Drill Problems

10.1 Determine the ROM size needed to realize the combinational logic function in each of the following figures: 4-39(b), 5-39, 5-77, 6-1, and 6-6.

10.2 Determine the ROM size needed to realize the combinational logic function performed by each of the following MSI parts: 74x49, 74x139, 74x153, 74x257, 74x381, 74x682.

10.3 Draw a logic symbol for and determine the size of a ROM that realizes a combinational logic function that can perform the function of either a 74x381 or a 74x382 depending on the value of a single MODE input.

10.4 Draw a logic symbol for and determine the size of a ROM that realizes an 8×8 combinational multiplier.

10.5 Show how to design a 2M×8 SRAM using HM628512 SRAMs and a combinational MSI part as building blocks.

Exercises

10.6 Our discussion of ROM sneak paths in connection with Figure 10-6 claimed that with A2–A0 = 101, bit lines D2_L and D0_L are pulled LOW through the direct connections. That's not really correct, unless the 74x138 is replaced with a decoder with open-collector outputs. Explain.

10.7 Describe the logic function of seven variables that is performed by the 128×1 ROM in Figure 10-7. Starting with the ROM pattern, one way to describe the logic function is to write the corresponding truth table and canonical sum. However, since the canonical sum has 64 7-variable product terms, you might want to look for a simple but precise word description of the function.

10.8 For the two-output logic function of Figure 4-39(b), compare the number of diodes or transistors needed in the AND-OR, AND, or storage array for PLA, PAL, and ROM realizations.

10.9 Show how to double the number of different attenuation amounts that can be selected in the digital attenuator of Figure 10-17 without increasing the size of the ROM.

10.10 Write the C functions `UlawToLinear` and `LinearToUlaw` that are required in Table 10-6. In `LinearToUlaw`, you should select the μ-law byte whose theoretical value is closest to the linear input. (*Hint:* The most efficient approach builds, at program initialization, a 256-entry table that is used by both functions.) Both functions should report out-of-range inputs.

10.11 Modify the C program in Table 10-6 to perform clipping in cases where the designer has specified a particular attenuation amount to be a gain, and multiplying an input value by the attenuation factor produces an out-of-range result.

10.12 Write a C program to generate the contents of a 256×8 ROM that converts from 8-bit binary to 8-bit Gray code. (*Hint:* Your program should embody the second Gray-code construction method in Section 2.11.)

10.13 Write a C program to generate the contents of a 256×8 ROM that converts from 8-bit Gray code to 8-bit binary. (*Hints:* You may use the results of Exercise 10.12. It doesn't matter if your program is slow.)

10.14 A certain communication system has been designed to transmit ASCII characters serially on a medium that requires an average signal level of zero, so a 5-out-of-10 code is used to code the data. Each 7-bit ASCII input character is transmitted as a 10-bit word with five 0s and five 1s. Write a C program to generate the contents of a 128×10 ROM that converts ASCII characters into coded words.

10.15 The receiving end of the system in Exercise 10.14 must convert each 10-bit coded word back into a 7-bit ASCII character. Write a C program to generate the contents of a $1K \times 8$ ROM that converts coded words onto ASCII characters. The extra output bit should be an "error" flag that indicates when a noncode word has been received.

10.16 How many ROM bits would be required to build a 16-bit adder/subtracter with mode control, carry input, carry output, and two's-complement overflow output? Be more specific than "billions and billions," and explain your answer.

10.17 Repeat Exercise 10.16 assuming that you may use *two* ROMs, so that the delay for a 16-bit addition or subtraction is twice the delay through one ROM. Assume that the two ROMs must be identical in both size and programming. Try to minimize the total number of ROM bits required, and sketch the resulting adder/subtracter circuit. Can the number of ROM bits be further reduced if the two ROMs are allowed to be different?

10.18 Show how, using additional SSI/MSI parts, a 2764 can be used as a $64K \times 1$ ROM. What is the access time of the $64K \times 1$ ROM?

10.19 Show how, using additional SSI/MSI parts, a 2764 can be used as a $2K \times 32$ ROM. You may assume the existence of a free-running clock signal whose period is slightly longer than the access time of the 2764. What is the access time of the $2K \times 32$ ROM?

10.20 Show how to build the μ-law adder circuit of Figure 10-18 with a $32K \times 8$ ROM and two XOR gates. Also, write a C program to generate the ROM contents.

10.21 Determine the ROM size needed to build the fixed-point to floating-point encoder of Figure 6-3. Draw a logic diagram using a commercially available ROM.

10.22 Write a C program to generate the ROM contents for Exercise 10.21. Unlike the original MSI solution, your C program should perform rounding; that is, for each fixed-point number it should generate the nearest possible floating-point number.

10.23 Draw a complete logic diagram for a ROM-based circuit that performs combinational multiplication of a pair of 8-bit unsigned or two's-complement integers. Signed versus unsigned operation should selected by a single input, SIGNED. You may use any of the commercial ROMs in Figure 10-11, as well as discrete gates.

10.24 Write and test a C program that generates the contents of the ROM(s) in Exercise 10.23.

10.25 Write a C program that generates a $256K \times 4$ ROM that computes the next move in a Tic-Tac-Toe game, using the input and output encodings of Section 6.2.7. Your program must be smart enough to pick a winning move whenever possible.

10.26 Repeat Exercise 10.25 using a 32K × 4 ROM. To accomplish this, the board state must be encoded in only 15 bits. Explain your coding algorithm, and write C functions to translate a cell number in either direction between your encoding and the encoding of Section 6.2.7.

10.27 For each of the timing parameters defined in Section 10.3.3, determine whether its value for the 128K × 8 SRAM you designed in Drill 10.5 is the same as the HM628512's. If it is different, indicate the new value. Use the worst-case values in the 74FCT column of Table 5-3 to determine the delays for the MSI part.

10.28 Using an HM6264 8K × 8 SRAM, a handful of MSI parts, and a PLD as building blocks, design an 8K × 8 late-write SSRAM with flow-through outputs.

10.29 Using an HM6264 8K × 8 SRAM, a handful of MSI parts, and a PLD as building blocks, design an 8K × 8 ZBT SSRAM with pipelined outputs.

10.30 Define the relevant timing parameters for the synchronous SRAM as defined in Exercise 10.28.

10.31 Calculate the values of the timing parameters you defined in Exercise 10.30 for your solution to Exercise 10.28.

10.32 In the style of Figure 10-27, draw the timing diagram for a late-write SSRAM with flow-through outputs for a series of interleaved reads and writes in the pattern R-R-W-W-R-W-R-W. Run the individual cycles as close together as possible, but be sure to account for resource conflicts that prevent back-to-back cycles. What is the average utilization of the SRAM array if the SSRAM is presented with a continuous streams of R-W-R-W-R-W requests?

10.33 Repeat the preceding exercise for a late-write SSRAM with pipelined outputs.

10.34 Using one of the theorems in Section 4.1, prove that the Xilinx XC4000-series CLB can realize any logic function of five variables.

10.35 Show how to use the function generators in an XC4000-series CLB to realize a 9-bit even parity function.

10.36 Show how to use the function generators in an XC4000-series CLB to realize an equality checker for two 4-bit operands. You should be able to cascade your design to check 4*n*-bit operands with just *n* CLBs.