Exercises

6.1 Explain how the 16-bit barrel shifter of Section 6.1.1 can be realized with a combination of 74x157s and 74x151s. How does this approach compare with the others in delay and parts count?

6.2 Show how the 16-bit barrel shifter of Section 6.1.1 can be realized in eight identical GAL22V10s.

6.3 Find a coding of the shift amounts (`S[3:0]`) and modes (`C[2:0]`) in the barrel shifter of Table 6-3 that further reduces the total number of product terms used by the design.

6.4 Make changes to the dual-priority encoder program of Table 6-6 to further reduce the number of product terms required. State whether your changes increase the delay of the circuit when realized in a GAL22V10. Can you reduce the product terms enough to fit the design into a GAL16V8?

6.5 Here's an exercise where you can use your brain, like the author had to when figuring out the equation for the SUM0 output in Table 6-12. Do each of the SUM1–SUM3 outputs require more terms or fewer terms than SUM0?

6.6 Complete the design of the ABEL and PLD-based ones-counting circuit that was started in Section 6.2.6. Use 22V10 or smaller PLDs and try to minimize the total number of PLDs required. State the total delay of your design in terms of the worst-case number of PLD delays in a signal path from input to output.

6.7 Find another code for the Tic-Tac-Toe moves in Table 6-13 that has the same rotation properties as the original code. That is, it should be possible to compensate for a 180° rotation of the grid using just inverters and wire rearrangement. Determine whether the TWOINHAF equations will still fit in a single 22V10 using the new code.

6.8 Using a simulator, demonstrate a sequence of moves in which the PICK2 PLD in Table 6-16 will lose a Tic-Tac-Toe game, even if X goes first.

6.9 Modify the program in Table 6-16 to give the program a better chance of winning, or at least not losing. Can your new program still lose?

6.10 Modify both the "other logic" in Figure 6-13 and the program in Table 6-16 to give the program a better chance of winning, or at least not losing. Can your new program still lose?

6.11 Write the VHDL functions for `Vror`, `Vsll`, `Vsrl`, `Vsla`, and `Vsra` in Table 6-17 using the `ror`, `sll`, `srl`, `sla`, and `sra` operations as defined in Table 6-3.

6.12 The iterative-circuit version of `fixup` in Table 6-20 has a worst-case delay path of 15 OR gates from the first decoded value of `i` (14) to the `FSEL(0)` signal. Figure out a trick that cuts this delay path almost in half with no cost (or negative cost) in gates. How can this trick be extended further to save gates or gate inputs?

6.13 Rewrite the `barrel16` entity definition in Table 6-17 and the architecture in Table 6-22 so that a single direction-control bit is made explicitly available to the architecture.

Copyright © 1999 by John F. Wakerly Copying Prohibited

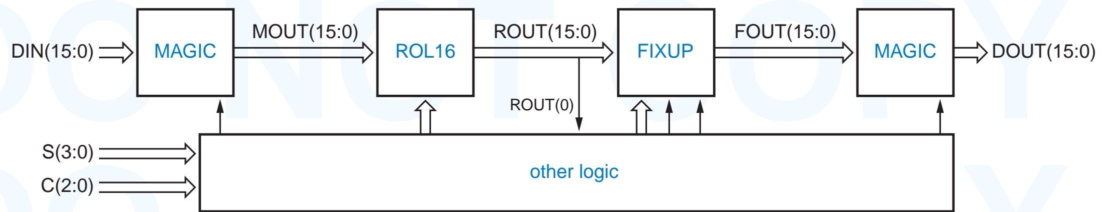

Figure X6.14

6.14 Rewrite the `barrel16` architecture definition in Table 6-22 to use the structure shown in Figure X6.14. Use the existing ROL16 and FIXUP entities; it's up to you to come up with MAGIC and the other logic.

6.15 Write a semibehavioral or structural version of the `fpencr_arch` architecture of Table 6-25 that generates only one adder in synthesis and that does not generate multiple 10-bit comparators for the nested "`if`" statement.

6.16 Repeat Exercise 6.15, including a structural definition of an efficient rounding circuit that performs the `round` function. Your circuit should require significantly fewer gates than a 4-bit adder.

6.17 Redesign the VHDL dual-priority encoder of Section 6.3.3 to get better, known performance, as suggested in the last paragraph of the section.

6.18 Write a structural VHDL architecture for a 64-bit comparator that is similar to Table 6-30 except that it builds up the comparison result serially from least to most significant stage.

6.19 What significant change occurs in the synthesis of the VHDL program in Table 6-31 if we change the statements in the "`when others`" case to "`null`"?

6.20 Write behavioral VHDL programs for the "ADDERx" components used in Table 6-34.

6.21 Write a structural VHDL programs for the "ADDERx" components in Table 6-34. Use a generic definition so that the same entity can be instantiated for ADDER2, ADDER3, and ADDER5, and show what changes must be made in Table 6-34 to do this.

6.22 Write a structural VHDL program for the "INCR5" component in Table 6-34.

6.23 Using an available VHDL synthesis tool, synthesize the Tic-Tac-Toe design of Section 6.3.7, fit it into an available FPGA, and determine how many internal resources it uses. Then try to reduce the resource requirements by specifying a different encoding of the moves in the `TTTdefs` package.

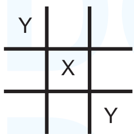

Figure X6.24

6.24 The Tic-Tac-Toe program in Section 6.3.7 eventually loses against an intelligent opponent if applied to the grid state shown in Figure X6.24. Use an available VHDL simulator to prove that this is true. Then modify the `PICK` entity to win in this and similar situations and verify your design using the simulator.

Copyright © 1999 by John F. Wakerly Copying Prohibited