Using the Agilent 3070 Tester for In-System Programming in Altera CPLDs

AN-628-1.0 Application Note

This application note describes how to use the Agilent 3070 test system to achieve faster programming times for Altera® MAX® II and MAX V devices. This application note also describes the development flow for the Agilent 3070 tester with and without the programmable logic device (PLD) in-system programmability (ISP) software by Agilent Technologies.

ISP is a mainstream feature in PLDs, offering system designers and test engineers significant cost benefits by integrating PLD programming into board-level testing. These benefits include reduced inventory of pre-programmed devices, lower costs, fewer devices damaged by handling, and increased flexibility in engineering changes.

Altera provides complete solutions for programming MAX II and MAX V devices using the Agilent 3070 test system together with other ISP-capable devices.

When programming MAX II and MAX V devices together with devices from other device families using the Agilent 3070 tester, ensure that you can program all the devices in the chain with the Agilent 3070 tester.

This chapter contains the following sections:

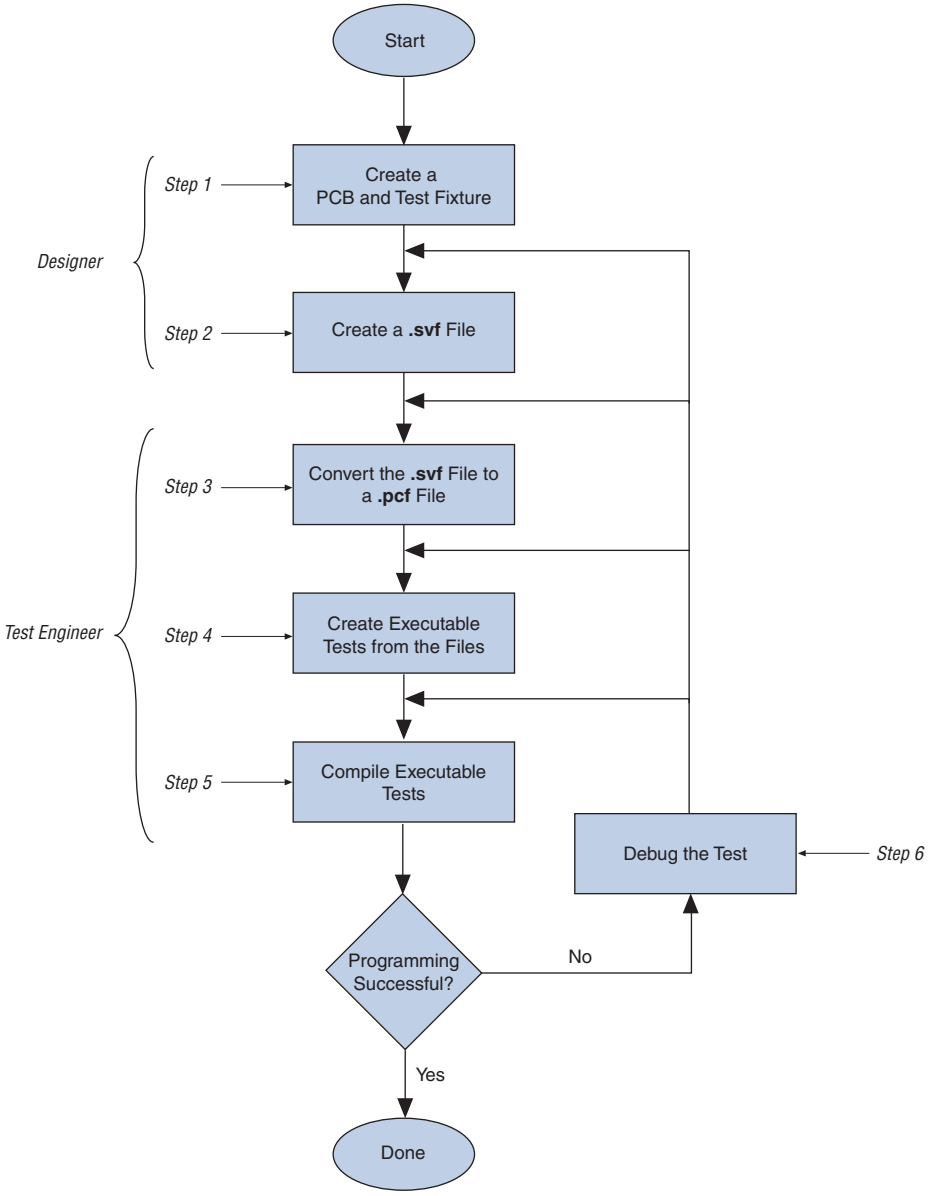
101 Innovation Drive
San Jose, CA 95134
www.altera.com

Development Flow for the Agilent 3070 Tester Without the PLD ISP Software

Figure 1 shows the steps required by Agilent when programming devices with the Agilent 3070 tester (with a Serial Vector Format [**.svf**] file) without the PLD ISP software.

The following section describes each step shown in Figure 1.

Figure 1. Agilent 3070 Development Flow for ISP Using a .svf File Without the PLD ISP Software



Step 1: Create a PCB and Test Fixture

The first step when using ISP is to properly layout your board and create the test fixture.

Creating the PCB

The following recommendations highlight important areas of PCB design issues:

- You must treat the TCK signal trace as carefully as a clock tree. TCK is the clock for the entire JTAG chain of devices. These devices are edge-triggered on the TCK signal; therefore, you must protect this signal from high-frequency noise and ensure good signal integrity. Ensure that the signal meets the pulse rise time (t_R) and pulse fall time (t_F) parameters specified in the respective *Device Family Datasheet*.
- Add a pull-down resistor to the TCK signal. Hold the TCK signal low using a pull-down resistor in-between the different pattern capture format (**.pcf**) file downloads. Hold the TCK signal low because the Agilent 3070 drivers go into a "high-Z" state in-between tests and briefly drive low when the next **.pcf** file is applied. When the TCK line "floats", the programming data stream is corrupted and the device is not correctly programmed.

  For more information about **.pcf** file downloads, refer to "Step 2: Create a .svf File".

- Provide VCC and GND test access points (TAPs) for the nails of the test fixture. During operation, there must be enough TAPs to allow quiet PCB operation. Having too few TAPs results in a noisy system that can disrupt the JTAG scans.
- To reduce system noise, turn off the on-board oscillators. During programming, on-board oscillators must have the ability to be electrically turned off.
- Add external resistors to pull outputs to a defined logic level during programming.

  During programming, the output pins are tri-stated and are pulled up by a weak internal resistor. However, Altera recommends forcing signals requiring a pre-defined level to the appropriate level using an external resistor.

For more information about board design for ISP, refer to *AN 100: In-System Programmability Guidelines*.

Creating the Fixture

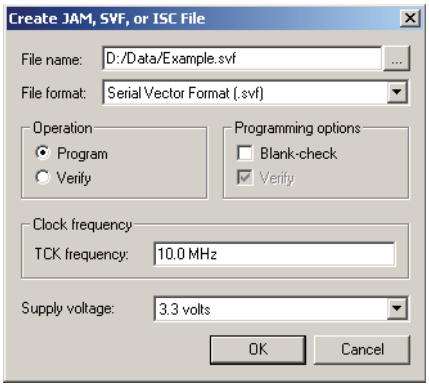
For successful ISP, provide a clean interface between the test fixture and the target board by using short wires in the test fixture to improve the TCK connection. Longer wires can introduce inductive noise into the system that can disrupt programming. Ensure the wire connecting TCK is not longer than one inch. Use the Agilent Fixture Consultant software to manage the layout and creation of the test fixture (refer to the *Agilent Board Test Family Manual*).

Step 2: Create a .svf File

The Quartus® II software generates **.svf** files for programming one or more devices. When targeting multiple devices in the same MAX II and MAX V device families, the Quartus II software automatically generates one **.svf** file to concurrently program the devices. Therefore, the programming time for all of the devices approaches the programming time for the largest CPLD device in the IEEE Std. 1149.1 JTAG chain.

Figure 2 shows the **Create JAM, SVF, or ISC File** dialog box in the File menu that you can use to generate the **.svf** file.

Figure 2. Create JAM, SVF, or ISC File Dialog Box



Before creating the **.svf** file, open the Programmer in the Quartus II software and add the Programmer Object File (**.pof**) for all the devices in the chain into the programmer. Each **.pof** corresponds to each targeted device.

In the **Create JAM, SVF, or ISC File** dialog box, the value in the **TCK frequency** box must match the frequency that `TCK` runs at during the test. If you enter a different frequency from the frequency used in the test, your programming might fail or you might experience an extended programming time.

You can select whether to perform a program operation or verify an operation and optionally to verify or blank-check the device by turning on the **Programming options**. Altera recommends generating **.svf** files that include verify vectors, which ensure that programming failures are identified and a limited amount of additional programming time is used. You can generate the necessary **.svf** files based on the scan-chain topology of the board and the Altera devices to be programmed. After the **.svf** file is generated, you can give the files to the test engineers for development.

If you program a device independently, generate an individual **.svf** file for each Altera device in the chain. When you create the **.svf** file for a single device in the chain, specify the **.pof** for the device and set the rest of the devices to ***<none>*** by selecting **Add Device** in the Programmer. These devices are bypassed during programming. Repeat this process until all your targeted devices have a **.svf** file.

Step 3: Convert the .svf Files to the .pcf Files

Use the Altera **svf2pcf** conversion utility to convert the **.svf** files to **.pcf** files for use with the Agilent 3070 tester. The **svf2pcf** utility creates multiple **.pcf** files for one device chain. Running the utility allows you to specify the number of vectors per file. The amount of memory used by the resulting files varies depending on the data.

The Agilent 3070 digital compiler detects repeating patterns of vectors and optimizes the directory and sequences the RAM on the tester control card to apply the maximum number of vectors before re-loading the files. The number of vectors in a compiled **.pcf** file ranges from 100,000 to over one million, depending on the size and density of the targeted devices.

Download the **svf2pcf** conversion utility from the Agilent ISP Support page on the Altera website.

Step 4: Create Executable Tests from the Files

To create digital tests for programming a chain of devices with the Agilent 3070 tester, follow the steps described in these sections:

a. "Create the Library for the Target Device or Scan Chain" on page 5

b. "Run the Test Consultant" on page 6

c. "Create Digital Tests" on page 6

d. "Create the Wirelist Information for the Tests" on page 6

e. "Modify the Test Plan" on page 6

Create the Library for the Target Device or Scan Chain

The initial program development for the board contains a setup-only node test library for the ISP boundary-scan chain interface. The test library ensures that the Agilent 3070 tester resources are reserved in the test fixture for programming the target devices. If only one target device is on the board and it is not part of an isolated boundary-scan chain, use a pin library; otherwise, use a node library. If you use a pin library, you must describe every device pin. Do not include test vectors in a test library.

The following code example shows a setup-only node test library.

```
!Setup only test for the boundary scan chain
assign  TCK  to  nodes   "TCK"! Node name for the TCK pin
assign  TMS  to  nodes   "TMS"! Node name for the TMS pin
assign  TDI  to  nodes   "TDI"! Node name for the TDI pin
assign  TDO  to  nodes   "TDO"! Node name for the TDO pin
inputs  TCK, TMS, TDI
outputs TDO
pcf order is TCK, TMS, TDI, TDO! The order is defined by the program
that
    ! generates the PCF files.
```

Mark the `TCK` and `TMS` boundary-scan nodes as `CRITICAL` in the Board Consultant. This critical attribute minimizes the wire length of the nodes in the test fixture.

Run the Test Consultant

Run the Test Consultant to create all of the files for your new board development. After the Test Consultant finishes running the setup-only test library, it creates an executable test (without vectors) with the correct fixture wiring resource information. Use this file as a template to create the executable test source code.

Create Digital Tests

Create the digital tests that are required to program the devices by copying the executable template to the desired program names. For example, if **svf2pcf** created four **.pcf** files, copy the template file to the four executable tests (for example, `prog_a`, `prog_b`, `prog_c`, and `prog_d`) in the digital directory.

Add these test names to your **testorder** file and mark them `permanent` with the following syntax:

```
test digital "prog_a";  permanent
test digital "prog_b";  permanent
test digital "prog_c";  permanent
test digital "prog_d";  permanent
```

Create the Wirelist Information for the Tests

Compile the executable tests to generate object files (refer to "Modify the Test Plan") for the setup-only versions of the tests. Run **Module Pin Assignment** to create the necessary entries in the **wirelist** file.

Next, modify the executable tests so the tests contain the vectors to program the target device. You can use an `include` statement in the executable test or you can merge the vectors into the file. Use the following syntax for the `include` statement, which must be the last statement in the executable test.

```
include "pcf1"
```

The **.pcf** file must reside in the digital directory and must be a digital file. To ensure that the digital file is in the correct directory, run the following command on the BT-Basic command line:

```
load  digital  "digital/pcf1"  |  re-save
```

You can also use the `chtype` command at a shell prompt to verify the location of the file:

```
chtype  -n6  digital/pcf1
```

Repeat this step for each **.pcf** file.

Modify the Test Plan

Add the test statements to the test plan with the following syntax:

```
test  "digital/prog_a" ! First program file
test  "digital/prog_b" ! Second program file
test  "digital/prog_c" ! Third program file
test  "digital/prog_d" ! Fourth program file
```

Keep the test execution in the same order in which the **.svf** file was split. For example, if the **.svf** file was split into four files (**pcf1**, **pcf2**, **pcf3**, and **pcf4**), execute the tests in the order that they were split (for example, `prog_a`, `prog_b`, `prog_c`, and `prog_d`). If the order is not preserved, the device fails to program correctly.

Step 5: Compile the Executable Tests

Altera recommends using batch-driven compilation with either a BT-Basic or a UNIX shell. Refer to the following batch file code in BT-Basic (assuming four executable tests to program the target device and generation of debugging object code):

```
compile  "digital/prog_a" ;  debug
compile  "digital/prog_b" ;  debug
compile  "digital/prog_c" ;  debug
compile  "digital/prog_d" ;  debug
```

Save this file in the board directory to allow engineering changes to take place at a later date. Refer to the corresponding shell script (the "–D" option generates debugging information):

```
dcomp  -D  digital/prog_a
dcomp  -D  digital/prog_b
dcomp  -D  digital/prog_c
dcomp  -D  digital/prog_d
```

Compilation can be time-consuming, depending on the number of **.pcf** file vectors contained in the source files, the type of controller, and controller loading. Altera recommends using a batch file to automate the compilation of the ISP tests.

If you use a boundary-scan chain containing defined Altera devices, only the defined Altera devices are programmed after the **.pcf** file vectors have been applied to the JTAG interface.

Step 6: Debug the Test

After you have created the executable tests, you can debug the test system. The applied vector set ensures that the device is correctly programmed by verifying the contents of the device. The programming algorithm uses the TDO pin to check the bitstream coming from the device. If any vector does not match the expected value, the test fails, resulting in one of two scenarios:

- The device ID does not match what is expected. This scenario is evident if the failure occurs at the beginning of the first test.
- The device programming failed.

You are not required to sift through each vector to determine the cause of the failure. If the device fails to program, use the following troubleshooting guidelines:

- Check the pull-down resistor in the test fixture. There may be pull-up resistors on the board for the TCK pin. If the pull-down resistor is too large, the TCK pin may be above the threshold of the device for a logic low. Adjust the value of the resistor accordingly. For the specification on the input logic levels, refer to the appropriate device family datasheet.
- If an overpower error on the TCK pin occurs, check the value of the resistors because they may be too low for the test system to back-drive for an extended period of time.
- Ensure that the test execution order is correct. If you execute the tests out of order, the programming information will be incorrect. Also, if you execute the same test twice in a row, the target device may be out of sequence and may not receive the correct programming information.

- Ensure that the actual vectors match the expected values for the input pins (`TCK`, `TMS`, and `TDI`). If they are not the same, recompile the tests.
- Ensure that the **.pcf** file order statement in the test matches the order of the **.pcf** file code generated in "Step 2: Create a .svf File" on page 4. If they do not match, change the order and recompile the tests.
- If possible, verify that the device is programmed correctly with the Quartus II software, the ByteBlaster™ II download cable, and the **.pof** file that you used to generate the **.svf** file. This action is not practical in a production situation, but is useful during test development and debugging.
- If you need to isolate an individual device, generate an individual **.svf** file for each targeted Altera device in the chain (refer to "Step 2: Create a .svf File" on page 4). Use this process when a verification error occurs and you have programmed more than one Altera device in the chain.
- If the device fails to program, review the boundary-scan chain definition. Ensure that the number of bits for the instruction register are correctly specified for each device in the chain. If you defined an incorrect number of bits for any device in the chain, the programming test fails.

After the test is running correctly, the board is ready for production programming. Altera recommends saving the **.pcf** files and object code for back-up purposes. To minimize the size of the stored binaries and files, use a compression program.

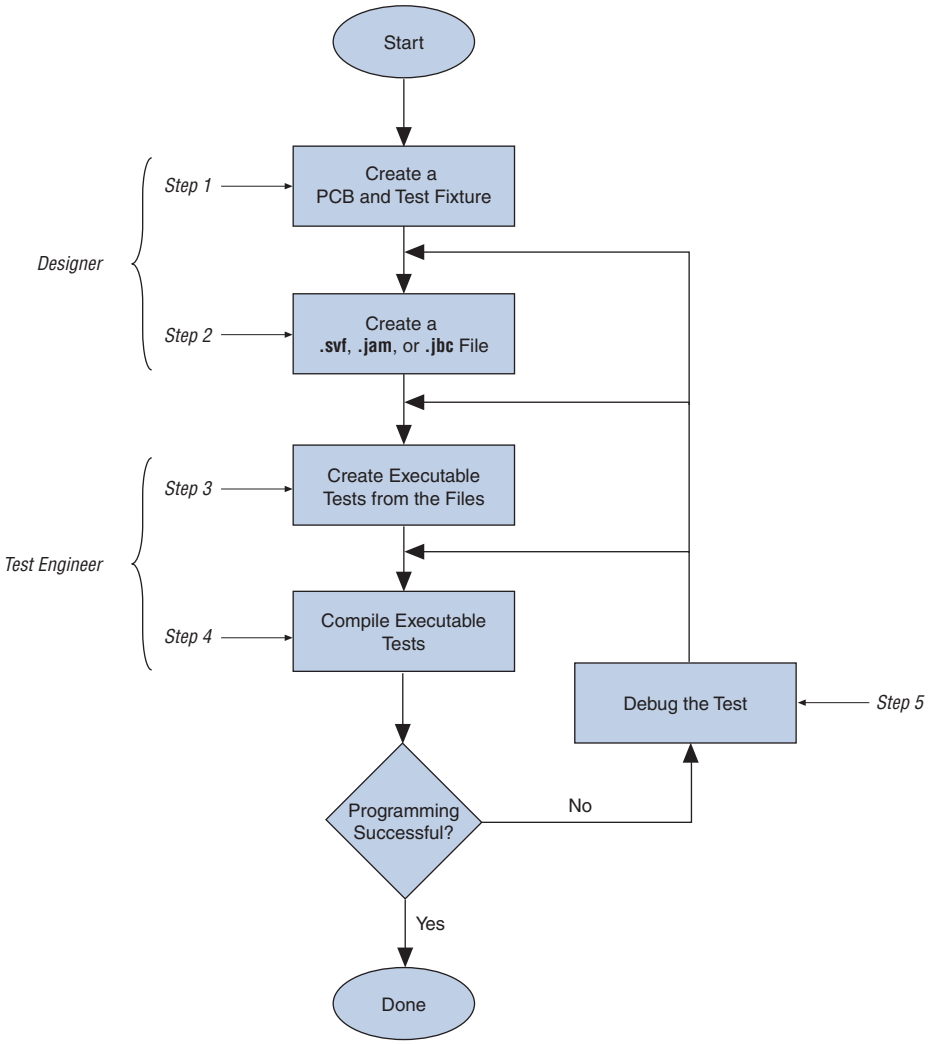
Development Flow for the Agilent 3070 Tester With the PLD ISP Software

Programming devices with the Agilent 3070 tester and the PLD ISP software is different from the steps shown in Figure 1 on page 2.

Figure 3 shows the development flow using the Agilent 3070 tester with the Agilent optional PLD ISP software.

The following section describes each step shown in Figure 3.

Figure 3. Agilent 3070 Development Flow for ISP using the Agilent PLD ISP Software



The advantages of using the Agilent PLD ISP software over the **svf2pcf** flow for device programming are:

- The tester can support the programming of devices with an **.svf** file, Jam™ Standard Test and Programming Language (STAPL) Format (**.jam**) file, or Jam Byte Code (**.jbc**) file directly (that is, no conversion to **.pcf** or VCL files).
- The Agilent 3070 digital test used to program a device is only one file.
- Pull-up and pull-down resistors are not required on the TCK and TMS lines because the device programming executes as one test.
- The size of the digital test source file and the compiled object file is much smaller than with the **svf2pcf** solution.
- Execution time for larger CPLDs and configuration devices is faster because you only execute a single digital test file.

Jam Byte-Code Player

With the Agilent PLD ISP software, a Jam Byte-Code Player is implemented in the Control XTP card of the tester. This allows you to program your devices with **.jbc** files generated from the Quartus II software. The tester also supports **.jam** and **.svf** files by using a JBC compiler to compile these files for programming. The Jam Byte-Code Player is executed through the micro controller on the Control XTP card and allows you to apply vectors algorithmically rather than executing a sequence of vectors. The Jam Byte-Code Player reads the programming and erase pulse width registers of the devices and uses those values to the programming and erase algorithms.

Programming Times

Programming times on the Agilent 3070 tester are very consistent. The only variable is the TCK frequency, which affects the programming times because the programming time is a function of the TCK clock rate. The faster the clock, the less time is required to shift data into the device. MAX II and MAX V devices support TCK clock rates up to 18 MHz.

Guidelines

When using the Agilent 3070 tester for programming, follow these guidelines:

- Use caution if you use a pin library to describe the target device in a stand-alone boundary-scan chain. Altera does not recommend describing all of the ISP device I/O pins as bidirectional. This practice uses a large number of hybrid card channels and potentially causes a fixture overflow error when developing the test.
- Do not include **.pcf** file vectors in the test library. Use a setup-only node library. Creating a test library with the **.pcf** file vectors creates a large library object file and results in slower test development time. This delay occurs because the integrated program generator looks at the entire vector set of the library object to determine if vectors must be commented out due to conflicts. Library object compiles are different from executable compiles. Additionally, the integrated program generator might fail due to the large library object file.

- To save time and disk space, generate the **.svf** files that include a verify in the programming operation. This process integrates verification vectors into one step, minimizing the amount of work in the test development process. This integrated verify accurately captures any programming errors; therefore, it is not necessary to add an additional stand-alone verify in the test sequence.
- While this document describes how to generate a test to apply vectors to the device for programming, you will need a boundary-scan description language (BSDL) file to test the device functionally. To perform a boundary-scan test or functional test, generate a BSDL file for the programmed state of the target device that contains the pin configuration information (for example, which pins are inputs, outputs, or bidirectional pins). Use the Agilent 3070 boundary-scan software to generate the test.

For more information about boundary-scan testing support, refer to the *IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices* chapter in the *MAX II Device Handbook* or the *JTAG Boundary-Scan Testing for MAX V Devices* chapter in the *MAX V Device Handbook*.

Document Revision History

Table 1 lists the revision history for this application note.

Table 1. Document Revision History

Date	Version	Changes
December 2010	1.0	Initial release.