MSP430™ Microcontroller Serial Programming Adapter

This document describes how to use the MSP430™ Microcontroller Serial Programming Adapter (MSP430-PRGS430). Instructions include how to install the software and hardware for the programmer and how to use the programmer to read to and write from MSP MCUs.

Contents

List of Figures

List of Tables

Trademarks

MSP430 is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Installation and Setup

This section describes how to install and program the hardware and software for the MSP430-PRGS430 programming adapter used with the MSP430 family of microcontrollers.

1.1 Installing the Software

To install the MSP-PRGS430 software, perform the following steps.

1. To ensure that you are using the latest version of the MSP-PRGS430 software, download the latest software from http://www.ti.com/lit/zip/slac029 and extract the installation executable (PRGS430_Rxxx.exe).
2. Run PRGS430_Rxxx.exe. A welcoming message is displayed.
3. Follow the setup instructions. The setup program guides you through the installation process.
4. During setup, the MSP430 program icons are installed in the selected folder. Click on the PRGS430 Read Me First icon (see Figure 1) for important information about the program device hardware and software.

Figure 1. ADT430 Program Icons

5. To start the programming adapter software, click the PRGS430 icon in the selected program group (default: ADT430).

1.2 Installing the Hardware

To install the programming adapter hardware, perform the following steps:

1. Using the 9-pin SUB-D connector, connect the programming adapter to the serial port (COM1 to COM4) of the PC.
2. Connect an external power supply to the programming adapter. The voltage of the power supply must be between 14 V and 20 V dc and must provide a minimum of 200 mA of power. The center terminal of the supply connector at the programming adapter is the positive pole.
3. The red LED on the programming adapter illuminates if the power supply is properly connected. If the LED does not illuminate and the power supply is properly connected, check the F1 fuse on the programming adapter printed-wire board (PWB).
4. Connect the MSP430 MCUs, in a socket or on a PWB, to the programming adapter through the 14-pin cable.

   The programming adapter provides the selected supply voltage VCC at pin 14 of the 25-pin SUB-D connector, or at pin 2 of the 14-pin connector to supply the MSP430 MCU. The signal name is VCC_MSP.

   If an external supply voltage VCC is used for the MSP430, the internal voltage VCC_MSP must be set to the same voltage level.

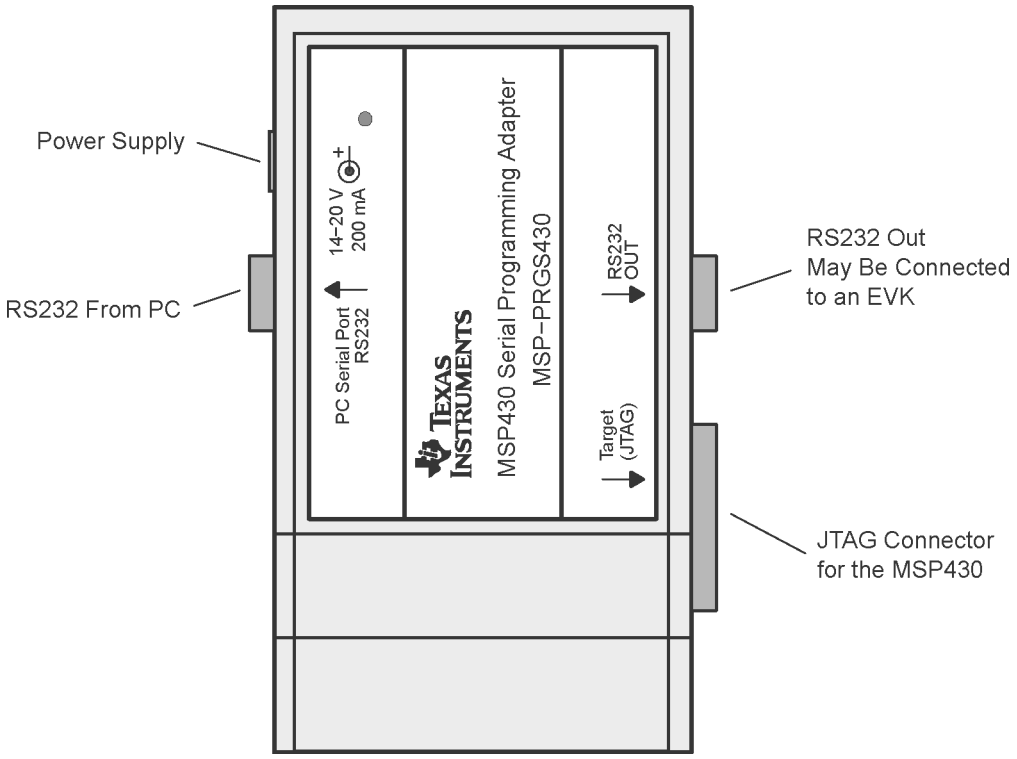


Figure 2. Serial Programming Adapter

2 Operation

This section describes the programming procedure for MSP430 MCUs and the error messages you may encounter during the procedure.

2.1 Software and Hardware Layers of the PRGS430 Environment

Figure 3 shows the layers of the environment and the communication options.

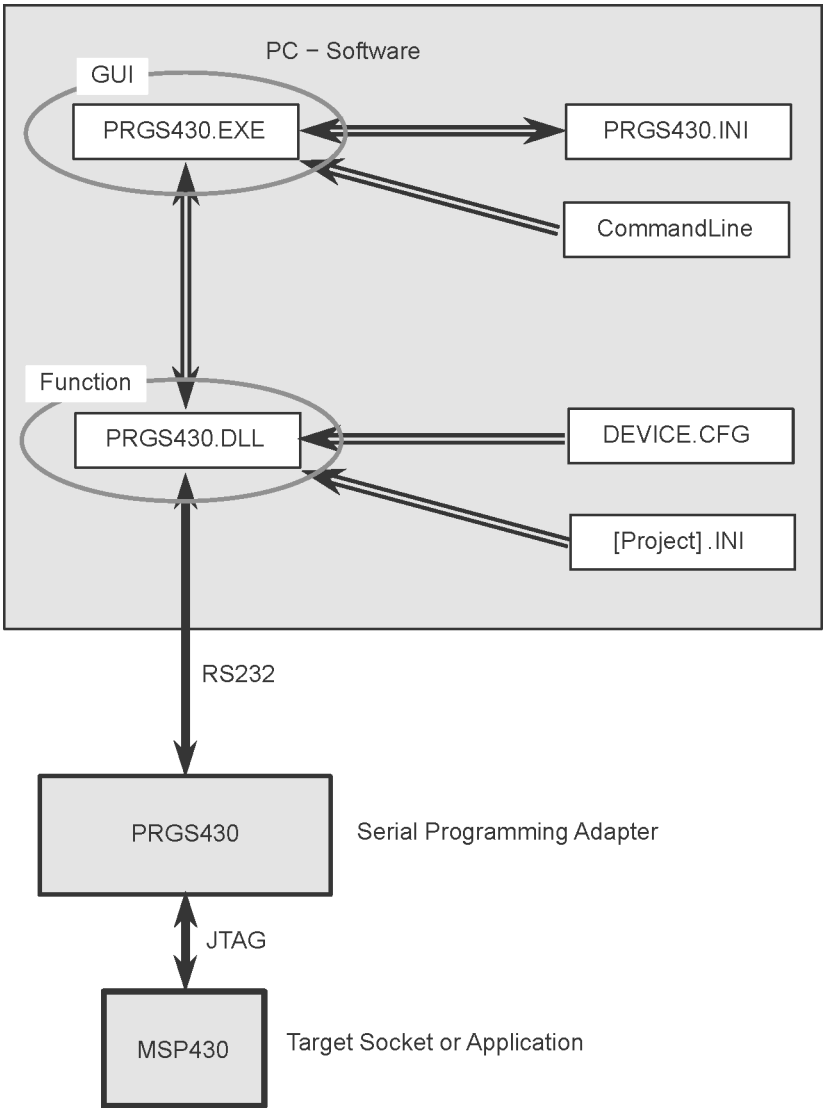


Figure 3. Software and Hardware Layers

The options to handle and communicate with the PRGS430 hardware are:

- Using the graphical user interface (see Section 2.2)
- Using command line parameters (see Section 2.3)
- Using the PRGS430.dll (see Section 2.4). This is the fastest way if the PRGS430 is used in an in-system program and test environment.

2.2 Programming MSP430 Devices With the GUI

2.2.1 Basic Procedure

To program the MSP430 MCUs:

1. Click the Program Device icon during the installation-selected program group (default: ADT430). The MSP430 programmer dialog box appears. The status line at the bottom of the window shows the actual or the most recent activity (see Figure 4). The status line displays the message "Connecting to adapter..." until the programming adapter is detected and the baud rate is set.
2. Select the correct device and supply voltage.
3. Select the name of the object file (TI-TXT (.txt) or Intel-hex (.a43) format).
4. Select the additional options to program, if necessary using Erase Flash, Erase Check, or Verify (see Table 1).
5. Click the Program button to start programming. The status line at the bottom of the window shows the actual or most recent activity (see Figure 4).

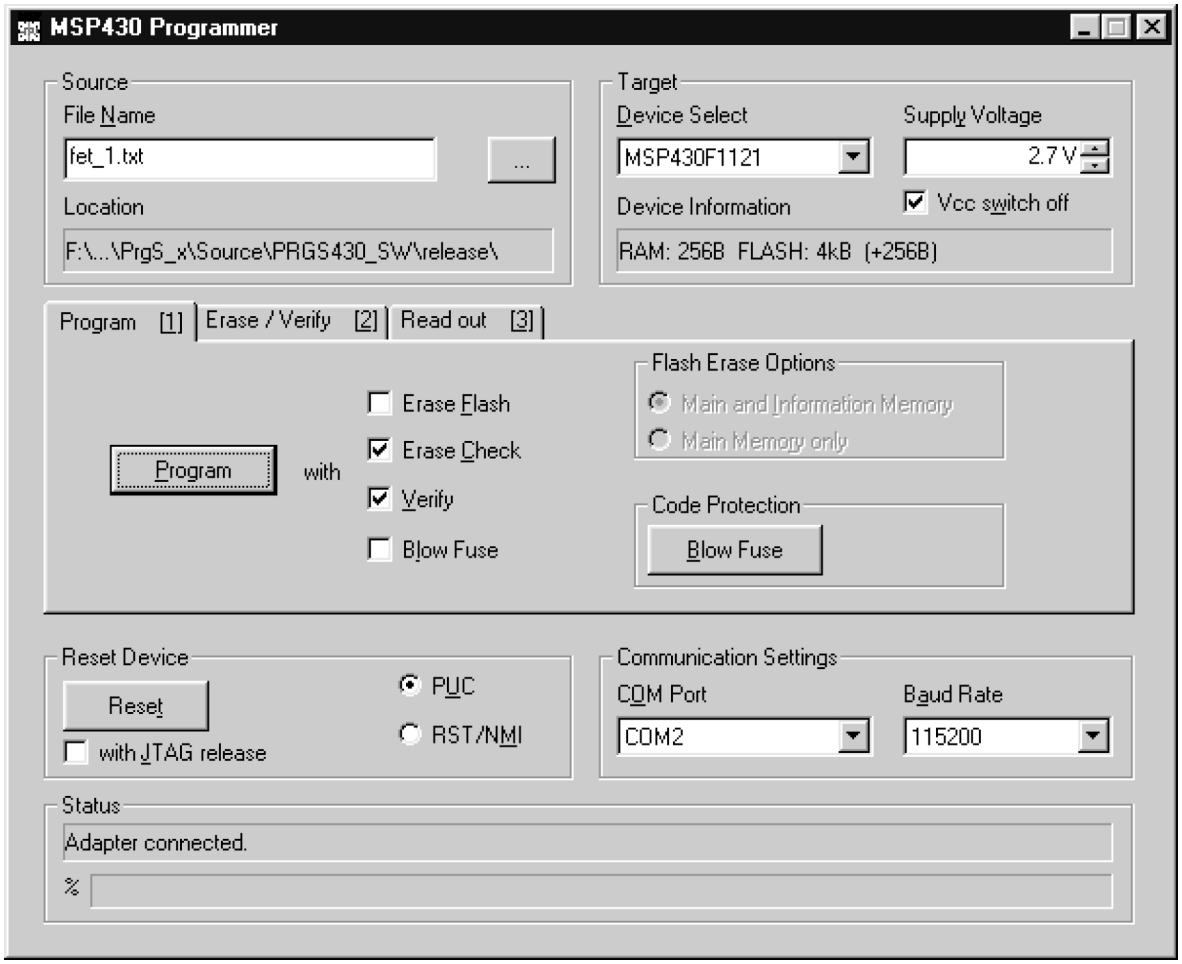


Figure 4. MSP430 Programmer Dialog Box

2.2.2 Description of the MSP-PRGS430 GUI

To program an MSP430 MCU:

1. Select the file that contains the data to program from the MSP430 programmer dialog box (see Figure 4).
2. Select the device. An error message appears on the screen if the device selected is different or not connected.
3. Set the required supply voltage, communication port COMx, and baud rate. The device configuration and memory type are selected automatically according to the selected device.
4. Click the Program button to start the programming operation.

Table 1 describes the function of the buttons for different options and combinations for the MSP430 programmer dialog box.

Table 1. MSP430 Function Buttons and Descriptions

Button Name	Functions	Description
File Name		Selects the name of the file to program (Intel-hex or TI-TXT format)
Device Select		Selects the MSP430 MCU type to program from a list
Supply Voltage	V_{CC} switch off	Selects the supply voltage for the MSP430
		If selected (default), the supply voltage is switched off after each MSP430 access; otherwise, the supply voltage remains connected.
Program		An object code is programmed to the on-chip memory using the select options.
	With Erase Flash	Memory is erased before programming (only with flash devices). The following options are possible:− Main and information memory− Main memory only
	With Erase Check	Erase check is performed before programming operation is executed.
	With Verify	Each section is verified after it is programmed, or an error message is displayed if verification fails.
	With Blow Fuse	The code-protection fuse is blown after the entire object code, with verify, is programmed. This action is irreversible and disables future on-chip memory access (reading or programming). This step is not performed if verify is disabled or verify fails. A warning is displayed.
Erase Flash		Erase operation can be done only with flash devices, according to the selected option.
	By file	Only the memory locations corresponding to the selected object file are erased. All other memory locations keep their old data (smart erase).
	By device	The entire flash memory of the device is erased.
	By range	An erase is performed depending on the values entered in the range fields.
Erase Check		Checks if memory locations are erased.
	By file	Checks only the memory locations used by the selected object file.
	By device	Checks the entire programmable memory of the device. (No RAM is checked.)
	By range	An erase check is performed according to the range of memory locations in the range for Erase Check/Readout field.
Verify		Verify the data in the MSP430 MCU according to the selected option.
	By file	A verification of the memory locations vs the selected object file is performed. (By file and by device are the same functions.)
	By device	
	By range	Verify memory locations defined in the range field vs the data in the selected file. The defined range should not contain memory locations outside the data stored in the selected file, otherwise an error is reported.
Blow Fuse		The on-chip security fuse is irreversibly disabled and any access, such as reading or programming of the MSP430, is impossible through JTAG. Access through bootstrap loader interface is possible for devices that support that interface.

Table 1. MSP430 Function Buttons and Descriptions (continued)

Button Name	Functions	Description
Read		Read data from MSP430 MCU. When this function is executed, a dialog box appears; the file name for the data to store should be selected.
	By device	Read the entire memory of the device and store the data into the file selected in the file name field.
	By range	Read the memory locations selected by the range field and store the data in the file selected in the file name field.
Reset		The reset of a MSP430 can be performed in two ways. After reset, the MSP430 may remain under JTAG control or can be released to operate normally and execute the program.
	PUC	A software reset of the chip is generated.
	$\overline{\text{RST}}$/NMI	Generates a hardware reset by applying a low pulse on $\overline{\text{RST}}$/NMI pin.
	With JTAG release	JTAG is released after the execution of the reset (through JTAG or $\overline{\text{RST}}$/NMI).
COM Port		Selects the COM port to which the programming adapter is connected
Baud Rate		Selects the baud rate for communication with the programming adapter hardware
Help		Help is available for programming MSP430 MCUs, command buttons, selectors, and the object file format used. The Help menu can be found in the system menu of the serial programming adapter software (right click on the symbol at the upper-left corner of the program window) or with the F1 function key.

NOTE: For some MSP430 family members, for example, MSP430F2xxx devices, portions of flash information memory are factory preprogrammed with calibration data. Depending on which method is used for erasing the flash memory, this calibration data may be erased. Should the calibration data be conserved, it must be read prior to the information memory erase or a flash erase method that does not affect the calibration data memory locations must be used. See the device-specific data sheet for further information on preprogrammed calibration data memory locations.

If it should be saved, the following erase options must not be used as the Info memory will be entirely erased:

- Program with Erase Flash (flash-erase options: Main and Info Memory)

  or
- Erase Flash by Device

Instead, "Erase Flash by File" or "Erase Flash by Range" should be used.

2.2.3 Error Messages

One of the following messages may show if JTAG communication is not established correctly:

- If the MSP430 MCU to program can not be found, the message shown in Figure 5 appears. This problem can be caused by the PRGS430 not being connected to the hardware, the device not inserted or incorrectly inserted into the socket, or the device not powered.
- The problem could be that the PRGS430 is not connected to the hardware, the device is not inserted or is incorrectly inserted into the socket, or the device is not powered.

Figure 5. Communication Error Box

If the fuse is already blown, the error message shown in Figure 6 appears.

Figure 6. Communication Error Box for Blown Fuse

Additional message boxes appear for general error messages, such as erase check (see Figure 7).

Figure 7. Erase Check Error Message

When a read error is detected in the input file, such as a format error, the following message is displayed (see Figure 8).

Figure 8. Data Error

Table 2. Error Messages

Error Type	Error Message
Communication	Communication failed!
Communication	Adapter not connected!
Communication	Synchronization with adapter failed!
Communication	The present adapter is not an MSP-PRGS430!
Communication	Missing setting of VCC!
MSP430	Target not connected!
MSP430	Wrong JTAG version!
MSP430	PUC failed!
MSP430	Wrong target!
MSP430	Target fuse is blown!
MSP430	Blown fuse failed!
MSP430	Supply voltage to low!
MSP430	Fuse not released for this device!
Setting	Unknown target!
Setting	No target selected!
Setting	Wrong VCC selected!
Setting	Wrong baud rate!
Setting	Communication port error!
Setting	The selected range is invalid!
Setting	Wrong argument!
Setting	Error at target address (during erase check or verify)
Setting	Unknown command line option
Setting	Command line option out of valid range
System	DEVICE.CFG corrupted
System	General error!
System	File type could not be detected!
System	Unexpected end of file!
System	PROJECT.INI corrupted!
System	Filename mismatch
System	Error in DEVICE.CFG
Windows	Error during file I/O

2.2.4 Content of PRGS430.ini File

The last settings of the PRGS430 graphical user interface (GUI) are stored in the .ini file before exiting the program. This information is stored under the Program Device System section.

Additionally, the following parameters are in the [Options] section and may be modified:

[Options]

\BlowFuse = 1 → The blow fuse button in the GUI is disabled to prevent accidental blow of the irreversible fuse.

LastResult = 0 → If the program is called with command-line parameter, the error code that is returned to the system when exiting the program is also stored here.

2.2.5 Use of a [Project].ini File

Some default options could be changed within a [Project].ini file. This file has to be in the same directory as the object code file. The following variables could be defined or redefined there.

The name of the file should have the same name as the object file with the extension .ini.

```
[ProgramDevice]
UserMemProtect = Start, Size
UserMemProtect2 = Start, Size
UserMemProtect3 = Start, Size
UserMemProtectn = Start, Size
DisableTIMemProtect = 0
```

Memory ranges defined in the UserMemProtect and UserMemProtect [n] option are read and reprogrammed after erase (flash device only). [n] could be a number ≥2 and be in ascending order.

If a memory protection is activated in the device definition file from TI, it can be switched off with the DisableTIMemProtect = 1 option.

2.3 Command Line Options

2.3.1 General Definitions

0: Off 1: First selectable option
1: On 2: Second selectable option
3: Third selectable option

The PRGS430.ini file options are used if they are not specified in the command line. The command line option overwrites the .ini file options.

The program exits automatically if a command is passed in the command line and the command was executed. There is only a small status window opened during the execution.

Only one command identifier (/cmd:) is allowed within the command line. Otherwise, the execution is canceled and an error is returned.

If an error in the command line parameter is detected, the program exits with an error message.

The *filename* may also contain a path. If special characters are used, the string has to be inside quotes (for example, \\server\adt430\PRG files\test.txt).

If an error is detected in the filename, the operation is canceled and an error is returned.

Table 3. Command Line Options

Command	
/cmd:PRG	Program command
/cmd:VFY	Verify command
/cmd:ERS	Erase command
/cmd:CHK	Erase check command
/cmd:READ	Read command
/cmd:RST	Reset command
/cmd:BLOW	Blow fuse command
Options	
/COM:x	Specifies the serial port: /COM:1, /COM:2, /COM:3, or /COM:4
/BR:xxxxxx	Sets baud rate to be used: 9600/19200/38400/57600/115200, for example, /BR:57600
/Dev:	Selects the device according to the name in the device.cfg file, for example, /Dev:MSP430F1121
/SVolt:x.x	Selects supply voltage MSP_VCC of the programming adapter. The voltage is supplied between GND and MSP VCC, for example, /Svolt: 3.0.

Table 3. Command Line Options (continued)

/SVoff:{0,1}	Switches off supply voltage MSP_VCC after execution 0: Disable (do not switch off) 1: Enable (switch off)
filename	Specifies name of the object file to be programmed or verified
/FILE filename	(Second way to define the filename − space separated)
Options for Program Command	
/PE:{0,1,2}	Option program with erase (flash only) 0: Without erase 1: Main and Info memory 2: Main memory only
/PC:{0,1}	Option program with erase check 0: Disable 1: Enable
/PV:{0,1}	Option program with verify 0: Disable 1: Enable
/PB:{0,1}	Option program with blow fuse (only valid with verify successful) 0: Disable 1: Enable
Options for Erase, Erase Check, and Verify Command	
/E:{1,2,..}	Option for the erase, erasecheck, or verify by file/device/range 1: File 2: Device 3: Range
/ERange:0xXXX, 0xYYYY	Option for the erase, erasecheck, or verify range (start: 0xXXXX, length: 0xYYYY)
Options for Read Command	
/RO:{1,2}	Option read by device or range 1: Device 2: Range
/RRange:0xXXXX,0xYYYY	Option read range (start: 0xXXXX, length: 0xYYYY)
/Rfile:file- name,{1,2}	Specifies read file name 1: TI-TXT 2: Intel-hex (default directory should be the last object file directory)
Log Options	
/Log:filename	Specifies log file name (default directory should be the PRGS430.exe directory)
/ALog:{0,1}	Option accumulative Log file 0: Disable 1: Enable

Example

```
PRGS430.exe "C:\adt430\test\test.txt" /Dev:MSP430F1121 /cmd:PRG /PE:1 /PC:0
/PV:1 /COM:2
```

This command programs the file test.txt, located in the directory C:adt430\test, into a MSP430F149 device. The device is erased before programming. The erase check is disabled. The code is verified after programming. The programming adapter is connected to ComPort 2. The baud rate is not passed with the command line, so the setting in the PRGS430.ini file is used.

2.3.2 Return Values and Error Codes in .ini File

The error code is returned to the PC operating system and also is stored in PRGS430.ini file in the [Options] section:

```
LastResult=0
```

0	Ok
2	Communication failed!
3	Target not connected!
4	Adapter not connected!
5	Wrong JTAG version!
6	PUC failed!
7	Synchronization with adapter failed!
8	The present adapter is not an MSP-PRGS430!
9	Unknown target!
10	Wrong target!
11	No target selected!
12	Target fuse is blown!
13	Blow fuse failed!
14	Missing setting of VCC!
15	Wrong VCC selected!
16	Wrong baudrate!
17	Communication port error!
18	DEVICE.CFG corrupted!
19	General error!
20	The selected range is invalid!
21	Wrong argument!
22	Error during file I/O
23	File type could not be detected!
24	Unexpected end of file!
25	PROJECT.INI corrupted!
26	Vcc voltage too low for selected function!
27	Fuse not release for this device!
101	Error at target address (during erase check or verify)
102	Unknown command line option
103	Command line option out of valid range
104	Filename mismatch
105	Error in device.cfg

2.4 PRGS430.DLL Description

The PRGS430.dll is used to communicate with the MSP-PRGS430 hardware and the connected MSP430 MCU.

The initialization of the PRGS430 should be done with this sequence:

InitCom
SetDeviceType
SetVCC
. InitTarget
. ...
. ReleaseTarget
ReleaseCom

Several examples showing how the DLL could be used are located in the DLL_Usage_Examples folder of the PRGS430 system. This dll can be used separately using the conventions in the following sections.

2.4.1 /FN0001/ InitCom

InitCom initializes (opens) the given communications port, establishes communication with the PRGS430 hardware, and sets the baud rate of the MSP-PRGS430. If successful, the MSP-PRGS430 is reset and V_{CC} is set to 0.0 V (the voltage should be set after the first user action to validate the correct value).

```
long int InitCom(char* lpszComPort, long int lBaudRate)
```

lBaudRate

Valid baud rates are: 9600, 19200, 38400, 56800, and 115200 baud. The default baud rate after installation is 115200 baud.

lpszComPort

The name of the communication port—COM1, COM2, COM3, or COM4.

Example

```
lFuncReturn = InitCom("COM1" 115200)
```

2.4.2 /FN0002/ ReleaseCom

This new function is the counterpart to InitCom. It allows to close a communication with the MSP-PRGS430 hardware. VCC is set to 0 and all outputs are set to the Hi-Z state.

```
long int ReleaseCom (void)
```

Example

```
lFuncReturn = ReleaseComm()
```

2.4.3 /FN0003/ SetDeviceType

Selects the device type.

```
lFuncReturn = SetDeviceType(char* lpszDeviceName)
```

lpszDeviceName

Name of the device in file device.cfg

Example

```
lFuncReturn = SetDeviceType("MSP430F1121")
```

2.4.4 /FN0004/ InitTarget

Initializes the JTAG access to the target device, detects the device type, and reports when the detected device does not match the parameter DeviceName passed.

```
long int InitTarget(char* lpszDeviceName)
```

lpszDeviceName

Name of the device in file device.cfg

Example

```
lFuncReturn = InitTarget ("MSP430F1121")
```

2.4.5 /FN0005/ ReleaseTarget

This function performs a PUC and releases the JTAG access to the target device. All JTAG signals from the serial programming adapter are switched to Hi-Z. The device starts program execution if it is still connected to VCC.

```
long int ReleaseTarget(void)
```

Example

```
lFuncReturn = ReleaseTarget()
```

2.4.6 /FN0006/ Erase

This function erases flash memory (if available). The protection of areas can be disabled by setting the DISABLE_TI_MEM_PROTECT−Bit in Flags.

```
long int Erase(long int wStart, long int wLength, long int Flags)
```

wStart

Start address of the area to be erased. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

wLength

Length of the area. Allowed values are 0x0000 to 0xFFFE (see the memory map of the corresponding device).

If erasing information and main memory segments, first erase the information memory segments, then erase the main memory segments.

The mass erase sequence then would be:

```
Erase(InfoStart,InfoLength,0);
Erase(MainStart,MainLength,0);
```

The function invokes a mass erase for information or main memory if the start address and the complete memory range of the information or main memory is specified. If the range specified is not complete for that device, a segment erase of the individual segments is performed.

Flags

DISABLE_TI_MEM_PROTECT (0x01)

If this bit is set, the memory protection settings in device.cfg are ignored.

Example

```
lFuncReturn = Erase(long:0xF000, long:0x1000, long:1)
```

For some MSP430 family members (for example, MSP430F2xxx devices) portions of flash information memory are factory preprogrammed with calibration data. Depending on which method is used for erasing the flash memory, this calibration data may be erased. Should the calibration data be conserved, it must be read prior to the information memory erase or a flash erase method that does not affect the calibration data memory locations must be used. See the device-specific data sheet for further information on preprogrammed calibration data memory locations.

2.4.7 /FN0007/ EraseFile

Erases all addresses used in the specified file.

```
long int EraseFile(char* lpszFileName, long int iFileType, long int Flags,
char* lpszProjectIni)
```

iFileType

FILETYPE_AUTO (0x00): Autodetection of file type (Intel-hex or TI-TXT)

FILETYPE_TI_TXT (0x01): File type is TI-TXT

FILETYPE_INTEL_HEX (0x02): File type is Intel-hex

Flags

DISABLE_TI_MEM_PROTECT (0x01)

If this bit is set, the memory protection setting device.cfg is ignored. lpszProjectIni: name of the {project}.ini file, if protection settings from this file shall be used. If there is no protection, replace lpszProjectIni with NULL.

Example

```
lFuncReturn = EraseFile("text.txt", long:0, long:0, NULL)
```

For some MSP430 family members, for example MSP430F2xxx devices, portions of flash information memory are factory preprogrammed with calibration data. Depending on which method is used for erasing the flash memory, this calibration data may be erased. Should the calibration data be conserved, it must be read prior to the information memory erase or a flash erase method that does not affect the calibration data memory locations must be used. See the device-specific data sheet for further information on preprogrammed calibration data memory locations.

2.4.8 /FN0008/ EraseCheck

Performs an erase check of an area of the target memory

```
long int EraseCheck(long int wStart, long int wLength)
```

wStart

Start address of the memory area. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

wLength

Size of the area. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

The function EraseCheck() simply uses PatternCheck(), with 0xFFFF as pattern.

```
EraseCheck(long int wStart, long int wLength)
{
return PatternCheck(wStart, wLength, 0xFFFF);
}
```

Example

```
lFuncReturn = EraseCheck(long:0xF000, long:0x1000)
```

2.4.9 /FN0009/ EraseCheckFile

Checks if all memory addresses in the file are erased.

```
long int EraseCheckFile(char* lpszFileName, long int iFileType)
```

lpszFileName

Name of the file

iFileType

FILETYPE_AUTO (0x00): autodetection of file type (Intel-hex or TI-TXT) FILETYPE_TI_TXT (0x01): file type is TI-TXT

FILETYPE_INTEL_HEX (0x02): file type is Intel-hex

The function returns success or the first address with mismatching data.

Example

```
lFuncReturn = EraseCheckFile("test.txt", long:0)
```

2.4.10 /FN00010/ PatternCheck

Checks a memory range with word pattern passed.

```
long int PatternCheck(long int wStart, long int wLength, long int wPattern)
```

wStart

Start address of the memory area. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

wLength

Size of the area. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

wPattern

Word pattern for check

The function returns success or the first address with mismatching data.

Example

```
lFuncReturn = PatternCheck(long:0xF000, long:0x1000, long:0xFFFF)
```

2.4.11 /FN00011/ VerifyData

This function verifies the content of the device with the data stored at passed pointer to data.

```
long int VerifyData(long int wStart, long int wLength, void* lpData)
```

wStart

Start address of memory area. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

wLength

Length of the memory area to be checked. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

lpData

Pointer to buffer with data bytes in it

The function returns success or the first address with mismatching data.

Example

```
lFuncReturn = VerifyData(long:0xF000, long:0x1000, void* lpData)
```

2.4.12 /FN00012/ VerifyFile

This function checks if the memory contents of the target device are equal to the file contents.

```
long int VerifyFile(char* lpszFileName, long int iFileType)
```

lpszFileName

Name of the file

iFileType

FILETYPE_AUTO (0x00): autodetection of file type (Intel-hex or TI-TXT)
FILETYPE_TI_TXT (0x01): file type is TI-TXT
FILETYPE_INTEL_HEX(0x02): file type is Intel-hex

The function returns success or the first address with mismatching data.

Example

```
lFuncReturn = VerifyFile("test.txt", long:0)
```

2.4.13 /FN00013/ VerifyFileRange

This function evaluates if the memory contents of the target device are equal to the file contents in a passed range.

```
long int VerifyFileRange(char* lpszFileName, long int iFileType, long int
wStart, long int wLength)
```

lpszFileName

Name of the file

iFileType

FILETYPE_AUTO (0x00): autodetection of file type (Intel-hex or
TI-TXT) FILETYPE_TI_TXT (0x01): file type is TI-TXT
FILETYPE_INTEL_HEX (0x02): file type is Intel-hex

wStart

Start address of memory area. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

wLength

Length of the memory area to be checked. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

The function returns success or the first address with mismatching data.

Example

```
lFuncReturn = VerifyFileRange("test.txt", long:0, long:0xF000, long:0x1000)
```

2.4.14 /FN0014/ ProgramData

This function writes data into an MSP430 MCU. Protection of ranges of memory locations defined in the DEVICE.CFG file can be disabled by setting the DISABLE_TI_MEM_PROTECT−Bit in Flags.

```
long int ProgramData(long int wStart, long int wLength, void* lpData)
```

wStart

Start address of the range that is to be erased. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

wLength

Length of the range. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

lpData

Pointer to the Data to be programmed

Flags

The bits in Flags control the operation of ProgramData().

Example

```
lFuncReturn = ProgramData(long:0xF000, long:0x1000, void* lpData)
```

2.4.15 /FN0015/ ProgramFile

This function writes data from the file to the MSP430 MCU. The protection of ranges of memory locations defined in the DEVICE.CFG file can be disabled by setting the DISABLE_TI_MEM_PROTECT−Bit in Flags.

If the PGM_WITH_ERASE option are not selected the program data is preserved. This means that the data is read from the flash and replaced with the new data. All other information is restored automatically. The handling of the information memory works with the same mechanism.

For the fastest programming speed, select the PGM_WITH_ERASE and PGM_ERASE_INFO options.

```
long int ProgramFile(char* lpszFileName, long int iFileType, long int iFlags,
char* lpszProjectIni)
```

lpszFileName

Name of the file to be written into the target

iFileType

FILETYPE_AUTO (0x00): autodetection of file type (Intel-hex or

TI-TXT) FILETYPE_TI_TXT (0x01): file type is TI-TXT

FILETYPE_INTEL_HEX (0x02): file type is Intel-hex

iFlags

DISABLE_TI_MEMPROTECT (0x01)

PGM_WITH_ERASE (0x02) // Erases the main memory before programming

PGM_ERASE_INFO (0x04) // Erases the info memory before programming

PGM_WITH_ERASECHECK (0x08) // Erase check by device and the programs the device

PGM_WITH_Verify (0x10) // Read device, merge with file to be programmed, and write back to device

Use PGM_ERASE_INFO only together with the PGM_WITH_ERASE flag.

lpszProjectIni

Name of the {project}.ini file, if protection settings from this file are used. If no protection is required, replace lpszProjectIni with NULL.

The added features do not need to be used. For example, to call ProgramFile according to older specification, call:

```
ProgramFile(FileName, FileType, 0, NULL)
```

If no {project}.ini file or erase check is used, call:

```
lFuncReturn = ProgramFile(FileName, 0, 0, NULL); // with autodetect file type
```

If an erase or erase-check function reports an error, the function ProgramFile() is aborted before programming is started.

2.4.16 /FN0016/ BlowFuse

This function blows the security fuse of the target device.

```
long int BlowFuse(void)
```

Example

```
lFuncReturn = BlowFuse(void)
```

2.4.17 /FN0017/ SetVcc

Sets the VCC_MSP voltage of the programming adapter to the given value.

```
long int SetVcc(long int iVoltage)
```

iVoltage

VCC in mV (for example, 3000 = 3 V)

The voltage range is limited to the voltage allowed for the selected MSP430 MCU.

Example

```
lFuncReturn = SetVcc(Long:3000)
```

2.4.18 /FN0018/ ReadOutData

Reads data from the device and writes it to the buffer.

```
long int ReadOutData(long int wStart, long int wLength, void* lpBuffer)
```

wStart

Start address of the area to be read. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

wLength

Length of the area. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

lpBuffer

Pointer points to a buffer that receives the data. The buffer must be large enough to hold the entire data; otherwise, a fatal error of the operating system may occur.

Example

```
lFuncReturn = ReadOutData(long:0xF000, long:0x1000, void* lpBuffer)
```

2.4.19 /FN0019/ ReadOutFile

Reads data from the device and writes it to a file.

```
long int ReadOutFile(long int wStart, long int wLength, char* lpszFileName,
long int iFileType)
```

wStart

Start address of the area to be read. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

wLength

Length of the area. Allowed values are 0x0000 to 0xFFFE (see the memory map in the device-specific data sheet for the supported memory size).

lpszFileName

Name of the file to receive data. If the file does not exist, it is created; If the file already exists, it is overwritten.

iFileType

FILETYPE_TI_TXT (0x01): file type is TI-TXT

FILETYPE_INTEL_HEX (0x02): file type is Intel-hex

Example

```
lFuncReturn = ReadOutFile(long:0xF000, long:0x1000, "test.out", long:1)
```

2.4.20 /FN0020/ Reset

This function provides the reset functionality for the target. Flags: Flags is a bitmap and determines the type of reset.

```
long int Reset(long int Flags)
```

Flags

PUC 0x01

RST_NMI 0x02

WITH_RELEASE 0x04

Reset | PUC means that the JTAG sends the command to the MSP430. Reset | RST_NMI performs a reset through the $\overline{\text{RST}}$/NMI pin of the MSP430 MCU. The JTAG is also reset.

If the WITH_RELEASE option is selected, the device is released from the JTAG access after the reset.

Example

```
lFuncReturn = Reset(long:5)
```

2.4.21 /FN0022/ SetNotificationWnd

SetNotificationWnd() enables the status notification of a window. Each time a notification of the status window is necessary, the DLL sends a IMessageID message to the hWnd window. The execution status of an operation is passed in the WParam of this message. Completion status of the current operation is passed (0..100) in the LParam.

```
LONG SetNotificationWnd(LONG hWnd, LONG IMessageID)
```

hWnd

A window handle

ImessageID

A message identifier

Table 4 lists the status codes.

Table 4. SetNotificationWnd Status Codes

Status	WParam	Current Operation
STATUS_CONNECTSPA	1	Connecting to SPA430
STATUS_CONNECTTARGET	3	Connecting to target
STATUS_RELEASETARGET	5	Releasing target
STATUS_RELEASESPA	7	Releasing SPA430
STATUS_RESETTARGET	9	Resetting target
STATUS_ERASE	11	Erasing target
STATUS_ERASECHECK	13	Erase checking target
STATUS_PATTERNCHECK	15	Pattern checking target
STATUS_VERIFY	17	Verifying target
STATUS_PROGRAM	19	Programming target
STATUS_READOUT	21	Reading target out
STATUS_BLOWFUSE	23	Blowing fuse

2.4.22 /FN0023/ GetDeviceCfgInfo

```
long int GetDeviceCfgInfo(long int InfoCmd, long int InfoIdx, void* lpBuf)
```

InfoType

DEVICE_COUNT (0x01): GetDeviceCfgInfo returns number of devices in Device.cfg; InfoIdx and lpBuf are ignored.

SELECT_DEVICE (0x02): Selects the given device for further commands (device number in InfoIndex, first device is number 0; lpBuf is ignored).

DEVICE_NAME (0x03): Fills the name of the selected device into lpBuf; InfoIdx is ignored.

DEVICE_ID (0x04): Fills the DeviceID into lpBuf, InfoIdx is ignored.

DEVICE_DEFAULTOPTIONS (0x05): Fills the default options into lpBuf, InfoIdx is ignored.

DEVICE_MEMDEF_COUNT (0x06): GetDeviceCfgInfo() returns the number of memory definitions for selected device; lpBuf and InfoIdx are ignored.

DEVICE_MEMDEF (0x07): Fills the definition of a memory definition (index passed by InfoIdx) into lpBuf.

DEVICE_MEMPROTECT_COUNT (0x08): GetDeviceCfgInfo() returns the number of memory-protection definitions for the selected device; lpBuf and InfoIdx are ignored.

DEVICE_MEMPROTECT (0x09): Fills the definition of a memory protection definition (index passed by InfoIdx) into lpBuf.

DEVICE_VCC (0x0A): GetDeviceCfgInfo() returns the Vcc setting for selected device in mV; lpBuf and InfoIdx are ignored.

DEVICE_VPP (0x0B): GetDeviceCfgInfo() returns the Vpp setting for selected device in mV; lpBuf and InfoIdx are ignored.

DEVICE_VFUSE (0x0C): GetDeviceCfgInfo() returns the blow-fuse setting for the selected device; lpBuf and InfoIdx are ignored.

2.4.23 /FN0024/ AccessSFR

This function writes or reads data into the special function registers of the MSP430 MCU. (implemented in PRGS320.dll versions 1.05 and higher).

```
long int AccessSFR(LONG wAddr, void *IpData, LONG iFlags);
```

wAddr

SFR address that should be accessed. Allowed values are 0x0000 to 0x1FE (see the memory map of the corresponding device)

IpData

Pointer to the data to be written, or buffer which should receive the read data

Flags

The bits in Flags control the operation of AccessSFR()

iFlags

SFR_READ (0x00)

SFR_Write 0x01)

Example

```
IFuncReturn = AccessSFR(long:0x0020, &IpData, 0)
```

2.4.24 Return Values and Error Codes From PRGS430.DLL

Table 5 lists the return values and error codes from PRGS430.DLL.

Table 5. Return Values and Error Codes

Status	Return Value	Comment
OK	0	
SUCCESS	−1	Operation OK
ERR_COMMUNICATION	−2	Communication error (SSP)
ERR_TARGET_NOT_CONNECTED	−3	No target connected
ERR_SPA430_NOT_CONNECTED	−4	No SPA430 connected
ERR_WRONG_JTAG_VERSION	−5	JTAG version above 3
ERR_PUC_FAILED	−6	PUC did not succeed
ERR_SPA430_SYNC_FAILED	−7	Could not sync SPA430
ERR_NO_SPA430	−8	Adapter is not SPA430
ERR_UNKNOWN_TARGET	−9	Target type unknown
ERR_WRONG_TARGET	−10	Target type does not match
ERR_NO_TARGET_SELECTED	−11	No target selected (missing SetDeviceType() call)
ERR_TARGET_FUSE_BLOWN	−12	No target access because of blown fuse
ERR_BLOW_FUSE_FAILED	−13	Blown-fuse command failed
ERR_VCC_NOT_SET	−14	No VCC selected (missing SetVolt() call)
ERR_WRONG_VCC	−15	VCC out of allowed range
ERR_WRONG_BAUDRATE	−16	Invalid baud rate
ERR_COMPORT	−17	Error accessing the communications port
ERR_DEVICE_CFG	−18	Device.cfg corrupted
ERR_GENERAL	−19	General error (should not occur!)
ERR_RANGE	−20	Wrong range specified
ERR_ARGUMENT	−21	Wrong argument
ERR_FILE_IO	−22	Error during file I/O
ERR_FILE_DETECT	−23	File type could not be detected
ERR_FILE_END	−24	Unexpected end of file
ERR_PROJECT_INI	−25	Error reading {project}.ini
ERR_VCC_BELOW_VCCMINPROG	−26	VCC to low for selected function
ERR_FUSE_NOT_RELEASED	−27	Fuse not release for this device
STATUS_CONNECTSPA	1	Connecting to SPA430
STATUS_CONNECTTARGET	3	Connecting to target
STATUS_RELEASETARGET	5	Releasing target
STATUS_RELEASESPA	7	Releasing SPA430
STATUS_RESETTARGET	9	Resetting target
STATUS_ERASE	11	Erasing target
STATUS_ERASECHECK	13	Erase checking target
STATUS_PATTERNCHECK	15	Pattern checking target
STATUS_VERIFY	17	Verifying target
STATUS_PROGRAM	19	Programming target
STATUS_READOUT	21	Reading out target
STATUS_BLOWFUSE	23	Blowing fuse
ERR_READOUT_LOCKED	−28	Read prohibited

3 Hardware

This section describes the hardware for the MSP430 family of microcontrollers, including specifications, components of the programming adapters, and connection of the programming adapter to the MSP430 MCU families.

3.1 Specifications

Table 6 lists the specifications for the MSP430 hardware.

Table 6. MSP430 Hardware Specifications

Specification	Value
Temperature Range	10°C to 45°C
Humidity	40% to 70%
Power supply	14 V to 20 V, 200 mA minimum
Dimensions	150 mm (W) × 30 mm (H) × 95 mm (D)

3.2 Hints

These hints are useful for programming MSP430 MCUs or MSP430 MCUs on printed-wire boards (PWB).

- All VCC pins of an MSP430 MCU are tied together and connected to the most positive terminal of the supply.
- All VSS pins of an MSP430 MCU are tied together and connected to the most negative terminal of the supply.
- The interface should supply the MSP430 with proper conditions according to the device data sheet, in terms of current, voltage levels, and timing conditions.
- Make sure the proper signal connections (see Section 3.3) are made.
- Short cables to interconnect the interface to the MSP430 MCU or PWB; less than 20 cm is recommended.
- Ensure low-impedance interconnections, especially for the path of the programming and fuse blow voltage.
- When a device with a transparent window (MSP430E3xx family) is programmed, the window should be already covered with an opaque label while the device is programmed. Because ambient light contains the correct wavelength for erasure, keep the transparent window covered after the device is programmed.

3.3 Programming Adapter Target Connector Signals

The target connector signals for the programming adapter ensure communication between the programming adapter and MSP430 MCUs and supply low energy to systems without extra supply sources.

Figure 9 and Figure 10 show the target connector signals for the programming adapter.

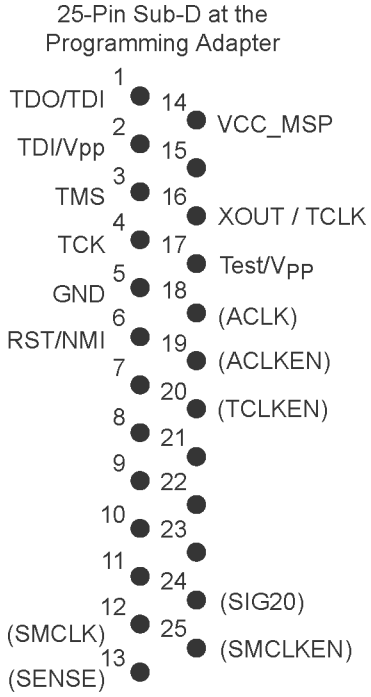


Figure 9. 25-Pin Sub-D at Programming Adapter

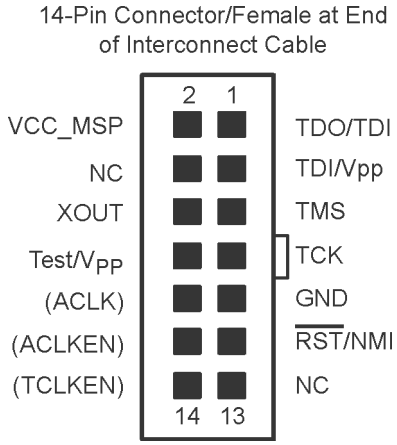


Figure 10. 14-Pin Connector at End of Interconnect Cable

Table 7 lists the target connector signals and describes their requirement statuses and functions.

Table 7. Target Connector Signal Functions

Signal	Required	Comment
TMS	Mandatory	Test mode select functions according to IEEE1149.1
TCK	Mandatory	Test clock functions according to IEEE1149.1
TDI/VPP	Mandatory	Test data input functions according to IEEE1149.1, but with additional programming voltage for 3xx devices.
TDO/TDI	Mandatory	Test data output functions according to IEEE1149.1, but additional data input is used when programming voltage is applied by TDI/VPP.
GND	Mandatory	GND is the most negative terminal.
VCC_MSP	Mandatory (if internal supply voltage is used)	Voltage source is used with MSP430 MCUs or PWBs. The voltage level is set by software.
XOUT	Mandatory	Signal supplies the MSP430 system with clock signals.
$\overline{\text{RST}}$/NMI	Optional	If not connected, $\overline{\text{RST}}$/NMI must be held high.
Test/VPP	Mandatory (depending on device)	Signal used to select pin or JTAG function or to apply VPP

The output signal levels of the programming adapter are near GND or VCC_MSP.

- The $\overline{\text{RST}}$/NMI terminal of the device must be high; otherwise, the access to the device through JTAG system may fail.
- The programming procedure (handling of the SW) is described in Chapters 1 and 2 of this manual.
- The connections from the MSP430 terminals must follow EMI rules, such as short lines and ground planes. If TMS line receives one negative pulse by EMI strike, the fuse current is activated (with fuse version 1.0). The fuse current flows from TDI(/VPP) pin to GND (or VSS).

Table 8. Programming Adapter Signal Levels

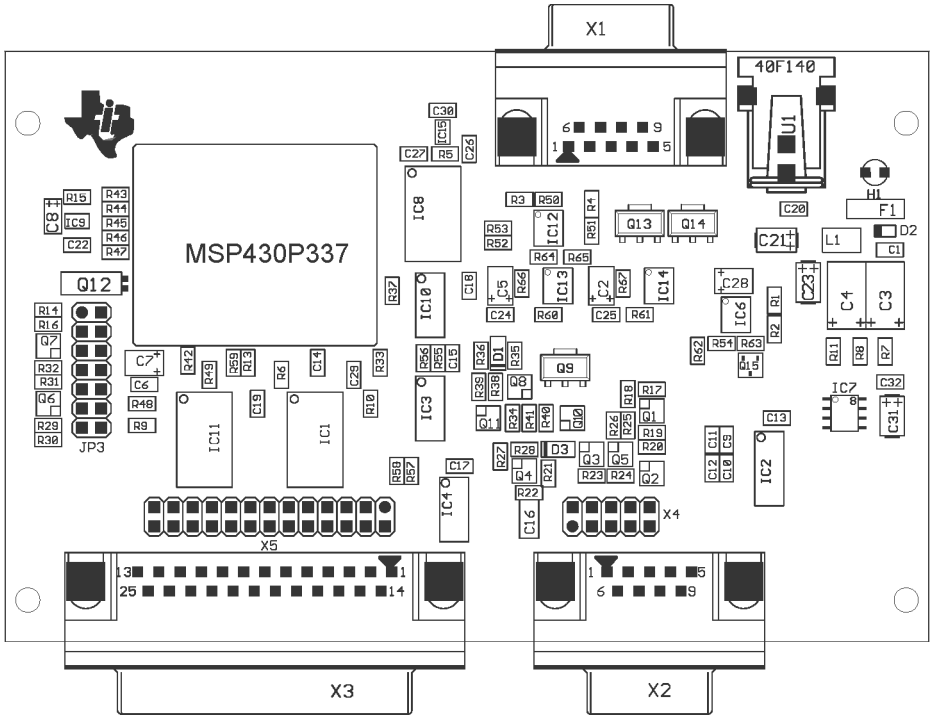
Signal	Signal Level
TMS	VSS or VCC_MSP
TCK	VSS or VCC_MSP
TDI/VPP	VSS or VCC_MSP or VPP
TDO/TDI	VSS or VCC_MSP
XOUT	VSS or VCC_MSP
$\overline{\text{RST}}$/NMI	VSS or VCC_MSP
Test/VPP	VSS or VCC_MSP or VPP

3.4 MSP-PRGS430 Circuit Diagrams

See Section 5 for the MSP-PRGS430 circuit diagrams.

3.5 Location of Components on MSP-PRGS430

Figure 11 shows the locations of the components.



NOTE: Do not use J2 pin 9 as $\overline{\text{RST}}$/NMI pullup.

Figure 11. MSP-PRGS430 Components

3.6 Interconnection of MSP-PRGS430 to OTP or EPROM-Based MSP430 Devices

The circuit diagram in Figure 12 shows the connections required to program OTP (MSP430Pxxx) and EPROM (MSP430Exxx) based MSP430 MCUs with the MSP-PRGS430 programming adapter. Consult the device data sheet for the specific device location of the supply and JTAG pins. Ensure that all positive and negative supply pins are connected together.

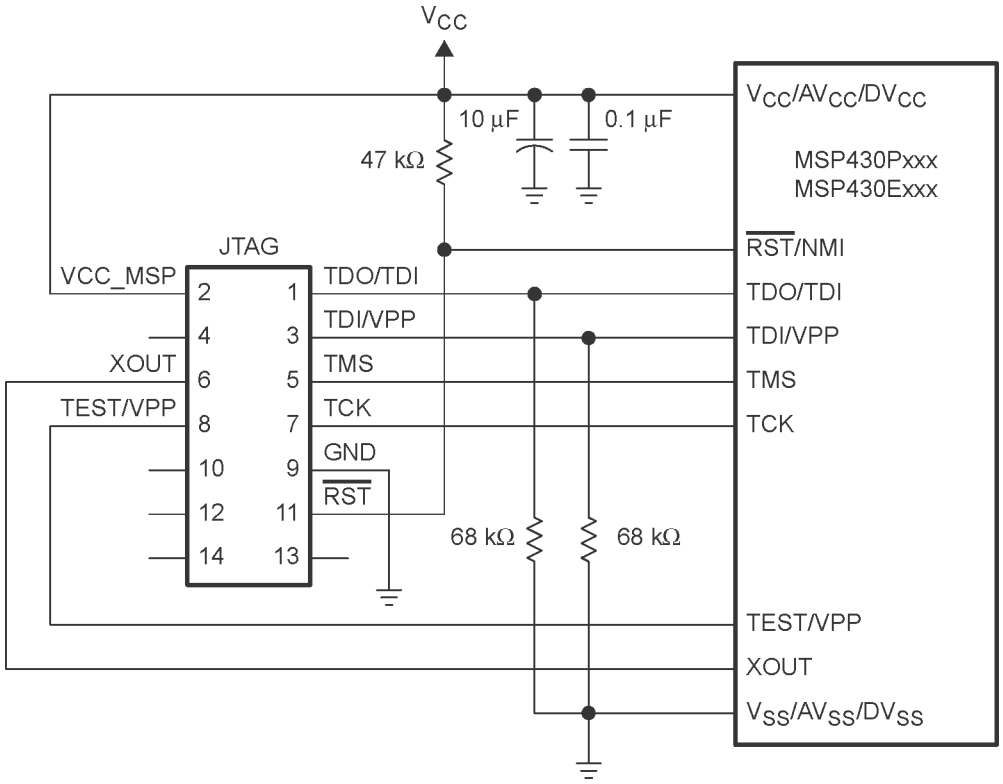


Figure 12. MSP-PRGS430 Used to Program OTP or EPROM-Based MSP430 Devices

The $\overline{\text{RST}}$/NMI terminal on the MSP430 MCU has to be held high by an external resistor during access of the device through JTAG. In a noisy environment, consider using an additional capacitor from $\overline{\text{RST}}$/NMI to VSS.

NOTE: The example schematic shows a system where the target voltage is supplied by the MSP-PRGS430. For in-system programming with an external supply voltage, do not connect pin 2 of the JTAG connector. In this case, the supply voltage setting in the PRGS430 must be adjusted to the external supply voltage level. The TEST/VPP connection is only required on lower pin-count devices with multiplexed JTAG pins.

3.7 Interconnection of MSP-PRGS430 to Flash-Based MSP430 Devices

The circuit diagram in Figure 13 shows the connections required to program flash-based MSP430 MCUs (MSP430Fxxx) with the MSP-PRGS430 programming adapter. Consult the device data sheet for the specific device location of the power supply and JTAG pins. Ensure that all positive and negative power supply pins are connected together.

The signal TEST/VPP is only required on lower pin-count devices with multiplexed JTAG pins. In this case, special attention must be given to the circuit design around the four JTAG pins (TDO/TDI, TDI, TMS, and TCK), because they are shared between the applications hardware and the JTAG interface used by programming adapter.

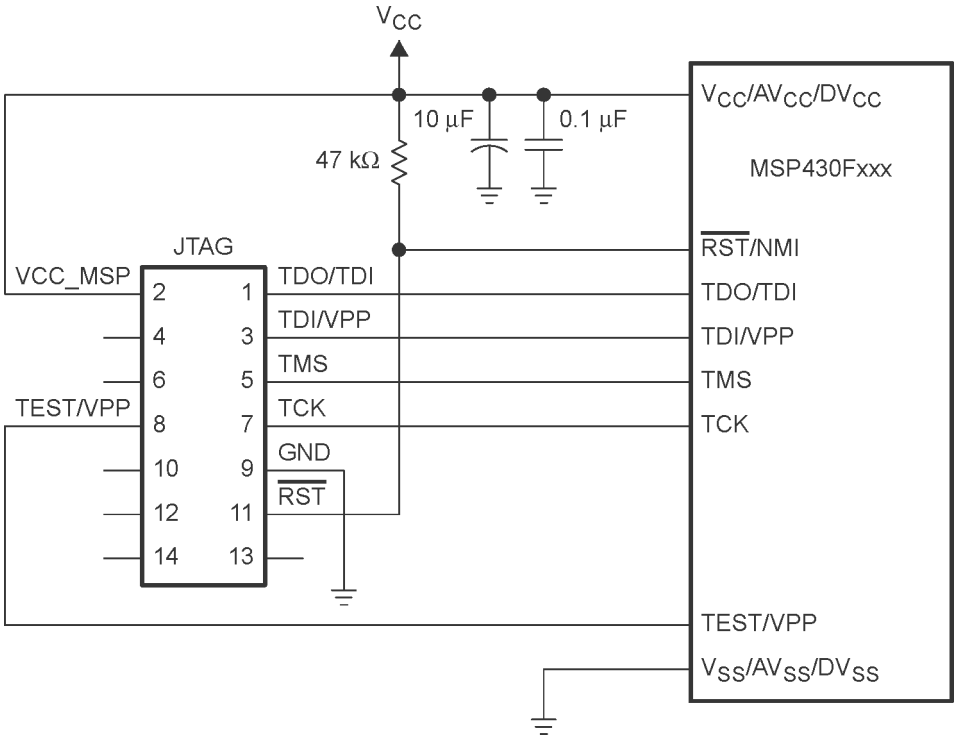


Figure 13. MSP-PRGS430 Used to Program Flash-Based MSP430 Devices

NOTE: The example schematic shows a system where the target voltage is supplied by the MSP-PRGS430. For in-system programming with an external supply voltage, do not connect pin 2 of the JTAG connector. In this case, the PRGS430 supply voltage setting must be adjusted to the external supply voltage level. The TEST/VPP connection is only required on lower pin-count devices with multiplexed JTAG pins.

4 Hex Object Format

This section describes the hex object format.

4.1 Intel-Hex Object Format

The Intel-hex object format supports 16-bit addresses and consists of a nine-character (four field) prefix that defines the start of record, byte count, load address, record type, and a two character sumcheck suffix.

The two record types, which are represented in the nine-character prefix, are described below:

00 = Data record (begins with the colon start character)

01 = End-of-file record

Record type 00, the data record, begins with the colon (:) start character and is followed by the byte count, the address of the first data byte, the record type (00), and the sumcheck. The sumcheck is the 2s complement (in binary) of the preceding bytes in the record, including the byte count, address, and data bytes.

Record type 01, the end-of-file record, also begins with the colon (:) start character. The colon is followed by the byte count, address, record type (01), and sumcheck.

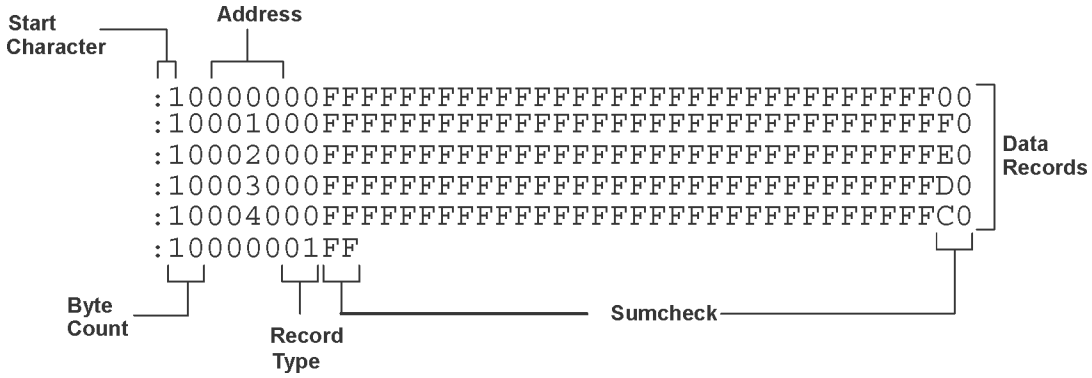


Figure 14. Intel-Hex Object Format

4.2 TI-TXT File Format

The TI-TXT file format used by the tool is shown as follows:

```
@ADDR1
DATA01  DATA02 ........ DATA16
DATA17  DATA32 ........ DATA32
           ........
DATAm ........DATAn
@ADDR2
DATA01 ..................... DATAn
q
```

Where

@ADDR is the start address of a section (hexadecimal)
DATAn represents a data byte (hexadecimal)
q is the termination of the file

Example

```
@F000
31 40 00 03 B2 40 80 5A 20 01 D2 D3 22 00 D2 E3
21 00 3F 40 E8 FD 1F 83 FE 23 F9 3F
@FFFE
00 F0
Q
```

Restrictions:

- The number of sections is unlimited.
- The start address must be even.
- Each line must have 16 data bytes, except the last line of a section.
- Data bytes are separated by a single space.
- The termination tag q indicates end-of-file is mandatory.

5 Schematics

This following figures show the schematic diagrams for the serial programming adapter.

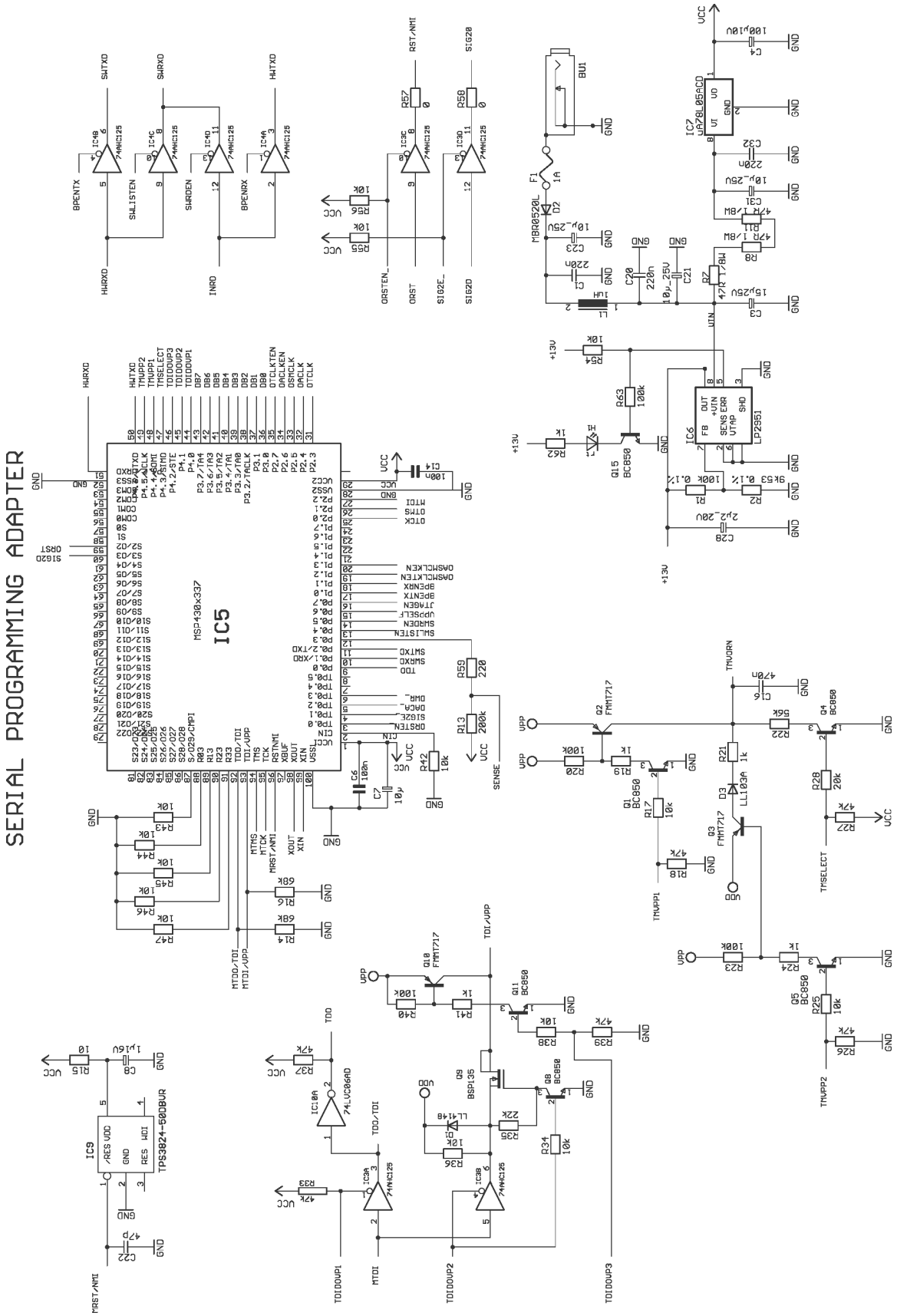


Figure 15. Schematics (1 of 2)

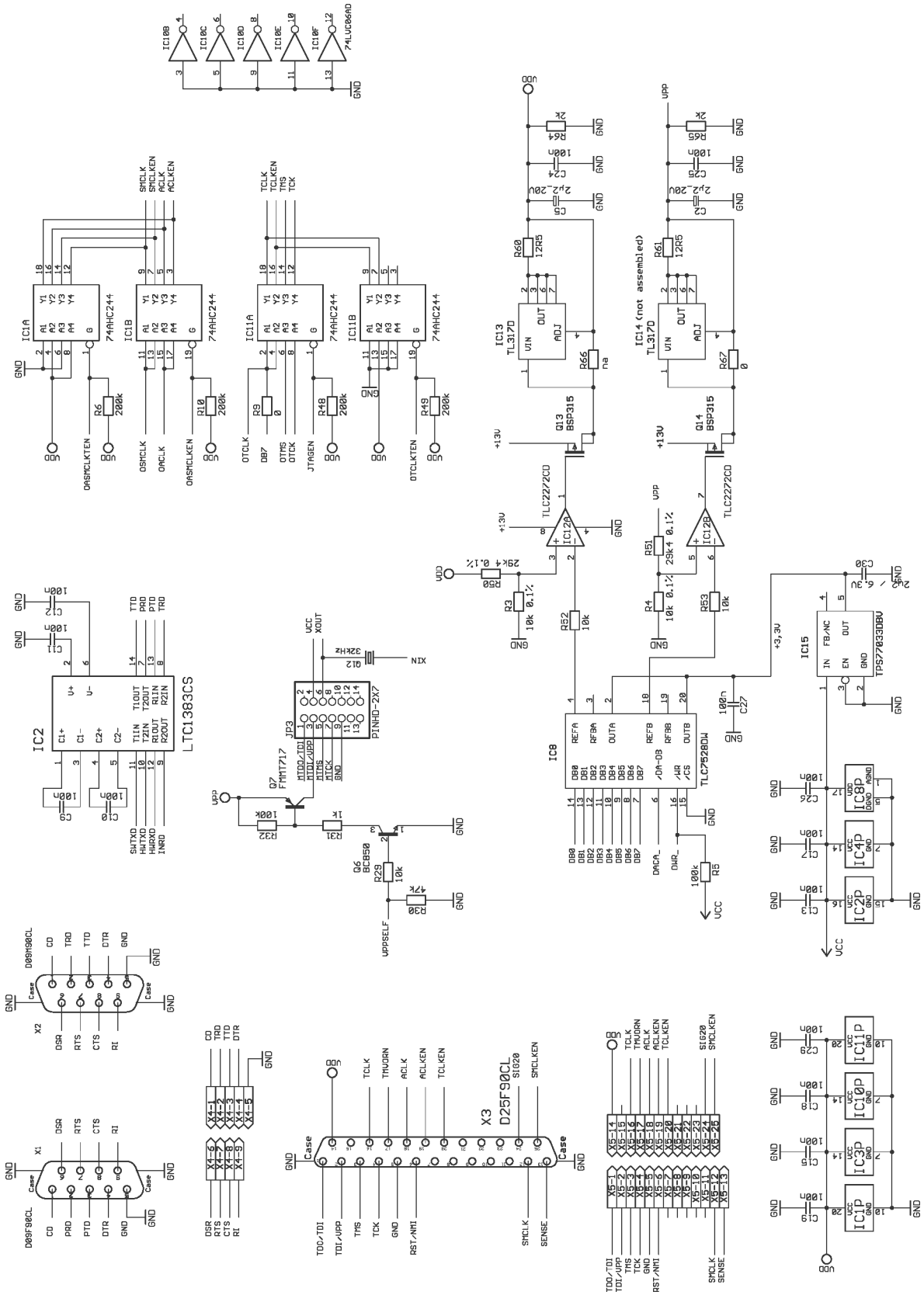


Figure 16. Schematics (2 of 2)

Revision History

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

Changes from July 27, 2009 to October 8, 2018	Page
• Formatting and editorial changes throughout document	1

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

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