IT9130 Series —
Programming Guide

Revision History

Revision	Date	Description
0.1	11/17/2009	Initial Release
0.2	01/29/2010	Modified for IT9133 package and register table
0.3	08/02/2010	Added the scripts for different configurations of application circuits. For 9130.api.V2.1.20100528.0 and later.
0.4	04/28/2011	Updated with the release of IT9130FN/BX series

Contents

Figures

Tables

1. Overview

This document is for users who would like to design DVB-T systems using IT9130. This document covers all information users need to know regarding IC control and performance information monitoring. Users can use the information on IC control to learn how to control IT9130. Users can use information on performance monitoring to learn how to read signal quality, signal strength, pre-Viterbi and post-Viterbi bit error rates, and RSD abort counts from the IC.

A complete set of application programming interface (API) is available, facilitating fast and easy integration into products on Windows Mobile, Windows CE, Windows 7, Windows XP, Vista, Linux, and many other operating systems. Furthermore, complete USB drivers on Windows 7/XP/Vista with Microsoft BDA (Broadcast Driver Architecture) compliant, USB drivers on Windows CE/Mobile, and Linux are provided for IT9130.

2. Operation Modes

IT9130 can operate in USB 2.0, and MPEG TS modes. The operation mode is chosen by appropriately setting the strapping pins as specified in Table 22 and register programming after strapping.

The USB2.0 mode is further divided into stand-alone, diversity, and dual-channel modes. To support diversity and dual-channel modes for DVB-T applications, one IT9137 is required as the master device of diversity mode and a second IT9133 or IT9137 is required as the slave device

The control and data paths in each operation mode of IT9130 are summarized in Table 1.

Table 1 Data and control paths of IT9130 in each operation mode

Mode	Data Path	Control Path
USB 2.0	USB	USB
MPEG TS	MPEG TS	2-wire bus

2.1. MPEG TS Mode

In the MPEG TS mode, IT9130 outputs MPEG2 transport streams to the host through either the parallel or serial MPEG2 TS output interface. A backend MPEG2 decoder is needed to interface with IT9130 using the 2-wire bus and the MPEG2 TS interface signals. The MPEG TS mode also supports diversity receiving. More details of the MPEG TS interface are given in Section 4.1.

2.2. Standard USB 2.0 Mode

In the standard USB 2.0 mode, IT9130 communicates with the host through the embedded USB 2.0 interface. The data streams decoded by IT9130 and control/status signals are all encapsulated in the USB frames. The standard USB2.0 mode also supports diversity receiving and dual-channel receiving for picture-in-picture (PIP) applications in DVB-T. More details of the USB interface are given in Section 7.2.

2.3. Dual-Channel Receiving

When operating in the USB mode, dual-channel receiving is supported. The dual-channel receiving function enables two IC's of the IT9130 series to be independently tuned into different RF channels. The contents of the RF channels are simultaneously delivered to the host for picture-in-picture (PIP) and similar applications.

For dual-channel receiving, one IT9135 or IT9137 is connected to the host through the USB interface, and configured as the master device that controls a slave device. The slave device can be an IT9133 or IT9137.

2.4. Diversity Receiving

When operating in the USB modes, diversity receiving is also supported. The diversity receiving function enables two IC's of the IT9130 series, connected through the diversity interface, to communicate and exchange data in order to significantly enhance receiver sensitivity and mobile performance. For diversity receiving, one IT9135 or IT9137 is connected to the host through the USB interface, and configured as the master device that controls an IT9133 or IT9137 slave device.

3. Controlling IT9130

IT9130 ANSI-C source files implement the standard demodulator controlling algorithm. This Section describes how customers can integrate the provided API into their application. Use of the provided ANSI-C source code requires some configuration by the software programmer. This chapter describes the steps required to integrate the API into the application software, and explains how to use the interface between the application software and the provided API.

3.1. Introduction

The ANSI-C API source files are listed in Table 2. The user should modify ONLY `user.h` and `user.cpp` and leave the remaining files unchanged. Adhering to this limitation will minimize the work required for any future updates or upgrades.

The user application should use the provided API for ALL communications with the demodulator. Accessing registers outside this API may produce unpredictable results.

The status returned by each provided function is described in `error.h`.

3.2. Source Files

The ANSI-C API source files are listed in Table 2. They have been developed to support single and multi-demodulator applications (diversity receiving or dual-channel receiving). The user should only modify `user.h` and `user.cpp`. Directly accessing registers without using the API could corrupt the internal state of the API. If the user modifies registers without using the supplied routines, the results are unexpected.

Table 2 IT9130 API source files.

File	User Modifiable	Description
cmd.h	NO	Protocol-related header file.
cmd.cpp	NO	Protocol-related implementation file.
error.h	NO	Status/error code definition.
firmware.h	NO	Firmware file.
it9130.h	NO	Demodulator API functions header file.
it9130.cpp	NO	Demodulator API functions implementation file.
register.h	NO	Registers definition.
standard.h	NO	Control algorithm header file.
standard.cpp	NO	Control algorithm implementation file.
tuner.h	NO	Tuner API functions header file.
tuner.cpp	NO	Tuner API functions implementation file.
type.h	NO	Data type and constants definition.

user.h	YES	API-related settings. Please refer to Section 0 for details.
user.cpp	YES	System-dependant function. Please refer to Chapter 3.4 for details.
variable.h	NO	Variables definition.
version.h	NO	User could check this file to know the version of API.
xxx_Script_X.h	NO	The script files to be loaded before IT9130 is being initialized.

3.3. User.h

`User.h` contains API-related settings. Users are recommended to fully understand this file before integrating IT9130 API in the target application platform. There are necessary changes in the settings of this file for each target application to assure proper functions of the entire system. All settings are summarized in Table 3.

Table 3 Settings in `user.h` file.

Setting	Description
User_I2C_SPEED	The default value is ***0x0D***, which means the I2C clock speed from the first device to the second device is 197KHz (1000000000 / (24.4 * 16 * User_I2C_SPEED)). Be aware of that this value is only valid in DCA and PIP mode.
User_Chip2_I2C_ADDRESS	I2C address of the 2nd device (only for DCA or PIP mode).

3.4. User.cpp

This file is where the host processor and hardware-specific functions are placed. The API utilizes these functions during normal operations. For example, each system could have different delay functions and each bus could have different read/write procedures. This file provides the empty functions and the user is responsible for writing the code for the functions in this file. The functions in this file are listed in Table 4.

Table 4 Functions in user.cpp.

Function	Description
User_delay	API utilizes this function for timeout purpose.
User_enterCriticalSection	API utilizes this function to lock synchronize object. If your system supporting multi-thread, you should implement this function to avoid racing condition.
User_leaveCriticalSection	API utilizes this function to release synchronize object. If your system supporting multi-thread, you should implement this function to avoid racing condition.
User_mpegConfig	API utilizes this function to configure MPEG-2 interface, like clock polarity, clock speed…etc..

User_busTx	API utilizes this function to send control message to desired bus.
User_busRx	API utilizes this function to receive control message from desired bus.
User_busRxData	API utilizes this function to receive data from desired bus. Because the user might need one bus for control message and another for data access, so API provide three function to fulfill such requirement. In most case, User_busRx could be identical to User_busRxData, if there is only one kind of bus.

3.5. Function Prototype Description

Most API functions have a prototype as follows:

```
Dword Demodulator_XXX (
     IN  Demodulator*     demodulator
     …
);
```

In the prototype, *demodulator* is the device handle used to store all device related information, and other arguments depend on individual functions. The user has to create the device handle in order to access API functions as shown below:

```
IT9130 it9130;
```

The user can then use it to access each of API functions as shown below:

```
Dword error = Error_NO_ERROR;

error = Demodulator_XXX ((Demodulator*) &it9130, …);
```

For more details about each API function, please refer to the comments of `it9130.h`.

3.6. Initialization

Before the initialization of IT9130, appropriate selection of the initialization script (Init_Script) to match the application circuit design is required in advance to achieve the optimal reception performance. The function, OMEGA_supportLNA( ), should be executed in the first place to select the script corresponding to your hardware design configuration. Three different RF front-end circuit designs are supported currently, summarized in Table 5.

Table 5 Parameters for OMEGA_supportLNA().

Type	Name	Description
Byte	supporttype	The parameter is used to indicate the used Init_Script. ● OMEGA , 0x00

		(for designs without external LNA; e.g. for IT9135) ● OMEGA_LNA_Config_1, 0x01 (for IT9133FN_NIM_TS_V14 and IT9133+IT9137 dual-channel reference design) ● OMEGA_LNA_Config_2, 0x02 (for IT9133FN_NIM_TS_V15, V16, V20 reference design)

Initialization is required before any operation. To initialize IT9130, the user has to call `Demodulator_initialize()`, which performs all necessary initialization operations including firmware downloading, firmware-related, and tuner-related initializations. The parameters required from the user include `chipNumber`, `sawBandwidth`, `streamType`, and `architecture`. These parameters and their meanings are summarized in Table 6.

Table 6 Parameters for initialization.

Type	Name	Description
Byte	chipNumber	This parameter means the number of IC's of the IT9130 series in your system. In other words, if you use one IC on your system, the value of this parameter should be 1. If you use two IC's for Diversity Combining or Dual-Channel Receiving (PIP), the value should be 2.
Word	sawBandwidth	The bandwidth of the tuner's SAW filter.
StreamType	streamType	The parameter is used to indicate desired output format. ● StreamType_DVBT_DATAGRAM: For DVB-T service and transmit the Transport Stream (TS) through read data bus operation. Please refer to 0. ● StreamType_DVBT_PARALLEL: For DVB-T service and transmit the Transport Stream through MPEG-2 parallel interface. ● StreamType_DVBT_SERIAL: For DVB-T service and transmit the Transport Stream through MPEG-2 serial interface.
Architecture	architecture	This parameter refers to the architecture (Diversity Receiving or Dual-Channel Receiving [PIP]) of a system with more than 1 IT9130 series IC. This parameter is valid only when numberOfChips is great than one. The possible values are Architecture_DCA and Architecture_PIP, for more information about DCA and PIP please refer to Section 0.

An example is given below:

```
IT9130 it9130;

Byte chipNumber = 1;                          // There is only one demodulator in the system.
```

```
Word sawBandwidth = 8000;                              // SAW filter is 8000 KHz.
StreamType streamType = StreamType_DVBT_DATAGRAM;  // Device will output DVB-T datagram.
Architecture architecture = Architecture_DCA;       // Device will operate in DCA mode.
Dword error = Error_NO_ERROR;

error = OMEGA_supportLNA(it9130, 0x02); // with External LNA Config. 2 design (IT9133FN_NIM_TS_ V15)
if (error)      return;

error = Demodulator_initialize ((Demodulator*) &it9130, chipNumber, sawBandwidth, streamType, architecture);
```

3.7. Changing Architecture (Only When `numberOfChips` > 1)

This Section can be ignored If there is only one IT9130 series IC in your system.

To change the system architecture after initialization, call `Demodulator_setArchitecture()`. An example is given below.

```
IT9130 it9130;
Architecture architecture;
Dword error = Error_NO_ERROR;

// Change the architecture to DCA
architecture = Architecture_DCA;
error = Demodulator_setArchitecture ((Demodulator*) &it9130, architecture);

// Change the architecture to PIP
architecture = Architecture_PIP;
error = Demodulator_setArchitecture ((Demodulator*) &it9130, architecture);
```

3.8. Getting Firmware and API Versions

The user can use `Demodulator_getFirmwareVersion()` to get the firmware versions respectively. Note that the user has to assign the correct processor type when calling `Demodulator_getFirmwareVersion()`.

The API version is stored in `version.h`.

An example is given below.

```
IT9130 it9130;
Dword linkFirmwareVersion;      // Used to store LINK firmware version.
```

```
Dword ofdmFirmwareVersion;      // Used to store OFDM firmware version.
Dword error = Error_NO_ERROR;

// Get LINK firmware version
error = Demodulator_getFirmwareVersion ((Demodulator*) &it9130, Processor_LINK, &linkFirmwareVersion);

// Get OFDM firmware version
error = Demodulator_getFirmwareVersion ((Demodulator*) &it9130, Processor_OFDM, &ofdmFirmwareVersion);
```

3.9. Acquiring Channel

Acquiring channel is accomplished by calling `Demodulator_acquireChannel()`. The desired channel frequency and bandwidth provided to the function are in units of kHz. After calling the function, the user can use `Demodulator_isLocked()` to check the condition of desired channel. If a DVB-T signal is present and MPEG lock can be achieved, the input argument `locked` will assume a value of TRUE. Otherwise it will assume a value of FALSE.

An example is given below.

```
IT9130 it9130;
Byte chip = 0;                  // Acquire a channel of chip 0 (first chip).
Dword frequency = 666000;       // Frequency is 666000 KHz.
Word bandwidth = 8000;          // Bandwidth is 8000 KHz.
Bool locked;                    // The output paramter to check if the channel is locked.
Dword error = Error_NO_ERROR;

error = Demodulator_acquireChannel ((Demodulator*) &it9130, chip, bandwidth, frequency);
if (error) {
     printf ("Error Code = %X", error);
     return;
}
error = Demodulator_isLocked ((Demodulator*) &it9130, chip, &locked);
if (error) {
     printf ("Error Code = %X", error);
     return;
}
```

3.10. Scanning Frequency Range

In a typical DVB-T deployment, multiple RF channels are available in the allocated spectrum. Each RF channel further contains one or more services that are each identified by the PID, as shown in Figure 1. A flow-chart for scanning for available DVB-T channels in a given frequency range using IT9130 is shown in

Figure 2.

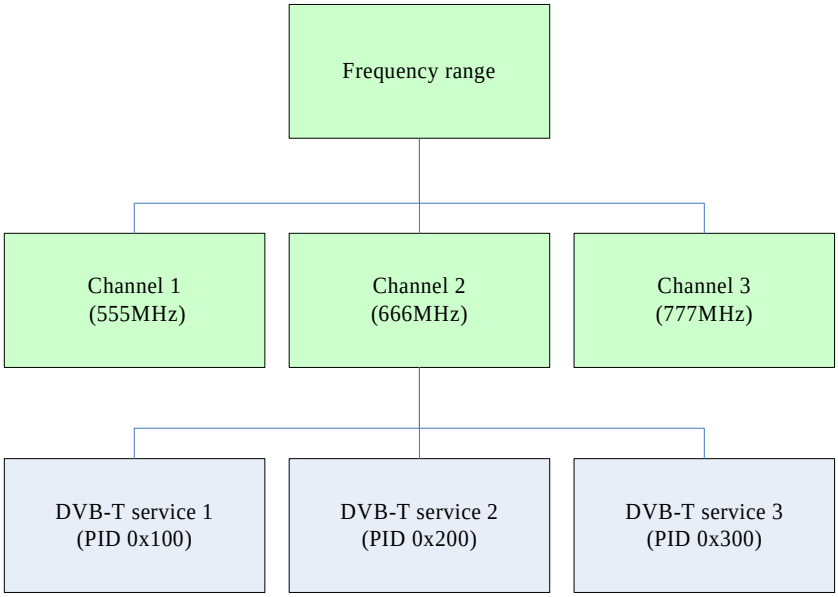

Figure 1 Typical DVB-T service structure.

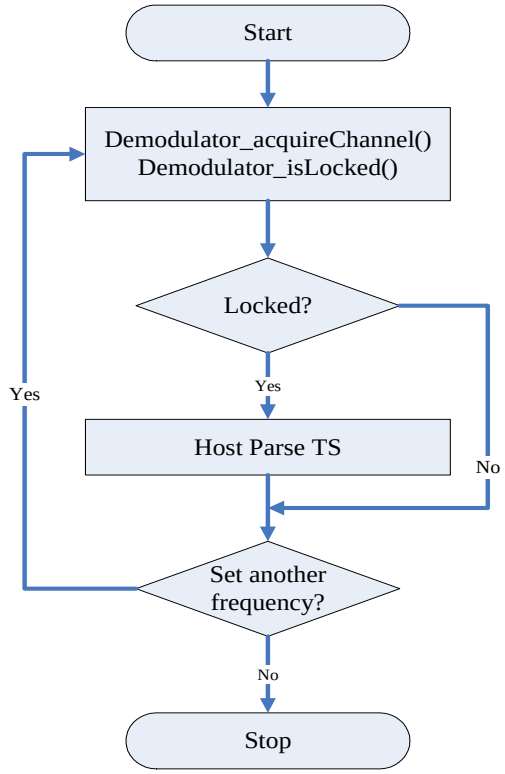

Figure 2 Flowchart for scanning for available DVB-T services in a frequency range.

An example program is given below.

```
IT9130 it9130;
Dword frequency;
Word bandwidth = 6000;
Bool locked;
Dword error = Error_NO_ERROR;

for (frequency = 533000; frequency <= 593000; frequency += bandwidth) {
      error = IT9130_acquireChannel ((Demodulator*) &it9130, 0, bandwidth, frequency);
      if (error) {
            printf ("Error Code = %X", error);
            return;
      }
      error = IT9130_isLocked ((Demodulator*) &it9130, 0, &locked);
      if (error) {
            printf ("Error Code = %X", error);
            return;
      }
      if (locked == True) {
            ......; // DVB-T: store valid frequencies.
      }
}
```

3.11. Selecting a Channel

Selecting a channel can be accomplished by calling `Demodulator_acquireChannel()`. Please refer to Section 0 for details.

3.12. Adding a PID to the PID Filter

`Demodulator_addPidToFilter()` is used for adding a PID to the hardware PID filter. At most 32 PID entries are supported in IT9130.

An example is given below.

```
IT9130 it9130;
Pid pid;
Dword error = Error_NO_ERROR;

// For DVB-T
```

```
pid.value = 0x0000;
error = Demodulator_addPidFilter ((Demodulator*) &it9130, 0, pid);
```

3.13. Resetting PID Filter

`Demodulator_resetPid()` is used to reset the hardware PID filter. All entries in the PID filter will be set to 0 after the PID filter is reset.

An example is given below.

```
IT9130 it9130;
Dword error = Error_NO_ERROR;

error = Demodulator_resetPid ((Demodulator*) &it9130, 0);
```

3.14. Getting Data

A BDA driver is provided for IT9130 users, and the user can use BDA Extension functions to get data accordingly. For driver makers and users without the BDA driver, `Demodulator_getDatagram()`is provided for retrieving data from IT9130 series IC's configured as `StreamType_DVBT_DATAGRAM`. It is very important to note that the user should never access the data bus without using `Demodulator_getDatagram()`.

For IT9130 series IC's configured as `StreamType_DVBT_PARALLEL` or `StreamType_DVBT_SERIAL`, the Transport Stream (TS) is directly output through the MPEG-2 parallel or serial interface. `Demodulator_getDatagram()` is not applicable in this case.

An example is given below.

```
IT9130 it9130;
Dword bufferLength;        // The maximum size of the buffer
Byte buffer[2048];         // The buffer to place DVB-T TS packets
                           // and recommended more than 2048 bytes in DVB-T
Dword error = Error_NO_ERROR;

// For DVB-T
bufferLength = 2048
error = Demodulator_getDatagram ((Demodulator*) &it9130, bufferLength, buffer);
```

3.15. Power Saving

Power consumption can be significantly reduced when the IT9130 is not being used. It can be achieved by

programming certain registers to slow down or power off some parts of the IT9130 IC. Resuming from the power-saving mode is also achieved by programming certain registers. `Demodulator_controlPowerSaving()` is provided for this purpose.

An example is given below.

```
IT9130 it9130;
Byte chip = 0;
Byte control = 0;                    // 1: Power up, 0: Power down
Dword error = Error_NO_ERROR;

error = Demodulator_controlPowerSaving ((Demodulator*) &it9130, chip, control);
```

3.16. Register Access

The signal processing parameters of IT9130 can be adjusted by appropriately programming its registers. Similarly, many performance and operation status indicators of IT9130 can be obtained by reading its registers. It is strongly emphasized that once `Demodulator_initialize()`has been called, registers should never be modified unless otherwise mentioned in this Manual. Modifying registers *before* calling `Demodulator_initialize()` is in general permitted. In this case the user can modify a register and then read it back to make sure that the control path is in good condition.

3.17. Reading and Writing Registers with API Functions

The functions `Demodulator_readRegister()` and `Demodulator_writeRegister()` are, respectively, used for reading and writing one register of the IT9130 series IC. The functions `Demodulator_readRegisters()` and `Demodulator_writeRegisters()` are used for burst access, i.e., reading and writing multiple registers.

Two reference register addresses and their default values are listed in Table 7 and the user can verify the bus integrity by reading these two registers.

Table 7 Reference registers and their default values

Address	Processor	Default Value
0xF000	Processor_LINK	0xAE
0xF103	Processor_LINK	0x0D

An example is given below for register reading. Note that the user must specify the target processor for which register reading and writing is to be performed.

```
IT9130 it9130;
Byte chip = 0;                          // Read a register from chip 0 (first chip).
Processor procssor = Processor_LINK;    // Read a register from LINK processor.
Dword reigster = 0xF000;                // Read a register at 0xF000.
Byte value;                             // Register value.
```

```
Dword error = Error_NO_ERROR;

error = Demodulator_readRegister ((Demodulator*) &it9130, chip, processor, register, &value);
```

3.18. Finalize

Please remember to call `Demodulator_finalize()` before shutting down the system. Once this function has been called, it will release tuner-related resources.

An example is given below.

```
IT9130 it9130;
Dword error = Error_NO_ERROR;

error = Demodulator_finalize ((Demodulator*) &it9130);
```

3.19. Using General Purpose Input/Output (GPIO)

Please refer to the application note, IT9130 Series GPIO User Manual.

4. Configuring the Host Interface

4.1. IT9130 Host Interface

IT9133 and IT9137 provides flexible interface, the so-called Host Interface, for connecting to the host in stand-alone applications, and for connecting to the master device, slave device, or the host in diversity receiving and dual-channel receiving modes. Depending on the application, the Host Interface can be configured into input interfaces or output interfaces. Host Interface comprises pins HOST_B0~HOST_B11.

IT9137 Host Interface (HOST_B0~HOST_B11) is used for connecting to the host in stand-alone modes, or for connecting to the master/slave device or the host in diversity receiving and dual-receiving modes. When connecting to the host, Host Interface (HOST_B0~HOST_B11) is configured into the MPEG TS mode with appropriate address selection by appropriate setting the strapping pins listed in Table 22. When connecting to the salve device, Host Interface should be configured as the master diversity interface for diversity receiving or configured as the MPEG TS input interface for dual-channel receiving. When connecting to the master device, Host Interface (HOST_B0~HOST_B11) should be configured into slave diversity interface for diversity receiving or MPEG TS output interface for dual-channel receiving, where the I2C address should be selected as 0x3A.

IT9133 Host Interface (HOST_B0~HOST_B11) is used to communicate with the host for MPEG-TS mode, and for communicating with the master device for diversity receiving and dual-channel receiving modes. When connecting to host, Host Interface (HOST_B0~HOST_B11) is configured as the MPEG TS mode with appropriate address selection, which is selected by setting the strapping pins listed in Table 22. When connecting to the master device, Host Interface (HOST_B0~HOST_B11) should be configured into slave diversity interface for diversity receiving or MPEG TS output interface for dual-channel receiving, where the I2C address of IT9133 should be selected as 0x3A by the strapping pins listed in Table 22.

5. Configuring MPEG-2 Transport Stream Interface

The IT9130 MPEG-2 transport stream interface pins are listed in Table 8. An example timing diagram of these pins is shown in Figure 3. Transitions of MPEG Data[7:0], MPEG Fail, MPEG Sync, and MPEG Valid are triggered by the falling edge of MPEG Clock in the example shown in Figure 3.

Table 8 MPEG2 TS interface pin list.

<table>
<tr><th rowspan="2">Pin Name</th><th colspan="2">MPEG TS Output Interface
(IT9133/IT9137)</th><th colspan="2">MPEG TS Input Interface
(IT9137)</th></tr>
<tr><th>Function</th><th>Description</th><th>Function</th><th>Description</th></tr>
<tr><td>HOST_B0</td><td>MPEG Fail</td><td>MPEG uncorrectable packet indicator</td><td>MPEG Data[7]</td><td rowspan="8"># MPEG transport stream data.
8-bit in the parallel mode and 1-bit in the serial mode.
Data[0] or Data[7] can be the data pin in the serial mode</td></tr>
<tr><td>HOST_B1</td><td>MPEG Sync</td><td>MPEG packet sync</td><td>MPEG Data[6]</td></tr>
<tr><td>HOST_B2</td><td>MPEG Valid</td><td>MPEG data valid</td><td>MPEG Data[5]</td></tr>
<tr><td>HOST_B3</td><td>MPEG Clock</td><td>MPEG clock</td><td>MPEG Data[4]</td></tr>
<tr><td>HOST_B4</td><td>MPEG Data[0]</td><td rowspan="8"># MPEG transport stream data.
8-bit in the parallel mode and 1-bit in the serial mode.
Data[0] or Data[7] can be the data pin in the serial mode</td><td>MPEG Data[3]</td></tr>
<tr><td>HOST_B5</td><td>MPEG Data[1]</td><td>MPEG Data[2]</td></tr>
<tr><td>HOST_B6</td><td>MPEG Data[2]</td><td>MPEG Data[1]</td></tr>
<tr><td>HOST_B7</td><td>MPEG Data[3]</td><td>MPEG Data[0]</td></tr>
<tr><td>HOST_B8</td><td>MPEG Data[4]</td><td>MPEG Clock</td><td>MPEG clock</td></tr>
<tr><td>HOST_B9</td><td>MPEG Data[5]</td><td>MPEG Valid</td><td>MPEG data valid</td></tr>
<tr><td>HOST_B10</td><td>MPEG Data[6]</td><td>MPEG Sync</td><td>MPEG packet sync pulse</td></tr>
<tr><td>HOST_B11</td><td>MPEG Data[7]</td><td>MPEG Fail</td><td>MPEG uncorrectable packet indicator</td></tr>
</table>

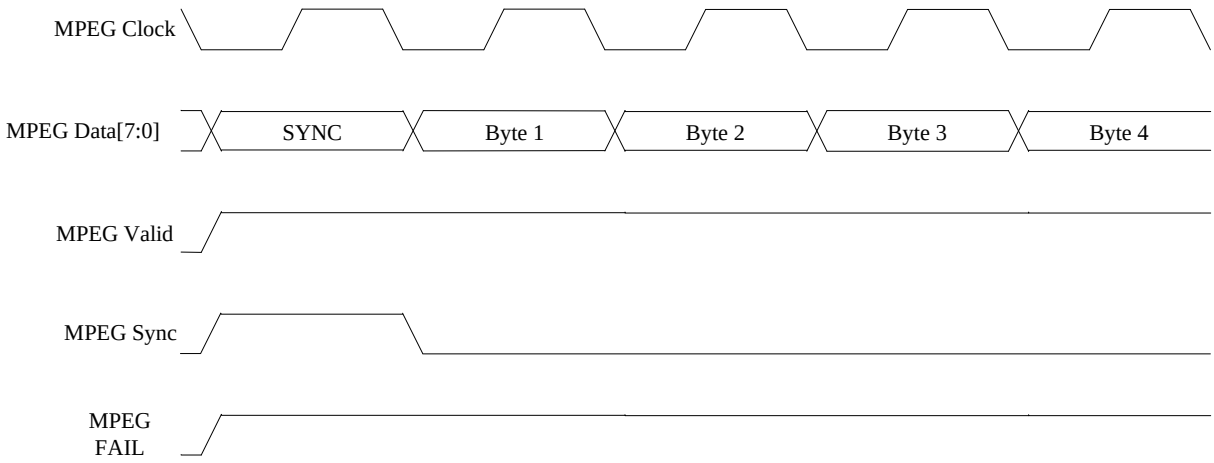

Figure 3 An example of MPEG2 parallel interface timing diagram.

5.1. Configurable Parameters

The MPEG-2 transport stream interface of IT9130 is fully configurable. The complete set of configuration registers is described in detail in Table 9. In this table, registers with address 0xDxxx should be accessed with the `processor` argument set to `Processor_LINK`, and those with address 0xFxxx should be accessed with the `processor` argument set to `Processor_OFDM`.

Table 9 Configuration registers for the MPEG-2 TS interface.

Register Name	Register Address[bits]	Description	Default Value
mpeg_ser_mode	0xF985[0]	Selection about **parallel output mode**, **serial output mode**, or **serial input mode**. See Table 10 for more details.	0
mpeg_par_mode	0xF986[0]		0
mpeg_ser_do7	0xF984[0]	Pin select for the **serial output mode** and **serial input mode** 0: TS data is carried on MPDATA0 1: TS data is carried on MPDATA7	0
msdo_msb	0xF98A[0]	For the **serial output mode**, select the endianness of the serial data output. 0: LSB first 1: MSB first For the **parallel output mode**, select the bit ordering of the parallel output 0: MPDATA7 is the MSB of TS data 1: MPDATA0 is the MSB of TS data	1
reg_ts_lsb_1st	0xF9B3[0]	Endianness of the **serial input mode** 0: LSB first 1: MSB first.	0

keep_sf_sync_byte	0xF982[0]	Style of MPEG-2 sync byte 0: MPEG-2 style. The inverted sync byte (0xB8) in DVB-T shall be inverted again so that the output sync byte is always 0x47 as required by MPEG-2. 1: DVB-T style. In this case the FEC module transfers the decoded data to the MPEG-2 TS interface without modification. The sync byte of every eighth super-frame is inverted (0xB8) according to the DVB-T specifications.	0
mpeg_clk_pol	0xF98F[0]	The polarity of MPCLK. 0: Data changes after the rising edge of MPCLK 1: Data changes after the falling edge of MPCLK	0
mpeg_vld_pol	0xF98E[0]	The polarity of MPEG-2 Valid 0: Active high (set to 1 to indicate valid data) 1: Active low	0
mpeg_sync_pol	0xF98D[0]	The polarity of MPEG-2 Sync 0: Active high (set to 1 to indicate frame sync) 1: Active low	0
mpeg_err_pol	0xF98C[0]	The polarity of MPEG-2 FAIL 0: Active high (set to 1 to indicate error) 1: Active low	0
mpeg_clk_gated	0xF98B[0]	The style of MPEG-2 Clock 0: Free running. MPEG-2 Clock is always present 1: Gated. There is no MPEG-2 Clock when no valid data is available.	0
reg_mpeg_vld_tgl	0xF99C[0]	The style of MPEG-2 Valid 0: MPFRM will be sent out continuously. 1: Allow gaps in between bytes in MPFRM.	0
reg_packet_gap	0xF9B0[7:0]	The gap between consecutive 188-byte payloads in units of byte times. Takes values of 0 to 255. Applies to the **parallel output mode** and **serial output mode.**	0x10
mssync_len	0xF989[0]	For the **serial output mode**, select whether MPEG-2 Sync is asserted only for the first bit or for all bits of the first byte. 0: Asserted for all bits of the first byte 1: Asserted for the first bit only	0
reg_mp2_sw_rst	0xF99D[0]	MPEG Interface software reset	1
reg_fw_table_en	0xF9AD[0]	MPEG-2 Clock frequency control: 0: The frequency of MPEG-2 Clock depends on the actual bandwidth and TPS parameters of the current DVB-T channel. 1: The frequency of MPEG-2 Clock depends on the	0

		target values in the registers reg_tpsd_bw_mp2if, reg_tpsd_gi_mp2if, reg_tpsd_cr_mp2if, and reg_tpsd_cons_mp2if.	
reg_tpsd_bw_mp2if	0xF9A9[0]	Target DVB-T channel bandwidth for MPEG-2 Clock frequency control	0x0
reg_tpsd_gi_mp2if	0xF9AA[0]	Target guard interval for MPEG-2 Clock frequency control	0x0
reg_tpsd_cr_mp2if	0xF9AB0]	Target code rate for MPEG-2 Clock frequency control	0x0
reg_tpsd_cons_mp2if	0xF9AC[0]	Target constellation for MPEG-2 Clock frequency control	0x0
reg_ts_capt_bg_sel	0xF9B4[0]	For the **serial input mode** 0: MPEG-2 Sync is ignored (can be wired to the ground) 1: MPEG-2 Sync is used.	1
reg_tsip_clk_inv	0xD820[0]	For the **parallel input mode**, invert the polarity of the input MPEG-2 Clock if set to 1.	0
reg_ts_clk_inv	0xD821[0]	For the **serial input mode**, invert the polarity of the input MPEG-2 Clock if set to 1.	0
reg_ts_dat_inv	0xF9B2[0]	For the **serial input mode**, invert the polarity of the input DATA[0] or DATA[7] if set to 1.	0
reg_ts_sync_inv	0xF9B7[0]	For the **serial input mode**, invert the polarity of the input MPEG-2 Sync if set to 1.	0
reg_ts_vld_inv	0xF9B8[0]	For the **serial input mode**, invert the polarity of the input MPEG-2 Valid if set to 1.	0
reg_top_padmpdrsr	0xD82D[0]	MPEG output slew rate control: 0: Default 1: Slew rate boosts	0
reg_top_padmiscdr2	0xD830[0]	Bit 0 of MPEG output driving control	0
reg_top_padmiscdr4	0xD831[0]	Bit 1 of MPEG output driving control	1
reg_top_padmiscdr8	0xD832[0]	Bit 2 of MPEG output driving control	0

5.2. Interface Modes

The MPEG TS interface of IT9130 offers both parallel and serial input and output. For IT9130 operating in the MPEG-TS mode, the MPEG TS interface can operate in the parallel output mode, serial output mode, or be disabled. The registers *reg_tsip_en* and *reg_tsis_en* must both be programmed to 0 for these part numbers. By programming the register *mpeg_ser_mode* to 1, the MPEG-2 transport stream interface will operate in the serial output mode. In this mode the MPEG TS data can be configured to be output on pins Data[7] or Data[0] by programming the register *mpeg_ser_do7* as shown in Table 9. On the other hand, the MPEG-2 transport stream interface will operate in the parallel output mode if the register *mpeg_ser_mode* is programmed to 0 and *mpeg_par_mode* is programmed to 1. If both *mpeg_ser_mode* and *mpeg_par_mode* are programmed to 0 the MPEG-2 transport stream interface will be disabled and there will be no output.

On the other hand, for IT9130 operating in the standard USB2.0 mode, the MPEG-2 transport stream interface is an input interface that accepts MPEG-2 transport streams in both serial and parallel fashion. The

register *reg_tsis_en* (for serial input mode) or *reg_tsip_en* for parallel input mode must be programmed to 1 to enable the input port. Furthermore, by programming the register *reg_ts_capt_bg_sel* to 0, MPEG Sync of IT9130 can be wired to the ground if MPEG Sync is not available from the MPEG TS stream source.

A truth table for selecting the IT9130 MPEG-2 TS interface mode is given in Table 10.

Table 10 The truth table for IT9130 MPEG-2 TS interface mode selection.

Operation Mode	*reg_tsip_en*	*reg_tsis_en*	*mpeg_ser_mode*	*mpeg_par_mode*	MPEG-2 TS interface operation mode
MPEG TS (IT9133/37)	0	0	1	X	Serial Output
	0	0	0	1	Parallel Output
	0	0	0	0	Disabled
USB 2.0 (IT9137)	1	0	0	0	Parallel Input
	0	1	0	0	Serial Input

5.3. Bit Ordering

The bit-ordering of the parallel output interface for IT9130 and the endianness of the serial input and output of IT9130 are configurable by programming the registers *msdo_msb* and *reg_ts_lsb_1st* as described in Table 9.

5.4. Sync Byte Style

In IT9130 the sync byte can be configured to be of the MPEG-2 style or DVB-T style. When the register *keep_sf_sync_byte* is set to 0, the sync byte shall be of the MPEG-2 style and will always be 0x47. On the other hand, if the register *keep_sf_sync_byte* is set to 1, the sync byte shall be of the DVB-T style and will be inverted (0xB8) once every eight packets.

5.5. Signal Polarity

The polarity of the MPEG Clock, MPEG Sync, MPEG Valid, and MPEG Error signals can also be configured to be active high or low by programming the corresponding registers described in Table 9.

5.6. Signal Timing

In IT9130, MPEG Valid can be selected to be continuous or gapped by programming the register *reg_mpeg_vld_tgl* as shown in Table 9. For the serial output mode, by programming the register *mssync_len*, MPEG Sync can be chosen to be high only during the first bit of the first byte or during the entire first byte, as shown in Table 9. Finally, by programming the register *mpeg_clk_gated* as shown in Table 9, the style of MPCLK can also be configured to be gated, i.e., present only when data is valid, or free-running, which means always present regardless of whether the data is valid or not.

For the parallel output and serial output modes, the MPEG-2 transport stream only outputs the 188-byte payload of the transport stream packets. The remaining 16 parity bytes are not output from this interface. The gap, in units of byte durations, between consecutive 188-byte payloads can be programmed through the register *reg_packet_gap* (0~255, default value is 16 byte durations).

5.7. Clock Frequency

There are four possible MPEG Clock output frequencies. These frequencies are 5.12MHz, 10.24 MHz, 20.48MHz, and 40.96 MHz for the serial output modes and 0.64MHz, 1.28MHz, 2.56MHz, and 5.12MHz for the parallel mode. The actual MPEG Clock frequency of the MPEG-2 transport stream interface can be configured to depend on the actual bandwidth and TPS parameters of the current DVB-T channel or on a set

of “target” bandwidth and TPS parameters. The actual MPEG Clock frequency is the lowest frequency capable of supporting the required data rate computed based on the actual bandwidth and TPS parameters of the current DVB-T channel or on the set of target bandwidth and TPS parameters.

5.8. Pin Driving Capability

The slew rate and driving capabilities of the MPEG-2 TS interface pins are also configurable by programming the registers *reg_top_padmiscdr*, *reg_top_padmiscdr8*, *reg_top_padmiscdr4, and reg_top_padmiscdr2* as shown in Table 9.

5.9. Parallel Output Interface

For IT9130, by programming the register *mpeg_par_mode* to 1 and *mpeg_ser_mode* to 0, each byte of the payload of the MPEG-2 transport stream will be output to pins HOST_B4~HOST_B11 in parallel. MPEG Valid is asserted when a payload byte is being output on HOST_B4~HOST_B11. MPEG Sync is asserted at the first payload byte of each transport stream packet. MPEG Fail is asserted throughout the entire duration of any erroneous transport stream packet.

Note that there can be gaps between bytes in MPEG Valid. If the register *reg_mpeg_vld_tgl* is programmed to 0, MPEG Valid will be sent out continually without any gaps in between. Example timing diagrams with continuous and gapped MPFRM are shown in Figure 4 and Figure 5, respectively.

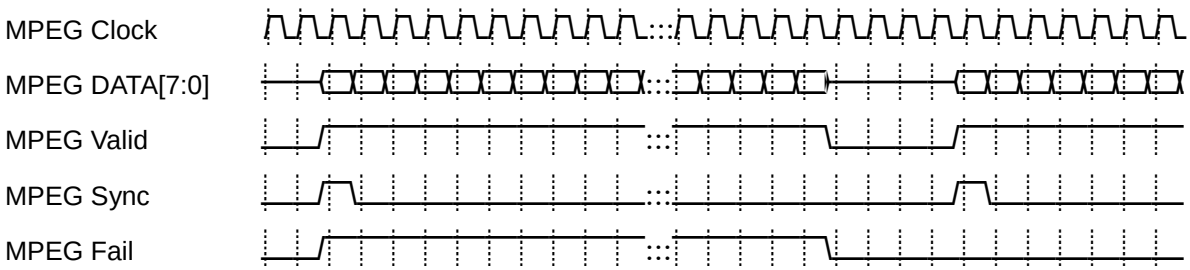

Figure 4 Timing diagram with continuous MPEG Valid in parallel mode.

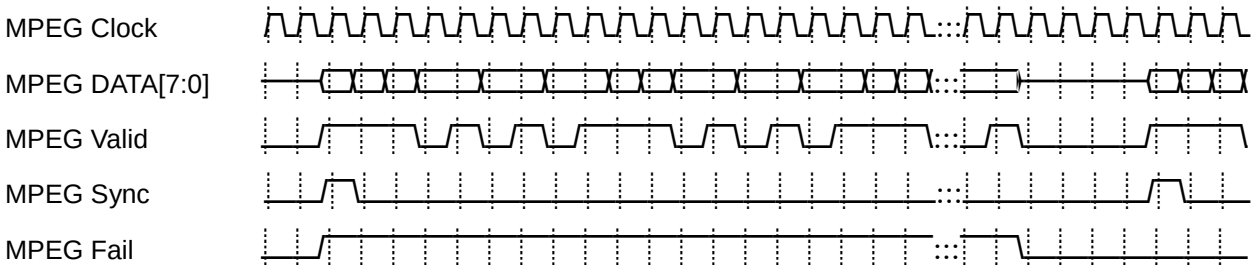

Figure 5 Timing diagram with gapped MPFRM in parallel mode.

The bit-ordering of the MPEG-2 data, polarities of MPEG Clock, MPEG Sync, MPEG Valid, and MPEG Fail, styles of the sync byte and MPEG Clock, gap between consecutive output bytes, output data rate, and pin driving capabilities are all configurable as described in Table 9.

5.10. Serial Output Interface

By programming the register *mpeg_ser_mode* to 1, the payloads of MPEG-2 transport stream will be output serially on pins HOST_B11 or HOST_B4 for IT9130. In this mode, MPEG Sync can be asserted at the first bit or first 8 bits of the payload of the transport stream, depending on whether the register *mssync_len* is programmed to 0 or 1. Furthermore, there can be gaps in MPEG Valid, but there can be no gaps within the 8

bits of a byte. Example timing diagrams of MPEG Valid without and with gaps are shown in Figure 6 and Figure 7, respectively. Finally, same as in the parallel mode, for any error packet, MPEG Fail is asserted throughout the entire duration of any erroneous transport stream packet.

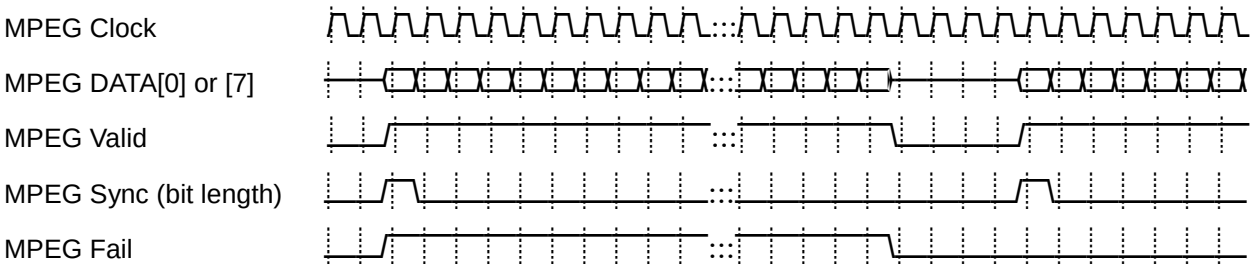

Figure 6 Timing diagram of continuous MPEG Valid signal in serial mode

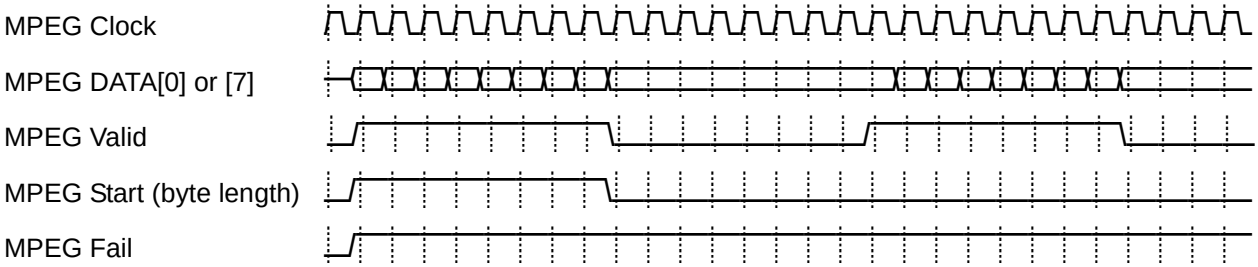

Figure 7 Timing diagram of gapped MPEG Valid signal in serial mode.

The input data pin, endianness of the serial output, polarities of MPEG Clock, MPEG Sync, MPEG Valid, and MPEG Fail, styles of the sync byte and MPEG Clock, gap between consecutive output bytes, output data rate, and pin driving capabilities are all configurable as described in Table 9.

5.11. Serial and Parallel Input Interface (at USB2.0 mode)

By programming the registers *mpeg_ser_mode*, *mpeg_par_mode*, *reg_tsip_en, and reg_tsis_en*, the MPEG2 TS interface of IT9130 becomes an input interface for accepting a secondary MPEG2 TS stream and forwarding to the PC through the USB2.0 interface. Both serial and parallel data are supported for receiving the secondary MPEG2 TS stream. An example timing diagram for the IT9130 MPEG2 TS serial input interface is shown in Figure 8.

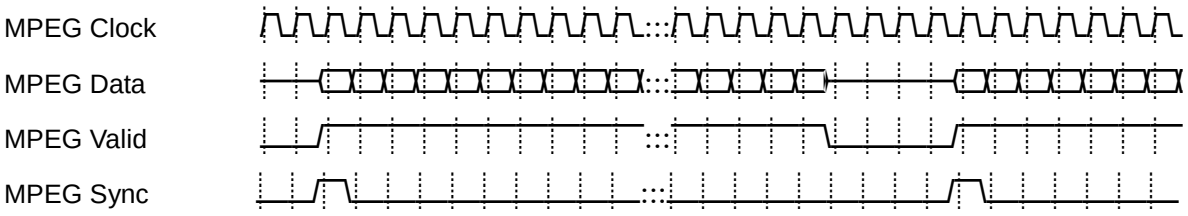

Figure 8 Timing diagram of the IT9130 MPEG-2 TS serial input interface.

Furthermore, by programming the register *reg_ts_capt_bg_sel* to 0, MPEG Sync of IT9130 can be wired to the ground if MPEG Sync is not available from the MPEG TS stream source.

6. Control Protocols

6.1. General

IT9130 uses the payload of each interface to hold IT9130 Proprietary Protocol packets, either a command or a reply. Commands are issued from the host (e.g., PC) to the device (IT9130), while replies are generated from the device to the host.

6.2. 2-Wire Bus Interface

In the IT9130 2-wire bus, a command packet starts with a 2-wire bus start condition, followed by the 2-wire bus address with a zero as the write flag, followed by the packet payload (from the host to the device). A reply packet starts with a 2-wire bus start condition, followed by the 2-wire bus address with a one as the read flag, followed by the packet payload (from device to host). The packet payload is used to hold the command or reply of the IT9130 Proprietary Protocol packets. Note that the 2-wire bus master needs to poll the device using the 2-wire bus read cycle for the reply packet.

6.3. USB Interface

For USB interface, please refer to the section 7.2.

7. Interface

7.1. Using the 2-Wire Interface

IT9130 (IT9133/IT9137) provides a 2-wire interface for communicating with the host or the second IT9130 IC for diversity receiving or dual-channel applications. In general the 2-wire interfaces are controlled through registers. Function calls for using the 2-wire interfaces are provided in the IT9130 API.

TheIT9130 2-wire Interface uses, respectively, pins IOSDA for the serial data and IOSCL for the serial clock. The bus address of the 2-Wire Interface is determined by the strapping pins GPIOH6 and GPIOH5. When IT9130 is first powered up, the RESETN pin should be held low. As the RESETN pin transitions from low to high, the logic level of the strapping pins GPIOH6 and GPIOH5 are latched to determine the 2-wire bus address, as shown in Table 11. For IT9130, the logic level of the strapping pins GPIOH6 and GPIOH5 also determines its operation mode, as described in Table 5

Table 11 IT9130 2-wire bus address mapping table.

{GPIOH6,GPIOH5} at strapping	2-Wire Bus Address
00	0x38
01*	0x3A
10	0x3C
11	0x3E

Note: {GPIOH6, GPIOH5} = {1, 0} address is only used as the salve device for DCA/dual-channel applications

The IT9130 2-wire Interface supports both read and write operations. The circuit works as a slave transmitter in the read operation mode and slave receiver in the write operation mode. The write and read operations of the 2-wire host interface are shown in Figure 9 and Figure 10, respectively. The legends used therein are listed in Table 12.

S	DEVICE ADDRESS	W	A	DA0	A	DA1	A	DA2	A	……..	A	DAn	A	P

Figure 9 2-wire bus write operation.

S	DEVICE ADDRESS	R	A	RDA0	A	RDA1	A	RDA2	A	……..	A	RDAn	NA	P

Figure 10 2-wire bus read operation

Table 12 Legends used in Figure 9 and Figure 10.

Legend	Description
S	Start condition
W	Write (=0)

P	Stop condition
R	Read (=1)
A	Acknowledge
NA	Not acknowledge
DAn	write data Data0, Data1……..Datan
RDAn	Read return data Data0,Data1…Datan

As shown in Figure 9, in the write operation mode (IT9130 acts as a slave receiver), each data byte definition need to reference to protocol control section. From hardware point of view, 2-wire module hardware does not parse any data bytes for information. It just moves data from the 2-wire bus to the internal mailbox. Similarly, in the read operation mode (IT9130 acts as a slave transmitter) the hardware moves the data in the internal mailbox to the 2-wire bus.

The registers `reg_sel_thirdi2c` (0xF104[2]) and `reg_sel_tuner` (0xF104[3]) must both be programmed to 0 before using the 2-wire Interface.

The transmission time per bit of the IT9130 2-wire interfaces can be adjusted by programming the register `reg_one_cycle_counter_tuner` (0xF103). Smaller value of the register represents *faster* data rate. The transmission time per bit is in units of 400ns, thus when `reg_one_cycle_counter_tuner` is programmed to 0x14 the resulting data rate is roughly 125 Kbit/second.

7.2. USB Interface

7.3. Endpoints

IT9130 (IT9135/IT9137) USB 2.0 interface includes Endpoint 0, which is the default endpoint defined in the USB 2.0 specification, and Endpoint 3, which is an interrupt channel.

7.4. USB Control Protocol

7.5. Default Endpoint

Endpoint 0 is the same as defined in the USB 2.0 standard.

7.6. Control Messages

Proprietary control messages are sent through a command-and-reply model for various control functions, including, among other things, boot-strapping, firmware downloading, frequency tuning, performance monitoring, and infra-red control.

7.7. Data Messages

Data messages are used for convey MPEG-2 transport stream and analog video stream received by IT9130.

7.8. EEPROM Format

An external EEPROM is used for storing system parameters in DVBT receivers using IT9130. Most strings

and parameters in the USB descriptors are configurable in the EEPROM, including:

Device descriptors: vender ID (VID), product ID (PID), device release number, manufacturer string index, product string index, serial number string index, configuration characteristics (self-powered, remote wake-up, …etc.), max power consumption, interrupt endpoint (Endpoint 3) polling interval.

Strings: the string description of the manufacturer, the product and the serial number. These strings are defined in the USB 2.0 standard.

The EEPROM format is given in Table 13.

Table 13 IT9130 EEPROM format.

Byte Offset	0	1	2	3	4	5	6	7
0x00	CFG Checksum		CFG Length	USP Offset	0x00	0x00	0x00	0x00
0x08	VID		PID		REV		MSI	PSI
0x10	SNI	CNF	CLK Detect	PWR11	PWR20	IPI		Reserved
0x18	IR mode	Production #		Group #		Date		
0x20	Date (continued)			Daily Serial #				
0x28	Reserved							
0x30	Selective suspend	TS mode	Mpeg2 2-wire bus Address	Suspend mode	IR remote type			
0x38					Tuner ID1			
0x40	Reserved							
0x48					Tuner ID2			
0x50						format Rev.	Reserved	Reserved
0x58 | 0xF8	USB String Pool							

where:

CFG Checksum: Checksum for configuration block, i.e. from offset 2 to end of USB strings, including

the last two zeros of the USB strings. Checksum is the remainder of the sum of all bytes (consider a byte as an unsigned char) divided by 65536. That is, in C-Language pseudo code:

```
unsigned short checksum(void)
{
    unsigned short sum = 0;
    int i;
    // No need to do divide operation for remainder, since it will
    for (i=2; i<2+ CFG Length; i++) sum += image[i];
    return sum;
}
```

CFG Length: Length for configuration block, including this length byte. i.e. from offset 2 to end of USB strings, including the last two zeros of the USB strings.

USP Offset: USB String Pool Offset, typical value is 0x58, as the example above

VID: little endian USB vendor ID

PID: little endian USB product ID

REV: little endian USB device revision number in BCD format

MSI: manufacturer description string index

PSI: product description string index

SNI: serial number string index

CNF: configuration characteristics for USB:
Bit 7: Must be 1, as defined in USB specifications
Bit 6: Self-power indicator (1 for self-powered, 0 for bus-powered)
Bit 5: Remote wakeup indicator
Bit 4-0: not used in our applications.

CLK Detect: USB PHY clock detection setting

0b00000000: OFF 0b00000001:ON

PWR11: max device power consumption for USB 1.1

PWR20: max device power consumption for USB 2.0

IPI: interrupt endpoint (Endpoint 3) polling interval

IR mode: 0: IR is disabled, 1: HID mode, 5: Raw mode

Production #, Group#, Date and Daily Ser#: These construct the 24-byte serial number.

TS mode: stand alone (0), DCA+PIP (1) , DCA only (2), PIP only (3).

Mpeg2 2-wire bus Addr:: Slave demodulator 2-Wire bus address used in dual-TS input mode

Suspend mode: Disable (0) or Enable (1).

IR remote type: NEC (0), or RC6 (1), RC5 (2)

XTALType1: Crystal frequency of the master chip.

1: 20480 KHz

2: 12000 KHz

Tuner ID1: Tuner ID (script id) of the master chip

XTALType2: Crystal frequency of the slave chip.

1: 20480 KHz

2: 12000 KHz

Tuner ID2: Tuner ID (script id) of the slave chip

Format Rev.: EEPROM format revision

7.9. IR Interface

IT9130 supports three IR remote protocols: NEC, RC5, and RC6. The IR function can be enabled or disabled and the IR protocol can be selected by appropriately setting the corresponding fields in the external EEPROM.

The IR function can be configured into two modes: the USB HID (Human Interface Devices) mode and raw code mode. In the USB HID mode, IT9130 is considered as a USB composite device with HID. It uses endpoint 3 as the HID interrupt endpoint. In this mode, IT9130 translates raw IR codes into HID codes according to a translation table. The HID translation table can be downloaded into IT9130 by the USB driver via the memory write protocol. In the raw code mode, IT9130 sends IR raw codes (decoded according to NEC, RC5, or RC6 protocol, but not transformed into an HID code) via control command and reply messages.

7.10. The Function Key and Alternative Keys

The function key (FN) is used to create alternative key sequences for a remote control. When FN of a remote control is pressed, an alternative key sequence is initiated, and any keys pressed are considered “alternative” if they are pressed within a predefined expiration time after the previous key press. This design enables a remote control with fewer keys (buttons) to almost double its “effective” number of keys.

7.11. IR Table

The IR table is a contiguous block of memory in IT9130 used for storing the mapping between IR remote control keys and HID keys. This table can be described using the following C program segment.

```
#define  MAX_IR_N_KEYS  50
typedef struct {
  byte  ir[4];
  byte  hid[3];
} IRKey_t;

USHORT IR_toggle_mask;
byte  IR_nKeys;
byte  IR_FN_expire_time;
byte  IR_Repeat_period;
IRKey_t  ir_table[MAX_IR_N_KEYS];
```

The above information is downloaded from the IT9130 driver via the memory write protocol. All the above information is lined up in a contiguous memory section, in the order appeared above. The memory location is fixed.

7.11.1.1. MAX_IR_N_KEYS

MAX_IR_N_KEYS is maximum number of IR keys. The values cannot exceed 50.

7.11.1.2. IR_toggle_mask

`IR_toggle_mask` is an unsigned short mask, having only one bit (the toggle bit) set to zero and all other bits set to one. The mask is to be applied to the two-byte IR key code to mask out the toggle bit. In the RC6 protocol, the key code can be defined by the vendor. For example, Microsoft chooses the first bit of the first byte as the toggle bit.

7.11.1.3. IR_nKeys

`IR_nKeys` is the number of IR keys, including the function key and alternative keys.

7.11.1.4. IR_FN_expire_time

`IR_FN_expire_time` is the function key expiration time (also known as the alternative key sequence expiration time) in the unit of 20ms. The default value is 62, i.e., 1240ms.

7.11.1.5. IR_Repeat_period

`IR_Repeat_period` is the repetition period of IR remote control buttons. When an IR remote control button is pressed and held, it can send out the repeat key or the signal key repetitively. Due to the unreliable communications characteristics of IR and the HID design, we should provide fixed-rate repeated signal. This parameter defines the repetition period to send out the same key again in units of 20ms. The default value is 25, i.e., 500ms.

7.11.1.6. Ir_table[ ]

`Ir_table[]` is an array of data structures of type `IRKEY_t` for storing the IR-to-HID code translation table. There should be one element for each key and alternative key, thus the number of elements in `Ir_table[]` should be equal to the number of keys and alternative keys of the remote control. .

The `Ir_table[]` entry for each IR remote control key has 7 or 14 bytes. It has 7 bytes if the key does not support the second or alternate key function and 14 bytes if the key supports the second key function. The bytes of each table entry are described in Table 14 and Table 15.

Table 14 The entry in `Ir_table` without the second key function.

IR Scan Key				**HID Key**		
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6

Table 15 The entry in `Ir_table` with the second function key.

IR Scan Key				**HID Key**		
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6

Must be set to 0				**HID Key**		
Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte13

Bytes 0 to 3 are the IR scan-key for the NEC protocol or Philips RC-6 Mode 6A protocol.

NEC IR protocol:

In general, byte 0 is the address of the remote control and byte 1 is the inverted value of byte 0. In some cases, however, byte 0 and byte 1 need not to be complements of each other. Byte 2 is data field and byte 3 is the inverted value of byte 2. An example is given as follows:

Address	~Address	Data	~Data	HID key		
0xAA	0x55	0x12	0xED	0x13	0x01	0x40

Philips RC-6 Mode 6A protocol (OEM):

Bytes 0 to 3 are defined by OEM. In IT9130, bytes 0 and 1 are the Long Customer Code and bytes 2 and 3 is the two-byte information field. An example is given as follows:

Long Customer Code		Information Field		HID Key		
Msb15… High byte	……Lsb0 Low byte	Data High byte	Data Low byte	0x13	0x01	0x40

Byte 4 is the HID Usage ID. It is a mapping to a key of the keyboard.

Byte 5 defines the LeftCtrl to RightGUI eight keys function as Table 16.

Table 16 Definition of Byte 5 of IR Table.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RightGUI	RightAlt	RightShift	RightCtrl	LeftGUI	LeftAlt	LeftShift	LeftCtrl

Byte 6: The definition of Byte 6 is given in Table 17. Bit 0 of Byte 6 is the Sys Request key of the keyboard. Bit 6 of should be set to 1 if this entry supports the second key function and the next 7 bytes describes the second key and 0 otherwise. Bit 7 should be set to 1 if this table entry is for the function key, and 0 otherwise.

Table 17 Definition of Byte 6 of IR Table.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Function Key	Second Key Supported	Reserved					SysRq

Bytes 7 to 10 must be all zeros if the second key is supported.

Byte 11 is HID Usage ID for the second key.

Bytes 12 and 13 should be set to 0 if the second key is supported.

Example IR Table entries for a key, which maps to CTRL-P, on remote controls using the NEC and RC6 protocols are given in Table 18 and Table 19, respectively. It is assumed that the second key function is

supported by this remote control and the alternative key is “3.”

With Table 18, this remote control supports the second key function, with the second key being “3.”

Table 18 IR Table entries for a key of a remote control using the NEC protocol that maps to CTRL-P.

Address	~Address	Data	~Data	HID key		
0xAA	0x55	0x06	0xF9	0x13	0x01	0x40

Address	~Address	Data	~Data	HID key		
0x00	0x00	0x00	0x00	0x20	0x00	0x00

With Table 19, this remote control supports the second key function, with the second key being “3.”

Table 19: IR Table entries for a key of a remote control using the RC6 protocol that maps to CTRL-P.

Long Customer Code		Information Field		HID key		
0x80	0x0F	0x04	0x16	0x13	0x01	0x40

Long Customer Code		Information Field		HID key		
0x00	0x00	0x00	0x00	0x20	0x00	0x00

7.12. Generating IR Table

The *ITE IR Remote Key Mapping Generator* can be used to generate the IR Table for downloading.

8. Boot Description

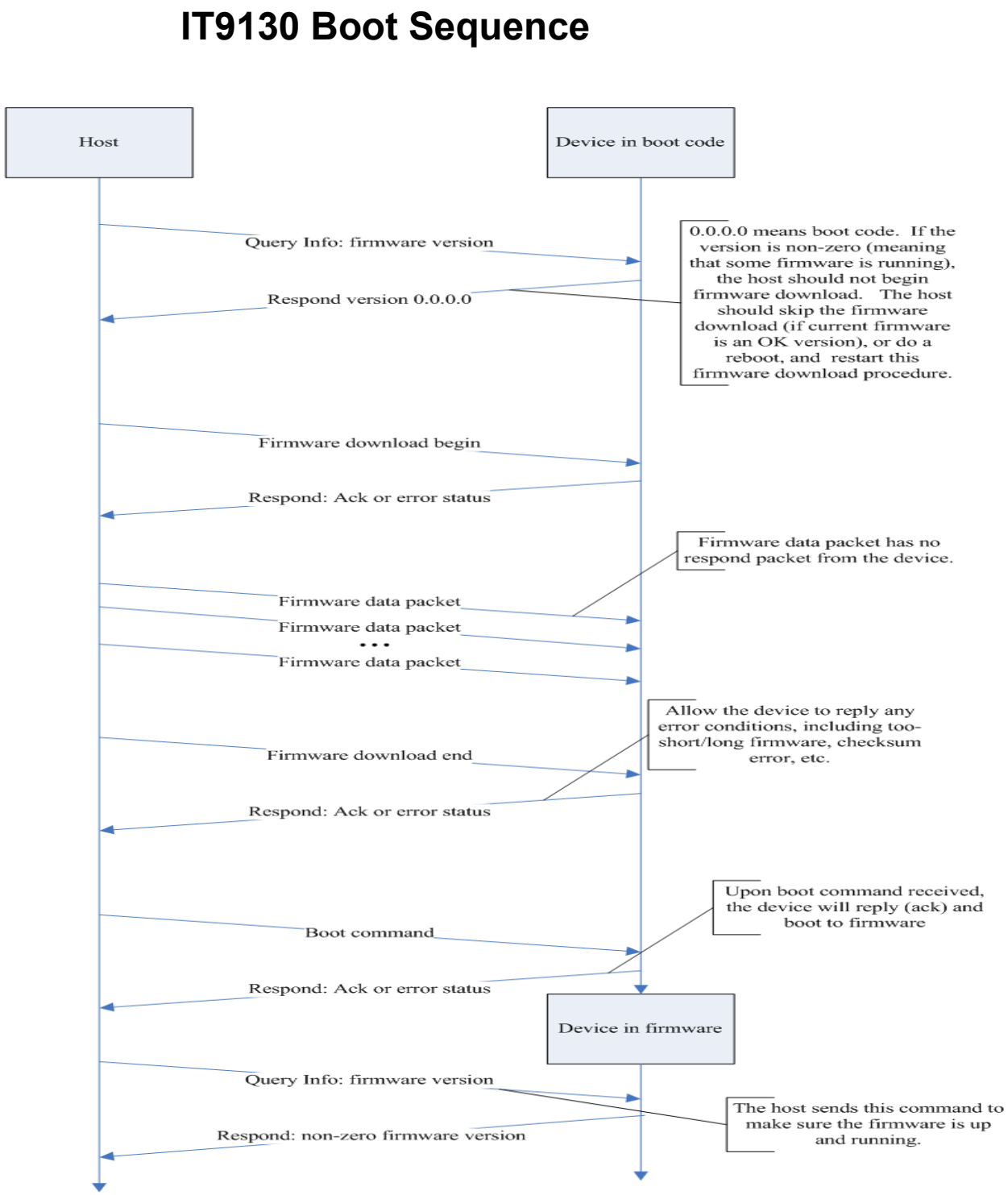

Figure 11 IT9130 boot sequence.

The above picture shows the procedure (protocol) used to boot IT9130. In summary, the Protocol consists of command packets (from the host to the device), and the reply packets (from the device to the host).

8.1. Payload Embedded in the USB 2.0 Mode

In USB 2.0 mode, control messages are used to carry the command and reply packets. The payload is the data field in an USB packet.

8.2. Payload Embedded in the 2-wire Bus Mode

In the 2-wire bus, the host needs a one-byte header consisting of a 7-bit address to identify the device, and a 1-bit read/write flag to specify if this packet is a write or a read. The body (every byte except for the first header byte) of a write-packet is used for the command packet, while the body of a read-packet is used for the reply packet. Also, polling is required to 'read' the reply packet, since both read-packet and write-packet are initiated by the host.

9. Performance and Status Monitoring

IT9130 provides a complete set of API functions for monitoring the performance and status of the demodulator. The parameters that can be monitored and derived include TPS and MPEG-2 lock detection, bit error rates and packet error rates, signal quality, and signal strength.

9.1. Checking the Presence of a DVB-T Signal (TPS Lock)

The user can use `Demodulator_isTpsLocked()` to check if a DVB-T signal is present. An example is given below.

```
IT9130 it9130;
Byte chip = 0;                          // Get modulation from chip 0 (first chip).
Bool isTpsLocked;
Dword error = Error_NO_ERROR;

error = Demodulator_isTpsLocked ((Demodulator*) &it9130, chip, &isTpsLocked);
if (!error)
{
     if (isTpsLocked)
          printf (“TPS is locked”);
     else
          printf (“TPS isn’t locked”);
}
```

9.2. Retrieving Transmission Parameters

The transmission parameter signaling (TPS) word detected from the received signal can by obtained by calling `Demodulator_getChannelModulation()`. This function retrieves the transmission parameters from registers listed in Table 20. These registers should be accessed with the `processor` argument set to `Processor_OFDM`.

Table 20 TPS information registers.

Parameters	Description	Address[msb:lsb]
Constellation	2’b00: QPSK 2’b01: 16QAM 2’b10: 64QAM	0xF903[1:0]
FFT Mode	2’b00: 2K mode 2’b01: 8K mode 2’b10: 4K mode	0xF900[1:0]
Guard Interval	2’b00: 1/32	0xF901[1:0]

	2'b01: 1/16 2'b10: 1/8 2'b11: 1/4	
Alpha	3'b000: Non-hierarchical 3'b001: alpha=1 3'b010: alpha=2 3'b011: alpha=4	0xF902[2:0]
Channel BW	2'b00: 6MHz 2'b01: 7MHz 2'b10: 8MHz 2'b11: 5MHz	0xF904[1:0]
Hierarchical Select	1b'01: Decode HP Stream 1b'00: Decode LP Stream	0xF905[0:0]
Code rate of high priority stream	3'b000: 1/2 3'b001: 2/3 3'b010: 3/4 3'b011: 5/6 3'b100: 7/8	0xF906[2:0]
Code rate of low priority stream	3'b000: 1/2 3'b001: 2/3 3'b010: 3/4 3'b011: 5/6 3'b100: 7/8	0xF907[2:0]
In-depth interleaving information	1'b0: Native interleaving 1'b1: In-depth interleaving	0xF908[0]

An example is given below.

```
IT9130 it9130;
Byte chip = 0;                          // Get modulation from chip 0 (first chip).
ChannelModulation *info;
Dword error = Error_NO_ERROR;

ChannelModulation *info;
error = Demodulator_getChannelModulation((Demodulator*) &it9130, chip, info);
if (!error)
```

```
{
    if (info->bandwidth == Bandwidth_6M)
        printf ("Bandwidth is 6MHz\n");
    else if (info->bandwidth == Bandwidth_7M)
        printf ("Bandwidth is 7MHz\n");
    else if (info->bandwidth == Bandwidth_8M)
        printf ("Bandwidth is 8MHz\n");
    else if (info->bandwidth == Bandwidth_5M)
        printf ("Bandwidth is 5MHz\n");
    else
        printf ("Get Bandwidth failed");


    if (info->constellation == Constellation_QPSK)
        printf ("Constellation is QPSK\n");
    else if (info->constellation == Constellation_16QAM)
        printf ("Constellation is 16QAM\n");
    else if (info->constellation == Constellation_64QAM)
        printf ("Constellation is 64QAM\n");
    else
        printf ("Get Constellation failed");

    printf("Frequency is %dkHz\n", info->frequency);

    if (info->hierarchy == Hierarchy_NONE)
        printf ("Signal is non-hierarchical\n");
    else if (info->hierarchy == Hierarchy_ALPHA_1)
        printf ("Signalling format uses alpha of 1\n");
    else if (info->hierarchy == Hierarchy_ALPHA_2)
        printf ("Signalling format uses alpha of 2\n");
    else if (info->hierarchy == Hierarchy_ALPHA_4)
        printf ("Signalling format uses alpha of 4\n");
    else
        printf ("Get Hierarchy failed");

    if (info->highCodeRate == CodeRate_1_OVER_2)
        printf ("Signal uses FEC coding ratio of 1/2\n");
```

```
    else if (info->highCodeRate == CodeRate_2_OVER_3)
        printf ("Signal uses FEC coding ratio of 2/3\n");
    else if (info->highCodeRate == CodeRate_3_OVER_4)
        printf ("Signal uses FEC coding ratio of 3/4\n");
    else if (info->highCodeRate == CodeRate_5_OVER_6)
        printf ("Signal uses FEC coding ratio of 5/6\n");
    else if (info->highCodeRate == CodeRate_7_OVER_8)
        printf ("Signal uses FEC coding ratio of 7/8\n");
    else if (info->highCodeRate == CodeRate_NONE)
        printf ("None, NXT doesn't have this one\n");
    else
        printf ("Get HighCodeRate failed");

    if (info->interval == Interval_1_OVER_32)
        printf ("Guard interval is 1/32 of symbol length\n");
    else if (info->interval == Interval_1_OVER_16)
        printf ("Guard interval is 1/16 of symbol length\n");
    else if (info->interval == Interval_1_OVER_8)
        printf ("Guard interval is 1/8 of symbol length\n");
    else if (info->interval == Interval_1_OVER_4)
        printf ("Guard interval is 1/4 of symbol length\n");
    else
        printf ("Get Interval failed");

    if (info->lowCodeRate == CodeRate_1_OVER_2)
        printf ("Signal uses FEC coding ratio of 1/2\n");
    else if (info->lowCodeRate == CodeRate_2_OVER_3)
        printf ("Signal uses FEC coding ratio of 1/2\n");
    else if (info->lowCodeRate == CodeRate_3_OVER_4)
        printf ("Signal uses FEC coding ratio of 1/2\n");
    else if (info->lowCodeRate == CodeRate_5_OVER_6)
        printf ("Signal uses FEC coding ratio of 5/6\n");
    else if (info->lowCodeRate == CodeRate_7_OVER_8)
        printf ("Signal uses FEC coding ratio of 7/8\n");
    else if (info->lowCodeRate == CodeRate_NONE)
        printf ("None, FEC doesn't have this code rate\n");
    else
```

```
        printf ("Get LowCodeRate failed");

    if (info->priority == Priority_HIGH)
        printf ("DVB-T - identifies high-priority stream\n");
    else if (info->priority == Priority_LOW)
        printf ("DVB-T - identifies low-priority stream\n");

    if (info->transmissionMode == TransmissionMode_2K )
        printf ("OFDM frame consists of 2048 subcarriers (2K FFT mode)\n");
    else if (info->transmissionMode == TransmissionMode_8K )
        printf ("OFDM frame consists of 8192 subcarriers (8K FFT mode)\n");
    else if (info->transmissionMode == TransmissionMode_4K )
        printf ("OFDM frame consists of 4096 subcarriers (4K FFT mode)\n");
    else
        printf ("Get TransmissionMode failed");

}
```

9.3. MPEG2 Lock

The user can use demodulator_`isMpeg2Locked()` to check if the DVB-T signal is of sufficient quality such that the transport stream can be correctly decoded (locked). An example is given below.

```
IT9130 it9130;
Byte chip = 0;                          // Get modulation from chip 0 (first chip).
Bool isLocked;
Dword error = Error_NO_ERROR;

error = Demodulator_isMpeg2Locked ((Demodulator*) &it9130, chip, &isLocked);
if (!error)
{
    if (isLocked)
        printf (“MPEG is locked”);
    else
        printf (“MPEG isn’t locked”);
}
```

9.4. Signal Quality

The user can use `Demodulator_getSignalQualityIndication ()`to get a signal quality indicator that ranges from 0 to 100 (100 is the best quality). An example is given below.

```
IT9130 it9130;
Byte chip = 0;                          // Get modulation from chip 0 (first chip).
Byte Quality;
Dword error = Error_NO_ERROR;

error = Demodulator_getSignalQualityIndication ((Demodulator*) &ganymede, chip, &Quality);
if (!error)
{
     printf (“The signal quality is %x”, Quality);
}
```

9.5. Signal Strength

The use can use `Demodulator_getSignalStrengthIndication ()` to get a signal strength indicator that ranges from 0 to 100 (100 is the strongest). An example is given below.

```
IT9130 it9130;
Byte chip = 0;                          // Get modulation from chip 0 (first chip).
Byte Strength;
Dword error = Error_NO_ERROR;

error = Demodulator_getSignalStrengthIndication ((Demodulator*) &ganymede, chip, &Strength);
if (!error)
{
     printf (“The signal strength is %x”, Strength);
}
```

9.6. Signal Strength (dBm)

The use can use `Demodulator_getSignalStrengthDbm()` to get the signal strength in unit of dBm. An example is given below.

```
IT9130 it9130;
Byte chip = 0;                          // Get modulation from chip 0 (first chip).
Long strengthDbm;
Dword error = Error_NO_ERROR;
```

```
error = Demodulator_getSignalStrengthDbm ((Demodulator*) &it9130, chip, &strengthDbm);
```

9.7. Post-Viterbi Bit Error Rate

The user can use `Demodulator_getPostVitBer()` to get the post-Viterbi bit error rate. An example is given below.

```
IT9130 it9130;
Byte chip = 0;                          // Get modulation from chip 0 (first chip).
Dword postErrorCount;
Dword postBitCount;
Word aboutCount;
Doubld postVitBER;
Dword error = Error_NO_ERROR;

error = Demodulator_getPostVitBer ((Demodulator*) &it9130, chip, &postErrorCount, &postBitCount,
&aboutCount);
if (!error)
{
     postVitBER = postErrorCount / postBitCount;
     printf (“The post viterbi BER is %f”, postVitBER);
     printf (“The about count is %x”, aboutCount);
}
```

9.8. Signal-to-Noise Ratio

The user can use Demodulator_getSNR( ) to get the signal to noise ratio. An example is given below.

```
IT9130 it9130;
Byte chip = 0;                          // Get modulation from chip 0 (first chip).
Byte snr = 0;
Dword error = Error_NO_ERROR;

error = Demodulator_getSNR ((Demodulator*) &it9130, chip, &snr);
if (!error)
{
     printf("Signal SNR = %d(DBm)\n", snr);
```

```
}
```

9.9. Carrier Frequency Offset

The registers for reporting the carrier frequency offset are listed in

Table *21* and can be read through the host 2-wire bus interface using the API functions.

Table 21 Registers for reporting the carrier frequency offset.

Register Name	Address	Valid Bits
r_bfs_fcw_q_22_16	0xF1B5	7
r_bfs_fcw_q_15_8	0xF1B4	8
r_bfs_fcw_q_7_0	0xF1B3	8
p_reg_bfs_fcw_22_16	0xF1A5	7
p_reg_bfs_fcw_15_8	0xF1A4	8
p_reg_bfs_fcw_7_0	0xF1A3	8

The carrier frequency offset in Hz is given by

$$\text{Carrier Frequency Offset (Hz)} = (f_{cur} - f_{base}) \times f_{adc} \times 2^{-23},$$

where

$$f_{cur} = \text{r\_bfs\_fcw\_q\_22\_16} \times 2^{16} + \text{r\_bfs\_fcw\_q\_15\_8} \times 2^{8} + \text{r\_bfs\_fcw\_q\_7\_0},$$

$$f_{base} = \text{p\_reg\_bfs\_fcw\_22\_16} \times 2^{16} + \text{p\_reg\_bfs\_fcw\_15\_8} \times 2^{8} + \text{p\_reg\_bfs\_fcw\_7\_0},$$

and f_{adc} is the nominal ADC sampling frequency in Hz. The carrier frequency offset in subcarrier spacings is given by

$$\text{Carrier Frequency Offset (Sub-carrier Spacings)} = \text{Carrier Frequency Offset (Hz)} \times N_{FFT} \times T_e,$$

Where N_{FFT} = 2048 for 2K mode and 8192 for 8K mode, and T_e is the elementary period defined in ETSI EN 300 744, which equals $(7/64) \times 10^{-6}$ for 8 MHz DVB-T channels, $1/8 \times 10^{-6}$ for 7 MHz DVB-T channels, $(7/48) \times 10^{-6}$ for 6 MHz DVB-T channels, and $(7/40) \times 10^{-6}$ for 5 MHz DVB-T channels.

Appendix A: IT9130 Pin Description

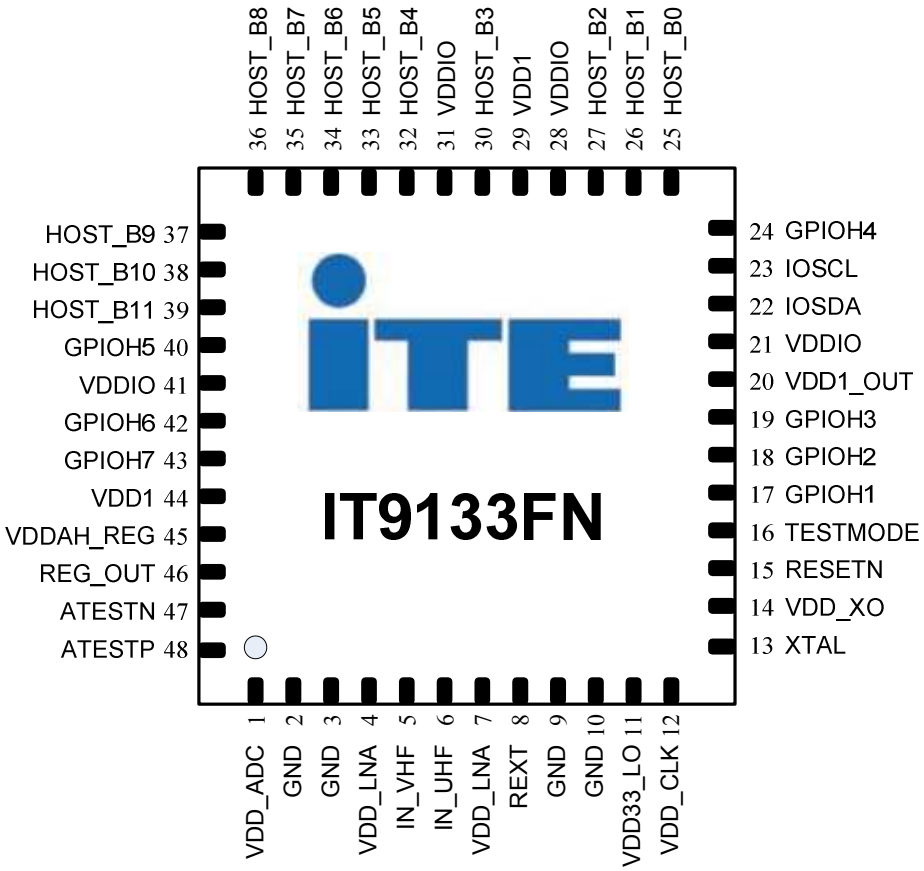

Figure 12 IT9133 pin configuration.

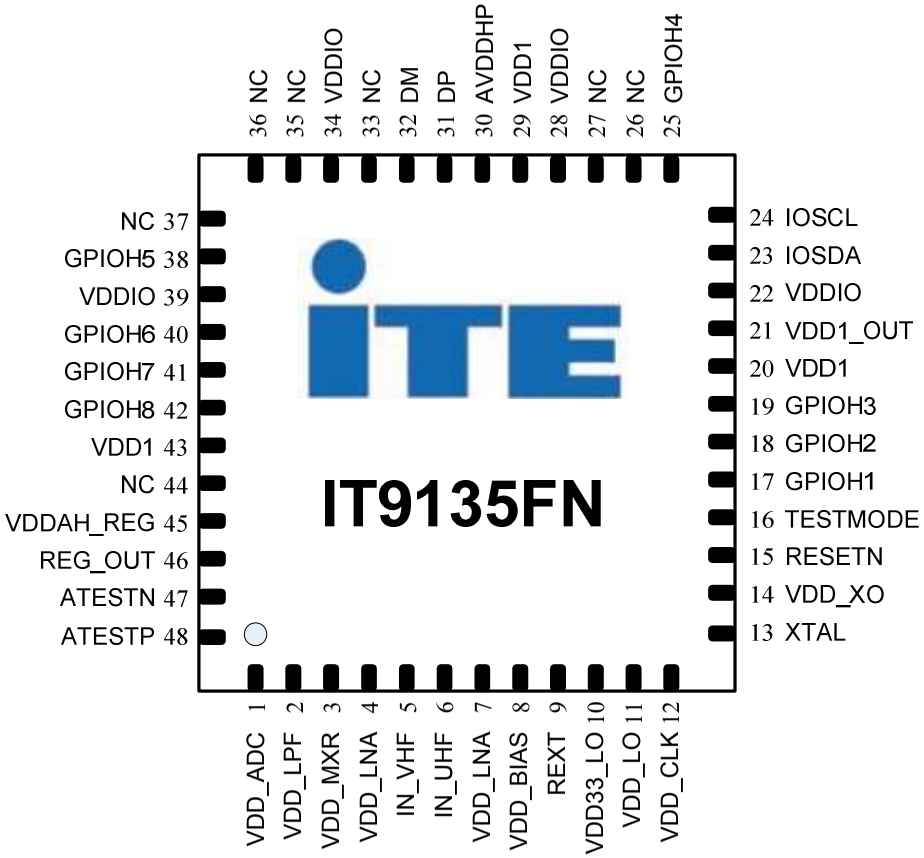

Figure 13 IT9135 pin configuration.

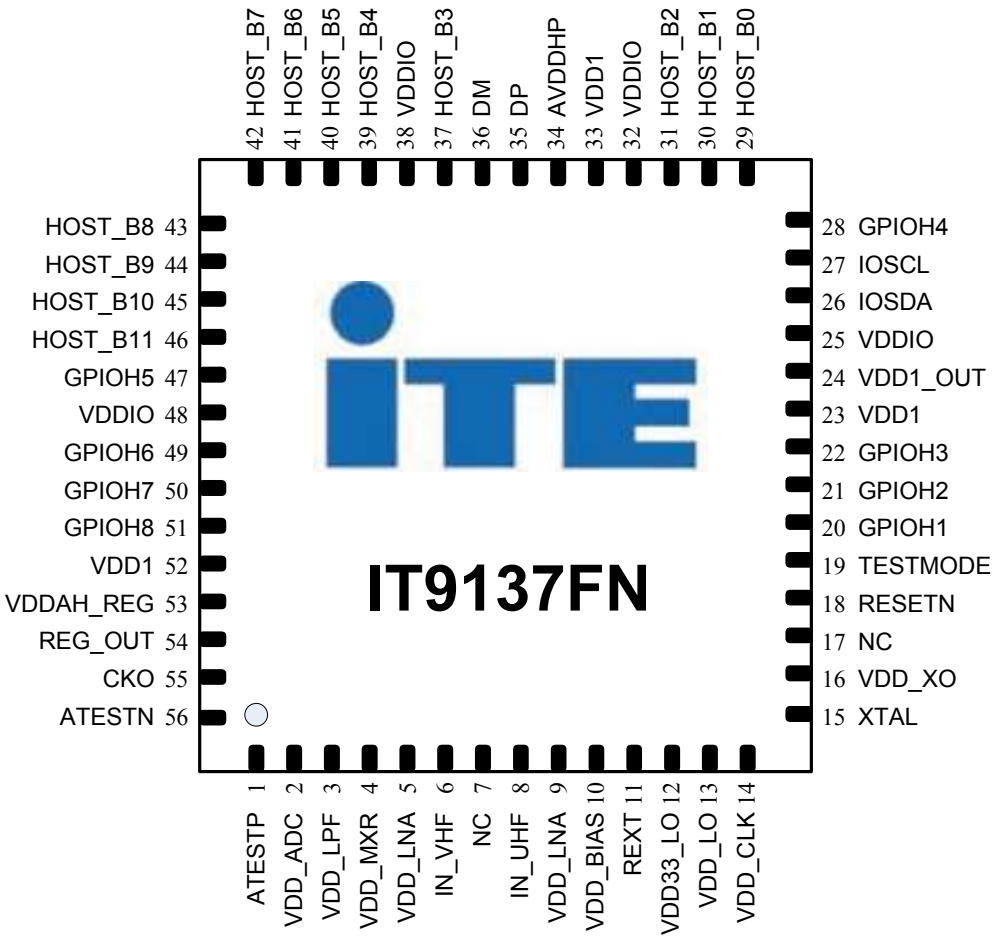

Figure 14 IT9137 pin configuration.

Table 22 Strapping pins sampled at the rising edge of the RESET signal.

Pin Name	**Selection**
{GPIOH4,GPIOH3,GPIOH2,GPIOH1}	Crystal frequency: 0000 crystal = 12 MHz 0001 crystal = 20.48MHz
{GPIOH8,GPIOH7,GPIOH6,GPIOH5}	Mode strapping and 2-wire bus address selection: 00xx TS mode, {GPIOH6, GPIOH5} = 2-wire address[1:0] 0001 Salve device of DCA mode 0101 USB mode

Table 23 List of Host Interface pins

Mode / Pin	**TS Output Mode**	**TS Input Mode**	**Diversity Master Mode**	**Diversity Slave Mode**
HOST_B0	MPEG Fail	MPEG Data[7]	Output Data[3]	Output Clock
HOST_B1	MPEG Sync	MPEG Data[6]	Output Data[2]	Output Valid
HOST_B2	MPEG Valid	MPEG Data[5]	Output Data[1]	Output Data[0]
HOST_B3	MPEG Clock	MPEG Data[4]	Output Data[0]	Output Data[1]
HOST_B4	MPEG Data[0]	MPEG Data[3]	Output Valid	Output Data[2]
HOST_ B5	MPEG Data[1]	MPEG Data[2]	Output Clock	Output Data[3]
HOST_B6	MPEG Data[2]	MPEG Data[1]	Input Data[3]	Input Clock
HOST_B7	MPEG Data[3]	MPEG Data[0]	Input Data[2]	Input Valid
HOST_B8	MPEG Data[4]	MPEG Clock	Input Data[1]	Input Data[0]

HOST_B9	MPEG Data[5]	MPEG Valid	Input Data[0]	Input Data[1]
HOST_B10	MPEG Data[6]	MPEG Sync	Input Valid	Input Data[2]
HOST_B11	MPEG Data[7]	MPEG Fail	Input Clock	Input Data[3]

Appendix B: Register Table

When accessing the following registers, the `processor` argument of `Demodulator_readRegister()` and `Demodulator_writeRegister()`should be set to Processor_OFDM. Registers with attribute RW or RWS are read-write registers. Registers with attribute R or RS are read-only registers.

Table 24 IT9130 register table in OFDM processor.

Address	Register Name	Bits	Attrib	Register Description	Default
0xF006	*reg_p_pwm_rf_if_from_hw*	0	RW	0: PWM source from micro-controller 1: PWM source from hardware	1'h1
0xF05E	*reg_p_pwm_cycle_unit[3:0]*	3:0	RW	Period of PWM = Crystal frequency / (256 x (p_pwm_cycle_unit+1))	4'h1
0xF05F	*reg_p_pwm_en*	0	RW	0: PDM generates control signal to RF/IF AGC 1: PWM generates control signal to RF/IF AGC	1'h0
0xF900	*reg_tpsd_txmod[1:0]*	1:0	R	DVB-T transmission mode (TPS bits S38, S39)	2'h0
0xF901	*reg_tpsd_gi[1:0]*	1:0	R	DVB-T guard interval (TPS bits S36, S37)	2'h0
0xF902	*reg_tpsd_hier[2:0]*	2:0	R	DVB-T hierarchy information (TPS bits S27,S28,S29)	3'h0
0xF903	*reg_tpsd_const[1:0]*	1:0	R	DVB-T constellation (TPS bits S25, S26)	2'h0
0xF904	*reg_bw[1:0]*	1:0	RW	DVB-T channel bandwidth. 00: 6MHz 01: 7MHz 10: 8MHz 11:5MHz	2'h0
0xF905	*reg_dec_pri*	0	RW	High/Low priority stream select. 0: LP 1: HP	1'h0
0xF906	*reg_tpsd_hpcr[2:0]*	2:0	R	Code Rate for HP Stream (TPS bits S30,S31,S32)	3'h0
0xF907	*reg_tpsd_lpcr[2:0]*	2:0	R	Code Rate for LP Stream (TPS bits S33,S34,S35)	3'h0
0xF908	*reg_tpsd_indep*	0	R	In-depth interleaving information (TPS bit S27)	1'h0
0xF1A3	*reg_bfs_fcw[7:0]*	7:0	RW	Carrier frequency control word	8'h0
0xF1A4	*reg_bfs_fcw[15:8]*	7:0	RW		8'h0
0xF1A5	*reg_bfs_fcw[22:16]*	6:0	RW		7'h0
0xF1B3	*fcw_q[7:0]*	7:0	RS	Carrier frequency offset	8'h0
0xF1B4	*fcw_q[15:8]*	7:0	RS		8'h0
0xF1B5	*fcw_q[22:16]*	6:0	RS		7'h0

0xF980	*psb_overflow*	0	RWS	Packet smooth buffer overflow indicator	1'h0
0xF981	*no_modify_tei_bit*	0	RW	0: Enable modification of TEI in TS output 1: Disable modification of TEI in TS output	1'h0
0xF982	*keep_sf_sync_byte*	0	RW	Style of MPEG-2 sync byte 0: MPEG-2 style. The inverted sync byte (0xB8) in DVB-T shall be inverted again so that the output sync byte is always 0x47 as required by MPEG-2. 1: DVB-T style. In this case the FEC module transfers the decoded data to the MPEG-2 TS interface without modification. The sync byte of every eighth super-frame is inverted (0xB8) according to the DVB-T specifications.	1'h0
0xF984	*mpeg_ser_do7*	0	RW	MPEG-2 interface data output at serial mode 0: Output to DATA0 1: Output to DATA1	1'h0
0xF985	*mpeg_ser_mode*	0	RW	MPEG-2 output is serial	1'h0
0xF986	*mpeg_par_mode*	0	RW	MPEG-2 output is parallel	1'h0
0xF987	*psb_empty*	0	RS	Packet smoothing buffer empty indicator	1'h0
0xF988	*ts_not_188*	0	RS	TS error signal	1'h0
0xF989	*mssync_len*	0	RW	For the MPEG serial output mode, select whether MPSTR is asserted only for the first bit or for all bits of the first byte. 0: Asserted for all bits of the first byte 1: Asserted for the first bit only	1'h0
0xF98A	*msdo_msb*	0	RW	For the serial output mode, select the endianness of the serial data output. 0: LSB first 1: MSB first For the parallel output mode, select the bit ordering of the parallel output 0: MPDATA7 is the MSB of TS data 1: MPDATA0 is the MSB of TS data	1'h1
0xF98B	*mpeg_clk_gated*	0	RW	The style of MPCLK 0: Free running. MPCLK is always present 1: Gated. There is no MPCLK when no valid data is available.	1'h0
0xF98C	*mpeg_err_pol*	0	RW	The polarity of MPFAIL 0: Active high (set to 1 to indicate error) 1: Active low	1'h0
0xF98D	*mpeg_sync_pol*	0	RW	The polarity of MPSTR 0: Active high (set to 1 to indicate frame sync) 1: Active low	1'h0
0xF98E	*mpeg_vld_pol*	0	RW	The polarity of MPFRM 0: Active high (set to 1 to indicate valid data) 1: Active low	1'h0

0xF98F	*mpeg_clk_pol*	0	RW	The polarity of MPCLK. 0: Data changes after the rising edge of MPCLK 1: Data changes after the falling edge of MPCLK	1'h0
0xF990	*reg_mpeg_full_speed*	0	RW	Full speed in mpeg interface	1'h0
0xF991	*pid_complement*	0	RW	PID table complement; 0: Pid out if table hit 1: Pid out if table not hit.	1'h0
0xF992	*pid_rst*	0	RWS	Reset PID table	1'h0
0xF993	*pid_en*	0	RW	PID_en	1'h0
0xF994	*pid_index_en*	0	RW	Enable current PID index	1'h1
0xF995	*pid_index[4:0]*	4:0	RWS	PID index	5'h0
0xF996	*pid_dat_l[7:0]*	7:0	RW	PID data register (bit 7-0)	8'h0
0xF997	*pid_dat_h[4:0]*	4:0	RW	PID data register (bit 12-8)	5'h0
0xF999	*sync_byte_locked*	0	RS	Indicate sync_byte_locked	1'h0
0xF99A	*ignore_sync_byte*	0	RW	Don't check sync byte x47	1'h1
0xF99C	*reg_mpeg_vld_tgl*	0	RW	The style of MPFRM 0: MPFRM will be sent out continuously. 1: Allow gaps in between bytes in MPFRM.	1'h0
0xF99D	*reg_mp2_sw_rst*	0	RW	MPEG software reset	1'h1
0xF99E	*psb_en*	0	RW	Packet smooth buffer enable	1'h1
0xF9A1	*lost_pkt_cnt_l[7:0]*	7:0	RS	Number of lost packet (low byte)	8'h0
0xF9A2	*lost_pkt_cnt_h[7:0]*	7:0	RS	Number of lost packet (high byte)	8'h0
0xF9A9	*reg_tpsd_bw_mp2if[1:0]*	1:0	RW	Target DVB-T channel bandwidth for MPCLK frequency control	2'h0
0xF9AA	*reg_tpsd_gi_mp2if[1:0]*	1:0	RW	Target guard interval for MPCLK frequency control	2'h0
0xF9AB	*reg_tpsd_cr_mp2if[2:0]*	2:0	RW	Target code rate for MPCLK frequency control	3'h0
0xF9AC	*reg_tpsd_cons_mp2if[1:0]*	1:0	RW	Target constellation for MPCLK frequency control	2'h0
0xF9AD	*reg_fw_table_en*	0	RW	MPCLK frequency control: 0: The frequency of MPCLK depends on the actual bandwidth and TPS parameters of the current DVB-T channel. 1: The frequency of MPCLK depends on the target values in the registers reg_tpsd_bw_mp2if, reg_tpsd_gi_mp2if, reg_tpsd_cr_mp2if, and reg_tpsd_cons_mp2if.	1'h0

0xF9B0	*reg_packet_gap[7:0]*	7:0	RW	The gap between consecutive 188-byte payloads in units of byte times. Takes values of 0 to 255. Applies to the parallel output mode and serial output mode.	8'h10
0xF9B2	*reg_ts_dat_inv*	0	RW	Serial TS input data polarity inversion. 0: Serial TS input data is not inverted. 1: Serial TS input data is inverted.	1h'0
0xF9B3	*reg_ts_lsb_1st*	0	RW	Endianness of the serial TS input 0: MSB first 1: LSB first	1h'0
0xF9B4	*reg_ts_capt_bg_sel*	0	RW	For serial TS input: 0: MPEG sync is ignored (can be wired to ground). 1: MPEG sync is used.	1'h1
0xF9B7	*reg_ts_sync_inv*	0	RW	Serial TS input sync inversion. 0: MPEG-2 sync is not inverted 1: MPEG-2 sync is inverted.	1'h0
0xF9B8	*reg_ts_vld_inv*	0	RW	Serial TS input valid inversion. 0: MPEG-2 valid is not inverted. 1: MPEG-2 valid is inverted	1'h0
9xF9CC	*reg_tsip_en*	0	RW	0: Disable parallel TS input 1: Enable parallel TS input	1'h0
0xF9CD	*reg_tsis_en*	0	RW	0: Disable serial TS input 1: Enable serial TS input	1'h0
0xF9D9	*reg_clk_sel[1:0]*	1:0	RS	Read back values related to the MPEG output clock rate: 0: clk_phy/8 1: clk_phy/4 2: clk_phy/2 3: clk_phy	2'h0
0xF9DA	*reg_tog_sel[1:0]*	1:0	RS	Read back values related to the MPEG output clock rate: 0: 5/8 1: 6/8 2: 7/8 3: 8/8	2'h0
0xF9E0	*reg_check_tpsd_hier*	0	RW	Enable reg_tpsd_hier for determining output speed 0: Don't care reg_tpsd_hier; 1: Depend on reg_tpsd_hier	1'h0

When accessing the following registers, the processor argument of Demodulator_readRegister() and Demodulator_writeRegister()should be set to Processor_LINK. Registers with attribute RW or RWS are read-write registers. Registers with attribute R or RS are read-only registers.

Table 25 IT9130 register table in LL processor.

Address	Register Name	Bits	Attrib	Register Description	Default
0xD800	*pwron_clk_strap[3:0]*	3:0	RS	Clock strapping information: 0000: Crystal frequency= 12MHz, 0001: Crystal frequency = 20.48MHz,	4'h0
0xD801	*pwron_mode_strap[3:0]*	3:0	RS	Peripheral mode strap info: 00xx: TS mode, I2C_address[1:0], 0001: Salve device of DCA mode 0101: USB mode Others: Reserved	4'h0
0xD806	*reg_ofsm_suspend*	0	RWS	Write 1 to enter suspend mode.	1'h0
0xD808	*wake_int*	0	RWS	External wake up interrupt status	1'h0
0xD809	*reg_top_pwrdw_hwen*	0	RW	Enable hardware suspend function without interrupting MCU 0: Disable 1: Enable	1'h0
0xD80A	*reg_top_pwrdw_inv*	0	RW	Hardware suspend polarity (via GPIOH5)	1'h0
0xD80C	*wake_int_en*	0	RW	Enable external wake up interrupt	1'h0
0xD80D	*pwrdw_int*	0	RWS	Hardware suspend interrupt status	1'h0
0xD81A	*reg_top_clkoen*	0	RW	Enable CLKO output	1'h1
0xD830	*reg_top_padmiscdr2*	0	RW	Bit 0 of output driving control	1'h0
0xD831	*reg_top_padmiscdr4*	0	RW	Bit 1 of output driving control	1'h1
0xD832	*reg_top_padmiscdr8*	0	RW	Bit 2 of output driving control	1'h0
0xD833	*reg_top_padmiscdrsr*	0	RW	MPEG output slew rate control: 0: Default 1: Slew rate boosts	1'h0
0xD8AE	*reg_top_gpioh1_i*	0	RS	GPIOh1 input	1'h0
0xD8AF	*reg_top_gpioh1_o*	0	RW	GPIOh1 output	1'h0
0xD8B0	*reg_top_gpioh1_en*	0	RW	GPIOh1 output enable 1: Output mode 0: Input mode	1'h0
0xD8B1	*reg_top_gpioh1_on*	0	RW	GPIOh1 enable	1'h0

0xD8B2	*reg_top_gpioh3_i*	0	RS	GPIOh3 input	1'h0
0xD8B3	*reg_top_gpioh3_o*	0	RW	GPIOh3 output	1'h0
0xD8B4	*reg_top_gpioh3_en*	0	RW	GPIOh3 output enable 1: Output mode 0: Input mode	1'h0
0xD8B5	*reg_top_gpioh3_on*	0	RW	GPIOh2 enable	1'h0
0xD8B6	*reg_top_gpioh2_i*	0	RS	GPIOh2 input	1'h0
0xD8B7	*reg_top_gpioh2_o*	0	RW	GPIOh2 output	1'h0
0xD8B8	*reg_top_gpioh2_en*	0	RW	GPIOh2 output enable 1: Output mode 0: Input mode	1'h0
0xD8B9	*reg_top_gpioh2_on*	0	RW	GPIOh2 enable	1'h0
0xD8BA	*reg_top_gpioh5_i*	0	RS	GPIOh5 input	1'h0
0xD8BB	*reg_top_gpioh5_o*	0	RW	GPIOh5 output	1'h0
0xD8BC	*reg_top_gpioh5_en*	0	RW	GPIOh5 output enable 1: Output mode 0: Input mode	1'h0
0xD8BD	*reg_top_gpioh5_on*	0	RW	GPIOh5 enable	1'h0
0xD8BE	*reg_top_gpioh4_i*	0	RS	GPIOh4 input	1'h0
0xD8BF	*reg_top_gpioh4_o*	0	RW	GPIOh4 output	1'h0
0xD8C0	*reg_top_gpioh4_en*	0	RW	GPIOh4 output enable 1: Output mode 0: Input mode	1'h0
0xD8C1	*reg_top_gpioh4_on*	0	RW	GPIOh4 enable	1'h0
0xD8C2	*reg_top_gpioh7_i*	0	RS	GPIOh7 input	1'h0
0xD8C3	*reg_top_gpioh7_o*	0	RW	GPIOh7 output	1'h0
0xD8C4	*reg_top_gpioh7_en*	0	RW	GPIOh7 output enable 1: Output mode 0: Input mode	1'h0
0xD8C5	*reg_top_gpioh7_on*	0	RW	GPIOh7 enable	1'h0
0xD8C6	*reg_top_gpioh6_i*	0	RS	GPIOh6 input	1'h0
0xD8C7	*reg_top_gpioh6_o*	0	RW	GPIOh6 output	1'h0
0xD8C8	*reg_top_gpioh6_en*	0	RW	GPIOh6 output enable 1: Output mode 0: Input mode	1'h0

0xD8C9	*reg_top_gpioh6_on*	0	RW	GPIOh6 enable	1'h0
0xD8CE	*reg_top_gpioh8_i*	0	RS	GPIOh6 input	1'h0
0xD8CF	*reg_top_gpioh8_o*	0	RW	GPIOh6 output	1'h0
0xD8D0	*reg_top_gpioh8_en*	0	RW	GPIOh8 output enable 1: Output mode 0: Input mode	1'h0
0xD8D1	*reg_top_gpioh8_on*	0	RW	GPIOh8 enable	1'h0
0xD8EE	*reg_top_lock2_out*	0	RW	GPIOH2 is used as lock indicator	1'h0
0xD8EF	*reg_top_lock2_tpsd*	0	RW	GPIOH2 lock indication 1: TPS lock 0: MPEG lock	1'h0
0xD8F3	*reg_top_lock1_out*	0	RW	GPIOH1 is used as lock indicator	1'h0
0xD8F4	*reg_top_lock1_tpsd*	0	RW	GPIOH1 lock indication 1: TPS lock 0: MPEG lock	1'h0
0xD8F8	*reg_top_lock4_out*	0	RW	GPIOH4 is used as lock indicator	1'h0
0xD8F9	*reg_top_lock4_tpsd*	0	RW	GPIOH4 lock indication 1: TPS lock 0: MPEG lock	1'h0
0xD8FD	*reg_top_lock3_out*	0	RW	GPIOH3 is used as lock indicator	1'h0
0xD8FE	*reg_top_lock3_tpsd*	0	RW	GPIOH3 lock indication 1: TPS lock 0: MPEG lock	1'h0
0xD902	*reg_top_pwm0_en*	0	RW	PWM0 enable	1'h0
0xD903	*reg_top_pwm1_en*	0	RW	PWM1 enable	1'h0
0xD904	*reg_top_pwm2_en*	0	RW	PWM2 enable	1'h0
0xD905	*reg_top_pwm3_en*	0	RW	PWM3 enable	1'h0
0xD907	*reg_top_pwm0_pos[2:0]*	2:0	RW	PWM positions	3'h0
0xD908	*reg_top_pwm0_width[1:0]*	1:0	RW	PWM pulse width	2'h0
0xD909	*reg_top_pwm0_duration[7:0]*	7:0	RW	PWM duration (0 to 255 maps to 0% to 99.6%)	8'h0
0xD90B	*reg_top_pwm1_pos[2:0]*	2:0	RW	PWM positions	3'h0
0xD90C	*reg_top_pwm1_width[1:0]*	1:0	RW	PWM pulse width	2'h0
0xD90D	*reg_top_pwm1_duration[7:0]*	7:0	RW	PWM duration (0 to 255 maps to 0% to 99.6%)	8'h0
0xD90F	*reg_top_pwm2_pos[2:0]*	2:0	RW	PWM positions	3'h0

0xD910	*reg_top_pwm2_width[1:0]*	1:0	RW	PWM pulse width	2'h0
0xD911	*reg_top_pwm2_duration[7:0]*	7:0	RW	PWM duration (0 to 255 maps to 0% to 99.6%)	8'h0
0xD913	*reg_top_pwm3_pos[2:0]*	2:0	RW	PWM positions	3'h0
0xD914	*reg_top_pwm3_width[1:0]*	1:0	RW	PWM pulse width	2'h0
0xD915	*reg_top_pwm3_duration[7:0]*	7:0	RW	PWM duration (0 to 255 maps to 0% to 99.6%)	8'h0
0xD91B	*reg_top_hostb_mpeg_par_mode*	0	RW	Host B TS mode register, 1: mpeg parallel mode	1'h0
0xD91C	*reg_top_hostb_mpeg_ser_mode*	0	RW	Host B TS mode register, 1: mpeg serial mode	1'h0
0xD91D	*reg_top_hostb_mpeg_ser_do7*	0	RW	0: Host_B mpeg serial mode data out by pin data0 1: Host_B mpeg serial mode data out by pin data7	1'h0
0xD91E	*reg_top_hostb_dca_upper*	0	RW	Host B TS mode register, 1: connect to DCA upper chip	1'h0
0xD91F	*reg_top_hostb_dca_lower*	0	RW	Host B TS mode register, 1: connect to DCA lower chip	1'h0
0xD920	*reg_top_host_reverse*	0	RW	Reverse all Host_B pins in order	1'h0
0xF103	*reg_one_cycle_counter_tuner[7:0]*	7:0	RWS	2-Wire interface speed control in units of 400 ns Period of SCL = (reg_one_cycle_counter_tuner * 400) ns	8'h10

Appendix C: Generic I2C support in Boot-code

Generic I2C Read command:

This command cause the FW to initiate one I2C read cycle to the specified I2C interface, with specified I2C slave address, and read back specified number of bytes. The read back data can be accessed from the reply packet. The host should wait for some time to allow IT9130 to complete the I2C read cycle. Otherwise the attempt to get the reply packet will be negatively acknowledged (although you may retry to receive the reply packet.)

Generic I2C Read Command packet

Field	Size	Comments
Command Length	1 byte	Number of the following bytes, always 4.
Command	1 byte	0x06, Generic I2C Read command
Interface #	1 byte	1, 2, or 3. IT9130 series normally has three I2C interfaces. Normally, #1 is for host, #2 is for tuner, and #3 is for DCA/PIP to the second chip. However, in USB mode, #1 can be for DCA/PIP, and some USB package do not have #3.
I2C Address + Read bit	1 byte	Bit [7~1]: I2C address Bit [0]: Read bit, must be 1, according to I2C specification
Number of bytes to read	1 byte	# of bytes to read: max 61 for USB interface, or 253 for all other interfaces

Generic I2C Read Reply packet:

Field	Size	Comments
Reply Length	1 byte	Number of the following bytes
Status	1 byte	0 for success, others for error codes.
Read back data bytes	n bytes	Bytes read back. 'n' will be the same as specified in the command packet.

Generic I2C Write command:

This command cause the FW to initiate one I2C write cycle to the specified I2C interface, with specified I2C slave address, and write the specified bytes. This command has no reply packet. The host should wait for some time to allow IT9130 to complete the I2C write cycle. Otherwise, new commands will be negatively acknowledged (although you may retry the command until acknowledged.)

Generic I2C Write Command packet:

Field	Size	Comments
Command Length	1 byte	Number of the following bytes, must be n + 3, where n is the number of bytes to write
Command	1 byte	0x07, Generic I2C Write command
Interface #	1 byte	1, 2, or 3. IT9130 series normally has three I2C interfaces. Normally, #1 is for host, #2 is for tuner, and #3 is for DCA/PIP to the second chip. However, in USB mode, #1 can be for DCA/PIP, and some USB package do not have #3.
I2C Address + Write bit	1 byte	Bit [7~1]: I2C address Bit [0]: Write bit, must be 0, according to I2C specification
Write data bytes	n byte	Max number of bytes: 59 for USB mode, and 251 for all other interfaces.