Application Report

Common Application Use Cases for the TPS272C45 Smart Industrial High Side Switch

Timothy Logan

ABSTRACT

The TPS272C45 is an industrial high-side switch used in a variety of applications including factory automation, robotic controls, and other industrial applications. While the device can be used as a standalone high-side switch with no underlying microcontroller, to get the most features out of the device a microcontroller is recommended to control, monitor, and configure the inputs and outputs of the TPS272C45. This application note goes over several examples on how to optimize the usage of the TPS272C45 high-side switch when writing software with a microcontroller. Several reference code examples are provided for the MSP-EXP430F5529LP LaunchPad as supplements to illustrate some of the more common use cases for using the TPS272C45.

Table of Contents

1 Basic Interaction....2
2 Current Sense....5
3 Multiplexed Current Limit....9

List of Figures

Figure 1-1. Output Toggling....3
Figure 2-1. Compiler Option for printf....5
Figure 2-2. Console Print with ADC Results....6
Figure 2-3. Current Sense with DIAG_EN Toggling....7
Figure 2-4. Comparator Flow Chart....8
Figure 3-1. Multiplexed Current Limit....9
Figure 3-2. Current Limit - Permanent Short 2A....10

List of Tables

Table 1-1. Connected BoosterPack Header Pins on TPS272C45EVM....2

Trademarks

1 Basic Interaction

The TPS272C45EVM is equipped with BoosterPack headers to allow for easy interaction with a Texas Instruments microcontroller. While the BoosterPack headers enable easy interaction with a Texas Instruments branded microcontroller, any microcontroller with basic GPIO functionality and an ADC interface can be used. The relevant signals that can be controlled from the microcontroller are listed in Table 1-1.

Table 1-1. Connected BoosterPack Header Pins on TPS272C45EVM

BoosterPack Pin	Function	Note
J1-1	3.3 V Power Rail	Disconnect ***J19*** if powering LaunchPad through USB.
J1-2	Current sensing through the SNS pin	Populate a Zener diode on ***D7*** if it is required to limit analog signal of pin
J1-5	DIA_EN pin used to enable diagnostics	Can be used to multiplex multiple TPS272C45 switches to one analog pin
J1-8	SEL pin used to change measurement reported on SNS pin	
J2-11	LATCH pin used to change fault behavior	
J2-12	ILIM1 pin used to enable 10kΩ resistor ***R13*** for ILIM1	Active high. Do not enable at same time as J2-13
J2-13	ILIM2 pin used to enable 4.99kΩ resistor ***R14*** for ILIM1	Active high. Do not enable as same time as J2-12
J2-17	FLT pin used to detect faults	Open drain input. Pull-up source can be controlled using jumper ***J18***
J2-18	EN2 to enable VOUT2	Active high. Can be connected to PWM
J2-19	EN1 to enable VOUT1	Active high. Can be connected to PWM
J2-20	Module GND	Do not connect to IC ground if ground network is used

Note that before the TPS272C45 can be used with the MSP-EXP430F5529LP, the jumpers on the EVM must be configured to work in microcontroller mode. By default the TPS272C45EVM is configured to be used as a standalone EVM so the following jumper configurations must be made to route the relevant signals to the connected LaunchPad:

- Disconnect jumpers J8, J9, J10, J11, J12. These will allow the microcontroller to control the digital signal pins without any interaction from the LDO or ground
- Disconnect jumper J13. This will disable the onboard 3.3 V LDO
- Disconnect J14 and J15. This will allow the MSP430 to control the multiplexed current limit
- Configure J17 to the *BP* configuration. This will power the VDD rail through the LaunchPad's 3.3 V signal
- Configure J18 to the *BP* configuration. This will pull the FAULT pin up through the LaunchPad's 3.3 V signal

A simple code example that shows the basic setup of the TPS272C45 can be found in the ***tps272c45_basic_setup*** folder of the source package included with this application note. In this code example the DIAG_EN, SEL, LATCH, and EN1/EN2 lines are setup as outputs, the FLT line is set as an interruptible input. The SNS pin is not used in this application. The S1 and S2 push buttons on the MSP-EXP430F5529LP are used in this application to toggle EN1 and EN2 on and off respectively. This is simply done by toggling the output state of the GPIO in each button's interrupt handler. The interrupt handlers for the S1/EN1 interrupt handler are shown here:

```
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{
    switch (__even_in_range(P2IV, 16))
    {
        case P2IV_NONE:
            break;   // Vector  0: no interrupt
        case P2IV_P2IFG0:
            break;   // Vector  2: P2.0
        case P2IV_P2IFG1:
            P1OUT ^= BIT0;    // Vector  4: P2.1 (EN1)
            en1On = !en1On;
            EN1_OUT ^= EN1_PIN;
            break;
        case P2IV_P2IFG2:
            break;   // Vector  6: P2.2
        case P2IV_P2IFG3:
            break;   // Vector  8: P2.3
        case P2IV_P2IFG4:
            break;   // Vector 10: P2.4
        case P2IV_P2IFG5:
            break;   // Vector 12: P2.5
        case P2IV_P2IFG6:
            break;   // Vector 14: P2.6
        case P2IV_P2IFG7:
            break;   // Vector 16: P2.7
        default:
            break;
    }
}
```

The oscilloscope screenshot of the TPS272C45 with an 18 V supply hooked up to an 18Ω load when EN1 is pressed can be shown in Figure 1-1:

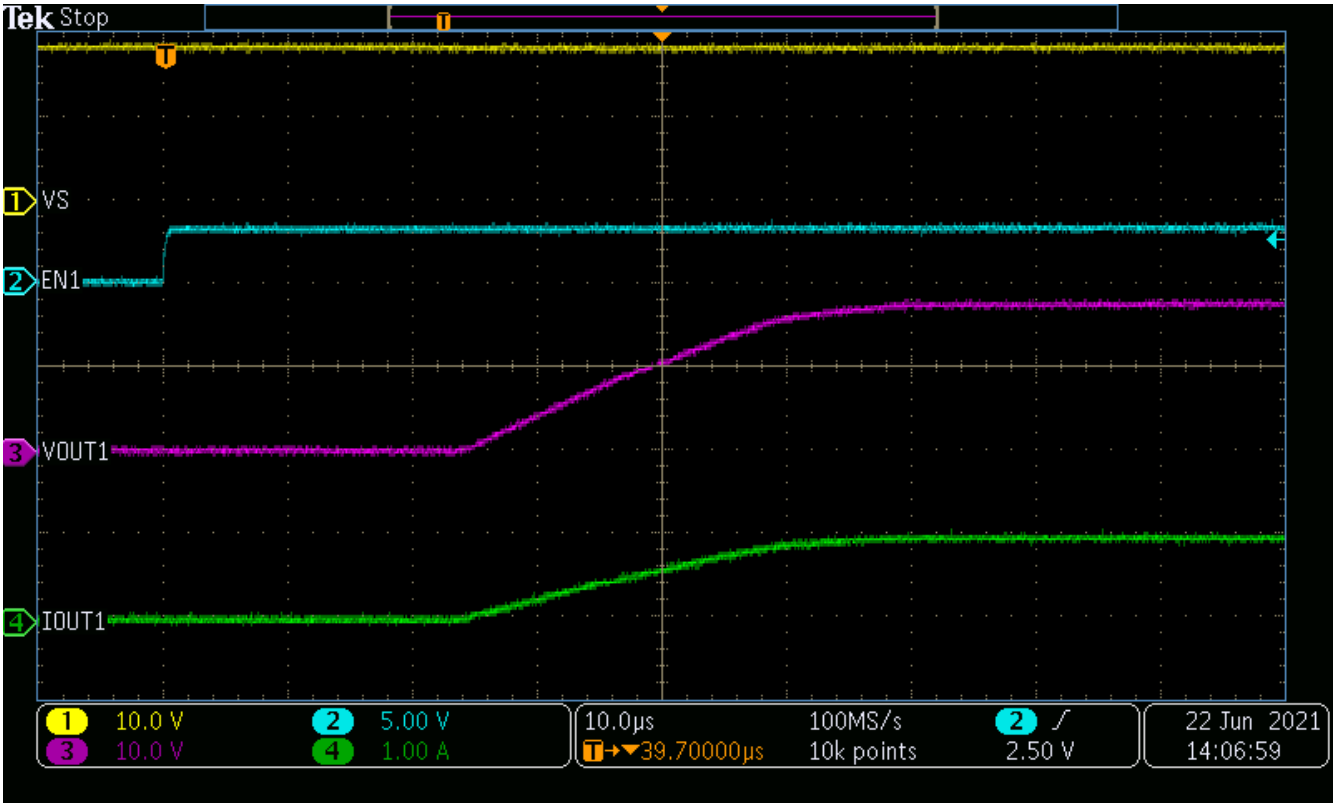

Figure 1-1. Output Toggling

When the EN1 channel is ON the green LED on the LaunchPad will be turned on and when the EN2 channel is ON the red LED of the LaunchPad will be turned on. Also note that when a fault occurs in the system (as detected with the FLT line) the LaunchPad's LEDs will blink quickly to illustrate the fault. The fault is detected in the software by setting up an edge triggered GPIO interrupt on the FLT line. When the fault is resolved the microcontroller will detect the rising edge on the fault line and resume normal LED behavior. Note that the FLT line must be pulled up to the logic level of the microcontroller by either an internal pull up in the microcontroller or an external pull-up. The interrupt handler for the fault line in the case of an over current event are shown here:

```
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    switch (__even_in_range(P1IV, 16))
    {
        case P1IV_NONE:
            break;   // Vector  0: no interrupt
        case P1IV_P1IFG0:
            break;   // Vector  2: P1.0
        case P1IV_P1IFG1:
            P4OUT ^= BIT7;    // Vector  4: P1.1 (EN2)
            en2On = !en2On;
            EN2_OUT ^= EN2_PIN;
            break;
        case P1IV_P1IFG2:
            break;   // Vector  6: P1.2
        case P1IV_P1IFG3:
            break;   // Vector  8: P1.3
        case P1IV_P1IFG4:
            FAULT_IE &= ~FAULT_PIN;   // Vector 10: P1.4 (Fault)
            TA0CTL = TASSEL_2 + MC_1 + TACLR;
            break;
        case P1IV_P1IFG5:
            break;   // Vector 12: P1.5
        case P1IV_P1IFG6:
            break;   // Vector 14: P1.6
        case P1IV_P1IFG7:
            break;   // Vector 16: P1.7
        default:
            break;
    }
}
```

2 Current Sense

The TPS272C45 high-side switch features a high accuracy current sense capability that can be used to detect granular variations on load current. The current sense block of the TPS272C45 outputs a reference current to the SNS pin that is a scaled down representation of the current on the load. This current is translated to a voltage through an attached SNS resistor and read by the ADC of the microcontroller.

In the ***tps272c45_adc_polling*** code example provided with this application the current of channel 1 (VOUT1) is polled at an interval of 500mS and the result is displayed through the debug terminal of the MSP-EXP430F5529LP. The raw ADC value is passed into the debugger's printf statement and displayed on the debug terminal through Code Composer Studio. Note that for the print statements to work correctly, printf support must be enabled in the compiler options as shown in Figure 2-1.

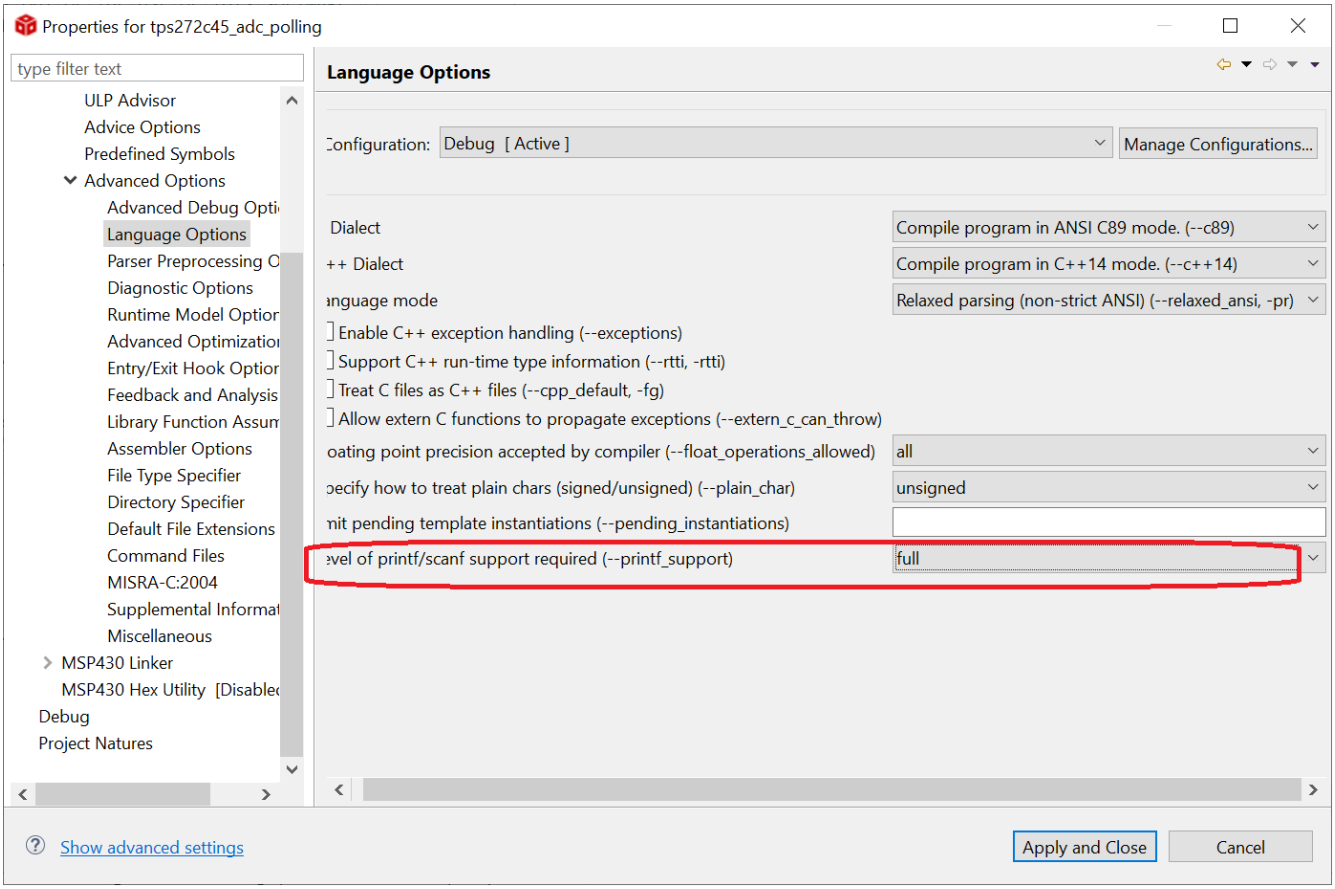

Figure 2-1. Compiler Option for printf

Once connected, press the S1 button on the top of the MSP-EXP430F5529LP to assert the EN1 signal high and enable the output on VOUT1. When the channel is enabled, a timer will start on the microcontroller and periodically sample/convert the voltage on the SNS pin, record the result, and then display the result through the debugger screen. The output of this code example is shown in Figure 2-2.

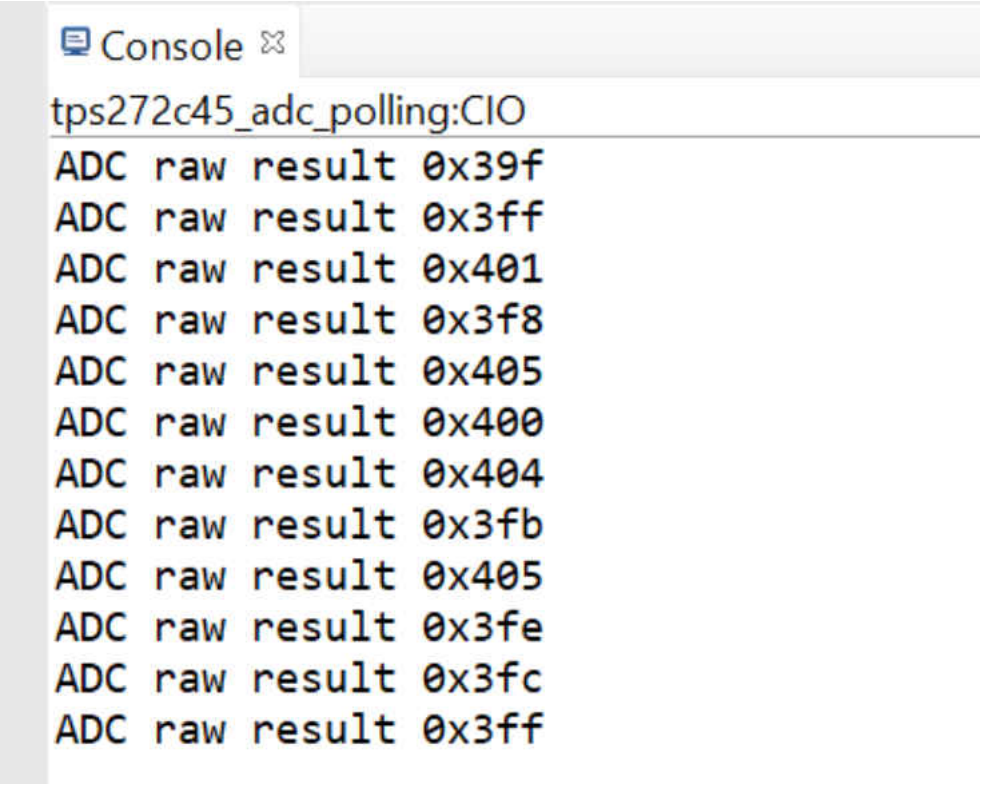

Figure 2-2. Console Print with ADC Results

One important aspect to note about a polling based approach for measuring the current is the behavior of the DIAG_EN pin on the TPS272C45. The DIA_EN pin will enable or disable the current sensing block on the TPS272C45. As a result the power consumption of the TPS272C45 will be significantly higher when DIAG_EN is high versus low. For this reason it is recommended only to assert the DIAG_EN signal high right before the ADC sample/conversion and assert it low immediately after. Note that from the time it takes from the initial rising edge of DIAG_EN to when the SNS signal has settled is specified in the data sheet as $t_{SNSION1}$. This value must be taken into account in the software through a delay to sample an accurate reading on the ADC. The relevant software code snippet is shown in Figure 2-3 with an oscilloscope screen shot of the DIAG_EN and SNS pin.

```
        /* Enabling the diagnostic block and delaying */
        DIA_EN_OUT |= DIA_EN_PIN;
        __delay_cycles(1000);

        /* Starting the conversion */
        ADC12CTL0 |= ADC12SC;

        /* Waiting for the interrupt */
        __bis_SR_register(LPM0_bits + GIE);

        /* Disabling diagnostic block */
        DIA_EN_OUT &= ~DIA_EN_PIN;

        /* Set breakpoint here to record result */
        __no_operation();
        printf("\nADC raw result 0x%x", adcConvRes);

        /* Delaying between measurements */
        __delay_cycles(1000000);
```

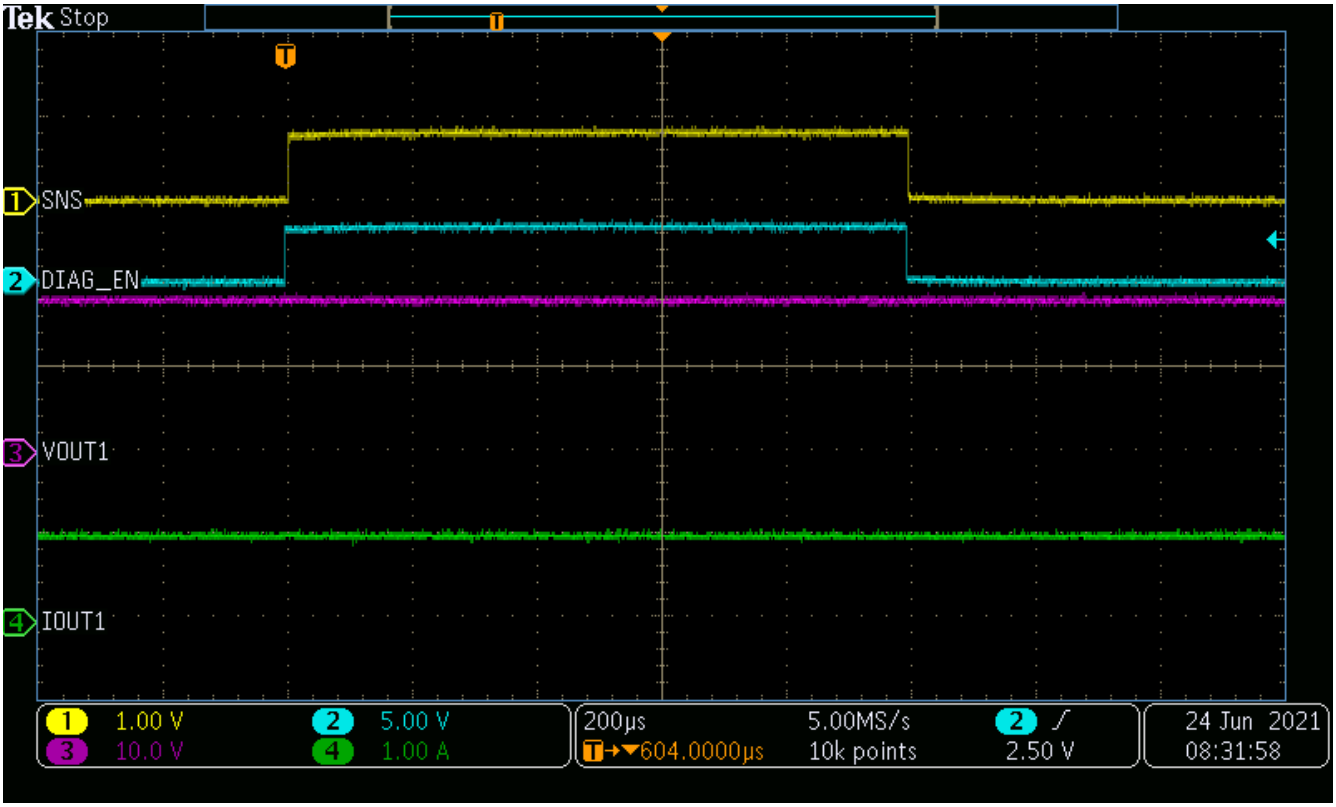

Figure 2-3. Current Sense with DIAG_EN Toggling

The ***tps272c45_comparator*** code example connects the output of the SNS pin to one of the MSP430's integrated fast comparator peripherals. This can be used as a signal monitor to continuously monitor the output current of the TPS272C45 and quickly trip when a threshold is crossed. This can either be used to quickly detect an undercurrent event by setting the comparator to trip on a falling edge or to provide a *current limit pre-warning* interrupt to the microcontroller when the comparator is configured to interrupt on a rising edge.

In this code example, the output of the SNS pin is connected to one of the MSP430's internal quick comparators and configured to fire an interrupt when the output load current exceeds 1-A on VOUT1. Once this interrupt fires the LED on the MSP-EXP430F5529LP will turn on to signify the event. As with previous code examples, the S1 button on the side of the LaunchPad is used to turn on and off VOUT1. For simplicity, all fault reporting and error handling has been removed from this code example. This behavior is shown in Figure 2-4.

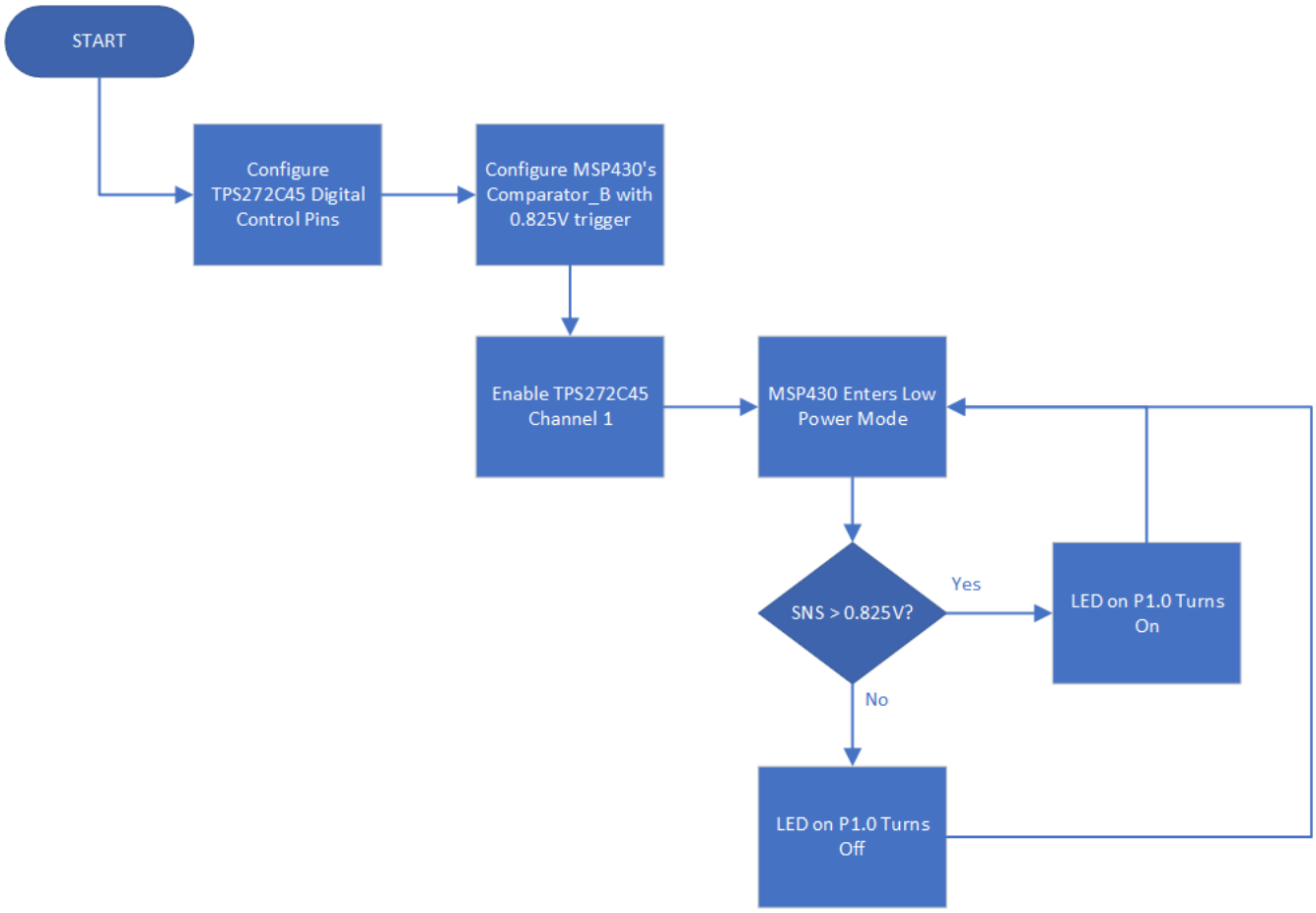

Figure 2-4. Comparator Flow Chart

Note that unlike the ADC polling example, for the comparator example to work, the DIAG_EN line must be either continuously enabled or periodically enabled long enough for the current sense to register a correct reading. This code example assumes that the power budget is not a big factor in the application and leaves the DIAG_EN line enabled through the entire program; however software optimizations such as DMA utilization could be adopted to automatically toggle the DIAG_EN line from a hardware timer.

3 Multiplexed Current Limit

The TPS272C45EVM provides an interface for the microcontroller to pragmatically switch between two or more current limits on an individual I_{LIM} pin. This is accomplished by the use of a MOSFET controlled by the microcontroller's GPIO pin to switch the resistance value between the I_{LIM} pin and GND. The hardware schematic of this setup is shown in Figure 3-1.

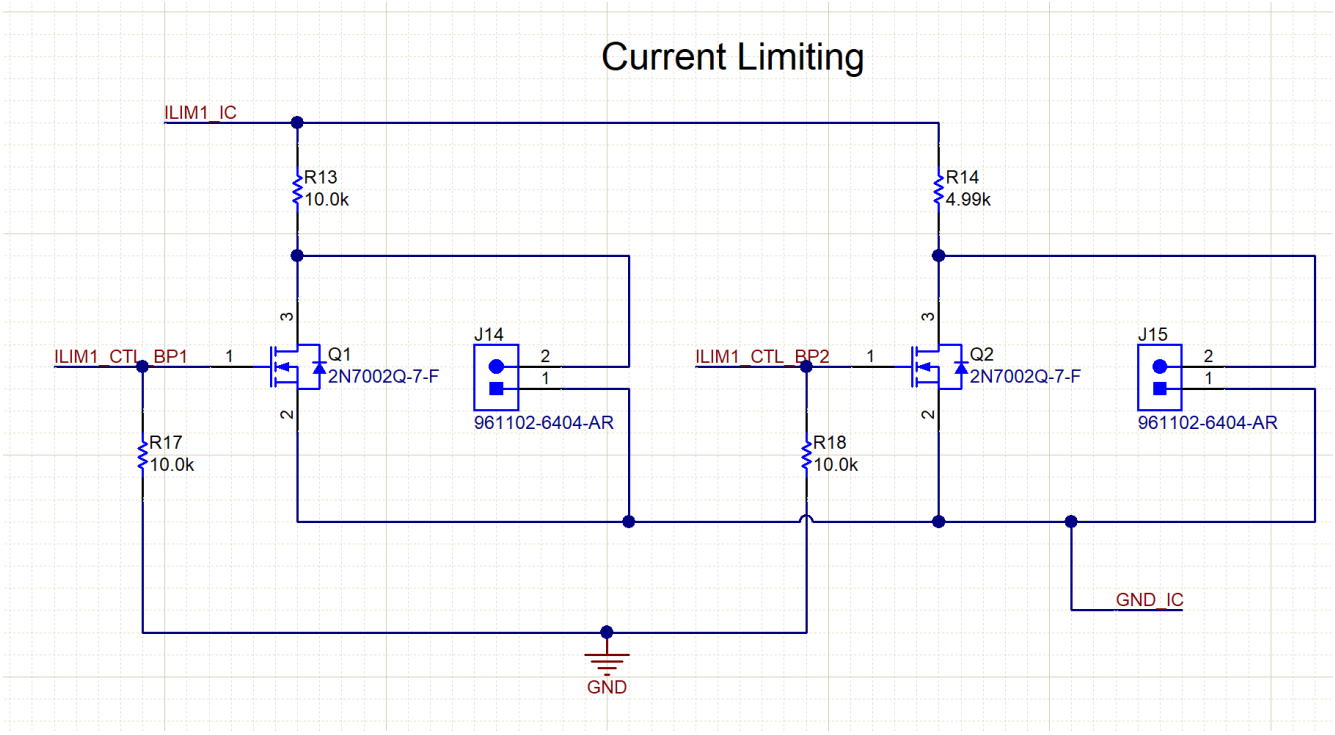

Figure 3-1. Multiplexed Current Limit

In the ***tps272c45_multiplexed_current_limit*** code example the two MOSFETs connected to the I_{LIM1} pin on the TPS272C45EVM are used to demonstrate how to switch between two separate current limits using software. By default, the MSP430 configures the TPS272C45EVM to use the R13 resistor as the current limit. This is done by using the Q1 MOSFET on the TPS272C45EVM to enable the path from R13 to GND_IC. With the Q1 MOSFET enabled the MSP430 microcontroller drives low the gate of Q2's MOSFET and effectively shuts off the connection to R14. In this application, S1 turns on and off VOUT1 while S2 switches between R13 and R14 as the configured current limit. VOUT2 is not used in this example.

It is important when using the multiplexed current limit configuration to enable only one resistor on the ILIM pin at a time. The ILIM pin is sensitive to any capacitance so enabling both resistor may lead to unreliable behavior due to the combined trace lengths. The software snippet used to enable R13 and disable R14 are shown here:

```
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    switch (__even_in_range(P1IV, 16))
    {
        case P1IV_NONE:
            break;   // Vector  0: no interrupt
        case P1IV_P1IFG0:
            break;   // Vector  2: P1.0
        case P1IV_P1IFG1:
            ILIM1_OUT ^= ILIM1_PIN;
            ILIM2_OUT ^= ILIM2_PIN;
            P1OUT ^= BIT0;
            P4OUT ^= BIT7;
            break;
        case P1IV_P1IFG2:
            break;   // Vector  6: P1.2
        case P1IV_P1IFG3:
            break;   // Vector  8: P1.3
        case P1IV_P1IFG4:
            break;
        case P1IV_P1IFG5:
            break;   // Vector 12: P1.5
        case P1IV_P1IFG6:
            break;   // Vector 14: P1.6
        case P1IV_P1IFG7:
            break;   // Vector 16: P1.7
        default:
            break;
    }
}
```

The current limit engaging on the R13 resistance of 10kΩ (2A nominal current limit) can be seen below in Figure 3-2

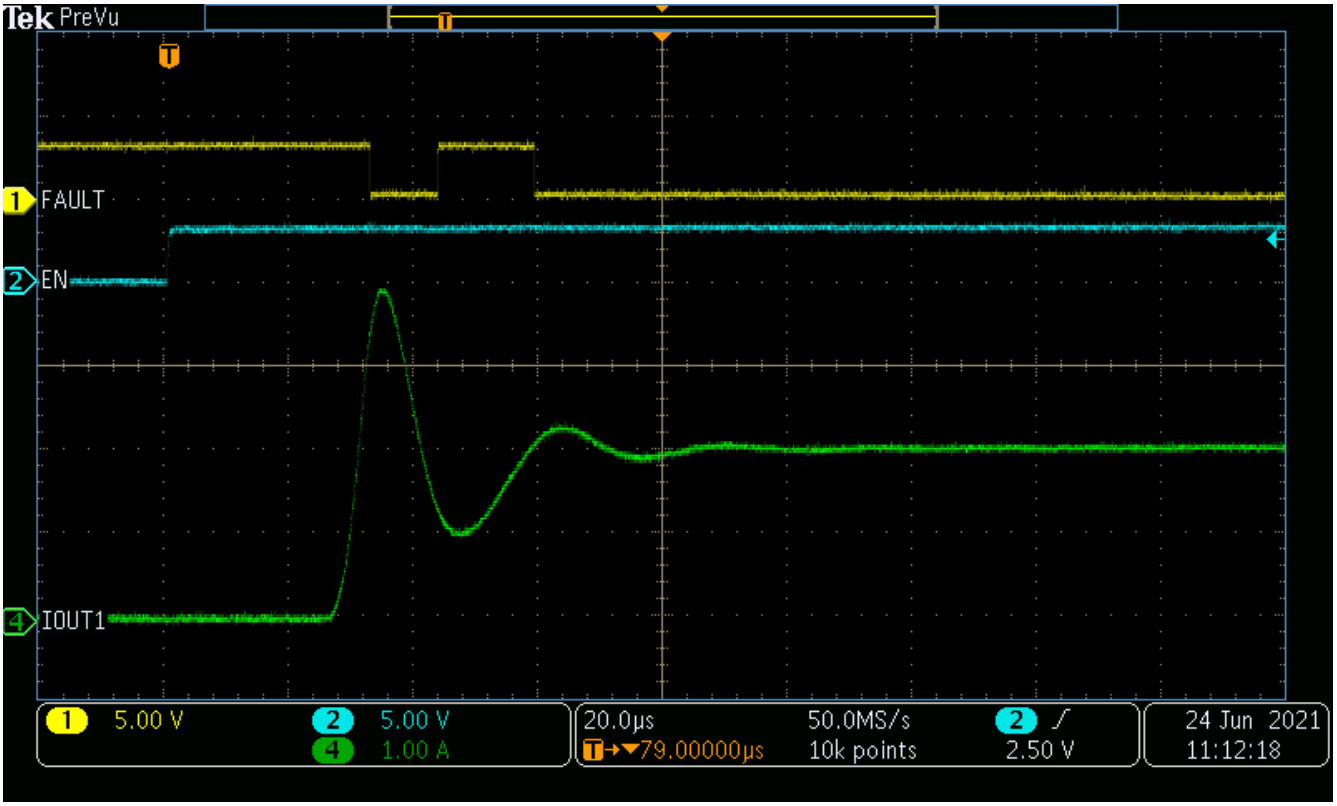

Figure 3-2. Current Limit - Permanent Short 2A

Note that in the waveform below that when the current limit engages the FAULT pin initially goes low, briefly goes high, and then remains low for the duration of the overcurrent event. The initial low state of the FAULT and brief high state is due to the current limit engaging and temporarily dropping the current to below the set current limit value. To prevent any race condition in software for handling the FAULT pin, the pin interrupt should trigger a timer that acts as a deglitch filter. This is implemented in the ***tps272c45_adc_polling*** code example and the code block of the timer ISR that implements the deglitch filter is shown here:

```
// Timer0 A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR(void)
{
    /* If the fault is gone, return to normal operation */
    if(FAULT_IN | FAULT_PIN)
    {
        TA0CTL = 0;

        if(en1On == true)
        {
            P1OUT |= BIT0;
        }
        else
        {
            P1OUT &= ~BIT0;
        }

        if (en2On == true)
        {
            P4OUT |= BIT7;
        }
        else
        {
            P4OUT &= ~BIT7;
        }
    }
}
```

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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, regulatory 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 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.

TI objects to and rejects any additional or different terms you may have proposed.

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