

## Programmer's Guide

### Synthesized Signal Generator

Hittite Model: HMC-T2000



---

Hittite Microwave Corporation, 20 Alpha Road, Chelmsford, MA 01824

Tel: (978) 250-3343 • Fax: (978) 250-3373 • E-mail: [sales@hittite.com](mailto:sales@hittite.com) • Internet: <http://www.hittite.com>

Applications Support: [applications@hittite.com](mailto:applications@hittite.com)

1	Introduction.....	3
2	Programmers Reference Guide .....	3
2.1	Synthesizer operating flow .....	3
2.1.1	Synthesizer setup .....	3
2.1.2	Map Synthesizers .....	3
2.1.3	Query available Synthesizers .....	3
2.1.4	Open desired Synthesizer .....	3
2.1.5	Enable Remote operation .....	3
2.1.6	Program Synthesizer .....	3
2.1.7	Close Unit .....	4
2.2	C Language Support.....	4
2.2.1	HMC_ERROR_CODES.....	4
2.2.2	HMC_MapSynths(void) .....	6
2.2.3	HMC_NumSynths(void).....	6
2.2.4	HMC_GetIDNByNum(int UnitNum, const char** IDN).....	7
2.2.5	HMC_GetIDNByHandle(void* SynthHandle, const char** IDN).....	8
2.2.6	HMC_Open(int UnitNum, void** SynthHandle) .....	8
2.2.7	HMC_Close(void* SynthHandle) .....	9
2.2.8	HMC_IsOpen(const void* SynthHandle) .....	10
2.2.9	HMC_RemoteAndLocalLockout(void* SynthHandle) .....	10
2.2.10	HMC_GoToLocal(void* SynthHandle).....	10
2.2.11	HMC_Frequency(void* SynthHandle, double Frequency_Hz) .....	10
2.2.12	HMC_GetFrequency(void* SynthHandle, double* Frequency_Hz).....	11
2.2.13	HMC_Power(void* SynthHandle, double Power_dBm).....	11
2.2.14	HMC_GetPower(void* SynthHandle, double* Power_dBm) .....	11
2.2.15	HMC_RFOut(void* SynthHandle, int Connection).....	12
2.2.16	HMC_GetRFOut(void* SynthHandle, int* Connection) .....	12
2.2.17	HMC_GetSerialByNum(int UnitNum, const char** Serial).....	12
2.2.18	HMC_GetSerialByHandle(void* SynthHandle, const char** Serial).....	13
2.2.19	HMC_GetDescriptionByNum(int UnitNum, const char** Description).....	13
2.2.20	HMC_GetDescriptionByHandle(void* SynthHandle, const char** Description) .....	13
2.2.21	HMC_GetFPGARevByNum(int UnitNum, const char** FPGARev) .....	14
2.2.22	HMC_GetFPGARevByHandle(void* SynthHandle, const char** FPGARev).....	14
2.2.23	HMC_GetModelByNum(int UnitNum, const char** Model) .....	14
2.2.24	HMC_GetModelByHandle(void* SynthHandle, const char** Model).....	14
2.2.25	HMC_TextByNum(int UnitNum, const char* Text) .....	15
2.2.26	HMC_TextByHandle(void* SynthHandle, const char* Text) .....	15
2.2.27	HMC_GetTextByNum(int UnitNum, const char** Text) .....	15
2.2.28	HMC_GetTextByHandle(void* SynthHandle, const char** Text) .....	16
2.2.29	HMC_DisablePowerLimitCheck(void* SynthHandle, int Disable) .....	16
2.2.30	HMC_GetDisablePowerLimitCheck(void* SynthHandle, int* Disable) .....	17
2.2.31	HMC_TimeoutsByNum(int UnitNum, unsigned long ReadTimeout_ms, unsigned long WriteTimeout_ms) 17	
2.2.32	HMC_GetTimeoutsByNum(int UnitNum, unsigned long* ReadTimeout_ms, unsigned long* WriteTimeout_ms) .....	18
2.2.33	HMC_TimeoutsByHandle(void* SynthHandle, unsigned long ReadTimeout_ms, unsigned long WriteTimeout_ms) .....	18
2.2.34	HMC_GetTimeoutsByHandle(void* SynthHandle, unsigned long* ReadTimeout_ms, unsigned long* WriteTimeout_ms) .....	19
3	Technical Support .....	19

## 1 Introduction

This Programmer's Guide provides advanced information for programmers that will generate their own application programs to control the HMC-T2000 Signal Generator unit (SG). Most of the pertinent information can be quickly located through the table of contents and highlighted sections.

## 2 Programmers Reference Guide

This section of the document defines the functions provided by the Hittite DLL's.

### 2.1 *Synthesizer operating flow*

This section defines the sequence of events required to control the synthesizers.

#### 2.1.1 **Synthesizer setup**

See para. 7., Equipment Setup and Operation. All Synthesizer units should be connected to a USB port, and power should be applied. The appropriate USB drivers should be installed on the attached computer.

#### 2.1.2 **Map Synthesizers**

The HMC\_MapSynths function should be called from the desired program. This function builds a database of the available Synthesizer units. This step must be performed before communications to the Synthesizer units can be established. Once this function is called, power and USB connections to the Synthesizers should not be interrupted.

#### 2.1.3 **Query available Synthesizers**

Use the HMC\_GetSerial, HMC\_NumSynths, and the HMC\_GetDescription functions to collect information on the attached Synthesizers. Synthesizers which have already been open by another process may not be visible.

#### 2.1.4 **Open desired Synthesizer**

Using the HMC\_Open function, open a Synthesizer.

#### 2.1.5 **Enable Remote operation**

Using the HMC\_RemoteEnable function, enable the Synthesizer to be controlled remotely via the USB connection.

#### 2.1.6 **Program Synthesizer**

Use the HMC\_Frequency function to program the specified unit's frequency. Use the HMC\_Power function to program the specified unit's power out. Use the HMC\_RFOut function to enable the RF output on the front panel.

### 2.1.7 Close Unit

Use the HMC\_Close function to disconnect the program from the computer via the USB.

## 2.2 C Language Support

This section defines the functions, their input arguments, and return values.

### General conventions:

Functions which write values just begin with HMC\_. (HMC\_Power() sets output power.)

Functions which read back values begin with HMC\_Get and have a pointer as their second argument. (HMC\_GetPower(handle, &power\_dBm))

Functions generally return HMC\_ERROR\_CODES, an enum typedef.

Functions ending in “ByNum” take an instrument number instead of a handle so they can be called before HMC\_Open(). This may help you decide which units should be opened.

Prototypes for the “C” functions, and the definition of their return values, can be found in HMC SynthC.h.

```
#include “HMC SynthC.h”
```

You may also need to add “HMC Synth.lib” to your project.

“HMC Synth.dll” (the functions below) and “FTD2xx.dll” (USB driver) must be in your path. They are most likely in “C:\Windows\System32.”

### 2.2.1 HMC\_ERROR\_CODES

Most of the HMC\_xxx() functions return an HMC\_ERROR\_CODES enum typedef. See “HMC SynthC.h”.

Some of the error codes you are more likely to see are:

**0 = HMC\_OK** – No error; everything seems to have worked.

**-2 = HMC\_FREQUENCY\_OUT\_OF\_RANGE** – The programmed frequency cannot be achieved by this model synthesizer.

**-3 = HMC\_POWER\_OUT\_OF\_RANGE** – The programmed power (dBm) is outside the specified operating range of the unit. Note that the maximum and minimum power settings are a function of frequency. See `HMC_DisablePowerLimitCheck()` to allow access to typical behavior of unit.

**-5 = HMC\_NO\_LONGER\_VALID** – The unit is not open. It may never have been opened, but if it had been working and then this error starts appearing then the unit has been closed because of an error. This can be caused by disconnecting the cable, turning the unit off, and closing the unit from one piece of code (debug window) without making sure it was not being used by another. It may be possible to call `HMC_Open()` to recover.

**-6 = HMC\_ALREADY\_OPEN** – Returned by `HMC_Open()` if the unit has already been opened by this process. Most of the time this is not really an error; it just happened because you are debugging something or trying something out, or opened the synthesizer from a debug window.

**-8 = HMC\_BAD\_UNIT\_NUMBER** – Make sure the specified unit number is greater than or equal to 0 and less than `HMC_NumSynths()`. This error can also happen if a unit which was not open when `HMC_MapHardware()` was called but has since been opened by another process. Only one process at a time is allowed to open a particular unit.

**-9 = HMC\_BAD\_POINTER** – Usually caused by a NULL pointer. For example, if `HMC_Open()` fails, setting `SynthHandle` to NULL, and a call to `HMC_GetSerialByHandle()` will return `HMC_BAD_POINTER`.

**-10 = HMC\_FREQUENCY\_ROUNDOFF** – If you write your code anticipating that the HMC-T2000 will always round to the nearest 1 MHz and then change to a HMC-T1000 which rounds to the nearest 1 mHz, the frequency you get from the two units will be different, and your system may not behave as you expect.

**-11 = HMC\_READ\_TIMEOUT** – The driver was unable to read the expected result from the hardware within the timeout period. Usually this is caused by disconnecting a cable or powering a device down. This allows your program to recover gracefully if communication with the hardware is lost. To change the timeouts, use `HMC_TimeoutsByNum()` or `HMC_TimeoutsByHandle()`.

**-12 = HMC\_BAD\_RFOUT\_CONN** – The legal connection values are 1 to connect to the front panel RF connector and 0 to disconnect the RF signal from the RF connector. Note that the unit does not have infinite isolation so a non-trivial signal may still be present even when “disconnected.”

**-17 = HMC\_SET\_REMOTE\_BEFORE\_ACCESSING** – Opening a unit reserves it for one process on the computer. However, the user still has control from the front panel. To give the computer control, and disable the front panel, call `HMC_RemoteAndLocalLockout()`. This will allow functions like `HMC_Frequency()`, `HMC_Power()`, and `HMC_RFOut()` to function normally. Use `HMC_GoToLocal()` to get front panel control back again.

The readback functions will generally work without forcing the unit into Remote control so computer based GUIs can show what's happening under front panel control.

### 2.2.2 **HMC\_MapSynths(void)**

`HMC_MapSynths()` gets a list of synthesizers from the USB driver and attempts to open each of them.

If a synthesizer can be opened, its identification information (description, serial number, etc.) is read and added to an internally maintained array, and `NumSynths` is incremented.

Each opened synthesizer is then closed to allow it to be opened by a different process.

If a synthesizer is powered up and properly attached to the computer but `HMC_MapSynths()` is not finding it, it is probably open in another process, such as the GUI Instrument Control.

Return value is `HMC_ERROR_CODES`.

Call this function if you add or remove a synthesizer, move a synthesizer from one USB connector to another, or power cycle a synthesizer.

If you are going to disconnect a synthesizer which you have called `HMC_Open()` on, you should call `HMC_Close()` first.

Synthesizers may be re-numbered by `HMC_MapSynths()`, even if the hardware configuration did not actually change.

### 2.2.3 **HMC\_NumSynths(void)**

`HMC_NumSynths()` returns the number of mapped synthesizers as an int.

If no synthesizers are mapped, the return value is 0.

#### 2.2.4 HMC\_GetIDNByNum(int UnitNum, const char\*\* IDN)

HMC\_GetIDNByNum() retrieves the “\*IDN?” string for a unit (physical T2000) as a null terminated ASCII string. (Standard 8 bit “C” string.) Since this function takes UnitNum (int) as an argument rather than a Handle (void\*), this function may be called before calling HMC\_Open().

The memory for IDN is allocated by the driver. Do not write over this value or attempt to free() it.

UnitNum is a 0 based index. Its maximum legal value is HMC\_NumSynths() – 1.

Return value is HMC\_ERROR\_CODES.

Sample usage:

```
/* Print out IDN strings */
/* You can substitute Serial, Description, FPGARev, Model, or Text for IDN */
/* in HMC_GetIDNByNum */
const char* IDN;
int UnitNum;
int NumUnits;
HMC_MapSynths();
NumUnits = HMC_NumSynths();
for (UnitNum = 0; UnitNum < NumUnits; ++UnitNum) {
    HMC_ERROR_CODES err = HMC_GetIDNByNum(UnitNum, &IDN);
    if (HMC_OK == err)
        printf("Unit #%d: IDN string = %s\n",
            UnitNum,
            SerialNumber);
    else
        printf("Unit #%d: Error %d trying to read IDN string\n",
            UnitNum,
            err);
}
```

See Also:

HMC\_GetSerialByNum()  
HMC\_GetDescriptionByNum()  
HMC\_GetFPGARevByNum()  
HMC\_GetModelByNum()  
HMC\_GetTextByNum() – User settable Text string for identifying unit

### 2.2.5 HMC\_GetIDNByHandle(void\* SynthHandle, const char\*\* IDN)

HMC\_GetIDNByHandle() retrieves the “\*IDN?” string for a unit (physical T2000) as a null terminated ASCII string. (Standard 8 bit “C” string.)

The memory for IDN is allocated by the driver. Do not write over this value or attempt to free() it.

SynthHandle is a void\* from HMC\_Open(). Do not modify or free() the memory pointed to by SynthHandle even after the unit is closed.

Return value is HMC\_ERROR\_CODES.

See Also:

HMC\_GetSerialByHandle()  
HMC\_GetDescriptionByHandle()  
HMC\_GetFPGARevByHandle()  
HMC\_GetModelByHandle()  
HMC\_GetTextByHandle() – User settable Text string for identifying unit

### 2.2.6 HMC\_Open(int UnitNum, void\*\* SynthHandle)

HMC\_Open() reserves a particular synthesizer for this process (so no other processes can interfere with it) and returns a “handle” (void\*) which can be used to refer to this synthesizer.

UnitNum is a 0 based index. Its maximum legal value is HMC\_NumSynths() – 1.

SynthHandle is a void\*. A NULL (zero) SynthHandle indicates an error. Do not write into the memory pointed to by SynthHandle or attempt to free() it.

Return value is HMC\_ERROR\_CODES.

Example:

```
int UnitNum = 0;
void* SynthHandle = (void*)0;
HMC_ERROR_CODES err = HMC_Open(UnitNum, &SynthHandle);
if (HMC_OK == err ||
    HMC_ALREADY_OPEN == err) {
    /* Looks good */
} else {
    /* Something wrong */
}
```

If you want to be able to write to the unit as well as read back from it, you will also need to call `HMC_RemoteAndLocalLockout()`.

`HMC_Open()` is a relatively slow operation. You may want to open your synthesizers once at the beginning of your program and leave them open.

If possible, you should call `HMC_Close()` on any synthesizer which you have opened before exiting your program. Units which are not closed (program crashed) may be left in an open state or stuck in local lockout. (Remove USB cable or power cycle to get local control back. Use the Windows Task Manager (Ctrl-Alt-Delete then Task Manager) to kill the process which is holding the file open if it does not exit normally.)

### 2.2.7 **HMC\_Close(void\* SynthHandle)**

`HMC_Close()` releases a synthesizer to allow it to be used by other processes.

`SynthHandle` is a `void*` from `HMC_Open()`. Do not modify or `free()` the memory pointed to by `SynthHandle` even after the unit is closed.

Return value is `HMC_ERROR_CODES`.

Using a `SynthHandle` from a closed unit may result in an error such as `HMC_NO_LONGER_VALID`. The `SynthHandle` is not nulled or otherwise corrupted in case there are multiple users of the handle, such as GUI code, which may not know that the unit has been closed until it starts getting errors.

`HMC_IsOpen()` can tell you whether a synthesizer is currently open (in this process) or not.

It is OK to close the same synthesizer multiple times; just ignore the error code.

### 2.2.8 **HMC\_IsOpen(const void\* SynthHandle)**

HMC\_IsOpen() returns true (non-zero) if this process has the synthesizer open or false (0) if the SynthHandle.

SynthHandle is a void\* from HMC\_Open().

### 2.2.9 **HMC\_RemoteAndLocalLockout(void\* SynthHandle)**

HMC\_RemoteAndLocalLockout() gives control over a unit to the computer and disables the front panel.

There is no mode where both the computer and the front panel can have control at the same time. To get front panel control back, use HMC\_GoToLocal() or disconnect the USB cable or power cycle.

SynthHandle is a void\* from HMC\_Open().

Return value is HMC\_ERROR\_CODES.

HMC\_RemoteAndLocalLockout() is much faster than HMC\_Open(). If you want to be able to switch back and forth between front panel and computer control, possibly from a GUI, HMC\_RemoteAndLocalLockout() and HMC\_GoToLocal() provide a reasonably efficient means to do so.

### 2.2.10 **HMC\_GoToLocal(void\* SynthHandle)**

HMC\_GoToLocal() restores front panel control which has been disabled by HMC\_RemoteAndLocalLockout() and disables writes from the computer.

SynthHandle is a void\* from HMC\_Open().

Return value is HMC\_ERROR\_CODES.

### 2.2.11 **HMC\_Frequency(void\* SynthHandle, double Frequency\_Hz)**

HMC\_Frequency() programs the output frequency of the unit. To allow this function to be well behaved across synthesizers with different resolution, the frequency is specified in Hz, regardless of the resolution of the synthesizer being programmed.

SynthHandle is a void\* from HMC\_Open().

Frequency\_Hz is a double with the value 1.0 corresponding to 1 Hz.

Return value is HMC\_ERROR\_CODES.

### 2.2.12 **HMC\_GetFrequency(void\* SynthHandle, double\* Frequency\_Hz)**

HMC\_GetFrequency() reads the output frequency of the unit. To allow this function to be well behaved across synthesizers with different resolution, the frequency is specified in Hz, regardless of the resolution of the synthesizer being programmed.

SynthHandle is a void\* from HMC\_Open().

Frequency\_Hz is a double with the value 1.0 corresponding to 1 Hz.

Return value is HMC\_ERROR\_CODES.

Call this as:

```
void* SynthHandle;  
HMC_ERROR_CODES err;  
double Freq_Hz;  
/* ... */  
err = HMC_GetFrequency(SynthHandle, &Freq_Hz);
```

### 2.2.13 **HMC\_Power(void\* SynthHandle, double Power\_dBm)**

HMC\_Power programs the output power (amplitude) of the unit in dBm into 50 Ohms.

SynthHandle is a void\* from HMC\_Open().

Power\_dBm is a double with maximum and minimum values that are frequency dependent. The resolution for the T2000 is 0.5 dB.

Return value is HMC\_ERROR\_CODES.

### 2.2.14 **HMC\_GetPower(void\* SynthHandle, double\* Power\_dBm)**

HMC\_GetPower reads the output power (amplitude) of the unit in dBm into 50 Ohms.

SynthHandle is a void\* from HMC\_Open().

Power\_dBm is a pointer to a double where the power value will be written.

Return value is HMC\_ERROR\_CODES.

### 2.2.15 **HMC\_RFOut(void\* SynthHandle, int Connection)**

HMC\_RFOut() enables or disables the RF output connection for the synthesizer.

SynthHandle is a void\* from HMC\_Open().

Connection is an int with the value 0 to disconnect and 1 to connect the front panel RF connector.

Return value is HMC\_ERROR\_CODES.

Note that the output does not have perfect isolation; a non-trivial signal may still be present even if the output connection is disabled.

### 2.2.16 **HMC\_GetRFOut(void\* SynthHandle, int\* Connection)**

HMC\_GetRFOut() reads the RF output connection state for the synthesizer.

SynthHandle is a void\* from HMC\_Open().

Connection is a pointer to the int for the state to be written into.

Return value is HMC\_ERROR\_CODES.

### 2.2.17 **HMC\_GetSerialByNum(int UnitNum, const char\*\* Serial)**

HMC\_GetSerialByNum() retrieves the serial number for a unit (physical T2000) as a null terminated ASCII string. (Standard 8 bit "C" string.) Since this function takes UnitNum (int) as an argument rather than a Handle (void\*), this function may be called before calling HMC\_Open().

The unit shows this serial number on its front panel display when it is first turned on to allow you to distinguish between multiple units in the same rack.

It is possible for two Hittite synthesizers to have the same serial number, but they will never have both the same serial number and the same description.

The memory for Serial is allocated by the driver at the time the synthesizer is mapped. Do not write over this value or attempt to free() it.

UnitNum is a 0 based index. Its maximum legal value is HMC\_NumSynths() – 1.

Return value is HMC\_ERROR\_CODES.

### 2.2.18 **HMC\_GetSerialByHandle(void\* SynthHandle, const char\*\* Serial)**

HMC\_GetSerialByHandle() retrieves the serial number for a unit as a null terminated ASCII string.

SynthHandle is a void\* from HMC\_Open().

The memory for Serial is allocated by the driver at the time the synthesizer is mapped. Do not write over this value or attempt to free() it.

Return value is HMC\_ERROR\_CODES.

### 2.2.19 **HMC\_GetDescriptionByNum(int UnitNum, const char\*\* Description)**

HMC\_GetDescriptionByNum() retrieves the description of the device as shown in the Windows Device Manager. This function can be called before HMC\_Open().

Example:

HMC-T2000 Signal Generator

The memory for Description is allocated by the driver at the time the synthesizer is mapped. Do not write over this value or attempt to free() it.

UnitNum is a 0 based index. Its maximum legal value is HMC\_NumSynths() – 1.

Return value is HMC\_ERROR\_CODES.

### 2.2.20 **HMC\_GetDescriptionByHandle(void\* SynthHandle, const char\*\* Description)**

HMC\_GetDescriptionByHandle() retrieves the description of the device as shown in the Windows Device Manager.

Example:

HMC-T2000 Signal Generator

The memory for Description is allocated by the driver at the time the synthesizer is mapped. Do not write over this value or attempt to free() it.

SynthHandle is a void\* from HMC\_Open().

Return value is HMC\_ERROR\_CODES.

### 2.2.21 **HMC\_GetFPGARevByNum(int UnitNum, const char\*\* FPGARev)**

HMC\_GetFPGARevByNum() retrieves the revision of the FPGA (Field Programmable Gate Array) firmware as a standard 8 bit character null terminated "C" string.

UnitNum is a 0 based index. Its maximum legal value is HMC\_NumSynths() – 1.

FPGARev is a string typically of the format "A.B", where A is the major revision and B is the minor revision. Do not write over FPGARev or attempt to free() it.

Return value is HMC\_ERROR\_CODES.

### 2.2.22 **HMC\_GetFPGARevByHandle(void\* SynthHandle, const char\*\* FPGARev)**

HMC\_GetFPGARevByHandle() retrieves the revision of the FPGA (Field Programmable Gate Array) firmware as a standard 8 bit character null terminated "C" string.

SynthHandle is a void\* from HMC\_Open().

FPGARev is a string typically of the format "A.B", where A is the major revision and B is the minor revision. Do not write over FPGARev or attempt to free() it.

Return value is HMC\_ERROR\_CODES.

### 2.2.23 **HMC\_GetModelByNum(int UnitNum, const char\*\* Model)**

HMC\_GetModelByNum() retrieves the model name of the unit as a standard 8 bit character null terminated "C" string.

UnitNum is a 0 based index. Its maximum legal value is HMC\_NumSynths() – 1.

Model is a string such as "HMC-T2000". Do not write over Model or attempt to free() it.

Return value is HMC\_ERROR\_CODES.

### 2.2.24 **HMC\_GetModelByHandle(void\* SynthHandle, const char\*\* Model)**

HMC\_GetModelByHandle() retrieves the model name of the unit as a standard 8 bit character null terminated "C" string.

SynthHandle is a void\* from HMC\_Open().

Model is a string such as "HMC-T2000". Do not write over Model or attempt to free() it.

Return value is HMC\_ERROR\_CODES.

### 2.2.25 **HMC\_TextByNum(int UnitNum, const char\* Text)**

HMC\_TextByNum() sets the Text name of the unit as a standard 8 bit character null terminated “C” string. This string can be used by the GUI in labels or pull down menus or anywhere else you want to identify this unit in a way that is meaningful to you.

This text string is not stored in the unit and will be lost when the process exits.

UnitNum is a 0 based index. Its maximum legal value is HMC\_NumSynths() – 1.

Text defaults to a string such as “HMC-T2000# 0012345”, but it can be changed to any value of your choosing. Examples of values include “LO 1”, “RF In”, the calibration sticker number, and the “Property of” tag number. The string is copied by the driver so it can be allocated on the stack or modified after this function is called.

Return value is HMC\_ERROR\_CODES.

### 2.2.26 **HMC\_TextByHandle(void\* SynthHandle, const char\* Text)**

HMC\_TextByHandle() sets the Text name of the unit as a standard 8 bit character null terminated “C” string. This string can be used by the GUI in labels or pull down menus or anywhere else you want to identify this unit in a way that is meaningful to you.

This text string is not stored in the unit and will be lost when the process exits.

SynthHandle is a void\* from HMC\_Open().

Text defaults to a string such as “HMC-T2000# 0012345”, but it can be changed to any value of your choosing. Examples of values include “LO 1”, “RF In”, the calibration sticker number, and the “Property of” tag number. The string is copied by the driver so it can be allocated on the stack or modified after this function is called.

Return value is HMC\_ERROR\_CODES.

### 2.2.27 **HMC\_GetTextByNum(int UnitNum, const char\*\* Text)**

HMC\_GetTextByNum() retrieves the Text name of the unit as a standard 8 bit character null terminated “C” string. This string can be used by the GUI in labels or pull down menus or anywhere else you want to identify this unit in a way that is meaningful to you.

Text defaults to a string such as “HMC-T2000# 0012345”, but it can be changed to any value of your choosing. Examples of values include “LO 1”, “RF In”, the calibration sticker number, and the “Property of” tag number. Please use HMC\_TextByNum() to set the value of the string rather than manipulating the pointer from this function directly.

Return value is HMC\_ERROR\_CODES.

### 2.2.28 **HMC\_GetTextByHandle(void\* SynthHandle, const char\*\* Text)**

HMC\_GetTextByHandle() retrieves the Text name of the unit as a standard 8 bit character null terminated "C" string. This string can be used by the GUI in labels or pull down menus or anywhere else you want to identify this unit in a way that is meaningful to you.

SynthHandle is a void\* from HMC\_Open().

Text defaults to a string such as "HMC-T2000# 0012345", but it can be changed to any value of your choosing. Examples of values include "LO 1", "RF In", the calibration sticker number, and the "Property of" tag number.

Return value is HMC\_ERROR\_CODES.

### 2.2.29 **HMC\_DisablePowerLimitCheck(void\* SynthHandle, int Disable)**

HMC\_DisablePowerLimitCheck() enables or disables the Power error checking for the synthesizer. This allows access to the typical (as opposed to specified and guaranteed) behavior of the unit. For the T2000, with the power limit check disabled, power can be programmed to +/-21.5 dBm, which is a wider range than any unit can reasonably be expected to cover. Power above +21.5 dBm will be clamped to +21.5 dBm; power below -21.5 dBm will be clamped to -21.5 dBm. Failing error codes will not be returned regardless.

Note: Typical behavior is not guaranteed. You may get 10 units which can all reach +xx.x dBm at yyy MHz and then get one that does not. The one that does not get there is not considered defective unless +xx.x dBm at yyy MHz is in the specified range of operation. Contact Hittite if you need units which are screened to guarantee performance in a normally unspecified region.

SynthHandle is a void\* from HMC\_Open().

Disable is an int with the value 1 to disable the frequency dependent error checking and the value 0 to re-enable error checking.

Return value is HMC\_ERROR\_CODES.

### 2.2.30 **HMC\_GetDisablePowerLimitCheck(void\* SynthHandle, int\* Disable)**

HMC\_GetDisablePowerLimitCheck() reads the Power error checking state for the synthesizer.

SynthHandle is a void\* from HMC\_Open().

Disable is a pointer to the int for the power limit check state to be written into.

Return value is HMC\_ERROR\_CODES.

### 2.2.31 **HMC\_TimeoutsByNum(int UnitNum, unsigned long ReadTimeout\_ms, unsigned long WriteTimeout\_ms)**

HMC\_TimeoutsByNum() sets the timeout for writing to and reading from the hardware.

The hardware should respond within a few milliseconds so the default settings of 0.1 second for write and 1 second for read is very conservative.

Set timeouts to 0 to disable.

Note: Setting timeouts to 0 will cause your program to hang in certain conditions, such as disconnecting a USB cable or power cycling a unit. Just re-attaching the cable or turning the power back on will generally not be sufficient to recover from the hang. You may need to use your debug environment or the Windows Task Manager to kill the process. Hardware may be left in a useless state, such as Local Lockout without the computer talking to it. Disconnect the USB cable, cycle power, or run a program which can open the device and call HMC\_GoToLocal() to get out of local lockout.

These values go to the USB driver and are not written to the synthesizer itself.

UnitNum is a 0 based index. Its maximum legal value is HMC\_NumSynths() – 1.

ReadTimeout\_ms and WriteTimeout\_ms are the timeouts for reading from and writing to the synthesizer.

Return value is HMC\_ERROR\_CODES.

### 2.2.32 **HMC\_GetTimeoutsByNum(int UnitNum, unsigned long\* ReadTimeout\_ms, unsigned long\* WriteTimeout\_ms)**

HMC\_GetTimeoutsByNum() reads the timeout for writing to and reading from the hardware from the driver.

UnitNum is a 0 based index. Its maximum legal value is HMC\_NumSynths() – 1.

ReadTimeout\_ms and WriteTimeout\_ms are pointers to unsigned longs which will get the values of the timeouts for reading from and writing to the synthesizer.

Return value is HMC\_ERROR\_CODES.

### 2.2.33 **HMC\_TimeoutsByHandle(void\* SynthHandle, unsigned long ReadTimeout\_ms, unsigned long WriteTimeout\_ms)**

HMC\_TimeoutsByHandle() sets the timeout for writing to and reading from the hardware.

The hardware should respond within a few milliseconds so the default settings of 0.1 second for write and 1 second for read is very conservative.

Set timeouts to 0 to disable. (Note: Setting timeouts to 0 will cause your program to hang in certain conditions, such as disconnecting a USB cable or power cycling a unit. Just re-attaching the cable or turning the power back on will generally not be sufficient to recover from the hang. You may need to use your debug environment or the Windows Task Manager to kill the process. Hardware may be left in a useless state, such as Local Lockout without the computer talking to it. Disconnect the USB cable, cycle power, or run a program which can open the device and call HMC\_GoToLocal() to get out of local lockout.)

These values go to the USB driver and are not written to the synthesizer itself.

SynthHandle is a void\* from HMC\_Open().

ReadTimeout\_ms and WriteTimeout\_ms are the timeouts for reading from and writing to the synthesizer.

Return value is HMC\_ERROR\_CODES.

### 2.2.34 **HMC\_GetTimeoutsByHandle(void\* SynthHandle, unsigned long\* ReadTimeout\_ms, unsigned long\* WriteTimeout\_ms)**

HMC\_GetTimeoutsByHandle() reads the timeout for writing to and reading from the hardware from the driver.

SynthHandle is a void\* from HMC\_Open().

ReadTimeout\_ms and WriteTimeout\_ms are pointers to unsigned longs which will get the values of the timeouts for reading from and writing to the synthesizer.

Return value is HMC\_ERROR\_CODES.

## 3 **Technical Support**

Please contact [applications@hittite.com](mailto:applications@hittite.com) or call (978) 250-3343 and request the HMC-T2000 technical support department. Hittite Microwave provides local direct support in many areas around the world. Please see the “Contact Us” page at [www.hittite.com](http://www.hittite.com).