Texas Instruments MSP-FET430 Flash Emulation Tool (FET) User's Guide

Introduction

Thank you for purchasing a Texas Instruments MSP-FET430 Flash Emulation Tool (FET) for our MSP430 ultralow power microcontroller.

Please read and follow the Get Started Now! section below. This section will enable you to inventory your FET, and then it will instruct you to install the software and hardware, and then run the demonstration programs. Once you've demonstrated how quick and easy it is to use the FET, we suggest that you complete the reading of this User's Guide. The User's Guide describes the set-up and operation of the FET, but does not fully teach the MSP430 or the software systems (the workbench, the assembler, the C compiler, the linker, and the debugger [from IAR]). For details of these items, please refer to the appropriate TI and IAR documents listed in "Sources of Additional Information" within the Appendices.

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

- MSP-FET430X110 (for the MSP430F11xIDW, MSP430F11x1IDW, and MSP430F11x2IDW devices)
- MSP-FET430P120 (for the MSP430F12xIDW and MSP430F12x2IDW devices)
- MSP-FET430P140 (for the MSP430F13xIPM and MSP430F14xIPM devices)
- MSP-FET430P410 (for the MSP430F41xIPM devices)
- MSP-FET430P440 (for the MSP430F43xIPZ and MSP430F44xIPZ devices)

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

Get Started Now!

Kit Contents, MSP-FET430X110

1. One READ ME FIRST document.
2. One MSP430 CD-ROM.
3. One MSP-FET430X110 Flash Emulation Tool. This is the PCB on which is mounted a 20-pin ZIF socket for the MSP430F11xIDW, MSP430F11x1IDW, or MSP430F11x2IDW device. A 25-conductor cable originates from the FET.
4. One small box containing two MSP430F1121IDW devices.

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

1. **MSP-FET430P440 only**: One Product Disclaimer document.
2. One READ ME FIRST document.
3. One MSP430 CD-ROM.
4. One MSP-FETP430IF FET Interface module. This is the unit that has a 25-pin male D-Sub connector on one end of the case, and a 2x7 pin male connector on the other end of the case.
5. **MSP-FET430P120**: One MSP-TS430DW28 Target Socket module. This is the PCB on which is mounted a 28-pin ZIF socket for the MSP430F12xIPW or MSP43012x2IPW device. A 2x7 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 or MSP430F14xIPM device. A 2x7 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 device. A 2x7 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 2x7 pin male connector is also present on the PCB.
6. One 25-conductor cable.
7. One 14-conductor cable.
8. **MSP-FET430P120:** Four PCB 1x14 pin headers (Two male and two female).
   MSP-FET430P140: Eight PCB 1x16 pin headers (Four male and four female).
   MSP-FET430P410: Eight PCB 1x16 pin headers (Four male and four female).
   MSP-FET430P440: Eight PCB 1x25 pin headers (Four male and four female).
9. One small box containing two or four devices.
   MSP-FET430P120: MSP430F123IDW and/or MSP430F1232IDW
   MSP-FET430P140: MSP430F149IPM
   MSP-FET430P410: MSP430F413IPM
   MSP-FET430P440: PMS430F449IPZ
   PMS devices are preliminary devices.
   Please consult the data sheet for device specifications. A list of device errata can be found at
   http://www.ti.com/sc/cgi-bin/buglist.cgi

Software Installation

1. Follow the instructions on the supplied READ ME FIRST document to install the IAR Workbench (assembler and limited C project development environment) and IAR C-SPY (assembler and C debugger). Please read the file ew430ks.htm for the latest information about the Workbench, and read the file cs430.htm for the latest information about C-SPY. For simplicity, the term Kickstart is used to refer to the Workbench and C-SPY collectively. Kickstart is supplied on the CD-ROM included with each FET, and the latest version is available from the MSP430 web site.

Kickstart is compatible with WINDOWS '95, '98, 2000, ME, NT4.0, and XP.

Hardware Installation, MSP-FET430X110

1. Connect the 25-conductor cable originating from the FET to the parallel port of your PC.
2. Ensure that the MSP430F1121IDW 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 jumpers J1 (near the non-socketed IC on the FET) and J5 (near the LED) are in place. Pictures of the FET and its parts are presented later in this document.

Hardware Installation, MSP-FET430Pxx0 ('P120, 'P140, 'P410, 'P440)

1. Use the 25-conductor cable to connect the FET Interface module to the parallel port of your PC.
2. Use the 14-conductor cable to connect the FET Interface module to the Target Socket module.
3. 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.
4. Ensure that the two jumpers (LED and Vcc) near the 2x7 pin male connector are in place. Pictures of the Target Socket module and its parts are presented later in this document.

"Flash"ing the LED (an example in assembler and C)

This section demonstrates on the FET the equivalent of the C-language "Hello, world!" introductory program; assembler and C applications that flash the LED are developed and downloaded to the FET, and then run.

Assembler Example

1. Start the Workbench (START->PROGRAMS->IAR SYSTEMS->IAR EMBEDDED WORKBENCH FOR MSP430 KICKSTART->IAR EMBEDDED WORKBENCH).
2. Use FILE->OPEN to open the project file at:
   <Installation root>\ew23\430\FET_examples\Fet###\Assembler\Fet###_1\Fet###_1.prj
   where ### is the Fet tool number (e.g. 110 for the FET430X110, 120 for the FET430P120, etc.).
3. Use PROJECT->BUILD ALL to assemble and link the source code. You can view the source code by double-clicking Common Sources, and then double-clicking on the file Fet###_1.s43 in the Fet###_1.prj window.
4. Ensure that C-SPY is properly configured (With DEBUG selected, PROJECT->OPTIONS->C-SPY);
   1. PARALLEL PORT->PARALLEL PORT->LPT1 (default) or LPT2 or LPT3
5. Use PROJECT->DEBUGGER to start C-SPY. C-SPY will erase the device Flash, and then download to the device Flash the application object file.
6. In C-SPY, use EXECUTE->GO to start the application. The LED should flash!
7. In C-SPY, use FILE->EXIT to exit C-SPY.
8. In the Workbench, use FILE->EXIT to exit the Workbench.

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

C Example

1. Start the Workbench (START->PROGRAMS->IAR SYSTEMS->IAR EMBEDDED WORKBENCH FOR MSP430 KICKSTART->IAR EMBEDDED WORKBENCH).
2. Use FILE->OPEN to open the project file at:
   <Installation root>\ew23\430\FET_examples\Fet###\C\Fet###_1\Fet###_1.prj
   where ### is the Fet tool number (e.g. 110 for the FET430X110, 120 for the FET430P120, etc.).
3. Use PROJECT->BUILD ALL to compile and link the source code. You can view the source code by double-clicking Common Sources, and then double-clicking on the file Fet###_1.c in the Fet###_1.prj window.
4. Ensure that C-SPY is properly configured (With DEBUG selected, PROJECT->OPTIONS->C-SPY);
   1. PARALLEL PORT->PARALLEL PORT->LPT1 (default) or LPT2 or LPT3
5. Use PROJECT->DEBUGGER to start C-SPY. C-SPY will erase the device Flash, and then download to the device Flash the application object file.
6. In C-SPY, use EXECUTE->GO to start the application. The LED should flash!
7. In C-SPY, use FILE->EXIT to exit C-SPY.
8. In the Workbench, use FILE->EXIT to exit the Workbench.

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

Solutions to a Common Problem When Using the FET

A common problem that users report when using the FET is that their PC cannot communicate with the device. Possible solutions to this problem include:

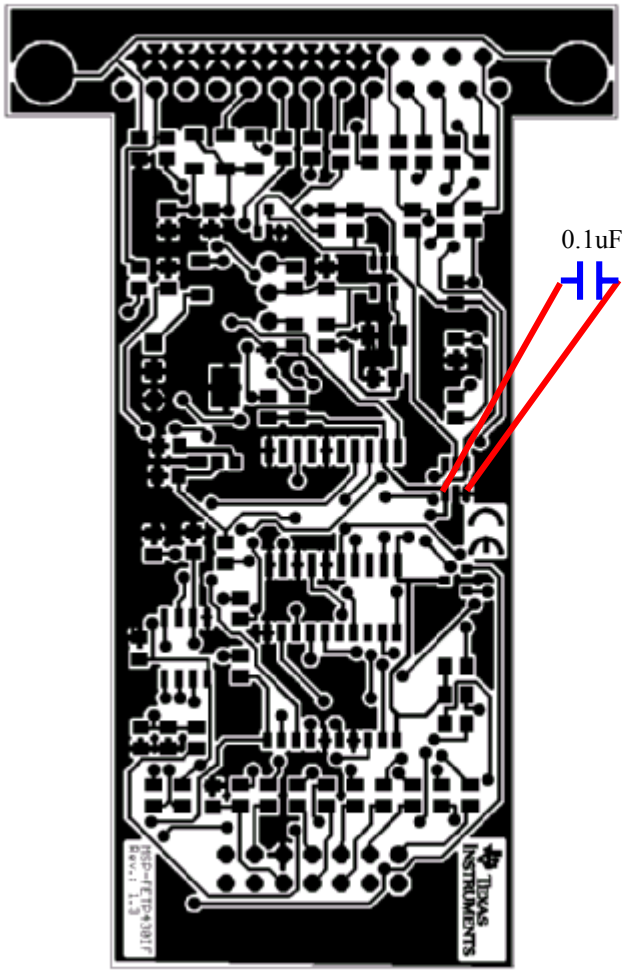
1. Insure that R6 on the MSP-FET430X110 and the FET Interface module has a value of 82 ohms. Early units were built using a 330 ohm resistor for R6. Refer to the diagrams in the Schematic and PCB Pictorials section to locate R6. The FET Interface module can be opened by inserting a thin blade between the case halves, and then carefully twisting the blade so as to pry the case halves apart.
2. Insure that the correct parallel port is being specified in the C-SPY configuration window (LPT1, 2, or 3). Check the PC BIOS for the parallel port address (0x378, 0x278, 0x3bc), and the parallel port configuration (ECP, Compatible, Bidirectional, or Normal). Refer to Tips and Tricks, Debugging #6 later in this document.
3. Insure that no other software application has reserved/taken control of the parallel port (say, printer drivers, ZIP drive drivers, etc.). Such software can prevent the C-SPY/FET driver from accessing the parallel port, and, hence, communicating with the device.
4. Revisions 1.0, 1.1, and 1.2 of the FET Interface module require a hardware modification; a 0.1uF capacitor needs to be installed between U1 pin 1 (signal VCC_MSP) and ground. A convenient (electrically equivalent) installation point for this capacitor is between pins 4 and 5 of U1. Refer to the diagram following.

   Note: The hardware modification may have already been performed during manufacturing, or your tool may contain an updated version of the FET Interface module.
5. Revisions 0.1 and 1.0 of the MSP-TS430PM64 Target Socket module require a hardware modification; the PCB trace connecting pin 6 of the JTAG connector to pin 9 of the MSP430 (signal XOUT) needs to be severed. Refer to the diagram following.

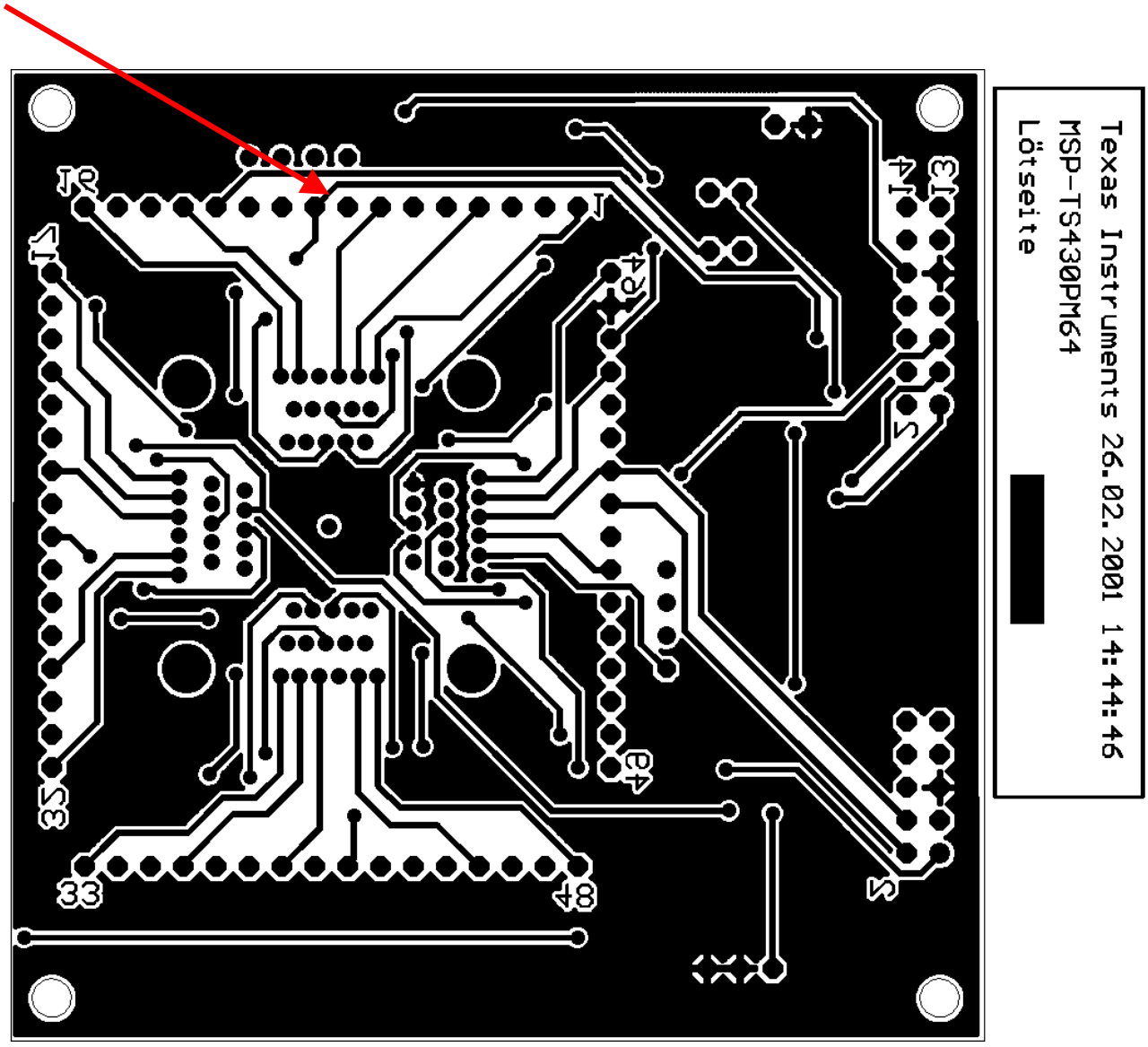
   Note 1: The hardware modification may have already been performed during manufacturing, or your tool may contain an updated version of the Target Socket module.

   Note 2: If the modified Target Socket module is used with the PRGS, Version 1.10 or greater of the PRGS software is required.

For revisions 1.0, 1.1, and 1.2 of the FET Interface module, install a 0.1uF capacitor between the indicated points (pins 4 and 5 of U1).



For revisions 0.1 and 1.0 of the MSP-TS430PM64 Target Socket module, severe the trace at the indicated point. Note: The following image depicts the back side of the PCB **when viewed from the component side of the PCB.**



Significant Changes from Software Version 3.01

1. Several errors in the C-SPY/FET driver have been corrected.
2. The RST/NMI pin is now asserted and negated after power is applied during device initialization. The device is then reset normally (via software (PUC), or RST/NMI (again), or by cycling power).
3. The user can configure C-SPY so that the RST/NMI pin is asserted and negated during a "RESET" operation (rather than a just a software reset). To enable this behavior, the .ini file variable HardReset must be set to 1 (HardReset=1). The default value of HardReset is 0 (i.e., only software reset is used).
4. The MSP430F43x, MSP430F44x, MSP430F11x2, and MSP430F12x2 devices are also now supported.
5. The default settings of the MSP430F41x/43x/44x Clock Control dialog were changed for consistency.
6. Existing projects built with Kickstart and Baseline software tools (from IAR) must be rebuilt (PROJECT->BUILD ALL) prior to use with the C-SPY debugger included with this software release.

Significant Changes from Software Version 3.02

1. Several errors in the C-SPY/FET driver relating to the Data Transfer Controller (DTC) of the MSP430F12x2 have been corrected.
2. Within Realtime mode, single step of machine instructions (i.e., assembler) is supported.

Important MSP430 Documents on the CD-ROM

The MSP430 CD-ROM supplied with the FET contains a wealth of information.

From the MSP430 main page on the CD-ROM, navigate to: Literature->MSP430 Literature->Data Sheets, to access the MSP430 device data sheets.

From the MSP430 main page on the CD-ROM, navigate to: Literature->MSP430 Literature->User's Guides, to access the User's Guides of our MSP430 tools.

Development Flow

Overview

Applications are developed in assembler and/or C using the Workbench, and they are debugged using C-SPY. C-SPY can be configured to operate with the FET (i.e., an actual MSP430 device), or with a software simulation of the device. Kickstart is used to refer to the Workbench and C-SPY collectively. The Kickstart software tools are a product of IAR.

Documentation for the MSP430 family and Kickstart is extensive. The CD-ROM supplied with this tool contains a large amount of documentation describing the MSP430. The MSP430 home page on the world wide web (www.ti.com/sc/msp430) is another source of MSP430 information. The components of Kickstart (workbench, assembler, compiler, linker, debugger) are fully documented in 430\doc under the Kickstart installation directory root (Refer to readme.htm). Additional files located throughout the Kickstart directory tree contain the most up to date information and supplement the .pdf files. In addition, Kickstart documentation is available on-line via HELP. Please ignore the references in the Kickstart documentation to DOS and "Command Line" commands. The IAR User's Guides (.pdf) do not make reference to the FET or the MSP430 Flash devices.

430\doc\FET_Doc_Overview.htm conveniently organizes the IAR and TI documents.

Tool	User's Guide	Most Up To Date Information
Workbench	ew430.pdf	Readme.htm & ew430ks.htm
Assembler	a430.pdf	A430.htm
Compiler	icc430.pdf	Icc430.htm
C library		Clib.txt
Linker		Xlink.htm
Linker		Xman.htm
C-SPY Debugger	cw430.pdf	Cs430.htm

Using Kickstart

The Kickstart development environment is function-limited. The following restrictions are in place:

1. The C compiler will not generate assembly code output.
2. The C library supports only basic floating-point operations (addition, subtraction, multiplication, and division).
3. The linker will link a maximum of 2K bytes of code originating from C source (but an unlimited amount of code originating from assembler source).
4. C-SPY does not support code profiling.
5. The IAR Simulator will input a maximum of 400 lines of C source (but an unlimited number of lines of assembler source). The TI Simulator has no such limitations. The TI Simulator is installed as the C-SPY Simulator as part of the Kickstart installation and upgrade.

A "Full" (i.e., unrestricted) version of the software tools can be purchased from IAR. A mid-featured tool set – called "Baseline", with an 8K byte C code size limitation and no support for floating-point arithmetic – is also available from IAR. Please consult the IAR web site (www.iar.se) for more information.

Project Settings

The settings required to configure the Workbench and C-SPY are numerous and detailed. Please read and thoroughly understand the documentation supplied by IAR when dealing with project settings. Please review the project files (.prj) supplied with the assembler and C examples; use these files as templates when developing your own project files. The project settings are accessed using: PROJECT->OPTIONS

The following project settings are recommended/required:

1. Choose the "–v0, 310/320 series (no hardware multiplier)" when developing with MSP430 devices without a hardware multiplier. Choose the "–v1, 330 series (hardware multiplier present)" when developing with MSP430 devices with a hardware multiplier. (GENERAL->TARGET)
2. Enable Debug Information in the compiler. (ICC430->DEBUG)
3. Enable Generate Debug Information in the assembler. (A430->CODE GENERATION)
4. Enable Debug Info in the linker Format section. (XLINK->OUTPUT)
5. Override the XCL File Name. Refer to System Files below. (XLINK->INCLUDE)
6. Select the C-SPY driver: Select Simulator to debug on the simulator. Select Flash Emulation Tool to debug on the FET. (C-SPY->SETUP). Select the active parallel port in PARALLEL PORT.
7. Override and select the correct Chip Description for C-SPY. Refer to System Files below. (C-SPY->SETUP)
8. Avoid the use of absolute pathnames when referencing files. Instead, use the relative pathname keywords $TOOLKIT_DIR$ and $PROJ_DIR$. Refer to the IAR documentation for a description of these keywords. The use of relative pathnames will permit projects to be moved easily, and projects will not require modification when IAR systems are upgraded (say, from Kickstart, or Baseline, to Full).

System Files

The following configuration and special files are provided to facilitate development of MSP430 applications under Kickstart. In each category, choose the file corresponding to the specific device being developed for (yyy).

1. Linker control files for point 5. above that support assembler development:
   $TOOLKIT_DIR$\icc430\msp430yyyA.xcl
2. Linker control files for point 5. above that support C development:
   $TOOLKIT_DIR$\icc430\msp430yyyC.xcl
3. Chip Description files for point 7. above that support debugging:
   $TOOLKIT_DIR$\cw430\msp430yyy.ddf
4. Device definition "#include" files:
   $TOOLKIT_DIR$\inc\msp430xyyy.h
5. C library files:
   $TOOLKIT_DIR$\lib\cl430ks.r43
   $TOOLKIT_DIR$\lib\cl430ksm.r43 // Support for hardware multiplier
   $TOOLKIT_DIR$\lib\cl430ks_pic.r43 // Position Independent Code
   // Support for hardware multiplier, Position Independent Code
   $TOOLKIT_DIR$\lib\cl430ksm_pic.r43

Stack Management within the .xcl files

The .xcl files are input to the linker, and contain statements that control the allocation of device memory (RAM, Flash). Refer to the IAR XLINK documentation for a complete description of these files. The .xcl files provided with the FET (MSP430xxxA.xcl [for assembler], MSP430xxxC.xcl [for C]) define a relocatable segment (RSEG) called CSTACK. CSTACK is used to define the region of RAM that is used for the system stack within C programs. CSTACK can also be used in assembler programs [MOV #SFE(CSTACK), SP]. CSTACK is defined to extend from the last location of RAM to the first location of RAM (i.e., the stack extends downwards through RAM).

Other statements in the .xcl file define other relocatable regions that are allocated from the first location of RAM to the last location of RAM. **It is critical to note that the supplied .xcl files do not explicitly reserve any RAM for the stack.** Instead, the supplied .xcl files instruct the linker to allocate RAM as needed to the other segments (UDATA0, IDATA0, ECSTR), and then assume that the remaining RAM (if any) is adequate for the stack (basically, CSTACK uses whatever RAM is left over). There is no explicit check that there is sufficient memory for the stack. A warning will only be output if there is insufficient memory for the other segments.

This approach to RAM allocation was taken to maximize the memory available to the other segments (and free the user from allocating memory for the stack that might not be used). Also, the RAM size differs between MSP430 devices, and it is not possible to anticipate how an end user will need to allocate RAM between the various segments.

The supplied .xcl files can be easily modified to explicitly allocate a region of RAM for the stack. For example, the following .xcl statements allocate 32 of the 256 bytes of RAM available in an F1121 for the CSTACK, and the remaining 224 bytes are available for the other segments:

```
-Z(DATA)UDATA0,IDATA0,ECSTR=0200-02DF

-Z(DATA)CSTACK+32#0300
```

Realtime/Non-Realtime Debugging

C-SPY supports two fundamental modes of debugging: Realtime and Non-Realtime. During Realtime debugging, the device operates at full device speed and makes use of a limited number of on-chip debugging resources (specifically, N breakpoint registers [See below]). During Non-Realtime debugging, the device executes under the control of the host PC; the system operates at a much slower speed, but offers a software breakpoint at every instruction address. During Non-Realtime mode, the PC effectively repeatedly single steps the device and interrogates the program counter after each operation to determine if a breakpoint address has been hit. Realtime is the default mode. CONTROL->REALTIME selects or deselects Realtime mode.

Attention: When Realtime is selected, (single) STEP is disabled and GO can only be executed with a maximum of N breakpoints active. If more than N breakpoints are active in Realtime mode and GO is selected, a message is output that informs the user that the operation is not possible.

Device	Breakpoints (N)
MSP430F11x1	2
MSP430F11x2	2
MSP430F12x	2
MSP430F12x2	2
MSP430F13x	3
MSP430F14x	3
MSP430F41x	2
MSP430F43x	3
MSP430F44x	3

Using Breakpoints

If there are N or fewer breakpoints active, C-SPY will always operate in Realtime mode (regardless of the setting of CONTROL->REALTIME).

If there are greater than N breakpoints active, C-SPY can only operate in Non-Realtime mode. An error message is output if execution is attempted with greater than N breakpoints active and the mode is Realtime.

The GO TO CURSOR operation implicitly requires a breakpoint. Consequently, only N-1 breakpoints can be active in Realtime mode when GO TO CURSOR is used.

RESET'ing a C program implicitly requires a breakpoint. Consequently, only N-1 breakpoints can be active in Realtime mode when the C program is RESET. Refer to Known Problems, Debugging #1.

If, while processing a breakpoint, an interrupt becomes active, C-SPY will stop at the first instruction of the interrupt service routine. Refer to Known Problems, Debugging #21.

Refer to Tips and Tricks, Debugging #1.

Using Single Step

Only single step (STEP) is supported in Realtime mode. Non-Realtime mode must be selected to enable the other single step functions (STEP INTO, GO OUT), in addition to STEP.

When debugging an assembler file, a single step (STEP) or STEP INTO operation of a non-CALL instruction executes the instruction at full device speed.

When debugging an assembler file, a STEP INTO operation of a CALL instruction will stop at the first instruction of the CALL'ed function.

When debugging an assembler file, a single step (STEP) operation of a CALL instruction to a function defined in the same file as the reset vector function and before the reset vector function will execute the CALL'ed function and will stop on the instruction following the CALL instruction. The CALL'ed function will execute in non-realtime.

When debugging an assembler file, a single step (STEP) operation of a CALL instruction to a function defined in a different file than the reset vector function or after the reset vector function will stop at the first instruction of the CALL'ed function. In this case, STEP operates identical to STEP INTO.

When debugging an assembler file, a (true) STEP OVER a CALL instruction that executes the CALL'ed function at full device speed (regardless of where the source of the CALL'ed function is located) can be synthesized by placing a breakpoint after the CALL and GO'ing (to the breakpoint [in Realtime mode]).

When debugging an assembler file, GO OUT is not supported; C-SPY will stop functioning if it is used.

When single stepping C, Non-Realtime mode should be selected. The descriptions below assume Non-Realtime mode. When single stepping C in Realtime mode, the unit of step is the machine instruction (i.e., assembler), and not the C statement.

When debugging a C file, a single step (STEP) operation executes the next C statement. Thus, it is possible to step over a function reference. Statements are executed in Non-Realtime mode. STEP INTO is supported. GO OUT is supported.

Within Disassembly mode (View->Toggle Source/Disassembly), a single step (STEP) operation of a CALL instruction will place – if possible - a hardware breakpoint after the CALL instruction, and then execute GO. The CALL'ed function will execute at full device speed. If no hardware breakpoint was available prior to the GO, the CALL'ed function will be executed in Non-Realtime mode. In either case, execution will stop at the instruction following the CALL. GO OUT is not supported in Disassembly mode.

It is only possible to single step when source statements are present. Breakpoints must be used when running code for which there is no source code (i.e., place the breakpoint after the CALL to the function for which there is no source, and then GO to the breakpoint [in Realtime mode]).

If, during a single step operation, an interrupt becomes active, the current instruction is completed and C-SPY will stop at the first instruction of the interrupt service routine. Refer to Known Problems, Debugging #21.

Using Watch Windows

The C-SPY Watch Window mechanism permits C variables to be monitored during the debugging session. Although not originally designed to do so, the Watch Window mechanism can be extended to monitor assembler variables. Presented here are two methods to do so.

Assume that the variables to watch are defined in RAM, say:

```
 RSEG udata0
varword ds 2 ; two bytes per word
varchar ds 1 ; one byte per character
```

Method 1:

In C-SPY:

1. Open the Watch Window: Window->Watch
2. Use Control->Quick Watch
3. To watch varword, enter in the Expression box: (unsigned int *) #varword
4. To watch varchar, enter in the Expression box: #varbyte
5. Press the Add Watch button
6. Close the Quick Watch window
7. For the created entry in the Watch Window, click on the + symbol. This will display the contents (or value) of the watched variable.

To change the format of the displayed variable (binary, ascii, hex, int, oct), select the value and then click the right mouse button. Select Properties. This will bring up a window that will permit the Display Format to be changed. From this window, the value of the Watched variable can also be changed.

The Watch Window is limited. It is very difficult to select the value of the watched variable so that it can be changed. If the address of the watched variable changes, delete the variable from the Watch Window and create a new variable in the Watch Window with the new address.

Method 2:

In C-SPY:

1. Use Control->Quick Watch
2. To watch varword, enter in the Expression box: #varword
3. To watch varchar, enter in the Expression box: #varbyte
4. Note the value of the expression. This is the address of the variable.
5. Close the Quick Watch window.
6. Choose: Options->Settings->Register Setup
7. Select New Virtual Register
8. Fill in the information: Name, Size (1 for bytes, 2 for words, 4 for long words), base (binary, decimal, hexadecimal), and the noted address from Step 3.
9. Select OK and OK.

The variable is now displayed in the Register window. The variable can be watched (and changed) like any other register.

Use Register Setup->Delete to remove this variable from the Register window.

If the address of the watched variable changes, delete the variable from the Register window and then add the new variable to the Register window (using the new address).

In C, variables can be watched by selecting them and then dragging-n-dropping then into the Watch Window.

Since the MSP430 peripherals are memory mapped, if is possible to extend the concept of watching variables to watching peripherals. Be aware that there may be side effects when peripherals are read and written by C-SPY. Refer to Known Problems, Debugging #18.

CPU core registers can be specified for watching by proceeding their name with # (i.e., #PC, #SR, #SP, #R5, etc.).

Note: Variables watched within the Watch Window are only updated when C-SPY gets control of the device (say, following a breakpoint hit, a single step, or a STOP/escape).

FET Specific Menus

Control->Real Time

Select Realtime mode or deselect Realtime mode (i.e., Non-Realtime mode).

FET Options

The current device type is displayed.

FET Options->Download Options

- Erase Flash Memory before Download
    - **Main and Information Memory**
      Erase both Flash memories before download.
    - **Main Memory only**
      Erase the Main Flash memory only before download. The Information memory is not erased.
    - **Retain Unchanged Memory**
      The Information and Main Flash memories are read into a buffer. The Flash memories are then erased. The data to be written is written to the buffer (overwriting the previous buffer contents at those selected locations). The new data effectively replaces the old data, and unaffected old data is retained. The buffer is then written to the Flash memories.
- When enabled, **Verify Download** verifies that program data has been correctly transferred from the PC to the device. This verification does increase the programming sequence time.

FET Options->Release JTAG on Go

C-SPY uses the device JTAG signals to debug the device. On some MSP430 devices, these JTAG signals are shared with the device port pins. Normally, C-SPY 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 GO is selected, the JTAG drivers are set to tri-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, C-SPY has no access to the device and cannot determine if an active breakpoint (if any) has been reached. C-SPY must be manually commanded to stop the device at which time the state of the device will be determined (i.e., Was a breakpoint reached?).

If RELEASE JTAG ON GO is selected, the JTAG pins will be released **if and only if** there are N or fewer active breakpoints.

Refer to Tips and Tricks, Debugging #10 and Known Problems, Debugging #2.

FET Options->Resynchronize JTAG

Regain control of the device.

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

FET Options->Init New Device

Initialize the device according to the settings in the Download Options. Basically, the current program file is downloaded to the device memory. The device is then reset. This option can be used to program multiple devices with the same program from within the same C-SPY session.

It is not possible to Init New Device while the device is operating.

FET Options->Advanced->Views->Registers

Display the device registers. The "@:" is an "indirect" control, and displays in the adjacent window the memory contents as addressed by the corresponding device register. The device register contents can be changed using this window (while the device is not operating). Note: If the IAR Window->Register is open, changes made using the new Registers window will not be reflected in the Register window immediately (and vise versa).

FET Options->Advanced->Views->Stack

Display the device memory as addressed by the Stack Pointer (SP). The highlighted address and contents indicates the "top of the stack". The stack contents can be changed using this window (while the device is not operating).

FET Options->Advanced->Views->LCD

Display the LCD. The LCD is only updated when C-SPY has control of the device (i.e., the device is not operating).

FET Options->Advanced->General Clock Control

Disable the specified system clock while C-SPY 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). Refer to Known Problems, Debugging #8.

FET Options->Advanced->Memory Dump

Write the specified device memory contents to a specified file. A conventional dialog is displayed that permits the user to specify a file name, a memory starting address, and a length. The addressed memory is then written in a text format to the named file. Options permit the user to select word or byte text format, and address information and register contents can also be appended to the file.

FET Options->Advanced->LCD Setup

Select the LCD configuration file.

FET Options->Advanced->Emulation Mode

Specify the device to be emulated. The device must be reset (or reinitialized (Init New Device)) following a change to the emulation mode.

Refer to the section "Signal-mapping when the MSP-TS430PZ100 module with an F449 is used to emulate an F44x or an F43x in an 80-pin package:" later in this document.

Note: Not all FET Options->Advanced menus are supported by all MSP430 devices.

Appendices

Sources of Additional Information

The primary sources of MSP430 information are the device specific data sheet and User's Guide. 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/sc/msp430) will contain the latest version of these documents.

The .pdf files in the 430\doc directory document the components of Kickstart: the workbench, and assembler, the C compiler, the linker, and the (C-SPY) debugger. Copies of the .pdf files are included on the MSP430 CD-ROM. The IAR User's Guides (.pdf) do not make reference to the FET or to the MSP430 Flash devices. The .htm files in the 430\doc directory supplement the Kickstart documentation, and contain the latest information.

Sources of 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/sc/msp430.

Note: Although Kickstart is a product of IAR, Texas Instruments provides the support for it. Therefore, please do not request support for Kickstart from IAR. Please consult the extensive documentation provided with the product before requesting assistance.

Design Considerations for In-Circuit Programming

Boot Strap Loader

The JTAG pins provide access to the Flash memory of the MSP430F device. On some devices, these pins must be "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, MSP430F devices contain a program (a "Boot Strap Loader") that permits the Flash memory to be erased and programmed simply, using a reduced set of signals. An Application Note that fully describes this interface in available from the MSP430 web site. Note: TI does not produce a BSL tool. However, customers can easily develop their own BSL tools using the information in the Application Note, or BSL tools can be purchased from 3rd parties. Please refer to the MSP430 web site for a list of MSP430 3rd party tool developers.

Texas Instruments suggests that customers of the MSP430F devices design their circuits with the BSL in mind (i.e., we suggest that the customer provide easy access to these needed signals (say, via a header)).

Device Signals

The following device signals should be brought out (i.e., made accessible) so that the FET and PRGS (Programming Adapter Serial) tools can be utilized:

RST/NMI, TMS, TCK, TDI, TDO, GND, VCC, and TEST (if present).

Note 1: Connections to XIN and XOUT are not required, and should not be made.

Note 2: PRGS software Version 1.10 or greater must be used.

The BSL tool requires the following device signals: RST/NMI, TCK, GND, VCC, P1.1, P2.2, and TEST (if present)

External Power

The PC parallel port is capable of sourcing a limited amount of current. Owing to the ultralow power requirement of the MSP430, a stand-alone FET does 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 MSP-FET430X110 and the Target Socket modules. Refer to the schematics and pictorials of the MSP-FET430X110 and the Target Socket modules presented later in this document to locate the external power connections.

When an MSP-FET430X110 is powered from an external supply, an on-board device regulates the external voltage to the level required by the MSP430.

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). Again, refer to the Target Socket module schematic.

Signal Connections for In-System Programming and Debugging, MSP-FET430X110

With the proper connections, you can use the C-SPY debugger and the MSP-FET430X110 to program and debug code on your own target board. In addition, the connections will support the MSP430 Serial Programming Adapter (PRGS), thus providing an easy way to program prototype boards, if desired.

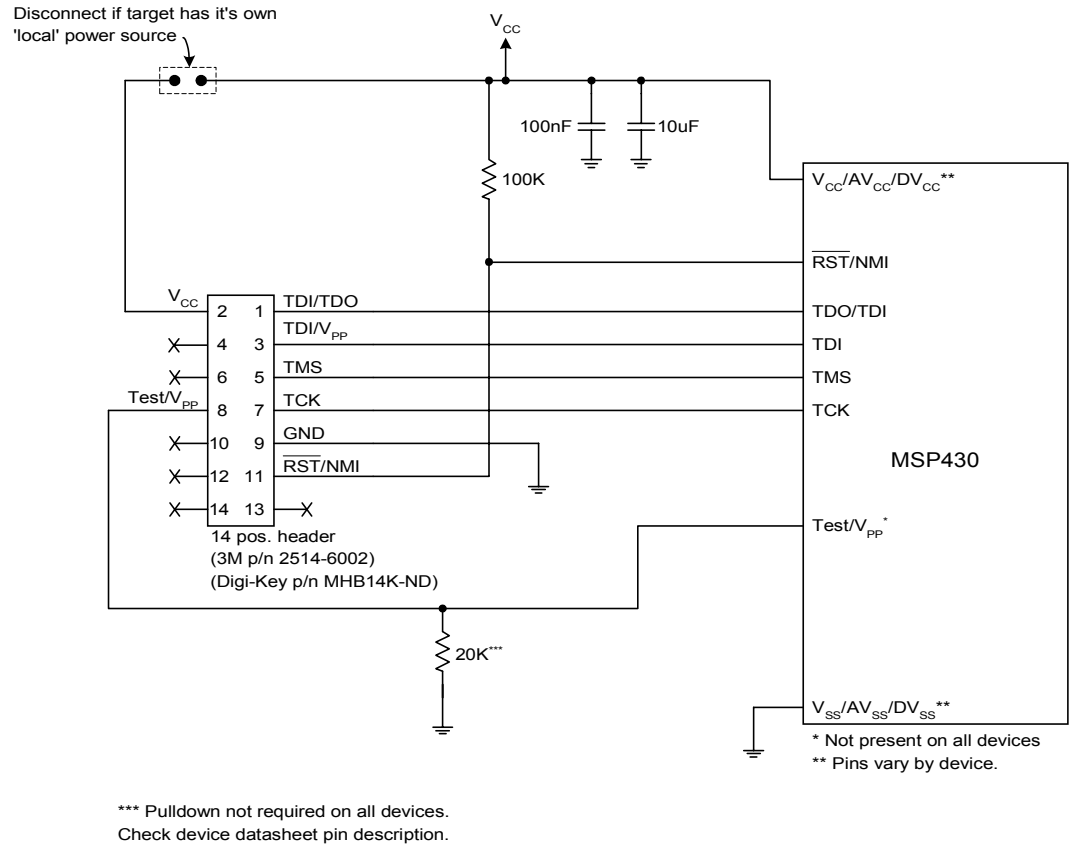
The figure below shows the connections between the FET and the target device required to support in-system programming and debugging using C-SPY. If your target board has it's own 'local' power supply, such as a battery, do not connect Vcc to pin 2 of the JTAG header. Otherwise, contention may occur between the FET and your local power supply.

The figure shows a 14-pin header (available from Digi-Key, p/n MHB14K-ND), being used for the connections on your target board. It is recommended that you build a wiring harness from the FET with a connector which mates to the 14-pin header, and mount the 14-pin header on your target board. This will allow you to unplug your target board from the FET as well as use the Serial Programming Adapter to program prototype boards, if desired.

The signals required are routed on the FET to header locations for easy accessibility. Refer to the device datasheet (for pin numbers) and the schematic and PCB information in this User's Guide to locate the signals.

After you make the connections from the FET to your target board, remove the MSP430 device from the socket on the FET so that it does not conflict with the MSP430 device in your target board. Now simply use C-SPY as you would normally to program and debug.

Connections



Note: No JTAG connection is required to the XOUT pin of the MSP430 as shown on some schematics.

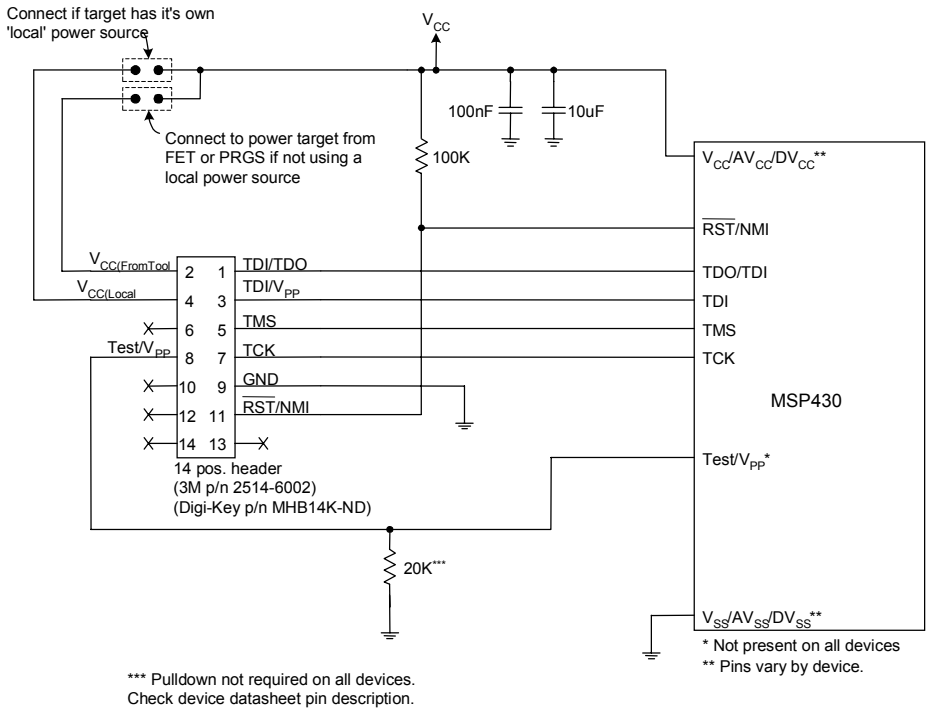
Signal Connections for In-System Programming and Debugging, MSP-FET430Pxx0 ('P120, 'P140, 'P410, 'P440)

With the proper connections, you can use the C-SPY debugger and the MSP-FET430Pxx0 ('P120, 'P140, 'P410, 'P440) to program and debug code on your own target board. In addition, the connections will support the MSP430 Serial Programming Adapter (PRGS), thus providing an easy way to program prototype boards, if desired.

The figure below shows the connections between the FET Interface module and the target device required to support in-system programming and debugging using C-SPY. The figure shows a 14-pin header (available from Digi-Key, p/n MHB14K-ND) connected to the MSP430. With this header mounted on your target board, the FET Interface module can be plugged directly into your target. Then simply use C-SPY as you would normally to program and debug.

The connections for the FET Interface module and the Serial Programming Adapter (PRGS) are identical with the exception of V_{CC}. Both the FET Interface module and PRGS can supply V_{CC} to your target board (via pin 2). In addition, the FET Interface module has a V_{CC}-sense feature that, if used, requires an alternate connection (pin 4 instead of pin 2). The FET Interface module 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 its output signals accordingly. The PRGS does not support this feature, but does provide the user the ability to adjust its JTAG signal levels to the V_{CC} level on your target board through the GUI. 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 of the FET Interface module 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 PRGS. 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 PRGS) the V_{CC} connection is made to pin 2 on the JTAG header and no connection is made to pin 4. The schematic shows a jumper block in use. The jumper block 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.

Connections



Note: No JTAG connection is required to the XOUT pin of the MSP430 as shown on some schematics.

How to generate Texas Instrument .TXT (and other format) files

The Kickstart linker can be configured to output objects in TI .TXT format for use with the PRGS. Select: PROJECT->OPTIONS->XLINK->OUTPUT->FORMAT->OTHER->msp430-txt. Intel and Motorola formats can also be selected.

Note: At this time C-SPY cannot input a .TXT file.

Refer to Tips and Tricks, Program development #4.

Overview of Example Programs

Example programs for MSP430 devices are provided in <Installation root>\ew23\430\FET_examples. Both assembler and C examples are provided in their respective sub-directories. The majority of the examples use the various resources of the MSP430 to time the flashing of the LED. Note: The examples often assume the presence of a 32KHz crystal, and not all FETs are supplied with a 32KHz crystal.

<Installation root>\ew23\430\FET_examples\Content.txt describes each of the example programs.

Tips and Tricks, Hardware (MSP430, FET)

1. The state of the device (registers, memory, etc.) is undefined following a reset, or (re)programming of the Flash. The only exception to the above statement is that the PC is loaded with the word at 0xfffe (i.e., the reset vector).
2. A common MSP430 "mistake" is to fail to disable the Watchdog mechanism; the Watchdog is enabled by default, and it will reset the device if not disabled or properly handled by your application. Refer to Known Problems, Program Development #9.
3. When the MSP-FET430X110 is used as an interface to an MSP430 on the user's circuit (i.e., there is no MSP430 device in the FET socket), the XOUT and XIN signals from the FET should not be connected to the corresponding pins of the in-circuit MSP430. Similarly, when using the Interface module, do not connect the XOUT and XIN signals from the Interface module to the corresponding pins of the in-circuit MSP430.
4. The 14-conductor cable connecting the FET Interface module and the Target Socket module must not exceed 8 inches (20 centimeters) in length.
5. To utilize the on-chip ADC voltage references (if present), C6 (10uF, 6.3V, low leakage) must be installed on the Target Socket module.
6. Crytstals/resonators Q1 and Q2 (if applicable) are not provided on the Target Socket module. For MSP430 devices which contain user selectable loading capacitors, the effective capacitance is the selected capacitance plus 3pF (pad capacitance) divided by two.
7. Crystals/resonators have no effect upon the operation of the tool and C-SPY (as any required clocking/timing is derived from the internal DCO/FLL).

Tips and Tricks, Program Development (Assembler, C, Linker)

1. When adding source files to a project, do not add files that are #include'ed by source files that have already been added to the project (say, an .h file within a .c or .s43 file). These files will be added to the project file hierarchy automatically.
2. In assembler, enclosing a string in double-quotes ("string") automatically prepends a zero byte to the string. Enclosing a string in single-quotes ('string') does not.
3. When using the compiler or the assembler, if the last character of a source line is backslash (\), the subsequent carriage return/line feed is ignored (i.e., it is as if the current line and the next line are a single line). When used in this way, the backslash character is a "Line Continuation" character.
4. The linker output format **must be** "Debug info" or "Debug info with terminal I/O" (.d43) for use with C-SPY. If the linker output format is .txt, the .d43 file is not updated and subsequent C-SPY sessions will utilize the existing .d43 file when present. Thus, you can loose synchronization between the source and the code being debugged. Do not launch C-SPY when you have "Other" output file format selected for the XLINK options.
5. Position Independent versions of the C libraries are provided. The libraries are named cl430ks_pic.r43 (no support for hardware multiply) and cl430ksm_pic.r43 (support for hardware multiply).
6. Within the C libraries that support devices with a hardware multiplier (cl430ksm.r43, cl430ksm_pic.r43), interrupts are disabled during critical sections of the floating-point functions. Contact TI if you wish the source code for these libraries so that this behavior can be disabled.
7. It is possible to mix assembler and C programs within the Workbench. TI is developing an application note describing how this is achieved. Please search the MSP430 web site for this Application Note.

Tips and Tricks, Debugging (C-SPY)

1. The break-on-data capability of the MSP430 is not utilized. At this time, breakpoints can only be set to occur during an instruction fetch (i.e., an address breakpoint). Future versions of C-SPY may provide this additional functionality. However, C-SPY does provide a non-realtime data breakpoint mechanism; it is possible to associate with an address breakpoint an expression that C-SPY evaluates when the breakpoint is hit. If the expression is FALSE, program execution resumes. And, if the expression is TRUE, C-SPY halts the program execution and displays the machine state. The breakpoint expression can be arbitrarily complex. Refer to the C-SPY documentation for a description of this data breakpoint mechanism.

2. C-SPY is capable of downloading data into RAM, INFORMATION, and Flash MAIN memories. A warning message is output if an attempt is made to download data outside of the device memory spaces.

3. C-SPY is capable of debugging applications that utilize interrupts and low power modes. Refer to Known Problems, Debugging #10 when single stepping low power mode instructions in C.

4. C-SPY is incapable of accessing the device registers and memory while the device is running. The user is warned if such an operation is attempted. The user must stop the device in order to access device registers and memory.

5. When C-SPY is started, the Flash memory is erased and the opened file is programmed in accordance with the previous Download Options. This initial erase and program operation can be disabled from within the Workbench by selecting PROJECT->OPTIONS->C-SPY->FLASH EMULATION TOOL->CODE->SUPPRESS DOWNLOAD. Programming of the Flash can be initiated manually with FET OPTIONS->INIT NEW DEVICE.

6. 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.

7. C-SPY attempts to reset the device using a hierarchy of operations. A soft reset (PUC) is used first. If this fails, a hard reset (RST/NMI) is used. If this fails, the device power is cycled. The device is reset when C-SPY is started and when the device is programmed. The device is also reset by the C-SPY RESET button, and when the device is manually reprogrammed and when the JTAG is resynchronized. When RST/NMI is not asserted (low), C-SPY sets the logic driving RST/NMI to high-impedance, and RST/NMI is pulled high via a resistor on the PCB.

   RST/NMI is now asserted and negated after power is applied during device initialization. The device is then reset normally (via software (PUC), or RST/NMI (again), or cycling power).

   The user can configure C-SPY so that RST/NMI is asserted and negated during a RESET button operation (rather than a just a software reset). To enable this behavior, the .ini file variable HardReset must be set to 1 (HardReset=1). The default value of HardReset is 0 (i.e., only software reset is used).

8. C-SPY is capable of debugging a device whose program reconfigures the function of the RST/NMI pin to NMI.

9. The level of the XOUT/TCLK pin is undefined when C-SPY resets the device. The logic driving XOUT/TCLK is set to high-impedance at all other times.

10. When making current measurements of the device, insure that the JTAG control signals are released (FET Options->Release JTAG on Go), otherwise the device will be powered by the signals on the JTAG pins and the measurements will be erroneous. Refer to Tips and Tricks, Debugging #12 and Known Problems, Debugging #2 and Hardware #2.

11. Within C-SPY, (most) settings (breakpoint settings, etc.) can be preserved by selecting OPTIONS->SETTINGS->WINDOW SETTINGS->GENERAL->RESTORE STATES). Note that the feature does not take effect until the next C-SPY session (i.e., C-SPY must be stopped and restarted to enable this feature). Refer to Known Problems, Debugging #19 and #22.

12. When C-SPY 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 conditions will be restored prior to STEP or GO. Consequently, do not measure the power consumed by the device while C-SPY has control of the device. Instead, run your application using GO with JTAG released. Refer to Tips and Tricks, Debugging #10 and Known Problems, Hardware #2.

Known Problems, Hardware (MSP430, FET)

1. **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.
2. **The device current increases by approximately 10uA when a device in low power mode is stopped** (using ESC)**, and then the low power mode is restored** (using GO)**.** This problem appears to happen on all devices except the MSP430F12x.

Known Problems, Program Development (Assembler, C, Linker)

1. The Workbench is capable of producing an object file in Texas Instruments .TXT format. **C-SPY is incapable of inputting an object file in Texas Instruments .TXT format**.
2. **The example programs giving in the Kickstart documentation (i.e., Demo, Tutor, etc.) are not correct.** The programs will work in the IAR simulator. However, the programs will not function correctly on an actual device because the Watchdog mechanism is active. The programs need to be modified to disable the Watchdog mechanism. Disable the Watchdog mechanism with the C statement: "WDTCTL = 0x5a80;", or "mov #5a80h,&WDTCTL" in assembler.
3. In assembler, **"#define string value" should not be followed with an assembler-style (;) comment** (as the comment will become part of "string" [since, by definition of #define, "value" extends to the end of the current line]). C-style (/*, */) and C++-style (//) comments can be used with #define without problem.
4. When defining modules, **special function definitions (SFRB, SFRW) are cleared from the assembler/compiler when a new module is started.** Consequently, it is required to re-#include the .h file that contains the SFR definitions. However, to enable this operation one must first circumvent the prevent-multiple-file-inclusion mechanism by use of the following statement prior to the #include operation: #undef __msp430xxxx (where xxxx is the device identification). If this operation is not done, the .h file will not be included, and the SFR definitions will be unknown.
5. The following **cryptic error message is output by the linker** when a C.xcl file is incorrectly used in an assembler project:

   Error[e46]: Undefined external "main" referred in CSTARTUP
   Warning[w52]: More than one definition for the byte at address 0xfffe in common segment INTVEC. It is defined in module "CSTARTUP" as well as in module "..."

   The solution to this problem is to use the correct A.xcl file (for assembler).

   Similarly, the following (or similar) **cryptic error message is output by the linker** when an A.xcl file is incorrectly used in a C project:

   Error[e46]: Undefined external "?CL430_1_25_L08" referred in <file> (<file path>)

   Again, the solution to this problem is to use the correct C.xcl file (for C).
6. **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.
7. **Access to MPY using an 8-bit operation is flagged as an error.** Within the .h files, 16-bit registers are defined in such a way that 8-bit operations upon them are flagged as an error. This "feature" is normally a good thing and can catch register access violations. However, in the case of MPY, it is also valid to access this register using 8-bit operators. If 8-bit operators are used to access MPY, the access violation check mechanism can be defeated by using "MPY_" to reference the register. Correspondingly, 16-bit operations on 8-bit registers are flagged as access violations.

8. The **IAR linker (XLINK) contains a bug that prevents the last byte in a memory segment specified within a .xcl file from being used.** An error message stating that **"Zero additional bytes are required"** is output. Work arounds to this condition are: 1) Ignore the error by specifying e16 within the linker->ignore error condition control, 2) Increasing the end of memory range specifier within the .xcl file by one.

9. **The CSTARTUP that is implicitly linked with all C applications does not disable the Watchdog timer.** Use WDT = WDTPW + WDTHOLD; to explicitly disable the Watchdog. This statement is best placed in the __low_level_init() function (before main()).

   If the Watchdog is not disabled and the Watchdog triggers and resets the device during CSTARTUP, **the source screen will go blank** as C-SPY is not able to locate the source code for CSTARTUP. Be aware that CSTARTUP can take a significant amount of time to execute if a large number of initialized global variables are used.

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

 WDTCTL = WDTPW + WDTHOLD; // Stop Watchdog timer

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

10. **Compiler optimization can remove unused variables and/or statements**, and can effect debugging. Optimization: NONE is supported within PROJECT->OPTIONS->ICC430->CODE GENERATION->OPTIMIZATION.

11. **The IAR Tutorial assumes a Full version of the Workbench.** Within a Kickstart or Baseline system, it is not possible to configure the C compiler to output assembler mnemonics.

12. The linker error **"Unable to open file 'cl430'"** is output when the settings for your Kickstart C project incorrectly reference a linker control file (C.xcl) of the Full system (and not the C.xcl files for the Kickstart system). The solution is to change the project settings to reference the Kickstart files.

Known Problems, Debugging (C-SPY)

1. **When debugging in C with N** (Refer to Realtime/Non-Realtime Debugging) **or more breakpoints active and in Realtime mode, a RESET causes the source window to go blank.** A breakpoint is implicitly required following the RESET of a C program; the breakpoint is used to halt on main(). If less than N breakpoints are active when RESET is asserted, there is no problem (since at most N breakpoints total are needed) and the application executes in Realtime mode to main(). However, following a RESET when N or more breakpoints are active, the debugger enters Non-Realtime mode (since more than N breakpoints are needed). In Non-Realtime mode, the debugger will execute the single instruction at the first line of the C set-up routine cstartup.s43. And since there is no source file for this routine in the project, a blank screen is displayed. If the system is set to display mixed C and assembler (VIEW->TOGGLE SOURCE/DISASSEMBLY), the assembler statements can be viewed. If the debugger was in Non-Realtime mode when the RESET was asserted, the CPU will automatically single step until it eventually reaches main(). At this time the source window will be refreshed with the cursor positioned on the first statement of main().

2. A consequence of implicitly GO'ing to a breakpoint on main() as described above is that **two GO'es are required to cause the RELEASE JTAG ON GO to work when working with assembler.** RELEASE JTAG ON GO requests C-SPY to effectively detach itself from the device, thus freeing the JTAG pins to their port functions (if applicable). However, once C-SPY releases its JTAG interface to the device, C-SPY looses the status of the device. If C-SPY was "blind" following a GO, it could not determine when the breakpoint at main() was reached. Therefore, it was decided that the JTAG pins would not be released until after the second GO was executed (the first GO was required to cause the CPU to execute from the first line to main().). Once the first (implicit) breakpoint is reached at main(), a second GO releases the CPU and disables the JTAG control.

   In assembler, however, there is no implicit initial breakpoint set. Thus, following a RESET, even if RELEASE JTAG ON GO is selected and GO is asserted, the JTAG controls will not be released. **To cause the JTAG controls to be released when using assembler, the device must be STOP'ped (from a running state), and then restarted using GO**. **Do NOT reset the device before restarting.** Thus, the sequence RESET-GO-STOP-GO is required to start the CPU with JTAG released (assuming that RELEASE JTAG ON GO is selected) when working in assembler.

3. The CONTROL->FILL MEMORY utility of C-SPY requires **hexadecimal values** for starting address and length to be **prefaced with "0x".** Otherwise the values are interpreted as decimal.

4. The MEMORY utility of C-SPY can be used to view the RAM, the INFORMATION memory, and the Flash MAIN memory. The MEMORY utility of C-SPY can be used to modify the RAM; **the INFORMATION memory and Flash MAIN memory cannot be modified using the MEMORY utility**. The INFORMATION memory and Flash MAIN memory can only be programmed when a project is opened and the data is downloaded to the device, or when FET OPTIONS->INIT NEW DEVICE is selected.

5. **The GO TO CURSOR operation implicitly requires one breakpoint.** Therefore, only N-1 additional breakpoint(s) can be supported while in Realtime mode when this feature is used. However, C-SPY will permit any number of breakpoints to be set, and it will indicate that the GO TO CURSOR function is available for use. C-SPY will output an error message indicating that too many breakpoints are set (either explicitly or implicitly) if program execution is then attempted.

6. **C-SPY does not permit the individual segments of the INFORMATION memory and the Flash MAIN memory to be manipulated separately**; consider the INFORMATION memory to be one contiguous memory, and the Flash MAIN memory to be a second contiguous memory.

7. 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.

8. C-SPY utilizes the system clock to control the device during debugging. Therefore, **device counters, etc., that are clocked by the Main System Clock (MCLK) will be effected when C-SPY 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 which support Clock Control (FET Options->Advanced->General Clock Control) can further minimize these effects by selecting to stop the clock(s) during debugging.

   Refer to Known Problems, Debugging #18.

9. **There is a time after C-SPY performs a reset of the device** (when the C-SPY session is first started, when the Flash is reprogrammed (via Init New Device), when JTAG is resynchronized (Resynchronize JTAG)) and before C-SPY has regained control of the device **that the device will execute normally**. This behavior may have side effects. Once C-SPY has regained control of the device, it will perform a reset of the device and retain control.

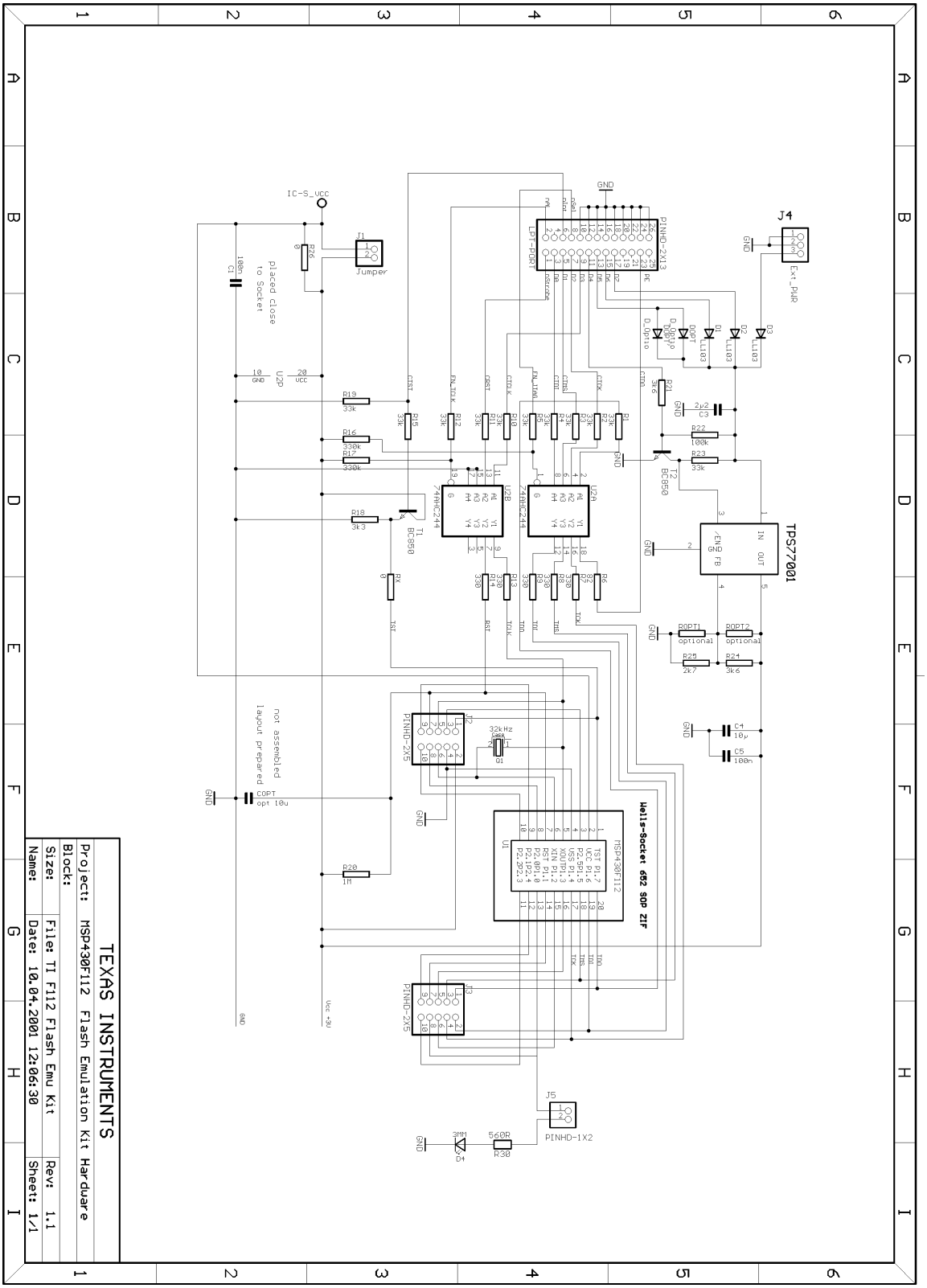
10. While debugging in C, **it is necessary to have a meaningful C statement following the LPM instruction** so that C-SPY can synchronize the returned PC with the C source (else C-SPY will repeatedly single step). Following the LPM with a _NOP(); works well in this case.
11. **GO OUT is not available while debugging assembler files. GO OUT operates like GO while debugging assembler files.**
12. When C-SPY is invoked, the Tutor (or Demo) project may always be opened. This can be very annoying (especially considering that C-SPY is capable of opening the last project opened). **Edit the system shortcut to C-SPY and delete the reference to the Tutor (or Demo) project.**
13. Within C-SPY, MEMORY->EDIT is used to read and write memory. Be aware that **MEMORY->EDIT always operates upon sixteen *** *bytes* *** of memory (xxx0h->xxxfh).** These unintended accesses may have undesirable side effects (especially when working with peripheral registers, etc., whose contents are cleared after being read and/or modified).
14. 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.
15. The **Dump Memory length specifier is restricted to four hexadecimal digits (0-ffff).** This limits the number of bytes that can be written from 0 to 65535. Consequently, it is not possible to write memory from 0 to 0xffff inclusive as this would require a length specifier of 65536 (or 10000h).
16. **The C-SPY Parallel Port "Log Communications" option is not supported.**
17. 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 C-SPY before these operations are complete. Consequently, **C-SPY will update its memory window with erroneous information.** A work around 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.
18. **Bits that are cleared when read during normal program execution** (i.e., Interrupt Flags) **will be cleared when read while being debugged** (i.e., memory dump, SFR display).

    Within MSP430F43x/44x devices, bits do not behave this way (i.e., the bits are not cleared by C-SPY read operations).
19. **The Advanced->Views->Registers and Advanced->Views->Stack windows are not preserved** between C-SPY sessions and when C-SPY is restarted.
20. **C-SPY cannot be used to debug programs that execute in the RAM of F12x and F41x devices.** A work around to this limitation is to debug programs in Flash.
21. **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 always stops on the first line of the ISR). However, this behavior is correct because the device will always process an active and enabled interrupt. A work-around for this behavior is, while within the ISR, to disable the GIE bit *on the stack* so that interrupts will be disabled after exiting the ISR. This will permit 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.

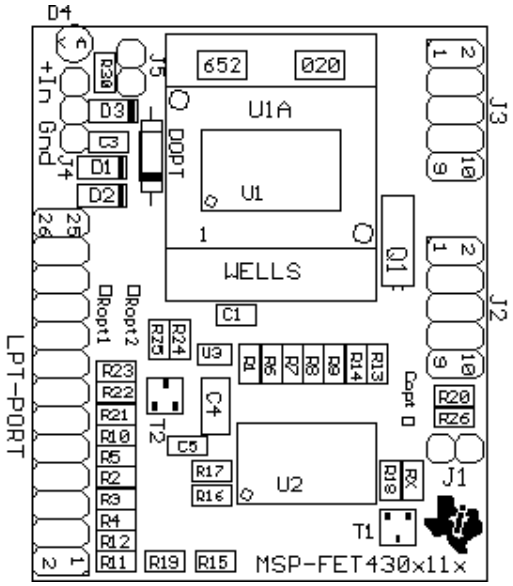
    On devices with Clock Control, it may be possible to suspend a clock between single steps and delay an interrupt request.
22. **The base** (decimal, hexadecimal, etc.) **property of Watch Window variables is not preserved when RESTORE STATES is enabled**; the base reverts to decimal.
23. On devices equipped with a Data Transfer Controller (DTC), **the completion of a data transfer cycle will preempt a single step of a low power mode instruction**. The device will advance beyond the low power mode instruction only after an interrupt is processed. Until an interrupt is processed, it will appear that the single step has no effect. A work around to this situation is to set a breakpoint on the instruction following the low power more instruction, and then execute (GO) to this breakpoint.

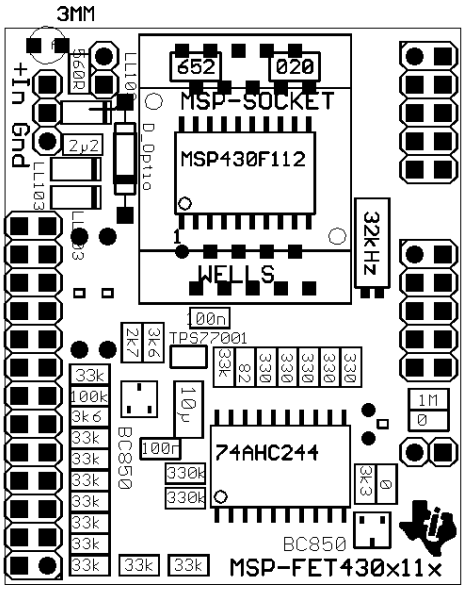
24. **The transfer of data by the Data Transfer Controller (DTC) may not stop precisely when the DTC is stopped in response to a single step or a breakpoint**. When the DTC is enabled and a single step is performed, one or more bytes of data can be transferred. When the DTC is enabled and configured 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.

MSP-FET430X110, Schematic

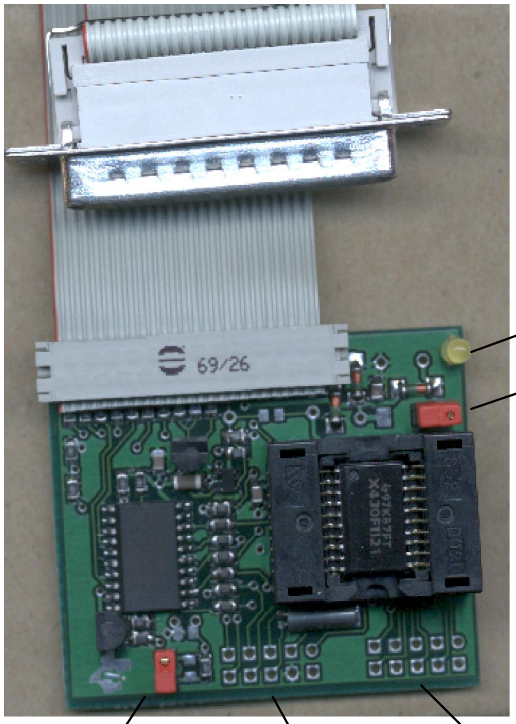


MSP-FET430X110, PCB Pictorials





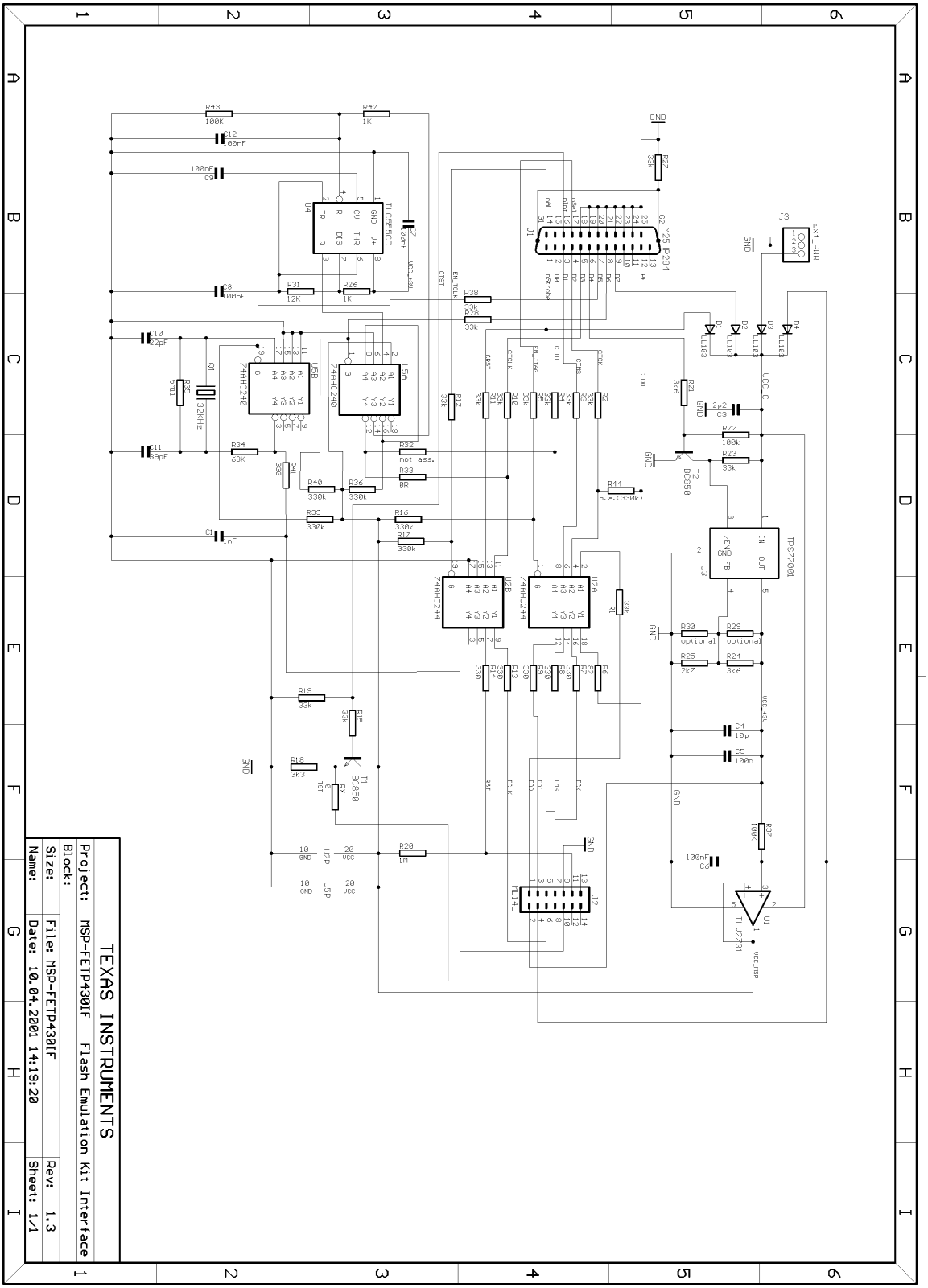
MSP-FET430X110, Photo



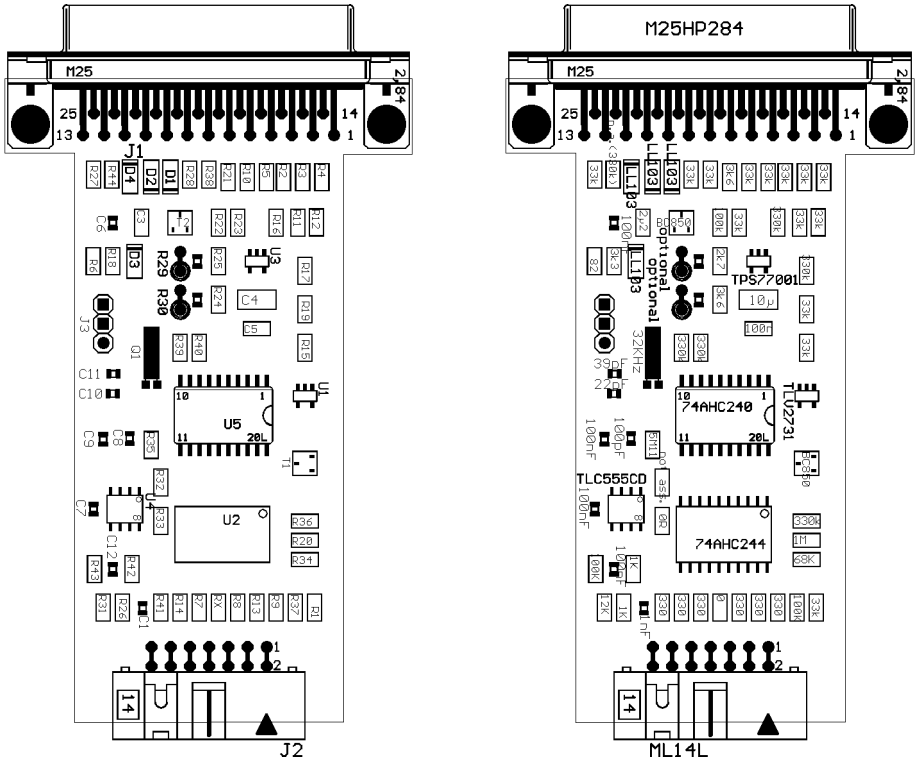
J2				
P2.1	RST	XOUT	P2.5	TST
P2.2	P2.0	XIN	VSS	VCC

J3				
P2.4	P1.1	P1.3	P1.5	P1.7
P2.3	P1.0	P1.2	P1.4	P1.6

MSP-FET430IF FET Interface module, Schematic



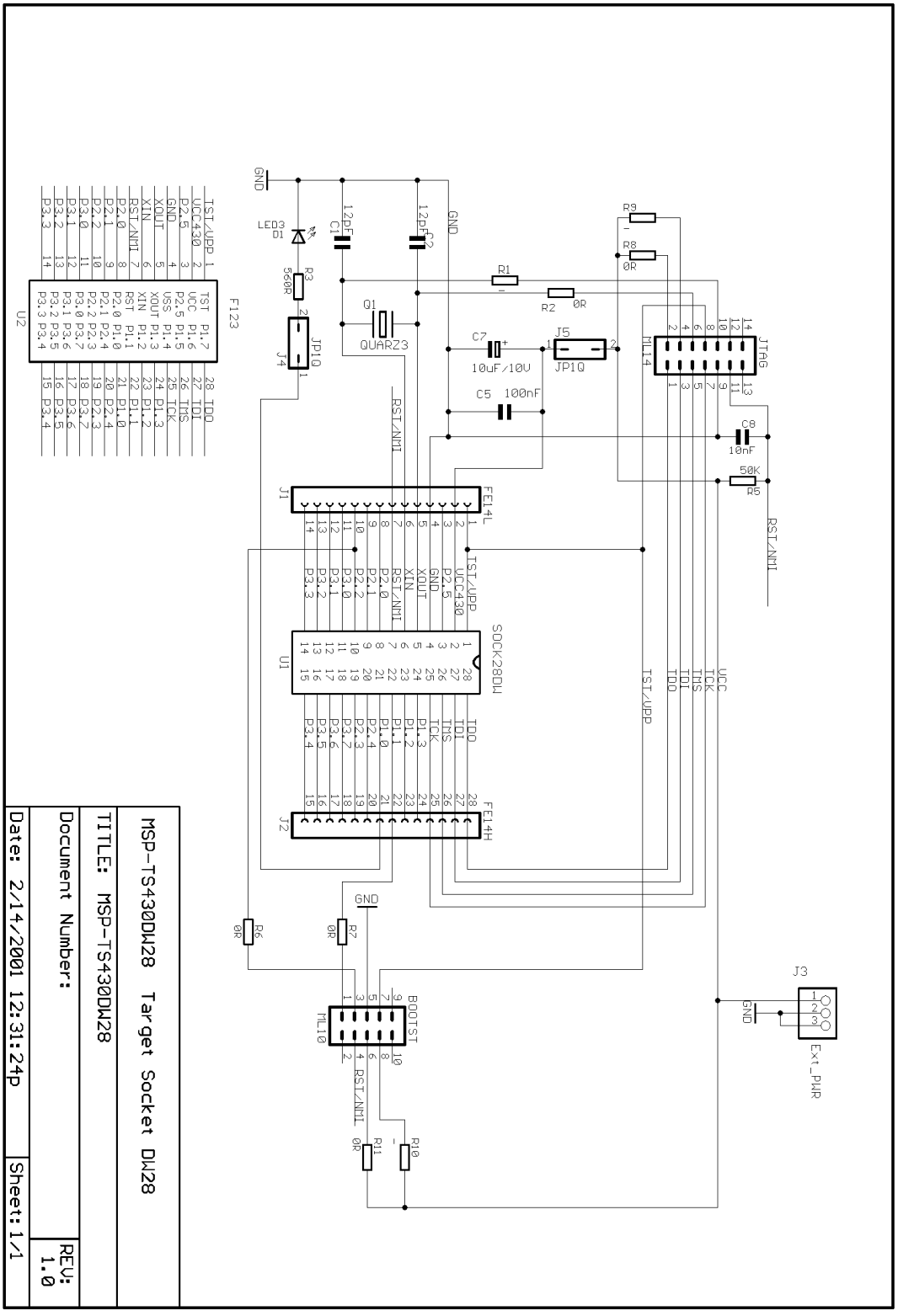
MSP-FET430IF FET Interface module, PCB Pictorials



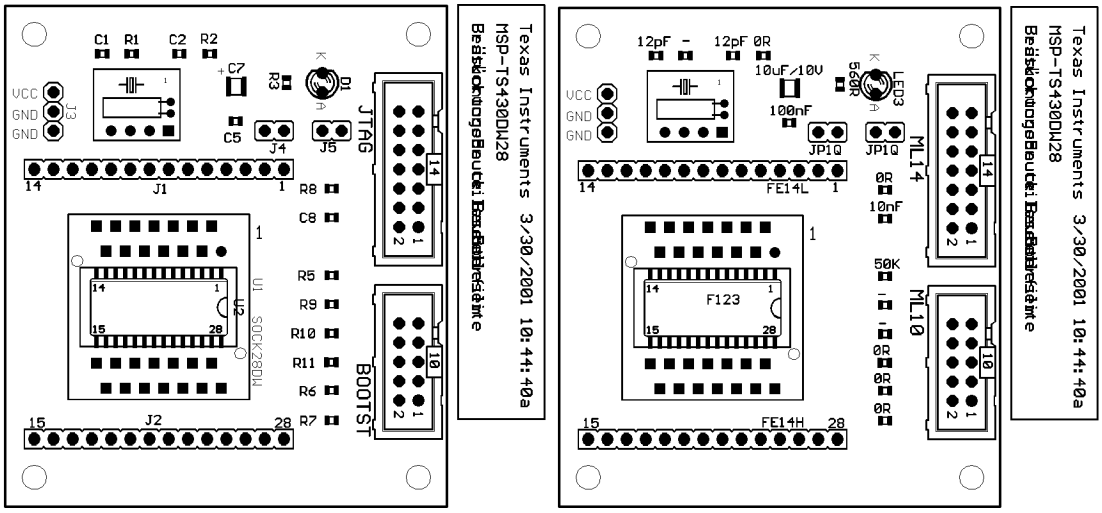
Note: The PCB of the FET Interface module provided with the tool is marked Rev.: 1.2. However, the electrical circuit corresponds to the Schematic Rev.: 1.3 above.

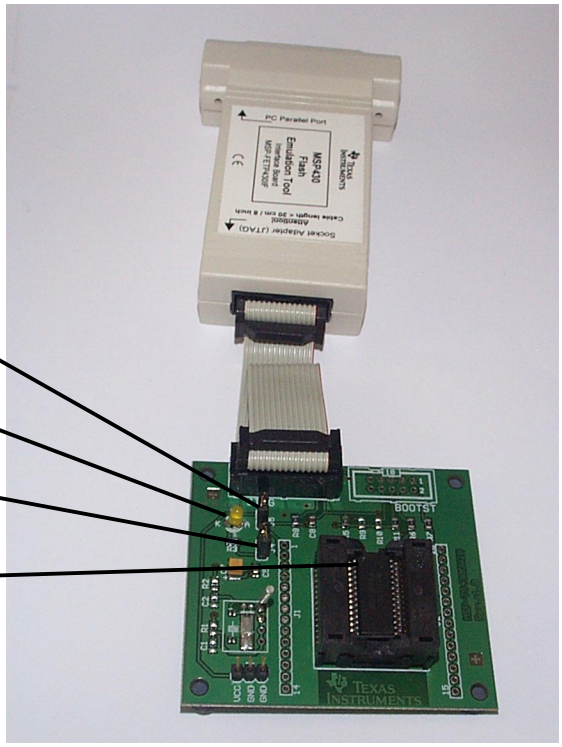
MSP-TS430DW28 Target Socket module, Schematic

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



MSP-TS430DW28 Target Socket module, PCB Pictorials and Photo





Jumper J5
Open to measure current

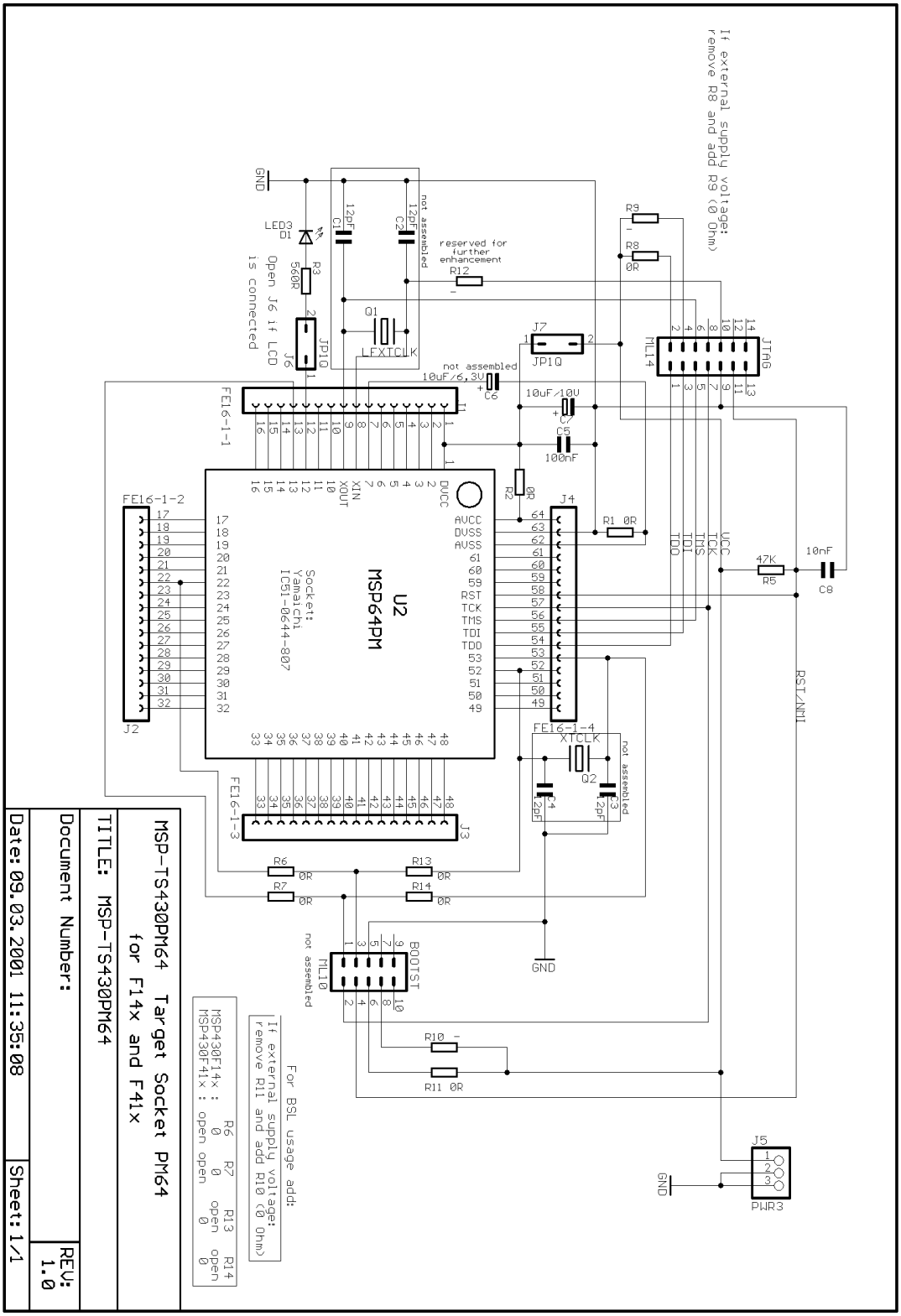
LED3 connected to P1.0

Jumper J4
Open to disconnect LED3

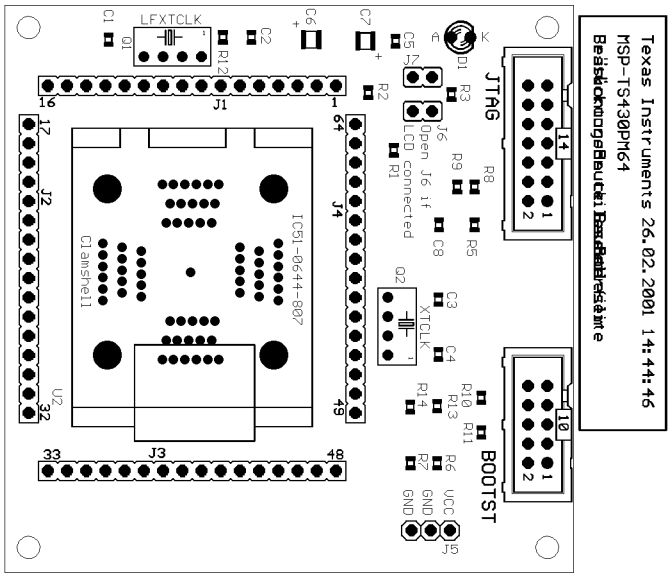
Orient Pin 1 of MSP430 device

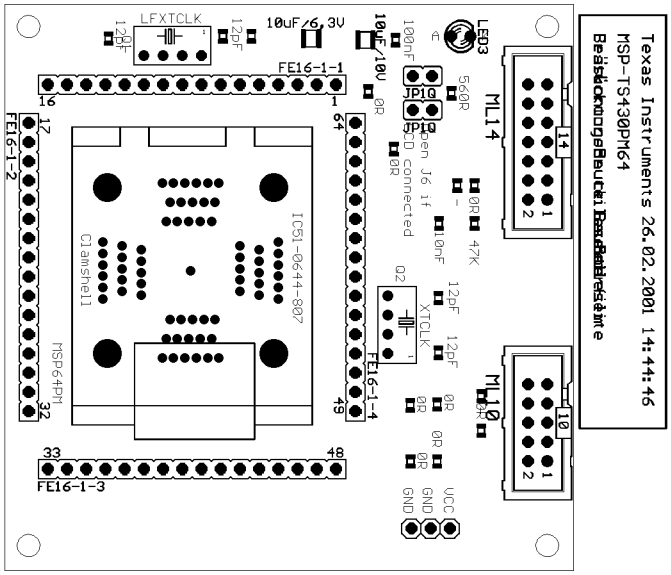
MSP-TS430PM64 Target Socket module, Schematic, Rev. 1.0

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



MSP-TS430PM64 Target Socket module, PCB Pictorials, Rev. 1.0





MSP-TS430PM64 Target Socket module, Photo, Rev. 1.0

LED3 connected to P1.0

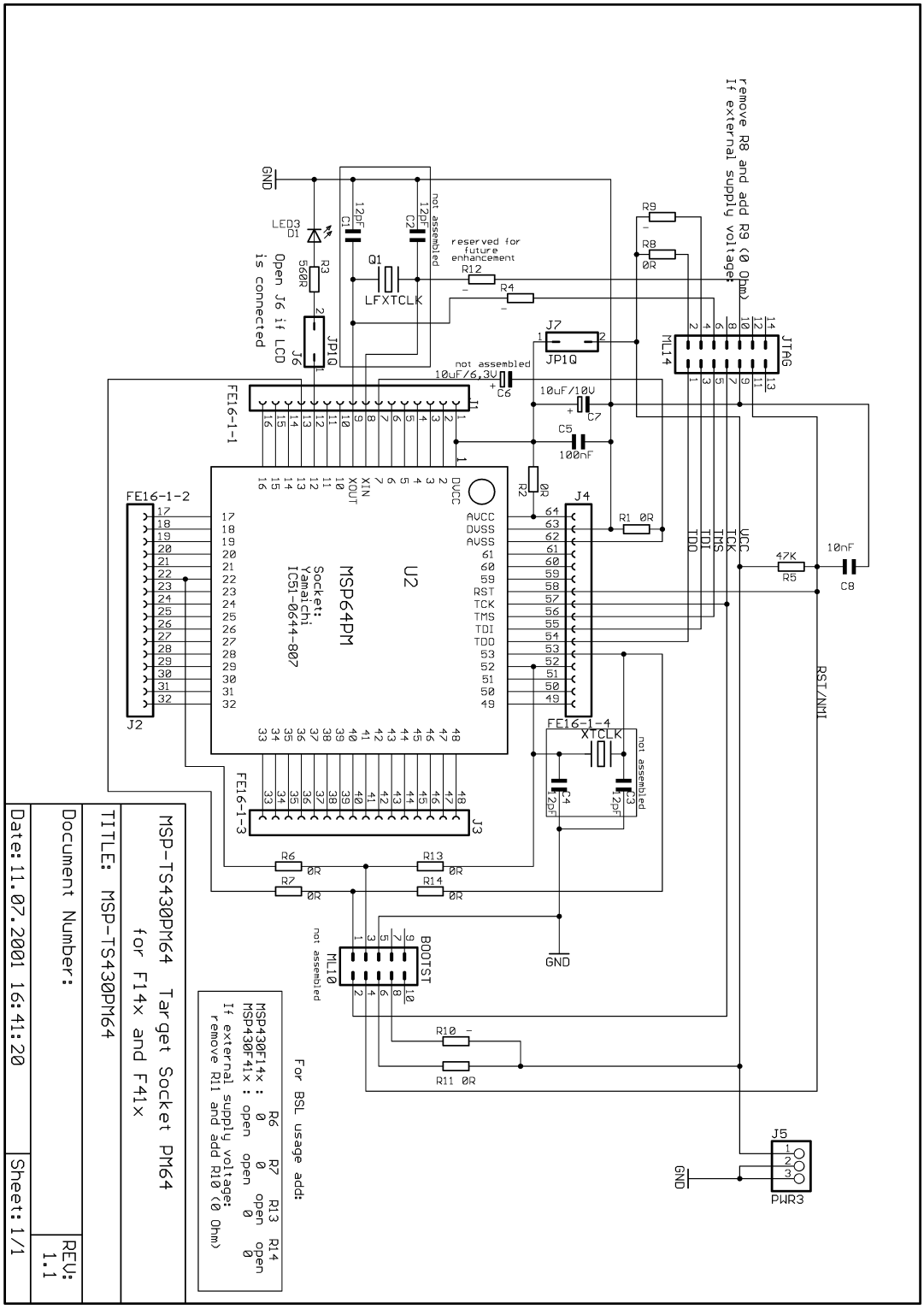
Jumper J7
Open to measure current

Jumper J6
Open to disconnect LED3

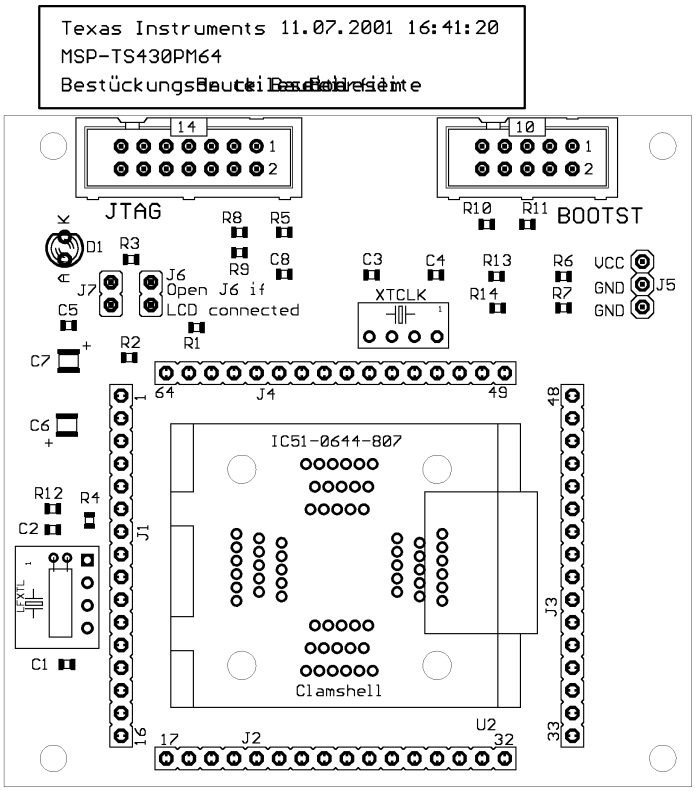
Orient Pin 1 of MSP430 device

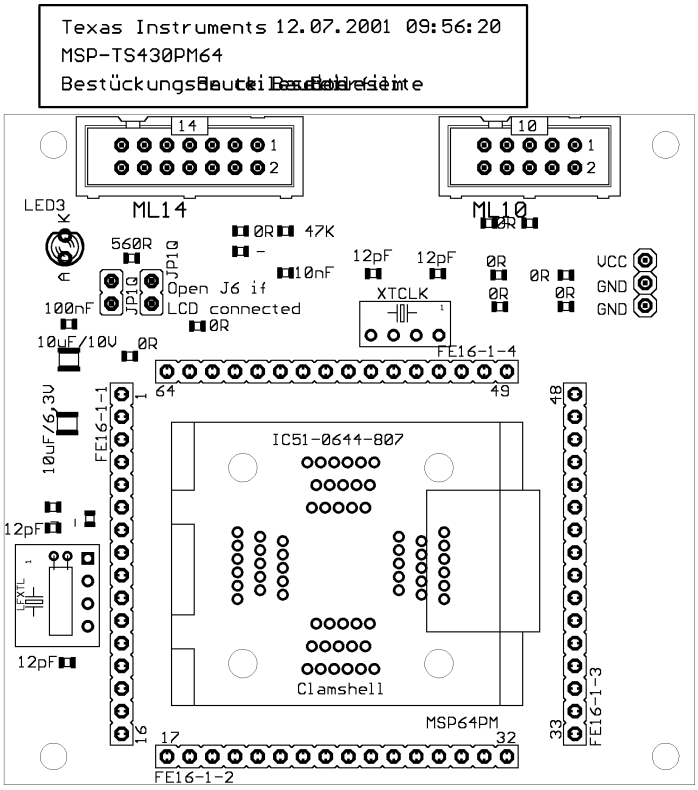
MSP-TS430PM64 Target Socket module, Schematic, Rev. 1.1

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



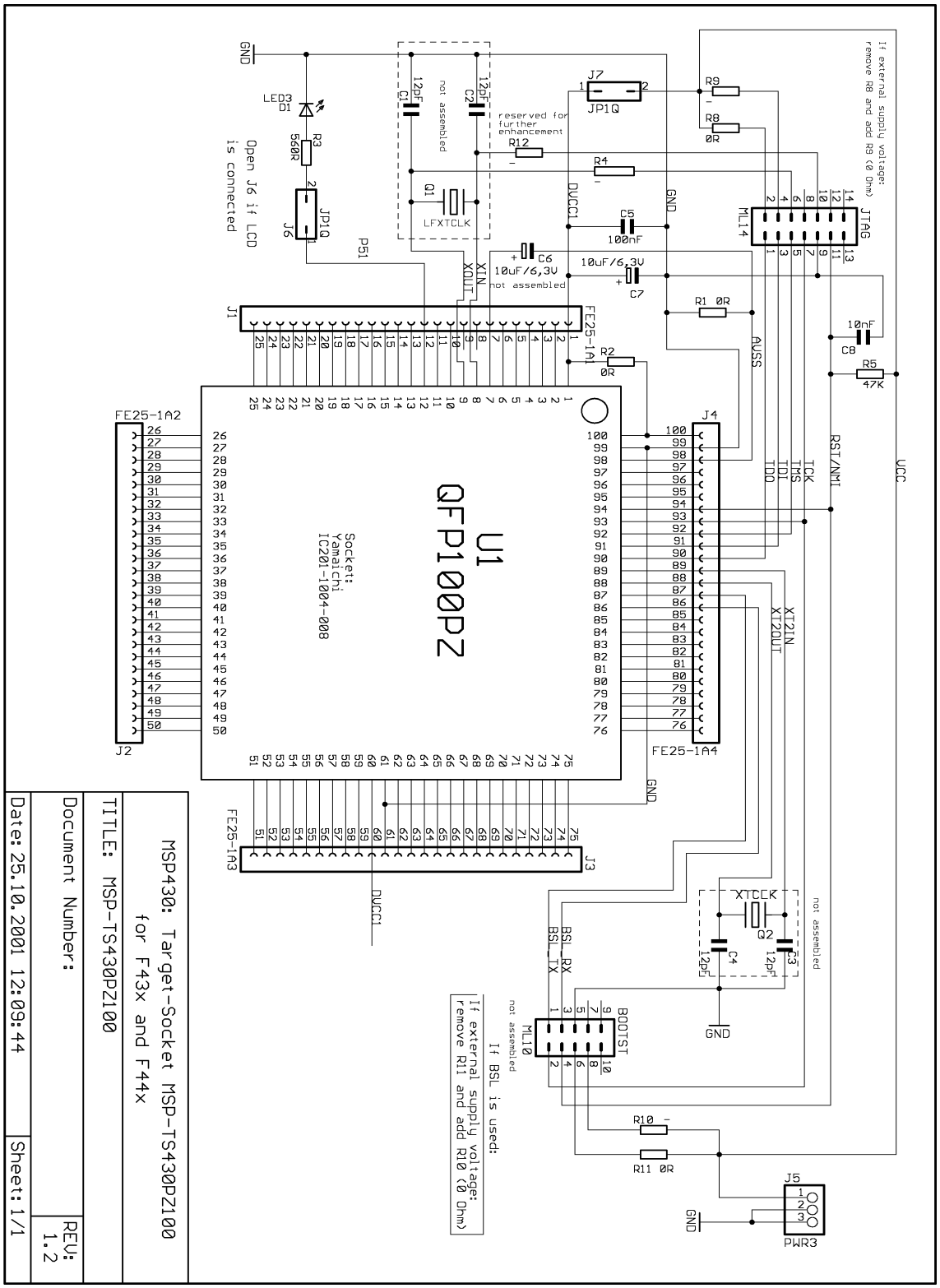
MSP-TS430PM64 Target Socket module, PCB Pictorials, Rev. 1.1



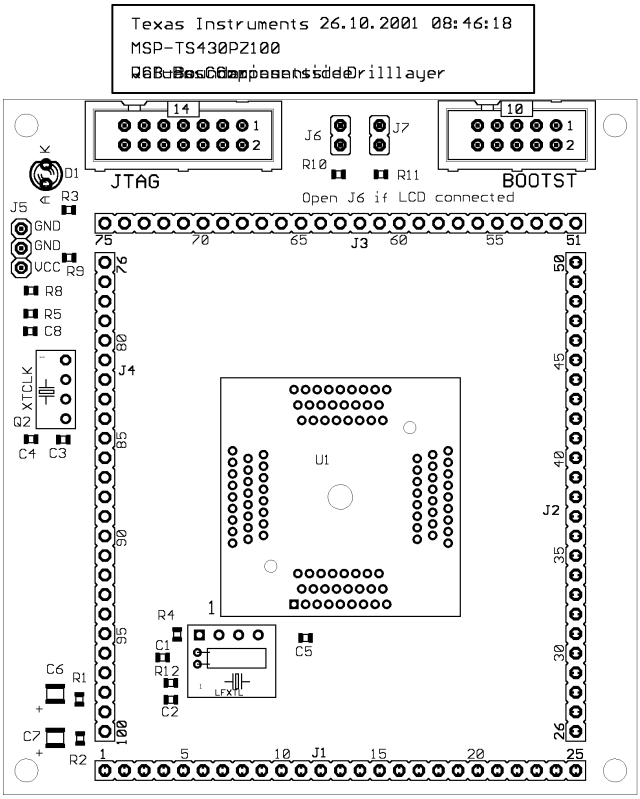


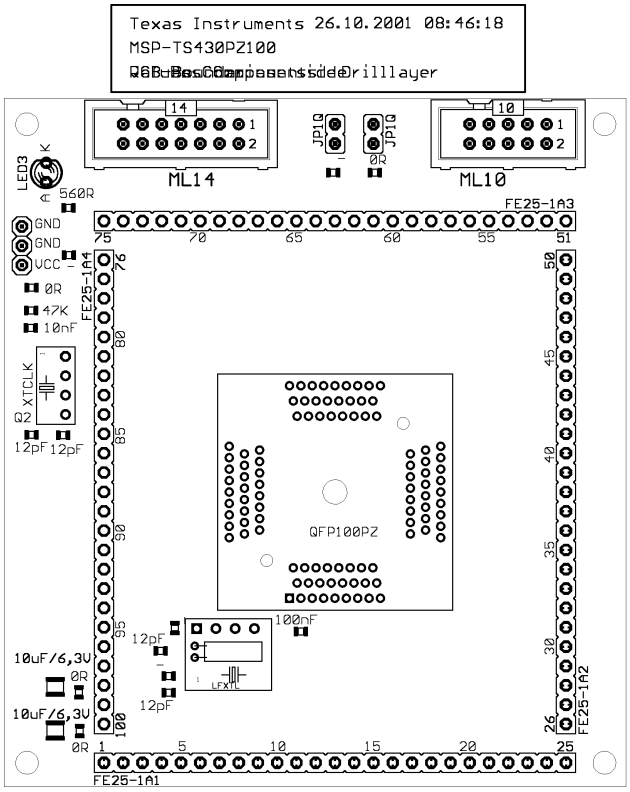
MSP-TS430PZ100 Target Socket module, Schematic

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



MSP-TS430PZ100 Target Socket module, PCB Pictorials





Signal-mapping when the MSP-TS430PZ100 module with an F449 is used to emulate an F44x or an F43x in an 80-pin package:

The 100-pin MSP430F449 can emulate the 80-pin MSP430F44x and MSP430F43x devices. The following table lists where the 80-pin device signals are located on the 100-pin device. Note that physical connections between some pins on the MSP-TS430PZ100 Target Socket module are required.

F449: Pin-#	F449: Signal	F4xx/80-pin: Pin-#	F4xx/80-pin: Signal	Connection required on the MSP-TS430PZ100 module (Connect pin xx and pin yy)
1	DVcc1	1	DVcc1	
2	P6.3/A3	2	P6.3/A3	
3	P6.4/A4	3	P6.4/A4	
4	P6.5/A5	4	P6.5/A5	
5	P6.6/A6	5	P6.6/A6	
6	P6.7/A7	6	P6.7/A7	
7	VREF+	7	VREF+	
8	XIN	8	XIN	
9	XOUT	9	XOUT	
10	VeREF+	10	VeREF+	
11	VREF-/VeREF-	11	VREF-/VeREF-	
12	P5.1/S0	12	P5.1/S0	
13	P5.0/S1	13	P5.0/S1	
14	S2	14	P4.7/S2	14-46
15	S3	15	P4.6/S3	15-47
16	S4	16	P4.5/S4	16-48
17	S5	17	P4.4/S5	17-49
18	S6	18	P4.3/S6	18-50
19	S7	19	P4.2/S7	19-51
20	S8	20	P4.1/S8	20-62
21	S9	21	P4.0/S9	21-63
22	S10	22	S10	
23	S11	23	S11	
24	S12	24	S12	
25	S13	25	S13	
26	S14	26	S14	
27	S15	27	S15	
28	S16	28	S16	
29	S17	29	S17	
30	S18	30	P2.7/ADC12CLK/S18	30-72
31	S19	31	P2.6/CAOUT/S19	31-73
32	S20	32	S20	
33	S21	33	S21	
34	S22	34	S22	
35	S23	35	S23	
36	S24	36	P3.7/S24	36-64
37	S25	37	P3.6/S25	37-65
38	S26	38	P3.5/S26	38-66
39	S27	39	P3.4/S27	39-67
40	S28	40	P3.3/UCLK0/S28	40-68
41	S29	41	P3.2/SOMI0/S29	41-69
42	S30	42	P3.1/SIMO0/S30	42-70
43	S31	43	P3.0/STE0/S31	43-71
44	S32			

45	S33		
46	P4.7/S34		
47	P4.6/S35		
48	P4.5/UCLK1/S36		
49	P4.4/SOMI1/S37		
50	P4.3/SIMO1/S38		
51	P4.2/STE1/S39		
52	COM0	44	COM0
53	P5.2/COM1	45	P5.2/COM1
54	P5.3COM2	46	P5.3COM2
55	P5.4/COM3	47	P5.4/COM3
56	R03	48	R03
57	P5.5/R13	49	P5.5/R13
58	P5.6/R23	50	P5.6/R23
59	P5.7/R33	51	P5.7/R33
60	DVcc2	52	DVcc2
61	DVss2	53	DVss2
62	P4.1/URXD1		
63	P4.0/UTXD1		
64	P3.7/TB6		
65	P3.6/TB5		
66	P3.5/TB4		
67	P3.4/TB3		
68	P3.3/UCLK0		
69	P3.2/SOMI0		
70	P3.1/SIMO0		
71	P3.0/STE0		
72	P2.7/ADC12CLK		
73	P2.6/CAOUT		
74	P2.5/URXD0	54	P2.5/URXD0
75	P2.4/UTXD0	55	P2.4/UTXD0
76	P2.3/TB2	56	P2.3/TB2
77	P2.2/TB1	57	P2.2/TB1
78	P2.1/TB0	58	P2.1/TB0
79	P2.0/TA2	59	P2.0/TA2
80	P1.7/CA1	60	P1.7/CA1
81	P1.6/CA0	61	P1.6/CA0
82	P1.5/TACLK/ACLK	62	P1.5/TACLK/ACLK
83	P1.4/TBCLK/SMCLK	63	P1.4/TBCLK/SMCLK
84	P1.3/TBOUTH/SVSOUT	64	P1.3/TBOUTH/SVSOUT
85	P1.2/TA1	65	P1.2/TA1
86	P1.1/TA0/MCLK	66	P1.1/TA0/MCLK
87	P1.0/TA0	67	P1.0/TA0
88	XT2OUT	68	XT2OUT
89	XT2IN	69	XT2IN
90	TDO/TDI	70	TDO/TDI
91	TDI	71	TDI
92	TMS	72	TMS
93	TCK	73	TCK
94	RST/NMI	74	RST/NMI
95	P6.0/A0	75	P6.0/A0
96	P6.1/A1	76	P6.1/A1
97	P6.2/A2	77	P6.2/A2
98	AVss	78	AVss
99	DVss1	79	DVss1
100	AVcc	80	AVcc

TI-to-IAR Assembler Directive Conversion

Texas Instruments makes a suite of development tools for the MSP430, including a comprehensive assembler and device simulator. The source of the TI assembler and the source of the Kickstart assembler are not 100% compatible; the instruction mnemonics are identical, while the assembler directives are somewhat different. The following section documents the differences between the TI assembler directives and the Kickstart assembler directives.

Segment Control

RSEG defines a Relocatable SEGment. A relocatable segment means that the code that follows the RSEG statement will be place *somewhere* in the region defined for that segment (in the .xcl file). In other words, the code can be "relocated", and you don't know (or care) where it's put. In the .xcl files provided with the FET, multiple segments are defined in the same memory regions. ASEG defines an Absolute SEGment. An absolute segment means that the code that follows the ASEG statement will be placed in the order it is encountered in the region defined for the segment (in the .xcl file). In other words, the placement of the code is fixed in memory. One significant difference between the new IAR assembler and the old TI assembler is the meaning of the ORG statement. In the old TI assembler, ORG would set the assembler code pointer to the specified absolute address. However, the IAR assembler uses ORG to set an offset from the current RSEG. Fortunately, if you don't use RSEG explicitly, it will default to 0 (zero) and your program will link as you expect (with your code at ORG). Be careful if you mix RSEG and ORG as ORG then becomes a relative offset. Use ASEG if you want the (absolute) behavior of the old TI ORG statement.

Translating Asm430 Assembler Directives to A430 Directives

1 Introduction

The following sections describe, in general, how to convert assembler directives for Texas Instruments' Asm430 assembler (Asm430) to assembler directives for IAR's A430 assembler (A430). These sections are only intended to act as a guide for translation. For detailed descriptions of each directive, refer to either *the MSP430 Assembly Language Tools User's Guide*, SLAUE12, from Texas Instruments, or *the MSP430 Assembler User's Guide* from IAR.

Note:
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.

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 below:

Character String	Asm430 Syntax (TI)	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"

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: UDATA0 for uninitialized data, CONST for constant (initialized) data and CODE for executable code. The table below 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 (TI)	A430 Directive (IAR)
Reserve size bytes in the .bss (uninitialized data) section	.bss	(1)
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)
Alignment on byte boundary	.align	(2)
Alignment on word boundary	.even	EVEN

(1) 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  UDATA0
LABEL:  DS    16
        RSEG  CODE
```

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

Additional A430 Directives (IAR)	A430 Directive (IAR)
Switch to an absolute segment	ASEG
Switch to a relocatable segment	RSEG
Switch to a common segment	COMMON
Switch to a stack segment (high-to-low allocation)	STACK
Alignment on specified address boundary (power of two)	ALIGN
Set the location counter	ORG

4 Constant Initialization Directives

Description	Asm430 Directive (TI)	A430 Directive (IAR)
Initialize one or more successive bytes or text strings	.byte or .string	DB
Initialize a 48-bit MSP430 floating-point constant	.double	(1)
Initialize a variable-length field	.field	(2)
Initialize a 32-bit MSP430 floating-point constant	.float	DF (3)
Reserve size bytes in the current section	.space	DS
Initialize one or more text strings	.string	DB
Initialize one or more 16-bit integers	.word	DW

(1) The 48-bit MSP430 format is not supported

(2) 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         /
```

(3) The 32-bit IEEE floating-point format, used by the C Compiler, is supported in the A430 assembler.

Additional A430 Directives (IAR)	A430 Directive (IAR)
Initialize one or more 32-bit integers	DL

5 Listing Control Directives

Description	Asm430 Directive (TI)	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.

Additional A430 Directives (IAR)	A430 Directive (IAR)
Allow/inhibit listing of macro definitions	LSTMAC (+/-)
Allow/inhibit multi-line code listing	LSTCOD (+/-)
Allow/inhibit partitioning of listing into pages	LSTPAG (+/-)
Generate cross reference table	LSTXREF (+/-)

6 File Referencing Directives

Description	Asm430 Directive (TI)	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 only included in the linked executable 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

7 Conditional-Assembly Directives

Description	Asm430 Directive (TI)	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
```

Additional A430 Directives (IAR)	A430 Directive (IAR)
Repeatable block assembly: Formal argument is substituted by each character of a string.	REPTC
Repeatable block assembly: formal argument is substituted by each string of a list of actual arguments.	REPTI

See also *Preprocessor Directives*

8 Symbol Control Directives

The scope of assembly-time symbols differs in the two assemblers. In Asm430, definitions are global to a file, but 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 (TI)	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 below:

```
MYSTRUCT:MACRO
         DS 4
         ENDM
LO       DEFINE  0
HI       DEFINE  2

         RSEG    UDATA0
X        MYSTRUCT

         RSEG    CODE
         MOV     X+LO,R4
         ...
```

Additional A430 Directives (IAR)	A430 Directive (IAR)
Define a file-wide symbol	DEFINE
Definition of special function registers (byte size)	SFRB
Definition of special function registers (word size)	SFRW

9 Macro Directives

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

Additional A430 Directives (IAR)	A430 Directive (IAR)
Create symbol, local to a macro	LOCAL (1)

(1) In Asm430 local symbols are suffixed by a question mark (?).

10 Miscellaneous Directives

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

(1) Warning messages cannot be user-defined. #Message may be used, but the warning counter will not be incremented.

(2) 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
```

(3) 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
        ...
```

Additional A430 Directives (IAR)	A430 Directive (IAR)
Set the default base of constants	RADIX
Enable case sensitivity	CASEON
Disable case sensitivity	CASEOFF

XXXXXXX: The #message directive may not be documented. I need to check to make sure and document it here, if not. The syntax is:

#message "the message" OR
#message 'the message'
These cause the following to be sent to standard out:
#message: the message

11 Preprocessor Directives

The A430 assembler includes a preprocessor similar to that used in C programming. The following preprocessor directives can be used in include files which are shared by assembly and C programs.

Additional A430 Directives (IAR)	**A430 Directive (IAR)**
Assign a value to a preprocessor symbol	#define
Undefine a preprocessor symbol	#undef
Conditional assembly	#if, #else, #elif
Assemble if a preprocessor symbol is defined (not defined)	#ifdef, #ifndef
End a #if, #ifdef or #ifndef block	#endif
Includes a file	#include
Generate an error	#error

12 Alphabetical Listing and Cross Reference of Asm430 Directives

Asm430 directive	**A430 directive**	**Asm430 directive**	**A430 directive**
.align	See *Section control directives*	.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)
.copy or .include	#include or $		LSTREP+ (loop blocks)
.data	RSEG	.mmsg	#message **(XXXXXX)**
.def	PUBLIC or EXPORT	.mnolist	LSTEXP- (macro)
.double	Not supported		LSTREP- (loop blocks)
.else	ELSE	.newblock	See *Symbol Control Directives*
.elseif	ELSEIF	.nolist	LSTOUT-
.emsg	#error	.option	See *Listing Control Directives*
.end	END	.page	PAGE
.endif	ENDIF	.ref	EXTERN or IMPORT
.endloop	ENDR	.sect	RSEG
.endm	ENDM	.setsect	See *Miscellaneous Directives*
.endstruct	See *Symbol Control Directives*	.setsym	See *Miscellaneous Directives*
.equ or .set	EQU or =	.space	DS
.eval	SET or VAR or ASSIGN	.sslist	Not supported
.even	EVEN	.ssnolist	Not supported
.fclist	LSTCND-	.string	DB
.fcnolist	LSTCND+	.struct	See *Symbol Control Directives*
.field	See *Constant Initialization Directives*	.tag	See *Symbol Control Directives*
.float	See *Constant Initialization Directives*	.text	RSEG
.global	See *File Referencing Directives*	.title	See *Listing Control Directives*
.if	IF	.usect	See *Symbol Control Directives*
.label	See *Miscellaneous Directives*	.width	COL
.length	PAGSIZ	.wmsg	See *Miscellaneous Directives*
.list	LSTOUT+	.word	DW

13 Additional A430 Directives (IAR)

Conditional-Assembly Directives	Constant Initialization Directives	Macro Directives
REPTC	DL	LOCAL
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	Preprocessor Directives	Symbol Control Directives
LSTMAC (+/-)	#define	ASEG
LSTCOD (+/-)	#undef	RSEG
LSTPAG (+/-)	#if, #else, #elif	COMMON
LSTXREF (+/-)	#ifdef, #ifndef	STACK
	#endif	ALIGN
	#include	ORG
	#error	

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

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

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.