AN13095

Smart Battery Charger by LPC845 with SMBus Interface

Rev. 0 — January 5, 2021

Application Note

by: NXP Semiconductors

1 Introduction

Batteries are used everywhere, such as smart phones, notebook computers, wearable devices, handheld electronic products, smart small appliances, etc. Users always want to know the battery temperature, voltage, current, capacity, how long it can be fully charged, and how long the battery will be exhausted. During the charging process, it is very important to ensure the safety of battery charging and provide a smooth and controllable charging curve. The above requirements are expected to be realized by a smart charger. A smart charging solution implemented with LPC845 is recommended.

Based on the Arm® Cortex®-M0+ core, LPC84x is a low-cost, 32-bit MCU family operating at frequencies of up to 30 MHz. The LPC84x MCU family supports up to 64 KB of flash memory and 16 KB of SRAM. This family features exceptional power efficiency in the low-current mode using the FRO as the clock source. The peripheral complement of the LPC84x MCU family includes:

- One CRC engine
- Four I2C-bus interfaces
- Up to five UARTs
- Up to two SPI interfaces
- One multi-rate timer
- Self-wake-up timer
- SCTimer/PWM
- One general purpose 32-bit counter/timer
- One DMA
- One 12-bit ADC
- Two 10-bit DACs
- One analog comparator
- Function-configurable I/O ports through a switch matrix
- One input pattern match engine
- Up to 54 general-purpose I/O pins

Contents

2 Overview

2.1 Function block diagram

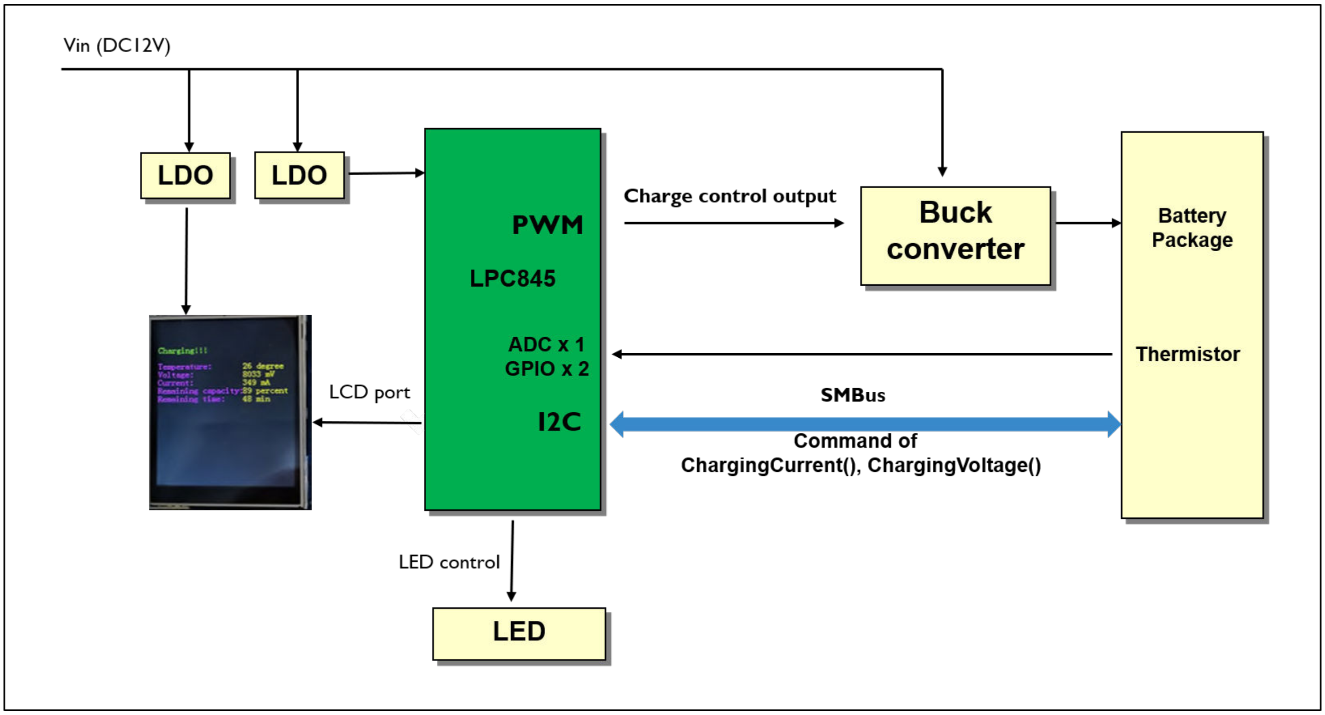
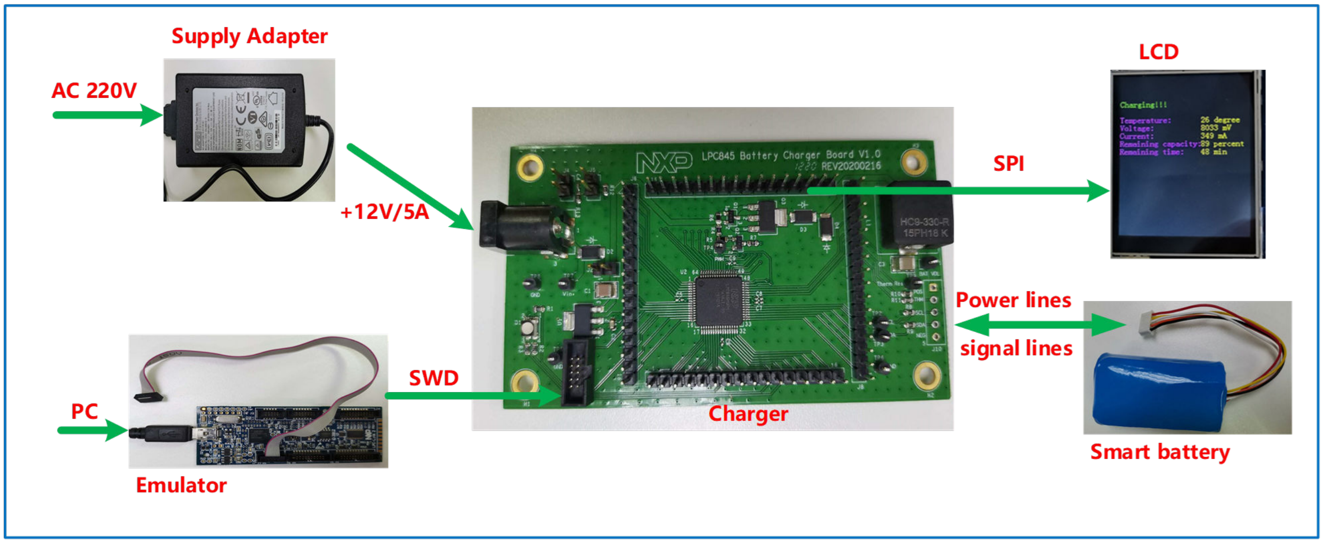


Figure 1. Function block diagram

LPC845 periodically communicates with smart battery through SMBUS bus to obtain battery information and dynamically controls PWM output to adjust charging voltage. At the same time, the charging status is displayed through LEDs and the charging information is displayed through the LCD screen.

2.2 System diagram

Figure 2 shows the system diagram, including NXP LPC845 battery charger board, +12 Adapter, smart batter, LCD, and emulator.



```
Charging!!!

Temperature:            26 degree
Voltage:                8033 mV
Current:                349 mA
Remaining capacity:89 percent
Remaining time:         48 min
```

Figure 2. System diagram

The system can charge smart batteries with a nominal voltage of 8.4 V. During the charging process, the battery voltage, current, temperature, and charging time are displayed on the LCD screen in real time. In the process of developing the program, we use LPC-link2 emulator to download it to the MCU, you can use any other emulators with 1.27' 10-pin SWD connector such as J-Link, U-Link, etc. Of course, the charging sequence can also be drawn in real time through the FreeMASTER software. In the charging process, it has gone through four stages: pre-charging, constant current charging, constant voltage charging, and charging full.

3 Hardware

3.1 Hardware overview

Figure 3 shows a complete demonstration system, including charger board, LCD, smart battery, and power adapter.

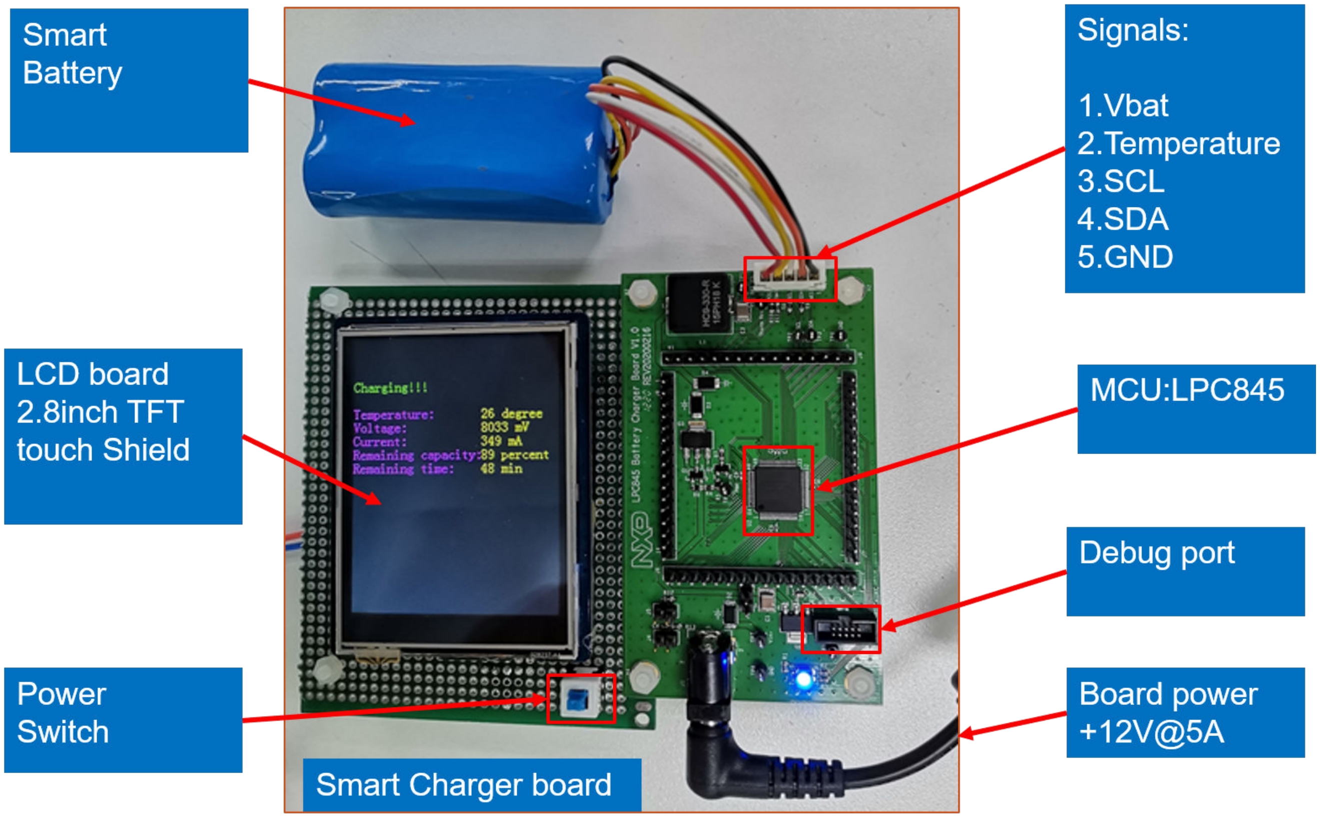


Figure 3. Hardware overview

Use 12 V and 5 A adapter as power supply. It provides the power supply for the charger board and voltage and current source for the charging battery. For the voltage source for battery charging, the standard voltage of the smart battery is 8.4 V and the LCD is a 320 × 240 resolution TFT screen.

3.2 System connection

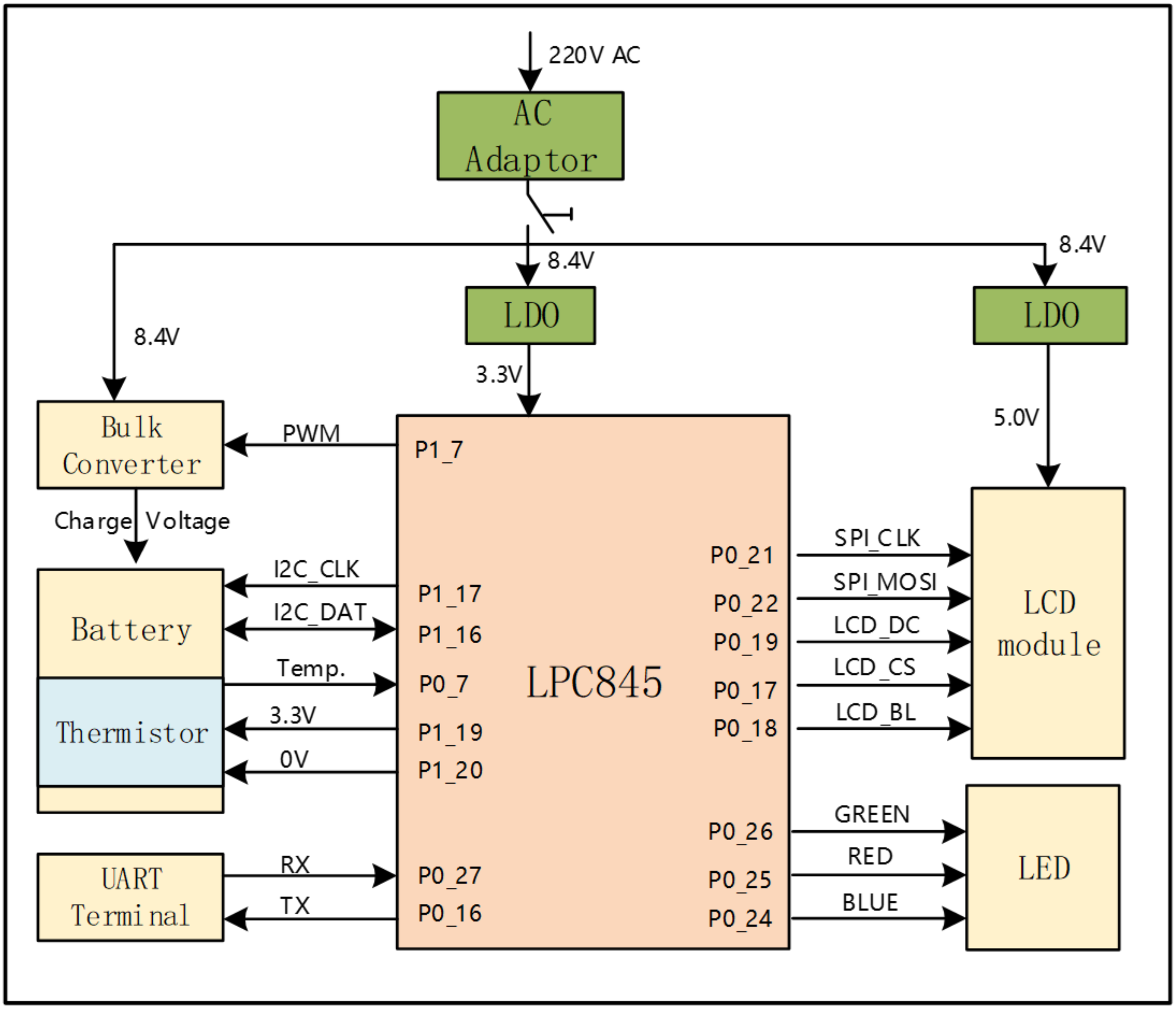


Figure 4. System connection

3.3 Charger board

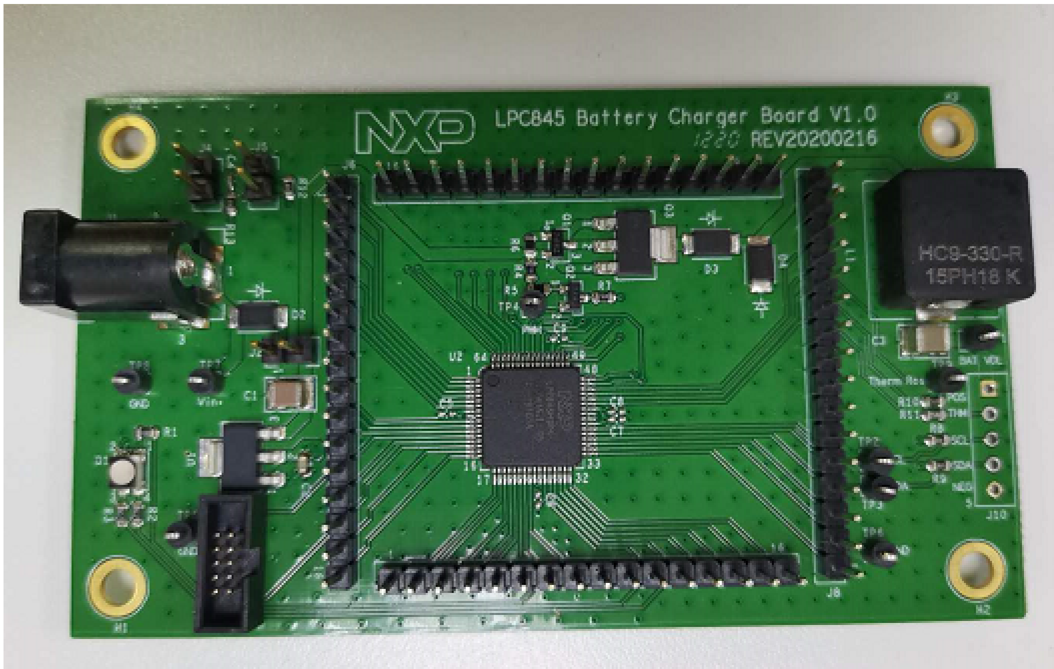


Figure 5. Charger board

The charger board includes the following modules:

- +12 V power socket: Connecting the 12 V power adapter
- Total 64 pins of MCU are led out: Providing signals such as SPI to drive LCD screen
- Smart battery interface: Connecting the smart battery
- Emulator interface: Connecting the MCU debugger
- Onboard LDO: Providing the 3.3 V power
- Board bulk circuit: Providing the adjustable voltage to the battery
- MCU controller: LPC845

3.4 Smart battery

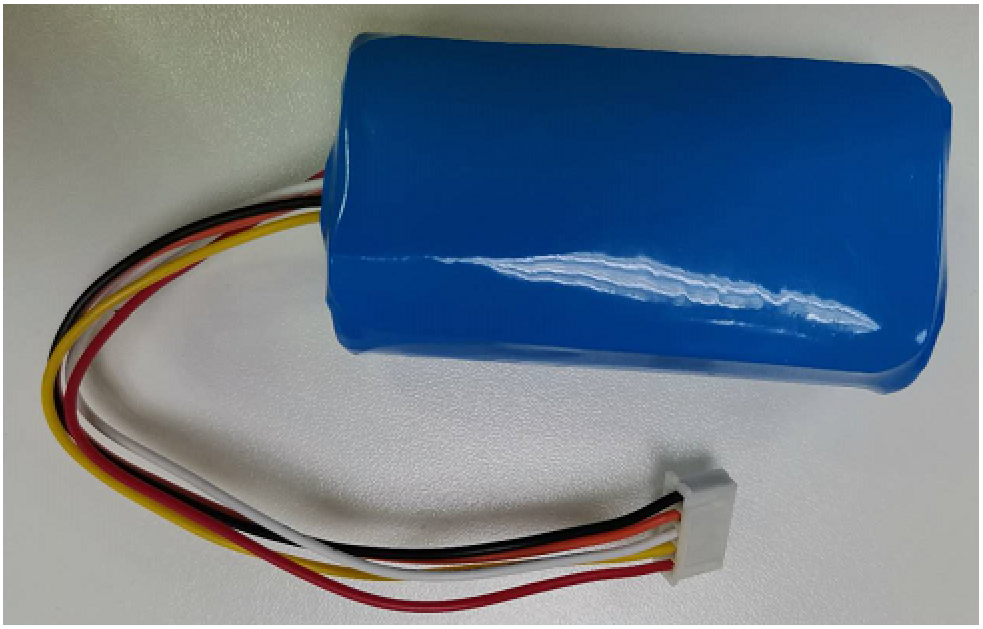


Figure 6. Smart battery

The battery pack is a smart battery, consisting of two Li-ion batteries with a standard voltage of 8.4 V. The battery package contains a Li-Ion battery pack manager chip named **bq40z50**. The battery pack supports Two-Wire SMBus v1.1 interface to communicate with the MCU. In addition, the battery pack contains a PTC thermistor of 10 k value and the MCU can sample the voltage through the ADC to calculate its temperature.

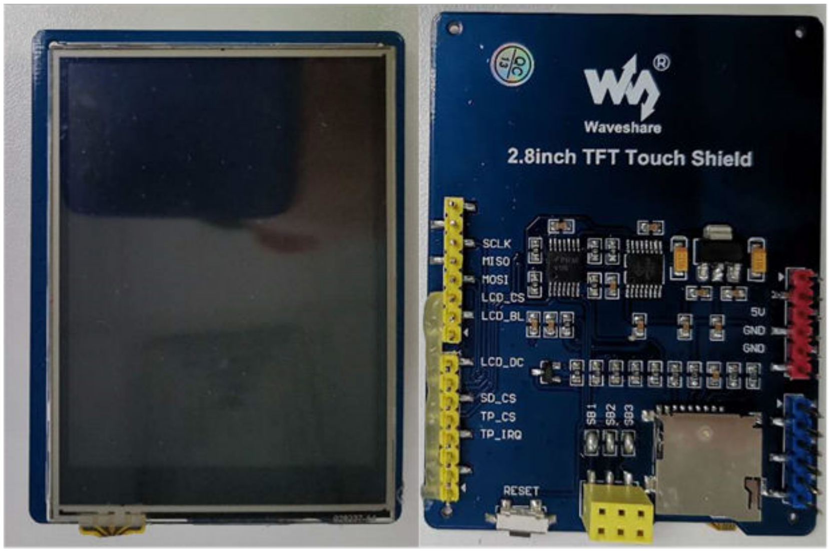


Figure 7. PTC thermistor

3.5 LCD display board

LCD board is from Waveshare, one 2.8-inch Touch LCD Shield for Arduino.

Its features are as below:

- Resistive touch screen TFT LCD, 2.8 inch, 320 × 240 resolution
- Standard Arduino interface, compatible with development boards, such as: Arduino UNO
- Controlled via SPI, only a few Arduino pins are used

3.6 Emulator

Jointly developed by NXP and Embedded Artists, the LPC-Link 2 is an extensible and stand-alone debug probe that can be configured to support various development tools and IDEs with a variety of different downloadable firmware images. It can also be used as an evaluation board in its own right for the NXP LPC4370 triple core MCU.

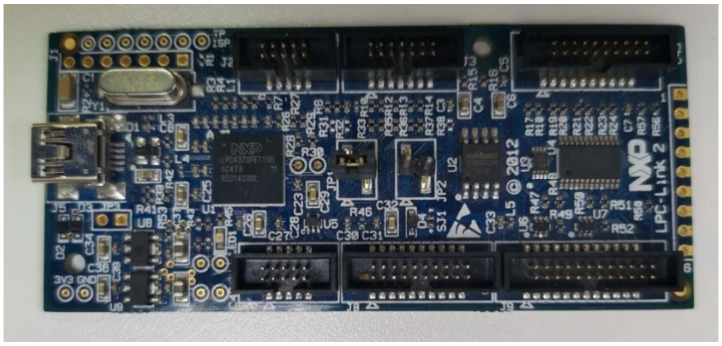


Figure 8. Emulator

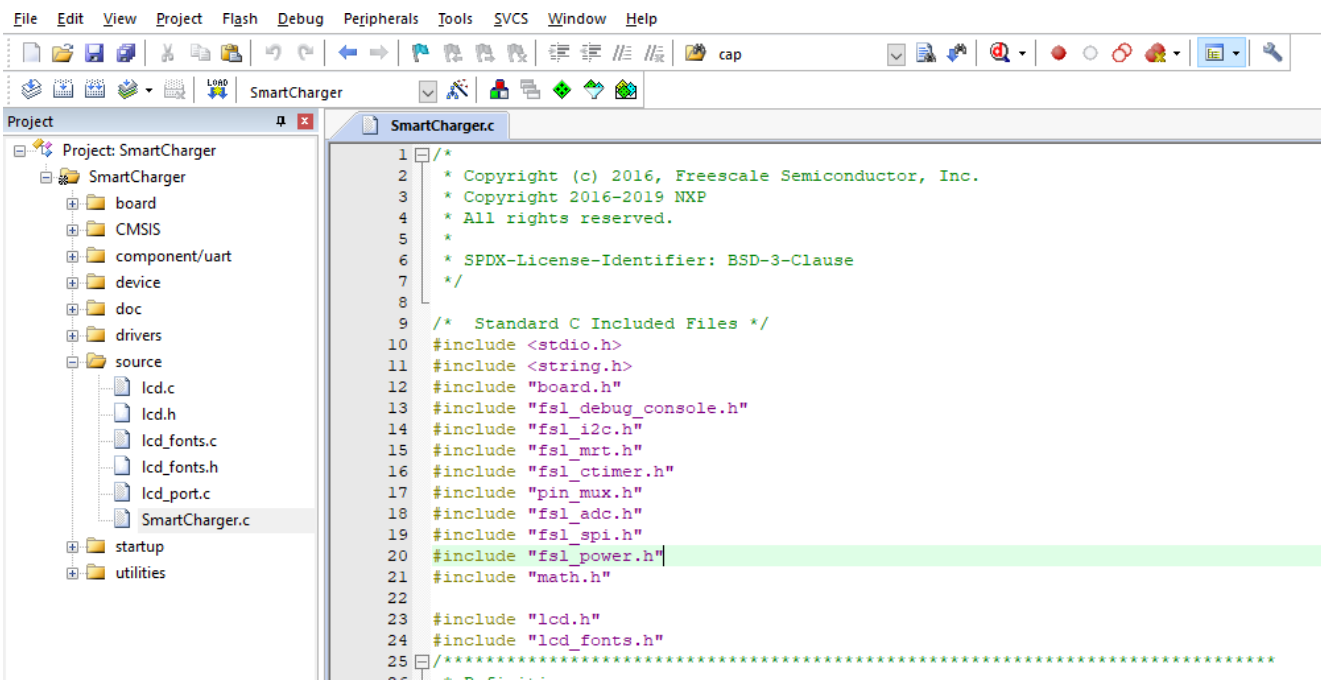
3.7 AC adaptor

The adapter converts 220 V AC to 12 V 5A DC power to provide power for the board and battery. Readers can use other DC power supply instead.

Figure 9. AC adaptor

4 Software

4.1 Source code



```
/*
 * Copyright (c) 2016, Freescale Semiconductor, Inc.
 * Copyright 2016-2019 NXP
 * All rights reserved.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 */

/*  Standard C Included Files */
#include <stdio.h>
#include <string.h>
#include "board.h"
#include "fsl_debug_console.h"
#include "fsl_i2c.h"
#include "fsl_mrt.h"
#include "fsl_ctimer.h"
#include "pin_mux.h"
#include "fsl_adc.h"
#include "fsl_spi.h"
#include "fsl_power.h"
#include "math.h"

#include "lcd.h"
#include "lcd_fonts.h"
/**********************************************************************
```

Figure 10. Source code

The code is developed on the MDK Keil5.28 IDE platform and uses the latest official SDK software development kit (SDK2.7.0). According to functional requirements, the peripherals include UART, SPI, I2C, Ctimer, Multimer, ADC.

Their functions are as below:

- UART is used to drive serial output.
- SPI is used to drive the LCD screen.
- I^{2}C is responsible for communicating with the battery pack.
- Ctimer generates PWM wave to control bulk circuit.
- Multimer generates periodic interrupts for periodic sampling and modulation.
- ADC is used to collect thermistor voltage and calculate temperature.

4.2 Program flow chart

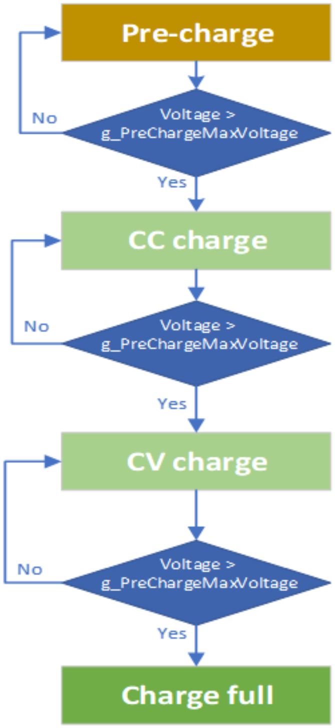

Pre-charge
Voltage > g_PreChargeMaxVoltage
No
Yes
CC charge
Voltage > g_PreChargeMaxVoltage
No
Yes
CV charge
Voltage > g_PreChargeMaxVoltage
No
Yes
Charge full

Figure 11. Program flow chart

The battery charging process includes four stages:

- Pre-charging stage

  The pre-charging phase is to prevent battery damage due to sinking a large current when the voltage is too low. Only when it exceeds a certain voltage, it enters the constant current charging.

- Constant current stage

  Constant current charging is the main stage of the battery. The battery obtains most energy in the constant current stage. At this stage, the battery voltage continues to rise.

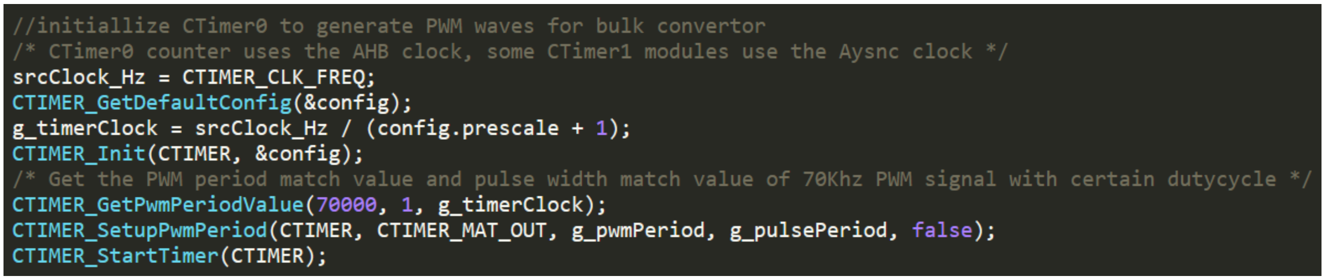
- Constant voltage stage

  When a relatively high voltage is reached, it enters the constant voltage charging stage. The constant voltage stage is the end work, when the battery is close to full capacity. At this time, it is not appropriate to provide a large current but a stable voltage instead. The charging current continues to decrease.

- Charging full

  When the current reaches a small value, it can be considered that the battery is fully charged.

4.3 PWM generation



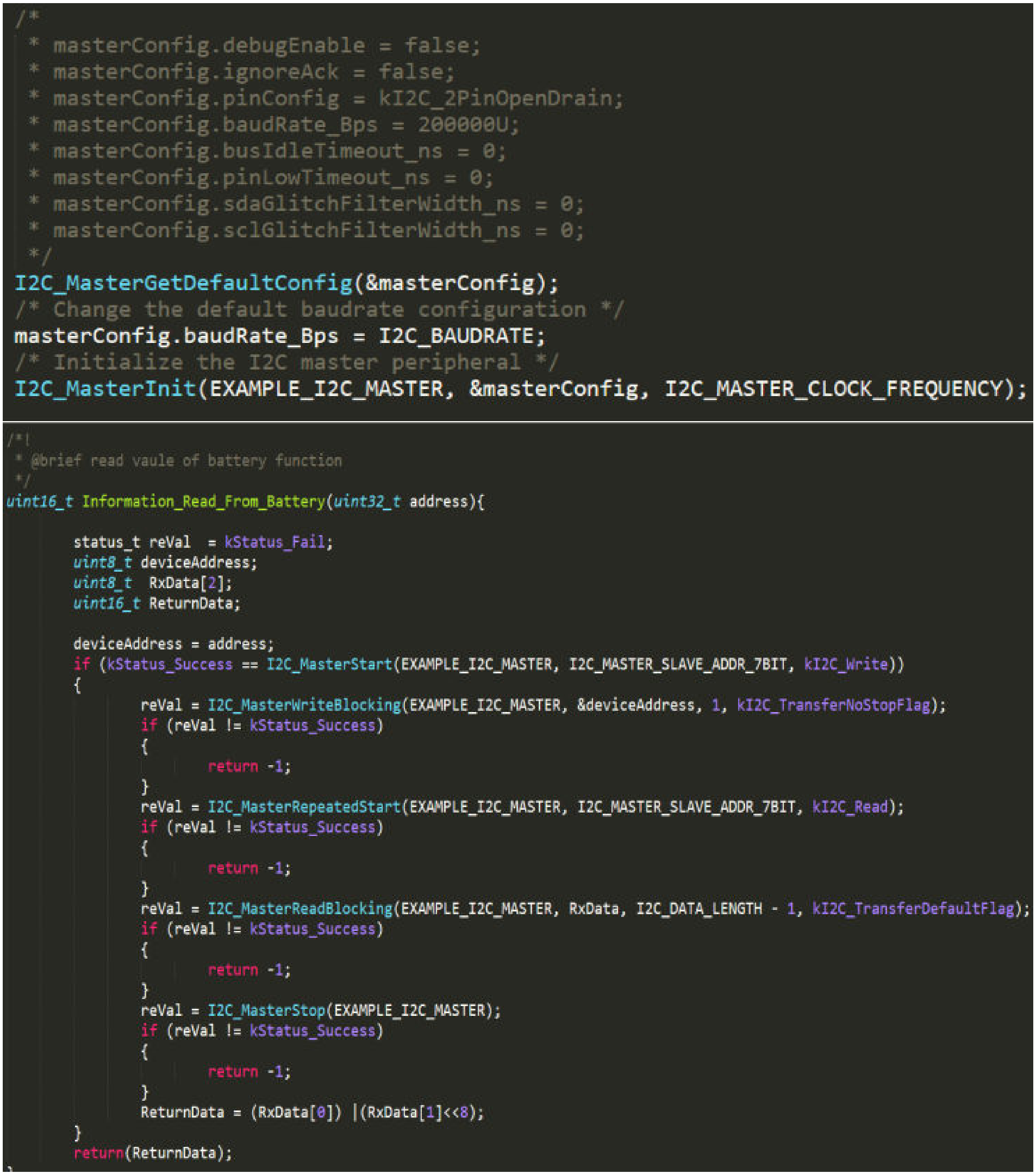
```
//initiallize CTimer0 to generate PWM waves for bulk convertor
/* CTimer0 counter uses the AHB clock, some CTimer1 modules use the Aysnc clock */
srcClock_Hz = CTIMER_CLK_FREQ;
CTIMER_GetDefaultConfig(&config);
g_timerClock = srcClock_Hz / (config.prescale + 1);
CTIMER_Init(CTIMER, &config);
/* Get the PWM period match value and pulse width match value of 70Khz PWM signal with certain dutycycle */
CTIMER_GetPwmPeriodValue(70000, 1, g_timerClock);
CTIMER_SetupPwmPeriod(CTIMER, CTIMER_MAT_OUT, g_pwmPeriod, g_pulsePeriod, false);
CTIMER_StartTimer(CTIMER);
```

Figure 12. PWM generation

PWM is used to adjust the output voltage of the bulk circuit and generated by Ctimer.

The counter/timer block is designed to count cycles of the APB bus clock or an externally supplied clock and can optionally generate pulse width modulator via match outputs. In this application, the 15 MHz APB clock is used as the clock source for Ctimer. The frequency of PWM is 70 kHz. The minimum duty cycle is five hundredths. Generate different charging voltages and charging currents by generating PWM waves with different duty cycles.

4.4 SMbus communication



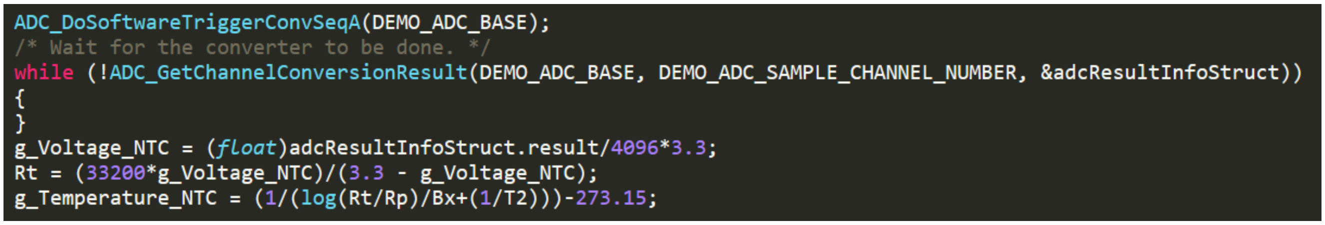
```
/*
 * masterConfig.debugEnable = false;
 * masterConfig.ignoreAck = false;
 * masterConfig.pinConfig = kI2C_2PinOpenDrain;
 * masterConfig.baudRate_Bps = 200000U;
 * masterConfig.busIdleTimeout_ns = 0;
 * masterConfig.pinLowTimeout_ns = 0;
 * masterConfig.sdaGlitchFilterWidth_ns = 0;
 * masterConfig.sclGlitchFilterWidth_ns = 0;
 */
I2C_MasterGetDefaultConfig(&masterConfig);
/* Change the default baudrate configuration */
masterConfig.baudRate_Bps = I2C_BAUDRATE;
/* Initialize the I2C master peripheral */
I2C_MasterInit(EXAMPLE_I2C_MASTER, &masterConfig, I2C_MASTER_CLOCK_FREQUENCY);
```

```
/*!
 * @brief read vaule of battery function
 */
uint16_t Information_Read_From_Battery(uint32_t address){

        status_t reVal  = kStatus_Fail;
        uint8_t deviceAddress;
        uint8_t  RxData[2];
        uint16_t ReturnData;

        deviceAddress = address;
        if (kStatus_Success == I2C_MasterStart(EXAMPLE_I2C_MASTER, I2C_MASTER_SLAVE_ADDR_7BIT, kI2C_Write))
        {
                reVal = I2C_MasterWriteBlocking(EXAMPLE_I2C_MASTER, &deviceAddress, 1, kI2C_TransferNoStopFlag);
                if (reVal != kStatus_Success)
                {
                        return -1;
                }
                reVal = I2C_MasterRepeatedStart(EXAMPLE_I2C_MASTER, I2C_MASTER_SLAVE_ADDR_7BIT, kI2C_Read);
                if (reVal != kStatus_Success)
                {
                        return -1;
                }
                reVal = I2C_MasterReadBlocking(EXAMPLE_I2C_MASTER, RxData, I2C_DATA_LENGTH - 1, kI2C_TransferDefaultFlag);
                if (reVal != kStatus_Success)
                {
                        return -1;
                }
                reVal = I2C_MasterStop(EXAMPLE_I2C_MASTER);
                if (reVal != kStatus_Success)
                {
                        return -1;
                }
                ReturnData = (RxData[0]) |(RxData[1]<<8);
        }
        return(ReturnData);
```

Figure 13. SMbus communication

There are four I^{2}C modules in the LPC845 which support SMBus. **I2C1** is selected for the communication bus between the MCU and the battery. Read the value of the battery-related register from the given device address to get the charging information such as voltage, current, temperature, etc.

4.5 Temperature sample



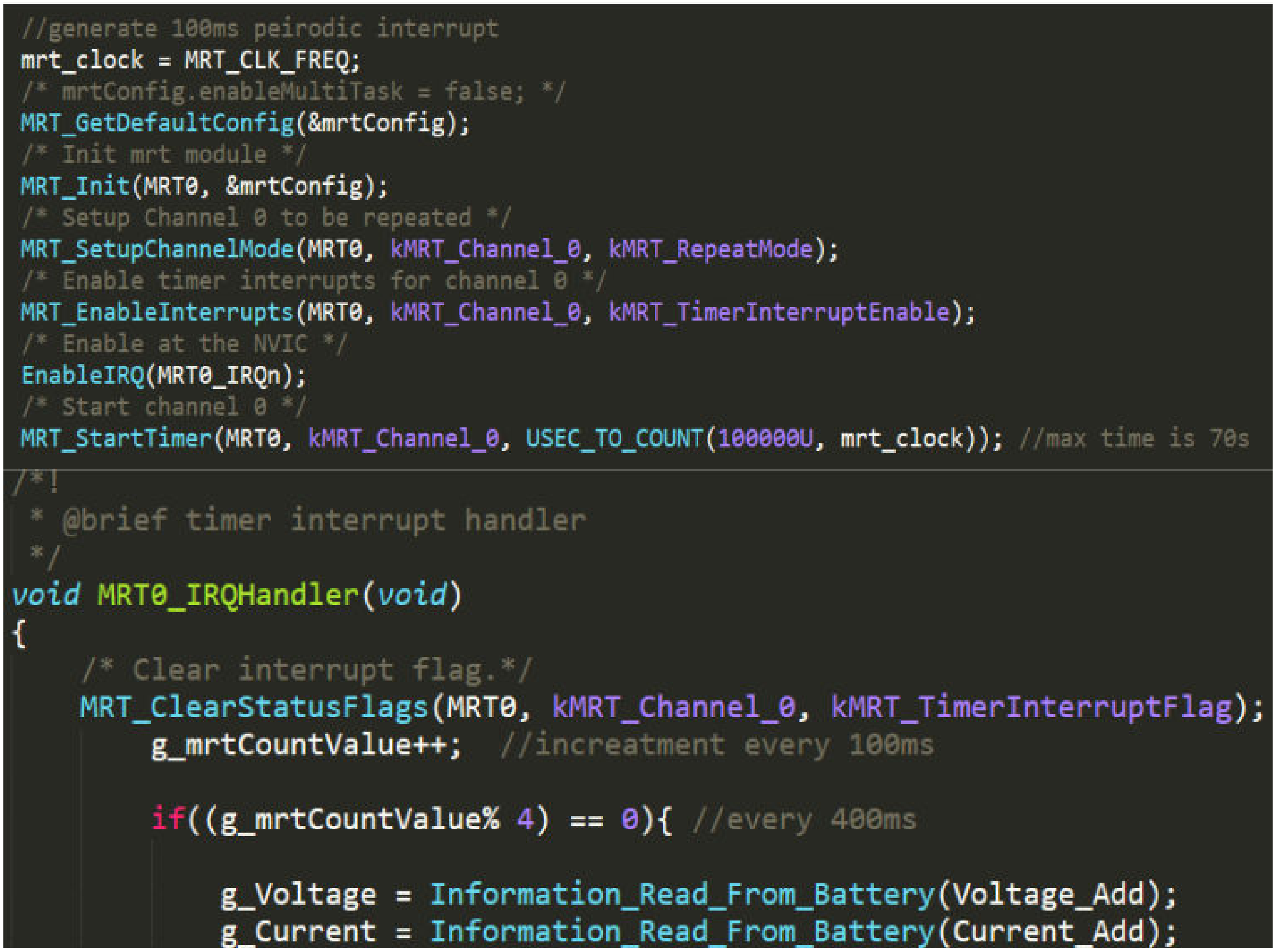
```
ADC_DoSoftwareTriggerConvSeqA(DEMO_ADC_BASE);
/* Wait for the converter to be done. */
while (!ADC_GetChannelConversionResult(DEMO_ADC_BASE, DEMO_ADC_SAMPLE_CHANNEL_NUMBER, &adcResultInfoStruct))
{
}
g_Voltage_NTC = (float)adcResultInfoStruct.result/4096*3.3;
Rt = (33200*g_Voltage_NTC)/(3.3 - g_Voltage_NTC);
g_Temperature_NTC = (1/(log(Rt/Rp)/Bx+(1/T2)))-273.15;
```

Figure 14. Temperature

GPIO outputs a high voltage to supply power for thermistor and then, according the voltage on thermistor, calculates the actual temperature of smart battery. If the temperature changes, the resistor value of thermistor changes as well. And the voltage on thermistor changes. The change can be captured by ADC sampling on MCU.

In LPC845, one 12-bit ADC with up to 12 input channels with sample rates of up to 1.2 Msamples/s. ADC channel 0 is responsible for sampling the thermistor voltage and the corresponding temperature is calculated. Once an over-temperature situation occurs, turn off the PWM signal.

4.6 Cycle interrupt

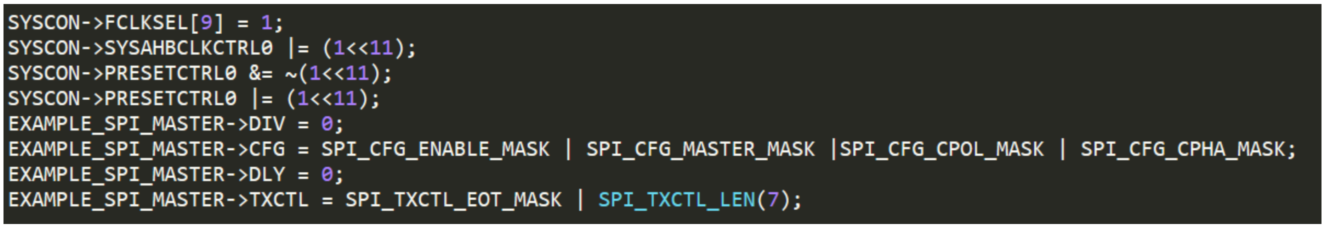


```
//generate 100ms peirodic interrupt
mrt_clock = MRT_CLK_FREQ;
/* mrtConfig.enableMultiTask = false; */
MRT_GetDefaultConfig(&mrtConfig);
/* Init mrt module */
MRT_Init(MRT0, &mrtConfig);
/* Setup Channel 0 to be repeated */
MRT_SetupChannelMode(MRT0, kMRT_Channel_0, kMRT_RepeatMode);
/* Enable timer interrupts for channel 0 */
MRT_EnableInterrupts(MRT0, kMRT_Channel_0, kMRT_TimerInterruptEnable);
/* Enable at the NVIC */
EnableIRQ(MRT0_IRQn);
/* Start channel 0 */
MRT_StartTimer(MRT0, kMRT_Channel_0, USEC_TO_COUNT(100000U, mrt_clock)); //max time is 70s
/*!
 * @brief timer interrupt handler
 */
void MRT0_IRQHandler(void)
{
    /* Clear interrupt flag.*/
    MRT_ClearStatusFlags(MRT0, kMRT_Channel_0, kMRT_TimerInterruptFlag);
        g_mrtCountValue++;  //increatment every 100ms

        if((g_mrtCountValue% 4) == 0){ //every 400ms

            g_Voltage = Information_Read_From_Battery(Voltage_Add);
            g_Current = Information_Read_From_Battery(Current_Add);
```

Figure 15. Cycle interrupt

In LPC845, four channels Multi-Rate Timer (MRT) for repetitive interrupt generation at up to four programmable, fixed rates. Through the configuration, a polling task can be made every 200 ms. Check the charging status and make related PWM signal adjustments, this behavior will run through all stages of charging.

4.7 LCD driver



```
SYSCON->FCLKSEL[9] = 1;
SYSCON->SYSAHBCLKCTRL0 |= (1<<11);
SYSCON->PRESETCTRL0 &= ~(1<<11);
SYSCON->PRESETCTRL0 |= (1<<11);
EXAMPLE_SPI_MASTER->DIV = 0;
EXAMPLE_SPI_MASTER->CFG = SPI_CFG_ENABLE_MASK | SPI_CFG_MASTER_MASK |SPI_CFG_CPOL_MASK | SPI_CFG_CPHA_MASK;
EXAMPLE_SPI_MASTER->DLY = 0;
EXAMPLE_SPI_MASTER->TXCTL = SPI_TXCTL_EOT_MASK | SPI_TXCTL_LEN(7);
```

Figure 16. LCD driver

There are two SPI modules on the LPC845, and the 30 MHz clock can be used as the baud rate clock. Excluding delays introduced by external device and PCB, the maximum supported bit rate for SPI master mode is 30 Mbit/s. Such a fast speed can effectively improve the refresh rate of the screen.

4.8 FreeMASTER

FreeMASTER is a user-friendly real-time debug monitor and data visualization tool that enables runtime configuration and tuning of embedded software applications. This application note provides the information of system will display through FreeMASTER, and the setting of charging stages can be made in the FreeMASTER.

The FreeMASTER install package can be downloaded from NXP website.

Display information include:

- Charging voltage
- Charging current
- Remaining capacity of battery
- Remaining charge time

Setting:

- Cross voltage from pre-charge to CC charge stage
- Cross voltage from CC charge to CV charge stage
- Cross current from CV charge to charge full stage

The charging voltage, current, and remaining capacity of battery are as shown in Figure 17.

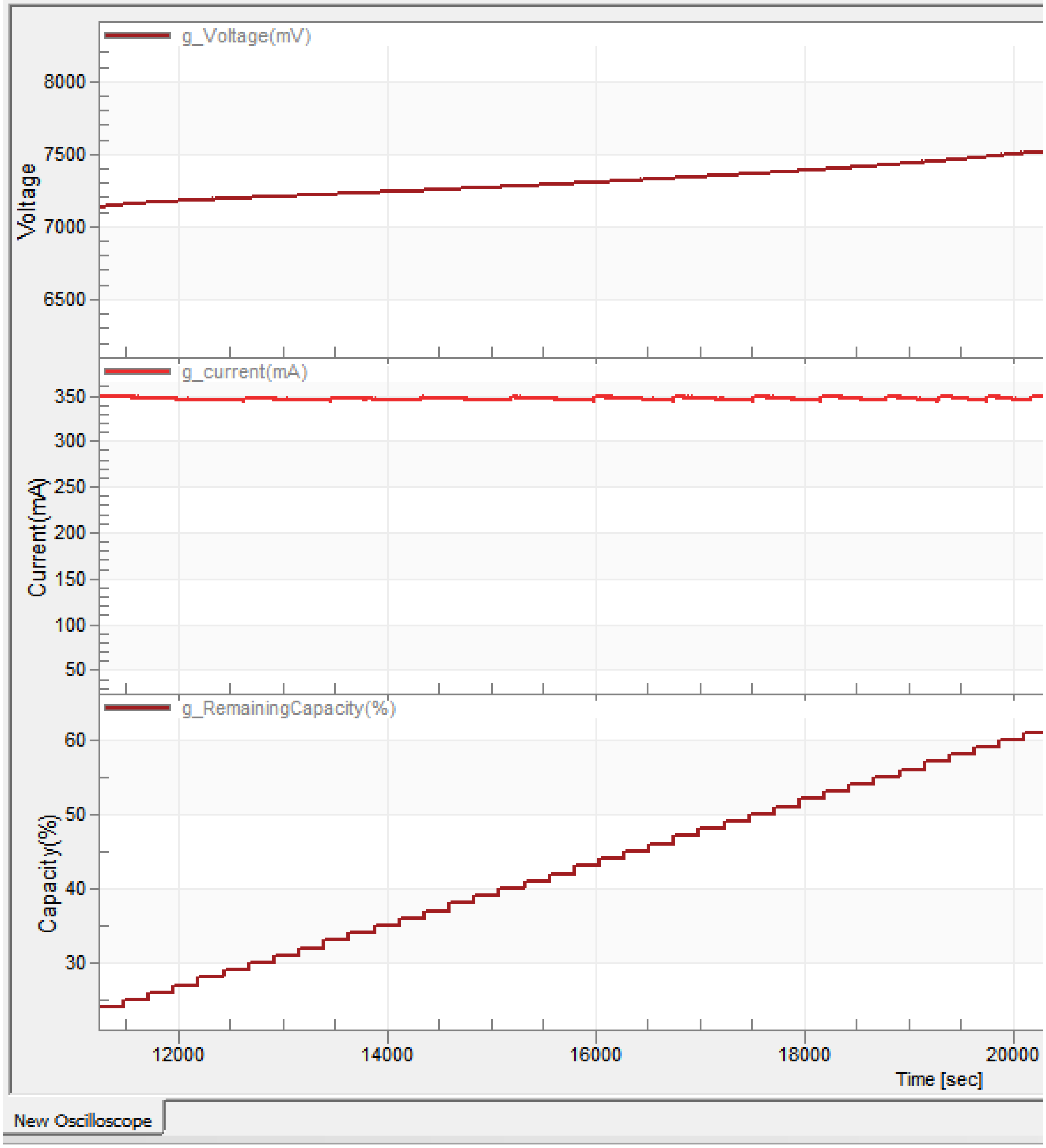
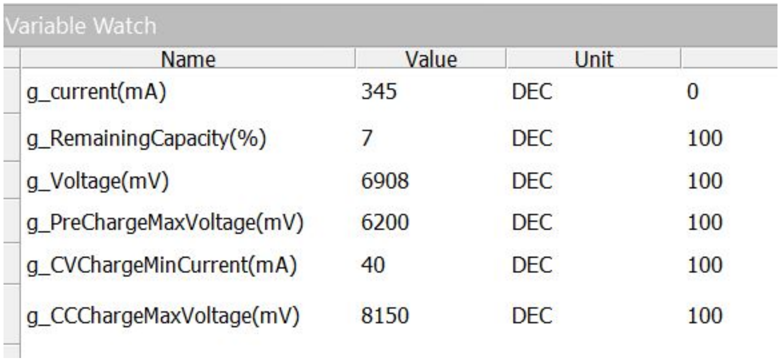


Figure 17. FreeMaster

The variables are as shown in Figure 18.



Variable Watch

Name	Value	Unit	
g_current(mA)	345	DEC	0
g_RemainingCapacity(%)	7	DEC	100
g_Voltage(mV)	6908	DEC	100
g_PreChargeMaxVoltage(mV)	6200	DEC	100
g_CVChargeMinCurrent(mA)	40	DEC	100
g_CCChargeMaxVoltage(mV)	8150	DEC	100

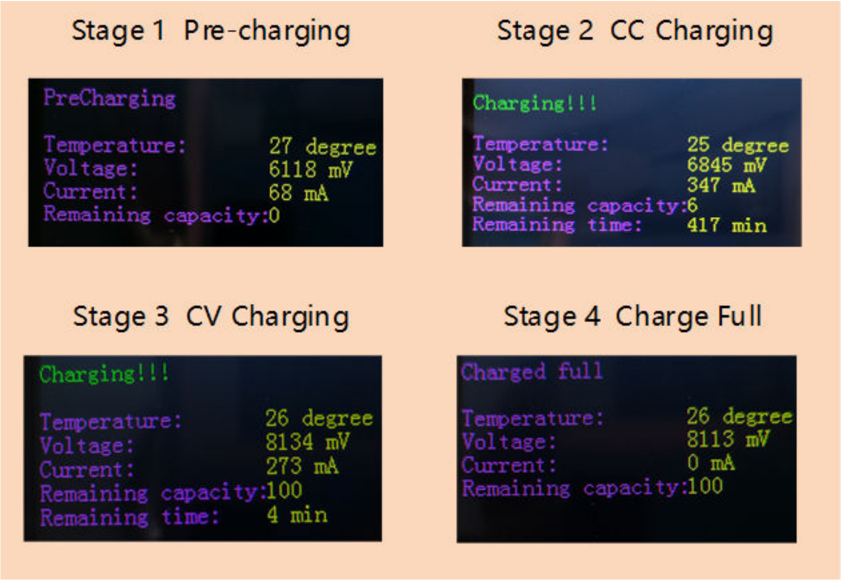
Figure 18. Variables

NOTE

The values of variables, `g_PreChargeMaxVoltage`, `g_CCChargeMaxVoltage`, `g_CVChargeMinCurrent`, can be modified by users. The values of variables, `g_Voltage`, `g_Current`, `g_RemainingCapacity`, can only be read by user.

5 Step-by-step demo

1. Connect the adapter, LCD, emulator with charger board.
2. Push down the power switch, comply and download the code into MCU.
3. Push up the power switch and connect battery with charge board.
4. If the battery voltage is lower than `g_PreChargeMaxVoltage`, it enters the pre-charge mode. When the battery voltage crosses `g_PreChargeMaxVoltage`, it enters the constant current charging mode. The constant voltage charging mode starts when the battery voltage reaches `g_CCChargeMaxVoltage`. Then the voltage is maintained at `g_CCChargeMaxVoltage` until the current drops to `g_CVChargeMinCurrent`, charging is completed. The states of these four charging stages are as shown in Figure 19.



Stage 1 Pre-charging

```
PreCharging

Temperature:          27 degree
Voltage:              6118 mV
Current:              68 mA
Remaining capacity:0
```

Stage 2 CC Charging

```
Charging!!!

Temperature:          25 degree
Voltage:              6845 mV
Current:              347 mA
Remaining capacity:6
Remaining time:       417 min
```

Stage 3 CV Charging

```
Charging!!!

Temperature:          26 degree
Voltage:              8134 mV
Current:              273 mA
Remaining capacity:100
Remaining time:       4 min
```

Stage 4 Charge Full

```
Charged full

Temperature:          26 degree
Voltage:              8113 mV
Current:              0 mA
Remaining capacity:100
```

Figure 19. State of Charging stages

6 Charging timing and specifications

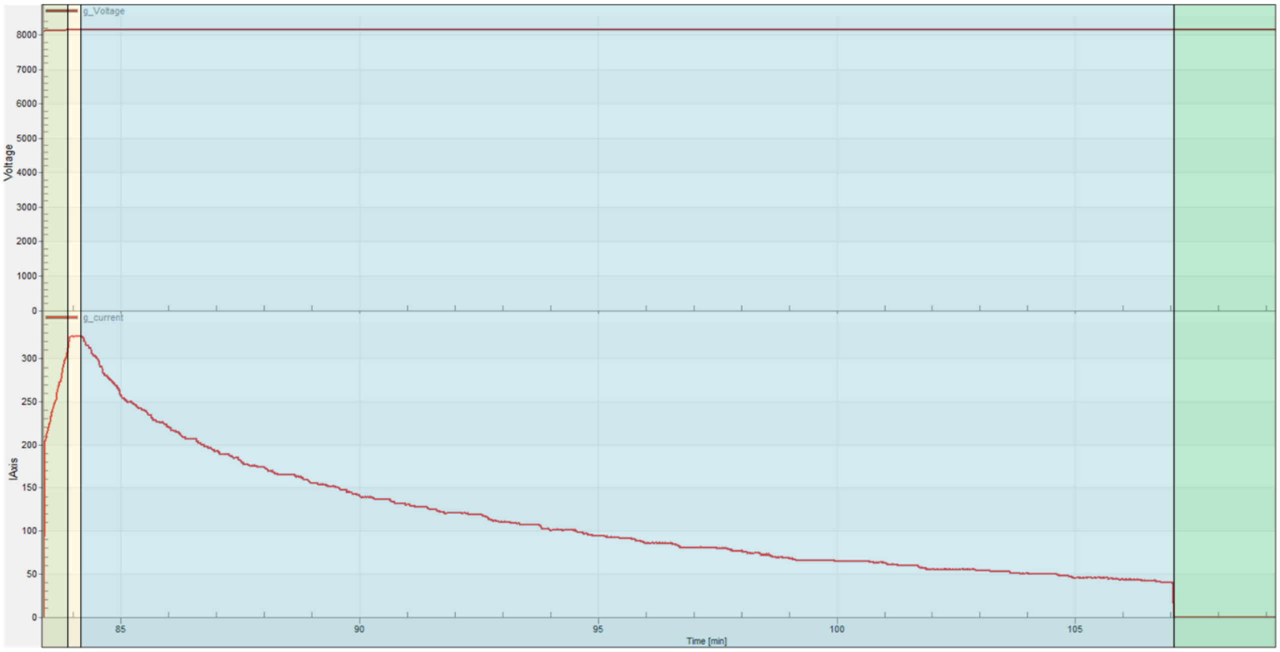


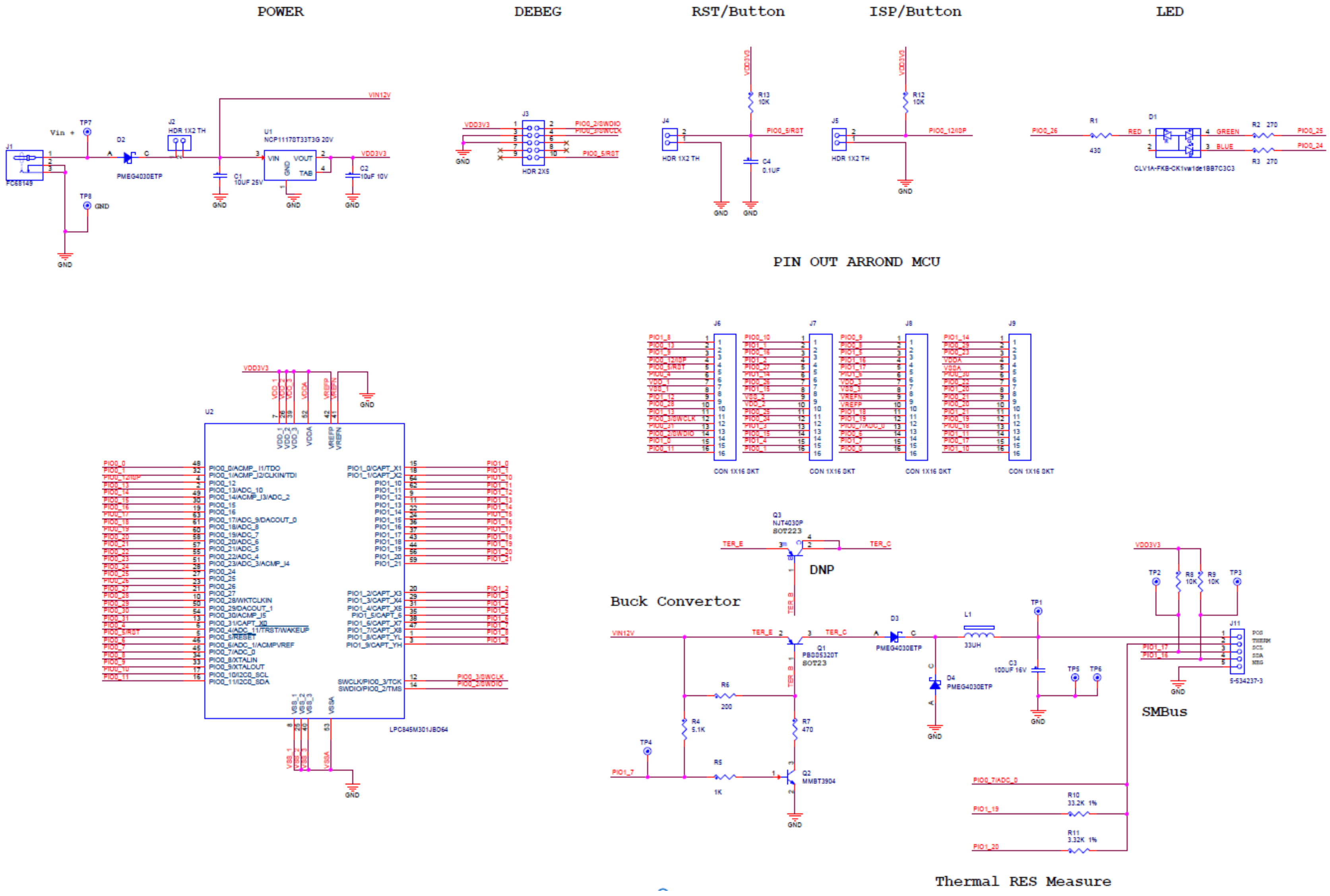
Figure 20. Charging timing and specifications

Figure 20 shows the real charging voltage and current curve captured from FreeMASTER. During the entire charging process, the battery capacity ranges from 94% to 100%. From left to right, there are four stages including pre-charging, constant current charging, constant voltage charging, and charging completion. They are marked with four colors.

Specification:

1. The range of charging voltage: 6 - 8.4 V
2. Charging voltage (CV charging mode): Set by `g_CCChargeMaxVoltage`, 8.15 V by default.
3. The range of charging current: 0 - 385 mA.
4. Charging current (CC charging mode): 350 mA
5. Charging time from empty to full: about seven hours.
6. Over charging voltage alert: 8.5 V.
7. Over charging current alert: 500 mA.
8. Over temperature alert: 50 degree.

A Schematic



J6 pin	J6 signal	J7 pin	J7 signal	J8 pin	J8 signal	J9 pin	J9 signal
1	PIO1_8	1	PIO0_10	1	PIO0_9	1	PIO1_14
2	PIO0_13	2	PIO1_1	2	PIO0_8	2	PIO0_29
3	PIO1_9	3	PIO0_16	3	PIO1_5	3	PIO0_23
4	PIO0_12/ISP	4	PIO1_2	4	PIO1_16	4	VDDA
5	PIO0_5/RST	5	PIO0_27	5	PIO1_17	5	VSSA
6	PIO0_4	6	PIO1_14	6	PIO1_6	6	PIO0_30
7	VDD_1	7	PIO0_26	7	VDD_3	7	PIO0_22
8	VSS_1	8	PIO1_15	8	VSS_3	8	PIO1_20
9	PIO1_12	9	VSS_2	9	VREFN	9	PIO0_21
10	PIO0_28	10	VDD_2	10	VREFP	10	PIO0_20
11	PIO1_13	11	PIO0_25	11	PIO1_18	11	PIO1_21
12	PIO0_3/SWCLK	12	PIO0_24	12	PIO1_19	12	PIO0_19
13	PIO0_31	13	PIO1_3	13	PIO0_7/ADC_0	13	PIO0_18
14	PIO0_2/SWDIO	14	PIO0_15	14	PIO0_6	14	PIO1_11
15	PIO1_0	15	PIO1_4	15	PIO1_7	15	PIO0_17
16	PIO0_11	16	PIO0_1	16	PIO0_0	16	PIO1_10

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: January 5, 2021
Document identifier: AN13095