ModelGauge m5 Host Side Software Implementation Guide

UG6595; Rev 4; 12/21

Abstract

The ModelGauge m5 Host Side Software Implementation Guide describes the startup sequence to configure and use the MAX17260/1/2/3, MAX20355/7, MAX77658 fuel-gauge functions for EZ config and custom models.

Table of Contents

List of Figures

List of Tables

Introduction

This document describes the startup sequence to configure and use the host side ModelGauge m5 fuel-gauge functions. The host side fuel-gauge should be initialized and loaded with a customized model and parameters at power-up. Then the reported state of charge (SOC) and other useful information can be easily read by the host system over the 2-wire bus system and displayed to the user. **Figure 1** is a flowchart of the power-up sequence that a host controller should implement.

Register LSBs

Similar register types in the ModelGauge™ m5 devices share similar formats, i.e., all the SOC registers share the same format, all the capacity registers share the same format, etc.

Table 1. Register LSBs

REGISTER TYPE	LSb SIZE	NOTES
Capacity	5.0µVh/R_{SENSE}	Or 0.5mAh with 10mΩ. For internal R_{Sense} parts, refer to the IC datasheet for Capacity LSB.
SOC	1/256%	Or 1% at bit D8.
Voltage	0.078125mV	Or 1.25mV at bit D4.
Current	1.5625µV/R_{SENSE}	Or 156.25µA with 10mΩ, signed two's complement number. For internal R_{Sense} parts, refer to the IC datasheet for Current LSB.
Temperature	1/256°C	Or 1°C at bit D8, signed two's complement number.

2-Wire/I2C Functions

The following I2C functions are needed in the load model process. They are described in pseudocode below.

WriteRegister

```
int WriteRegister (u8 reg, u16 value)
 {
         int ret = i2c_smbus_write_word_data(client, reg, value);
         if (ret < 0)
                 dev_err(&client->dev, "%s: err %d\n", __func__, ret);
         return ret;
 }
```

ReadRegister

```
int ReadRegister (u8 reg)
 {
         int ret = i2c_smbus_read_word_data(client, reg);
         if (ret < 0)
                 dev_err(&client->dev, "%s: err %d\n", __func__, ret);
         return ret;
 }
```

WriteAndVerifyRegister

```
void WriteAndVerifyRegister (char RegisterAddress, int RegisterValueToWrite){
    int Attempt=0;
    do {
        WriteRegister (RegisterAddress, RegisterValueToWrite);
        Wait(1);       //1ms
        RegisterValueRead = ReadRegister (RegisterAddress) ;
        }
    while (RegisterValueToWrite != RegisterValueRead && attempt++<3);
}
```

Initialize Registers to Recommended Configuration

The IC should be initialized prior to being used. The registers described in this guide should be written to the correct values for the IC to perform at its best. These values are written to RAM, so they must be written to the device any time power is applied or restored to the device. Some registers are updated internally, so it is necessary to verify that the register was written correctly to prevent data collisions. During the whole initialization process, it is important to keep the system consumption to minimum and disable any charging. Ideally, the battery is in the relaxed state during initialization. This helps to get better initial SOC estimation.

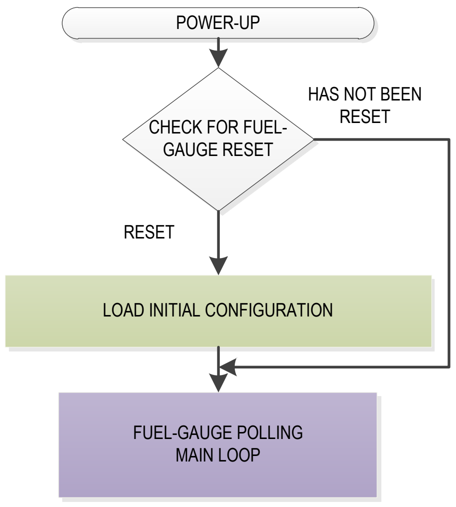
POWER-UP
CHECK FOR FUEL-GAUGE RESET
HAS NOT BEEN RESET
RESET
LOAD INITIAL CONFIGURATION
FUEL-GAUGE POLLING MAIN LOOP

Figure 1. ModelGauge m5 host side fuel-gauge model loading sequence.

Step 0: Check for POR

The POR bit is bit 1 of the Status register.

```
StatusPOR = ReadRegister(0x00) & 0x0002;
if (StatusPOR=0){goto Step 3.2;} //then go to Step 3.2.
else  { //then do Steps 1-2.}
```

Step 1. Delay until FSTAT.DNR bit == 0

After power-up, wait for the IC to complete its startup operations.

```
while(ReadRegister(0x3D)&1) Wait(10);
        //10ms Wait Loop. Do not continue until FSTAT.DNR==0
```

Step 2: Initialize Configuration

Any battery is supported by one of three types of configuration data. According to the configuration data, only one of the following sections 2.1, 2.2, or 2.3 needs execution.

```
HibCFG = ReadRegister(0xBA) ;         //Store original HibCFG value
WriteRegister (0x60 , 0x90)      ;  // Exit Hibernate Mode step 1
WriteRegister (0xBA , 0x0)       ;  // Exit Hibernate Mode step 2
WriteRegister (0x60 , 0x0)       ;  // Exit Hibernate Mode step 3

2.1 OPTION 1 EZ Config (No INI file is needed):
WriteRegister (0x18 , DesignCap) ;  // Write DesignCap
WriteRegister (0x1E , IchgTerm)  ;  // Write IchgTerm
WriteRegister (0x3A , VEmpty)    ;  // Write VEmpty

if (ChargeVoltage>4.275)
        WriteRegister (0xDB , 0x8400) ;     // Write ModelCFG
else
        WriteRegister (0xDB , 0x8000) ;     // Write ModelCFG

//Poll ModelCFG.Refresh(highest bit),
//proceed to Step 3 when ModelCFG.Refresh=0.
while (ReadRegister(0xDB)&0x8000) Wait(10);
        //do not continue until ModelCFG.Refresh==0

WriteRegister (0xBA , HibCFG)  ; // Restore Original HibCFG value

Proceed to Step 3.

2.2 OPTION 2 Custom Short INI without OCV Table:
WriteRegister (0x18 , DesignCap) ;  // Write DesignCap
WriteRegister (0x1E , IchgTerm)  ; // Write IchgTerm
WriteRegister (0x3A , VEmpty)    ;  // Write VEmpty
WriteAndVerifyRegister (0x28 , LearnCFG)  ;// (Optional in the INI)
WriteAndVerifyRegister (0x13 , FullSOCThr)  ; // (Optional in the INI)

WriteRegister (0xDB , ModelCfg) ;     // Write ModelCFG

//Poll ModelCFG.Refresh(highest bit)
//until it becomes 0 to confirm IC completes model loading
while (ReadRegister(0xDB)&0x8000) Wait(10);
        //do not continue until ModelCFG.Refresh==0

WriteRegister (0x38 , RCOMP0)     ; // Write RCOMP0
WriteRegister (0x39 , TempCo)     ; // Write TempCo
WriteRegister (0x12 , QRTable00)  ; // Write QRTable00
WriteRegister (0x22 , QRTable10)  ; // Write QRTable10
WriteRegister (0x32 , QRTable20)  ; //(Optional in the INI)  Write QRTable20
WriteRegister (0x42 , QRTable30)  ; //(Optional in the INI)  Write QRTable30
WriteRegister (0xBA , HibCFG)     ; // Restore Original HibCFG value

Proceed to Step 3.
```

```
2.3 OPTION 3 Custom Full INI with OCV Table:
2.3.1   Unlock Model Access
        WriteRegister (0x62, 0x0059) ; //Unlock Model Access step 1
        WriteRegister (0x63, 0x00C4) ; //Unlock Model Access step 2

2.3.2   Write/Read/Verify the Custom Model
Once the model is unlocked, the host software must write the 32-word model to
the IC. The model is located between memory locations 0x80h and 0x9Fh.

//Actual bytes to transmit will be provided by Analog Devices after cell
characterization.

//See INI file at the end of this document for an example of the data to be
written.
Write16Registers (0x80, Table [0]) ; Table [0] is 16 words described data
described in INI file format
Write16Registers (0x90, Table [1]) ; Table [1] is 16 words described data
described in INI file format
WriteRegister (0xAF, RCompSeg); // RCompSeg value is described in INI file
format

The model can be read directly back from the IC. So simply read the 48 words
of the model back from the device to verify if it was written correctly. If
any of the values do not match, return to step 2.3.1.
Read16Registers (0x80) ;
Read16Registers (0x90) ;
ReadRegister (0xAF) ;

2.3.3 Lock Model Access
WriteRegister (0x62, 0x0000) ;        //Lock Model Access
WriteRegister (0x63, 0x0000) ;

2.3.4. Verify that Model Access is locked
If the model remains unlocked, the IC will not be able to monitor the
capacity of the battery. Therefore it is very critical that the Model Access
is locked. To verify it is locked, simply read back the model. However, this
time, all values should be read as 0x00h. If any values are non-zero, repeat
Step 2.3.3 to make sure the Model Access is locked.

2.3.5. Write Custom Parameters
Wait 100ms
WriteAndVerifyRegister (0x05,0x0000); // Clear RepCap
WriteRegister (0x18, DesignCap); // Write DesignCap
if (saved parameters history exists) {
    LoadFullCapRep = Saved_FullCapRep;
    LoadFullCapNom = Saved_FullCapNom;
    LoadRCOMP0 = Saved_RCOMP0;
    LoadTempCo = Saved_TempCo;
    LoadCycles = Saved_Cycles;
} else {
    LoadFullCapRep = DesignCap;
    LoadFullCapNom = DesignCap;
    LoadRCOMP0 = RCOMP0;
    LoadTempCo = TempCo;
    LoadCycles = 0;
}
WriteRegister (0x10,LoadFullCapRep); // Write FullCapRep
intAttempt = 0;
do {
    WriteRegister (0x45, LoadFullCapNom/2); // Write dQAcc
    WriteRegister (0x46, 0x0C80; // Write dPAcc
    Wait 10ms
    WriteRegister (0x23, LoadFullCapNom); // Write FullCapNom
}
While (((ReadRegister (0x23) ! = LoadFullCapNom) |
(ReadRegister (0x45) ! = LoadFullCapNom/2) |
(ReadRegister (0x46) ! = 0xC80)) && attempt++<3) ;
Update_Capacity = Integer (ReadRegister (0xFF)/ 25600.00 * LoadFullCapNom);
WriteRegister (0x0F, Update_Capacity); // Write MixCap
WriteRegister (0x1F, Update_Capacity); // Write AvCap
Wait 200ms
WriteRegister (0x1E, IchgTerm); // Write IchgTerm
WriteRegister (0x3A, VEmpty); // Write VEmpty
WriteRegister (0x38, LoadRCOMP0); // Write RCOMP0
WriteRegister (0x39, LoadTempCo); // Write TempCo
WriteRegister (0x12, QRTable00); // Write QRTable00
WriteRegister (0x22, QRTable10); // Write QRTable10
WriteRegister (0x32, QRTable20); // Write QRTable20 (optional)
WriteRegister (0x42, QRTable30); // Write QRTable30 (optional)
```

```
2.3.6 Updating optional registers. Some or all of the registers listed below
could be optional and may not be included in the INI.
WriteAndVerifyRegister (0x28 , LearnCFG); // Write LearnCFG
WriteRegister (0x2A, RelaxCFG)          ; //Write RelaxCFG
WriteRegister (0x1D, Config)            ; //Write Config
WriteRegister (0xBB, Config2)           ; //Write Config2
WriteRegister (0x13, FullSOCthr)        ; //Write FullSOCthr
WriteRegister (0x2C, TGAIN) ;       //Write TGAIN for the selected Thermistor
WriteRegister (0x2D, TOFF)  ;       //Write TOFF for the selected Thermistor
WriteRegister (0xB9, Curve)  ;      //Write Curve for the selected Thermistor
WriteRegister (0x2B, MiscCfg) ;     //Write MiscCfg

2.3.7 Initiate Model Loading
Config2value = ReadRegister(0xBB)
;    //read the Config2 register (0xBB)
WriteRegister(0xBB,((Config2value) | (0x0020))) ;   // Setting the LdMdl bit
in the Config2 register


Poll the LdMdl bit in the Config2 register, proceed to step 2.3.8 when LdMdl
bit becomes 0.

//Poll Config2.LdMdl(0x0020)
//until it becomes 0 to confirm IC completes model loading
while (ReadRegister (0xBB)&0x0020){
WriteRegister (0x0A, 0x0000);
WriteRegister (0x0B, 0x0000);
Wait 10ms ; }

2.3.8 Update QRTable20, QRTable30 and Cycles
WriteAndVerifyRegister (0x32, QRTable20); //Write QRTable20
WriteAndVerifyRegister (0x42, QRTable30); //Write QRTable30
WriteAndVerifyRegister (0x17, LoadCycles); //Write Cycles

2.3.9 Restore HibCFG
WriteRegister (0xBA ,HibCFG)       ; // Restore Original HibCFG value

Proceed to Step 3.
```

Step 3: Initialization Complete

Clear the POR bit to indicate that the custom model and parameters are successfully loaded.

```
Status = ReadRegister(0x00)  ;                          //Read Status
WriteAndVerifyRegister (0x00, Status AND 0xFFFD)  ;   //Write and Verify
Status with POR bit Cleared
```

Monitor the Battery

Once the IC is initialized and customized, the host can simply read the required information from the IC and display that information to the user.

Step 3.1: Check for IC Reset

Periodically the host should check if the fuel gauge has been reset and initialize it if needed.

```
StatusPOR = ReadRegister(0x00) & 0x0002; //Read POR bit in Status Register
If StatusPOR = 0, then go to Step 3.2.
If StatusPOR = 1, then go to Step 0.
```

Read the Fuel-Gauge Results

Step 3.2: Read the RepCap and RepSOC Registers

The IC automatically calculates and reports the cell’s state of charge in terms of a percentage and the mAhrs remaining. The RepSOC (as a percent) is read from memory location 0x06 and the RepCap (in mAHrs) is read from memory location 0x05.

```
RepCap = ReadRegister(0x05) ;         //Read RepCap
RepSOC = ReadRegister(0x06) ;         //Read RepSOC
```

The RepSOC has a LSB of 1/256%. Round the RepSOC to the nearest integer value.

Step 3.3: Read the Remaining TTE Register

The IC also calculates the Time-to-Empty register (TTE). TTE is in memory location 0x11h. The LSB of the TTE register is 5.625 seconds.

```
TTE = ReadRegister(0x11) ;                //Read TTE
```

Step 3.4: Save Learned Parameters

It is recommended saving the learned capacity parameters every time bit 6 of the Cycles register toggles (so that it is saved every 64% change in the battery) so that if power is lost, the values can easily be restored.

```
Saved_RCOMP0 = ReadRegister(0x38)      ;    //Read RCOMP0
Saved_TempCo = ReadRegister(0x39)      ;    //Read TempCo
Saved_FullCapRep = ReadRegister(0x10) ;    //Read FullCapRep
Saved_Cycles = ReadRegister(0x17)      ;    //Read Cycles
Saved_FullCapNom = ReadRegister(0x23) ;    //Read FullCapNom
```

Quick Start

This section is not needed in most applications. In some cases, if the battery protection is triggered or voltage transient appears during IC initialization, the initial OCV detection is off from the real OCV. This results in bad initial SOC estimation. In this case, the quick start command helps the device to get the correct initial SOC after the supply voltage is stable.

Step T1: Set the Quick-Start Bits

```
Data= ReadRegister(0x2B)       ;  //Read MiscCFG
Data |= 0x0400                ;  //Set bits 10
WriteRegister (0x2B, Data)    ;  //Write MiscCFG
```

Step T2: Wait for Quick Start to Complete

```
//Poll MiscCFG.QS(0x0400) and FSTAT.DNR until they becomes 0 to
confirm Quickstart is finished
While (ReadRegister(0x2B)&0x0400 and ReadRegister(0x3D)&1) Wait(10);
       //do not continue until quickstart is complete
```

MAX1726x INI File Format

When high accuracy is needed and when Analog Devices generates a custom INI file, the INI is in one of the formats mentioned below. The examples below can be used to structure the software to read the custom INI for the battery. Option 1 (not shown) is the EZ mode, which does not have a custom INI.

Option 2: Short Format

```
Device=MAX1726X
Title=C:/xxxx/1234_1_111111.csv
ModelVersion=8745  //This keeps track of the version of the INI generator

DesignCap=0x1450
ichgterm=0x333
modelcfg=0x8000
QRTable00=0x1050
QRTable10=0x2012
VEmpty=0xa561
RCOMP0=0x004d
TempCo=0x223e
```

Option 3: Long Format

```
Device=MAX1726X
Title= C:/xxxx/1234_1_111111.csv
ModelVersion=8745

DesignCap=0x06ae
fullsocthr=0x5f05
ichgterm=0x100
modelcfg=0x8410
QRTable00=0x1050
QRTable10=0x0014
QRTable20=0x1300
QRTable30=0x0c00
VEmpty=0x965a
RCOMP0=0x0070
TempCo=0x223e

;;; Begin binary data
;;; This is formatted as 16-bit words, each on a new line.
;;; Numbers are formatted in hex, for example: 0x0000
; Ignore the first 16 words. These are used by EVKit software only
; 16 words. Data starts here for Address 0x80 for use in Step 2.3.2
; 16 words. Data starts here for Address 0x90 for use in Step 2.3.2
; 16 words. They are the same value for RCompSeg register. The value is for
loading register 0xAF in Step 2.3.2
; Ignore the remaining 32 words
```

Trademarks

ModelGauge is a trademark of Analog Devices Products, Inc.

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	3/18	Initial release	—
1	4/18	Added WriteRegister (0x10, FullCapRep) and WriteRegister (0x45, DesignCap/2) lines of code to 2.3.5 Write Custom Parameters	8
2	6/18	Updated 2.3.5 Write Custom Parameters	8
3	5/21	Updated Section 2.3.2, 2.3.5, 2.3.7. Quick Start, INI file format	—
4	12/21	Updated the Title and Abstract, and section's "Introduction", "Register LSBs", "Initialize Registers to Recommended Configuration", "2.2", "2.3", "Monitor the Battery", "Read the Fuel-Gauge Results", "Quick Start", "Table 1" and title of the Figure 1	1, 4, 6-10, 12-14