![Texas Instruments]

MSP430 USB Connectivity Using TUSB3410

Andreas Dannenberg *MSP430*

ABSTRACT

This application report presents a ready-to-use USB connectivity reference design for MSP430 microcontrollers using the Texas Instruments TUSB3410 USB-to-serial bridge controller. The provided solution enables high-speed data transfers with speeds of up to 921,600 bit/s as well as MSP430 Flash code download through the USB port. The reference design includes MSP430 and PC software, drivers, schematics, layout, and BOM information.

Contents

TEXAS
INSTRUMENTS

Figures

Tables

1   Introduction

To enable USB connectivity for MSP430 devices, the Texas Instruments TUSB3410 USB-to-serial bridge controller can be used. The TUSB3410 is USB 2.0 full speed compliant and supports baud rates from 50 Baud up to 921.6 kBaud. It has a built-in 8052 CPU that can execute custom firmware. Both self- and USB-powered applications are supported. UART, handshake, and GPIO pins provide a variety of interface options.
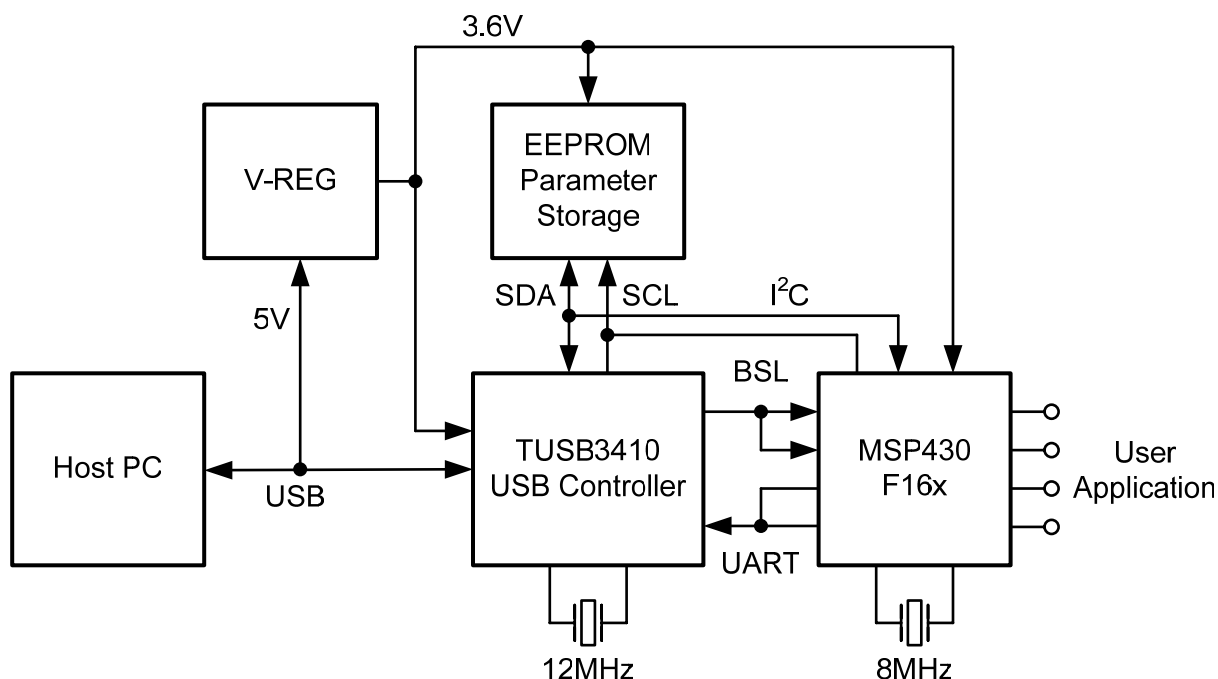
Figure 1.   MSP430F16x-TUSB3410 Reference Design Overview

Figure 1 shows the block diagram of one of the USB connectivity solutions presented in this application report using an MSP430F1612. This is the most universal approach, and this solution is applicable to all available MSP430 devices. However, depending on the selected MSP430 device as well as the specific application requirements, lower-cost solutions using fewer components can be implemented. Figure 2 shows an example of such a lower-cost solution, using an MSP430F2274. The main difference is that the lower-cost solution uses only a single crystal and uses the MSP430 to emulate the external USB configuration EEPROM. Note that both solutions are proposals only, and that it is not in the scope of this document to cover every aspect of every possible implementation.
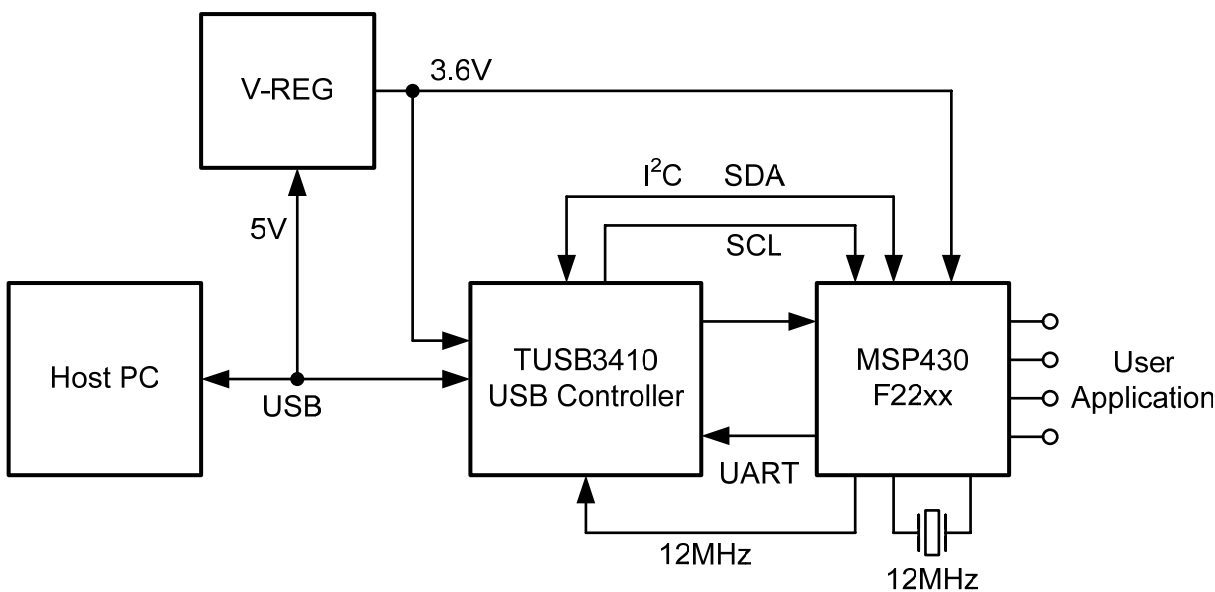
Figure 2.   MSP430F22xx-TUSB3410 Reference Design – Lower Cost Solution

The connection between host PC and MSP430 is established through a full-duplex UART link. On the PC side, a virtual COM port (VCP) provided by the TUSB3410 driver suite is used to exchange data with the MSP430 hardware USART module. This process is straightforward by using standard Windows API calls on the PC side and MSP430 USART accesses. All underlying USB transfers are transparent for both PC and MSP430 applications.

The MSP430-TUSB3410 reference designs are bus-powered USB devices. Also, the solution shown in Figure 1 is capable of programming a blank MSP430 through the USB link via the bootstrap loader (BSL). Furthermore, the TUSB3410 configuration EEPROM can be programmed in-system through the MSP430 via I^2C.

Both PC and MSP430 demonstration software is provided, along with this application report to offer a complete end-to-end turnkey solution. Bidirectional data transfer is demonstrated by having the PC and the MSP430 displaying each others' key and push-button status.

Note: The software and drivers provided with this application report are for use with PCs running Windows XP™ or Windows 2000™. Older versions of Windows are not supported.

2   Design Decisions and USB Configuration

The TUSB3410 supports a variety of different application setups such as:

    Use as UART-type device

    Use as standard Windows class-type device

    Use as custom device

    Firmware storage on host PC

    Firmware storage on external EEPROM

    Device serialization

A detailed overview of the different TUSB3410 operating scenarios can be found in [8]. For the MSP430-TUSB3410 reference design, the following configuration setup is used:

Table 1.   Reference Design USB Setup

Item	Value
TUSB3410 usage	UART-type device
TUSB3410 firmware location	Host PC
Device serialization	Not used
Vendor ID	0x0451
Product ID	0xbeef
Manufacturer descriptor string	"Texas Instruments"
Product descriptor string	"MSP430-TUSB3410 Reference Design"

More information on using the TUSB3410 as a UART-type device can be found in [11].

For USB compliance, any USB product must have a unique vendor ID and product ID. The OS uses the VID/PID combination to determine what driver to load. The VID and PID are reported to the USB host in the USB device descriptor when this descriptor is requested by the host. A unique VID can be obtained from the USB Implementers Forum (www.usb.org). A product ID can be whatever a vendor chooses, but since the VID/PID pair determines what driver is loaded, the same PID should not be given to two different products. Also, these values must match to host values stored in the driver INF files. See chapter 5 for more information.

Using the TUSB3410, an external EEPROM must be used for parameter storage. An MSP430 MCU may also be used to emulate this external EEPROM. However, due to limitations of the TUSB3410 I^{2}C implementation, not all MSP430 devices can be used. The inability of the TUSB3410 to support I^{2}C clock stretching requires that the I^{2}C slave reacts immediately on the I^{2}C fast mode transactions. This requires either an MSP430 device that can operate at speeds of at least 12 MHz in combination with a USCI module (e.g., MSP430F22xx), or an MSP430 device with USCI module and DMA (e.g., MSP430FG461x). The alternative, lower-cost solution provided in this application report demonstrates the EEPROM emulation using an MSP430F22xx device. In addition to that, an EEPROM emulation code example using an MSP430FG461x device is contained in the ZIP file associated with this application report.

To generate a vendor-specific EEPROM image for use with the TUSB3410, TI provides a header generator utility for download [12]. This tool expects a configuration file (*.cfg) as input, which defines the EEPROM image contents and contains vendor-specific details, such as VID/PID reported by the USB device to the host. Example configuration files are provided along with the tool and can be modified using a text editor. The output of the header generator utility is a binary EEPROM image file (*.bin). The configuration file and the associated binary image used with the MSP430-TUSB3410 reference design are provided with this application report (see Table 4). Here, the binary output file was converted into a C-compiler constant and directly included into the MSP430 firmware. This enables EEPROM programming at MSP430 application runtime, eliminating the need for an external EEPROM programmer. See section 4.2 for more information. Note that TI also provides a tool that allows programming of a blank EEPROM directly over the USB through the TUSB3410 [9]. This EEPROM image can also be used for purposes of emulating an actual EEPROM using the MSP430, which is described in section 4.3.

3   Reference Design Hardware

3.1   Standard Solution Using MSP430F1612

The schematics of the MSP430F1612-TUSB3410 reference design can be found in section A.1. Also, PCB layout and BOM information can be found in sections A.2 and A.3. The two main components that can be identified are the TUSB3410 USB-to-serial bridge controller (U1), and the MSP430F1612 (U2). This particular higher-end MSP430 device was selected to allow more flexibility during evaluation and code development. Any MSP430F16x device can directly drop into this design, whereas other MSP430 family members would require changes to the design. The MSP430 choice is somewhat arbitrary, as almost any MSP430 can be used to interface with the TUSB3410. However, the use of a hardware USART module is recommended.

The TUSB3410 USB data lines are connected to a standard USB B-type PCB mount connector. TI's transient voltage suppressor SN75240 (U3) is used on the USB lines to provide an increased level of ESD protection.

The reference design hardware is designed as a bus-powered USB application. The USB supplies 5 V on each port, and devices can generally draw up to 100 mA from the bus without any special considerations. However, up to 500 mA can be made available by the host upon request. Here, a TI TPS77301 LDO is used (U4) to generate 3.6 V with a maximum output current of 250 mA. This supply voltage is used to power the entire circuit. LED5 is illuminated when power is supplied by the USB.

For this reference design, the MSP430 is operated at 8 MHz. This clock is provided by Q1, which is connected to the LFXT1 oscillator. All MSP430 port pins of ports 1 through 6 are brought out on 8-pin headers. This allows easy access to the signals used to communicate with the TUSB3410, as well as for attaching any custom circuitry. Furthermore, four push-buttons (SW1…SW4) and four LEDs (LED1…LED4) are connected to I/O port 4 for demonstration purposes. An MSP430-standard 14-pin JTAG header is provided for in-system debugging and programming. Note that the board's VCC is routed to pin 4 of the JTAG connector to allow the JTAG debugger to adjust its voltage levels.

The circuitry external to the TUSB3410 has been designed according to [5], [6], and [10]. This reference design uses an external EEPROM (U5) with I^{2}C interface for USB configuration parameter storage. The EEPROM size can be selected according the amount of data that needs to be stored. More information on EEPROM selection can be found in [11]. Note that there is a jumper (JP1) in the EEPROM SCL line that allows disconnecting the EEPROM. In this case, the TUSB3410 would report its standard VID/PID values to the host upon USB connection. The 12-MHz crystal Q2 provides the required clock for device operation. Table 2 shows the signal connections made between TUSB3410 and MSP430.

Table 2. MSP430F16x-TUSB3410 Signal Connections

MSP430 Signal	TUSB3410 Signal	Description
P3.5/URXD0	SOUT	Data received from the USB
P3.4/UTXD0	SIN	Data sent to the USB
P2.2/TA0†	SOUT	Receive data connection for BSL
P1.1/TA0†	SIN	Transmit data connection for BSL
P3.1/SIMO0/SDA†	SDA	I2C data line. Also connected to EEPROM (U5).
P3.3/UCLK0/SCL†	SCL	I2C clock line. Also connected to EEPROM (U5).
P3.0/STE0†	RESET	Signal allows controlling the TUSB3410 operation.
XT2IN	CLKOUT‡	Allows TUSB3410 clock output to be used for the MSP430
RST/NMI†	DTR	Reset connection used for BSL entry
TCK†	RTS	JTAG TCK connected used for BSL entry
P1.6/TA1	DSR‡	UART hardware handshake signal
P1.7/TA2	CTS‡	UART hardware handshake signal

† The MSP430 can be disconnected from this signal by not populating a 0R resistor. See application schematic for details.
‡ This signal is not used by the software provided with this application report.

For the actual data transmission, the MSP430 USART0 module is used in UART mode. It can handle all baud rates supported by the TUSB3410. This module is also used in I^{2}C mode for communication with the EEPROM. This enables in-system programming of the external EEPROM through the MSP430 and eliminates the need for using an external EEPROM programmer. This method is used by the software provided here. Note that these connections are optional and that an I^{2}C master can easily be implemented with GPIO pins on any MSP430. As a design idea, available EEPROM space could also be used as additional MSP430 storage.

In addition to the serial connections to USART0, the same signals are also connected to the MSP430 BSL pins. This way, a blank MSP430 can be programmed through the USB UART link. Refer to section 4.5 for more information.

The connection from the MSP430 P3.0/STE0 pin to the TUSB3410 RESET pin allows keeping the TUSB3410 in reset state while the MSP430 is accessing the EEPROM. Also, by cycling this signal, a USB unplug-replug event can be generated without actually disconnecting the application PCB.

3.2 Lower-Cost Solution Using MSP430F2274

The schematics of the lower-cost version of the MSP430-TUSB3410 reference design using an MSP430F2274 can be found in section B.1. Also, PCB layout and BOM information can be found in sections B.2 and B.3. The two main components that can be identified are the TUSB3410 USB-to-serial bridge controller (U2) and the MSP430F2274 (U1). This particular mid-range MSP430 device was selected because of its 16-MHz architecture and its USCI communication module to allow for a lower-cost implementation. Any MSP430F22xx device can be directly used with this hardware design.

The TUSB3410 USB data lines are connected to a standard USB B-type PCB mount connector. TI's transient voltage suppressor SN75240 (U3) is used on the USB lines to provide an increased level of ESD protection.

The reference design hardware is designed as a bus-powered USB application. The USB supplies 5 V on each port and devices can generally draw up to 100 mA from the bus without any special considerations. However, up to 500 mA can be made available by the host upon request. Here, a TI TPS77301 LDO is used (U4) to generate 3.6 V with a maximum output current of 250 mA. This supply voltage is used to power the entire circuit. LED5 is illuminated when power is supplied by the USB.

For this reference design, the MSP430 is operated at 12 MHz. This clock is provided by Q1, which is connected to the LFXT1 oscillator. All MSP430 port pins of ports 1 through 4 are brought out on 8-pin headers. This allows easy access to the signals used to communicate with the TUSB3410 as well as for attaching any custom circuitry. Furthermore, four push-buttons (SW1…SW4) and four LEDs (LED1…LED4) are connected to I/O port 1 for demonstration purposes. An MSP430-standard 14-pin JTAG header is provided for in-system debugging and programming. The connections make use of the 2-wire JTAG protocol ("Spy-Bi-Wire") and, therefore, require the use of an appropriate JTAG emulator. Note that the board's VCC is routed to pin 4 of the JTAG connector to allow the JTAG debugger to adjust its voltage levels.

The circuitry external to the TUSB3410 has been designed according to [5], [6], and [10]. This version of the reference design operates the MSP430F2274 USCI_B0 communication module in I^{2}C slave mode to emulate the external USB parameter storage EEPROM. The actual data transmission is handled by the USCI_A0 module used in UART mode. The TUSB3410 is sourced by a 12-MHz clock signal that is output by the MSP430 through the P2.0/ACLK pin. Note that there is a voltage divider used before the signal is fed into the TUSB3410 X1/CLKI pin to not exceed the maximum allowed input voltage level as per TUSB3410 datasheet [5]. Table 2 shows the signal connections made between TUSB3410 and MSP430.

Table 3. MSP430F22xx-TUSB3410 Signal Connections

MSP430 Signal	TUSB3410 Signal	Description
P3.5/UCA0RXD	SOUT	Data received from the USB
P3.4/UCA0TXD	SIN	Data sent to the USB
P3.1/UCB0SDA	SDA	I2C data line
P3.2/UCB0SCL	SCL	I2C clock line
P3.0	RESET	Signal allows controlling the TUSB3410 operation.
P2.0/ACLK	X1/CLKI	Allows MSP430 clock output to be used for the TUSB3410

The connection from the MSP430 P3.0 pin to the TUSB3410 RESET pin allows keeping the TUSB3410 in reset state while the MSP430 is accessing the EEPROM. Also, by cycling this signal, a USB unplug-replug event can be generated without actually disconnecting the application PCB.

4 Reference Design Software

4.1 Overview

The ZIP archive associated with this application report contains a variety of different files. Table 4 provides an overview.

Table 4. ZIP Archive Contents Overview

Folder / File(s)	Description
EXE\TUSB3410Demo.exe	PC demo application
EXE\BSLDEMO2.exe†	BSL software, modified for code download through USB
EXE\F16xDemoFirmware.txt†	Demo firmware image in MSP430-TXT format, MSP430F16x solution
EXE\F16xFET1.txt†	Flashing LED demo software image in MSP430-TXT format, MSP430F16x solution
F16X_GERBERS*.*	Board layout files in Gerber format, MSP430F16x solution
F22XX_GERBERS*.*	Board layout files in Gerber format, MSP430F22xx solution
MSP430F16X_SW_CCE*.*	Code Composer Essentials V2.0 MSP430 source code, MSP430F16x solution
MSP430F16X_SW_IAR*.*	IAR Embedded Workbench V3.41A MSP430 source code, MSP430F16x solution
MSP430F22XX_SW_CCE*.*	Code Composer Essentials V2.0 MSP430 source code, MSP430F22xx solution
MSP430F22XX_SW_IAR*.*	IAR Embedded Workbench V3.41A MSP430 source code, MSP430F22xx solution
MSP430FG461x_EEPROM_CCE*.*	Code Composer Essentials V2.0 MSP430FG461x code example of an EEPROM emulation
MSP430FG461x_EEPROM_IAR*.*	IAR Embedded Workbench V3.41A MSP430FG461x code example of an EEPROM emulation
MSP430_TUSB3410_2KXP_V103*.*	Windows VCP driver for MSP430-TUSB3410 reference design
ORIG_TUSB3410_2KXP_V103*.*	Windows standard TUSB3410 driver
PC_BSL*.*†	Microsoft Visual C++ source code of BSL software
PC_DEMO*.*	Microsoft Visual C++ source code of PC demo software
TUSB3410_EEPROM*.*	EEPROM image used for MSP430-TUSB3410 reference design

† These files can only be used in conjunction with the BSL-capable version of the reference design.

4.2 MSP430F16x Demo Firmware

The MSP430 software provided with this application report is contained in the single C file "MSP430F16x-TUSB3410_Demo.c". Versions for use with IAR Embedded Workbench and TI's Code Composer Essentials are included. The software demonstrates in-system EEPROM programming (U5) and serial communication with 460,800 Baud. The lower nibble of a received character is output to the LEDs on the demo board and, on press or release of any push button, a byte is transmitted to the PC containing the updated button state. The software is intended to be used together with the PC software as presented in section 4.4, but can also be used to communicate with any terminal software such as HyperTerm. Figure 3 and Figure 4 give an overview of the software flow.

Upon MSP430 reset, the function InitSystem() is called to configure used peripherals and the application board. Watchdog timer and I/O ports are set up for proper use, and the clock system is set up to operate from the external crystal with a frequency of 8 MHz. Furthermore, USART0 is set up for I^{2}C operation, which will be used for communication with the EEPROM. Refer to [1] for more information on MSP430F1xx module operation.
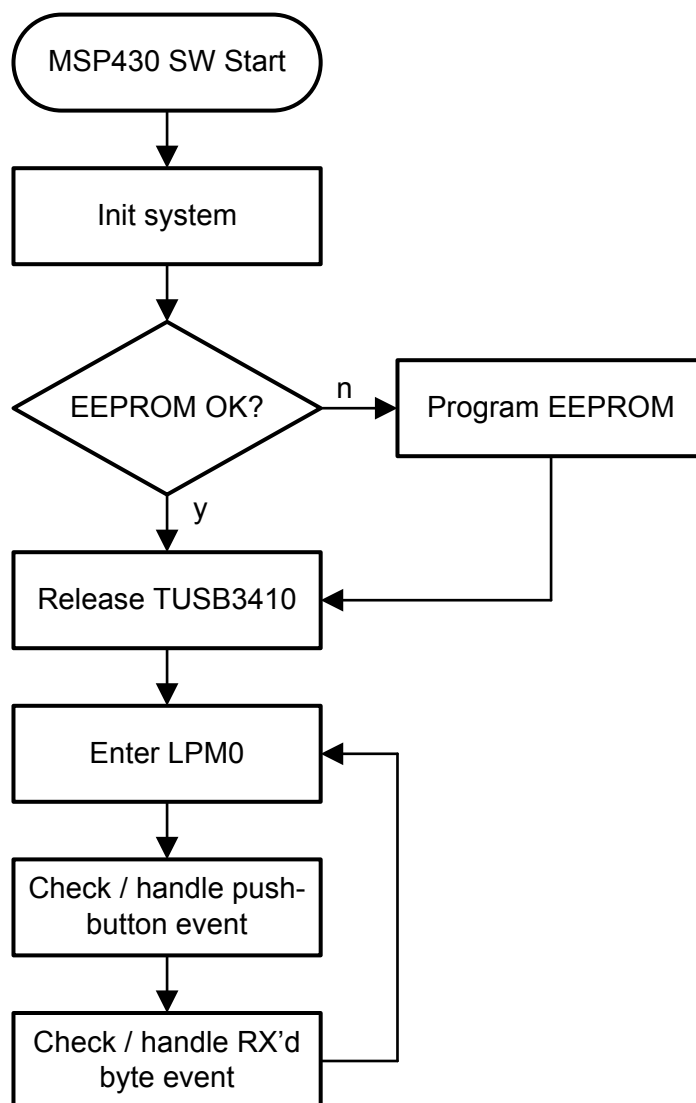
Figure 3.   MSP430F16x Demo Firmware main() Flow
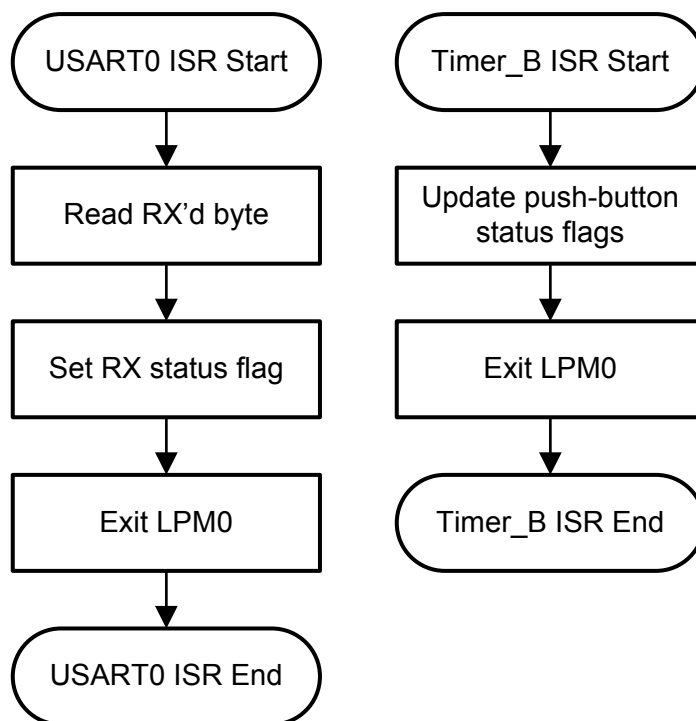
Figure 4.   MSP430F16x Demo Firmware Interrupt Handler Flow

As part of the initialization sequence, the TUSB3410 is brought into a reset state. In this mode, the MSP430 can access the I^2C bus without interfering with the TUSB3410 operation. Then, the presence of an external EEPROM is detected by sending out its address and checking for a valid I^2C ACK condition. In the case an ACK was received, the EEPROM contents are verified against the image stored in the MSP430 Flash memory constant EEPROMImage[] by calling EEPROM_Verify(). In the case of this reference design, the size of this image is ~140 Byte. If there is a mismatch, e.g., when the EEPROM is blank, the EEPROM contents is programmed by calling EEPROM_Write(). After the EEPROM update procedure, the TUSB3410 reset signal is de-asserted and the device resumes normal operation. Note that at this point, the MSP430 RESET pin is deactivated by switching it to NMI mode. This is necessary due to the BSL capability of the solution presented here (see chapter 3) and the fact that the TUSB3410 toggles the DTR signal for a short while after device connection, which would cause unintended MSP430 resets. When the TUSB3410 resumes operation, it reads out the EEPROM contents, and then connects to the USB host controller reporting the configuration data that was stored in the EEPROM.

The last step is the configuration of USART0 for UART mode. This mode is used for the actual data communication. A bit rate of 460,800 Baud using 8 data bits, 1 stop bit, and no parity is configured. One may adjust the communication speed to meet specific application requirements. The MSP430 is capable of communicating with the TUSB3410 at all Baud rates this device offers, which is up to 921,600 Baud. It must be confirmed that the settings on the PC side, which are used to open the virtual COM port (VCP), match the settings for which the MSP430 is configured. See section 4.4 for more information.

After the system initialization, the main() function configures the Timer_B7 module of the MSP430F1612 to be used to query the status of the push-buttons SW1…SW4. The capture/ compare (CC) blocks which are connected to the push-button signal lines are set up to capture the rising edge and generate an interrupt. The TIMERB0_ISR() and TIMERB1_ISR() functions are executed upon button press and switch the CC block in question to compare mode, thus effectively polling the button state with a defined interval. This way, an effective button query and debounce are implemented. The three flag variables ButtonState, ButtonSet, and ButtonReleased are used to indicate to the main() program context any push-button status change. Furthermore, the CPU is awakened on any push-button events. This is used to implement event-driven program flow.

After Timer_B7 setup, the USART0 module receive interrupt is activated, and the main event handling loop is entered. The program resides in low-power mode 0 (all clocks are on, CPU is off) until an event occurs. On any button event, the updated button state is transmitted by loading the variable ButtonState into the USART0 transmit buffer. Note that data is only transmitted if there is an actual change in the state of any push-button. If a UART character is received, LED1…LED4 are set according to the lower nibble of the received byte.

In a custom application, where more data is moved over the USB, the MSP430 DMA controller can be used to enable fast and low CPU overhead movement of large data blocks. Furthermore, USB devices usually buffer bytes before actually transmitting them; therefore, transferring data in blocks rather than byte-by-byte is faster.

4.3 MSP430F22xx Demo Firmware

The MSP430 software provided with this application report is contained in the single C file "MSP430F22xx-TUSB3410_Demo.c". Versions for use with IAR Embedded Workbench and TI's Code Composer Essentials are included. The code discussed here is similar to the one described in section 4.2. The main difference is that the MSP430 is used to emulate the USB configuration EEPROM.

The application demonstrates serial communication with 460,800 Baud using USCI_A0 in UART mode. The lower nibble of a received character is output to the LEDs on the demo board, and on press or release of any push-button, a byte is transmitted to the PC containing the updated button state. The software is intended to be used together with the PC software as presented in section 4.4, but can also be used to communicate with any terminal software such as HyperTerm. Figure 5 and Figure 6 give an overview about the software flow.

Upon MSP430 reset, the function InitSystem() is called to configure used peripherals and the application board. Watchdog timer and I/O ports are set up for proper use, and the clock system is set up to operate from the external crystal with a frequency of 12 MHz. This clock is also being output to port pin P2.0/ACLK to supply a 12-MHz clock to the TUSB3410, which is required for its operation. Note that while the system is configured, the TUSB3410 is held in a reset state. As a part of the configuration process, the USCI_B0 module is set up for I²C slave-mode operation and used to emulate the configuration EEPROM. Refer to [2] for more information on MSP430F2xx module operation.
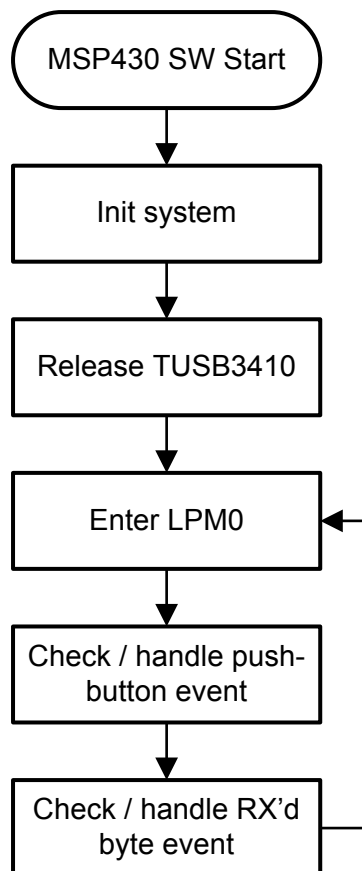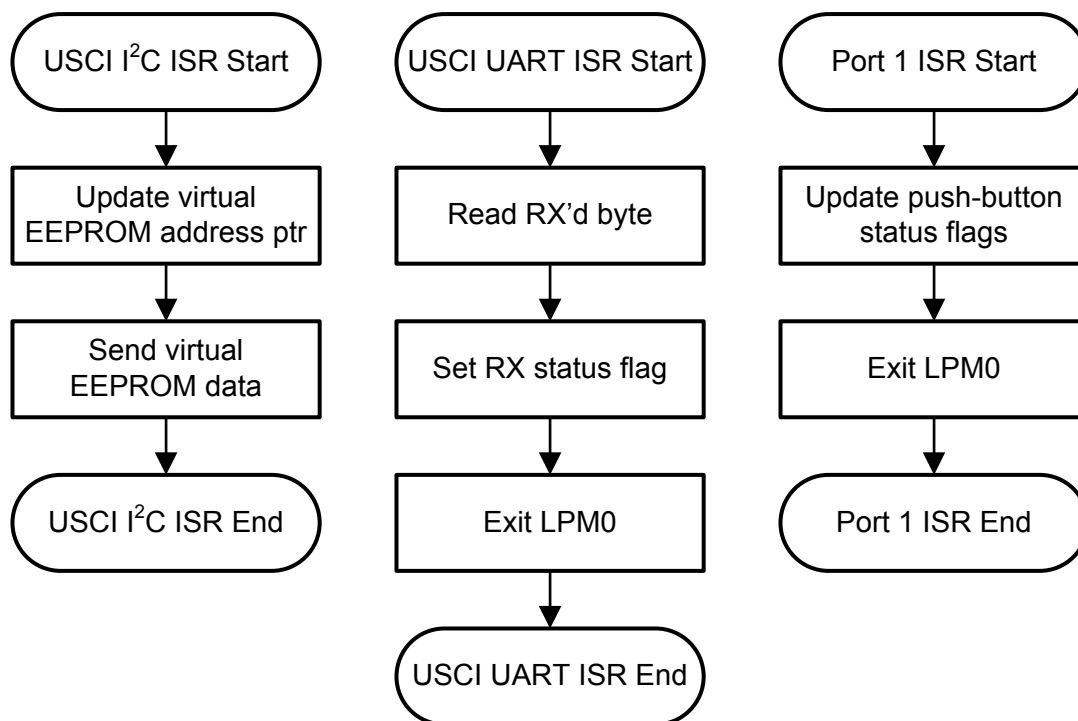
Figure 5.   MSP430F22xx Demo Firmware main() Flow

By setting the slave mode address to 0x50, the MSP430 reacts on I²C master transactions coming from the TUSB3410 after it is powered up. An optimized software approach, combined with the CPU execution speed of 12 MHz, enables delay-free reaction on I²C fast-mode transactions. This is needed due to the non I²C-compliant implementation of the TUSB3410 I²C module (see note below). The only two transactions that are supported by the MSP430 firmware are the setting of the 16-bit virtual EEPROM address pointer and the read-out of consecutive data bytes while auto-incrementing the address pointer. The I²C EEPROM contents itself is contained in the MSP430 Flash memory constant EEPROMImage[], while the variable EE_AddrPtr is used as the address pointer. In the case of this reference design, the size of this image is ~140 Byte.

NOTE: MSP430 USED FOR EEPROM EMULATION
The time from the falling edge of SCL of the TUSB3410 I²C read-transaction's R/W bit until the USCI_B0 transmit buffer UCB0TXBUF is loaded is CRITICAL. The transmit buffer must have been loaded BEFORE the falling edge of the read-transaction's address byte's clock signal SCL. This window is exactly 2.5 µs (one SCL clock period in I²C fast mode). This timing requirement can be met with an MSP430 MCLK frequency of at least 12 MHz, as used in the application. Alternatively, a DMA channel can be used instead to load the transmit buffer allowing it to meet the I²C timing requirement at much lower CPU speeds.

Figure 6.   MSP430F22xx Demo Firmware Interrupt Handler Flow

The last step is the configuration of USCI_A0 for UART mode. This mode is used for the actual data communication. A bit rate of 460,800 Baud using 8 data bits, 1 stop bit, and no parity is configured. One may adjust the communication speed to meet specific application requirements. The MSP430 is capable of communicating with the TUSB3410 at all Baud rates this device offers, which is up to 921,600 Baud. It must be made sure that the settings on the PC side, which are used to open the virtual COM port (VCP), match the settings the MSP430 is configured for. See section 4.4 for more information. Furthermore, the USCI_A0 module receive interrupt is activated.

Now, the TUSB3410 reset signal is deasserted, and the device resumes normal operation. Note that at this point, the TUSB3410 reads out the EEPROM image from the MSP430 Flash memory. As a next step, it then connects to the USB host controller reporting the configuration data that it just read out from the MSP430.

After the system initialization, the main() function, processes all occurring events, is entered. The program resides in low-power mode 0 (all clocks are on, CPU is off) until an event occurs. On any button event, the updated button state is transmitted by loading the variable ButtonState into the USCI_A0 transmit buffer. Note that data is only transmitted if there is an actual change in the state of any push-button. If a UART character is received, LED1…LED4 are set according to the lower nibble of the received byte.

Push button presses SW1…SW4 trigger the execution of the PORT1_ISR(). Inside this ISR, the Timer_A is activated and used to generate a periodic interrupt. The associated TIMERA1_ISR() is then used to poll the state of all pushed buttons with a defined interval. This way, an effective button-query and de-bounce is implemented. The ISR is executed periodically as long as at least one button is pressed. The three flag variables—ButtonState, ButtonSet, and ButtonReleased—are used to indicate to the main() program context any push-button status change. Furthermore, the CPU is awakened on any push-button events. This is used to implement event-driven program flow.

Received characters cause the USCIAB0RX_ISR() to be executed. There, the data is simply fetched from the module and a flag is set indicating to the main event handler that data was received. The CPU is then awakened from low-power mode to process any pending events within the main event handler.

4.4 PC Demo Application

The PC demo application provided with this application report is designed to work together with both the MSP430 firmware versions. It is written as a Windows 32-Bit console application and runs under Windows XP and 2000. The application's source code is contained in a single file, which is also included. It was successfully built under both Microsoft Visual C++ and Borland C++.

The purpose of this program is to demonstrate how the MSP430-TUSB3410 reference design hardware can be automatically detected and used. The auto-detection feature makes this solution more user friendly as the need for manual inspection of the Windows device manager for the board's associated VCP number is removed. The keys <1>…<4> are used to light the corresponding LEDs on the reference design board, and push-button presses on the demo board are output to the console. Figure 7 gives an overview of the program operation.
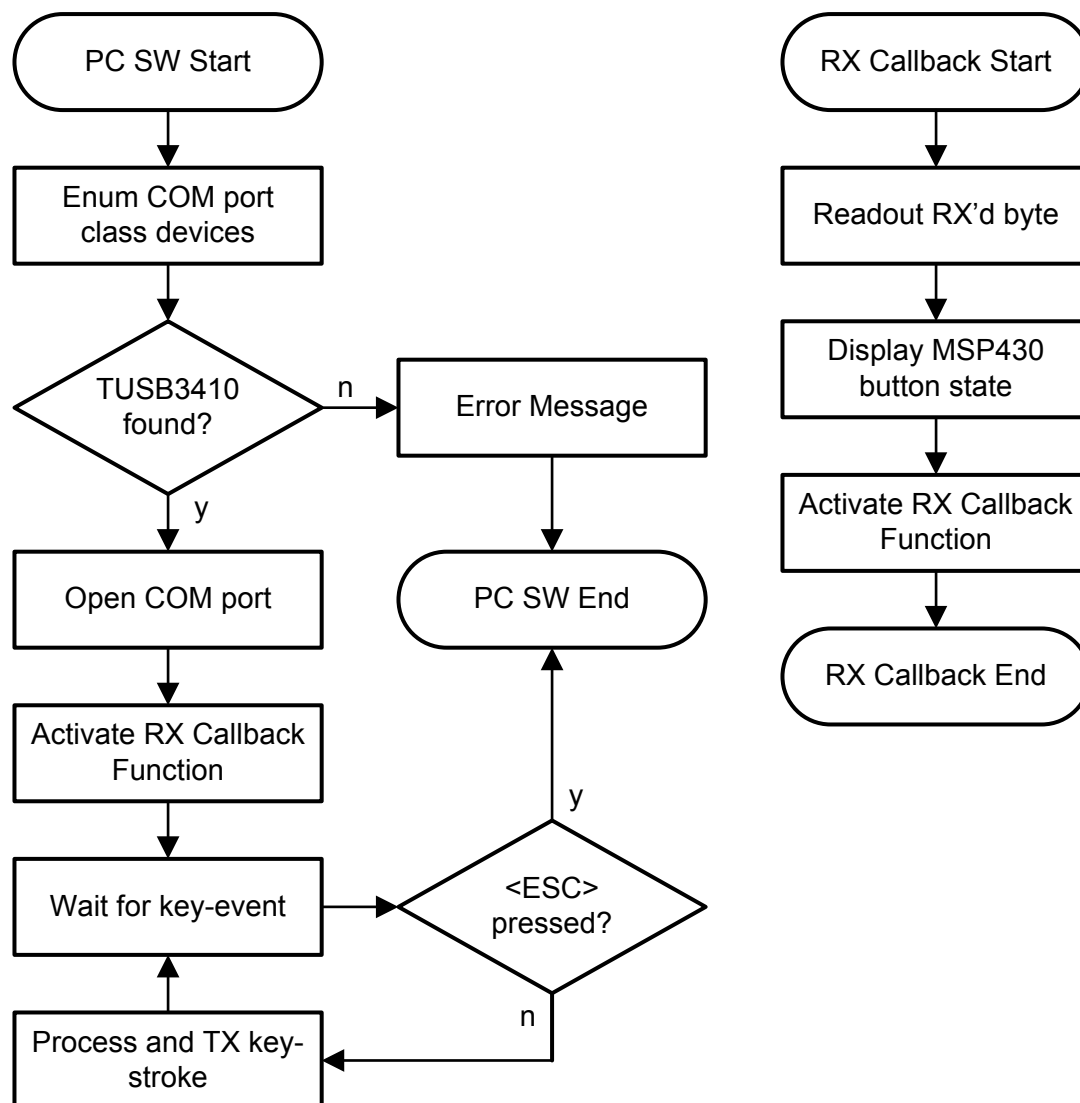
Figure 7.   PC Demo Application Flow

The program is started by executing "TUSB3410Demo.exe". After outputting a welcome message, the demo program calls the function EnumComPorts(). This function is used to identify which COMnn port[s] are attached to the Texas Instruments TUSB3410 Universal Multiport (UMP) controller. On the initial function call with the parameter dwIndex set to 0, it enumerates all currently connected serial communication devices (GUID_CLASS_COMPORT). Then, it checks the hardware ID (HWID) of the virtual COM port (stored in the TUSB3410 driver INF-files) and returns the COMnn port, which Windows embeds in the "Friendly Name". The TUSB3410 drivers include the ability to customize the "Friendly Name" before the "(COMnn)", and the HWID after the "umpport\".

The HWID search string is contained in the local constant szHardwareIDump[] within the EnumComPorts() function. For this reference design, it is set to "umpport\VID_0451_BEEF_com", as this corresponds with the entries made in the TUSB3410 driver INF file (see section 5.4). This constant needs to be customized for different USB devices. EnumComPorts() returns a result code that indicates if the device referred to with the list index (dwIndex) matches the HWID. Subsequent calls to EnumComPorts() can be used to identify all connected devices. A return value of ERROR_SUCCESS indicates a match, and the parameter lpszName contains the associated COM port. The code ERROR_NO_MORE_ITEMS indicates that the end of the enumeration list has been reached. EnumComPorts() uses the Windows SetupAPI for reading out low-level device interface data. Note that this requires the library "setupapi.lib" to be linked to the project for a successful build. More information on SetupAPI can be found in the Windows Driver Kit section on [15].

If the MSP430-TUSB3410 Reference Design is found, a message is output, displaying the associated VCP port number. This number should match what is shown in the Windows device manager. After this, the VCP is opened like any other COM port in Windows using the Windows API call CreateFile(). Note that the COM port is opened with the attribute OVERLAPPED to allow for an event-driven program flow. Also, port configuration and the actual data transfer do not differ as if a real hardware serial port is used. Extensive information on serial port programming can be found on [15].

The VCP is configured by filling a Windows DCB structure with all the parameters. Note that a baud rate of 460,800 with 8 data bits, 1 stop bit, and no parity is used and that any flow control mechanisms are disabled. These settings must match the MSP430 USART settings for successful communication.

After this, the main event processing loop is entered. At the beginning of this loop, the program enters an alterable wait mode using the function WaitForSingleObjectEx(). When any console keyboard event is detected, the wait mode is exited and the program checks the keyboard input buffer for a press or release of any of the keys <1>…<4>. If the status of these keys changes, the variable PCButtonState is updated and the new keyboard state is transmitted over the VCP to the MSP430-TUSB3410 reference design board using WriteFile(). In analogy to the MSP430 firmware, the lower nibble of the transmitted byte contains the current state of the keys <1>…<4>. Note that data is only transmitted if there is an actual change in the state of these keys. Also, if <ESC> is pressed, a flag gets set, the event processing loop is left, the VCP is closed by calling CloseHandle(), and the program is exited.

For any character that is received from the demo board through the VCP connection, the function FileIOCompletionRoutine() is automatically executed by Windows. This is possible as the COM port is used in OVERLAPPED mode. This function serves as an event handler, reads out the received byte from COMBuffer[] and displays the updated MSP430 button state to the console.

4.5   Customized Bootstrap Loader Tool

With the MSP430F16x-TUSB3410 reference design presented in section 3.1, it is possible to use the MSP430 built-in bootstrap loader (BSL) firmware to program a blank MSP430 through the USB interface. Details on the operation of the BSL can be found in [13]. TI also provides an application report that presents a BSL programming solution consisting of serial-port hardware and PC software [14].

All the connections needed for BSL programming as required by [14] are made (see Table 2). When comparing TI's BSL programmer hardware with the reference design, it can be seen that the two serial handshake lines, DTR and RTS, which are used to invoke the BSL, are inverted. To accommodate this, minor modifications were made to the BSL software that was available on the web at the time of writing this application report to invert these two signals in software. The modified source code and a new PC application are provided as part of this application report.

To program a blank MSP430F16x-TUSB3410 reference design board, the following flow can be used:

1.) Generate an MSP430-TXT output file of the program you want to download.

2.) Connect the application board to a PC USB port.

3.) Install the standard TI TUSB3410 VCP drivers for the "TUSB3410 Boot Device". This is the device the TUSB3410 reports as with its default configuration (either with blank EEPROM, or with disconnected EEPROM by removing JP1). These drivers are supplied with this application report. For the most up-to-date drivers, always refer to [7].

4.) Open the Windows device manager to identify the VCP to which the TUSB3410 is connected (see Figure 9).

5.) From the command line, start the provided BSL tool BSLDEMO2.exe with the COM port and the MSP430-TXT file to program as parameters. For example, to download the blinking LED demo code supplied with this application report to an MSP430 connected on COM5, the following command line would need to be executed:

C:\>BSLDEMO2 –cCOM5 FET1.TXT

After starting, the tool connects to the MSP430 internal BSL via USB and download the code file. After this is finished, the MSP430 RESET signal is released and the code execution starts. All four LEDs on the demo board should now be blinking.

Note that when re-flashing an MSP430 that is running the firmware presented in section 4.2, one additional step must be performed. While the firmware is running, it deactivates the MSP430 RESET pin to prevent unintended device resets due to DTR serial handshake line operation, but this also disables the ability for the BSL to gain control over the device. In order to maintain reset functionality and BSL download capability, hold down any of the push-buttons SW1...SW4 from the moment the board is connected until the BSL download is completed. Doing so traps the firmware in an endless loop and switch on all LEDs. During the connection process, the board's LEDs blink several times as the TUSB3410 operates the DTR handshake signal.

5 Reference Design USB Drivers

5.1 Overview

Together with this application report, two sets of drivers are provided (see Table 4). The drivers are based on the current TI TUSB3410 driver suite at the time of writing this application report. It is recommended to check the TUSB3410 product folder on the TI web page for possible driver updates [7]. The driver is available under the order number TUSBWINVCP. Note that per driver installation, two actual drivers get installed: one universal multi-port serial adapter (UMP) and one virtual serial COM port that makes use of the services provided by the UMP. The dual-driver setup is due to the chosen TUSB3410 Windows driver architecture.

TI's current TUSB3410 solution has successfully passed WHQL certification. Therefore, one should also be able to achieve WHQL certification with a customized solution using different VID and PID values.

5.2 Manual Driver Installation

All drivers supplied with this application report must be manually installed upon USB hardware connection. When Windows detects the new device, the Found New Hardware Wizard pops up (Figure 8). A different device name is shown, depending on whether or not an EEPROM is used for USB parameter storage, and the descriptor strings are stored there. The screenshot shows the wizard when the standard TUSB3410 descriptor string is used.

Figure 8. Found New Hardware Wizard

On the first dialog page, "Install from a list or specific location" must be selected. Then, one must point the wizard to the actual directory where the driver files reside. The directory choice depends on if the reference design EEPROM is blank (or JP1 is removed), or the EEPROM was already programmed by the MSP430 firmware. Refer to Table 4 for folder names. For the remainder of the driver installation, simply push "Next >" until everything is done. After the first driver is installed (Multi-port serial adapter, UMP), the Found New Hardware Wizard pops up again and asks for drivers for the actual VCP. Again, direct the wizard to the same directory, and push "Next >" until the second driver is installed. After installation, the Windows device manager should appear as shown in Figure 9. Behind the VCP port name, the associated COMnn number of the VCP is displayed. This number can immediately be used to communicate with the demo board. The PC software provided with this application report shows how this VCP port number detection process can be automated. See section 4.4 for details.
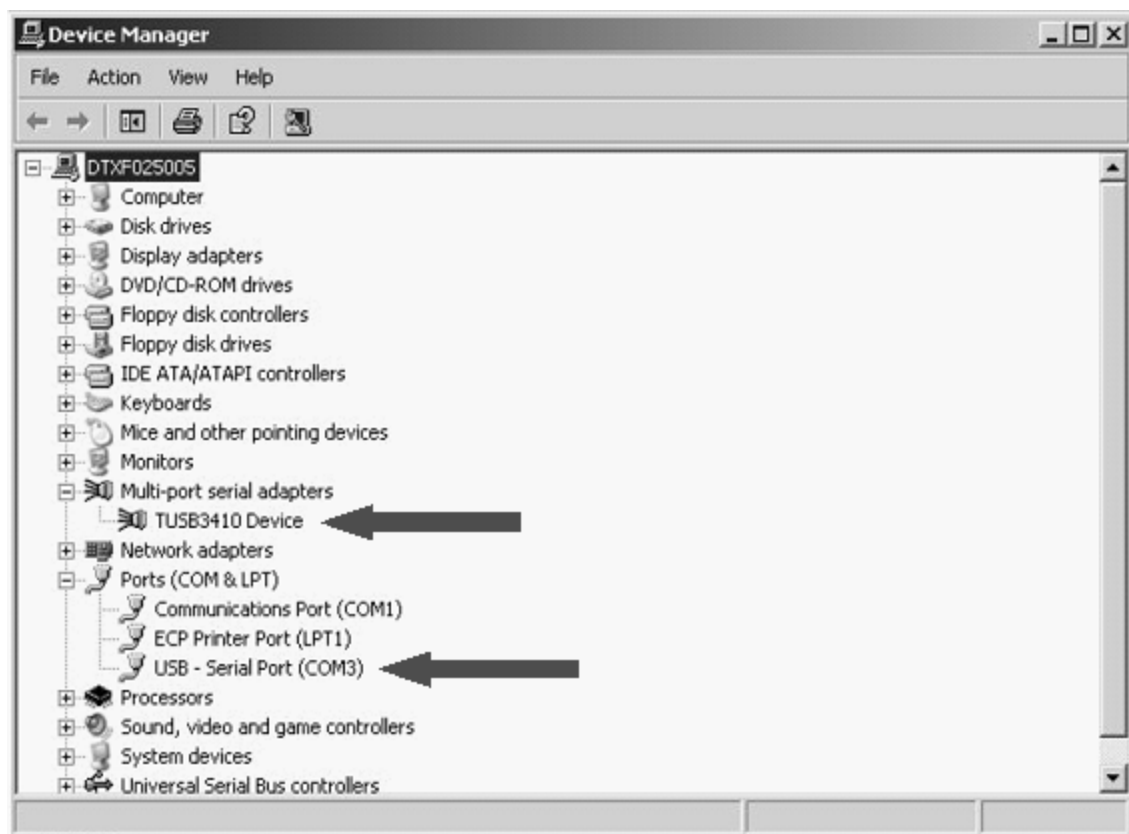
Figure 9.   Device Manager After Driver Install

5.3   Standard TUSB3410 Driver

This driver suite is used in the case the EEPROM on the reference design hardware is blank or jumper JP1 is removed at the time the board is connected. In this case, the board reports as "TUSB3410 Boot Device". The "Found New Hardware" wizard must be directed to the "ORIG_TUSB3410_2KXP_V103" folder. After driver installation, the TUSB3410 VCP is listed in the Windows device manager and can be used, e.g., to download code to a blank MSP430 using the BSL tool as described in 4.5.

5.4 Custom MSP430-TUSB3410 Reference Design Driver

The other driver supplied with this application report is a customized TUSB3410 driver for use with the VID/PID values used with the reference design demo board (Table 1). Also, the descriptor strings were modified to show the board as "MSP430-TUSB3410 Reference Design" in the Windows device manager. The driver suite basically is the standard TUSB3410 set of driver files, with the two modified INF-files umpusbXP.inf (for the UMP) and UmpComXP.inf (for the VCP). Inspect these files to see exactly which changes were made.

The process for creating a custom application-specific driver set is to take the standard TUSB3410 driver suite and then modify these two INF-files using a text editor. Throughout these files there are instructions and hints, placed as comments, that indicate places to fill in custom VID, PID, and descriptor string values. Make sure that the UMP hardware ID used in both driver INF-files matches and is unique to your company and product. Note that for a custom application, it is recommended to delete all comments regarding occurrences of TI's VID/PIDs throughout these files. Otherwise, conflicts with different USB devices using the TUSB3410 drivers can occur.

5.5 Automated Driver Installer

The TUSB3410 driver suite, which is available in the TUSB3410 product folder, also contains a driver installer. It is an InstallShield project that allows installing the Windows drivers prior to USB device connection. Using this installer results in an easier driver installation, as Windows is already aware of the device drivers by the time the actual device gets connected, and the need for manually pointing the "Found New Hardware" wizard to the driver directory is removed. In this case, the user only needs to push "Next >" in the dialog boxes until everything is done.

The driver installer can be customized for VID / PID values without rebuilding the actual InstallShield project. The automated driver and instructions on how to customize are included in the TUSBWINVCP driver package, which is available on [7].

6 Other Lower-Cost Options

6.1 No External EEPROM

It is possible to design a TUSB3410-based device that does not use an external EEPROM and also doesn't use the MSP430 to emulate one. This solution reports the default descriptors located in the TUSB3410 boot code, including TI's default VID/PID, and the TUSB3410 firmware is downloaded from the host. However, doing so has two consequences:

1.) The solution is not USB-compliant because it does not have the vendor's unique VID.

2.) The host that sees two devices with the same VID/PID and same serial number may not allow both of them to function.

For these reasons, TI strongly discourages this configuration, unless the system is known to be an isolated bus that will not encounter any unknown devices that may conflict with your device.

TEXAS INSTRUMENTS

6.2 No External MSP430 Crystal

The TUSB3410 can output its UART baud clock or a fixed frequency of 3.556MHz on pin CLKOUT. This signal can be directly connected to an MSP430 oscillator input (XIN or XT2IN) with the respective oscillator operated in high-speed mode. This way, a stable clock source can be made available to the MSP430 and used for CPU and peripheral operation, without attaching an external crystal.

However, this option requires the customer to modify the existing TUSB3410 firmware. A line of code needs to be added which enables the CLKOUT feature by setting the bit CLKOUTEN in the MODECNFG configuration register. The bit CLKSLCT allows selecting between the UART clock and the fixed 3.556-MHz frequency output.

The modified firmware needs to be stored either in the external configuration EEPROM or included into the Windows driver package (file name: umpf3410.i51). In the latter case, the new firmware automatically is downloaded by the Windows drivers once the application is connected. The MSP430 OSCFAULT detect feature can be used to determine if the external clock is present or not and, if so, use it. For example, with a TUSB3410 output clock frequency of 7.3728MHz driving the MSP430 USART module (divide by 8), high-speed UART communication with 921,600 Baud can be established. The TUSB3410 firmware source code package is available from TI upon request.
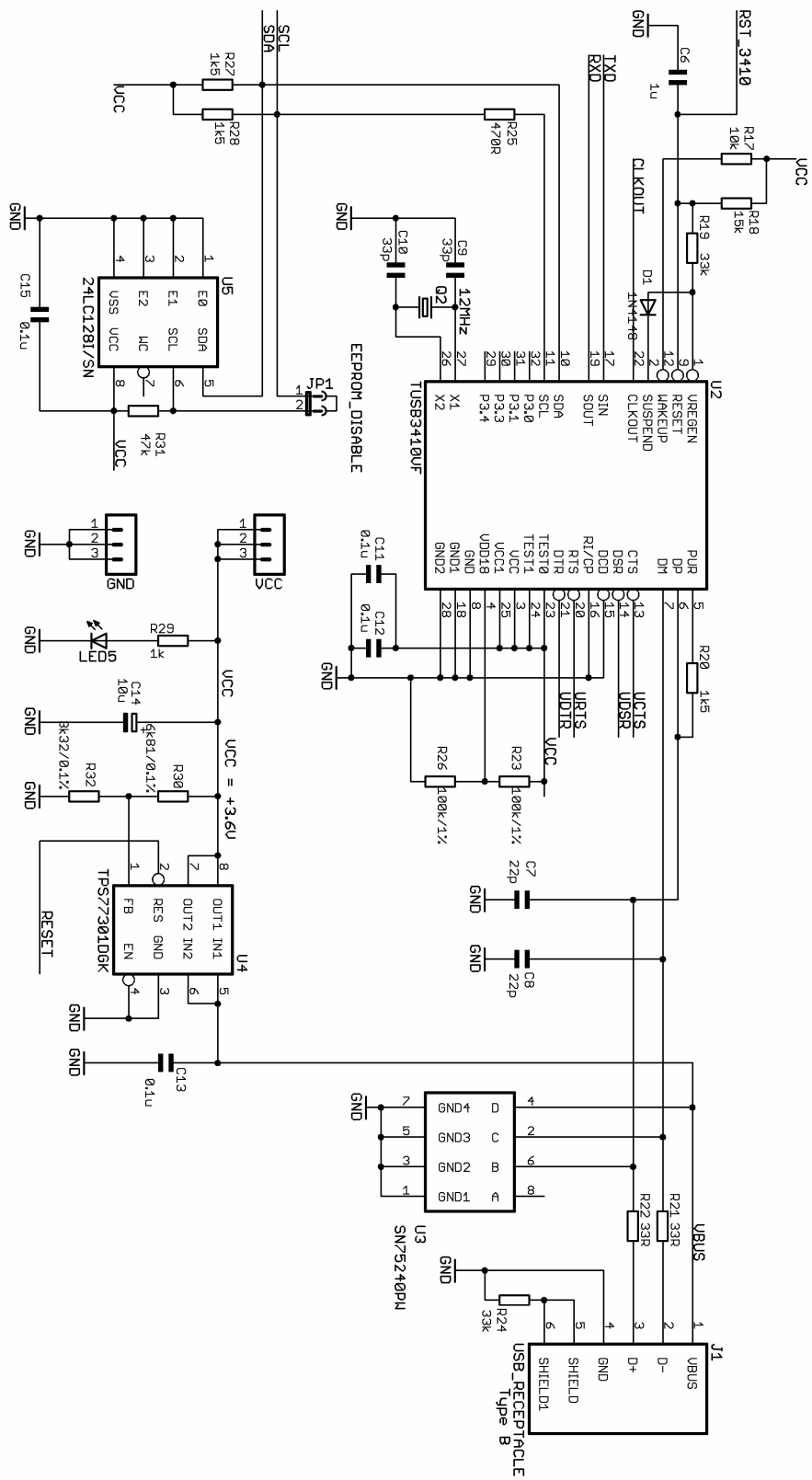
7 Summary

The MSP430-TUSB3410 reference design presents an example of how USB connectivity can be achieved with MSP430 devices without hardware USB module. Transfer speeds of up to ~1Mbit/s can be achieved reliably. Extensive support collateral is available from TI, allowing a hassle-free implementation.

8 References

1. *MSP430x1xx Family User's Guide* (SLAU049)
2. *MSP430x2xx Family User's Guide* (SLAU144)
3. *MSP430F16x Mixed Signal Microcontroller Datasheet* (SLAS368)
4. *MSP430F22xx Mixed Signal Microcontroller Datasheet* (SLAS504)
5. *TUSB3410, TUSB3410I USB To Serial Port Controller Data Manual* (SLLS519)
6. *TUSB3410 Errata* (SLLZ021)
7. *TUSB3410 Product Folder* (http://focus.ti.com/docs/prod/folders/print/tusb3410.html)
8. *VIDs, PIDs and Firmware: Design Decisions When Using TI USB Device Controllers* (SLLA154)
9. *TI USB EEPROM Burner Utility for the TUSB3410* (SLLC259)
10. *TUSB3410UARTPDK User's Guide* (SLLU043)
11. *USB/Serial Applications Using TUSB3410/5052 and the VCP Software* (SLLA170)
12. *USB Header Generator For VCP Applications* (SLLC251)
13. *Features of the MSP430 Bootstrap Loader* (SLAA089)
14. *Application of Bootstrap Loader in MSP430 w/Flash* (SLAA096)
15. *Microsoft Developer Network* (http://msdn.microsoft.com)

TEXAS INSTRUMENTS

Appendix A. F16x Solution Hardware Description

A.1 Schematics

TITLE: TUSB3410_Demo

MSP430F16x-TUSB3410 Reference Design

Document Number::

Date: 9/29/2006 09:41:56a
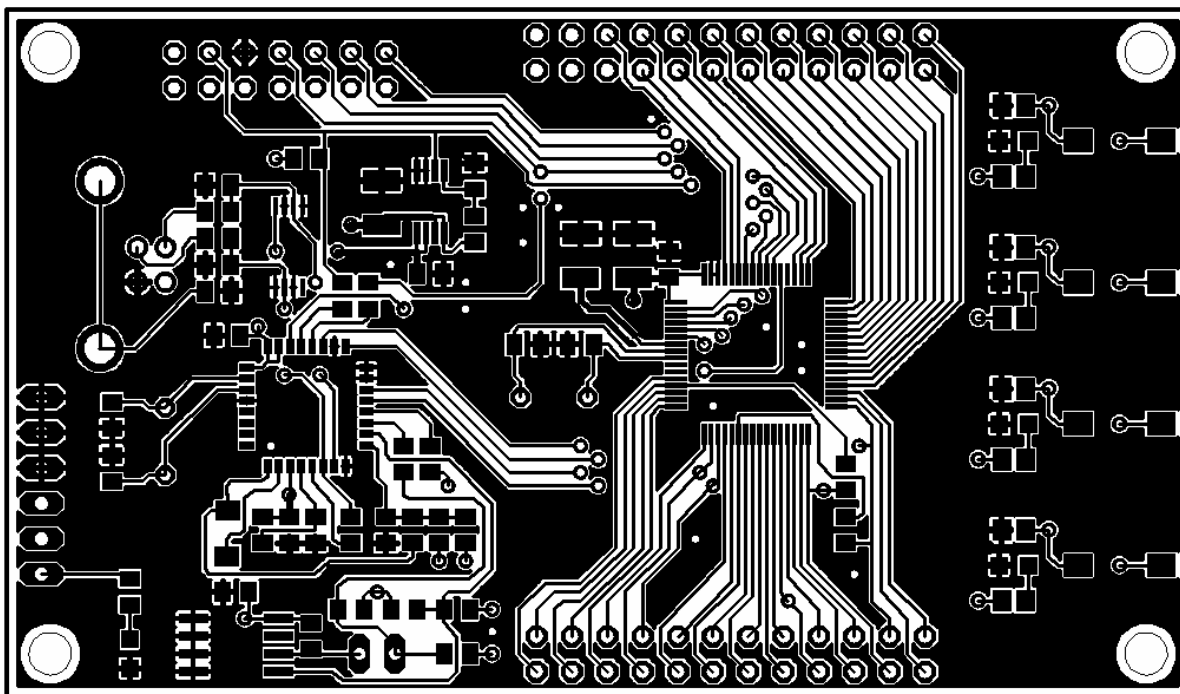
Sheet: 1/2

REV: 1.0

A.2 Board Layout
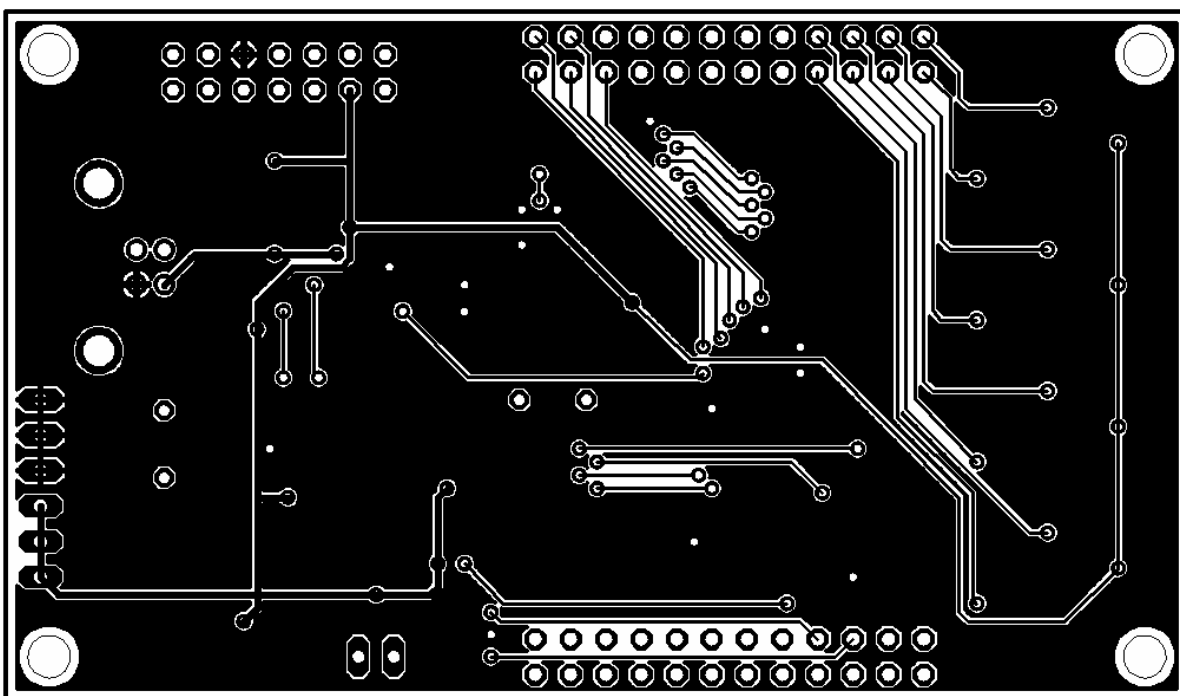

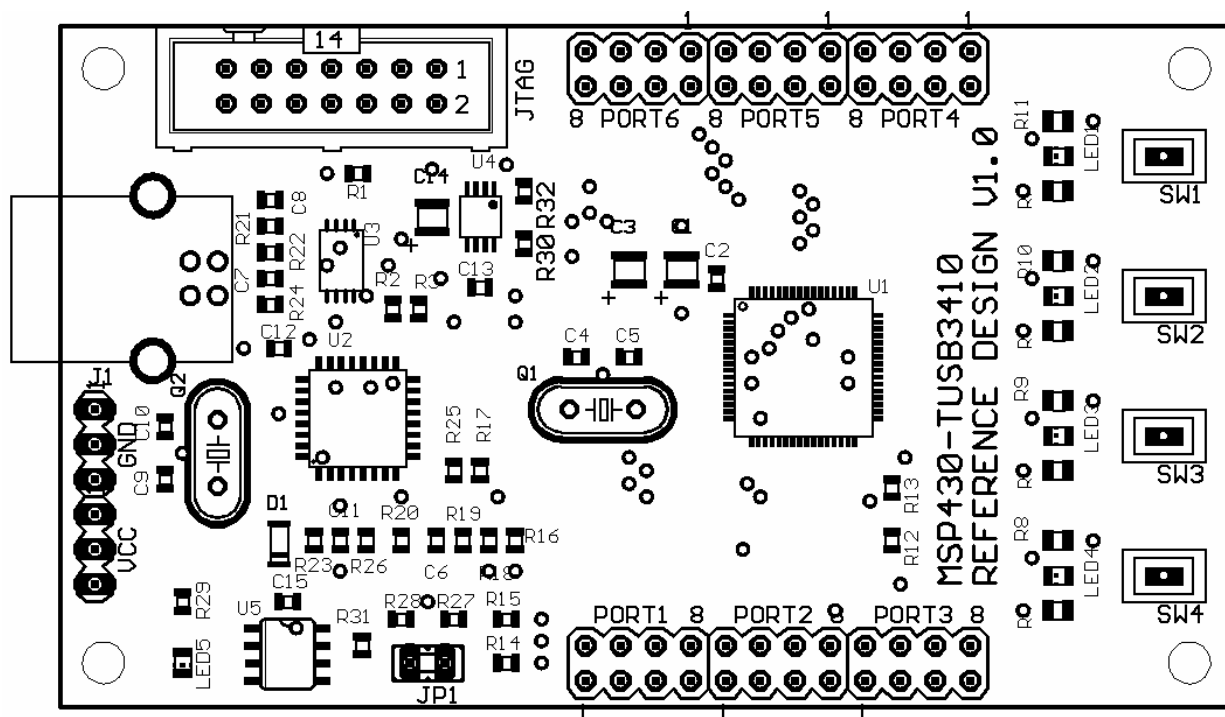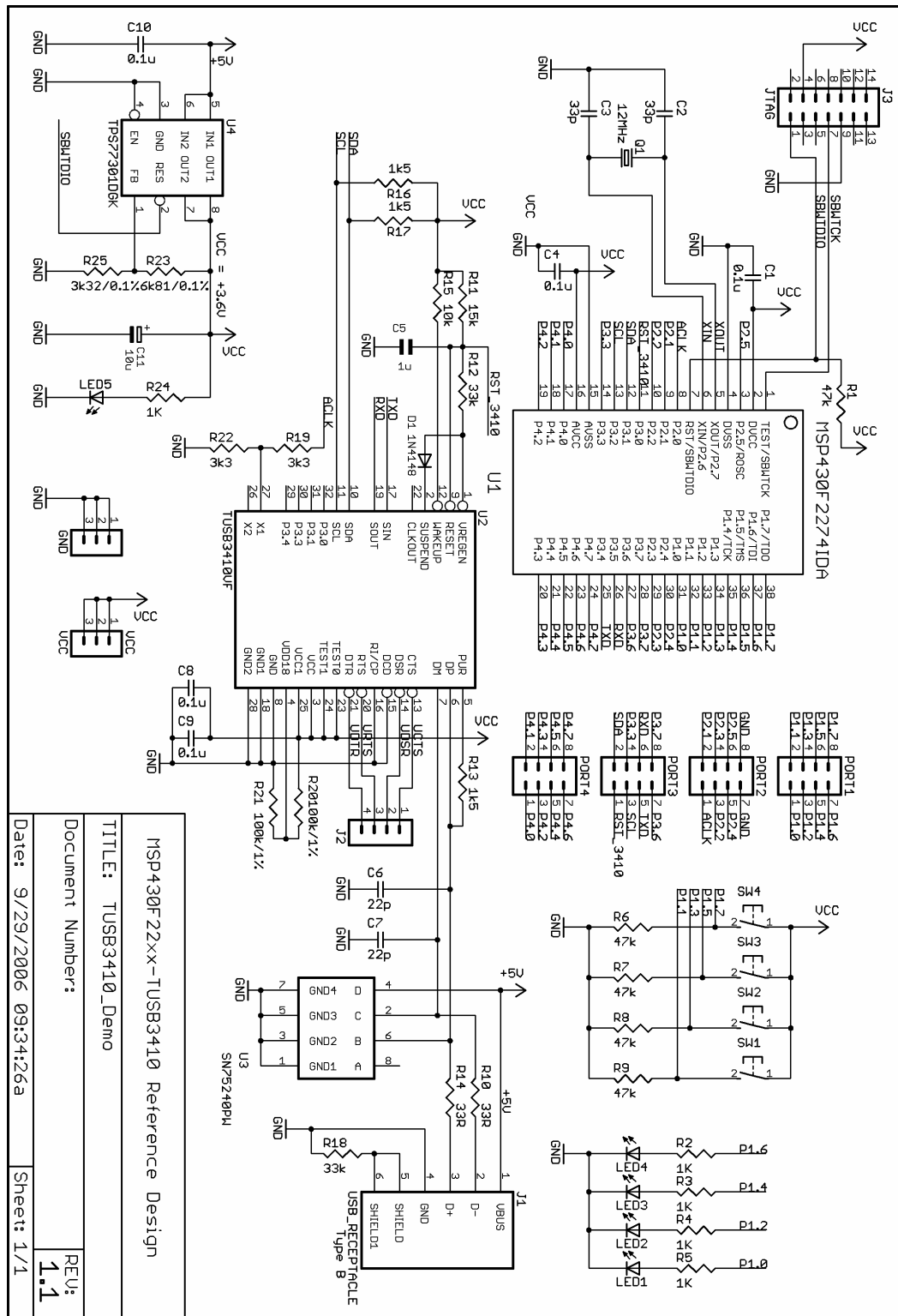**Figure 10. PCB Layout Component Side**

Figure 11. PCB Layout Solder Side

Figure 12. PCB Component Placement

A.3 Bill of Material

Qty.	Value	Device	Parts	Digi-Key #
4		EVQPPDASTD	SW1, SW2, SW3, SW4	P8087SCT-ND
1		JP1E	JP1	A26529-01-ND
1		JUMPER	J2	S9000-ND
1	RED	LED0805	LED5	160-1176-1-ND
4	GREEN	LED0805	LED1, LED2, LED3, LED4	160-1414-1-ND
2		MA03-1	GND, VCC	WM6503-ND
6		MA04-2	PORT1, PORT2, PORT3, PORT4, PORT5, PORT6	A34268-04-ND
1		ML14	JTAG	A26269-ND
5	0.1u	CSMD0805	C2, C11, C12, C13, C15	311-1142-1-ND
5	0R	R_0805	R12, R13, R14, R15, R16	311-0.0ACT-ND
7	1K	RES0805	R2, R3, R4, R5, R6, R7, R29	P1.0KACT-ND
1	1N4148	D	D1	1N4148WTPMSCT-ND
3	1k5	R_0805	R20, R27, R28	P1.5KACT-ND
1	1u	CSMD0805	C6	PCC1807CT-ND
1	3k32/0.1%	R_0805	R32	RR12P3.32KBCT-ND
1	6k81/0.1%	R_0805	R30	RR12P6.81KBCT-ND
1	8MHz	HC49/US	Q1	X165-ND
1	10k	R_0805	R17	P10KACT-ND
3	10u	CSMD1210	C1, C3, C14	399-1563-1-ND
1	12MHz	HC49/US	Q2	X172-ND
1	15k	R_0805	R18	P15KACT-ND
2	22p	CSMD0805	C7, C8	311-1103-1-ND
1	24LC128I/SN	M24XXXSO8	U5	24LC128-I/SN-ND
2	33R	R_0805	R21, R22	311-33ARCT-ND
2	33k	R_0805	R19, R24	P33KACT-ND
4	33p	CSMD0805	C4, C5, C9, C10	311-1105-1-ND
6	47k	R_0805	R1, R8, R9, R10, R11, R31	P47KACT-ND
2	100k/1%	R_0805	R23, R26	311-100KCRCT-ND
1	470R	R_0805	R25	311-470ARCT-ND
1	MSP430F1612IPM	F1XXQFP64PM	U1	296-17885-ND
1	SN75240PW	SN75240	U3	296-6596-1-ND
1	TPS77301DGK	TPS773XX	U4	296-8108-5-ND
1	TUSB3410VF	TUSB3410	U2	296-12699-ND
1	USB_RECPT	USB_RECPT	J1	ED90064-ND

Appendix B. F22xx Solution Hardware Description
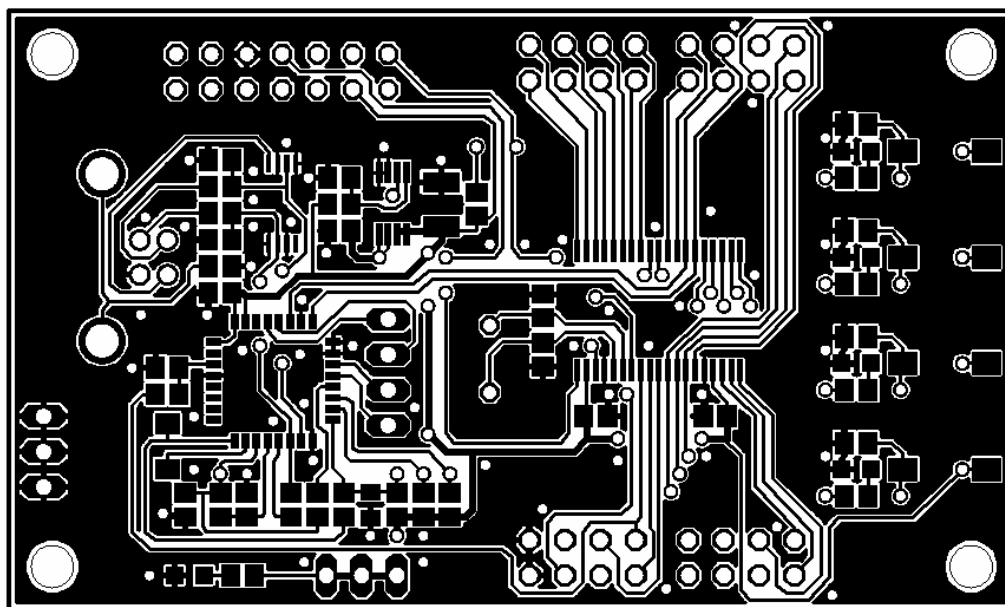
B.1 Schematics

B.2 Board Layout
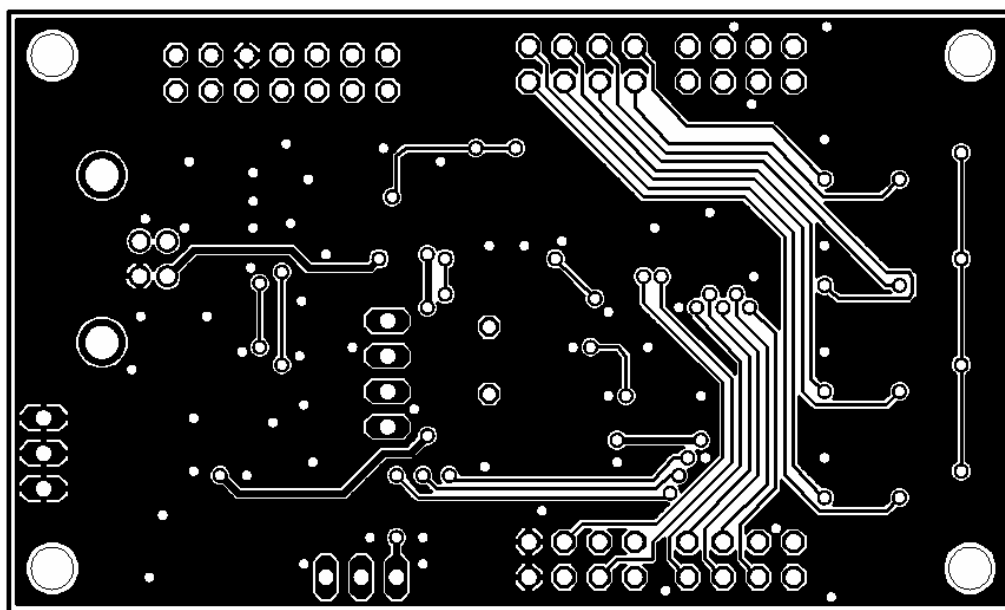

Figure 13.   PCB Layout Component Side
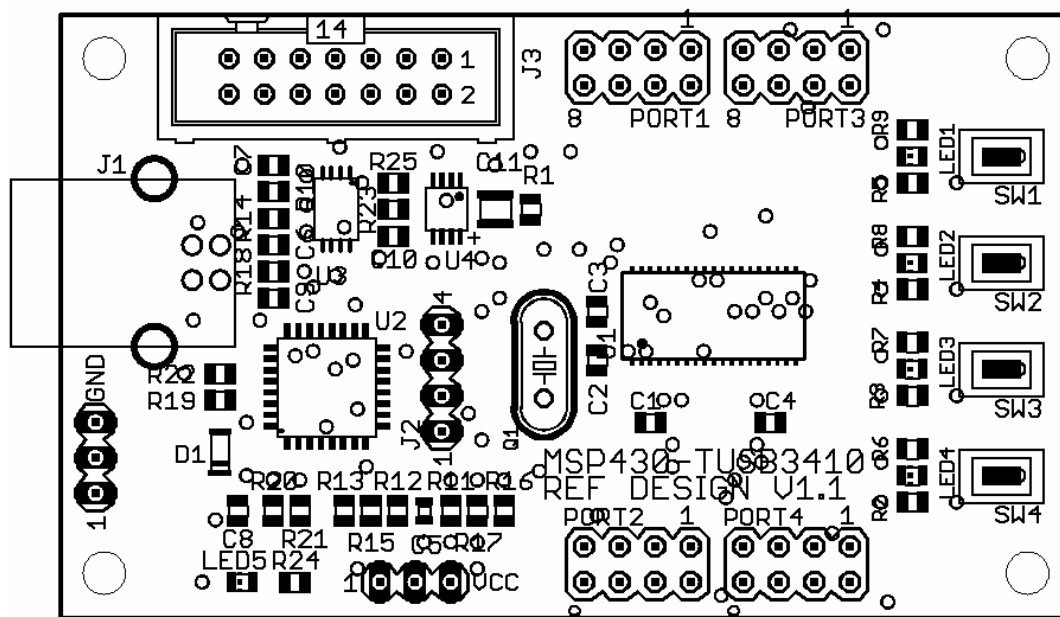

Figure 14.   PCB Layout Solder Side

Figure 15.   PCB Component Placement

TEXAS
INSTRUMENTS

B.3 Bill of Material

Qty.	Value	Device	Parts	Digi-Key #
4		EVQPPDASTD	SW1, SW2, SW3, SW4	P8087SCT-ND
5		LED0805	LED1, LED2, LED3, LED4, LED5	160-1414-1-ND
2		MA03-1	GND, VCC	WM6503-ND
1		MA04-1	J2	WM6502-ND
4		MA04-2	PORT1, PORT2, PORT3, PORT4	A34268-04-ND
5	0.1u	CAP-NP0805	C1, C4, C8, C9, C10	311-1142-1-ND
5	1K	RES0805	R2, R3, R4, R5, R24	P1.0KACT-ND
1	1N4148	D	D1	1N4148WTPMSCT-ND
3	1k5	RES0805	R13, R16, R17	P1.5KACT-ND
1	1u	CSMD0805	C5	PCC1807CT-ND
2	3k3	RES0805	R19, R22	P3.3KACT-ND
1	3k32/0.1%	RES0805	R25	RR12P3.32KBCT-ND
1	6k81/0.1%	RES0805	R23	RR12P6.81KBCT-ND
1	10k	RES0805	R15	P10KACT-ND
1	10u	ELKO_1210	C11	399-3684-1-ND
1	12MHz	HC49/S	Q1	X172-ND
1	15k	RES0805	R11	P15KACT-ND
2	22p	CAP-NP0805	C6, C7	311-1103-1-ND
2	33k	RES0805	R12, R18	P33KACT-ND
2	33R	RES0805	R10, R14	311-33ACT-ND
2	33p	CAP-NP0805	C2, C3	311-1105-1-ND
5	47k	RES0805	R1, R6, R7, R8, R9	P47KACT-ND
2	100k/1%	RES0805	R20, R21	311-100KCRCT-ND
1	JTAG	ML14	J3	A31135-ND
1	MSP430F2274IDA	MSP430F22XXDA	U1	
1	SN75240PW	SN75240	U3	296-6596-1-ND
1	TPS77301DGK	TPS773XX	U4	296-8108-5-ND
1	TUSB3410VF	TUSB3410	U2	296-12699-ND
1	USB_RECEP	USB_RECEPT	J1	ED90064-ND

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information 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.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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