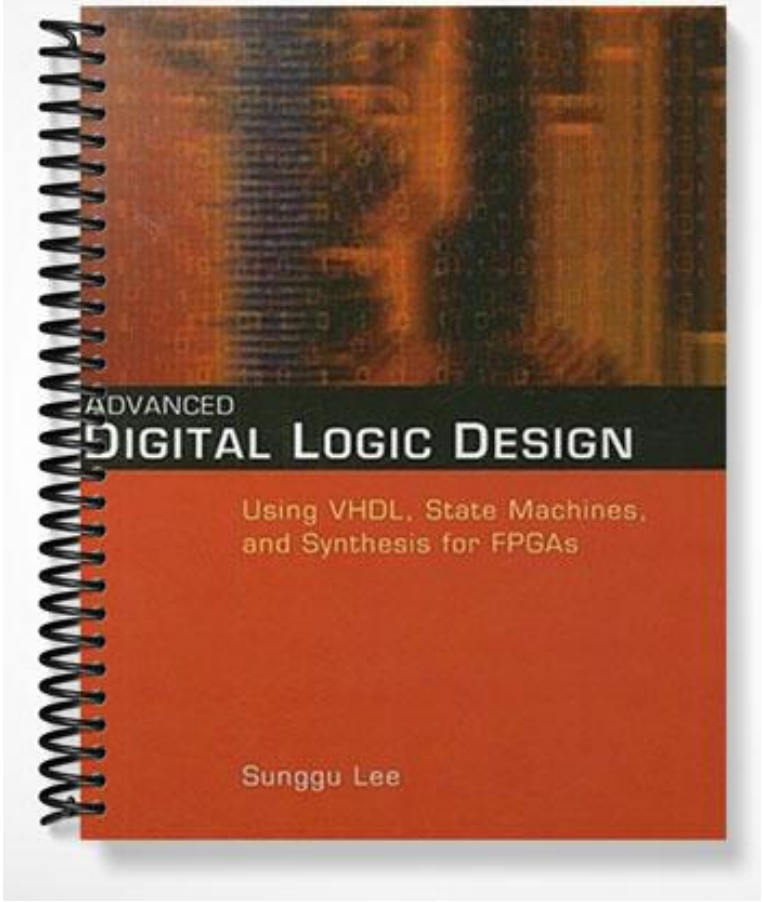
SOLUTIONS MANUAL



Chapter 2

DIGITAL LOGIC DESIGN USING HARDWARE DESCRIPTION LANGUAGES

P2.1 The pros and cons of schematic- versus HDL-based design entry are listed in Section 2.1.

P2.2 The various types of synthesis possible are described in detail in Section 2.3. Logic synthesis (synthesis from logic schematics or logic diagram descriptions) and register-transfer-level synthesis are supported by most commercial synthesis tools. Behavioral synthesis, especially when the level of description is at a very high level, is the most difficult type of synthesis.

P2.3 The following is a possible sequence of RTL statements for an 8:1 MUX.

```
sel0 := (not S2) & (not S1) & (not S0);
sel1 := (not S2) & (not S1) & S0;
...
sel7 := S2 & S1 & S0;
out := (sel0 & I0) | (sel1 & I1) | (sel2 & I2) | (sel3 & I3) |
       (sel4 & I4) | (sel5 & I5) | (sel6 & I6) | (sel7 & I7);
```

The following is a sequence of RTL statements for a 3:8 decoder.

```
O0 := (not I2) & (not I1) & (not I0);
O1 := (not I2) & (not I1) & I0;
...
O7 := I2 & I1 & I0;
```

The following is a sequence of RTL statements for an 8-bit even parity checker.

```
f := not ( I7 ^ I6 ^ I5 ^ I4 ^ I3 ^ I2 ^ I1 ^ I0 );
```

P2.4 Figure 2.1 shows a 2-input exclusive-OR gate designed using four 2-input NAND gates. Labels have been added to external and internal signals in order to aid in the description of the operation of this circuit. It is given that $(a, b) = (0, 0)$ at time $t = 0$, $(a, b) = (0, 1)$ at $t = 4$, $(a, b) = (1, 1)$ at $t = 6$,

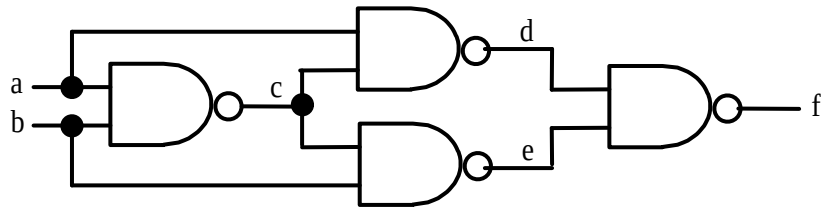


Figure 2.1. A 2-input exclusive-OR gate designed using 2-input NAND gates.

and $(a, b) = (1, 0)$ at $t = 8$.

Assuming two time-unit delays for each NAND gate, $(a, b) = (0, 0)$ at time $t = 0$ causes the signal c to change from unknown (`1'bX`) to logic 1 (`1'b1`) at time $t = 2$. Signal d changes to `1'b1` from `1'bX` at $t = 2$ since $a = 0$ at $t = 0$. Likewise, signal e changes to `1'b1` from `1'bX` at $t = 2$ since $b = 0$ at $t = 0$. Then, f changes to `1'b0` from `1'bX` at $t = 4$ since $(d, e) = (1, 1)$ at $t = 2$. Note that this is the correct result since $0 \oplus 0 = 0$. The change of signal c to `1'b1` at time $t = 2$ does not result in any additional changes. Next, $(a, b) = (0, 1)$ at $t = 4$ results in the following signal changes: e changes to `1'b0` at $t = 6$ since its two inputs are $(1, 1)$, f changes to `1'b1` at $t = 8$ due to the change in e. Next, $(a, b) = (1, 1)$ at $t = 6$ results in the following signal changes: c changes to `1'b0` at $t = 8$ since its two inputs are $(1, 1)$, d changes to `1'b0` at $t = 8$ since $(a, c) = (1, 1)$ at $t = 6$, (note that c is still `1'b1` at $t = 6$), d changes to `1'b1` at $t = 10$ since $(a, c) = (1, 0)$ at $t = 8$, e changes to `1'b1` at $t = 10$ since $(c, b) = (0, 1)$ at $t = 8$, f changes to `1'b0` at $t = 12$ since $(d, e) = (1, 1)$ at $t = 10$. Finally, $(a, b) = (1, 0)$ at $t = 8$ results in the following signal changes: c changes to `1'b1` at $t = 10$ since $(a, b) = (1, 0)$ at $t = 8$, d changes to `1'b0` at $t = 12$ since $(a, c) = (1, 1)$ at $t = 10$, f changes to `1'b1` at $t = 14$ since $(d, e) = (0, 1)$ at $t = 12$.

P2.5 Let us use the following NAND-gate circuit. NAND-gate $n1$ has inputs a and b and output c. NAND-gate $n2$ has inputs a and c and output d. NAND-gate $n3$ has inputs c and b and output e. Finally, NAND-gate $n3$ has inputs d and e and output f. f is assumed to be output of the exclusive-OR circuit that we are implementing. Then, given delays of 2 ns for each NAND-gate, the resulting timing diagram is shown in Figure 2.2. The f signal waveform is a 6 ns delayed form of the exclusive OR of a and b, but the 1-pulse starts after only a 4 ns delay. This is due to the fact that there are signal paths with both 4 ns and 6 ns delays to the output f from the two inputs a and b.

P2.6 In a two-dimensional graph, a *vector* is an arrow pointing from the origin to a point in the two-dimensional graph. Such a vector can also be represented as a 2-tuple (a list of two values). In a similar manner, a *test vector* is an n-tuple, where the n values in the n-tuple correspond to the values of n primary inputs to a circuit. Thus, a sequence of test vectors constitutes a test bench for a circuit. The test vectors used in Problem P2.4 are (0,0), (0,1), (1,1), and (1,0). The first and second values in these 2-tuples correspond to the values of the a and b primary inputs, respectively. The times when these test vectors are active are time units $t = 0$, $t = 4$, $t = 6$, and $t = 8$. If these time unit values are included in the first position of the tuples used to represent the test vectors, then the test vectors for this circuit can be written as (0,0,0), (4,0,1), (6,1,1), and (8,1,0).

P2.7 Although two path from the inputs to the output f go through both the NAND-gate and the OR-gate, there is also a more direct path from the signal

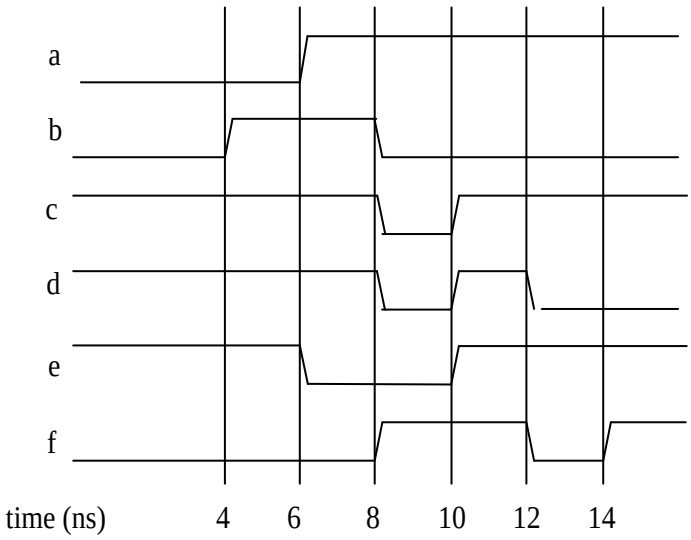


Figure 2.2. The timing diagram for Problem P2.5.

b to the output f that goes through only the OR-gate. Since b is initially logic 1, the OR of any value with this b value will be a logic 1. Thus, since the OR-gate has a 2 ns delay, the f value will be a logic 1 after a 2 ns delay.

P2.8 Since c (the speed of light) is approximately 3×10^8 m/s, the delay for a 10 cm length of PCB trace material is 0.1 m/d, where d is the speed of travel of an electrical signal in the PCB trace. Then, assuming $d = 0.3c$ to $0.7c$, the required delay is somewhere in between 1.11 ns and 0.48 ns. This amount of delay is quite significant. For example, with a 2.2 GHz Pentium IV personal computer, the clock period of the 2.2 GHz CPU clock is 0.45 ns. The delay through a 10 cm PCB trace is *longer* than the clock period of the CPU of a personal computer!

P2.9 Many different methods can be used to send a logic 1 followed by a logic 0 from one end of a wire to the other. Such methods include the following: (1) raising the voltage to +5 V and then dropping the voltage to 0 V, (2) injecting a current of 10 mA and then halting the injection of current, and (3) applying +100 mV followed by -100 mV on one wire and -100 mV followed by +100mV on another wire. These methods have the following pros and cons: (1) is simple to implement but may be a bit slow and require a significant amount of power, (2) may require less power than method (1) but requires a circuit capable of controlling the amount of current produced even with large variations in the type of load presented at the end of the signal wire, and (3) is fast, immune to noise on the wire, and low-power but requires two wires to send a single bit value. Other possible methods include voltage transitions, sinusoidal voltage waves with two different frequencies, sinusoidal voltage waves with two different amplitudes, monochromatic light passed through an optical fiber, etc.

P2.10 If voltage levels are used to represent logic values, then the transmission of a logic value from one point to another implies that a voltage level must be transmitted from one point to another. This in turn implies the movement of *electrons* and *holes* (which are the vacancies left by departing electrons). Since the movement of holes in one direction corresponds to the movement of electrons in the other direction, let us simply focus on the movement of electrons. Then, in order to transmit a logic 1 value from one point to another, electrons must be *pulled in* from the destination point in order to raise the voltage of the destination point to the voltage corresponding to a logic 1. Thus, when a logic 1 is transmitted from output pin of one chip to ten input pins spread out among ten different chips, the output pin has to provide a powerful suction force that can pull in electrons from all ten input pins. As the output pin is pulling in all of those electrons, the electrons are suctioned out through the Vdd power supply input. After sufficient numbers of electrons have been pulled in from all ten input pins, the voltage level at those ten input pins will rise to a value that is recognizable as logic 1. This movement of electrons can be viewed as analogous to the movement of water in pipes,

with the Vdd pin corresponding to the sink, the ground pin corresponding to the water source, and electrical wires corresponding to water pipes. A similar analogy can be made with the change of water temperature from one end of a bathtub to another when hot water is poured into one end of the tub. Clearly, as the number of input pins to be driven (to a specific logic value by the output pin) increases, the time required to transmit a logic value increases and it becomes more difficult to transmit such a logic value. As the number of input pins to be driven increases beyond a certain threshold, it will no longer become possible to transmit a logic value to those input pins from a specific output pin.

When the output pin value changes from a logic 1 to a logic 0, electrons must be *pushed out* from the output pin to all ten input pins. These electrons will come mostly from the ground pin of the output pin's chip. The signal transmission delay will become longer as the number of input pins to be driven increases.

P2.11 For this problem, the student should write a C or C++ program first. Then, once the program has been debugged and found to work satisfactorily, the algorithmic steps used in the program can be converted into RTL format.

P2.12 The following is an algorithm that can be used to compute and output the greatest common divisor of two n-bit numbers A and B.

1. $i \leftarrow max(A, B)$; // assume A, B positive
2. while $(i > 0)$ do begin
2a. if $((A \ mod \ i) = 0)$ then // use combinational logic for "mod"
if $((B \ mod \ i) = 0)$ then
$gcd \leftarrow i$; // gcd will be ≥ 1
go to Step 3;
2b. $i \leftarrow i - 1$;
end while;
3. output gcd as the greatest common divisor;

P2.13 Let us use TB to refer to the toothbrush and TP to refer to the toothpaste. Then, an RTL algorithm for a toothbrush/toothpaste vending machine circuit can be written as follows.

1. $MONEY \leftarrow 0$;
$drop\_TB \leftarrow 0$;
$drop\_TP \leftarrow 0$;
2. while (TRUE) do begin
2a. if $(select\_TB \ and \ (MONEY \geq 1))$ then
$MONEY \leftarrow MONEY - 1$;
$drop\_TB \leftarrow 1$;

```
2b.   else if (select_TP and (MONEY ≥ 2)) then
          MONEY ← MONEY − 2;
          drop_TP ← 1;
      end if;
2c.   wait for 1 clock cycle;
2d.   drop_TB ← 0;
      drop_TP ← 0;
2e.   if (dollar_detected) then
          MONEY ← MONEY + 1;
      end if;
  end while;
```