

MSP430 Programming Via the JTAG Interface

User's Guide



Literature Number: SLAU320
July 2010

1	Programming Via the JTAG Interface	5
1.1	Introduction	5
1.2	Interface and Instructions	5
1.2.1	JTAG Interface Signals	6
1.2.2	JTAG Access Macros	8
1.2.3	Spy-Bi-Wire (SBW) Timing and Control	10
1.2.4	JTAG Communication Instructions	14
1.3	Memory Programming Control Sequences	20
1.3.1	Start-Up	20
1.3.2	General Device (CPU) Control Functions	23
1.3.3	Accessing Non-Flash Memory Locations With JTAG	32
1.3.4	Programming the Flash Memory (Using the Onboard Flash Controller)	36
1.3.5	Erasing the Flash Memory (Using the Onboard Flash Controller)	41
1.3.6	Reading From Flash Memory	44
1.3.7	Verifying the Flash Memory	45
1.4	JTAG Access Protection	45
1.4.1	Burning the JTAG Fuse - Function Reference for 1xx/2xx/4xx Families	45
1.4.2	Programming the JTAG Lock Key - Function Reference for 5xx Family	47
1.4.3	Testing for a Successfully Protected Device	48
1.5	JTAG Function Prototypes	48
1.5.1	Low-Level JTAG Functions	48
1.5.2	High-Level JTAG Routines	50
1.6	References	55
2	JTAG Programming Hardware and Software Implementation	57
2.1	Implementation History	57
2.2	Implementation Overview	57
2.3	Software Operation	58
2.4	Software Structure	59
2.5	Programmer Operation	61
2.6	Hardware Setup	61
2.6.1	Host Controller	61
2.6.2	Target Connection	61
2.6.3	Host Controller/Programmer Power Supply	63
2.6.4	Third Party Support	63
3	Internal MSP430 JTAG Implementation	65
3.1	TAP Controller State Machine	65
3.2	MSP430 JTAG Restrictions (Non-Compliance With IEEE Std 1149.1)	65
A	Errata and Revision Information	67
A.1	Known Issues	67
A.2	Revisions and Errata from Previous Documents	67

List of Figures

1-1.	Spy-Bi-Wire Basic Concept.....	7
1-2.	Timing Example for IR_SHIFT (0x83) Instruction.....	8
1-3.	Data Register I/O: DR_SHIFT16 (0x158B) (TDO Output Is 0x55AA).....	9
1-4.	Address Register I/O: DR_SHIFT20 (0x12568) (TDO Output Is 0xA55AA)	9
1-5.	SetTCLK	10
1-6.	ClrTCLK.....	10
1-7.	Timing Diagram (Alternative Timing).....	11
1-8.	SBW-to-JTAG Interface Diagram.....	11
1-9.	Detailed SBW Timing Diagram	12
1-10.	Synchronization of TDI/TCLK During Run-Test/Idle	13
1-11.	JTAG Access Entry Sequences (for Devices Supporting SBW)	21
1-12.	Fuse Check and TAP Controller Reset.....	22
1-13.	JTAG Entry Sequence for 430Xv2 Devices.....	29
1-14.	Accessing Flash Memory.....	39
1-15.	Flash Access Code Binary Image Map.....	40
1-16.	Fuse Blow Timing	47
2-1.	Replicator Application Schematic	62
3-1.	TAP Controller State Machine	65

List of Tables

1-1.	Standard 4-Wire JTAG Signals	6
1-2.	JTAG Signal Implementation Overview	7
1-3.	JTAG Communication Macros	8
1-4.	Memory Access Instructions	14
1-5.	JTAG Control Signal Register for 1xx/2xx/4xx Families.....	16
1-6.	JTAG Control Signal Register for 5xx Family.....	18
1-7.	Shared JTAG Device Pin Functions	20
1-8.	Erase/Program Minimum TCLK Clock Cycles	36
1-9.	Flash Memory Parameters ($f_{FTG} = 450$ kHz).....	43
1-10.	MSP430 Device JTAG Interface (Shared Pins)	45
1-11.	MSP430 Device Dedicated JTAG Interface.....	45
1-12.	JTAG Features Across Device Families	53
1-13.	MSP430x5xx, CC430 JTAG Features	54

Programming Via the JTAG Interface

This document describes the functions that are required to erase, program, and verify the memory module of the MSP430 flash-based microcontroller family using the JTAG communication port. In addition, it describes how to program the JTAG access security fuse that is 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, which includes software (source code is provided) and the corresponding hardware, is described in [Chapter 2](#). This example is intended as a reference to aid in understanding of the concepts presented in this report and to aid in development of similar MSP430 programmer solutions. In that sense, it is not meant to be a fully featured programming tool but, instead, it is intended as a construction manual for those. Those users who are looking for a ready-to-use tool should see Texas Instruments complete programming tool solution called [MSP430 In-System Gang Programmer](#).

1.1 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 1.2](#), 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 1.3](#), 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 1.4](#), 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.

[Chapter 2](#) 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 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 3-1](#). [Section 3.2](#) also lists various specialities of the MSP430 JTAG implementation which are non-compliant with IEEE Std 1149.1.

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.

All trademarks are the property of their respective owners.

1.2.1 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.

1.2.1.1 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 V_{CC} when powered by the programmer. These signals are described in [Table 1-1](#).

Table 1-1. 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 document, implement TCLK on the TDI input pin.

1.2.1.2 2-Wire Spy-Bi-Wire (SBW) JTAG Interface

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

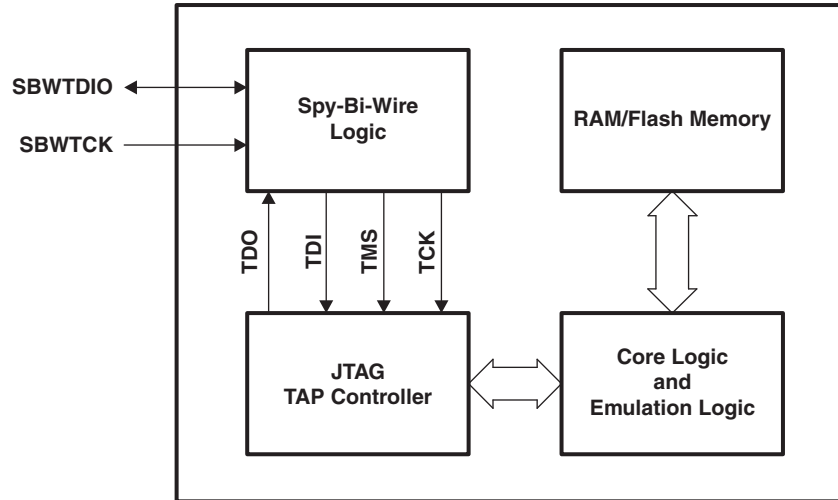


Figure 1-1. Spy-Bi-Wire Basic Concept

The 2-wire interface is made up of the SBWTCK (Spy-Bi-Wire test clock) and SBWTDIO (Spy-Bi-Wire test data input/output) pins. The SBWTCK signal is the clock signal and is a dedicated pin. In normal operation, this pin is internally pulled to ground. The SBWTDIO signal represents the data and is a bidirectional connection. To reduce the overhead of the 2-wire interface, the SBWTDIO line is shared with the RST/NMI pin of the MSP430.

[Table 1-2](#) gives a general overview of MSP430 devices and their respective JTAG interface implementation.

Table 1-2. 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
MSP430F5xx devices	YES	YES	YES

1.2.2 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. [Section 1.2.2.1](#) summarizes the macros used throughout this document and their associated functionality. See the accompanying software for more information.

Table 1-3. 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

1.2.2.1 Macros for 4-Wire JTAG Interface

1.2.2.1.1 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 1-2](#) shows how to load the ADDR_16BIT instruction into the JTAG IR register. See [Section 1.2.4](#) for a complete list of the JTAG interface communication instructions used to access the target device flash memory module.

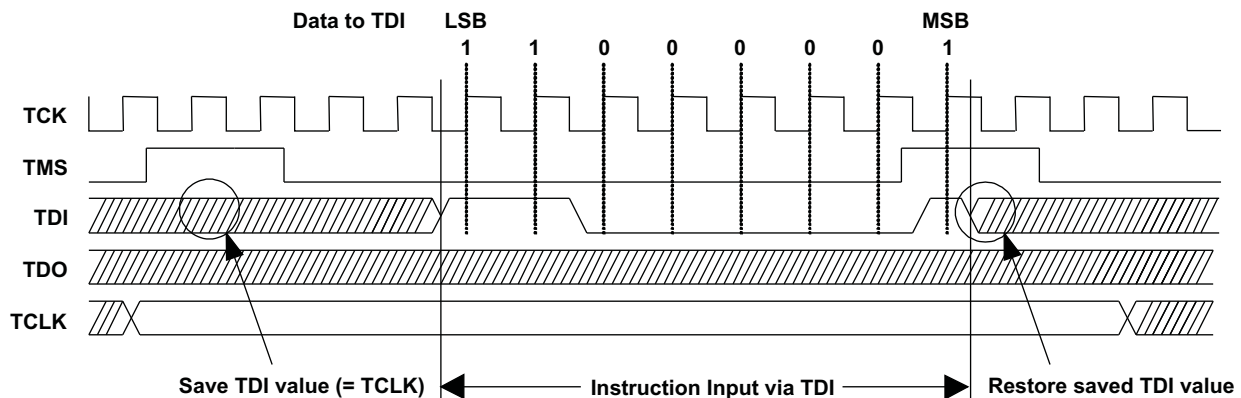


Figure 1-2. Timing Example for IR_SHIFT (0x83) Instruction

1.2.2.1.2 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 1-3](#) shows how to load a 16-bit word into the JTAG DR and read out a stored value via TDO.

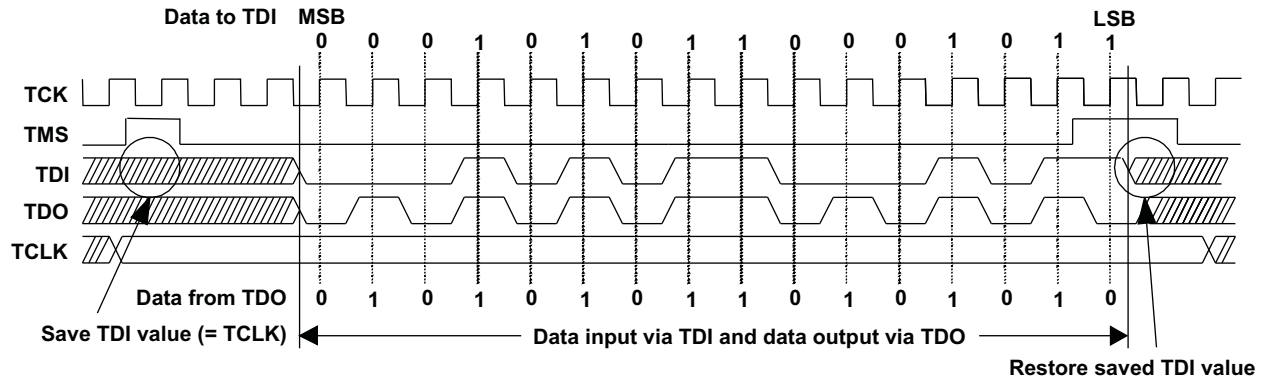


Figure 1-3. Data Register I/O: DR_SHIFT16 (0x158B) (TDO Output Is 0x55AA)

1.2.2.1.3 DR_SHIFT20 (20-bit Address) (Applies Only to MSP430X Devices)

The MSP430X architecture is based on a 20-bit memory address bus (MAB), to address up to 1 MB of continuous memory. No new JTAG instructions are needed to control the 20-bit MAB (for details on instructions, see [Section 1.2.4.1](#)), 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 1-4](#) shows how to load a 20-bit address word into the JTAG address register and read out a stored value via TDO.

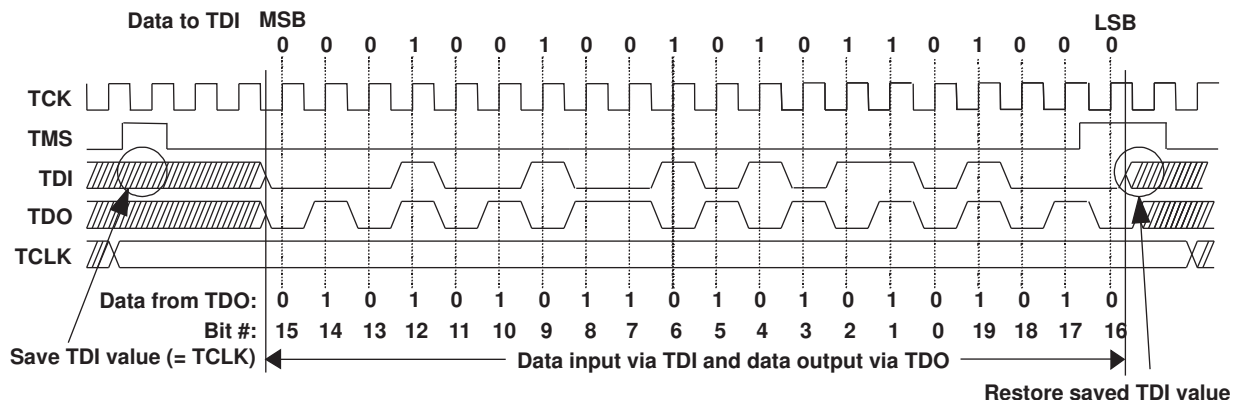


Figure 1-4. Address Register I/O: DR_SHIFT20 (0x12568) (TDO Output Is 0xA55AA)

1.2.2.1.4 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.

1.2.2.1.5 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 1.2.3.3](#) and [Figure 1-10](#) for SBW-specific constraints).

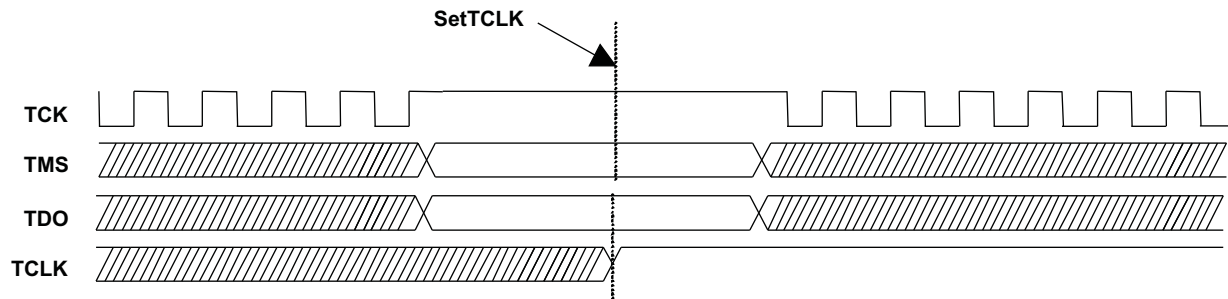


Figure 1-5. SetTCLK

1.2.2.1.6 ClrTCLK

This macro resets the TCLK input clock low. TCK and TMS must hold their last value while this action is performed (see [Section 1.2.3.3](#) and [Figure 1-10](#) for SBW-specific constraints).

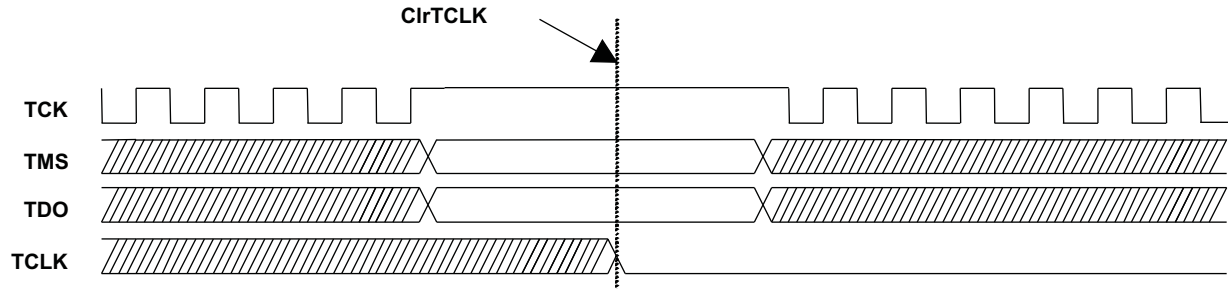


Figure 1-6. ClrTCLK

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

All JTAG macros described in [Section 1.2.2.1](#) also apply to the 2-wire interface and are provided as software source along with this document.

1.2.3 Spy-Bi-Wire (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.

1.2.3.1 Basic Timing

The SBW interface serial communication uses time-division multiplexing, allocating three time slots: TMS_SLOT, TDI_SLOT, and TDO_SLOT. 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 1-7](#).

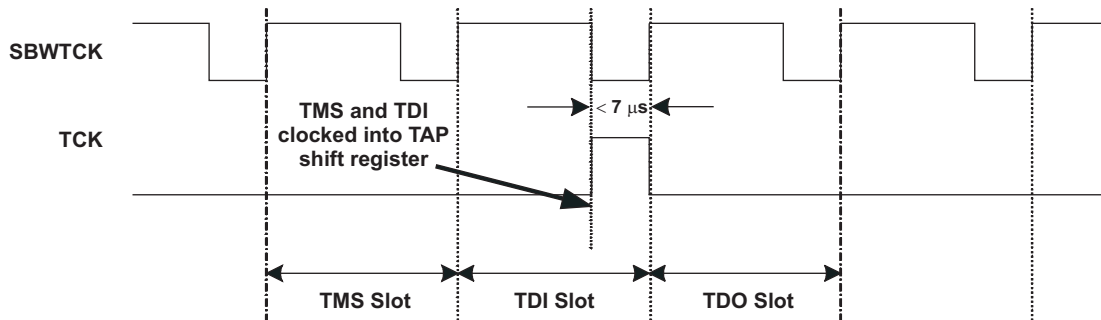


Figure 1-7. Timing Diagram (Alternative Timing)

The implemented logic used to translate between the 2-wire and 4-wire interfaces is shown in [Figure 1-8](#).

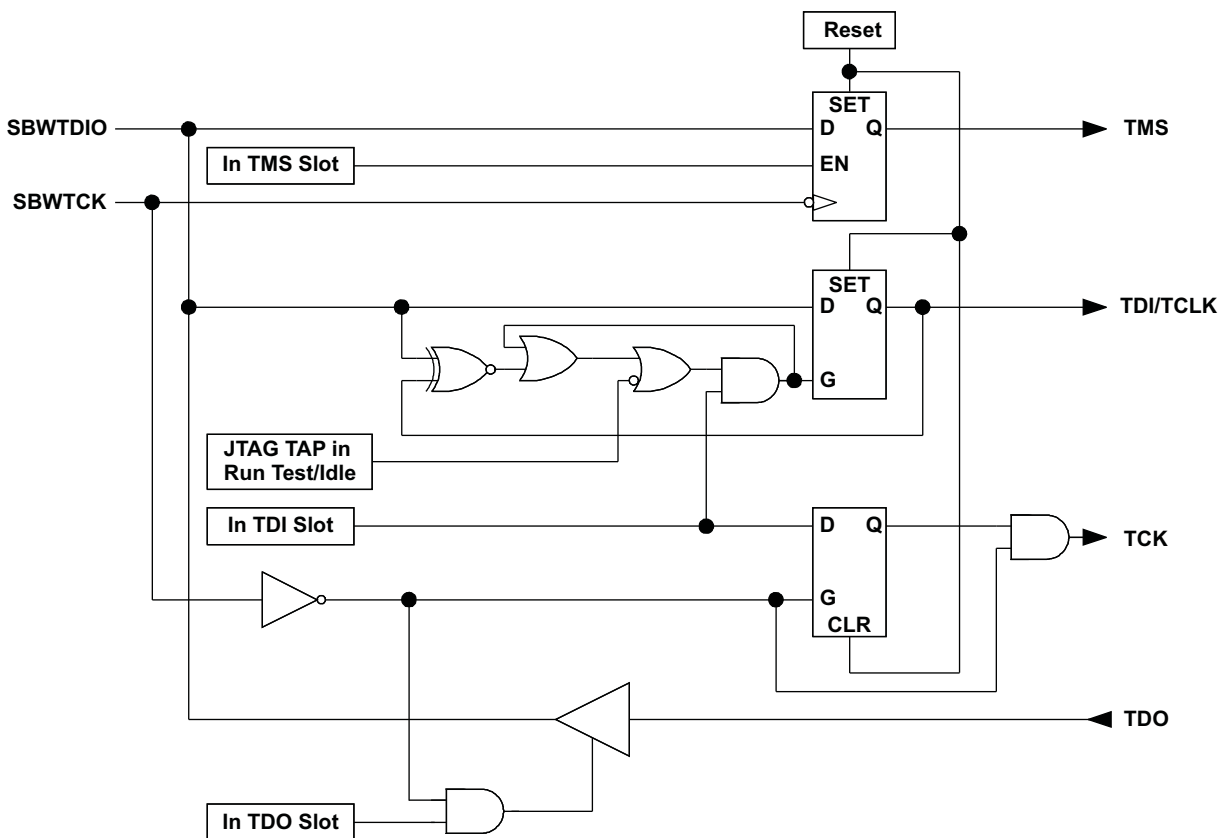


Figure 1-8. SBW-to-JTAG Interface Diagram

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 1-10](#). The synchronization logic is only active in the Run-Test/Idle state.

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

1.2.3.2 TDO Slot

As shown in Figure 1-7, the TDO operation is allocated one time slot (see also the detailed timing shown in Figure 1-9). 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 μ s, else SBW logic is deactivated and must be activated again according to Section 1.3.1.

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

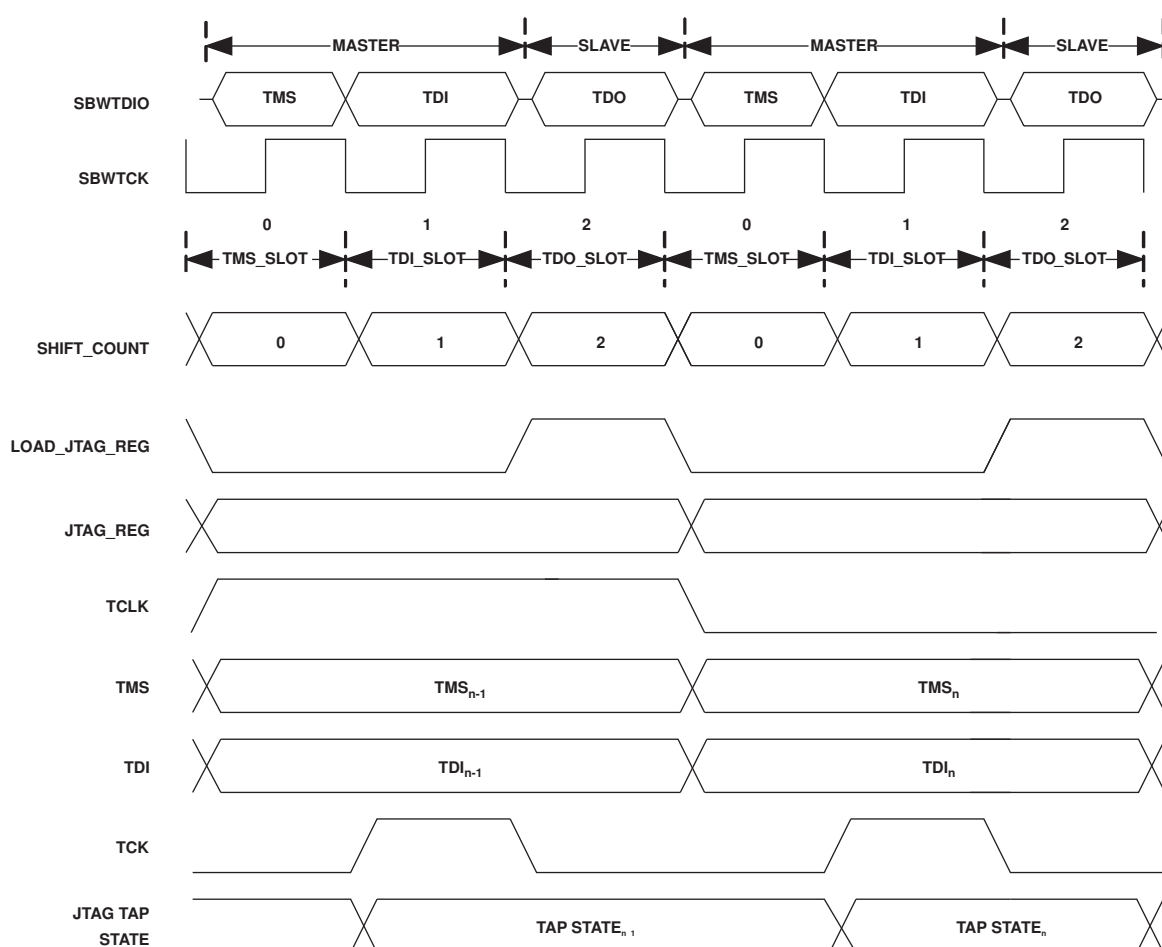


Figure 1-9. Detailed SBW Timing Diagram

1.2.3.3 SetTCLK and ClrTCLK in Spy-Bi-Wire (SBW) Mode

Figure 1-10 shows handling and synchronization of TCLK in SBW mode while the JTAG TAP controller is in Run-Test/Idle state. See reference function SetTCLK_sbww and ClrTCLK_sbww for software implementation. Note that the provided code example for the MSP430Xv2 architecture uses preprocessor definitions to enable a better layered software architecture. The upper software layers can simply reference the SetTCLK and ClrTCLK symbols while the actual implementation symbols are SetTCLK_4wire and ClrTCLK_4wire for 4-wire JTAG and SetTCLK_sbww and ClrTCLK_sbww for Spy-Bi-Wire (SBW).

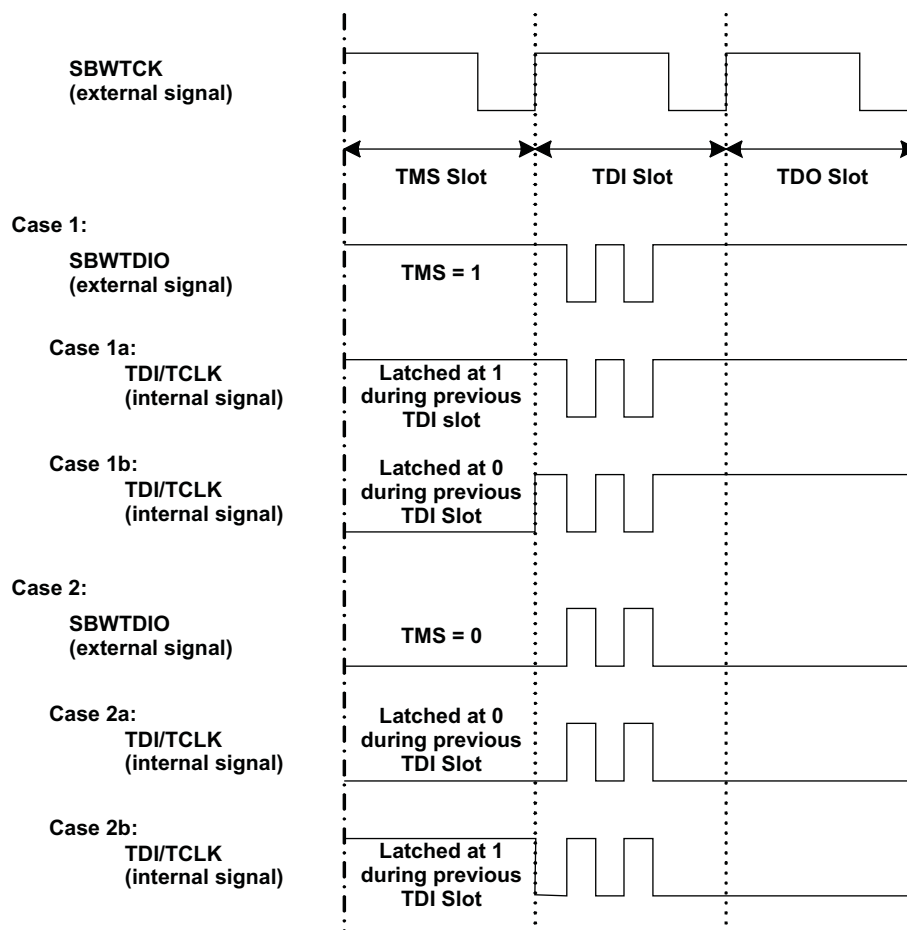


Figure 1-10. Synchronization of TDI/TCLK During Run-Test/Idle

1.2.4 JTAG Communication Instructions

Selecting a JTAG register and controlling the CPU is done by shifting in a JTAG instruction using the IR_SHIFT macro described in [Section 1.2.2.1.1](#). 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 1-4. Memory Access Instructions

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

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

NOTE: When a new JTAG instruction is shifted into the JTAG instruction register, it takes effect with the UPDATE-IR state of the TAP controller. When accessing a JTAG data register, the last value written is captured with the CAPTURE-DR state, and the new value shifted in becomes valid with the UPDATE-DR state. In other words, there is no need to go through Run-Test/Idle state of the JTAG TAP controller to shift in instructions or data. Be aware of the fact that clocking TCLK is only possible in the Run-Test/Idle state. This is why the provided software example application exclusively makes use of the JTAG macros described in [Section 1.2.2](#), which always go through Run-Test/Idle state.

1.2.4.1 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.

1.2.4.1.1 *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.

1.2.4.1.2 *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).

1.2.4.2 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.

1.2.4.2.1 *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.

1.2.4.2.2 *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.

1.2.4.2.3 IR_DATA_QUICK

This instruction enables setting of the MSP430 MDB to a specific value shifted in with the next 16-bit JTAG data access. The 16-bit MSP430 MDB is set to the value written to the JTAG MDB register. During the 16-bit data transfer, the previous MDB value is captured and shifted out on TDO. The MAB value is set by the program counter (PC) of the CPU. This instruction auto-increments the program counter by two on every falling edge of TCLK to automatically point to the next 16-bit memory location. The 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.

1.2.4.2.4 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.

1.2.4.3 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 1-5](#) describes the bit functions making up the JTAG control signal register used for memory access.

Table 1-5. JTAG Control Signal Register for 1xx/2xx/4xx Families

Bit No.	Name	Description
0	R/W	Controls the read/write (RW) signal of the CPU 1 = Read 0 = Write
1	(N/A)	Always write 0
2	(N/A)	Always write 0
3	HALT_JTAG	Sets the CPU into a controlled halt state 1 = CPU stopped 0 = CPU operating normally
4	BYTE	Controls the BYTE signal of the CPU used for memory access data length 1 = Byte (8-bit) access 0 = Word (16-bit) access
5	(N/A)	Always write 0
6	(N/A)	Always write 0
7	INSTR_LOAD	Read only: Indicates the target CPU instruction state 1 = Instruction fetch state 0 = Instruction execution state
8	(N/A)	Always write 0
9	TCE	Indicates CPU synchronization 1 = Synchronized 0 = Not synchronized
10	TCE1	Establishes JTAG control over the CPU 1 = CPU under JTAG control 0 = CPU free running

Table 1-5. JTAG Control Signal Register for 1xx/2xx/4xx Families (continued)

Bit No.	Name	Description
11	POR	Controls the power-on-reset (POR) signal 1 = Perform POR 0 = No reset
12	Release low byte	Selects control source of the RW and BYTE bits 1 = CPU has control 0 = Control signal register has control
13	TAGFUNCSAT	Sets flash module into JTAG access mode 1 = CPU has control (default) 0 = JTAG has control
14	SWITCH	Enables TDO output as TDI input 1 = JTAG has control 0 = Normal operation
15	(N/A)	Always write 0

Table 1-6. JTAG Control Signal Register for 5xx Family

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

1.2.4.3.1 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.

1.2.4.3.2 IR_CNTRL_SIG_CAPTURE

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

1.2.4.3.3 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.

1.2.4.4 Memory Verification Via Pseudo Signature Analysis (PSA)

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

1.2.4.4.1 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.

1.2.4.4.2 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.

1.2.4.5 JTAG Access Security Fuse Programming

The following instructions are used to access and program the built-in JTAG access protection fuse, available on every MSP430F1xx/2xx/4xx 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. Note that these instructions are not available for the MSP430F5xx family. A different software-based mechanism is used for these families to enable JTAG access protection. See [Section 1.4](#) for details.

1.2.4.5.1 IR_PREPARE_BLOW

This instruction sets the MSP430 into program-fuse mode.

1.2.4.5.2 IR_EX_BLOW

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

1.3 Memory Programming Control Sequences

1.3.1 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.

1.3.1.1 Enable JTAG Access

Reference function: GetDevice, GetDevice_sbww, GetDevice_430X, GetDevice_430Xv2

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

To use the JTAG features of MSP430 devices with shared JTAG and a TEST pin, it is necessary to enable the shared JTAG pins for JTAG communication mode. (Devices with dedicated JTAG inputs/outputs and no TEST pin do not require this step.) The shared pins are enabled for JTAG communication by connecting the TEST pin to V_{CC} . For normal operation (non-JTAG mode), this pin should be released and allowed to be internally pulled to ground. [Table 1-7](#) shows the port 1 pins that are used for JTAG communication.

Table 1-7. Shared JTAG Device Pin Functions

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

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

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

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

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

To select the 2-wire SBW mode, the SBWTDIO line is held high and the first clock is applied on SBWTCCK. 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/SWBCLK low for more than 100 μs .

In devices implementing the Bootstrap Loader (BSL), the TEST/SBWTCCK and $\overline{RST}/NMI/SBWTDIO$ are also used to invoke the BSL. In [Figure 1-11](#), different cases used to enter the SBW/JTAG or BSL mode are shown.

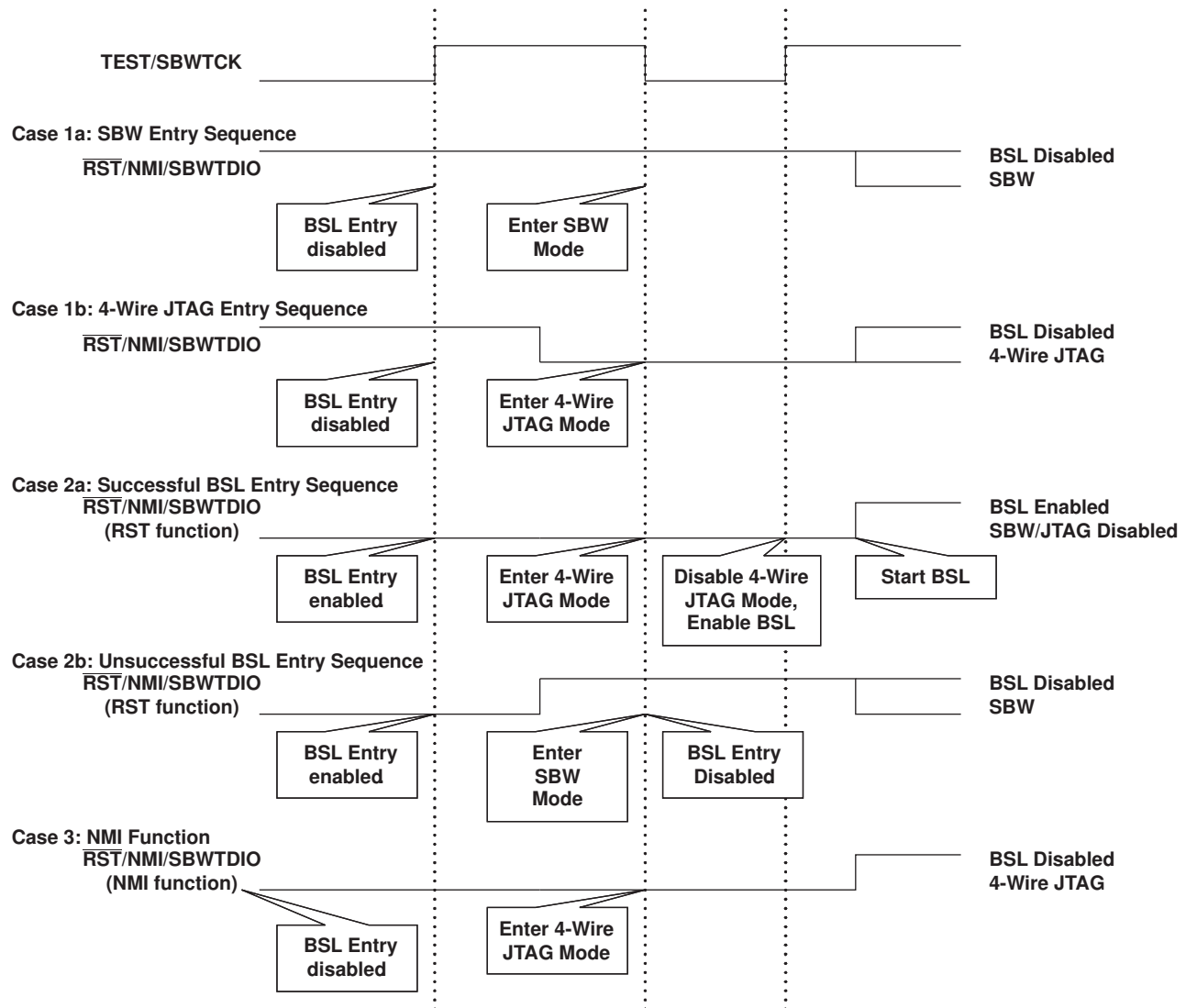


Figure 1-11. JTAG Access Entry Sequences (for Devices Supporting SBW)

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

- SBWTCK needs to be actively driven low before powering up the device or during the connector plug in action to avoid unintentional rising signal edges.
- Run the according initialization sequence multiple times (two to three repeats are typically sufficient to establish a stable connection).

1.3.1.2 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.

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

Under certain circumstances (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 is 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. Thus, it is recommended to first perform a reset of the TAP and then check the JTAG fuse status as shown in [Figure 1-12](#). To perform a reset of the TAP controller it is recommended that a minimum of six TCK clocks be sent to the target device while TMS is high followed by setting TMS low for at least one TCK clock. This sets the JTAG state machine (TAP controller) to a defined starting point: the Run-Test/Idle state. This procedure can also be used at any time during JTAG communication to reset the JTAG port.

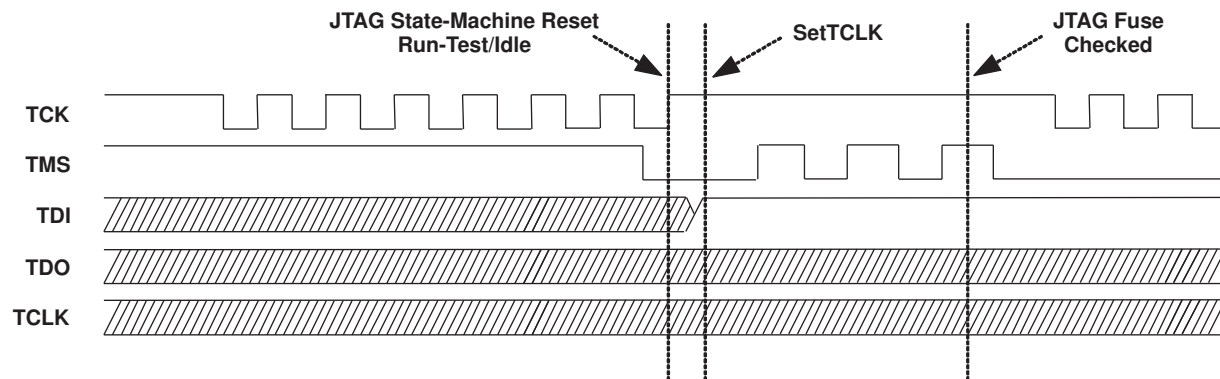


Figure 1-12. Fuse Check and TAP Controller Reset

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 1.2.3.1](#), the internal signal TCK is generated automatically in every TDI_SLOT. Performing a fuse check in SBW mode, starting directly after a reset of the TAP controller, ends in its Exit2-DR state. Two more dummy TCKs must be generated to return to Run-Test/Idle state; one TCK with SBWTDIO being high during the TMS_SLOT followed by one TCK with SBWTDIO being low during the TMS_SLOT (reference function: ResetTAP_sbww).

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

1.3.2 General Device (CPU) Control Functions

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

1.3.2.1 Function Reference for 1xx/2xx/4xx Families

1.3.2.1.1 Taking the CPU Under JTAG Control

Reference function: GetDevice, GetDevice_sbw, GetDevice_430X

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)	
Bit 9 of TDOWord = 1?	No
Yes	
CPU is under JTAG control	

1.3.2.1.2 Set CPU to Instruction-Fetch

Reference function: SetInstrFetch

Sometimes it is useful for the target device to directly execute an instruction presented by a host over the JTAG port. To accomplish this, the CPU must be set to the instruction-fetch state. With this setting, the target device CPU loads and executes an instruction as it would in normal operation, except that the instruction is transmitted 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	
Bit 7 of TDOvalue = 0?	
ClrTCLK	
SetTCLK	
CPU is in the instruction-fetch state	

1.3.2.1.3 Setting the Target CPU Program Counter (PC)

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

- MSP430 architecture: Reference function: SetPC

<i>CPU must be in the instruction-fetch state 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_430X

<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>

1.3.2.1.4 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>	
HaltCPU	IR_SHIFT("IR_DATA_16BIT")
	DR_SHIFT16(0x3FFF) : "JMP \$" instruction to keep CPU from changing the state
	ClrTCLK
	IR_SHIFT("IR_CNTRL_SIG_16BIT")
	DR_SHIFT16(0x2409) : set HALT_JTAG bit
	SetTCLK
<i>Now the CPU is in a controlled state and is not altered during memory accesses. Note: Do not reset the HALT_JTAG bit (= 0) while accessing the target memory.</i>	
Memory Access Performed Here	
<i>The CPU is switched back to normal operation using ReleaseCPU.</i>	
ReleaseCPU	ClrTCLK
	IR_SHIFT("IR_CNTRL_SIG_16BIT")
	DR_SHIFT16(0x2401) : Clear HALT_JTAG bit
	IR_SHIFT("IR_ADDR_CAPTURE")
	SetTCLK
<i>The CPU is now in the instruction-fetch state and ready to receive a new JTAG instruction. If the PC has been changed while the memory was being accessed, the PC must be loaded with the correct address.</i>	

1.3.2.1.5 Resetting the CPU While Under JTAG Control

Reference function: ExecutePOR

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

IR_SHIFT("IR_CNTRL_SIG_16BIT")
DR_SHIFT16(0x2C01) : Apply Reset
DR_SHIFT16(0x2401) : Remove Reset
ClrTCLK
SetTCLK
ClrTCLK
SetTCLK
ClrTCLK
IR_SHIFT("IR_ADDR_CAPTURE")
SetTCLK
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 0xFFFEh and all registers are set to their respective power-up values.
The target device's watchdog timer must now be disabled to avoid an undesired reset of the target.
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
The target CPU is now released for the next operation.
ClrTCLK
IR_SHIFT("IR_CNTRL_SIG_16BIT")
DR_SHIFT16(0x2401) : Set to Read
IR_SHIFT("IR_ADDR_CAPTURE")
SetTCLK

1.3.2.1.6 Release Device From JTAG Control

Reference function: ReleaseDevice

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

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

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

1.3.2.2 Function Reference for 5xx Family

1.3.2.2.1 Taking the CPU Under JTAG Control

Reference function: GetDevice_430Xv2

Also for the 5xx family, the CPU is taken under JTAG control by setting bit 10 (TCE1) of the JTAG control signal register to 1. While the flow to take the target device under JTAG control is identical to the flow described in [Section 1.3.2.1.1](#), additional actions must be taken to completely take over control of the target CPU; e.g., it is not recommended to take over control without performing a CPU reset by setting the POR signal in the JTAG Control Signal Register. Also, care must be taken that the CPU is in the Full-Emulation-State (equivalent to the Instruction-Fetch state for MSP430/MSP430X architectures) by setting the CPUSUSP signal and providing a number of TCLK until the CPU pre-fetch pipes are cleared. The following flow shows the full sequence required to get the CPU in the Full-Emulation-State.

IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x1501)	
IR_SHIFT("IR_CNTRL_SIG_CAPTURE")	
DR_SHIFT16(0x0000)	
Bit 9 of TDOWord = 1?	
Yes	
CPU is under JTAG control - always apply Power on Reset (POR) afterwards.	
ClrTCLK	
SetTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x0C01) : clear CPUSUSP signal and apply POR	
DR_SHIFT16(0x0401) : clear POR signal	
ClrTCLK	
SetTCLK	
DR_SHIFT16(0x0501) : set CPUSUSP signal again	
ClrTCLK	
SetTCLK	
CPU is now in Full-Emulation-State.	

Reference function: GetCoreID

[Figure 1-13](#) shows the JTAG-entry sequence for 430Xv2 devices. If no valid JTAG-ID is returned by the first entry sequence, a second one that uses a magic pattern is applied. The differences between the two entry sequences are that the second sequence hold the device in reset and feeds in a magic pattern (0xA55A) by using the JTAG-Mailbox. The magic pattern is read by the BootCode and the device is set in LPM4. User code execution is stopped in LPM4.

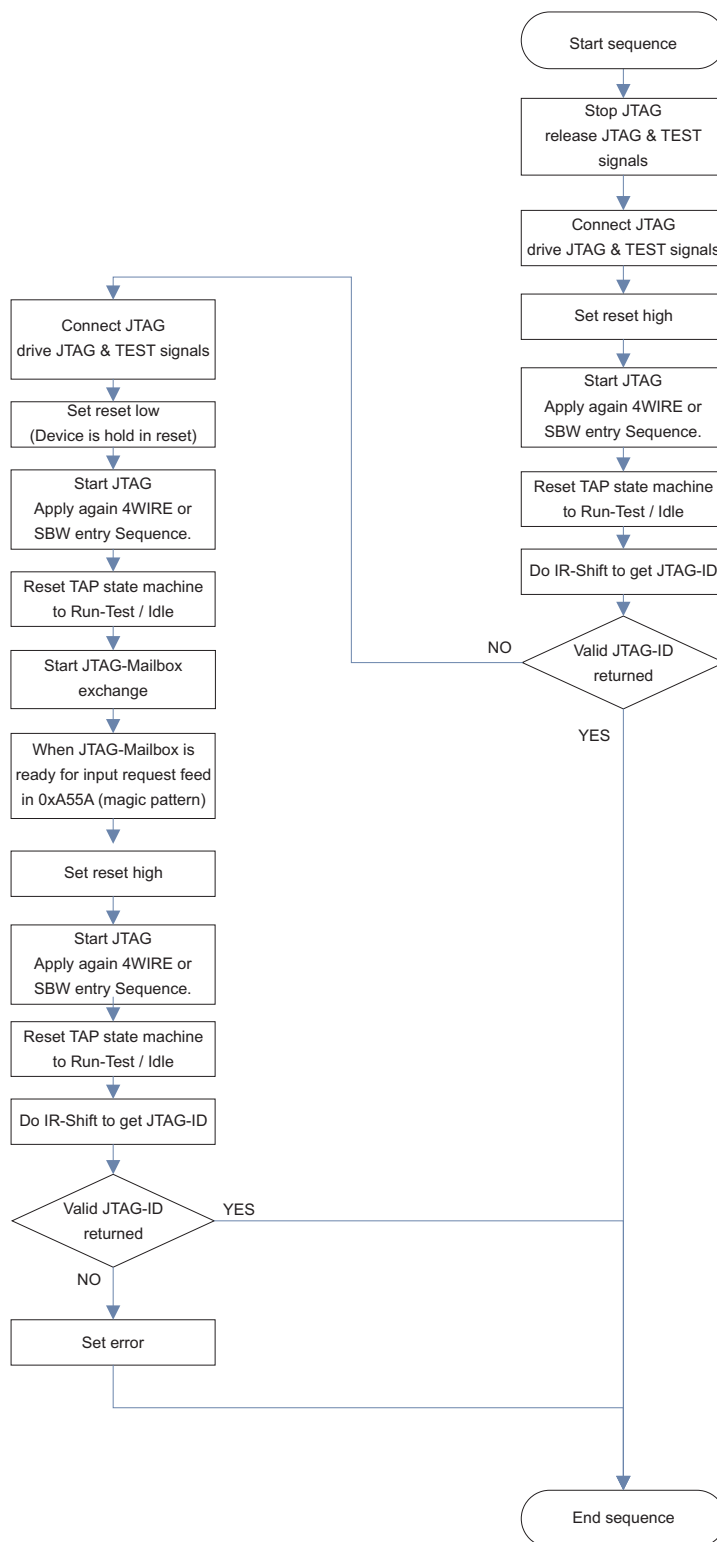


Figure 1-13. JTAG Entry Sequence for 430Xv2 Devices

1.3.2.2.2 Setting the Target CPU Program Counter (PC)

To use some of the features of the JTAG interface provided by the MSP430Xv2 architecture, setting of the CPU PC of the target device is required. The following flow is used to accomplish this. With the MSP430Xv2 architecture it is strongly recommended that after setting the PC no additional memory access is performed other than the described quick access methods under [Section 1.3.3.3](#) and [Section 1.3.7](#). After setting the PC the target device can be either released from JTAG control or continued to be clocked by providing TCLK to execute user program code which was previously stored at the memory location the PC is now pointing to. In any way, before the memory can be accessed again the CPU must be put again into the Full-Emulation-State like described in [Section 1.3.2.1](#).

- MSP430Xv2 architecture: Reference function: SetPC_430Xv2

<i>CPU must be in the Full-Emulation-State prior to the following sequence.</i>
ClrTCLK
IR_SHIFT("IR_DATA_16BIT")
SetTCLK
DR_SHIFT16("MOVA opcode incl. upper nibble of 20 bit PC value")
IR_SHIFT("IR_CNTRL_SIG_16BIT")
DR_SHIFT16(0x1400) : release low byte
IR_SHIFT("IR_DATA_16BIT")
ClrTCLK
SetTCLK
DR_SHIFT16("PC_Value") : Insert the lower 16 bit value for PC
ClrTCLK
SetTCLK
DR_SHIFT16(0x4303) : insert NOP instruction to be pre-fetched by CPU
ClrTCLK : Now PC is set
IR_SHIFT("IR_ADDR_CAPTURE")
DR_SHIFT20(0x00000)
<i>Load PC completed</i>

1.3.2.2.3 Release Device from JTAG control

Also for 5xx family, both ways described in [Section 1.3.2.1.6](#) can be applied. Note that a POR (power-on-reset) using the JTAG control signal register is not equivalent to a true power-up reset, which additionally issues a BOR (brownout reset) prior to the POR. Only a BOR causes the target devices' boot code to be executed, which performs various calibration and configuration tasks. Thus, the 5xx JTAG interface is enhanced with the capability to generate a BOR through the JTAG interface by accessing a dedicated JTAG data register. As soon as the appropriate BOR bit in the JTAG data register is set, the device is released from JTAG and performs a complete brownout reset startup sequence. See the `ReleaseDevice_Xv2` reference function for implementation details. Also see the [MSP430x5xx Family User's Guide](#) *System Resets, Interrupts and Operating Modes, System Control Module (SYS)* chapter for more information about various reset sources and device boot behavior.

Reference function: `ReleaseDevice_430Xv2`

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

1.3.3 Accessing Non-Flash Memory Locations With JTAG

1.3.3.1 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

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_SHIFT16("Address") : Set desired address	
IR_SHIFT("IR_DATA_TO_ADDR")	
SetTCLK	
ClrTCLK	
DR_SHIFT16(0x0000) : Memory value shifted out on TDO	
Read again?	
No	
ReleaseCPU should now be executed, returning the CPU to normal operation.	

- MSP430X architecture, Reference function: ReadMem_430X

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

- MSP430Xv2 architecture, Reference function: ReadMem_430Xv2

CPU must be in the Full-Emulation-State prior to the following sequence.		
ClrTCLK	Yes	
IR_SHIFT("IR_ADDR_16BIT")		
DR_SHIFT20("Address") : Set desired address		
IR_SHIFT("IR_DATA_TO_ADDR")		
SetTCLK		
ClrTCLK		
DR_SHIFT16(0x0000) : Memory value shifted out on TDO		
SetTCLK		
ClrTCLK		
SetTCLK		
Read again?	No	
CPU is now again in Full-Emulation-State.		

1.3.3.2 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_430X

<i>Set CPU to stopped state (HaltCPU)</i>	
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
No	
<i>ReleaseCPU should now be executed, returning the CPU to normal operation.</i>	

- MSP430Xv2 architecture, Reference function: WriteMem_430Xv2

CPU must be in the Full-Emulation-State prior to the following sequence.	
ClrTCLK	Yes
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x0500) : Write Memory	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT20("Address") : Set desired address	
SetTCLK	
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16("Data") : Send 16-bit Data	
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x0501)	
SetTCLK	
ClrTCLK	
SetTCLK	
Write again?	
No	
CPU is now again in Full-Emulation-State.	

1.3.3.3 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.

1.3.3.3.1 Flow for Quick Read (All Memory Locations)

- Both MSP430 and MSP430X architecture, Reference function: ReadMemQuick

<i>Set PC to start address – 4 (SetPC resp. SetPC_430X)</i>	
<i>Switch CPU to stopped state (HaltCPU)</i>	
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	

DR_SHIFT16(0x2409)	: Set RW to read	Yes
IR_SHIFT("IR_DATA_QUICK")		
SetTCLK		
DR_SHIFT16(0x0000)	: Memory value shifted out on TDO	
ClrTCLK	: Auto-increments PC	
Read From Next Address?		No
ReleaseCPU should now be executed, returning the CPU to normal operation. Reset the target CPU's PC if needed (SetPC).		

- MSP430Xv2 architecture, Reference function: ReadMem_430Xv2

<i>CPU must be in the Full-Emulation-State prior to the following sequence.</i>	
<i>Set PC to start address (SetPC_Xv2)</i>	
SetTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x0501) : Set RW to read	
IR_SHIFT("IR_ADDR_CAPTURE")	
IR_SHIFT("IR_DATA_QUICK")	
SetTCLK	Yes
ClrTCLK : Auto-increments PC	
DR_SHIFT16(0x0000) : Memory value shifted out on TDO	
Read From Next Address?	
No	No
<i>Get CPU in Full-Emulation-State.</i>	

NOTE: For the MSP430F5xx family quick memory access must be used with care as the PC already points to one address ahead of the actual address to be read. This could easily lead to security access violations especially at the end of a physical memory block.

1.3.3.3.2 Flow for Quick Write (RAM and Peripheral Memory Only)

- Both MSP430 and MSP430X architecture, Reference function: WriteMemQuick

Set PC to start address – 4 (SetPC)		
Switch CPU to stopped state (HaltCPU)		
ClrTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		
DR_SHIFT16(0x2408)	: 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).		

NOTE: Quick memory write access is not supported for the MSP430F5xx family.

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

1.3.4.1 Function Reference for 1xx/2xx/4xx Families

Reference function: WriteFLASH

This section describes one method available to program the flash memory module in an MSP430 device. It uses the same procedure that user-defined application software would 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, see the corresponding MSP430 user's guide and specific device data sheet. [Table 1-8](#) shows the required minimum number of TCLK cycles, depending on the action performed on the flash (for FCTL2 register bits 0 to 7 = 0x40 as defined in the MSP430 user's guide).

Table 1-8. Erase/Program Minimum TCLK Clock Cycles

Flash Action	Minimum TCLK Count
Segment erase	4820
Mass erase	5300 to 10600 ⁽¹⁾
Program word	35

⁽¹⁾ MSP430 device dependent, see device-specific data sheet. See [Section 1.3.5](#) for more details.

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

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

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

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

⁽³⁾ Correct timing required. Must meet min/max TCLK frequency requirement of 350 kHz ± 100 kHz.

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

1.3.4.2 Function Reference for 5xx Family

Having its own dedicated Timing Generator available on chip, Flash access with the 5xx family becomes significantly easier compared to the other MSP430 families. One need not take care about providing a certain erase/program frequency via the TCLK signal. All timings, required for both Flash memory erase- and write-access, are generated automatically. Basis for the following is that the Flash memory access operation can be initiated from within RAM, like described in the relevant *MSP430x5xx Family User's Guide* chapters. This report describes how to load an appropriate code in the target devices' RAM and control the correct execution of the code using the JTAG interface. Controlling the execution of the target code can be done in two ways; either the target is kept under JTAG control or released from JTAG control. Keeping the device under JTAG control requires a certain amount of TCLK cycles being provided to the device to clock the CPU through the program code. Releasing the device from JTAG control makes the CPU execute the program code in free running mode. After the desired operation is finished the device must be taken under JTAG control again. Both methods have their pros and cons. While having a free-running device can increase Flash programming speed to its upper limit, it requires a polling mechanism via JTAG to retrieve the current target device state. On the other hand such a polling mechanism is not suitable for systems where more than one target device is to be accessed in parallel. As all targets would not run at exactly the same frequency, keeping them under JTAG control would be the recommended approach for those parallel access setups.

Exchanging information between the target devices' CPU and JTAG (e.g. for device state polling purposes) is realized by using a new feature of the 5xx JTAG implementation: The JTAG mailbox system. The idea behind the JTAG mailbox system is to have a direct interface to the CPU during debugging, programming and test that is identical for all '430 devices of this family and uses only few or no user application resources (reference to [MSP430x5xx Family User's Guide](#) *System Resets, Interrupts and Operating Modes, System Control Module (SYS)* chapter).

[Figure 1-14](#) shows the general flow required to perform Flash memory operations on 5xx devices through the JTAG interface. The term Flash-Access-Code stands for an appropriate executable MSP430 code that can be used to perform the according Flash access operation. The following sections refer to the term Flash-Write-Code for code that is used to program the flash memory and Flash-Erase-Code for code that is used to erase the flash memory.

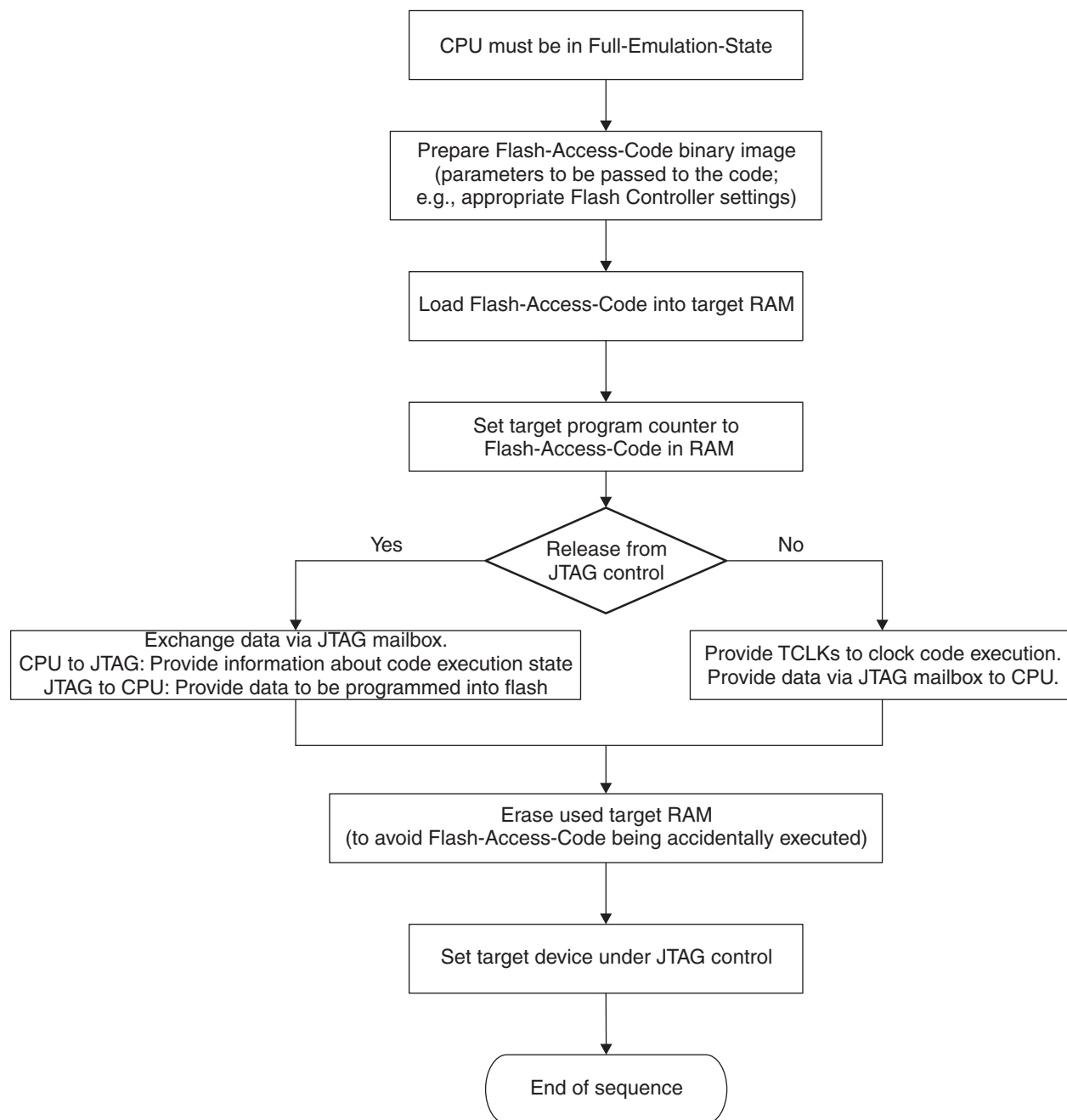


Figure 1-14. Accessing Flash Memory

Reference function: WriteFLASH_430Xv2 and WriteFLASH_430Xv2_wo_release

This section describes one method to program the flash memory subsequently with 16-bit word data by executing an appropriate Flash-Write-Code in RAM and providing the data to the CPU via the JTAG mailbox system. The provided source code example includes a Flash-Write-Code example that has the capability to be parameterized in binary state. [Figure 1-15](#) shows a generic map of the binary image of the Flash-Access-Code(s) provided with this document.

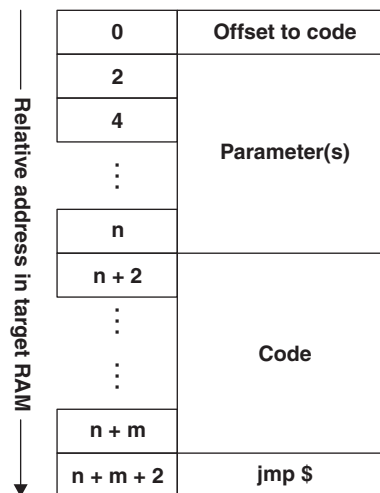


Figure 1-15. Flash Access Code Binary Image Map

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

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

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

When executing the Flash-Write-Code (either under JTAG control or free-running) the data to be programmed into the targets' flash memory needs to be provided through the JTAG mailbox system. The following sequences show how this is established in both JTAG-control- and free-running-mode.

- Under JTAG control

Target device is under JTAG control		
IR_SHIFT("IR_JB_EXCHANGE")		
DR_SHIFT16(0x0001)	: Send input request to JTAG mailbox	Yes
DR_SHIFT16("Data")	: Shift 16 bit word into JTAG mailbox	
Provide TCLK cycles for Flash-Write-Code processing (at least 30 cycles in a minimum time of t_{Word} MAX (see the device-specific data sheet))		
Write Another Flash Address?		
No		
Get target device in Full-Emulation-State		

- Released from JTAG control

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

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

1.3.5.1 Function Reference for 1xx/2xx/4xx Families

Reference function: EraseFLASH

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

1.3.5.1.1 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) ⁽¹⁾	: Point to FCTL1 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA502)	: Enable FLASH segment erase
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x012A) ⁽¹⁾	: Point to FCTL2 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA540)	: Source is MCLK, divider by 1
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x012C) ⁽¹⁾	: Point to FCTL3 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500) ⁽²⁾	: Clear FCTL3 Register
SetTCLK	
ClrTCLK	

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

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

IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16("EraseAddr") ⁽¹⁾	: Set Address for Erase ⁽³⁾
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 ⁽⁴⁾
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2408)	: Set RW to Write
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x0128) ⁽⁵⁾	: 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) ⁽⁵⁾	: Point to FCTL3 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500) ⁽⁶⁾	: Disable FLASH Write Access
SetTCLK	
ReleaseCPU should now be executed, returning the CPU to normal operation.	

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

⁽⁴⁾ Correct timing required. Must meet min/max TCLK frequency requirement of 350 kHz ±100 kHz.

⁽⁵⁾ Replace with DR_SHIFT20("Address") when programming an MSP430X architecture device.

⁽⁶⁾ Substitute 0xA540 for '2xx devices for Info-Segment A programming.

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

Beside the TCLK signal at a frequency of $350 \text{ kHz} \pm 100 \text{ kHz}$ (used for the Flash Timing Generator, data sheet parameter f_{FTG}), two more data sheet parameters must be taken into account when using the described method to perform a mass or main memory erase. The first is t_{CMERase} (cumulative mass erase time) and the second is $t_{\text{Mass Erase}}$ (mass erase time). Two different specification combinations of these parameters are currently implemented in dedicated MSP430 devices. Table 1-9 shows an overview of the parameters (assuming a maximum TCLK frequency of 450 KHz).

Table 1-9. Flash Memory Parameters ($f_{\text{FTG}} = 450 \text{ kHz}$)

Implementation	t_{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, 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) ⁽²⁾	: 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) ⁽²⁾	: 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) ⁽²⁾	: FCTL3 address
IR_SHIFT("IR_DATA_TO_ADDR")	

Perform once or
Repeat 19 times⁽¹⁾

⁽¹⁾ Correct timing required. Must meet min/max TCLK frequency requirement of $350 \text{ kHz} \pm 100 \text{ kHz}$.

⁽²⁾ Replace with DR_SHIFT20("Address") when programming an MSP430X architecture device.

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

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

⁽⁴⁾ 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.

1.3.5.2 Function Reference for 5xx Family

Reference function: EraseFLASH_430Xv2 and EraseFLASH_430Xv2_wo_release

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

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

The Flash-Erase-Code can be executed either under JTAG control or in free-running mode. Similar to what is described in [Section 1.3.4.2](#) the JTAG mailbox system is used to retrieve the current execution state of the Flash-Erase-Code in the target device.

1.3.6 Reading From Flash Memory

Reference function: ReadMem or ReadMemQuick

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

1.3.7 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 ≈ 23 ms/4 kB.

1.4 JTAG Access Protection

1.4.1 Burning the JTAG Fuse - Function Reference for 1xx/2xx/4xx Families

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

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

Devices with a TEST pin:

Table 1-10. 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 1-11. 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 $6.5\text{ V} \pm 0.5\text{ V}$.

1.4.1.1 Standard 4-Wire JTAG

Reference function: BlowFuse

1.4.1.1.1 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 1-11 describes the dual functionality for the TDI and TDO pins.) The settling time of the V_{pp} source must be taken into account when generating the proper timing to blow the fuse. The following flow details the fuse-programming sequence built into the BlowFuse function.

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

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

1.4.1.2 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 1-16. 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.

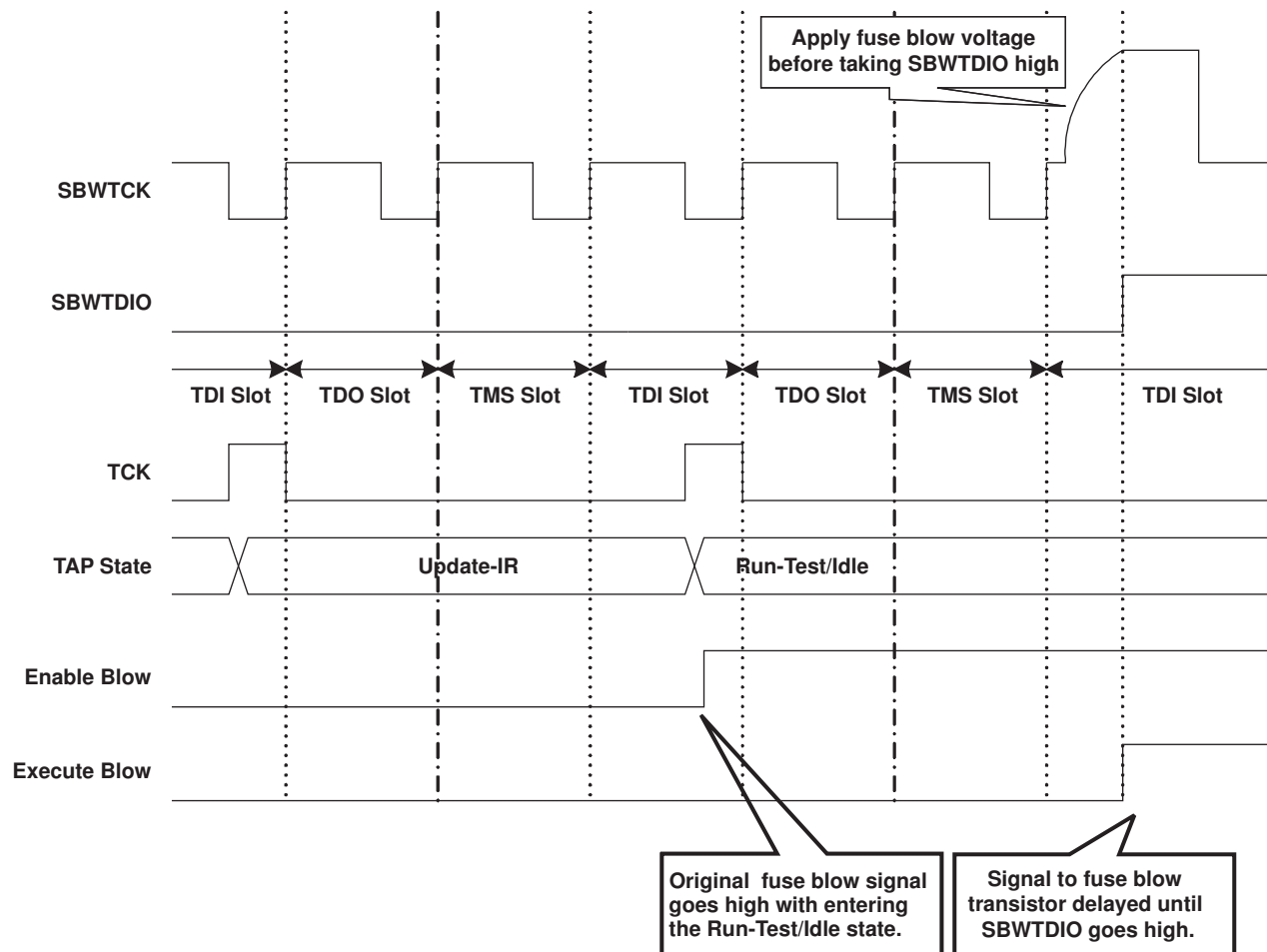


Figure 1-16. Fuse Blow Timing

1.4.2 Programming the JTAG Lock Key - Function Reference for 5xx Family

Reference function: ProgramLockKey

NOTE: For the MSP430F5xx family it is NOT required to apply a special high voltage to the device's TEST pin.

Other than for the 1xx/2xx/4xx families, where special handling was required to burn the JTAG security fuse, with the 5xx family the JTAG is locked by programming a certain signature into the devices' flash memory at dedicated addresses. The JTAG security lock key resides at the end of the bootstrap loader (BSL) memory at addresses 0x17FC to 0x17FF. Anything other than 0 or 0xFFFFFFFF programmed to these addresses locks the JTAG interface irreversibly. All of the 5xx MSP430 devices come with a preprogrammed BSL (TI-BSL) code which by default protects itself from unintended erase and write access. This is done by setting the SYSBSLPE bit in the according SYSBSLC register of the SYS module (see [MSP430F5xx Family User's Guide](#) SYS Module chapter for details). As the JTAG security lock key resides in the BSL memory address range, appropriate action must be taken to unprotect the memory area before programming the protection key. This can be done by a regular memory write access as described in [Section 1.3.3.2](#) by writing directly to the SYSBSLC register address and setting the SYSBSLPE to 0. Afterwards the BSL memory behaves like regular flash memory and a JTAG lock key can be programmed at addresses 0x17FC to 0x17FF like described in [Section 1.3.4.2](#). Note that a

Brownout Reset (BOR) is required to activate the JTAG security protection during boot. The BOR can be issued like described in [Section 1.3.2.2.3](#). If the hardware setup does not allow performing a power cycle (e.g., the battery is already soldered to the PCB) a BOR can also be generated by JTAG by writing into a dedicated JTAG test data register. Note that a BOR also resets the JTAG interface, which causes the device to be released from JTAG control.

1.4.3 Testing for a Successfully Protected Device

Reference function: IsFuseBlown, IsLockKeyProgrammed

Once the JTAG Fuse is burned (for 1xx/2xx/4xx devices) or the JTAG Lock Key is programmed (for 5xx devices) 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 being protected, 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: Device IS protected	No: Device NOT protected

1.5 JTAG Function Prototypes

1.5.1 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 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)

1.5.2 High-Level JTAG Routines

word GetDevice (void)

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

Arguments: None

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

void ReleaseDevice (word Addr)

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

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

Result: None

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

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

Arguments: word Format (F_BYTE or F_WORD)
word Addr (destination address for data to be written)
word Data (data value to be written)

Result: None

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

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

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

Result: None

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

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

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

Result: None

word WriteFLASHallSections (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)

Table 1-12. JTAG Features Across Device Families⁽¹⁾

Device	Device ID @ 0x0FF0	Device ID @ 0x0FF1	Device ID @ 0x0FFD	Test Pin	CPUX	DataQuick ⁽²⁾	FastFlash ⁽³⁾	EhhVerify ⁽⁴⁾	JTAG	Spy-Bi-Wire
F11x(1)(A)	0xF1	0x12	-	True	False	True	False	False	True	False
F11x2	0x11	0x32	-	True	False	True	False	False	True	False
F12x(A)	0xF1	0x23	-	True	False	False	False	False	True	False
F12x2	0x12	0x32	-	True	False	True	False	False	True	False
F13x, F14x	0xF1	0x49	-	False	False	True	False	False	True	False
F15x, F16x	0xF1	0x69	-	False	False	True	False	False	True	False
F161x	0xF1	0x6C	-	False	False	True	False	False	True	False
F20x3	0xF2	0x01	0x03	True	False	True	True	False	True	True
F20x2	0xF2	0x01	0x02	True	False	True	True	False	True	True
F20x1	0xF2	0x01	0x01	True	False	True	True	False	True	True
F21x1	0xF2	0x13	0x01	True	False	True	False	True	True	False
F21x2	0xF2	0x13	0x02	True	False	True	True	True	True	True
F22x2, F22x4	0xF2	0x27	-	True	False	True	True	True	True	True
F23x0	0xF2	0x37	-	True	False	True	True	True	True	False
F23x, F24x, F24x1, F2410	0xF2	0x49	-	False	False	True	True	True	True	False
F241x, F261x	0xF2	0x6F	-	False	True	True	True	True	True	False
F41x	0xF4	0x13	-	False	False	False	False	False	True	False
F41x2	0x41	0x52	-	True	False	True	True	True	True	True
F(E)42x	0xF4	0x27	'E'	False	False	True	False	False	True	False
FW42x	0xF4	0x27	'W'	False	False	True	False	False	True	False
F(G)42x0	0xF4	0x27	'G'	False	False	True	False	False	True	False
F43x (80 pin)	0xF4	0x37	-	False	False	True	False	False	True	False
FG43x	0xF4	0x39	-	False	False	True	False	False	True	False
F44x, F43x (100 pin)	0xF4	0x49	0x00	False	False	True	False	False	True	False
F47xx	0xF4	0x49	0x02	False	False	True	True	True	True	False
FG461x, F461x1, F461x	0xF4	0x6F	'G'	False	True	True	True	True	True	False
FE42x2	0x42	0x52	'E'	False	False	True	False	False	True	False
F(G)47x	0xF4	0x79	'G'	False	False	True	True	True	True	False
F(E)42xA	0x42	0x7A	'E'	False	False	True	True	False	True	False
F471xx	0xF4	0x7F	-	False	True	True	True	True	True	False
G2x2x	0xF2	0x01	0x02	True	False	True	True	False	True	True
G2x3x	0xF2	0x01	0x02	True	False	True	True	False	True	True
G2x0x	0xF2	0x01	0x01	True	False	True	True	False	True	True
G2x1x	0xF2	0x01	0x01	True	False	True	True	False	True	True

⁽¹⁾ All devices in the MSP430F1xx, MSP430F2xx, and MSP430F4xx families have JTAG ID 0x89.

⁽²⁾ DataQuick: If True, device supports read/write memory locations in quick mode using the JTAG DATA_QUICK instruction (see [Section 1.3.3.3](#)).

⁽³⁾ FastFlash: If True, device has a cumulative erase time ($t_{CMErase}$) of 20 ms; if False, $t_{CMErase}$ is 200 ms (see [Section 1.3.5.1.2](#)).

⁽⁴⁾ EhhVerify: If True, device supports a more advanced memory content verification mechanism (PSA checksum calculation). If False, the CPU of the device still works in the background while the PSA checksum algorithm is executed. This fact requires a POR being performed after checksum calculation. With the enhanced PSA hardware implementation, the CPU is completely halted during checksum calculation. A POR is not required afterwards.

Table 1-13. MSP430x5xx, CC430 JTAG Features^{(1) (2) (3)}

Device	Device ID @ 0x1A04	Device ID @ 0x1A05	Device ID @ 0x1A06
XMS430F5438	0x54	0x38	0x01
MSP430F5438	0x54	0x38	>0x01
MSP430F5437	0x54	0x37	>0x01
MSP430F5436	0x54	0x36	>0x01
MSP430F5435	0x54	0x35	>0x01
MSP430F5419	0x54	0x19	>0x01
MSP430F5418	0x54	0x18	>0x01
MSP430F5438A	0x05	0x80	—
MSP430F5437A	0x04	0x80	—
MSP430F5436A	0x03	0x80	—
MSP430F5435A	0x02	0x80	—
MSP430F5419A	0x01	0x80	—
MSP430F5418A	0x00	0x80	—
CC430F6137	0x61	0x37	—
CC430F6135	0x61	0x35	—
CC430F6126	0x61	0x26	—
CC430F6125	0x61	0x25	—
CC430F5137	0x51	0x37	—
CC430F5135	0x51	0x35	—
CC430F5133	0x51	0x33	—
MSP430F5513	0x55	0x13	—
MSP430F5514	0x55	0x14	—
MSP430F5515	0x55	0x15	—
MSP430F5517	0x55	0x17	—
MSP430F5519	0x55	0x19	—
MSP430F5521	0x55	0x21	—
MSP430F5522	0x55	0x22	—
MSP430F5524	0x55	0x24	—
MSP430F5525	0x55	0x25	—
MSP430F5526	0x55	0x26	—
MSP430F5527	0x55	0x27	—
MSP430F5528	0x55	0x28	—
MSP430F5529	0x55	0x29	—
MSP430F5510	0x80	0x31	
MSP430F5131	0x80	0x26	
MSP430F5132	0x80	0x28	
MSP430F5151	0x80	0x2A	
MSP430F5152	0x80	0x2C	
MSP430F5171	0x80	0x2E	
MSP430F5172	0x80	0x30	
MSP430F5630	0x80	0x3C	
MSP430F5631	0x80	0x3E	
MSP430F5632	0x80	0x40	

⁽¹⁾ All devices in this table have JTAG ID 0x91.

⁽²⁾ All devices with JTAG ID 0x91 contain an MSP430Xv2 architecture and s0x80 support both 4-wire JTAG and Spy-Bi-Wire.

⁽³⁾ See Section 1.11 *Device Descriptor Table* in the *MSP430x5xx Family User's Guide* ([SLAU208](#)) for more details on identification information.

Table 1-13. MSP430x5xx, CC430 JTAG Features^{(1) (2) (3)}
(continued)

Device	Device ID @ 0x1A04	Device ID @ 0x1A05	Device ID @ 0x1A06
MSP430F5633	0x80	0x42	
MSP430F5634	0x80	0x44	
MSP430F5635	0x80	0x0E	
MSP430F5636	0x80	0x10	
MSP430F5637	0x80	0x12	
MSP430F5638	0x80	0x14	
MSP430F6630	0x80	0x46	
MSP430F6631	0x80	0x48	
MSP430F6632	0x80	0x4A	
MSP430F6633	0x80	0x4C	
MSP430F6634	0x80	0x4E	
MSP430F6635	0x80	0x16	
MSP430F6636	0x80	0x18	
MSP430F6637	0x80	0x1A	
MSP430F6638	0x80	0x1C	

1.6 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

JTAG Programming Hardware and Software Implementation

2.1 Implementation History

There are three Replicator implementation. The latest version is discussed in this document, while the previous versions are found in the associated source code ZIP file as reference for legacy users. The main difference between the first 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. The implementation described in this document is the preferred implementation, the previous two implementations are no longer maintained.

2.2 Implementation Overview

The following sections document the examples provided with this document (<http://www.ti.com/lit/zip/slau265>). Each example demonstrates the software functions described in the previous sections using an MSP430F5437 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 user's guide. 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 Spy-Bi-Wire devices when programmed in 4-wire mode)
 - Replicator for Spy-Bi-Wire: 2-wire interface implementation for Spy-Bi-Wire devices only
 - Replicator for MSP430X: For 4-wire MSP430X extended architecture devices only
 - Replicator for MSP430Xv2: For MSP430Xv2 (5xx Family) extended architecture devices only (includes both 4-wire JTAG and Spy-Bi-Wire implementation)

NOTE: The Replicator source files are provided in independent folders with the same names as previously given. Within these folders, 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 250 KB (due to the limited memory resources of the MSP430F5437 host controller of 256 KB)
- 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)

2.3 Software Operation

The host controller stores the JTAG communication protocol code and the target program in its flash memory (256 KB available on the MSP430F5437). 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 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 to erase, program, and verify the target device. While the system is active, yellow LED on the programmer board is 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 program the host MSP430F5437 different development environments can be used—IAR Embedded Workbench™ or CCS™ by Texas Instruments. The free versions of IAR and CCE impose code size restrictions. To use the 250 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.

2.4 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 JTAGfunc430Xv2.c	Contain the MSP430-specific functions needed for flash programming
	JTAGfunc.h JTAGfuncSBW.h JTAGfunc430X.h JTAGfunc430Xv2.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 MSP430F5437 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 that the input file should be converted in. For this example only, C-header and assembly formats are allowed. Choose one format for your purpose.
- `-output_word` : The parameter is necessary, because the source code expects words to write to the target device. Otherwise, `srec_cat.exe` would write bytes.
- `-c_compressed` : This statement is additional to the `c_array` output. If specified, the output does not fill any address gap with a `0xFF` pattern, and does not increase the file size.
- The following statements are additional to the assembly output. Choose one to specify your format.
 - `-a430` : Writes an assembly file that is understood by the IAR Embedded Workbench in the Replicator context.
 - `-cl430` : Writes an assembly file that is understood by TI 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/>. This software was tested to function correctly with version 1.36, but is not necessarily compatible with future versions.

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

2.5 Programmer Operation

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

2.6 Hardware Setup

The hardware consists of the host controller MSP430F5437, programmable voltage regulator that can supply target device with V_{cc} 2.1V to 3.6V with step 0.1V, two JTAG interface connectors, and one BSL interface connector. An external power supply delivering 8 V to 10 V dc at 200 mA is required for operation (see Figure 9-1) if blow the security fuse option is required, or 4 V to 10 V dc at 200 mA can be used otherwise. The REP430F can also be supplied from the target's device $V_{cc} \geq 3V$ (via JTAG pin 2 or 4 – see [Figure 2-1](#)) if blow the security fuse option is not required.

2.6.1 Host Controller

To achieve maximum programming speed, the host controller MSP430F5437 can run at a maximum CPU clock frequency of 18MHz that is used from PLL and XTAL 12MHz provided on LFX1. The host is programmed via a dedicated JTAG port labeled Host JTAG (see [Figure 2-1](#)).

2.6.2 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) with extra two pins that can be used for BSL connection. The programmable target device supply voltage of 2.1V to 3.6 V with step 0.1V is available on pin 2 of this connector, eliminating the need for an additional supply for the target system. The required Spy-Bi-Wire or 4-wire JTAG and GND must be connected. (On devices requiring the TEST pin, the TEST signal also must be provided from the programmer to the target MSP430 device.) Host controller in the REP430F is supplied from the $V_{cc}=3.0V$, while target device can be supplied with the V_{cc} from 2.1V to 3.6V. To avoid a problem with I/O levels, the REP430V contains voltage level translators between target device and host controller. Voltage translators are supplied from the host controller $V_{cc}=3V$ from one side, and from the target's device V_{cc} (provided on pin 2 of the target JTAG connector) from the other side. That allows supply the target device with the I/O levels exactly as required by the target device.)

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

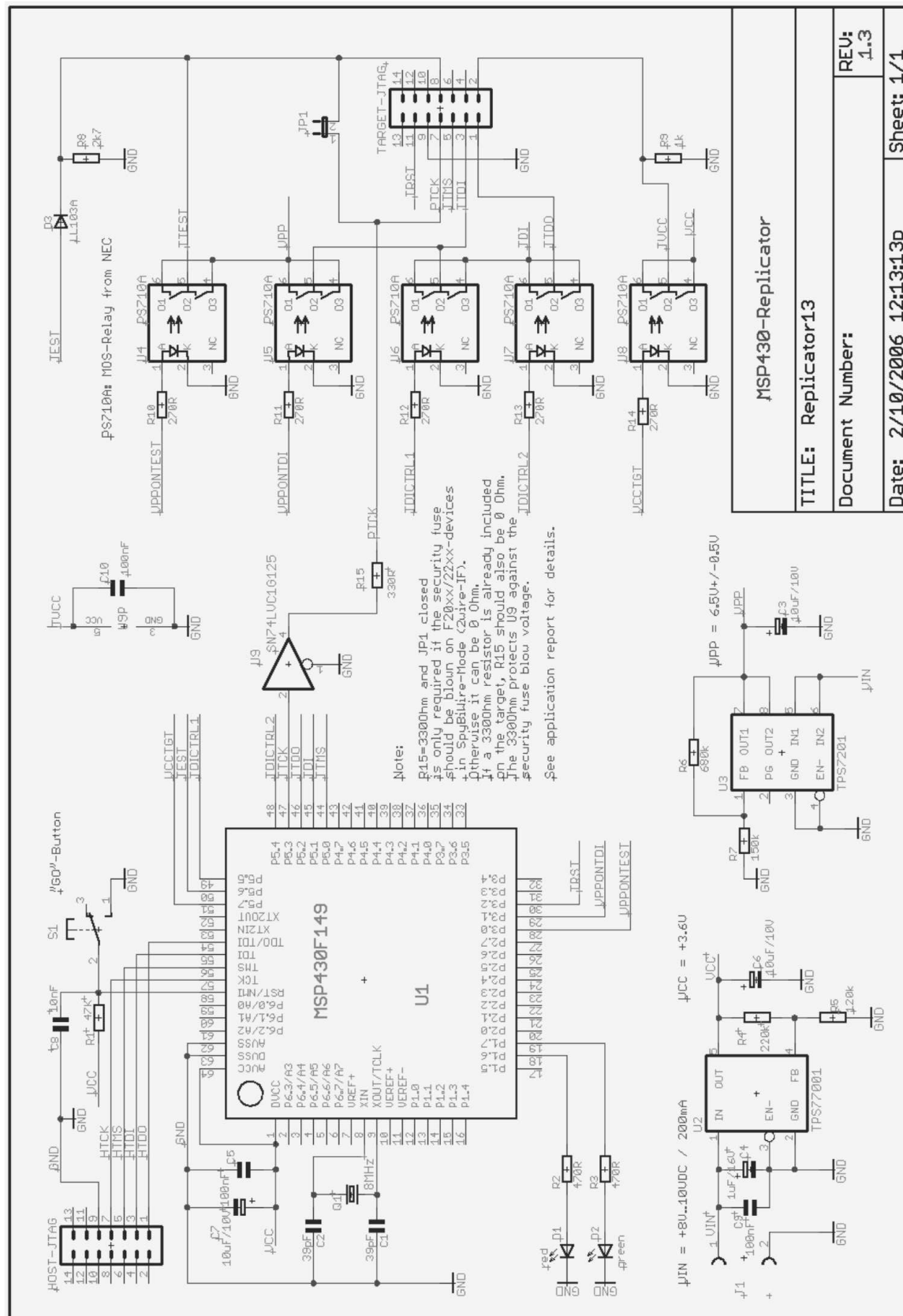


Figure 2-1. Replicator Application Schematic

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.

2.6.3 Host Controller/Programmer Power Supply

From the input voltage of 8 V to 10 V dc, two onboard voltages are generated using adjustable LDOs: U3 - $V_{CC} = 3.0$ V as supply voltage for the host controller MSP430F5437, U4 - $V_{CC} = 2.1$ V to 3.6 V supply voltage to the target device, and U1 - $V_{PP} = 6.7$ V to program the JTAG access fuse. While the fuse is being programmed, a peak current of 100 mA can flow through the TEST or TDI input pin (see the corresponding target MSP430 device data sheet).

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

2.6.4 Third Party Support

Elprotronic Incorporated offers a complete system as shown in this chapter, which is compatible with the software available with this application report. This information can be found by accessing the following URL: <http://www.elprotronic.com/rep430f.html>

Elprotronic Inc. 16
Crossroads Drive
Richmond Hill, ON, L4E5C9
Canada
Tel.: +905-780-5789
Fax: +905-780-2414
E-mail: info@elprotronic.com
Web site: www.elprotronic.com

Internal MSP430 JTAG Implementation

3.1 TAP Controller State Machine

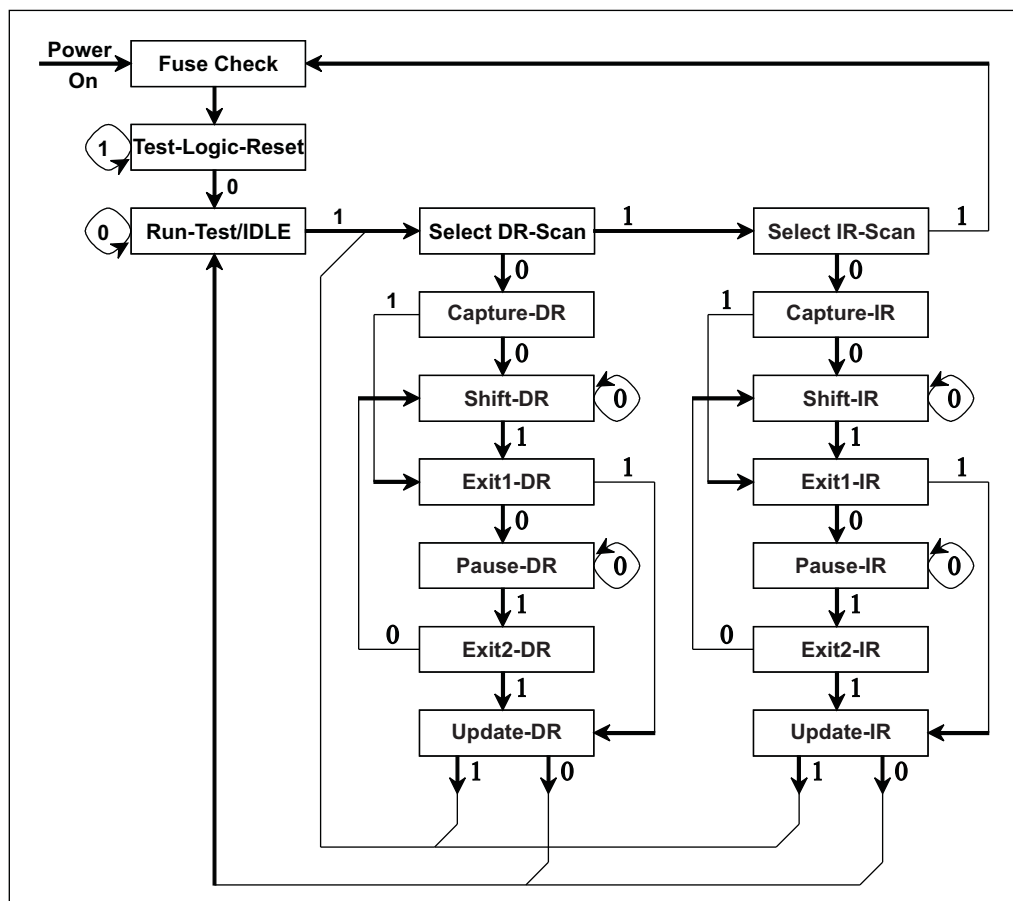


Figure 3-1. TAP Controller State Machine

3.2 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.

Errata and Revision Information

A.1 Known Issues

Description

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

Workaround

None

A.2 Revisions and Errata from Previous Documents

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

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

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

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

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		Applications	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless-apps