Agilent Technologies

In-System Programming

on the

Agilent Technologies 3070

Test System

Q. Is it possible to program ISP devices on the Agilent 3070 and if so, how do I?

A. This question is being asked by customers more and more frequently and the answer is yes, you can. In fact, performance can be quite good on the Agilent 3070. For example, four Altera ISP devices programmed concurrently in under 6 seconds (see details later in this document)! This white paper will help guide you through that process.

Please note that this user-contributed paper addresses the issues related to programming ISP devices and not for other programmable devices such as FLASH and EEPROMs. In addition, most of the discussion refers to the IEEE-1149.1 Boundary Scan method of programming ISP device. At the end of this paper is a glossary to help explain key acronyms used throughout this discussion.

ISP BACKGROUND

In-System Programmable devices are being more widely used on today's boards to provide flexibility to the design engineers and to reduce the cost of a product. Some of the advantages of ISP devices over traditional "program once" logic devices and custom gate arrays are listed below:

- Reduced development time by allowing the devices to be reconfigured on the board. This translates into a faster time to market.
- The use of sockets for PLDs can be eliminated due to ISP reprogrammability on the board thus resulting in a single PCB layout for the design.
- Reduced manufacturing costs by not stocking pre-programmed devices. Additional manufacturing steps associated with programming blank parts and putting them into a parts inventory can be eliminated. Now you only need to stock the unprogrammed device. In addition, the reworking of boards due to placing a programmed part at an incorrect location will be eliminated.
- By eliminating the programmed parts stocking, reduced handling of the devices occurs. As a result, you can reduce scrap costs associated with coplanarity and ESD damage to the devices, not to mention the PCBs the devices are mounted on.
- ISP devices can provide an easy means for engineering changes or feature set enhancements to a product. With ISP devices in a system, you could implement an engineering change to reconfigure the logic within the device without removing the device from the PCB. This could even be done in the field if the system is designed with that capability.

With all of these advantages, it is no wonder that engineers are taking advantage of these devices on the new designs. However, because many of the advantages are related to manufacturing, process flows are being changed accordingly. It is here that this technology affects the Agilent 3070 test programmer.

GENERAL CONSIDERATIONS for ON-BOARD ISP PROGRAMMING

Because these devices are programmable after the device is placed on the board, it is being requested that the ISP devices are programmed at the test process step. Traditional PLDs were pre-programmed and placed on the board where the test engineer would develop a device test for the programmed PLD. Now that ISP devices provide the

capability of being programmed on the board, the test engineer is being asked to not only test the device, but to program it as well.

Regardless of whose ATE system you use, you must, as a test engineer, be aware of the trade-offs related to programming ISP devices on the ATE system. ISP devices traditionally take a relatively long time to program. This is related to the physics of the ISP device and the algorithm used to program it. The time to program an individual device may be in the tens of seconds on the ATE system. If the board under test has many of these devices on it, then you must multiply the programming time by the number of devices on the board.

Programming time is important to understand because the manufacturing line "beat rate" may not allow a 15 second test step plus a 60 second on-board programming step to program the ISP devices on it. The ideal solution would be to have the ISP device's programming algorithm go at "databook" speed (TCK rate at 10Mhz). This could be accomplished through a single-board computer installed in the test system or the test fixture itself that would take programming information that has been downloaded to it and directly program the ISP devices from that information. However, these added electronics may not be desirable for a variety of reasons.

Parallel (concurrent) programming of ISP devices can reduce the overall programming time. Most ISP devices support "concurrent" programming. This is the technique for programming multiple ISP devices (from the same semiconductor manufacturer) at "the same time". In reality, the configuration bit stream is applied to all the devices in the chain and a single programming pulse is applied. Thus all devices can be programmed quicker than programming each device individually. This can save significant time depending on how many programming pulses are required to program the device. However, you must understand that if the programming algorithm fails in the concurrent programming mode, you will not know which device in the chain caused the problem.

In addition, development time is a factor. You will need to re-generate the programming bitstream files and re-create the ATE specific programming test files when an engineering change is done to a device. The total time to re-generate the programming files may mean that you will not be able to implement an engineering change of the ISP devices in a timely manner.

WHAT STRATEGY SHOULD I USE TO PROGRAM THE ISP DEVICES?

The answer to this question depends on your manufacturing process requirements. If throughput across the test system is most critical and your manufacturing process is mature enough such that you do not expect to see many bad devices on the boards, then a recommended flow would be this:

1. Use Agilent TestJet to ensure that there are no lifted leads on the ISP devices (particularly on the IEEE-1149.1 interface pins).
2. Program the devices in concurrent (parallel) programming mode. This will minimize the time to program the devices.

Note that if your board contains different manufacturer's ISP devices within the same boundary scan chain, you must revert to programming each manufacturer's devices separately. This serial programming mode will result in longer ISP programming times.

If manufacturing line "beat rate" considerations allow, greater control of the individual ISP devices can be gained with the following strategy:

1. Use Agilent TestJet as described above. It is the best method for checking solder-related faults on the board.
2. Individually program each device while placing all the other devices in the boundary chain into “BYPASS” mode. Programming ISP devices in this way is referred to “serial” programming mode; program one, then the next, then the next and so on. If there is an internal failure with a device or the wrong type of ISP device is placed on the board at one of the locations in the chain, the test will fail at the time that device is being programmed.

You may want to consider creating a TAP integrity test for the boundary scan chain that could be run prior to programming the devices. This test will ensure that your boundary scan chain path works properly for the chain you defined. This test is automatically created for any boundary scan chain test generated by the optional Agilent InterconnectPlus Boundary Scan software. Note that this software is not required to program ISP devices on the Agilent 3070. Refer to the Agilent 3070 Boundary-Scan manual for more information on Agilent InterconnectPlus Boundary Scan software.

Using either of these strategies, once the device has been programmed, you could then do a digital test on the device in its programmed state. The test could be a boundary scan test (if the device is fully IEEE-1149.1 compliant), or a test generated by a third-party software package (like AcuGen or Flynn Systems). In the first scenario, you have already minimized the chance that manufacturing faults exist on the card and a digital test may not add value.

A third alternative is available for consideration. It may not be cost effective to tie-up the process test system with the programming of the ISP devices at all. You may want to try the following flow:

1. Use the normal Agilent 3070 ICT test on the board, but do not program the ISP devices at all at the Agilent 3070 process step.
2. Utilize an off-line process step featuring a power supply and a PC with an ISP download cable attached. The PC can then program the board without tying-up the Agilent 3070 system.

HOW DO I GET THE DEVICES PROGRAMMED ON THE Agilent 3070?

The flow diagram shown in Figure 1 below highlights the general steps required to create the necessary files to program the device on the Agilent 3070 test system. Detailed information for each step is provided later in this paper.

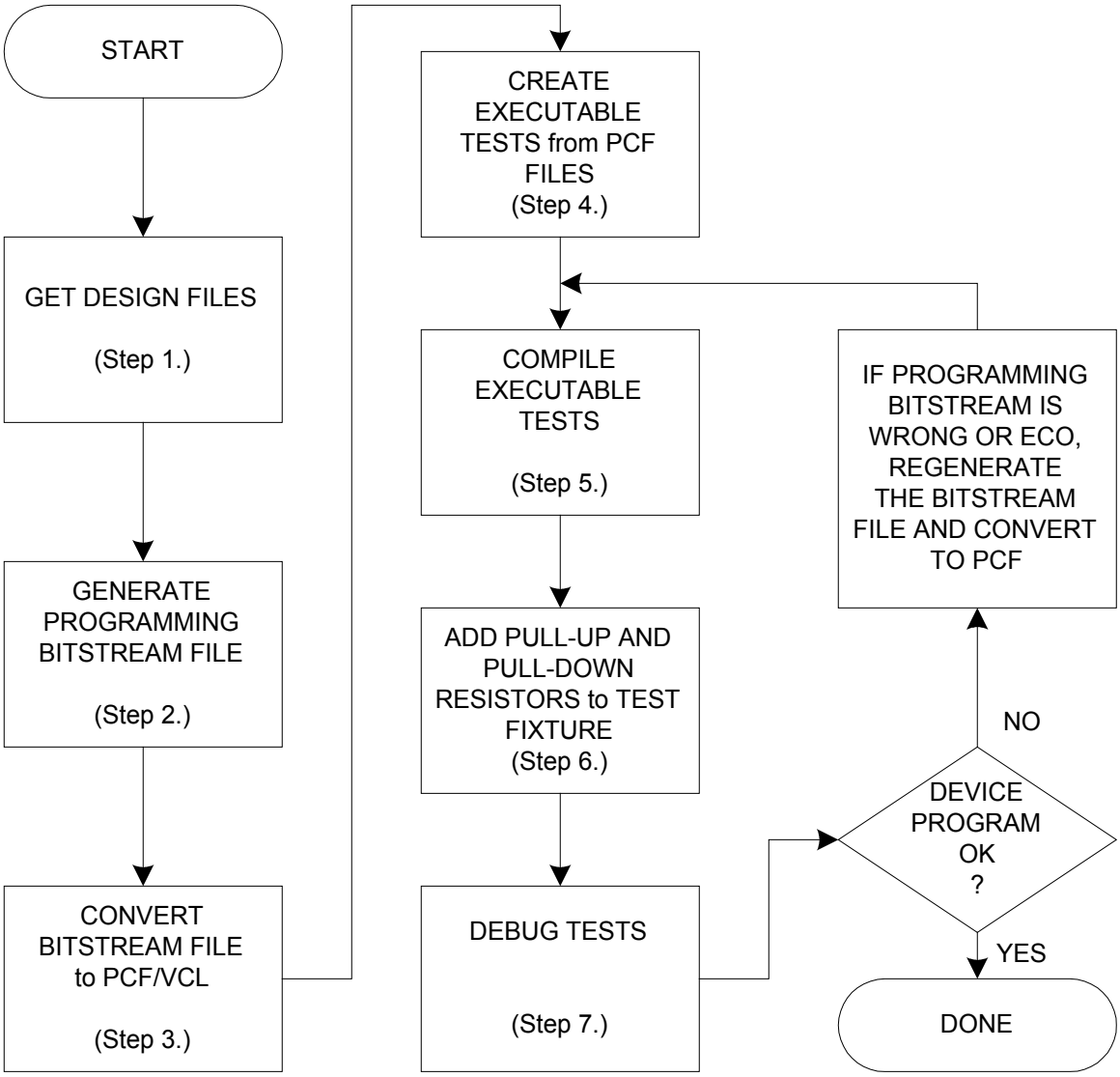

FIGURE 1. Flow for Developing ISP Programming on Agilent 3070

1. GET THE DESIGN FROM THE DESIGN ENGINEER

To create a bitstream with which to program an ISP device, you must start with a final design file from the engineer for each device. This may be in a JEDEC format or some other proprietary format from the ISP manufacturer. If you have only one ISP device, then you will need only one design file. Note that you may need to transfer this file to and/or from a PC to do some of the later steps, so you should be familiar with using a file transfer program between the PC and the Agilent 3070 workstation.

You may also have to work closely with the design engineer as the design software may be tightly integrated with the generation of the programming files. In addition, a couple of iterations of generating the programming information may be necessary before the test can be released to manufacturing. Also, you may need the design engineer's help in verifying that the ISP devices are indeed programmed correctly.

2. GENERATE THE ISP PROGRAMMING BIT STREAM FILE

Use a **vendor-supplied** tool to create the ISP programming bit stream. Many vendors will create an SVF file to program the device. Note that some ISP vendors supply this tool free of charge while others incorporate it into the design software used to develop the device configuration. In addition, this tool may be used to actually download the programming information directly to the device via some adapter connected to the PC or workstation. Check with your ISP device vendor for detailed information on how to create this bitstream file.

If multiple ISP devices are daisy-chained together and you wish to take advantage of the concurrent programming algorithm, then this bit stream file will contain the necessary bits to program all devices at once provided you described the boundary scan chain to the tool properly. Most bit stream generation software will understand a chain that does not have a homogeneous chain of devices (i.e., all devices in the chain are from the same vendor). You will need to describe the devices contained in the chain that are not made by that ISP vendor to the software by telling the software how many bits are in that device's boundary scan instruction register. This information can be determined by looking at the "INSTRUCTION_LENGTH" attribute in the BSDL file. An example of this line in a BSDL file is shown below:

attribute INSTRUCTION_LENGTH of epm7128sq160 :entity is 10;

In this example, the instruction register's length is 10 bits.

Once generated, the bit stream file will contain extra bits in it to place the devices that are not supplied by that vendor into BYPASS mode. Note that if your boundary scan chain has ISP devices from different vendors, you will need to generate bit stream files for each vendor's device or devices.

Note that if you want to be able to isolate individual ISP devices, which could fail during programming, you will need to generate several bitstream files - one for every ISP device in the chain. You would describe the chain as one ISP device with everything else in the chain placed in BYPASS mode (including other ISP devices) and generate your bitstream file. Then you would describe the chain again with the next ISP device in the chain as your target and everything else including the first ISP device in BYPASS mode and generate the next bitstream file. Repeat this process until all ISP devices have a corresponding bitstream file.

3. CONVERT THE BITSTREAM FILE INTO Agilent 3070 FORMAT

Use another tool from the ISP vendor to convert the bitstream information into Agilent 3070 VCL/PCF format. Some vendor's tools will create a VCL/PCF directly while others will use an SVF-to-PCF or SVF-to-VCL converter program.

Most of the time, the resulting PCF vectors will be split into multiple files as the total vector set will contain more vectors than can be executed with the Agilent 3070 testhead hardware. This number will vary due to the "randomness" of the vectors being applied to the ISP device to program it. The digital compiler looks for repeating patterns of vectors being applied to the device. It can then optimize the directory and sequence RAM on the control card in the testhead to apply the maximum number of vectors before re-loading. Note that you may be able to get more vectors to compile with a certain bitstream file than with another bitstream for the same ISP device type due to one bitstream file having more repeating vector sequences than the other. The number of vectors in a PCF file that can

compile could range anywhere from 100,000 to over a million. NOTE: These PCF source files will typically require greater than 5Mb of disk space each.

4. CREATE THE VCL EXECUTABLE TESTS (SOURCE CODE)

This section will describe the Agilent 3070 test development steps required to create the digital tests used to program the ISP devices. The process has 5 major parts: create library for ISP device or scan chain, run Test Consultant, create the various tests, generate wirelist information for the tests, and modify the testplan.

A. Create library test

The initial program development for the board should contain a "setup-only" node library test for the ISP boundary scan chain interface similar to that below. This will ensure that the Agilent 3070 tester resources are reserved in the test fixture for programming the ISP devices. If you only have one ISP device on the board, or your ISP devices are not part of a boundary scan chain (isolated), then you could use a pin library rather than a node library. However, you will need to describe ALL pins of the device in the pin library (see Issues section below). Do NOT include test vectors in a library test (see Issues section below).

The following code is an example of what a "setup-only" node library test may look like:

```
! 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.
```

Note that you should mark the boundary scan nodes "TCK" and "TMS" as "critical" in Board Consultant. This will minimize the wire length for these nodes in the test fixture.

B. Run Test Consultant

Run Test Consultant as you normally would to create all of the files for a new board development. Once Test Consultant finishes running with this "setup-only" library, you will have an executable test (without vectors) with the correct fixture wiring resource information. This file will be used as a template to create the executable test's source code.

C. Create the tests to program the ISP device.

You can now create the handful of digital tests that are required to program the device(s) by copying the "executable" template to the various program names. For example, if the bit stream conversion program created 4 PCF files, you would want to copy the template file to 4 executable tests (say, 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
```

D. Create wirelist information for the tests.

Compile these executable tests to get requirements object files created (see next section) for the set-up only versions of the tests. You should then run "module pin assignment" to create the necessary entries in the "wirelist" file.

Next, you should modify the executable tests to actually contain the vectors to program the ISP device. You can either use an "include" statement in your executable test to include the PCF vectors, or actually merge the vectors into the file. Use the following syntax for the include statement. It should be located as the last statement in your executable test.

```
include "pcf1"
```

Remember that the PCF file must reside in the "digital" directory AND it must be a "digital" file. You can ensure this by doing the following statement on the BT-Basic command line:

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

A quicker way to do this is through the use of the "chtype" command at a Unix shell prompt:

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

Repeat this step for each PCF file.

E. Modify the testplan.

The last thing to do is to add the test statements to the "testplan" 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
```

Be sure to keep the test execution in the same order in which the bit stream file was split. For example, if the bit stream file was split into 4 PCF 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 you do not, then the ISP device(s) will not be programmed correctly.

5. COMPILE THE EXECUTABLE TESTS

The best thing to do here is to create a batch file. This can be done in either BT-Basic or in the UNIX shell. In BT-Basic, the code should look like this (assuming four executable tests to program the device and you want debug object):

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

You should save this file in the board directory in case you have to change the ISP programming information at a later date. A UNIX shell script would look similar:

```
dcomp -D digital/prog_a  # -D option generates debug information.
dcomp -D digital/prog_b
dcomp -D digital/prog_c
dcomp -D digital/prog_d
```

Note that the compile times can be long depending on the number of PCF vectors contained in the source files, the type of controller, and loading of the controller you are compiling them on. A batch file is recommended to automate the compilation of all the ISP programming tests.

Remember that if you defined a boundary scan chain that contains several ISP devices, all of those devices from the same vendor will be programmed once the entire bit stream has been applied to the JTAG interface.

6. ADD PULL-UP AND PULL-DOWN RESISTORS TO THE FIXTURE

You will need to hold the state of the TCK and TMS lines on the boundary scan interface via a pull-up and pull-down resistor respectively in between PCF file downloads. This is because the Agilent 3070 drivers go into a "high-Z" state in between tests and if the TCK and TMS lines experience transitions on them due to noise, the programming algorithm will not be correct and the device will not be programmed correctly.

Typically, when the PCF file is split, the program that splits the file will add additional vectors to the PCF file to put the ISP device into the boundary scan "RUN TEST/IDLE" or "PAUSE-DR" or "PAUSE-IR" states. Any transients on the TCK line will have no effect on the ISP device's boundary scan state while the TMS pin is held low. If the TMS pin were held high, transients occurring on TCK would place the device in the "TEST LOGIC RESET" state. This would effectively negate the programming that we are trying to do.

7. DEBUG THE TEST

Now that we have our executable tests and the fixture has been modified, we can go to the test system to debug the test. Typically, the vector set will contain vectors that verify the device has been programmed correctly. This bitstream will show up on the TDO pin of the boundary scan chain. If the bitstream coming from the device doesn't match the expected value, the test will fail indicating that the programming of the device failed. However, since we are talking about a huge number of vectors to be applied to the board, actually determining why the device does not program may be futile. Our experience has been that ISP programming either works the first time or it doesn't work at all. Here are some of the things to look for if the device fails to program:

- Check the pull-up and pull-down resistors in the test fixture. If the PCB was designed per the IEEE1149.1 specifications, the design engineer may have put pull-up resistors on the TCK and TMS pins. If the pull-down resistor is too large, the TMS pin may be above the threshold of a low on the device. Adjust the value of the resistors accordingly.
- If you are seeing an overpower error on the TCK or TMS pins, check the value of the resistors as they may be too low for the test system to backdrive for long amounts of time.
- Make sure that the order you are executing the tests in is correct. If you execute the tests out of order, then the programming information will be incorrect. Also, if you execute the same test twice in a row, you will be programming the ISP device out of sequence and it will not have the correct configuration.
- Make sure that the actual vectors match the expected values for the input pins (TCK, TMS, and TDI). If they are not, you may need to recompile the test or try merging the PCF vectors into the VCL code rather than using an "include" statement.
- Be sure the "pcf order" statement in the test matches the order of the PCF code generated in step 3 above. If they do not match, you must change the order and recompile the tests.
- If possible, verify that the devices programmed correctly by having a design engineer try the board with the programmed parts on it in the application.
- If you are still having problems, look at the boundary scan chain definition. Make sure that the number of bits for the instruction register are specified correctly for non-ISP devices and ISP devices from a different vendor (see step 2 above). If you have defined an incorrect number of bits for any device in the chain, the programming test will fail.

AN ACTUAL EXAMPLE

Internally at Agilent Technologies, a board was developed that contained four Altera EPM7128SQC160-7 ISP devices in a chain. Each device was a 160-pin PQFP package and was configured differently than the other Altera devices. A "PlayPen" bread board containing these four devices was designed to match the boundary scan chain on the actual PCB. It was desirable to verify that we could actually program these devices on the Agilent 3070 test system before the actual board was available.

Note that the evaluation board was tested on a “PlayPen” fixture and wires connecting to the Altera devices were wire-wrapped long wires. We do not have the luxury of specifying the “critical” attribute on nodes on the “PlayPen” test fixture.

Figure 2 below shows some of the final results.

Agilent 3070 Software Revision	B.02.54	
Controller Type	725/100	
Number of PCF files created	15	
Number of vectors per file	about 700,000	
Total number of vectors executed	9,925,512	
Size of each PCF file	**5.4Mb**	
Total disc storage for PCF files	**78.7Mb**	
Total size of object files (15)	1.5Mb (about 100,000 bytes each)	
Total size of debug objects (15)	430,901 (about 28,800 bytes each)	
Total Compile time	3 hours, 17 minutes	
TCK clock rate (specific to Altera)	500kHz	2Mhz
Vector Cycle Time	1000n (1uS)	250n
Test time to program all four devices (first run)	52 seconds	41 seconds
Test time to program all four devices (subsequent runs)	23 seconds	**10 seconds**

Figure 2. ISP Programming Example Details

Note that the test object file size is much less than that for the source files (source is roughly 50 times the size of the object file). Once debugged, the source files should be compressed to save disk space (use the UNIX command **compress pcfA.pcf** to create a **pcfA.pcf.Z** compressed file).

ISSUES TO BE AWARE OF

There are a few things to keep in mind when doing ISP programming on the Agilent 3070:

- Be careful if you use a pin library to describe your ISP device in a stand-alone boundary scan chain. **It is not recommended to describe all of the ISP device’s “I/O” pins as bi-directional as you WILL use a large number of hybrid card channels and potentially end up with a fixture overflow error when you develop your test**.
- **Do NOT include PCF vectors in the library test**. Use a setup-only node library. Creating a library test with the PCF vectors in it will create a HUGE library object file and result in a much slower test development time. This is because IPG will look at the entire vector set of the library object to determine if vectors need to be commented out due to conflicts. This was tried on a board with a MACH445 device and resulted in a 7.8Mb source file and a 27.5Mb library object file! Library object compiles are very

different from executable compiles. In addition, IPG may fail due to the large library object file.

- Remember, disk space usage is a factor when developing the tests to program ISP devices. Always compress the source files once you have the tests working using the UNIX **compress** command to preserve disk space.
- To save some time and disk space, you could just generate a programming bitstream. The example shown in Figure 2 above has a programming cycle (which includes a verify pass) in addition to a separate verify pass. The test could have been developed which would only have the one program/verify pass. However, you must verify the programming sequence completed without problems and that the ISP device contains the proper configuration information at some point.
- Keep in mind that the logic levels for the programming pins may be different than the rest of the pins of the device. On some devices, these may be 5 volt logic while the other pins on the device are at 3.3 volt logic.
- While this document describes how to generate a "test" to apply vectors to the device to program it, you must use a different technique to actually apply vectors to functionally test the device. If it is possible, generate a BSDL file for the "programmed" state of the ISP device that contains the pin configuration information (what pins are inputs, outputs, or bi-directional pins). Then use the Agilent 3070 Boundary Scan software to generate a test for it. AcuGen has a product that can help generate a BSDL file from the design files called "TESTBSDL".
- Do not forget to install the pull-up and pull-down resistors in the fixture on the TCK and TMS pins respectively. Omitting these will cause the ISP programming to fail.
- Be aware that if an ISP vendor changes, their manufacturing process for ISP devices, you may see failures when programming the device. Programming pulse width variations created by the new manufacturing process could cause this. Some ISP vendors offer "fixed" programming pulse width devices guaranteed to program with a pulse width less than or equal to the fixed value. It would be useful to discuss this with your ISP vendor.

WHAT ABOUT Jam?

Jam is a file format that describes how to program ISP devices via the IEEE 1149.1 interface. Jam has been proposed to JEDEC to become an industry standard among most ISP vendors. Its advantages are a vendor-independent file format and small file size. The ISP vendor would have a tool to create a Jam file and the ATE supplier would have a Jam "player" that would interpret the Jam language from the Jam file and be able to apply the appropriate vectors to the devices to program them (See figure 3 below).

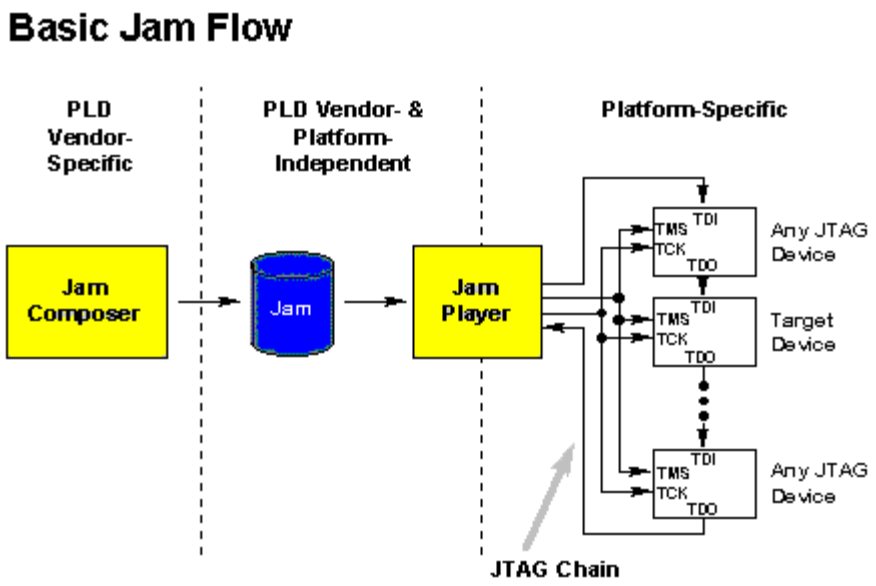

Figure 3. Basic Jam programming flow

There are a few issues with Jam:

- Jam is a relatively new (proposed to JEDEC in September 1997) format. There may be changes to the language before it is adopted as an industry standard that makes developing a Jam "player" uncertain at best.
- A vendor may use Jam's ability to make decisions about the ISP programming algorithm based upon information that is read back from an individual ISP device. An example would be the programming pulse width. If the ATE system does not have sophisticated branching capabilities, this may not be possible without fixture electronics. Currently, the Agilent 3070 can only do this branching within BT-Basic based upon information passed back to it from the digital test. This will slow the overall programming time.
- At this time, the Agilent 3070 would not know what to do with a Jam file as the only thing it understands is a compiled version of VCL/PCF. To program an ISP device, the Agilent 3070 basically executes a series of test vectors fed to it from the executable object file of a digital test. What would need to be developed is a Jam-to-PCF converter program by an ISP vendor or a third party software supplier.

WHAT ABOUT IEEE-1532?

There is a new IEEE standard which is the IEEE-1532 standard. The title of this standard is "Standard for Boundary-Scan-based In System Configuration of Programmable Devices". This is relatively new and at this writing, the hardware portion of the standard has been accepted and the software portion is in the final review and will be up for ballot shortly.

The purpose of the standard is to have a common programming interface for a device that is configurable via the IEEE-1149.1 boundary scan interface. As you can see from information within this document, there are several different file formats used to configure

ISP type devices. Furthermore, the programming algorithms within the devices vary significantly and as a result, complicate the configuring of these devices. A standard that most semiconductor vendors adhere to will result in easier in-system configuration of devices and multi-vendor concurrent programming.

IEEE-1532 describes some hardware characteristics as well as the software interface to configure the device. The software interface consists of two files: an extended BSDL file and a configuration data file. The extended BSDL will have In-System Configuration (ISC) extensions that describe the algorithms used to erase, program, and verify a device. The data file contains the configuration data used during the program and verify phases of the programming process.

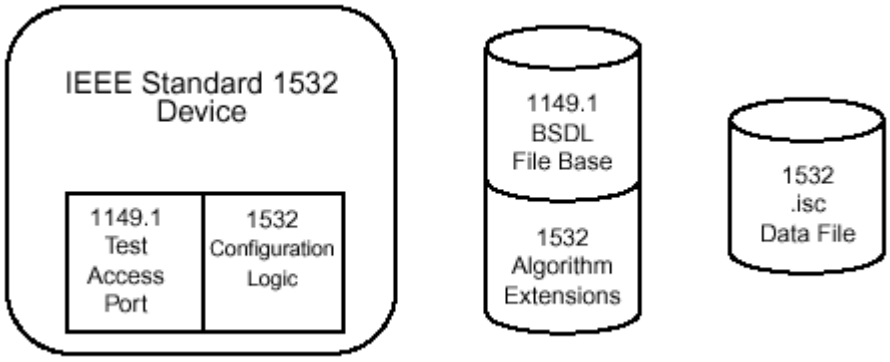

Figure 4. IEEE-1532 Standard components

The status of the IEEE-1532 standard with regard to the Agilent 3070 is that we are watching the evolution of this standard very closely. Agilent has a representative on the IEEE-1532 committee. Since the standard is not fully ratified and may require semiconductor vendors to change silicon to implement the hardware portion of the standard, we feel that a solution at this point may not be the best use of our resources. As the standard develops, we will respond accordingly.

CONCLUSION

While the steps listed above may seem overwhelming at first, it is possible to program ISP devices with the Agilent 3070 test system with the proper planning. Millions of vectors will be applied to the ISP device in order to program it. Getting reliable and repeatable programming of ISP devices has been accomplished by customers using the above steps. It is to be hoped that this white paper has taken the “mystique” out of this development effort on the Agilent 3070 series of test systems.

APPENDIX A: GLOSSARY

ASIC **A**pplication-**S**pecific **I**ntegrated **C**ircuit. A custom digital device that is manufactured at a semiconductor "foundry". An ASIC is not programmable.

BSDL **B**oundary **S**can **D**escription **L**anguage. A language that describes a boundary scan device's internal configuration (inputs, outputs, register length, recognized boundary scan commands, etc.).

Beat Rate The rate at which a PCB manufacturing line can build boards (i.e., the number of boards per hour or shift, for example).

CPLD **C**omplex **P**rogrammable **L**ogic **D**evice. High gate-count device that is programmable (via a device programmer or otherwise).

ESD **E**lectro**S**tatic **D**ischarge. This can lead to internal damage of a device.

EEPROM **E**lectrically-**E**rasable **P**rogrammable **R**ead-**O**nly **M**emory. A EPROM that can be erased and re-programmed by going through an algorithm rather than be placed under a UV light source for erasing and then on a device programmer for programming.

FLASH The term for EEPROM technology related to a memory device. An ISP device is not a memory device, although its internal design may contain memory elements. In fact, many ISP devices use similar EE technology to the FLASH device.

JAM A new file format describing programming information for programmable devices. This has been proposed by several semiconductor vendors.

JEDEC **J**oint **E**lectronic **D**evice **E**ngineering **C**ommittee. A committee overseeing standards in the electronic industry. Commonly refers to a file format used to describe programming information for CPLDs.

JTAG **J**oint **T**est **A**ction **G**roup. A committee formed in the 1980s that developed the Boundary Scan testing specification IEEE-1149.1.

ISP **I**n-**S**ystem **P**rogrammability. The ability of a device to be programmed while mounted on a Printed Circuit Board (PCB). The device may even be programmed by an on-board processor.

ISR **I**n-**S**ystem **R**eprogramable. See ISP.

OBP **O**n-**B**oard **P**rogrammable. Phrase referring to the ability of a device to be programmed on the board. This includes FLASH, EEPROMS, and ISP devices.

PCB **P**rinted **C**ircuit **B**oard. A copper-clad board that has been etched away to leave traces and mounting pads for components.

PCF **P**attern **C**apture **F**ormat. An extension to VCL to represent test vectors as a series of ones and zeros. Useful for creating test vectors from simulator output files.

PLD **P**rogrammable **L**ogic **D**evice. Low gate-count device that is programmable (via a device programmer or otherwise).

SVF **S**erial **V**ector **F**ormat. This file type is a standard for interfacing to boundary scan devices.

VCL **V**ector **C**ontrol **L**anguage. The programming language for the Agilent 3070 to test digital devices.

APPENDIX B:

References

Connor, Doug, “In-System Programmable Logic Simplifies Prototyping to Production”, EDN Magazine, September 26, 1996, page 37.

Talen, Gerald, “Use Boundary Scan to Test Low-Cost ISP devices”, Test & Measurement World, September, 1997, page 73.

Boutin,Matt, Bonnett,Dave, “Program ICs in your system via IEEE 1149.1 and enjoy the benefits throughout the system’s life”, EDN Magazine, November 20, 1997, page 131.

Abramovici M, Lee E, Stroud C, Underwood M, “Self-test for FPGAs and CPLDs requires no overhead”, EDN Magazine, November 6, 1997, page 121.

ISP Semiconductor Vendor Web Addresses

VENDOR	Web Address	Software Tool	Output File
Altera	www.altera.com	Max+Plus II	SVF file
AMD Mach (Vantis)	www.lattice.com	ispDCD	PCF file
Atmel	www.atmel.com	Atmel ISP	SVF file
Cypress	www.cypress.com	Warp2, JamISR	Jam file
Lattice	www.latticesemi.com	LatticePRO,ispDCD	PCF file
Xilinx	www.xilinx.com	JTAG Programmer	SVF file

The following companies provide information useful for ISP programming and testing:

Asset InterTech, Inc. www.asset-intertech.com	Provider of Boundary Scan diagnostic hardware and software solutions. Also provides SVF specification.
AcuGen Software, Inc www.acugen.com	Provides programmable device test vector generator software for the Agilent 3070. They also have a product (PROGBSDL) to create a programmed version of the BSDL file for the ISP device.
Flynn Systems Corp. www.flynn.com	Provides programmable device test vector generator software for the Agilent 3070.