

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® MAX® 9000 (including MAX 9000A), MAX 7000S, and MAX 7000A devices can be programmed and reprogrammed using ISP 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
- 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 V_{CCINT} pins 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 V_{CCINT} pins using an oscilloscope. Set the oscilloscope's trigger level to 4.75 V and 5.25 V. Then, measure the voltage between V_{CCINT} and ground, probed at the pins of the device. If the oscilloscope is triggered, 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 V) and ($V_{CCINT} + 1$ 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.



Go to the [Operating Requirements for Altera Devices Data Sheet](#) in the **1998 Data Book** for more information.

Interrupting In-System Programming

Altera does not recommend interrupting the programming process because partially programmed devices have unpredictable operation. Partially programmed devices also cause signal conflicts, which can lead to permanent device damage.

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.

Programming via a Download Cable

If you are using the BitBlaster™, ByteBlaster™, or ByteBlasterMV™ 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 the BitBlaster, ByteBlaster, or ByteBlasterMV download cables, go to the [BitBlaster Serial Download Cable Data Sheet](#), [ByteBlaster Parallel Port Download Cable Data Sheet](#), or [ByteBlasterMV Parallel Port Download Cable Data Sheet](#).

Disabling IEEE Std. 1149.1 Circuitry

If your design does not use ISP or 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.

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

Note:

(1) Information on MAX 7000A devices is preliminary.

JTAG Permanently Disabled (MAX 7000S & MAX 7000A Devices)

MAX 7000S and MAX 7000A 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)

JTAG is always enabled in MAX 9000 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 before you pull TCK 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 MAX 7000S, MAX 7000A, MAX 9000, and MAX 9000A devices that use JTAG for either in-system programming or boundary-scan testing, JTAG must be enabled during ISP and BST but disabled at all other times. You control JTAG operation through the JTAG pins. Like permanently disabling MAX 9000 devices, either pull TMS high and TCK low, or pull TMS high before pulling TCK high.

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 4](#).

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 (i.e., a full 3.3-V device powered by V_{CCIO}), the next device's 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-Styled during In-System Programming

All device I/O pins are tri-stated during in-system programming. In addition, MAX 7000S and MAX 7000A devices have a weak pull-up resistor. The value of this pull-up resistor is listed in the [MAX 7000 Programmable Logic Device Family Data Sheet](#) and the [MAX 7000A Programmable Logic Device Family Data Sheet](#), respectively.

You must add sufficient pull-up or pull-down resistors on signals requiring 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 will 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.

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 Unrecognized Device error is generated. Typical causes for this error are shown below:

- Incorrectly connected download cable
- TDO is not connected
- Incomplete JTAG chain
- Noisy TCK signal
- Jam™ Player ported incorrectly

Incorrectly Connected Download Cable

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 BitBlaster, ByteBlaster, or ByteBlasterMV download cable, go to the [BitBlaster Serial Download Cable Data Sheet](#), [ByteBlaster Parallel Port Download Cable Data Sheet](#), or [ByteBlasterMV Parallel Port Download Cable Data Sheet](#).

TDO Is Not Connected

You will receive an error if the TDO port is not connected. The TDO port must be connected for the device(s) to program successfully.

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 watch 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.

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 through a Jam File (**.jam**) or Serial Vector Format File (**.svf**). You can use a Jam File or SVF 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](#) shows a sample Jam File for reading the IDCODE.

Figure 1. Sample Jam File for Reading the IDCODE

```
'This example reads the IDCODE for the second device in the chain below:
'TDI -> EPM7128S -> EPM7064S -> EPM7256S -> EPM7256S -> TDO

'The device IDCODE can be found in AN 39: IEEE 1149.1 (JTAG) Boundary-Scan Testing
'in Altera Devices

BOOLEAN expected_data[32] = BIN 10111011000000100110000011100000;

'EPM7064S IDCODE

PREIR 20;           ' This command bypasses the 2 devices after EPM7064S
POSTIR 10;         ' This command bypasses the 1st device
PREDR 2;           ' This command bypasses the 2 devices after EPM7064S
POSTDR 1;         ' This command bypasses the 1st device
STATE RESET;
IRSCAN 10, 057;
DRSCAN 32, FFFFFFFF COMPARE expected_data[0..31], FFFFFFFF, error;
IF error==0 then PRINT "IDCODE read successfully";
IF error==1 then PRINT "IDCODE not read successfully";
```

Figure 2 shows a sample SVF File for reading the IDCODE. When a device is part of a larger chain, you must add appropriate header and trailer information to the file.

Figure 2. Sample SVF File for Reading the IDCODE

```
! If the device is in a chain of devices, HIR, HDR, TIR, and TDR
! statements must be added here.

STATE RESET;
SIR 10 TDI (057);
SDR 32 TDI (FFFFFFF) TDO (<add appropriate IDCODE here>) MASK (FFFFFFF);

! If the IDCODE is wrong, the previous line gives compare errors when executed
```

These sample files can be extended to implement more error checking when the IDCODE instruction is read out of the device. Contact the ISP Support Program at ISPembed@altera.com or ISPATE@altera.com for extended versions of these files. In addition, the `readme.txt` file that accompanies the Jam Player source code provides further information on porting the Jam Player software.

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 Bytecode Jam 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, go to [Application Note 88 \(Using the Jam Language for ISP via an Embedded Processor\)](#).

Porting the Jam Player

The Altera Jam Player (both Bytecode 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 Bytecode Jam Player, respectively). All other files should stay 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 attaching the ByteBlaster or ByteBlasterMV download cable on your processor, the TDO value may be read in reversed polarity. This problem occurs if the parallel port registers are read in inverted polarity. Refer to the `readme.txt` file on the ***In-System Programmability 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's recommended implementation uses 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 3 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 world-wide web site at <http://www.altera.com>.

Figure 3. Sample Jam File For Debugging Porting Problems (Part 1 of 4)

```
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
#####
```

Figure 3. Sample Jam File For Debugging Porting Problems (Part 2 of 4)

```

##### Initialization #####

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

BOOLEAN ID_9320[32]    = BIN 10111011000000000100110010010000;
BOOLEAN ID_9400[32]    = BIN 101110110000000000000001010010000;
BOOLEAN ID_9480[32]    = BIN 101110110000000000001001010010000;
BOOLEAN ID_9560[32]    = BIN 10111011000000000110101010010000;
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_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;

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 is what is expected to be shifted out of the instruction register
BOOLEAN expected_data[10] = BIN 0101010101;

```

Figure 3. Sample Jam File For Debugging Porting Problems (Part 3 of 4)

```

BOOLEAN ir_data[10];
' 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;

LET COMPARE_FLAG2 = COMP_7128S_IDCODE || COMP_7128A_IDCODE || COMP_7160S_IDCODE
    || COMP_7192S_IDCODE || COMP_7256S_IDCODE || COMP_7256A_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];

NEXT I;

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

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

```

Figure 3. Sample Jam File For Debugging Porting Problems (Part 4 of 4)

```

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;

' ##### 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 -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 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 version 8.2 and higher generates SVF Files for “F” devices. Because the algorithms in SVF Files are constant, you can always use these SVF 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 Altera non-“F” 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.

HP 3070 Tester

Altera’s **svf2pcf** utility automatically divides an HP PCF file into multiple small files so the HP tester’s memory can support the number of vectors required for in-system programming.

A small delay occurs when the tester applies each vector file to the device. During this interval, the HP tester tri-states all pins, briefly drives all pins low, and then drives the first vector in the subsequent PCF file.

To avoid unintentional TCK transitions during this interval, you should follow the two steps shown below:

- Each PCF vector file must start and end with all signals low. Altera's **svf2pcf** utility performs this function automatically.
- You must add a pull-down resistor to the TCK signal. This resistor should be designed into the board to comply with the recommendation outlined in "JTAG Enabled for ISP/BST & Disabled in User Mode" on page 5. If the resistor is not designed into the board, it can be added to the test fixture. The value of this pull-down resistor is dependent on the number and load of each device in the chain. A good rule of thumb is 1 k Ω to 5 k Ω if the pull-down resistor is located on the board and 500 Ω to 1 k Ω if it is located in the test fixture.

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 the ISP Support Program at **ISPembed@altera.com** or **ISPATE@altera.com**. You can receive information on the Altera ISP Support Program by visiting the Altera web site at **<http://www.altera.com>**.

Revision History

The information contained in *Application Note 100 (In-System Programmability Guidelines)* version 1.01 supersedes information published in *Application Note 100 (In-System Programmability Guidelines)* version 1.0.

Application Note 100 (In-System Programmability Guidelines) version 1.01 contains corrected information on how to disable the IEEE Std. 1149.1 circuitry in **Table 1 on page 4**.



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

Altera, MAX, MAX+PLUS, MAX+PLUS II, MAX 9000, MAX 9000A, MAX 7000S, MAX 7000A, BitBlaster, ByteBlaster, ByteBlasterMV, EPM7128S, EPM7256S, EPM7064S, EPM9400, EPM9480, and Jam are trademarks and/or service marks of Altera Corporation in the United States and other countries. Altera acknowledges the trademarks of other organizations for their respective products or services mentioned in this document. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

Copyright © 1998 Altera Corporation. All rights reserved.



I.S. EN ISO 9001

