JTAG and Jam Programming

D. W. Hawkins

12/22/2001

Contents

1  Introduction ................................................................. 3
2  Boundary-scan testing .................................................... 3
3  Altera device programming using JTAG and Max+Plus II ... 6
4  Device programming using Jam .......................................... 6
   4.1 EPC2 programming and erasing .................................... 7
   4.2 FLEX10K programming ............................................. 8
   4.3 Cypress CY37064 programming ................................... 8
   4.4 A final note ......................................................... 8
1 Introduction

The Altera FPGAs, Altera EPC2 EPROMs, and the Cypress CPLD used in the COBRA correlator system all contain JTAG interfaces. The JTAG interface can be used to perform boundary-scan testing (BST) [14] and device programming [4, 6].

2 Boundary-scan testing

In the 1980s the Joint Test Action Group (JTAG) developed a specification for boundary-scan testing that was later standardized as the IEEE Std. 1149.1-1990 specification [13, 14]. This boundary-scan test (BST) architecture offers the capability to efficiently test components on PCBs with tight lead spacing [4, 11].

Boundary-scan testing is used to test pin connections on a PCB without using physical test probes and is used to capture functional data while a device is operating normally. Figure 1 shows the basic concept of BST. Boundary scan cells within a device can force signals onto pins, or capture data from pins or core logic signals. Forced test data is serially shifted into the boundary-scan cells. Captured data is serially shifted out and externally compared to expected results [4].

BST of devices usually requires an external piece of test equipment. This piece of test equipment and the controlling software take as inputs the Boundary-Scan Description Language (BSDL) files for each of the JTAG-capable components, and the PCB netlist. The BSDL file for a device maps the boundary-scan cells within a device to the device I/O pins. The boundary-scan equipment uses the netlist to determine which pins on JTAG-capable devices are connected and then generates test patterns for all pin connections on the board that are accessible over JTAG. The control software then uses the JTAG chain on the PCB to shift in each of the test patterns that are to be driven by the JTAG-capable output pins. The JTAG chain is then used to capture the result on JTAG-capable input pins. The result is shifted out over JTAG, compared to the expected result, and the process repeats for subsequent test patterns. The results of the boundary-scan tests confirm the PCB netlist and determine if errors, such as shorts or open connections, exist on individual PCBs.

BST equipment is very expensive (tens-of-thousands of dollars), so we have not tested any boards using BST equipment. However, the BSDL files are useful in that they contain the device IDs and BST commands. I have used these files to confirm JTAG command bit patterns and device identification (32-bit JTAG defined IDs). Table 1 shows the set of boundary-scan instructions and their bit patterns for the devices used in the correlator system. Table 2 shows the IDCODE responses of the devices used in the correlator system. Table 3 shows additional information obtained from the BSDL files. The JTAG specification requires that the 2 LSBs of the instruction register capture pattern must be equal to 01. When data is shift out after an instruction register capture, then the number of of transitions high and low must be at least as many as the number of devices in the JTAG chain [11, p7].
Table 1: JTAG BST instructions [4, p15], [11, p5]

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FLEX 10K</td>
<td>EPC2</td>
</tr>
<tr>
<td>SAMPLE/PRELOAD</td>
<td>0001010101 (0x055)</td>
<td>0001010101 (0x055)</td>
</tr>
<tr>
<td></td>
<td>Allows a snapshot of the signals at the device pins to be captured and examined during normal device operation, and permits an initial pattern to be output at the device pins.</td>
<td></td>
</tr>
<tr>
<td>EXTEST</td>
<td>0000000000 (0x000)</td>
<td>0000000000 (0x000)</td>
</tr>
<tr>
<td></td>
<td>Allows the external circuitry and board-level interconnections to be tested by forcing a test pattern at the output pins and capturing results at the input pins.</td>
<td></td>
</tr>
<tr>
<td>BYPASS</td>
<td>1111111111 (0x3FF)</td>
<td>1111111111 (0x3FF)</td>
</tr>
<tr>
<td></td>
<td>Places the 1-bit bypass register between the TDI and TDO pins, which allows the BST data to pass synchronously through the selected device to adjacent devices during normal device operation.</td>
<td></td>
</tr>
<tr>
<td>IDCODE</td>
<td>0000000110 (0x006)</td>
<td>0010110001 (0x059)</td>
</tr>
<tr>
<td></td>
<td>Selects the IDCODE register and places it between TDI and TDO for readback.</td>
<td></td>
</tr>
<tr>
<td>UESCODE</td>
<td>0000000111 (0x007)</td>
<td>0011110001 (0x079)</td>
</tr>
<tr>
<td></td>
<td>Selects the user electronic signature (UES) register and places it between TDI and TDO for readout.</td>
<td></td>
</tr>
<tr>
<td>USERCODE</td>
<td>not supported</td>
<td>not supported</td>
</tr>
<tr>
<td></td>
<td>Selects the user code register and places it between TDI and TDO for readout.</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Device 32-bit ID codes (response to IDCODE instruction)

<table>
<thead>
<tr>
<th>Device</th>
<th>Device 32-bit IDCODE</th>
<th>Version (4-bits)</th>
<th>Part Number (16-bits)</th>
<th>Manufacturer (16-bits)</th>
<th>“1”</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLEX10KAB356</td>
<td>0x001000DD</td>
<td>0000</td>
<td>0000000010000000</td>
<td>000011011110</td>
<td>1</td>
</tr>
<tr>
<td>FLEX10KEB356</td>
<td>0x201000DD</td>
<td>0010</td>
<td>0000000100000000</td>
<td>000011011110</td>
<td>1</td>
</tr>
<tr>
<td>EPC2L20</td>
<td>0x010020DD</td>
<td>0000</td>
<td>0001000000000000</td>
<td>000011011110</td>
<td>1</td>
</tr>
<tr>
<td>CY37064-TQFP44</td>
<td>0x8102009</td>
<td>??00</td>
<td>1000000100000000</td>
<td>000001101000</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 3: Additional BST information from the device BSDL files

<table>
<thead>
<tr>
<th>Device</th>
<th>Instruction Register Width</th>
<th>Instruction Capture Code</th>
<th>Boundary-scan length</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLEX10KAB356</td>
<td>10</td>
<td>0101010101</td>
<td>1248</td>
</tr>
<tr>
<td>FLEX10KEB356</td>
<td>10</td>
<td>0101010101</td>
<td>1050</td>
</tr>
<tr>
<td>EPC2L20</td>
<td>10</td>
<td>0101010101</td>
<td>24</td>
</tr>
<tr>
<td>CY37064-TQFP44</td>
<td>6</td>
<td>000001</td>
<td>89</td>
</tr>
</tbody>
</table>

Figure 2: Altera JTAG/BST circuitry [4].
Altera Application Note 39 [4] gives detailed information on the boundary-scan architecture of Altera devices. Figure 2 shows a block diagram of the Altera JTAG/BST circuitry. Cypress semiconductor has a similar application note [11]. In addition to describing the JTAG Cypress circuitry, this application gives an overview of the layout of BSDL files.

3 Altera device programming using JTAG and Max+Plus II

To program devices in a multi-device JTAG chain in Max+Plus II, perform the following procedure:

1. In Max+Plus II, bring up the device programmer.
2. Turn on the check mark for the menu item JTAG → Multi-device JTAG chain.
3. Select JTAG → Multi-device JTAG chain setup
4. Select the .sof FLEX 10K configuration file(s) and/or EPC2 .pof programming file(s).
5. Add the configuration file to the device list (leave the filename blank for devices in the JTAG chain that are to be bypassed). Add enough devices to describe the full JTAG chain.
6. Turn on the target board and click on ‘Detect JTAG chain info’ to confirm the JTAG chain description.
7. Click ok to exit the JTAG setup.
8. On the device programmer; the buttons Program, Verify, Examine, and Blank-check are used for programming EPC2 devices. The button Configure is used to configure FLEX devices. Select the appropriate button to perform the desired action.

4 Device programming using Jam

Altera and Cypress developed an In-System Reconfiguration (ISR) or In-System Programming (ISP) programming language to configure devices over the JTAG chain. This language was named ‘Jam’ (www.jamisp.com). The language has been adopted as the JEDEC Standard Test and Programming Language (STAPL) [12].

Both Altera and Cypress have a number of useful application notes regarding JTAG chain design and the programming of their devices. For more information see the following references

- Altera: [1–6]
- Cypress: [7–10]

The Altera SRAM-based FLEX10K FPGAs and EPC2 configuration EPROMs, and the Cypress EEPROM-based CY37000 CPLDs are all be programmable via JTAG using Jam programming files. The JTAG instructions used to program the devices are propriety and are not available in published application notes. However, these codes do appear in the ASCII versions of the Jam programming files. There is really no need to know the specific instructions, as the Jam player takes care of device programming.

There is one case where knowing the JTAG instruction for an ISP specific task is of use; erasing an EPC2 EEPROM. Once programming using Jam or Max+Plus II, an EPC2 located in a multi-device JTAG chain can not be erased. The device can be reprogrammed, which entails an initial erase operation, however, the device can not just be erased.
4.1 EPC2 programming and erasing

The two EPCs contain the configuration data for the system controller and PCI core FPGAs. To generate the EPC2 programming files, the following procedure is followed:

1. In Max+Plus II, bring up the device programmer.
2. Select File → Convert SRAM object files.
3. Select the system controller or PCI SRAM object file (.sof file) and add it to the list.
4. Select the output file format .pof (sequential).
5. Click on the ‘Output File Options’ button and select the EPC2LC20 device (don’t select the configuration device pull-up resistor). Click ok.
6. Click ok to generate the EPC2 .pof file.

The EPC2 .pof (programmer object file) can be added to the multi-device JTAG chain programming list just as is done to configure the FPGAs over JTAG. If Max+Plus II is used to program the EPC2, it will first erase the device, then program the device, and finally verify the device contents. Max+Plus II can be used to check whether a device is erased and to verify the contents of an EPC2 against a selected .pof file. Max+Plus II does not give you the option of simply erasing an EPC2.

To generate a Jam file from the EPC2 .pof file, a further conversion is required (the following assumes that ONLY one EPC2 in the JTAG chain is to be manipulated):

1. In Max+Plus II, bring up the device programmer.
2. Select File → Create Jam or SVF file.
3. Add to the device list any devices past the target device (do not select a programming file).
4. Select the .pof file just created and add it to the device list.
5. Add to the device list any devices before the target device (do not select a programming file).
6. Select the output file format (.jam files are ASCII, .jbc files are binary).
7. Click ok to generate the Jam file (it can take a minute or so).

For example, to program the PCI core EPC2 in in the correlator JTAG chain, first add the .pof file, then a bypassed EPC2, then 12 bypassed FLEX10K100A devices. The resulting Jam file can be used to erase, program, and verify the PCI core EPC2 using the Jam Player with command line options as follows (this assumes the ByteBlasterMV cable attached to the parallel port at 0x378):

- ERASING:
  ```
  jam -p378 -dDO_ERASE=1 pci_epc2.jam
  ```

- BLANK CHECKING:
  ```
  jam -p378 -dDO_BLANKCHECK=1 pci_epc2.jam
  ```

- PROGRAMMING:
  ```
  jam -p378 -dDO_PROGRAM=1 pci_epc2.jam
  ```

- VERIFY:
  ```
  jam -p378 -dDO_VERIFY=1 pci_epc2.jam
  ```

- PROGRAMMING with VERIFY:
  ```
  jam -p378 -dDO_PROGRAM=1 -dDO_VERIFY=1 pci_epc2.jam
  ```
- **Read the UESCODE:**
  
  ```
  jam -p378 -dDO_READ_UES=1 pci_epc2.jam
  ```

  The device programming command line option implicitly erases the device prior to programming.

  When an EPC2 is programmed for the first time, or the configuration code stored in the EPC2 is updated, you must pulse the CONFIG# signal that is routed between the EPC2 and the attached FLEX 10K device. CONFIG# is pulsed low by either cycling the power or by transmitting the JTAG command INIT_CONF to the EPC2 device. To transmit the JTAG command INIT_CONF, under Max+Plus II, bring up the programming and select JTAG→Initiate configuration from configuration device. Pulsing CONFIG# initiates passive serial download and updates the contents of the target FLEX 10K attached to the EPC2.

  During debugging of the correlator boards ‘on the bench-top’, it is advisable to erase the contents of the EPC2 devices prior to powering-up the boards. Erasing the EPC2s will result in the FPGA passive serial configuration not being performed after power-up and hence avoids potential floating CMOS inputs (for example, on the PCI core PCI-bus inputs). This same effect can be implemented by loading the 2-pin headers next to the EPC2 devices. These headers short CONFIG# to ground. Since CONFIG# never pulses high again, passive serial configuration does not start. The down-side of the 2-pin header approach is that JTAG configuration of the attached FLEX 10K devices will also fail, due to the fact that as soon as they enter user mode, they see CONFIG# low, and enter a reset-like state.

### 4.2 FLEX10K programming

FLEX 10K programming is performed in a similar manner to the EPC2 EPROM programming. Max+Plus II provides adequate flexibility for programming the devices on the correlator board, so the use of Jam has not been investigated in detail.

One thing that should be kept in mind is that if you want to program a single .sof file to all 10 FPGAs, the Jam file format generated by Max+Plus II does not support the PRE_IR, POST_IR, PRE_DR, POST_DR commands necessary to tell the Jam Player how many dummy bits to insert before and after the file contents.

### 4.3 Cypress CY37064 programming

Cypress supplies a user interface to their Jam software known as JamISR. The instructions for use of this software are given in [10]. Cypress also provide source code for a command line Jam Player. This source code is an edited version of the Altera code. The edits remove all but the PC functionality and provide an interface to Cypress’ JTAG cable.

### 4.4 A final note

The programming, verification, and erasing commands are proprietry, so they are not directly published. However, it appears that the ASCII versions of the Jam files store these 10-bit commands in the parameter A18[0..9]. The commands must be scanned into the instruction register using IRSCAN, so it would be straightforward to modify the Jam Player to output to the screen whenever it was writing a command into the instruction register. This would allow the identification of the ISR command codes. Since the Jam Player command line gave the functionality desired (program and erase), this idea was not investigated further.
References


