Designing with FPGAs Compared with PLD Devices

Field programmable gate arrays (FPGAs) are powerful devices for implementing complex digital systems. FPGAs are best used with an understanding of the key differences between FPGAs and previous logic technologies. This document focuses on FPGAs compared with programmable logic devices (PLDs). Understanding these differences and using design techniques appropriate for FPGAs results in 50 to 100 percent improvement in speed and density compared with design styles that treat FPGAs and PLDs equally.

Comparing Technologies

PLDs are array-oriented devices that typically have an AND-OR structure with wide-input AND gates feeding a narrower OR gate. A register is typically available at the output of each OR. This architecture is termed *logic rich* because there are typically many more logic gates than registers available. The ratio of gates to registers can be as high as 5 to 1. Because of the large delay through the wider logic module, PLDs pay a significant speed penalty when multiple levels of logic are required. Speeds tend to be more predictable in PLDs because of the larger *speed quanta*.

FPGAs on the other hand, are register rich, with a logic-to-register ratio closer to 2 to 1. (This ratio is equivalent to the traditional gate array usage and tends to be related to high density designs' need for more registers than are needed by the traditional "glue logic" oriented low-density applications.) FPGA logic structures are optimized for functions narrower than those of PLDs. FPGAs have a smaller speed quanta than do PLDs, so logic functions can be incremented in complexity while incrementing the delay only a little each time. In addition, signals that need to be fast can be sourced near the bottom of the logic tree, minimizing the number of logic levels required, and slow signals can be sourced at the top of the logic tree, where more logic levels are required.

Estimating PLD Logic Replacement

Estimating the number of logic modules needed to replace an existing PLD is straightforward. Figures 1 shows an example of a typical PLD file converted to Actel logic modules. Typical PLD functions are address decoding and state machine control. Figure 1 is a wide-register AND function decoding 25 inputs and using only 6 logic modules. Since the first AND5B gate is combined with the DFC1B flip-flop in a sequential module, the 25-bit decode is done in only one logic level. A second level could be added to provide 32-bit decode with 8 modules or 64-bit decode with 16 modules.

Figure 2 is a familiar three-product term AND/OR array using only four logic modules to implement a typical state machine equation. Figure 3 shows the product and sum terms expanded further. In the PLD equation examples, note how easily the number of either product or sum terms can be expanded. Small incremental changes in both delay and size are added each time the array is enlarged. This is in sharp contrast to the large step-function increase in delay and size when a second PLD array must be used to implement a particular equation.

PLD equation conversion shows the flexibility of the Actel logic module, implementing a variety of different combinatorial and sequential functions in few modules without wasting dedicated resources. With ACT 2, 1200XL, 3200DX, and ACT 3 devices, the cost of using a sequential function is equal to the cost of using a combinatorial function. In contrast, PLDs are rich in AND/OR gating but lacking in registers. Encoding states in a PLD requires using the smallest number of flip-flops possible, but with Actel you are free to use any combination of flip-flops and logic.

State Machine Techniques

The traditional PLD design techniques for implementing state machines are geared toward the logic-rich and register-lean architecture of the standard PLD. A small number of state registers are used (usually the theoretical minimum), since registers are scarce. This approach requires a larger amount of combinatorial logic to decode the state, but PLDs usually are able to provide enough combinatorial logic to do this effectively. Using this technique for FPGAs would not be an efficient use of FPGA strengths—numerous registers and fast narrow logic gates. A bit-per-state approach, whereby each state uses a separate register instead of encoding states in multiple registers, results in faster and more efficient state machines in FPGAs. (Refer to the "Designing State Machines for FPGAs" application note in this data book.) In many cases, speed improves by 50 to 100 percent compared with the PLD-oriented methodology of an encoded state machine.

In PLD-oriented designs, logic is typically used to develop outputs from state machines. Usually this requires an additional level of logic after the state register and adds

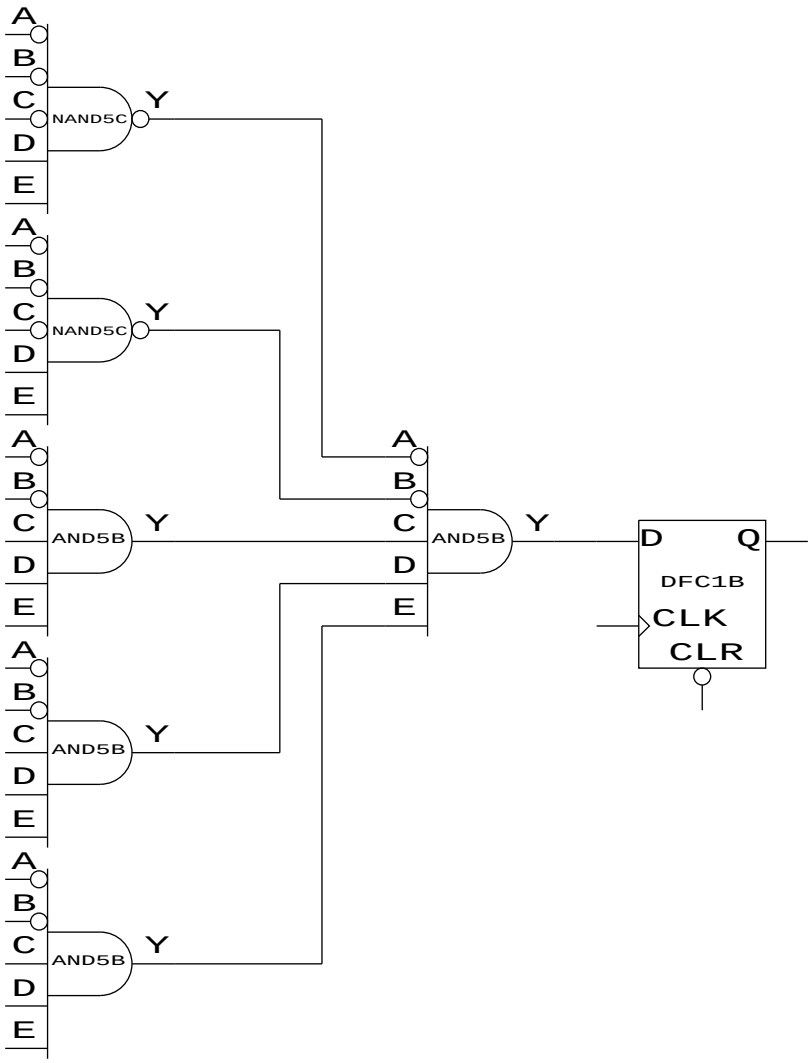

Figure 1 • *PLD Implementation with Wide Fan-In*

delay. In FPGAs, this level of logic can be eliminated in many cases by combining the logic in front of the state bits in which an output is active. For example, if the chip enable (CE) output from a state machine needs to be active in states 3 and 5, the logic feeding state bits 3 and 5 can be ORed together and registered to create the CE output without incurring a logic delay after the register. Since the logic in front of state bits is simple, usually no additional delay or logic resources are required in front of the new register.

Another popular state machine design technique for PLDs uses counters to generate a sequence of wait states. For example, a state machine may need to wait for 16 cycles until a data transfer can begin. A 4-bit counter can be used to generate the required state sequence. This is fairly efficient in PLD architectures because of the logic-rich and register-lean characteristics of the count function. It is not as good a fit for FPGAs, however. In FPGAs, registers are rich, and a shift register is more efficient and faster than a counter. A normal shift register will require one register per wait state. If very large delays are required, a feedback shift register can be used to implement only one state fewer than a counter, but it requires much less logic and is significantly faster.

Actel provides an automated tool (ACTgen Macro Builder) to assist designers with the creation of counters, shift registers, and feedback shift registers. These functions can be used to augment state machine designs and simplify the design process.

Conclusion

PLD designers can use FPGA technology to reap the benefits of lower cost, smaller board size, and lower power. However, using new techniques that are better suited for FPGAs will allow between 50 and 100 percent improvement in performance and capacity.

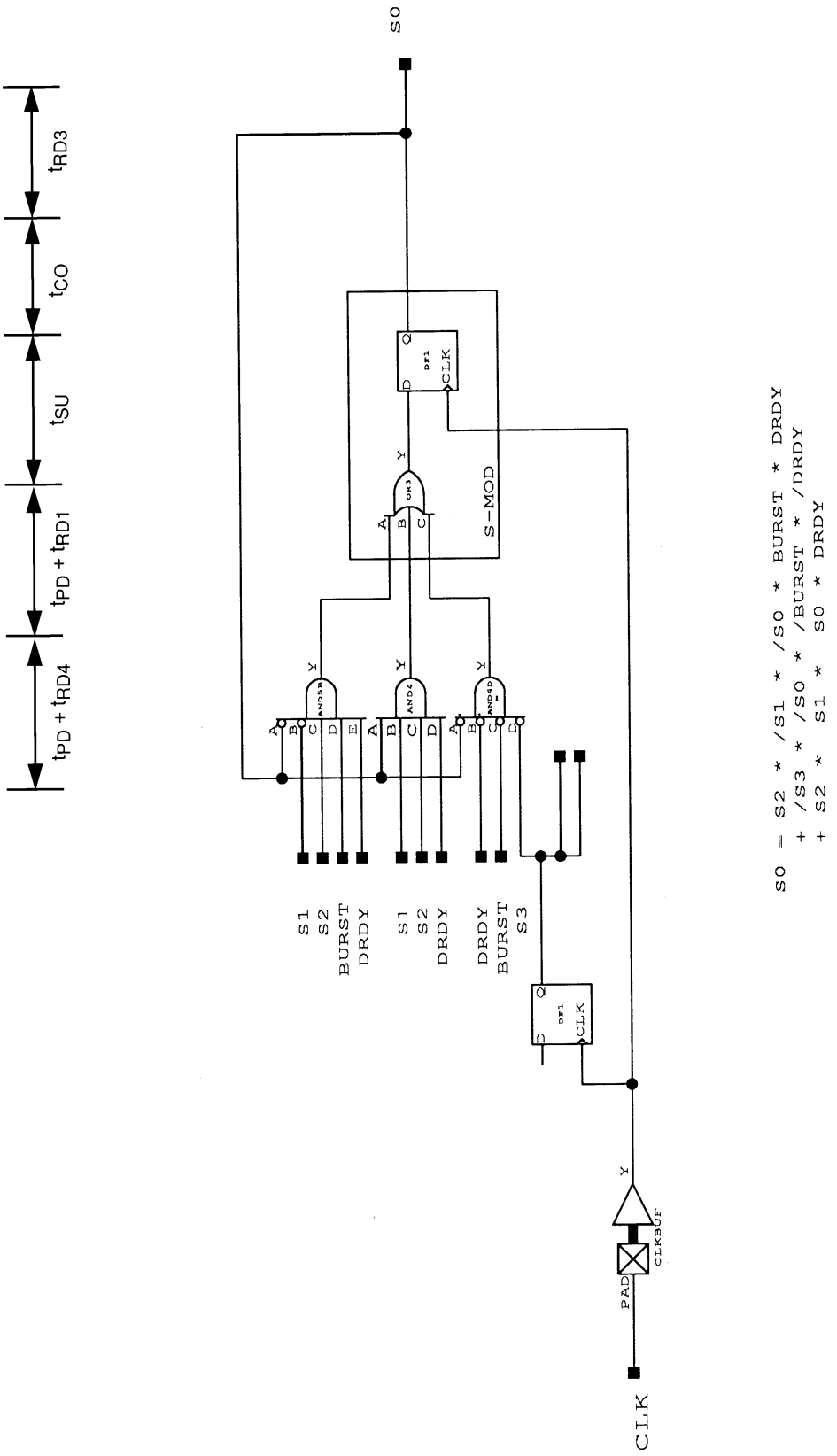

Figure 2 • *Implementing State Machine*

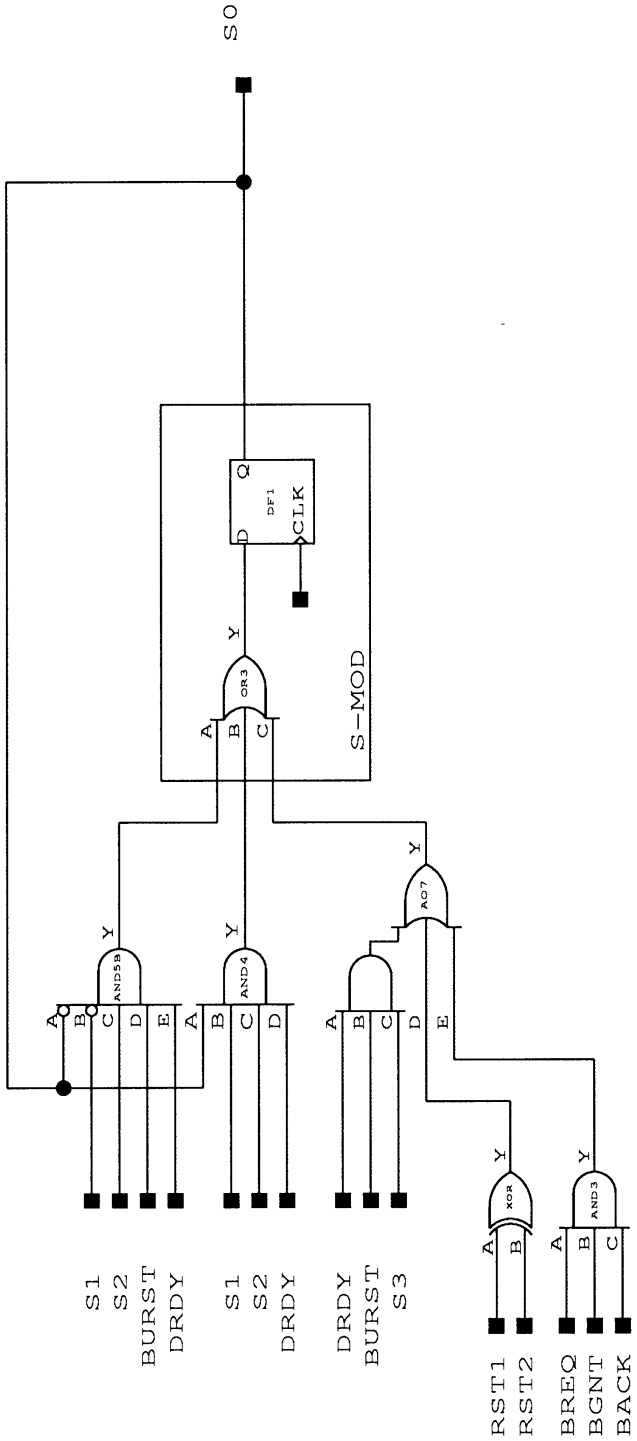

Figure 3 • *Expanded State Machine*