Agilent Technologies

PLD Programming

on the

Agilent Technologies 3070

using the

PLD ISP Product

OVERVIEW

Late in 2001, Agilent Technologies release two new products for the Agilent 3070 to help address the issues around the programming of devices. These products are focused at Flash memory programming (Flash ISP) and Programmable Logic Device (PLD) programming (PLD ISP). This white paper is designed to describe the aspects of PLD programming on the Agilent 3070 test system using the PLD ISP product.

PLD BACKGROUND

In-System Programmable PLDs 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 PLD 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 PLD 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. Consequently, you can reduce scrap costs associated with coplanarity and ESD damage to the devices, not to mention the PCBs the devices are mounted on.
- PLD devices can provide an easy means for engineering changes or feature set enhancements to a product. With PLD 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 PLD PROGRAMMING

Because PLDs are programmable after the device is placed on the board, it is being requested that the PLD devices be programmed at the test process step. In the traditional board manufacturing process, PLDs were pre-programmed by an off-line device programmer and placed on the board during assembly. The test engineer would then develop a device test for the programmed PLD. This may be just a TestJet test or a digital in-circuit test using specific test vectors generated for the device using a third party tool.

Now that PLD devices provide the capability of being re-programmed on the board, the test engineer is being asked to not only test the device, but to program it as well.

Regardless of the ATE system being used, you must, as a test engineer, be aware of the trade-offs related to programming PLD devices on the ATE system. To program a PLD, you send a configuration bit stream to the device via the Boundary Scan interface and wait a small amount of time for the devices to program itself internally. This process is repeated until the PLD is completely programmed. PLDs traditionally require a relatively long time to program. This is due to the physics of the PLD and the algorithm used to program it. The time to program an individual device may be in the tens of seconds. 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 PLD devices on it. The ideal solution would be to have the PLD device's programming algorithm go at "databook" speed (TCK rate at 10Mhz or higher) for the fastest possible throughput. However, the TCK frequency is not the limiting factor for throughput as it is minor compared with the time for the PLD to program itself after the bit stream has been sent.

Parallel (concurrent) programming of PLDs can reduce the overall programming time. Most PLD vendors support "concurrent" programming, although some do not support it on an ATE system. This is the technique for programming multiple 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. This entails getting the design file from the engineer, converting it to a bit stream file (SVF, Jam, or STAPL), and re-compiling the PLD ISP test.

WHAT STRATEGY SHOULD I USE TO PROGRAM THE PLD DEVICES?

The answer to this question depends on your manufacturing process requirements. A few scenarios are shown below:

Fastest 3070 Programming Throughput:

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 PLD devices (particularly on the IEEE-1149.1 interface pins).
2. Program the devices (from the same vendor) in concurrent (parallel) programming mode if possible. This will minimize the time to program the devices. The tradeoff is that if a device in the chain fails to program, you will not know which device it is as the programming for the entire chain failed. Also, note that some vendors do not support concurrent programming on an ATE system.

Note that if your board contains different manufacturer's PLD 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 PLD programming times.

Diagnostics Most Important:

If manufacturing line "beat rate" considerations allow, greater control of the individual PLD 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 PLD 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, the test will fail at the time the device is being programmed and indict the PLD as being bad. The trade-off here is that is takes more time than programming devices "concurrently".

Off-line PLD Programming:

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 PLD 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 PLD devices at the Agilent 3070 process step.
2. Utilize an off-line process step featuring a power supply and a PC with an PLD download cable attached. The PC can then program the board without tying-up the Agilent 3070 system.

Additional Considerations:

You may want to consider creating a TAP integrity test for the boundary scan chain that could be run before 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 PLD 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.

PLD ISP FEATURES

There are several new features with the PLD ISP product. If you have tried to program PLDs on the 3070 in the past, you had to create and manage many files. The procedure, while possible to do, was tedious and error prone. The new PLD ISP product eliminates much of these issues. Table 1 shown below is a list of features and benefits for the test developer:

Feature	Benefit
Only a single VCL file plus the configuration data file are required.	• Eliminates the need for many VCL/PCF files. • No pull-up/pull-down resistors required for the fixture. • This reduces the overall compile time significantly.
Support for SVF directly.	No conversion from SVF to many PCF/VCL files.
Support for Jam, STAPL, and JBC directly.	• We support multiple data file formats directly. • We can now program Altera “non-F” PLDs.
We have a “JBC” player in the testhead.	Because of the architecture on the XTP card, we will be able to implement new standard players relatively easily (like IEEE-1532).
Support for print statements in Jam/STAPL files – output to BT-Basic window.	This provides a good diagnostic tool for debug of the PLD programming tests.

Table 1. PLD ISP feature/benefits.

The new PLD ISP product allows you to specify a configuration bit stream file in the VCL source code (much like programming a Flash memory device). In addition, multiple data formats are supported: Serial Vector Format (SVF), Standard Test And Programming Language (STAPL), Jam, and Jam Byte Code (JBC). We have implemented a JBC player in the Control XTP card that allows us to play “JBC” files. Because of this, SVF, STAPL, and Jam files must be compiled into a JBC object file format. In addition, SVF files must first be converted to a STAPL file format before it is compiled.

We are using some new hardware features on the Control XTP card to allow us to quickly apply serial data through the 3070 Hybrid card resources to the device under test (DUT). Also, the JBC player is executed via the microprocessor on the Control XTP card which allows us to apply vectors algorithmically (i.e., conditional branching) rather than execute a sequence of vectors. This is how we are able to program Altera PLDs that are not of a fixed programming and erase pulse width. The JBC player reads the programming and erase pulse width registers from the Altera device and uses those values in the programming and erase algorithms.

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

Now for the important information. 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 white paper.

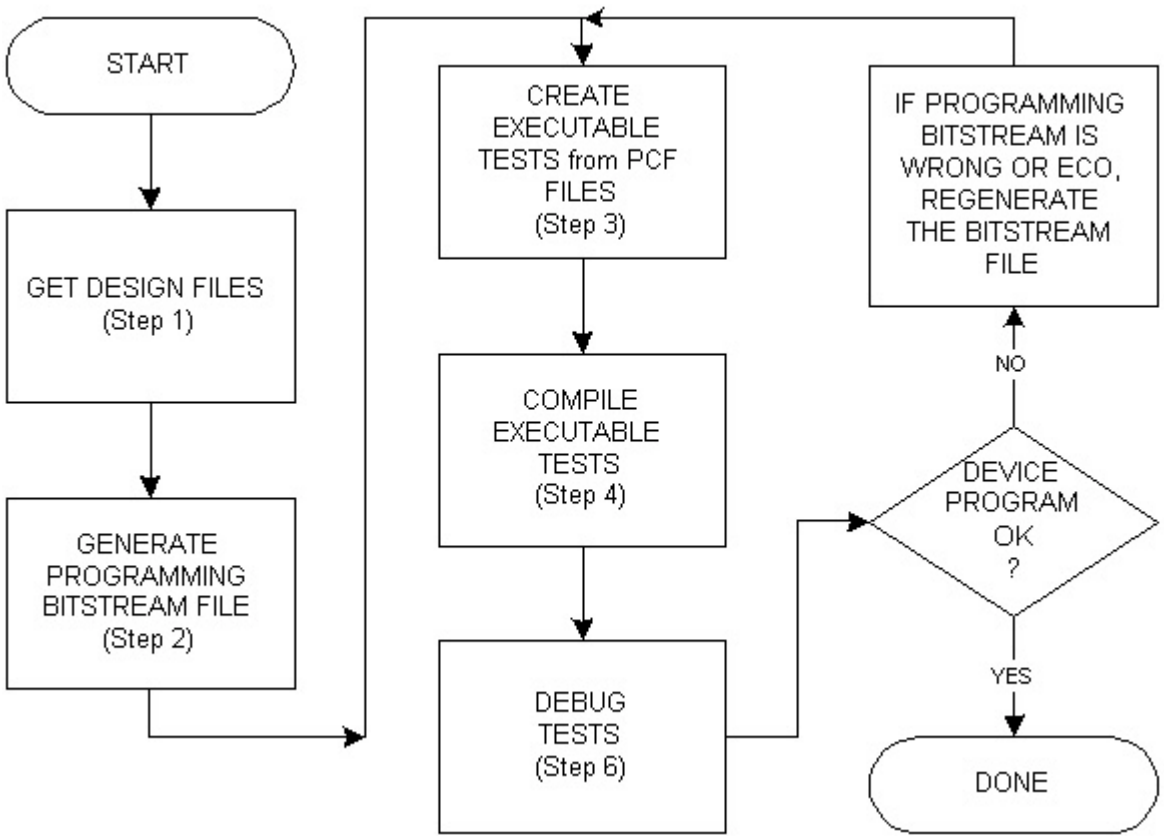


FIGURE 1. Flow for Developing PLD Programming on Agilent 3070

1. GET THE DESIGN FROM THE DESIGN ENGINEER

To create a bit stream data file used to program a PLD device, you must start with the design file from the engineer for each device. This may be in a JEDEC format or some other proprietary format from the PLD manufacturer. If you have only one PLD device, then you will need only one design file. Note that you may need to transfer this file to and from a PC to do some of the later steps. Therefore, you should be familiar with using a file transfer program (i.e., FTP) between the PC and the Agilent 3070 workstation.

You also have to work closely with the design engineer as the design software may be tightly integrated with the generation of the programming files (i.e., the programming file represents the entire chain or just the one device). In addition, a couple of iterations of generating the programming information may be necessary before the test can be released to manufacturing. You will also need the design engineer's help in verifying that the PLD devices are indeed programmed correctly by the 3070.

2. GENERATE THE PLD PROGRAMMING BIT STREAM FILE

Use a **vendor-supplied** tool to create the PLD programming bit stream. Note that most PLD vendors supply this tool free of charge while others incorporate it into the design software used to develop the device configuration. Many times, this same tool is used to download the programming information directly to the device via some adapter connected to the PC or workstation. Check with your PLD device vendor for detailed information on how to create this bit stream file. Development and device programming software to create SVF/Jam/STAPL file can be downloaded from the following websites:

Vendor	Website	Tool name
Altera	http://www.altera.com	Max+II Stand-Alone Programmer 10.0
Xilinx	http://www.xilinx.com	JTAG Programmer WebPack 2.1
Lattice	http://www.laticesemi.com	ispVM 9.05b
Cypress	http://www.cypress.com	ISR 3.0.7
Atmel	http://www.atmel.com	ATMISP 3.0.1

Table 2. PLD vendors and websites.

While we support multiple bit stream file formats, we must convert an SVF file to STAPL prior to compiling the STAPL file to a JBC object file. This will take a few seconds during the PLD ISP test compile. In general, if you can generate different file formats from the vendor-supplied tool, target the STAPL or Jam format. If you have a choice of Jam, JBC, or STAPL file versions, always choose the latest revision (e.g., choose Jam 1.1 over Jam 1.0).

Sometimes you will have a chain where multiple PLD devices are connected and you wish to take advantage of the concurrent programming algorithm. Usually, the bit stream file created by the tool will contain the necessary bits to program all devices at once provided you described the boundary scan chain to the tool properly. For more information on concurrent programming, contact the vendor directly.

Most bit stream generation software will allow you to create 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 PLD 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 length of the instruction register 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 PLD 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 PLD devices, which could fail during programming, you will need to generate several bitstream files - one for every PLD device in the chain. You would describe the chain as one PLD device with everything else in the chain placed in BYPASS mode (including other PLD devices) and generate your bitstream file. Then you would describe the chain again with the next PLD device in the chain as your target and everything else including the first PLD device in BYPASS mode and generate the next bitstream file. Repeat this process until all PLD devices have a corresponding bitstream file.

The bit stream generation tool from the vendor generally allows you to control the actions done to the chain. That means you can create a bitstream file that does an erase-program-verify, or just an erase, or just a verify. You need to be aware that each tool is different and you should try to become familiar with it to understand what its capabilities are. Make sure you generate a bit stream file that matches what you are trying to do.

Also, be aware that you might have to control a pin on the PLD to place the device into a IEEE-1149.1 compliant state. This should be noted on the PLD datasheet as well as in the BSDL file for that device. This pin can be added to the test and a default state can be assigned to the pin to enable the IEEE-1149.1 compliance mode.

3. 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 PLD devices. The process has six major parts: create library for PLD device or scan chain, enter the tests in Board Consultant, run Test Consultant, modify the executable tests, generate wirelist information for the tests, and modify the testplan.

A. Create PLD ISP node library test

The initial program development for the board should contain a "setup-only" node library test for the PLD boundary scan chain interface. This will ensure that the Agilent 3070 tester resources are reserved in the test fixture for programming the PLD devices. If your PLD 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).

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.

pld isp                          ! Specify the PLD ISP test

vector cycle 160n                ! Vector cycle for a 6Mhz system
receive delay 120n

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.

family  FLASH_3V3           ! Specify a logic family for the pins.

inputs  TCK, TMS, TDI
outputs TDO
```

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

In addition, you can define the "`scan tap`" and "`program`" blocks in the library test. An example of these is shown below:

```
scan tap
   tck    TCK   ! Associate the player's TCLK resource with a group
   tdi    TDI   ! Associate the player's TDI resource with a group
   tdo    TDO   ! Associate the player's TDO resource with a group
   tms    TMS   ! Associate the player's TMS resource with a group
end scan tap
```

If you want, you can even add the “program” block and the execution section to the test. The library compile will fail unless these blocks are commented as shown below:

```
!program Program_PLDs
!   file "PLD_file.jam" stapl  ! name of bit file and file format
!   program time 60 sec        ! Timeout limit for test (optional)
!end program

!unit "Main"
   ! execute My_Vector
!   play Program_PLDs action "PROGRAM"  ! Execute the player.
   ! play Program_PLDS action "VERIFY"
!end unit
```

The “`program time`” statement is optional. It specifies the maximum time for the test to execute before being halted. This will halt the test if the JBC player if it is caught in an infinite loop.

Note that only STAPL files have actions. Other file formats (Jam, SVF, and JBC) do not use the “`action`” keyword on the “`play`” statement. You must examine the STAPL file manually to see what actions are available for a device. An excerpt from the top of a STAPL file is shown below:

```
  .  .  .
NOTE "STAPL_VERSION" "JESD71";
NOTE "JAM_VERSION" "2.0";
NOTE "ALG_VERSION" "10";
ACTION PROGRAM = PR_INIT_PROGRAM, DO_BLANK_CHECK OPTIONAL, DO_VERIFY
RECOMMENDED, DO_SECURE OPTIONAL,
DO_READ_USERCODE OPTIONAL,
PR_EXECUTE;
ACTION BLANKCHECK = PR_INIT_BLANKCHECK, PR_EXECUTE;
ACTION VERIFY = PR_INIT_VERIFY, PR_EXECUTE;
ACTION READ_USERCODE = PR_INIT_READ_USERCODE, PR_EXECUTE;
DATA DEVICE_DATA;
INTEGER V0 = 4;
INTEGER V1 = 3;
```

Notice the “ACTION” statements that define what actions are available for the STAPL file. In this case, you have the following four actions available: PROGRAM, BLANKCHECK, VERIFY, and READ_USERCODE.

After the PLD ISP node library test has been created, you can compile it with the library option. Correct any compile errors before continuing.

B. Use Board Consultant to add the PLD programming tests.

Use Board Consultant to create “Node Library” tests for the PLDs you want to program. You have a couple of strategies here:

- If the PLDs are not part of a chain, then you should create a node library for each PLD you want to program. Note that there should be a library for each device that is not part of a chain.
- If there are multiple PLDs in the chain that can be programmed concurrently, create one node library for the entire chain.

- If there is a mixture of vendors for the PLDs in the chain, create a node library for each vendor's PLDs all pointing to the same library you created in step A above.

An example of the Node Library entry form is shown below in figure 2.

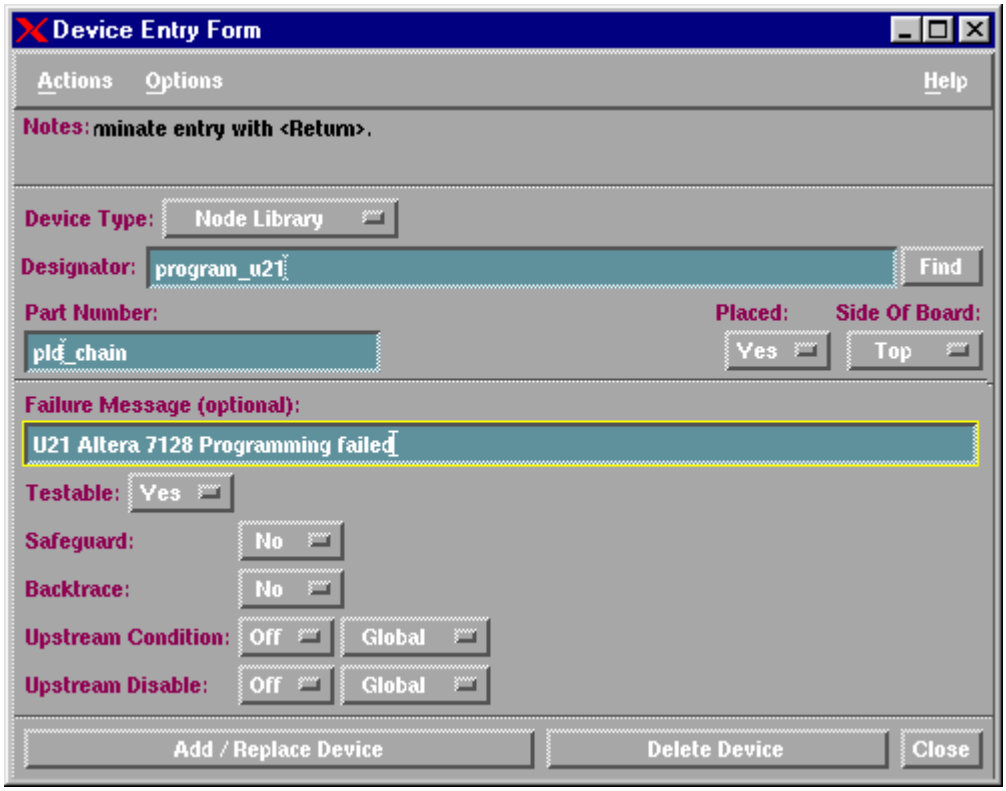


Figure 2. Node Library Entry form.

The PLD ISP software is very flexible. You can have multiple "program" blocks pointing to different data files for a single chain and can execute multiple "play" statements to program all devices in that chain. However, the test would be difficult to debug as there are many variables that could affect the programming operation. It is probably best to limit the node library to a single device or a group of devices that can be programmed concurrently.

As an example, let's say that we have two PLD devices in our chain. One is an Altera EPM7128S (U21) and the other is a Xilinx XC9536 (U43). Since these PLDs are from two different vendors, we should create two node library entries in Board Consultant. I would name the tests "`program_u21`" and "`program_u43`" respectively. Note that both node library tests point to the same library since both are in the same boundary scan chain.

C. 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 with the correct fixture wiring resource information. This file will be used as a template to create the source code of the executable test.

D. Modify the tests to program the PLD device.

You can now modify the executable tests to point to the correct data file(s) and execute the corresponding actions. For example, if we use the scenario above with an Altera and

Xilinx device in the chain, we would get two tests created by Test Consultant. These would be "u21_program" and "u43_program" respectively. You would modify each one to point to the correct bitstream file. Note that Xilinx generally outputs in SVF format, so you would need to make sure the file type is "svf" and that no actions are defined (i.e., just use "play" statement with no "action" parameter).

If the tests were not generated by Test Consultant, you should add these test names to your "testorder" file and mark them "permanent" using the following syntax:

```
test digital "u21_program";  permanent
test digital "u43_program";  permanent
```

E. Create wirelist information for the tests.

Note that you do not need to do this step if the tests were generated by Test Consultant. If you are adding these tests to an already existing board, make sure the executable tests are listed in the "`testorder`" file. An example of this entry in the "`testorder`" file is shown below:

```
digital "u21_program"; comment
digital "u43_program"; comment
```

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.

F. Modify the testplan.

The last thing to do is to add the test statements to the "testplan" using the following syntax:

```
test  "digital/u21_program"     ! program the Altera EPM7128S
test  "digital/u43_program"     ! program the Xilinx XC9536
```

If the tests were created by Test Consultant, there will already be entries in the testplan for these tests. They will be located in the Digital subroutine.

If you want, you can create a separate subroutine to program all PLDs on the board. It might look something like this:

```
sub Program_PLDs
   test "digital/u21_program"
   test "digital/u43_program"
subend
```

In general, the call to this subroutine should be in the "Test_Sections" subroutine in the testplan. Obviously, you should place this call after the power is applied to the board. If you want to program the PLDs after all testing is done, it will probably look like this code from the "Test_Sections" subroutine:

```
print tab(5);Mode$;DigitalMsg$
Status = Failed_In_Digital
call Digital_Tests
if boardfailed then subexit

print tab(5);Mode$;DigitalFuncMsg$
```

```
Status = Failed_In_Functional
call Functional_Tests
if boardfailed then subexit

print tab(5);Mode$;AnalogPoweredMsg$
Status = Failed_In_Functional
call Analog_Functional_Tests
if boardfailed then subexit

PLD_ProgMsg$ = "Programming PLDs"
print tab(5);Mode$;PLD_ProgMsg$  ! New section
Status = Failed_In_Digital       !  added for
call Program_PLDs                !  PLD
if boardfailed then subexit      !  programming.
```

Note that you could create a new status code named "`Failed_In_PLD`" in the "`Initialize_Constants`" subroutine and set it equal to eight (the same as the "`Failed_In_Digital`" status). You must remember to make it "global" in the subroutines that need the "`Failed_In_*`" variable.

5. COMPILE THE EXECUTABLE TESTS

Once the PLD programming executable tests have been modified, remember to compile them before running them or doing debug. You could compile them with the "debug" option so that any tests requiring debug will have the debug object file there already. After completing this step, you can try the tests on the testhead.

A NOTE ABOUT COMPILING PLD ISP TESTS

The compile process for PLD ISP tests is shown in figure 3 below:

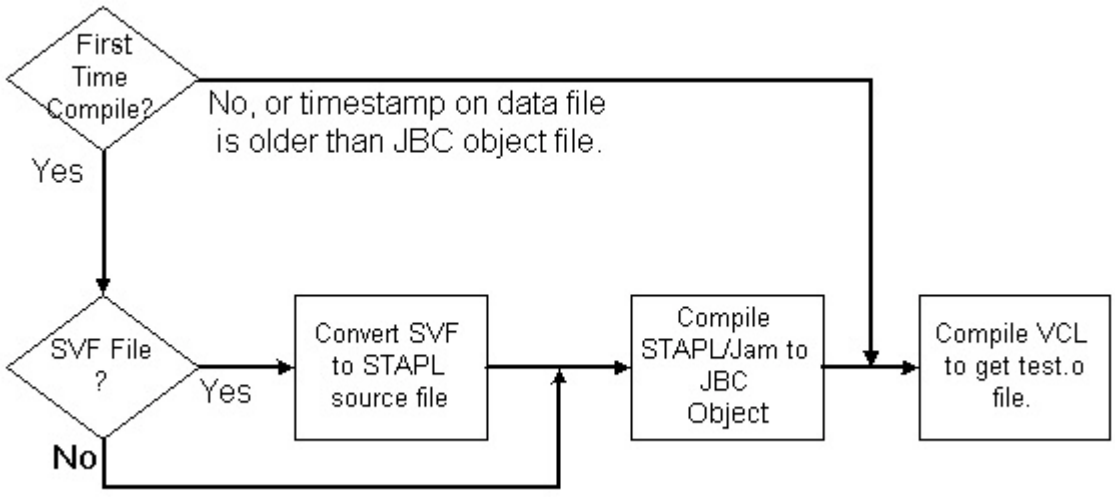


Figure 3. PLD ISP Compile flow

Note that unless the source data file has a newer timestamp than the JBC object file, then the process to compile the SVF/STAPL/Jam file to the JBC object file is omitted. This is important to understand because if you need to change something to the VCL file for debug purposes (like vector cycle, drive and receive levels, etc) you will not incur the compile hit to create the JBC object file. This could save several minutes of time during the debug of the test.

7. DEBUG THE TEST

Now that we have our executable tests and test fixture, we can go to the 3070 test system to debug the test. The bitstream will show up on the TDO pin of the boundary scan chain. If the bitstream coming from the device does not 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 PLD programming either works or it does not work at all.

Keep in mind that not all of the features of Push-Button debug will be available to PLD ISP tests. In particular, the graphical display of the pin states will not reflect what is going on during the playing of the JBC file. This is because the sequencer (which has the hooks for capturing the pin states) is not controlling the testhead when the player is running. Furthermore, even if you had the pin states to look at, what value would it be? Programming PLDs requires millions of vectors to be applied to the Boundary Scan interface. If one bit is wrong somewhere in the bitstream, the programming sequence could fail. It is very difficult to correlate the SVF/STAPL/Jam statements to vectors applied to the PLD.

Here are some of the things to look for if the device fails to program:

- Try varying the vector cycle/receive delay in the debug environment. In general, try to maximize the application rate for the vectors. This translates into a faster TCK rate and results in faster throughput.
- Also, try varying the drive and receive levels on the boundary scan pins. You may have to optimize these voltages to prevent excessive ringing and possible double clocking of the TCK pin.
- If you have access, use an oscilloscope the view the Boundary Scan signals. TCK is especially important as any noise or glitches will cause the IEEE-1149.1 TAP state machine to transition to a state that the bitstream is not expecting.
- If you can, create a boundary scan integrity test for the chain. This verifies the correct device is placed on the board at the right position in the chain and that there is no problem propagating signals through the entire chain. This integrity test is automatically generated if you have the Agilent Interconnect Boundary Scan software license on your system.
- Many times, the order of the chain was defined incorrectly in the vendor tool that creates the bitstream file. Double-check that the chain order defined in the tool matches the order of the devices on the board. Remember, devices that are not being programmed are generally placed in “BYPASS” mode. Make sure that the number of bits for the instruction register are specified correctly for non-PLD devices and PLD 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.
- 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.
- The bit stream files created by the vendor’s software generally does a check to see that the targeted PLD device has the correct ID code. If this portion of the test fails,

check that the boundary scan chain you defined in the software tool matches what is on the board.

- STAPL and Jam support the use of "print" statements. The outputs of the print statements come to the BT-Basic window as the test is executed. You can use additional "print" statements in the STAPL or Jam file to show where the file is during execution. You can indicate what step it is doing. This pertains to STAPL or Jam files only. However, remember that SVF files are converted to a STAPL file before it is compiled to a JBC object file. You can edit this STAPL file to add "print" statements to help describe what the test is doing. For more information, see the JESD-71 programming standard available at http://www.jedec.org/download. The STAPL language is described in detail in the document.
- Altera has a useful STAPL file that will check the boundary scan chain and read the device ID for every device in the chain. It is available for download at http://www.altera.com/products/devices/common/dev-jam_download.html. You can create a simple test for your chain pointing to this STAPL file.
- Note that you can use the "print level pld isp is all" statement in BT-Basic to add more information to the output from the test. Any "NOTE" statements will be displayed in the BT-Basic window as well as the output of "PRINT" statements. This pertains to STAPL or Jam files only. However, remember that SVF files are converted to a STAPL file before it is compiled to a JBC object file. You can edit this STAPL file to add "print" statements to help describe what the test is doing.

ISSUES TO BE AWARE OF

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

- Be careful if you use a pin library to describe your PLD device in a stand-alone boundary scan chain. **It is not recommended to describe all of the PLD 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**.
- 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 PLD 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".
- Be aware that if a PLD vendor changes, their manufacturing process for PLD devices, you may see failures when programming the device. The internal programming characteristics may change along with the device revision code. It may be useful to be notified of any changes to the PLD by the PLD vendor.

PLD ECO PROCESS

If there is an Engineering Change (ECO) to the PLD design, you must do the following process to implement the new design on the board:

1. Create a new bit stream file for the design by using the vendor's tool as described in step 2 above.
2. Modify the PLD ISP test to point to this new bit stream file if it's file name is different than the previous bit stream file.
3. Compile the PLD ISP test. This will create a new JBC object file and PLD ISP test object file.
4. Debug the new version of the test to make sure it works correctly. If there is a problem, check to see that the bit stream file was created correctly.

You should now be done implementing the PLD ECO.

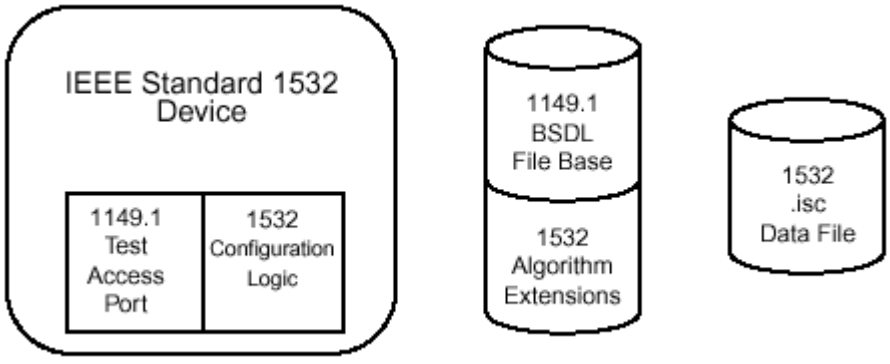
WHAT ABOUT IEEE-1532?

There is a new PLD programming standard which is known as 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 PLD vendors are migrating their devices to support this standard for programming.

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 PLD 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 concerning the Agilent 3070 is that we are watching the evolution of this standard very closely. Agilent has a representative on the IEEE-1532 committee. We are planning to support this standard in a future release of PLD ISP software.

ADVANCED TOPIC: STAPL Player Execution Control

The STAPL (JESD71) standard defines a powerful language that provides many features over SVF:

- True algorithmic language (i.e., conditional branching).
- Execution status output via "PRINT" statements.
- Selectable execution by "ACTION" names.

In addition to these features, "PROCEDURE" blocks defined in the ACTION statements can be controlled through parameters passed from BT-Basic. To understand this better, let's look at an example "ACTION" statement as shown below:

```
ACTION PROGRAM = PR_INIT_PROGRAM, DO_BLANK_CHECK OPTIONAL,
   DO_VERIFY RECOMMENDED, DO_SECURE OPTIONAL,
   DO_READ_USERCODE OPTIONAL,
   PR_EXECUTE;
```

Figure 5. Example "ACTION" statement from a STAPL file.

The syntax shown above is listing the "PROCEDURE" blocks that are called by the "ACTION". In this case, the PROGRAM action will call the following procedure blocks in this order:

- `PR_INIT_PROGRAM`
- `DO_BLANK_CHECK` (optional)
- `DO_VERIFY` (recommended)
- `DO_SECURE` (optional)
- `DO_READ_USERCODE` (optional)
- `PR_EXECUTE`

Procedure blocks that do not have the "OPTIONAL" or "RECOMMENDED" keyword will always be executed. If no parameters are passed to the STAPL player, any "recommended" procedures will also be executed while the "optional" procedures will not be executed. The "optional" and "recommended" flags on procedures can be overridden by parameters passed to the STAPL player.

Using the example ACTION shown in figure 4, we can have the STAPL player execute the "`DO_BLANK_CHECK`" procedure by setting the procedure block name "`DO_BLANK_CHECK`" to a "1" and passing that as a parameter to the player. Likewise, we can prevent the "`DO_VERIFY`" procedure from running by setting the "`DO_VERIFY`" procedure block name to a "0" and passing it to the player.

An example of how this would look in BT-Basic is shown below:

```
! Enable a Blank Check and Disable a Verify in STAPL.
STAPL$ = "BLANK_CHECK=1,VERIFY=0"
```

```
test "digital/program_u21"; STAPL$
```

The corresponding VCL code for this looks like this:

```
!!!!    6    0    1 1010706036  V4f00
! IPG: rev 05.00pb Fri Feb 22 17:55:12 2002
!
! Program the Altera U21 CPLD chain

pld isp

vector cycle  160n
receive delay 120n

default device "program_u21"

test digital; Symbol_List$

set terminators to on
assign Altera_TDI  to nodes "$$__TN93.1"
assign Altera_TDO  to nodes "$$__R13.1"
assign Altera_TMS  to nodes "$$__TN92.1"
assign Altera_TCK  to nodes "$$__TN91.1"

family FLASH_3V3

inputs Altera_TDI, Altera_TCK, Altera_TMS
outputs Altera_TDO

!IPG: Add defaults or loads to floating inputs and bidirs.
assign Floating__Inputs to nodes "$$__TN91.1","$$__TN92.1" default "00"
assign Floating__Inputs to nodes "$$__TN93.1" default "0"
inputs Floating__Inputs

scan tap
  tck    Altera_TCK
  tdi    Altera_TDI
  tdo    Altera_TDO
  tms    Altera_TMS
end scan tap

program Program_U21
   file "u21_stapl.jam" stapl  ! name of file and source identifier
   program time 3600 sec  ! Timeout limit for running test
end program

unit "Main"
   ! Start the player and pass the BT-Basic parameter to it.
   play Program_U21 action "PROGRAM"; import symbols Symbol_list$
end unit
```

This shows the keyword "`import symbols`" on the "play" statement. This takes the string passed in from BT-Basic and defined by "`Symbol_List$`" and passes it to the STAPL player. Since this string contains "`BLANK_CHECK=1,VERIFY=0`", the STAPL player will execute the optional procedure "DO_BLANK_CHECK" and not execute the recommended procedure "DO_VERIFY".

Note that this feature is not something you will take advantage of frequently, but is available if you would like to provide the extra control over what the STAPL player is doing. You might want to see if the device to be programmed was blank when you tried to program it. If it was not, a PRINT statement will be issued saying that the device was not blank when it was programmed. This information might indicate a quality problem with the vendor of the device.

This feature would not be useful for devices that are being programmed with an SVF file. Remember that SVF files are first converted to a STAPL file and this file is compiled to a JBC object file. The ACTION statement from a STAPL file that was converted from and SVF file is shown below:

```
ACTION RUN_FILE "Execute Converted Vectors" = EXECUTE;
```

The STAPL file that is created from SVF only has one action (`RUN_FILE`) and one procedure block (`EXECUTE`). SVF is linear code to configure a PLD device and the SVF-to-STAPL converter simply translates the SVF syntax to STAPL syntax.

CONCLUSION

The programming of PLDs on the Agilent 3070 is now possible to do easily by using the PLD ISP product. Agilent is committed to providing product enhancements targeted at ease-of-use issues and product capabilities. Hopefully, you have a better understanding as to how to program PLDs on the 3070 using the features in PLD ISP.

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 PLD device is not a memory device, although its internal design may contain memory elements. In fact, many PLD 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 PLD.

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 PLD 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).

STAPL **S**tandard **T**est **A**nd **P**rogramming **L**anguage. This is a new standard from the JEDEC organization to define a language primarily for the programming of PLDs. It is also know as the JESD-71 standard.

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 PLD 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.

PLD Semiconductor Vendor Web Addresses

VENDOR	Web Address	Software Tool	Output File
Altera	www.altera.com	Max+Plus II	Jam,STAPL,JBC,SVF
AMD Mach (Vantis)	www.latticesemi.com	ispVM 9.05b	SVF
Atmel	www.atmel.com	Atmel PLD	SVF
Cypress	www.cypress.com	Warp2, JamISR	Jam
Lattice	www.latticesemi.com	ispVM 9.05b	SVF
Xilinx	www.xilinx.com	JTAG Programmer WebPack 2.1	SVF, IEEE-1532

The following companies provide information useful for PLD 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 PLD device.
Flynn Systems Corp. www.flynn.com	Provides programmable device test vector generator software for the Agilent 3070.
Altera Corp. www.jamisp.com	Information about Jam and STAPL standards.
Xilinx Corporation www.xilinx.com	Information about IEEE-1532 standard and their J-Drive software to program devices using IEEE-1532.