requirements, the chip generates an internal CAS signal at a later clock tick. How many ticks later is a function of the CLK frequency and the speed grade of the SDRAM chip itself. To accommodate differing requirements, the RAS-to-CAS delay, called the CAS latency, is programmable. This and several other important operating parameters must be downloaded into an SDRAM at initialization. Downloading is fairly easy; the "load parameters" command is recognized when RAS_L, CAS_L, and WE_L control signals are all asserted simultaneously, and the parameters themselves are transferred on the address lines.

10.5 Complex Programmable Logic Devices

Since their introduction years ago, programmable logic devices such as the 16V8 and 22V10 have been very flexible workhorses of digital design. As IC technology advanced, there was naturally great interest in creating larger PLD architectures to take advantage of increased chip density. The question is, why didn't manufacturers just scale the existing architectures?

For example, if DRAM densities increased by a factor of 64 over the last 10 years, why couldn't manufacturers scale the 16V8 to create a "128V64"? Such a device would have 64 input pins, 64 I/O pins, and some number of 128-variable product terms (say, 8) for each of its 128 logic macrocells. It could combine the functions of a larger collection of 16V8s and offer terrific performance and flexibility in using any input in any output function.

Or could it? Yes, a 128V64 would be very flexible, but it would not have very good performance. Unlike the 16V8, which has 32 inputs (16 signals and their complements) per AND term, this device would have 256. Due to capacitive effects, leakage currents, and so on, such a large wired-AND structure would be at least eight times slower than the 16V8's AND array.

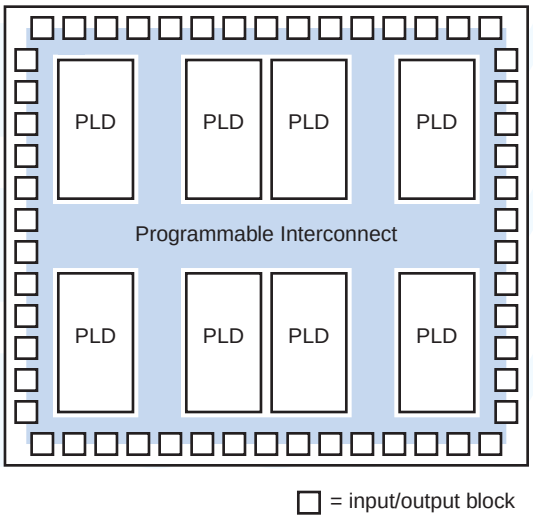

Figure 10-37 General CPLD architecture.

Worse from a manufacturer's point of view, a 128V64 would not make very cost-effective use of chip area. It would use about 64 times the chip area of a 16V8, but provide the same number of inputs and outputs as only eight 16V8s. That is, for n times as much logic (in terms of inputs, outputs and AND terms), the 128V64 uses n^2 as much chip area. In terms of efficiency of silicon use, a clever designer would be better off partitioning a desired function into eight 16V8s, if such a partitioning were possible.

This is where the idea of complex programmable logic devices (CPLDs) came from. As shown in Figure 10-37, a CPLD is just a collection of individual PLDs on a single chip, accompanied by a programmable interconnection structure that allows the PLDs to be hooked up to each other on-chip in the same way that a clever designer might do with discrete PLDs off-chip. Here, the chip area for n times as much logic is only n times the area of a single PLD plus the area of the programmable interconnect structure.

Different manufacturers have taken many different approaches to the general architecture shown in the figure. Areas in which they differ include the individual PLDs (AND array and macrocells), the input/output blocks, and the programmable interconnect. We'll discuss each of these areas in the rest of this section, using the Xilinx 9500-series CPLD architecture as a representative example.

10.5.1 Xilinx XC9500 CPLD Family

The Xilinx XC9500 series is a family of CPLDs with a similar architecture but varying numbers of external input/output (I/O) pins and internal PLDs (which Xilinx calls function blocks—FBs). As we'll see later, each internal PLD has 36 inputs and 18 macrocells and outputs and might be called a "36V18." As shown in Table 10-8, devices in the family are named according to the number of macrocells they contain. The smallest has 2 FBs and 36 macrocells, and the largest has 16 FBs and 288 macrocells.

Another important feature of this and most CPLD families is that a given chip, such as the XC95108, is available in several different packages. This is important not only to accommodate different manufacturing practices but also to provide some choice and potential savings in the number of external I/O pins provided. In most applications, it is not necessary for all internal signals of a state machine or subsystem to be visible to and used by the rest of the system.

So, even though the XC95108 has 108 internal macrocells, the outputs of at most 69 of them can be connected externally in the 84-pin-PLCC version of the device. In fact, many of the 69 I/O pins would typically be used for inputs, in which case even fewer outputs would be visible externally. That's OK; the remaining macrocell outputs are still quite usable internally, since they can be hooked up internally through the CPLD's programmable interconnect. Macrocells whose outputs are usable only internally are sometimes called *buried macrocells*.

buried macrocell

Table 10-8 Function blocks and external I/O pins in Xilinx 9500-series CPLDs.

	Part Number					
	XC9536	XC9572	XC95108	XC95144	XC95216	XC95288
FBs / macrocells	2 / 36	4 / 72	6 / 108	8 / 144	12 / 216	16 / 288
Package	**Device I/O Pins**					
44-pin VQFP	34					
44-pin PLCC	34	34				
48-pin CSP	34					
84-pin PLCC		69	69			
100-pin TQFP		72	81	81		
100-pin PQFP		72	81	81		
160-pin PQFP			108	133	133	
208-pin HQFP					166	168
352-pin BGA					166	192

Another important dimension of Table 10-8 is the horizontal one. All but two of the packages support at least two different devices in the same package and, as it turns out, with compatible pinouts. This is a life-saver in making last-minute design changes. For example, suppose you were to target a design to an XC9572 in an 84-pin PLCC. You might find that the 69 I/O pins of the device are quite adequate. You would like to use the XC9572 for its low cost, but you might be a little nervous if your initial design used 68 out of the 72 available internal macrocells (I know that I would be!). Based on Table 10-8, you can sleep well knowing that if bug fixes or design-specification changes necessitate a more complex design internally, you can always move up to the XC95108 in the same package and pick up another 36 macrocells.

Figure 10-38 is a block diagram of the internal architecture of a typical XC9500-family CPLD. Each external I/O pin can be used as an input, an output, or a bidirectional pin according to the device's programming, as discussed later. The pins at the bottom of the figure can also be used for special purposes. Any of three pins can be used as "global clocks" (GCK); as we'll see later, each macrocell can be programmed to use a selected clock input. One pin can be used as a "global set/reset" (GSR); again, each macrocell can programmably use this signal as an asynchronous preset or clear. Finally, two or four pins (depending on device) can be used as "global three-state controls" (GTS); one of these signals can be selected in each macrocell to output-enable the corresponding output driver when the macrocell's output is hooked up to an external I/O pin.

Only four FBs are shown in the figure, but the XC9500 architecture scales to accommodate 16 FBs in the XC95288. Regardless of the specific family

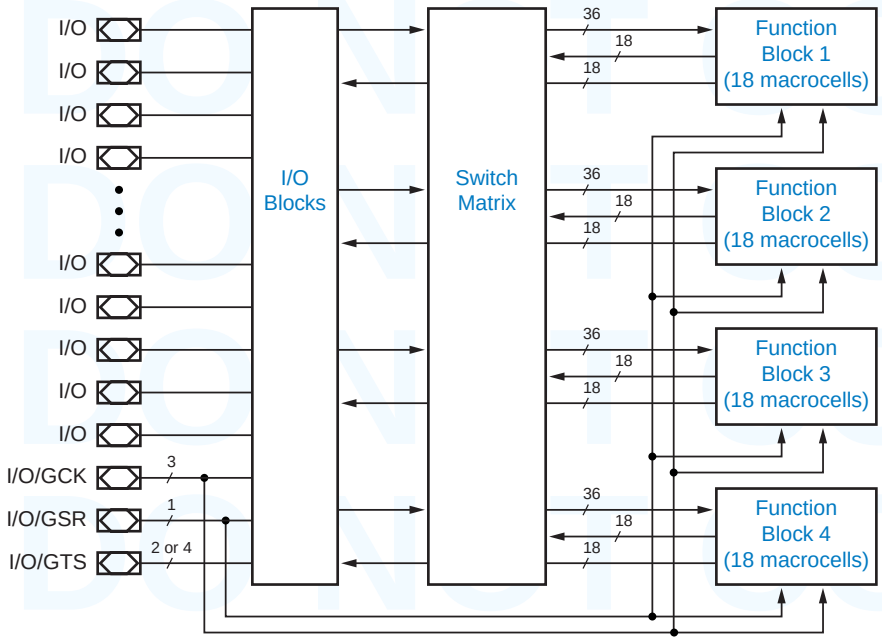

Figure 10-38 Architecture of Xilinx 9500-family CPLDs.

member, each FB programmably receives 36 signals from the switch matrix. The inputs to the switch matrix are the 18 macrocell outputs from each of the FBs and the external inputs from the I/O pins. We'll say more about how the switch matrix hooks things up in Section 10.5.4.

Each FB also has 18 outputs that run "under" the switch matrix as drawn in Figure 10-38 and connect to the I/O blocks. These are merely the output-enable signals for the I/O-block output drivers; they're used when the FB macrocell's output is hooked up to an external I/O pin.

10.5.2 Function-Block Architecture

The basic structure of an XC9500 FB is shown in Figure 10-39. The programmable AND array has just 90 product terms. Compared to 16V8- and 22V10-style PLDs, the XC9500 and most CPLD macrocells have fewer AND terms per macrocell—where the 16V8 has 8 and the 22V10 has 8–16, the XC9500 has only 5. However, this is not all that bad, because of *product-term allocation*. The

product-term allocation

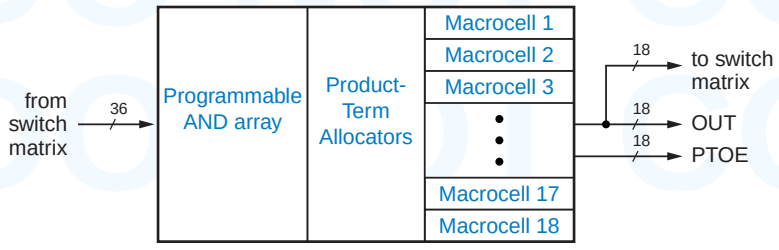

Figure 10-39 Architecture of function block (FB).

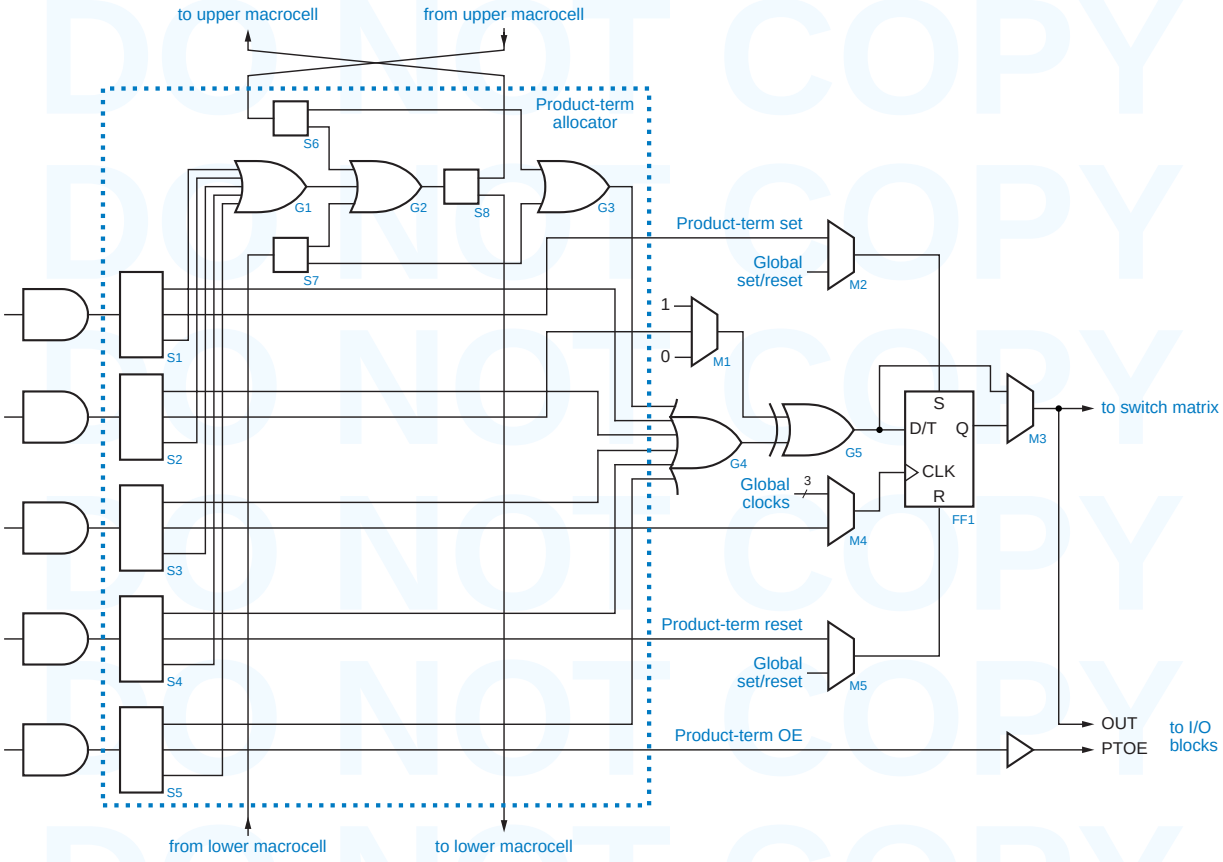

Figure 10-40 XC9500 product-term allocator and macrocell.

product-term allocator

XC9500 and other CPLDs have *product-term allocators* that allow a macrocell's unused product terms to be used by other nearby macrocells in the same FB.

Figure 10-40 is a logic diagram of the XC9500 product-term allocator and macrocell. In this figure, the rectangular boxes labeled S1–S8 are programmable signal-steering elements that connect their input to one of their two or three outputs. The trapezoidal boxes labeled M1–M5 are programmable multiplexers that connect one of their two to four inputs to their output.

The five AND gates associated with the macrocell appear on the lefthand side of the figure. Each one is connected to a signal-steering box whose top output connects the product term to the macrocell's main OR gate G4. Considering just this, only five product terms are available per macrocell. However, the top, sixth input of G4 connects to another OR gate G3 that receives product terms from the macrocells above and below the current one.

Any of the macrocell's product terms that are not otherwise used can be steered through S1–S5 to be combined in an OR gate G1 whose output can eventually be steered to the macrocell above or below by S8. Before steering,

ONE WAY In a given XC9500 macrocell, you wouldn't normally steer daisy-chained product terms back in the direction from which they came. For example, if product terms are arriving at S6 from above, we can use them locally by having S6 steer them to G3, or S6 can steer them to G2. In the latter case, S8 should steer the output of G2 down to the lower macrocell; there's no point in steering the product terms back up to the upper macrocell. In fact, if S6 and S8 in this macrocell were steering product terms up, and S7 and S8 the macrocell above were steering product terms down, we would have a nasty loop.

these product terms may be combined with product terms from below or above through S6, S7, and G2. Thus, product terms can be "daisy-chained" through successive macrocells to create larger sums of products. In principle, all 90 product terms in the FB could be combined and steered to one macrocell, although that would leave 17 out of the FB's 18 macrocells with no product terms at all.

Besides depriving other macrocells of product terms, daisy-chaining terms has an additional price. A small additional delay is incurred for each "hop" made by steered product terms. This delay can be minimized by careful allocation of product-term-hungry macrocells so they are adjacent to macrocells with low product-term requirements. For example, a macrocell can make use of 13 product terms with only one extra hop delay if it is positioned between two macrocells that use only one product term each.

The third, middle choice for each of the steering boxes S1–S5 is to use the product term for a "special function." The special functions are flip-flop clock, set, and reset; XOR control; and output enable. Most of these special functions are not normally used.

Getting closer to the heart of the macrocell, OR gate G4 forms a sum-of-products expression using all selected product terms and feeds it into XOR gate G5. The other input of G5 can be 0, 1, or a product term, as selected by multiplexer M1. Setting this input to 1 inverts G4's sum-of-products expression, so the macrocell can be configured to use either polarity of minimized logic equations. Setting this input to a product term is useful in the design of counters. The product term is arranged to be 1 when the lower-order counter bits are 1 and counting is enabled, and the output of G4 is arranged to be the current value of the counter bit; thus, the counter bit is complemented as required.

The macrocell's flip-flop FF1 can be programmed to behave either as a D flip-flop or as a T flip-flop with enable; the latter is also useful in some styles of counter realization. Multiplexer M4 selects the flip-flop's clock input from one of four sources—the CPLD's three global clock inputs or a product term. This last choice is a no-no in synchronous design methodologies, except in well-defined synchronization applications, as in Figure 8-111 on page 784.

The flip-flop also has asynchronous set and reset inputs with input sources controlled by multiplexers M2 and M5. In most applications, set or reset would be connected to the CPLD's global set/reset input and would be used only at system initialization. However, these inputs can also be used to access an S-R latch in which the CLK input is not used, or, if you're very careful, you can use CLK and S or R in synchronization applications as in Figure 8-111.

A final multiplexer M3 selects either the flip-flop output or its data input to be used as the macrocell output, OUT. This signal is sent to the switch matrix, where it can be used by any other macrocell. It is also sent to the I/O blocks, along with a product term selected by S5 that can be used as the output-enable signal PTOE if needed.

10.5.3 Input/Output-Block Architecture

I/O block (IOB)

The structure of the XC9500 *I/O block (IOB)* is shown in Figure 10-41. There are seven, count them, seven choices of output-enable signals for the three-state driver buffer. It can be always on, always off, controlled by the product term PTOE from the corresponding macrocell, or controlled by any of up to four global output enables. The global output enables are selectable as active-high or active-low versions of the external GTS pins.

The XC9500's IOB is a good example of an important trend in CPLD and FPGA I/O architectures—providing many "analog" controls in addition to "logic" ones like output enables. Three different analog controls are provided:

- *Slew-rate control*. The rise and fall time of the output signals can be set to be fast or slow. The fast setting provides the fastest possible propagation delay, while the slow setting helps to control transmission-line ringing and system noise at the expense of a small additional delay.
- *Pull-up resistor.* When enabled, the pull-up resistor prevents output pins from floating as the CPLD is powered up. This is useful if the outputs are used to drive active-low enable inputs of other logic that is not supposed to be enabled during power up.
- *User-programmable ground*. This feature actually reallocates an I/O pin to be a ground pin, not a signal pin at all. This is useful in high-speed, high-slew-rate applications. Extra ground pins are needed to handle the high dynamic currents that flow when multiple outputs switch simultaneously.

In addition to these features, the XC9500 family provides compatibility with both 5-V and 3.3-V external devices. The input buffer and the internal logic run from a 5-V power supply (V_{CCINT}). Depending on the operating voltage of external devices, the output driver uses either a 5-V or a 3.3-V supply (V_{CCIO}). Notice that the pull-up resistor pulls to the I/O supply voltage, V_{CCIO}. Diodes *D1* and *D2* are used to clamp voltages above V_{CCINT} or below ground that can occur due to transmission-line ringing.

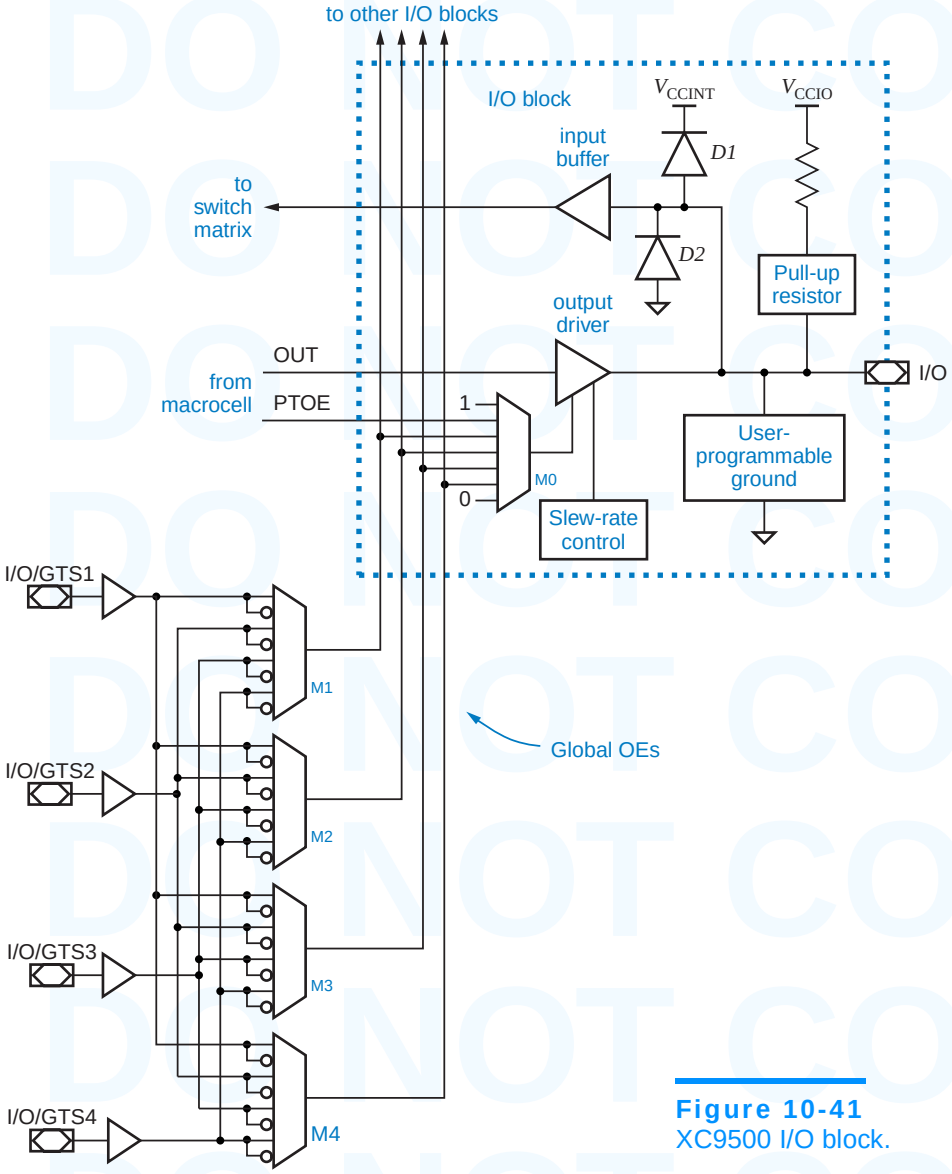

Figure 10-41
XC9500 I/O block.

10.5.4 Switch Matrix

Theoretically, a CPLD's programmable interconnect should allow any internal PLD output or external input pin to be connected to any internal PLD input. Likewise, it should allow any internal PLD output to connect to any external output pin. But if you think about this, you'll see that we're back to the same n^2 problem that we would have in building a 128V64.

The case of a typical Xilinx XC9500 family member, the XC95108, is shown in Figure 10-42. There are 108 internal macrocell outputs and 108 external-pin inputs, a total of 216 signals which should be connected as inputs to the switch matrix. Since the XC95108 has 6 FBs with 36 inputs each, the switch matrix should have (coincidentally) 216 outputs, each of which is a 216-input multiplexer driving one input of an FB's AND array.

A switch matrix such as the one shown in the figure can be built in a chip as a rectangular structure, with a column for each input, a row for each output, and a pass transistor (or transmission gate) at each crosspoint to control whether a given input is connected to a given output. But this is still a big structure—216 columns and 216 columns in the example.

With today's high-density IC technology, the problem is not so much size but speed. Having a large number of transistors connected to each row or column makes for high capacitance, which makes for slow speed. Therefore, CPLD manufacturers look for ways to reduce the size of the switch matrix.

For example, we can observe in Figure 10-42 that not every switch-matrix input must be connectable to every output. We only need every input to be connectable to *some* input of each FB, since any FB input can connect to any AND gate within the FB's AND array.

Then again, the requirement is not quite as simple as we just stated. Suppose that the switch-matrix output for FB input i ($0 \le i \le 35$) is created by a simple 8-input multiplexer whose inputs are switch-matrix inputs $8i$ through $8i + 7$. There are many interconnection patterns that cannot be accommodated by such a switch matrix. For example, connecting switch-matrix inputs 0–35 to the same FB would not be possible. As soon as switch-matrix input 0 is connected to FB input 0, switch-matrix inputs 1–7 are blocked.

Thus, the switch-matrix connectivity requirement must be stated more broadly: For each FB, any combination of switch-matrix inputs must be connectable to some combination of the FB's inputs.

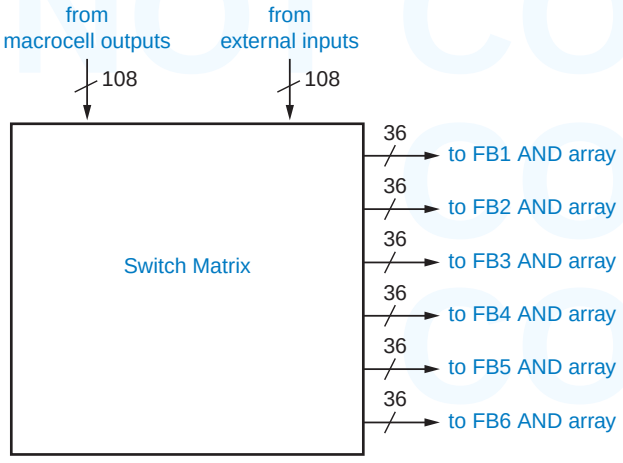

Figure 10-42 XC95108 switch-matrix requirements.

A typical CPLD switch matrix is a compromise between the minimal multiplexer scheme and a full, nonblocking crosspoint array. With anything less than a nonblocking crosspoint array, the problem of allocating input-output connections in the switch matrix is nontrivial. For each different CPLD-based design, a set of switch-matrix connections must be found by "fitter" software provided to the designer by the CPLD's manufacturer.

Finding a complete set of connections through a sparse switch matrix is one of those *NP*-complete problems that you hear about in computer science. In lay terms, that means that for some designers, the fitter software may have to run a lot longer than you care to wait to find out whether there is a solution. If the switch matrix has too few crosspoints, as in our minimal multiplexer example, even the best fitter software running "forever" will not be able to find a complete set of connections for some designs.

Thus, the design of a CPLD's switch matrix is a compromise between chip performance (speed, area, cost) and fitter-software capabilities. The fitter software usually determines not only the final connections through the switch matrix, but also the assignment of CPLD inputs and outputs to FBs, macrocells, and external pins, and of "buried logic" to FBs and macrocells. These assignments interact in turn with both the switch-matrix-connection and product-term allocation problems. The solutions to these problems are the "secret sauce" of CPLD chip and software design, and are not typically disclosed by CPLD manufacturers.

PIN LOCKING

Another important issue in CPLD chip and software design is *pin locking*. In most CPLD applications, it's OK to let the fitter software pick any pins that it likes for the device's external input and output signals. However, once the design is complete and a PCB has been fabricated, the designer would like to "lock down" the pin assignments so they remain the same even if small (or large!) changes are made to the design for bug-fixing purposes. This spares everyone the time, expense, and hassle of reworking or redesigning and refabricating the PCB.

"Locked" pin assignments are typically specified in a file that is read by the fitter software. With early CPLDs and FPGAs, locking down pins after making a small change did not guarantee success—the fitter would throw up its hands and complain that it was too constrained. If you unlocked the pins, the fitter could find a new allocation that worked, but it might be completely scrambled from the original.

These problems were not necessarily the fault of the fitter software; the CPLDs and FPGAs simply did not have rich enough internal connectivity to support frequent design changes under the constraint of pin locking. The device manufacturers have learned from this experience and improved their internal device architectures to accommodate frequent design changes. For example, some devices contain an "output switch matrix" that guarantees that any internal macrocell signal can be connected to any external I/O pin.