WP214 (v1.0) January 10, 2005

TTL "Burn Rate" for Xilinx CPLDs

By: Jesse Jenkins

The goal of this document is to familiarize logic designers that have previously not used programmable logic with the benefits of Xilinx CPLDs, and show how to go about translating existing TTL type logic into similar CPLD solutions. You might want to skip this whole discussion and go right to the TTL "Burn Rate" table at the end of this document, but for clarity, we will explain how it was derived. The basis of the discussion begins with a familiarity of 7400 style logic, which is actually comprised of many different variations (7400, 74LS00, 74S00, 74C00, 74HC00, 74HCT00, 74ABT00, 74F00, etc.) These families were phenomenally successful, and many designers still use them. However, greater integration, lower power, faster solutions and in general, less noisy designs can now be had with a substantial price reduction by using Xilinx CPLDs. But, first, some basics need to be covered.

Basics

CPLDs are constructed from a building block called a "macrocell", which is a logic function that can be manipulated to form various different functions, suiting the needs of the designer. The macrocells are clustered in groups and called "Function Blocks" Figure 1 shows a macrocell for the Xilinx CoolRunner-II family of CPLDs. This is a very compressed diagram, which does not immediately expose its true strengths. For instance, on the left side of the diagram, there are a series of what appear to be "one input AND" gates. This is shorthand, which is necessary to show the relationship of these gates to the other gates in the picture. The "one input ANDs" are in fact 40 input ANDs, like that shown in Figure 2. We also annotate on Figure 1, the number of "p-terms" existing at a specific site. The top left p-term states "49 p-terms" when we show only one. This can be confusing, but a diagram that shows all 56 p-terms available (add up the p-terms in the diagram), each with 40 inputs, would be more confusing! We will not go into too much detail on the diagram. Suffice it to say that the building block we call a CoolRunner-II macrocell contains some gates built up with very large input ANDs, and has a flip-flop in it, that can be used to build up logic designs.

Most designers do not want to build up logic by manually connecting a part's basic gates, so Xilinx provides free software that does it for you. You still have work to do – define your design in the software, but much of the process is automatically done for you.

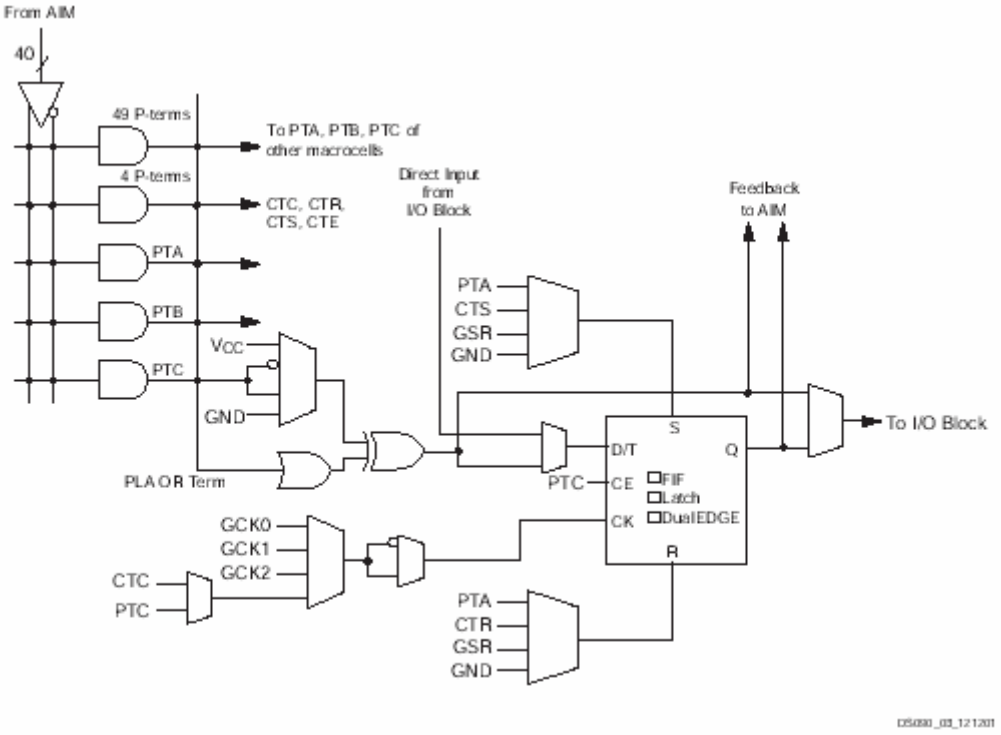

Figure 1: **CoolRunner-II Macrocell Diagram**

Your main task will be to figure out which logic chips you want to replace with CPLDs, then describe it for the design software, and let it optimize and enhance the design for you, creating a superior overall solution.

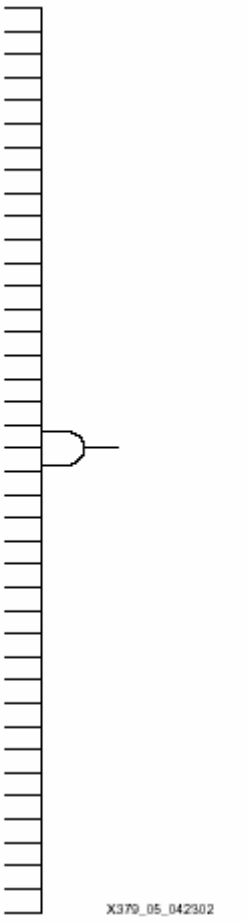

Figure 2: **CoolRunner-II AND Gate (Showing All Possible Inputs)**

An early task – which we show later – is to select an appropriate part for the logic. In theory, you will want to purchase the smallest, slowest part possible to solve your problem. In some cases, you might be unsure about whether you will want to make future changes, so you might wish to select a slightly larger part, to accommodate future, unforeseen changes. Let us look a little deeper at the Figure 1 macrocell, by identifying a very important place in the macrocell. See Figure 3.

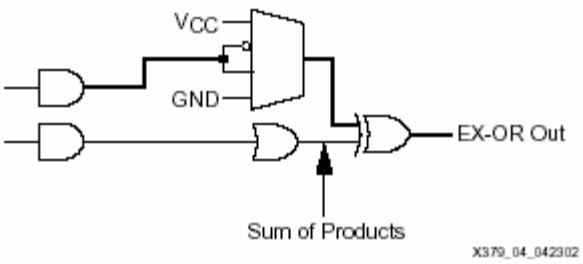

Figure 3: **"Trimmed down" Macrocell**

Figure 3 has removed most of the p-terms, the flip-flop and some of the multiplexers in the macrocell, and specifically identifies the OR gate output. This is the site where most combinational logic forms a function. As we learned in logic design classes, all combinational functions can be created from two level "sum of product" (SOP) logic. This is literally creating ANDed terms and ORing them up. Pretty basic. In Figure 3, we see that the SOP point attaches to the bottom leg of an EX-OR gate, and it is that site which actually becomes usable. The mux tied to the other leg serves a number of functions, but one is simply polarity control for the output of the EX-OR gate. For instance, if the mux VCC input is chosen, it will present a logic one to the mux output, driving one leg of the EX-OR. Recalling that if the two inputs of an EX-OR are A and B, then the output becomes A*/B + /A*B. If A is the top leg of the EX-OR and it is set to A=VCC =1, this expression becomes /B. That means the value attached to the other EX-OR leg inverts as it passes through the EX-OR. If the mux selects GND =0, then the EX-OR will simply pass the SOP site on the bottom leg directly through the EX-OR to its output. One way of looking at the mux, is that it can make the SOP invert, or simply pass through the EXOR gate. Although the OR gate is connectable to all 56 p-terms available to it, it can be configured to select any number from 0 to 56 p-terms, so lets consider the case where a single p-term is chosen. That means a 40 input AND gate passes through the SOP OR gate, and can then be passed as an AND, or inverted into a NAND.

We now see that one macrocell is capable of at least building up AND gates, or NAND gates, up to 40 inputs wide. That means specifically, that a 2 input NAND gate (sn7400) could occupy a macrocell. But, so could a 2 input AND gate (sn7408). It could also create an sn7430 (8 input AND), or an sn74133 (13 input NAND). All of these functions would only occupy a single macrocell, and have logic left over available to other macrocells for creating even more logic. As you might have guessed, you could also build up a 40 input AND/NAND gate, which has never had a 7400 equivalent! This function is particularly useful for decoding a whole microprocessor's address bus with a single gate.

The main idea here is that individual gates can easily be made with a single macrocell, for a large number of inputs. EX-OR gates are not quite as efficient in this architecture. By using two p-terms at a macrocell, you can form A*/B + /A*B on the sum of products OR gate output, then attach another operand – say C, through the mux in Figure 3, and achieve a three input EX-OR function coming out of the EX-OR gate. This is not as good as the other gates, but good enough to form the SUM bit of an adder. It still needs the carry created (another macrocell), but you can quickly see that adders will "burn through" two macrocells per bit for addition.

Figure 1 is Xilinx CoolRunner-II CPLD family, that operates with a core voltage of 1.8V, but can translate input signals from as high as 3.3V down to 1.5V, if appropriately connected to various power supplies and batteries. Figure 4 shows a similar macrocell – the XC9500XL, which is designed to have a core voltage of 3.3V, but can accept input signals as high as 5V, and deliver signals out as low as 2.5V. Xilinx actually makes two key CPLD families, the XC9500, XC9500XL, and XC9500XV products, and the CoolRunner XPLA3 and CoolRunner-II products. The XC9500 parts focus on standard, low cost, high speed applications, where the CoolRunner families focus on low cost, low power, high speed applications. Selection among them is usually done

based on package needs, I/O pins, power consumption, speed and price, with those factors prioritized by the end designer.

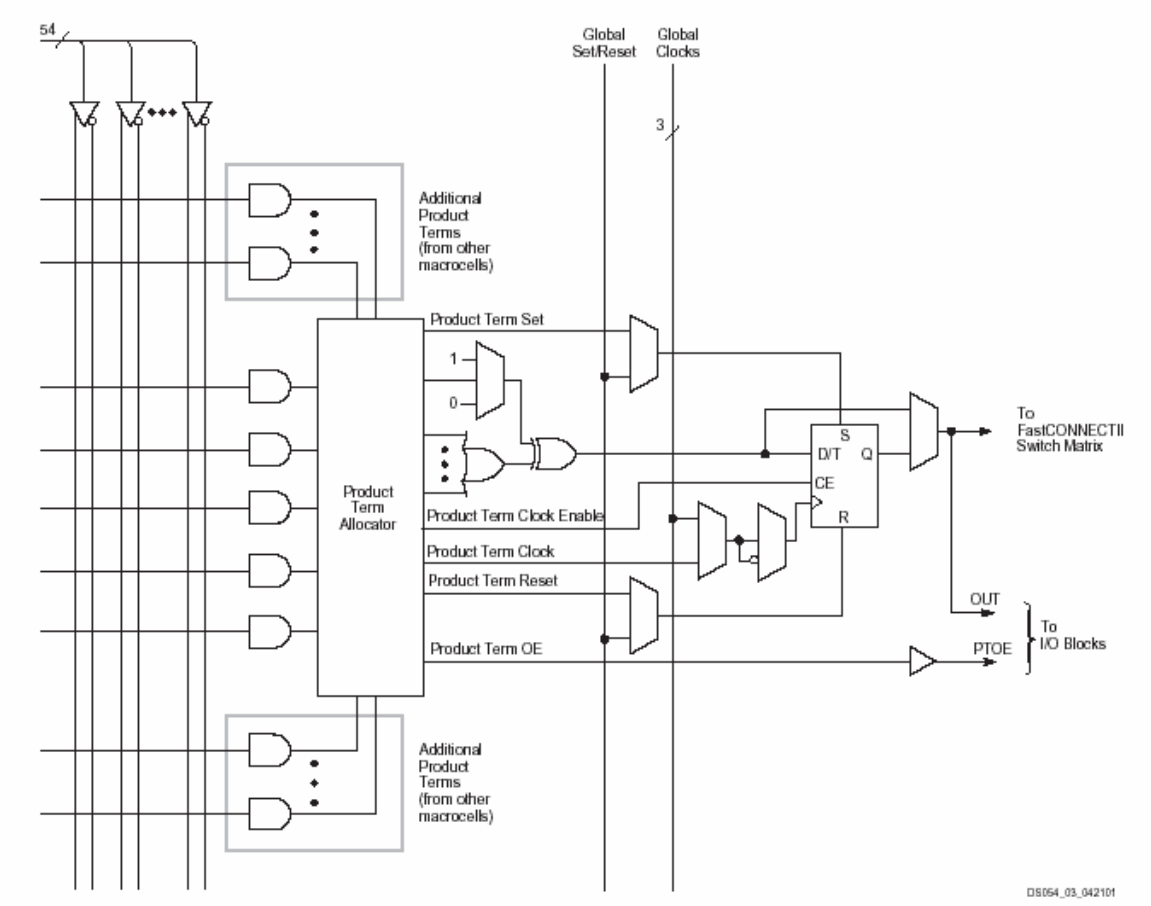

Figure 4: **XC9500XL Macrocell**

The XC9500XL macrocell is very similar in its capabilities to that of CoolRunner-II. P-terms on the left, OR gate, EX-OR gate and flip-flop. The way that it manages assigning p-terms to the OR gate is a bit different, but in general, similar. The next few diagrams expose underlying detail on how the product terms at one macrocell site can be forwarded to a neighboring macrocell. This is important, because as we saw earlier, when building up a single gate at a macrocell, we are not using everything at each

macrocell, so we need to re-allocate it to a neighbor that might better use the left over logic.

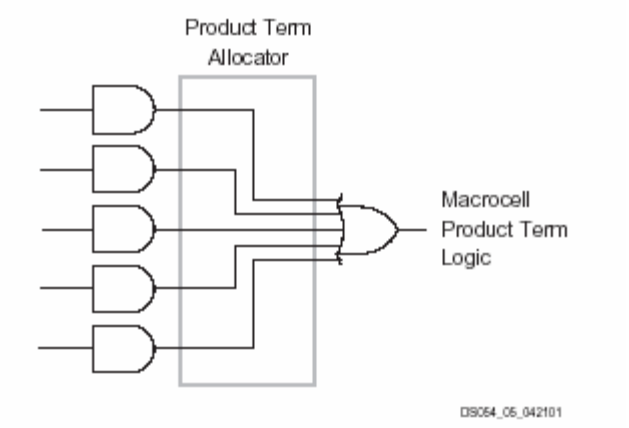

Figure 5: **XC9500/XL/XV Product Term Allocator**

Figure 6 shows how the product terms from adjacent macrocells within the part can be re-assigned (or re-allocated) to another macrocell. In Figure 6, the problem being solved is creating a sum of products function that has 15 AND terms in it, which is fairly rare, but occasionally happens. Collecting neighbor AND gates within the Function Blocks comes with a tiny time penalty in this architecture, but it needs to be understood. Xilinx design software keeps track of the p-term allocation time delays automatically for you, and summarizes them in a timing report.

Just to show the flexibility, Figure 7 shows an 18 product term SOP being built up, which involves collecting AND gates from neighbors within the Function Block that are not directly adjacent, but involve skipping over a macrocell. The architecture is

quite flexible. Figure 5, Figure 6 and Figure 7 are all trimmed down versions of the larger picture of Figure 4, just focusing on the p-term allocation and re-distribution.

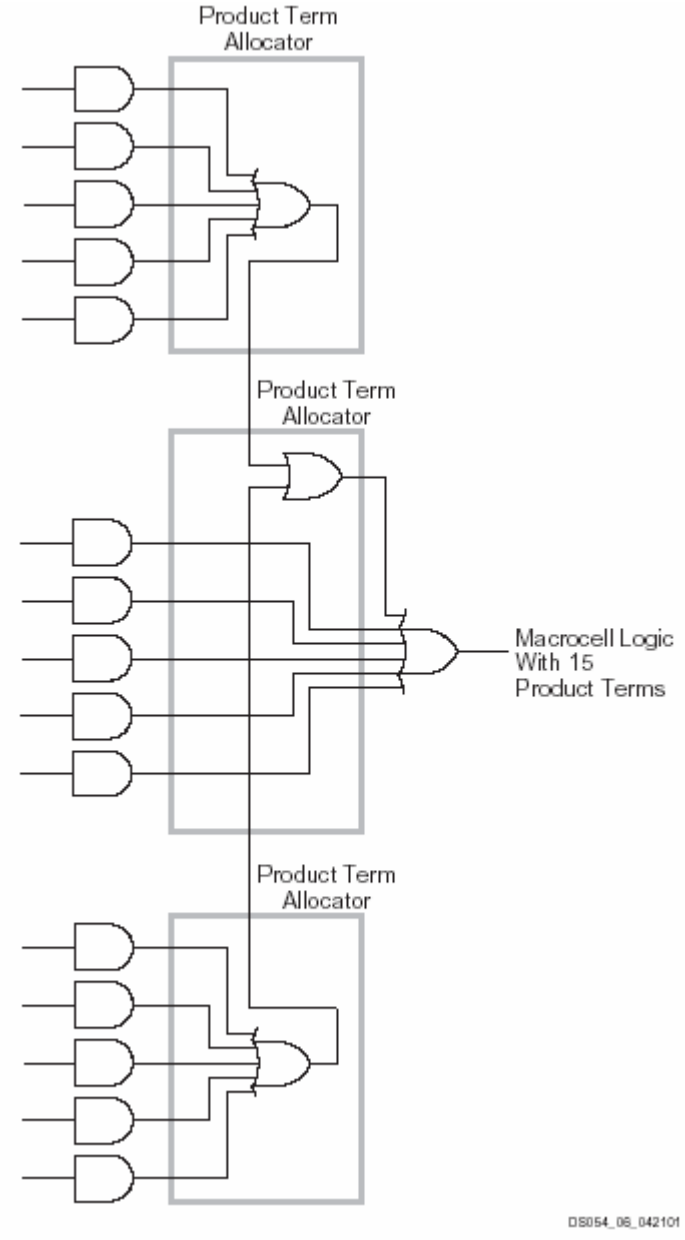

Figure 6: **Collecting Neighboring Product Terms & Providing 15 to a SOP**

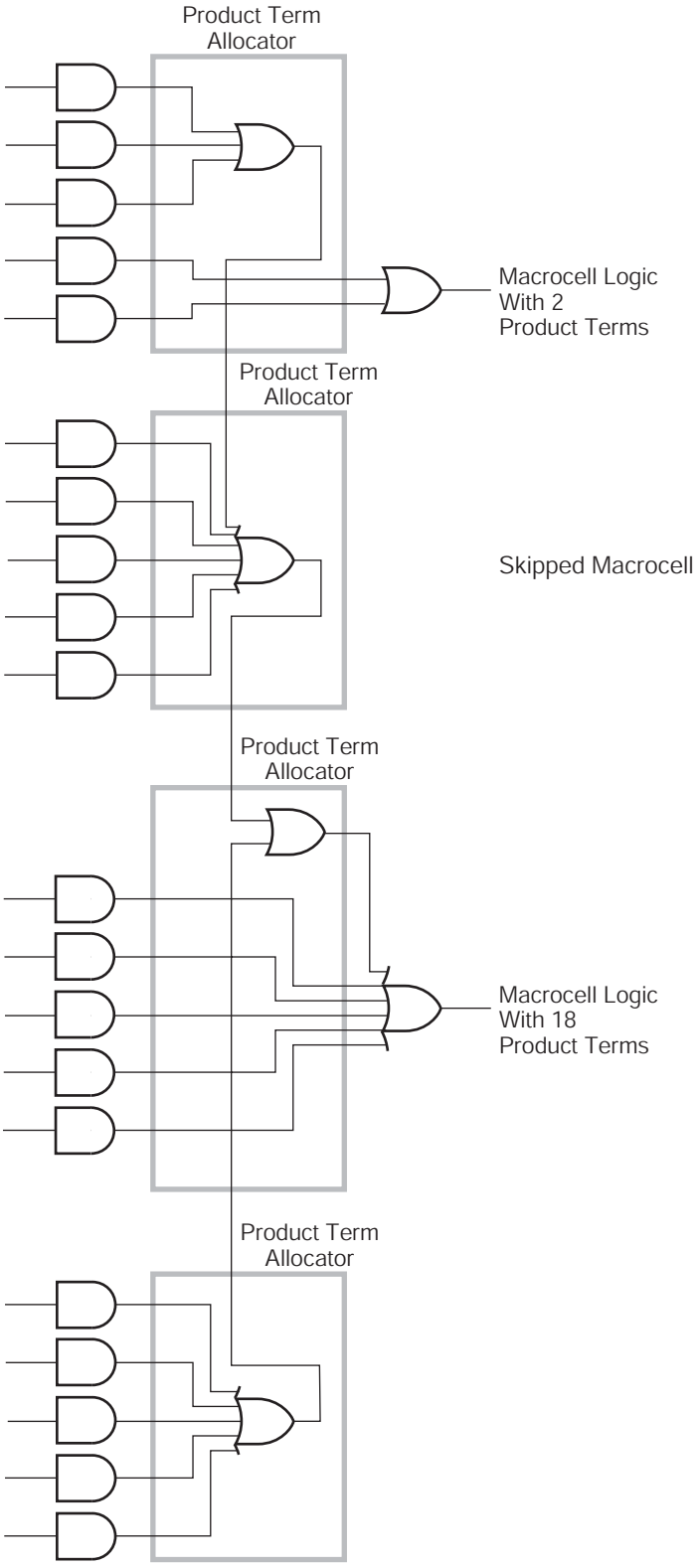

Figure 7: **Taking the SOP Up to 18 P-terms**

We have seen how the macrocells can build up logic functions for small gates and sums of products, but our goal is to have a more systematic process. So, for starters, we will need to know how "big" the parts are. Xilinx CoolRunner-II devices, as a family, are all built from macrocells as shown in Figure 1. They occur in groups called Function Blocks, with 16 macrocells in each FB. The smallest part has 32 macrocells and is called the XC2C32A. The next part has 64 macrocells and is called the XC2C64A. It then goes up to the XC2C128, XC2C256, XC2C384 and XC2C512. In each case, the number of macrocells are the digits to the right of the right most C. The XC9500, XC9500XL, and XC9500XV parts are very similar, having macrocells grouped into 18 macrocell Function Blocks, and the parts typically increase in multiples of 18, starting at say the XC9536. The largest XC9500, XC9500XL, and XC9500XV parts only go up to 288 macrocells, which is smaller than CoolRunner CPLDs.

With this in mind, the approach will be to examine the target 7400 logic and figure how many macrocells get burned through creating an equivalent design. More examples are in order.

Examples

We described simple gates earlier, and found macrocells easily handle small to large gates in a single macrocell. They also handle combinational sum of products chunks, as shown in Figure 6 and Figure 7, in a single macrocell per function.

Registers

Flip-flops – both D and T – are available at the rate of one per macrocell. If you deliver data directly from an input pin to the flip-flop, you do not even burn any p-terms making the delivery. If you deliver data from within the part, then the method of delivery is through an AND gate, which burns a p-term, in doing it. Clocking is usually done with a global clock pin, which burns no AND gates, but the macrocells also let you build logic onto the clock for a flip-flop, using an AND gate, if so needed. So, how much logic you burn making flops depends on what you are doing. Note that transparent D latches can be automatically built from the D/T flip-flops, so basically, you get sn7474 or sn7475 structures inside every macrocell.

Simple State Machines

We cannot really anticipate how you might be making state machines, but usually if its encoded well, you will burn one macrocell per bit of your state machine. To illustrate some basic ones, lets look at a couple of standard shifters and a standard counter.

Shifters

Figure 8 shows a SN74164N Shift register. Usually, this function is delivered in a 14 pin DIP, but modern packaging might also offer it in various SOP packages. It consists of eight consecutively connected flip-flops, which in this case are configured as Set/Reset flip-flops, where the Q of the flop to the left, feeds the Set of the flop to the right, and /Q on the left feeds Reset to the flop on the right. This is equivalent to a

single left flip-flop Q feeding a right flip-flop's D, if you have synchronous D flip-flops, which is the case with CPLDs.

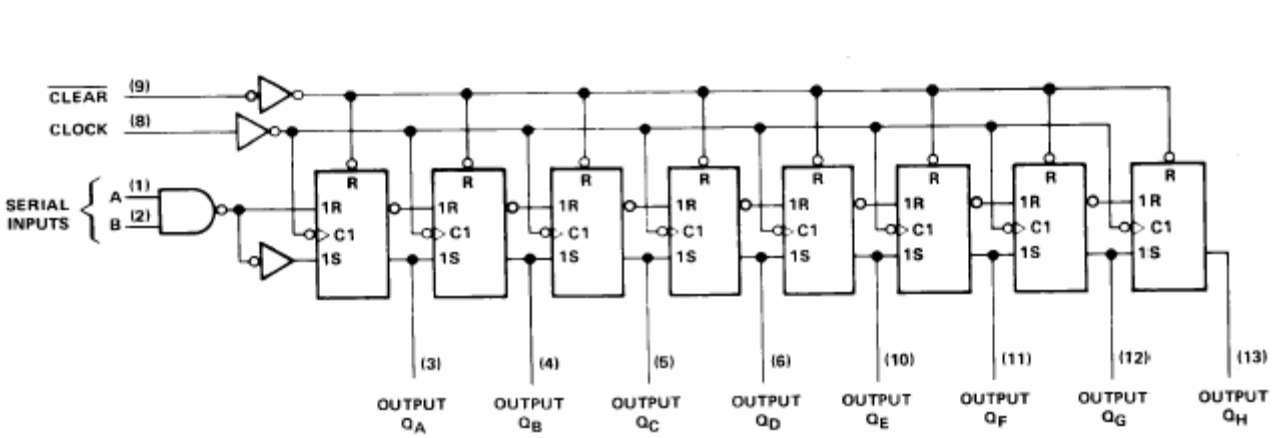

Figure 8: **SN74164 Shifter**

Figure 9 shows how equivalent circuitry would be built up in the CoolRunner-II macrocells, where some of the relevant connection circuitry is shown. In Figure 9, the bottom flip-flop would correspond to the "left" flip-flop in Figure 8. Figure 9 only shows two bits, of the set of eight, but you should get the idea. The additional logic to the left of Figure 8, handling the serial inputs, clocks and asynchronous inputs, are included. Figure 8 would take 8 macrocells to create, but remaining logic would exist.

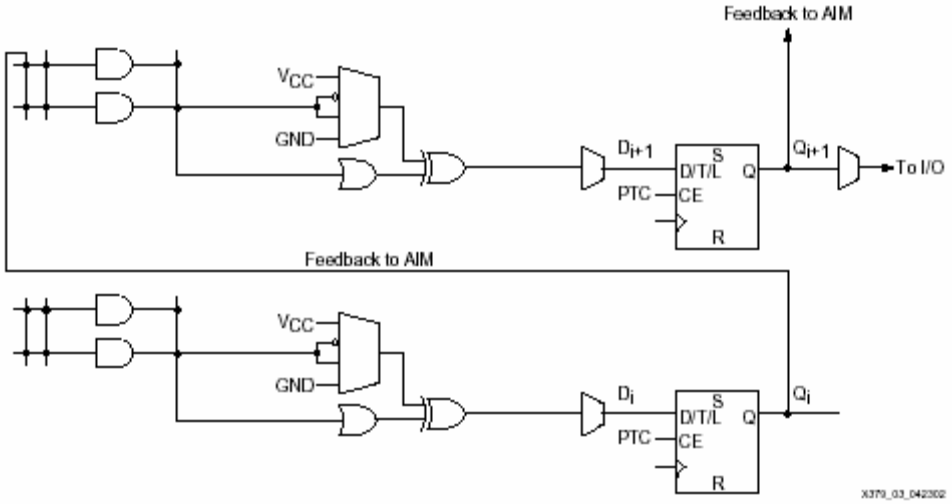

Figure 9: **Building Up Shift Register Bits in Macrocells**

Figure 10 shows another shift register from the 7400 catalog. In this case the data is loaded in parallel, then serialized to a single bit going to the outside world. This would be created on the CPLD with Q feeding D for consecutive flip-flops; but, rather than using the Set and Reset inputs for initializing the contents, the CPLD uses a simple sum of products configuration with a single data value ORed with the

neighboring flip-flop's Q output, then applied to the D input of each flip-flop. The burn rate would be one macrocell per bit.

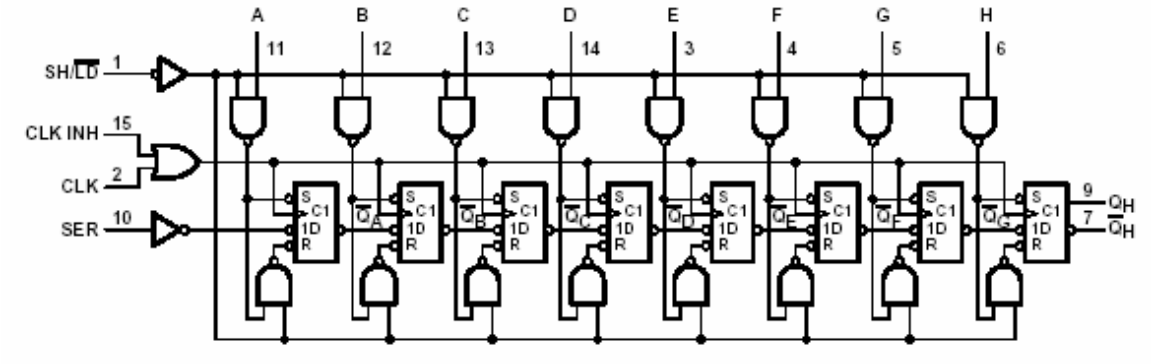

Figure 10: **SN74165 Shifter**

Figure 11 shows a standard SN74161 4 bit cascadable counter.

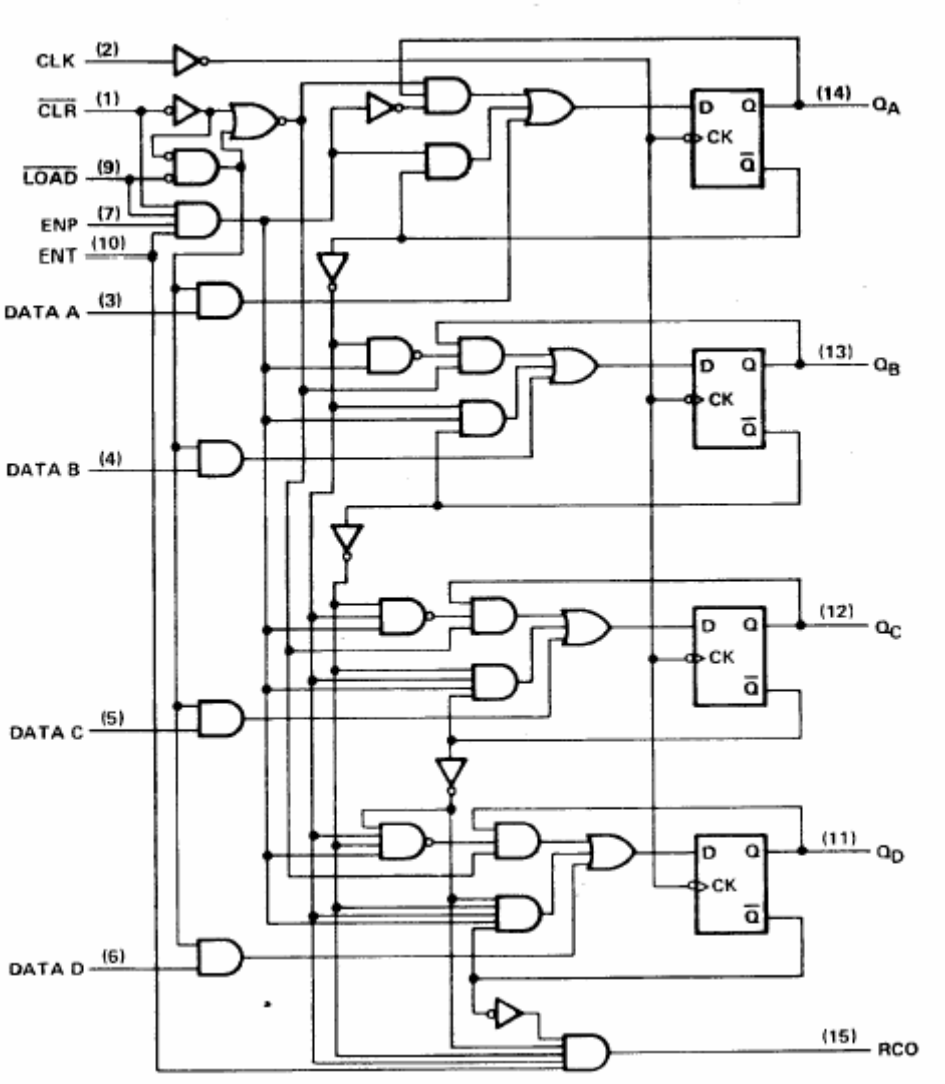

Figure 11: **SN74161 4 Bit Cascadable Counter**

In Figure 11, the chip can be loaded in parallel, and can count up, using the various control signals. Because CoolRunner and XC9500 parts have flip-flops that are D or T, the design software translating this function would save logic by converting the flip-flop to the T configuration. The logic and flip-flop from each cell would build up the bits of the counter. Depending on the desired length, the RCO cascade output would be implemented or not. Typically, it would fold that logic into the next higher bit (fifth) when cascading.

These examples are selected to give a rough feel for how to account for the logic needed to build up 7400 parts. It is part "art", part "science" and part intuition. Experience in transforming logic frequently shows that many logic functions easily fit into a CPLD. A key point is to not force the design software to "pinout" every gate output. Be sure to only pinout the functions needed, as intermediate sites can defeat the ability to pack the CPLDs for your best interests.

Burn Rate Table

A burn rate table is presented in Table 1, which identifies how many macrocells would get used to create various functions.

Table 1: **Macrocell "Burn Rate" for Common TTL Functions**

Function	Macrocells	P-terms	Flip-Flops
Shift register (simple)	1 per bit	1 per bit	1 per bit
Counter (simple)	1 per bit	1 per bit	1 per bit
2:1 Mux	1	2	0
4:1 Mux	1	4	0
8:1 Mux	1	8	0
8 bit loadable shifter	8	16	8
8 bit loadable / SL / SR shifter	8	24	8
8 bit loadable counter	8	16	8
8 bit load / up / dn counter	8	24	8
Full Adder / bit	2	7	0 / 1 (optional)
2:4 Decoder	4	4	0
3:8 Decoder	8	8	0
4:16 Decoder	16	16	0
8 bit Equality Comparator	1	16	0
And / Nand gate (1-40 inputs)	1	1	0
Or / Nor gate (1-40 inputs)	1	11	0
Ex-or / Ex-nor (2-3 inputs)	1	2-3	0
Level translator (per bit)	1	1	0

A new tool, the CPLD Logic Consolidator is designed to approximate the macrocell count for you. In theory, you should be able to identify various functions on your PCB, then, knowing what they do, look them up in this table and calculate how many

macrocells you would use. Go through your design tallying them up. Then, knowing the various CPLD part numbers, you can identify a part by the number of needed macrocells. It is sometimes cheaper to use multiple small CPLDs versus few larger ones, but depending on your target – cheapest price, smallest power consumption, or minimum board area, will dictate the choice. The Logic Consolidator is available from Xilinx for free at **http://www.xilinx.com/products/cpldsolutions/logic_tool.htm**.

Conclusion

It is hoped that the ideas presented have encouraged you to start on your adventure of consolidating outdated 7400 logic into modern Xilinx CPLDs. The cost savings, improved performance and lower power should make it worth your while. Xilinx WebPack ISE software is freely available on the Xilinx Web site to aid you in your task. Many Xilinx application notes, white papers and tutorials are available at the same site.

Additional Information

CoolRunner-II Data Sheets and Application Notes

XC9500XL Data Sheets and Application Notes

Online Store

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/10/05	1.0	Initial Xilinx release.