Technical Document - Confidential

G23 PROTOCOL STACK

TIF – TEST INTERFACE DRIVER

API

Document Number:	8434.103.01.002
Version:	0.4
Status:	Draft
Approval Authority:	
Creation Date:	2000-Jan-31
Last changed:	2015-Mar-08 by TI Employee
File Name:	tif_api.doc

Important Notice

Texas Instruments Incorporated and/or its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products, software and services at any time and to discontinue any product, software or service without notice. Customers should obtain the latest relevant information during product design and before placing orders and should verify that such information is current and complete.

All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment. TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI products, software and/or services. To minimize the risks associated with customer products and applications, customers should provide adequate design, testing and operating safeguards.

Any access to and/or use of TI software described in this document is subject to Customers entering into formal license agreements and payment of associated license fees. TI software may solely be used and/or copied subject to and strictly in accordance with all the terms of such license agreements.

Customer acknowledges and agrees that TI products and/or software may be based on or implement industry recognized standards and that certain third parties may claim intellectual property rights therein. The supply of products and/or the licensing of software does not convey a license from TI to any third party intellectual property rights and TI expressly disclaims liability for infringement of third party intellectual property rights.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products, software or services are used.

Information published by TI regarding third–party products, software or services does not constitute a license from TI to use such products, software or services or a warranty, endorsement thereof or statement regarding their availability. Use of such information, products, software or services may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Change History

Date	Changed by	Approved by	Version	Status	Notes
2000-Jan-31	MP et al		0.1		1
2001-Sep-07	MP et al		0.2		2
2003-Feb-28	RME		0.3		3
2003-Sep-16	TI Employee		0.4	Draft	

Notes:

1. Initial version
2. document number changed
3. new TIF header introduced

Table of Contents

List of Figures and Tables

List of References

[ISO 9000:2000] International Organization for Standardization. Quality management systems - Fundamentals and vocabulary. December 2000

1.1 References

[C_8415.026] Generic Driver Interface – Functional Specification; Condat 8415.026.99.012; March 19, 1999

[C_8415.033] VSI/PEI – Frame/Body Interfaces - Next Generation; Condat 8415.033.99.100

[mp_memo_tstuart] Concept for Integration of Drivers into the Test Interface; Condat internal, Nov 99

2 Introduction

G23 is a software package implementing Layers 2 and 3 of the ETSI-defined GSM air interface signaling protocol, and as such represents the part of a GSM mobile station's protocol software which is both, platform and manufacturer independent. Therefore, G23 can be viewed as a building block providing standardized functionality through generic interfaces for easy integration.

The G23 suite of products consists of the following items:

- Layers 2 and 3 for speech & short message services,
- Layers 2 and 3 for fax & data services,
- Application Control Interface,
- Slim MMI [02.30] and
- Test and integration support tools.

This document describes the functional interface of the G23 Layer 3 test interface driver, TIF. The design of the test interface driver structure is explained in more detail in [mp_memo_tstuart].

2.1 G23 Test Interface Layer 3: TIF driver header format

In the TIF driver, currently two distinct header formats are supported. The header precedes any data content of protocol primitives, system primitives or traces sent through a stack/tool side interface. When using an up-to-date FRAME on a sample target, only the new format will be used. Emulated stacks in a NucWin environment will offer both formats, actually.

Hence, building a NucWin stack requieres a decision regarding the Tif-header format. The format is specified in xxxxcomp.c. If not, unresolved reference whilst stack-linkage will occour:

```
#ifndef _TARGET_
BOOL newTstHeader = TRUE;
#endif
```

2.1.1 Old TIF header

The following tables show the structure of the old message format, used before FRAME 2.9.0.

Id	Time	Size	Sender	Receiver	[Opcode]	Data
1	4	4	4	4	[4/8]	Size-8 [-12/-16]
←		Header size	17|21|25	Bytes	→	

Parameter	Description	Format
Id	'P' for GSM-protocol primitives with 16 Bit opc 'Q' for GSM-protocol primitives with 32 Bit opc 'S' for sys-primitives, 'T' for traces	1 Byte ascii
Time	time since "System start" in milliseconds	4 Bytes ascii
Size	number of bytes beginning at 'sender'	4 Bytes ascii
Sender	name of process 'sender'	4 Bytes ascii
Receiver	name of process 'receiver'	4 Bytes ascii
Opcode	only for GSM-protocol primitives, primitive-opcode	4/8 Bytes ascii (opt.)
Data	data	Size – 8[-12] Bytes

Remarks:

- All messages may be encapsulated in an STX at the beginning and a LF at the end
- The names of sender and receiver must not be longer than 4 characters.
- the numbers like 4/8 are to be understood as the values for 16bit opc/32bit opc primitives

2.1.2 New TIF header

In contrary, the following tables show the structure of the new message format, used by FRAME 2.9.0 and later.

Inf.	Size	Time	Sender	Receiver	[Org. Rcv.	Opcode]	Data
1	2	4	4	4	[4	4]	Size-12 [-20]
←		Header size	15 or 23	Bytes		→	

Parameter	Description	Format
Id	Bits 7/6: '01' old header, '10' new header, others undefined 5/4: '01' for GSM-protocol primitives '10' for traces '11' for sys-primitives ('00' interface internal use only) 3/2: '01' milliseconds, '10' TDMA-frames 1/0: reserved for future use	1 Byte bitfield: 4x2 bits
Size	number of bytes beginning at 'time'	2 Bytes Hex
Time	time since "System start" in milliseconds or TDMA-frames	4 Bytes Hex
Sender	name of process 'sender'	4 Bytes ascii
Receiver	name of process 'receiver'	4 Bytes ascii
Org. Rcv.	name of original process 'org receiver'	4 Bytes ascii (opt.)
Opcode	only for GSM-protocol primitives, primitive-opcode	4 Bytes hex (opt.)
Data	data	Size –12[-20] Bytes

Remarks:

- Now a header is shortened by 2 Bytes, consuming less bandwidth on interface, though payload is increased.
- The header starts with an info byte, replacing the old Id-byte, providing 4 classes of 2-Byte header type information.
- The size field precedes the time field now. 'Size' is shortened to 2 bytes, retaining the same range as before by using hex coding instead of ascii.
- The time and opcode fields are also send in hex presentation.
  - The time field range is extended from 2^{16} to 2^{32} seconds. Alternatively, time can be outlined in TDMA-frames.
  - The opcode field is carrying 32-bit values. 16-bit opcodes aren't supported by FRAME 2.9.0 and higher.
- A field denouncing the original receiver is added, mentioning redirected protocol primitives. Only Protocol primitives will include this field (and the 4-byte opcode).
- All messages may be encapsulated in an STX at the beginning and a LF at the end
- The names of sender and receiver must not be longer than 4 characters.

2.2 G23 Test Interface Layer 2

Layer 2 is performed by another driver. The functional interface of this driver has to be the same in structure as described in [C_8415.026].

2.3 G23 Test Interface Layer 1

Layer 1 is performed by another driver, which is called by the Layer 2 driver . The functional interface of this driver has to be the same in structure as described in [C_8415.026].

3 Interface description of the TIF driver

3.1 Data Types

For types not defined here see [C_8415.026]

3.1.1 T_DRV_FUNC – Driver Functions

Definition:

```
typedef struct
{
  void (*drv_Exit)();
  USHORT (*drv_Read)();
  USHORT (*drv_Write)();
  USHORT (*drv_Look)();
USHORT (*drv_Clear)();
  USHORT (*drv_Flush)();
  USHORT (*drv_SetSignal)();
  USHORT (*drv_ResetSignal)();
  USHORT (*drv_SetConfig)();
  USHORT (*drv_GetConfig)();
  void (*drv_Callback)();
} T_DRV_FUNC
```

Description: The structure of the type T_DRV_FUNC contains the addresses of the driver entry functions.

3.1.2 T_DRV_EXPORT – Driver Properties

Definition:

```
typedef struct
{
  char *          Name
  USHORT          Flags
  T_DRV_FUNC      DrvFunc
} T_DRV_EXPORT
```

Description:

This data type defines the properties exported by the driver.

Name	Name of the driver
Flags	Bit (0): Callback function is called during ISR(1)/not called during ISR(0)
DrvFunc	functions to access the driver

3.1.3 T_DRV_SIGNAL – Driver Signal

Definition:

```
typedef struct
{
    USHORT SignalType
```

```
        USHORT DataLength
        void *     UserData
        USHORT DrvHandle
} T_DRV_SIGNAL
```

Description:

This type defines the signal information data used to identify a signal. This data type is used to define and to report a signal. A signal is defined by a process by calling the driver function *drv*_SetSignal(). An event is signaled by a driver by calling the signal call-back function (see 0, 3.3, 3.4.1).

3.1.4 T_DRV_CB_FUNC – Driver Callback Function

Definition:

```
typedef void (*T_DRV_CB_FUNC) (T_DRV_SIGNAL * Signal) ;
```

Description:

This type defines a call-back function used to signal driver events, e.g. driver is ready to accept data. The driver calls the signal call-back function when a specific event occurs and the driver has been instructed to signal the event to a specific process.

A process can set or reset event signaling by calling one of the driver functions TIF_SetSignal(), TIF_ResetSignal(). Event signaling can only be performed when a signal call-back function has been installed at driver initialization.

For more information about the T_DRV_SIGNAL data type, refer to 3.1.3.

3.2 Constants

3.2.1 Return Codes

Name	Description
DRV_OK	Return value indicating the function completed successfully
DRV_NOTCONFIGURED	Driver is not configured
DRV_INITFAILURE	Driver initialization failed
DRV_INVALID_PARAMS	One or more parameters are out of range or invalid
DRV_ERROR	Other error

3.2.2 Signal Codes

Name	Description
DRV_SIGTYPE_READ	Used to specify read event signaling
DRV_SIGTYPE_CONNECT	Used to specify connection established event signaling
DRV_SIGTYPE_DISCONNECT	Used to specify connection release event signaling

3.3 Signals

Signals are used to asynchronously inform the process using services about selected events. A signal call-back function is passed to the driver at the time of initialization (refer to 3.4.1). When no call-back is defined, event signaling cannot be performed. A signal can be set using the function TIF_SetSignal(), which can be found in Chapter 3.4.5. Event signaling can be disabled by calling the function TIF_ResetSignal().

The signaling is actually performed by calling the callback function, which has the following prototype:

```
void callback( T_DRV_SIGNAL *Signal );
```

The TIF driver uses only the *DrvHandle* and *SignalType* of the Signal, *UserData* and *DataLength* are nor used. The following table shows the values for *SignalType* as used by TIF:

SignalType	**meaning**
DRV_SIGTYPE_READ	The driver has received data which can be read by the TIF_Read() function.
DRV_SIGTYPE_CONNECT	TIF detected connection to the test system. Data may be sent.
DRV_SIGTYPE_DISCONNECT	EMI detected that the connection to the test system is lost.

3.4 Functions

Name	Description
TIF_Init	Initialization of TIF
TIF_Exit	Termination of TIFTIF_Read Read data from the driver
TIF_Write	Send data to the remote end
TIF_SetSignal	Enable a signal type to indicate an event
TIF_ResetSignal	Disable a signal type to indicate an event
TIF_SetConfig	Set driver configuration
TIF_Callback	Callback entry for layer 1 driver

The GDI standard functions TIF_Look, TIF_Flush, TIF_Clear and TIF_GetConfig are not implemented.

3.4.1 TIF_Init – Driver Initialization

Definition:

```
USHORT TIF_Init
(
    USHORT              DrvHandle,
    T_DRV_CB_FUNC       CallbackFunc,
    T_DRV_EXPORT **     DrvInfo
) ;
```

Parameters:

Name	Description
DrvHandle	This is the number, which identifies the driver in the system. The driver has to store it. It is needed, when the callback function is called.
CallbackFunc	The address of the callback function. The events at which to call back are configured using TIF_SetSignal(). If this address is equal to NULL the driver will not try to call back.
DrvInfo	Via this parameter the driver returns the addresses of it's functions. Some function addresses may be set to NULL, which means that these functions are not implemented and cannot be called.

Return values:

Name	Description
DRV_OK	Initialization successful
DRV_INITFAILURE	Initialization failed

Description

This function initializes the driver's internal data and returns the addresses of it's entry functions. TIF_Init() returns DRV_OK in the case of a successful completion.

In case of an initialization failure, which means that the driver cannot be used, the function returns DRV_INITFAILURE.

All signals have to be disabled during initialization and may be enabled later.

3.4.2 TIF_Exit – Driver Finalization

Definition:

```
void TIF_Exit
(
    void
) ;
```

Parameters:

Name	Description
-	

Return values:

Name	Description
-	-

Description

The function is called when the driver functionality is no longer required. The data exchange terminates immediately regardless of any outstanding data to be sent.

3.4.3 TIF_Read - Read Data from the Driver

Definition:

```
USHORT TIF_Read
(
        void *          Buffer,
        USHORT *        Length
) ;
```

Parameters:

Name	Description
Buffer	This parameter points to the buffer wherein the data is to be copied
Length	On call: number of characters to read. If the function returns DRV_OK, it contains the number of characters read. If the function returns DRV_INPROCESS, it contains 0.

Return values:

Name	Description
DRV_OK	Function successful
DRV_INPROCESS	The driver is currently reading data. The data is incomplete.
DRV_NOTCONFIGURED	The driver is not yet configured
DRV_NOCONNECT	Connection not available

Description

This function is used to read one system or protocol primitive from the driver. TIF_Read() copies the address of an allocated buffer that was filled with received data according to the structure of T_TST_RCV_DATA (refer to 0) to the address where the parameter buffer points to.

The parameter *Length contains the size of the buffer in characters.

If the driver is not configured, the function returns DRV_ NOTCONFIGURED.

If no connection exists the driver returns DRV_NOCONNECT.

NOTE: When calling the function with a buffer size of 0, the function will return DRV_OK. The size of the buffer needed to store the available data is stored in the parameter *Length. In this case, Buffer can be set to NULL.

3.4.4 TIF_Write – Write Data to the Driver

Definition:

```
USHORT TIF_Write
(
        void *          Buffer,
        USHORT *        Length
) ;
```

Parameters:

Name	Description
Buffer	This parameter points to the buffer that is passed to the driver for further processing
Length	On call: number of characters to write. If the function returns DRV_OK, it contains the number of characters written. This number is always equal to the requested number. In all other cases it contains 0.

Return values:

Name	Description
DRV_OK	Function successful, data frame written
DRV_BUFFER_FULL	Not enough space
DRV_NOTCONFIGURED	The driver is not yet configured
DRV_NOCONNECT	Connection not available

Description

This function is used to write one message to the driver. The parameter *Length contains the number of characters to write.

The header of the type T_SYST_HEADER is extracted and replaced with the message header according to 2.1. The data itself is then copied behind the message header.

The TIF_Write() function has to call vsi_d_write() – see prototype below - to call the layer 2 *drv*_Write() function.

```
USHORT vsi_d_write ( USHORT Caller, USHORT DrvHandle, void *Buffer, USHORT *Length
)
```

The parameter Caller is the handle of the TIF driver passed to TIF_Init(). The parameter DrvHandle has to be set to 0 to indicate to the frame that it must call the corresponding lower layer *drv*_Write() function. Buffer and Length are the parameters passed to TIF_Write(). For detailed information about vsi_d_write(), refer to [C_8415.033].

In the case of a successful completion, the function returns DRV_OK.

In all other cases the data was not written and the call to TIF_Write() should be repeated later.

NOTE: When calling the function with a buffer size of 0, the function will return the number of characters that can be written in the parameter *Length. In this case, Buffer can be set to NULL, no data will be written.

3.4.5 TIF_SetSignal – Setup a Signal

Definition:

```
USHORT TIF_SetSignal
(
        USHORT                  SignalType
) ;
```

Parameters:

Name	Description
SignalType	Signal type to be set

Return values:

Name	Description
DRV_OK	Function completed successfully
DRV_INVALID_PARAMS	Signal type not supported by TIF
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available

Description

This function is used to define a single signal that the driver should indicated via the callback function.

To remove a signal, call the function TIF_ResetSignal().

If no signal call-back function has been defined at the time of initialization, the driver returns DRV_SIGFCT_NOTAVAILABLE.

3.4.6 TIF_ResetSignal – Remove a Signal

Definition:

```
USHORT TIF_ResetSignal
(
        USHORT                  SignalType
) ;
```

Parameters:

Name	Description
SignalType	Signal type to be reset

Return values:

Name	Description
DRV_OK	Function completed successfully
DRV_INVALID_PARAMS	Signal type not supported by TIF
DRV_SIGFCT_NOTAVAILABLE	Event signaling functionality is not available

Description

This function is used to remove a previously set signal.

If no signal call-back function has been defined at the time of initialization, the driver returns DRV_SIGFCT_NOTAVAILABLE.

3.4.7 TIF_SetConfig – Set the Driver Configuration

Definition:

```
USHORT TIF_SetConfig
(
    char *          Config
) ;
```

Parameters:

Name	Description
Config	Pointer to driver configuration string

Return values:

Name	Description
DRV_OK	Function successfully completed
DRV_INVALID_PARAMS	Error in configuration parameters

Description

This function is used to configure the driver. In the current version the TIF driver need not to be configured.

3.4.8 TIF_Callback – Callback Function

Definition:

```
void TIF_Callback
(
    T_DRV_SIGNAL *      Signal
) ;
```

Parameters:

Name	Description
Signal	Pointer to Signal sent by layer 1 driver

Return values:

Name	Description

-

Description

This function is called by the frame if the lower layer driver calls the vsi_d_callback() function.

If the signal type in the passed signal is DRV_SIGTYPE_READ, the TIF_Callback() function has to call vsi_d_read() – see prototype below - to read the layer 2 data via its *drv*_Read() function.

```
USHORT vsi_d_read ( USHORT Caller, USHORT DrvHandle, void *Buffer, USHORT *Length )
```

The parameter Caller is the handle of the TIF driver passed to TIF_Init(). The parameter DrvHandle has to be set to 0 to indicate to the frame that it must call the corresponding lower layer *drv*_Read() function. For detailed information about vsi_d_read() refer to [C_8415.033].

The TIF driver has to allocate a memory partition to store data read from the lower layer driver. To get the size of the buffer to be allocated, it has to call vsi_d_read() with the parameter Length set to 0. Vsi_d_read() returns the number of bytes to be read. An appropriate memory partition of the returned size plus the size of T_SYST_PRIM plus one for zero termination can now be allocated by calling vsi_c_new().

The pointer returned by vsi_c_new has to be cast to a structure of the type T_SYST_PRIM (refer to 0), data read from the lower layer driver has to be analyzed and the parameters MsgId, Fmt, Size, Sender, Receiver and the time information have to be formatted in the allocated buffer at the position of the header of the type T_SYST_HEADER. The received data beginning from the address of 'Data' (refer to 2.1) has to be copied to address of 'Data' in T_TEST_MSG, refer to **Error! Reference source not found.**.

After this the callback function, which address was passed to TIF_Init(), has to be called with a pointer to a signal of the type T_DRV_SIGNAL (refer to 3.1.3). The signal type has to be set to DRV_SIGTYPE_READ and the DrvHandle is the one passed to TIF_Init().

Signal types different from DRV_SIGTYPE_READ like DRV_SIGTYPE_CONNECT and DRV_SIGTYPE_DISCONNECT also have to be forwarded to the test interface process - if the corresponding signal type is enabled - by calling the callback function, which address was passed to TIF_Init().

Appendices

A. Acronyms

DS-WCDMA Direct Sequence/Spread Wideband Code Division Multiple Access

B. Glossary

International Mobile Telecommunication 2000 (IMT-2000/ITU-2000) Formerly referred to as FPLMTS (Future Public Land-Mobile Telephone System), this is the ITU's specification/family of standards for 3G. This initiative provides a global infrastructure through both satellite and terrestrial systems, for fixed and mobile phone users. The family of standards is a framework comprising a mix/blend of systems providing global roaming. <URL: http://www.imt-2000.org/>