IT913X Linux SDK Programmer's Guide

Revision Log:

Revision	Date	Author	Remarks
1.0	2016/05/19	**Jennifer**	**First Release**
1.1	2016/07/07	**Jennifer**	**Change the way to set dynamic URB buffer size**
1.2	2016/07/25	**Jennifer**	**Modify some detail information in Figure 2**
1.3	2016/08/12	**Jennifer**	**Add a new Board_id for Downconverter RFFC2071**
1.4	2016/08/30	**Jennifer**	**Add Inverse Null Packet Filter and add set PID filter**
1.5	2016/09/22	**Jennifer**	**Add remind information for changing URBSIZE**
1.6	2016/10/25	**Jennifer**	**Add Find 2.4G free band**
1.7	2016/11/04	**Jennifer**	**Add "TS mode" for EEPROM**

ITE Tech. Inc.

Easy HD Expressway

Table of Contents

1 Introduction

This document describes how to use and program IT913X (also known as Omega) Digital TV modulator USB Dongle under Linux platforms. This document is especially for Linux software programmers. Windows developers please refer to IT913X Windows SDK Programmer's Guide. IT913X series include IT9135 and IT9137.

Figure 1: IT9135 USB Dongle

1.1. Control IT913X

A host CPU can control IT913X through either I^2C or USB bus. This document describes how to control and how to let video transport streams received by IT913X via USB bus.

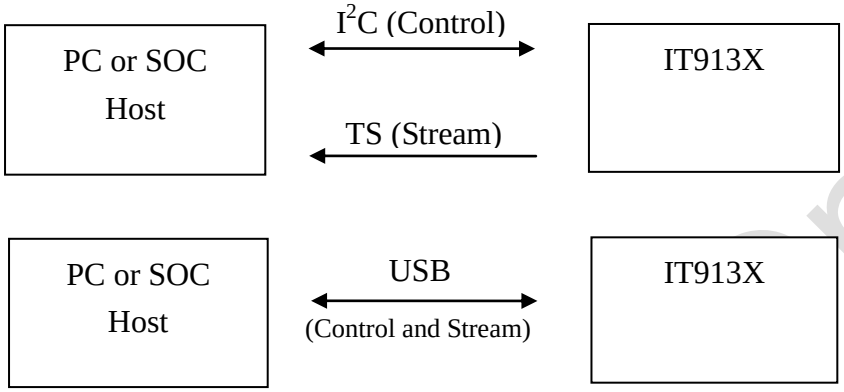


Figure 2: IT913X controlled by a Hos

1.2. IT913X Linux application software Hierarchy

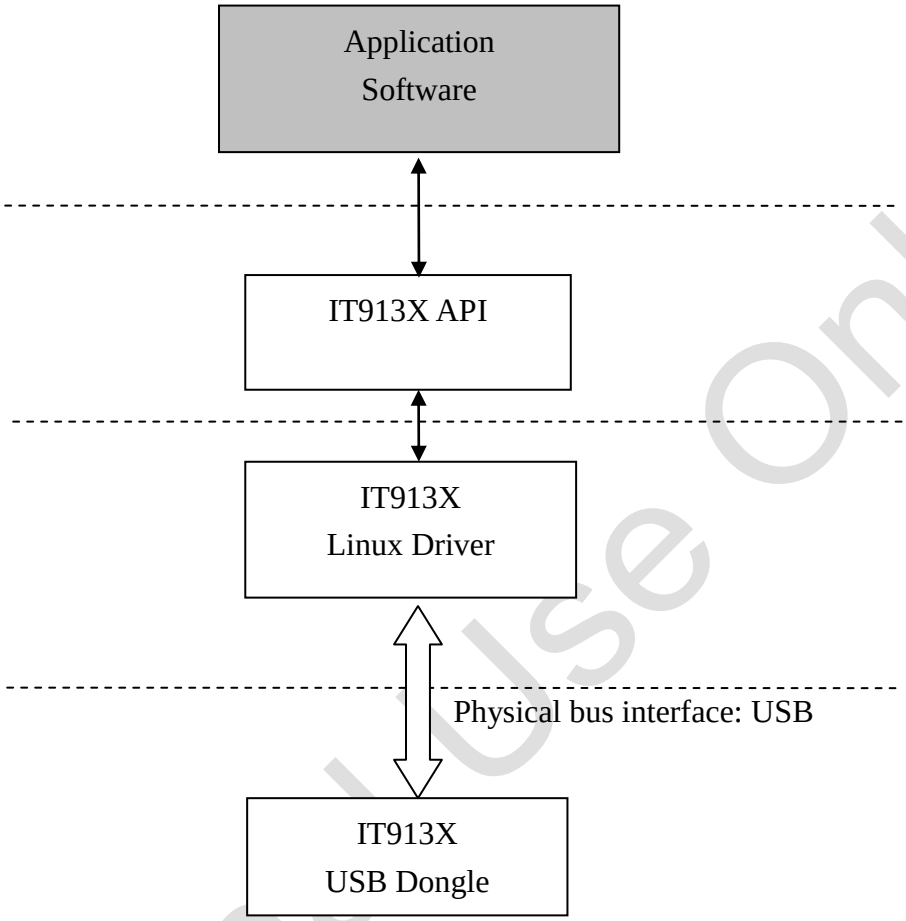


Figure 3: IT913X Software Hierarchy

2 Package Contents

2.1 IT913X Driver source code

The Linux driver source code is put under "**it913x_driver**" folder.

2.2 IT913X Testkit source code

The sample testkit source codes can be found in "**it913x_linux/it913x_testkit**" folder.

The sample testkit can only read and handle 188-byte TS files, while do not support 204-byte format TS files.

3 Launch Testkit

3.1 Environment Setup

Step 1: Install Linux driver and plug in IT913X USB dongle properly

Step 2: Users should compile source code first - make IT913x_testkit test tool

Step 3: Execute it913x_testkit test tool

For more detail information of driver and testkit installation steps, please refer to **README** file (This file put under "**it913x_linux**" folder).

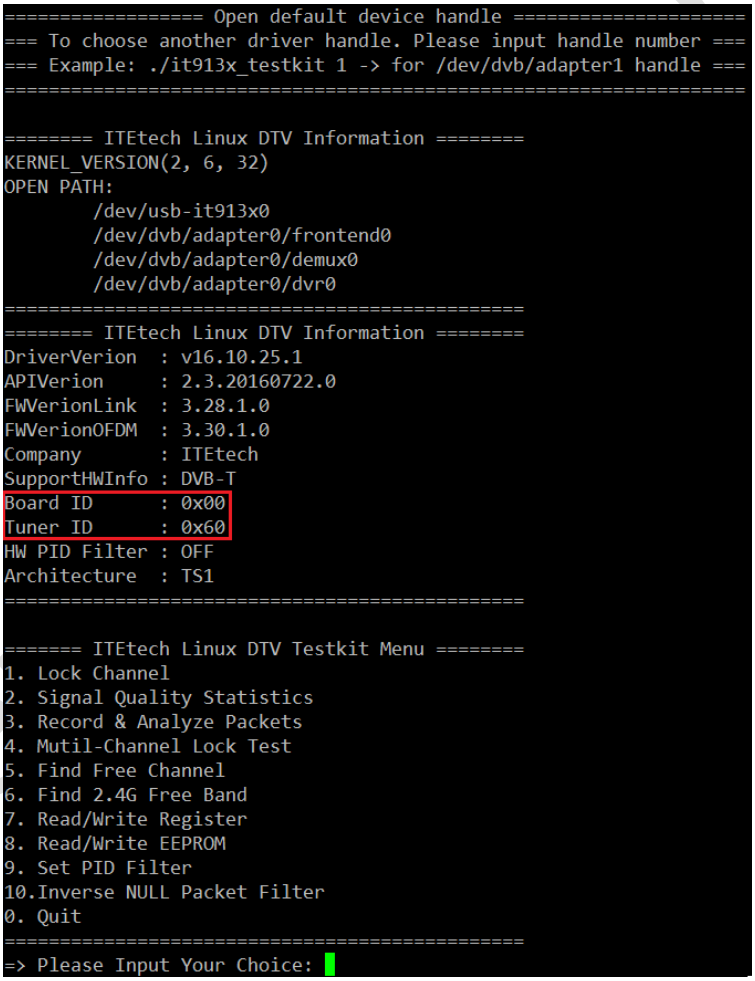


Figure 4: Initialization Messages of testkit

By default, testkit would directly bind to the first device file (device name = usb-it913x0). When users execute testkit, some information would be demonstrated as Figure 4 shown. These important messages inform users about driver's information which includes: **Driver Version**, **Tuner ID**, and **Board ID. (Table 1** shows the detail information of different "Board ID".)

1. Driver Version: Let users know the current driver's version.

2. Tuner ID: Indicate this dongle using which firmware script in driver code.

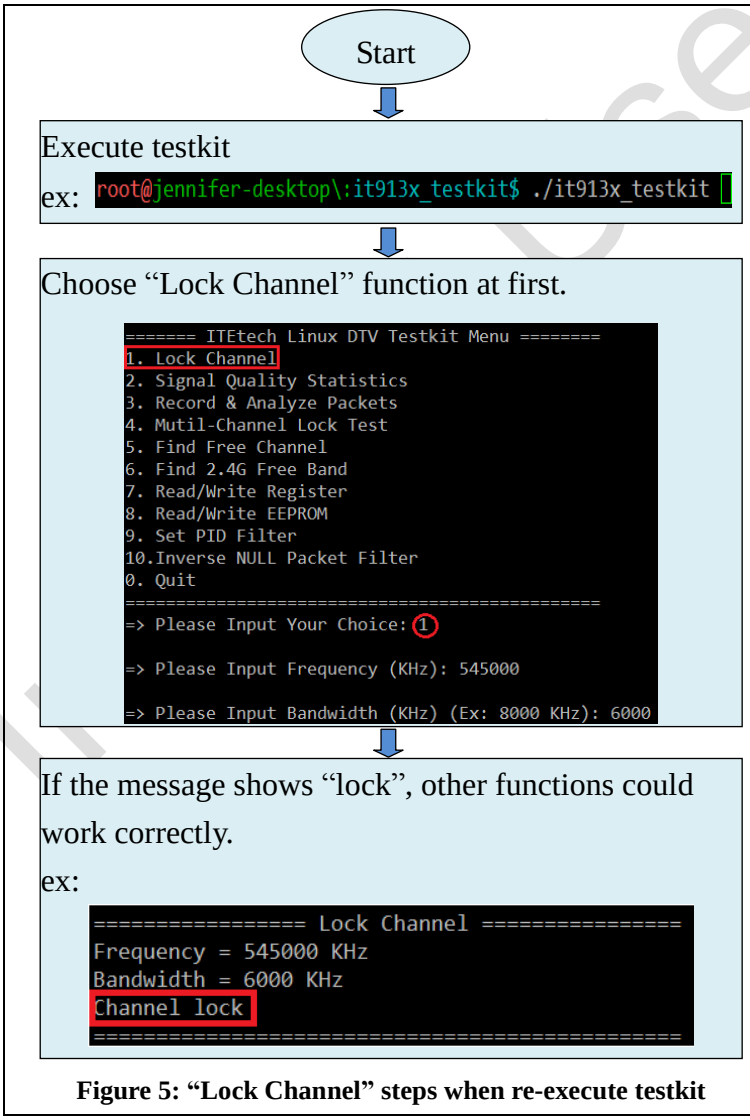
3. Board ID: Indicate this dongle has Down Converter or not.

		Has Down Converter	Down Converter model number
Board ID	00(d) = 0x00(h)		
	49(d) = 0x31(h)	✔	ADRF6655
	50(d) = 0x32(h)	✔	RFFC2072
	51(d) = 0x33(h)	✔	RFFC2071

Table 1: Detail information of different Board ID

3.2 Brief Using Steps

3.2.1 Scenario 1: Lock Channel



Please follow the steps in Figure 5 in left column, whenever users re-execute testkit.

No matter when users re-execute testkit, users **should choose "Lock Channel" function at first place**. Not until the "Lock" message is showing up, should users use other function in testkit menu.

Figure 5: "Lock Channel" steps when re-execute testkit

3.2.2 Scenario 2: Signal Quality Statistics

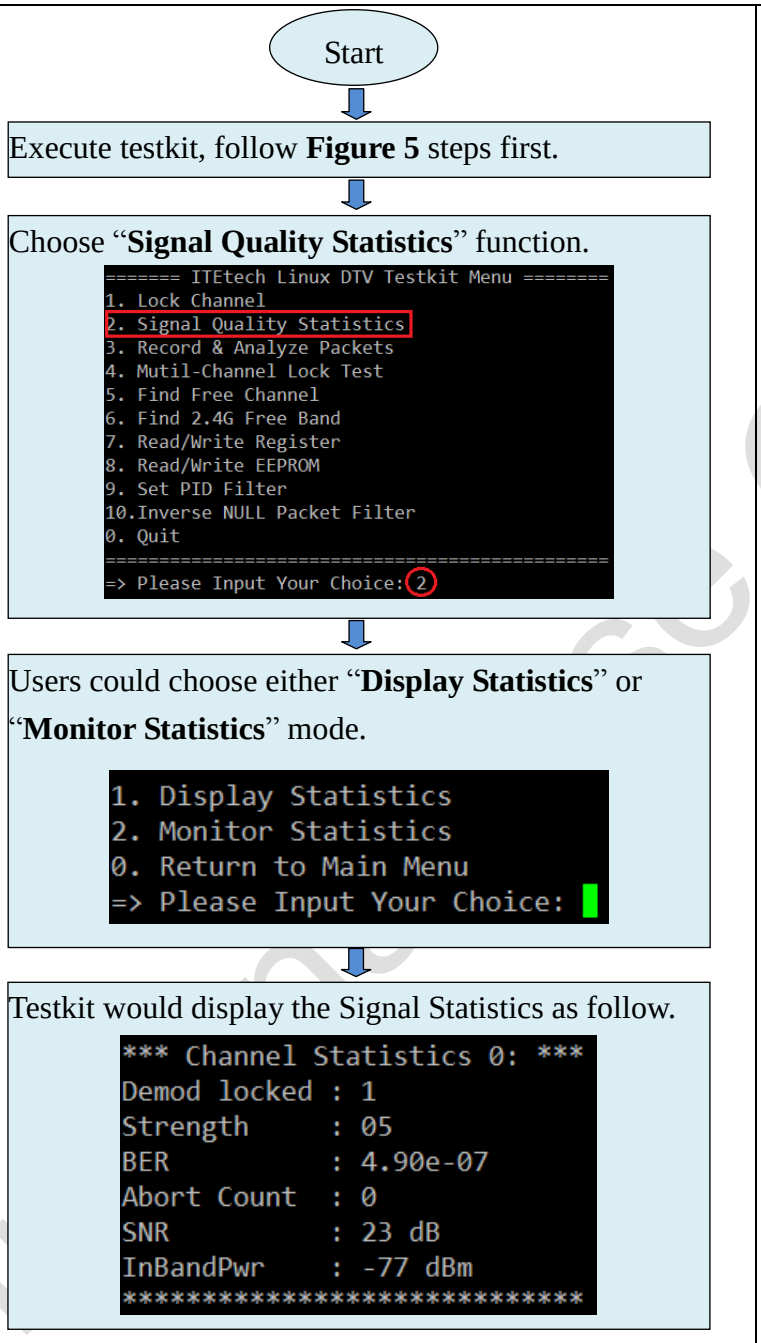


Figure 6: "Signal Quality Statistics" using steps

3.2.3 Scenario 3: Record & Analyze Packets

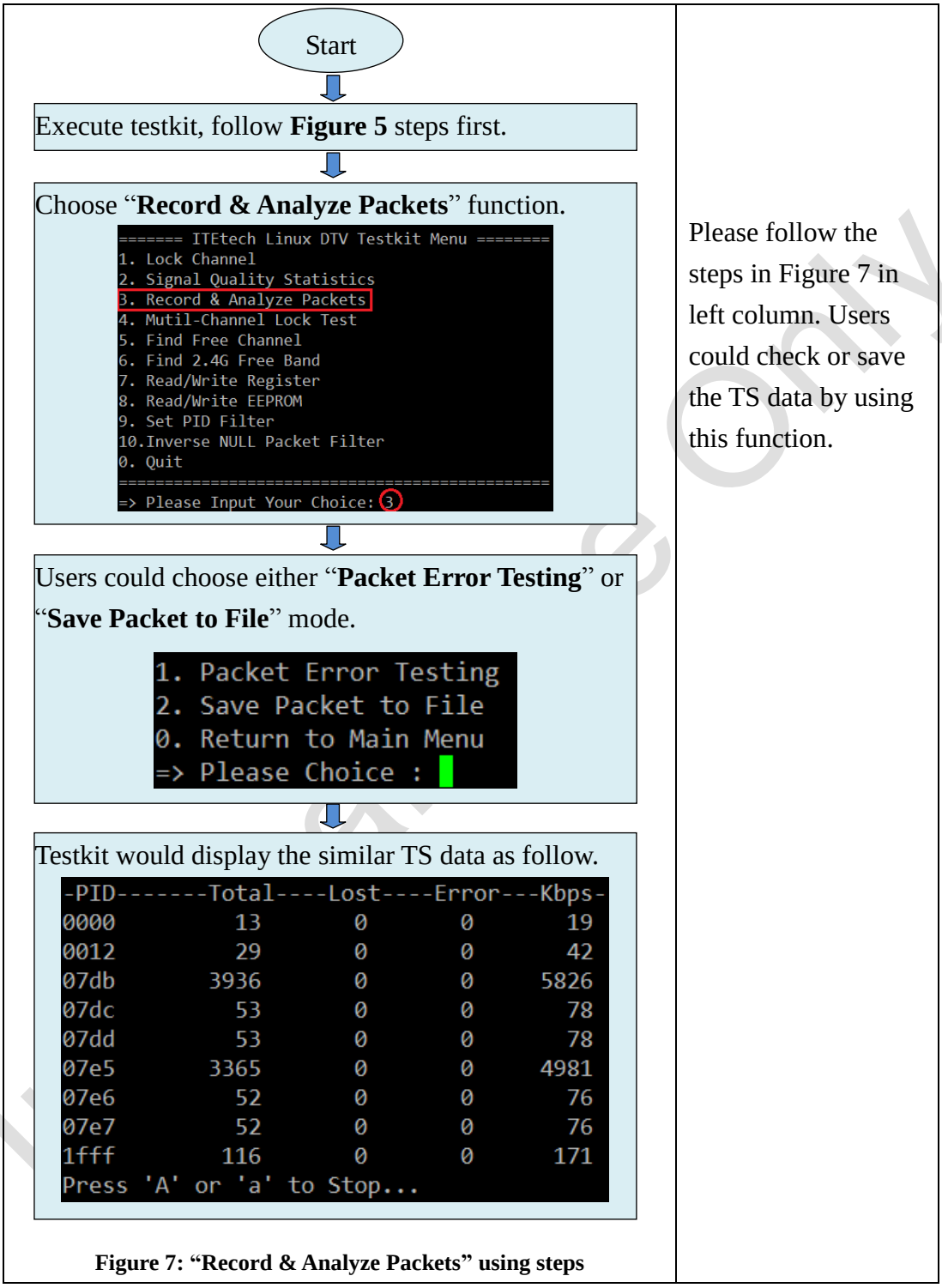


Figure 7: "Record & Analyze Packets" using steps

3.2.4 Scenario 4: Set PID Filter

How to filter some particular PIDs (Program IDs) packets?

(Figure 8 is an example)

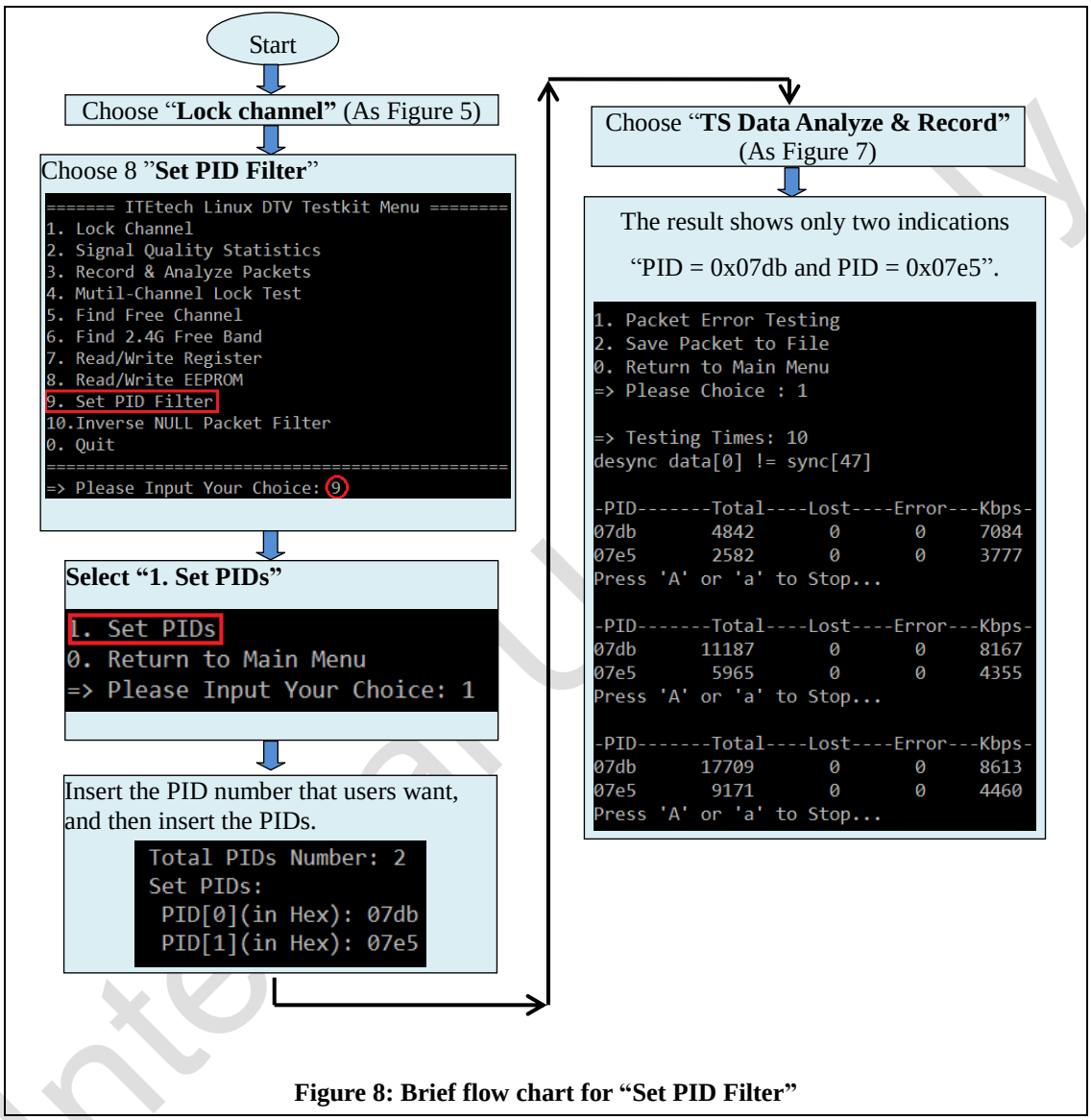


Figure 8: Brief flow chart for "Set PID Filter"

3.2.5 Scenario 5: Set Null Packet Filter

How to filter out Null packet (PID: 0x1FFF)?

(Figure 9 is an example)

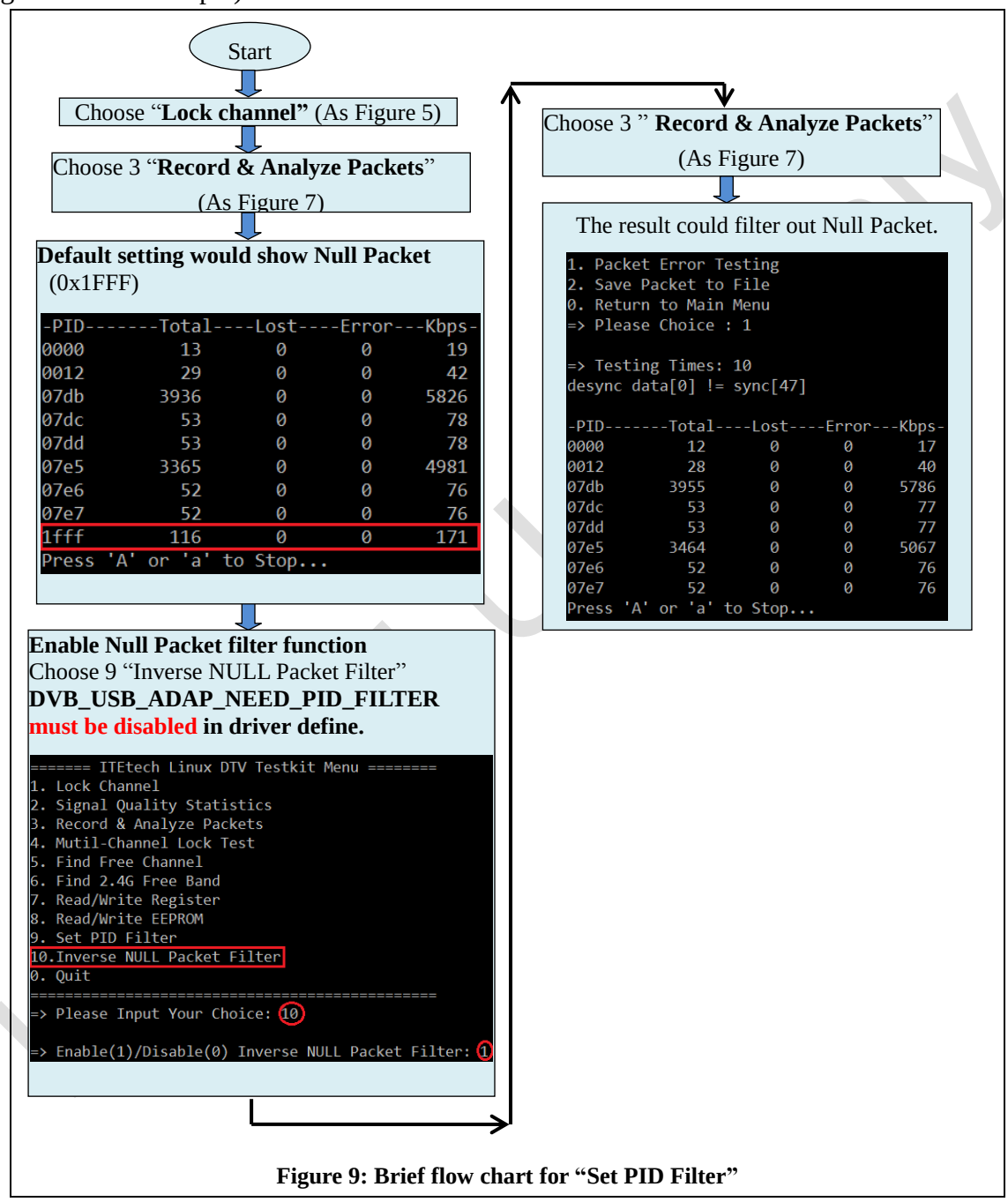


Figure 9: Brief flow chart for "Set PID Filter"

3.3 Testkit Usage

3.3.1 Lock Channel

User Selection	Select 1 "Lock Channel"
Function Description	This function would make the usb device acquire a specific frequency between the assigned bandwidth.
Calling Function	int **Omega_Lock_Channel** (int frontend);
Function Parameters	(1) **int frontend** This is "/dev/dvb/adapter0/frontend0" device file's control handle. `/dev/char/212:3 -> ../dvb/adapter0/frontend0`

Operation Manual	Figure 10 illustrates an example:
	If users want this device to acquire the frequency 545MHz between 6MHz bandwidth. The following two steps should be done by users:
	Step 1: Insert 545000 in "Input Frequency" blank,
	Step 2: Insert 6000 in "Input Bandwidth".
	If the particular frequency could be obtained by IT913X dongle successfully, "**Channel lock**" message would be displayed; on the contrary "**Channel unlock**" message would be shown in notification blank.
	`=> Please Input Frequency (KHz): 545000`
	`=> Please Input Bandwidth (KHz) (Ex: 8000 KHz): 6000`
	`================ Lock Channel ================`
	`Frequency = 545000 KHz`
	`Bandwidth = 6000 KHz`
	`Channel lock`
	`==============================================`

Figure 10: Lock Channel parameters setting

3.3.2 Signal Quality Statistics

User Selection	Select 2 "Signal Quality Statistics"
Function Description	This function would display some signal information about the lock frequency.
Calling Function	void **Omega_Get_Statistic_Menu** (int frontend, int handle);

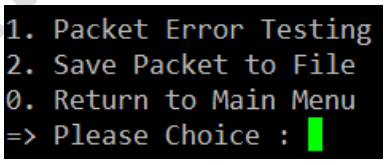
Function Parameters	(1) **int frontend**
	This is "/dev/dvb/adapter0/frontend0" device file's control handle.
	`/dev/char/212:3 -> ../dvb/adapter0/frontend0`
	(2) **int handle**
	This is "/dev/usb-it913x0" device file's control handle.
	`/dev/char/180:191 -> ../usb-it913x0`

Operation Manual	Users could see two choices under this selection: (As Figure 11 shows)
	(1) **Display Statistics** and (2) **Monitor Statistics**
	`1. Display Statistics`
	`2. Monitor Statistics`
	`0. Return to Main Menu`
	`=> Please Input Your Choice:`
	Figure 11: Two modes under Signal Quality Statistics selection
	All these two choices show the current signal statistics of this dongle, as Figure 12. The mainly different is that "**Monitor Statistics**" mode lets users monitor signal status under a specified number of times.
	`*** Channel Statistics 0: ***`
	`Demod locked : 1`
	`Strength : 05`
	`BER : 4.90e-07`
	`Abort Count : 0`
	`SNR : 23 dB`
	`InBandPwr : -77 dBm`
	`**********`
	Figure 12: The detail of channel statistics

3.3.3 Record & Analyze Packets

User Selection	Select 3 "Record & Analyze Packets"
Function Description	This function displays or saves different PID (Packet ID) information. (Data information includes total receiving packet number, lost packet number, error packet number, and data bitrate.) Users **should choose "Lock Channel" function first**. After the particular frequency is acquired successfully, users should use this function.
Calling Function	void **Omega_Get_Packet_Menu** (int *demux, int dvr);

Function Parameters	(1) **int *demux**
	This is "/dev/dvb/adapter0/demux0" device file's control pointer handle.
	`/dev/char/212:0 -> ../dvb/adapter0/demux0`
	(2) **int dvr**
	This is "/dev/dvb/adapter0/dvr0" device file's control handle.
	`/dev/char/212:1 -> ../dvb/adapter0/dvr0`

Operation Manual	Users could see two choices under this selection: (As Figure 13 shows) (1) **Packet Error Testing** and (2) **Save Packet to File**  **Figure 13: Two modes under "Record & Analyze Packets" selection** All these two choices would analyze the TS data. After analyzing the TS data, testkit would display or save the data. (As Figure 14 illustrates.) (1) **Packet Error Testing** In "**Packet Error Testing**" mode, users could set the monitor times. Testkit would show all the PIDs data separately under a specified number of times. (2) **Save Packet to File** In "**Save Packet to File**" mode, testkit would ask user "What size of data do users want to save?" and then data would be saved to a DVB.ts file. This DVB.ts file could use some media player (such as vlc) to display the recording data.

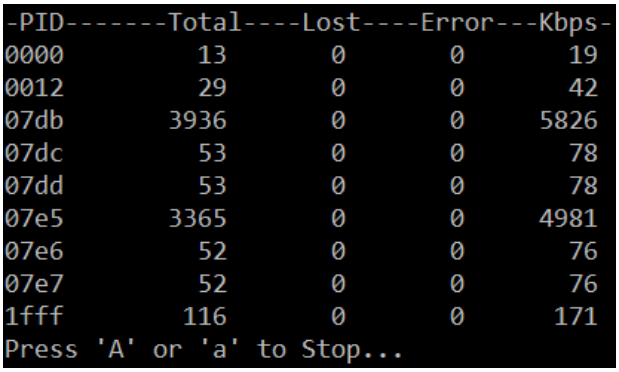


Figure 14: Testkit would show all the PIDs data separately

3.3.4 Mutil-Channel Lock Test

User Selection	Select 4 "Mutil-Channel Lock Test"
Function Description	This function helps users to find some locked frequencies in a particular range.
Calling Function	void **Omega_Lock_Multi_Channel** (int frontend, int handle);

Function Parameters	(1) **int frontend**
	This is "/dev/dvb/adapter**0**/frontend0" device file's control handle.
	`/dev/char/212:3 -> ../dvb/adapter0/frontend0`
	(2) **int handle**
	This is "/dev/usb-it913x0" device file's control handle.
	`/dev/char/180:191 -> ../usb-it913x0`

Operation Manual	**(1) How to use**
	Figure 15 illustrates an example:
	If users want to find which frequencies were be used, the following step should be followed:
	Step 1: Insert 530000 in "Input Start Frequency" blank,
	Step 2: Insert 545000 in "Input End Frequency".
	Step 3: Insert 6000 in "Input Bandwidth".
	`=> Please Input Start Frequency (KHz): 530000`
	`=> Please Input End Frequency (KHz): 545000`
	`=> Please Input Bandwidth (KHz) (Ex: 8000 KHz): 6000`
	Figure 15: The usage of "Mutil-Channel Lock Test"
	(2) Messages show during scanning
	During scanning, users would see some locked frequency signal information. Users could see signal statistics from "**Start Frequency** "till "(**Start Frequency + n * Bandwidth**) < **End Frequency**."
	(n is a positive integer, in our example, the scanning frequencies are 530000, 536000, and 542000.)
	Figure 16 is an example showing the messages during scanning:

```
Scan 530000 KHz
*** Channel Statistics: ***
Frequency    : 530000 KHz
Demod locked : 1
Strength     : 00
BER          : 3.87e-02
Abort Count  : 0
SNR          : 00 dB
InBandPwr    : -81 dBm
**************************
Scan 536000 KHz
*** Channel Statistics: ***
Frequency    : 536000 KHz
Demod locked : 1
Strength     : 00
BER          : 3.87e-02
Abort Count  : 0
SNR          : 00 dB
InBandPwr    : -81 dBm
**************************
Scan 542000 KHz
*** Channel Statistics: ***
Frequency    : 542000 KHz
Demod locked : 1
Strength     : 02
BER          : 3.87e-02
Abort Count  : 0
SNR          : 00 dB
InBandPwr    : -81 dBm
**************************
```

Figure 16: An example: Some messages shows up during scanning

(3) Final result

The final result message would tell users how many channels have been locked. (As Figure 17 shown.)

```
================== Analysis ====================
Frequency 530000(KHz) ~  545000(KHz)
Scan Channel : 3 channels Locked
================================================
```

Figure 17: Final "Mutil-Channel Lock Test" result message

3.3.5 Find Free Channel

User Selection	Select 5 "Find Free Channel"
Function Description	This function let users find some unused frequencies.
Operation Manual	void **Omega_Find_Free_Channel** (int frontend, int handle);

Function Parameters	(1) **int frontend**
	This is "/dev/dvb/adapter0/frontend0" device file's control handle.
	`/dev/char/212:3 -> ../dvb/adapter0/frontend0`
	(2) **int handle**
	This is "/dev/usb-it913x0" device file's control handle.
	`/dev/char/180:191 -> ../usb-it913x0`

Operation Manual	**(1) How to use**
	Figure 18 illustrates an example:
	If users want find which frequencies were not be used:
	Step 1: Insert 545000 in "Input Start Frequency" blank,
	Step 2: Insert 555000 in "Input End Frequency".
	Step 3: Insert 6000 in "Input Bandwidth".
	Step 4: Insert 0 in "Scan Scale".
	(If Users want to get much more accurate information, the recommend "Scan Scale" should be followed.)
	In this example, we use the dongle which supports 900MHz; therefore, Scan Scale "0" should be selected.
	`=> Please Input Start Frequency (KHz): 545000`
	`=> Please Input End Frequency (KHz): 555000`
	`=> Please Input Bandwidth (KHz) (Ex: 8000 KHz): 6000`
	`=> Please Input Scan Scale:`
	`0: 1 MHz (recommend for 900MHz)`
	`1: 1/2 bandwidth (recommend for 2.4G)`
	`Select:0`
	Figure 18: The usage of "Find Free Channel"
	(2) Messages show during scanning
	During scanning, users would see the messages as below Figure 19. Users could see the RF power from each scanning frequency, and the final "Free Channel RF Power" will show "**Input_Start_Frequency +**

Input_Bandwidth/2" to "**Input_End_Frequency - Input_Bandwidth/2**" (In our example will show results from 548000KHz to 552000KHz).

```
Scan 542000 KHz:
        Frequency = 542000 KHz,  RF power = -81 dBm
Scan 543000 KHz:
        Frequency = 543000 KHz,  RF power = -79 dBm
Scan 544000 KHz:
        Frequency = 544000 KHz,  RF power = -78 dBm
Scan 545000 KHz:
        Frequency = 545000 KHz,  RF power = -77 dBm
Scan 546000 KHz:
        Frequency = 546000 KHz,  RF power = -77 dBm
Scan 547000 KHz:
        Frequency = 547000 KHz,  RF power = -78 dBm
Scan 548000 KHz:
        Frequency = 548000 KHz,  RF power = -80 dBm
Scan 549000 KHz:
        Frequency = 549000 KHz,  RF power = -83 dBm
Scan 550000 KHz:
        Frequency = 550000 KHz,  RF power = -87 dBm
Scan 551000 KHz:
        Frequency = 551000 KHz,  RF power = -92 dBm
Scan 552000 KHz:
        Frequency = 552000 KHz,  RF power = -88 dBm
Scan 553000 KHz:
        Frequency = 553000 KHz,  RF power = -84 dBm
Scan 554000 KHz:
        Frequency = 554000 KHz,  RF power = -81 dBm
Scan 555000 KHz:
        Frequency = 555000 KHz,  RF power = -78 dBm
Scan 556000 KHz:
        Frequency = 556000 KHz,  RF power = -77 dBm
Scan 557000 KHz:
        Frequency = 557000 KHz,  RF power = -77 dBm
Scan 558000 KHz:
        Frequency = 558000 KHz,  RF power = -77 dBm


---------------- Free Channel RF Power: ----------------
Frequency: 548000 KHz, RF Local Max: -80 dBm
Frequency: 549000 KHz, RF Local Max: -83 dBm
Frequency: 550000 KHz, RF Local Max: -87 dBm
Frequency: 551000 KHz, RF Local Max: -92 dBm
Frequency: 552000 KHz, RF Local Max: -88 dBm
------------------------------------------------
```

Figure 19: An example: Some messages shows up during scanning

(3) Final result

Testkit will using the scanning results to find the best free frequency and then users could see the similar message as Figure 20.

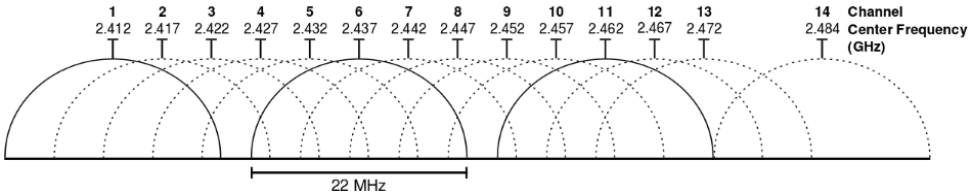
```
=========== Best Free Frequency : =============
         Frequency    : 551000 KHz
         InBandPwr    : -92 dBm
================================================
```

Figure 20: Messages indicate the "Best Free Frequency"

3.3.6 Find 2.4G Free Band

User Selection	Select 6 "Find 2.4G Free Band"
Function Description	This function let users find the free band under 2.4GHz WLAN channels.
Operation Manual	void **Omega_Find_2_4G_Free_Band** (int frontend, int handle);

Function Parameters	(1) **int frontend**
	This is "/dev/dvb/adapter0/frontend0" device file's control handle.
	`/dev/char/212:3 -> ../dvb/adapter0/frontend0`
	(2) **int handle**
	This is "/dev/usb-it913x0" device file's control handle.
	`/dev/char/180:191 -> ../usb-it913x0`

Operation Manual	**(1) How to use**
	Detail descriptions:
	Under this function, users could see the following two selections as Figure 21.
	`=> Please select the Free Band width you want :`
	`0: 11 chan`
	`1: 13 chan`
	`Select:`
	Figure 21: Two selections under this function.
	0:11 chan:
	We would find the free band between channel 1 to channel 11, as Figure 22 displays.
	1:13 chan:
	We would find the free band between channel 1 to channel 13, as Figure 22 displays.
	 
	Figure 22: Graphical representation of 2.4 GHz band channels overlapping (source from : https://en.wikipedia.org/wiki/List_of_WLAN_channels)

(2) Final result

Testkit would using the scanning results to find the best free band and then users could see the similar message as Figure 23.

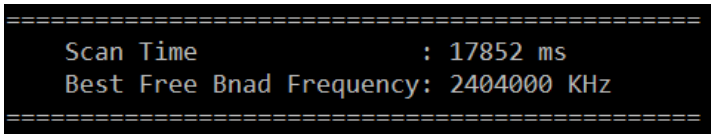


Figure 23: The final results would display the "Best Free Band Frequency" and requiring scan time.

3.3.7 Read/Write Register

User Selection	Select 7 "Read/Write Register"
Function Description	Users could modify or read omega registers by this function. If not particular purpose **users should not only avoid using this function** but also **should use this function carefully**. (Due to some values of registers could not be modified.)
Calling Function	void **Omega_RW_Register** (int handle);
Function Parameters	(1) **int handle** This is "/dev/usb-it913x0" device file's control handle. `/dev/char/180:191 -> ../usb-it913x0`
Operation Manual	**(1) How to use** Figure 24 and Figure 25 illustrate two examples of Read/Write register: If users want read/write omega's register: **Step 1**: Choose **"0" as read** and **"1" as write**. **Step 2**: Choose processor type – "LINK" or "OFDM". **Step 3**: Insert the omega register address in Hexadecimal value. **Step 4**: If function works successfully, some messages would displayed -t link processor, register address and the register value.

```
=> Please Choose Read/Write Register (0: Read, 1: Write): 0

=> Please Choose LINK or OFDM (0: LINK, 1: OFDM): 0

=> Please Enter Read/Write Register Address (Hex): d830
Read LINK Address[0xD830] Value[0x0] success.
```

Figure 24: An example of reading omega register

```
=> Please Choose Read/Write Register (0: Read, 1: Write): 1

=> Please Choose LINK or OFDM (0: LINK, 1: OFDM): 0

=> Please Enter Read/Write Register Address (Hex): d830

=> Please Enter Write Value (Hex): 1
Write LINK Address[0xD830] Value[0x1] success.
```

Figure 25: An example of writing omega register

3.3.8 Read/Write EEPROM

User Selection	Select 8 "Read/Write EEPROM "
Function Description	This function helps users to get or set some EEPROM items of the device. (Could be used EEPROM items: **Serial Number**, **TS mode**, **Board ID**, and **Tuner ID**)
Calling Function	void **Omega_EEPROM_raw_data_editor** (int handle);
Function Parameters	(1) **int handle** This is "/dev/usb-it913x0" device file's control handle. `/dev/char/180:191 -> ../usb-it913x0`
Operation Manual	At the time users choose this function, the current EEPROM information would be displayed, as shown in Figure 26. Avoiding of wrong EEPROM values, EEPROM checksum procedure would be checked at first. If the value is not correct, the default EEPROM values would be loaded to overwrite all the EEPROM values. **After users set new EEPROM values, users should re-plug the device; in order the driver could get the latest EEPROM modifications.**

```
++++++++++++++++++++++++ Current EEPROM Info ++++++++++++++++++++++++
Serial Number:0x41 0x46 0x30 0x31 0x30 0x32 0x30 0x32 0x30 0x37 0x30 0x30 0x30 0x35 0x37
              A    F    0    1    0    2    0    2    0    7    0    0    0    5    7
TS mode : 0x00 (TS1)
Tuner ID: 0x60
Board ID: 0x00

======= Raw Data Editor =======
1. Read EEPROM Table
2. Modify EEPROM Value
0. Return to Main Menu
===============================
=> Please Input Your Choice:
```

Figure 26: EEPROM interface

3.3.8.1 Read EEPROM Table

User Selection	Select 8 "Read/Write EEPROM " ---> Select 1 "Read EEPROM Table"
Function Description	This function would display all the current EEPROM values of this device.
Calling Function	int **Omega_EEPROM_read** (int handle, uint8_t *eep_tmpbuf, Word *initok_flag);

Function Parameters	(1) **int handle** This is "/dev/usb-it913x0" device file's control handle. `/dev/char/180:191 -> ../usb-it913x0` (2) **uint8_t *eep_tmpbuf** The original default EEPROM value would be loaded in this buffer. We would calculate the EEPROM checksum by using the values in this buffer. (3) **Word *initok_flag** This function would read the original EEPROM value, and also check the checksum of EEPROM values. If EEPROM checksum value is not correct, the default EEPROM value would be loaded, and ***initok_flag** would be set as "0"; otherwise, the checksum is correct, ***initok_flag** would be set as true (1).

Operation Manual	**For example**, if the current device EEPROM vales are as follows:
	(1)Serial Number: AF0102020700057
	(2)TS mode: 0x00 (TS1)
	(3)Tuner ID: 0x60
	(4)Board ID: 0x00
	The following information would be displayed. (As Figure 27)
	`======= Raw Data Editor =======`
	`1. Read EEPROM Table`
	`2. Modify EEPROM Value`
	`0. Return to Main Menu`
	`==============================`
	`=> Please Input Your Choice: 1`
	`++++++++++++++++++++++ Current EEPROM Info ++++++++++++++++++++++`
	`Serial Number:0x41 0x46 0x30 0x31 0x30 0x32 0x30 0x32 0x30 0x37 0x30 0x30 0x30 0x35 0x37`
	`A F 0 1 0 2 0 2 0 7 0 0 0 5 7`
	`TS mode : 0x00 (TS1)`
	`Tuner ID: 0x60`
	`Board ID: 0x00`
	Figure 27: Read EEPROM Table information

3.3.8.2 Modify EEPROM value

User Selection	Select 8 "Read/Write EEPROM " ---> Select 2 "Modify EEPROM value"
Function Description	This function would help users easily change some particular EEPROM value. (Including Serial Number, Tuner ID, and Board ID)
Calling Function	int **Omega_EEPROM_write** (int handle, Word bChangeSN, uint8_t *eep_tmpbuf, uint8_t *sn_buf);
Function Parameters	(1) **int handle** This is "/dev/usb-it913x0" device file's control handle. `/dev/char/180:191 -> ../usb-it913x0`

	(2) **Word bChangeSN** If users choose to change serial number, "**bChangeSN"** would be set as "1"; otherwise, this value would be set as "0". (3) **uint8_t *eep_tmpbuf** This buffer contains the modification of Board ID and Tuner ID. (4) **uint8_t *sn_buf** Modification of Serial Number would be saved in this temporary buffer. (Due to the fact that Serial Number is not in a particular address, this address should be calculated in a special formula.)

Operation Manual	**1) Serial Number:**
	First of all, our testkit would ask users "whether you want to change the Serial Number?"
	If users want to modify this value, they should choose "1" as "yes", and users could insert their own Serial Number. (**15 characters or number** would be accepted. If users insert more than 15 characters or number, only the front 15 characters or number would be adopted.) If users do not want to change this value, just choose "0" as "no", and driver setting would not modify this value. Figure 28 illustrates an example.
	`Change Serial Number? (yes:1, no:0): 1`
	`Please key in the new Serial Number:(max 15 input)BF0102020700057`
	Figure 28: Modify EEPROM value – Change Serial Number example
	2) TS mode:
	If users want to alter different TS mode, our testkit would ask users "whether you want to change TS mode?" at first. If users want to modify this value, they could choose "1" as "yes", and then users could insert the TS mode they want (0: TS1, 2: DCA, 3. PIP); otherwise users choose "0" as "no", and our code would not modify this value. **The insert value should be decimal number**. Figure 29 is an example.
	`Change TS mode? (yes:1, no:0): 1`
	`Please choose the mode you want, 0: TS1, 2: DCA, 3:PIP (decimal number): 2`
	Figure 29: Modify EEPROM value – Change Tuner ID example
	3) Tuner ID:
	If users want to alter Tuner ID, our testkit would ask users "whether you want to change Tuner ID?" at first. If users want to modify this value, they

could choose "1" as "yes", and then users could insert their own Tuner ID; otherwise users choose "0" as "no", and our code would not modify this value. **The insert value should be decimal number**. Figure 30 is an example.

```
Change Tuner ID? (yes:1, no:0): 1
Please key in the new Tuner ID(decimal number):101
```

Figure 30: Modify EEPROM value – Change Tuner ID example

4) Board ID:

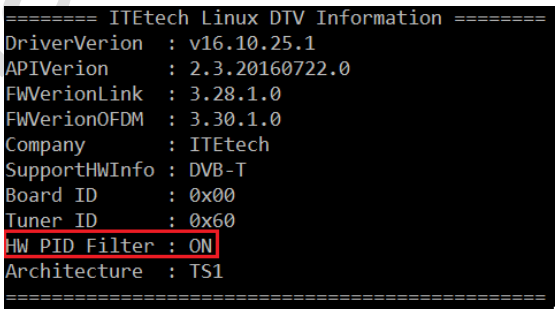
If users want to alter Board ID, at first place, our testkit would ask users "whether you want to change Board ID?" If users want to modify this value, they could choose "1" as "yes", and then users could insert their own Board ID; otherwise users choose "0" as no, and our code would not modify this value. **The insert value should be decimal number**. As Figure 31 is an example.

```
Change Board ID? (yes:1, no:0): 1
Please key in the new Board ID(decimal number):0
```

Figure 31: Modify EEPROM value – Change Board ID example

If users want to check whether driver receives the "new EEPROM values" correctly or not, users could choose "Read EEPROM Table" this item again, and double check the new EEPROM modification.

3.3.9 Set PID Filter

User Selection	Select 9 "Set PID Filter"
Function Description	This function could help user set PIDs via V4L interface. **Note:** 1. The precondition --- "**DVB_USB_ADAP_NEED_PID_FILTER**" must be **enabled** in driver code, due to the restriction of V4L. 2. If users want to reset other PIDs, please **re-execute** the testkit.
Calling Function	void **Omega_Set_PID_Filter** (int *demux);
Function Parameters	(1) **int *demux** This is "/dev/dvb/adapter0/demux0" device file's control pointer handle. `/dev/char/212:0 -> ../dvb/adapter0/demux0`
Operation Manual	This function has to use with "Record & Analyze Packets" together. Before using this function, testkit would check the define tag "**DVB_USB_ADAP_NEED_PID_FILTER**" is enabled or not. Users could get messages from the following two ways. **Method 1**: If "HW PID Filter" is ON, as Figure 32 shown, this function could be used.  **Figure 32: An example of "HW PID Filter" is enabled** **Method 2**: If the "**DVB_USB_ADAP_NEED_PID_FILTER**" is disabled, testkit would show the below message to warn users to enable the define tag, as Figure 33.

```
======= ITEtech Linux DTV Testkit Menu ========
1. Lock Channel
2. Signal Quality Statistics
3. Record & Analyze Packets
4. Mutil-Channel Lock Test
5. Find Free Channel
6. Find 2.4G Free Band
7. Read/Write Register
8. Read/Write EEPROM
9. Set PID Filter
10.Inverse NULL Packet Filter
0. Quit
================================================
=> Please Input Your Choice: 10

 DVB_USB_ADAP_NEED_PID_FILTER should be DISABLE in driver/src/it913x.h
```

Figure 33: Message that warn users that define tag in driver should be enabled

This function would ask users to insert the total number of PID that users want, and also ask users to key in the PID value users' want. Figure 34 is a simple example, the total number of PID that user want is 2, and the PIDs are 0x07db and 0x07e5.

```
1. Set PIDs
0. Return to Main Menu
=> Please Input Your Choice: 1

 Total PIDs Number: 2
 Set PIDs:
  PID[0](in Hex): 07db
  PID[1](in Hex): 07e5
```

Figure 34: PID filter settings.

After finishing the above setting, and choose "Record & Analyze Packets" again, users could see only 0x07db and 0x07e5 would pass the PID filter, as Figure 35.

```
1. Packet Error Testing
2. Save Packet to File
0. Return to Main Menu
=> Please Choice : 1

=> Testing Times: 10
desync data[0] != sync[47]

-PID-------Total----Lost----Error---Kbps-
07db       4842       0       0     7084
07e5       2582       0       0     3777
Press 'A' or 'a' to Stop...

-PID-------Total----Lost----Error---Kbps-
07db      11187       0       0     8167
07e5       5965       0       0     4355
Press 'A' or 'a' to Stop...

-PID-------Total----Lost----Error---Kbps-
07db      17709       0       0     8613
07e5       9171       0       0     4460
Press 'A' or 'a' to Stop...
```

Figure 35: Only the indicating PIDs number would pass the PID filter.

3.3.10 Inverse NULL Packet Filter

User Selection	Select 10 "Inverse NULL Packet Filter"
Function Description	This function could help user filter out Null Packet(PID: 0x1FFF) The precondition --- "**DVB_USB_ADAP_NEED_PID_FILTER**" must be **disabled** in driver code.
Calling Function	void **Omega_Set_Inverse_NULLPacket_Filter** (int handle);
Function Parameters	(1) **int handle** This is "/dev/usb-it913x0" device file's control handle. `/dev/char/180:192 -> ../usb-it913x0`

Operation Manual	This function has to use with "Record & Analyze Packets" together.
	Before using this function, testkit would check the define tag "**DVB_USB_ADAP_NEED_PID_FILTER**" is disabled or not.
	Users could get messages from the following two ways.
	Method 1:
	If "HW PID Filter" is OFF, as Figure 36 shown, this function could be used.
	`======== ITEtech Linux DTV Information ========`
	`DriverVerion : v16.10.25.1`
	`APIVerion : 2.3.20160722.0`
	`FWVerionLink : 3.28.1.0`
	`FWVerionOFDM : 3.30.1.0`
	`Company : ITEtech`
	`SupportHWInfo : DVB-T`
	`Board ID : 0x00`
	`Tuner ID : 0x60`
	`HW PID Filter : OFF`
	`Architecture : TS1`
	`==========`
	Figure 36: An example of "HW PID Filter" is disabled
	Method 2:
	If the "**DVB_USB_ADAP_NEED_PID_FILTER**" is enable, testkit would show the below message to warn users to disable the define tag, as Figure 37.

```
======= ITEtech Linux DTV Testkit Menu ========
1. Lock Channel
2. Signal Quality Statistics
3. Record & Analyze Packets
4. Mutil-Channel Lock Test
5. Find Free Channel
6. Read/Write Register
7. Read/Write EEPROM
8. Set PID Filter
9. Inverse NULL Packet Filter
0. Quit
================================================
=> Please Input Your Choice: 9

 DVB_USB_ADAP_NEED_PID_FILTER should be DISABLE in driver/src/it913x.h
```

Figure 37: Message that warn users that define tag in driver should be disabled

There are only two selections under this choice: Enable or Disable.

1) Disable:

Choose 0 to **Disable** this function. After disable this function and choose "TS Data Analyze & Record", users could see the 0x1FFF packet in the receiving packet list, as Figure 38.

```
-PID-------Total----Lost----Error---Kbps-
0000          13       0       0       19
0012          29       0       0       42
07db        3936       0       0     5826
07dc          53       0       0       78
07dd          53       0       0       78
07e5        3365       0       0     4981
07e6          52       0       0       76
07e7          52       0       0       76
1fff         116       0       0      171
Press 'A' or 'a' to Stop...
```

Figure 38: Null Packet(0x1FFF) could be seen in the receiving packet list.

2) Enable:

Choose 1 to **Enable** this function. After enable this function and choose "TS Data Analyze & Record", users could not see the 0x1FFF packet in the receiving packet list, as Figure 39.

```
-PID-------Total----Lost----Error---Kbps-
0000          12       0       0       17
0012          28       0       0       40
07db        3955       0       0     5786
07dc          53       0       0       77
07dd          53       0       0       77
07e5        3464       0       0     5067
07e6          52       0       0       76
07e7          52       0       0       76
Press 'A' or 'a' to Stop...
```

Figure 39: Null Packet (0x1FFF) could not be seen in the receiving packet list.

4 IOCTL interface

4.1.1 v4l (video for linux) IOCTL Interface

V4l IOCTL Interface	
FE_GET_INFO	
Description	This ioctl call returns information about the front-end. This call only requires read-only access to the device.
FE_SET_FRONTEND	
Description	This ioctl call starts a tuning operation using specified parameters. The result of this call will be successful if the parameters were valid and the tuning could be initiated. The result of the tuning operation in itself, however, will arrive asynchronously as an event (see documentation for FE_GET_EVENT and FrontendEvent.) If a new FE_SET_FRONTEND operation is initiated before the previous one was completed, the previous operation will be aborted in favor of the new one. This command requires read/write access to the device.
FE_READ_STATUS	
Description	This ioctl call returns status information about the front-end. This call only requires read-only access to the device.
FE_READ_SIGNAL_STRENGTH	
Description	This ioctl call returns the signal strength value for the signal currently received by the front-end. For this command, read-only access to the device is sufficient.
FE_READ_SNR	
Description	This ioctl call returns the signal-to-noise ratio for the signal currently received by the front-end. For this command, read-only access to the device is sufficient.
FE_READ_BER	
Description	This ioctl call returns the bit error rate for the signal currently received/demodulated by the front-end. For this command, read-only access to the device is sufficient.

FE_READ_UNCORRECTED_BLOCKS	
Description	This ioctl call returns the number of uncorrected blocks detected by the device driver during its lifetime. For meaningful measurements, the increment in block count during a specific time interval should be calculated. For this command, read-only access to the device is sufficient. Note that the counter will wrap to zero after its maximum count has been reached.
DMX_SET_BUFFER_SIZE	
Description	This ioctl call is used to set the size of the circular buffer used for filtered data. The default size is two maximum sized sections, i.e. if this function is not called a buffer size of 2 * 4096 bytes will be used.
DMX_SET_PES_FILTER	
Description	This ioctl call sets up a PES filter according to the parameters provided. By a PES filter is meant a filter that is based just on the packet identifier (PID), i.e. no PES header or payload filtering capability is supported. The transport stream destination for the filtered output may be set. Also the PES type may be stated in order to be able to e.g. direct a video stream directly to the video decoder. Finally there is a flag field where it is possible to state whether the filtering operation should be started immediately (without waiting for a DMX_START ioctl call). If a filter was previously set-up, this filter will be cancelled, and the receive buffer will be flushed.
DMX_START	
Description	This ioctl call is used to start the actual filtering operation defined via the ioctl calls DMX_SET_FILTER or DMX_SET_PES_FILTER.
DMX_STOP	
Description	This ioctl call is used to stop the actual filtering operation defined via the ioctl calls DMX_SET_FILTER or DMX_SET_PES_FILTER and started via the DMX_START command.

4.1.2 ITE IOCTL Interface

ITE IOCTL Interface	
IOCTL_ITE_DEMOD_GETDRIVERINFO	
Description	This ioctl call is used to get some important information from driver, as Figure 40 shown. By using this ioctl command, testkit could obtain data from driver. This call only requires read-only access to the device.
IOCTL_ITE_DEMOD_GETSIGNALSTRENGTHDBM	
Description	This ioctl call is used to get signal strength from the receiver. This call only requires read-only access to the device.
IOCTL_ITE_DEMOD_WRITEREGISTERS	
Description	This ioctl call is used to modify some blanks of omega registers. This command requires write access to the device.
IOCTL_ITE_DEMOD_READREGISTERS	
Description	This ioctl call is used to read omega registers. This call only requires read-only access to the device.
IOCTL_ITE_DEMOD_WRITEEEPROMVALUES	
Description	This ioctl call is used to modify omega's EEPROM values. (Only Serial Number, Tuner ID, and Board ID are allowed to modify.) This command requires write access to the device.
IOCTL_ITE_DEMOD_READEEPROMVALUES	
Description	This ioctl call is used to read EEPROM of omega. (Only Serial Number, Tuner ID, and Board ID are allowed to read.) This call only requires read-only access to the device.

```
======== ITEtech Linux DTV Information ========
DriverVerion   : v16.10.25.1
APIVerion      : 2.3.20160722.0
FWVerionLink   : 3.28.1.0
FWVerionOFDM   : 3.30.1.0
Company        : ITEtech
SupportHWInfo : DVB-T
Board ID       : 0x00
Tuner ID       : 0x60
HW PID Filter : OFF
Architecture   : TS1
===============================================
```

Figure 40: Testkit display the driver information

Appendix A: Bit Rate Calculation for DVB-T Modulation

DVB-T modulator maximum play rate depends on code rate, constellation, guard interval, bandwidth.
The maximum bit rate can be calculated as below.

Tbandwidth = {6,000,000, 7,000,000, 8,000,000} in Hz for 6MHz, 7MHz, 8MHz
Tcode_rate = {1/2, 2/3, 3/4, 5/6, 7/8}
TConstellation = {2, 4, 6} <- QPSK = 2, 16QAM=4, 64QAM = 6
TGuardInterval = {4/5, 8/9, 16/17, 32/33}, 1/4 = 4/5, 1/8 = 8/9, 1/16 => 16/17, 1/32 => 32/33}
2K/8K mode does not matter

Maximum bit rate = 1512 / 2048 * 188 / 204 * 64 / 56 * TBandwidth * Tcode_rate * TConstellation * TGuardInterval (bps)
= 423 / 544 * TBandwidth * Tcode_rate * TConstellation * TGuardInterval (bps)

Refer to the following tables for calculated results for various configurations.

5 M Bandwidth Maximum bit rate(bps):

GI	CR	QPSK	16-QAM	64-QAM
1 / 4	1 / 2	3110294	6220588	9330882
	2 / 3	4147059	8294118	12441176
	3 / 4	4665441	9330882	13996324
	5 / 6	5183824	10367647	15551471
	7 / 8	5443015	10886029	16329044
1 / 8	1 / 2	3455882	6911765	10367647
	2 / 3	4607843	9215686	13823529
	3 / 4	5183824	10367647	15551471
	5 / 6	5759804	11519608	17279412
	7 / 8	6047794	12095588	18143382
1 / 16	1 / 2	3659170	7318339	10977509
	2 / 3	4878893	9757785	14636678
	3 / 4	5488754	10977509	16466263
	5 / 6	6098616	12197232	18295848
	7 / 8	6403547	12807093	19210640

1 / 32	1 / 2	3770053	7540107	11310160
	2 / 3	5026738	10053476	15080214
	3 / 4	5655080	11310160	16965241
	5 / 6	6283422	12566845	18850267
	7 / 8	6597594	13195187	19792781

6 M Bandwidth Maximum bit rate(bps):

GI	CR	QPSK	16-QAM	64-QAM
1 / 4	1 / 2	3732353	7464706	11197059
	2 / 3	4976471	9952941	14929412
	3 / 4	5598529	11197059	16795588
	5 / 6	6220588	12441176	18661765
	7 / 8	6531618	13063235	19593853
1 / 8	1 / 2	4147059	8294118	12441176
	2 / 3	5529412	11058824	16588235
	3 / 4	6220588	12441176	18661765
	5 / 6	6911765	13823529	20735294
	7 / 8	7257353	14514706	21772059
1 / 16	1 / 2	4391003	8782007	13173010
	2 / 3	5854671	11709343	17564014
	3 / 4	6586505	13173010	19759516
	5 / 6	7318339	14636678	21955017
	7 / 8	7684256	15368512	23052768
1 / 32	1 / 2	4524064	9048128	13572193
	2 / 3	6032086	12064171	18096257
	3 / 4	6786096	13572193	20358289
	5 / 6	7540107	15080214	22620321
	7 / 8	7917112	15834225	23751337

7 M Bandwidth Maximum bit rate (bps):

GI	CR	QPSK	16-QAM	64-QAM
1 / 4	1 / 2	4354412	8708824	13063235
	2 / 3	5805882	11611765	17417647
	3 / 4	6531618	13063235	19593853
	5 / 6	7257353	14514706	21772059
	7 / 8	7620221	15240441	22860662

1 / 8	1 / 2	4838235	9676471	14514706
	2 / 3	6450980	12901961	19352941
	3 / 4	7257353	14514706	21772059
	5 / 6	8063725	16127451	24191176
	7 / 8	8466912	16933824	25400735
1 / 16	1 / 2	5122837	10245675	13568512
	2 / 3	6830450	13660900	20491349
	3 / 4	7684256	15368512	23052768
	5 / 6	8538062	17076125	25614187
	7 / 8	8964965	17929931	26894896
1 / 32	1 / 2	5278075	10556150	15834225
	2 / 3	7037433	14074866	21112299
	3 / 4	7917112	15834225	23751337
	5 / 6	8796791	17593583	26390374
	7 / 8	9236631	18473262	27709893

8 M Bandwidth Maximum bit rate (bps):

GI	CR	QPSK	16-QAM	64-QAM
1 / 4	1 / 2	4976471	9952941	14929412
	2 / 3	6635294	13270588	19905882
	3 / 4	7464706	14929412	22394118
	5 / 6	8294118	16588235	24882353
	7 / 8	8708824	17417647	26126471
1 / 8	1 / 2	5529412	11058824	16588235
	2 / 3	7372549	14745098	22117647
	3 / 4	8294118	16588235	24882353
	5 / 6	9215686	18431373	27647059
	7 / 8	9676471	19352941	29029412
1 / 16	1 / 2	5854671	11709343	17564014
	2 / 3	7806228	15612457	23418685
	3 / 4	8782007	17564014	26346021
	5 / 6	9757785	19515571	29273356
	7 / 8	10245675	20491349	30737024
1 / 32	1 / 2	6032086	12064171	18096257
	2 / 3	8042781	16085561	24128342
	3 / 4	9048128	18096257	27144385

	5 / 6	10053476	20106952	30160428
	7 / 8	10556150	21112299	31668449

Appendix B: Calculation of TS Bitrate

TS bitrate - depends on the PCR for that particular stream. They're sent out at a somewhat standard time gap (mine's at 90ms). All that's involved is getting the PCR base and ext, calculating the PCR based on the formula in the ISO 13818-1 doc, and then using this formula to get the *byte* rate:

27000000 * (packets between PCR final byte) / (PCR2 - PCR1)

multiply that by 8 to get the *bit* rate. basically the PCRs are measurements on the "system clock" and that system clock for TS is 27Mhz. so the units look like this:
ticks/s * bytes / (ticks) === bytes / s

Specifically:

PCR(*i*) *PCR_base*(*i*) 300 *PCR_ext*(*i*)

where:

PCR_base(*i*) ((*system_clock_ frequency* *t*(*i*)) *DIV* 300) *%* 2^{33}

PCR_ext(*i*) ((*system_clock_ frequency* *t*(*i*)) *DIV* 1) *%* 300

system_clock_ frequency =27 000 000 Hz

Refer to ISO 13818-1 2.4.3 Specification of the Transport Stream syntax and semantics and 2.4.4.8 Program Map Table

Table 2-28 – Transport Stream program map section

Syntax	No. of bits	Mnemonic
TS_program_map_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
program_number	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
reserved	3	bslbf
PCR_PID	13	uimsbf
reserved	4	bslbf
program_info_length	12	uimsbf
for (i = 0; i < N; i++) {		
descriptor()		
}		
for (i = 0; i < N1; i++) {		
stream_type	8	uimsbf
reserved	3	bslbf
elementary_PID	13	uimsbf
reserved	4	bslbf
ES_info_length	12	uimsbf
for (i = 0; i < N2; i++) {		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

Table 2-2 – Transport packet of this Recommendation | International Standard

Syntax	No. of bits	Mnemonic
transport_packet(){		
sync_byte	8	bslbf
transport_error_indicator	1	bslbf
payload_unit_start_indicator	1	bslbf
transport_priority	1	bslbf
PID	13	uimsbf
transport_scrambling_control	2	bslbf
adaptation_field_control	2	bslbf
continuity_counter	4	uimsbf
if(adaptation_field_control == '10' || adaptation_field_control == '11'){		
adaptation_field()		
}		
if(adaptation_field_control == '01' || adaptation_field_control == '11') {		
for (i = 0; i < N; i++){		
data_byte	8	bslbf
}		
}		
}		

Table 2-6 – Transport Stream adaptation field

Syntax	No. of bits	Mnemonic
adaptation_field() {		
adaptation_field_length	8	uimsbf
if (adaptation_field_length > 0) {		
discontinuity_indicator	1	bslbf
random_access_indicator	1	bslbf
elementary_stream_priority_indicator	1	bslbf
PCR_flag	1	bslbf
OPCR_flag	1	bslbf
splicing_point_flag	1	bslbf
transport_private_data_flag	1	bslbf
adaptation_field_extension_flag	1	bslbf
if (PCR_flag == '1') {		
program_clock_reference_base	33	uimsbf
reserved	6	bslbf
program_clock_reference_extension	9	uimsbf
}		
if (OPCR_flag == '1') {		
original_program_clock_reference_base	33	uimsbf
reserved	6	bslbf
original_program_clock_reference_extension	9	uimsbf
}		
if (splicing_point_flag == '1') {		

Appendix C: PAT, SDT, NIT

Reference:

ISO_IEC_13818-1 2.4.4 Program specific information and Annex C

ETSI EN_300468

Repetition rates and random access:

The minimum time interval between the arrival of the last byte of a section to the first byte of the next transmitted section with the same PID, table_id and table_id_extension and with the same or different section_number shall be 25 ms.

ETSI TR 101 211 defines the maximum timer interval as,

Table	**Maximum Repetition Rate**
PAT	<100ms
CAT	<1s
TSDT	
reserved	
NIT, ST	<10s
SDT, BAT, ST	<2s
EIT, ST	<2s
RST, ST	
TDT, TOT, ST	<30s

Table 1: PID allocation for SI

Table	PID value
PAT	0x0000
CAT	0x0001
TSDT	0x0002
reserved	0x0003 to 0x000F
NIT, ST	0x0010
SDT, BAT, ST	0x0011
EIT, ST CIT (TS 102 323 [15])	0x0012
RST, ST	0x0013
TDT, TOT, ST	0x0014
network synchronization	0x0015
RNT (TS 102 323 [15])	0x0016
reserved for future use	0x0017 to 0x001B
inband signalling	0x001C
measurement	0x001D
DIT	0x001E
SIT	0x001F

Table 2: Allocation of table_id values

Value	Description
0x00	program_association_section
0x01	conditional_access_section
0x02	program_map_section
0x03	transport_stream_description_section
0x04 to 0x3F	reserved
0x40	network_information_section - actual_network
0x41	network_information_section - other_network
0x42	service_description_section - actual_transport_stream
0x43 to 0x45	reserved for future use
0x46	service_description_section - other_transport_stream
0x47 to 0x49	reserved for future use
0x4A	bouquet_association_section
0x4B to 0x4D	reserved for future use
0x4E	event_information_section - actual_transport_stream, present/following
0x4F	event_information_section - other_transport_stream, present/following
0x50 to 0x5F	event_information_section - actual_transport_stream, schedule
0x60 to 0x6F	event_information_section - other_transport_stream, schedule
0x70	time_date_section
0x71	running_status_section
0x72	stuffing_section
0x73	time_offset_section
0x74	application information section (TS 102 812 [17])
0x75	container section (TS 102 323 [15])
0x76	related content section (TS 102 323 [15])
0x77	content identifier section (TS 102 323 [15])
0x78	MPE-FEC section (EN 301 192 [4])
0x79	resolution notification section (TS 102 323 [15])

PAT: **Program Association Table**

Table 2-25 – Program association section

Syntax	No. of bits	Mnemonic
program_association_section() {		
table_id	**8**	**uimsbf**
section_syntax_indicator	**1**	**bslbf**
'0'	**1**	**bslbf**
reserved	**2**	**bslbf**
section_length	**12**	**uimsbf**
transport_stream_id	**16**	**uimsbf**
reserved	**2**	**bslbf**
version_number	**5**	**uimsbf**
current_next_indicator	**1**	**bslbf**
section_number	**8**	**uimsbf**
last_section_number	**8**	**uimsbf**
for (i = 0; i < N; i++) {		
program_number	**16**	**uimsbf**
reserved	**3**	**bslbf**
if (program_number == '0') {		
network_PID	**13**	**uimsbf**
}		
else {		
program_map_PID	**13**	**uimsbf**
}		
}		
CRC_32	**32**	**rpchof**
}		

NIT: Network Information Table

Table 3: Network information section

Syntax	Number of bits	Identifier
network_information_section(){		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
network_id	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
reserved_future_use	4	bslbf
network_descriptors_length	12	uimsbf
for(i=0;i<N;i++){		
descriptor()		
}		
reserved_future_use	4	bslbf
transport_stream_loop_length	12	uimsbf
for(i=0;i<N;i++){		
transport_stream_id	16	uimsbf
original_network_id	16	uimsbf
reserved_future_use	4	bslbf
transport_descriptors_length	12	uimsbf
for(j=0;j<N;j++){		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

SDT: Service Description Table

Table 5: Service description section

Syntax	Number of bits	Identifier
service_description_section(){		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
transport_stream_id	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
original_network_id	16	uimsbf
reserved_future_use	8	bslbf
for (i=0;i<N;i++){		
service_id	16	uimsbf
reserved_future_use	6	bslbf
EIT_schedule_flag	1	bslbf
EIT_present_following_flag	1	bslbf
running_status	3	uimsbf
free_CA_mode	1	bslbf
descriptors_loop_length	12	uimsbf
for (j=0;j<N;j++){		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

Appendix D: Shorten Latency by Decreasing URB Size

1. Definition of IT913X URB size

The URB size can be changed and moreover this size has an important relation with latency. If users want to shorten the latency, users should try to find the best suitable buffer size to use. The default definition of URB buffer size is in driver code.

- Open "it913x-core.c" of "it913x_driver \src" folder
- Figure 41 illustrates the definition of default URB size.

```
/********** For Dynamic setting URB buffer ************/
static int URB_BUFSIZE = 65424;
//let modinfo could get the URB_NUM information
module_param(URB_BUFSIZE, int, S_IRUGO);
MODULE_PARM_DESC(URB_BUFSIZE, "URB_BUFSIZE = 188*xxx");
/******** For Dynamic setting URB buffer end ***********/
```

Figure 41: URB Definition

2. How to temporary modify IT913x URB size

If users want to tune their own URB size rapidly, here is an easy way for users to do this testing.

The following steps could let customers easily change urb buffer size during debug.

(For the original `# make install` --- would use the default definition URB size)

The usage steps are as follows:

Step 1: Remove the original driver

```
# make remove
```

Step 2: Compiler the driver code:

```
# make
```

Step 3: Customer could set the urb buffer size by

```
# make cu_install URB_BUFSIZE=$(user_set_number)
```

Here are two examples:

Ex: (1) `# make cu_install URB_BUFSIZE=32712` --- set URB_BUFSIZE=188*174 =32712

Ex: (2) `# make cu_install`

--- if users not setting URB_BUFSIZE parameter after the cu_install rule, driver would use the default URB size.

Step 4: If users want to change different URB buffer size

```
# make cu_remove
# make cu_install URB_BUFSIZE=$(new_user_set_number)
```

Ex:
```
# make cu_remove
# make cu_install URB_BUFSIZE=65424
```

(Users could use "**cu_remove**" and "**cu_install**" these two makefile rules reapeatedly until users find the most fit urb buffer size.)

3. Modify IT913x URB size

If users find the most fit URB size, users should substitute the red rectangle (Figure 42) for users' own size.

```
/********** For Dynamic setting URB buffer **************/
static int URB_BUFSIZE = 65424;
//let modinfo could get the URB_NUM information
module_param(URB_BUFSIZE, int, S_IRUGO);
MODULE_PARM_DESC(URB_BUFSIZE, "URB_BUFSIZE = 188*xxx");
/******** For Dynamic setting URB buffer end ***********/
```

Figure 42: Users should modify the red rectangle part to their own URB size

Note:

If users want the better performance of omega, **the URB size in testkit is recommended as the same size as the user setting**. If users do not follow the recommendation, the omega performance would not so good and even could see lost packets.

The definition in testktit is in **it913x_testkit/ it913x_testkit.h**, and the definition size users could modify is as the red rectangle in Figure 43.

```
#include "error.h"

#define READ_DATA_SIZE (188 * 348)
#define PID_SIZE 0x2001 //13bit + 1, ref spec
```

Figure 43: URBSIZE definition in testkit