adders, comparators, multipliers, subtractors, arithmetic-logic units, shifters, counters, timers, and register files. For each component, the chapter examined two aspects: the internal design of the component, and the use of the component as part of a datapath to implement a desired system.

The chapter ended (Section 4.13) by describing some basic principles underlying the operation of an ultrasound machine, and showing how several of the datapath components might be used to implement parts of such a machine. One thing you might notice is how designing a real ultrasound machine would require some knowledge of the domain of ultrasound. The requirement that a software programmer or digital designer have some understanding of an application domain is quite common.

In the coming chapter, you will apply your knowledge of combinational logic design, sequential logic design (controller design), and datapath components, to build digital circuits that can implement general and powerful computations.

▶ 4.15 EXERCISES

An asterisk (*) indicates an especially challenging problem.

For exercises relating to datapath components, each problem may indicate whether the problem emphasizes the component's internal design or the component's use.

SECTION 4.2: REGISTERS

4.1 Trace the behavior of an 8-bit parallel-load register with 8-bit input *I*, 8-bit output *Q*, and load control input `ld` by completing the timing diagram in Figure 4.95.

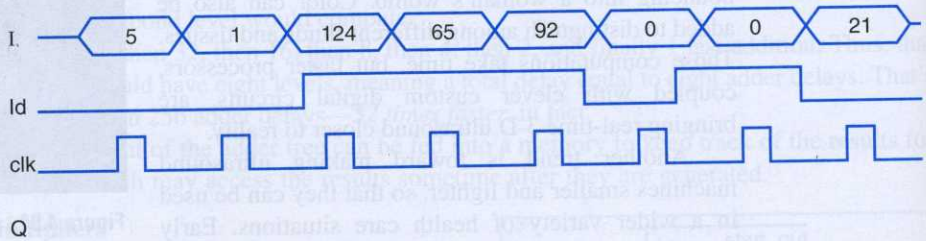

Figure 4.95 Timing diagram.

4.2 Trace the behavior of an 8-bit parallel-load register with 8-bit input *I*, 8-bit output *Q*, load control input `ld`, and synchronous clear input `clr` by completing the timing diagram in Figure 4.96.

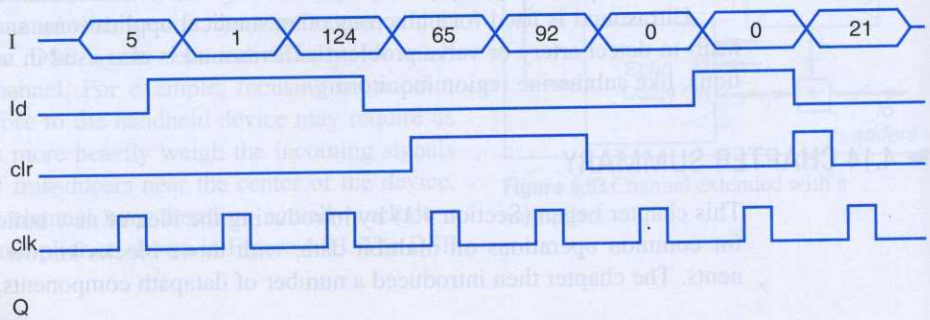

Figure 4.96 Timing diagram.

4.3 Design a 4-bit register with 2 control inputs `s1` and `s0`; 4 data inputs `I3`, `I2`, `I1`, and `I0`; and 4 data outputs `Q3`, `Q2`, `Q1`, and `Q0`. When `s1s0=00`, the register maintains its value. When `s1s0=01`, the register loads `I3...I0`. When `s1s0=10`, the register clears itself to `0000`. When `s1s0=11`, the register complements itself, so for example, `0000` would become `1111`, and `1010` would become `0101`. *(Component design problem.)*

4.4 Repeat the previous problem, but when `s1s0=11`, the register reverses its bits, so `1110` would become `0111`, and `1010` would become `0101`. *(Component design problem.)*

4.5 Design an 8-bit register with 2 control inputs `s1` and `s0`, 8 data inputs `I7...I0`, and 8 data outputs `Q7...Q0`. `s1s0=00` means maintain the present value, `s1s0=01` means load, and `s1s0=10` means clear. `s1s0=11` means to swap the high nibble with the low nibble (a nibble is 4 bits), so `11110000` would become `00001111`, and `11000101` would become `01011100`. *(Component design problem.)*

4.6 The radar gun used by a police officer outputs a radar signal and measures the speed of cars as they pass. However, when an officer wants to ticket an individual for speeding, he must save the measured speed of the car on the radar unit. Build a system to implement a speed save feature for the radar gun. The system has an 8-bit speed input S, an input `B` from the save button on the radar gun, and an 8-bit output D that will be sent to the radar's gun speed display. *(Component use problem.)*

4.7 Design a system with an 8-bit input I that can be stored in 8-bit registers A, B, and/or C when input `La`, `Lb`, and/or `Lc` is `1`, respectively. So if inputs `La` and `Lb` are `1`, then registers A and B will be loaded with input I, but register C will keep its current value. Furthermore, if input `R` is `1`, then the register values swap such that A=B, B=C, and C=A. Input `R` has priority over the `L` inputs. The system has one clock input also. *(Component use problem.)*

SECTION 4.3: ADDERS

4.8 Trace the values appearing at the outputs of a 3-bit carry-ripple adder for every one full-adder-delay time period when adding `111` with `011`. Assume all inputs were previously `0` for a long time.

4.9 Assuming all gates have a delay of 1 ns, compute the longest time required to add two numbers using an 8-bit carry-ripple adder.

4.10 Assuming AND gates have a delay of 2 ns, OR gates have a delay of 1 ns, and XOR gates have a delay of 3 ns, compute the longest time required to add two numbers using an 8-bit carry-ripple adder.

4.11 Design a 10-bit carry-ripple adder using 4-bit carry-ripple adders. *(Component use problem.)*

4.12 Design a system that computes the sum of three 8-bit numbers, using 8-bit carry-ripple adders. *(Component use problem.)*

4.13 Design an adder that computes the sum of four 8-bit numbers, using 8-bit carry-ripple adders. *(Component use problem.)*

4.14 Design a digital thermometer system that can compensate for errors in the temperature sensing device's output T, which is an 8-bit input to the system. The compensation amount can be positive only and comes to the system as a 3-bit binary number `c`, `b`, and `a` (`a` is the least significant bit), which come from a 3-pin DIP switch. The system should output the compensated temperature on an 8-bit output U. *(Component use problem.)*

4.15 We can add three 8-bit numbers by chaining one 8-bit carry-ripple adder to the output of another 8-bit carry-ripple adder. Assuming every gate has a delay of 1 time-unit, compute the longest delay of this three 8-bit number adder. Hint: you may have to look carefully inside the carry-ripple adders, even inside the full-adders, to correctly compute the longest delay from any input to any output. *(Component use problem.)*

SECTION 4.4: COMPARATORS

4.16 Trace through the execution of the 4-bit magnitude comparator shown in Figure 4.43 when a=15 and b=12. Be sure to show how the comparisons propagate thought the individual comparators.

4.17 Design a system that determines if three 4-bit numbers are equal, by connecting 4-bit magnitude comparators together and using additional components if necessary. *(Component use problem.)*

4.18 Design a 4-bit carry-ripple-style magnitude comparator that has two outputs, a greater-than or equal-to output `gte`, and a less-than or equal-to output `lte`. Be sure to clearly show the equations used in developing the individual 1-bit comparators and how they are connected to form the 4-bit circuit. *(Component design problem.)*

4.19 Design a circuit that outputs `1` if the circuit's 8-bit input equals 99:

(a) using an equality comparator,

(b) using gates only.

Hint: In the case of (b), you need only 1 AND gate and some inverters. *(Component use problem.)*

4.20 Use magnitude comparators and logic to design a circuit that computes the minimum of three 8-bit numbers. *(Component use problem.)*

4.21 Use magnitude comparators and logic to design a circuit that computes the maximum of two 16-bit numbers. *(Component use problem.)*

4.22 Use magnitude comparators and logic to design a circuit that outputs `1` when an 8-bit input a is between 75 and 100, inclusive. *(Component use problem.)*

4.23 Design a human body temperature indicator system for a hospital bed. Your system takes an 8-bit input representing a person's body temperature, which can range from 0 to 255. If the measured temperature is 95 or less, set output `A` to `1`. If the temperature is 96 to 104, set output `B` to `1`. If the temperature is 105 or above, set output `C` to `1`. Use 8-bit magnitude comparators and additional logic as required. *(Component use problem.)*

4.24 You are working as a weight guesser in an amusement park. Your job is to try to guess the weight of an individual before they step on a scale. If your guess is not within ten pounds of the individual's actual weight (higher or lower), the individual wins a prize. So if you guess 85 and the actual weight is 95, the person does not win; if you'd guessed 84, the person wins. Build a weight guess analyzer system that outputs whether the guess was within ten pounds. The weight guess analyzer has an 8-bit guess input G, an 8-bit input from the scale W with the correct weight, and a bit output `C` that is `1` if the guessed weight was within the defined limits of the game. Use 8-bit magnitude comparators and additional logic or components as required. *(Component use problem.)*

SECTION 4.5: MULTIPLIER—ARRAY-STYLE

4.25 Assuming all gates have a delay of 1 time-unit, which of the following designs will compute the 8-bit multiplication A*9 faster:

(a) a circuit as designed in Exercise 4.45, or

(b) an 8-bit array style multiplier with one input connected to a constant value of nine.

4.26 Design an 8-bit array-style multiplier. *(Component design problem.)*

4.27 Design a circuit to compute $F=(A*B*C)+3*D+12$. A, B, C, and D are 16-bit inputs, and F is a 16-bit output. Use 16-bit multiplier and adder components, and ignore overflow issues.

SECTION 4.6: SUBTRACTORS AND SIGNED NUMBERS

4.28 Convert the following two's complement binary numbers to decimal numbers:
(a) 00001111
(b) 10000000
(c) 10000001
(d) 11111111
(e) 10010101

4.29 Convert the following two's complement binary numbers to decimal numbers:
(a) 01001101
(b) 00011010
(c) 11101001
(d) 10101010
(e) 11111100

4.30 Convert the following two's complement binary numbers to decimal numbers:
(a) 11100000
(b) 01111111
(c) 11110000
(d) 11000000
(e) 11100000

4.31 Convert the following 9-bit two's complement binary numbers to decimal numbers:
(a) 011111111
(b) 111111111
(c) 100000000
(d) 110000000
(e) 111111110

4.32 Convert the following decimal numbers to 8-bit two's complement binary form:
(a) 2
(b) –1
(c) –23
(d) –128
(e) 126
(f) 127
(g) 0

4.33 Convert the following decimal numbers to 8-bit two's complement binary form:
(a) 29
(b) 100
(c) 125
(d) –29
(e) –100
(f) –125
(g) –2

4.34 Convert the following decimal numbers to 8-bit two's complement binary form:
(a) 6
(b) 26
(c) –8
(d) –30
(e) –60
(f) –90

4.35 Convert the following decimal numbers to 9-bit two's complement binary form:
(a) 1
(b) –1
(c) –256
(d) –255
(e) 255
(f) –8
(g) –128

4.36 Repeat Exercise 4.14 except that the compensation amount can be positive or negative, coming to the system via four inputs `d`, `c`, `b`, and `a` from a 4-pin DIP switch (`d` is the most significant bit). The compensation amount is in two's complement form (so the person setting the DIP switch must know that). Design the circuit. What is the range by which the input temperature can be compensated? *(Component use problem.)*

4.37 Create the internal design of a full-subtractor. *(Component design problem.)*

4.38 Create an absolute value component *abs* with an 8-bit input A that is a signed binary number, and an 8-bit output Q that is unsigned and that is the absolute value of A. So if the input is `00001111` (+15) then the output is also `00001111` (+15), but if the input is `11111111` (–1) then the output is `00000001` (+1).

4.39 Using 4-bit subtractors, build a circuit that has three 8-bit inputs A, B, and C, and a single 8-bit output F, where $F=(A–B)–C$. *(Component use problem.)*

SECTION 4.7: ARITHMETIC-LOGIC UNITS—ALUS

4.40 Design an ALU with two 8-bit inputs A and B, and control inputs `x`, `y`, and `z`. The ALU should support the operations described in Table 4.3. Use an 8-bit adder and an arithmetic/logic extender. *(Component design problem.)*

TABLE 4.3 Desired ALU operations.

Inputs			Operation
x	y	z	
0	0	0	S = A – B
0	0	1	S = A + B
0	1	0	S = A * 8
0	1	1	S = A / 8
1	0	0	S = A NAND B (bitwise NAND)
1	0	1	S = A XOR B (bitwise XOR)
1	1	0	S = Reverse A (bit reversal)
1	1	1	S = NOT A (bitwise complement)

4.41 Design an ALU with two 8-bit inputs A and B, and control inputs `x`, `y`, and `z`. The ALU should support the operations described in Table 4.4. Use an 8-bit adder and an arithmetic/logic extender. *(Component design problem.)*

TABLE 4.4 Desired ALU operations.

Inputs			Operation
x	y	z	
0	0	0	S = A + B
0	0	1	S = A AND B (bitwise AND)
0	1	0	S = A NAND B (bitwise NAND)
0	1	1	S = A OR B (bitwise OR)
1	0	0	S = A NOR B (bitwise NOR)
1	0	1	S = A XOR B (bitwise XOR)
1	1	0	S = A XNOR B (bitwise XNOR)
1	1	1	S = NOT A (bitwise complement)

4.42 An instructor teaching Boolean algebra wants to help her students learn and understand basic Boolean operators by providing the students with a calculator capable of performing bitwise AND, NAND, OR, NOR, XOR, XNOR, and NOT operations. Using the ALU specified in Exercise 4.41, build a simple logic calculator using DIP switches for input and LEDs for output. The logic calculator should have three DIP switch inputs to select which logic operation to perform. *(Component use problem.)*

SECTION 4.8: SHIFTERS

4.43 Design an 8-bit shifter that shifts its inputs two bits to the right (shifting in `0`s) when the shifter's shift control input is `1`. *(Component design problem.)*

4.44 Design a circuit that outputs the average of four 8-bit unsigned binary inputs
(a) ignoring overflow issues,
(b) using wider internal components or wires to avoid losing information due to overflow.
(Component use problem.)

4.45 Design a circuit whose 16-bit output is nine times its 16-bit input D representing an unsigned binary number. Ignore overflow issues. *(Component use problem.)*

4.46 Design a special multiplier circuit that can multiply its 16-bit input by 2, 4, 8, 16, or 32, specified by three inputs `a`, `b`, `c` (`abc=000` means no multiply, `abc=001` means multiply by 2, `abc=010` means by 4, `abc=011` means by 8, `abc=100` means by 16, `abc=101` means by 32). *Hint:* A simple solution consists entirely of just one copy of a component from this chapter. *(Component use problem.)*

4.47 Use strength reduction to create a circuit that computes P=27*Q using only shifts and adds. P is a 12-bit output and Q is a 12-bit input. Estimate the transistors in the circuit and compare to the estimated transistors in a circuit using a multiplier.

4.48 Use strength reduction to create a circuit that approximately computes P=(1/3)*Q using only shifters and adders. Strive for accuracy to the hundredths place (0.33). P is a 12-bit output and Q is a 12-bit input. Use wider internal components and wires as necessary to prevent internal overflow.

4.49 Show the internal values of the barrel shifter of Figure 4.64, when I=`01100101`, `x=1`, `y=0`, and `z=1`. Be sure to show how the input I is shifted after each internal shifter stage. *(Component design problem.)*

4.50 Using the barrel shifter shown in Figure 4.64, what settings of the inputs `x`, `y`, and `z` are required to shift the input *I* left by six positions?

SECTION 4.9: COUNTERS AND TIMERS

4.51 Design a 4-bit up-counter that has two control inputs: `cnt` enables counting up, while `clear` synchronously resets the counter to all `0`s:
(a) using a parallel load register as a building block,
(b) using flip-flops and muxes by following the register design process of Section 4.2.
(Component design problem.)

4.52 Design a 4-bit down-counter that has three control inputs: `cnt` enables counting up, `clear` synchronously resets the counter to all `0`s, and `set` synchronously sets the counter to all `1`s:
(a) using a parallel load register as a building block,
(b) using flip-flops and muxes by following the register design process of Section 4.2.
(Component design problem.)

4.53 Design a 4-bit up-counter with an additional output `upper`. `upper` outputs a `1` whenever the counter is within the upper half of the counter's range, `8` to `15`. Use a basic 4-bit up-counter as a building block. *(Component design problem.)*

4.54 Design a 4-bit up/down-counter that has four control inputs: `cnt_up` enables counting up, `cnt_down` enables counting down, `clear` synchronously resets the counter to all `0`s, and `set` synchronously sets the counter to all `1`s. If two or more control inputs are `1`, the counter retains its current count value. Use a parallel-load register as a building block. *(Component design problem.)*

4.55 Design a circuit for a 4-bit decrementer. *(Component design problem.)*

4.56 Assume an electronic turnstile internally uses a 64-bit counter that counts up once for each person that passes through the turnstile. Knowing that California's Disneyland park attracts about 15,000 visitors per day, and assuming they all pass that one turnstile, how many days would pass before the counter would roll over? *(Component use problem.)*

4.57 Design a circuit that outputs a 1 every 99 clock cycles:
(a) Using an up-counter with a synchronous clear control input, and using extra logic,
(b) Using a down-counter with parallel load, and using extra logic.
(c) What are the tradeoffs between the two designs from parts (a) and (b)?
(Component use problem.)

4.58 Give the count range for the following sized up-counters:
(a) 8-bits, 12-bits, 16-bits, 20-bits, 32-bits, 40-bits, 64-bits, and 128-bits.
(b) For each size of counter in part (a), assuming a 1 Hz clock, indicate how much time would pass before the counter wraps around; use the most appropriate units for each answer (seconds, minutes, hours, days, weeks, months, or years).
(Component use problem.)

4.59 Create a clock divider that converts a 14 MHz clock into a 1 MHz clock. Use a down-counter with parallel load. Clearly indicate the width of the down-counter and the counter's load value. *(Component use problem.)*

4.60 Assuming a 32-bit microsecond timer is available to a controller, and a controller clock frequency of 100 MHz, create a controller FSM that blinks an LED by setting an output `L` to `1` for 5 ms and then to `0` for 13 ms, and then repeats. Use the timer to achieve the desired timing (i.e., do not use a clock divider). For this example, the blinking rate can vary by a few clock cycles. *(Component use problem.)*

SECTION 4.10: REGISTER FILES

4.61 Design an 8x32 two-port (1 read, 1 write) register file. *(Component design problem.)*

4.62 Design a 4x4 three-port (2 read, 1 write) register file. *(Component design problem.)*

4.63 Design a 10x14 register file (one read port, one write port). *(Component design problem.)*

4.64 A 4x4 register file's four registers initially each contain `0101`.

(a) Show the input values necessary to read register 3 and to simultaneously write register 3 with the value `1110`.

(b) With these values, show the register file's register values and output values before the next rising clock edge, and after the next rising clock edge.

▶ DESIGNER PROFILE

Roman began studying Computer Science in college due to his interest in software development. During his undergraduate studies, his interests expanded to also include the fields of digital design and embedded systems, which eventually led him to become involved in research developing new methods to help designers quickly build large integrated circuits (ICs). Roman continued his education through graduate studies and received his MS in Computer Science, after which Roman worked for both a large company designing ICs for consumer electronics, as well as a start-up company focusing on high-performance processing.

Roman enjoys working as both a software developer and hardware engineer and believes that "fundamentally software and hardware design are very similar, both relying on efficiently solving difficult problems. While good problem solving skills are important, good learning skills are also important." Contrary to what many students may believe, he points out that "learning is a fundamental activity and skill that does not end when you receive your degree. In order to solve problems, you often are required to learn new skills, adopt new programming languages and tools, and determine if existing solutions will help you solve the problems you face as an engineer." Roman points out that digital design has changed at a rapid pace over the last few decades, requiring engineers to learn new design techniques, learn new programming languages, such as VHDL or SystemC, and be able to adopt new technologies to stay successful. "As the industry continues to advance at such a rapid pace, companies do not only hire engineers for what they already know, but more so on how well those engineers can continue to expand their knowledge and learn new skills." He points out that "college provides students with an excellent opportunity to not only learn the essential information and skills from their course work but also to learn additional information on their own, possibly by learning different programming languages, getting involved in research, or working on larger design projects."

Roman is motivated by his enjoyment of the work he does as well being able to work with other engineers who share his interests. "Motivation is one of the keys to success in an engineering career. While motivation can come from many different sources, finding a career that you are truly interested in and enjoy really helps. Co-workers are also a great source of motivation as well as knowledge and technical advice. Working as a member of a team that communicates well is very rewarding. You are able to motivate each other and use your strengths along with the strengths of your co-workers to achieve goals far beyond that which you could achieve on your own."