IT9500 Series —
Programming Guide

Revision History

Revision	Date	Description
1.0	05/21/2012	Initial Release
1.1	07/02/2012	Add register reg_tsip_br description
1.2	09/26/2012	Add IT9507_accessFwPSITable() & IT9507_setFwPSITableTimer() API
1.3	12/05/2012	Add IT9507_setSystemConfig() to define the IT9500 GPIO setting
1.4	12/21/2012	Modify API architecture
1.5	01/16/2013	Modify & rename StreamType struct to TsInterface Modify EEPROM format
1.6	05/09/2013	Modify Register Table
1.7	6/19/2013	1. IQ table format 2. IT9507_setDCCalibrationValue() 3. IT9507_isTsBufferOverflow() 4. cell id in big-endian

Contents

Figures

Tables

1. Overview

This document describes how to control the IT9500 (also known as Eagle) for DVB-T transmission. This document covers all information users need to know regarding IC control.

A complete set of application programming interface (API) is available, facilitating fast and easy integration into products on Windows Mobile, Windows CE, Windows 7/Vista/XP, Linux, and many other operating systems.

Furthermore, complete USB drivers and test kits for Windows 7/Vista/XP and Linux are provided for IT9500.

IT9500 series family includes IT9503 and IT9507. IT9507 is a full featured DVB-T transmitter, while IT9503 supports QPSK constellation only.

2. Operation Modes

IT9500 can operate in TS/IIC, TS/USB, or USB modes. The operation mode is chosen by appropriately setting the strapping pins as specified in Appendix A and register programming.

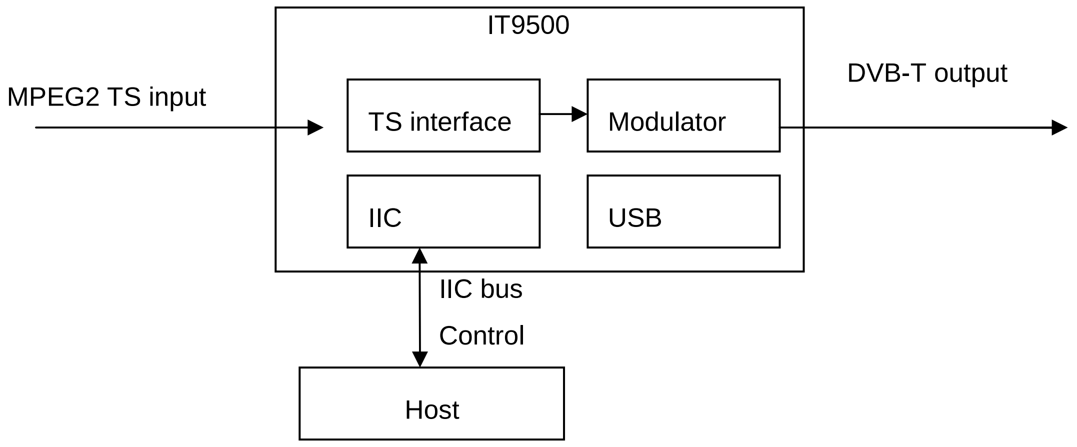
The control and data paths in each operation mode of IT9500 are summarized in Table 1.

Table 1 Data and control paths of IT9500 in each operation mode

Mode	Data Path	Control Path
USB	USB	USB
TS/IIC	MPEG TS	IIC bus
TS/USB	MPEG TS	USB

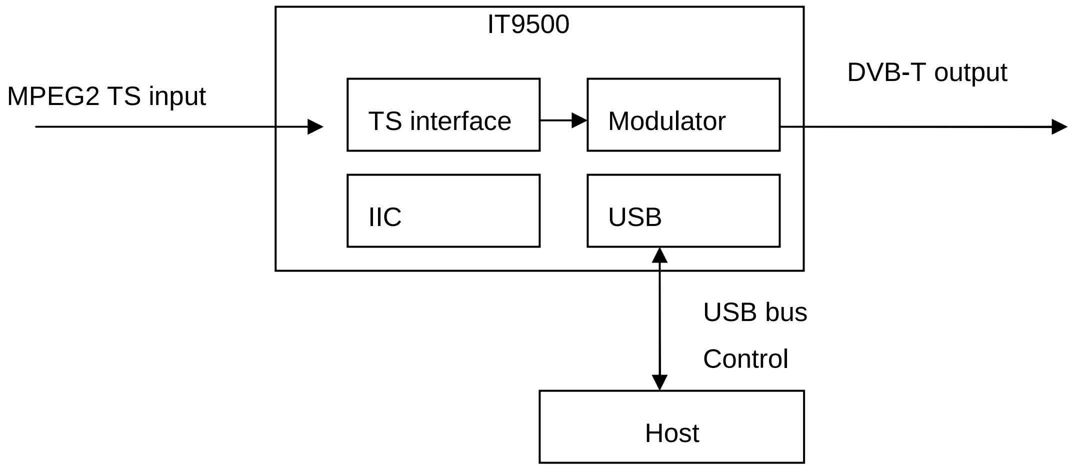
2.1. TS/IIC Mode

In the TS/IIC mode, the transport stream to be transmitted comes through either the parallel or serial TS input interface. IT9500 is controlled by an external host controller (CPU) via the IIC bus as an IIC slave device



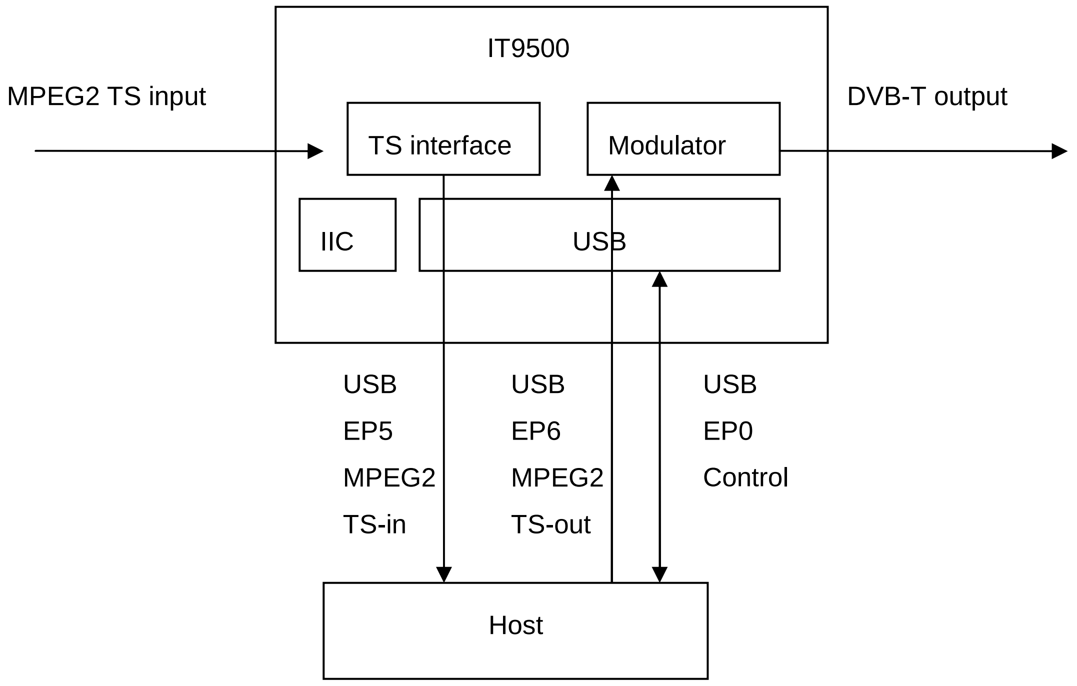
2.2. TS/USB Mode

In the TS/USB mode, the transport stream to be transmitted comes through either the parallel or serial TS input interface. IT9500 is controlled by an external host controller (CPU) via the USB bus as a USB device.



2.3. USB Mode

In the USB mode, IT9500 communicates with the host through the embedded USB 2.0 interface. The transport stream to be transmitted by IT9500 and control/status signals are all encapsulated in the USB frames. Besides, the host can access (read) the TS input thru USB EP5. More details of the USB interface are given in Section 6.2.



2.4. Mode Strapping and IIC bus address

Refer to Appendix A, pin strapping.

3. Controlling IT9500 through API's

IT9500 ANSI-C source files implement the standard modulator 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.

IT9500 software hierarchy is shown below. This chapter describes IT9500 API and user functions. The command packet formats will be described in next chapter.

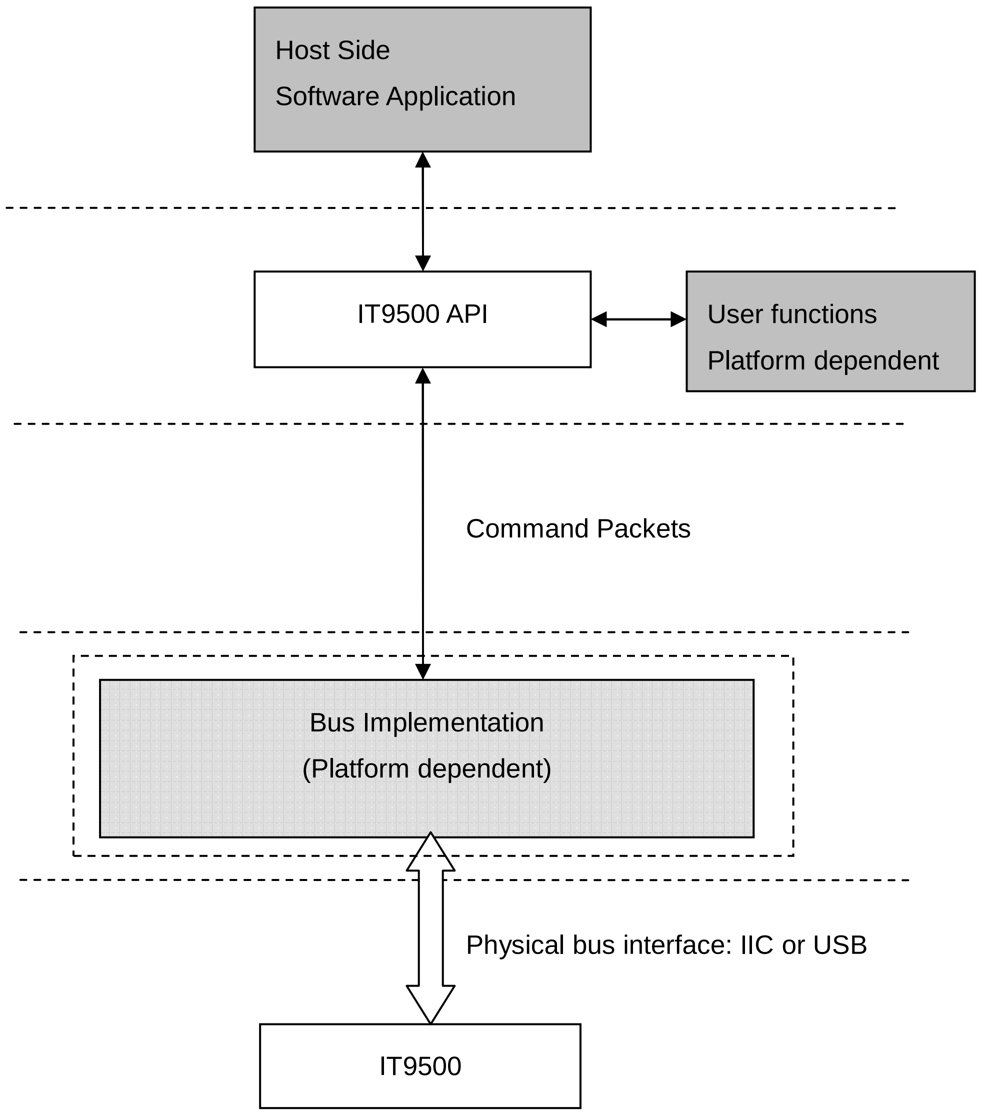


Figure 1 IT9500 software hierarchy

3.1. Introduction

The ANSI-C API source files are listed in Table 2. The user should modify ONLY `modulatorUser.h` and `modulatorUser.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 modulator. Accessing registers outside this API may produce unpredictable results.

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

3.2. Source Files

The ANSI-C API source files are listed in Table 2. The user should only modify `modulatorUser.h` and `modulatorUser.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 IT9500 API source files.

File	User Modifiable	Description
modulatorError.h	NO	Status/error code definition.
modulatorFirmware.h	NO	Firmware file.
IT9507.h	NO	Modulator API functions header file.
IT9507.cpp	NO	Modulator API functions implementation file.
modulatorRegister.h	NO	Registers definition.
eagleTuner.h	NO	Tuner API functions header file.
eagleTuner.cpp	NO	Tuner API functions implementation file.
modulatorType.h	NO	Data type and constants definition.
modulatorVariable.h	NO	Variables definition.
modulatorVersion.h	NO	User could check this file to know the version of API.
modulatorUser.h	**YES**	**API-related settings. Please refer to Chapter 3.3 for details.**
modulatorUser.cpp	**YES**	**System-dependant function. Please refer to Chapter 3.4 for details.**
IQ_fixed_table.h	**YES**	**Customized calibration table**

3.3. modulatorUser.h

`modulatorUser.h` contains API-related settings. Users are recommended to fully understand this file before integrating IT9500 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 `modulatorUser`.h file.

Setting	Description
EagleUser_RETRY_MAX_LIMIT	Define bus retry times when bus error
EagleUser_IIC_SPEED	Define IIC master speed, the default value 0x07 means 366KHz (1000000000 / (24.4 * 16 * EagleUser_I2C_SPEED))

3.4. modulatorUser.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 may have a different way to implement delay function and bus read/write functions. This file provides the function body templates 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 modulatorUser.cpp.

Function	Description
EagleUser_delay	API utilizes this function to implement sleep function.
EagleUser _enterCriticalSection	API utilizes this function to lock synchronize object. If your system supporting multi-thread, you should implement this function to avoid racing condition.
EagleUser _leaveCriticalSection	API utilizes this function to release synchronize object. If your system supporting multi-thread, you should implement this function to avoid racing condition.
EagleUser _mpegConfig	API utilizes this function to configure MPEG-2 TS interface, like clock polarity, clock speed…etc.. This function is called by IT9507_ setTsInterface ()
EagleUser _busTx	API utilizes this function to send control message to desired bus.
EagleUser _busRx	API utilizes this function to receive control message from desired bus.
EagleUser_setBus	This function will be called by upper layer applications in the very beginning to initilialize the control bus, refer to the following section.
EagleUser_Initialization	API utilizes this function to define GPIO settings or customer initialize function, refer to the following section. This function is called by IT9507_initialize ()
EagleUser_Finalize	API utilizes this function to define customer finalize function or release menory and Handle. This function is called by IT9507_ finalize ()
EagleUser_acquireChannel	API utilizes this function to define customized behaviors while acquiring channel. Ex: U/V filter switching or external Lo control. This function is called by IT9507_ acquireTxChannel ()

EagleUser_setTxModeEnable	API utilizes this function to define customized behaviors while turning TX RF on/off. Ex: RF power on/off or RF gain control. This function is called by IT9507_ setTxModeEnable ()
EagleUser_getChannelIndex	API utilizes this function to get the default channel number setting. For example, the default channel setting could be configured by a dip switch which is connected to GPIO's of IT9507. This function is called by IT9507_getDefaultChannelSetting ()

3.5. Control bus implementations in API test kit

IT9500 can be controlled thru either USB or IIC bus. In API test kit, three sample control buses are implemented for reference.

3.5.1. PC printer port for IIC control bus implementation

Refer to the source codes i2Cimpl.cpp and I2Cimpl.h.

3.5.2. USB control bus implementation

Refer to the source codes usb2impl.cpp and usb2impl.h.

3.5.3. AF9035 USB-to-IIC adapter for IIC control bus implementation

Refer to the source codes af9035u2iimpl.cpp and af9035u2iimpl.h.

3.6. Function Prototype Description

Most API functions have a prototype as follows:

```
Dword IT9507_XXX (
     IN  Modulator * modulator
     …
);
```

In the prototype, `modulator` 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:

```
Modulator eagle;
```

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

```
Dword error = ModulatorError_NO_ERROR;


error = IT9507_XXX ((Modulator*) &eagle, …);
```

For more details about each API function, please refer to the comments of IT9500.h.

3.7. Initialization

To initialize IT9500, the user has to call `IT9507_initialize()`, which performs all necessary initialization operations including firmware downloading, firmware-related, and tuner-related initializations. If there is an EEPROM attached to IT9500, it will also load the settings in EEPROM as boot defaults.

The parameter "`streamType`" defines IT9500 TS input interface type. The parameter and these meanings are summarized in Table 5.

Table 5 Parameters for initialization.

Type	Name	Description
TsInterface	streamType	The parameter is used to indicate desired input stream format. ● PARALLEL_TS_INPUT: the Transport Stream (TS) to be transmitted comes from MPEG-2 TS parallel interface. ● SERIAL_TS_INPUT: the Transport Stream (TS) to be transmitted comes from MPEG-2 TS serial interface. ● INTERFACE_UNUSE: No data in via TS interface.

`IT9507_initialize() will call` EagleUser_setBus() to initialize the control bus.

The definition of parameter "Bus_id" is used to indetify the control bus., refer to the following table.

Table 6 *busId definition*

Type	Bus_id	Constants	IT9500 Control Bus Interface
Byte	1	Bus_I2C	IIC
	2	Bus_USB	USB

The third parameter specifies the IIC address of IT9500 when the control bus is IIC. The IIC address is determined by pin strapping, refer to Appendix A. If the control bus is USB, this parameter can be ignored.

An example is given below:

```
Modulator eagle;
Byte Bus_id = Bus_USB;   // The control bus is USB.
TsInterface tsin_streamType = PARALLEL_TS_INPUT;     // TS input type : MPEG-2 parallel interface.

error = IT9507_initialize (&eagle, tsin_streamType, Bus_id, 0);
if (error) goto exit;
```

3.8. Getting Firmware and API Versions

The user can use IT9507_getFirmwareVersion () to get the firmware versions respectively. Note that there are two processors running inside IT9500, so users have to specify the correct processor when calling IT9507_getFirmwareVersion ().

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

An example is given below.

```
Modulator eagle;
Dword linkFirmwareVersion;      // Used to store LINK firmware version.
Dword ofdmFirmwareVersion;      // Used to store OFDM firmware version.
Dword error = ModulatorError_NO_ERROR;

// Get LINK firmware version
error = IT9507_getFirmwareVersion(&eagle, Processor_LINK, &linkFirmwareVersion);

// Get OFDM firmware version
error = IT9507_getFirmwareVersion((&eagle, Processor_OFDM, &ofdmFirmwareVersion);
```

3.9. Set Channel Modulation

The device transmission data rate depends on channel modulation setting. The user can use IT9507_setTXChannelModulation () to setting channel modulation, each setting of transmission data rate shown in Table 7.

Table 7 Transmission data rate in BPS (bits per second)

bandwidth	GI	CR	QPSK	16-QAM	64-QAM
5M	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
6M	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
7M	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
8M	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

An example is given below.

```
/**
 * The type defination of Constellation.
 */
typedef enum {
    Constellation_QPSK = 0,     /** Signal uses QPSK constellation  */
    Constellation_16QAM,        /** Signal uses 16QAM constellation */
    Constellation_64QAM         /** Signal uses 64QAM constellation */
} Constellation;
/**
 * The type defination of CodeRate.
 */
typedef enum {
    CodeRate_1_OVER_2 = 0,      /** Signal uses FEC coding ratio of 1/2 */
    CodeRate_2_OVER_3,          /** Signal uses FEC coding ratio of 2/3 */
    CodeRate_3_OVER_4,          /** Signal uses FEC coding ratio of 3/4 */
    CodeRate_5_OVER_6,          /** Signal uses FEC coding ratio of 5/6 */
    CodeRate_7_OVER_8,          /** Signal uses FEC coding ratio of 7/8 */
    CodeRate_NONE               /** None, NXT doesn't have this one     */
} CodeRate;
/**
 * The type defination of TransmissionMode.
 */
typedef enum {
    TransmissionMode_2K = 0,    /** OFDM frame consists of 2048 different carriers (2K FFT mode) */
    TransmissionMode_8K = 1,    /** OFDM frame consists of 8192 different carriers (8K FFT mode) */
    TransmissionMode_4K = 2     /** OFDM frame consists of 4096 different carriers (4K FFT mode) */
} TransmissionModes;

/**
 * The type defination of Interval.
 */
typedef enum {
    Interval_1_OVER_32 = 0,     /** Guard interval is 1/32 of symbol length */
    Interval_1_OVER_16,         /** Guard interval is 1/16 of symbol length */
    Interval_1_OVER_8,          /** Guard interval is 1/8 of symbol length  */
    Interval_1_OVER_4           /** Guard interval is 1/4 of symbol length  */
} Interval;
/**
 * The defination of ChannelInformation.
 */
typedef struct {
    Dword frequency;                    /** Channel frequency in KHz.                                */
    TransmissionModes transmissionMode; /** Number of carriers used for OFDM signal                  */
    Constellation constellation;        /** Constellation scheme (FFT mode) in use                   */
    Interval interval;                  /** Fraction of symbol length used as guard (Guard Interval) */
    Priority priority;                  /** The priority of stream                                   */
    CodeRate highCodeRate;              /** FEC coding ratio of high-priority stream                 */
    CodeRate lowCodeRate;               /** FEC coding ratio of low-priority stream                  */
    Hierarchy hierarchy;                /** Hierarchy levels of OFDM signal                          */
    Bandwidth bandwidth;
} ChannelModulation;



Modulator eagle;
Dword error = ModulatorError_NO_ERROR;
ChannelModulation      channelModulation;

channelModulation.constellation = Constellation_16QAM;
channelModulation.highCodeRate = CodeRate_3_OVER_4;
channelModulation.interval = Interval_1_OVER_32;
channelModulation.transmissionMode = TransmissionMode_8K;

error = IT9507_setTXChannelModulation(&eagle, &channelModulation);
if (error) {
     printf ("Error Code = %X", error);
     return;
}
```

Note: IT9503 supports QPSK constellation only.

3.10. Set TPS (Transmission Parameter Signaling) Cellid

The cell id in TPS is configurable. Please call IT9507_setTPS () to set cell id.

Note: The cell id specified in IT9507_setTPS () **is big-endian.**

Note: IT9507_setTPS () can only be used to set cell id, the other parameters in TPS are determined by IT9507_setTXChannelModulation ().

3.11. Acquiring Channel

Acquiring transmit channel is accomplished by calling IT9507_acquireTxChannel (). The desired channel frequency and bandwidth provided to the function are in units of kHz.

The user function EagleUser_acquireChannel() will be called by IT9507_acquireTxChannel () before function exit, so customized features and behaviors can be implemented therein.

An example is given below.

```

Modulator eagle;
Dword frequency = 474000;       // Frequency is 474000 KHz.
```

```
Word bandwidth = 8000;          // Bandwidth is 8000 KHz.
Dword error = ModulatorError_NO_ERROR;

error = IT9507_acquireTxChannel (&eagle, bandwidth, frequency);
if (error) {
     printf ("Error Code = %X", error);
     return;
}
```

Refer to Appendix D for more details about how the internal local oscillator (LO) is set for the RF frequency.

Note: An IT9500-based modulator application can be designed to use an external LO instead of IT9500's internal LO. In sample test kit, the codes in EagleUser_acquireChannel() shows how an external LO (ADF4351) is configured to set RF frequency.

3.12. Enable Transmission Mode

IT9507_ setTxModeEnable () is used for enable/disable data transmission .When enable transmission mode, device starts to transmit data with the specified channel frequency, bandwidth and channel modulation.

The user function EagleUser_setTxModeEnable() will be called by IT9507_ setTxModeEnable()before function exit, so customized features and behaviors can be implemented therein.

An example is given below.

```
Modulator eagle;
error = IT9507_setTxModeEnable(&eagle, 1); // transmit Data
```

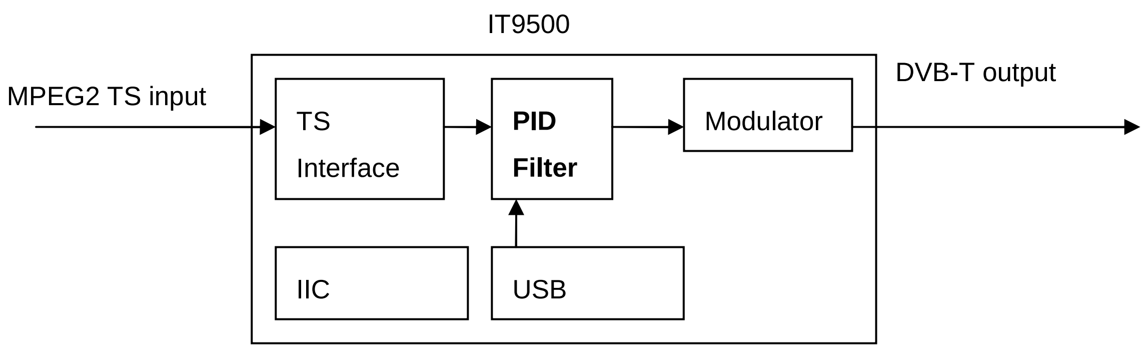
3.13. Adding a PID to the PID Filter

A hardware PID filter is built-in with the TS input interface of IT9500. The stream fed to the modulator (either from TS interface or USB interface) will be pre-processed by the PID filter. 31-way PID filters are supported(index 1~31). The PID filter behavior can be specified as either "Pass" or "Block" mode.

In "Pass" mode, PID filter allows the specified PID packets pass.

In "Block" mode, PID filter will block the specified PID packets.

IT9507_addPidToFilter () is used for adding a PID to the hardware PID filter.



An example is given below.

```
Modulator eagle;
Pid pid;
Byte index;
Dword error = ModulatorError_NO_ERROR;

pid.value = 0x0000;
index = 1;
error = IT9507_addPidToFilter(&eagle, index, pid);
```

3.14. Resetting PID Filter

`IT9507_resetPidFilter` ( ) 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.

```
Modulator eagle;
Dword error = ModulatorError_NO_ERROR;

error = IT9507_resetPidFilter (&eagle);
```

3.15. Control PID Filter

IT9507_controlPidFilter () is used to control hardware PID filter. Besides filter enable/disable, the filter behavior is specified with this API. Pass mode (control=0) will block all PID packets except the PID's specified. Block mode (control=1) will block all the specified PID packets. All entries in the PID filter will be set to 0 after the PID filter is reset.

An example is given below.

```
Modulator eagle;
```

```
Byte control = 1; //0:filter packets except the specied PID's(Pass mode) ; 1:filter packets with the specified PID's (Block mode)

Byte enable = 1; //0:disable 1:enable PID filter

Dword error = ModulatorError_NO_ERROR;


error = IT9507_controlPidFilter(&eagle, control, enable);
```

3.16. Transmitting Custom SI/PSI Packets

IT9500 supports two mechanisms for custom SI/PSI table insertion.

3.16.1. One-shot Table Transmission

The user can use IT9507_sendHwPSITable () to transmit a 188-byte custom SI/PSI packet or any proprietary user packet. **The packet is transmitted once and only once.**

This function is achieved with IT9500 hardware registers. More details can be found in Appendix E.

An example is given below.

```
Modulator eagle;

Byte PSIbuffer[188];

Dword error = ModulatorError_NO_ERROR;


/** edit PSIbuffer*/

for(i=0;i<188;i++){

     if(i==0)

          temp = 0x47;

     else

          temp = i;

          PSIbuffer[i] = temp;

}

error = IT9507_sendHwPSITable(&eagle, PSIbuffer);

if (error)

     printf("%d:IT9507_sendHwPSITable failed! Error = %08x\n",i ,error);

else

     printf("%d:IT9507_sendHwPSITable successful \n");
```

3.16.2. Periodical Custom Table Transmission

The user can set up to 5 custom packets (each at 188 Bytes) and specify the repetition rate (interval) to transmitting the packets periodically. **The five buffers are indexed from 1~5.**

The user can use IT9507_accessFwPSITable ( ) to set the custom packets and use IT9507_setFwPSITableTimer ( ) to set repetition interval in units of ms. Setting the interval timer to zero, will disable the corresponding custom packet.

This function is achieved by IT9500 MCU and firmware.

An example is given below.

```
Modulator eagle;
Dword error = ModulatorError_NO_ERROR;
Byte index = 1;       //Buffer index =1
Byte timer = 50;      // periodically send time = 50ms

error = IT9507_accessFwPSITable(&eagle, index,PSIbuffer);
if(error)
     printf("IT9507_accessFwPSITable failed! Error = %08x\n", error);
error = IT9507_setFwPSITableTimer(&eagle,index ,timer);
if(error)
     printf("IT9507_getDeviceType failed! Error = %08x\n", error);
else
printf("IT9507_getDeviceType successful index = %d Timer = %d(ms)\n",index ,timer);
```

3.17. Adjust Output Gain

The user can use IT9507_AdjustOutputGain ( ) to adjust IT9500 output gain in units of dB.

The gain adjustment is implemented digitally before the on-chip ADC, so there will be MER loss (signal distortion) in higher gain or lower attenuation. Recommend range is between -10db~+3 db.

The real gain value set will be updated when the function exits.

An example is given below.

```
Modulator eagle;
int adjustgain = 1;
Dword error = ModulatorError_NO_ERROR;

error = IT9507_AdjustOutputGain(&eagle, &adjustgain);
if(error)
     printf("IT9507_AdjustOutputGain failed! Error = %08x\n", error);
else
     printf("IT9507_AdjustOutputGain successful setting gain = %d\n", adjustgain);
```

3.18. Register Access

All IT9500 API's functions are implemented by register read/write. The register definitions are abstracted by the API layer. In general, it's not necessary for users to understand or access the registers directly.

In case of need, it should be very careful about direct register access. Once IT9507_initialize () has been called, improper register access may crash IT9500.

The functions `IT9507_readRegister()` and `IT9507_writeRegister()` are, respectively, used for reading and writing one register of the IT9500 series IC. The functions `IT9507_readRegisters()` and `IT9507_writeRegisters()` are used for burst access, i.e., reading and writing multiple registers.

There are two read-only registers which can be read to verify the bus connectivity. The addresses and their default values are listed in Table 8.

Table 8 Reference registers and their default values

Address	Processor	Default Value
0x1223	Processor_LINK	0x07
0x1224	Processor_LINK	0x95

An example is given below for register reading. Note that there are two processors inside IT9500, so the user must specify the target processor for which register reading and writing is to be performed.

```
Modulator eagle;
Processor procssor = Processor_LINK;       // Read a register from LINK processor.
Dword reigster = 0xF000;                   // Read a register at 0xF000.
Byte value;                                // Register value.
Dword error = ModulatorError_NO_ERROR;

error = IT9507_readRegister (&eagle, processor, register, &value);
```

3.19. Finalize

Please remember to call `IT9507_finalize ()` before shutting down the system. Once this function has been called, it will release allocated resources. Besides, EagleUser_Finalize () will be called for customized features.

An example is given below.

```
Modulator eagle;
Dword error = ModulatorError_NO_ERROR;

error = IT9507_finalize (&eagle);
```

3.20. Performance optimization with IQ calibration

Because of different characteristics with different designs, it’s necessary to compensate IT9500 IQ output for optimized signal quality.

In API, the IQ calibration table is kept in the array IQtable IQ_fixed_table0[] defined in IQ_fixed_table.h.

The calibration data array contains a sequence of discrete channel frequencies in ascending order and associated calibration dAmp and dPhi values. Statically, users may update the calibration data by modifying IQ_fixed_table.h.

While setting (acquiring) a channel, the API will search the array to get the corresponding calibration dAmp and dPhi values, then set them to IT9500. For channel frequencies not listed in the array, interpolation will be used to derive the calibration values.

The calibration table can also be stored in an IQ calibration bin file released by ITE for dynamical update. Users may call IT9507_setIQtable() to inform the underlying API that a new IQ calibration table should be referenced instead of the built-in IQ_fixed_table0[].

Refer to the following table for the bin file format. A file dump sample is also shown in the following figure.

Table 9 File format of IQ calibration bin file

Offset	Size (bytes)	Descriptions
0~0x9	10	Mnemonic descriptions for this file
0xa~0xd	4	Version code
0xe~0x0f	2	The number of calibration item entries (in Big-endian)
0x10~0x13	4	#1 Frequency in KHz
0x14~0x15	2	dAmp
0x16~0x17	2	dPhi
0x18~0x1b	4	#2 Frequency in KHz
0x1c~0x1d	2	dAmp
0x1e~0x1f	2	dPhi
…		

Note: all entry values are in little-endian except the “number of calibration item entries”.

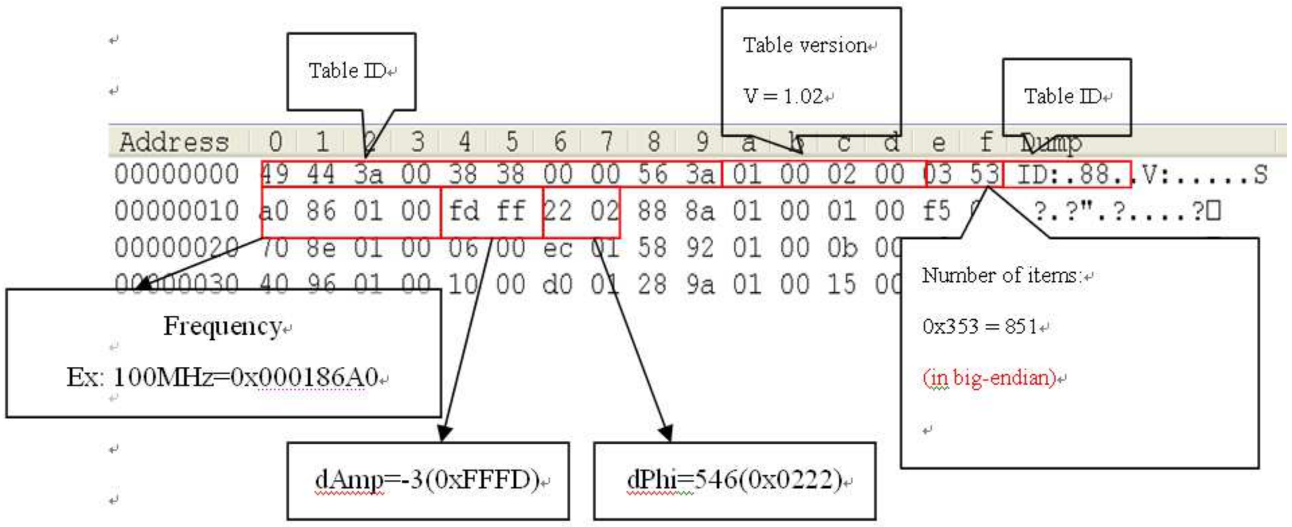

```
Address   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000000 49 44 3a 00 38 38 00 00 56 3a 01 00 02 00 03 53  ID:.88..V:.....S
00000010 a0 86 01 00 fd ff 22 02 88 8a 01 00 01 00 f5     ?.?".?....?
00000020 70 8e 01 00 06 00 ec 01 58 92 01 00 0b 00
00000030 40 96 01 00 10 00 d0 01 28 9a 01 00 15 00
```

Figure 2 IQ calibration bin file format.

Custom IQ table is loaded with the API IT9507_setIQtable (). A piece of sample codes shows how an IQ calibration bin file is loaded to replace the default IQ table.

```
static IQtable IQ_tableEx [65536];


Dword Example_LoadIQCalibrationTable () {
    Dword error = ModulatorError_NO_ERROR;
     FILE *pfile = NULL;
     //FILE *pfile2 = NULL;
     char buf[16]={0};
     char inputFile[255]={0};
     int dAmp = 0;
     int dPhi = 0;
     int row;
     Word groups;
     //IQtable IQ_tableEx [91];
     Dword freq = 0;


     printf("input File name ?\n");
     scanf ("%s", inputFile);
     pfile = fopen(inputFile,"rb");
     if(pfile != NULL){
           fread(buf,16,1,pfile);
           groups = buf[14]<<8 | buf[15];  //frequency groups ex:(950000-50000)/10000 + 1 = 91
           for(row = 0; row < groups; row++){
            fread(&freq,4,1,pfile);
            fread(&dAmp,1,2,pfile);
            fread(&dPhi,1,2,pfile);

            IQ_tableEx[row].frequency = freq;
            IQ_tableEx[row].dAmp = (short)dAmp;
            IQ_tableEx[row].dPhi = (short)dPhi;

        }
        error = IT9507_setIQtable(&eagle, &IQ_tableEx[0],groups);
        if(error)
            printf("ModulatorError_NULL_PTR \n");
        else
            printf("Load IQ Calibration Table successful\n");
        fclose(pfile);
        pfile = NULL;
    }else{
        error = ModulatorError_OPEN_FILE_FAIL;
    }

    return error;
```

Besides, there is a new API IT9507_setDCCalibrationValue (), which can be called to compensate I/Q DC offset. The offset values are not fixed and varies from board to board. A calibration process is required to find out the correct value.

3.21. Accessing General Purpose Input/Output (GPIO)

IT9500 supports 8 general-purpose input/output (GPIO) pins. The GPIO pins can be programmed as either input (GPI) or output (GPO).

All GPIO pins are accessed and controlled through the link processor registers.

Take GPIOH1 as an example.

First of all, the register reg_top_gpioh1_on (0xD8B1) should set to 1 to enable GPIO1.

Second, if set the register reg_top_gpioh1_en (0xD8B0) to 0, GPIO1 is an input pin.

if set the register reg_top_gpioh1_en (0xD8B0) to 1, GPIO1 is an output pin.

If GPIO1 is an input pin, the pin state is controlled by the register reg_top_gpioh1_i[0] (0xD8AE).

If GPIO1 is an output pin, the pin state is controlled by the register reg_top_gpioh1_o[0] (0xD8AF).

The register addresses are defined in modulatorRegister.h.

An example is given below.

```
Modulator eagle;

// Set GPIOH1 to output pin
error = IT9507_writeRegister (&eagle, Processor_LINK, p_eagle_reg_top_gpioh1_en, 1);
if (error) goto exit;
error = IT9507_writeRegister (&eagle, Processor_LINK, p_eagle_reg_top_gpioh1_on, 1);
if (error) goto exit;
// Set GPIOH1 to Hi
error = IT9507_writeRegister (&eagle, Processor_LINK, p_eagle_reg_top_gpioh1_o, 1);
if (error) goto exit;

// Set GPIOH4 to input pin
error = IT9507_writeRegister (&eagle, Processor_LINK, p_eagle_reg_top_gpioh4_en, 0);
if (error) goto exit;
error = IT9507_writeRegister (&eagle, Processor_LINK, p_eagle_reg_top_gpioh4_on, 1);
if (error) goto exit;
// read GPIOH4 input
error = IT9507_readRegister (&eagle, Processor_LINK, r_eagle_reg_top_gpioh4_i, &temp);
if (error) goto exit;
```

3.22. IIC Bus Master Read/Write

IT9500 supports an IIC bus master which can be used to control other IIC slave devices, for example, a DVBT demodulator or EEPROM. IT9507_readGenericRegisters () and IT9507_writeGenericRegisters () are used to read/write slave devices which are connected to IT9500 IIC bus master. The IT9500 master IIC bus speed is defined by EagleUser_I2C_SPEED in modulatorUser.h. The default value 0x07 means 366KHz (1000000000 / (24.4 * 16 * EagleUser_I2C_SPEED))

An example is given below.

```
Modulator eagle;
Dword error = ModulatorError_NO_ERROR;
//write slave device, IIC address is 0x3a
error = IT9507_writeGenericRegisters(&eagle,0x3a,bufferLength,buffer)

//read slave device, IIC address is 0x3a
error = IT9507_readGenericRegisters(&eagle,0x3a,bufferLength,buffer)
```

3.23. EEPROM Read/Write

For USB application, an optional EEPROM can be connected to IT9500 for default configurations.
Users can call IT9507_readEepromValues () and IT9507_writeEepromValues () to read/write a contiguous block of byes in the EEPROM if any. The maximum burst size is restricted by the bus capability. If the bus can transfer N bytes in a transaction cycle, then the maximum bufferLength is N – 5. (firmware will detect EEPROM address).
More details about EEPROM can be found in later chapter.

An example is given below.

```
Dword error = Error_NO_ERROR;
Byte buffer[3] = { 0x00, 0x01, 0x02 };
Modulator eagle;
// Set the value of cell 0x0000 in EEPROM to 0.
// Set the value of cell 0x0001 in EEPROM to 1.
// Set the value of cell 0x0002 in EEPROM to 2.
error = IT9507_writeEepromValues ((Modulator*) &eagle, 0x0000, 3, buffer);
if (error)
printf ("Error Code = %X", error);
else
printf ("Success");

// Get the value of cell 0x0000, 0x0001, 0x0002 in EEPROM.
error = IT9507_readEepromValues ((Modulator*) &eagle, 0, 0x0000, 3, buffer);
if (error)
printf ("Error Code = %X", error);
else
      printf ("Success");
printf ("The value of 0x0000 is %2x", buffer[0]);
printf ("The value of 0x0001 is %2x", buffer[1]);
printf ("The value of 0x0002 is %2x", buffer[2]);
```

3.24. Miscellaneous functions

3.24.1. IT9507_isTsBufferOverflow

IT9507_isTsBufferOverflow (IN   Modulator*modulator, IN   Bool   *overflow);

This API is called to check if the transmitter channel capacity can afford the TS stream input from the TS interface. If the channel capacity is smaller than the TS input stream data rate, there will be buffer overflow.

Then, it's recommended to increase the channel capacity by setting new constellation (QAM size), code rate, and guard interval properly. Refer to chapter 3.9.

4. Command Packet Formats

API communicates with IT9500 by command packets over the bus (either IIC or USB). This chapter describes the formats of the command packets.

4.1. General (Common) Command Format

Field	Size	Comments
Command Length	1 byte	= 2 + 1 + N + 2, i.e. The whole command length excluding this command length byte.
Command	2 bytes	Individual commands will be explained separately later.
Sequence #	1 byte	Increment by 1 (wrapped around after 255) for temporal identification of a command
Command Payload	N bytes	This depends on command, see each command below. For some command that does not have the payload, this will be zero.
Checksum	2 bytes	1's complement of the unsigned 16-bit integer add-sum from Command" field to the last byte of the "Command Payload", ignoring the carry. Big endian is assumed for interpreting 16-bit integers. If the number of bytes to be summed is odd, a virtual byte of zero value is padded as the last byte to make it even.

Notes:

Bit 15 of command:

0 means link-layer command,

1 means OFDM command.

Big-endian is assumed for all fields larger than 1 byte, i.e. Most significant byte will be in the first place.

Commands are supported by the firmware only by default, unless specified otherwise. Some commands are supported by boot-code only, yet some commands are supported by both.

4.2. General (Common) Command Reply Format:

Each command that needs a reply must have its reply following this format, except for "Data Read" command, which has its own reply format (explained later).

This is a general command reply format:

Field	Size	Comments
Reply Length	1 byte	= 1 + 1 + N + 2, i.e. The whole reply length excluding this reply length byte.
Sequence #	1 byte	Must be the same as the sequence # of the corresponding command.
Status	1 byte	0 for OK. Others for error code. See each command reply for detail.

Reply Payload	N bytes	This depends on the command reply, see each command reply below. For some command reply that does not have the payload, this will be zero.
Checksum	2 bytes	1's complement of the unsigned 16-bit integer add-sum from sequence # field" to the last byte of the "Reply Payload", ignoring the carry. Big endian is assumed for interpreting 16-bit integers. If the number of bytes to be summed is odd, a virtual byte of zero value is padded as the last byte to make it even.

Unknown commands or commands with checksum errors will be quietly ignored with no reply. The host may need to implement timeout for those situations.

4.3. Write Register Command: Command 0x0001

#define COMMAND_REG_DEMOD_WRITE 0x0001

Field	SubField	Size	Comments
Command Length	--	1 byte	Defined in "General Command Format".
Command	--	2 bytes	0x0001 for modulator register write.
Sequence #	--	1 byte	Defined in "General Command Format"
Command Payload (N = 1+1+4+n)	Length	1 byte	= n, the number of bytes to write.
	Address Length	1 byte	The length of address 2 for 8051 processor
	Starting Address	4 bytes	Starting address to write
	Data	n bytes	Data to be written
Checksum	--	2 bytes	Defined in "General Command Format"

Write Register Command has the reply in the form described in General (Common) Command Reply Format, with no reply payload. Error code is listed right below:

#define ERR_REG_WRITE_FAIL 0x2

4.4. Read Register Command: Command 0x0000

#define COMMAND_REG_DEMOD_READ 0x0000

Field	SubField	Size	Comments
Command Length	--	1 byte	Defined in "General Command Format".

Command	--	2 bytes	0x0000 for demodulator register read.
Sequence #	--	1 byte	Defined in “General Command Format”
Command Payload 6 bytes	Length	1 byte	The number of bytes to read.
	Address Length	1 byte	The length of address 2 for 8051 processor
	Starting Address	4 bytes	Starting address to read
Checksum	--	2 bytes	Defined in “General Command Format”

Read Register Command has the reply in the form described in General (Common) Command Reply Format, with the reply payload containing only an array of read data, with the same length as the “Length” subfield of the requesting command. Error code is listed right below:

```
#define ERR_REG_READ_FAIL0x3
```

4.5. Query Info Command: Command 0x0022

```
#define COMMAND_QUERYINFO 0x0022
```

Field	SubField	Size	Comments
Command Length	--	1 byte	Defined in “General Command Format”.
Command	--	2 bytes	0x0022
Sequence #	--	1 byte	Defined in “General Command Format”
Command Payload	Query Type	1 byte	0 for firmware version query. (Currently only one query type)
Checksum	--	2 bytes	Defined in “General Command Format”

Reply :

Field	Size	Comments
Reply Length	1 byte	= 1 + 1 + 4 + 2 = 8 byte
Sequence #	1 byte	Must be the same as the sequence # of the corresponding command.
Status	1 byte	0 for OK.
Firmware Version	4 bytes	Firmware Version
Checksum	2 bytes	1’s complement of the unsigned 16-bit integer add-sum from sequence #

		field" to the last byte of the "Reply Payload" ignoring the carry. Big endian is assumed for interpreting 16-bit integers. If the number of bytes to be summed is odd, a virtual byte of zero value is padded as the last byte to make it even.

4.6. Boot Command: Command 0x0023

#define COMMAND_BOOT 0x0023

Field	SubField	Size	Comments
Command Length	--	1 byte	Defined in "General Command Format".
Command	--	2 bytes	0x0023
Sequence #	--	1 byte	Defined in "General Command Format"
Checksum	--	2 bytes	Defined in "General Command Format"

Command reply for this command follows the general command reply format, and is always successful (with zero status byte), except for checksum error. No reply payload is needed.

4.7. SCATTER WRITE Command: Command 0x0029

#define COMMAND_SCATTER_WRITE 0x0029

Field	SubField	Size	Comments
Command Length	--	1 byte	Defined in "General Command Format".
Command	--	2 bytes	0x0029
Sequence #	--	1 byte	Defined in "General Command Format"
Command Payload (N = 1+1+n)	Mode	1 byte	0x03
	CPU	1 byte	0x0 for link layer CPU
	Payload	n byte	Firmware
Checksum	--	2 bytes	Defined in "General Command Format"

Scatter Write Command has the reply in the form described in General (Common) Command Reply Format, with no reply payload.

4.8. Generic IIC Read Command: Command 0x002A

#define COMMAND_GENERIC_READ 0x002A

Field	SubField	Size	Comments
Command Length	--	1 byte	Defined in "General Command Format".
Command	--	2 bytes	0x002A
Sequence #	--	1 byte	Defined in "General Command Format"
Command Payload 3 byte	Length	1 byte	= n, the number of bytes to read.
	IIC Bus	1 byte	IIC bus,IT9500 set to 2.
	IIC Slave Add	1 byte	IIC slave address.
Checksum	--	2 bytes	Defined in "General Command Format"

Generic IIC Read Register Command has the reply in the form described in General (Common) Command Reply Format, with the reply payload containing only an array of read data, with the same length as the "Length" subfield of the requesting command.

4.9. Generic IIC Write Command: Command 0x002B

#define Command_GENERIC_WRITE 0x002B

Field	SubField	Size	Comments
Command Length	--	1 byte	Defined in "General Command Format".
Command	--	2 bytes	0x002B
Sequence #	--	1 byte	Defined in "General Command Format"
Command Payload (N = 1+1+1+n)	Length	1 byte	= n, the number of bytes to write.
	IIC Bus	1 byte	IIC bus,IT9500 set to 2.
	IIC Slave Add	1 byte	IIC slave address.
	Data	n bytes	Data to be written

Checksum	--	2 bytes	Defined in “General Command Format”

Generic IIC Write Command has the reply in the form described in General (Common) Command Reply Format, with no reply payload.

5. Configuring MPEG-2 Transport Stream Interface

5.1. IT9500 Host Interface

IT9500 provides an so-called Host Interface for connecting to other slave device. The Host Interface can be configured into TS input interfaces or disable. Host Interface comprises pins HOST_B0~HOST_B11.

IT9500 Host Interface (HOST_B0~HOST_B11) is used for connecting to the slave device. In general, Host Interface is configured as the MPEG TS input interface for TS receiving.

The IT9500 MPEG-2 transport stream interface pins are listed in Table 10. 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 10 MPEG2 TS interface pin list.

Pin Name	**MPEG TS Input Interface**	
	Function	**Description**
HOST_B0	MPEG Data[7]	# 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
HOST_B1	MPEG Data[6]	
HOST_B2	MPEG Data[5]	
HOST_B3	MPEG Data[4]	
HOST_B4	MPEG Data[3]	
HOST_B5	MPEG Data[2]	
HOST_B6	MPEG Data[1]	
HOST_B7	MPEG Data[0]	
HOST_B8	MPEG Clock	MPEG clock
HOST_B9	MPEG Valid	MPEG data valid
HOST_B10	MPEG Sync	MPEG packet sync pulse
HOST_B11	MPEG Fail	MPEG uncorrectable packet indicator

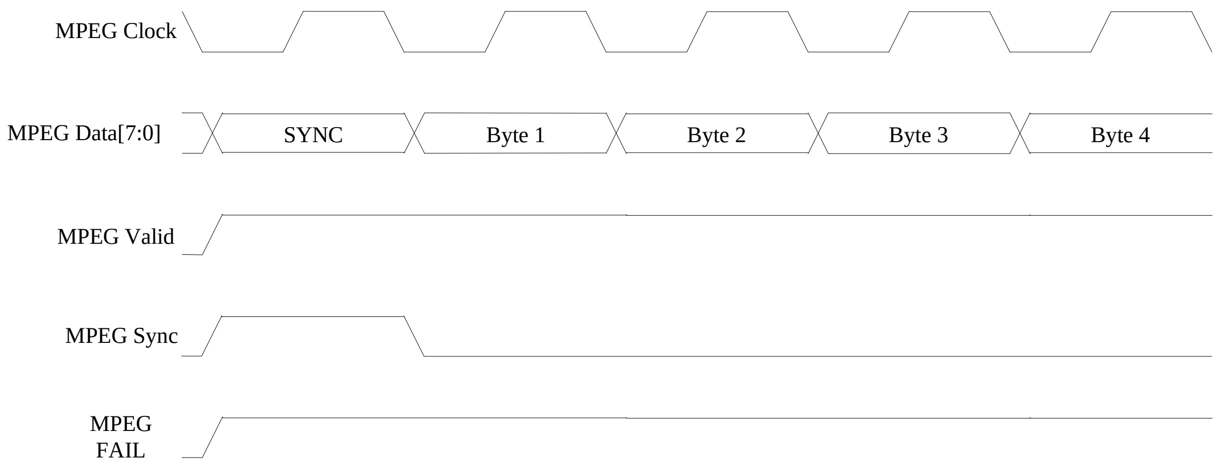


Figure 3 An example of MPEG2 parallel interface timing diagram.

5.2. Configurable Parameters

The MPEG-2 transport stream interface of IT9500 is fully configurable. The complete set of configuration registers is described in detail in Table 11. 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 11 Configuration registers for the MPEG-2 TS interface.

Register Name	Register Address[bits]	Description	Default Value
mpeg_ser_mode	0xF985[0]	Selection about **parallel input mode**, or **serial input mode**. See Table 12 for more details.	0
mpeg_par_mode	0xF986[0]		0
mpeg_ser_do7	0xF984[0]	Pin select for **serial input mode** 0: TS data is carried on MPDATA0 1: TS data is carried on MPDATA7	0
reg_ts_lsb_1st	0xF9B3[0]	Endianness of the **serial input mode** 0: MSB first 1: LSB first.	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 target values in the registers reg_tpsd_bw_mp2if, reg_tpsd_gi_mp2if, reg_tpsd_cr_mp2if, and reg_tpsd_cons_mp2if.	0

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_br	0xF9CE[0]	For the parallel input mode, invert the bit sequence of the input MPEG-2 data. 0: LSB first. 1: MSB first.	0
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

5.3. Interface Modes

IT9500 MPEG-2 transport stream interface is an input interface that accepts MPEG-2 transport streams in either serial or parallel stream mode. 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 IT9500 can be wired to the ground if MPEG Sync is not available from the MPEG TS stream source.

A truth table for selecting the IT9500 MPEG-2 TS interface mode is given in Table 12.

Table 12 The truth table for IT9500 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**
TS/IIC TS/USB USB	1	0	0	0	Parallel Input
	0	1	0	0	Serial Input

5.4. Bit Ordering

The bit-ordering of the parallel output interface for IT9500 and the endianness of the serial input and output of IT9500 are configurable by programming the registers *reg_tsip_br* and *reg_ts_lsb_1st* as described in Table 11.

5.5. Sync Byte Style

In IT9500, the sync byte is supported to be of the MPEG-2 style or DVB-T style.

5.6. 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 11.

5.7. Clock Frequency

There are four possible MPEG Clock frequencies. These frequencies are 5.12MHz, 10.24 MHz, 20.48MHz, and 40.96 MHz for the serial 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. Serial and Parallel Input Interface

By programming the registers *mpeg_ser_mode*, *mpeg_par_mode*, *reg_tsip_en, and reg_tsis_en*, the MPEG2 TS interface of IT9500 becomes an input interface for accepting a 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 IT9500 MPEG2 TS serial input interface is shown in Figure 4.

```
MPEG Clock    cccccccccccc:::cccccccccccccccc
MPEG Data     zmssssssssss:::ssssizzzmsssssss
MPEG Valid    lkhhhhhhhhhh:::hhhhtlllkhhhhhhh
MPEG Sync     lktlllllllll:::lllllllktllllll
```

Figure 4 Timing diagram of the IT9500 MPEG-2 TS serial input interface.

Furthermore, by programming the register *reg_ts_capt_bg_sel* to 0, MPEG Sync of IT9500 can be wired to the ground if MPEG Sync is not available from the MPEG TS stream source.

6. Control Interface

6.1. Using the IIC Interface

IT9500 provides a IIC interface for communicating with the host or the second IC. In general, the IIC interfaces are controlled through registers. Function calls for using the IIC interfaces are provided in the IT9500 API.

TheIT9500 IIC Interface uses, respectively, pins IOSDA for the serial data and IOSCL for the serial clock. The bus address of the IIC Interface is determined by the strapping pins GPIOH6 and GPIOH5. When IT9500 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 IIC bus address, as shown in Table 13. For IT9500 the logic level of the strapping pins GPIOH6 and GPIOH5 also determines its operation mode, as described in Table 21.

Table 13 IT9500 IIC bus address mapping table.

{GPIOH6,GPIOH5} at strapping	IIC Bus Address
00	0x38
01*	0x3A

The IT9500 IIC 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 IIC host interface are shown in Figure 5 and Figure 6, respectively. The legends used therein are listed in Table 14.

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

Figure 5 IIC bus write operation.

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

Figure 6 IIC bus read operation

Table 14 Legends used in Figure 5 and Figure 6.

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 5, in the write operation mode (IT9500 acts as a slave receiver), each data byte definition need to reference to protocol control section. From hardware point of view, IIC module hardware does not parse any data bytes for information. It just moves data from the IIC bus to the internal mailbox. Similarly, in the read operation mode (IT9500 acts as a slave transmitter) the hardware moves the data in the internal mailbox to the IIC bus.

The transmission time per bit of the IT9500 IIC interfaces can be adjusted by programming the register p_eagle_reg_lnk2ofdm_data_63_56 (0x F6A7). Smaller value of the register represents *faster* data rate. The transmission time per bit is in units of 400ns, thus when p_eagle_reg_lnk2ofdm_data_63_56 is programmed to 0x14 the resulting data rate is roughly 125 Kbit/second.

6.2. USB Interface

IT9500 supports USB 2.0 standard with many configurable parameters in the Endpoint 0 descriptors.

6.2.1. Endpoints

IT9500 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. Besides, the host can access (read) the TS input thru USB EP5, and transmitter data from host to device thru USB EP6.

6.2.2. USB Control Protocol

Basic USB transfer and control scenario detail as follow:

Default Endpoint

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

Control Messages

Proprietary control messages are sent through a request-and-reply model. Any request packet corresponds to a reply packet, unless the communication is malfunctioning. A sequence number field is employed in each control packet to resolve the late reply and duplicate request/reply problems.

The available control messages include those for getting the current configuration, downloading the firmware, computing firmware checksum, booting IT9500, copying the firmware to a slave device, reading and writing the IT9500 memory, as well as IIC bus control messages, software reset control messages, and Control Unit command control messages.

Data Messages

Data messages are used for convey transport stream by IT9500.

7. EEPROM and IR Remote Controller

When the control bus is USB, an external EEPROM can be used for storing some system parameters. Besides, there is an IR interface which can be used to support NEC, RC-5, or RC-6 remote controller. There two features are very helpful specifically when IT9500 is designed as a USB dongle.

7.1. EEPROM Format

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 15.

Table 15 IT9500 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		Slave IIC bus Address	Suspend mode	IR remote type		TS interface type	
0x38								
		Calibration Type						
0x48								
0x50								
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.

Slave IIC bus Addr:: Slave demodulator IIC 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)

TS interface type: No Ts input, Eagle only(3), TS-IN / Parallel mode(4), TS-IN / Serial mode(5)

Calibration Type(Device type): **EVB board (0), DB-01-01-V01(1), DB-01-02-V01(2), DB-01-01-V03(3).**

7.2. IR Interface

IT9500 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, IT9500 is considered as a USB composite device with HID. It uses endpoint 3 as the HID interrupt endpoint. In this mode, IT9500 translates raw IR codes into HID codes according to a translation table. The HID translation table can be downloaded into IT9500 by the USB driver via the memory write protocol.

In the raw code mode, IT9500 sends IR raw codes (decoded according to NEC, RC5, or RC6 protocol, but not transformed into an HID code) via control command (EP2) and reply (EP1) messages.

7.2.1. 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.

7.2.2. IR Table

The IR table is a contiguous block of memory in IT9500 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 IT9500 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.

MAX_IR_N_KEYS

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

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.

IR_nKeys

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

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.

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.

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 16 and Table 17.

Table 16 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 17 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 IT9500, 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 18.

Table 18 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 19. 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 19 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 20 and Table 21, respectively. It is assumed that the second key function is supported by this remote control and the alternative key is "3."

With Table 20, this remote control supports the second key function, with the second key being "3."

Table 20 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 21, this remote control supports the second key function, with the second key being “3.”

Table 21: 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

7.3. Generating IR Table

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

8. Boot Description

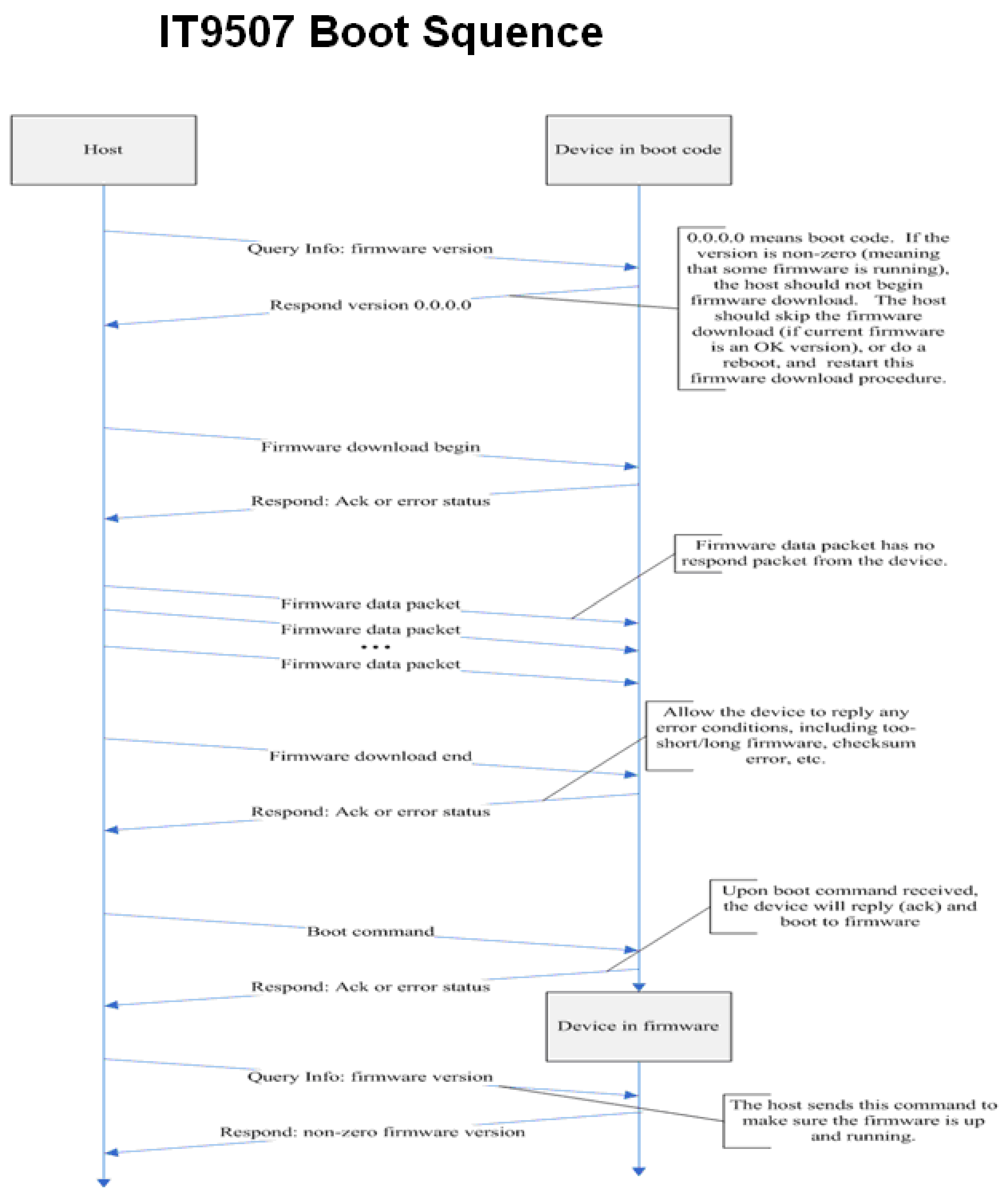

IT9507 Boot Squence
Host
Device in boot code
Query Info: firmware version
Respond version 0.0.0.0
0.0.0.0 means boot code. If the version is non-zero (meaning that some firmware is running), the host should not begin firmware download. The host should skip the firmware download (if current firmware is an OK version), or do a reboot, and restart this firmware download procedure.
Firmware download begin
Respond: Ack or error status
Firmware data packet has no respond packet from the device.
Firmware data packet
Firmware data packet
• • •
Firmware data packet
Allow the device to reply any error conditions, including too-short/long firmware, checksum error, etc.
Firmware download end
Respond: Ack or error status
Upon boot command received, the device will reply (ack) and boot to firmware
Boot command
Respond: Ack or error status
Device in firmware
Query Info: firmware version
Respond: non-zero firmware version
The host sends this command to make sure the firmware is up and running.

Figure 7 IT9500 boot sequence.

The above picture shows the procedure (protocol) used to boot IT9500. 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 IIC Bus Mode

In the IIC 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.

Appendix A: IT9500 Pin Description and Strapping

Pin Configuration (9503 only different in print)

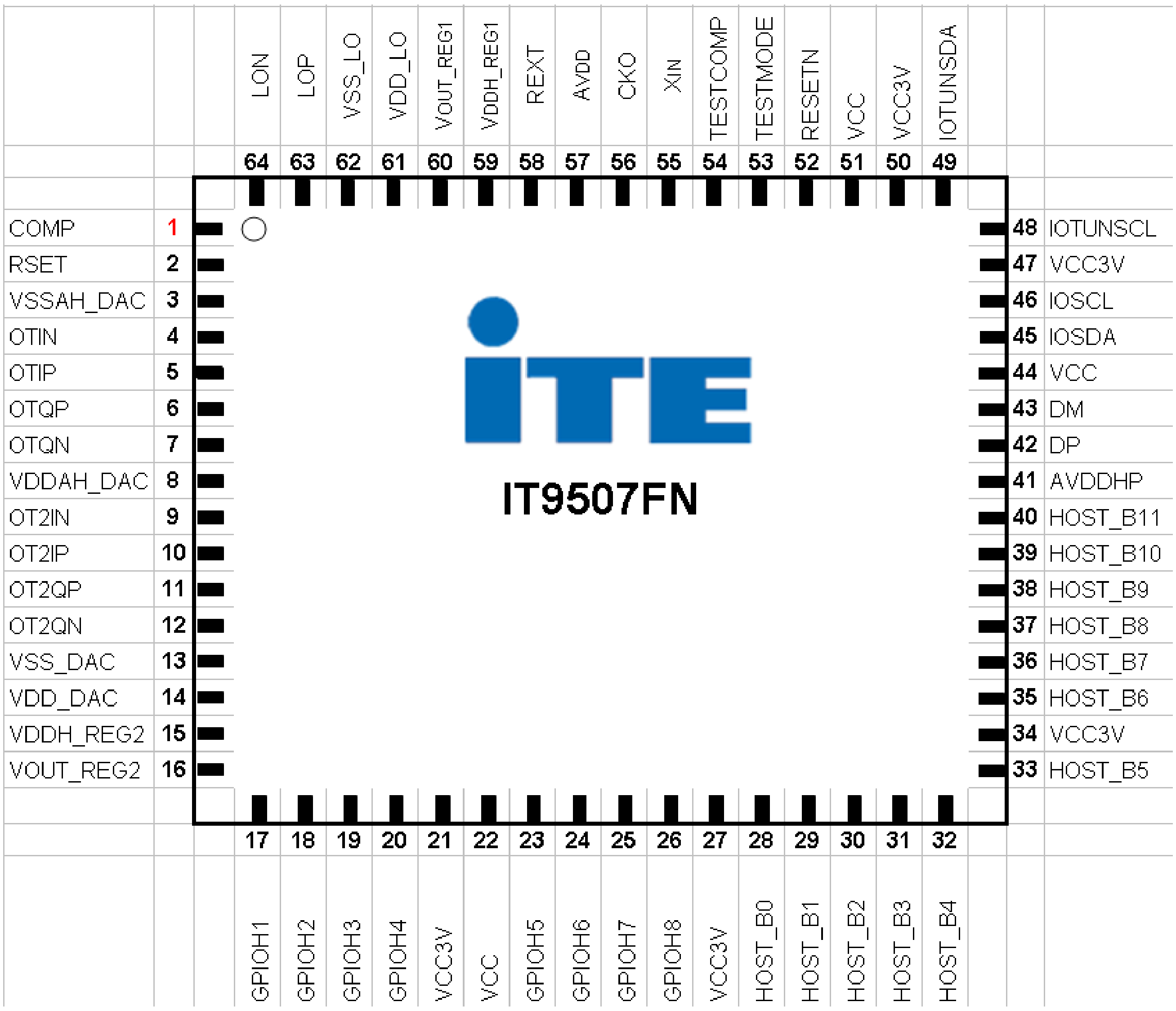


Figure 8 IT9500 pin configuration

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

Pin Name	Selection
{ GPIOH3,GPIOH2,GPIOH1}	Crystal frequency: 000 crystal = 12 MHz

Pin Name	**Mode and IIC bus address**
{ GPIOH7,GPIOH6,GPIOH5}={0,0,x}	TS/IIC mode **{ GPIOH5}= IIC bus address bit 1** **{0} -> 0x38(R)/0x39(W)** **{1} -> 0x3A(R)/0x3B(W)**
{ GPIOH7,GPIOH6,GPIOH5}={1,0,x}	TS/USB mode
{ GPIOH7,GPIOH6,GPIOH5}={1,1,x}	USB mode

Appendix B: Register Table

When accessing the following registers, the `processor` argument of IT9507_`readRegister()` and IT9507_`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 23 IT9500 register table in OFDM processor.

Address	Type	Name	Default	Description
0xF701 0xF700	RW	reg_intlv_sym_th	'h5E6	specifies the gap between read and write pointer of symbol interleaving
0xF702	RW	reg_fec_sw_rst	1'b0	1: Reset OFDM
0xF703	RW	reg_sync_byte_inv	1'b1	1: the MPEG2 sync byte of first packet in a group of 8 packets is bit-wise inverted from 8'h47 to 8'hB8
0xF708	RW	reg_fixed_mpeg_en	1'b0	1: Fixed MPEG TS in
0xF70A 0xF709	RW	reg_fixed_mpeg_hdr[15:0]	'h1FFF	Partial header of fixed MPEG TS Bit 15:Transport Error Indicator(TEI) Bit 14: Payload Unit Start Indicator Bit 13: Transport Priority Bit12~0: Packet ID(PID)
0xF713	RW	reg_fft_vld_prd[4:0]	'd4	reg_fft_vld_prd = floor(elementary period / clk_phy period) 8MHz: ‘d4, 7MHz: ‘d5, 6MHz: ‘d5, 5MHz: ‘d7, 4MHz: ‘d8, 3MHz: ‘d11, 2MHz: ‘d17
0xF720	RW	reg_tps_leng_ind[5:0]	6'b011111	Whether to support Cell Identification
0xF721	RW	reg_tps_const[1:0]	2'b00	Modulation scheme (00: QPSK, 01: 16QAM, 10: 64QAM, 11: reserved)
0xF722	RW	reg_tps_hier[2:0]	3'b000	Hierarchy Information (000: Non hierarchical)
0xF723	RW	reg_tps_hpcr[2:0]	3'b000	Coding rate of HP (000: 1/2, 001: 2/3, 010: 3/4, 011: 5/6, 100: 7/8, others: reserved)

0xF724	RW	reg_tps_lpcr[2:0]	3'b000	Not support hierarchical transmission
0xF725	RW	reg_tps_gi[1:0]	2'b00	CP ratio (11: 1/4, 10: 1/8, 01: 1/16, 00: 1/32
0xF726	RW	reg_tps_txmod[1:0]	2'b00	Transmission mode (00: 2K, 01: 8K, others: reserved)
0xF728 0xF727	RW	reg_tps_cell_id[15:0]	'h0	the cell Identifier
0xF729	RW	reg_tps_rsv[5:0]	'h0	Reserved bits
0xF740	RW	reg_intp_vld_phase	1'b0	
0xF741	RW	reg_intp_ds[2:0]	1'b0	down sample rate extension 0: 21/1, 1: 21/2, **2: 21/3**, 3: 21/4, 4: 21/6
0xF751	RW	reg_iqik_sine	1'b0	1: Turn on sine wave test mode
0xF753 0xF752	RW	reg_iqik_c1[10:0]	'h3FF	IQIK coefficient 1
0xF755 0xF754	RW	reg_iqik_c2[10:0]	'h0	IQIK coefficient 2
0xF757 0xF756	RW	reg_iqik_c3[10:0]	'h3FF	IQIK coefficient 3
0xF758	RW	reg_iqik_iq_switch	'h0	1: switch IQ
0xF759	RW	reg_iqik_msb_inv	'h0	1: revert MSB of I/Q
0xF75D 0xF75C	RW	reg_iqik_fix_i [9:0]	10'b0	fixed output I to DAC, Q(10,0)
0xF75F 0xF75E	RW	reg_iqik_fix_Q [9:0]	10'b0	fixed output Q to DAC, Q(10,0)
0xF760	RW	reg_iqik_fix_mode	1'b0	output to DAC in fixed value mode
0xF761	RW	reg_iqik_i_neg	1'b0	set output I to –I to DAC
0xF762	RW	reg_iqik_Q_neg	1'b0	set output Q to –Q to DAC
0xF763	RW	reg_iqik_dc_i[7:0]	'h0	DC Compensation for I channel, Q(8,0)

0xF765	RW	reg_iqik_dc_q[7:0]	'h0	DC Compensation for Q channel, Q(8,0)

When accessing the following registers, the processor argument of IT9507_readRegister () and IT9507_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 24 IT9500 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, IIC_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	IIC interface speed control in units of 400 ns Period of SCL = (reg_one_cycle_counter_tuner * 400) ns	8'h10

Appendix C: Generic IIC support in Boot-code

Generic IIC Read command:

This command cause the FW to initiate one IIC read cycle to the specified IIC interface, with specified IIC 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 IT9500 to complete the IIC read cycle. Otherwise the attempt to get the reply packet will be negatively acknowledged (although you may retry to receive the reply packet.)

Generic IIC Read Command packet

Field	Size	Comments
Command Length	1 byte	Number of the following bytes, always 4.
Command	1 byte	0x06, Generic IIC Read command
Interface #	1 byte	1, 2, or 3. IT9500 series normally has three IIC 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.
IIC Address + Read bit	1 byte	Bit [7~1]: IIC address Bit [0]: Read bit, must be 1, according to IIC specification
Number of bytes to read	1 byte	# of bytes to read: max 61 for USB interface, or 253 for all other interfaces

Generic IIC 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 IIC Write command:

This command cause the FW to initiate one IIC write cycle to the specified IIC interface, with specified IIC slave address, and write the specified bytes. This command has no reply packet. The host should wait for some time to allow IT9500 to complete the IIC write cycle. Otherwise, new commands will be negatively acknowledged (although you may retry the command until acknowledged.)

Generic IIC 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 IIC Write command
Interface #	1 byte	1, 2, or 3. IT9500 series normally has three IIC 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.
IIC Address + Write bit	1 byte	Bit [7~1]: IIC address Bit [0]: Write bit, must be 0, according to IIC specification
Write data bytes	n byte	Max number of bytes: 59 for USB mode, and 251 for all other interfaces.

Appendix D: IT9507 AFE LO register table

A006 LO0 register----Address : 0XFB2A

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	1	1	0	0	0
lo_freq[7:0]							

A007 LO1 register ----Address : 0XFB2B

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	0	1	0	0
lo_freq[15:8]							

LO1[D7-D0], LO0[D7-D0]: lo_freq[15:0]: LO frequency setting.

Default 16b1011_0100_0101_1000 = 0xB458, default LO frequency is 434MHz

A008 LO2 register ----Address : 0XFB2C

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	1	0	0	0	0
lo_icpset[2:0]			lo_ibset[1:0]		lo_lband	lo_pdiv[1:0]	

D7-D5: lo_icpset[2:0]: LO charge pump current setting, default 3b010. Change pump current increases from low to high vs. setting value from 3b000 to 3b100

D4-D3: lo_ibset[1:0]: LO charge pump bias current setting, default 2b10.

D2: lo_lband: LO L-band setting, default 0. 0: VHF and UHF; 1: L-band.

D1-D0: lo_pdiv[1:0]: LO pre-divider setting, default 2b00.

Table 25 LO pre-divider settings.

lo_pdiv[1:0]	**f_ref**
00	f_ref = f_xtal /18 (f_xtal=12MHz)

01	f_ref = f_xtal / 32 (f_xtal=20.48MHz)
10	f_ref = f_xtal / 30
11	reserved

A009 LO3 register ----Address : 0XFB2D

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0
not in use						lo_strap_sel	lo_rst

D7-D2: not in use.

D1: lo_strap_sel: LO strap selection, default 0.

1: apply lo divider number register to circuit.

0: do not apply lo divider number register to circuit

lo_strap_sel straps divider number registers of lo, including lo_freq[15:0], and lo_lband. To change lo output clock frequency, one may set all lo divider number registers first, and then change lo_strap_sel from low to high. With rising of lo_strap_sel, lo will update all divider numbers simultaneously and change its output clock frequency.

D0: lo_rst: LO reset signal, default 0. 1: reset.

Appendix E: Hardware Registers for SI/PSI Table Insertion

(1) PSI Table Access

Wr ---- reg_psi_dat 188 times to fill 188 byte PSI packet

Wr ---- reg_psi_access 1 to start PSI Table access

Rd ---- reg_psi_access until 0 to make sure PSI Table access is complete

(2) Register Table

Name	Addr.	bit	default	Description
reg_psi_access	F980	0	1'h0	write to 1 to enable PSI Table access; return 0 means access complete
reg_psi_dat	F981	7:0	8'h0	PSI Table Data
reg_psi_index	F982	7:0	8'h0	PSI Table Index (0 ~ 187)
reg_psi_acc_err	F983	0	1'h0	PSI access error (Data transfer at the same time). Write 1 to clear this bit.