MSP430 DLL Developer's Guide

MSP430

ABSTRACT

The MSP430.dll is a Dynamic Link Library (DLL) that provides functions for controlling the Texas Instruments MSP430 Ultra-low Power MCU at development. The MSP430.dll controls the MSP430 MCU via the device's JTAG interface, and provides device control, memory programming and debugging functions. The DLL provides support for the standard 4-wire JTAG implementation as well as the low pin count debug interface called Spy-bi-Wire, or 2-wire JTAG. The MSP430.dll interfaces with the device's JTAG pins using a set of functions encapsulated in another Dynamic Link Library called the Hardware Interface Layer [HIL.dll]. The HIL.dll provides a unified API for MSP430.dll which abstracts the peculiarities of the physical interface to the MSP430. The described software layer hierarchy permits the MSP430.dll to be used with any physical interface (that supports the required control of the JTAG pins) by providing a corresponding HIL.dll. The MSP430.dll greatly simplifies the control of the MSP430, as the user is completely isolated from the complexities of the JTAG. This application note provides an overview of the MSP430.dll, how to use the DLL to control the MSP430 and supplements the information provided in the DLL's C-Header files. Several sample programs that use the MSP430.dll to control the MSP430 are provided.

NOTE: This application note assumes knowledge of the C language, the Dynamic Link Library mechanism, the MSP430, and the JTAG mechanism.

NOTE: Refer to the MSP430 Hardware Tools User's Guide (SLAU278) for information on actual hardware connection to the devices' JTAG pins. For further details on the MSP430 specific JTAG implementation in silicon refer to the MSP430 Memory Programming User's Guide (SLAU320).

Contents

Figures

Revisions

Table 1. Document Revision History

Revision	Date	Author	Notes
0.1	06/2005	W. Lutsch	Initial draft
0.2	09/2005	W. Lutsch	Added Appendix B: Installation of VCP for MSP-FET430UIF
0.3	10/2005	W. Lutsch	Added Spy-bi-Wire information Added Figure 5. Configuring the JTAG protocol Added Abbreviations
0.4	03/2006	W. Lutsch	Added Speed up Flash Programming
0.5	06/2006	W. Lutsch	Added Supporting more than one MSP-FET430UIF
0.6	05/29/2007	W. Lutsch	Added Attach to a running device
0.7	02/10/2009	W. Lutsch	Added eZ430 tool information (both eZ430-F2013 and eZ430-RF2500, Supporting eZ430 emulator dongles) Added information about certified VCP driver (affected: Supporting MSP-FET430UIF, Appendix B Installation of VCP for MSP-FET430UIF) Added Appendix C Switching between certified and non-certified VCP driver Added UseCases
0.8	06/16/2009	W. Lutsch	Added NOTE to abstract which references to SLAU278 & SLAU265
0.9	03/23/2010	F.Berenbrinker	Added notes for new API functions and the automatic protocol scan
1.0	08/09/2011	F.Berenbrinker	Added Code Examples Added MSP430 Flasher as an example for MSP430.dll usage
1.1	08/22/2011	F.Berenbrinker	Remove EEMgui. Example Remove Appendix C

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

Abbreviations

- **MSP-FET430PIF:** Official product designation of Texas Instruments MSP430 Parallel Port JTAG interface (LPT FET).
  http://focus.ti.com/docs/toolsw/folders/print/msp-fet430pif.html
- **MSP-FET430UIF:** Official product designation of Texas Instruments MSP430 USB JTAG interface (USB FET).
  http://focus.ti.com/docs/toolsw/folders/print/msp-fet430uif.html
- **eZ430-F2013:** Official product designation of Texas Instruments MSP430 USB Stick Development Tool (eZ430).
  http://focus.ti.com/docs/toolsw/folders/print/ez430-f2013.html
- **eZ430-RF2500:** Official product designation of Texas Instruments MSP430 Wireless Development Tool (eZ430-RF).
  http://focus.ti.com/docs/toolsw/folders/print/ez430-rf2500.html
- **SBW:** Spy-bi-Wire JTAG debug interface utilized on MSP430 low pin count devices.
- **VCP:** Virtual Com Port, in the context of this document Texas Instruments Virtual Com Port Driver for TUSB3410.\
- **CDC:** Communication Device Class

Developer's Package Folder and File Structure

The MSP430.dll "system" is composed of the following folders and files. Installing the provided MSP430 DLL Distribution package will create the following folders and files in the selected installation destination directory.

- **ApplicationExamples**: This folder contains a set of application examples on how to apply the DLL. Refer to section Application Examples for specific details on each code example.
- **Doc**: This folder contains the complete API documentation of the DLL in HTML and Compressed HTML format as well as this document, the *MSP430 DLL Developer's Guide*.
- **Driver**: This folder contains according driver setup and files for
  - **DriverX**: Teradyne's DriverX (refer to section 5 and Appendix A for details) used to support MSP-FET430PIF JTAG interface
  - **VCP**: TI's VCP driver (refer to Supporting MSP-FET430UIF and Appendix B for details) to support MSP-FET430UIF JTAG interface.
  - **CDC**: TI's CDC driver (refer to Supporting MSP-FET430UIF and Appendix C for details) to support MSP-FET430UIF JTAG interface.
  - **INF**: MS-Windows driver information file for MSP430 Application UART available with eZ430-RF2500 emulator dongles (refer to Supporting eZ430 emulator dongles for details). Refer also to the **eZ430-RF2500 Development User's Guide** (SLAU227) for

further information.
This folder also contains a subfolder called 'PreinstalCDC'. It again contains an example source code which shows how to install the driver INF file on a MS-Windows PC. Refer to the according Appendix sections for more information.

- **Inc**: This folder contains all the C-Header files needed to make use of MSP430.dll. The C-Header files document the DLL functions in detail, including the function prototypes, function parameters, and function return values. They also provide all typedefs, #defines, enumerations, and data structures required to use the DLL.
  - **MSP430.h**: This file is the header file for the MSP430.dll, and provides the function prototypes, typedefs, #defines, enumerations, and data structures for the functions of the DLL. This file is normally located in the same directory as your application's source file, and should be #included by your application's source file. This file is used during compile-time. *MSP430.h is the main header file of the MSP430.dll. (refer to* General application and device handling *for more general information)*
  - **MSP430_Debug.h**: This file is a header file for the MSP430.dll, and provides the function prototypes, typedefs, #defines, enumerations, and data structures for the debugging functions of the DLL. This file is normally located in the same directory as your application's source file, and should be #included by your application's source file. This file is used during compile-time. *MSP430_Debug.h is the main header file of the debugging functions of the MSP430.dll. (refer to* Controlling device program execution *for more general information)*
  - **MSP430_EEM.h**: This file is a header file for the MSP430.dll, and provides the function prototypes, typedefs, #defines, enumerations, and data structures for the **enhanced** debugging functions of the DLL. This file is normally located in the same directory as your application's source file, and should be #included by your application's source file. This file is used during compile-time. *MSP430_EEM.h is the main header file of the **enhanced** debugging functions of the MSP430.dll. (refer to* Enhanced Emulation Module (EEM) Access – EEM API *for more general information)*
  - **MSP430_FET.h**: This file is a header file for the MSP430.dll, and provides the function prototypes, typedefs, #defines, enumerations, and data structures for MSP-FET430UIF maintenance functions of the DLL. This file is normally located in the same directory as your application's source file, and should be #included by your application's source file. This file is used during compile-time. *MSP430_FET.h is the main header file of the MSP-FET430UIF maintenance functions of the MSP430.dll. (refer to* MSP-FET430UIF Firmware Update Support *for more general information)*
  - **HIL.h**: This file is provided as a reference for provided HIL.dll (refer to Hardware Interface Layer (HIL) Library for more details)
- **Lib**: This folder contains according Library files.
  - **MSP430.lib**: This file is the library file for the MSP430.dll, and is required to access the functions of the DLL. This file is normally located in the same directory as your application's source file, and should be added to the Linker Object/Library Modules list of your application. This file is used during link-time.
  - **HIL.lib**: This file is the library file for the provided HIL.dll.

- **MSP430.dll**: This file is the dynamic link library, and contains the device control functions. This file is normally located in the same directory as your application’s executable file, or in your computer system’s default DLL folder. This file is used during run-time.

The MSP430.dll interfaces to the device’s JTAG pins using a set of functions encapsulated in another Dynamic Link Library called the Hardware Interface Layer (HIL.dll). The HIL.dll provides a unified API for MSP430.dll which abstracts the peculiarities of the physical interface to the MSP430. The described software layer hierarchy permits the MSP430.dll to be used with any physical interface (that supports the required control of the JTAG pins) by providing a corresponding HIL.dll. A HIL.dll is supplied that controls the JTAG pins via a Texas Instruments FET Interface Module and the parallel port of a PC. Appendix A describes how to write a HIL.dll for a different physical interface.

- **HIL.dll**: This file is the dynamic link library, and contains the HIL functions. This file is normally located in the same directory as your application’s executable file, or in your computer system’s default folder of DLLs. This file is used during run-time.
- **revisions.txt**: This file provides information about added features of dedicated versions of the DLL.

Using MSP430 DLL

General application and device handling

Use of the MSP430.dll is straightforward. The functions of the DLL are sequenced as follows:

1. The interface is initialized: MSP430_Initialize()
2. The device Vcc is set: MSP430_GetExtVoltage()††, MSP430_VCC()†, MSP430_GetCurVCCT()
3. Configuring the JTAG protocol (Spy-bi-Wire 2-Wire JTAG, 4-wire JTAG) is optional. By default the protocol is selected automatic: MSP430_Configure()
4. The device is identified: MSP430_OpenDevice()
5. Return the identified device: MSP430_GetFoundDevice
6. The mode for verification is optionally configured: MSP430_Configure()
7. The device memory is manipulated using:

   erase [MSP430_Erase()]

   read/write [MSP430_Memory(), MSP430_ReadOutFile(), MSP430_ProgramFile()]

   verify [MSP430_VerifyFile(), MSP430_VerifyMem(), MSP430_EraseCheck()]
8. The device function is manipulated by:

   blowing the security fuse [MSP430_Secure()]††

   reset [MSP430_Reset()]
9. The device interface is closed: MSP430_Close()
10. Errors are handled: MSP430_Error_Number(), MSP430_Error_String()

† Function not supported by MSP-FET430PIF. This function only switches Vcc ON and OFF.

†† Function not supported by MSP-FET430PIF.

Figure 1 shows the flow when starting an MSP430 debug session as the MSP430.dll is used.

Figure 2 shows a code example for a debug session start and error handling using MSP430_Error_Number() and MSP430_Error_String(). The MSP430_DLL.chm help-file in \Doc offers detailed information on all DLL functions and their parameters.

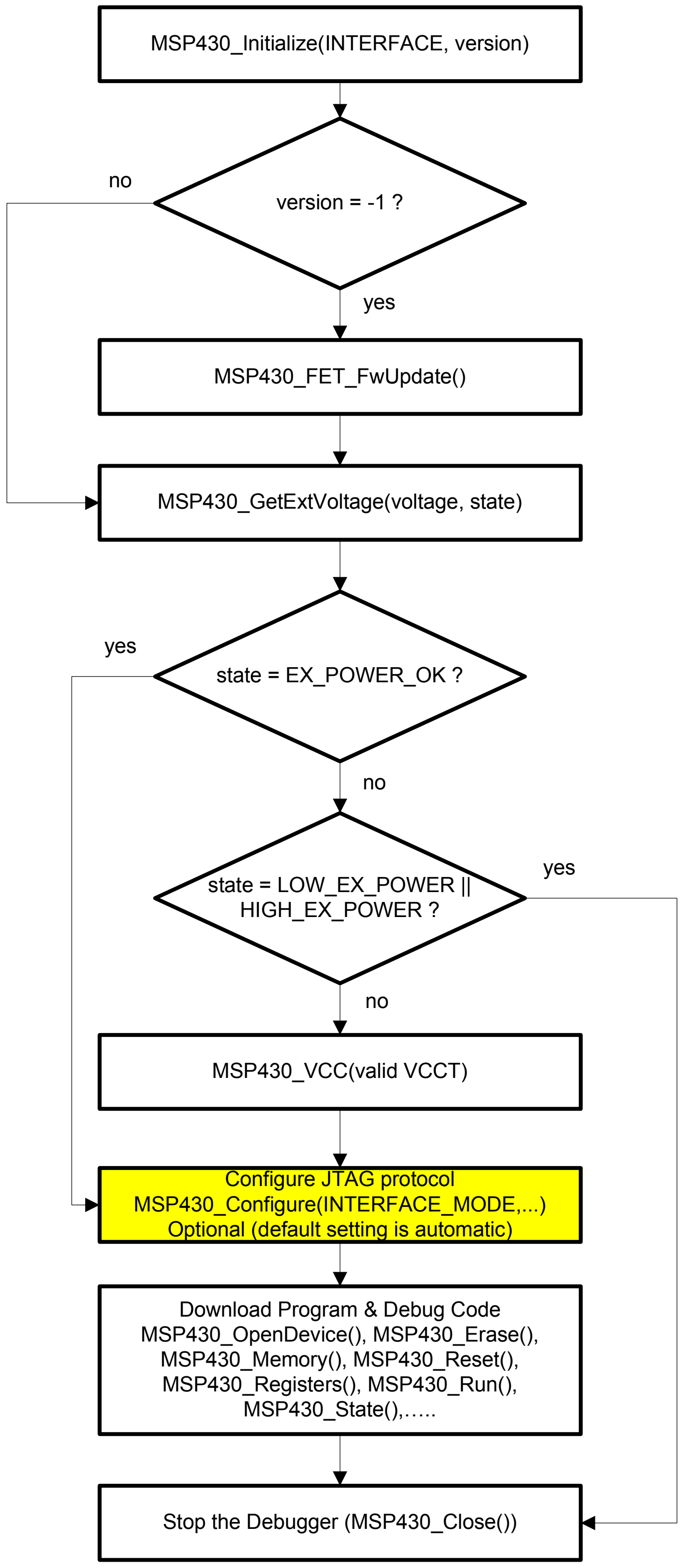

MSP430_Initialize(INTERFACE, version)
version = -1 ?
no
yes
MSP430_FET_FwUpdate()
MSP430_GetExtVoltage(voltage, state)
state = EX_POWER_OK ?
yes
no
state = LOW_EX_POWER || HIGH_EX_POWER ?
yes
no
MSP430_VCC(valid VCCT)
Configure JTAG protocol
MSP430_Configure(INTERFACE_MODE,...)
Optional (default setting is automatic)
Download Program & Debug Code
MSP430_OpenDevice(), MSP430_Erase(),
MSP430_Memory(), MSP430_Reset(),
MSP430_Registers(), MSP430_Run(),
MSP430_State(),.....
Stop the Debugger (MSP430_Close())

Figure 1 Recommended flow to start an MSP430 debug session

```
#include "stdio.h"
#include "MSP430_FET.h"
#include "MSP430_Debug.h"

long lVersion;     // DLL version
long verify = 0;   // verify the filetransfer?

// init JTAG interface. LPT1, LPT2, .., TIUSB is possible.
printf("MSP430_Initialize()\n");
if(MSP430_Initialize("TIUSB", &lVersion) == STATUS_ERROR)
{
   printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number()));           // print error string
   MSP430_Close(1);     // close the debug session
}
// Check firmware compatibility
if(lVersion == -1)                        // firmware outdated?
{
   // perform firmware update
   printf("MSP430_FET_FwUpdate()\n");
   if(MSP430_FET_FwUpdate(NULL, NULL, NULL) == STATUS_ERROR)
   {
            printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number()));  // print error string
            MSP430_Close(1);     // close the debug session
   }
}
// power up the target device
printf("MSP430_VCC()\n");
if(MSP430_VCC(3000) == STATUS_ERROR)         // target VCC in millivolts
{
   printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number()));           // print error string
   MSP430_Close(1);     // close the debug session
}
// configure interface - this is optional! automatic interface selection is the default
printf("MSP430_Configure()\n");
if(MSP430_Configure(INTERFACE_MODE, AUTOMATIC_IF) == STATUS_ERROR)
{
   printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number()));           // print error string
   MSP430_Close(1);     // close the debug session
}
// program .txt file into device memory
printf("MSP430_ProgramFile()\n");
if(MSP430_ProgramFile("C:\file.txt", ERASE_ALL, verify) == STATUS_ERROR)
{
   printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number()));           // print error string
   MSP430_Close(1);     // close the debug session
}
// open the device
printf("MSP430_OpenDevice()\n");
if(MSP430_OpenDevice("DEVICE_UNKNOWN","",0,0,DEVICE_UNKNOWN) == STATUS_ERROR)
{
   printf("Error: %s\n", MSP430_Error_String(MSP430_Error_Number()));           // print error string
   MSP430_Close(1);     // close the debug session
}
/**************************** debug session is started ****************************/
```

Figure 2   Code example for a debug session start

Attach to a running device

The MSP430.DLL offers the possibility to connect to a running MSP430 target device without corrupting the targets' program execution. This can be used to debug an application code which has been running already for a while in the target device. Only the JTAG interface to the target device will be initialized. Especially, no reset of the microcontroller will be performed which might corrupt memory contents of the running application which could contain various information of interest for the debug session.

Establishing the physical JTAG connection to the target device is not always trivial, especially when the RST signal of the target processor is connected to the JTAG header. A successful connection is subject to stable signals on the JTAG connector (a bouncing signal on the RST pin will definitely perform a reset of the microcontroller). Care has been taken in MSP430.DLL to increase the probability of a successful connection. Following the provided flow in Figure 3 offers highest probability of successfully attaching to a running target device. Figure 4 offers a code example for attaching to a running target.

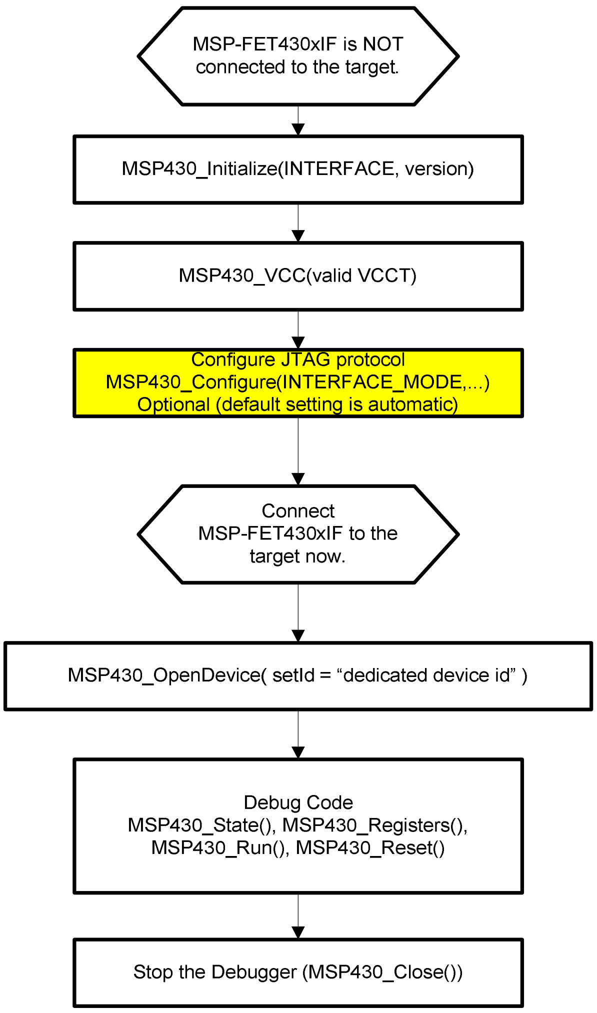

MSP-FET430xIF is NOT connected to the target.
MSP430_Initialize(INTERFACE, version)
MSP430_VCC(valid VCCT)
Configure JTAG protocol MSP430_Configure(INTERFACE_MODE,...) Optional (default setting is automatic)
Connect MSP-FET430xIF to the target now.
MSP430_OpenDevice(setId = "dedicated device id")
Debug Code MSP430_State(), MSP430_Registers(), MSP430_Run(), MSP430_Reset()
Stop the Debugger (MSP430_Close())

Figure 3 Attach to running target

```
long lVersion, state, pCpuCycles;
DEVICE_T TargetDevice;

// get device information - determine device id
printf("MSP430_GetFoundDevice()\n");
if(MSP430_GetFoundDevice((char*)&TargetDevice, sizeof(TargetDevice.buffer)) == STATUS_ERROR)
{
   printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
   MSP430_Close(1);
}

// release the target from JTAG control
printf("MSP430_Run(FREE_RUN, release from JTAG)\n");
if(MSP430_Run(FREE_RUN, TRUE) == STATUS_ERROR)
{
   printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
   MSP430_Close(1);
}

printf("MSP430_Close(VccOff = false)\n");     // close the interface connection
if(MSP430_Close(FALSE) == STATUS_ERROR)      // do NOT turn off Vcc power supply
{
   printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
   MSP430_Close(1);
}

Sleep(100);        // wait a few milliseconds

// initialize the interface again
printf("MSP430_Initialize()\n");
if(MSP430_Initialize("TIUSB", &lVersion) == STATUS_ERROR)
{
   printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
   MSP430_Close(1);
}

// attach to the running target with correct device string and/or device id
printf("MSP430_OpenDevice(DeviceNameString,…,…, TargetDevice.id)\n");
if(MSP430_OpenDevice((char*)TargetDevice.string,"", 0, 0, TargetDevice.id) == STATUS_ERROR)
{
   printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
   MSP430_Close(1);
}

// check CPU state - state should be "RUNNING"
printf("MSP430_State(...,stop = FALSE,...) -> check CPU state\n");
if(MSP430_State(&state, FALSE, &pCpuCycles) == STATUS_ERROR)
{
   printf("%s\n", MSP430_Error_String(MSP430_Error_Number()));
   MSP430_Close(1);
}
```

Figure 4 Code Example for "Attach to running target" – Open a debug session prior to utilizing this code (see. Fig. 2)

Supporting more than one MSP-FET430UIF

Till MSP430.DLL version 2.01.07.000 it was not possible to support more than one MSP-FET430UIF tool on one PC system. The DLL would always try to open the first tool being detected. E.g. opening two instances of an IDE (or two different IDEs that make use of the DLL) and loading the DLL using MSP-FET430UIF would result in a conflict (even if more than one MSP-FET430UIFs have been connected to the PC system) as the DLL always tries to work with the first tool being detected on the USB bus.

Two more API calls have been added to the DLL since version 2.01.08.000 to work around this conflict. The API calls are literally:

- MSP430_GetNumberOfUsbIfs()
- MSP430_GetNameOfUsbIf()

In previous DLL releases MSP430_Initialize() should have been the function to be called at the very first beginning after loading the DLL (see Figure 1). Now the API calls listed above can be used to

- determine how many MSP-FET430UIFs are connected to the PC system and
- get the name (e.g. COM5, COM19,…) and connect-status (ENABLE/DISABLE) of the VCP assigned to a certain MSP-FET430UIF tool

prior to MSP430_Initialize() being called.

Having retrieved the information about how many and which VCPs are available on the PC system a dedicated MSP-FET430UIF tool can be employed by directly passing the VCP name to MSP430_Initialize(), e.g. MSP430_Initialize("COM5",…).

Figure 5 shows the typical flow which is needed to get the needed information about available MSP-FET430UIFs.

Figure 6 offers an example code for initializing multiple MSP-FET430UIFs one by one.

Please also refer to the example project MultipleUifs for a possible application implementation proposal.

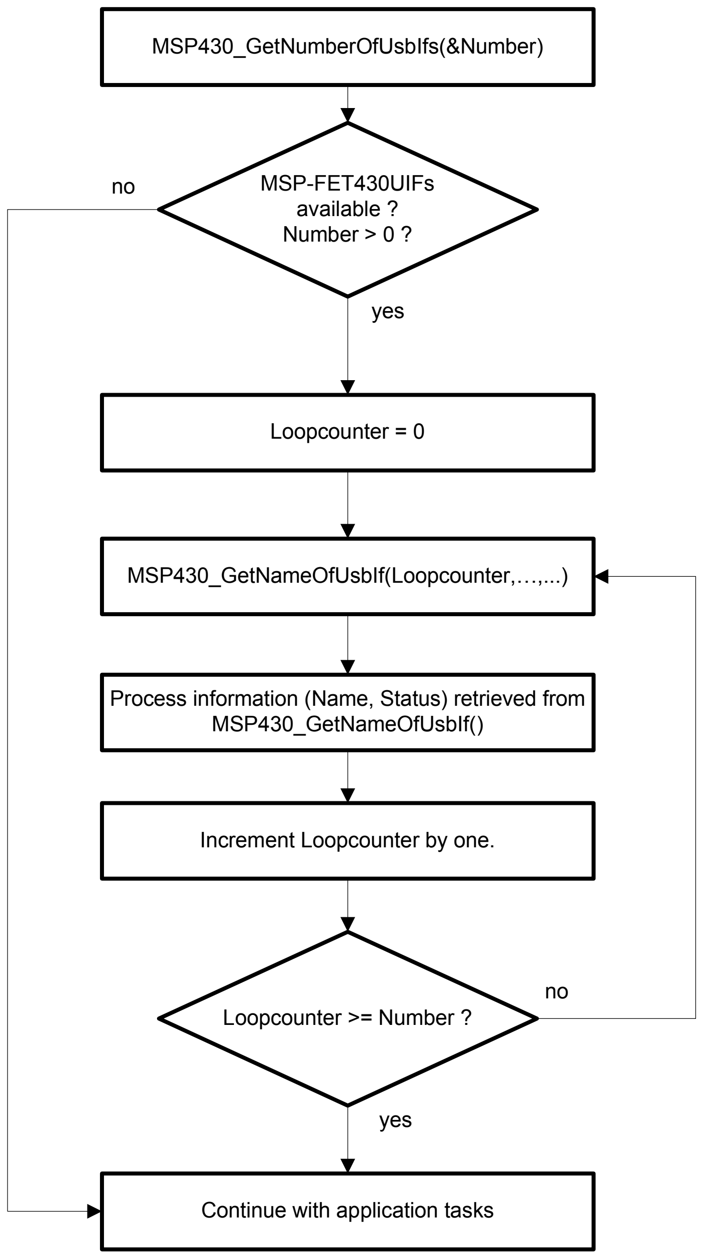

MSP430_GetNumberOfUsbIfs(&Number)
MSP-FET430UIFs available ? Number > 0 ?
no
yes
Loopcounter = 0
MSP430_GetNameOfUsbIf(Loopcounter,…,...)
Process information (Name, Status) retrieved from MSP430_GetNameOfUsbIf()
Increment Loopcounter by one.
Loopcounter >= Number ?
no
yes
Continue with application tasks

Figure 5 Retrieve info about available MSP-FET430UIFs

```
#include "stdio.h"
#include "MSP430.h"

long number, count, status, lVersion, lErrorNumber;
char * name;

// determine the number of connected UIFs
printf("MSP430_GetNumberOfUsbIfs()\n");
if(MSP430_GetNumberOfUsbIfs(&number) == STATUS_ERROR)
{
        printf("Error: Could not determine number of UIFs!\n");
        lErrorNumber = MSP430_Error_Number();
        printf("Reason: %s\n", MSP430_Error_String(lErrorNumber));
}
else
{
        printf("Found %d UIF(s).\n", number);
        for(count = 0; count < number; count++)
        {
        // get the VCP name
                printf("MSP430_GetNameOfUsbIf()\n");
                if(MSP430_GetNameOfUsbIf(count, &name, &status) == STATUS_ERROR)
                {
                        printf("Error: Could not obtain VCP name for UIF %d.\n", count+1);
                        lErrorNumber = MSP430_Error_Number();
                        printf("Reason: %s\n", MSP430_Error_String(lErrorNumber));
                }
                else
                {
                        // initialize the interface
                        printf("Initializing UIF @ %s.\n", name);
                        printf("MSP430_Initialize(UIF %d)\n", count+1);
                        if(MSP430_Initialize(name, &lVersion) == STATUS_ERROR)
                        {
                            lErrorNumber = MSP430_Error_Number();
                            printf("Error: %s\n", MSP430_Error_String(lErrorNumber));
                        }
                        else
                        {
                            printf("Success!\n");

                            // commence with debug session start here...

                            // close the interface
                            printf("MSP430_Close()\n");
                            MSP430_Close(1);
                        }
                }
        }
}
```

Figure 6 Code Example for communication with multiple MSP-FET430UIFs

Configuring the JTAG protocol

By default the DLL is configured to perform an automatic protocol scan to start communication with MSP430 devices. With introduction of the low pin count 2-wire JTAG protocol Spy-bi-Wire (SBW), another configuration mode was implemented in the DLL. The configuration mode is called **INTERFACE_MODE** (refer to MSP430.h file for details). One has to distinguish between four different interface modes dependent on the desired JTAG protocol used for debugging the connected MSP430 derivative.

- **JTAG_IF**: The normal standard 4-wire JTAG communication
- **SPYBIWIRE_IF**: Spy-bi-Wire (2-wire) JTAG protocol (NOTE: only supported by MSP-FET430UIF)
- **SPYBIWIREJTAG_IF**: Standard 4-wire JTAG communication for MSP430 devices which also support Spy-bi-Wire (a special entry sequence is needed to switch these MSP430 derivatives into 4-wire mode which cannot be applied to any MSP430 devices)
- **AUTOMATIC_IF**: JTAG communication protocol is selected automatically by the DLL (default)

If MSP430_Configure() is called to configure the JTAG protocol, it must be done before MSP430_OpenDevice() is called as it is shown in Figure 1.

Speed up Flash Programming

The API routines MSP430_Erase() and MSP430_Memory() enable manipulation of the target devices Flash Memory. By applying the DLL with an LPT port interface these routines make use of the target devices RAM. All content of the target devices RAM is preserved before the Flash manipulation and restored afterwards. This mechanism is applied that way to allow Flash Memory manipulation during an active debug session without corrupting any RAM content. Anyway, it takes perceivable time to preserve/restore RAM contents. Thus this mechanism might be considered to be not very useful under some circumstances, e.g. during an initial Flash Programming at the beginning of a debug session.

Therefore the RAM preserve/restore mechanism can be disabled by an additional MSP430_Configure () function call. The configuration mode is called **RAM_PRESERVE_MODE**.

The following sequence might be used, e.g. for an initial Flash Programming sequence:

(1) MSP430_Configure(RAM_PRESERVE_MODE, DISABLE);

(2) MSP430_Erase(ERASE_ALL,..,..);

(3) MSP430_Memory(..., ..., ..., WRITE);

(4) MSP430_Memory(..., ..., ..., READ);

..... Flash Programming/Download finished

(n) MSP430_Configure(RAM_PRESERVE_MODE, ENABLE);

Controlling device program execution

The MSP430.dll provides additional debugging functions to developers of third party tools for the MSP430. The debugging functions include execution control (free run, run to breakpoints, single step, state, stop, set breakpoint), device control (read/write registers, reset, clock configuration, device configuration), and low-level access to the advanced features of the Enhanced Emulation Module (EEM) that provides such features as complex breakpoints, trace buffers, etc.. The low-level access to EEM registers (namely Read/Write EEM register) is basically kept in the DLL due to compatibility reasons. TI encourages developers to apply the Enhanced Emulation Module Access (EEM API) mentioned in the next section. TI reserves the right to remove low-level EEM access in future releases of the DLL with according notification to affected developers.

Enhanced Emulation Module (EEM) Access – EEM API

Additionally, the MSP430.dll provides an enhanced debug API that allows access to MSP430's Enhanced Emulation Module on a higher application level than accessing EEM registers directly. Refer to source code of application examples, on how to apply the EEM API. *Notice that some API functions are no longer allowed to be called in case EEM API is used. These functions are namely:*

- MSP430_Configure() with parameter 'mode' set to CLK_CNTRL_MODE
- MSP430_Configure() with parameter 'mode' set to MCLK_CNTRL_MODE
- MSP430_State() with parameter 'stop' set to FALSE
- MSP430_EEM_Open()
- MSP430_EEM_Read_Register()
- MSP430_EEM_Read_Register_Test()
- MSP430_EEM_Write_Register()
- MSP430_EEM_Close()

Refer to the detailed documentation in MSP430_EEM.h.

Error handling

Most functions of the MSP430.dll return an indication of success (STATUS_OK) or failure (STATUS_ERROR). If STATUS_ERROR is returned, MSP430_Error_Number() can be used to obtain a detailed error code. MSP430.h contains an enumeration of all error codes, and lists the error codes returned by each DLL function. MSP430_Error_String() will return the string corresponding to the error code parameter.

STATUS_ERROR is returned at the first error condition. The DLL typically does not attempt to retry and/or recover from the error condition. It is the responsibility of the application to retry the failed operation, and to possibly implement some sort of "back-out" recovery mechanism.

Miscellaneous

The MSP430.dll is a *partially intrusive* tool; accessing the device via JTAG can affect the device (i.e. clocking the Watchdog mechanism). However, steps are taken within the DLL to minimize the effects upon the device caused by JTAG. Refer to the Notes: section of the function comments for function specific information (including known effects).

Unless noted in the "Notes:" section of the function comments, the functions of the MSP430.dll do not assume or make use of any device resources (i.e. a reference crystal, registers, regions of memory, etc.).

Supporting MSP-FET430PIF

Tetradyne's DriverX Parallel Port (LPT) Hardware Driver

Tetradyne's DriverX is used to control the PC's parallel port. HIL.dll, described in the next section makes use of this kernel driver. Refer to Appendix A on how to install DriverX on a PC system. Normally the first call to MSP430_Initialize('LPTx',…) installs DriverX on the system. Rebooting the system is required afterwards.

Hardware Interface Layer (HIL) Library

The MSP430.dll interfaces to the device's JTAG pins using a set of functions encapsulated in another Dynamic Link Library called the Hardware Interface Layer [HIL.dll]. The HIL.dll provides a unified API for MSP430.dll which abstracts the peculiarities of the physical interface to the MSP430. The described software layer hierarchy permits the MSP430.dll to be used with any physical interface (that supports the required control of the JTAG pins) by providing a corresponding HIL.dll. A HIL.dll is supplied that controls the JTAG pins via a Texas Instruments FET Interface Module and the parallel port of a PC. However, the user may supply a functionally equivalent HIL.dll in order to interface the MSP430.dll to a different target environment. The HIL.dll also interfaces the MSP430.dll to timer functions of the host operating system.

The following files are supplied in the HIL project folder:

- **HIL.h**: This file is the header file for the HIL.dll, and provides the function prototypes, typedefs, #defines, and enumerations for the dll. This file is normally located in the same directory as the HIL source file, and should be #included by the HIL source file. This file is used during compile-time. *As a check of the compliance of the HIL.dll functions to those required by MSP430.dll, do not modify this file.*

- **HIL.def**: This file specifies the name by and order in which the functions of the HIL.dll must be exported. This file is normally located in the same directory as the HIL source file, and should be added to the HIL project to affect the linker.

- **HIL.c**: This file is the source for the HIL.dll. As supplied, HIL.c provides an interface to the device JTAG pins via the TI FET Interface Module and the PC parallel port. The PC parallel port is itself controlled via a third party driver (Tetradyne's DriverX). HIL.c uses the "queued command" mechanism provided by DriverX for reasons of improved performance. Note: Texas Instruments does not provide the DriverX.h file required by HIL.c, or the DriverX library.

Use HIL.c as a template when implementing a new HIL.dll. Each function specified in HIL.h must be implemented in the new HIL.dll, and must comply with the function prototype.

The functions of the HIL.dll can be roughly partitioned as follows:

- General control; these functions initialize, open, control, and close the interface: HIL_Initialize(), HIL_Open(), HIL_Connect(), HIL_Release(), HIL_Close(), HIL_TEST_VPP()

- JTAG instruction and data transfer; these functions transfer JTAG instructions to the device, and data to and from the device:
  HIL_JTAG_IR(), HIL_JTAG_DR()
- Device signal control; these functions control the JTAG and reset signals, and the device voltages:
  HIL_TST(), HIL_TCK(), HIL_TMS(), HIL_TDI(), HIL_TCLK(), HIL_RST(), HIL_VCC(), HIL_VPP()
- Time delay and timing; these functions provide time delay and time delay measurement:
  HIL_DelayMSec(), HIL_StartTimer(), HIL_ReadTimer(), HIL_StopTimer()

For a detailed discussion of the JTAG signals and signaling requirements (as required by HIL_JTAG_IR() and HIL_JTAG_DR()), refer to TI Application Report SLAA149 *Programming a Flash-Based MSP430 Using the JTAG Interface*.

The supplied HIL.dll that interfaces to the JTAG pins via the TI FET Interface Module and the PC parallel port has the following limitations:

- It is not possible to set the device Vcc to a specific value using HIL_VCC(); Vcc is set to the maximum voltage supported by the interface when a non-zero voltage is specified, and Vcc is set to zero when a zero voltage is specified.
- It is not possible to set the device Vpp to a specific value using HIL_VPP(); Vpp is set to the maximum voltage supported by the interface when a non-zero voltage is specified, and Vpp is set to zero when a zero voltage is specified.

As a result of the above two limitations, it is not possible to use MSP430.dll functions MSP430_VCC() to set the device Vcc and MSP430_Secure() to secure the device (i.e., “blow” the device security fuse).

Supporting MSP-FET430UIF

Virtual Com Port (VCP) Driver for MSP-FET430UIF

MSP-FET430UIF uses a TUSB3410 device from TI for USB communication. The VCP driver is provided in two different versions. The more recent version is digitally signed and certified by Microsoft. The certification is available for four different MS-Windows platforms: XP32, XP64, Vista32 and Vista64. Since version 2.04.00.000 of the MSP430.dll both VCP drivers are supported. Previous MSP430.dll versions are only able to recognize and work with MSP-FET430UIF tools which are installed with the older, not-certified version of the VCP driver. Refer to Appendix B for details and installation information.

Enhancements over MSP-FET430PIF

- Supports blowing of the JTAG fuse by calling *MSP430_Secure()*.
- Supports setting of the target voltage between 1.7V to 3.6V using *MSP430_VCC()*.
- Support for reading back currently set target Vcc using *MSP430_GetCurVCCT()*.
- Detection and measurement of external target power supply by *MSP430_GetExtVoltage()*.

MSP-FET430UIF Firmware Update Support

With every new version of MSP430.dll the firmware of the MSP-FET430UIF JTAG interface needs to be updated accordingly. MSP430.dll includes a binary image of the corresponding MSP-FET430UIF firmware. Calling MSP430_FET_FwUpdate() as described in the flow chart in Figure 1 assures consistency between versions of MSP-FET430UIF firmware and the DLL applied on the PC.
With this release the major DLL version has changed and the firmware now consists of a communication core and a JTAG stack which can be updated independently. Therefore it was necessary to extend the firmware update mechanism. As you can see in figure 7, MSP430_Initialize() returns either -3, -1 or the actual DLL firmware Version.

In case MSP430_Initialize() returns -3, a major firmware version update is required. Afterwards call MSP430_FET_FwUpdate() to update the firmware through a DLL internal binary image, a given update file will be ignored.
If MSP430_Initialize() returns -1, the UIF firmware is already updated to the major version, but either the communication core or JTAG stack does not match the DLL version. If required, the core, in any case, is updated through the DLL internal image. The JTAG stack can be updated through a ti-txt or intel-hex file by passing a file to MSP430_FET_FwUpdate(), otherwise an DLL internal update will be performed.

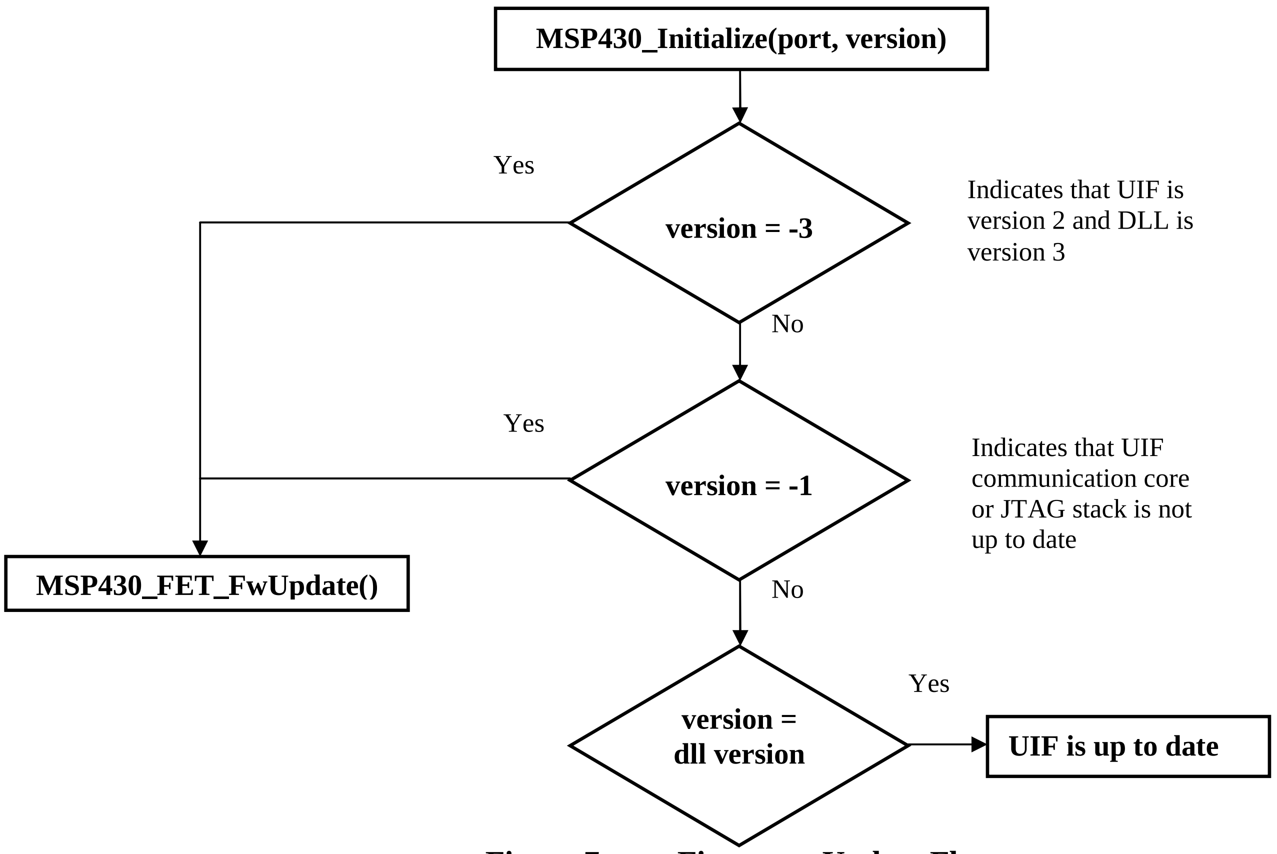
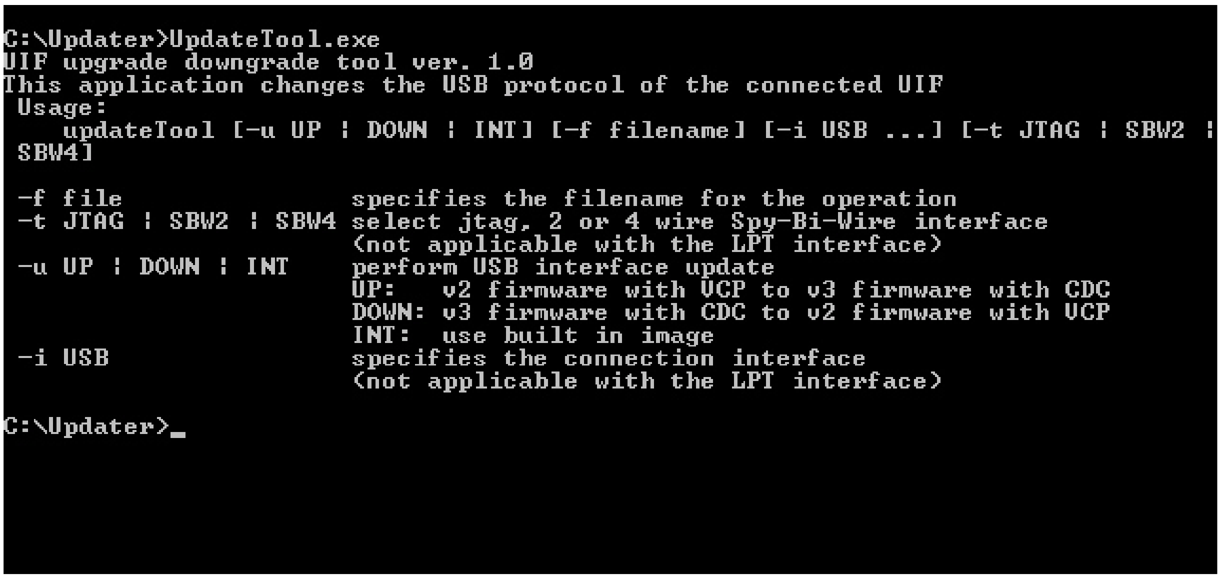

MSP430_Initialize(port, version)
Yes
version = -3
Indicates that UIF is version 2 and DLL is version 3
No
Yes
version = -1
Indicates that UIF communication core or JTAG stack is not up to date
MSP430_FET_FwUpdate()
No
Yes
version = dll version
UIF is up to date

Figure 7 Firmware Update Flow

Firmware update with Update-Tool

To update the UIF firmware without an IDE use the command line based Update-Tool.
The Update-Tool also provides the possibility of firmware up/downgrade between major firmware versions.
Please refer to Appendix D to determine if you are using an UIF with revision 1.3 because this requires an additional update step.



```
C:\Updater>UpdateTool.exe
UIF upgrade downgrade tool ver. 1.0
This application changes the USB protocol of the connected UIF
 Usage:
    updateTool [-u UP | DOWN | INT] [-f filename] [-i USB ...] [-t JTAG | SBW2 |
 SBW4]

 -f file                specifies the filename for the operation
 -t JTAG | SBW2 | SBW4  select jtag, 2 or 4 wire Spy-Bi-Wire interface
                        (not applicable with the LPT interface)
 -u UP | DOWN | INT     perform USB interface update
                        UP:   v2 firmware with UCP to v3 firmware with CDC
                        DOWN: v3 firmware with CDC to v2 firmware with UCP
                        INT:  use built in image
 -i USB                 specifies the connection interface
                        (not applicable with the LPT interface)

C:\Updater>_
```

Figure 8 Update-Tool

Usage:

updateTool –u **UP**: updates the UIF's major firmware version (e.g. version 2 to 3)

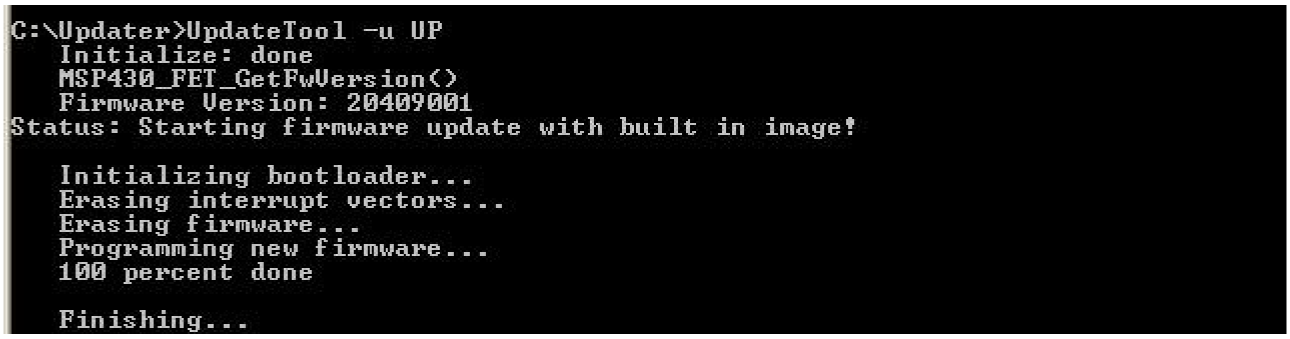
updateTool –u **DOWN**: downgrades UIF's major firmware version through the binary image stored in Uifv3Downgrader.txt

updateTool –u **INT**: updates the UIF with the DLL internal firmware image

Important: Make sure that the CDC driver is already installed before performing a major firmware version update. Also a file called "CDC.log" with the content "True" must be in the same folder as the DLL, which indicates that the CDC driver was successfully installed. Otherwise the update process returns an update error.

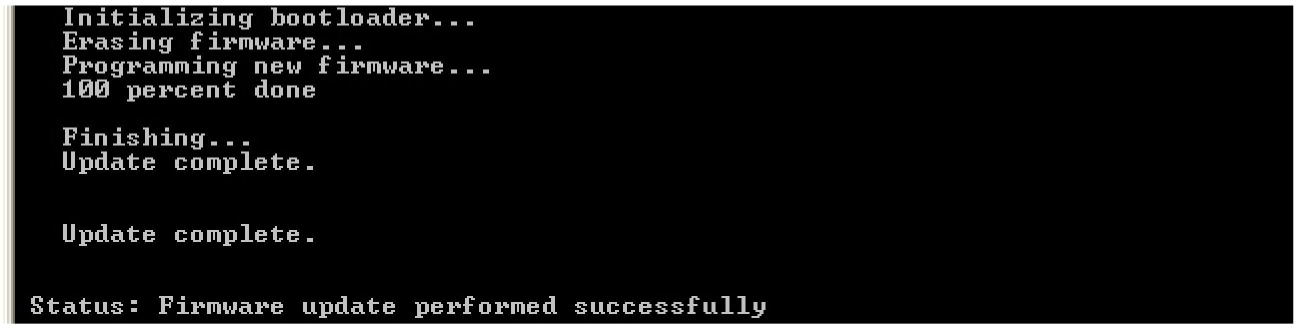
Additional update step for MSP-FET430UIF with hardware revision 1.3

After calling updateTool –u UP the update process starts and you can see the following command line window



```
C:\Updater>UpdateTool -u UP
   Initialize: done
   MSP430_FET_GetFwVersion()
   Firmware Version: 20409001
Status: Starting firmware update with built in image!

   Initializing bootloader...
   Erasing interrupt vectors...
   Erasing firmware...
   Programming new firmware...
   100 percent done

   Finishing...
```

On finishing, the TUSB3410 should be reseted and the UIF show up as a CDC device.
Due to the reason mentioned in Appendix D that is not possible, so it is necessary to disconnect the UIF and connect it again.
After doing so, the update process continues.



```
   Initializing bootloader...
   Erasing firmware...
   Programming new firmware...
   100 percent done

   Finishing...
   Update complete.


   Update complete.


 Status: Firmware update performed successfully
```

Supporting eZ430 emulator dongles

There exist several different versions of the eZ430 emulator.

- **eZ430-F2013**: The dongle appears almost identical as an MSP-FET430UIF. It requires the VCP driver which enumerates a virtual COM port on the host PC. By calling MSP430_GetHwVersion() one can distinguish between the two different tools. From a USB driver perspective they are identical.
- **eZ430-RF2500**: The dongle enumerates as a Human Interface Device (HID) (e.g. like a mouse). The HID class driver is part of the Windows operation system, thus the enumeration does not require any user interaction. The HID interface is used for the JTAG communication to the target device. Beside the HID channel the dongle also tries to enumerate a Virtual Com Port (which is called MSP430 Application UART). Other than the VCP of MSP-FET430UIF (and eZ430-F2013) this port is based on Communication Device Class (CDC) driver. This CDC driver class is also part of the Windows operating system but it requires an INF file for installation. The provided INF file (430CDC.inf, to be found in folder Driver/Inf) is certified for MS-Windows operating systems XP32, XP64, Vista32 and Vista64. The folder Driver/Inf contains a subfolder PreinstalCDC. This subfolder contains an example source code that shows how to install the INF file on a MS-Windows PC. It is recommended to install the INF file like described in the example. If not done like that the Windows Hardware Wizard will pop up as soon as the user connects the tools to the PC. Afterwards the user has to manually point the Wizard to the correct location of the INF file.

Other supported eZ430 tools that make use of the HID interface include the eZ430 Chronos or the Launchpad and the MSP-EXP430FR5739 FRAM Experimenter’s board, where the eZ emulator is onboard.

Application Examples

The DLL Developer's Package features a series of example projects to illustrate the usage of the DLL functions. After the build, the executables can be found in ApplicationExample/Executables. Refer to the source code for details on how to call DLL functions and correctly pass parameters to those functions.

Example

Example is an example project that demonstrates how the basic functions of the DLL are called to initialize the interface, identify the device, configure the device, manipulate the device memory (erase, program, verify, read), blow the device security fuse, reset the device, close the interface, and handle error conditions. Refer to the source file *Example.c*.

ExampleDebug

ExampleDebug is an example project that demonstrates how the functions of the DLL are called to initialize the interface, identify the device, configure the device, manipulate the device memory (erase, program, verify, read), read the device registers, set device breakpoints, run the device (free, with breakpoints, single step), reset the device, close the interface, and handle error conditions. Refer to the source file *Example Debug.c*.

UifUpdate

UifUpdate is an example project that demonstrates how to perform an MSP-FET430UIF firmware update by calling MSP430_FET_FwUpdate() including handling of the notify callback mechanism during the update process. Refer to the DLL API documentation of MSP430_FET_FwUpdate() for technical details.

MultipleUifs

MultipleUifs is an example project that demonstrates how to support multiple MSP-FET430UIF tools connected to one PC system. The example project comes along with a GUI that shows a possible support implementation.

UseCases

UseCases contains a set of subfolders, each containing a small example code project demonstrating a certain use case for the MSP430.dll. A use case usually builds to a single executable file which can be executed on the command line. It takes a set of standard, plus a set of use case specific (optional), command line arguments to perform certain actions. In the root directory of the use case MSP430 family and/or device specific batch files can be found demonstrating possible command line invocations. The source code for the use case is located in the Src/Visual folder. Most of the use cases require an MSP430 target code which gets loaded through the MSP430.dll into the target devices' memory. The source code for MSP430 target code is located in Src/Msp430.

MSP430 Flasher

The MSP430 Flasher gives an example of how to use the MSP430.dll in a simple console-based command line tool. It implements nearly all of the DLLs functions and can be used by calling the MSP430Flasher.exe with a series of parameters from a batch file. For a list of the available parameters and their descriptions refer to the MSP430 Flasher manual or the MSP430 Flasher Wiki page.

For more detailed information on how to structure a DLL-based communication with an MSP430 device, see the MSP430 Flasher source code in ApplicationExample/MSP430Flasher/Source.

Appendix A. Installation of DriverX

The supplied HIL.dll uses the DriverX product from Teradyne to interface to the parallel port of the PC. In order to demonstrate the example project, DriverX must be installed in the computer system as follows:

NOTE: DriverX is currently not supported by Windows 7 – 64 bit.

A.1 For Win95, 98, and ME machines:

1. Copy the DriverX Windows kernel-mode driver, Driverx.vxd, to the \System subdirectory of the root Windows directory.
2. Create the following registry keys:

```
[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\DRIVERX]
 "ErrorControl"=dword:00000001
 "Start"=dword:00000002
 "Type"=dword:00000001

[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\DRIVERX\FET]
 "IgnoreConflicts"=dword:00000001
```

A.2 For Win NT machines:

1. Copy the DriverX NT kernel-mode driver, Driverx.sys, to the \System32\Drivers subdirectory of the root NT directory.
2. Create the following registry keys:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\DriverX]
 "ErrorControl"=dword:00000001
 "Start"=dword:00000002
 "Type"=dword:00000001

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\DriverX\Parameters]

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\DriverX\Parameters\FET]
 "IgnoreConflicts"=dword:00000001

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\DriverX\Enum]
 "0"="Root\\LEGACY_DRIVERX\\0000"
 "Count"=dword:00000001
 "NextInstance"=dword:00000001

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\System\DriverX]
 "TypesSupported"=dword:00000007
 "EventMessageFile"="%SystemRoot%\System32\IoLogMsg.dll;%SystemRoot%\System32\Dri
vers\DriverX.sys"
```

A.3 New Driver X Installation Instructions:

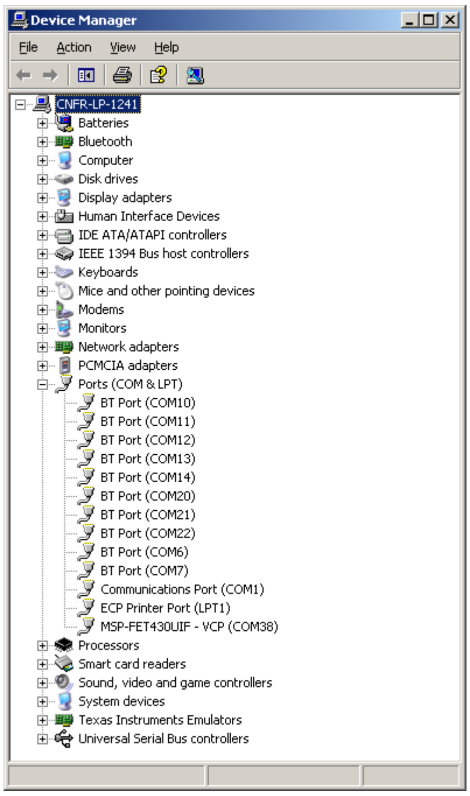
```
1. Replace:
    - Driverx.sys
    - Driverx.vxd
   with the new versions

2.  Add Drvx40.dll to the 'Setup Files / Language Independent / windows..' folder
    within Install Shield
    Replace
     - HIL.dll
     - msp430.dll
    with the new versions

3.  Add InstallDriverX function to install script.

4.  Call InstallDriverX after the files have been copied.
```

```
/*---------------------------------------------------------------------------*\
 *
 * Function:    InstallDriverX
 *
 *  Purpose:    Install DriverX
 *
 *    Input:
 *
 *  Returns:
 *
 * Comments:
\*---------------------------------------------------------------------------*/
   #define DLL_FILE  SUPPORTDIR^"Drvx40.dll"
   prototype  Drvx40.HwConfigureDriver();
   prototype  Drvx40.HwConfigureParPort(LONG, LONG, BYREF LONG);
                       //(DWORD nPort, DWORD fFlags, PDWORD pfOutFlags);

function  InstallDriverX()
   NUMBER  nResult;
   BOOL    bDone, bDone1, bDone2, bDone3;
   LONG    OutFlags;
begin

   // Load the dll into memory.
   nResult = UseDLL (DLL_FILE);

   // Call Install DriverX.
   bDone = HwConfigureDriver();

   bDone1 = HwConfigureParPort(0, 1, OutFlags);
   bDone2 = HwConfigureParPort(1, 1, OutFlags);
   bDone3 = HwConfigureParPort(2, 1, OutFlags);

   // Remove the dll from memory.
   UnUseDLL (DLL_FILE);
   return(bDone && (bDone1 || bDone2 || bDone3));

end;// of InstallDriverX()

/*--------------------------------------------------------------------------*\
 *
 * Function:    SetupRegistry2
 *
 *  Purpose:    Generates Registry Keys for Win
 *
 *    Input:
 *
 *  Returns:
 *
 * Comments:    Functional only when additional files on disk1:
 *              - _INST32.EX_
 *              - _SETUP32.LIB (compressed with ICOMP) including
 *                1) UNINST.EXE and
 *                2) _ISRES.DLL renamed from _ISRES32.DLL
\*--------------------------------------------------------------------------*/

function  SetupRegistry2()
  STRING  szKey,szKey1,szClass,szKeyRoot,szNumName,szNumValue;

begin
        RegDBSetDefaultRoot( HKEY_LOCAL_MACHINE );

    //Key DriverX

        if bWinNT then
          szKey       = "SYSTEM\\CurrentControlSet\\Services\\DriverX";   // NT
          szKey1      = "SYSTEM\\CurrentControlSet\\Services\\DriverX\\Parameters";   // NT
        else
          szKey1      = "System\\CurrentControlSet\\Services\\VxD\\DRIVERX";  // 95 + 98
          szKey = szKey1;
        endif;


        szClass     = "";

        if ( RegDBCreateKeyEx( szKey, szClass ) < 0 ) then
            MessageBox("Cannot create Key DriverX", WARNING);
        endif;

        szNumName   = "ErrorControl";
        szNumValue  = "1";
        if ( RegDBSetKeyValueEx( szKey, szNumName, REGDB_NUMBER, szNumValue, -1 ) < 0 ) then
            MessageBox("Cannot set value ErrorControl", WARNING);
        endif;

        szNumName   = "Start";
        szNumValue  = "2";
        if ( RegDBSetKeyValueEx( szKey, szNumName, REGDB_NUMBER, szNumValue, -1 ) < 0 ) then
            MessageBox("Cannot set value Start", WARNING);
        endif;

        szNumName   = "Type";
        szNumValue  = "1";
        if ( RegDBSetKeyValueEx( szKey, szNumName, REGDB_NUMBER, szNumValue, -1 ) < 0 ) then
            MessageBox("Cannot set value Type", WARNING);
        endif;


    //Key FET in DriverX

        szKey       = szKey1 + "\\FET";
        szClass     = "";

        if ( RegDBCreateKeyEx( szKey, szClass ) < 0 ) then
            MessageBox("Cannot create Key FET", WARNING);
        endif;

        szNumName   = "IgnoreConflicts";
        szNumValue  = "1";
        if ( RegDBSetKeyValueEx( szKey, szNumName, REGDB_NUMBER, szNumValue, -1 ) < 0 ) then
            MessageBox("Cannot set value IgnoreConflicts in LPT01", WARNING);
        endif;


    //Key DriverX in EventLog  (NT only)
        if bWinNT then

         szKey       = "SYSTEM\\CurrentControlSet\\Services\\EventLog\\System\\DriverX";
         szClass     = "";

         if ( RegDBCreateKeyEx( szKey, szClass ) < 0 ) then
            MessageBox("Cannot create Key DriverX in EventLog", WARNING);
         endif;

         szNumName   = "TypesSupported";
         szNumValue  = "7";
         if ( RegDBSetKeyValueEx( szKey, szNumName, REGDB_NUMBER, szNumValue, -1 ) < 0 )
then
            MessageBox("Cannot set value TypesSupported", WARNING);
         endif;

         szNumName   = "EventMessageFile";
         szNumValue  =
"%SystemRoot%\\System32\\IoLogMsg.dll;%SystemRoot%\\System32\\Drivers\\DriverX.sys";
         if ( RegDBSetKeyValueEx( szKey, szNumName, REGDB_STRING_EXPAND, szNumValue, -1 ) <
0 ) then
            MessageBox("Cannot set value EventMessageFile", WARNING);
         endif;

         //LaunchAppAndWait ( "net" , "start driverx" , WAIT );

        endif;

end;// of SetupRegistry2()
```

Appendix B. Installation of VCP for MSP-FET430UIF

This document contains the installation and un-installation procedure of the MSP-FET430UIF Driver on Windows XP (32Bit & 64 Bit), Windows Vista (32Bit & 64 Bit) and Windows 7 (32Bit & 64 Bit). The MSP-FET430UIF Driver is based on the TUSB3410/TUSB5052 Virtual COM port driver provided by Texas Instruments under TI's non-cost license agreement and is adapted for use with MSP-FET430UIF according to *USB/Serial Applications Using TUSB3410/5052 and the VCP Software* (SLLA170B). The dialog boxes shown below in this document indicate what will be seen when installing a MSP-FET430UIF JTAG Debug Interface which comes along with VID (Vendor Id): 0x0451 and PID (Product Id): 0xF430.

Important Note: Since MSP430.dll version 2.04.03.000 only one version of the VCP driver is provided with the Developer package. The installation of this new certified driver is much easier compared to the old non-certified version (not included in this packet). The installation requires running a driver installer, which is part of this packet on the PC. The operation system and all necessary settings are detected and set automatically by the driver installer. The installer contains driver for the MSP-FET430UIF, the eZ430-RF2500 dongle and the DriverX for the TI MSP430 PIF Tool.



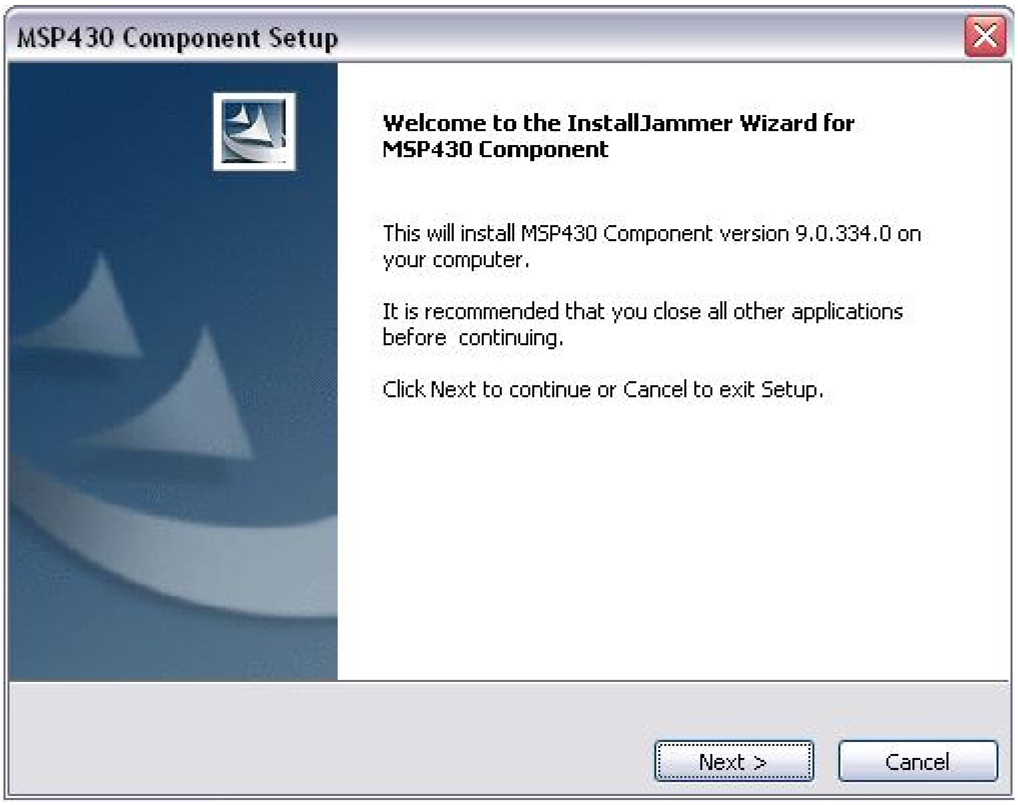
CNFR-LP-1241
- Batteries
- Bluetooth
- Computer
- Disk drives
- Display adapters
- Human Interface Devices
- IDE ATA/ATAPI controllers
- IEEE 1394 Bus host controllers
- Keyboards
- Mice and other pointing devices
- Modems
- Monitors
- Network adapters
- PCMCIA adapters
- Ports (COM & LPT)
  - BT Port (COM10)
  - BT Port (COM11)
  - BT Port (COM12)
  - BT Port (COM13)
  - BT Port (COM14)
  - BT Port (COM20)
  - BT Port (COM21)
  - BT Port (COM22)
  - BT Port (COM6)
  - BT Port (COM7)
  - Communications Port (COM1)
  - ECP Printer Port (LPT1)
  - MSP-FET430UIF - VCP (COM38)
- Processors
- Smart card readers
- Sound, video and game controllers
- System devices
- Texas Instruments Emulators
- Universal Serial Bus controllers

Figure 9 Windows Device Manager (VCP)

The following paragraphs describe installation/de-installation of the certified VCP driver Installation

B.1 OS: Windows XP 32/64 Bit, Windows Vista 32/64 Bit and Windows 7 32Bit/64 Bit

To install the driver, run setup.exe, located at the directory \Driver\Usb\Certified\DriverInstall_3.2 on the installation media. This will install all the necessary files in the default directory C:\Program Files\Texas Instruments\MSP430\Drivers. The progress is shown by the following windows.



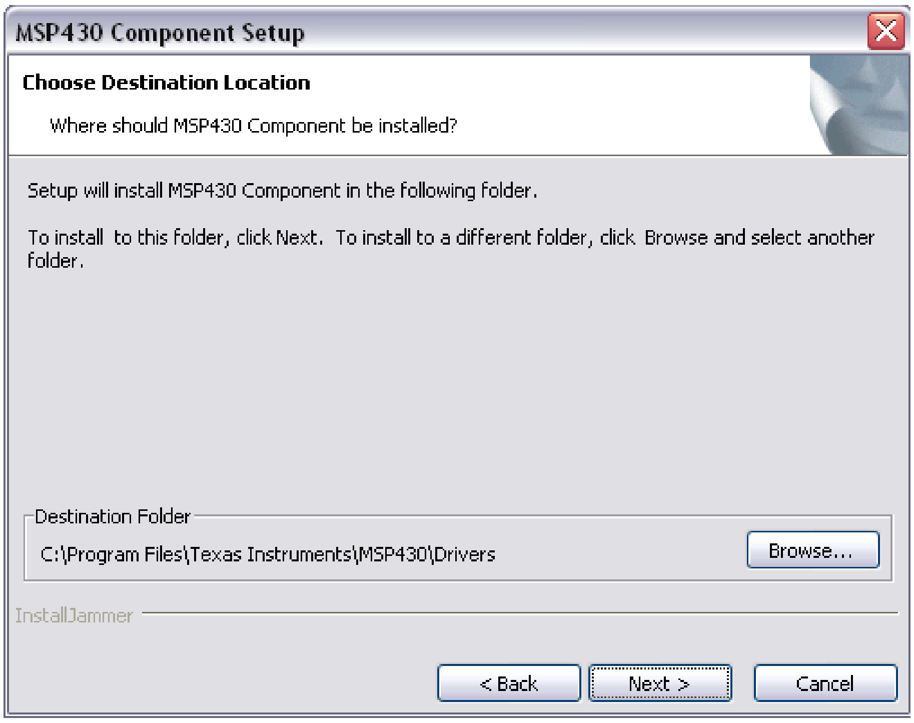
Welcome to the InstallJammer Wizard for MSP430 Component

This will install MSP430 Component version 9.0.334.0 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue or Cancel to exit Setup.

Figure 10 Driver installation process



MSP430 Component Setup

Choose Destination Location

Where should MSP430 Component be installed?

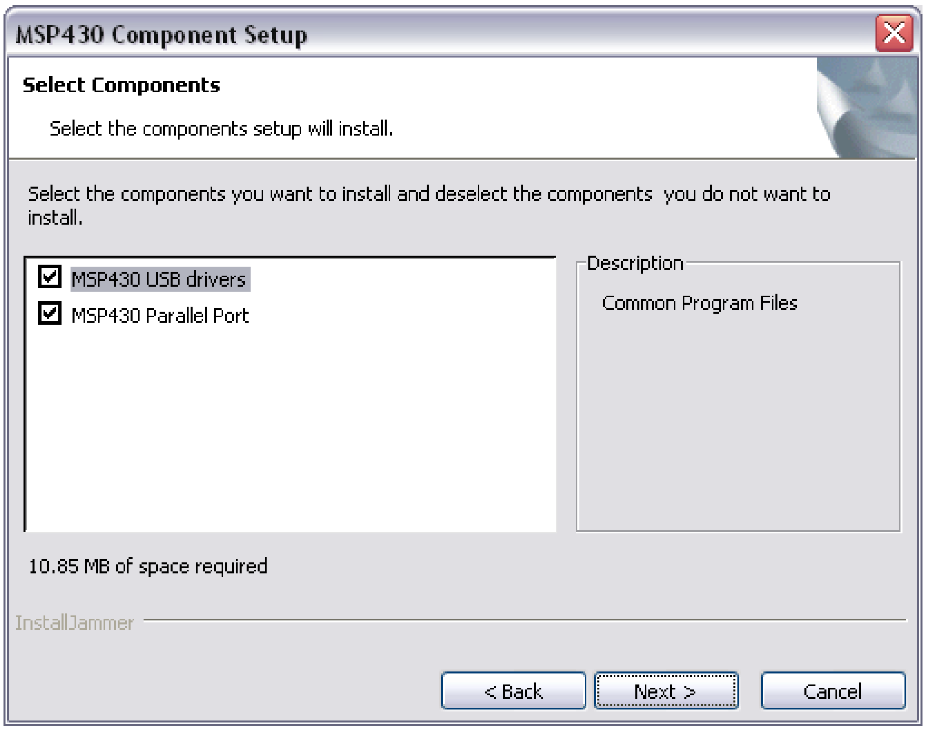
Setup will install MSP430 Component in the following folder.

To install to this folder, click Next. To install to a different folder, click Browse and select another folder.

Destination Folder

C:\Program Files\Texas Instruments\MSP430\Drivers

Figure 11 Default install folder



MSP430 Component Setup

Select Components

Select the components setup will install.

Select the components you want to install and deselect the components you do not want to install.

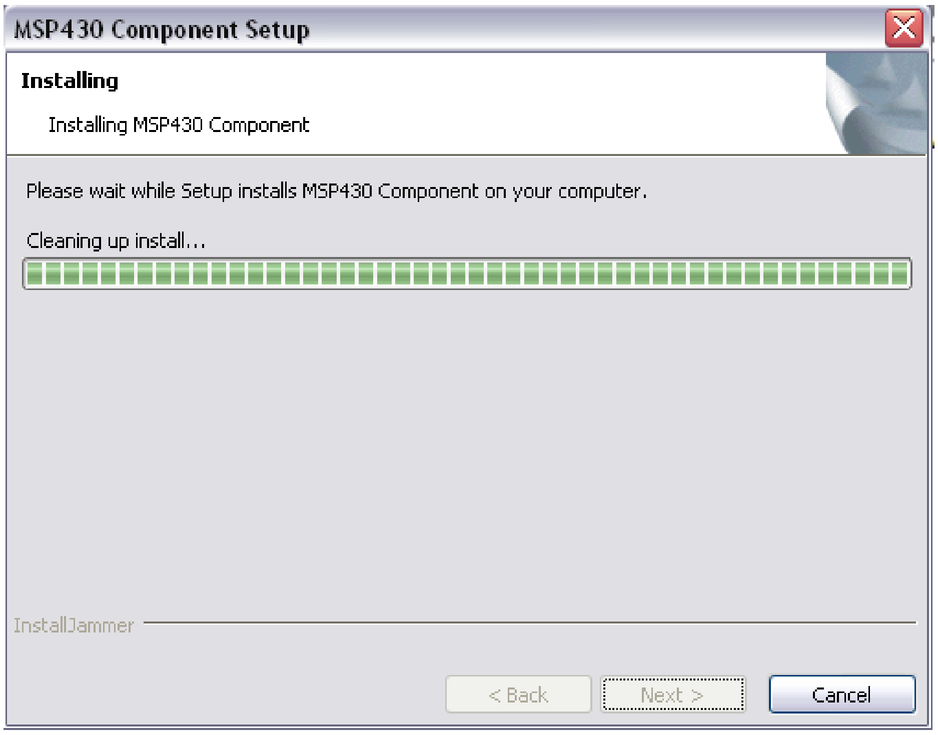
☑ MSP430 USB drivers
☑ MSP430 Parallel Port

Description

Common Program Files

10.85 MB of space required

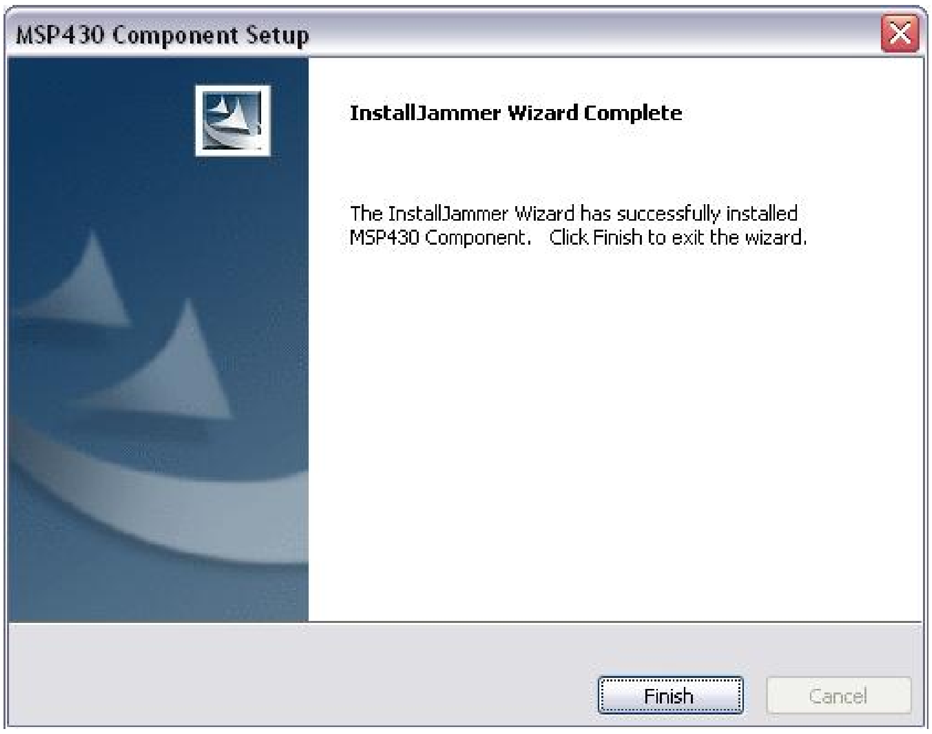
Figure 12 Select components for install



Please wait while Setup installs MSP430 Component on your computer.

Cleaning up install...

Figure 13 Installation process

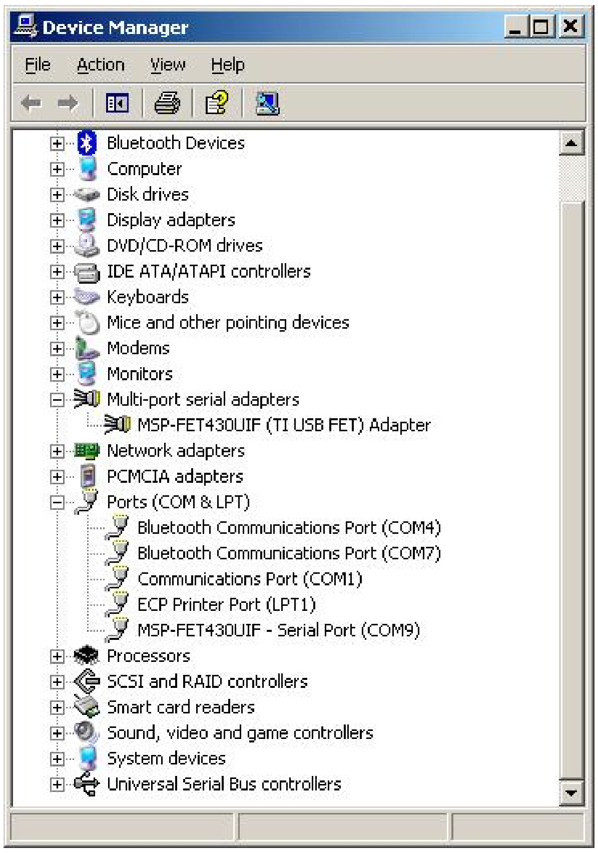


InstallJammer Wizard Complete

The InstallJammer Wizard has successfully installed MSP430 Component. Click Finish to exit the wizard.

Figure 14 Driver installation complete

The device manager display for a successful installation is shown in Figure 14 Choose My Computer → Properties → Hardware →Device Manager to see this window.



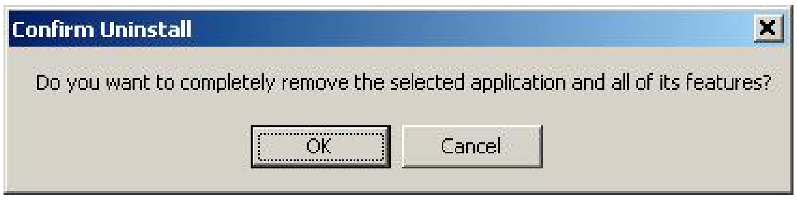
- Bluetooth Devices
- Computer
- Disk drives
- Display adapters
- DVD/CD-ROM drives
- IDE ATA/ATAPI controllers
- Keyboards
- Mice and other pointing devices
- Modems
- Monitors
- Multi-port serial adapters
  - MSP-FET430UIF (TI USB FET) Adapter
- Network adapters
- PCMCIA adapters
- Ports (COM & LPT)
  - Bluetooth Communications Port (COM4)
  - Bluetooth Communications Port (COM7)
  - Communications Port (COM1)
  - ECP Printer Port (LPT1)
  - MSP-FET430UIF - Serial Port (COM9)
- Processors
- SCSI and RAID controllers
- Smart card readers
- Sound, video and game controllers
- System devices
- Universal Serial Bus controllers

Figure 15 Windows Device Manager

B.2 Driver Uninstallation: Windows XP 32/64 Bit, Windows Vista 32/64 Bit and Windows 7 32/64 Bit

To uninstall the drivers, run setup.exe, located at directory \Installer of the installation media. Note that when setup.exe is run for the first time, it pre-installs the drivers. Running setup.exe a second time uninstalls all of the driver files and cleans up the registry entries that were added during installation.

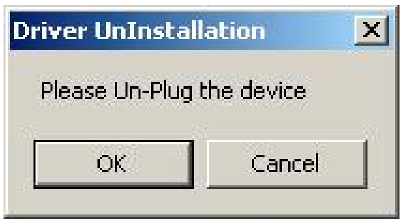
The uninstaller displays the dialog box shown in Figure 15 below. Select "OK" to uninstall the drivers.


Confirm Uninstall

Do you want to completely remove the selected application and all of its features?

Figure 16 Confirm uninstall

To continue uninstallation, un-plug MSP-FET430UIF hardware.


Driver UnInstallation

Please Un-Plug the device

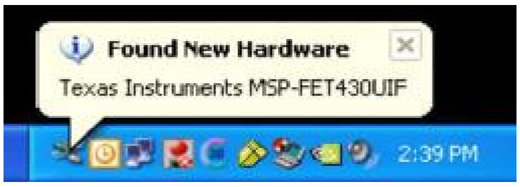
Figure 17 Driver uninstall

Once the device is unplugged, the uninstaller proceeds to clean up all the files and registry entries added during installation.

Click "Finish" to complete uninstallation.

Appendix C. Installation of CDC for MSP-FET430UIF

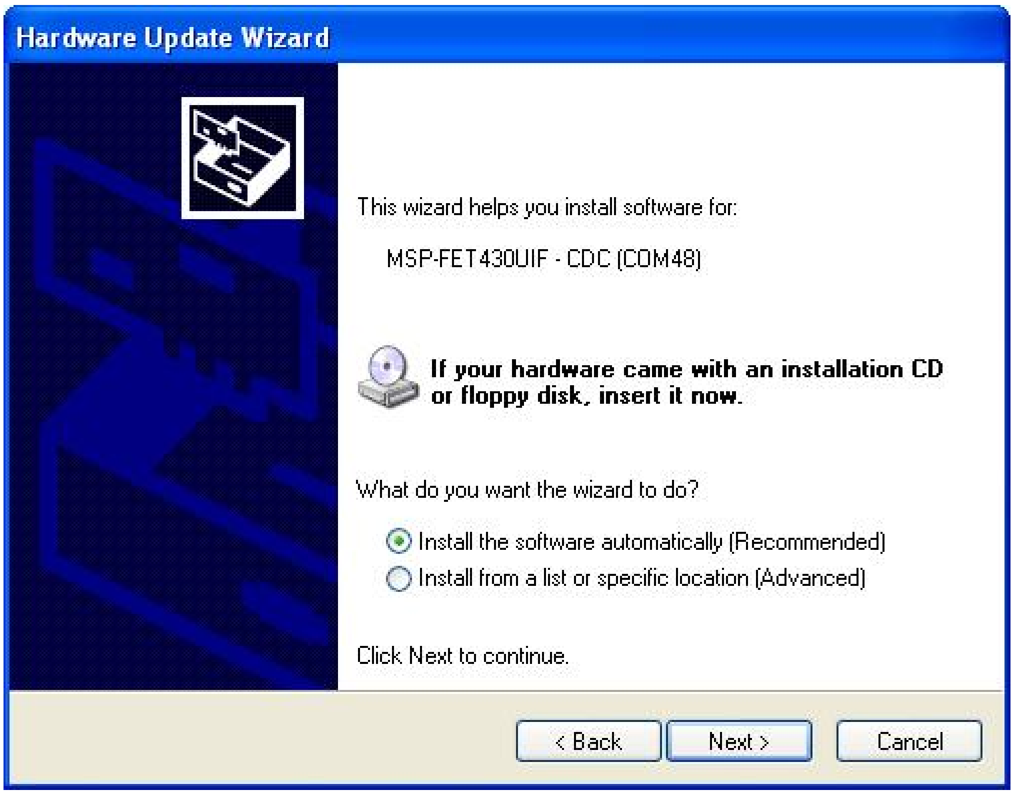
The UIF tries to enumerate a Virtual Com Port, which is based on Communication Device Class (CDC) driver. This CDC driver class is part of the Windows operating system but it requires an INF file for installation.
After plugging in the UIF, Windows recognizes a new hardware called MSP-FET430UIF and the following dialog appears.



Found New Hardware
Texas Instruments MSP-FET430UIF

Figure 18 New hardware

Afterwards the hardware wizard opens a new dialog window.
If CCS version 5 or IAR IDE is already installed, select "Install the software automatically".



Hardware Update Wizard

This wizard helps you install software for:

MSP-FET430UIF - CDC (COM48)

If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

[✓ Install the software automatically (Recommended)]
Install from a list or specific location (Advanced)

Click Next to continue.

Figure 19 Update Wizard

Appendix D. Update MSP-FET430UIF with hardware revision 1.3

If you are using a MSP-FET430UIF with hardware revision 1.3 your update process has one additional step, due to the fact that it is not possible to reset the TUSB3410 USB port controller during a firmware update.
Without a reset the TUSB can't change the VCP protocol to CDC and afterwards install the new communication core and JTAG stack. So it is necessary to reset the device manually by disconnecting the UIF and connect it to the PC again.
For IDE specific information on how to update an UIF with revision 1.3 please refer to the MSP-FET430UIF Debug FAQ (CCS > v5.1 and IAR EW > 5.40)

First you have to make sure that you are using an UIF with hardware revision 1.3. As you can see in Figure 20 and Figure 21 revision 1.3 has a CE sign on the front and no label with a revision number on the rear side.

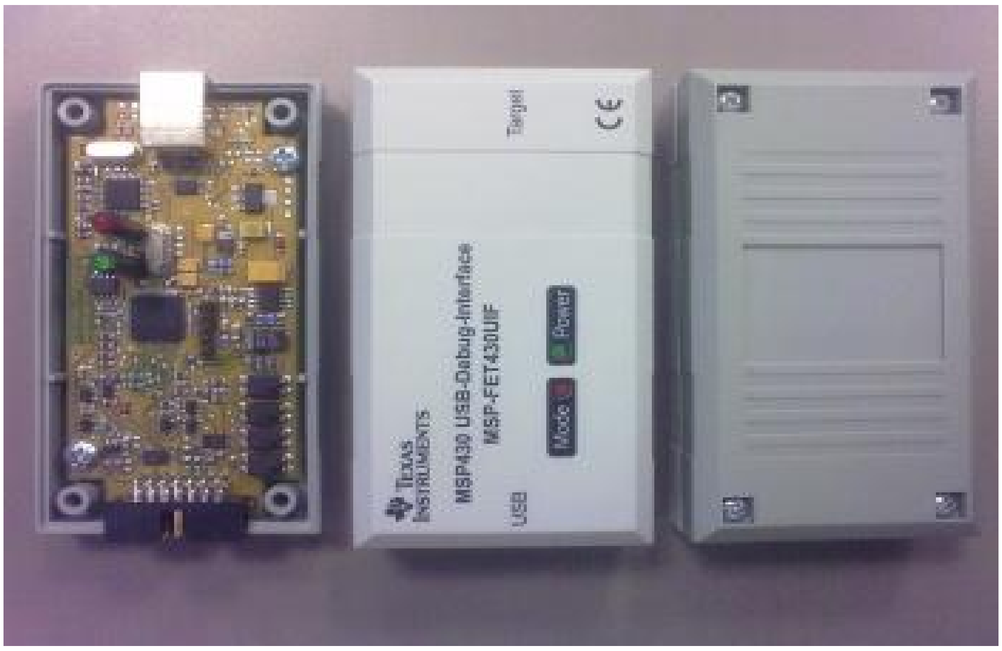


Figure 20 UIF Revision 1.3

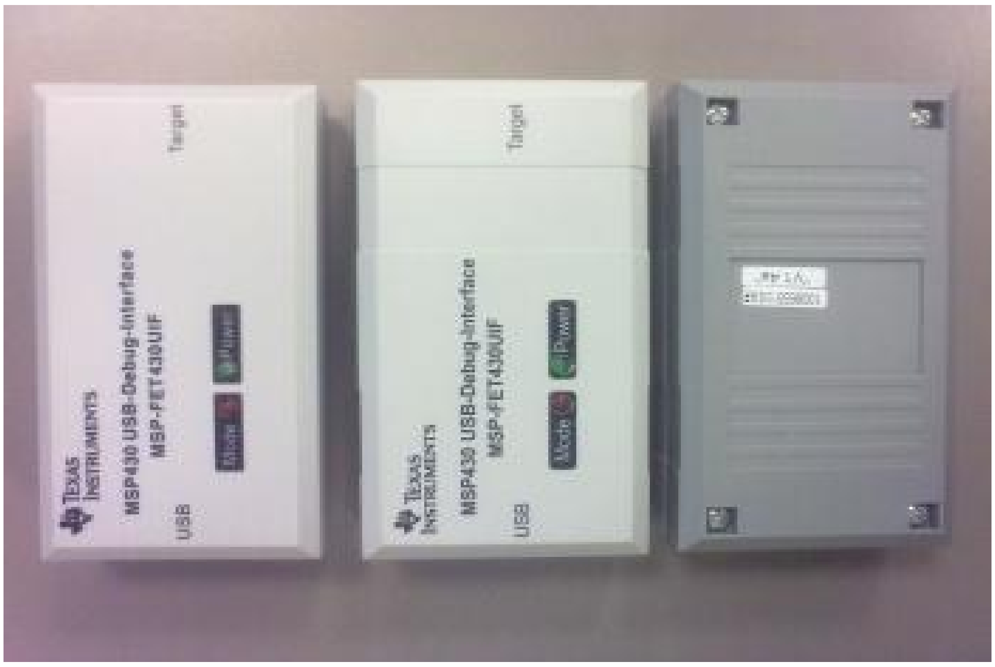


Figure 21 UIF Revision 1.4

A.1 References

TUSB3410 RS232/IrDA Serial-to-USB Converter
http://focus.ti.com/docs/prod/folders/print/tusb3410.html

TI Virtual COM Port Windows Drivers and Firmware
http://processors.wiki.ti.com/index.php/MSP430_JTAG_Interface_USB_Driver

USB/Serial Applications Using TUSB3410/5052 and the VCP Software (SLLA170)