

**⚠ This work is protected by
US copyright laws and is for
instructors' use only.**

Lab Solutions Manual

for

Lab Manual: A Design Approach

by Gregory L. Moss

and for

Lab Manual: A Troubleshooting Approach

by Jim C. DeLoach and Frank J. Ambrosio

to accompany

DIGITAL SYSTEMS

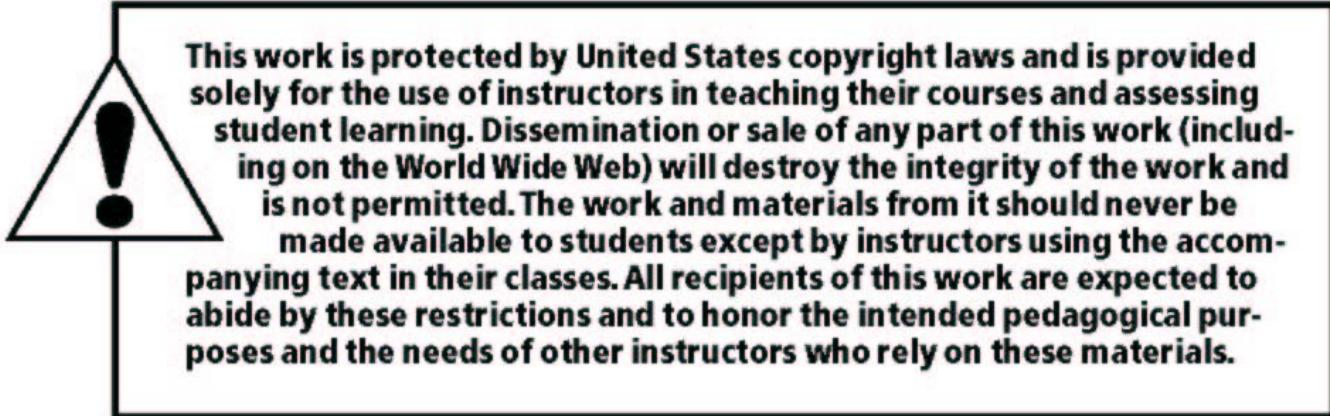
Principles and Applications

Tenth Edition

**Ronald J. Tocci
Neal S. Widmer
Gregory L. Moss**



Upper Saddle River, New Jersey
Columbus, Ohio



Copyright © 2007 by Pearson Education, Inc., Upper Saddle River, New Jersey 07458.

Pearson Prentice Hall. All rights reserved. Printed in the United States of America. This publication is protected by Copyright and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permission(s), write to: Rights and Permissions Department.

Pearson Prentice Hall™ is a trademark of Pearson Education, Inc.

Pearson® is a registered trademark of Pearson plc

Prentice Hall® is a registered trademark of Pearson Education, Inc.

Instructors of classes using Tocci, Widmer, and Moss, *Digital Systems: Principles and Applications, 10th Edition*, may reproduce material from the lab solutions manual for classroom use.

10 9 8 7 6 5 4 3 2 1

PEARSON
Prentice
Hall

ISBN 0-13-175123-9

CONTENTS

Solutions to *Lab Manual: A Design Approach, Tenth Edition* by Gregory L. Moss

Unit 1	Introduction to Digital Test Equipment	1
Unit 2	Breadboarding Combinational Logic Circuits	2
Unit 3	Analyzing Combinational Logic Circuits	3
Unit 4	Combinational Circuit Design with Karnaugh Mapping	5
Unit 5	Schematic Capture of Combinational Logic Circuits	9
Unit 6A	Combinational Circuit Design with Altera Hardware Description Language	19
Unit 6V	Combinational Circuit Design with VHDL	24
Unit 7	Analyzing Flip-Flops and Basic Sequential Circuits	31
Unit 8	Timing and Waveshaping Circuits	33
Unit 9	Arithmetic Circuits	35
Unit 10	Analyzing and Testing Synchronous Counters	41
Unit 11	Testing Register and Counter Functions	44
Unit 12	Counter Applications Using Macrofunctions	47
Unit 13	Synchronous Counter Design with Flip-Flops	55
Unit 14A	Sequential Circuit Design with Altera Hardware Description Language	62
Unit 14V	Sequential Circuit Design with VHDL	72
Unit 15	Shift Register Applications	83
Unit 16	Library of Parameterized Modules	99
Unit 17	Decoders and Displays	102
Unit 18	Encoders	112
Unit 19	Multiplexers	117
Unit 20	Demultiplexers	124
Unit 21	Magnitude Comparators	132
Unit 22	Digital/Analog and Analog/Digital Conversion	137
Unit 23	Memory Systems	144

Solutions to Lab Manual: A Troubleshooting Approach, Tenth Edition
by Jim C. DeLoach and Frank J. Ambrosio

Experiments

1	Preliminary Concepts	151
2	Logic Gates I: OR, AND, and NOT	152
3	Troubleshooting OR, AND, and NOT	153
4	Basic Combinational Circuits	154
5	Logic Gates II: NOR and NAND	*
6	Troubleshooting NOR and NAND Gates	157
7	Boolean Theorems	158
8	Simplification Using Boolean Theorems	159
9	DeMorgan's Theorems	161
10	The Universality of NAN and NOR Gates	163
11	Implementing Logic Circuit Designs	167
12	Exclusive-OR and Exclusive-NOR Circuits	170
13	Designing with Exclusive-OR and Exclusive-NOR Circuits	171
14	Troubleshooting Exclusive-OR and Combinational Circuits	*
15	Flip-Flops I: Set/Clear Latches and Clocked Flip-Flops	175
16	Flip-Flops II: D Latch; Master/Slave Flip-Flops	176
17	Schmitt Trigger, One Shots, and Astable Multivibrators	177
18	Designing with Flip-Flop Devices	181
19	Troubleshooting Flip-Flop Circuits	183
20	Binary Adders and 2's Complement System	184
21	Asynchronous Counters	187
22	BCD IC Counters	190
23	Synchronous Counters	191
24	IC Counter Application: Frequency Counter	195
25	Troubleshooting Counters: Control Waveform Generation	196
26	Shift Register Counters	199
27	IC Registers	200
28	Designing with Counters and Registers	201
29	TTL and CMOS IC Families—Part I	206
30	TTL and CMOS IC Families—Part II	207
31	Using a Logic Analyzer	*
32	IC Decoders	210
33	IC Encoders	212
34	IC Multiplexers and Demultiplexers	214
35	Troubleshooting Systems Containing MSI Logic Circuits	*
36	IC Magnitude Comparators	218

37	Data Busing	223
38	Digital-to-Analog Converters	225
39	Analog-to-Digital Converters	227
40	Semiconductor Random Access Memory (RAM)	229
41	Synchronous Counter Design (Optional)	231
42	Programmable Function Sequencer (Optional)	235

Supplemental Experiments

S 1	Logic Gates: OR, AND, and NOT	239
S 2	Basic Combinational Circuits	242
S 3	Logic Gates: NOR and NAND	245
S 4	Boolean Theorems	247
S 5	Simplification Using Boolean Theorems	249
S 6	DeMorgan's Theorems	252
S 7	Implementing Digital Gates and Circuits Using VHDL	255
S 8	Implementing Logic Circuit Designs	256
S 9	Exclusive-OR and Exclusive-NOR Circuits	259
S10	Designing with Exclusive-OR and Exclusive-NOR Circuits	261
S11	Implementing Exclusive-OR and Exclusive-NOR Circuits Using VHDL	266
S12	Latches and D-Type Flip-Flops	267
S13	J-K and T-Type Flip-Flops	272
S14	Flip-Flop Applications	275
S15	Implementing Flip-Flops and Flip-Flop Devices with VHDL	280
S16	Binary Adders and 2's Complement System	282
S17	Asynchronous Counters	284
S18	Synchronous Counters	287
S19	BCD Counters	291
S20	Shift Register Counters	293
S21	IC Registers	296
S22	One-Shots, Counters, and Registers with VHDL	299
S23	IC Decoders and Encoders	302
S24	IC Multiplexers and Demultiplexers	305
S25	VHDL State Machines	308

* = No solutions

Solutions to
Lab Manual: A Design Approach
Tenth Edition
by Gregory L. Moss

Unit 1 Introduction to Digital Test Equipment

- 1.1 TTL power supply output voltage: +5.0 V DC nominal
- 1.2 # of logic switches & # of wiring connections depends on digital tester
logic switch voltages: Up \cong +5 V "high" logic level and Down \cong 0 V "low" logic level
- 1.3 logic probe: switch up = high light and down = low light
Both LEDs are off if neither high or low (invalid) logic level
- 1.4 # of lamp monitors & # of wiring connections depends on digital tester
Switch high (1) = lamp on and switch low (0) = lamp off
Lamp monitor only indicates that a high voltage is present or not present while logic probe also indicates that a low voltage is present or invalid voltage is present (with no lights on)
- 1.5 # of logic pushbuttons or pulsers depends on digital tester
Operation depends on digital tester
- 1.6 Clock operation at low frequency: alternates between high & low signal
Speed of alternating logic signal increases if increase "fine" control
Both high & low lights appear to be on if increase "coarse" control
If both LEDs appear to be lit, signal is alternating between high & low too fast for eye to see.
- 1.7 breadboarding socket: 1 & 2 are shorted together and A & B are shorted together; all other combinations are opens
- 1.8 AND gate produces high output when both (all) inputs are high
- 1.9 OR gate produces high output when any input is high

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

- 1.10 NOT gate logic output is opposite logic level of input

B	X
0	1
1	0

Unit 2 Breadboarding Combinational Logic Circuits

2.1 One-chip circuits

$$L = \overline{\overline{A} \ B \ C} \rightarrow 7420 \quad \text{3-input NAND function}$$

$$M = \overline{\overline{A} + B} \rightarrow 7427 \quad \text{2-input NOR function}$$

$$N = \overline{A \ B \ C} \rightarrow 7408 \quad \text{3-input AND function}$$

$$R = (A + B) + C = A + B + C \rightarrow 7432 \quad \text{3-input OR function}$$

$$P = \overline{\overline{A}} = A \rightarrow 7404 \quad \text{non-inverting buffer}$$

$$Q = \overline{\overline{A} \ A} = \overline{A} \rightarrow 7400 \quad \text{NOT function}$$

2.2 Two-chip circuits

$$Z = \overline{\overline{A} \ B} \quad C = \overline{\overline{A} \ B \ C} \rightarrow 7400/7404 \quad \text{3-input NAND function}$$

$$S = \overline{\overline{A} \ B} = A + B \rightarrow 7400/7404 \quad \text{2-input OR function}$$

$$T = \overline{\overline{A}} + \overline{\overline{B}} = A \ B \rightarrow 7402/7404 \quad \text{2-input AND function}$$

$$U = \overline{(A \ B) \ (C \ D)} = \overline{\overline{A} \ \overline{B} \ \overline{C} \ \overline{D}} \rightarrow 7408/7404 \quad \text{4-input NAND}$$

2.3 Three-chip circuits

$$J = A \ \overline{B} + \overline{A} \ B = A \oplus B \rightarrow 7404/7408/7432 \quad \text{XOR}$$

$$K = A \ B + \overline{A} \ \overline{B} = \overline{\overline{A} \oplus B} \rightarrow 7404/7408/7432 \quad \text{XNOR}$$

$$V = \overline{A} \ B + A \ B + A \ \overline{B} = A + B \rightarrow 7404/7408/7432 \quad \text{OR}$$

Unit 3 Analyzing Combinational Logic Circuits

3.1 Simple circuits

$$V = B \bar{C} + \bar{A} \bar{C}$$

$$W = B C + A \bar{B}$$

A	B	C	V	W
0	0	0	1	0
0	0	1	0	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

3.2 Equivalent circuits 1

$$X = \bar{A} \bar{B} + A B + A C$$

$$Y = \bar{A} \bar{B} + A B + \bar{B} C$$

A	B	C	X	Y
0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

3.3 Equivalent circuits 2

$$P = \overline{B} \overline{C} + A \overline{C} + \overline{A} B C$$

$$Q = \overline{\overline{B} \overline{C}} \quad \overline{A \overline{C}} \quad \overline{\overline{A} B C}$$

A	B	C	P	Q
0	0	0	1	1
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

3.4 Exclusive-OR/NOR circuits

$$L = \overline{A \overline{A} B} \quad \overline{B \overline{A} B} = A \oplus B$$

$$M = \overline{A} + \overline{\overline{A} + B} + \overline{B} + \overline{\overline{A} + B} = \overline{A \oplus B}$$

3.5 Multiple-output circuit (adder)

$$S = (A \oplus B) \oplus C_i$$

$$C_o = \overline{\overline{A} B} \quad \overline{(A \oplus B)} \quad C_i$$

A	B	C _i	C _o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Unit 4 Combinational Circuit Design with Karnaugh Mapping

4.1 Two-input multiplexer

S	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$\begin{aligned}
 Y &= \bar{S} A \bar{B} + \bar{S} A B + S \bar{A} B + S A \bar{B} \\
 &= \bar{S} A + S B \quad \rightarrow \text{use 1 chip (7400)}
 \end{aligned}$$

4.2 3-bit equality detector

A	B	C	E
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

$$\begin{aligned}
 E &= \bar{A} \bar{B} C + \bar{A} B \bar{C} + \bar{A} B C + A \bar{B} \bar{C} + A \bar{B} C + A B \bar{C} \\
 &= \bar{A} C + B \bar{C} + A \bar{B} \quad \rightarrow \text{use 7410 \& 7400} \\
 \text{or} \\
 &= \bar{A} B + \bar{B} C + A \bar{C} \quad \rightarrow \text{use 7410 \& 7400}
 \end{aligned}$$

4.3 Greater than 9 detector

A	B	C	D	GT9
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$$\begin{aligned}
 GT9 &= A \bar{B} C \bar{D} + A \bar{B} C D + A B \bar{C} \bar{D} + A B \bar{C} D \\
 &\quad + A B C \bar{D} + A B C D \\
 &= A C + A B \quad \rightarrow \text{use 7400}
 \end{aligned}$$

4.4 Two-bit comparator

A1	A0	B1	B0	GE	LT	GE = A0 $\bar{B}1$ + A1 $\bar{B}1$ + A1 $\bar{B}0$
0	0	0	0	1	0	+ A1 A0 + $\bar{B}1 \bar{B}0$
0	0	0	1	0	1	
0	0	1	0	0	1	
0	0	1	1	0	1	
0	1	0	0	1	0	LT = $\bar{A}1 B1 + \bar{A}1 \bar{A}0 B0$
0	1	0	1	1	0	+ $\bar{A}0 B1 B0$
0	1	1	0	0	1	\rightarrow use 7410 & 7400
0	1	1	1	0	1	
1	0	0	0	1	0	GE = $\bar{L}T$
1	0	0	1	1	0	\rightarrow simpler for GE
1	0	1	0	1	0	
1	0	1	1	0	1	
1	1	0	0	1	0	
1	1	0	1	1	0	
1	1	1	0	1	0	
1	1	1	1	1	0	

4.5 Alarm circuit

T	P	F	L	A
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$$A = P F + T P + T L$$

$$= P F + T (P + L)$$

→ use 7408 & 7432

or 7410 & 7400

4.6 Number detector

D	C	B	A	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

$$Z = D A + C \bar{B} A + \bar{C} B A$$

→ use 7410 & 7400

(1 ea.)

4.7 Multiplier circuit

A1	A0	B1	B0	P2	P1	P0	
0	0	0	0	0	0	0	
0	0	0	1	0	0	0	
0	0	1	0	0	0	0	
0	0	1	1	0	0	0	$P_2 = A_1 \cdot B_1$
0	1	0	0	0	0	0	
0	1	0	1	0	0	1	$P_1 = A_0 \cdot B_1 + A_1 \cdot B_0$
0	1	1	0	0	1	0	
0	1	1	1	0	1	1	$P_0 = A_0 \cdot B_0$
1	0	0	0	0	0	0	
1	0	0	1	0	1	0	
1	0	1	0	1	0	0	
1	0	1	1	1	1	0	\rightarrow use 7408 & 7432
1	1	0	0	X	X	X	
1	1	0	1	X	X	X	
1	1	1	0	X	X	X	
1	1	1	1	X	X	X	

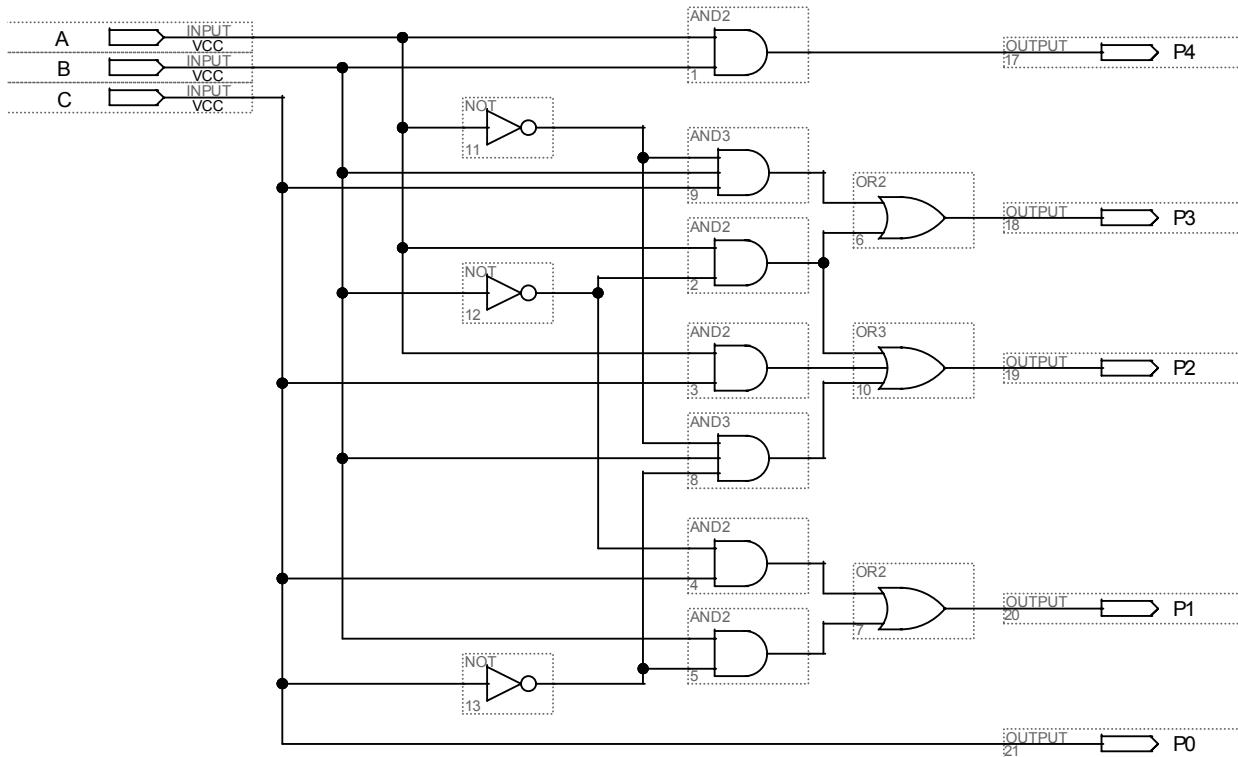
Unit 5 Schematic Capture of Combinational Logic Circuits

MAX+PLUS II & Quartus II notes to instructor:

- Student version of MAX+PLUS II & Web version of Quartus II are on the accompanying CD-ROM. Install on lab PCs and obtain software license at www.altera.com
- Schematic solutions have been drawn in MAX+PLUS II. Quartus II schematic symbols have minor differences.
- The procedures for 5.1 & 5.2 are presented in step-by-step tutorials (MAXplus schematic or Quartus schematic) on the accompanying CD-ROM.
- Each student should place their name (use the text button) on their design files (schematics) & simulation files so they can be identified when printed out. Comments can be used for student names in text design files.
- It is recommended, at least for beginning Altera labs, that the instructor establish check-off points in the lab procedures since there are so many steps for the student to deal with. Suggested student check-offs might include:
 1. Design file(s) neatly created & printed for student reporting
 2. Design compiled (check for errors or create errors for students to observe)
 3. Selection of pinout drawing in report file for printing
 4. Simulation results printed for verification & student reporting
 5. Correct wiring of programmed PLD before applying power
 6. Demonstration of working circuit

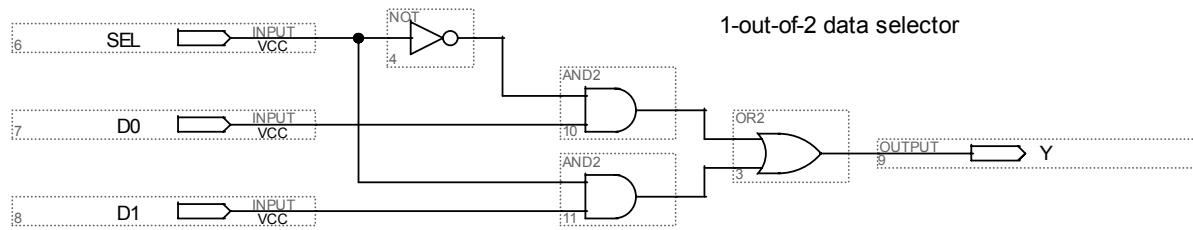
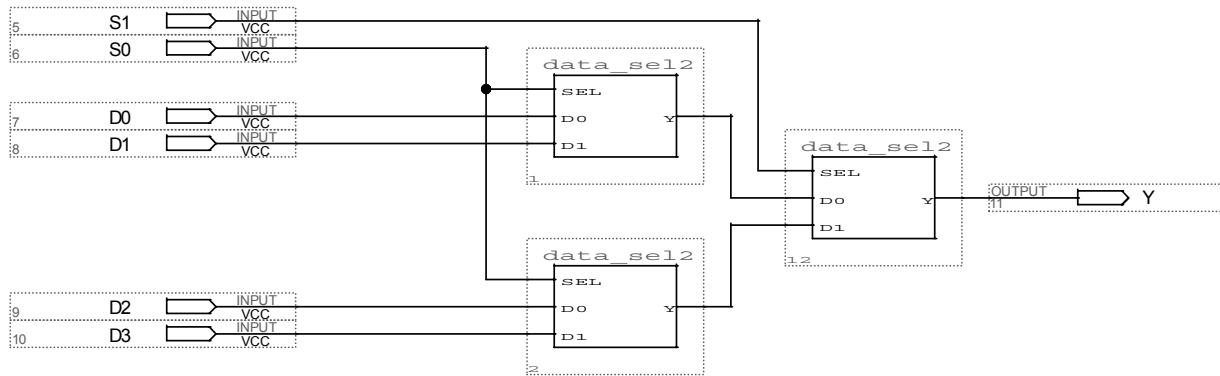
5.1 Multiply-by-3 circuit

multiplier\mult_by_3.gdf



5.2 1-out-of-4 data selector

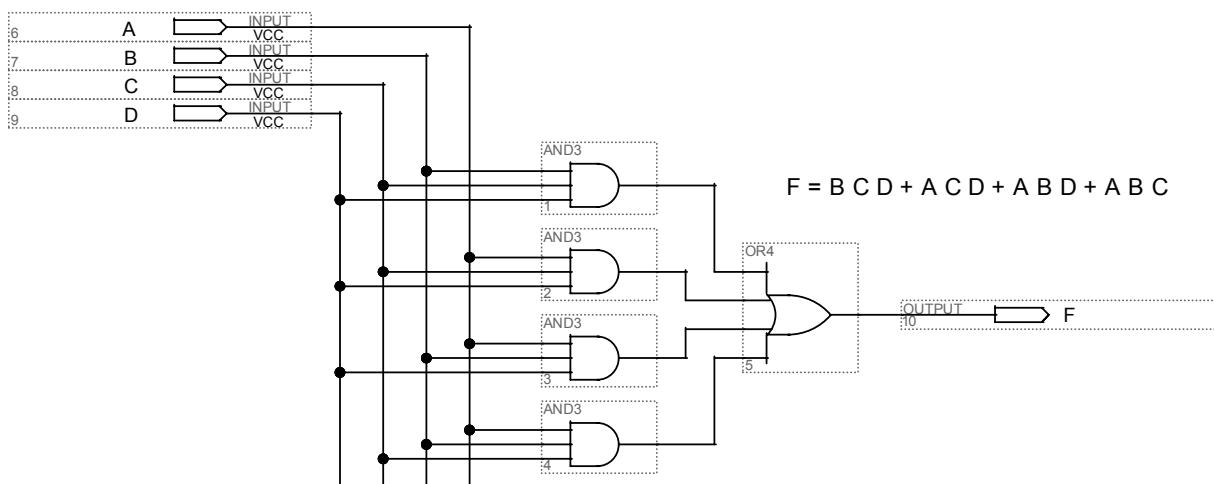
mux\data_sel4.gdf



5.3 Elevator control

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

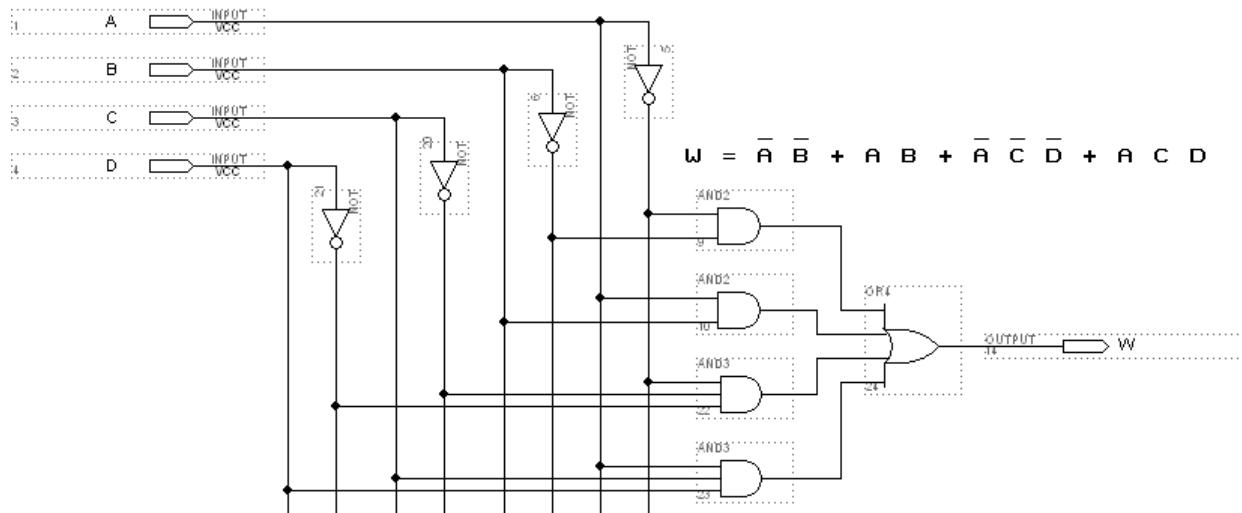
$$\begin{aligned}
 F &= \overline{\overline{A}} \cdot \overline{B} \cdot C \cdot D + A \cdot \overline{\overline{B}} \cdot C \cdot D + A \cdot B \cdot \overline{\overline{C}} \cdot D + A \cdot B \cdot C \cdot \overline{\overline{D}} + A \cdot B \cdot C \cdot D \\
 &= B \cdot C \cdot D + A \cdot C \cdot D + A \cdot B \cdot D + A \cdot B \cdot C
 \end{aligned}$$



5.4 Window detector

A	B	C	D	W
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

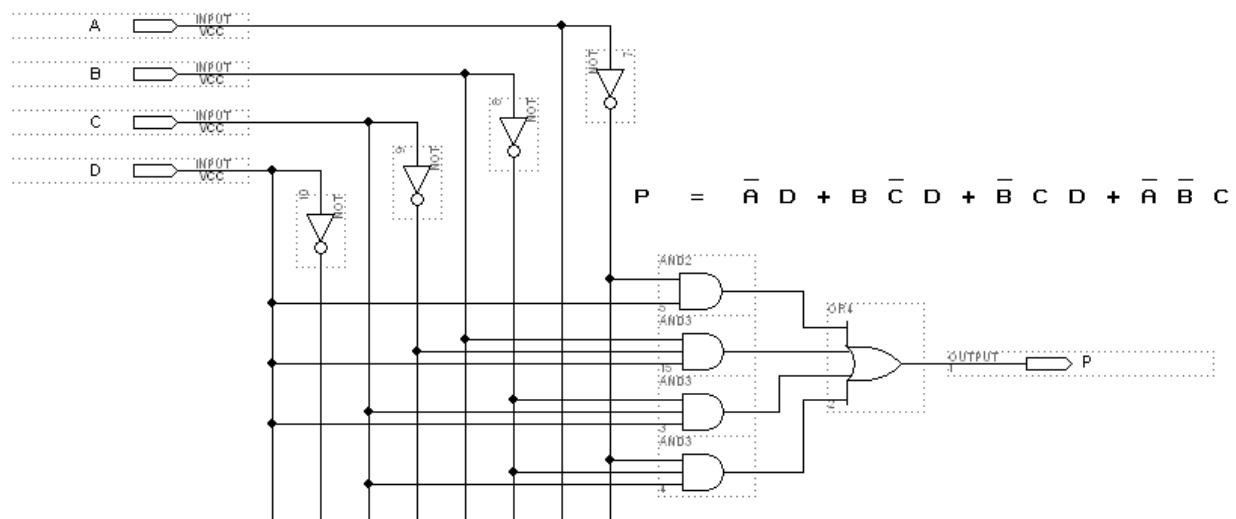
$$\begin{aligned}
 W &= \overline{A} \overline{B} + A B + \overline{A} \overline{C} \overline{D} + A C D \\
 &= \overline{A} \overline{B} + A B + B \overline{C} \overline{D} + \overline{B} C D \quad \text{or others}
 \end{aligned}$$



5.5 Prime number detector

A	B	C	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

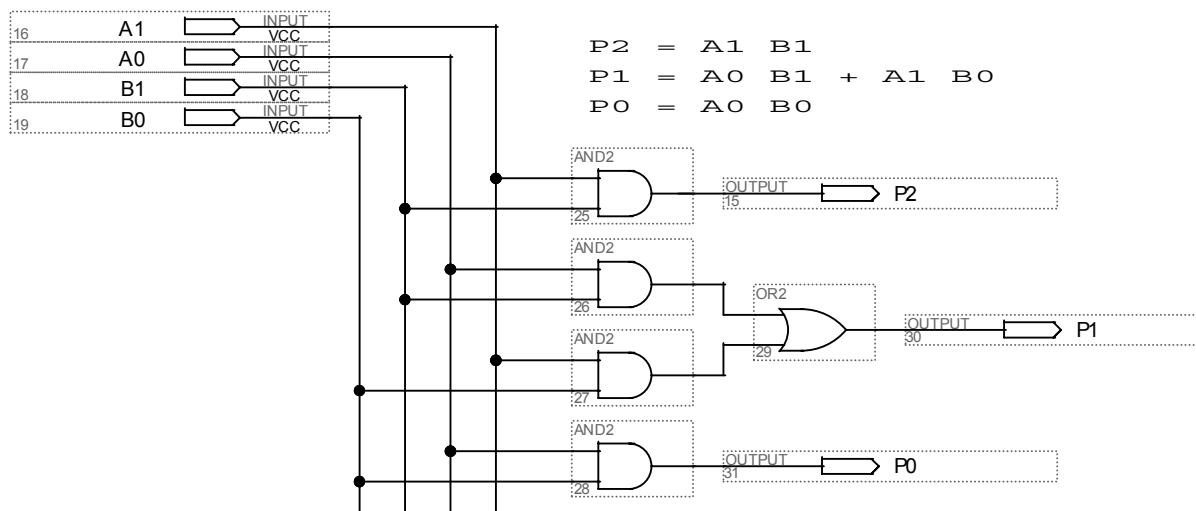
$$P = \bar{A}D + B\bar{C}D + \bar{B}CD + \bar{A}\bar{B}C$$



5.6 Multiplier circuit

A1	A0	B1	B0	P2	P1	P0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	1	1	0
1	1	0	0	X	X	X
1	1	0	1	X	X	X
1	1	1	0	X	X	X
1	1	1	1	X	X	X

$P_2 = A_1 \cdot B_1$
 $P_1 = A_0 \cdot B_1 + A_1 \cdot B_0$
 $P_0 = A_0 \cdot B_0$

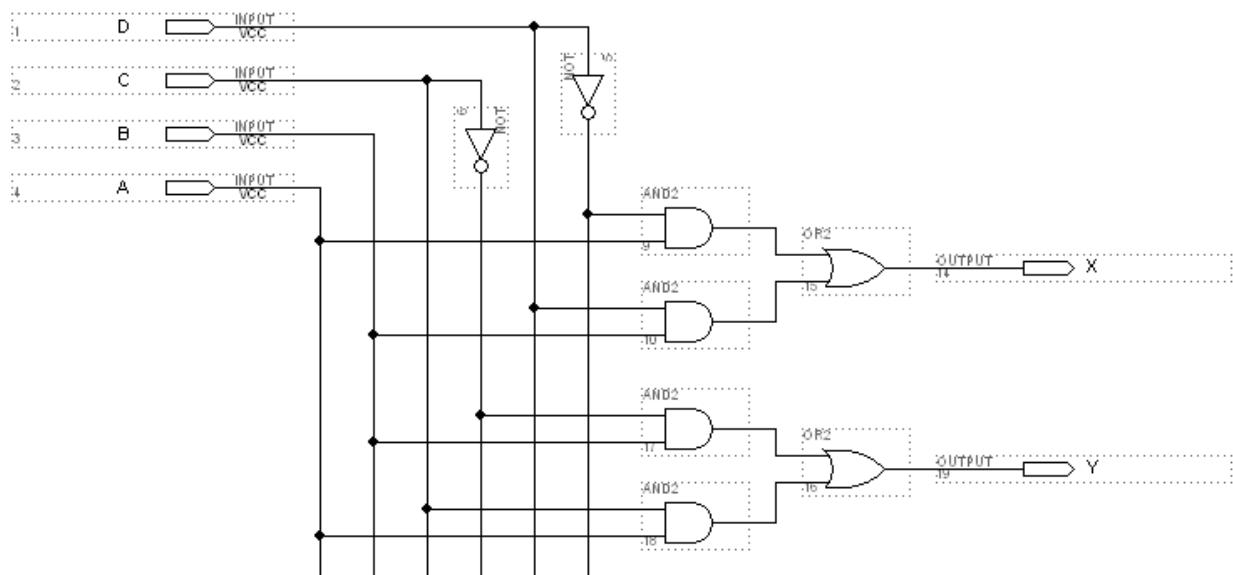


5.7 Digital switcher

D C B A	X	Y
0 0 0 0	0	0
0 0 0 1	1	0
0 0 1 0	0	1
0 0 1 1	1	1
0 1 0 0	0	0
0 1 0 1	1	1
0 1 1 0	0	0
0 1 1 1	1	1
1 0 0 0	0	0
1 0 0 1	0	0
1 0 1 0	1	1
1 0 1 1	1	1
1 1 0 0	0	0
1 1 0 1	0	1
1 1 1 0	1	0
1 1 1 1	1	1

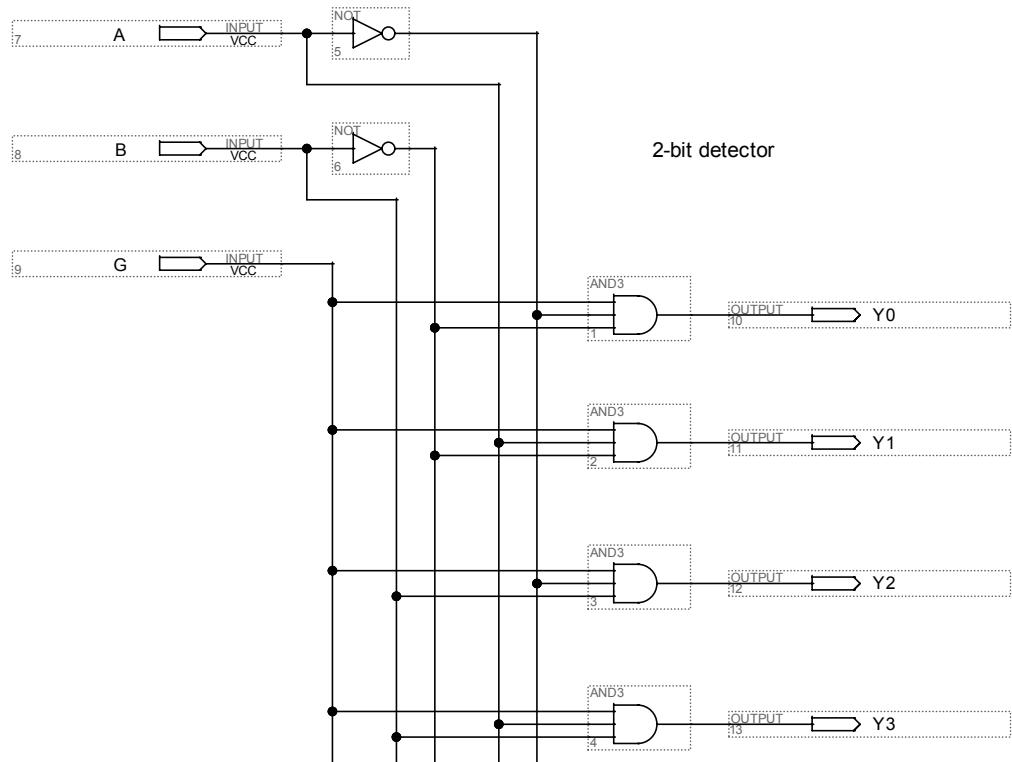
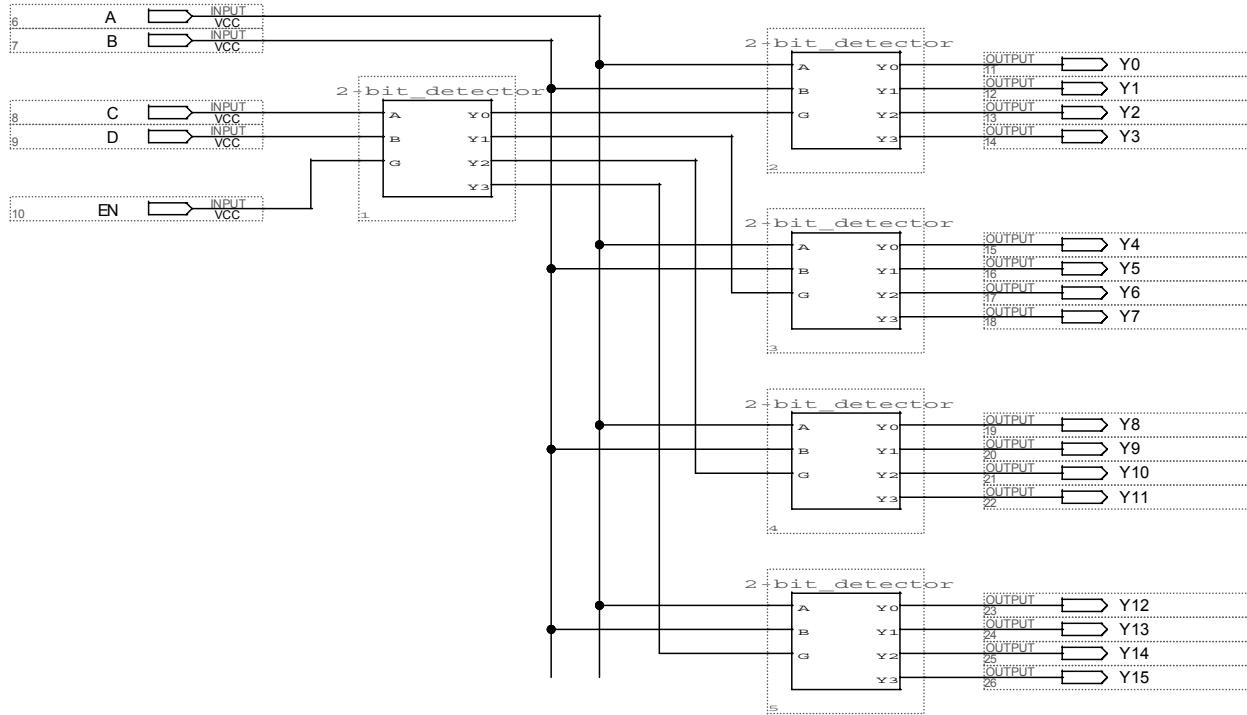
$X = \overline{D} \cdot A + D \cdot B$

$Y = \overline{C} \cdot B + C \cdot A$



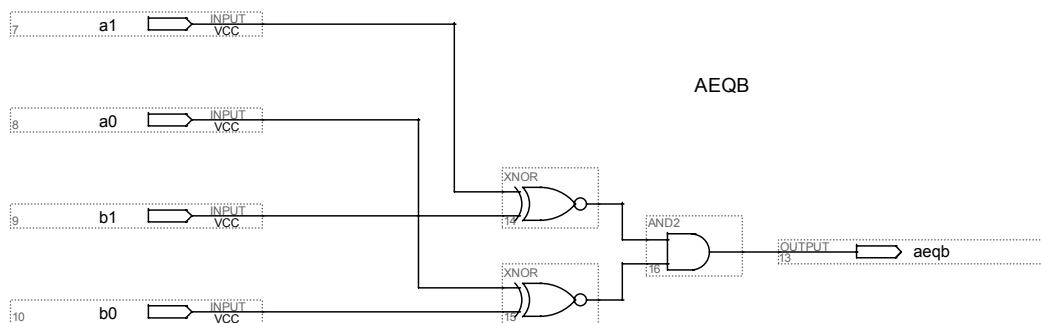
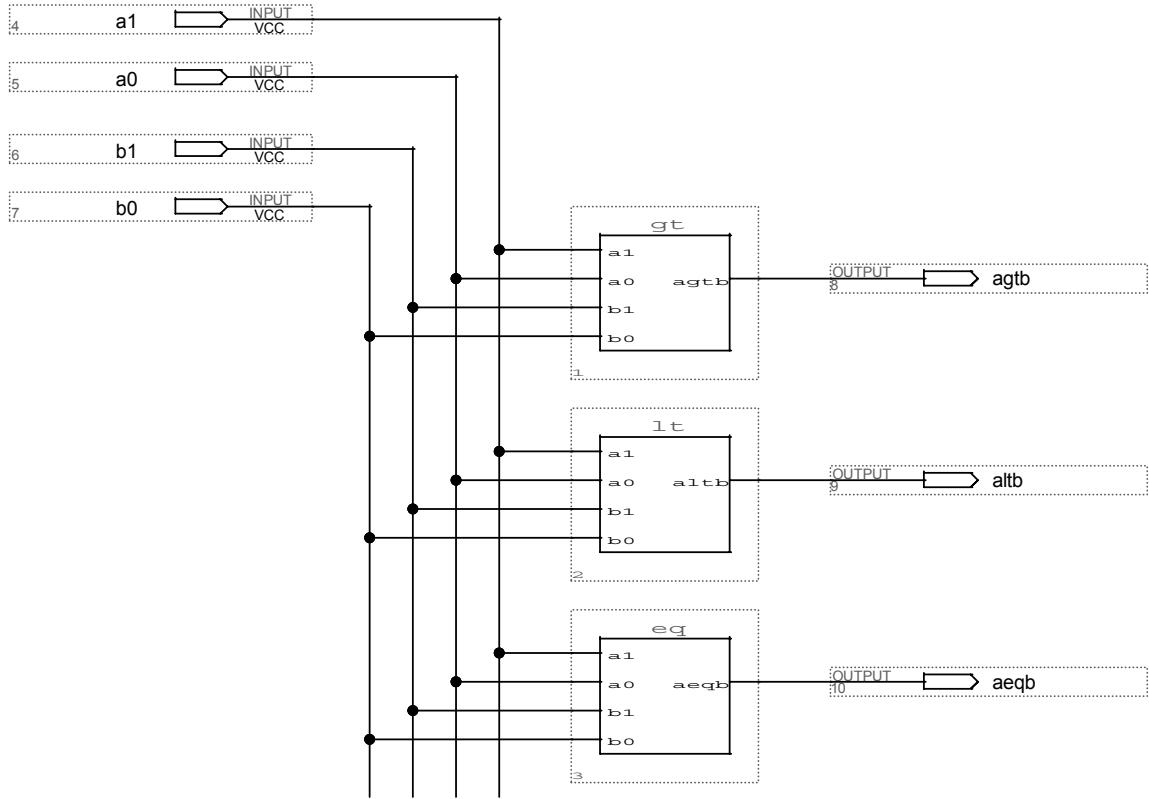
5.8 Binary number detector

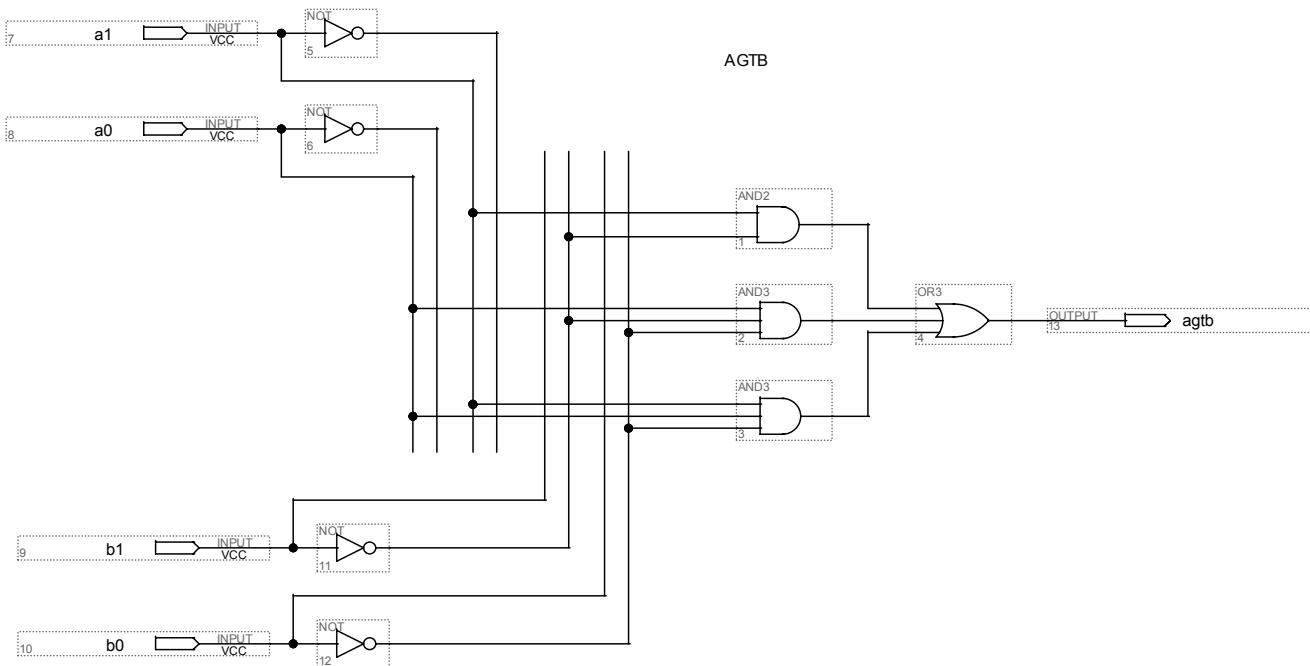
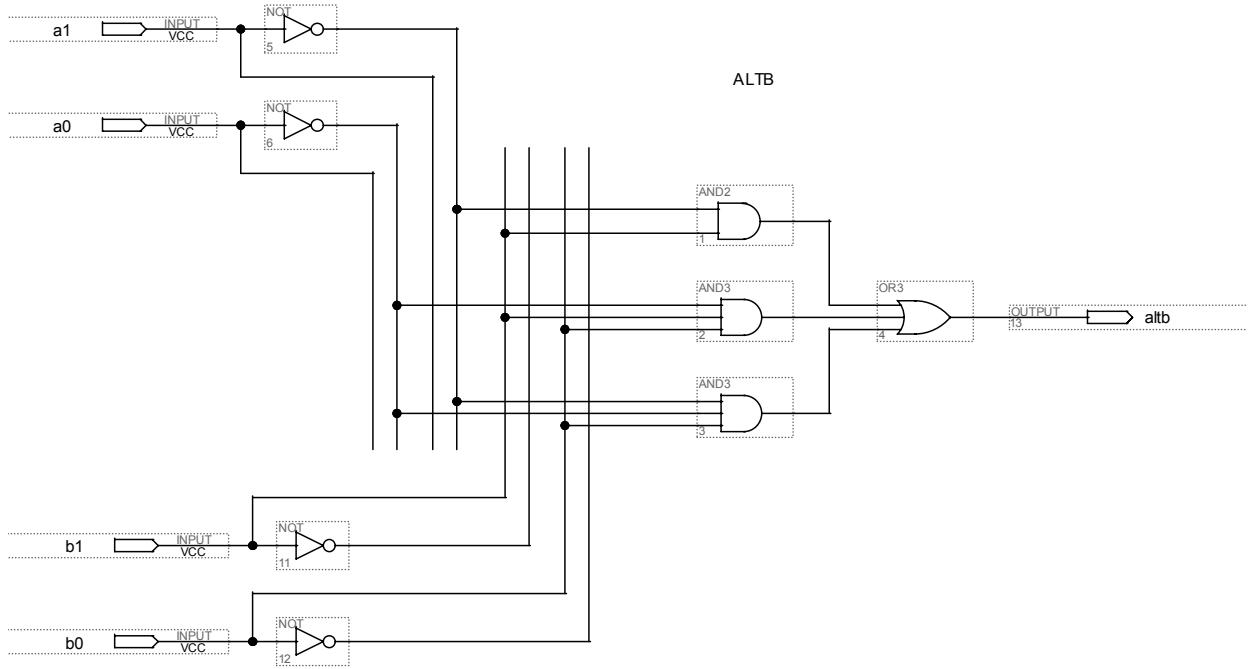
bin_num_detector\4-bit_detector.gdf



5.9 2-bit comparator

compare_schematic\compare.gdf





Unit 6A Combinational Circuit Design with Altera Hardware Description Language

- 6A.1 see lab manual Example 6-1 & AHDL tutorial (MAXplus AHDL or Quartus AHDL)
- 6A.2 see lab manual Example 6-2 & AHDL tutorial (MAXplus AHDL or Quartus AHDL)
- 6A.3 see lab manual Example 6-3 & AHDL tutorial (MAXplus AHDL or Quartus AHDL)
- 6A.4 BCD code to 2421 code conversion

```
SUBDESIGN code2421
(
    d, c, b, a      : INPUT;
    p, q, r, s      : OUTPUT;
)

VARIABLE
    nc, nb, na      : NODE;      -- declare buried nodes
BEGIN
    nc = !c;           -- NOT functions
    nb = !b;
    na = !a;
    p = d # c & a # c & b;      -- Boolean functions
    q = d # c & b # c & na;
    r = d # nc & b # c & nb & a;
    s = a;
END;
```

- 6A.5 2-bit binary adder

```
SUBDESIGN adder
(
    a[2..1], b[2..1]      : INPUT;
    s[3..1]                : OUTPUT;
)

VARIABLE
    c1                      : NODE;      -- buried node
BEGIN
    c1 = a[1] & b[1];          -- AND
    s[1] = a[1] $ b[1];        -- XOR
    s[2] = a[2] $ b[2] $ c1;
    s[3] = a[2] & b[2] # (a[2] $ b[2]) & c1;
END;
```

6A.6 Gray-code-to-binary conversion

```
SUBDESIGN gray
(
    gray[5..0]      :INPUT;
    bin[5..0]       :OUTPUT;
)
BEGIN
    bin5 = gray5;           -- MSBs are the same
    bin4 = bin5 $ gray4;   -- XOR prev out $ next in
    bin3 = bin4 $ gray3;
    bin2 = bin3 $ gray2;
    bin1 = bin2 $ gray1;
    bin0 = bin1 $ gray0;
END;
```

6A.7 BCD-to-binary converter

```
SUBDESIGN BCD2bin
(
    bcd[4..0]      :INPUT;
    bin[3..0], err :OUTPUT;
)
BEGIN
    DEFAULTS          -- error results
        bin[] = B"0000";
        err = VCC;
    END DEFAULTS;
    TABLE            -- define valid results
        bcd[]    => bin[], err;
        H"00"    => H"0", 0;
        H"01"    => H"1", 0;
        H"02"    => H"2", 0;
        H"03"    => H"3", 0;
        H"04"    => H"4", 0;
        H"05"    => H"5", 0;
        H"06"    => H"6", 0;
        H"07"    => H"7", 0;
        H"08"    => H"8", 0;
        H"09"    => H"9", 0;
        H"10"    => H"A", 0;
        H"11"    => H"B", 0;
        H"12"    => H"C", 0;
        H"13"    => H"D", 0;
        H"14"    => H"E", 0;
        H"15"    => H"F", 0;
    END TABLE;
END;
```

6A.8 Binary-to-BCD converter

```
SUBDESIGN bin2BCD
(
    bin[4..0]      :INPUT;
    bcd[5..0]      :OUTPUT;
)
BEGIN
    TABLE          -- use hex to define each BCD digit
        bin[]      => bcd[];
        B"00000"   => H"00";
        B"00001"   => H"01";
        B"00010"   => H"02";
        B"00011"   => H"03";
        B"00100"   => H"04";
        B"00101"   => H"05";
        B"00110"   => H"06";
        B"00111"   => H"07";
        B"01000"   => H"08";
        B"01001"   => H"09";
        B"01010"   => H"10";
        B"01011"   => H"11";
        B"01100"   => H"12";
        B"01101"   => H"13";
        B"01110"   => H"14";
        B"01111"   => H"15";
        B"10000"   => H"16";
        B"10001"   => H"17";
        B"10010"   => H"18";
        B"10011"   => H"19";
        B"10100"   => H"20";
        B"10101"   => H"21";
        B"10110"   => H"22";
        B"10111"   => H"23";
        B"11000"   => H"24";
        B"11001"   => H"25";
        B"11010"   => H"26";
        B"11011"   => H"27";
        B"11100"   => H"28";
        B"11101"   => H"29";
        B"11110"   => H"30";
        B"11111"   => H"31";
    END TABLE;
END;
```

6A.9 Tens digit detector

```
SUBDESIGN decade
(
    num[5..0]           :INPUT;
    tens[6..0]          :OUTPUT;
)
BEGIN
    -- 1st true condition determines output
    IF      num[] <= 9     THEN    tens[] = B"0000001";
    ELSIF   num[] <= 19    THEN    tens[] = B"0000010";
    ELSIF   num[] <= 29    THEN    tens[] = B"0000100";
    ELSIF   num[] <= 39    THEN    tens[] = B"0001000";
    ELSIF   num[] <= 49    THEN    tens[] = B"0010000";
    ELSIF   num[] <= 59    THEN    tens[] = B"0100000";
    ELSE
        tens[] = B"1000000";
    END IF;
END;
```

6A.10 Lamp display

```
SUBDESIGN lamps
(
    in[2..0]           :INPUT;
    out[7..1]          :OUTPUT;
)
BEGIN
    -- test every in[] condition
    CASE in[] IS
        WHEN 0      =>    out[] = B"0000000";
        WHEN 1      =>    out[] = B"0000001";
        WHEN 2      =>    out[] = B"0000011";
        WHEN 3      =>    out[] = B"0000111";
        WHEN 4      =>    out[] = B"0001111";
        WHEN 5      =>    out[] = B"0011111";
        WHEN 6      =>    out[] = B"0111111";
        WHEN 7      =>    out[] = B"1111111";
    END CASE;
END;
```

6A.11 Programmable logic unit

```
SUBDESIGN logic_unit
(   a[3..0], b[3..0], s[1..0], en           :INPUT;
    f[3..0]                      :OUTPUT;   )
BEGIN
    -- control input selects function
    IF     s[] == 0 & en  THEN    f[] = a[] # b[];
    ELSIF  s[] == 1 & en  THEN    f[] = a[] & b[];
    ELSIF  s[] == 2 & en  THEN    f[] = a[] $ b[];
    ELSIF  s[] == 3 & en  THEN    f[] = !a[];
    ELSE    f[] = B"0000";      -- output disabled
    END IF;
END;
```

6A.12 Number range detector

```
SUBDESIGN range
(   num[4..0], en      :INPUT;
    range[4..1]        :OUTPUT;   )
BEGIN
    -- determine if num[] in ranges when enabled
    -- active-hi outs
    range1 =  num[] >= 4 & num[] <= 12 & en;
    range2 =  num[] >= 15 & num[] <= 20 & en;
    range3 =  num[] >= 18 & num[] <= 24 & en;
    -- active-low out
    range4 = !(num[] >= 26 & num[] <= 30 & en);
END;
```

6A.13 Data switcher

```
SUBDESIGN switcher
(   en, in[1..0], s[1..0]      :INPUT;
    out[1..0]                 :OUTPUT;   )
BEGIN
    IF !en      THEN      -- detect active-low enable
        CASE s[] IS      -- determine control input
            WHEN 0 => out1 = in1; out0 = in0;
            WHEN 1 => out1 = in0; out0 = in0;
            WHEN 2 => out1 = in1; out0 = in1;
            WHEN 3 => out1 = in0; out0 = in1;
        END CASE;
    ELSE    out[] = B"00"; -- disabled
    END IF;
END;
```

Unit 6V Combinational Circuit Design with VHDL

- 6V.1 see lab manual Example 6-1 & VHDL tutorial (MAXplus VHDL or Quartus VHDL)
- 6V.2 see lab manual Example 6-2 & VHDL tutorial (MAXplus VHDL or Quartus VHDL)
- 6V.3 see lab manual Example 6-3 & VHDL tutorial (MAXplus VHDL or Quartus VHDL)
- 6V.4 BCD code to 2421 code conversion

```
ENTITY code2421 IS
  PORT
  (
    d, c, b, a      : IN BIT;
    p, q, r, s      : OUT BIT
  );
END code2421 ;

ARCHITECTURE karnaugh OF code2421 IS
SIGNAL nc, nb, na          : BIT;           -- declare buried nodes
BEGIN
  nc <= NOT c;                      -- NOT functions
  nb <= NOT b;
  na <= NOT a;
  p <= d OR (c AND a) OR (c AND b);      -- Boolean
  q <= d OR (c AND b) OR (c AND na);     -- functions
  r <= d OR (nc AND b) OR (c AND nb AND a);
  s <= a;
END karnaugh ;
```

- 6V.5 2-bit binary adder

```
ENTITY adder IS
  PORT ( a, b          : IN BIT_VECTOR (2 DOWNTO 1);
         s          : OUT BIT_VECTOR (3 DOWNTO 1) );
END adder;

ARCHITECTURE twobit OF adder IS
SIGNAL c1          : BIT;           -- buried node
BEGIN
  c1    <= a(1) AND b(1);
  s(1) <= a(1) XOR b(1);
  s(2) <= a(2) XOR b(2) XOR c1;
  s(3) <= (a(2) AND b(2)) OR ((a(2) XOR b(2)) AND c1);
END twobit;
```

6V.6 Gray-code-to-binary conversion

```

ENTITY gray IS
PORT (      gray      : IN BIT_VECTOR (5 DOWNTO 0);
            bin       : OUT BIT_VECTOR (5 DOWNTO 0)      );
END gray;
ARCHITECTURE converter OF gray IS
SIGNAL temp          : BIT_VECTOR (5 DOWNTO 0);
BEGIN
    temp(5) <= gray(5);           -- MSBs are the same
    temp(4) <= temp(5) XOR gray(4);
    temp(3) <= temp(4) XOR gray(3);
    temp(2) <= temp(3) XOR gray(2);
    temp(1) <= temp(2) XOR gray(1);
    temp(0) <= temp(1) XOR gray(0);
    bin <= temp;                -- assign temp to output
END converter;

```

6V.7 BCD-to-binary converter

```

ENTITY bcd2bin IS
PORT (      bcd       : IN BIT_VECTOR (4 DOWNTO 0);
            bin       : OUT BIT_VECTOR (3 DOWNTO 0);
            err       : OUT BIT        );
END bcd2bin;
ARCHITECTURE a OF bcd2bin IS
BEGIN
    PROCESS (bcd)           -- bcd change invokes process
    BEGIN
        CASE bcd IS         -- test all bcd conditions
            WHEN "0000" => bin <= "000";   err <= '0';
            WHEN "0001" => bin <= "001";   err <= '0';
            WHEN "0010" => bin <= "010";   err <= '0';
            WHEN "0011" => bin <= "011";   err <= '0';
            WHEN "0100" => bin <= "100";   err <= '0';
            WHEN "0101" => bin <= "101";   err <= '0';
            WHEN "0110" => bin <= "110";   err <= '0';
            WHEN "0111" => bin <= "111";   err <= '0';
            WHEN "1000" => bin <= "1000";  err <= '0';
            WHEN "1001" => bin <= "1001";  err <= '0';
            WHEN "10000" => bin <= "1010";  err <= '0';
            WHEN "10001" => bin <= "1011";  err <= '0';
            WHEN "10010" => bin <= "1100";  err <= '0';
            WHEN "10011" => bin <= "1101";  err <= '0';
            WHEN "10100" => bin <= "1110";  err <= '0';
            WHEN "10101" => bin <= "1111";  err <= '0';
            WHEN OTHERS => bin <= "0000";  err <= '1';
        END CASE;
    END PROCESS;
END a;

```

6V.8 Binary-to-BCD converter

```
ENTITY bin2bcd IS
PORT (      bin      : IN BIT_VECTOR (4 DOWNTO 0);
            bcd      : OUT BIT_VECTOR (5 DOWNTO 0)      );
END bin2bcd;
ARCHITECTURE a OF bin2bcd IS
BEGIN
    PROCESS (bin)          -- bin change invokes process
    BEGIN
        CASE bin IS      -- check all input values
            WHEN "00000" => bcd <= "000000";
            WHEN "00001" => bcd <= "000001";
            WHEN "00010" => bcd <= "000010";
            WHEN "00011" => bcd <= "000011";
            WHEN "00100" => bcd <= "000100";
            WHEN "00101" => bcd <= "000101";
            WHEN "00110" => bcd <= "000110";
            WHEN "00111" => bcd <= "000111";
            WHEN "01000" => bcd <= "001000";
            WHEN "01001" => bcd <= "001001";
            WHEN "01010" => bcd <= "010000";
            WHEN "01011" => bcd <= "010001";
            WHEN "01100" => bcd <= "010010";
            WHEN "01101" => bcd <= "010011";
            WHEN "01110" => bcd <= "010100";
            WHEN "01111" => bcd <= "010101";
            WHEN "10000" => bcd <= "010110";
            WHEN "10001" => bcd <= "010111";
            WHEN "10010" => bcd <= "011000";
            WHEN "10011" => bcd <= "011001";
            WHEN "10100" => bcd <= "100000";
            WHEN "10101" => bcd <= "100001";
            WHEN "10110" => bcd <= "100010";
            WHEN "10111" => bcd <= "100011";
            WHEN "11000" => bcd <= "100100";
            WHEN "11001" => bcd <= "100101";
            WHEN "11010" => bcd <= "100110";
            WHEN "11011" => bcd <= "100111";
            WHEN "11100" => bcd <= "101000";
            WHEN "11101" => bcd <= "101001";
            WHEN "11110" => bcd <= "110000";
            WHEN "11111" => bcd <= "110001";
        END CASE;
    END PROCESS;
END a;
```

6V.9 Tens digit detector

```
ENTITY decade IS
PORT (      num      :IN INTEGER RANGE 0 TO 63;
            tens     :OUT BIT_VECTOR (6 DOWNTO 0)      );
END decade;

ARCHITECTURE b OF decade IS
BEGIN
    PROCESS (num)          -- num invokes process
    BEGIN -- 1st true condition determines output
        IF      num <= 9  THEN    tens <= B"0000001";
        ELSIF  num <= 19  THEN   tens <= B"0000010";
        ELSIF  num <= 29  THEN   tens <= B"0000100";
        ELSIF  num <= 39  THEN   tens <= B"0001000";
        ELSIF  num <= 49  THEN   tens <= B"0010000";
        ELSIF  num <= 59  THEN   tens <= B"0100000";
        ELSE
                tens <= B"1000000";
        END IF;
    END PROCESS;
END b;
```

6V.10 Lamp display

```
ENTITY lamps IS
PORT (      num      :INTEGER RANGE 0 TO 7;
            lites    :OUT BIT_VECTOR (7 DOWNTO 1)      );
END lamps;

ARCHITECTURE cases OF lamps IS
BEGIN
    PROCESS (num)          -- num invokes process
    BEGIN
        CASE num IS      -- test every num condition
            WHEN 0      =>    lites <= "0000000";
            WHEN 1      =>    lites <= "0000001";
            WHEN 2      =>    lites <= "0000011";
            WHEN 3      =>    lites <= "0000111";
            WHEN 4      =>    lites <= "0001111";
            WHEN 5      =>    lites <= "0011111";
            WHEN 6      =>    lites <= "0111111";
            WHEN 7      =>    lites <= "1111111";
        END CASE;
    END PROCESS;
END cases;
```

6V.11 Programmable logic unit

```
ENTITY logic_unit IS
PORT (      a, b          : IN BIT_VECTOR (3 DOWNTO 0);
            s          : IN BIT_VECTOR (1 DOWNTO 0);
            en         : IN BIT;
            f          : OUT BIT_VECTOR (3 DOWNTO 0)      );
END logic_unit;

ARCHITECTURE program OF logic_unit IS
BEGIN
    PROCESS (a, b, s, en)      -- any input change
    BEGIN
        -- control input selects function
        IF           s = "00" AND en = '1'
                    THEN      f <= a OR b;
        ELSIF        s = "01" AND en = '1'
                    THEN      f <= a AND b;
        ELSIF        s = "10" AND en = '1'
                    THEN      f <= a XOR b;
        ELSIF        s = "11" AND en = '1'
                    THEN      f <= NOT a;
        ELSE         f <= "0000";      -- disabled
        END IF;
    END PROCESS;
END program;
```

6V.12 Number range detector

```
ENTITY val_range IS
PORT (    en          : IN BIT;
           num         : IN INTEGER RANGE 0 TO 31;
           values      : OUT BIT_VECTOR (4 DOWNTO 1) );
END val_range;
ARCHITECTURE proj OF val_range IS
BEGIN
    PROCESS (en, num)
    BEGIN
        IF en = '1' THEN                      -- enabled
            IF      (num >= 4 AND num <= 12)
                THEN values <= "1001";
            ELSIF (num >= 15 AND num <= 20)
                THEN values <= "1010";
            ELSIF (num >= 21 AND num <= 24)
                THEN values <= "1100";
            ELSIF (num >= 26 AND num <= 30)
                THEN values <= "0000";
            ELSE           values <= "1000";
            END IF;      -- determine num
        ELSE           values <= "1000";      -- disabled
        END IF;
    END PROCESS;
END proj;
```

6V.13 Data switcher

```
ENTITY switcher IS
PORT ( en : IN BIT;
       input, sel : IN BIT_VECTOR (1 DOWNTO 0);
       output : OUT BIT_VECTOR (1 DOWNTO 0) );
END switcher;

ARCHITECTURE switch OF switcher IS
BEGIN
    PROCESS (en, input, sel)          -- sensitivity list
    BEGIN
        IF en = '0'      THEN      -- active-low enable
            CASE sel IS
                WHEN "00"  =>
                    output(1) <= input(1);
                    output(0) <= input(0);
                WHEN "01"  =>
                    output(1) <= input(0);
                    output(0) <= input(0);
                WHEN "10"  =>
                    output(1) <= input(1);
                    output(0) <= input(1);
                WHEN "11"  =>
                    output(1) <= input(0);
                    output(0) <= input(1);
            END CASE;
        ELSE
            output <= "00";      -- disabled
        END IF;
    END PROCESS;
END switch;
```

Unit 7 Analyzing Flip-Flops and Basic Sequential Circuits

7.1 NAND SR latch

Set	Reset	Q	Qbar	
0	0	1	1	invalid
0	1	1	0	set
1	0	0	1	reset
1	1	Q	Qbar	hold

active-low control inputs

7.2 Debouncing a logic switch

- (a) counter appears to count erratically due to receiving multiple pulses on clock input from bouncing switch
- (b) counter should count correctly with latch
 - pull-up resistors should not appear to be needed if TTL gates used since open inputs in TTL will float high, but they should be used to reduce noise problems

7.3 D latch

D latch is transparent when $EN = 1$, stores data when $EN = 0$

7.4 4-bit register

Q_1 = LSB & Q_4 = MSB

7474 contains D latches that are positive-edge triggered
register holds data between triggers
level enabled latch would read input data continuously while enabled

7.5 JK flip-flop

see 74112 data sheets

J & K inputs are synchronous, active-low PRE & CLR are asynchronous controls
require clock for triggering, asynchronous controls have priority, NGT clock is triggering control

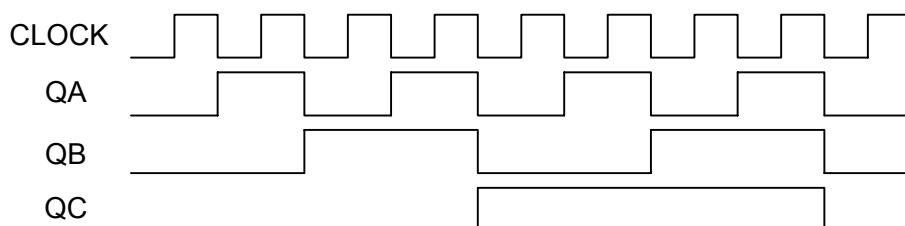
7.6 Binary counter

count sequence: 000 thru 111 & recycle

mod-8 up counter

synchronous counter → FFs are clocked simultaneously

7.7 Counter timing diagram



7.8 Frequency division

(assuming 10 kHz clock input frequency)

Signal	Frequency	Frequency Relationship
Clock	10 kHz	Input (assumed frequency)
QA	5 kHz	$\text{Freq}_{\text{QA}} = \text{Freq}_{\text{Clock}} \div 2$
QB	2.5 kHz	$\text{Freq}_{\text{QB}} = \text{Freq}_{\text{Clock}} \div 4$
QC	1.25 kHz	$\text{Freq}_{\text{QC}} = \text{Freq}_{\text{Clock}} \div 8$

7.9 Modifying count sequence

QC QB QA count sequence:

$$000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100 \rightarrow 000$$

NAND gate output goes low when count reaches 101 and asynchronously clears counter.
Mod-5 counter.

The CLR input is active-low.

CLR immediately clears the counter to zero.

The CLR control is asynchronous since it clears right away instead of waiting on the next clock.

The NAND gate detects when the QC and QBnot and QA outputs are simultaneously high.

A mod-6 counter would be produced if the NAND gate inputs are QC and QB and QAnot.

A mod-7 counter would be produced if the NAND gate inputs are QC and QB and QA.

7.10 Counter using D flip-flops

QC QB QA count sequence:

$$000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100 \rightarrow 101 \rightarrow 110 \rightarrow 111 \rightarrow 000$$

Mod-8, recycling counter

QA toggles due to QAnot fed into D input when clocked.

XOR gate causes QB or QC flip-flops to toggle by producing opposite state for FF input.

Logic expressions:

$$D_{\text{QA}} = \text{QAnot}$$

$$D_{\text{QB}} = \text{QB} \oplus \text{QA} = \text{QB QAnot} + \text{QBnot QA}$$

$$D_{\text{QC}} = \text{QC} \oplus (\text{QB QA}) = \text{QC QBnot} + \text{QC QAnot} + \text{QCnot QB QA}$$

For mod-16, add 4th D flip-flop with D input generated by XORing QD output with ANDing of QC QB QA.

Unit 8 Timing & Waveshaping Circuits

8.1 Schmitt trigger waveshaper

Rounded output waveform for 7404

Square output waveform for 74LS14; triggers at approximately 1.6 V and 0.8 V on input waveform.

8.2 Pulse stretcher

$$t_p = 0.693 RC \quad \text{example solution:}$$

$$t_p = 0.47 \text{ s. using } C = 10 \mu\text{f} \text{ & } R = 68 \text{ k}\Omega$$

8.3 Variable frequency clock

$$\text{duty cycle} = (R_A + R_B)/(R_A + 2R_B) \approx 50\% \text{ if } R_A \ll R_B \text{ with } R_A \geq 1 \text{ k}\Omega$$

$$\text{frequency} = 1/(R_A + 2R_B)C$$

example solution:

$$\text{duty cycle} = 50.4\% \text{ with } R_A = 1 \text{ k}\Omega \text{ & } R_B = 68 \text{ k}\Omega$$

capacitor	frequency
10 μf	1.05 Hz
0.01 μf	1.05 kHz
0.001 μf	10.53 kHz

8.4 Waveform generator

SIGNAL1: $T = 1 \text{ ms.}$ & duty cycle = 80% \rightarrow freq. = 1 kHz

on 555: using $R_A = 82 \text{ k}\Omega$, $R_B = 27 \text{ k}\Omega$, $C = 0.01 \mu\text{f}$:
 $T = 0.94 \text{ ms}$ & duty cycle = 80.1%

on 74221: $1B = 2A = \text{SIGNAL1}$
 $1A = 0 \text{ & } 2B = 1 \text{ & } 1CLR = 2CLR = 1$
 $t_p = 0.693 RC$

SIGNAL2: $t_p = 0.3 \text{ ms.}$ \rightarrow use $1Q$ output
using $C = 0.01 \mu\text{f} \rightarrow R = 43.3 \text{ k}\Omega$ \rightarrow try $47 \text{ k}\Omega$

SIGNAL3: $t_p = 0.1 \text{ ms}$ \rightarrow use $\overline{2Q}$ output
using $C = 0.0047 \mu\text{f}$ $\rightarrow R = 30.7 \text{ k}\Omega \rightarrow$ try $33 \text{ k}\Omega$

8.5 Variable duty-cycle square wave generator

$$\begin{aligned} T = t_L + t_H &= 0.75(R_A+R_B)C & \therefore f = 1/[0.75(R_A+R_B)C] \\ \text{for } f = 10 \text{ kHz, try } 0.01 \mu\text{f} & & \rightarrow (R_A+R_B) = 13.3 \text{ k}\Omega \\ \text{for D.C.} = 15\% = R_A/(R_A+R_B) &= 0.15 & \rightarrow R_A = 0.15 (13.3 \text{ k}\Omega) = 2.0 \text{ k}\Omega \\ \text{for D.C.} = 85\% = R_A/(R_A+R_B) &= 0.85 & \rightarrow R_A = 0.85 (13.3 \text{ k}\Omega) = 11.3 \text{ k}\Omega \\ \therefore \text{use } 2.0 \text{ k}\Omega + 10 \text{ k}\Omega \text{ pot} + 1.3 \text{ k}\Omega \text{ resistors} &= 13.3 \text{ k}\Omega = R_A+R_B & \end{aligned}$$

8.6 Delayed pulse

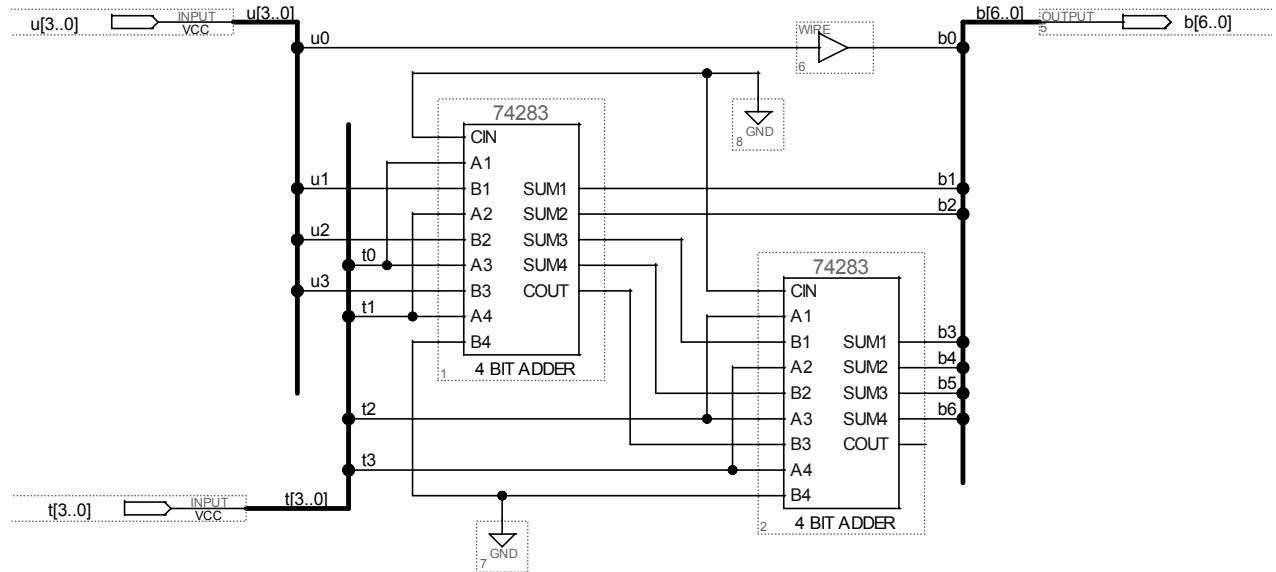
$$\begin{aligned} \text{use both one-shots in 74221: } 1A &= \text{trigger, } 2A = 1Q \\ t_w &= 0.22 \text{ ms} = 0.693R_2C_2 & \rightarrow \text{if } C_2 = 0.0047 \mu\text{f}, R_2 = 68 \text{ k}\Omega \\ t_d &= 0.33 \text{ ms} = 0.693R_1C_1 & \rightarrow \text{if } C_2 = 0.01 \mu\text{f}, R_1 = 47 \text{ k}\Omega \end{aligned}$$

8.7 Clock generator using one-shots

$$\begin{aligned} \text{on 74221: } 1A &= 2Q, 2A = 1Q \\ t_l &= 0.693R_1C = 0.693R_1(0.1 \mu\text{f}) & \rightarrow \text{for } 5 \text{ ms, } R_1 = 72 \text{ k}\Omega \\ &\text{try } R_1 = 72 \text{ k}\Omega & \rightarrow t_l = 5.68 \text{ ms} \\ t_2 &= 0.693R_2(0.1 \mu\text{f}) & \rightarrow \text{for } 10 \text{ ms}-5.68 \text{ ms} = 4.32 \text{ ms} \\ R_2 &= 62 \text{ k}\Omega & \end{aligned}$$

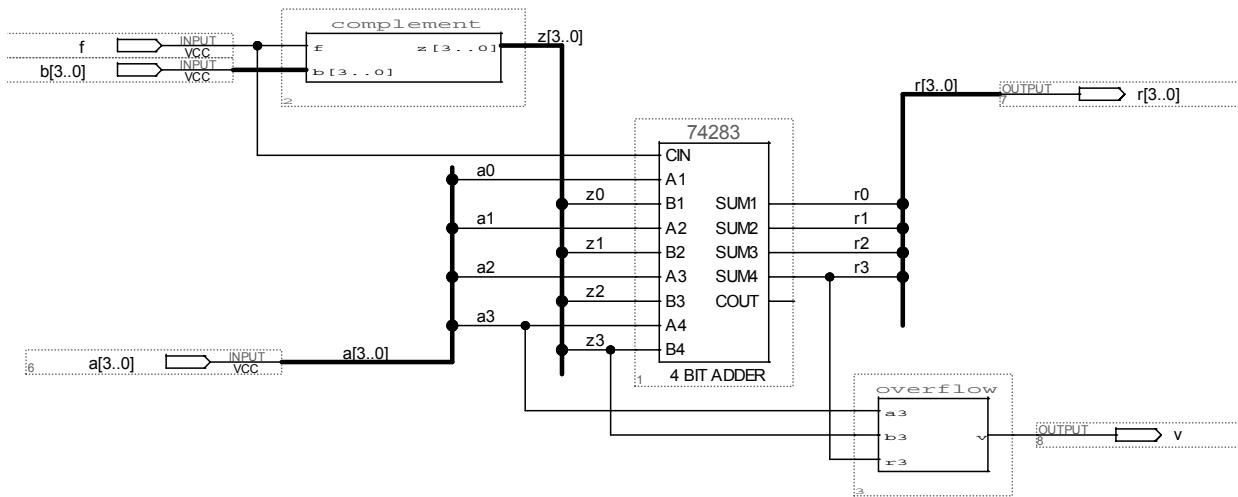
Unit 9 Arithmetic Circuits

- 9.1 Eight-bit adder see Unit 9 in Lab Manual
9.2 Incrementer/decrementer circuit see Unit 9 in Lab Manual
9.3 BCD to binary converter bcd2bin_2digit\bcd2bin.gdf



9.4 Four-bit adder/subtractor

add_sub.gdf



```
SUBDESIGN complement
(
    f, b[3..0]           :INPUT;
    z [3..0]             :OUTPUT;
)
BEGIN
    IF f THEN z [] = !b [];
    ELSE          z [] = b [];
END IF;
END;
```

```
SUBDESIGN overflow
(
    a3, b3, r3           :INPUT;
    v                   :OUTPUT;
)
BEGIN
    DEFAULTS
        v = GND;      -- no overflow
    END DEFAULTS;
    TABLE                 -- only 2 overflow cases
        a3, b3, r3     => v;
        0, 0, 1         => 1;
        1, 1, 0         => 1;
    END TABLE;
END;
```

```

ENTITY complement IS
PORT (
    f          : IN BIT;
    b          : IN BIT_VECTOR (3 DOWNTO 0);
    z          : OUT BIT_VECTOR (3 DOWNTO 0)
);
END complement;

ARCHITECTURE vhdl OF complement IS
BEGIN
    PROCESS (f, b)
    BEGIN
        IF  f = '1'  THEN  z <= NOT b;    -- complement
        ELSE                  z <= b;           -- no complement
        END IF;
    END PROCESS;
END vhdl;

```

```

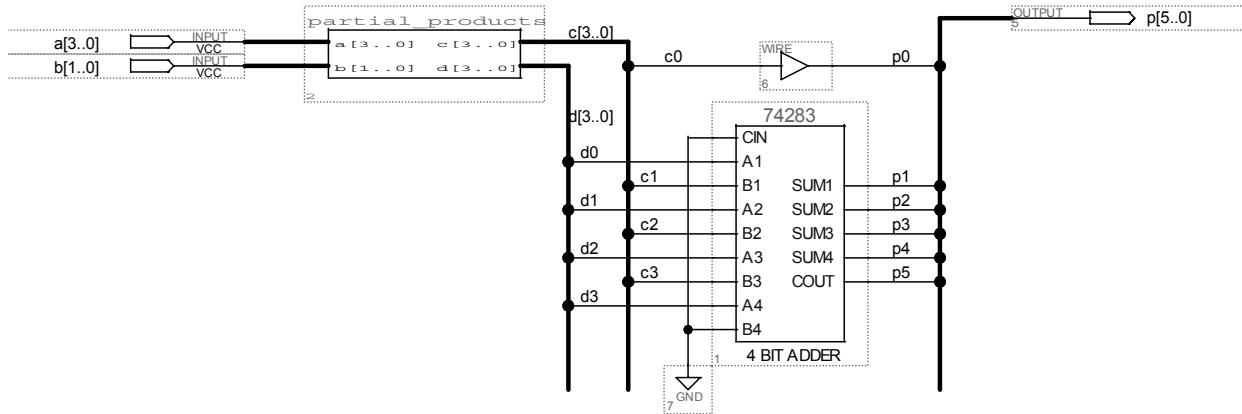
ENTITY overflow IS
PORT (
    a3, b3, r3          : IN BIT;
    v                   : OUT BIT
);
END overflow;

ARCHITECTURE vhdl OF overflow IS
BEGIN
    PROCESS (a3, b3, r3)
    BEGIN      -- detects 2 overflow cases
        IF          a3 = '0' AND b3 = '0' AND r3 = '1'
        THEN        v <= '1';
        ELSIF       a3 = '1' AND b3 = '1' AND r3 = '0'
        THEN        v <= '1';
        ELSE        v <= '0';           -- no overflow
        END IF;
    END PROCESS;
END vhdl;

```

9.5 Binary multiplier

bin_mult.gdf



```
SUBDESIGN partial_products
(
    a[3..0], b[1..0] :INPUT;
    c[3..0], d[3..0] :OUTPUT;
)
BEGIN
    c0 = a0 & b0;          -- 1st partial product
    c1 = a1 & b0;          -- "
    c2 = a2 & b0;          -- "
    c3 = a3 & b0;          -- "
    d0 = a0 & b1;          -- 2nd partial product
    d1 = a1 & b1;          -- "
    d2 = a2 & b1;          -- "
    d3 = a3 & b1;          -- "
END;
```

```

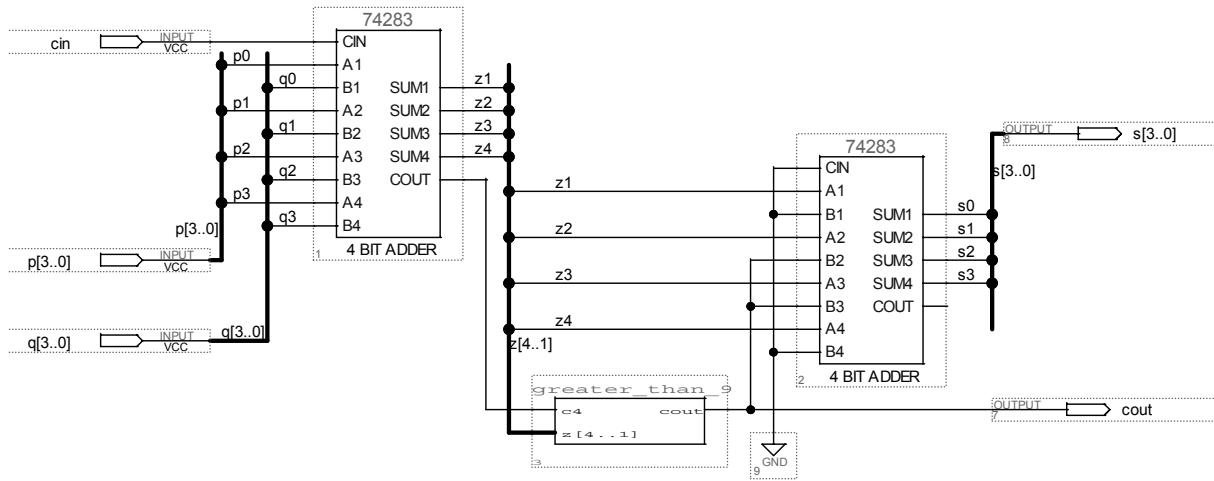
ENTITY partial_products IS
PORT (
    a      : IN BIT_VECTOR (3 DOWNTO 0);
    b      : IN BIT_VECTOR (1 DOWNTO 0);
    c      : OUT BIT_VECTOR (3 DOWNTO 0);
    d      : OUT BIT_VECTOR (3 DOWNTO 0)
);
END partial_products;

ARCHITECTURE vhdl OF partial_products IS
BEGIN
    c(0) <= a(0) AND b(0);      -- 1st partial product
    c(1) <= a(1) AND b(0);      -- "
    c(2) <= a(2) AND b(0);      -- "
    c(3) <= a(3) AND b(0);      -- "
    d(0) <= a(0) AND b(1);      -- 2nd partial product
    d(1) <= a(1) AND b(1);      -- "
    d(2) <= a(2) AND b(1);      -- "
    d(3) <= a(3) AND b(1);      -- "
END vhdl;

```

9.6 BCD adder

bcdadder.gdf



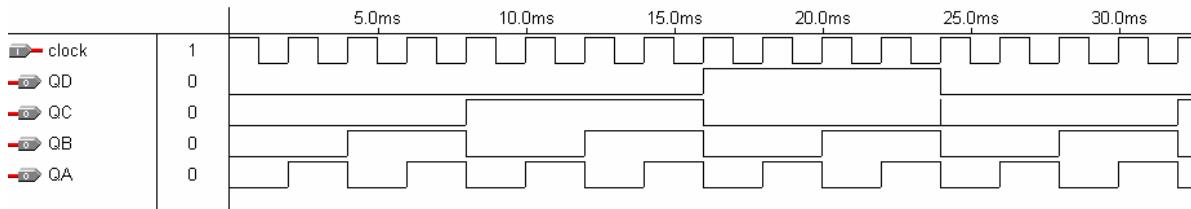
```
SUBDESIGN greater_than_9
(
    c4, z[4..1]      :INPUT;
    cout              :OUTPUT;
)
BEGIN      -- detect if binary sum >9
    IF c4 # z[] > B"1001" THEN cout = VCC;
    ELSE                      cout = GND;
    END IF;
END;
```

```
ENTITY greater_than_9 IS
PORT (
    c4          : IN BIT;
    z           : IN BIT_VECTOR (4 DOWNTO 1);
    cout        : OUT BIT
);
END greater_than_9;
ARCHITECTURE vhdl OF greater_than_9 IS
BEGIN
    PROCESS (c4, z)
    BEGIN      -- detect if binary sum >9
        IF c4 = '1' OR z > "1001" THEN cout <= '1';
        ELSE                           cout <= '0';
        END IF;
    END PROCESS;
END vhdl;
```

Unit 10 Analyzing and Testing Synchronous Counters

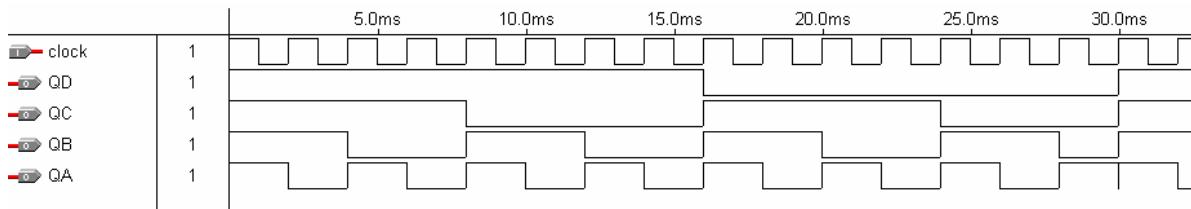
10.1 Counter 1

Mod-12 up counter (counts 0 to 11 with asynchronous clear on 12)



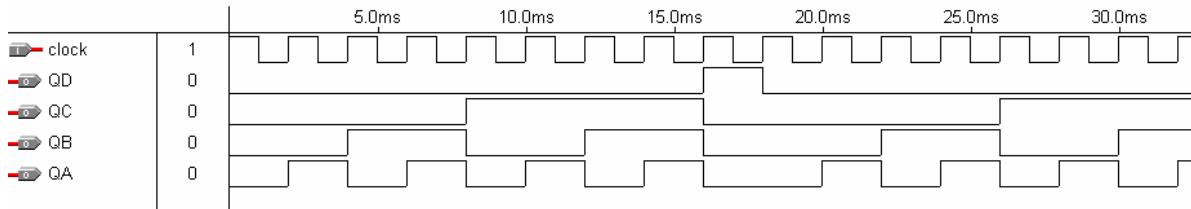
10.2 Counter 2

Mod-15 down counter (counts 15 down to 1 with asynchronous preset on 0)



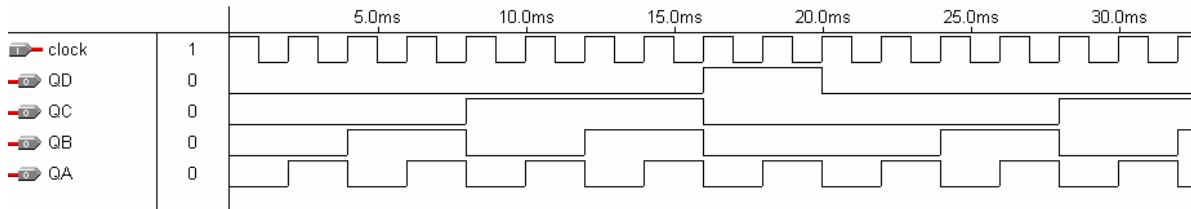
10.3 Counter 3

Mod-9 up counter (counts 0 to 8)



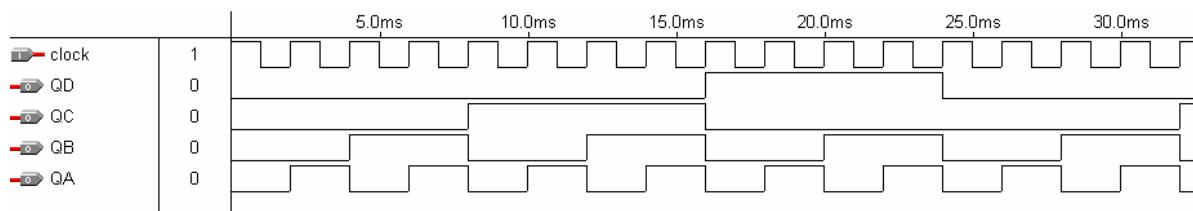
10.4 Counter 4

Mod-10 up counter (counts 0 to 9)



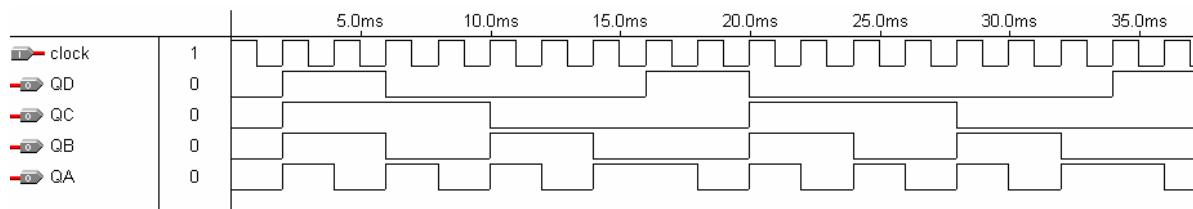
10.5 Counter 5

Mod-12 up counter (counts 0 to 11)



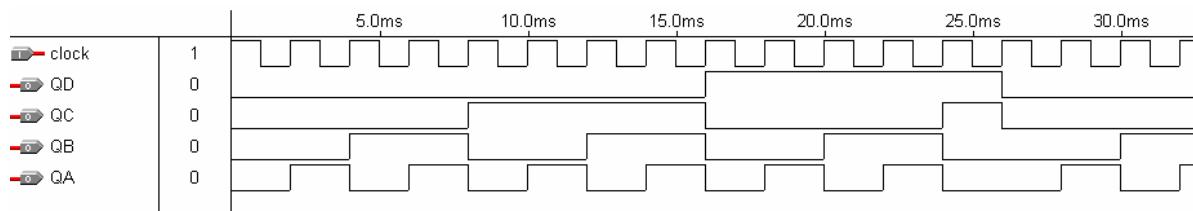
10.6 Counter 6

Mod-9 down counter (counts 9 down to 1 after start-up sequence if reset on power-up)



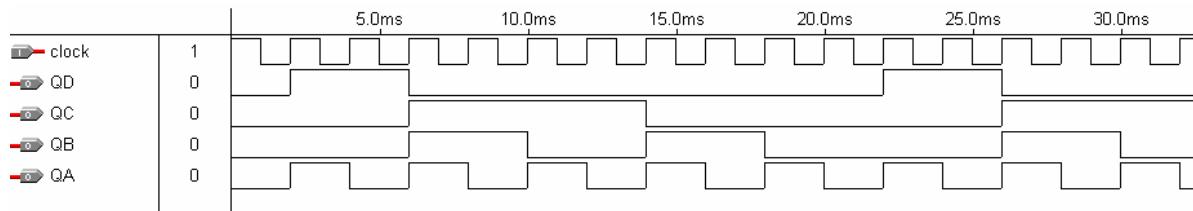
10.7 Counter 7

Mod-13 up counter (counts 0 to 12)



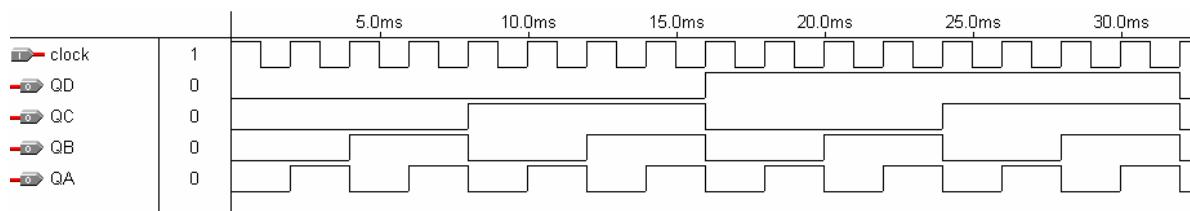
10.8 Counter 8

Mod-10 down counter (counts 9 to 0)



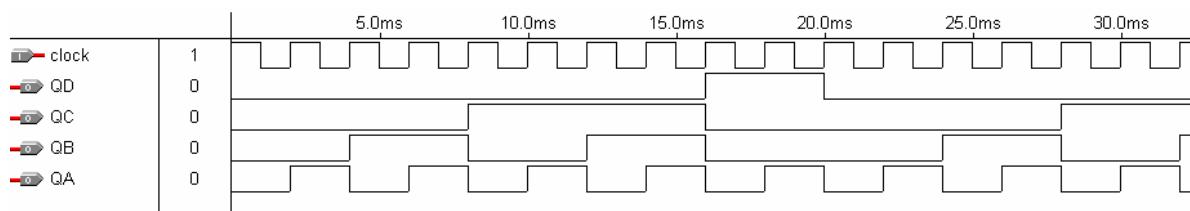
10.9 Counter 9

Mod-16 up counter (counts 0 to 15)



10.10 Counter 10

Mod-10 up counter (counts 0 to 9) – illustrates circuit implementation used in PLDs



Unit 11 Testing Register and Counter Functions

11.1 Latch and register operation

QA is the LSB & QD is the MSB on the 74163.

74163 is a recycling, mod-16, binary counter with sequence 0 thru 15.

74175 register is positive-edge triggered.

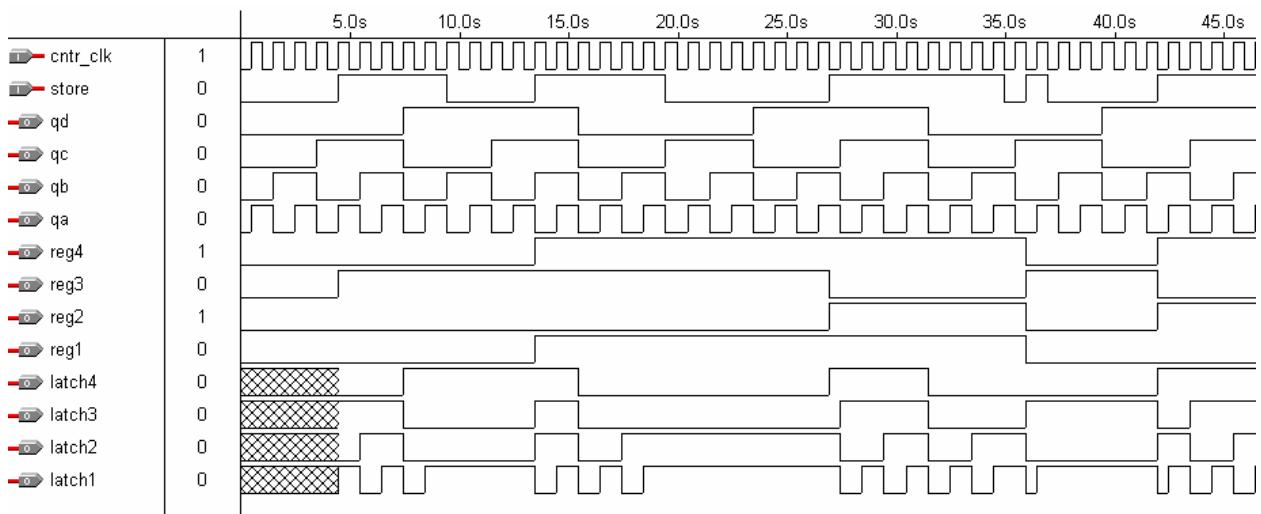
Reg4 (4Q) is the MSB for 74175 since its input is from 74163 QD.

74375 latch is high level enabled.

74375 is called a transparent latch because data flows thru when enabled.

Latch4 (4Q) is the MSB for 74375 since its input is from 74163 QD.

74175 register only stores new data on rising edge of clock (store) while 74375 stores data input entire time that enable (store) is high.



11.2 Binary counter signal frequencies

signal	Clock	QA	QB	QC	QD	RCO
frequency	64kHz	32kHz	16kHz	8kHz	4kHz	4kHz

$$\text{freq}_{\text{RCO}} = \text{freq}_{\text{QD}} = \frac{1}{2} \text{ freq}_{\text{QC}} = \frac{1}{2} \text{ freq}_{\text{QB}} = \frac{1}{2} \text{ freq}_{\text{QA}} = \frac{1}{2} \text{ freq}_{\text{Clock}}$$

11.3 Modifying the count sequence

The original counter produces recycling sequence 0000 to 1111 for mod-16.

The modified counter produces recycling sequence 0000 to 1001 for mod-10 (decade counter).

NAND gate output goes low when counter reaches 1001 & clears counter on next clock.

CLR input is active-low.

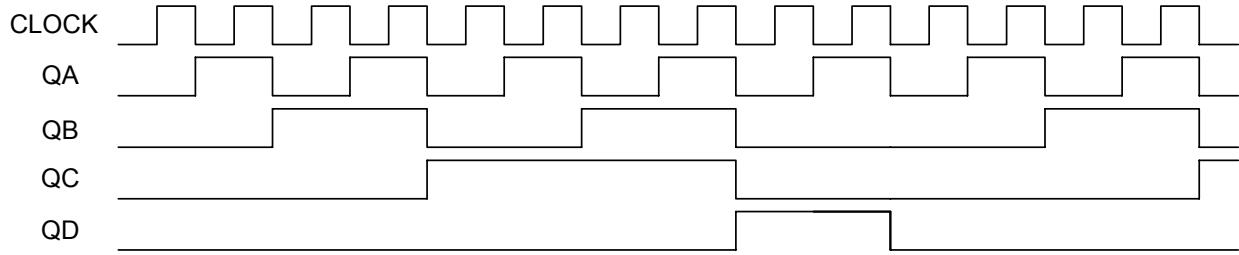
CLR input synchronously clears counter when active.

Counter clears on next clock pulse after reaching state 1001.

Change NAND inputs to QD & QC for mod-13.

Mod-13 recycling count sequence is 0000 thru 1100.

11.4 Decade counter timing diagram



With 10 kHz clock frequency, frequency of QD is 1 kHz.

Frequency of QD is 1/10 of frequency of clock.

Divide-by factor for the MSB output is the same as the modulus of the counter.

11.5 Decade counter operation

The 74160 is a mod-10 up-counter.

The RCO output detects the last state (1001) in the count sequence.

ENT & ENP are active-high count enables.

The CLR control is an active-low asynchronous clear.

The CLR has priority because if you try to count & clear at the same time, it will clear.

The LOAD control allows a BCD value to be loaded into the counter when it is clocked.

The LOAD is active-low and is synchronous.

The LOAD function has priority over the count because if you try to do both at the same time, it will load.

The CLR has the highest priority of all.

11.6 Self-stopping counter

The counter will clear to 0000 when the pulser applies a low.

When pulser is high, the counter counts up to 1001 and stops.

Counter is called "self-stopping" since circuit will automatically stop when it reaches its terminal state.

The RCO will detect the last decade state (1001) and output a high. After NOT gate, this will make ENP = 0 and disable (stop) the counter.

The counter is "restarted" by taking the pulser low momentarily.

11.7 Mod-9 counters

Circuit 1 recycling count sequence is 1 through 9 for a modulus of 9.

RCO goes high at state 1001, the terminal state for a decade counter.

The LOAD function is synchronous since counter must wait until next clock after RCO goes high.

Circuit 2 recycling count sequence is 0 through 8 for a modulus of 9.

The CLR function is asynchronous since counter clears as soon as reaches state 1001.

RCO only momentarily goes high – too fast to observe on lamp.

State 1001 is cleared right away and cannot be observed on lamp.

State 1001 is a transient state.

Both circuits produce count sequences that have 9 states.

11.8 Up/Down counter operation

CTEN is active-low count enable control.

74190 counts up when D/U = 0 & counts down when D/U = 1.

The output MAX/MIN will detect the decade counter's "terminal" state which will be 9 if D/U = 0 or 0 if D/U = 1. MAX/MIN is controlled by D/U.

LOAD is active-low, asynchronous parallel data load control.

LOAD has priority over counting – if both CTEN & LOAD are active, counter will load data.

11.9 Mod-N counter

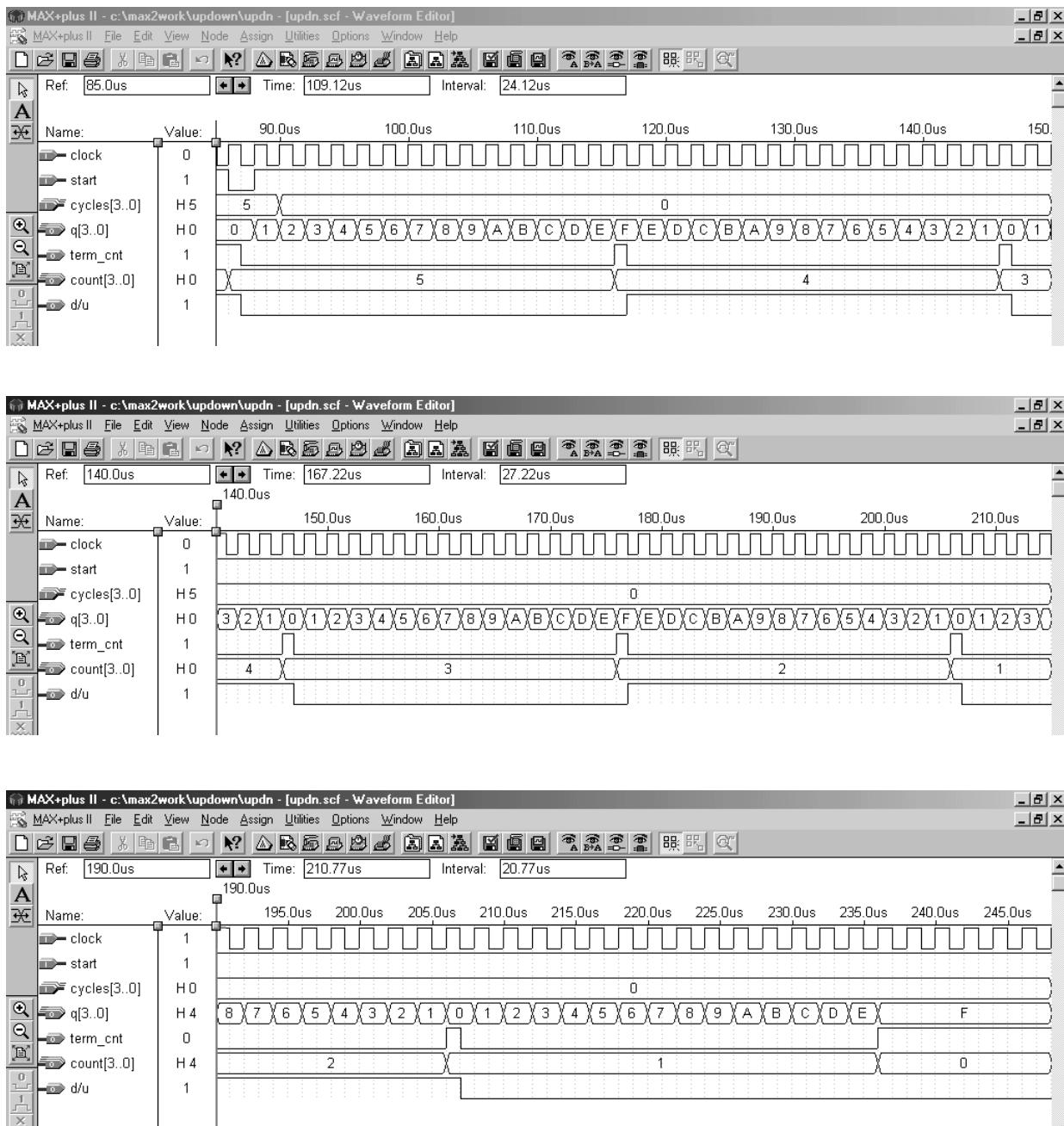
The counter will count down from BCD switch input value to 0001 & then load value set on switches. Terminal counter state (0000) will be a transient state. Number on switches will be modulus of counter.

If switch set to 0000 or 0001, counter will be "stuck" at that state.

Unit 12 Counter Applications Using Macrofunctions

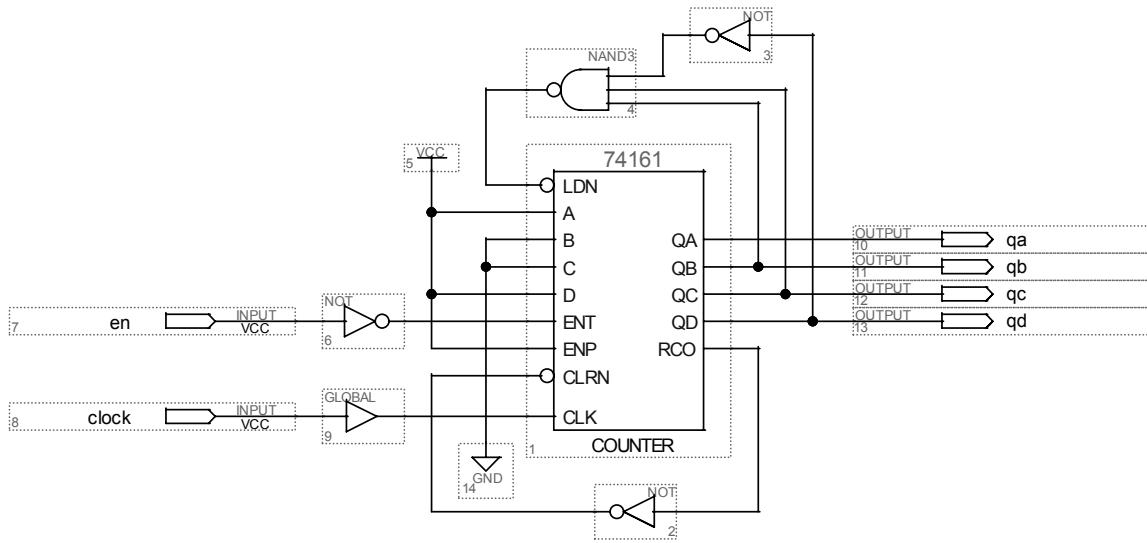
12.1 Counter circuit upgrade

updown.gdf



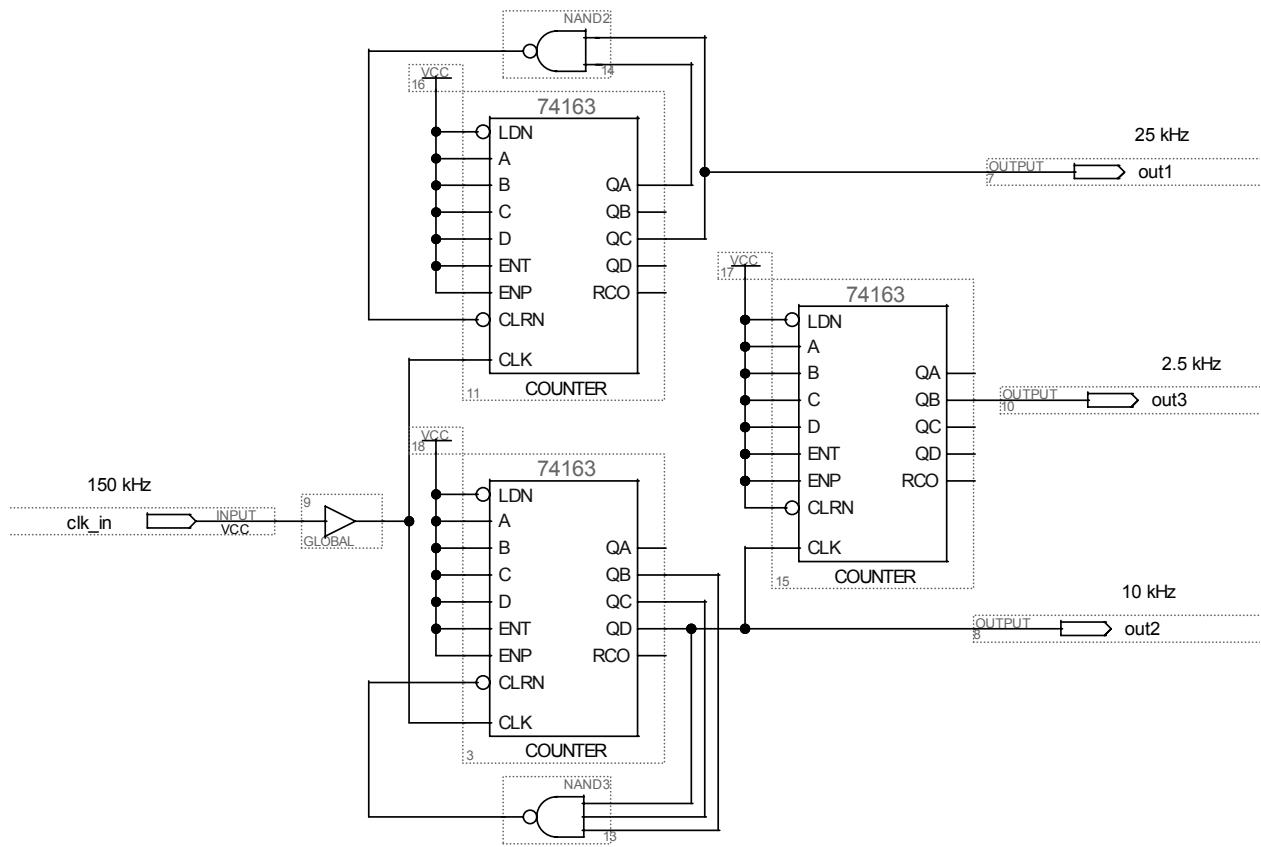
12.2 Mod-13 count sequence

mod13.gdf



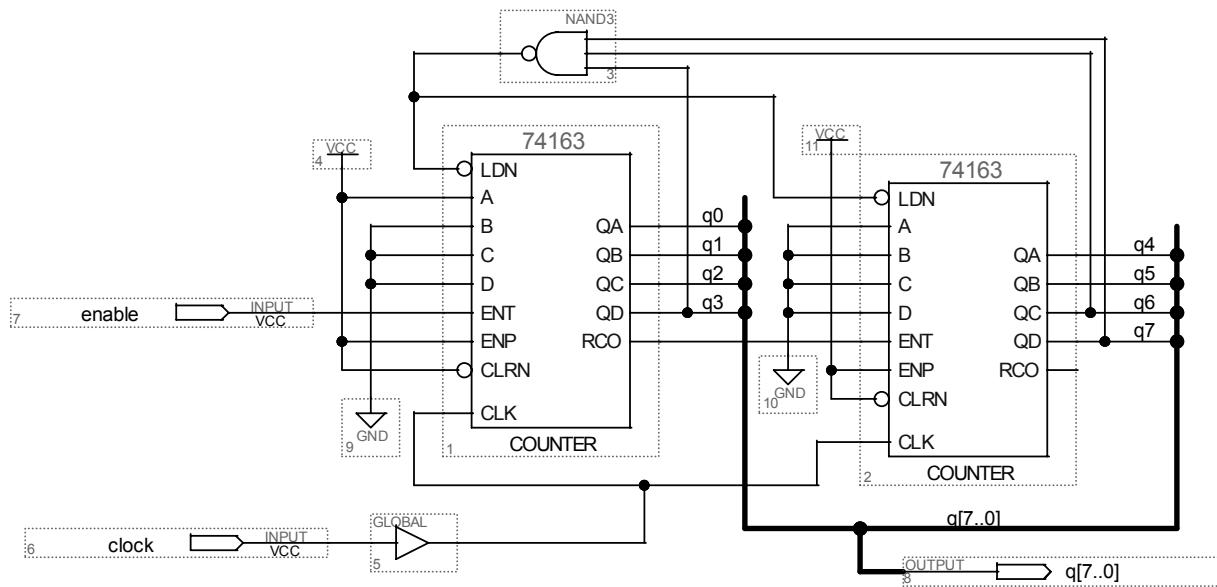
12.3 Frequency divider

freqdiv12-3.gdf



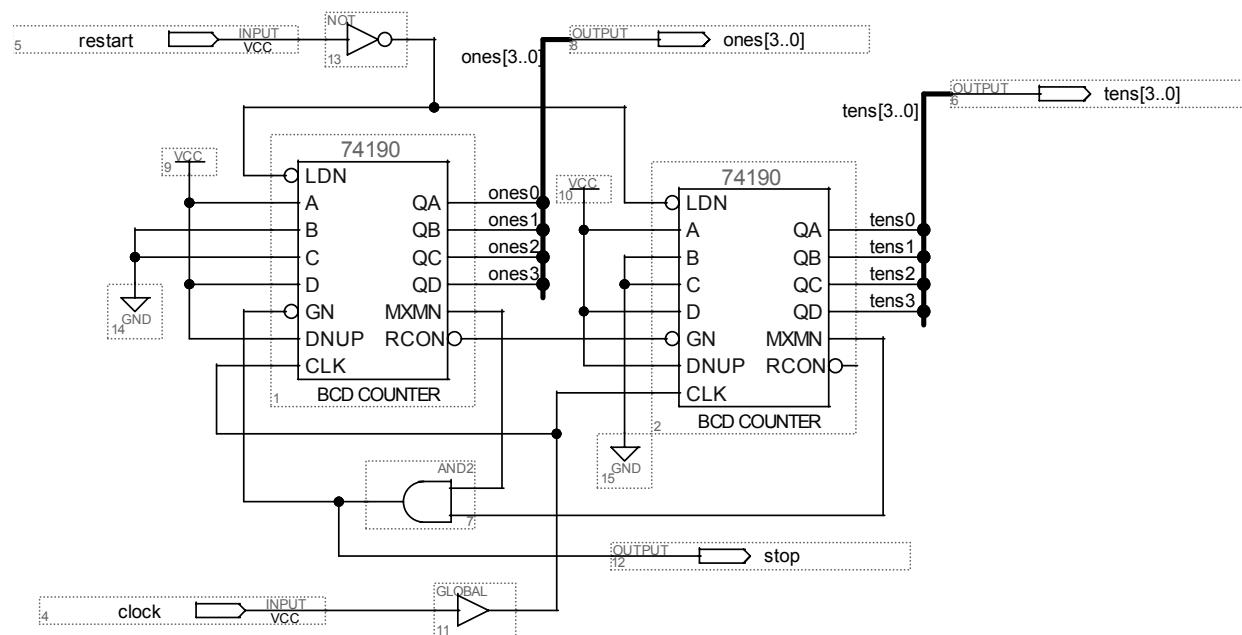
12.4 Mod-200 binary counter

mod200.gdf



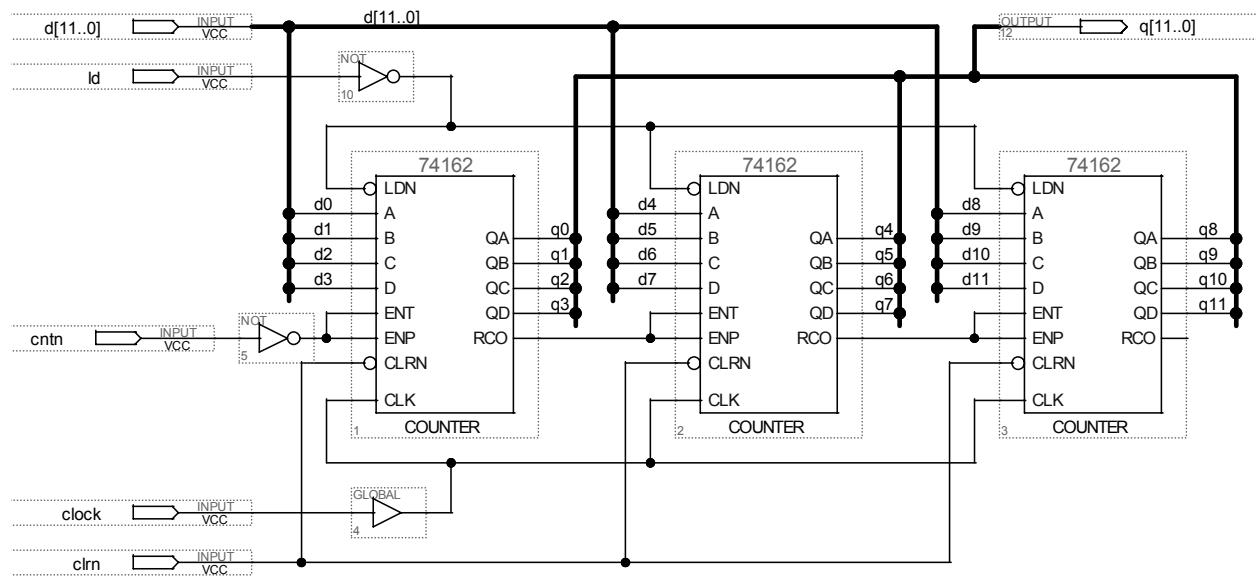
12.5 Mod-100, self-stopping, BCD down-counter

mod100BCDdn.gdf



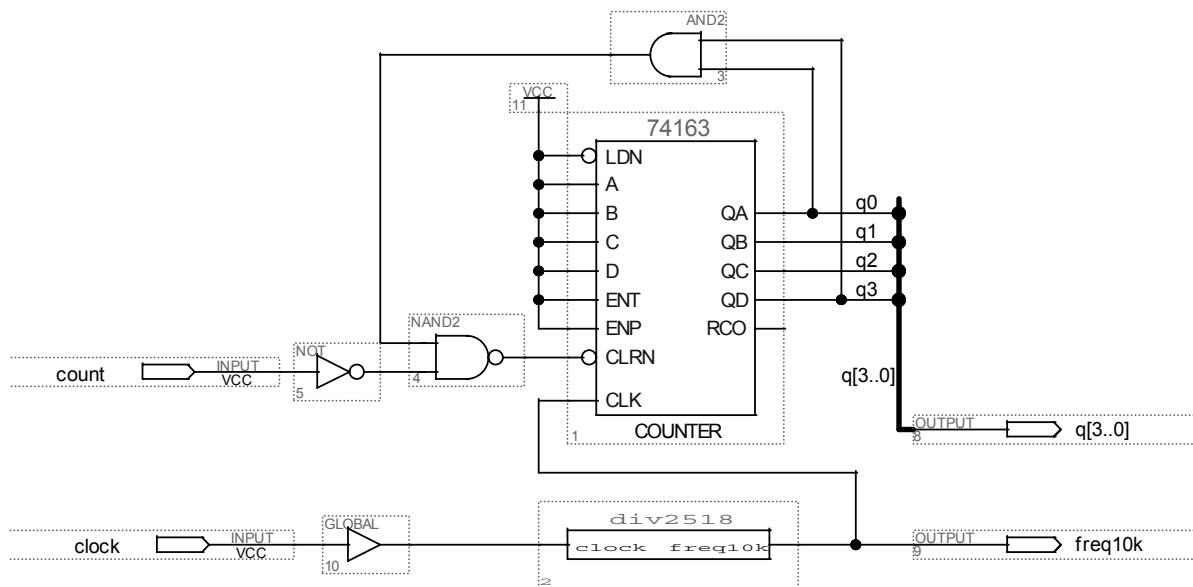
12.6 Mod-1000 BCD counter

mod1000.gdf

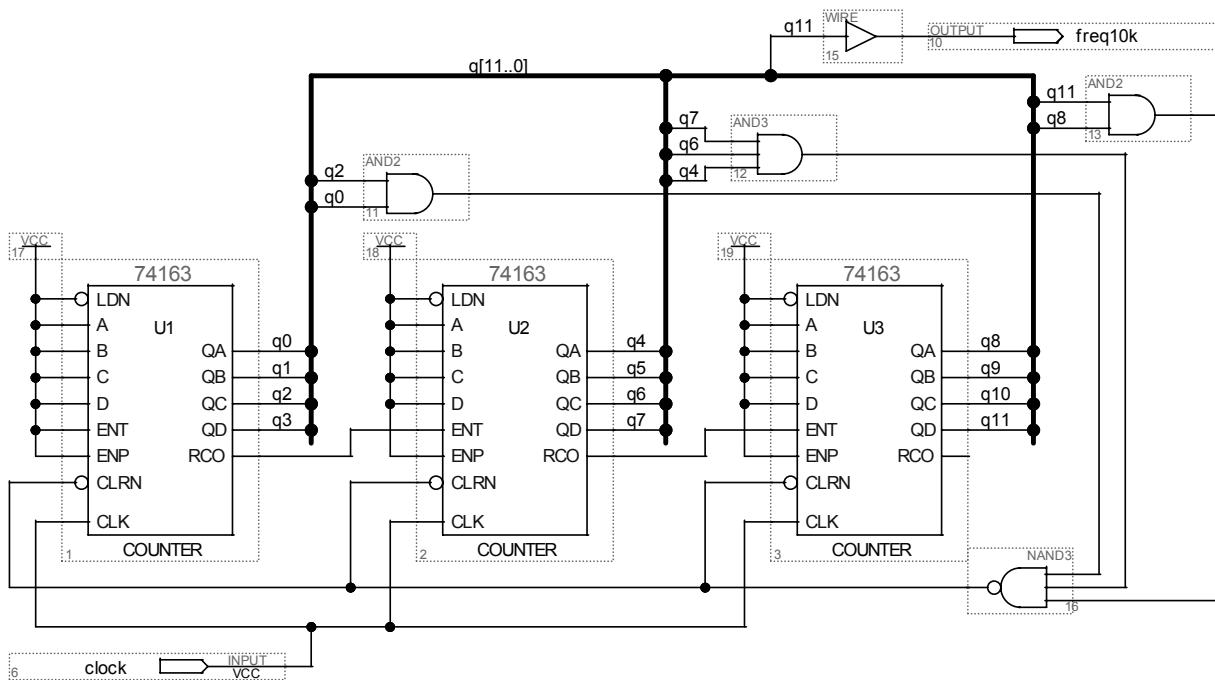


12.7 On-board clock divider and switchable counter circuit
 (assuming 25.175 MHz clock on UP2)

switchcount.gdf

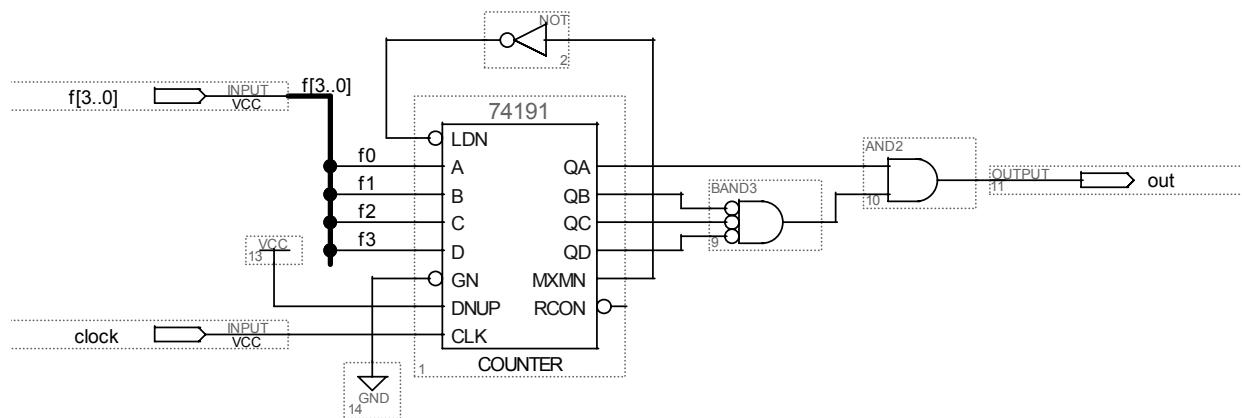


div2518.gdf



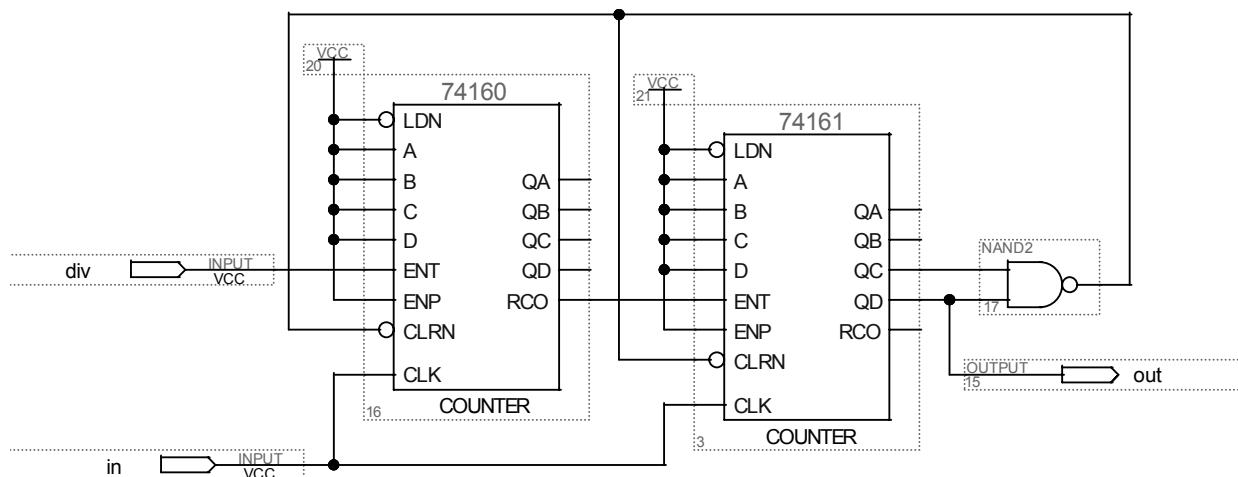
12.8 Programmable frequency divider

progdiv\progdiv.gdf

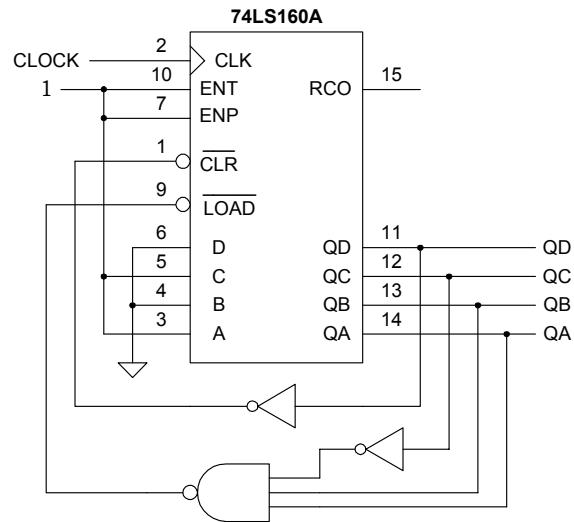


12.9 Cascaded frequency divider

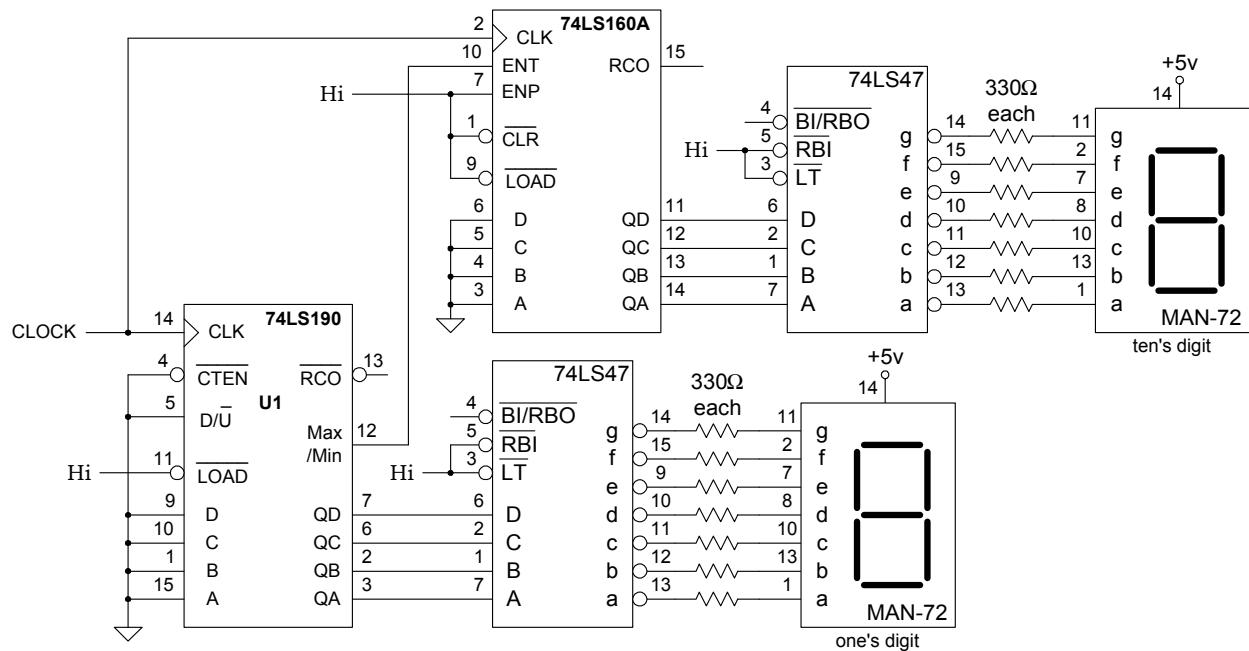
divider\divider.gdf



12.10 Mod-7 count sequence

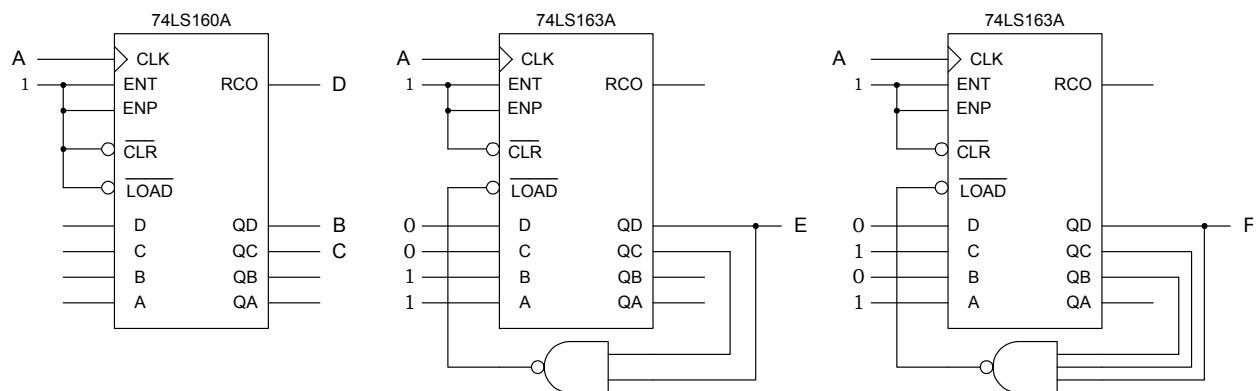


12.11 Mod-100 BCD counter and display

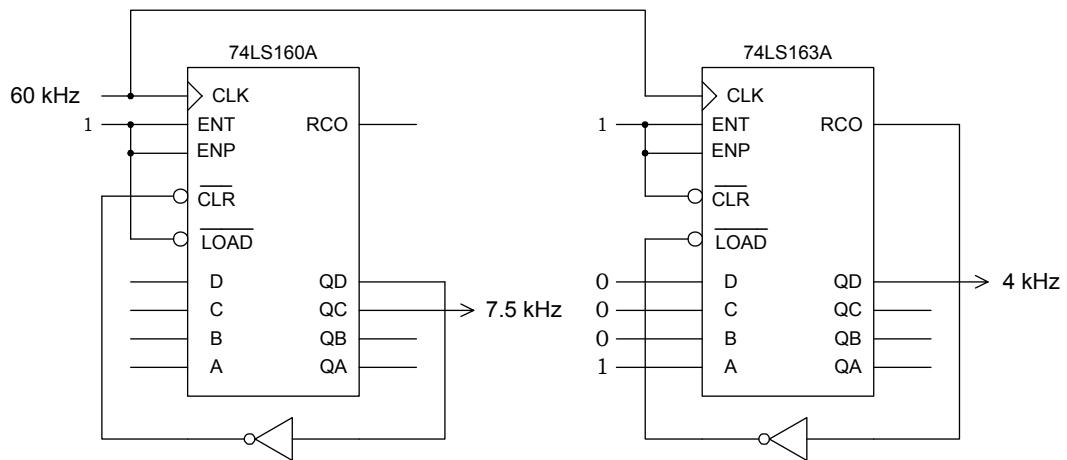


12.12 Waveform generator circuits

Signal	Division factor	Duty cycle	Frequency
A			100kHz
B	$\div 10$	20%	10kHz
C	$\div 10$	40%	10kHz
D	$\div 10$	10%	10kHz
E	$\div 10$	50%	10kHz
F	$\div 10$	70%	10kHz

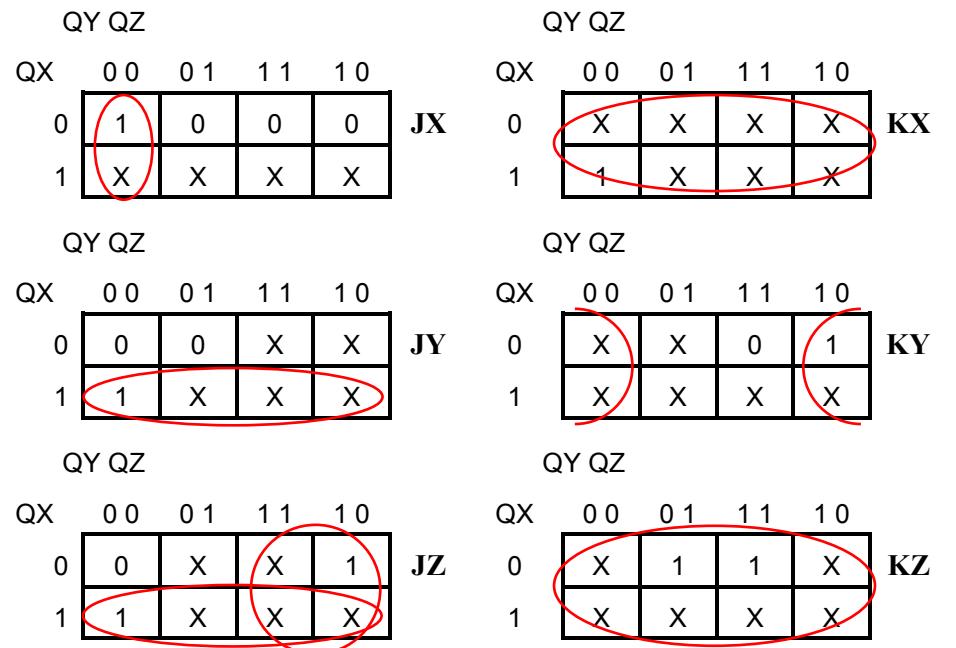


12.13 Frequency divider circuit



13.2 Mod-5 synchronous, down counter design

Present States			Next States			Flip-flop Inputs					
QX_n	QY_n	QZ_n	QX_{n+1}	QY_{n+1}	QZ_{n+1}	JX	KX	JY	KY	JZ	KZ
0	0	0	1	0	0	1	X	0	X	0	X
0	0	1	0	0	0	0	X	0	X	X	1
0	1	0	0	0	1	0	X	X	1	1	X
0	1	1	0	1	0	0	X	X	0	X	1
1	0	0	0	1	1	X	1	1	X	1	X
1	0	1	X	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X



$$JX = \overline{QY} \ \overline{QZ}$$

$$KX = 1$$

$$JY = QX$$

$$KY = \overline{QZ}$$

$$JZ = QY + QX$$

$$KZ = 1$$

QY QZ

QX	00	01	11	10
0	1	0	0	0
1	0	X	X	X

QY QZ

QX	00	01	11	10
0	0	0	1	0
1	1	X	X	X

QY QZ

QX	00	01	11	10
0	0	0	0	1
1	1	X	X	X

$$D_{QX} = \overline{QX} \ \overline{QY} \ \overline{QZ}$$

$$D_{QY} = QX + QY \ QZ$$

$$D_{QZ} = QX + QY \ \overline{QZ}$$

QB	QA
QD	QC
0 0	0 1
0 0	0 0
X	X
X	X
1 0	1 0

JC = $QD \overline{QA}$

QB	QA
QD	QC
0 0	0 1
0 0	1 0
1	0
X	X
1 0	X

KC = $\overline{QB} \overline{QA}$

QB	QA
QD	QC
0 0	0 1
0 0	1 1
1	X
X	X
1 0	X

JB = $QC \overline{QA} + QD \overline{QA}$

QB	QA
QD	QC
0 0	0 1
0 0	1 1
X	X
1	0
1 0	X

KB = \overline{QA}

QB	QA
QD	QC
0 0	0 1
0 0	1 1
X	X
X	X
1 0	X

JA = 1

QB	QA
QD	QC
0 0	0 1
0 0	1 1
X	X
1	X
1 0	X

KA = 1

QB	QA
QD	QC
0 0	0 1
0 0	1 0
0	0
0	1
1	X
X	X
1 0	X

DQC = $QC QB + QC QA + QD \overline{QA}$

QB	QA
QD	QC
0 0	0 1
0 0	1 1
1	X
X	X
1 0	X

DQA = \overline{QA}

QB	QA				
QD	QC	0 0	0 1	1 1	1 0
0 0		0	0	0	0
0 1		0	0	1	0
1 1		X	X	X	X
1 0		1	1	X	0

$$DQD = QD \overline{QB} + QC QB QA$$

QB	QA				
QD	QC	0 0	0 1	1 1	1 0
0 0		0	1	0	1
0 1		0	1	0	1
1 1		X	X	X	X
1 0		0	1	X	0

$$DQB = \overline{QB} QA + \overline{QD} QB \overline{QA}$$

QB	QA				
QD	QC	0 0	0 1	1 1	1 0
0 0		0	0	1	0
0 1		1	1	0	1
1 1		X	X	X	X
1 0		0	0	X	0

$$DQC = QC \overline{QB} + QC \overline{QA} + \overline{QC} QB QA$$

QB	QA				
QD	QC	0 0	0 1	1 1	1 0
0 0		1	0	0	1
0 1		1	0	0	1
1 1		X	X	X	X
1 0		1	0	X	0

$$DQA = \overline{QB} \overline{QA} + \overline{QD} \overline{QA}$$

Unit 14A Sequential Circuit Design with Altera Hardware Description Language

14A.1 Gray code counter

```
SUBDESIGN gray
(
    clock, dir           :INPUT;
    q[3..0], index       :OUTPUT;
)

VARIABLE
    gray: MACHINE OF BITS (q[3..0])      -- state machine
        WITH STATES (                  -- gray code bit patterns
            s0  = B"0000",
            s1  = B"0001",
            s2  = B"0011",
            s3  = B"0010",
            s4  = B"0110",
            s5  = B"0111",
            s6  = B"0101",
            s7  = B"0100",
            s8  = B"1100",
            s9  = B"1101",
            s10 = B"1111",
            s11 = B"1110",
            s12 = B"1010",
            s13 = B"1011",
            s14 = B"1001",
            s15 = B"1000");
);

BEGIN
    gray.clk = clock;          -- state machine clock

    TABLE          -- present state/next state table
        gray,     dir      =>      gray,      index;
        s0,       0         =>      s1,       0;
        s0,       1         =>      s15,      0;
        s1,       0         =>      s2,       1;
        s1,       1         =>      s0,       1;
        s2,       0         =>      s3,       1;
        s2,       1         =>      s1,       1;
        s3,       0         =>      s4,       1;
        s3,       1         =>      s2,       1;
        s4,       0         =>      s5,       1;
        s4,       1         =>      s3,       1;
        s5,       0         =>      s6,       1;
        s5,       1         =>      s4,       1;
        s6,       0         =>      s7,       1;
        s6,       1         =>      s5,       1;
```

s7,	0	=>	s8,	1;
s7,	1	=>	s6,	1;
s8,	0	=>	s9,	1;
s8,	1	=>	s7,	1;
s9,	0	=>	s10,	1;
s9,	1	=>	s8,	1;
s10,	0	=>	s11,	1;
s10,	1	=>	s9,	1;
s11,	0	=>	s12,	1;
s11,	1	=>	s10,	1;
s12,	0	=>	s13,	1;
s12,	1	=>	s11,	1;
s13,	0	=>	s14,	1;
s13,	1	=>	s12,	1;
s14,	0	=>	s15,	1;
s14,	1	=>	s13,	1;
s15,	0	=>	s0,	1;
s15,	1	=>	s14,	1;

END TABLE;

END;

14A.2 Up/down BCD counter

```

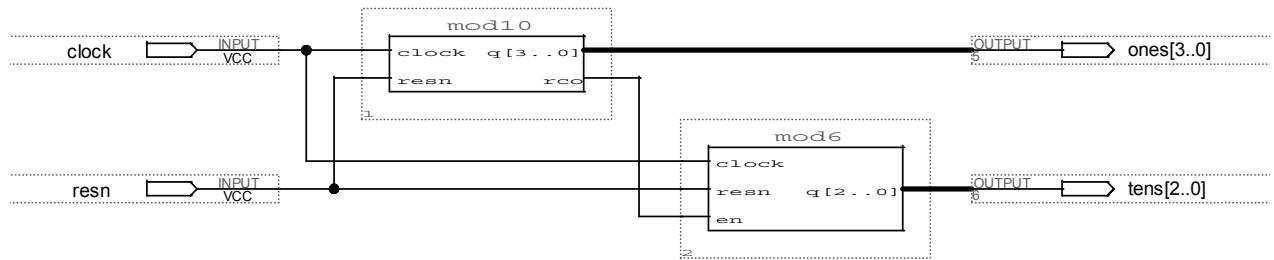
SUBDESIGN updn_mod10
(
    clock, c[1..0]           :INPUT;
    count[3..0], carryn     :OUTPUT;
)
VARIABLE
    count[3..0]             :DFF;      -- counter FFs
BEGIN
    DEFAULTS
        carryn = VCC;       -- inactive output
    END DEFAULTS;

    count[].clk = clock;    -- clock connection

    CASE c[] IS            -- determine control
        WHEN H"0" =>        -- reset
            count[].d = 0;
        WHEN H"1" =>        -- count down
            IF      count[] == 0 THEN -- at terminal state?
                count[].d = 9; carryn = GND;
            ELSE    count[].d = count[].q - 1;      -- decr.
            END IF;
        WHEN H"2" =>        -- count up
            IF      count[] == 9 THEN -- at terminal state?
                count[].d = 0; carryn = GND;
            ELSE    count[].d = count[].q + 1;      -- incr.
            END IF;
        WHEN H"3" =>        -- hold count
            count[].d = count[].q;
    END CASE;
END;

```

14A.3 Mod-60 BCD counter mod60.gdf



```

SUBDESIGN mod10
(
    clock, resn           :INPUT;
    q[3..0], rco          :OUTPUT;
)
VARIABLE
    q[3..0]                :DFF;      -- 4 FFs
BEGIN
    DEFAULTS
        rco = GND;          -- inactive output
    END DEFAULTS;

    q[].clk = clock;

    IF !resn THEN          -- active-low reset
        q[].d = 0;
    ELSIF q[].q == 9 THEN   -- at terminal state?
        q[].d = B"0000";     -- recycle
        rco = VCC;           -- assert rco
    ELSE
        q[].d = q[].q + 1;   -- increment count
    END IF;
END;

```

```

SUBDESIGN mod6
(
    clock, resn, en       :INPUT;
    q[2..0]                :OUTPUT;
)
VARIABLE
    q[2..0]                :DFF;      -- 3 FFs
BEGIN
    q[].clk = clock;

    IF !resn THEN          -- active-low reset
        q[].d = 0;
    ELSIF en THEN          -- increment tens digit
        IF q[].q == 5 THEN   -- at terminal state?
            q[].d = B"000";   -- recycle
        ELSE
            q[].d = q[].q + 1;  -- increment count
        END IF;
    ELSE
        q[].d = q[].q;        -- hold current state
    END IF;
END;

```

14A.4 Mod-100 binary counter

```
SUBDESIGN mod100
(
    clock, enable, w      :INPUT;
    q[7..0], pulse        :OUTPUT;
)
VARIABLE
    q[7..0]              :DFF;          -- 8 FFs
BEGIN
    q[].clk = !clock;           -- clock on NGT

    IF enable THEN           -- count enabled?
        IF q[].q == 99 THEN  -- at terminal state?
            q[].d = 0;       -- recycle
        ELSE
            q[].d = q[].q + 1; -- increment count
        END IF;
    ELSE
        q[].d = q[].q;       -- hold count
    END IF;

    -- control pulse width of output
    IF !w & q[].q >= 95 THEN pulse = VCC;
    ELSIF w & q[].q >= 90 THEN pulse = VCC;
    ELSE pulse = GND;
    END IF;
END;
```

14A.5 Stepper motor sequence controller

```

SUBDESIGN 'half-step'
(
    step, cw, go           :INPUT;
    q[3..0]                 :OUTPUT;
)

VARIABLE
    stepper: MACHINE OF BITS (q[3..0]) -- state machine
        WITH STATES ( -- define stepper states
            initial = B"0000",
            s1 = B"0101",
            s2 = B"0001",
            s3 = B"1001",
            s4 = B"1000",
            s5 = B"1010",
            s6 = B"0010",
            s7 = B"0110",
            s8 = B"0100");
BEGIN

    stepper.clk = step;      -- clock port
    stepper.ena = go;        -- active-high enable port

    TABLE          -- present state/next state table
        stepper,      cw       =>  stepper;
        initial,      X        =>  s1;
        s1,           1        =>  s2;
        s1,           0        =>  s8;
        s2,           1        =>  s3;
        s2,           0        =>  s1;
        s3,           1        =>  s4;
        s3,           0        =>  s2;
        s4,           1        =>  s5;
        s4,           0        =>  s3;
        s5,           1        =>  s6;
        s5,           0        =>  s4;
        s6,           1        =>  s7;
        s6,           0        =>  s5;
        s7,           1        =>  s8;
        s7,           0        =>  s6;
        s8,           1        =>  s1;
        s8,           0        =>  s7;
    END TABLE;
END;

```

14A.6 Variable frequency divider

```
SUBDESIGN  divider
(
    freq_in, m[1..0]           :INPUT;
    freq_out                   :OUTPUT;
)
VARIABLE
    count [3..0]                :DFF;
    div5, div10, div12, div15  :NODE;      -- buried
    recycle                     :NODE;
BEGIN
    count [].clk = freq_in;

        -- define desired counter mods
    div5  = !m1 & !m0 & count [] == 4;
    div10 = !m1 & m0 & count [] == 9;
    div12 = m1 & !m0 & count [] == 11;
    div15 = m1 & m0 & count [] == 14;
    recycle = div5 # div10 # div12 # div15;

    IF  recycle  THEN          -- reached terminal state?
        count [].d = 0;         -- recycle
    ELSE
        count [].d = count [].q + 1;   -- count up
    END IF;

        -- detect counter states 0 & 1
    freq_out = count [] <= 1;
END;
```

14A.7 Digital lock

```

CONSTANT  comb1 = H"7";
CONSTANT  comb2 = H"F";
CONSTANT  comb3 = H"A";
CONSTANT  comb4 = H"2";
    -- define desired lock combination

SUBDESIGN  lock
(
    enter          : INPUT;
    data[3..0]      : INPUT;
    unlock         : OUTPUT;
)

VARIABLE
    sm           : MACHINE
    WITH STATES (start, state1, state2, state3, done);
    -- don't care about bit patterns for state machine

BEGIN
    DEFAULTS
        unlock = GND;
    END DEFAULTS;
    sm.clk = enter;
    unlock = sm == done;           -- unlocked when at "done"

    CASE sm IS
        WHEN start =>           -- beginning state
            IF data[] == comb1 THEN sm = state1;
            ELSE sm = start;     -- wrong number
            END IF;
        WHEN state1 =>           -- 1st number ok
            IF data[] == comb2 THEN sm = state2;
            ELSE sm = start;     -- wrong number
            END IF;
        WHEN state2 =>           -- 2nd number ok
            IF data[] == comb3 THEN sm = state3;
            ELSE sm = start;     -- wrong number
            END IF;
        WHEN state3 =>           -- 3rd number ok
            IF data[] == comb4 THEN sm = done;
            ELSE sm = start;     -- wrong number
            END IF;
        WHEN done=>              -- last number ok
            sm = start;          -- relocks when hit enter
    END CASE;
END;

```

14A.8 Programmable frequency divider

```
SUBDESIGN divide_by
(
    freq_in, b[7..0]      :INPUT;
    freq_out              :OUTPUT;
)

VARIABLE
    divide_by[7..0]       :DFF;          -- 8 FFs

BEGIN
    divide_by[].clk = freq_in;

    IF divide_by[] <= 1 THEN           -- detect last state
        divide_by[].d = b[];          -- reload number
        freq_out = VCC;             -- make pulse
    ELSE divide_by[].d = divide_by[].q - 1;   -- decr.
    END IF;
END;
```

14A.9 State machine

```
SUBDESIGN state
(
    clock, r, s, t          :INPUT;
    q[2..0]                  :OUTPUT;
)

VARIABLE
sm      :MACHINE OF BITS (q[2..0])
WITH STATES   (s0 = B"000",
               s1 = B"001",
               s2 = B"010",
               s3 = B"011",
               s4 = B"100",
               s5 = B"101",
               s6 = B"110",
               s7 = B"111");

BEGIN
sm.clk = clock;

TABLE      -- present state/next state table
sm, r, s, t      =>      sm;
s0, X, X, X      =>      s1;
s1, X, X, X      =>      s7;
s2, X, X, X      =>      s1;
s3, 1, X, X      =>      s3;
s3, 0, 1, X      =>      s4;
s3, 0, 0, X      =>      s7;
s5, X, X, X      =>      s1;
s4, X, X, X      =>      s3;
s6, X, X, X      =>      s4;
s7, X, X, 0      =>      s1;
s7, X, X, 1      =>      s6;
END TABLE;

END;
```

Unit 14V Sequential Circuit Design with VHDL

14V.1 Gray code counter

```
ENTITY gray IS
PORT (
    clock, dir      : IN BIT;
    q               : OUT BIT_VECTOR (3 DOWNTO 0);
    index          : OUT BIT
);
END gray;

ARCHITECTURE vhdl OF gray IS
BEGIN
    PROCESS (clock)           -- detect clock change
        VARIABLE seq      : BIT_VECTOR (3 DOWNTO 0);
        -- declare bit vector variable for state machine
    BEGIN
        IF (clock'EVENT AND clock = '1')      THEN -- PGT
            IF (dir = '0')   THEN             -- count up
                CASE seq IS              -- determine next state
                    WHEN "0000" => seq := "0001";
                    WHEN "0001" => seq := "0011";
                    WHEN "0011" => seq := "0010";
                    WHEN "0010" => seq := "0110";
                    WHEN "0110" => seq := "0111";
                    WHEN "0111" => seq := "0101";
                    WHEN "0101" => seq := "0100";
                    WHEN "0100" => seq := "1100";
                    WHEN "1100" => seq := "1101";
                    WHEN "1101" => seq := "1111";
                    WHEN "1111" => seq := "1110";
                    WHEN "1110" => seq := "1010";
                    WHEN "1010" => seq := "1011";
                    WHEN "1011" => seq := "1001";
                    WHEN "1001" => seq := "1000";
                    WHEN "1000" => seq := "0000";
                END CASE;
            ELSE                         -- count down
                CASE seq IS              -- determine next state
                    WHEN "1000" => seq := "1001";
                    WHEN "1001" => seq := "1011";
                    WHEN "1011" => seq := "1010";
                    WHEN "1010" => seq := "1110";
                    WHEN "1110" => seq := "1111";
                    WHEN "1111" => seq := "1101";
                    WHEN "1101" => seq := "1100";
                    WHEN "1100" => seq := "0100";
                    WHEN "0100" => seq := "0101";
                    WHEN "0101" => seq := "0111";
                END CASE;
            END IF;
        END IF;
    END PROCESS;
END vhdl;
```

```

        WHEN "0111" => seq := "0110";
        WHEN "0110" => seq := "0010";
        WHEN "0010" => seq := "0011";
        WHEN "0011" => seq := "0001";
        WHEN "0001" => seq := "0000";
        WHEN "0000" => seq := "1000";
    END CASE;
END IF;
END IF;

q <= seq;           -- output counter variable to port

-- produce index signal
IF (seq = "0000") THEN      index <= '1';
ELSE                      index <= '0';
END IF;

END PROCESS;
END vhdl;

```

14V.2 Up/down BCD counter

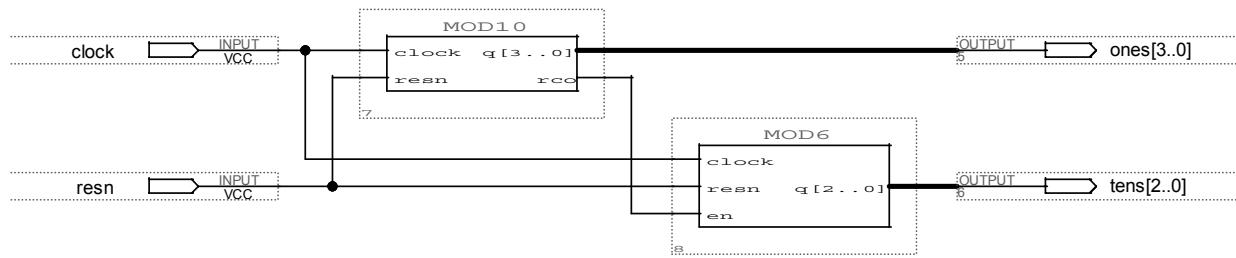
```
ENTITY updn_mod10 IS
PORT (
    clock      : IN BIT;
    c          : IN BIT_VECTOR (1 DOWNTO 0);
    count      : OUT INTEGER RANGE 0 TO 15;
    carryn    : OUT BIT
);
END updn_mod10;

ARCHITECTURE vhdl OF updn_mod10 IS
BEGIN
PROCESS (clock)           -- clock invokes process
VARIABLE bcd      : INTEGER RANGE 0 TO 9;  -- counter variable
BEGIN
    IF (clock'EVENT AND clock = '1') THEN      -- on PGT
        CASE  c  IS      -- determine control input
            WHEN  "00"  =>      -- reset
                bcd := 0;
            WHEN  "01"  =>      -- count down
                IF (bcd = 0) THEN      bcd := 9;
                ELSE                  bcd := bcd - 1;
                END IF;
            WHEN  "10"  =>      -- count up
                IF (bcd = 9) THEN      bcd := 0;
                ELSE                  bcd := bcd + 1;
                END IF;
            WHEN  "11"  =>      -- hold count
                bcd := bcd;
        END CASE;
    END IF;
    count <= bcd;                      -- output counter

    -- assert active-low ripple carry output
    IF      (c = "01" AND bcd = 0)      THEN      carryn <= '0';
    ELSIF  (c = "10" AND bcd = 9)      THEN      carryn <= '0';
    ELSE                                carryn <= '1';
    END IF;
END PROCESS;
END vhdl;
```

14V.3 Mod-60 BCD counter

mod60.gdf



```

ENTITY mod10 IS
PORT (
    clock, resn          :IN BIT;
    q                      :OUT INTEGER RANGE 0 TO 9;
    rco                   :OUT BIT    );
END mod10;

ARCHITECTURE vhdl OF mod10 IS
BEGIN
    PROCESS (clock)
    VARIABLE count         :INTEGER RANGE 0 TO 9;
    BEGIN

        IF (clock'EVENT AND clock = '1')      THEN
            IF (resn = '0' OR count = 9)      THEN
                count := 0;           -- reset/recycle
            ELSE
                count := count + 1;   -- increment
            END IF;
        END IF;

        IF (count = 9)      THEN rco <= '1';      -- RCO
        ELSE rco <= '0';
        END IF;

        q <= count;           -- output count

    END PROCESS;
END vhdl;

```

```

ENTITY mod6 IS
PORT (
    clock, resn, en      :IN BIT;
    q                      :OUT INTEGER RANGE 0 TO 5  );
END mod6;

ARCHITECTURE vhdl OF mod6 IS
BEGIN
    PROCESS (clock)
    VARIABLE count          :INTEGER RANGE 0 TO 5;      -- mod-6
    BEGIN

        IF (clock'EVENT AND clock = '1') THEN
            IF (resn = '0') THEN count := 0;          -- reset
            ELSIF (en = '1') THEN                      -- enabled
                IF (count = 5)  THEN
                    count := 0;                      -- recycle
                ELSE
                    count := count + 1;      -- increment
                END IF;
            END IF;
        END IF;

        q <= count;           -- output count

    END PROCESS;
END vhdl;

```

14V.4 Mod-100 binary counter

```
ENTITY mod100 IS
PORT (
    clock, enable, w      :IN BIT;
    q                      :OUT INTEGER RANGE 0 TO 99;
    pulse                  :OUT BIT);
END mod100;

ARCHITECTURE vhdl OF mod100 IS
BEGIN
    PROCESS (clock)
        VARIABLE count      :INTEGER RANGE 0 TO 99;
    BEGIN

        IF (clock'EVENT AND clock = '0') THEN          -- on NGT
            IF (enable = '1')      THEN                -- enabled?
                IF (count = 99)  THEN
                    count := 0;                      -- recycle
                ELSE
                    count := count + 1;           -- increment
                END IF;
            END IF;
        END IF;

        q <= count;                                -- output count

        -- control pulse width of output
        IF (w = '0' AND count >= 95)     THEN  pulse <= '1';
        ELSIF (w = '1' AND count >= 90)   THEN  pulse <= '1';
        ELSE  pulse <= '0';
        END IF;

    END PROCESS;
END vhdl;
```

14V.5 Stepper motor sequence controller

```

ENTITY half_step IS
PORT (
    step, cw, go           : IN BIT;
    q                      : OUT BIT_VECTOR (3 DOWNTO 0));
END half_step;

ARCHITECTURE vhdl OF half_step IS
BEGIN
PROCESS (step)
VARIABLE stepper        :BIT_VECTOR (3 DOWNTO 0);
BEGIN

    IF (step'EVENT AND step = '1')      THEN
        IF (go = '1' AND cw = '0')      THEN
            -- enabled for CCW steps
            CASE stepper IS          -- define CCW sequence
                WHEN "0101"      => stepper := "0100";
                WHEN "0100"      => stepper := "0110";
                WHEN "0110"      => stepper := "0010";
                WHEN "0010"      => stepper := "1010";
                WHEN "1010"      => stepper := "1000";
                WHEN "1000"      => stepper := "1001";
                WHEN "1001"      => stepper := "0001";
                WHEN "0001"      => stepper := "0101";
                WHEN OTHERS       => stepper := "0101";
            END CASE;

            ELSIF (go = '1' AND cw = '1')      THEN
                -- enabled for CW steps
                CASE stepper IS          -- define CW sequence
                    WHEN "0101"      => stepper := "0001";
                    WHEN "0001"      => stepper := "1001";
                    WHEN "1001"      => stepper := "1000";
                    WHEN "1000"      => stepper := "1010";
                    WHEN "1010"      => stepper := "0010";
                    WHEN "0010"      => stepper := "0110";
                    WHEN "0110"      => stepper := "0100";
                    WHEN "0100"      => stepper := "0101";
                    WHEN OTHERS       => stepper := "0101";
                END CASE;
            END IF;
        END IF;

        q <= stepper;           -- output count
    END PROCESS;
END vhdl;

```

14V.6 Variable frequency divider

```
ENTITY divider IS
PORT (
    freq_in          : IN BIT;
    m                : IN BIT_VECTOR (1 DOWNTO 0);
    freq_out         : OUT BIT);
END divider;

ARCHITECTURE vhdl OF divider IS
BEGIN
    PROCESS (freq_in)
        VARIABLE count      : INTEGER RANGE 0 TO 14;
                                -- buried counter
    BEGIN

        IF (freq_in'EVENT AND freq_in = '1') THEN
            -- determine if at selected terminal state
            IF      (m = "00" AND count = 4) THEN
                count := 0;
            ELSIF (m = "01" AND count = 9) THEN
                count := 0;
            ELSIF (m = "10" AND count = 11) THEN
                count := 0;
            ELSIF (m = "11" AND count = 14) THEN
                count := 0;
            ELSE
                count := count + 1; -- incr.
            END IF;
        END IF;

        -- freq-out detects first 2 counter states
        IF (count <= 1) THEN      freq_out <= '1';
        ELSE                      freq_out <= '0';
        END IF;

    END PROCESS;
END vhdl;
```

14V.7 Digital lock

```

ENTITY lock IS
PORT (
    enter          : IN BIT;
    data           : IN INTEGER RANGE 0 TO 15;
    unlock         : OUT BIT      );
END lock;

ARCHITECTURE vhdl OF lock IS
CONSTANT comb1      : INTEGER RANGE 0 TO 15 := 7;
CONSTANT comb2      : INTEGER RANGE 0 TO 15 := 15;
CONSTANT comb3      : INTEGER RANGE 0 TO 15 := 10;
CONSTANT comb4      : INTEGER RANGE 0 TO 15 := 2;
                                         -- define desired lock combination
BEGIN
PROCESS (enter)
TYPE lock_state IS (start, state1, state2, state3, done);
-- enumerated data type
VARIABLE sm          : lock_state;
BEGIN
    IF (sm = done) THEN      unlock <= '1';      -- unlocked
    ELSE                      unlock <= '0';      -- locked
    END IF;

    IF (enter'EVENT AND enter = '1') THEN
        CASE sm IS
            WHEN start =>          -- beginning state
                IF (data = comb1) THEN sm := state1;
                ELSE sm := start;   -- wrong number
                END IF;
            WHEN state1 =>         -- 1st number ok
                IF (data = comb2) THEN sm := state2;
                ELSE sm := start;   -- wrong number
                END IF;
            WHEN state2 =>         -- 2nd number ok
                IF (data = comb3) THEN sm := state3;
                ELSE sm := start;   -- wrong number
                END IF;
            WHEN state3 =>         -- 3rd number ok
                IF (data = comb4) THEN sm := done;
                ELSE sm := start;   -- wrong number
                END IF;
            WHEN done=> sm := start; -- relocks
        END CASE;
    END IF;

END PROCESS;
END vhdl;

```

14V.8 Programmable frequency divider

```
ENTITY divide_by IS
PORT (
    freq_in          : IN BIT;
    b                : IN INTEGER RANGE 0 TO 255;
    freq_out         : OUT BIT
);
END divide_by;

ARCHITECTURE vhdl OF divide_by IS
BEGIN
    PROCESS (freq_in)
    VARIABLE divider      :INTEGER RANGE 0 TO 255;
                    -- variable modulus counter

    BEGIN
        IF (freq_in'EVENT AND freq_in='1') THEN
            IF divider <= 1 THEN          -- reload
                divider := b;
            ELSE
                divider := divider - 1;   -- decr.
            END IF;
        END IF;

        IF divider <= 1 THEN          -- detect state 1
            freq_out <= '1';
        ELSE
            freq_out <= '0';
        END IF;
    END PROCESS;
END vhdl;
```

14V.9 State machine

```
ENTITY state IS
PORT (
    clock, r, s, t      :IN BIT;
    q                   :OUT BIT_VECTOR (2 DOWNTO 0)
);
END state;

ARCHITECTURE vhdl OF state IS
BEGIN
    PROCESS (clock)
    VARIABLE sm           :BIT_VECTOR (2 DOWNTO 0);
    BEGIN

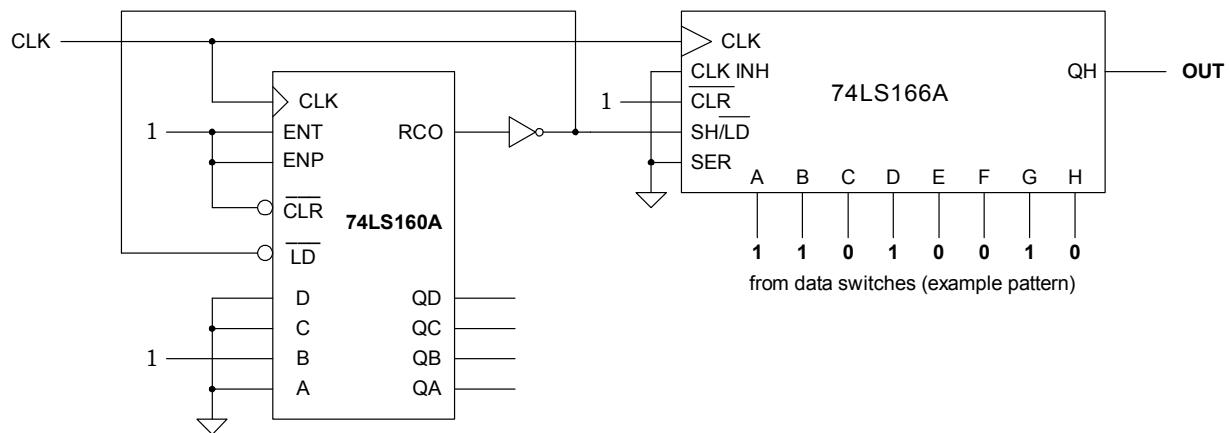
        IF (clock'EVENT AND clock = '1') THEN
            CASE sm IS
                WHEN "001" => sm := "111"; -- define sequence
                WHEN "111" => -- conditional
                    IF (t = '1') THEN
                        sm := "110";
                    ELSE
                        sm := "001";
                    END IF;
                WHEN "110" => sm := "100";
                WHEN "100" => sm := "011";
                WHEN "011" => -- conditional
                    IF (r = '0' AND s = '0') THEN
                        sm := "111";
                    ELSIF (r = '0' AND s = '1') THEN
                        sm := "100";
                    ELSE
                        sm := "011";
                    END IF;
                WHEN OTHERS => sm := "001"; -- self-correcting & starting
            END CASE;
        END IF;

        q <= sm; -- output state machine

    END PROCESS;
END vhdl;
```

Unit 15 Shift Register Applications

15.1 Waveform pattern generator



15.2 Waveform pattern generator pattern_gen_

```
%      waveform pattern generator - AHDL solution %
SUBDESIGN  pattern_gen_a
(
    clk, data[0..7]           :INPUT;
    serial_out, shift/load   :OUTPUT;
)

VARIABLE
    q[0..7]                  :DFF;          -- shift register
    control                   :MACHINE WITH STATES
        (sh/ld, s1, s2, s3, s4, s5, s6, s7);
        -- mod-8 control counter

BEGIN
    q[].clk = clk;
    control.clk = clk;
    serial_out = q7.q;

    TABLE      -- present state/next state table
        control    =>    control;
        sh/ld      =>    s1;
        s1         =>    s2;
        s2         =>    s3;
        s3         =>    s4;
        s4         =>    s5;
        s5         =>    s6;
        s6         =>    s7;
        s7         =>    sh/ld;
    END TABLE;

    IF (control == sh/ld)      THEN
        q[0..7].d = data[0..7];           -- parallel load
        shift/load = VCC;                -- shift/load pulse
    ELSE
        q[0..7].d = (GND, q[0..6].q);   -- shift data
        shift/load = GND;
    END IF;
END;
```

```

-- waveform pattern generator - VHDL solution
ENTITY pattern_gen_v IS
PORT (
    clk                      :IN BIT;
    data                     :IN BIT_VECTOR (0 TO 7);
    serial_out, shift_load  :OUT BIT
);
END pattern_gen_v;

ARCHITECTURE vhdl OF pattern_gen_v IS
BEGIN
PROCESS (clk)
    TYPE count_state IS
        (sh_ld, s1, s2, s3, s4, s5, s6, s7);
        -- enumerated data type
    VARIABLE control          :count_state;
    VARIABLE q                :BIT_VECTOR (0 TO 7);
    VARIABLE ser              :BIT;
    BEGIN
        ser := '0';           -- serial input
        serial_out <= q(7);   -- serial output

        -- define shift/load control signal
        IF (control = sh_ld) THEN shift_load <= '1';
        ELSE shift_load <= '0';
        END IF;

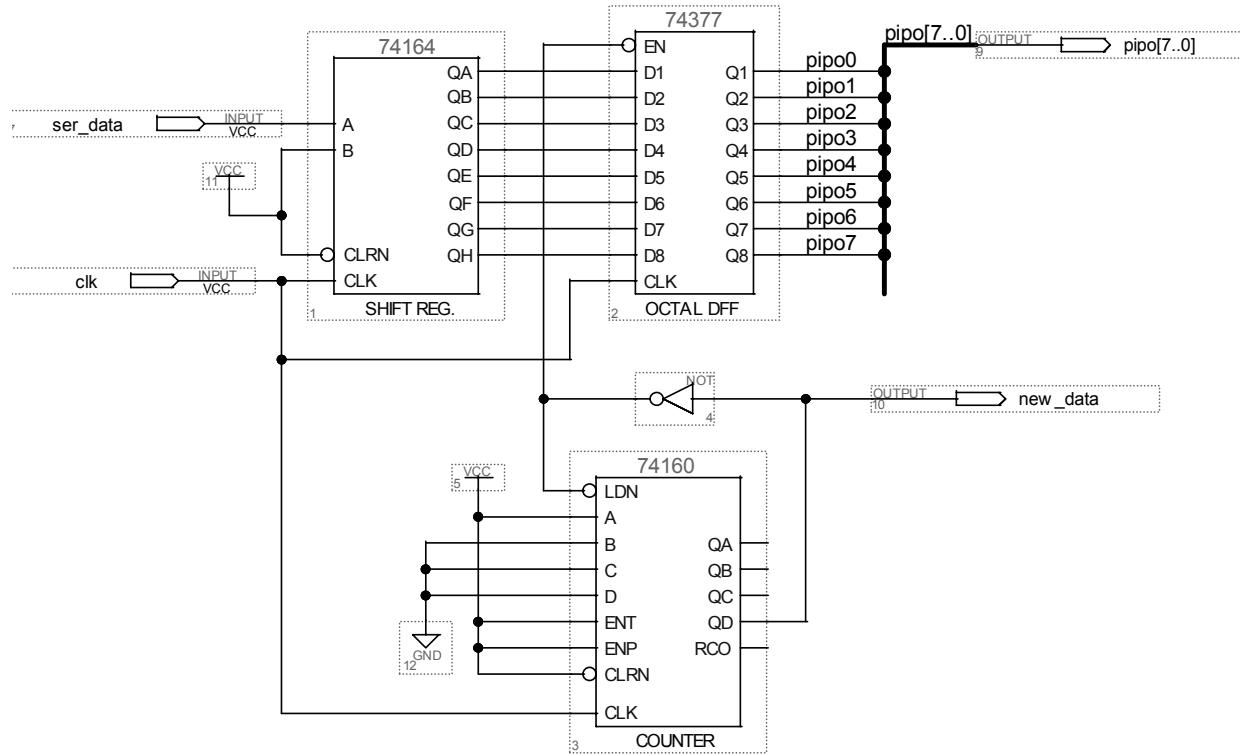
        IF (clk'EVENT AND clk = '1')      THEN
            IF (control = sh_ld)  THEN
                q := data;           -- parallel load
            ELSE q := (ser & q(0 TO 6)); -- serial shift
            END IF;
            -- define control sequence
            CASE control IS
                WHEN sh_ld      => control := s1;
                WHEN s1        => control := s2;
                WHEN s2        => control := s3;
                WHEN s3        => control := s4;
                WHEN s4        => control := s5;
                WHEN s5        => control := s6;
                WHEN s6        => control := s7;
                WHEN s7        => control := sh_ld;
            END CASE;
        END IF;

    END PROCESS;
END vhdl;

```

15.3 Serial data buffer

`serial_data.gdf`



```

SUBDESIGN serial_data_a          -- AHDL solution
(
    clk, ser_data           :INPUT;
    pip0[0..7], new_data    :OUTPUT;
)

VARIABLE
    pip0[0..7]      :DFFE;      -- D FFs with enable
    sipo[0..7]       :DFF;
    control         :MACHINE WITH STATES
                      (s0, s1, s2, s3, s4, s5, s6, s7);

BEGIN
    pip0[].clk = clk;
    pip0[].ena = new_data;      -- enable pip0 register
    pip0[].d = sipo[] .q;      -- parallel out to parallel in
    sipo[] .clk = clk;
    sipo[0..7].d = (ser_data, sipo[0..6].q);   -- ser shift
    control.clk = clk;
    new_data = (control == s7);      -- time to parallel load

    TABLE          -- present state/next state table
        control    =>    control;
        s0          =>    s1;
        s1          =>    s2;
        s2          =>    s3;
        s3          =>    s4;
        s4          =>    s5;
        s5          =>    s6;
        s6          =>    s7;
        s7          =>    s0;
    END TABLE;
END;

```

```

ENTITY serial_data_v IS          -- VHDL solution
PORT (
    clk, ser_data      :IN BIT;
    pipo               :OUT BIT_VECTOR (0 TO 7);
    new_data           :OUT BIT
);
END serial_data_v;

ARCHITECTURE vhdl OF serial_data_v IS
BEGIN
PROCESS (clk)
    TYPE count_state IS (s0, s1, s2, s3, s4, s5, s6, s7);
                                -- enumerated data type
    VARIABLE control      :count_state;
    VARIABLE reg          :BIT_VECTOR (0 TO 7);
    VARIABLE sipo         :BIT_VECTOR (0 TO 7);

    BEGIN
        pipo <= reg;      -- connect buried register to port
        -- detect data load
        IF (control = s7) THEN new_data <= '1';
        ELSE new_data <= '0';
        END IF;

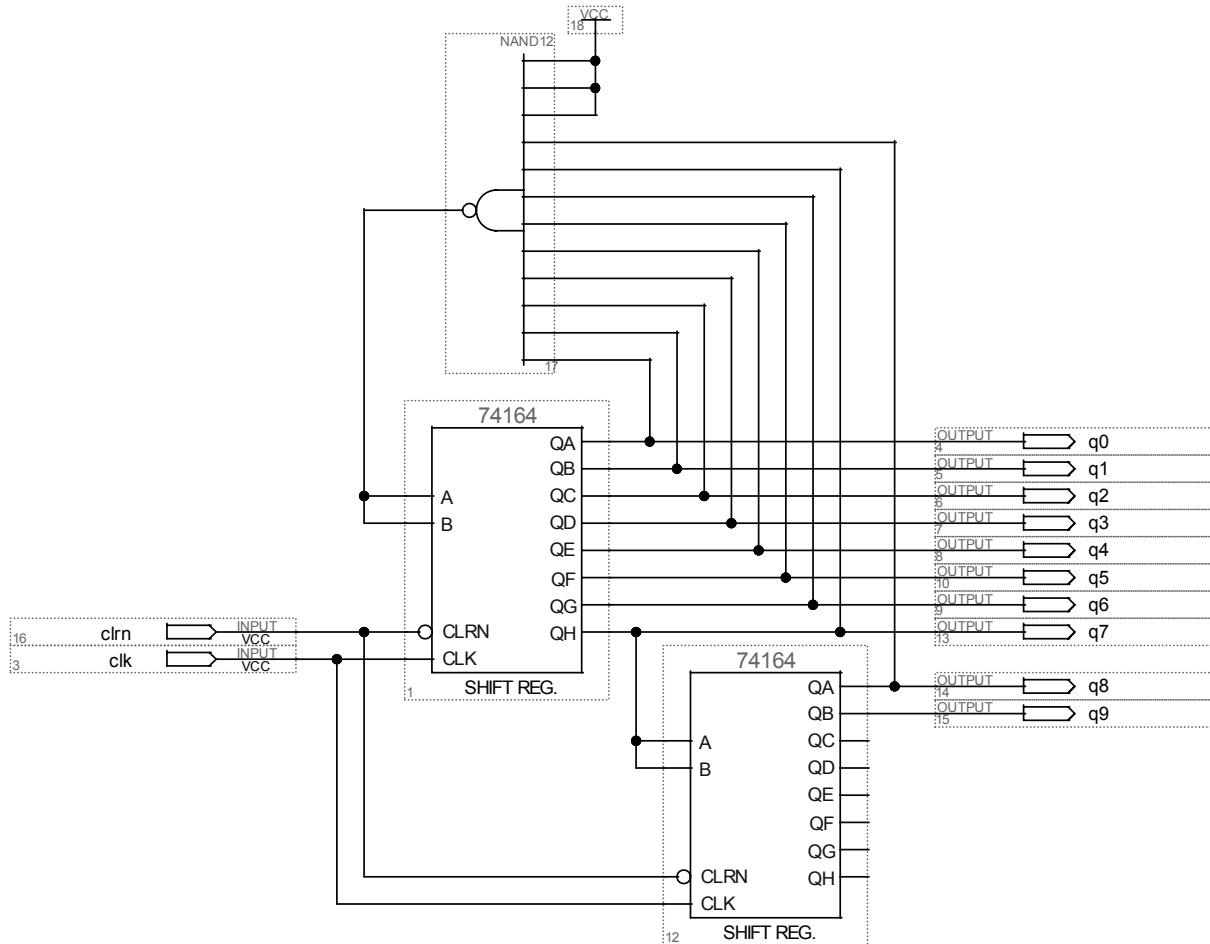
        IF (clk'EVENT AND clk = '1') THEN
            sipo := (ser_data & sipo(0 TO 6));      -- shift
            CASE control IS                      -- control sequence
                WHEN s0    => control := s1;
                WHEN s1    => control := s2;
                WHEN s2    => control := s3;
                WHEN s3    => control := s4;
                WHEN s4    => control := s5;
                WHEN s5    => control := s6;
                WHEN s6    => control := s7;
                WHEN s7    => control := s0;
            END CASE;
            IF (control = s7) THEN
                reg := sipo;      -- parallel load
            END IF;
        END IF;

    END PROCESS;
END;

```

15.4 Mod-10 ring counter

ring10.gdf



```

SUBDESIGN ring10a          -- AHDL solution
(
    clk           :INPUT;
    q[0..9]       :OUTPUT;
)

VARIABLE
    q[0..9]       :DFF;      -- 10 bit register
    ser           :NODE;     -- serial input signal

BEGIN
    q[].clk = clk;

    ser = !(q0 & q1 & q2 & q3 & q4 & q5 & q6 & q7 & q8);
           -- serial feedback (NAND)

    q[0..9].d = (ser, q[0..8].q);      -- shift
END;

```

```

ENTITY ring10v IS          -- VHDL solution
PORT (
    clk           :IN BIT;
    q            :OUT BIT_VECTOR (0 TO 9)
);
END ring10v;

ARCHITECTURE vhdl OF ring10v IS
SIGNAL ser           :BIT;      -- serial feedback signal
BEGIN
PROCESS (clk)
    VARIABLE r        :BIT_VECTOR (0 TO 9);
BEGIN
    ser <= NOT (r(0) AND r(1) AND r(2) AND r(3) AND
                 r(4) AND r(5) AND r(6) AND r(7) AND r(8));
                -- serial feedback function (NAND)

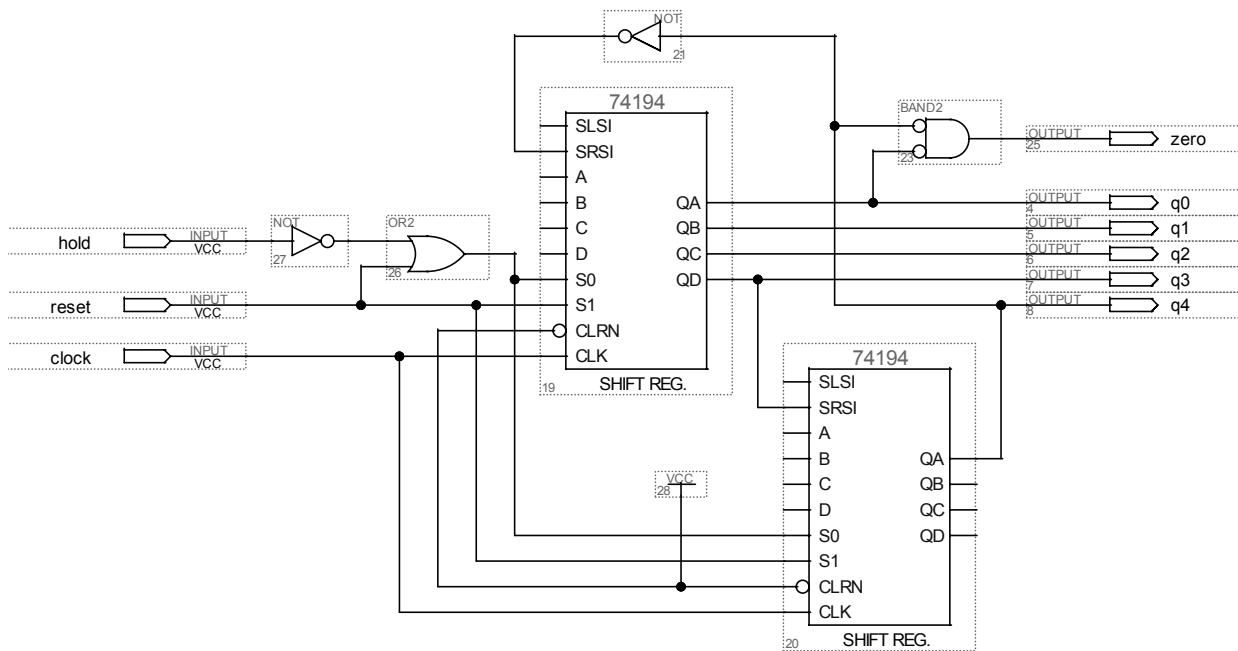
    IF (clk'EVENT AND clk = '1')      THEN
        r := (ser & r(0 TO 8));      -- shift
    END IF;

    q <= r;                      -- connect to output port
END PROCESS;
END vhdl;

```

15.4 Mod-10 Johnson counter

johnson10.gdf



```

SUBDESIGN johnson10a          -- AHDL solution
(
    clock, reset, hold      :INPUT;
    q[0..4], zero           :OUTPUT;
)

VARIABLE
    q[0..4]                 :DFF;        -- 5-bit reg
    ser                      :NODE;       -- feedback

BEGIN
    q[].clk = clock;
    ser = !q4;                -- feedback function
    zero = !q0 & !q4;         -- decode 00000

    IF      reset  THEN  q[].d = B"00000";      -- reset
    ELSIF   hold   THEN  q[].d = q[].q;        -- hold
    ELSE    q[0..4].d = (ser, q[0..3].q);     -- count
    END IF;
END;

```

```

ENTITY johnson10v  IS          -- VHDL solution
PORT (
    clock, reset, hold      :IN BIT;
    q                         :OUT BIT_VECTOR (0 TO 4);
    zero                     :OUT BIT
);
END johnson10v;

ARCHITECTURE vhdl OF johnson10v IS
SIGNAL ser                  :BIT;        -- feedback
BEGIN
PROCESS (clock)
    VARIABLE r                :BIT_VECTOR (0 TO 4);
    BEGIN
        ser <= NOT r(4);        -- feedback function

        IF (clock'EVENT AND clock = '1') THEN
            IF (reset = '1') THEN r := "00000";  -- reset
            ELSIF (hold = '1') THEN r := r;        -- hold
            ELSE r := (ser & r(0 TO 3));        -- count
            END IF;
        END IF;

        q <= r;                  -- output register
        zero <= (NOT r(0)) AND (NOT r(4));      -- decode
    END PROCESS;
END vhdl;

```

15.6 Mod-31 LFSR counter

```

SUBDESIGN mod31lfsr          -- AHDL solution
(
    clock, resetn, enable      :INPUT;
    q[0..4]                   :OUTPUT;
)

VARIABLE
    q[0..4]                  :DFF;
    ser                      :NODE;

BEGIN
    q[].clk = clock;
    ser = !(q4 $ q2);           -- XNOR

    IF !resetn THEN q[].d = B"00000";      -- reset
    ELSIF !enable THEN q[].d = q[].q;        -- disable
    ELSE q[0..4].d = (ser, q[0..3].q);      -- count
    END IF;
END;

```

```

ENTITY mod31lfsr_v IS          -- VHDL solution
PORT (
    clock, resetn, enable      :IN BIT;
    q                          :OUT BIT_VECTOR (0 TO 4)
);
END mod31lfsr_v;

ARCHITECTURE vhdl OF mod31lfsr_v IS
SIGNAL ser                      :BIT;
BEGIN
PROCESS (clock)
    VARIABLE reg                 :BIT_VECTOR (0 TO 4);
    BEGIN
        ser <= NOT (reg(4) XOR reg(2));           -- XNOR

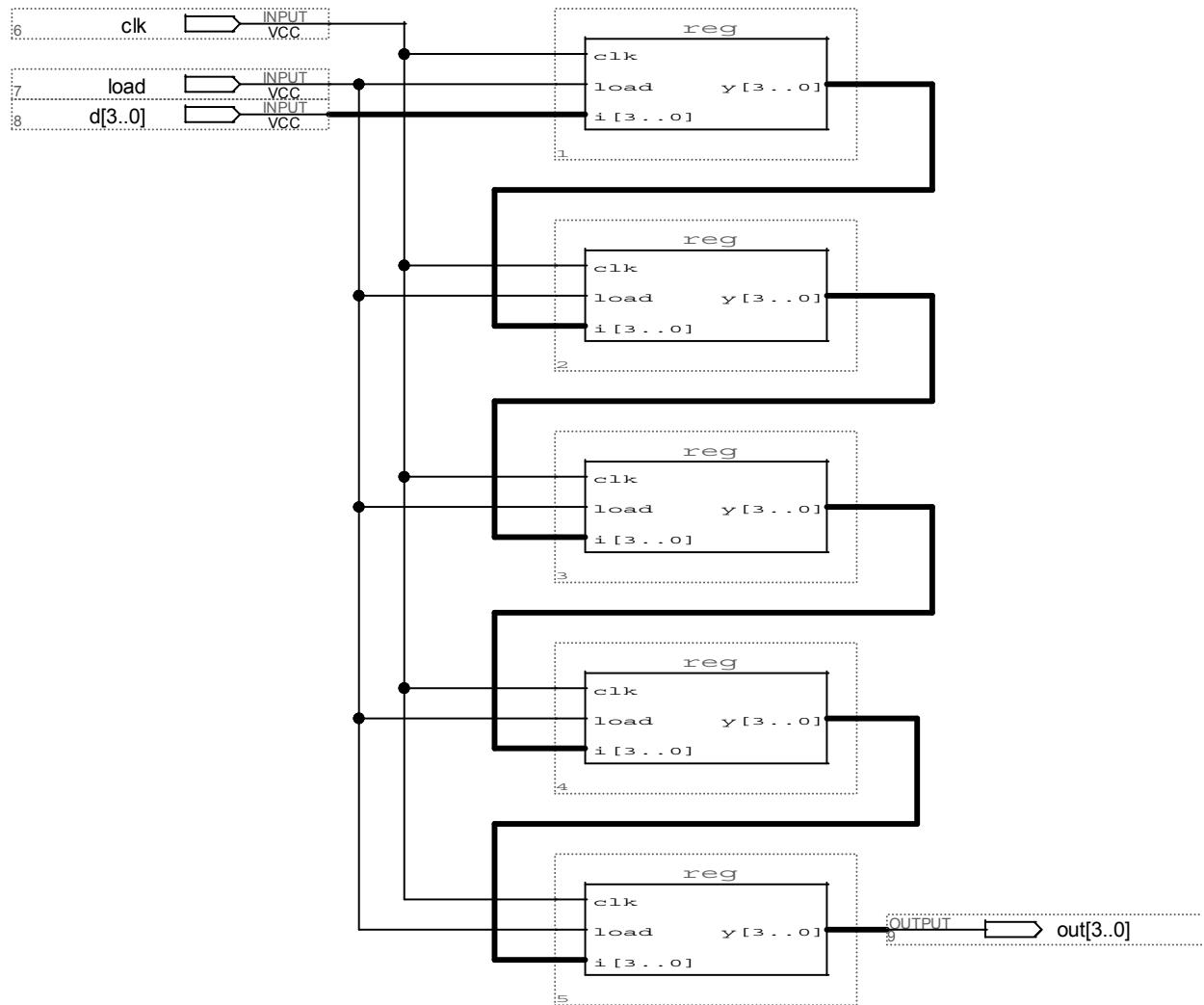
        IF (clock'EVENT AND clock = '1') THEN
            IF (resetn = '0') THEN
                reg := "00000";                     -- reset
            ELSIF (enable = '0') THEN
                reg := reg;                       -- disable
            ELSE reg := (ser & reg(0 TO 3));      -- count
            END IF;
        END IF;

        q <= reg;                         -- output count
    END PROCESS;
END vhdl;

```

15.7 Parallel data pipeline

pipeline.gdf



```

SUBDESIGN reg          -- AHDL solution
(
    clk, load, i[3..0]      :INPUT;
    y[3..0]                 :OUTPUT;
)
VARIABLE
    y[3..0]                :DFF;      -- 4-bit reg.
BEGIN
    y[].clk = clk;

    IF load THEN y[].d = i[];      -- load
    ELSE y[3..0].d = y[3..0].q;    -- hold
    END IF;
END;

```

```

ENTITY reg IS           -- VHDL solution
PORT (
    clk, load      :IN BIT;
    i              :IN BIT_VECTOR (3 DOWNTO 0);
    y              :OUT BIT_VECTOR (3 DOWNTO 0)
);
END reg;

ARCHITECTURE vhdl OF reg IS
BEGIN
PROCESS (clk)
    VARIABLE reg1      :BIT_VECTOR (3 DOWNTO 0);
                    -- 4-bit register

    BEGIN
    IF (clk'EVENT AND clk = '1') THEN
        IF (load = '1') THEN reg1 := i;      -- load
        END IF;
    END IF;

    y <= reg1;      -- register to output port

END PROCESS;
END vhdl;

```

15.8 Special-purpose data register

```
SUBDESIGN special_data          -- AHDL solution
(
    clk, s[1..0], i[3..0]      :INPUT;
    q[7..0]                     :OUTPUT;
)

VARIABLE
    q[7..0]                   :DFF;           -- 8-bit register

BEGIN
    q[].clk = clk;

    CASE s[] IS
        WHEN 0 =>           -- determine operation
                            -- load low nibble
                            q[7..0].d = (q[7..4].q, i[3..0]);
        WHEN 1 =>           -- load high nibble
                            q[7..0].d = (i[3..0], q[3..0].q);
        WHEN 2 =>           -- swap nibbles
                            q[7..0].d = (q[3..0].q, q[7..4].q);
        WHEN 3 =>           -- rotate right
                            q[7..0].d = (q0.q, q[7..1].q);
    END CASE;

END;
```

```

ENTITY special_data_v IS
PORT (
    clk           :IN BIT;
    s             :IN INTEGER RANGE 0 TO 3;
    i             :IN BIT_VECTOR (3 DOWNTO 0);
    q             :OUT BIT_VECTOR (7 DOWNTO 0)
);
END special_data_v;

ARCHITECTURE vhdl OF special_data_v IS
BEGIN
PROCESS (clk)
VARIABLE reg          :BIT_VECTOR (7 DOWNTO 0);
                    -- 8-bit register
BEGIN

IF (clk'EVENT AND clk = '1') THEN

    CASE s IS          -- determine operation
        WHEN 0 => reg(7 DOWNTO 0) :=
            (reg(7 DOWNTO 4) & i(3 DOWNTO 0));
            -- load low nibble
        WHEN 1 => reg(7 DOWNTO 0) :=
            (i(3 DOWNTO 0) & reg(3 DOWNTO 0));
            -- load high nibble
        WHEN 2 => reg(7 DOWNTO 0) :=
            (reg(3 DOWNTO 0) & reg(7 DOWNTO 4));
            -- swap nibbles
        WHEN 3 => reg(7 DOWNTO 0) :=
            (reg(0) & reg(7 DOWNTO 1));
            -- rotate right
    END CASE;

END IF;

q <= reg;           -- register to output port

END PROCESS;

END vhdl;

```

15.9 Shift/Rotate register

```
SUBDESIGN shift_rotate           -- AHDL solution

(
    clk, m[2..0]          :INPUT;
    ser, d[7..0]          :INPUT;
    q[7..0]                :OUTPUT;
)

VARIABLE
    q[7..0]                :DFF;      -- 8-bit register

BEGIN
    q[].clk = clk;

    CASE m[] IS           -- determine control function

        WHEN 2 => q[7..0].d = d[7..0];
        -- load data

        WHEN 3 => q[7..0].d = q[7..0].q;
        -- hold data

        WHEN 4 => q[7..0].d = (ser, q[7..1].q);
        -- shift right

        WHEN 5 => q[7..0].d = (q[6..0].q, ser);
        -- shift left

        WHEN 6 => q[7..0].d = (q0.q, q[7..1].q);
        -- rotate right

        WHEN 7 => q[7..0].d = (q[6..0].q, q7.q);
        -- rotate left

        WHEN OTHERS => q[7..0].d = B"00000000";
        -- clear register with 0 or 1

    END CASE;

END;
```

```

ENTITY shift_rotate_v IS          -- VHDL solution
PORT (
    clk, ser      :IN BIT;
    m            :IN INTEGER RANGE 0 TO 7;
    d            :IN BIT_VECTOR (7 DOWNTO 0);
    q            :OUT BIT_VECTOR (7 DOWNTO 0)
);
END shift_rotate_v;

ARCHITECTURE vhdl OF shift_rotate_v IS
BEGIN
PROCESS (clk)
VARIABLE reg           :BIT_VECTOR (7 DOWNTO 0);
                           -- 8-bit register
BEGIN
IF (clk'EVENT AND clk = '1') THEN

    CASE m IS           -- determine function

        WHEN 2 => reg(7 DOWNTO 0) := d(7 DOWNTO 0);
                           -- load data

        WHEN 3 => reg(7 DOWNTO 0) := reg(7 DOWNTO 0);
                           -- hold data

        WHEN 4 => reg(7 DOWNTO 0) := (ser & reg(7 DOWNTO 1));
                           -- shift right

        WHEN 5 => reg(7 DOWNTO 0) := (reg(6 DOWNTO 0) & ser);
                           -- shift left

        WHEN 6 => reg(7 DOWNTO 0) := (reg(0) & reg(7 DOWNTO 1));
                           -- rotate right

        WHEN 7 => reg(7 DOWNTO 0) := (reg(6 DOWNTO 0) & reg(7));
                           -- rotate left

        WHEN OTHERS => reg(7 DOWNTO 0) := "00000000";
                           -- clear register with 0 or 1

    END CASE;

END IF;

q <= reg;           -- connect register to port

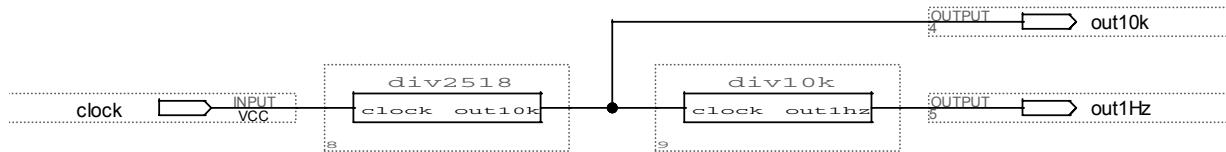
END PROCESS;
END vhdl;

```

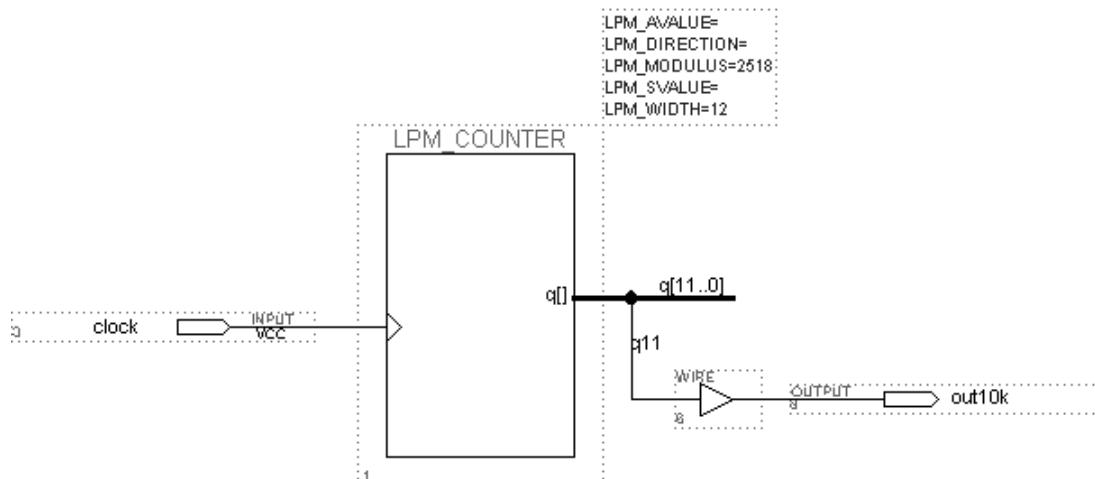
Unit 16 Library of Parameterized Modules

16.1 Serial data communications circuit (see Example 16-1)

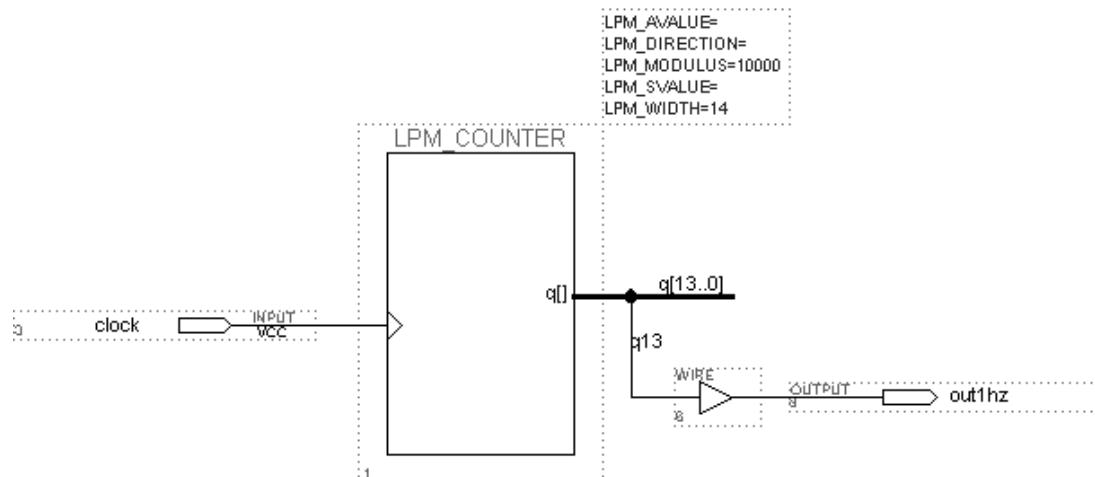
16.2 Clock divider *clock_div.gdf*



div2518.gdf

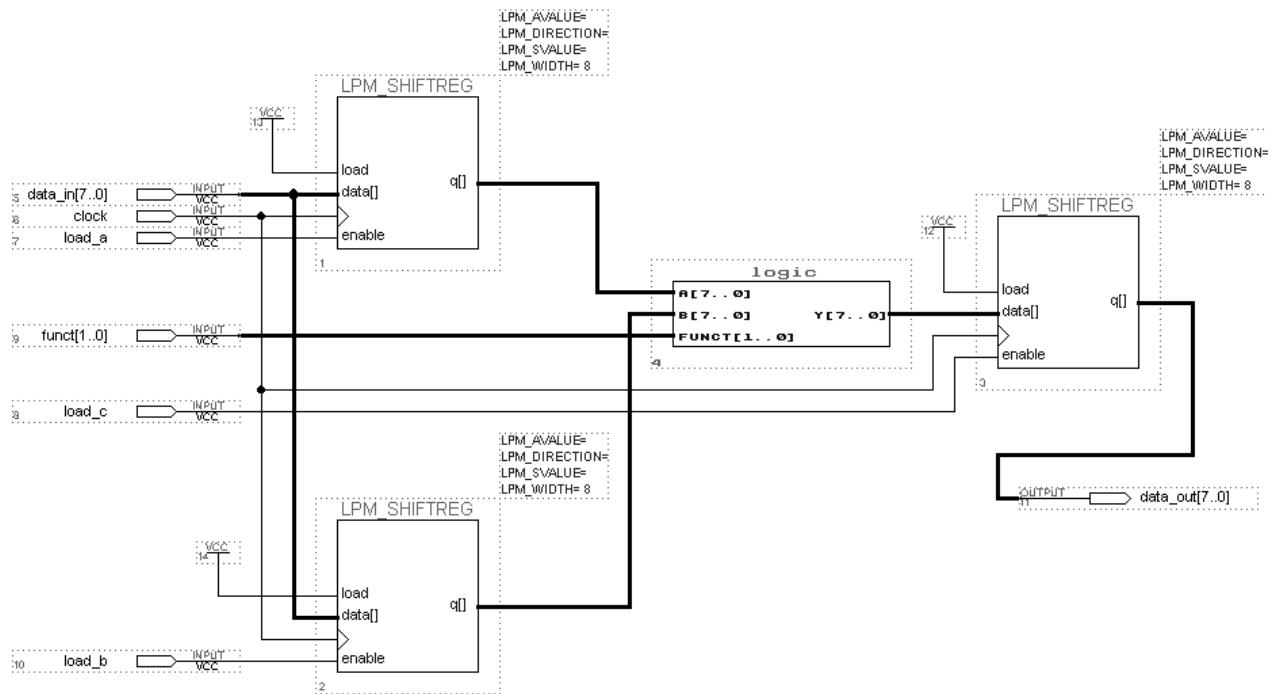


div10k.gdf



16.3 Data word logic unit

logic_registers.gdf



```

SUBDESIGN logic          -- AHDL solution
(
    a[7..0], b[7..0], funct[1..0]      : INPUT;
    y[7..0]                           : OUTPUT;
)

BEGIN
    CASE funct[] IS
        WHEN 0 => y[] = !a[];
        WHEN 1 => y[] = a[] # b[];
        WHEN 2 => y[] = a[] & b[];
        WHEN 3 => y[] = a[] $ b[];
    END CASE;
END;

```

```

ENTITY logic IS          -- VHDL solution
PORT (
    a, b      : IN BIT_VECTOR (7 DOWNTO 0);
    funct     : IN INTEGER RANGE 0 TO 3;
    y         : OUT BIT_VECTOR (7 DOWNTO 0)
);
END logic;

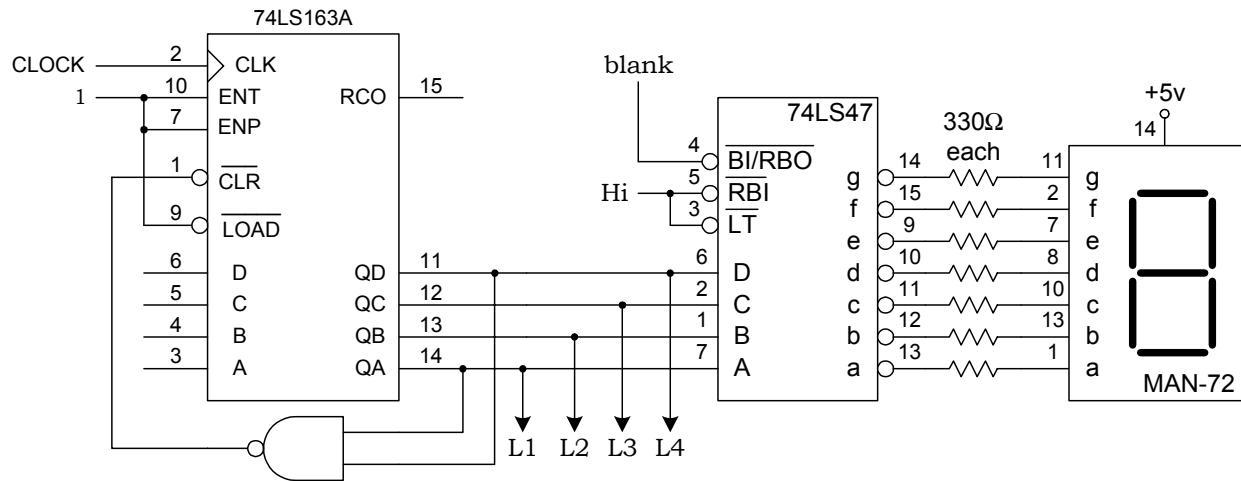
ARCHITECTURE vhdl OF logic IS
BEGIN
PROCESS (a, b, funct)
    BEGIN

        CASE funct IS
            WHEN 0 => y <= NOT a;           -- NOT A
            WHEN 1 => y <= a OR b;         -- A OR B
            WHEN 2 => y <= a AND b;        -- A AND B
            WHEN 3 => y <= a XOR b;       -- A XOR B
        END CASE;
    END PROCESS;
END vhdl;

```

Unit 17 Decoders and Displays

17.1 Modified 4-bit binary counter and display



17.2 Hex decoder/driver

```

SUBDESIGN hex_display          -- AHDL solution
(
    bin[3..0]           : INPUT;
    a,b,c,d,e,f,g      : OUTPUT;
)

BEGIN
    TABLE      -- common-anode truth table
        bin[]    =>    a , b , c , d , e , f , g;
        H"0"     =>    0 , 0 , 0 , 0 , 0 , 0 , 1;
        H"1"     =>    1 , 0 , 0 , 1 , 1 , 1 , 1;
        H"2"     =>    0 , 0 , 1 , 0 , 0 , 1 , 0 ;
        H"3"     =>    0 , 0 , 0 , 0 , 1 , 1 , 0 ;
        H"4"     =>    1 , 0 , 0 , 1 , 1 , 0 , 0 ;
        H"5"     =>    0 , 1 , 0 , 0 , 1 , 0 , 0 ;
        H"6"     =>    0 , 1 , 0 , 0 , 0 , 0 , 0 ;
        H"7"     =>    0 , 0 , 0 , 1 , 1 , 1 , 1 ;
        H"8"     =>    0 , 0 , 0 , 0 , 0 , 0 , 0 ;
        H"9"     =>    0 , 0 , 0 , 0 , 1 , 0 , 0 ;
        H"A"     =>    0 , 0 , 0 , 1 , 0 , 0 , 0 ;
        H"B"     =>    1 , 1 , 0 , 0 , 0 , 0 , 0 ;
        H"C"     =>    0 , 1 , 1 , 0 , 0 , 0 , 1 ;
        H"D"     =>    1 , 0 , 0 , 0 , 0 , 1 , 0 ;
        H"E"     =>    0 , 1 , 1 , 0 , 0 , 0 , 0 ;
        H"F"     =>    0 , 1 , 1 , 1 , 0 , 0 , 0 ;
    END TABLE;
END;

```

```

ENTITY hex_display IS          -- VHDL solution
PORT ( bin                  : IN BIT_VECTOR (3 DOWNTO 0);
       a,b,c,d,e,f,g      : OUT BIT );
END hex_display;

ARCHITECTURE vhdl OF hex_display IS
SIGNAL segments             : BIT_VECTOR (1 TO 7);
BEGIN
    PROCESS (bin)
    BEGIN

        CASE bin IS           -- common-anode display
            WHEN X"0"      => segments <= "0000001";
            WHEN X"1"      => segments <= "1001111";
            WHEN X"2"      => segments <= "0010010";
            WHEN X"3"      => segments <= "0000110";
            WHEN X"4"      => segments <= "1001100";
            WHEN X"5"      => segments <= "0100100";
            WHEN X"6"      => segments <= "0100000";
            WHEN X"7"      => segments <= "0001111";
            WHEN X"8"      => segments <= "0000000";
            WHEN X"9"      => segments <= "0000100";
            WHEN X"A"      => segments <= "0001000";
            WHEN X"B"      => segments <= "1100000";
            WHEN X"C"      => segments <= "0110001";
            WHEN X"D"      => segments <= "1000010";
            WHEN X"E"      => segments <= "0110000";
            WHEN X"F"      => segments <= "0111000";
        END CASE;

    END PROCESS;

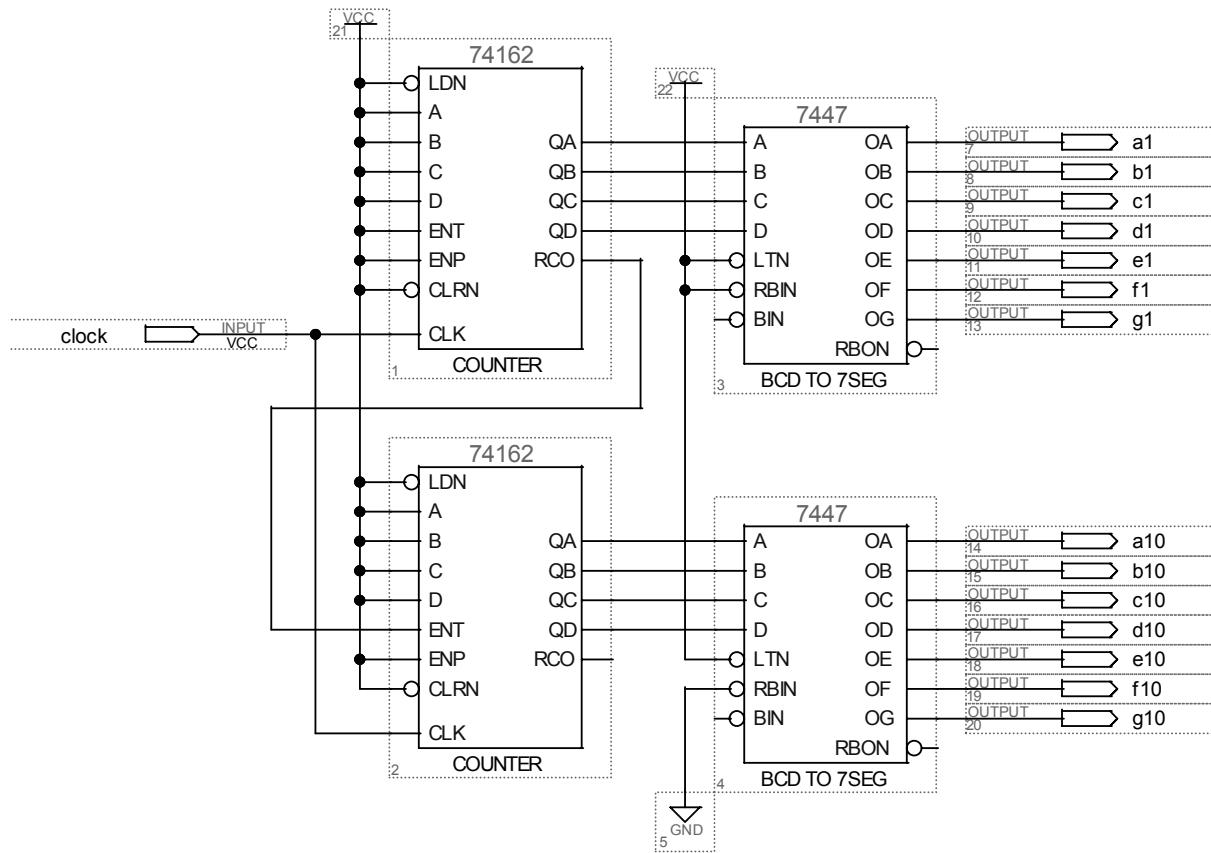
    a <= segments(1);           -- connect each cathode
    b <= segments(2);
    c <= segments(3);
    d <= segments(4);
    e <= segments(5);
    f <= segments(6);
    g <= segments(7);

END vhdl;

```

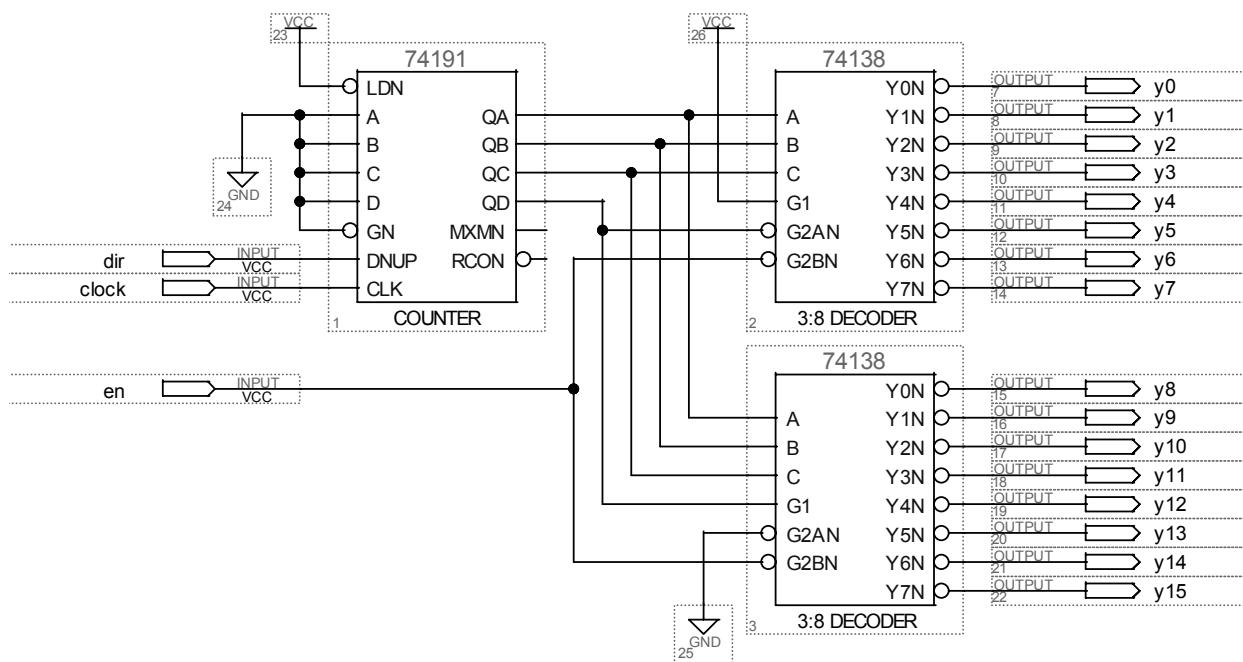
17.3 Mod-100 BCD counter and 7-segment display

`mod100_display.gdf`



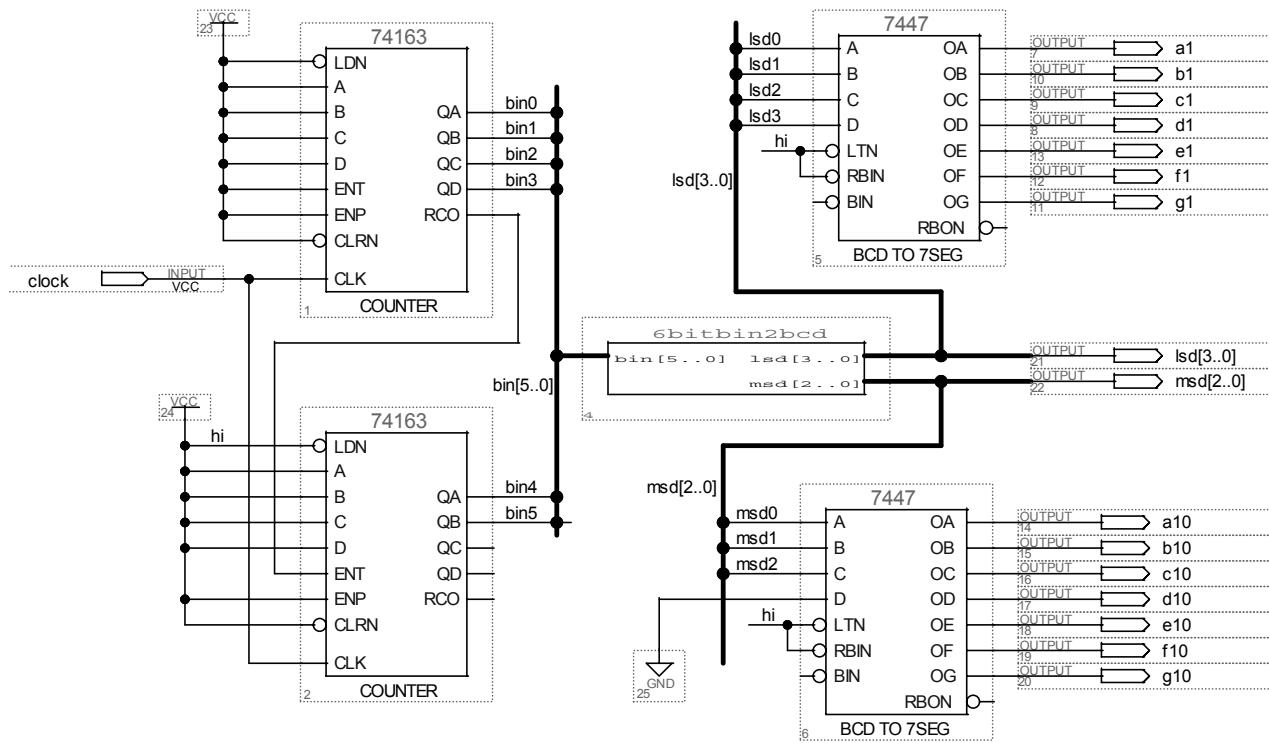
17.4 Mod-16 counter and decoder

`mod16_decoder.gdf`

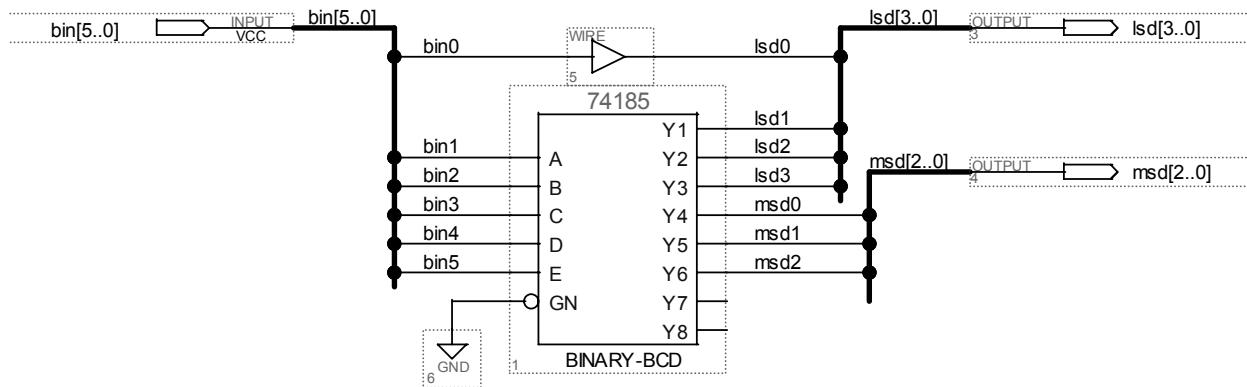


17.5 Binary-to-BCD converter and display

mod64_2bcd.gdf



6bitbin2bcd.gdf



17.6 Memory decoder

```
SUBDESIGN mem_decode          -- AHDL solution
(
    a[15..0], memr, memw      :INPUT;
    cs[5..0]                  :OUTPUT;
)

VARIABLE
    mem_en                   :NODE;
                            -- buried node

BEGIN
    DEFAULTS
        cs[] = B"111111";    -- default inactive
    END DEFAULTS;

    mem_en = memr $ memw;    -- XOR strobe signals
                            -- only enables if one is active

    TABLE                  -- Xs are don't care address bits
        a[], mem_en         => cs[];
        B"000000XXXXXXXXXX", 1   => B"111110";
        B"000001XXXXXXXXXX", 1   => B"111101";
        B"000010XXXXXXXXXX", 1   => B"111011";
        B"000011XXXXXXXXXX", 1   => B"110111";
        B"111010XXXXXXXXXX", 1   => B"101111";
        B"111011XXXXXXXXXX", 1   => B"011111";
    END TABLE;

END;
```

```

ENTITY mem_decode IS
PORT (      a          : IN INTEGER RANGE 0 TO 65535;
            memr, memw   : IN BIT;
            cs           : OUT BIT_VECTOR (5 DOWNTO 0)
);
END mem_decode;

ARCHITECTURE vhdl OF mem_decode IS
SIGNAL     mem_en       : BIT;        -- buried node
BEGIN
mem_en <= memr XOR memw;           -- XOR strobe signals
                                    -- only enables if one is active

PROCESS (a, mem_en)
    -- invoke on changes to address or strobes
BEGIN

IF mem_en = '1' THEN             -- only 1 strobe active?

    -- evaluate hex address ranges
IF      a >= 16#0000# AND a <= 16#03FF#
THEN    cs <= "111110";

ELSIF   a >= 16#0400# AND a <= 16#07FF#
THEN    cs <= "111101";

ELSIF   a >= 16#0800# AND a <= 16#0BFF#
THEN    cs <= "111011";

ELSIF   a >= 16#0C00# AND a <= 16#0FFF#
THEN    cs <= "110111";

ELSIF   a >= 16#E800# AND a <= 16#EBFF#
THEN    cs <= "101111";

ELSIF   a >= 16#EC00# AND a <= 16#EFFF#
THEN    cs <= "011111";

ELSE     -- not in desired address range
        cs <= "111111";
END IF;

ELSE      -- 0 or 2 strobes active
        cs <= "111111";
END IF;

END PROCESS;

END vhdl;

```

17.7 State decoder

```

SUBDESIGN state_decoder          -- AHDL solution
(    in[6..0]      :INPUT;
    y[9..1]       :OUTPUT;    )

BEGIN
    DEFAULTS
        y[] = B"000000000";           -- default inactive
    END DEFAULTS;
    TABLE                      -- selected states to decode
        in[] => y[];
        10   => B"000000001";
        20   => B"000000010";
        30   => B"000000100";
        40   => B"000001000";
        50   => B"000010000";
        60   => B"000100000";
        70   => B"001000000";
        80   => B"010000000";
        90   => B"100000000";
    END TABLE;
END;

```

```

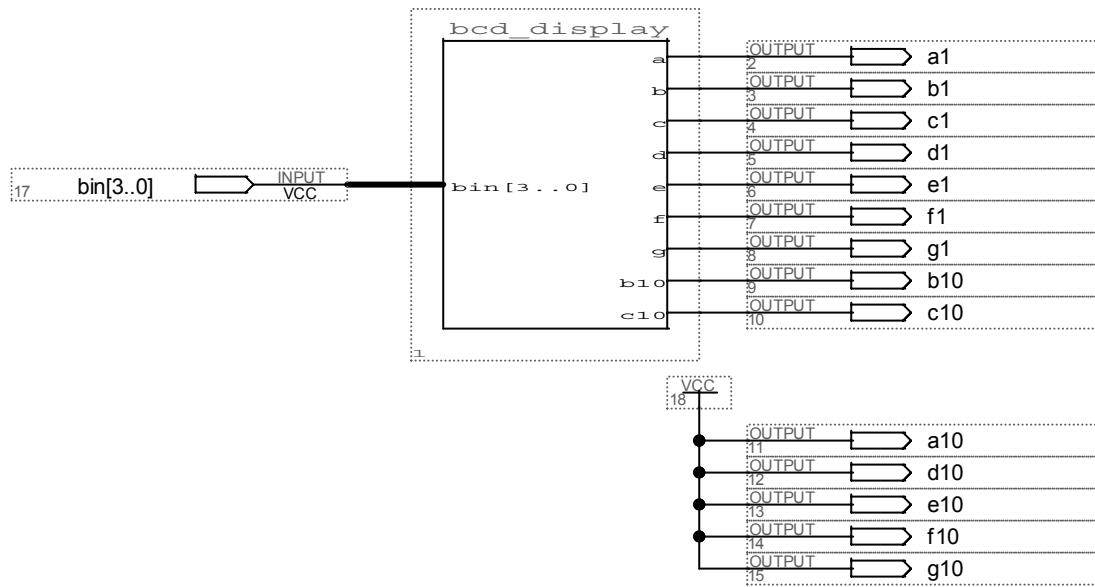
ENTITY state_decoder IS          -- VHDL solution
PORT (    input      : IN INTEGER RANGE 0 TO 99;
          y         : OUT BIT_VECTOR (9 DOWNTO 1) );
END state_decoder;

ARCHITECTURE vhdl OF state_decoder IS
BEGIN
    PROCESS (input)      -- invoke on input change
    BEGIN
        CASE input IS          -- decoded states
            WHEN 10      => y <= "000000001";
            WHEN 20      => y <= "000000010";
            WHEN 30      => y <= "000000100";
            WHEN 40      => y <= "000001000";
            WHEN 50      => y <= "000010000";
            WHEN 60      => y <= "000100000";
            WHEN 70      => y <= "001000000";
            WHEN 80      => y <= "010000000";
            WHEN 90      => y <= "100000000";
            WHEN OTHERS  => y <= "000000000";
        END CASE;
    END PROCESS;
END vhdl;

```

17.8 Binary-to-7-segment display decoder/driver

bcd_display2.gdf



```

SUBDESIGN bcd_display          -- AHDL solution
(
    bin[3..0]           : INPUT;
    a,b,c,d,e,f,g,b10,c10   : OUTPUT;
)

BEGIN
    TABLE             -- cathode at 0 lights segment
        bin[]  =>  a ,b, c, d, e, f, g,  b10, c10;
        H"0"   =>  0, 0, 0, 0, 0, 0, 1, 1 , 1;
        H"1"   =>  1, 0, 0, 1, 1, 1, 1, 1 , 1;
        H"2"   =>  0, 0, 1, 0, 0, 1, 0, 1 , 1;
        H"3"   =>  0, 0, 0, 0, 1, 1, 0, 1 , 1;
        H"4"   =>  1, 0, 0, 1, 1, 0, 0, 1 , 1;
        H"5"   =>  0, 1, 0, 0, 1, 0, 0, 1 , 1;
        H"6"   =>  0, 1, 0, 0, 0, 0, 0, 1 , 1;
        H"7"   =>  0, 0, 0, 1, 1, 1, 1, 1 , 1;
        H"8"   =>  0, 0, 0, 0, 0, 0, 0, 1 , 1;
        H"9"   =>  0, 0, 0, 0, 1, 0, 0, 1 , 1;
        H"A"   =>  0, 0, 0, 0, 0, 0, 1, 0 , 0;
        H"B"   =>  1, 0, 0, 1, 1, 1, 1, 0 , 0;
        H"C"   =>  0, 0, 1, 0, 0, 1, 0, 0 , 0;
        H"D"   =>  0, 0, 0, 0, 1, 1, 0, 0 , 0;
        H"E"   =>  1, 0, 0, 1, 1, 0, 0, 0 , 0;
        H"F"   =>  0, 1, 0, 0, 1, 0, 0, 0 , 0;
    END TABLE;
END;

```

```

ENTITY bcd_display IS          -- VHDL solution
PORT ( bin           : IN BIT_VECTOR (3 DOWNTO 0);
       a,b,c,d,e,f,g,b10,c10 : OUT BIT );
END bcd_display;

ARCHITECTURE vhdl OF bcd_display IS
SIGNAL ones_digit      : BIT_VECTOR (1 TO 7);
SIGNAL tens_digit      : BIT_VECTOR (2 TO 3);
BEGIN
    a <= ones_digit(1);      -- connect to output ports
    b <= ones_digit(2);
    c <= ones_digit(3);
    d <= ones_digit(4);
    e <= ones_digit(5);
    f <= ones_digit(6);
    g <= ones_digit(7);

    b10 <= tens_digit(2);
    c10 <= tens_digit(3);

```

-- continues on next page

```

PROCESS (bin)
BEGIN
    CASE bin IS
        -- determine segments lit for binary number

        WHEN X"0"    => ones_digit <= "0000001";
                            tens_digit <= "11";
        WHEN X"1"    => ones_digit <= "1001111";
                            tens_digit <= "11";
        WHEN X"2"    => ones_digit <= "0010010";
                            tens_digit <= "11";
        WHEN X"3"    => ones_digit <= "0000110";
                            tens_digit <= "11";
        WHEN X"4"    => ones_digit <= "1001100";
                            tens_digit <= "11";
        WHEN X"5"    => ones_digit <= "0100100";
                            tens_digit <= "11";
        WHEN X"6"    => ones_digit <= "0100000";
                            tens_digit <= "11";
        WHEN X"7"    => ones_digit <= "0001111";
                            tens_digit <= "11";
        WHEN X"8"    => ones_digit <= "0000000";
                            tens_digit <= "11";
        WHEN X"9"    => ones_digit <= "0000100";
                            tens_digit <= "11";

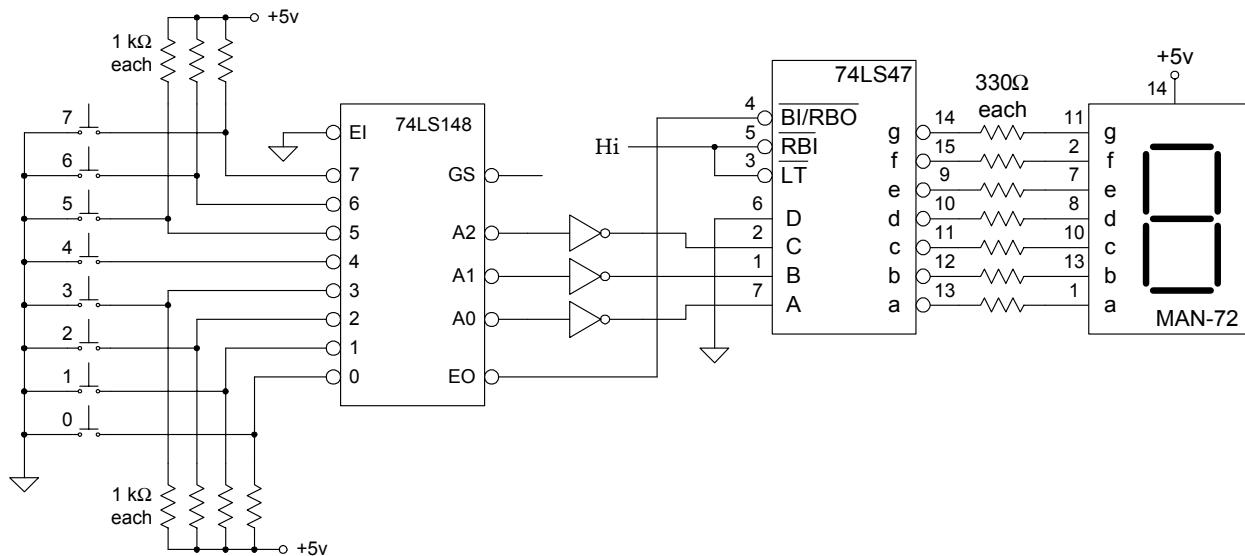
        WHEN X"A"   => ones_digit <= "0000001";
                            tens_digit <= "00";
        WHEN X"B"   => ones_digit <= "1001111";
                            tens_digit <= "00";
        WHEN X"C"   => ones_digit <= "0010010";
                            tens_digit <= "00";
        WHEN X"D"   => ones_digit <= "0000110";
                            tens_digit <= "00";
        WHEN X"E"   => ones_digit <= "1001100";
                            tens_digit <= "00";
        WHEN X"F"   => ones_digit <= "0100100";
                            tens_digit <= "00";

    END CASE;
END PROCESS;
END vhdl;

```

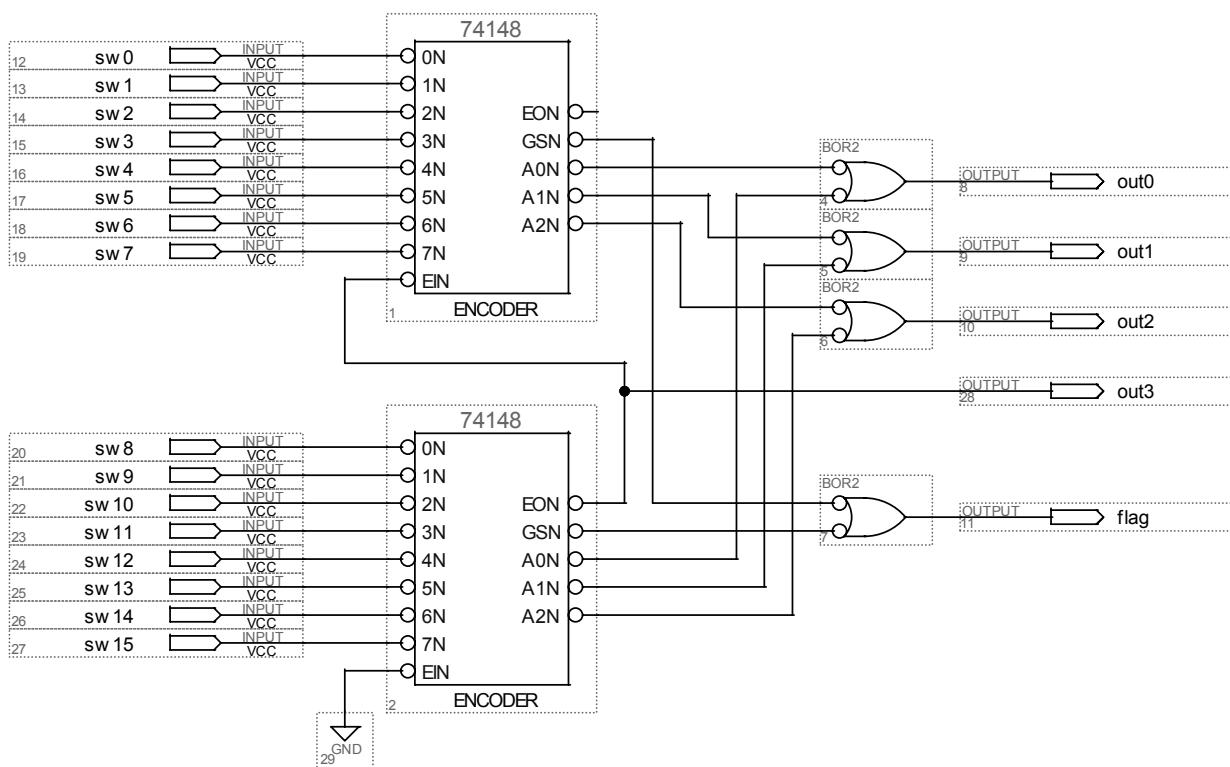
Unit 18 Encoders

18.1 Encoder and display



18.2 16-line priority encoder

encoder16.gdf



18.3 BCD encoder

```
SUBDESIGN encoder10          -- AHDL solution
(
    sw[9..0], en           :INPUT;
    out[3..0], strobe      :OUTPUT;
)
BEGIN
    TABLE      -- active-low decimal priority encoder
        sw[], en            =>    out[], strobe;
        B"XXXXXXXXXX", 0    =>    B"0000", 1;
        B"1111111111", 1    =>    B"0000", 1;
        B"1111111110", 1    =>    B"0000", 0;
        B"111111110X", 1    =>    B"0001", 0;
        B"11111110XX", 1    =>    B"0010", 0;
        B"1111110XXX", 1    =>    B"0011", 0;
        B"111110XXXX", 1    =>    B"0100", 0;
        B"11110XXXXX", 1    =>    B"0101", 0;
        B"1110XXXXXX", 1    =>    B"0110", 0;
        B"110XXXXXXX", 1    =>    B"0111", 0;
        B"10XXXXXXX", 1     =>    B"1000", 0;
        B"0XXXXXXX", 1      =>    B"1001", 0;
    END TABLE;
END;
```

```
ENTITY encoder10 IS          -- VHDL solution
PORT (
    sw       : IN BIT_VECTOR (9 DOWNTO 0);
    en       : IN BIT;
    output   : OUT BIT_VECTOR (3 DOWNTO 0);
    strobe   : OUT BIT );
END encoder10;

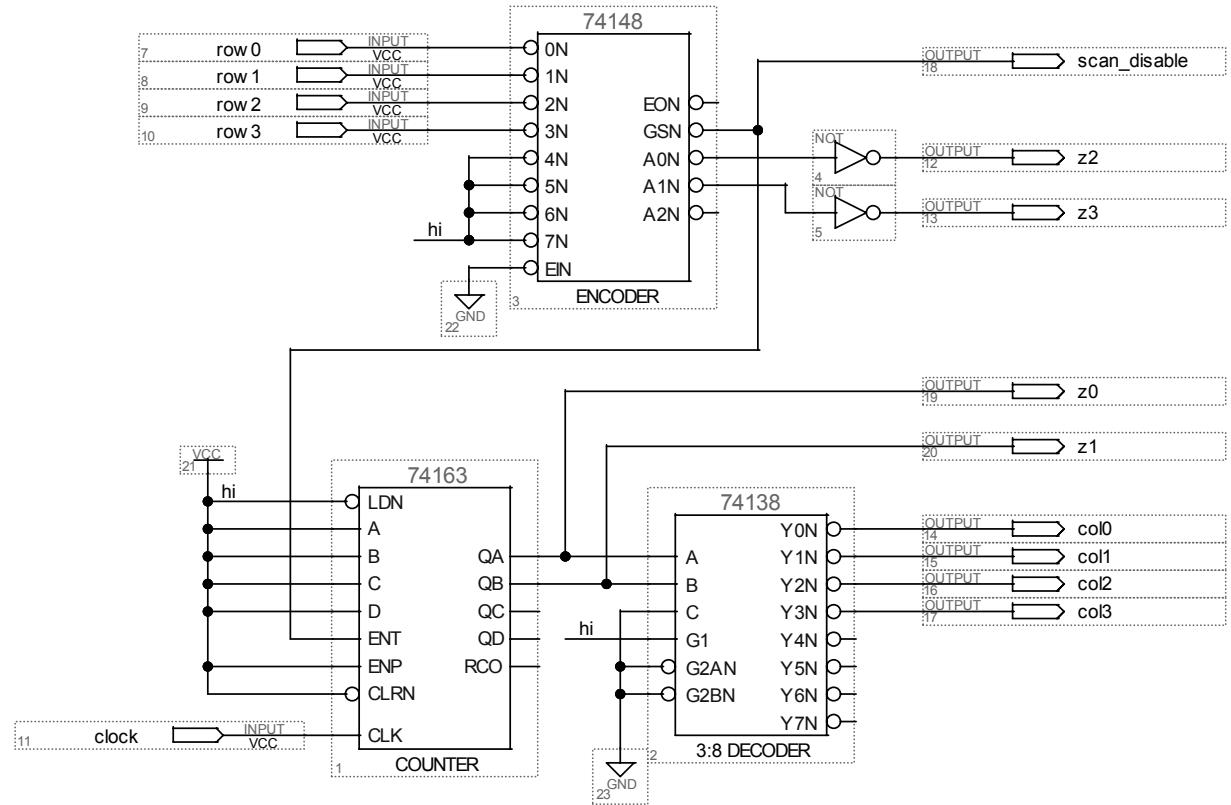
ARCHITECTURE vhdl OF encoder10 IS
BEGIN

    strobe <= '1' WHEN (en = '0' OR sw = "1111111111")
                ELSE '0';

    output <=    "0000" WHEN en = '0' ELSE
                 "1001" WHEN sw(9) = '0' ELSE
                 "1000" WHEN sw(8) = '0' ELSE
                 "0111" WHEN sw(7) = '0' ELSE
                 "0110" WHEN sw(6) = '0' ELSE
                 "0101" WHEN sw(5) = '0' ELSE
                 "0100" WHEN sw(4) = '0' ELSE
                 "0011" WHEN sw(3) = '0' ELSE
                 "0010" WHEN sw(2) = '0' ELSE
                 "0001" WHEN sw(1) = '0' ELSE
                 "0000";
END vhdl;
```

18.4 Scanning encoder

scan_encoder.gdf



18.5 HDL 74148 encoder

```
SUBDESIGN encoder_ahdl           -- AHDL solution
(
    in[7..0], ei                  :INPUT;
    a[2..0], gs, eo               :OUTPUT;
)

BEGIN

IF !ei THEN                      -- active-low enable

    TABLE
        -- priority encoder with inverted outputs
        in[]      => a[], gs, eo;
        B"11111111"  => B"111", VCC, GND;
        B"11111110"  => B"111", GND, VCC;
        B"1111110X"  => B"110", GND, VCC;
        B"111110XX"  => B"101", GND, VCC;
        B"11110XXX"  => B"100", GND, VCC;
        B"1110XXXX"  => B"011", GND, VCC;
        B"110XXXXX"  => B"010", GND, VCC;
        B"10XXXXXX"  => B"001", GND, VCC;
        B"0XXXXXXX"  => B"000", GND, VCC;
    END TABLE;

ELSE a[] = B"111"; gs = VCC; eo = VCC;      -- disabled

END IF;
END;
```

```

ENTITY encoder_vhdl IS          -- VHDL solution
PORT (
    input           :IN BIT_VECTOR (7 DOWNTO 0);
    ei             :IN BIT;
    a              :OUT BIT_VECTOR (2 DOWNTO 0);
    gs, eo         :OUT BIT
);
END encoder_vhdl;

ARCHITECTURE vhdl OF encoder_vhdl IS
BEGIN
PROCESS (input, ei)
BEGIN

    IF ei = '0' THEN          -- active-low enable
        -- check highest priority input first
        IF      input(7) = '0' THEN
            a <= "000"; gs <= '0'; eo <= '1';

        ELSIF   input(6) = '0' THEN
            a <= "001"; gs <= '0'; eo <= '1';

        ELSIF   input(5) = '0' THEN
            a <= "010"; gs <= '0'; eo <= '1';

        ELSIF   input(4) = '0' THEN
            a <= "011"; gs <= '0'; eo <= '1';

        ELSIF   input(3) = '0' THEN
            a <= "100"; gs <= '0'; eo <= '1';

        ELSIF   input(2) = '0' THEN
            a <= "101"; gs <= '0'; eo <= '1';

        ELSIF   input(1) = '0' THEN
            a <= "110"; gs <= '0'; eo <= '1';

        ELSIF   input(0) = '0' THEN
            a <= "111"; gs <= '0'; eo <= '1';

        ELSE          -- no input
            a <= "111"; gs <= '1'; eo <= '0';
        END IF;

    ELSE          -- disabled
        a <= "111"; gs <= '1'; eo <= '1';
    END IF;

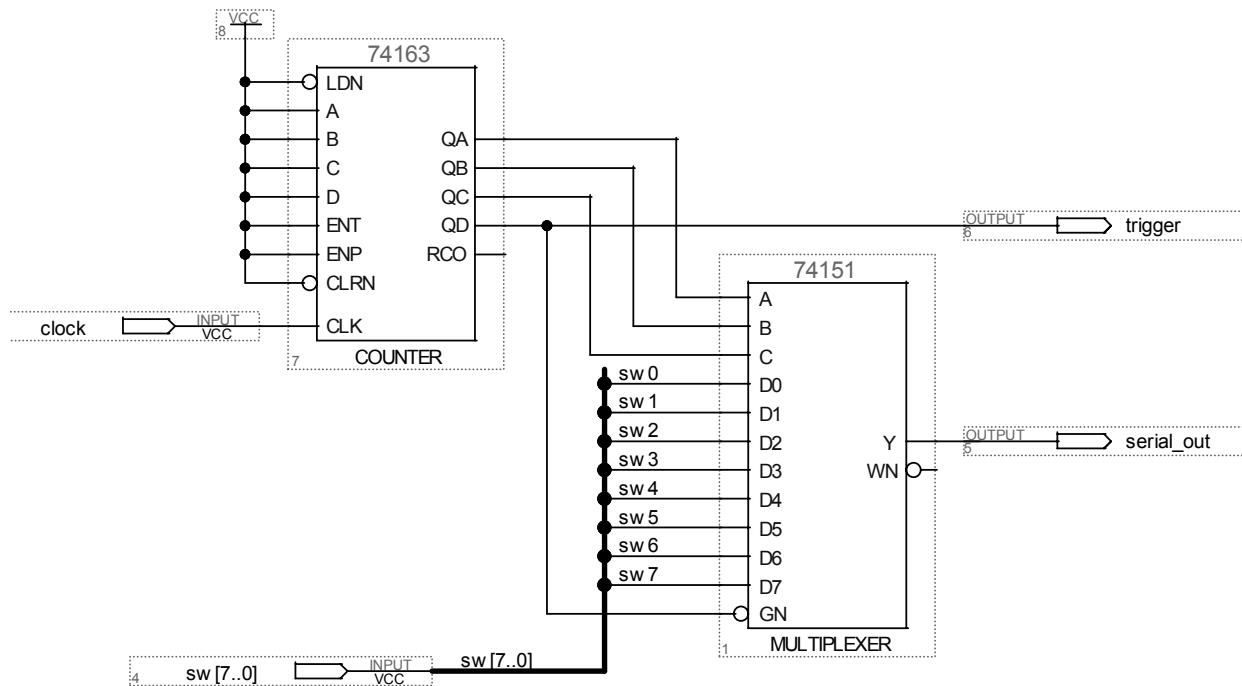
END PROCESS;
END vhdl;

```

Unit 19 Multiplexers

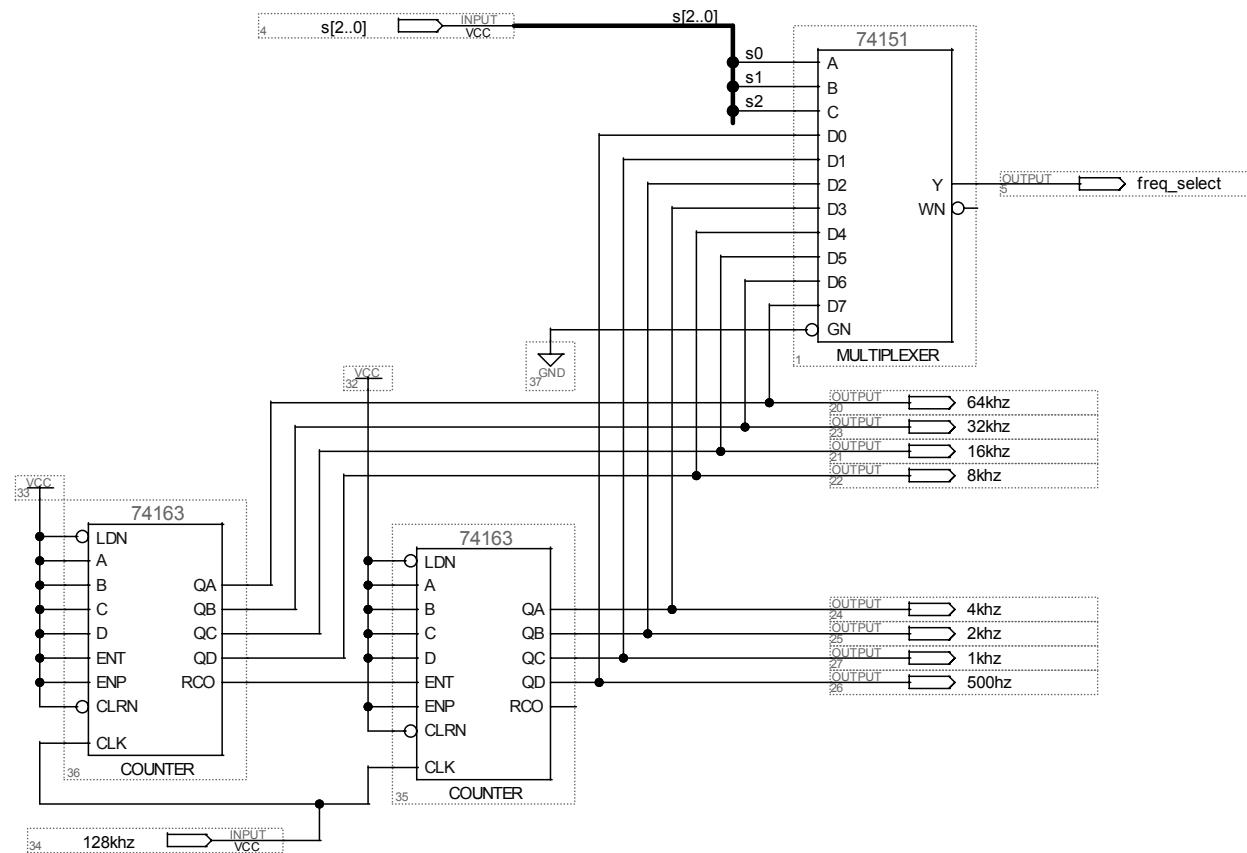
19.1 Serial data word generator

word_gen.gdf



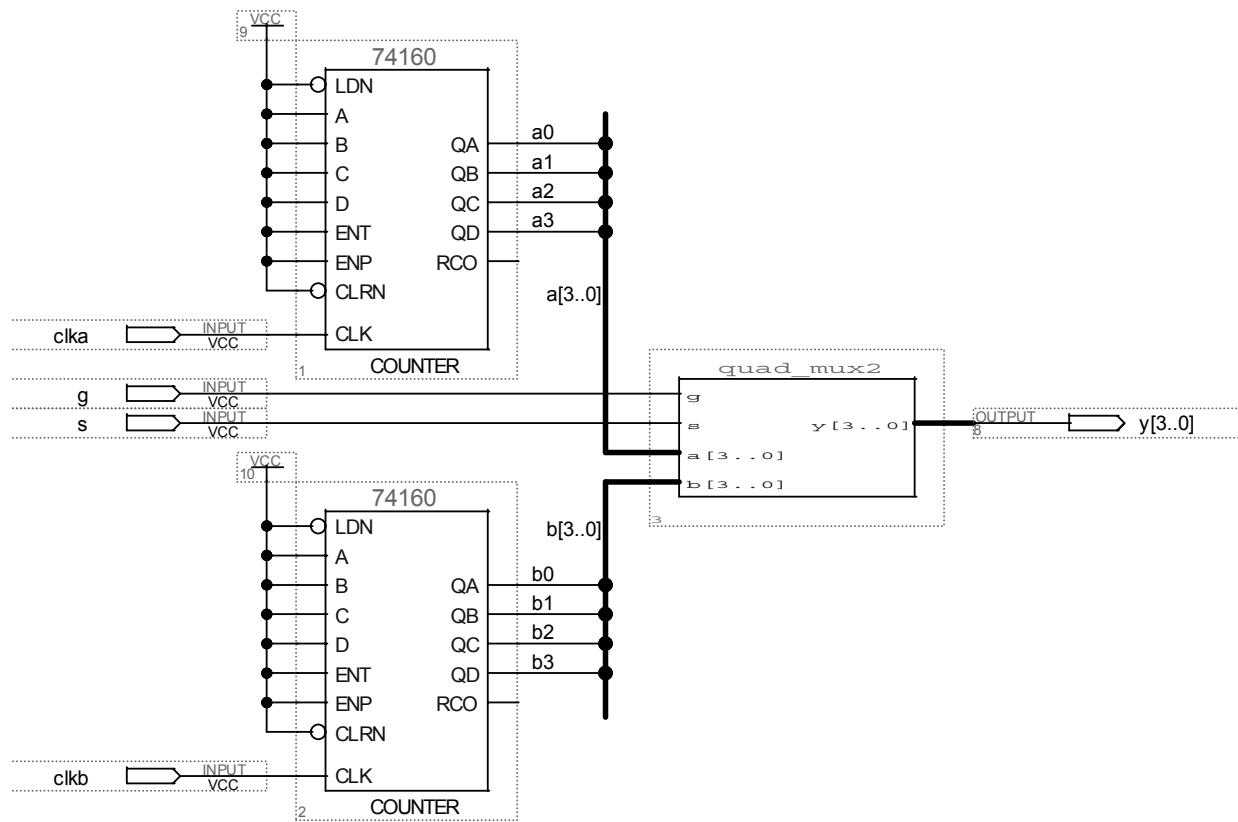
19.2 Variable frequency divider

`mux_freq_div.gdf`



19.3 Multiplexed BCD display

`mux_bcd.gdf`



```
SUBDESIGN quad_mux2          -- AHDL solution
(
    g, s, a[3..0], b[3..0]      :INPUT;
    y[3..0]                      :OUTPUT;
)

BEGIN
    IF g THEN                  -- enabled
        CASE s IS              -- select input channel
            WHEN 0 => y[] = a[];   -- channel A
            WHEN 1 => y[] = b[];   -- channel B
        END CASE;
    ELSE y[] = H"F";           -- output if disabled
    END IF;
END;
```

```

ENTITY quad_mux2 IS                                -- VHDL solution
PORT (      g, s          : IN BIT;
            a, b          : IN BIT_VECTOR (3 DOWNTO 0);
            y          : OUT BIT_VECTOR (3 DOWNTO 0) );
END quad_mux2;

ARCHITECTURE vhdl OF quad_mux2 IS
BEGIN
    PROCESS (g, s, a, b)           -- monitor all inputs
    BEGIN
        IF g = '1' THEN           -- enabled
            CASE s IS             -- select input
                WHEN '0' => y <= a;      -- A
                WHEN '1' => y <= b;      -- B
            END CASE;
        ELSE y <= "1111";        -- output if disabled
        END IF;
    END PROCESS;
END vhdl;

```

19.4 Quad, 3-channel MUX

```

SUBDESIGN quad_mux3                                -- AHDL solution
(
    s[1..0], a[3..0], b[3..0], c[3..0]      :INPUT;
    y[3..0]                                  :OUTPUT;
)

BEGIN
    CASE s[] IS                         -- input channel select
        WHEN 0 => y[] = 0;              -- disabled
        WHEN 1 => y[] = a[];           -- channel A
        WHEN 2 => y[] = b[];           -- channel B
        WHEN 3 => y[] = c[];           -- channel C
    END CASE;
END;

```

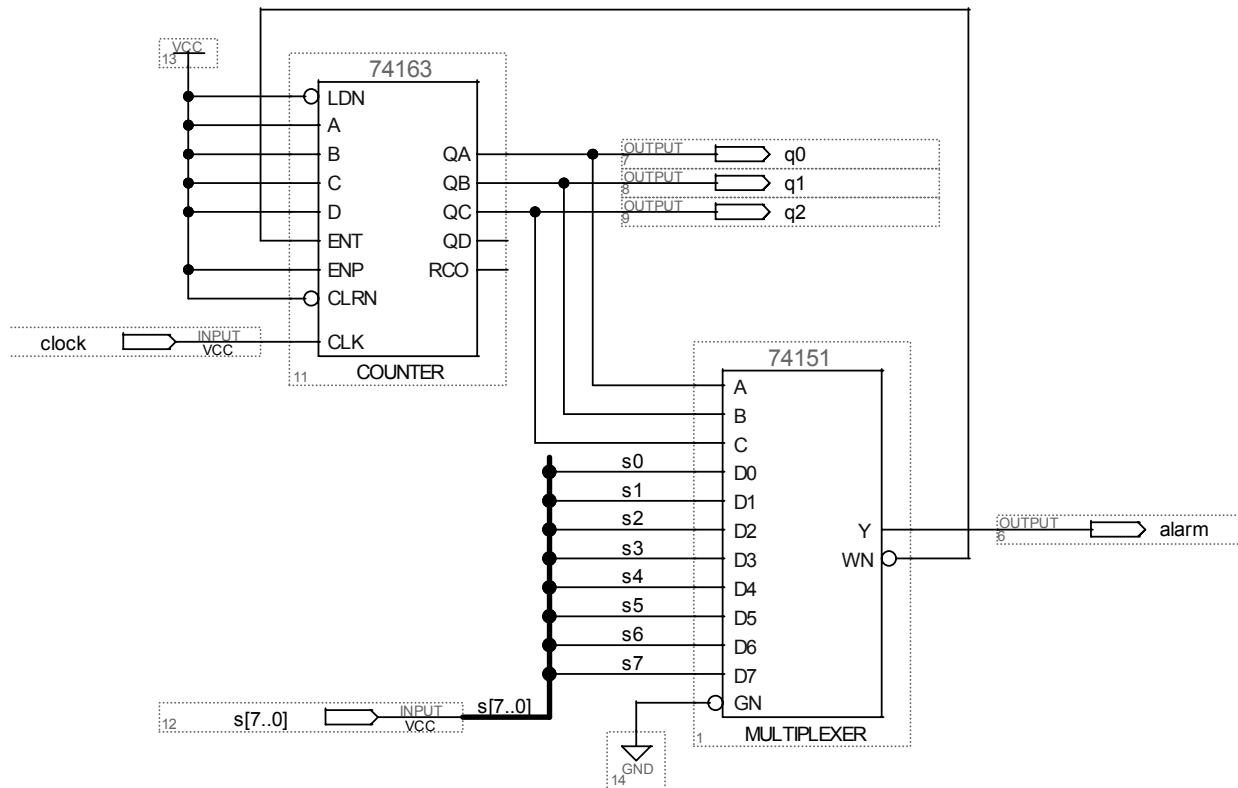
```

ENTITY quad_mux3 IS
PORT (      s          : IN INTEGER RANGE 0 TO 3;
             a, b, c    : IN BIT_VECTOR (3 DOWNTO 0);
             Y          : OUT BIT_VECTOR (3 DOWNTO 0)
);
END quad_mux3;

ARCHITECTURE vhdl OF quad_mux3 IS
BEGIN
    PROCESS (s, a, b, c)          -- monitor all inputs
    BEGIN
        CASE s IS              -- input channel select
            WHEN 0 => Y <= "0000";   -- disabled
            WHEN 1 => Y <= a;       -- channel A
            WHEN 2 => Y <= b;       -- channel B
            WHEN 3 => Y <= c;       -- channel C
        END CASE;
    END PROCESS;
END vhdl;

```

19.5 Alarm system alarm_detector.gdf



19.6 16-channel, 1-bit MUX

```
SUBDESIGN mux16chan      -- AHDL solution
(
    d[15..0]      :INPUT;
    sel[3..0]      :INPUT;
    Y              :OUTPUT;
)

BEGIN
    CASE sel[] IS          -- select input channel
        WHEN 0 => Y = d[0];
        WHEN 1 => Y = d[1];
        WHEN 2 => Y = d[2];
        WHEN 3 => Y = d[3];
        WHEN 4 => Y = d[4];
        WHEN 5 => Y = d[5];
        WHEN 6 => Y = d[6];
        WHEN 7 => Y = d[7];
        WHEN 8 => Y = d[8];
        WHEN 9 => Y = d[9];
        WHEN 10 => Y = d[10];
        WHEN 11 => Y = d[11];
        WHEN 12 => Y = d[12];
        WHEN 13 => Y = d[13];
        WHEN 14 => Y = d[14];
        WHEN 15 => Y = d[15];
    END CASE;
END;
```

```

ENTITY mux16chan IS          -- VHDL solution
PORT (
    d           : IN BIT_VECTOR (15 DOWNTO 0);
    sel         : IN INTEGER RANGE 0 TO 15;
    y           : OUT BIT );
END mux16chan;

ARCHITECTURE vhdl OF mux16chan IS
BEGIN

PROCESS (d, sel)           -- monitor inputs
BEGIN

CASE sel IS                -- select input channel
    WHEN 0 => y <= d(0);
    WHEN 1 => y <= d(1);
    WHEN 2 => y <= d(2);
    WHEN 3 => y <= d(3);
    WHEN 4 => y <= d(4);
    WHEN 5 => y <= d(5);
    WHEN 6 => y <= d(6);
    WHEN 7 => y <= d(7);
    WHEN 8 => y <= d(8);
    WHEN 9 => y <= d(9);
    WHEN 10 => y <= d(10);
    WHEN 11 => y <= d(11);
    WHEN 12 => y <= d(12);
    WHEN 13 => y <= d(13);
    WHEN 14 => y <= d(14);
    WHEN 15 => y <= d(15);
END CASE;

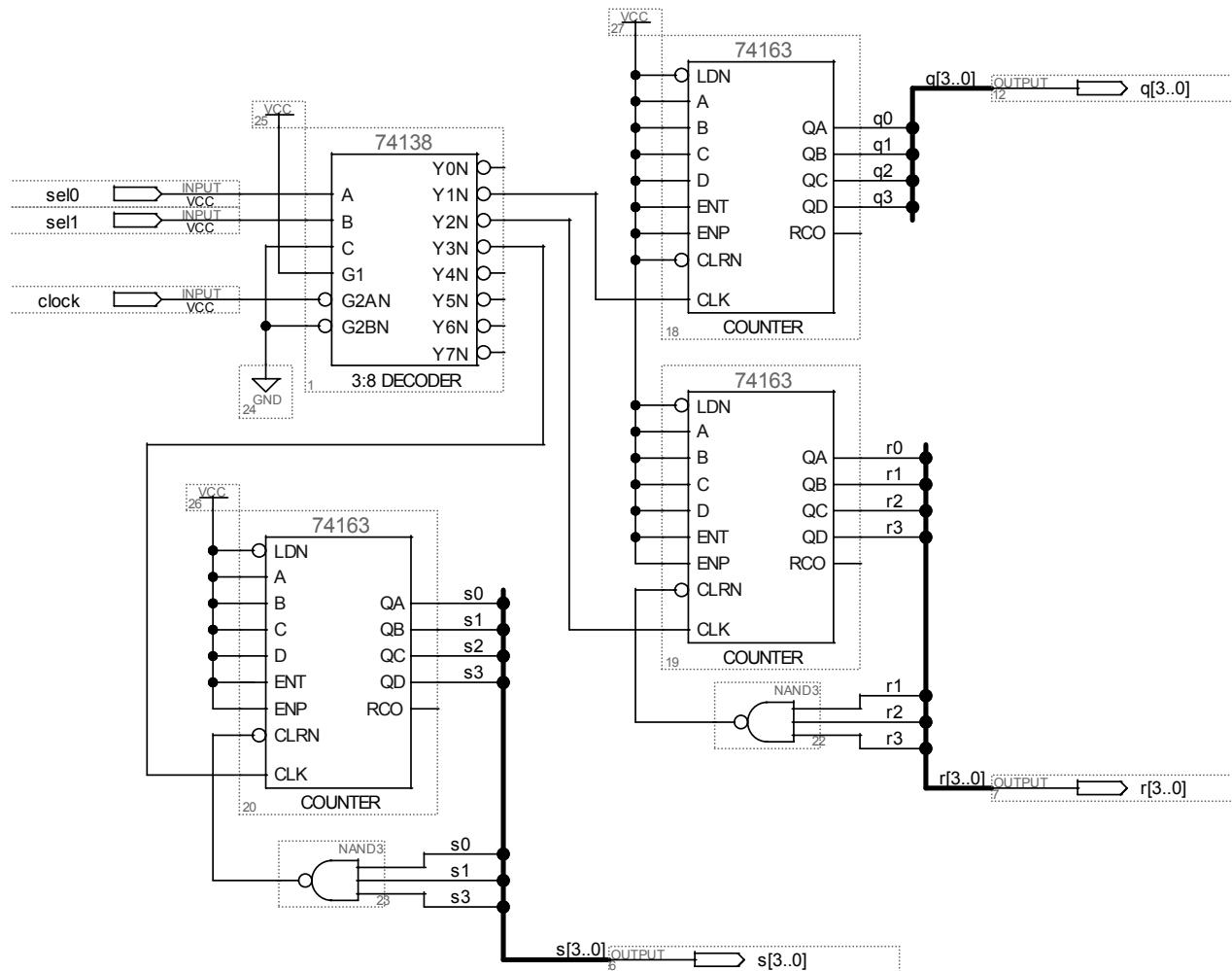
END PROCESS;
END vhdl;

```

Unit 20 Demultiplexers

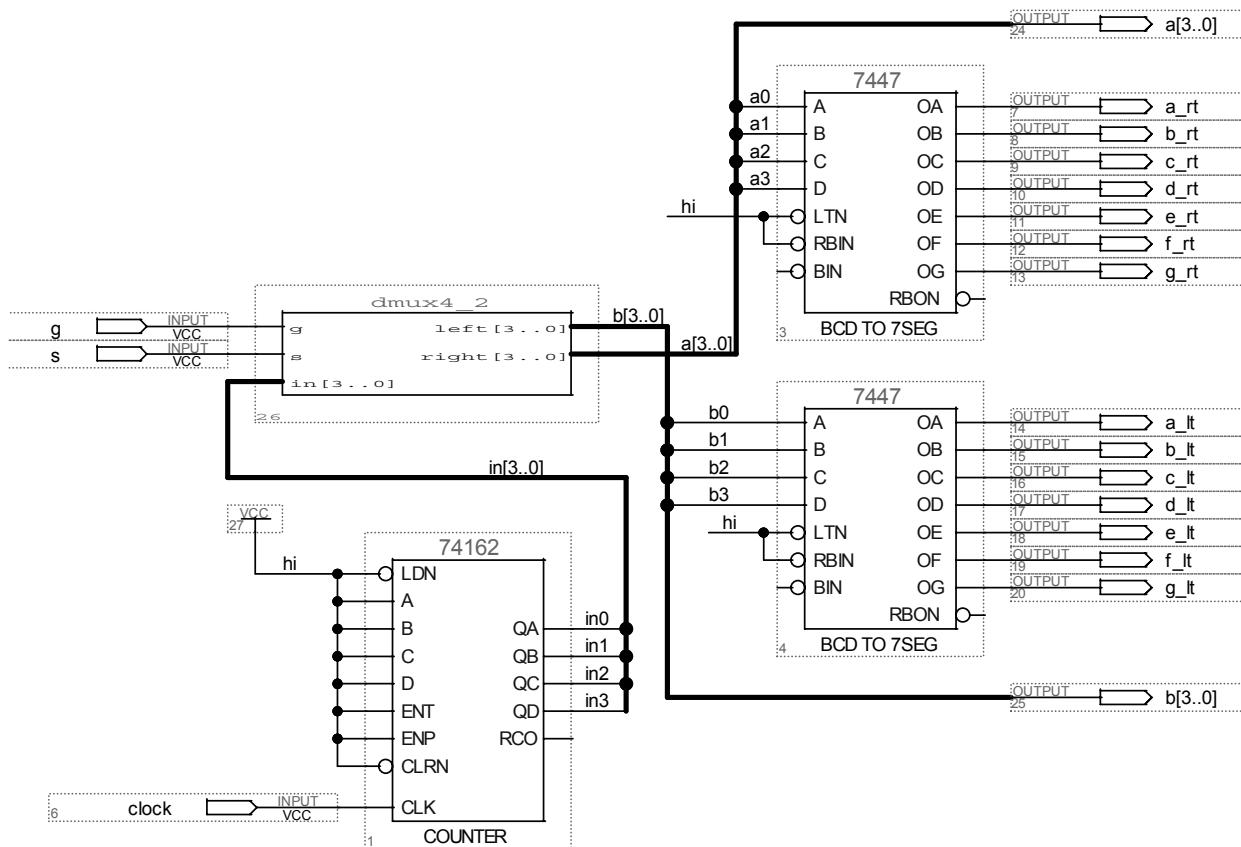
20.1 Clock DMUX

`clock_dmux.gdf`



20.2 BCD counter demultiplexer

dmux_display.gdf



```

SUBDESIGN dmux4_2          -- AHDL solution
(
    g, s, in[3..0]           :INPUT;
    left[3..0], right[3..0]   :OUTPUT;
)

BEGIN
    DEFAULTS               -- 1111 blanks 7447 display
        left[] = H"F";
        right[] = H"F";
    END DEFAULTS;

    IF !g THEN             -- active-low enable
        CASE s IS          -- select output channel
            WHEN 0 => left[] = in[];
            WHEN 1 => right[] = in[];
        END CASE;
    END IF;
END;

```

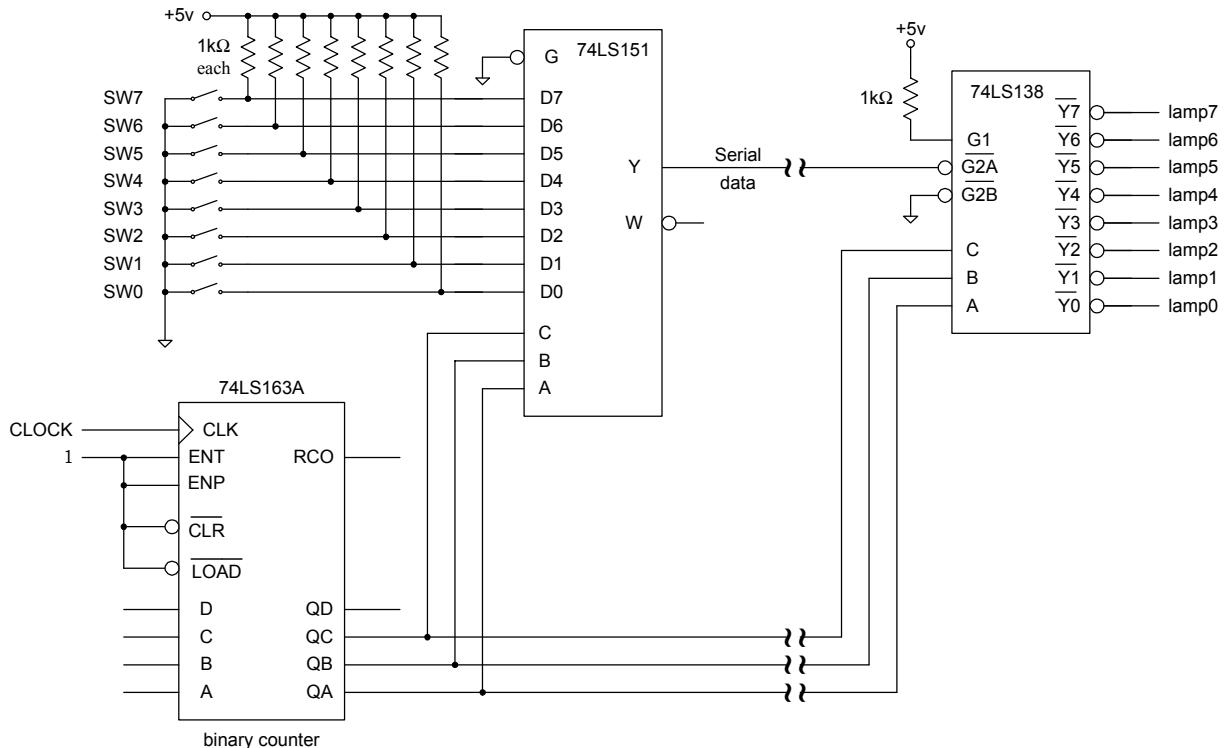
```

ENTITY dmux4_2 IS          -- VHDL solution
PORT ( g, s                 : IN BIT;
        input                : IN BIT_VECTOR (3 DOWNTO 0);
        left, right           : OUT BIT_VECTOR (3 DOWNTO 0) );
END dmux4_2;

ARCHITECTURE vhdl OF dmux4_2 IS
BEGIN
    PROCESS (g, s, input)
    BEGIN
        IF  g = '1'  THEN          -- disabled
            left <= "1111";   right <= "1111";
        ELSIF s = '0'  THEN       -- left
            left <= input;     right <= "1111";
        ELSE                      -- right
            left <= "1111";   right <= input;
        END IF;
    END PROCESS;
END vhdl;

```

20.3 Alarm indicator



20.4 8-channel, 1-bit DMUX

```
SUBDESIGN demux_ahdl      -- AHDL solution
(
    sel[2..0]      :INPUT;
    data, en       :INPUT;
    y[7..0]        :OUTPUT;
)

BEGIN
    DEFAULTS
        y[] = B"00000000";
                    -- inactive outputs are low
    END DEFAULTS;

    IF en THEN          -- active-high enable
        CASE sel[] IS   -- select output channel
            WHEN 0 => y[0] = data;
            WHEN 1 => y[1] = data;
            WHEN 2 => y[2] = data;
            WHEN 3 => y[3] = data;
            WHEN 4 => y[4] = data;
            WHEN 5 => y[5] = data;
            WHEN 6 => y[6] = data;
            WHEN 7 => y[7] = data;
        END CASE;
    END IF;
END;
```

```

ENTITY demux_vhdl IS          -- VHDL solution
PORT (
    sel           :IN BIT_VECTOR (2 DOWNTO 0);
    data, en      :IN BIT;
    y             :OUT BIT_VECTOR (7 DOWNTO 0)
);
END demux_vhdl;

ARCHITECTURE vhdl OF demux_vhdl IS
BEGIN
    PROCESS (data, sel, en)
    BEGIN

        IF en = '1' THEN      -- active-high enable

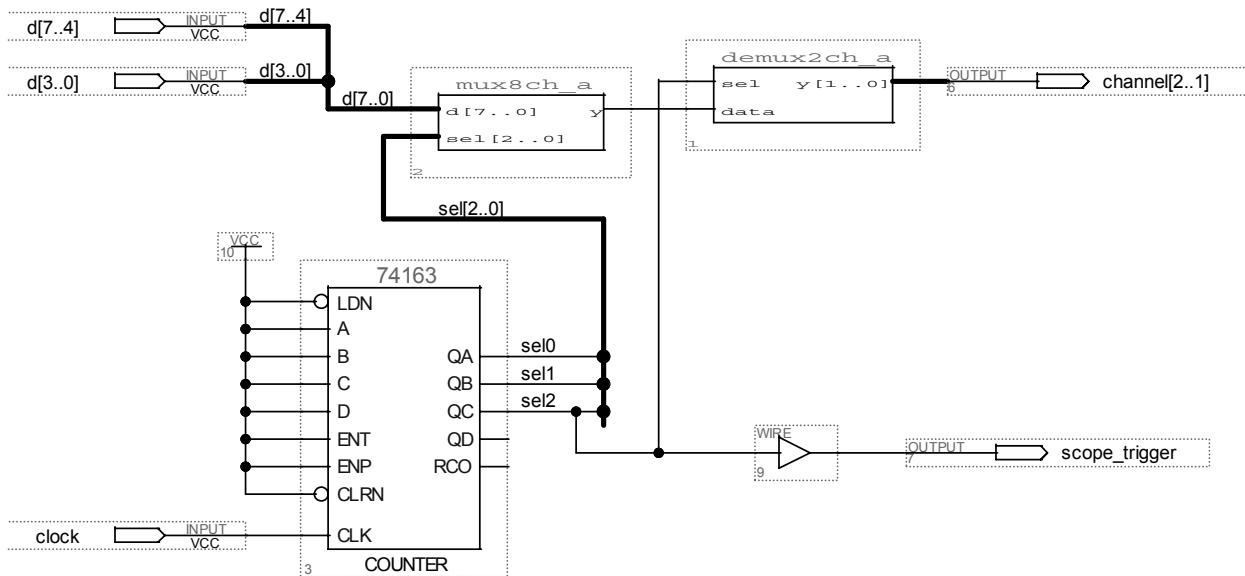
            CASE sel IS      -- select output channel
                WHEN "000" => y <= "0000000" & data;
                -- concatenate output bit vector
                WHEN "001" => y <= "000000" & data & '0';
                WHEN "010" => y <= "00000" & data & "00";
                WHEN "011" => y <= "0000" & data & "000";
                WHEN "100" => y <= "000" & data & "0000";
                WHEN "101" => y <= "00" & data & "00000";
                WHEN "110" => y <= '0' & data & "000000";
                WHEN "111" => y <= data & "0000000";
            END CASE;

        ELSE      y <= "00000000";      -- disabled output
        END IF;

    END PROCESS;
END vhdl;

```

20.5 2-channel data transmitter

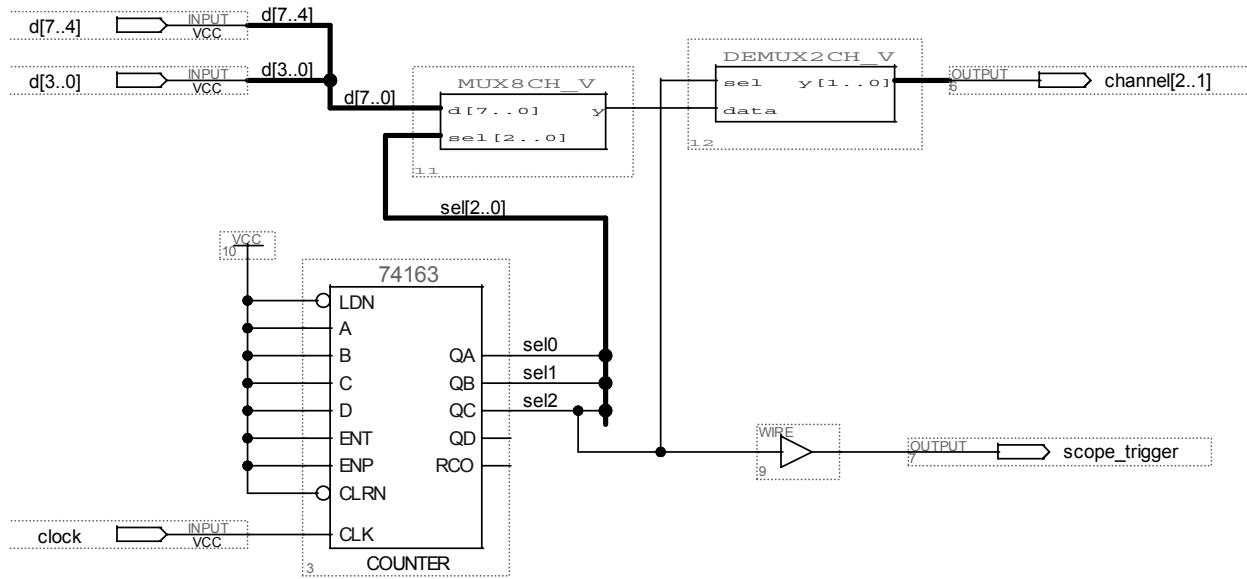


```
SUBDESIGN      mux8ch_a
(
    d [7..0]          : INPUT;
    sel [2..0]         : INPUT;
    Y                  : OUTPUT;
)
BEGIN
    CASE sel [] IS
        WHEN 0 => Y = d[0];
        WHEN 1 => Y = d[1];
        WHEN 2 => Y = d[2];
        WHEN 3 => Y = d[3];
        WHEN 4 => Y = d[4];
        WHEN 5 => Y = d[5];
        WHEN 6 => Y = d[6];
        WHEN 7 => Y = d[7];
    END CASE;
END;
```

```

SUBDESIGN demux2ch_a
(
    sel, data      :INPUT;
    y[1..0]        :OUTPUT;
)
BEGIN
    DEFAULTS
        y[] = B"00";
    END DEFAULTS;
    CASE sel IS
        WHEN 0      => y[0] = data;
        WHEN 1      => y[1] = data;
    END CASE;
END;

```



```

ENTITY mux8ch_v IS
PORT (
    d           :IN BIT_VECTOR (7 DOWNTO 0);
    sel         :IN INTEGER RANGE 0 TO 7;
    Y           :OUT BIT
);
END mux8ch_v;
ARCHITECTURE vhdl OF mux8ch_v IS
BEGIN
    PROCESS (d, sel)
    BEGIN
        CASE sel IS
            WHEN 0 => Y <= d(0);
            WHEN 1 => Y <= d(1);
            WHEN 2 => Y <= d(2);
            WHEN 3 => Y <= d(3);
            WHEN 4 => Y <= d(4);
            WHEN 5 => Y <= d(5);
            WHEN 6 => Y <= d(6);
            WHEN 7 => Y <= d(7);
        END CASE;
    END PROCESS;
END vhdl;

```

```

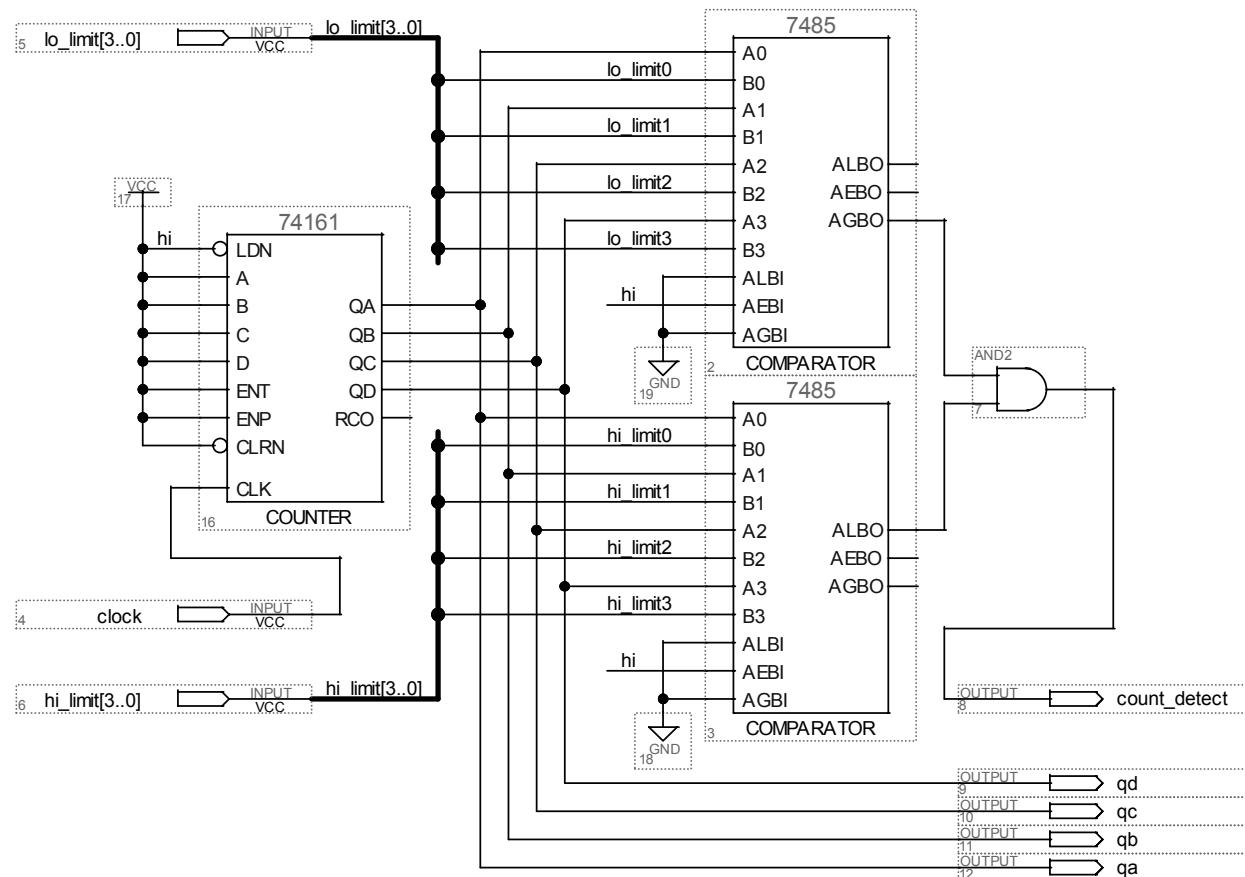
ENTITY demux2ch_v IS
PORT (
    sel, data      :IN BIT;
    Y             :OUT BIT_VECTOR (1 DOWNTO 0)
);
END demux2ch_v;
ARCHITECTURE vhdl OF demux2ch_v IS
BEGIN
    PROCESS (sel, data)
    BEGIN
        CASE sel IS
            WHEN '0' => Y <= ('0' & data);
            WHEN '1' => Y <= (data & '0');
        END CASE;
    END PROCESS;
END vhdl;

```

Unit 21 Magnitude Comparators

21.1 Count detector

count_detect.gdf



21.2 Adjustable range detector

```
SUBDESIGN range_detect          -- AHDL solution
(
    value[4..0], r[1..0]           :INPUT;
    ltr, gtr, rng                 :OUTPUT;
)

VARIABLE
    lo_value[4..0]                :node;
    hi_value[4..0]                :node;

BEGIN
    DEFAULTS
        ltr = GND;
        gtr = GND;
        rng = GND;
    END DEFAULTS;

    CASE r[] IS          -- set range limits
        WHEN 0 => lo_value[] = 16; hi_value[] = 16;
        WHEN 1 => lo_value[] = 15; hi_value[] = 17;
        WHEN 2 => lo_value[] = 14; hi_value[] = 18;
        WHEN 3 => lo_value[] = 13; hi_value[] = 19;
    END CASE;

        -- determine active output
    IF      value[] < lo_value[] THEN ltr = VCC;
    ELSIF  value[] > hi_value[] THEN gtr = VCC;
    ELSE                           rng = VCC;
    END IF;

END;
```

```

ENTITY range_detect IS
PORT (    value          : IN INTEGER RANGE 0 TO 31;
           r             : IN BIT_VECTOR (1 DOWNTO 0);
           ltr, gtr, rng : OUT BIT      );
END range_detect;

ARCHITECTURE vhdl OF range_detect IS
SIGNAL    lo_value        : INTEGER RANGE 0 TO 31;
SIGNAL    hi_value        : INTEGER RANGE 0 TO 31;
BEGIN

    PROCESS (value, r)
    BEGIN

        CASE r IS
            WHEN "00" => lo_value <= 16; hi_value <= 16;
            WHEN "01" => lo_value <= 15; hi_value <= 17;
            WHEN "10" => lo_value <= 14; hi_value <= 18;
            WHEN "11" => lo_value <= 13; hi_value <= 19;
        END CASE;

        -- determine active output
        IF      value < lo_value  THEN
            ltr <= '1'; gtr <= '0'; rng <= '0';

        ELSIF   value > hi_value THEN
            ltr <= '0'; gtr <= '1'; rng <= '0';

        ELSE
            ltr <= '0'; gtr <= '0'; rng <= '1';
        END IF;

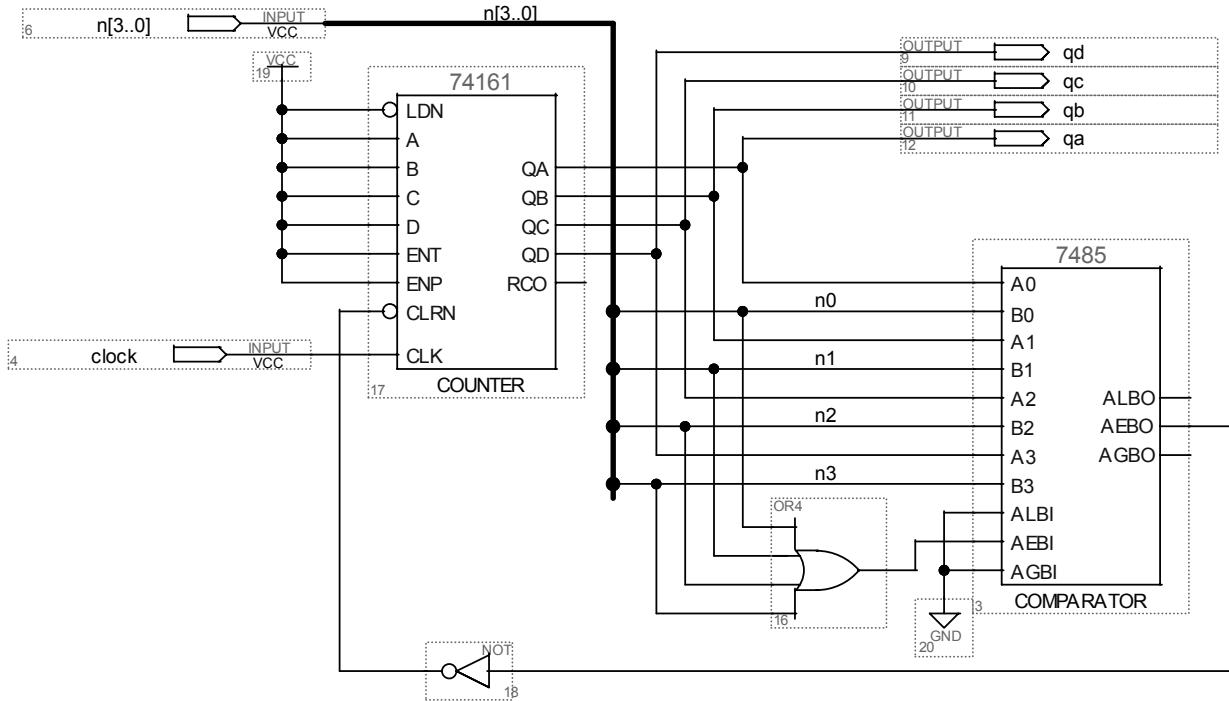
    END PROCESS;

END vhdl;

```

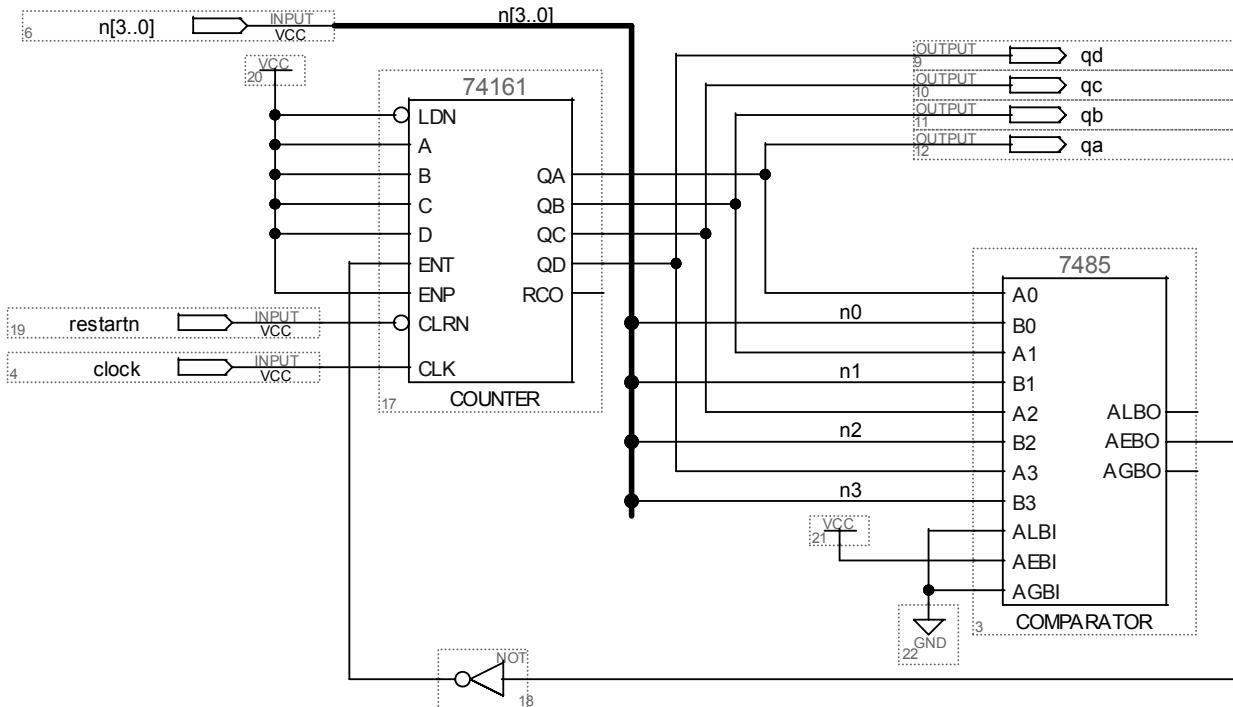
21.3 Variable modulus counter

vary_mod.gdf



21.4 Variable self-stopping counter

vary_stop.gdf



21.5 Number selector

```

SUBDESIGN num_select          -- AHDL solution
(
    f, a[3..0], b[3..0]      :INPUT;
    y[3..0]                  :OUTPUT;
)

VARIABLE
    agtb                   :NODE;      -- buried node

BEGIN

    agtb = a[] > b[];           -- hi if a>b

    y[] = a[] & !(f $ agtb) # b[] & (f $ agtb);
                            -- MUX function to select a or b input
                            -- XNOR selects a & XOR selects b

END;

```

```

ENTITY num_select IS          -- VHDL solution
PORT (   f            : IN BIT;
         a, b        : IN INTEGER RANGE 0 TO 15;
         y            : OUT INTEGER RANGE 0 TO 15 );
END num_select;

ARCHITECTURE vhdl OF num_select IS
SIGNAL    agtb      : BIT;
SIGNAL    sel_a     : BIT;

BEGIN
    agtb <= '1'      WHEN (a > b) ELSE '0';
                    -- hi if a>b

    sel_a <= '1' WHEN ((f XOR agtb) = '0') ELSE '0';
                    -- determine when to select input a

    y <= a WHEN (sel_a = '1') ELSE b;
                    -- MUX function selects a or b input

END vhdl;

```

Unit 22 Digital/Analog and Analog/Digital Conversion

- 22.1 Digital-to-analog converter (see Fig. 22-1)

sample data:

Digital Input	Output Voltage
0000 0000	0.000 V
0000 0001	0.010 V
0000 0010	0.020 V
0000 0100	0.040 V
0000 1000	0.080 V
0001 0000	0.160 V
0010 0000	0.320 V
0100 0000	0.640 V
1000 0000	1.280 V
0000 1111	0.150 V
0011 0010	0.500 V
0110 0100	1.000 V
0111 0000	1.120 V
0111 1111	1.270 V
1010 0000	1.600 V
1100 0000	1.920 V
1100 1000	2.000 V
1111 1111	2.550 V

- 22.2 Analog-to-digital converter (see Fig. 22-2)

WARNING: Do not exceed 5.0 V on analog input!

<u>sample data:</u>	
Analog Input	Digital Output
0.000 V	0000 0000
0.025 V	0000 0001
0.050 V	0000 0010
0.100 V	0000 0101
0.250 V	0000 1100
0.500 V	0001 1001
1.000 V	0011 0011
2.000 V	0110 0110
2.500 V	1000 0000
3.000 V	1001 1001
3.500 V	1011 0011
4.000 V	1100 1100
4.500 V	1110 0110
4.981 V	1111 1111
5.000 V	1111 1111

- 22.3 Free-running ADC with span adjust (see Fig. 22-3)
 WARNING: Do not exceed 5.0 V on analog input!

$$\text{resolution} = 15 \text{ mV}$$

$$\text{full-scale voltage} = 15 \text{ mV} \times 255 = 3.825 \text{ V}$$

$$V_{\text{REF}} = 256 \times 15 \text{ mV} = 3.84 \text{ V}$$

$$\text{adjust pot to } V_{\text{REF}}/2 = 3.84 \text{ V} \div 2 = 1.92 \text{ V}$$

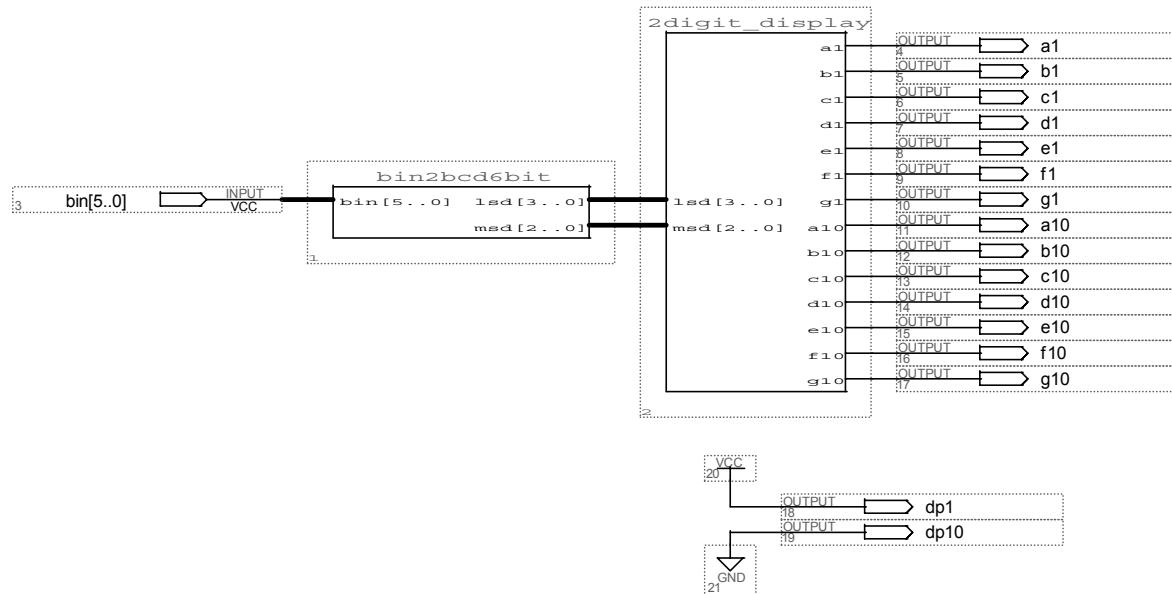
$$[R_2/(R_1+R_2)][5 \text{ V}] = 0.5 \text{ V} \rightarrow \text{use } R_2=1.1 \text{ k}\Omega \text{ & } R_1 = 10 \text{ k}\Omega$$

sample data:

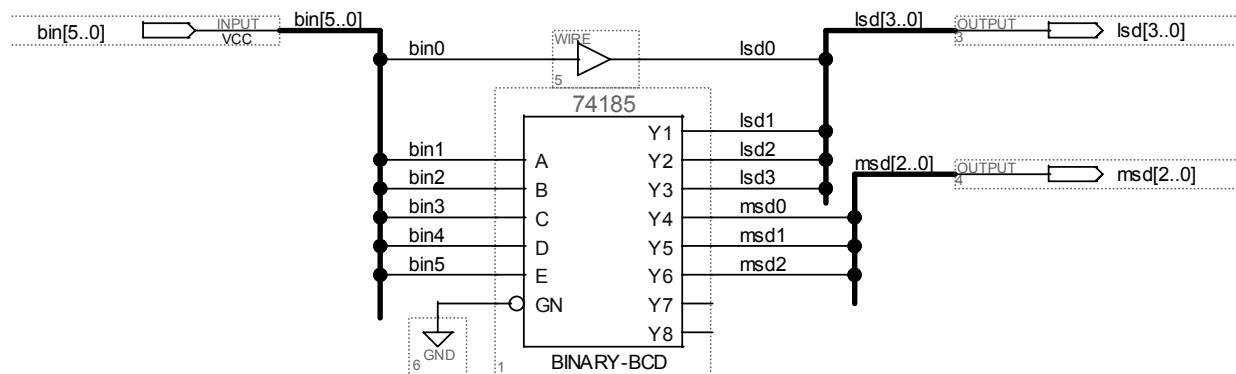
Analog Input	Digital Output
0.000 V	0000 0000
0.100 V	0000 0000
0.250 V	0000 0000
0.400 V	0000 0000
0.500 V	0000 0000
0.520 V	0000 0001
0.540 V	0000 0010
0.570 V	0000 0100
0.630 V	0000 1000
0.800 V	0001 0100
1.000 V	0010 0001
1.500 V	0100 0010
1.750 V	0101 0011
2.000 V	0110 0100
2.600 V	1000 1100
3.000 V	1010 0110
3.400 V	1100 0001
4.000 V	1110 1001
4.325 V	1111 1111
4.500 V	1111 1111

22.4 Digital voltmeter

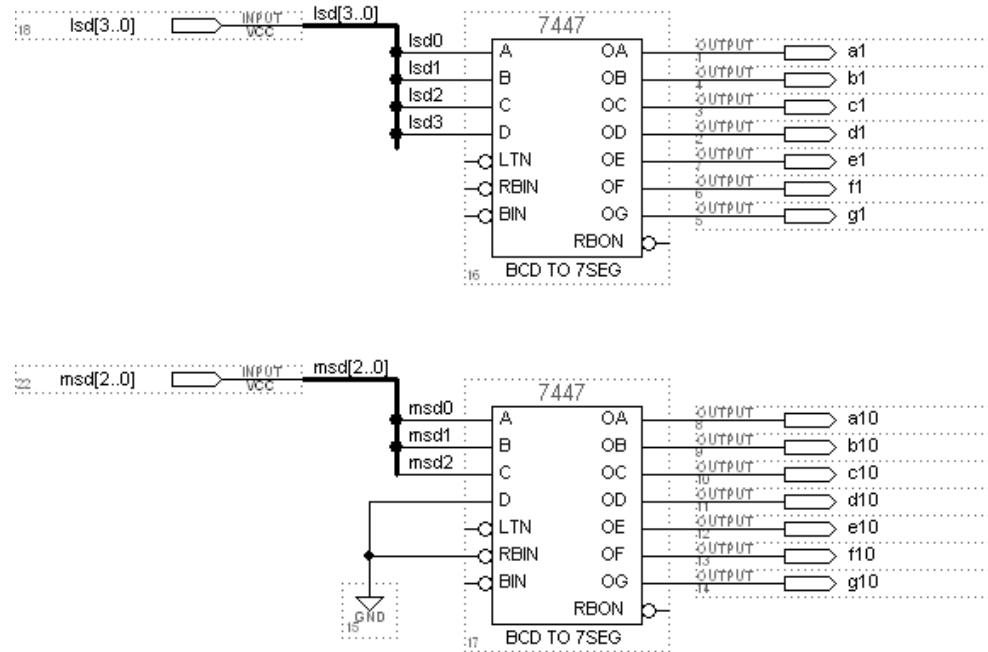
voltmeter.gdf



bin2bcd6bit.gdf

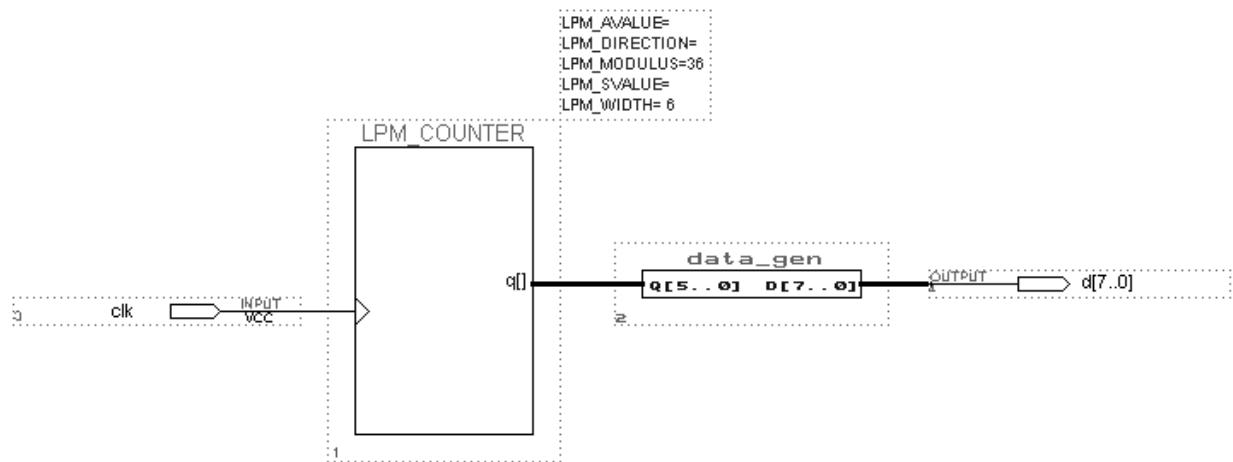


2digit_display.gdf



22.5 Digitized sine-wave generator

sine_wave.gdf



```
-- AHDL solution
SUBDESIGN data_gen
(
    q[5..0]      :INPUT;
    d[7..0]      :OUTPUT;
)

BEGIN
-- D/A input values at
-- 10 degree increments

    TABLE
        q[]  =>  d[];
        0    =>  150;
        1    =>  167;
        2    =>  184;
        3    =>  200;
        4    =>  214;
        5    =>  227;
        6    =>  237;
        7    =>  244;
        8    =>  248;
        9    =>  250;
        10   =>  248;
        11   =>  244;
        12   =>  237;
```

13	=>	227;
14	=>	214;
15	=>	200;
16	=>	184;
17	=>	167;
18	=>	150;
19	=>	133;
20	=>	116;
21	=>	100;
22	=>	86;
23	=>	73;
24	=>	63;
25	=>	56;
26	=>	52;
27	=>	50;
28	=>	52;
29	=>	56;
30	=>	63;
31	=>	73;
32	=>	86;
33	=>	100;
34	=>	116;
35	=>	133;

```
END TABLE;
END;
```

```

ENTITY data_gen IS          -- VHDL solution
PORT (      q           : IN INTEGER RANGE 0 TO 35;
            d           : OUT INTEGER RANGE 0 TO 255      );
END data_gen;

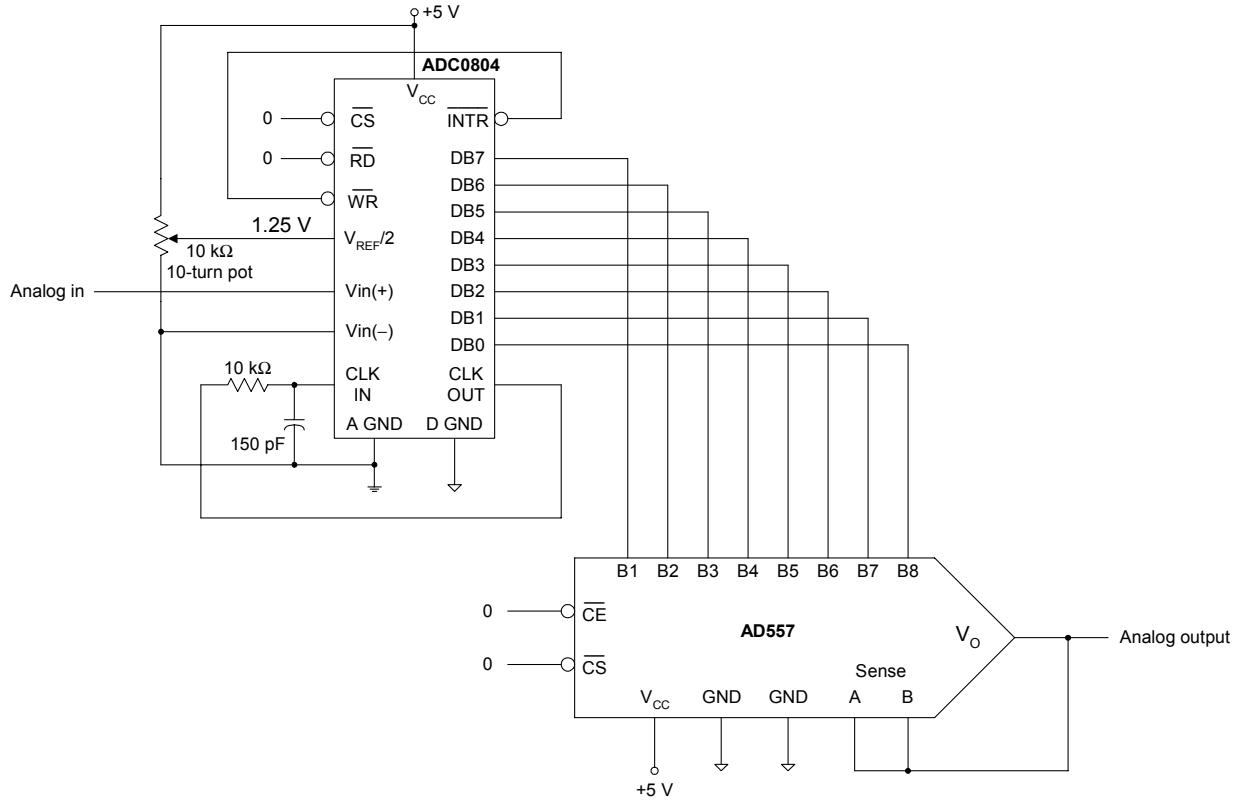
ARCHITECTURE vhdl OF data_gen IS
BEGIN
    -- D/A input values at 10 degree increments

    WITH q      SELECT
        d    <=   150 WHEN 0,
                  167 WHEN 1,
                  184 WHEN 2,
                  200 WHEN 3,
                  214 WHEN 4,
                  227 WHEN 5,
                  237 WHEN 6,
                  244 WHEN 7,
                  248 WHEN 8,
                  250 WHEN 9,
                  248 WHEN 10,
                  244 WHEN 11,
                  237 WHEN 12,
                  227 WHEN 13,
                  214 WHEN 14,
                  200 WHEN 15,
                  184 WHEN 16,
                  167 WHEN 17,
                  150 WHEN 18,
                  133 WHEN 19,
                  116 WHEN 20,
                  100 WHEN 21,
                   86 WHEN 22,
                   73 WHEN 23,
                   63 WHEN 24,
                   56 WHEN 25,
                   52 WHEN 26,
                   50 WHEN 27,
                   52 WHEN 28,
                   56 WHEN 29,
                   63 WHEN 30,
                   73 WHEN 31,
                   86 WHEN 32,
                  100 WHEN 33,
                  116 WHEN 34,
                  133 WHEN 35;

END vhdl;

```

22.6 Reconstructing a digitized signal



$$V_{REF} = 2.5 \text{ V} \rightarrow V_{REF}/2 = 1.25 \text{ V}$$

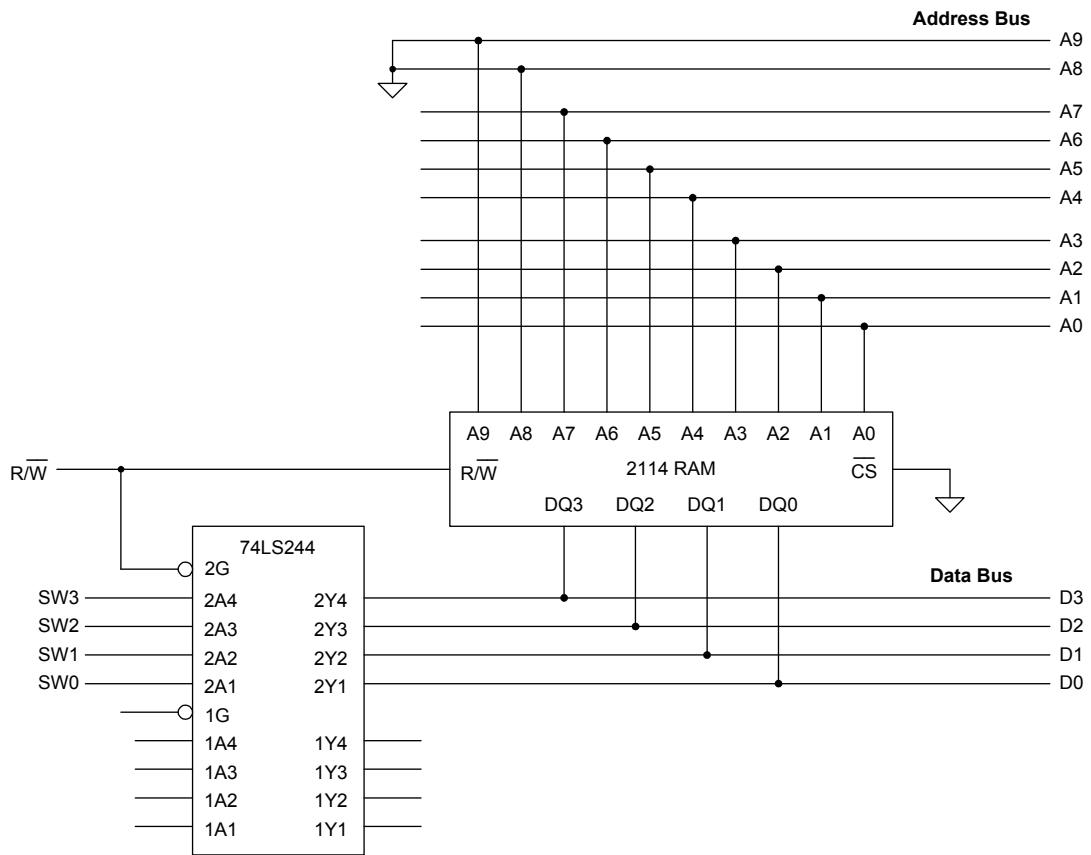
Analog-out should approximate the analog-in voltage

Analog-out will be ragged with staircase output

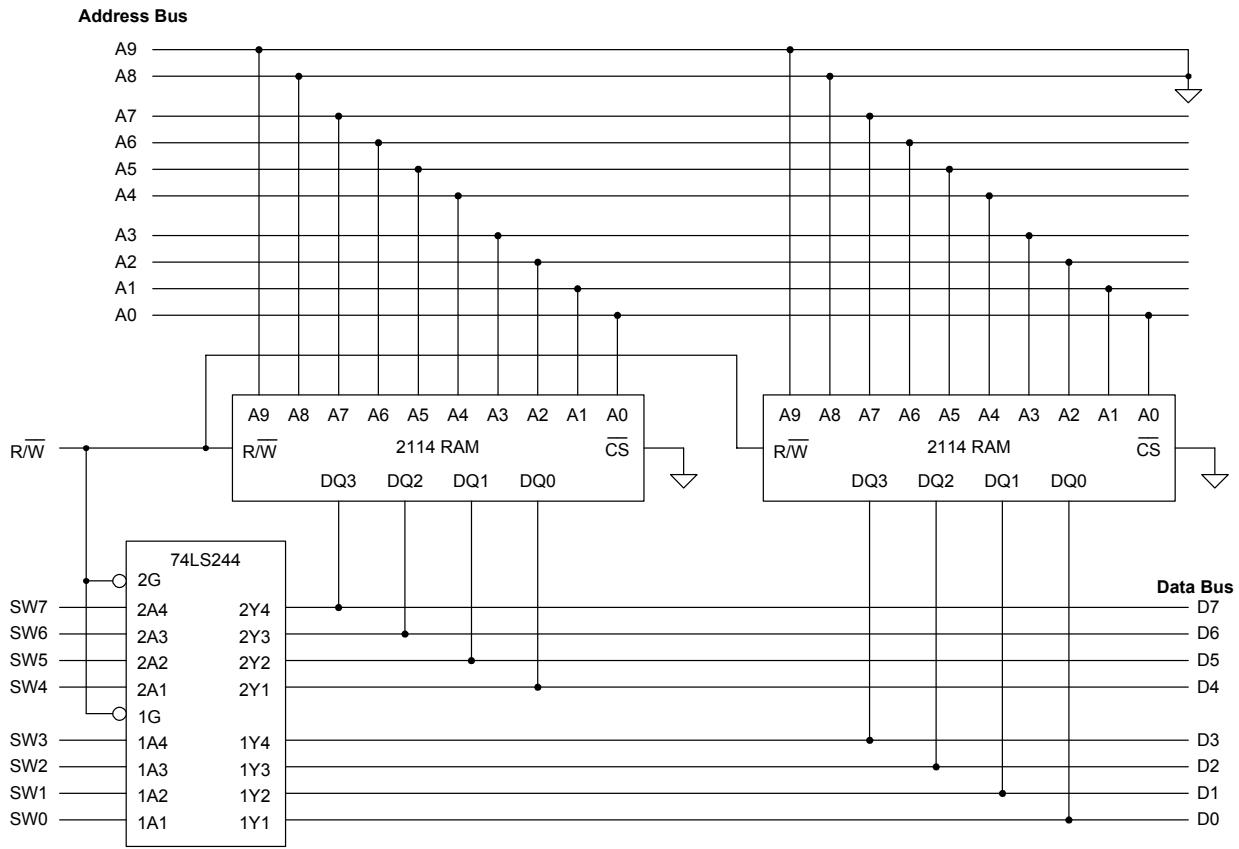
At higher frequencies, analog in will overrun conversion time of A/D converter (aliasing)

Unit 23 Memory Systems

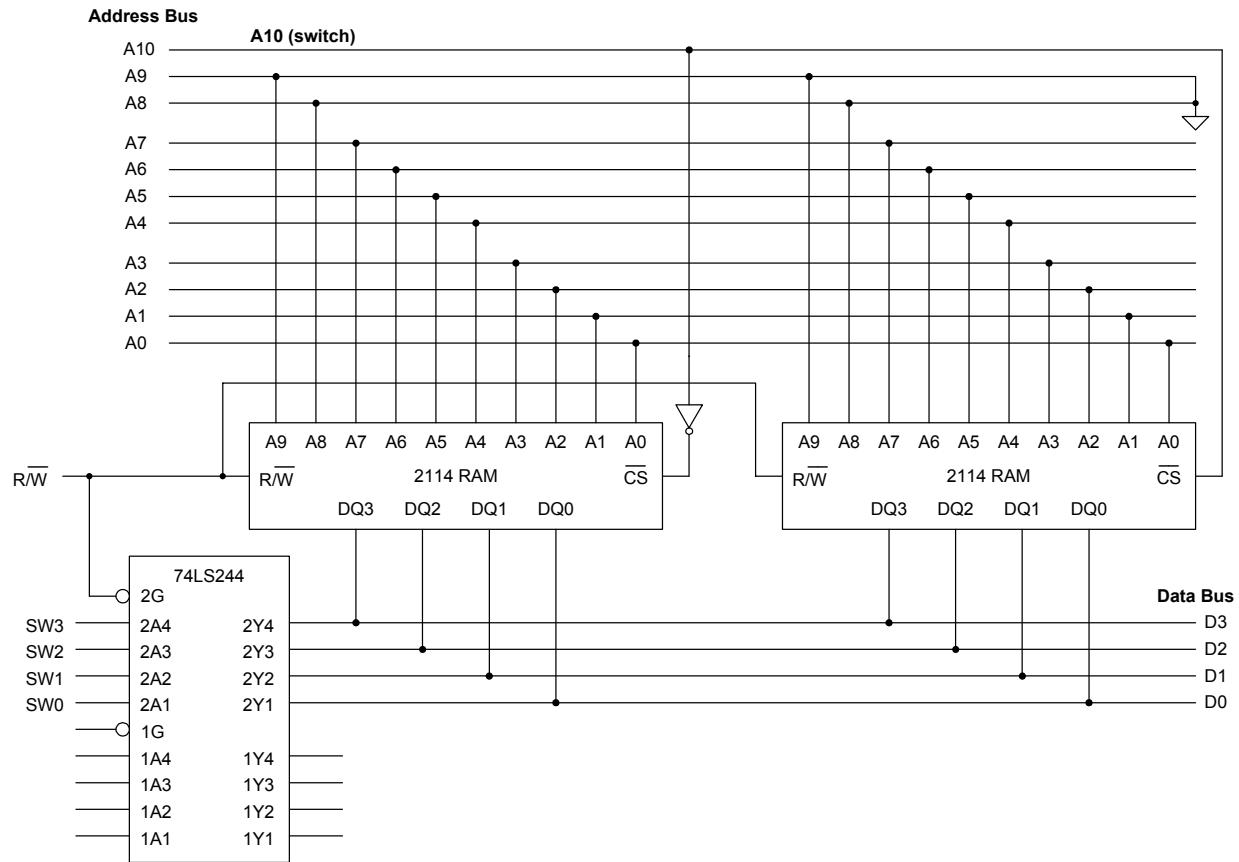
23.1 RAM memory chip



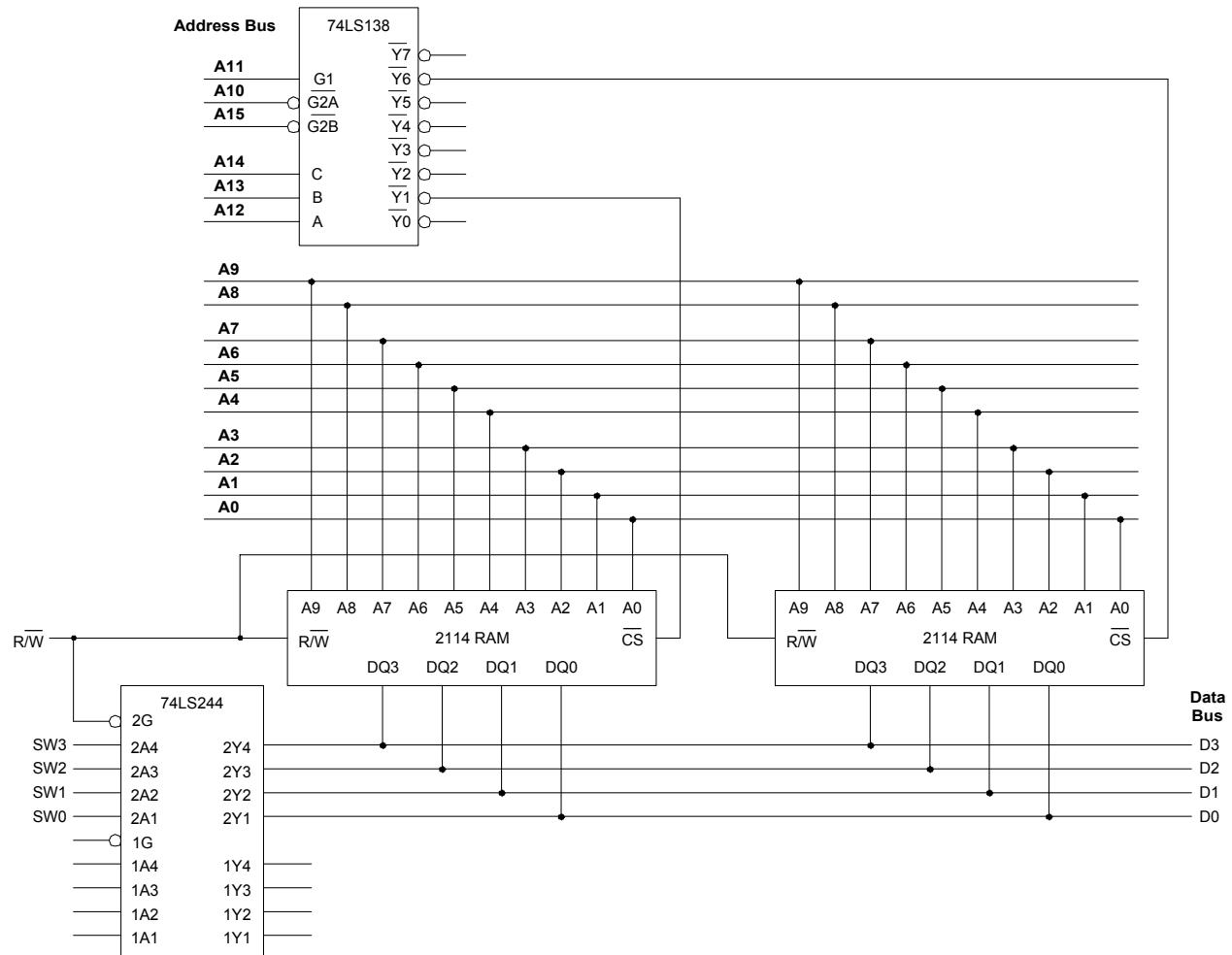
23.2 Memory word-size expansion



23.3 Memory capacity expansion



23.4 Memory address decoding



Solutions to
Lab Manual: A Troubleshooting Approach
Tenth Edition
by Jim C. DeLoach
and Frank J. Ambrosio

Introduction

The purpose of this manual is to provide the instructor with the following:

1. Solutions for procedures not calling for an opinion or observation made by the students
2. Model solutions for some of the procedures and questions calling for observations
3. Solutions to selected Review Questions
4. Suggestions for alternative exercises and laboratory setups

Not all experiments are included in this Solutions Manual. Some are troubleshooting exercises that do not have questions for the student to answer. The experiments without solutions are so designated in the Table of Contents by means of an asterisk.

The model solutions given for some of the procedures and review questions are just that—models. It is hoped that their inclusion will be helpful but not used as a “key” for grading exercises. Some variance in student results is to be expected on these exercises.

The logic symbols in the illustrations were produced with an electronic drawing program, DesignWorks from Capilano Computing. The symbols for the gates are (or are close to) the actual ANSI symbols. The labels used in MSI device printouts and logic symbols should also be recognizable. In most of the logic diagrams, the use of “1” for +5V (HIGH or V_{cc}) and “0” for 0V (LOW or GND) is liberal.

Jim C. DeLoach

EXPERIMENT 1

PRELIMINARY CONCEPTS

d) As with all set-ups requiring observations of digital signals with oscilloscopes, encourage the student to experiment with different pulse frequencies and pulse widths. For your particular lab model, you may want to choose these values differently than those given in the lab manual.

e) Review:

1. digital, Analog, digital
2. 0 to +0.8, unlighted
3. +2.0 to +5.0 volts
4. 1010_2
5. 6
6. 37.5

EXPERIMENT 2
LOGIC GATES I: OR,
AND, AND NOT

m) Review:

1. all inputs are LOW (binary 0)
2. LOW
3. the complement of (the opposite state from)
4. V_{CC}
5. HIGH (binary 1)

EXPERIMENT 3
TROUBLESHOOTING OR, AND and NOT

Model Solutions for Experiment 3:

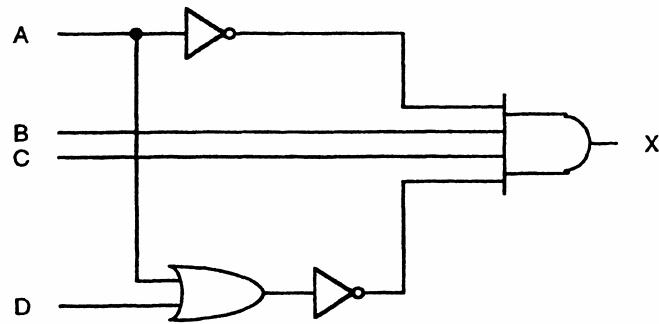
(Lab Manual Page 20) The logic probe should indicate a constant LOW since the LOW output of gate 2 will pull the output of gate 3 LOW no matter what the level of switch B is. Thus both inputs of gate 1 are LOW, driving its output LOW.

(Lab Manual Page 22) The logic probe should indicate a constant LOW since the LOW output of gate 2 will pull the output of gate 3 LOW no matter what the level of switch B is. Thus both inputs of gate 1 are LOW, driving its output LOW.

(Lab Manual Page 25) The logic probe should remain LOW no matter what level switch A is set to. This is because the output of gate 2 is LOW, pulling the output of gate 1 LOW.

EXPERIMENT 4
BASIC COMBINATORIAL
CIRCUITS

c)



Model Circuit for Procedure c

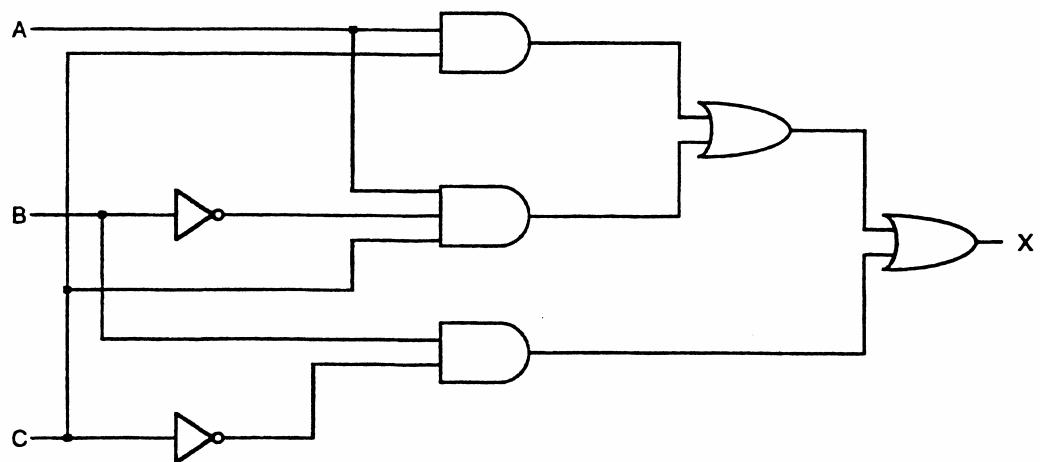
Model Solution:

Toggle Switches				Outputs		
A	B	C	D	$\bar{A}BC$	$\bar{A} + \bar{D}$	$\bar{A}BC(\bar{A} + \bar{D})$
0	0	0	0	0	1	0
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	0	1	1	0	0	0
0	1	0	0	0	1	0
0	1	0	1	0	0	0
0	1	1	0	1	1	1
0	1	1	1	1	0	0
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	0	0	0
1	1	0	1	0	0	0
1	1	1	0	1	0	0
1	1	1	1	1	0	0

Table 4.3

g) Review:

1. AND, OR
2. 3, AND
3. truth table
4. evaluating
5. output
6. static
- 7.



EXPERIMENT 5
LOGIC GATES II: NOR AND NAND

(NO SOLUTIONS)

EXPERIMENT 6
TROUBLESHOOTING NOR and NAND Gates

Model Solutions for Experiment 6:

(Lab Manual Page 41) The logic probe should indicate a constant HIGH since the LOW output of gate 2 will pull the output of gate 3 LOW no matter what the level of switch B is. Thus both inputs of gate 1 will be LOW, driving its output HIGH.

(Lab Manual Page 44) The logic probe should indicate a constant HIGH since the LOW output of gate 2 will pull the output of gate 3 LOW no matter what the level of switch B is. Thus the input of gate 1 connected to the outputs of gate 3 and gate 2 will be LOW, driving the output of gate 1 HIGH.

EXPERIMENT 7

BOOLEAN THEOREMS

k) Review:

1. $X * 0 = 0$
2. $X + 1 = 1$
3. constant LOW (stuck-LOW), $X * 0 = 0$
4. HIGH

EXPERIMENT 8
SIMPLIFICATION
USING BOOLEAN THEOREMS

a) Model answer: $\bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}\bar{B}D$

b)

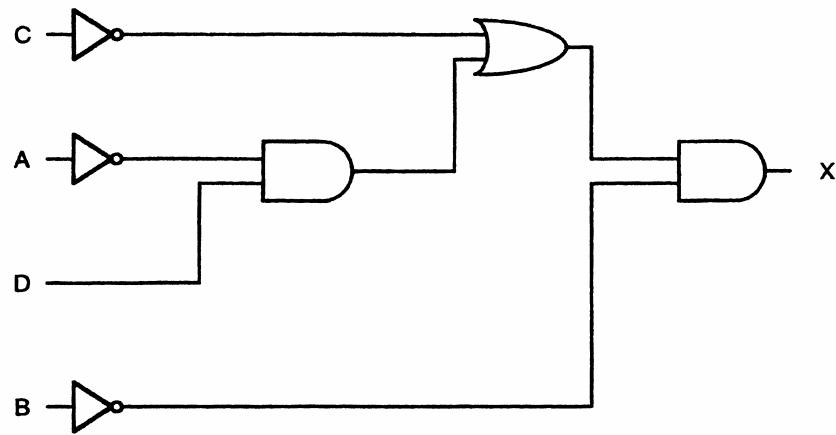
A	Inputs			Output
	B	C	D	
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Table 8.1

e) Model solution:

$$\begin{aligned}
 x &= \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}\bar{B}D \\
 &= \bar{B}(\bar{A}\bar{C} + A\bar{C} + \bar{A}D) && \text{Theorem 13} \\
 &= \bar{B}(\bar{C}(\bar{A} + A) + \bar{A}D) && \text{Theorem 1} \\
 &= \bar{B}(\bar{C} + \bar{A}D) && \text{Theorem 8}
 \end{aligned}$$

f) Model solution:

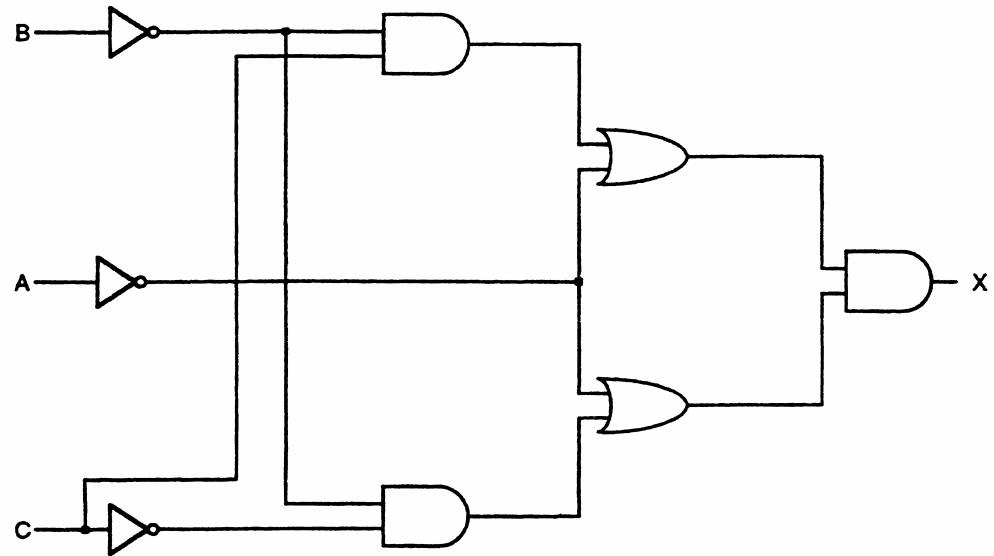


h) Review:

1. Two expressions are equivalent if their truth tables are equivalent. A 3-variable truth table is equivalent to a 4-variable truth table if the output for each combination of the variables in the 3-variable table produces the same output in the 4-variable table regardless of the value of the eliminated variable. This statement can be generalized to cover any number of variables.
2. No. $x = \bar{A}\bar{B}\bar{C} + AD$ and $y = BC + \bar{A}\bar{B}D$ are two examples. Example 4-6 in the text is another.

EXPERIMENT 9
DEMORGAN'S THEOREMS

g) Model solution:



h)

Inputs			Output
A	B	C	$(\bar{A} + \bar{B}C)(\bar{A} + \bar{B}\bar{C})$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Table 9.5

i) \bar{A}

j) Model solution:

$$\begin{aligned}x &= (\overline{A} + \overline{B}C)(\overline{A} + \overline{B}\overline{C}) \\&= \overline{\overline{A}\overline{B}C} \overline{\overline{A}\overline{B}\overline{C}} \\&= \overline{A(\overline{B} + \overline{C})} \overline{A(\overline{B} + \overline{C})} \\&= \overline{(AB + A\overline{C})(AB + AC)} \\&= \overline{AB + A\overline{C} + AB + AC} \\&= \overline{AB + A(\overline{C} + C)} \\&= \overline{AB + A} \\&= \overline{A}\end{aligned}$$

k) Review:

1. OR

2. AND

3. Model solution:

$$AB = \overline{\overline{A}\overline{B}} = \overline{\overline{A} + \overline{B}}$$

4. Model Solution:

$$A + B = \overline{\overline{A} + \overline{B}} = \overline{\overline{A} \bullet \overline{B}}$$

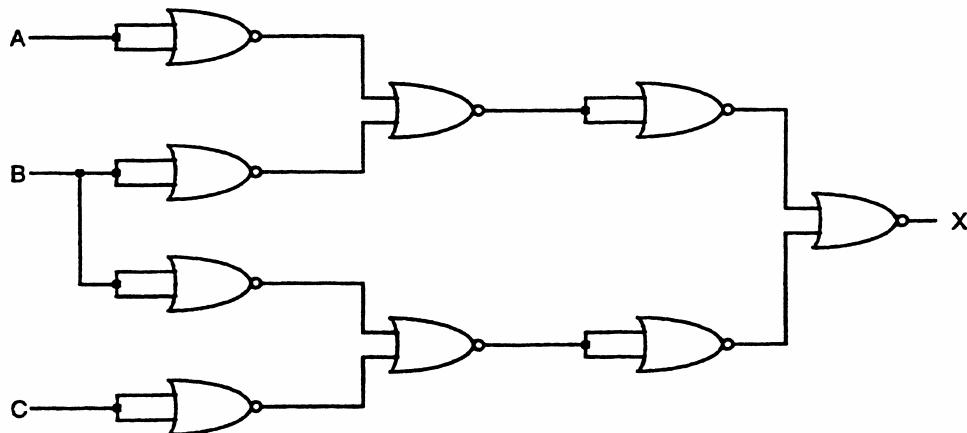
EXPERIMENT 10
THE UNIVERSALITY OF
NAND AND NOR GATES

i)

Inputs			Output
A	B	C	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 10.5

Model solution for circuit using all NORs:



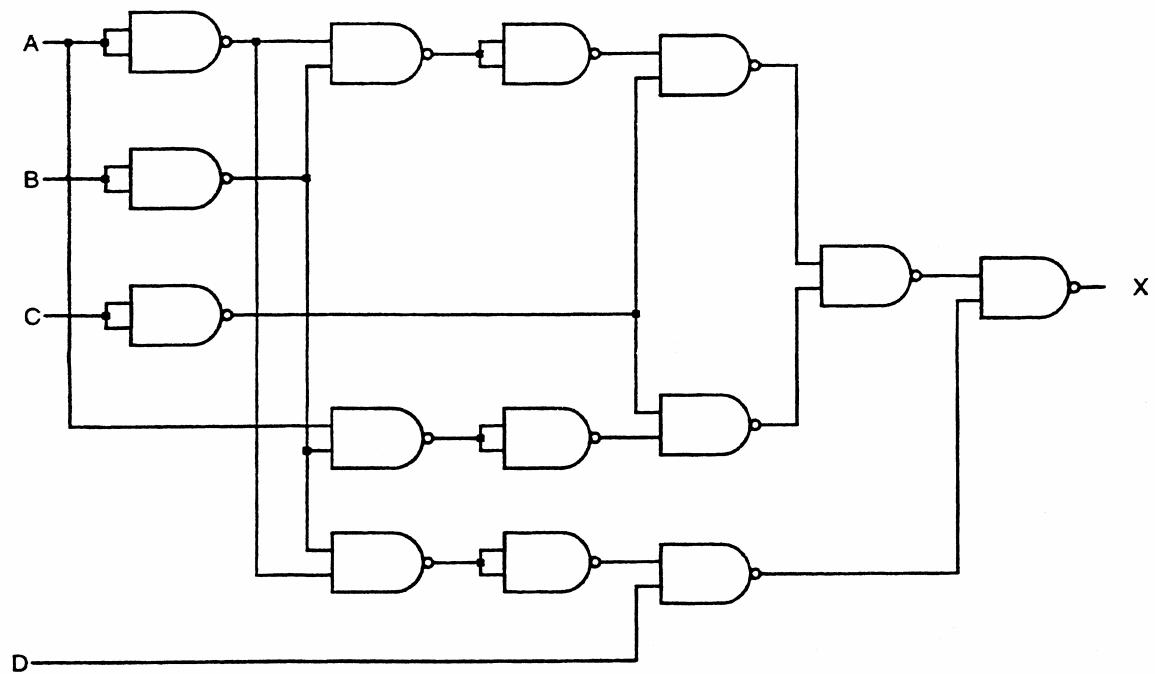
(Note: a much simpler solution can be had using 3-input NORs. This one uses 2-input NORs.)

j)

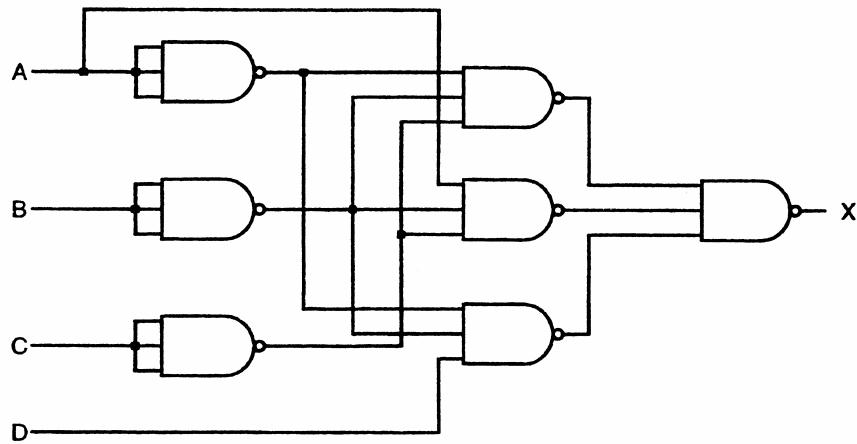
A	Inputs			Output
	B	C	D	
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Table 10.6

Model solution using 2-input NANDs:

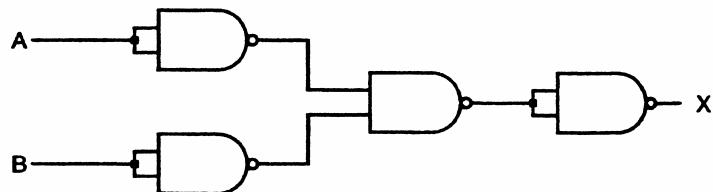


Alternate using 3-input NANDs:

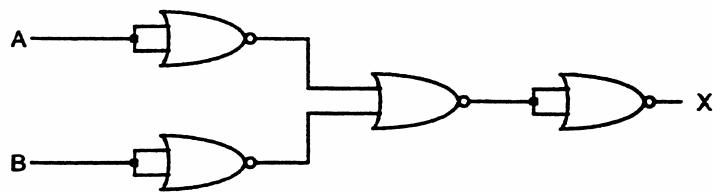


k) Review:

1. Solution using for 2-input NOR using 2-input NANDs:

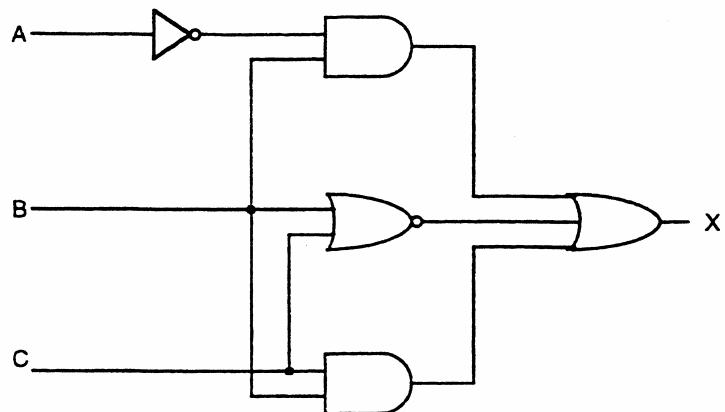


2. Solution for 2-input NAND using 2-input NORs:



EXPERIMENT 11
IMPLEMENTING LOGIC
CIRCUIT DESIGNS

a) Model solution:

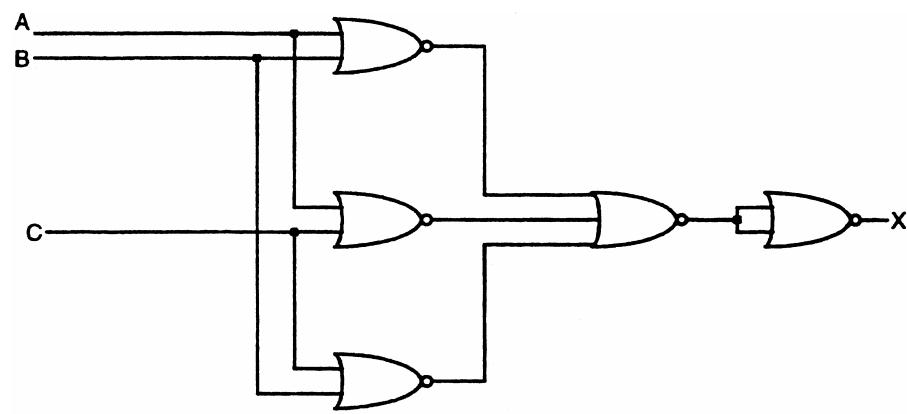


b) Solution : _____

Inputs			Output
A	B	C	x
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Table 11.2

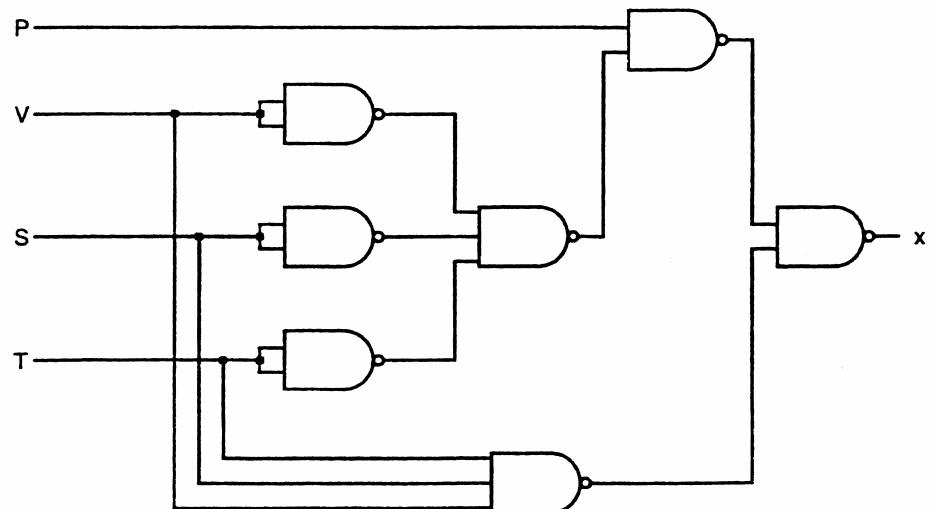
Model solution for circuit:



d) Model Solution:

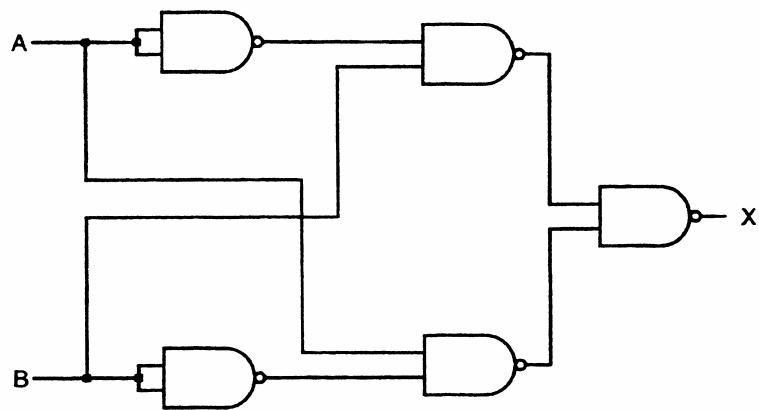
P	Inputs			Output
	V	S	T	
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

(NOTE: For output, 0 = motion failed; 1 = motion carried)



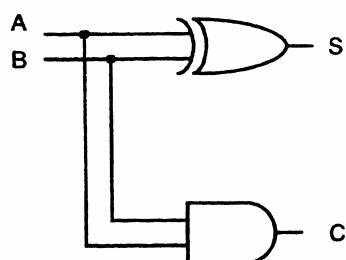
EXPERIMENT 12
EXCLUSIVE-OR AND
EXCLUSIVE-NOR CIRCUITS

- b) Model solution for the sum-of-product EXCLUSIVE-OR using all NANDs:



h) Review:

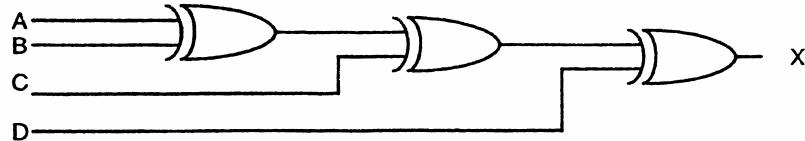
1. different (not equal)
2. the same (equal)
- 3.



4. For example: $X \oplus \bar{Y} = X\bar{\bar{Y}} + \bar{X}\bar{Y} = XY + \bar{X}\bar{Y} = \bar{\bar{X}}\bar{\bar{Y}}$

EXPERIMENT 13
DESIGNING WITH
EXCLUSIVE-OR AND
EXCLUSIVE-NOR CIRCUITS

a)



- c) A 2-input device which produces a HIGH output whenever its two inputs, say A_k and B_k are the same ($A_k = B_k$) is the X-NOR. A 2-input device which produces a HIGH output whenever one of its inputs is greater than the other, say $A_k > B_k$ is a circuit which has the Boolean expression of $A\bar{B}$.

Two three-bit numbers, say $A = A_2A_1A_0$ and $B = B_2B_1B_0$ are equal if and only if $A_2 = B_2$, $A_1 = B_1$, and $A_0 = B_0$. The Boolean equation for this is

$$\overline{(A_2 \oplus B_2)} \overline{(A_1 \oplus B_1)} \overline{(B_0 \oplus B_0)} = 1.$$

Also, $A > B$ if and only if any one of the following holds:

1. $A_2 > B_2$
2. $A_2 = B_2$ and $A_1 > B_1$
3. $A_2 = B_2$ and $A_1 = B_1$ and $A_0 > B_0$

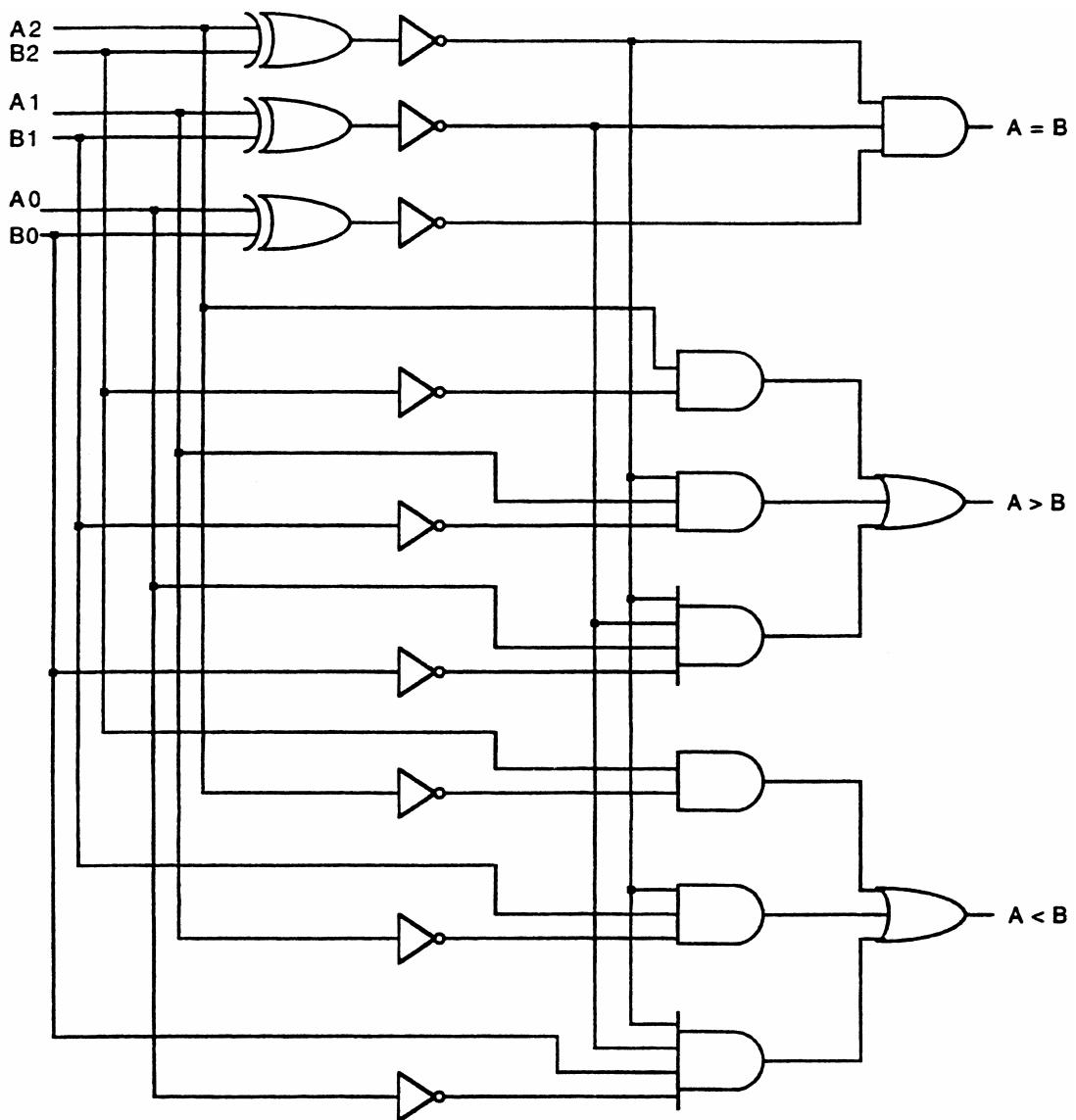
The equation for a device which can detect this is given by

$$A_2 \overline{B_2} + \overline{(A_2 \oplus B_2)}(A_1 \overline{B_1}) + \overline{(A_2 \oplus B_2)}(A_1 \oplus B_1)(A_0 \overline{B_0}) = 1.$$

Similarly, the equation for a device which can detect $B > A$ is given by

$$B_2 \overline{A_2} + \overline{(A_2 \oplus B_2)}(B_1 \overline{A_1}) + \overline{(A_2 \oplus B_2)}(A_1 \oplus B_1)(B_0 \overline{A_0}) = 1.$$

The circuit below is for the 3-bit comparator described by the Boolean equations above:



d) Review:

1. One way would be to invert the output of the even parity circuit.
2. \bar{A} , A.

3. Any expression which contains the form $A\bar{B} + \bar{A}B$ or $\bar{A}\bar{B} + A\bar{B}$ are candidates.
For example:

$$A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}\bar{C} + AB\bar{C} = C(A \oplus B) + \overline{C(A \oplus B)}$$

Further investigation will reveal that this expression can be further simplified using an EX-NOR:

$$C(A \oplus B) + \overline{C(A \oplus B)} = \overline{C(A \oplus B)}$$

EXPERIMENT 14
TROUBLESHOOTING EXCLUSIVE-OR AND
COMBINATORIAL CIRCUITS

(NO SOLUTIONS)

EXPERIMENT 15
FLIP-FLOPS I:
SET/CLEAR LATCHES
AND CLOCKED FLIP-FLOPS

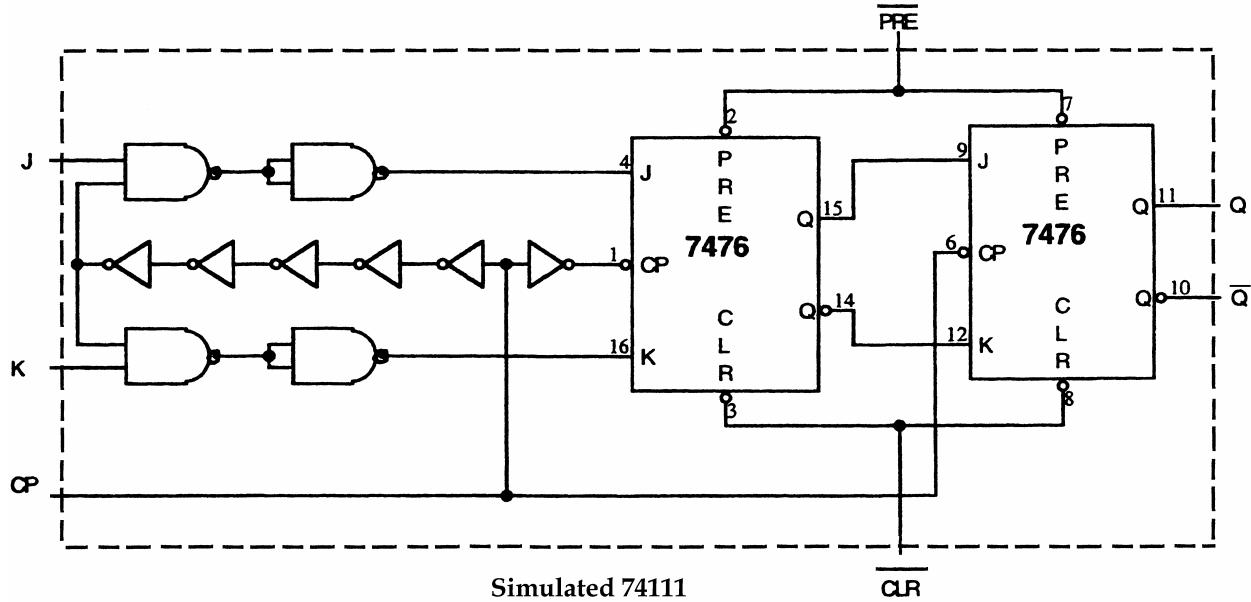
s) Review:

1. S=1 and C=0; S=0 and C=1.
2. S=1 and C=0; S=0 and C=1.
3. Negative-Going Transition (NGT)
4. LOW, operate independently from the clock
5. Positive-Going Transition (PGT)

EXPERIMENT 16
FLIP-FLOPS II:
D LATCH;
MASTER/SLAVE FLIP-FLOPS

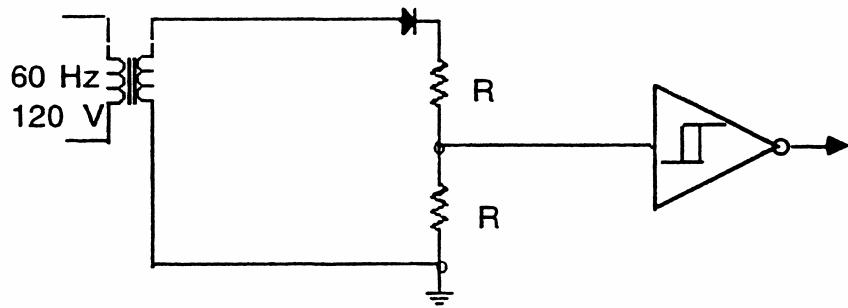
h) Review:

1. HIGH; CLK experiences a NGT.
2. CLK experiences a NGT.
3. Data at the D input of the 7475 FF is continually transferred to Q as long as CLK input is HIGH and is not latched until an NGT is received at CLK. Data at the D input of a 7474 FF is transferred and latched to Q on a PGT at the CLK input.
5. The data lockout feature possessed by some Master/Slave FFs like the 74111 effectively disconnects the J and K inputs from the flip-flop after a very short period of time following the clock's transition to its active level (usually about 20-30 ns). This prevents a glitch occurring after "lockout" from affecting Q.



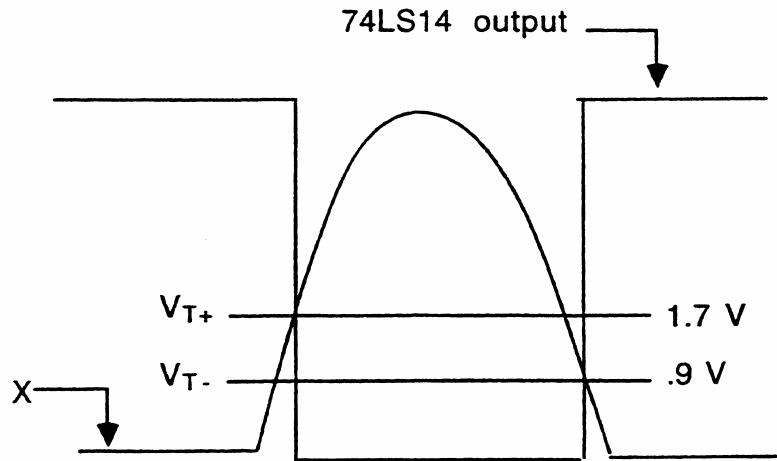
EXPERIMENT 17
SCHMITT TRIGGER,
ONE-SHOTS, AND
ASTABLE
MULTIVIBRATORS

- b) This step is very important since an improper hookup can cost the student an IC and time. An alternative setup is shown in the diagram below:



R must be chosen small enough so that V_{IL} will not be too large.

- c) A typical oscilloscope display:



- d) Typical values: $V_{T+} = 1.7$ volts and $V_{T-} = 0.9$ volts.

- f) 2.31 seconds

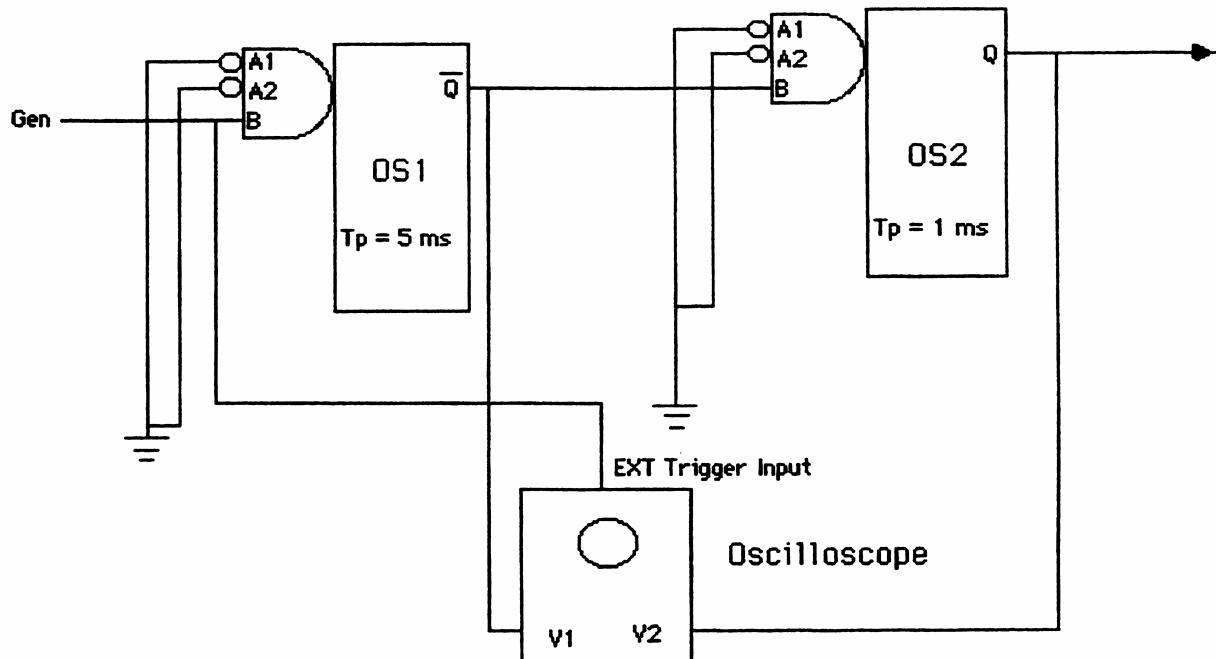
i) Make sure that the student uses a square wave or a digital signal with a duty cycle no greater than about 67%. Also the student should be made aware that differences between computed T_p and observed T_p can be as high as 20%.

When the frequency of the generator is increased slightly, the frequency of the OS output is also increased while T_p remains the same. This means that the duty cycle of the OS output must be increasing. When the OS duty cycle reaches its maximum (about 67% for timing resistors near 2 k-ohms and 90% when R_t is near 40 k-ohms), the OS output will become unstable.

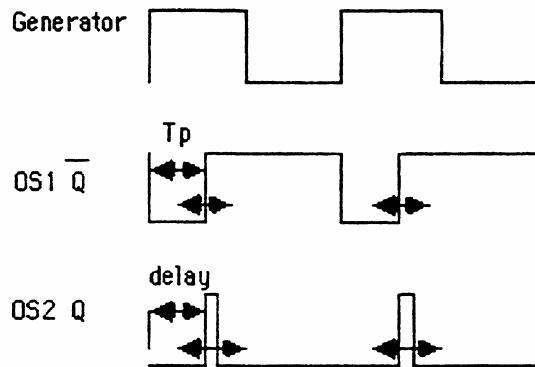
j) For better stability of oscilloscope display, have the student connect his/her set-up as follows:

1) Connect the output of OS2 to one vertical input of an oscilloscope. If a dual trace oscilloscope is available, connect the output of OS1 to the other vertical input.

2) Connect the output of the generator to the external (EXT) input of the oscilloscope. Set the horizontal sweep control to 5 ms.



3) Select negative-edge triggering and adjust the level for a stable display. The display should be similar to the display illustrated below:



Note that the first OS2 pulse occurs several milliseconds after the beginning of the trace. This corresponds to the delay of the OS2 pulse. The arrows on the PGT of OS1 \bar{Q} indicates that the edge is variable according to the value of the timing components. The arrows on the pulses of OS2 Q indicate that the pulse positions vary according to the amount of delay selected.

4) Observe that the OS1 output pulse width (T_p) varies as the potentiometer is adjusted and that the delay of the output pulse of OS2 changes with it. Depending on the timing capacitor selected for OS1, you should have a wide range of delay.

- 1)
 - 1) 4.7 ms
 - 2) 7.9 ms
 - 3) $T = 12.6 \text{ ms}$
 - 4) 79365 Hz
 - 5) duty cycle = 62.7%

o) Review:

1.
 - a) The purpose of the diodes is to limit the input excursions to the 7414 to approximately 0 to 5 volts. When the transformer input exceeds 5 volts, the top diode clamps the input of the 7414 to 5 volts by turning on. The resistor prevents excessive current through the diode. When the transformer falls below 0, the bottom diode turns on and clamps the input to ground.
 - b) Any unipolar signal within the limits of TTL.
2. Use A_1 (or A_2) as the trigger input and tie B and A_2 (or A_1) to HIGH , respectively.
3. 40 kOhms, 1000 mF.
4. $C_T = 7/0.7 \times 10^4 = 0.001 \text{ F} = 1000 \text{ mF}$

5. OS1 produces a negative pulse at Q with width T_p . The PGT of this pulse occurs 5 milliseconds after a PGT of the generator. The OS1 output pulse PGT triggers OS2. The OS2 output pulse at Q will be a positive pulse 1 millisecond wide. More significant is the fact that its leading edge occurs about 5 milliseconds after the generator triggered OS1. Thus with respect to the generator, the OS2 output pulse is said to be delayed 5 milliseconds. By controlling T_p of OS1, we control the delay of the OS2 output pulse.

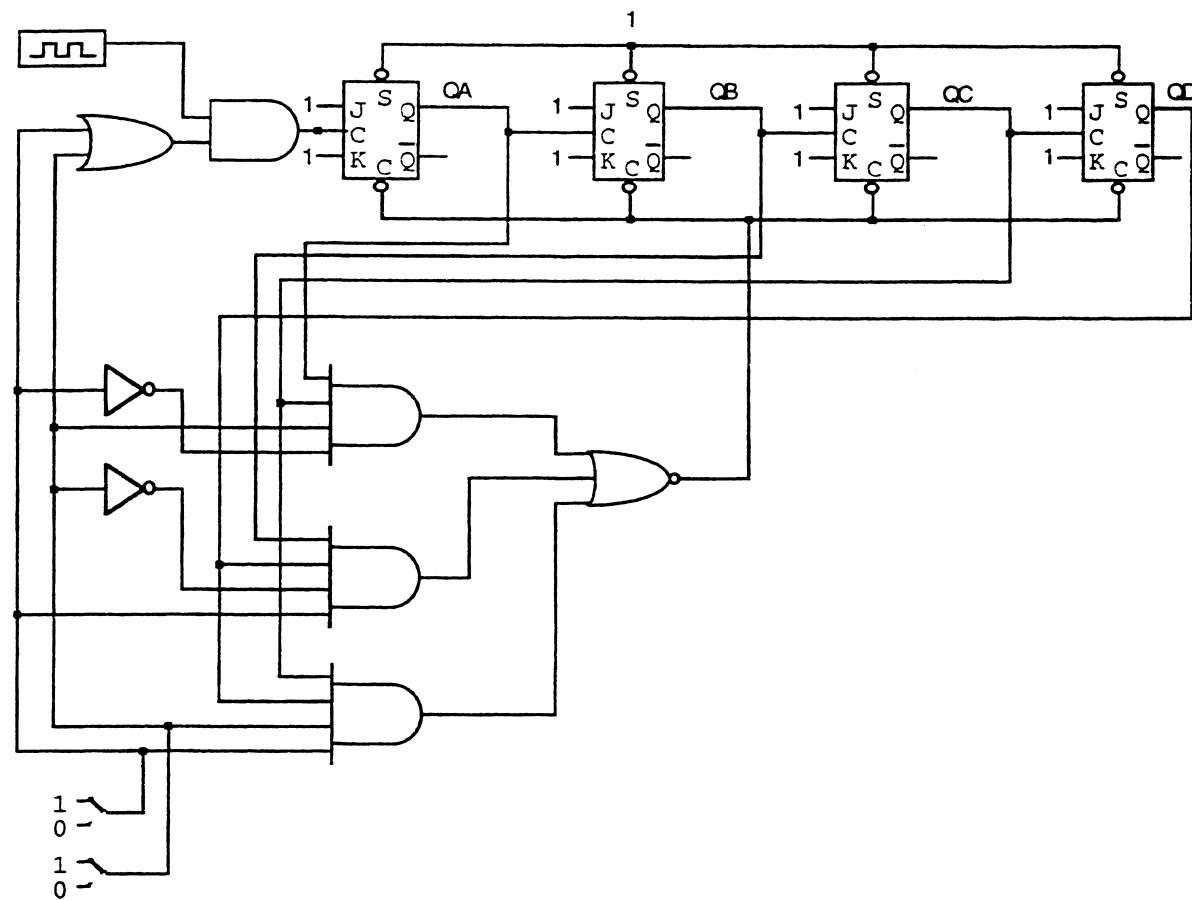
6. stable

7. true

- 8.
- a) 1.04 ms
 - b) 5.75 ms
 - c) 6.79 ms
 - d) 85 %

EXPERIMENT 18
DESIGNING WITH
FLIP-FLOP DEVICES

- i) A model solution for the programmable counter:



- j) Review:

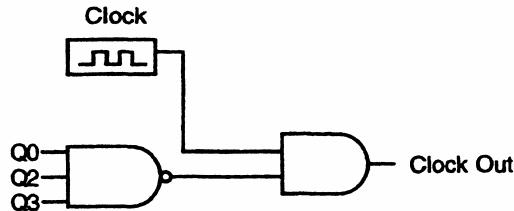
1. Note that there are eight different states in the table. Also note that output X_3 goes through one complete cycle in 8 counts. Input the signal you wish to divide by eight at the counter clock input and the desired output can be taken out at X_3 .
2. 15, 16, 16.
3. One difference between the two methods is that the asynchronous transfer can be accomplished without the system clock, unlike the synchronous transfer. Thus data can be jammed into flip-flops without regard to what is going on at the synchronous inputs. Another difference is the relative complexity of the asynchronous circuitry compared to the synchronous circuitry.

4. 8

5. Yes. Since parallel transfer takes about one clock pulse to transfer 8 bits (or any number of bits) and serial transfer takes about 8 clock pulses, parallel transfer is 8 times as fast.

6. The NAND gate is connected so that a LOW is produced at its output on the count of 5. This is referred to as "decoding the number 5" and the NAND gate and its input circuits are referred to as a decoder. When the counter reaches a count of 5, the output of the NAND causes the counter flip-flops to clear and the counter begins counting at 0 again. Thus the counter counting sequence is 0,1,2,3,4,0,... The count of 5 is not present but a few nanoseconds and is not included in the sequence. Since there are 5 different states, the counter's MOD number is 5.

7. One way would be to decode the desired count with a NAND gate decoder and connect its output to one input of a two-input AND gate. The clock can be connected to the other AND gate input and the AND gate's output applied to the clock input of the binary counter. See the circuit below:



This circuit could be used in a 4-bit counter's clock circuit to stop the counter at the count of 13 ($Q_3 = Q_2 = Q_0 = 1$, and $Q_1 = 0$).

EXPERIMENT 19
TROUBLESHOOTING
FLIP-FLOP CIRCUITS

k) Review:

1. This condition will normally cause Q to be stuck HIGH (see text, Example 5-16). If the counter is started from 000, the input flip-flop will toggle while the second flip-flop will be stuck-HIGH, and the output flip-flop will be stuck-LOW. The count sequence will be 2,3,2,3,...
2. Yes, since t_{PHL} for 74L00 gates range up to 60 ns, typically around 31 ns. Thus the total propagation delay will exceed 62 ns. Since t_{PHL} for the 74LS74 is about 25 ns, D_2 will be HIGH before the PGT arrives at CLK. Since set-up time for the 74LS74 is 20 ns, the flip-flop will respond to the HIGH at D_2 and Q_2 will go HIGH.
3. No, probably not, since t_{PHL} for 7400 gates range up to 15 ns, typically around 7 ns. The total propagation delay is about 14 ns. Since t_{PHL} for the 74LS74 is about 25 ns, D_2 will be HIGH 11 ns after the PGT arrives at CLK. Thus Q_2 will most likely stay LOW.

EXPERIMENT 20
BINARY ADDERS AND
2'S COMPLEMENT SYSTEM

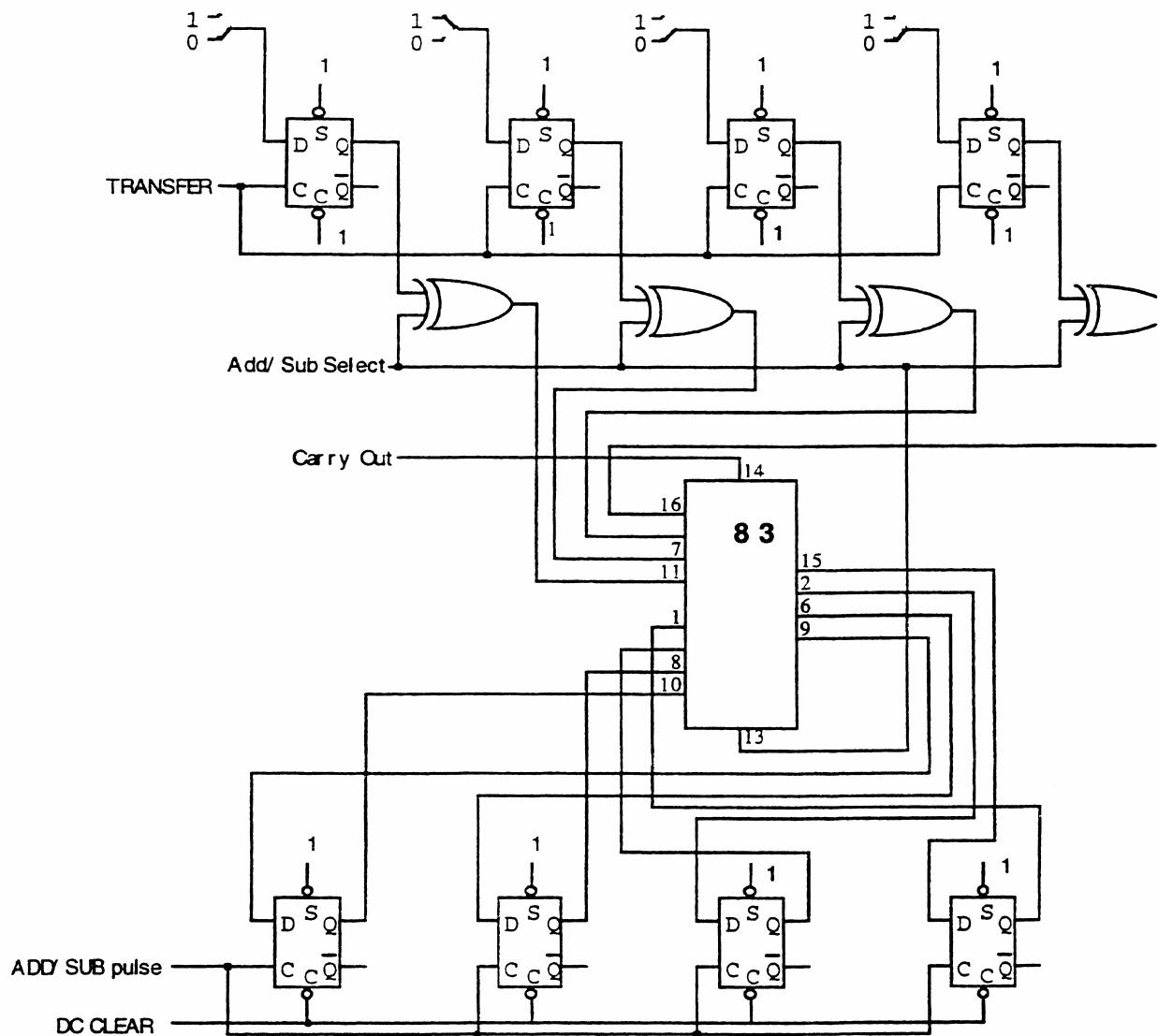
c) See Figure 6-7, page 281 of the text.

h) Model solution:

Inputs								Outputs				
A ₃	A ₂	A ₁	A ₀	B ₃	B ₂	B ₁	B ₀	C ₄	S ₃	S ₂	S ₁	S ₀
0	0	1	1	0	0	0	1	0	0	1	0	0
0	1	1	1	1	0	0	1	1	0	0	0	0
1	0	1	1	0	1	0	1	1	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	0

Table 20.3

i) Model diagram for Adder/Subtractor:



l)	<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>	S_3	S_2	S_1	S_0
1)		1	0	1	0 (overflow)
2)		0	0	0	1
3)		0	1	1	0
4)		0	1	0	0 (overflow)
5)		9 is too big			
6)		1	0	0	0 (special case: -8)
7)		0	1	0	0 (overflow)

m) Review:

1. Connecting the carry input of the full adder to LOW.
2. nine
3. -8 to +7
4. The accumulator in a digital arithmetic unit accumulates the sums from when performing strings of additions as does register A.

EXPERIMENT 21
ASYNCHRONOUS COUNTERS

d) The signal should be a square wave.

f) Model:

Input Pulse Applied	Output States				Decimal Number
	Q ₃	Q ₂	Q ₁	Q ₀	
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9
10	1	0	1	0	10
11	1	0	1	1	11
12	0	0	0	0	0
13	0	0	0	1	1
14	0	0	1	0	2
15	0	0	1	1	3

Table 21.1

g) MOD 12; the signal should not be a square wave.

h) Model:

Input Pulse Applied	Output States				Decimal Number
	Q_3	Q_2	Q_1	Q_0	
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9
10	0	0	0	0	0
11	0	0	0	1	1
12	0	0	1	0	2
13	0	0	1	1	3
14	0	1	0	0	4
15	0	1	0	1	5

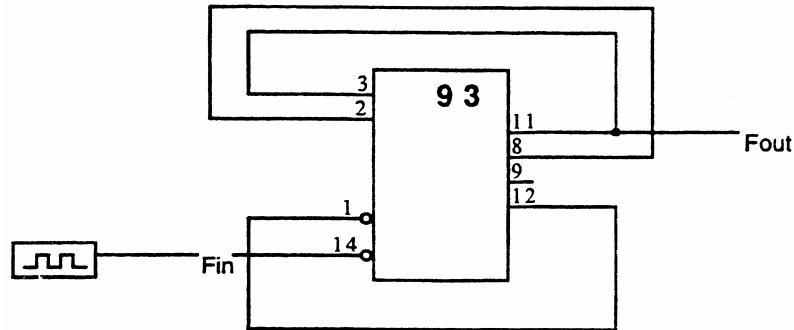
Table 21.2

h) Note that the number of 1s and 0s in the Q_3 column are not equal. This indicates that the output signal here is not a square wave.

i) 10 Hz; MOD 600.

k) Review:

1. Model solution:



2. 15,14,13,12,...,0,15... In other words, the counter counts down.

3. In order for the output at Q_3 to be a square wave, there must be an equal number of counts for both 0 and 1 states during each count cycle. Since the counter is forced to reset before this occurs (the count cycle is shortened), the result is an unequal number of counts for the 0 and 1 states. Thus the signal cannot be a square wave.

4. 1 kHz

EXPERIMENT 22

BCD IC COUNTERS

f) Omit

g) Omit

clk	Carry	Q3	Q2	Q1	Q0
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	1	0	0	0	0

Table 22-1

b) What is the maximum count displayed by your counter: 99

EXPERIMENT 23
SYNCHRONOUS IC COUNTERS

d)

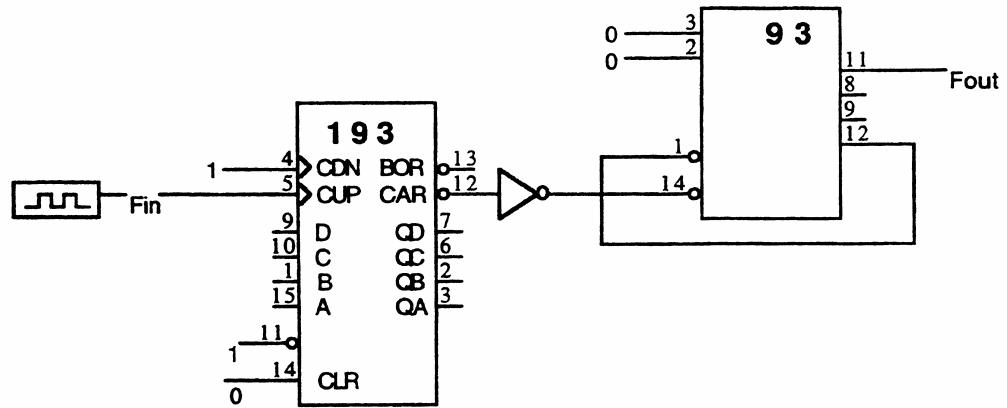
Input Pulse Applied	Output States				Decimal Number	$\overline{T\bar{C}_U}$
	Q_3	Q_2	Q_1	Q_0		
None	0	0	0	0	0	1
1	0	0	0	1	1	1
2	0	0	1	0	2	1
3	0	0	1	1	3	1
4	0	1	0	0	4	1
5	0	1	0	1	5	1
6	0	1	1	0	6	1
7	0	1	1	1	7	1
8	1	0	0	0	8	1
9	1	0	0	1	9	1
10	1	0	1	0	10	1
11	1	0	1	1	11	1
12	1	1	0	0	12	1
13	1	1	0	1	13	1
14	1	1	1	0	14	1
15	1	1	1	1	15	0
16	0	0	0	0	16	1

g)

Input Pulse Applied	Output States				Decimal Number	$\bar{T}\bar{C}_D$
	Q_3	Q_2	Q_1	Q_0		
None	0	0	0	0	0	0
1	1	1	1	1	15	1
2	1	1	1	0	14	1
3	1	1	0	1	13	1
4	1	1	0	0	12	1
5	1	0	1	1	11	1
6	1	0	1	0	10	1
7	1	0	0	1	9	1
8	1	0	0	0	8	1
9	0	1	1	1	7	1
10	0	1	1	0	6	1
11	0	1	0	1	5	1
12	0	1	0	0	4	1
13	0	0	1	1	3	1
14	0	0	1	0	2	1
15	0	0	0	1	1	1
16	0	0	0	0	0	0

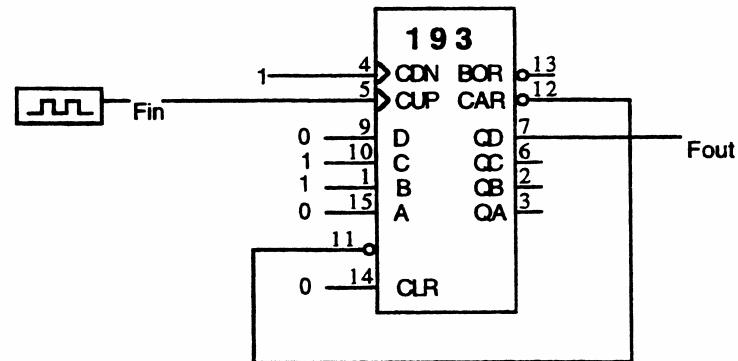
Table 23.2

i)

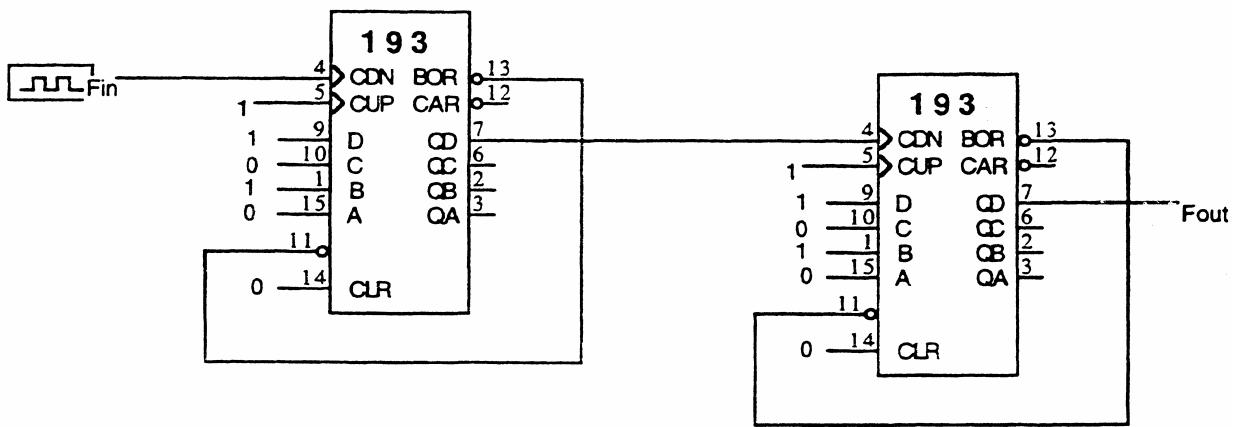


k) Review:

1. Model solution:



2.



3. $\overline{TG_U} = 1$, $MR = 0$, $\overline{PL} = 1$, and a PGT at TCD .

EXPERIMENT 24
IC COUNTER APPLICATION:
FREQUENCY COUNTER

g) Review:

1. From t_1 to t_2 : SAMPLE PULSES provides an NGT to J-K flip-flop X which causes the FF to toggle from LOW to HIGH, providing one enable to NAND gate Z. This PGT also causes OS Y to produce a RESET pulse to the counters.

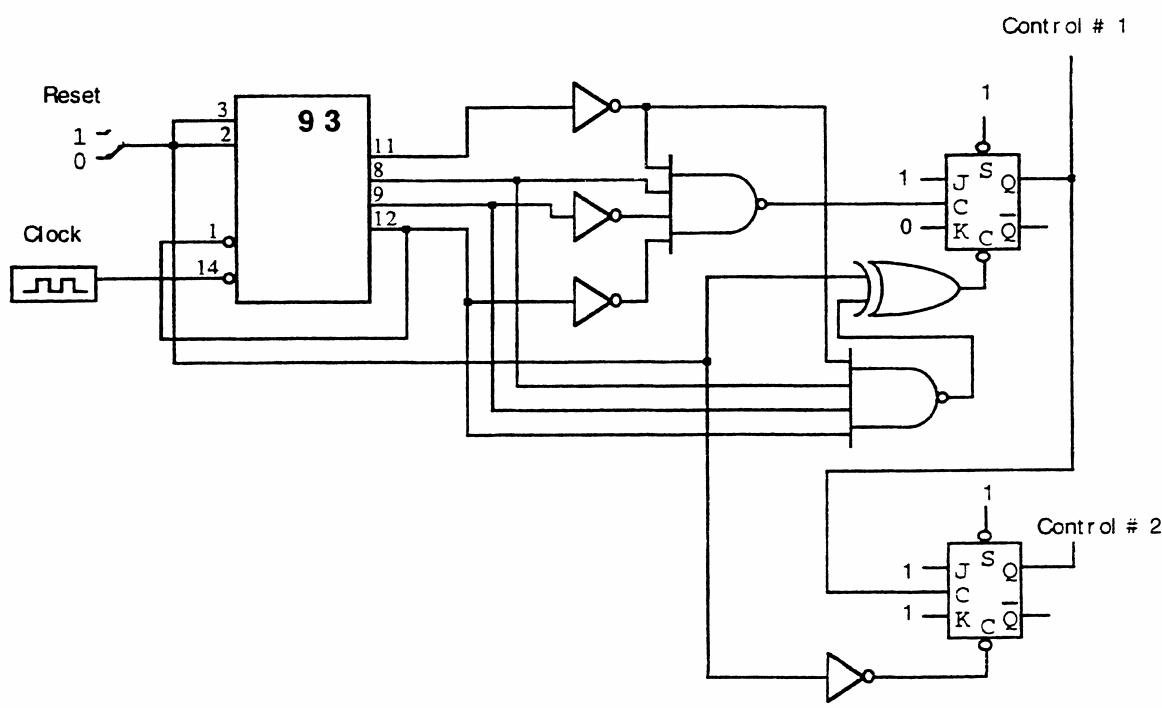
From t_2 to t_3 : SAMPLE PULSES provides the 2nd enable to NAND gate Z. The input signal is now applied to the BCD counters. The BCD display displays the count.

From t_3 to t_4 : SAMPLE PULSES provides an NGT to flip-flop X which causes the FF to toggle from HIGH to LOW. The LOW disables NAND gate Z which, along with the LOW from SAMPLES PULSES, inhibits the input signal from further clocking the BCD counters. The BCD display units display the number of input pulses in one second.

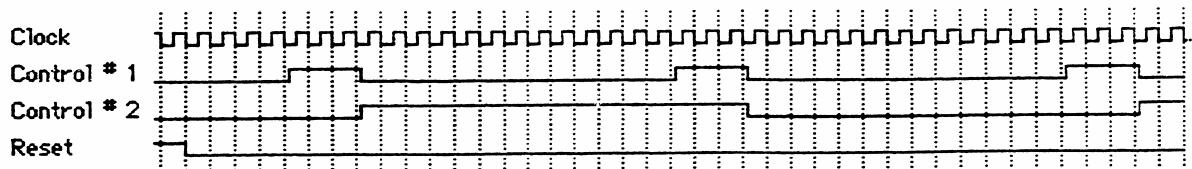
2. The 74273 IC latch requires a PGT to transfer data from the counters to the display units. This PGT must arrive at the time the input is inhibited from the counters. The X output of FF X produces a NGT at this time. Therefore the X output will produce the required PGT at this time.

EXPERIMENT 25
TROUBLESHOOTING COUNTERS:
CONTROL WAVEFORM GENERATION

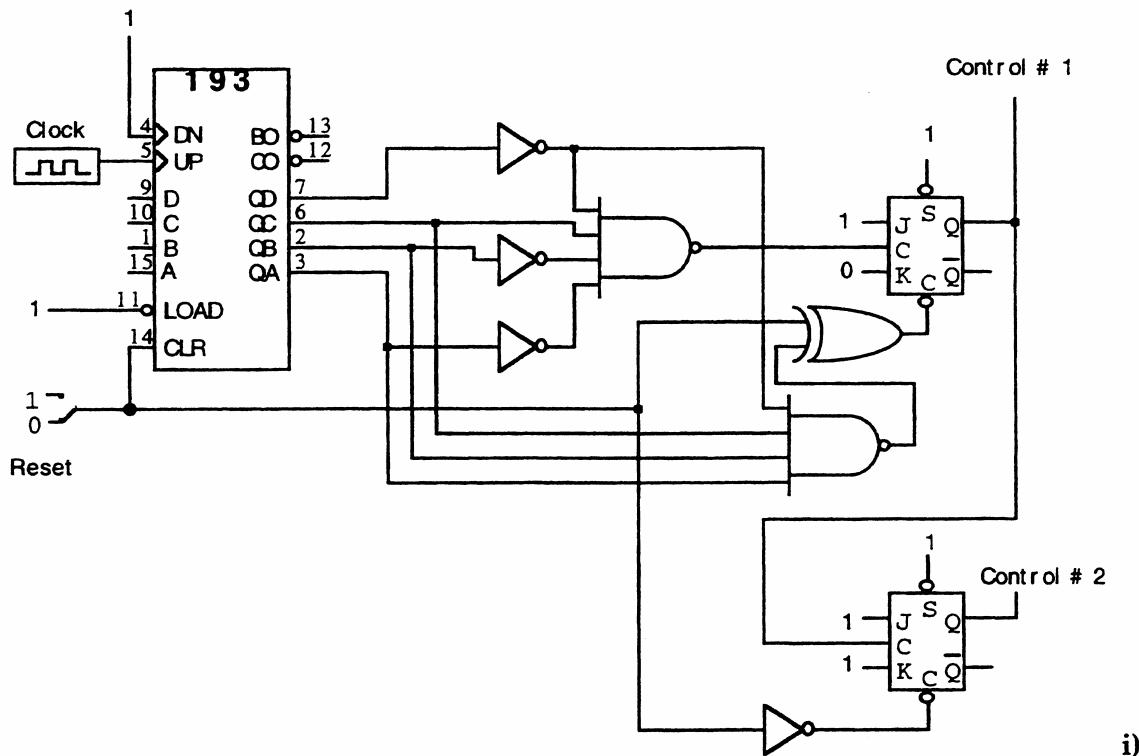
a)



b)



e)



Review:

1. A glitch is typically produced when decoding an asynchronous counter. The asynchronous counter can produce "invalid counts" because of propagation delay. For example, when the counter counts from 3 to 4, the counter produces a temporary count of 2 and then 0 - each of a few nanoseconds in duration - before reaching 4. These spurious counts cause decoders for these numbers to produce outputs that are also a few nanoseconds wide - glitches.

Decoders used in reduced modulus counters also produce a glitch when decoding the MOD number of the counter.

Signals having glitches can cause circuitry to "misfire" - to trigger at the wrong time or to SET or CLEAR at the wrong time.

2. Again, propagation delay is the culprit. If the flip-flops in a synchronous counter have different propagation delays, usually because of different loading, glitches can still occur.

3. Strobing.

4. When the circuit starts from the reset condition, flip-flop X will be SET on the count of 4 since the decoder will produce a NGT at that time. On the count of 7, flip-flop X will be cleared creating a NGT at its output which causes flip-flop Y to toggle

to HIGH. After the counter reaches a count of 4 again, flip-flop X will be SET again so that on the next count of 7, flip-flop Y will toggle to LOW. This sequence is repeated as long as the clock is applied.

EXPERIMENT 26
SHIFT REGISTER COUNTERS

- d) The MOD-number of this counter is 4. The outputs of this counter are not square waves. The frequency of each output is $1/4$ the clock frequency.

e)

Shift Pulse	Output States				Number
	Q_3	Q_2	Q_1	Q_0	
0	0	0	0	0	0
1	1	0	0	0	8
2	1	1	0	0	12
3	1	1	1	0	14
4	1	1	1	1	15
5	0	1	1	1	7
6	0	0	1	1	3
7	0	0	0	1	1
8	0	0	0	0	0

Table 26.1

- f) The MOD-number of this counter is 8. The outputs are square waves. The frequency of each output is $1/8$ the clock frequency.

g) Review:

1. at least one, SET.
2. the number of flip-flops.
3. twice the number of flip-flops.
4. Johnson

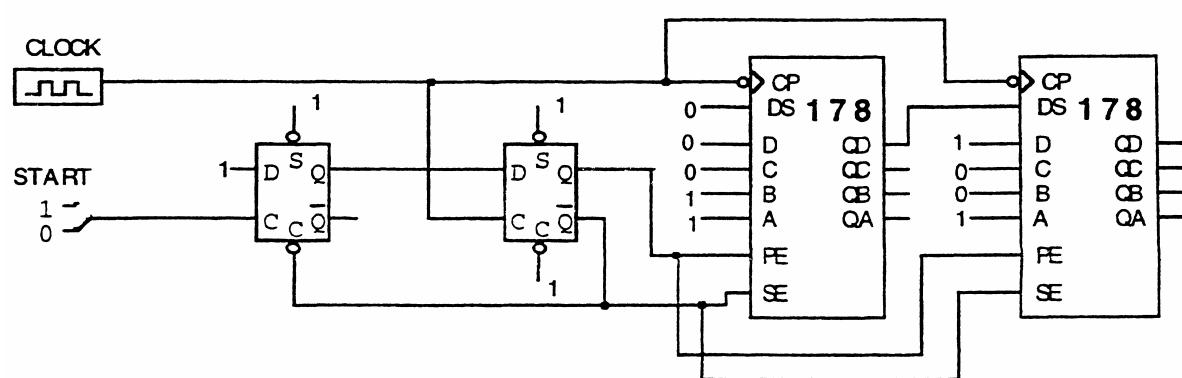
EXPERIMENT 27

IC REGISTERS

j) Review:

1. Each operation requires a minimum set-up time of 35 nanoseconds. Asynchronous control of either operation could result in this requirement not being met, thus causing the operation to fail.
2. The START signal occurs asynchronously. When it does occur, flip-flop Y transfers a 1 to flip-flop X. The next system clock pulse causes flip-flop X to enable the parallel load mode. Simultaneously, flip-flop X clears flip-flop Y, which results in a 0 being applied to the D input of flip-flop X. On the next clock pulse, flip-flop X disables the parallel load mode and enables the shift mode. Thus the two operations execute synchronously.
3. 1, the number of bits being transferred.

4. Model solution:



EXPERIMENT 28
DESIGNING WITH
COUNTERS AND REGISTERS

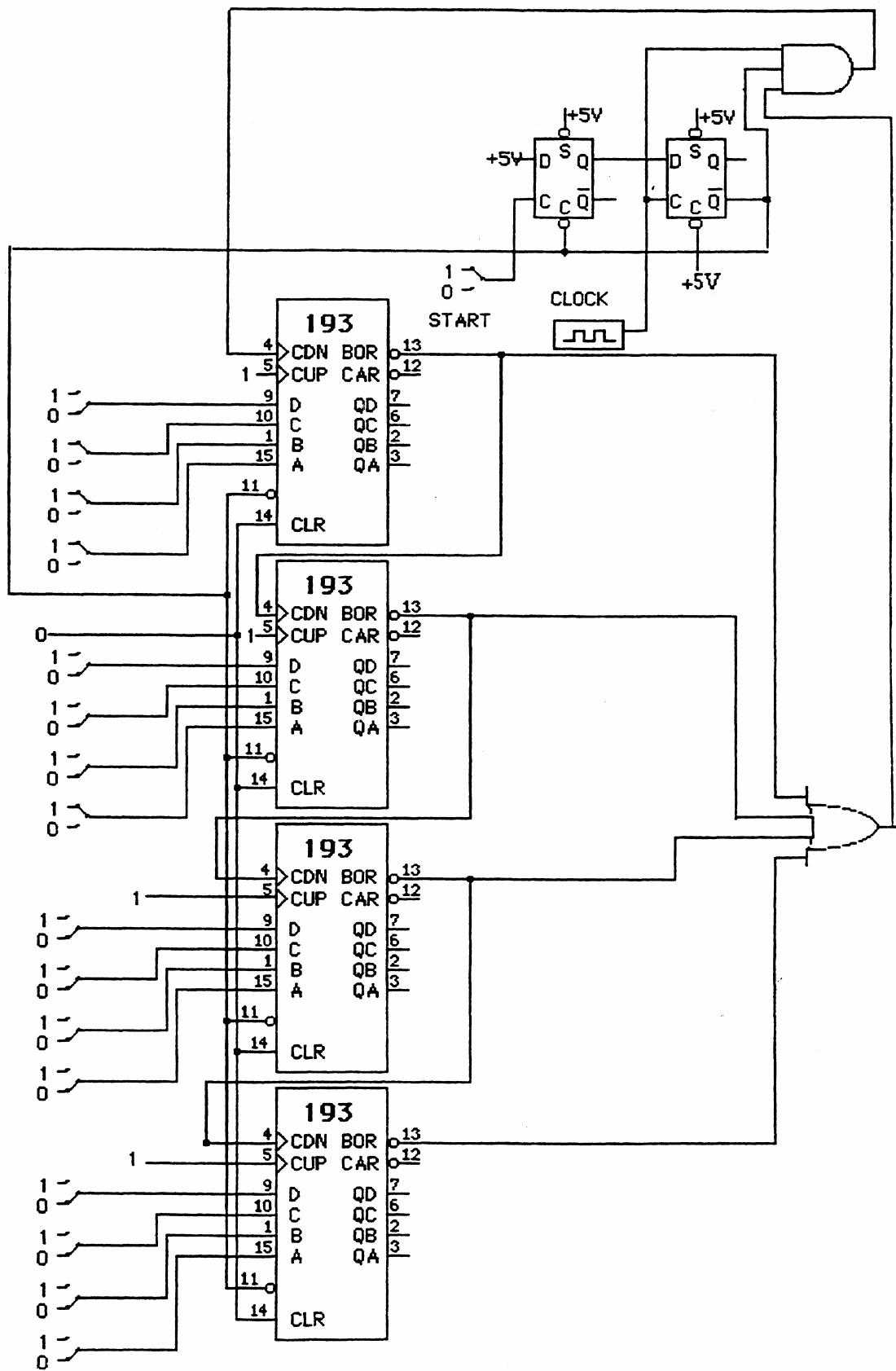
- a) One solution to this problem is to implement a Schmitt-trigger oscillator followed by a MOD 16 counter as shown below. When selecting the timing components, first choose a 330 W resistor for R. Then, using the formula

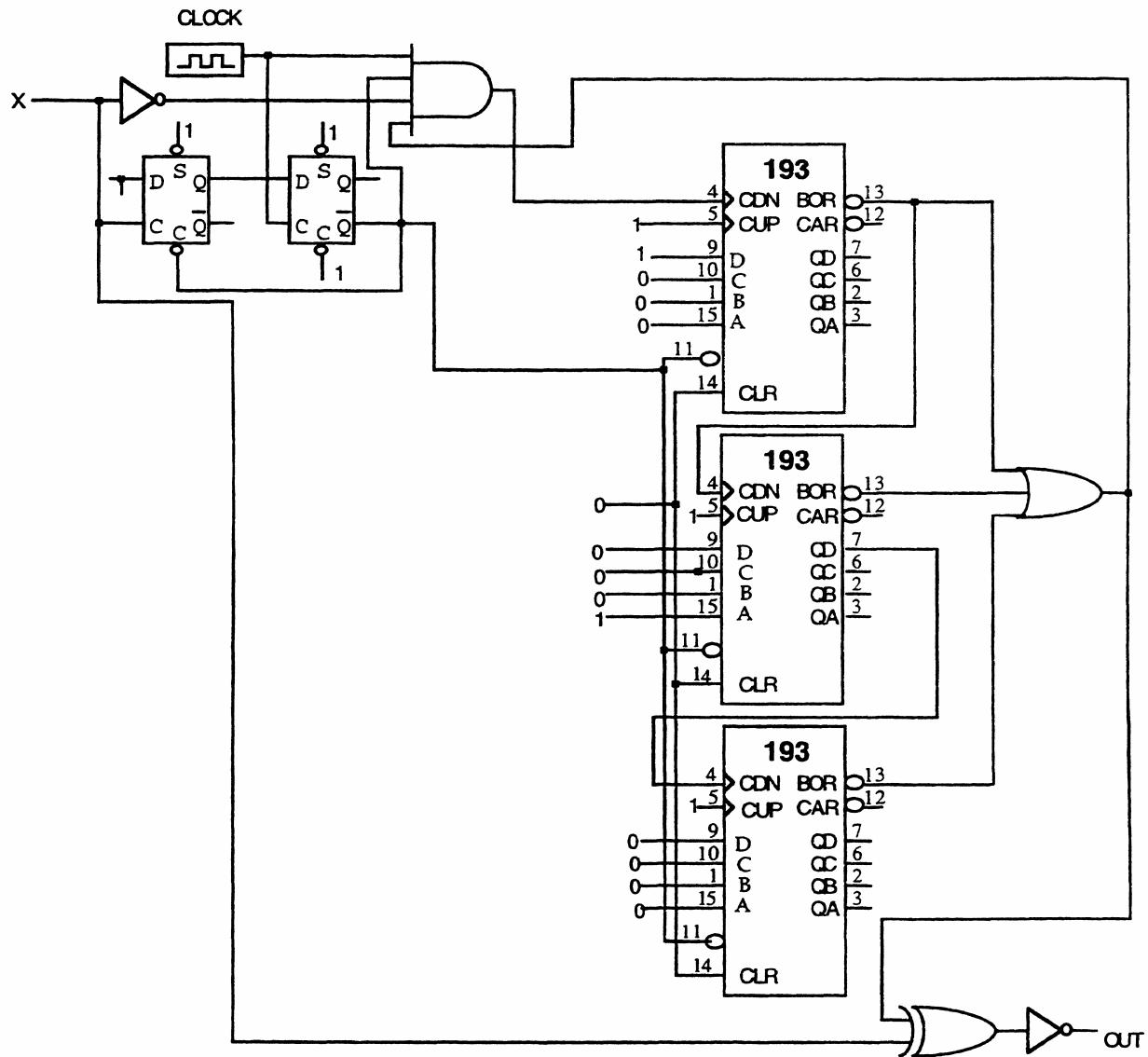
$$f = \frac{0.8}{RC}$$

you will find that $C = .024 \text{ mF}$.

Other solutions may use an IC astable oscillator such as a 555.

- b) Model solution (next two pages):



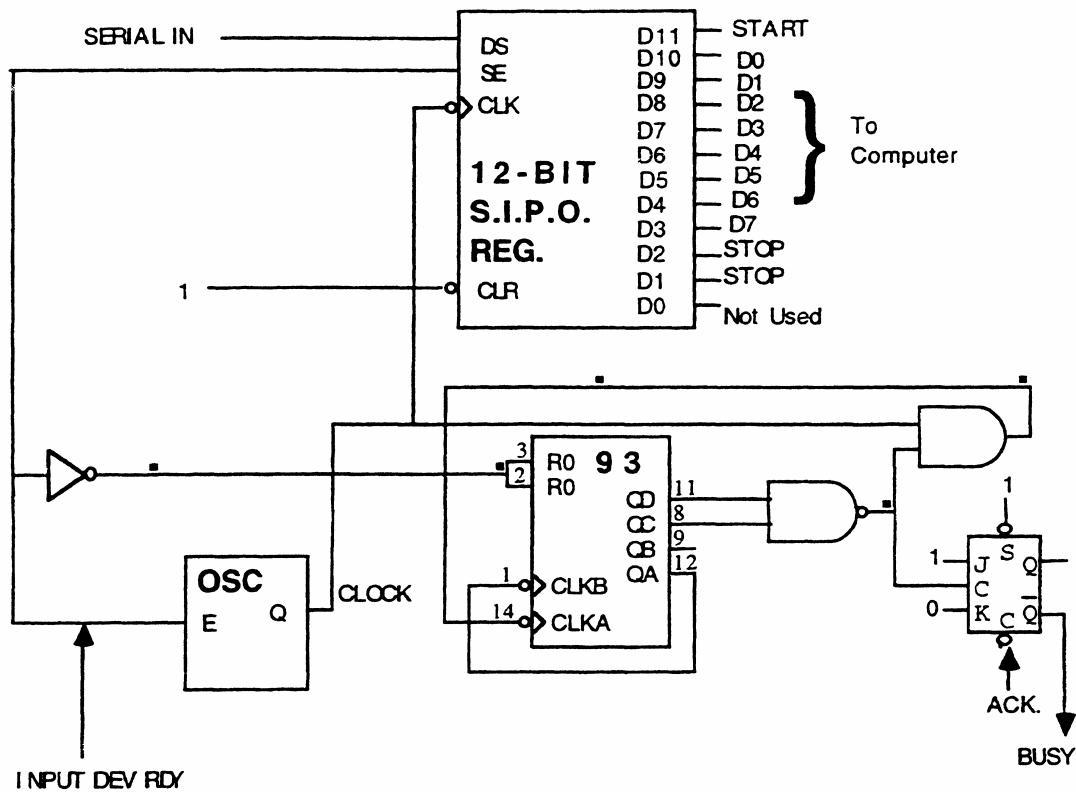


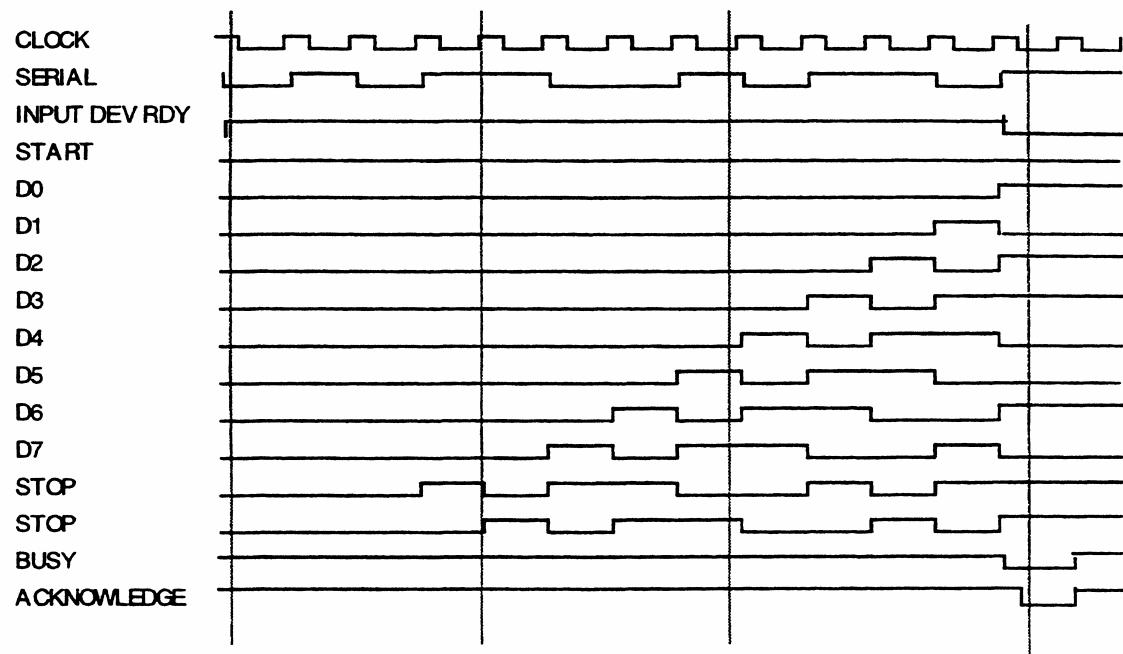
c) Model solution:

There are a couple of clarifications for this problem. There are two STOP bits. There are varying degrees of difficulty in solving this problem, depending on the constraints the instructor wishes to impose on the students. For example, the instructor may wish to have the student detect the START bit and use this event to indicate that the input device is ready rather than use the handshaking signal INPUT DEVICE READY. This would require that the student provide circuitry for verifying a valid START bit. Some may want a synchronous design; others may

want an asynchronous design. Finally, some instructors may want the student to design the circuits necessary for testing the interface.

The solution given below is an asynchronous design. The 12-bit S.I.P.O. register can be implemented using three 74178 ICs (or two 74164 ICs). The oscillator is a square wave generator with a period of 100 ms. The clock oscillator is started when the INPUT DEVICE READY is asserted by the input device. It is disabled after the last bit is received. The J-K flip-flop generates the BUSY signal and receives the ACKNOWLEDGE from the computer.





Example waveforms for problem c.

The data in the above example waveforms is 4D.

EXPERIMENT 29
TTL AND CMOS IC
FAMILIES - PART I

o) Review:

1. $t_{PHL}, t_{PLH}; \frac{t_{PHL} + t_{PLH}}{2}$
2. Increasing the number of loads increases the sink current through the driver output transistor. This increases the voltage drop across this transistor and hence increases the output voltage V_{OL} .
3. 1.6 milliamperes
4. 5

EXPERIMENT 30
TTL AND CMOS IC
FAMILIES - PART II

h) 410.7 ohms

i)

Inputs			Output
A	B	C	x
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Table 30.1

j) NOR

n)

E	Bus
0	A ₁
1	A ₂

Table 30.2

s) Review:

1. $V_{OH(min)}$ of the TTL gate is too low compared to $V_{IH(min)}$ of the CMOS gate. The pull-up resistor is needed to raise V_{OH} to a level acceptable to the CMOS gate.
2. the output of the CMOS gate is LOW
3. Many LEDs require more current than a TTL totem-pole output can handle. The open-collector buffer-driver between a totem-pole output and an LED solves this problem.

4. When one gate is HIGH while the other gate is LOW, the sink transistor of the gate that is LOW can draw potentially damaging current (up to 55 mA).

5. $\bar{A} \bar{B} \bar{C} \bar{D} \bar{E} \bar{F} = \overline{A + B + C + D + E + F}$

6. Busses are designed to carry signals from more than one source. Tristate buffers are used to prevent bus contention between the sources by enabling only one source at a time while disabling all others.

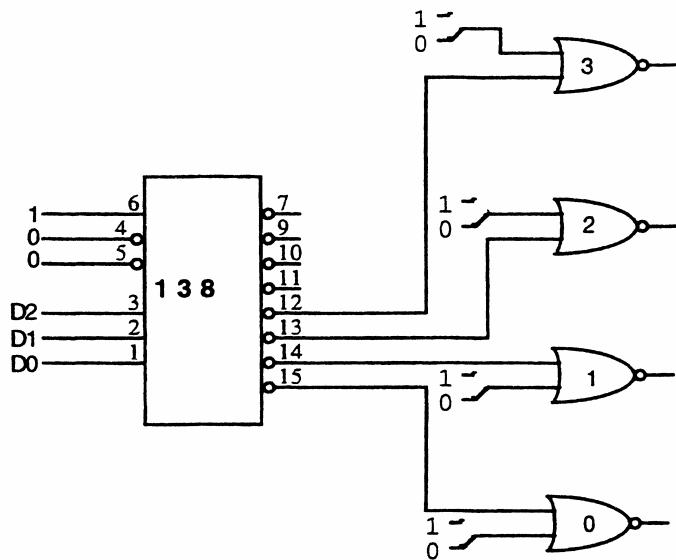
7. TTL tristate can switch signal sources onto a bus in only one direction. Also, the signals must be TTL compatible. The signals through a CMOS transmission gate can generally go in either direction and may be digital or analog provided that the range of the signals stay within 0 to V_{DD}. There are some CMOS transmission gates which even permit bipolar signals.

EXPERIMENT 31
USING A LOGIC ANALYZER

(NO SOLUTIONS)

EXPERIMENT 32 IC DECODERS

e) Model:



Select Inputs			Outputs							
A ₂	A ₁	A ₀	\bar{O}_0	\bar{O}_1	\bar{O}_2	\bar{O}_3	\bar{O}_4	\bar{O}_5	\bar{O}_6	\bar{O}_7
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

Table 32.1

n) Review:

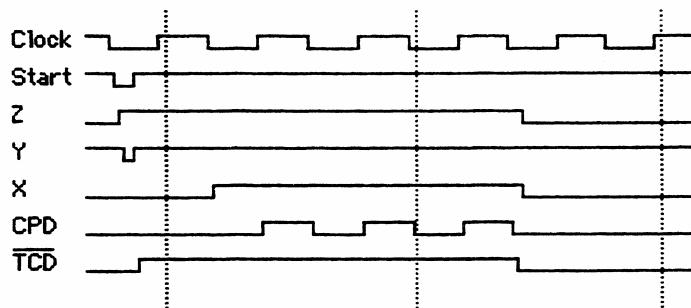
1. Constructing reduced modulus counters, converting binary to decimal, converting BCD to 7-segment, enabling (and inhibiting) circuits when a given number is detected, and others.
2. $\overline{E_1}$ and $\overline{E_2}$ must be LOW at the same time E_3 is held HIGH.
3. If the logic analyzer and 7442 are connected as in step h, output 3 will be enabled. The logic analyzer will be triggered on 0011.
5. slightly more than 0.25 ms.

EXPERIMENT 33 IC ENCODERS

c)

- 2) positive
- 3) LOW, 3, HIGH
- 4) HIGH, negative, clock
- 5) 2, LOW, X
- 6) the next application of the START switch.

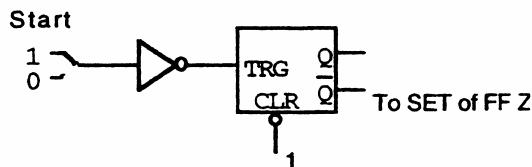
d)



Timing Diagram 33-1

g) The timer does not operate. This is due to the fact that $\overline{TG_D}$ goes LOW while the START switch is depressed. Thus flip-flop Z is not RESET and the next time START is depressed, flip-flop Z cannot provide the necessary PGT to OS. Hence the 74192 cannot be preset - it stays at 0000.

Another OS can be connected to the START circuit. This will insure that no matter how long START remains depressed, the START pulse will return HIGH before the arrival of $\overline{TG_D}$ LOW, provided an appropriate t_p is selected for the OS. See the following figure:



j) Review:

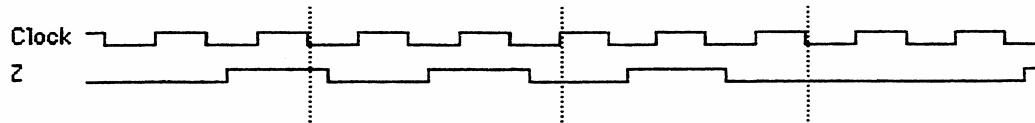
1. Switch contact bounce could cause the OS to trigger several times since t_p is extremely small (100 ns). This would cause $\overline{TG_D}$ to go LOW prematurely, clearing flip-flop X and disabling the clock.

2. The $\overline{TG_D}$ output's loading could easily be exceeded, causing that signal to be marginal.
3. See g above.

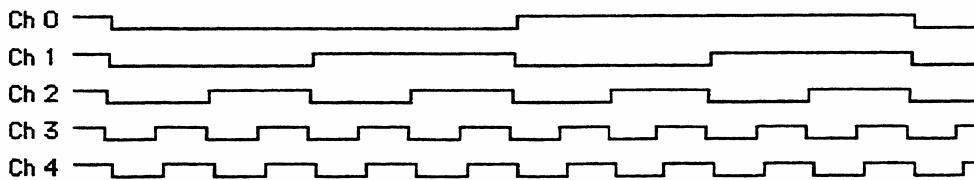
EXPERIMENT 34

IC MULTIPLEXERS AND DEMULTIPLEXERS

b) Example waveforms:



e) Example waveforms:



i) Transmitter section operation:

Each register A, B, C, and D are configured as circular shift registers. Each register will shift on the PGT of the SHIFT pulses from AND gate 2. The word counter selects the register data that will appear at Z. This counter counts from 00 to 11. The bit counter makes sure that 4 data bits from each register are transmitted through the multiplexer before advancing to the next register. This counter advances one count per SHIFT pulse, so that after 4 SHIFT pulses, it recycles to 0. The word counter is incremented by one. The Z signal contains 4 bits from each register for a total of 16 bits. The transmission system is controlled by the two D flip-flops, AND gates 1 and 2, and the one-shot.

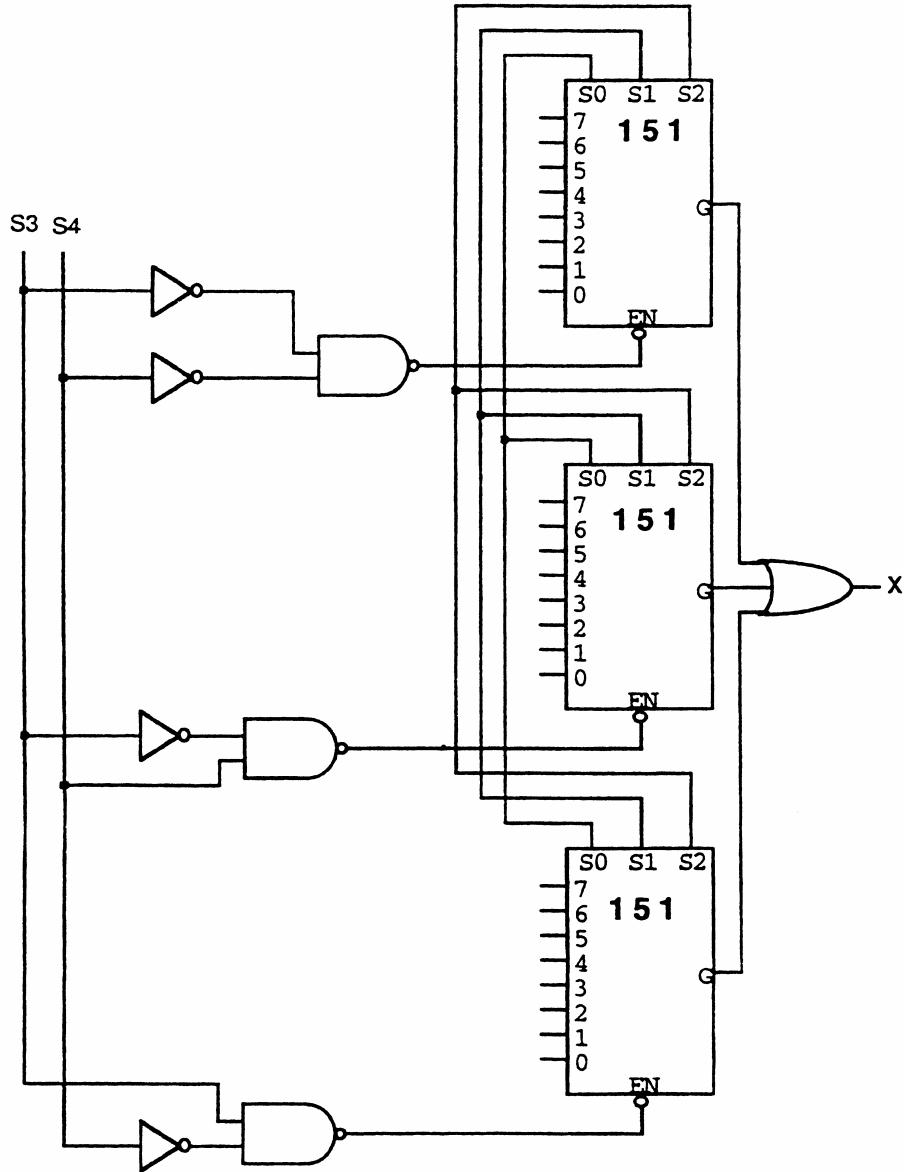
Receiver section operation:

The receiver demultiplexes the Z signal into 4 sets of data and distributes them to their respective outputs.

o) Review:

1. Change the multiplexer and demultiplexer to 8-channel and change the word counter to a MOD-8.
2. Register A's output to the multiplexer would be all 1's since the register is connected as a ring counter.

3. Model solution:



BCD input	V _{A'} expected	V _{A'} observed
00	-0.0 V	-0.04 V
05	-0.5 V	-0.51 V
10	-1.0 V	-1.03 V
15	-1.5 V	-1.51 V
20	-2.0 V	-2.02 V
25	-2.5 V	-2.50 V
30	-3.0 V	-3.01 V
35	-3.5 V	-3.49 V
40	-4.0 V	-4.04 V
45	-4.5 V	-4.52 V
50	-5.0 V	-5.03 V
55	-5.5 V	-5.51 V
60	-6.0 V	-6.03 V
65	-6.5 V	-6.50 V
70	-7.0 V	-7.02 V
75	-7.5 V	-7.49 V
80	-8.0 V	-8.02 V
85	-8.5 V	-8.50 V
90	-9.0 V	-9.00 V
95	-9.5 V	-9.50 V
99	-9.9 V	-9.88 V

Table 38.1

k) Have the students draw the waveform for only a few steps.

l) Review:

$$1. \text{ % Resolution} = \frac{1}{\# \text{ Steps}} \times 100\% = \frac{1}{99} \times 100\% = 1.01\%.$$

2. You probably would not have been able to achieve the correct settings for Full Scale and Step Size which would make the DVM grossly inaccurate.

EXPERIMENT 35
TROUBLESHOOTING SYSTEMS
CONTAINING MSI LOGIC CIRCUITS

(NO SOLUTIONS)

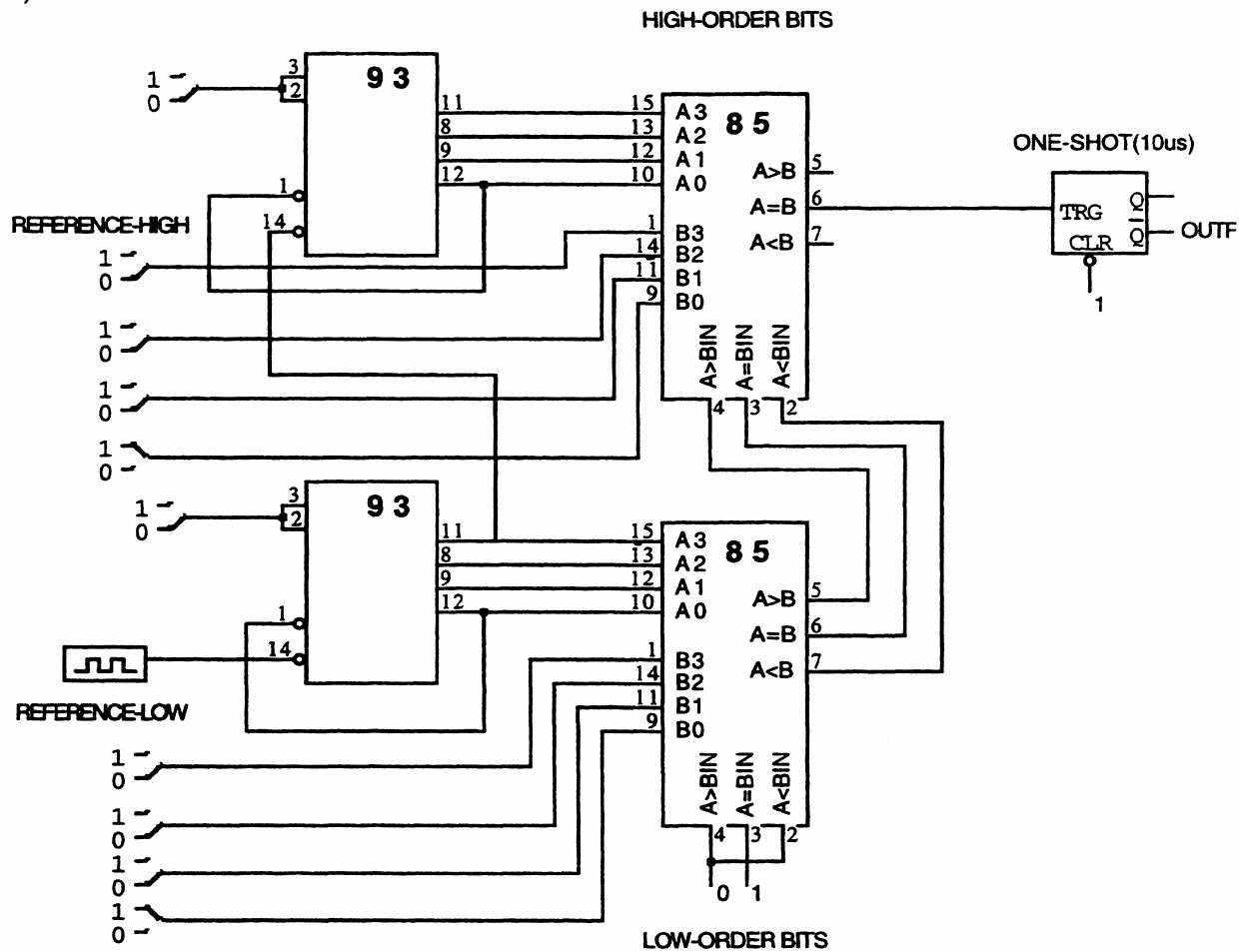
EXPERIMENT 36
IC MAGNITUDE
COMPARATORS

b)

Inputs								LED Monitors		
A_3	A_2	A_1	A_0	B_3	B_2	B_1	B_0	$A > B_{out}$	$A < B_{out}$	$A = B_{out}$
0	0	0	0	1	1	1	1	0	1	0
0	0	1	0	0	1	1	1	0	1	0
0	1	0	0	0	0	1	1	1	0	0
1	0	0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	0	0	0	1
1	1	1	1	0	0	0	0	1	0	0

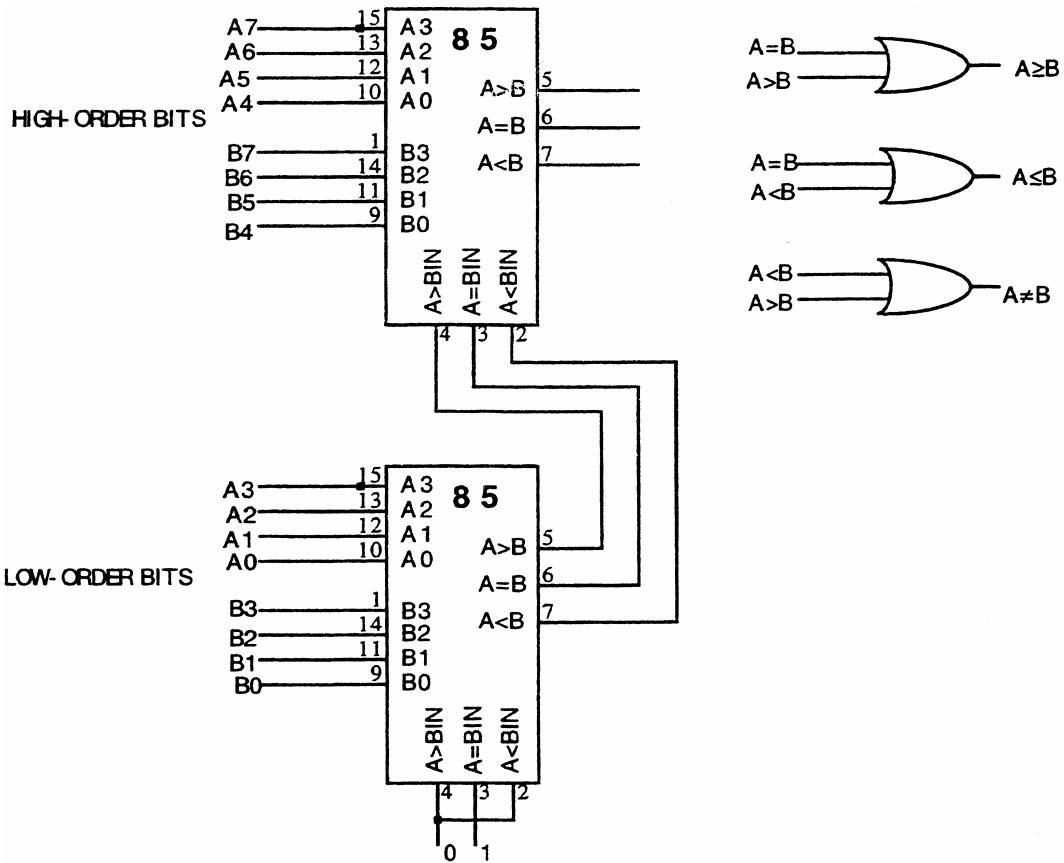
Table 36.1

e)

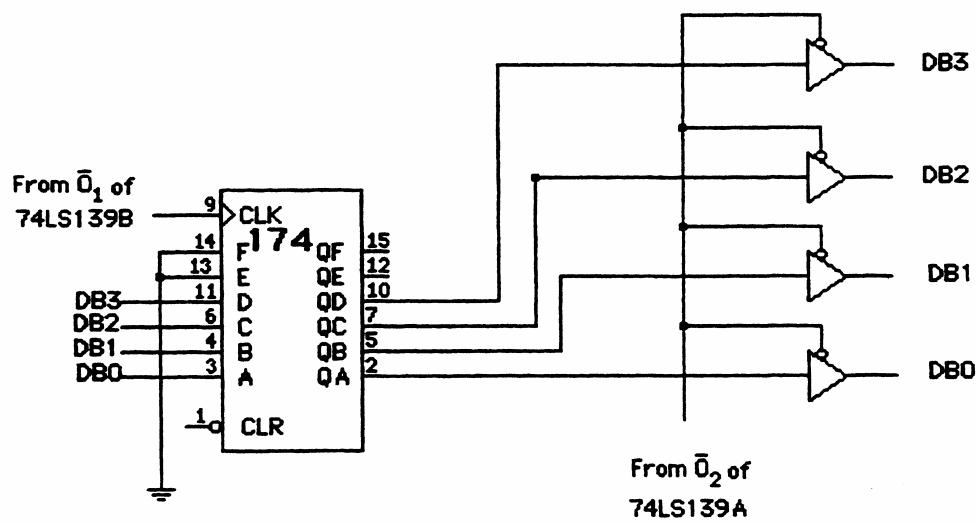


f) Review:

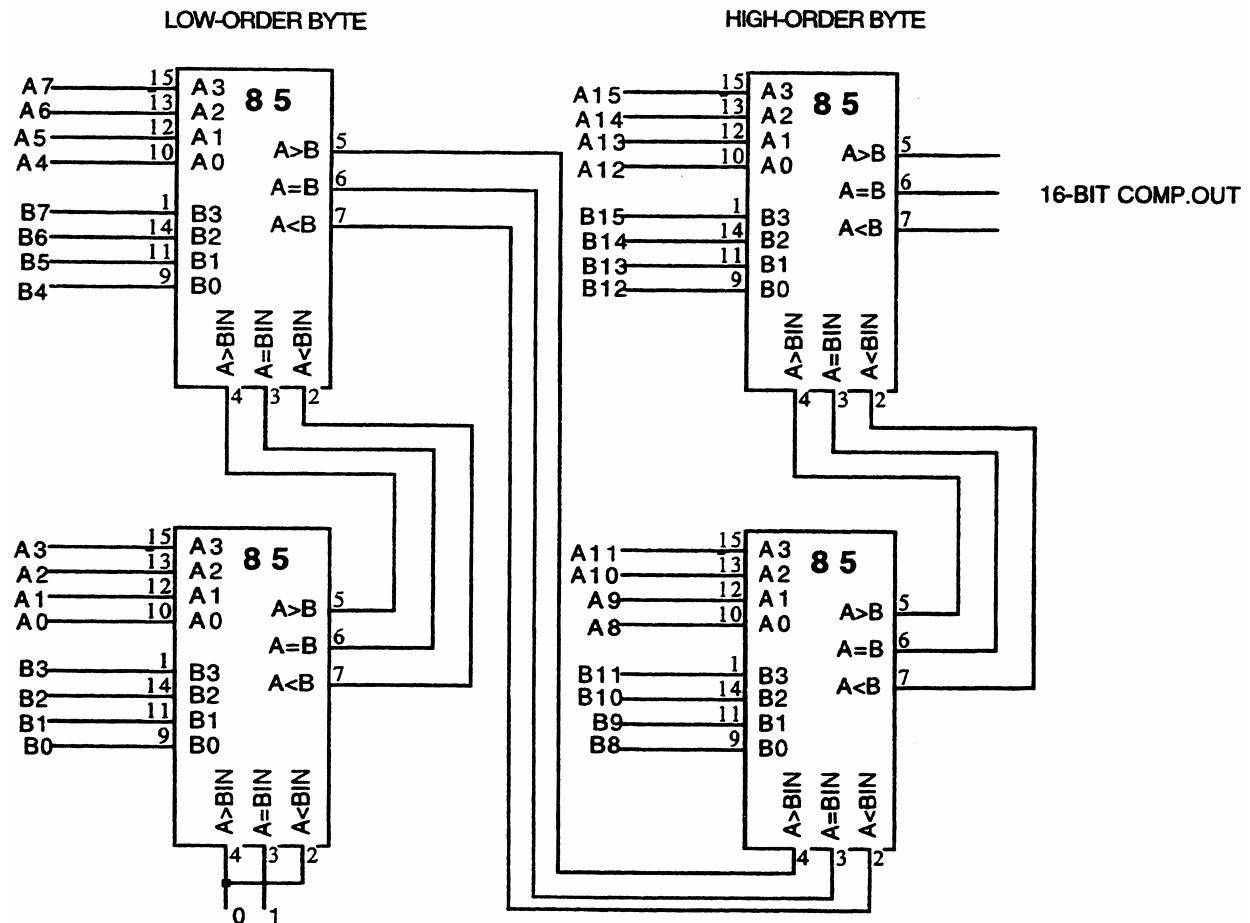
1. The following is a model solution for three comparison cases, $A \geq B$, $B \geq A$, and $A \neq B$. Select the appropriate OR circuit to the HIGH-ORDER 74LS85 outputs.



- 1) Modification requires tristate buffers at the outputs of the 74LS174 flip-flops. The tristate buffers can be controlled by the decoder. One way of accomplishing this is illustrated below:



2. Model solution:



EXPERIMENT 37

DATA BUSING

- e) 1) Press and hold \overline{OE}_B and \overline{IE}_C LOW at the same time.
- 2) Pulse the clock and release \overline{OE}_B and \overline{IE}_C .
- 3) Press and hold \overline{OE}_C LOW.
- 4) Pulse the clock and release \overline{OE}_C .
- 5) Press and hold \overline{OE}_A and \overline{IE}_B and \overline{IE}_C all LOW at the same time.
- 6) Pulse the clock and release \overline{OE}_A , \overline{IE}_B and \overline{IE}_C .

Time	[A]	[B]	[C]
t_1	1011	1000	1000
t_2	1011	1000	1000
t_3	1011	1011	1011

Table 37.1

- g) 1) Set data switches to 1101 and enable the buffers.
- 2) Press and hold \overline{IE}_A LOW.
- 3) Pulse the clock and release \overline{IE}_A . Disable the buffers.
- 4) Set data switches to 1110 and enable the buffers.
- 5) Press and hold \overline{IE}_B LOW.
- 6) Pulse the clock and release \overline{IE}_B . Disable the buffers.
- 7) Set data switches to 0101 and enable the buffers.
- 8) Press and hold \overline{IE}_C LOW.
- 9) Pulse the clock and release \overline{IE}_C . Disable the buffers.

For the above procedure, the state of the LEDs should be 0101 (contents of C).

h)

Selects				Data Transfer	
IS ₁	IS ₀	OS ₁	OS ₀	To Register	From Register
0	0	0	0	A	A
0	0	0	1	A	B
0	0	1	0	A	C
0	0	1	1	A	SW
0	1	0	0	B	A
0	1	0	1	B	B
0	1	1	0	B	C
0	1	1	1	B	SW
1	0	0	0	C	A
1	0	0	1	C	B
1	0	1	0	C	C
1	0	1	1	C	SW
1	1	0	0	No Register Affected	A
1	1	0	1	No Register Affected	B
1	1	1	0	No Register Affected	C
1	1	1	1	No Register Affected	SW

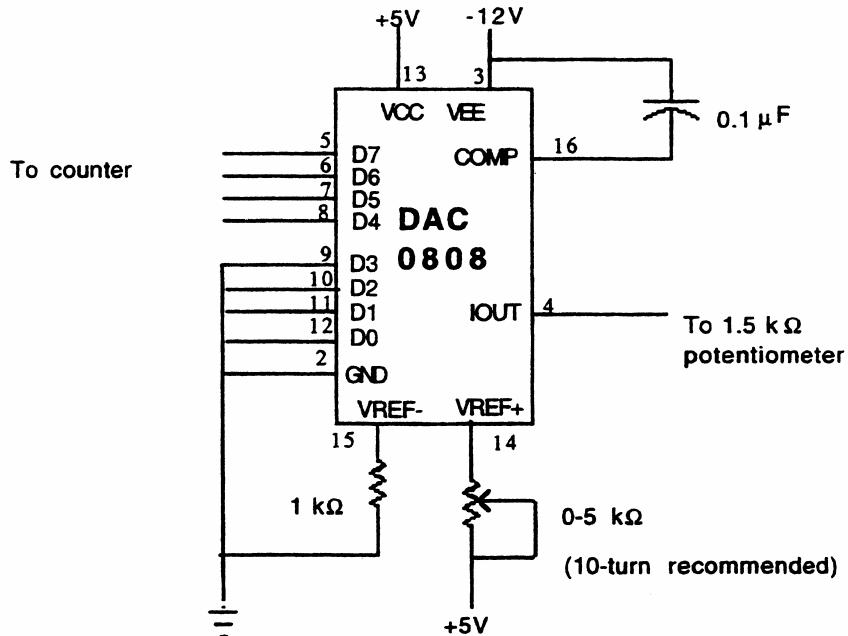
Table 37.2

i) **Review:**

1. Whenever two or more sources of data are placed on the same common connector (bus) and each are trying to assert its own level onto the connection.
2. Unbuffered switches connected to enabled outputs of devices can damage the devices.
3. With tristate devices, more than one data source may be connected to the same common connection as long as one tristate device can be enabled at a time. The disabled devices effectively disconnect their input sources from the bus.
4. The decoder enables only one output at a time since only one of its outputs can be LOW at a time.

EXPERIMENT 38
DIGITAL-TO-ANALOG
CONVERTERS

a - d) A popular DAC, the DAC0808 (or MC1408) can be wired as follows to produce a DAC unit for this experiment. More parts are necessary and an additional calibration step required.



(NOTE: MC1408 may be used.)

To calibrate this DAC, first connect a 0-10 milliammeter in series with VREF+ and the 0-5kW potentiometer. Adjust the potentiometer for a 1 milliampere reading. Remove the milliammeter and reconnect the potentiometer to pin 14 of the DAC. Now follow the calibration procedures outlined in steps d-g.

Note: Since this set-up yields a negative output from the DAC and a positive output from the op amp, all voltage readings for the calibration will be of opposite polarity from those for the DAC561 units. Also the student should use a positive 0-15V for V_A in Experiment 39.

- i) Model results for Table 38.1 follow. I have included the observations I recorded after calibrating the DAC units. All measurements were made using a Fluke 8010A. The calibration results are easy with the 10-turn potentiometers, but not absolutely necessary for good results. Please note that Figure 38.1 shows an output of V_A . It should be V_A' to match the text.

BCD input	V _{A'} expected	V _{A'} observed
00	-0.0 V	-0.04 V
05	-0.5 V	-0.51 V
10	-1.0 V	-1.03 V
15	-1.5 V	-1.51 V
20	-2.0 V	-2.02 V
25	-2.5 V	-2.50 V
30	-3.0 V	-3.01 V
35	-3.5 V	-3.49 V
40	-4.0 V	-4.04 V
45	-4.5 V	-4.52 V
50	-5.0 V	-5.03 V
55	-5.5 V	-5.51 V
60	-6.0 V	-6.03 V
65	-6.5 V	-6.50 V
70	-7.0 V	-7.02 V
75	-7.5 V	-7.49 V
80	-8.0 V	-8.02 V
85	-8.5 V	-8.50 V
90	-9.0 V	-9.00 V
95	-9.5 V	-9.50 V
99	-9.9 V	-9.88 V

Table 38.1

k) Have the students draw the waveform for only a few steps.

l) Review:

$$1. \text{ % Resolution} = \frac{1}{\# \text{ Steps}} \times 100\% = \frac{1}{99} \times 100\% = 1.01\%.$$

2. You probably would not have been able to achieve the correct settings for Full Scale and Step Size which would make the DVM grossly inaccurate.

EXPERIMENT 39
ANALOG-TO-DIGITAL
CONVERTERS

- a) Example results for comparator test:

V _A	Count	V _{A'}	Output
-5.0	45	-4.51	-0.77 V
-5.0	55	-5.51	+5.05V

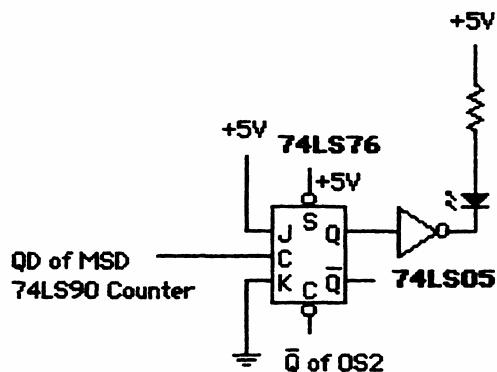
Table 39.1

- c) The resolution of the ADC is 1.01%.

- d) When making conversion time measurements, the student should be encouraged to make sure that their leads are as small as possible and well dressed, and that the signal lines are devoid of noise. ADCs are notoriously sensitive to noise and so should be provided with adequate bypassing, especially at the output of the comparator.

If the pulse of square wave generator is set precisely to 1 kHz, the conversion times for the voltages given in Table 39.2 should be 10 milliseconds/volt, except for -11V. The student should be able to state that this voltage is out of range and so the comparator will never switch. The counter will continue to count however.

- e) Model solution:



f) Review:

1. V_A' must exceed V_A in order for the comparator to switch. Therefore the counters will reflect the higher value of V_A .
2. Since V_A' is negative, V_A must be negative also. Otherwise V_A' can never exceed V_A .

EXPERIMENT 40
SEMICONDUCTOR
RANDOM ACCESS
MEMORY (RAM)

a)

\overline{WE}	\overline{CS}	Operation
0	1	<u>Chip disabled</u> ; I/O lines are in Hi-Z state. No Read or Write operations can take place. Memory contents are unaffected.
1	1	
0	0	WRITE operation; Data on I/O lines is written into location in memory selected by address inputs.
1	0	READ operation; I/O lines contain data word stored at location selected by address inputs.

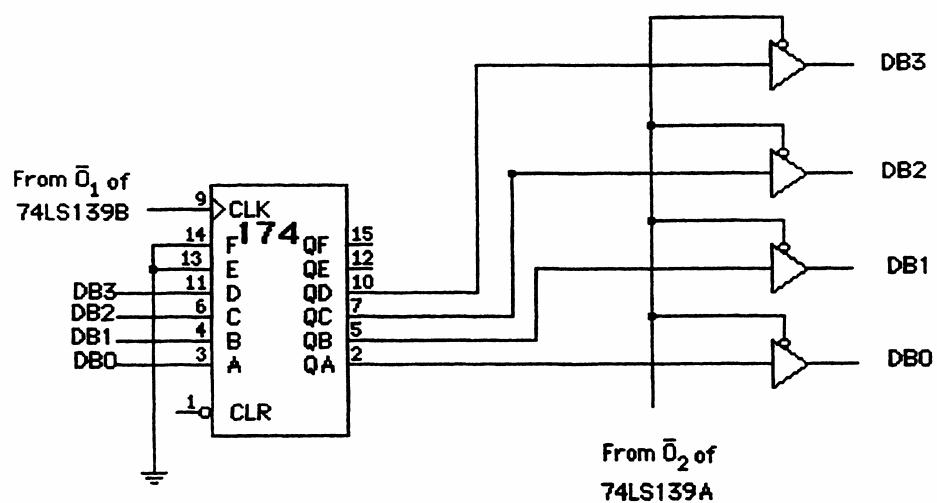
Table 40.1

k)

IS ₁	IS ₀	OS ₁	OS ₀	Selects		Data Transfer	
				To Register	From Register		
0	0	0	0	No data transfer takes place			
0	0	0	1	memory	data switches		
0	1	0	0	data latch	memory		
0	1	0	1	data latch	data switches		

Table 40.3

- l) Modification requires tristate buffers at the outputs of the 74LS174 flip-flops. The tristate buffers can be controlled by the decoder. One way of accomplishing this is illustrated below:



**EXPERIMENT 41
SYNCHRONOUS
COUNTER DESIGN
(OPTIONAL)**

(1)

D	C	B	A
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	0	1	1
1	1	0	0
1	1	1	0
1	1	1	1

Figure 41-1

(2)

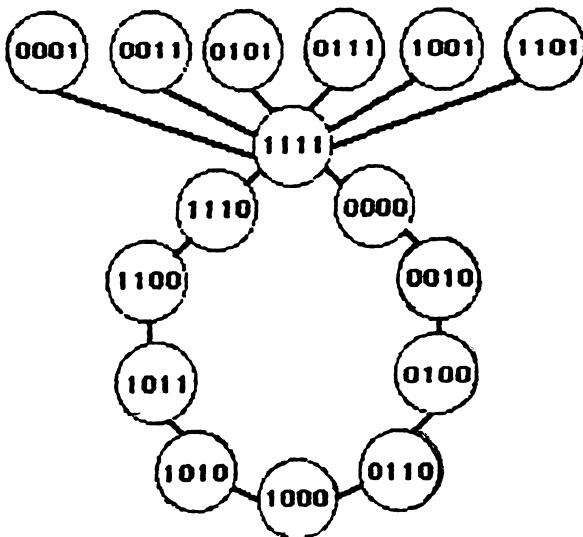


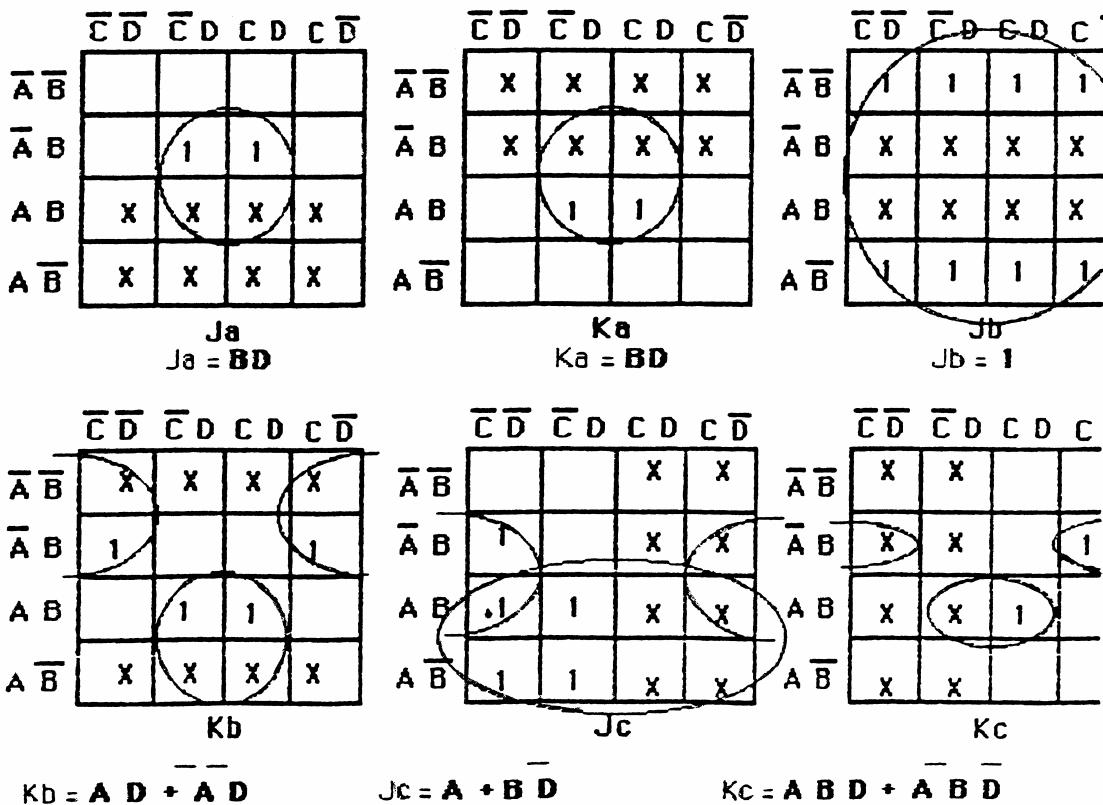
Figure 41-2

(3). (4)

PRESENT States D C B A	NEXT States D C B A	J K Levels				
		Jd Kd	Jc Kc	Jb Kb	Ja	
0 0 0 0	0 0 1 0	0 X	0 X	1 X	0	
1 0 0 1	1 1 1 1	1 X	1 X	1 X	X	
2 0 1 0	0 1 0 0	0 X	1 X	X 1	0	
3 0 0 1 1	1 1 1 1	1 X	1 X	X 0	X	
4 0 1 0 0	0 1 1 0	0 X	X 0	1 X	0	
5 0 1 0 1	1 1 1 1	1 X	X 0	1 X	X	
6 0 1 1 0	1 0 0 0	1 X	X 1	X 1	0	
7 0 1 1 1	1 1 1 1	1 X	X 0	X 0	X	
8 1 0 0 0	1 0 1 0	X 0	0 X	1 X	0	
9 1 0 0 1	1 1 1 1	X 0	1 X	1 X	X	
10 1 0 1 0	1 0 1 1	X 0	0 X	X 0	I	
11 1 0 1 1	1 1 0 0	X 0	1 X	X 1	X	
12 1 1 0 0	1 1 1 0	X 0	X 0	I X	0	
13 1 1 0 1	1 1 1 1	X 0	X 0	1 X	X	
14 1 1 1 0	1 1 1 1	X 0	X 0	X 0	I	
15 1 1 1 1	0 0 0 0	X 1	X 1	X 1	X	

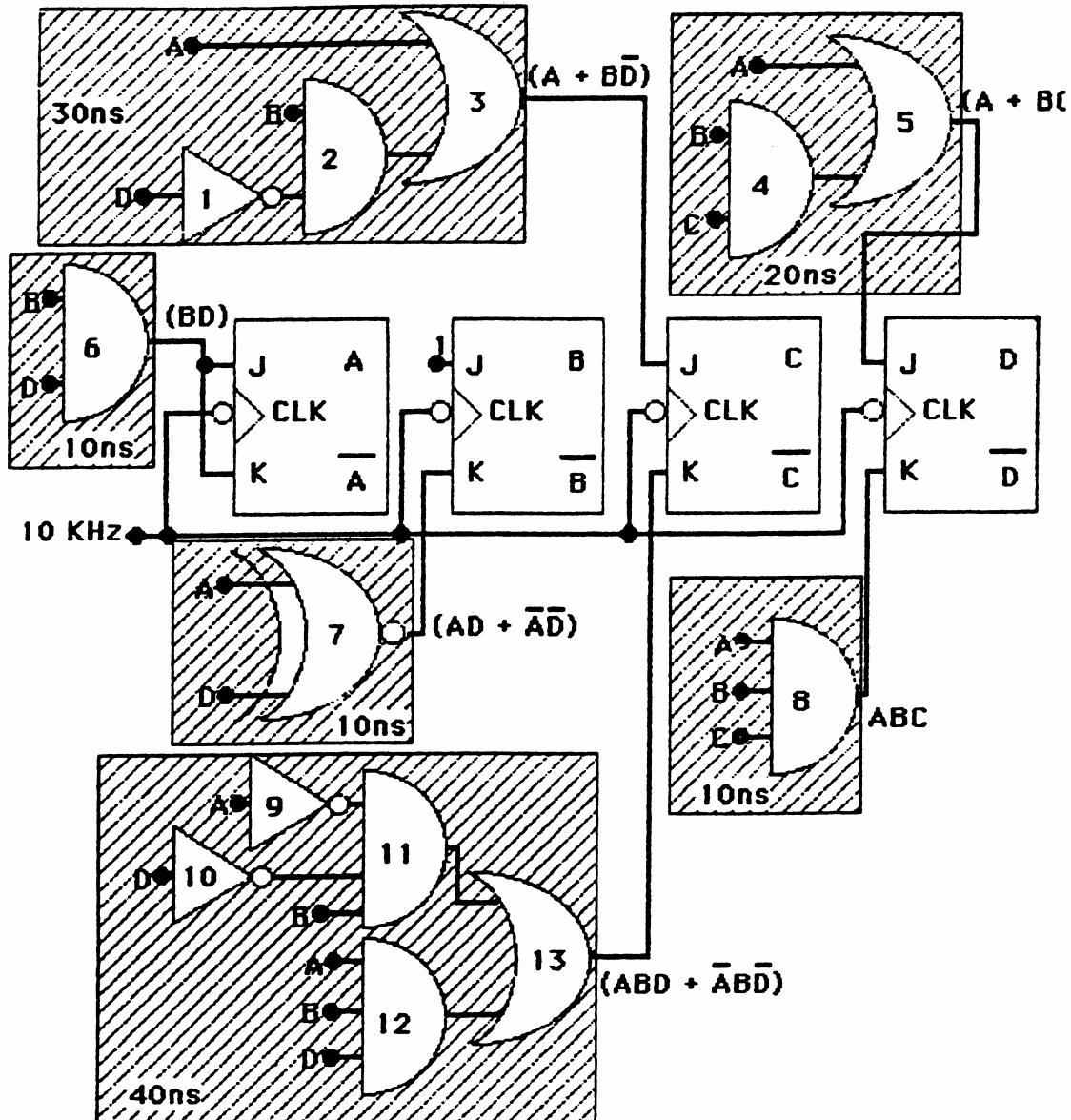
Figure 41-3

(5)



	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$		
$\bar{A}\bar{B}$	X	X				
$\bar{A}B$	X	X	1			
$A\bar{B}$	1	X	X	1		
AB	1	X	X	1		
	Jd				Kd	
$Jd = A + B C$				$Kd = A B C$		

(6)



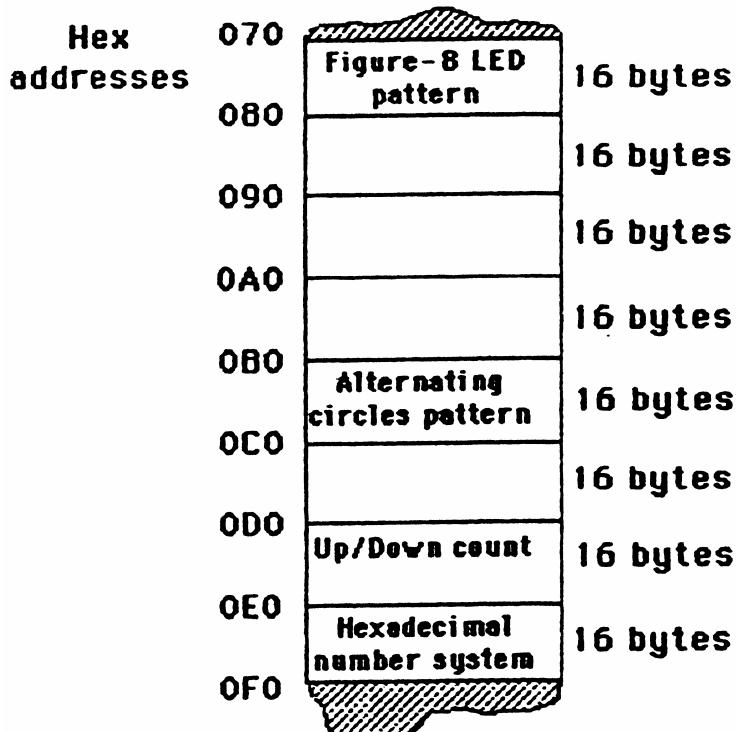
Questions:

1. (a) One advantage is that it eliminates the temporary states that normally exist in serial addition. Also, fmax for the counter is not affected by the number of bits in the counter.
(b) One disadvantage is that it's a much more complex and costly circuit to design and to implement.
2. MOD-10 counter [10 different states]
3. Clk=10KHz; D=1KHz
4. See the shaded areas in the circuit of step 6. Fmax ~18.1MHz. [1/40ns+15ns]

EXPERIMENT 42
PROGRAMMABLE
FUNCTION SEQUENCER
(OPTIONAL)

II.

A7 A6 A5 A4	A3 A2 A1 A0	
1 1 1 0	X X X X	{E0 ₁₆ -EF ₁₆ } (Hex # system)
1 1 0 1	X X X X	{D0 ₁₆ -DF ₁₆ } (Up/Down count)
1 0 1 1	X X X X	{B0 ₁₆ -BF ₁₆ } (Alternating circles LED pattern)
0 1 1 1	X X X X	{70 ₁₆ -7F ₁₆ } (Figure-8 LED pattern)



III.

FIGURE-8 LED PATTERN

Hex Address	EPROM Contents (Hex)
70	01
71	02
72	40
73	10
74	08
75	04
76	40
77	20
78	01
79	02
7A	40
7B	10
7C	08
7D	04
7E	40
7F	20

ALTERNATING CIRCLES LED PATTERN

Hex Address	EPROM Contents (Hex)
B0	01
B1	02
B2	40
B3	20
B4	40
B5	10
B6	08
B7	04
B8	01
B9	02
BA	40
BB	20
BC	40
BD	10
BE	08
BF	04

HEXADECIMAL NUMBER SYSTEM

Hex Address	EPROM Contents (Hex)
E0	3F
E1	06
E2	58
E3	4F
E4	66
E5	6D
E6	5C
E7	07
E8	7F
E9	67
EA	77
EB	7C
EC	39
ED	5E
EE	79
EF	71

UP/DOWN COUNT SEQUENCE

Hex Address	EPROM Contents (Hex)
D0	06
D1	5B
D2	4F
D3	66
D4	6D
D5	5C
D6	07
D7	7F
D8	7F
D9	07
DA	5C
DB	6D
DC	66
DD	4F
DE	5B
DF	06

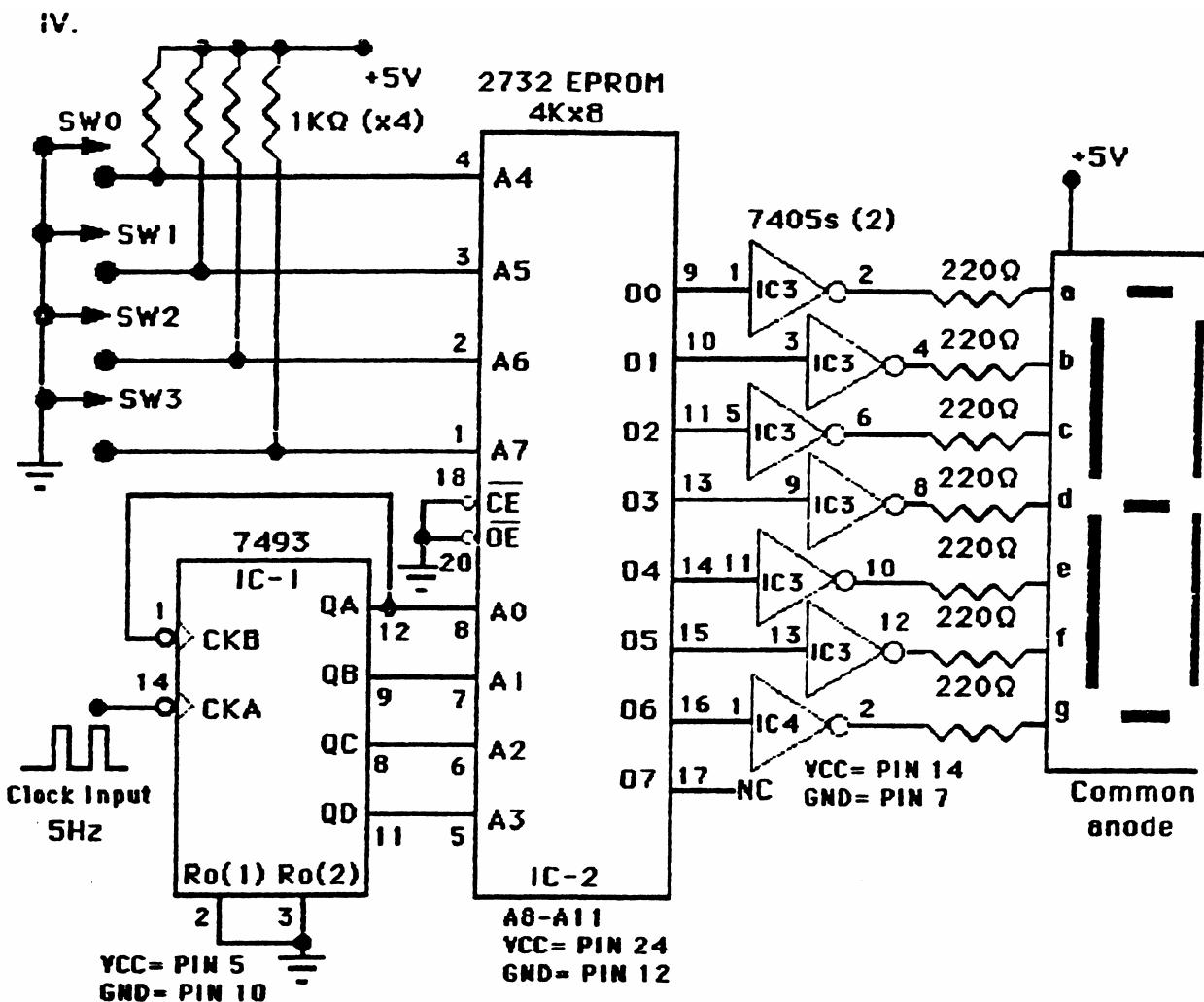


Figure 42-1

QUESTIONS:

- (a) **One possibility:** Use a 3-to-8 line decoder and have the three switches (SW0-SW2) the decoder's inputs. The decoder's outputs O0-O7 would be connected to the address line A11 of the EPROM.
- (b) Figure-8 LED sequence pattern $\{80_{16}-8F_{16}\}$
 Alternating circles LED pattern $\{40_{16}-4F_{16}\}$
 Up/Down count $\{20_{16}-2F_{16}\}$
 Hexadecimal number system $\{10_{16}-1F_{16}\}$
- (C) **One possibility:**
 (a) The LED common lead would have to be connected to Gnd instead of +5V.
 (b) Replace the 7405 open-collector inverters with open-collector buffers.

SUPPLEMENTAL EXPERIMENT 1

Logic Gates: OR, AND, and NOT

Solutions for Part 1 (in Appendix D):

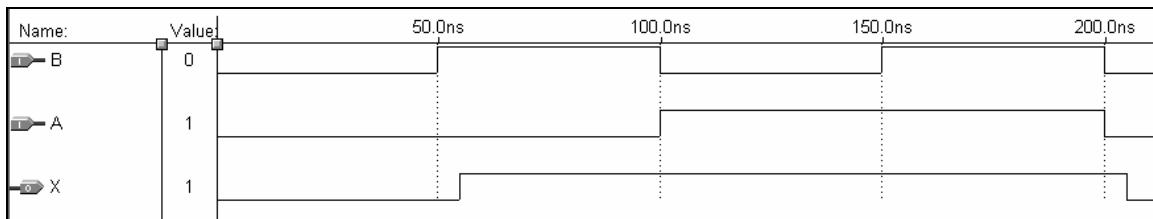


Figure D-31

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table

Solutions for Part 2:

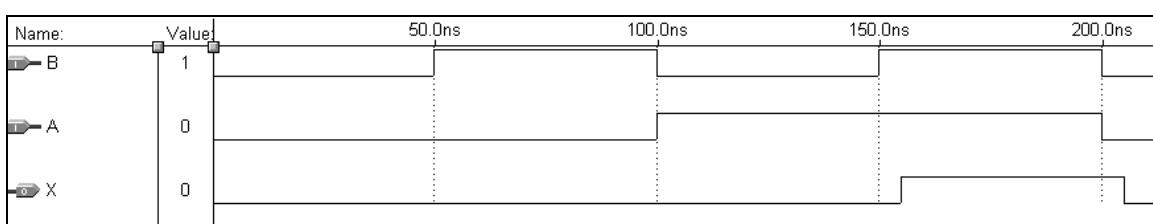


Figure S1-4

SUPPLEMENTAL EXPERIMENT 1

Logic Gates: OR, AND, and NOT

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Table S1-3

q)

1) $X = \underline{\hspace{2cm}} AB \underline{\hspace{2cm}}$

Solutions for Part 3:

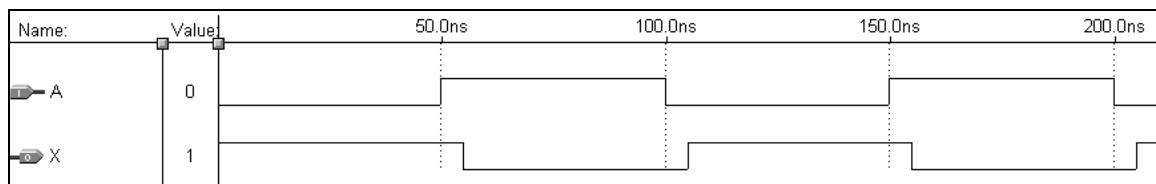


Figure S1 – 4

A	X
0	1
1	0

Table S1-3

q)

SUPPLEMENTAL EXPERIMENT 1

Logic Gates: OR, AND, and NOT

1) $X = \underline{\hspace{2cm}} \bar{A} \underline{\hspace{2cm}}$

SUPPLEMENTAL EXPERIMENT 2

Basic Combinatorial Circuits

Solutions for Part 1:

q)
1) $X = \underline{\hspace{2cm}}AB + C\underline{\hspace{2cm}}$

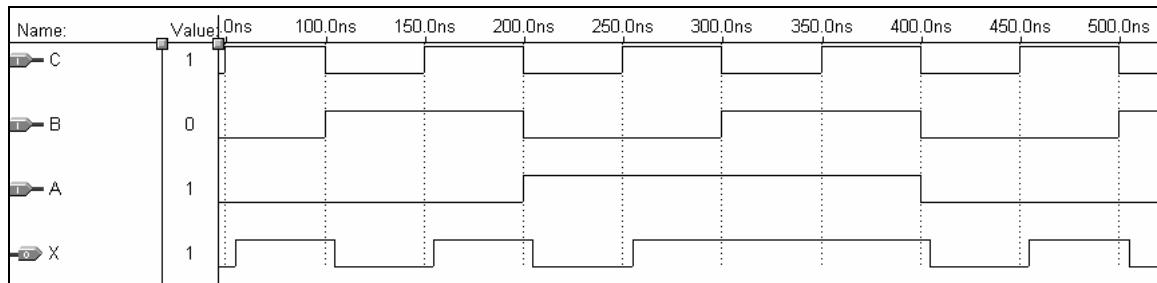


Figure S2 - 2

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table S2-1

Solutions for Part 2:

q)
1) Write the Boolean expression for X:

$$X = \underline{\hspace{2cm}}(A + B)C\underline{\hspace{2cm}}$$

SUPPLEMENTAL EXPERIMENT 2

Basic Combinatorial Circuits

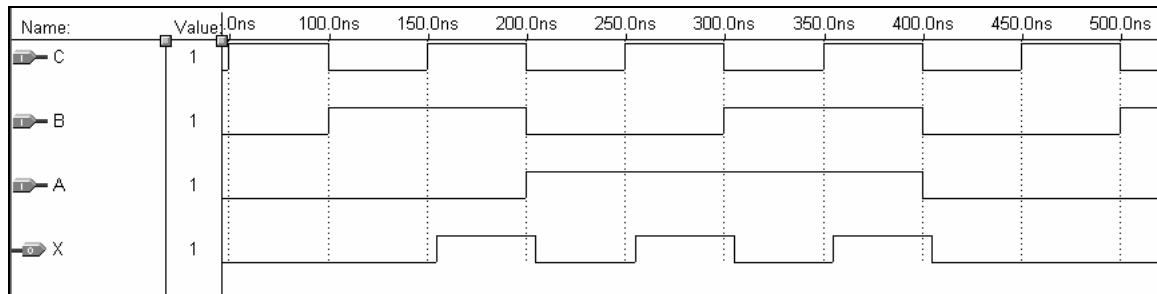


Figure S2-4

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table S2-2

Solutions for Part 3:

q)

- 1) Write the Boolean expression for X:

$$X = \underline{\hspace{2cm}} (\overline{A}\overline{B}C)(\overline{A}+\overline{D}) \underline{\hspace{2cm}}$$

SUPPLEMENTAL EXPERIMENT 2

Basic Combinatorial Circuits

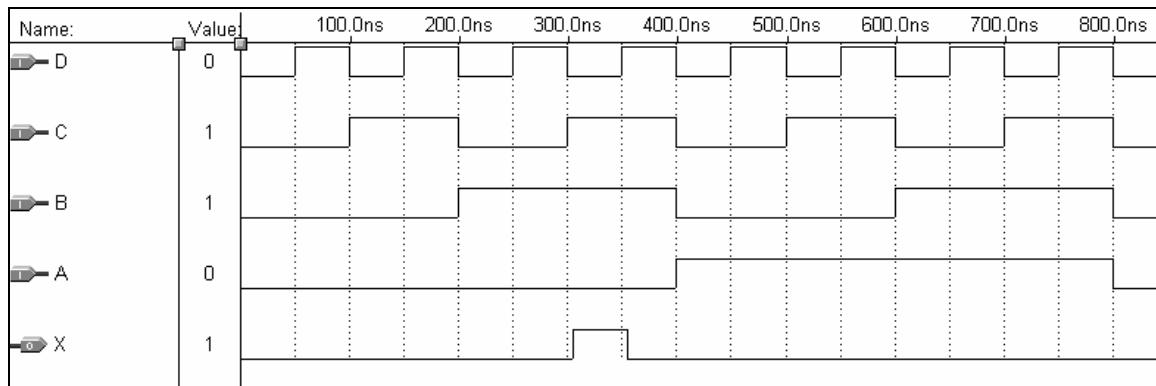


Figure S2-6

A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Table S2-3

SUPPLEMENTAL EXPERIMENT 3

Logic Gates: NOR and NAND

Solutions for Part 1:

q)

1) $X = \underline{\hspace{2cm}} \overline{A + B} \underline{\hspace{2cm}}$

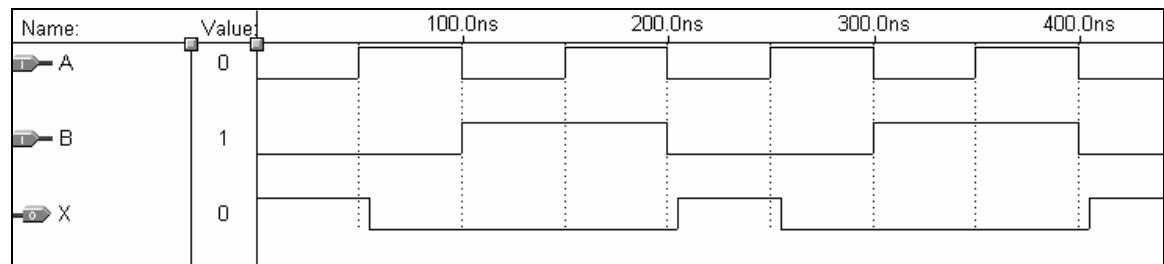


Figure S3 - 2

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Table S3-1

Solutions for Part 2:

q)

1) $X = \underline{\hspace{2cm}} \overline{AB} \underline{\hspace{2cm}}$

SUPPLEMENTAL EXPERIMENT 3

Logic Gates: NOR and NAND

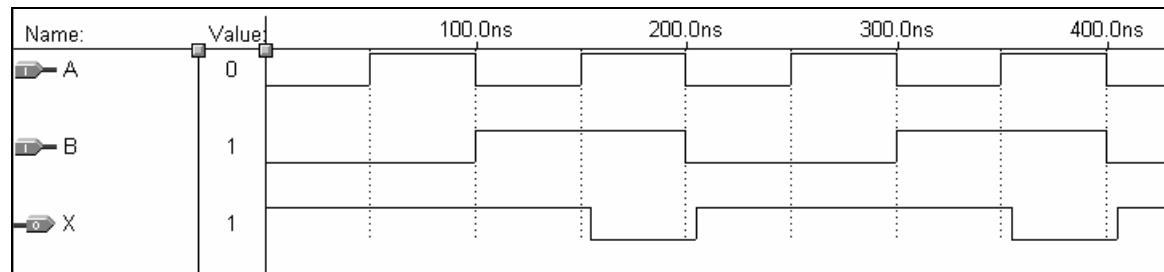


Figure S3-4

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Table S3-2

SUPPLEMENTAL EXPERIMENT 4

Boolean Theorems

Solutions for Part 1:

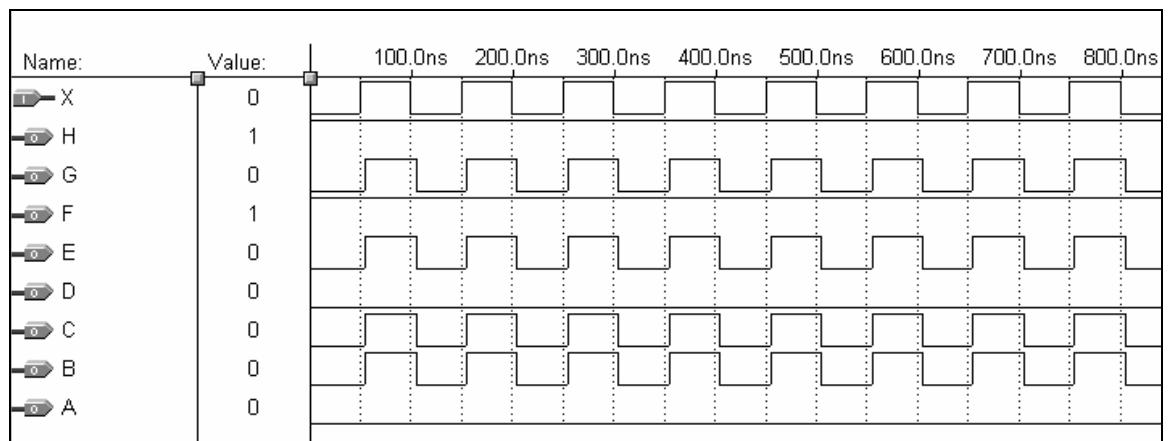


Figure S4 - 2

Theorem	Output	Output verifies theorem?
1	0	yes
2	X	yes
3	X	yes
4	0	yes
5	X	yes
6	1	yes
7	X	yes
8	1	yes

Table S4-1

Solutions for Part 2:

q)

2) Yes

3) Z = _____ z=x+xy=x _____

SUPPLEMENTAL EXPERIMENT 4

Boolean Theorems

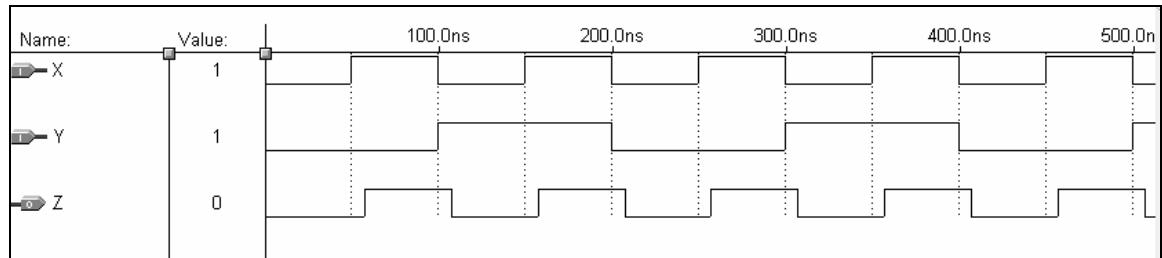


Figure S4-4

X	Y	Z
0	0	0
0	1	0
1	0	1
1	1	1

Table S4-2

Solutions for Part 3:

1)

1) Yes

2) $X = \underline{\hspace{2cm}} z = x + \bar{x}y = x + y \underline{\hspace{2cm}}$

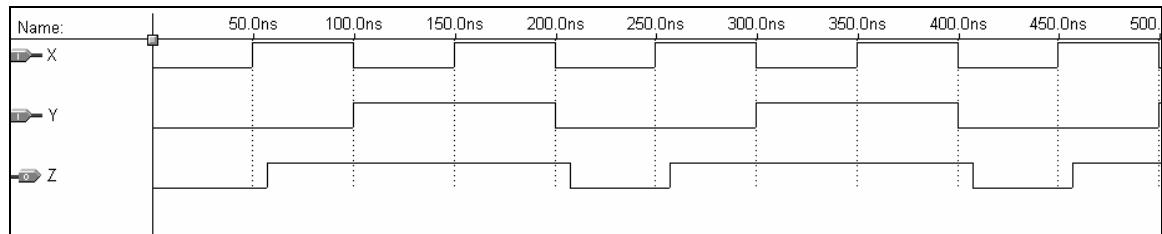


Figure S4-6

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Table S4-3

SUPPLEMENTAL EXPERIMENT 5

Simplification Using Boolean Theorems

Solutions:

A	B	C	D	X
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Table S5-1

b)

2) $X = \underline{\hspace{2cm}} \overline{ABC} + A\overline{BC} + \overline{ABD} \underline{\hspace{2cm}}$

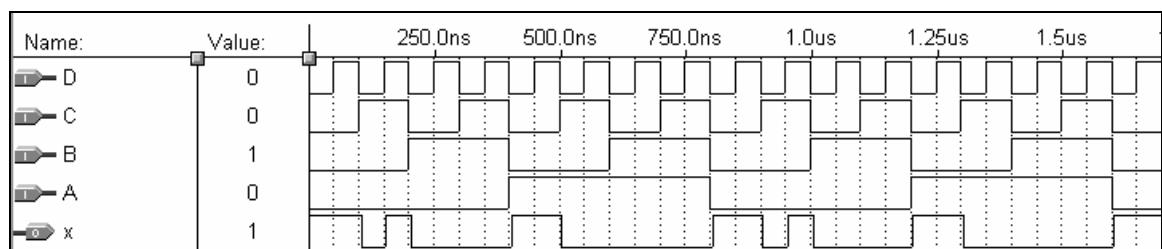


Figure S5 - 2

SUPPLEMENTAL EXPERIMENT 5

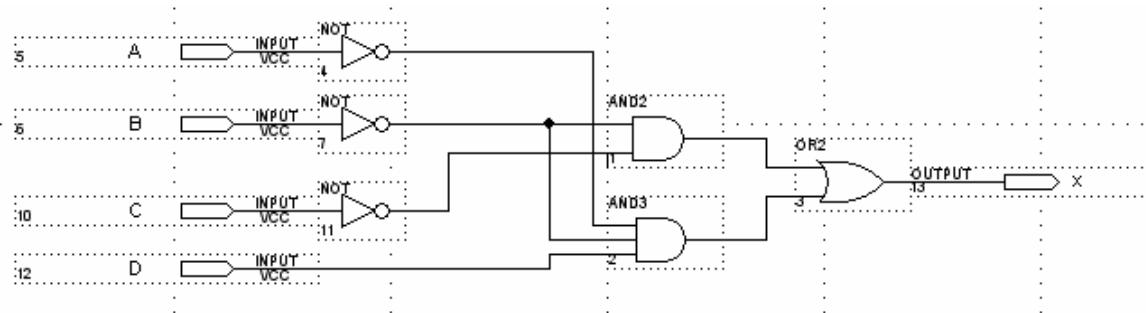
Simplification Using Boolean Theorems

s)

$$\begin{aligned}
 & \overline{ABC} + \overline{ABC} + \overline{ABD} \\
 &= \overline{B}(\overline{AC} + \overline{AC} + \overline{AD}) \quad \text{apply distributive rule} \\
 &= \overline{B}([\overline{AC} + \overline{AC}] + \overline{AD}) \quad \text{apply associative rule} \\
 &= \overline{B}(\overline{\overline{C}}[\overline{A} + A] + \overline{AD}) \quad \text{apply distributive rule} \\
 &= \overline{B}(\overline{C} + \overline{AD}) \quad \overline{A} + A = 1 \\
 &= \overline{BC} + \overline{ABD} \quad \text{apply distributive rule}
 \end{aligned}$$

t) $X = \underline{\overline{BC}} + \underline{\overline{ABD}}$

u)



s)

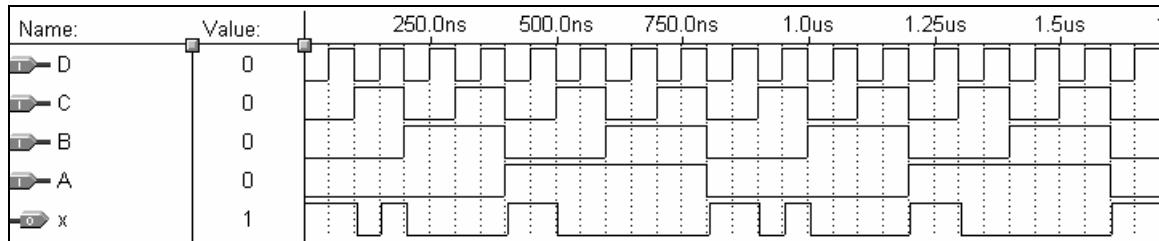


Figure S5-3

- m) 1) The output waveforms of Figures S5-2 and S5-3 are clearly identical. Therefore we can conclude that $\overline{ABC} + \overline{ABC} + \overline{ABD} = \overline{BC} + \overline{ABD}$.

SUPPLEMENTAL EXPERIMENT 5

Simplification Using Boolean Theorems

A	B	C	D	X
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Table S5-2

SUPPLEMENTAL EXPERIMENT 6

DeMorgan's Theorems

Solutions for Part 1:

q)

1) _____ Yes _____

$$\overline{x \cdot y} = \overline{x} + \overline{y}$$

2) _____ Yes _____

$$\overline{x + y} = \overline{x} \cdot \overline{y}$$

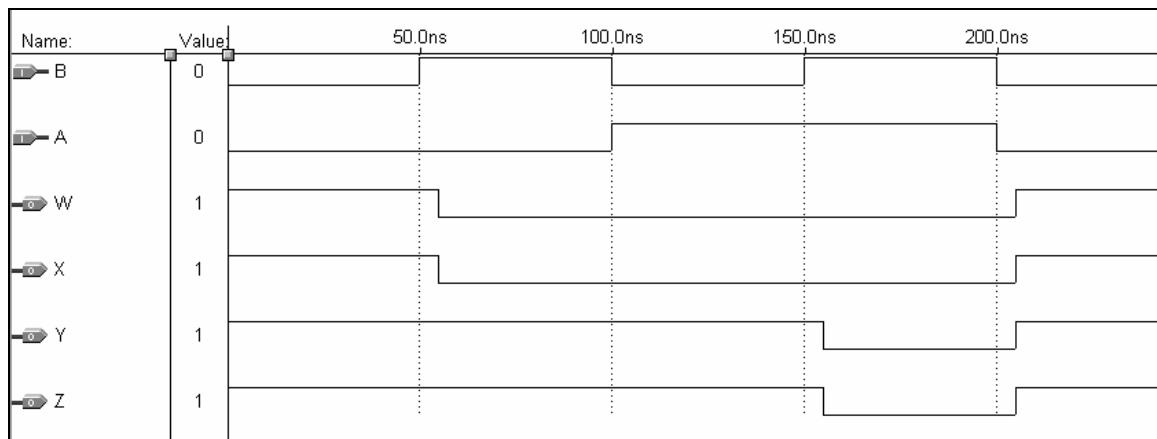


Figure S6 - 2

A	B	W	X
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Table S6-1

SUPPLEMENTAL EXPERIMENT 6

DeMorgan's Theorems

A	B	Y	Z
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Table S6-2

Data collection for Part 2:

b) $X = \underline{\hspace{2cm}} (\overline{A} + \overline{B}C)(\overline{A} + \overline{B}\overline{C}) \underline{\hspace{2cm}}$

$$\begin{aligned}
 & (\overline{A} + \overline{B}C)(\overline{A} + \overline{B}\overline{C}) \\
 &= \overline{\overline{ABC}} \overline{\overline{ABC}} && \text{using DeMorgan 1} \\
 &= \overline{(A(B+C))} \overline{(A(B+C))} && \text{using DeMorgan 2} \\
 &= \overline{(AB+A\overline{C})} \overline{(AB+A\overline{C})} && \text{using Distributive Law} \\
 &= \overline{(AB+A\overline{C})} + \overline{(AB+A\overline{C})} && \text{using DeMorgan 1} \\
 \text{s)} &= \overline{AB+A\overline{C}} + \overline{AB+A\overline{C}} && \text{using Associative Rule} \\
 &= \overline{AB+A\overline{C}} + \overline{AC} && \text{using } AB+AB=AB \\
 &= \overline{AB+(A\overline{C}+AC)} && \text{using Associative Rule} \\
 &= \overline{AB+A(\overline{C}+C)} && \text{using Distributive Rule} \\
 &= \overline{AB+A} && \text{using } \overline{C}+C=1 \\
 &= \overline{A} && \text{using } AB+A=A \text{ (Theorem 14)}
 \end{aligned}$$

t) $X = \overline{A}$

u) Yes.

SUPPLEMENTAL EXPERIMENT 6

DeMorgan's Theorems

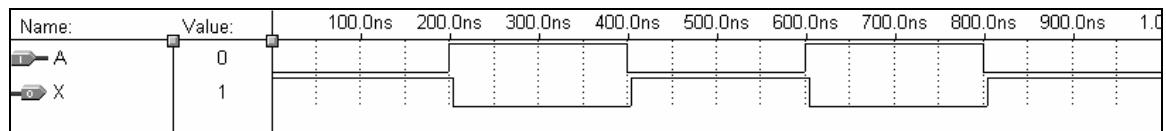


Figure S6-4

A	B	C	X	Simplified X
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

Table S6-3

SUPPLEMENTAL EXPERIMENT 7

Implementing Digital Gates and Circuits Using VHDL

Solutions for Part 1:

Data collection for Part 2:

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

Table S7-1

Data collection for Part 3:

a	y
0	1
1	0

Table S7-2

Data collection for Part 4:

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

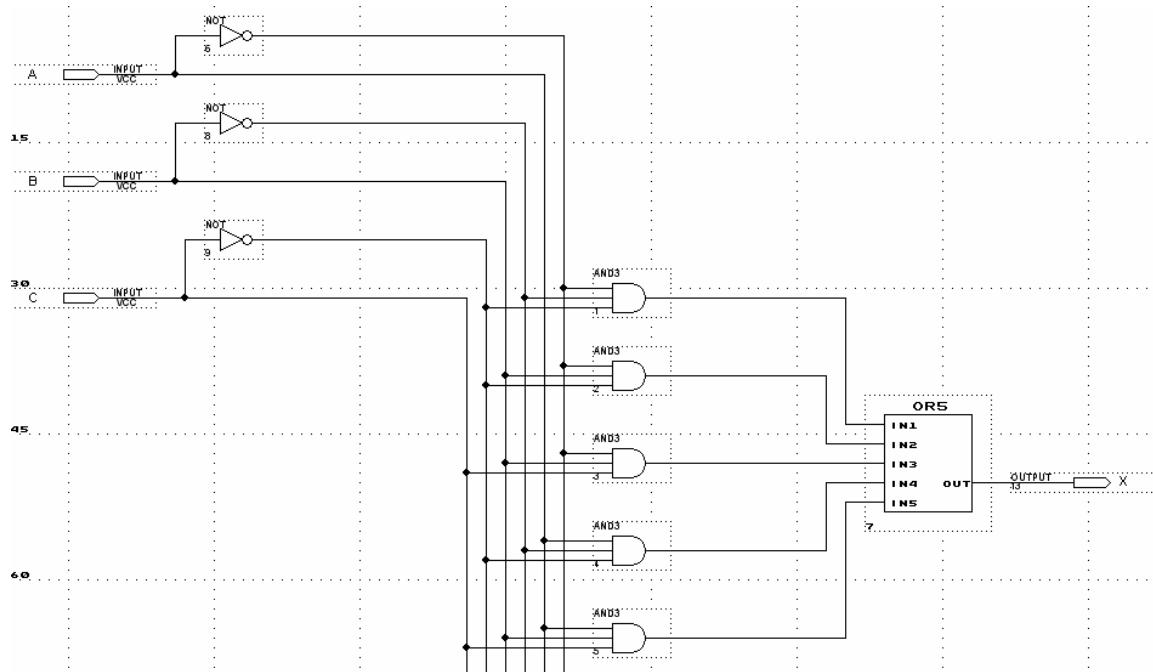
Table S7-3

SUPPLEMENTAL EXPERIMENT 8

Implementing Logic Circuit Designs

Solutions for Part 1:

a) one possible circuit appears below:



A	B	C	X	X after simulation
0	0	0	1	1
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table S8-1

SUPPLEMENTAL EXPERIMENT 8

Implementing Logic Circuit Designs

b) $X = \overline{ABC} + \overline{ABC} + \overline{ABC} + A\overline{BC} + ABC$

t) Yes

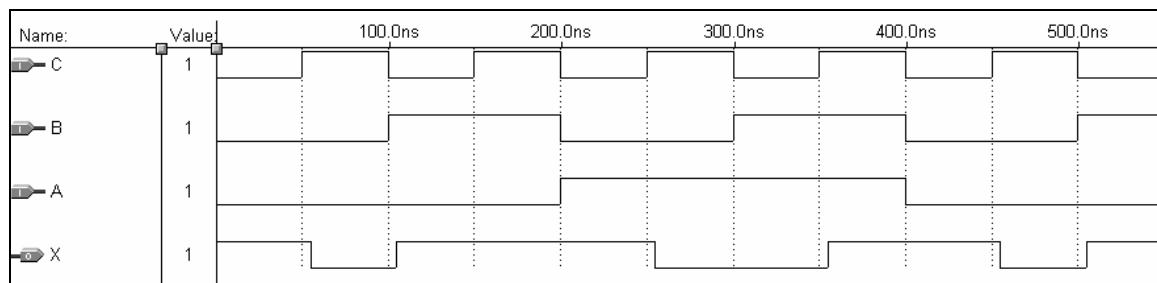


Figure S8 - 1

Solutions for Part 2:

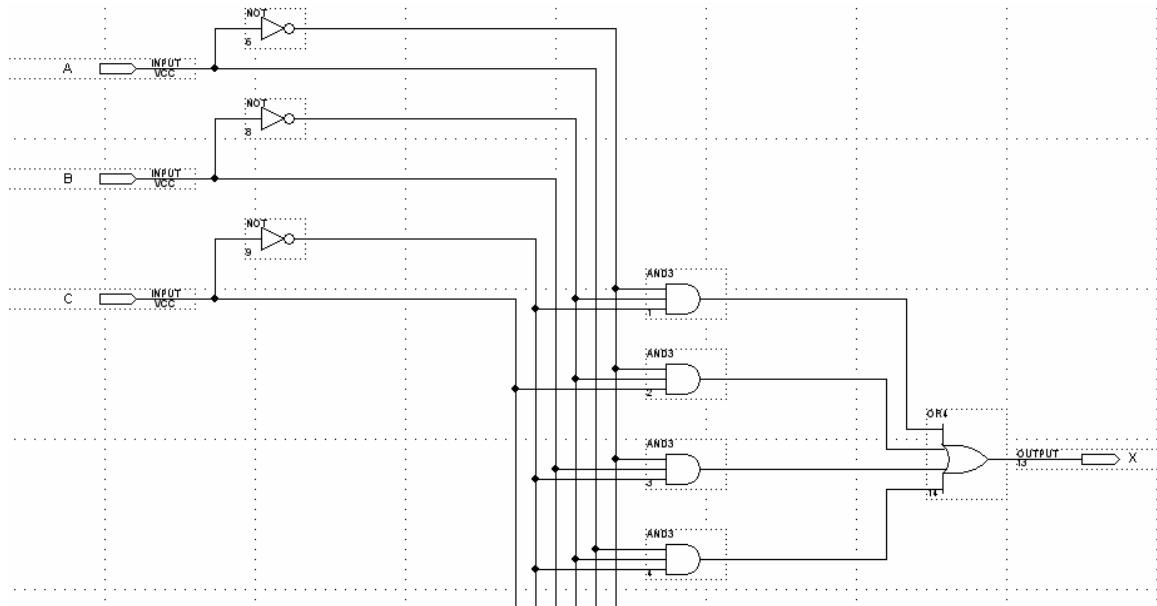
A	B	C	X	X after simulation
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	0

Table S8-2

SUPPLEMENTAL EXPERIMENT 8

Implementing Logic Circuit Designs

One possible circuit appears below:



a) $X = \overline{ABC} + \overline{ABC} + \overline{ABC} + ABC$

q)
3) ___ Yes ___

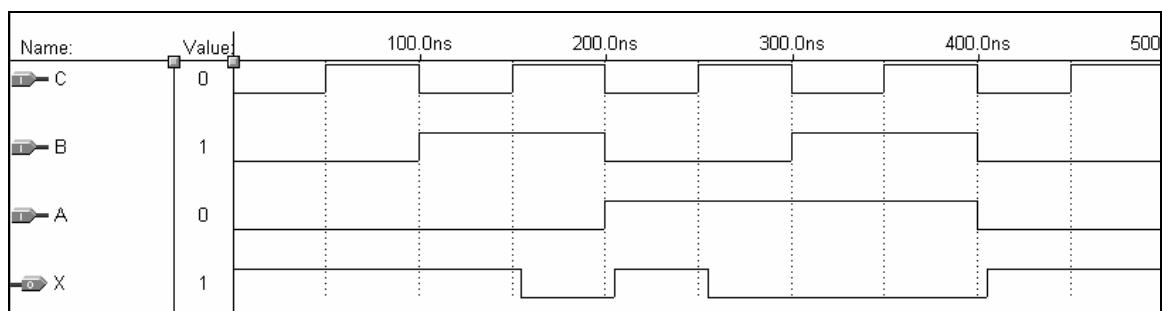


Figure S8 - 2

SUPPLEMENTAL EXPERIMENT 9

Exclusive-OR and Exclusive-NOR Circuits

Solutions for Part 1:

p)

- 2) Yes
3) $X = \underline{\quad} A \oplus B \underline{\quad}$ $Y = \underline{\quad} A \oplus B \underline{\quad}$

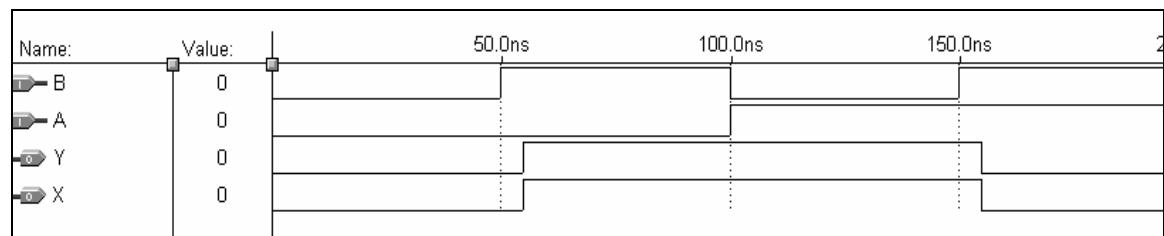


Figure S9 - 2

A	B	X	Y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Table S9-1

Solutions for Part 2:

p)

- 4) Yes
5) $X = \underline{\quad} \overline{A \oplus B} \underline{\quad}$ $Y = \underline{\quad} \overline{A \oplus B} \underline{\quad}$

SUPPLEMENTAL EXPERIMENT 9

Exclusive-OR and Exclusive-NOR Circuits

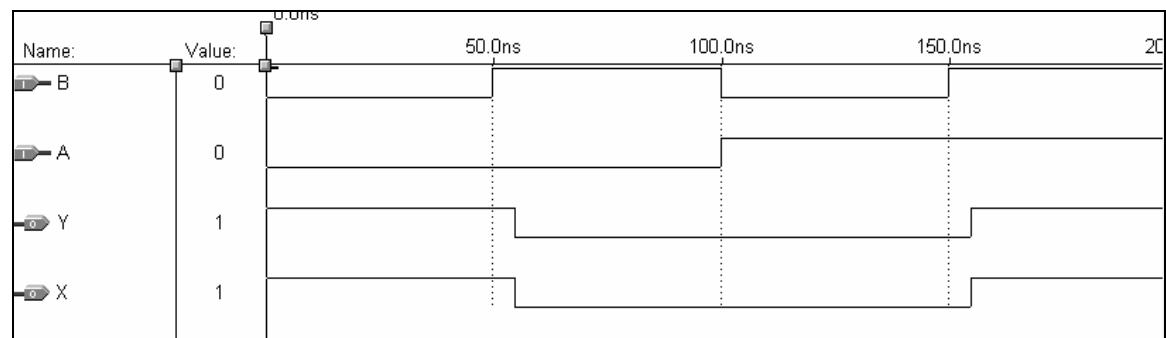


Figure S9 - 4

A	B	X	Y
0	0	1	1
0	1	0	0
1	0	0	0
1	1	1	1

Table S9-2

SUPPLEMENTAL EXPERIMENT 10

Designing With Exclusive-OR and Exclusive-NOR Circuits

Solutions for Part 1:

s) _____ Yes _____

t) $X = \underline{\hspace{2cm}} (A \oplus B) \oplus C \underline{\hspace{2cm}}$

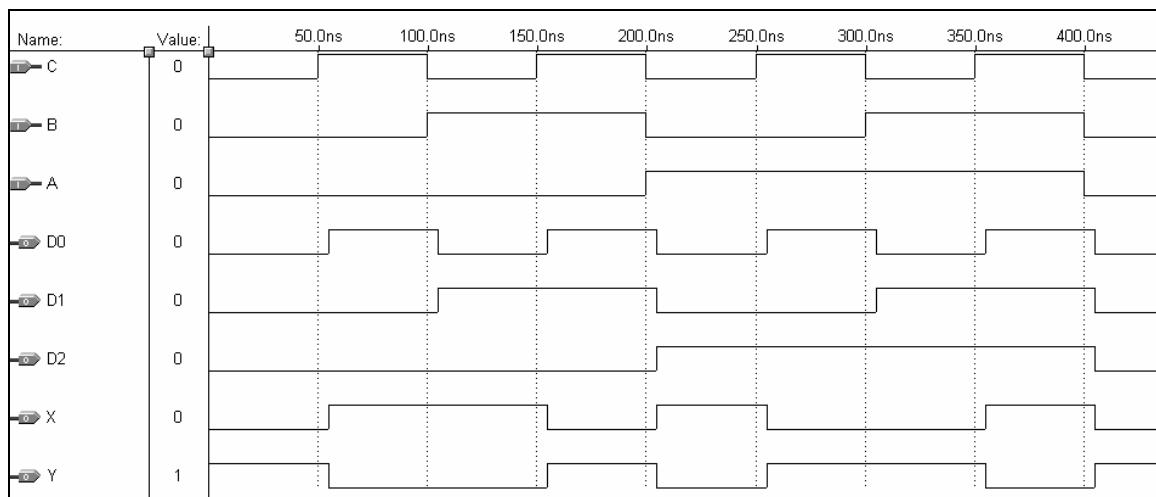


Figure S10 – 2

X	D2	D1	D0
0	0	0	0
1	0	0	1
1	0	1	0
0	0	1	1
1	1	0	0
0	1	0	1
0	1	1	0
1	1	1	1

Table S10-1a: EVEN Parity

Y	D2	D1	D0
1	0	0	0
0	0	0	1
0	0	1	0
1	0	1	1
0	1	0	0
1	1	0	1
1	1	1	0
0	1	1	1

Table S10-1b: ODD Parity

SUPPLEMENTAL EXPERIMENT 10

Designing With Exclusive-OR and Exclusive-NOR Circuits

Time Intervals	X	D2	D1	D0
0-50ns	0	0	0	0
50ns-100ns	1	0	0	1
100ns-150ns	1	0	1	0
150ns-200ns	0	0	1	1
200ns-250ns	1	1	0	0
250ns-300ns	0	1	0	1
300ns-350ns	0	1	1	0
350ns-400ns	1	1	1	1

Table S10-2a

Time Intervals	Y	D2	D1	D0
0-50ns	1	0	0	0
50ns-100ns	0	0	0	1
100ns-150ns	0	0	1	0
150ns-200ns	1	0	1	1
200ns-250ns	0	1	0	0
250ns-300ns	1	1	0	1
300ns-350ns	1	1	1	0
350ns-400ns	0	1	1	1

Table S10-2b

Solutions for Part 2:

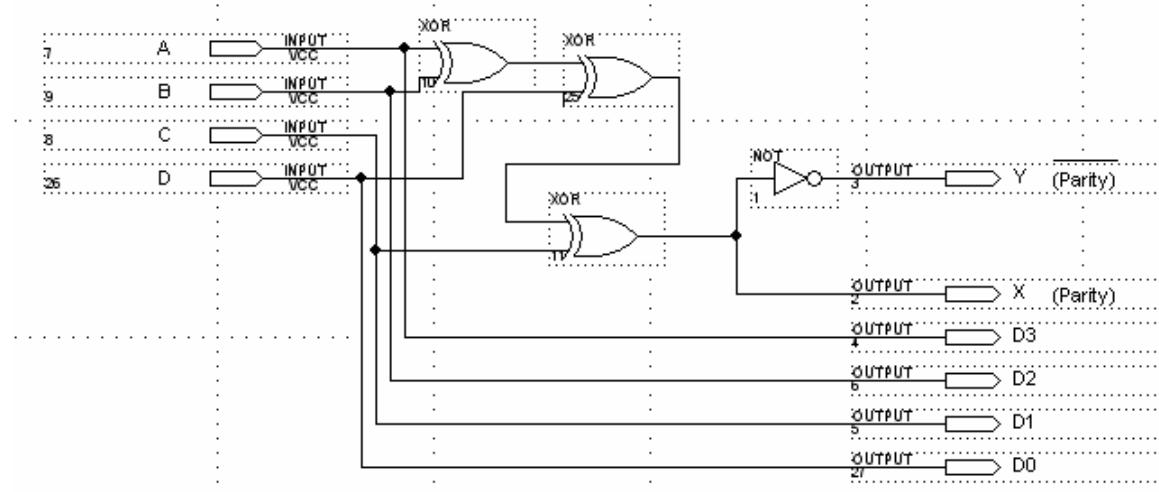
A	B	C	D	X	Y
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

Table S10-3

SUPPLEMENTAL EXPERIMENT 10

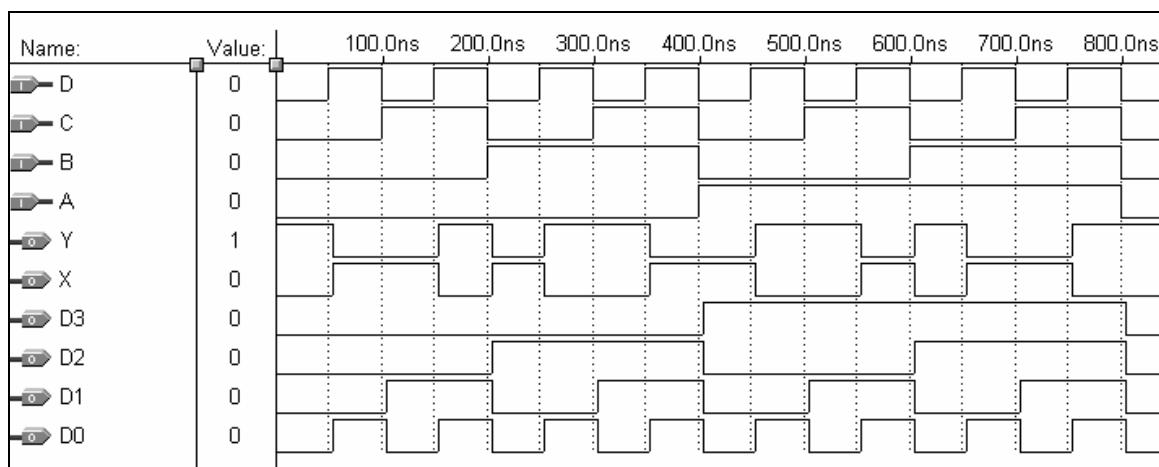
Designing With Exclusive-OR and Exclusive-NOR Circuits

b)



c) $X = \underline{\hspace{10em}} (((A \oplus B) \oplus C) \oplus D) \underline{\hspace{10em}}$

u) Yes
 v) $X = \underline{\hspace{10em}} (((A \oplus B) \oplus C) \oplus D) \underline{\hspace{10em}}$



SUPPLEMENTAL EXPERIMENT 10

Designing With Exclusive-OR and Exclusive-NOR Circuits

Time Intervals	X	D3	D2	D1	D0
0-50ns	0	0	0	0	0
50ns-100ns	1	0	0	0	1
100ns-150ns	1	0	0	1	0
150ns-200ns	0	0	0	1	1
200ns-250ns	1	0	1	0	0
250ns-300ns	0	0	1	0	1
300ns-350ns	0	0	1	1	0
350ns-400ns	1	0	1	1	1
400ns-450ns	1	1	0	0	0
450ns-500ns	0	1	0	0	1
550ns-600ns	0	1	0	1	0
600ns-650ns	1	1	0	1	1
650ns-700ns	0	1	1	0	0
700ns-750ns	1	1	1	0	1
750ns-800ns	1	1	1	1	0
800ns-850ns	0	1	1	1	1

Time Intervals	Y	D3	D2	D1	D0
0-50ns	1	0	0	0	0
50ns-100ns	0	0	0	0	1
100ns-150ns	0	0	0	1	0
150ns-200ns	1	0	0	1	1
200ns-250ns	0	0	1	0	0
250ns-300ns	1	0	1	0	1
300ns-350ns	1	0	1	1	0
350ns-400ns	0	0	1	1	1
400ns-450ns	0	1	0	0	0
450ns-500ns	1	1	0	0	1
550ns-600ns	1	1	0	1	0
600ns-650ns	0	1	0	1	1
650ns-700ns	1	1	1	0	0
700ns-750ns	0	1	1	0	1
750ns-800ns	0	1	1	1	0
800ns-850ns	1	1	1	1	1

Figure S10 -3

Table S10-4a

Table S10-4b

Solutions for Part 3:

m)

2) X ($\overline{A_0 \oplus B_0}$) • ($\overline{A_1 \oplus B_1}$)

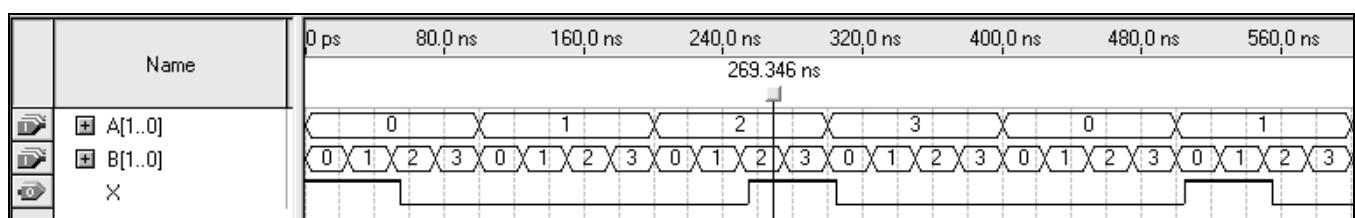


Figure S10 – 6

SUPPLEMENTAL EXPERIMENT 10

Designing With Exclusive-OR and Exclusive-NOR Circuits

Time Intervals	X	A1	A0	B1	B0
0-50ns	1	0	0	0	0
50ns-100ns	0	0	0	0	1
100ns-150ns	0	0	0	1	0
150ns-200ns	0	0	0	1	1
200ns-250ns	0	0	1	0	0
250ns-300ns	1	0	1	0	1
300ns-350ns	0	0	1	1	0
350ns-400ns	0	0	1	1	1
400ns-450ns	0	1	0	0	0
450ns-500ns	0	1	0	0	1
550ns-600ns	1	1	0	1	0
600ns-650ns	0	1	0	1	1
650ns-700ns	0	1	1	0	0
700ns-750ns	0	1	1	0	1
750ns-800ns	0	1	1	1	0
800ns-850ns	1	1	1	1	1

Table S10-5

SUPPLEMENTAL EXPERIMENT 11

Implementing Exclusive-OR and Exclusive-NOR Circuits Using VHDL

Solutions for Part 1:

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

Table S11-1

Solutions for Part 2:

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Table S11-2

Solutions for Part 3:

A	B	Less	Equal	Greater
0	0	0	1	0
1	2	1	0	0
2	1	0	0	1
3	4	1	0	0
4	2	0	0	1
5	6	1	0	0
6	5	0	0	1
7	7	0	1	0

Table S11-3

SUPPLEMENTAL EXPERIMENT 12

Latches and D-Type Flip-Flops

Solutions for Part 1:

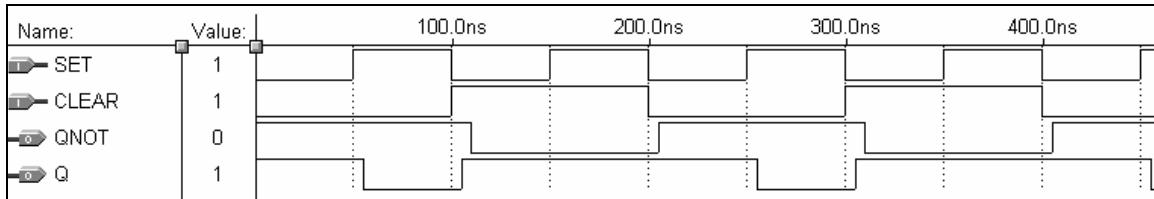


Figure S12 - 2

SET	RESET	Q	QNOT	
0	0	1	1	invalid
1	0	0	1	
0	1	1	0	
1	1	NC	NC	

Table S12-1a

SET	RESET	Q	QNOT	
0	0	1	1	invalid
1	0	0	1	
0	1	1	0	
1	1	NC	NC	

Table S12-1b

Solutions for Part 2:

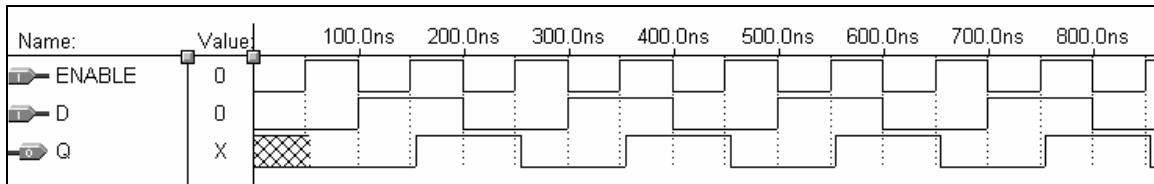


Figure S12-4

SUPPLEMENTAL EXPERIMENT 12

Latches and D-Type Flip-Flops

ENABLE	D	Q
0	0	NC
0	1	NC
1	0	0
1	1	1

Table S12-2

ENABLE	D	Q
0	0	NC
0	1	NC
1	0	0
1	1	1

Table S12-2a

Solutions for Part 3:

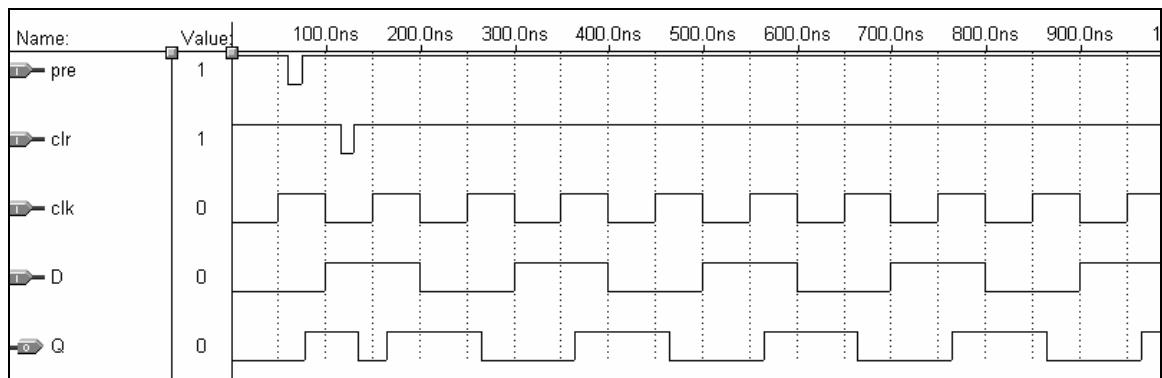


Figure S12-6

Preset	Clear	Clock	D	Q
1	1	↑	0	0
1	1	↑	1	1
0	1	x	x	1
1	0	x	x	0

Table 12-3

SUPPLEMENTAL EXPERIMENT 12

Latches and D-Type Flip-Flops

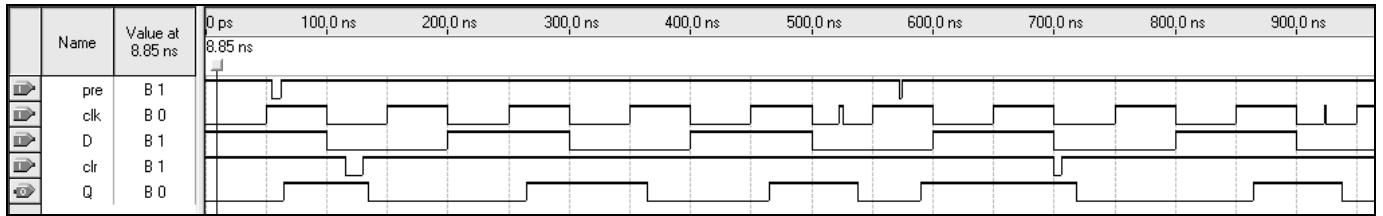


Figure 12-7

bb) Analyze the results:

[Note: The procedure in the lab manual does not ask the student to check for “glitches” and define the glitch interval. This is done in **Assignments | Settings | Simulator**. If you want the student to check for glitches, get the student to exercise care in creating and placing the “glitch”.]

- 1) You should have gotten several warnings from the simulator. In general, what were the warnings? In general, the warnings indicate clock high time violations and glitches were uncovered (if you are checking for glitches).
- 2) Compare the Q outputs for this simulation against that of the simulation done in step k (see Figure 12-6). Generally, what is different? The noise spikes on the clock, preset, and clear causes Q to change to a value depending on the value of D when the spike occurs rather than at the next positive clock transition as in Figure S12-6.
- 3) What did the noise spike on the “clk” signal (near 550ns) do to the Q waveform? It caused Q to go LOW before the next positive clock transition.
- 4) What did the noise spike on the “pre” signal (near 600ns) do to the Q waveform? It caused Q to go HIGH before the next positive clock transition.
- 5) What did the noise spike on the “clr” signal (near 700ns) do to the Q waveform? It caused Q to go LOW before the next positive clock transition.
- 6) What did the noise spike on the “clk” signal (near 900ns) do to the Q waveform? It caused Q to go LOW before the next positive clock transition.

SUPPLEMENTAL EXPERIMENT 12

Latches and D-Type Flip-Flops

Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed paths
Worst-case tsu	N/A	None	4 ns	D	1		Clk	0
Worst-case tco	N/A	None	15 ns	1	Clk			0
Worst-case th	N/A	None	4 ns	D	1		Clk	0
Clock Setup: 'clk'	N/A	None	NA	NA	NA	NA	NA	NA
Total number of failed paths								0

Table S12-4

Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed paths
Worst-case tsu	15 ns	20.000 ns	4 ns	D	1		Clk	0
Worst-case tco	5 ns	20.000 ns	15 ns	1	Clk			0
Worst-case th	6 ns	10.000 ns	4 ns	D	1		Clk	0
Clock Setup: 'clk'	NA	NA	NA	NA	NA	NA	NA	NA
Clock Hold: 'clk'	NA	NA	NA	NA	NA	NA	NA	NA
Total number of failed paths								0

Table S12-5

[Note: Worst-case th Required Time in Table S12-5 was given as 0 ns in the Lab Manual. In this case, the flip-flop would not meet the requirement.]

i)

- 1) If the Q output is HIGH, clear the flip-flop by momentarily pulsing the “clr” input LOW.
- 2) Set the J input HIGH and K to LOW. Momentarily pulse the “clk” (if using external or on-board switch) input. $Q = \underline{\hspace{2cm}}1\underline{\hspace{2cm}}$.
- 3) Set the J input LOW. Momentarily pulse the “clk” (if using external or on-board switch) input. $Q = \underline{\hspace{2cm}}0\underline{\hspace{2cm}}$.
- 4) Set the K input HIGH. Momentarily pulse the “clk” (if using external or on-board switch) input.
- 5) Momentarily pulse the “pre” input. $Q = \underline{\hspace{2cm}}1\underline{\hspace{2cm}}$.
- 6) Set the J input HIGH. Momentarily pulse the “clk” input (if using external or on-board switch).

SUPPLEMENTAL EXPERIMENT 12

Latches and D-Type Flip-Flops

Preset	Clear	Clock	D	Q
1	1	↑	0	0
1	1	↑	1	1
0	1	x	x	1
1	0	x	x	0

Table S12-4

Solutions for Part 4:

p)

- 1) Set the D input HIGH. Momentarily pulse the “pb3” input. Q = 1.
- 2) Set the D input LOW. Momentarily pulse the “pb3” input. Q = 0.
- 3) Momentarily pulse the “pre” input. Q = 1.

pre	clr	clk	D	Q
1	1	↑	0	0
1	1	↑	1	1
0	1	x	x	1
1	0	x	x	0

Table S12-5

SUPPLEMENTAL EXPERIMENT 13

J-K and T-Type Flip-Flops

Solutions for Part 1:

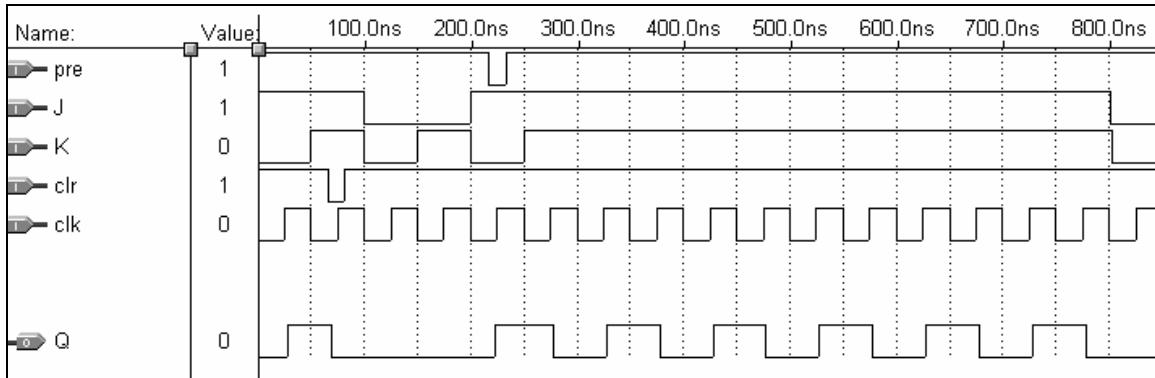


Figure S13-2

pre	clr	clk	J	K	Q
1	1	↑	0	0	NC
1	1	↑	1	0	1
1	1	↑	0	1	0
1	1	↑	1	1	Toggle
0	1	↑	x	x	1
1	0	↑	x	x	0

Table S13-1

[Note: Table S13-1 of the data collection sheet should be changed to agree with the table above.]

Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed paths
Worst-case tsu	N/A	None	11.000 ns	K	7		clk	0
Worst-case tco	N/A	None	8.000 ns	7	Q	clk		0
Worst-case th	N/A	None	-3.000 ns 76.92 MHz (period = 13.000 ns)	K	7		clk	0
Clock Setup: 'clk'	N/A	None			7	7	clk	0
Clock Hold: 'clk'	N/A	None	N/A		7	7	clk	0
Total number of failed paths								0

Figure S13-3

SUPPLEMENTAL EXPERIMENT 13

J-K and T-Type Flip-Flops

Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed paths
Worst-case tsu	9 ns	20 ns	11.000 ns	K	7		clk	0
Worst-case tco	12 ns	20 ns	8.000 ns	7	Q	clk		0
Worst-case th	3 ns	0 ns	-3.000 ns	K	7		clk	0
Clock Setup: 'clk'	20.333 ns	30 MHz	76.92 MHz	7	7	clk	clk	0
Clock Hold: 'clk'	5.000 ns	30 MHz	N/A	7	7	clk	clk	0
Total number of failed paths								0

Figure S13-4

- y) If the Q output is HIGH, clear the flip-flop by momentarily pulsing the “clr” input LOW.
1. Set the J input HIGH and K to LOW. Momentarily pulse the “clk” (if using external or on-board switch) input. Q = 1 .
 2. Set the J input LOW. Momentarily pulse the “clk” (if using external or on-board switch) input. Q = NC .
 3. Set the K input HIGH. Momentarily pulse the “clk” (if using external or on-board switch) input. Q = 0 .
 4. Momentarily pulse the “pre” input. Q = 1 .

pre	clr	clk	J	K	Q
1	1	↑	0	0	NC
1	1	↑	1	0	1
1	1	↑	0	1	0
1	1	↑	1	1	Toggle
0	1	↑	x	x	1
1	0	↑	x	x	0

Table S23-2

[Note: Table S13-2 of the data collection sheet should be changed to agree with the table above.]

SUPPLEMENTAL EXPERIMENT 13

J-K and T-Type Flip-Flops

Solutions for Part 2:

v)

- 1) What is the simulated frequency of “clk”: $f_{CLK} = 10 \text{ MHz}$
- 2) Compute the simulated frequency of Q: $F_Q = 5 \text{ MHz}$
- 3) The relationship between f_{CLK} and F_Q is $F_Q = 0.5x f_{CLK}$
- 4) Repeat 1) – 4) for the programmed CPLD.
- 5) $f_{CLK} = 10 \text{ MHz} \text{ (typical)}$
- 6) Measure the frequency of Q at PIN_4: $F_Q = 5 \text{ MHz} \text{ (typical)}$
- 7) The relationship between f_{CLK} and F_Q is $F_Q = 0.5 \times f_{CLK} \text{ (typical)}$
- 8) Are the results of 3) and 7) the same? _____ Yes _____

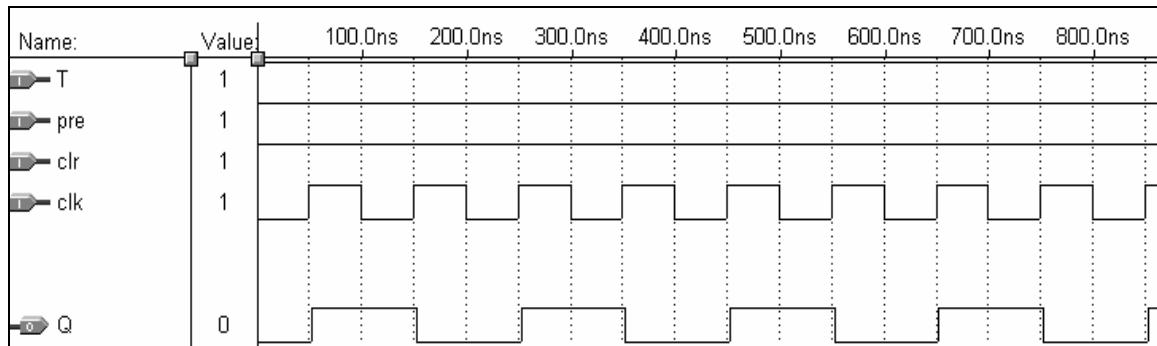


Figure S13-5

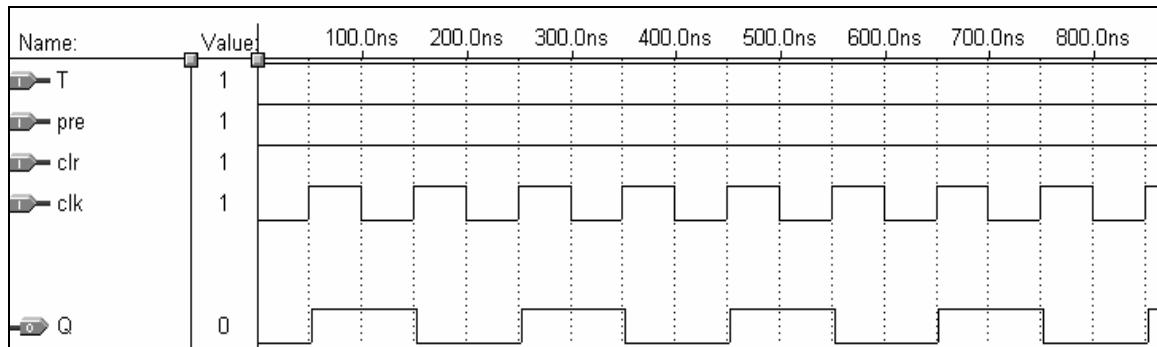


Figure S13-6

pre	clr	clk	T	Q
1	1	↑	0	NC
1	1	↑	1	Toggle
0	1	↑	0	1
1	0	↑	1	0

Table S13-1

SUPPLEMENTAL EXPERIMENT 14

Flip-Flop Applications

Solutions for Part 1:

- I) Measure worst-case t_{co} : Input the appropriate data in Table S18-1.

Slack	Required t_{co}	Actual t_{co}	From	To	From Clock
N/A	None	26 ns	QA	clk	
N/A	None				
N/A	None				

Table S14-1

r)

- 1) The frequency of the clock signal is 10 MHz.
- 2) The frequency of QC is 5 MHz. QC divides “clk” by 2.
- 3) The frequency of QB is 2.5 MHz. QB divides “clk” by 4.
- 4) The frequency of QA is 1.25 MHz. QA divides “clk” by 8.
- 5) The MOD number of the counter is 8.
- 7) Does the counter count up or down? Down

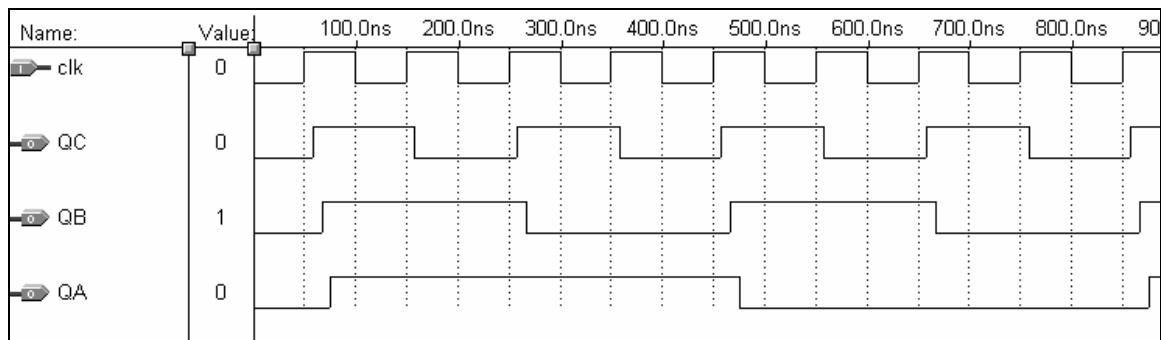


Figure S14 – 2

SUPPLEMENTAL EXPERIMENT 14

Flip-Flop Applications

clk	Time Interval	QA	QB	QC
0	0 – 100ns	1	1	1
1	100ns – 200ns	1	1	0
2	200ns – 300ns	1	0	1
3	300ns – 400ns	1	0	0
4	400ns – 500ns	0	1	1
5	500ns – 600ns	0	1	0
6	600ns – 700ns	0	0	1
7	700ns – 800ns	0	0	0

Table S14-1

clk	Time Interval	QA	QB	QC
0	0 – 100ns	1	1	1
1	100ns – 200ns	1	1	0
2	200ns – 300ns	1	0	1
3	300ns – 400ns	1	0	0
4	400ns – 500ns	0	1	1
5	500ns – 600ns	0	1	0
6	600ns – 700ns	0	0	1
7	700ns – 800ns	0	0	0

Table S24-2

SUPPLEMENTAL EXPERIMENT 14

Flip-Flop Applications

Solutions for Part 2:

v)

- 1) Examine the waveforms in Figure S14-6. Did you get any glitches? _____ Yes _____
- 2) If so, measure the time at which they occur with a Time Bar? _428 ns_____, ____826 ns_____
- 3) Counting the glitch as a 1, what counter numbers produce a glitch? _____4_____, _____
- 4) What is the count sequence of the counter? _0,1,2,3,0_(not counting glitch)_____.
- 5) The MOD number of the counter is ____4 ____.
- 7) Does the counter count up or down? ____Up_____
- 8) What is the frequency of QB? ____2.5 MHz_____

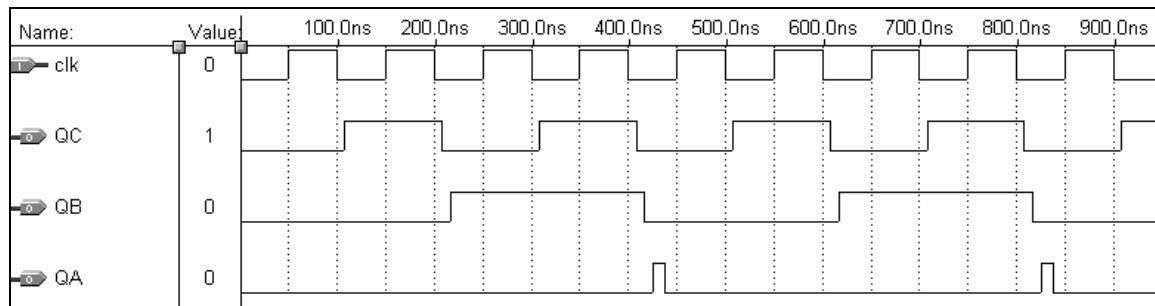


Figure S14 – 5

clk	Time Interval	QA	QB	QC	Glitch?
0	0 – 100ns	0	0	0`	No
1	100ns – 200ns	0	0	1	No
2	200ns – 300ns	0	1	0	No
3	300ns – 400ns	0	1	1	No
4	400ns – 500ns	0	0	0	Yes
5	500ns – 600ns	0	0	1	No
6	600ns – 700ns	0	1	0	No
7	700ns – 800ns	0	1	1	No

Table S14-2

SUPPLEMENTAL EXPERIMENT 14

Flip-Flop Applications

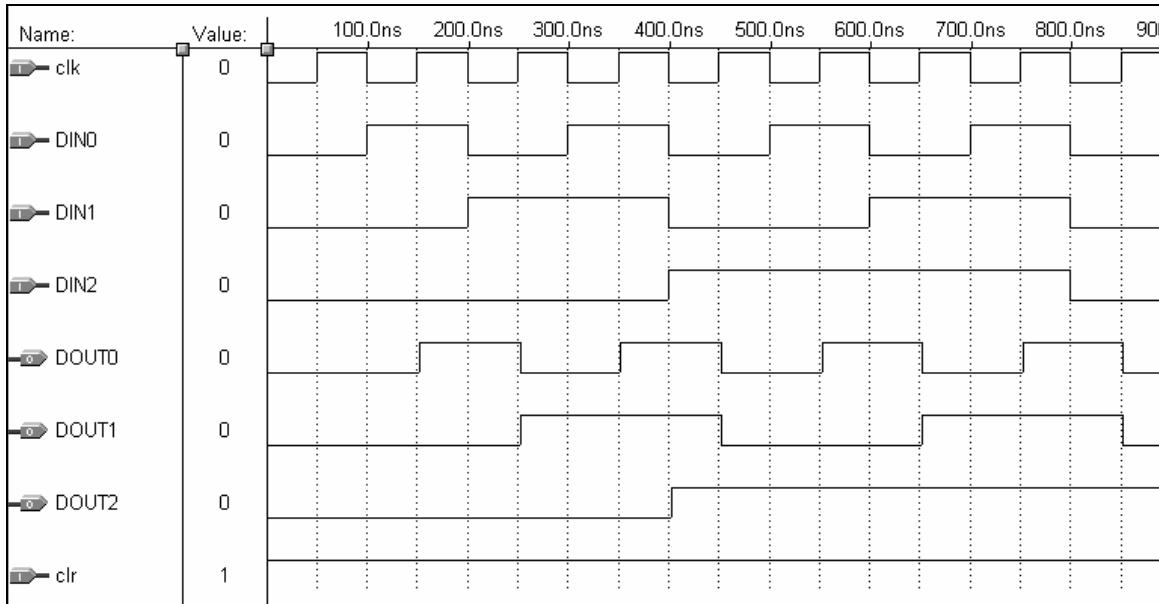


Figure S14-7

Solutions for Part 4:

p)

- 1) Write down the state of DOUTA[2..0]: 000. If this value is not 000, press “**clr1**”. Does DOUTA[2..0] clear? If not, then you most likely have a logical error. Recheck your circuit, procedure, and assignments. If you have to change your circuit or perhaps a logic option, don’t forget to re-compile the project.
- 2) Write down the state of DOUTB[2..0]: 000. If this value is not 000, press “**clr2**”. Take the same precautions as in step 1.
- 3) Set DIN[2..0] to 111 and clock the data transfer system by pressing “**pb3**” momentarily.
- 4) Write down the state of DOUTA[2..0]: 111. Write down the state of DOUTB[2..0]: 000.
- 5) Set DIN[2..0] and clock the system one more time. Write down the state of DOUTA[2..0]: 110. Write down the state of DOUTB[2..0]: 111.
- 6) In steps 3 – 6, you should have observed that DOUTA[2..0] changes one clock cycle earlier than DOUTB[2..0]. Does this agree with your simulation results?
Yes _____

SUPPLEMENTAL EXPERIMENT 14

Flip-Flop Applications

k)

- 1) DOUTA[2..0]: 000 _____.
- 2) DOUTB[2..0]: 000 _____.
- 3) DOUTA[2..0]: 111 _____. DOUTB[2..0]: 000 _____.
- 4) DOUTA[2..0]: 110 _____. DOUTB[2..0]: 111 _____.
- 5) DOUTA[2..0]: 101 _____. DOUTB[2..0]: 110 _____.
- 6) DOUTA[2..0]: 100 _____. DOUTB[2..0]: 101 _____.
- 7) _____ Yes _____

Clr1	Clr2	DIN[2..0]	Clock	DOUTA[2..0]	DOUTB[2..0]
0	1	111	x	000	000
1	0	111	x	000	000
1	1	111	↑	111	000
1	1	110	↑	110	111
1	1	101	↑	101	110
1	1	100	↑	100	101
1	1	011	↑	011	100
1	1	010	↑	010	011
1	1	001	↑	001	010
1	1	000	↑	000	001

Table S14-3

SUPPLEMENTAL EXPERIMENT 15

Implementing Flip-Flops and Flip-Flop Devices With VHDL

Solutions for Part 1:

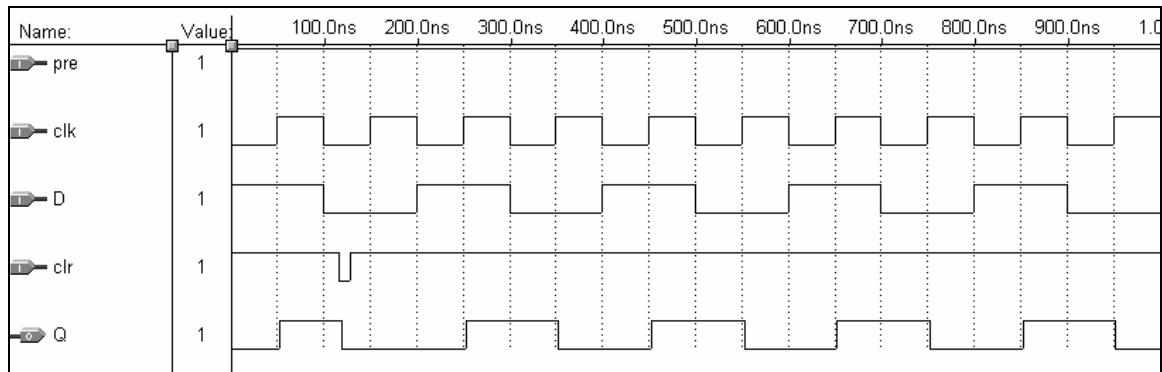


Figure S15-2

pre	clr	clk	D	Q
1	1	↑	0	0
1	1	↑	1	1
0	1	x	1	0
1	0	x	0	1

Table S15-1

Solutions for Part 2:

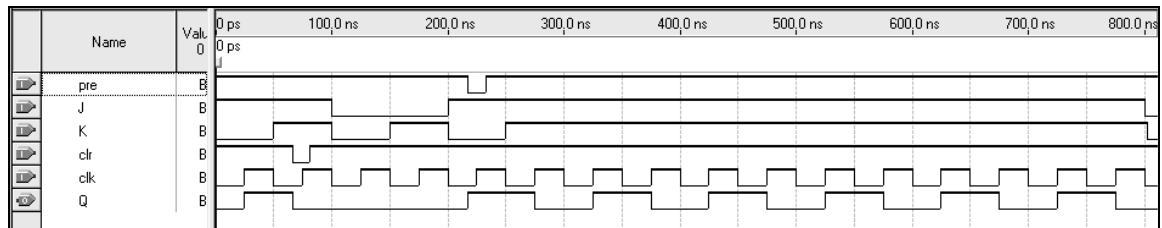


Figure S15-4

SUPPLEMENTAL EXPERIMENT 15

Implementing Flip-Flops and Flip-Flop Devices With VHDL

pre	clr	clk	J	K	Q
1	1	↑	0	0	NC
1	1	↑	0	1	0
1	1	↑	1	0	1
1	1	↑	1	1	T
0	1	↑	0	1	1
1	0	↑	1	1	0

Table S15-2

Solutions for Part 3:

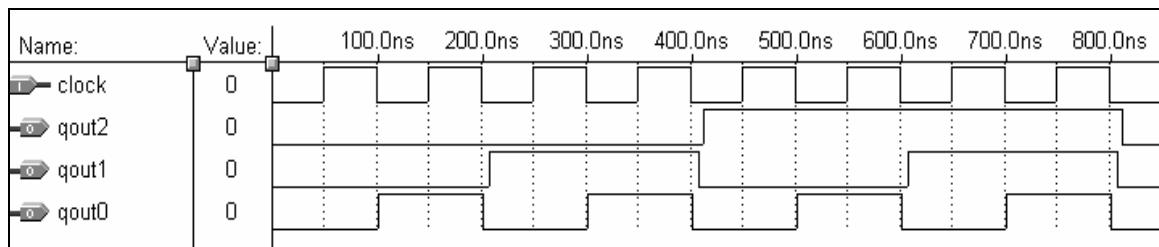


Figure S15-6

SUPPLEMENTAL EXPERIMENT 16

Binary Adders and 2's Complement System

Solutions for Part 1:

j) Record and analyze results:

- 2) $\text{Sum}0 = \underline{\hspace{2cm}} \text{A}0 \oplus \text{B}0 \underline{\hspace{2cm}}$
 3) $\text{Carry}0 = \underline{\hspace{2cm}} \text{A}0 \bullet \text{B}0 \underline{\hspace{2cm}}$

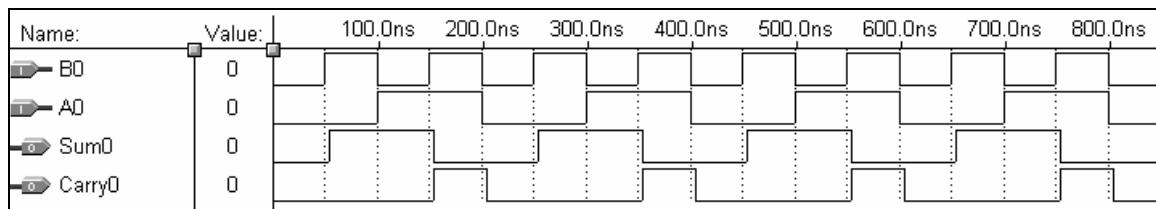


Figure S16 – 2

A0	B0	Sum0	Carry0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table S16-1

Solutions for Part 2:

j) Record and analyze results:

- 4) $\text{Sum}0 = \underline{\hspace{2cm}} \text{A}0 \oplus \text{B}0 \oplus \text{Cin} \underline{\hspace{2cm}}$
 5) $\text{Carry}0 = \underline{\hspace{2cm}} (\text{A}0 \oplus \text{B}0) + (\text{A}0 \oplus \text{Cin}) + (\text{B}0 \oplus \text{Cin}) \underline{\hspace{2cm}}$

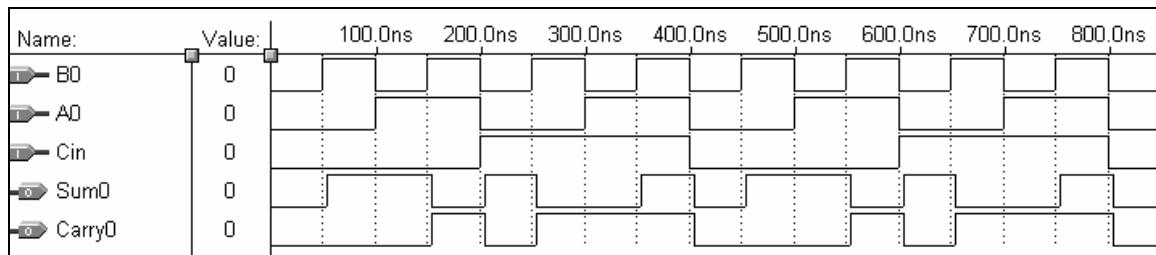


Figure S16 – 5

SUPPLEMENTAL EXPERIMENT 16

Binary Adders and 2's Complement System

Cin	A0	B0	Sum0	Carry0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table S16-2

Solutions for Part 3:

i)

- 7) 0 0011
- 8) 0 0110
- 11) 0 1101
- 14) 0 1000
- 18) 1 0110
- 21) 1 1000

Operations	S	CARRY
1-2 + 3+4+0	6	0
5+4+1+2-7+0	5	0
1+3+5+6+0	15	0
-1-2-4-6-0	2	1

Table 16-3

SUPPLEMENTAL EXPERIMENT 17

Asynchronous Counters

Solutions for Part 1:

- j) Examine the **Timing Analyzer Summary** report. What is the worst-case t_{co} ? _____

Slack	Required t_{co}	Actual t_{co}	From	To	From Clock
N/A	None	17 ns	74293:1/15	QB	clk
N/A	None				
N/A	None				

Table S17-1

o) Record and analyze results:

1. The frequency of the clock signal is 20 MHz.
2. The frequency of QD is 1.25 MHz. QD divides clk by _____.
3. The frequency of QC is 2.5 MHz. QC divides clk by _____.
4. The frequency of QB is 5 MHz. QB divides clk by _____.
5. The frequency of QA is 10 MHz. QA divides clk by _____.
6. The MOD number of the counter is 16.
10. Does the counter count up or down? UP

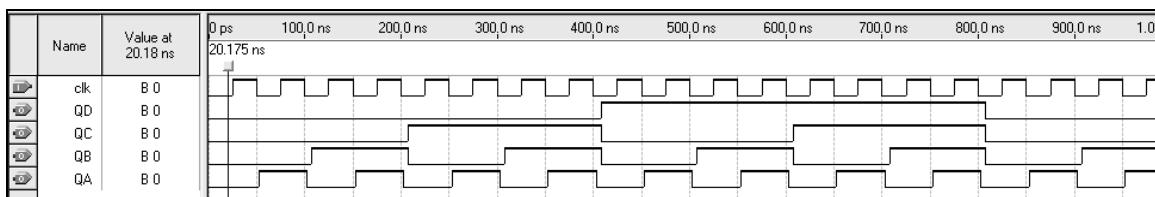


Figure S17 – 2

SUPPLEMENTAL EXPERIMENT 17

Asynchronous Counters

clk	Time Interval	QA	QB	QC	QD
0	25ns – 50ns	0	0	0	0
1	75ns – 100ns	1	0	0	0
2	125ns – 150ns	0	1	0	0
3	175ns – 200ns	1	1	0	0
4	225ns – 250ns	0	0	1	0
5	275ns – 300ns	1	0	1	0
6	325ns – 350ns	0	1	1	0
7	375ns – 400ns	1	1	1	0
8	425ns – 450ns	0	0	0	1
9	475ns – 500ns	1	0	0	1
10	525ns – 550ns	0	1	0	1
11	575ns – 600ns	1	1	0	1
12	625ns – 650ns	0	0	1	1
13	675ns – 700ns	1	0	1	1
14	725ns – 750ns	0	1	1	1
15	775ns – 800ns	1	1	1	1

Table S17-1

Solutions for Part 2:

- h) Examine the **Timing Analyzer Summary** report. What is the worst-case t_{co} ? 35 ns

Slack	Required t_{co}	Actual t_{co}	From	To	From Clock
N/A	None	35 ns	74293:20/15	QF	clk
N/A	None				
N/A	None				

Table S17-4

SUPPLEMENTAL EXPERIMENT 17

Asynchronous Counters

p) Record and analyze results:

- 1) The frequency of the clock signal is 40MHz.
- 2) The frequency of QH is 0.156250 MHz. QH divides clk by 256.
- 3) The frequency of QG is 0.312500 MHz. QG divides clk by 128.
- 4) The frequency of QF is 0.625000 MHz. QF divides clk by 64.
- 5) The frequency of QE is 1.25 MHz. QE divides clk by 32.
- 6) The frequency of QD is 2.5 MHz. QD divides clk by 16.
- 7) The frequency of QC is 5 MHz. QC divides clk by 8.
- 8) The frequency of QB is 10 MHz. QB divides clk by 4.
- 9) The frequency of QA is 20 MHz. QA divides clk by 2.
- 10) The MOD number of the counter is 256.
- 11) Does the counter count up or down? Up

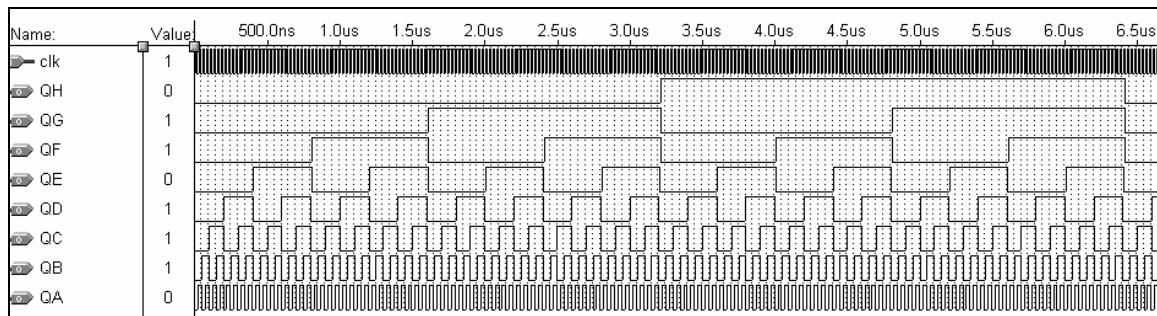


Figure S17-4

[Note: Figure S17.6 in the lab manual and data collection sheet should be changed to agree with Figure S17-6 above. The clock period should be set to 25 ns and the End Time to 7 us.]

SUPPLEMENTAL EXPERIMENT 18

Synchronous Counters

Solutions for Part 1:

h) typical:

- h)** Examine the **Timing Analyzer Summary** report. What is the worst-case t_{co} ? 8.000 ns (typical))

Slack	Required t_{co}	Actual t_{co}	From	To	From Clock
N/A	None	8.000 ns	15	QC	clk
N/A	None				
N/A	None				

Table S18-1

- i)** In a few well chosen words, compare these results with those in Part 1 of Experiment 14 (Table S14-1). The t_{co} in this experiment is extremely small compared to that of Experiment 14 Part 1 (less than 1/3).

q) Record and analyze results:

- 1) The frequency of the clock signal is 20 MHz.
- 2) The frequency of QA is 2.5 MHz. QA divides clk by 8.
- 3) The MOD number of the counter is 8.

s) Does the counter count up or down? Up

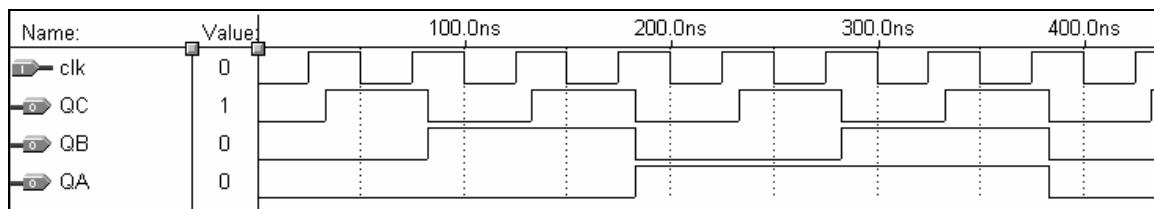


Figure S18 – 2

SUPPLEMENTAL EXPERIMENT 18

Synchronous Counters

clk	Time Interval	QA	QB	QC
0	0ns – 25ns	0	0	0
1	25ns – 75ns	0	0	1
2	75ns – 125ns	0	1	0
3	175ns – 200ns	0	1	1
4	225ns – 250ns	1	0	0
5	275ns – 300ns	1	0	1
6	325ns – 350ns	1	1	0
7	375ns – 400ns	1	1	1
8	425ns – 450ns	0	0	0

Table S18-2

Solutions for Part 2:

o)

- 2) The frequency of the clock signal is 20 MHz.
- 3) The frequency of QD is 1.25 MHz. QD divides clk by 16.
- 4) The MOD number of the counter is 16.

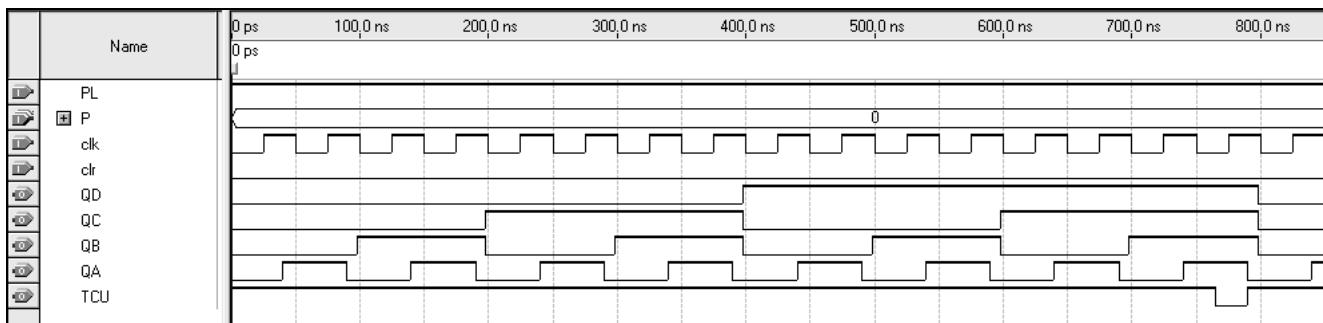


Figure S18-4

p)

- 4) Does the counter count up or down? Up
- 5) What must you do to make the counter count in the opposite direction? Change 74193 UP connection to Vcc and DN connection to "clk".

SUPPLEMENTAL EXPERIMENT 18

Synchronous Counters

clk	Time Interval	QA	QB	QC	QD
0	25ns – 50ns	0	0	0	0
1	75ns – 100ns	1	0	0	0
2	125ns – 150ns	0	1	0	0
3	175ns – 200ns	1	1	0	0
4	225ns – 250ns	0	0	1	0
5	275ns – 300ns	1	0	1	0
6	325ns – 350ns	0	1	1	0
7	375ns – 400ns	1	1	1	0
8	425ns – 450ns	0	0	0	1
9	475ns – 500ns	1	0	0	1
10	525ns – 550ns	0	1	0	1
11	575ns – 600ns	1	1	0	1
12	625ns – 650ns	0	0	1	1
13	675ns – 700ns	1	0	1	1
14	725ns – 750ns	0	1	1	1
15	775ns – 800ns	1	1	1	1

Table S18-2

Solutions for Part 3:

[Note: set the clock period to 100 ns and the end time to 21 us.]

q) ____ 200 _____
s) ____ 20 us _____

SUPPLEMENTAL EXPERIMENT 18

Synchronous Counters

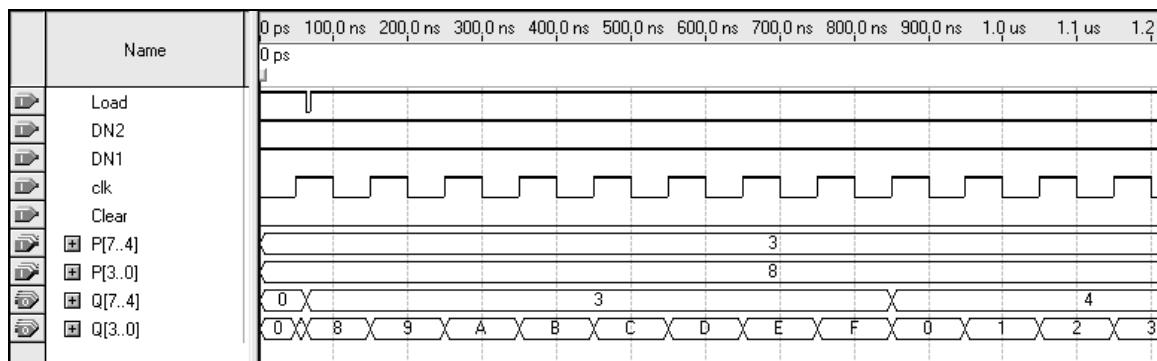


Figure S18-6

SUPPLEMENTAL EXPERIMENT 19

BCD Counters

Data collection for Part 1:

o) Record and analyze results:

- 3) What is the count sequence for this counter? 0,1,2,3,4,5,6,7,8,9,0
- 4) Does the counter count up or down? _____Up_____

q) Test the program:

- 1) Toggle the Enable switch to HIGH. What value is displayed on the LEDs? 00000

Pulse the clock pushbutton through the counter's counting sequence.

- 3) What is the counter's sequence? 0,1,2,3,4,5,6,7,8,9,0

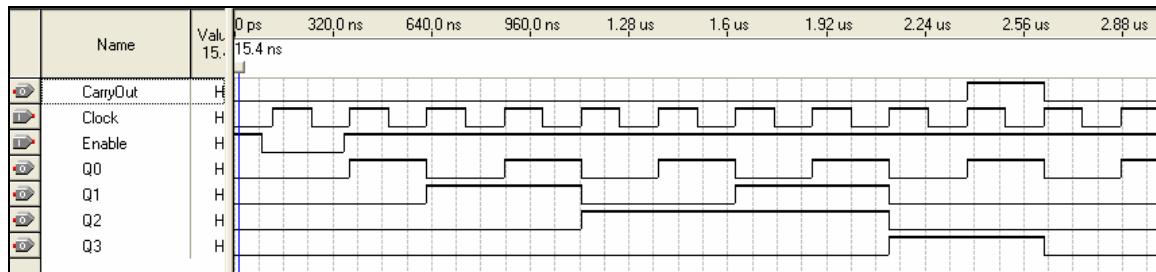


Figure S19-2

clk	Time Interval	Carry Out	Q3	Q2	Q1	Q0
0	375ns – 425ns	0	0	0	0	0
1	425ns – 675ns	0	0	0	0	1
2	675ns – 925ns	0	0	0	1	0
3	925ns – 1.175us	0	0	0	1	1
4	1.175us – 1.425us	0	0	1	0	0
5	1.425us – 1.675us	0	0	1	0	1
6	1.675us – 1.925us	0	0	1	1	0
7	1.925us – 2.175us	0	0	1	1	1
8	2.175us – 2.425us	0	1	0	0	0
9	2.425us – 2.675us	1	1	0	0	1
10	2.675us – 2.925us	0	0	0	0	0

Table S19-1

SUPPLEMENTAL EXPERIMENT 19

BCD Counters

Data collection for Part 2:

- f)**
- 1) 00 .
- 2) 99 .

SUPPLEMENTAL EXPERIMENT 20

Shift Register Counters

Solutions for Part 1:

o) Record and analyze results:

- 1) What is the purpose of presetting the first flip-flop while keeping the other flip-flops cleared?
At least one FF must be preset. The first FF is convenient.

- 4) The MOD number of the counter is 4.

- 5) What could you do to the circuit to change the count sequence to 1001, 1100, 0110, 0011, 1001, ...?
One way would be to preset the first and last FF before starting the count

- 6) How long does the counter take in your simulation to complete one complete count cycle?
400ns

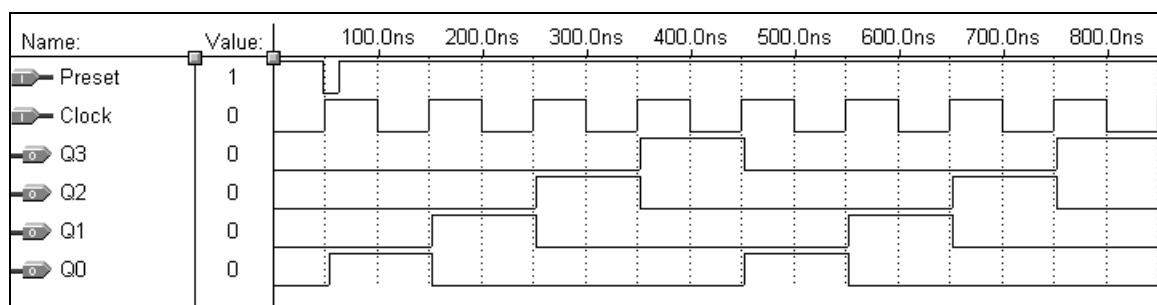


Figure S20 – 2

clk	Time Interval	Q0	Q1	Q2	Q3
0	0ns – 50ns	0	0	0	0
1	50ns – 100ns	1	0	0	0
2	150ns – 200ns	0	1	0	0
3	250ns – 300ns	0	0	1	0
4	350ns – 400ns	0	0	0	1
5	450ns – 500ns	1	0	0	0
6	550ns – 600ns	0	1	0	0

Table S20-1

SUPPLEMENTAL EXPERIMENT 20

Shift Register Counters

Solutions for Part 2:

o)

- 3) What is the count sequence for this counter?

_____ 0,1,3,7,15,14,12,8,0,... _____

- 4) The MOD number of the counter is ____ 8 ____.

- 5) Name two major differences between the ring and Johnson counters:

The ring counter feeds the Q output of the last FF back to the D input of the first FF while the Johnson counter feeds the complement of the Q output of the last FF back to the D input of the first FF. The MOD number of the ring is the number of FFs while the Johnson's MOD number is twice the number of FFs (among others).

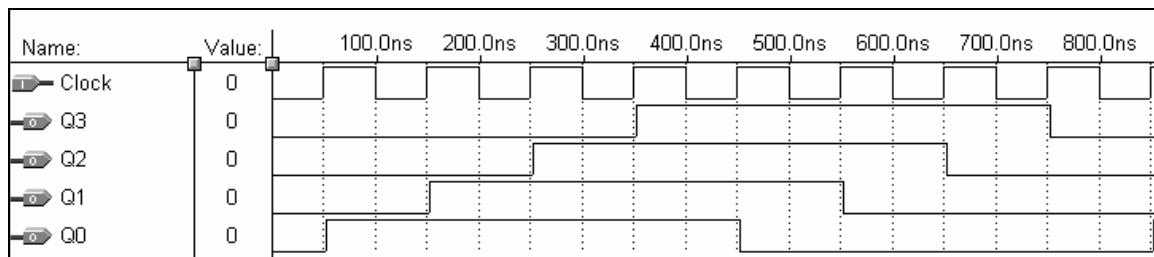


Figure S20-4

SUPPLEMENTAL EXPERIMENT 20

Shift Register Counters

clk	Time Interval	Q0	Q1	Q2	Q3
0	0 – 50ns	0	0	0	0
1	100ns – 150ns	1	0	0	0
2	200ns – 250ns	1	1	0	0
3	300ns – 350ns	1	1	1	0
4	400ns – 450ns	1	1	1	1
5	500ns – 550ns	0	1	1	1
6	600ns – 650ns	0	0	1	1
7	700ns – 750ns	0	0	0	1
8	800ns – 850ns	0	0	0	0

Table S20-2

SUPPLEMENTAL EXPERIMENT 21

IC Registers

Solutions for Part 1:

o) Record and analyze results:

- 4) What values of S0 and S1 are necessary for parallel loading? S0 = 1 S1 = 1
- 5) What values of S0 and S1 are necessary for shifting? S0 = 1 S1 = 0 (shift right)
- 6) Is parallel loading in the 74194A IC synchronous or asynchronous? Synchronous

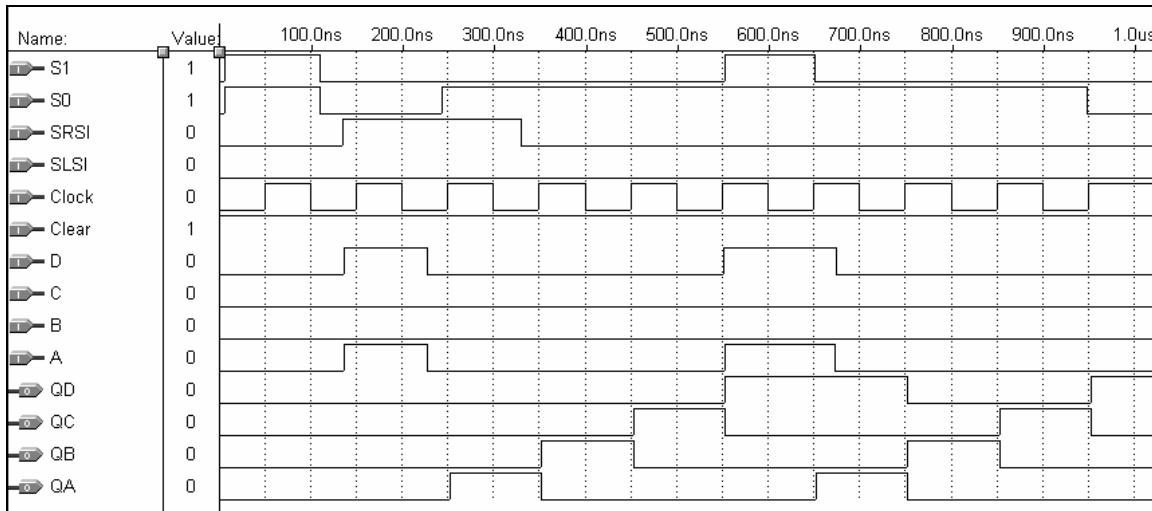


Figure S21 – 2

S0	S1	SRSI	SLSI	A	B	C	D	clk	Time Interval	QA	QB	QC	QD
1	1	0	0	0	0	0	0	↑	0ns – 100ns	0	0	0	0
0	0	1	0	1	0	0	1	↑	150ns – 200ns	0	0	0	0
1	0	1	0	0	0	0	0	↑	250ns – 300ns	1	0	0	0
1	0	0	0	0	0	0	0	↑	350ns – 400ns	0	1	0	0
1	0	0	0	0	0	0	0	↑	450ns – 500ns	0	0	1	0
1	0	0	0	1	0	0	1	↑	550ns – 600ns	0	0	0	1
1	1	0	0	1	0	0	1	↑	650ns – 700ns	1	0	0	1
1	0	0	0	0	0	0	0	↑	750ns – 800ns	0	1	0	0
1	0	0	0	0	0	0	0	↑	850ns – 900ns	0	0	1	0
1	0	0	0	0	0	0	0	↑	950ns – 1us	0	0	0	1

Table S21-1

SUPPLEMENTAL EXPERIMENT 21

IC Registers

Solutions for Part 2:

- o) Record and analyze results:
- 3) What is the count sequence for this counter?
1000,0100,0010,0001,1000,...
- 4) The MOD number of the counter is 4.
- 5) Name two major differences between the ring and Johnson counters:
The ring counter feeds the Q output of the last FF back to the D input of the first FF while the Johnson counter feeds the complement of the Q output of the last FF back to the D input of the first FF. The MOD number of the ring is the number of FFs while the Johnson's MOD number is twice the number of FFs (among others).

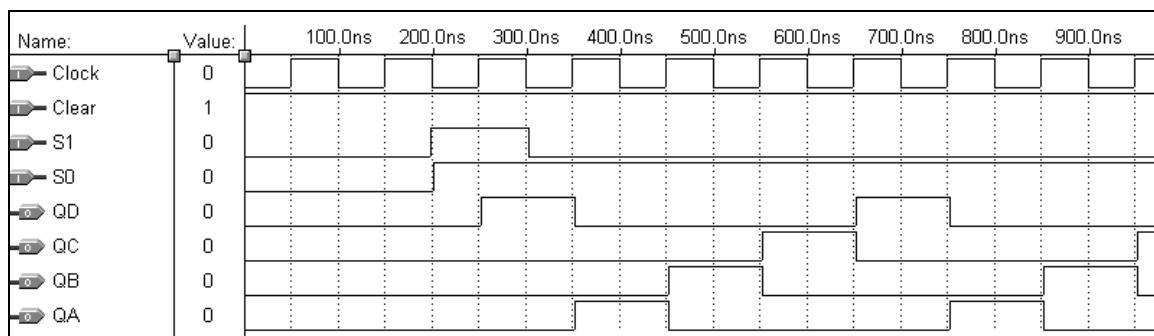


Figure S21-4

clk	Time Interval	Q0	Q1	Q2	Q3
0	0 – 50ns	0	0	0	0
1	100ns – 150ns	0	0	0	0
2	200ns – 250ns	0	0	0	0
3	300ns – 350ns	1	0	0	0
4	400ns – 450ns	0	1	0	0
5	500ns – 550ns	0	0	1	0
6	600ns – 650ns	0	0	0	1
7	700ns – 750ns	1	0	0	0
8	800ns – 850ns	0	1	0	0

Table S21-2

SUPPLEMENTAL EXPERIMENT 21

IC Registers

Solutions for Part 3:

- q) Record and analyze results:
- 2) Why is it necessary to synchronize the load and shift operations of the 74194A? Unless the load and shift are synchronized, shifting could occur before parallel data is loaded.
 - 3) Explain how the flip-flops accomplish this synchronization in the circuit of 21-5. The first flip-flop transfers a 1 to the D input of the second flip-flop. The 1 is then transferred synchronously from the second flip-flop to PE of the 74194 while a 0 is applied to SE. This 0 clears the first flip-flop sending a 0 to PE and 1 to SE of the 74194 on the next clock pulse. This action guarantees that the load and shift are synchronized and cannot overlap.
 - 4) How would you modify the circuit of 21-5 to change it into an eight-bit converter? Cascade another 74194 to the current 74194 by connecting QD of the current 74194 to the SRSI input of the 74194 being added. All control inputs (S0, S1, CLK, CLRN) of the current 74194 are connected to the corresponding control inputs of the 74194 being added.

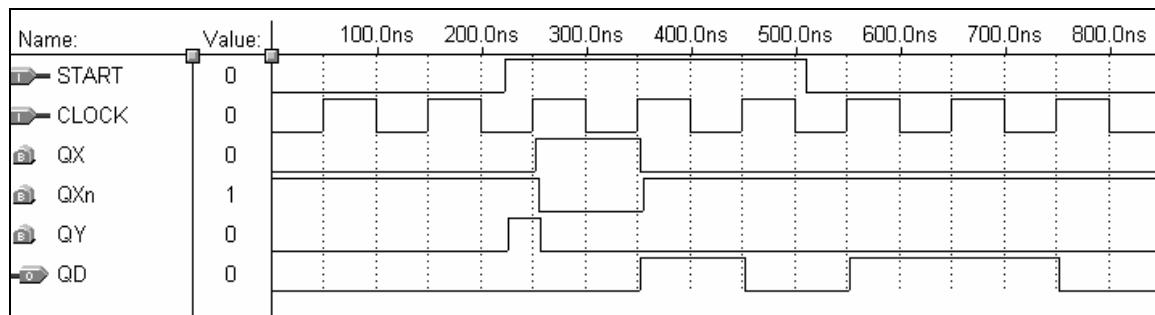


Figure S21- 6

SUPPLEMENTAL EXPERIMENT 22

One-Shots, Counters, and Registers With VHDL

Solutions for Part 1:

u) Record and analyze results:

- 1) Record the output waveform q on Figure S22-2 or print out the Simulation Waveforms report from QUARTUS® II and attach it to the data collection sheet at the end of this experiment.
- 2) With the clock period used in the simulation, what is the maximum amount of delay possible? 1.5μs
- 3) How would you modify this one-shot to get a maximum delay of 255 clock periods? Change the range of delay to be 0 TO 255 in the VHDL program

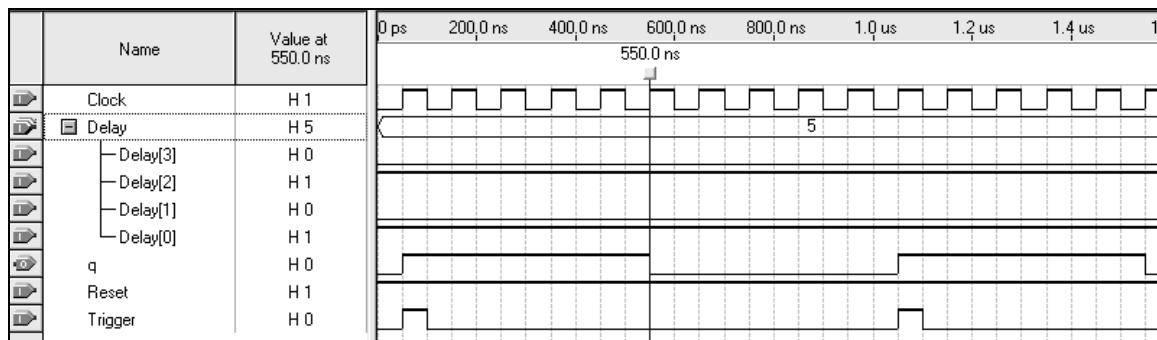


Figure S22 – 2

Solutions for Part 2:

u) Record and analyze results:

- 2) What is the count sequence for this counter when din[3..0]=9?
9,10,11,12,13,14,15,0,...
- 3) When the counter reaches its maximum count, to what value does it reset?
0

SUPPLEMENTAL EXPERIMENT 22

One-Shots, Counters, and Registers With VHDL

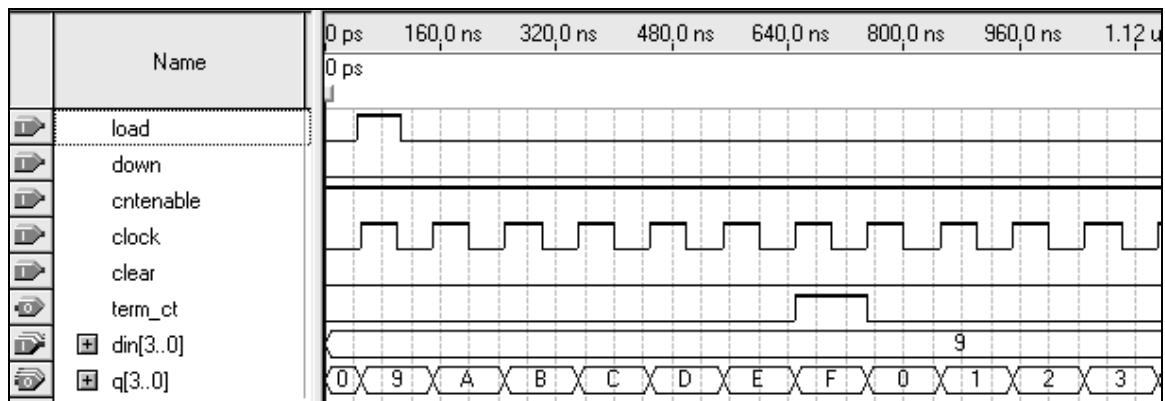


Figure S22-5

Solutions for Part 3:

v) Record and analyze results:

- 2) What is the procedure for loading a 6 in parallel and then shifting the number out to the right replacing bits to the left with 0? Place a 6 on din input, set mode to 3, place 0 on ser_in, then set mode to 1.
- 3) In step u, what counter can you recall has the same pattern of output? Johnson counter

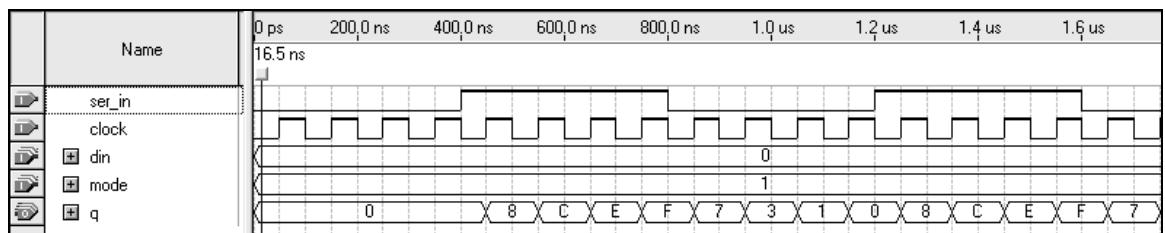


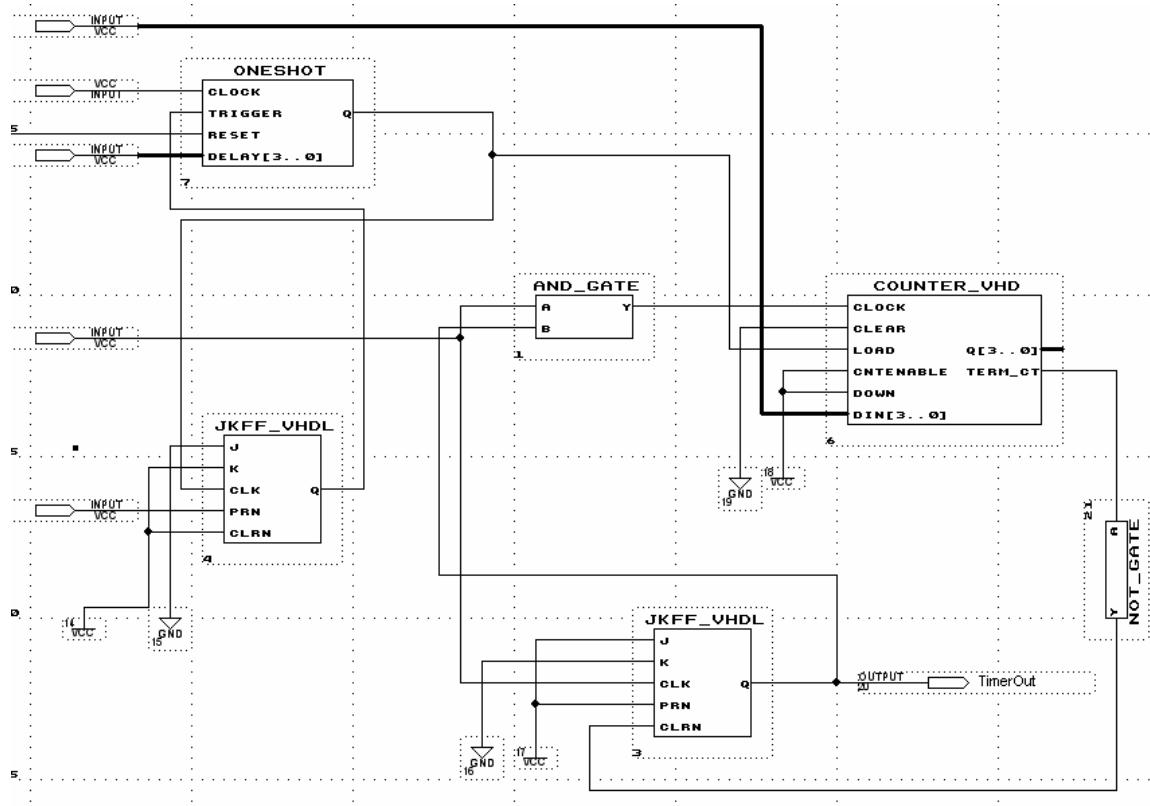
Figure S22-8

SUPPLEMENTAL EXPERIMENT 22

One-Shots, Counters, and Registers With VHDL

Solutions for Part 4:

- a) Draw your diagram here:
One of several solutions:



SUPPLEMENTAL EXPERIMENT 23

IC Decoders and Encoders

Solutions for Part 1:

p) Record and analyze results:

- 3) What values of A, B, and C are required to enable O6n?
 $A = \underline{0}$ $B = \underline{1}$ $C = \underline{1}$
- 4) If G1 is LOW, what are the output values for O0n – O7n? All HIGH
- 5) Is it possible for two outputs to be enabled simultaneously? No

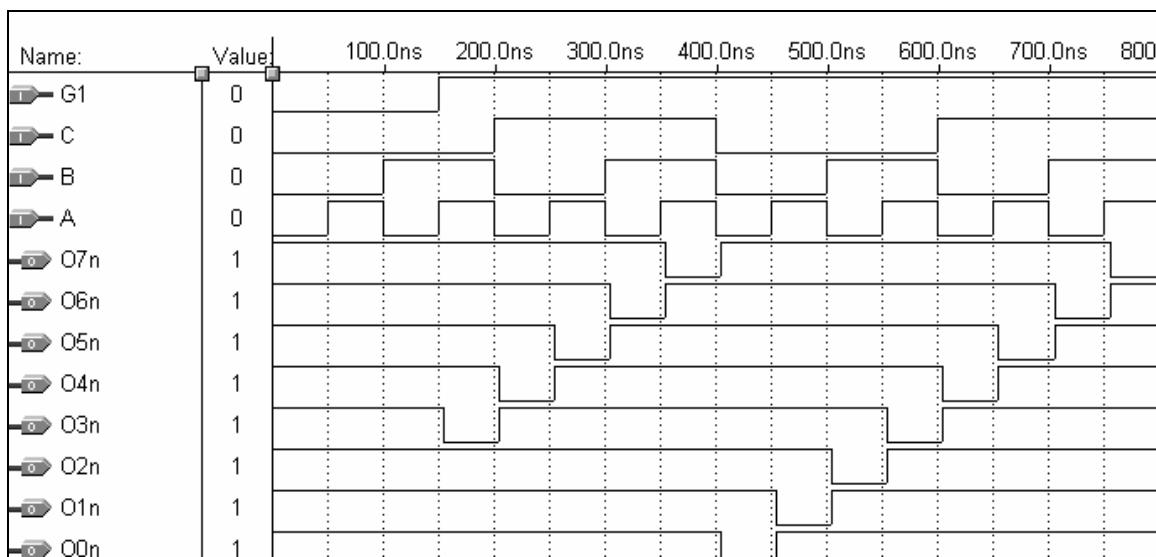


Figure S23 – 2

SUPPLEMENTAL EXPERIMENT 23

IC Decoders and Encoders

G1	G2An	G2Bn	A	B	C	Time Interval	O0n	O1n	O2n	O3n	O4n	O5n	O6n	O7n
0	0	0	0	0	0	0ns – 50ns	1	1	1	1	1	1	1	1
0	0	0	1	0	0	50ns – 100ns	1	1	1	1	1	1	1	1
0	0	0	0	1	0	100ns – 150ns	1	1	1	1	1	1	1	1
1	0	0	1	1	0	150ns – 200ns	1	1	1	0	1	1	1	1
1	0	0	0	0	1	200ns – 250ns	1	1	1	1	0	1	1	1
1	0	0	1	0	1	250ns – 300ns	1	1	1	1	1	0	1	1
1	0	0	0	1	1	300ns – 350ns	1	1	1	1	1	1	0	1
1	0	0	1	1	1	350ns – 400ns	1	1	1	1	1	1	1	0
1	0	0	0	0	0	400ns – 450ns	0	1	1	1	1	1	1	1
1	0	0	1	0	0	450ns – 500ns	1	0	1	1	1	1	1	1
1	0	0	0	1	0	500ns – 550ns	1	1	0	1	1	1	1	1
1	0	0	1	1	0	550ns – 600ns	1	1	1	0	1	1	1	1
1	0	0	0	0	1	600ns – 650ns	1	1	1	1	0	1	1	1
1	0	0	1	0	1	650ns – 700ns	1	1	1	1	1	0	1	1
1	0	0	0	1	1	700ns – 750ns	1	1	1	1	1	1	0	1
1	0	0	1	1	1	750ns – 800ns	1	1	1	1	1	1	1	0

Table S23-1

Solutions for Part 2:

- o) Record and analyze results:
- 2) What are the output values if DCBA = 1010? All outputs are HIGH
- 3) Can you suggest a way to convert the 7442 to an octal decoder with enable? Use Input D as an active LOW enable. If D is HIGH all outputs O0-O7 will be HIGH.

SUPPLEMENTAL EXPERIMENT 23

IC Decoders and Encoders

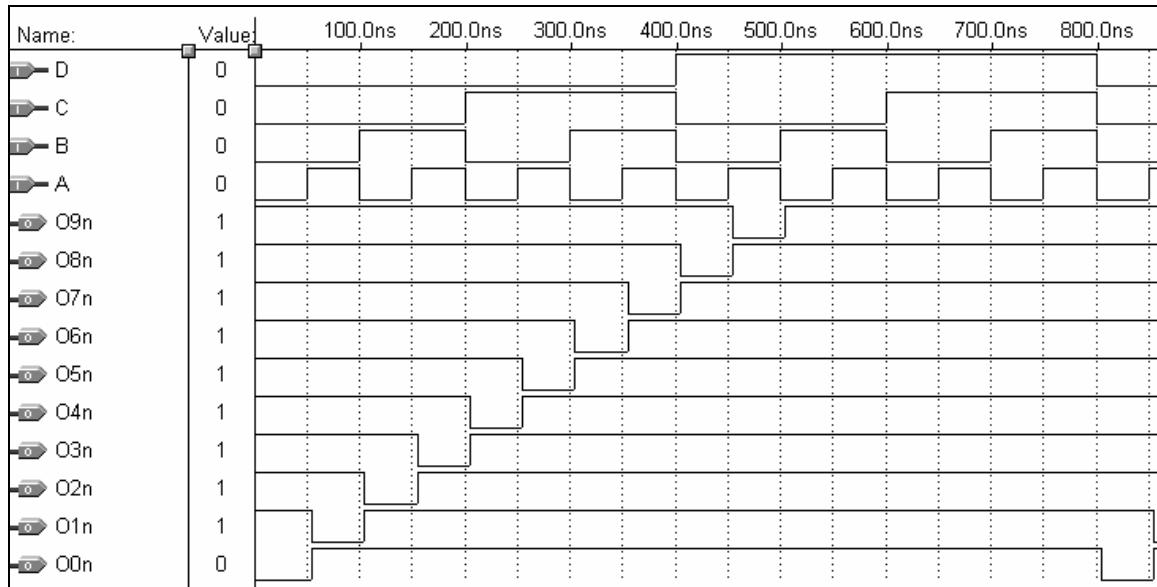


Figure S23-4

Solutions for Part 3:

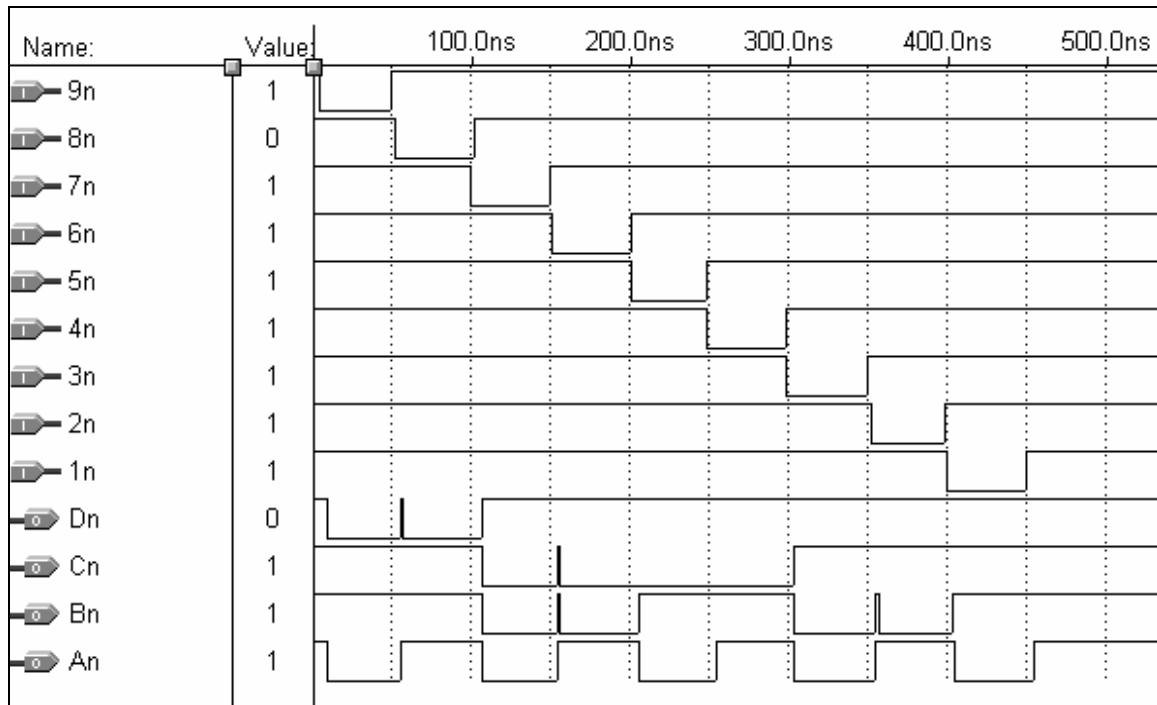


Figure S23- 6

SUPPLEMENTAL EXPERIMENT 24

IC Multiplexers and Demultiplexers

Solutions for Part 1:

- o) Record and analyze results:
- 4) What values of A, B, and C are required to select from D3? A = 1 B = 1 C = 0.
 - 5) If GN is HIGH, what are the output values for Y and WN?
Y = 0 and WN = 1
 - 6) Is it possible for two inputs to be selected simultaneously?
No

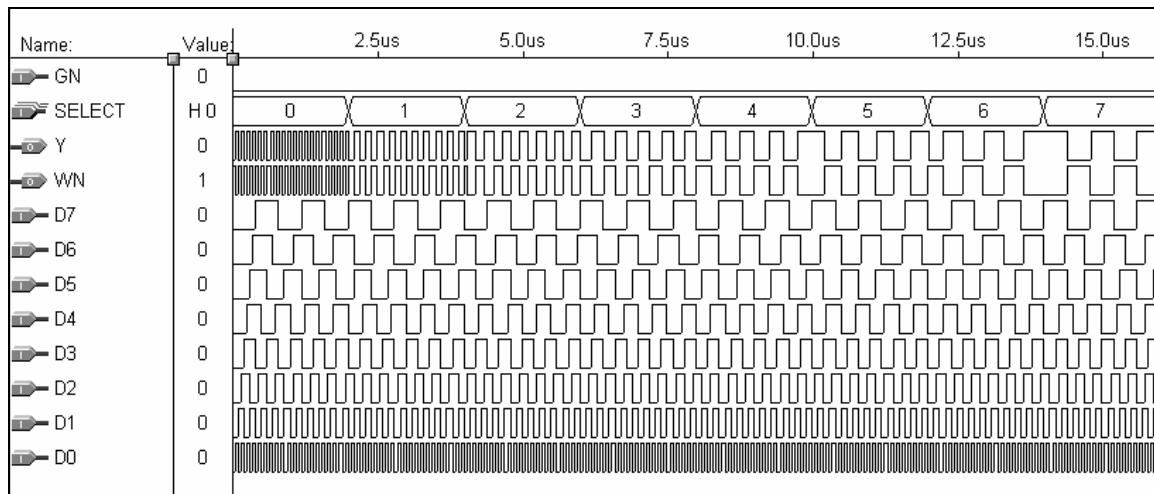


Figure S24 – 2

SELECT A B C			Time Interval	Y
0	0	0	0ns – 2ns	D0
1	0	0	2us – 4us	D1
0	1	0	4us – 6us	D2
1	1	0	6us – 8us	D3
0	0	1	8us – 10us	D4
1	0	1	10us – 12us	D5
0	1	1	12us – 14us	D6
1	1	1	14us – 16us	D7

Table S24-1

SUPPLEMENTAL EXPERIMENT 24

IC Multiplexers and Demultiplexers

Solutions for Part 2:

v) Record and analyze results:

- 4) Suppose separate counters were used to generate the SELECT for the multiplexer and the demultiplexer. Explain what would happen if the counters did not have the same count sequence:
The selected input of the multiplexer would be routed to the wrong demultiplexer output at least some of the time.
- 5) Why is G1 referred to as the 74138's inverting input? The input at G1 is passed to the selected output through the output's inverter. (G2A and G2B are inverted at both the input and output so the input is apparently not inverted).

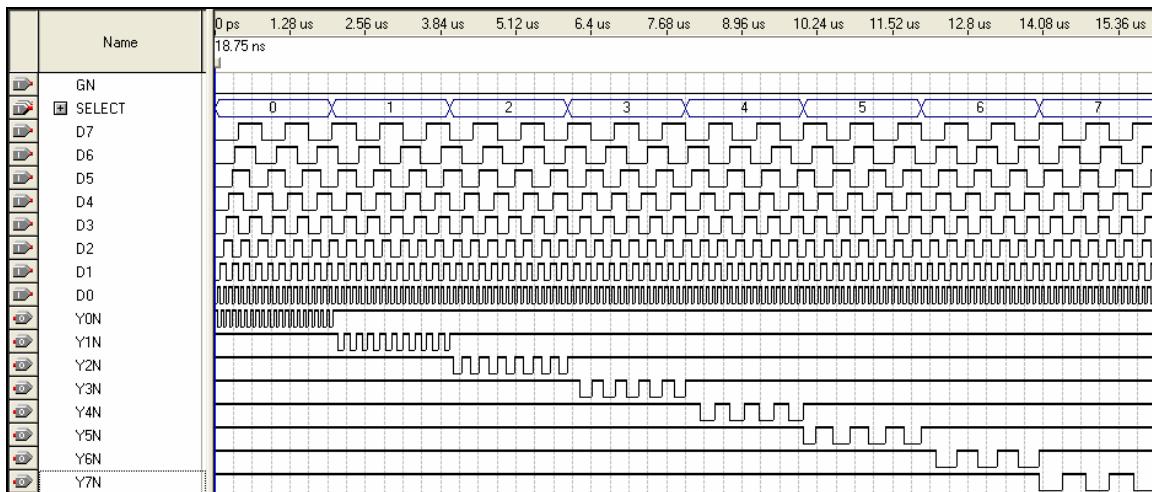


Figure S24-4

SELECT A B C			Time Interval	Y0N	Y1N	Y2N	Y3N	Y4N	Y5N	Y6N	Y7N
0	0	0	0ns – 2ns	D0	1	1	1	1	1	1	1
1	0	0	2us – 4us	1	D1	1	1	1	1	1	1
0	1	0	4us – 6us	1	1	D2	1	1	1	1	1
1	1	0	6us – 8us	1	1	1	D3	1	1	1	1
0	0	1	8us – 10us	1	1	1	1	D4	1	1	1
1	0	1	10us - 12us	1	1	1	1	1	D5	1	1
0	1	1	12us– 14us	1	1	1	1	1	1	D6	1
1	1	1	14us – 16us	1	1	1	1	1	1	1	D7

Table 24-2

SUPPLEMENTAL EXPERIMENT 25

VHDL State Machines

Solutions for Part 1:

Current State /Condition	Idle	Fill	Agitate	Spin
Start	Fill			
Full 1		Agitate		
Time Out 1 (1 min)			Spin	
Time Out 2 (1 min)				Fill
Full 2		Agitate		
Time Out 3 (1 min)			Spin	
Dry				Idle

Table S25-1

Solutions for Part 2:

Current State	Codes on 7-Segment Displays
1	00
2	04
3	02
4	01
5	14
6	12
7	11

Table S25-3

Demonstrated To
Instructor/FA _____ Date _____

Part 1 [] Part 3 []