In-System Programmability Guidelines

May 1999, ver. 3 Application Note 100

Introduction

As time-to-market pressures increase, design engineers require advanced system-level products to ensure problem-free development and manufacturing. Programmable logic devices (PLDs) with in-system programmability (ISP) can help accelerate development time, facilitate in-field upgrades, simplify the manufacturing flow, lower inventory costs, and improve printed circuit board (PCB) testing capabilities. Altera® ISP-capable devices can be programmed and reprogrammed in-system via the IEEE Std. 1149.1 Joint Test Action Group (JTAG) interface. This interface allows devices to be programmed and the PCB to be functionally tested in a single manufacturing step, saving testing time and assembly costs. This application note describes guidelines you should follow to design successfully with ISP, including:

- General ISP Guidelines
- IEEE Std. 1149.1 Signals
- Sequential vs. Concurrent Programming
- ISP Troubleshooting Guidelines
- ISP via Embedded Processors
- ISP via In-Circuit Testers

General ISP Guidelines

This section provides guidelines that will help you design successfully for ISP-capable devices. These guidelines should be used regardless of your specific design implementation.

Operating Conditions

Each Altera device has several parametric ratings, or operating conditions, that are required for proper operation. Although Altera devices can exceed these conditions when in user mode and still operate correctly, these conditions should not be exceeded during in-system programming. Violating any of the operating conditions during in-system programming can result in programming failures or incorrectly programmed devices.

V_{CCISP} Voltage

All Altera ISP-capable devices have a specification called V_{CCISP}. The V_{CCISP} level must be maintained on the VCCINT pins (i.e., $V_{CCINT} = V_{CCISP}$) during in-system programming to ensure that the device's EEPROM cells are programmed correctly. The V_{CCISP} specification applies for both commercial- and industrial-temperature-grade devices.

Because power consumption during in-system programming can exceed the power consumption during user mode, you may need to adjust your in-system programming setup to maintain correct voltage levels during both modes. Altera recommends that you test the V_{CCISP} levels on the device's VCCINT pins using an oscilloscope. First, test the V_{CCISP} levels with the oscilloscope's trigger level set to the minimum V_{CC} level listed in the recommended operating conditions table in the appropriate device family data sheet. Measure the voltage between VCCINT and ground, probed at the pins of the device. Then, repeat this test with the oscilloscope's trigger level set to the maximum V_{CC} level listed in the recommended operating conditions table. If the oscilloscope is triggered at either voltage level, you should adjust your programming setup.

Input Voltages

Each device family data sheet lists device input voltage specifications in the absolute maximum ratings and recommended operating conditions tables. The input voltages in the absolute maximum ratings tables refer to the voltages beyond which the device risks permanent damage. For example, MAX® 9000 devices have a maximum input voltage of 7.0 V and a minimum input voltage of –2.0 V.

The recommended operating conditions tables specify the voltage range for safe device operation. All devices can operate safely with input voltages between (ground – 1.0 V) and (V_{CCINT} + 1.0 V), and with input currents up to 100 mA. Make sure all pins that transition during in-system programming do not have a ground or V_{CC} overshoot. Overshoot problems typically occur on free-running clocks or data buses that can toggle during in-system programming. All pins that have an overshoot greater than 1.0 V must have series termination.

For more information on operating conditions and termination, see the *Operating Requirements for Altera Devices Data Sheet* and *Application Note 75 (High-Speed Board Designs)*, respectively.

Interrupting In-System Programming

Altera does not recommend interrupting the programming process because partially programmed devices operate unpredictably. Partially programmed devices also cause signal conflicts, which can lead to permanent device damage and can affect the proper operation of other devices on the board.

MultiVolt Devices & Power-Up Sequences

For the JTAG circuitry to operate correctly during in-system programming or boundary-scan testing, all devices in a JTAG chain must be in the same state. Therefore, in systems with multiple power supply voltages, the JTAG pins must be held in the test-logic-reset state until all devices in the chain are completely powered up. This procedure is particularly important because systems with multiple power supplies cannot power all voltage levels simultaneously.

Altera devices with the MultiVolt™ feature can use two power supply voltages: V_{CCINT} and V_{CCIO}. V_{CCINT} provides power to the JTAG circuitry; V_{CCIO} provides power to output drivers for all pins, including TDO. Therefore, when these devices use two power supply voltages, the JTAG circuitry must be held in the test-logic-reset state until both power supplies are turned on. If the JTAG pins are not held in the test-logic-reset state, in-system programming errors can occur.

V_{CCINT} Powered before V_{CCIO}

If V_{CCINT} is powered up before V_{CCIO}, the JTAG circuitry is active but unable to drive signals out. Thus, any transition on TCK can cause the state machine to transition to an unknown JTAG state. If TMS and TCK are connected to V_{CCIO} and V_{CCIO} is not powered up, the JTAG signals are left floating. These floating values can cause the device to transition to unintended JTAG states, leading to incorrect operation when V_{CCIO} is finally powered up. Therefore, all JTAG signals must be disabled as described in "Disabling IEEE Std. 1149.1 Circuitry" on page 5.

V_{CCIO} Powered before V_{CCINT}

If V_{CCIO} is powered up before V_{CCINT}, the JTAG circuitry is not active but TDO is tri-stated. Even though the JTAG circuitry is not active, if the next device in the JTAG chain is powered up with the same trace as V_{CCIO}, its JTAG circuitry must stay in the test-logic-reset state. Because all TMS and TCK signals are common, they must be disabled for all devices in the chain. Therefore, the JTAG pins must be disabled by pulling TCK low.

I/O Pins Tri-Stated during In-System Programming

All device I/O pins are tri-stated during in-system programming. In addition, MAX 7000S, MAX 7000A, MAX 7000AE, MAX 7000B, and MAX 3000A devices have a weak pull-up resistor. The purpose of this weak pull-up resistor is to eliminate the need for external pull-ups on unused I/O pins. The value of this pull-up resistor is listed in the individual device family data sheets.

Sufficient pull-up or pull-down resistors must be added on signals that require a particular value during in-system programming (e.g., output enable or chip enable signals). If a pull-up or pull-down resistor is not added, the device could have high current during in-system programming (caused by conflicts on the board), in-system programming failures with either unrecognized device or verify errors, or a power-up after in-system programming fails.

IEEE Std. 1149.1 Signals

This section provides guidelines for programming with the IEEE Std. 1149.1 (JTAG) interface.

TCK Signal

Most in-system programming failures are caused by a noisy TCK signal. Noisy transitions on rising or falling edges can cause incorrect clocking of the IEEE Std. 1149.1 Test Access Port (TAP) controller. Incorrect clocking can cause the state machine to transition to an unknown state, leading to in-system programming failures.

Further, because the TCK signal must drive all IEEE Std. 1149.1 devices in the chain in parallel, the signal may have a high fan-out. Like any other high fan-out user-mode clock, you must manage a clock tree to maintain signal integrity. Typical errors that result from clock integrity problems are invalid ID messages, blank-check errors, or verification errors.

Altera recommends pulling the TCK signal low through a resistor. Typical resistor values are 1 kΩ to 5 kΩ, depending on the amount of current being consumed and the number of devices on the board.

Fast TCK edges combined with board inductance can cause overshoot problems. When this combination occurs, you must either reduce inductance on the trace or reduce the switching rate by selecting a transistor-to-transistor logic (TTL) driver chip with a slower slew rate. Altera does not recommend using resistor and capacitor (RC) networks to slow down edge rates, because they can violate the device's input specifications. In most cases, using a driver chip prevents the edge rate from being too slow. Altera recommends using driver chips that do not glitch upon power-up.

Programming via a Download Cable

If you are using the MasterBlaster™, ByteBlasterMV™, ByteBlaster™, or BitBlaster™ download cable and your JTAG chain contains three or more devices, Altera recommends adding a buffer to the chain. You should select a buffer with slow transitions to minimize noise.

If you must extend the download cable, you can attach a standard PC parallel or serial port cable to the download cable. Do not extend the 10-pin header portion of the download cable; extending this portion of the cable can cause noise and in-system programming problems.

For more information on using the MasterBlaster, ByteBlasterMV, ByteBlaster, or BitBlaster download cables, see the *MasterBlaster Serial/USB Communications Cable Data Sheet*, *ByteBlasterMV Parallel Port Download Cable Data Sheet*, *ByteBlaster Parallel Port Download Cable Data Sheet*, or *BitBlaster Serial Download Cable Data Sheet*.

Disabling IEEE Std. 1149.1 Circuitry

If your design does not use ISP or boundary-scan test (BST) circuitry, Altera recommends disabling the IEEE Std. 1149.1 circuitry. Table 1 summarizes how to disable the IEEE Std. 1149.1 circuitry when it is not in use.

Table 1. Disabling IEEE Std. 1149.1 Circuitry

Devices	Permanently Disabled	Enabled for ISP & BST, Disabled During User Mode
MAX 7000S MAX 7000B *(1)* MAX 7000A *(1)* MAX 7000AE *(1)* MAX 3000A *(1)*	In the MAX+PLUS® II software, turn off the *Enable JTAG Support* option.	Either pull TMS high and TCK low, or pull TMS high before pulling TCK high. *(2)*
MAX 9000 MAX 9000A	Either pull TMS high and TCK low, or pull TMS high before pulling TCK high. *(2)*	Either pull TMS high and TCK low, or pull TMS high before pulling TCK high. *(2)*

Notes:

(1) Information on MAX 7000B, MAX 7000A, MAX 7000AE, and MAX 3000A devices is preliminary.

(2) Typical pull-up resistor values are 1 kΩ to 5 kΩ. This value may vary depending on the amount of current being consumed and the number of devices on the board.

JTAG Permanently Disabled (MAX 7000S, MAX 7000B, MAX 7000A, MAX 7000AE & MAX 3000A Devices)

MAX 7000S, MAX 7000B, MAX 7000A, MAX 7000AE, and MAX 3000A device JTAG pins can be used as either JTAG ports or I/O pins. You should specify how the pins will be used before compiling your design in the MAX+PLUS II software by turning the *Enable JTAG Support* option on or off. When the *Enable JTAG Support* option is turned on, the pins act as JTAG ports for in-system programming and boundary-scan testing; when the *Enable JTAG Support* option is turned off, the pins act as I/O pins and you cannot perform in-system programming or boundary-scan testing.

For more information on how to disable the JTAG circuitry using the MAX+PLUS II software, search for "Classic & MAX Global Project Device Options Dialog Box" or "Classic & MAX Individual Device Options Dialog Box" in MAX+PLUS II Help.

JTAG Permanently Disabled (MAX 9000 & MAX 9000A Devices)

The JTAG circuitry is always enabled in MAX 9000 and MAX 9000A devices because they have dedicated JTAG pins and circuitry. Therefore, if you do not plan to use the ISP and BST circuitry, you can disable the circuitry through the JTAG pins. To disable JTAG, the JTAG specification instructs you to pull TMS high but does not explain what to do with TCK. Altera recommends pulling TMS high and TCK low. Pulling TCK low ensures that a rising edge does not occur on TCK during the power-up sequence.

You can pull TCK high, but you must first pull TMS high. Pulling TMS high first ensures that the rising edge or edges on TCK do not cause the JTAG state machine to leave the test-logic-reset state.

JTAG Enabled for ISP/BST & Disabled in User Mode

For Altera ISP-capable devices that use JTAG for either in-system programming or boundary-scan testing, the JTAG circuitry must be enabled during ISP and BST but disabled at all other times. You control JTAG operation through the JTAG pins. To permanently disable the JTAG circuitry on MAX 9000 devices, either pull TMS high and TCK low, or pull TMS high before pulling TCK high.

Working with Different Voltage Levels

When devices in a JTAG chain operate at different voltage levels, a device's output voltage specification must meet the subsequent device's input voltage specification. If the devices do not meet this criteria, you must add additional circuitry, such as a level-shifter, to adjust the voltage levels. For example, when a 5.0-V device drives a 2.5-V device, you must adjust the 5.0-V device's output voltage to meet the 2.5-V device's input voltage specification.

Because all devices in a JTAG chain are tied together, you must also ensure that the first device's `TDO` output meets the subsequent device's `TDI` input voltage specification to program a chain of devices successfully.

All Altera ISP-capable devices include a MultiVolt I/O feature, which allows these devices to interface with systems that have different supply voltages. All 5.0-V MultiVolt devices can be set for 3.3-V or 5.0-V I/O operation. All 3.3-V MultiVolt devices can be set for 2.5-V, 3.3-V, or 5.0-V I/O operation.

Sequential vs. Concurrent Programming

This section describes how to program multiple devices using sequential and concurrent programming. For more information on sequential and concurrent programming, see *Product Information Bulletin 26 (Concurrent Programming through the JTAG Interface for MAX Devices)*.

Sequential Programming

Sequential programming is the process of programming multiple devices in a chain one device at a time. After the first device in the chain is finished being programmed, the next device is programmed. This sequence continues until all specified devices in the JTAG chain are programmed. After a device is programmed, it uses the JTAG BYPASS instruction to pass data to subsequent devices in the chain. However, any device loaded with the JTAG BYPASS instruction will, by definition, operate in normal user mode.

Concurrent Programming

Concurrent programming is used to program devices from the same family in parallel. The time required to program multiple devices concurrently is only slightly longer than the time required to "burn" data into the largest device's EEPROM or FLASH cells, resulting in considerably faster programming times than sequential programming. Higher clock rates for shifting data result in an even greater time savings. However, using a parallel port rather than a serial port to transfer data greatly reduces the time savings because serial ports have limited bandwidth. Because FLEX® 10K devices are SRAM-based, they do not "burn" data and thus support serial configuration only.

Selecting Sequential or Concurrent Programming

When programming using a Programmer Object File (**.pof**) and a MasterBlaster, ByteBlasterMV, ByteBlaster, or BitBlaster download cable, sequential programming is selected automatically. When using a Jam™ File (**.jam**) or Serial Vector Format (**.svf**) File, devices are programmed or configured in the following order:

1. FLEX 10K devices sequentially
2. APEX™ 20K devices sequentially
3. MAX 7000S and MAX 7000A devices concurrently
4. MAX 7000AE and MAX 3000A devices concurrently
5. EPC2 devices sequentially
6. MAX 9000 devices concurrently

You can perform sequential programming with a Jam or SVF File if you create individual files for each device. In this scheme, FLEX and APEX devices will not begin configuration until you click the **Configure** button in the MAX+PLUS II Programmer.

Devices in Different Modes

Errors can occur if some devices in the chain are operational while others are still being programmed. For this reason, MAX 7000S, MAX 7000A, MAX 7000AE, MAX 7000B, and MAX 3000A devices use a special ISP instruction that prevents the devices from entering normal operation until all devices in the chain finish in-system programming. In this mode, these devices pass all boundary-scan data synchronously and wait for all other devices in the same family to complete programming before beginning operation. Thus, all of these devices begin operation simultaneously. APEX 20K, FLEX 10K, MAX 9000, and MAX 9000A devices do not currently support this mode. These devices are held in tri-state mode by the programming software until all device families have been programmed or configured.

ISP Trouble-shooting Guidelines

This section provides a few tips for troubleshooting ISP-related problems.

Invalid ID & Unrecognized Device Messages

The first step during in-system programming is to check the device's silicon ID. If the silicon ID does not match, an `Invalid ID` or `Unrecognize Device` error is generated. Typical causes for this error are shown below:

- Download cable connected incorrectly
- `TDO` is not connected
- Incomplete JTAG chain
- Noisy `TCK` signal
- Jam Player ported incorrectly

Download Cable Connected Incorrectly

You will receive an error if the download cable is connected incorrectly to the parallel port or if it is not receiving power from your board.

For more information on installing the MasterBlaster, ByteBlasterMV, ByteBlaster, or BitBlaster download cable, see the *MasterBlaster Serial/USB Communications Cable Data Sheet*, *ByteBlasterMV Parallel Port Download Cable Data Sheet*, *ByteBlaster Parallel Port Download Cable Data Sheet*, or *BitBlaster Serial Download Cable Data Sheet*.

TDO Is Not Connected

You will receive an error if the `TDO` port of one device in the chain is not connected. During in-system programming, data must be shifted in and out of each device in the JTAG chain through the JTAG pins. Therefore, each device's `TDO` port must be connected to the subsequent device's `TDI` port, and the last device's `TDO` port must be connected to the download cable's `TDO` port.

Incomplete JTAG Chain

You will receive an error if the JTAG chain is not complete. To check if an incomplete JTAG chain is causing the error, use an oscilloscope to monitor vectors coming out of each device in the chain. If each device's `TDO` port does not toggle during in-system programming, your JTAG chain is not complete.

Noisy TCK Signal

Noise on the TCK signal is the most common reason for in-system programming errors. Noisy transitions on rising or falling edges can cause incorrect clocking of the IEEE Std. 1149.1 TAP controller, causing the state machine to be lost and in-system programming to fail. For more information on dealing with noisy TCK signals, See "TCK Signal" on page 4.

Jam Player Ported Incorrectly

You will receive an error if the Jam Player was not ported correctly for your platform. To check if the Jam Player is causing the error, apply the IDCODE instruction to the target device using a Jam File. You can use a Jam File to load an IDCODE instruction and then shift out the IDCODE value. This test determines if the JTAG chain is set up correctly and if you can read and write to the JTAG chain properly. Figure 1 on page 13 shows a sample file you can use to read the IDCODE.

Troubleshooting Tips

This section discusses some additional suggestions for troubleshooting ISP issues.

Verify the JTAG Chain Continuity

For in-system programming to occur successfully, the number of devices physically in the JTAG chain must match the number reported in the MAX+PLUS II software. The following steps show one simple way to verify that the JTAG chain is connected properly.

1. In the MAX+PLUS II Programmer, choose **Multi-Device JTAG Chain Setup**.

2. In the **Multi-Device JTAG Chain Setup** dialog box, click the **Detect JTAG Chain Info** button. The MAX+PLUS II software reports how many devices it found on the JTAG chain.

Check the V_{CC} Level of the Board During In-System Programming

Using an oscilloscope, monitor the V_{CCINT} signal on your JTAG chain and set the trigger to the minimum V_{CC} level listed in the recommended operating conditions table of the appropriate device family data sheet. If a trigger occurs during in-system programming, the devices may need more current than is being supplied by the existing power supply. Try replacing the existing power supply with one that provides more current.

Power-Up Problems

Excessive voltage or current on I/O pins during power-up can cause one of the devices in the JTAG chain to experience latch-up. Check if any of the devices are hot to the touch; hot devices have probably experienced latch-up and may have been damaged. In this situation, check all voltage sources to make sure that excessive voltage or current is not being fed into the device. Then, replace the affected device and try programming again.

Random Signals on JTAG Pins

During normal operation, each device's TAP controller must be in the test-logic-reset state. To force the device back into this state, try pulling the TMS signal high and pulsing the TCK clock signal six times. If the device then powers-up successfully, you must add a higher pull-down resistor on the TCK signal.

Software Issues

Failures during in-system programming are occasionally related to the MAX+PLUS II software. All software-related issues are documented in the Altera Technical Support (Atlas^{SM}) section of the Altera web site at **http://www.altera.com**. Simply search the Atlas database for information relating to software issues that interfere with in-system programming.

ISP via Embedded Processors

This section provides guidelines for programming ISP-capable devices using the Jam programming and test language and an embedded processor.

Processor & Memory Requirements

The Jam Byte-Code Player supports 8-bit and higher processors; the ASCII Jam Player supports 16-bit and higher processors. The Jam Player uses memory in a predictable manner, which simplifies in-field upgrades by confining updates to the Jam File. The Jam Player memory uses both ROM and dynamic memory (RAM). ROM is used to store the Jam Player binary and the Jam File; dynamic memory is used when the Jam Player is called.

For information on how to estimate the maximum amount of RAM and ROM required by the Jam Player, see *Application Note 88 (Using the Jam Language for ISP & ICR via an Embedded Processor)*.

Porting the Jam Player

The Altera Jam Player (both Byte-Code and ASCII versions) works with a PC parallel port. To port the Jam Player to your processor, you only need to modify the **jamstub.c** or **jbistub.c** file (for the ASCII Jam Player or Jam Byte-Code Player, respectively). All other files should remain the same.

If the Jam Player is ported incorrectly, an `Unrecognized Device` error is generated. The most common causes for this error are listed below:

- After porting the Jam Player, the `TDO` value may be read in reversed polarity. This problem may occur because the default I/O code in the Jam Player assumes the use of the PC parallel port. Refer to the Jam Player **readme.txt** file on the ***Altera Digital Library CD-ROM*** for more detailed information on how to solve the problem.
- Although the `TMS` and `TDI` signals are clocked in on the rising edge of `TCK`, outputs do not change until the falling edge of `TCK`. This situation causes a half `TCK` clock cycle lag in reading out the values. If the `TDO` transition is expected on the rising edge, the data appears to be offset by one clock.
- Altera recommends using registers to synchronize the output transitions. In addition, some processor data ports use a register to synchronize the output signals. For example, reading and writing to the PC's parallel port is accomplished by reading and writing to registers. The use of these registers must be taken into consideration when reading and writing to the JTAG chain. Incorrect accounting of these registers can cause the values to either lead or lag the expected value.

You can use a test Jam File to determine if the Jam Player is ported correctly. Figure 1 shows a sample Jam File that helps debug potential porting problems, including the three issues discussed previously. You can download this example file from the literature page on Altera's web site at **http://www.altera.com**.

Figure 1. Sample Jam File for Debugging Porting Problems (Part 1 of 5)

```
NOTE JAM_VERSION "1.1 ";
NOTE DESIGN "IDCODE.jam version 1.4 4/28/98";
'################################################################################
'#This Jam File compares the IDCODE read from a JTAG chain with the
'#expected IDCODE. There are 5 parameters that can be set when executing
'#this code.
'#
'#COMP_IDCODE_[device #]=1, for example -dCOMP_IDCODE_9400=1
'#compares the IDCODE with an EPM9400 IDCODE.
'#PRE_IR=[IR_LENGTH] is the length of the instruction registers you want
'#to bypass after the target device. The default is 0, so if your
'#JTAG length is 1, you don't need to enter a value.
'#POST_IR=[IR_LENGTH] is the length of the instruction registers you
'#want to bypass before the target device. The default is 0, so if
'#your JTAG length is 1, you don't need to enter a value.
'#PRE_DR=[DR_LENGTH] is the length of the data registers you want
'#to bypass after the target device. The default is 0, so if your
'#JTAG length is 1, you don't need to enter a value.
'#POST_DR=[DR_LENGTH] is the length of the data registers you want
'#to bypass before the target device. The default is 0, so if your
'#JTAG length is 1, you don't need to enter a value.
'#Example: This example reads the IDCODE out of the second device in the
'#chain below:
'#
'#TDI -> EPM7128S -> EPM7064S -> EPM7256S -> EPM7256S -> TDO
'#
'#In this example, the IDCODE is compared to the EPM7064S IDCODE. If the JTAG
'#chain is set up properly, the IDCODEs should match.
'# C:\> jam -dCOMP_IDCODE_7064S=1 -dPRE_IR=20 -dPOST_IR=10 -dPRE_DR=2
'#-dPOST_DR=1 -p378 IDCODE.jam
'#
'#
'# Example: This example reads the IDCODE of a single device JTAG chain
'# and compares it to an EPM9480 IDCODE:
'#
'# C:\> jam -dCOMP_IDCODE_9480=1 -p378 IDCODE.jam
'################################################################################

'########################## Initialization ########################

BOOLEAN read_data[32];
BOOLEAN I_IDCODE[10] = BIN 1001101000;
BOOLEAN I_ONES[10] = BIN 1111111111;
BOOLEAN ONES_DATA[32]= HEX FFFFFFFF;
```

Figure 1. Sample Jam File for Debugging Porting Problems (Part 2 of 5)

```
BOOLEAN ID_9320[32]          = BIN 10111011000000000100110010010000;
BOOLEAN ID_9400[32]          = BIN 10111011000000000000001010010000;
BOOLEAN ID_9480[32]          = BIN 10111011000000000000100101001000;
BOOLEAN ID_9560[32]          = BIN 10111011000000000011010101001000;
BOOLEAN ID_7032S[32]         = BIN 10111011000001001100000011100000;
BOOLEAN ID_7064S[32]         = BIN 10111011000000100110000011100000;
BOOLEAN ID_7128S[32]         = BIN 10111011000000010100100011100000;
BOOLEAN ID_7128A[32]         = BIN 10111011000000010100100011100000;
BOOLEAN ID_7160S[32]         = BIN 10111011000000000110100011100000;
BOOLEAN ID_7192S[32]         = BIN 10111011000001001001100011100000;
BOOLEAN ID_7256S[32]         = BIN 10111011000001101010010011100000;
BOOLEAN ID_7256A[32]         = BIN 10111011000001101010010011100000;

BOOLEAN COMP_9320_IDCODE     = 0;
BOOLEAN COMP_9400_IDCODE     = 0;
BOOLEAN COMP_9480_IDCODE     = 0;
BOOLEAN COMP_9560_IDCODE     = 0;
BOOLEAN COMP_7032S_IDCODE    = 0;
BOOLEAN COMP_7064S_IDCODE    = 0;
BOOLEAN COMP_7096S_IDCODE    = 0;
BOOLEAN COMP_7128S_IDCODE    = 0;
BOOLEAN COMP_7128A_IDCODE    = 0;
BOOLEAN COMP_7160S_IDCODE    = 0;
BOOLEAN COMP_7192S_IDCODE    = 0;
BOOLEAN COMP_7256S_IDCODE    = 0;
BOOLEAN COMP_7256A_IDCODE    = 0;
BOOLEAN COMP_7032AE_IDCODE   = 0;
BOOLEAN COMP_7064AE_IDCODE   = 0;
BOOLEAN COMP_7128AE_IDCODE   = 0;
BOOLEAN COMP_7256AE_IDCODE   = 0;
BOOLEAN COMP_7512AE_IDCODE   = 0;
INTEGER PRE_IR        = 0;
INTEGER PRE_DR        = 0;
INTEGER POST_IR       = 0;
INTEGER POST_DR       = 0;

BOOLEAN SET_ID_EXPECTED[32];
BOOLEAN COMPARE_FLAG1  = 0;
BOOLEAN COMPARE_FLAG2  = 0;
BOOLEAN COMPARE_FLAG   = 0;

' This information is what is expected to be shifted out of the instruction
' register

BOOLEAN expected_data[10] = BIN 0101010101;
BOOLEAN ir_data[10];
```

Figure 1. Sample Jam File for Debugging Porting Problems (Part 3 of 5)

```
' These values default to 0, so if you have a single device JTAG chain, you do
' not have to set these values.

PREIR PRE_IR;
POSTIR POST_IR;
PREDR PRE_DR;
POSTDR POST_DR;

INTEGER i;

' ########################## Determine Action ########################

LET COMPARE_FLAG1= COMP_9320_IDCODE || COMP_9400_IDCODE || COMP_9480_IDCODE ||
  COMP_9560_IDCODE || COMP_7032S_IDCODE || COMP_7064S_IDCODE ||
  COMP_7096S_IDCODE || COMP_7032AE_IDCODE || COMP_7064AE_IDCODE ||
  COMP_7128AE_IDCODE;

LET COMPARE_FLAG2 = COMP_7128S_IDCODE || COMP_7128A_IDCODE || COMP_7160S_IDCODE
  || COMP_7192S_IDCODE || COMP_7256S_IDCODE || COMP_7256A_IDCODE ||
  COMP_7256AE_IDCODE || COMP_7512AE_IDCODE;

LET COMPARE_FLAG = COMPARE_FLAG1 || COMPARE_FLAG2;
IF COMPARE_FLAG != 1 THEN GOTO NO_OP;

FOR i=0 to 31;
IF COMP_9320_IDCODE    == 1 THEN LET SET_ID_EXPECTED[i] = ID_9320[i];
IF COMP_9400_IDCODE    == 1 THEN LET SET_ID_EXPECTED[i] = ID_9400[i];
IF COMP_9480_IDCODE    == 1 THEN LET SET_ID_EXPECTED[i] = ID_9480[i];
IF COMP_9560_IDCODE    == 1 THEN LET SET_ID_EXPECTED[i] = ID_9560[i];
IF COMP_7032S_IDCODE   == 1 THEN LET SET_ID_EXPECTED[i] = ID_7032S[i];
IF COMP_7064S_IDCODE   == 1 THEN LET SET_ID_EXPECTED[i] = ID_7064S[i];
IF COMP_7128S_IDCODE   == 1 THEN LET SET_ID_EXPECTED[i] = ID_7128S[i];
IF COMP_7128A_IDCODE   == 1 THEN LET SET_ID_EXPECTED[i] = ID_7128A[i];
IF COMP_7160S_IDCODE   == 1 THEN LET SET_ID_EXPECTED[i] = ID_7160S[i];
IF COMP_7192S_IDCODE   == 1 THEN LET SET_ID_EXPECTED[i] = ID_7192S[i];
IF COMP_7256S_IDCODE   == 1 THEN LET SET_ID_EXPECTED[i] = ID_7256S[i];
IF COMP_7256A_IDCODE   == 1 THEN LET SET_ID_EXPECTED[i] = ID_7256A[i];
IF COMP_7032AE_IDCODE  == 1 THEN LET SET_ID_EXPECTED[i] = ID_7032AE[i];
IF COMP_7064AE_IDCODE  == 1 THEN LET SET_ID_EXPECTED[i] = ID_7064AE[i];
IF COMP_7128AE_IDCODE  == 1 THEN LET SET_ID_EXPECTED[i] = ID_7128AE[i];
IF COMP_7256AE_IDCODE  == 1 THEN LET SET_ID_EXPECTED[i] = ID_7256AE[i];
IF COMP_7512AE_IDCODE  == 1 THEN LET SET_ID_EXPECTED[i] = ID_7512AE[i];


NEXT I;
```

Figure 1. Sample Jam File for Debugging Porting Problems (Part 4 of 5)

```
' ############################ Actual Loading ############################

IRSTOP IRPAUSE;
STATE RESET;
IRSCAN 10, I_IDCODE[0..9], CAPTURE ir_data[0..9];
STATE IDLE;

DRSCAN 32, ONES_DATA[0..31], CAPTURE read_data[0..31];

' ############################ Printing ############################

PRINT "EXPECTED IRSCAN : 1010101010";
PRINT "ACTUAL IRSCAN: ",ir_data[0], ir_data[1], ir_data[2], ir_data[3],
    ir_data[4], ir_data[5], ir_data[6], ir_data[7], ir_data[8], ir_data[9];

PRINT "";PRINT "EXPECTED IDCODE : ", SET_ID_EXPECTED[0], SET_ID_EXPECTED[1],
    SET_ID_EXPECTED[2], SET_ID_EXPECTED[3], SET_ID_EXPECTED[4],
    SET_ID_EXPECTED[5], SET_ID_EXPECTED[6], SET_ID_EXPECTED[7],
    SET_ID_EXPECTED[8], SET_ID_EXPECTED[9], SET_ID_EXPECTED[10],
    SET_ID_EXPECTED[11], SET_ID_EXPECTED[12], SET_ID_EXPECTED[13],
    SET_ID_EXPECTED[14], SET_ID_EXPECTED[15], SET_ID_EXPECTED[16],
    SET_ID_EXPECTED[17], SET_ID_EXPECTED[18], SET_ID_EXPECTED[19],
    SET_ID_EXPECTED[20], SET_ID_EXPECTED[21], SET_ID_EXPECTED[22],
    SET_ID_EXPECTED[23], SET_ID_EXPECTED[24], SET_ID_EXPECTED[25],
    SET_ID_EXPECTED[26], SET_ID_EXPECTED[27], SET_ID_EXPECTED[28],
    SET_ID_EXPECTED[29], SET_ID_EXPECTED[30], SET_ID_EXPECTED[31];

PRINT "ACTUAL IDCODE : ", READ_DATA[0], READ_DATA[1], READ_DATA[2],
    READ_DATA[3], READ_DATA[4], READ_DATA[5], READ_DATA[6], READ_DATA[7],
    READ_DATA[8], READ_DATA[9], READ_DATA[10], READ_DATA[11], READ_DATA[12],
    READ_DATA[13], READ_DATA[14], READ_DATA[15], READ_DATA[16], READ_DATA[17],
    READ_DATA[18], READ_DATA[19], READ_DATA[20], READ_DATA[21], READ_DATA[22],
    READ_DATA[23], READ_DATA[24], READ_DATA[25], READ_DATA[26], READ_DATA[27],
    READ_DATA[28], READ_DATA[29], READ_DATA[30], READ_DATA[31];

GOTO END;
```

Figure 1. Sample Jam File for Debugging Porting Problems (Part 5 of 5)

```
' ########################## If no parameters are set ########################

NO_OP: PRINT "jam [-d<var=val>] [-p<port>] [-s<port>] IDCODE.jam";
PRINT "-d : initialize variable to specified value";
PRINT "-p : parallel port number or address <for ByteBlaster>";
PRINT "-s : serial port name <for BitBlaster>";
PRINT " ";
PRINT "Example: To compare IDCODE of the 4th device in a chain of 5 Altera ";
PRINT "devices with EPM7192S IDCODE";
PRINT " ";
PRINT "jam -dCOMP_7192S_IDCODE=1 -dPRE_IR=10 -dPOST_IR=30 -dPRE_DR=1";
PRINT "dPOST_DR=3 -p378 IDCODE.jam";
PRINT " ";

END:

EXIT 0;
```

ISP via In-Circuit Testers

This section addresses specific issues associated with programming ISP-capable devices via in-circuit testers.

Using "F" vs. Non-"F" Devices

MAX devices use either fixed algorithms ("F") or branching algorithms (non-"F"). Most in-circuit tester file formats, e.g., SVF, Hewlett-Packard's Pattern Capture Format (**.pcf**), DTS, and ASC, are "fixed" or deterministic, which means they can only support one fixed algorithm without branching. The MAX+PLUS II software generates SVF Files for "F" devices. Because the algorithms in SVF Files are constant, you can always use these files to program future "F" devices.

Altera does not recommend programming non-"F" devices via in-circuit testers. Non-"F" devices require branching based on three variables read from the device: programming pulse time, erase pulse time, and manufacturer silicon ID. These three variables are programmed into all non-"F" Altera devices. Using only "F" devices eliminates problems you may experience if these variables change.

Maximum Vectors per File

The file formats for "bed of nails" in-circuit testers generally require very large vector files for in-system programming. When the file is larger than the tester's available memory, the file must be divided into smaller files. For example, Altera's **svf2pcf** utility automatically divides a single SVF File into several smaller files. In addition, the utility allows users to either specify the maximum number of vectors per file or use a default value. If you put too many vectors in a single file, an error message occurs. If you receive this error, simply reduce the number of vectors per file.

Pull-Up & Pull-Down Resistors

Testers may require pull-up or pull-down resistors on various signal traces. Contact the in-circuit tester manufacturer directly for specific information.

Summary

The information provided in this document is based on development experiences and customer issues resolved by Altera. For more information on resolving in-system programming problems, contact Altera Applications at (800) 800-3753, or via e-mail at **sos@altera.com**. Twenty-four hour support is available through the Atlas section of the Altera web site at **http://www.altera.com**.

Notes:

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 544-7104
Literature Services:
(888) 3-ALTERA
lit_req@altera.com