controller is BUSY, Driver internal error
THOR_INVALID_BOARD_NO       // Function was supplied an invalid Board number
THOR_HDLC_INVALID_PIPE_NO    // Function was supplied an invalid Pipe number
```

See Also

```
hdlcReceiveOff()
```

11.3.7 hdlcMemoryAlloc()

Synopsis

Allocate a blob of on-board memory to be used for sending data. Returns a handle (dataId) to the memory area which can be used to access the memory with other hdlcMemoryXXXX() functions. The function also returns the number of memory units (nrBlobs) allocated for the data. The blob size is set with the *maxFrameLen* parameter to the *drvInitHdlc()* function.

Definition

```
ThorRc hdlcMemoryAlloc(
  short boardNo,   // Number of the Thor-2 board hosting the HDLC Controller
  Ulong size,      // Size of memory requested
  short *dataId,   // OUT PARAMETER: Data Identifier for the allocated area
  Word  *nrBlobs   // OUT PARAMETER: Number of data blobs allocated
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_OUT_OF_MEMORY      // Not enough allocatable memory to complete the
                        // request
```

See Also

```
hdlcMemoryFree()
hdlcMemoryWrite()
hdlcMemoryRead()
```

11.3.8 hdlcMemoryFree()

Synopsis

Release on-board memory identified with the specified handle (dataId)

Definition

```
ThorRc hdlcMemoryFree(
  short boardNo,   // Number of the Thor-2 board hosting the HDLC Controller
  short dataId     // Data Identifier whose data blobs are to be freed
);
```

Returns

```
THOR_SUCCES             // OK
THOR_NO_DATA            // No allocated data found for the given Id
```

See Also

```
hdlcMemoryAlloc()
hdlcMemoryWrite()
hdlcMemoryRead()
```

11.3.9 hdlcMemoryWrite()

Synopsis

Writes data into the on-board HDLC memory. Use the dataId and blobNo to identify the memory area to be written. The dataId ties memory blobs together and the HDLC controller will treat data blobs with the same dataId as continuous data. For example, if 24K of raw data is to be sent, one must write the data in to the memory in 3 passes with blob numbers 0, 1, and 2.

Definition

```
ThorRc hdlcMemoryWrite(
  short  boardNo,   // Number of the Thor-2 board hosting the HDLC Controller
  short  dataId,    // Data Identifier, must be kept track by the application
  Word   blobNo,    // Blob number to be overwritten (0, 1, 2, ...)
  Byte   data[],    // One data blob (max size 8K)
  Word   dataLen    // Length of the data to be loaded
);
```

Returns

```
THOR_SUCCESS          // OK
```

```
THOR_INVALID_BOARD_NO  // Provided Board Number is not valid
THOR_SIZE_TOO_LARGE    // Data size must be <= 8192 Bytes
THOR_WRONG_CONTEXT     // The HDLC controller must be initialized before
                       // this function can be called so that the HDLC
                       // memory requirements are known
THOR_OUT_OF_MEMORY     // No free memory to load the data to
```

See Also

```
hdlcMemoryAlloc()
hdlcMemoryFree()
hdlcMemoryRead()
```

11.3.10 hdlcMemoryRead()

Synopsis

Reads data from the on-board HDLC memory corresponding the dataId and the blobNo. Copies the data into user allocated buffer.

Definition

```
ThorRc hdlcMemoryRead(
  short  boardNo,   // Number of the Thor-2 board hosting the HDLC Controller
  short  dataId,    // Data Identifier, must be kept track by the application
  Word   blobNo,    // Data Blob number to be read (0, 1, 2, ...)
  Byte   data[],    // User allocated buffer for one data blob (max size 8K)
  Word   bufLen,    // Length of the user buffer (max size 8K)
  Word   *dataLen   // OUT PARAMETER: Length of the data returned
);
```

Returns

```
THOR_SUCCESS           // OK
THOR_INVALID_BOARD_NO  // Provided Board Number is not valid
THOR_SIZE_TOO_LARGE    // Provided user buffer is not large enough to
                       // hold all the data
THOR_HDLC_NO_DATA      // No data found
```

See Also

```
hdlcMemoryWrite()
hdlcMemoryAlloc()
hdlcMemoruFree()
```

11.3.11 hdlcMemoryCheckId()

Synopsis

Checks whether a data ID is free or in use

Definition

```
ThorRc hdlcMemoryCheckId(
  short  boardNo,  // Number of the Thor-2 board hosting the HDLC Controller
  short  dataId,   // Data ID to be checked
  Bool   *inUse,   // OUT PARAMETER: OTS_TRUE if in use, OTS_FALSE if free
  Word   *size,    // OUT PARAMETER: Data Size
  Word   *nrBlobs  // OUT PARAMETER: Number of data blobs
);
```

Returns

```
THOR_SUCCESS           // OK
THOR_INVALID_BOARD_NO  // Provided Board Number is not valid
```

See Also

```
hdlcMemoryCheckUsage()
```

11.3.12 hdlcMemoryCheckUsage()

Synopsis

Provides a status of how much allocatable HDLC memory is available and is currently in use.

Definition

```
ThorRc hdlcMemoryCheckUsage(
  short boardNo,       // Number of the Thor-2 board hosting the HDLC Controller
  Ulong *totalMem,     // OUT PARAMETER: Total allocatable HDLC memory
  Ulong *memInUse,     // OUT PARAMETER: HDLC memory currently in use
  Ulong *memAvailable  // OUT PARAMETER: HDLC memory available
);
```

Returns

```
THOR_SUCCESS           // OK
THOR_INVALID_BOARD_NO  // Provided Board Number is not valid
```

See Also

`hdlcMemoryCheckId()`

11.3.13 hdlcMemoryStartIdlePattern()

Synopsis

Begin sending an idle pattern on the specified pipe. The user must first load two data blobs into the memory using hdlcMemoryWrite. One of the data blobs are used for the primary idle pattern and the second one for secondary idle pattern. Upon completion of this function, the HDLC controller will constantly send the primary idle pattern. The user can now send data between the idle patterns using the *hdlcMemorySend*() function. After the data has been send, the HDLC controller will start sending the secondary idle pattern.

Note: Idle pattern can contain only one descriptor and the max data size for the idle pattern is 8K.

Note: The datasize is set with the *maxFrameLen* parameter for *drvInitHdlc()* function.

Definition

```
ThorRc hdlcMemoryStartIdlePattern(
  short boardNo,  // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,    // Number of the Pipe (channel)
  short primaryIdleDataIdx,         // Data index for the primary idle pattern
  short secondaryIdleDataIdx,       // Data index for the secondary idle pattern
  Bool  dataEndFlag,        // Set to OTS_TRUE if the pattern (frame) should
                            // end with a frame end (0x7E for HDLC)
  short nrInterFrameTimeFills       // No of interframe time-fill characters
                                    // (0x7E) after this 2 pattern (frame)
                                    // (1=shared flags, 2=non-shared flags)
                                    // Range: 1 <= x <= 4096
);
```

Returns

```
THOR_SUCCESS                    // OK
THOR_HDLC_MSG_TOO_LONG          // Message is too long to be sent
THOR_HDLC_INVALID_STATE_TRANS   // HDLC Invalid State Trsnsition, Driver
                                // Internal error
THOR_HDLC_AR_BUSY           // HDLC Controller is BUSY, Driver internal error
THOR_INVALID_BOARD_NO       // Function was supplied an invalid Board number
THOR_HDLC_INVALID_PIPE_NO   // Function was supplied an invalid Pipe number
THOR_HDLC_NO_DATA           // No data found from the specified data index
```

See Also

`hdlcMemorySendData()`

11.3.14 hdlcMemorySendData()

Synopsis

Send the data identified with the Id. When called, the HDLC controller will first complete the sending of the current Idle pattern, it will then send the data identified with the Id, after which it will immediately continue sending the idle patterns.

Definition

```
ThorRc hdlcMemorySendData(
  short boardNo,   // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,    // Number of the Pipe (channel)
  short dataId     // Data Identifier whose data blobs is to be sent
);
```

Returns

```
THOR_SUCCESS               // OK
THOR_WRONG_CONTEXT         // Idle sending must be started before this
                           // function can be called
THOR_INVALID_BOARD_NO      // Provided Board Number is not valid
THOR_HDLC_INVALID_PIPE_NO  // Function was supplied an invalid Pipe number
THOR_TX_BUSY               // The sending of the previous data has not been
                           // completed, try again later
THOR_OUT_OF_MEMORY         // Not enough transmit descriptors to send the
                           // data. Allocate more descriptor or increase
                           // the maximum data size.
```

See Also

`hdlcMemorySendDataList()`
`hdlcMemoryGetSendStatus()`
`hdlcMemoryStartIdlePattern()`

11.3.15 hdlcMemorySendDataList()

Synopsis

Sends a list of data identified with the data Ids. The data to be send must have been loaded to memory earlier. When called, the HDLC controller will first complete the sending of the current Idle pattern, it will then send the data identified with the list of IDs, after which it will immediately continue sending the idle patterns.

Definition

```
ThorRc hdlcMemorySendDataList(
  short boardNo,   // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,    // Number of the Pipe (channel)
  short dataId[],  // Array of Data Identifiers whose data blobs are to be sent
  short nrDataIds  // Number of Data Identifiers in the list
);
```

Returns

```
THOR_SUCCESS               // OK
THOR_WRONG_CONTEXT         // Idle sending must be started before this
                           // function can be called
THOR_INVALID_BOARD_NO      // Provided Board Number is not valid
THOR_HDLC_INVALID_PIPE_NO  // Function was supplied an invalid Pipe number
THOR_TX_BUSY               // The sending of the previous data has not been
                           // completed, try again later
THOR_DATA_TOO_LARGE        // Too many elements in the dataId array
```

See Also

```
hdlcMemorySendData()
hdlcMemoryGetSendStatus()
hdlcMemoryStartIdlePattern()
```

11.3.16 hdlcMemoryGetSendStatus()

Synopsis

Send the data identified with the Id. When called, the HDLC controller will first complete the sending of the current Idle pattern, it will then send the data identified with the Id, after which it will immediately continue sending the idle patterns.

Definition

```
ThorRc hdlcMemoryGetSendStatus(
  short boardNo,   // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo     // Number of the Pipe (channel)
);
```

Returns

```
THOR_TX_IDLE               // HDLC controller is sending IDLE pattern
                          // on this pipe. User Data can be sent
THOR_TX_BUSY               // HDLC controller is currently busy sending
                           // user data on this pipe. Try again later
THOR_INVALID_BOARD_NO      // Provided Board Number is not valid
```

```
THOR_HDLC_INVALID_PIPE_NO  // Function was supplied an invalid Pipe number
```

See Also

```
hdlcMemorySendData()
hdlcMemorySendDataList()
```

11.3.17 hdlcSS7SetFisu()

Synopsis

Sets the Signalling System #7 (SS#7) Fill-In Signalling Unit (FISU) to be sent on the specified pipe. When this function is called for the first time after *hdlcInitPipe()*, begins sending the specified FISU. If a FISU sending is already on, switches into sending the newly specified FISU.

Definition

```
ThorRc hdlcSS7SetFisu(
  short boardNo,      // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,       // Number of Pipe used for SS#7
  HdlcSS7FisuT *fisu  // FISU to be sent on the pipe
);
```

Returns

```
THOR_SUCCESS               // OK
THOR_INVALID_BOARD_NO      // Provided Board Number is not valid
THOR_HDLC_INVALID_PIPE_NO  // Function was supplied an invalid Pipe number
THOR_WRONG_CONTEXT         // The HDLC controller must be initialized
                           // before this function can be called
```

See Also

```
hdlcSS7GetSendStatus()
hdlcSS7GetReceiveStatus()
```

11.3.18 hdlcSS7GetSendStatus()

Synopsis

Return the Signalling System #7 (SS#7) Fill-In Signalling Unit (FISU) being sent on the specified pipe

Definition

```
ThorRc hdlcSS7GetSendStatus(
```

```
  short boardNo,      // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,       // Number of the Pipe used for SS#7
  HdlcSS7FisuT *fisu  // OUT PARAMETER: FISU currently being sent on the pipe
);
```

Returns

```
THOR_SUCCESS               // OK
THOR_INVALID_BOARD_NO      // Provided Board Number is not valid
THOR_HDLC_INVALID_PIPE_NO  // Function was supplied an invalid Pipe number
THOR_WRONG_CONTEXT         // The HDLC controller must be initialized and
                           // a Fisu must have been set with hdlcSetFisu()
                           // before this function can be called
```

See Also

```
hdlcSS7GetReceiveStatus()
hdlcSS7SetFisu()
```

11.3.19 hdlcSS7GetReceiveStatus()

Synopsis

Retrieve status and statistics of the incoming SS7 link on the specified pipe.

Definition

```
ThorRc hdlcSS7GetReceiveStatus(
  short boardNo,      // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,       // Number of the Pipe used for SS#7
  Ulong *fisuCount,   // OUT PARAMETER: Number of FISUs received
  Ulong *lssuCount,   // OUT PARAMETER: Number of LSSUs received
  Ulong *msuCount,    // OUT PARAMETER: Number of MSUs received
  HdlcSS7FisuT *fisu  // OUT PARAMETER: The last received FISU on the pipe
);
```

Returns

```
THOR_SUCCESS               // OK
THOR_INVALID_BOARD_NO      // Provided Board Number is not valid
THOR_HDLC_INVALID_PIPE_NO  // Function was supplied an invalid Pipe number
THOR_WRONG_CONTEXT         // The HDLC controller must be initialized and
                           // a Fisu must have been set with hdlcSetFisu()
                           // before this function can be called
```

See Also

```
hdlcSS7GetSendStatus()
hdlcSS7SetFisu()
```

11.3.20 hdlcSS7SetFilter()

Synopsis

Set the filter mask for the filtering out SS#7 FISUs, LSSUs, and/or MSUs.

Definition

```
ThorRc hdlcSS7SetFilter(
  short boardNo,     // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,      // Number of the Pipe used for SS#7
  unsigned long filterMask   // Filter Mask (see HDLC_SS7_FILTER_XXXX Macros)
);
```

Returns

```
THOR_SUCCESS               // OK
THOR_INVALID_BOARD_NO      // Provided Board Number is not valid
THOR_HDLC_INVALID_PIPE_NO  // Function was supplied an invalid Pipe number
```

See Also

```
hdlcSS7GetFilter()
```

11.3.21 hdlcSS7GetFilter()

Synopsis

Retrieve the currently active filter mask for filtering out SS#7 FISUs, LSSUs, and/or MSUs

Definition

```
ThorRc hdlcSS7GetFilter(
  short boardNo,    // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,     // Number of the Pipe used for SS#7
  unsigned long *filterMask  // OUT PARAMETER: Currently active Filter Mask
                             // (see HDLC_SS7_FILTER_XXXX Macros)
);
```

Returns

```
THOR_SUCCESS               // OK
```

```
THOR_INVALID_BOARD_NO      // Provided Board Number is not valid
THOR_HDLC_INVALID_PIPE_NO  // Function was supplied an invalid Pipe number
```

See Also

```
hdlcSS7SetFilter()
```

11.3.22 hdlcSS7SendData()

Synopsis

Send a message (LSSU, MSU, or arbitrary data) on a SS#7 pipe. After the sending of the message has been completed, continues sending the new FISU provided as a parameter to the function.

Definition

```
ThorRc hdlcSS7SendData(
  short boardNo,      // Number of the Thor-2 board hosting the HDLC Controller
  short pipeNo,       // Number of the Pipe used for SS#7
  Byte *data,         // Data to be sent
  Word dataLen,       // Length of the data
  HdlcSS7FisuT *nextFisu   // Next FISU to be sent after the data
);
```

Returns

```
THOR_SUCCESS               // OK
THOR_TX_BUSY               // Sending of the previous message has not been
                           // completed, try again later
THOR_INVALID_BOARD_NO      // Provided Board Number is not valid
THOR_HDLC_INVALID_PIPE_NO  // Function was supplied an invalid Pipe number
```

See Also

```
hdlcSS7SetFisu()
```

11.4 Time-Space Switch Functions - *tss.h*

11.4.1 tssClear()

Synopsis

Clears all the Cross-connects from the time-space switch. I.e., every sample of the time-slot will contain the same constant value.

Definition

```
ThorRc tssClear(
  short boardNo
);
```

Returns

```
THOR_SUCCESS           // OK
THOR_INVALID_BOARD_NO  // The function is supplied an invalid board number
```

See Also

```
tssXConnect()
thorConnectChannel()
```

11.4.2 tssConstByte()

Synopsis

Generates a constant byte value on an output time-slot. I.e., every time-slot sample (8000 per second) will have the same value. To turn off the constant byte generation, specify byte value 0, or use the *tssClear()* function.

Definition

```
ThorRc tssConstByte(
  short boardNo,     // The number of the Thor-2 board hosting the TSS
  short pcmHwOut,    // Highway to output the constant byte
  short channelOut,  // Time-slot to output the constant byte
  Byte constVal      // Constant value to output
);
```

Returns

```
THOR_SUCCESS             // OK
THOR_TSS_INVALID_PCM_HW  // The function is supplied an invalid PCH
                         // highway number
```

```
THOR_TSS_INVALID_CHANNEL // The function is supplied an invalid channel number
THOR_INVALID_BOARD_NO    // The function is supplied an invalid board number
```

See Also

```
thorByteOnCh()
thorByteOffCh()
```

11.4.3 tssDisable()

Synopsis

Disables the FMIC after it has been initialized and enabled. The TSS must be disabled when the HDLC controller is being initialized.

Definition

```
ThorRc tssDisable(
  short boardNo
);
```

Returns

```
THOR_SUCCESS           // OK
THOR_INVALID_BOARD_NO  // The function is supplied an invalid board number
```

See Also

```
tssInit()
tssEnable()
```

11.4.4 tssEnable()

Synopsis

Enables the Time-Space Switch after it has been initialized and configured.

Definition

```
ThorRc tssEnable(
  short boardNo        // The number of the Thor-2 board hosting the TSS
);
```

Returns

```
THOR_SUCCESS           // OK
THOR_INVALID_BOARD_NO  // The function is supplied an invalid board number
```

See Also

```
tssInit()
tssDisable()
```

11.4.5 tssInit()

Synopsis

Initializes and configures the Time-Space Switch. When using the low-level libraries, this function must be called before the time-space switch can be used.

Definition

```
ThorRc tssInit(
  short boardNo        // The number of the Thor-2 board hosting the TSS
);
```

Returns

```
THOR_SUCCESS           // OK
THOR_INVALID_BOARD_NO  // The function is supplied an invalid board number
```

See Also

```
tssEnable()
```

11.4.6 tssReadDataMemory()

Synopsis

Reads a snapshot (one Byte) of a particular channel in the Time-Space Switch Data Memory. The Time-Space Switch buffers the data from the time-slots to be switched in the data memory.

Note: The data-memory access is slow this function will return one byte at a arbitrary time from the incoming stream.

Definition

```
ThorRc tssReadDataMemory(
  short boardNo,     // The number of the Thor-2 board hosting the TSS
  short pcmHwIn,     // Incoming highway
  short channelIn,   // Time-slot on the highway
  Byte *dataVal      // Value (snapshot) in the time-slot
);
```

Returns

```
THOR_SUCCESS             // OK
THOR_TSS_INVALID_PCM_HW  // The function is supplied an invalid PCH
                         // highway number
THOR_TSS_INVALID_CHANNEL // The function is supplied an invalid channel number
THOR_INVALID_BOARD_NO    // The function is supplied an invalid board number
```

11.4.7 tssTimingMode()

Synopsis

Sets the MVIP timing mode for the Time-Space Switch. For more information on the timing modes, please refer to the documentation the MVIP-90 Standard. The timing mode can be changed dynamically.

Definition

```
ThorRc tssTimingMode(
  short boardNo,       // The number of the Thor-2 board hosting the TSS
  short mode           // Timing Mode
);
```

Returns

```
THOR_SUCCESS                 // OK
THOR_TSS_INVALID_TIMING_MODE // The function was supplied an invalid
                             // timing mode
THOR_INVALID_BOARD_NO        // The function was supplied an invalid
                             // board number
```

11.4.8 tssXConnect()

Synopsis

Cross-connects a time-slot from one highway to another through the time-space switch.

Note: This function only makes a one-way connection. To make a two way connection, this function must be called twice with the parameters swapped.

Definition

```
ThorRc tssXConnect(
  short boardNo,       // The number of the Thor-2 board hosting the TSS
  short pcmHwIn,       // The Incoming Highway
  short channelIn,     // The time-slot on the incoming highway
```

```
  short pcmHwOut,      // The outgoing highway
  short channelOut     // The time-slot on the outgoing highway
);
```

Returns

```
THOR_SUCCESS             // OK
THOR_TSS_INVALID_PCM_HW  // The function is supplied an invalid PCH
                         // highway number
THOR_TSS_INVALID_CHANNEL // The function is supplied an invalid channel number
THOR_INVALID_BOARD_NO    // The function is supplied an invalid board number
```

See Also

```
tssClear()
thorConnectChannel()
```

11.5 On-board Processor Functions - *lpu.h*

11.5.1 lpuBoot()

Synopsis

Boots up the on-board Processor. The on-board processor is reset and the LPU will begin executing the bootstrap from the flash memory. The LPU can be booted up unconditionally (always) or conditionally only if the LPU is not already running.

Definition

```
ThorRc lpuBoot(
  short boardNo        // Number of the Thor-2 Board Hosting the LPU
  LpuBootCondT condition  // Either BOOT_UNCONDITIONALLY or BOOT_IF_NOT_RUNNING
);
```

Returns

```
THOR_SUCCESS          // OK
THOR_BAD_BOOT_VECTOR  // Corrupted LPU Boot Vector, Problems in Flash Memory
THOR_LPU_BOOT_FAILED  // LPU Boot Failed, the LPU is not running
THOR_NO_MEM_WIN       // Memory Window has not been configured, cannot
                      // access memory
THOR_INVALID_BOARD_NO // The function was supplied an invalid board number
```

11.5.2 lpuFloat()

Synopsis

Disconnects the LPU from the Local Memory Bus by forcing its pins into a high-impedance state. Note: if the LPU is floated, the DRAM memory will not be refreshed and the data in the DRAM will be lost. However, when the LPU is floated, the host can still access the flash memory. This function is used during updating of the flash boot sector.

Definition

```
ThorRc lpuFloat(
  short boardNo        // Number of the Thor-2 Board Hosting the LPU
);
```

Returns

```
THOR_SUCCESS          // OK
THOR_INVALID_BOARD_NO // The function was supplied an invalid board number
```

11.5.3 lpuGetStatus()

Synopsis

Get the Status of the LPU, I.e. Check if it is running of floating (i.e. disconnected from the local on-board buses).

Definition

```
ThorRc lpuGetStatus(
  short       boardNo,   // Number of the Thor-2 Board Hosting the LPU
  LpuStatusT *lpuStatus  // Current Status of the LPU (Output)
);
```

Returns

```
THOR_SUCCESS          // OK
THOR_INVALID_BOARD_NO // The function was supplied an invalid board number
```

11.5.4 lpuInstallIsr()

Synopsis

Loads an interrupt service routine for the LPU into the on-board memory (RAM).

Definition

```
ThorRc lpuInstallIsr(
  short boardNo,       // Number of the Thor-2 Board Hosting the LPU
  int   lpuIsrVect,    // Number of the LPU interrupt vector
  Ulong lpuIsrAddr,    // Starting address of the LPU isr Vector
  Byte  *isr,          // Interrupt Service routine
  Word  isrLen         // Length of the Interrupt service routine
);
```

Returns

```
THOR_SUCCESS          // OK
THOR_INVALID_BOARD_NO // The function was supplied an invalid board number
```

11.5.5 lpuIntr()

Synopsis

Generates an interrupt towards the LPU.

Definition

```
ThorRc lpuIntr(
  short boardNo        // Number of the Thor-2 Board Hosting the LPU
);
```

Returns

```
THOR_SUCCESS          // OK
THOR_INVALID_BOARD_NO // The function was supplied an invalid board number
```

11.5.6 lpuLoadApp()

Synopsis

Loads an application program for the LPU into the on-board memory (RAM).

Definition

```
ThorRc lpuLoadApp(
  short boardNo,                  // Number of the Thor-2 Board
  char *fileName,                 // The name of the file to be loaded
  Ulong baseAddr,                 // Starting Address in the on-board
                                  // memory where to load the file
  Ulong *noOfBytes                // Number of bytes in the file
                                  // (output from this function)
);
```

Returns

```
THOR_SUCCESS          // OK
THOR_FILE_NOT_FOUND   // Unable to find the file
THOR_INVALID_BOARD_NO // The function was supplied an invalid board number
```

11.6 Flash Memory Functions - *flash.h*

11.6.1 flshCheckSectorUsage()

Synopsis

Counts the number of words in use in a specific sector of the Flash. I.e. count all the word that are not equal to 0xFFFF.

Note: Locations containing data 0xFFFF are treated as non-used locations, which may result in an inaccurate count.

Definition

```
ThorRc flshCheckSectorUsage(
  short   boardNo,     // Number of the Thor-2 board hosting the flash
  short   sectNo,      // Number of the sector to be examined
  Ulong   *usage       // Usage count (output from this function)
);
```

Returns

```
THOR_SUCCESS                // OK
THOR_NO_MEM_WIN            // No Memory Window Configured, Cannot access memory
THOR_INVALID_BOARD_NO      // The function was supplied an invalid board number
THOR_FLASH_INVALID_SECT_NO  // The provided sector number is not valid
```

See Also

```
flshCheckUsage()
```

11.6.2 flshCheckUsage()

Synopsis

Counts the number of words in use in the entire Flash. I.e. count all the word that are not equal to 0xFFFF.

Note: Locations containing data 0xFFFF are treated as non-used locations, which may result in an inaccurate count.

Definition

```
ThorRc flshCheckUsage(
  short boardNo        // Number of the board hosting the flash memory
  Ulong   *usage       // Usage count (output from this function)
);
```

Returns

```
THOR_SUCCESS                // OK
THOR_NO_MEM_WIN            // No Memory Window Configured, Cannot access memory
THOR_INVALID_BOARD_NO      // The function was supplied an invalid board number
```

See Also

`flshCheckSectorUsage()`

11.6.3 flshEraseSector()

Synopsis

Erases (empties) one of the flash's 7 user sectors. Note: the user can erase and rewrite sectors number: 1-6 and 9. For more information on the Thor-2 flash sectors, please see the Thor-2 Technical Description.

Definition

```
ThorRc flshEraseSector(
  short boardNo,            // Number of the board hosting the flash memory
  short sectNo              // Number of the sector to be erased (0 - 6)
);
```

Returns

```
THOR_SUCCESS                // Sector was erased
THOR_FLASH_PRG_FAIL         // Erasure Failed
THOR_INVALID_BOARD_NO      // The function was supplied an invalid board number
THOR_FLASH_INVALID_SECT_NO  // The provided sector number is not valid
```

11.6.4 flshLoadData()

Synopsis

Loads an array of data bytes anywhere into the flash memory.

Definition

```
ThorRc flshLoadData(
  short   boardNo,            // Number of the board hosting the flash memory
  Byte   *data,               // Array of data bytes to be written to flash
  Ulong   dataLen,            // Length of the data array
  Ulong   targetBaseAddr      // Absolute memory start address where to place
                              // the data in target memory (must be even)
);
```

Returns

```
THOR_SUCCESS              // OK
THOR_FLASH_BAD_ADDR       // The Address supplied was not even or not
                          // within the flash  memory area
THOR_FLASH_PRG_FAIL       // The Flash write operation failed, the flash may
                          // be corrupted
THOR_INVALID_BOARD_NO     // The supplied board number is not valid
```

11.6.5 flshLoadPrg()

Synopsis

Loads (writes) an arbitrary program or data sequence into the flash memory.

Definition

```
ThorRc flshLoadPrg(
  short   boardNo,          // Number of the board hosting the flash memory
  char *binFileName,        // Name of file containing the binary code
  Ulong targetBaseAddress   // Absolute starting memory address where to place
                            // the data in target memory (must be even)
);
```

Returns

```
THOR_SUCCESS              // Loading was successful
THOR_FLASH_BAD_FILE       // File was corrupt
THOR_FLASH_PRG_FAIL       // Loading Failed
THOR_FLASH_BAD_ADDR       // Provided Target address was invalid
THOR_INVALID_BOARD_NO     // The supplied board number is not valid
```

See Also

`flshLoadData()`

11.6.6 flshReadMaintSect()

Synopsis

Reads the Thor-2 Maintenance sector from the flash (sector number 10) into a struct. This sector contains information about Boot Vectors and Device Revisions on the board.

Definition

```
ThorRc flshReadMaintSect(
```

```
  short boardNo,              // Number of the board hosting the flash memory
  MaintDataT *md              // Maintenance Data (Boot vectors and revisions)
);
```

Returns

```
THOR_SUCCESS              // OK
THOR_NO_MEM_WIN           // No Memory Window Configured, Cannot access memory
THOR_INVALID_BOARD_NO     // The supplied board number is not valid
```

See Also

```
drvReadConfigData()
```

11.6.7 flshWriteMem()

Synopsis

Writes a data word (16 bits) into the Flash memory.

Note: The address to be written to must be an even address.

Note: This function cannot be used to overwrite old data. The sector must first be erased (i.e. make all locations contain 0xFFFF) and then this function can be used to write into an empty sector.

Definition

```
ThorRc flshWriteMem(
  short boardNo,    // Number of the board hosting the flash memory
  Ulong addr,       // Absolute memory start address where to place
                    // the data in target memory (must be even)
  Word  data        // Data to be written
);
```

Returns

```
THOR_SUCCESS              // OK
THOR_FLASH_BAD_ADDR       // The Address supplied was not even or not
                          // within the flash memory area
THOR_FLASH_PRG_FAIL       // The Flash write operation failed, the flash may
                          // be corrupted
THOR_INVALID_BOARD_NO     // The supplied board number is not valid
```

See Also

```
fldhEraseSector()
```

11.7 Codec Functions - *cd.h*

11.7.1 cdConnectDtmf()

Synopsis

Connects the DTMF chip to the codec transmit path, so that the DTMF chip can be used to generate and receive DTMF tones.

Definition

```
ThorRc cdConnectDtmf(
  short boardNo,       // Number of the Thor-2 Board Hosting the Codec
  short codecNo        // Number of the Codec
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

11.7.2 cdConnectHandsetMic()

Synopsis

Connects handset microphone to one of the codecs.

Definition

```
ThorRc cdConnectHandsetMic(
  short boardNo,       // Number of the board hosting the codec
  short codecNo        // Number of the codec to be connected
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

```
cdDisconnectHandsetMic()
cdConnectHandsetSpeaker()
```

```
cdConnectHandsfreeSpeaker()
```

11.7.3 cdConnectHandsetSpeaker()

Synopsis

Connects a handset speaker (ear piece) to a codec.

Definition

```
ThorRc cdConnectHandsetSpeaker(
  short boardNo,       // Number of the board hosting the codec
  short codecNo        // Number of the codec to be connected
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

```
cdDisconnectHandsetSpeaker()
cdConnectHandsetMic()
cdConnectHandsfreeSpeaker()
```

11.7.4 cdConnectHandsfreeSpeaker()

Synopsis

Connects a hands free speaker to a codec.

Definition

```
ThorRc cdConnectHandsfreeSpeaker(
  short boardNo,       // Number of the board hosting the codec
  short codecNo        // Number of the codec to be connected
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

```
cdDisconnectHandsfreeSpeaker()
cdConnectHandsetMic()
cdConnectHandsetSpeaker()
```

11.7.5 cdDigitalGain()

Synopsis

Sets the Digital gain in a codec (for transmit and received directions). The digital gain can be set in 3dB increments. The total gain for the Codec is the sum of the Digital and Filter gain.

Definition

```
ThorRc cdDigitalGain(
  short boardNo,              // Number of the board hosting the codec
  short codecNo,              // Number of the codec to be adjusted
  CdDigitalGainT txGain,      // Transmit Gain
  CdDigitalGainT rxGain       // Receive Gain
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

cdFilterGain()

11.7.6 cdDisconnectHandsetMic()

Synopsis

Disconnects a handset speaker from a codec (if it has been previously connected with cdConnectHandsetMic().

Definition

```
ThorRc cdDisconnectHandsetMic(
  short boardNo,       // Number of the board hosting the codec
  short codecNo        // Number of the codec to be connected
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

```
cdConnectHandsetMic()
cdDisconnectHandsetSpeaker()
cdDisconnectHandsfreeSpeaker()
```

11.7.7 cdDisconnectHandsetSpeaker()

Synopsis

Disconnects a handset speaker (ear piece) from a codec (if it has been previously connected with cdConnectHandsetSpeaker()).

Definition

```
ThorRc cdDisconnectHandsetSpeaker(
  short boardNo,       // Number of the board hosting the codec
  short codecNo        // Number of the codec to be connected
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

```
cdConnectHandsetSpeaker()
cdDisconnectHandsetMic()
cdDisconnectHandsfreeSpeaker()
```

11.7.8 cdDisconnectHandsfreeSpeaker()

Synopsis

Disconnects a handset speaker from a codec (if it has been previously connected with cdConnectHandsfreeSpeaker())

Definition

```
ThorRc cdDisconnectHandsfreeSpeaker(
  short boardNo,       // Number of the board hosting the codec
  short codecNo        // Number of the codec to be connected
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

```
cdConnectHandsfreeSpeaker()
cdDisconnectHandsetMic()
cdDisconnectHandsetSpeaker()
```

11.7.9 cdFilterGain()

Synopsis

Sets the Codec Filter gain in both receive and transmit directions. The total gain for the Codec is the sum of the Digital and Filter gain.

Definition

```
ThorRc cdFilterGain(
  short boardNo,     // Number of the board hosting the codec
  short codecNo,     // Number of the codec to be adjusted
  short txGain,      // Transmit Gain in dB (0 dB through 7dB)
  short rxGain       // Receive Gain in dB (-7 dB through 0dB)
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_INVALID_TX_GAIN // Supplied Transmit (TX) gain is invalid
THOR_CD_INVALID_RX_GAIN // Supplied Receive (RX) gain is invalid
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

11.7.10 cdInit()

Synopsis

Initializes a Codec chip to use a specified coding law and code assignment.

Definition

```
ThorRc cdInit(
  short boardNo,     // Number of the Thor-2 Board Hosting the Codec
  short codecNo,     // Number of the Codec
  CdLawT law,        // u-law or A-law
  CdCodeT code       // sign magnitude or CCITT code assignment
                     // (for input/output)
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_INVALID_LAW     // Supplied Codec Coding Law is invalid
THOR_CD_INVALID_CODE    // Supplied Codec Coding code is invalid
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

11.7.11 cdMuteOff()

Synopsis

Enables the phone (handset) after it has been muted with *cdMuteOn()*.

Definition

```
ThorRc cdMuteOff(
  short boardNo,       // Number of the Thor-2 Board Hosting the Codec
  short codecNo        // Number of the Codec
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

```
cdMuteOn()
```

11.7.12 cdMuteOn()

Synopsis

Mutes the phone (handset). To enable the handset again, use *cdMuteOff()*.

Definition

```
ThorRc cdMuteOn(
  short boardNo,       // Number of the Thor-2 Board Hosting the Codec
  short codecNo        // Number of the Codec
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

```
cdMuteOff()
```

11.7.13 cdReset()

Synopsis

Performs a reset on a Codec chip.

Definition

```
ThorRc cdReset(
  short boardNo,       // Number of the board hosting the codec
  short codecNo        // Number of the codec to be reset
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

11.7.14 cdSendDtmf()

Synopsis

Sends a DTMF tone from the codec. Note that both the Codecs and the DTMF transceivers can be used to send DTMF tones. However, only the DTMF transceivers are capable of receiving and detecting DTMF tones.

Definition

```
ThorRc cdSendDtmf(
  short boardNo,     // Number of the Thor-2 Board Hosting the Codec
  short codecNo,     // Number of the Codec
  char cDigit        // The DTMF digit to be sent
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

```
dtmfSend()
dtmfSendBurst()
```

11.7.15 cdVoiceSideToneOff()

Synopsis

Turns the side tone in the speaker off.

Definition

```
ThorRc cdVoiceSideToneOn(
  short boardNo,       // Number of the board hosting the codec
  short codecNo        // Number of the codec in use
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

```
cdVoiceSideToneOn()
```

11.7.16 cdVoiceSideToneOn()

Synopsis

Turns the side tone in the speaker on.

Definition

```
ThorRc cdVoiceSideToneOn(
  short boardNo,       // Number of the board hosting the codec
  short codecNo        // Number of the codec in use
);
```

Returns

```
THOR_SUCCESS            // OK
THOR_CD_COMM_FAILURE    // Communication to the Codec Failed
THOR_INVALID_CODEC_NO   // The Supplied Codec number is not valid
THOR_INVALID_BOARD_NO   // The Supplied Codec number is not valid
```

See Also

```
cdVoiceSideToneOff()
```

11.8 DTMF Transceiver Function - *dtmf.h*

11.8.1 dtmfBurstStatus()

Synopsis

Checks whether the DTMF transceiver is busy sending tones or idle and ready to send.

Definition

```
ThorRc dtmfBurstStatus(
  short boardNo,         // Number of the board hosting the DTMF chips
  short dtmfNo           // DTMF chip number
);
```

Returns

```
THOR_SUCCESS             // not busy sending DTMF tones
THOR_DTMF_BUSY           // busy sending DTMF tones
THOR_INVALID_BOARD_NO    // Supplied Board Number is not valid
THOR_INVALID_DTMF_NO     // Supplied DTMF Number is not valid
```

11.8.2 dtmfEnable()

Synopsis

Enables a DTMF Transceiver chip. This function must be called before the DTMF chips are used.

Definition

```
ThorRc dtmfEnable(
  short boardNo,             // Number of the board hosting the DTMF chips
  short dtmfNo               // Number of the DTMF chip
  Bool store                 // Determines whether to store all
                             // detected DTMF tones in the FIFO.
);
```

Returns

```
THOR_SUCCESS             // OK
THOR_INVALID_BOARD_NO    // Supplied Board Number is not valid
THOR_INVALID_DTMF_NO     // Supplied DTMF Number is not valid
```

11.8.3 dtmfReceived()

Synopsis

A non-blocking function that either returns the latest detected DTMF tone or THOR_NO_TONE if there was no tone received.

Note that the codec must be properly cross-connected with *tssXConnect()* prior to calling this function. If there was one or more tones (THOR_SUCCESS), they can be read with the *drvRead()* function (like any other message) if the store option is set in the call to *dtmfEnable()*. This function stores the latest DTMF digit that was detected, and it returns that value in the 'dtmfDigit' parameter. It then clears its internally stored digit.

Note also that DTMF chip 0 is always tied to Codec 0, and DTMF chip 1 is always tied to Codec 1.

Definition

```
ThorRc dtmfReceived(
  short boardNo,          // Board number
  short dtmfNo,           // Number of the Dtmf chip (0 or 1) to be used.
  char *dtmfDigit         // Last DTMF digit received (or \0 if none)
);
```

Returns

```
THOR_SUCCESS             // a DTMF tone was detected
THOR_DTMF_NO_TONE        // no DTMF tones were detected
THOR_INVALID_BOARD_NO    // Supplied Board Number is not valid
THOR_INVALID_DTMF_NO     // Supplied DTMF Number is not valid
```

11.8.4 dtmfReset()

Synopsis

Initializes and resets a DTMF chip.

Definition

```
ThorRc dtmfReset(
  short boardNo,               // Number of the board hosting the DTMF chips
  short dtmfNo                 // Number of the DTMF chip
);
```

Returns

```
THOR_SUCCESS             // OK
THOR_INVALID_BOARD_NO    // Supplied Board Number is not valid
THOR_INVALID_DTMF_NO     // Supplied DTMF Number is not valid
```

11.8.5 dtmfSend()

Synopsis

Sends DTMF tones in a non-burst mode; I.e., the digits are sent with specified tone on and pause intervals.

Definition

```
ThorRc dtmfSend(
  short boardNo,               // Number of the board hosting the DTMF chips
  short dtmfNo,                // DTMF chip number
  char *digits,                // Digits to be sent
  Word onTime,                 // Time in msec to keep the tone on
  Word offTime                 // Time un msec between the tones
);
```

Returns

```
THOR_SUCCESS             // OK
THOR_INVALID_BOARD_NO    // Supplied Board Number is not valid
THOR_INVALID_DTMF_NO     // Supplied DTMF Number is not valid
```

See Also

```
dtmfSendBurst()
thorSendDtmf()
```

11.8.6 dtmfSendBurst()

Synopsis

Sends DTMF tones in burst mode. I.e. the tones are send approximately with 51 msec +- 1 msec tone on and pause intervals.

Definition

```
ThorRc dtmfSendBurst(
  short boardNo,                // Number of the board hosting the DTMF chips
  short dtmfNo,                 // DTMF chip number
  char *digits                  // Digits to be sent
```

```
);
```

Returns

```
THOR_SUCCESS             // OK
THOR_INVALID_BOARD_NO    // Supplied Board Number is not valid
THOR_INVALID_DTMF_NO     // Supplied DTMF Number is not valid
```

See Also

```
thorSendDtmf()
```

11.8.7 dtmfToneOff()

Synopsis

Turns off a constant DTMF tone generation.

Definition

```
ThorRc dtmfToneOff(
  short boardNo,                  // Number of the board hosting the DTMF chips
  short dtmfNo,                   // DTMF chip number
);
```

Returns

```
THOR_SUCCESS             // OK
THOR_INVALID_BOARD_NO    // Supplied Board Number is not valid
THOR_INVALID_DTMF_NO     // Supplied DTMF Number is not valid
```

See Also

```
dtmfToneOn()
```

11.8.8 dtmfToneOn()

Synopsis

Turns on a constant DTMF tone generation.

Definition

```
ThorRc dtmfToneOn(
  short boardNo,                 // Number of the board hosting the DTMF chips
  short dtmfNo,                  // DTMF chip number
  short digit                    // Digit to be send
);
```

Returns

```
THOR_SUCCESS             // OK
THOR_INVALID_BOARD_NO    // Supplied Board Number is not valid
THOR_INVALID_DTMF_NO     // Supplied DTMF Number is not valid
```

See Also

`dtmfToneOff()`

12. High-Level API - *thorhapi.h*

12.1 Driver Functions

12.1.1 thorIdentDriver( )

Synopsis

Returns a pointer to an identification string for the driver. This function is useful in situations where the driver and the applications are not statically linked, and where the application may want to query for the revision or name of a currently dynamically linked driver.

The identification string contains the Odin TeleSystems' product number, the driver revision, and the date the driver was compiled.

Definition

```
char *thorIdentDriver(
void
);
```

Returns

Pointer to a string containing the driver identification.

See Also

`drvIdent()`

12.1.2 thorConstructDriver( )

Synopsis

Initializes the driver. This function needs to be executed before the high-level driver can be used. The function reads the configuration data from the flash and sets up the sliding memory, the I/O windows, and the clock source. It also checks for the existence of the board, resets the devices on the board, sets the clock source, boots up the LPU, and initializes all the driver data structures.

If a valid IRQ number is provided, the *thorConstructDriver()* function installs the interrupt service routine. If the provided IRQ number is THOR_NO_IRQ (-1), the driver will be set up in a polling mode.

Definition

```
ThorRc thorConstructDriver(
  short boardType,               // Type of the board used. Boardtype for
                                 // Thor-2 is 106.
  short boardBaseAddr,           // I/0-base address configured with a
                                 // DIP-switch on the board. Base address
                                 // for Thor boards is valid if:
                                 //   (0x000 < Addr <= 0x400)
                                 //      &&
                                 //   (Addr % 0x10) == 0.
                                 // Default value is 0x280.
  short noOfBoards,              // Number of Boards installed (up to 4).
  short irq,                     // IRQ for all the installed THOR boards
                                 // (they share the same IRQ), or
                                 // THOR_NO_IRQ if the polling mode should
                                 // be used.
  Word infoBufferSize,           // Size of fifo for board status messages
                                 // Recommended value: 2048
  Word liHdlcBufferSize,         // Size of the HDLC fifo in each Li. This
                                 // fifo is only used if an HDLC pipe
                                 // is configured for Sa or Si bits (see
                                 // thorConfigureSaBitPipe() ). If an Sa
                                 // bit pipe is not used then this size
                                 // could be set to 0.
  short hostIoOffset[],          // Table of absolute addresses of the IO
                                 // windows in the host (e.g. 0x290).
  short hostIoWindowSize[],      // Table of sizes (no of bytes) of the IO
                                // windows in the host (4 <= size <= 1kByte).
                                 // Most common value would be 16.
  Ulong hostMemoryOffset[],      // Table of absolute addresses of the
                                 // memory windows in the host
                                 // (e.g. 0x0D0000).
  Ulong hostMemoryWindowSize[],  // Table of sizes (no of bytes) of the
                                 // memory windows in the host
                                 // (256 <= size <= 16MByte). Typical value
                                 // would be 16kBytes, 32kBytes, or 64kBytes.
  Byte cpuIrqMask[],             // Table of interrupt masks. A '1' in
                                 // the mask will enable the chip interrupt
                                 // to be forwarded to the CPU.
  Byte lpuIrqMask[],             // Table of interrupt masks. A '1' in the
                                 // mask will enable the chip interrupt to
                                 // be forwarded to the LPU.
  short maxFrameLen[]            // Maximum frame length (common for
                                 // all pipes) Indexed per board.
```

```
);
```

Returns

```
THOR_SUCCESS                // Construction of the Driver successful
THOR_WRONG_CONTEXT          // Driver already constructed
THOR_INVALID_BOARD_TYPE
THOR_INVALID_BOARD_NO
THOR_OUT_OF_MEMORY
THOR_FPGA_NOT_LOADED
THOR_INVALID_ADDRESS
THOR_NO_BOARD             // No board found at the provided I/O address
THOR_INVALID_BOARD_NO
```

See Also

```
thorDestructDriver()
drvInit()
drvSetupMemWin()
drvSetupIoWin()
```

12.1.3 thorDestructDriver( )

Synopsis

Releases the driver from memory. This function must be called before the application is exited. Cleans up and clears the driver after use. Uninstalls the Interrupt Service Routines if installed by the *thorConstructDriver().*

Definition

```
ThorRc thorDestructDriver(
  void
);
```

Returns

```
THOR_SUCCESS
THOR_WRONG_CONTEXT            // No driver constructed
```

See Also

```
thorConstructDriver()
```

12.1.4 thorRegisterCallback()

Registers a callback function (implemented by the application) which will be called by the driver upon reception of a message or a hardware/line status change. The use of a callback function allows implementation of event driven applications.

NOTE: Only available with Windows 95 and Windows NT drivers. DOS applications must poll the driver.

Definition

```
ThorRc thorRegisterCallback(
  void (*hapiCallBack)(void)    // function pointer to the callback function in
                                // the application to be called from the driver
);
```

Returns

```
THOR_SUCCESS                          // OK
THOR_INVALID_CALLBACK_FUNCTION        // Invalid function pointer
THOR_CALLBACK_ALREADY_SET             // Callback function can be set only once
THOR_UNABLE_TO_CREATE_CALLBACK_THREAD // Creation of a new thread failed
```

See Also

```
drvRegisterCallback()
IOCTL_START_EVENT_NOTIFICATIONS
```

12.1.5 thorResetDriver( )

Synopsis

The *thorResetDriver()* function resets the driver software. Clears the dynamic data such as the message FIFOs in the host memory.

Definition

```
ThorRc thorResetDriver(
  void
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
thorResetLi()
drvResetDriver()
```

```
drvResetDevices()
```

12.1.6 thorSetCpuIntrMask( )

Synopsis

The *thorSetCpuIntrMask()* function is used to specify which device (chip) interrupts from the board are forwarded to the host processor to be serviced. The value in parameter cpuIrqMask will be written to the CPU Interrupt Mask (CIM) register. The bits in the CIM register are described in Table 6 on page 143. For more information on the Thor-2 internal registers, please refer to the *Thor-2 Technical Description* (Odin TeleSystems Inc. Doc. No. 1112-1-HSA-1002-1).

TABLE 6. CIM Bit Description

Bit	Function
0	0: Mask Interrupts from the HDLC Controller 1: Allow Interrupts
1	0: Mask Interrupts from Li0 1: Allow Interrupts
2	0: Mask Interrupts from Li1 1: Allow Interrupts
3	0: Mask Interrupts from the Time-Space Switch 1: Allow Interrupts
4	0: Mask Interrupts from DTMF Transceiver 0 1: Allow Interrupts
5	0: Mask Interrupts from DTMF Transceiver 1 1: Allow Interrupts
6	0: Mask Interrupts from the LPU 1: Allow Interrupts

Definition

```
ThorRc thorSetCpuIntrMask(
  short boardNo,          // Board number
  short cpuIrqMask        // A '1' in the mask will enable the chip IRQ
                          // to be forwarded to the CPU
);
```

Returns

```
THOR_SUCCESS
```

See Also

`thorSetLpuIntrMask()`

12.1.7 thorSetLpuIntrMask( )

Synopsis

The *thorSetLpuIntrMask()* function is used to specify which device (chip) interrupts from the board are forwarded to the on-board processor (LPU) to be serviced. The value in parameter lpuIrqMask will be written to the LPU Interrupt Mask (LIM) register. The bits in the LIM register are described in Table 7 on page 144. For more information on the Thor-2 internal registers, please refer to the *Thor-2 Technical Description* (Odin TeleSystems Inc. Doc. No. 1112-1-HSA-1002-1).

TABLE 7. LIM Bit Description

Bit	Function
0	0: Mask Interrupts from the CPU 1: Allow Interrupts
1	0: Mask Interrupts from the HDLC Controller 1: Allow Interrupts
2	0: Mask Interrupts from Li0 1: Allow Interrupts
3	0: Mask Interrupts from Li1 1: Allow Interrupts
4	0: Mask Interrupts from the Time-Space Switch 1: Allow Interrupts
5	0: Mask Interrupts from DTMF Transceiver 0 1: Allow Interrupts
6	0: Mask Interrupts from DTMF Transceiver 1 1: Allow Interrupts

Definition

```
ThorRc thorSetLpuIntrMask(
  short boardNo,          // Board number
  short lpuIrqMask        // A '1' in the mask will enable the chip IRQ
                          // to be forwarded to the CPU
);
```

Returns

`THOR_SUCCESS`

See Also

`thorSetCpuIntrMask()`

12.2 Line Interface Functions

12.2.1 thorConfigureLi( )

Synopsis

The function *thorConfigureLi()* sets up one Line Interface chip in either T1 or E1 mode.

Note: This function uses the configuration parameters specified in the Thor-2 configuration file and stored in the on-board flash for the physical layer configuration.

Definition

```
ThorRc thorConfigureLi(
  short boardNo,     // Board no for the Li chip.
  short liNo,        // Number of the Li chip to be configured.
  LiMode liMode      // Mode to be configured to: T1 or E1
);
```

Returns

```
THOR_SUCCESS
THOR_CONFIG_FAILURE
```

See Also

```
liInitDeviceT1()
liInitDeviceE1()
```

12.2.2 thorGetStatusLi( )

Synopsis

The function *thorGetStatusLi()* reports the status of an Line Interface.

Definition

```
ThorRc thorGetStatusLi(
  short boardNo,               // Board number.
  short liNo                   // Number of the Line Interface to be read.
);
```

Returns

```
THOR_L1_OK
THOR_L1_DOWN
```

See Also

`liGetStatus()`

12.2.3 thorSaBytesOn( )

Synopsis

The *thorSaBytesOn()* function sets the values to be sent in the SaX bits (X = 4 to 8) . Eight bits (1 byte) for each SaX are specified at the same time (same function call).

In CRC-Multiframe format (16 frames per multi-frame), one bit from each SaX is sent in time-slot 0 in every odd frame (in frames that do not contain frame alignment information). The least significant bit of the saXVal byte is sent in frame 1 of the multiframe and the most significant bit of the Byte is sent in frame 15 of the multiframe (i.e LSb is shifted out first).

In Doubleframe format one bit of each saXVal byte is sent in every frame that does not contain alignment info (every other frame). LSb is shifted out first.

NOTE: Only meaningful in E1 mode

Definition

```
ThorRc thorSaBytesOn(
  short boardNo,           // Board number.
  short liNo,              // Number of the Line Interface to send the service
                           // word from.
  Byte sa4Val,    // Sa4 bits to be sent. LSB will be sent first.
  Byte sa5Val,    // Sa5 bits to be sent. LSB will be sent first.
  Byte sa6Val,    // Sa6 bits to be sent. LSB will be sent first.
  Byte sa7Val,    // Sa7 bits to be sent. LSB will be sent first.
  Byte sa8Val     // Sa8 bits to be sent. LSB will be sent first.
);
```

Returns

`THOR_SUCCESS`

See Also

`thorSaBytesOff()`

12.2.4 thorSaBytesOff( )

Synopsis

The *thorSaBytesOff()* function returns the Thor-2 board to normal Sa-bit operation. The LIs will stop sending the Sa byte value specified in the *thorSaBytesOn()* function.

NOTE: Only meaningful in E1 mode

Definition

```
ThorRc thorSaBytesOff(
  short boardNo,          // Board number.
  short liNo              // Number of the Line Interface to stop
                          // sending the service word from.
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
thorSaBytesOff()
```

12.2.5 thorGetSaBitValue( )

Synopsis

The *thorGetSaBitValue()* function fetches the value to of the received SaX bits. Returns a byte (8-bits) received during the last CRC-Multiframe.

NOTE: Only meaningful in E1 mode

Definition

```
ThorRc thorGetSaBitValue(
  short        boardNo,          // Board number.
  short        liNo,             // Line Interface number
  LiSaBit    saBit,              // Sa bit to read
  Byte *saVal                    // Last received 8-bits
);
```

Returns

```
THOR_SUCCESS
```

See Also

`thorGetSiBitValue()`

12.2.6 thorSetSiBitValue( )

Synopsis

The *thorSetSiBitValue()* function sets the value to be sent at the Si bit positions (Spare bits for international use). In Doubleframe format, these are the first bits of each frame.In CRC-Multiframe format, the Si bit are the first bits of frames 13 and 15. In CRC-Multiframe format these bits are also known as the E-bits.

NOTE: Only meaningful in E1 mode

Definition

```
ThorRc thorSetSiBitValue(
  short boardNo,        // Board number.
  short liNo,           // Number of the Line Interface to send the
                        // service word from.
  Byte si1Val,          // Value of the FAS Si-bit in doubleframe format
                        // or Si (E) bit in frame 13 in CRC-multiframe
                        // format
  Byte si2Val           // Value of the service word Si bit in DoubleFrame
                        // format or Si (E) bit in frame 15 in
                        // CRC-multiframe format
);
```

Returns

`THOR_SUCCESS`

See Also

`thorGetSiBitValue()`

12.2.7 thorGetSiBitValue( )

Synopsis

The *thorGetSiBitValue()* function fetches the value to Si bits received in the last frame.

In Doubleframe format, these are the first bits of each frame. In CRC-Multiframe format, the Si bit are the first bits of frames 13 and 15. In CRC-Multiframe format these bits are also known as the E-bits.

NOTE: Only meaningful in E1 mode

Definition

```
ThorRc thorGetSiBitValue(
  short  boardNo,        // Board number.
  short  liNo,           // Line Interface number
  Byte *si1Val,          // Value of the FAS Si-bit in doubleframe format
                         // or Si (E) bit in frame 13 in CRC-multiframe
                         // format
  Byte *si2Val           // Value of the service word Si bit in
                         // DoubleFrame format
                         // or Si (E) bit in frame 15 in CRC-multiframe format
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
thorSetSiBitValue()
```

12.2.8 thorAlarmOn( )

Synopsis

The *thorAlarmOn()* function begins transmitting an Alarm towards the Remote End. An Auxiliary Pattern (AUXP) is an unframed signal that contains an continuous alternating bit stream (010101...). The AUXP can be used when Loss of Signal (LOS) has been detected by the receiver. The Alarm Indication Signal (AIS) is an unframed signal that contains an continuous bit stream of 1's. The Remote Alarm Indication (RAI) is send by setting the A-bit (bit 3 in time-slot 0 of E1) to 1.

Note: If both AUXP and AIS is specified, AIS will be send.

Definition

```
ThorRc thorAlarmOn(
  short boardNo,          // Board number.
  short liNo,            // Number of the Line Interface to send the alarm from.
  LiAlarmType alarmType   // Alarm type to send
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
thorAlarmOff()
```

12.2.9 thorAlarmOff( )

Synopsis

The *thorAlarmOff()* function turns the alarm transmission towards the Remote End off.

Definition

```
ThorRc thorAlarmOff(
  short boardNo,          // Board number.
  short liNo,             // Number of the LI to send the alarm from
  LiAlarmType alarmType   // Alarm type to send
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
thorAlarmOn()
```

12.2.10 thorResetLi( )

Synopsis

The function *thorResetLi()* performs a hardware reset on the Line Interface.

Definition

```
ThorRc thorResetLi(
  short liNo              // Number of the Line Interface to reset
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
thorResetDriver()
thorResetHdlc()
```

12.3 Switching

12.3.1 thorConnectChannel( )

Synopsis

The *thorConnectChannel()* function connects one channel from an incoming highway in to another channel in an outgoing highway.

Definition

```
ThorRc thorConnectChannel(
  short          boardNo,      // Board number.
  ThorPhwType  inHighway,      // Incoming PCM highway.
  short        inChannel,      // The channel in the incoming highway to be
                               // connected to outChannel.
  ThorPhwType  outHighway,     // Outgoing PCM highway.
  short        outChannel      // The channel in the outgoing highway to be
                               // connected to inChannel.
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
thorDisconnectChannel()
```

12.3.2 thorDisconnectChannel( )

Synopsis

The *thorDisonnectChannel()* function disconnect channels connected with the *thorConnectChannel()* function.

Definition

```
ThorRc thorDisconnectChannel(
  short boardNo,               // Board number.
  ThorPhwType  outHighway,     // Outgoing PCM highway.
  short        outChannel      // The channel in the outgoing highway to be
                               // disconnected from whichever inChannel it
                               // is currently connected to)
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
thorConnectChannel()
thorDisconnectAllChannels()
```

12.3.3 thorDisconnectAllChannels( )

Synopsis

The *thorDisonnectAllChannels()* function disconnects all outgoing channels from whichever channels they are connected to (on one Time-Space Switch on one board).

Definition

```
ThorRc thorDisconnectAllChannels(
  short boardNo                  // Board number
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
thorConnectChannel()
thorDisconnectChannel()
```

12.4 Message Sending and Receiving

12.4.1 thorConfigurePipe( )

Synopsis

The function *thorConfigurePipe()* configures a pipe in the HDLC controller. A pipe can contain one time-slot, only certain bits of a time-slot (sub-channel), or several time-slots (super-channel). Each bit that is to be included in the pipe is passed as a bit rate mask. The bit rate mask is an array of 32 bytes, where index 0 is time-slot 0, etc. A '1' in a bit position indicates that the corresponding bit in the time-slot if included in the pipe. A pipe needs to be configured before data or HDLC frames can be received or sent.

Definition

```
ThorRc thorConfigurePipe(
  short boardNo,                // Board number.
  short pipeNo,                 // Pipe to be configured.
  Byte txBitRateMask[],         // Array of 32 8-bit masks. A '1' indicates
                                // that this bit of this channel (array
                                // index) in the THOR_PHW_CTRL highway is
                                // included in the
                                // transmit direction of the pipe.
  Byte rxBitRateMask[],         // Bit rate mask for the receive direction
                                // of the pipe. See txBitRateMask[] above.
  ThorFrameFillType frameFillType // What to send between frames. Either
                                // flags(0x7E) or all ones (0xFF)
);
```

Returns

```
THOR_SUCCESS        // Configuration successful
THOR_HDLC_AR_BUSY   // Configuration failed
```

12.4.2 thorRead( )

Synopsis

The function *thorRead()* fetches a received message. Driver will automatically sort the received messages from different pipes according to the time of reception. If several messages are waiting to be fetched, this function will fetch and return the oldest one (messages are buffered in realtime when they arrive, awaiting fetching by this function).

Definition

```
ThorRc thorRead(
  Byte   fmBuf[],             // Buffer into which the received message will
                              // be written. NOTE: The buffer must be
                              // allocated by the application.
  short           fmBufSize,  // Size of fmBuf[].
  ThorFrameHeader *fmHeader   // Pointer to header structure that will be
                              // fillied in by the function. NOTE: The struct
                              // must be allocated by the application.
);
```

Returns

```
THOR_SUCCESS     // A frame was read successfully
THOR_NO_FRAMES   // No complete frames ready for reading
```

See Also

```
thorWritePipe()
```

12.4.3 thorWritePipe( )

Synopsis

The function *thorWritePipe()* sends an HDLC message (including layer 2) on a Pipe.

Definition

```
ThorRc thorWritePipe(
  short           boardNo,    // Board number.
  short           pipeNo,     // The pipe to write to.
  Byte            fmBuf[],    // Buffer containing the message to be send.
  short           fmLength    // Length of the message to be sent.
);
```

Returns

```
THOR_SUCCESS        // Message accepted and is being sent
THOR_TX_BUSY        // Previous message not yet sent completely
                    // Try again later
THOR_MSG_TOO_LONG   // Message is too long to be sent... Try again later
```

See Also

```
thorRead()
```

12.4.4 thorResetHdlc( )

Synopsis

The *thorResetHdlc()* performs a hardware reset on the HDLC controller.

Definition

```
ThorRc thorResetHdlc(
  short boardNo           // Board number
);
```

Returns

`THOR_SUCCESS`

See Also

`thorResetDriver()`
`thorResetLi()`
`drvInitHdlc()`

12.5 Phone Functions

12.5.1 thorPhoneOn( )

Synopsis

The function *thorPhoneOn()* connects a handset to a specified Codec (CD0 or CD1). The appropriate data channel must have been properly cross-connected on the Time-Space Switch to the Codec prior to calling this function (see *thorConnectChannel()*). Note also that Codec 0 is communicating on channel 2 on the Auxiliary PCM highway. Codec 1 is communicating on channel 3 of the Auxiliary PCM highway.

Definition

```
ThorRc thorPhoneOn(
  short boardNo,          // Board number
  short codecNo,          // Number of the Codec to be used (0 or 1).
  CdLawT law,             // u-law or A-law
  CdCodeT code,           // sign magnitude or CCITT code assignment
                          // (for input/output)
  DtmfOptT dtmfOption     // Determines whether to enable DTMF detection and
                          // if so whether to store all detected DTMF tones
                          // in the FIFO.
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
thorPhoneOff()
thorConnectChannel()
```

12.5.2 thorPhoneOff( )

Synopsis

The function *thorPhoneOff()* disconnects the handset from the codec.

Definition

```
ThorRc thorPhoneOff(
  short boardNo,              // Board number.
  short codecNo               // Number of the Codec to be used (0 or 1).
);
```

Returns

`THOR_SUCCESS`

See Also

`thorPhoneOn()`

12.5.3 thorSendDtmf( )

Synopsis

The function *thorSendDtmf()* sends DTMF tones on a B-channel. The time between tones and the duration of the tone can be specified.

Note: The DTMF transceiver is connected to the codec (DTMF chip #0 to Codec chip #0 on aux time-slot #2 and DTMF chip #1 to Codec chip #1 on aux time -slot #3). The codec must be properly cross-connected in the time-space switch to a time-slot in an outgoing T1/E1 span for the tones to be send out.

Definition

```
ThorRc thorSendDtmf(
  short boardNo,          // Board number.
  short codecNo,          // Number of the Codec (0 or 1) to be used.
  char *phoneNumber,      // Phone number to dial.
  DtDurationT duration,   // Burst and pause duration
  CdLawT law,             // u-law or A-law
  CdCodeT code            // sign magnitude or CCITT code
                          // assignment (for input/output)
);
```

Returns

`THOR_SUCCESS`

12.5.4 thorReceivedDtmf( )

Synopsis

The *thorReceivedDtmf( )* function is a non-blocking function that either returns the latest detected DTMF tone or THOR_NO_TONE if there was no tone received.

If one or more tones had been detected (THOR_SUCCESS), they can be read with the *thorRead()* function (like any other message) if the dtmfOption is set to DT_DETECT_STORE. If the dtfmOption is set to DT_DETECT_LAST, then the digits will not be stored in the *thorRead()* fifo at all and can only be retrieved with this function.

The function stores the last DTMF digit that was detected, and it returns that value in the 'dtmfDigit' parameter. It then clears its internally stored digit.

Note: The DTMF transceiver is connected to the codec (DTMF chip #0 to Codec chip #0 on aux time-slot #2 and DTMF chip #1 to Codec chip #1 on aux time -slot #3). The codec must be properly cross-connected in the time-space switch to a time-slot in an outgoing T1/E1 span for the tones to be send out.

Definition

```
ThorRc thorReceivedDtmf(
  short boardNo,          // Board number.
  short dtmfNo,           // Number of the Dtmf chip (0 or 1) to be used.
  char *dtmfDigit         // Last DTMF digit received (or \0 if none)
);
```

Returns

```
THOR_SUCCESS      // a DTMF tone was detected
THOR_NO_TONE      // no tone was detected
```

See Also

```
thorSendDtmf()
```

12.6 Test Functions

12.6.1 thorByteOnBch( )

Synopsis

thorByteOnBch() begins sending a constant byte on the specified highway and time-slot and keeps sending it until turned off with the *thorByteOffCh()* function. This constant byte is inserted in every frame for the particular time-slot (channel). The channel byte can have the value between 0 and 255.

Definition

```
ThorRc thorByteOnCh(
  short boardNo,             // Board number
  ThorPhwType  outHighway,   // PCM Highway to send the byte on.
  short        outChannel,   // Channel (0-31) within the PCM highway to send
                             // the byte on.
  Byte aByte                 // Byte to be sent
);
```

Returns

```
THOR_SUCCESS
```

See Also

```
thorByteOffBch()
```

12.6.2 thorByteOffBch( )

Synopsis

The function *thorByteOffBch()* stops the sending of a constant byte value on the specified B-channel.

Definition

```
ThorRc thorByteOffCh(
  short boardNo,             // Board number.
  ThorPhwType  outHighway,   // PCM Highway to turn off the byte from.
  short        outChannel    // Channel (0-31) within the PCM highway to turn
                             // off the byte from.
);
```

Returns

THOR_SUCCESS

See Also

thorByteOnBch()

12.6.3 thorByteReadBch( )

Synopsis

The *thorByteReadBch()* function reads one byte from the specified channel and returns it in **recByte*. The sampling of the byte is done at no particular time. If the byte being read is not constant, random results should be expected. To verify that the channel in fact does carries constant bytes, several samples should be taken.

Definition

```
ThorRc thorByteReadCh(
  short          boardNo,    // Board number.
  ThorPhwType    inHighway,  // PCM Highway to read the byte from.
  short          inChannel,  // Channel (0-31) within the PCM highway to
                             // read the byte from.
  Byte          *recByte     // Received byte from the channel.
);
```

Returns

THOR_SUCCESS

Receiced Byte is returned in **recByte* parameter.

See Also

thorByteOnBch()
thorByteOffBch()

12.7 Miscellaneous

12.7.1 thorGetErrMsg( )

Synopsis

The *thorGetErrMsg()* function converts a Thor-function return code into a string. Returns a pointer to a string describing the error code in a general fashion. Can be used for quick and dirty solutions when the return code is not analyzed properly by the application but at least something needs to be printed.

Definition

```
char *thorGetErrMsg(
  ThorRc errCode         // Error code to be converted.
);
```

Returns

Pointer to a static string owned by the function.

See Also

```
drvStatus2Str()
*drvThorRc2Str()
```

12.7.2 thorBoardExistence( )

Synopsis

The *thorBoardExistence()* function checks if a Thor-2 board exists at a specified I/O address.

Definition

```
ThorRc thorBoardExistence(
  short boardNo,
  short aIoBaseAddr     // I/0-base address configured with a
                        // DIP-switch on the board. Base address
                        // for Thor boards is valid if:
                        //   (0x000 < Addr <= 0x400)
                        //      &&
                        //   (Addr % 0x10) == 0.
                        // Default value is 0x280.
);
```

Returns

```
THOR_SUCCESS   // if a THOR-2 board was found.
THOR_NO_BOARD  // if no THOR-2 board was found.
```

See Also

```
drvBoardExistence()
liExistenceChk()
```

12.7.3 thorLoopLi( )

Synopsis

The *thorLoopLi()* function loops the transmit lines of the Line Interface Transceiver to the receive lines.

Definition

```
ThorRc thorLoopLi(
  short boardNo,            // Board number.
  short liNo,               // Number of the line interface to be used.
  LiLoopT loopType          // Line loop or Remote loop
);
```

Returns

```
THOR_SUCCESS
```

13. Index

D

E

F

H

I

L

Doc. No. 1211-1-SDA-1001
For more information on this product, please contact:

Odin TeleSystems Inc.
800 E. Campbell Road, Suite 300
Richardson, Texas 75081-1873
U. S. A.

Tel: +1-972-664-0100
Fax: +1-972-664-0855
Email: Info@OdinTS.com
URL: http://www.OdinTS.com/