Application Report
SLAA476B–February 2011–Revised July 2019

Wide-Vin Battery Charger Using SMBus Communication Interface Between MSP430™ MCUs and bq Fuel Gauges

Abhishek A. Joshi — *MSP430 Systems Solutions*
Keith J. Keller — *Analog Field Applications*

ABSTRACT

This application report describes a smart-battery charger reference design with a wide-input-voltage range. The reference design implements the System Management Bus (SMBus) protocol for communication between the MSP430™ microcontroller and a SMBus-compatible battery fuel gauge. The MSP430 MCU interrogates the fuel gauge for voltage, current, and other parameters through SMBus. The MCU then adjusts the pulse width modulation (PWM) duty-cycle output signals that are fed to the DC/DC converter to deliver the power requested by the battery.

Hardware schematic diagrams, software source code, and other information can be downloaded from http://www.ti.com/lit/zip/slaa476.

NOTE: While the software has been designed for use with the MSP430F550x family of microcontrollers, it can be ported to other MSP430 MCUs with minor modifications. The charger scheme demonstrated in this application report is specific to a Li-ion and Li-polymer battery chemistry. However, the overall battery charging concept is applicable to any type of battery chemistry.

Contents

List of Figures

List of Tables

Trademarks

MSP430, Code Composer Studio, eZ430-Chronos are trademarks of Texas Instruments.
IAR Embedded Workbench is a trademark of IAR Systems.
All other trademarks are the property of their respective owners.

1 Introduction

Smart-battery fuel gauges made by Texas Instruments (TI), such as the bq20Zxx, bq78PLxxx, bq2060A, and bq3060 (or any other SMBus-compatible fuel gauge) provide safety and protection functions as well as detailed information on a battery's present state and allow the application to set the desired charging parameters. These fuel gauges can be programmed for different battery chemistries such as Li-ion or NiMH and have built-in algorithms for charging and discharging cycles to optimize battery performance. Additionally, battery fuel gauges monitor many different parameters throughout the life of the battery to provide accurate state-of-charge information [1]. All of this information can be easily read by a microcontroller such as the MSP430 devices.

The MSP430 microcontrollers are 16-bit RISC instruction set processors with an ultra-low-power architecture and a variety of peripheral options. The peripheral options include ADC (slope, sigma-delta, or SAR), DAC, op-amps, comparators, LCD drivers, USART, and other integrated analog and digital components, all on one silicon die. The MSP430F550x family of microcontrollers features a rich peripheral set such as 10-bit SAR ADC10_A module, multiple timers (capture or compare registers with PWM output capability), USB interface for firmware upgrades, USCI module, watchdog timers, and more [2].

Communication between the microcontroller and the fuel gauge uses the System Management Bus (SMBus) communication protocol. The SMBus standard was developed by a group of companies collaborating together under the umbrella of Smart Battery System (SBS) Implementers Forum to implement one standard communication protocol for smart batteries and other digital devices [3]. SMBus is based on the popular Inter-IC Communication (I^2C) standard and adds enhancements and restrictions to the original I^2C protocol [4]. SMBus is the primary method of communication with the smart-battery fuel gauges. On the MSP430F550x MCUs, the SMBus protocol can be implemented using the I^2C USCI module.

Figure 1 shows a high-level system block diagram of this reference design smart-battery charger.

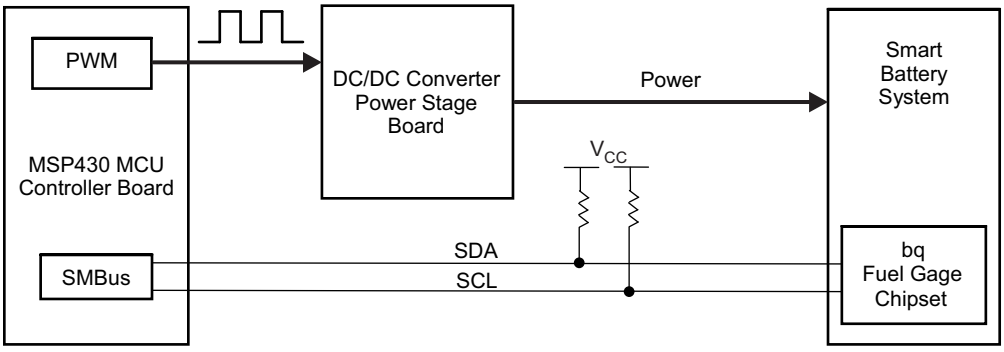


Figure 1. High-Level System Block Diagram of Smart-Battery Charger

This battery charger reference design employs the MSP430F5510 as the microcontroller configured in the SMBus (I^2C) master mode to interrogate the fuel gauge for desired charging voltage, current, and other parameters [5]. The MSP430F5510 then outputs two PWM signals per battery to control both charging voltage and current provided by the DC/DC converter power stage.

Based on the parametric values received through SMBus, the MSP430F5510 either adjusts the PWM duty cycle or shuts off the PWM outputs, if the battery is fully charged or reports a terminate charging condition. A smart-battery containing the bq20z90 fuel gauge with open access to the SMBus terminals was used to test this reference design. If an open smart battery is not available, the bq20z90 fuel gauge evaluation module kit can be used to emulate a smart battery [6]. The reference design assumes that the bq20z90 is configured with charging broadcasts disabled (BCAST = 0 in Operation Cfg B register). However, if the battery fuel gauge does place charging broadcast requests on the SMBus lines, the MSP430F5510 ignores them. Therefore, the fuel gauge responds with parameters only when the MSP430F5510 addresses commands to it.

2 Hardware

2.1 Overall System Description

Figure 2 shows the wide-Vin battery charger system block diagram.

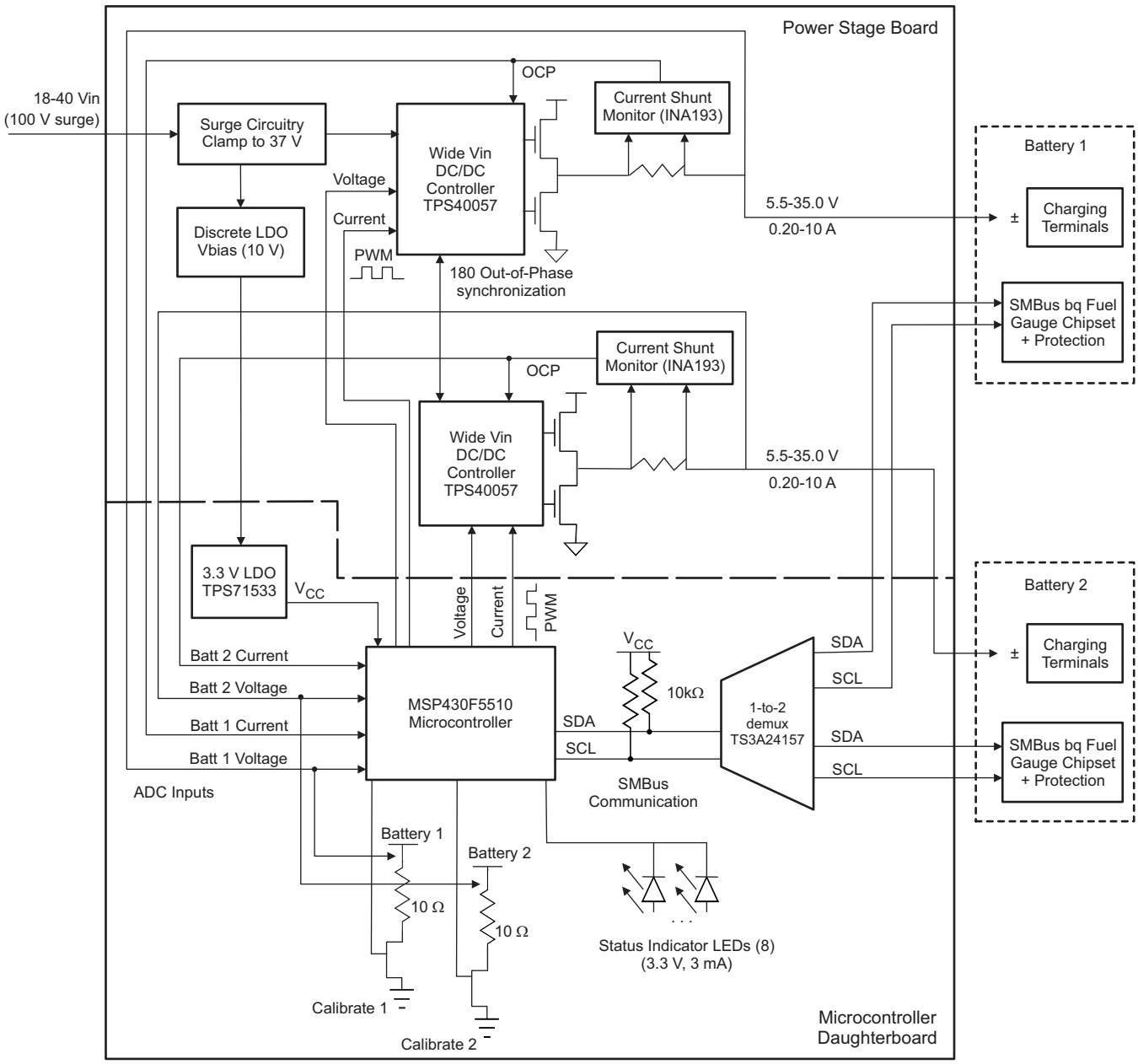


Figure 2. System Block Diagram

This particular system can monitor and charge two smart batteries at the same time. The system primarily comprises two subsystems (boards):

- MSP430F5510 daughterboard subsystem
- Power-stage board with the DC/DC converter subsystem

The MSP430F5510 board contains all of the digital logic and components of the system, while the power-stage board has all of the analog and power components. The MSP430F5510 board docks to the power-stage board through a 10-pin header. The following sections describe each of the subsystems.

2.2 MSP430F5510 Daughterboard Subsystem

2.2.1 Subsystem Description

Figure 3 shows the block diagram of the MSP430F5510 daughterboard subsystem.

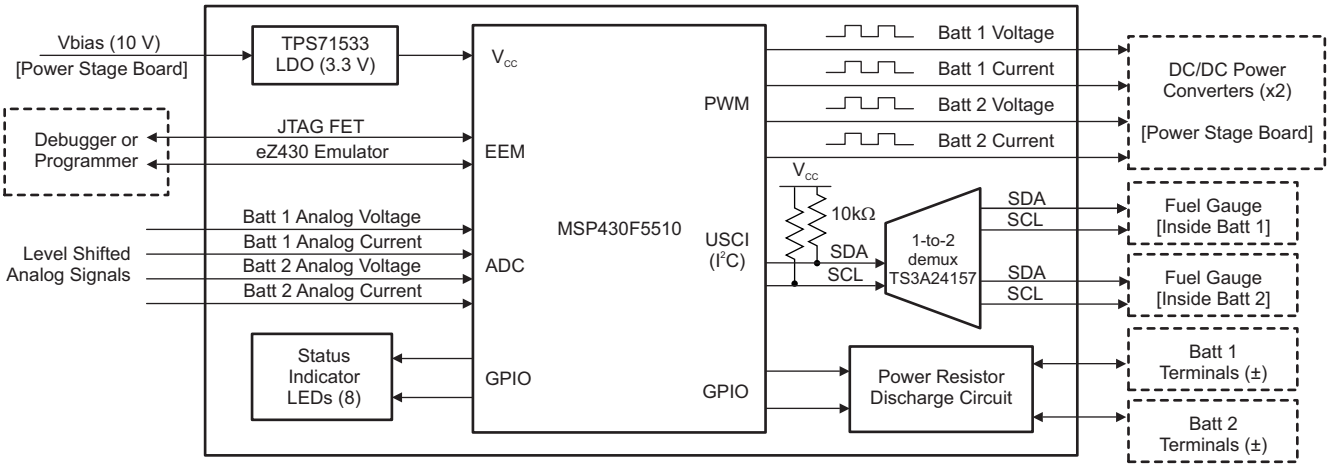


Figure 3. MSP430F5510 Daughterboard Subsystem Block Diagram

The daughtercard subsystem has the following features:

- The I²C USCI module within the MSP430F5510 is used to implement the SMBus protocol for communication with the battery fuel gauges.
- A 1-to-2 demultiplexer (demux), TS3A24157, is used to separate the SMBus clock (SCL) and data (SDA) lines for the two batteries [15]. During the manufacturing process, all of the fuel gauges for a particular battery series are programmed to the same SMBus slave address. The advantage of using a demux is that one microcontroller with one I²C/SMBus USCI module can be used to communicate with multiple fuel gauges within multiple smart-batteries.
- The MSP430F5510 outputs voltage and current PWM signals at a frequency of 20 kHz to control the power delivered by the DC/DC converters on the power stage subsystem.
- The on-chip 10-bit ADC is used to convert voltage and current signals from the batteries. The voltage from the battery is divided down from the wide-input range to the ADC range by means of a resistor-divider circuit on the MSP430F5510 daughterboard. The current is fed into a shunt resistor on the power-stage board, and the resulting voltage is fed into the ADC channels directly.
- Eight status LED indicators; seven are software programmable, and one indicates power-on status.
- Two sets of power resistors for discharging two batteries independently. The discharge circuitry can be turned on or off by the microcontroller to calibrate battery pack voltages. Section 9 has details on setting up these circuits.
- Fan control output to power a heat venting circulation fan on or off.
- Two options to program the software on the MSP430F5510 daughterboard:
  - 14-pin JTAG interface (four-wire) for connecting the flash emulation tool (FET)
  - 6-pin Spy-Bi-Wire interface (two-wire) for connecting the eZ430 emulator

- Three options to power the MSP430F5510 daughterboard:
  - JTAG interface (voltage level programmable in the integrated development environment (IDE) options)
  - eZ430 emulator interface (supply voltage fixed at 3.6 V)
  - The charger board supply power (approximately 10 V), which is routed through the TPS71533 LDO to supply 3.3 V to the MSP430F5510 [16]. For a wider input supply range up to 50 V, the TPS79801 LDO can also be used to supply 3.3 V [17].

2.2.2 MSP430F5510 Port Pins Functionality Description

Table 1 shows the port/pin name to functionality mapping for the MSP430F5510 microcontroller. The signal name column represents the net names referred to in the daughterboard schematic. The right-most column describes the purpose and functionality of the signal net.

Table 1. MSP430F5510 Port-to-Functionality Mapping

Port Name	Signal Name	Description
P1.0	LED0	Status Indicator LED – D1 (Green)
P1.1	LED1	Status Indicator LED – D3 (Green)
P1.2	V_PWM1	Voltage PWM output for Battery 1
P1.3	I_PWM1	Current PWM output for Battery 1
P1.4	V_PWM2	Voltage PWM output for Battery 2
P1.5	I_PWM2	Current PWM output for Battery 2
P1.6	LED2	Status Indicator LED – D4 (Orange)
P1.7	LED3	Status Indicator LED – D5 (Orange)
P2.0	FAN-CTL	Fan Control
P2.1	LED4	Status Indicator LED – D6 (Red)
P2.2	LED5	Status Indicator LED – D7 (Red)
P2.3	LED6	Status Indicator LED – D8 (Green)
P4.0	SMB-CH-SELECT	SMBus Battery Channel Selector
P4.1	430-SMBUS-DATA	SMBus Data Line (SDA)
P4.2	430-SMBUS-CLK	SMBus Clock Line (SCL)
P4.3	LED-ON	Power On Indicator LED – D2 (Green)
P4.6	CAL-CH1	Turns on calibration circuit for Battery 1
P4.7	CAL-CH2	Turns on calibration circuit for Battery 2
P6.0	ISNS1	Current Sampling ADC Channel for Battery 1
P6.1	ISNS2	Current Sampling ADC Channel for Battery 2
P6.2	VBATT1	Voltage Sampling ADC Channel for Battery 1
P6.3	VBATT2	Voltage Sampling ADC Channel for Battery 2

For more details on signal net names and connections, see Section 7 for the MSP430F5510 daughterboard schematic.

2.3 Power Stage Board Subsystem

2.3.1 Subsystem Description

Figure 4 shows a block diagram of the power-stage board.

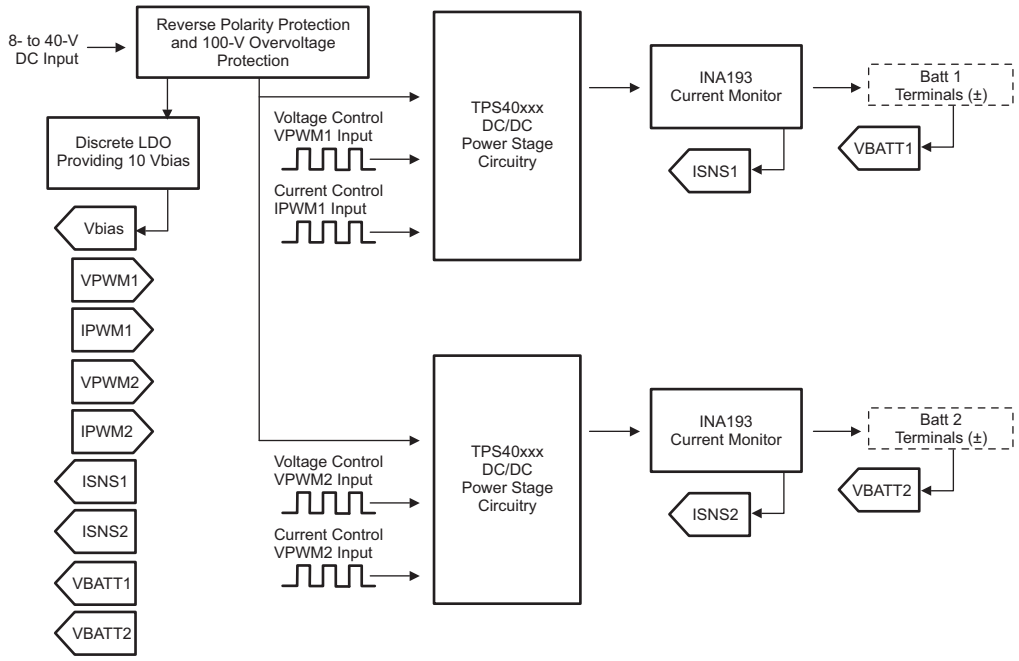


Figure 4. Power Stage Subsystem Block Diagram

The power-stage board has the following features:

- Dual wide-input-voltage buck DC/DC converters for charging two batteries independently. Two power-stage boards with different input voltage ranges were built for this reference design.
  - 40-V input range employing the TPS40057 DC/DC converter (see Section 12) [18]
  - 60-V input range employing the TPS40170 DC/DC converter (see Section 14) [19]
- Output voltage and current controlled by 20-kHz PWM signals provided by MSP430F5510 daughterboard.
- Precision current measurements provided by an INA193 current shunt monitor [20]
- Battery charging feedback network to first provide constant current regulation followed by constant voltage regulation.
- 100-V overvoltage and reverse-polarity protection
- 180° out-of-phase operation

2.3.2 Input Protection Features

This reference design includes protection circuitry for both overvoltage (up to 100 V) as well as reverse voltage (positive and negative leads swapped). This portion of the circuitry is shown in Figure 5.

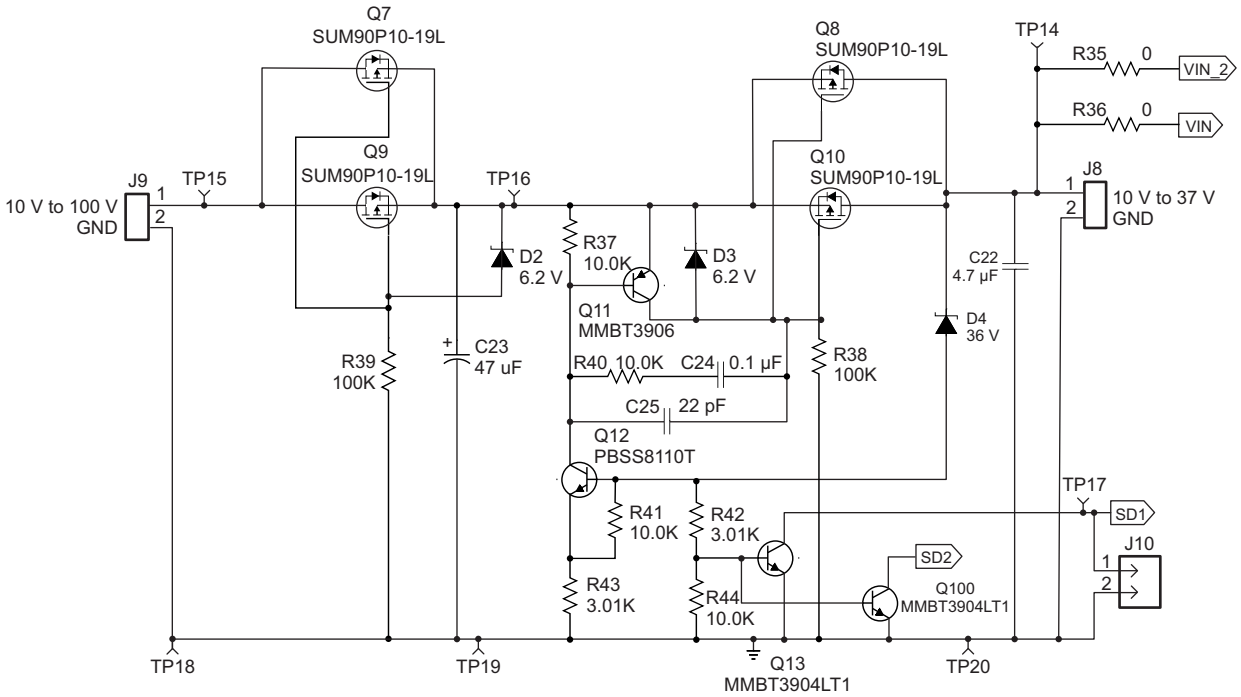


Figure 5. Overvoltage and Reverse Polarity Protection Circuitry

- Reverse voltage protection – FETs Q7 and Q9 along with D2 provide reverse voltage protection in case the input voltage is connected backwards. This does not allow a negative voltage to be applied to the system.
- Input overvoltage protection – FETs Q8 and Q10 provide an overvoltage protection circuit. The zener diode D4 sets the voltage that the circuit starts to clamp. Once the zener voltage is exceeded, the gate-to-source voltage of the FETs starts to drop. This causes the FETs to operate in the linear region. At the same time, the battery charging circuits are turned off with signals SD1 and SD2.

2.3.3 Constant-Voltage and Constant-Current Feedback

Properly charging a battery requires constant current control followed by constant voltage control as the current tapers. Two separate feedback loops are required to address this requirement (see Figure 6 and Figure 7).

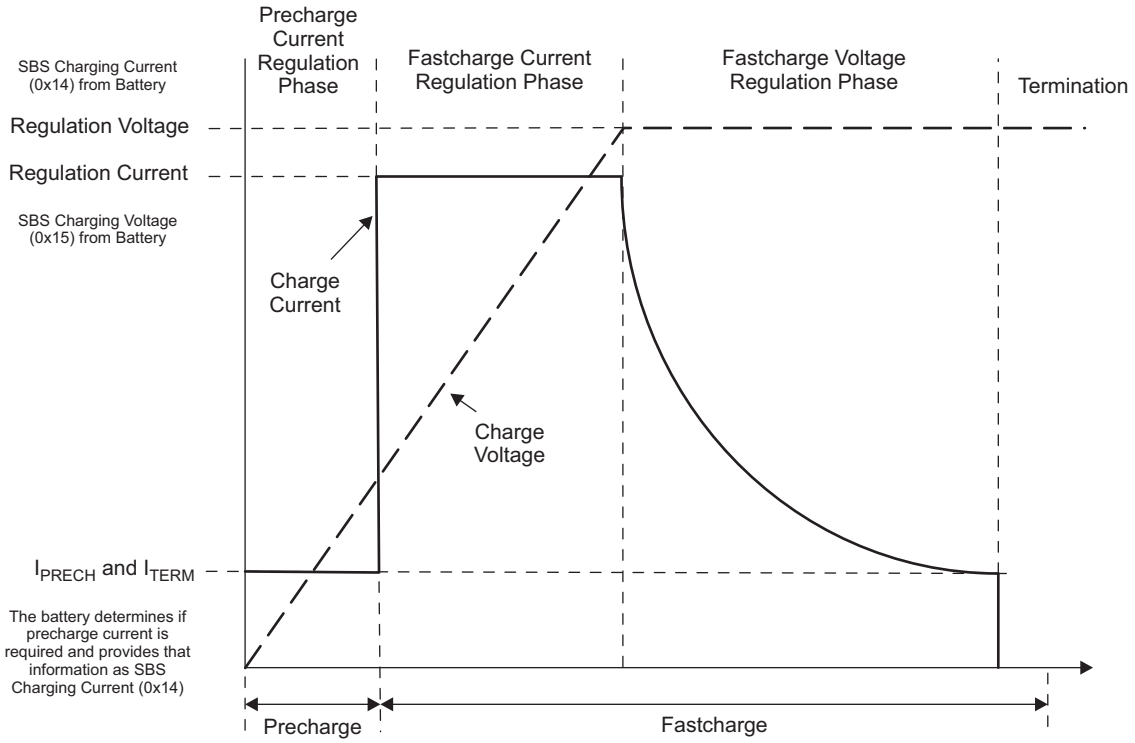


Figure 6. Typical Charging Profile

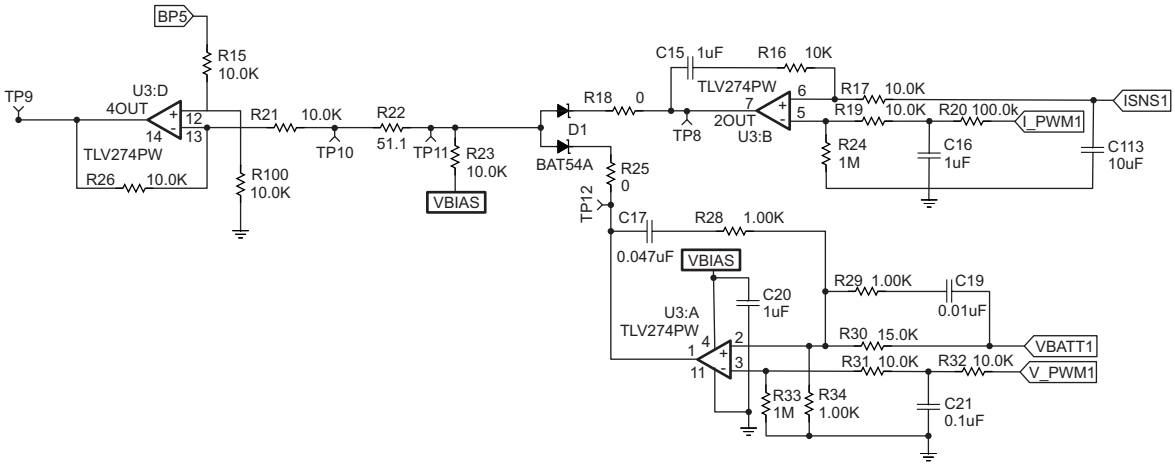


Figure 7. Constant Current and Voltage Feedback to Charge the Battery

- The feedback to the TPS40057 is controlled by two loops, one voltage loop and one current loop. Only one of the loops is in control of the power supply at a given time.
- The current and voltage levels are set by the PWM outputs of the MSP430F5510. Duty cycles between 0% and 100% are filtered to produce an analog voltage reference.
- The current into the battery is measured using shunt resistors (R5, R7, and R8) along with an INA193 (U2) (see schematic in Section 12). This circuit provides a voltage that is proportional to the output current (see Equation 1).

$$\text{Gain} = 20\ \text{V/V} \times \text{R105} / \text{R13} = 13.32 \quad (1)$$

For I_{OUT} = 10 A; I_{SNS} = 10 A × 0.025 Ω × 12.1 = 3.33 V
For I_{OUT} = 10 A; I_{SNS} = 10 A × 0.025 Ω × 12.1 = 3.33 V

- The current sense voltage is then compared to the current reference voltage using U3:B. If the reference voltage is higher, the output of the amplifier is high. If the reference voltage is lower, the output of the amplifier is low.
- The output voltage is measured using a resistor divider (R30 and R34). This voltage is then compared to the output voltage reference. If the reference voltage is higher, the output of the amplifier is high. If the reference voltage is lower, the output of the amplifier is low.

$$V_{OUT\_MAX} = V_{SNS} / \text{R34} \times (\text{R34} + \text{R30}) \quad (2)$$

- Diode D1 combines the outputs of the two amplifiers with a logical OR. The voltage that is lowest is fed into an inverting amplifier that makes the error signal polarity correct for the TPS40057 controller.
- The basic operation is that the controller tries to put out a set current, and if the load can accept this current, the controller regulates to that current level. If the load cannot accept the full amount of current, the voltage begins to rise and eventually reaches V_{OUT_MAX}. When this happens, the voltage loop takes over and regulates the output voltage.

3 Software

3.1 SMBus Protocol Description

The MSP430 microcontroller communicates with the fuel gauge in the smart battery through the SMBus communication protocol. SMBus is based on the I^2C protocol and is a two-wire serial interface: serial clock (SCL) and serial data (SDA). The two lines are connected through pullup resistors to V_{CC}, and the idle state of the bus is V_{CC} or logic HIGH. Multiple devices are connected to the SMBus lines in a wired-AND configuration with the premise that the active devices can drive the lines LOW while other devices relinquish bus control by staying in high-impedance state.

Devices that use the SMBus protocol to communicate can be classified as a master or slave device. A master or slave device can transmit or receive data. The device that initiates the communication packet by placing a START condition on the bus and providing clock pulses on the SCL line is considered as the master. After the master places the START condition on the bus, this command is followed by a 7-bit slave address and a Read/$\overline{\text{Write}}$ bit to indicate if the master is receiving or transmitting data, respectively. As each slave device can have a unique 7-bit address, a maximum of 128 devices can be connected to the bus. If there is a slave device with the address requested by the master, the slave acknowledges (ACK) by pulling the SDA line LOW. If not, the SDA line remains HIGH (during the ninth clock pulse on SCL), which is interpreted as a no-acknowledge (NACK). A successful ACK is followed by a stream of 8-bit packets that can be data, SBS commands, or PEC byte. The receiving device must ACK every time it receives an 8-bit packet. Communication ends when the master device places a STOP condition on the bus.

Unlike the I^2C protocol, the SMBus protocol has a minimum clock frequency of 10 kHz and a maximum clock frequency of 100 kHz. The SMBus protocol also places a time-out restriction that prevents slave devices from extending the clock (SCL) line LOW for a certain interval before the master issues a STOP condition. A slave can hold the SCL line LOW for 25 ms before time-out occurs, after which the slave should be able to receive a new START condition within 35 ms. Another SMBus feature to improve communication robustness is Packet Error Checking (PEC). The PEC byte is generated by using the cyclic redundancy check (CRC-8) polynomial and is calculated on all bytes including device addresses, Read/$\overline{\text{Write}}$ bits, and SBS commands. The PEC calculation, however, does not include the START, repeated START, ACK, NACK, or STOP bits.

The reference design implements the time-out feature on the MSP430F5510 and also has the API calls to access the fuel gauge with or without PEC. The bq fuel gauges can also transmit or receive data with or without PEC. When the MSP430F5510 is receiving data from the fuel gauge and if it clocks out eight extra cycles, then the fuel gauge correspondingly outputs the PEC byte. When the MSP430F5510 is writing data to the fuel gauge and if it passes the PEC byte, then the fuel gauge compares the value against its own hardware-generated PEC value. If the values match, the fuel gauge issues an acknowledge (ACK); if the values do not match, a no-acknowledge (NACK) is issued before the STOP condition.

For additional details on timing diagrams and specifications, see the System Management Bus (SMBus) Specification [3].

3.2 Software File Structure

This section describes the organization of the software code file structure.

- **main.c** – Contains the main() function. The file also contains other functions such as:

Name	Brief Description
Battery_Charger_Controller()	Implements the demo reference design software flow chart.
Initialize_Battery_Definitions()	Initializes the hardware to charge two smart batteries simultaneously.
Get_Battery_Charger_SMBus_ Parameters()	Interrogates the battery fuel gauge for the desired charging voltage, desired charging current, and other parameters.
Safety_Checks_Primary_SMBus()	Compares the parameters received through SMBus against a set of limits to check whether the nominal operating values have been exceeded or not.
Safety_Checks_Secondary_ADC()	Compares the parameters received through ADC conversions against a set of limits to check whether the nominal operating values have been exceeded or not.
Set_LED_Indicator_Status()	Set the hardware LEDs to indicate status such as charging, fully charged or error condition.
Interrupt Service Routines (ISRs)	ISRs for Timer, I^2C/SMBus USCI, ADC10, and other peripherals

- **init.c** – Functions for initializing modules/peripherals on the controller board:

Name	Brief Description
UCS_Init()	Initializes the clock system within the MSP430F5510.
Timer_Init()	Initializes the timer.
PWM_Init()	Initializes the PWM outputs.
ADC_Init()	Initializes the ADC for voltage/current conversions.

- **init.h** – Header file that contains device and controller board specific constants. The constants defined in this file should be used during structure declaration.
- **led.c** – These functions control the operation of the status LEDs on the controller board.

Name	Brief Description
LED_Init()	Initializes the LEDs on the board.
LED_Control()	Set the LEDs to a steady on/off state or set the LEDs to blink with an adjustable period.

- **led.h** – Header file that defines the constants used in functions within the *LED.c* source file.
- **pwm.c** – Functions that control the PWM duty cycle output to the charger board.
- **pwm.h** – Constants associated with the functions defined in *pwm.c* source file.
- **smbus.c** – Source code that uses the I^2C USCI module for SMBus communication.

Name	Brief Description
SMBus_Initialize()	Initializes the SMBus communication lines with the option to configure the MSP430F5510 in master or slave mode.
SMBus_Access()	Configures the MSP430F5510 in master mode and interrogates the battery fuel gauge with the user-specified command.
SMBus_Access_PEC()	Configures the MSP430F5510 in master mode and interrogates the battery fuel gauge with the user-specified command with packet-error-checking (PEC).
SMBus_Select()	Selects which battery channel to send the SMBus commands to.
crc8MakeBitwise()	Implements the CRC-8 algorithm to compute the PEC byte.

- **smbus.h** – Header file with SMBus related constants and definitions.
- **misc.c** – Source file with miscellaneous functions.

Name	Brief Description
Fan_Init()	Initializes the fan structure.
Fan_Control()	Turns the fan on or off.
VI_ADC_Read()	Reads the voltage or current using the ADC.
Calibrate_Battery()	Turns on or off the calibration resistor circuitry.
Delay_Timer()	Generic delay function.

- **misc.h** – Header file with constants and definitions used in *misc.c* source file.
- **device.h** – Header file for target device declaration.
- **F5XX_6XX_Core_Lib** – This folder contains code files for configuring the power management module (PMM) within the MSP430F5510. These source code files are derived from an external set of Hardware Abstraction Libraries (HAL) for the 5xx and 6xx series of devices [7].
  - **hal_pmm.c** – Function library for setting the PMM VCore voltage level. To operate the DCO that drives the SMCLK at a higher or lower frequency, the VCore level must be raised up or lowered down by calling these functions.
  - **hal_pmm.h** – Header file with function declarations and constants.
- **demo.c** – Demo functions that exemplify API function call usage.
- **demo.h** – Header file with function call prototypes declared in *demo.c* source file.

NOTE: The files *demo.c* and *demo.h* should be excluded from the project settings when compiling/building the sample application software. These files are provided for example purposes only.

3.3 API Calls Description

This section describes the individual API function calls with details such as function call syntax, parameters passed and returned, and example use-case declarations. The example values denoted in the parameters section are aliases defined in the respective header files. It is recommended to use these aliases when writing system software to maintain ease of use and documentation.

3.3.1 UCS_Init ( )

This function initializes the universal clock system (UCS) within the MSP430F5510 to:

- Select internally generated REFO (approximately 32 kHz) as the FLL reference clock
- Select internally generated REFO (approximately 32 kHz) as ACLK
- Program the DCO to approximately 20 MHz (for SMCLK and MCLK)

Function Definition

```
void UCS_Init(void) {...}
```

Inputs

None

Return

None

Example Function Call

```
UCS_Init();
```

3.3.2 Timer_Init ( )

This function initializes the Timer module to establish a TIMER_TICK time-base and time-out duration. TIMER_TICK constant is defined in *init.h* header file. The time-out duration for the SMBus clock low extension and the blinking period of the LEDs are based on this Timer.

Function Definition

```
void Timer_Init(void) {...}
```

Inputs

None

Return

None

Example Function Call

```
Timer_Init();
```

3.3.3 PWM_Init ( )

This function initializes the MSP430F5510 port pins to output a 20-kHz PWM with 10-bit resolution. The PWM outputs are initialized to low.

Function Definition

```
void PWM_Init(void) {...}
```

Inputs

None

Return

None

Example Function Call

```
PWM_Init();
```

3.3.4 ADC_Init ( )

This function initializes the on-chip integrated 10-bit ADC to do single-channel single-conversions.

Function Definition

```
void ADC_Init(void) {...}
```

Inputs

None

Return

None

Example Function Call

```
ADC_Init();
```

The function uses a structure to map the MSP430F5510 port pins to the ADC channels and to also keep the assignments separated when multiple batteries are connected to the system.

Structure Definition Name

ADCDescription_t

Name	Type	Description	Example Value
BatteryNum	*unsigned char*	Value that represents battery channel number	BATT_1, BATT_2
ChannelType	*unsigned char*	Channel type (voltage or current)	CHANNEL_VOLTAGE, CHANNEL_CURRENT
PortSelAddr	*unsigned int*	Address of the Port Select Register listed in the register description table of the data sheet.	P6SEL_ADDR
PortBit	*unsigned int*	Bit of the port (value can range from 0 to 15)	BIT0, BIT1, ..., BITF
InputChanNum	*unsigned char*	Value that represents ADC input channel number	ADC10INCH_1, ADC10INCH_2

Example Structure Declaration

For battery 1 voltage on P6.2 (A2) channel:

```
// Constant defined in header file
#define ADC_DEFAULT_STATE {BATT_1, CHANNEL_VOLTAGE, P6SEL_ADDR, BIT2, ADC10INCH_2}

// Declared in source code
ADCDescription_t ADC = ADC_DEFAULT_STATE;
```

3.3.5 Fan_Init ( )

This function configures the port connected to the fan to be in the output direction.

Function Definition

```
void Fan_Init(void) {...}
```

Inputs

None

Return

None

Example Function Call

```
Fan_Init();
```

The function uses a structure to map the MSP430F5510 port pins to the fan control channel.

Structure Definition Name

FanDescription_t

Name	Type	Description	Example Value
PortOutAddr	*unsigned int*	Address of the port out register of the fan control output. This register controls the on or off state of the output.	P2OUT_ADDR
PortDirAddr	*unsigned int*	Address of the port direction register. This register controls the input or output capability of the port.	P2DIR_ADDR
PortBit	*unsigned int*	Bit of the port (value can range from 0 to 15)	BIT0, BIT1, ..., BITF

Example Structure Declaration

For the fan control pin on Port P2.0:

```
// Constant defined in header file
#define FAN_DEFAULT_STATE { P2OUT_ADDR, P2DIR_ADDR, BIT0 }

// Declaration in source code file
FanDescription_t Fan = FAN_DEFAULT_STATE;
```

3.3.6 LED_Init ( )

This function configures all of the port bits connected to the LEDs to output direction and initializes all LEDs to off state.

Function Definition

```
void LED_Init(void) {...}
```

Inputs

None

Return

None

Example Function Call

```
LED_Init();
```

The function uses a structure to map the MSP430F5510 port pins to the LEDs. To allow multiple LEDs to blink at different intervals, virtual timer counters are used to keep track of the time-out duration. This structure declaration keeps track of the virtual count for each LED as well as the time-out duration.

Structure Definition Name

LEDDescription_t

Name	Type	Description	Example Value
PortOutAddr	*unsigned int*	Address of the Port Out Register of the LED control output. This register controls the on/off state of the output.	P2OUT_ADDR
PortDirAddr	*unsigned int*	Address of the Port Direction Register. This register controls the input/output capability of the port.	P2DIR_ADDR
PortBit	*unsigned int*	Bit of the Port (value can range from 0 to 15)	BIT0, BIT1, ..., BITF
LEDVirtualTimer (1)	*unsigned char*	Virtual Timer keeps tracks of timer ticks before time-out expires and LED is toggled.	0
LEDBlinkPeriod (1)	*unsigned char*	This parameter contains the time-out period that the virtual timer starts counting down from. This parameter is used to adjust the LED blink rate.	0

(1) The parameters LEDVirtualTimer and LEDBlinkPeriod are initialized to zero and are used by the LED_Control function to set the blink period/timer value based on the user's input for that particular LED. Initializing these parameters to non-zero values can cause the LEDs to function improperly.

Example Structure Declaration

For a case with two LEDs on P2.0 and P2.1:

```
// Constant defined in header file
#define LED_DEFAULT_STATE { { P2OUT_ADDR, P2DIR_ADDR, BIT0, 0, 0 }, { P2OUT_ADDR, P2DIR_ADDR,
BIT1, 0, 0 }  }

// Declaration in source code file
LEDDescription_t LEDs[2] = LED_DEFAULT_STATE;
```

3.3.7 SMBus_Initialize ( )

This function initializes the I²C USCI module within the MSP430F5510 for SMBus communication.

- Uses SMCLK as synchronous clock source to operate SMBus clock (SCL) at approximately 100 kHz.
- Can configure the MSP430F5510 to act in master or slave mode. The master mode is useful when only the MSP430F5510 interrogates the fuel gauge; the slave mode is useful when the battery is setup in Broadcast Mode.
- Master and slave addresses for the MSP430F5510 are defined in the header file *smbus.h*

Function Definition

```
void SMBus_Initialize (unsigned char SMBus_Mode) {...}
```

Inputs

Name	Type	Description	Example Value
SMBus_Mode	*unsigned char*	Option to configure the MSP430F5510 in SMBus master or SMBus slave mode.	SMBUS_MASTER_MODE, SMBUS_SLAVE_MODE

Return

None

Example Function Call

To configure the MSP430F5510 in master mode:

```
SMBus_Initialize SMBUS_MASTER_MODE);
```

The function uses a structure to map the MSP430F5510 port pins to the 1-to-2 demux (to prevent SMBus signal collision) and the SMBus clock and data Lines.

Structure Definition Name

SMBusDescription_t

Parameters

Name	Type	Description	Example Value
PortChanDir	*unsigned int*	Address of the port direction register connected to the 1-to-2 demux. This register controls the input or output capability of the port.	P4DIR_ADDR
PortChanOut	*unsigned int*	Address of the port output register connected to the 1-to-2 demux. This register controls the on or off state of the output.	P4OUT_ADDR
PortChanBit	*unsigned int*	Bit of the port connected to the 1-to-2 demux (value can range from 0 to 15)	BIT0, BIT1, ..., BITF
PortComAddr	*unsigned int*	Address of the port select register connected to the I²C USCI lines for SMBus.	P6SEL_ADDR
PortComBitData	*unsigned int*	Bit of the port connected to the data line of the I²C USCI module (value can range from 0 to 15)	BIT0, ...,BITF
PortComBitClock	*unsigned int*	Bit of the port connected to the clock line of the I²C USCI module (value can range from 0 to 15)	BIT0, ...,BITF

Example Structure Declaration

For the SMBus channel select pin on P4.0 and SMBus communication lines SDA on P4.1 and SCL on P4.2:

```
// Constant defined in header file
#define SMBUS_DEFAULT_STATE {P4DIR_ADDR, P4OUT_ADDR, BIT8, P4SEL_ADDR, BIT9, BITA}

// Declaration in source code file
SMBusDescription_t SMBus = SMBUS_DEFAULT_STATE;
```

3.3.8 LED_Control ( )

This function provides control of the LEDs on the board. Call the *LED_Init()* function to initialize the LED ports to output direction before making calls to this function.

Function Definition

```
void LED_Control (unsigned char led_num, unsigned char led_mode, unsigned char
led_blink_rate) {...}
```

Inputs

Name	Type	Description	Example Value
led_num	*unsigned char*	Number designated to the LED on the hardware	LED_NUM_0, LED_NUM_1, ..., LED_NUM_7
led_mode	*unsigned char*	Configures the LED to stay in either a steady on/off state or blink periodically.	LED_MODE_ON, LED_MODE_OFF, LED_MODE_BLINK
led_blink_rate	*unsigned char*	Configures the blink rate of the LED, if it has been programmed to blink. Value has no effect when LED is set to a steady on or off state.	LED_BLINK_RATE_SLOW, LED_BLINK_RATE_MEDIUM , LED_BLINK_RATE_FAST

Return

None

Example Function Call

To make LED0 blink at a medium rate:

```
LED_Control(LED_NUM_0, LED_MODE_BLINK, LED_BLINK_RATE_MEDIUM);
```

To turn LED1 off:

```
LED_Control(LED_NUM_1, LED_MODE_OFF, LED_BLINK_RATE_SLOW);
```

3.3.9 Fan_Control ( )

This function provides control of the fan. Call the *Fan_Init()* function to initialize the fan port before configuring the fan to an on or off state.

Function Definition

```
void char Fan_Control (unsigned char on_off) {...}
```

Inputs

Name	Type	Description	Example Value
on_off	*unsigned char*	Sets the state of the venting fan output to either ON or OFF.	FAN_ON, FAN_OFF

Return

None

Example Function Call

To turn the fan on:

```
Fan_Control(FAN_ON);
```

3.3.10 VI_ADC_Read ( )

This function provides access to battery voltage and current using ADC conversion. Call the *ADC_Init()* function to initialize the ADC for single-channel single-conversions before using this function to receive digital conversion values.

Function Definition

```
unsigned int VI_ADC_Read (unsigned char batt_num, unsigned char channel_vi) {...}
```

Inputs

Name	Type	Description	Example Value
batt_num	*unsigned char*	Value that represents battery channel number	BATT_1, BATT_2
channel_vi	*unsigned char*	Channel type (voltage or current)	CHANNEL_VOLTAGE, CHANNEL_CURRENT

Return

Name	Type	Description
conversion_value	*unsigned int*	Digital conversion value from the ADC (can be voltage or current based on the specified input parameters)

Example Function Call

To read the ADC conversion voltage value on Battery 1:

```
conversion_value = VI_ADC_Read(BATT_1, CHANNEL_VOLTAGE);
```

3.3.11 SMBus_Select ( )

This function selects the active SMBus battery channel.

Function Definition

```
void SMBus_Select(unsigned char batt_num) {...}
```

Inputs

Name	Type	Description	Example Value
batt_num	*unsigned char*	Value that represents battery channel number	BATT_1, BATT_2

Return

None

Example Function Call

Select battery 2 for SMBus communication:

```
SMBus_Select(BATT_2);
```

3.3.12 Calibrate_Battery ( )

This function provides control over the power resistor circuitry to discharge the selected battery. This discharging circuit can be used for calibrating battery pack voltages and can be turned on or off by calling this function. For details on hardware setup for this circuit, see Section 9.

Function Definition

```
void Calibrate_Battery (unsigned char batt_num, unsigned char on_off) {...}
```

Inputs

Name	Type	Description	Example Value
batt_num	*unsigned char*	Value that represents battery number	BATT_1, BATT_2
on_off	*unsigned char*	Turns the calibration circuit to on or off state.	CAL_ON, CAL_OFF

Return

None

Example Function Call

To turn on the discharge circuit for battery 1 voltage calibration:

```
Calibrate_Battery(BATT_1, CAL_ON);
```

3.3.13 Delay_Timer ( )

This function implements a fixed delay in the program. It halts program execution and places the CPU in low-power mode. Rather than consume CPU cycles, it uses a Timer running in the background to count delay. This delay function acts like a software virtual counter where one count for the virtual counter is equivalent to TIMER_TICK counts of the hardware timer. TIMER_TICK is a definition in the *misc.h* that can be easily modified. It is set to default of 25 ms as it is used to implement SMBus time-out and LED refresh blink period. When the function reaches the desired delay, it wakes up the CPU and program execution resumes.

Function Definition

```
void Delay_Timer(int number_of_ticks) {...}
```

Inputs

Name	Type	Description	Example Value
number_of_ticks	*unsigned char*	Amount of delay required. Delay is a multiple of the TIMER_TICK duration.	1, 2, 3

Return

None

Example Function Call

To have a 100 ms delay where TIMER_TICK has been defined for 25-ms delay:

```
Delay_Timer(4);
```

3.3.14 PWM_Control ( )

This function provides access to PWM output control. The timer within the MSP430F5510 is configured to output the PWM at a rate of 20 kHz and can independently adjust two outputs for controlling voltage and current to the DC/DC converter.

Function Definition

```
void PWM_Control (unsigned char batt_num, unsigned char pwm_channel, unsigned char
on_off, unsigned int pwm_duty_period) {...}
```

Inputs

Name	Type	Description	Example Value
batt_num	*unsigned char*	Value that represents battery channel number	BATT_1, BATT_2
pwm_channel	*unsigned char*	Channel type to output PWM: voltage or current	CHANNEL_VOLTAGE, CHANNEL_CURRENT
on_off	*unsigned char*	Set the PWM output to HIGH or LOW	PWM_ON, PWM_OFF
pwm_duty_period	*unsigned int*	Configure the duty cycle of the PWM output	PWM_DUTY_0 (approximately 0%)... PWM_DUTY_100 (approximately 100%)

Return

None

Example Function Call

To turn off PWM on battery 1 voltage channel:

```
PWM_Control(BATT_1, CHANNEL_VOLTAGE, PWM_OFF, PWM_DUTY_0);
```

To specify 10% duty cycle on battery 1 voltage channel:

```
PWM_Control(BATT_1, CHANNEL_VOLTAGE, PWM_ON, 0.1 * PWM_DUTY_100);
```

Where pwm_duty_period = 0.1 × PWM_DUTY_100.

3.3.15 Smbus_Access ( )

This function is implemented as a generic SMBus access function when the MSP430F5510 is initialized to master transmitter and receiver mode. This function handles the communication between the MSP430F5510 master and the slave bq fuel gauge device. It also assumes that the slave fuel gauge has the broadcast mode disabled.

NOTE: When using this function to write to the fuel gauge, use the *Delay_Timer()* function to wait for 50 to 100 ms before issuing a read or write command to the fuel gauge. This delay is necessary to give time for the data flash memory within the fuel gauge to be written without any corruption.

Function Definition

```
unsigned char Smbus_Access (unsigned char smbus_command, unsigned char read_write,
unsigned char size_in_bytes {...}
```

Inputs

Name	Type	Description	Example Value
smbus_command	*unsigned char*	One byte of SMBus command (0x00 to 0x7F). See Section 4 for a list of supported SMBus commands.	SBS_CMD_VOLTAGE, SBS_CMD_CHARGING_CURRENT
read_write	*unsigned char*	Set the mode to either read data from or write data to the slave device.	SMBUS_MASTER_MODE_READ, SMBUS_MASTER_MODE_WRITE
size_in_bytes	*unsigned char*	Based on the command syntax, size of the data set to be sent or received.	0x01 to 0x20
SMBus_Data_To_Slave	*unsigned char[]*	Array of data to be sent to the slave. The calling function should parse the data into byte-size elements and stuff them into this array.	

Return

Name	Type	Description	Example Value
smbus_access_status	*unsigned char*	Returns the status of whether the command was successful or not	FLAG_SUCCESS=0, FLAG_FAIL=1, FLAG_NACK=2
SMBus_Data_From_Slave	*unsigned char[]*	Array of data to be received from the slave. The calling function should concatenate the byte-size elements to construct the data received.	

Example Function Call

To interrogate the battery voltage through SMBus:

```
smbus_access_status = SMBus_Access(SBS_CMD_VOLTAGE, SMBUS_MASTER_MODE_READ, 2);

// Tables for data to/from Slave Device declared with size of 32 bytes
#define SMBUS_DATA_TO_SLAVE 32               //Table length
unsigned char SMBus_Data_To_Slave[SMBUS_DATA_TO_SLAVE];

#define SMBUS_DATA_FROM_SLAVE 32            //Table length
unsigned char SMBus_Data_From_Slave[SMBUS_DATA_FROM_SLAVE];
```

3.3.16 Smbus_Access_PEC ( )

This function is implemented as a generic SMBus access function with packet error checking (PEC) implementation on the MSP430F5510 initialized to master mode. While similar to the SMBus_Access function, this function can either generate the PEC byte (if the master is transmitter) or can compare the PEC byte received against the internally computed one (if the master is receiver).

This function calls another function *crc8MakeBitwise()*, which computes the PEC byte within the MSP430F5510. In the master transmitter case, the PEC byte is appended last to the transmit buffer. If the slave fuel gauge returns an ACK immediately following the PEC byte, then data integrity was preserved during the transaction. On the other hand, if the return is a NACK, then some bits are being altered unintentionally during the transaction. In the case of the master receiver, the last byte in the receive buffer is the PEC byte. The PEC byte is internally generated, compared with the received one and if the bytes are equal, then a valid transaction occurred.

Function Definition

```
unsigned char Smbus_Access_PEC (unsigned char smbus_command, unsigned char
read_write, unsigned char size_in_bytes {...}
```

Inputs

Name	Type	Description	Example Value
smbus_command	*unsigned char*	One byte of SMBus command (0x00 to 0x7F). See Section 4 for a list of supported SMBus commands.	SBS_CMD_VOLTAGE, SBS_CMD_CHARGING_CURRENT
read_write	*unsigned char*	Set the mode to either read data from or write data to the slave device.	SMBUS_MASTER_MODE_READ, SMBUS_MASTER_MODE_WRITE
size_in_bytes	*unsigned char*	Based on the command syntax, size of the data set to be sent or received.	0x01 to 0x20
SMBus_Data_To_Slave	*unsigned char[]*	Array of data to be sent to the slave. The calling function should parse the data into byte-size elements and stuff them into this array.	

Return

Name	Type	Description	Example Value
smbus_access_status	*unsigned char*	Returns the status flag if the PEC byte sent by the slave fuel gauge matches the ones computed in the MSP430F5510 or if the PEC byte written by the MSP430F5510 is acknowledged by the slave device.	FLAG_SUCCESS=0, FLAG_FAIL=1, FLAG_NACK=2
SMBus_Data_From_Slave	*unsigned char[]*	Array of data to be received from the slave. The calling function should concatenate the byte-size elements to construct the data received.	

Example Function Call

To access the battery voltage through SMBus with PEC:

```
smbus_access_status = SMBus_Access_PEC(SBS_CMD_VOLTAGE, SMBUS_MASTER_MODE_READ, 2);

// Tables for data to/from Slave Device declared with size of 32 bytes
#define SMBUS_DATA_TO_SLAVE 32              //Table length
unsigned char SMBus_Data_To_Slave[SMBUS_DATA_TO_SLAVE];

#define SMBUS_DATA_FROM_SLAVE 32           //Table length
unsigned char SMBus_Data_From_Slave[SMBUS_DATA_FROM_SLAVE];
```

3.3.17 crc8MakeBitwise ( )

This function implements the cyclic redundancy check (CRC) algorithm to generate the PEC byte. It has been derived from the CRC16 and CRC32 functions presented in *CRC Implementation With MSP430 MCUs* and has been modified to output a CRC-8 error check byte [8]. It performs XOR operations in the order the bits are received using the CRC-8 polynomial: $C(x) = x^8 + x^2 + x^1 + 1$. The calculation is done on all bytes including device addresses, Read/$\overline{\text{Write}}$ bits, and SBS commands. However, it does not include the START, repeated START, ACK, NACK, or STOP bits.

Function Definition

```
unsigned short crc8MakeBitwise(unsigned char CRC, unsigned char Poly, unsigned char
*Pmsg, unsigned int Msg_Size) {...}
```

Inputs

Name	Type	Description	Example Value
CRC	*unsigned char*	Initial value of the CRC byte	CRC8_INIT_REM
Poly	*unsigned char*	CRC-8 polynomial 0x07 (the '1' in the polynomial 0x107 is implied)	CRC8_POLY
*Pmsg	*unsigned char*	Input stream for which the CRC-8 PEC byte is computed.	Data stream parsed into an array of byte-size elements
Msg_Size	*unsigned char[]*	Number of bytes in the input bit stream	5 (for five bytes)

Return

Name	Type	Description
unsigned short	*unsigned short*	The computed CRC-8/PEC byte.

Example Function Call

To calculate the CRC-8/PEC byte with a five-element array:

```
unsigned char crc_msg[5];
crc_generated = crc8MakeBitwise(CRC8_INIT_REM, CRC8_POLY, crc_msg, crc_msg_size);
```

3.4 Sample Application Description

The sample application included with this reference design showcases a demo smart battery charger. The application runs on the MSP430F5510 daughterboard connected to the Power Stage Board by the means of a 10-pin connector. Instructions for setting up the MSP430F5510 daughterboard are listed in Section 8 and instructions for setting up the Power Stage Board are listed in Section 15.

The hardware setup requires connecting the charger setup to a smart battery with access to the battery terminals and the SMBus communication bus. The MSP430F5510 daughterboard has two headers to connect the SMBus lines from two batteries and are illustrated in the schematic as J2 and J3 (Section 7). The Power Stage Board can independently charge two batteries and the output terminals are denoted as J2 and J12 in the illustrated schematic (Section 12 for 40 V and Section 14 for 60 V versions).

If a smart battery is not available, the system can be tested out by using a SMBus battery fuel gauge evaluation module. This reference design was tested with the bq20z90EVM module, which has a four-pin header for SMBus communication and an input connector for resistors. The idea is to apply an external voltage across the resistors connected in series to simulate the cell voltages of a multi-cell Li-ion battery. For additional details on setting up the EVM, see the Quick Start Guide [1] and the bq20z90EVM User's Guide [9].

Another advantage of this EVM is that the bq device can easily be programmed and configured unlike a sealed battery in which the bq register settings are locked. An example programmer that can accomplish such task is the EV2300. It supports multiple protocols to communicate with fuel gauges such as SMBus, I^2C and HDQ [11]. The EV2300 communicates to the PC through USB and the control panel display GUI on the PC allows easy modifications to register settings [12]. For detailed instructions on setting up the evaluation software to communicate with the EVM, see the *bqEASY Evaluation Software User's Guide* [13].

Figure 8 shows an abbreviated flow chart of the sample application provided with this reference design. The detailed flow chart with safety checks is presented in Section 5.

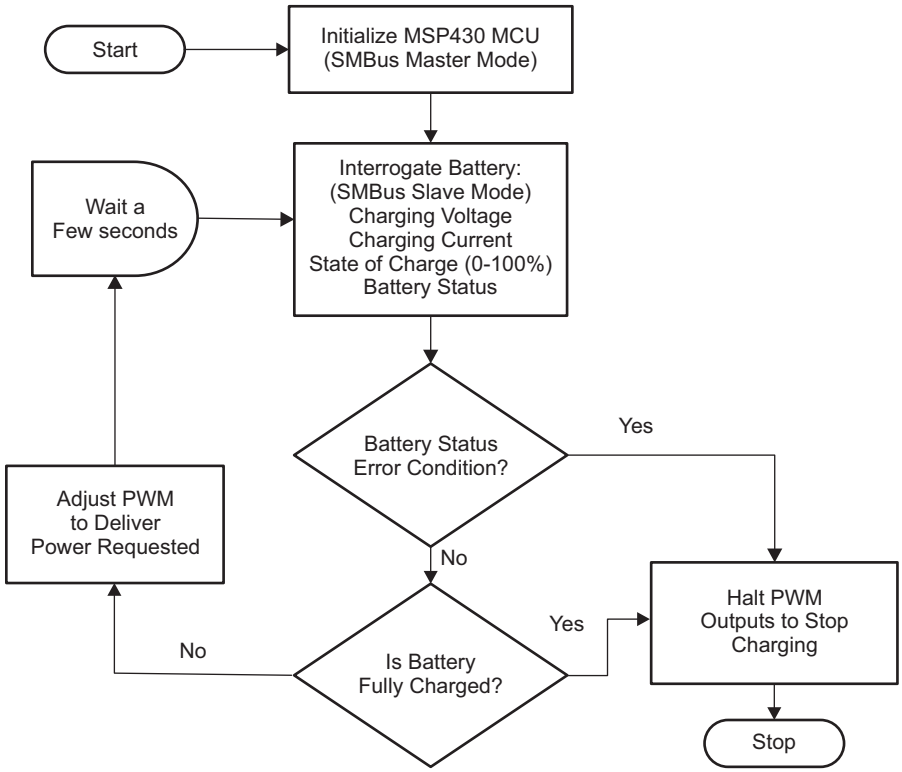


Figure 8. Sample Application Flow Chart (Brief)

The application initializes the MSP430F5510 in SMBus master mode and assumes that the bq fuel gauge is configured in SMBus slave mode (with broadcasts turned off). With charge broadcasts disabled, the fuel gauge does not seize control of the SMBus by becoming the SMBus master when it detects an idle condition. It is recommended to keep the fuel gauge charge broadcasts disabled for robust operation of this reference design application.

The MSP430F5510 interrogates the fuel gauge for parameters such as charging voltage, charging current, state of charge, and battery status register value. The choice of parameters is governed by two factors: the desired charging power requested by the battery and any indication of error or warning condition with the battery. For a full list of parameters that can be interrogated from the fuel gauge through SMBus, see Section 4.

The MSP430F5510 outputs voltage and current PWM signals at 20 kHz to the DC/DC converters on the power stage board. Based on the values of the desired charging voltage and charging current required, the duty cycle of the PWM signals is adjusted accordingly. Section 10 gives a duty cycle computation example along with mapping table for voltage, and Section 11 gives an example for current.

Another level of protection involves taking the voltage from the battery terminals and the current from the power-stage board and level-shifting down to be sampled by the ADC10 on the MSP430F5510. If the voltage or current sampled by the ADC exceed a certain range reported over the SMBus, then the PWM outputs are switched off to prevent any hazard.

4 SBS Supported Commands Using SMBus Protocol

Table 2 shows SBS commands that are supported by the bq fuel gauges using the Smbus_Access or Smbus_Access_PEC function. For details on each individual SBS command, see the *bq20z90-V1.50 + bq29330, bq20z95 Technical Reference* [10].

Table 2. SBS Commands

SBS CMD	Mode	Name	Format	Size In Bytes	Min Value	Max Value	Default Value	Unit
0x00	R/W	ManufacturerAccess	hex	2	0x0000	0xffff	-	
0x01	R/W	RemainingCapacityAlarm	unsigned int	2	0	65535	-	mAh or 10 mWh
0x02	R/W	RemainingTimeAlarm	unsigned int	2	0	65535	-	min
0x03	R/W	BatteryMode	hex	2	0x0000	0xffff	-	
0x04	R/W	AtRate	signed int	2	-32768	32767	-	mA or 10 mW
0x05	R	AtRateTimeToFull	unsigned int	2	0	65535	-	Min
0x06	R	AtRateTimeToEmpty	unsigned int	2	0	65535	-	Min
0x07	R	AtRateOK	unsigned int	2	0	65535	-	
0x08	R	**Temperature** (1)	unsigned int	2	0	65535	-	0.1K
0x09	R	**Voltage** (1)	unsigned int	2	0	20000	-	mV
0x0a	R	**Current** (1)	signed int	2	-32768	32767	-	mA
0x0b	R	AverageCurrent	signed int	2	-32768	32767	-	mA
0x0c	R	MaxError	unsigned int	1	0	100	-	%
0x0d	R	**RelativeStateOfCharge** (1)	unsigned int	1	0	100	-	%
0x0e	R	AbsoluteStateOfCharge	unsigned int	1	0	100	-	%
0x0f	R	RemainingCapacity	unsigned int	2	0	65535	-	mAh or 10 mWh
0x10	R	FullChargeCapacity	unsigned int	2	0	65535	-	mAh or 10 mWh
0x11	R	RunTimeToEmpty	unsigned int	2	0	65535	-	min
0x12	R	AverageTimeToEmpty	unsigned int	2	0	65535	-	min
0x13	R	AverageTimeToFull	unsigned int	2	0	65535	-	min
0x14	R	**ChargingCurrent** (1)	unsigned int	2	0	65535	-	mA
0x15	R	**ChargingVoltage** (1)	unsigned int	2	0	65535	-	mV
0x16	R	**BatteryStatus** (1)	unsigned int	2	0x0000	0xffff	-	
0x17	R	CycleCount	unsigned int	2	0	65535	-	
0x18	R/W	DesignCapacity	unsigned int	2	0	65535		mAh or 10 mWh
0x19	R/W	DesignVoltage	unsigned int	2	700	16000	14400	
0x1a	R/W	SpecifcationInfo	unsigned int	2	0x0000	0xffff	0x0031	
0x1b	R/W	ManufactureDate	unsigned int	2	0	65535	0	
0x1c	R/W	SerialNumber	hex	2	0x0000	0xffff	0x0001	
0x20	R/W	ManufacturerName	string	11+1	-	-	TI	ASCII
0x21	R/W	DeviceName	string	7+1	-	-	bq20z80	ASCII
0x22	R/W	DeviceChemistry	string	4+1	-	-	LION	ASCII
0x23	R	ManufacturerData	string	14+1	-	-	-	ASCII
0x2f	R/W	Authenticate	string	20+1	-	-	-	
0x3c	R	CellVoltage4	unsigned int	2	0	65535		mV
0x3d	R	CellVoltage3	unsigned int	2	0	65535		mV
0x3e	R	CellVoltage2	unsigned int	2	0	65535		mV
0x3f	R	CellVoltage1	unsigned int	2	0	65535		mV

(1) This parameter is used in the sample application.

5 Detailed Sample Application Flow Chart

Figure 9 shows a detailed flow chart for the sample software application implemented in this reference design. Figure 10 gives a detailed view of the primary and secondary safety checks being performed on the parameters received from SMBus and the ADC, respectively.

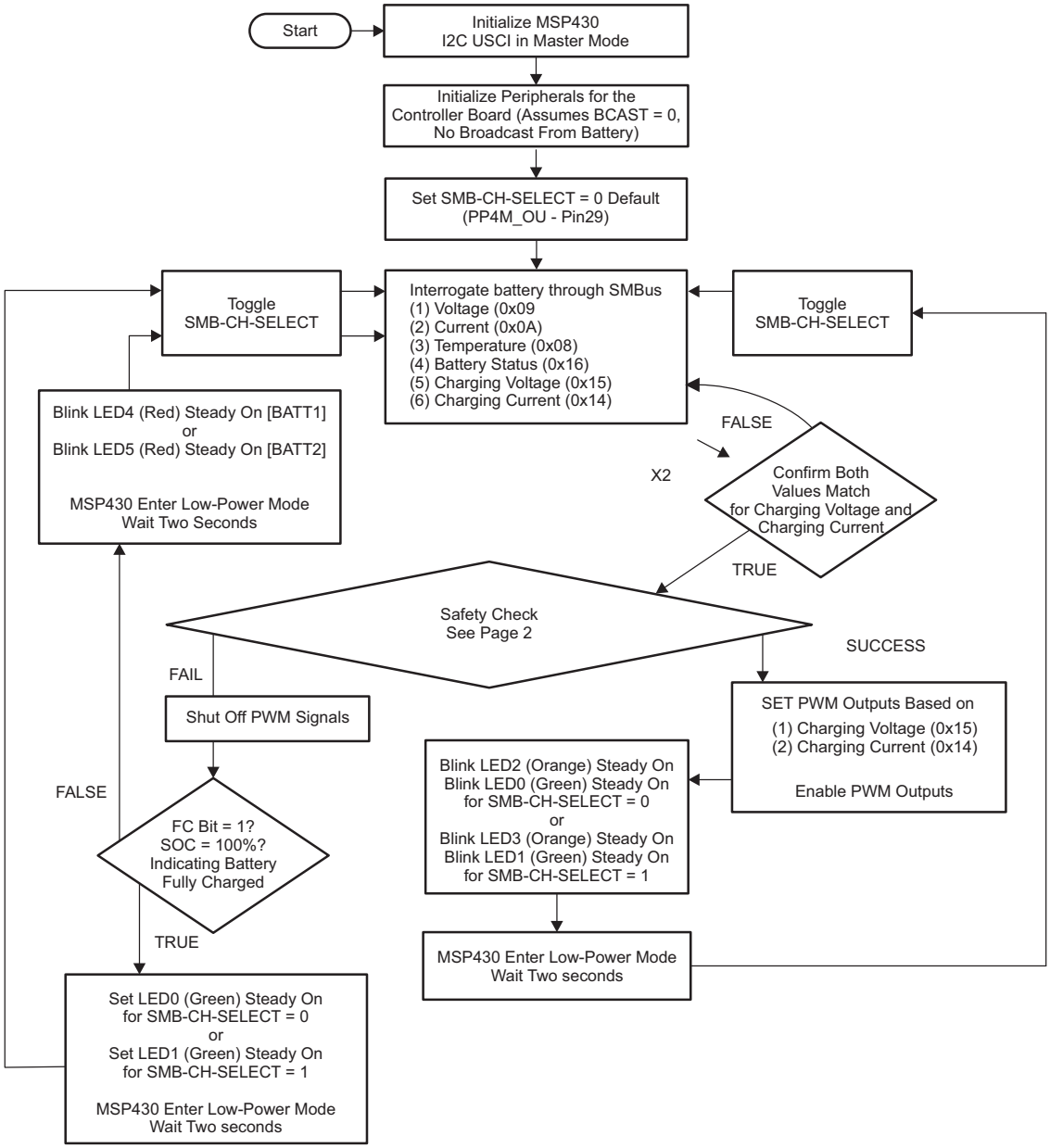
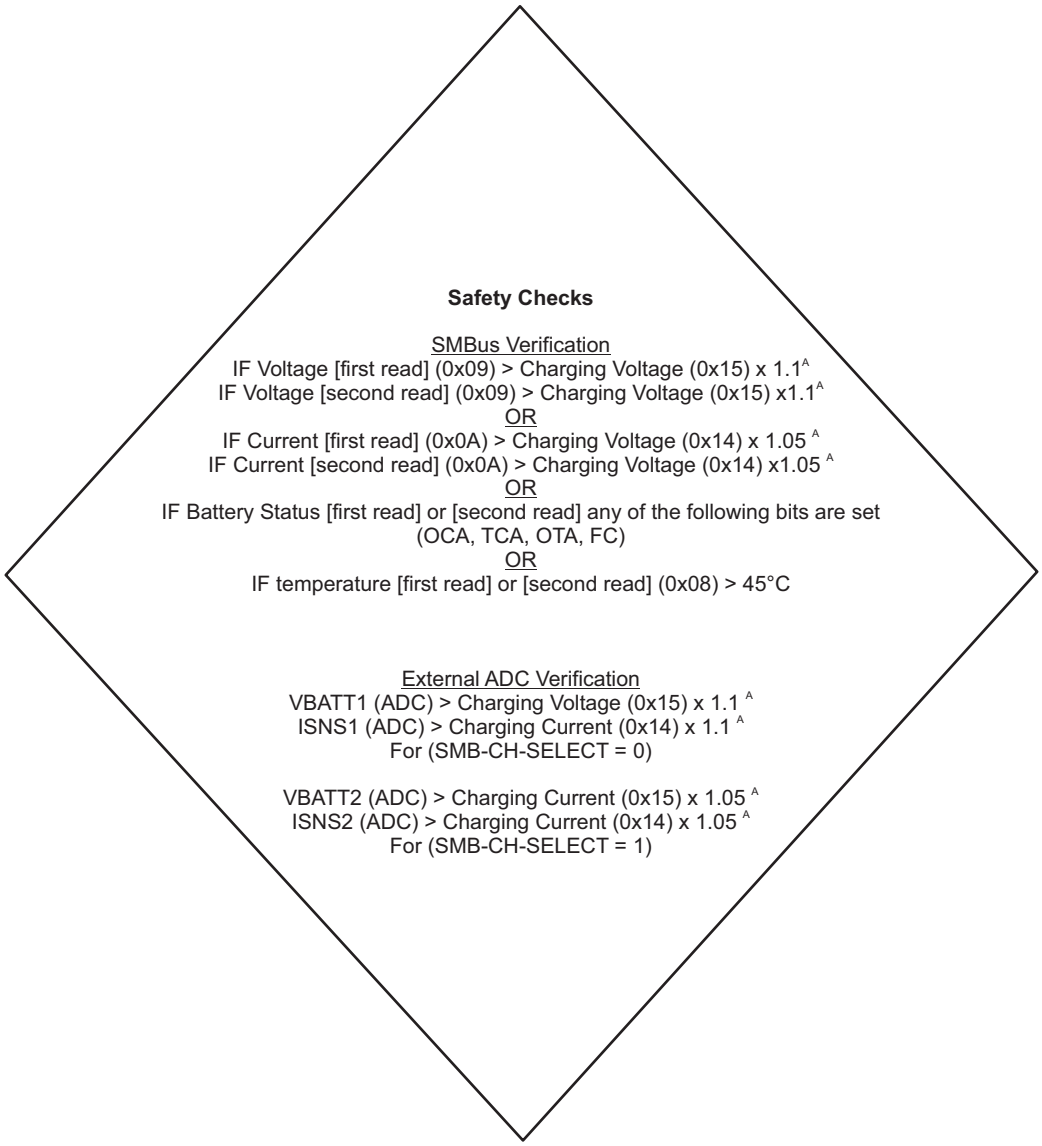


Figure 9. Sample Application Flow Chart (Detailed)



NOTE: These factors are adjustable in the code by changing the percentage factor.

Figure 10. Safety Checks

6 Battery Status Register Description

The following section describes the individual bits within the Battery Status Register. The corresponding SMBus Command is 0x16 and the sample application software monitors the following bits for safety checks: **OCA**, **TCA**, **OTA**, and **FC**. Additional details are available in the *bq20z90-V1.50 + bq29330, bq20z95 Technical Reference* [10].

6.1 BatteryStatus (0x16)

This read-word function returns the status of the bq20z90 or bq20z95-based battery.

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
High Byte	OCA	TCA	RSVD	OTA	TDA	RSVD	RCA	RTA
Low Byte	INIT	DSG	FC	FD	EC3	EC2	EC1	EC0

LEGEND: All Values Read Only; RSVD = Reserved

OCA: **1 = Over Charged Alarm**(1)

TCA: **1 = Terminate Charge Alarm**(1)

OTA: 1 = Over Temperature Alarm(1)

TDA: 1 = Terminate Discharge Alarm

RCA: Remaining Capacity Alarm

1 = Remaining Capacity Alarm is set [see: SBS:RemainingCapacityAlarm(0x01)]

RTA: Remaining Time Alarm

1 = Remaining Time Alarm is set [see: SBS:RemainingTimeAlarm(0x02)]

INIT: 1 = Initialization. This flag is cleared approximately 1 second after device reset, after all SBS parameters have been measured and updated.

DSG: Discharging

0 = bq20z90 or bq20z95 is in charging mode

1 = bq20z90 or bq20z95 is in discharging mode, relaxation mode, or valid charge termination has occurred (see the *Gas Gauging* section in *bq20z90-V1.50 + bq29330, bq20z95 Technical Reference* (SLUU264) [10]

FC: 1 = **Fully Charged**(1)

FD: 1 = Fully Discharged

EC3, EC2, EC1, EC0: Error Code, returns status of processed SBS function

0,0,0,0 =	OK	bq20z90 or bq20z95 processed the function code with no errors detected.
0,0,0,1 =	BUSY	bq20z90 or bq20z95 is unable to process the function code at this time.
0,0,1,0 =	Reserved	bq20z90 or bq20z95 detected an attempt to read or write to a function code reserved by this version of the specification or bq20z90 or bq20z95 detected an attempt to access an unsupported optional manufacturer function code.

(1) This parameter is used in the sample application.

7 MSP430F5510 Daughterboard Schematics

Figure 11 and Figure 12 show the schematics of the MSP430F5510 microcontroller daughterboard (PMP5385).

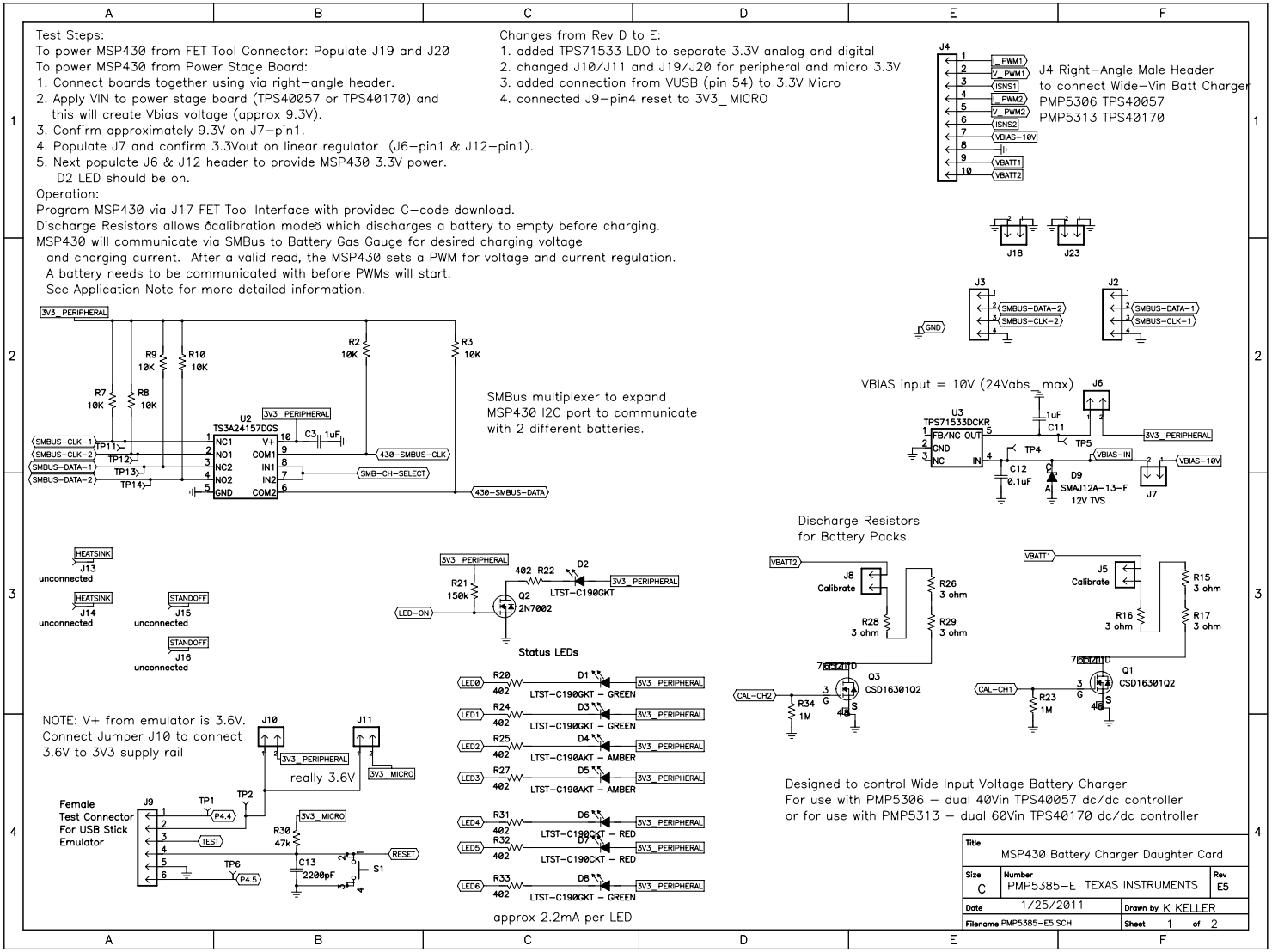


Figure 11. MSP430F5510 Daughterboard Schematic (Page 1)

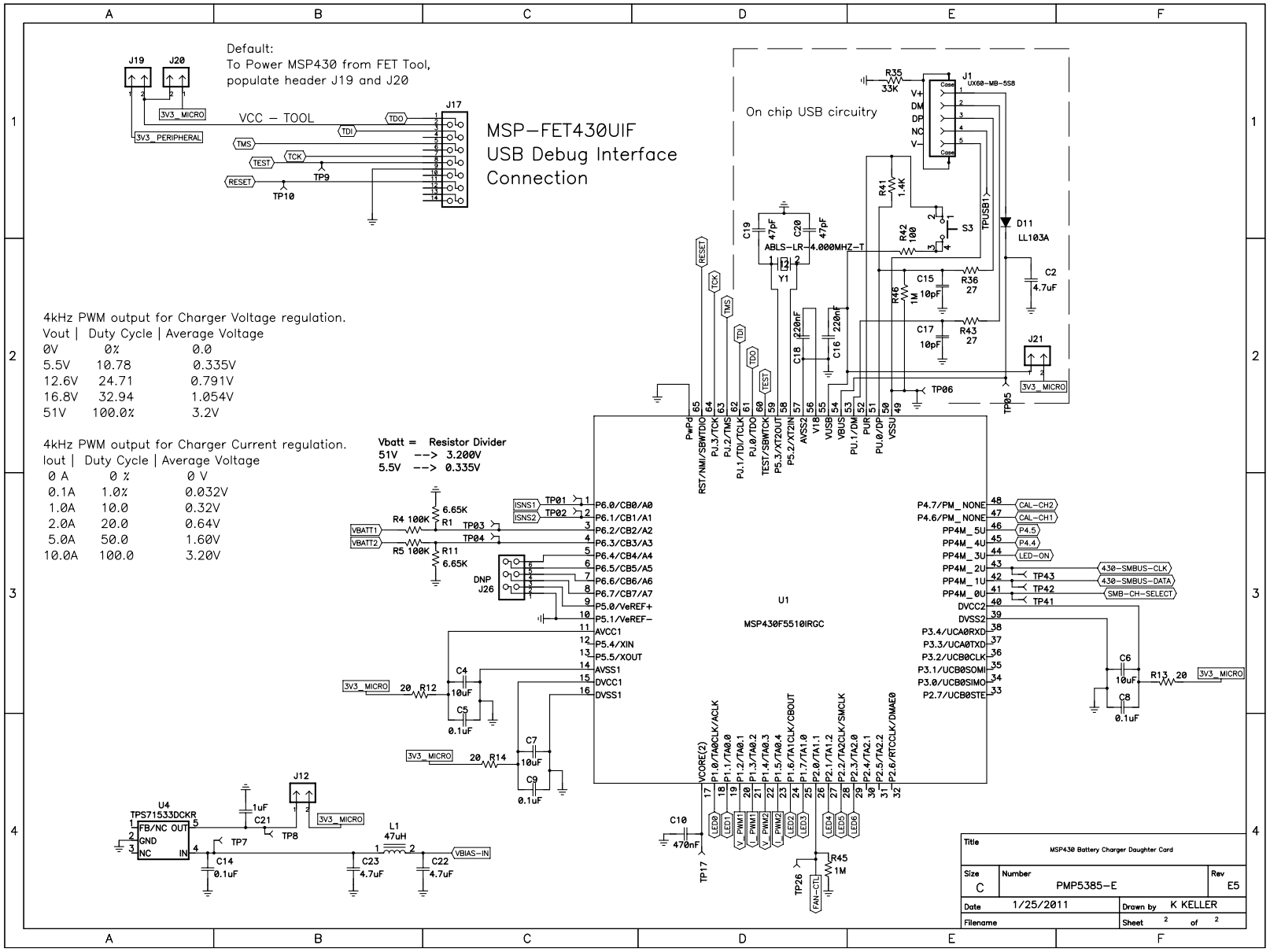


Figure 12. MSP430F5510 Daughterboard Schematic (Page 2)

8 Setting Up the MSP430F5510 Daughterboard Hardware

There are multiple ways of powering and programming the MSP430F5510 Daughterboard. Detailed explanations and recommendations for MSP430F5510 hardware design can be found in the *MSP430™ Hardware Tools User's Guide* [14].

8.1 JTAG FET Debugger Interface (Power Up, Program and Debug Options)

The MSP-FET430UIF (MSP430 Flash Emulation Tool with USB Debug Interface, referred to as JTAG FET Debugger, uses the four-wire JTAG connection (along with V_{DD} and GND) to power and program the target MSP430 device. When used with an IDE, such as the Code Composer Studio™ IDE or IAR Embedded Workbench™ IDE, the JTAG FET debugger allows the target voltage to be programmed, multiple hardware and software breakpoints to be set, and extensive debugging to be performed.

The following steps should be performed with the JTAG FET debugger to power and program the board:

1. Populate jumpers J19 and J20. This connects the V_{CC} power rail from the FET debugger to the 3V3 power rail of the board.
2. Connect the 14-pin female dual-row ribbon cable from the target end of the FET debugger to jumper J17 of the board.

The power indicator LED D2 should light up. Pressing the Debug button in the IDE environment downloads the program on the hardware for debug and testing.

8.2 eZ430 Emulator Interface (Power Up, Program and Debug Options)

NOTE: This procedure works only with the eZ430 emulator (Black) supplied with the eZ430-Chronos™ software development tool. eZ430 emulators supplied with eZ430-F2013 (Blue) and eZ430-RF2500 (Red) are NOT able to program this daughterboard, as they do not contain the firmware for programming 5xx devices. Only the black eZ430-Chronos emulator can program this daughterboard out-of-the-box.

The eZ430 emulator Interface uses the Spy-Bi-Wire JTAG protocol to power and program the hardware. It uses only two wires, along with V_{DD} and GND, for programming the target MSP430 device. The target voltage, however, is fixed at 3.6 V and there is limited debug capability when compared to the FET JTAG debugger. Nonetheless, the small USB stick form-factor of a six-pin header with low cost has made this emulator very popular.

The following steps should be performed with the eZ430 emulator interface to power and program the board:

1. Populate jumpers J10 and J11. This connects the 3.6-V power rail from the FET Debugger to the 3.3-V power rail of the board. In other words, the daughterboard operates at 3.6 V, instead of 3.3 V.
2. Connect the 6-pin male header of the eZ430 USB stick Emulator to Jumper J9 of the board.

Power indicator LED D2 should light up. Pressing the Debug button in the IDE environment downloads the program on the hardware for debug and testing.

8.3 Power Stage Board (Power Up Option Only)

When debugging is not necessary and the whole system needs to run independently, the MSP430F5510 daughterboard can be powered directly from the power-stage board with the following steps:

1. Connect the right-angle male header (J4) to the corresponding female header on the Power Stage Board.
2. Apply VIN to power-stage board to create the VBIAS voltage (approximately 10 V).
3. Confirm approximately 10 V on Jumper J7, pin1.
4. Populate jumper 7 to supply the VBIAS-IN to the TPS71533 3.3-V LDO. Confirm 3.3-V output on jumper 6, pin 1 and on jumper 12, pin 1.
5. Populate jumper J6 to supply 3.3 V to daughterboard peripherals.
6. Populate jumper J12 to supply 3.3 V to the MSP430F5510 microcontroller.

Power indicator LED D2 should light up, and the software should start up and begin execution.

9 Battery Calibration Circuit Setup

The MSP430F5510 daughterboard hardware has power resistors that can be connected to battery terminals for discharging. The discharge circuitry can be turned on or off by the microcontroller to calibrate battery pack voltages. There are two independent circuits for calibrating two batteries simultaneously.

- To enable the calibration circuit for Battery 1, populate J5, pins 1 and 2.
- To enable the calibration circuit for Battery 2, populate J8, pins 1 and 2.

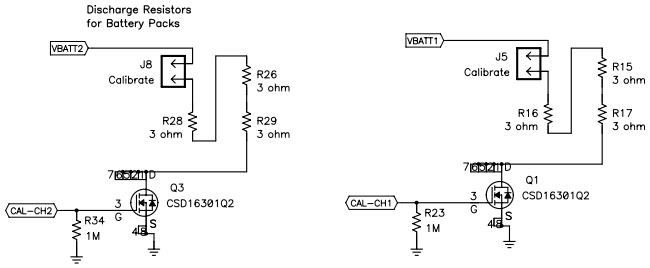


Figure 13. Battery Calibration Circuit Setup

To enable the power FETs, the respective calibration signal from the microcontroller must be set to a logic high (CAL-CH1 or CAL-CH2). This task can be easily accomplished by calling the *Calibrate_Battery()* function.

10 Battery Voltage and PWM Conversions

Table 3 represents the mapping between the MSP430F5510 PWM duty cycle and the voltage output from the DC/DC converter. The PWM frequency is 20 kHz and modifying the PWM duty cycle for the voltage input changes the voltage output of the DC/DC converter. Pulling the PWM signals LOW causes the DC/DC converter to switch off and stop delivering power to the battery.

Table 3. Battery Voltage and PWM Conversions

Duty Cycle (%)	PWM Average Voltage (V)	Average Output Voltage (V)
0	0	0
10.78	0.335	5.5
24.71	0.791	12.6
32.94	1.054	16.8
68.6	2.196	35 (maximum allowed with TPS40054 Power Stage Board)
100.0	3.200	51 (maximum allowed with TPS40170 Power Stage Board)

Sample Calculation Example:

- Desired charger output voltage (V_{out}) = 16.8 V
- Duty cycle (based on 51 V maximum output voltage) = 16.8 V/51 V = 0.3294 (32.94%)
- Desired average voltage of PWM = Duty cycle × 3.2 V = 1.054 V, where 3.2 V is the MSP430F5510 supply range (and also the conversion range for the ADC10)

11 Battery Current and PWM Conversions

Table 4 represents the mapping between the MSP430F5510 PWM duty cycle and the current output from the DC/DC converter. The PWM frequency is 20 kHz and modifying the PWM duty cycle for the current input changes the current output of the DC/DC converter. Pulling the PWM signals LOW causes the DC/DC converter to switch off and stop delivering power to the battery.

Table 4. Battery Current and PWM Conversions

Duty Cycle (%)	PWM Average Voltage (V)	Average Output Current (A)
0	0	0
1.0	0.033	0.1
10.0	0.33	1.0
20.0	0.67	2.0
50.0	1.67	5.0
100.0	3.3	10.0

Sample Calculation Example:

- Desired charger output current (I_{out}) = 1 A
- Duty cycle (based on 10-A maximum output current) = 1 A / 10 A = 0.1 (10%)
- Desired average voltage of PWM = Duty cycle × 3.3 V = 0.330 V, where 3.3 V is the MSP430F5510 supply range (and also the conversion range for the ADC10)

12 Power Stage Board Schematics (Generation 1: 40-V Input)

Figure 14 and Figure 15 show the schematics of the 40-V tolerant input power-stage board (PMP5306) with the TPS40057 DC/DC controller.

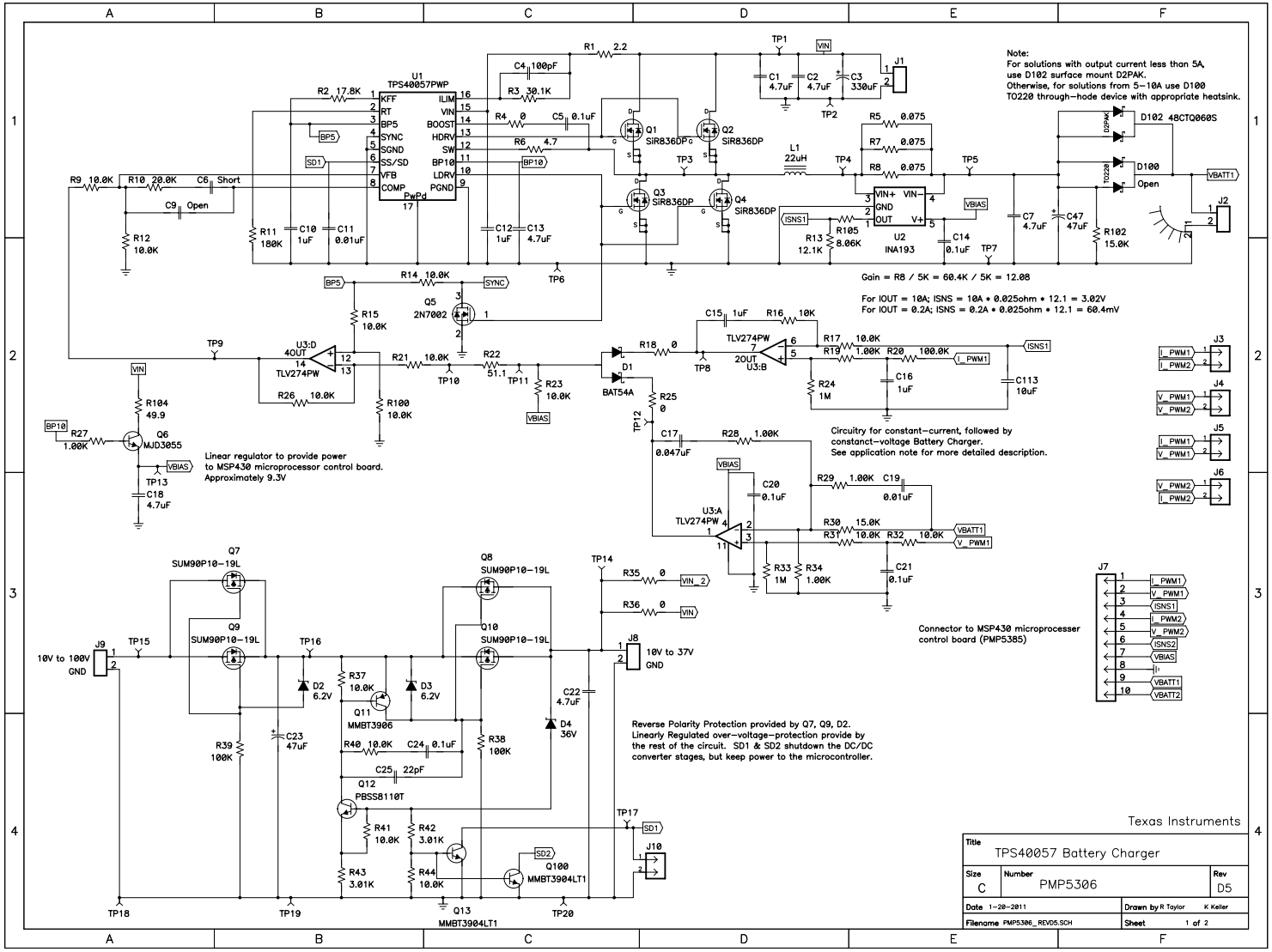


Figure 14. 40-V Input Power Stage Board Schematic (Page 1)

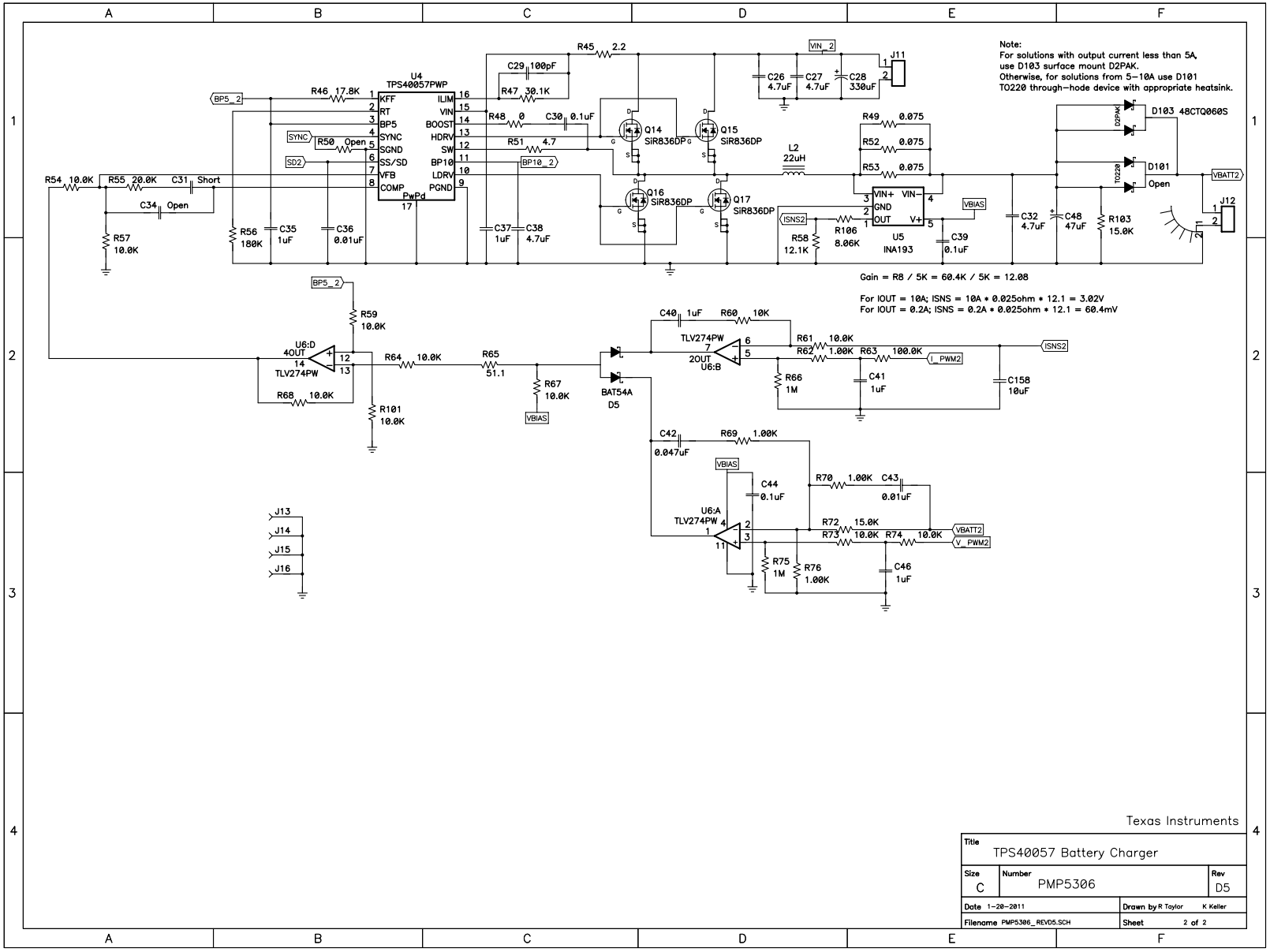


Figure 15. 40-V Input Power Stage Board Schematic (Page 2)

13 Bode Plot Measurement for Feedback Loop Stability Analysis

Figure 16 shows the bode plot measurement for feedback loop stability analysis. The gain is represented on the left side of the phase on the right side of the y-axis.

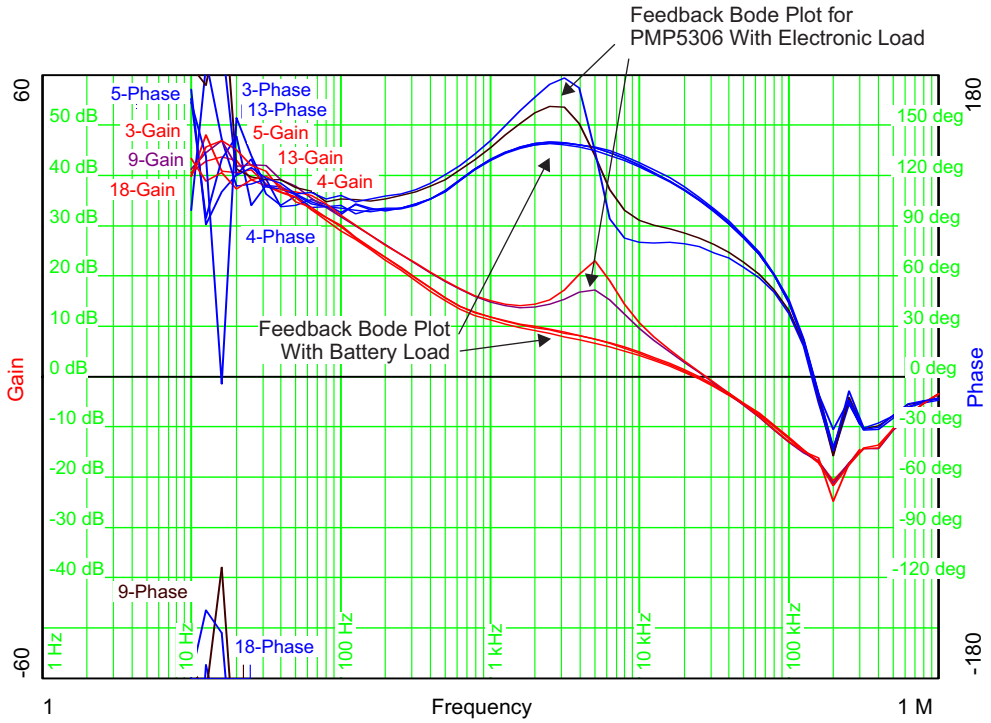


Figure 16. Bode Plot Measurement Graph - Gain (left) and Phase (right)

14 Power Stage Board Schematics (Generation 2: 60-V Input)

Figure 17 and Figure 18 show the schematics of the 60-V tolerant input power-stage board (PMP5313) with the TPS40170 DC/DC controller.

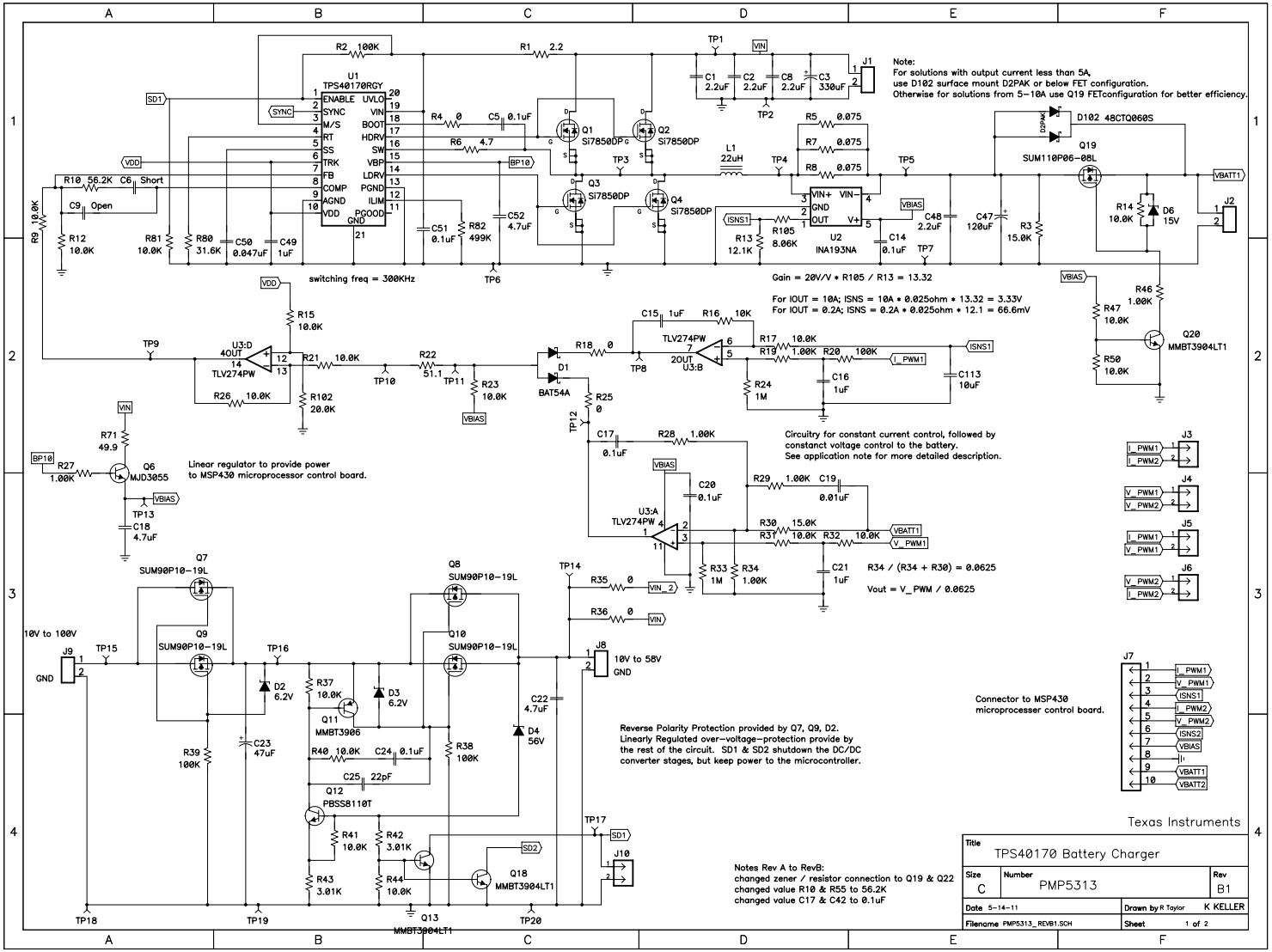


Figure 17. 60-V Input Power Stage Board Schematic (Page 1)

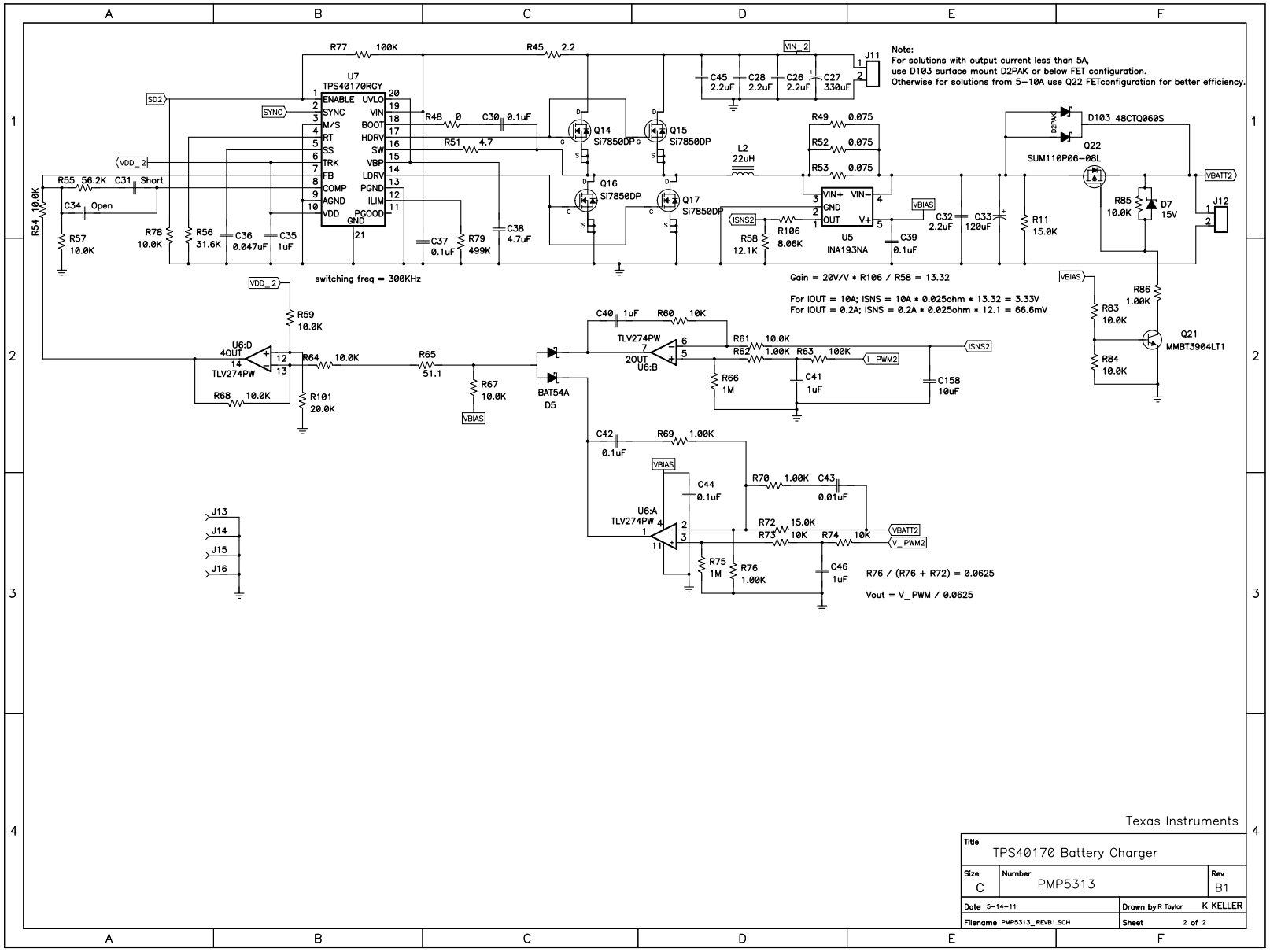


Figure 18. 60-V Input Power Stage Board Schematic (Page 2)

15 Setting Up the Power Stage Board Hardware

The power-stage board can be set up in the following steps:

- Each DC/DC converter stage can be tested independently. If both stages are being used, each stage switches 180° out-of-phase to lower the input capacitor current ripple.
- J1 is the input connector to the first power stage buck converter. Apply a voltage to J1 that is higher than the battery voltage to charge including 15% overhead for the duty cycle limitation of the converter.
- The 20-kHz PWM inputs from the MSP430F5510 microprocessor board feed into I_PWM1 and V_PWM1. Analog voltages could also be fed onto these pins to control the output voltage and current of the DC/DC controller.
- A protection diode is used on the output of each power stage to protect the converter and prevent discharge of the battery. For output currents less than 5 A, the surface mount output diode can be used. For current levels from 5 to 10 A, heatsinking is required to remove heat from the TO-220 diode.
- To test the reverse polarity and overvoltage protection, apply the input voltage to J9. The voltage on J8 feeding VIN and VIN_2 is limited by the breakdown voltage of zener diode D4. In an overvoltage condition, both DC/DC converters are shut off by signals SD1 and SD2 but continue to power the microprocessor though the Vbias voltage created by Q6 discrete linear regulator.

16 References

1. *Quick Start Guide for bq20zxx Family Gas Gauges*
2. *MSP430F5xx and MSP430F6xx Family User's Guide*
3. System Management Bus (SMBus) Specification v2.0, Aug 2000 (http://www.smbus.org)
4. I^2C-Bus Specification and User Manual, Rev. 03, Jun 2007 (http://www.i2c-bus.org/)
5. *MSP430F550x Mixed-Signal Microcontrollers data sheet*
6. *bq20z90 SBS 1.1-Compliant Gas Gauge Enabled With Impedance Track™ Technology for Use With the bq29330 data sheet*
7. *PMM, UCS, Port Mapping, and Flash Libraries for the MSP430x5xx and MSP430x6xx Devices*
8. *CRC Implementation With MSP430 MCUs*
9. *bq20z90EVM-001 SBS 1.1 Impedance Track™ Technology Enabled Battery Management Solution Evaluation Module*
10. *bq20z90-V1.50 + bq29330, bq20z95 Technical Reference*
11. *EV2300 Evaluation Module Interface Board User's Guide*
12. *bq20z90EVM-001 – bqEV-Easy-SW Setup Evaluation Software for Windows*
13. *bqEASY Evaluation Software User's Guide*
14. *MSP430™ Hardware Tools User's Guide*
15. *TS3A24157 0.65-Ω Dual SPDT Analog Switch Dual-Channel 2:1 Multiplexer/Demultiplexer data sheet*
16. *TPS715xx 50 mA, 24 V, 3.2-µA Supply Current Low-Dropout Linear Regulator in SC70 Package data sheet*
17. *TPS79801-Q1, TPS79850-Q1 50 mA, 3V to 50 V, Micropower, Low-Dropout Linear Regulator data sheet*
18. *TPS40054, TPS40055, TPS40057 Wide-Input Synchronous Buck Controller data sheet*
19. *TPS40170 4.5-V to 60-V Wide-Input Synchronous PWM Buck Controller data sheet*
20. *INA193, INA194, INA195, INA196, INA197, INA198 Current Shunt Monitor -16 V to + 80 V Common-Mode Range data sheet*

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from July 26, 2011 to July 12, 2019	Page
• Editorial and formatting updates throughout document	1

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265