

**ABSTRACT**

This document describes the functions that are required to erase, program, and verify the memory module of the MSP430™ flash-based and FRAM-based microcontroller families using the JTAG communication port.

Table of Contents

1 Introduction	3
1.1 About This Document	3
1.2 Organization of This Document	3
2 Programming Using the JTAG Interface	4
2.1 Introduction	4
2.1.1 MSP430 JTAG Restrictions (Noncompliance With IEEE Std 1149.1)	4
2.1.2 TAP Controller State Machine	4
2.2 Interface and Instructions	4
2.2.1 JTAG Interface Signals	5
2.2.2 JTAG Access Macros	6
2.2.3 Spy-Bi-Wire (SBW) Timing and Control	9
2.2.4 JTAG Communication Instructions	14
2.3 Memory Programming Control Sequences	20
2.3.1 Start-Up	20
2.3.2 General Device (CPU) Control Functions	23
2.3.3 Accessing Non-Flash Memory Locations With JTAG	33
2.3.4 Programming the Flash Memory (Using the Onboard Flash Controller)	37
2.3.5 Erasing the Flash Memory (Using the Onboard Flash Controller)	42
2.3.6 Reading From Flash Memory	45
2.3.7 Verifying the Target Memory	45
2.3.8 FRAM Memory Technology	48
2.4 JTAG Access Protection	48
2.4.1 Burning the JTAG Fuse - Function Reference for 1xx, 2xx, 4xx Families	50
2.4.2 Programming the JTAG Lock Key - Function Reference for 5xx, 6xx, and FRxx Families	52
2.4.3 Testing for a Successfully Protected Device	53
2.4.4 Unlocking an FRAM Device in Protected and Secured Modes	54
2.4.5 Memory Protection Unit Handling	54
2.4.6 Intellectual Property Encapsulation (IPE)	55
2.4.7 FRAM Write Protection	55
2.5 JTAG Function Prototypes	56
2.5.1 Low-Level JTAG Functions	56
2.5.2 High-Level JTAG Routines	58
2.6 JTAG Features Across Device Families	63
2.7 References	73
3 JTAG Programming Hardware and Software Implementation	74
3.1 Implementation History	74
3.2 Implementation Overview	74
3.3 Software Operation	74
3.4 Software Structure	75
3.4.1 Programmer Firmware	75
3.4.2 Target Code	76
3.5 Hardware Setup	77
3.5.1 Host Controller	78
3.5.2 Target Connection	78
3.5.3 Host Controller or Programmer Power Supply	79

3.5.4 Third-Party Support.....	80
4 Errata and Revision Information.....	81
4.1 Known Issues.....	81
4.2 Revisions and Errata From Previous Documents.....	81
5 Revision History.....	82

Trademarks

MSP430™ and Code Composer Studio™ are trademarks of Texas Instruments.

IAR Embedded Workbench® is a registered trademark of IAR Systems.

All trademarks are the property of their respective owners.

1 Introduction

1.1 About This Document

This document describes the functions that are required to erase, program, and verify the memory module of the MSP430 flash-based and FRAM-based microcontroller families using the JTAG communication port. In addition, it describes how to program the JTAG access security fuse that is available on all MSP430 devices. This document describes device access using both the standard 4-wire JTAG interface and the 2-wire JTAG interface, which is also referred to as Spy-Bi-Wire (SBW).

An example programmer system, which includes software (source code is provided) and the corresponding hardware, is described in . This example is intended as a reference for understanding the concepts presented in this report and to aid in development of similar MSP430 programmer solutions. In that sense, it is not meant to be a fully featured programming tool but, rather, is intended as a construction manual for such a tool. Those users who are looking for a ready-to-use tool should see the complete programming tool solution from TI called [MSP-GANG Production Programmer](#).

An example programmer system, including source code and the corresponding hardware, is described in [Chapter 3](#). This example is intended as a reference for understanding the concepts presented in this report and to aid in development of similar MSP430 programmer solutions. In that sense, it is not meant to be a fully featured programming tool but, rather, is intended as a basic guide to construct such a tool.

Users looking for ready-to-use tools can refer to:

- [MSP-GANG Production Programmer](#): device programmer that can program up to 8 MSP430 devices at the same time
- [MSP Debug Stack](#): host static library and embedded firmware running on [MSP debuggers](#)
- [MSP BSL](#): software and hardware to communicate with MSP430 bootloaders during prototyping phase, final production, and in service

1.2 Organization of This Document

Four main topics are presented in this document:

[Section 2.2, *Interface and Instructions*](#), describes the required JTAG signals and associated pin functionality for programming the MSP430 family. In addition, this section includes the descriptions of the provided software macro routines and JTAG instructions used to communicate with and control a target MSP430 through the JTAG interface.

[Section 2.3, *Memory Programming Control Sequences*](#), describes how to use the provided macros and function prototypes in a software-flow format to control a target MSP430 device and program or erase the memory.

[Section 2.4, *JTAG Access Protection*](#), describes the mechanism that can disable memory access through JTAG to the memory of the target device, which can eliminate undesired memory access for security purposes.

[Chapter 3, *JTAG Programming Hardware and Software Implementation*](#), describes how to develop an example MSP430 flash programmer using an MSP430F5437 as the host controller. This chapter includes a schematic and the required software and project files. A thorough description of how to use the given implementation is also included, to provide an example system that can be referenced for custom MSP430 programmer solutions.

2 Programming Using the JTAG Interface

2.1 Introduction

This document provides an overview of how to program the memory module of an MSP430 flash-based or FRAM-based device using the on-chip JTAG interface [4-wire or 2-wire Spy-Bi-Wire (SBW) interface]. A focus is maintained on the high-level JTAG functions used to access and program the memory and the required timing.

2.1.1 MSP430 JTAG Restrictions (Noncompliance With IEEE Std 1149.1)

- The JTAG pins are shared with port functions on all devices with a TEST pin. This includes the 5xx, 6xx, and FRxx families as well as certain device groups from the 2xx and 4xx families (see [Table 2-14](#)). On these devices, a special entry sequence must be sent to enable 4-wire JTAG connection. This sequence is described in [Section 2.3.1.1](#).
- The MSP430 device must be the first device in the JTAG chain (because of clocking on TDI and JTAG fuse check sequence).
- Only the BYPASS instruction is supported. There is no support for SAMPLE, PRELOAD, or EXTEST instructions.
- The REP430 Software Implementation described within this document should not be used along with real time operating systems (RTOS) or an interrupt system, as this could interrupt the programming using JTAG or SBW, in particular the IR/DR shifts and the TCLK clock generation.

2.1.2 TAP Controller State Machine

The MSP430 JTAG interface implements the test access port state machine (TAP controller) as specified by IEEE Std 1149.1. References to the TAP controller and specific JTAG states identified in IEEE Std 1149.1 are made throughout this document. [Figure 2-1](#) shows the TAP state machine.

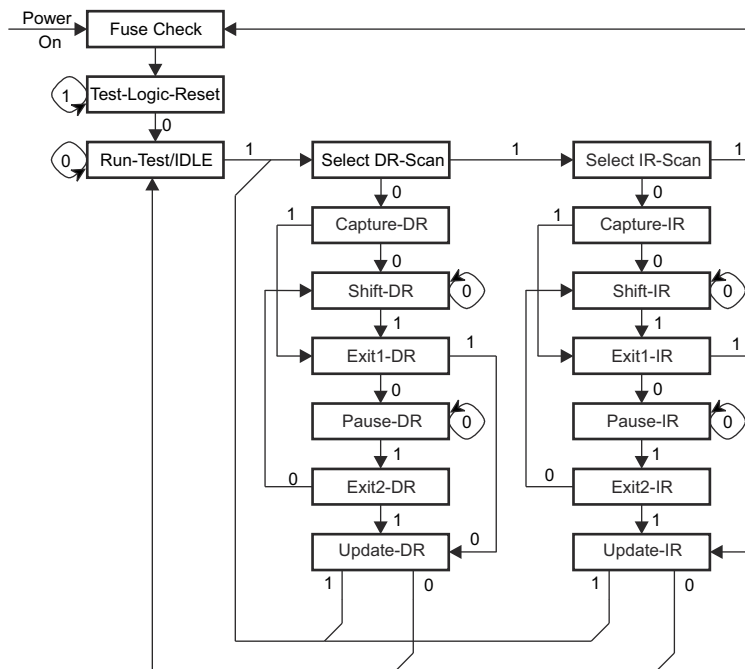


Figure 2-1. TAP Controller State Machine

2.2 Interface and Instructions

This section describes the hardware connections to the JTAG interface of the MSP430 devices and the associated pin functions that are used during programming. This section also describes the software macro routines that are used to program a MSP430 target and the JTAG instructions that are used to communicate with and control the target through the JTAG interface.

The Replicator example project detailed in [Chapter 3](#) implements the following high-level flow to clarify the minimum requirements for programming an MSP430 target memory.

- Enable JTAG Access (see [Section 2.3.1.1](#))
- Erase Target Memory (see [Section 2.3.5](#) and [Section 2.3.8.2](#))
- Verify Target Memory (special case: Erase Check) (see [Section 2.3.7](#))
- Write Target Memory (see [Section 2.3.4](#) and [Section 2.3.8.1](#))
- Verify Target Memory (see [Section 2.3.7](#))
- Release Target from JTAG Control (see [Section 2.3.2.1.6](#))

2.2.1 JTAG Interface Signals

The MSP430 family supports in-circuit programming of flash and FRAM memory through the JTAG port, which is available on all MSP430 devices. All devices support the JTAG 4-wire interface. Some devices also support the next-generation optimized 2-wire JTAG (Spy-Bi-Wire) interface. Using these protocols, an interface connection that can access the MSP430 JTAG port using a PC or other controller can be established. See the section *Signal Connections for In-System Programming and Debugging, MSP-FET430PIF, MSP-FET430UIF, GANG430, PRGS430* in the [MSP430 Hardware Tools User's Guide](#).

2.2.1.1 Pros and Cons of 2-Wire Spy-Bi-Wire and 4-Wire JTAG

Table 2-1. Pros and Cons of 2-Wire Spy-Bi-Wire

Pros	Cons
Only two pins are used (TEST and RST)	Slower than 4-wire JTAG
No overlap between JTAG and IO pins	

Table 2-2. Pros and Cons of 4-Wire JTAG

Pros	Cons
Faster than 2-wire Spy-Bi-Wire	Four GPIO pins are used
	When using JTAG, other pin functions not usable

2.2.1.2 4-Wire JTAG Interface

The standard JTAG interface requires four signals for sending and receiving data. On larger MSP430 devices, these pins are dedicated for JTAG. Smaller devices with fewer total pins multiplex these JTAG lines with general-purpose functions. On these smaller devices, one additional signal is required that is used to define the state of the shared pins. This signal is applied to the TEST pin. The remaining connections required are ground and V_{CC} when powered by the programmer. These signals are described in [Table 2-3](#).

Table 2-3. Standard 4-Wire JTAG Signals

Pin	Direction	Description
TMS	IN	Signal to control the JTAG state machine
TCK	IN	JTAG clock input
TDI	IN	JTAG data input and TCLK input
TDO	OUT	JTAG data output
TEST	IN	Enable JTAG pins (shared JTAG devices only)

The TEST input exists only on MSP430 devices with shared JTAG function, usually assigned to port 1. For normal operation (non-JTAG mode), this pin is internally pulled down to ground, which enables the shared pins as standard port I/O. To enable these pins for JTAG communication, refer to [Section 2.3.1.1](#).

The TCLK signal is an input clock that must be provided to the target device from an external source. This clock is used internally as the system clock of the target device, MCLK, to load data into memory locations and to clock the CPU. There is no dedicated pin for TCLK; instead, the TDI pin is used as the TCLK input. This occurs while the MSP430 TAP controller is in the Run-Test/Idle state.

Note

TCLK input support on the MSP430 XOUT pin exists but has been superseded by the TDI pin on all current MSP430 flash-based and FRAM-based devices. Existing FET tools, as well as the software provided with this document, implement TCLK on the TDI input pin.

2.2.1.3 2-Wire Spy-Bi-Wire (SBW) JTAG Interface

The core JTAG logic integrated into devices that support 2-wire mode is identical to 4-wire-only devices. The fundamental difference is that 2-wire devices implement additional logic that is used to convert the 2-wire communication into the standard 4-wire communication internally. In this way, the existing JTAG emulation methodology of the MSP430 can be fully used.

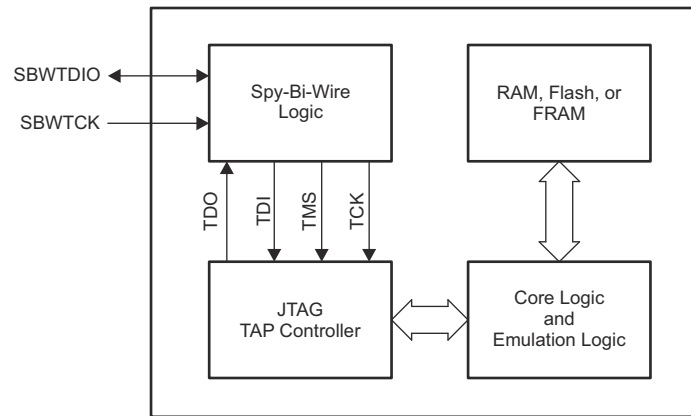


Figure 2-2. Spy-Bi-Wire Basic Concept

The 2-wire interface is made up of the SBWTCK (Spy-Bi-Wire test clock) and SBWTDIO (Spy-Bi-Wire test data input/output) pins. The SBWTCK signal is the clock signal and is a dedicated pin. In normal operation, this pin is internally pulled to ground. The SBWTDIO signal represents the data and is a bidirectional connection. To reduce the overhead of the 2-wire interface, the SBWTDIO line is shared with the $\overline{\text{RST}}/\text{NMI}$ pin of the device.

[Section 2.6](#) describes the MSP430 devices and their respective JTAG interface implementation.

2.2.2 JTAG Access Macros

To keep descriptions of the JTAG functions in the following sections simple, high-level macros have been used to describe the JTAG access. This document does not detail the basic JTAG functionality; rather, it focuses on the MSP430-specific implementation used for memory access and programming. For the purpose of this document, it is important to show the instructions that must be loaded into the JTAG instruction register, as well as when these instructions are required. [Section 2.2.2.1](#) summarizes the macros used throughout this document and their associated functionality. See the accompanying software for more information.

Table 2-4. JTAG Communication Macros

Macro Name	Function
IR_SHIFT (8-bit instruction)	Shifts an 8-bit JTAG instruction into the JTAG instruction register. At the same time, the 8-bit value is shifted out through TDO.
DR_SHIFT16 (16-bit data)	Shifts a 16-bit data word into a JTAG data register. At the same time, the 16-bit value is shifted out through TDO.
DR_SHIFT20 (20-bit address)	Shifts a 20-bit address word into the JTAG Memory Address Bus register. At the same time, the 20-bit value is shifted out through TDO. Only applicable to MSP430X architecture devices.
MsDelay (time)	Waits for the specified time in milliseconds
SetTCLK	Sets TCLK to 1
ClrTCLK	Sets TCLK to 0
TDOvalue	Variable containing the last value shifted out on TDO

2.2.2.1 Macros for 4-Wire JTAG Interface

2.2.2.1.1 IR_SHIFT (8-Bit Instruction)

This macro loads a JTAG instruction into the JTAG instruction register (IR) of the target device. In the MSP430, this register is eight bits wide with the least significant bit (LSB) shifted in first. The data output from TDO during a write to the JTAG instruction register contains the version identifier of the JTAG interface (or JTAG ID) that is implemented on the target device. The JTAG ID is shifted out with MSB first.

Regardless of the 8-bit instruction sent out on TDI, the return value on TDO is always the JTAG ID. Each instruction bit is captured from TDI by the target MSP430 on the rising edge of TCK. TCLK should not change state while this macro is executed (TCLK = TDI while the TAP controller is in the Run-Test/Idle state). [Figure 2-3](#) shows how to load the ADDR_16BIT instruction into the JTAG IR. See [Section 2.2.4](#) for a complete list of the JTAG interface communication instructions that are used to access the flash memory module of the target device.

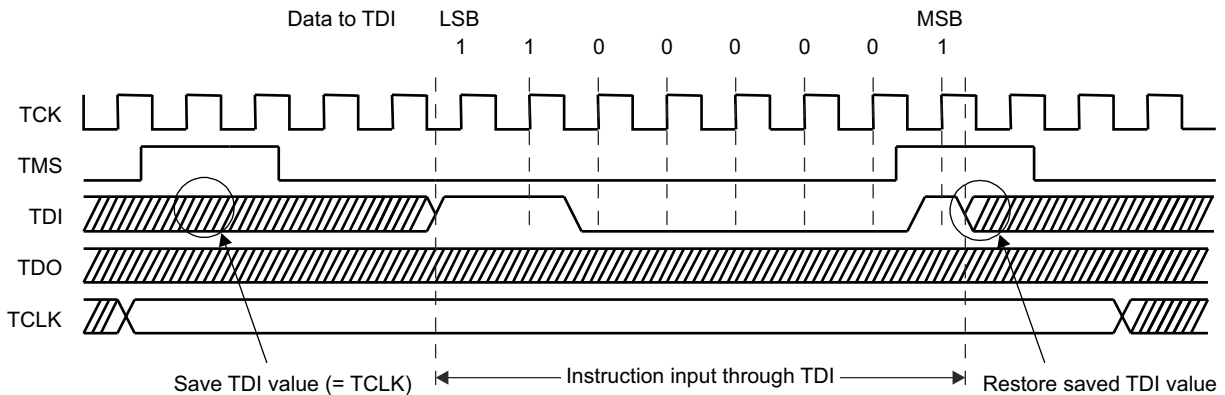


Figure 2-3. Timing Example for IR_SHIFT (0x83) Instruction

2.2.2.1.2 DR_SHIFT16 (16-Bit Data)

This macro loads a 16-bit word into the JTAG data register (DR) (in the MSP430 devices, a data register is 16 bits wide). The data word is shifted, most significant bit (MSB) first, into the TDI input of the target MSP430 device. Each bit is captured from TDI on a rising edge of TCK. At the same time, TDO shifts out the last captured and stored value in the addressed data register. A new bit is present at TDO with a falling edge of TCK. TCLK should not change state while this macro is executing. [Figure 2-4](#) shows how to load a 16-bit word into the JTAG DR and read out a stored value through TDO.

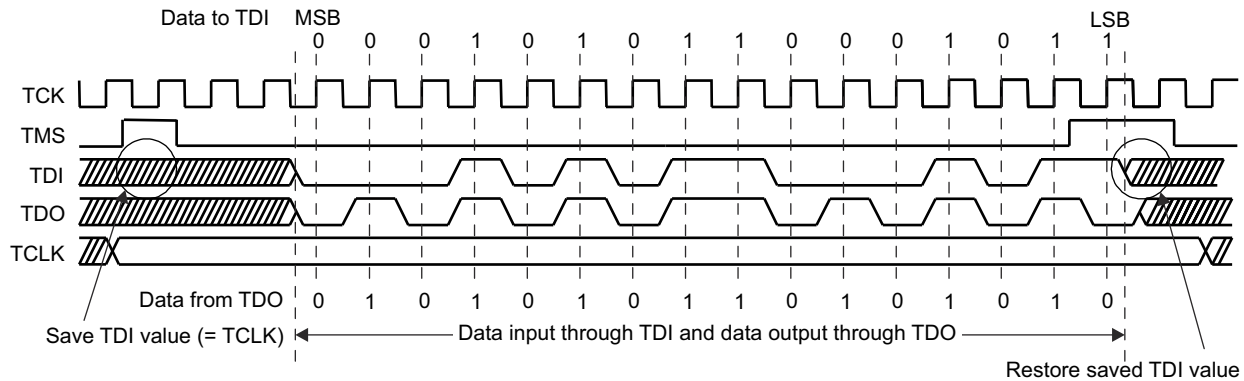


Figure 2-4. Data Register I/O: DR_SHIFT16 (0x158B) (TDO Output Is 0x55AA)

2.2.2.1.3 DR_SHIFT20 (20-Bit Address) (Applies Only to MSP430X Devices)

The MSP430X architecture is based on a 20-bit memory address bus (MAB), to address up to 1 MB of continuous memory. No new JTAG instructions are needed to control the 20-bit MAB (for details on instructions, see [Section 2.2.4.1](#)), only the JTAG address register itself has been extended to 20 bits. This macro loads a 20-bit address word into the 20-bit wide JTAG MAB register. The address word is shifted, MSB first, into the TDI input of the target MSP430 device. Each bit is captured from TDI on a rising edge of TCK. At the same time, TDO shifts out the last captured and stored value in the JTAG MAB register. A new bit is present at TDO with a falling edge of TCK. TCLK should not change state while this macro is executing. This macro should only be used when IR_ADDR_16BIT or IR_ADDR_CAPTURE has been loaded into the JTAG instruction register before the MAB is manipulated through JTAG. On a 20-bit shift access, the upper four bits (19:16) of the JTAG address register are shifted out last. Therefore, bit 15 of the MAB is read first when the lower part of the MAB is accessed by performing a 16-bit shift. This implementation ensures compatibility with the original MSP430 architecture and its JTAG MAB register implementation.

Note

The DR_SHIFT20 (20-bit Address) macro in the associated C-code software example application automatically reconstructs the swapped TDO (15:0) (19:16) output to a continuous 20-bit address word (19:0) and simply returns a 32-bit LONG value.

Figure 2-5 shows how to load a 20-bit address word into the JTAG address register and read out a stored value through TDO.

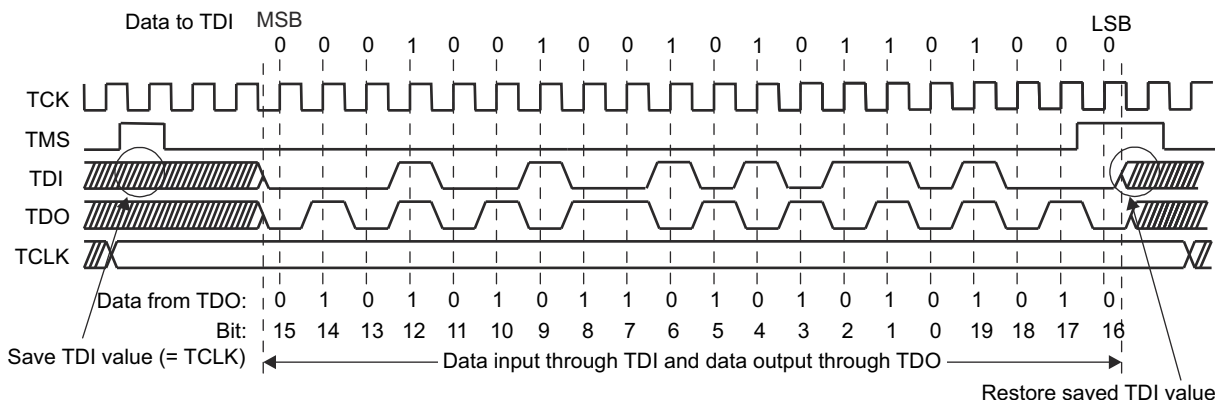


Figure 2-5. Address Register I/O: DR_SHIFT20 (0x12568) (TDO Output Is 0xA55AA)

2.2.2.1.4 MsDelay (Time)

This macro causes the programming interface software to wait for a specified amount of time in milliseconds (ms). While this macro is executing, all signals to and from the target MSP430 must hold their previous values.

2.2.2.1.5 SetTCLK

This macro sets the TCLK input clock (which is provided on the TDI signal input) high. TCK and TMS must hold their last value while this macro is performed (see Section 2.2.3.5.1 and Figure 2-11 for SBW-specific constraints).

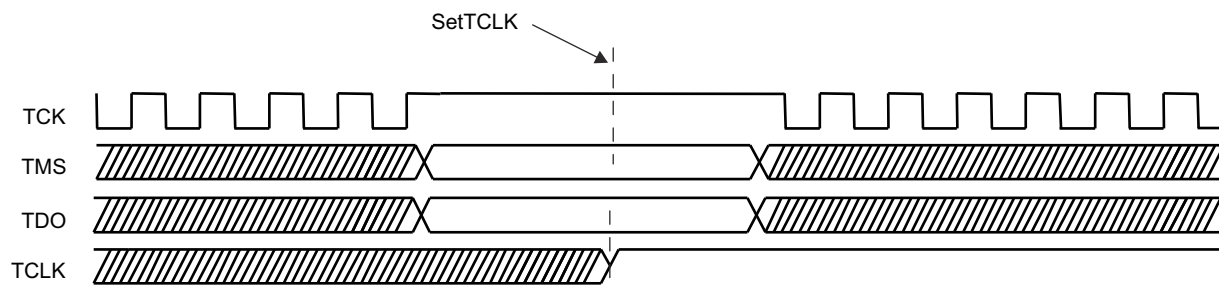


Figure 2-6. SetTCLK

2.2.2.1.6 ClrTCLK

This macro resets the TCLK input clock low. TCK and TMS must hold their last value while this action is performed (see Section 2.2.3.5.1 and Figure 2-11 for SBW-specific constraints).

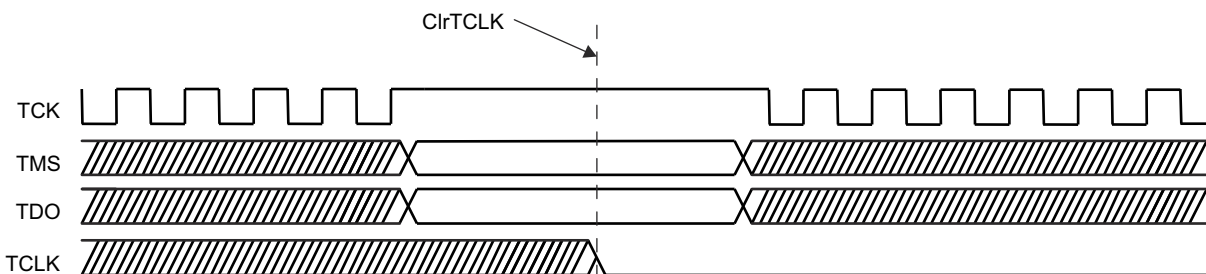


Figure 2-7. ClrTCLK

2.2.2.2 Macros for Spy-Bi-Wire (SBW) Interface

All JTAG macros described in [Section 2.2.2.1](#) also apply to the 2-wire interface and are provided as software source along with this document.

2.2.3 Spy-Bi-Wire (SBW) Timing and Control

The following sections described the fundamentals of the SBW implementation as it relates to supporting generation of the macro function timing signals. This is intended to enable development of custom MSP430 programming solutions, rather than just relying on the example application code also provided.

2.2.3.1 Basic Timing

The SBW interface serial communication uses time-division multiplexing and allocates three time slots: TMS_SLOT, TDI_SLOT, and TDO_SLOT. To clock TCLK through the SBW interface in a similar method as it is clocked through TDI during 4-wire JTAG access, an alternative JTAG timing method is implemented. This implementation makes use of the fact that the TDI and TMS signals are captured with the falling edge of SBWTCK in their respective slots as shown in [Figure 2-8](#).

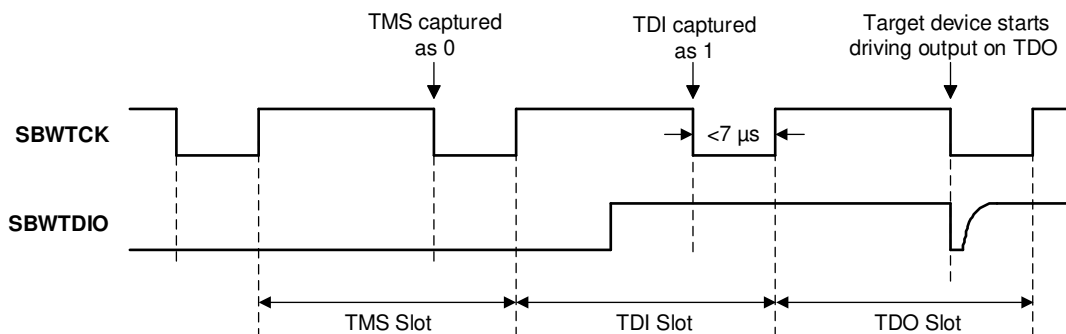


Figure 2-8. Spy-Bi-Wire Timing Diagram

The logic that translates between the 2-wire and 4-wire interfaces is shown in [Figure 2-9](#).

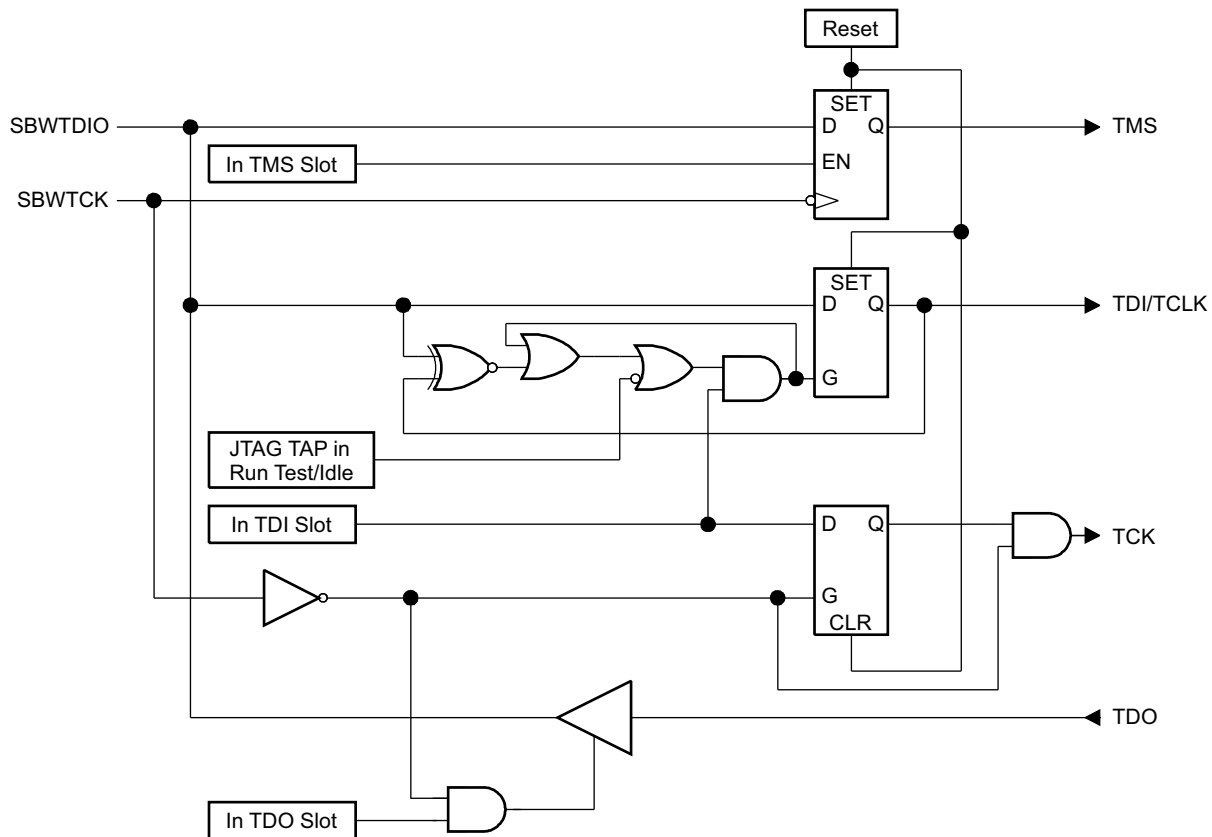


Figure 2-9. SBW-to-JTAG Interface Diagram

The advantages of this implementation are:

- Data on TDI and data on TDO are aligned.
- During the TDI_SLOT of the 2-wire interface, SBWTDIO can be used as TCLK input if the JTAG TAP controller is in its Run-Test/Idle state. For this purpose, the TDI output must be synchronized to its input as shown in [Figure 2-11](#). The synchronization logic is only active in the Run-Test/Idle state.

2.2.3.2 TMS Slot

The TMS Slot is used to switch between states in the TAP Controller state machine (see [Section 2.1.2](#)) of the JTAG module of the target device. The following macros are located in the LowLevelFunc header file in the Replicator example project.

2.2.3.2.1 TMSH Macro

Sets SBWTDIO high for TMS slot (no special TDI preparation handling)

- Set SBWTDIO high
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK low
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK high

Sets SBWTDIO low for TMS slot (no special TDI preparation handling)

- Set SBWTDIO low
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK low
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK high

2.2.3.2.3 TMSLDH Macro

Sets SBWTDIO low for TMS slot, but brings it back high after SBWTCK falling edge before TMS slot ends. Used for some cases of ClrTCLK and SetTCLK.

- Set SBWTDIO low
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK low
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTDIO high
- Set SBWTCK high

2.2.3.3 TDI Slot

In most TAP Controller states (such as Shift-IR), TDI is used for shifting data into the target device. In Run-Test/IDLE state, the TDI slot can also be used to clock the target CPU (see [Section 2.2.3.5](#)) The following macros are located in the LowLevelFunc header file in the Replicator example project.

2.2.3.3.1 TDIH Macro

Sets SBWTDIO high for TDI slot.

- Set SBWTDIO high
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK low
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK high

2.2.3.3.2 TDIL Macro

Sets SBWTDIO low for TDI slot.

- Set SBWTDIO low
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK low
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK high

2.2.3.4 TDO Slot

As shown in [Figure 2-8](#), the TDO operation is allocated one time slot (see also the detailed timing shown in [Figure 2-10](#)). The master should release control of the SBWTDIO line based off of the rising edge of SBWTCK of the TDI cycle. After the master releases the SBWTDIO line, an internal bus keeper holds the voltage on the line. The next falling edge of SBWTCK triggers the slave to start driving the bus. The slave only drives the SBWTDIO line during the low time of the SBWTCK cycle. The master should not enable its drivers until the slave has released the SBWTDIO line. Therefore, the master could use the rising edge of the SBWTCK signal as a trigger point to enable its driver.

Note

The low phase of the clock signal supplied on SBWTCK must not be longer than 7 μ s. If the low phase is longer, the SBW logic is deactivated, and it must be activated again according to [Section 2.3.1](#).

When using the provided source code example, make sure that interrupts are disabled during the SBWTCK low phase to ensure accurate timings.

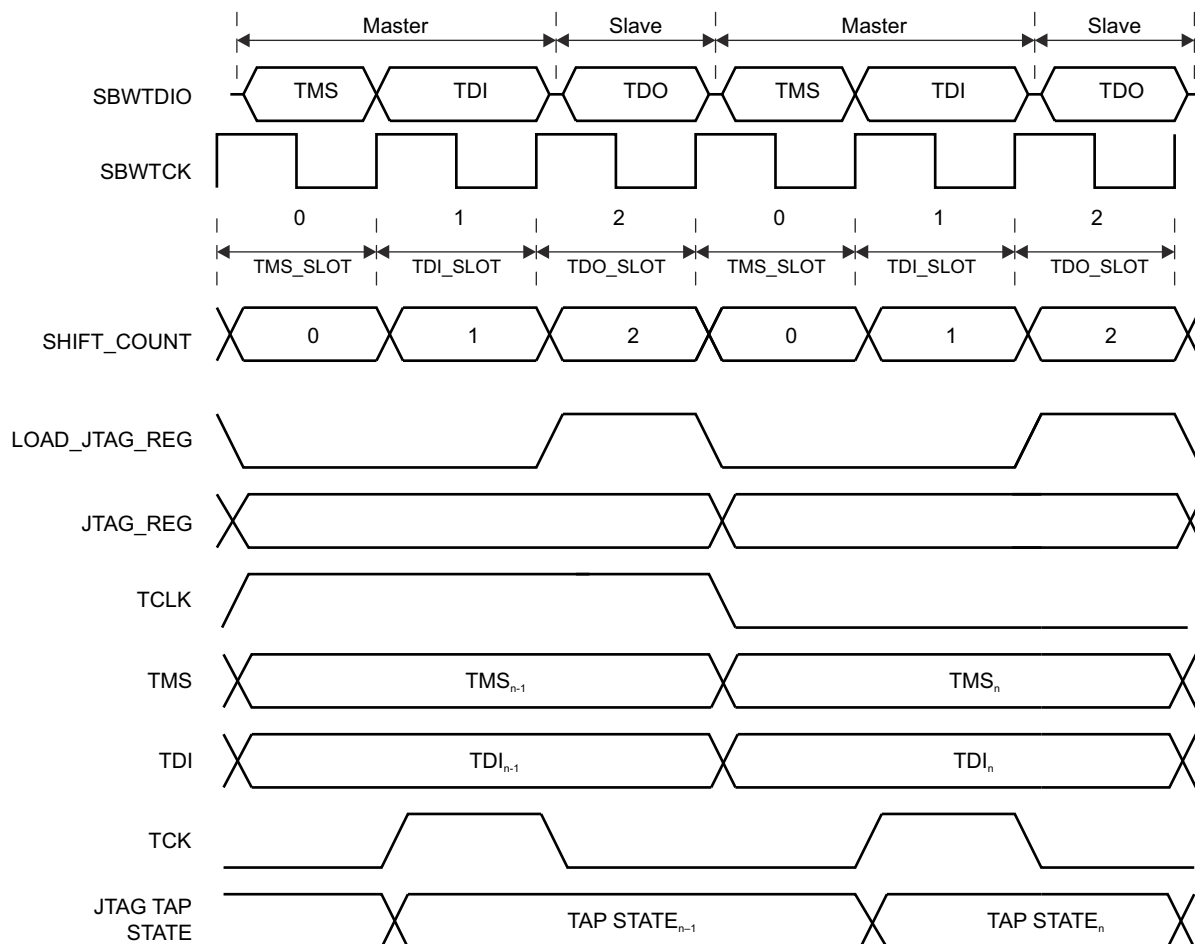


Figure 2-10. Detailed SBW Timing Diagram

The following macros are located in the LowLevelFunc header file in the Replicator example project.

2.2.3.4.1 TDO_RD Macro

`TRSLDIR |= TDOI_DIR` sets the SN74LVC1T45 transceiver on the board to Hi-Z, and
`TRSLDR &= TDOI_DIR` sets it back to being driven.

Used for reading TDO value during TDO slot.

- Set SBWTDIO Hi-Z
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK low
- NOP 5 cycles (delay at 18 MHz)
- Read SBWTDIO line
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK high
- Set SBWTDIO to driven again

2.2.3.4.2 TDOsbw Macro (No Read)

Used for clocking through TDO slot, when no read is required.

- Set SBWTDIO Hi-Z
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK low
- NOP 5 cycles (delay at 18 MHz)
- Set SBWTCK high
- Set SBWTDIO to driven again

2.2.3.5 TCLK Handling in Spy-Bi-Wire (SBW) Mode

2.2.3.5.1 SetTCLK and ClrTCLK

If the JTAG TAP controller is in the Run-Test/Idle state, the TDI slot can provide the TCLK signal (that is, it can clock the target CPU). Following this implementation, the generation of a complete TCLK clock cycle requires two TDI slots, one of which sets the TCLK signal and one of which clears it. In each case, the SBWTDIO signal must be set low in the TMS slot to keep the TAP controller from leaving the Run-Test/Idle state. To provide only a falling edge for ClrTCLK, the SBWTDIO signal must be set high before entering the TDI slot. The corresponding rising edge must occur in the low phase of SBWTCK in the TMS slot. Otherwise it would be interpreted as a trigger for TMS = 1 and the TAP controller would leave Run-Test/Idle mode.

Figure 2-11 shows handling of TCLK in SBW mode. See the reference functions SetTCLK_sb and ClrTCLK_sb in the MSP430 Replicator project ([slau320.zip](#)) for software implementation. The provided code example for the MSP430Xv2 architecture uses preprocessor definitions to enable a better layered software architecture. The upper software layers can simply reference the SetTCLK and ClrTCLK symbols while the actual implementation symbols are SetTCLK_4wire and ClrTCLK_4wire for 4-wire JTAG and SetTCLK_sb and ClrTCLK_sb for Spy-Bi-Wire (SBW).

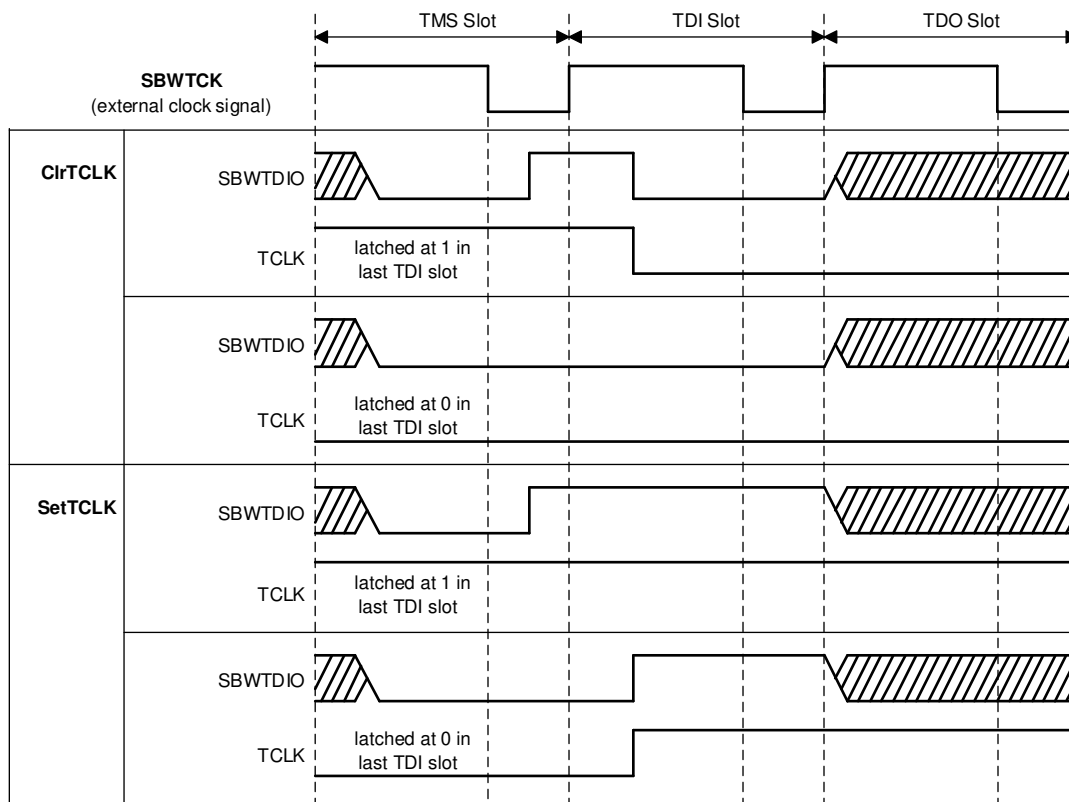


Figure 2-11. Synchronization of TDI and TCLK During Run-Test/Idle

2.2.3.5.2 TCLK Strokes

For MSP430 devices from the F1xx, F2xx, G2xx, and F4xx families, a custom number of TCLK clocks can be provided within a single TDI slot (refer to the example projects Replicator430 and Replicator430X). See reference function: TCLKstrokes(). This implementation is not applicable for the F5xx or F6xx devices. For those devices, the flash timing is generated by the internal MODOSC.

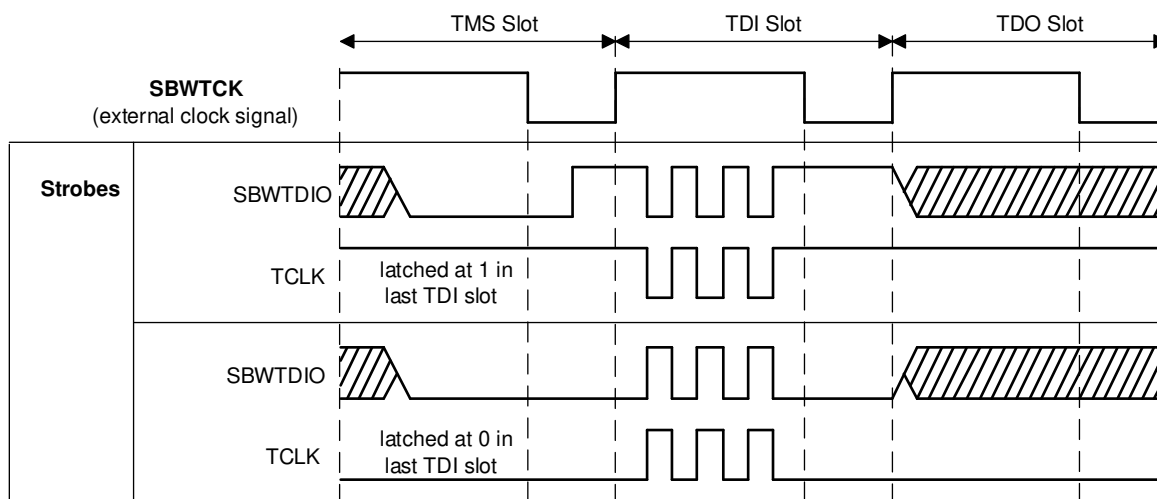


Figure 2-12. Clocking TCLK Using TCLK Strokes

2.2.4 JTAG Communication Instructions

Selecting a JTAG register and controlling the CPU is done by shifting in a JTAG instruction using the IR_SHIFT macro described in [Section 2.2.2.1.1](#). The following instructions that can be written to the JTAG IR are used to program the target memory. All instructions sent to the target MSP430 through the JTAG register are transferred LSB first.

Table 2-5. Memory Access Instructions

Instruction Name	8-Bit Instruction Value
Controlling the Memory Address Bus (MAB)	
IR_ADDR_16BIT	0x83
IR_ADDR_CAPTURE	0x84
Controlling the Memory Data Bus (MDB)	
IR_DATA_TO_ADDR	0x85
IR_DATA_16BIT	0x41
IR_DATA_QUICK	0x43
IR_BYPASS	0xFF
Controlling the CPU	
IR_CNTRL_SIG_16BIT	0x13
IR_CNTRL_SIG_CAPTURE	0x14
IR_CNTRL_SIG_RELEASE	0x15
Memory Verification by Pseudo Signature Analysis (PSA)	
IR_DATA_PSA	0x44
IR_SHIFT_OUT_PSA	0x46
JTAG Access Security Fuse Programming	
IR_Prepare_Blow	0x22
IR_Ex_Blow	0x24
JTAG Mailbox System	
IR_JMB_EXCHANGE	0x61

Note

Do not write any unlisted values to the JTAG instruction register. Instruction values written to the MSP430 JTAG register other than those listed above may cause undesired device behavior.

Note

When a new JTAG instruction is shifted into the JTAG instruction register, it takes effect with the UPDATE-IR state of the TAP controller. When accessing a JTAG data register, the last value written is captured with the CAPTURE-DR state, and the new value shifted in becomes valid with the UPDATE-DR state. In other words, there is no need to go through Run-Test/Idle state of the JTAG TAP controller to shift in instructions or data. Be aware of the fact that clocking TCLK is only possible in the Run-Test/Idle state. This is why the provided software example application exclusively makes use of the JTAG macros described in [Section 2.2.2](#), which always go through Run-Test/Idle state.

2.2.4.1 Controlling the Memory Address Bus (MAB)

The following instructions control the MAB of the target MSP430. To accomplish this, a 16-bit (or 20-bit in MSP430X architectures) register, which is called the JTAG MAB register, is addressed. By using the JTAG data path of the TAP controller, this register can be accessed and modified.

2.2.4.1.1 IR_ADDR_16BIT

This instruction enables setting of the MAB to a specific value, which is shifted in with the next JTAG 16-bit data access using the DR_SHIFT16 (16-bit Data) macro or the next JTAG 20-bit address word access using the DR_SHIFT (20-bit Address) macro. The MAB of the MSP430 CPU is set to the value written to the JTAG MAB register. The previous value stored in the JTAG MAB register is simultaneously shifted out on TDO while the new 16- or 20-bit address is shifted in through TDI.

Note

In MSP430X devices, a 16-bit shift to update the JTAG MAB register does not automatically reset the upper four bits (19:16) of the JTAG MAB register. Always use the 20-bit shift macro to ensure that the upper four bits (19:16) are set to a defined value.

2.2.4.1.2 IR_ADDR_CAPTURE

This instruction enables readout of the data on the MAB with the next 16- or 20-bit data access. The MAB value is not changed during the 16- or 20-bit data access; that is, the 16- or 20-bit data sent on TDI with this command is ignored (0 is sent as a default in the provided software).

In several places throughout the example code, the IR_ADDR_CAPTURE instruction is also used to set the CPU to a defined state after accessing the data bus with the IR_DATA_16BIT instruction.

2.2.4.2 Controlling the Memory Data Bus (MDB)

The following instructions control the MDB of the MSP430 CPU. To accomplish this, a 16-bit register, termed the JTAG MDB register, is addressed. By using the JTAG data path of the TAP controller, this register can be accessed and modified.

2.2.4.2.1 IR_DATA_TO_ADDR

This instruction enables setting of the MSP430 MDB to a specific value shifted in with the next JTAG 16-bit data access using the DR_SHIFT16 (16-bit Data) macro. The MDB of the MSP430 CPU is set to the value written to the JTAG MDB register. As the new value is written into the MDB register, the prior value in the MSP430 MDB is captured and shifted out on TDO. The MSP430 MAB is set by the value in the JTAG MAB register during execution of the IR_DATA_TO_ADDR instruction. This instruction is used to write to all memory locations of the MSP430.

2.2.4.2.2 IR_DATA_16BIT

This instruction enables setting of the MSP430 MDB to the specified 16-bit value shifted in with the next 16-bit JTAG data access. The complete MSP430 MDB is set to the value of the JTAG MDB register. At the same

time, the last value of the MSP430 MDB is captured and shifted out on TDO. In this situation, the MAB is still controlled by the CPU. The program counter (PC) of the target CPU sets the MAB value.

2.2.4.2.3 IR_DATA_QUICK

This instruction enables setting of the MSP430 MDB to a specific value shifted in with the next 16-bit JTAG data access. The 16-bit MSP430 MDB is set to the value written to the JTAG MDB register. During the 16-bit data transfer, the previous MDB value is captured and shifted out on TDO. The MAB value is set by the program counter (PC) of the CPU. This instruction auto-increments the program counter by two on every falling edge of TCLK to automatically point to the next 16-bit memory location. The program counter of the target CPU must be loaded with the starting memory address before execution of this instruction, which can be used to quickly read or write to a memory array (see [Section 2.3.2.1.3](#) for more information on setting the PC).

Note

IR_DATA_QUICK cannot be used to write flash memory.

Note

IR_DATA_QUICK cannot be used to read or write USB RAM as this is dual ported RAM. It needs to be accessed word by word.

2.2.4.2.4 IR_BYPASS

This instruction delivers the input to TDI as an output on TDO delayed by one TCK clock. When this instruction is loaded, the IR_CNTRL_SIG_RELEASE instruction (see [Section 2.2.4.3.3](#)) is performed simultaneously. After execution of the bypass instruction, the 16-bit data shifted out on TDI does not affect any register of the JTAG control module of the target MSP430 device.

2.2.4.3 Controlling the CPU

The following instructions enable control of the MSP430 CPU through a 16-bit register accessed through JTAG. This data register is called the JTAG control signal register. [Table 2-6](#) describes the bit functions making up the JTAG control signal register used for memory access.

Table 2-6. JTAG Control Signal Register for 1xx, 2xx, 4xx Families

Bit No.	Name	Description
0	R/W	Controls the read/write (RW) signal of the CPU 1 = Read 0 = Write
1	(N/A)	Always write 0
2	(N/A)	Always write 0
3	HALT_JTAG	Sets the CPU into a controlled halt state 1 = CPU stopped 0 = CPU operating normally
4	BYTE	Controls the BYTE signal of the CPU used for memory access data length 1 = Byte (8-bit) access 0 = Word (16-bit) access
5	(N/A)	Always write 0
6	(N/A)	Always write 0
7	INSTR_LOAD	Read only: Indicates the target CPU instruction state 1 = Instruction fetch state 0 = Instruction execution state
8	(N/A)	Always write 0
9	TCE	Indicates CPU synchronization 1 = Synchronized 0 = Not synchronized
10	TCE1	Establishes JTAG control over the CPU 1 = CPU under JTAG control 0 = CPU free running
11	POR	Controls the power-on-reset (POR) signal 1 = Perform POR 0 = No reset
12	Release low byte	Selects control source of the RW and BYTE bits 1 = CPU has control 0 = Control signal register has control
13	TAGFUNCSAT	Sets flash module into JTAG access mode 1 = CPU has control (default) 0 = JTAG has control
14	SWITCH	Enables TDO output as TDI input 1 = JTAG has control 0 = Normal operation
15	(N/A)	Always write 0

Table 2-7. JTAG Control Signal Register for 5xx and 6xx Families

Bit No.	Name	Description
0	R/W	Controls the read/write (RW) signal of the CPU, same as previous families. 1 = Read 0 = Write
1	(N/A)	Always write 0, same as previous families.
2	(N/A)	Always write 0, same as previous families.
3	WAIT	Wait signal to the CPU. Read only. 1 = CPU clock stopped - waiting for an operation to complete 0 = CPU clock not stopped
4	BYTE	Controls the BYTE signal of the CPU used for memory access data length, same as previous families. 1 = Byte (8-bit) access 0 = Word (16-bit) access
5	(N/A)	Always write 0
6	(N/A)	Always write 0
7	INSTR_LOAD	Read only: Indicates the target CPU instruction state. The actual state is not the same as previous families. 1 = Instruction fetch state 0 = Instruction execution state
8	CPUSUSP	Suspend CPU. The CPU pipeline is emptied by asserting the CPUSUSP bit and driving a minimum number of clocks required to complete the longest possible instructions. When CPUSUSP is high, no instructions are fetched or executed. To execute a forced external instruction sequence through JTAG (for example, set the Program Counter), the CPUSUSP bit must be zero. 0 = CPU active 1 = CPU suspended Reading CPUSUSP (Bit 8) shows if pipeline is empty: 0 = Pipeline is not empty yet 1 = Pipeline is empty
9	TCE0	Indicates CPU synchronization, same as previous families. 1 = Synchronized 0 = Not synchronized
10	TCE1	Establishes JTAG control over the CPU, same as previous families. 1 = CPU under JTAG control 0 = CPU free running
11	POR	Controls the power-on-reset (POR) signal, same as previous families. 1 = Perform POR 0 = No reset
12	RELEASE_LBYTE0	Release control bits in low byte from JTAG control. 00 = All bits are controlled by JTAG if TCE1 is 1 01 = RW (bit 0) and BYTE (bit 4) are released from JTAG control 10 = RW (bit 0), HALT (bit 1), INTREQ (bit 2), and BYTE (bit 4) are released from JTAG control 11 = Reserved
13	RELEASE_LBYTE1	
14	INSTR_SEQ_NO0	Instruction sequence number. Read only. Shows the instruction sequence number of the pipelined CPU currently using the CPU bus (there is a maximum of three instructions in the pipe). 00 = CPU instruction sequence 0 01 = CPU instruction sequence 1 10 = CPU instruction sequence 2 11 = CPU generated "no-operation" cycle; data on buses is not used
15	INSTR_SEQ_NO1	

2.2.4.3.1 IR_CNTRL_SIG_16BIT

This instruction enables setting of the complete JTAG control signal register with the next 16-bit JTAG data access. Simultaneously, the last value stored in the register is shifted out on TDO. The new value takes effect when the TAP controller enters the UPDATE-DR state.

2.2.4.3.2 IR_CNTRL_SIG_CAPTURE

This instruction enables readout of the JTAG control signal register with the next JTAG 16-bit data access instruction.

2.2.4.3.3 IR_CNTRL_SIG_RELEASE

This instruction completely releases the CPU from JTAG control. Once executed, the JTAG control signal register and other JTAG data registers no longer have any effect on the target MSP430 CPU. This instruction is normally used to release the CPU from JTAG control.

2.2.4.4 Memory Verification by Pseudo Signature Analysis (PSA)

The following instructions support verification of the MSP430 memory content by means of a PSA mode.

2.2.4.4.1 IR_DATA_PSA

The IR_DATA_PSA instruction switches the JTAG_DATA_REG into the PSA mode. In this mode, the program counter of the MSP430 is incremented by every two system clocks provided on TCLK. The CPU program counter must be loaded with the start address before execution of this instruction. The number of TCLK clocks determines how many memory locations are included in the PSA calculation.

2.2.4.4.2 IR_SHIFT_OUT_PSA

The IR_SHIFT_OUT_PSA instruction should be used in conjunction with the IR_DATA_PSA instruction. This instruction shifts out the PSA pattern generated by the IR_DATA_PSA command. During the SHIFT-DR state of the TAP controller, the content of the JTAG_DATA_REG is shifted out through the TDO pin. While this JTAG instruction is executed, the capture and update functions of the JTAG_DATA_REG are disabled.

2.2.4.5 JTAG Access Security Fuse Programming

The following instructions are used to access and program the built-in JTAG access protection fuse, available on every MSP430F1xx, 2xx, and 4xx flash device. When the fuse is programmed (or blown), future access to the MSP430 through the JTAG interface is permanently disabled. This allows for access protection of the final MSP430 firmware programmed into the target device. These instructions are not available for the MSP430F5xx and F6xx devices. A different software-based mechanism is used for these families to enable JTAG access protection (see [Section 2.4](#) for details).

2.2.4.5.1 IR_PREPARE_BLOW

This instruction sets the MSP430 into program-fuse mode.

2.2.4.5.2 IR_EX_BLOW

This instruction programs (blows) the access-protection fuse. To execute properly, it must be loaded after the IR_PREPARE_BLOW instruction is given.

2.3 Memory Programming Control Sequences

2.3.1 Start-Up

Before the main memory programming routine can begin, the target device must be initialized for programming. This section describes how to perform the initialization sequence.

2.3.1.1 Enable JTAG Access

This step is only required for devices in which JTAG pins are shared with port I/Os, which is indicated by the presence of a TEST pin. This is the case on all devices that support the Spy-Bi-Wire protocol. Furthermore, some older device groups require special handling to enable 4-wire JTAG (see the "TEST Pin" column in [Table 2-14](#)).

Reference function: GetDevice, GetDevice_sbww, GetDevice_430X, GetDevice_430Xv2

- MSP430 devices with TEST pin and 4-wire JTAG access only (no SBW)

To use the JTAG features of MSP430 devices with shared JTAG and a TEST pin, it is necessary to enable the shared JTAG pins for JTAG communication mode. Devices with dedicated JTAG inputs/outputs and no TEST pin do not require this step. The shared pins are enabled for JTAG communication by connecting the TEST pin to V_{CC} . For normal operation (non-JTAG mode), this pin should be released, so that it is pulled to ground by the internal pulldown. [Table 2-8](#) shows the port 1 pins that are used for JTAG communication.

Table 2-8. Shared JTAG Device Pin Functions

Port 1 Function (TEST = Open)	JTAG Function (TEST = V_{CC})
P1.4	TCK
P1.5	TMS
P1.6	TDI/TCLK
P1.7	TDO

- MSP430 devices with Spy-Bi-Wire (SBW) access

The SBW interface and any access to the JTAG interface is disabled while the TEST/SBWTCK pin is held low. This is accomplished by an internal pulldown resistor. The pin can also be tied low externally.

Pulling the TEST/SBWTCK pin high enables the SBW interface and disables the \overline{RST}/NMI functionality of the $\overline{RST}/NMI/SBWTDO$ pin. While the SBW interface is active, the internal reset signal is held high, and the internal NMI signal is held at the input value seen at \overline{RST}/NMI with TEST/SBWTCK going high.

Devices with SBW also support the standard 4-wire interface. The 4-wire JTAG interface access is enabled by pulling the SBWTDO line low and then applying a clock on SBWTCK. Exit the 4-wire JTAG mode by holding the TEST/SBWTCK low for more than 100 μs .

To select the 2-wire SBW mode, the SBWTDO line is held high and the first clock is applied on SBWTCK. After this clock, the normal SBW timings are applied starting with the TMS slot, and the normal JTAG patterns can be applied, typically starting with the Tap Reset and Fuse Check sequence. Exit the SBW mode by holding the TEST/SBWTCK low for more than 100 μs .

In devices implementing the bootloader (BSL), the TEST/SBWTCK and $\overline{RST}/NMI/SBWTDO$ are also used to invoke the BSL. [Figure 2-13](#) shows different cases that are used to enter the SBW/JTAG or BSL mode.

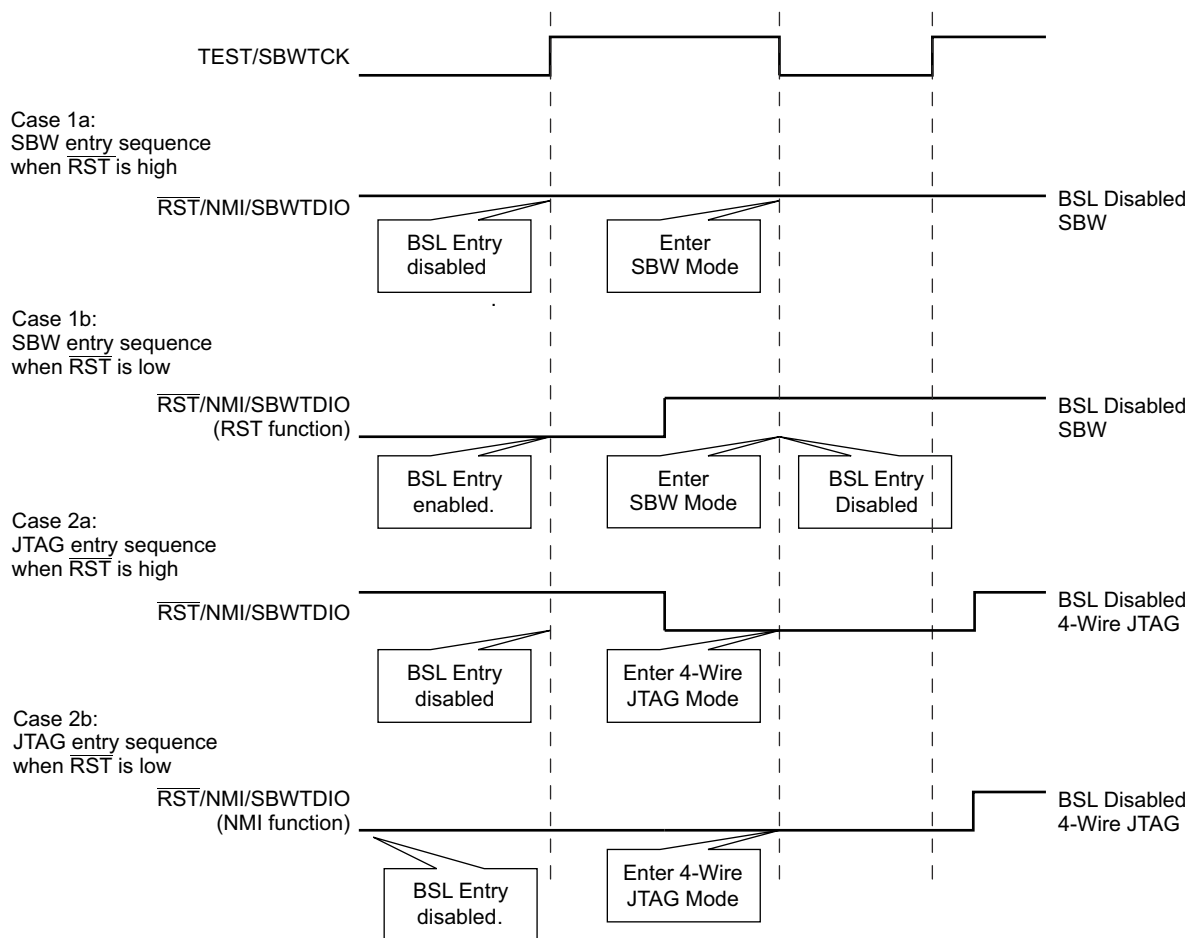


Figure 2-13. JTAG Access Entry Sequences (for Devices That Support SBW)

Note

On some Spy-Bi-Wire capable MSP430 devices, TEST/SBWTCK is very sensitive to rising signal edges that can cause the test logic to enter a state where an entry sequence (either 2-wire or 4-wire) is not recognized correctly and JTAG access stays disabled. Unintentional edges on SBWTCK can occur when the JTAG connector is connected to the target device. There are two possibilities to work around this problem and ensure a stable JTAG access initialization:

- Actively drive SBWTCK low before powering up the device or while plugging in the connector to avoid unintentional rising signal edges.
- Run the initialization sequence multiple times (two to three repeats are typically sufficient to establish a stable connection).

2.3.1.2 Fuse Check and Reset of the JTAG State Machine (TAP Controller)

Reference functions: ResetTAP, ResetTAP_sbww

Each MSP430F1xx, 2xx, and 4xx flash device includes a physical fuse that is used to permanently disable memory access through JTAG communication. When this fuse is programmed (or blown), access to memory through JTAG is permanently disabled and cannot be restored. When initializing JTAG access after power up, a fuse check must be done before JTAG access is granted. Toggling of the TMS signal twice performs the check.

While the fuse is tested, a current of up to 2 mA flows into the TDI input (or into the TEST pin on devices without dedicated JTAG pins). To enable settling of the current, the low phase of the two TMS pulses should last a minimum of 5 μ s.

Under certain circumstances (for example, plugging in a battery), a toggling of TMS may accidentally occur while TDI is logical low. In that case, no current flows through the security fuse, but the internal logic remembers that a fuse check was performed. Thus, the fuse is mistakenly recognized as programmed (that is, blown). To avoid the issue, newer MSP430 JTAG implementations (devices with CPUXv2 - see [Table 2-15](#)) also reset the internal fuse-check logic on performing a reset of the TAP controller. Thus, it is recommended to first perform a reset of the TAP and then check the JTAG fuse status as shown in [Figure 2-14](#). To perform a reset of the TAP controller it is recommended that a minimum of six TCK clocks be sent to the target device while TMS is high followed by setting TMS low for at least one TCK clock. This sets the JTAG state machine (TAP controller) to a defined starting point: the Run-Test/Idle state. This procedure can also be used at any time during JTAG communication to reset the JTAG port.

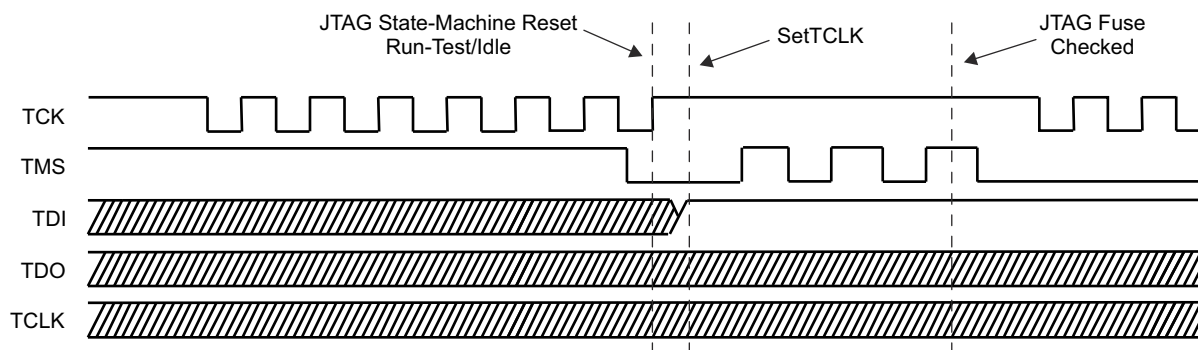


Figure 2-14. Fuse Check and TAP Controller Reset

Following the same sequence in SBW mode has the side effect of changing the TAP controller state while the fuse check is performed. As described in [Section 2.2.3.1](#), the internal signal TCK is generated automatically in every TDI_SLOT. Performing a fuse check in SBW mode, starting directly after a reset of the TAP controller, ends in its Exit2-DR state. Two more dummy TCK cycles must be generated to return to Run-Test/Idle state; one TCK with SBWTDIO being high during the TMS_SLOT followed by one TCK with SBWTDIO being low during the TMS_SLOT (reference function: ResetTAP_sbww).

Note

A dedicated fuse check sequence (toggling TMS twice) is not required for the MSP430F5xx and F6xx families. These families implement a software mechanism rather than a hardware fuse (which needs to be checked or burned) to enable JTAG security protection.

2.3.2 General Device (CPU) Control Functions

The functions described in this section are used for general control of the target MSP430 CPU, as well as high-level JTAG access and bus control.

2.3.2.1 Function Reference for 1xx, 2xx, 4xx Families

2.3.2.1.1 Taking the CPU Under JTAG Control

Reference function: GetDevice, GetDevice_sbww, GetDevice_430X

After the initial fuse check and reset, the CPU of the target device must be taken under JTAG control. This is done by setting bit 10 (TCE1) of the JTAG control signal register to 1. Thereafter, the CPU needs some time to synchronize with JTAG control. To check if the CPU is synchronized, bit 9 (TCE) is tested (sync successful if set to 1). After this bit is verified as high, the CPU is under the control of the JTAG interface. Following is the flow used to take the target device under JTAG control.

IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2401)	
IR_SHIFT("IR_CNTRL_SIG_CAPTURE")	
DR_SHIFT16(0x0000)	
Bit 9 of TDOWord = 1?	No
Yes	
CPU is under JTAG control	

2.3.2.1.2 Set CPU to Instruction-Fetch

Reference function: SetInstrFetch

Sometimes it is useful for the target device to directly execute an instruction presented by a host over the JTAG port. To accomplish this, the CPU must be set to the instruction-fetch state. With this setting, the target device CPU loads and executes an instruction as it would in normal operation, except that the instruction is transmitted through JTAG. Bit 7 of the JTAG control signal register indicates that the CPU is in the instruction-fetch state. TCLK should be toggled while this bit is zero. After a maximum of seven TCLK clocks, the CPU should be in the instruction-fetch mode. If not (bit 7 = 1), a JTAG access error has occurred and a JTAG reset is recommended.

IR_SHIFT("IR_CNTRL_SIG_CAPTURE")	
DR_SHIFT16(0x0000) = Readout data	
Bit 7 of TDOWord = 0?	
	ClrTCLK
	SetTCLK
CPU is in the instruction-fetch state	

2.3.2.1.3 Setting the Target CPU Program Counter (PC)

To use some of the features of the JTAG interface provided by the MSP430, setting of the CPU PC of the target device is required. The following flow is used to accomplish this. Implementations for both the MSP430 and MSP430X architectures are shown.

- MSP430 architecture: Reference function: SetPC

<i>CPU must be in the instruction-fetch state before the following sequence.</i>
IR_SHIFT("IR_CNTRL_SIG_16BIT")
DR_SHIFT16(0x3401) : release low byte
IR_SHIFT("IR_DATA_16BIT")
DR_SHIFT16(0x4030) : Instruction to load PC
ClrTCLK
SetTCLK
DR_SHIFT16("PC_Value") : Insert the value for PC
ClrTCLK
SetTCLK
IR_SHIFT("IR_ADDR_CAPTURE") : Disable IR_DATA_16BIT
ClrTCLK : Now PC is set to "PC_Value"
IR_SHIFT("IR_CNTRL_SIG_16BIT")
DR_SHIFT16(0x2401) : low byte controlled by JTAG
<i>Load PC completed</i>

- MSP430X architecture: Reference function: SetPC_430X

<i>CPU must be in the instruction-fetch state before the following sequence.</i>
IR_SHIFT("IR_CNTRL_SIG_16BIT")
DR_SHIFT16(0x3401) : release low byte
IR_SHIFT("IR_DATA_16BIT")
DR_SHIFT16(0x0X80) : Instruction to load PC, X = PC(19:16)
ClrTCLK
SetTCLK
DR_SHIFT16("PC(15:0)") : Insert the value for PC(15:0)
ClrTCLK
SetTCLK
IR_SHIFT("IR_ADDR_CAPTURE") : Disable IR_DATA_16BIT
ClrTCLK : Now PC is set to "PC_Value"
IR_SHIFT("IR_CNTRL_SIG_16BIT")
DR_SHIFT16(0x2401) : low byte controlled by JTAG
<i>Load PC completed</i>

2.3.2.1.4 Controlled Stop or Start of the Target CPU

Reference function: HaltCPU/ReleaseCPU

While a memory location is accessed by the JTAG interface, the target device's CPU should be taken into a defined halt state. Stopping of the CPU is supported by the HALT_JTAG bit (bit 3) in the JTAG control signal register, which is set to 1 with execution of the HaltCPU function. After accessing the required memory locations, the CPU can be returned to normal operation. This function is implemented by the ReleaseCPU prototype and simply resets the HALT_JTAG bit.

<i>CPU must be in the instruction-fetch state before the following sequence</i>	
HaltCPU	IR_SHIFT("IR_DATA_16BIT")
	DR_SHIFT16(0x3FFF) : "JMP \$" instruction to keep CPU from changing the state
	ClrTCLK
	IR_SHIFT("IR_CNTRL_SIG_16BIT")
	DR_SHIFT16(0x2409) : set HALT_JTAG bit
	SetTCLK
<i>Now the CPU is in a controlled state and is not altered during memory accesses. Note: Do not reset the HALT_JTAG bit (= 0) while accessing the target memory.</i>	
Memory Access Performed Here	
<i>The CPU is switched back to normal operation using ReleaseCPU.</i>	
ReleaseCPU	ClrTCLK
	IR_SHIFT("IR_CNTRL_SIG_16BIT")
	DR_SHIFT16(0x2401) : Clear HALT_JTAG bit
	IR_SHIFT("IR_ADDR_CAPTURE")
	SetTCLK
<i>The CPU is now in the instruction-fetch state and ready to receive a new JTAG instruction. If the PC has been changed while the memory was being accessed, the PC must be loaded with the correct address.</i>	

2.3.2.1.5 Resetting the CPU While Under JTAG Control

Reference function: ExecutePOR

Sometimes it is required to reset the target device while under JTAG control. It is recommended that a reset be performed before programming or erasing the flash memory of the target device. When a reset has been performed, the state of the target CPU is equivalent to that after an actual device power up. The following flow is used to force a power-up reset.

IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2C01)	: Apply Reset
DR_SHIFT16(0x2401)	: Remove Reset
ClrTCLK	
SetTCLK	
ClrTCLK	
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_CAPTURE")	
SetTCLK	
<i>The target CPU is now reset; the PC points to the start address of the user program, which is the address pointed to by the data stored in the reset vector memory location 0xFFFFh and all registers are set to their respective power-up values.</i>	
<i>To avoid an undesired reset, the target device's watchdog timer must now be disabled by writing 0x5A80 to the device's Watchdog Timer Control Register. WriteMem() can be used for this register access.</i>	
<i>Unless WriteMem() was used to halt the watchdog timer, ReleaseCPU() needs to be called now.</i>	

2.3.2.1.6 Release Device From JTAG Control

Reference function: ReleaseDevice

After the desired JTAG communication is completed, the CPU is released from JTAG control. There are two ways to accomplish this task:

- Disconnect the external JTAG hardware and perform a true power-up reset. The MSP430 then starts executing the program code beginning at the address stored at 0xFFFFh (the reset vector).
- Release MSP430 from JTAG control. This is done by performing a reset using the JTAG control signal register. The CPU must then be released from JTAG control by using the IR_CNTRL_SIG_RELEASE instruction. The target MSP430 then starts executing the program at the address stored at 0xFFFFh.

Flow to release the target device:

IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2C01)	: Apply Reset
DR_SHIFT16(0x2401)	: Remove Reset
IR_SHIFT("IR_CNTRL_SIG_RELEASE")	
<i>The target CPU starts program execution with the address stored at location 0x0FFFFE (reset vector).</i>	

Note

It is not recommended to release the device from JTAG control (or perform a power-up cycle) during an erase-program-verify memory access cycle. Releasing the device from JTAG control starts execution of the previously programmed user code, which might change the flash memory content. In that case, verification of the memory content against the originally programmed code image would fail.

2.3.2.2 Function Reference for 5xx and 6xx Families

2.3.2.2.1 Taking the CPU Under JTAG Control

Reference function: GetDevice_430Xv2

For the 5xx and 6xx families, the CPU is taken under JTAG control by setting bit 10 (TCE1) of the JTAG control signal register to 1. While the flow to take the target device under JTAG control is identical to the flow described in [Section 2.3.2.1.1](#), additional actions must be taken to completely take over control of the target CPU; for example, it is not recommended to take over control without performing a CPU reset by setting the POR signal in the JTAG Control Signal Register. Also, care must be taken that the CPU is in the Full-Emulation-State (equivalent to the Instruction-Fetch state for MSP430 and MSP430X architectures) by setting the CPUSUSP signal and providing a number of TCLK until the CPU prefetch pipes are cleared. [Figure 2-15](#) shows the full sequence required to get the CPU in the Full-Emulation-State.

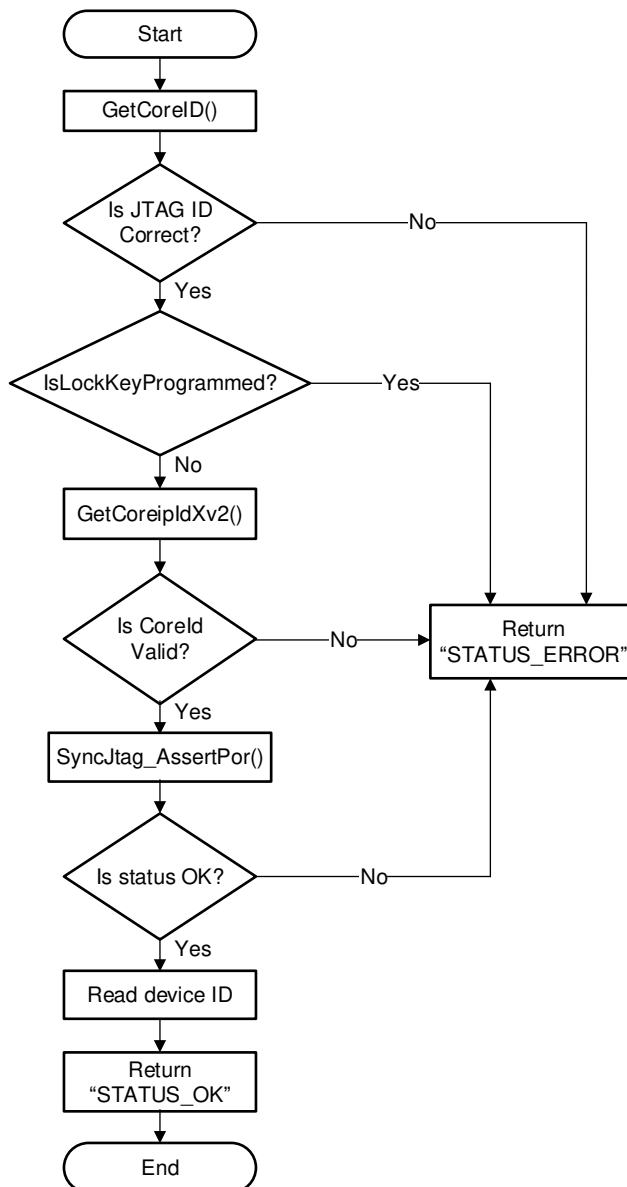


Figure 2-15. Taking the CPU Under JTAG Control

Reference function: GetCoreID

Figure 2-16 shows the JTAG-entry sequence for MSP430Xv2 devices. If no valid JTAG-ID is returned by the first entry sequence approach, a second one that uses a "magic pattern" is executed. The differences between these two entry approaches is: the second approach holds the device in reset and shifts in a "magic pattern" (0xA55A) by using the JTAG mailbox. The magic pattern is read by the BootCode, and the device is sent into LPM4. If the device is in LPM4, no user code is executed. The magic pattern mode forces a reset of the device.

In the special case that the device is in LPMx.5 (low-power mode where JTAG is unpowered and the JTAG pins are locked by the ioLock), another mechanism is needed to take the device under JTAG control. Only the TEST and the REST pin are not pulled down by the ioLock. That means that these pins must be used to get control over the device. Compared to the normal SBW communication, which uses TDI/TDO and TMS pins in the replicator implementation, TEST and RST are used for SBW communication to disable the ioLock.

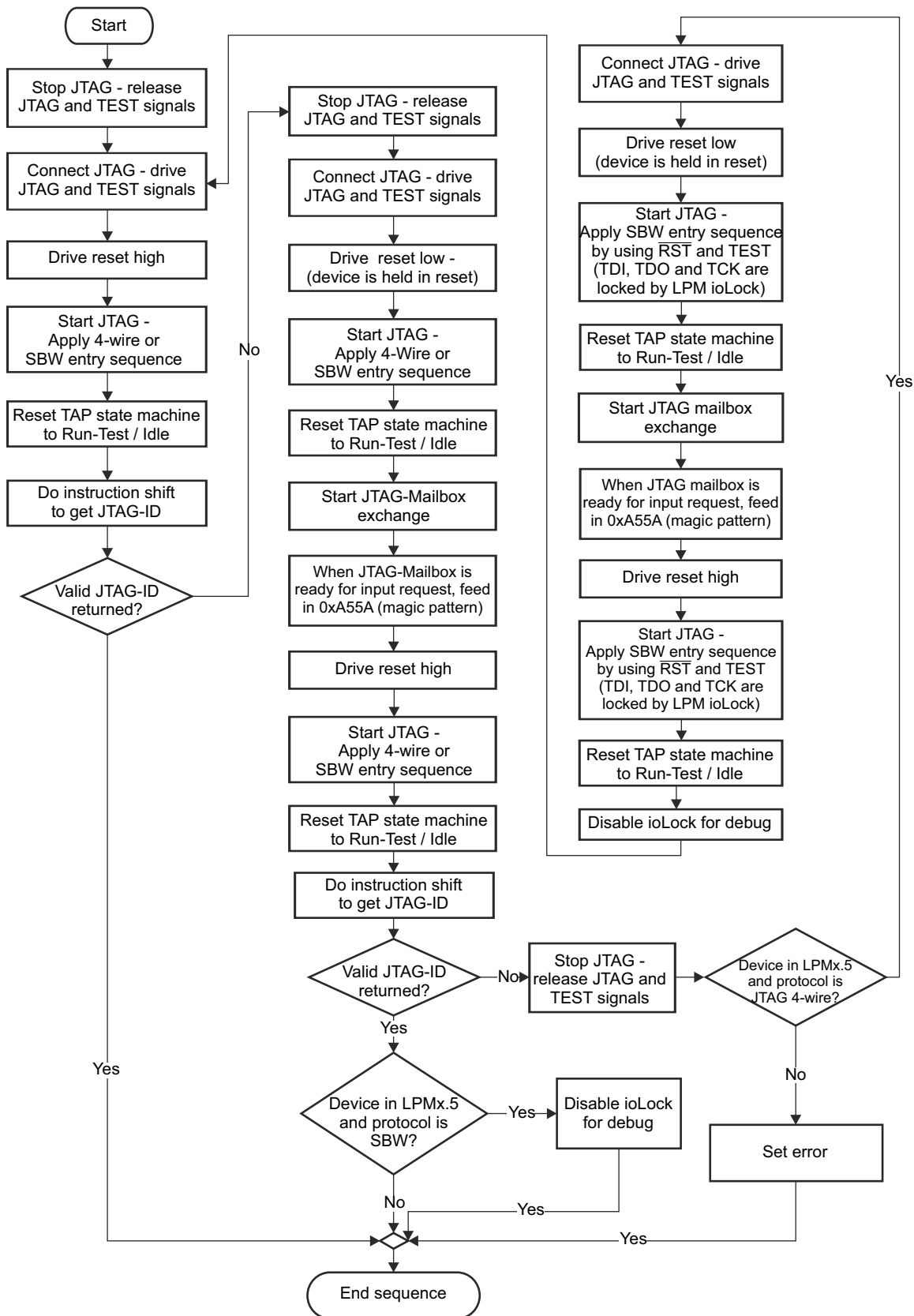


Figure 2-16. JTAG Entry Sequence for 430Xv2 Devices

- Reference function: GetCoreIdXv2()

IR_Shift(IR_COREID_ID)	
CoreId = DR_Shift16(0)	
CoreId = 0 ?	Yes
No	
IR_Shift(IR_DEVICE_ID)	
DeviceIdPointer = DR_Shift20(0)	

- Reference function: SyncJtag_AssertPor()

IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x1501)	
IR_SHIFT("IR_CNTRL_SIG_CAPTURE")	
DR_SHIFT16(0x0000)	No
Bit 9 of TDOword = 1?	
Yes	
CPU is under JTAG control - always apply Power on Reset (POR) afterwards.	

2.3.2.2.2 Setting the Target CPU Program Counter (PC)

To use some of the features of the JTAG interface provided by the MSP430Xv2 architecture, setting of the CPU PC of the target device is required. The following flow is used to accomplish this. With the MSP430Xv2 architecture, it is strongly recommended that after setting the PC no additional memory access is performed other than the described quick access methods under [Section 2.3.3.3](#) and [Section 2.3.7](#). After setting the PC, the target device can be either released from JTAG control or continued to be clocked by providing TCLK to execute user program code that was previously stored at the memory location the PC is now pointing to. In any case, before the memory can be accessed again, the CPU must be put again into the Full-Emulation-State as described in [Section 2.3.2.2.1](#).

- MSP430Xv2 architecture: Reference function: SetPC_430Xv2

CPU must be in the Full-Emulation-State before the following sequence.	
ClrTCLK	
IR_SHIFT("IR_DATA_16BIT")	
SetTCLK	
DR_SHIFT16("MOVA opcode incl. upper nibble of 20 bit PC value")	
clrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x1400) : release low byte	
IR_SHIFT("IR_DATA_16BIT")	
ClrTCLK	
SetTCLK	
DR_SHIFT16("PC_Value") : Insert the lower 16 bit value for PC	
ClrTCLK	
SetTCLK	
DR_SHIFT16(0x4303) : insert NOP instruction to be pre-fetched by CPU	
ClrTCLK : Now PC is set	
IR_SHIFT("IR_ADDR_CAPTURE") : Disable IR_DATA_16BIT	
DR_SHIFT20(0x000000)	

Load PC completed

2.3.2.2.3 Resetting the CPU While Under JTAG Control

Reference function: ExecutePOR_430Xv2

Note

On devices with Low Energy Accelerator (LEA), executing a POR using the described functions resets the LEA module.

- MSP430Xv2 architecture with Flash memory

IR_SHIFT("IR_CNTRL_SIG_CAPTURE") : returns JTAG ID
ClrTCLK : provide one clock cycle to empty the pipe
SetTCLK
IR_SHIFT("IR_CNTRL_SIG_16BIT") : prepare access to the JTAG CNTRL SIG register
DR_Shift16(0x0C01) : release CPUSUSP signal and apply POR signal
DR_Shift16(0x0401) : release POR signal again
ClrTCLK
SetTCLK
ClrTCLK
SetTCLK
ClrTCLK
SetTCLK
ClrTCLK : two more clock cycles to release CPU internal POR delay signals
SetTCLK
ClrTCLK
SetTCLK
IR_SHIFT("IR_CNTRL_SIG_16BIT") : set CPUSUSP signal again
DR_Shift16(0x0501)
ClrTCLK : ...and provide one more clock cycle
SetTCLK
<i>The CPU is now in Full-Emulation-State'</i>
<i>Disable Watchdog Timer on target device now by setting the HOLD signal in the WDT_CNTRL register (that is, by using WriteMem_430Xv2</i>
<i>'Full-Emulation-State' can be checked by the running the commands below</i>
IR_Shift("IR_CNTRL_SIG_CAPTURE")
DR_Shift16(0) : return value & 0x0301 should be true

- MSP430Xv2 architecture with FRAM memory

ClrTCLK : provide one clock cycle to empty the pipe
SetTCLK
IR_SHIFT("IR_CNTRL_SIG_16BIT") : prepare access to the JTAG CNTRL SIG register
DR_Shift16(0x0C01) : release CPUSUSP signal and apply POR signal
DR_Shift16(0x0401) : release POR signal again
IR_Shift(IR_DATA_16BIT) : set PC to safe memory location

ClrTCLK
SetTCLK
ClrTCLK
SetTCLK
DR_Shift16(SAFE_FRAM_PC) : PC is set to 0x4 - MAB value can be 0x6 or 0x8
ClrTCLK : Drive safe address into PC
SetTCLK
IR_SHIFT("IR_DATA_CAPTURE")
ClrTCLK : two more clock cycles to release CPU internal POR delay signals
SetTCLK
ClrTCLK
SetTCLK
IR_SHIFT("IR_CNTRL_SIG_16BIT") : set CPUSUSP signal again
DR_Shift16(0x0501)
ClrTCLK : ...and provide one more clock cycle
SetTCLK
The CPU is now in Full-Emulation-State'
Disable Watchdog Timer on target device now by setting the HOLD signal in the WDT_CNTRL register (i.e. by using WriteMem_430Xv2 – note different WDT addresses for individual FRAM device groups)
Initialize Test Memory with default values to ensure consistency between PC value and MAB (MAB is +2 after sync) – Use WriteMem_430Xv2 to write 0x3FFF to addresses 0x06 and 0x08 (this is only applicable for devices with JTAG ID 0x91 or 0x99)
'Full-Emulation-State' can be checked by the running the commands below
IR_Shift("IR_CNTRL_SIG_CAPTURE")
DR_Shift16(0) : return value & 0x0301 should be true

2.3.2.2.4 Release Device From JTAG Control

Also for 5xx and 6xx family, both ways described in [Section 2.3.2.1.6](#) can be applied. A POR (power-on-reset) using the JTAG control signal register is not equivalent to a true power-up reset, which additionally issues a BOR (brownout reset) before the POR. Only a BOR causes the target devices' boot code to be executed, which performs various calibration and configuration tasks. Thus, the 5xx and 6xx JTAG interface is enhanced with the ability to generate a BOR through the JTAG interface by accessing a dedicated JTAG data register. As soon as the appropriate BOR bit in the JTAG data register is set, the device is released from JTAG and performs a complete brownout reset startup sequence. See the ReleaseDevice_Xv2 reference function for implementation details. Also see the [MSP430F5xx and MSP430F6xx Family User's Guide System Resets, Interrupts and Operating Modes, System Control Module \(SYS\)](#) chapter for more information about various reset sources and device boot behavior.

Reference function: ReleaseDevice_430Xv2

2.3.2.2.5

Note

It is not recommended to release the device from JTAG control (or perform a power-up cycle) during an erase-program-verify memory access cycle. Releasing the device from JTAG control starts execution of the previously programmed user code, which might change the flash memory content. In that case, verification of the memory content against the originally programmed code image would fail.

2.3.3 Accessing Non-Flash Memory Locations With JTAG

2.3.3.1 Read Access

To read from any memory address location (peripherals, RAM, or flash/FRAM), the R/W signal must be set to READ using the JTAG control signal register (bit 0 set to 1). The MSP430 MAB must be set to the specific address to be read using the IR_ADDR_16BIT instruction while TCLK is 0. To capture the corresponding value of the MSP430 MDB, the IR_DATA_TO_ADDR instruction must be executed. After the next rising edge of TCLK, the data of this address is present on the MDB. The MDB can now be captured and read out from the TDO pin using a 16-bit JTAG data access. When TCLK is set low again, the address of the next memory location to be read can be applied to the target MAB. Following is the flow required to read data from any memory address of a target device. Implementations for both the MSP430 and MSP430X architectures are shown.

- MSP430 architecture, Reference function: ReadMem

Set CPU to stopped state (HaltCPU)		
ClrTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x2409)	: Read one word from memory. To read a byte, the value to shift is 0x2419.	
IR_SHIFT("IR_ADDR_16BIT")	Yes	
DR_SHIFT16("Address")		: Set desired address
IR_SHIFT("IR_DATA_TO_ADDR")		
SetTCLK		
ClrTCLK		
DR_SHIFT16(0x0000)		: Memory value shifted out on TDO
Read again?		
No		
ReleaseCPU should now be executed, returning the CPU to normal operation.		

- MSP430X architecture, Reference function: ReadMem_430X

Set CPU to stopped state (HaltCPU)		
ClrTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x2409)	: Read one word from memory. To read a byte, the value to shift is 0x2419.	
IR_SHIFT("IR_ADDR_16BIT")	Yes	
DR_SHIFT20("Address")		: Set desired address
IR_SHIFT("IR_DATA_TO_ADDR")		
SetTCLK		
ClrTCLK		
DR_SHIFT16(0x0000)		: Memory value shifted out on TDO
Read again?		
No		
ReleaseCPU should now be executed, returning the CPU to normal operation.		

- MSP430Xv2 architecture, Reference function: ReadMem_430Xv2

CPU must be in the Full-Emulation-State before the following sequence.	
ClrTCLK	Yes
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x0501) : Read one word from memory. To read a byte, the value to shift is 0x0511.	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT20("Address") : Set desired address	
IR_SHIFT("IR_DATA_TO_ADDR")	
SetTCLK	
ClrTCLK	
DR_SHIFT16(0x0000) : Memory value shifted out on TDO	
SetTCLK	
ClrTCLK	
SetTCLK	
Read again?	
No	
CPU is now again in Full-Emulation-State.	

2.3.3.2 Write Access

To write to a memory location in peripherals or to RAM (but not to flash or FRAM), the R/W signal must be set to WRITE using the JTAG control signal register (bit 0 set to 0). The MAB must be set to the specific address using the IR_ADDR_16BIT instruction while TCLK is low. The MDB must be set to the data value to be written using the IR_DATA_TO_ADDR instruction and a 16-bit JTAG data input shift. On the next rising edge of TCLK, this data is written to the selected address set by the value on the MAB. When TCLK is asserted low, the next address and data to be written can be applied to the MAB and MDB. After completion of the write operation, it is recommended to set the R/W signal back to READ. Following is the flow for a peripheral or RAM memory address write. Implementations for both the MSP430 and MSP430X architectures are shown.

- MSP430 architecture, Reference function: WriteMem

Set CPU to stopped state (HaltCPU)		
ClrTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x2408)	: To write a word in memory. For a byte, the value to shift is 0x2418.	
IR_SHIFT("IR_ADDR_16BIT")	Yes	
DR_SHIFT16("Address")		: Set desired address
IR_SHIFT("IR_DATA_TO_ADDR")		
DR_SHIFT16("Data")		: Send 16-bit Data
SetTCLK		
Write again?		
No		
ReleaseCPU should now be executed, returning the CPU to normal operation.		

- MSP430X architecture, Reference function: WriteMem_430X

Set CPU to stopped state (HaltCPU)		
ClrTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x2408)	: To write a word in memory. For a byte, the value to shift is 0x2418.	
IR_SHIFT("IR_ADDR_16BIT")	Yes	
DR_SHIFT20("Address")		: Set desired address
IR_SHIFT("IR_DATA_TO_ADDR")		
DR_SHIFT16("Data")		: Send 16-bit Data
SetTCLK		
Write again?		
No		
ReleaseCPU should now be executed, returning the CPU to normal operation.		

- MSP430Xv2 architecture, Reference function: WriteMem_430Xv2

<i>CPU must be in the Full-Emulation-State before the following sequence.</i>	
ClrTCLK	Yes
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x0500) : To write a word in memory. For a byte, the value to shift is 0x0510.	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT20("Address") : Set desired address	
SetTCLK	
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16("Data") : Send 16-bit Data	
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x0501)	
SetTCLK	
ClrTCLK	
SetTCLK	
Write again?	
No	
<i>CPU is now again in Full-Emulation-State.</i>	

2.3.3.3 Quick Access of Memory Arrays

The JTAG communication implemented on the MSP430 also supports access to a memory array in a more efficient manner. The instruction `IR_DATA_QUICK` is used to accomplish this operation. The R/W signal selects whether a read or write access is to be performed. Before this instruction can be loaded into the JTAG IR register, the program counter (PC) of the target MSP430 CPU must be set to the desired memory starting address. After the `IR_DATA_QUICK` instruction is shifted into the IR register, the PC is incremented by two with each falling edge of `TCLK`, automatically pointing the PC to the next memory location. The `IR_DATA_QUICK` instruction allows setting the corresponding MDB to a desired value (write), or captures (reads) the MDB with a `DR_SHIFT16` operation. The MDB should be set when `TCLK` is low. On the next rising `TCLK` edge, the value on the MDB is written into the location addressed by the PC. To read a memory location, `TCLK` must be high before the `DR_SHIFT16` operation is executed.

2.3.3.3.1 Flow for Quick Read (All Memory Locations)

- Both MSP430 and MSP430X architecture, Reference function: `ReadMemQuick`

Set PC to start address – 4 (SetPC or SetPC_430X, depending on the architecture)		
Switch CPU to stopped state (HaltCPU)		
ClrTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x2409)	: Set RW to read	
IR_SHIFT("IR_DATA_QUICK")		
SetTCLK	Yes	
DR_SHIFT16(0x0000)		: Memory value shifted out on TDO
ClrTCLK		: Auto-increments PC
Read From Next Address?		
No		
ReleaseCPU should now be executed, returning the CPU to normal operation. Reset the PC of the target CPU if needed (SetPC).		

- MSP430Xv2 architecture, Reference function: `ReadMem_430Xv2`

<i>CPU must be in the Full-Emulation-State before the following sequence.</i>		
<i>Set PC to start address (SetPC_430Xv2)</i>		
SetTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x0501)	: Set RW to read	
IR_SHIFT("IR_ADDR_CAPTURE")		
IR_SHIFT("IR_DATA_QUICK")		
SetTCLK	Yes	
ClrTCLK		: Auto-increments PC
DR_SHIFT16(0x0000)		: Memory value shifted out on TDO
Read From Next Address?		
No		
<i>Get CPU in Full-Emulation-State.</i>		

Note

For the MSP430F5xx and MSP430F6xx families, quick memory access must be used with care because the PC already points to one address ahead of the actual address to be read. This can lead to security access violations, especially at the end of a physical memory block.

2.3.3.3.2 Flow for Quick Write

- Both MSP430 and MSP430X architecture, Reference function: WriteMemQuick

Set PC to start address – 4 (SetPC)		
Switch CPU to stopped state (HaltCPU)		
ClrTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x2408)	Yes	
IR_SHIFT("IR_DATA_QUICK")		
DR_SHIFT16("Data")		
SetTCLK		
ClrTCLK		
Write To Next Address?		
No		
ReleaseCPU should now be executed, returning the CPU to normal operation. Reset the PC of the target CPU if needed (SetPC).		

- MSP430Xv2 architecture

For the MSP430Xv2 architecture in both flash and FRAM devices, there is no specific implementation of a quick write operation. The available implementation in the REP430 firmware is to call the WriteMem_430Xv2() function to write word by word into the specified memory.

2.3.4 Programming the Flash Memory (Using the Onboard Flash Controller)

2.3.4.1 Function Reference for 1xx, 2xx, 4xx Families

Reference function: WriteFLASH

This section describes one method available to program the flash memory module in an MSP430 device. It uses the same procedure that user-defined application software would use, which would be programmed into a production-equipment MSP430 device. Nonconsecutive flash memory addressing is supported.

This programming method requires a TCLK frequency of 350 kHz \pm 100 kHz while the erase or programming cycle is being executed. The frequency that must be applied to SBWTCK in Spy-Bi-Wire mode is the same frequency that is applied to TCK in 4-wire mode.

For more information on the flash controller timing, see the corresponding MSP430 user's guide and device-specific data sheet. [Table 2-9](#) shows the required minimum number of TCLK cycles, depending on the action performed on the flash (for FCTL2 register bits 0 to 7 = 0x40 as defined in the MSP430 user's guide).

Table 2-9. Erase and Program Minimum TCLK Clock Cycles

Flash Action	Minimum TCLK Count
Segment erase	4820 (default) or 9628 (MSP430ixx family)
Mass erase	5300 to 10600 ⁽¹⁾
Program word	35

- (1) MSP430 device dependent, see device-specific data sheet. See [Section 2.3.5](#) for more details.

The following JTAG communication flow shows programming of the MSP430 flash memory using the onboard flash controller. In this implementation, 16-bit words are programmed into the main flash memory area. To program bytes, the BYTE bit in the JTAG CNTRL_SIG register must be set high while in programming mode. StartAddr is the starting address of the flash memory array to be programmed.

Switch CPU to stopped state (HaltCPU)		
ClrTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x2408) : Set RW to Write		
IR_SHIFT("IR_ADDR_16BIT")		
DR_SHIFT16(0x0128) ⁽¹⁾ : Point to FCTL1 Address		
IR_SHIFT("IR_DATA_TO_ADDR")		
DR_SHIFT16(0xA540) : Enable FLASH Write Access		
SetTCLK		
ClrTCLK		
IR_SHIFT("IR_ADDR_16BIT")		
DR_SHIFT16(0x012A) ⁽¹⁾ : Point to FCTL2 Address		
IR_SHIFT("IR_DATA_TO_ADDR")		
DR_SHIFT16(0xA540) : Source is MCLK, divider by 1		
SetTCLK		
ClrTCLK		
IR_SHIFT("IR_ADDR_16BIT")		
DR_SHIFT16(0x012C) ⁽¹⁾ : Point to FCTL3 Address		
IR_SHIFT("IR_DATA_TO_ADDR")		
DR_SHIFT16(0xA500) ⁽²⁾ : Clear FCTL3 Register		
SetTCLK		
ClrTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x2408) : Set RW to Write		Yes
IR_SHIFT("IR_ADDR_16BIT")		
DR_SHIFT16("Address") ⁽¹⁾ : Set Address for Write		
IR_SHIFT("IR_DATA_TO_ADDR")		
DR_SHIFT16("Data") : Set Data for Write		
SetTCLK		
ClrTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x2409) : Set RW to Read		
SetTCLK		
ClrTCLK		
Repeat 35 times ⁽³⁾		
Write Another Flash Address?		
No		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x2408) : Set RW to Write		
IR_SHIFT("IR_ADDR_16BIT")		
DR_SHIFT16(0x0128) ⁽¹⁾ : Point to FCTL1 Address		
IR_SHIFT("IR_DATA_TO_ADDR")		
DR_SHIFT16(0xA500) : Disable FLASH Write Access		
SetTCLK		
ClrTCLK		
IR_SHIFT("IR_ADDR_16BIT")		

DR_SHIFT16(0x012C) ⁽¹⁾	: Point to FCTL3 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA510) ⁽⁴⁾	: Set LOCK bit in FCTL3
SetTCLK	
<i>ReleaseCPU should now be executed, returning the CPU to normal operation.</i>	

- (1) Replace with DR_SHIFT20("Address") when programming an MSP430X architecture device.
- (2) Substitute 0xA540 for 2xx devices for Info-Segment A programming.
- (3) Correct timing required. Must meet minimum and maximum TCLK frequency requirement of 350 kHz \pm 100 kHz.
- (4) Substitute 0xA550 for 2xx devices for Info-Segment A programming.

2.3.4.2 Function Reference for 5xx and 6xx Families

Because the 5xx and 6xx devices have a dedicated timing generator available on chip, flash access is significantly easier compared to the other MSP430 families. There is no need for the user to ensure a certain erase or program frequency on the TCLK signal. All timings that are required for memory erase and write access are generated automatically.

The basis for the following description is that the flash memory access operation can be initiated from within RAM, as described in the relevant [MSP430F5xx and MSP430F6xx Family User's Guide](#) chapters. This document describes how to load an appropriate code in the target device RAM and how to control the correct execution of the code using the JTAG interface. Controlling the execution of the target code can be done by releasing the device from JTAG control. Releasing the device from JTAG control makes the CPU execute the program code in free running mode. After the desired operation is finished, the device must be taken under JTAG control again.

This method has advantages and disadvantages. Having a free-running device can increase flash programming speed to its upper limit; it requires a polling mechanism through JTAG to retrieve the current target device state. On the other hand, such a polling mechanism is not suitable for systems in which more than one target device is accessed in parallel. As all targets would not run at exactly the same frequency, keeping them under JTAG control would be the recommended approach for a parallel access system. This is not implemented in the REP430 firmware.

Exchanging information between the target devices' CPU and JTAG (for example, for device state polling purposes) uses a new feature of the 5xx JTAG implementation: The JTAG mailbox system. The idea behind the JTAG mailbox system is to have a direct interface to the CPU during debugging, programming, and test that is identical for all devices in this family and that uses few or no user application resources (refer to the [MSP430F5xx and MSP430F6xx Family User's Guide System Resets, Interrupts and Operating Modes, System Control Module \(SYS\)](#) chapter).

[Figure 2-17](#) shows the general flow required to perform flash memory operations on 5xx and 6xx devices through the JTAG interface. The term Flash-Access-Code stands for an appropriate executable MSP430 code that can be used to perform the flash access operation. The following sections use the term Flash-Write-Code for code that is used to program the flash memory and Flash-Erase-Code for code that is used to erase the flash memory.

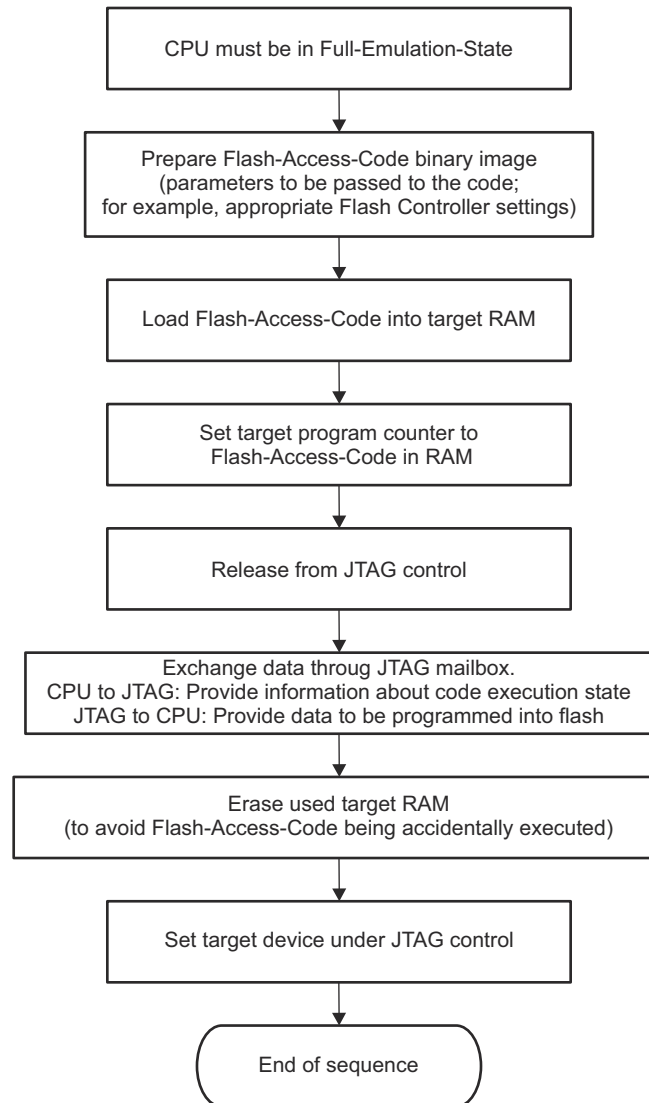


Figure 2-17. Accessing Flash Memory

Reference function: WriteFLASH_430Xv2

This section describes one method to program the flash memory subsequently with 16-bit word data by executing an appropriate Flash-Write-Code in RAM and providing the data to the CPU through the JTAG mailbox system. The provided source code example includes a Flash-Write-Code example that has the capability to be parameterized in binary state. [Figure 2-18](#) shows a generic map of the binary image of the flash access codes provided with this document.

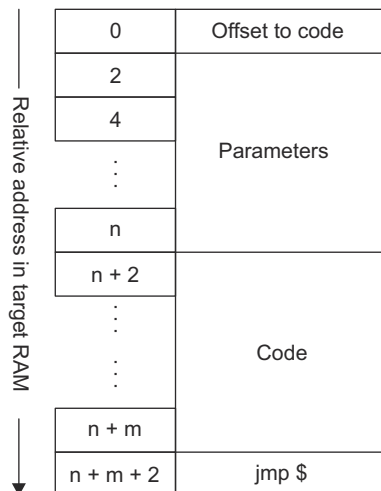


Figure 2-18. Flash Access Code Binary Image Map

- The code is position independent.
- The first address holds an offset value relative to the actual program code start address. The current address plus the offset value results in the value that must be assigned to the Program Counter before starting execution of the code.
- Space for code specific parameters.
- Actual program code.
- Endless loop at the end.

The Flash-Write-Code in particular takes the following parameters:

- *StartAddr*: First address in target memory to be written to
- *Length*: Number of 16-bit words to be written
- *FCTL3*: The value to be written into FCTL3 of the flash controller module (basically to define whether LOCKA should be set or not)

When executing the Flash-Write-Code, the data to be programmed into the target flash memory must be provided through the JTAG mailbox system. The following sequence shows how this is established.

- Released from JTAG control

Target device is released from JTAG control (free running)		
IR_SHIFT("IR_JB_EXCHANGE")		Yes
DR_SHIFT16(0x0000)	No	
Bit 0 of TDOWord = 1 ?		
Yes		
DR_SHIFT16(0x0001)	: Send input request to JTAG mailbox	
DR_SHIFT16("Data")	: Shift 16 bit word into JTAG mailbox	
Write Another Flash Address?		
No		
Get target device in Full-Emulation-State		

2.3.5 Erasing the Flash Memory (Using the Onboard Flash Controller)

2.3.5.1 Function Reference for 1xx, 2xx, 4xx Families

Reference function: EraseFLASH

This section describes how to erase one segment of flash memory (ERASE_SGMT), how to erase the device main memory (ERASE_MAIN), and how to perform an erase of the complete flash memory address range including main and info flash segments (ERASE_MASS). This method requires the user to provide a TCLK signal at a frequency of 350 kHz \pm 100 kHz while the erase cycle is being executed, as is also the case when programming the flash memory. The following tables show the segment and mass erase flows, respectively, and the minimum number of TCLK cycles required by the flash controller to perform each action (FCTL2 register bits 0 to 7 = 0x40).

2.3.5.1.1 Flow to Erase a Flash Memory Segment

Switch CPU to stopped state (HaltCPU)	
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2408)	: Set RW to Write
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x0128) ⁽¹⁾	: Point to FCTL1 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA502)	: Enable FLASH segment erase
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x012A) ⁽¹⁾	: Point to FCTL2 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA540)	: Source is MCLK, divider by 1
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x012C) ⁽¹⁾	: Point to FCTL3 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500) ⁽⁴⁾	: Clear FCTL3 Register
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16("EraseAddr") ⁽¹⁾	: Set Address for Erase ⁽²⁾
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0x55AA)	: Write Dummy Data for Erase Start
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2409)	: Set RW to Read
SetTCLK	Repeat 4819 times ⁽³⁾
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2408)	: Set RW to Write
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x0128) ⁽¹⁾	: Point to FCTL1 Address

IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500)	: Disable FLASH Erase
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x012C) ⁽¹⁾	: Point to FCTL3 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA510) ⁽⁴⁾	: Set LOCK bit in FCTL3
SetTCLK	
<i>Release CPU should now be executed, returning the CPU to normal operation.</i>	

- (1) Replace with DR_SHIFT20("Address") when programming an MSP430X architecture device.
- (2) The EraseAddr parameter is the address pointing to the flash memory segment to be erased.
- (3) Correct timing required. Must meet minimum and maximum TCLK frequency requirement of 350 kHz \pm 100 kHz.
- (4) Substitute 0xA540 for 2xx devices for Info-Segment A programming.

2.3.5.1.2 Flow to Erase the Entire Flash Address Space (Mass Erase)

In addition to the TCLK signal at a frequency of 350 kHz \pm 100 kHz (used for the flash timing generator, see data sheet parameter f_{FTG}), two more data sheet parameters must be taken into account when using the described method to perform a mass or main memory erase. The first is $t_{CMErase}$ (cumulative mass erase time) and the second is $t_{Mass Erase}$ (mass erase time). Two different specification combinations of these parameters are currently implemented in dedicated MSP430 devices. [Table 2-10](#) shows an overview of the parameters (assuming a maximum TCLK frequency of 450 KHz).

Table 2-10. Flash Memory Parameters ($f_{FTG} = 450$ kHz)

Implementation	$t_{CMErase}$	$t_{Mass Erase}$	Mass Erase Duration Generated by the Flash Timing Generator
1	200 ms	$5300 \times t_{FTG}$	11.1 ms
2	20 ms	$10600 \times t_{FTG}$	20 ms

For implementation 1, to assure the recommended 200-ms erase time to safely erase the flash memory space, 5300 TCLK cycles are transmitted to the target MSP430 device and repeated 19 times. With implementation 2, the following sequence needs to be performed only once.

Note

MSP430F2xx devices have four information memory segments of 64 bytes each. Segment INFOA (see the [MSP430F2xx Family User's Guide](#) for more information) is a lockable flash information segment and contains important calibration data for the MSP430F2xx clock system (DCO) unique to the given device programmed at production test. The remaining three information memory segments (INFOB, INFOC, and INFOD) cannot be erased by a mass erase operation as long as INFOA is locked. INFOB, INFOC, and INFOD can be erased segment by segment, independent of the lock setting for INFOA. Unlocking INFOA allows performing the mass erase operation.

<i>Switch CPU to stopped state (HaltCPU)</i>	
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2408)	: set RW to write
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x0128) ⁽¹⁾	: FCTL1 address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA506)	: Enable FLASH mass erase
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x012A) ⁽¹⁾	: FCTL2 address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA540)	: Source is MCLK and divider is 0
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x012C) ⁽¹⁾	: FCTL3 address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500) ⁽⁴⁾	: Clear FCTL3 register
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16("EraseAddr") ⁽¹⁾	: Set address for erase ⁽²⁾
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0x55AA)	: Write dummy data for erase start
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2409)	: set RW to read
SetTCLK	Perform 10600 or 5300 times ⁽³⁾
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2408)	: set RW to write
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x0128) ⁽¹⁾	: FCTL1 address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500)	: Disable FLASH erase
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x012C) ⁽¹⁾	: Point to FCTL3 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA510) ⁽⁴⁾	: Set LOCK bit in FCTL3
SetTCLK	
<i>ReleaseCPU should now be executed, returning the CPU to normal operation.</i>	

Perform once or
Repeat 19 times⁽³⁾

Perform once or
Repeat 19 times⁽³⁾

(1) Replace with DR_SHIFT20("Address") when programming an MSP430X architecture device.

- (2) The EraseAddr parameter is the address pointing to the flash memory segment to be erased. For mass erase, an even value in the address range of the information memory should be used. For main memory erase, an even value in the address range of the main memory should be used.
- (3) Correct timing required. Must meet minimum and maximum TCLK frequency requirement of 350 kHz \pm 100 kHz.
- (4) Substitute 0xA540 for 2xx devices for INFO Segment A programming.

2.3.5.2 Function Reference for 5xx and 6xx Families

Reference function: EraseFLASH_430Xv2 and EraseFLASH_430Xv2_wo_release

Similar to what is used for flash programming also the erase operation is handled by a executable code loaded into RAM of the target device. The Flash-Erase-Code provided with this document takes the following parameters.

- **EraseAddr:** Valid flash memory address used for the dummy write that triggers the Flash memory operation
- **EraseMode:** The value to be written into FCTL1 of the flash controller module (basically to define the erase mode by the MERAS and ERASE bits)
- **FCTL3:** The value to be written into FCTL3 of the flash controller module (basically to define whether LOCKA should be set or not)

The Flash-Erase-Code can be executed either under JTAG control or in free-running mode. Similar to what is described in [Section 2.3.4.2](#), the JTAG mailbox system is used to retrieve the current execution state of the Flash-Erase-Code in the target device.

Note

Single stepping over a flash memory segment erase operation is not supported. Any attempt might result in corruption of the program counter.

2.3.6 Reading From Flash Memory

Reference function: ReadMem or ReadMemQuick

The flash memory can be read using the normal memory read flow given earlier for non-flash memory addresses. The quick access method can also be used to read flash memory.

2.3.7 Verifying the Target Memory

Reference function: VerifyMem

Verification is performed using a pseudo signature analysis (PSA) algorithm, which is built into the MSP430 JTAG logic and executes in approximately 23 ms/4KB.

- Both MSP430 and MSP430X architecture, Reference functions: VerifyPSA, VerifyPSA_430X

ExecutePOR	
PSA_CRC = StartAddr-2	
Device has EnhancedVerify (see Table 2-14)	Device does not have EnhancedVerify (see Table 2-14)
SetPC (StartAddr-4)	SetPC (StartAddr-2)
HaltCPU	SetTCLK
ClrTCLK	ClrTCLK
IR_SHIFT ("IR_DATA_16BIT")	
DR_SHIFT16 (StartAddr-2)	
IR_SHIFT ("IR_DATA_PSA")	

Calculate the PSA value:		
PSA_CRC & 0x8000 == 0x8000	PSA_CRC & 0x8000 != 0x8000	
XOR PSA_CRC with 0x0805	PSA_CRC <= 0x1	
PSA_CRC <= 0x1		
PSA_CRC = 0x0001		
Actual Verify or EraseCheck?		
Verify	EraseCheck	
XOR PSA_CRC with next word from reference data	XOR PSA_CRC with 0xFFFF	
Clock through the PSA:		
SetTCLK	No	
ClrTCK		
SetTMS		
SetTCK : Select DR-Scan		
ClrTCK		
ClrTMS		
SetTCK : Capture-DR		
ClrTCK		
SetTCK : Shift-DR		
ClrTCK		
SetTMS		
SetTCK : Exit-DR		
ClrTCK		
SetTCK		
ClrTMS		
ClrTCK		
SetTCK		
ClrTCLK		
Reached end of memory area to verify?		
Yes		
IR_SHIFT("IR_SHIFT_OUT_PSA")		
DR_SHIFT16(0x0000) : Read out the PSA value		
SetTCLK		
Call ReleaseCPU() here if device has Enhanced Verify feature (see Table 2-14)		
Compare shifted out PSA value with PSA_CRC		

- MSP430Xv2 architecture (including FRxx devices), Reference function: VerifyPSA_430Xv2

ExecutePOR	
PSA_CRC = StartAddr-2	
SetPC(StartAddr)	
SetTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x0501)	
IR_SHIFT("IR_DATA_16BIT")	
DR_SHIFT16(StartAddr-2)	
IR_SHIFT("IR_DATA_PSA")	
Calculate the PSA value:	
PSA_CRC & 0x8000 == 0x8000	PSA_CRC & 0x8000 != 0x8000
XOR PSA_CRC with 0x0805	PSA_CRC <=<= 0x1
PSA_CRC <=<= 0x1	
PSA_CRC = 0x0001	
Actual Verify or EraseCheck?	
Verify	EraseCheck
XOR PSA_CRC with next word from reference data	XOR PSA_CRC with 0xFFFF
Clock through the PSA:	
ClrTCLK	
ClrTCK	
SetTMS	
SetTCK : Select DR-Scan	
ClrTCK	
ClrTMS	
SetTCK : Capture-DR	
ClrTCK	
SetTCK : Shift-DR	
ClrTCK	
SetTMS	
SetTCK : Exit-DR	
ClrTCK	
SetTCK	
ClrTMS	
ClrTCK	
SetTCK	
SetTCLK	
Reached end of memory area to verify?	
Yes	
IR_SHIFT("IR_SHIFT_OUT_PSA")	
DR_SHIFT16(0x0000) : Read out the PSA value	

No

Call `ExecutePOR_430Xv2()` here if device has Enhanced Verify feature (see [Table 2-14](#))

Compare shifted out PSA value with `PSA_CRC`

2.3.8 FRAM Memory Technology

FRAM memory is a non-volatile memory that behaves like standard SRAM. FRAM can be read in a similar fashion to SRAM and has no special requirements. Similarly, any writes to unprotected segments can be written in the same fashion as SRAM.

An FRAM read is destructive and, therefore, always requires a write back to the same memory location with the information read. This write back is part of the FRAM memory controller itself and requires no user interaction. These write backs are different from the normal write access by application code or programming tool.

2.3.8.1 Writing and Reading FRAM

Referring to the statement above, the `WriteMem_430Xv2` and `ReadMem_430Xv2` functions can be used to read and write FRAM. `WriteMemQuick_430Xv2` and `ReadMemQuick_430Xv2` are applicable as well.

2.3.8.2 Erasing FRAM

One advantage of FRAM is that erases are not required before writing memory. If a "flash-style" erase needs to be executed, a dummy erase can be done by using the write function to write `0xFF` to all locations in the memory.

The Replicator430FR example project offers two different reference implementations. For segment erases in FRAM, refer to the function `EraseFRAM_430Xv2`. An FRAM mass erase is done using the flow described in `EraseFRAMViaBootCode_430Xv2`.

2.4 JTAG Access Protection

There are various ways of protecting memory access to an MSP device. [Table 2-11](#) is an overview of all available methods and the applicable device families. All mechanisms directly related to the JTAG interface are described in the referenced sections.

For the sake of completeness, this list also includes the lock mechanisms of the BSL interface. See the documents listed in the *Reference* column for a detailed description of these features and instructions for their use.

Table 2-11. Overview Of Memory Protection Mechanisms

Protection Mode	Applicable Devices	Permanently Locked	Description	Unlocking Method	Reference
JTAG fuse	Flash 1xx, 2xx, and 4xx families (except FRxx and i20xx devices)	Yes	Applying a high voltage to the TEST pin (TDI pin for devices without TEST pin) blows an actual physical polyfuse and renders the JTAG interface unusable	none	Section 2.4.1
Combination of JTAG "soft"-fuse (e-fuse) and disabling BSL	5xx, 6xx, and FRxx families	Yes	To prevent unlocking the target memory by either JTAG or BSL interface, both must be disabled.	none	Section 2.4.2.1 , Section 2.4.2.2 , and MSP430™ Flash Devices Bootloader (BSL) User's Guide

Table 2-11. Overview Of Memory Protection Mechanisms (continued)

Protection Mode	Applicable Devices	Permanently Locked	Description	Unlocking Method	Reference
JTAG lock without password	Flash 5xx and 6xx families	No	Writing a pattern other than 0x00000000 or 0xFFFFFFFF at the address 0x17FC locks the JTAG interface	Resetting the lock key in BSL memory using the BSL interface	Section 2.4.2.1
	FR2xx and FR4xx devices	No	Writing the pattern 0x55555555 at the address 0xFF80 locks the JTAG interface (solution implemented in the REP430FR firmware). Note NOTE: For FR2xx and FR4xx devices, it is also possible to lock the JTAG access by writing a pattern other than 0x00000000 and 0xFFFFFFFF at the address 0xFF80	1. Using the BSL to erase the main memory 2. Using a special erase command (User_Code_Erase 01A1A) applied through the JTAG mailbox. The function to use in the REP430FR firmware is EraseFRAMViaBootCode_430Xv2().	Section 2.4.2.2
	FR5xx and FR6xx devices	No		Using the BSL to erase the main memory.	
JTAG lock with password	FR5xx and FR6xx devices	No	Writing a JTAG lock signature (0xAAAA), password length and the password itself to 0xFF80 and the following memory segment (password might extend up to the reset vector at 0xFFFE) locks the JTAG interface	Using the JTAG mailbox in combination with the device BootCode, the given password is compared to the applied password.	See the device family user's guide for details on how to set the password. Unlocking: Section 2.4.4.1
Memory Protection Unit (MPU)	FRxx devices	No	User can specify up to three memory segments with different access rights (read, write, execute)	Brown-Out-Reset (BOR) will reset all MPU settings	See the device family user's guide for details on how to set up the MPU. Unlocking: Section 2.4.5
IP-Encapsulation	FR5xx and FR6xx families (except FR57xx) devices	No	User can specify start and end address of one memory segment to be protected from any read, write, or execute access	Performing a "total erase" ("erase main, information and IP-protected area on connect" in CCS) will erase the entire memory and reset IPE settings	See the device family user's guide for details on how to set up the IP Encapsulation. Unlocking: Section 2.4.6
BSL Password protection	All devices with BSL	No	BSL is protected by default against reading and writing	BSL password equals interrupt vector table content – can be provided by reduced BSL command-set in locked mode	MSP430™ Flash Devices Bootloader (BSL) User's Guide

Table 2-11. Overview Of Memory Protection Mechanisms (continued)

Protection Mode	Applicable Devices	Permanently Locked	Description	Unlocking Method	Reference
Disable BSL	Flash 1xx, 2xx, and 4xx families	No	BSL interface is disabled by writing a lock signature to a certain address (depending on BSL implementation)	Can be unlocked using JTAG interface (see device family user guides for details)	MSP430™ Flash Devices Bootloader (BSL) User's Guide
	Flash 5xx and 6xx families	No	BSL interface is locked by clearing certain signatures at the end of the BSL memory (0x17F6 and 0x17F4)	Signatures can be restored by erasing and reprogramming the Flash segment containing it using the JTAG interface	MSP430F5xx and MSP430F6xx Family User's Guide
	FRxx devices	No	BSL interface is locked by writing a given BSL lock signature at 0xFF84 and 0xFF86 in the main memory	Access to the BSL interface can be re-enabled by overwriting the lock signature using the JTAG interface	MSP430FR58xx, MSP430FR59xx, and MSP430FR6xx Family User's Guide
Memory protection by customer-written startup code (SUC)	MSP430i20xx family (MSP430i2040)	Depends on SUC implementation	User can write startup code to set up JTAG access protection	Depends on SUC implementation	See the MSP430i2xx Family User's Guide for details

2.4.1 Burning the JTAG Fuse - Function Reference for 1xx, 2xx, 4xx Families

Two similar methods are described and implemented, depending on the target MSP430 device family.

All devices having a TEST pin use this input to apply the programming voltage, V_{PP} . As previously described, these devices have shared-function JTAG interface pins. The higher pin count MSP430 devices with dedicated JTAG interface pins use the TDI pin for fuse programming.

Devices with a TEST pin:

Table 2-12. MSP430 Device JTAG Interface (Shared Pins)

Pin	Direction	Use
P1.5/TMS	IN	Signal to control JTAG state machine
P1.4/TCK	IN	JTAG clock input
P1.6/TDI	IN	JTAG data input/TCLK input
P1.7/TDO	OUT	JTAG data output
TEST	IN	Logic high enables JTAG communication; V_{PP} input while programming JTAG fuse

Devices without a TEST pin (dedicated JTAG pins):

Table 2-13. MSP430 Device Dedicated JTAG Interface

Pin	Direction	Use
TMS	IN	Signal to control JTAG state machine
TCK	IN	JTAG clock input
TDI	IN	JTAG data input/TCLK input; V_{PP} input while programming JTAG fuse
TDO	OUT/IN	JTAG data output; TDI input while programming JTAG fuse

Note

The value of V_{PP} required for fuse programming can be found in the corresponding target device data sheet. For existing flash devices, the required voltage for V_{PP} is 6.5 V \pm 0.5 V.

2.4.1.1 Standard 4-Wire JTAG

Reference function: BlowFuse

2.4.1.1.1 Fuse-Programming Voltage on TDI Pin (Dedicated JTAG Pin Devices Only)

When the fuse is being programmed, V_{PP} is applied through the TDI input. Communication data that is normally sent on TDI is sent through TDO during this mode. (Table 2-13 describes the dual functionality for the TDI and TDO pins.) The settling time of the V_{PP} source must be taken into account when generating the proper timing to blow the fuse. The following flow details the fuse-programming sequence built into the BlowFuse function.

IR_SHIFT("IR_CNTRL_SIG_16BIT")
DR_SHIFT_IN(0x7201) : Configure TDO as TDI
<i>TDI signal releases to target, TDI is now provided on TDO.</i>
IR_SHIFT("IR_PREPARE_BLOW") (through TDO pin)
MsDelay(1) : Delay for 1ms
<i>Connect V_{PP} to TDI pin</i>
<i>Wait until V_{PP} input has settled (depends on V_{PP} source)</i>
IR_SHIFT("IR_EX_BLOW") : Sent to target on TDO
MsDelay(1) : Delay for 1ms
<i>Remove V_{PP} from TDI pin</i>
<i>Switch TDI pin back to TDI function and reset the JTAG state machine (ResetTAP)</i>

2.4.1.1.2 Fuse-Programming Voltage On TEST Pin

The same method is used to program the fuse for the TEST pin MSP430 devices, with the exception that the fuse-blow voltage, V_{PP} , is now applied to the TEST input pin.

IR_SHIFT("IR_PREPARE_BLOW")
MsDelay(1) : Delay for 1ms
<i>Connect V_{PP} to TEST pin</i>
<i>Wait until V_{PP} input has settled (depends on V_{PP} source)</i>
IR_SHIFT("IR_EX_BLOW")
MsDelay(1) : Delay for 1ms
<i>Remove V_{PP} from TEST pin</i>
<i>Reset the JTAG state machine (ResetTAP)</i>

2.4.1.2 Fuse-Programming Voltage Using SBW

Reference function: BlowFuse_sbv

In SBW mode, the TEST/SBWTDCK pin is used to apply fuse-blow voltage V_{PP} . The required timing sequence is shown in Figure 2-19. The actual fuse programming happens in the Run-Test/Idle state of the TAP controller. After the IR_EX_BLOW instruction is shifted in through SBW, one more TMS_SLOT must be performed. Then a stable V_{PP} must be applied to SBWTDCK. Taking SBWTDIO high as soon as V_{PP} has been settled blows the fuse. It is required that SBWTDIO is low on exit of the IR_EX_BLOW instruction shift.

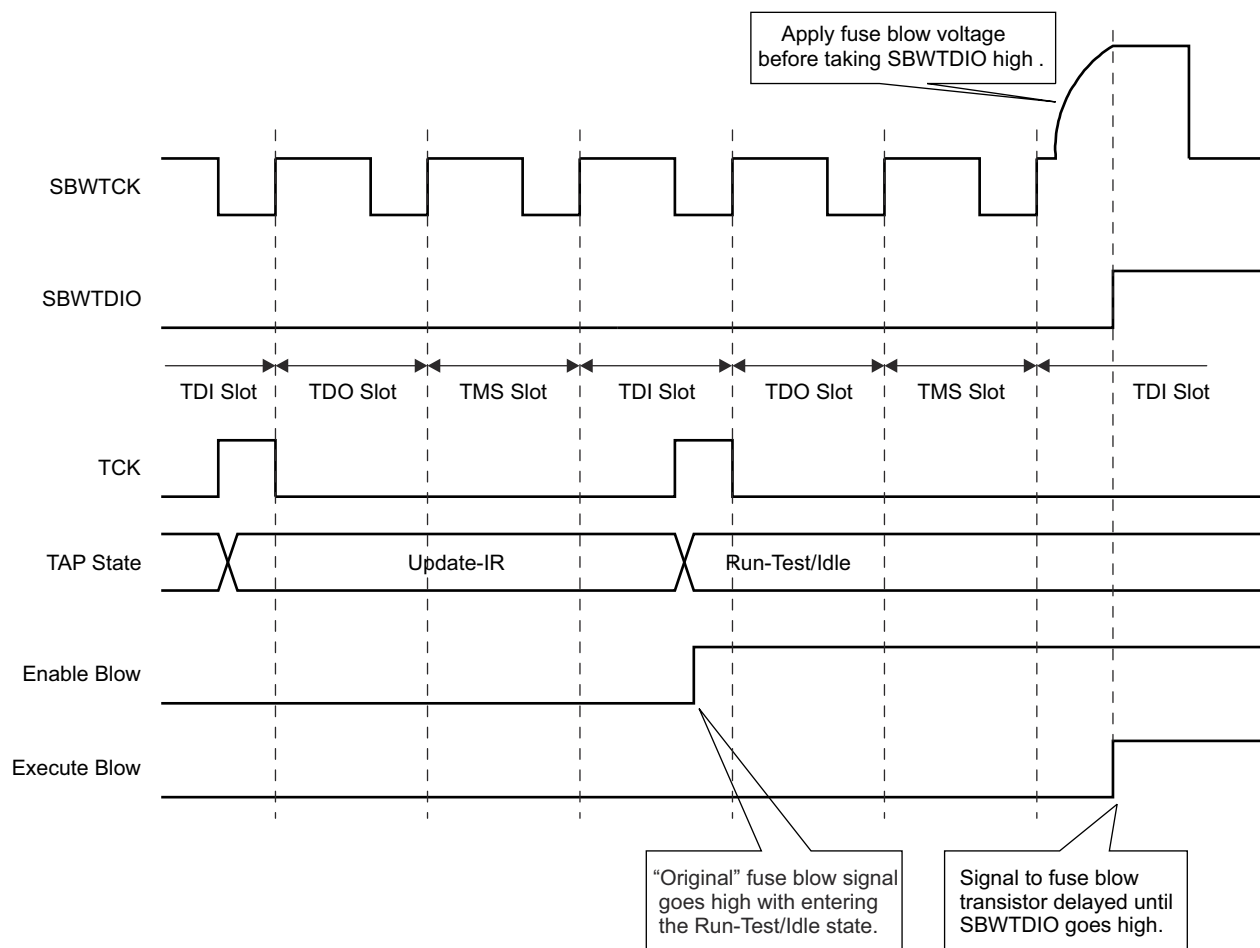


Figure 2-19. Fuse Blow Timing

2.4.2 Programming the JTAG Lock Key - Function Reference for 5xx, 6xx, and FRxx Families

2.4.2.1 Flash Memory Devices

Reference function: ProgramLockKey

Note

For the MSP430F5xx and MSP430F6xx families, it is NOT required to apply a special high voltage to the device's TEST pin.

In contrast to the 1xx, 2xx, 4xx families, which require special handling to burn the JTAG security fuse, the 5xx and 6xx families' JTAG is locked by programming a certain signature into the devices' flash memory at dedicated addresses. The JTAG security lock key resides at the end of the bootloader (BSL) memory at addresses 0x17FC to 0x17FF. Any value other than 0 or 0xFFFFFFFF programmed to these addresses irreversibly locks the JTAG interface. All of the 5xx and 6xx MSP430 devices come with a preprogrammed BSL (TI-BSL) code that, by default, protects itself from unintended erase and write access. This is done by setting the SYSBSLPE bit in the SYSBSLC register of the SYS module (see the [MSP430F5xx and MSP430F6xx Family User's Guide](#) SYS Module chapter for details). Because the JTAG security lock key resides in the BSL memory address range, appropriate action must be taken to unprotect the memory area before programming the protection key. This can be done by a regular memory write access as described in [Section 2.3.3.2](#) by writing directly to the SYSBSLC register address and setting the SYSBSLPE to 0. Afterward, the BSL memory behaves like regular flash memory and a JTAG lock key can be programmed at addresses 0x17FC to 0x17FF as described in [Section 2.3.4.2](#). A brownout reset (BOR) is required to activate the JTAG security protection during boot. The BOR can be issued as described in [Section 2.3.2.2.4](#). If the hardware setup does not allow performing a power cycle (for example, the battery is already soldered to the PCB) a BOR can also be generated by JTAG by writing into a dedicated

JTAG test data register. A BOR also resets the JTAG interface, which causes the device to be released from JTAG control.

2.4.2.2 FRAM Memory Devices

Reference function: ProgramLockKey

Note

For the MSP430FR5xx family, it is NOT required to apply a special high voltage to the device's TEST pin.

FRAM-based devices use a LockKey that is written into a special location to secure the device. The devices support two different levels of protection: "protected mode" for FR5xx and FR6xx devices only and "secured mode" for all FRAM devices.

In the protected mode, the application can define a password and protect the device with this password. The UnlockDevice function could be used to connect to the device by applying the correct password (see [Section 2.4.4](#) for detailed information). For general information about the password, see the [MSP430FR57xx Family User's Guide](#).

In the secured mode, the device cannot be accessed through JTAG. To enable the secured mode, write 0x55555555 to the memory location 0xFF80. After writing the password, a BOR is required to enable the security fuse.

2.4.3 Testing for a Successfully Protected Device

Reference function: IsFuseBlown, IsLockKeyProgrammed

After the JTAG Fuse is burned (for 1xx, 2xx, or 4xx devices) or the JTAG Lock Key is programmed (for 5xx or 6xx devices) and a RESET (by the JTAG ExecutePOR command or the $\overline{\text{RST}}$ /NMI pin in hardware) has been issued, the only JTAG function that is available on the target MSP430 is BYPASS. When the target is in BYPASS, data sent from host to target is delayed by one TCK pulse and then output on TDO, where it can be received by other devices downstream of the target MSP430.

To test a device for being protected, access to any JTAG data register can be attempted. In the following communication sequence, the JTAG CNTRL_SIG register is accessed.

<i>Initialize JTAG access (ResetTAP)</i>	
IR_SHIFT("IR_CNTRL_SIG_CAPTURE")	
DR_SHIFT16(0xAAAA)	
<i>Is TDO output value = 0x5555?</i>	
<i>Yes: Device IS protected</i>	<i>No: Device NOT protected</i>

2.4.4 Unlocking an FRAM Device in Protected and Secured Modes

2.4.4.1 FR5xx and FR6xx Devices

Devices that support protection by user password can be unlocked by providing the correct password. To unlock the device, the JTAG mailbox is used in combination with the device BootCode.

To activate the password unlock mechanism, the password exchange request (0x1E1E) must be applied to the device (see the detailed sequence diagram below).

Note

After executing the password exchange request, there is a timeslot of only 1.2 s to apply the correct password. If the password is not applied during this time frame or if the password is wrong, a BOR event is executed by the device.

<i>Initialize JTAG access (ResetTAP) and hold device in RESET</i>		
IR_Shift(IR_JMB_EXCHANGE)	:do JTAG mailbox exchange request	
DR_Shift16(0x0011)	:configure JTAG mailbox to 32-bit mode	
DR_Shift16(0xA55A)	:send device after BootCode execution into LPM4	
DR_Shift16(0x1E1E)	:perform password exchange request	
<i>Stop JTAG (release and run device)</i>		
<i>Wait until BootCode is executed and device is in LPM4</i>		
<i>Initialize JTAG access (ResetTAP)</i>		
IR_Shift(IR_JMB_EXCHANGE)	:do JTAG mailbox exchange request	
DR_Shift16(0x0001)	:configure JTAG mailbox to 16-bit mode	
DR_Shift16(password[i])	:configure JTAG mailbox to 16-bit mode	Repeat for password length

2.4.4.2 FR4xx and FR2xx Devices

The FR4xx and FR2xx device families do not support JTAG protection by a user-defined password. These device families have only a simple JTAG lock mechanism. If a JTAG lock signature is written to memory address 0xFF80, JTAG access is locked. To unlock these device families, a special erase command (User_Code_Erase 0x1A1A) must be applied through the JTAG mailbox (see the following detailed sequence diagram).

<i>Restart device, initialize JTAG and feed in Erase command while device is in reset</i>	
ClrTST()	Restart device
ClrRST()	Restart device
StartJtag(RSTLOW)	Restart JTAG – Keep RST low device do not start
ResetTAP()	Reset JTAG TAP state machine
IR_Shift(IR_JMB_EXCHANGE)	Do JTAG mailbox exchange request
DR_Shift(0xA55A)	Send device after BootCode execution into LPM4
DR_Shift(0x1A1A)	Send device UserCode erase command
<i>Stop JTAG (release device to run)</i>	
<i>Wait until BootCode is executed and device is in LPM4</i>	
<i>Initialize JTAG access (Rest Tap)</i>	
<i>Get back JTAG control calling SyncJtag_AssertPor() function</i>	

2.4.5 Memory Protection Unit Handling

If a device has a Memory Protection Unit (MPU), it must be disabled before erasing or writing memory. The MPU can separate the memory into different parts. Each part can have different access rights such as READ, WRITE, and EXECUTE. A nondisabled MPU could cause a write or erase to be incomplete. Furthermore, the MPU settings, which are stored in a register, could be locked. To disable this register lock, a BOR must be performed. The sequence diagram below shows how to disable the MPU. The example code can be found in the [associated zip file](#) in the function DisableMpu430Xv2().

Is MPU enabled (read register at 0x05A0)?			
YES			NO
Are MPU settings locked?			Return
YES		NO	
IR_Shift(IR_JMB_EXCHANGE)	:do JTAG mailbox exchange request	Write 0xA500 at address 0x05A0	:disable MPU
DR_Shift16(0x0001)	:configure mailbox to 16-bit mode	Return	:MPU is disabled
DR_Shift16(0xA55A)	:send device after BootCode execution into LPM4		
Execute BOR			
Initialize JTAG access (ResetTAP)			
Synchronize JTAG and assert POR			
Write 0xA500 at address 0x05A0	:disable MPU		
Return	:MPU is disabled		

2.4.6 Intellectual Property Encapsulation (IPE)

In addition to the Memory Protection Unit (MPU), FR59xx, FR58xx, FR69xx, and FR68xx devices feature the IP Encapsulation (IPE) module. The IPE module allows the user to specify a single memory segment to be protected against external access. See the [MSP430FR58xx, MSP430FR59xx, and MSP430FR6xx Family User's Guide](#) for information on how to invoke the IP Encapsulation feature.

To remove IPE settings, the target memory must be completely erased. Because a regular mass erase does not affect the IPE protected area, a special erase sequence, which is detailed in the following diagram below, must be performed to reset the IPE settings.

Restart device, initialize JTAG and feed in Erase command while device is in reset	
ClrTST()	Restart device
ClrRST()	Restart device
StartJtag(RSTLOW)	Restart JTAG – Keep RST low device does not start
ResetTAP()	Reset JTAG TAP state machine
IR_Shift(IR_JMB_EXCHANGE)	Do JTAG mailbox exchange request
DR_Shift(0xA55A)	Send device into LPM4 after BootCode execution
DR_Shift(0x1B1B)	Send Total Erase command
Stop JTAG (release device to run)	
Wait until BootCode is executed and device is in LPM4	
Initialize JTAG access (Rest Tap)	
Get back JTAG control calling SyncJtag_AssertPor() function	

2.4.7 FRAM Write Protection

On MSP430FR2xx and MSP430FR4xx devices, the FRAM is protected against unwanted access. To enable FRAM access, clear bits DFWP and PFWP in the SYSCFG0 register. On some devices from these families, the SYSCFG0 register is password protected. See the device family user's guide for more details. Reference function: DisableMPU_430Xv2.

2.5 JTAG Function Prototypes

2.5.1 Low-Level JTAG Functions

static word IR_Shift (byte Instruction)

Shifts a new instruction into the JTAG instruction register through TDI. (The instruction is shifted in MSB first; the MSB is interpreted by the JTAG instruction register as the LSB.)

Arguments: byte Instruction (8-bit JTAG instruction)

Result: word TDOWord (value shifted out on TDO = JTAG_ID)

static word DR_Shift16 (word Data)

Shifts a given 16-bit word into the JTAG data register through TDI (data shift MSB first)

Arguments: word data (16-bit data value)

Result: word (value shifted out simultaneously on TDO)

static void ResetTAP (void)

Performs fuse-blow check, resets the JTAG interface, and sends the JTAG state machine (TAP controller) to the Run-Test/Idle state

Arguments: None

Result: None

static word ExecutePOR (void)

Executes a power-up clear command through the JTAG control signal register. This function also disables the target device's watchdog timer to avoid an automatic reset condition.

Arguments: None

Result: word (STATUS_OK if the queried JTAG ID is valid, STATUS_ERROR otherwise)

static word SetInstrFetch (void)

Sends the target device's CPU into the instruction fetch state

Arguments: None

Result: word (STATUS_OK if instruction-fetch state is set, STATUS_ERROR otherwise)

static void SetPC (word Addr)

Loads the target device CPU's program counter (PC) with the desired 16-bit address

Arguments: word Addr (desired 16-bit PC value)

Result: None

static void HaltCPU (void)

Sends the target CPU into a controlled, stopped state

Arguments: None

Result: None

static void ReleaseCPU (void)

Releases the target device's CPU from the controlled, stopped state. (Does not release the target device from JTAG control. See ReleaseDevice.)

Arguments: None

Result: None

word VerifyPSA (word StartAddr, word Length, word *DataArray)

Compares the computed pseudo signature analysis (PSA) value to the PSA value shifted out from the target device. It can be used for very fast data block or erasure verification (called by the EraseCheck and VerifyMem prototypes discussed previously).

Arguments: word StartAddr (start address of the memory data block to be checked)
word Length (number of words within the data block)
word *DataArray (pointer to an array containing the data, 0 for erase check)
Result: word (STATUS_OK if comparison was successful, STATUS_ERROR otherwise)

For MSP430X and MSP430Xv2 architecture devices, the following function is defined:

static unsigned long DR_Shift20(unsigned long address)

Shifts a given 20-bit address word into the JTAG data register through TDI (data shift MSB first)

Arguments: long address (20-bit address word)
Result: long TDO value

For MSP430Xv2 architecture devices, the following functions are defined:

word GetCoreIdXv2()

Determines and compares core identification info (Xv2)

Arguments: None
Result: word (STATUS_OK if correct JTAG ID was returned, STATUS_ERROR otherwise)

void jResetJtagTap(void)

Resets target JTAG interface and perform fuse-HW check

Arguments: None
Result: None

void StartJtagJSbw(byte states)

Starts JTAG communication in JSBW mode

Arguments: byte states (reset state)
Result: None

void jRelease(void)

Releases the JSBW logic

Arguments: None
Result: None

long jsbw_Shift(word Format, long Data)

Shifts a value into TDI (MSB first) and simultaneously shift out a value from TDO (MSB first)

Arguments: word Format (number of bits shifted, 8 (F_BYTE), 16 (F_WORD), 20 (F_ADDR) or 32 (F_LONG))
long Data (data to be shifted into TDI)
Result: unsigned long (scanned TDO value)

long jsbw_IR_Shift(byte instruction)

Shifts a new instruction into the JTAG instruction register through JSBW (MSB first, but with interchanged MSB - LSB, to simply use the same shifting function Shift()).

Arguments: instruction (8 bit JTAG instruction)

Result: word (TDOword - value shifted out from TDO: JTAG identification)

long jsbw_DR_Shift(long data)

Shifts data into the JTAG data register through JSBW (MSB first, but with interchanged MSB - LSB, to simply use the same shifting function Shift()).

Arguments: long data

Result: word (TDOword - value shifted out from TDO: JTAG identification)

void JsbwMagicPattern(void)

Applies the magic pattern through JSBW

Arguments: None

Result: None

void jsbwJtagUnlock(void)

Resets the JTAG lock through JSBW

Arguments: None

Result: None

2.5.2 High-Level JTAG Routines

word GetDevice (void)

Takes the target MSP430 device under JTAG control. Sets the target device's CPU watchdog to a hold state; sets the global DEVICE variable.

Arguments: None

Result: word (STATUS_ERROR if fuse is blown, JTAG_ID is incorrect (not = 0x89) or synchronizing time-out occurs; STATUS_OK otherwise)

void ReleaseDevice (word Addr)

Releases the target device from JTAG control; CPU starts execution at the specified PC address

Arguments: word Addr (0xFFFFE: perform reset; address at reset vector loaded into PC; otherwise address specified by Addr loaded into PC)

Result: None

void WriteMem (word Format, word Addr, word Data)

Writes a single byte or word to a given address (RAM/peripheral only)

Arguments: word Format (F_BYTE or F_WORD)

word Addr (destination address for data to be written)

word Data (data value to be written)

Result: None

void WriteMemQuick (word StartAddr, word Length, word *DataArray)

Writes an array of words into the target device memory (RAM/peripheral only)

Arguments: word StartAddr (start address of destination memory)
word Length (number of words to be programmed)
word *DataArray (pointer to array containing the data)

Result: None

void WriteFLASH (word StartAddr, word Length, word *DataArray)

Programs/verifies an array of words into flash memory using the flash controller of the target device

Arguments: word StartAddr (start address of destination flash memory)
word Length (number of words to be programmed)
word *DataArray (pointer to array containing the data)

Result: None

word WriteFLASHallSections(const unsigned int *data, const unsigned long *address, const unsigned long *length_of_sections, const unsigned long sections)

Programs and verifies a set of data arrays of words into a flash memory by using the WriteFLASH() function. It conforms with the CodeArray structure convention of file Target_Code_(IDE).s43 or Target_Code.h.

Arguments: const unsigned int *DataArray (pointer to array with the data)
const unsigned long *address (pointer to array with the startaddresses)
const unsigned long *length_of_sections (pointer to array with the number of words counting from startaddress)
const unsigned long sections (number of sections in code file)

Result: word (STATUS_OK if verification was successful, STATUS_ERROR otherwise)

word ReadMem (word Format, word Addr)

Reads one byte or word from a specified target memory address

Arguments: word Format (F_BYTE or F_WORD)
word Addr (target address for data to be read)

Result: word (data value stored in the target address memory location)

void ReadMemQuick (word StartAddr, word Length, word *DataArray)

Reads an array of words from target memory

Arguments: word StartAddr (start address of target memory to be read)
word Length (number of words to be read)
word *DataArray (pointer to array for data storage)

Result: None

void EraseFLASH (word EraseMode, word EraseAddr)

Performs a mass erase (with or without information memory) or a segment erase of a flash module specified by the given mode and address

Arguments: word EraseMode (ERASE_MASS, ERASE_MAIN or ERASE_SGMT)
word EraseAddr (any address within the selected segment to be erased)

Result: None

word EraseCheck (word StartAddr, word Length)

Uses the VerifyPSA function to perform an erase check over the given memory range

Arguments: word StartAddr (start address of memory to be checked)
word Length (number of words to be checked)

Result: word (STATUS_OK if erase check was successful, STATUS_ERROR otherwise)

word VerifyMem (word StartAddr, word Length, word *DataArray)

Performs a program verification over the given memory range

Arguments: word StartAddr (start address of memory to be verified)
word Length (number of words to be verified)
word *DataArray (pointer to array containing the data)

Result: word (STATUS_OK if verification was successful, STATUS_ERROR otherwise)

For MSP430 and MSP430X architecture devices, the following functions are defined:

word BlowFuse (void)

Programs (or blows) the JTAG interface access security fuse. This function also checks for a successfully programmed fuse using the `IsFuseBlown()` prototype.

Arguments: None

Result: word (STATUS_OK if fuse blow was successful, STATUS_ERROR otherwise)

word IsFuseBlown (void)

Determines if the security fuse has been programmed on the target device

Arguments: None

Result: word (STATUS_OK if fuse is blown, STATUS_ERROR otherwise)

void UnlockInfoA(void)

Unlocks segment A of the InfoMemory (Flash)

Arguments: None

Result: None

For MSP430Xv2 architecture (flash and FRAM devices), the following functions are defined:

word ProgramLockKey(void)

Disables JTAG access to the target device.

Arguments: None

Result: word (TRUE if fuse blow was successful, FALSE otherwise)

word IsLockKeyProgrammed(void)

Checks if the JTAG lock key is programmed.

Arguments: None

Result: word (STATUS_OK if fuse is blown, STATUS_ERROR otherwise)

For MSP430Xv2 architecture (only flash devices), the following functions are defined:

void UnlockInfoA_430Xv2(void)

Unlocks segment A of the InfoMemory (flash).

Arguments: None

Result: None

void UnlockBsl_430Xv2Flash(void)

Unlocks the BSL memory protection.

Arguments: None

Result: None

For MSP430Xv2 architecture (only FRAM devices), the following functions are defined:

word UnlockDevice_430Xv2(unsigned short* password, unsigned long passwordLength)

Unlocks the FRAM memory when a JTAG password is set

Arguments: unsigned short* password (Pointer to array containing the JTAG Password)
unsigned long passwordLength (length of the password in words)

Result: word (STATUS_OK if memory unlock was successful, STATUS_ERROR otherwise)

void EraseFRAM_430Xv2(unsigned long StartAddr, unsigned long Length)

Performs an erase of a user defined FRAM memory section. For FRAM devices the erase equals a write operation of 0xFFFF to the respective memory section. (Could be extended with erase check via PSA) This function should be used for "segment erases" only. For a "mass erase", consider using EraseFRAMViaBootCode_430Xv2 instead.

Arguments: word StartAddr (start address for the erase)
word Length (length of the memory section in WORDS)

Result: None

word DisableMpu_430Xv2(void)

Disables the Memory Protection Unit (FRAM devices only)

Arguments: None

Result: word (STATUS_OK if MPU was disabled successfully, STATUS_ERROR otherwise)

word DisableFramWprod_430Xv2(void)

Disables the Memory Write Protection (FRAM devices only FR6047, FR5994)

Arguments: None

Result: word (STATUS_OK if Memory Write Protection was disabled successfully, STATUS_ERROR otherwise)

void UnlockBsl_430Xv2FRAM(void)

Unlocks the BSL memory protection.

Arguments: None

Result: None

word EraseFRAMViaBootCode_430Xv2(word mailBoxMode, word data1, word data2)

Performs a Erase of FRxx devices using the JTAG mailbox.

Arguments: word mailBoxMode 32Bit 16Bit mode
word data1 mailbox data - first 16BIT
word data2 mailbox data - second 16BIT

Result: word (STATUS_OK if erase was successful, STATUS_ERROR otherwise)

short DownloadProgram(struct_Program* program)

This function downloads a converted MSP430.txt file

Arguments: struct_Program* program Structure containing executable code and memory data

Result: word STATUS_OK if verification was successful, STATUS_ERROR otherwise

short DownloadMsp430Code()

This function configures all needed information to download a program into target memory. After configuration is done it calls the DownloadProgram() function.

Arguments: None

Result: None

2.6 JTAG Features Across Device Families

Table 2-14. MSP430F1xx, MSP430F2xx, MSP430F4xx, MSP430Gxx JTAG Features

Device ^{(1) (2)}	Device ID at 0x0FF0	Device ID at 0x0FF1	Device ID at 0x0FFD	TEST Pin	CPUX	Data Quick Access to SRAM ⁽³⁾	Data Quick Access to Flash ⁽⁴⁾	Fast Flash ⁽⁵⁾	Enh Verify ⁽⁶⁾	Spy-Bi-Wire
AFE2xx	0x02	0x53	-	TRUE	FALSE	READ/ WRITE	READ	TRUE	FALSE	TRUE
F11x1(A)	0xF1	0x12	-	TRUE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F11x2	0x11	0x32	-	TRUE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F12x	0xF1	0x23	-	TRUE	FALSE	FALSE	READ	FALSE	FALSE	FALSE
F12x2	0x12	0x32	-	TRUE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F13x, F14x, F14x1	0xF1	0x49	-	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F15x, F16x	0xF1	0x69	-	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F161x	0xF1	0x6C	-	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F20x1	0xF2	0x01	0x01	TRUE	FALSE	READ/ WRITE	READ	TRUE	FALSE	TRUE
F20x2	0xF2	0x01	0x02	TRUE	FALSE	READ/ WRITE	READ	TRUE	FALSE	TRUE
F20x3	0xF2	0x01	0x03	TRUE	FALSE	READ/ WRITE	READ	TRUE	FALSE	TRUE
F21x1	0xF2	0x13	0x01	TRUE	FALSE	READ/ WRITE	READ	FALSE	TRUE	FALSE
F21x2	0xF2	0x13	0x02	TRUE	FALSE	READ/ WRITE	READ	TRUE	TRUE	TRUE
F22x2, F22x4, G2x44	0xF2	0x27	-	TRUE	FALSE	READ/ WRITE	READ	TRUE	TRUE	TRUE
F23x, F24x, F24x1, F2410	0xF2	0x49	-	FALSE	FALSE	READ/ WRITE	READ	TRUE	TRUE	FALSE
F23x0	0xF2	0x37	-	TRUE	FALSE	READ/ WRITE	READ	TRUE	TRUE	FALSE
F241x, F261x	0xF2	0x6F	-	FALSE	TRUE	READ/ WRITE	READ	TRUE	TRUE	FALSE
F412, F413	0xF4	0x13	-	FALSE	FALSE	FALSE	READ	FALSE	FALSE	FALSE
F415, F417	0xF4	0x27	'W'	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F41x2	0x41	0x52	-	TRUE	FALSE	READ/ WRITE	READ	TRUE	TRUE	TRUE
F(E)42x	0xF4	0x27	'E'	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F(E)42xA	0x42	0x7A	'E'	FALSE	FALSE	READ/ WRITE	READ	TRUE	FALSE	FALSE

Table 2-14. MSP430F1xx, MSP430F2xx, MSP430F4xx, MSP430Gxx JTAG Features (continued)

Device ^{(1) (2)}	Device ID at 0x0FF0	Device ID at 0x0FF1	Device ID at 0x0FFD	TEST Pin	CPUX	Data Quick Access to SRAM ⁽³⁾	Data Quick Access to Flash ⁽⁴⁾	Fast Flash ⁽⁵⁾	Enh Verify ⁽⁶⁾	Spy-Bi-Wire
F(G)42x0	0xF4	0x27	'G'	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F43x (80 pin)	0xF4	0x37	-	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F43x, F44x (100 pin)	0xF4	0x49	-	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F471xx	0xF4	0x7F	-	FALSE	TRUE	READ/ WRITE	READ	TRUE	TRUE	FALSE
F47xx	0xF4	0x49	0x02	FALSE	FALSE	READ/ WRITE	READ	TRUE	TRUE	FALSE
FE42x2	0x42	0x52	'E'	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
FG43x	0xF4	0x39	'G'	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
F(G)461x, F461x1	0xF4	0x6F	'G'	FALSE	TRUE	READ/ WRITE	READ	TRUE	TRUE	FALSE
F(G)47x	0xF4	0x79	'G'	FALSE	FALSE	READ/ WRITE	READ	TRUE	TRUE	FALSE
FW428	0xF4	0x29	'W'	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
FW429	0xF4	0x29	'W'	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
FW42x	0xF4	0x27	'W'	FALSE	FALSE	READ/ WRITE	READ	FALSE	FALSE	FALSE
G2x01, G2x11	0xF2	0x01	0x01	TRUE	FALSE	READ/ WRITE	READ	TRUE	FALSE	TRUE
G2x21, G2x31	0xF2	0x01	0x02	TRUE	FALSE	READ/ WRITE	READ	TRUE	FALSE	TRUE
G2xx2	0x24	0x52	-	TRUE	FALSE	READ/ WRITE	READ	TRUE	FALSE	TRUE
G2xx3	0x25	0x53	-	TRUE	FALSE	READ/ WRITE	READ	TRUE	FALSE	TRUE
G2xx5	0x29	0x55	-	TRUE	FALSE	READ/ WRITE	READ	TRUE	FALSE	TRUE
TCH5E	0x25	0x5C	-	TRUE	FALSE	READ/ WRITE	READ	TRUE	FALSE	TRUE

- (1) All devices listed in this table have JTAG ID 0x89.
- (2) All devices listed in this table support 4-wire JTAG.
- (3) DataQuick Access to SRAM: If READ/WRITE, the device supports read and write of SRAM in quick mode using the IR_DATA_QUICK instruction (see [Section 2.3.3.3](#)). Certain devices might support this instruction for either READ or WRITE operations only.
- (4) DataQuick Access to Flash: If READ/WRITE, the device supports read and write of Flash memory in quick mode using the IR_DATA_QUICK instruction (see [Section 2.3.3.3](#)). Certain devices might support this instruction for either READ or WRITE operations only.
- (5) FastFlash: If TRUE, device has a cumulative erase time ($t_{CMErase}$) of 20 ms; if FALSE, $t_{CMErase}$ is 200 ms (see [Section 2.3.5.1.2](#)).
- (6) EnhVerify: If TRUE, the device supports a more advanced memory content verification mechanism (PSA checksum calculation). If FALSE, the CPU of the device works in the background while the PSA checksum algorithm is executed. Therefore, a POR must be performed after checksum calculation. With the enhanced PSA hardware implementation, the CPU is completely halted during checksum calculation, and a POR is not required after calculation.

Table 2-15. MSP430F5xx, MSP430F6xx, CC430 JTAG Features

Device	Device ID at 0x1A04	Device ID at 0x1A05	Data Quick Access to SRAM ⁽³⁾	Data Quick Access to Flash ⁽⁴⁾
CC430F5123	0x3C	0x81	READ/WRITE	READ
CC430F5125	0x3B	0x81	READ/WRITE	READ
CC430F5133	0x51	0x33	READ/WRITE	READ
CC430F5135	0x51	0x35	READ/WRITE	READ
CC430F5137	0x51	0x37	READ/WRITE	READ
CC430F5143	0x3A	0x81	READ/WRITE	READ
CC430F5145	0x39	0x81	READ/WRITE	READ
CC430F5147	0x38	0x81	READ/WRITE	READ
CC430F6125	0x61	0x25	READ/WRITE	READ
CC430F6126	0x61	0x26	READ/WRITE	READ
CC430F6127	0x61	0x27	READ/WRITE	READ
CC430F6135	0x61	0x35	READ/WRITE	READ
CC430F6137	0x61	0x37	READ/WRITE	READ
CC430F6143	0x37	0x81	READ/WRITE	READ
CC430F6145	0x36	0x81	READ/WRITE	READ
CC430F6147	0x35	0x81	READ/WRITE	READ
MSP430F5131	0x26	0x80	READ/WRITE	READ
MSP430F5132	0x28	0x80	READ/WRITE	READ
MSP430F5151	0x2A	0x80	READ/WRITE	READ
MSP430F5152	0x2C	0x80	READ/WRITE	READ
MSP430F5171	0x2E	0x80	READ/WRITE	READ
MSP430F5172	0x30	0x80	READ/WRITE	READ
MSP430F5212	0x40	0x81	READ/WRITE	READ
MSP430F5213	0x41	0x81	READ/WRITE	READ
MSP430F5214	0x42	0x81	READ/WRITE	READ
MSP430F5217	0x45	0x81	READ/WRITE	READ
MSP430F5218	0x46	0x81	READ/WRITE	READ
MSP430F5219	0x47	0x81	READ/WRITE	READ
MSP430F5222	0x4A	0x81	READ/WRITE	READ
MSP430F5223	0x4B	0x81	READ/WRITE	READ
MSP430F5224	0x4C	0x81	READ/WRITE	READ
MSP430F5227	0x4F	0x81	READ/WRITE	READ
MSP430F5228	0x50	0x81	READ/WRITE	READ
MSP430F5229	0x51	0x81	READ/WRITE	READ
MSP430F5232	0xFA	0x81	READ/WRITE	READ
MSP430F5234	0xF9	0x81	READ/WRITE	READ
MSP430F5237	0xF8	0x81	READ/WRITE	READ
MSP430F5239	0xF7	0x81	READ/WRITE	READ
MSP430F5242	0xF6	0x81	READ/WRITE	READ
MSP430F5244	0xF5	0x81	READ/WRITE	READ
MSP430F5247	0xF4	0x81	READ/WRITE	READ
MSP430F5249	0xF3	0x81	READ/WRITE	READ
MSP430F5252	0x06	0x82	READ/WRITE	READ
MSP430F5253	0x05	0x82	READ/WRITE	READ
MSP430F5254	0x04	0x82	READ/WRITE	READ
MSP430F5255	0x03	0x82	READ/WRITE	READ

Table 2-15. MSP430F5xx, MSP430F6xx, CC430 JTAG Features (continued)

Device	Device ID at 0x1A04	Device ID at 0x1A05	Data Quick Access to SRAM ⁽³⁾	Data Quick Access to Flash ⁽⁴⁾
MSP430F5256	0x02	0x82	READ/WRITE	READ
MSP430F5257	0x01	0x82	READ/WRITE	READ
MSP430F5258	0x00	0x82	READ/WRITE	READ
MSP430F5259	0xFF	0x81	READ/WRITE	READ
MSP430F5304	0x12	0x81	READ/WRITE	READ
MSP430F5308	0x13	0x81	READ/WRITE	READ
MSP430F5309	0x14	0x81	READ/WRITE	READ
MSP430F5310	0x15	0x81	READ/WRITE	READ
MSP430F5324	0x16	0x81	READ/WRITE	READ
MSP430F5325	0x17	0x81	READ/WRITE	READ
MSP430F5326	0x18	0x81	READ/WRITE	READ
MSP430F5327	0x19	0x81	READ/WRITE	READ
MSP430F5328	0x1A	0x81	READ/WRITE	READ
MSP430F5329	0x1B	0x81	READ/WRITE	READ
MSP430F5333	0x25	0x81	READ/WRITE	READ
MSP430F5335	0x27	0x81	READ/WRITE	READ
MSP430F5336	0x28	0x81	READ/WRITE	READ
MSP430F5338	0x2A	0x81	READ/WRITE	READ
MSP430F5340	0x1C	0x81	READ/WRITE	READ
MSP430F5341	0x1D	0x81	READ/WRITE	READ
MSP430F5342	0x1E	0x81	READ/WRITE	READ
MSP430F5357	0x34	0x81	READ/WRITE	READ
MSP430F5358	0x33	0x81	READ/WRITE	READ
MSP430F5359	0x32	0x81	READ/WRITE	READ
MSP430F5418	0x54	0x18	READ/WRITE	READ
MSP430F5418A	0x00	0x80	READ/WRITE	READ
MSP430F5419	0x54	0x19	READ/WRITE	READ
MSP430F5419A	0x01	0x80	READ/WRITE	READ
MSP430F5435	0x54	0x35	READ/WRITE	READ
MSP430F5435A	0x02	0x80	READ/WRITE	READ
MSP430F5436	0x54	0x36	READ/WRITE	READ
MSP430F5436A	0x03	0x80	READ/WRITE	READ
MSP430F5437	0x54	0x37	READ/WRITE	READ
MSP430F5437A	0x04	0x80	READ/WRITE	READ
MSP430F5438	0x54	0x38	READ/WRITE	READ
MSP430F5438A	0x05	0x80	READ/WRITE	READ
MSP430F5500	0x3B	0x80	READ/WRITE	READ
MSP430F5501	0x32	0x80	READ/WRITE	READ
MSP430F5502	0x33	0x80	READ/WRITE	READ
MSP430F5503	0x34	0x80	READ/WRITE	READ
MSP430F5504	0x35	0x80	READ/WRITE	READ
MSP430F5505	0x36	0x80	READ/WRITE	READ
MSP430F5506	0x37	0x80	READ/WRITE	READ
MSP430F5507	0x38	0x80	READ/WRITE	READ
MSP430F5508	0x39	0x80	READ/WRITE	READ
MSP430F5509	0x3A	0x80	READ/WRITE	READ

Table 2-15. MSP430F5xx, MSP430F6xx, CC430 JTAG Features (continued)

Device	Device ID at 0x1A04	Device ID at 0x1A05	Data Quick Access to SRAM ⁽³⁾	Data Quick Access to Flash ⁽⁴⁾
MSP430F5510	0x31	0x80	READ/WRITE	READ
MSP430F5513	0x55	0x13	READ/WRITE	READ
MSP430F5514	0x55	0x14	READ/WRITE	READ
MSP430F5515	0x55	0x15	READ/WRITE	READ
MSP430F5517	0x55	0x17	READ/WRITE	READ
MSP430F5519	0x55	0x19	READ/WRITE	READ
MSP430F5521	0x55	0x21	READ/WRITE	READ
MSP430F5522	0x55	0x22	READ/WRITE	READ
MSP430F5524	0x55	0x24	READ/WRITE	READ
MSP430F5525	0x55	0x25	READ/WRITE	READ
MSP430F5526	0x55	0x26	READ/WRITE	READ
MSP430F5527	0x55	0x27	READ/WRITE	READ
MSP430F5528	0x55	0x28	READ/WRITE	READ
MSP430F5529	0x55	0x29	READ/WRITE	READ
MSP430F5630	0x3C	0x80	READ/WRITE	READ
MSP430F5631	0x3E	0x80	READ/WRITE	READ
MSP430F5632	0x40	0x80	READ/WRITE	READ
MSP430F5633	0x42	0x80	READ/WRITE	READ
MSP430F5634	0x44	0x80	READ/WRITE	READ
MSP430F5635	0x0E	0x80	READ/WRITE	READ
MSP430F5636	0x10	0x80	READ/WRITE	READ
MSP430F5637	0x12	0x80	READ/WRITE	READ
MSP430F5638	0x14	0x80	READ/WRITE	READ
MSP430F5658	0x31	0x81	READ/WRITE	READ
MSP430F5659	0x30	0x81	READ/WRITE	READ
MSP430F6433	0x1F	0x81	READ/WRITE	READ
MSP430F6435	0x21	0x81	READ/WRITE	READ
MSP430F6436	0x22	0x81	READ/WRITE	READ
MSP430F6438	0x24	0x81	READ/WRITE	READ
MSP430F6458	0x2E	0x81	READ/WRITE	READ
MSP430F6459	0x2D	0x81	READ/WRITE	READ
MSP430F6630	0x46	0x80	READ/WRITE	READ
MSP430F6631	0x48	0x80	READ/WRITE	READ
MSP430F6632	0x4A	0x80	READ/WRITE	READ
MSP430F6633	0x4C	0x80	READ/WRITE	READ
MSP430F6634	0x4E	0x80	READ/WRITE	READ
MSP430F6635	0x16	0x80	READ/WRITE	READ
MSP430F6636	0x18	0x80	READ/WRITE	READ
MSP430F6637	0x1A	0x80	READ/WRITE	READ
MSP430F6638	0x1C	0x80	READ/WRITE	READ
MSP430F6658	0x2C	0x81	READ/WRITE	READ
MSP430F6659	0x2B	0x81	READ/WRITE	READ
MSP430F6700	0x54	0x80	READ/WRITE	READ
MSP430F6701	0x55	0x80	READ/WRITE	READ
MSP430F6702	0x56	0x80	READ/WRITE	READ
MSP430F6703	0x57	0x80	READ/WRITE	READ

Table 2-15. MSP430F5xx, MSP430F6xx, CC430 JTAG Features (continued)

Device	Device ID at 0x1A04	Device ID at 0x1A05	Data Quick Access to SRAM ⁽³⁾	Data Quick Access to Flash ⁽⁴⁾
MSP430F6720	0x58	0x80	READ/WRITE	READ
MSP430F6720A	0x76	0x82	READ/WRITE	READ
MSP430F6721	0x59	0x80	READ/WRITE	READ
MSP430F6721A	0x77	0x82	READ/WRITE	READ
MSP430F6722	0x60	0x80	READ/WRITE	READ
MSP430F6723	0x61	0x80	READ/WRITE	READ
MSP430F6723A	0x79	0x82	READ/WRITE	READ
MSP430F6724	0x6D	0x81	READ/WRITE	READ
MSP430F6724A	0x7A	0x82	READ/WRITE	READ
MSP430F6725	0x6E	0x81	READ/WRITE	READ
MSP430F6725A	0x7B	0x82	READ/WRITE	READ
MSP430F6726	0x6F	0x81	READ/WRITE	READ
MSP430F6726A	0x7C	0x82	READ/WRITE	READ
MSP430F6730	0x62	0x80	READ/WRITE	READ
MSP430F6730A	0x80	0x82	READ/WRITE	READ
MSP430F6731	0x63	0x80	READ/WRITE	READ
MSP430F6731A	0x81	0x82	READ/WRITE	READ
MSP430F6732	0x64	0x80	READ/WRITE	READ
MSP430F6733	0x65	0x80	READ/WRITE	READ
MSP430F6733A	0x83	0x82	READ/WRITE	READ
MSP430F6734	0x6A	0x81	READ/WRITE	READ
MSP430F6734A	0x84	0x82	READ/WRITE	READ
MSP430F6735	0x6B	0x81	READ/WRITE	READ
MSP430F6735A	0x85	0x82	READ/WRITE	READ
MSP430F6736	0x6C	0x81	READ/WRITE	READ
MSP430F6736A	0x86	0x82	READ/WRITE	READ
MSP430F6745	0x88	0x81	READ/WRITE	READ
MSP430F67451	0x97	0x81	READ/WRITE	READ
MSP430F67451A	0x25	0x82	READ/WRITE	READ
MSP430F6745A	0x16	0x82	READ/WRITE	READ
MSP430F6746	0x89	0x81	READ/WRITE	READ
MSP430F67461	0x98	0x81	READ/WRITE	READ
MSP430F67461A	0x26	0x82	READ/WRITE	READ
MSP430F6746A	0x17	0x82	READ/WRITE	READ
MSP430F6747	0x8A	0x81	READ/WRITE	READ
MSP430F67471	0x99	0x81	READ/WRITE	READ
MSP430F67471A	0x27	0x82	READ/WRITE	READ
MSP430F6747A	0x18	0x82	READ/WRITE	READ
MSP430F6748	0x8B	0x81	READ/WRITE	READ
MSP430F67481	0x9A	0x81	READ/WRITE	READ
MSP430F67481A	0x28	0x82	READ/WRITE	READ
MSP430F6748A	0x19	0x82	READ/WRITE	READ
MSP430F6749	0x8C	0x81	READ/WRITE	READ
MSP430F67491	0x9B	0x81	READ/WRITE	READ
MSP430F67491A	0x29	0x82	READ/WRITE	READ
MSP430F6749A	0x1A	0x82	READ/WRITE	READ

Table 2-15. MSP430F5xx, MSP430F6xx, CC430 JTAG Features (continued)

Device	Device ID at 0x1A04	Device ID at 0x1A05	Data Quick Access to SRAM ⁽³⁾	Data Quick Access to Flash ⁽⁴⁾
MSP430F67621	0x38	0x82	READ/WRITE	READ
MSP430F67621A	0x87	0x82	READ/WRITE	READ
MSP430F67641	0x39	0x82	READ/WRITE	READ
MSP430F67641A	0x88	0x82	READ/WRITE	READ
MSP430F6765	0x8D	0x81	READ/WRITE	READ
MSP430F67651	0x9C	0x81	READ/WRITE	READ
MSP430F67651A	0x2A	0x82	READ/WRITE	READ
MSP430F6765A	0x1B	0x82	READ/WRITE	READ
MSP430F6766	0x8E	0x81	READ/WRITE	READ
MSP430F67661	0x9D	0x81	READ/WRITE	READ
MSP430F67661A	0x2B	0x82	READ/WRITE	READ
MSP430F6766A	0x1C	0x82	READ/WRITE	READ
MSP430F6767	0x8F	0x81	READ/WRITE	READ
MSP430F67671	0x9E	0x81	READ/WRITE	READ
MSP430F67671A	0x2C	0x82	READ/WRITE	READ
MSP430F6767A	0x1D	0x82	READ/WRITE	READ
MSP430F6768	0x90	0x81	READ/WRITE	READ
MSP430F67681	0x9F	0x81	READ/WRITE	READ
MSP430F67681A	0x2D	0x82	READ/WRITE	READ
MSP430F6768A	0x1E	0x82	READ/WRITE	READ
MSP430F6769	0x91	0x81	READ/WRITE	READ
MSP430F67691	0xA0	0x81	READ/WRITE	READ
MSP430F67691A	0x2E	0x82	READ/WRITE	READ
MSP430F6769A	0x1F	0x82	READ/WRITE	READ
MSP430F6775	0x92	0x81	READ/WRITE	READ
MSP430F67751	0xA1	0x81	READ/WRITE	READ
MSP430F67751A	0x2F	0x82	READ/WRITE	READ
MSP430F6775A	0x20	0x82	READ/WRITE	READ
MSP430F6776	0x93	0x81	READ/WRITE	READ
MSP430F67761	0xA2	0x81	READ/WRITE	READ
MSP430F67761A	0x30	0x82	READ/WRITE	READ
MSP430F6776A	0x21	0x82	READ/WRITE	READ
MSP430F6777	0x94	0x81	READ/WRITE	READ
MSP430F67771	0xA3	0x81	READ/WRITE	READ
MSP430F67771A	0x31	0x82	READ/WRITE	READ
MSP430F6777A	0x22	0x82	READ/WRITE	READ
MSP430F6778	0x95	0x81	READ/WRITE	READ
MSP430F67781	0xA4	0x81	READ/WRITE	READ
MSP430F67781A	0x32	0x82	READ/WRITE	READ
MSP430F6778A	0x23	0x82	READ/WRITE	READ
MSP430F6779	0x96	0x81	READ/WRITE	READ
MSP430F67791	0xA5	0x81	READ/WRITE	READ
MSP430F67791A	0x33	0x82	READ/WRITE	READ
MSP430F6779A	0x24	0x82	READ/WRITE	READ
MSP430FG6425	0x37	0x82	READ/WRITE	READ
MSP430FG6426	0x36	0x82	READ/WRITE	READ

Table 2-15. MSP430F5xx, MSP430F6xx, CC430 JTAG Features (continued)

Device	Device ID at 0x1A04	Device ID at 0x1A05	Data Quick Access to SRAM ⁽³⁾	Data Quick Access to Flash ⁽⁴⁾
MSP430FG6625	0x35	0x82	READ/WRITE	READ
MSP430FG6626	0x34	0x82	READ/WRITE	READ
MSP430SL5438A	0xEE	0x81	READ/WRITE	READ

Table 2-16. MSP430FRxx JTAG Features

Device	Device ID at 0x1A04	Device ID at 0x1A05	Data Quick Access to SRAM ⁽³⁾	Data Quick Access to FRAM	JTAG ID
MSP430FR2000	0x20	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2032	0x78	0x82	READ/WRITE	READ/WRITE	0x98
MSP430FR2033	0x75	0x82	READ/WRITE	READ/WRITE	0x98
MSP430FR2100	0x21	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2110	0xFB	0x82	READ/WRITE	READ/WRITE	0x98
MSP430FR2111	0xFA	0x82	READ/WRITE	READ/WRITE	0x98
MSP430FR2153	0x1D	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2155	0x1E	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2310	0xF1	0x82	READ/WRITE	READ/WRITE	0x98
MSP430FR2311	0xF0	0x82	READ/WRITE	READ/WRITE	0x98
MSP430FR2353	0x0D	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2355	0x0C	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2422	0x11	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2433	0x40	0x82	READ/WRITE	READ/WRITE	0x98
MSP430FR2475	0x2B	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2476	0x2A	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2512	0x1C	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2522	0x10	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2532	0x3F	0x82	READ/WRITE	READ/WRITE	0x98
MSP430FR2533	0x3D	0x82	READ/WRITE	READ/WRITE	0x98
MSP430FR2632	0x3E	0x82	READ/WRITE	READ/WRITE	0x98
MSP430FR2633	0x3C	0x82	READ/WRITE	READ/WRITE	0x98
MSP430FR2675	0x29	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR2676	0x28	0x83	READ/WRITE	READ/WRITE	0x98
MSP430FR4131	0xF2	0x81	READ/WRITE	READ/WRITE	0x98
MSP430FR4132	0xF1	0x81	READ/WRITE	READ/WRITE	0x98
MSP430FR4133	0xF0	0x81	READ/WRITE	READ/WRITE	0x98
MSP430FR5041	0x0F	0x83	READ/WRITE	READ/WRITE	0x99
MSP430FR5043	0x17	0x83	READ/WRITE	READ/WRITE	0x99
MSP430FR50431	0x18	0x83	READ/WRITE	READ/WRITE	0x99
MSP430FR5720	0x70	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5721	0x77	0x80	READ/WRITE	READ/WRITE	0x91
MSP430FR5722	0x71	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5723	0x72	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5724	0x73	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5725	0x78	0x80	READ/WRITE	READ/WRITE	0x91
MSP430FR5726	0x74	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5727	0x79	0x80	READ/WRITE	READ/WRITE	0x91
MSP430FR5728	0x7A	0x80	READ/WRITE	READ/WRITE	0x91
MSP430FR5729	0x7B	0x80	READ/WRITE	READ/WRITE	0x91

Table 2-16. MSP430FRxx JTAG Features (continued)

Device	Device ID at 0x1A04	Device ID at 0x1A05	Data Quick Access to SRAM ⁽³⁾	Data Quick Access to FRAM	JTAG ID
MSP430FR5730	0x7C	0x80	READ/WRITE	READ/WRITE	0x91
MSP430FR5731	0x7E	0x80	READ/WRITE	READ/WRITE	0x91
MSP430FR5732	0x75	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5733	0x7F	0x80	READ/WRITE	READ/WRITE	0x91
MSP430FR5734	0x00	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5735	0x76	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5736	0x77	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5737	0x01	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5738	0x02	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5739	0x03	0x81	READ/WRITE	READ/WRITE	0x91
MSP430FR5847	0x53	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5848	0x54	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5849	0x55	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5857	0x57	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5858	0x58	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5859	0x59	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5867	0x5B	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5868	0x5C	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5869	0x5D	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5870	0x5E	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR5872(1)	0x60	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR5887	0xC1	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5888	0xC2	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5889	0xC3	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5922(1) DGG package	0x61	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR5922(1) PM or RGC package	0x62	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR5947	0x5F	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5948	0x60	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5949	0x61	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5957	0x63	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5958	0x64	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5959	0x65	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5962	0xA6	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR5964	0xA4	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR5967	0x67	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5968	0x68	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5969	0x69	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5970	0x5B	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR5972(1)	0x5D	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR5986	0xDF	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5987	0xA9	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5988	0xAA	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5989(1)	0xAB	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR5992	0xA3	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR5994	0xA1	0x82	READ/WRITE	READ/WRITE	0x99

Table 2-16. MSP430FRxx JTAG Features (continued)

Device	Device ID at 0x1A04	Device ID at 0x1A05	Data Quick Access to SRAM ⁽³⁾	Data Quick Access to FRAM	JTAG ID
MSP430FR59941	0xA2	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6035	0xED	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR60371	0xEF	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6037	0xEC	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6041	0x14	0x83	READ/WRITE	READ/WRITE	0x99
MSP430FR6043	0x12	0x83	READ/WRITE	READ/WRITE	0x99
MSP430FR60431	0x1A	0x83	READ/WRITE	READ/WRITE	0x99
MSP430FR6045	0xEB	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR60471	0xEE	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6047	0xEA	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6820 DGG package	0x55	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6820 PM or RGC package	0x56	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6822(1) DGG package	0x59	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6822(1) PM or RGC package	0x5A	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6870	0x4C	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6872(1)	0x4E	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6877	0xC4	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR6879	0xC6	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR6887	0xBE	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR6888	0xBF	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR6889	0xC0	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR6920 DGG package	0x4F	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6920 PM or RGC package	0x50	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6922(1) DGG package	0x53	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6922(1) PM or RGC package	0x54	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6927	0xB2	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR6928	0xB3	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR6970	0x49	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6972(1)	0x4B	0x82	READ/WRITE	READ/WRITE	0x99
MSP430FR6977	0xAC	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR6979	0xAE	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR6987	0xA6	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR6988	0xA7	0x81	READ/WRITE	READ/WRITE	0x99
MSP430FR6989(1)	0xA8	0x81	READ/WRITE	READ/WRITE	0x99

Table 2-17. MSP430iix JTAG Features

Device	Device ID at 0x0FF0	Device ID at 0x0FF1	Data Quick Access to SRAM ⁽¹⁾	Data Quick Access to Flash ⁽²⁾
MSP430i2020	0x40	0x20	READ/WRITE	READ
MSP430i2021	0x40	0x20	READ/WRITE	READ
MSP430i2030	0x40	0x20	READ/WRITE	READ

Table 2-17. MSP430ixx JTAG Features (continued)

Device	Device ID at 0x0FF0	Device ID at 0x0FF1	Data Quick Access to SRAM ⁽¹⁾	Data Quick Access to Flash ⁽²⁾
MSP430i2031	0x40	0x20	READ/WRITE	READ
MSP430i2040	0x40	0x20	READ/WRITE	READ
MSP430i2041	0x40	0x20	READ/WRITE	READ

- (1) DataQuick Access to SRAM: If READ/WRITE, the device supports read and write of SRAM in quick mode using the IR_DATA_QUICK instruction (see [Section 2.3.3.3](#)). Certain devices might support this instruction for either READ or WRITE operations only.
- (2) DataQuick Access to Flash: If READ/WRITE, the device supports read and write of Flash memory in quick mode using the IR_DATA_QUICK instruction (see [Section 2.3.3.3](#)). Certain devices might support this instruction for either READ or WRITE operations only.

2.7 References

[MSP430 device data sheets](#)

[CC430 device data sheets](#)

[MSP430x1xx Family User's Guide](#)

[MSP430x4xx Family User's Guide](#)

[MSP430x2xx Family User's Guide](#)

[MSP430F5xx and MSP430F6xx Family User's Guide](#)

[MSP430FR2xx and MSP430FR4xx Family User's Guide](#)

[MSP430FR57xx Family User's Guide](#)

[MSP430FR58xx, MSP430FR59xx, and MSP430FR6xx Family User's Guide](#)

[CC430 Family User's Guide](#)

IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Std 1149.1

3 JTAG Programming Hardware and Software Implementation

3.1 Implementation History

There are four Replicator implementations. The latest version is included in the [associated source code ZIP file](#) and discussed in this document. The main difference between the first two implementations is the use of the `srec_cat.exe` function in place of `FileMaker.exe`. The implementation described in this document is the preferred implementation, the previous two implementations are no longer maintained.

Note

This version of the Replicator software only supports the REP430F hardware with an MSP430F5437 as the host controller. Older versions of the Replicator hardware are no longer supported.

3.2 Implementation Overview

The following sections describe the examples that are provided with this document (download examples from <http://www.ti.com/lit/zip/slau320>). Each example demonstrates the software functions described in the previous sections using an MSP430F5437 as the host controller that programs the given target MSP430 device of choice. The complete C source code and project files are provided in the examples zip file. A schematic for the system as implemented in this discussion is also provided.

Key features of the JTAG Replicator programmer implementations are as follows:

- Support all MSP430 flash- and FRAM-based devices. There are specific software projects for the following target device Replicator implementations:
 - Replicator430: Original MSP430 architecture devices (includes Spy-Bi-Wire implementation for devices that support it)
 - Replicator430X: MSP430 devices from the 2xx and 4xx families with extended address space (20 bit), also referred to as MSP430X devices (4-wire JTAG only)
 - Replicator430Xv2: MSP430 devices from the 5xx and 6xx families with flash memory and extended address space (20 bit), also referred to as MSP430Xv2 devices (includes both 4-wire JTAG and Spy-Bi-Wire implementation)
 - Replicator430FR: FRAM devices (FR2xx, FR4xx, FR5xx, and FR6xx families) (includes both 4-wire JTAG and Spy-Bi-Wire implementation)

Note

The Replicator source files are provided in independent folders with the same names as previously given. Within these folders, file names are assigned accordingly when applicable to a certain device type. For example, the file `JTAGfunc430.c` that is used in the Replicator430 version is renamed to `JTAGfunc430X.c` in the Replicator430X version and `JTAGfunc430Xv2.c` in the Replicator430Xv2 version.

- Maximum target device program code size: approximately 250KB (due to the limited memory resources of the MSP430F5437 host controller of 256KB)
- Programming speed (erase, program, verify): approximately 8KB in 1.5 s, 48KB in 8 s
- Fast verify and erase check: 17KB in 10 ms
- Support programming of the JTAG access fuse (permanently disables device memory access through JTAG)
- Stand-alone target programming operation (no personal computer or additional supporting hardware or software required)

3.3 Software Operation

The host controller stores the JTAG communication protocol code and the target program in its flash memory (256KB available on the MSP430F5437). The programming software itself occupies between 4KB and 6KB, so approximately 250KB remain for the target device program. The Replicator host can be loaded with the target source code by the flash emulation tool (FET) or by the MSP430 serial programming adapter (visit the [MSP430 website](#) for more information on device programming tools).

The basic functionality of the programmer is as follows:

1. Push the GO button to generate a hardware reset and start the host controller's JTAG communication routine to erase, program, and verify the target device.
2. While the system is active, the yellow LED on the programmer board is on.
3. After successful completion, only the green LED is on.
4. If an error occurs or communication to the target device fails, only the red LED remains on.

The entire procedure takes approximately 3 seconds for a target program size of 8KB. (Some code not strictly required to erase, program, and verify the target MSP430 MCU is executed in the Replicator.c source file, increasing the specified programming times. These additional instructions can be customized to fit the individual system programming requirements.)

To program the host MSP430F5437, different development environments can be used— IAR Embedded Workbench® IDE (IAR) or Code Composer Studio™ IDE (CCS) by Texas Instruments. The free version of IAR imposes code size restrictions. To use the 250KB previously mentioned, the full version of IAR or is needed. The folder structure provides both an IAR and CCS folder, each of which contains the environment-specific files. For IAR, the workspace file (extension .eww) must be started to open the IAR workbench. Using CCS, each Replicator project must be imported into the user's workspace. This can be done by right-clicking in the project's view and selecting "Import" in the context menu. After choosing the desired Replicator folder, the project is imported and ready to use.

3.4 Software Structure

3.4.1 Programmer Firmware

The programming software is partitioned in three levels and consists of nine files in addition to the target program (see [Table 3-1](#)).

Table 3-1. Programmer Firmware

Top level	Specifies which programming functions (erase, program, verify, blow fuse) are to be executed.	
	Replicator430.c Replicator430X.c Replicator430Xv2.c Replicator430FR.c	Contains the main section, which can be modified to meet custom requirements. In the main section of this program, the target device is erased, checked for successful erasure, and programmed. Programming loads the provided example code to the target device's memory space. (The provided code file simply flashes port pins P1.0 and/or P5.1, which drive the LEDs on the socket board provided with the FET tools, available from Texas Instruments MSP430 Group. This is the compiled FETXXX_1.s43 example code file.) This file must be replaced by the required user program and added to the project to be compiled and loaded into the host. To demonstrate the capabilities of the MSP430 JTAG interface, additional code is included, which manipulates the I/O-ports and RAM of the target device. These routines can be used to test the target device and PCB for successful communication.
	Config430.h Config430X.h Config430Xv2.h Config430FR.h	Contains high-level definitions to be set by the user before initially programming the host controller firmware. These so called 'quick start options' specify among other things the voltage level supplied by the Replicator hardware and the interface used to communicate with the target device.
	Target_Code.h	Contains the basic declarations of the program code of the target device. If a C-header file should be implemented to program the target device instead of an assembly file, replace the content of Target_Code.h by the output of srec_cat.exe and remove Target_Code.s43 (IAR) or Target_Code.asm (CCS) from the project. The Target_Code.h file is generated by the srec_cat.exe file directly or by the srec.bat file.
JTAG functions	All MSP430-specific functions are defined here. These files should not be modified under any circumstance.	
	JTAGfunc.c JTAGfunc430X.c JTAGfunc430Xv2.c JTAGfunc430FR.c	Contain the MSP430-specific functions needed for flash programming
	JTAGfunc.h JTAGfunc430X.h JTAGfunc430Xv2.h JTAGfunc430FR.h	Contain constant definitions and function prototypes used for JTAG communication
	JSBW.c	Contains special functions that emulate the Spy-Bi-Wire interface to wake up MSP430 devices from LPMx.5 (Replicator430Xv2 and Replicator430FR only)

Table 3-1. Programmer Firmware (continued)

Low-level functions	All functions that depend specifically on the host controller (JTAG port I/O and timing functions) are located here. These files need to be adapted if a host controller other than the MSP430F5437 is implemented.	
	LowLevelFunc430.c LowLevelFunc430X.c LowLevelFunc430Xv2.c	Contain the basic host-specific functions
	LowLevelFunc430.h LowLevelFunc430X.h LowLevelFunc430Xv2.h	Contain host-specific definitions and function prototypes
Devices	Describes features and differences between MSP430 devices with respect to FLASH programming. (Replicator430 and Replicator430X only)	
	Devices430.c Devices430x.c	Functions to distinguish MSP430 devices concerning FLASH programming.
	Devices430.h Devices430x.h	Device function prototypes and definitions for FLASH programming.
Funclets	Contain target code which is written into the target's RAM to speed up writing to and erasing from the device's NVM. (These files can only be rebuilt using IAR Embedded Workbench and SRecord)	
	FlashErase.c FlashWrite.c	Funclets to speed up flash erase and programming
	FramErase.c FramWrite.c	Funclets to speed up FRAM erase and programming

3.4.2 Target Code

3.4.2.1 Target Code Download for Replicator430, Replicator430X, and Replicator430Xv2

As mentioned previously, the target device's program code must be supplied separately. There are two ways to include the provided example in the project space of the program to be sent to the host. Either include a separate file (for example, Target_Code.s43 (IAR) or Target_Code.asm (CCS)) or replace the C-Array in the Target_Code.h header file. Both alternatives must provide the binary target code and conform to the format expected by the [source code](#).

To build these files from the TI-txt format output from the compiler, an open-source conversion program called SRecord can be downloaded from <http://sourceforge.net/projects/srecord/>. The SRecord package includes the executable srec_cat.exe.

This executable is a command line application that expects parameters in the following format:

```
srec_cat.exe Target_Code.txt -ti_txt -Output Target_Code.h -c_array -output_word -c_compressed
```

or

```
(IAR) srec_cat.exe Target_Code.txt -ti_txt -Output Target_Code.s43 -asm -output_word -a430
(CCS) srec_cat.exe Target_Code.txt -ti_txt -Output Target_Code.asm -asm -output_word -cl430
```

Parameter description:

- srec_cat.exe : The name of the application
- Target_Code.txt -ti_txt : This is the input file by name and its format
- -Output : A keyword to make clear that following parameters describe the output file and format
- Target_Code.x -[c_array,asm] : This is the output file by name and the format that the input file should be converted to. For this example only, C-header and assembly formats are allowed. Choose one format for your purpose.
- -output_word : The parameter is necessary because the source code expects words to write to the target device. Otherwise, srec_cat.exe would write bytes.
- -c_compressed : This statement is additional to the c_array output. If specified, the output does not fill any address gap with a 0xFF pattern, and does not increase the file size.
- The following statements are additional to the assembly output. Choose one to specify your format.
 - -a430 : Writes an assembly file that is understood by the IAR Embedded Workbench in the Replicator context.

- -cl430 : Writes an assembly file that is understood by TI CCS in the Replicator context.

The provided file srec.bat generates all three types of output files (.h, .asm, and .s43) simultaneously. The command line format is: srec Target_Code. In addition to generating the actual target code, SRecord is also used to convert the funclets for flash-/Fram-write/erase which can only be rebuilt using IAR Embedded Workbench. In this case the SRecord is started as a post build process in the corresponding projects to convert the TI-txt code to a format accepted by the [source code](#). TI-txt format can be output by the IAR Linker by setting the required compiler/linker options (see the IAR tool instruction guides for more information or see the funclet projects that were described previously). This can also be done in CCS using the hex430 command line executable.

Note

If the TI-txt source file includes odd segment addresses or an odd number of data bytes, additional byte padding might be required to generate appropriate word-aligned output format. Use srec_cat.exe with a "--fill 0xFF --within <input> --range-padding 2" filter to fix this problem. The srec.bat automatically filters the output format for appropriate word alignment. For example, "srec_cat.exe Target_Code.txt -ti_txt --fill 0xFF --within Target_Code.txt -ti_txt --range-padding 2 -Output Target_Code.h -c_array -output_word -c_compressed".

Note

If using assembly source code that contains the target code, make sure that the array declarations are stored in target_code.h. An example can be seen in the included basic header file.

Note

The SRecord conversion program is open source and has a much larger range of functions. For more information and documentation see <http://srecord.sourceforge.net/>.

This software was tested to function correctly with version 1.36, but is not necessarily compatible with future versions.

Note

To enable easy porting of the software to other microcontrollers, the provided source code is written in ANSI C. As always, it is recommended that the latest available version of the applicable MSP430 development software be installed before beginning a new project.

3.4.2.2 Target Code Download for Replicator430FR (FRAM)

How to download an msp430.txt file generated by EW430 or CCS:

To download different code with the Replicator430FR, generate an msp430.txt file using EW430 or CSS. Name this file msp430Code.txt and save it into the folder \$ProjectDir\$\Targetcode\SRecord.

Convert this msp430Code.txt file by running the batch file "ConvertMsp430_txt.bat" in the \$ProjectDir\$\Targetcode\SRecord folder. The output is a C header file including the converted target code. Copy this C header file into the Replicator430FR code folder and include it in the JTAGfunc430FR.c file (#include "msp430Code.h").

Call the function DownloadMsp430Code() in the JTAGfunc430FR.c file to load the converted program into the target device memory.

3.5 Hardware Setup

The hardware consists of the host controller MSP430F5437, programmable voltage regulator that can supply target device with V_{CC} 2.1 V to 3.6 V with step 0.1 V, two JTAG interface connectors, and one BSL interface connector. An external power supply delivering 8 V to 10 V dc at 200 mA is required for operation if the option to blow the security fuse is required, or 4 V to 10 V dc at 200 mA can be used otherwise. The REP430F can

also be supplied from the target's device $V_{CC} \geq 3\text{ V}$ (on JTAG pin 2 or 4; see [Figure 3-1](#)) if the option to blow the security fuse is not required.

3.5.1 Host Controller

To achieve maximum programming speed, the host controller MSP430F5437 can run at a maximum CPU clock frequency of 18 MHz that is used from PLL and XTAL 12 MHz provided on LFXT1. The host is programmed through a dedicated JTAG port labeled Host JTAG (see [Figure 3-1](#)).

3.5.2 Target Connection

The target MSP430 device is connected to the host controller/programmer through the 14-pin connector labeled Target JTAG, which has the same standard signal assignment as all available MSP430 emulation tools with extra two pins that can be used for BSL connection. The programmable target device supply voltage of 2.1 V to 3.6 V with step 0.1 V is available on pin 2 of this connector, eliminating the need for an additional supply for the target system. The required Spy-Bi-Wire or 4-wire JTAG and GND must be connected. (On devices requiring the TEST pin, the TEST signal also must be provided from the programmer to the target MSP430 device.) Host controller in the REP430F is supplied from the $V_{CC} = 3\text{ V}$, while target device can be supplied with the V_{CC} from 2.1 V to 3.6 V. To avoid a problem with I/O levels, the REP430V contains voltage level translators between target device and host controller. Voltage translators are supplied from the host controller $V_{CC} = 3\text{ V}$ from one side, and from the target's device V_{CC} (provided on pin 2 of the target JTAG connector) from the other side. That allows supply the target device with the I/O levels exactly as required by the target device.)

To enable programming of all MSP430 flash-based devices including a JTAG access fuse, voltage translators are used and MOSFET switches are controlled by the host MSP430. MOSFET Q2 controls V_{pp} on devices with a TEST pin; Q1 connects V_{pp} to TDI on devices not requiring a TEST signal. U8 isolates the host controller from the target TEST pin while V_{pp} is connected to the target TEST input, while U6 isolates the host controller from the target TDI pin while V_{pp} is connected to the target TDI input. U7 connects the host TDI signal to the target TDO pin while the fuse is programmed (for devices without a TEST pin). The host controller program includes delays, which consider a MOSFET switching time of a maximum of 1 ms. Q1 and Q2 should have a $R_{on} < 2\ \Omega$ to minimize voltage drop during fuse programming. While the fuse is being programmed, a maximum current flow of 100 mA is possible for approximately 2 μs into the TDI pin (or the TEST pin, depending on the target device).

Note

An MSP430 flash programmer system designed for a specific MSP430 target device or a system not implementing fuse-blow functionality may require fewer relays or no relays at all. The programmer system described herein was developed with the intention that it can be used with any MSP430 flash- or FRAM-based device, across all families, including all memory access functionality, as well as fuse-blow capability.

Note

Never unplug the JTAG cable during an active connection to the target device. Make sure that the debug or programming routine is completed before removing the JTAG connection.

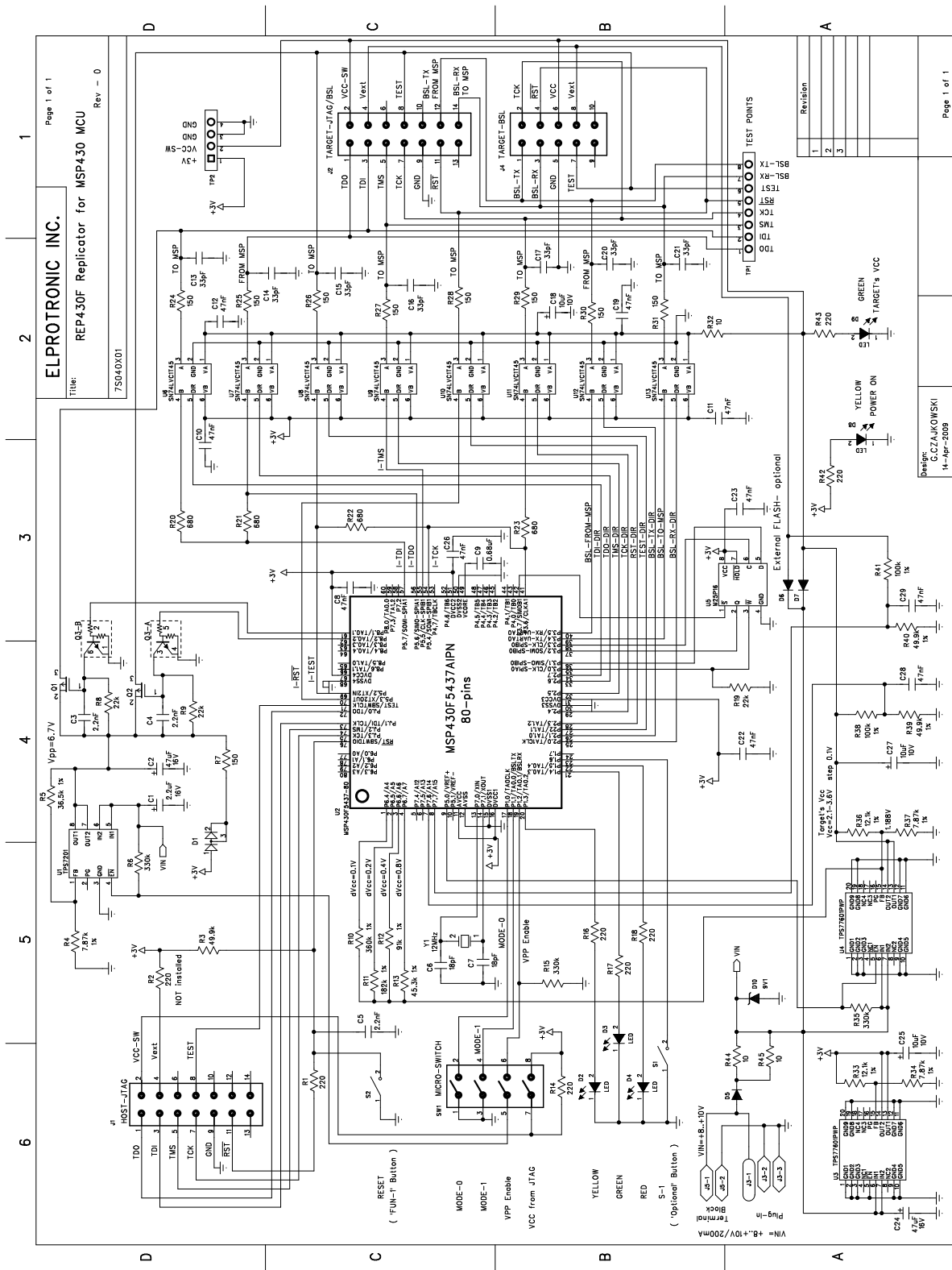


Figure 3-1. Replicator Application Schematic

3.5.3 Host Controller or Programmer Power Supply

From the input voltage of 8 V to 10 V dc, onboard voltages are generated using adjustable LDOs: U3 generates V_{CC} of 3.0 V as the supply voltage for the host controller MSP430F5437, U4 generates V_{CC} of 2.1 V to 3.6 V as the supply voltage to the target device, and U1 generates V_{PP} of 6.7 V to program the JTAG access fuse. While

the fuse is being programmed, a peak current of 100 mA can flow through the TEST or TDI input pin (see the corresponding target MSP430 device data sheet).

When using a target system that is powered locally, the V_{CC} level from the target device should be connected to pin 2 of the JTAG connector that supplies the I/O voltage translator in the REP430F. This ensures that the I/O levels of the REP430F match the I/O levels of the target device. The programmable LDO that can supply the target device should be disabled when the target device is powered locally. When pin 2 of the JTAG connector from REP430F is not connected to the target V_{CC} , a difference between the I/O voltage rails of the target device and the REP430F can occur and communication between host and target may fail due to invalid logic levels. It is also possible under these conditions that device damage can occur.

3.5.4 Third-Party Support

Elprotronic Incorporated offers a complete REP430F system that is compatible with the software available with this user's guide. Visit <http://www.elprotronic.com> for more information.

Elprotronic Inc.
35 Austin Rumble Crt.
King City
ON, L7B 0B2
Canada
Tel.: +(905)539-0424
Fax: +(905)539-0474
E-mail: info@elprotronic.com
Web site: www.elprotronic.com

4 Errata and Revision Information

4.1 Known Issues

Description

Dual port memory must be read in slow mode (word by word)

Workaround

None

4.2 Revisions and Errata From Previous Documents

The following is a summary of the errata in former revisions of *Application of Bootstrap Loader in MSP430 With Flash Hardware and Software Proposal* (SLAA096).

- Appendix D: Universal Bootstrap Loader Interface Board: Operational amplifier IC2 must be replaced with TL062D or equivalent type.

The following is a summary of changes in former revisions of *Programming a Flash-Based MSP430 Using the JTAG Interface* (SLAA149).

Version	Date	Changes/Comments
SLAA149F	October 2008	<ul style="list-style-type: none"> • Added timing requirements to Section 3.4.2. • Removed section 4.4, which was a duplicate of Section 3.7. • Added notes to Release from JTAG control sections.
SLAA149E	September 2008	<ul style="list-style-type: none"> • Added Figure 1. • Enhanced Section 2.3.3. • Changed Table 5 caption. • Added Table 6 for 5xx family. • Updated Section 2.4.5 with 5xx information. • Reworked and updated Section 3.1.2 with 5xx information. • Renamed Section 3.2 and moved Section 3.2.1.1 in this section. • Divided Section 3.2, Section 3.4, Section 3.5 and Section 4 in subsections for both 1xx, 2xx, 4xx and 5xx family. • Added IR_JMB_EXCHANGE to Table 4
SLAA149D	February 2008	<ul style="list-style-type: none"> • Fixed Section 3.5.1.1. The instruction IR_CNTRL_SIG_16BIT instead of IR_ADDR_16BIT was shown to be loaded before shifting in the according address value. • Updated Figure 10. • Added note about disabling interrupts to Section 2.3.2. • Referenced MSP430 Family user's guides for JTAG signal connections in Section 2.1.
SLAA149C	September 2007	<ul style="list-style-type: none"> • Added information about MSP430 JTAG restrictions, Section A.3 • Renamed bit 11 of the JTAG control signal register from PUC to POR, Section 2.4.3 • Added Section A.1 • Updated Section A.4 with description for the use of SRecord conversion tool

5 Revision History

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

Changes from November 23, 2019 to May 4, 2021	Page
• Updated the numbering format for tables, figures, and cross references throughout the document.....	3
• Added links to ready-to-use tools in Section 1.1 About This Document	3
• Added bullet point to Section 2.1.1	4

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), 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, 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 (<https://www.ti.com/legal/termsofsale.html>) or other applicable terms available either on [ti.com](https://www.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.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2021, Texas Instruments Incorporated