TPS62260LED-338 Three-Color LED Driver Evaluation Module (EVM)

www.ti.com/led

User's Guide

Literature Number: SLVU240B
May 2008–Revised August 2018

Contents

List of Figures

List of Tables

Read This First

About This Manual

This user's guide describes the characteristics, setup, and use of the TPS62260LED-338 three-color light-emitting diode (LED) driver evaluation module (EVM). This EVM contains three TPS62260 2.25-MHz, 600-mA step-down voltage converters and an MSP430F2131 microcontroller (MCU). Each TPS62260 applies power to one of the three high brightness LEDs (red, green, or blue). The MSP430F2131 controls the output current of the three converters individually.

How to Use This Manual

This document contains the following sections:

- Chapter 1 – Introduction
- Chapter 2 – Setup
- Chapter 3 – Supported Colors and Operation Modes
- Chapter 4 – Design Description
- Chapter 5 – Schematic and Bill of Materials
- Chapter 6 – Board Layout
- Appendix A – Reprogramming

Read Chapter 2 before connecting the board to a power supply for the first time.

Information About Cautions and Warnings

This user's guide may contain cautions and warnings.

CAUTION

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

WARNING

This is an example of a warning statement.

A warning statement describes a situation that could potentially cause harm to you.

The information in a caution or a warning is provided for your protection. Read each caution and warning carefully.

Related Documentation From Texas Instruments

TPS6226x 2.25-MHz 600-mA Step Down Converter in 2 x 2 WSON and SOT Package data sheet

MSP430x2xx Family User's Guide

MSP430F21x1 Mixed-Signal Microcontrollers data sheet

MSP430F2131 Device Erratasheet

If You Need Assistance

Contact your local TI sales representative or ask a question on the TI E2E™ Community forums.

Trademarks

Introduction

This TPS62260LED-338 EVM enables the user to individually control the brightness of three high-brightness LEDs. By adjusting the relative brightness of each of the three LEDs, the user can generate a wide range of colors across the spectrum.

The desired output color can be adjusted by means of a rotary encoder. Alternatively, the board can be left in its default power-up state, in which it continuously cycles through the whole range of colors.

For more details on the operation of the board, refer to Chapter 3.

1.1 Requirements

To operate this EVM, it has to be supplied with a voltage between 3.6 V and 6 V. The current capability of the power supply should be 1 A or higher. The EVM kit contains everything necessary to operate the EVM except the DC power supply.

Reprogramming of the MSP430F2131 microcontroller can be done by connecting an MSP430™ JTAG interface board (for example, the MSP-FET MSP MCU Programmer and Debugger). This MSP430 development tool is not included in the TPS62260LED-338 EVM.

1.1.1 Power Supply Requirements

The EVM requires a regulated DC power supply that can deliver 3.6 V to 6 V at 1 A.

CAUTION

Use of a poorly regulated AC adapter may generate overvoltage conditions exceeding the absolute maximum ratings of some of the components used. It is recommended that only well-regulated (5 V ± 0.5 V) adapters be used with this EVM.

CAUTION

AC adapters with long cables (approximately 1 m or longer) can exhibit significant stray capacitance that may interact with the EVM input capacitance and cause ringing when hot plugged. To avoid such potentially damaging overvoltage conditions, first plug the adapter DC plug into J3, then connect the AC adapter to the mains.

1.1.2 Printed Circuit Board Assemblies (PCBs)

The EVM comprises a single PCB featuring four connectors, with one of them designed to accommodate an optional low-power wireless interface. The wireless interface connection is not part of this EVM. For more details, refer to Section 2.1.2.

Setup

This chapter explains the function of the connectors on the PCB as well as how to properly connect, set up, and use the TPS62260LED-338 EVM.

2.1 Input/Output Connector Descriptions

2.1.1 J1, J2, and J3 – Power Supply Connectors

J1 and J2 are two-pin headers for easy connection of a lab power supply. Connect a positive input voltage between 3.6 V and 6 V to J1 and 0 V (ground) to J2. The lab power supply current limit has to be set to at least 1 A.

J3 can be connected with a 5-mm × 2.5-mm barrel connector of the type commonly used with low-cost AC adapters. The inner contact is positive (5 V) relative to the outer contact (GND). Possible wall adapters could be Egston P2CFSW3–5 V/1.2 A, 5.0 VDC, 1.2 A with universal connector set (make sure that the orientation of the connector is correct) or SL Power Electronics PW170KA05V/2A with the right country connector.

2.1.2 JP1 – Wireless Interface Connector

The JP1 is a pin header that can be used for plug-on of the RF board from the eZ430-RF2500 kit, which is separately available. With this additional module, the colors of the lamp can be controlled remotely through the wireless RF interface.

NOTE: The firmware included in the standard EVM does not currently support wireless modules. Users who wish to use wireless are required to write their own (simple) routines. Different software libraries support the realization of star-networks or point-to-point wireless connections using MSP430 microcontrollers. These libraries include SimpliciTi (star network) or MSP430 and CC2500 code library (point-to-point connection).

NOTE: When the wireless module is connected to the standard EVM, the power requirement increases. To support this higher power requirement, change resistor R11 to a 0805 68-Ω resistor.

2.1.3 JP2 – JTAG Interface Connector

The JTAG connector JP2 is used as the programming interface for the MSP430F2131 microcontroller. Only MSP430 JTAG programming adaptors (like the MSP-FET MSP MCU Programmer and Debugger) can be used for the program download. This tool also allows debugging of a new MSP430 software.

The connector on the TPS62260LED-338 EVM uses standard pinning also used on the different MSP430 JTAG tools. This means that the cable delivered with MSP-FET can be plugged into the JP2 connector of the TPS62260LED-338 EVM.

NOTE: Programming and debugging with the MSP430 JTAG interface tools works only if the TPS62260LED-338 EVM is supplied by a 5-V power supply.

The MSP430 JTAG programming adaptor is not part of this package but can be ordered from the TI Store. For instructions to set up the programming adapter and update the program already installed on the MSP430F2131, see Appendix A.

2.2 Hardware Setup

WARNING

Turning on the power supply lights the high-brightness LEDs on the board, starting with blue and cycling automatically through the color range.

Protective eyewear is recommended!

To set up the hardware:

- Plug the DC connector of the AC adapter into J3 or connect a lab power supply set to 3.6 V to 6 V between J1 and J2.
- Plug the mains connector of the AC adapter into the mains supply.
- Turn on the mains supply.

CAUTION

Use of a poorly regulated AC adapter may generate overvoltage conditions exceeding the absolute maximum ratings of some of the components used. It is recommended that only well-regulated (5 V ± 0.5 V) adapters be used with this EVM.

CAUTION

AC adapters with long cables (approximately 1 m or longer) can exhibit significant stray capacitance that may interact with the EVM input capacitance and cause ringing when hot plugged. To avoid such potentially damaging overvoltage conditions, first plug the adapter DC plug into J3, then connect the AC adapter to the mains.

Supported Colors and Operation Modes

This chapter explains the two control modes of the TPS62260LED-338 EVM. If the LEDs are controlled in a different way, specialized software can be downloaded through the JTAG connector JP2 to the MSP430F2131. For the download process, refer to Appendix A.

3.1 Color Range

The TPS62260LED-338 EVM with the default software does not support the entire range of colors described by the CIE chromaticity diagram, but instead supports a reduced subset. This approach greatly reduces the design effort while simultaneously providing a range of possible colors spanning the spectrum. Figure 3-1 shows the CIE chromaticity diagram representing the range of possible colors; the inner triangle represents the reduced range of colors supported by the EVM with the default software.

When the EVM scrolls through its range of colors, it effectively traces the edges of the inner triangle shown in Figure 3-1. As can be seen from this triangle, when tracing the edges clockwise this color transitions from blue to green, through yellow and orange to red, and then from red through purple back to blue.

3.2 Auto-Scroll Mode

The EVM powers up in auto-scroll mode starting with blue. In auto-scroll mode the MSP430 autonomously cycles clockwise through the range of supported colors ad infinitum.

After switching to manual control mode by using the rotary encoder, auto-scroll mode can be entered by powering up the board again.

3.3 Manual Control Mode

The EVM leaves auto-scroll mode and enters manual control mode the first time the rotary control is activated. In manual control mode, the color balance of the three LEDs is adjusted by the user turning the rotary encoder S1. The faster the encoder is turned, the faster the colors transition from one to another; the direction of rotation determines whether the EVM traces the color triangle in Figure 3-1 clockwise or counterclockwise.

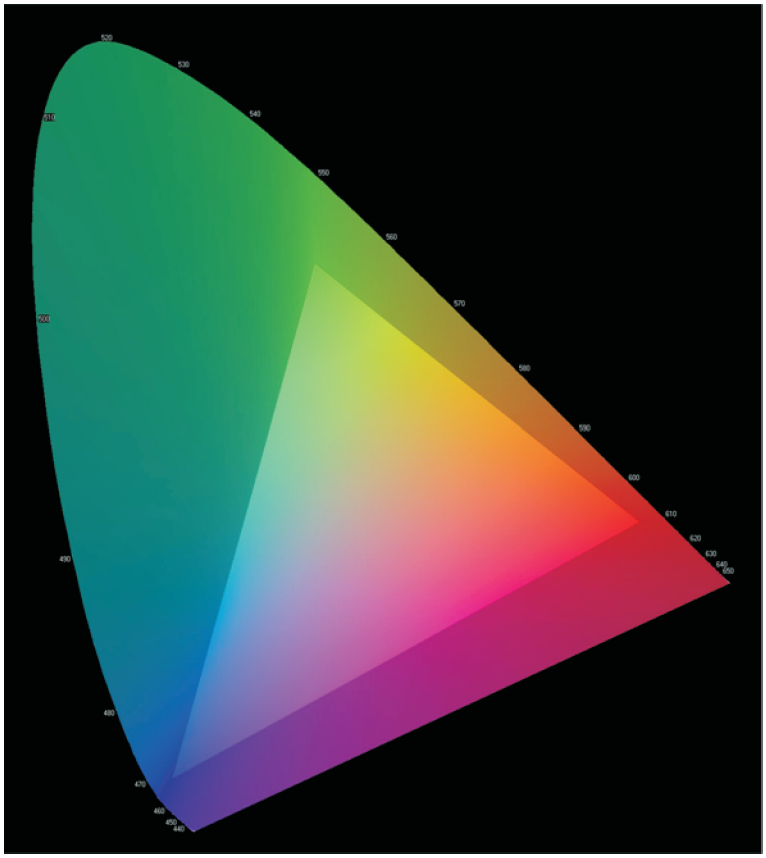


Figure 3-1. CIE Chromaticity Diagram

Design Description

This chapter describes the design steps to build the hardware and firmware for the TPS62260LED-338 EVM. This chapter includes the board description and the detailed instructions used in the firmware program loaded by default onto the MSP430F2131.

4.1 Hardware Design

The schematics and bill of materials referred to in the following design description are contained in Chapter 5. The design contains three identical LED driver stages; only one (the red channel) is described.

4.1.1 LED Power Stages

Because the brightness of an LED is determined by the current flowing through it, not the voltage across it, LEDs tend to be powered by current sources in all the simplest of applications (that is, when uniform intensity and color balance are not important). In this application, each power stage uses a TPS62260 DC/DC converter configured as a controllable current source. Instead of using a resistor divider between the output and GND to generate the feedback voltage, a small current-sensing resistor is inserted between the LED cathode and GND. In this configuration, the TPS62260 controls its duty cycle at whatever value is needed to regulate the voltage across the current-sensing resistor to 0.6 V (the internal reference voltage of the IC). Using a 2-Ω current-sensing resistor, the LED current is therefore regulated to a value given by:

$$I_{LED} = \frac{V_{FB}}{R_{SNS}} = \frac{0.6\ V}{2\ \Omega} = 300\ mA \quad (1)$$

Brightness is varied by pulse width modulating the current flowing through each LED. This approach has two main advantages compared with using analog methods to control LED current.

First, the color balance of an LED changes with the current flowing through it, so not only is the brightness of an LED at 10 mA different than at 100 mA, its color is, too. With pulse width modulation (PWM) dimming, the current flowing through the LED when it is active is always the same so its color does not change. As long as the dimming frequency is high enough, the only effect the human eye sees is a variation in LED intensity. In this application, a nominal dimming frequency of 122 Hz is used.

Second, it is simpler and cheaper to generate three PWM signals using a microcontroller than three analog voltages, which would require a 3-channel DAC.

The PWM dimming scheme works as follows:

- When the NET_DIMM_LED1 signal is low, D1 blocks any current flow away from the FB pin and U2 regulates LED current to its full-scale value of 300 mA (see Equation 1).
- When NET-DIMM_LED1 is high (close to the 3.3-V supply voltage of U2) the U2 FB pin is held at 1.35 V, which forces the duty cycle of U2 and consequently the LED current to zero (see Equation 2).

$$V_{FB} = V_{R9} + (V_{CC} - V_{D1} - V_{R9}) \times \left(\frac{R_7}{R_7 + R_5}\right)$$

$$V_{FB} = V_{R9} + (3.3\ V - 0.6\ V - V_{R9}) \times \left(\frac{10\ k\Omega}{10\ k\Omega + 10\ k\Omega}\right)$$

$$V_{FB} = V_{R9} + \frac{2.7\ V}{2} - \frac{V_{R9}}{2}$$

$$V_{FB} = \frac{V_{R9}}{2} + 1.35\ V \quad (2)$$

The average current flowing through the LED and its brightness are simply the products of the full-scale (FS) LED current multiplied by the duty cycle of the dimming signal (NET_DIMM_LED1):

$$I_{LED(AVG)} = I_{LED(FS)} \times D_{NET\_DIMM\_LED1}$$

$$I_{LED(AVG)} = 300\ mA \times D_{NET\_DIMM\_LED1} \quad (3)$$

4.1.2 Output Filter Design

The TPS62260 data sheet states that the part is optimized for use with an output filter comprising a 2.2-µH inductor and a 10-µF capacitor. In this application, the standard inductor value was used. However, because output voltage ripple is uncritical (at 2.5 MHz the human eye detects no worsening of performance) a smaller capacitor value of 4.7 µF was used to reduce cost.

4.1.3 MODE and EN Pins

The TPS62260 features a power-save mode to improve efficiency at low output powers, but it is not needed in this application. By connecting the U2 MODE pin to VIN, this feature is disabled, and the device operates permanently in PWM mode.

All three LED driver circuits are connected to a common enable signal (NET_EN) that allows the MSP430 MCU to completely enable and disable the power stages if necessary. A pulldown resistor is used for the TPS62260 enable signal. This pulldown resistor causes all LEDs to switch off after the supply voltage is applied to the EVM. The MSP430 MCU activates the LEDs and avoid flashing LEDs during startup.

4.1.4 MSP430 MCU Design

The MSP430 MCU is powered through a simple (and inexpensive) 3.3-V Zener diode linear regulator. Resistor R11 sets the D5 bias current to approximately 5 mA, which is significantly more than the current drawn by the MSP430. This maintains good regulation in the face of changing load currents. Because the input supply to the board is regulated to 5 V, the Zener regulator circuit experiences almost no input voltage variation.

The rotary encoder S1 allows manual control of the color interfaces of the lamp to the MSP430 MCU using two digital input pins. When rotated, this type of encoder generates two pulse trains 90 degrees out of phase. The number of turns can be determined by counting the number of pulses generated, the direction of rotation can be determined by comparing the relative phase of the two signals, and the speed of rotation can be determined by measuring the frequency of the pulses. This can be easily achieved using one of the built-in timers of the MSP430 MCU.

R4 holds the three LED power stages in a disabled state until the MSP430 has powered up and is ready to assume control. R1 and C1 provide a power-up reset for the MSP430. JP2 provides a JTAG interface to allow easy debugging and JP1 provides the connections to the optional low-power wireless interface.

4.2 LED Color Table

To obtain the correct optical response from the LEDs, their relative brightness (that is, the current flowing through them) is not varied linearly, but is varied according to a lookup table derived from the CIE chromaticity diagram. This lookup table is stored in the flash memory of the MSP430 MCU and defines the edges of the inner triangle shown in Figure 3-1, which the EVM traces. The standard EVM lookup table contains 252 locations, each of which contains the correct value for the red, green, and blue LED PWM signals.

4.3 Firmware Design

This section details the function of the default software loaded onto the MSP430F2131. Because the color scheme is set through integer values given in three LED color arrays, no software change is needed for changing the color scheme. Changing the values in the LEDx[ ] arrays changes the colors.

```
#include "msp430x21x1.h"
```

All peripheral control registers and control bits of the MSP430F2131 are defined in the header file msp430x21x1.h.

```
#define LED_TabLength 252*4
```

Here the length of the arrays LED1[ ], LED2[ ], and LED3[ ] is defined. It is used to detect the overflow of the LEDptr. Care should be taken that all three arrays LED1[ ], LED2[ ], and LED3[ ] have the length that is defined here.

```
const unsigned int LED1[]={65385,65385,65385, ¼,65385,65385}; //blue LED
const unsigned int LED2[]={ 150, 295, 622, ¼, 150, 150}; //green LED
const unsigned int LED3[]={ 150, 150, 150, ¼, 311, 150}; //red LED
```

The three arrays are used for the PWM duty cycle adjustment. For each LED (red, green, and blue) there is an own array. But there is only one pointer (LEDptr) that is used to find the PWM settings for each of the arrays. The values of the array should be within the range 100 to 65535.

```
unsigned int LEDptr;
```

This is the variable used as the pointer for the three arrays LED1[ ], LED2[ ], and LED3[ ].

```
unsigned char BAold;
```

This variable is used for the detection of a change of the incremental encoder.

```
void main(void)
{ unsigned int i,temp;
WDTCTL=WDTPW+WDTHOLD; // disable Watchdog
The Watchdog is not used in this program. So it is disabled.
BCSCTL1= CALBC1_8MHZ; //--- System Clock Settings ----------------------
DCOCTL = CALDCO_8MHZ; // use calibrated 8MHz settings
```

The Watchdog is not used in this program. By setting the control bit WDTHOLD in control register WDTCTL the Watchdog is disabled.

```
BCSCTL1= CALBC1_8MHZ; //--- System Clock Settings ----------------------
DCOCTL = CALDCO_8MHZ; // use calibrated 8MHz settings
```

There are calibration values available in the flash memory of the MSP430 MCU. These two commands move the calibration value for 8-MHz DCO output frequency into the clock system control registers.

```
//---- PWM Timer Initialization ------------------
TACTL = TASSEL_2+ID_0+MC_0+TACLR+TAIE; // Timer clock = SMCLK = 8MHz
TACCTL0 = CM_0+CCIS_2+OUTMOD_1; // All Output Units will set PWM outputs if
TACCTL1 = CM_0+CCIS_2+OUTMOD_1; // TACCRx=TAR. Resetting PWM outputs is done
TACCTL2 = CM_0+CCIS_2+OUTMOD_1; // by software.
```

The Timer_A module is initialized here. It uses the calibrated 8-MHz DCO clock signal. A timer overflow generates an interrupt. The three capture and compare blocks CCR0, CCR1, and CCR2 are used in compare mode. The output unit of each CCR block is used to generate a PWM signal. The output units are automatically setting the PWM output, while the resetting of the output signal is done by software as soon as a timer overflow (Timer_A interrupt) happens.

```
LEDptr=0;
TACCR0=LED1[LEDptr>>2]; // LEDptr is shifted right twice,
```

```
TACCR1=LED2[LEDptr>>2]; // this means divided by 4
TACCR2=LED3[LEDptr>>2];
```

LEDptr is cleared. This means the first values of the arrays LED1[], LED2[], and LED3[] are used at the beginning. When LEDptr is used for the array, it is divided by 4 (shifting LEDptr two times right is the same as divided by 4). This is done to avoid issues with the bouncing of the incremental encoder.

```
//--- Port Initialization ---------------------------------
P1SEL = 0x0E; // P1.1, P1.2, P1.3 are used as PWM Timer Outputs
P1OUT = 0x00; // P1.0 is output (Enable for TPS62260)
P1DIR = 0xFF; // P1.4, P1.5, P1.6, P1.7 are not used => digital outputs
P2OUT = 0x04; // P2.0 and P2.1 are not used => digital outputs
P2DIR|= 0xE4; // P2.3, P2.4 are digital inputs => incremental encoder
// P2.5, P2.6, P2.7 are not used => digital outputs
```

Initialization of the digital I/Os. P1.1, P1.2, and P1.3 are used as Timer_A PWM output (module function). All pins that are not used are defined as digital outputs.

```
BAold=0x01; //--- initialize decoder for incremental encoder -----------
```

Initialization of the variable used for incremental encoder detection.

```
Delay(); // Delay loop
TACTL |= MC_2; // start Timer_A (continuous mode)
```

After a short delay loop the Timer_A is started. The Timer_A is used in continuous mode, this means it counts from 0 to 65535. If the counter is 65535 and the timer gets another clock the counter value is set to 0 and an overflow interrupt is generated.

```
__enable_interrupt(); // enables maskable interrupts
```

All maskable interrupts are enabled. Now the interrupt service routines are called if there is an interrupt event.

```
temp=P2IN&0x18;
//--- Main Loops ------------------------------------------------------------
while ((P2IN&0x18)==temp) //--- change settings automatically till
```

This is the first operating mode of the application. It stays in this loop as long as pins P2.3 and P2.4 do not change (this means as long as the incremental encoder S1 was not used).

```
{ Delay(); // incremental encoder is operated
LEDptr=LEDptr+1;
if (LEDptr>=LED_TabLength)
LEDptr=0;
}
```

The first operating mode of the application automatically changes the LEDptr. This is done by using a simple delay loop and afterwards the LEDptr is incremented. After incrementing the LEDptr it is checked if the maximum table length is reached. If this is the case the LEDptr is reset.

```
while(1) //--- change settings manually (incremental encoder)
{ Inc_Decoder(0x03&(P2IN>>3)); // check incremental decoder
for(i=0;i<=1000;i++); // delay loop (used for debouncing)
}
```

When the incremental encoder S1 is turned, this loop is executed. This is an entire loop. Here the incremental encoder is checked. Afterwards, there is a short delay loop that is used for debouncing of the incremental encoder.

```
}
//-----------------------------------------------------------------------------
// Delay Loop
void Delay(void)
{ unsigned int i,j;
for(i=0;i<=10000;i++) // delay loop
for(j=0;j<=3;j++);
}
```

This is the delay loop that is mainly used for the automatic mode. It defines the time the single settings of the arrays are used before the next one is moved to the Timer_A control registers.

```
//-----------------------------------------------------------------------------
```

```
// Incremental Encoder Subroutine:
void Inc_Decoder(unsigned char BAnew)
{
if (BAnew==0x02)
{ if (BAold==0x00)
{ LEDptr=LEDptr-1; // decrement pointer if new state is 'b' and
if (LEDptr>=LED_TabLength) // old state was 'a'
LEDptr=LED_TabLength;
}
}
if (BAnew==0x00)
{ if (BAold==0x02)
{ LEDptr=LEDptr+1; // increment pointer if new state is 'a' and
if (LEDptr>=LED_TabLength) // old state was 'b'
LEDptr=0;
}
}
BAold=BAnew; // store new state
}
```

The incremental encoder generates a gray code. From the four different states the program is testing for only two states. The two states are P2.3='0', P2.4='0' and P2.3='1', P2.4='0'. Incrementing/decrementing of LEDptr is only done if the state that was read during the previous execution of the Inc_Decoder() function (this was stored in BAold) is different.

```
//-------------------------------------------------------------------------
// Timer_A Interrupt Service Routine:
#pragma vector=TIMERA1_VECTOR
__interrupt void ISR_TimerA(void)
{ P1OUT |= 0x01; // activate LEDs
```

This is the Timer_A interrupt service routine. It is called as soon as there is a Timer_A overflow. The first instruction is setting the P1.0 pin. This enables the DC/DC converters.

```
//--- update PWM duty cycle settings using color table
TACCR0=LED1[LEDptr>>2]; // LEDptr is shifted right twice,
TACCR1=LED2[LEDptr>>2]; // this means divided by 4
TACCR2=LED3[LEDptr>>2];
```

The settings for the PWM duty cycles are read from the three LED arrays. This is done in the interrupt service routine to ensure that it is synchronized with the timer. If this would be done in the main loop, the LED would flicker in case of a change.

The PWM signal is generated from the Timer Capture and Compare Block, that is, a digital comparator compares the Control Register (TACCRx) with the actual timer value. Are both values identical, the PWM output is set. The reset of the PWM output has to be done by software. This is done with an interrupt when there is a timer overflow and its Interrupt Service Routine updates the TACCRs register if necessary.

If this is done in the main loop, the update could happen asynchronous to the timer. Then it could happen that for one PWM cycle the PWM signal becomes inverted and the LED flickers.

```
//--- PWM signal generation
TACTL &= ~TAIFG;
TACCTL0 &= ~OUTMOD_7; // OUTMOD_0 => PWM output=L
TACCTL0 |= OUTMOD_1; // OUTMOD_1 => set PWM output TA0 as soon as TACCR0=TAR
TACCTL1 &= ~OUTMOD_7; // OUTMOD_0 => PWM output=L
TACCTL1 |= OUTMOD_1; // OUTMOD_1 => set PWM output TA1 as soon as TACCR1=TAR
TACCTL2 &= ~OUTMOD_7; // OUTMOD_0 => PWM output=L
TACCTL2 |= OUTMOD_1; // OUTMOD_1 => set PWM output TA2 as soon as TACCR2=TAR
// TAR = Timer_A counter
}
```

Finally the Timer_A interrupt flag is cleared and all output signals are reset. The Timer CCR blocks are used to set the TAx output signals. Resetting the TAx outputs has to be done by software. That is done here.

4.3.1 Firmware C-Code Listing

A complete listing of the firmware used in the EVM is contained below:

```
/*****************************************************************************/
/*  RGB-LED Demo using MSP430F2131 and TPS62260 from Texas Instruments       */
/*                                                                           */
/*  Description:                                                             */
/*  Timer_A3 is used to generate 3 PWM signals. Timer overflow generates an  */
/*  interrupt and in its interrupt service routine the PWM outputs are       */
/*  reseted. Output Units will set the PWM outputs.                          */
/*  Software starts in a demo mode that automatically changes the PWM        */
/*  settings. As soon as the incremental encoder is operated the automatic   */
/*  mode stops and the adjustment of the colour can be done manually.        */
/*---------------------------------------------------------------------------*/
/*  Texas Instruments Deutschland GmbH                                       */
/*  Christian Hernitscheck, November 2007                                    */
/*****************************************************************************/
#include "msp430x21x1.h"


#define  LED_TabLength    252*4

const unsigned int LED1[]={65385,65385,65385,65385,65385,65385,65385,65385,
65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,
65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,
65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,
42173,36943,34327,31712,29096,26481,23866,21250,19942,18635,17327,16019,14712,
13404,12096,10789, 9481, 8173, 7519, 6865, 6212, 5558, 4904, 4577, 4250, 3923,
 3596, 3269, 2942, 2615, 2288, 1962, 1635, 1308,  981,  654,  327,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  327,  654,  981, 1308, 1635, 1962, 2288, 2615,
 2942, 3269, 3596, 3923, 4250, 4577, 4904, 5558, 6212, 6865, 7519, 8173, 9481,
10789,12096,13404,14712,16019,17327,18635,19942,21250,23866,26481,29096,31712,
34327,36943,39558,42173,44789,47404,52635,57866,65385,65385,65385,65385,65385,
65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,
65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,
65385,65385,65385,65385,65385,65385,65385,65385,65385,65385,65385};  //blue LED
const unsigned int LED2[]={  150,  295,  622,  949, 1276, 1603, 1930, 2256,
 2583, 2910, 3237, 3564, 3891, 4218, 4545, 4872, 5526, 6180, 6833, 7487, 8141,
 9449,10757,12064,13372,14680,15987,17295,18603,19910,21218,23834,26449,29064,
31680,34295,36911,39526,42141,44757,47372,52603,57834,65353,65353,65353,65353,
65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,
65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,
65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,
65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,
65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,65353,
65353,65353,65353,57834,52603,47372,42141,36911,34295,31680,29064,26449,23834,
21218,19910,18603,17295,15987,14680,13372,12064,10757, 9449, 8141, 7487, 6833,
 6180, 5526, 4872, 4545, 4218, 3891, 3564, 3237, 2910, 2583, 2256, 1930, 1603,
 1276,  949,  622,  295,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150};  //green LED
const unsigned int LED3[]={  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
```

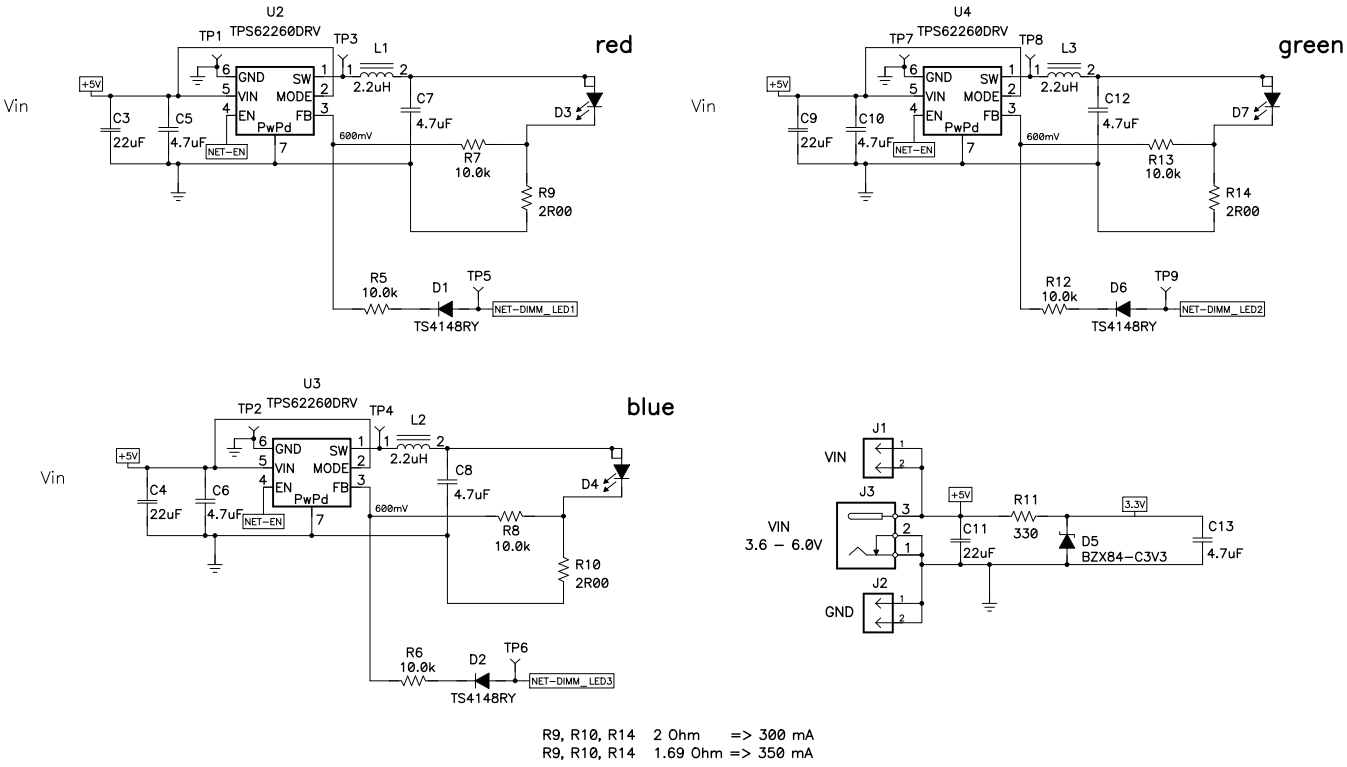
```
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,
  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  150,  311,
  638,  965, 1292, 1619, 1946, 2272, 2599, 2926, 3253, 3580, 3907, 4234, 4561,
 4888, 5542, 6196, 6849, 7503, 8157, 9465,10773,12080,13388,14696,16003,17311,
18619,19926,21234,23850,26465,29080,31696,34311,36927,39542,42157,44773,47388,
52619,57850,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,
65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,
65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,
65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,
65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,
65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,65369,
65369,65369,65369,65369,65369,65369,65369,65369,65369,57850,52619,47388,42157,
36927,34311,31696,29080,26465,23850,21234,19926,18619,17311,16003,14696,13388,
12080,10773, 9465, 8157, 7503, 6849, 6196, 5542, 4888, 4561, 4234, 3907, 3580,
 3253, 2926, 2599, 2272, 1946, 1619, 1292,  965,  638,  311,  150};  //red LED

unsigned int LEDptr;
unsigned char BAold;

void Inc_Decoder(unsigned char BAnew);
void Delay(void);

//-------------------------------------------------------------------------------
// Main Program
void main(void)
{ unsigned int i,temp;

  WDTCTL=WDTPW+WDTHOLD;       // disable Watchdog

  BCSCTL1= CALBC1_8MHZ;       //--- System Clock Settings ----------------------
  DCOCTL = CALDCO_8MHZ;       // use calibrated 8MHz settings

                              //---- PWM Timer Initialization ------------------
  TACTL = TASSEL_2+ID_0+MC_0+TACLR+TAIE;  // Timer clock = SMCLK = 8MHz

  TACCTL0 = CM_0+CCIS_2+OUTMOD_1; // All Output Units will set PWM outputs if
  TACCTL1 = CM_0+CCIS_2+OUTMOD_1; // TACCRx=TAR. Resetting PWM outputs is done
  TACCTL2 = CM_0+CCIS_2+OUTMOD_1; // by software.

  LEDptr=0;
  TACCR0=LED1[LEDptr>>2]; // LEDptr is shifted right twice,
  TACCR1=LED2[LEDptr>>2]; //                  this means divided by 4
  TACCR2=LED3[LEDptr>>2];

                    //--- Port Initialization ---------------------------------
  P1SEL = 0x0E;     // P1.1, P1.2, P1.3 are used as PWM Timer Outputs
  P1OUT = 0x00;     // P1.0 is output (Enable for TPS62260)
  P1DIR = 0xFF;     // P1.4, P1.5, P1.6, P1.7 are not used => digital outputs
  P2OUT = 0x04;     // P2.0 and P2.1 are not used => digital outputs
  P2DIR|= 0xE4;     // P2.3, P2.4 are digital inputs => incremental encoder
                    // P2.5, P2.6, P2.7 are not used => digital inputs

  BAold=0x01;       //--- initialize decoder for incremental encoder -----------

  Delay();          // Delay loop
  TACTL |= MC_2;    // start Timer_A (continuous mode)

  __enable_interrupt();   // enables maskable interrupts

  temp=P2IN&0x18;

//--- Main Loops -----------------------------------------------------------
  while ((P2IN&0x18)==temp)   //--- change settings automatically till
  { Delay();                  //             incremental encoder is operated
    LEDptr=LEDptr+1;
```

```
    if (LEDptr>=LED_TabLength)
      LEDptr=0;
  }

  while(1)                  //--- change settings manually (incremental encoder)
  { Inc_Decoder(0x03&(P2IN>>3));   // check incremental decoder
    for(i=0;i<=1000;i++);          // delay loop (used for debouncing)
  }
}


//-------------------------------------------------------------------------------
// Delay Loop
void Delay(void)
{ unsigned int i,j;
  for(i=0;i<=10000;i++)   // delay loop
    for(j=0;j<=3;j++);

}

//-------------------------------------------------------------------------------
// Incremental Encoder Subroutine:
void Inc_Decoder(unsigned char BAnew)
{
  if (BAnew==0x02)
  { if (BAold==0x00)
    { LEDptr=LEDptr-1;           // decrement pointer if new state is 'b' and
      if (LEDptr>=LED_TabLength) //    old state was 'a'
        LEDptr=LED_TabLength;
    }
  }
  if (BAnew==0x00)
  { if (BAold==0x02)
    { LEDptr=LEDptr+1;           // increment pointer if new state is 'a' and
      if (LEDptr>=LED_TabLength) //    old state was 'b'
        LEDptr=0;
    }
  }
  BAold=BAnew;                   // store new state
}

//-------------------------------------------------------------------------------
// Timer_A Interrupt Service Routine:
#pragma vector=TIMERA1_VECTOR
__interrupt void ISR_TimerA(void)
{ P1OUT |= 0x01;          // activate LEDs
  //--- update PWM duty cycle settings using colour table
  TACCR0=LED1[LEDptr>>2]; // LEDptr is shifted right twice,
  TACCR1=LED2[LEDptr>>2]; //                this means divided by 4
  TACCR2=LED3[LEDptr>>2];

  //--- PWM signal generation
  TACTL &= ~TAIFG;
  TACCTL0 &= ~OUTMOD_7; // OUTMOD_0 => PWM output=L
  TACCTL0 |= OUTMOD_1;  // OUTMOD_1 => set PWM output TA0 as soon as TACCR0=TAR
  TACCTL1 &= ~OUTMOD_7; // OUTMOD_0 => PWM output=L
  TACCTL1 |= OUTMOD_1;  // OUTMOD_1 => set PWM output TA1 as soon as TACCR1=TAR
  TACCTL2 &= ~OUTMOD_7; // OUTMOD_0 => PWM output=L
  TACCTL2 |= OUTMOD_1;  // OUTMOD_1 => set PWM otuput TA2 as soon as TACCR2=TAR
                        //     TAR = Timer_A counter
}
```

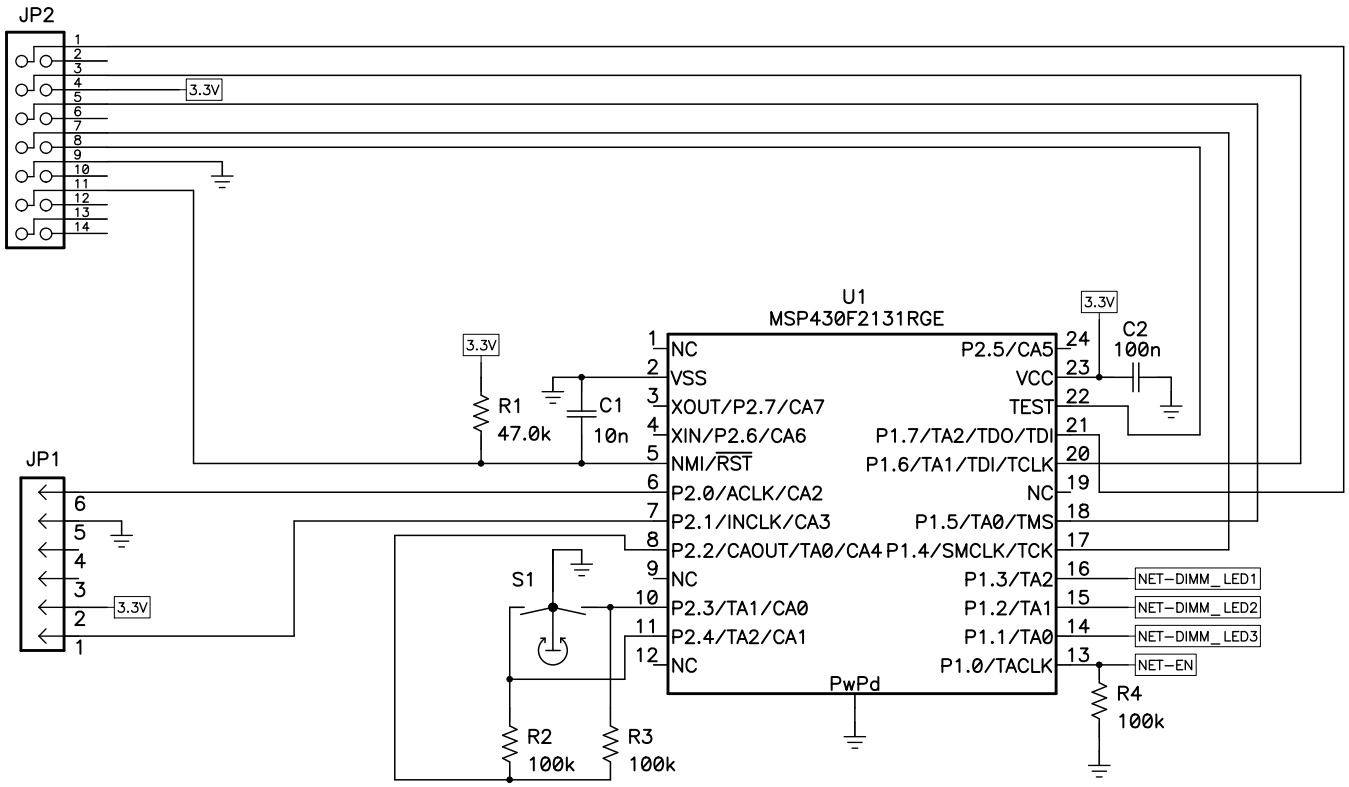
Schematic and Bill of Materials

This chapter provides the TPS62260LED-338 schematics and bill of materials.

5.1 Schematics



R9, R10, R14 2 Ohm => 300 mA
R9, R10, R14 1.69 Ohm => 350 mA



5.2 Bill of Materials

Table 5-1. Bill of Materials(1) (2) (3)

COUNT	RefDes(4)	VALUE	DESCRIPTION	SIZE	PART NO.	MANUFACTURER
1	C1	10nF	Capacitor, Ceramic, 50V, X5R, 20%	0603	std	std
1	C2	100nF	Capacitor, Ceramic, 50V, X5R, 20%	0603	std	std
4	C3, C4, C9, C11	22uF	Capacitor, Ceramic, 16V, X5R, 20%	1210	C1210C226M4PAC	KEMET
7	C5 - C8, C10, C12, C13	4.7uF	Capacitor, Ceramic, 6.3V, X5R, 20%	0603	C0603C475M9PAC	KEMET
3	D1, D2, D6	TS4148RY	Diode, Hi-Speed, 150mA, 100V, 500mW	0805	TS4148RY	Taiwan Semiconductor
1	D3	LR W5SM	Diode, LED Red, 500-mA	0.244 × 0.441 inch	LR W5SM-HYJY-1-0-400-R18-Z	Osram
1	D4	LB W5SM	Diode, LED Blue, 500-mA	0.244 × 0.441 inch	LB W5SM-EYGX-35-0-350-R18-Z	Osram
1	D5	BZX84-C3V3	Diode, Zener, 3.3V, 250mW, 5%	SOT23	BZX84-C3V3	NXP Semiconductor
1	D7	LT W5SM	Diode, LED Green, 500-mA	0.244 × 0.441 inch	LT W5SM-HYJZ-25-0-350-R18-Z	Osram
2	J1, J2	PTC36SAAN	Header, Male 2-pin, 100mil spacing, (36-pin strip)	0.100 inch × 2	PTC36SAAN	Sullins
1	J3	RAPC 712	Connector, Pin dia.2.5mm, DC Jack,	0.57 × 0.35 inch	RAPC 712	Switchcraft
1	JP1	850-106-10-S-RA	Header, 1x6-pin, 50mil spacing	1.000 × 0.085 inch	850-10-006-20-001000	Millmax
1	JP2	2514-6002UB	Connector, Male Straight 2x7 pin, 100mil spacing, 4 Wall	0.100 inch × 2X7	2514-6002UB	3M
3	L1, L2, L3	2.2uH	Inductor, SMT, 2.2uH, 1.1A, 110-milliohm orAlternate Inductor; SMT, 2.2uH, 1.0A, 120-milliohm	2.5 × 2.0 mm 2.5 × 2.0 × 1.2mm	MIPSA2520D2R2LQM2HPN2R2MJ0L	FDKmuRata
1	R1	47.0k	Resistor, Chip, 1/16W, 1%	0603	Std	Std
1	R11	330	Resistor, Chip, 1/16W, 1%	0603	Std	Std
3	R2, R3, R4	100k	Resistor, Chip, 1/16W, 1%	0603	Std	Std
6	R5 - R8, R12, R13	10.0k	Resistor, Chip, 1/16W, 1%	0603	Std	Std
3	R9, R10, R14	2.00	Resistor, Chip, 1/8W, 1%	1206	Std	Std
1	S1	3315C-001	Encoder, Sealed Incremental, 9 mm Square	0.375 × 0.400 inch	3315C-001	Bourns
9	TP1 - TP9	5001	Test Point, Black, Thru Hole Color Keyed	0.100 × 0.100 inch	5001	Keystone
1	U1	MSP430F2131TRGE	IC, Mixed Signal Microcontroller	QFN-24	MSP430F2131TRGE	TI
3	U2, U3, U4	TPS62260DRV	IC, 2.25MHz 600mA Step-Down Converter	SON-6[DRV]	TPS62260DRV	TI
1	--		PCB, 4.33 In x 2.4 In x 0.062 In		HPA338 Rev. A	Any
1	--	020-2220(5)	Knob, collet locking, black plastic	10mm overall dia. 1/8" shaft dia.	"020-2220 orequivalent"	ELMA
1	--	040-1020(6)	Cap, black plastic	10mm overall dia.	040-1020 or equivalent	ELMA

(1) These assemblies are ESD sensitive; ESD precautions shall be observed.
(2) These assemblies must be clean and free from flux and all contaminants. Use of clean flux is required.
(3) These assemblies must comply with workmanship standards IPC-A-610 Class 2.
(4) Reference designators marked with an asterisk ('**') cannot be substituted. All other components can be substituted with equivalent components from other manufacturers.
(5) Knob 020-2220 shall be installed onto S1 after final board assembly
(6) Cap 040-1020 shall be installed onto Knob 020-2220

Board Layout

This chapter provides the TPS62260LED-338 EVM board layout and illustrations.

6.1 Photographs of Top and Bottom

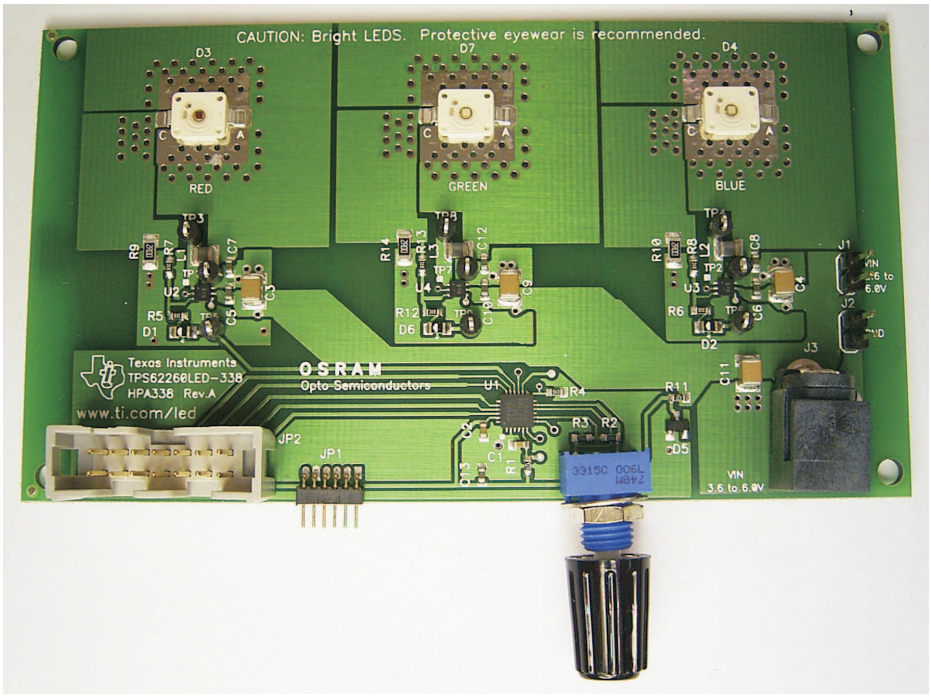


Figure 6-1. HPA338 Top View

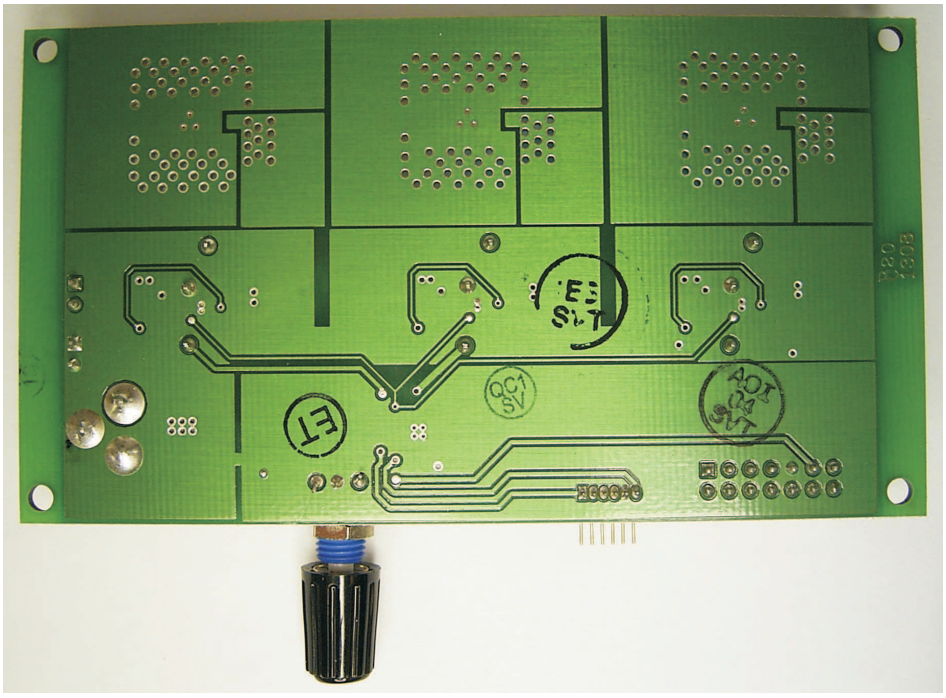


Figure 6-2. HPA338 Bottom View

6.2 Layout

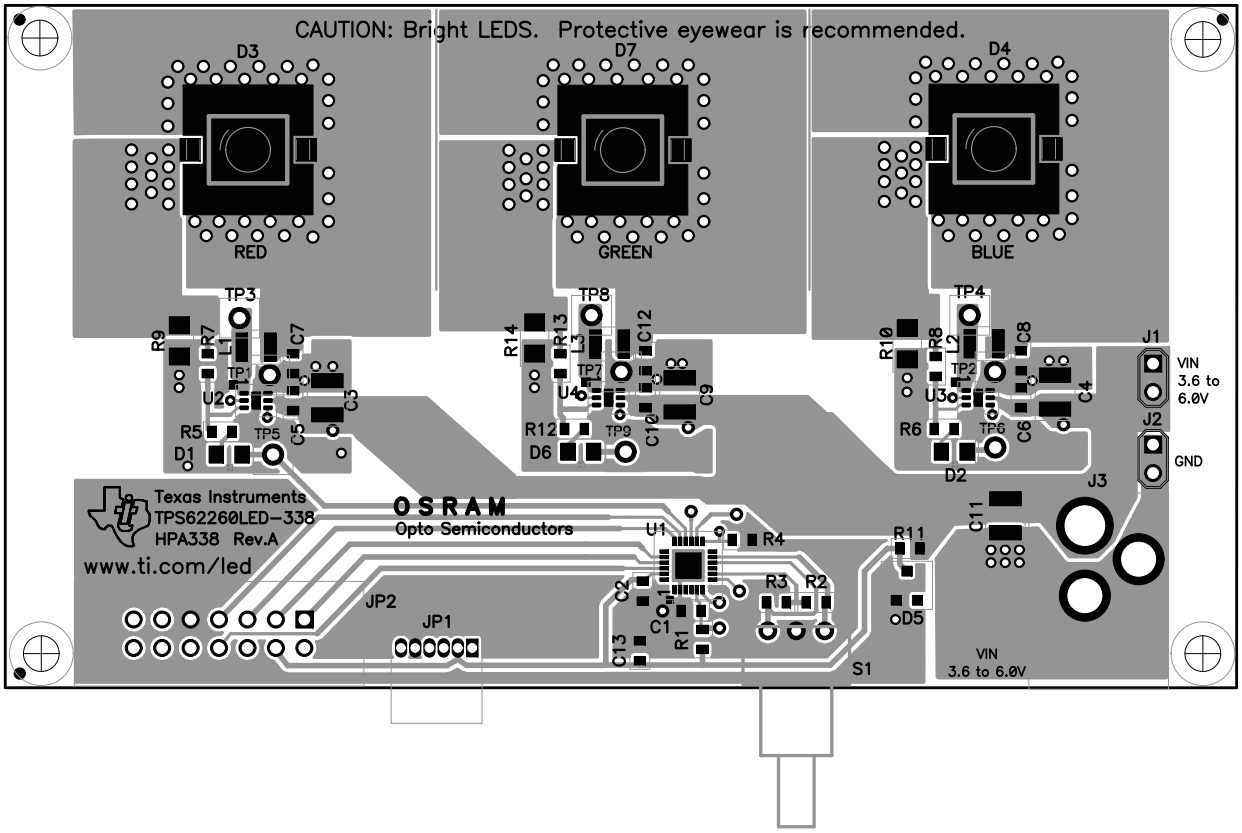


Figure 6-3. PCB Top Assembly Layer

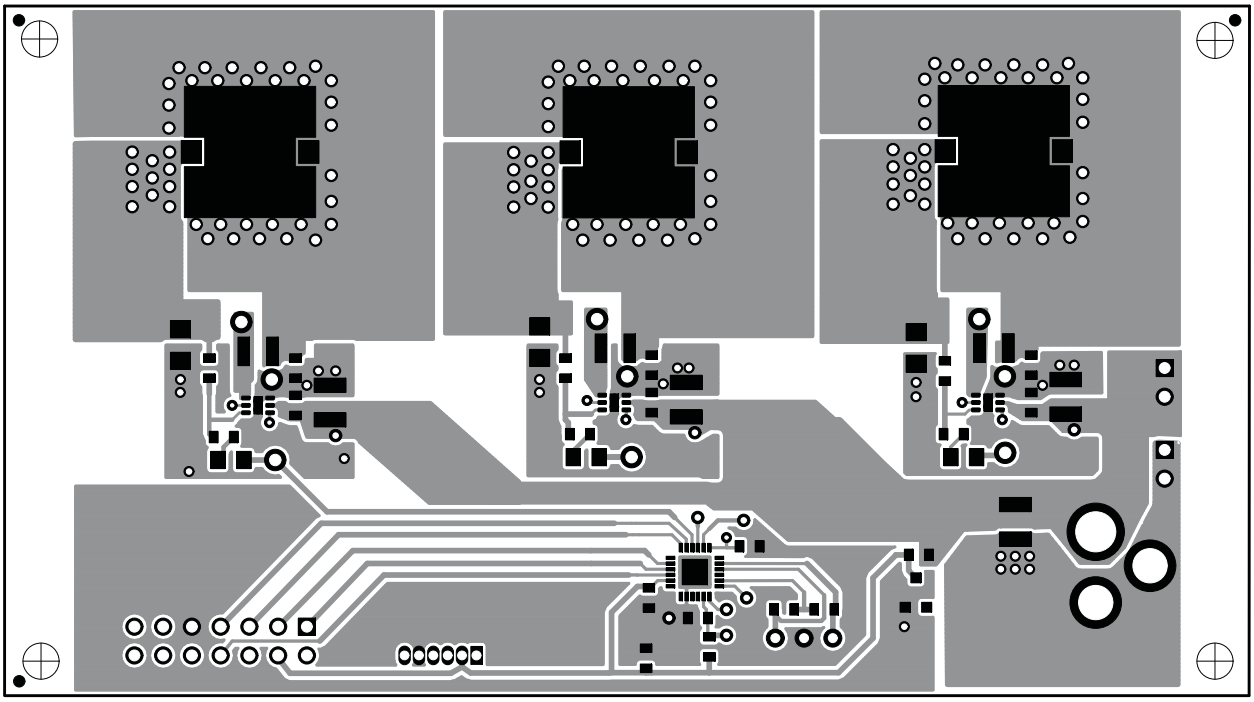


Figure 6-4. PCB Layer One

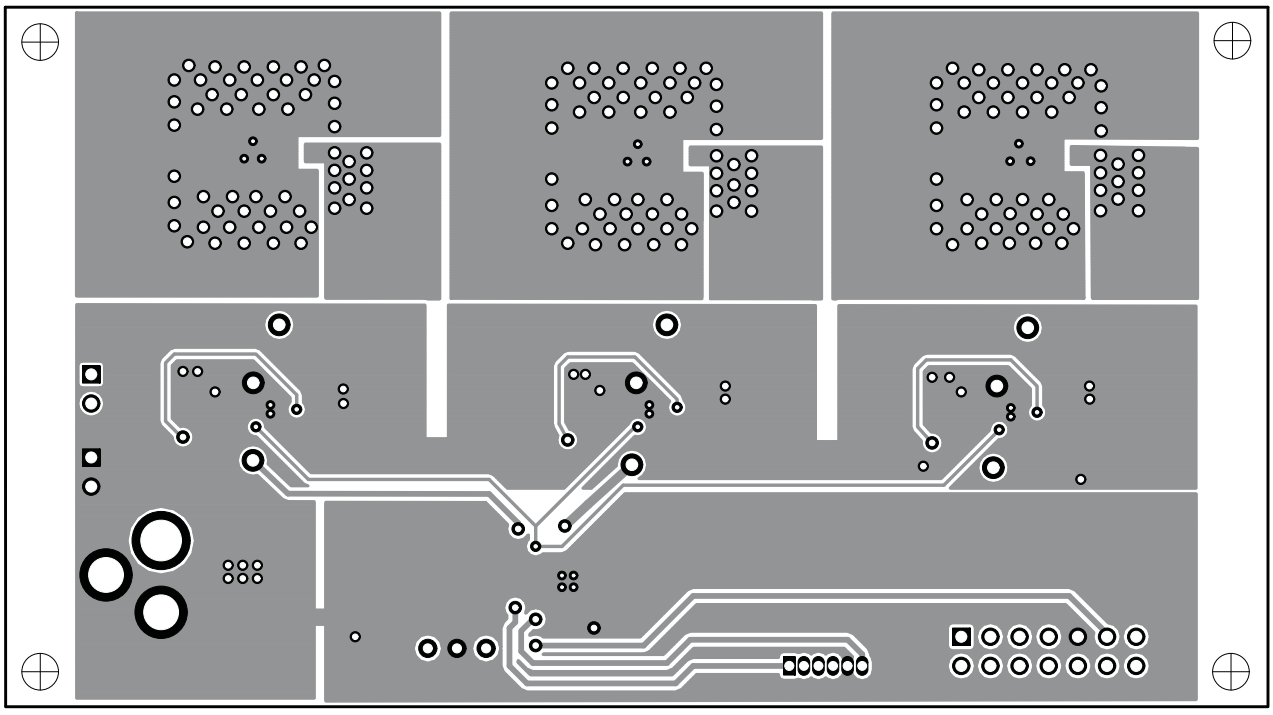


Figure 6-5. PCB Layer Two

6.3 Thermal Images

Figure 6-6 and Figure 6-7 show the thermal images of two EVM boards. The image in Figure 6-6 was obtained without using a heatsink and the image in Figure 6-7 with a heatsink mounted on the bottom of the PCB. Using a heatsink not only reduces the operating temperature of each LED, it also helps significantly to spread the heat more evenly.

These images were obtained using modified firmware that forced all three LEDs to operate with a 100% dimming duty cycle, meaning that they are fully on. Using the standard firmware, each LED is on for only one third of the time and off for two thirds of the time. As a result, the heat dissipation during normal use with the standard firmware is significantly lower than in this test case.

The heatsink used was SK 477 100 fixed with the thermally conductive foil WLFT 404 R25 both from Fischer Elektronik.

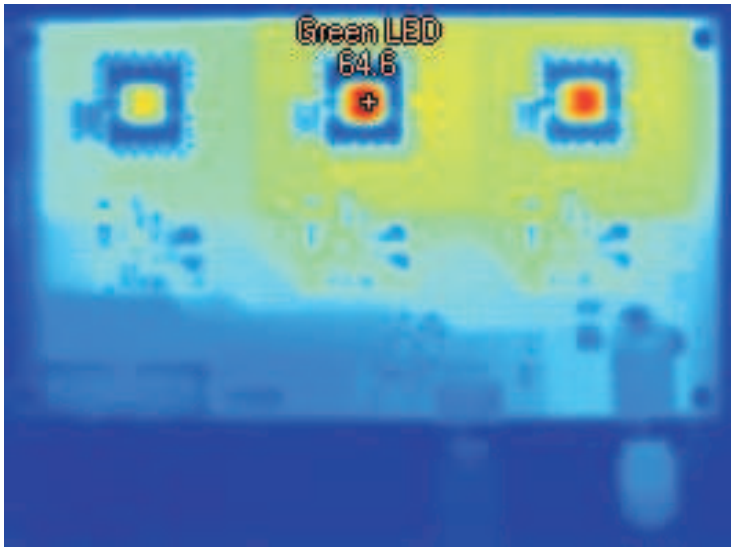


Figure 6-6. EVM Without Heatsink

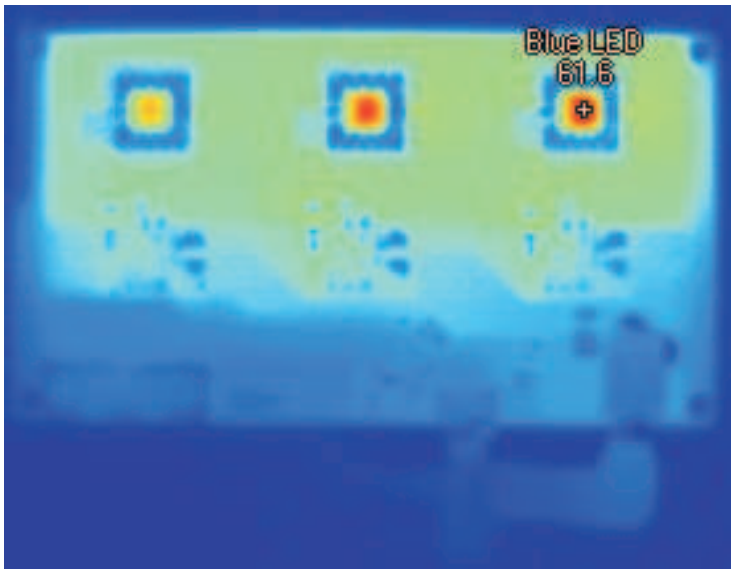


Figure 6-7. EVM With Heatsink

Reprogramming

A.1 Additional Software and Hardware Needed

1. IAR Embedded Workbench KickStart

   This is a free version that can be downloaded from www.ti.com/tool/iar-kickstart.

2. Source Code

   This source code can be found in the TPS62260LED-338EVM product folder. Self-written code can also be used. After downloading the zip file, extract it to a single folder.

3. MSP-FET MSP MCU Programmer and Debugger

A.2 IAR Embedded Workbench KickStart Software Installation

Extract the downloaded zip file and execute the install file (FET_Rxxx.exe). Read through the release notes to make sure that the software is installed properly.

Follow the instructions on the supplied release notes to install the IAR Embedded Workbench® KickStart Kit. The term KickStart refers to the function-limited version of IAR Embedded Workbench (including C-SPY® debugger). KickStart is supplied on the CD-ROM included with each FET, and the latest version is available from the TI web site. The release notes can be accessed using Start → Programs → IAR Systems → IAR Embedded Workbench KickStart for MSP430 Vx. KickStart is compatible with Windows® 98, Windows 2000, Windows ME, Windows NT 4.0, Windows XP, and Windows Vista. However, the USB FET interface works only with Windows 2000, Windows XP, and Windows Vista.

A.3 Hardware Installation

For MSP-FET:

1. Use the USB cable to connect the USB FET interface module to a USB port of your PC. The USB FET should be recognized instantly, as the USB device driver should have been installed with the KickStart software.

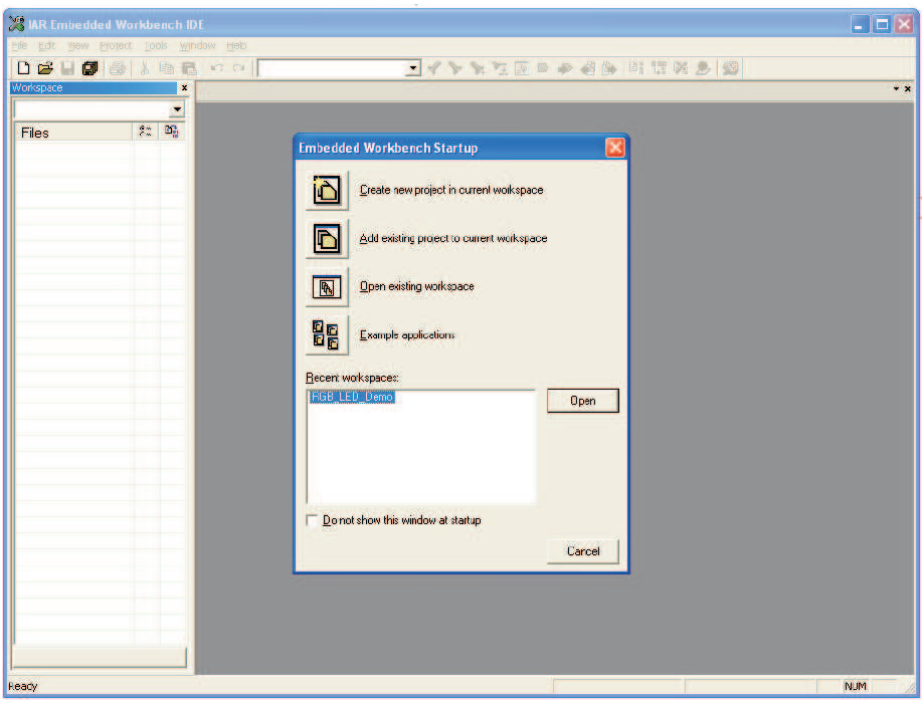
   If for any reason the Install Wizard starts, respond to the prompts and, when prompted, browse to the driver files that are located in <Installation Root>\Embedded Workbench x.x\430\bin\WinXP. Detailed driver installation instructions can be found in the MSP Debuggers User's Guide.

2. After connecting to a PC, the USB FET performs a self test during which the red LED flashes for about two seconds. After the self test passed successfully, the green LED lights permanently.

3. Use the 14-pin cable to connect the USB FET interface module to the HPA338 board.

A.4 Using IAR Embedded Workbench to Download Code on MSP430 MCUs

1. Start the Workbench (Start → Programs → IAR Systems → IAR Embedded Workbench KickStart for MSP430 Vx → IAR Embedded Workbench).



2. Click Open existing workspace to open an existing file (for example, HPA338RevA.eww, which is one of the files that are part of the software files available in the TPS62260LED-338EVM product folder The workspace window opens. You can also generate a new project and program the MSP430 by your own to light the LEDs as desired, but this is beyond the topic of this user's guide. For more details on programming the MSP430, refer to www.ti.com/msp430.
3. Click Project → Options → FET Debugger → Setup → Texas Instruments USC-IF for the USC Interface (MSP430-FET).
4. Click Project → Rebuild All to build and link the source code.
5. Click Project → Debug to start the C-SPY debugger. C-SPY erases the device Flash and then downloads the application object file to the device Flash. The LEDs on the board turn off.
6. Click Debug → Stop Debugging. Reset the MSP430 MCU by disconnecting and reconnecting the power plug. This restarts the program on the MCU.

Revision History

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

Changes from October 15, 2008 to August 7, 2018 **Page**

- Editorial and formatting changes and updated links throughout document 4
- Changed the MSP430 programmer from MSP-FET430UIF to MSP-FET 6
- Removed paragraph that began "After installation, a PC reboot is required..." in Section A.2, *IAR Embedded Workbench KickStart Software Installation* 26
- Removed all references to MSP-FET430PIF (obsolete) 26

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

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