

# Programming a Flash-Based MSP430 Using the JTAG Interface

Markus Koesler, Wolfgang Lutsch

MSP430

## ABSTRACT

This application report details the functions required to erase, program, and verify the memory module of the MSP430 flash-based microcontroller family using the JTAG communication port, as well as how to program the JTAG access security fuse, available on all MSP430 devices. Device access using standard 4-wire JTAG and 2-wire JTAG, also referred to as Spy-Bi-Wire (SBW), is discussed. In addition, an example programmer system, including software (source code is provided) and the corresponding hardware implementation, is demonstrated in [Appendix A](#). This example is intended as a reference for further understanding of the concepts presented in this report and to help aid in development of similar MSP430 programmer solutions.

## Contents

1	Introduction .....	2
2	Interface and Instructions.....	3
2.1	JTAG Interface Signals .....	3
2.2	JTAG Access Macros .....	4
2.3	SBW Timing and Control .....	7
2.4	JTAG Communication Instructions.....	10
3	Memory Programming Control Sequences .....	15
3.1	Start-Up.....	15
3.2	General Functions .....	18
3.3	Accessing Non-Flash Memory Locations With JTAG .....	21
3.4	Programming the Flash Memory (Using the Onboard Flash Controller).....	24
3.5	Reading From Flash Memory .....	26
3.6	Verifying the Flash Memory.....	26
3.7	Erasing the Flash Memory (Using the Onboard Flash Controller).....	26
4	Programming the JTAG Access Protection Fuse .....	30
4.1	Standard 4-Wire JTAG .....	31
4.2	Fuse-Programming Voltage Via SBW .....	31
4.3	Testing for a Successfully Programmed Fuse .....	32
5	JTAG Function Prototypes .....	33
5.1	Low-Level JTAG Functions .....	33
5.2	High-Level JTAG Routines.....	34
6	References .....	37
7	Third-Party Support .....	37
Appendix A	Implementation .....	38
Appendix B	MSP430 JTAG Implementation .....	45

## List of Figures

1	Timing Example for the IR_SHIFT (0x83) Instruction.....	5
2	Data Register I/O: DR_SHIFT16 (0x158B) (TDO Output is 0x55AA) .....	5
3	Address Register I/O: DR_SHIFT20 (0x12568) (TDO Output Is 0xA55AA) .....	6
4	SetTCLK .....	6

All trademarks are the property of their respective owners.

5	ClrTCLK.....	7
6	Timing Diagram (Alternative Timing).....	7
7	SBW-to-JTAG Interface Diagram.....	8
8	Synchronization of TDI/TCLK During Run-Test/Idle .....	9
9	Detailed SBW Timing Diagram.....	10
10	JTAG Access Entry Sequences (for Devices Supporting SBW).....	16
11	Fuse Check and TAP Controller Reset .....	17
12	Fuse Blow Timing .....	32
A-1	Replicator Application Schematic .....	43
B-1	TAP Controller State Machine.....	45

### List of Tables

1	Standard 4-Wire JTAG Signals .....	3
2	JTAG Signal Implementation Overview.....	4
3	JTAG Communication Macros .....	4
4	Memory Access Instructions.....	11
5	JTAG Control Signal Register .....	13
6	Shared JTAG Device Pin Functions .....	15
7	Erase/Program Minimum TCLK Clock Cycles.....	24
8	Flash Memory Parameters ( $f_{FTG} = 450 \text{ kHz}$ ).....	28
9	MSP430 Device JTAG Interface (Shared Pins).....	30
10	MSP430 Device Dedicated JTAG Interface .....	30

## 1 Introduction

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

Four main elements are presented:

[Section 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 via the JTAG interface.

[Section 3](#), Memory Programming Control Sequences, demonstrates use of the provided macros and function prototypes in a software-flow format that are used to control a target MSP430 device and program and/or erase the flash memory.

[Section 4](#), Programming the JTAG Access Protection Fuse, details the fuse mechanism used to disable memory access via JTAG to the target device's memory, eliminating the possibility of undesired memory access for security purposes.

[Appendix A](#) illustrates development of an example MSP430 flash programmer using an MSP430F149 as the host controller and includes a schematic and required software/project files. A thorough description of how to use the given implementation is also included, providing an example system that can be used directly or referenced for custom MSP430 programmer solutions.

---

**Note:** 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 the 1149.1 standard are made throughout this document. The TAP state machine is shown in [Figure B-1](#). [Section B.2](#) also lists various specialities of the MSP430 JTAG implementation which are non-compliant with IEEE Std 1149.1.

---

## 2 Interface and Instructions

This section describes the hardware connections to the JTAG interface of the MSP430 devices and the associated pin functionality used during programming. In addition, the descriptions of the software macro routines used to program a MSP430 target and the JTAG instructions used to communicate with and control the target via the JTAG interface are detailed.

### 2.1 JTAG Interface Signals

The MSP430 family supports in-circuit programming of flash memory via the JTAG port, available on all MSP430 devices. All devices support the JTAG 4-wire interface. In addition, some devices also support the next-generation optimized 2-wire JTAG (Spy-Bi-Wire) interface. Using these signals, an interface connection to 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 *MSP-FET430 Flash Emulation Tool (FET) (For Use With CCE v2.0) User's Guide (SLAU157)* or the *MSP-FET430 Flash Emulation Tool (FET) (For Use With IAR v3.x) User's Guide (SLAU138)* and the device data sheet for the connections required by a specific device.

#### 2.1.1 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 VCC when powered by the programmer. These signals are described in [Table 1](#).

**Table 1. Standard 4-Wire JTAG Signals**

Pin	Direction	Usage
TMS	IN	Signal to control the JTAG state machine
TCK	IN	JTAG clock input
TDI	IN	JTAG data input/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. To enable these pins for JTAG communication, a logic level 1 must be applied to the TEST pin. For normal operation (non-JTAG mode), this pin is internally pulled down to ground, enabling the shared pins as standard port I/O.

The TCLK signal is an input clock, which must be provided to the target device from an external source. This clock is used internally as the target device's system clock, 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 devices. Existing FET tools, as well as the software provided with this application report, implement TCLK on the TDI input pin.

---

#### 2.1.2 2-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 utilized.

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. In order to reduce the overhead of the 2-wire interface, the SBWTDIO line is shared with the RST/NMI pin of the MSP430.

Table 2 gives a general overview of MSP430 devices and their respective JTAG interface implementation.

**Table 2. JTAG Signal Implementation Overview**

Devices	TEST Pin	4-Wire JTAG	Spy-Bi-Wire
20- and 28-pin MSP430F1xx devices	YES	YES	NO
64-, 80-, and 100-pin MSP430F1xx/4xx devices	NO	YES	NO
MSP430F21x1 family	YES	YES	NO
14-, 20-, 28-, and 38-pin MSP430F2xx devices	YES	YES	YES
64-, 80-, and 100-pin MSP430F2xx devices	NO	YES	NO

## 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 need to be loaded into the JTAG instruction register, as well as when these instructions are required. The following section summarizes the macros used throughout this document and their associated functionality. See the accompanying software for more information.

**Table 3. 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

### 2.2.1 Macros for 4-Wire JTAG Interface

#### 2.2.1.1 IR\_SHIFT (8-bit Instruction)

This macro loads a desired 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) implemented on the target device. 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 1 shows how to load the ADDR\_16BIT instruction into the JTAG IR register. See Section 2.4 for a complete list of the JTAG interface communication instructions used to access the target device flash memory module.

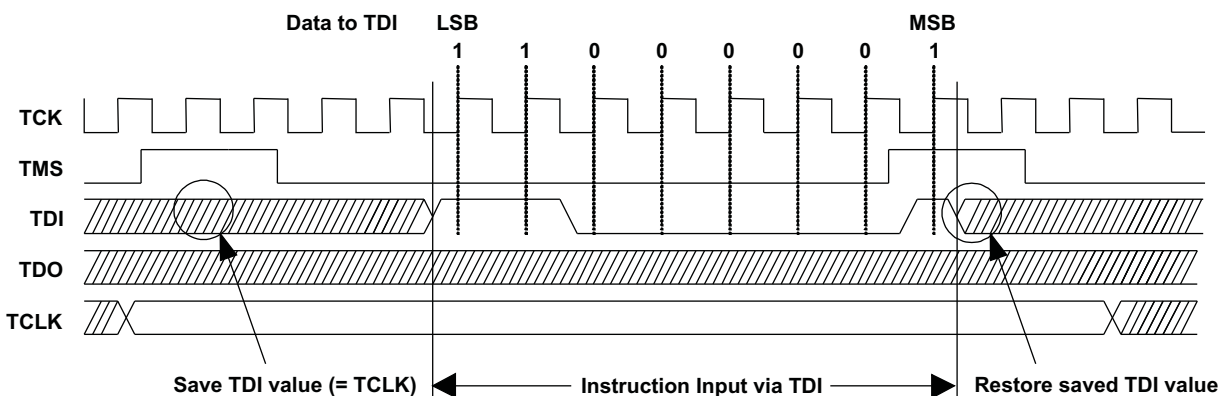


Figure 1. Timing Example for the IR\_SHIFT (0x83) Instruction

### 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, a data register is 16 bits wide.) The data word is shifted, most significant bit (MSB) first, into the target MSP430's TDI input. Each bit is captured from TDI on a rising edge of TCK. At the same time, TDO shifts out the last captured/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](#) shows how to load a 16-bit word into the JTAG DR and read out a stored value via TDO.

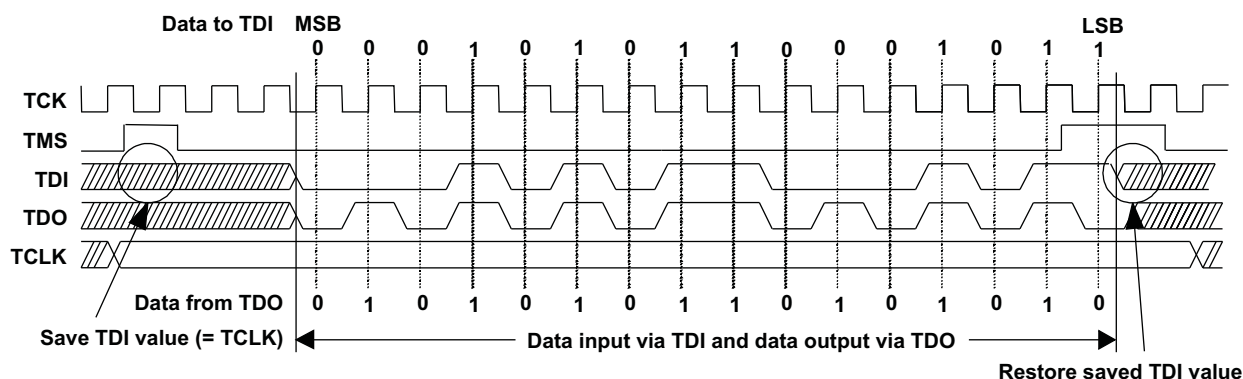


Figure 2. Data Register I/O: DR\_SHIFT16 (0x158B) (TDO Output is 0x55AA)

### 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), in order 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 2.4.1](#)), 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 target MSP430's TDI input. Each bit is captured from TDI on a rising edge of TCK. At the same time, TDO shifts out the last captured/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 have been loaded into the JTAG instruction register before the MAB is manipulated via JTAG. Note that on a 20-bit shift access, the upper four bits (19:16) of the JTAG address register are shifted out last. That means bit 15 of the MAB is read first when the lower part of the MAB is accessed by performing a 16-bit shift. This kind of 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 3 shows how to load a 20-bit address word into the JTAG address register and read out a stored value via TDO.

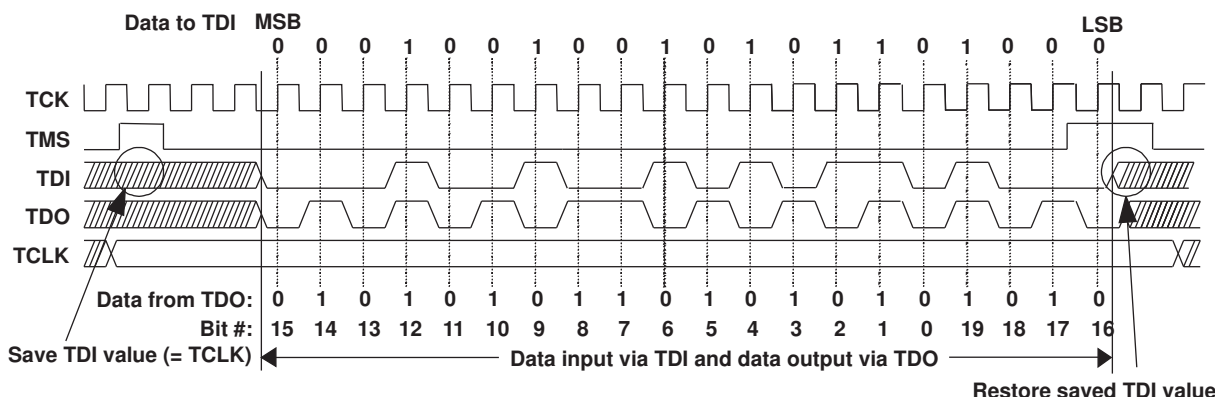


Figure 3. Address Register I/O: DR\_SHIFT20 (0x12568) (TDO Output Is 0xA55AA)

#### 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.

#### 2.2.1.5 SetTCLK

This macro sets the TCLK input clock (provided on the TDI signal input) high. TCK and TMS must hold their last value while this macro is performed (see Section 2.3.3 and Figure 8 for SBW-specific constraints).

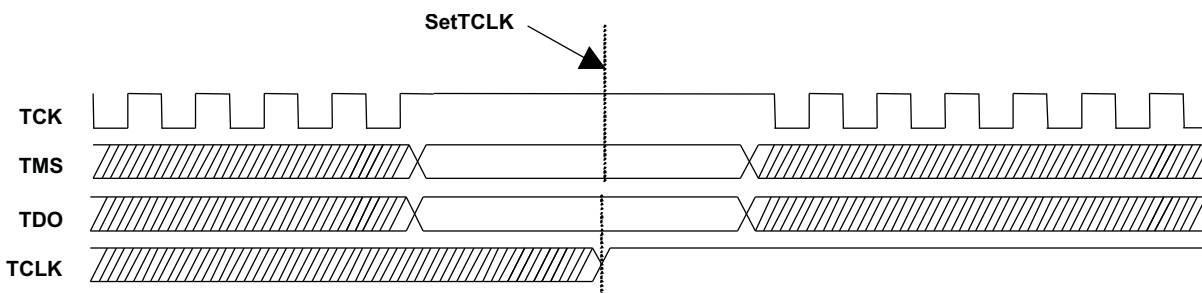


Figure 4. SetTCLK

### 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 2.3.3](#) and [Figure 8](#) for SBW-specific constraints).

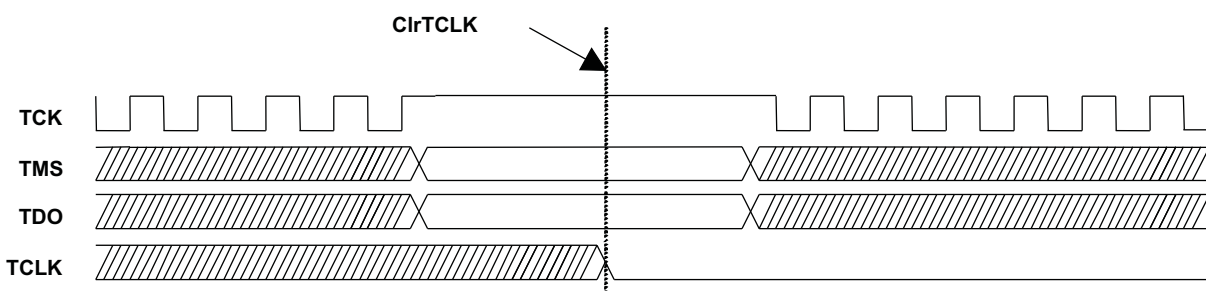


Figure 5. *ClrTCLK*

## 2.2.2 Macros for 2-Wire JTAG (SBW) Interface

All JTAG macros described in the previous section also apply to the 2-wire interface and are provided as software source along with this document.

## 2.3 SBW Timing and Control

The following sections provide a basic understanding 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.

### 2.3.1 Basic Timing

The SBW interface serial communication uses time-division multiplexing, allocating three time slots: TMS\_SLOT, TDI\_SLOT, and TDO\_SLOT. In order to clock TCLK via the SBW interface in a similar method as it is clocked via 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 clocked into the TAP controller or shift register with the rising edge of TCK as shown in [Figure 6](#).

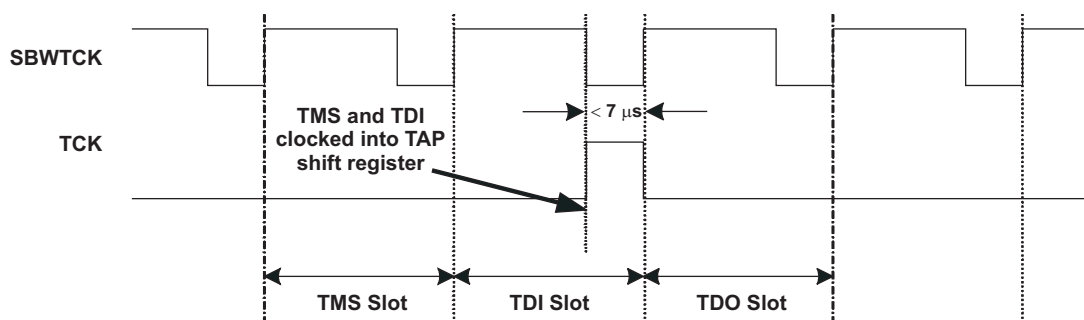
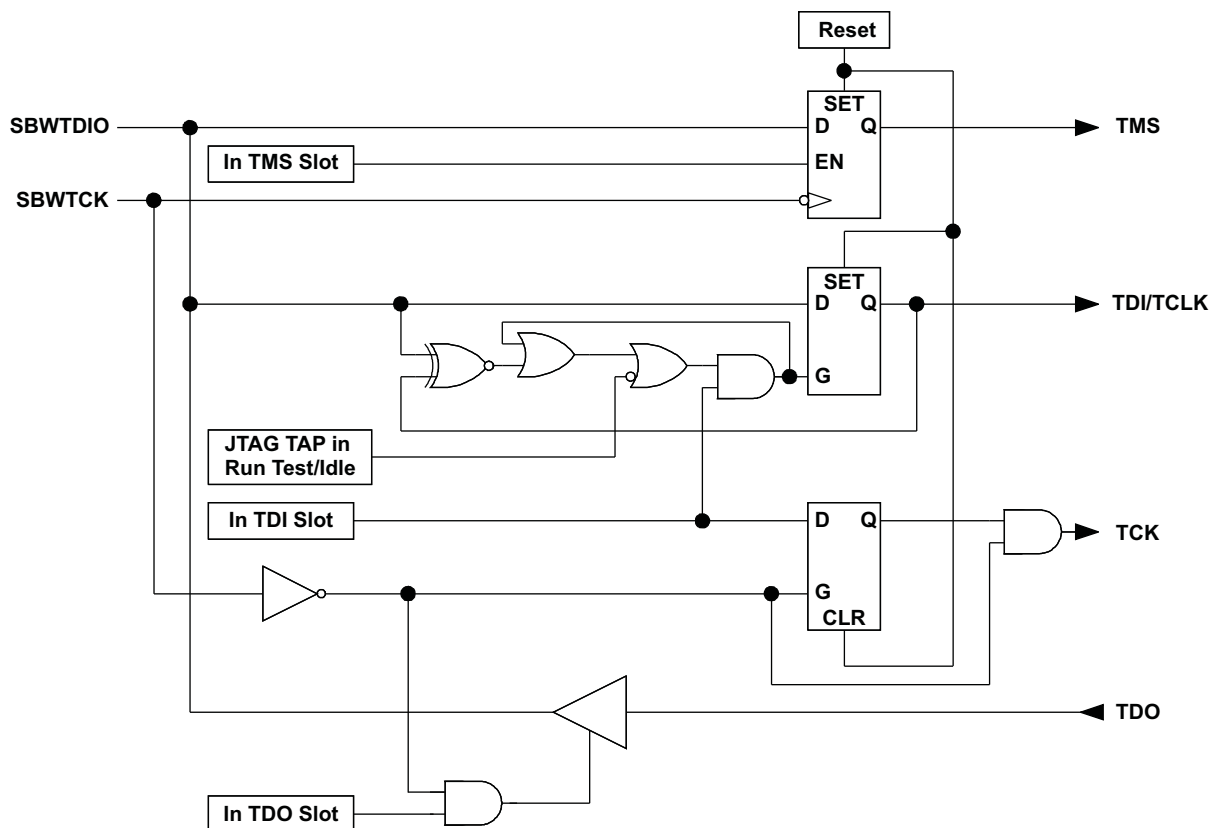


Figure 6. Timing Diagram (Alternative Timing)

The implemented logic used to translate between the 2-wire and 4-wire interfaces is shown in [Figure 7](#).



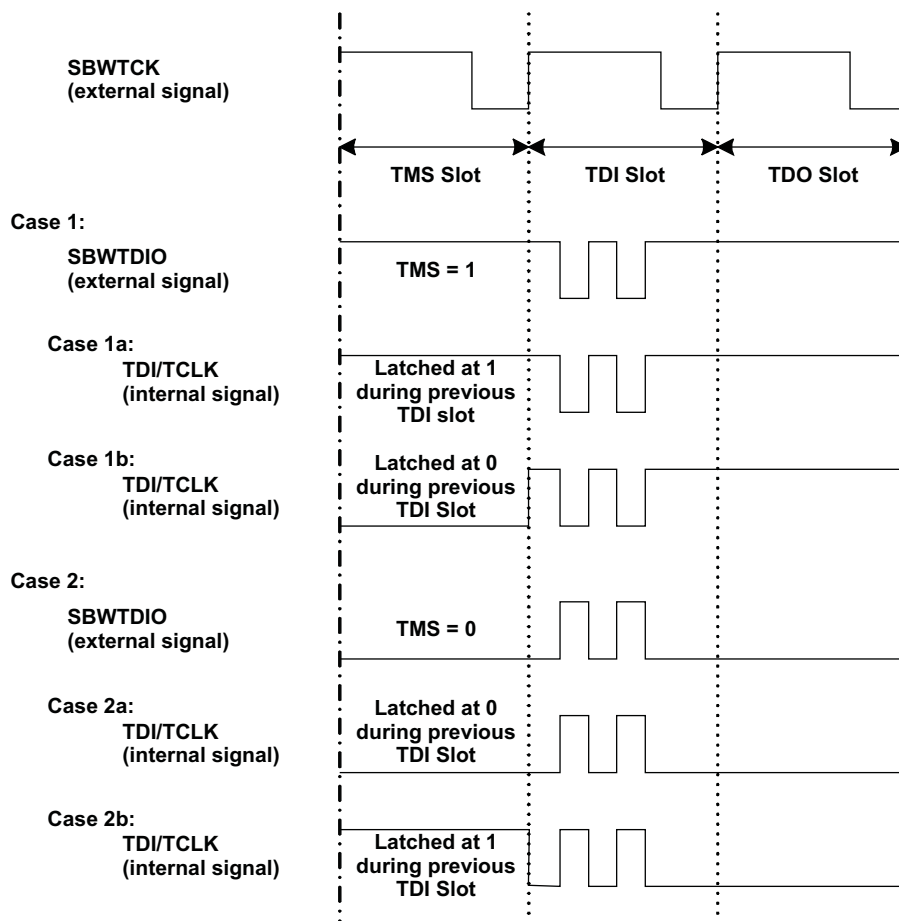


**Figure 7. SBW-to-JTAG Interface Diagram**

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 8](#). The synchronization logic is only active in the Run-Test/Idle state.





**Figure 8. Synchronization of TDI/TCLK During Run-Test/Idle**

After power up, as long as the SBW interface is not activated yet, TMS and TDI are set to logic 1 level internally.

### 2.3.2 TDO Slot

As shown in [Figure 6](#), the TDO operation is allocated one time slot (see also the detailed timing shown in [Figure 9](#)). The master should release control of the SBWTDIO line based off of the rising edge of SBWTCK of the TDI cycle. Once 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  $\mu$ s, else SBW logic is deactivated and must be activated again according to [Section 3.1](#).

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

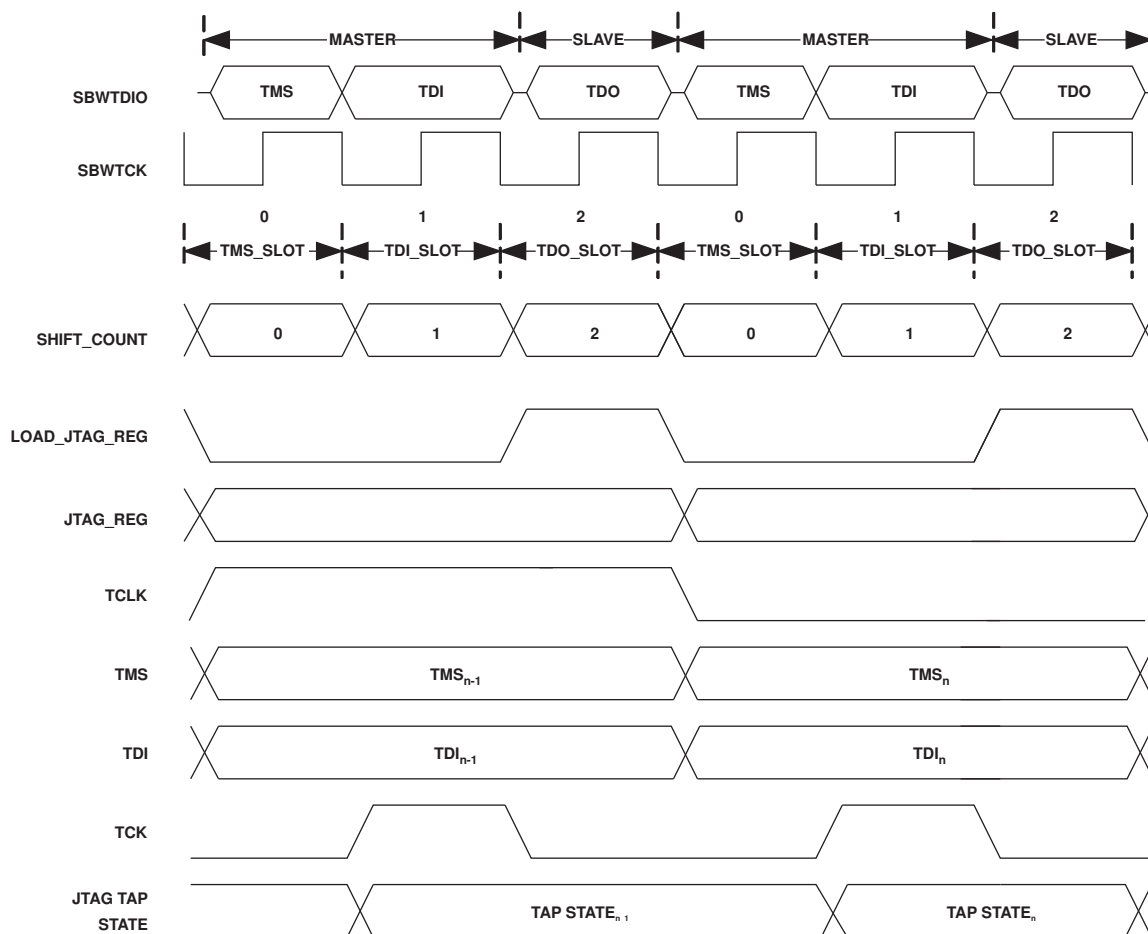


Figure 9. Detailed SBW Timing Diagram

### 2.3.3 SetTCLK\_sbwt and ClrTCLK\_sbwt in SBW Mode

Figure 8 shows handling and synchronization of TCLK in SBW mode while the JTAG TAP controller is in Run-Test/Idle state. See reference function SetTCLK\_sbwt and ClrTCLK\_sbwt for software implementation.

## 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 the previous section. The following instructions that can be written to the JTAG IR are used to program the target flash memory. All instructions sent to the target MSP430 via the JTAG register are transferred LSB first.

**Table 4. Memory Access Instructions**

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

**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](#), which always go through Run-Test/Idle state.

### 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 (20-bit in MSP430X architectures) register, termed the JTAG MAB register, is addressed. By using the JTAG data path of the TAP controller, this register can be accessed and modified.

#### 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 MSP430 CPU's MAB 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 via 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.

---

#### **2.4.1.2 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).

### **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.

#### **2.4.2.1 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 MSP430 CPU's MDB 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.

#### **2.4.2.2 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.

#### **2.4.2.3 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 in order to automatically point to the next 16-bit memory location. The target CPU's program counter must be loaded with the starting memory address prior to execution of this instruction, which can be used to quickly read or write to a memory array. (See Section 3.2 for more information on setting the PC.)

---

**Note:** IR\_DATA\_QUICK cannot be used on flash memory.

---

#### **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, which is defined in the following section, is performed simultaneously. After execution of the bypass instruction, the 16-bit data shifted out on TDI does not affect any register of the target MSP430's JTAG control module.

### 2.4.3 Controlling the CPU

The following instructions enable control of the MSP430 CPU through a 16-bit register accessed via JTAG. This data register is called the JTAG control signal register. [Table 5](#) describes the bit functions making up the JTAG control signal register used for memory access.

**Table 5. JTAG Control Signal Register**

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

#### 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.

#### 2.4.3.2 IR\_CNTRL\_SIG\_CAPTURE

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

### **2.4.3.3 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.

### **2.4.4 Memory Verification Via Pseudo Signature Analysis (PSA)**

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

#### **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 prior to execution of this instruction. The number of TCLK clocks determines how many memory locations are included in the PSA calculation.

#### **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 via the TDO pin. While this JTAG instruction is executed, the capture and update functions of the JTAG\_DATA\_REG are disabled.

### **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 MSP430 flash device. Once the fuse is programmed (or blown), future access to the MSP430 via the JTAG interface is permanently disabled. This allows for access protection of the final MSP430 firmware programmed into the target device.

#### **2.4.5.1 IR\_PREPARE\_BLOW**

This instruction sets the MSP430 into program-fuse mode.

#### **2.4.5.2 IR\_EX\_BLOW**

This instruction programs (blows) the access-protection fuse. In order to execute properly, it must be loaded after the IR\_PREPARE\_BLOW instruction is given.

## 3 Memory Programming Control Sequences

### 3.1 Start-Up

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

#### 3.1.1 Enable JTAG Access

Reference function: `GetDevice`, `GetDevice_sbwr`

- 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 VCC. For normal operation (non-JTAG mode), this pin should be released and allowed to be internally pulled to ground. [Table 6](#) shows the port 1 pins that are used for JTAG communication.

**Table 6. Shared JTAG Device Pin Functions**

Port 1 Function (TEST = Open)	JTAG Function (TEST = V <sub>CC</sub> )
P1.4	TCK
P1.5	TMS
P1.6	TDI/TCLK
P1.7	TDO

- MSP430 devices with 2-wire (SBW) JTAG access

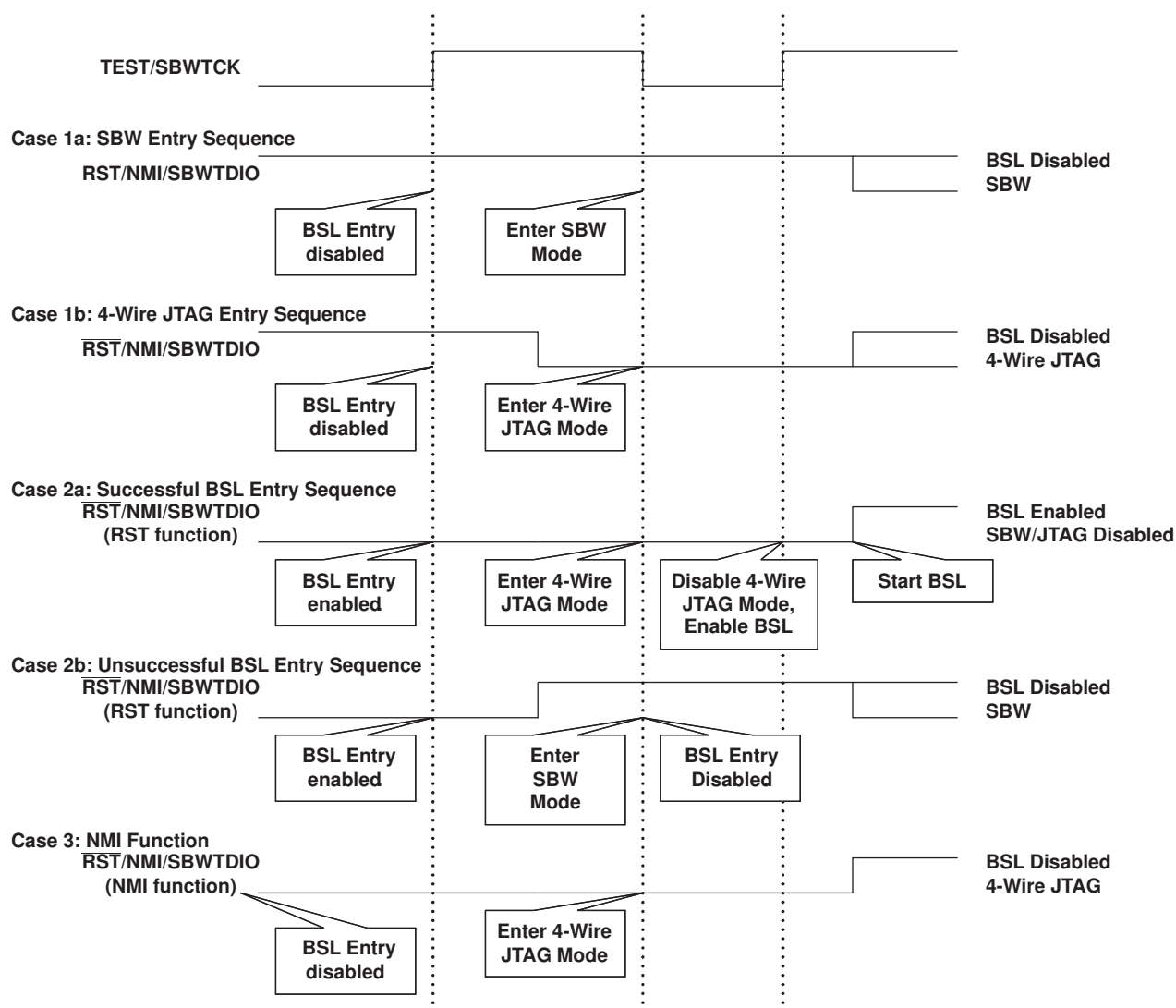
The SBW interface and any access to the JTAG interface is disabled while the TEST/SBW<sub>TCK</sub> pin is held low. This is accomplished by an internal pulldown resistor. The pin can also be tied low externally. Pulling the TEST/SBW<sub>TCK</sub> pin high enables the SBW interface and disables the  $\overline{\text{RST}}$ /NMI functionality of the  $\overline{\text{RST}}$ /NMI/SBW<sub>TDIO</sub> 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  $\overline{\text{RST}}$ /NMI with TEST/SBW<sub>TCK</sub> going high.

Devices with SBW also support the standard 4-wire interface. The 4-wire JTAG interface access is enabled by pulling the SBW<sub>TDIO</sub> line low and then applying a clock on SBW<sub>TCK</sub>. The 4-wire JTAG mode is exited by holding the TEST/SBW<sub>CLK</sub> low for more than 100  $\mu\text{s}$ .

To select the 2-wire SBW mode, the SBW<sub>TDIO</sub> line is held high and the first clock is applied on SBW<sub>TCK</sub>. 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. The SBW mode is exited by holding the TEST/SBW<sub>CLK</sub> low for more than 100  $\mu\text{s}$ .

In devices implementing the Bootstrap Loader (BSL), the TEST/SBW<sub>TCK</sub> and  $\overline{\text{RST}}$ /NMI/SBW<sub>TDIO</sub> are also used to invoke the BSL. In [Figure 10](#), different cases used to enter the SBW/JTAG or BSL mode are shown.





**Figure 10. JTAG Access Entry Sequences (for Devices Supporting SBW)**

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

Reference functions: ResetTAP, ResetTAP\_sbww

Each MSP430 family device includes a physical fuse used to permanently disable memory access via JTAG communication. When this fuse is programmed (or blown), access to memory via 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. 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.

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  $\mu$ s.

Under certain circumstances (e.g., 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 will be mistakenly recognized as programmed (e.g., blown). To avoid the issue, newer MSP430 JTAG implementations also reset the internal fuse-check logic on performing a reset of the TAP controller as previously described. Thus, it is recommended to first perform a reset of the TAP and then check the JTAG fuse status as shown in Figure 11.

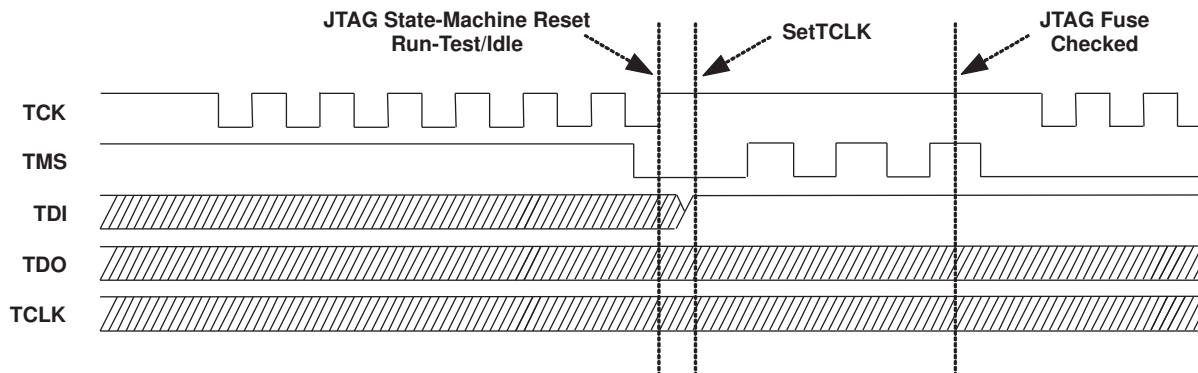


Figure 11. Fuse Check and TAP Controller Reset

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 2.3.1, 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, will end up in its Exit2-DR state. Two more dummy TCKs must be generated to get back into 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).

### 3.1.3 Taking the CPU Under JTAG Control

Reference function: GetDevice, GetDevice\_sbww

After the initial fuse check and reset, the target device's CPU 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). Once 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)	No
Bit 9 of TDOWord = 1?	
Yes	
CPU is under JTAG control	

## 3.2 General 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.

### 3.2.1 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 via 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
<b>Bit 7 of TDOvalue = 0?</b>
ClrTCLK
SetTCLK
<i>CPU is in the instruction-fetch state</i>

### 3.2.2 Setting the Target CPU Program Counter (PC)

In order 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 prior to 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
IR_SHIFT("IR_ADDR_CAPTURE")
SetTCLK
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\_x

<i>CPU must be in the instruction-fetch state prior to 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
	IR_SHIFT("IR_ADDR_CAPTURE")
	SetTCLK
	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>	

### 3.2.3 Controlled Stop/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 location(s), the CPU can be returned to normal operation. This function is implemented via the ReleaseCPU prototype and simply resets the HALT\_JTAG bit.

<i>CPU must be in the instruction-fetch state prior to the following sequence</i>	
<b>HaltCPU</b>	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>	
<b>Memory Access Performed Here</b>	
<i>The CPU is switched back to normal operation using ReleaseCPU.</i>	
<b>Release CPU</b>	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>	

### 3.2.4 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>The target device's watchdog timer must now be disabled in order to avoid an undesired reset of the target.</i>	
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	
ClrTCLK	
IR_SHIFT( "IR_CNTRL_SIG_16BIT" ) : Disable Watchdog	
DR_SHIFT16(0x2408)	: Set to Write
IR_SHIFT( "IR_ADDR_16BIT" )	
DR_SHIFT16(0x0120)	: Set Watchdog Control Register Address
IR_SHIFT( "IR_DATA_TO_ADDR" )	
DR_SHIFT16(0x5A80)	: Write to Watchdog Control Register
SetTCLK	
<i>The target CPU is now released for the next operation.</i>	
ClrTCLK	
IR_SHIFT( "IR_CNTRL_SIG_16BIT" )	
DR_SHIFT16(0x2401)	: Set to Read
IR_SHIFT( "IR_ADDR_CAPTURE" )	
SetTCLK	

### 3.2.5 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 0xFFFF.

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 0x0FFFE (reset vector).</i>	

## 3.3 Accessing Non-Flash Memory Locations With JTAG

### 3.3.1 Read Access

To read from any memory address location (peripherals, RAM, or flash), 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 via 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

<i>Set CPU to stopped state (HaltCPU)</i>	
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2409)	: Read Memory
IR_SHIFT("IR_ADDR_16BIT")	<b>Yes</b>
DR_SHIFT16("Address")	
IR_SHIFT("IR_DATA_TO_ADDR")	
SetTCLK	
ClrTCLK	
DR_SHIFT16(0x0000)	: Memory value shifted out on TDO
<b>Read again?</b>	
<b>No</b>	
<i>ReleaseCPU should now be executed, returning the CPU to normal operation.</i>	

- MSP430X architecture, Reference function: ReadMem\_x

Set CPU to stopped state (HaltCPU)		
ClrTCLK		
IR_SHIFT( "IR_CNTRL_SIG_16BIT" )		
DR_SHIFT16(0x2409)	: Read Memory	
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.		

### 3.3.2 Write Access

To write to a memory location in peripherals or RAM (but not flash), 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) : Write Memory	
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\_x

Set CPU to stopped state (HaltCPU)	
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2408) : Write Memory	
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.	

### 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.

#### 3.3.3.1 Flow for Quick Read (All Memory Locations)

Reference function: ReadMemQuick

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

### 3.3.3.2 Flow for Quick Write (RAM and Peripheral Memory Only)

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)	: Set RW to write	Yes
IR_SHIFT("IR_DATA_QUICK")		
DR_SHIFT16("Data")	: Set data	
SetTCLK		
ClrTCLK	: Auto-increments PC	
Write To Next Address?		
No		
ReleaseCPU should now be executed, returning the CPU to normal operation. Reset the target CPU's PC if needed (SetPC).		

## 3.4 Programming the Flash Memory (Using the Onboard Flash Controller)

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 utilize, which would be programmed into a production-equipment MSP430 device. Note that 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. (For more information on the flash controller timing, please see the corresponding MSP430 user's guide and specific device data sheet.) The following table shows the required minimum number of TCLK cycles, depending on the action performed on the flash (for FCTL2 register bits 0 – 7 = 0x40 as defined in the MSP430 user's guide).

**Table 7. Erase/Program Minimum TCLK Clock Cycles**

FLASH Action	Minimum TCLK Count
Segment erase	4820
Mass erase	5300 to 10600 <sup>(1)</sup>
Program word	35

<sup>(1)</sup> MSP430 device dependent, see device-specific data sheet. See [Section 3.7](#) 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) <sup>(1)</sup>	: 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) <sup>(1)</sup>	: 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) <sup>(1)</sup>	: Point to FCTL3 Address	
IR_SHIFT("IR_DATA_TO_ADDR")		
DR_SHIFT16(0xA500) <sup>(2)</sup>	: 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") <sup>(1)</sup>	: 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 <sup>(3)</sup>		
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) <sup>(1)</sup>	: Point to FCTL1 Address	

<sup>(1)</sup> Replace with DR\_SHIFT20("Address") when programming an MSP430X architecture device.

<sup>(2)</sup> Substitute 0xA540 for '2xx devices for Info-Segment A programming.

<sup>(3)</sup> Correct timing required. Must meet min/max TCLK frequency requirement of 350 kHz  $\pm$  100 kHz.

IR_SHIFT("IR_DATA_TO_ADDR")
DR_SHIFT16(0xA500) : Disable FLASH Write Access
SetTCLK
ClrTCLK
IR_SHIFT("IR_ADDR_16BIT")
DR_SHIFT16(0x012C) <sup>(1)</sup> : Point to FCTL3 Address
IR_SHIFT("IR_DATA_TO_ADDR")
DR_SHIFT16(0xA500) <sup>(2)</sup> : Disable FLASH Write Access
SetTCLK
<i>ReleaseCPU should now be executed, returning the CPU to normal operation.</i>

### 3.5 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.

### 3.6 Verifying the Flash 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  $\approx 23$  ms/4 kB.

### 3.7 Erasing the Flash Memory (Using the Onboard Flash Controller)

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 0–7 = 0x40).

#### 3.7.1 Flow to Erase a Flash Memory Segment

<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) <sup>(1)</sup> : 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) <sup>(1)</sup> : Point to FCTL2 Address	
IR_SHIFT("IR_DATA_TO_ADDR")	

<sup>(1)</sup> Replace with DR\_SHIFT20("Address") when programming an MSP430X architecture device.

DR_SHIFT16(0xA540)	: Source is MCLK, divider by 1
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x012C) <sup>(1)</sup>	: Point to FCTL3 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500) <sup>(2)</sup>	: Clear FCTL3 Register
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16("EraseAddr") <sup>(1)</sup>	: Set Address for Erase <sup>(3)</sup>
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 <sup>(4)</sup>
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2408)	: Set RW to Write
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x0128) <sup>(1)</sup>	: 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) <sup>(1)</sup>	: Point to FCTL3 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500) <sup>(2)</sup>	: Disable FLASH Write Access
SetTCLK	
ReleaseCPU should now be executed, returning the CPU to normal operation.	

<sup>(2)</sup> Substitute 0xA540 for '2xx devices for Info-Segment A programming.

<sup>(3)</sup> The EraseAddr parameter is the address pointing to the flash memory segment to be erased.

<sup>(4)</sup> Correct timing required. Must meet min/max TCLK frequency requirement of 350 kHz  $\pm$ 100 kHz.

### 3.7.2 Flow to Erase the Entire Flash Address Space (Mass Erase)

Beside the TCLK signal at a frequency of 350 kHz  $\pm$  100 kHz (used for the Flash Timing Generator, data sheet parameter  $f_{\text{FTG}}$ ), 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  $t_{\text{CMERase}}$  (cumulative mass erase time) and the second is  $t_{\text{Mass Erase}}$  (mass erase time). Two different specification combinations of these parameters are currently implemented in dedicated MSP430 devices. Table 8 shows an overview of the parameters (assuming a maximum TCLK frequency of 450 KHz).

**Table 8. Flash Memory Parameters ( $f_{\text{FTG}} = 450 \text{ kHz}$ )**

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

For implementation 1, in order 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.

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) <sup>(2)</sup>	: 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) <sup>(2)</sup>	: 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) <sup>(2)</sup>	: FCTL3 address
IR_SHIFT("IR_DATA_TO_ADDR")	

Perform once or  
Repeat 19 times<sup>(1)</sup>

<sup>(1)</sup> Correct timing required. Must meet min/max TCLK frequency requirement of 350 kHz  $\pm$  100 kHz.

<sup>(2)</sup> Replace with DR\_SHIFT20("Address") when programming an MSP430X architecture device.

DR_SHIFT16(0xA500) <sup>(3)</sup>		: Clear FCTL3 register	Perform once or Repeat 19 times <sup>(1)</sup>
SetTCLK			
ClrTCLK			
IR_SHIFT("IR_ADDR_16BIT")			
DR_SHIFT16("EraseAddr") <sup>(2)</sup>		: Set address for erase <sup>(4)</sup>	
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 <sup>(1)</sup>		
ClrTCLK			
IR_SHIFT("IR_CNTRL_SIG_16BIT")			
DR_SHIFT16(0x2408)		: set RW to write	
IR_SHIFT("IR_ADDR_16BIT")			
DR_SHIFT16(0x0128) <sup>(2)</sup>		: FCTL1 address	
IR_SHIFT("IR_DATA_TO_ADDR")			
DR_SHIFT16(0xA500)		: Disable FLASH erase	
SetTCLK			
ClrTCLK			
IR_SHIFT("IR_ADDR_16BIT")			
DR_SHIFT16(0x012C) <sup>(2)</sup>		: Point to FCTL3 Address	
IR_SHIFT("IR_DATA_TO_ADDR")			
DR_SHIFT16(0xA500) <sup>(3)</sup>		: Disable FLASH Write Access	
SetTCLK			
ReleaseCPU should now be executed, returning the CPU to normal operation.			

<sup>(3)</sup> Substitute 0xA540 for '2xx devices for Info-Segment A programming.

<sup>(4)</sup> 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.



## 4 Programming the JTAG Access Protection Fuse

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 9. MSP430 Device JTAG Interface (Shared Pins)**

Pin	Direction	Usage
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 10. MSP430 Device Dedicated JTAG Interface**

Pin	Direction	Usage
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  $6.5\text{ V} \pm 0.5\text{ V}$ .

---

## 4.1 Standard 4-Wire JTAG

Reference function: BlowFuse

### 4.1.1 Fuse-Programming Voltage Via TDI Pin (Dedicated JTAG Pin Devices Only)

When the fuse is being programmed,  $V_{PP}$  is applied via the TDI input. Communication data that is normally sent on TDI is sent via TDO during this mode. (Table 10 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 <math>V_{PP}</math> to TDI pin</i>
<i>Wait until <math>V_{PP}</math> input has settled (depends on <math>V_{PP}</math> source)</i>
IR_SHIFT("IR_EX_BLOW") : Sent to target via TDO
MsDelay(1) : Delay for 1ms
<i>Remove <math>V_{PP}</math> from TDI pin</i>
<i>Switch TDI pin back to TDI function and reset the JTAG state machine (ResetTAP)</i>

### 4.1.2 Fuse-Programming Voltage Via 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 <math>V_{PP}</math> to TEST pin</i>
<i>Wait until <math>V_{PP}</math> input has settled (depends on <math>V_{PP}</math> source)</i>
IR_SHIFT("IR_EX_BLOW")
MsDelay(1) : Delay for 1ms
<i>Remove <math>V_{PP}</math> from TEST pin</i>
<i>Reset the JTAG state machine (ResetTAP)</i>

## 4.2 Fuse-Programming Voltage Via SBW

Reference function: BlowFuse\_sbww

In SBW mode, the TEST/SBWTCK pin is used to apply fuse-blow voltage  $V_{PP}$ . The required timing sequence is shown in Figure 12. The actual fuse programming happens in the Run-Test/Idle state of the TAP controller. After the IR\_EX\_BLOW instruction is shifted in via SBW, one more TMS\_SLOT must be performed. Then a stable  $V_{PP}$  must be applied to SBWTCK. 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.

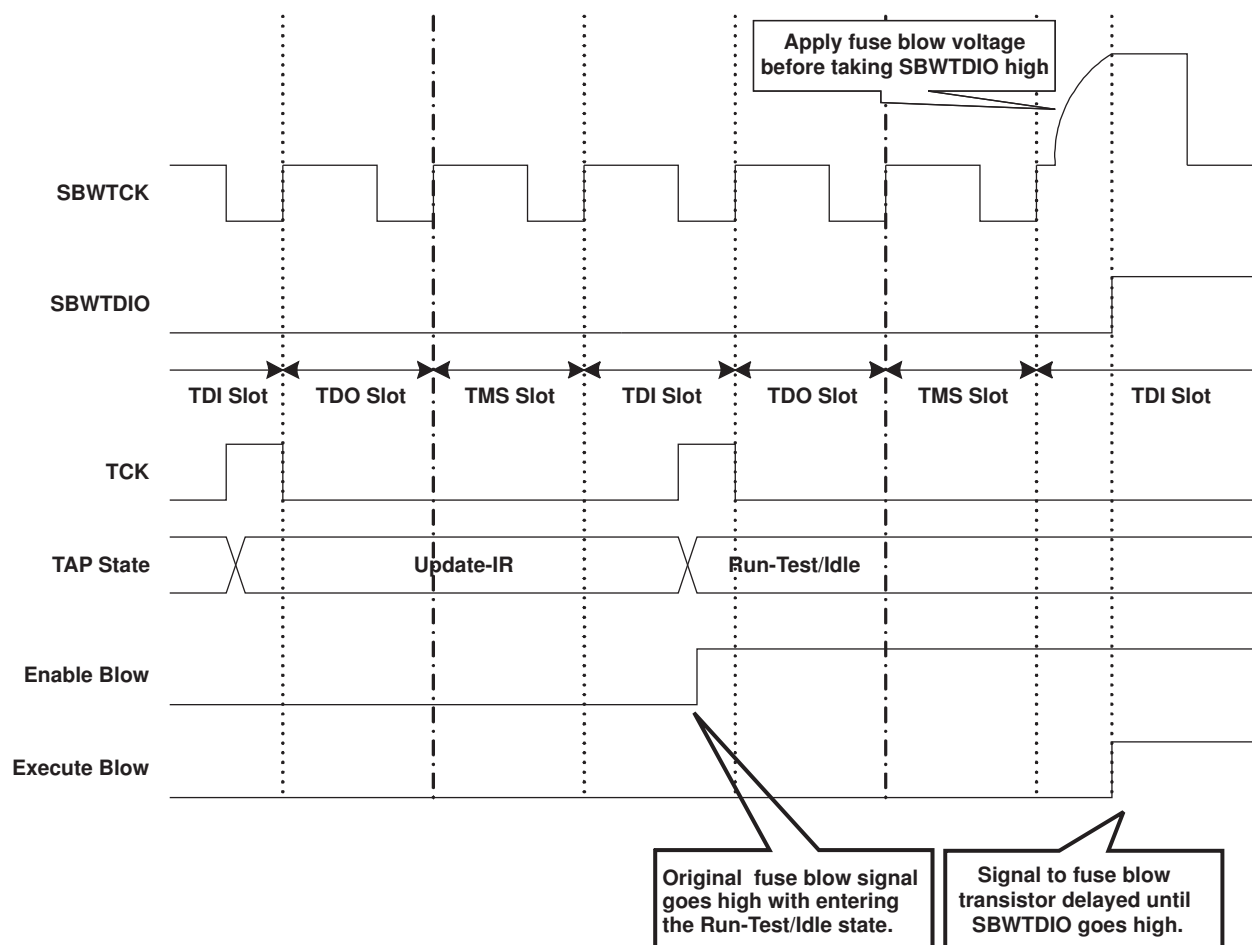


Figure 12. Fuse Blow Timing

### 4.3 Testing for a Successfully Programmed Fuse

Reference function: IsFuseBlown

Once the fuse is programmed and a RESET (via the JTAG ExecutePOR command or the RST/NMI pin in hardware) has been issued, the only JTAG function 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 output on TDO, where it can be received by other devices downstream of the target MSP430.

To test a device for a programmed fuse, access to any JTAG data register can be attempted. In the following communication sequence, the JTAG CNTRL\_SIG register is accessed.

Initialize JTAG access (ResetTAP)	
IR_SHIFT("IR_CNTRL_SIG_CAPTURE")	
DR_SHIFT16(0xAAAA)	
Is TDO output value = 0x5555?	
Yes: Fuse IS programmed	No: Fuse NOT programmed

## 5 JTAG Function Prototypes

### 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 via the JTAG control signal register. This function also disables the target device's watchdog timer in order 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

**static 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)

## 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 (0xFFFF: 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 (word \*DataArray)**

Programs/verifies a set of arrays of words into flash memory by using the WriteFLASH() function. It conforms to the CodeArray structure convention of the target device program file: Target\_Code.txt. (See Appendix A for more information on file structure.)

Arguments: word \*CodeArray (pointer to an array set containing the data)  
Result: word (STATUS\_OK if write/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)**

Performs 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)

**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)



## 6 References

MSP430Fxxx device data sheets

*MSP430x1xx Family User's Guide*, literature number [SLAU049](#)

*MSP430x4xx Family User's Guide*, literature number [SLAU056](#)

*MSP430x2xx Family User's Guide*, literature number [SLAU144](#)

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

## 7 Third-Party Support

SoftBaugh, Inc., offers a complete system as shown in [Appendix A](#), which is compatible with the software available with the *MSP430 Flash Programming Replicator* application report. This information can be found at this address: <http://www.softbaugh.com/ExtREP430.html>

SoftBaugh, Inc.  
5400 Laurel Springs Parkway  
Suite 1001  
Suwanee GA 30024  
Tel.: 800-794-5756  
Fax: 770-886-1777  
E-mail: [info@softbaugh.com](mailto:info@softbaugh.com)  
Web site: [www.softbaugh.com](http://www.softbaugh.com)

## Appendix A Implementation

### A.1 Implementation History

There are two Replicator implementations. The latest version is discussed in this document, while the previous version is found in a separate file, slaa149b.zip. The main difference between the two implementations is the use of the srec\_cat.exe function in place of FileMaker.exe. Corresponding changes to function calls and declarations were made to the Replicator.c file. While the implementation described in this document is preferred, the previous implementation is maintained for legacy users.

### A.2 Implementation Overview

The following sections document the examples provided via .zip file along with this application report. Each example demonstrates the software functions described in the previous sections using an MSP430F149 as the host controller that programs the given target MSP430 flash-based device of choice. The complete C source code and project files are provided in the attachment accompanying this application report. 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-based devices. There are specific software projects for the following target device Replicator implementations:
  - Replicator: All 4-wire JTAG, MSP430 architecture devices (includes SBW devices when programmed in 4-wire mode)
  - Replicator for Spy-Bi-Wire: 2-wire interface implementation for SBW devices only
  - Replicator for MSP430X: For 4-wire MSP430X extended architecture devices only

---

**Note:** The Replicator source files are provided in independent folders with the same names as previously given. Within these folders, filenames are assigned accordingly when applicable specifically to a certain device type. For example, the file JTAGfunc.c used in the Replicator version, is renamed JTAGfuncSBW.c in the Replicator for SBW version and JTAGfunc430X.c in the Replicator for MSP430X version.

---

- Maximum target device program code size: approximately 57 KB
- MSP430X target device source code must be placed in lower 64-kB address space.
- Programming speed (Erase, Program, Verify): approximately 8 KB in 1.5 s, 48 KB in 8 s
- Fast verify and erase check: 17 KB/10 ms
- Support programming of the JTAG access fuse (permanently disables device memory access via JTAG)
- Stand-alone target programming operation (no personal computer or additional supporting hardware/software required)

### A.3 Software Operation

The host controller stores the JTAG communication protocol code and the target program in its flash memory (61 KB available on the MSP430F149). The programming software itself occupies about 3.5 KB, so approximately 57 KB remain for the target device program. The Replicator host can be loaded with the target source code via the flash emulation tool (FET) or the MSP430 serial programming adapter. (See the MSP430 website at [www.ti.com](http://www.ti.com) for more information on device programming tools.)

The basic functionality of the programmer is as follows. Pushing the GO button generates a hardware reset and starts the host controller's JTAG communication routine in order to erase, program, and verify the target device. While the system is active, two LEDs on the programmer board are on; after successful completion, only the green LED is on. If an error occurs or communication to the target device fails, only the red LED remains on. The entire procedure takes approximately 3 s for a target program size of 8 KB. (Some code not strictly required to erase/program/verify the target MSP430 is executed at the end of the Replicator.c source file, increasing the specified programming times. These additional instructions can be customized to fit the individual system programming requirements.)

To achieve optimum performance, the JTAG communication protocol uses the SPI module on the host MSP430F149 for the basic JTAG data shift function. To simplify code portability to alternative host platforms, this shift function is also provided in the attached code as a software loop using the general-purpose I/O port functionality as an alternative. See the included source code file LowLevelFunc.h, LowLevelFuncSBW.h, and LowLevelFunc430X.h.

To program the host MSP430F149 different development environments can be used—IAR Embedded Workbench™ or CCE™ by Texas Instruments. The free versions of IAR and CCE impose code size restrictions. In order to use the 57 KB previously mentioned, the full version of IAR or CCE is needed. The folder structure provides both an IAR and CCE 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 CCE, 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.

## A.4 Software Structure

The programming software is partitioned in three levels and consists of eight files in addition to the target program (see below):

Top level	Specifies which programming functions (erase, program, verify, blow fuse) are to be executed.	
	Replicator.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 in order 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.
	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 simply replace the content of Target_Code.h by the output of srec_cat.exe and remove Target_Code.s43 (IAR) resp. Target_Code.asm (CCE) from the project. The Target_Code.h file is generated by the srec_cat.exe file directly or via the srec.bat file.
JTAG functions	All MSP430-specific functions are defined here. These files should not be modified under any circumstance.	
	JTAGfunc.c JTAGfuncSBW.c JTAGfunc430X.c	Contain the MSP430-specific functions needed for flash programming
	JTAGfunc.h JTAGfuncSBW.h JTAGfunc430X.h	Contain constant definitions and function prototypes used for JTAG communication
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 MSP430F149 is implemented.	
	LowLevelFunc.c LowLevelFuncSBW.c LowLevelFunc430X.c	Contain the basic host-specific functions
	LowLevelFunc.h LowLevelFuncSBW.h LowLevelFunc430X.h	Contain host-specific definitions and function prototypes
Devices	Describes features and differences between MSP430 devices with respect to FLASH programming.	
	Devices.c	Functions to distinguish MSP430 devices concerning FLASH programming.
	Devices.h	Device function prototypes and definitions for FLASH programming.

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 (e.g., Target\_Code.s43 (IAR) or Target\_Code.asm (CCE)), which contains the target code in assembly format, or replace the C-Array in the Target\_Code.h header file. Both alternatives must conform to the format expected by the slaa149 source code.

To build these files from the TI-txt format output from the compiler, a conversion program called `srec_cat.exe` and a batch file, `srec.bat`, are provided. 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). This can also be done in CCE using the `hex430` command line executable. `srec_cat.exe` is a command line application which 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
'srec_cat.exe Target_Code.txt -ti_txt -Output Target_Code.s43 -asm -output_word -a430' (IAR)
resp.
'srec_cat.exe Target_Code.txt -ti_txt -Output Target_Code.asm -asm -output_word -cl430' (CCE)
```

Parameter description:

- '`srec_cat.exe`': The name of the application
- '`Target_Code.txt -ti_txt`': This is the input file by name and the format of it
- '`-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 the input file should be converted in. For the `slaa149` application report 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 will not be fill any address gap with a `0xFF` pattern, what finally will 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 CCE in the Replicator context.

The `srec.bat` file generates all three types of output files (`.h`, `.asm`, and `.s43`) simultaneously. The command line format is: '`srec Target_Code`'.

---

**Note:** If the TI-txt source file includes odd segment addresses and/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 provided conversion program is Open Source and has a much larger range of functions. For more information and documentation see <http://srecord.sourceforge.net/>. Additionally, this software was tested to function correctly with version 1.36, but will not necessarily be 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.

---

## A.5 Programmer Operation

The following is a step-by-step procedure that demonstrates how the JTAG Replicator programmer could be used together with any MSP430 FETXXX development tool using the IAR MSP430 development environment.

## A.6 Hardware Setup

The hardware consists of the host controller MSP430F149, five semiconductor relays, two voltage regulators and two JTAG interface connectors. An external power supply delivering 8 V to 10 V dc at 200 mA is required for operation (see [Figure A-1](#)).

### A.6.1 Host Controller

To achieve maximum programming speed, the host controller MSP430F149 runs at a maximum CPU clock frequency of 8 MHz, provided on LFXT1. CPU operation at this frequency requires a supply voltage of 3.6 V for the host controller, which is provided by U2 in the schematic. The host is programmed via a dedicated JTAG port labeled Host JTAG (see [Figure A-1](#)).

### A.6.2 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 tools (FET and PRGS tools). The host supply voltage of 3.6 V is also available on pin 2 of this connector, eliminating the need for an additional supply for the target system, but it does not have to be used at the target. 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.)

To enable programming of all MSP430 flash-based devices including a JTAG access fuse, five semiconductor relays are used, which are controlled by the host MSP430. Relay U4 controls  $V_{PP}$  on devices with a TEST pin; U5 connects  $V_{PP}$  to TDI on devices not requiring a TEST signal. 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). U8 controls availability of  $V_{CC}$  to the target device. The host controller program includes delays, which consider a relay switching time of a maximum of 5 ms. U4 and U5 should have a  $R_{ON} < 1 \Omega$  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  $\mu$ s into the TDI pin (or the TEST pin, depending on the target device).

The following recommended relays meet the above requirements:

NAIS	AQV212 (as shown in <a href="#">Figure A-1</a> )
NEC	PS710A
Matsushita	AQV251
Matsushita	AQY211EH
CP Clare	LCA710

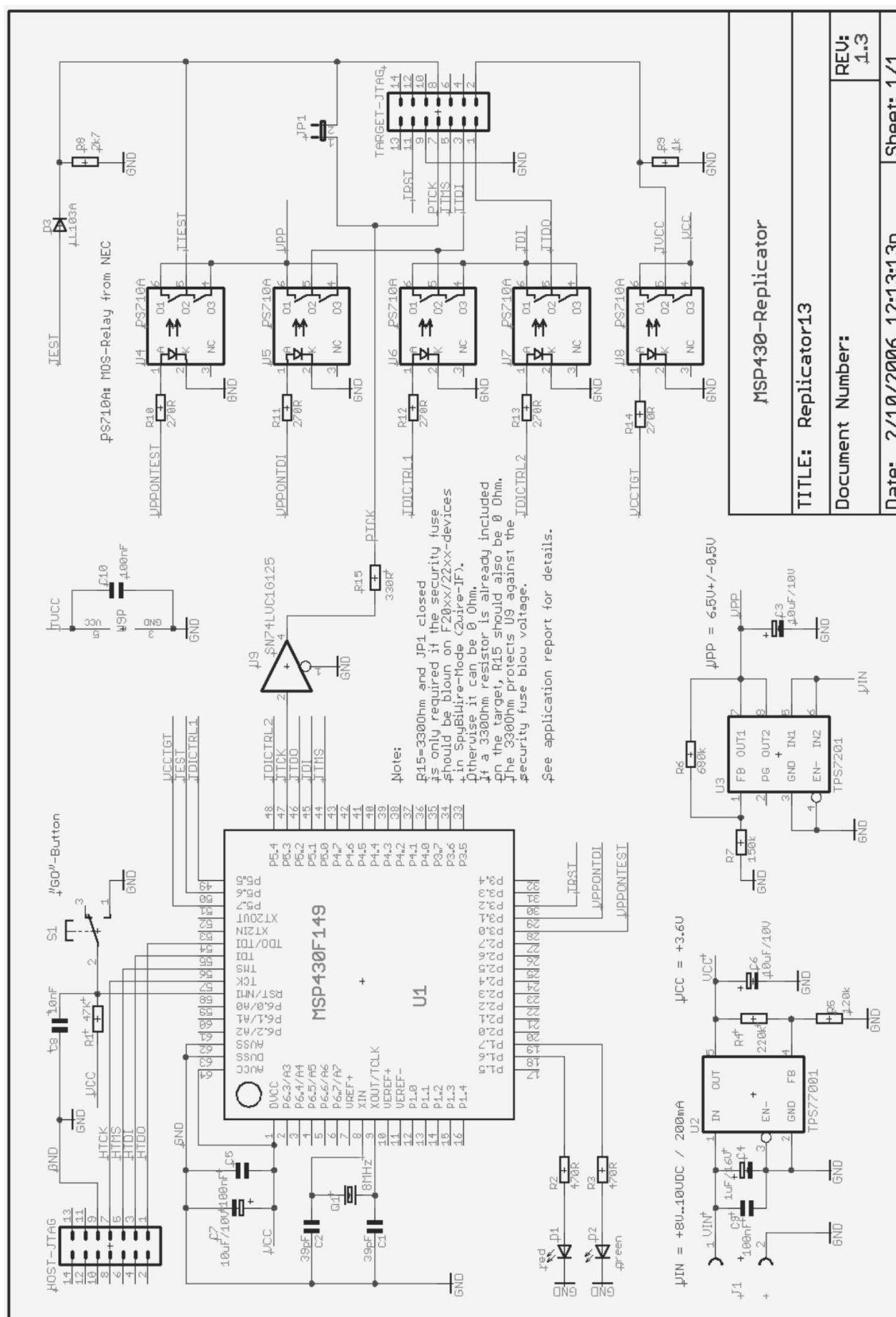


Figure A-1. Replicator Application Schematic

---

**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-based device, across all families, including all memory access functionality, as well as fuse-blow capability.

---

### **A.6.3 Host Controller/Programmer Power Supply**

From the input voltage of 8 V to 10 V dc, two onboard voltages are generated using adjustable LDOs:  $V_{CC} = 3.6$  V as supply voltage for the host controller MSP430F149 and target device, and  $V_{PP} = 6.5$  V to program the JTAG access fuse. While the fuse is being programmed, a peak current of 100 mA can flow through the TEST input pin (see the corresponding target MSP430 device data sheet).

If the target hardware requires a supply voltage lower than 3.6 V, the  $V_{CC}$  output of the programmer can be reduced accordingly (the host controller's CPU crystal frequency should be reduced as well), or level shifters can be added to translate the required supply voltage to the desired level.

When using a target system that is powered locally, the  $V_{CC}$  level of the host programmer should match that locally at the target. When differences exist between these voltage rails, communication between host and target may fail due to invalid logic levels. It is also possible under these conditions that device damage can occur.



## Appendix B MSP430 JTAG Implementation

### B.1 TAP Controller State Machine

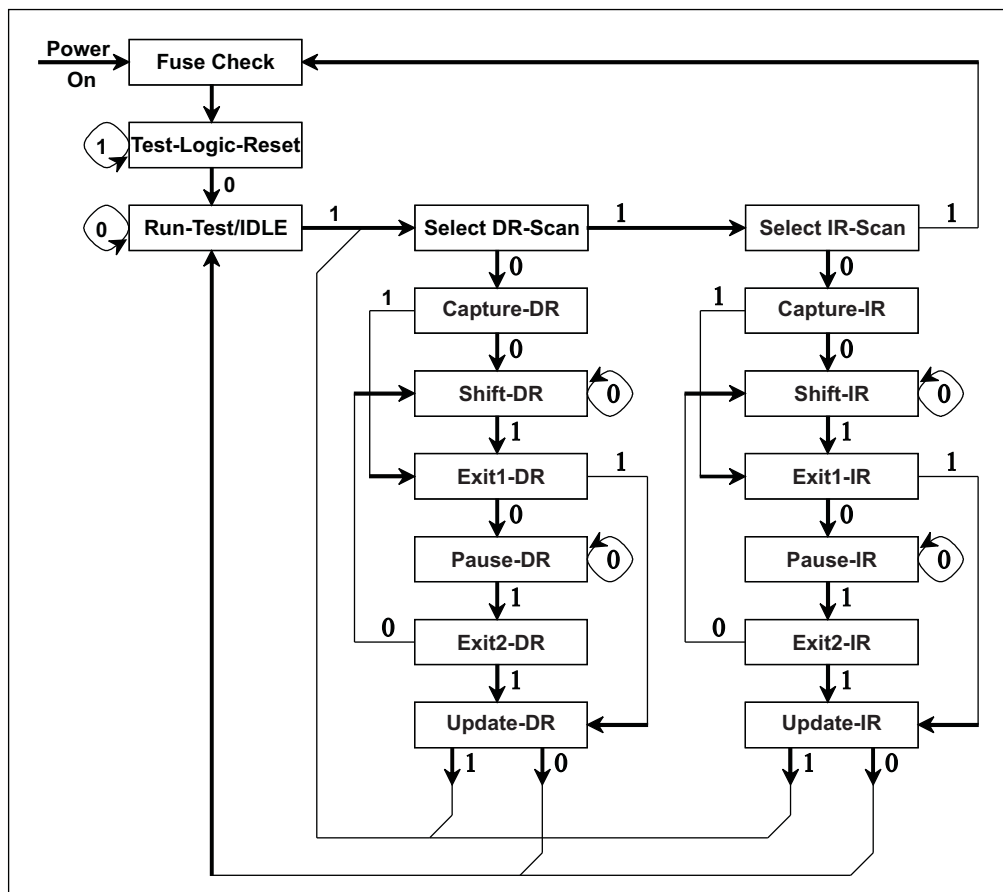


Figure B-1. TAP Controller State Machine

### B.2 MSP430 JTAG Restrictions (Non-Compliance With IEEE Std 1149.1)

- The MSP430 device must be the first device in the JTAG chain (because of clocking via TDI and JTAG fuse check sequence).
- Only the BYPASS instruction is supported. There is no support for SAMPLE, PRELOAD, or EXTEST instructions.
- The JTAG pins are shared with port functions on certain devices – JTAG function controlled via TEST pin.

### Document Revision History

Version	Date	Changes/Comments
SLAA149D	February 2008	<ul style="list-style-type: none"> <li>Fixed <a href="#">Section 3.7.1</a>. The instruction IR_CNTRL_SIG_16BIT instead of IR_ADDR_16BIT was shown to be loaded before shifting in the according address value.</li> <li>Updated <a href="#">Figure 8</a>.</li> <li>Added note about disabling interrupts to <a href="#">Section 2.3.2</a>.</li> <li>Referenced MSP430 Family user's guides for JTAG signal connections in <a href="#">Section 2.1</a>.</li> </ul>
SLAA149C	September 2007	<ul style="list-style-type: none"> <li>Added information about MSP430 JTAG restrictions, <a href="#">Section A.3</a></li> <li>Renamed bit 11 of the JTAG control signal register from PUC to POR, <a href="#">Section 2.4.3</a></li> <li>Added <a href="#">Section A.1</a></li> <li>Updated <a href="#">Section A.4</a> with description for the usage of SRecord conversion tool</li> </ul>

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

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

### Products

Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>
Clocks and Timers	<a href="http://www.ti.com/clocks">www.ti.com/clocks</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>
RFID	<a href="http://www.ti-rfid.com">www.ti-rfid.com</a>
RF/IF and ZigBee® Solutions	<a href="http://www.ti.com/lprf">www.ti.com/lprf</a>

### Applications

Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Automotive	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
Broadband	<a href="http://www.ti.com/broadband">www.ti.com/broadband</a>
Digital Control	<a href="http://www.ti.com/digitalcontrol">www.ti.com/digitalcontrol</a>
Medical	<a href="http://www.ti.com/medical">www.ti.com/medical</a>
Military	<a href="http://www.ti.com/military">www.ti.com/military</a>
Optical Networking	<a href="http://www.ti.com/opticalnetwork">www.ti.com/opticalnetwork</a>
Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Telephony	<a href="http://www.ti.com/telephony">www.ti.com/telephony</a>
Video & Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>
Wireless	<a href="http://www.ti.com/wireless">www.ti.com/wireless</a>

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2008, Texas Instruments Incorporated