XAPP388 (v1.2) May 15, 2003

On the Fly Reconfiguration with CoolRunner-II CPLDs

Summary

This application notes describes the CoolRunner™-II CPLD capability called “On the Fly” (OTF) Reconfiguration. OTF permits the CPLD to be operating with a design pattern and simultaneously acquire a second pattern during the operation of the first pattern. The second pattern can be configured into the device with a minimal disturbance to the operation of the device. Additional capabilities, applications and limits to this operation will be discussed in further sections.

Introduction

In system programming (ISP) was initially seen as a way to program Complex Programmable Logic Devices (CPLD) while the devices were attached to printed circuit boards (PCB). Originally, this capability primarily focused on delivering the device configuration without having to use an external programmer. Many changes to that capability have evolved, including JTAG testing and pin-locking architectures capable of sustaining multiple design edits while attached to the PCB. OTF takes ISP to a new level, and will undoubtedly spawn many new applications to take advantage of this capability. OTF permits subsequent designs to be reloaded in succession to a device while it is operating on a previous design configuration. The key to optimized OTF reconfiguration is to use the Xilinx-patented Real Digital technology which permits two simultaneous design images to reside within the part at any point in time.

Discussion

Real Digital technology combines a nonvolatile programming cell with a volatile one. Most digital designers are familiar with shadow memories, where one memory array tracks another one. At some point, their contents may be different, but they become “resynchronized” at another point. With Real Digital, the nonvolatile cell is a version of EPROM that is electrically programmable and the volatile cell is similar to an SRAM cell, but not organized with the same random architecture of an SRAM. When a new pattern is programmed into the CPLD through the JTAG pins, it is first delivered into a staging shift register, then transferred into the nonvolatile cells. After the programming of the nonvolatile cells completes, there is a specific step where the nonvolatile contents transfer into the volatile cells. Operation of the logic actually occurs outside of the volatile cells. Once the nonvolatile cells duplicate contents into the volatile cells, the nonvolatile cells are available for accepting a new pattern.

Figure 1 shows the high level CoolRunner-II CPLD family architecture (further detailed in the application notes and data sheets listed in **Appendix B: CoolRunner-II Resources**, page 9). Figure 2 shows the underlying structure of an area of programmable cells that dictate the actual functional behavior of the CPLD when programmed with a bit pattern.

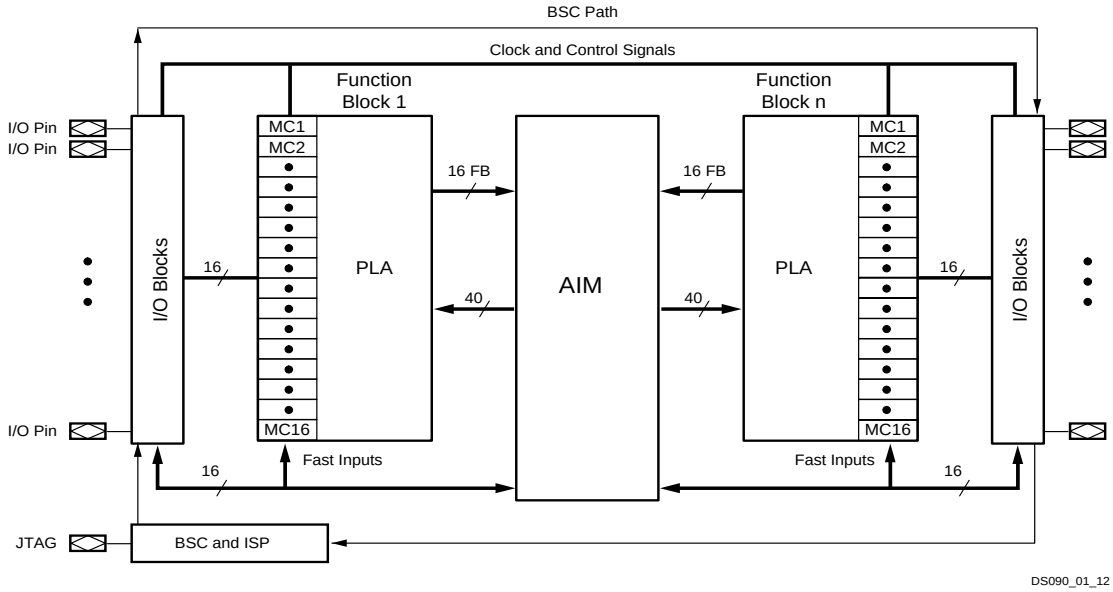

Figure 1: **CoolRunner-II CPLD High Level Architecture**

Figure 3A expands the detail on a single programmable cell, revealing its two-element nature, with the nonvolatile portion (NV) on the left and the volatile (V) or SRAM portion on the right. In particular, Figure 3 B-F show successive stages of OTF programming. Initially, the cell is in the virgin state as shown in Figure 3B where both cell portions are blank. Then, the first pattern (P1) is delivered into the NV portion, but the V portion is still blank. This is shown in Figure 3C. After all such NV cells are programmed, the part undergoes a transistion where the NV cells copy into the V cells, which is shown in Figure 3D. At this point, the part is loaded and running with pattern P1 programmed into the configuration cells. Figure 3E shows the part operating with pattern P1 in the volatile cells, but a new pattern (P2) loaded into the NV cells. Note that it will require a specific command to be delivered to transfer the contents of the NV cells into the V cells, and this has not yet occurred. Upon issuing the command to accomplish OTF, the P2 pattern copies into the volatile cells and the part is now operating with the P2 pattern. At this point, if needed, it would be possible to keep going and deliver a third programming pattern.

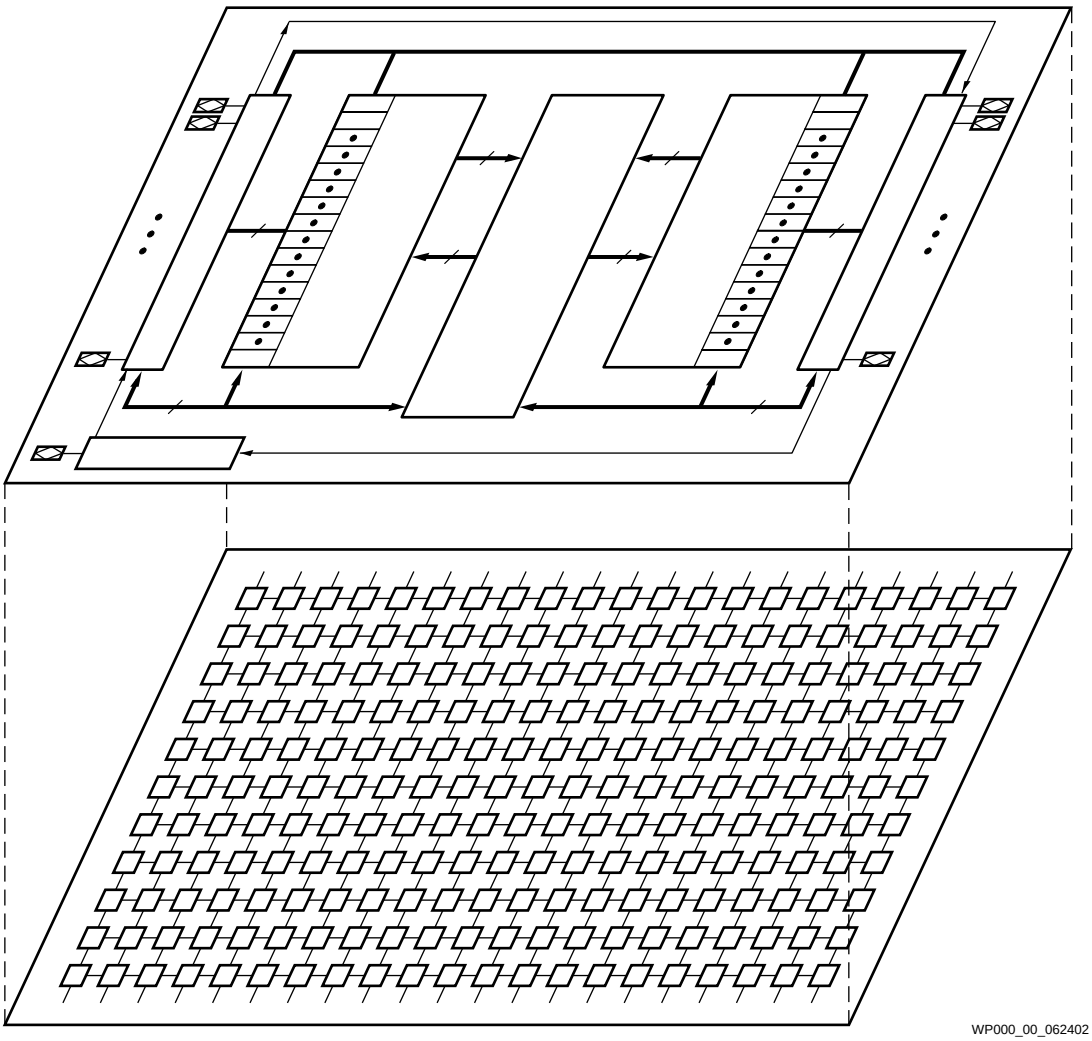

Figure 2: **CoolRunner-II Architecture with Configuration Memory Exposed**

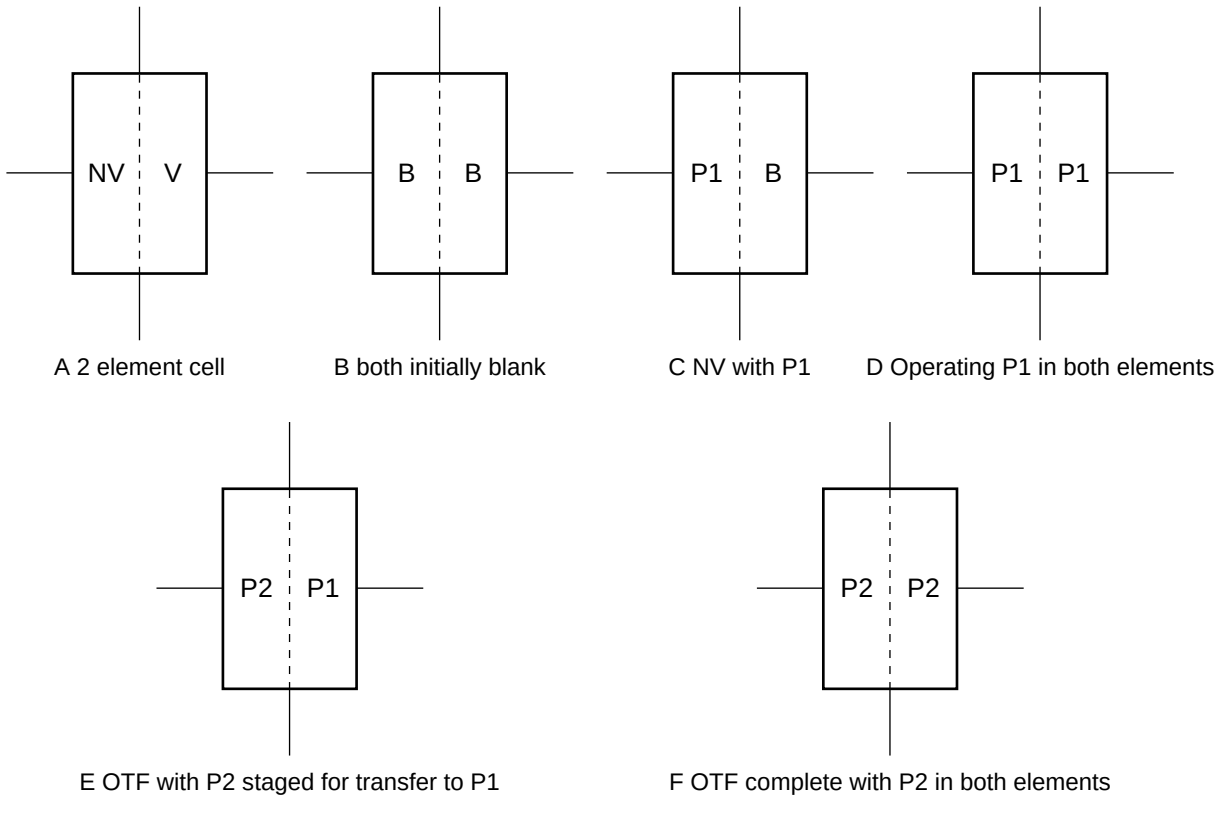

Figure 3: **Two-Element Cell Structure and Transition Steps for OTF Reconfiguration**

Operation Details

Naturally, it is important that designers be given easy-to-use tools that permit the seamless operation of OTF. Xilinx provides this in its ISE Design Software (including ISE WebPACK™). Design-creation using ISE design software is a standard flow starting with design capture and moving through fitting, application of constraints, and device programming. Details of this flow are well outlined in the Xilinx software help packages and numerous software tutorials.

To return to the process of OTF reconfiguration outlined earlier, we will assume the existence of two programming patterns, P1 and P2, which already exist as JEDEC files in the design directory. Going further, we will assume that pattern P1 is already programmed into the part with the iMPACT software. IMPACT has many options, and the one we are interested in is chosen from the menu shown in Figure 4. In this case, we are showing a single XC2C64 macrocell part in a JTAG chain. It is possible to target any part within a chain and program it as needed. After clicking OKAY on the window shown in Figure 4, a progress bar will indicate that the NV cells are being loaded. When that progress completes, the target part will be in a condition similar to that shown in Figure 3E. Specifically, it will be up and operating out of pattern P1, with pattern P2 already loaded into NV memory.

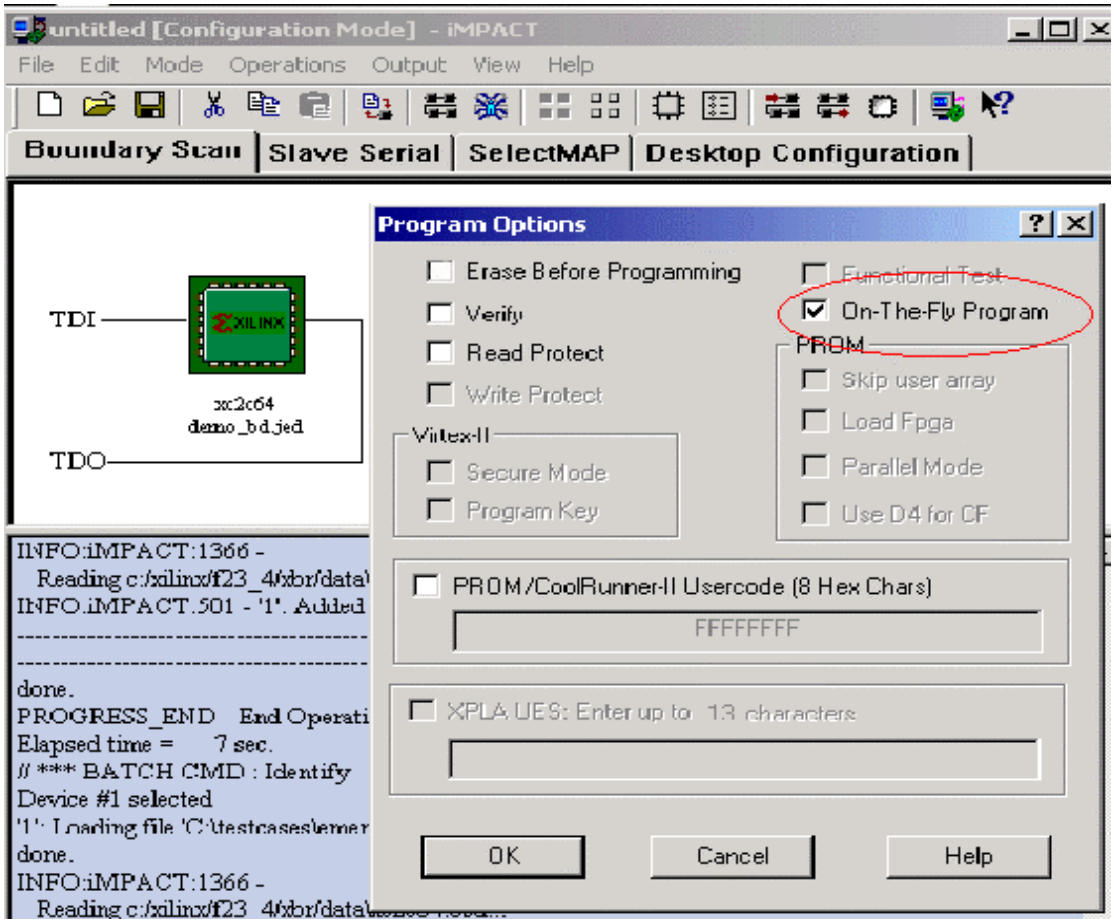

Figure 4: **iMPACT Programmer Window with OTF Programming Option Checked**

At the appropriate point in time, when we wish to activate pattern P2, we will need to supply the appropriate JTAG commands. It is possible to do this with a set of JTAG operations coming from a microprocessor, but consistent with this example, we will proceed to activate that operation from the iMPACT GUI as shown in Figure 5.

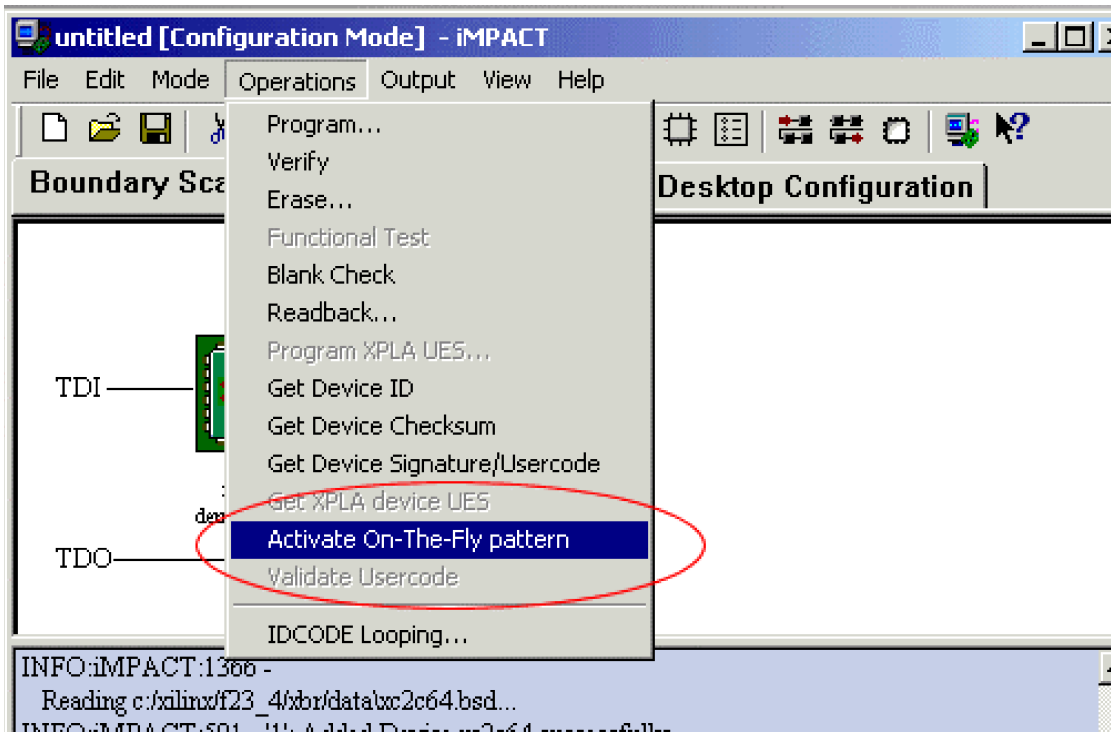

Figure 5: **Delivering the OTF Activation Command from the iMPACT GUI**

After issuing the activate command, the new image will copy P2 from the NV cells into the volatile ones and the new pattern comes up running. There is a brief (~50 µsec) time period when the pattern is transitioning and the I/O pins assume a momentary tri-state (weakly pulled up) configuration, that ends with the pins in their new "P2" configuration. The brief transition time is a substantial improvement over stopping operation for a full program operation.

Applications

This section outlines several applications that can take advantage of OTF programming.

Example 1 – Configuring an FPGA

Figure 6 shows two classic CPLD applications being done by a single CPLD. First, it is quite common for the CPLD to be used as a configuration controller for an FPGA. The figure outlines the sequence. At power up, the CPLD already has its FPGA configuration internally loaded (P1). With task P1, the CPLD drives the FPGA pattern through the FPGAs parallel configuration port called the Select Map mode. After completing that task, the CPU loads a new pattern into the CPLD, giving it a new identity (P2) as an interrupt handler. After loading P2, the CPU subsequently loads a copy of P1 back into the CPLD (taken from the "Select MAP JEDEC" in an EPROM), but does not activate P1 into the volatile cells.. Note that on the next power up, the CPLD will already be loaded and ready to configure the FPGA again, as it has planned for that eventuality.The advantage of this configuration is that the CPLD actually serves two purposes with one chip.

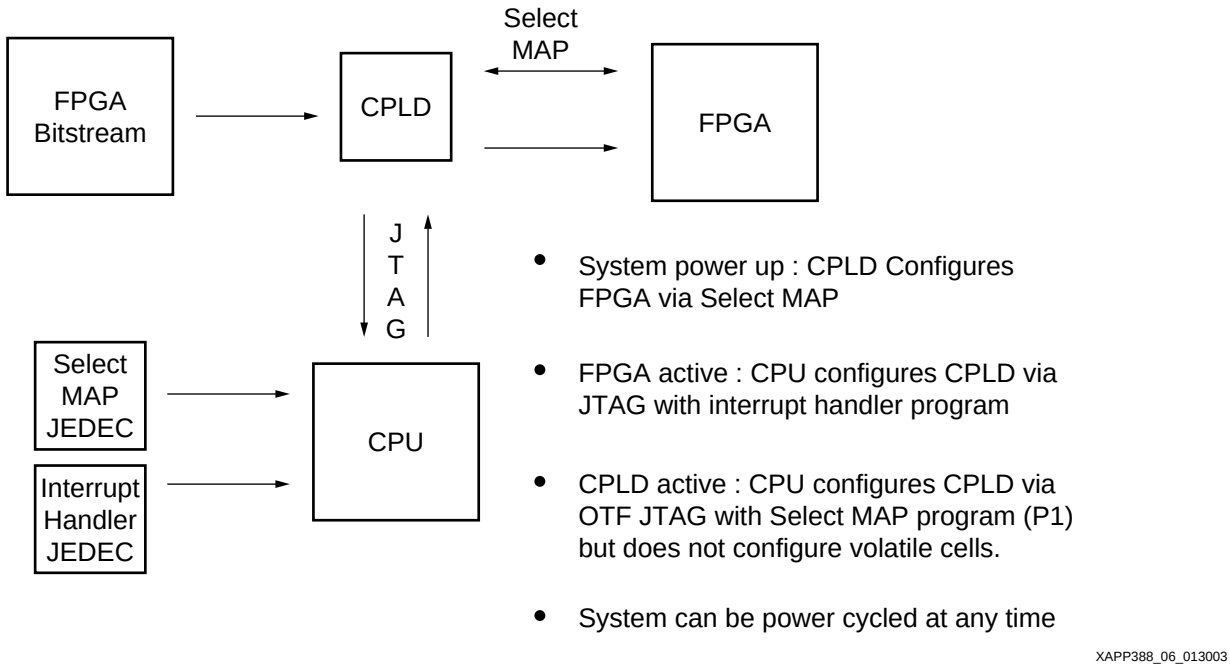

Figure 6: **Combining Two Classic CPLD Applications**

Example 2 – Updating an Instruction Set

The PicoBlaze microcontroller design is another option for an interesting OTF operation. In this case, it is possible to take a program that is running with one version of the processor executing a specific instruction stream and substitute another version, with exactly the same pinout, while operating. If the program is sufficiently small, it can even be folded within the design so that both the program as well as the architecture become simultaneously updated with the same activation command. See **XAPP387** on the PicoBlaze 8-bit microcontroller for more information.

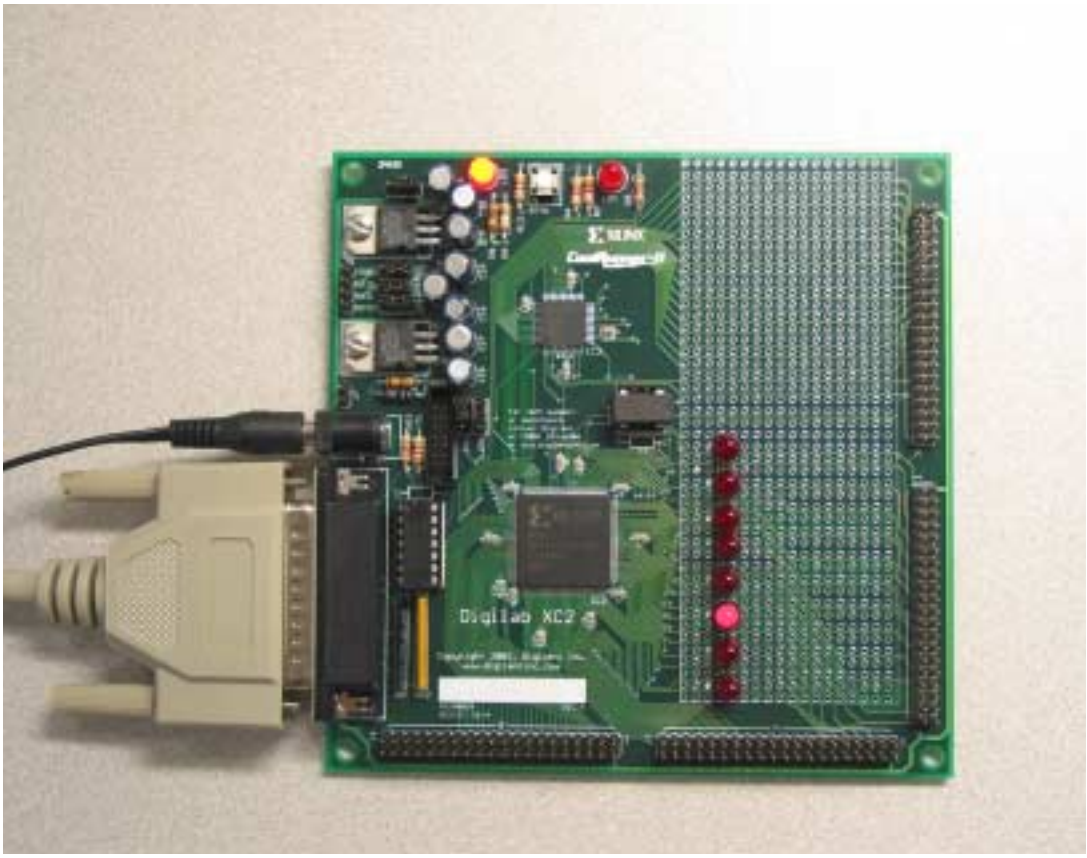

Figure 7: **CoolRunner-II Design Kit Board with PicoBlaze**

Example 3 – Reprogramming a fixed Cross-Bar Switch

XAPP380 shows the design of a generic cross bar switch. This design version uses external signals to select the switch connections to be made, but it can be easily altered to make that selection be internally hardwired, if need be. Then, the switch selections can be accomplished by dropping an alternate version of the design into the chip through OTF even while the chip is currently switching.

Example 4 – Updating a small EPROM table

All PLDs were originally built up as replacements for EPROMs. It is fitting that we can emulate an EPROM inside a CoolRunner-II CPLD in order to do whatever functions we might wish. For instance, it is possible to make a small multiplication table, a logarithm table or simply insert a preferred profile for an A/D converter as a table look up of values to use. It is possible to have multiple designs where the only thing that changes within the design (as far as the outside world can tell) is the internal EPROM. Using OTF to make those changes permits “real time” transition to new behaviors, as needed.

Example 5 – Re-keying a Stream Cipher

Flip flops, EX-ORs and clocks are about all you need to build up LFSR based stream cipher designs. Switching keys on the fly permits robust protocols that can be tough to crack for data communication designs. Stream Ciphers are of particular interest for CPLD implementations in that today’s CPLDs can contain lots of flip flops and logic without any speed restrictions or fitting difficulties as the designs tend to underuse the architecture. Additional references for this are given in the appendix. Figure 8 shows an extremely simple stream cipher. Many more elaborate structures can be found in the literature.

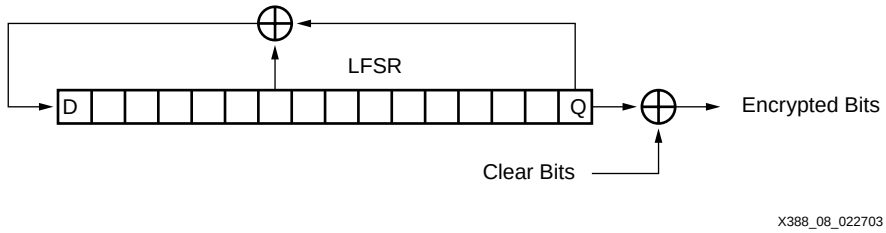

Figure 8: **Simple Stream Cipher with LFSR**

Example 6 – Board level testing

Similar to **Example 1**, it is possible to use the CPLD for multiple very different applications. Instead of first programming an FPGA, then assuming a processor support task like interrupt vector handling, it is possible to insert signature analysis circuitry to perform board level tests during initialization periods or alternately during some other board level BIST functionality. Of course, after that occurs, a third personality can be overlayed such as the previously mentioned CPU support function. Again, additional references for this application are given in **Appendix B: CoolRunner-II Resources**, page 9.

Conclusions

Applications beyond those listed above can be envisioned. CoolRunner-II is both fast and low enough in power that portable applications for communications and even signal processing are now a possibility. Portable FIR filters for brainwaves would be one possibility and additional high speed cryptography for wireless and cellphone applications are another. The limits to what types of applications might take advantage of OTF programming are just now being explored. Although there is an upper limit on the number of times a part may be programmed, it is high. Jump in.

Appendix A: Additional References

"Pseudorandom bit generator based on dynamic linear feedback topology," R. Mita, G. Palumbo, S. Pennisi and M. Poli, Electronic Letters, Vol. 38, No. 19, pp 1097 –98.

Logic Design Principles, E.J. McCluskey, Prentice-Hall 1986.

Digital Design Principles and Practices, J.F. Wakerly, Prentice-Hall, 2000

Appendix B: CoolRunner-II Resources

Application Notes

http://www.xilinx.com/xapp/xapp375.pdf (Timing Model)

http://www.xilinx.com/xapp/xapp376.pdf (Logic Engine)

http://www.xilinx.com/xapp/xapp377.pdf (Low Power Design)

http://www.xilinx.com/xapp/xapp378.pdf (Advanced Features)

http://www.xilinx.com/xapp/xapp379.pdf (High Speed Design)

http://www.xilinx.com/xapp/xapp380.pdf (Cross Point Switch)

http://www.xilinx.com/xapp/xapp381.pdf (Demo Board)

http://www.xilinx.com/xapp/xapp382.pdf (I/O Characteristics)

http://www.xilinx.com/xapp/xapp383.pdf (Single Error Correction Double Error Detection)

http://www.xilinx.com/xapp/xapp384.pdf (DDR SDRAM Interface)

http://www.xilinx.com/xapp/xapp387.pdf (PicoBlaze Microcontroller)

http://www.xilinx.com/xapp/xapp388.pdf (On the Fly Reconfiguration)

http://www.xilinx.com/xapp/xapp389.pdf (Powering CoolRunner-II CPLDs)

http://www.xilinx.com/xapp/xapp393.pdf (8051 Microcontroller Interface)

http://www.xilinx.com/xapp/xapp394.pdf (Interfacing with Mobile SDRAM)

CoolRunner-II Data Sheets

http://direct.xilinx.com/bvdocs/publications/ds090.pdf (CoolRunner-II Family Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds091.pdf (XC2C32 Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds092.pdf (XC2C64 Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds093.pdf (XC2C128 Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds094.pdf (XC2C256 Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds095.pdf (XC2C384 Datasheet)

http://direct.xilinx.com/bvdocs/publications/ds096.pdf (XC2C512 Datasheet)

CoolRunner-II White Papers

http://www.xilinx.com/publications/products/cool2/wp_pdf/wp165.pdf (Chip Scale Packaging)

http://www.xilinx.com/publications/whitepapers/wp_pdf/wp170.pdf (Security)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
02/28/03	1.0	Initial Xilinx release.
03/13/03	1.1	Minor revisions.
05/15/03	1.2	Minor revisions