15. Using the Agilent 3070 Tester for In-System Programming

MII51016-1.5

Introduction

In-system programming is a mainstream feature in programmable logic devices (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 software and device support that integrates in-system programmability (ISP) into the existing test flows for the Agilent 3070 system. This chapter discusses how to use the Agilent 3070 test system to achieve faster programming times for Altera's MAX® II devices.

This chapter contains the following sections:

New PLD Product for Agilent 3070

Agilent Technologies, the manufacturer of the Agilent 3070 tester, has introduced a new PLD ISP software product to help address the issues of programming PLDs. There are several advantages of using the new product that are discussed later in this chapter.

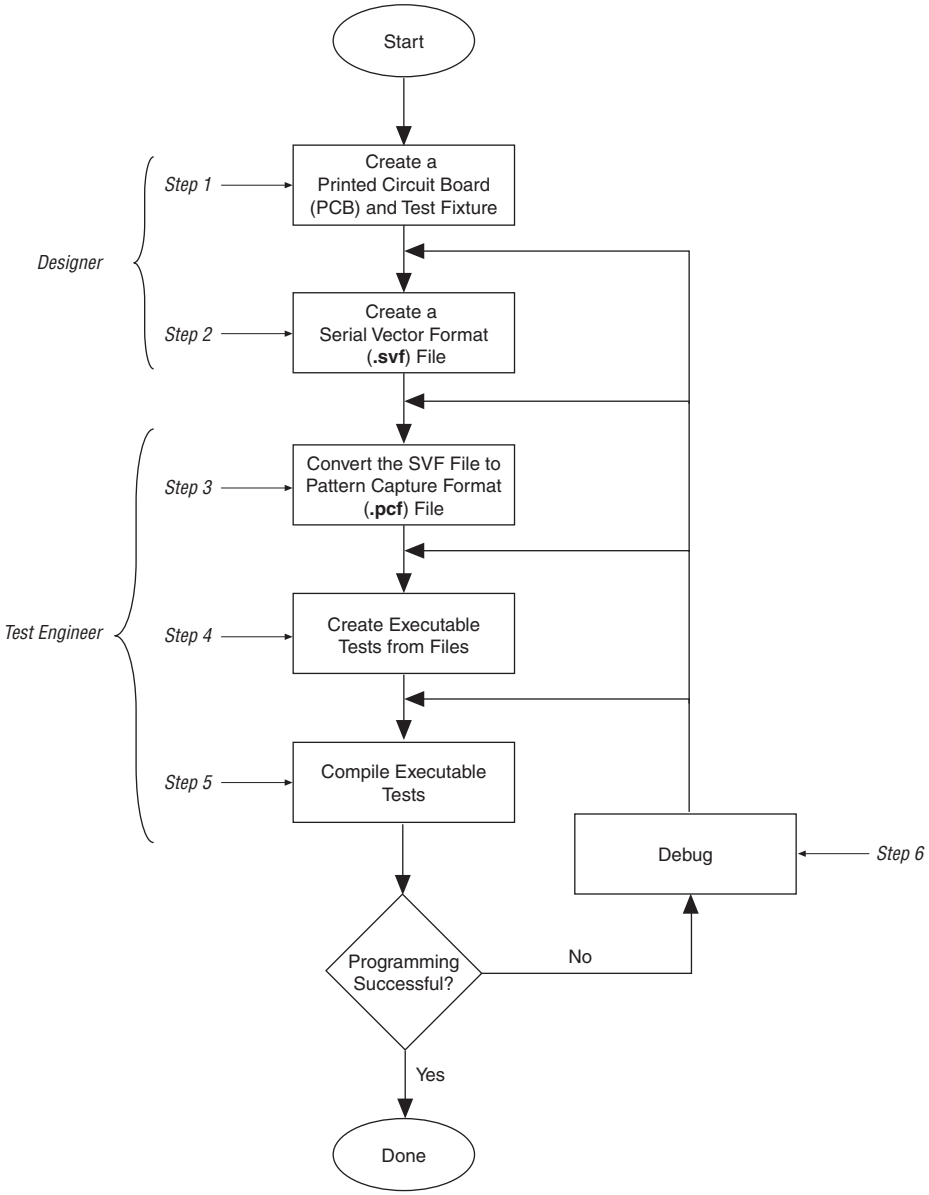
Device Support

When programming MAX II devices together with devices from other families using the Agilent 3070 tester, ensure that all devices in the chain can be programmed using the tester.

Agilent 3070 Development Flow without the PLD ISP Software

Programming devices with the Agilent 3070 tester (using a Serial Vector Format (**.svf**) File) without Agilent's PLD ISP software requires the following steps. Refer to Figure 15–1.

Figure 15–1. Agilent 3070 Development Flow for In-System Programming Using SVF File without PLD ISP



Step 1: Create a PCB and Test Fixture

Before starting test development, the first step to successful in-system programming is the proper layout of the board and the subsequent creation of the test fixture.

Creating the PCB

The following recommendations highlight important areas of PCB design issues:

- The TCK signal trace should be treated as carefully as a clock tree. TCK is the clock for the entire Joint Test Action Group (JTAG) chain of devices. These devices are edge-triggered on the TCK signal, so it is imperative that this signal be protected from high-frequency noise and have good signal integrity. Ensure that the signal meets the tR and tF parameters specified in the device data sheet.
- Add a pull-down resistor to TCK. The TCK signal should be held low through a pull-down resistor in-between PCF downloads. For more information about pattern capture format (PCF) downloads, refer to "Step 2: Create a Serial Vector Format File". You should hold TCK low because the Agilent 3070 drivers go into a "high-Z" state in-between tests and briefly drive low as the next PCF is applied. When the TCK line "floats", the programming data stream is corrupted and the device is not programmed correctly.
- Provide VCC and GND test access points for the nails of the test fixture. During operation, there should be enough access points to allow quiet PCB operation. Having too few access points results in a noisy system that can disrupt JTAG scans.
- Turn off on-board oscillators. During programming, on-board oscillators should have the ability to be electrically turned off to reduce system noise.
- Add external resistors to pull outputs to a defined logic level during programming.

Output pins are tri-stated during programming and are pulled up by a weak internal resistor. However, Altera recommends that signals requiring a pre-defined level be externally forced to the appropriate level using an external resistor.

For more information about board design for ISP, refer to the *In-System Programmability Guidelines for MAX II Devices* chapter in the *MAX II Device Handbook*.

Creating the Fixture

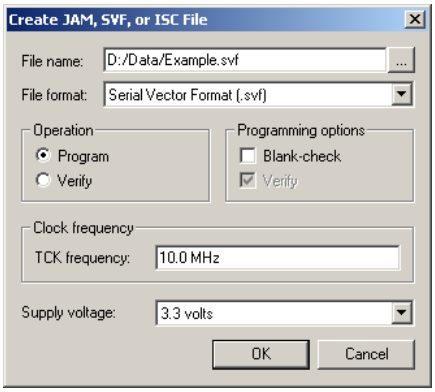
Providing a clean interface between the test fixture and the target board is essential for successful in-system programming. To provide a clean interface, use short wires in the test fixture to improve the TCK connection. Longer wires can introduce inductive noise into the system, which can disrupt programming. The wire connecting TCK should be no longer than 1 inch. Use the Agilent Fixture Consultant to manage the layout and creation of the test fixture (see the Agilent Board Test Family Manual).

Step 2: Create a Serial Vector Format File

The Quartus II software generates SVF Files for programming one or more devices. When targeting multiple devices in the same MAX II CPLD family, the Quartus II software automatically generates one SVF File to program the devices concurrently. 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 15–2 shows the **Create JAM, SVF, or ISC File** dialog box (File menu), which is used to generate the SVF File.

Figure 15–2. Create JAM, SVF, or ISC File Dialog Box



Before creating the SVF File, you must open the Programmer in the Quartus II and add the Programmer Object File (**.pof**) for all the devices in the chain into the programmer. Each POF corresponds to a targeted device, respectively.

In the **Create JAM, SVF, or ISC File** dialog box, the value in the TCK frequency box should match the frequency that TCK runs at during the test. If you enter a different frequency from the one used in actual testing, programming may fail or you may experience an excessively long programming time.

You can also select whether to perform a program or verify operation and optionally verify or blank-check the device by turning on 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 File based on the scan-chain topology of the board and the Altera devices to be programmed. Once the SVF File is generated, it can be given to test engineers for development.

If a device must be programmed independently, you can generate individual SVF Files for each Altera device in the chain. When creating the SVF File for a single device in the chain, specify the POF for the device and leave the rest of the devices set to <none>. This can be done by selecting **Add Device** in the Programmer. These devices are bypassed during programming. Repeat this process until all targeted devices have an SVF File.

Step 3: Convert SVF Files to PCF Files

You must convert the SVF Files to PCF Files for use with the Agilent 3070 tester with the Altera **svf2pcf** conversion utility. The **svf2pcf** utility can create 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 looks for repeating patterns of vectors and optimizes the directory and sequences 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.

You can download the **svf2pcf** conversion utility from the Agilent ISP Support website at www.altera.com.

Step 4: Create Executable Tests from Files

Creating digital tests for programming a chain of devices with the Agilent 3070 tester requires the following steps:

1. Create the library for the target device or scan chain.
2. Run the Test Consultant.
3. Create digital tests.
4. Create the wirelist information for the tests.
5. Modify the test plan.

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 Agilent 3070 tester resources are reserved in the test fixture for programming the targeted devices. If only one target device is on the board and it is not part of a boundary-scan chain (isolated), use a pin library; otherwise, use a node library. If using 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 nodes' wire length in the test fixture.

Run the Test Consultant

Run the Test Consultant to create all of the files for new board development. Once the Test Consultant finishes running with this
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's source code.

Create Digital Tests

Create the digital tests, which are required to program the device(s), by copying the executable template to the desired program names. For example, if **svf2pcf** created four PCF Files, copy the template file to 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 using 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 these executable tests to generate object files (see "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 that they contain the vectors to program the target device. An `include` statement can be used in the executable test, or the vectors can be merged into the file. Use the following syntax for the `include` statement, which should be the last statement in the executable test.

```
include "pcf1"
```

Remember that 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 using 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**), the tests must be executed in the order that they split (execute `prog_a` followed by `prog_b` followed by `prog_c` followed by `prog_d`). If the order is not preserved, the device(s) will fail to program correctly.

Step 5: Compile the Executable Tests

Altera recommends batch-driven compilation using either BT-Basic or a UNIX shell. See 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
```

This file should be saved in the board directory to allow engineering changes to take place at a later date. See the corresponding shell script (–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
```

Compile times can be long, depending on the number of PCF 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 a boundary-scan chain containing Altera devices is defined, only the Altera devices will be programmed when the PCF vectors have been applied to the JTAG interface.

Step 6: Debug the Test

Once the executable tests have been created, the test system can be debugged. The applied vector set ensures that the device is programmed correctly 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, indicating one of two things:

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

Because many vectors are verified, it may not be practical to sift through each vector to determine the cause of the failure. Use the following troubleshooting guidelines if the device fails to program:

- Check the pull-down resistor in the test fixture. The design engineer may have placed 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 device's threshold for a logic low. Adjust the value of the resistor accordingly. See the appropriate device family data sheet for the specification on input logic levels.

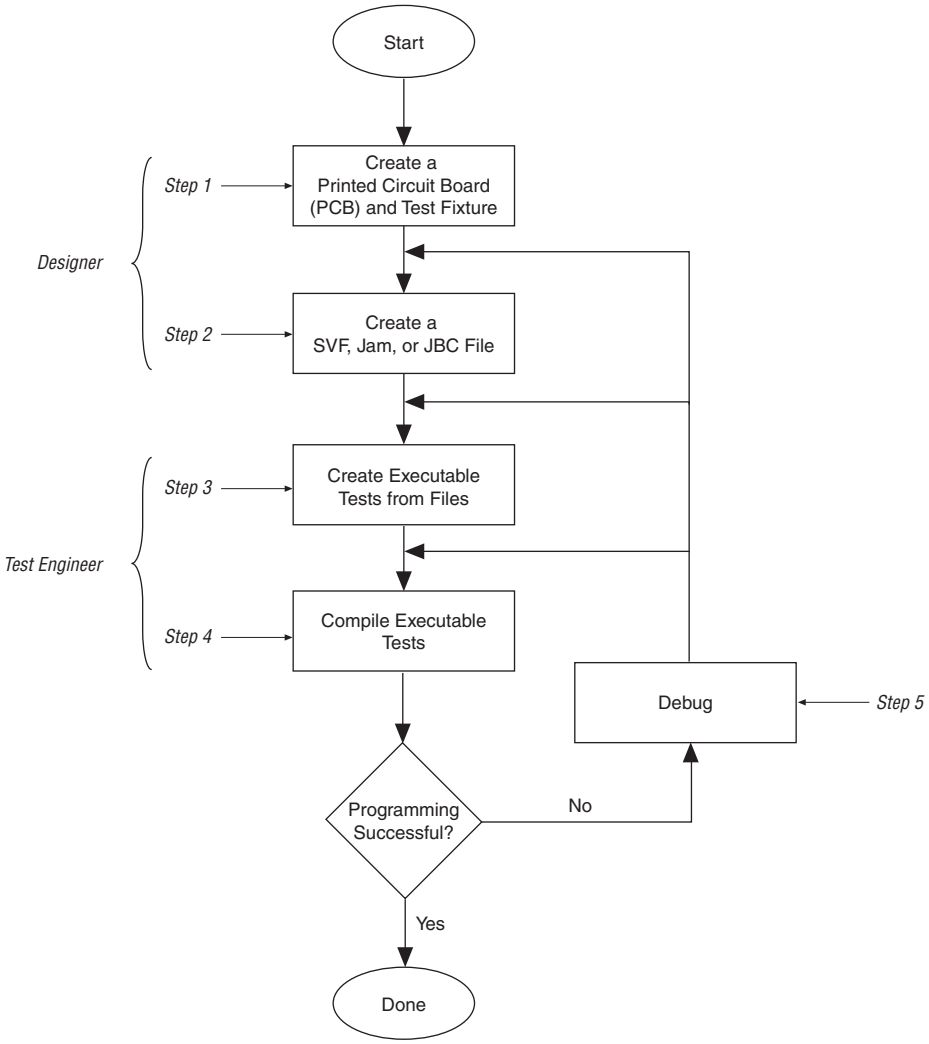
- If an overpower error on the TCK pin occurs, check the value of the resistors because they may be too low for the test system to back-drive for an extended period of time.
- Ensure that the test execution order is correct. If the tests are executed out of order, the programming information is incorrect. Also, if the same test is executed twice in a row, the target device will be out of sequence and will 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, the tests may need to be recompiled.
- Ensure that the pcf order statement in the test matches the order of the PCF code generated in "Step 2: Create a Serial Vector Format File" on page 15–4. If they do not match, the order must be changed and the tests recompiled.
- If possible, verify that the device is programmed correctly by using the Quartus II software, the ByteBlaster™ II download cable, and the POF that was 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, you can generate an individual SVF File for each targeted Altera device in the chain. The process of generating the SVF Files is explained in "Step 2: Create a Serial Vector Format File" on page 15–4. This process is useful when a verification error occurs and more than one Altera device is programmed in the chain.
- If you still have problems, look at the boundary-scan chain definition. Make sure that the number of bits for the instruction register are specified correctly for each device in the chain. If an incorrect number of bits have been defined for any device in the chain, the programming test will fail.

Once the test is running smoothly, the board is ready for production programming. Altera recommends saving the PCF Files and object code for back-up purposes. Use a compression program to minimize the size of the stored binaries and files.

Development Flow for Agilent 3070 with PLD ISP Software

Programming devices with the Agilent 3070 tester and PLD ISP software is slightly different than the steps in Figure 15–1. Figure 15–3 shows the development flow using the Agilent 3070 tester with Agilent's optional PLD ISP software.

Figure 15–3. Agilent 3070 Development Flow for In-System Programming with Agilent's PLD ISP Software



Some advantages of using the Agilent PLD ISP software over the SVF2PCF flow for device programming are:

- The tester can support the programming of devices using SVF, Jam STAPL, or JBC file formats directly (that is, no conversion to PCF or VCL).
- The Agilent 3070 digital test to program a device is only one file.
- Pull-up and pull-down resistors are not required on the TCK and TMS lines in the fixture of the tester since the device programming executes entirely as one test.
- The size of the digital test source file as well as the compiled object file is much smaller than with the SVF2PCF solution.

- Execution time for larger CPLDs and configuration devices is faster as only a single digital test file is executed.

With Agilent's PLD ISP software, a Jam Byte-Code Player is implemented in the Control XTP card of the tester. This allows users to program devices using JBC files created directly from Quartus II. The tester also supports Jam or SVF files as it has a JBC compiler to compile these files for programming. The Jam Byte-Code Player is executed via the microcontroller on the Control XTP card and allows users 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 in the programming and erase algorithms.

Programming Times

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

Guidelines

While using the Agilent 3070 tester for programming, use the following guidelines:

- Use caution if a pin library is used to describe the target device in a stand-alone boundary-scan chain. Altera does not recommend describing all of the ISP device's 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 vectors in the test library. Use a setup-only node library. Creating a test library with PCF vectors creates a large library object file and results in a much slower test development time. This delay occurs because the integrated program generator (IPG) looks at the entire vector set of the library object to determine if vectors need to be commented out due to conflicts. Library object compiles are different from executable compiles. Additionally, the IPG may fail due to the large library object file.
- To save time and disk space, generate 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, a boundary-scan description language (BSDL) file is required to functionally test the device. If you need 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 a test.

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

Conclusion

Altera provides complete solutions for programming all MAX II devices using the Agilent 3070 test system. All MAX II devices can be programmed together with other ISP-capable devices. With software and device support, the opportunity for cutting costs and increasing manufacturing productivity is available to any Agilent 3070 user.

Referenced Documents

This chapter references the following documents:

- *IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices* chapter in the *MAX II Device Handbook*
- *In-System Programmability Guidelines for MAX II Devices* chapter in the *MAX II Device Handbook*

Document Revision History

Table 15–1 shows the revision history for this chapter.

Table 15–1. Document Revision History

Date and Revision	Changes Made	Summary of Changes
October 2008, version 1.5	■ Updated New Document Format.	—
December 2007, version 1.4	■ Added "Referenced Documents" section.	—
December 2006, version 1.3	■ Added document revision history.	—
June 2005, version 1.2	■ Text edit to the "Programming Times" section (25 MHz to 18 MHz).	—
January 2005, version 1.1	■ Previously published as Chapter 16. No changes to content.	—