



78M6612 M-API Library User Guide

**September 30, 2010
Rev. 1.00
UG_6612_036**

Maxim cannot assume responsibility for use of any circuitry other than circuitry entirely embodied in a Maxim product. No circuit patent licenses are implied. Maxim reserves the right to change the circuitry and specifications without notice at any time.

Maxim Integrated Products, 120 San Gabriel Drive, Sunnyvale, CA 94086 408-737-7600

© 2010 Maxim Integrated Products

Maxim is a registered trademark of Maxim Integrated Products.

Table of Contents

1	Introduction	5
1.1	Terminology	5
1.2	Library Measurement Equations	6
1.3	Library Parameters	7
1.3.1	Global Symbols	7
1.3.2	Formulae for Scaled Parameters	7
1.3.3	Example Calculation of Sensor Parameters	8
1.4	Reference Documentation	8
2	Firmware Partitions	9
3	Build Environment and Software/Firmware Configurations	10
3.1	Firmware Library and CE Memory Configurations	10
3.2	Program and RAM Memory	10
3.2.1	Program Memory	10
3.2.2	RAM/Data Memory	11
3.3	Build Configuration.....	11
3.3.1	Flash/Code and RAM Size	11
3.3.2	Flash/Code Space Assignment.....	12
3.3.3	Library Assignment.....	13
3.3.4	Module's Selection within a Library	14
4	MAPI Libraries.....	17
4.1	Library Initialization and Operation	17
4.1.1	MAPI_Init().....	17
4.1.2	MAPI_MeterRun()	17
4.1.3	MAPI_MinMaxSetGet()	17
4.2	Library Inputs and Outputs	18
4.2.1	MAPI_GetSetRegister().....	18
4.3	CE and MPU Control	23
4.3.1	MAPI_CEOn().....	23
4.3.2	MAPI_CEOff().....	23
4.3.3	MAPI_GetCEName()	23
4.3.4	MAPI_SoftReset()	23
4.3.5	MAPI_HardReset()	24
4.3.6	MAPI_UpdateMPU().....	24
4.3.7	MAPI_UpdateCE()	24
4.4	Calibration and Power Measurement	25
4.4.1	MAPI_CalSetGet()	25
4.4.2	MAPI_Calibrate()	27
4.4.3	MAPI_MeterStatus()	28
4.4.4	MAPI_TimeToAverage().....	30
4.4.5	MAPI_GetEnergyUsed()	30
4.4.6	MAPI_GetVoltageCurrent()	31
4.5	Zero Crossing and Relay Control	32
4.5.1	MAPI_RelayConfig().....	32
4.5.2	MAPI_RelayControl ()	33
4.5.3	MAPI_CloseCircuit_0X()	33
4.5.4	MAPI_OpenCircuit_0X()	33
4.6	Soft-Timers	34
4.6.1	MAPIstm_init()	34
4.6.2	MAPIstm_fn_start()	34
4.6.3	MAPIstm_run()	34
4.6.4	MAPIstm_wait()	34

4.7	DIO Configuration (<i>future release</i>).....	35
4.7.1	MAPI_DIOConfig()	35
4.7.2	MAPI_DIORW()	36
4.8	Flash Management	37
4.8.1	Memcpy_rx().....	37
4.8.2	Memcpy_xr().....	38
4.8.3	Memcpy_xx().....	38
4.9	Serial/RS232 Interface.....	39
4.9.1	MAPI_UARTInit().....	39
4.9.2	MAPI_UARTTx().....	39
4.9.3	MAPI_TxLen()	40
4.9.4	MAPI_UARTRx()	40
4.9.5	MAPI_RxLen()	40
5	Default Values	41
6	Contact Information.....	42
	Revision History.....	43

Figure

Figure 1: High Level Host/Firmware Interface Architecture	9
---	---

Tables

Table 1: CE Configuration.....	10
Table 2: Flash Memory Map	10
Table 3: Data RAM Shared by CE and MPU	11
Table 4: Flash and RAM Consumption by M-API and Sample Application	11
Table 5: Breakdown of M-API's Flash Consumption	11
Table 6: M-API Feature/Module Selections	15

1 Introduction

This document describes the Metrology Application Programming Interface (M-API) version 2.00 firmware libraries available from Teridian for use with the 78M6612 IC. These libraries are specifically designed for measurement and switch control of two (2) single-phase AC outlets (same phase).

The firmware delivery is a set of metrology libraries that configure and operate the measurement front end (e.g. MUX, ADC, CE, etc.) and provides simplified access to measurement output data such as Power, Voltage, Current, accumulated Energy and Line Frequency. All measurement calculations are computed by the MAPI library every accumulation interval and mapped to a dedicated block of registers reserved by the library.

Measurement data are made available to the user application via API calls. A demo application making use of the M-API library set and serial UART interface is included as part of the 78M6612 OMU Evaluation Kit.

Timer functions using the hardware RTC are also available and can be accessed directly by the application. APIs specific to RTC are not available in the current library. Contact a Teridian representative for more information on non-Volatile RTC operation (e.g. battery backup modes).

1.1 Terminology

The following terminology is used throughout this document:

- CREEP – Threshold value where measurement outputs are squelched to zero.
- IMAX – External RMS current corresponding to 250 mVpk at the current input of the 78M6612. It should be set $IMAX = (Vpk/\sqrt{2})/R_{SENSE}$.
- VARs – Reactive Power (Q)
- VAs – Apparent Power (S).
- Watts – Active Power (P).
- VMAX – External RMS voltage corresponding to 250 mVpk at the voltage input of the 78M6612 (VA, VB). It must be set high enough to account for over-voltages.
- NB – Narrowband values.
- WB – Wideband values.

1.2 Library Measurement Equations

The integrated Compute Engine (CE) accumulates the raw samples from the ADC and provides to the 80515 MPU the critical **atomic measurements* needed to derive all other data. This consists of RMS Voltage, Voltage Sag Status, and AC Line Frequency data as well as RMS Current, Active Power, and Reactive Power for each outlet. The MAPI library provides the application developer with two equation options for processing the atomic values before updating the libraries output data registers. One equation option is defined as “Narrowband” (**NB**) and the other is defined as “Wideband” (**WB**).

When using **NB** equations; RMS Voltage, Active Power, and Reactive Power data is provided by the CE and used to derive RMS Current, Apparent Power, and Power Factor in the MPU for each outlet. Harmonic content is not included in the reported current measurement.

$$\begin{aligned}
 * \text{Voltage } (V_{\text{RMS}}) &= \sqrt{\sum v(t)^2} \\
 \text{Current } (I_{\text{RMS}}) &= S/V_{\text{RMS}} \\
 * \text{Active Power } (P) &= \sum (i(t) * v(t)) \\
 * \text{Reactive Power } (Q) &= \sum (i(t) * v(t) \text{ shift } 90^\circ) \\
 \text{Apparent Power } (S) &= \sqrt{(P^2 + Q^2)} \\
 \text{Power Factor } (\text{PF}) &= P/S
 \end{aligned}$$

When using **WB** equations (recommended); RMS Voltage, RMS Current, and Active Power data provided by the CE is used to derive Reactive Power, Apparent Power, and Power Factor in the MPU for each outlet.

$$\begin{aligned}
 * \text{Voltage } (V_{\text{RMS}}) &= \sqrt{\sum v(t)^2} \\
 * \text{Current } (I_{\text{RMS}}) &= \sqrt{\sum i(t)^2} \\
 * \text{Active Power } (P) &= \sum (i(t) * v(t)) \\
 \text{Reactive Power } (Q) &= \sqrt{(S^2 - P^2)} \\
 \text{Apparent Power } (S) &= V_{\text{RMS}} * I_{\text{RMS}} \\
 \text{Power Factor } (\text{PF}) &= P/S
 \end{aligned}$$

1.3 Library Parameters

This section describes critical constants and variable parameters of the M-API library and their recommended usage.

1.3.1 Global Symbols

The following symbols are fixed constants for the M-API v2.00 firmware library:

Samples	: 1806
FS	: 3641
POWERSCALE	: 2.3439E-06
VRMS_MSCALE	: 9.6827E-05
IRMS_MSCALE	: 2.4207E-05

The following variables are unique to the sensor configuration and represent the real world values mapped to the upper range of the 78M6612 analog front end.

IMAX	: 52 Amps for 4mOhm shunt
VMAX	: 471.5 Volts

1.3.2 Formulae for Scaled Parameters

The measurement outputs (and respective alarm thresholds) for the M-API library are stored in a raw format to preserve native resolution of the computed measurements. When using API calls to access or fetch measurement data, the values are automatically scaled and converted according to the data types, below.

I _{rms}	= float(Val)* IRMS_MSCALE* IMAX*√(FS/Samples)
V _{rms}	= float(Val)* VRMS_MSCALE* VMAX*√(FS/Samples)
Watts	= float(Val)* POWERSCALE*IMAX*VMAX/1000
Frequency	= integer(Val)/100
Val	: library data

In some rare cases, when raw measurement data (Val) is directly accessed without going through the library, which is not recommended, scaling to and from real world or usable values must be done using the formulae above.

1.3.3 Example Calculation of Sensor Parameters

This example demonstrates the calculation of IMAX and VMAX for the default sensor configuration. For more information on sensor selection and configuration, refer to the *78M6612 Hardware Design Guidelines*.

IMAX Calculation:

$$\text{IMAX} = \text{Imax (pk)} / \sqrt{2} = \text{Imax (rms)}$$

$$\text{Max ADC input} = 250 \text{ mV} = \text{IMax (pk)} * \text{R shunt}$$

Example:

With a 4 MΩ current shunt, IMax (pk) = 62.5A

$$\Rightarrow \text{IMAX} = 52 \text{ Amps.}$$

VMAX Calculation:

$$\text{VMAX} = \text{Vmax (pk)} / \sqrt{2} = \text{Vmax (rms)}$$

$$\text{Max ADC input} = 250 \text{ mV} = \text{Vmax (pk)} * \text{Shunt R} / (\text{Series R} - \text{Shunt R})$$

Example:

With a Series R of 2 MΩ and a Shunt R of 750 ohms, VMax (pk) = 666.42 Volts.

$$\Rightarrow \text{VMAX} = 471.23 \text{ Volts}$$

1.4 Reference Documentation

- *78M6612 Data Sheet*
- *78M6612 Hardware Reference Manual*
- *78M6612 OMU Evaluation Board User Manual* (reference schematics)
- *78M6612 OMU Demo Application User Manual* (CLI)
- *78M6612 OMU Firmware Description Document* (register descriptions)

2 Firmware Partitions

The 78M6612 firmware provided by Teridian is partitioned into three main components:

- The Compute Engine (CE) firmware, although a separate set of source code, is a component of the M-API libraries responsible for precision Voltage, Current, Watts, and VARs measurements handled by a dedicated processor. The source code for the CE is not described in this document or made available for user modification.
- The 80515-based M-API firmware, in combination with the CE firmware, completes the M-API library set and provides all the necessary IC configuration, calibration sub-routines, scaling, data conversion, and timing control. This set of libraries is to be linked to the application firmware specific to the desired host interface.
- The Application firmware exercises the M-API library and manages the communication to the host controller/application. Teridian provides an example Serial Driver to be used as sample code as well as the application firmware that uses this Serial Driver to exercise the M-API library. Refer to the applicable Firmware Description Document for more information on the application firmware.

Figure 1 shows a high level partitioning of the firmware architecture. The black boxes indicate object code to be provided by Teridian. The white box indicates object and source code provided by Teridian.

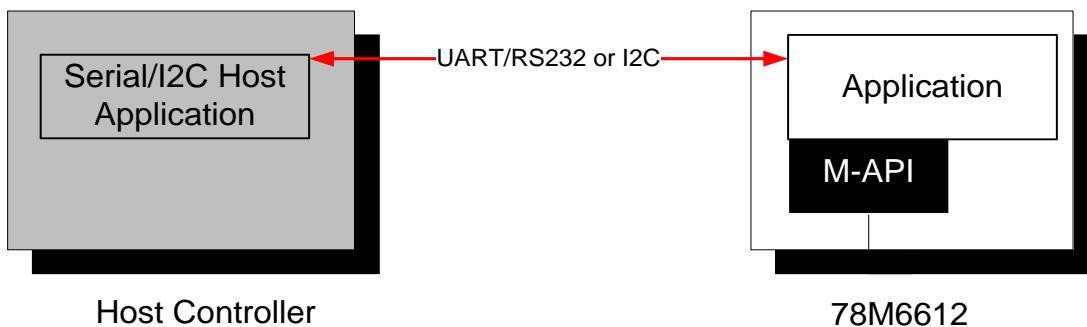


Figure 1: High Level Host/Firmware Interface Architecture

3 Build Environment and Software/Firmware Configurations

The Metrology Application Programming Interface (MAPI) is built using Keil Compiler version 8.02, although any Keil version 7.00 or higher is also compatible.

The MAPI is built using specific default configuration as listed below.

3.1 Firmware Library and CE Memory Configurations

Processing of ADC inputs is done by the CE. The MAPI libraries are built to link to the specific CE code according to the specific hardware configuration. The table below shows the different libraries as built with specific CE code to support different configurations:

Table 1: CE Configuration

Library Name	Default Accumulation Interval = 500ms
6612S22_WB_MAPI.lib	Built for Wide-Band.
6612S22_NB_MAPI.lib	Built for Narrow-Band.

3.2 Program and RAM Memory

The embedded 80515 MPU within the 78M6612 has separate program memory (32K bytes) and data memory (2K bytes).

3.2.1 Program Memory

The Flash program memory is non-volatile and is segmented into 512-byte pages. The last 2.5K (five 512-byte pages) of Flash space is reserved for CE data, CE code and default values used for Energy Calibration and Measurement. The code for the Compute Engine program resides in the MPU program memory (flash) and must begin on 1K-byte boundary.

Table 1 below shows the partitions of Flash memory.

Table 2: Flash Memory Map

Address (hex)	Type	Typical Usage	Wait States (at 5 MHz)	Size (bytes)
0000-75FF	Non-volatile	Program and any non-volatile data. Shared by the MAPI and application layer.	0	29.5KB
7600-773E	Non-volatile	Reserved for MAPI energy measurement constants (defaults).	0	< ~350B
777F-7FFF	Non-volatile	Reserved for CE code and CE data.	0	2KB

3.2.2 RAM/Data Memory

The 78M6612 has 2K bytes of Data Memory for exclusive use by the embedded 80515 MPU library and application layers. In addition, there are 512 bytes reserved for the Compute Engine.

Table 3: Data RAM Shared by CE and MPU

Address (hex)	Type	Typical Usage	Size (bytes)
0000-07FF	Volatile	MPU data XRAM,	2KB
1000-11FF	Volatile	CE data	512
2000-20FF	Volatile	Miscellaneous I/O RAM (configuration RAM)	256

3.3 Build Configuration

The accompanying example application source code has been set up to link to each library under specific hardware configuration. Each configuration is defined under a different 'Target'. However, for all 'Targets', specific area of Flash is reserved for specific function as described below.

3.3.1 Flash/Code and RAM Size

The attached zipped file (78M6612MAPIv200.zip) contains a sample Application code, of which source code is also included. The functions of this application are detailed in [Section 6](#).

This sample application is taking up Flash/RAM usage according to the table below. It was written to demonstrate the use of the MAPI libraries via two types of commands: the 'M' commands (as shown in [Section 6](#)) to specifically address the MAPI calls; and the 'CLI', aka Command Line Interface, commands for backward compatibility to support those who have evaluated, or are familiar with, Teridian OMU products in the past.

The following tables show RAM and Flash consumptions by this sample application. They also show a breakdown of Flash consumption by different features as implemented in the MAPI libraries.

Table 4: Flash and RAM Consumption by MAPI and Sample Application

Firmware Module	RAM Used	Flash Used	Note
6612S22_WB_MAPI.lib 6612S22_NB_MAPI.lib	~	26K	The RAM usage is overlay-able. See the breakdown of Flash usage in Table 5.
6612IVIV_50F0_v200WB.hex 6612IVIV_50F0_v200NB.hex	1.7K	6K + 26K(lib)	The app from this build uses every single API library call.

Table 5: Breakdown of M-API's Flash Consumption

Library:	Flash Used	Features
6612S22_WB_MAPI.lib 6612S22_NB_MAPI.lib		
Calibration	5.7K	Includes calibration for: Temperature, Current, Voltage, Watt and Phase.
Initialization, Default Settings	5.8K	Initializes Defaults, Metering Constants Tables, Registers, etc.
MAPI Interface (to App)	1.5K	Direct interface to the library from application.
Flash Manager	2K	Flash Read/Write management.

Metering	6K	Metrology functions (alarms, power factor, RMS, etc.)
Measurement	2K	Metrology computations
Serial Driver	1K	Serial 0/RS232 I/O
Timer	< 0.5K	Timer 0 driver.
Relay Management/Misc.	2K	Relay Control/On/Off, some math logic, IRQ.

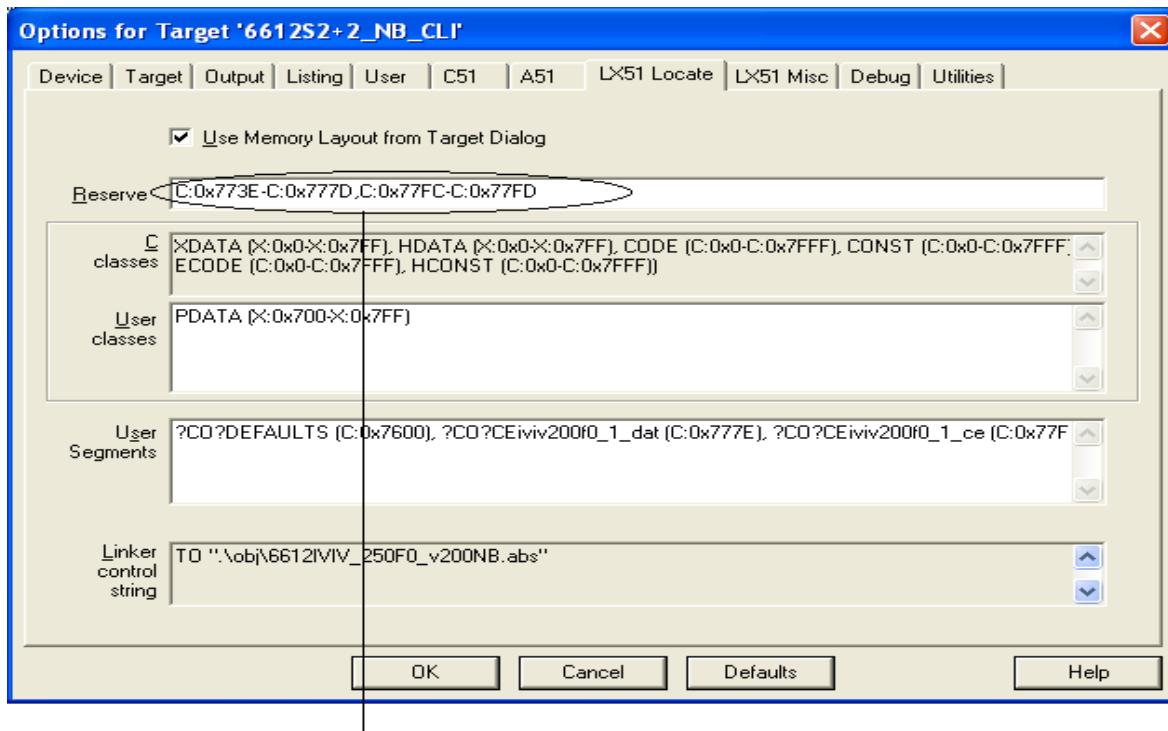
3.3.2 Flash/Code Space Assignment

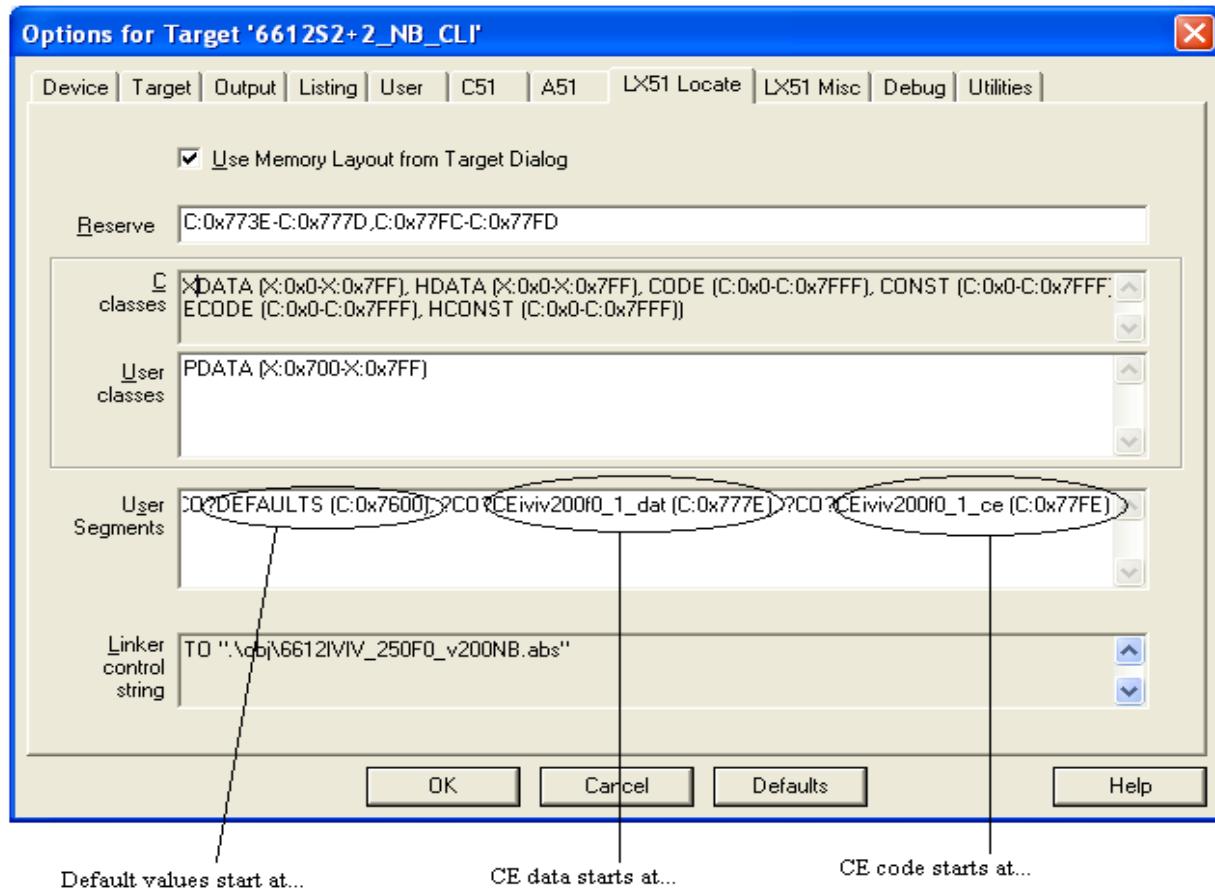
As shown in Table 1, in order to reserve specific regions of Flash for specific MAPI default data, CE code and CE data, the build file of the firmware application must be setup to specify where the modules will reside. It is necessary to keep images of CE code/data and default data isolated at the application's build file. The accompanying application code has setup these assignments as follows: Default data is at starting address 0x7600. CE Data is at starting address 0x777E. CE Code is at starting address 0x77FE.

This information is represented in the mapped file (.M51) as follows:

```
SEGMENTS (?CO?DEFAULTS(C:0X7600),
?CO?CEIVIV200F0_1_DAT(C:0X777E),
?CO?CEIVIV200F0_1_CE (C:0X77FE))
```

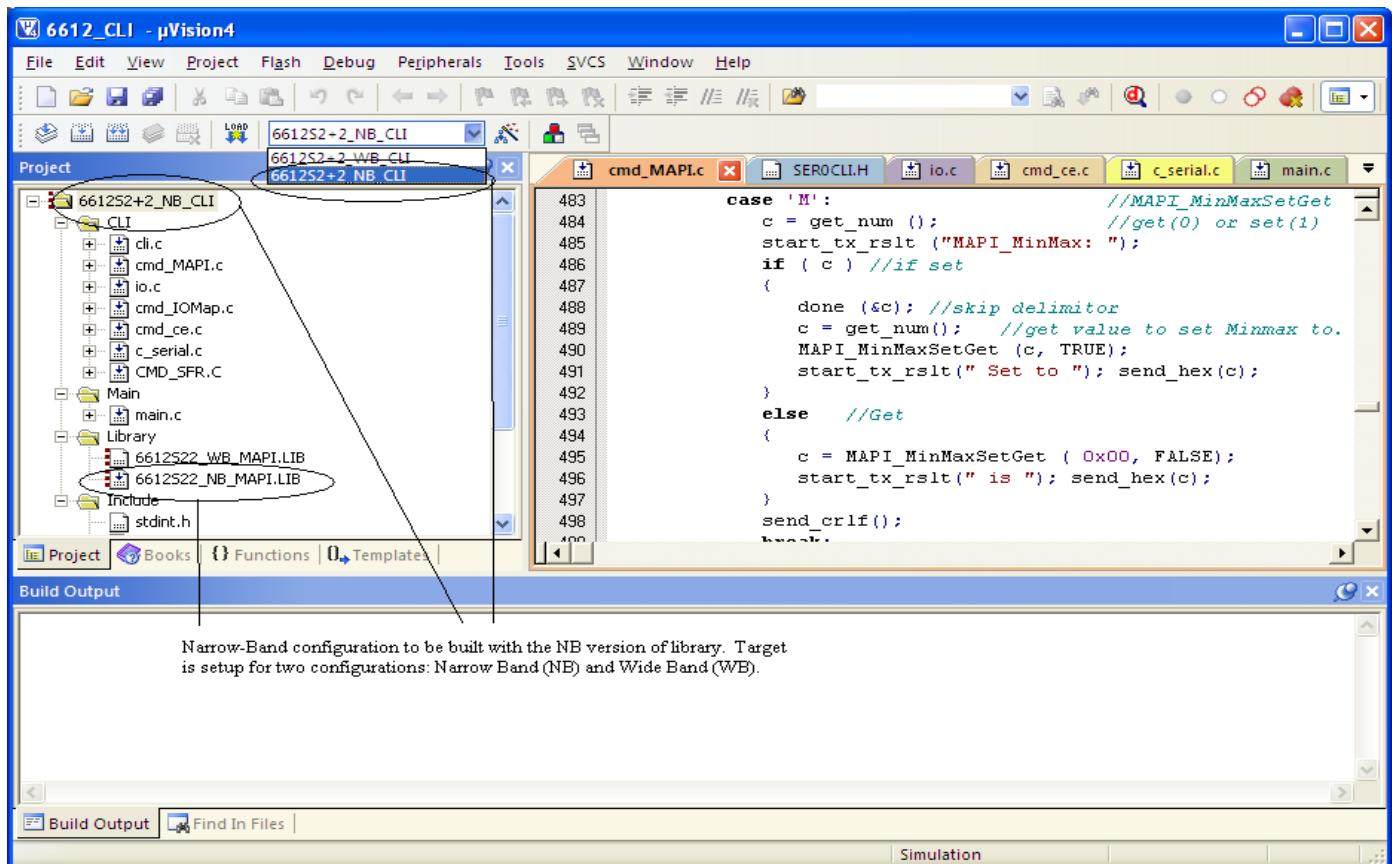
The following figures show how the setup is done.





3.3.3 Library Assignment

The firmware application source code, as shipped with this document, is setup such that its project file contains all the supported libraries as built for the specific CE code and data to support different configuration options. As shown in the project, the libraries built for Narrow-Band and Wide-Band are included in the project but only one library shall be selected to be built with a specific 'Target' at a time. The 'Target's as defined in the Keil project file for the sample application are shown below. When modify this project's targets, it is important that the correct library is selected:



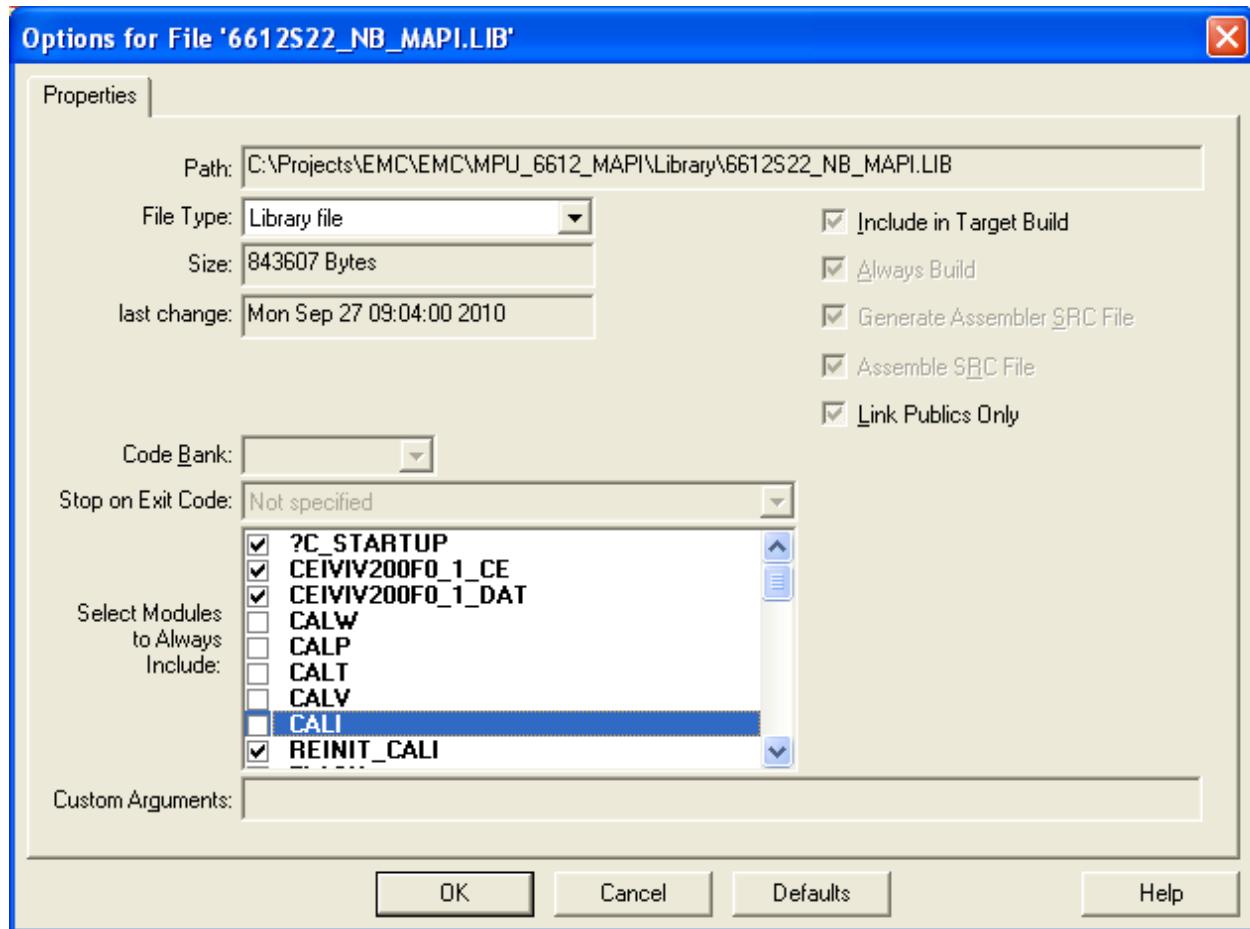
3.3.4 Module's Selection within a Library

The M-API libraries are organized into separate modules based on their functions so that code size and RAM size can be easily optimized during Keil's build/link time. Some of the modules are mandatory and must be included while others can be de-selected to save Flash space if the functions/features are not used. Typically, when an API is called directly, the Keil Linker would automatically include the module. However, in some cases, a needed module used by the library would not be included as it is only called within the library. In this case, the application would need to exclusively include the module. Table 6 shows the individual modules as built with the library and the selection criteria that an application is recommended to follow.

Table 6: MAPI Feature/Module Selections

Module	Optional (O)/Mandatory (M)	Description
?C_STARTUP	M	80515 Startup
CEIVIV200F0_1_CE CEIVIV200F0_1_DAT	M	CE data and code modules
CALW (Watt cal) CALP (Phase cal) CALT (Temperature cal) CALV (Volt cal) CALI (Current cal) REINIT_CALI (Current cal)	O O O O O M (if any CALW/CALI is M)	If calibration is required, these modules are needed based on the type of calibration is required. If Phase calibration is not needed, the application can exclusively de-select it by unchecking CALP.
FLASH LIBRARY	M	Required by Meter, Measure, and calibration modules
TSC ACCESS DEFAULTS SET_DEFAULTS	M	Required for all initialization of Energy measurement.
MAPI	M	Interface layer between the library and the app.
IOMAP	O	When a permanent change to any of the initialization/default constant values for MPU or CE is necessary.
MEASURE Rms CE6612 CE CE_ACCESS IO6612	M	Part of Energy Measurement function.
FREQ METER PEAK_ALERTS PHASE_ANGLE POWER PWRFACT RMS_METER VAH VARH WH MATH UPDATE IRQ	M	Part of Metering function.
SER0CLI	O	Serial UART/RS232 driver
TMR0 STM	M	Soft-Timer and Timer0 interrupt service routine.
TRIP	O	Relay control

The following figure shows the Keil IDE/UV4 IDE menu where the module can be selected or de-selected. To invoke this menu, left-click on a highlighted library and select 'Options for File '6612S22_NB_MAPI.LIB' or 6612S22_WB_MAPI.LIB':



4 MAPI Libraries

The following sections describe the available MAPI calls supported by these libraries.

4.1 Library Initialization and Operation

4.1.1 MAPI_Init()

Purpose	Initialize all critical variables, start the Compute Engine (CE) and its interrupts, start the MPU timer, setup all default values. The application must first call this API before any attempt to use other APIs.
Synopsis	Void MAPI_Init(void) ;
Parameters	None.
Return Codes	None.

4.1.2 MAPI_MeterRun()

Purpose	Run this function in foreground mode periodically to reset the watchdog, update all measurement outputs, and compare data to Min/Max thresholds. Once MAPI_Init is called, the CE will update all atomic measurements periodically. It is the task of the application to put this API into its main loop so that it can post-process the data just imported from the CE. If this API is called more than once within the same accumulation interval, no changes will be updated and a FALSE will be returned. Typically, the application layer will then call MAPI_MeterStatus() to check for any alarm conditions and call MAPI_GetSetRegister() to get updated measurement data.
Synopsis	Bool MAPI_MeterRun (void) ;
Parameters	None.
Return Codes	TRUE (1) – New data has been updated. FALSE(0) – No change from the last update run.

4.1.3 MAPI_MinMaxSetGet()

Purpose	Start or Stop MinMax control.
Synopsis	uint8_t MAPI_MinMaxSetGet(uint8_t SetData, uint8_t Op)
Parameters	Op: input parameter. Get = 0, Set = 1. SetData: input parameter. When Op = 1, set MinMax control register to SetData. MinMax control register is defined as: Bit1=Start_MinMax, Bit0=Reset_MinMax. It is customary that when Start_MinMax is set (Bit1 = 1), Reset_MinMax (Bit0 = 1) shall also be set. Reset_MinMax is auto-cleared by the library.
Return Codes	Current value of MinMax register.

4.2 Library Inputs and Outputs

4.2.1 MAPI_GetSetRegister()

Purpose Get or Set the value of a specific register location. An error will be returned if the address is out of range or within the restricted location. Care must be taken when calling this API to Set the value into a register. There are three types of registers: MPU, CE and I/O Hardware Control. The range of address indicates the type of registers as follows:

0x0000 – 0x03FF: MPU Address
 0x0400 – 0x07FF: CE Address
 0x2001 – 0x2001: Hardware I/O Control registers. Only address 2001 is accessible.

When calling this function, make sure the Reg.Address is OR'ed with the specific type of register (CE or MPU or I/O RAM) defined such as:

```
enum REG_TYPE {
    MPU_ADDRESS = 0x0000,
    CE_ADDRESS = 0x1000,
    RI_ADDRESS = 0x2000};
```

For example:

```
{ //example code to get different types of registers
    Struct Reg_t xdata Regt;
    //CE Address
    //shift 2 to accommodate 4-byte increment. Get CE add 100E
    Regt.Address = (unsigned long) (CE_ADDRESS | ( 0x01 << 2));
    MAPI_GetSetRegister (READ, &Regt);

    //Change CE contents at address 0x100E
    Regt.Value = 0x6050; //Change SAG counts
    MAPI_GetSetRegister (WRITE, &Regt);

    //MPU Address
    //Get MPU address 0x07 - Read-only
    Regt.Address = (unsigned long) (MPU_ADDRESS | (0x07));
    MAPI_GetSetRegisters (READ, &Regt);

    //RI address
    Regt_Address = (unsigned long) ((RI_ADDRESS) | (0x01));
    MAPI_GetSetRegisters (READ, &Regt);
    ....
}
```

Synopsis

```
enum MAPI_RC MAPI_GetSetRegister( bool Operation,
                                  struct Reg_t *Reg)
```

Where Reg_t is defined as:

```
Struct Reg_t
{
    Unsigned Integer      Address;
    Unsigned Long         Value;
    Unsigned Char          TypeSize;
    Unsigned Integer        ScaleFactor;
};
```

Parameters	Operation	Input parameter. WRITE(1) – Set Operation, READ(0) – Get Operation.
Struct _Reg_t		When Operation=READ, Output parameter. When Operation=WRITE, Input parameter.
Address		Four-byte address location. Address where its content will be extracted (Operation=READ) or stored (Operation = WRITE).
Value		Four-byte value to be stored or retrieved. If Content from Address to be extracted (Operation=READ) or stored (Operation = WRITE).
TypeSize		Type and Size of register's content. Read-only. This byte is defined as follows:

Bit 8	Bit 7...4	Bit 3...1
SIGNED: Register value is a signed value (1-negative, 0-positive).	TYPE: Register value is one of the following types. Note this is a read-only field used to determine the type of the Register content that is returned (see mapi_t.h): INTEGER 0x00 FLOAT 0x10 IRMS_M 0x40 VRMS_M 0x50 CONTROL 0x60 CONFIG 0x70 POWER 0x18 ENERGY 0x28 FREQ 0x38 COUNT 0x48 TIMER 0x58 STATUS 0x68 STRING 0x78	SIZE: Indicates storage size (in bytes) of register where: 1 – Char, 1-byte 2 - Integer, 2-byte 4 - Word, 4-byte
Typically used for power factor, phase adjust.		

ScaleFactor	Output parameter. This byte indicates presentation format of the register content defined as follows: 10 – Tenth unit factor. 100 – Hundredth unit factor. 1000 – Thousandth unit factor.
Return Codes	MAPI_OK – successful operation. MAPI_RESTRICTED – specified address is restricted.

4.2.1.1 Auto-Scaling

When retrieving measurement data using the MAPI_GetSetRegister() call, the returned data value is automatically converted to usable data according to the TypeSize of the register location. The ScaleFactor is for information only.

When writing alarm thresholds to library input registers, usable data values are automatically converted to raw values according to the TypeSize of the target register address.

4.2.1.2 Output Data Address Locations

The following output data is updated once per accumulation interval and available via the MAPI_GetSetRegister() call.

MPU Address		Name	LSB	Type	Description
NB	WB				
00	20	Delta Temperature	0.1 °C	FLOAT	Temperature difference from 22 °C.
01	21	Line Frequency	0.01 Hz	FREQ	Line Frequency.
02	22	Alarm Status (common)		STATUS	See Section 4.4.3 – MAPI_MeterStatus() for details.
03	23	OverCurrent Event Count	1	INTEGER	Number of event counts where value exceeded over current alarm threshold.
04	24	Under Voltage Event Count	1	INTEGER	Number of event counts where value exceeded under voltage alarm threshold.
05	25	Over Voltage Event Count	1	INTEGER	Number of event counts where value exceeded over voltage alarm threshold.
06	26	Volts	mVrms	VRMS_M	AC Line Voltage (RMS)

MPU Address				Name	LSB	Type	Description
Outlet 1		Outlet 2					
NB	WB	NB	WB				
07	27	47	67	Active Power	mW	POWER	Active Power (per accum interval).
08	28	48	68	Energy	mWh	ENERGY	Accumulated Energy.
09	29	49	69	Cost	mUnits	FLOAT	Accumulated Cost.
0A	2A	4A	6A	Current	mArms	IRMS_M	RMS Current.
0B	2B	4B	6B	Reactive Power	mW	POWER	Reactive Power (per accum interval).
0C	2C	4C	6C	Apparent Power	mW	POWER	Apparent Power (per accum interval).
0D	2D	4D	6D	Power Factor	–	FLOAT	Power factor. (output will be between -1.00 and 1.00)

4.2.1.3 MPU Library Inputs

The following inputs can be modified (and saved to Flash) using API calls. For more information on register descriptions, refer to the 6612_OMU Firmware Description Document.

Category	Name	LSB	MPU Address		Description
Common Alarm Threshold	Temperature		D0 D1		Min Temperature Alarm Threshold Max Temperature Alarm Threshold
	Frequency		D2 D3		Min Frequency Alarm Threshold Max Frequency Alarm Threshold
	Voltage (A)		D4 D5 D6		SAG (A) Voltage Alarm Threshold Min Voltage (A) Alarm Threshold Max Voltage (A) Alarm Threshold
Outlet Specific Alarm Thresholds			NB WB		
	Current - Outlet 1	D8	D9		Max Current Alarm Threshold
	Power Factor - Outlet 1	DA DB	DC DD		Power Factor Alarm - Threshold Power Factor Alarm + Threshold
	Current - Outlet 2	DE	DF		Max Current Alarm Threshold
	Power Factor - Outlet 2	E0 E1	E2 E3		Power Factor Alarm - Threshold Power Factor Alarm + Threshold
	Total Current	E4	E5		Max Current Alarm Threshold
Alarm Mask	Alarm Mask for Status Registers		E6		32-BIT alarm mask (0=masked)
Alarm Mask	Alarm Mask for Alarm DIO4		E7		32-BIT alarm mask (0=masked)
Sensor Scaling	Voltage - V(A) Current -Outlet 1 Current - Outlet 2		A0 A2 A4		VMAX A IMAX Outlet 1 (IA) IMAX Outlet 2 (IB)
Cost Factor	Cost		AA AB		Cost per KWh Cost Unit string
Relay Control	Relay Configuration		AC AD AE AF		Polarity, Latch type Sequence Delay Energize Delay De-energize Delay
Calibration	Calibration Configuration Parameters		BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF		Tolerance on Phase Calibration Calibration Type Calibration Voltage (Target) Calibration Current (Target) Calibration Phase (Target) Tolerance on Voltage Calibration Tolerance on Current Calibration Average Count for Voltage Average Count for Current Max Iterations for Voltage Max Iterations for Current Tolerance on Watts Calibration Average Count for Watts Max Iterations for Watts Calibration WRATE Calibration Temperature Calibration Wattage (Target)
Creep Threshold	Current -Outlet 1 Current - Outlet 2		A1 A3		Imin(IA) - "creep" or squelch level Imin(IB) - "creep" or squelch level

4.2.1.4 CE Library Inputs

The following inputs can only be modified (and saved to Flash) using API calls. For more information on register descriptions, refer to the 6612 OMU Firmware Description Document.

CATEGORY	CE Address	Description
Calibration	08-09	Calibration Gain IA-IB (Outlet 1-2)
	0A-0B	Calibration Gain VA, VB
Phase Adjust	0C-0D	Phase Adjust IA - IB
CE Configuration	0E	CE State
Pulse Rate	0F	Wrate
Quantization Corrections	12-13	Quantization offset Watt 1 to Watt 2
	14-15	Quantization offset VAR A – VAR B
	16-17	Quantization offset IA – IB
Temperature Compensation	19	Temperature Gain Adjust
SAG Threshold	10	SAG Threshold on kVar
	11	SAG Threshold
More Temperature Compensation	A6	Temperature Nominal
	A7	Degree Scale
	A8	ppm / °c
	A9	ppm / °c ²

4.3 CE and MPU Control

4.3.1 MAPI_CEOn()

Purpose Turn CE on. After calling `MAPI_Init()`, CE is automatically turned on. This API is provided as a convenience for the Application level to control the CE, especially when reading/writing to Flash is necessary (writing to Flash is not allowed when CE is on).

Synopsis `void MAPI_CEOn (void);`

Parameters None.

Return Codes None.

4.3.2 MAPI_CEOff()

Purpose Turn CE off. When writing to flash, the CE must be turned off. This API is necessary with `MAPI_UpdateCE`, `MAPI_UpdateMPU` or `memcpy_rx` calls.

Synopsis `void MAPI_CEOff (void);`

Parameters None.

Return Codes None.

4.3.3 MAPI_GetCEName()

Purpose Get the CE name embedded as part of the CE data. The CE name size is 24 bytes and these exact 24 bytes will be returned in the XDATA string pointed to by `ptr`.

Synopsis `void MAPI_GetCEName (unsigned char xdata *ptr);`

Parameters `ptr` Output parameter.
The CE name will be returned at the `xdata` location pointed to by `ptr`. For version 2.00 Library release, the string will look as follows:
`CE6612_OMU_S2+2_A01_V1_4.`

Return Codes None.

4.3.4 MAPI_SoftReset()

Purpose Reset the MPU, this includes stopping the CE, and setting all registers to hardware Power-On Reset values.

Synopsis `void MAPI_SoftReset (Void);`

Parameters None.

Return Codes None.

4.3.5 MAPI_HardReset()

Purpose Reset the MPU, this includes stopping the CE, and setting all registers to hardware Power-On Reset values.

Synopsis `void MAPI_HardReset (Void);`

Parameters None.

Return Codes None.

4.3.6 MAPI_UpdateMPU()

Purpose Update the MPU contents permanently into Flash. The MPU measurement input and calibration default values are stored in Flash. During power up, its content is copied to XRAM to be used as a working copy. The application can change some of the registers' values using the `MAPI_SetGetRegister()`. This change only takes effect in the XRAM copy of the data. To permanently save the new data into Flash, the application must exclusively call this function to perform the permanent save. This function is not to, and should not, be called too often as Flash Write does have a life-expectancy. It is typically used after the part is calibrated successfully to save the coefficient values. See the OMU_MAPICLI.uproj for sample usage of this API. In order for this function to perform properly, the CE must be turned off by calling `MAPI_CEOFF()`.

Synopsis `Bool MAPI_UpdateMPU (void);`

Parameters none

Return Codes TRUE – Successful write of MPU data to Flash.
 FALSE – Write was not successful; perhaps CE is still running.

4.3.7 MAPI_UpdateCE()

Purpose Update the CE Data contents permanently into Flash. The CE Data Image is programmed into Flash, starting at address 0x775E. During power up, its content is copied to XRAM to be used as a working copy. Though it is NOT recommended to change any CE Data, occasionally such needs arise, such as changing the IMAX, VMAX or WRATE values. The application can change some of the registers' values using the `MAPI_SetGetRegister()`. This change only takes effect in the XRAM copy of the data. To permanently save the data into Flash, the application must exclusively call this function to perform the permanent save into its specific location of Flash. This function is not, and should not, be called too often as Flash Write does have a life-expectancy. See the OMU_MAPICLI.uproj for sample usage of this API. In order for this function to perform properly, the CE must be turned off by calling `MAPI_CEOFF()`.

Synopsis `Bool MAPI_UpdateCE (void);`

Parameters None.

Return Codes TRUE – Successful write of MPU data to Flash.
 FALSE – Write was not successful; perhaps CE is still running.

4.4 Calibration and Power Measurement

4.4.1 MAPI_CalSetGet()

Purpose Set or Get Calibration referenced and tolerance parameters as specified in the `MPU_CParms_t` structure. New values are kept in RAM only. When all calibration data is setup and calibrated correctly, it shall be kept and recorded, permanently in Flash by calling `MAPI_UpdateCE()` and `MAPI_UpdateMPU()`.

A typical calibration of the part proceeds as follows:

1. Call `MAPI_CalSetGet (FALSE, ...)` to get current calibration data.
2. If necessary, call `MAPI_SetGetCal (TRUE, ...)` to set new calibration data (tolerance values, referenced values, etc.).
3. Call `MAPI_Calibrate()` to start the calibration. If calibration passes, continue to step 4. If calibration fails, repeat step 1.
4. Call `MAPI_UpdateCE()` and `MAPI_UpdateMPU()` to permanently update the new values in Flash.
5. Call `MAPI_CalSetGet (FALSE, ...)` to make sure the new data is written, preserved and correct.

Synopsis

```
Bool MAPI_CalSetGet(IN unsigned char SetData,
                     Struct MPU_CParms_t *MAPI_CParams);

Struct MPU_CParms_t
{
    uint8_t    C_Tcal;      // Type calibration. Read-Only.
    float     C_Wcal;      // Wattage calibration value (Watts).
    float     C_Vcal;      // Voltage calibration value (Vrms).
    float     C_Ical;      // Current calibration value (Arms).
    int16_t   C_Pcal;      // Phase calibration value (Degrees).

    float     C_Wtolerance; // Watts Tolerance (Watts).
    float     C_Vtolerance; // Voltage Tolerance (Vrms).
    float     C_Itolerance; // Current Tolerance (Arms).
    float     C_Ptolerance; // Phase Tolerance (degrees).

    uint8_t   C_Vavg_cnt;   // Voltage Average count.
    uint8_t   C_Iavg_cnt;   // Current Average count.
    uint8_t   C_Wavg_cnt;   // Watts Average count.

    uint16_t  C_Vmax_cnt;   // Voltage Max count.
    uint16_t  C_Imax_cnt;   // Current Max count.
    uint16_t  C_Wmax_cnt;   // Watts Max count.

    uint16_t  C_Wrate_cal; // Wrate during calibration (~.32Kh). Read-Only
    uint16_t  C_Tempcal;   // Calibration temperature (0.1 degree).
};
```

Parameters	SetData	Input parameter. TRUE(1) – Set calibration data as specified in <code>MPU_CParams_t</code> . FALSE(0) – Get current calibration data and return values in <code>MPU_CParams_t</code> .
MAPI_CParams:		
C_Tcal		Input parameter – read-only. Calibration type. None(0x00).
C_Wcal		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Referenced Wattage calibration value (in Watts). Default = 120W.
C_Vcal		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Referenced Voltage calibration value (Vrms) . Default = 120V.
C_Ical		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Referenced Current calibration value (Arms). Default = 1A.
C_Pcal		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Referenced Phase calibration value (in 0.1 C Degrees). Default = 0 degree.
C_Wtolerance		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE). Watts Tolerance (Watts). Default = 0.01W or 10mW.
C_Vtolerance		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Voltage Tolerance (Vrms). Default = 10mV.
C_Itolerance		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Current Tolerance (Arms). Default = 10mA.
C_Ptolerance		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Phase Tolerance (degrees). Default = 0.1 Degree.
C_Vavg_cnt		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Voltage Average count. Default = 3.
C_Iavg_cnt		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Current Average count. Default = 3.
C_Wavg_cnt		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Watts Average count. Default = 3.
C_Vmax_cnt		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Maximum number of voltage reads to test for pass/fail result. Default = 10.
C_Imax_cnt		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Maximum number of current reads to test for pass/fail result. Default = 10.
C_Wmax_cnt		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE) Maximum number of Watts reads to test for pass/fail result. Default = 10.
C_Wrate_cal		Output parameter. Wrate use during calibration (~.32kh). Read-only.
C_Tempcal		Output parameter (SetData = FALSE), Input parameter (SetData = TRUE). Calibration temperature. In 0.1 degree unit.
Return Codes	MAPI_OK	//Calibration succeeded.
	MAPI_ERROR	//Calibration failed. //For more detailed descriptions of the failure, call <code>MAPI_CALStatus()</code> .

4.4.2 MAPI_Calibrate()

Purpose Calibrate the part using referenced meter values and tolerance values as specified in the `MPU_CParms_t` data structure (see the [MAPI_CalSetGet](#) API for more information).

A typical application shall use this API as follows:

1. Call `MAPI_CalSetGet (FALSE, ...)` to get current calibration data.
2. If necessary, call `MAPI_CalGetSet (TRUE, ...)` to set new calibration data (tolerance values, referenced values, average count, maximum count, etc.).
3. Call `MAPI_Calibrate()` to start the calibration.
4. Call `MAPI_UpdateCE()` and `MAPI_UpdateMPU()` to permanently update the new values in Flash.
5. Call `MAPI_CalSetGet (FALSE, ...)` to make sure the new data is written/preserved.

It is up to the application level to select the CAL_TYPE appropriately. It is recommended that temperature calibration (CTYPE = C_TEMP) shall always be part of CAL_TYPE; thus, C_TEMP shall always be OR'ing with c_type. When this API is called with multiple CAL_TYPES, calibration will be done in the following order:

Temperature,
Phase,
Voltage,
Current,
Wattage.

Make sure the referenced values are setup correctly, using `MAP_CalSetGet()`; otherwise, calibration will not pass. See [Section 4](#) for default values.

When the return code is not MAPI_OK, call [MAPI_CalStatus\(\)](#) to get detailed descriptions of the calibration error(s). If an error occurs during this call, the part is NOT calibrated. It is the task of the application to call this API again if recalibration of the part is necessary.

Synopsis `MAPI_RC MAPI_Calibrate(IN enum CAL_TYPE c_type,
 IN Unsigned Char Outlets);`

Parameters `c_type` Input parameter.
The following types are acceptable:

```
Enum CAL_TYPE {
    C_WATT      = 0x01 - Calibrate Wattage.
    C_VOLT      = 0x02 - Calibrate Voltage.
    C_CURRENT   = 0x04 - Calibrate Current.
    C_PHASE     = 0x08 - Calibrate Phase Adjust.
    C_TEMP      = 0x10 - Calibrate Temperature.
}
```

Calibration source. The values can be OR'ing. For example: C_WATT | C_VOLT indicate calibration to be done on Voltage first, then Wattage; C_TEMP | C_VOLT indicate calibration to be done on Temperature then Voltage.

	Outlets	Input parameter Bit representations of outlet # to be calibrated. For example: 0x03 represents outlet #1 and 2 to be calibrated.
Return Codes	MAPI_OK MAPI_ERROR	//Calibration passed. //Calibration failed. Call <code>MAPI_CalStatus()</code> for specifics.

4.4.3 **MAPI_MeterStatus()**

Purpose	Run this function after a failed calibration or periodically to detect any error/warning. Any non-zero value returned indicates some failure/warning has occurred. MPU Output Threshold levels and mask settings for the alarms can be read, modified, and saved using the <code>MAPI_GetSetRegister()</code> API call.	
Synopsis	<code>void MAPI_MeterStatus (unsigned long Status);</code>	
Parameters	Status Output parameter. A 32-bit word status that indicates statuses as follows:	
	Min Temperature exceeded	Bit 0 = 1 //0000 0001
	Max Temperature exceeded	Bit 1 = 1 //0000 0002
	Min Frequency exceeded	Bit 2 = 1 //0000 0004
	Max Frequency exceeded	Bit 3 = 1 //0000 0008
	SAG A detected	Bit 4 = 1 //0000 0010
	Under Min VA on A	Bit 5 = 1 //0000 0020
	Over Max Voltage on A	Bit 6 = 1 //0000 0040
	Over Max I on A – narrow Band	Bit 7 = 1 //0000 0080
	Over Max I on A – wide Band	Bit 8 = 1 //0000 0100
	Min Power Factor exceeded on A -NB	Bit 9 = 1 //0000 0200
	Max Power Factor exceeded on A-NB	Bit 10 = 1 //0000 0400
	Min Power Factor exceeded on A -WB	Bit 11 = 1 //0000 0800
	Max Power Factor exceeded on A-WB	Bit 12 = 1 //0000 1000
	Max Current exceeded on B - NB	Bit 13 = 1 //0000 2000
	Max Current exceeded on B - WB	Bit 14 = 1 //0000 4000
	Min Power Factor exceeded on B -NB	Bit 15 = 1 //0000 8000
	Max Power Factor exceeded on B-NB	Bit 16 = 1 //0001 0000
	Min Power Factor exceeded on B -WB	Bit 17 = 1 //0002 0000
	Max Power Factor exceeded on B-WB	Bit 18 = 1 //0004 0000
	Max Current Total exceeded - NB	Bit 19 = 1 //0008 0000
	Max Current Total exceeded - WB	Bit 20 = 1 //0010 0000
	Creep on A	Bit 21 = 1 //0020 0000
	Creep on B	Bit 22 = 1 //0040 0000
	_FAULT – Neutral/Line reversal	Bit 23 = 1 //0080 0000
	Power Register Not Accurate	Bit 27 = 1 //0800 0000
Return Codes	None.	

4.4.4 MAPI_TimeToAverage()

Purpose	Set/Get the time increment (in seconds) in which the power (in Watts) is to be averaged. If a Time Interval has been set previously, it will be reset from this point forward and the accumulator's value will also be reset. Default value will be 10 seconds if this API is not called or if it is called with TimeInterval = 0. This time is global time that is applicable to both channels.	
Synopsis	<pre>Void MAPI_TimeToAverage (IN Bool SetTIME, IN unsigned int TimeInterval);</pre>	
Parameters	SetTIME	Input parameter. TRUE (1) will set the time to TimeInterval seconds. FALSE(0) will return the current Time Increment Query to TimeInterval.
	TimeInterval	Input parameter (SetTIME = TRUE), Output parameter (SetTIME = FALSE). Time in units of seconds, used to average the Watts used.
Return Codes	None.	

4.4.5 MAPI_GetEnergyUsed()

Purpose	Get the energy used per time increment query. This API will return the current values as recorded according to Index and the Time Increment Interval.	
Synopsis	<pre>Void MAPI_GetEnergyUsed(OUT struct E_Query_t *EQuery); Where E_Query_t is defined as: struct E_Query_t { Unsigned char Chan; //Channel/Outlet number float mWattsUsed; float WattHrsUsed; Unsigned char Index; Unsigned int cLineFreq;</pre>	
Parameters	Chan	Input parameter. Channel/Outlet number to get the energy information from.
	mWattsUsed	Output parameter. Instantaneous Power used in milli-Watts.
	WattHrsUsed	Output parameter. WattHour used over the past hour as pointed to by Index.
	Index	Output parameter. Number of hours past since power up. This value will reset to 0 (wrap-around) once the maximum is reached, i.e. 255 hours.
	cLineFreq	Output: Current line frequency value in ##.##Hz format.
Return Codes	None.	

4.4.6 MAPI_GetVoltageCurrent()

Purpose Get the current values of Voltage and Current . This function can be useful for determining referenced voltage and current values for calibration. Follow the steps below for a sample usage of this API:

1. Call `MAPI_CalSetGet()` to get default calibration values.
2. Call `MAPI_GetVoltageCurrent()` to get current Voltage and Current.
3. Set new parameters with call to `MAPI_CalSetGet()`.
4. Call `MAPI_Calibrate()` to start the calibration.
5. As a check call `MAPI_GetVoltageCurrent()` to confirm new calibrated values.

Synopsis `void MAPI_GetVoltageCurrent(OUT struct VIT_t *VIT_Params);`

Where `VIT_t` is defined as:

```
struct VIT_t
{
    Unsigned char Channel;
    float Current_I;
    float Current_V;
};
```

Parameters Channel Input parameter.
Channel/outlet number (1 or 2) where present Current value will be read.

 Current_I Output parameter.
Present value of Current (in mAmps) for the specified outlet.

 Current_V Output parameter
Present value of Voltage A (in mVolts).

Return Codes None.

4.5 Zero Crossing and Relay Control

4.5.1 MAPI_RelayConfig()

Purpose Read/Write relay configuration values. Relay configuration is applicable to all channels/outlets.

Synopsis void MAPI_RelayControl(IN Bool Operation, OUT/IN struct Relay_Config_t Relay_Config);

Where Relay_Config_t is defined as:

```
struct Relay_Config_t
{
    Unsigned Char      Relay_CTL;
    Unsigned Integer   SeqDlyTime;
    Unsigned Integer   DeEnergizedTime;
    Unsigned Integer   EnergizedTime;
    Unsigned Char      InvertPNonLatch;
};
```

Parameters Operation Input parameter.

TRUE (1) = set operation. FALSE (0) = get operation,
Relay_Config will all be output parameter.

When Operation = TRUE:

Relay_CTL	Output parameter. This parameter is output only. It is set via MAPI_RelayControl().
SeqDlyTime	Input parameter. Set the new Sequence Delay Time in 10ms units. Delay time is time between turning Relays ON (close circuit). Default value is 10 → 100ms.
DeEnergizedTime	Input parameter. Time in msec + 1msecond to delay after open circuit.
EnergizedTime	Input parameter. Time in msec + 1msecond to delay after close circuit.
InvertPolarity	Input parameter. Bit 1 = Polarity (0 = non-inverted). Bit 0 = Latch type (0 = non-latched).

Notes:

Sequence Delay Time will be used between turning relays ON.

Energized Delay Time is used as the delay time to wait
immediately after a zero-crossing before close circuit.

DeEnergized Delay Time is used as delay time to wait
immediately after a zero-crossing before open circuit.

Return Codes None.

4.5.2 MAPI_RelayControl ()

Purpose	Turning Relay ON/OFF (close/open circuit) on all channels.		
Synopsis	<code>void MAPI_RelayControl(IN unsigned char Channels);</code>		
Parameters	<table border="0"> <tr> <td>Channels</td> <td>Input parameter. Each bit indicates turning ON(1) or OFF(0). The bit's position represents the specific channel/outlet number. For example:</td> </tr> </table>	Channels	Input parameter. Each bit indicates turning ON(1) or OFF(0). The bit's position represents the specific channel/outlet number. For example:
Channels	Input parameter. Each bit indicates turning ON(1) or OFF(0). The bit's position represents the specific channel/outlet number. For example:		
	0x01h – Turn relay ON on channel 1/A. Turn relays OFF on channel 2/B.		
	0x02h – Turn relay ON on channel 2/B. Turn relays OFF on channel 1/A.		
	Both channels can be turned ON with a single call to this API by setting 1 at the bits representing the channels. For example: 0x03h – Turn relay ON on both channels.		
	Note: in the library code, channel 1 relay is mapped to DIO7 and channel 2 to DIO19. But all other evaluation boards (6612, 6613, etc) are setup such that DIO19 to TP4 and DIO7 is grounded. This means relay control doesn't mean much since only DIO19 would be suitable to be controlled in this case but since there is only one channel, there is no point of having a control.		
Return Codes	None.		

4.5.3 MAPI_CloseCircuit_0X()

Purpose	Get zero-crossing status when positioning from open circuit to close circuit. The value returned TRUE/FALSE indicates whether <u>Voltage</u> zero-crossing has occurred. The usage of this API is intended to be for when the circuit is transitioning from open to close. This API uses the hardware Timer 1 in the case when zero crossing does not take place. A delay of 12ms is set for such break.
Synopsis	<code>bool MAPI_CloseCircuit_0X(void);</code>
Parameters	None.
Return Codes	TRUE – Zero crossing occurs. FALSE – Zero crossing does not occur.

4.5.4 MAPI_OpenCircuit_0X()

Purpose	Get zero-crossing status when transitioning from close circuit to open circuit. The value returned TRUE/FALSE indicates whether <u>Current</u> zero-crossing has occurred for a specific channel. There is no indication of either direction (falling/rising). The usage of this API is intended to be for when the circuit is transitioning from close to open. This API uses the hardware Timer 1 in the case when zero crossing does not take place. A delay of 10ms is set for such break.		
Synopsis	<code>bool MAPI_OpenCircuit_0X(unsigned char Channel);</code>		
Parameters	<table border="0"> <tr> <td>Channel</td> <td>Input parameter. Channel/outlet number (1 or 2) where zero crossing will be detected.</td> </tr> </table>	Channel	Input parameter. Channel/outlet number (1 or 2) where zero crossing will be detected.
Channel	Input parameter. Channel/outlet number (1 or 2) where zero crossing will be detected.		
Return Codes	TRUE – Zero crossing occurs. FALSE – Zero crossing does not occur.		

4.6 Soft-Timers

There are eight soft-timers in the 8051 Timer 0. Only six of these timers are available to application level as two are used internally by the library. The timer is a fixed 10-milisecond time increment. When `TSC_Init()` is called, the timer will be started and setup. It also calls `MAPIstm_init()` so the application layer does NOT need to call `MAPIstm_init()`.

4.6.1 `MAPIstm_init()`

Purpose	Initialize soft-timers variables and structures.
Synopsis	<code>void MAPIstm_init (void);</code>
Parameters	None.
Return Codes	None.

4.6.2 `MAPIstm_fn_start()`

Purpose	Start a soft-timer with a call-back function pointer. Upon expiration of the timer, execute the function then remove/free the timer.				
Synopsis	<code>unsigned integer *MAPIstm_fn_start (unsigned integer tenms_count, void (code *fn_ptr) (void))</code>				
Parameters	<table> <tr> <td><code>tenms _count</code></td> <td>Input parameter. Number of counts (in 10ms units) passed to execute the call back function <code>fn_ptr</code>.</td> </tr> <tr> <td><code>*fn_ptr</code></td> <td>Input parameter. Pointer to the call-back function to execute when the <code>tenms_count</code> has expired. The function should be a reentrant.</td> </tr> </table>	<code>tenms _count</code>	Input parameter. Number of counts (in 10ms units) passed to execute the call back function <code>fn_ptr</code> .	<code>*fn_ptr</code>	Input parameter. Pointer to the call-back function to execute when the <code>tenms_count</code> has expired. The function should be a reentrant.
<code>tenms _count</code>	Input parameter. Number of counts (in 10ms units) passed to execute the call back function <code>fn_ptr</code> .				
<code>*fn_ptr</code>	Input parameter. Pointer to the call-back function to execute when the <code>tenms_count</code> has expired. The function should be a reentrant.				
Return Codes	<code>NULL_PTR</code> : if there is no more soft-timer available. Or, The address of where <code>tenms_count</code> value is stored, and counted down, is returned. This is useful when the application needs to stop the timer at any time before it is expired.				

4.6.3 `MAPIstm_run()`

Purpose	Keep the soft-timers updated and current. This API shall be called in the application's main loop after <code>MAPIstm_fn_start()</code> is used. This is where <code>fn_ptr</code> (in <code>MAPIstm_fn_start()</code> above) is processed.
Synopsis	<code>void MAPIstm_run (void);</code>
Parameters	None.
Return Codes	None.

4.6.4 `MAPIstm_wait()`

Purpose	Delay processing for a fixed time (in 10ms increment). Typically use when it is necessary to wait for something to stabilize.
Synopsis	<code>void MAPIstm_wait (IN unsigned integer DelayTime);</code>
Parameters	<code>DelayTime</code> Input parameter Wait (hold up CPU processing) for <code>DelayTime * 10ms</code> .
Return Codes	None.

4.7 DIO Configuration (*future release*)

4.7.1 MAPI_DIOConfig()

Purpose Configure the DIOs to be a specific function. Note: DIO 1 – 8 are part of 80515 USER0 SFR (0x80). DIO8-15 are part of 80515 USER1 SFR (0x90). And DIO16-21 are part of 80515 USER2 SFR (0xA0). These are all bit-addressable SFR. An application shall call this API either right before or right after calling MAPI_Init() to setup all the signals properly. Each member of the Signals_t structure shall be assigned the appropriate DIO pin as defined in the enum type DIO_NO. The library is built with specific default assignments as follows: ActiveB = DIO20, FaultB = DIO21, ReadyB = DIO11, Relay1 = DIO7, Relay2 = DIO19. When unsure, it's best to call this API with a Get Operation (OP=1) to get current settings prior to changing any DIO assignments.

Synopsis

```
void MAPI_DIOConfig (struct Signals_t *Signals, uint8_t OP)
{
    Where:
    //all possible DIO signals
    struct Signals_t {
        uint8_t ActiveB;
        uint8_t Alarm;
        uint8_t FaultB;
        uint8_t ReadyB;
        uint8_t Relay1;
        uint8_t Relay2;
        uint8_t SAG_sign;
        uint8_t WPulse;
        uint8_t RFU;           //Customer's specific -delete?
        uint8_t Unused1;
        uint8_t Unused2;
        uint8_t Unused3;
    };
    enum DIO_NO { DIO0=0x00, DIO1, DIO2, DIO3, DIO4, DIO5, DIO6, DIO7,
                  DIO8, DIO9, DIO10, DIO11, DIO12, DIO13, DIO14, DIO15,
                  DIO16, DIO17, DIO18, DIO19, DIO20, DIO21, DIO22, DIO23};
}
```

Parameters

- OP - Input parameter
Get (1) or Set DIO assignments.
- Signals** - Input parameter (OP = 1)/ Output parameter (OP = 0)
When OP = 1, return the current DIO pin assignments.
When OP = 0, set all DIO pin assignment according to the value assigned to Signals..

Return Codes None.

Note: care must be taken when assigning the same DIO to be used for different functions as there is no checking in the library for any conflict that may arise. For example, DIO6 may be used to drive a LED and to be used as part of the I2C function. The library does NOT check for this conflict of usage.

4.7.2 MAPI_DIORW()

Purpose Perform the Read/Write operation of/to the DIO.

Synopsis `void MAPI_DIORW (enum SIGNAL_TYPE Signal,
 unsigned char *Value,
 uint8_t OP);`

Where:

`enum SIGNAL_TYPE {ACTIVEB, ALARM, FAULTB, READYB, RELAY1B,
 RELAY2B, SAG, WATTPULSE,
 AC_OK, PFC_CTRL, PFC_OK, PFC_ON, Q183_ON,
 UNUSED1, UNUSED2, UNUSED3};`

Also where OP = (READ/ 0, WRITE = 1

Parameters OP: Input parameter.
 0 = read, 1 = write.

Value: Input parameter (OP=0)/ Output parameter (OP=1)..
Read the value of the specified signal (OP=0) or write (OP=1) to it (assuming pin direction is output). There is no checking of pin direction.

Signal: Input parameter
Value is one of: ACTIVEB, ALARM, FAULTB, READYB, RELAY1B,
 RELAY2B, SAG, WATTPULSE,
 AC_OK, PFC_CTRL, PFC_OK, PFC_ON, Q183_ON,
 UNUSED1, UNUSED2, UNUSED3

Return Codes None.

4.8 Flash Management

4.8.1 Memcpy_rx()

Purpose Write to Flash the content data from a specific RAM location. If the length of the source and the starting ROM location cause the write operation to span more than one 512-byte Flash page, the Read/Erase/Verify/Write will take place on all the pages involved. An erase operation will result in the Flash contents being set to 0xFF. CE will be disabled during execution of this API. After the write, this API will validate the write by comparing the Flash content against the RAM content and the return code is reflected from this comparison.

Note1: When calling this API, CE must be turned off. See `MAPI_CEOff` and `MAPI_CEOn` APIs in the sections above.

Note2: it is the task of the application to setup the specific Flash bank before calling this API. An example use of this API is as follows:

```
ok = memcpy_rx ((int8r_t *) ROMData, (int8x_t *) RAMData, ROMSIZE);
```

Synopsis

```
Bool memcpy_rx ( Unsigned char code *dst,
                 Unsigned char xdata *src,
                 Unsigned integer len );
```

Parameters

dst	Input parameter. Specifies starting ROM address of Flash to be written (destination).
src	Input parameter. Use contents at this RAM address location as the source data.
len	Input parameter. Length (in bytes) of data to write to Flash.

Return Codes

- TRUE if the Write was successful.
- FALSE if the Write was not completed.

Note: To avoid accidental write to Flash, this function requires a ‘flash write’ confirmation from the application layer. This confirmation is done such as follows: there shall be a function called `get_buff()`, at application level, which returns a pointer of Xdata whose content is checked as the following:

```
uint8x_t xdata *bptr;
bptr = get_buff ();

if (((']' == toupper (*(bptr + 0)) && 'U' == toupper (*(bptr + 1))) ||
     (']' == toupper (*(bptr + 0)) && 'U' == toupper (*(bptr + 1))) ||
     ('C' == toupper (*(bptr + 0)) && 'A' == toupper (*(bptr + 1))) ||
     ('C' == toupper (*(bptr + 0)) && 'L' == toupper (*(bptr + 1))) ||
     ('C' == toupper (*(bptr + 1)) && 'C' == touper (*(bptr + 2))) ||
     ('U' == toupper (*(bptr + 1)) && 'C' == toupper (*(bptr + 2))) ||
     ('U' == toupper (*(bptr + 1)) && 'M' == toupper (*(bptr + 2))) )
{
    uint8x_t *get_buff (void)
    {
        return Bufptr;
    }
}
```

4.8.2 Memcpy_xr()

Purpose	Use to write to Flash the content data to a specific RAM location. An example use of this API is as follows:	
	<pre>memcpy_xr (RAMData, ROMData, ROMSIZE);</pre>	
Synopsis	Void memcpy_xr (Unsigned char xdata *dst, Unsigned char code *src, Unsigned integer len);	
Parameters	dst Input parameter Specifies starting RAM address to be written (destination).	
	src Input parameter Use contents at this ROM/Flash address location as the source data.	
	len Input parameter Length (in bytes) of data to write to RAM.	
Return Codes	TRUE if the Write was successful. FALSE if the Write was not completed.	

4.8.3 Memcpy_xx()

Purpose	Use to copy data from an xdata location to another xdata location.	
Synopsis	Void memcpy_xx (Unsigned char xdata *dst, Unsigned char xdata *src, Unsigned integer len);	
Parameters	dst Input parameter Specifies starting RAM address to be written (destination).	
	src Input parameter Use contents at this RAM address location as the source data.	
	len Input parameter Length (in bytes) of data to write to RAM.	
Return Codes	None.	

4.9 Serial/RS232 Interface

4.9.1 MAPI_UARTInit()

Purpose Configure the communication speed, flow control, character parity and number of stop bits. The serial interrupt service routine is NOT maskable, the interrupt vector is set internally. Xon/Xoff will always be enabled. Non-parity, 8 data bit, 1 stop bit is the only supported configuration. The only thing the application can change is the baud rate.

Synopsis `Bool MAPI_UARTInit (IN enum SERIAL_SPD speed, Bool XOnOff)`

Parameters **speed:** Input parameter
This selects the communication speed. Possible values are:

<code>_RATE_300</code>	0
<code>_RATE_600</code> ,	1
<code>_RATE_1200</code> ,	2
<code>_RATE_2400</code> ,	3
<code>_RATE_4800</code> ,	4
<code>_RATE_9600</code> ,	5
<code>_RATE_19200</code> ,	6
<code>_RATE_38400</code> ,	7 (default)

Return Codes None.

4.9.2 MAPI_UARTTx()

Purpose Setup the Tx buffer before sending data to the PC UART. Subsequent call to this API will append the buffer to the existing transmittal buffer. An application should call this API immediately after calling `MAPI_UARTInit()`. This is a non-blocking call so always call `MAPI_TxLen` to make sure the transmission is completed successfully.

Synopsis `enum SERIAL_RC data MAPI_UARTTx (U08x xdata *buffer, U16 len)`

Parameters **buffer** Input parameter.
Specifies a pointer to the data buffer containing data to send to the PC UART.
Len Input parameter.
Specifies the current number of bytes to be sent.

Return Codes: `S_EMPTY` Successful transmission.
`S_PENDING`, Successful transmission thus far but not yet finished.
Where return code `SERIAL_RC` is defined as: `Enum SERIAL_RC`.

After calling this API, an application can make sure all bytes were transmitted by checking that `MAPI_TxLen()` returns a 0.

4.9.3 MAPI_TxLen()

Purpose	Number of bytes transmitted thus far.
Synopsis	<code>Unsigned integer MAPI_TxLen (void)</code>
Parameters	None.
Return Value	Unsigned integer specifying the number of bytes left in the Tx buffer, i.e. the remaining bytes to be sent.

4.9.4 MAPI_UARTRx()

Purpose	Setup the receive buffer and start receiving. Always call this function after MAPI_UARTInit() to make sure the receive buffer is available. Subsequent call to this API will append the data to the existing buffer queue, if data is being received. This is a non-blocking call so always check for completion of the reception by calling MAPI_RxLen().				
Synopsis	<code>uint16_t MAPI_UARTRx (U08x xdata *buffer, U16 len)</code>				
Parameters	<table><tr><td>buffer</td><td>Input parameter. Specifies a pointer to the data buffer to store the data received from the PC UART.</td></tr><tr><td>len</td><td>Input parameter. Specifies the maximum number of bytes to receive at any one time.</td></tr></table>	buffer	Input parameter. Specifies a pointer to the data buffer to store the data received from the PC UART.	len	Input parameter. Specifies the maximum number of bytes to receive at any one time.
buffer	Input parameter. Specifies a pointer to the data buffer to store the data received from the PC UART.				
len	Input parameter. Specifies the maximum number of bytes to receive at any one time.				
Return Codes	Unsigned integer specifying the number of bytes fetched thus far. Use MAPI_RxLen() to keep track of the remaining number of bytes unfetched in the Rx buffer.				

4.9.5 MAPI_RxLen()

Purpose	Number of bytes received thus far.
Synopsis	<code>Unsigned Integer MAPI_RxLen (void)</code>
Parameters	None.
Return Value	Unsigned integer specifying the number of bytes in the queued buffer.

5 Default Values

The following default values are used to build the MAPI library:

```

struct MPU_CParms_t  MPU_CParams =
{
    Unsigned Char  Channels = 0
    float      C_Wcal = 120W
    float      C_Vcal = 120V
    float      C_Ical = 1A
    Unsigned Integer t  C_Pcal = 0

    float      C_Wtolerance = 10mA
    float      C_Vtolerance = 10mV
    float      C_Itolerance = 10mA
    float      C_Ptolerance = 0.1 degree

    Unsigned Char C_Vavg_cnt = 3
    Unsigned Char C_Iavg_cnt = 3
    Unsigned Char C_Wavg_cnt = 3

    Unsigned Integer C_Vmax_cnt = 10
    Unsigned Integer C_Imax_cnt = 10
    Unsigned Integer C_Wmax_cnt = 10

    Unsigned Integer C_Wrate_cal = 4860  (shall not change)
    Unsigned Integer C_Tempcal (read-only)
};

#define DefaultTIQ      10 //Default Time Increment Query

struct VIT_t xdata mapi_tVIT =
{
    0x01,           //Channel 1
    0.00,          //Current_I;
    0.00,          //Current_V;
};

struct Relay_Config_t xdata RlyConfig =
{
    0x00,           //unsigned char Relay_CTL;
    0x64,          //unsigned int SeqDlyTime;
    0x01,          //unsigned int DeEnergizedTime;
    0x01,          //unsigned int EnergizedTime;
    0x00,          //InvertPNonLatch
};

```

6 Contact Information

For more information about Maxim products or to check the availability of the 78M6612, contact technical support at www.maxim-ic.com/support.

Revision History

Revision	Date	Description
1.00	09/30/2010	First publication.