UM1836
User manual

STSW-BAT001 open source driver for the STC3115

Aurelien Mazard

Introduction

The STC3115 open source driver is a ready-to-use, configurable software (STSW-BAT001) which allows the STC3115 gas gauge IC to be rapidly integrated to the final system software. The driver is split into five C language files and can be compiled with any operating system for any processor.

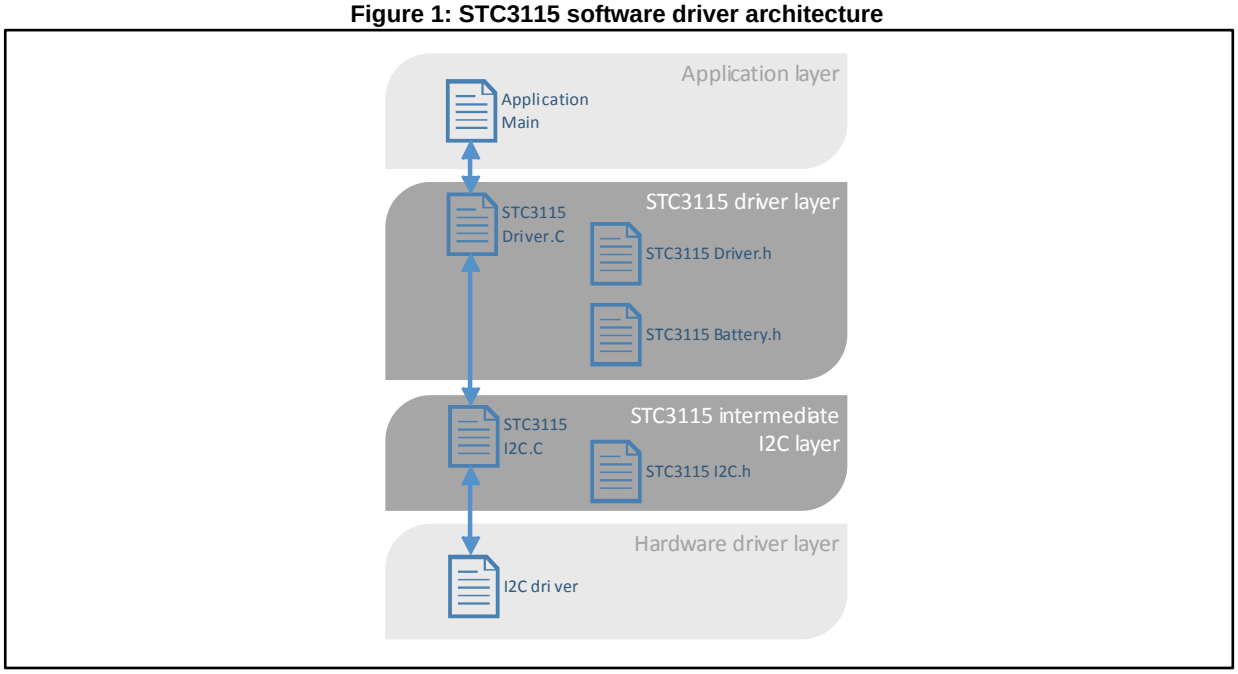
The STC3115 open source driver comes with a full API which uses simple input information to initialize and control the STC3115 registers and RAM and return battery status information in a simple structure to the system.

The open source driver is available for free to manage system requirements and to handle the complexity of STC3115 initialization. It is intended only for example purposes.

This user manual describes the STC3115 open source driver (STSW-BAT001) package content, capabilities, and correct usage. It also explains the step-by-step procedure to configure and integrate the open source driver in a complete software architecture.

The figure below shows an overview of the STC3115 software driver architecture. No additional software is needed to manage the STC3115.

Figure 1: STC3115 software driver architecture



Contents

1 Software overview

The STC3115 device is a battery monitoring IC which uses simple battery parameters and system information to provide information on the battery state, such as the remaining battery capacity, battery voltage, and battery current.

The STC3115 software driver package manages the STC3115 including device configuration and battery information read back. Management and configuration of the device is in accordance with battery and application events.

The STC3115 software driver package provides several files; the content of each is described in the sections below. Their organization and connections are shown in *Figure 1*.

1.1 stc3115_Driver.c

This file contains the functions to initialize and report the battery state based on STC3115 algorithms. The functions include the following features:

- Dedicated I^2C functions which are linked to the interface driver functions in the stc3115_I2C.c file (see *Section 1.4*). The hardware system has to support multiple I^2C access commands to guarantee data integrity during the 16-bit register read/write operations.
- The STC3115 driver's own functions which are contained in the first part of the stc3115_Driver.c file. These functions should not be accessed by external functions.
- Gas gauge functions which are the main interface functions and which have to be used by the application to initialize, monitor, and stop the STC3115 device.
- Power saving functions which can be used to manage the STC3115 operating mode (mixed mode or voltage mode). Power saving functions have to be called by an external program to change the STC3115 operating mode after STC3115 initialization.
- Alarm management functions which have to be called to set, stop, get, clear, and change the alarm thresholds after STC3115 initialization.

1.2 stc3115_Driver.h

This file defines the STC3115 constant numbers and the driver API structures and functions as follows:

- The STC3115_ConfigData_TypeDef structure is filled by the driver with battery configuration data from the stc3115_Battery.h (see *Section 1.3*) information.
- The STC3115_BatteryData_TypeDef structure is filled by the driver with the battery state.
- The RAMData structure is the RAM memory map description which is used internally by the driver. This structure must not be modified by external accesses.

1.3 stc3115_Battery.h

This file is used to describe the battery and application configuration. It has to be filed with battery and application data (see *Section 2*).

1.4 stc3115_I2C.c

This file is used to define the I^2C exchange functions between the STC3115 driver and the application I^2C driver. This file has to be updated with the functions of the application I^2C driver. The hardware system has to support multiple I^2C access commands to guarantee data integrity during the 16-bit register read/write operations.

1.5 stc3115_I2C.h

This file is used to declare the I^2C exchange functions between the STC3115 driver and the application I^2C driver.

2 Software configuration

Configuration of the STC3115 open source driver is done using the stc3115_Battery.h file. This file allows the battery and application parameters to be configured with the minimum variable set.

2.1 Battery parameters configuration

An extract of the stc3115_Battery.h file where battery parameters are defined is shown below. A detailed explanation of the terms used in this code (CAPACITY, RINT, and OCV_OFFSET_TAB) is also available below.

```
/*Battery parameters define ---------------------------------------*/
#define CAPACITY 1500 /* battery nominal capacity in mAh            */
#define RINT 200 /* Internal battery impedance in mOhms,0 if unknown*/

#define OCV_OFFSET_TAB {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
/* OCVTAB */
```

The CAPACITY parameter describes the typical battery capacity in mAh.

The RINT parameter describes the internal battery impedance in mOhm. The value has to include battery cell impedance and battery security IC impedance. If the battery is connected to the STC3115 by long and/or small wires, the wire impedance has also to be included in this parameter. If this parameter is unknown, it can be declared as zero. In this case, a default value of 200 mOhms is used for the RINT.

The OCV_OFFSET_TAB is the REG_OCVTAB register described in the STC3115 datasheet. This register can be filled by using the battery open circuit voltage (OCV) curve (see AN4324). If the OCV adjustment table is set up at 0, the OCV curve used to monitor the battery is the default OCV programmed in the STC3115 internal registers (see STC3115 datasheet). This default configuration is adequate to maintain a functional system. The OCV adjustment table must not be filled with any value below -127 or above 127.

2.2 Application parameters configuration

An extract of the stc3115_Battery.h file where the application parameters are defined is shown below. A detailed explanation of the parameters used in this code (VMODE, ALM_EN, ALM_SOC, ALM_VBAT, RSENSE, APP_EOC_CURRENT, and APP_CUTOFF_VOLTAGE) is also available below.

```
/*Application parameters define -------------------------------------*/
#define VMODE MIXED_MODE/* running mode constant, VM_MODE or
MIXED_MODE*/
#define ALM_EN 0      /* Alarm enable constant, set at 1 to enable  */
#define ALM_SOC 10    /* SOC alarm in %                             */
#define ALM_VBAT 3600 /* Voltage alarm in mV                        */
#define RSENSE 10     /* sense resistor in mOhms                    */

#define APP_EOC_CURRENT 75      /* end charge current in mA         */
#define APP_CUTOFF_VOLTAGE 3000 /* application cut-off voltage in mV*/
```

The VMODE parameter describes the VMODE bit in the REG_MODE register (see STC3115 datasheet). This parameter has to be set up as the default operating mode of the application. It can be initialized with MIXED_MODE or VM_MODE. If mixed mode is selected, an external sense resistor is mandatory to measure the battery current that is flowing. Also, if mixed mode is selected, it is possible to switch to voltage mode after STC3115 initialization (see *Section 3: API overview*)

The ALM_EN parameter is dedicated to the alarm. If the value of ALM_EN is set to 0, the alarm is not enabled by default. Consequently, dedicated API alarm functions have to be used to enable and configure the alarm. If the value of ALM_EN is set to 1, the alarm is enabled by default and alarm thresholds such as ALM_SOC and ALM_VBAT (see below) can be initialized.

The ALM_SOC parameter is an alarm threshold in % which is based on the state-of-charge (SOC) value. If the alarm is enabled and the REG_SOC register is below the ALM_SOC parameter, the alarm pin is driven low (active) automatically.

The ALM_VBAT parameter is an alarm threshold in mV which is based on the voltage value. If the alarm is enabled and the REG_VOLTAGE register is below the ALM_VBAT parameter, the alarm pin is driven low (active) automatically.

The RSENSE parameter is the sense resistor value in milliohms. This parameter can be a value between 5 and 50 milliohms. If there is no sense resistor, RSENSE can be set to 0 but, in this case, VMODE must be set to 1 (VM_MODE).

The APP_EOC_CURRENT parameter has to be configured with a targeted end-of charge (EOC) current in mA. Typically, the EOC current is equal to the battery capacity divided by 20. However, this value has to be set according to application charger behavior and can be different to the standard value.

The APP_CUTOFF_VOTLAGE parameter has to be configured with the application minimum voltage in mV. This value is the minimum voltage which cuts off the application. Below this voltage, the STSW-BAT001 driver reports an SOC of 0 %, the application cuts off and is thereby secured.

3 API overview

The STC3115 open source driver comes with a full API which allows the driver to be more easily used. The API is declared in the stc3115_Driver.h file and is described in the two sections below. The first of these sections deals with exchange data structures which are used to exchange information between the STC3115 layer and the upper software layer (see *Figure 1: "STC3115 software driver architecture"*). The second section describes the main functions of the API.

3.1 Exchange data structures

There are two exchange data structures:

- STC3115_ConfigData_TypeDef
- STC3115_BatteryData_TypeDef

They have to be created to exchange information between the STC3115 layer and the upper software layer (see *Figure 1: "STC3115 software driver architecture"*). The STC3115_ConfigData_TypeDef structure is filled by the STSW-BAT001 driver based on the stc3115_Driver.h file content. Its content is used to configure the STC3115. The STC3115_BatteryData_TypeDef structure is filled by the STSW-BAT001 driver and is returned to the upper software layer with an updated battery state. It contains the battery state information.

The text below shows an extract of the ConfigData code and the BatteryData code respectively.

```
/*stc3115 configuration structure ---------------------------------- */
  typedef struct {
   int Vmode;            /* 1=Voltage mode, 0=mixed mode */
   int Alm_SOC;          /* SOC alarm level in % */
   int Alm_Vbat;         /* Vbat alarm level in mV */
   int CC_cnf;           /* nominal battery CC_cnf */
   int VM_cnf;           /* nominal battery VM cnf */
   int Cnom;             /* nominal battery capacity in mAh */
   int Rsense;           /* sense resistor in mOhms */
   int RelaxCurrent;     /* relaxation current(< C/20) in mA */
   unsigned char OCVOffset[16]; /* OCV curve adjustment in 0.55mV */
} STC3115_ConfigData_TypeDef;
```

```
/*battery output structure ----------------------------------------- */
typedef struct {
 int status; /* STC3115 status registers                              */
 int HRSOC; /* battery relative SOC (%) in 1/512%                     */
 int SOC; /* battery relative SOC (%) in 0.1%                         */
 int Voltage; /* battery voltage in mV                                */
 int Current; /* battery current in mA                                */
 int Temperature; /* battery temperature in 0.1°C                     */
 int ConvCounter; /* STC3115 convertion counter in 0.5s               */
 int OCV; /* battery relax voltage in mV                              */
 int Presence; /* battery presence                                    */
 int ChargeValue; /* battery remaining capacity in mAh                */
 int RemTime; /* battery remaining operating time during discharge    */
 } STC3115_BatteryData_TypeDef;
```

3.2 Main functions of the API

Four main functions of the API are used to start, stop, and benefit from the battery monitoring features of the STC3115. In general, these functions are sufficient to manage the STC3115. The functions are described below:

int GasGauge_Initialization(STC3115_ConfigData_TypeDef*,STC3115_BatteryData_TypeDef*)

* Description: starts the STC3115 battery tracking algorithm system

* Input: algorithm parameters in the stc3115_Battery.h file

* Return: 0 means ok, -1 means the STC3115 is not found or because there is an I^2C error

* Result: initializes ConfigData and returns the early battery status in BatteryData

void GasGauge_Reset(void)

* Function name: GasGauge_Reset

* Description: resets the gas gauge system, this function must not be used frequently

* Input: none

* Return: 0 means ok, -1 means there is an I^2C error

int GasGauge_Stop(void)

* Function name: GasGauge_Stop

* Description: stops the gas gauge system and saves the context in the RAM of the STC3115.

* Input: none

* Return: 0 means ok, -1 means there is an I^2C error

int GasGauge_Task(STC3115_ConfigData_TypeDef*,STC3115_BatteryData_TypeDef*)

* Function name: GasGauge_Task

* Description: periodic gas gauge task, to be called periodically (every 1 to 60 seconds)

* Input: STC315 ConfigData structure filled by GasGauge_Initialization function

* Return: 1 means data are available, 0 means no data are available, and -1 means there is an error.

* Result: fills the BatteryData structure with the updated battery status

3.3 Additional functions of the API

Several additional functions of the API are used to manage some features of the STC3115. These functions have to be called after the GasGauge_Initialization function.

If the STC3115 is configured by default in MIXED MODE, the device can be set up in power-saving mode on the fly while the application is running. This saves energy when the application is in low power periods. The following two API functions concern power-saving mode.

int STC3115_SetPowerSavingMode(void)

* Function name: STC3115_SetPowerSavingMode

* Description: sets power-saving mode

* Input: none

* Return: error status (OK, !OK)

int STC3115_StopPowerSavingMode(void)

* Function name: STC3115_StopPowerSavingMode

* Description: stops power-saving mode

* Input: none

* Return: error status (OK, !OK)

The STC3115 provides an alarm pin which can be used to report the battery status to the application. This alarm pin can be managed by the following API functions:

int STC3115_AlarmSet(void)

* Function name: STC3115_AlarmSet

* Description: sets the alarm function

* Input: none

* Return: error status (OK, !OK)

int STC3115_AlarmStop(void)

* Function name: STC3115_AlarmStop

* Description: stops the alarm function

* Input: none

* Return: error status (OK, !OK)

int STC3115_AlarmGet(void)

* Function name: STC3115_AlarmGet

* Description: returns the alarm status

* Input: none

* Return: alarm status: 00 = no alarm, 01 = SOC alarm, 10 = voltage alarm, and 11 = SOC and voltage alarm.

int STC3115_AlarmClear(void)

* Function name: STC3115_AlarmClear

* Description: clears the alarm signal

* Input: none

* Return: error status (OK, !OK)

int STC3115_AlarmSetVoltageThreshold(STC3115_ConfigData_TypeDef*, int)

* Function name: STC3115_AlarmSetVoltageThreshold

* Description: sets the alarm threshold

* Input: internal voltage threshold in mV

* Return: error status (OK, !OK)

int STC3115_AlarmSetSOCThreshold(STC3115_ConfigData_TypeDef*, int)

* Function name: STC3115_AlarmSetSOCThreshold

* Description: sets the alarm threshold

* Input: internal SOC threshold in %

* Return: error status (OK, !OK)

4 How to use the driver

There are three basic steps to using the driver. They are

- Hardware initialization
- Software initialization
- Periodic update of the battery state

These steps are described in more detail in the three sections below.

4.1 Hardware initialization

1. Update the I2C_Write and I2C_Read functions in the stc3115_I2C.c file with the I^2C driver functions of the application.
2. Initialize the STC3115 dedicated hardware externally from the driver (e.g. the I^2C interface, GPIOs, etc).

4.2 Software initialization

1. Fill the STC3115_Battery.h file with the battery and application parameters
2. Declare one STC3115_ConfigData_TypeDef structure and one STC3115_BatteryData_TypeDef parameter structure.
3. At application start-up, call once only the GasGauge_Initialization function with un-initialized structures as parameters. This:
   - Fills the STC3115_ConfigData_TypeDef structure with STC3115_Battery.h data.
   - Fills the STC3115_BatteryData_TypeDef structure with the STC3115 algorithm results which returns the battery state to the application.

GasGauge_initialization function

- Initialize the software structures
- Check the battery history, for example, is it a new battery, a swapped battery, or is the battery already connected?
- Initialize the STC3115 registers according to the battery history
- Initialize the RAM memory of the STC3115 according to the battery history

GasGauge_initialization returns

- (-1) indicates that the STC3115 cannot be accessed. Either the I^2C bus is not properly configured or the STC3115 is not power supplied.
- (0) indicates that initialization has been successful

4.3 Periodic update of the battery state

Call the GasGauge_Task function when battery information is needed. Normally this is every 1 to 30 seconds (no frequency accuracy is needed). Transfer the STC3115_ConfigData_TypeDef structure and the STC3115_BatteryData_TypeDef structure to the GasGauge_Task function.

The GasGauge_Task function:

- Uses the STC3115_ConfigData_TypeDef structure in case the STC3115 has to be initialized again.
- Fills the STC3115_BatteryData_TypeDef structure with the STC3115 algorithm results (which returns the battery state to the application).

GasGauge_Task function

- Check the state of the STC3115, the battery, and the application
- If the battery is ok, update the current battery data
- Battery state information is returned

GasGauge_Task returns

- (-1) indicates that the STC3115 cannot be assessed. In other words, the I^2C bus is busy, the STC3115 is not power supplied, or the BATD/CD pin is in high level state.
- (0) indicates that only the battery SOC, voltage, and OCV data are available
- (1) indicates that all STC3115_BatteryData_typeDef data are available

4.4 Stopping and resetting the STC3115

1. Call the GasGauge_Stop function during the application switch off sequence. This stops the STC3115 and saves energy which, in turn, helps recover the battery context during the next GasGauge_Initialization function.
2. Do not use the GasGauge_Reset function by default. This function should only be called in abnormal use cases.

5 Code to call the API of the STC3115 open source driver

The code below is an example of how to call the API of the STC3115 open source driver to report battery information to the application.

```
#include "stc3115_Driver.h"

int main(void)
{
 int i=0;

 STC3115_ConfigData_TypeDef STC3115_ConfigData;
 STC3115_BatteryData_TypeDef STC3115_BatteryData;

 /* Initialize I2C and GPIO */
 IO_Init();

 /* Initialize Gas Gauge */
 GasGauge_Initialization(&STC3115_ConfigData, &STC3115_BatteryData);

 /* Read and print Gas Gauge values */
 while(1){
   /* Update Gas Gauge driver */
   if(i == GASGAUGE_PERIOD){
     GasGauge_Task(&STC3115_ConfigData, &STC3115_BatteryData);
     printf("Vbat: %i mV, I=%i mA SoC=%i, C=%i, P=%i A=%i , CC=%d\r\n",
                                     STC3115_BatteryData.Voltage,
                                     STC3115_BatteryData.Voltage,
                                     STC3115_BatteryData.Current,
                                     STC3115_BatteryData.SOC,
                                     STC3115_BatteryData.ChargeValue,
                                     STC3115_BatteryData.Presence,
                                     STC3115_BatteryData.status >> 13,
                                     STC3115_BatteryData.ConvCounter);
     i=0;
     }
   i++;
   }
 Return 0;
}
```

6 Related documentation

- STC3115 datasheet: Gas gauge IC with alarm output for handheld applications
- STSW-BAT001 data brief: STC3115 open source driver
- AN4324: STC3115 system integration

7 Revision history

Table 1: Document revision history

Date	Revision	Changes
10-Dec-2014	1	Initial release.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.