XAPP395 (v1.2) September 22, 2003

Using DataGATE in CoolRunner-II CPLDs

Summary

This application note outlines the various ways designers can utilize the DataGATE feature of CoolRunner™-II CPLDs.

Introduction

CoolRunner-II CPLDs deliver the lowest power consumption in today's CPLD marketplace. Built from standard CMOS structures, they achieve the classic I_{CC} = CVF relationship where "C" is driven capacitance, "V" is voltage swing and "F" is the switching frequency of gates driven within the part. The capacitance and voltage values in the equation are essentially demanded by the 0.18 micron process, but the "F" part of the expression is due to the design characteristics of the customer design inside. The desire to lower the "F" aspect resulted in the DataGATE feature in CoolRunner-II CPLDs.

As the name implies, the feature "gates" data. This application note shows how to get as near to the origin of the I_{CC} versus "F" curve as we can with today's technology. It describes the practical aspects of how DataGATE works, the timing model, and what you will need to do in the software to make it work. Further, it shows four different ways the circuitry can be used for power reduction, as well as for circuit debugging, printed circuit board "hot plugging," and device security. These are just some ideas on how to use this feature, and others may come to mind as users learn to operate DataGATE.

The final section of this application note lists additional application notes which cover other aspects of Xilinx CoolRunner-II CPLDs.

Operation

Figure 1 details the delivery of the DataGATE Assertion Rail throughout the I/O structure of a CoolRunner-II CPLD. This image doesn't convey the generation of the DataGATE activation method, but simply the delivery of the control signal to each input pin, with independent control of whether that pin will participate in the gating action. Specifically, two items should be noted. First, each pin can be programmed to participate in the DataGATE operation or not—the standard default is not to participate. The second aspect is that the individual pass transistors permit signal entry into the CPLD, or blocking of the input pin. Figure 2 expands the detail for a single input pin. If the input pin becomes blocked, the last driven value into the CPLD will automatically be latched and held so a solid binary value is delivered into the CPLD core. It should be noted that the Assertion Rail is driven from a specific macrocell within the CPLD, and that when that macrocell drives high, input data will be blocked at participating pins. If the Assertion Rail is low, data passes freely into the chip. The condition of the DataGATE Assertion Rail is manifested at a specific pin (designated DGE), which will vary from chip to chip within the CoolRunner-II family.

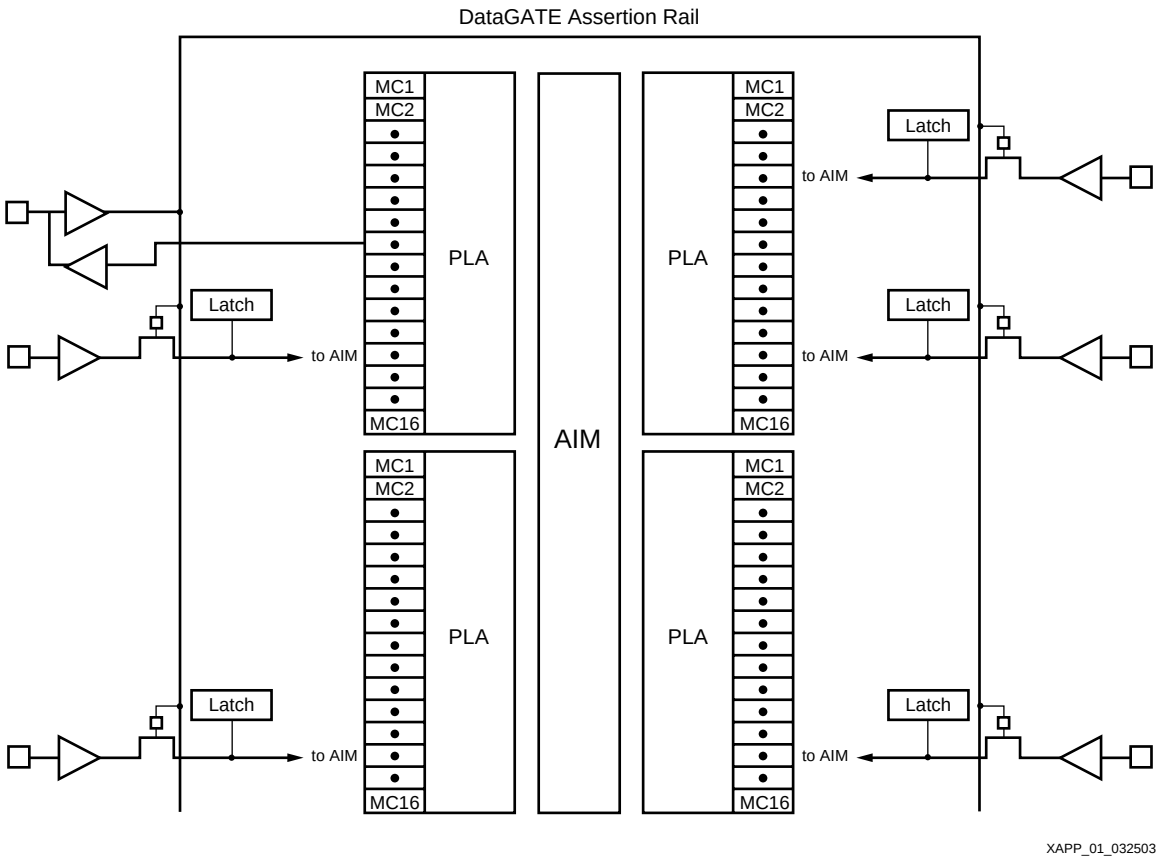

Figure 1: **DataGATE Assertion Rail**

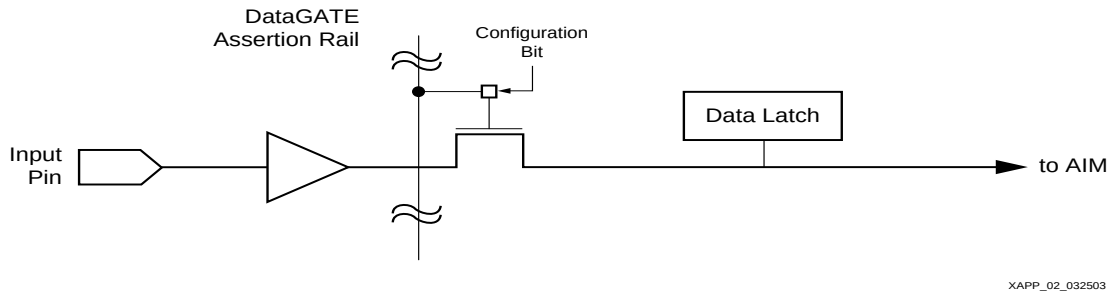

Figure 2: **DataGATE Assertion Rail Input Pin, Latching Old State**

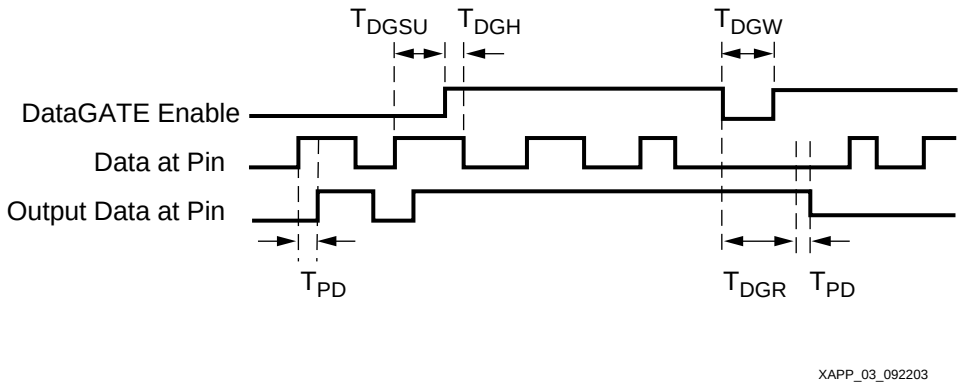

Figure 3: **DataGATE Timing Parameters**

Figure 3 details the timing action of the DataGATE operation. Specifically, the DataGATE Enable pin, if driven from an external pin, can be viewed as a clocking type function. Given that, it has standard clock type timing parameters, a setup time (T_{DGSU}), hold time (T_{DGH}), a minimum width (T_{DGW}) and DataGATE recovery time (T_{DGR}). For clarity, Figure 3 also shows the propagation time (T_{PD}) from input to output of a simple signal. DataGATE operation is described in detail in **XAPP378**, which describes how to declare the appropriate conditions in ABEL, VHDL and Verilog. DataGATE pin participation is also described, so you can easily dictate which pins will participate in the operation.

Applications

Simple Power saving

Figure 4 shows the simplest DataGATE configuration, and possibly the most frequently used method. Here, a simple external signal is delivered on a pin to the CPLD, which connects that signal through the Advanced Interconnect Matrix (AIM) to the DataGATE macrocell. The DataGATE macrocell is a specific macrocell in each part, which can drive the assertion rail. The logic created at the macrocell is arbitrary, so for this example, we can assume it is a simple connection. Hence, when the external pin drives high, that signal drives the Assertion Rail high. When the external pin drives low, so goes the Assertion Rail. As discussed earlier, when driven low, participating and nonparticipating input pins will have their signals forwarded into the CPLD. When driven high, participating input pins will be blocked and latched, while nonparticipating pins will still pass their signals into the CPLD. It is possible for all input pins to be “gated,” so care should be used in choosing signals. Blocking clocks typically has the highest payoff, but also the highest risk. Figure 4 shows the case where an external condition decides when to save power.

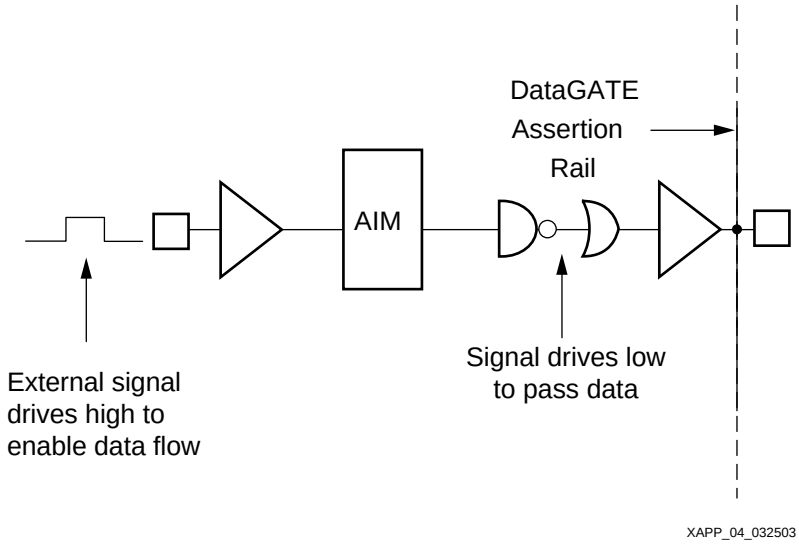

Figure 4: **Simple External DataGATE Control Signal**

Timer

Figure 5 expands on the idea of Figure 4, including a timeout circuit (timer) that is employed to drive the Assertion Rail whenever the timeout signal asserts. We can assume that the timer delivers a “0” when the timer is counting clocks, but when it hits a previously chosen timeout period, it asserts a logical "1" and drives the blocking signal to the Assertion Rail. Because most timers are fairly simple counters, and frequently, it is not important to be too fussy about exactly which count value is chosen (ie, plus or minus a clock or two won’t matter), it may be smart to pick the simplest logic structure that can solve the timer problem. In all likelihood, this will be a Linear Feedback Shift Register (LFSR), which will produce most of the values of a binary counter, but in a non-consecutive order and with minimum logic beyond the macrocell flip flops. By either "And"ing the appropriate state, or choosing a state variable for direct connection to the Assertion Rail, the timer can drive the Assertion Rail as needed.

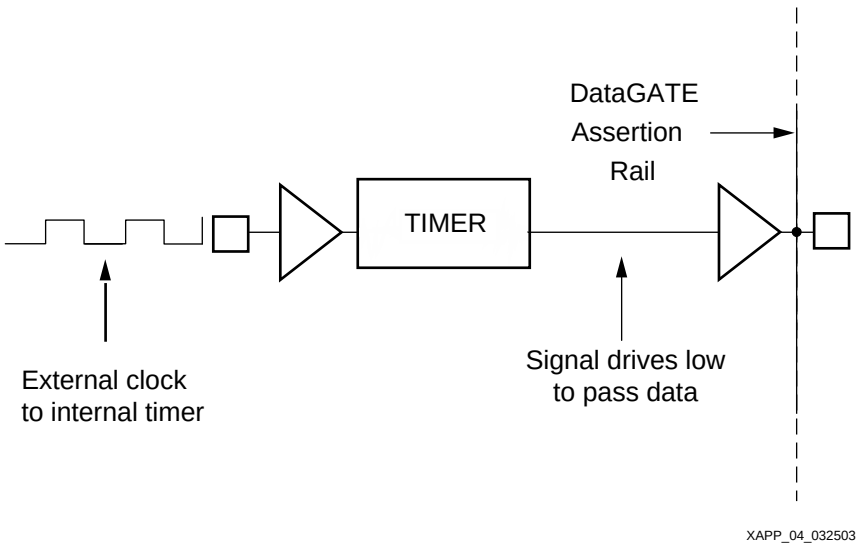

Figure 5: **DataGATE Under Control of Internal Timer**

It would also be possible to use the timer circuit to lock out signals that are deemed to be infrequently used, but periodically “wake up” to sample the pins for whatever reason that might be needed. Alternately, selected signals simply become permanently blocked at the completion of a timeout period. Another view would not have the counter counting “time,” but rather some other number of events that drive the “timer” clock input site—say, count 400 interrupt signals then block this set of pins.

Controller

As suggested earlier, any logic can be constructed to drive the Assertion Rail, so state machines can be created, too. In this sense, external variables can be combined with state variables to create the condition that drives the rail. As described with the timer, external signals can create a tally that is used to power down, but also, combinations of events and states can create the blocking event. Figure 6 shows the insertion of an arbitrary controller module, which includes flip flops and logic to interact with the external signals and clocks.

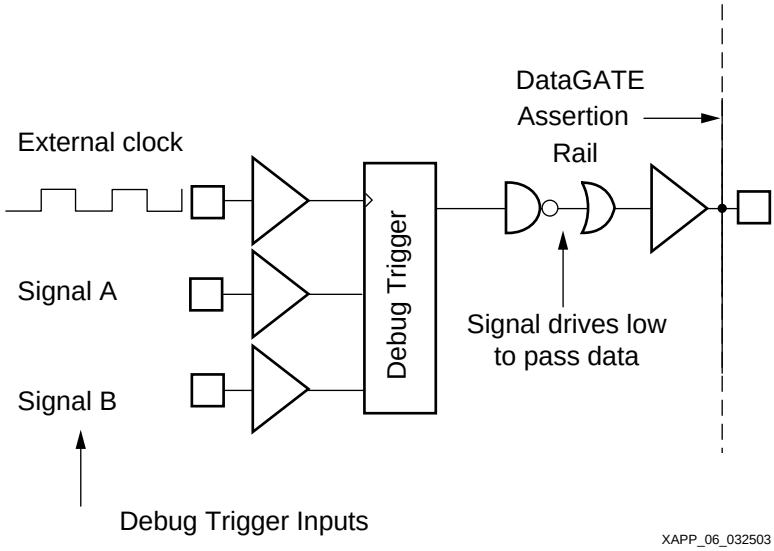

Figure 6: **DataGATE Rail Driven by Internal State Machine Controller**

Another important point to consider when using DataGATE is the overall time operation of a system. At initial power-on, certain actions occur in many systems—processor bootstrap, memory initialization, memory space and I/O space definition, etc. After these initial activities, many items remain in their initial condition and stay there throughout the operation. If certain registers or logic are only used at the beginning, but free running strobe signals, clocks, or data continue unnecessary switching, then those logic sections might benefit from “DataGating.”

Debugging with DataGATE

Latching the input pins when the Assertion Rail blocks inputs creates an additional capability: an on-chip debugger. Logic analyzers typically capture the contents of binary signals—the “state”—by triggering on an event, then snapshotting the data into a memory buffer for later readback. Combining the previously described controller with the JTAG circuitry inherent in a CoolRunner-II permits a simpler, but effective capability. For instance, assume that most of the pins are participating in the DataGATE operation. Assume next that we wish to isolate the state of the CPLD when a certain combination of inputs and internal CPLD states occurs—say an “error” state. By creating a state machine out of scrap gates and flops within the CPLD, the inputs can be blocked and held until the JTAG circuit reads back the entire state of the CPLD for dissection and analysis with a PC and JTAG cable. Alternately, the triggering event of a logic analyzer can be taken over to the CPLD and applied just as in the first, simple “power saving” model described above. Figure 7 gives some detail on how this might occur.

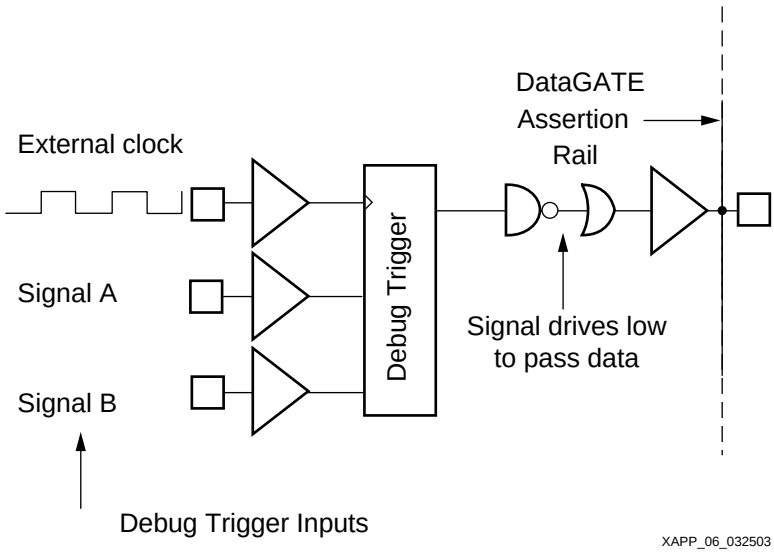

Figure 7: **Debugging with DataGATE**

This model of debugging is one way to use the input latches on the CPLD, although others view this as a very large set of input latches that can be controlled for simple data capture at the pins. The only drawback to this is that the timing for the data capture tends to increase on the larger parts, so users should plan for a little extra time delay.

Hot Plugging with DataGATE

Hot Plugging, Hot Socketing and Live Insertion are all terms that are used interchangeably. The standard situation is this: there is a rack of equipment that is powered up, and a user needs to insert a board into that rack without turning the power off. The methodology for this is somewhat nebulous because expectations vary widely between engineers: depending on how much they know about the situation, engineers all expect different behavior . Some expect the board insertion to go perfectly, without any control signals being disturbed or data bits being dropped. Others expect the board insertion to result in possible data corruption, but not in any catastrophic crashing of the systems. Experienced designers cross their fingers and are satisfied if the event doesn't result in the rack catching on fire.

A user's expectation should be that there will be no actual damage, but that extra care must be taken to prevent corrupt behavior. For instance, all the devices on the "cold board" are uncharged where most of the elements within the "hot" rack are charged up (bus lines, decoupling capacitors, whatever). Inserting a discharged board presents a substantial capacitive load to a system that has finite charge on it. This will require charging the "cold" board to the appropriate level before operation can occur. One solution to this has been the "extended finger" method whereby little extension tabs are placed on the cold board, so electricity is delivered to the board before the logic enters the rack connectors; however, this requires advanced planning to make the little tabs. We present an alternative method which uses the CoolRunner-II DataGATE feature to eliminate corrupt data.

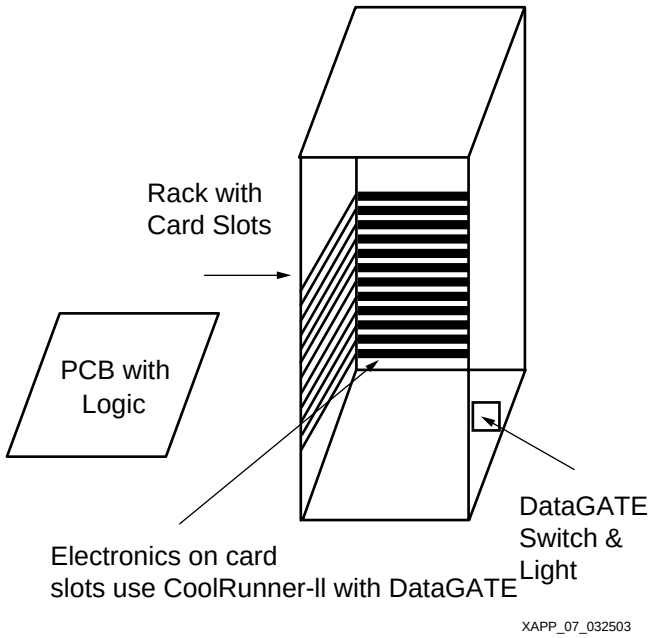

Figure 8: **Hot Plugging with DataGATE**

For this description, we assume all cards in the system, including the cold one to be inserted, have CoolRunner-II CPLDs attached to the slot connector pins. Also, all of the rack boards are connected to a control signal, which is either located on the rack or controlled by software on a card within the rack. The control signal can come from an internal processor signal, or from a switch on the rack (Figure 8). Before the hotplug event occurs, the control signal asserts to make all inputs within the rack latch their current state until the plug completes and the control signal releases. The release of the control signal should occur after the cold board is warmed up and initialized. The system then proceeds to operate. Depending on the system requirements, additional buffering or protocol actions should be included to account for the brief pause in the operation.

Security

Figure 9 shows another DataGATE application. In this case, the CPLD is involved in securing some aspect of the system operation. For simplicity, let's assume a password is involved in the operation. The password is delivered into the CoolRunner-II CPLD, which—if the password is correct—permits other signals to continue entering the device. If the password is incorrect, the DataGATE blocks all the signals coming into the CPLD to forbid future trials. Designing the rest of the CPLD to perform some "mission critical" aspect of the whole system is critical here, so the system won't work while the chip is being gated. To attempt more passwords, the whole system must be power cycled. A more permanent action would be to erase the CPLD if the password doesn't match, but doing so would use the On the Fly Reconfiguration capability instead of only DataGATE.

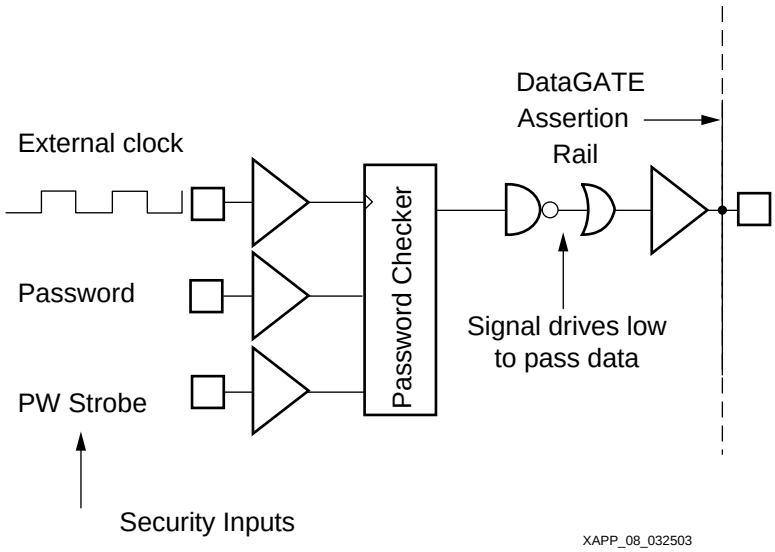

Figure 9: **DataGATE Dynamic Security**

Conclusions

DataGATE is a versatile and helpful feature designers can utilize when designing with CoolRunner-II CPLDs, although they are not required to use it and it can be used only over chosen pins, as needed.

A simple strategy for using DataGATE is to design with no initial regard for it, except in saving the macrocell that drives the assertion rail. Once the design is complete, evaluate the current drawn against the target current budget. Next, consider implementing CoolClock or other power savings options. Again, evaluate consumed current against target current. Review **XAPP 377** for other things to try.

Next, if the design does not hit its target, or you simply wish to see how low the current can be taken, *then* consider using DataGATE. Consider sections of your design that initially "wake up" and perform active behavior, then go "quiet" for the rest of the operation. Assuming they are driven from active pins, can you identify some time or event that would permit them to be blocked by the DataGATE circuit? If so, define the time or event, create a logic circuit to manifest at that time and have it drive the DataGATE rail. Evaluate the current again. If you need less current, can you include more pins? Can you slightly change the block point (in time) to include more pins? If so, alter your design and re-evaluate. Should you encounter a "tweak" that adversely affects your design, you can back up a revision to the last working one and know that it is better than your original solution.

Further Reading

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
04/01/03	1.0	Initial Xilinx release.
05/15/03	1.1	Minor revisions
09/22/03	1.2	Fixed Figure 3.