MSP-FET430 Flash Emulation Tool (FET) (For Use With Code Composer Essentials for MSP430 Version 2.0)

User's Guide

Literature Number: SLAU157E
May 2005–Revised January 2008

TEXAS INSTRUMENTS

Contents

List of Figures

List of Tables

Read This First

About This Manual

This manual documents the Texas Instruments MSP-FET430 Flash Emulation Tool (FET). The FET is the program development tool for the MSP430 ultralow-power microcontroller. Both available interfaces, the Parallel-Port interface and the USB interface, are described.

How to Use This Manual

Read and follow the Get Started Now! chapter. This chapter lists the contents of the FET, provides instruction on installing the software and hardware, and describes how to run the demonstration programs. After running the demonstration programs and seeing how quick and easy it is to use the FET, TI recommends that all users read this complete manual.

This manual describes the setup and operation of the FET but does not fully describe the MSP430 or the development software systems. For details of these items, see the appropriate TI documents listed in Section 1.10, *Important MSP430 Documents on the CD-ROM and Web*.

This manual is applicable to the following tools (and devices):

MSP-FET430P120 (for the MSP430F12xIDW and MSP430F12x2IDW devices)

MSP-FET430P140 (for the MSP430F13xIPM, MSP430F14xIPM, MSP430F15xIPM, MSP430F16xIPM, and MSP430F161xIPM devices)

MSP-FET430P410 (for the MSP430F41xIPM devices)

MSP-FET430P430 (for the MSP430F43xIPN devices)

MSP-FET430P440 (for the MSP430F43xIPZ and MSP430F44xIPZ devices)

MSP-FET430UIF (debug interface with USB connection, for all MSP430 Flash based devices)

The following tools contain the USB debug interface and the respective target socket module:

MSP-FET430U14 (for MSP430 devices in 14-pin PW packages)

MSP-FET430U28 (for MSP430 devices in 20- and 28-pin DW or PW packages)

MSP-FET430U38 (for MSP430 devices in 38-pin DA packages)

MSP-FET430U23x0 (for MSP430F2330/F2350/F2370 devices in 40-pin RHA packages only)

MSP-FET430U48 (for MSP430 devices in 48-pin DL packages)

MSP-FET430U64 (for MSP430 devices in 64-pin PM packages)

MSP-FET430U80 (for MSP430 devices in 80-pin PN packages)

MSP-FET430U100 (for MSP430 devices in 100-pin PZ packages)

This tool contains the most up-to-date materials available at the time of packaging. For the latest materials (data sheets, user's guides, software, applications, etc.), visit the TI MSP430 web site at www.ti.com/msp430 or contact your local TI sales office.

Information About Cautions and Warnings

This document 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

MSP430xxxx Device data sheets

MSP430x1xx Family User's Guide, TI literature number SLAU049

MSP430x2xx Family User's Guide, TI literature number SLAU144

MSP430x3xx Family User's Guide, TI literature number SLAU012

MSP430x4xx Family User's Guide, TI literature number SLAU056

If You Need Assistance

Support for the MSP430 device and the FET is provided by the Texas Instruments Product Information Center (PIC). Contact information for the PIC can be found on the TI web site at www.ti.com. Additional device-specific information can be found on the MSP430 web site at www.ti.com/msp430.

FCC Warning

This equipment is intended for use in a laboratory test environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio-frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case, the user is required to take whatever measures may be required to correct this interference at his own expense.

Get Started Now!

This chapter lists the contents of the FET, provides instruction on installing the software and hardware, and shows how to run the demonstration programs.

Topic **Page**

1.1 Kit Contents, MSP-FET430Pxx0 ('P120, 'P140, 'P410, 'P430, 'P440)

- One READ ME FIRST document
- One MSP430 CD-ROM
- One MSP-FET430PIF FET interface module. This is the unit that has a 25-pin male D-Sub connector on one end of the case and a 2×7-pin male connector on the other end of the case.
- One target socket module

  MSP-FET430P120: One MSP-TS430DW28 target socket module. This is the PCB on which is mounted a 28-pin ZIF socket for the MSP430F12xIDW, MSP43012x2IDW, MSP43011x2IDW, or MSP43021x2IDW device. A 2×7-pin male connector is also present on the PCB.

  MSP-FET430P140: One MSP-TS430PM64 target socket module. This is the PCB on which is mounted a 64-pin clam-shell-style socket for the MSP430F13xIPM, MSP430F14xIPM, MSP430F15xIPM, MSP430F16xIPM, or MSP430F161xIPM device. A 2×7-pin male connector is also present on the PCB.

  MSP-FET430P410: One MSP-TS430PM64 target socket module. This is the PCB on which is mounted a 64-pin clam-shell-style socket for the MSP430F41xIPM, MSP430FE42xIPM, or MSP430FW42xIPM device. A 2×7-pin male connector is also present on the PCB.

  MSP-FET430P430: One MSP-TS430PN80 target socket module. This is the PCB on which is mounted an 80-pin ZIF socket for the MSP430F43xIPN device. A 2×7-pin male connector is also present on the PCB.

  MSP-FET430P440: One MSP-TS430PZ100 target socket module. This is the PCB on which is mounted a 100-pin ZIF socket for the MSP430F43xIPZ or MSP430F44xIPZ device. A 2×7-pin male connector is also present on the PCB.
- One 25-conductor cable
- One 14-conductor cable
- Four or eight PCB headers

  MSP-FET430P120: Four PCB 1×14-pin headers (two male and two female)

  MSP-FET430P140: Eight PCB 1×16-pin headers (four male and four female)

  MSP-FET430P410: Eight PCB 1×16-pin headers (four male and four female)

  MSP-FET430P430: Eight PCB 1×20-pin headers (four male and four female)

  MSP-FET430P440: Eight PCB 1×25-pin headers (four male and four female)
- One small box containing two or four MSP430 device samples.

  MSP-FET430P120: MSP430F123IDW and/or MSP430F1232IDW

  MSP-FET430P140: MSP430F149IPM and/or MSP430F169IPM

  MSP-FET430P410: MSP430F413IPM

  MSP-FET430P430: MSP430F437IPN and/or MSP430FG439IPN

  MSP-FET430P440: MSP430F449IPZ

  Consult the device data sheets for device specifications. A list of device errata can be found at www.ti.com/sc/cgi-bin/buglist.cgi or in the device-specific web product folder.

1.2 Kit Contents, MSP-FET430UIF

- One READ ME FIRST document
- One MSP430 CD-ROM
- One MSP-FET430UIF interface module
- One USB cable
- One 14-conductor cable

1.3 Kit Contents, MSP-FET430Uxx ('U14, 'U28, 'U38, 'U23x0, 'U48, 'U64, 'U80, 'U100)

- One READ ME FIRST document
- One MSP430 CD-ROM
- One MSP-FETP430UIF USB interface module. This is the unit that has a USB B-connector on one end of the case and a 2×7-pin male connector on the other end of the case.
- One target socket module

  MSP-FET430U14: One MSP-TS430PW14 target socket module. This is the PCB on which is mounted a 14-pin ZIF socket. It fits all MSP430 devices in 14-pin PW packages. A 2×7-pin male connector is also present on the PCB.

  MSP-FET430U28: One MSP-TS430DW28 or MSP-TS430PW28 target socket module. This is the PCB on which is mounted a 28-pin ZIF socket. It fits all MSP430 devices in 20- and 28-pin DW or PW packages. A 2×7-pin male connector is also present on the PCB.

  MSP-FET430U38: One MSP-TS430DA38 target socket module. This is the PCB on which is mounted a 38-pin ZIF socket. It fits all MSP430 devices in 38-pin DA packages. A 2×7-pin male connector is also present on the PCB.

  MSP-FET430U23x0: One MSP-TS430QFN23x0 target socket module (former name MSP-TS430QFN40). This is the PCB on which is mounted a 40-pin ZIF socket. It fits only MSP430F2330/F2350/F2370 devices in 40-pin RHA packages. A 2×7-pin male connector is also present on the PCB.

  MSP-FET430U48: One MSP-TS430DL48 target socket module. This is the PCB on which is mounted a 48-pin ZIF socket. It fits all MSP430 devices in 48-pin DL packages. A 2×7-pin male connector is also present on the PCB.

  MSP-FET430U64: One MSP-TS430PM64 target socket module. This is the PCB on which is mounted a 64-pin ZIF socket. It fits all MSP430 devices in 64-pin PM packages. A 2×7-pin male connector is also present on the PCB.

  MSP-FET430U80: One MSP-TS430PN80 target socket module. This is the PCB on which is mounted a 80-pin ZIF socket. It fits all MSP430 devices in 80-pin PN packages. A 2×7-pin male connector is also present on the PCB.

  MSP-FET430U100: One MSP-TS430PZ100 target socket module. This is the PCB on which is mounted a 100-pin ZIF socket. It fits all MSP430 devices in 100-pin PZ packages. A 2×7-pin male connector is also present on the PCB.
- One USB cable
- One 14-conductor cable
- Four or eight PCB headers

  MSP-FET430U14: Four PCB 1×7-pin headers (two male and two female)

  MSP-FET430U28: Four PCB 1×14-pin headers (two male and two female)

  MSP-FET430U38: Four PCB 1×19-pin headers (two male and two female)

  MSP-FET430U23x0: Eight PCB 1×10-pin headers (four male and four female)

  MSP-FET430U48: Four PCB 2×24-pin headers (two male and two female)

  MSP-FET430U64: Eight PCB 1×16-pin headers (four male and four female)

  MSP-FET430U80: Eight PCB 1×20-pin headers (four male and four female)

  MSP-FET430U100: Eight PCB 1×25-pin headers (four male and four female)
- One small box containing two or four MSP430 device samples

  MSP-FET430U14: MSP430F2013IPW

  MSP-FET430U28: MSP430F123IDW and/or MSP430F1232IDW or MSP430F2132IPW

  MSP-FET430U38: MSP430F2274IDA

  MSP-FET430U23x0: MSP430F2370IRHA

  MSP-FET430U48: MSP430F4270IDL

  MSP-FET430U64: MSP430F417IPM and MSP430F169IPM

  MSP-FET430U80: MSP430FG439IPN

  MSP-FET430U100: MSP430F449IPZ

Consult the device data sheets for device specifications. Device errata can be found in the respective device product folder on the web, provided as a PDF document. Depending on the device, errata also may be found in the device bug database at www.ti.com/sc/cgi-bin/buglist.cgi.

1.4 Software Installation

To install Code Composer Essentials 2.0 for MSP430 (CCE), run CCE_V2_0_XXX.exe from the CD-ROM and follow the instructions shown on the screen. The hardware drivers for the parallel-port FET (MSP-FET430PIF) and the drivers for the USB JTAG emulator (MSP-FET430UIF) are installed automatically when installing CCE.

Code Composer Essentials 2.0 for MSP430, as well as the different TI JTAG emulators, are compatible with Windows® 2000, Windows XP, and Widows Vista.

NOTE: In Windows Vista, the CCE should be started in XP Compatibility Mode.

1.5 Hardware Installation, MSP-FET430Pxx0, MSP-FET430PIF

1. Use the 25-conductor cable to connect the FET interface module to the parallel port of the PC. The necessary driver for accessing the PC parallel port is installed automatically during CCE installation. Note that a restart is required after the CCE installation for the driver to become active.
2. Use the 14-conductor cable to connect the parallel-port debug interface module to a target board, such as an MSP-TS430xxx target socket module. Module schematics and PCBs are shown in Appendix B.

1.6 Hardware Installation, MSP-FET430UIF

1. Use the USB cable to connect the USB-FET interface module to a USB port on the PC. The USB FET should be recognized, as the USB device driver should have been installed already with the CCE software. **If for any reason the Install Wizard starts, respond to the prompts and point the wizard to the driver files, which are located in this directory: <Installation Root>\Texas Instruments\CC Essentials 2.0\win32_drivers\MSP430_USB\program files\Texas Instruments\TI3410XP\. Detailed driver installation instructions are found in Appendix F.**
2. After connecting to a PC, the USB FET performs a selftest during which the red LED flashes for about two seconds. If the selftest passes successfully, the green LED lights permanently.
3. Use the 14-conductor cable to connect the USB-FET interface module to a target board, such as an MSP-TS430xxx target socket module.
4. Ensure that the MSP430 device is securely seated in the socket, and that its pin 1 (indicated with a circular indentation on the top surface) aligns with the "1" mark on the PCB.
5. Compared to the parallel-port debug interface, the USB FET has additional features including JTAG security fuse blow and adjustable target V_{CC} (1.8 V to 3.6 V). Supply the module with up to 100 mA.

1.7 Hardware Installation, MSP-eZ430-F2013, MSP-eZ430-RF2500

1. Connect the tool to a USB port of your PC.
2. The USB FET should be recognized instantly, as the USB device driver should have been installed already 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 Appendix F.

1.8 Hardware Installation, MSP-FET430Uxx ('U14, 'U28, 'U38, 'U23x0, 'U48, 'U64, 'U80, 'U100) and MSP-FET430Pxx0 ('P120, 'P140, 'P410, 'P430, 'P440)

1. Connect the MSP-FET430PIF or MSP-FET430UIF debug interface to the appropriate port of the PC. Use the 14-conductor cable to connect the FET interface module to the supplied target socket module.
2. Ensure that the MSP430 device is securely seated in the socket and that its pin 1 (indicated with a circular indentation on the top surface) aligns with the "1" mark on the PCB.
3. Ensure that the two jumpers (LED and V_{CC}) near the 2×7-pin male connector are in place. Illustrations of the target socket modules and their parts are found in Appendix B.

1.9 Flashing the LED

This section demonstrates on the FET the equivalent of the C-language "Hello world!" introductory program. An application that flashes the LED is developed, downloaded to the FET, and run.

1. Start Code Composer Essentials (Start → All Programs → Texas Instruments → Code Composer Essentials 2.0 → Code Composer Essentials 2.0).
2. Open the Code Composer Essentials help by selecting the menu item Help → Open Code Composer Essentials Help.
3. Select "Quick Start Guide" from the help main page. Then select "Creating Managed Make C projects". Follow the steps to create a new project, and make sure that a linker command file is included in the project that matches the actual MSP430 device being used.
4. Add the flashing LED code example to the project. All code examples are located in the CCE installation directory under "Examples". Use the following table to select the appropriate source code file:

MSP430 Devices	Code Example
MSP430F11x1, MSP430F11x2, MSP430F21x1	<…>\fet110\ fet110_1.c
MSP430F12x, MSP430F12x2, MSP430F22x4	<…>\fet120\ fet120_1.c
MSP430F13x, MSP430F14x, MSP430F14x1, MSP430F15x, MSP430F16x, MSP430F16xx, MS430F23x0	<…>\fet140\ fet140_1.c
MSP430F41x, MSP430F42x, MSP430F42x, MSP430FW42x, MSP430FE42x, MSP430F42x0	<…>\fet410\ fet410_1.c
MSP430F43x, MSP430FG43x	<…>\fet430\ fet430_1.c
MSP430F44x, MSP430FG461x	<…>\fet440\ fet440_1.c

5. Make sure the correct JTAG debug interface is chosen by selecting the menu item Project → Project Properties, and then click on "Debug Properties". Select either LPTx for using the parallel port JTAG emulator (MSP-FET430PIF) or USB for the MSP-FET430UIF and for the eZ430 interfaces.
6. Now select "Debugging Managed Make C Projects" on the help page and follow the steps on how to start the debug session. The project is built automatically when starting the debug session. Also, the Flash memory is erased, and the newly built object file is downloaded to the device.

See FAQ Debugging #1 if the CCE debugger (GDB430) is unable to communicate with the device.

Congratulations, you've just built and tested your first MSP430 application!

1.10 Important MSP430 Documents on the CD-ROM and Web

The primary sources of MSP430 information are the device-specific data sheets and user's guides. The most up-to-date versions of these documents available at the time of production have been provided on the CD-ROM included with this tool. The MSP430 web site (www.ti.com/msp430) contains the latest version of these documents.

Development Flow

This chapter discusses how to use Code Composer Essentials (CCE) to develop application software and how to debug that software.

Topic **Page**

2.1 Using Code Composer Essentials (CCE)

The following sections are a brief overview of how to use CCE. For a full discussion of software development flow with CCE in assembly or C, see *MSP430 Assembly Language Tools User's Guide* (TI literature number SLAU131) and *MSP430 Optimizing C/C++ Compiler User's Guide* (TI literature number SLAU132).

2.1.1 Creating a Project from Scratch

This section presents step-by-step instructions to create an assembly or C project from scratch and to download and run the application on the MSP430 (see Section 2.1.2, *Project Settings*). Also, the MSP430 Code Composer Essentials Help presents a more comprehensive overview of the process.

1. Start the CCE (Start → All Programs → Texas Instruments → Code Composer Essentials 2.0 → Code Composer Essentials 2.0).
2. Create new project by clicking File → New → Project → C → Managed Make C/ASM Project (Recommended). Click Next and enter the name for the project. Click Next until reaching the Device Selection page. Select the appropriate device and click Finish.
3. Create a new text file by clicking File → New → Source File. Remember to enter the suffix .c or .asm.
4. Enter the program text into the file.

Note: Use .h files to simplify code development

CCE is supplied with files for each device that define the device registers and the bit names, and these files can greatly simplify the task of developing a program. The files are located in <Installation Root>\tools\compiler\msp430\include\. Include the .h file corresponding to the target device in the text file (#include "msp430xyyy.h").

---

5. Configure the project options (Project → Properties), or accept the default factory settings.
6. Build the project (Project → Build Active Project).
7. Debug the application (Run → Debug Active Project). This starts GDB430, and GDB430 gains control of the target, erases the target memory, programs the target memory with the application, and resets the target.
   See FAQ Debugging #1 if GDB430 is unable to communicate with the device.
8. Click Run → Run to start the application.
9. Click Run → Terminate to stop the application, to exit GDB430. To return to code editor use Window → Open Perspective → C/C++.
10. Click File → Exit to exit the CCE.

2.1.2 Project Settings

The settings required to configure the CCE are numerous and detailed. Most of project can be compiled and debugged with default factory settings. The project settings are accessed using: PROJECT → PROPERTIES for the active project. The following project settings are recommended/required:

- Specify the target device for debug session (Project → Properties → Debug Properties → Device)
- To most easily debug a C project, disable optimization (Project → Properties → C/C++ Build → MSP430 Compiler → Optimization).
- Specify the search path for the C preprocessor (Project → Properties → C/C++ Build → MSP430 Compiler → General Options).
- Specify the search path for any used libraries (Project → Properties → C/C++ Build → MSP430 Executable Linker).
- Specify the debugger interface. Select Project → Properties → Debug Properties → Connection → LPTx for the parallel FET interface or USB Emulator for the USB interface.
- Enable the erasure of the Main and Information memories before object code download (Project → Properties → Debug Properties → Erase Main and Information Memory)
- To maximize system performance during debug, disable Virtual Breakpoints (Project → Properties → Debug Properties → Use Virtual Breakpoints)

2.1.3 Using an Existing CCE V1.x Project

CCE 2.0 does not fully support conversion of workspaces and projects created in version 1.0 to the version 2.0 format. For projects, the files contained in the project can be viewed, but there may be issues building or debugging the project.

To use a CCE 1.0 project in CCE 2.0, the project must be converted. Create a new project and add the original project files to the new project. After the project is created, the compile, link, and debug options must also be updated to match the v1.0 options.

2.1.4 Stack Management

The reserved stack size can be configured through the project options dialog (Project → Properties → C/C++ Build → MSP430 Executable Linker → General Options → Set C System Stack Size). Stack size is defined to extend from the last location of RAM for 50 to 80 bytes (i.e., the stack extends downwards through RAM for 50 to 80 bytes, depending on the RAM size of the selected device).

Note that the stack can overflow due to small size or application errors. See FAQ Debugging #20 for a method of tracking the stack size.

2.1.5 How to Generate Texas Instruments .TXT (and other binary-format) Files

The CCE installation includes the conversion tool hex430.exe. This tool can be configured to generate output objects in TI-TXT format for use with the GANG430 and PRGS430 programmers. This tool is in <Installation Root>\tools\compiler\msp430\bin. The post-build step can be configured to generate the TXT file automatically after every build (Project → Properties → C/C++ Build/Build Steps/Post-Build Step). Intel and Motorola hex formats can also be selected.

2.1.6 Overview of Example Programs

Example programs for MSP430 devices are provided in <Installation Root>\examples. Each tool folder contains folders that contain the assembly and C sources.

To use the examples, create a new project and add the example source file to the project. (Project → Add Files to Active Project).

See the CCE help, Examples Overview for more information.

2.2 Using Integrated Debugger GDB430

See Appendix E for a description of FET-specific menus within CCE.

2.2.1 Breakpoint Types

The debugger breakpoint mechanism uses a limited number of on-chip debugging resources (specifically, N breakpoint registers, see Table 2-1). When N or fewer breakpoints are set, the application runs at full device speed (or "Realtime"). When greater than N breakpoints are set and Use Virtual Breakpoints is enabled (Project → Properties → Debug Properties → Use Virtual Breakpoints), the application runs under the control of the host PC; the system operates at a much slower speed, but offers unlimited software breakpoint (or "Non-Realtime"). During Non-Realtime mode, the PC effectively repeatedly single steps the device and interrogates the device after each operation to determine if a breakpoint has been hit.

Both (code) address and data (value) breakpoints are supported. Data breakpoints and range breakpoints each require two MSP430 hardware breakpoints.

Table 2-1. Number of Device Breakpoints and Other Emulation Features

Device	4-Wire JTAG	2-Wire JTAG[1]	Breakpoints (N)	Range Breakpoints	Clock Control	State Sequencer	Trace Buffer
MSP430F11x1	X		2				
MSP430F11x2	X		2				
MSP430F12x	X		2				
MSP430F12x2	X		2				
MSP430F13x	X		3	X			
MSP430F14x	X		3	X			
MSP430F15x	X		8	X	X	X	X
MSP430F16x	X		8	X	X	X	X
MSP430F161x	X		8	X	X	X	X
MSP430F20xx	X	X	2		X		
MSP430F21x1	X		2		X		
MSP430F22x2	X	X	2		X		
MSP430F21x2	X	X	2		X		
MSP430F22x4	X	X	2		X		
MSP430F23x	X		3	X	X		
MSP430F23x0	X		2		X		
MSP430F24x	X		3	X	X		
MSP430F241x	X		8	X	X	X	X
MSP430F2410	X		3	X	X		
MSP430F261x	X		8	X	X	X	X
MSP430F41x	X		2		X		
MSP430F42x	X		2		X		
MSP430FE42x	X		2		X		
MSP430FW42x	X		2		X		
MSP430F42x0	X		2		X		
MSP430FG42x0	X		2		X		
MSP430F43x	X		8	X	X	X	X
MSP430FG43x	X		2		X		
MSP430F43x1	X		2		X		
MSP430F44x	X		8	X	X	X	X
MSP430FG461x	X		8	X	X	X	X

(1) The 2-wire JTAG debug interface is also referred to as Spy-Bi-Wire interface. Note that this interface is only supported by the MSP-FET430UIF USB JTAG emulator and the MSP-GANG430 production programming tool. The MSP-FET430PIF parallel port JTAG emulator does not support communication in 2-wire JTAG mode.

Table 2-1. Number of Device Breakpoints and Other Emulation Features (continued)

Device	4-Wire JTAG	2-Wire JTAG[1]	Breakpoints (N)	Range Breakpoints	Clock Control	State Sequencer	Trace Buffer
MSP430F47x3	X		2		X		
MSP430F47x4	X		2		X		

2.2.2 Using Breakpoints

If GDB430 debugger is started with greater than N breakpoints set and virtual breakpoints are disabled, a message is output that informs the user that not all breakpoints can be enabled. Note that the CCE permits any number of breakpoints to be set, regardless of the Use Virtual Breakpoints setting of CCE. If virtual breakpoints are disabled, a maximum of N breakpoints can be set within GDB430.

Resetting a program requires a breakpoint, which set on the address defined in Project → Properties → Debug Properties → Run To.

The Run To Cursor operation temporarily requires a breakpoint.

There are new types of breakpoints in CCE 2.0:

1. **Breakpoint on stack overflow**. It is possible to debug the applications that cause the stack overflow. Set Break on Stack Overflow (right mouse click in the debug window and select in context menu "Break on Stack Overflow". The program execution stops on the instruction that caused stack overflow. The size of stack can be adjusted in Project → Properties → C/C++ Build → MSP430 Executable Linker → General Options.
   Note: This option is not available on all devices (see Table 2-1).
2. **Breakpoint on variable** stops the execution on read/write access to the variable with a defined condition; e.g., halt if var1 > 10. To set the breakpoint, select the variable in editor (Debug perspective), mouse right click and select in the context menu "Variable Storage breakpoint".
   Restriction 1: This breakpoint is applicable to global variables and register variables. In the latter case, to set the breakpoint (BP), halt execution in the function where observation of the variable is desired (set code BP there). Set the breakpoint and delete (or disable) the code breakpoint in the function and run/restart the application.
   Restriction 2: This breakpoint is applicable to variables 8 bits and 16 bits wide.

Design Considerations for In-Circuit Programming

This chapter presents signal requirements for in-circuit programming of the MSP430.

Topic **Page**

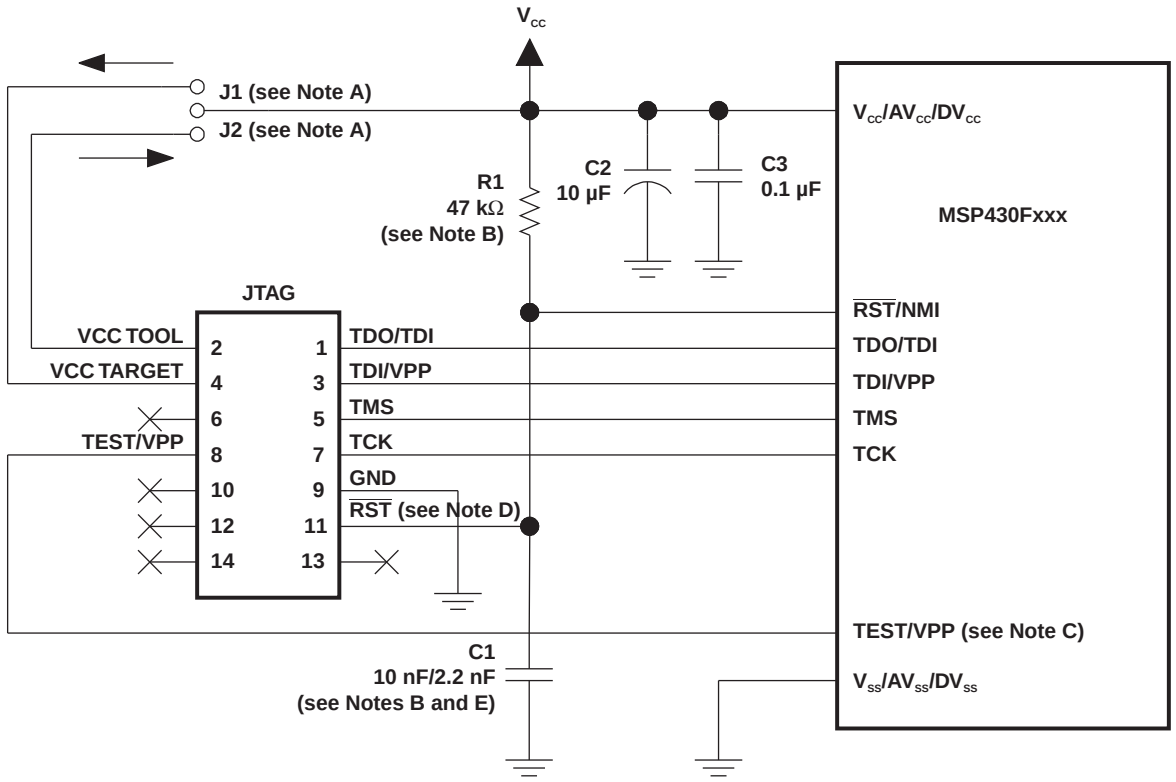
3.1 Signal Connections for In-System Programming and Debugging, MSP-FET430PIF, MSP-FET430UIF, GANG430, PRGS430

With the proper connections, the debugger and an FET hardware JTAG interface (such as the MSP-FET430PIF and MSP-FET430UIF) can be used to program and debug code on the target board. In addition, the connections also support the GANG430 or PRGS430 production programmers, thus providing an easy way to program prototype boards, if desired.

Figure 3-1 shows the connections between the 14-pin FET interface module connector and the target device required to support in-system programming and debugging using GDB430 for 4-wire JTAG communication. Figure 3-2 shows the connections for 2-wire JTAG mode (Spy-Bi-Wire). While 4-wire JTAG mode is generally supported on all MSP430 devices, 2-wire JTAG mode is available on selected devices only. See Table 2-1 for information on which interfacing method can be used on which device.

The connections for the FET interface module and the GANG430 or PRGS430 are identical. Both the FET interface module and GANG430 can supply V_{CC} to the target board (via pin 2). In addition, the FET interface module and GANG430 have a V_{CC}-sense feature that, if used, requires an alternate connection (pin 4 instead of pin 2). The V_{CC}-sense feature senses the local V_{CC} present on the target board (i.e., a battery or other local power supply) and adjusts the output signals accordingly. If the target board is to be powered by a local V_{CC}, then the connection to pin 4 on the JTAG should be made, and not the connection to pin 2. This utilizes the V_{CC}-sense feature and prevents any contention that might occur if the local on-board V_{CC} were connected to the V_{CC} supplied from the FET interface module or the GANG430. If the V_{CC}-sense feature is not necessary (i.e., the target board is to be powered from the FET interface module or the GANG430) the V_{CC} connection is made to pin 2 on the JTAG header and no connection is made to pin 4. Figure 3-1 and Figure 3-2 show a jumper block that supports both scenarios of supplying V_{CC} to the target board. If this flexibility is not required, the desired V_{CC} connections may be hard-wired eliminating the jumper block. Pins 2 and 4 must not be connected simultaneously.

Note that in 4-wire JTAG communication mode (see Figure 3-1), the connection of the target RST signal to the JTAG connector is optional and not required when using 4-wire JTAG communication mode capable-only devices. However, when using 2-wire JTAG communication mode capable devices in 4-wire JTAG mode, the RST connection must be made. The MSP430 development tools and device programmers perform a target reset through issuing a JTAG command to gain control over the device. However, in the case this should be unsuccessful, the RST signal of the JTAG connector may be used by the development tool or device programmer as an additional way to assert a device reset.



A Make either connection J1 in case a local target power supply is used or connection J2 to power target from the debug/programming adapter.

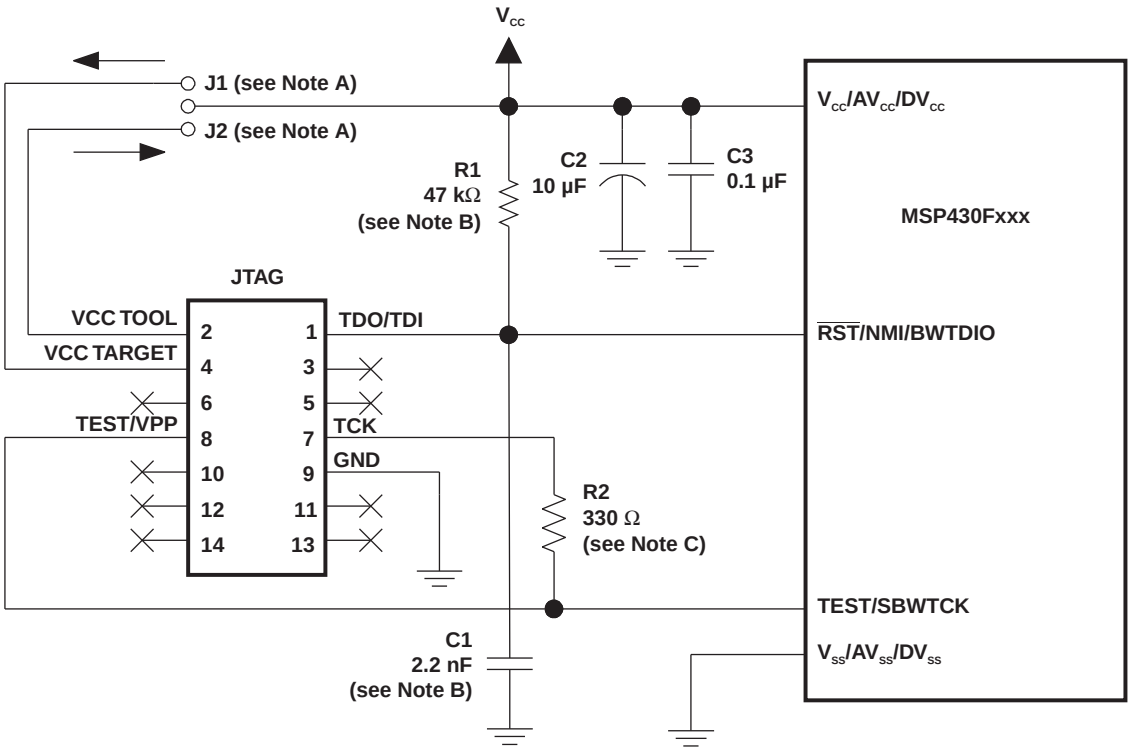
B The $\overline{\text{RST}}$/NMI pin R1/C1 configuration is device family dependent. See the respective MSP430 family user's guide for the recommended configuration.

C The TEST/VPP pin is available only on MSP430 family members with multiplexed JTAG pins. See the device data sheet to see if this pin is available.

D The connection to the JTAG connector RST pin is optional when using 4-wire JTAG communication mode capable-only devices and not required for device programming or debugging. However, this connection is required when using 2-wire JTAG communication mode capable devices in 4-wire JTAG mode.

E When using 2-wire JTAG communication capable devices in 4-wire JTAG mode the upper limit for C1 should not exceed 2.2 nF. This applies to both TI FET interface modules (LPT/USB FET).

Figure 3-1. Signal Connections for 4-Wire JTAG Communication



A Make either connection J1 in case a local target power supply is used or connection J2 to power target from the debug/programming adapter.

B The device $\overline{\text{RST}}$/NMI/SBWTDIO pin is used in 2-wire mode for bidirectional communication with the device during JTAG access and that any capacitance attached to this signal may affect the ability to establish a connection with the device. The upper limit for C1 is 2.2 nF when using current TI FET interface modules (USB FET).

C R2 is used to protect the JTAG debug interface TCK signal against the JTAG security fuse blow voltage that is supplied by the TEST/VPP pin during the fuse blow process. In the case that fuse blow functionality is not needed, R2 is not required (becomes 0 Ω), and the connection TEST/VPP must not be made.

Figure 3-2. Signal Connections for 2-Wire JTAG Communication (Spy-Bi-Wire)

3.2 External Power

The PC parallel port can source a limited amount of current. Because of the ultralow-power requirement of the MSP430, a stand-alone FET does not exceed the available current. However, if additional circuitry is added to the tool, this current limit could be exceeded. In this case, external power can be supplied to the tool via connections provided on the target socket modules. See the schematics and pictorials of the target socket modules presented in Appendix B to locate the external power connectors.

The MSP-FET430UIF can supply targets with up to 100 mA through pin 2 of the 14-pin connector. V_{CC} for the target can be selected between 1.8 V and 5 V in steps of 0.1 V. Alternatively, the target can be supplied externally. In this case, the external voltage should be connected to pin 4 of the 14-pin connector. The MSP-FET430UIF then adjusts the level of the JTAG signals to external V_{CC} automatically. Only pin 2 (MSP-FET430UIF supplies target) or pin 4 (target is externally supplied) must be connected; not both at the same time.

When a target socket module is powered from an external supply, the external supply powers the device on the target socket module and any user circuitry connected to the target socket module, and the FET interface module continues to be powered from the PC via the parallel port. If the externally supplied voltage differs from that of the FET interface module, the target socket module must be modified so that the externally supplied voltage is routed to the FET interface module (so that it may adjust its output voltage levels accordingly). See the target socket module schematics in Appendix B.

3.3 Bootstrap Loader

The JTAG pins provide access to the Flash memory of the MSP430Fxxx devices. On some devices, these pins are shared with the device port pins, and this sharing of pins can complicate a design (or it may simply not be possible to do so). As an alternative to using the JTAG pins, most MSP430Fxxx devices contain a program (a "Bootstrap Loader") that permits the Flash memory to be erased and programmed simply, using a reduced set of signals. Application reports SLAA089 and SLAA096 fully describe this interface. TI does not produce a BSL tool. However, customers can easily develop their own BSL tools using the information in the application reports, or BSL tools can be purchased from third parties. See the MSP430 web site for the application reports and a list of MSP430 third-party tool developers.

TI suggests that MSP430Fxxx customers design their circuits with the BSL in mind (i.e., TI suggests providing access to these signals via, for example, a header).

See FAQ Hardware #10 for a second alternative to sharing the JTAG and port pins.

The BSL tool requires the following device signals:

- $\overline{\text{RST}}$/NMI
- TEST(1)
- TCK(1)
- GND
- VCC
- P1.1
- P2.2 or P1.0(2)

(1) If present on device

(2) '1xx / '2xx devices use pins P1.1 and P2.2 for the BSL. '4xx devices use pins P1.0 and P1.1 for the BSL.

Frequently Asked Questions

This appendix presents solutions to frequently asked questions regarding hardware, program development, and debugging tools.

Topic **Page**

A.1 Hardware

1. **The state of the device** (CPU registers, RAM memory, etc.) **is undefined following a reset.** Exceptions to the above statement are that the PC is loaded with the word at 0xFFFE (i.e., the reset vector), the status register is cleared, and the peripheral registers (SFRs) are initialized as documented in the device family user's guides. The CCE debugger (GDB430) resets the device after programming it.
2. **MSP430F22xx Target Socket Module (MSP-TS430DA38) – Important Information**

   Due to the large capacitive coupling introduced by the device socket between the adjacent signals XIN/P2.6 (socket pin 6) and $\overline{\text{RST}}$/SBWTDIO (socket pin 7), in-system debugging can disturb the LFXT1 low-frequency crystal oscillator operation (ACLK). This behavior only applies to the Spy-Bi-Wire (2-wire) JTAG configuration and only to the period while a debug session is active.

   Workarounds:

   - Use the 4-wire JTAG mode debug configuration instead of the Spy-Bi-Wire (2-wire) JTAG configuration. This can be achieved by placing jumpers JP4 through JP9 accordingly.
   - Use the debugger option "Release JTAG On Go" that can be selected from the IDE dropdown menu. This prevents the debugger from accessing the MSP430 while the application is running. Note that in this mode a manual halt is required to see if a breakpoint was hit. See the IDE documentation for more information on this feature.
   - Use the debugger option "Release JTAG On Go" that can be selected from the IDE dropdown menu. This prevents the debugger from accessing the MSP430 while the application is running. Note that in this mode a manual halt is required to see if a breakpoint was hit. See the IDE documentation for more information on this feature.
   - Use an external clock source to drive XIN directly.
3. The 14-conductor **cable** connecting the FET interface module and the target socket module **must not exceed 8 inches (20 centimeters) in length**.
4. The signal assignment on the **14-conductor cable** is **identical** for the **parallel port interface** and the **USB FET**.
5. **To utilize the on-chip ADC voltage references, C6** (10 µF, 6.3 V, low leakage) **must be installed** on the target socket module.
6. **To utilize the charge pump on the devices with LCD+ Module, C4** (10 µF, low leakage) **must be installed** on the target socket module.
7. **Crystals/resonators Q1 and Q2** (if applicable) **are not provided on the target socket module.** For MSP430 devices that contain user selectable loading capacitors, the effective capacitance is the selected capacitance plus 3 pF (pad capacitance) divided by two.
8. **Crystals/resonators have no effect upon the operation of the tool and the CCE debugger GDB430** (as any required clocking/timing is derived from the internal DCO/FLL).
9. **On 20-pin and 28-pin devices** with multiplexed port/JTAG pins (P1.4 to P1.7), **"Release JTAG on run" (from CCE menu Run) must be selected to use these pins in their port capacity.**
10. **As an alternative to sharing the JTAG and port pins** (on 20 and 28 pin devices), **consider using an MSP430 device that is a "superset" of the smaller device.** A very powerful feature of the MSP430 is that the family members are code and architecturally compatible, so code developed on one device (e.g., without shared JTAG and port pins) port effortlessly to another (assuming an equivalent set of peripherals).
11. **Information memory may not be blank** (erased to 0xff) when the device is delivered from TI. Customers should erase the Information Memory before its first usage. Main Memory of packaged devices is blank when the device is delivered from TI.
12. **The device current increases by approximately 10 µA when a device in low power mode is stopped** (using Halt) **and then the low power mode is restored** (using Run). This behavior appears to happen on all devices except the MSP430F12x.

13. The following **ZIF sockets** are used in the FET tools and target socket modules:
    - 14-pin device (PW package): ENPLAS OTS-14-065-01
    - 28-pin device (PW package): Enplas OTS-28-0.65-01
    - 28-pin device (DW package): Wells-CTI 652 D028
    - 38-pin device (DA package): Yamaichi IC189-0382-037
    - 40-pin device (RHA package): ENPLAS QFN-40B-0.5-01
    - 48-pin device (DL package): Yamaichi IC51-0482-1163
    - 64-pin device (PM package): Yamaichi IC51-0644-807
    - 80-pin device (PN package): Yamaichi IC201-0804-014
    - 100-pin device (PZ package): Yamaichi IC201-1004-008

    Enplas: www.enplas.com

    Wells-CTI: www.wellscti.com/

    Yamaichi: www.yamaichi.us/

A.2 Program Development (Assembler, C-Compiler, Linker, IDE)

Note: Consider the CCE Release Notes

For the case of unexpected behavior, see the CCE Release Notes document for known bugs and limitations of the current CCE version. This information can be accessed through the menu item Help → Welcome, and then selecting Notes.

1. **A common MSP430 "mistake" is to fail to disable the Watchdog mechanism**; the Watchdog is enabled by default, and it resets the device if not disabled or properly managed by the application.
2. **Within the C libraries, GIE (Global Interrupt Enable) is disabled before** (and restored after) **the hardware multiplier is used.**
3. **It is possible to mix assembly and C programs within CCE.** See the "Interfacing C/C++ With Assembly Language" chapter of the *MSP430 Optimizing C/C++ Compiler User's Guide* (TI literature number SLAU132).
4. **Constant definitions (#define) used within the .h files are effectively reserved** and include, for example, C, Z, N, and V. Do not create program variables with these names.
5. **Compiler optimization can remove unused variables and/or statements that have no effect** and can affect debugging. To prevent this, these variables can be declared *volatile*; e.g., *volatile int i;*.

A.3 Debugging

GDB430 is the underlying debugger software of CCE and can be used as a stand-alone application. This section is applicable when using GDB430 both stand-alone and from the CCE IDE.

Note: Consider the CCE release notes

For the case of unexpected behavior, see the CCE Release Notes document for known bugs and limitations of the current CCE version. To access this information, click Help → Welcome and select Notes.

1. **GDB430 reports that it cannot communicate with the device.** Possible solutions to this problem include:

   Ensure that the correct port (LPT1, 2, 3, or USB) is being specified in the Debug Properties. Check the PC BIOS for the parallel port address (0x378, 0x278, 0x3BC), and the parallel port configuration (ECP, Compatible, Bidirectional, or Normal) (see FAQ Debugging #5). For users of IBM Thinkpad computers, try port specifications LPT2 and LPT3, despite the fact that the operating system reports the parallel port is located at LPT1.

   Ensure that no other software application has reserved/taken control of the parallel port (for example, printer drivers, ZIP drive drivers, etc.). Such software can prevent the GDB430 driver from accessing the parallel port, and, hence, communicating with the device.

It may be necessary to reboot the computer to complete the installation of the required parallel port drivers.

Ensure that the MSP430 device is securely seated in the socket (so that the "fingers" of the socket completely engage the pins of the device), and that its pin 1 (indicated with a circular indentation on the top surface) aligns with the "1" mark on the PCB.

> **CAUTION**
>
> **Possible Damage To Device**
>
> Always handle MSP430 devices with using vacuum pick-up tool only; do not use your fingers, as you can easily bend the device pins and render the device useless. Also, always observe and follow proper ESD precautions.

2. **GDB430 can debug applications that utilize interrupts and low power modes.** See FAQ Debugging #17).
3. **GDB430 cannot access the device registers and memory while the device is running.** The user must stop the device to access device registers and memory.
4. **GDB430 reports that the device JTAG security fuse is blown**. With current MSP-FET430PIF and MSP430-FET430UIF JTAG interface tools, there is a weakness when adapting target boards that are powered externally. This leads to an accidental fuse check in the MSP430 and results in the JTAG security fuse being recognized as blown although it is not. This occurs for MSP-FET430PIF and MSP-FET430UIF but is mainly seen on MSP-FET430UIF.

   Workarounds:

   - Connect the device $\overline{\text{RST}}$/NMI pin to JTAG header (pin 11), MSP-FET430PIF/MSP-FET430UIF interface tools are able to pull the $\overline{\text{RST}}$ line, this also resets the device internal fuse logic.
   - Do not connect both V_{CC} Tool (pin 2) and V_{CC} Target (pin 4) of the JTAG header. Specify a value for V_{CC} in the debugger that is equal to the external supply voltage.
5. **The parallel port designators (LPTx) have the following physical addresses: LPT1: 378h, LPT2: 278h, LPT3: 3BCh.** The configuration of the parallel port (ECP, Compatible, Bidirectional, Normal) is not significant; ECP seems to work well. See FAQ Debugging #1 for additional hints on solving communication problems between GDB430 and the device.
6. **GDB430 asserts $\overline{\text{RST}}$/NMI to reset the device** when GDB430 is started and when the device is programmed. The device is also reset by the GDB430 Reset button, and when the device is manually reprogrammed (using Reload), and when the JTAG is resynchronized (using Resynchronize JTAG). When $\overline{\text{RST}}$/NMI is not asserted (low), GDB430 sets the logic driving $\overline{\text{RST}}$/NMI to high-impedance, and $\overline{\text{RST}}$/NMI is pulled high via a resistor on the PCB.

   $\overline{\text{RST}}$/NMI is asserted and negated after power is applied when GDB430 is started. $\overline{\text{RST}}$/NMI is then asserted and negated a second time after device initialization is complete.
7. **GDB430 can debug a device whose program reconfigures the function of the $\overline{\text{RST}}$/NMI pin to NMI.**
8. **The level of the XOUT/TCLK pin is undefined when GDB430 resets the device.** The logic driving XOUT/TCLK is set to high-impedance at all other times.
9. **When making current measurements of the device, ensure that the JTAG control signals are released**, otherwise the device is powered by the signals on the JTAG pins and the measurements are erroneous. See FAQs Debugging #10 and Hardware #12.
10. **When GDB430 has control of the device, the CPU is on** (i.e., it is not in low power mode) regardless of the settings of the low power mode bits in the status register. Any low-power mode condition is restored prior to STEP or GO. Consequently, do not measure the power consumed by the device while GDB430 has control of the device. Instead, run the application using Release JTAG on run.
11. The MEMORY window correctly displays the contents of memory where it is present. However, **the MEMORY window incorrectly displays the contents of memory where there is none present.** Memory should only be used in the address ranges as specified by the device data sheet.

12. GDB430 utilizes the system clock to control the device during debugging. Therefore, **device counters, etc., that are clocked by the Main System Clock (MCLK) are affected when GDB430 has control of the device.** Special precautions are taken to minimize the effect upon the Watchdog Timer. The CPU core registers are preserved. All other clock sources (SMCLK, ACLK) and peripherals continue to operate normally during emulation. In other words, **the Flash Emulation Tool is a partially intrusive tool.**

    Devices that support Clock Control can further minimize these effects by selecting to stop the clock(s) during debugging. Set in RUN → CLOCK_CONTROL.
13. When programming the Flash, **do not set a breakpoint on the instruction immediately following the write to Flash operation.** A simple work-around to this limitation is to follow the write to Flash operation with a NOP, and set a breakpoint on the instruction following the NOP.
14. Multiple internal machine cycles are required to clear and program the Flash memory. **When single stepping over instructions that manipulate the Flash,** control is given back to GDB430 before these operations are complete. Consequently, **GDB430 updates its memory window with erroneous information.** A workaround to this behavior is to follow the Flash access instruction with a NOP, and then step past the NOP before reviewing the effects of the Flash access instruction.
15. **Bits that are cleared when read during normal program execution** (i.e., interrupt flags) **are cleared when read while being debugged** (i.e., memory dump, peripheral registers).

    Using certain MSP430 devices with enhanced emulation logic such as MSP430F43x/44x devices, bits do not behave this way (i.e., the bits are not cleared by GDB430 read operations).
16. **GDB430 cannot be used to debug programs that execute in the RAM of F12x and F41x devices.** A workaround to this limitation is to debug programs in Flash.
17. **While single stepping with active and enabled interrupts, it can appear that only the interrupt service routine (ISR) is active** (i.e., the non-ISR code never appears to execute, and the single step operation stops on the first line of the ISR). However, this behavior is correct because the device processes an active and enabled interrupt before processing non-ISR (i.e., mainline) code. A workaround for this behavior is, while within the ISR, to disable the GIE bit on the stack, so that interrupts are disabled after exiting the ISR. This permits the non-ISR code to be debugged (but without interrupts). Interrupts can later be re-enabled by setting GIE in the status register in the Register window.

    On devices with Clock Control, it may be possible to suspend a clock between single steps and delay an interrupt request. Set in RUN → CLOCK_CONTROL.
18. On devices equipped with a Data Transfer Controller (DTC), **the completion of a data transfer cycle preempts a single step of a low power mode instruction.** The device advances beyond the low-power mode instruction only after an interrupt is processed. Until an interrupt is processed, it appears that the single step has no effect. A workaround to this situation is to set a breakpoint on the instruction following the low-power more instruction, and then execute (Run) to this breakpoint.
19. **The transfer of data by the Data Transfer Controller (DTC) may not stop precisely when the DTC is stopped in response to a single step or a breakpoint.** When the DTC is enabled and a single step is performed, one or more bytes of data can be transferred. When the DTC is enabled and configured for two-block transfer mode, the DTC may not stop precisely on a block boundary when stopped in response to a single step or a breakpoint.
20. **The stack overflow can be debugged.** It is possible to debug an application which causes a stack overflow. Just set Break on Stack Overflow (right mouse click in the debug window and select in context menu "Break on Stack Overflow". The program execution stops on the instruction that caused the stack overflow. The size of stack can be adjusted in Project → Properties → C/C++ Build → MSP430 Executable Linker → General Options.

    Note: This option is not available on the devices with small Enhanced Emulation Module (EEM).
21. **Breakpoint on Variable** stops the execution on read/write access to the variable with defined condition, e.g., halt if var1>10. To set the breakpoint select the variable in editor (Debug perspective), mouse right click and select in the context menu "Variable Storage breakpoint".

    Restriction 1: This breakpoint is applicable to global variables and register variables. To set the breakpoint in the latter case, halt execution in the function where observation of the variable is desired (set a code breakpoint there). Set the breakpoint and delete (or disable) the code breakpoint in the function and run/restart the application.

    Restriction 2: This breakpoint applicable to the variables 8 and 16 bit wide.

Hardware

This appendix contains information relating to the FET hardware, including schematics and PCB pictorials.

Topic **Page**

B.1 Schematics and PCBs

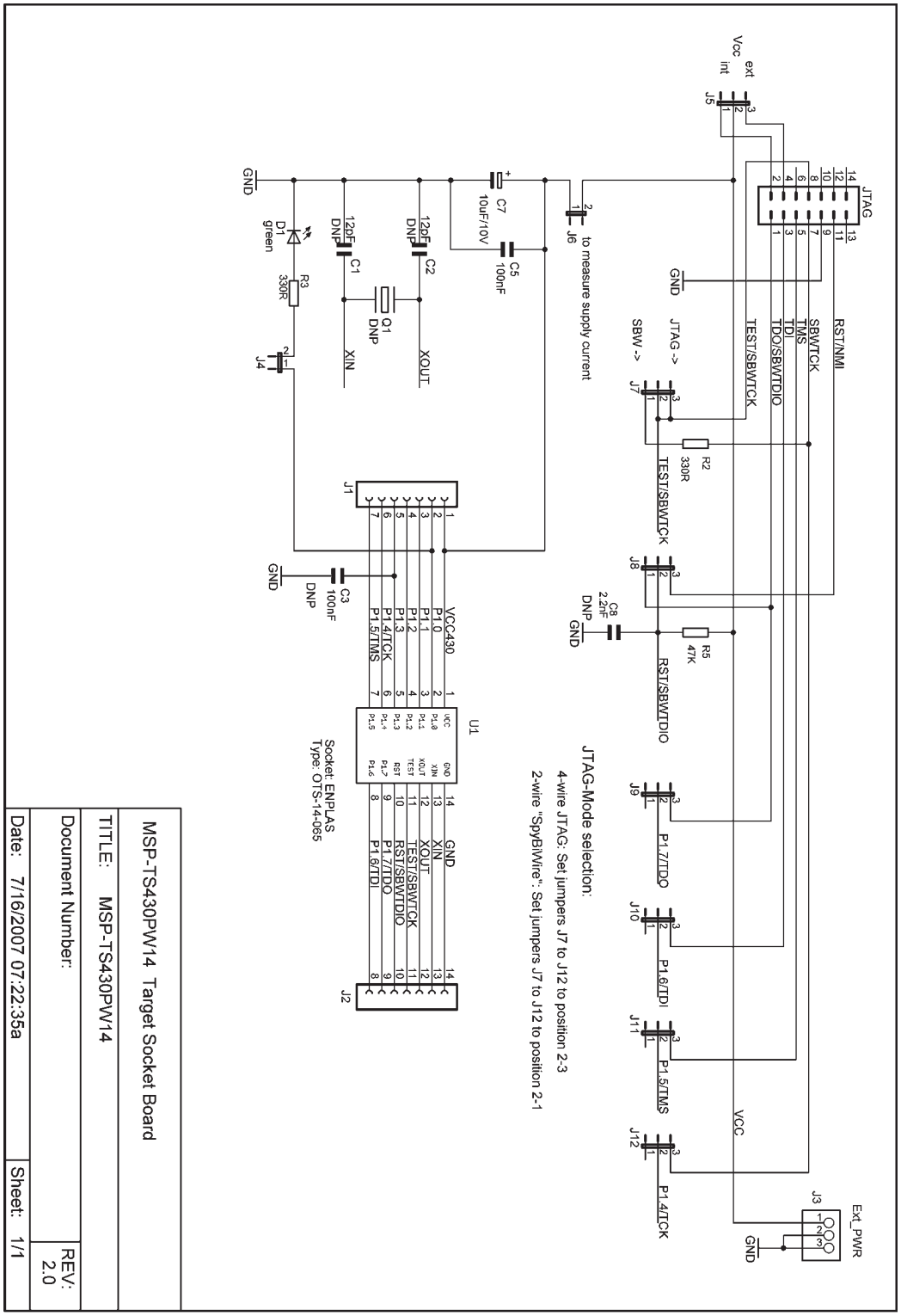


Figure B-1. MSP-TS430PW14 Target Socket Module, Schematic

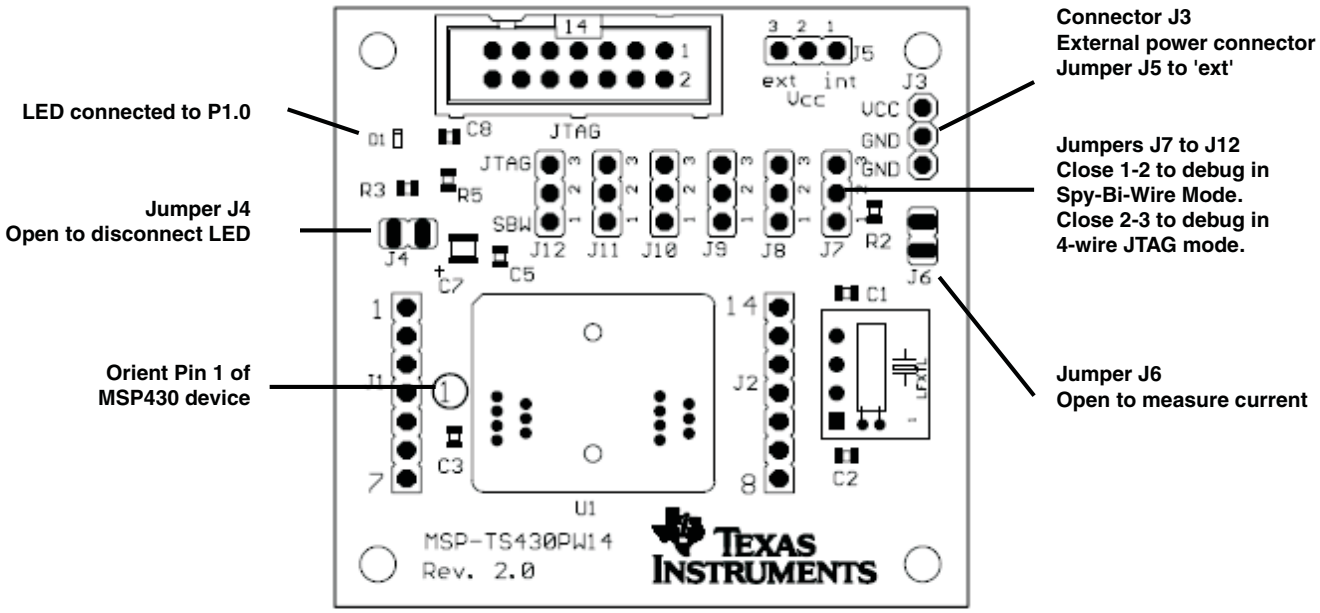


Figure B-2. MSP-TS430PW14 Target Socket Module, PCB

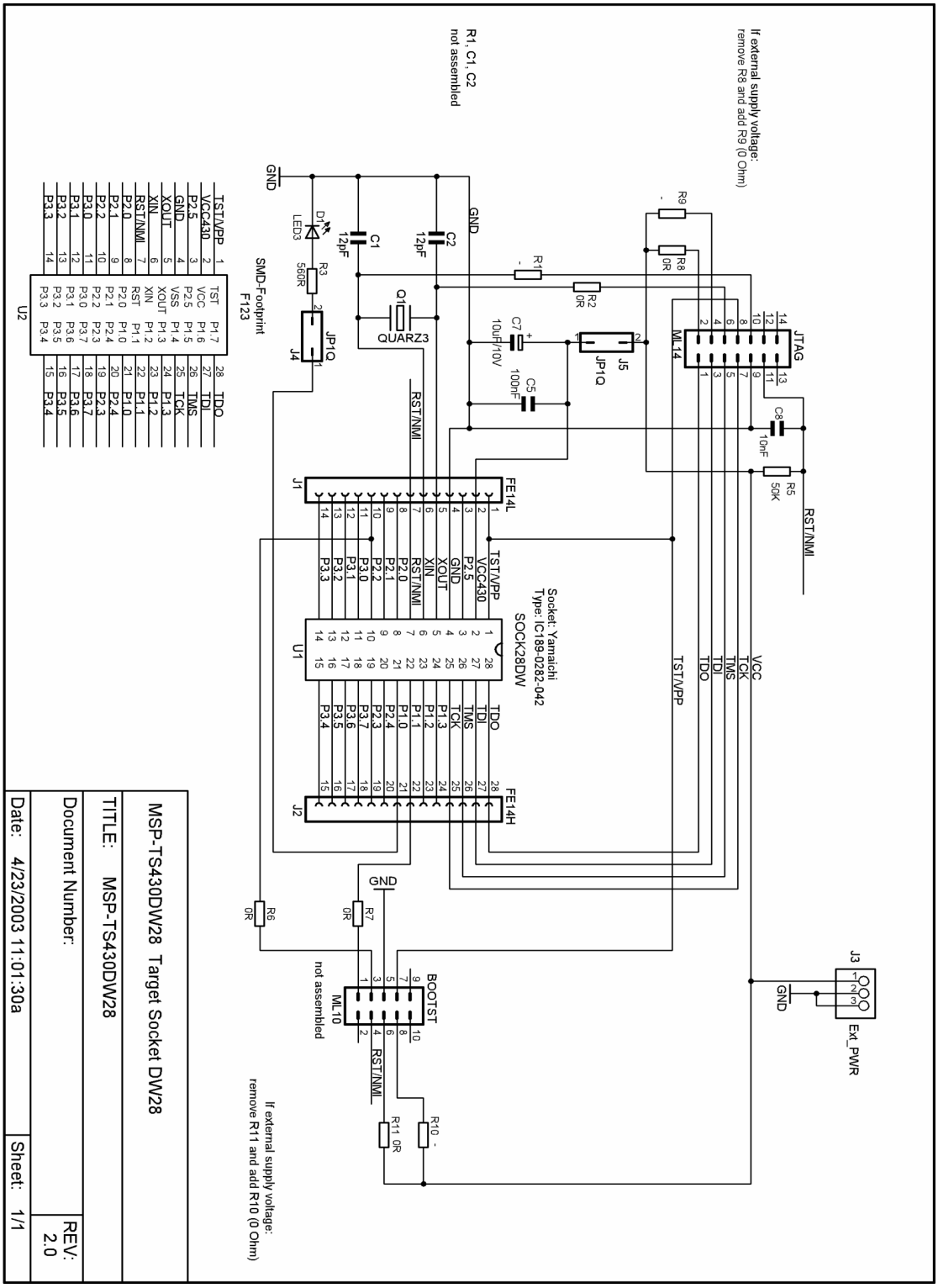


Figure B-3. MSP-TS430DW28 Target Socket Module, Schematic

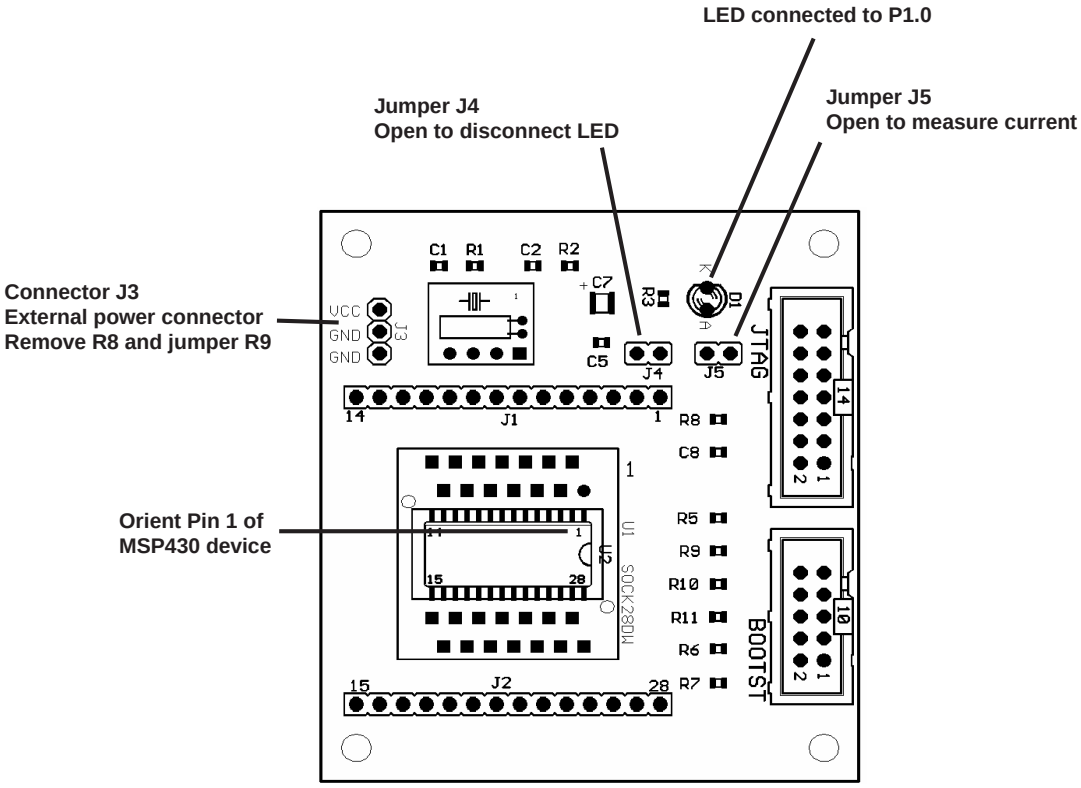


Figure B-4. MSP-TS430DW28 Target Socket Module, PCB

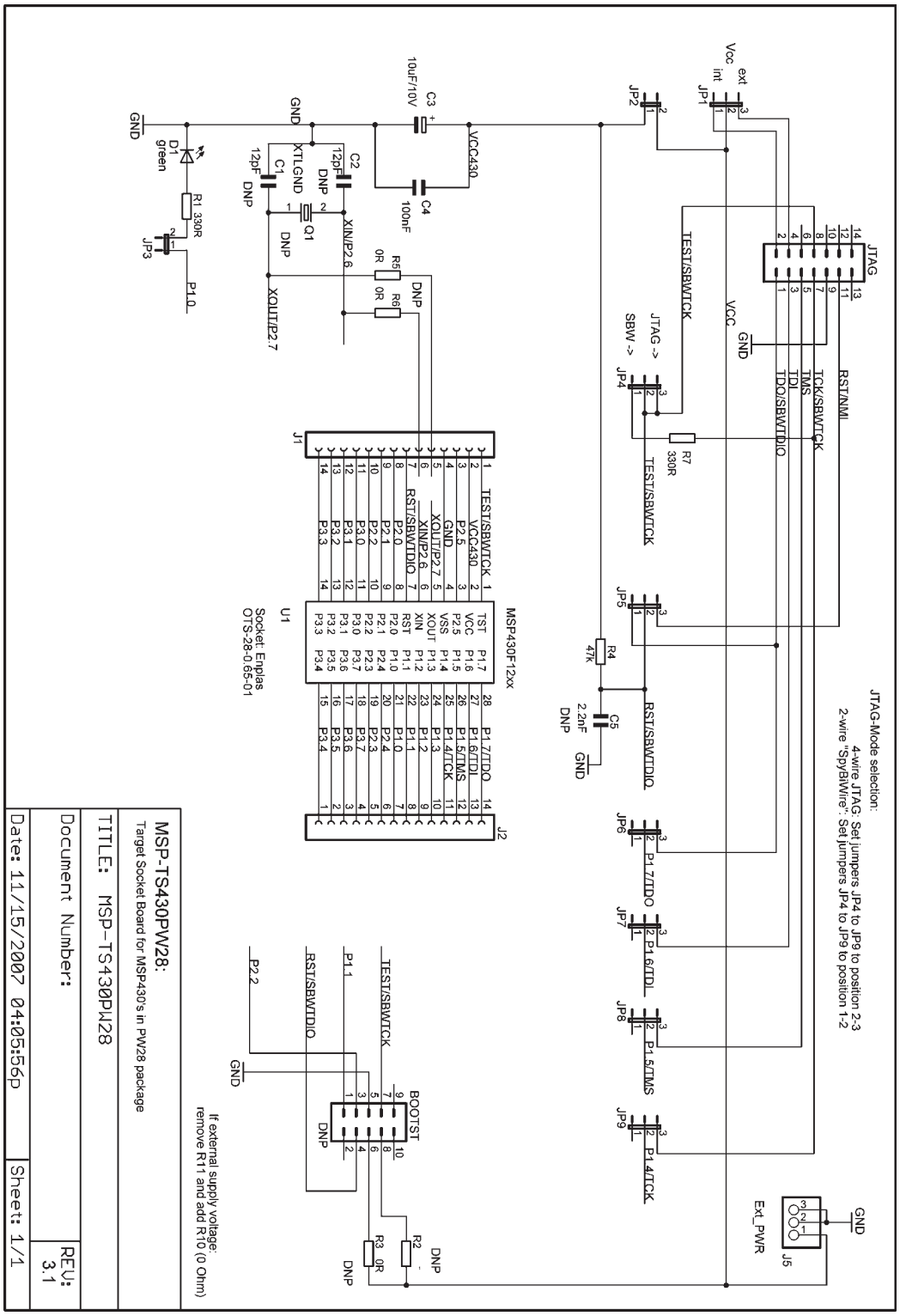


Figure B-5. MSP-TS430PW28 Target Socket Module, Schematic

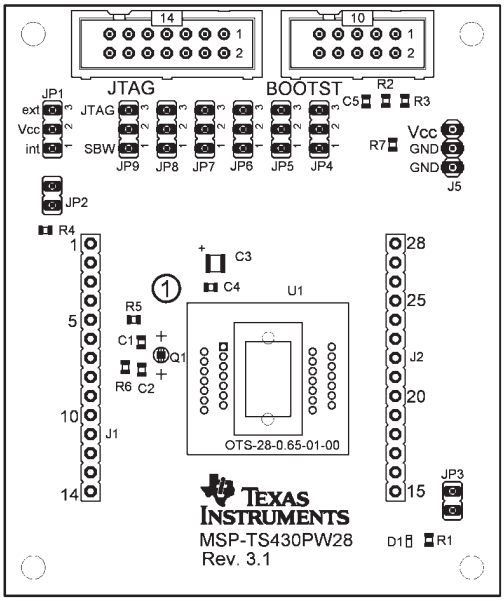


Figure B-6. MSP-TS430PW28 Target Socket Module, PCB

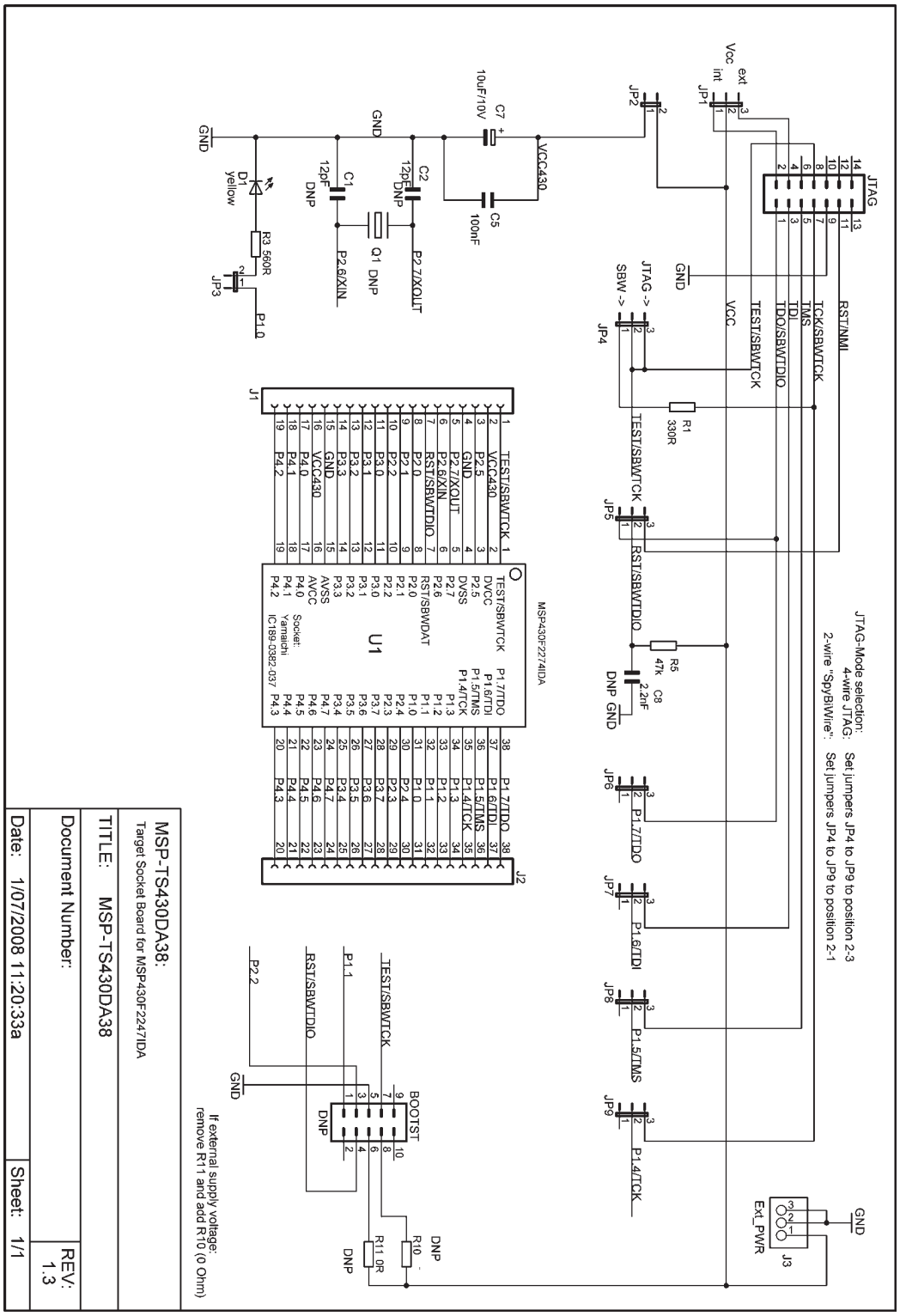


Figure B-7. MSP-TS430DA38 Target Socket Module, Schematic

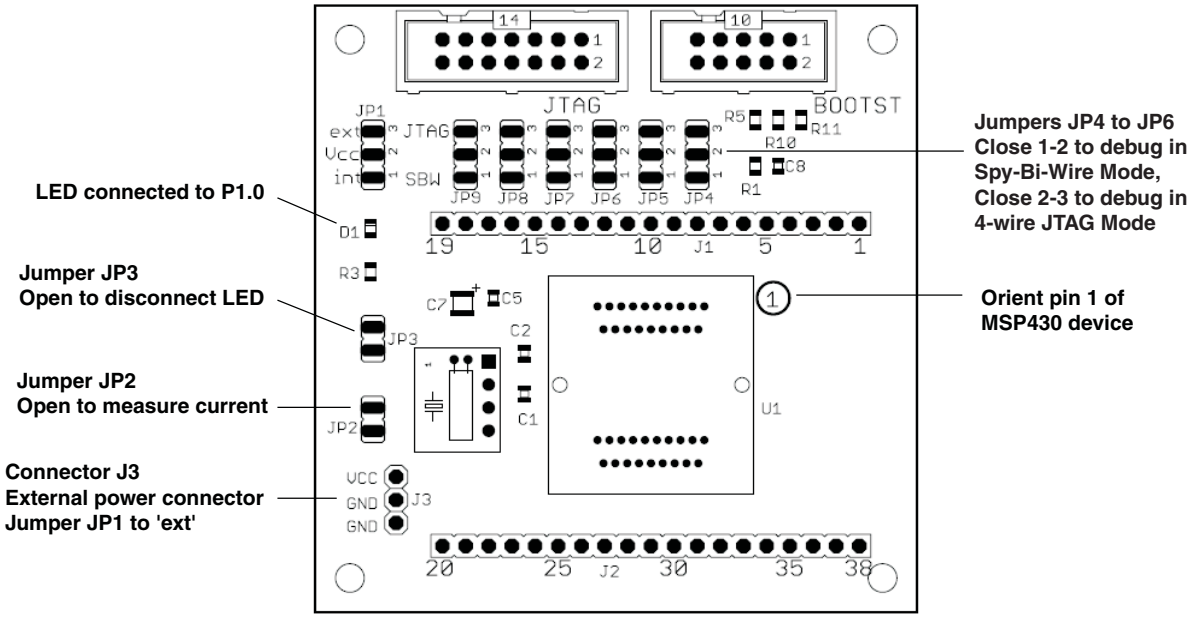


Figure B-8. MSP-TS430DA38 Target Socket Module, PCB

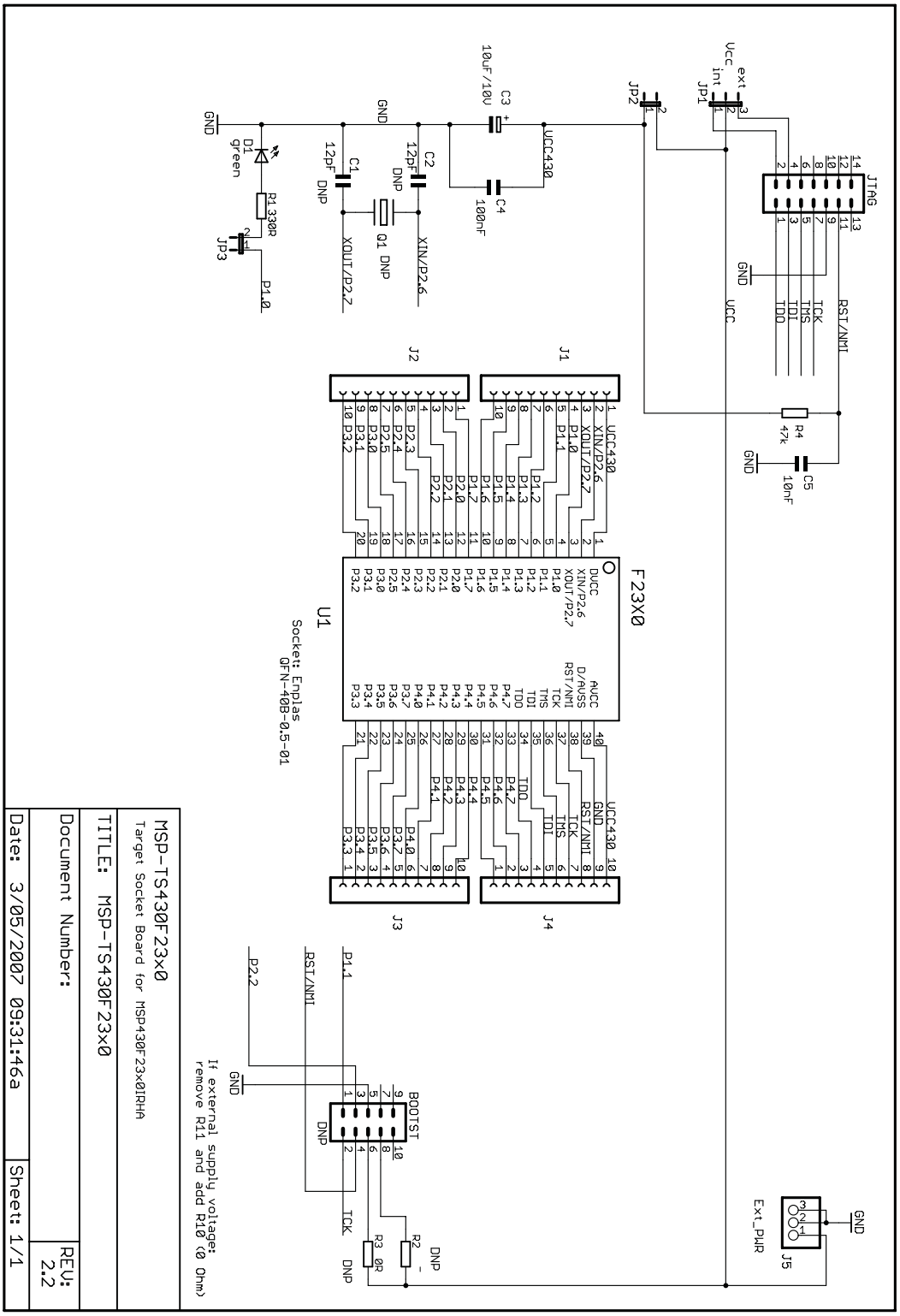


Figure B-9. MSP-TS430QFN23x0 Target Socket Module, Schematic

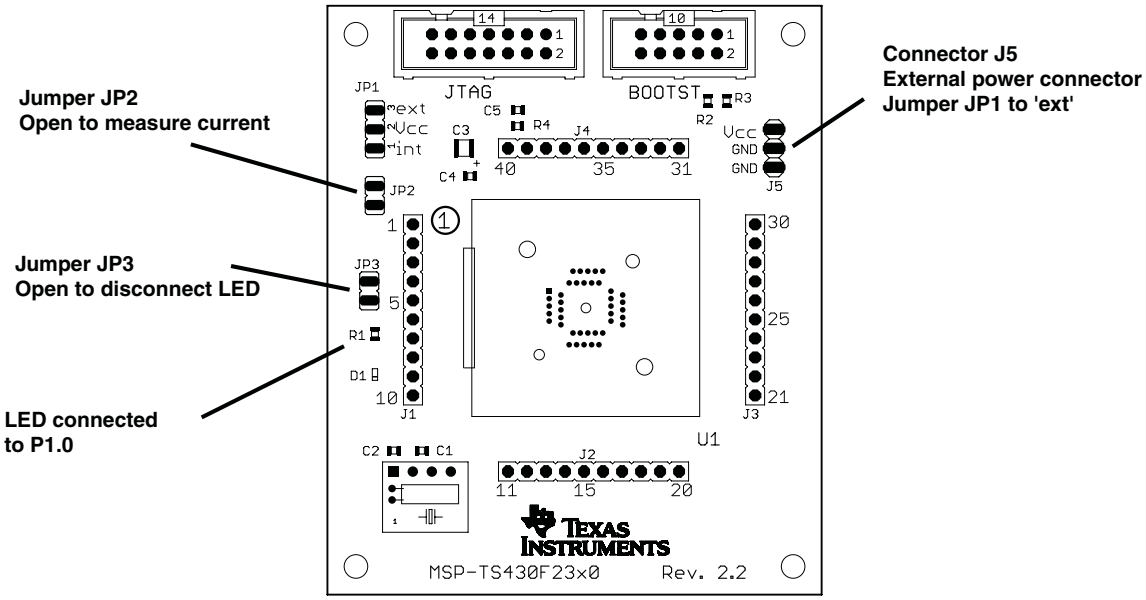


Figure B-10. MSP-TS430QFN23x0 Target Socket Module, PCB

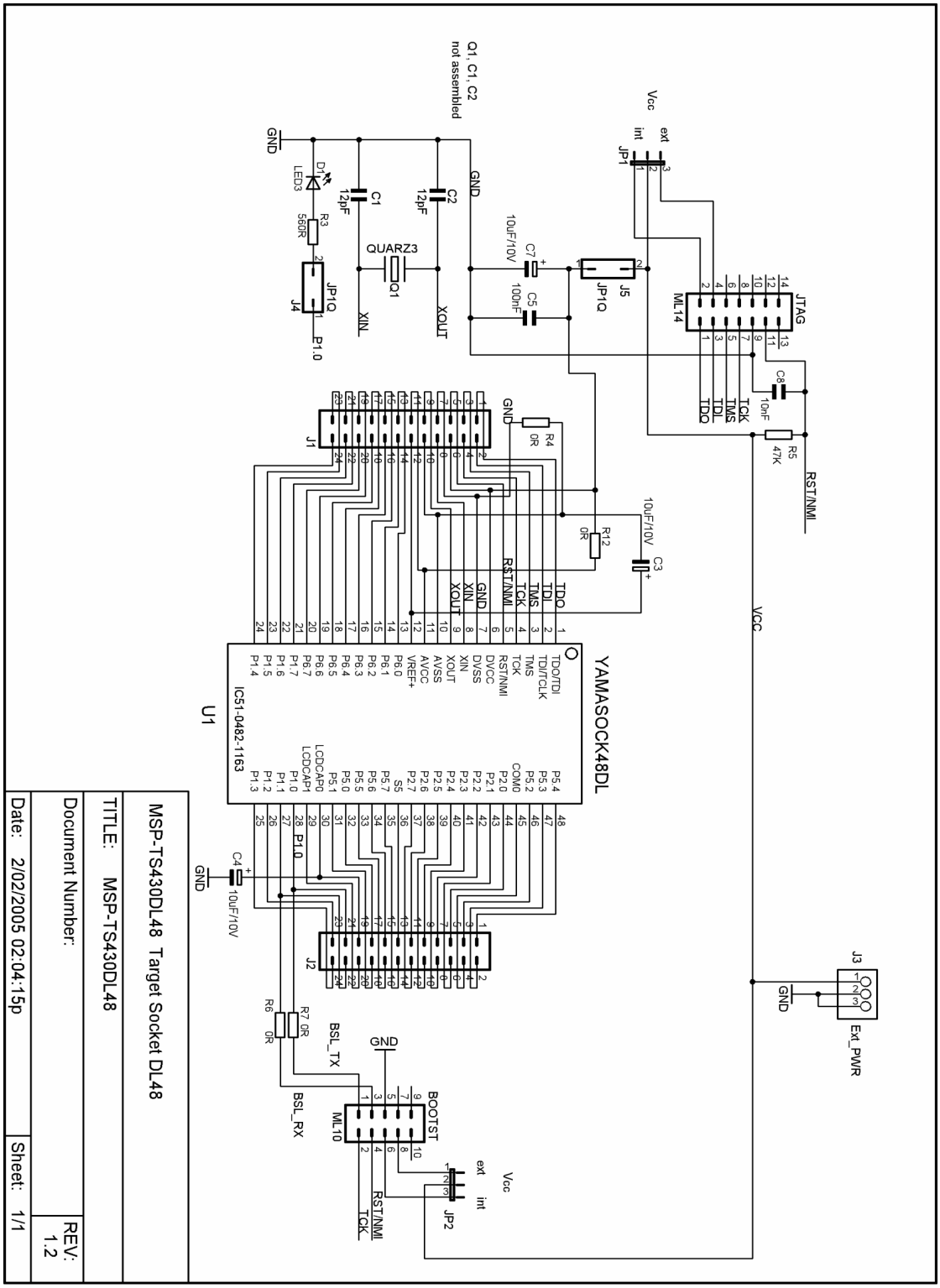


Figure B-11. MSP-TS430DL48 Target Socket Module, Schematic

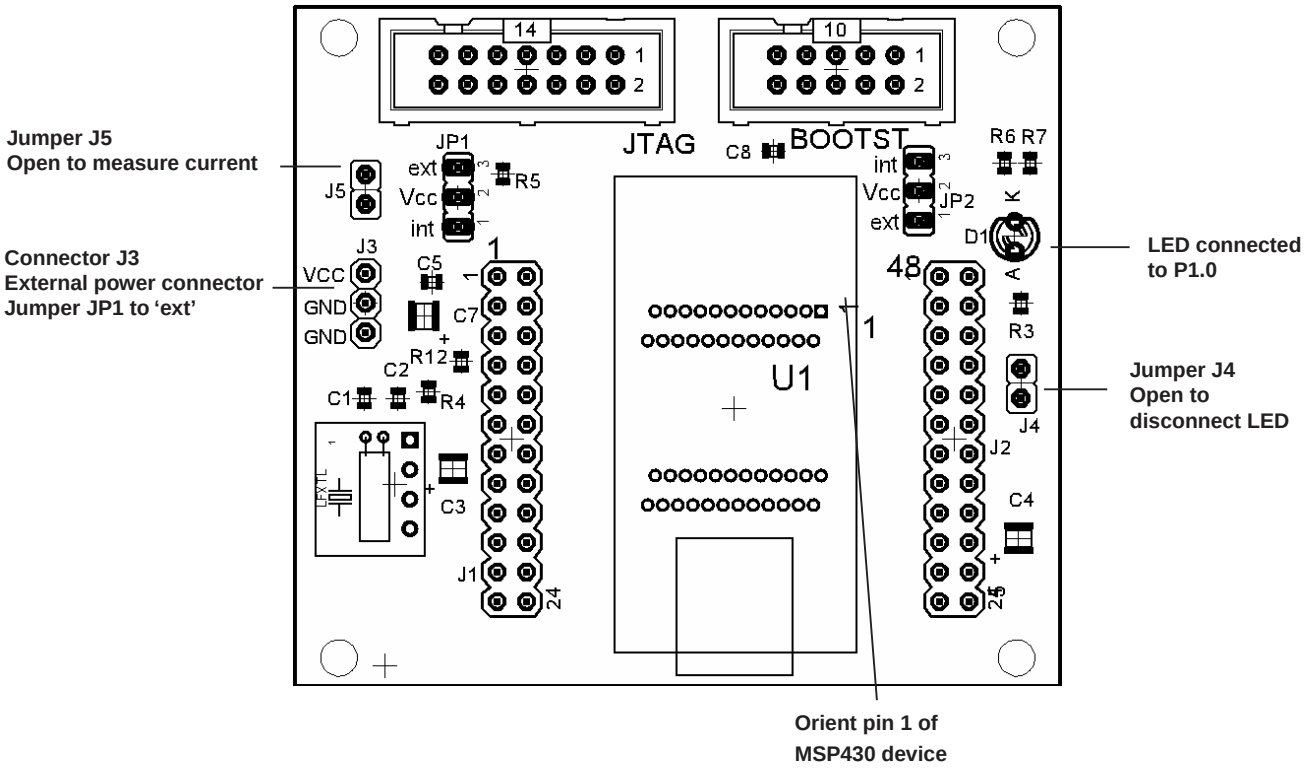
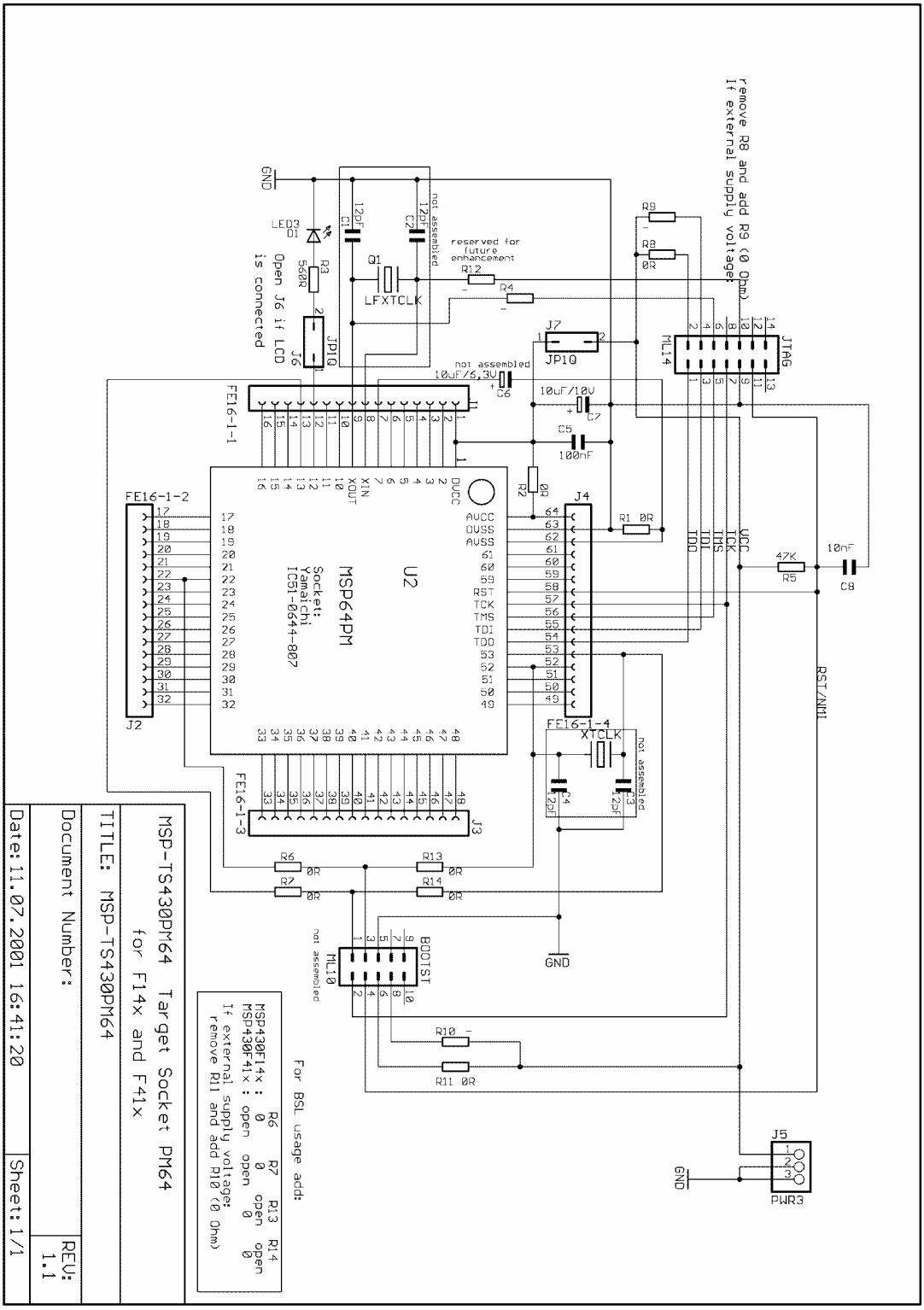


Figure B-12. MSP-TS430DL48 Target Socket Module, PCB



Note: Connections between the JTAG header and pins XOUT and XIN are no longer required and should not be made.

Figure B-13. MSP-TS430PM64 Target Socket Module, Schematic

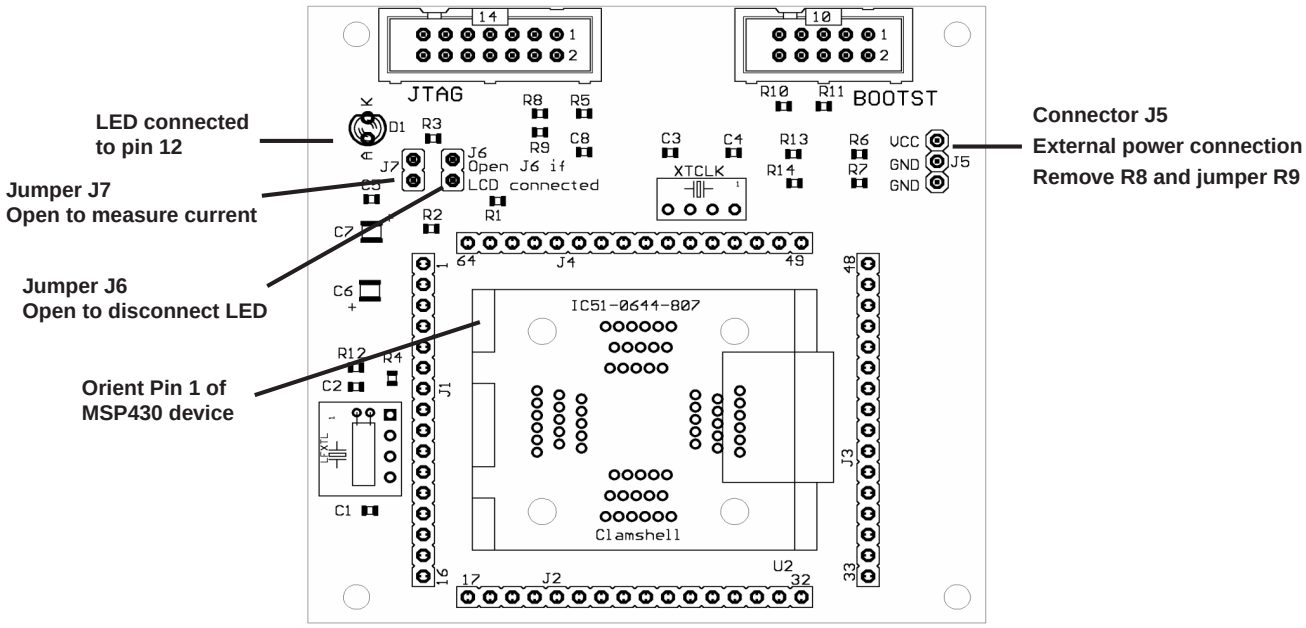


Figure B-14. MSP-TS430PM64 Target Socket Module, PCB

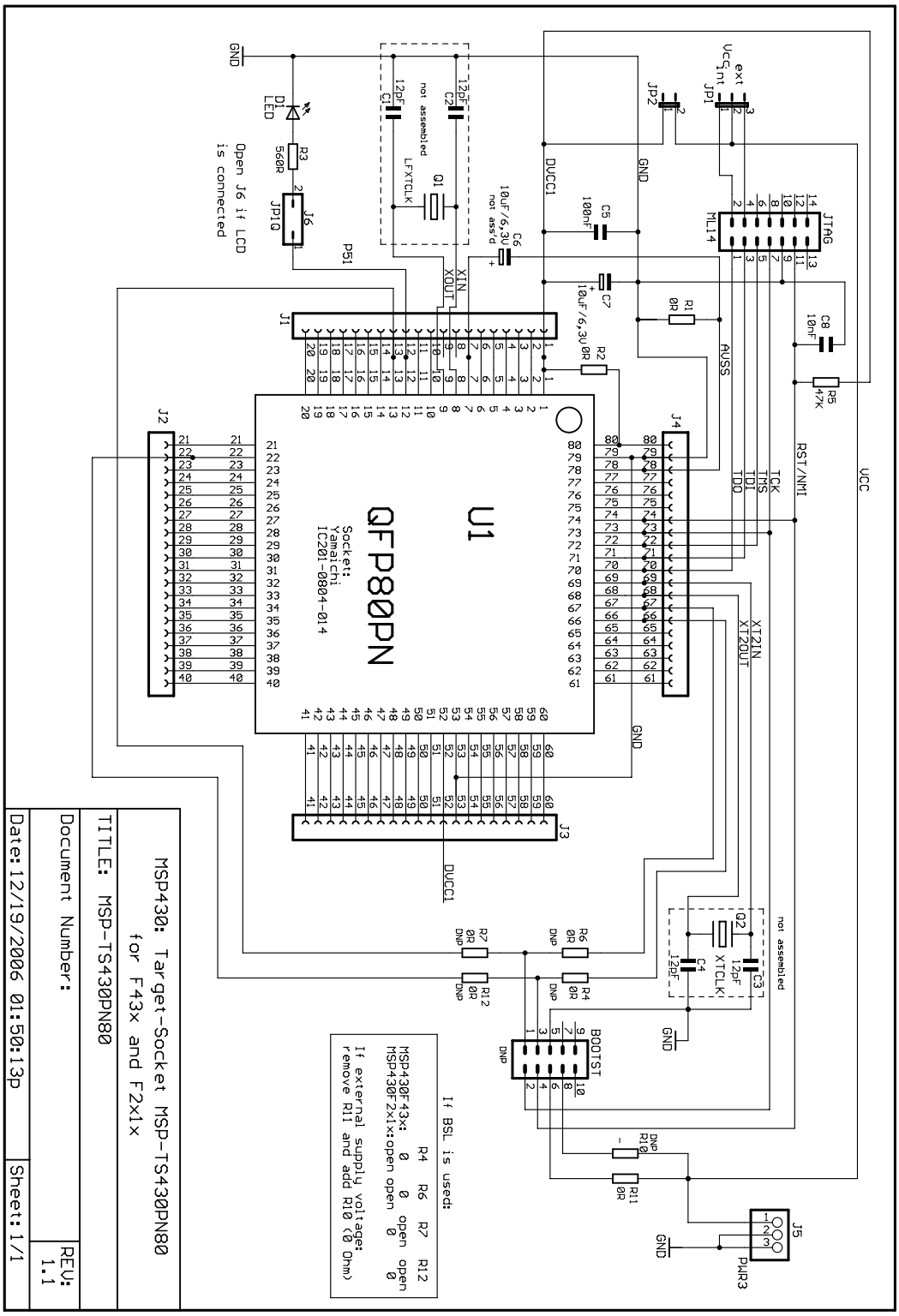


Figure B-15. MSP-TS430PN80 Target Socket Module, Schematic

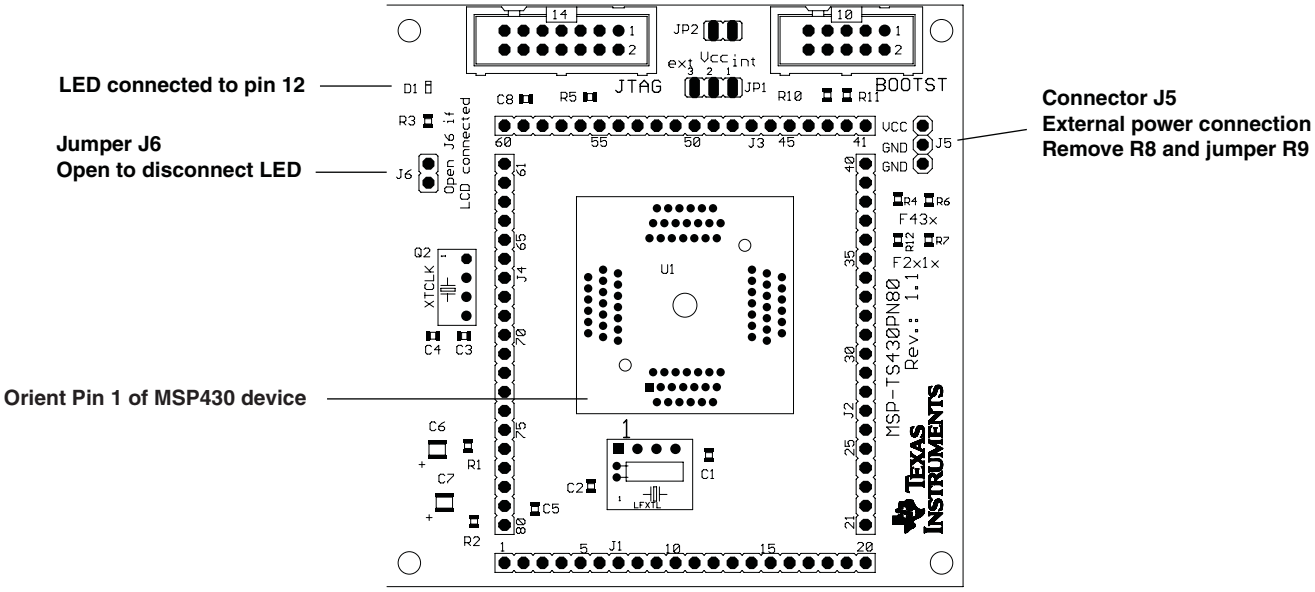
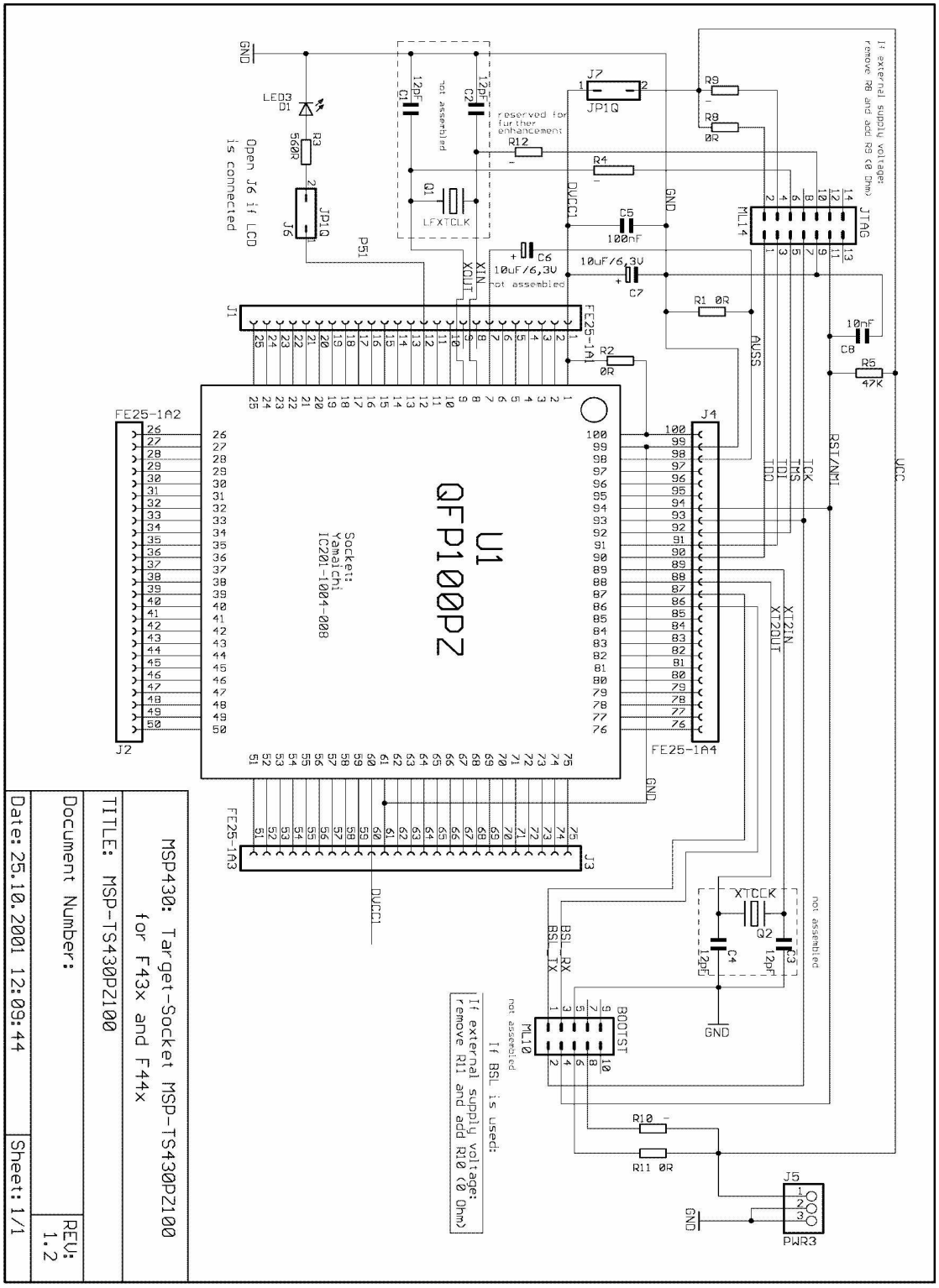


Figure B-16. MSP-TS430PN80 Target Socket Module, PCB



Note: Connections between the JTAG header and pins XOUT and XIN are no longer required and should not be made.

Figure B-17. MSP-TS430PZ100 Target Socket Module, Schematic

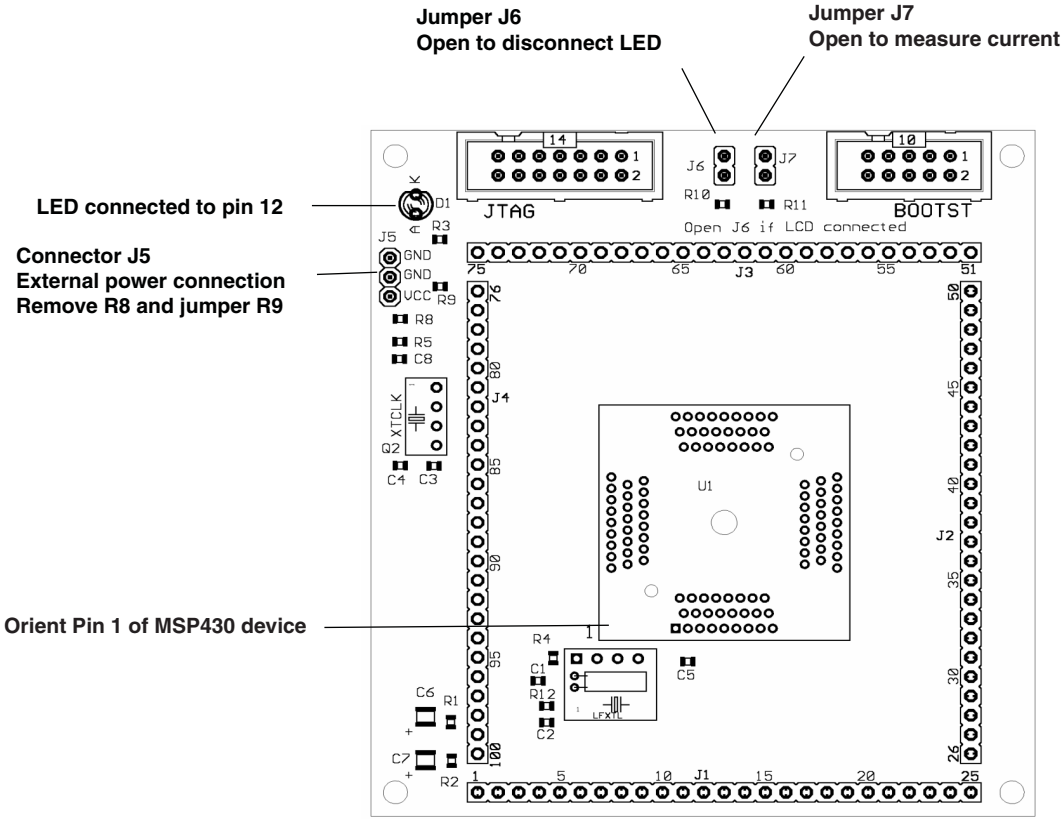


Figure B-18. MSP-TS430PZ100 Target Socket Module, PCB

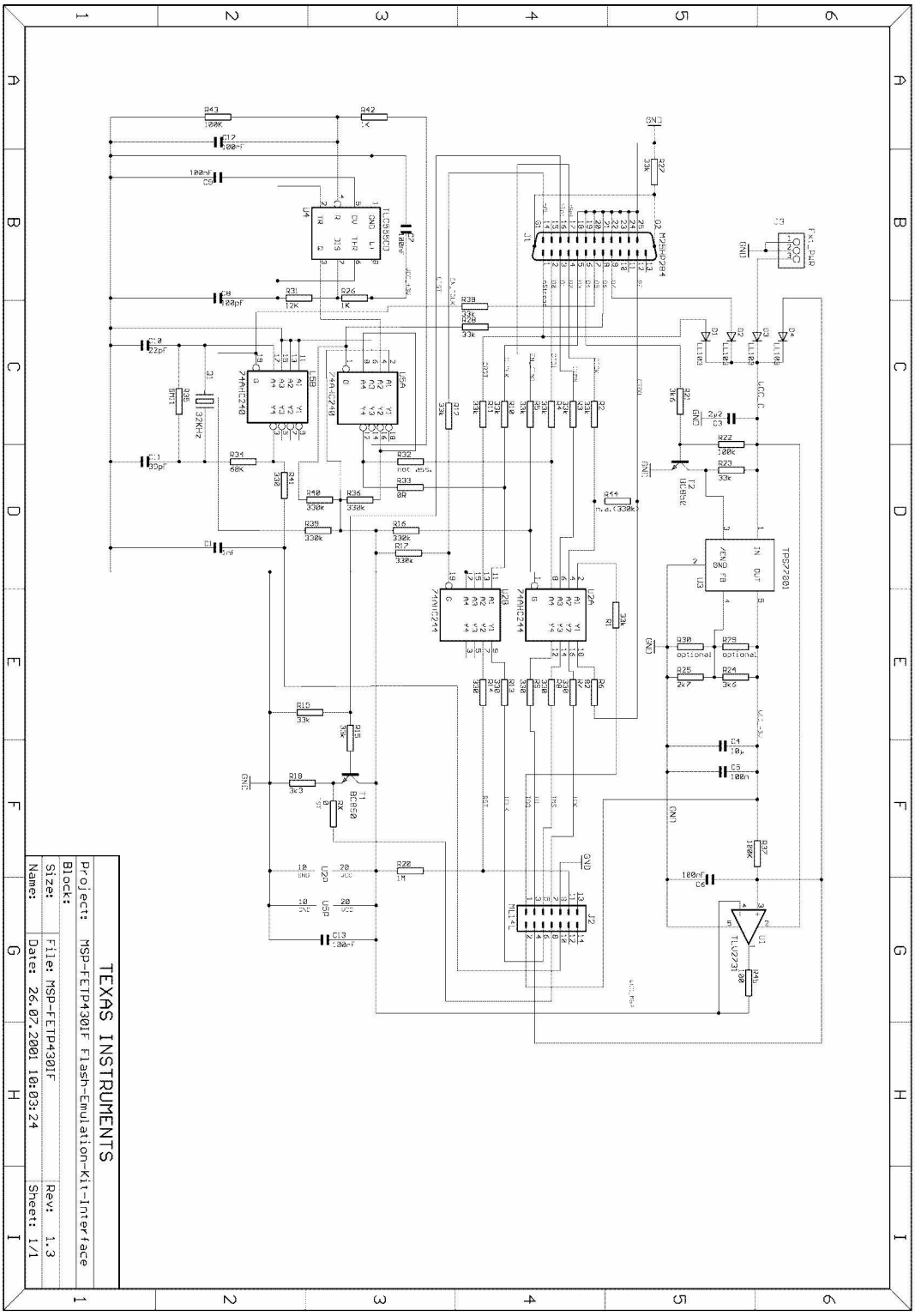


Figure B-19. MSP-FET430PIF FET Interface Module, Schematic

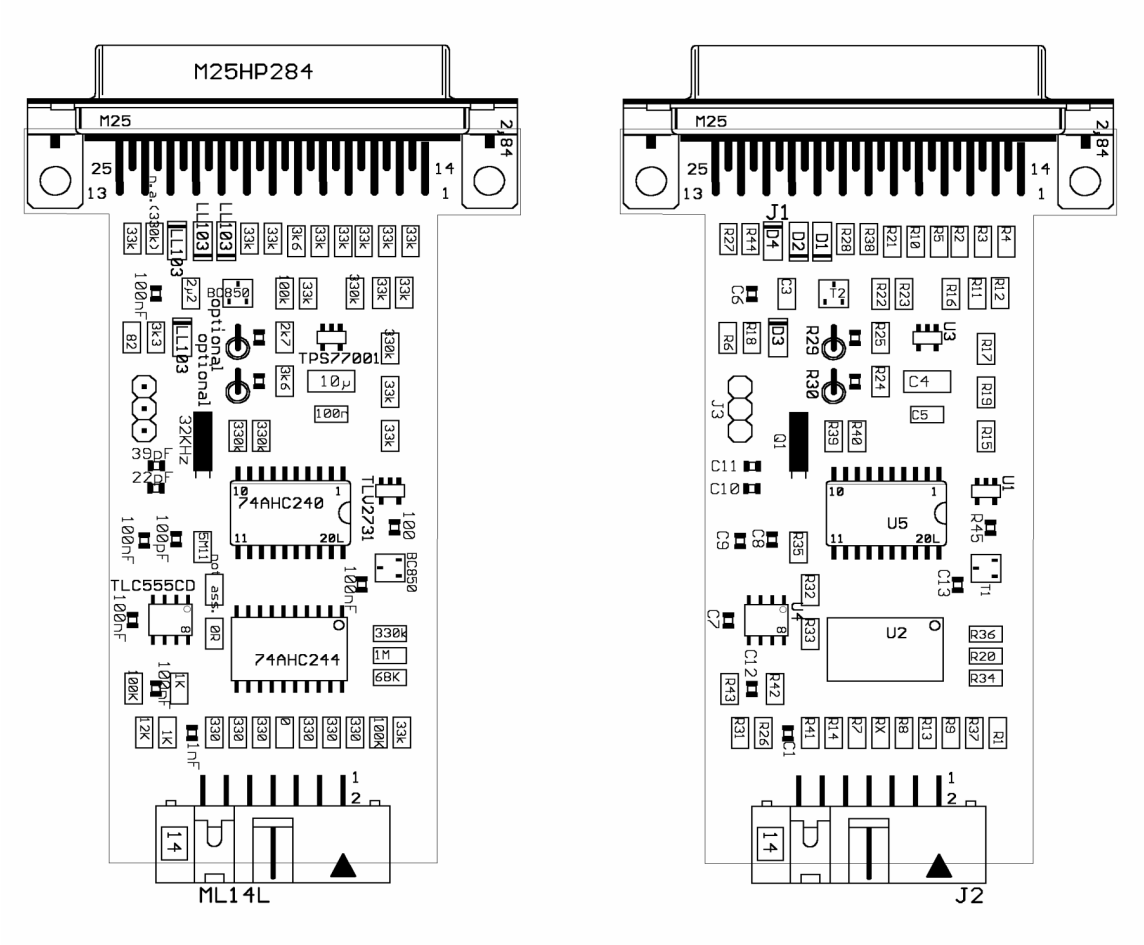


Figure B-20. MSP-FET430PIF FET Interface Module, PCB

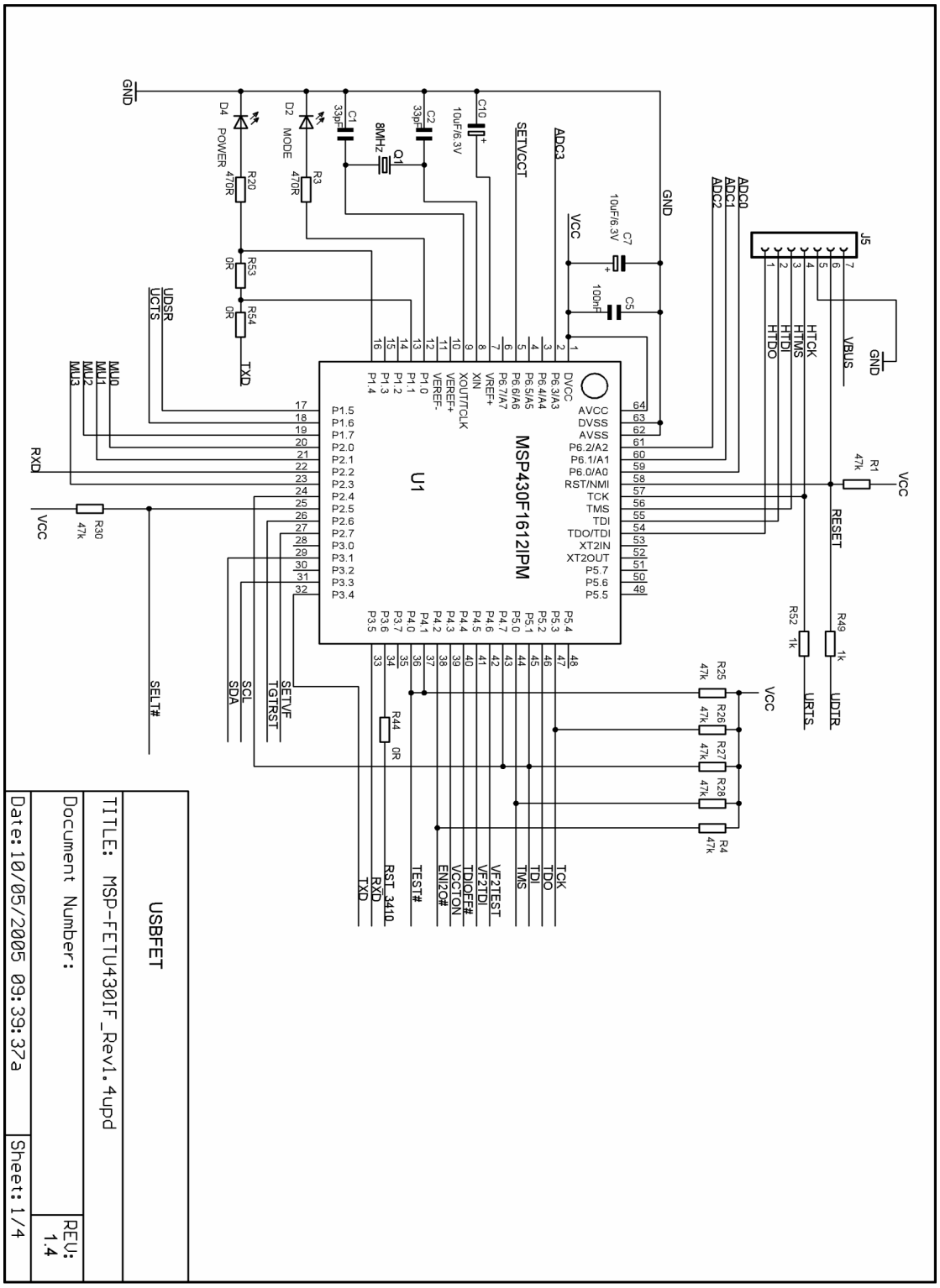


Figure B-21. MSP-FET430UIF USB Interface, Schematic (1 of 4)

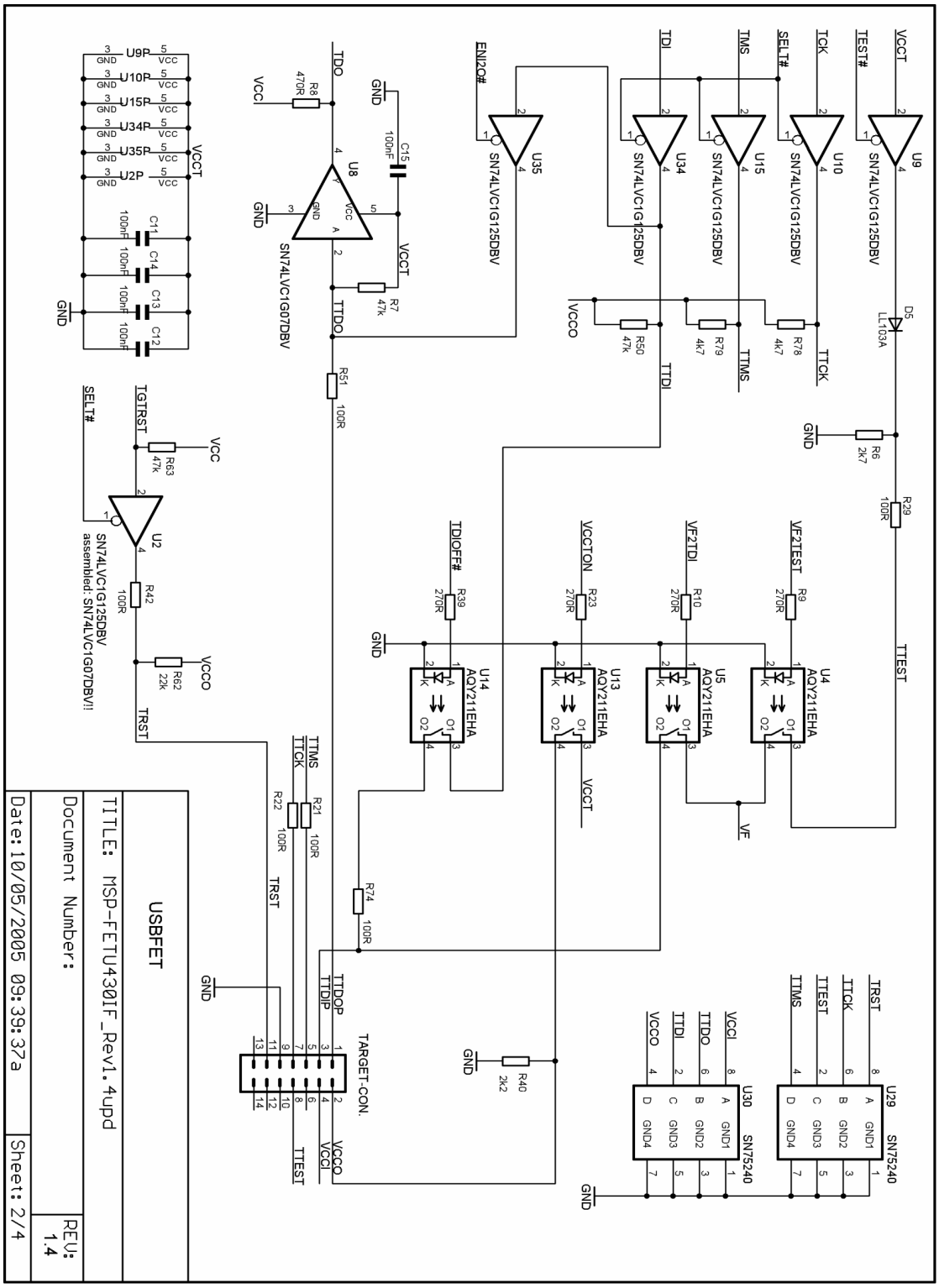


Figure B-22. MSP-FET430UIF USB Interface, Schematic (2 of 4)

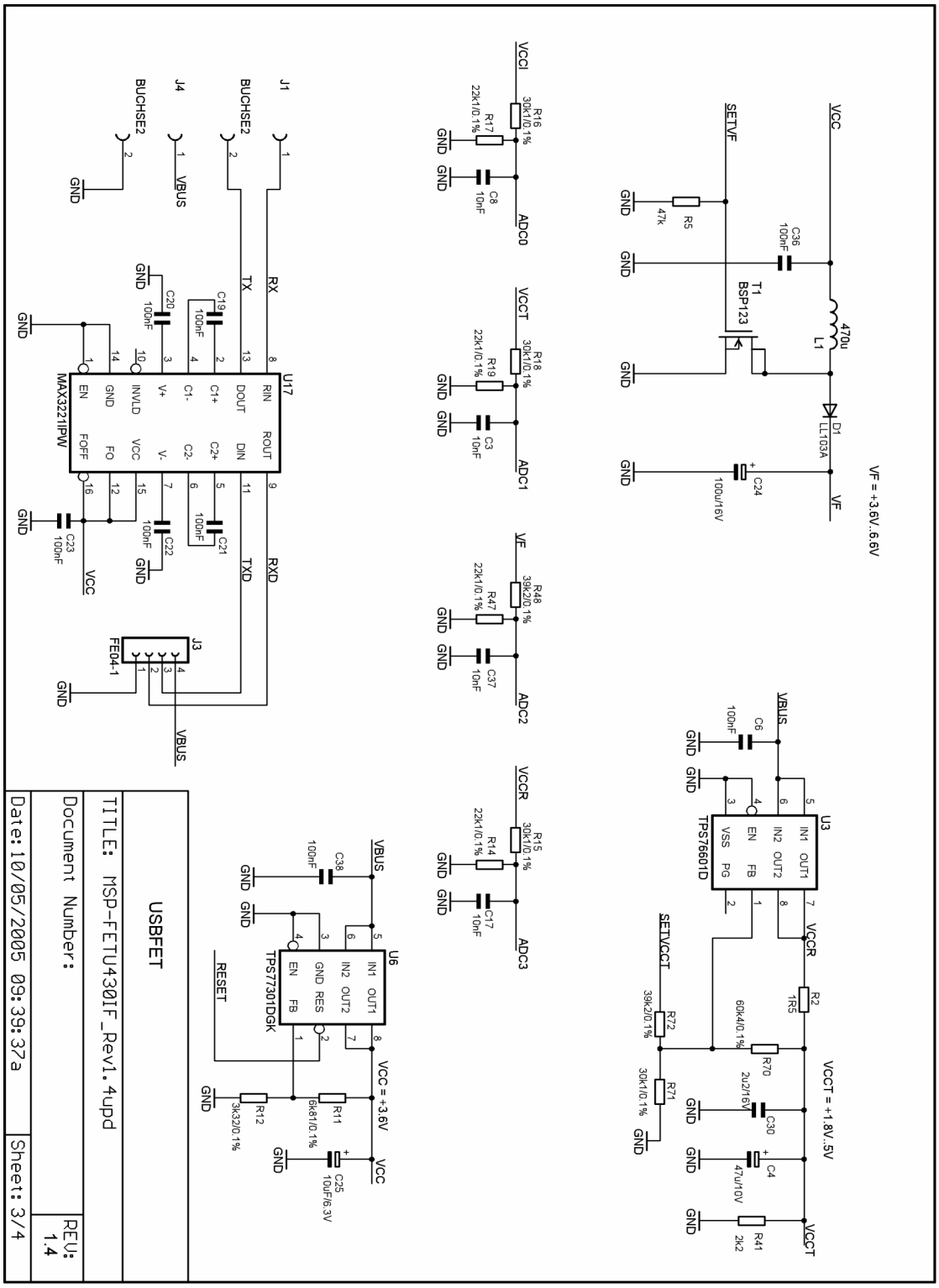


Figure B-23. MSP-FET430UIF USB Interface, Schematic (3 of 4)

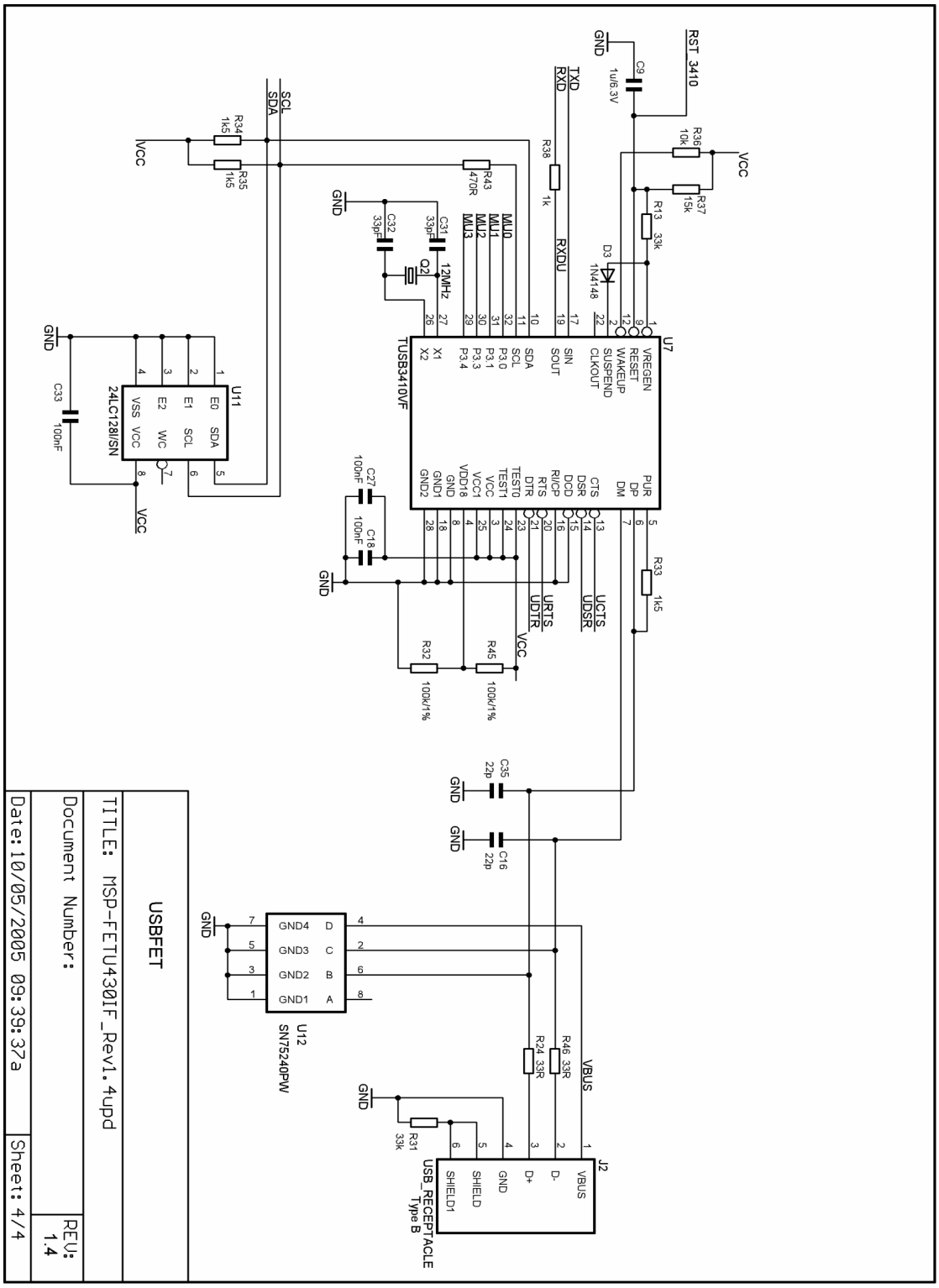


Figure B-24. MSP-FET430UIF USB Interface, Schematic (4 of 4)

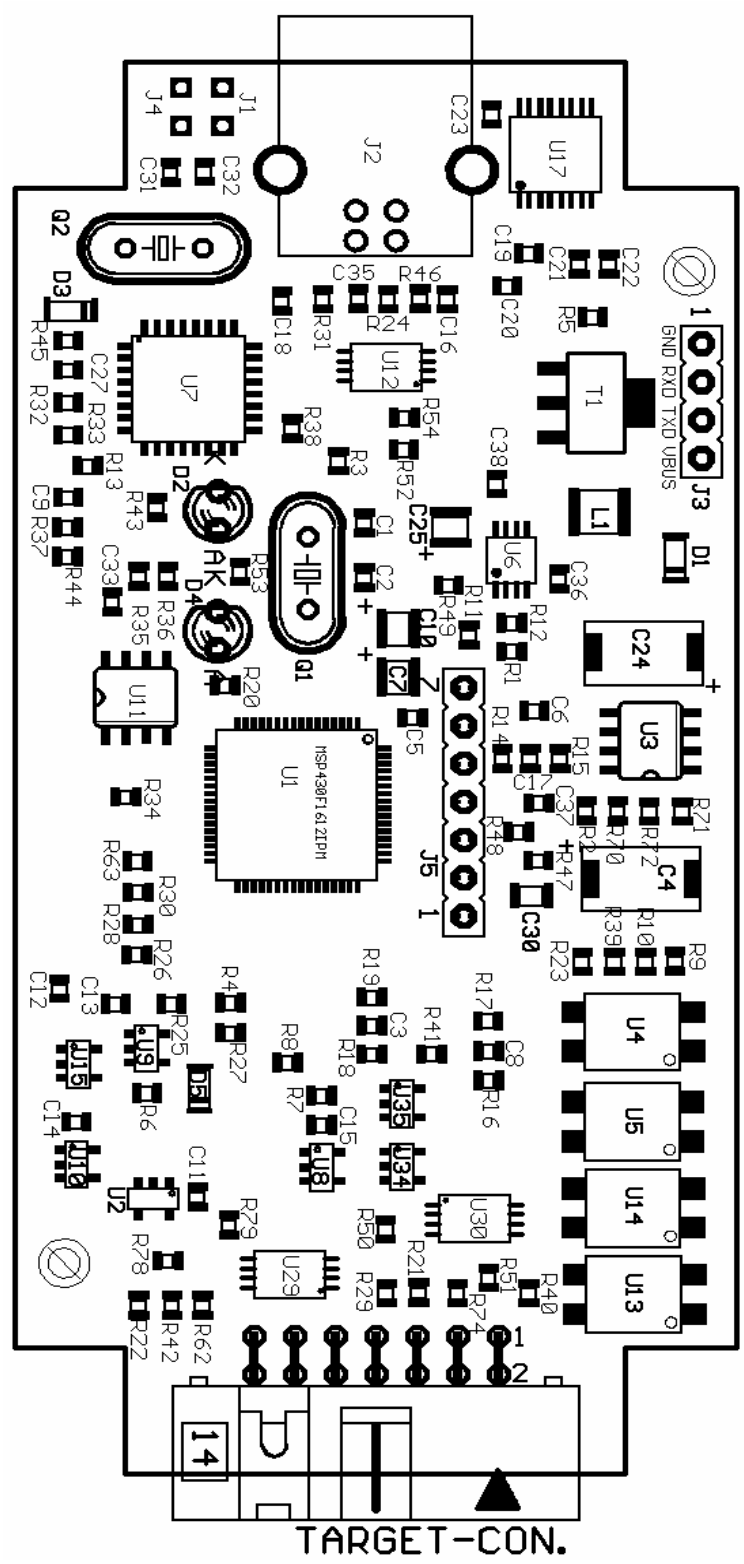


Figure B-25. MSP-FET430UIF USB Interface, PCB

B.2 MSP-FET430UIF Revision History

Revision 1.3

- Initial released hardware version

Assembly change on 1.3 (May 2005)

- R29, R51, R42, R21, R22, R74: value changed from 330R to 100R

Changes 1.3 to 1.4 (Aug 2005)

- J5: VBUS and RESET additionally connected
- R29, R51, R42, R21, R22, R74: value changed from 330R to 100R
- U1, U7: F1612 can reset TUSB3410; R44 = 0R added
- TARGET-CON.: pins 6, 10, 12, 13, 14 disconnected from GND
- Firmware-upgrade option through BSL: R49, R52, R53, R54 added; R49, R52 are currently DNP
- Pullups on TCK and TMS: R78, R79 added
- U2: Changed from SN75LVC1G125DBV to SN75LVC1G07DBV

Note: Using a locally powered target board with hardware revision 1.4

Using an MSP-FET430UIF interface hardware revision 1.4 with populated R62 in conjunction with a locally powered target board is not possible. In this case, the target device RESET signal is pulled down by the FET tool. It is recommended to remove R62 to eliminate this restriction. This component is located close to the 14-pin connector on the MSP-FET430UIF PCB. See the schematic and PCB drawings in this document for the exact location of this component.

Assembly change on 1.4 (January 2006)

- R62: not populated

IAR 2.x/3.x to CCE C-Migration

Source code for the TI CCE C compiler and source code for the IAR™ Embedded Workbench™ compiler are not 100% compatible. While the standard ANSI/ISO C code is portable among these tools, implementation-specific extensions, such as for interrupt vector definition and processor intrinsics, differ and need to be ported. This appendix documents the major differences between the two compilers.

Topic **Page**

C.1 Interrupt Vector Definition

The IAR C-compiler offers a #pragma directive to assign interrupt vectors, which can be used as illustrated in the following example:

```
// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  P1OUT ^= 0x01;                          // Toggle P1.0
  CCR0 += 50000;                          // Add Offset to CCR0
}
```

This code snippet writes the address of the Timer_A function to the indicated MSP430 interrupt vector TIMERA0_VECTOR. The constants for the interrupt vectors are defined in the MSP430 device-specific header files (msp430xxx.h) which is supplied with the IAR tools.

Using the CCE C-compiler, interrupt vectors must be assigned by defining 16-bit constants that are pointing to the appropriate interrupt service function. Then, these constants need to be placed in the MSP430 interrupt vector table using segment placing directives. To make this process easier, convenience macros are provided for all the device interrupt vectors. These macros are defined in the device-specific header files that are included in the CCE installation. Note that the use of these macros requires that the function must be known before the macro is processed. This can be achieved by adding a prototype for the ISR at the top of the source code or by placing the interrupt vector definitions after the interrupt service functions.

```
__interrupt void Timer_A(void);        // Prototype for ISR
// Timer A0 interrupt service routine
TIMERA0_ISR(Timer_A)
__interrupt void Timer_A (void)
{
  P1OUT ^= 0x01;                          // Toggle P1.0
  CCR0 += 50000;                          // Add Offset to CCR0
}
```

Note that the macro names such as TIMERA0_ISR are reserved and cannot be used as function names.

C.2 Intrinsic Functions

CCE and IAR tools use similar but different names for MSP430 processor specific intrinsic functions. The most used functions can be converted easily using the following table.

Generated Assembly	CCE Intrinsic Function	IAR Intrinsic Function
EINT	_enable_interrupts()	__enable_interrupt()
DINT	_disable_interrupts()	__disable_interrupt()
BIC a, SR	_bic_SR_register(a)	__bic_SR_register(a)
BIC a, saved_SR	_bic_SR_register_on_exit(a)	__bic_SR_register_on_exit(a)
BIS a, SR	_bis_SR_register(a)	__bis_SR_register(a)
BIS a, saved_SR	_bis_SR_register_on_exit(a)	__bis_SR_register_on_exit(a)
MOV SR, dst	_get_SR_register()	__get_SR_register()
MOV saved_SR, dst	_get_SR_register_on_exit()	__get_SR_register_on_exit()
MOV a, dst SWPB dst	_swap_bytes(a)	__swap_bytes(a)
NOP	_nop()	__no_operation()

To facilitate the code migration process, the header file in430.h is provided. It contains convenience macros for mapping the IAR intrinsic syntax to CCE. This file is included by the device-specific header files such as msp430x14x.h, so most intrinsic functions do not need to be converted. Note that some of the IAR intrinsic functions do not have an equivalent, such as the functions for BCD arithmetic. If such functions are used, the user needs to rewrite them in C or assembly.

C.3 Data and Function Placement

C.3.1 Data Placement at an Absolute Location

The scheme implemented in the IAR compiler using either the @ operator or the #pragma location directive is not supported with the CCE compiler:

```
/* IAR C Code */
__no_init char alpha @ 0x0200;  /* Place ‘alpha' at address 0x200 */
#pragma location = 0x0202
const int beta;
```

If absolute data placement is needed, this can be achieved with entries into the linker command file, and then declaring the variables as extern in the C code:

```
/* CCE Linker Command File Entry */
alpha = 0x200;
beta = 0x202;
/* CCE C Code */
extern char alpha;
extern int beta;
```

The definitions of the peripheral register map in the linker command files (lnk_msp430xxxx.cmd) and the device specific header files (msp430xxxx.h) that are supplied with CCE are an example of placing data at absolute locations.

C.3.2 Data Placement Into Named Segments

In IAR, it is possible to place variables into named segments using either the @ operator or a #pragma directive:

```
/* IAR C Code */
__no_init int alpha @ "MYSEGMENT";   /* Place ‘alpha' into ‘MYSEGMENT' */
#pragma location="MYSEGMENT"         /* Place ‘beta' into ‘MYSEGMENT' */
const int beta;
```

With the CCE compiler, the #pragma DATA_SECTION() directive must be used:

```
/* CCE C Code */
#pragma DATA_SECTION(alpha, "MYSEGMENT")
int alpha;
#pragma DATA_SECTION(beta, "MYSEGMENT")
const int beta;
```

See Section C.5.3 for information on how to translate memory segment names between IAR and CCE.

C.3.3 Function Placement Into Named Segments

With the IAR compiler, functions can be placed into a named segment using the @ operator or the #pragma location directive:

```
/* IAR C Code */
void g(void) @ "MYSEGMENT"
{
}
#pragma location="MYSEGMENT"
void h(void)
{
}
```

With the CCE compiler, the following scheme with the #pragma CODE_SECTION() directive must be used:

```
/* CCE C Code */
#pragma CODE_SECTION(g, "MYSEGMENT")
void g(void)
{
}
```

See Section C.5.3 for information on how to translate memory segment names between IAR and CCE.

C.4 C Calling Conventions

The CCE and IAR C-compilers use different calling conventions for passing parameters to functions. When porting a mixed C and assembly project to the TI CCE code generation tools, the assembly functions need to be modified to reflect these changes. For detailed information about the calling conventions, see the *TI MSP430 Optimizing C/C++ Compiler User's Guide* (TI literature number SLAU132) and the *IAR MSP430 C/C++ Compiler Reference Guide*.

The following example is a function that writes the 32-bit word 'Data' to a given memory location in big-endian byte order. It can be seen that the parameter ‘Data' is passed using different CPU registers.

IAR Version:

```
;-------------------------------------------------------------------------
; void WriteDWBE(unsigned char *Add, unsigned long Data)
;
; Writes a DWORD to the given memory location in big-endian format. The
; memory address MUST be word-aligned.
;
; IN:  R12          Address       (Add)
;      R14          Lower Word    (Data)
;      R15          Upper Word    (Data)
;-------------------------------------------------------------------------
WriteDWBE
            swpb    R14                         ; Swap bytes in lower word
            swpb    R15                         ; Swap bytes in upper word
            mov.w   R15,0(R12)                  ; Write 1st word to memory
            mov.w   R14,2(R12)                  ; Write 2nd word to memory
            ret
```

CCE Version:

```
;-------------------------------------------------------------------------
; void WriteDWBE(unsigned char *Add, unsigned long Data)
;
; Writes a DWORD to the given memory location in big-endian format. The
; memory address MUST be word-aligned.
;
; IN:  R12          Address       (Add)
;      R13          Lower Word    (Data)
;      R14          Upper Word    (Data)
;-------------------------------------------------------------------------
WriteDWBE
            swpb    R13                         ; Swap bytes in lower word
            swpb    R14                         ; Swap bytes in upper word
            mov.w   R14,0(R12)                  ; Write 1st word to memory
            mov.w   R13,2(R12)                  ; Write 2nd word to memory
            ret
```

C.5 Other Differences

C.5.1 Initializing Static and Global Variables

The ANSI/ISO C standard specifies that static and global (extern) variables without explicit initializations must be pre-initialized to 0 (before the program begins running). This task is typically performed when the program is loaded and is implemented in the IAR compiler:

```
/* IAR, global variable, initialized to 0 upon program start */
int Counter;
```

However, the TI CCE compiler does not pre-initialize these variables; therefore, it is up to the application to fulfill this requirement:

```
/* CCE, global variable, manually zero-initilized */
int Counter = 0;
```

C.5.2 Custom Boot Routine

With the IAR compiler, the C startup function can be customized, giving the application a chance to perform early initializations such as configuring peripherals, or omit data segment initialization. This is achieved by providing a customized __low_level_init() function:

```
/* IAR C Code */
int __low_level_init(void)
{ =
  /* Insert your low-level initializations here */

  /*=================================*/
  /* Choose if segment initialization */
  /* should be done or not.        */
  /* Return: 0 to omit initialization    */
  /*           1 to run initialization  */
  /*=================================*/
  return (1);
}
```

The return value controls whether or not data segments are initialized by the C startup code. With the CCE C compiler, the custom boot routine name is _system_pre_init(). It is used the same way as in the IAR compiler.

```
/* CCE C Code */
int _system_pre_init(void)
{
  /* Insert your low-level initializations here */

  /*=================================*/
  /* Choose if segment initialization */
  /* should be done or not.        */
  /* Return: 0 to omit initialization    */
  /*           1 to run initialization  */
  /*=================================*/
  return (1);
}
```

Note that omitting segment initialization with both compilers omits both explicit and non-explicit initialization. The user must ensure that important variables are initialized at run time before they are used.

C.5.3 Predefined Memory Segment Names

Memory segment names for data and function placement are controlled by device-specific linker command files in both CCE and IAR tools. However, different segment names are used. See the linker command files for more detailed information. The following table shows how to convert the most commonly used segment names.

Description	CCE Segment Name	IAR Segment Name
RAM	.bss	DATA16_N DATA16_I DATA16_Z
Stack (RAM)	.stack	CSTACK
Main memory (Flash or ROM)	.text	CODE
Information memory (Flash or ROM)	.infoA .infoB	INFOA INFOB INFO
Interrupt vectors (Flash or ROM)	.int00 .int01 … .int14	INTVEC
Reset vector (Flash or ROM)	.reset	RESET

C.5.4 Predefined Macro Names

Both IAR and CCE compiler support a few non ANSI/ISO standard predefined macro names which help creating code that can be compiled and used on different compiler platforms. Check if a macro name is defined using the #ifdef directive.

Description	CCE Macro Name	IAR Macro Name
Is MSP430 the target and is a particular compiler platform used?	__MSP430__	__ICC430__
Is a particular compiler platform used?	__TI_COMPILER_VERSION__	__IAR_SYSTEMS_ICC__
Is a C header file included from within assembly source code?	__ASM_HEADER__	__ IAR_SYSTEMS_ASM__

The following code example shows how an MSP430 interrupt vector can be assigned in a way that can be used with both the CCE and IAR compilers.

```
/* ADC10 interrupt service routine example */

#ifdef __MSP430__               /* Is CCE MSP430 compiler used? */
ADC10_ISR(ADC_ISR)              /* Yes, use CCE interrupt vector scheme */
#endif
#ifdef __ICC430__               /* Is the IAR MSP430 compiler used? */
#pragma vector = ADC10_VECTOR   /* Yes, use the IAR int. vector scheme */
#endif
__interrupt void ADC_ISR(void)  /* Example ISR body */
{
  Result = ADC10MEM;            /* Do some processing */
}
```

IAR 2.x/3.x to CCE Assembler Migration

Source for the TI CCE assembler and source code for the IAR assembler are not 100% compatible. The instruction mnemonics are identical, while the assembler directives are somewhat different. This appendix documents the differences between the CCE assembler directives and the IAR 2.x/3.x assembler directives.

Topic **Page**

D.1 Sharing C/C++ Header Files With Assembly Source

The IAR A430 assembler supports certain C/C++ preprocessor directives directly, and thereby allows to directly include C/C++ header files such as the MSP430 device-specific header files msp430xxxx.h into the assembly code:

```
#include "msp430x14x.h"  // Include device header file
```

With the CCE Asm430 assembler, a different scheme which is using the .cdecls directive must be used. This directive allows programmers in mixed assembly and C/C++ environments to share C/C++ headers containing declarations and prototypes between the C/C++ and assembly code:

```
.cdecls C,LIST,"msp430x14x.h"  ; Include device header file
```

More information on the .cdecls directive can be found in the *MSP430 Assembly Language Tools User's Guide* (TI literature number SLAU131).

D.2 Segment Control

The CCE Asm430 assembler does not support any of the IAR A430 segment control directives like ORG, ASEG, RSEG, and COMMON.

Description	Asm430 Directive (CCE)
Reserve space in the .bss uninitialized section	.bss
Reserve space in a named uninitialized section	.usect
Allocate program into the default program section (initialized)	.text
Allocate data into a named initialized section	.sect

To allocate code and data sections to specific addresses with the CCE assembler, it is necessary to create/use memory sections defined in the linker command files. The following example demonstrates interrupt vector assignment in both IAR and CCE assembly to highlight the differences.

```
;-------------------------------------------------------------------------
;           Interrupt Vectors Used MSP430x11x1/12x(2) – IAR Assembler
;-------------------------------------------------------------------------
              ORG     0FFFEh                  ; MSP430 RESET Vector
            DW      RESET                   ;
            ORG     0FFF2h                  ; Timer_A0 Vector
            DW      TA0_ISR                 ;

;-------------------------------------------------------------------------
;           Interrupt Vectors Used MSP430x11x1/12x(2) - CCE Assembler
;-------------------------------------------------------------------------
               .sect   ".reset"                    ; MSP430 RESET Vector
               .short  RESET                       ;
                  .sect   ".int09"                 ; Timer_A0  Vector
               .short  TA0_ISR                     ;
```

Both examples assume that the standard device support files (header files, linker command files) are used. Note that the linker command files are different between IAR and CCE and cannot be reused. See Section C.5.3 for information on how to translate memory segment names between IAR and CCE.

D.3 Translating A430 Assembler Directives to Asm430 Directives

D.3.1 Introduction

The following sections describe, in general, how to convert assembler directives for the IAR A430 assembler (A430) to Texas Instruments CCE Asm430 assembler (Asm430) directives. These sections are intended only as a guide for translation. For detailed descriptions of each directive, see either the *MSP430 Assembly Language Tools User's Guide* (TI literature number SLAU131), from Texas Instruments, or the *MSP430 IAR Assembler Reference Guide* from IAR.

Note: Only the assembler *directives* require conversion

Only the assembler directives require conversion, not the assembler instructions. Both assemblers use the same instruction mnemonics, operands, operators, and special symbols such as the section program counter ($) and the comment delimiter (;).

The A430 assembler is not case sensitive by default. These sections show the A430 directives written in uppercase to distinguish them from the Asm430 directives, which are shown in lower case.

D.3.2 Character Strings

In addition to using different directives, each assembler uses different syntax for character strings. A430 uses C syntax for character strings: A quote is represented using the backslash character as an escape character together with quote (\") and the backslash itself is represented by two consecutive backslashes (\\). In Asm430 syntax, a quote is represented by two consecutive quotes (""); see examples:

Character String	Asm430 Syntax (CCE)	A430 Syntax (IAR)
PLAN "C"	"PLAN ""C"""	"PLAN \"C\""
\dos\command.com	"\dos\command.com"	"\\dos\\command.com"
Concatenated string (i.e. Error 41)	-	"Error " "41"

D.3.3 Section Control Directives

Asm430 has three predefined sections into which various parts of a program are assembled. Uninitialized data is assembled into the .bss section, initialized data into the .data section, and executable code into the .text section.

A430 also uses sections or segments, but there are no predefined segment names. Often, it is convenient to adhere to the names used by the C compiler: DATA16_Z for uninitialized data, CONST for constant (initialized) data and CODE for executable code. The following table uses these names.

A pair of segments can be used to make initialized, modifiable data PROM-able. The ROM segment would contain the initializers and would be copied to RAM segment by a start-up routine. In this case, the segments must be exactly the same size and layout.

Description	Asm430 Directive (CCE)	A430 Directive (IAR)
Reserve size bytes in the .bss (uninitialized data) section	.bss(1)	(2)
Assemble into the .data (initialized data) section	.data	RSEG const
Assemble into a named (initialized) section	.sect	RSEG
Assemble into the .text (executable code) section	.text	RSEG code
Reserve space in a named (uninitialized) section	.usect(1)	(2)
Alignment on byte boundary	.align 1	(3)
Alignment on word boundary	.align 2	EVEN

(1) .bss and .usect do not require switching back and forth between the original and the uninitialized section. For example:

```
; IAR Assembler Example
             RSEG   DATA16_N       ; Switch to DATA segment
             EVEN                  ; Ensure proper alignment
ADCResult:   DS     2              ; Allocate 1 word in RAM
Flags:       DS     1              ; Allocate 1 byte in RAM
             RSEG   CODE           ; Switch back to CODE segment
; CCE Assembler Example #1
ADCResult   .usect  ".bss",2,2     ; Allocate 1 word in RAM
Flags       .usect  ".bss",1       ; Allocate 1 byte in RAM
; CCE Assembler Example #2
            .bss    ADCResult,2,2 ; Allocate 1 word in RAM
            .bss    Flags,1       ; Allocate 1 byte in RAM
```

(2) Space is reserved in an uninitialized segment by first switching to that segment, then defining the appropriate memory block, and then switching back to the original segment. For example:

```
            RSEG    DATA16_Z
LABEL:      DS      16            ; Reserve 16 byte
            RSEG    CODE
```

(3) Initialization of bit-field constants (.field) is not supported, therefore, the section counter is always byte-aligned.

D.3.4 Constant Initialization Directives

Description	Asm430 Directive (CCE)	A430 Directive (IAR)
Initialize one or more successive bytes or text strings	.byte or .string	DB
Initialize a 32-bit IEEE floating-point constant	.double or .float	DF
Initialize a variable-length field	.field	(1)
Reserve size bytes in the current section	.space	DS
Initialize one or more text strings	Initialize one or more text strings	DB
Initialize one or more 16-bit integers	.word	DW
Initialize one or more 32-bit integers	.long	DL

(1) Initialization of bit-field constants (.field) is not supported. Constants must be combined into complete words using DW.

```
; Asm430 code                    ; A430 code
.field 5,3    \
.field 12,4   | ->               DW (30<<(4+3))|(12<<3)|5   ; equals 3941
.field 30,8   /
```

D.3.5 Listing Control Directives

Description	Asm430 Directive (CCE)	A430 Directive (IAR)
Allow false conditional code block listing	.fclist	LSTCND-
Inhibit false conditional code block listing	.fcnolist	LSTCND+
Set the page length of the source listing	.length	PAGSIZ
Set the page width of the source listing	.width	COL
Restart the source listing	.list	LSTOUT+
Stop the source listing	.nolist	LSTOUT-
Allow macro listings and loop blocks	.mlist	LSTEXP+ (macro) LSTREP+ (loop blocks)
Inhibit macro listings and loop blocks	.mnolist	LSTEXP- (macro) LSTREP- (loop blocks)
Select output listing options	.option	(1)
Eject a page in the source listing	.page	PAGE
Allow expanded substitution symbol listing	.sslist	(2)
Inhibit expanded substitution symbol listing	.ssnolist	(2)
Print a title in the listing page header	.title	(3)

(1) No A430 directive directly corresponds to .option. The individual listing control directives (above) or the command-line option -c (with suboptions) should be used to replace the .option directive.
(2) There is no directive that directly corresponds to .sslist/.ssnolist.
(3) The title in the listing page header is the source file name.

D.3.6 File Reference Directives

Description	Asm430 Directive (CCE)	A430 Directive (IAR)
Include source statements from another file	.copy or .include	#include or $
Identify one or more symbols that are defined in the current module and used in other modules	.def	PUBLIC or EXPORT
Identify one or more global (external) symbols	.global	(1)
Define a macro library	.mlib	(2)
Identify one or more symbols that are used in the current module but defined in another module	.ref	EXTERN or IMPORT

(1) The directive .global functions as either .def if the symbol is defined in the current module, or .ref otherwise. PUBLIC or EXTERN must be used as applicable with the A430 assembler to replace the .global directive.
(2) The concept of macro libraries is not supported. Include files with macro definitions must be used for this functionality.

Modules may be used with the Asm430 assembler to create individually linkable routines. A file may contain multiple modules or routines. All symbols except those created by DEFINE, #define (IAR preprocessor directive) or MACRO are "undefined" at module end. Library modules are, furthermore, linked conditionally. This means that a library module is included in the linked executable only if a public symbol in the module is referenced externally. The following directives are used to mark the beginning and end of modules in the A430 assembler.

Additional A430 Directives (IAR)	A430 Directive (IAR)
Start a program module	NAME or PROGRAM
Start a library module	MODULE or LIBRARY
Terminate the current program or library module	ENDMOD

D.3.7 Conditional Assembly Directives

Description	Asm430 Directive (CCE)	A430 Directive (IAR)
Optional repeatable block assembly	.break	(1)
Begin conditional assembly	.if	IF
Optional conditional assembly	.else	ELSE
Optional conditional assembly	.elseif	ELSEIF
End conditional assembly	.endif	ENDIF
End repeatable block assembly	.endloop	ENDR
Begin repeatable block assembly	.loop	REPT

(1) There is no directive that directly corresponds to .break. However, the EXITM directive can be used with other conditionals if repeatable block assembly is used in a macro, as shown:

```
SEQ     MACRO   FROM,TO      ; Initialize a sequence of byte constants
        LOCAL   X
X       SET     FROM
        REPT    TO-FROM+1    ; Repeat from FROM to TO
        IF      X>255        ; Break if X exceeds 255
        EXITM
        ENDIF
        DB      X            ; Initialize bytes to FROM...TO
X       SET     X+1          ; Increment counter
        ENDR
        ENDM
```

D.3.8 Symbol Control Directives

The scope of assembly-time symbols differs in the two assemblers. In Asm430, definitions can be global to a file or local to a module or macro. Local symbols can be undefined with the .newblock directive. In A430, symbols are either local to a macro (LOCAL), local to a module (EQU) or global to a file (DEFINE). In addition, the preprocessor directive #define can also be used to define local symbols.

Description	Asm430 Directive (CCE)	A430 Directive (IAR)
Assign a character string to a substitution symbol	.asg	SET or VAR or ASSIGN
Undefine local symbols	.newblock	(1)
Equate a value with a symbol	.equ or .set	EQU or =
Perform arithmetic on numeric substitution symbols	.eval	SET or VAR or ASSIGN
End structure definition	.endstruct	(2)
Begin a structure definition	.struct	(2)
Assign structure attributes to a label	.tag	(2)

(1) No A430 directive directly corresponds to .newblock. However, #undef may be used to reset a symbol that was defined with the #define directive. Also, macros or modules may be used to achieve the .newblock functionality because local symbols are implicitly undefined at the end of a macro or module.

(2) Definition of structure types is not supported. Similar functionality is achieved by using macros to allocate aggregate data and base address plus symbolic offset, as shown:

```
MYSTRUCT: MACRO
            DS 4
            ENDM
LO          DEFINE  0
HI          DEFINE  2
            RSEG    DATA16_Z
X           MYSTRUCT
            RSEG    CODE
            MOV     X+LO,R4
            ...
```

D.3.9 Macro Directives

Description	Asm430 Directive (CCE)	A430 Directive (IAR)
Define a macro	.macro	MACRO
Exit prematurely from a macro	.mexit	EXITM
End macro definition	.endm	ENDM

D.3.10 Miscellaneous Directives

Description	Asm430 Directive (CCE)	A430 Directive (IAR)
Send user-defined error messages to the output device	.emsg	#error
Send user-defined messages to the output device	.mmsg	#message(1)
Send user-defined warning messages to the output device	.wmsg	(2)
Define a load address label	.label	(3)
Directive produced by absolute lister	.setsect	ASEG(4)
Directive produced by absolute lister	.setsym	EQU or =(4)
Program end	.end	END

(1) The syntax of the #message directive is: #message "<string>"
This causes '#message <string>' to be output to the project build window during assemble/compile time.

(2) Warning messages cannot be user-defined. #message may be used, but the warning counter is not incremented.

(3) The concept of load-time addresses is not supported. Run-time and load-time addresses are assumed to be the same. To achieve the same effect, labels can be given absolute (run-time) addresses by the EQU directives.

```
; Asm430 code           ; A430 code
.label  load_start      load_start:
Run_start:                  <code>
   <code>               load_end:
Run_end:                run_start: EQU 240H
.label  load_end        run_end:   EQU run_start+load_end-load_start
```

(4) Although not produced by the absolute lister ASEG defines absolute segments and EQU can be used to define absolute symbols.

```
MYFLAG    EQU   23EH       ; MYFLAG is located at 23E
          ASEG  240H       ; Absolute segment at 240
MAIN:     MOV   #23CH, SP  ; MAIN is located at 240
...
```

D.3.11 Alphabetical Listing and Cross Reference of Asm430 Directives

Asm430 Directive (CCE)	A430 Directive (IAR)	Asm430 Directive (CCE)	A430 Directive (IAR)
.align	ALIGN	.loop	REPT
.asg	SET or VAR or ASSIGN	.macro	MACRO
.break	See Conditional Assembly Directives	.mexit	EXITM
.bss	See Symbol Control Directives	.mlib	See File Referencing Directives
.byte or .string	DB	.mlist	LSTEXP+ (macro)
.cdecls	C pre-processor declarations are inherently supported.		LSTREP+ (loop blocks)
.copy or .include	#include or $	.mmsg	#message (XXXXXX)
.data	RSEG	.mnolist	LSTEXP- (macro)
.def	PUBLIC or EXPORT		LSTREP- (loop blocks)
.double	Not supported	.newblock	See Symbol Control Directives
.else	ELSE	.nolist	LSTOUT-
.elseif	ELSEIF	.option	See Listing Control Directives
.emsg	#error	.page	PAGE
.end	END	.ref	EXTERN or IMPORT
.endif	ENDIF	.sect	RSEG
.endloop	ENDR	.setsect	See Miscellaneous Directives
.endm	ENDM	.setsym	See Miscellaneous Directives
.endstruct	See Symbol Control Directives	.space	DS
.equ or .set	EQU or =	.sslist	Not supported
.eval	SET or VAR or ASSIGN	.ssnolist	Not supported
.even	EVEN	.string	DB
.fclist	LSTCND-	.struct	See Symbol Control Directives
.fcnolist	LSTCND+	.tag	See Symbol Control Directives
.field	See Constant Initialization Directives	.text	RSEG
.float	See Constant Initialization Directives	.title	See Listing Control Directives
.global	See File Referencing Directives	.usect	See Symbol Control Directives
.if	IF	.width	COL
.label	See Miscellaneous Directives	.wmsg	See Miscellaneous Directives
.length	PAGSIZ	.word	DW
.list	LSTOUT+		

D.3.12 Unsupported A430 Directives (IAR)

The following IAR assembler directives are not supported in the CCE Asm430 assembler:

Conditional Assembly Directives	**Macro Directives**	
REPTC[1]	LOCAL[2]	
REPTI		
File Referencing Directives	**Miscellaneous Directives**	**Symbol Control Directives**
NAME or PROGRAM	RADIX	DEFINE
MODULE or LIBRARY	CASEON	SFRB
ENDMOD	CASEOFF	SFRW
Listing Control Directives	**C-Style Preprocessor Directives[3]**	**Symbol Control Directives**
LSTMAC (+/-)	#define	ASEG
LSTCOD (+/-)	#undef	RSEG
LSTPAG (+/-)	#if, #else, #elif	COMMON
LSTXREF (+/-)	#ifdef, #ifndef	STACK
	#endif	ORG
	#include	
	#error	

(1) There is no direct support for IAR REPTC/REPTI directives in CCE. However, equivalent functionality can be achieved using the CCE .macro directive:

```
; IAR Assembler Example
            REPTI   zero,"R4","R5","R6"
            MOV     #0,zero
            ENDR
; CCE Assembler Example
zero_regs .macro list
            .var item
            .loop
            .break ($ismember(item, list) = 0)
            MOV #0,item
            .endloop
            .endm
```

Code that is generated by calling "zero_regs R4,R5,R6":

```
            MOV #0,R4
            MOV #0,R5
            MOV #0,R6
```

(2) In CCE, local labels are defined by using $n (with n=0…9) or with NAME?. Examples are $4, $7, or Test?.

(3) The use of C-style preprocessor directives is supported indirectly through the use of .cdecls. More information on the .cdecls directive can be found in the *MSP430 Assembly Language Tools User's Guide* (TI literature number SLAU131).

FET-Specific Menus

This appendix describes the CCE menus that are specific to the FET.

Topic **Page**

E.1 Menus

E.1.1 RUN → RELEASE JTAG ON RUN

GDB430 uses the device JTAG signals to debug the device. On some MSP430 devices, these JTAG signals are shared with the device port pins. Normally, GDB430 maintains the pins in JTAG mode so that the device can be debugged. During this time, the port functionality of the shared pins is not available.

However, when RELEASE JTAG ON RUN is selected, the JTAG drivers are set to 3-state and the device is released from JTAG control (TEST pin is set to GND) when GO is activated. Any active on-chip breakpoints are retained and the shared JTAG port pins revert to their port functions.

At this time, GDB430 has no access to the device and cannot determine if an active breakpoint (if any) has been reached. GDB430 must be manually commanded to stop the device, at which time the state of the device is determined (i.e., was a breakpoint reached?).

See FAQ Debugging #9.

E.1.2 RUN → RESYNCHRONIZE JTAG

Regains control of the device.

It is not possible to RESYNCHRONIZE JTAG while the device is operating.

E.1.3 RUN → MAKE DEVICE SECURE

Blows the fuse on the target device. After the fuse is blown, no further communication via JTAG with the device is possible.

E.1.4 RUN → CLOCK CONTROL

Disables the specified system clock while GDB430 has control of the device (following a STOP or breakpoint). All system clocks are enabled following a GO or a single step (STEP/STEP INTO). See FAQ Debugging #12.

E.1.5 WINDOW → SHOW VIEW → OTHER → MSP430 BREAKPOINTS → ADVANCED MSP430 BREAKPOINTS

Opens the Advanced MSP430 Breakpoints Window. In the window it is possible to set hardware breakpoints (Conditional Trigger and Register Trigger). The window permits per drag and drop to combine the breakpoints. A combined breakpoint is triggered when all breakpoints are triggered.

E.1.6 WINDOW → SHOW VIEW → OTHER → MSP430 STATE STORAGE → MSP 430 STATE STORAGE BUFFER VIEW

The State Storage Window permits one to use the state storage module. The state storage module is present only in those devices that contain the EEM.

Open the State Storage window, and display the stored state information as configured by the State Storage dialog.

E.1.7 PROJECT → PROPERTIES → DEBUG PROPERTIES → TARGET VOLTAGE

On the USB FET the target supply voltage can be adjusted between 1.8 V and 3.6 V. This voltage is available on pin 2 of the 14-pin target connector to supply the target from the USB FET. If the target is supplied externally, the external supply voltage should be connected to pin 4 of the target connector, so the USB FET can set the level of the output signals accordingly.

Hardware Installation Guide

This section describes the hardware installation process of the following USB debug interfaces on a PC running Windows XP:

- MSP-FET430UIF
- MSP-eZ430-F2013
- MSP-eZ430-RF2500

The installation procedure for a Windows 2000 system is very similar and, therefore, not shown here.

Topic **Page**

F.1 Hardware Installation

1. Connect the USB Debug Interface with a USB cable to a USB port of your PC. (MSP-eZ430-F2013 and MSP-eZ430-RF2500 can be connected without a cable.)
2. Windows should now recognize the new hardware as an "MSP430 XXX x.xx.xx" (see Figure F-1). The device name may be different from the one shown here.

Figure F-1. WinXP Hardware Recognition

3. The Hardware Wizard starts automatically and opens the "Found New Hardware Wizard" window.
4. Click "Next". The Hardware Wizards try to find the driver in the system. If the driver was found - please continue with step 8. If not - press "Back" and continue with step 5.
5. Select "Install from a list or specific location (Advanced)" (see Figure F-2).

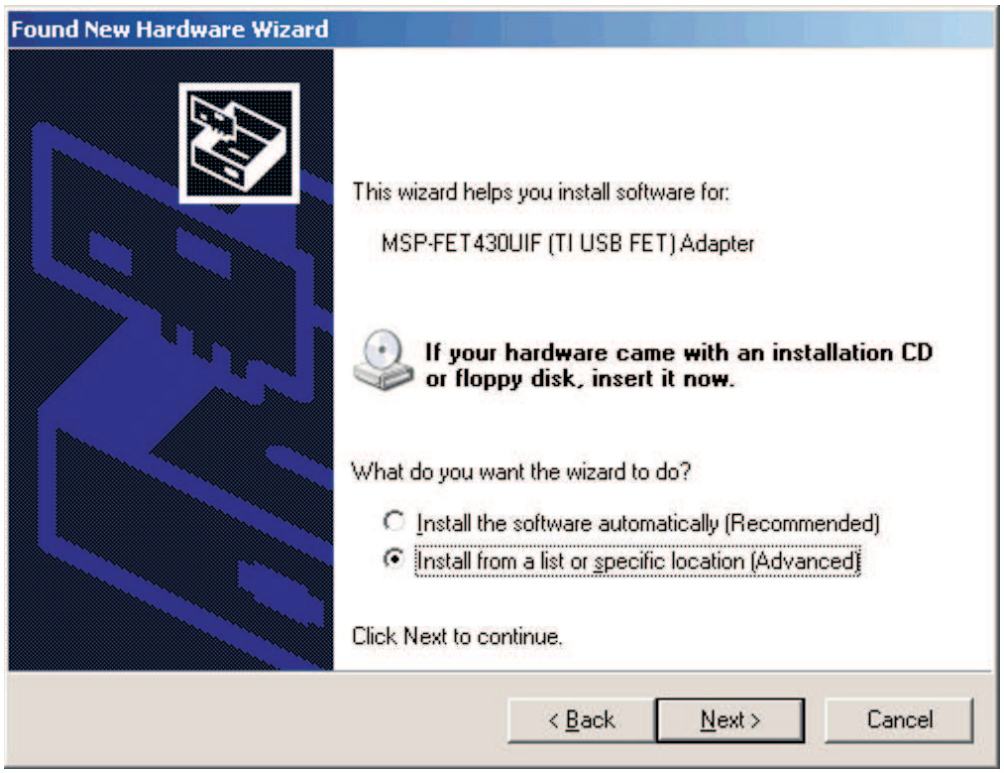


Figure F-2. WinXP Hardware Wizard

6. Browse to the folder where the driver information files are located (see Figure F-3). On a default installation the files are located in the following directory: "C:\Program Files\Texas Instruments\CC Essentials 2.0_FET\win32_drivers\MSP430_USB\program files\Texas Instruments\TI3410XP\"

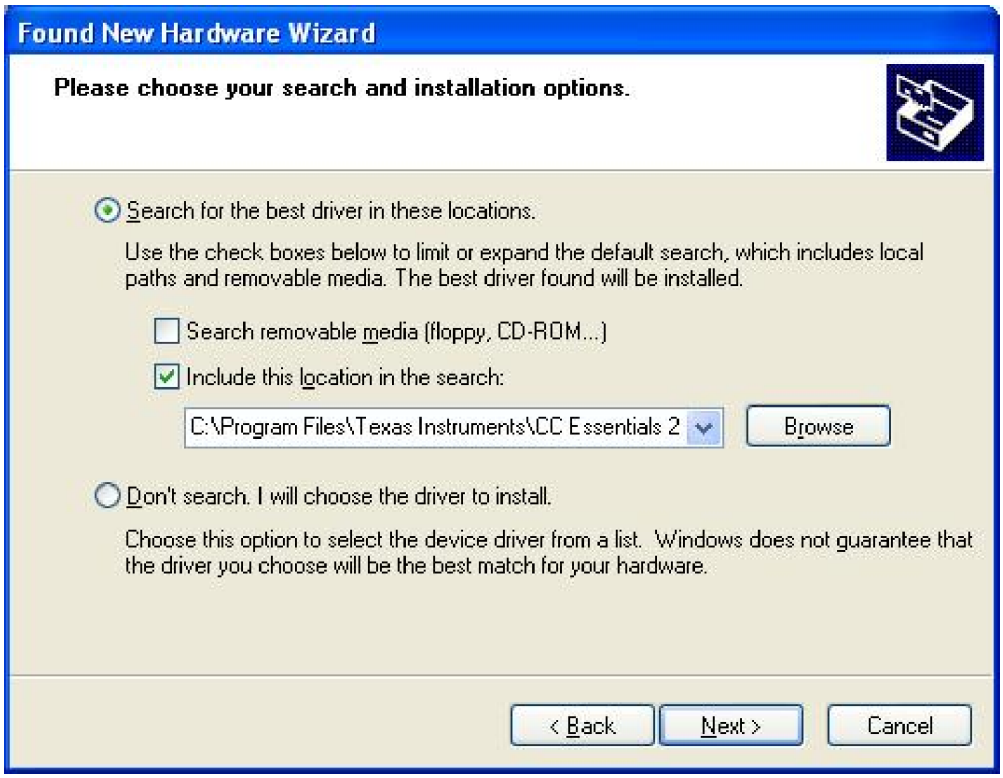


Figure F-3. WinXP Driver Location Selection Folder

7. The Wizard generates a message that an appropriate driver has been found.

8. Note that WinXP shows a warning that the driver is not certified by Microsoft®. Ignore this warning and click "Continue Anyway" (see Figure F-4).

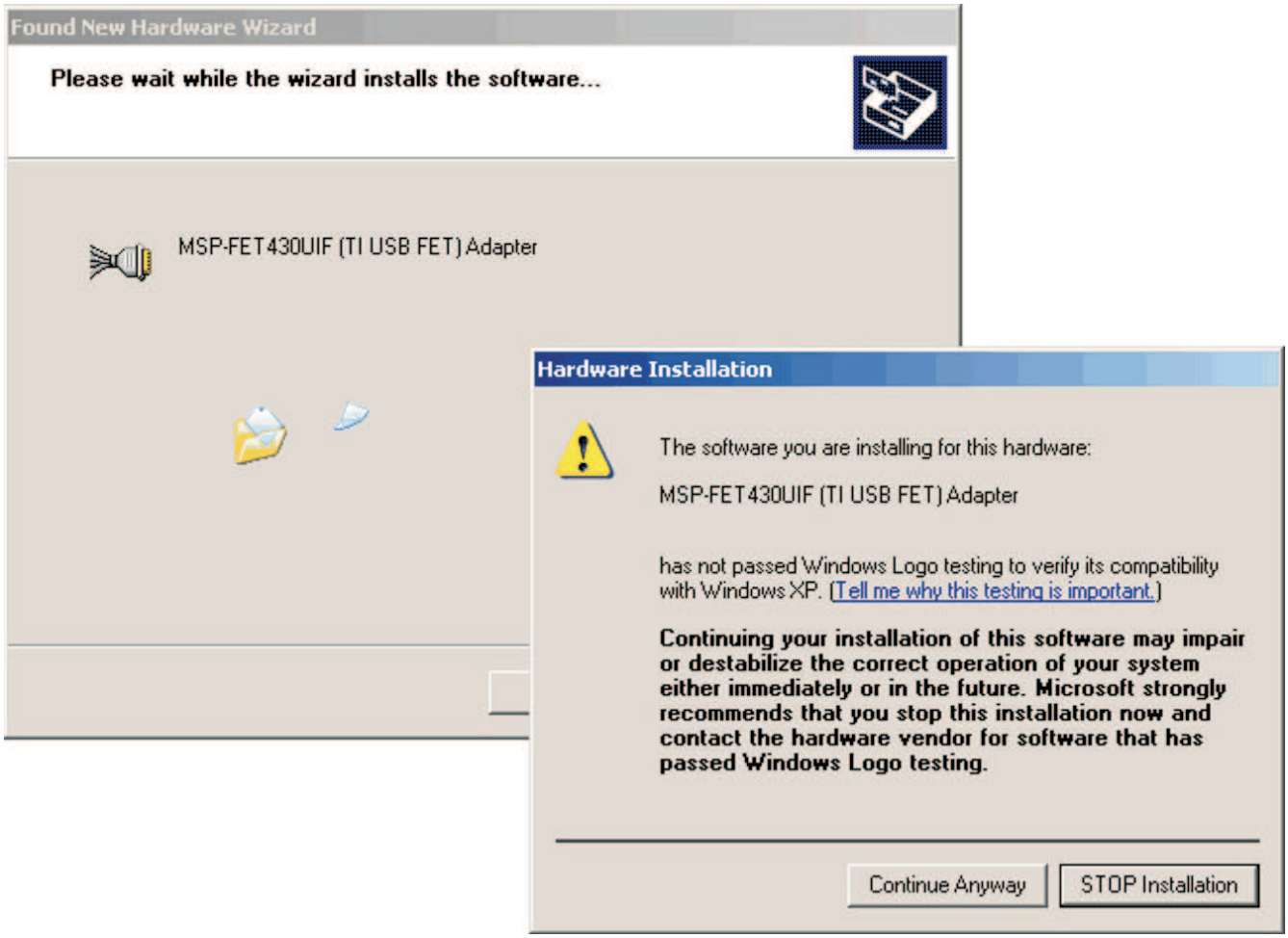


Figure F-4. WinXP Driver Installation

9. The wizard installs the driver files.
10. The wizard shows a message that it has finished the installation of the software for "MSP-FET430UIF (TI USB FET) Adapter" (or "MSP430 Application UART").
11. **This step is only for MSP-FET430UIF and MSP-eZ430-F2013.** After closing the hardware wizard, Windows automatically recognizes another new hardware device called "MSP-FET430UIF - Serial Port".
12. **This step is for MSP-FET430UIF and MSP-eZ430-F2013 only.** Depending on the current update version of the operating system, corresponding drivers are installed automatically or the hardware wizard opens again. **If the wizard starts again, repeat the steps described above.**

13. The USB debug interface is installed and ready to use. The Device Manager lists a new entry as shown in Figure F-5 or Figure F-6.

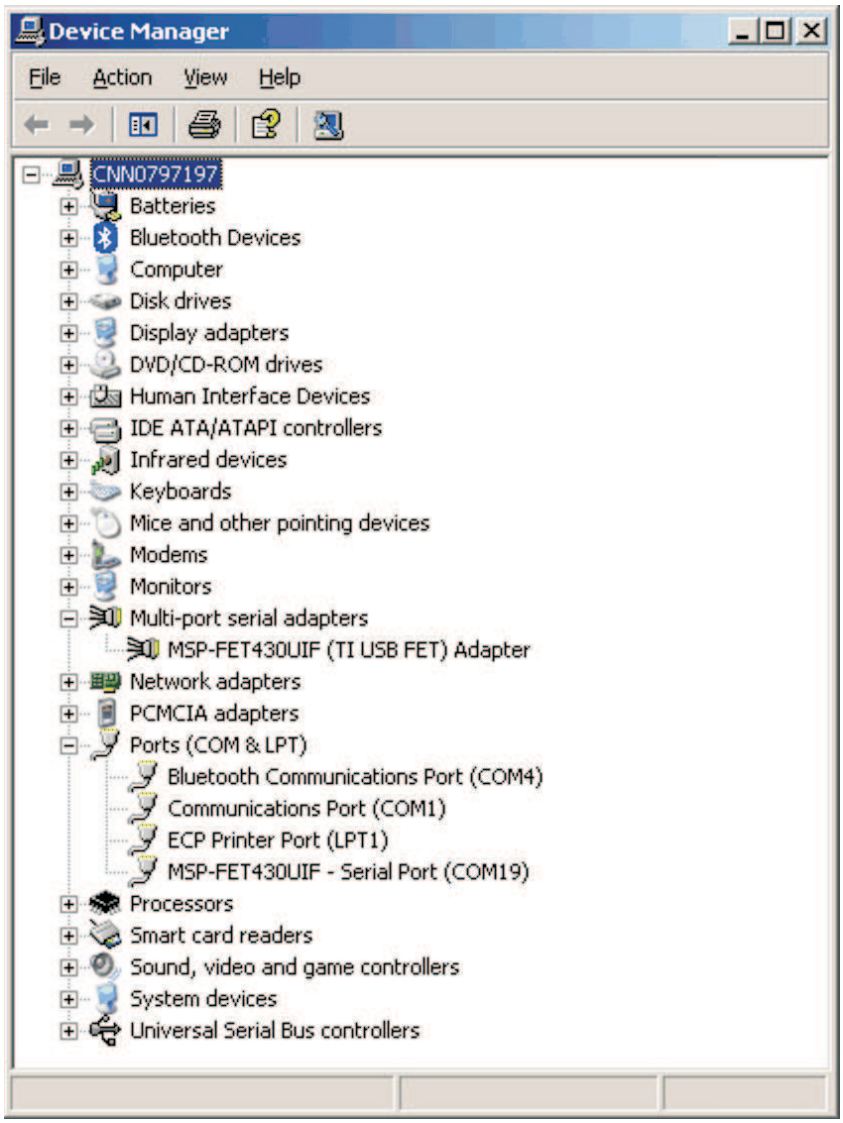


Figure F-5. Device Manager Using MSP-FET430UIF or MSP-eZ430-F2013

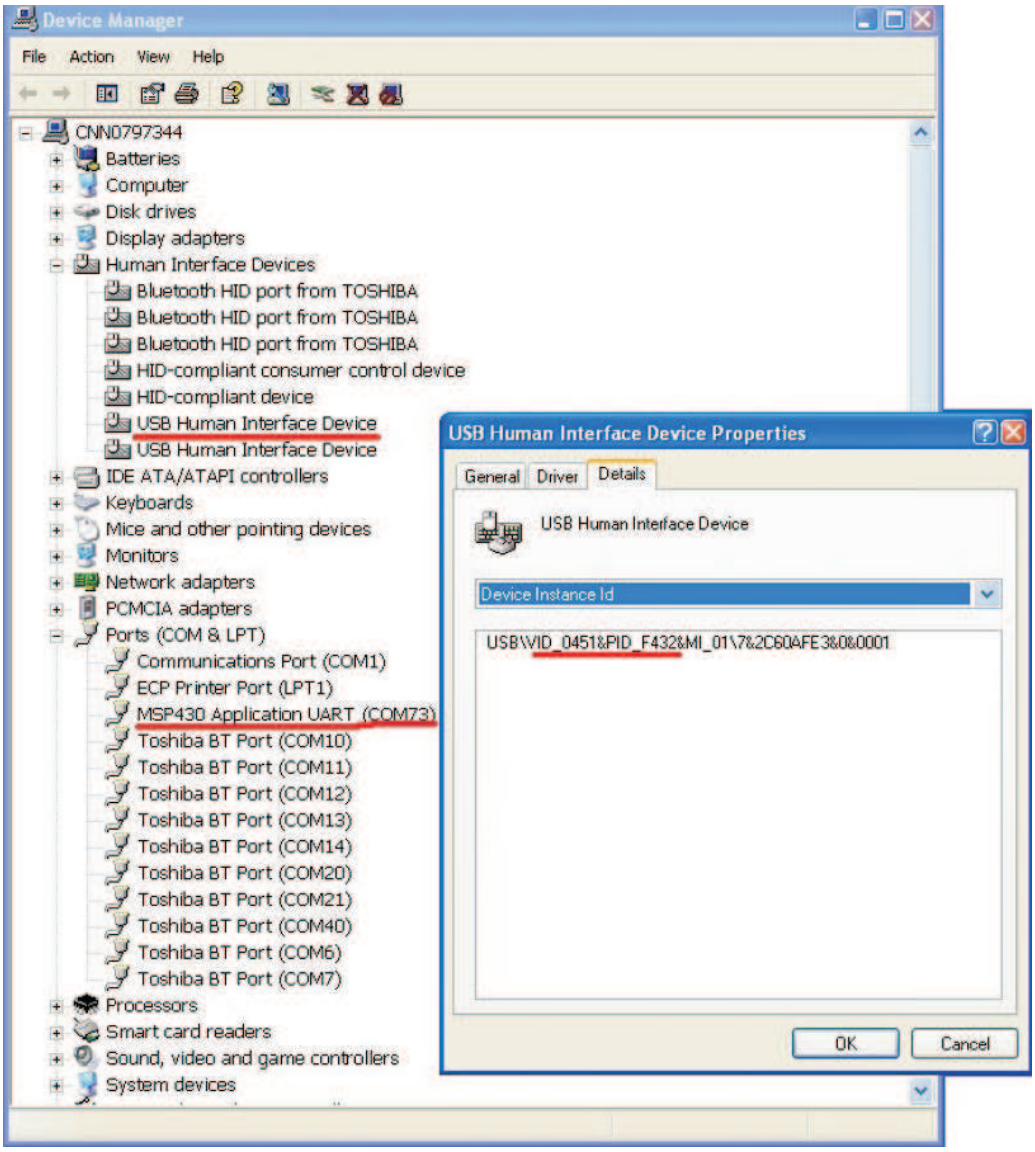


Figure F-6. Device Manager Using MSP-eZ430-RF2500

Document Revision History

Version	Changes/Comments
SLAU157E	Added MSP-TS430PW28 Target Socket Module, Schematic (Figure B-5) and PCB (Figure B-6). Updated MSP-FET430U28 Kit content information (DW or PW package support) in Section 1.3. Added Emulation features for MSP430F21x2 to Table 2-1. Updated MSP-TS430PW14 Target Socket Module, Schematic (Figure B-1). Updated MSP-TS430DA38 Target Socket Module, Schematic (Figure B-7).
SLAU157D	Added Section 1.7 Updated Table 2-1 Updated Appendix F
SLAU157C	Updated Appendix F Added emulation features for MSP430F22x2, MSP430F241x, MSP430F261x, MSP430FG42x0 and MSP430F43x in Table 2-1
SLAU157B	Renamed MSP-FET430U40 to MSP-FET430U23x0 Replaced MSP-FET430U40 schematic and PCB figures with renamed MSP-FET430U23x0 figures Added FAQ Hardware #2 in Section A.1 Added FAQ Debugging #4 in Section A.3

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

EVALUATION BOARD/KIT IMPORTANT NOTICE

Texas Instruments (TI) provides the enclosed product(s) under the following conditions:

This evaluation board/kit is intended for use for **ENGINEERING DEVELOPMENT, DEMONSTRATION, OR EVALUATION PURPOSES ONLY** and is not considered by TI to be a finished end-product fit for general consumer use. Persons handling the product(s) must have electronics training and observe good engineering practice standards. As such, the goods being provided are not intended to be complete in terms of required design-, marketing-, and/or manufacturing-related protective considerations, including product safety and environmental measures typically found in end products that incorporate such semiconductor components or circuit boards. This evaluation board/kit does not fall within the scope of the European Union directives regarding electromagnetic compatibility, restricted substances (RoHS), recycling (WEEE), FCC, CE or UL, and therefore may not meet the technical requirements of these directives or other related directives.

Should this evaluation board/kit not meet the specifications indicated in the User's Guide, the board/kit may be returned within 30 days from the date of delivery for a full refund. THE FOREGOING WARRANTY IS THE EXCLUSIVE WARRANTY MADE BY SELLER TO BUYER AND IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

The user assumes all responsibility and liability for proper and safe handling of the goods. Further, the user indemnifies TI from all claims arising from the handling or use of the goods. Due to the open construction of the product, it is the user's responsibility to take any and all appropriate precautions with regard to electrostatic discharge.

EXCEPT TO THE EXTENT OF THE INDEMNITY SET FORTH ABOVE, NEITHER PARTY SHALL BE LIABLE TO THE OTHER FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES.

TI currently deals with a variety of customers for products, and therefore our arrangement with the user **is not exclusive.**

TI assumes **no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein.**

Please read the User's Guide and, specifically, the Warnings and Restrictions notice in the User's Guide prior to handling the product. This notice contains important safety information about temperatures and voltages. For additional information on TI's environmental and/or safety programs, please contact the TI application engineer or visit www.ti.com/esh.

No license is granted under any patent right or other intellectual property right of TI covering or relating to any machine, process, or combination in which such TI products or services might be or are used.

FCC Warning

This evaluation board/kit is intended for use for **ENGINEERING DEVELOPMENT, DEMONSTRATION, OR EVALUATION PURPOSES ONLY** and is not considered by TI to be a finished end-product fit for general consumer use. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures may be required to correct this interference.

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 TI 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. Information of third parties may be subject to additional restrictions.

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.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

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
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
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
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

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