MxL 5007T

Driver API User Guide

Revision 4.1.3

August 12, 2008

Revision History

Version	Date	Description
4.1.2	07/29/2008	Initial Release
4.1.3	08/12/2008	

Table of Contents

1. INTRODUCTION

The MxL5007T Driver API is developed to help the user to easily and quickly integrate the MxL5007T driver into their application software.

File description:

- MxL5007.c : The source file for MxL5007T driver
- MxL5007.h : The header file for MxL5007T driver
- MxL5007_API.c : The source file for MxL5007T API
- MxL5007_API.h : The header file for MxL5007T API
- MxL_User_Define.c : The source file of customer define function for MxL5007T API

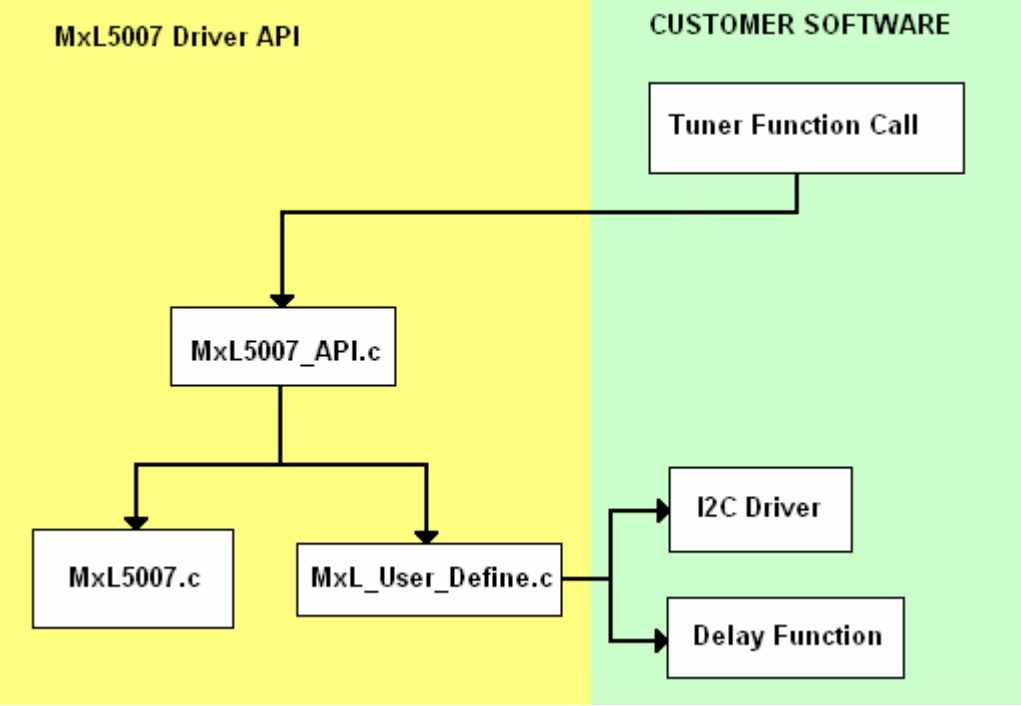

Figure 1 - API Software Structure

2. I2C PROTOCOL AND FORMAT

2.1 I2C Protocol Characteristics

For all practical purposes, the MxL 500T will be a slave device during the I2C programming. The I2C protocol for this case is completely characterized by the features described below. Conditions 1-7 are illustrated in Figure .

1) Start condition (S)
   Indicates the start of a new I2C transmission begins.
   Is generated whenever there is an SCL → low transition when SDA is low.
2) Stop condition (P)
   Indicates the completion of a transmission.
   Is generated whenever there is an SDA → high transition when the SCL is high.
3) Addressing (ADDR<6:0>)
   The slave address consists of 7 bits.
4) Read/write bit (R/Wb)
   Read/write bit (R/Wb) is high when reading from, and low when writing to the device.
5) Acknowledge (ACK)
   A correct address to the interface is acknowledged.
   During write mode, each byte is acknowledged.
   During read mode, the acknowledge bit must be left available (i.e. not pulled down).
   When the device is not being addressed, ensure no spurious pull-down.
6) Read-back
   Allows read-back of bits from the device.
7) Start repeat condition (SR)
   The master device does not need to issue a stop condition to write to another (or the same) slave device- though it still needs to send out the address and R/Wb after the SR.
   This mode can be used to indicate High Speed mode.
8) Frequency range
   The maximum clock frequency specified for the I2C bus is 400kHz in Fast Mode.
   Timing: All setup and hold times are 160ns min, and all high and low periods are >160ns for 100pF Cload.

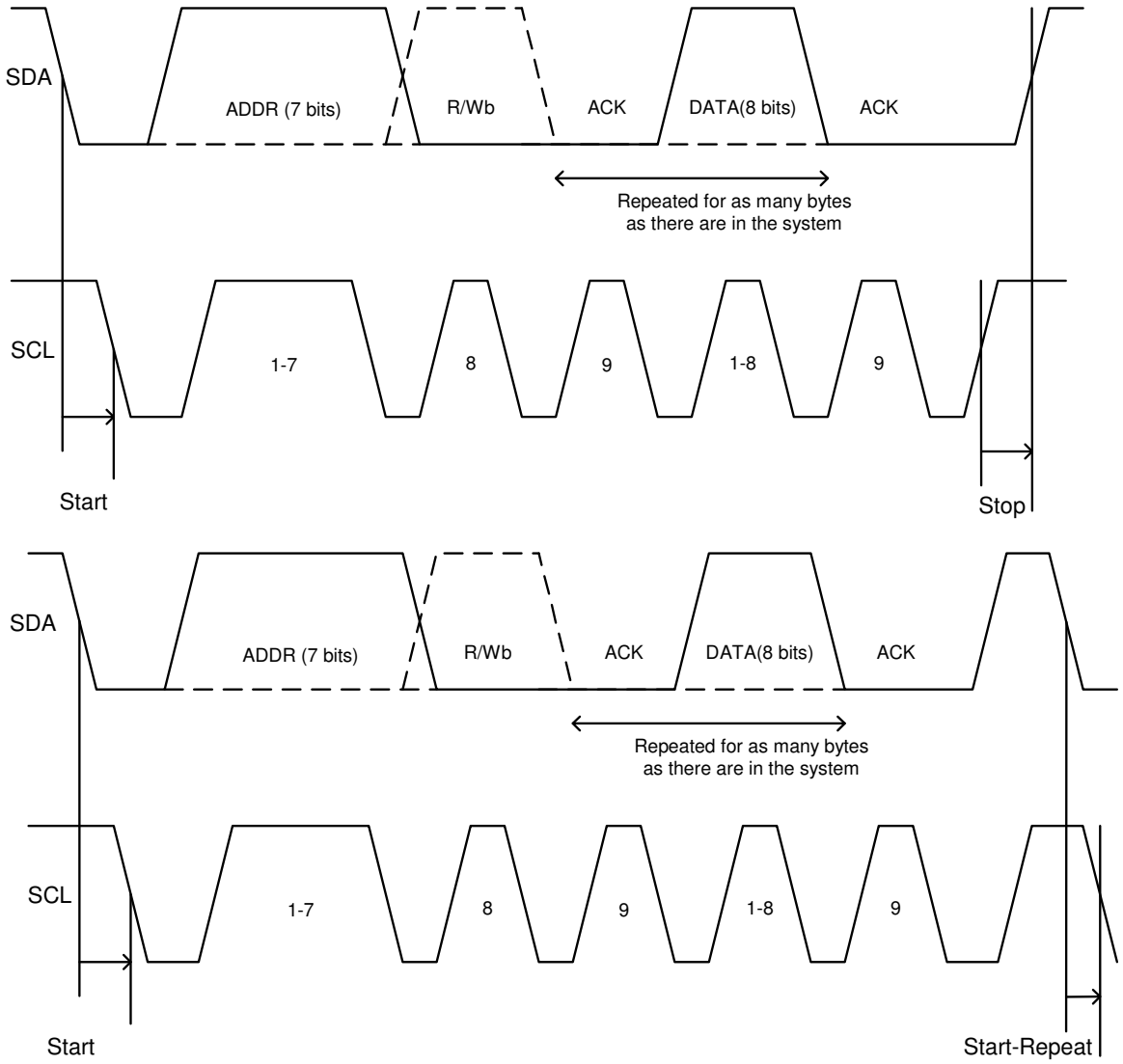

Figure 2 - Illustration of start, stop, start-repeat, address, read/write bit, acknowledge, and data signaling for the I2C bus specification. MSB is transmitted first

2.2 I2C Programming Slave Address

The I2C slave address is a 7 bit word. The address is (MSB to LSB) 11000XX, where X denotes a programmable bit. The two LSBs are set by choosing an appropriate external resistor connected between the AS pin and ground. The correspondence between the resistor value and the I2C slave address <6:0> is given in Table 1

Table 1 - Required Resistor values on AS pin for the 4 different I2C slave addresses

Resistor Value (Ohms)	I2C Slave Address <6:0>	I2C Slave Address (DEC)
Open	1100011	99
4.7k	1100010	98
2.4k	1100001	97
0	1100000	96

2.3 I2C Operating Modes for MXL5007T

There are 3 modes of operation for I2C programming for MxL5007T. These are 1) Address Initiated Command (AIC) Reset, 2) Write Mode and 3) Read Mode.

2.3.1 Address Initiated Command (AIC) Reset

Address Initiated Command (AIC) can be used to perform specific operations on the IC by sending specially allocated address values. The format is shown in Table 2. The first step involves configuring the IC (slave) into write mode. The next step involves writing a specific address that will initiate a command on the IC. Only one such command is possible on MxL5007T which is shown below.

AIC Reset: 0xFF, this command resets all the control registers to their chip default values

Table 2 - I2C Data format for AIC Reset

S	DEVICE ID <6:0>	R/Wb	ACK	AIC <7:0>	ACK	P

S = Start condition
DEVICE ID<6:0> = <1100001> = $[97]_{dec}$
R/Wb = Read /Writeb, 0 for write
ACK = Acknowledge
AIC <7:0> =Address initiated reset, 0xFF
ACK = Acknowledge
P = Stop condition

2.3.2 Write Mode

In this mode, data is written to control registers on the IC. The first step involves configuring the IC (slave) into write mode. This is followed by a writing a sequence of 8 bit register address and 8 bit data to the IC. Finally a stop command is issued to complete the write sequence. There are several ways to write data into registers if it involves more than one byte. A couple of examples for writing 2 bytes of data are illustrated below in Table 3 and Table 4. It is possible to write 2 bytes of data in a single step as shown in Table 3.This can be extended to more than 2 bytes of data in a similar fashion.

Table 3 - I2C Data format for Write Mode, example 1

S	DEVICE ID <6:0>	R/Wb	ACK	RG_ADD1 <7:0>	ACK	DATA1 <7:0>	ACK	RG_ADD2 <7:0>	ACK	DATA2 <7:0>	ACK	P

S = Start condition
DEVICE ID<6:0> = <1100001> = $[97]_{dec}$
R/Wb = Read /Writeb, 0 for write
ACK = Acknowledge
RG ADD1<7:0> = Register Address of 1st BYTE
ACK = Acknowledge
DATA1 <7:0> = 1st BYTE Data
ACK = Acknowledge
RG ADD2<7:0> = Register Address of 2nd BYTE
ACK = Acknowledge
DATA2<7:0> = 2nd BYTE Data
ACK = Acknowledge
P = Stop condition

Sometimes, due to memory size limitations on the customer application boards, it is possible to write only a few bytes of data at a time. In such a situation, it is possible to send data to registers in multiple steps. This is illustrated in Table 4 where data is sent to one register at a time.

Table 4 - I2C Data format for Write Mode, example 2 using 2 steps

Step1

S	DEVICE ID <6:0>	R/Wb	ACK	RG_ADD1<7:0>	ACK	DATA1 <7:0>	ACK	P

Step 2

S	DEVICE ID <6:0>	R/Wb	ACK	RG_ADD2<7:0>	ACK	DATA2 <7:0>	ACK	P

Step 1
S = Start condition
DEVICE ID<6:0> = <1100001> = $[97]_{dec}$
R/Wb = Read /Writeb, 0 for write
ACK = Acknowledge
RG ADD1<7:0> = Register Address of 1st BYTE
ACK = Acknowledge
DATA1 <7:0> = 1st BYTE Data
ACK = Acknowledge
P = Stop

Step 2
S = Start condition
DEVICE ID<6:0> = <1100001> = $[97]_{dec}$
R/Wb = Read /Writeb, 0 for write
ACK = Acknowledge
RG ADD2<7:0> = Register Address of 1st BYTE
ACK = Acknowledge
DATA2<7:0> = 2nd BYTE Data
ACK = Acknowledge
P = Stop condition

2.3.3 Read Mode

In this mode, data is read back from some of the registers on the IC. The format for Read operation is show in Table 5. The read back operation includes two steps. The first step configures the IC (slave) in write mode. This is followed by ***writing*** the address of the register whose contents are to be read back to a register with address 0xFB. The second step configures the IC (slave) in ***read*** mode. After this the I2C master releases control of the SDA line to MxL5007T which together with the pull up resistor on board will facilitate read back of the register contents. After this, MxL5007T (slave) sends data which is shifted out on the rising edge of the clock pulses on the SCL line.

Table 5 - I2C Data Format for Read Mode

Step 1

S	DEVICE ID <6:0>	R/Wb	ACK	0xFB	ACK	READ RG ADD<7:0>	ACK	P

Step 2

S	DEVICE ID <6:0>	R/Wb	ACK	READ DATA<7:0>	NACK	P

Step 1

S	= Start condition
DEVICE ID<6:0>	= <1100001> = $[97]_{dec}$
R/Wb	= Read /Writeb, 0 for write
ACK	= Acknowledge
0xFB	= Address of the register which stores the contents of the register address from which data is to be read out
ACK	= Acknowledge
READ RG ADD <7:0	= Address of the register whose data is to be read out
ACK	= Acknowledge
P	= Stop condition

Step 2

S	= Start condition
DEVICE ID<6:0>	= <1100011> = $[99]_{dec}$
R/Wb	= Read /Writeb, 1 for Read
ACK	= Acknowledge
READ DATA<7:0>	= The data scrolls out of the IC during this time. During this time, the I2C master releases the control of SDA line. The pull up resistor on the board on SDA line is critical for the proper operation of read mode as the SDA line could be pulled down or not by MXL5007T IC depending on whether a “0” or “1” is being sent onto SDA line.
NACK	= During this the I2C master acknowledges. This has no relevance to MXL5007T IC as it completely ignores the NACK pulse.
P	= Stop condition

3. MxL5007T API FUNCTIONS

3.1 User Define Function for MxL5007T API

The user-defined functions are located in MxL_User_Define.c. The MxL_User_Define.c provides utility function such as I2C read/write and delay function which are called by MxL5007T API. The user is required to implement the functions listed in MxL_User_Define.c.

3.1.1 : MxL_I2C_Write

```
/**************************************************************************
**
** Name: MxL_I2C_Write
**
** Description:    I2C write operations
**
** Parameters:
**                  DeviceAddr        : MxL5007T Device address
**                  pArray            : Write data array pointer
**                  count             : total number of element in pArray
**
** Returns:       0 if success
**
**************************************************************************/
UINT32 MxL_I2C_Write(UINT8 DeviceAddr, UINT8* pArray, UINT32 count);
```

Description:

MxL_I2C_Write function performs I2C write to MxL5007T. After this function is called, the following I2C command is sent to MxL5007T.

S	DeviceAddr <6:0>	0	ACK	pArray <0>	ACK	pArray <1>	ACK	……	pArray <count-1>	ACK	P

3.1.2 : MxL_I2C_Read

```
/**************************************************************************
**
** Name: MxL_I2C_Read
**
** Description:    I2C read operations
**
** Parameters:
**                  DeviceAddr        : MxL5007T Device address
**                  Addr              : register address for read
**
** Returns:       Read Data in unsigned char data type
**
**************************************************************************/
UINT32 MxL_I2C_Read(UINT8 DeviceAddr, UINT8 Addr, UINT8* mData);
```

Description:

MxL_I2C_Read function reads one 8-bit register from MxL5007T through I2C. After this function is called, the following I2C command is performed and the function returns the 8bit data READ DATA<7:0>.

Step 1

S	**DeviceAddr <6:0>**	**0**	**ACK**	**0xFB**	**ACK**	**Addr<7:0>**	**ACK**	**P**

Step 2

S	**DEVICE ID <6:0>**	**1**	**ACK**	**READ DATA<7:0>**	**NACK**	**P**

3.1.3 : MxL_Delay

```
/**************************************************************************
**
** Name: MxL_Delay
**
** Description:    Delay function in millisecond
**
** Parameters:
**                  mSec     : millisecond to delay
**
** Returns:        None
**
**************************************************************************/
void MxL_Delay(UINT32 mSec);
```

Description:

MxL_Delay function performs delay in millisecond.

3.2 Function for MxL5007T API

The MxL5007T API provides all the required functions in programming the MxL5007T and checking MxL5007T's status.

3.2.1 : MxL_Set_Register

```
/**************************************************************************
**
**  Name: MxL_Set_Register
**
**  Description:    Write one register to MxL5007T
**
**  Parameters:
**                  myTuner: pointer to MxL5007_TunerConfigS
**                  RegAddr: Register address to be written
**                  RegData: 8-bit Data to be written
**
**  Returns:
**                  MxL_ERR_MSG  : MxL_OK if success
**                               : MxL_ERR_SET_REG if fail
**
**************************************************************************/
MxL_ERR_MSG MxL_Set_Register(MxL5007_TunerConfigS* myTuner, UINT8 RegAddr, UINT8 RegData);
```

Description:

MxL_Set_Register function writes 8-bit data to one specified register address of MxL5007T.

3.2.2 : MxL_Get_Register

```
/**************************************************************************
**
**  Name: MxL_Get_Register
**
**  Description:    Read one register from MxL5007T
**
**  Parameters:
**                  myTuner: Pointer to MxL5007_TunerConfigS
**                  RegAddr          : Register address to be read
**                  RegData: Point to readback register value
**
**  Returns:
**                  MxL_ERR_MSG : MxL_OK if success
**                                : MxL_ERR_GET_REG if fail
**
**************************************************************************/
MxL_ERR_MSG MxL_Get_Register(MxL5007_TunerConfigS* myTuner, UINT8 RegAddr, UINT8* RegData);
```

Description:

MxL_Get_Register function reads and returns 8-bit data from one specified register address of MxL5007T.

3.2.3 : MxL_Tuner_Init

```
/**************************************************************************
**
**  Name: MxL_Tuner_Init
**
**  Description:    MxL5007T Initialization
**
**  Parameters:
**                  myTuner: Pointer to MxL5007_TunerConfigS
**
**  Returns:
**                  MxL_ERR_MSG  : MxL_OK if success
**                               : MxL_ERR_INIT if fail
**
**************************************************************************/
MxL_ERR_MSG MxL_Tuner_Init(MxL5007_TunerConfigS* myTuner);
```

Description:

MxL_Tuner_Init function initializes the MxL5007T.

3.2.4 : MxL_Tuner_RFTune

```
/**************************************************************************
**
**  Name: MxL_Tuner_RFTune
**
**  Description:    Frequency tuning for channel
**
**  Parameters:
**                  myTuner: Pointer to MxL5007_TunerConfigS
**                  RF_Freq_Hz      : RF Frequency in Hz
**                  BWMHz           : Bandwidth 6, 7 or 8 MHz
**
**  Returns:
**                  MxL_ERR_MSG  : MxL_OK if success
**                               : MxL_ERR_RFTUNE if fail
**
**************************************************************************/
MxL_ERR_MSG MxL_Tuner_RFTune(MxL5007_TunerConfigS* myTuner, UINT32 RF_Freq_Hz,
MxL5007_BW_MHz BWMHz);
```

Description:

MxL_Tuner_RFTune function sets the receiving RF frequency and bandwidth of the MxL5007T.

3.2.5 : MxL_Soft_Reset

```
/**************************************************************************
**
**  Name: MxL_Soft_Reset
**
**  Description:    Software Reset the MxL5007T Tuner
**
**  Parameters:
**                  myTuner: Pointer to MxL5007_TunerConfigS
**
**  Returns:
**                  MxL_ERR_MSG  : MxL_OK if success
**                               : MxL_ERR_OTHERS if fail
**
**************************************************************************/
MxL_ERR_MSG MxL_Soft_Reset(MxL5007_TunerConfigS* myTuner);
```

Description:

MxL_Soft_Reset function resets all the MxL5007T registers to the default values.

3.2.6 : MxL_Loop_Through_On:

```
/**************************************************************************
**
**  Name: MxL_Loop_Through_On
**
**  Description:    Turn On/Off on-chip Loop-through
**
**  Parameters:     myTuner                         - Pointer to MxL5007_TunerConfigS
**                  isOn                            - MxL5007_LoopThru
**
**  Returns:        MxL_ERR_MSG                     - MxL_OK if success
**                                                  - MxL_ERR_OTHERS if fail
**
**************************************************************************/
MxL_ERR_MSG MxL_Loop_Through_On(MxL5007_TunerConfigS*, MxL5007_LoopThru);
```

Description:

MxL_Loop_Through_On function turns on/off the on-chip loop-through. Loop-through is always on after initialization, user can turn it off by calling this function.

3.2.7 : MxL_Stand_By:

```
/**************************************************************************
**
**  Name: MxL_Standby
**
**  Description:    Enter Standby Mode
**
**  Parameters:    myTuner                          - Pointer to MxL5007_TunerConfigS
**
**  Returns:       MxL_ERR_MSG                      - MxL_OK if success
**                                                  - MxL_ERR_OTHERS if fail
**
**************************************************************************/
MxL_ERR_MSG MxL_Stand_By(MxL5007_TunerConfigS*);
```

Description:

MxL_Stand_By function set MxL5007T into stand by mode.

3.2.8 : MxL_Wake_Up:

```
/**************************************************************************
**
**  Name: MxL_Wake_Up
**
**  Description:    Wake Up from Stand by Mode
**
**  Parameters:    myTuner                          - Pointer to MxL5007_TunerConfigS
**
**  Returns:       MxL_ERR_MSG                      - MxL_OK if success
**                                                  - MxL_ERR_OTHERS if fail
**
**************************************************************************/
MxL_ERR_MSG MxL_Wake_Up(MxL5007_TunerConfigS*);
```

Description:

MxL_Wake_Up function wakes up MxL5007T from stand by mode.

3.2.9 : MxL_Check_ChipVersion

```
/**************************************************************************
**
**  Name: MxL_Check_ChipVersion
**
**  Description:    Return the MxL5007T Chip Version
**
**  Parameters:
**                  myTuner: Pointer to MxL5007_TunerConfigS
**
**  Returns:        MxL5007_ChipVersion
**
**************************************************************************/
MxL5007_ChipVersion MxL_Check_ChipVersion(MxL5007_TunerConfigS* myTuner);
```

Description:

MxL_Check_ChipVersion function returns the MxL5007_ChipVersion.
MxL5007_ChipVersion shows the chip version and it is defined in MxL5007_Common.h.

3.2.10 : MxL_RFSynth_Lock_Status

```
/**************************************************************************
**
**  Name: MxL_RFSynth_Lock_Status
**
**  Description:    RF synthesizer lock status of MxL5007
**
**  Parameters:
**                  myTuner: Pointer to MxL5007_TunerConfigS
**                  isLock              : RF Synthesizer Lock Status
**
**  Returns:
**                  MxL_ERR_MSG  : MxL_OK if success
**                               : MxL_ERR_OTHERS if fail
**
**************************************************************************/
MxL_ERR_MSG MxL_RFSynth_Lock_Status(MxL5007_TunerConfigS* myTuner, BOOL* isLock);
```

Description:

MxL_RFSynth_Lock_Status function returns the lock status of the MxL5007T RF Synthesizer.

3.2.11 : MxL_REFSynth_Lock_Status

```
/**************************************************************************
**
**  Name: MxL_REFSynth_Lock_Status
**
**  Description:    REF synthesizer lock status of MxL5007
**
**  Parameters:
**                  myTuner: Pointer to MxL5007_TunerConfigS
**
**  Returns:
**                  MxL_ERR_MSG  : MxL_OK if success
**                                 : MxL_ERR_OTHERS if fail
**
**************************************************************************/
MxL_ERR_MSG MxL_REFSynth_Lock_Status(MxL5007_TunerConfigS* myTuner, BOOL* isLock);
```

Description:

MxL_RFSynth_Lock_Status function returns the lock status of the MxL5007 Reference Synthesizer.

4. IMPLEMENTATION EXAMPLE

The following is an example of calling the API function. For more detail example, please see MxL5007T_API_Example.

```
#include "MxL5007_Common.h"
#include "MxL5007_API.h"

MxL_ERR_MSG Status = MxL_OK;
BOOL RFSynthLock, REFSynthLock;
SINT32 RF_Input_Level;
MxL5007_TunerConfigS myTuner;

//Set Tuner's I2C Address
myTuner.I2C_Addr = MxL_I2C_ADDR_96;

//Set Tuner Mode to DVB-T/ATSC mode
myTuner.Mode = MxL_MODE_OTA_DVBT_ATSC;

//Setting for Cable mode only
myTuner.IF_Diff_Out_Level = -8;

//Set Tuner's XTAL freq
myTuner.Xtal_Freq = MxL_XTAL_16_MHZ;

//Set Tuner's IF Freq
myTuner.IF_Freq = MxL_IF_4_MHZ;

//Set Tuner's Clock out setting
myTuner.ClkOut_Setting = MxL_CLKOUT_ENABLE;
myTuner.ClkOut_Amp = MxL_CLKOUT_AMP_0;

//Init Tuner
if(Status = MxL_Tuner_Init(&myTuner))
        //Init Tuner fail

//Tune Tuner
if(Status = MxL_Tuner_RFTune(&myTuner, 666*MHz, MxL_BW_8MHz))
        //Tune Tuner fail

//Check Lock Status
MxL_RFSynth_Lock_Status(&myTuner, &RFSynthLock);
MxL_REFSynth_Lock_Status(&myTuner, &REFSynthLock);

//Enter Stand By Mode
MxL_Stand_By(&myTuner);

//Wake Up
MxL_Wake_Up(&myTuner);
```