PICD-430™

The Phyton In-circuit Debugger for the Texas Instruments MSP430F™ Microcontrollers

Phyton, Inc.

PICD-430™
The Phyton In-circuit Debugger for the Texas Instruments MSP430F™ Microcontrollers

Introduction

PICD-430™ is a development tool that supports program debugging and in-system programming for the MSP430F™ family of embedded microcontrollers from Texas Instruments.

PICD-430 can be hosted by any personal computer (PC) that has a USB port and works under control of Windows® 9x/ME/2000/XP.

To understand this manual you should read the Appendix A that describes all the basic terms and definitions used in this document.

Deliverables

The PICD-430 package includes the following items (some of them are optional):

1) PICD-430 unit
2) USB cable to connect a PICD-430 unit to the PC's USB port
3) Ribbon cable to connect a JTAG port of a PICD-430 unit to a target
4) CD ROM with the Project-430 software
5) A target board (option)
6) A wall-plugged regulated 5V power adapter (option)

The PICD-430 debugger unit is enclosed in a small plastic case. It has two microcontrollers on board. One of these supports communication via the USB port and the second converts high-level commands sent from the PC to low-level commands and data that are sent and received to and from a target microcontroller via the JTAG interface. Besides these microcontrollers, the debugger has an on-board voltage regulator that adjusts and transfers power to the target. This eliminates the need for an external power supply in most cases.

PICD-430 software is supplied on a CDROM and includes the following items:

- Project-430 integrated development environment (IDE)
- MCA-430 macro assembler from MicroCOSM-ST
- PDS-430 command set simulator
- PICD-430 debugger

- A set of application programs and user guides in electronic form

Project-430 IDE includes an editor, project manager, linker, librarian, object-hex converter and other utilities that help to handle the whole development process in one environment.

A target board (or boards), which can be optionally supplied with the PICD-430, has an MSP430 device that the PICD-430 uses as a target microcontroller. In addition, there's a JTAG plug for hooking up this board to the PICD-430 unit. The debugger emulates the behavior of the target microcontroller as it runs a debugging program while it is under control of the PICD-430. Some types of target boards can include LCD panels, LEDs, switches, buttons and other peripheral devices.

Main Features

- Supports MSP430F11x, MSP430F11x1A, MSP430F11x2, MSP430F12x, MSP430F12x2, MSP430F13x, MSP430F14x, MSP430F41x, MSP430F43x, MSP430F44x devices (new devices will be supported in future)

- The Project-430 IDE, MCA-430 macro assembler and PDS-430 simulator are supplied without charge

- PICD-430 integrates MCC-430 C compiler from MicroCOSM-ST or EW-430 C compiler from the IAR Systems, and provides project management and source-level debugging for applications written with these compilers (other compilers will be supported in future)

- Provides in-system flash programming, real-time run and single step program execution, breakpoints, and real-time tracing (tracing is available for selected derivatives only)

- High-level steps for C programs and low-level steps for assembly code

- Up to 8 unconditional breakpoints at an address, or a range of addresses, in code memory

- Up to 8 complex breakpoints/triggers on access to "bus events" and "register writes" which can be logically ORed, ANDed or sequentially programmed to break emulation or to trigger trace recording

- Enables examination and modification of internal resources of the target MSP430 MCU when execution is stopped

- For some MSP430 derivatives, the internal clock can be enabled after the break, so that the target MCU will continue to generate clocks for an external LCD or other peripherals

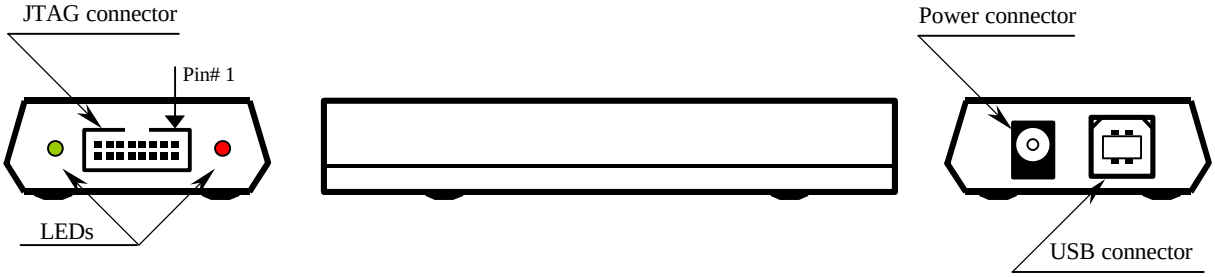
- The trace buffer records 40-bit frames displaying real-time signals on the internal MSP430 bus: there are 16 lines of address, 16 lines of data and 8 controls
- The depth of the trace buffer is defined by the particular type of MSP430 device (8 frames min)
- Ability to search filter and search trace frames
- The target MCU status can be monitored on-the-fly without stopping real-time execution
- The embedded programmable voltage regulator allows setting any voltage on the target microcontroller, in a range from 1.8V to 3.6V with 0.1V resolution.
- Works under control of Windows 9x/ME/2000/XP
- Communicates to a PC via a USB 1.1 interface

Software Installation

The installation procedure is standard for all users of PCs. Insert the CDROM supplied with the PICD-430 into the CD drive of your PC. After a few seconds you will get a start-up dialog on the screen. Click on the "Project-430" button and follow the instructions to install the software. If you download a software update from our website then invoke the *setup.exe* file to begin installation or software update. At the end of the installation the program will open a Project-430 folder with the tools and documentation icons (shortcuts).

Hardware Installation

The PICD-430 unit, its connectors and LEDs are shown below.



PICD-430

Connect the PICD-430 JTAG 14-wire ribbon cable to a JTAG male plug on the target; plug the opposite end of the cable into the JTAG socket on the face panel of the PICD-430 unit (see Appendix B).

If it's necessary, connect the PICD-430 unit to power via a 5V wall-plug regulated power adapter (read further about the conditions when you need external power to work with the PICD-430). Make sure that this adapter is regulated and that the center contact on its cable is positive! Connect a USB input of the PICD-430 unit to a USB port of your PC by means of the attached cable.

Choosing an Appropriate Power Scheme for the PICD-430

The PICD-430 unit itself consumes less than 40 mA at 5V, so it gets enough power from the USB port of a PC. The target can be powered either from an external (user's) power adapter or, if necessary, it can get limited power from the PICD-430 unit. The PICD-430 has an input for an external power adapter (5V/1A) that enables power transfer to the target via a built-in programmable regulator and the JTAG port.

If the target consumes less than 50mA you needn't be concerned about supplying it from a separate power source – the PICD-430 unit will transfer enough power from a PC's USB port to the target via the built-in programmable regulator and the JTAG port.

If the target consumes between 50 mA and 400mA you can supply the PICD-430 unit from a 5V regulated power adapter and the target will get power passing through the PICD-430 unit.

If the target consumes more than 400mA you should supply it from an external power source and make sure that Vcc on the target MCU is in the range specified by Texas Instruments for the MSP430. If the target, and particularly the target MCU, gets power from an external (user's) power adapter, then you do not need to supply the PICD-430 unit from a 5V power adapter.

Take into account that a single USB port cannot generate more than 500 mA of output current. Therefore, if you connect the PICD-430 unit to a hub that has other connected USB devices, you may limit the power received by the PICD-430 unit and the connected target.

When you connect the PICD-430 unit to a USB port the red LED on the PICD-430 should go on.

Starting the PICD-430

When you start the "Phyton Project-430" program you are prompted to start either the PDS-430 debugger/simulator or the PICD-430 in-circuit hardware debugger. Here we describe how to operate with the PICD-430. Note that both debuggers have very similar

user interfaces and methods of control; therefore this manual will be helpful for operating with the PDS-430 as well.

Invoke the PICD-430 in-circuit hardware debugger (you can also invoke the PICD.EXE program from the Phyton folder). A green LED on the face panel should blink. This indicates that the PICD-430 unit is communicating with the PC. Also, you will see a communication dialog ("PICD-430 Communications") that will prompt you to specify the type of interface. When working with the PICD-430, select USB as the "Communication Type." "Printer Port" is an alternative choice that is intended to drive the "MSP430Fxxxx Flash Emulation Tool" produced by Texas Instruments.

If you get an error message showing that the program could not establish a connection to the PICD-430 unit, check whether you properly set the communication parameters. Also, check the power scheme and connection of the cable to the USB slot on your PC. After the PICD-430 successfully establishes communications to the PC, it brings to the screen the Hardware Configuration dialog.

First, select the type of the target MCU. The PICD-430 automatically identifies the target MSP430 subfamily or group that's connected to the PICD-430 unit. However, you should manually select the particular microcontrollers that's installed on the target. For example, MSP430 group = MSP430F44x, target MCU = MSP430F449.

Second, select one of the power schemes. Select "Emulation MCU Power Management" and choose either the "Follow target board voltage" or "User-specified value" and type any value between 1.8V to 3.6V. The embedded voltage regulator will precisely support the entered value on the Vcc pin of the target MCU. If you select "Follow target board voltage" this will eliminate any conflicts between voltages generated by the PICD-430 unit and the target MCU. This is safer but can be used only in case you supply the target from a separate power source.

After all the settings are done, click OK to enter the PICD-430 program. It will open the main window and will open a dialog that offers to load one of the demo examples. Working with these examples will allow you to familiarize yourself with the product's features and with the GUI controls and menus. Or you can close the example dialog and open a new project. If you would like to load some examples for evaluation later, you can always invoke the list via the *Main menu > User > Show example* list.

Configuring the PICD-430

All PICD-430 settings are grouped under the "Configure" submenu of the Main menu. Prime and most important settings are accessible via the *Hardware Configuration* dialog. To open this dialog select the *Main menu > Configure > Hardware Configuration*. The dialog box has three tabs: *Target, Target MCU Power Management*, and *Options*. It also has two panes. Choose a parameter to set in the left pane and then type it in, or just pick from the list in the right pane.

Target

The PICD-430 automatically detects to which subfamily a target MCU belongs. Each subfamily includes a number of the MSP430 derivatives shown in the Target Chip field. The program prompts a full list of the target microcontrollers for each subfamily. Pick the particular type that's set on your target board.

Target MCU Power Management

The PICD-430 has an embedded voltage regulator that controls the Vcc applied to the target microcontroller. You can set one of two alternative modes of power management for the target microcontroller.

Follow Target Board Voltage

This mode is recommended when the target is supplied from an external power adapter. In this mode an embedded programmable voltage regulator constantly monitors the real level on the Vcc pin of the target microcontroller, and accurately adjusts levels on its JTAG logic outputs. If you choose this mode you must supply the target from an external power supply; otherwise the PICD-430 will not work. By setting this mode you avoid conflicts between logical signals on the target and on the PICD-430 unit. This protects both parts of the system from damage.

User Specified, Value in Volts

This is a default mode of the power management. In this mode you can enter any voltage value in a range from 1.80 to 3.60. An embedded regulator will set, monitor and stabilize this value on a Vcc pin of the target microcontroller with an accuracy of +/- 0.1Volt. Actual voltage levels on the target microcontroller and the target board are always shown under the programmed value.

It is possible to use this mode when you supply your prototype from an external power adapter. Then, it's possible that the voltage generated by the PICD-430 unit will differ from the voltage applied to the target microcontroller. To prevent any damage of the target the PICD-430 program blocks any attempts to mistakenly set voltage that differs from the level measured on the Vcc pin of the target MCU by more than -/+ 400 mV and issues the warning shown here.

You can click on the *Hardware Configuration* button to correct the voltage settings. By clicking Yes you accept all the consequences of your settings.

Options

This dialog allows setting some options that are specific for each particular MSP430 subfamily.

Enable Hardware Reset

If this option is checked it enables direct passing of the hardware reset signal from the JTAG connector to the input RESET of the target microcontroller. If the option is unchecked then the PICD-430 will reset the target microcontroller via the JTAG interface by means of the *Enhanced Emulation Module* (EEM).

Reset Delay, ms

Here you can set a programmable delay between the time when the PICD-430 generates a hardware reset and the time that this signal appears at the RESET input of the target microcontroller.

General Clock Control in Break Mode

Here you can block clocking of some target microcontroller peripherals (timers, ADC, etc.) in the break mode, when the target microcontroller is not running in real-time mode.

Emulation Mode

This option, *embedded into the MSP430F449 derivative in a 100-pin QFP package only*, enables you to emulate LCD control for the 100-pin MSP430F43x and 80-pin versions of the MSP430F44x and MSP430F43x derivatives. See the pin mapping in Appendix E.

Development with the PICD-430 IDE

After you have set up the PICD-430 and clicked OK, the program opens the main window with a few windows inside – usually the *Source*, *Disassembler* and *Project* windows. By default the program saves all configuration parameters (including windows layout) and restores them next time you start the PICD-430. The program always opens an example list menu that offers to load some project examples for evaluation and familiarization with the PICD-430 functions. Keep in mind that all the changes in these examples will be saved automatically when you quit the program unless you switch off the *Autosave session on exit* function before closing the program. For this, go to *Main menu > Files* and toggle the function line.

The top line of the main window (just under the window title) shows submenus: *File, Edit, View, Run, Breakpoints, Configure, Project, Commands, Scripts, Windows* and *Help*. The main toolbar below includes buttons that trigger some frequently used commands. Every window has its own local menu, which can be invoked by setting the mouse cursor over a particular window and then clicking the right mouse button. Local menus also display frequently used commands and settings associated with the window.

There are two modes of use for the debugger:

- Project-level support
- Source-level debugging

Project-level Support

Project-level support is the most convenient way of developing programs because it allows you to use the complete set of development tools as an integrated development environment (IDE). These include entering raw source code, editing, compiling, linking, debugging, and finally writing a completely debugged body of code into the target MCU's flash memory.

Here we will mostly describe how to use the PICD-430 for the project-level kind of development. It enables you to set up a compiler, linker and other tools right from the Project-430 IDE. Then a project manager will automatically invoke the necessary tools, issue error messages, highlight lines of invalid source code, etc. A built-in editor enables you to promptly correct your code right in the Source window and to start emulation

immediately. The whole development process becomes faster and easier. The Project-430 IDE and PICD-430 support project mode development for C compilers and assemblers from IAR Systems (EW-430) and MicroCOSM-ST (MCA-430 and MCC-430). A macro assembler, MCA-430, is a part of the Project-430 IDE. It is available at no charge and is always included with the IDE tool set.

Source-level Debugging

If you prefer to use your own favorite editor and compiler you can load your program for debugging via *Main memu > File > Load program for debugging*... Then enter the file name into the *File name* field or use the *Browse* button to find the file. Specify the file format, its start address in the destination memory, and the memory type – code or data. Click OK and you will see the program text in the *Source* window.

PICD-430 Windows

The PICD-430 allows you to open the following windows to control the development process:

- ***Source (combined with Editor)***
- ***Disassembler***
- ***Project***
- ***Message***
- ***Watches***
- ***AutoWatches***
- ***Inspector***
- ***Memory Dump***
- ***Memory Layout***
- ***Code Browser***
- ***Peripherals***
- ***Tracer***
- ***Console***
- ***User***
- ***I/O Stream***
- ***Script Source***

To open a particular window go to *Main Menu > View* and then click on the desired window type. You can open an unlimited number of windows, even of the same type, except for the *Source* window that opens automatically when the program is loaded. Every window has its own local menu that offers a set of frequently used commands associated with this particular window, as well as an individual toolbar with buttons for frequently used commands. Data displayed in the PICD-430 windows can be set in

several formats (hexadecimal, decimal, binary, ASCII); select the most convenient for viewing.

Source Window

In this manual, we use the term *Source window*, because it reflects the main purpose of this type of window: to display the program source text. Actually, the *Source* window can display a text file of any type and provide full-scale editing features. From this point of view, it can be treated as the *Editor* window. As mentioned above, when you load a program or a project for debugging, its text is displayed in the *Source* window. Any action on the text is treated as switching to the *Editor* mode. Starting any command (*Make, Run, Step,* etc.) is treated as switching to the debug mode.

The *Source/Editor* window can have a pane on the right that displays information according to the current window mode. For writing the source text, the right pane displays the automatic word completion list. For the debugging mode, the right pane displays the variable values located in the corresponding lines of the left (main) pane. To have the right pane present in a newly opened *Source* window, set up the *Auto word/AutoWatch* pane option in the *Editor Options* dialog box. Each *Source* window has a toolbar button that toggles the right pane on and off.

The instruction addresses, which correspond to the current value of the target processor PC (Program Counter) and the set breakpoints, are highlighted. A blue strip highlights a line of the current assembly instruction. The *Quick watch* feature works as follows: if you point at a variable name in the window, a small box containing the value of the variable is displayed next to the cursor. This box disappears when the cursor is moved.

Here is a list of the commands associated with the *Source* window:

- Run to cursor
- Inspect
- Add to Watches window
- Check variable
- View disassembly
- Functions list
- Toggle code breakpoint
- New PC
- Display from address
- Origin
- Pick Source file
- Compile
- Show next compiler error
- Mixed with disassembly

Disassembler Window

The *Disassembler* window displays a disassembled text of the program. If the loaded code has symbolic information, you can also see the code labels and names of functions; the instruction operands are replaced with the symbol names, where possible (this feature can be disabled). The *Disassembler* window has the on-line assembler.

Instructions that match the start addresses of the C operators are marked with squares on the left side. Instruction addresses that correspond to the current value of the target processor PC and the set breakpoints are highlighted. A blue strip highlights a line of the current assembly instruction. The *Quick watch* feature works as follows: if you point at (hover over) a variable name in the window, a small box containing the value of the variable is displayed next to the cursor. This box disappears when the cursor is moved.

Windows for Watching Internal Resources and Program Objects

The PICD-430 allows watching and modifying contents of the target microcontroller's registers, memory, and stack, as well as the variables, arrays, structures and other objects defined in the loaded application program.

AutoWatches Window

The names and values of the variables that are currently visible in the Source window are automatically displayed in the right pane of the Source window and in the *AutoWatches* window if it is open. If no *Source* window is opened, then the symbolic names are taken from the *Disassembler* window. If no *Disassembler* windows are opened, then the *AutoWatches* window is blank.

The first lines of the *AutoWatches* window display the variables that appear in the *Source* window line corresponding to the target processor PC value. The list of names of variables is denoted with the red line.

Below the lines with variables, the *AutoWatches* window displays the names of the current function parameters and its local variables. The *AutoWatches* window sorts and displays the variables according to their significance; that is, by their proximity to the debug context. The list in the *Autowatches* window automatically changes as you scroll through the *Source* window. The current object is highlighted. You can move the highlight either using the mouse (a single click moves the highlight bar to the current mouse pointer position) or the arrow keys of the keyboard.

Watches Window

The *Watches* window displays the variable values. The variable value is defined by its name. The Expressions can be used as names. Unlike the *AutoWatches* window, the variables ("watches") are manually added to and removed from the window. The complex objects (such as structures or arrays) are displayed in one line. If an object is complex and it is necessary to view it in detail, then use the *Inspect* window command for this object.

Use the following ways to add a variable to the Watches window:

- With the *Add Watch* command
- With the *Copy to Watches window* command of the *AutoWatches* and *Memory Layout* windows
- With the *Add to Watches window* command of the *Source* window
- With the *Add Watch* command of the *Commands* menu in the *Mmain* window
- By clicking the appropriate button on the control bar of this window

The current object is highlighted. You can move the highlight either using the mouse (a single click moves the highlight bar to the current mouse pointer position) or the arrow keys of the keyboard.

You can add and delete watches, and inspect and modify them via the local menu's commands. To change the value of a highlighted variable, you can type the new value from the keyboard. The prompting dialog box will appear automatically.

Peripheral Windows

The *Peripheral* device window shows the status of the target microcontroller's special function registers (SFRs) and allows adding a specified SFR to the *Watches* window to monitor its value. It is expected that later versions of the PICD-430 system will allow other peripherals to be watched.

Memory Dump Window

The *Memory Dump* window displays the memory dump of the target processor memory. Since memory is one of the most important processor resources, PICD-430 provides several options for the data presentation. Note that you can open as many memory dumps as you want.

When the window is active (when it has the keyboard input focus) a caret appears. It can be moved with the cursor keys or the mouse: a single click of the left mouse button places the caret at the position of the mouse cursor.

Working with Projects

PROJECT is a set of parameters that completely describe your task. This set includes:

- Project *name* and comments on the project
- Names of all *source* files used in the project
- Project *objective* (a promable file or a library)
- Operation modes of the built-in *make* administrator
- Tool set used for this project (the compiler, linker, assembler and librarian) and their configuration
- Default paths to the library files; the *include* files and executable compiler files
- Current state of the project and the desktop configuration

You do not need to remember compiler options and linker command lines, because all of them are set up through user-friendly dialog boxes. The PICD-430 IDE keeps the specified parameters throughout the development process until you change them. As mentioned above, the PICD-430 program includes a few examples of projects written with different compilers. You can always invoke a list of these examples through the *Scrip* submenu of the *Main menu*.

Creating a New Project

To create a new project, use *Main menu* > *Project* > *New*. In the *Create New Project* dialog, specify parameters for this project in the *General Properties* group, select a cross-tool kit in the *Cross-tools* group and click OK. The previously loaded project, if any, will be automatically saved. Other project options will be set to their default values. Later, you can change the cross-tools and their parameters (see How to Set up Cross-tools).

General Properties

Project Name

The name of the project should not include symbols that aren’t permitted in file names, such as ‘/’, '?', '*'.

Project Directory

When you work with multiple projects and use the same source files for different purposes, it is strongly recommended that you use an individual directory for each project, and that you specify different paths for their output files in the *Folders* group of

the *Create New Project* dialog. This ensures that the *Make* utility works correctly, since any ambiguity relating to object file versions will be eliminated. You do not need to create the individual directories – the program will do this for you if necessary.

Description

This is an optional field that describes a new project.

Target Microcontroller for the Project

After you specify the type of target microcontroller, the IDE will always check and warn you if there is a conflict between the setting in the project and the setting in the hardware configuration.

Cross-tools

In order for third party cross compilers and linkers to work properly with PICD-430, you should set up these tools properly. First, highlight the *Cross tools* line in the left pane and select the name of the compiler vendor. Then highlight the individual tools (C compiler, Assembler, Linker, Librarian) at left and set up the highlighted tools, following the tools vendor's instructions. Read further about preparing the files for source-level debugging.

Memory Areas

Here you can redefine addresses for Code and Data memories of the target microcontroller.

Folders

Highlight the *Folder* line at left and specify paths to includes files, default libraries and binary components (compiler, linker, assembler, etc.). You can browse for the destination subdirectories (folders) or search for reserved names and file extensions for the selected tools.

Make Options

Here you can define parameters for the *Make* utility that will compile and link your projects.

Managing Projects

When you click OK, completing creation of a new project, the program will open a blank window with the title "Project: *name of your project.*" Now you need to gather programs comprising your project. A toolbar on the top of the Project window offers the most frequently used command buttons.

Adding Files to a Project

Click on the *Add* button to get the *Add file to project* dialog. Then pick the files that you would like to add to the project. If the *Copy files to project directory* option is checked, then selected files will be copied into the project directory unless they are already there.

Editing Project Files

In order to view and modify text of a file that's included in the project, highlight this file and click on the *Edit* button on the window's toolbar.

Save Projects

To save a project on disk, select *Main menu > Project > Save*. The IDE automatically saves an open project on exit. In the Project-430 IDE, project files have .ide extensions (for example *J_Smith_project_3.ide*).

Open an Existing Project

Use the *Main menu > Project > Open...* command to open an existing project stored on a disk. Type in the project name or browse a PC disk to load the project you need (select a file with an .ide extension).

Building Projects

To launch compilation and link steps for a loaded project, press the *F6* button on your keyboard or click the *Make* button on the main toolbar (second from left). If the process runs to completion with no errors you will get an OK status message. Otherwise the program will show you that there were some errors and/or warnings and will automatically open the *Messages* window that contains the error messages. If you click on an error message in the *Messages* window, then a program string in the *Source* window that's associated with the error will be highlighted. Then you can correct your source text and immediately recompile the project.

To go directly to debugging without the compile/make steps, you can start the target program in the run mode or the single step mode. If the program has not yet been compiled, the PICD-430 project manager will detect that condition and will automatically invoke the compiler and linker. If errors are detected, you will get error messages and you will be prompted to correct the program sources. If there are no errors, the output file will be written into the target microcontroller flash memory and the program will automatically start in the run mode or single step mode.

Program Debugging

The PICD-430 is a powerful tool that provides all necessary debugging operations: loading a program for debugging, program execution in several modes, examining and modifying variables and other program objects, setting breakpoints, tracing, etc.

Preparing Programs for Source-level Debugging

Only the compilers listed below can be used for source-level debugging. The object programs generated by other compilers can also be debugged if a hexadecimal executable file of the program or binary memory image is available, but in this case the powerful features associated with source-level debugging are not available. Here is the list of compilers supported by the PICD-430:

- EW-430 C compiler from IAR Systems
- MCC-430 C compiler from MicroCOSM-ST

The MCC-430 itself is a command-line compiler that does not have an IDE of its own, but it has been integrated with the Phyton Project-430 IDE. All options are preset to work with the Phyton IDE, which automatically generates all information necessary for source-level debugging.

Setting EW-430 from IAR Systems

The EW-430 compiler from IAR Systems is a very popular tool for development of programs written for MSP430 microcontrollers. Many developers use it, together with the Embedded Workbench IDE. If you prefer to continue to edit and compile your programs in the EW-430 environment, then you can still use the PICD-430 for source-level debugging. Here's how it's done.

- Open the EW-430 IDE.
- Create a new project and add your source files to it.
- Select *Debug* in the *Targets* field of the *Project* window.
- In the Project window click the right mouse button onto a very first string *Debug*.

- Select *Options* from the pop-up menu. A dialog will appear for setting the cross-tools options.
- In the *Category* list select the 'ICC430' line. It will bring to the screen a list of the compiler's options. Click on the tab *Debug*.
- Check the *Generate debug information* option.
- Set the option '2 NOP's' in the field at the bottom that's labeled *Code added to statements*.
- In the *Category* list select the 'A430' line. It will bring to the screen a list of the assembler's options. Click on the *Debug* tab.
- Check the *Generate debug information* and *File references* options.
- In the *Category* list select the 'XLINK' line. It will bring to the screen a list of the linker's options.
- Check the *Debug info* option. Select 'Include all' in the field *Module-local symbols*.
- Click OK to complete settings.

Now, after the project is linked, EW-430 will create an output file with the extension .D90. By default EW-430 creates this file in the subdirectory "Debug\Exe". This is the file that should be loaded to PICD-430 for debugging.

For debugging, click on *Main menu > File > Load program for debugging*, then select IAR System UBROF format, and type in or browse a name of the file with extension .D90. Then click OK to load the file into the PICD-430.

Program Execution

A list of the commands used to initiate execution in several modes is presented in the "Run" submenu. Some of these are duplicated on the main toolbar.

- *Step* (or *High-Level Step* or F7) executes only one high-level C language operation in the user program.
- *Step Over* (or *High-Level Step Over* or F8) executes one step of the program at the "high level" without breaking at the functions called from the current operator.
- *Low-level Step* (or Ctrl-F7) executes only one machine instruction.
- *Low-level Step Over* (or Ctrl-F8) executes only one machine instruction without breaking at the functions called from the current operation.
- *Run* (F9) starts running the target program in the continuous mode. The program runs until a breakpoint is encountered or the program is stopped in some other way. Click on the same button (F9) to stop running.
- *Run to Address* command starts running the program in the continuous mode, up to a specified address.

Breakpoints

The PICD-430 allows unconditional and conditional breakpoints (also known as complex breakpoints).

Unconditional Code Breakpoints

Each MSP430F subfamily has its own specific number of code breakpoints:

MSP430F subfamily	Number of breakpoints
MSP430F11x1	2
MSP430F11x2	2
MSP430F12x	2
MSP430F12x2	2
MSP430F13x	3
MSP430F14x	3
MSP430F41x	2
MSP430F43x	3
MSP430F44x	8

Single (also simple) breakpoints are set at one program address. You can set and clear the breakpoints in an entire address range. This is useful for tracing a program's "flights" to unavailable addresses. The code breakpoint is a breakpoint of the "break-before-execution" type. This means that when your program reaches the breakpoint address, it will stop before executing the instruction located at this address.

Use the following ways to set and clear the code breakpoints:

1) Set the cursor in the instruction line within the *Source* or the *Disassembler* window and click on the *Break* button of the window toolbar. The breakpoint is set/cleared at the address corresponding to the cursor position in the window.

2) Select *Main menu* > *Breakpoints* > *Code Breakpoints* options of the *Breakpoints* menu. This menu has also the *Clear All Breakpoints* command.

The *Source* and *Disassembler* windows display the code breakpoints by highlighted lines with the user–defined background color (red by default). When the program reaches a breakpoint and stops, the line color is changed (to yellow by default).

When the program being debugged is reloaded, all breakpoints are cleared by default, because usually the program is reloaded after recompilation, which may change the particular code addresses. To turn off this default option, use the *Debug Options* dialog.

Complex Event and Triggers

The PICD-430 has a special logical unit known as the *Event Processor* that enables setting complex breakpoints. With complex breakpoints, program execution breaks when a set of programmed conditions is true. These conditions can also be set to trigger some events, such as starting or stopping trace recording, without stopping program execution. Complex events or triggers are enabled for two of the MSP430F44x and MSP430F43x subfamilies only.

Note!
In order to use complex events and triggers, a user should have a full understanding of the MSP430 architecture, as well as the principles of the target MCU and the terminology used in this manual.

If you have set a trigger condition that contradicts the logic of the target MCU, this trigger will never work or will work only as a result of malfunctioning of the target.

To invoke a setting dialog, go to *Main menu > Breakpoints > Complex events* . Two options will be presented: *Simplified scheme* and *Advanced scheme*. The *Simplified scheme* allows you to set complex events that directly stop application program execution. The *Advanced scheme* has many more options. Besides programming the events breaking the program execution it enables program OR, AND and sequential combinations for both stopping program execution and starting or stopping trace recording.

Simplified Scheme

When you select this scheme you get a mnemonic picture showing two programmable logical blocks: *MCU events* and *Enable switches*. Click on the button *Edit* in the *MCU events* block – you will enter the *MCU events setup* dialog

MCU Events

The EEM of the enhanced subfamilies of the MSP430 has two kind of triggers: *Memory Bus* (MB0 .. MB7) and *Register Write* (RW0, RW1) – so altogether there are 10 complex events accessible for programming. The *MCU events setup* dialog enables you to set the events defined by the real status of the target MCU's internal buses.

MBx Settings

Click on the tab MB0…MB7 to program *Memory Bus event* (trigger). A dialog appears.

Bus Value – here you can *either* directly specify the value in binary or hex form *or* pick a symbolic name from the list of variables *or* pick an instruction code.

Bus Value Type – here you can choose whether the Bus Value defined above applies to the address bus (MAB) or the data bus (MDB).

Compare Mode – set a comparison condition of an actual bus value and those set above: equal to, not equal to, greater or equal to, less or equal to.

Bus Cycle Type – here you can specify the cycle type – Read or Write – in which the condition will be checked. Checking the *Don't care* box enables checking in both types of cycles.

RWx Settings

Click on one of two tabs – *RW1* or *WR2* – to program the triggering condition for the register bus. Here you can set a condition to trigger the event on writing into a specified register of the target microcontroller (not reading from!).

Bus Value – here you can directly specify the value in binary or hex form. You can mask some digits or nibbles of the value.

Compare Mode – set a comparison condition of an actual bus value and those set above: equal to, not equal to, greater or equal to, less or equal to.

Register – select the register (R0…R15) which triggers the programming event when writing to that register takes place.

Event Hold – check this box if you would like to hold the event until the next writing to the specified register.

Restrictions of the Register Write Trigger Settings

There are some restrictions for settings in this dialog:

1. Do not check the *Event Hold* option in conjunction with the R0 (Program Counter) and R2 (Status Register) registers because not all the changes in these registers are observed; therefore, they're not completely accessible for the trigger *Register Write Event.*

2. 'Increment the R0 register' is not accessible for the trigger *Register Write* (RWx).

3. R2 (Status Register) can be modified either by program instructions or by setting its flags after execution of some arithmetic operations by the target MCU's ALU. All writes to R2 (Status Register; SR) via instructions are observed. Changes of

ALU status bits due to arithmetic operations are not seen. Thus checking the *Event Hold* feature might give unexpected results. We do not recommend using the R2 register for setting a *Register Write* (RWx) trigger.

4. If you have selected the R3 (ConstantGenerator) register, take note that only the values of R2's functional bits are observed. If the *Event Hold* option is unchecked, then all writes (either via instructions or via arithmetic operations affecting status flags) are observed. However, the reset of SR due to an interrupt is not seen (an interrupt is not a write operation). If the *Event Hold* option is checked, the contents of the functional bits are continuously observed and all changes within these bits are seen. The trigger is active as long as the chosen comparison of 'value' with the functional bits is true. Since the case is quite complicated we also do not recommend usage of the R3 register for setting a *Register Write* (RWx) trigger.

5. Be careful when setting the *Compare Mode* options. If you select the R3 register do not set both “Equal to” and “Not equal to” options. If you select the R2 register be careful setting “Less or equal” and “Greater or equal” options.

Enabling Events

You can individually enable or disable actions of the programmed MCU events. Click on the *Edit* button in the “Enable switches” block and check only those boxes, which correspond to the MCU events that should cause a break in application program execution.

Advanced Scheme

This scheme allows you to program much more complex conditions for stopping the target MCU execution and trace recording.

MCU Events

Setting of the MCU events’options is absolutely the same as was described above in the chapter called *Simplified scheme.*

AND Matrix

In order to program a combination of MCU events, click on the *Edit* button in the block *AND Matrix*. To program an AND check multiple boxes in one vertical line.

Sequencer

This state machine allows programming a chain of events (even events that aren't concurrent) that elaborates an MCU breakpoint or a trigger for trace recording.

The *Sequencer* has 4 state units (State0, State1, State3, State3) and every state unit has a pair of outputs; thus there are 8 transfers altogether. State0 is the initial default state, just after the Sequencer reset. The *Sequencer* goes to reset *either* upon any reprogramming by a user *or* by the MB4 event if the *Reset* box is checked. State3 is the output state. If you check the *Sequencer Enable* box, then the *Sequencer* output will come to the *Enable Switches* matrix instead of output #7 from the *AND Matrix*.

The *Sequencer* jumps from a current stage to a following one (*Destination Stage*) upon triggering an appropriate output from the AND Matrix: #4, #5, #6 or #7. By setting *Destination Stage* numbers at the stage block outputs, you can program very complex breakpoints and triggers.

Enabling Breakpoints and Triggers

You can individually enable or disable actions of the programmed *MCU events*. Click on the *Edit* button in the *Enable switches* block and check only those boxes, which correspond to the MCU events that are relevant for the break you want.

Tracing

The trace buffer is a part of memory inside some MSP430 microcontrollers. It allows real-time recording of the execution of the target program. We will call the process of write/read to/from this memory as *trace recording* or *tracing*. Only the MSP430F44x and MSP430F43x subfamilies have trace buffers that support program tracing.

The MSP430 trace buffer is a looped, 8 frame deep, FIFO memory file that records trace frames that include:

- Address bus states (MAB);
- Data bus states (MDB);
- States of the control signals bus (MCB).

The PICD-430 tracer can record to the trace buffer either continuously or when triggered by the MCU events described above in the chapter *Complex Breakpoints And Triggers.*

Tracer Window

The window can be opened via the *Main menu* > *View* > *Tracer* menu. Each frame displayed in the window is a set of data captured at the moment when the frame is recorded. The data shown are augmented by some information added by the debugger.

The window can display up to 8 frames, which are numbered from 0 to 7. The number of the last frame is always zero (0) and the preceding frames are numbered in a decreasing order: -1, -2,…-7. Each frame contains the following data:

- Frame number (0, -1, -2, …-7);

- States of the control signals bus (MCB) as shown in the table below:

F	Instruction Fetch Cycle	Shows the fetch memory cycle
R	Read/Write Cycle	Shows type of the memory access: '**R**' for read, '**W**' for write
B	Byte/Word Cycle	Shows type of the memory fetch – '**b**' for 8-bit (byte), '**w**' for 16-bit (word)
I	Interrupt Cycle	Shows interrupt cycles
C	CPU off State	Shows that the target MCU is off
P	Power Up Clean State	Shows the target MCU reset after power was on
Z	Zero State	Shows an idle cycle
T	Event Condition State	Shows the Complex Event (trigger)

- *Addr* - an actual status of the address bus captured in real time (2 bytes).

- *Op* – opcode, or an actual status of the data bus captured in real time (2 bytes). *Tracer* capures on this bus code all of the commands executed by the target MCU and data.
- *Instruction* - this field shows mnemonics of the codes read in the *Instruction Fetch* cycles.

- *Source* – a line of the source code that's associated with the first instruction of the object code for that line.

Tracer Window Commands

Frequently used commands that control the PICD-430 tracer can be accessed either via the window's local menu (right mouse button) or by clicking on the buttons that comprise the window's toolbar.

Clicking on the *Clear* button erases the trace buffer and window contents.

Clicking on the *Source* button invokes the *Source* or *Disassembler* window displaying a portion of the code corresponding to the current frame in the *Tracer* window.

Clicking on the *Setup* button invokes the setup dialog.

Tracer Window Setup Dialog

Enable State Storage (Tracer) – By checking/unchecking this box you enable or disable trace recording. Keep in mind that the tracer hardware consumes some energy that is not included in the target MCU specifications.

Tracer Store Mode – This allows you to set basic modes for trace recording and triggering.

The tracer can fill the trace buffer in the following three modes:

1. *On event* – Record only the cycles for triggers set in the "Complex Event Processor and Tracer" dialog. If the trace record was triggered by a programmed *M?mory Bus Event* with the option *Fetch hold*, then the first captured cycle will consist of the instruction that triggered the trace record; the last captured cycle will consist of the instruction fetch following it.

2. *Instruction Fetch only* – Record only the instruction fetch cycles.

3. *All cycles* – Record all the clock (MCLK) cycles.

The tracer can be programmed to one of three different conditions of start/stop recording:

1. *Stop tracing when buffer is full* – stop filling the trace buffer when it overflows.

2. *Start tracing on event* – begin trace buffer filling on a programmed condition (trigger). The first recorded cycle is the cycle following the one in which the condition has occurred. If the *Instruction Fetch only* was checked (see above) then the tracer will record the first instruction following the cycle in which the condition has occurred.

3. *Stop tracing on event* – stop trace recording on the programmed trigger event. The last recorded frame (#0) will store the cycle in which the condition has occurred.

Display frame contents – Here you can filter out some fields in the trace frames if you do not want to see them. By default all the fields are checked and are shown in the Tracer window. You can uncheck unwanted flags or fields to make them invisible and to make the frame format more convenient to watch.

Symbol names in disassembly – checking this box enables the tracer to show symbolic names is the *Disassembler* field, if the names are accessible.

Working With The Tracer In Low-Power Mode

There are some notable aspects of tracing when the target microcontroller is in a Low-Power Mode (LPM). In this case, only the CPU Off bit will be recorded to the frame

when the target microcontroller runs in LPM because the tracer cannot record frames while the clock (MCLK) is off.

The table below shows the frames recorded by the PICD-430 tracer for the case when the program sets Low-Power Mode and then an interrupt occurs (option “All cycles” was set):

Frame	**Addr**	**Op**	**F**	**R**	**I**	**C**	Comment
-7	F028	D032	x	R	-	-	Set_LPM (read Instruction Code)
-6	F02A	0018	-	R	-	-	Set_LPM (read Insruction Data)
-5	F02C	4303	x	R	-	x	Read Instruction Code after "Set_LPM"
CPU and MCLK are off, the tracer does not work							
-4	F02C	4303	x	R	x	x	Branch into ISR (1. Cycle) wake up
-3	F02C	4303	-	R	x	x	Branch into ISR (2. Cycle)
-2	02FE	F02C	-	W	x	x	Branch into ISR (3. Cycle) save PC
-1	F02C	4303	-	R	x	x	Branch into ISR (4. Cycle)
0	02FC	0018	-	W	x	x	Branch into ISR (5. Cycle) save SR

Tracing Modes

Tracers are the tools, which are intended to display the behavior of the target MCU when it’s under control of the debugging program. Sometimes you need to investigate the MCU’s behavior just after some event, sometimes before some event, and sometimes between two events. To realize these functions the PICD-430 tracer can be programmed in three different modes:

- Reverse tracing
- Forward tracing
- Dynamic tracing

Reverse Tracing

This mode should be used in case you want to investigate the behavior of the target MCU *right before some event*, and when it’s under control of the debugger, without breaking the program execution. To program this mode set:

- *Stop tracing on event* – on;
- *Stop tracing when buffer is full* & *Start tracing on event* – off.

Tracing starts synchronously with program execution (emulation) and stops when the trigger (complex event) occurs. If the trigger doesn’t occur, the trace recording process will stop only when you break the program execution manually or at a breakpoint. When program execution stops, the 8 frames stored in the trace buffer will represent the information that preceded the moment when the trigger occurred.

Forward Tracing

This mode should be used in case you want to investigate the behavior of the target MCU *right after some event*, and when it's under control of the debugger, without breaking the program execution. To program this mode set:

- *Stop tracing on event* – off;
- *Stop tracing when buffer is full* & *Start tracing on event* – on.

Tracing starts only when the trigger (complex event) occurs and stops when the trace buffer overflows; i.e., upon recording 8 frames. If the trigger doesn't occur, the trace recording process will never start. When program execution stops, the 8 frames stored in the trace buffer will represent the information immediately following the moment when the trigger occurred.

Dynamic Tracing

If you need to catch an elusive bug, then recording only the frames with very specific contents might be the best approach. Dynamic tracing allows the filtering out of the frames you do not need while leaving in the window only those records that carry some essential information. In this way, you can investigate the behavior of the target MCU program without breaking the program execution *between two events*. To program this mode set:

- *Start tracing on event* – on.
- *Stop tracing on event* – on;

and program two triggers to start and stop trace recording.

Tracing starts recording when one trigger occurs and stops when another occurs. Only enabled triggers control trace recording.

Appendix A

Basic terms and definitions

Target microcontroller (or *target MCU*) - one of the MSP430 devices installed on the user's system (board, prototype), to which the PICD-430's JTAG is connected.

Target is a board or a system (usually a prototype) based on the target microcontroller.

JTAG interface is a well-known abbreviation of the IEEE STD1149.1 standard debug interface widely use in electronic industry. The PICD-430 uses this interface to obtain access to the Enhanced Emulation Module (EEM) embedded in every MSP430 device. The JTAG-interface implemented in the PICD-430 uses the TMS, TDI, TDO and TCK pins of the target MCU. Some MSP430 low-pin-count derivatives have a multiplexed JTAG and PORT debug pin known as the "T? ST" input.

Enhanced Emulation Module (or *EEM*) is a functional unit embedded in every MSP430 device to support on-chip debug functions. Features of the EEM are different for different MSP430 derivatives – some have very limited features, some are enhanced, and include tracing support. EEM includes special internal registers, logic and a limited trace buffer.

Real-time run mode. When in the real-time run mode a target MCU executes an application program and is not under PICD-430 program control, except for the possibility of halting execution by a user command or a breakpoint. In this mode most of the target's resources (CPU registers, SFRs, internal memory - SRAM, PROM, FLASH) are inaccessible.

Halt debug mode. When it's in the halt mode the MCU does not execute a program. The CPU stops and it is under control of the debugger, which communicates with the target MCU via the JTAG port. The Halt mode of the PICD-430 has the following peculiarities:

- All internal resources of the target microcontroller are accessible for the debugger
- All interrupts of the target microcontroller are disabled
- The WatchDog Timer (WDT) of the target microcontroller is frozen
- Peripheral units of the target microcontroller (Timer, ADC, etc.) can either continue running or they can be frozen, depending on the target MCU type (see the General Clock Control option in the Hardware Configuration dialog).

Low-level step. In this mode the target MCU executes only one machine instruction at a time, under control of the debugger.

High-level step. In this mode the target MCU executes only one C language operation at a time, under control of the debugger's hardware and software. The application program executes in real-time mode with breakpoints set at all addresses corresponding to the first operations of the application program.

In order to support high-level stepping, the debugger inserts special debug commands (CALL to_breakpoint) in the application program; this increases code size and slows down its execution. However, the size increase and performance slowdown are generally insignificant. This mode uses one level of stack and can be used only if an application program is prepared for source-level symbolic debug (see "Preparing Program For Source-Level Debugging").

Event trigger or *trigger* is an event that occurs when an output of a special logical comparator implemented in the debugger becomes true. This n-bit comparator compares statuses of the target MCU with a combination set by the user in several PICD-430 dialogs. Each digit can be compared to be equal to '0' or to '1' or it may be masked; masking means that the actual status of the target MCU will not be determined by the programming trigger. The trigger becomes 'true' or occurs when *all* unmasked digits become equal to the corresponding settings.

The most important triggers, known also as *MCU Events*, are *Memory Bus* (MB) and *Register Write* (RW) events. The MB triggers check the statuses of the target MCU's bus Address (MAB), bus Data (MDB) and bus Controls (MCB). The RWs check the status of the target MCU's register bus when writing into a specified register.

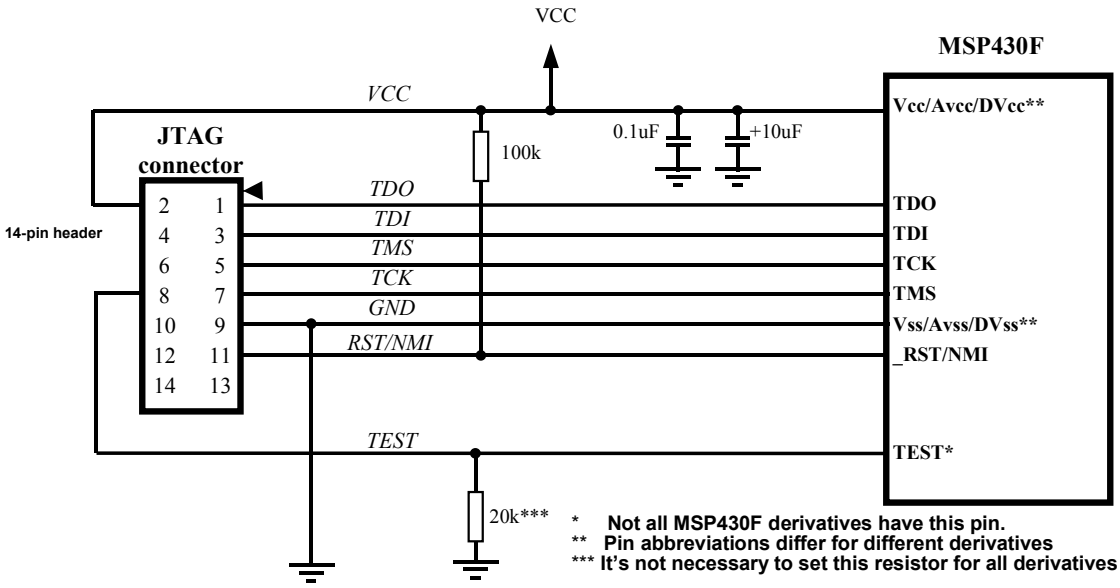
Since the operations of forming and buffering signals in the target microcontroller, as well as the comparison, take at least one clock cycle (MCLK) the trigger always occurs with a small propagation delay.

Trace frame – is a single record in the trace buffer. It is synchronized by a rising edge of the MCLK clock signal. Each frame is 40 bits wide and consists of 8 bits of the control bus (MCB), 16 bits of the address bus (MAB), and 16 bits of the data bus (MDB). The MSP430's trace buffer is 8 frames deep and is implemented only in the MSP430F44x and MSP430F43x derivatives.

Code memory – is the memory from which the target microcontroller fetches instructions of application programs. In the MSP430 the code memory is common for program codes, data and Special Function Registers (SFRs). The program can reside in any type of memory: SRAM, PROM or FLASH.

Appendix B
Connecting Targets To The PICD-430 Unit

To make the PICD-430 debugger workable with the target, the following circuits should be properly connected to the output 14-pin JTAG connector on the debugger unit's front panel: RST/NMI, TMS, TCK, TDI, TDO, GND, VCC, and TEST (if it exists). See the diagram below:



Many standard evaluation boards (header boards, etc.) can be used as target boards to support the PICD-430 debugger and to match its requirements. Target boards from Texas Instruments that are part of the very popular MSP430 FET toolsets can be used with no changes with PICD-430. All these boards have 14-pin sockets that fit the PICD-430 ribbon cables with 14-pin plugs.

When working on MSP430 applications, keep in mind that in some MSP430 devices the pins TMS, TDI, TDO, TCK have dual functions and can be used either as the debug JTAG port or as general-purpose ports. Such chips have a special TEST pin for switching between functions—logical '1' on the TEST pin switches the MSP430 device into the debug mode (JTAG is active).

If in the PICD-430 *Hardware Configuration* dialog you have enabled *Hardware Reset* (box is checked), then the debugger will use the "RST/NMI" for the target MCU reset. The "RST/NMI" pin should be connected to the Vcc line via a 100 kOhm pull-up resistor. You can continue using this pin for purposes other than for the MCU reset, but you should drive this pin only by an FET with an open stock.

You can leave the "RST/NMI" pin unconnected to the PICD-430 but, in case you plan using it as the NMI input, you should uncheck (disable) the *Enable Hardware Reset* box in the *Hardware Configuration* dialog.

Appendix C

Restrictions on the of Use PICD-430

Use of the PICD-430 tool is restricted by some fundamental limitations caused by the target architecture and basic principles of the in-circuit debugger. Here is a list of important restrictions:

1. In the Halt debug mode all interrupts are disabled. The debugger enables them when it goes to the real-time mode.

2. In the Halt debug mode the target MCU's WatchDog Timer (WDT) is frozen. The debugger enables WDT clocking when it goes to the real-time mode. On each transaction from real-time execution to the halt state and back the WDT loses time. Therefore, the WDT becomes inaccurate in the single step mode – it always triggers earlier than it should.

3. Sometimes it is impossible to debug programs residing in the SRAM for the MSP430F12x and MSP430F41x subfamilies. Use FLASH for these devices.

Appendix D

Emulation of MSP430F44xPZ80 and MSP430F43xPZ80 Devices by Means of the MSP430F449PZ100 Microcontroller.

The table below shows how to wire leads of the MSP430F44xPZ80 and MSP430F43xPZ80 devices by means of the MSP430F449PZ100 microcontroller.

F449 PZ100		F4xx PZ80		Connections of the MSP430F449PZ100 leads
Pin#	*Signal*	*Pin#*	*Signal*	*xx - 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
43	S31	43	P3.0/STE0/S31	43-71
39	S27	39	P3.4/S27	39-67
42	S30	42	P3.1/SIMO0/S30	42-70
40	S28	40	P3.3/UCLK0/S28	40-68
41	S29	41	P3.2/SOMI0/S29	41-69
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	P5.3/COM2	46	P5.3/COM2	
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	0	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	

Appendix E

Target boards, approved for use with PICD-430

Here is the list of target boards optionally supplied with the PICD-430 debuggers:

Target MSP430 device	MSP430 subfamily	Recommended Olimex target board	MCU on the target board
	MSP430F11x1D	MSP430-H1121	MSP430F1121A
MSP430F110		TI Active, Not Recomended for New	
MSP430F112		TI Obsolete	
MSP430F1101		TI Obsolete	
MSP430F1111		TI Obsolete	
MSP430F1121		TI Obsolete	
	MSP430F11x1A	MSP430-H1121	MSP430F1121A
MSP430F1101A			
MSP430F1111A			
MSP430F1121A			
	MSP430F11x2	(MSP430-H1121)	MSP430F1121A
MSP430F1122		Partially	
MSP430F1132		Partially	
	MSP430F12x	MSP430-H123	MSP430F123
MSP430F122			
MSP430F123			
	MSP430F12x2	MSP430-H1232	MSP430F1232
MSP430F1222			
MSP430F1232			
	MSP430F133	MSP430-H149	MSP430F149
MSP430F133			
	MSP430F135	MSP430-H149	MSP430F149
MSP430F135			
	MSP430F147	MSP430-H149	MSP430F149
MSP430F147			
	MSP430F148	MSP430-H149	MSP430F149
MSP430F148			
	MSP430F149	MSP430-H149	MSP430F149
MSP430F149			
	MSP430F41x	MSP430-H413	MSP430F413
MSP430F412			
MSP430F413			
	MSP430F43x	MSP430-H449	MSP430F449
MSP430F435			
MSP430F436			
MSP430F437			
	MSP430F44x	MSP430-H449	MSP430F449
MSP430F435			
MSP430F436			
MSP430F437			
MSP430F447			
MSP430F448			
MSP430F449			