Programming a Flash-Based MSP430 Using the JTAG Interface

Markus Koesler, Franz Graf, Zack Albus *MSP430*

ABSTRACT

This application report details the functions required to erase, program, and verify the memory module of the MSP430 flash-based microcontroller family using the JTAG communication port, as well as how to program the JTAG access fuse, available on all MSP430 devices. In addition, a complete programmer system including software (source code is provided) and corresponding hardware implementation is demonstrated in Appendix A. This example is intended as a reference for development of similar MSP430 programmer solutions or as a stand-alone MSP430 flash programmer.

Contents

Figures

Tables

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. Focus is on the high-level JTAG functions used to access and program the flash memory and the respective timing, rather than the basic JTAG functionality.

The document contains four main elements. Section 2, *Interface and Instructions*, describes the required JTAG signals and associated pin functionality for 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 the target MSP430 via the JTAG interface. Section 3, *Memory Programming Control Sequences*, demonstrates use of the provided macros and function prototypes in a software-flow format that can be used to control the target device and program/erase the flash module. Section 4, *Programming the JTAG Access Protection Fuse*, details the fuse mechanism used to disable memory access via JTAG to the program code once it has been written to the target device's memory, eliminating for security purposes the possibility of undesired memory access. Finally, Appendix A illustrates development of a typical 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 a complete system that can be used directly or referenced for custom MSP430 programmer solutions.

NOTE: The MSP430 JTAG interface implements the test access port state machine (TAP controller) as specified by IEEE STD1149.1. 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 Appendix B, Figure 7.

2 Interface and Instructions

2.1 JTAG Interface Signals

The MSP430 family supports in-circuit programming of flash memory via the JTAG port, available on all MSP430 devices. The JTAG interface requires four signals (one additional signal is required on the 20- and 28-pin MSP430 devices), along with ground and V_{CC} (if powered externally). These signals are described in the table below:

Table 1. JTAG Interface 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 (20- and 28-pin devices only)

Using these signals, an interface connection to access the MSP430 JTAG port using a PC or other controller can be established. See the respective MSP430 device data sheets for the connections required by a specific MSP430 device. (Note: The RST/NMI input is not a required connection in order to establish JTAG communication with the MSP430 flash-based family.)

Two signals that are used in addition to the standard TMS, TCK, TDI and TDO signals are TCLK and TEST. 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 while the MSP430 TAP controller is in the Run-Test/Idle state. (TLCK input support on the MSP430 XOUT input exists but has been superseded by the TDI input on all current MSP430 flash-based devices. Existing FET tools, as well as the software provided with this application report, implement TCLK on the TDI input pin.)

The TEST input exists only on the 20 and 28-pin MSP430 devices. Due to low pin count, these devices share the I/O (input/output) functionality of Port1 with the JTAG signals. 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 can be connected to ground, enabling the shared pins as standard port I/O.

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, the MSP430-specific implementation used for memory access. For the purpose of this document, it is important to illustrate the instructions that need to be loaded into the JTAG instruction register, as well as when these instructions are required. The following section summarizes the macros used throughout this document and their associated functionality. (See the accompanying software for more information.)

Table 2. 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.
Delay (time)	Wait for the specified time in ms
SetTCLK	Set TCLK to 1
ClrTCLK	Set TCLK to 0
TDOvalue	Variable containing the last value shifted out on TDO

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 8 bits wide with the 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). The following timing diagram shows how to load the ADDR_16BIT instruction into the JTAG IR register. See Section 2.3 for a complete list of the JTAG interface communication instructions used to access the target device flash memory module.

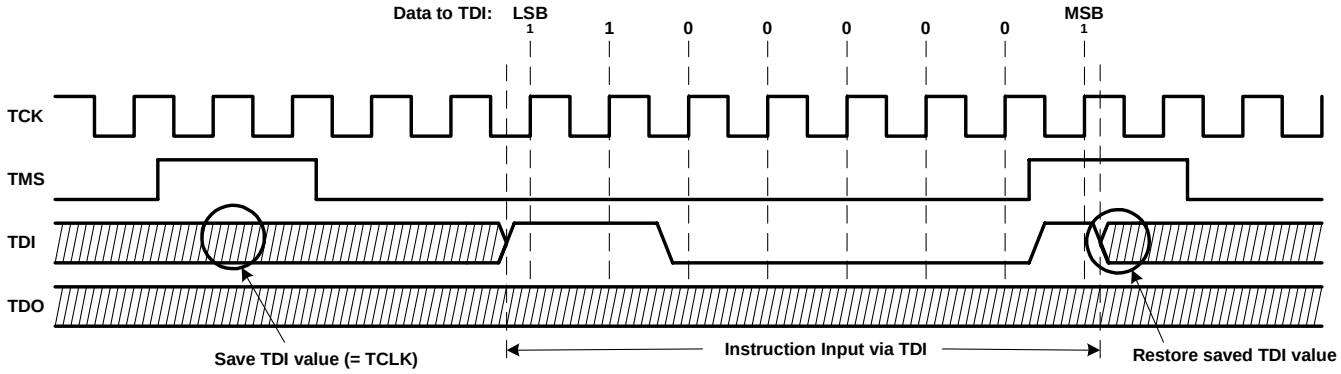


Figure 1. Timing Example for the IR_SHIFT (0x83) Instruction

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, 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. The following timing diagram shows how to load a 16-bit word into the JTAG DR and read out a stored value via TDO.

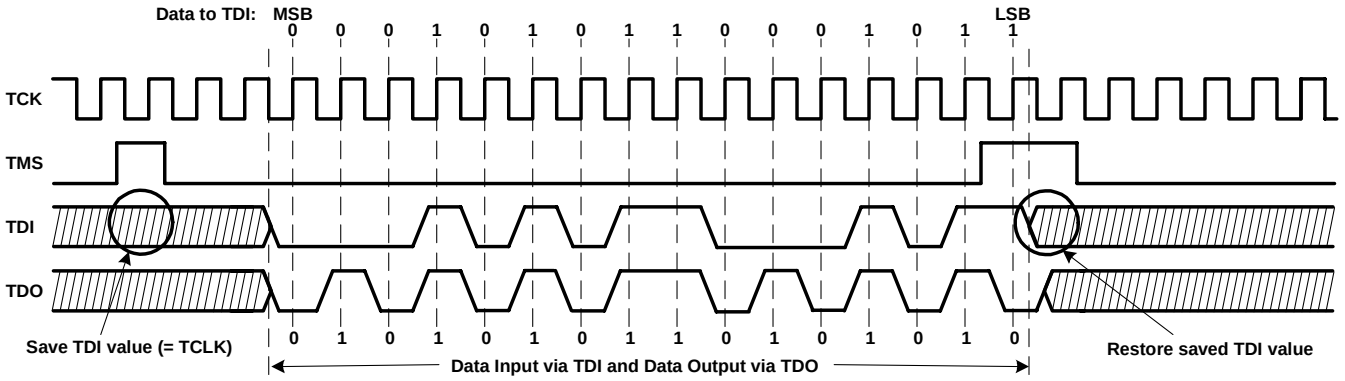


Figure 2. Data Register I/O: DR_SHIFT16 (0x158B) (TDO Output Is 0x55AA)

Delay(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 value.

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.

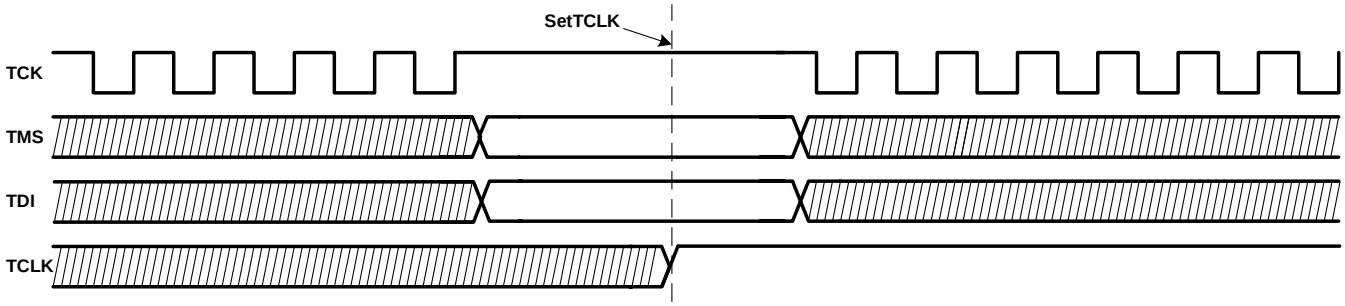


Figure 3. SetTCLK

ClrTCLK

This macro resets the TCLK input clock low. TCK and TMS must hold their last value while this action is performed.

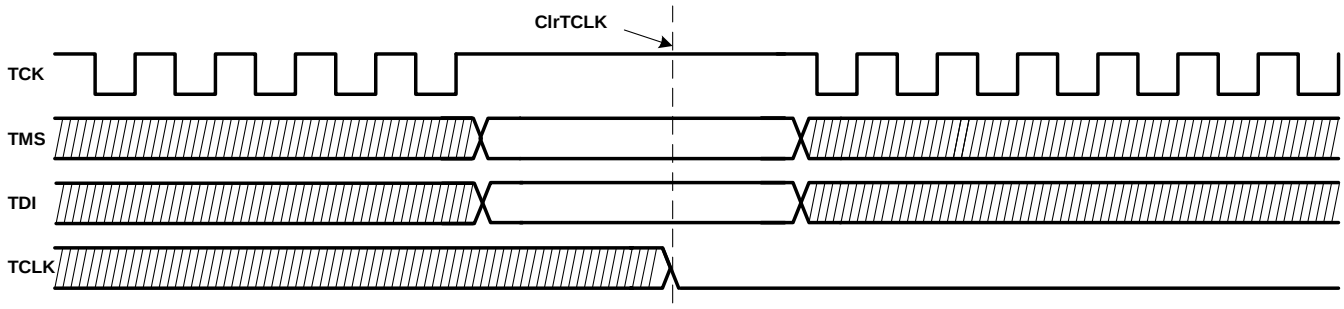


Figure 4. ClrTCLK

2.3 JTAG Communication Instructions

Selecting a JTAG register and controlling the CPU is done by shifting in a JTAG instruction using the IR_SHIFT macro described in the previous section. The following instructions that can be written to the JTAG IR are used to program the target flash memory. All instructions sent to the target MSP430 via the JTAG register are transferred LSB first.

Table 3. Memory Access Instructions

Instruction Name	8-Bit Instruction Value (Hex)
Controlling the MAB (Memory Address Bus)	
IR_ADDR_16BIT	0x83
IR_ADDR_CAPTURE	0x84
Controlling the MDB (Memory Data Bus)	
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 Signature Analysis)	
IR_DATA_PSA	0x44
IR_SHIFT_OUT_PSA	0x46
Fuse Programming	
IR_Prepare_Blow	0x22
IR_Ex_Blow	0x24

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

NOTE: When a new JTAG instruction is shifted into the JTAG instruction register, it takes effect with the UPDATE-IR state of the TAP controller. When accessing a JTAG data register, the last value written is captured with the CAPTURE-DR state and the new value shifted in becomes valid with the UPDATE-DR state.

2.3.1 Controlling the Memory Address Bus (MAB)

The following instructions control the memory address bus (MAB) of the target MSP430. To accomplish this, a 16-bit 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.

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. 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-bit address is shifted in via TDI.

IR_ADDR_CAPTURE

Enables readout of the data on the MAB with the next 16-bit data access. The MAB value is not changed during the 16-bit data access; that is, the 16-bit data sent on TDI with this command is ignored (0x0000 is sent as a default in the provided software).

2.3.2 Controling the Memory Data Bus (MDB)

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

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.

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.

IR_DATA_QUICK

This instruction enables setting of the MSP430 MDB to a specific value shifted in with the next 16-bit JTAG data access. The 16-bit MSP430 MDB is set to the value written to the JTAG MDB register. During the 16-bit data transfer, the previous MDB value is captured and shifted out on TDO. The MAB value is set by the program counter (PC) of the CPU. This instruction auto-increments the program counter by two on every falling edge of TCLK in order to automatically point to the next 16-bit memory location. The target CPU's program counter must be loaded with the starting memory address prior to execution of this instruction, which can be used to quickly read or write to a memory array. (See Section 3.2 for more information on setting the PC.)

IR_BYPASS

This instruction delivers the input to TDI as an output on TDO delayed by one TCK clock. When this instruction is loaded, the IR_CNTRL_SIG_RELEASE instruction, which is defined in the following section, is performed simultaneously. After execution of the bypass instruction, the 16-bit data shifted out on TDI does not affect any register of the target MSP430's JTAG control module.

2.3.3 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 4 describes the bit functions making up the JTAG control signal register used for memory access.

Table 4. JTAG Control Signal Register

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

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.

IR_CNTRL_SIG_CAPTURE

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

IR_CNTRL_SIG_RELEASE

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

2.3.4 Memory Verification via Signature Analysis

The following instructions support verification of the MSP430 memory content by means of a pseudo signature analysis (PSA) mode.

IR_DATA_PSA

The IR_DATA_PSA instruction switches the JTAG_DATA_REG into the pseudo signature analysis (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.

IR_SHIFT_OUT_PSA

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

2.3.5 JTAG Access Fuse Programming

The following instructions are used to access and program the built-in JTAG access protection fuse, available on every MSP430 flash device. Once the fuse is programmed (or blown), future access to the MSP430 via the JTAG interface is permanently disabled. This allows for access protection of the final MSP430 firmware programmed into the target device.

IR_PREPARE_BLOW:

This instruction sets the MSP430 into program-fuse mode.

IR_EX_BLOW:

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

3 Memory Programming Control Sequences

The content of this section demonstrates the use of the high-level macros and JTAG communication instructions described earlier as they pertain to communicating with the flash memory module of the target MSP430 device. In the following sub-sections, descriptions are given for the provided C-routines, which can be found in the JTAGfunc.c file, included in the software accompanying this application report. A summarized listing of each function is given in Section 5.

NOTE: It is recommended that when a device is first used, it be completely erased (information and program memory), as it is possible that some flash memory cells may have been programmed during production testing. To accomplish this, a mass erase should be performed prior to memory programming.

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.

Enable JTAG Access (20- and 28-Pin Devices Only)

To use the JTAG features of the 20- and 28-pin MSP430 devices, it is necessary to enable the shared JTAG pins for JTAG communication mode. (The larger-pin-count devices provide dedicated JTAG inputs/outputs and do not require this step.) The shared pins are enabled for JTAG communication by connecting the TEST pin to V_{DD}. For normal operation (non-JTAG mode), this pin must be biased to ground. Table 5 shows the Port1 pins that are used for JTAG communication.

Table 5. 20- and 28-Pin Package Device Shared Functions

Port1 Function	JTAG Function
P1.4	TCK
P1.5	TMS
P1.6	TDI/TCLK
P1.7	TDO

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

Reference function: ResetTAP

Each MSP430 family device includes a physical fuse used to permanently disable memory access via JTAG communication. When this fuse is programmed (or blown), access to memory via JTAG is permanently disabled and cannot be restored. When initializing JTAG access after power-up, a fuse check must be done before JTAG access is granted. Toggling of the TMS signal twice performs the check. It is recommended that a minimum of six TCK clocks be sent to the target device while TMS is high followed by setting TMS low for at least one TCK clock. This sets the JTAG state machine (TAP controller) to a defined starting point: the Run-Test/Idle state. This procedure can also be used at any time during JTAG communication to reset the JTAG port.

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

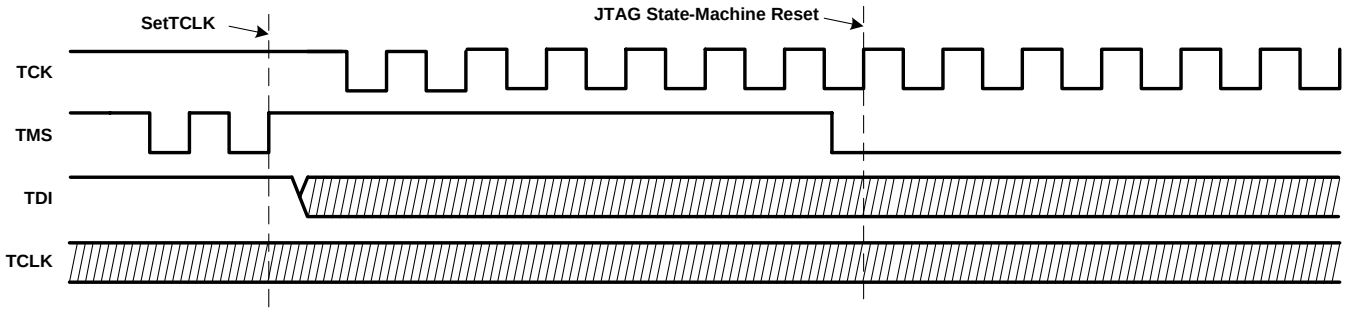


Figure 5. Fuse Check and TAP Controller Reset

Taking the CPU Under JTAG Control

Reference function: GetDevice

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

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

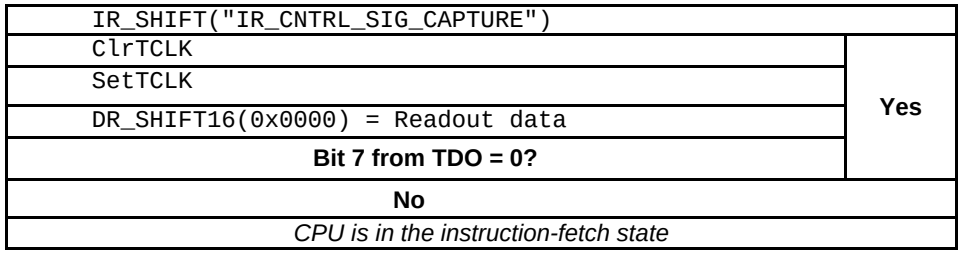
3.2 General Functions

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

Set CPU to Instruction-Fetch

Reference function: SetInstrFetch

Sometimes it is useful for the target device to execute directly 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.



Setting the Target CPU Program Counter (PC)

Reference function: SetPC

In order to use some of the features of the JTAG interface provided by the MSP430, setting of the CPU program counter (PC) of the target device is required. The following flow is used to accomplish this.

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

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.

CPU must be in the instruction-fetch state prior to the following sequence	
HaltCPU	IR_SHIFT("IR_CNTRL_SIG_16BIT")
	DR_SHIFT16(0x2401) : Set device into JTAG mode + read
	IR_SHIFT("IR_DATA_16BIT")
	DR_SHIFT16(0x3FFF) : "JMP $" instruction to keep CPU from changing the state
	SetTCLK
	ClrTCLK
	IR_SHIFT("IR_CNTRL_SIG_16BIT")
	DR_SHIFT16(0x2409) : set HALT_JTAG bit
	SetTCLK
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.*	
MEMORY ACCESS PERFORMED HERE	
The CPU is switched back to normal operation using HaltCPU.	
ReleaseCPU	ClrTCLK
	IR_SHIFT("IR_CNTRL_SIG_16BIT")
	DR_SHIFT16(0x2401) : Clear HALT_JTAG bit
	IR_SHIFT("IR_ADDR_CAPTURE")
	SetTCLK
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.	

Resetting the CPU While Under JTAG Control

Reference function: ExecutePUC

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 in order to avoid an undesired reset of the target.
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

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 0xFFFEh (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 0xFFFE.

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")
The target CPU starts program execution with the address stored at location 0x0FFFE (reset vector)

3.3 Accessing Non-Flash Memory Locations With JTAG

Read Access

Reference function: ReadMem

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.

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	
`SetTCLK`	
ReleaseCPU should now be executed, returning the CPU to normal operation.	

Write Access

Reference function: WriteMem

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.

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`	
`ClrTCLK`	
Write again?	
No	
ReleaseCPU should now be executed, returning the CPU to normal operation.	

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.

Flow for Quick Read (All Memory Locations)

Reference function: ReadMemQuick

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

Flow for Quick Write (RAM and Peripherial Memory Only)

Reference function: WriteMemQuick

Set PC to start address - 4 (SetPC)	
Switch CPU to stopped state (HaltCPU)	
`ClrTCLK`	
`IR_SHIFT("IR_CNTRL_SIG_16BIT")`	
`DR_SHIFT16(0x2408)  : Set RW to write`	
`IR_SHIFT("IR_DATA_QUICK")`	
`DR_SHIFT16("Data")  : Set data`	**Yes**
`SetTCLK`	
`ClrTCLK  : Auto-increments PC`	
Write to Next Address?	
No	
`SetTCLK`	
ReleaseCPU should now be executed, returning the CPU to normal operation. *Reset the target CPU's PC if needed (SetPC).*	

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

Reference function: WriteFLASH

This section describes one method available to program the flash memory module in an MSP430 device. It uses the same procedure that user-defined application software, which would be programmed into a production-equipment MSP430 device, would utilize. (Note: Non-consecutive flash memory addressing *is* supported.)

This programming method requires a TCLK frequency of 350 kHz ±100 kHz while the erase or programming cycle is being executed. (For more information on the flash controller timing, please see the corresponding MSP430 user's guide and specific device data sheet.) The following table shows the required minimum number of TCLK cycles, depending on the action performed on the flash (for FCTL2 register bits 0 thru 7 = 0x40 as defined in the MSP430 user's guide).

Table 6. Erase/Program Minimum TCLK Clock Cycles

FLASH Action	Minimum TCLK Count
Page erase	4820
Mass erase	5300
Program word	35

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. To program the information flash memory, the IFREN bit located in the same 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)	: 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)	: 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)	: Point to FCTL3 Address	
IR_SHIFT("IR_DATA_TO_ADDR")		
DR_SHIFT16(0xA500)	: Clear FCTL3 Register	
SetTCLK		
ClrTCLK		
IR_SHIFT("IR_CNTRL_SIG_16BIT")		**Yes**
DR_SHIFT16(0x2408)	: Set RW to Write	
IR_SHIFT("IR_ADDR_16BIT")		
DR_SHIFT16("Address")	: 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	*Repeat 34 times†*	
ClrTCLK		
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)	: Point to FCTL1 Address	
IR_SHIFT("IR_DATA_TO_ADDR")		
DR_SHIFT16(0xA500)	: Disable FLASH Write Access	
SetTCLK		
ReleaseCPU should now be executed, returning the CPU to normal operation.		

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

3.5 Reading From Flash Memory

Reference function: ReadMem or ReadMemQuick

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

3.6 Verifying the Flash Memory

Reference function: VerifyMem

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

3.7 Erasing the Flash Memory Using the Onboard Flash Controller

Reference function: EraseFLASH

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

Flow to Erase a Segment

Switch CPU to stopped state (HaltCPU)	
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2408)	: Set RW to Write
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x0128)	: 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)	: Point to FCTL2 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA540)	: Source is MCLK, divide by 1
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x012C)	: Point to FCTL3 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500)	: Clear FCTL3 Register
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(“EraseAddr”)	: Set Address for Erase‡
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 §*
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2408)	: Set RW to Write
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x0128)	: Point to FCTL1 Address
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500)	: Disable FLASH Erase
SetTCLK	
ReleaseCPU should now be executed, returning the CPU to normal operation.	
Erasing of flash-memory segment finished	

‡ The EraseAddr parameter is the 16-bit address pointing to the flash memory segment to be erased.
§ Correct timing required. Must meet min/max TCLK frequency requirement of 350 kHz ±100 kHz

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

In order to assure the recommended 200-ms erase time to safely erase the flash memory space, 5300 TCLK cycles are transmitted to the target MSP430 device and repeated 19 times (assuming a maximum TCLK frequency of 450 kHz).

Switch CPU to stopped state (HaltCPU)	*Repeat 19 times §*
ClrTCLK	
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2408) : set RW to write	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x0128) : 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) : 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) : FCTL3 address	
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500) : Clear FCTL3 register	
SetTCLK	
ClrTCLK	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(“EraseAddr”) : Set address for erase‡	
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 / ClrTCLK	*Repeat 5300 times §*
IR_SHIFT("IR_CNTRL_SIG_16BIT")	
DR_SHIFT16(0x2408) : set RW to write	
IR_SHIFT("IR_ADDR_16BIT")	
DR_SHIFT16(0x0128) : FCTL1 address	
IR_SHIFT("IR_DATA_TO_ADDR")	
DR_SHIFT16(0xA500) : Disable FLASH erase	
SetTCLK	
ReleaseCPU should now be executed, returning the CPU to normal operation.	
Erase of complete flash memory finished	

‡ The EraseAddr parameter is the 16-bit address pointing to the flash memory segment to be erased.
§ Correct timing required. Must meet min/max TCLK frequency requirement of 350 kHz ±100 kHz

4 Programming the JTAG Access Protection Fuse

Reference function: BlowFuse

Two similar methods are described and are to be implemented depending on the target MSP430 device family:

All devices having a TEST pin (20- and 28-pin MSP430 devices) use this input to apply the programming voltage, V_{PP}. As described earlier, these devices have shared-function JTAG interface pins. The higher pin count MSP430 devices having dedicated JTAG interface pins use the TDI pin for fuse programming.

Devices with a TEST pin:

Table 7. Devices With a TEST Pin: 20- and 28-Pin 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 8. 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/TDI	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 datasheet. For existing flash devices, the required voltage for V_{PP} is 6.5 V ±0.5 V.

4.1 Fuse Programming via TDI (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 8 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
TDI signal releases to target, TDI is now provided on TDO.
IR_SHIFT(“IR_PREPARE_BLOW”) (through TDO pin)
Delay(1) : Delay for 1ms
Connect V_{PP} to TDI pin
Wait until V_{PP} input has settled (depends on V_{PP} source)
IR_SHIFT(“IR_EX_BLOW”) : Sent to target via TDO
Delay(1) : Delay for 1ms
Remove V_{PP} from TDI pin
Switch TDI pin back to TDI function and reset the JTAG state machine (ResetTAP)

4.2 Fuse-Programming Voltage via TEST Pin (20- and 28-Pin Devices)

The same method is used to program the fuse for the 20- and 28-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”)
Delay(1) : Delay for 1ms
Connect V_{PP} to TEST pin
Wait until V_{PP} input has settled (depends on V_{PP} source)
IR_SHIFT(“IR_EX_BLOW”)
Delay(1) : Delay for 1ms
Remove V_{PP} from TEST pin
Reset the JTAG state machine (ResetTAP)

4.3 Testing for a Successfully Programmed Fuse

Reference function: IsFuseBlown

Once the fuse is programmed and a RESET (via the JTAG ExecutePUC command or the RST/NMI pin in hardware) has been issued, the only JTAG function available on the target MSP430 is BYPASS. When the target is in BYPASS, data sent from host to target is delayed by one TCK pulse and output on TDO, where it can be received by other devices downstream of the target MSP430.

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

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

5 JTAG Function Prototypes

The following descriptions detail the high-level C-language MSP430 JTAG functions available in the software provided with this application report (JTAGfunc.c). Many of these functions were described in the previous sections in order to implement the required JTAG communication sequences.

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)

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)

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

word ExecutePUC(void)

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

Arguments: None

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

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.)

void PrepTCLK(void)

Sends the target device's TAP controller back into the Run-Test/Idle state after a shift access occurs.

Arguments: None

Result: None

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

void HaltCPU(void)

Sends the target CPU into a controlled, stopped state.

Arguments: None

Result: None

void ReleaseCPU(void)

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

Arguments: None

Result: None

word 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 (0xFFFE: 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 *CodeArray)

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.s43. (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 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)

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)

References

MSP430Fxxx device data sheets
MSP430x1xx Family User's Guide, literature number SLAU049
MSP430x4xx Family User's Guide, literature number SLAU056
IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE STD1149.1

Third-Party Support

SoftBaugh, Incorporated offers a complete system as shown in Appendix A, which is compatible with the software available with this application report: the MSP430 Flash Programming Replicator. This information can be found by accessing the following URL: http://www.softbaugh.com/ExtREP430.html

SoftBaugh, Inc.
5040 Oakmont Bend Drive
Alpharetta, GA 30004

Tel.: 800-794-5756
Fax: 770-772-9030

E-mail: e-mail info@softbaugh.com
Web site: www.softbaugh.com

Appendix A: Example Programmer Implementation

A.1 Implementation Overview

The following example demonstrates the application of the software functions described in the previous sections using an MSP430F149 as the host controller that programs target MSP430 flash-based devices. The complete C source code and project files are provided in the attachment accompanying this application report. A schematic for the system as implemented in this discussion is also provided.

Key features of the JTAG replicator programmer implementation are as follows:

- Supports all MSP430 flash-based devices
- Maximum target device program code size: approximately 57 KB
- Programming speed (Erase, Program, Verify): approximately 8 KB in 1.5 seconds, 48 KB in 8 seconds
- Fast verify and erase check: 17 KB/10 ms
- Supports programming of the JTAG access fuse (permanently disables device memory access via JTAG)
- Stand-alone target programming operation (no personal computer or additional supporting hardware/software required)

A.2 Software Operation

The host controller stores the JTAG communication protocol code as well as the target program in its flash memory (61 KB available on the MSP430F149). The programming software (S/W) 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 (H/W) reset and starts the host controller's JTAG communication routine in order to erase, program, and verify the target device. While the system is active, two LEDs on the programmer board are on; after successful completion only the green LED is on. If an error has occurred or communication to the target device has failed, only the red LED remains on. The entire procedure takes approximately 3 seconds for a target program size of 8 KB. (Some code not strictly required to erase/program/verify the target MSP430 is executed at the end of the Replicator.c source file, increasing the specified programming times. These additional instructions can be customized to fit the individual system programming requirements.)

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

A.3 Software Structure

The programming S/W is partitioned in three levels and consists of five files in addition to the target program:

- Top level: Specifies which programming functions (erase, program, verify, blow fuse) are to be executed.
  - Replicator.c: Consists of the main section, which can be modified to meet custom requirements. In the main section of this program, the target device is completely erased, checked for successful erasure, and programmed. Programming loads the provided example code 'Target_code.s43' to the target device's memory space. (The provided Target_code.s43 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 complied 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.

- JTAG functions: All MSP430-specific functions are defined here. These files should not be modified under any circumstance.
  - JTAGfunc.c: Contains the MSP430-specific functions needed for flash programming
  - JTAGfunc.h: Contains constant definitions and function prototypes used for JTAG communication

- Low-level functions: All functions that depend specifically on the host controller (JTAG port I/O and timing functions) are located here. These files need to be adapted if a host controller other than the MSP430F149 is implemented.
  - LowLevelFunc.c: Contains the basic host-specific functions
  - LowLevelFunc.h: Contains host-specific definitions and function prototypes

As mentioned previously, the target device's program code has to be supplied in a separate file, Target_Code.s43, which must be included in the project space of the program to be sent to the host MSP430. The format of the target file is a data array in assembly source code (.s43) made up of the target program's compiled instructions. To build this file from the TI-.txt format, a conversion program called FileMaker.exe is provided. TI-.txt format can be output by the IAR Linker by setting the required complier/linker options. (Please refer to the IAR tool instruction guides for more information.) FileMaker.exe can optionally make a Target-Code.h file, which contains the data array in C source code. In the provided example, the default user project is located in the directory Target_Code.

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

A.4 Programmer Operation

Following is a step-by-step procedure that demonstrates how the JTAG replicator programmer could be used together with any MSP430 FETXXX development tool using the IAR MSP430 development environment:

1. Generate the user.txt file of the target program in the target project using the IAR Linker with following options which are to be set from the IAR Workbench:
   – Project -> Options -> XLINK -> Format: Other: Output Format: msp430-txt
   – Run Make.
   – The output file user.txt is located in the target_dir\exe\ directory.
2. Convert the user.txt file to user.s43 using the FileMaker.exe conversion program (provided with this application report) and copy it to the programmer project directory.
3. Open the programmer project in the IAR Workbench.
4. In the project window the default file Target_Code.s43 must be replaced by the user.s43 user program.
   – Delete Target_Code.s43 from the project window.
   – Add the target code file user.s43 to the programmer project space.
5. Run *Make* from the programmer project environment.
6. Download the programmer object code into the host controller by starting C-Spy using the FET.
7. The programmer can be disconnected from the FET after download is complete
8. Connect power and the target device to the host system and push the GO button to program.

A.5 Hardware Setup

The hardware (H/W) consists of the host controller MSP430F149, five semiconductor relays, two voltage regulators and two JTAG interface connectors. An external power supply delivering 8 V to 10 V dc at 200 mA is required for operation (see Figure 6).

Host Controller

To achieve maximum programming speed, the host controller MSP430F149 runs at a maximum CPU clock frequency of 8 MHz, provided on LFXT1. CPU operation at this frequency requires a supply voltage of 3.6 V for the host controller, which is provided by U3 in the schematic. The host is programmed via a dedicated JTAG port labeled Host JTAG (see Figure 6).

Target Connection

The target MSP430 device is connected to the host controller/programmer through the remaining 14-pin connector labeled Target JTAG, which has the same standard signal assignment as all available MSP430 tools (FET and PRGS tools). The host supply voltage of 3.6 V is also available on pin 2 of this connector, eliminating the need for an additional supply for the target system, but does not have to be used at the target. At a minimum, the four JTAG signals and GND must be connected. (On devices requiring the TEST pin, the TEST signal also must be provided from the programmer to the target MSP430 device.)

To enable programming of all MSP430 flash-based devices including a JTAG access fuse, five semiconductor relays are used which are controlled by the host MSP430. Relay U4 controls V_{PP} on devices with a TEST pin; U5 connects V_{PP} to TDI on devices not requiring a TEST signal. U6 isolates the host controller from the target TDI pin while V_{PP} is connected to the target TDI input. U7 connects the host TDI signal to the target TDO pin while the fuse is programmed (for devices without a TEST pin). U8 controls availability of V_{CC} to the target device. The host controller program includes delays, which consider a relay switching time of a maximum of 5 ms. U4 and U5 should have a $R_{ON} < 1.0\ \Omega$ 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).

The following recommended relays meet the above requirements:

NAIS	AQV212 (as shown in Figure 6)
NEC	PS710A
Matsushita	AQV251
Matsushita	AQY211EH
CP Clare	LCA710

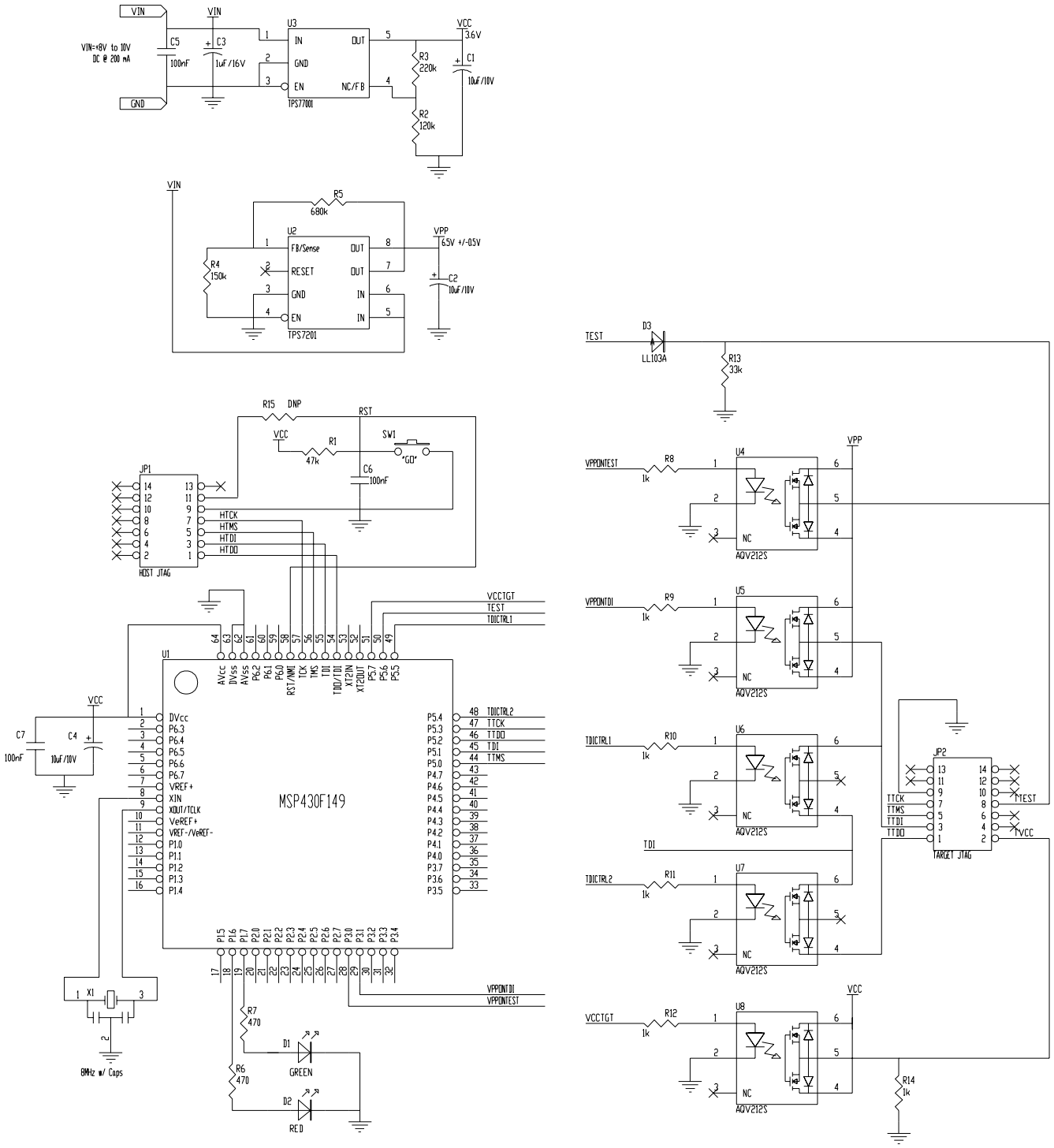


Figure 6. Example Programmer Schematic

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.

Host Controller/Programmer Power Supply

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

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

Appendix B: TAP Controller State Machine

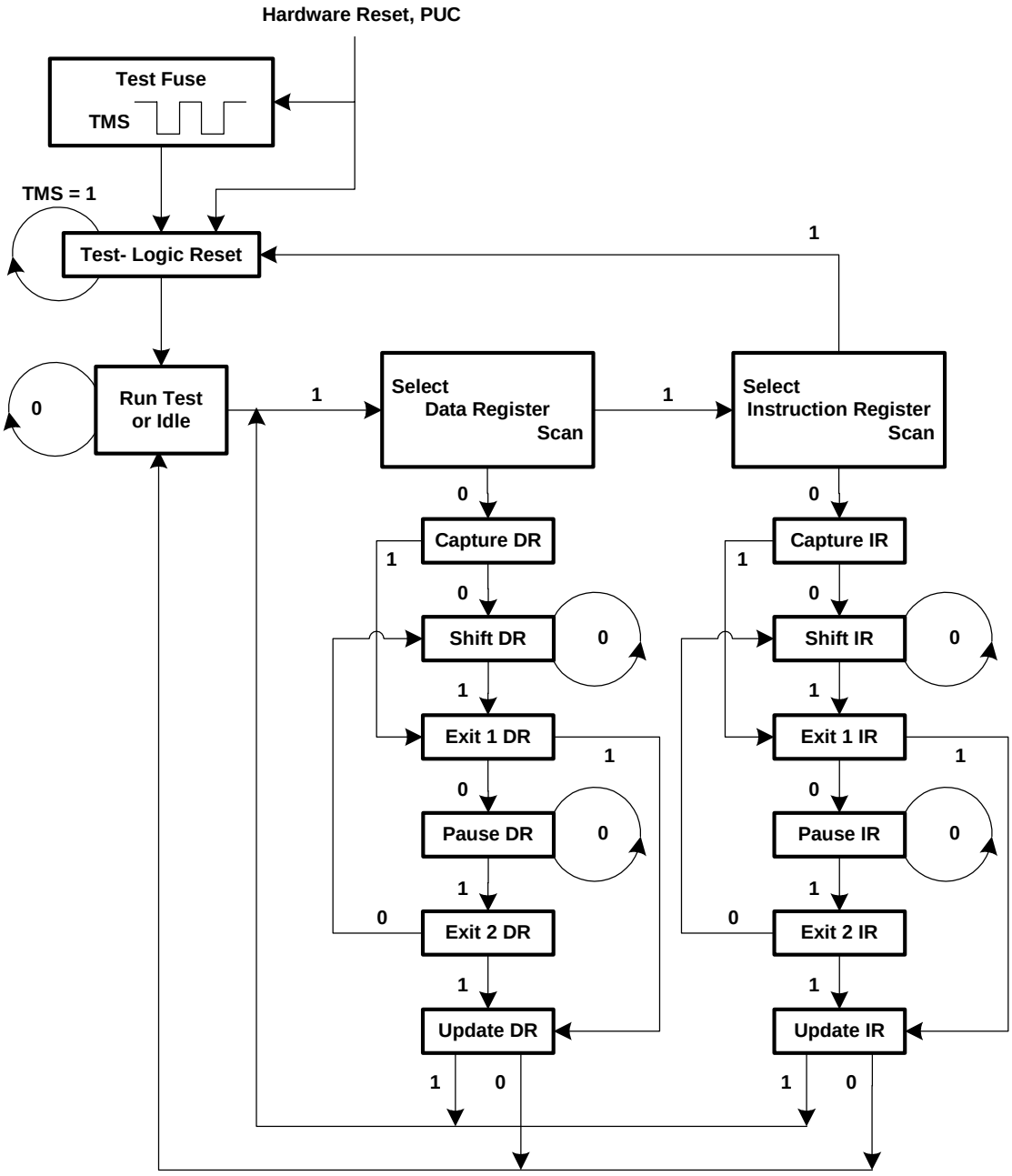


Figure 7. TAP Controller State Machine

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 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.

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.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265