In-System Programmability Guidelines

2014.09.22

AN-100 Subscribe Send Feedback

In-system programming (ISP) allows you to program ISP-capable Altera® devices through the IEEE Std. 1149.1 JTAG interface. This interface allows you to program devices and functionally test the PCB in a single manufacturing step, saving testing time and assembly costs.

As time-to-market pressure increases, design engineers require advanced system-level products to ensure problem-free development and manufacturing. Programmable logic devices (PLDs) with ISP can help accelerate development time, facilitate in-field upgrades, simplify the manufacturing flow, lower inventory costs, and improve PCB testing capabilities.

General ISP Guideline

The are several guidelines that will help you to design successfully for ISP-capable devices. You must use these guidelines regardless of your specific design implementation.

The following lists the guidelines:

- Operating Conditions
- User Flash Memory Operations During In-System Programming
- Interrupting In-System Programming
- MultiVolt Devices and Power-Up Sequences
- I/O Pins Tri-Stated During In-System Programming
- Pull-Up and Pull-Down of JTAG Pins During In-System Programming

Operating Conditions

Each Altera device has several parametric ratings (operating conditions) required for proper operation.

When in user mode, Altera devices can exceed these conditions and still operate correctly; however, Altera device must not exceed these conditions during in-system programming. Violating any of the operating conditions during in-system programming can result in programming failures or incorrectly programmed devices.

Table 1: Power-up Requirements for ISP Functions for Altera devices

Device	Power-up Requirements
Single supply MAX® 10 devices	• V_{CC_ONE} of the device • V_{CCA} of the device • V_{CCIO} for all I/O banks

Device	Power-up Requirements
Dual supply MAX 10 devices	• V_{CC} of the device • V_{CCA} of the device • V_{CCIO} for all I/O banks
All supported devices except MAX 10	• V_{CCINT} of the device • V_{CCIO} of all I/O banks

ISP Voltage

Altera devices have different ISP voltage requirements depending on the device family. You have to be comply all requirements to ensure the device are programmed correctly.

MAX 10 Devices

For MAX 10 devices, you must maintain the V_{CC}/V_{CC_ONE} level, V_{CCA} level, and V_{CCIO} level on the V_{CC}/V_{CC_ONE}, V_{CCA}, and V_{CCIO} pins during in-system programming to ensure the flash cells of the device are correctly programmed. The V_{CC}/V_{CC_ONE}, V_{CCA} and V_{CCIO} specification applies for both commercial- and industrial-temperature grade devices.

MAX II and V Devices

For MAX II and MAX V devices, you must maintain the V_{CCINT} level and V_{CCIO} level on the V_{CCINT} and V_{CCIO} pins during in-system programming to ensure the flash cells of the device are correctly programmed. The V_{CCINT} and V_{CCIO} specification applies for both commercial- and industrial-temperature grade devices.

MAX 3000, MAX 7000, and MAX 9000 Devices

MAX 3000, MAX 7000, and MAX 9000 devices have specified ISP voltage known as V_{CCISP}. You must maintain the V_{CCISP} level on the V_{CCINT} pins (for example, V_{CCINT} equals to V_{CCISP}) during ISP to ensure that the EEPROM cells of the device are programmed correctly. The V_{CCISP} specification applies for both commercial- and industrial- temperature- grade devices.

You have to adjust your in-system programming setup to maintain correct voltage levels if power consumption during ISP exceeds the power consumption when in user mode. Altera recommends that you test the V_{CCISP} levels on the device's V_{CCINT} pins with an oscilloscope. First, test the V_{CCISP} levels with the oscilloscope's trigger level set to the recommended minimum V_{CC} level. Measure the voltage between V_{CCINT} and ground, probed at the pins of the device. Then, repeat this test with the oscilloscope's trigger level set to the recommended maximum V_{CC} level. If the oscilloscope is triggered at either voltage level, you must adjust your programming setup.

The recommended voltage levels are specified in the Recommended Operating Conditions section of the appropriate Device Family Datasheet. Refer to the related information.

Related Information

- **MAX 10 Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 10 Devices.
- **MAX V Device Datasheet**
  Provides more information about the recommended operating conditions for MAX V Devices.
- **MAX II Device Datasheet**
  Provides more information about the recommended operating conditions for MAX II Devices.

- **MAX 3000A Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 3000A Devices.
- **MAX 7000 Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 7000 Devices.
- **MAX 7000A Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 7000A Devices.
- **MAX 7000B Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 7000B Devices.
- **MAX 9000 Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 9000 Devices.

Input Voltages

Every Altera device family has a range for safe device operation. You need to ensure that 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. Pins that have an overshoot greater than 1.0 V must have series termination.

Each Device Family Datasheet lists the device input voltage specification in the “Absolute Maximum Ratings” and the “Recommended Operating Conditions” tables. The input voltages in the “Absolute Maximum Rating” table refers to the maximum voltage that the device can tolerate before risking permanent damage.

Related Information

- **MAX 10 Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 10 Devices.
- **MAX V Device Datasheet**
  Provides more information about the recommended operating conditions for MAX V Devices.
- **MAX II Device Datasheet**
  Provides more information about the recommended operating conditions for MAX II Devices.
- **MAX 3000A Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 3000A Devices.
- **MAX 7000 Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 7000 Devices.
- **MAX 7000A Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 7000A Devices.
- **MAX 7000B Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 7000B Devices.
- **MAX 9000 Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 9000 Devices.

User Flash Memory Operations During In-System Programming

If your design allows you to erase or write the MAX II, MAX V or MAX 10 device’s user flash memory (UFM), you must ensure that all the erase or write operations of the UFM are completed before starting any ISP session (including stand-alone verify, examine, setting security bit, and reading the contents of the UFM). If the UFM is performing any erase or write operation, you must not start an ISP session as this may put the device in an unrecoverable state. However, this restriction does not apply to the read operation of the UFM.

If you cannot ensure that any erase or write operation of the UFM is complete before attempting an ISP operation to the MAX II, MAX V or MAX 10 device, you must enable the real-time ISP feature. If used properly, this feature can help guard against any UFM or ISP operation contention. If you enable real-time ISP feature, the programming algorithm from the Quartus® II software, or Jam™ Standard Test and Programming Language (STAPL) Format (.jam) file/Jam Byte-Code (.jbc) file waits for 500 ms before it begins any operation, the same amount of time it takes to erase one UFM sector (that is, the real-time ISP programming algorithm waits for a previously started UFM erase operation to complete).

However, if you are using a real-time ISP feature, no other UFM operations are allowed after that time (no address shifting, no data shifting, and no read, write, or erase operations). You can control the UFM operations by monitoring the `RTP_BUSY` signal on the ALTUFM_NONE megafunction. If you are performing a real-time ISP operation, the `RTP_BUSY` output signal on the UFM block goes high. You can monitor the `RTP_BUSY` signal and ensure that all UFM operations from the logic array cease until real-time ISP is complete. This user-generated control logic is only necessary for the ALTUFM_NONE megafunction, which provides no auto-generated logic. The other parameter editors for the ALTUFM megafunction (ALTUFM_PARALLEL, ALTUFM_SPI, and ALTUFM_I2C) contain control logic that automatically monitors the `RTP_BUSY` signal and ceases UFM operations if you are performing the real-time ISP operation.

Related Information

AN 630: Real-Time ISP and ISP Clamp for Altera Devices
Provides more information about the Real-time ISP and ISP Clamp for MAX II, MAX V and MAX 10 Devices.

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.

MAX II, MAX V and MAX 10 devices have an `ISP_DONE` bit that you can only set at the end of a successful program sequence. The I/O pins can only drive out if this bit is set. This prevents a partially programmed device from driving out and operating unpredictably.

MultiVolt Devices and 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. If you do not hold the JTAG pins in the test-logic-reset state, in-system programming errors can occur.

Therefore, in systems with multiple power supply voltages, you must hold the JTAG pins in the test-logic-reset state until all devices in the chain are fully powered up. This procedure is important because systems that have multiple power supplies cannot power all voltage levels simultaneously.

MAX devices have the MultiVolt™ feature and can use more than one power supply voltage:

- MAX 10 devices—V_{CC}/V_{CC_ONE} provides power to core and periphery, V_{CCIO} provides power to input pins and output buffer, and V_{CCA} provides power to PLL regulator.
- Other MAX devices—V_{CCINT} and V_{CCIO} for each I/O bank. V_{CCINT} provides power to the JTAG circuitry; V_{CCIO} provides power to input pins and output drivers for output pins, including `TDO` pin.

V_{CCINT} Powered before V_{CCIO}

If V_{CCINT} is powered up before V_{CCIO}, the JTAG circuitry is active but unable to drive out signals. Thus, any transition on the TCK pin can cause the state machine to transition to an unknown JTAG state. If TMS and TCK pins 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, you must disable all JTAG signals.

V_{CCIO} Powered before V_{CCINT}

If V_{CCIO} is powered up before V_{CCINT}, the JTAG circuitry is not active but the TDO pin is tri-stated. Although the JTAG circuitry is not active and 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, you must disable these signals for all devices in the chain. Therefore, you must disable the JTAG pins by pulling the TCK signal low and the TMS signal high. With the MAX 10 device hot-socketing feature, you no longer need to ensure a proper power up sequence for each device on the board.

Related Information

Disabling IEEE Std. 1149.1 Circuitry on page 7

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

By default, all device I/O pins are tri-stated during in-system programming. In addition, MAX devices provide a weak pull-up resistor during ISP. The purpose of this weak pull-up resistor is to eliminate the requirement for external pull-up resistors on tri-stated I/O pins.

You must add sufficient pull-up or pull-down resistors on signals that require a particular value during in-system programming (for example, JTAG configuration 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.

For MAX II, MAX V and MAX 10 devices, you can use the in-system programming clamp feature or the real-time ISP feature to ensure that each I/O pin is clamped to a specific state during in-system programming.

Related Information

AN 630: Real-Time ISP and ISP Clamp for Altera Devices
Provides more information about the Real-time ISP and ISP Clamp for MAX II, MAX V and MAX 10 Devices.

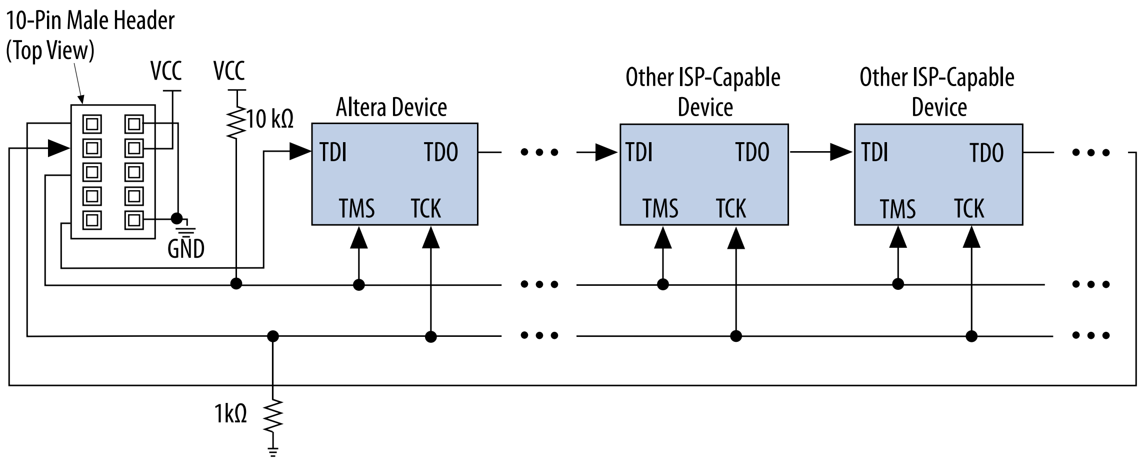
Pull-Up and Pull-Down of JTAG Pins During In-System Programming

An Altera device operating in in-system programming mode require four pins: TDI, TDO, TMS, and TCK. Three of the four JTAG pins have internal weak pull-up or pull-down resistors.

The TDI and TMS pins have internal weak pull-up resistors, while the TCK pin has an internal weak pull-down resistor. However, for device programming in a JTAG chain, there might be devices that do not have internal pull-up or pull-down resistors. Altera recommends to pull the TMS signal high externally through 10-kΩ and the TCK signal low through 1-kΩ resistors. Pulling up the TDI signal externally for the device is optional.

Figure 1: External Pull-Up and Pull-Down Resistors for TMS and TCK of a JTAG Chain in Altera Devices

Figure shows the external pull-up and pull-down for the `TMS` and `TCK` pins of the JTAG chain in Altera devices. The `TDO` pin does not have internal pull-up or pull-down resistors, and does not require external pull-up or pull-down resistors.



The `TMS` signal is pulled high so that the test access port (TAP) controller remains in the `TEST_LOGIC` or `RESET` state even if there is input from `TCK` signal. During power up, you must pull the `TCK` signal low to prevent the `TCK` signal from pulsing high. Pulling the `TCK` signal high is not recommended because the increase in power supply to the pull-up resistor causes the `TCK` signal to pulse high; therefore, it is possible for the TAP controller to reach an unintended state.

Related Information

- **MAX 10 JTAG Boundary-Scan Testing User Guide**
  Provides more information about IEEE 1149.1 circuitry and JTAG pins function for MAX 10 Devices.
- **JTAG Boundary-Scan Testing in MAX V Devices**
  Provides more information about IEEE 1149.1 circuitry and JTAG pins function for MAX V Devices.
- **IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices**
  Provides more information about IEEE 1149.1 circuitry and JTAG pins function for MAX II Devices.
- **IEEE 1149.1 JTAG Boundary-Scan Testing in Altera Devices**
  Provides more information about IEEE 1149.1 circuitry and JTAG pins function for MAX 3000, MAX 7000 and MAX 9000 Devices.

IEEE Std. 1149.1 Signals

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

TCK Signal

A noisy `TCK` signal causes most in-system programming failures. Noisy transitions on rising or falling edges can cause incorrect clocking of the IEEE Std. 1149.1 TAP controller. Incorrect clocking can cause the state machine to transition to an unknown state, leading to in-system programming failures.

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 the internal weak pull-down resistor or an external 1-kΩ resistor.

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. You must not use resistor and capacitor networks to slow down edge rates, because resistor and capacitor networks can violate the input specifications of the device. Use a driver chip to prevent the edge rate from being too slow. Altera recommends using driver chips that do not glitch after power up.

Programming Through a Download Cable

You can program Altera devices with a MasterBlaster™, ByteBlasterMV™, ByteBlaster™ II, ByteBlaster, BitBlaster™, EthernetBlaster, or USB Blaster download cable. Using a PC or UNIX workstation with the Quartus II Programmer, you can download Programmer Object File (**.pof**), **.jam**, or **.jbc** files to the Altera devices through the download cable.

If you use the download cables and your JTAG chain contains three or more devices, Altera recommends adding a buffer to the chain. You must select a buffer with slow transitions to minimize noise, but ensure that the transition rates can still meet `TCK` performance requirements of your chain.

If you must extend the download cable, you can attach a standard PC parallel or USB 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.

Note: Different download cables have different programming times. Refer related information.

Related Information

- MasterBlaster Serial/USB Communications Cable User Guide
- ByteBlasterMV Download Cable User Guide
- ByteBlaster II Download Cable User Guide
- BitBlaster Serial Download Cable User Guide
- USB-Blaster Download Cable User Guide
- EthernetBlaster II Communications Cable User Guide

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 2: Disabling IEEE Std.1149.1 Circuitry in MAX Devices

Device	Permanently Disabled	Enabled for ISP and BST, Disabled During User Mode
Max 10	• Pull the `TMS` and `TDI` signal high • Pull the `TCK` signal low	

Device	Permanently Disabled	Enabled for ISP and BST, Disabled During User Mode
MAX II	Either: • Pull the `TMS` signal high and the `TCK` signal low or • Pull the `TMS` signal high before pulling the `TCK` signal high	
MAX V		
MAX 9000		
MAX 9000A		
MAX 7000S	Turn off the **Enable JTAG BST Support** option in the Quartus II software.	Either: • Pull the `TMS` signal high and the `TCK` signal low or • Pull the `TMS` signal high before pulling the `TCK` signal high
MAX 7000B		
MAX 7000A		
MAX 7000AE		
MAX 3000A		

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

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

JTAG Permanently Disabled (MAX 10, MAX V, MAX II, MAX 9000 and MAX 9000A Devices)

By default, the JTAG circuitry is always enabled in MAX 10, MAX V, MAX II, MAX 9000, and MAX 9000A devices after power-up. You must enable the JTAG circuitry during ISP and boundary-scan testing, but must be disabled at all times. 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 the `TMS` signal high but does not explain what to do with the `TCK` signal. Altera recommends pulling the `TMS` signal high and the `TCK` signal low. Pulling the `TCK` signal low ensures that a rising edge does not occur on the `TCK` signal during the power-up sequence.

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

JTAG Enabled for ISP or BST and Disabled in User Mode

For Altera ISP-capable devices that use JTAG for either in-system programming or boundary-scan testing, you must enable the JTAG circuitry during ISP and BST but must be disabled at all other times. You control JTAG operation through the JTAG pins. To disable the JTAG circuitry on MAX 10, MAX V, MAX II, MAX 9000, and MAX 9000A devices permanently, either pull the TMS signal high and the TCK signal low, or pull the TMS signal high before pulling the TCK signal high.

Related Information

- MultiVolt Devices and Power-Up Sequences on page 4

- **MAX 10 JTAG Boundary-Scan Testing User Guide**
  Provides more information about IEEE 1149.1 circuitry and JTAG pins function for MAX 10 Devices.
- **JTAG Boundary-Scan Testing in MAX V Devices**
  Provides more information about IEEE 1149.1 circuitry and JTAG pins function for MAX V Devices.
- **IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices**
  Provides more information about IEEE 1149.1 circuitry and JTAG pins function for MAX II Devices.
- **IEEE 1149.1 JTAG Boundary-Scan Testing in Altera Devices**
  Provides more information about IEEE 1149.1 circuitry and JTAG pins function for MAX 3000, MAX 7000 and MAX 9000 Devices.

Working with Different Voltage Levels

When devices in a JTAG chain operate at different voltage levels, an output voltage specification of a device must meet the input voltage specification of the subsequent device.

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 output voltage of the device to meet the 2.5-V input voltage specification of the subsequent device.

Because all devices in a JTAG chain are tied together, you must also ensure that the `TDO` output of the first device meets the `TDI` input voltage specification of the subsequent device 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. You can set all MultiVolt devices for 3.3-V, 2.5-V, 1.8-V, or 1.5-V I/O operation. The JTAG pins of MultiVolt devices support these voltage levels.

Related Information

- **MAX 10 Device Datasheet**
  Provides more information about the I/O standard compatibility for each voltages for MAX 10 Devices.
- **MAX V Architecture**
  Provides more information about the I/O standard compatibility for each voltages for MAX V Devices.
- **MAX II Architecture**
  Provides more information about the I/O standard compatibility for each voltages for MAX II Devices.
- **MAX 3000A Device Datasheet**
  Provides more information about the I/O standard compatibility for each voltages for MAX 3000A Devices.
- **MAX 7000 Device Datasheet**
  Provides more information about the I/O standard compatibility for each voltages for MAX 7000 Devices.
- **MAX 7000A Device Datasheet**
  Provides more information about the I/O standard compatibility for each voltages for MAX 7000A Devices.
- **MAX 7000B Device Datasheet**
  Provides more information about the I/O standard compatibility for each voltages for MAX 7000B Devices.

- **MAX 9000 Device Datasheet**
  Provides more information about the I/O standard compatibility for each voltages for MAX 9000 Devices.

Sequential versus Concurrent Programming

This section describes how to program multiple devices with sequential and concurrent programming. The JTAG chain setup for sequential and concurrent programming is similar and only the programming algorithms are different.

Sequential Programming

Sequential programming is the process of programming multiple devices in a chain, one device at a time. After the process of programming the first device in the chain is complete, the next device is programmed. This sequence continues until all specified devices in the JTAG chain are programmed. After a device is successfully programmed, the device is in bypass mode that allows passing of data to the subsequent devices in the chain. The devices in the chain do not go into user mode until all the devices are programmed.

Concurrent Programming

Use concurrent programming to program devices from the same device family in parallel. The programming time is longer than the time required to program the largest device in the chain, resulting in considerably faster programming times than sequential programming (where programming time is equal to the sum of individual programming times for all devices). Higher clock rates for shifting data result in even greater time savings.

To perform concurrent programming of devices with Serial Vector Format File (**.svf**), **.jam** files, or **.jbc** files created from the Quartus II software, follow these steps:

1. On the Tools menu, click **Programmer**.
2. Click **Add File** and select programming files for the respective devices.
3. On the **File** menu, point to **Create/Update** and click **Create JAM, SVF, or ISC File**.
4. Specify a file in the **File format** list.
5. Click **OK**.

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 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** if you create individual files for each device. In this scheme, FLEX and APEX devices do 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 specific 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 Troubleshooting Guidelines

This section provides tips for troubleshooting ISP-related problems.

Invalid ID and Unrecognized Device Messages

The first step after the device enters ISP mode is to check the silicon ID or the JTAD ID of the device. If the silicon ID or the JTAD ID does not match, an `Invalid ID` or `Unrecognized Device` error is generated.

The following section describes the typical causes for this error:

- Download Cable Connected Incorrectly
- TDO Is Not Connected
- Incomplete JTAG Chain
- Noisy TCK Signal
- Jam Player Ported Incorrectly

Download Cable Connected Incorrectly

An error is generated if the download cable is connected incorrectly to the parallel or USB port, or if the download cable is not receiving power from your board.

TDO Is Not Connected

An error is generated 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, you must connect the `TDO` port of each device to the `TDI` port of subsequent device, and you must connect the `TDO` port of the last device to the `TDO` port of the download cable.

Incomplete JTAG Chain

An error is generated 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 `TDO` port of each device 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

An error is generated if the Jam Player is not ported correctly for your platform. To check if the Jam Player is causing the error, apply the `IDCODE` instruction to the target device with a **.jam** file. You can use the **.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.

Related Information

- **TCK Signal** on page 6
  Provides more information about noisy TCK signal.
- **IDCODE Reader Jam File**

Verify the JTAG Chain Continuity

For in-system programming to occur successfully, the number of devices in the JTAG chain must match the number reported in the Quartus II software or the MAX+Plus II software.

To verify in the Quartus II software that the JTAG chain is connected properly, follow these steps:

1. Open the **Programmer** in the Quartus II software.
2. Click **Auto Detect** in the Programmer. The Quartus II software reports the number of devices found on the JTAG chain. If this fails, check the JTAG chain to make sure it is not broken.

To verify in the MAX+Plus II software that the JTAG chain is connected properly, follow these steps:

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 the amount of devices found on the JTAG chain.

Check the Voltage Levels of the Board During In-System Programming

Monitor the V_{CCINT} or V_{CCA}, V_{CC_ONE}/V_{CC} and V_{CCIO} signal for all banks on your JTAG chain with an oscilloscope.

Set the trigger to the minimum level listed in the “Recommended Operating Conditions” table of the appropriate Device Family Datasheet. If a trigger occurs during in-system programming, the devices may need more current than is being supplied by the existing power supply. You can replace the existing power supply with one that provides more current.

Related Information

- **MAX 10 Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 10 Devices.
- **MAX V Device Datasheet**
  Provides more information about the recommended operating conditions for MAX V Devices.
- **MAX II Device Datasheet**
  Provides more information about the recommended operating conditions for MAX II Devices.

- **MAX 3000A Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 3000A Devices.
- **MAX 7000 Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 7000 Devices.
- **MAX 7000A Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 7000A Devices.
- **MAX 7000B Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 7000B Devices.
- **MAX 9000 Device Datasheet**
  Provides more information about the recommended operating conditions for MAX 9000 Devices.

Random Signals on JTAG Pins

During normal operation, TAP controller of each device 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` signal six times. If the device successfully powers up, you must add a higher pull-down resistor on the `TCK` signal.

Software Issues

Failures during in-system programming may occasionally be related to the Quartus II software or the MAX+Plus II software.

Software-related issues are documented in the Knowledge Center section under the Support Center on the Altera website. Search the database for information relating to software issues that interfere with in-system programming.

Related Information

Knowledge Center

ISP through Embedded Processors

This section provides guidelines for programming ISP-capable devices with the Jam STAPL and an embedded processor.

Processor and 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.

Related Information

AN 425: Using the Command-Line Jam STAPL Solution for Device Programming
Provides more information about how to estimate the maximum amount of RAM and ROM required by the Jam Player.

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

- 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.
- Although the `TMS` and `TDI` signals are clocked in on the rising edge of `TCK` signal, outputs do not change until the falling edge of `TCK` signal. This situation causes a half `TCK` clock cycle to 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 parallel port of the PC is accomplished by reading and writing to registers. When reading and writing to the JTAG chain, you must take into consideration the use of these registers. 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. The following examples show parts of a sample .jam file that helps debug potential porting problems.

Example 1: Sample 1

```
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
'############################################################################
#
```

Example 2: Sample 2

```
' ######################### 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 10111011000000000000001010010000;
BOOLEAN ID_9480[32] = BIN 10111011000000000001001010010000;
BOOLEAN ID_9560[32] = BIN 10111011000000000110101010010000;
BOOLEAN ID_7032S[32] = BIN 10111011000001001100000011100000;
BOOLEAN ID_7064S[32] = BIN 10111011000001001100000011100000;
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];
```

Example 3: Sample 3

```
' 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];
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;
```

Example 4: Sample 4

```
' ######################### 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;
' ######################### 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 through In-Circuit Testers

This section addresses specific issues associated with programming ISP-capable devices through 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—for example, .svf, 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 or JTAG 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 available memory in the tester, you must divide the file 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 and 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.

Related Information

AN 425: Using the Command-Line Jam STAPL Solution for Device Programming
Provides more information about using Agilent's 3070 in-circuit tester to in-system program MAX II and MAX V devices.

Document Revision History

Date	Version	Changes
September 2014	2014.09.22	• Converted document to the new template. • Updated document to include information about MAX 10 devices.
December 2010	4.0	• Converted document to the new template. • Updated document to include information about MAX II and MAX V devices.