Low Level Design

PROTOCOL STACK INTERFACE PSI

Document Number:	8462.725.04.001
Version:	1.3
Status:	Update
Approval Authority:	
Creation Date:	2004-Feb-26
Last changed:	2015-Mar-08 by Ricarda Marzillier
File Name:	lld_psi.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
2004-Feb-26	Ricarda Marzillier		0.1	Draft	1
2004-Mar-16	Ricarda Marzillier		0.2	Draft	2
2004-May-07	Ricarda Marzillier		0.3	Update	3
2004-Aug-17	Ricarda Marzillier		0.4	Update	4
2004-Sep-07	Ricarda Marzillier		1.0	Update	5
2004-Nov-04	Ricarda Marzillier		1.1	Update	6
2004-Dez-16	Ricarda Marzillier		1.2	Update	7
2005-Apr-28	Ricarda Marzillier		1.3	Update	8

Notes:

1. Initial version
2. Add some functions and comments after pre-review
3. Update some functions and add state machine pictures
4. Update some functions and descriptions after integration procedure
5. Update RX Service Machine
6. Update Service Machines, add new simulation functions
7. Update description psi_check_control_info()
8. Update after enhancement for packet data and flushing

Table of Contents

List of Figures and Tables

List of References

[7010.801] References and Vocabulary, Texas Instruments

[1400.201.02.004] DIO – Data I/O Driver Interface

[8448.201.01.012] DTI Data Transmission Interface Library, Detailed Specification

[8462.720.01.001] HLD for Protocol Stack Interface PSI

1 Introduction

This document is a Low Level Design (LLD) Description of the new entity PSI – Protocol Stack Entity on basis of [4]. It describes the interface of PSI to the TI Protocol Stack (PS) and to the Driver Interface (DIO). The entity PSI should replace the current entities for serial driver (UART) and for packet driver (PKTIO). Furthermore PSI is used for access to applications (using BAT – Binary AT Inteface) to the PS. The integration of new hardware drivers is much more easier because PSI supports the DIO interface, version 4. It is assumed that the reader has a basic understanding about the Driver Interface DIO – Data I/O, version 4, like described in [2]. This driver can be used for communication between several drivers and PS and it includes support of serial, packet and multiplexer devices. Up to now a previous version of DIO (version 3) is only used by the entity PKTIO. Also it is assumed that the reader has a basic knowledge about the current Data Transmission Interface (DTI) described in [3]. It serves to realize the data flow between PSI and a different protocol stack entity.

The document describes the behaviour of PSI via message sequence charts and the needed functions. The introduction of a new service access point (SAP) for PSI is needed and replaces the SAPs of PKTIO and UART. The PSI SAP is usable for BAT too.

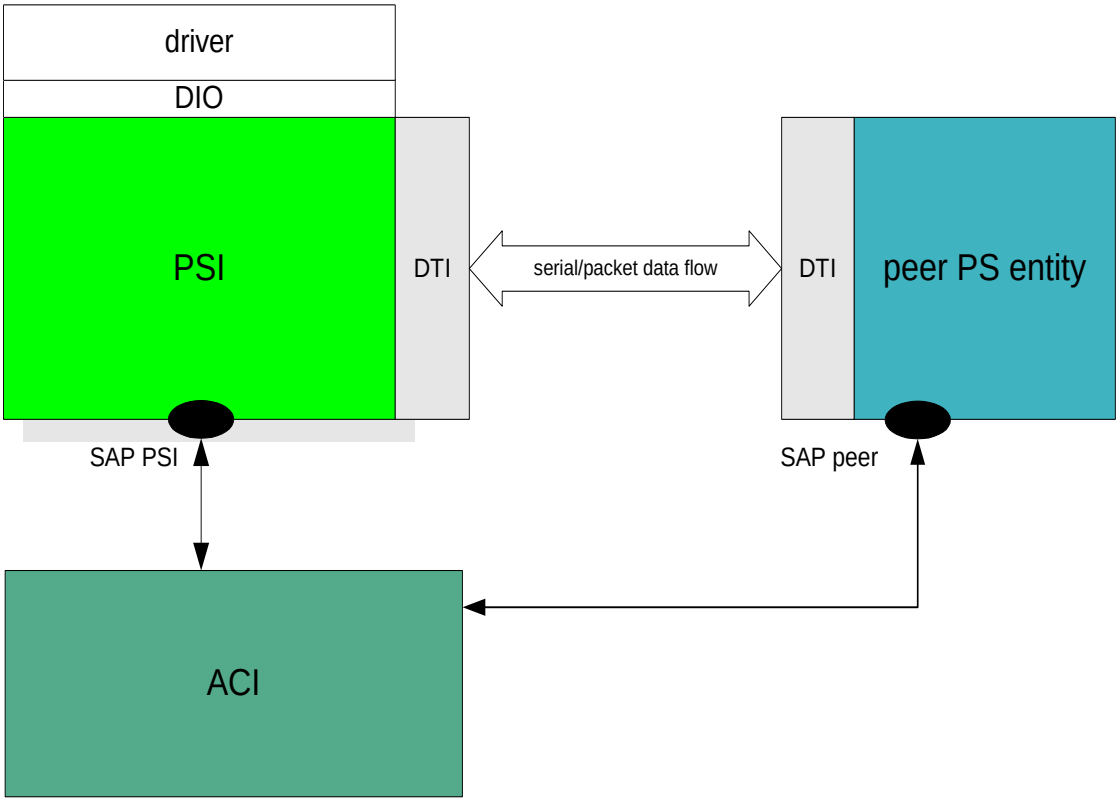

Figure 1: Schema of PSI integration in TI protocol stack

2 Message Sequence Charts

2.1 Initialization of DIO and Registration of DIO user PSI

The entity PSI starts the initialization of DIO via dio_init. The layer part of DIO interface initializes the hardware driver via the driver own init function dio_init_*drv* (). PSI calls the function dio_user_init for the user registration in DIO interface layer and provides its signal callback function. This function serves DIO for receiving of driver signaling. The driver is informed by DIO that PSI is now ready to receive signals from the driver. Afterwards the driver starts opening devices and returns the signal DRV_SIGTYPE_CONNECT.

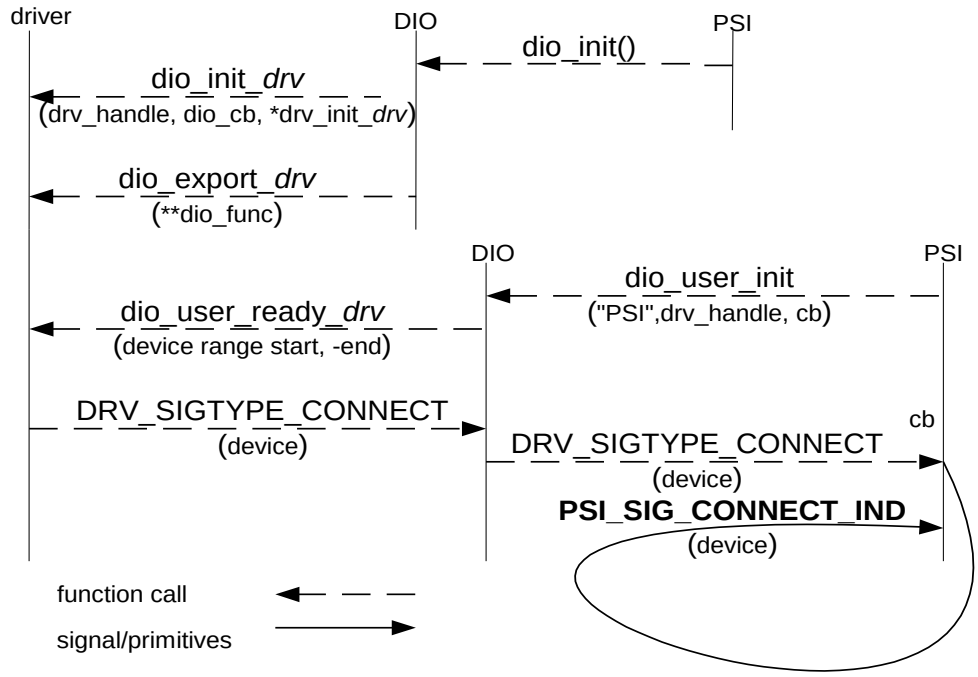

Figure 2: Initializing DIO interface layer and User Registration within DIO

2.2 Request of Driver Capabilities and Registration in ACI

PSI requests the specific driver capabilities calling the function dio_get_capabilities. DIO supports driver specific capability structures for serial, packet and multiplexer devices. ACI has access to the driver capabilities via the parameter “capPtr” of the primitive PSI_CONN_IND. The parameter “devId” contains information about the driver number (UART, USB, Bluetooth…), device type, and supported data type (serial, packet, multiplex). PSI provides information whether AT command and / or data are sent by the device in the parameter “data_mode” (in the final version “data_mode” refering to a specific device identifier is stored in FFS).
In the first PSI version PSI sets the USB driver configuration parameter “dcb” hardcoded. In the final version PSI uses driver configuration parameter stored in FFS.

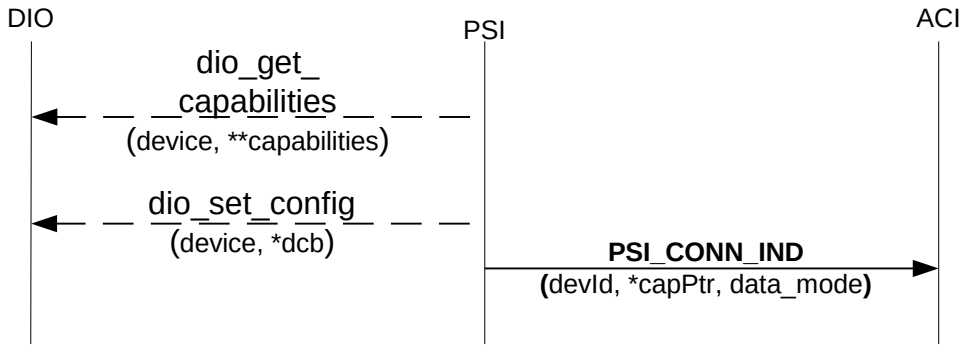

Figure 3: Requirement Driver Capabilities by PSI and Registration in ACI

2.3 Confirmation of Device Registration

If the device registration is successful, ACI sends the primitive PSI _CONN_RES to PSI.

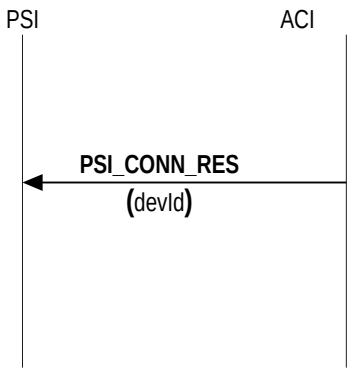

Figure 4: Confirmation of Registration in ACI

2.4 Rejecting of Device Registration

If the device registration is not successful, ACI sends the primitive PSI_CONN_REJ to the PSI. The PSI entity starts the closing of this device.

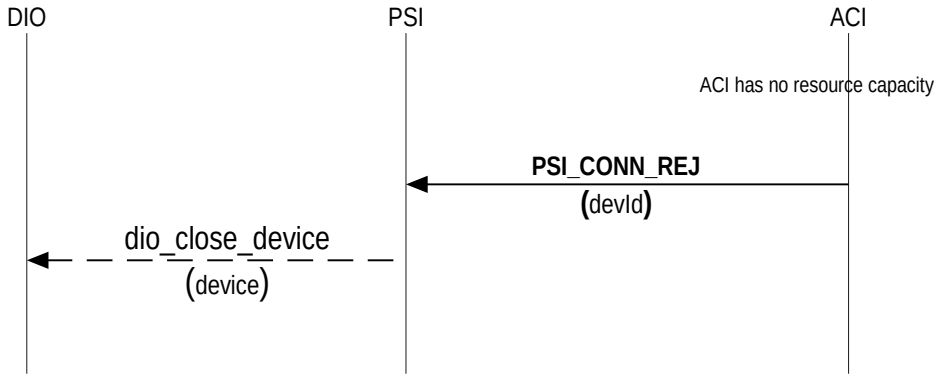

Figure 5: Rejecting of Registration in ACI

2.5 Close Device Driver initiated

The driver indicates PSI via signaling that it can't be used any longer. PSI calls the DIO functions dio_read and dio_get_tx_buffer. On this way PSI gets back the control of every send and read buffer provided to the driver. If a DTI connection must be closed PSI informs ACI via primitive PSI_DTI_CLOSE_IND. In every case ACI gets the disconnect reason via the primitive PSI_DISCONN_IND. At the end PSI calls dio_close_device closing the device.

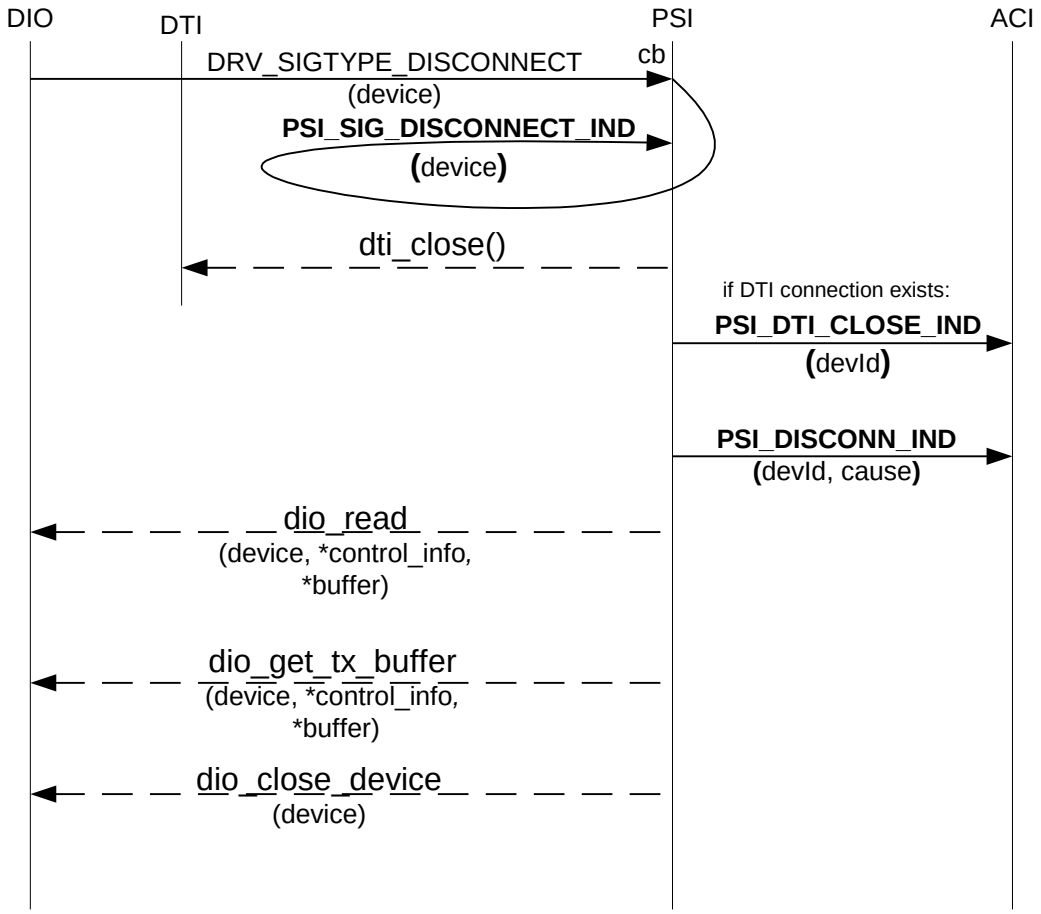

Figure 6: Closing Requirement by the driver

2.6 Close Device User initiated

If the user is not being able to use the device any longer ACI requests PSI via primitive PSI_DTI_CLOSE_REQ closing a existing DTI connection. PSI calls dti_close (). Therefore PSI confirms the request with the primitive PSI_DTI_CLOSE_CNF. After that ACI requests closing of the device via the primitive PSI_CLOSE_REQ. The entity PSI calls the DIO functions dio_read and dio_get_tx_buffer to get back the control of send and read buffer provided to the driver. In the last step PSI stops the driver connection calling dio_close_device and sends the confirmation primitive PSI_CLOSE_CNF to ACI.

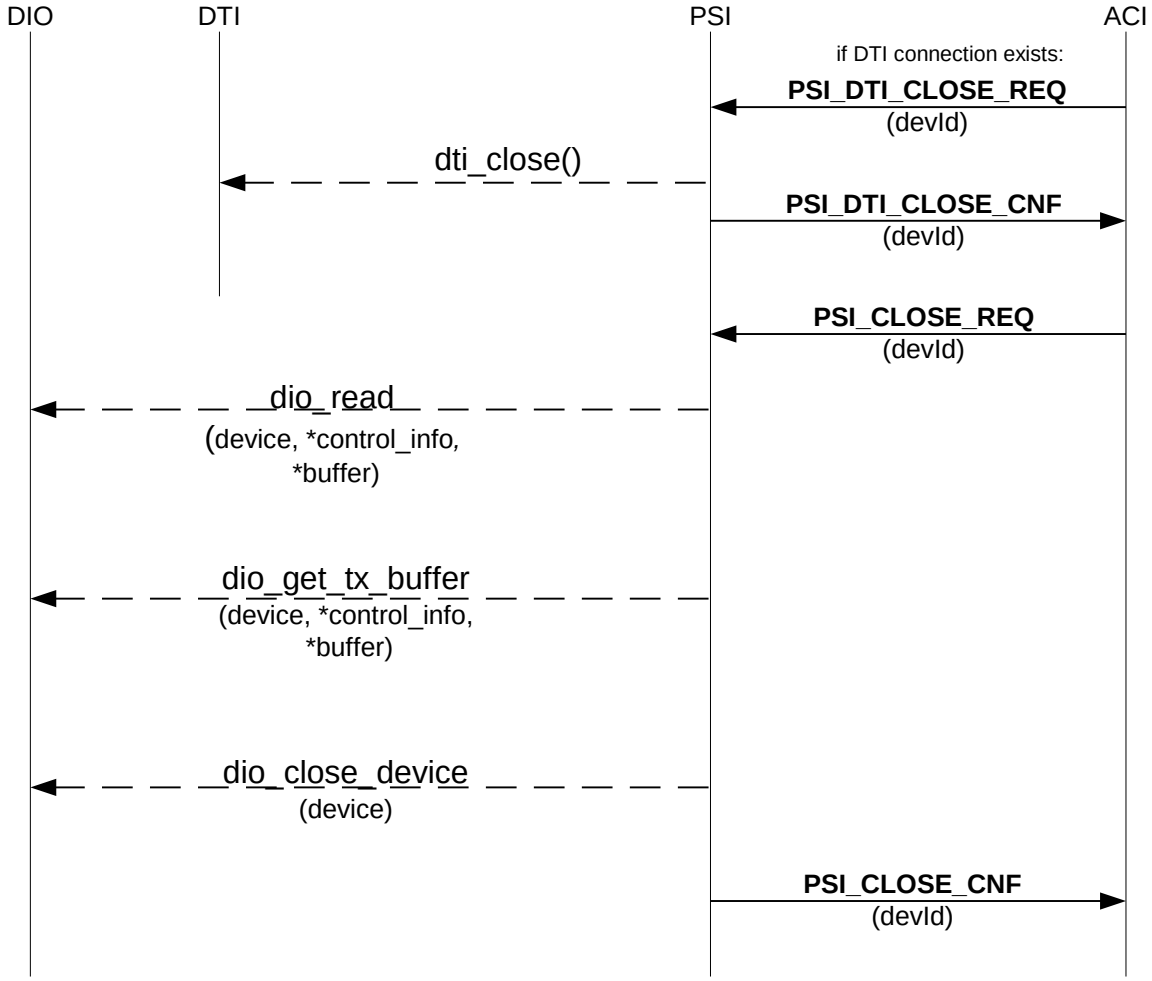

Figure 7: Closing Requirement by the user

2.7 Start DTI connection

For data transfer the user configures the data flow and starts a data call. ACI has to initiate the DTI connection between the PSI and the peer protocol stack entity. PSI opens the port to DTI after receiving the primitive PSI_DTI_OPEN_REQ. If DTI confirms successfully the entity sends the primitive PSI_DTI_OPEN_CNF to ACI.

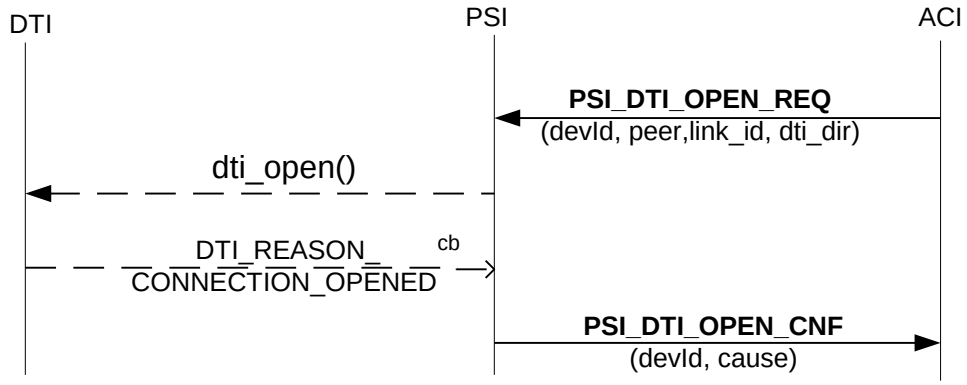

Figure 8: Start DTI connection by ACI

2.8 Stop DTI connection

If the user terminates the data call ACI has to initiate the DTI disconnection between the PSI and the peer entity. PSI closes the port to DTI after receiving the primitive PSI_DTI_CLOSE_REQ. Afterwards the PSI entity confirms the ACI request with the primitive PSI_DTI_CLOSE_CNF sent to ACI.

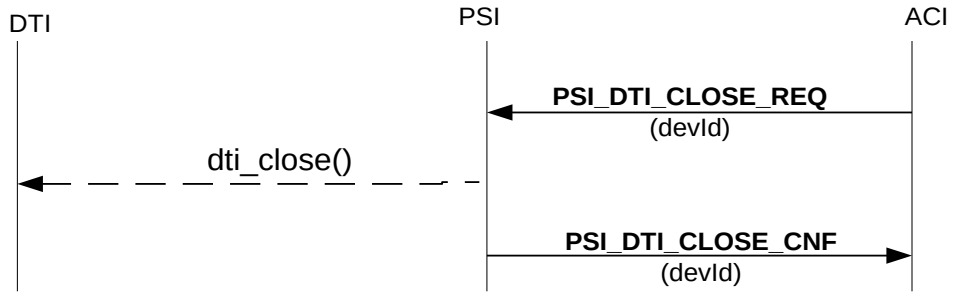

Figure 9: Stop DTI connection by ACI

2.9 Modify of Device Configuration

ACI can change device configuration parameters of serial devices at any time. The changed parameters are sent in the primitive PSI_SETCONF_REQ. After flushing of the hardware write buffer (described in 1.16) PSI sends these configuration parameters in driver specified format to the device via the DIO interface. If the device configuration is successful changed, PSI sends the confirmation primitive PSI_SETCONF_CNF to ACI.

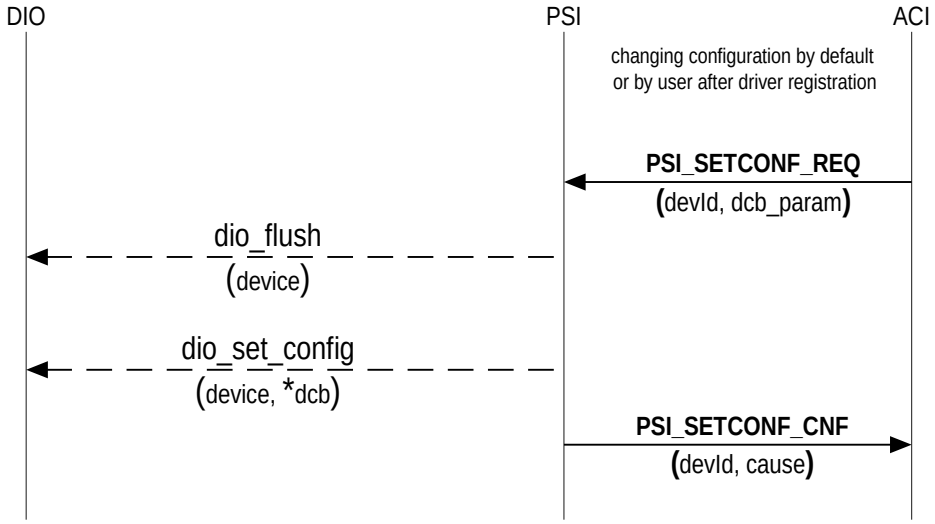

Figure 10: Modification of device configuration for serial devices by ACI

2.10 Query of Device Configuration

The DIO Interface provides the possibility to retrieve the device configuration calling the DIO function dio_get_config .

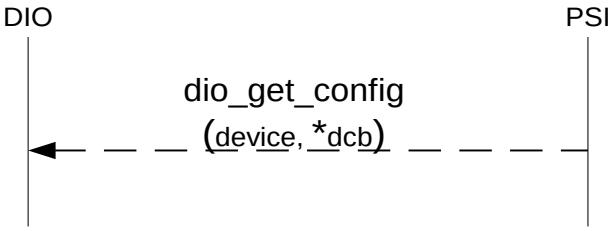

Figure 11: Query of device configuration by PSI

2.11 Line State Configuration initiated by user

ACI can set/reset certain line states like RING/DCD for serial connections. After the needed flushing procedure the entity sends the requested states in the parameter line_state via primitive PSI_LINE_STAT_REQ. PSI inform the driver calling the DIO function dio_write. ACI receives the confirmation via PSI_LINE_STAT_CNF.

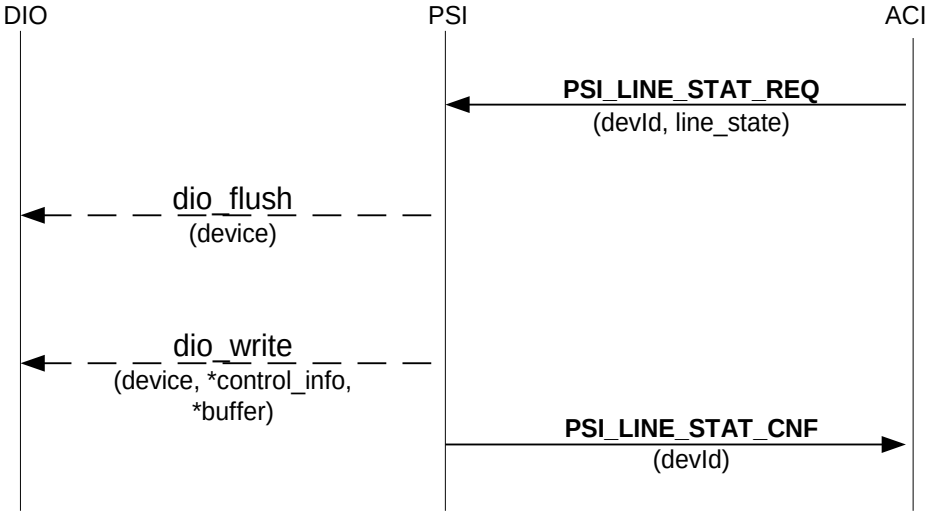

Figure 12: Set Line States for serial devices by ACI

2.12 Line State Configuration initiated by device

The serial device can detect escape sequences and DTR line drops. For this reason the function dio_read returns driver control information which includes the specific indication. ACI receives the line states parameter in primitive PSI_LINE_STAT_IND.

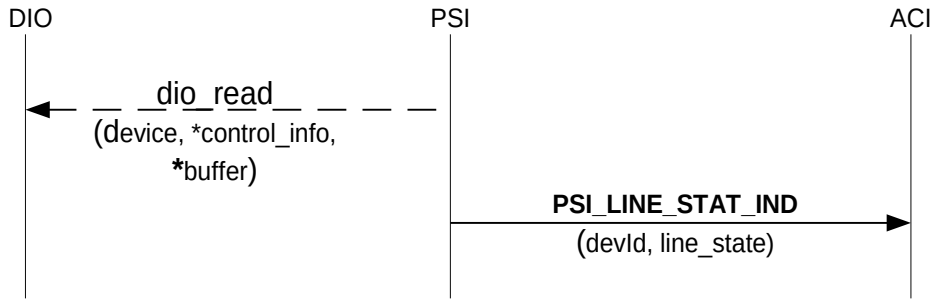

Figure 13: Indication of Line States by serial device

2.13 Read Data

The entity PSI provides the read buffers which are segmented to the driver. For this reason the DIO function dio_set_rx_buffer is used. To avoid data transmission problems PSI can provide several buffer. In the first implementation two read buffer with one segment are used. The driver indicates the start of data reading by calling the PSI driver callback function with the parameter DRV_SIGTYPE_READ. PSI gets back the filled buffer via the DIO function dio_read . The buffer control returns to the entity PSI.

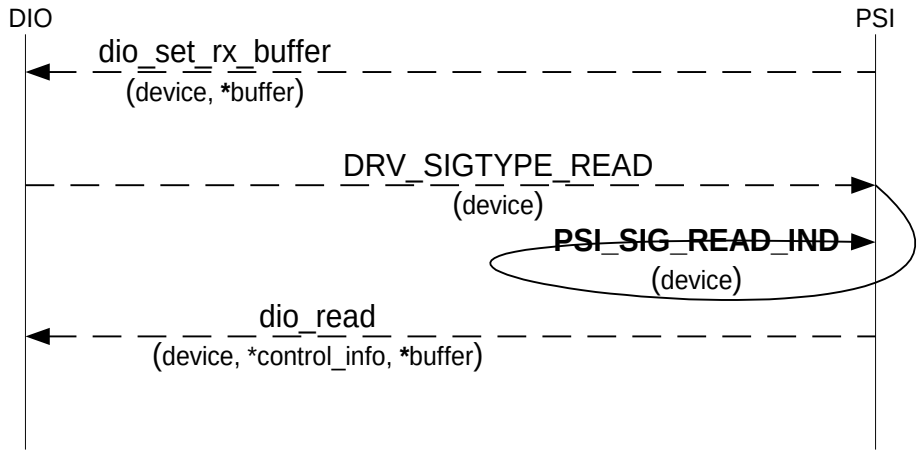

Figure 14: PSI receives data by the device (read data)

2.14 Write Data

The entity PSI can provide the segmented write buffer to the driver. The PS can send changed line states for serial connections. In certain cases the driver must be flushed before the received data are sent to the driver. If the driver has written the write buffer completely it calls the PSI driver callback function with the parameter DRV_SIGTYPE_WRITE.
PSI gets back the write buffer control via the DIO function dio_get_tx_buffer. The number of segments equals the number of descriptor elements sent by DTI.

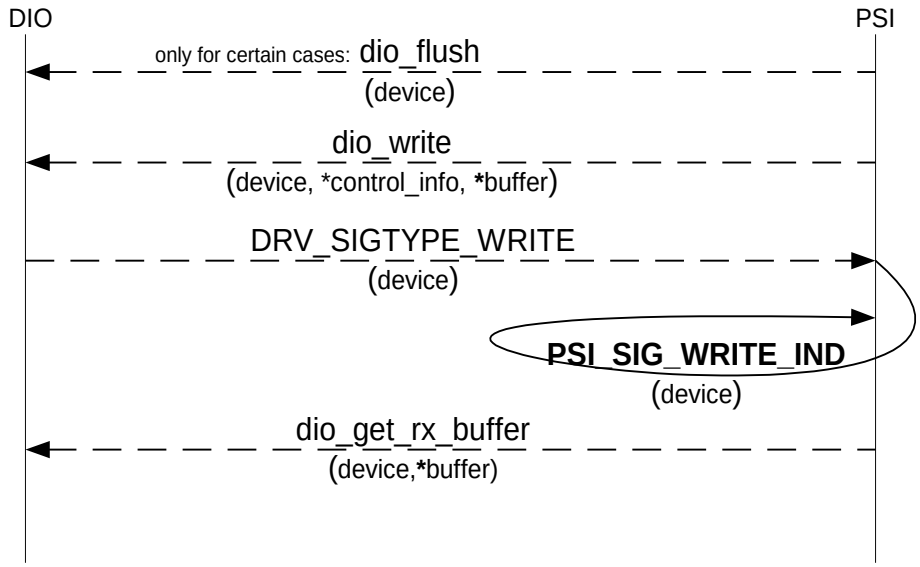

Figure 15: PSI sends data to the device (write data)

2.15 Clear hardware write buffer

If PSI wants to clear the hardware write buffer the DIO function dio_clear can be used. The driver confirms the successful procedure via PSI signal callback function with the parameter DRV_SIGTYPE_CLEAR. This functionality is not used currently.

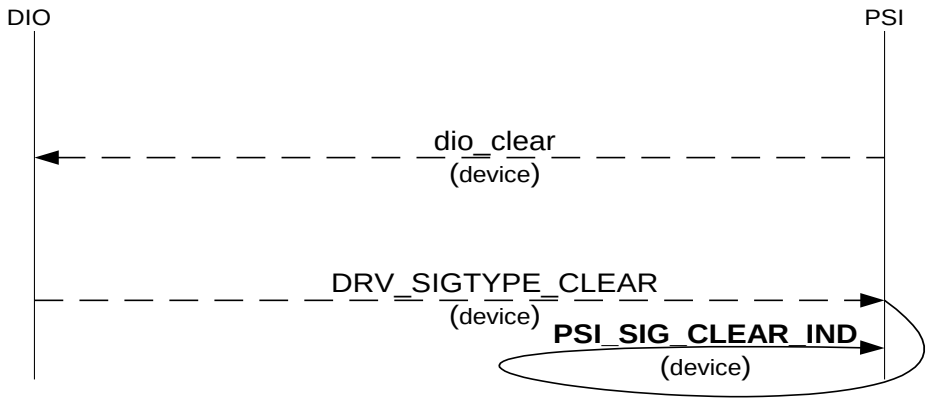

Figure 16: Clearing hardware write buffer

2.16 Flush hardware write buffer

If the user changes device configuration parameter or line states are changed PSI need to know whether the last sent data is written successfully (hardware write buffer is empty). Due to this fact the DIO function dio_flush is used. In the case that the buffer is not available immediately the function returns DRV_INPROCESS. After the flush procedure is finished completely the driver sends the signal event DRV_SIGTYPE_FLUSH.

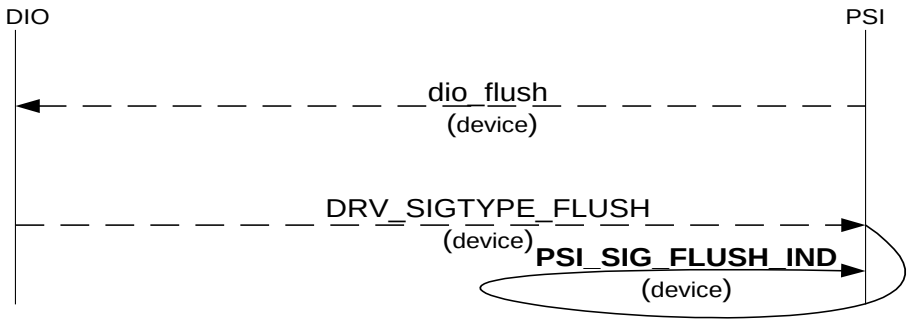

Figure 17: Flushing hardware write buffer

3 Overview State Machines of PSI services

3.1 PSI Kernel Service

Controling of commonly internal entity state.

Figure 18: State machine of kernel service in PSI

3.2 PSI DTI Kernel Service

Controling of internal state regarding DTI connection handling

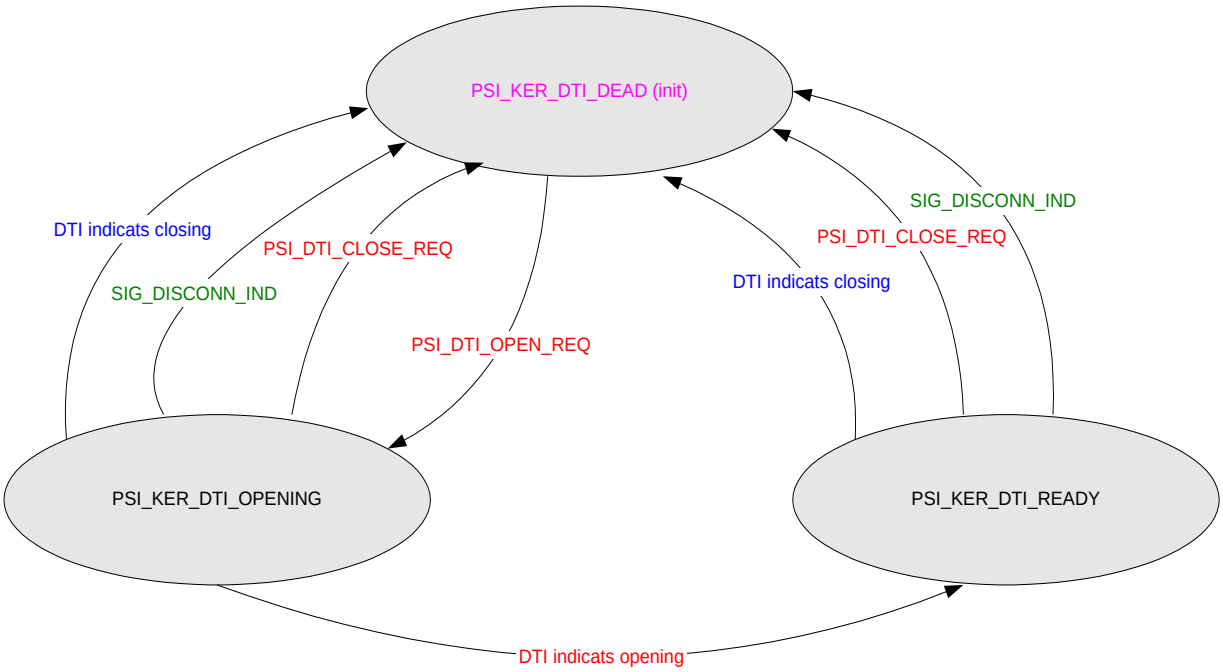

Figure 19: State machine of dti service in PSI

3.3 PSI TX Service

Controling of internal data transmission state to the driver

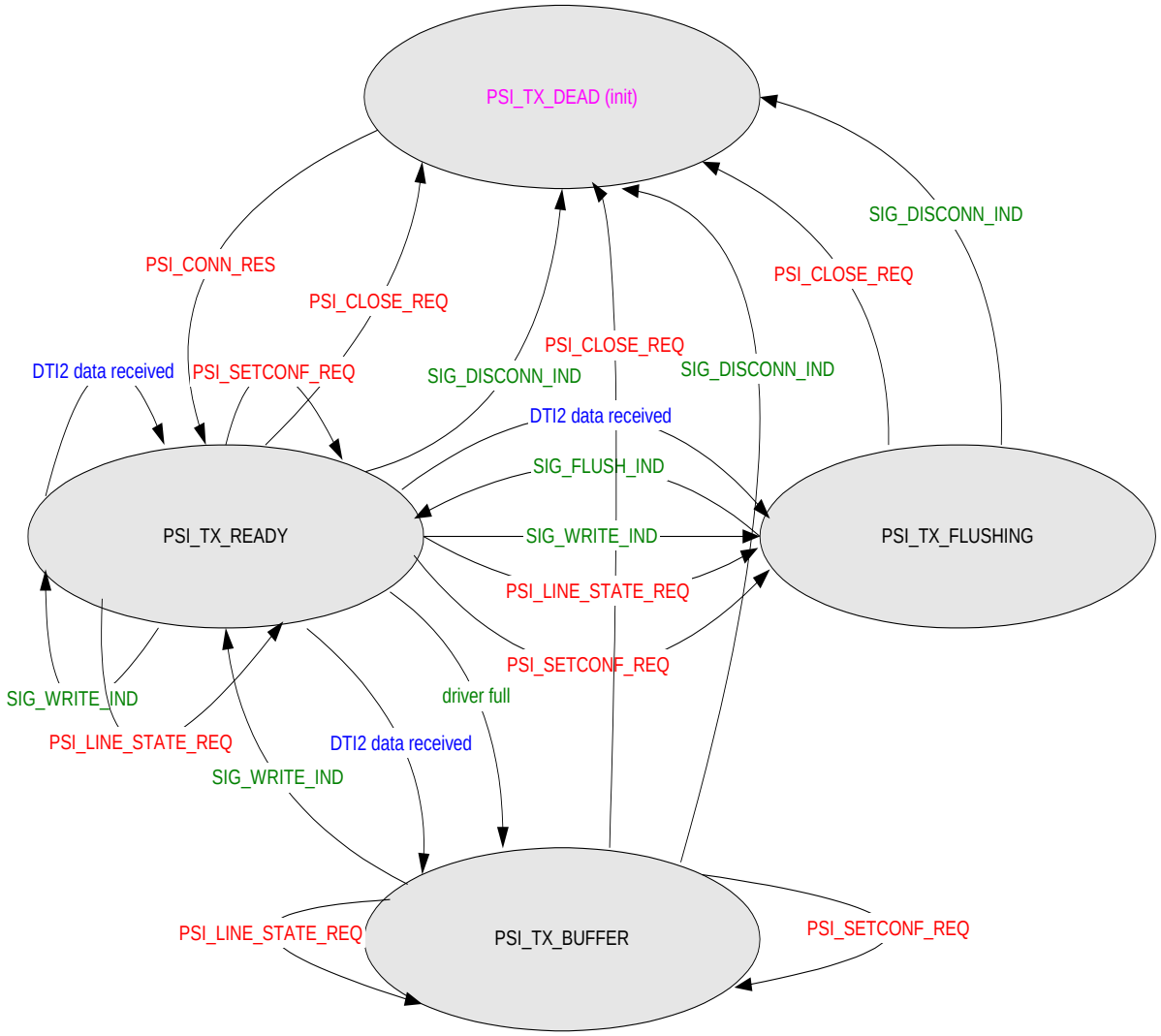

Figure 20: State machine of tx service in PSI

3.4 PSI RX Service

Controling of internal data transmission state from the driver

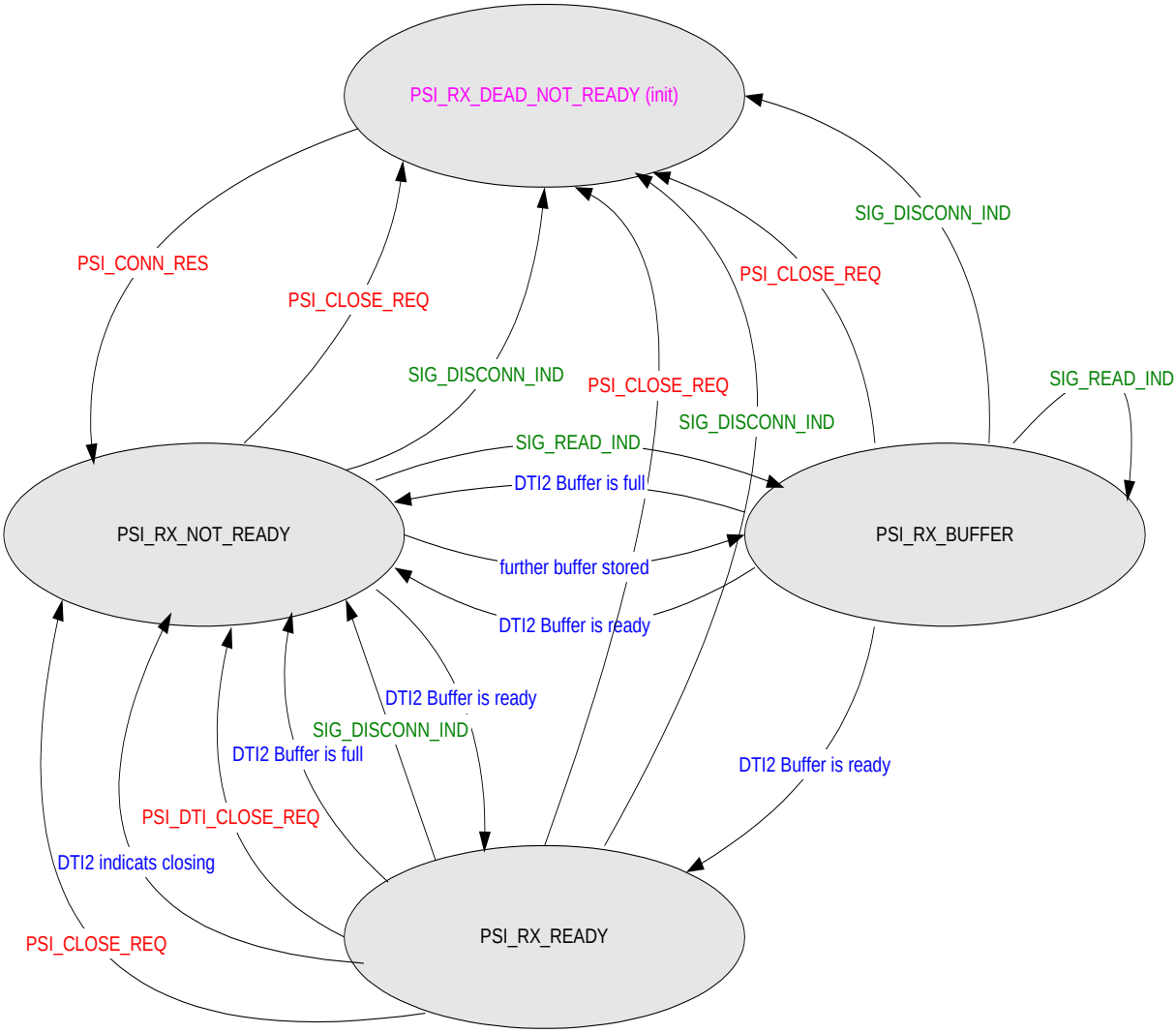

Figure 21: State machine of rx service in PSI

3.5 PSI DTX Service

Controling of internal data transmission state from peer entity via DTI

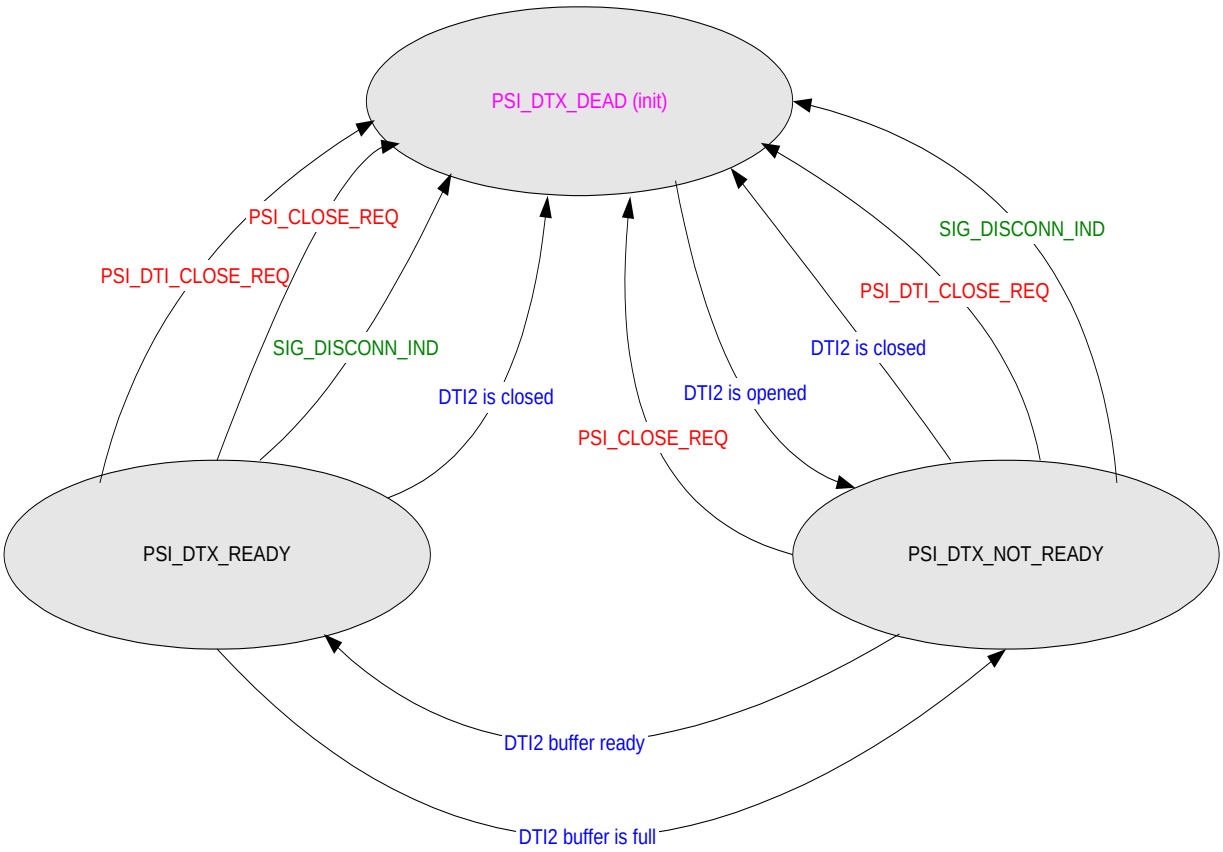

Figure 22: State machine of dtx service in PSI

3.6 PSI DRX Service

Controling of internal data transmission state to the peer entity via DTI

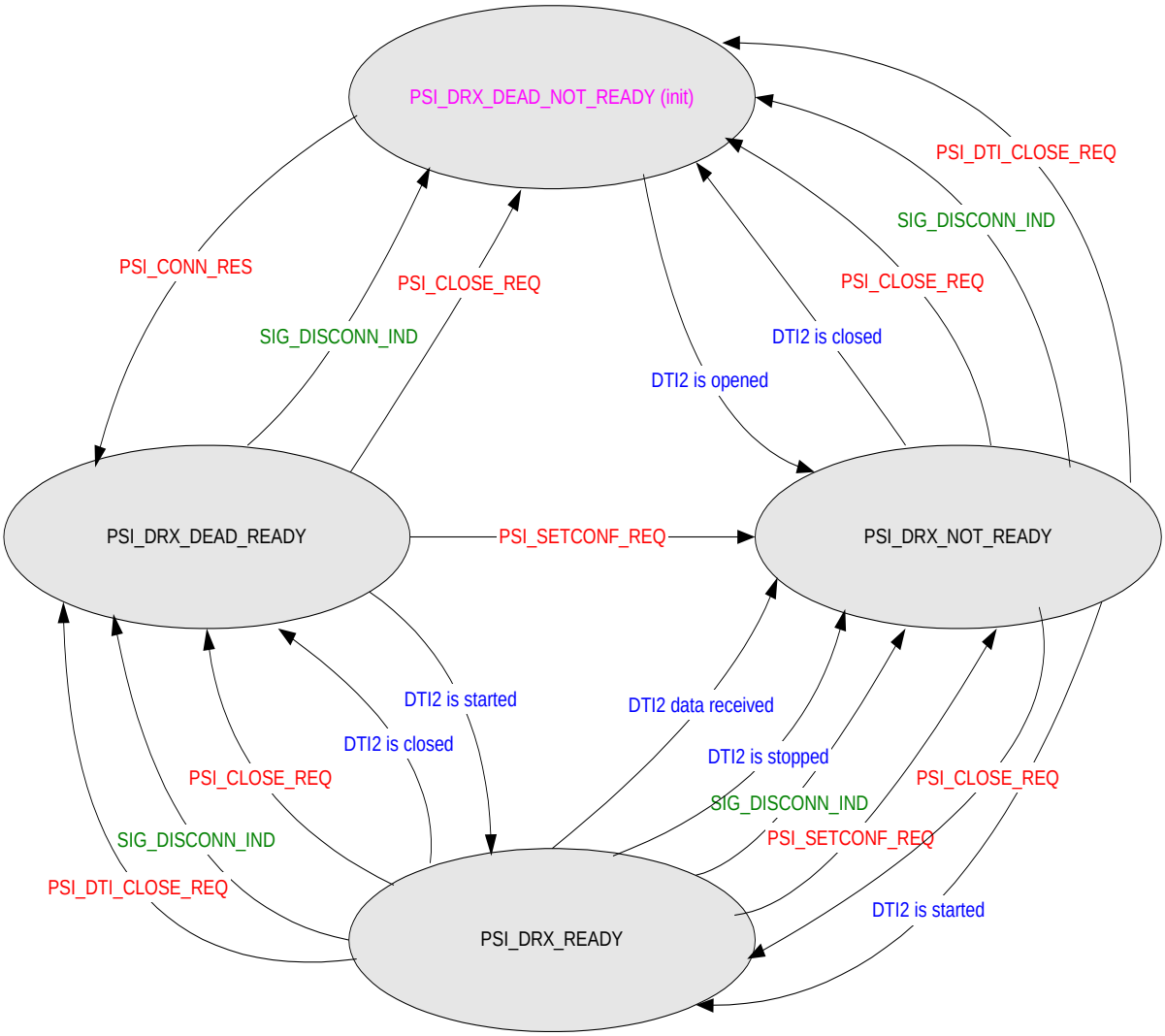

Figure 23: State machine of drx service in PSI

4 New Functions and Data Structures

4.1 New Functions

4.1.1 Module psi_diosim.c

This module is valid only for the windows simulation. It contains all DIO functions for testing.

4.1.1.1 dio_clear

Prototype:

GLOBAL U16 dio_clear (U32 device)

Parameters:

device	device identifier

Return:

DRV_OK	function successful
DRV_INPROCESS	the driver is busy flushing the buffer
DRV_INVALID_PARAMS	unknown device
DRV_INTERNAL_ERROR	internal driver error
DRV_NOTCONFIGURED	DIO interface is not yet configured

Description:

This function simulates the DIO function dio_clear () used to clear the hardware write buffer. Currently it is not used.

4.1.1.2 dio_close_device

Prototype:

GLOBAL U16 dio_close_device (U32 device)

Parameters:

device	device identifier

Return:

DRV_OK	device successfully closed
DRV_INVALID_PARAMS	the device can not be closed yet
DRV_INTERNAL_ERROR	internal driver error

Description:

This function simulates the DIO function dio_close_device () used to close the device. It can not be closed if driver still controls PSI buffers.

4.1.1.3 dio_exit

Prototype:

GLOBAL void dio_exit (void)

Parameters:

Return:

Description:

This function simulates the DIO function dio_exit () used to terminate the DIO interface layer by protocol stack.

4.1.1.4 dio_flush

Prototype:

GLOBAL U16 dio_ flush (U32 device)

Parameters:

device	device identifier

Return:

DRV_OK	function successful
DRV_INPROCESS	the driver is busy flushing the buffer
DRV_INVALID_PARAMS	unknown device
DRV_INTERNAL_ERROR	internal driver error
DRV_NOTCONFIGURED	DIO interface is not yet configured

Description:

This function simulates the DIO function dio_ flush () used to flush the hardware write buffer. It is used if the driver configuration parameter should be changed.

4.1.1.5 dio_get_capabilities

Prototype:

GLOBAL U16 dio_get_capabilities (U32 device, T_DIO_CAP const** capabilities)

Parameters:

device	device identifier
capabilities	return : pointer to a static struc of constant capability values

Return:

DRV_OK	function successful
DRV_INVALID_PARAMS	unknown device
DRV_INTERNAL_ERROR	internal driver error

Description:

This function simulates the DIO function dio_get_capabilities () used to get the driver capabilities.

4.1.1.6 dio_get_config

Prototype:

GLOBAL U16 dio_get_config (U32 device, T_DIO_DCB* dcb)

Parameters:

device	device identifier
dcb	pointer to device control block containing configuration parameter filled by driver

Return:

DRV_OK	function successful
DRV_INVALID_PARAMS	unknown device or wrong dcb structure provided
DRV_INTERNAL_ERROR	internal driver error
DRV_NOTCONFIGURED	DIO interface is not yet configured

Description:

This function simulates the DIO function dio_get_config () used to retrieve the driver configuration parameter.

4.1.1.7 dio_get_tx_buffer

Prototype:

GLOBAL U16 dio_get_tx_buffer (U32 device, T_dio_buffer ** buffer)

Parameters:

device	device identifier
buffer	data buffer description

Return:

DRV_OK	function successful
DRV_INVALID_PARAMS	unknown device
DRV_INTERNAL_ERROR	internal driver error
DRV_NOTCONFIGURED	DIO interface is not yet configured

Description:

This function simulates the DIO function dio_get_tx_buffer () used to get back the control about the provided write buffer.

4.1.1.8 dio_init

Prototype:

GLOBAL U16 dio_init (void)

Parameters:

Return:

DRV_OK	Initializating successful
DRV_INITIALIZED	Interface already initialized or already in use
DRV_INITFAILURE	Initializing failed

Description:

This function simulates the DIO function dio_init () used to initialize the DIO interface layer and all DIO drivers.

4.1.1.9 dio_read

Prototype:

GLOBAL U16 dio_ read (U32 device, T_DIO_CTRL* control_info, T_dio_buffer ** buffer)

Parameters:

device	device identifier
control_info	driver delivers control information
buffer	return : data buffer description

Return:

DRV_OK	function successful
DRV_INVALID_PARAMS	unknown device
DRV_INTERNAL_ERROR	internal driver error
DRV_NOTCONFIGURED	DIO interface is not yet configured

Description:

This function simulates the DIO function dio_ read () used to get the read buffer filled by the driver. The control information of driver is delivered too.

4.1.1.10 dio_set_config

Prototype:

GLOBAL U16 dio_set_config (U32 device, T_DIO_DCB* dcb)

Parameters:

device	device identifier
dcb	pointer to device control block containing configuration parameter filled by PSI

Return:

DRV_OK	function successful
DRV_INVALID_PARAMS	configuration parameter invalid, out of range or not supported
DRV_INTERNAL_ERROR	internal driver error

Description:

This function simulates the DIO function dio_set_config () used to configure a device.

4.1.1.11 dio_set_rx_buffer

Prototype:

GLOBAL U16 dio_set_rx_buffer (U32 device, T_dio_buffer * buffer)

Parameters:

device	device identifier
buffer	data buffer description

Return:

DRV_OK	function successful
DRV_BUFFER_FULL	buffer queue full
DRV_INVALID_PARAMS	unknown device or data buffer is not big enough
DRV_INTERNAL_ERROR	internal driver error
DRV_NOTCONFIGURED	DIO interface is not yet configured

Description:

This function simulates the DIO function dio_set_rx_buffer () used to provide read buffer to the driver.

4.1.1.12 dio_user_exit

Prototype:

GLOBAL U16 dio_user_exit (char* user_name)

Parameters:

user_name	pointer of zero terminated string

Return:

DRV_OK	user operation successfully terminated
DRV_INVALID_PARAMS	user operation can not be terminated yet.
DRV_INTERNAL_ERROR	internal driver error

Description:

This function simulates the DIO function dio_user_exit () used to terminate the user connection with DIO. All devices must be closed before it is allowed to call this function.

4.1.1.13 dio_user_init

Prototype:

GLOBAL U16 dio_user_init (char* user_name, U16 drv_handle, T_DRV_CB_FUNC signal_callback)

Parameters:

user_name	pointer of zero terminated string
drv_handle	callback function handle
signal_callback	callback function for driver signals

Return:

DRV_OK	callback set successful
DRV_INVALID_PARAMS	unknown user_name
DRV_INTERNAL_ERROR	internal driver error
DRV_NOTCONFIGURED	DIO interface is not yet initialized

Description:

This function simulates the DIO function dio_user_init () used to inform about the DIO user and its driver signal callback function.

4.1.1.14 dio_write

Prototype:

GLOBAL U16 dio_ write (U32 device, T_DIO_CTRL* control_info, T_dio_buffer ** buffer)

Parameters:

device	device identifier
control_info	PSI delivers control information
buffer	data buffer description

Return:

DRV_OK	function successful
DRV_BUFFER_FULL	buffer queue full
DRV_INVALID_PARAMS	unknown device or data buffer is to big
DRV_INTERNAL_ERROR	internal driver error
DRV_NOTCONFIGURED	DIO interface is not yet configured

Description:

This function simulates the DIO function dio_write () used to provide filled write buffer to the driver.

4.1.1.15 psi_diosim_get_conf

Prototype:

LOCAL const T_PRIM_HEADER* psi_diosim_get_conf (ULONG awaited_opc)

Parameters:

awaited_opc	opc of the awaited simulation primitive

Return:

T_PRIM_HEADER*	received primitive

Description:

This function is needed for the simulation of the driver functional interface. It waits for the primitive sent by TAP.

4.1.1.16 psi_diosim_sign_ind

Prototype:

GLOBAL void psi_diosim_sign_ind (const T_PSI_DIOSIM_SIGN_IND *psi_diosim_sign_ind)

Parameters:

psi_diosim_sign_ind	driver signal

Return:

Description:

This function simulates the callback function handling the DIO signals.

4.1.2 Module psi_drxf.c

4.1.2.1 psi_drx_dti_reason_data_received

Prototype:

GLOBAL void psi_drx_dti_reason_data_received(T_DTI2_DATA_IND *dti2_data_ind)

Parameters:

dti2_data_ind	Pointer to received data (sent by peer entity via DTI)

Return:

Description:

This function receives the data of peer entity. If the service DRX equals PSI_DRX_READY the DTI channel is stopped calling the DTI function dti_stop ().PSI sends this data to the driver via psi_drx_tx_data and the DRX service is set to PSI_DRX_NOT_READY. The primitve dti2_data_ind is freed but except the descriptor list containing the user data.

4.1.2.2 psi_drx_init

Prototype:

GLOBAL void psi_drx_init (void)

Parameters:

Return:

Description:

This function initializes the PSI service PSI_SERVICE_DRX with value PSI_DRX_DEAD_NOT_READY. This service controls the receiving of data from DTI.

4.1.3 Module psi_drxs.c

4.1.3.1 psi_ker_drx_close

Prototype:

GLOBAL void psi_ker_drx_close (void)

Parameters:

Return:

Description:

This function handles the internal service PSI_SERVICE_DRX for the following cases : the user requests closing of the DTI connection or DTI disconnects the connection or the driver sends a disconnecting event.

4.1.3.2 psi_ker_drx_open

Prototype:

GLOBAL void psi_ker_drx_open (void)

Parameters:

Return:

Description:

After receiving of the DTI event « DTI_REASON_CONNECTION_OPENED » and for case PSI_DRX_DEAD_READY this function calls the DTI function dti_start () starting the flow control procedure. That means the flow control primitive DTI_GETDATA_REQ is sent to the peer entity indicating that PSI is able to receive data.

4.1.3.3 psi_tx_drx_close

Prototype:

GLOBAL void psi_tx_drx_close (void)

Parameters:

Return:

Description:

This function handles the internal service PSI_SERVICE_DRX if user requests closing of driver connection resp. driver signals disconnection.

4.1.3.4 psi_tx_drx_ready

Prototype:

GLOBAL void psi_tx_drx_ready (void)

Parameters:

Return:

Description:

After driver indication confirming the provided write buffer is written, this function calls the DTI function dti_start () to allow getting further data from the peer entity.

4.1.4 Module psi_dtxf.c

4.1.4.1 psi_dtx_dti_reason_tx_buffer_full

Prototype:

GLOBAL void psi_dtx_dti_reason_tx_buffer_full (void)

Parameters:

Return:

Description:

This function handles the DTI callback event « DTI_REASON_TX_BUFFER_FULL ». The internal service PSI_SERVICE_DTX is not ready to send data to DTI until the read buffer are again empty (event « DTI_REASON_TX_BUFFER_READY»). The following state is PSI_DTX_NOT_READY.

4.1.4.2 psi_dtx_dti_reason_tx_buffer_ready

Prototype:

GLOBAL void psi_dtx_dti_reason_tx_buffer_ready (void)

Parameters:

Return:

Description:

This function handles the DTI callback event « DTI_REASON_TX_BUFFER_READY ». The internal service PSI_SERVICE_DTX is ready to send data to DTI that means the state changes from PSI_DTX_NOT_READY to PSI_DTX_READY. The function psi_dtx_rx_ready is called for checking of buffered data.

4.1.4.3 psi_dtx_init

Prototype:

GLOBAL void psi_dtx_init (void)

Parameters:

Return:

Description:

This function initializes the PSI service PSI_SERVICE_DTX with value PSI_DTX_DEAD. This PSI service controls the data sending to DTI.

4.1.5 Module psi_dtxs.c

4.1.5.1 psi_ker_dtx_close

Prototype:

GLOBAL void psi_ker_dtx_close (void)

Parameters:

Return:

Description:

This function handles the internal service PSI_SERVICE_DTX for all of release processes. The state is set to PSI_DTX_DEAD. The function psi_dtx_rx_close is called for processing internal state PSI_SERVICE_RX.

4.1.5.2 psi_ker_dtx_open

Prototype:

GLOBAL void psi_ker_dtx_open (void)

Parameters:

Return:

Description:

This function handles the internal service PSI_SERVICE_DTX after the successful opening of a DTI connection. The state is set from PSI_DTX_DEAD to PSI_DTX_NOT_READY and waits for the DTI callback event « DTI_REASON_TX_BUFFER_READY ».

4.1.5.3 psi_rx_dtx_data

Prototype:

GLOBAL void psi_rx_dtx_data(T_desc2* buffer, U16 len, T_DIO_CTRL* control_info)

Parameters:

buffer	Pointer to first element of descriptor element chain containing received data
len	number of user data bytes
control_info	Pointer to a structure containing line states

Return:

Description:

If the internal service PSI_SERVICE_DTX state equals PSI_DTX_READY this function sends the data sent by the driver to DTI calling dti_send_data (). The functions deals serial data filling the received control_info (serial line states) in the data primitive.

4.1.5.4 psi_rx_dtx_data_pkt

Prototype:

GLOBAL void psi_rx_dtx_data_pkt(T_desc2* buffer, U16 len, U8 pid)

Parameters:

buffer	Pointer to first element of descriptor element chain containing received data
len	number of user data bytes
pid	protocol identifier

Return:

Description:

If the internal service PSI_SERVICE_DTX state equals PSI_DTX_READY this function sends the data sent by the driver to DTI calling dti_send_data (). The functions deals packet data, the protocol identifier is filled in the data primitive.

4.1.6 Module psi_kerf.c

4.1.6.1 check_flow_control

Prototype:

GLOBAL BOOL check_flow_control(U32* dio_flow_control, U32 sap_flow_control)

Parameters:

dio_flow_control	previous set configuration parameter flow control
sap_flow_control	configuration parameter set by user flow control

Return:

Description:

This function checks the requested flow control parameter against the driver capabilities.

4.1.6.2 check_char_frame

Prototype:

GLOBAL BOOL check_char_frame(U32* dio_char_frame, U32 sap_char_frame)

Parameters:

dio_char_frame	previous set configuration parameter character frame
sap_char_frame	configuration parameter set by user character frame

Return:

Description:

This function checks the requested character frame parameter against the driver capabilities.

4.1.6.3 check_baudrate

Prototype:

GLOBAL BOOL check_baudrate(U32* dio_baudrate,U32 sap_baudrate)

Parameters:

dio_baudrate	previous set configuration parameter baud rate
sap_baudrate	configuration parameter set by user baud rate

Return:

Description:

This function checks the requested baud rate parameter against the driver capabilities of automatically detected and fixed baud rates.

4.1.6.4 psi_ker_assign_cause

Prototype:

GLOBAL void psi_ker_assign_cause (U16* cause, U16 result)

Parameters:

cause value of PSI SAP

result value of DIO SAP

Return:

Description:

This function maps a result value from a driver call to a cause value for PSI SAP.

4.1.6.5 psi_ker_assign_ctrl

Prototype:

GLOBAL void psi_ker_assign_ctrl (T_DIO_CTRL_LINES *ptr_dio_ctrl, U16 line_state)

Parameters:

ptr_dio_ctrl Pointer to DIO_CTRL structure in PSI data base

line_state line states sent by ACI

Return:

Description:

This function stores the line state parameter sent by ACI in global PSI data base. Moreover the function checks whether the flushing procedure have to be start.

4.1.6.6 psi_ker_assign_dcb

Prototype:

GLOBAL BOOL psi_ker_assign_dcb (T_DIO_DCB* dcb, U32 dev_type,
const T_DIO_DCB* sap_dcb)

Parameters:

dcb Pointer to DIO_DCB structure in PSI data base

dev_type kind of driver (seriell, parallel, mux)

sap_dcb Pointer to configuration parameter structure sent by ACI

Return:

Description:

This function copies the device configuration parameter provided by the SAP into a structure defined by the DIO driver and store parameter in global PSI data base.

4.1.6.7 psi_ker_assign_dcb_sim

Prototype:

GLOBAL BOOL psi_ker_assign_dcb_sim (T_DIO_DCB* dcb, U32 dev_type,
T_DIO_DCB_UN* sap_dcb)

Parameters:

dcb Pointer to DIO_DCB structure in PSI data base

dev_type kind of driver (seriell, parallel, mux)

sap_dcb Pointer to configuration parameter structure sent by ACI

Return:

Description:

This function copies the device configuration parameter provided by the SAP into a structure defined by the DIO driver and store parameter in global PSI data base. It is used only for windows testcases. The union structure T_DIO_DCB_UN contains three structures: for serial, packet and mux parameter.

4.1.6.8 psi_ker_dti_reason_connection_closed

Prototype:

GLOBAL void psi_ker_dti_reason_connection_closed (void)

Parameters:

Return:

Description:

This function processes the DTI callback event « DTI_REASON_CONNECTION_CLOSED ». The primitive PSI_DTI_CONN_IND is sent to ACI depending on the internal kernel state of PSI. In case PSI_KER_DTI_READY the states of services PSI_SERVICE_DRX, PSI_SERVICE_DTX and PSI_SERVICE_RX have to be deactivated (functions psi_ker_drx_close and psi_ker_dtx_close.

4.1.6.9 psi_ker_dti_reason_connection_opened

Prototype:

GLOBAL void psi_ker_dti_reason_connection_opened (void)

Parameters:

Return:

Description:

This function processes the DTI callback event « DTI_REASON_CONNECTION_OPENED ». The primitive PSI_DTI_CONN_CNF is sent to ACI indicating a successful built DTI connection. Furthermore the internal states PSI_SERVICE_DRX and PSI_SERVICE_DTX are activated.

4.1.6.10 psi_ker_init

Prototype:

GLOBAL void psi_ker_init (void)

Parameters:

Return:

Description:

This function initializes the PSI kernel service PSI_SERVICE_KER. This PSI service handles the driver registration / deregistration signals and the primitives sent by ACI.

4.1.6.11 psi_ker_new_instance

Prototype:

GLOBAL U16 psi_ker_new_instance (U32 device)

Parameters:

device	device identifier

Return:

PSI_NEW_OK	new PSI instance is registered
PSI_NEW_FULL	new PSI instance is not possible
PSI_NEW_USED	PSI instance is already used

Description:

This function creates a new PSI instance.

4.1.6.12 psi_ker_instance_switch

Prototype:

GLOBAL U16 psi_ker_instance_switch (U8 instance)

Parameters:

instance	PSI instance identifier

Return:

PSI_INST_OK	PSI instance is registered
PSI_INST_NOT_FOUND	PSI instance is not known

Description:

This function checks the PSI instance given by DTI.

4.1.6.13 psi_ker_search_basic_data_by_device

Prototype:

GLOBAL U16 psi_ker_search_basic_data _by_device (U32 device)

Parameters:

device	device identifier

Return:

PSI_DEVICE_FOUND	PSI instance is registered
PSI_DEVICE_NOT_FOUND	PSI instance does not exist

Description:

This function checks for the given device within the PSI basic data.

4.1.6.14 psi_ker_set_init_conf

Prototype:

GLOBAL BOOL psi_ker_set_init_conf (void)

Parameters:

Return:

Description:

This function sets the initial driver configuration parameter. In the final version these values are stored in FFS.

4.1.7 Module psi_kerp.c

4.1.7.1 psi_ker_close_req

Prototype:

GLOBAL void psi_ker_close_req (T_ PSI_CLOSE_REQ* psi_close_req)

Parameters:

psi_close_req	data of primitive PSI_CLOSE_REQ

Return:

Description:

This function handles the primitive PSI_CLOSE_REQ sent by ACI to cancel the device connection. Depending on internal kernel PSI state PSI has to call back read and write buffer which are still under driver control, free such resp. pending buffer resp. stored descriptor element chains. Then the function starts the device closing procedure(dio_close_device). The internal states of PSI_SERVICE_RX, PSI_SERVICE_DRX, PSI_SERVICE_DTX and PSI_SERVICE_TX are deactivated (functions psi_ker_rx_close and psi_ker_tx_close). The primitive PSI_CLOSE_CNF is sent to ACI.

4.1.7.2 psi_ker_conn_rej

Prototype:

GLOBAL void psi_ker_conn_rej (T_PSI_CONN_REJ* psi_conn_rej)

Parameters:

psi_conn_rej	data of primitive PSI_CONN_REJ

Return:

Description:

This function handles the primitive PSI_CONN_REJ sent by ACI if the device is not successful registered there.
The device closing starts by calling of the DIO function dio_close_device ().

4.1.7.3 psi_ker_conn_res

Prototype:

GLOBAL void psi_ker_conn_res (T_PSI_CONN_RES* psi_conn_res)

Parameters:

psi_conn_res data of primitive PSI_CONN_RES

Return:

Description:

This function handles the primitive PSI_CONN_RES sent by ACI if the device is successful registered there. Calling the function psi_ker_tx_open and psi_ker_rx_open the internal states of PSI_SERVICE_TX, PSI_SERVICE_DRX and PSI_SERVICE_RX are activated. The read buffer are created and sent to the driver. For packet devices the needed segment sizes is adapted via psi_rx_reconf_pkt.

4.1.7.4 psi_ker_dti_close_req

Prototype:

GLOBAL void psi_ker_dti_close_req (T_ PSI_DTI_CLOSE_REQ* psi_dti_close_req)

Parameters:

psi_dti_close_req data of primitive _ PSI_DTI_CLOSE_REQ

Return:

Description:

This function handles the primitive PSI_DTI_CLOSE_REQ sent by ACI to cancel a DTI connection. The function dti_close () is called and the primitive PSI_DTI_CLOSE_CNF is sent to ACI. Depending of the internal kernel PSI state the internal services PSI_SERVICE_DRX, PSI_SERVICE_DTX and PSI_SERVICE_RX are deactivated.

4.1.7.5 psi_ker_dti_open_req

Prototype:

GLOBAL void psi_ker_dti_open_req (T_ PSI_DTI_OPEN_REQ* psi_dti_open _req)

Parameters:

psi_dti_open_req data of primitive PSI_DTI_OPEN_REQ

Return:

Description:

This function handles the primitive PSI_DTI_OPEN_REQ sent by ACI to start a new DTI connection. The function dti_open () containing the peer entity, PSI instance and link_id is called. The DTI link_id is stored in the PSI data base.

4.1.7.6 psi_ker_free_buffers

Prototype:

GLOBAL void psi_ker_free_buffers (T_dio_buffer *buffers)

Parameters:

Buffers pointer to the PSI buffer which should be free

Return:

Description:

This function frees the specific buffer with segment array.

4.1.7.7 psi_ker_free_all_buffers

Prototype:

GLOBAL void psi_ker_free_all_buffers (void)

Parameters:

Return:

Description:

This function calls dio_read and dio_get_tx_buffer to get back the buffer control. Afterwards all buffers, stored descriptor element chains and stored driver configuration parameter structure are deallocated.

4.1.7.8 psi_ker_line_state_req

Prototype:

GLOBAL void psi_ker_line_state_req (T_ PSI_LINE_STATE_REQ * psi_line_state_req)

Parameters:

psi_line_state_req data of primitive PSI_LINE_STATE_REQ

Return:

Description:

This function handles the primitive PSI_LINE_STATE_REQ sent by ACI to configure line states of serial and multiplexer devices. The subfunction psi_ker_assign_ctrl maps the line states sent by ACI in the DIO driver interface structure and checks the values. If flushing is needed and possible the procedure is started via psi_ker_tx_flush. In case that flushing is not needed PSI calls the DIO function dio_write to send the changed control information to the driver and the confirmation primitive PSI_LINE_STATE_CNF is sent to ACI.

4.1.7.9 psi_ker_setconf_req

Prototype:

GLOBAL void psi_ker_setconf_req (T_ PSI_SETCONF_REQ* psi_setconf_req)

Parameters:

psi_setconf_req data of primitive PSI_SETCONF_REQ

Return:

Description:

This function handles the primitive PSI_SETCONF_REQ sent by ACI to configure a device. The configuration parameter are checked in psi_ker_assign_dcb. In case of invalid parameters PSI sends the primitive PSI_SETCONF_CNF with cause PSICS_INVALID_PARAMS to ACI.
PSI stops the DTI connection via dti_stop(). Afterwards PS checks the status of sent write buffer. If all of write buffer are under PCI control it starts the flushing procedure to get informed whether the data in the hardware write buffer is completely sent. Otherwise the flush flag is set.

4.1.7.10 psi_ker_setconf_req_test

Prototype:

GLOBAL void psi_ker_setconf_req_test (T_ PSI_SETCONF_REQ_TEST* psi_setconf_req_test)

Parameters:

psi_setconf_req_test data of primitive PSI_SETCONF_REQ_TEST

Return:

Description:

This function – only used for windows testcases - handles the primitive PSI_SETCONF_REQ_TEST sent by ACI to configure a device. The configuration parameter are checked in psi_ker_assign_dcb .
In case of invalid parameters PSI sends the primitive PSI_SETCONF_CNF with cause PSICS_INVALID_PARAMS to ACI.
PSI stops the DTI connection via dti_stop(). Afterwards PS checks the status of sent write buffer. If all of write buffer are under PCI control it starts the flushing procedure to get informed whether the data in the hardware write buffer is completely sent. Otherwise the flush flag is set.

4.1.7.11 psi_ker_sig_connect_ind

Prototype:

GLOBAL void psi_ker_sig_connect_ind (U32 device)

Parameters:

device device identifier

Return:

Description:

This function handles the PSI own signal PSI_SIG_CONNECT_IND based on driver signal DRV_SIGTYPE_CONNECT. With this signal the driver indicates the existence of a new device, and is ready to send resp. receive data. First a new PSI instance is created by psi_ker_new_instance. PSI interrogates the driver capabilities and sends the primitive PSI_CONNECT_IND to ACI. The primitive contains also the parameter data_mode read by FFS. It describes the kind of handled driver data like AT commands, data or both. Afterwards PSI initializes the configuration parameter psi_ker_set_init_conf and sends this data to the driver via dio_set_config. The internal kernel PSI state is set to PSI_KER_CONNECTING.

4.1.7.12 psi_ker_sig_disconnect_ind

Prototype:

GLOBAL void psi_ker_sig_disconnect_ind (U32 device)

Parameters:

device device identifier

Return:

Description:

This function handles the PSI own signal PSI_SIG_DISCONNECT_IND based on driver signal DRV_SIGTYPE_DISCONNECT. With this signal the driver indicates that the device cannot be used any longer. If it is necessary PSI closes the DTI connection and sends the primitive PSI_DTI_CLOSE_IND to ACI. Depending on internal kernel PSI state PSI has to call back read and write buffer which are still under driver control, free such resp. pending buffer resp. stored descriptor element chains. Then the function starts the device closing procedure dio_close_device. The internal states of PSI_SERVICE_RX, PSI_SERVICE_DRX, PSI_SERVICE_DTX and PSI_SERVICE_TX are deactivated (psi_dtx_rx_close, psi_ker_tx_close, psi_ker_drx_close, psi_ker_dtx_close). The primitive PSI_DISCONN_IND sent to ACI indicates that the device is not longer available.

4.1.8 Module psi_kers.c

4.1.8.1 psi_ker_tx_flushed

Prototype:

GLOBAL void psi_ker_tx_flushed (void)

Parameters:

Return:

Description:

After successful flush of the hardware write buffer this function gives the changed driver configuration parameter via dio_set_config to the serial device. In case that changed line states are received via the DTI data primitive the according data are written to the device too.
For the state PSI_KER_MODIFY existing only if the configuration parameter were sent in PSI_SETCONF_REQ the entity sends - according to the PSI SAP - the primitive PSI_SETCONF_CNF with the result code to ACI. If changed line states are received in PSI_LINE_STATE_REQ the confirmation primitive PSI_LINE_STATE_CNF is sent.

4.1.9 Module psi_pei.c

The following primitive tables are needed :

1. primitives of SAP PSI
2. DTI primitives of uplink and downlink

4.1.9.1 pei_config

Prototype:

LOCAL SHORT pei_config (char *inString)

Parameters:

inString	command string

Return:

PEI_OK	entity created successfuly
PEI_ERROR	entity could not be created

Description:

This function serves for dynamic configuration of this entity.

4.1.9.2 pei_create

Prototype:

GLOBAL SHORT pei_create (T_PEI_INFO **info)

Parameters:

info	entity configuration parameter

Return:

PEI_OK	entity created successfuly
PEI_ERROR	entity could not be created

Description:

This function creates the entity PSI.

The further functions from 4.1.9.3 to 4.1.9.14 handles the primitives of DTI. They convert the received primitives in function calls.

4.1.9.3 psi_dti_dti_connect_cnf

Prototype:

LOCAL const void psi_dti_dti_connect_cnf (T_DTI2_CONNECT_CNF *dti2_connect_cnf)

Parameters:

dti2_connect_cnf	parameter of primitive

Return:

Description:

This function handles the primitive DTI2_CONNECT_CNF.

4.1.9.4 psi_dti_dti_connect_ind

Prototype:

LOCAL const void psi _dti_dti_connect_ind (T_DTI2_CONNECT_IND *dti2_connect_ind)

Parameters:

dti2_connect_ind	parameter of primitive

Return:

Description:

This function handles the primitive DTI2_CONNECT_IND.

4.1.9.5 psi_dti_dti_connect_req

Prototype:

LOCAL const void psi _dti_dti_connect_req (T_DTI2_CONNECT_REQ *dti2_connect_req)

Parameters:

dti2_connect_req	parameter of primitive

Return:

Description:

This function handles the primitive DTI2_CONNECT_REQ.

4.1.9.6 psi_dti_dti_connect_res

Prototype:

LOCAL const void psi _dti_dti_connect_res (T_DTI2_CONNECT_RES *dti2_connect_res)

Parameters:

dti2_connect_res parameter of primitive

Return:

Description:

This function handles the primitive DTI2_CONNECT_RES.

4.1.9.7 psi_dti_dti_disconnect_ind

Prototype:

LOCAL const void psi _dti_dti_disconnect_ind (T_DTI2_DISCONNECT_IND *dti2_disconnect_ind)

Parameters:

dti2_disconnect_ind parameter of primitive

Return:

Description:

This function handles the primitive DTI2_DISCONNECT_IND.

4.1.9.8 psi_dti_dti_disconnect_req

Prototype:

LOCAL const void psi _dti_dti_disconnect_req (T_DTI2_DISCONNECT_REQ *dti2_disconnect_req)

Parameters:

dti2_disconnect_req parameter of primitive

Return:

Description:

This function handles the primitive DTI2_DISCONNECT_REQ.

4.1.9.9 psi_dti_dti_data_ind

Prototype:

LOCAL const void psi _dti_dti_data_ind (T_DTI2_DATA_IND *dti2_data_ind)

Parameters:

dti2_data_ind parameter of primitive

Return:

Description:

This function handles the primitive DTI2_DATA_IND.

4.1.9.10 psi_dti_dti_data_req

Prototype:

LOCAL const void psi _dti_dti_data_req (T_DTI2_DATA_REQ *dti2_data_req)

Parameters:

dti2_data_req parameter of primitive

Return:

Description:

This function handles the primitive DTI2_DATA_REQ.

4.1.9.11 psi_dti_dti_getdata_req

Prototype:

LOCAL const void psi _dti_dti_getdata_req (T_DTI2_GETDATA_REQ *dti2_getdata_req)

Parameters:

dti2_getdata_req	parameter of primitive

Return:

Description:

This function handles the primitive DTI2_GETDATA_REQ.

4.1.9.12 psi_dti_dti_data_test_ind

Prototype:

LOCAL const void psi _dti_dti_data_test_ind (T_DTI2_DATA_TEST_IND *dti2_data_test_ind)

Parameters:

dti2_data_test_ind	parameter of primitive

Return:

Description:

This function handles the primitive DTI2_DATA_TEST_IND.

4.1.9.13 psi_dti_dti_data_test_req

Prototype:

LOCAL const void psi _dti_dti_data_test_req (T_DTI2_DATA_TEST_REQ *dti2_data_test_req)

Parameters:

dti2_data_test_req	parameter of primitive

Return:

Description:

This function handles the primitive DTI2_DATA_TEST_REQ.

4.1.9.14 psi_dti_dti_ready_ind

Prototype:

LOCAL const void psi _dti_dti_ready_ind (T_DTI2_READY_IND *dti2_ready_ind)

Parameters:

dti2_ready_ind	parameter of primitive

Return:

Description:

This function handles the primitive DTI2_READY_IND.

4.1.9.15 pei_exit

Prototype:

LOCAL SHORT pei_exit (void)

Parameters:

Return:

PEI_OK	exit successful
PEI_ERROR	exit not successful

Description:

This function closes the communication channel of the entity PSI.

4.1.9.16 pei_init

Prototype:

LOCAL SHORT pei_init (T_HANDLE handle)

Parameters:

handle	task handle

Return:

PEI_OK	exit successful
PEI_ERROR	exit not successful

Description:

This function initializes the communication channel of PSI and the connection to the DTI. For the access to the DIO interface PSI calls dio_init in the first step. It serves for the initializing of the DIO interface layer. In the second step the PSI enrolls with the driver calling dio_user_init. The PSI services are initialized too.

4.1.9.17 pei_monitor

Prototype:

LOCAL SHORT pei_monitor (void ** out_monitor)

Parameters:

out_monitor	pointer to be monitored data

Return:

PEI_OK	exit successful
PEI_ERROR	exit not successful

Description:

This function is used to monitor entity data called by frame.

4.1.9.18 primitive_not_supported

Prototype:

LOCAL const void primitive_not_supported (void *data)

Parameters:

data	data of not supported data

Return:

Description:

This function frees memory of not supported primitives.

4.1.9.19 pei_primitive

Prototype:

LOCAL SHORT pei_primitive (void * primptr)

Parameters:

primptr	Pointer to the received primitive

Return:

PEI_OK	exit successful
PEI_ERROR	exit not successful

Description:

This function processes PSI primitives.

4.1.9.20 pei_run

Prototype:

LOCAL SHORT pei_run (T_HANDLE TaskHandle, T_HANDLE ComHandle)

Parameters:

handle	Communication handle

Return:

PEI_OK	exit successful
PEI_ERROR	exit not successful

Description:

This function is called by frame when entering the main loop and the entity runs in the active variant.

4.1.9.21 pei_signal

Prototype:

LOCAL SHORT pei_signal (ULONG opc, void *data)

Parameters:

opc	signal opcode
data	pointer to primitive

Return:

PEI_OK	exit successful
PEI_ERROR	exit not successful

Description:

This function handles the own signals sent by the DIO callback function.

4.1.9.22 pei_timeout

Prototype:

LOCAL SHORT pei_timeout (USHORT index)

Parameters:

index	timer index

Return:

Description:

This function handles expired PSI timer.

4.1.9.23 psi_dio_sign_callback

Prototype:

GLOBAL void psi_dio_sign_callback (T_DRV_SIGNAL *SigPtr)

Parameters:

SigPtr	pointer of structure: signal type, driver handle, data length, user data

Return:

Description:

This function is the callback function of the driver signals driven by DIO.

4.1.9.24 psi_dti_sign_callback

Prototype:

LOCAL void psi_dti_sign_callback(U8 instance, U8 interfac, U8 channel, U8 reason,
T_DTI2_DATA_IND *dti2_data_ind)

Parameters:

instance	DTI2 parameter
interfac	DTI2 parameter
channel	DTI2 parameter
reason	DTI2 parameter
dti2_data_ind	DTI2 parameter

Return:

Description:

This function is the callback function of DTI.

4.1.10 Module psi_rxf.c

4.1.10.1 psi_check_control_info

Prototype:

GLOBAL void psi_check_control_info(U32 device, T_DIO_CTRL_LINES* control_info_new)

Parameters:

device	device identifier
control_info_new	current driver control info delivered by the driver

Return:

Description:

This function checks the indicators of DTR line drop and the escape sequence detection. If one of these indicators has been found PSI sends the primitive PSI_LINE_STATE_IND to ACI.

4.1.10.2 psi_rx_init

Prototype:

GLOBAL void psi_rx_init (void)

Parameters:

Return:

Description:

This function initializes the internal service PSI_SERVICE_RX. This PSI service controls the data received from driver.

4.1.10.3 psi_rx_read

Prototype:

GLOBAL void psi_rx_read (void)

Parameters:

Return:

Description:

This function allocates and provides the read two buffers to the driver via the DIO function dio_set_rx_buffer. First a chain of descriptor elements depending on segment number of read buffer is allocated. Second the structures T_dio_buffer and T_dio_segment are created. The last structure contains pointers of each created descriptor element to avoid data copy.

4.1.10.4 psi_rx_reconf_pkt

Prototype:

GLOBAL void psi_rx_reconf_pkt (void)

Parameters:

Return:

Description:

This function reconfigures the segment sizes for packet data. The size of segment 0 refers to the packet capability parameter mtu_data. The second segment contains only the protocol identifier.

4.1.10.5 psi_ rx_send_data_to_dtx

Prototype:

GLOBAL void psi_rx_send_data_to_dtx (T_dio_buffer* buffer, T_DIO_CTRL* control_info)

Parameters:

buffer	pointer to received driver data
control_info	contains the delivered driver control information

Return:

Description:

This function checks the descriptor lists of the read buffer filled by the serial device, set the number of used data bytes in the according chain of descriptor elements and calls the function psi_rx_dtx_data which sends the data and the control info via DTI to the peer entity.

4.1.10.6 psi_rx_send_data_to_dtx_pkt

Prototype:

GLOBAL void psi_rx_send_data_to_dtx_pkt (T_dio_buffer* buffer)

Parameters:

buffer pointer to received driver data

Return:

Description:

This function checks the descriptor lists of the read buffer filled by the packet device, set the number of used data bytes in the according chain of descriptor elements, separates the protocol identifier from the first segment. Then the function calls the function psi_rx_dtx_data_pkt which sends the data and the protocol identifier via DTI to the peer entity.

4.1.11 Module psi_rxp.c

4.1.11.1 psi_tx_sig_read_ind

Prototype:

GLOBAL void psi_tx_sig_read_ind (U32 device)

Parameters:

device device identifier

Return:

Description:

This function handles the PSI own signal PSI_SIG_READ_IND based on driver signal DRV_SIGTYPE_READ. It indicates that the driver starts to write data in the first read buffer. PSI gets back the buffer controlling via the DIO function dio_read and sends immediately the read data to the peer entity (via DTI) calling psi_rx_send_data_to_dtx bzw. psi_rx_send_data_to_dtx_pkt . In case of changed driver control info the primitive PSI_LINE_STAT_IND is sent to ACI. The further process depends on the state of internal service PSI_SERVICE_RX : a)in case PSI_RX_READY after the data are sent to DTI the last read read buffer is sent to the driver (psi_rx_read), b) in case PSI_RX_NOT_READY the read data and descriptor list is stored in the PSI data base until DTI is ready to receive data again. That means the state PSI_RX_NOT_READY changes to PSI_RX_BUFFER, furthermore the last read read buffer is sent to the driver (psi_rx_read), c) in case PSI_RX_BUFFER the second read buffer and descriptor list is stored in the PSI data base until DTI is ready to receive data again.

4.1.12 Module psi_rxs.c

4.1.12.1 psi_dtx_rx_ready

Prototype:

GLOBAL void psi_dtx_rx_ready (void)

Parameters:

Return:

Description:

If DTI signals PSI that the read buffers are ready via « DTI_REASON_TX_BUFFER_READY » the internal buffered data are sent to DTI calling psi_rx_send_data_to_dtx bzw. psi_rx_send_data_to_dtx_pkt. After the data are sent to DTI the last sent read buffer is sent to the driver (psi_rx_read). Only if both buffered read buffer are sent to DTI the state PSI_RX_BUFFER changes to PSI_RX_NOT_READY.

4.1.12.2 psi_dtx_rx_close

Prototype:

GLOBAL void psi_dtx_rx_close (void)

Parameters:

Return:

Description:

This function handles the services PSI_SERVICE_RX if user, DTI or driver wants to close the connection.

4.1.12.3 psi_ker_rx_close

Prototype:

GLOBAL void psi_ker_rx_close (void)

Parameters:

Return:

Description:

This function handles the services PSI_SERVICE_RX after driver deregistration. The internal buffered read buffer and descriptor lists are deallocated.

4.1.12.4 psi_ker_rx_open

Prototype:

GLOBAL void psi_ker_rx_open (void)

Parameters:

Return:

Description:

This function handles the service PSI _SERVICE_RX after driver registration. PSI provides two read buffer to the driver calling psi_rx_read.

4.1.13 Module psi_txf.c

4.1.13.1 psi_create_send_buffer

Prototype:

GLOBAL void psi_create_send_buffer(T_dio_buffer** buffer, T_desc_list2 *list, U8 pid)

Parameters:

buffer	contains the allocated write buffers
list	pointer to the descriptor list containing data sent by DTI
p_id	protocol identifier (only for packet devices)

Return:

Description:

This function allocates the write buffer structure T_dio_buffer and T_dio_segment and sets the addresses of filled descriptor element buffers in the segment structure to avoid data copy. For packet devices the protocol identifier p_id is stored in a separate segment.

4.1.13.2 psi_converts_control_info_data

Prototype:

GLOBAL void psi_converts_control_info_data (T_parameters *dtx_ctrl_info)

Parameters:

dtx_ctrl_info	pointer to the internal state parameter sent by DTI

Return:

Description:

This function converts the line states sent by DTI in the DIO control info structures and checks whether flushing is needed.

4.1.13.3 psi_tx_init

Prototype:

GLOBAL void psi_tx_init (void)

Parameters:

Return:

Description:

This function initializes the internal service structures of PSI_SERVICE_TX and according data of the global PSI data base.

4.1.14 Module psi_txp.c

4.1.14.1 psi_free_tx_buffer

Prototype:

GLOBAL void psi_free_tx_buffer (T_dio_buffer* write_buffer)

Parameters:

write_buffer	contains written data

Return:

Description:

After the signal DRV_SIGTYPE_WRITE this function frees the successful sent descriptor lists and the according write buffer structures.

4.1.14.2 psi_fill_tx_buf_list

Prototype:

GLOBAL void psi_fill_tx_buf_list (T_dio_buffer* write_buffer, T_desc2* list)

Parameters:

write_buffer	contains written data
list	descriptor list

Return:

Description:

After the sending of data to the device this function stores the descriptor list and the according write buffer structures in the according lists of the psi data base.

4.1.14.3 psi_tx_sig_flush_ind

Prototype:

GLOBAL void psi_tx_sig_flush_ind (U32 device)

Parameters:

device	device identifier

Return:

Description:

This function handles the PSI own signal PSI_SIG_FLUSH_IND based on driver signal DRV_SIGTYPE_FLUSH. It indicates that data of the hardware write buffer are completely sent. In case PSI_TX_FLUSHING PSI sends the new configuration parameter to the driver via psi_ker_tx_flushed. The state changes to PSI_TX_READY. The DTI connection is started again via psi_dtx_rx_ready.

4.1.14.4 psi_tx_sig_write_ind

Prototype:

GLOBAL void psi_tx_sig_write_ind (U32 device)

Parameters:

device	device identifier

Return:

Description:

This function handles the PSI own signal PSI_SIG_WRITE_IND based on driver signal DRV_SIGTYPE_WRITE. It indicates that the driver sent the first write buffer completely. PSI gets back the buffer controlling via the DIO function dio_get_tx_buffer.

In case PSI_TX_READY the according chain of descriptor elements and the write buffer structures are deallocated. The number of sent write buffer decreases. For certain circumstances the flushing procedure is started.

In case PSI_TX_BUFFER the according chain of descriptor elements and the write buffer structures are deallocated. Afterwards the pending write buffer is sent to the driver. Buffer and descriptor list are stored in the global PSI data base. The number of sent write buffer increases.Under certain circumstances the DTI connection is started callingpsi_dtx_rx_ready .

4.1.15 Module psi_txs.c

4.1.15.1 psi_ker_tx_close

Prototype:

GLOBAL void psi_ker_tx_close (void)

Parameters:

Return:

Description:

This function handles the services PSI_SERVICE_TX after driver disconnect indication. The internal state of PSI_SERVICE_DRX is set via psi_tx_drx_close.

4.1.15.2 psi_ker_tx_flush

Prototype:

GLOBAL void psi_ker_tx_ flush (void)

Parameters:

Return:

Description:

This function handles the services PSI _SERVICE_TX during flushing of hardware write buffer. In case PSI_TX_READY the function dio_flush is called. The flush result is evaluated. In best case the new configuration parameter are sent to the driver (psi_ker_tx_flushed). If DTI connection is down PSI starts the connection via psi_dtx_rx_ready. In worse case the state PSI_TX_FLUSHING is set.

4.1.15.3 psi_ker_tx_open

Prototype:

GLOBAL void psi_ker_tx_open (void)

Parameters:

Return:

Description:

This function handles the services PSI _SERVICE_TX after driver registration.

4.1.15.4 psi_drx_tx_data

Prototype:

GLOBAL void psi_drx_tx_data (T_desc_list2* list, T_parameters* dtx_control_info)

Parameters:

list	pointer to data sent by DTI
dtx_control_info	pointer to line states (serial devices only) sent by DTI

Return:

Description:

This function converts the via DTI received data structures in the DIO4 refered structures (via psi_create_send_buffer). It is also necessary to check the line states sent in the DTI primitive. For certain circumstances the flushing procedure has to be start. If already the maximal number of write buffers are sent to the driver the new write buffer and the according descriptor element chain are depoted in the global PSI data base. Otherwise PSI sends the data immediately to the driver calling the DIO functiondio_write . Furthermore the internal service states of PSI_SERVICE_TX are handled. The number of sent write buffer increases and the DTI connection is started again (psi_tx_drx_ready).

4.1.15.5 psi_drx_tx_data_pkt

Prototype:

GLOBAL void psi_drx_tx_data_pkt (T_desc_list2* list, U8 protocol_identifier)

Parameters:

list	pointer to data sent by DTI
protocol_identifier	protocol identifier (packet devices only)

Return:

Description:

This function converts the via DTI received data structures in the DIO4 refered structures (via psi_create_send_buffer). If already the maximal number of write buffers are sent to the driver the new write buffer and the according descriptor element chain are depoted in the global PSI data base. Otherwise PSI sends the data immediately to the driver calling the DIO function dio_write. Furthermore the internal service states of PSI_SERVICE_TX are handled. The number of sent write buffer increases and the DTI connection is started again (psi_dtx_rx_ready).

4.2 New Data Structures

4.2.1 PSI services and states

PSI_SERVICE_KER PSI kernel service; handles the SAP primitives

PSI_KER_DEAD
PSI_KER_CONNECTING
PSI_KER_READY
PSI_KER_MODIFY

PSI_SERVICE_RX PSI driver data receiving service

PSI_RX_DEAD_NOT_READY
PSI_RX_NOT_READY
PSI_RX_READY
PSI_RX_BUFFER

PSI_SERVICE_TX PSI driver data sending service

PSI_TX_DEAD
PSI_TX_READY
PSI_TX_BUFFER
PSI_TX_FLUSHING

PSI_SERVICE_DTX PSI DTI data sending service

PSI_DTX_DEAD
PSI_DTX_NOT_READY
PSI_DTX_READY

PSI_SERVICE_DRX PSI DTI data receiving service

PSI_DRX_DEAD_NOT_READY
PSI_DRX_DEAD_READY
PSI_DRX_NOT_READY
PSI_DRX_READY

4.2.2 Further PSI states

DTI connection related internal states:

PSI_KER_DTI_DEAD
PSI_KER_DTI_OPENING
PSI_KER_DTI_READY

4.2.3 PSI internal data structure

```
PSI_INSTANCES          6    /* max number of PSI instances */
PSI_SEG _NUM           4    /* max number of data segments */
PSI_SEG_SER_NUM        1    /* max number of read buffer segments for serial data    */
PSI_SEG_PKT_NUM        2    /* max number of read buffer segments  for packet data */
PSI_WRITE_BUF_MAX      2    /* max number of provided write buffer   */
PSI_READ_BUF_MAX       2    /* max number of provided read buffer   */
PSI_NUM_PID_BYTE       2    /* max number of bytes for protocol identifier */
```

```
Typedef enum
{
        BUFFER_1 =1,
        BUFFER_2,
        BUFFER_MAX
}T_NEXT_SEND_DTI_ID;

typedef struct
{
  UBYTE              state;              /* kernel service state    */
  UBYTE              dti_state;          /* state of the dti connection*/
  T_DIO_CAP*         capabilities;       /* stored driver capabilities */
} T_KER_DATA;

typedef struct
{
  UBYTE              state;                          /* rx service state */
  U16                psi_seg_size [PSI_SEG_NUM]; /* segment sizes for read buffer */
  U8                 number_segm;                    /* How many segments exist */
  T_dio_buffer*      psi_buffer_1;                   /* pointer to read buffer 1 */
  T_dio_buffer*      psi_buffer_2;                   /* pointer to read buffer 2 */
  T_dio_buffer*      psi_buffer_pending[PSI_READ_BUF_MAX];     /* pointer to pending read
buffer list */
  T_desc2*           psi_buffer_1_desc;              /* pointer to desc list of buffer 1  */
  T_desc2*           psi_buffer_2_desc;              /* pointer to desc list of buffer 2  */
```

```
  T_desc2*              psi_buffer_desc_pending[PSI_READ_BUF_MAX];      /* pointer to pending
desc list of read buffer */
  BOOL                  psi_buffer_1_used;              /* flag for buffer using state */
  BOOL                  psi_buffer_2_used;
  BOOL                  psi_buffer_pend[PSI_READ_BUF_MAX];/* flag for buffer send state (to DTI) */
  BOOL                  psi_buffer_1_read;              /* control if buffer is already processed */
  U8                    next_send_id;                   /* index of next sent buffer */
  T_DIO_CTRL_LINES psi_control_info_ser;                /* stored control info for serial data sent by
DIO */
} T_RX_DATA;

typedef struct
{
  UBYTE                 state;                          /* tx service state */
  T_dio_buffer*         psi_buffer_pending;             /* pointer to pending write buffer */
  T_ desc2*             psi_buffer_desc_pending;        /* pointer to pending desc list */
  T_dio_buffer *        psi_buffer_pending_flush;        /* buffered packet, send after flushing */
  T_desc2 *             psi_buffer_desc_pending_flush; /* buffered packet for the DESC, send after
flushing  */
  T_dio_buffer*         psi_buffer[PSI_WRITE_BUF_MAX];      /* pointer to write buffer array */
  T_ desc2*             psi_buf_desc[PSI_WRITE_BUF_MAX]; /* pointer to desc list array 1*/
  BOOL                  psi_buffer_1_used;              /* flag for buffer using state */
  BOOL                  psi_buffer_1_pend;              /* flag for buffer send state (to DIO)*/
  BOOL                  psi_dio_flush;                  /* flag flushing procedure*/
  BOOL                  flag_line_state_req;            /* flag for PSI_LINE_STATE_REQ */
  U8                    in_driver;                      /* number of sent write buffer */
  T_DIO_CTRL_ LINES  psi_control_info_ser;              /* stored control info sent by DTI */
} T_TX_DATA;

typedef struct
{
  UBYTE                 state;                  /* dtx service state                    */
} T_DTX_DATA;

typedef struct
{
  UBYTE                 state;                  /* drx service state                    */
} T_DRX_DATA;

typedef struct
{
  UBYTE                 version;
  T_KER_DATA            ker;                    /* kernel service data          */
  T_RX_DATA             rx;                     /* RX service data              */
  T_TX_DATA             tx;                     /* TX service data              */
  T_DRX_DATA            drx;                    /* DRX service data             */
  T_DTX_DATA            dtx;                    /* DTX service data             */
  U8                    instance;               /* PSI instance                 */
  BOOL                  used;                   /* used instance                */
  U32                   device_no;              /* affected device identifier   */
  T_DIO_DCB_SER*        dcb;                    /* pointer to serial device control block   */
  T_DIO_DCB_PKT*        dcb;                    /* pointer to packet device control block */
  DTI_HANDLE            hDTI;                   /* DTI handle                   */
  U32                   link_id;                /* according DTI link identifier    */
} T_PSI_DATA;
```

5 Test Strategy

5.1 Host Test

For testing of the functional DIO interface some test primitives are introduced called PSI_DIOSIM_xx. The C file psi_diosim.c contains the simulation of the DIO functions described in [2]. The simulation sends test primitives to TAP and awaits the appropriate confirmation primitives. The primitive handling is created in psi_diosim_get_conf.

Casting of structures within TDC is not possible. Due to this fact for the first implementation step of interface PSI – ACI some testprimitives, including serial transmission parameter, are introduced.

5.2 Target Test

For the first step the entity PSI handles similar to the UART entity supporting AT commands and serial data exchange. It is possible to configure the USB driver via AT commands and to test gprs data calls, internet surfing and ftp up – and downloading.

Appendices

A. Acronyms

ACI	Application Control Interface
DTI	Data Transmission Interface
DIO	Data I/O
FFS	Flash File System
PSI	Protocol Stack Interface

B. Glossary

Entity	Program that executes the functions of a layer.
Message	A message is a data unit, which is transferred between the entities of the same layer (peer-to-peer) of the mobile and infrastructure side. Message is used as a synonym to protocol data unit (PDU). A message may contain several information elements.
Primitive	A primitive is a data unit, which is transferred between layers on one component (mobile station or infrastructure). The primitive has an operation code, which identifies the primitive and its parameters.
Service Access Point	A Service Access Point is a data interface between two layers on one component (mobile station or infrastructure).
Device	A data sink or source out of GSM/GPRS protocol stack