AR7 ATM Device Drive Porting Guide

Revision: 0.5

Revision Date: 7th April, 2005

Texas Instruments Inc.

Dallas, TX

Foreword

The AR7 Linux ATM Device Driver Porting Guide provides documention for porting purpose.

Revision History

Rev.	Date	Description	Author
0.1	05/12/2003	Original document	Zhicheng Tang
0.2	04/15/2004	Update	Zhicheng Tang
0.3	07/19/2004	Added two new commands for bitswap control (bitswapon, bitswapoff)	Brian Chen
0.4	08/10/2004	Added environment variable setting for DSP speed configuration; added READSL modes to the training mode table reference	Ram Parasuraman
0.5	04/07/2005	Updated the API for the tn7sar_activate_vc and updated the environment variables definitions.	Arvind Vasudev

Contents

1 Introduction

This document describes implementation details of AR7 Linux ATM driver. Details of CPSAR HAL and DSL HAL are excluded from this documentation. For references on them, please refer to “Ax7 DSL HAL Programmer's Guide” and “Communication Processor Hardware Abstraction Layer (CPHAL) API Reference Manual”.

2 References

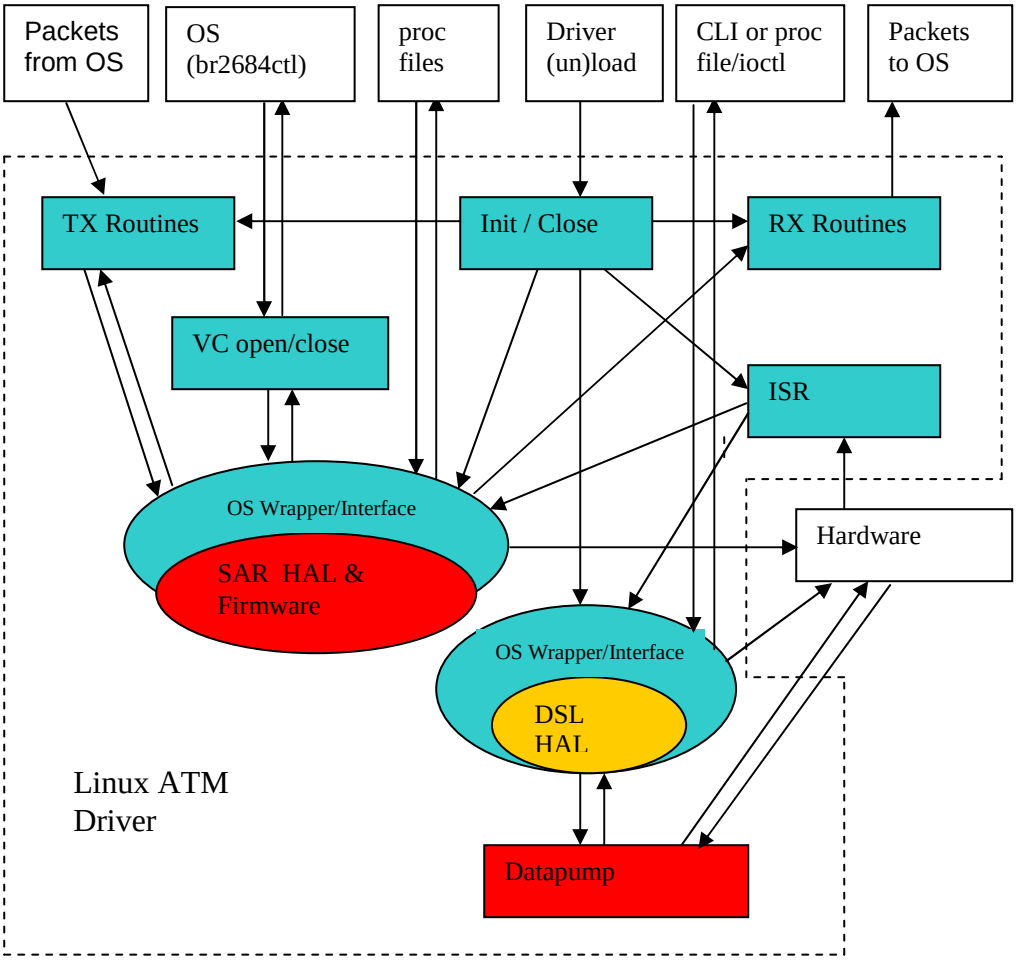
[1] Texas Instruments Inc., Ax7 DSL HAL Programmer's Guide.

[2] Texas Instruments Inc., Communication Processor Hardware Abstraction Layer (CPHAL) API Reference Manual.

[3] Texas Instruments Inc., AR7 Feature Control API Application Note Version 1.2.

3 AR7 Driver Architecture

AR7 Linux ATM driver is built as dynamically loadable modules. Figure 1 below shows the functional interfaces and software delineation of AR7 Linux ATM driver.



OS: Linux operating system.
CLI: Command line interface or any user space applications.
Proc file: Application and kernel (driver) interface for Linux.
Ioctl: I/O interface between application and the driver.

Figure 1. DSL driver architecture and module interactions.

4 ATM Device Driver Files

The TNETD73xx chipset provides hardware supports for both AAL5 packet segmentation and reassemble and ADSL physical layer. The Linux driver provides APIs to all interfaces as well to IP stack. The files for AR7 ATM device driver are described in following table.

Table 1. ATM device driver files

Files	Description
Makefile	Primary Makefile for the ATM driver.
makefile.loc	Makefile to build driver alone.
tn7atm.c	Driver entry, OS interface.
tn7atm.h	Private data structures and ATM device specific structures.
tn7api.h	API declarations.
tn7dsl.c	OS related DSL physical layer wrapper and functions.
tn7sar.c	OS related SAR wrapper and functions
turbodsl.c	Implemented turbo DSL feature.
queue.h	Private structure definition for tubodsl.c.
dsl_hal_api.c	OS independent functions of DSL functions
dsl_hal_api.h	include file for OS independent functions of DSL functions
dsl_hal_support.c	Support functions of OS independent functions of DSL functions
dsl_hal_support.c	Include file for dsl_hal_support.c.
dsl_hal_register.h	DSL related register defines.
dev_host_interface.h, dev_host_verdef.h, env_def_defines.h, env_def_typedefs.h time.h	DSL related include files.
tnetd7300_sar_firm.h	PDSP firmware for Tnetd73xx.
ar0700mp.bin	DSL firmware for Annex A.
ar0700db.bin	DSL firmware for Annex B.
ar0700dc.bin	DSL firmware for Annex C.
aal5sar.c, cpcommon_cpaal5.c, cpcommon_cpsar.c, cppi_cpaal5.c, cpremap_cpaal5.c, cpremap_cpsar.c, cpsar.c, aal5sar.h, cp_sar_reg.h, cpcommon_cpaal5.h, cpcommon_cpsar.h, cpsar.h, cpsar_cpaal5.h, cpswhal_cpaal5.h, cpswhal_cpsar.h, ec_errors_cpaal5.h, ec_errors_cpsar.h	SAR HAL related files.

4 Driver Functionality Description

4.1 Driver initialization

The AR7 ATM driver is initialized through the **tn7atm_init** function. The **tn7atm_init** function allocates memory for the private data structure used by ATM drive and calls the initialization routines of SAR HAL and DSL HAL to initialize both SAR and DSL subsystems . Upon completion of **tn7atm_init** function the driver is ready to accept socket calls from ATM applications at user level. Interrupt registrations are also handled in **tn7atm_init .** The **tn7atm_open** function is called when a user applications such as PPPoE or BR2684ctl makes a socket call to the driver to open a new connection to the SAR interface.

The **tn7atm_open** function initializes the VCC structure with the appropriate ATM VPI/VCI and QoS options. The ATM device operation structure is shown in Figure 12.

```
static const struct atmdev_ops tn7atm_ops = {
        open:           tn7atm_open,
        close:          tn7atm_close,
        ioctl:          tn7atm_ioctl,
        getsockopt:     NULL,
        setsockopt:     NULL,
        send:           tn7atm_send,
        sg_send:        NULL,
        phy_put:        NULL,
        phy_get:        NULL,
        change_qos:     tn7atm_change_qos,
};
```

4.2 Packet transmission

Once a proper VPI/VCI pair is initialized through user application, data can be transmitted through the **tn7atm_send** function. The **tn7atm_send** function checks the status of DSL interface and AR7 modem to ensure that a valid connection is available to send data on. The type of data is also checked and a determination is made to whether to queue the packet to high priority queue or low priority queue for further data transmission. The driver checks whether there is packets to send in every **tn7_send** call. If there is any packet to be sent, the driver will locate channel information from vcc structure and call SAR HAL send routine to send out packet. The packets are de-queued from priority queue first. Upon completion, the SAR interrupt is handled through the **tn7atm_sar_irq** function. The **tn7atm_sar_irq** calls the interrupt handle routine of SAR HAL which in turn calls the **tn7atm_send_complete** to free the packets.

4.3 Packet Reception

The ATM driver processes the reception of incoming packets (i.e., from DSL interface) through the registered SAR interrupt (**tn7atm_sar_irq)**. **tn7atm_sar_irq** calls SAR HAL interrupt handling routine which in turn calls the **tn7atm_receive** to pass on received packets. If no errors exist within the PDU, skb is passed to the IP stack through the **atm_charge** and **vcc->push** routines. The SAR HAL will call **tn7atm_allocate_rx_skb** to replenish the used skb.

4.4 Command interface

The ATM driver provides a single command interface that an user application can call to pass command or information to the driver. To access this interface, user application needs to write a string to the proc file **/proc/sys/dev/dslmod** with following,

echo command > /proc/sys/dev/dslmod

The **dslmod_sysctl** in tn7dsl.c will parse the string and take appropriate action. The following commands are currently supported. They must be in the *string* format. Table 2 shows their name and descriptions.

Table 2. ioctl commands and description

Command	Description
T1413	Set DSL training mode to T1.413.
GDMT	Set DSL training mode to G.dmt.
GLITE	Set DSL training mode to G.Lite.
MMOD	Set DSL training mode to multi mode.
NMOD	Set no training mode.
AD2MOD	Set DSL training mode to ADSL2
AD2DEL	Set DSL training mode to ADSL2 DELT
A2PMOD	Set DSL training mode to ADSL2+ mode
A2PDEL	Set DSL training mode to ADSL2+ DELT mode
tmode nn	Set DSL training mode based on bit cmd, where nn is the two-digit hex number, which specifies the training mode.
exxxpyyyyc	Send end to end F5 cells with VPI=xxx and VCI=yyyy. xxx and yyyy are in decimal.
sxxxpyyyyc	Send seg to seg F5 cells with VPI=xxx and VCI=yyyy. xxx and yyyy are in decimal.
exxxp0c	Send end to end F4 cells with VPI=xxx.
sxxxp0c	Send segment F4 cells with VPI=xxx.
exxxpyyyycdzzzt	Send end to end F5 cells with VPI=xxx and VCI=yyyy. xxx and yyyy are in decimal. zzz is time out value in millisecond
sxxxpyyyycdzzzt	Send seg to seg F5 cells with VPI=xxx and VCI=yyyy. xxx and yyyy are in decimal. zzz is time out value in millisecond.
trellison	Set trellis on
trellisoff	Set trellis off
bitswapon	Set bitswap ON
bitswapoff	Set bitswap OFF

4.5 Proc files

AR7 ATM driver provides several proc files for user application to retrieve information from the driver. They are listed in following table.

Table 3 Proc file list.

File name and path	Description
/proc/avalanche/avsar_modem_stat	Statistics for SAR and DSL.
/proc/avalanche/avsar_modem_training	First line: modem training state.
/proc/avalanche/avsar_ver	Version information for the Driver, PDSP, DSP, and etc.
/proc/avalanche/avsar_oam_ping	Oam ping result.
/proc/avalanche/avsar_pvc_table	Received PVC table information.
/proc/avalanche/avsar_sarhal_stats	Unformatted SAR HAL statistics. (debug only)
/proc/avalanche/avsar_channels	Channel information. (debug only)

4.5.1 Important Defines

The table below lists the important defines and their corresponding values used to interpret the meaning of the data in the modem stat file.

Table 4 Trellis and Bitswap Status

Name	Value
DISABLE	0
ENABLE	1

Table 5 Modem Training Mode

Name	Value	Comments
NO_MODE	0	
MULTI_MODE	1	Default modem modulation mode
T1413_MODE	2	
GDMT_MODE	3	
GLITE_MODE	4	
ADSL2_MODE	8	
ADSL2_DELT	9	
ADSL2_PLUS	16	
ADSL2_PLUS_DELT	17	
READSL2_MODE	32	
READSL2_DELT	33	

4.6 Environment variables

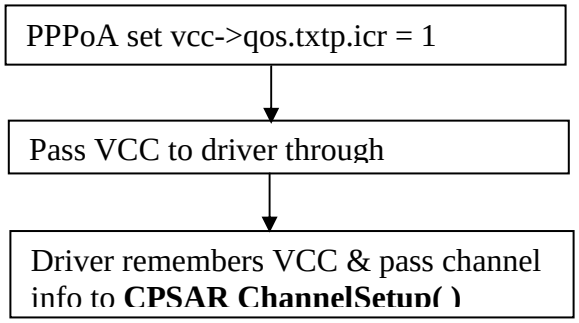
The following environment variables are supported for various DSL operational needs.

Name	Value type	Meaning
modulation mmm	String	Same as command, where mmm is one of the first 9 string commands from Table 2.
modulation nnn	String	Same as command, where nnn is a decimal number specifying the corresponding command code (0 – 255) of one of the first 9 string commands in Table 2.
trellis	Integer	0, trellis off; 1 trellis on.
eoc_vendor_id	Hex	8 byte in ascii
eoc_vendor_revision	Integer	Revision In decimal.
eoc_vendor_serialnum	String	Null terminated string.
fine_gain_control	Integer	0, control off; 1 control on.
fine_gain_value	Integer	Gain value in hex.
maximum_bits_per_carrier	Integer	Value sets the maximum bits per carrier. The valid range is between 0-15.
DSL_FEATURE_CNTL_0	Integer	Controls the features for the datapump software are defined in AR7 Feature Control API Application Note [3].
DSL_FEATURE_CNTL_1	Integer	Controls the features for the datapump software are defined in AR7 Feature Control API Application Note [3].
dsp_noboost	Integer	0 = enable attempt to boost Dsp frequency to 250 MHz, 1 = disable attempt to boost Dsp frequency to 250 MHz
dsp_freq	integer	250 -> Run DSP at 250 Mhz Any other value -> Run DSP at 200 Mhz

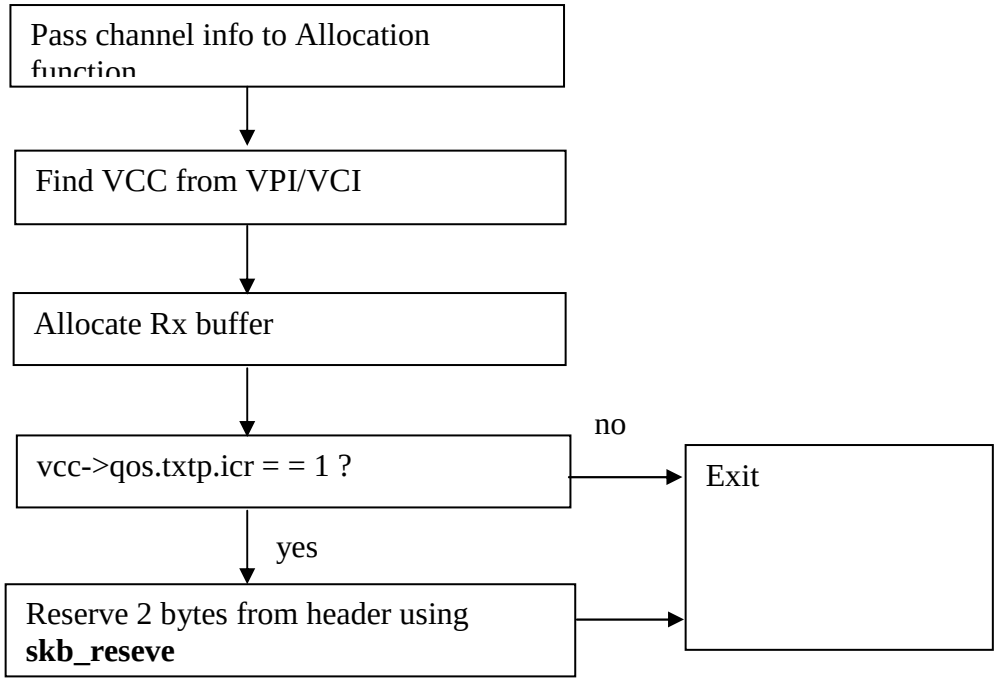
4.7 PPPoA No Copy Support

Current PPPoA support uses packet copy function to align IP portion of the packets to 32-bit boundary. This is costly operation that consumes a lot of CPU cycles. In order to overcome this problem, DSL/ATM driver needs to provide PPP packets to upper layer aligned to 16-bit boundary. The driver can not blankly align all packets to 16-bit boundary because it will mess up all other packets. The designed work around is shown below.

1. During channel setup, PPPoA will set



2. When CPSAR needs to Allocate RX buffers

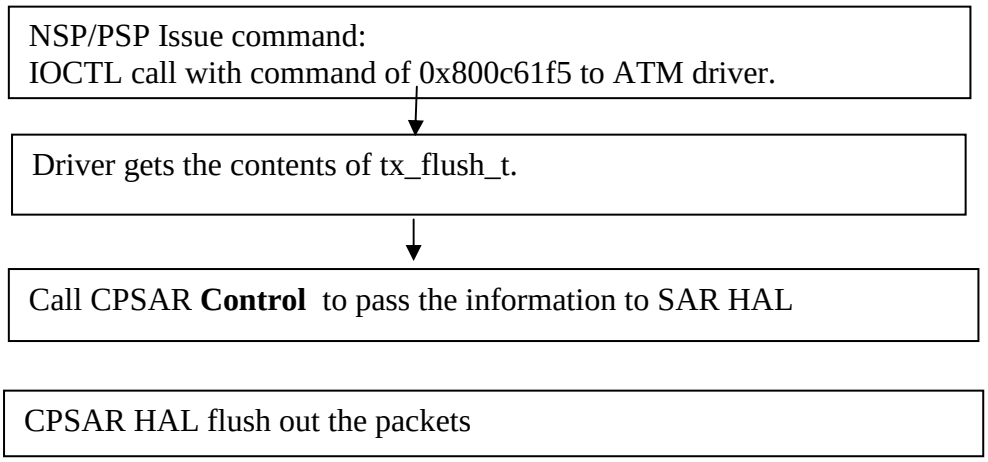


4.8 Flush TX buffers queued in Queue

In an application that supports both data and voice in a single PVC setup, there is a problem of head line blocking. The head line blocking occurs when a high priority voice packet has to wait for any low priority packet already in transmission. To reduce that problem, there is need to flush out all data packet already in TX queue during voice call startup. To implement this, a struct as following is defined

```
typedef struct _tx_flush
{
  struct atm_vcc *vcc;
  int queue;
  int num_skip;
}tx_flush_t;
```

Application or other kernel level component can issue an IOCTL call to ATM driver with command of 0x800c61f5 and passes a pointer to above struct.



5 APIs

5.1 APIs from tn7atm.c

5.1.1 int tn7atm_detect(void)

Description:
This function is called when device driver is loaded into memory. This function will call tn7atm_init.
Parameters:
void
Returns:
0, if the function succeeds.
< 0, if failed.

5.1.2 int tn7atm_init(struct atm_dev *dev)

Description:
This function handles initialization of SAR and DSL subsystems as well as registering interrupts with the Linux Kernel.
Parameters:
dev, pointer to the Linux atm device structure. The structure contains the device private data structure, configuration, and device state information.
Returns:
0, if the function succeeds.
< 0, if failed.

5.1.3 int tn7atm_ irq_request (struct atm_dev *dev)

Description:
This function initializes interrupts required for SAR and DSL subsystems.
Parameters:
dev, pointer to the Linux atm device structure. The structure contains the device private data structure, configuration, and device state information.
Returns:
0, if the function succeeds.
< 0, if failed.
Possible causes of error could be failure to register the irq with the kernel.

5.1.4 tn7atm_sar_irq(int irq , void *voiddev , struct pt_regs *regs)

Description:
This function is SAR interrupt handler.
Parameters:
irq, SAR interrupt number.
dev, a void device pointer to associate the interrupt with the ATM device.
regs, used by the kernel request_irq function.
Returns:
0, if the function succeeds.
< 0, if failed.

5.1.5 tn7atm_dsl_irq(int irq , void *voiddev , struct pt_regs *regs)

Description:
This function is DSL interrupt handler.
Parameters:
irq, DSL interrupt number.
dev, a void device pointer to associate the interrupt with the ATM device.
regs, used by the kernel request_irq function.
Returns:
0, if the function succeeds.
< 0, if failed.

5.1.6 int tn7atm_ get_ESI (struct atm_dev *dev)

Description:
This function gets the ATM ESI (MAC) information from ADAM2 environment space.
Parameters:
dev, pointer to the Linux atm device structure. The structure contains the device private data structure, configuration, and device state information.
Returns:
0, always succeeds.

5.1.7 static int tn7atm_open (struct atm_vcc *vcc, short vpi, int vci)

Description:
This function is called by upper layers to open a PVC connection.
Parameters:
vcc, pointer to per PVC structure.
vpi, VPI value of a PVC.
vci, VCI value of a PVC.
Returns:
0, if the function succeeds.
< 0, if failed.

5.1.8 static void tn7atm_close (struct atm_vcc *vcc)

Description:
This function is called by upper layers to close a PVC connection.
Parameters:
vcc, pointer to per PVC structure.
Returns:
None.

5.1.9 static int tn7atm_send (struct atm_vcc *vcc, struct sk_buff *skb)

Description:
This function is called by upper layers to pass a skb (packet) to the driver to be sent.
Parameters:
vcc, pointer to per PVC structure.
skb, packet to be sent.

Returns:
0, if the function succeeds.
1, if failed.

5.1.10 int tn7atm_send_complete(void *osSendInfo)

Description:
Called by SAR HAL to pass the packet sent to be freed.
Parameters:
osSendInfo, pointer to skb.
Returns:
0, if the function succeeds.
1, if failed.
Possible causes of error could be failure to register.

5.1.11 int tn7atm_receive(void *os_dev, int ch, unsigned int packet_size, void *os_receive_info, void *data)

Description:
Called by SAR HAL to pass received packets to driver.
Parameters:
os_dev, pointer to atm device structure.
packet_size, packet length of received packet.
os_receive_info, pointer to skb.
data, pointer to data buffer of a skb.
Returns:
0, if the function succeeds.
1, if failed.
Possible causes of error could be failure to register.

5.1.12 void *tn7atm_allocate_rx_skb(void *os_dev, void **os_receive_info, unsigned int size)

Description:
Called by SAR HAL to allocate a skb for Rx.
Parameters:
os_dev, pointer to atm device structure.
os_receive_info, pointer to skb.
size, buffer length of allocated skb.
Returns:
pointer to skb buffer pointer.

5.1.13 void tn7atm_free_rx_skb(void *skb)

Description:
Called by SAR HAL to free a Rx skb.
Parameters:
skb, pointer to skb.
Returns:

5.1.14 static int tn7atm_register (Tn7AtmPrivate * priv)

Description:
This function registers Linux ATM operational functions with upper layer.
Parameters:
priv, pointer to device's private structure.
Returns:
0, if the function succeeds.
1, if failed.
Possible causes of error could be failure to register.

5.1.15 static void tn7atm_exit (void)

Description:
Exit function for the driver. The function is called when "rmmod" command is issued. All resources allocated for the driver are de-allocated here.
Parameters:
void.
Returns:
None.

5.1.16 int tn7atm_queue_packet_to_sar(void *vcc1, void *skb1)

Description:
This function forwards skb from a PVC to SAR to be sent.
Parameters:
vcc1, void pointer to per PVC structure.
skb1, packet to be sent.
Returns:
0, if the function succeeds.
1, if failed.

5.1.17 static int tn7atm_lut_clear(struct atm_vcc *vcc, int chan)

Description:
This function resets LUT for the channel.
Parameters:

vcc, pointer to per OS allocated PVC structure
chan, channel number that need to be cleared from LUT.
Returns:
Always 0.

5.1.18 int tn7atm_lut_find(short vpi, int vci)

Description:
This function finds SAR channel number for an opened PVC.
Parameters:
vpi, VPI value.
vci, VCI value.
Returns:
Channel number associated with vpi and vci..

5.1.19 tn7atm_set_lut(Tn7AtmPrivate *priv, struct atm_vcc *vcc, int chan)

Description:
Sets the information for a channel in the LUT.
Parameters:
priv, pointer to device private structure.
vcc, pointer to per PVC structure.
chan, channel to set.
Returns:
1, if the function succeeds.
< 0, failed.

5.1.20 void tn7atm_free_packet(void *pVc, void *pDev, void *pPacket)

Description:
This function frees a Linux skb (packet).
Parameters:
pVc, pointer to per PVC structure.
pDev, pointer to device structure.
pPacket, pointer to a Linux skb.
Returns:
None.

5.1.21 int tn7atm_device_connect_status(void *priv, int state)

Description:
This function sets the connection status
Parameters:
priv, pointer to private structure.
state, state to set.
Returns:
0.

5.1.22 static int tn7atm_proc_version(char* buf, char **start, off_t offset, int count,int *eof, void *data)

Description:
Linux proc file to provide version information.

5.1.23 static int tn7atm_proc_channels (char* buf, char **start, off_t offset, int count,int *eof, void *data)

Description:
Linux proc file to provide SAR channel information.

5.1.24 static int tn7atm_proc_private (char* buf, char **start, off_t offset, int count,int *eof, void *data)

Description:
Linux proc file to provide device private date structure information.

5.2 APIs from tn7dsl.c

5.2.1 int tn7dsl_init(void *priv)

Description:
This function starts the initialization process of DSL HAL
Parameters:
priv, pointer to driver's private pointer.
Returns:
0.

5.2.2 void tn7dsl_exit (void)

Description:
This function shuts down the DSL subsystem.
Parameters:
void.
Returns:
None.

5.2.3 static tn7dsl_proc_stats (char* buf, char **start, off_t offset, int count,int *eof, void *data)

Description:
Linux proc file to provide DSL/SAR statistical information.

5.2.4 static tn7dsl_proc_stats (char* buf, char **start, off_t offset, int count,int *eof, void *data)

Description:
Linux proc file to provide DSL modem training information.

5.2.5 int tn7dsl_get_dsl_version(char *pVer)

Description:
This function gets the DSL HAL version
Parameters:
pVer, pointer to pre-allocated memory which version information will return.
Returns:
0.

5.2.6 int tn7dsl_get_dsp_version(char *pVer)

Description:
This function gets the DSP datapump version
Parameters:
pVer, pointer to pre-allocated memory which version information will return.
Returns:
0.

5.2.7 static int tn7dsl_get_modulation(void)

Description:
This function gets initial DSL training mode from Adam2 environment space.
Parameters:

void.

Returns:

0, always succeeds.

5.2.8 static void tn7dsl_set_modulation(void* data)

Description:

This function sets DSL training mode according to input.

Parameters:

data, DSL training mode string.

Returns:

None.

5.2.9 static void tn7dsl_chng_modulation(void* data)

Description:

This function changes DSL training mode according to input and starts training with the new mode.

Parameters:

data, DSL training mode string.

Returns:

None.

5.2.10 static int tn7dsl_process_oam_string(int *type)

Description:

This function purses the string from dslmod call for oam cell generation.

Parameters:

**type*, returns the type of oam cell.

Returns:

channel number to send oam cell.

None.

5.2.11 static int dslmod_sysctl(ctl_table *ctl, int write, struct file * filp, void *buffer, size_t *lenp)

Description:

This function processes proc sys file /proc/sys/dev/dslmod call commands .

Returns:

0, failed.

> 0, succeeds.

5.2.12 void dslmod_sysctl_register(void)

Description:

This function registers proc sys file /proc/sys/dev/dslmod.

Parameters:

void.

Returns:

None.

5.2.13 void dslmod_sysctl_unregister(void)

Description:

This function de-registers proc sys file /proc/sys/dev/dslmod.

Parameters:

void.
Returns:
None.

5.2.14 int tn7dsl_handle_interrupt(void)

Description:
This function passes DSL interrupt to DSL HAL and set connection status and LED accordingly.
Parameters:
Returns:
0.

5.2.15 int shim_osLoadFWImage(char *ptr)

Description:
This function reads DSP firmware from flash file system and copy it to a pre-allocated memory loction so DSL HAL can load it into DSP.
Parameters:
ptr, pointer to a pre-allocated memory location.
Returns:
0.

5.2.16 void shim_osClockWait(int val)

Description:
Wait function need by DSL HAL.
Parameters:
val, time to wait .
Returns:
0.

5.3 APIs from turbodsl.c

5.3.1 int turbodsl_init(void)

Description:
This function initializes variables for the turbo DSL application.
Parameters:
void.
Returns:
0, always succeeds.

5.3.2 int turbodsl_uninit(int vpi, int vci)

Description:
This function sets pointer to a per channel structure to NULL.
Parameters:
vpi, VPI value of a channel.
vci, VCI value of a channel.
Returns:
0, always succeeds.

5.3.3 int turbodsl_create_tx_queue(void *vcc, void *priv, short vpi, int vci)

Description:
This function creates and allocates per PVC resources for an open channel.
Parameters:
vcc, pointer to OS per PVC context.
priv, pointer to device private structure.
vpi, VPI value of a channel.
vci, VCI value of a channel.
Returns:
0, if succeeds.
1, if failed.
Possible causes of error could be failure to allocate memory.

5.3.4 int turbodsl_memory_compare(unsigned char *pIn, unsigned char *pOut, unsigned int len)

Description:
This function compares values of given memories.
Parameters:
pIn, pointer to first memory location.
pOut, pointer to second memory location.
len, length to compare.
Returns:
0, if not the same
1, if the same.

5.3.5 int turbodsl_check_aal5_encap_type(unsigned char *pData)

Description:
This function returns AAL5 encapsulation type for a given network packet.
Parameters:
pData, pointer to beginning of data buffer of a packet.
Returns:
0, AAL5_ENCAP_PPP_LLC.
1, AAL5_ENCAP_PPP_VCMUX.
2, AAL5_ENCAP_RFC2684_LLC.

5.3.6 int turbodsl_check_priority_type(unsigned char *pData, short vpi, int vci, int length)

Description:
This function checks whether a network packet needs to be prioritized base on the type.
Parameters:
pData, pointer to beginning of data buffer of a packet.
vpi, VPI value to send the packet with.
vci, VCI value to send the packet with.
length, data length of the packet.
Returns:
0, not to be prioritized.
1, to be prioritized.

5.3.7 int turbodsl_queue_tx_packet(short vpi, int vci, void *skb, int queue)

Description:
This function queues a packet to either normal queue or priority queue base on input value.

Parameters:
vpi, VPI value to send the packet with.
vci, VCI value to send the packet with.
skb, network packet to be queued.
queue, 0 queues to normal queue; 1 queues to priority queue.
Returns:
0, succeeds.
1, failed.
Possible causes of error could be failure to find a channel with correct VCI or VPI value.

5.3.8 int turbodsl_send_tx_packet(void)

Description:
This function sends packets to the hardware based queue type. It will service the packets in priority queue before it services the packets in normal queue.
Parameters:
void.
Returns:
0, always succeeds.

5.3.9 int turbodsl_reset_pending_flag(short vpi, int vci)

Description:
This function resets the pending flag of a PVC when hardware completes the packet sending.
Parameters:
vpi, VPI value of a packet sent.
vci, VCI value of a packet sent.
Returns:
0, succeeds.
1, failed.

5.3.10 int os_queue_packet_to_hardware(void *vcc1, void *skb1)

Description:
This function re-maps OS specific routine to send packet to hardware.
Parameters:
vcc1, pointer to OS context.
skb1, pointer to OS packet.
Returns:
0, succeeds.
1, failed.

5.3.11 void os_free_packet(void *pVc, void *pDev, void *pPacket)

Description:
This function re-maps OS provided packet free routine .
Parameters:
pVc, pointer to OS specific per PVC context.
pDev, pointer to OS specific device context.
pPacket, pointer to OS specific network packet.
Returns:
None.

5.3.12 void osAcquireSpinLock(OS_SPIN_LOCK *pLock)

Description:
This function re-maps OS provided spin lock acquiring routine.
Parameters:
pLock, pointer to OS specific spin lock structure.
Returns:
None.

5.3.13 void osReleaseSpinLock (OS_SPIN_LOCK *pLock)

Description:
This function re-maps OS provided spin lock releasing routine.
Parameters:
pLock, pointer to OS specific spin lock structure.
Returns:
None.

5.3.14 void osAllocateSpinLock (OS_SPIN_LOCK *pLock)

Description:
This function re-maps OS provided spin lock initialization routine.
Parameters:
pLock, pointer to OS specific spin lock structure.
Returns:
None.

5.4 APIs from tn7sar.c

5.4.1 int tn7sar_init_module(OS_FUNCTIONS *os_funcs)

Description:
Init SAR HAL OS functions.
Parameters:
os_funcs, pointer to SAR HAL OS call back function.
Returns:
0.

5.4.2 static void tn7sar_init_dev_parm(void)

Description:
This function init default SAR HAL device parameter.
Parameters:

Returns:
None.

5.4.3 int tn7sar_get_stats(void *priv1)

Description:
This function retrieves CRC errors from SAR HAL.
Parameters:
priv1, pointer to driver's private structure.

Returns:
0.

5.4.4 int tn7sar_init(struct atm_dev *dev, Tn7AtmPrivate *priv)

Description:
This function starts initialization of SAR HAL.
Parameters:
dev, pointer to ATM device structure.
priv, pointer to driver's private structure.
Returns:
0, success.
1, Failed.

5.4.5 static int tn7sar_atm_header(int vpi, int vci)

Description:
This function generates the ATM header required by SAR firmware.
Parameters:
vpi, VPI value.
vci, VCI value.
Returns:
ATM header.

5.4.6 int tn7sar_activate_vc(Tn7AtmPrivate *priv, short vpi, int vci, int pcr, int scr, int mbs, int cdvt, int chan, int qos)

Description:
This function calls SAR to setup a channel.
Parameters:
priv, pointer to driver's private structure.
vpi, VPI value.
vci, VCI value.
pcr, peak cell rate for CBR and VBR.
scr, sustained cell rate for VBR.
mbs, minimum burst size.
cdvt, cell delay variation tolerance.
chan, channel number to activate.
qos, Quality of service.
Returns:
0, success.
1, Failed.

5.4.7 int tn7sar_send_packet(Tn7AtmPrivate *priv, int chan, void *new_skb, void *data,unsigned int len, int priority)

Description:
This function calls SAR HAL to send a packet.
Parameters:
priv, pointer to driver's private structure.
chan, channel number to send packet.
new_skb, pointer to a packet.

data, pointer to data buffer of the packet.
len, length of the packet.
Returns:
0, success.
1, Failed.

5.4.8 int tn7sar_handle_interrupt(struct atm_dev *dev, Tn7AtmPrivate *priv)

Description:
This function calls SAR HAL's interrupt handling routine.
Parameters:
dev, pointer to ATM device structure.
priv, pointer to driver's private structure.
Returns:
0, success.
1, Failed.

5.4.9 int tn7sar_deactivate_vc(Tn7AtmPrivate *priv, int chan)

Description:
This function calls SAR HAL to tear down a channel.
Parameters:
priv, pointer to driver's private structure.
chan, channel number to tear down
Returns:
0.

5.4.10 void tn7sar_exit(struct atm_dev *dev, Tn7AtmPrivate *priv)

Description:
This function de-initialized the SAR.
Parameters:
dev, pointer to ATM device structure.
priv, pointer to driver's private structure.
Returns:
None.

5.4.11 int tn7sar_oam_generation(void *privContext, int chan, int type)

Description:
This function calls SAR HAL to generate far end loop back OAM cell.
Parameters:
privContext, pointer to driver's private structure.
chan, channel number to send OAM cell.
type, type of OAM cell to send.
Returns:
0.

5.4.12 int tn7sar_proc_sar_stat (char* buf, char **start, off_t offset, int count,int *eof, void *data)

Description:

Linux proc file to provide SAR HAL statistical information

5.4.13 others

There other files in this modules which are required OS functions for SAR HAL. Please refer to "*Texas Instruments Inc., Communication Processor Hardware Abstraction Layer (CPHAL) API Reference Manual*." for details. These files are listed here for completeness.

tn7sar_control;
tn7sar_critical_on;
tn7sar_critical_off;
tn7sar_data_invalidate;
tn7sar_data_writeback;
tn7sar_find_device;
tn7sar_get_device_parm_uint;
tn7sar_get_device_parm_value;
tn7sar_free;
tn7sar_free_buffer;
tn7sar_free_dev;
tn7sar_free_dma_xfer;
tn7sar_sarhal_isr_register;
tn7sar_isr_unregister;
tn7sar_malloc;
tn7sar_malloc_rxbuffer;
tn7sar_malloc_dev;
tn7sar_malloc_dma_xfer;
tn7sar_memset;
tn7sar_printf;
tn7sar_receive;
tn7sar_send_complete;
tn7sar_strtoul;
tn7sar_teardown_complete;