For More Practice

Truth Tables

B.3 [10] <§B.2> Show that there are 2^n entries in a truth table for a function with n inputs.

B.4 [10] <§B.2> One logic function that is used for a variety of purposes (including within adders and to compute parity) is *exclusive OR*. The output of a two-input exclusive OR function is true only if exactly one of the inputs is true. Show the truth table for a two-input exclusive OR function and implement this function using AND gates, OR gates, and inverters.

Building Logic Gates

B.5 [15] <§B.2> Prove that the NOR gate is universal by showing how to build the AND, OR, and NOT functions using a two-input NOR gate.

B.6 [15] <§B.2> Prove that the NAND gate is universal by showing how to build the AND, OR, and NOT functions using a two-input NAND gate.

Multiplexors

B.17 [5] <§§B.2, B.3> Show a truth table for a multiplexor (inputs *A, B,* and *S;* output *C*), using don't cares to simplify the table where possible.

Flip-Flops and Latches

B.35 [5] <§B.8> Quite often, you would expect that given a timing diagram containing a description of changes that take place on a data input *D* and a clock input *C* (as in Figures B.8.3 and B.8.6 on pages B-52 and B-53, respectively), there would be differences between the output waveforms (*Q*) for a D latch and a D flip-flop. In a sentence or two, describe the circumstances (e.g., the nature of the inputs) for which there would not be any difference between the two output waveforms.

B.36 [5] <§B.8> Figure B.8.8 on page B-55 illustrates the implementation of the register file for the MIPS datapath. Pretend that a new register file is to be built, but that there are only two registers and only one read port, and that each register has only 2 bits of data. Redraw Figure B.8.8 so that every wire in your diagram corresponds to only 1 bit of data (unlike the diagram in Figure B.8.8, in which some wires are 5 bits and some wires are 32 bits). Redraw the registers using D flip-flops. You do not need to show how to implement a D flip-flop or a multiplexor.

Finite State Machines

B.37 [10] <§B.10> A friend would like you to build an "electronic eye" for use as a fake security device. The device consists of three lights lined up in a row, controlled by the outputs Left, Middle, and Right, which, if asserted, indicate that a light should be on. Only one light is on at a time, and the light "moves" from left

to right and then from right to left, thus scaring away thieves who believe that the device is monitoring their activity. Draw the graphical representation for the finite state machine used to specify the electronic eye. Note that the rate of the eye's movement will be controlled by the clock speed (which should not be too great) and that there are essentially no inputs.

B.38 [10] <§B.10> {Ex. B.37} Assign state numbers to the states of the finite state machine you constructed for Exercise B.37 and write a set of logic equations for each of the outputs, including the next state bits.

Constructing a Counter

B.39 [15] <§§B.2, B.8, B.10> Construct a 3-bit counter using three D flip-flops and a selection of gates. The inputs should consist of a signal that resets the counter to 0, called *reset*, and a signal to increment the counter, called *inc*. The outputs should be the value of the counter. When the counter has value 7 and is incremented, it should wrap around and become 0.

B.40 [20] <§B.10> A *Gray code* is a sequence of binary numbers with the property that no more than 1 bit changes in going from one element of the sequence to another. For example, here is a 3-bit binary Gray code: 000, 001, 011, 010, 110, 111, 101, and 100. Using three D flip-flops and a PLA, construct a 3-bit Gray code counter that has two inputs: *reset*, which sets the counter to 000, and *inc*, which makes the counter go to the next value in the sequence. Note that the code is cyclic, so that the value after 100 in the sequence is 000.

Timing Methodologies

B.41 [25] <§B.10> We wish to add a yellow light to our traffic light example on page B-68. We will do this by changing the clock to run at 0.25 Hz (a 4-second clock cycle time), which is the duration of a yellow light. To prevent the green and red lights from cycling too fast, we add a 30-second timer. The timer has a single input, called *TimerReset*, which restarts the timer, and a single output, called *TimerSignal,* which indicates that the 30-second period has expired. Also, we must redefine the traffic signals to include yellow. We do this by defining two output signals for each light: green and yellow. If the output NS green is asserted, the green light is on; if the output NSyellow is asserted, the yellow light is on. If both signals are off, the red light is on. Do *not* assert both the green and yellow signals at the same time, since American drivers will certainly be confused, even if European drivers understand what this means! Draw the graphical representation for the finite state machine for this improved controller. Choose names for the states that are *different* from the names of the outputs.

B.42 [15] <§B.10> Write down the next-state and output-function tables for the traffic light controller described in Exercise B.41.

B.43 [15] <§§B.2, B.10> Assign state numbers to the states in the traffic light example of Exercise B.41 and use the tables of Exercise B.42 to write a set of logic equations for each of the outputs, including the next-state outputs.

B.44 [15] <§§B.3, B.10> Implement the logic equations of Exercise B.43 as a PLA.