3e8.1 8.2 The new expression describes exactly the input combinations in which the 8 high-order bits of ABUS are 00000001_2, the same as the original expression using don't-cares.

3e8.3 8.3 There are $64 \times 32 = 2048$ fuses in the AND array (for example, see Figure 8–18). Each of the eight macrocells has one fuse to control the output polarity and one fuse to select registered vs. combinational configuration in the 16V8R, or to assert the output-enable in the 16V8S. There are also two global fuses to select the overall configuration (16V8C, 16V8R, or 16V8S). The total number of fuses is therefore $2048 + 16 + 2 = 2066$.

A real 16V8 (depending on the manufacturer) has at least 64 additional fuses to disable individual product terms, 64 user-programmable fuses that do nothing but store a user code, and a security fuse. (Once the security fuse is programmed, the rest of the fuse pattern can no longer be read.)

3e8.5 8.5 The f_{maxE} column below gives the answers in MHz.

Part numbers	Suffix	t_{PD}	t_{CO}	t_{CF}	t_{SU}	t_{H}	f_{maxE}	f_{maxI}
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	-5	5	4	–	4.5	0	117.7	117.7
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	-7	7.5	6.5	–	7	0	74.1	74.1
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	-10	10	8	–	10	0	55.6	55.6
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	B	15	12	–	15	0	37.0	37.0
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	B-2	25	15	–	25	0	25.0	25.0
PAL16L8, PAL16Rx, PAL20L8, PAL20Rx	A	25	15	–	25	0	25.0	25.0
PALCE16V8, PALCE20V8	-5	5	4	–	3	0	142.9	142.9
GAL16V8, GAL20V8	-7	7.5	5	3	5	0	100.0	125.0
GAL16V8, GAL20V8	-10	10	7.5	6	7.5	0	66.7	74.1
GAL16V8, GAL20V8	-15	15	10	8	12	0	45.5	50.0
GAL16V8, GAL20V8	-25	25	12	10	15	0	37.0	45.5
PALCE22V10	-5	5	4	–	3	0	142.9	142.9
PALCE22V10	-7	7.5	4.5	–	4.5	0	111.1	111.1
GAL22V10	-10	10	7	2.5	7	0	71.4	105.3
GAL22V10	-15	15	8	2.5	10	0	55.6	80.0
GAL22V10	-25	25	15	13	15	0	33.3	35.7

3e8.6 8.6 The f_{maxI} column above gives the answers in MHz.

3e8.7 8.7 See the ABEL program `Z74x374` below or in the accompanying `.zip` file (if posted by your instructor).

```
module Eight_Bit_Reg
title '8-bit Edge-Triggered Register'
Z74X374 device 'P16V8R';

" Input pins
CLK, !OE                          pin 1, 11;
D1, D2, D3, D4, D5, D6, D7, D8    pin 2, 3, 4, 5, 6, 7, 8, 9;

" Output pins
Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8    pin 19, 18, 17, 16, 15, 14, 13, 12 istype 'reg';

" Set definitions
D = [D1,D2,D3,D4,D5,D6,D7,D8];
Q = [Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8];

equations
Q.CLK = CLK;
Q := D;

end Eight_Bit_Reg
```

3e8.10 8.10 If EN or CLK is 0, the output will be stable. If both are 1, the results are unpredictable, since they depend on circuit timing. It is certain that the circuit's output will be unstable as long as this condition is true.

3e8.11 8.11 The counter is modified to return to a count of 0 when count 9 is reached. See the ABEL program `Z74x162` below or in the accompanying `.zip` file (if posted by your instructor)..

```
module Z74x162
title '4-bit Decade Counter'
"Z74X162 device 'P16V8R';
" Input pins
CLK, !OE                          pin 1, 11;
A, B, C, D                        pin 2, 3, 4, 5;
!LD, !CLR, ENP, ENT               pin 6, 7, 8, 9;
" Output pins
QA, QB, QC, QD                    pin 19, 18, 17, 16 istype 'reg';
RCO                               pin 15;
" Set definitions
INPUT = [ D,  C,  B,  A ];
COUNT = [QD, QC, QB, QA ];
equations
COUNT.CLK = CLK;
COUNT := !CLR & (  LD & INPUT
                # !LD &  (ENT & ENP) & (COUNT < 9) & (COUNT + 1)
                # !LD &  (ENT & ENP) & (COUNT == 9) & 0
                # !LD & !(ENT & ENP) &  COUNT);
RCO = (COUNT == 9) & ENT;
end Z74x162
```

3e8.13 8.13 The counting direction is controlled by QD: count up when QD=1, count down when QD=0. A load occurs when the counter is in the terminal state, 1111 when counting up, 0000 when counting down. The MSB is complemented during a load and the other bits are unchanged.

Let us assume that the counter is initially in one of the states 0000–0111. Then the counter counts down (QD=0). Upon reaching state 0000, it loads 1000 and subsequently counts up (QD=1). Upon reaching state 1111, the counter loads 0111, and subsequently counts down, repeating the cycle.

If the counter is initially in one of the states 1000–1111, the same cyclic behavior is observed. The counting sequence has a period of 16 and is, in decimal,

8, 9, 10, 11, 12, 13, 14, 15, 7, 6, 5, 4, 3, 2, 1, 0, 8, 9, ...

If only the three LSBs are observed, the sequence is

0, 1, 2, 3, 4, 5, 6, 7, 7, 6, 5, 4, 3, 2, 1, 0, 0, 1, ...

3e8.17 8.15 The path from the Q1 counter output (B decoder input) to the Y2_L output has 10 ns more delay than the Q2 and Q0 (C and A) paths. Let us examine the possible Y2_L glitches in Figure 8–43 with this in mind:

3→4 (011→100) Because of the delay in the Q1 path, this transition will actually look like 011→110→100. The Y6_L output will have a 10-ns glitch, but Y2_L will not.

7→0 (111→000) Because of the delay in the Q1 path, this transition will actually look like 111→010→000. The Y2_L output will have a 10-ns glitch, but the others will not.

3e8.20 8.19 The synchronizer fails if META has not settled by the beginning of the setup-time window for FF2, which is 5 ns before the clock edge. Since the clock period is 40 ns, the available metastability resolution time is 35 ns. The MTBF formula is

$$\text{MTBF}(t_r) = \frac{\exp(t_r/\tau)}{T_0 \cdot f \cdot a}$$

Substituting the proper values of τ and T_0 for the ’F74, and of f and a for the problem, we calculate

$$\text{MTBF}(35\text{ns}) = \frac{\exp(35/0.4)}{2.0 \cdot 10^{-4} \cdot 10^{6} \cdot 10^{6}} \approx 2 \cdot 10^{28}\text{s}$$

3e8.22 8.21 For TTL, refer to the sample data sheet on page 168 of the text: “Not more than one output should be shorted at a time; duration of short-circuit should not exceed one second.” In the switch debounce circuit, the short lasts only for a few tens of *nanoseconds*, so it’s OK. High-current-drive CMOS families, such as AC and ACT, recommend against even momentary shorting of the outputs.

3e8.23 8.22 CMOS outputs can “latch up” under certain conditions. According to the Motorola *High-Speed CMOS Logic Data* book (1988 edition, pp. 4–10), a 74HCT output can latch up if a voltage outside the range $-0.5 \le V_{out} \le V_{CC} + 0.5\text{V}$ is forced on the output by an external source. In a switch debounce circuit using 74HCT04s, the switch connection to ground is an external source, but the voltage (0 V) is within the acceptable range and should not be a problem.

Another potential problem is excessive short-circuit current, but again the data book indicates that shorting the output briefly is not a problem, as long as “the maximum package power dissipation is not violated” (i.e., the short is not maintained for a long time).

Similar considerations apply to 74AC and 74ACT devices, but in the balance, such devices are *not* recommended in the switch-debounce application, as we’ll explain. On one hand, typical 74AC/74ACT devices are even less susceptible to latch-up than 74HCT devices. (For example, see the Motorola *FACT Data* book, 1988 edition, pp. 2–9.) On the other hand, 74AC/74ACT’s high performance may create noise problems for *other* devices in a system. In particular, when the 74AC/74ACT HIGH output is shorted to ground, it may momentarily drag the local 5 V power-supply rail down with it, especially if the decoupling capacitors are small, far away, or missing. This will in turn cause incorrect operation of the other, nearby logic devices.

3e8.25 8.24 TTL inputs require significant current, especially in the LOW state. The bus holder cannot supply enough current unless the series resistor is made much smaller, which then creates a significant load on the bus.

3e8.26 8.25 Please see the VHDL program `latch_decode.vhd` below or in the accompanying `.zip` file (if posted by your instructor). This program was kindly written and contributed by Xilinx application engineering, but it has not been further checked for correctness and coding style.

```
library IEEE;
use IEEE.std_logic_1164.all;
--
-- Exercise 8-26
-- This code combines the address latch and
-- and the decoder and its latch
entity latch_decode is
port (
       abus : in std_logic_vector ( 31 downto 0);
       avalid : in std_logic;
       la : out std_logic_vector ( 19 downto 0);
       romcs, ramcs0, ramcs1, ramcs2 : out std_logic
);
end entity latch_decode;

architecture behave of latch_decode is


begin
process (avalid, abus)
begin
if (avalid = '1') then
       la <= abus (19 downto 0);
end if;
end process;

process (abus, avalid)
variable rom, ram1, ram2, ram0 : std_logic_vector (11 downto 0);
begin
rom  := "111111111111";
ram0 := "000000000000";
ram1 := "000000010000";
ram2 := "000000100000";

If (avalid= '1') then
    if (abus (31 downto 20) = rom ) then romcs  <= '1'; else romcs  <= '0'; end if;
    if (abus (31 downto 20) = ram0) then ramcs0 <= '1'; else ramcs0 <= '0'; end if;
    if (abus (31 downto 20) = ram1) then ramcs1 <= '1'; else ramcs1 <= '0'; end if;
    if (abus (31 downto 20) = ram2) then ramcs2 <= '1'; else ramcs2 <= '0'; end if;
end if;
end process;

end behave;
```

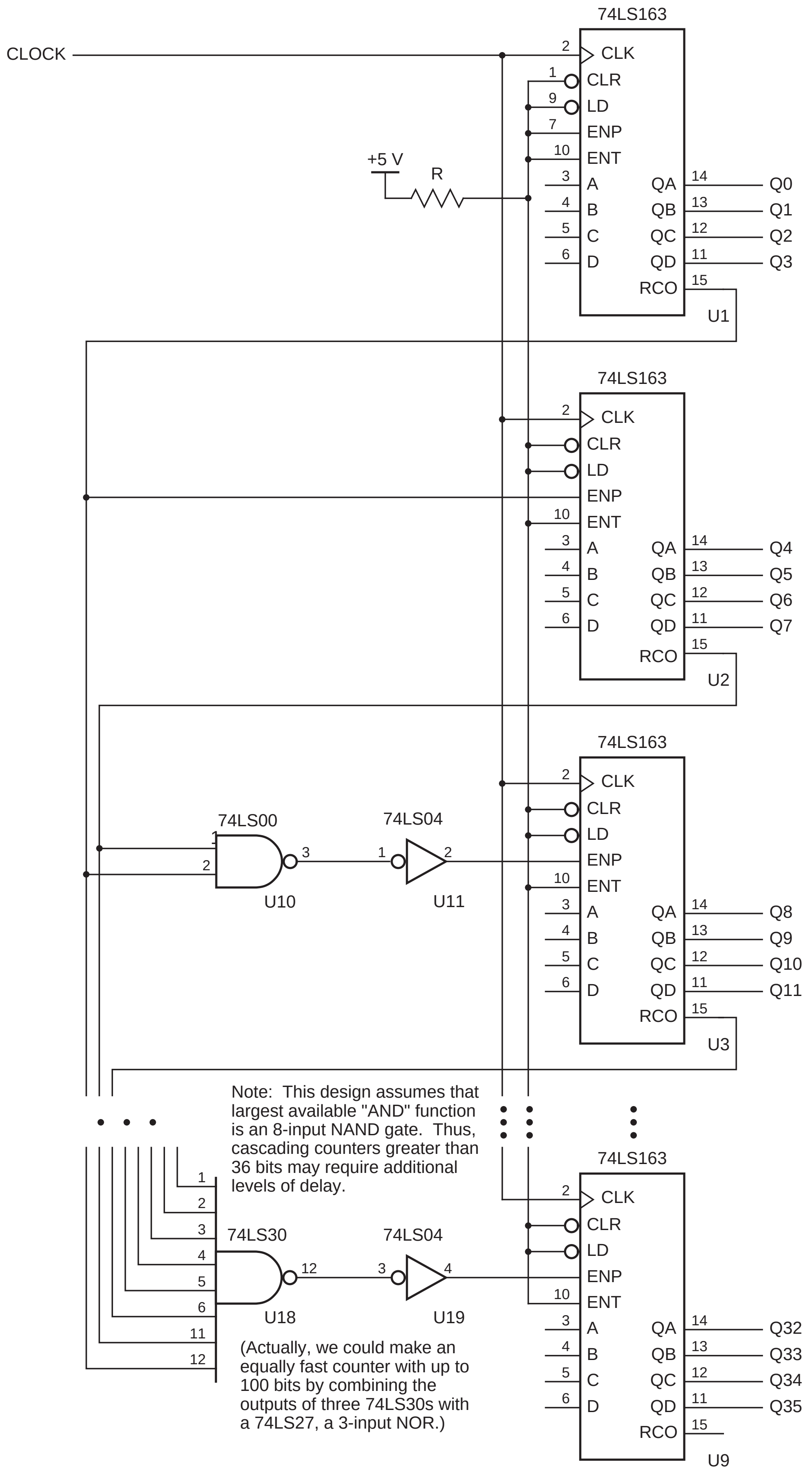
3e8.32 8.31

$$t_{\text{period(min)}} = t_{\text{pTQ}} + 3t_{\text{AND}} + t_{\text{setup}}$$
$$f_{\text{max}} = 1/t_{\text{period(min)}}$$

3e8.37 8.36

Inputs				Current state				Next state			
CLR_L	LD_L	ENT	ENP	QD	QC	QB	QA	QD*	QC*	QB*	QA*
0	x	x	x	x	x	x	x	0	0	0	0
1	0	x	x	x	x	x	x	D	C	B	A
1	1	0	x	x	x	x	x	QD	QC	QB	QA
1	1	x	0	x	x	x	x	QD	QC	QB	QA
1	1	1	1	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	1	0	0	1	0
1	1	1	1	0	0	1	0	0	0	1	1
1	1	1	1	0	0	1	1	0	1	0	0
1	1	1	1	0	1	0	0	0	1	0	1
1	1	1	1	0	1	0	1	0	1	1	0
1	1	1	1	0	1	1	0	0	1	1	1
1	1	1	1	0	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0	1	0	0	1
1	1	1	1	1	0	0	1	0	0	0	0
1	1	1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	0	1	1	0	1	0	0
1	1	1	1	1	1	0	0	1	1	0	1
1	1	1	1	1	1	0	1	0	1	0	0
1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	1	1	1	1	1	0	0	0	0

3e8.38 8.37



Note: This design assumes that largest available "AND" function is an 8-input NAND gate. Thus, cascading counters greater than 36 bits may require additional levels of delay.

(Actually, we could make an equally fast counter with up to 100 bits by combining the outputs of three 74LS30s with a 74LS27, a 3-input NOR.)

The minimum clock period is the sum of:

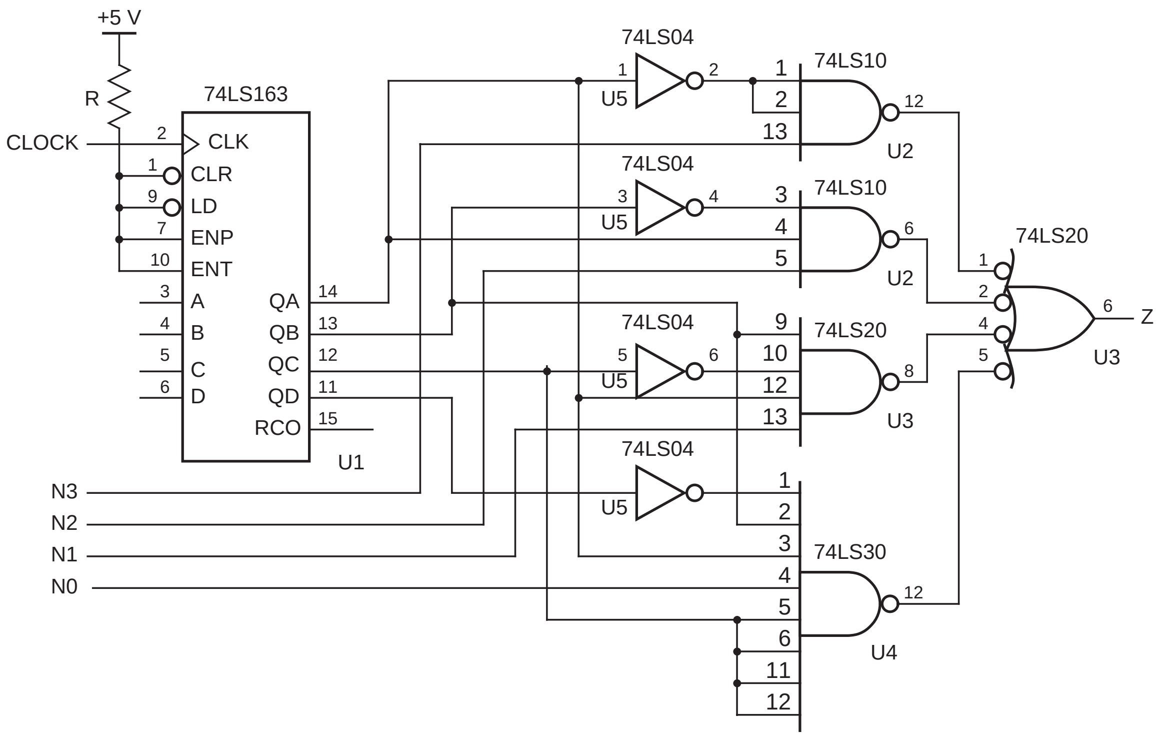
(a) The delay from the clock edge to any RCO output (35 ns).

(b) The delay from any RCO output to any ENP input, that is, two gate delays ($2 \cdot 15 = 30\,\text{ns}$).

(c) The setup time to the next clock edge required by the ENP inputs (20 ns).

Thus, the minimum clock period is 85 ns, and the corresponding maximum clock frequency is 11.76 MHz.

3e8.41 8.42 To get even spacing, the strategy is for the MSB (N3) to select half the states, the ones where QA is 0. The next bit down (N2) selects one-fourth of the states, the ones where QB is 0 and the less significant counter bits (i.e., QA) are all 1. Likewise, N1 selects the one-eighth of the states where QC is 0 and QB and QA are 1, and N0 selects the state where QD is 0 and QC, QB, and QA are all 1. In this way, each non-1111 counter state is assigned to one input bit.

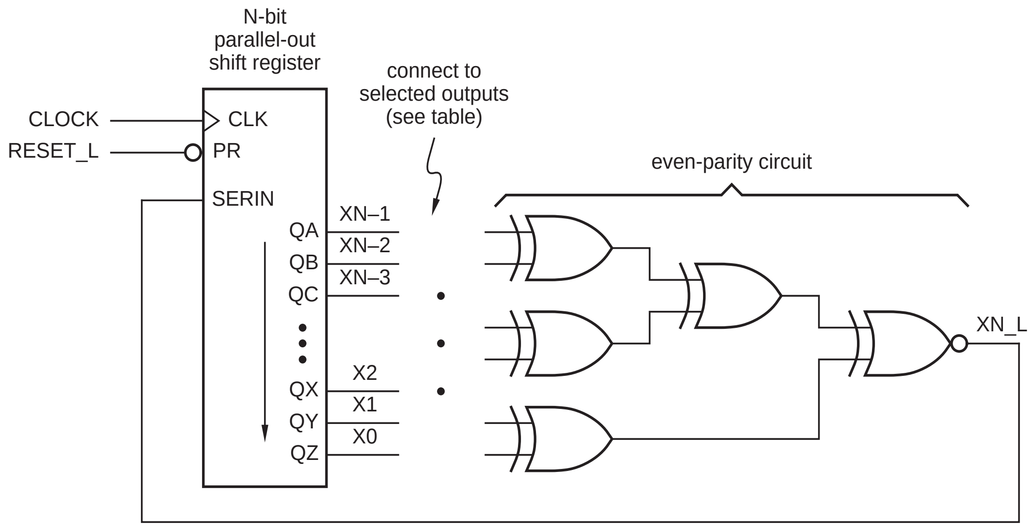


3e8.53 8.53 Please see the VHDL program `v74x163s.vhd` below or in the accompanying `.zip` file (if posted by your instructor). This program was kindly written and contributed by Chris Dunlap of Xilinx application engineering, but it has not been further checked for correctness and coding style.

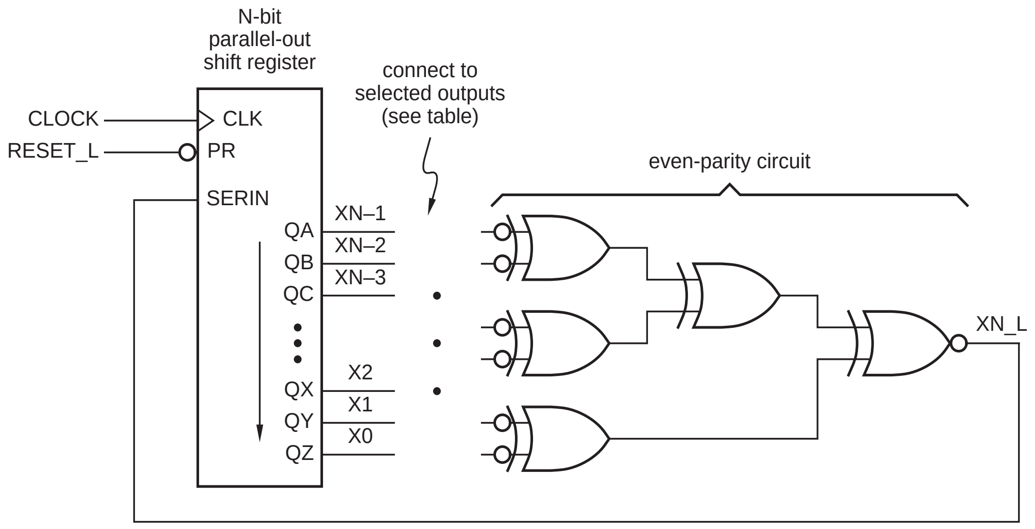
```
--Chris Dunlap
--Xilinx Applications

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity v74x163s is
generic(size : integer := 8);  --added generic
port   (clk,clr_l,ld_l,enp,ent : in std_logic;
       d : in std_logic_vector (size-1 downto 0); --changes range of input
       q : out std_logic_vector (size-1 downto 0); --changes range of output
       rco : out std_logic);
end v74x163s;

architecture v74x163_arch of v74x163s is

component synsercell is
  port (clk, ldnoclr, di, coclrorld,cntenp,cnteni : in std_logic;
       qi,cnteni1 : out std_logic);
end component;

signal ldnoclr,noclrorld : std_logic;
signal scnten : std_logic_vector (size downto 0); --creates a ranged temp with overflow room

begin
ldnoclr <= (not ld_l) and clr_l;
noclrorld <= ld_l and clr_l;
scnten(0) <= ent;
rco <= scnten(size);
gi: for i in 0 to size-1 generate --counts for size of counter
  U1: synsercell port map (clk, ldnoclr, noclrorld, enp, d(i), scnten(i), scnten(i+1), q(i));
  end generate;
end v74x163_arch;
```

3e8.62 8.60 Regardless of the number of shift-register outputs connected to the odd-parity circuit, its output in state 00...00 is 0, and the 00...00 state persists forever. However, suppose that an odd number of shift-register outputs are connected. Then the output of the odd-parity circuit in state 11...11 is 1, and the 11...11 state also persists forever. In this case, the number of states in the “maximum-length” sequence can be no more than $2^n - 2$, since two of the states persist forever. Therefore, if an LFSR counter generates a sequence of length $2^n - 1$, it must have an even number of shift-register outputs connected to the odd-parity circuit.

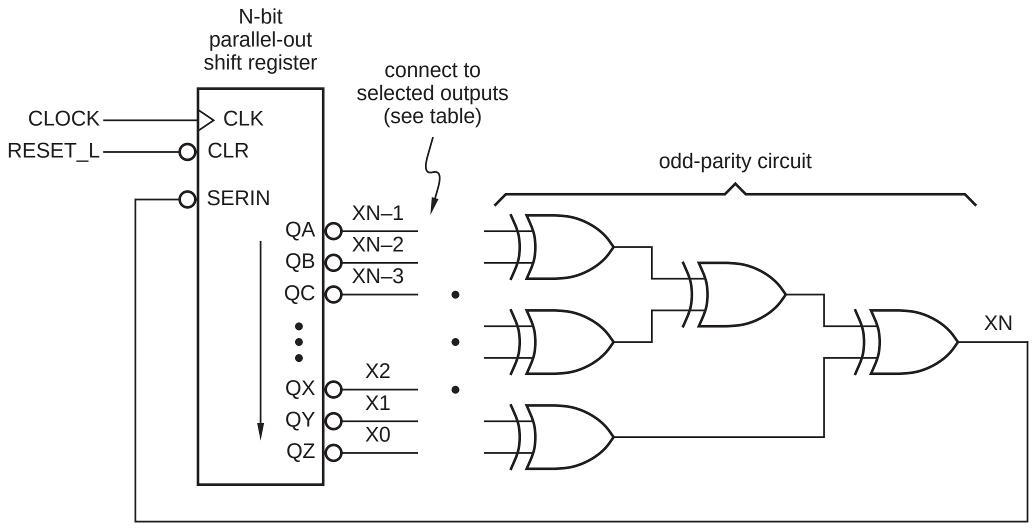
3e8.64 8.62 The figure below shows the effect of physically changing the odd-parity circuit to an even-parity circuit (i.e., inverting its output).



From Exercise 8.60, we know that an even number of shift-register outputs are connected to the parity circuit, and we also know that complementing two inputs of an XOR gate does not change its output. Therefore, we can redraw the logic diagram as shown below.



Finally, we can move the inversion bubbles as shown below.



This circuit has exactly the same structure as Figure 8–51, except that the shift register stores complemented data. When we look at the external pins of the shift register in the first figure in this solution, we are looking at that complemented data. Therefore, each state in the counting sequence of the even-parity version (our first figure) is the complement of the corresponding state in the odd-parity version (Figure 8–51). The odd-parity version visits all states except 00...00, so the even-parity version visits all states except 11...11.

3e8.68 8.69 The order of bit transmission does not matter for parity checking. A VHDL program `Parity_Check.vhd` is contained in the accompanying `.zip` file. This program was kindly written and contributed by Johnny West of Xilinx application engineering, but it has not been further checked for correctness and coding style. The second part of the program (instantiating the individual modules) clearly can be done more concisely using a `generate` statement.

3e8.75 8.76 Please see the ABEL program `TIMEGEN6.abl` below or in the accompanying `.zip` file (if posted by your instructor). In this design, RESET is not recognized until the end of phase 6. RESTART is still recognized at the end of any phase; otherwise it would have no real use (i.e., only going back to phase 1 after the end of phase 6, which happens anyway.) Presumably, RESTART would be used only with great care or in unusual circumstances (e.g., during debugging).

```
module TIMEGEN6
title 'Six-phase Master Timing Generator'

" Input and Output pins
MCLK, RESET, RUN, RESTART                    pin;
T1, P1_L, P2_L, P3_L, P4_L, P5_L, P6_L       pin istype 'reg';

" State definitions
PHASES = [P1_L, P2_L, P3_L, P4_L, P5_L, P6_L];
NEXTPH = [P6_L, P1_L, P2_L, P3_L, P4_L, P5_L];
SRESET = [1, 1, 1, 1, 1, 1];
P1 =     [0, 1, 1, 1, 1, 1];
P6 =     [1, 1, 1, 1, 1, 0];

equations
T1.CLK = MCLK; PHASES.CLK = MCLK;

WHEN (RESET & PHASES==P6 & !T1) THEN {T1 := 1; PHASES := SRESET;}
ELSE WHEN (PHASES==SRESET) # RESTART THEN {T1 := 1; PHASES := P1;}
ELSE WHEN RUN & T1 THEN {T1 := 0; PHASES := PHASES;}
ELSE WHEN RUN & !T1 THEN {T1 := 1; PHASES := NEXTPH;}
ELSE {T1 := T1; PHASES := PHASES;}

end TIMEGEN6
```

3e8.90 8.82 Transitions on SYNCIN occur a maximum of 20 ns after the rising edge of CLOCK. Given a 40-ns clock period and a 10-ns setup-time requirement for the other ’ALS74s, 10 ns is the maximum propagation delay of the combinational logic.