MAX17055
Software Implementation Guide

UG6365; Rev 0; 1/17

Description

This document describes the startup sequence to configure and utilize the fuel gauge functions of MAX17055. The MAX17055 should be initialized and loaded with a customized model and parameters at power up and then the reported state of charge 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 with the MAX17055.

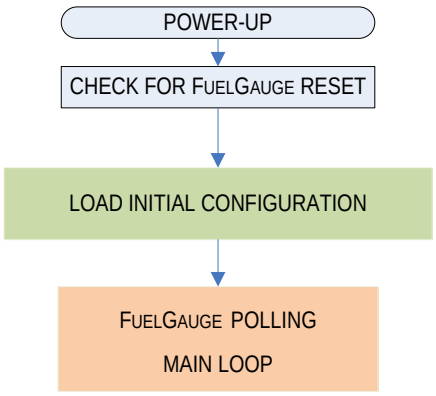

Figure 1. MAX17055 fuel gauge model loading sequence.

Register LSBs for MAX17055

Similar register types in the ModelGauge m5 devices share similar formats: e.g., all of the SOC registers share the same format, all of the Capacity registers share the same format.

Table 1. Register LSbs

REGISTER TYPE	LSb SIZE	NOTES
Capacity	5.0µVH/R_{SENSE}	or 0.5mAh with 10mΩ
SOC	1/256%	or 1% at bit D8
Voltage	0.078125mV	or 1.25mV at bit D4
Current	1.5625 µV/R_{SENSE}	or 156.25µA with 10mΩ, signed 2's complement number
Temperature	1/256°C	or 1°C at bit D8, signed 2's complement number

Resistor Selection

Table 2. Current Sense Resistor Selection.

BATTERY FULL CAPACITY (mAh)	SENSE RESISTOR (mΩ)	CURRENT REGISTER RESOLUTION (µA)	CURRENT REGISTER RANGE (A)	CAPACITY RESOLUTION (mAh)
1600 – 24576	5	312.5	±10.24	1
800-12288	10	156.25	±5.12	0.5
400-6144	20	78.125	±2.56	0.25
160-2458	50	31.25	±1.024	0.1
40-614	200	7.8125	±0.256	0.025

An easy rule is to scale Rsense so that the hex value of DesignCap is between 0x640 to 0x6000.

2-Wire/I²C Functions

The following I²C functions are needed in the load model process. They are described in pseudo code 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 MAX17055 should be initialized prior to being used. The following registers should be written to these values in order for the MAX17055 to perform optimally. These values are written to RAM, so they must be written to the device any time that 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.

1. Check for POR

The POR bit is bit 1 of the Status register.

```
StatusPOR = ReadRegister(0x00) & 0x0002;
if (StatusPOR==0) {goto Step 4.3;} //then go to Step 4.3.
else { //then do Steps 2–3.}
```

2. Delay until FSTAT.DNR bit == 0

After power-up, wait for the MAX17055 to complete its startup operations.

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

3. Initialize configuration

```
3.1 OPTION 1 EZ Config (no INI file is needed):
WriteRegister (0x18 , DesignCap) ; // Write DesignCap
WriteRegister (0x45 , DesignCap/32) ; //Write dQAcc
WriteRegister (0x1E , IchgTerm) ; // Write IchgTerm
WriteRegister (0x3A , VEmpty)  ; // Write VEmpty
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

if (ChargeVoltage>4.275)
                WriteRegister (0x46, int(DesignCap/32)*51200/DesignCap) ;          // Write dPAcc
                WriteRegister (0xDB , 0x8400) ;  // Write ModelCFG
Else
                WriteRegister (0x46 , int(DesignCap/32)*44138/DesignCap) ;       //Write dPAcc
                WriteRegister (0xDB , 0x8000) ;  // Write ModelCFG

//Poll ModelCFG.Refresh(highest bit), proceed to Step 4 when ModelCFG.Refresh = 0.
While (ReadRegister(0xDB)&0x8000) Wait(10) ; //10ms wait loop.  Do not continue until ModelCFG.Refresh == 0.

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

Proceed to Step 4.

3.2 OPTION 2 Custom Short INI without OCV table:

```
WriteRegister (0x18 , DesignCap) ; // Write DesignCap
WriteRegister (0x45 , DesignCap/16) ; //Write dQAcc
WriteRegister (0x1E , IchgTerm) ; // Write IchgTerm
WriteRegister (0x3A , VEmpty)  ; // Write VEmpty
WriteAndVerifyRegister (0x28 , LearnCFG) ;// (Optional in the INI) Write LearnCFG
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
WriteRegister (0x46 , dPAcc) ; //Write dPAcc
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 4.

3.3 OPTION 3 Custom Full INI with OCV Table:

3.3.1 Unlock Model Access

```
                WriteRegister (0x62, 0x0059) ; //Unlock Model Access step 1
                WriteRegister (0x63, 0x00C4) ; //Unlock Model Access step 2
```

3.3.2 Write/read/verify the custom model

Once the model is unlocked, the host software must write the 48-word model to the MAX17055. The model is located between memory locations 0x80h and 0xAFh.

```
//Actual bytes to transmit are provided by Maxim after cell characterization.

//See INI file at the end of this document for an example of the data to be written.
        Write16Registers (0x80) ;
        Write16Registers (0x90) ;
        Write16Registers (0xA0) ;
```

The model can be read directly back from the MAX17055. 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 3.3.1.

```
        Read16Registers (0x80) ;
        Read16Registers (0x90) ;
        Read16Registers (0xA0) ;
```

3.3.3 Lock model access.

```
        WriteRegister (0x62, 0x0000) ;   //Lock model access
        WriteRegister (0x63, 0x0000) ;
```

3.3.4. Verify that model access is locked.

If the model remains unlocked, the MAX17055 cannot 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 3.3.3 to make sure the model access is locked.

3.3.5. Write custom parameters

```
        WriteRegister (0x18 , DesignCap) ; // Write DesignCap
        WriteRegister (0x45 , DesignCap/16) ; //Write dQAcc
        WriteRegister (0x46, 0x0C80) ;          // Write dPAcc
        WriteRegister (0x1E , IchgTerm) ; // Write IchgTerm
        WriteRegister (0x3A , VEmpty) ; // Write VEmpty
        WriteRegister (0x38 , RCOMP0) ; // Write RCOMP0
        WriteRegister (0x39 , TempCo) ; // Write TempCo
        WriteRegister (0x12 , QRTable00) ; // Write QRTable00
        WriteRegister (0x22 , QRTable10) ; // Write QRTable10
```

Updating required registers.

```
        WriteRegister (0x05, 0x0)     ; //Write 0 to RepCap
        VFSOC = ReadRegister (0xFF)     ; //Read Value of VFSOC
        WriteAndVerifyRegister (0x48, VFSOC) ; //WriteAndVerify VFSOC0
        WriteAndVerifyRegister (0x10, DesignCap); //WriteAndVerify FullCapRep
        WriteAndVerifyRegister (0x23, DesignCap); //WriteAndVerify FullCapNom
        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
```

Updating optional registers. Some or all of the below registers may 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
3.3.6 Initiate Model Loading
WriteRegister (0xBB, Config2)     ; //Write Config2
Config2value=ReadRegister(0xBB)                ;  //read the Config2 register (0xBB)
WriteRegister(0xBB,((Config2value) | (0x0020))) ;  // Setting the LdMdl bit in the Config2 register


3.3.7
Poll the LdMdl bit in the Config2 register, proceed to step 3.3.8 when LdMdl bit becomes 0.

3.3.8 Update QRTable20 and QRTable30
WriteAndVerifyRegister (0x32 , QRTable20)   ; // Write QRTable20
WriteAndVerifyRegister (0x42 , QRTable30)   ; // Write QRTable30

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

Proceed to Step 4.
```

4. Initialization complete

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

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

4.1 Identify the battery

If the host recognizes the battery pack as one with a saved history, go to Step 4.5 to restore all of the saved parameters, otherwise, continue to Step 4.1.

Monitor the battery.

Once the MAX17055 is initialized and customized, the host can simply read the desired information from the MAX17055 and display that information to the user.

4.2 Check for MAX17055 reset

```
StatusPOR = ReadRegister(0x00) & 0x0002; //Read POR bit in Status register
If StatusPOR = 0, then go to Step 4.3.
If StatusPOR = 1, then go to Step 1.
```

Read the fuel gauge results.

4.3 Read the reported capacity and state of charge (SOC)

The MAX17055 automatically calculates and reports the state of charge of the cell in terms of a percentage and the mAhrs remaining. The reported state of charge (RepSOC), as a percent, is read from memory location 0x06 and the reported capacity (RepCap), in mAHrs, is read from memory location 0x05.

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

The RepSOC_HiByte has a unit of 1%, so the RepSOC_HiByte can be directly displayed to the user for 1% resolution. It is recommended to save the RepSOC value so that it can be used to restore the capacity information in the event of a power loss.

4.4 Read the remaining time to empty (TTE)

The MAX17055 also calculates the time to empty (TTE). TTE is in memory location 0x11h. The LSB of the TTE register is 5.625s.

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

4.5 Save learned parameters

It is recommended to save the learned capacity parameters every time bit 2 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
```

4.6 Restoring capacity parameters

If power is lost, then the capacity information can be easily restored with the following procedure.

```
WriteAndVerifyRegister(0x38, Saved_RCOMP0) ; //WriteAndVerify RCOMP0
WriteAndVerifyRegister(0x39, Saved_TempCo) ; //WriteAndVerify TempCo
WriteAndVerifyRegister(0x23, Saved_FullCapNom) ; //WriteAndVerify FullCapNom
```

4.7 Wait 350ms

4.8 Restore FullCap

```
FullCapNom= ReadRegister(0x23)       ; //Read FullCapNom
MixCap=(ReadRegister(0x0D)*FullCapNom)/25600 ;
WriteAndVerifyRegister(0x0F, MixCap)     ; //WriteAndVerify MixCap
WriteAndVerifyRegister(0x10, Saved_FullCapRep) ; //WriteAndVerify FullCapRep
//Write dQacc to 200% of Capacity and dPacc to 200%
dQacc = (Saved_FullCapNom/ 16)    ;
WriteAndVerifyRegister (0x46, 0x0C80) ;   //Write and Verify dPacc
WriteAndVerifyRegister (0x45, dQacc) ;   //Write and Verify dQacc
```

4.10 Wait 350ms

4.11 Restore Cycles register

```
WriteAndVerifyRegister(0x17, Saved_Cycles) ;   //WriteAndVerify Cycles
```

Quickstart and Production Test Verification

If the IC is being production tested, the following procedure updates the fuel gauge outputs to a known state to verify proper operation of the IC. The sequence for configuring the MAX17055, as outlined in the MAX17055 Software Implementation Guide, should be followed prior to sending the Quickstart command to verify that the device was set up correctly. Set the power supply to the desired voltage and issue the Quickstart command as described below.

Step T1. Set the Quickstart and Verify bits

A second bit must also be set to 1 to test for possible memory leak since the IC also updates this register internally.

```
Data= ReadRegister(0x2B  ; //Read MiscCFG
Data |= 0x1400       ; //Set bits 10 and 12
WriteRegister (0x2B, Data) ; //Write MiscCFG
```

Step T2. Verify there are no memory leaks during Quickstart writing

If the IC was writing the MiscCFG register internally at the same time as host software, bit 12 does not remain set. Check bit 12 to confirm Quickstart was correctly received by the IC.

```
Data= ReadRegister(0x2B)     ; //Read MiscCFG
Data &= 0x1000          ;
if (Data==0x1000) go to step T3 ; //Quickstart Success
else go to step T1        ;
```

Step T3. Clear the Verify bit

After a successful Quickstart, the verify bit must now be cleared.

```
Data= ReadRegister(0x2B)    ;  //Read MiscCFG
Data &= 0xEFFF        ;  //Clear bit 12
WriteRegister (0x2B, Data)    ;  //Write MiscCFG
```

Step T4. Verify there are no memory leaks during Verify bit clearing

If the IC was writing the MiscCFG register internally at the same time as host software, bit 12 does not clear. Check bit 12 to confirm.

```
Data= ReadRegister(0x2B)     ; //Read MiscCFG
Data &= 0x1000          ;
If (Data==0x0000) go to step T5 ; //Verify clear success
Else go to step T3        ;
```

Step T5. Delay 500ms

After Quickstart, the MAX17055 requires 500ms to perform signal debouncing and initial calculations.

Step T6. Write and verify FullCAP Register value.

The FullCAP register value must be rewritten after a Quickstart event.

```
WriteAndVerifyRegister(0x10, Capacity) ; //WriteAndVerify FullCapRep
```

Step T7. Delay 500ms

After updating FullCAP value, the MAX17055 requires 500ms to calculate output register results.

Step T8. Read and Verify Outputs

The RepCap and RepSOC register locations should now contain accurate fuel gauge results based on a battery voltage of 3.900V. The reported state of charge (RepSOC), as a percent, is read from memory location 0x06h and the reported capacity (RepCap), in mAHrs, is read from memory location 0x05h.

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

Fail the unit if RepCap and RepSOC do not report expected results to within ±1% not including power supply tolerance. Note that any error in the voltage forced on V_{BATT} for testing creates a much larger error in the output results from the fuel gauge.

MAX17055 INI File Format

Option 2: Short Format without OCV Table

```
Device = MAX17055
Rsense = 20mOhm
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
dPacc=0x0c80
```

Option 3: Long Format with OCV Table

```
Device = MAX17055
Rsense = 20mOhm
Title = C:/xxxx/1234_1_111111.csv
ModelVersion = 8745

designCap = 0x06ae
fullsocthr = 0x5f05
ichgterm = 0x100
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 EV kit software only.
; 16 words. Data starts here for Address 0x80 for use in Step 3.3.
; 16 words. Data starts here for Address 0x90 for use in Step 3.3.
; 16 words. Data starts here for Address 0xA0 for use in Step 3.3.

; Ignore the remaining 32 words.
```