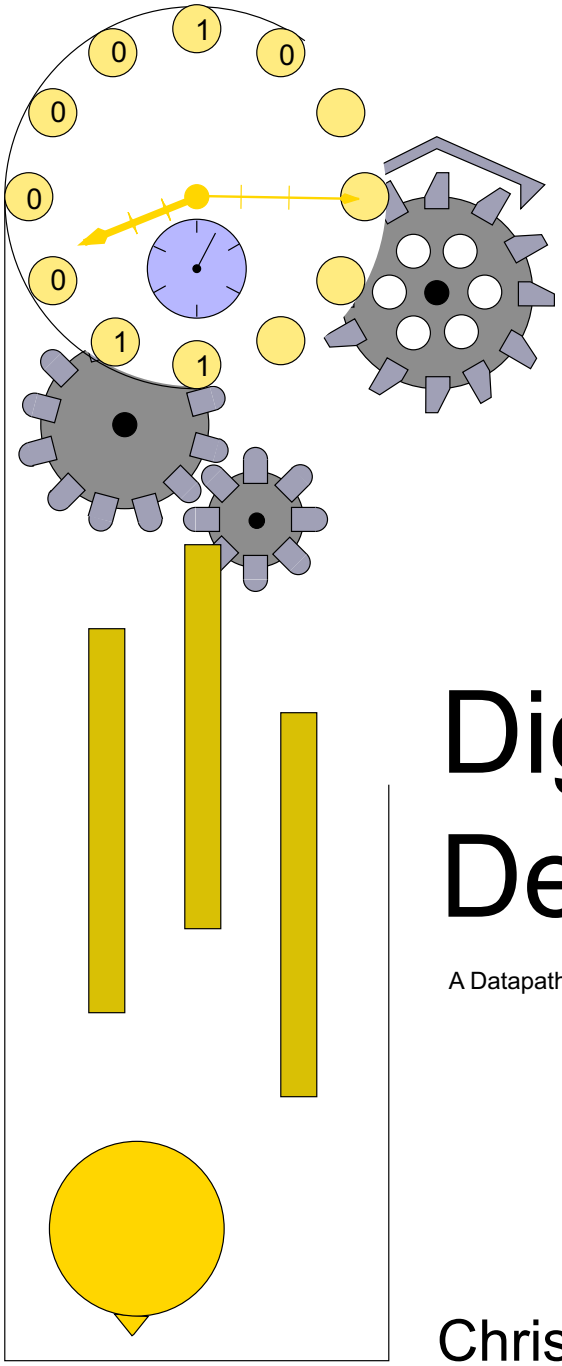


Digital Design

A Datapath and Control Approach

Chris Coulston

November 20, 2024

This document was prepared with LaTeX.

Contents

Chapter 1

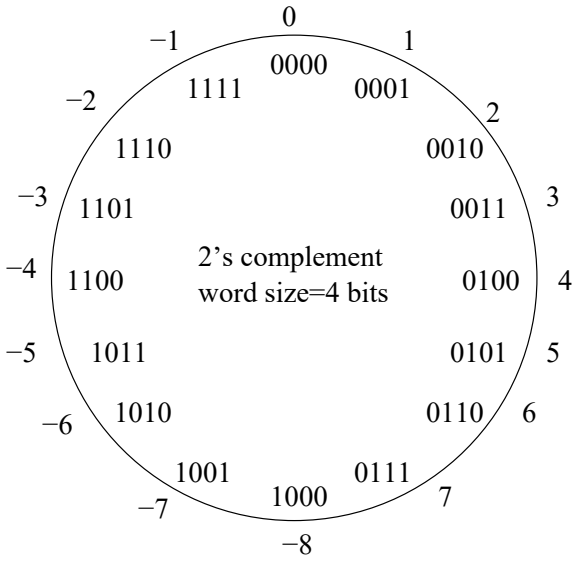
Numbering Systems

1.1 Helpfull Stuff

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4		4
5	0101	5
6		6
7		7
8	1000	8
9		9
10	1010	A
11		B
12	1100	C
13	1101	D
14		E
15	1111	F

i	0	1	2	3	4	5	6	7	8	9
2^i	1	2	4	8	16	32	64	128	256	512

$1110101011_2 =$
$1 * 2^9 + 1 * 2^8 + 1 * 2^7 + 0 * 2^6 + 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 =$
$2^8(0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0) + 2^4(1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0) + 2^0 * (1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0) =$
$2^8(0011_2) + 2^4(1010_2) + 2^0(1011_2) =$
$2^{4*2}(0011_2) + 2^{4*1}(1010_2) + 2^{4*0}(1011_2) =$
$16^2(0011_2) + 16^1(1010_2) + 16^0 * (1011_2) =$
$16^2(3) + 16^1(A) + 16^0 * (B) =$
$3AB_{16}$



Chapter 2

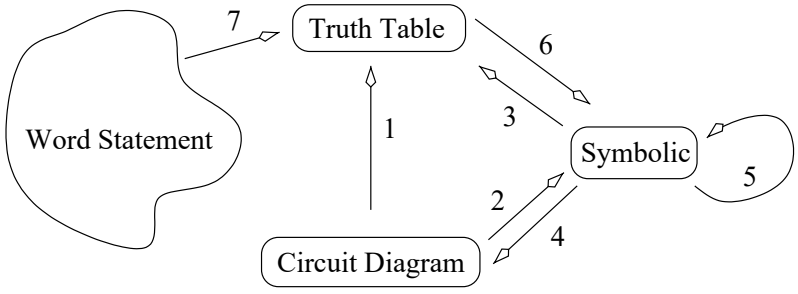
Representations of Logical Functions

2.1 Helpfull Stuff

A	B	A*B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

A	A'
0	1
1	0



	Regular Algebra	Boolean Algebra
Performed First	Parenthesis	Parenthesis
	Exponents	Not
	multiplication/division	And
Performed Last	addition/subtraction	Or

Axiom	Primary	Dual
1.	x+0=x	x*1=x
2.	x+1=1	x*0=0
3.	x+x=x	x*x=x
4.	x"=x	
5.	x+x'=1	x*x'=0
6.	x+y=y+x	x*y=y*x
7.	x+(y+z)=(x+y)+z	x*(y*z)=(x*y)*z
8.	x*(y+z)=x*y+x*z	x+(y*z)=(x+y)*(x+z)
9.	(x+y)'=x'*y'	(x*y)'=x'+y'

2.2 Definitions

Define each of the following. Some of the definitions should use terms you've defined.

Minterm

Maxterm

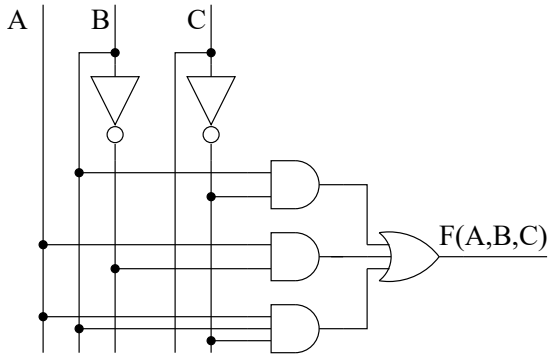
Minterm Trick

Expansion Trick

2.3 Problems

Solve the following problems in the space provided.

1. Given the circuit diagram below, produce the corresponding truth table.



A	B	C	F(A,B,C)
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

2. Given the symbolic expression below, produce the corresponding circuit diagram.

   F(A,B,C)=AB'+A(B'+C)

3. Given the symbolic expression below, produce the corresponding circuit diagram.

   F(A,B,C,D)=A(BC+A(C'+D)')' + B'CD'

4. Given the symbolic expression below, produce the corresponding truth table.

   F(A,B,C) = AB'+A(B'+C)

A	B	C			F(A,B,C)
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

5. Given the symbolic expression below, produce the corresponding truth table.

   F(A,B,C,D)=A(BC+A(C'+D)')' + B'CD'

A	B	C	D						F(A,B,C,D)
0	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	0	0	0						
1	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						

6. Given the truth table below, produce the corresponding symbolic expression.

A	B	C	F(A,B,C)	minterm	maxterm
0	0	0	0		
0	0	1	1		
0	1	0	1		
0	1	1	1		
1	0	0	1		
1	0	1	0		
1	1	0	0		
1	1	1	1		

7. Given the word state below, produce the corresponding truth table. Design a circuit with two 2-bit inputs called $A = a_1a_0$ and $B = b_1b_0$ The single bit output F should equal 1 when A+B > 6, otherwise F should equal 0.

a_1	a_0	b_1	b_0	A	B	$F(a_1, a_0, b_1, b_0)$
0	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	0	0	0			
1	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			

Chapter 3

Minimization of Logical Functions

3.1 Definitions

Simplification Trick

3.2 Problems

Simplify Utilize the simplification trick.

A	B	C	F	G	H
0	0	0	1	0	1
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	0	0	0
1	1	1	0	0	0

F(A,B,C)=
G(A,B,C)=
H(A,B,C)=

Kmap 3 variable Solve each kmap.

A	B	C	F	G	H	I	J	K	L	M
0	0	0	0	0	0	1	1	1	1	0
0	0	1	0	0	1	1	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	1	1	1	1	0	0	0	1	0
1	0	0	1	0	0	0	1	0	1	0
1	0	1	1	0	1	1	1	1	1	0
1	1	0	0	0	0	0	0	1	0	0
1	1	1	0	1	1	1	0	1	1	0

A \ BC	00	01	11	10
0				
1				

F

A \ BC	00	01	11	10
0				
1				

G

A \ BC	00	01	11	10
0				
1				

H

A \ BC	00	01	11	10
0				
1				

I

A \ BC	00	01	11	10
0				
1				

J

A \ BC	00	01	11	10
0				
1				

K

A \ BC	00	01	11	10
0				
1				

L

A \ BC	00	01	11	10
0				
1				

M

Minimize Determine the SOP $_{\min}$ for the following functions.

F(A,B,C)=A'BC' + A'BC + AB'C' + AB'C

G(A,B,C)=A'B' + AB'C' + AC

H(A,B,C)=B+AB'C + B'C'

I(A,B,C)=A(B+C')+A'B'C'

A \ BC	00	01	11	10
0				
1				

F

A \ BC	00	01	11	10
0				
1				

G

A \ BC	00	01	11	10
0				
1				

H

A \ BC	00	01	11	10
0				
1				

I

Minimize Determine the SOP $_{\min}$ realization

$F(A,B,C,D) = \sum m(0,1,4,5,8,9)$

$G(A,B,C,D) = \sum m(0,5,7,10,11,14,15)$

$H(A,B,C,D) = \sum m(0,2,3,5,6,7,8,10,11,14,15)$

$I(A,B,C,D) = \sum m(1,4,6,9,11,12,14,15)$

AB\CD	00	01	11	10
00				
01				
11				
10				

F

AB\CD	00	01	11	10
00				
01				
11				
10				

G

AB\CD	00	01	11	10
00				
01				
11				
10				

H

AB\CD	00	01	11	10
00				
01				
11				
10				

I

Minimize Determine the SOP $_{\min}$ for the following functions.

$F(A,B,C,D,E) = \sum m(0,1,2,5,7,8,10,15,16,18,23,24,26,28,29,31)$
$G(A,B,C,D,E) = \sum m(0,2,4,6,7,8,9,15,16,18,20,21,22,24,25,29)$

BC\DE	00	01	11	10
00				
01				
11				
10				

A=0

BC\DE	00	01	11	10
00				
01				
11				
10				

A=1

F(A,B,C,D,E)=

BC\DE	00	01	11	10
00				
01				
11				
10				

A=0

BC\DE	00	01	11	10
00				
01				
11				
10				

A=1

G(A,B,C,D,E)=

Minmize Determine the SOP $_{\min}$ reaization of the following functions.

$F(A,B,C,D) = \sum m(0,1,5,14,15) + \sum d(4,13)$
$G(A,B,C,D) = \sum m(0,6,7,9,10,12) + \sum d(2,4,8,13)$

AB\CD	00	01	11	10
00				
01				
11				
10				

F

AB\CD	00	01	11	10
00				
01				
11				
10				

G

Chapter 4

Combinational Logic Building Blocks

Helpful Stuff

The following is a list of the devices covered in this chapter.

Nomenclature:	N:M decoder
Data Input:	1-bit D
Data Output:	M-bit vector $y = y_{M-1} \dots y_1 y_0$
Control:	N-bit vector $s = s_{N-1} \dots s_1 s_0$
Status:	none
Behavior:	$y_s = D$ all other outputs equal 0

Nomenclature:	N:1 multiplexer
Data Input:	M-bit vector $y = y_{M-1} \dots y_1 y_0$
Data Output:	1-bit F
Control:	$log_2(N)$-bit vector $s = s_{log_2(N)} \dots s_1 s_0$
Status:	none
Behavior:	$F = y_s$

Nomenclature:	N-bit adder subtractor
Data Input:	two N-bit vectors A and B
Data Output:	N-bit vector s
Control:	1-bit c
Status:	1-bit ovf
Behavior:	if c=0 then $s = A + B$ else $s = A - B$

Nomenclature:	N-bit comparator
Data Input:	two N-bit vectors X and Y
Data Output:	none
Control:	none
Status:	1-bit G, L, E
Behavior:	

cond	E	L	G
$X = Y$	1	0	0
$X < Y$	0	1	0
$X > Y$	0	0	1

Definitions

Data in

Control

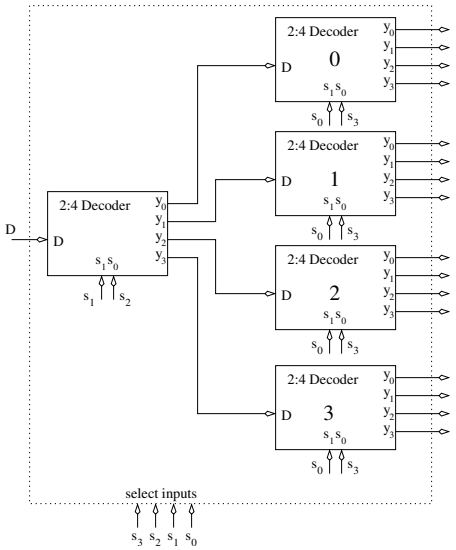
Data out

Status

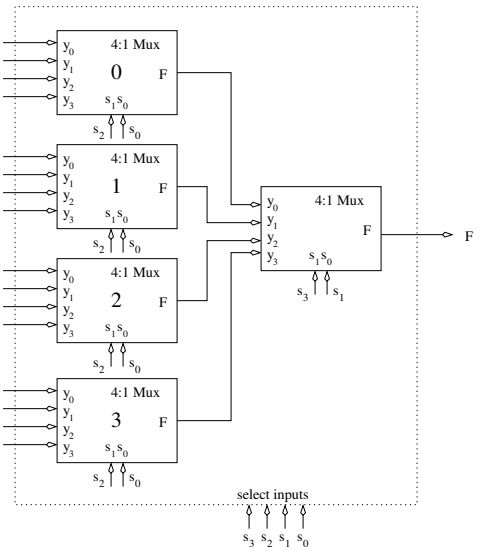
Problems

Here are a couple of problems to work on.

Label the outputs of the decoder.



Label the inputs of the mux.



Build the circuit. `if (Y<4) then X=2 else X=3`

Build the circuit. `if (Y>=4) then X=Y+2 else X=Y+3`

Build the circuit. `if (Y+1>15) then X=2 else X=3`

Chapter 5

Sequential Circuits

5.1 Helpful Stuff

When the bit is stored.

1. Latches - a latch continuously samples its inputs. Latches do not have a clock input.

2. Clocked latches - a clocked latch samples its inputs when the clock input equals 1. When the clock input equals 0 the clocked latch does not change the currently stored bit.

3. Flip flops - a flip flop samples its input when the its clock input rises. The clock is said to rise when it goes from a logic 0 to a logic 1. When the clock input is not rising the flip flop does not change the currently stored bit.

How the input(s) is transformed into a stored bit.

D	Q+
0	0
1	1

T	Q+
0	Q
1	Q'

S	R	Q+
0	0	Q
0	1	0
1	0	1
1	1	x

J	K	Q+
0	0	Q
0	1	0
1	0	1
1	1	Q'

The *When* and *How* dimensions are arranged in the following table.

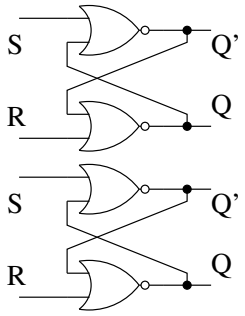
	Latch	Clocked Latch	Flip Flop
D			
T			
SR			
JK			

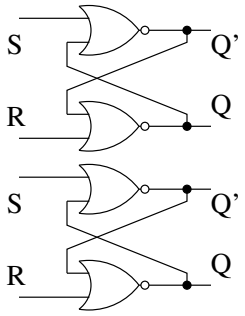
5.2 Problems

Here are some problems to work on.

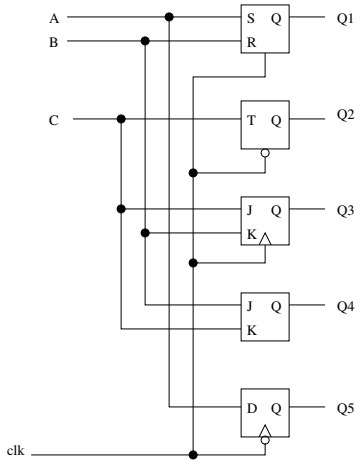
Complete the state table with the help of the following figures.

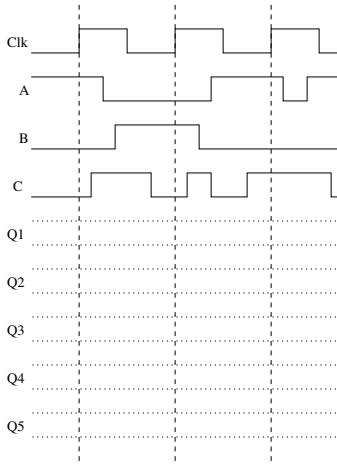
S	R	Q^+	Q'^+
0	0		
0	1		
1	0		
1	1		





Complete the timing diagram.





Chapter 6

Sequential Building Blocks

6.1 Helpful Stuff

Here are the devices introduced in this chapter.

Nomenclature:	N-bit register
Data Input:	N-bits vector $D = d_{N-1} \dots d_1 d_0$.
Data Output:	N-bit vector $Q = q_{N-1} \dots q_1 q_0$
Control:	1-bit c
Status:	none
Others:	1-bit edge-sensitive clock. 1-bit asynchronous active low reset.

Behavior:

reset	clk	C	D	Q^+	comment
0	x	x	x	0	reset
1	0,1,falling	x	x	Q	hold
1	rising	0	x	Q	hold
1	rising	1	D	D	load

Nomenclature:	N-bit shift register with parallel load
Data Input:	N-bits vector $D = d_{N-1} \dots d_1 d_0$.
Data Output:	N-bit vector $Q = q_{N-1} \dots q_1 q_0$
Control:	2-bits $c = c_1 c_0$
Status:	none
Others:	1-bit edge-sensitive clock. 1-bit asynchronous active low reset.

Behavior:

reset	clk	C	D	Q^+	comment
0	x	xx	x	0	reset
1	0,1,falling	xx	x	Q	hold
1	rising	00	x	Q	hold
1	rising	01	x	$Q >> 1$	shift right
1	rising	10	x	$Q << 1$	shift left
1	rising	11	x	D	load

Nomenclature:	N-bit counter with parallel load
Data Input:	N-bits vector $D = d_{N-1} \dots d_1 d_0$.
Data Output:	N-bit vector $Q = q_{N-1} \dots q_1 q_0$
Control:	2-bits $c = c_1 c_0$
Status:	none
Others:	1-bit edge-sensitive clock. 1-bit asynchronous active low reset.
Behavior:	

reset	clk	C	D	Q^+	comment
0	x	xx	x	0	reset
1	0,1,falling	xx	x	Q	hold
1	rising	00	x	Q	hold
1	rising	01	x	D	count up
1	rising	10	D	D	count up
1	rising	11	x	D	load

Nomenclature:	three state buffer
Data Input:	1-bit X.
Data Output:	1-bit Y
Control:	1-bit c
Status:	none
Others:	none
Behavior:	Output equals input when $C = 1$ otherwise output disconnected from input.

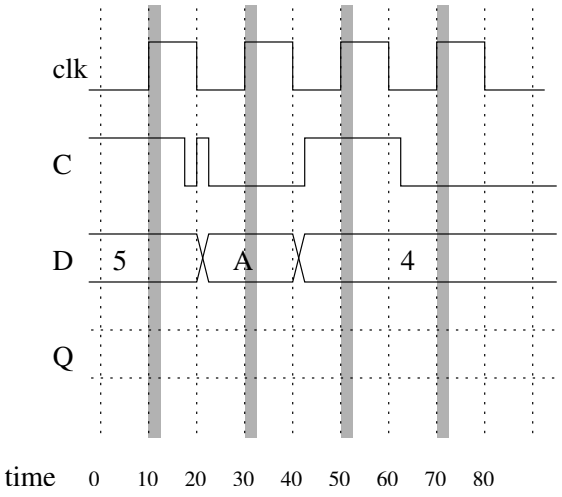
Nomenclature:	NxM RAM (random access memory)
Data Input:	M-bit vector $D = d_{M-1} \dots d_1 d_0$ $log_2(N)$-bit address $A = a_{log_2(N)-1} \dots a_1 a_0$
Data Output:	M-bit vector $D = d_{M-1} \dots d_1 d_0$
Control:	1-bit CS (chip select), RE (Read enable), WE (write enable)
Status:	none
Others:	none
Behavior:	

A	CS	RE	WE	D	Note
x	0	x	x	Z	RAM deactivated
x	1	0	0	Z	RAM deactivated
A	1	0	1	D	RAM[A] = D (write)
A	1	1	0	RAM[A]	D = RAM[A] (read)

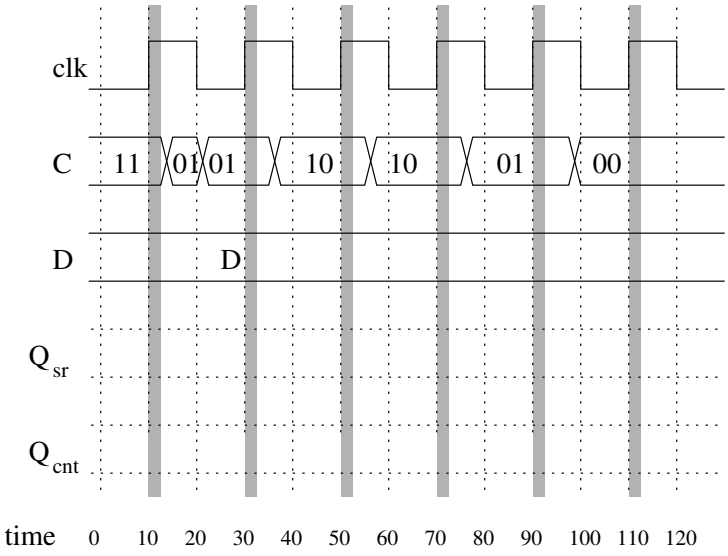
	Left	Right
Arithmetic	$x_2x_1x_00$	$x_3x_3x_2x_1$
Circular	$x_2x_1x_0x_3$	$x_0x_3x_2x_1$
Logical	$x_2x_1x_00$	$0x_3x_2x_1$

6.2 Problems

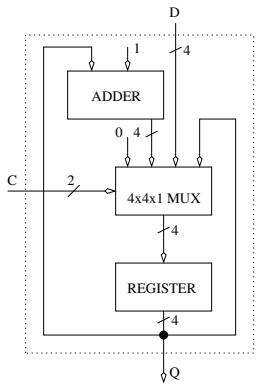
Plot Complete the timing diagram for the register. You may assume that Q is initially 0.



Plot Let Q_{sr} is the output of a logical shift register (assume that 0 is shifted into the vacated bit position. Let Q_{cnt} is the output of a counter.

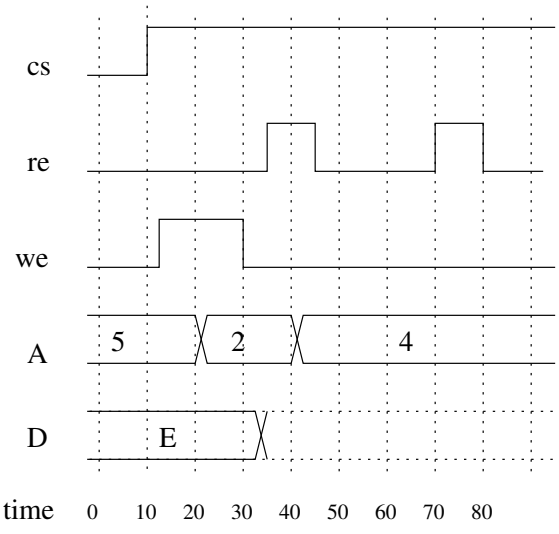


Label the mux inputs. Make the resulting circuit operate according to the truth table shows at left.



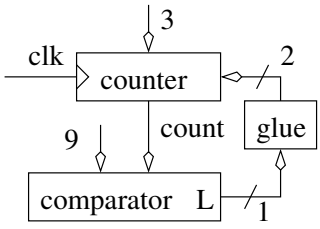
reset	clk	C	D	Q^+	comment
0	x	xx	x	0	reset
1	0,1,falling	xx	x	Q	hold
1	rising	00	x	Q	hold
1	rising	01	x	0	clear
1	rising	10	D	$Q+1$	count up
1	rising	11	x	D	load

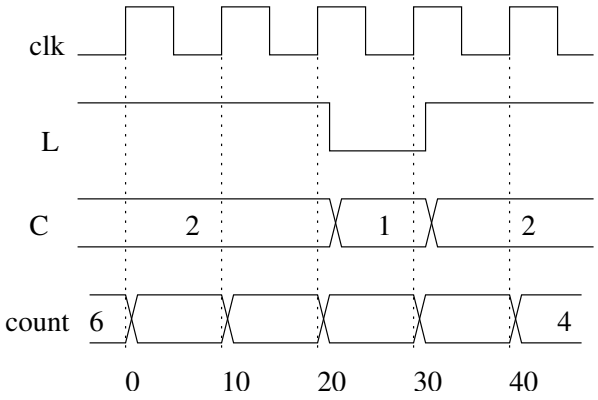
Complete the timing diagram. Note any changes in the RAMs content.



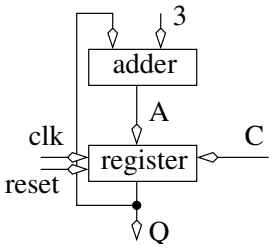
0	0110
1	1010
2	1101
3	0010
4	1000
5	0001
6	1101
7	1111

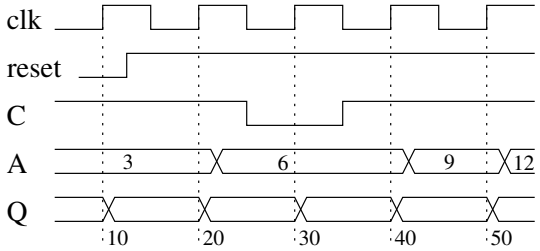
Complete the timing diagram. Use the counter control table from page 128. Assume that c1=L and c0=L'.



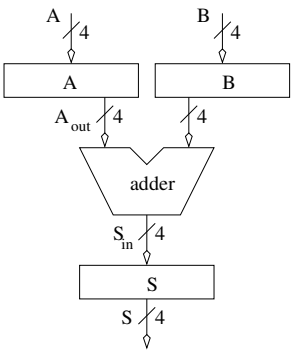


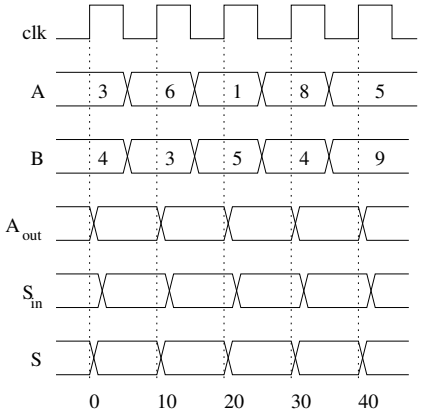
Complete the timing diagram. Put "u" in spaces where the output is undefined.



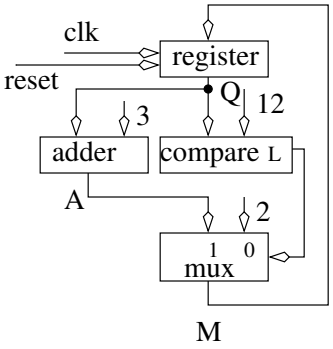


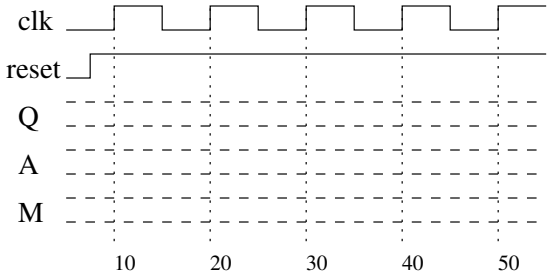
Complete the timing diagram. Put "u' in spaces where the output is undefined.



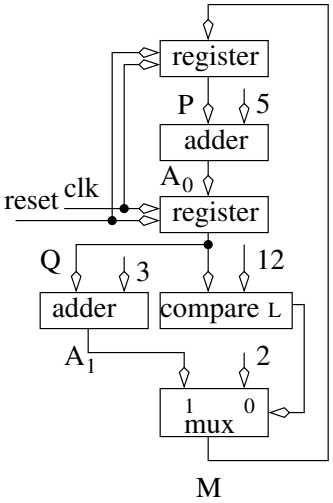


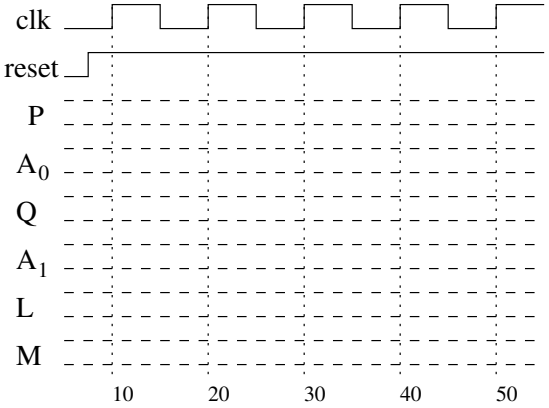
Complete the timing diagram for the following circuit. Note the labels of the signals.





Complete the timing diagram for the following circuit. Note the labels of the signals.

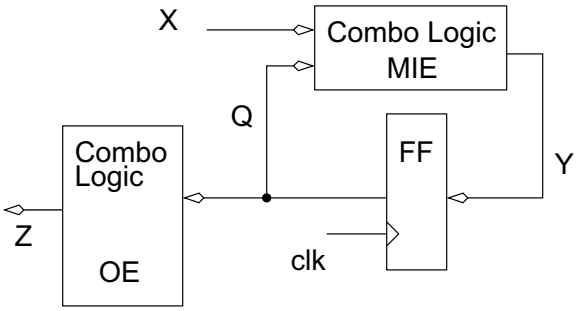




Chapter 7

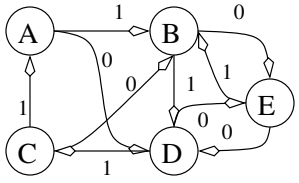
Finite State Machines

7.1 Helpful Stuff



7.2 Problems

Convert the state diagram into a state table. The input variable is X.



CURRENT STATE	X=0	X=1
A		
B		
C		
D		
E		

entries should be next state

Convert the state table into a transistion kmap. Use the state assignment provided.

State	$Q_2Q_1Q_0$
A	010
B	101
C	100
D	001
E	011

$Q_2Q_1 \backslash Q_0X$	00	01	11	10
00				
01				
11				
10				

entries should be $Q_2^+Q_1^+Q_0^+$

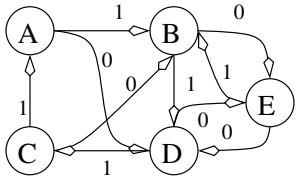
Determine the memory input equations. Use the transition kmap given in the previous problem.

$Q_2Q_1 \backslash Q_0X$	00	01	11	10
00				
01				
11				
10				

$Q_2Q_1 \backslash Q_0X$	00	01	11	10
00				
01				
11				
10				

$Q_2Q_1 \backslash Q_0X$	00	01	11	10
00				
01				
11				
10				

Write the MIEs Assume a 1's hot encoding of the states.



$D_A=$
$D_B=$
$D_C=$
$D_D=$
$D_E=$
$Z=$

Write the MIEs Use the state Assume a 1's hot encoding of the states.

Q	X=0	X=1
A	C	C
B	D	A
C	D	E
D	A	E
E	C	B

$D_A=$
$D_B=$
$D_C=$
$D_D=$
$D_E=$
$Z=$

Given the state table and state assignment determine the MIEs.

State	ND=00	ND =
0	0	10
5	5	15
10	10	20
15	15	25
20	20	30
25	25	35
30	30	35
35	0	0

$Q_1Q_0\backslash ND$	00	01	11	10
00				
01				
11				
10				

$Q_2 = 0$

$Q_1Q_0\backslash ND$	00	01	11	10
00				
01				
11				
10				

$Q_2 = 1$

Entries are $Q_2^+Q_1^+Q_0^+$

$Q_1Q_0\backslash ND$	00	01	11	10
00				
01				
11				
10				

$Q_2 = 0$

$Q_1Q_0\backslash ND$	00	01	11	10
00				
01				
11				
10				

$Q_2 = 1$

Entries are $Q_2^+ = D_2$

$Q_1Q_0\backslash ND$	00	01	11	10
00				
01				
11				
10				

$Q_2 = 0$

$Q_1Q_0\backslash ND$	00	01	11	10
00				
01				
11				
10				

$Q_2 = 1$

Entries are $Q_1+ = D_1$

$Q_1Q_0\backslash ND$	00	01	11	10
00				
01				
11				
10				

$Q_2 = 0$

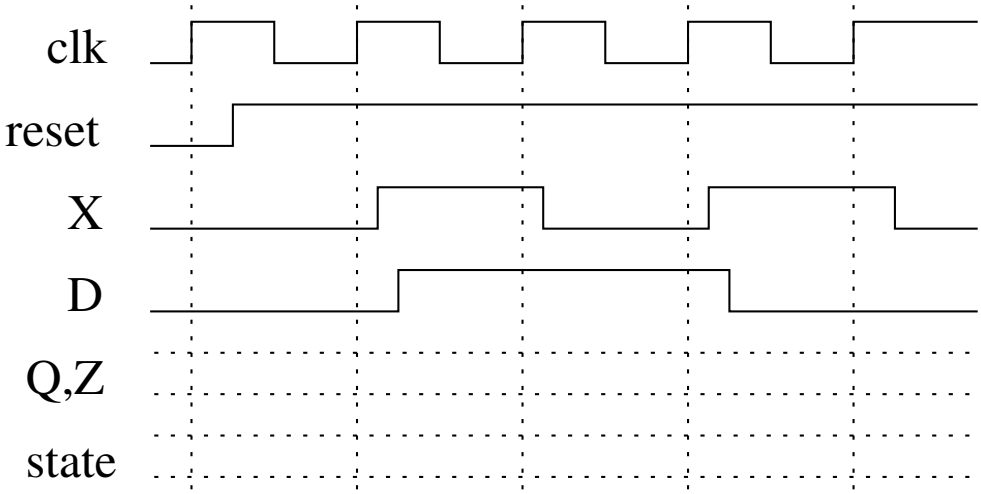
$Q_1Q_0\backslash ND$	00	01	11	10
00				
01				
11				
10				

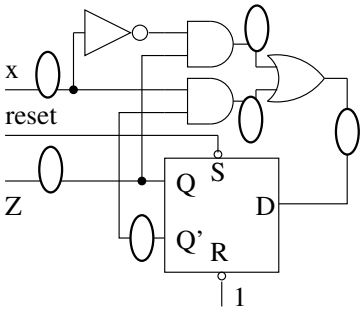
$Q_2 = 1$

Entries are $Q_0+ = D_0$

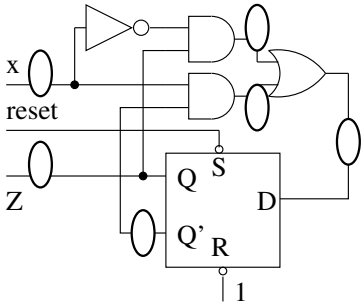
Parity Checker When data is transmitted, there is a chance that noise will corrupt some portion of the data. To detect errors, a single parity bit can be added to each data item to detect single bit errors. The value of the parity bit is determined by the number of 1's in the data packet the parity bit is being added to. In an even-parity scheme, the parity bit's value, 0 or 1, is set so that the number of 1's in the data packet plus the parity bit is an even number. For example if the data packet 1101 0011 has a total of 5 1's. The even-parity bit would be 1 and you would transmit 1101 0011 1, the last 1 being the parity bit. The parity bit is then re-computed by the receive and checked against the transmitted bit. Clearly, the transmitter and receiver must agree on the even or odd before the parity bit is useful.

The following timing diagram and circuit shows a parity checking circuit. The serial data input is x and the parity bit is z. Complete the timing diagram using the circuit diagram to plug in values at the given time and computing the signal values out of the gates and the SR flip flop. you should assume that the SR flip flop is initially reset (Q = 0). Does this circuit realize a even or odd parity circuit?

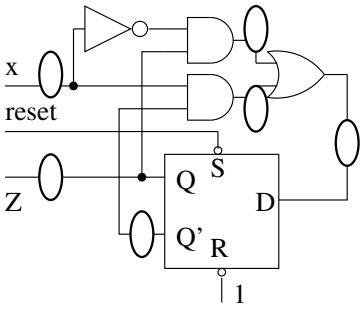




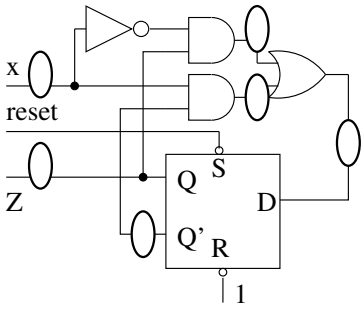
Time=5



Time=25



Time=15

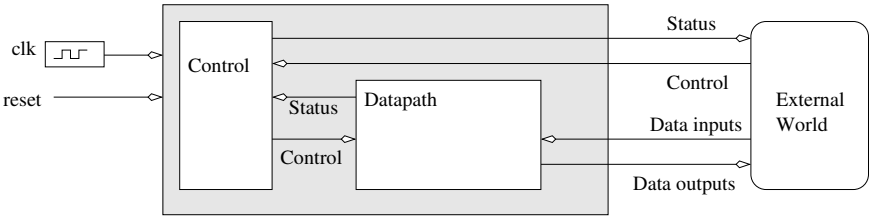


Time=35

Chapter 8

Datapath and Control

8.1 Helpful Stuff

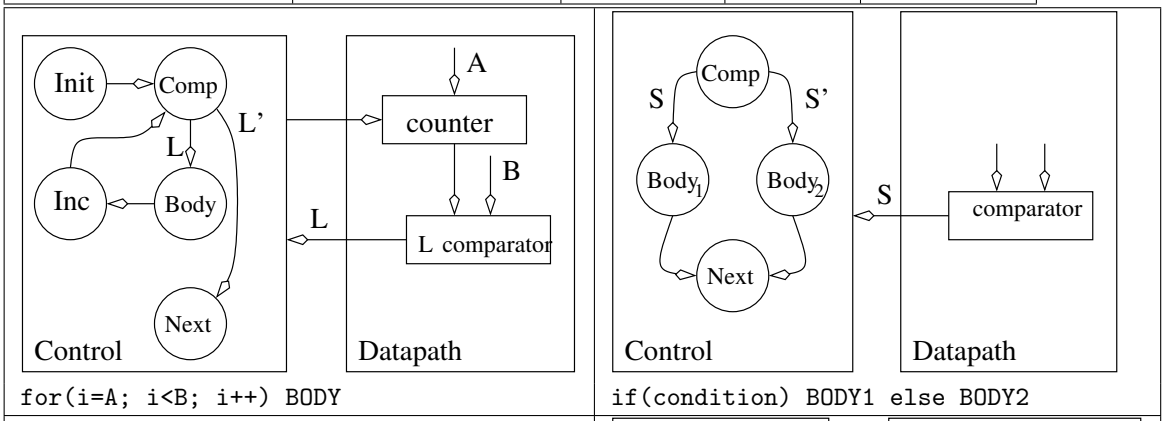


Mini-C statements

- if (condition) then BODY_1 else BODY_2
- for (i=A; i<B; i + = 1) BODY
- while(condition) BODY
- X = value

where BODY contains 0 or more statements.

Device	Data in	Data out	Status	Control
N:M Decoder	1 bit	M bits		N bits
N:1 Mux	N bits	1 bit		$\log_2(N)$ bits
MxNx1 Mux	N, each M bits	M bits		$\log_2(N)$ bits
N bit adder	2, each N bits	N bits	Overflow	
N bit add/sub	2, each N bits	N bits	Overflow	1 bit
N bit comparator	2, each N bits		3 bits	
BCD to 7-segment	4-bits	7-bits		
N-bit priority encoder	N-bits	$\log(N)$-bits		
N bit register	N bits	N bits		1 bit
N bit shift register	N bits	N bits		2 bits
N bit counter	N bits	N bits		2 bits
Three state buffer	N bits	N bits		1 bit
N:M RAM	$\log_2(N)$ bits, M bits	M bit		2 bits
N-bit Bus transceiver	N bits	N bits		2 bit

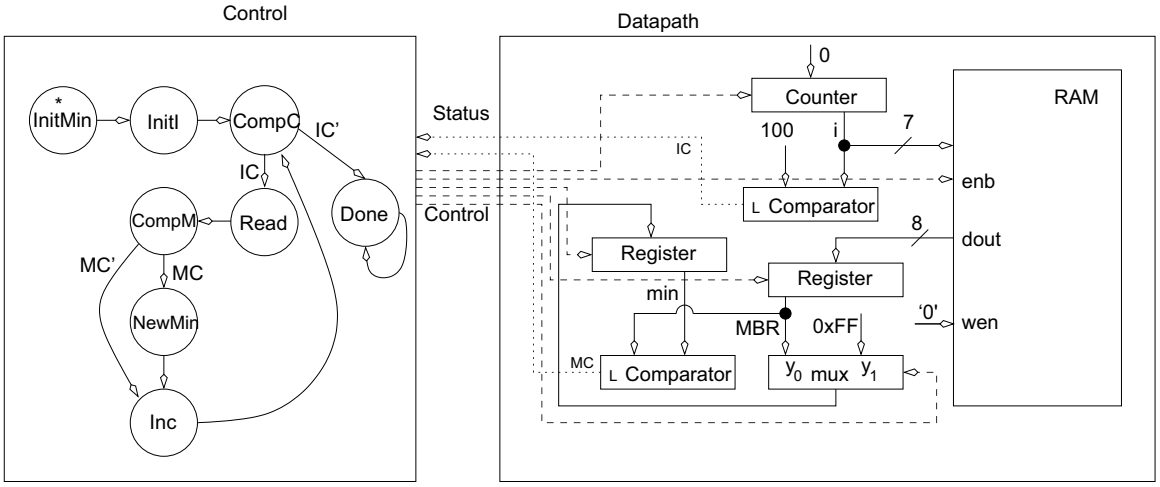


8.2 Problems

Minimum Search Design a digital circuit that looks for the smallest 8-bit integer in a 128x8 bit RAM. The numbers are stored at addresses $0 \ldots 99$, you may assume that the RAM is preloaded with data.

```
1.  min = 0xFF;                 // Set the min reg to largest value
```

```
2.  for (i=0; i<100; i++) {   // Search through the entire array
3.      MBR=RAM[i];           // read an 8-bit value from the RAM
4.      if (MBR<min) then     // If MBR is smaller than min
5.          min = MBR;        //   then set min to the smallest value
6   } // end for
```

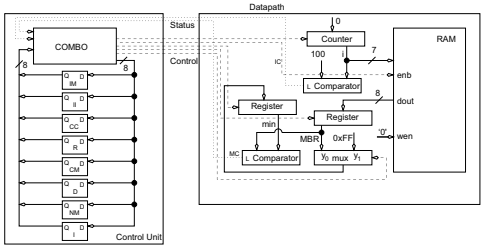


State	enb	Reg Min	Min mux	Counter	MBR
	0	0 hold	0 load FF	00 hold	0 hold
	1 read	1 load	1 load MBR	01 load	1 load
				10 count	
				11 reset	
InitMin					
InitI					
CompC					
Read					
CompM					
NewMin					
Inc					
Done					

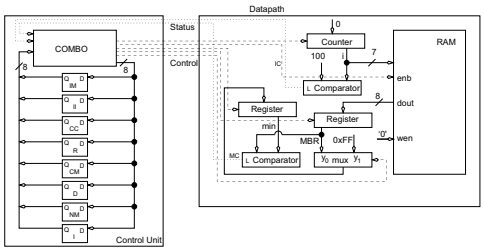
$D_{IM} =$
$D_{II} =$
$D_{CC} =$
$D_R =$
$D_{CM} =$
$D_{NM} =$
$D_I =$
$D_D =$

$Z_{ENB} =$
$Z_{RM} =$
$Z_{MM} =$
$Z_{C1} =$
$Z_{C0} =$
$Z_{MBR} =$

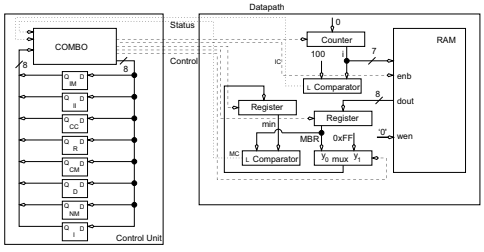
Shade the active FF and any BBBs which are read or written.



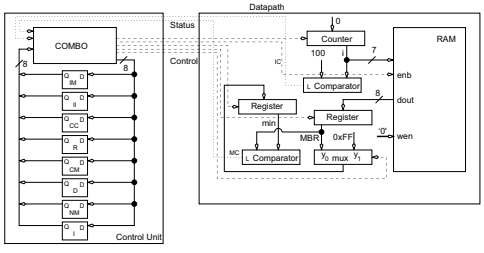
1. State **InitMin**



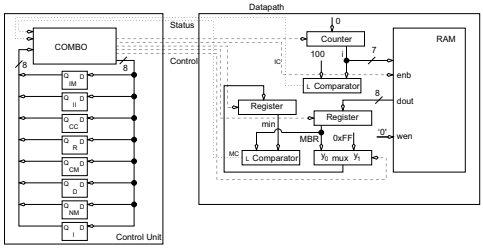
2. State **InitI**



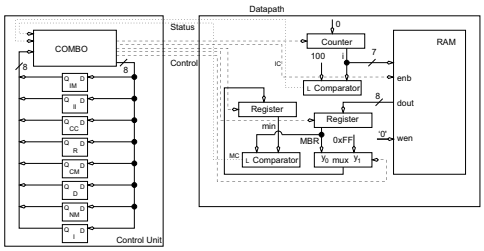
3. State **CompC**



4. State **Read**

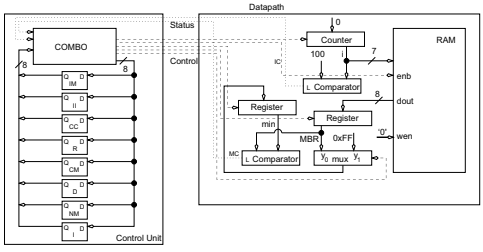


5. State **CompM**

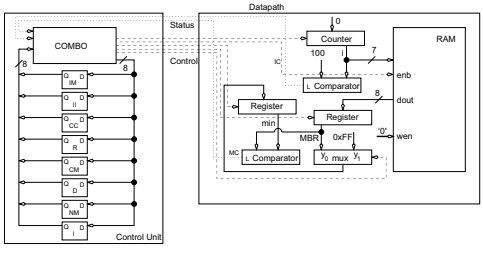


i

6. State **NewMin**



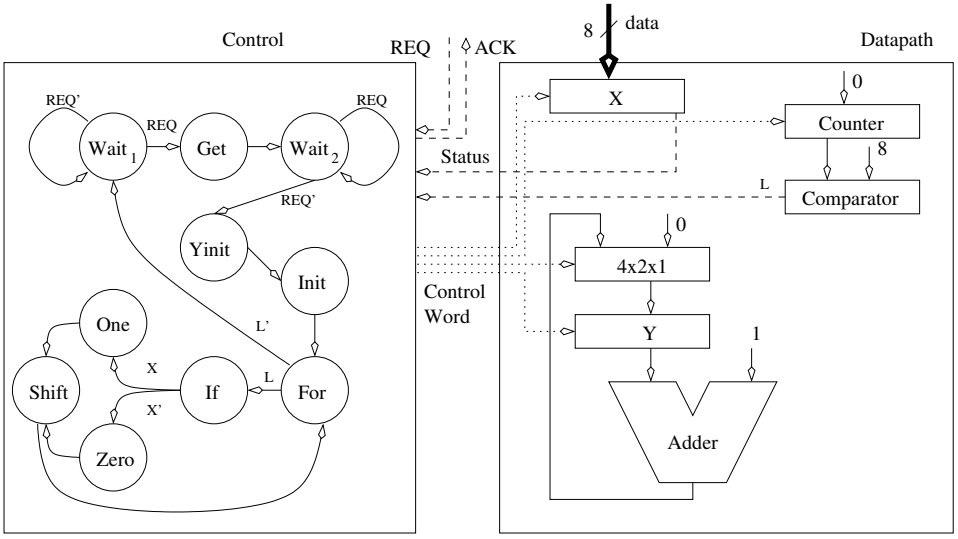
7. State **Inc**



8. State **CompC**

Bit Counter Design a digital circuit that counts the number of 1's in an 8 bit number X and puts the result into a register Y. The 8 bit number is provided by an external user using a two-line handshake protocol. The digital circuit is to play the role of the passive consumer. Your circuit should do this task forever. That is, after counting the number of 1's, you circuit should read in a new value of X.

```
1.  while(true) {                // Forever
2.      while(REQ==0);           // Wait for the REQ signal
3.      X = data;                // When we get a REQ latch data
4.      ACK = 1;
5.      while(REQ == 1);         // Wait for REQ to go low
6.      ACK = 0;                 // then drop the ACK
7.      Y = 0;                   // Clear the bit count
8.      for (i=0; i<8; i++) {    // For each bit of X
9.          if (X(0) == 1) then // If the LSB of X is a 1
10             Y = Y + 1;       //   then increment Y
11         X = X >> 1;          // Shift X to the right 1 bit
12     } // end for
13  } // end while
```



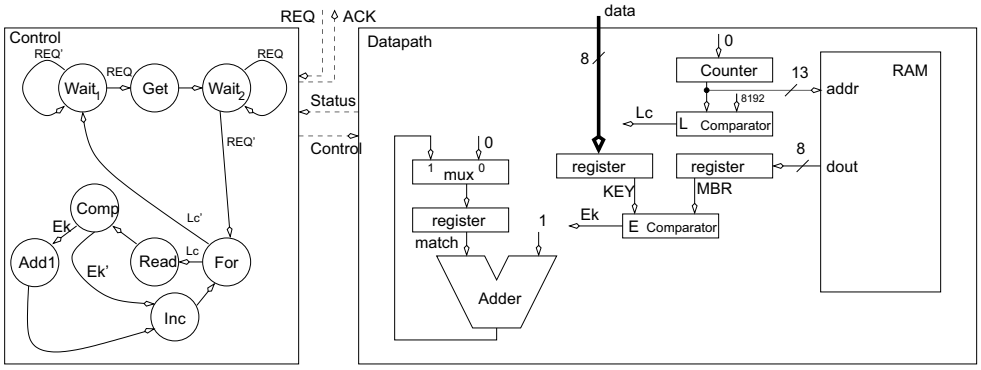
State	ACK	X	Reg Y	Y mux	Counter
		00 hold	0 hold	0 load 0	00 hold
		01 lsr			01 load
		10 lsl	1 load	1 load Add	10 count
		11 load			
Wait1					
Get					
Wait2					
Yinit					
Init					
For					
If					
One					
Zero					
Shift					

$D_{W1} =$
$D_G =$
$D_{W2} =$
$D_{YI} =$
$D_I =$
$D_F =$
$D_{If} =$
$D_{One} =$
$D_{Zer} =$
$D_{Sft} =$

$Z_{ACK} =$
$Z_X =$
$Z_{RegY} =$
$Z_{Ymux} =$
$Z_{C1} =$
$Z_{C0} =$

RAM counter Build a circuit that reads in an 8 bit value, KEY, using a two line handshake; your circuit is a passive consumer. Your circuit should search an 8kx8 RAM counting the number of words that match KEY. It takes one full clock cycle to read the RAM. You may assume that the RAM is pre-loaded with data. Your circuit should do this task forever.

```
1.    while(1) {
2.        while(REQ == 0);
3.        KEY = data;
4.        ACK = 1;
5.        while(REQ == 1);
6.        ACK = 0;
7.        match = 0;
8.        for(i=0; i<8191; i++) {
9.            MBR = RAM[i];
10.            if (MBR == KEY) {
11.                match=match+1;
12.            } // end if
13.        } // end for
14.    } // end while
```



State	ACK	mux	Reg match	Reg KEY	Counter	MBR	enb
	0	0 pass 0	0 hold	0 hold	00 hold	0 hold	0
					01 load		
	1	1 match+1	1 load	1 load	10 count	1 load	1 read
Wait1							
Get							
Wait2							
For							
Read							
Comp							
Add1							
Inc							

Extra Design a circuit that moves M consecutive words from address S (source) to address D (destination). For example, if $M = 4$, $S = 3EA$ and $D = 1FE$ then the circuit would move words 3EA, 3EB, 3EC and 3ED to address 1FE, 1FF, 200 and 201. Each of M, S, D is preloaded into a register. While this problem appears simple, its really rather treacherous. The circuit will have to handle cases where $S + M > D$. In such a case the order of the data movement must be carefully planned. In order to simplify the design, assume that $S < D$. Turn in an algorithm, datapath and control, the control word, MIEs and OEs. These circuit will need a three-state buffer to be able to both read and write to the RAM. Do not worry about the sizes of the registers or RAM.

Roulette You are to construct a digital circuit that plays a game of roulette, allows betting and keeps track of total earnings. The roulette wheel has 8 slots, labeled $1 \ldots 8$. The player can play one of the numbers straight or play even or odd. The player starts with \$10. The layout of the machine is shown in Figure 8.1.

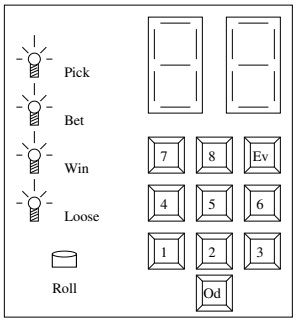


Figure 8.1: The layout of the roulette playing machine. The two 7 segment displays at the top are used for a variety of purposes.

The sequence of events is as follows:

1. The circuit lights up the PICK LED. The player enters their guess; a number between 1-8, even or odd. While holding down their guess they press the roll button.
2. The circuit displays the picked number in the left most 7 segment display. The circuit lights up the BET LED. The player enters a one digit bet between 1 to 8. While holding down their bet they press the roll button.
3. The circuit displays the bet on the rightmost 7-segment display. The player pushes and holds down the roll button. The circuit increments a mod 8 counter while the roll button is depressed. It would be nice to display the current count value on right 7-seven segment display. Since the clock cycle is on the order of milliseconds, then the user would not be able to anticipate the roll.
4. The player releases the roll button. The final roll is displayed on the rightmost 7-segment display. The circuit stops incrementing the counter and checks to see if the final value matches the players guess. If the match is correct then light the WIN LED and increment the players earnings. If the match is incorrect then light the LOOSE LED and decrement the players earnings.
5. The play hits the roll button to clear the roll information from the 7-segment displays.
6. The circuit displays the players earnings on the 7-segment display.

7. When the user pushes the roll button then goto step 1.

Set reasonable bounds on the maximum winnings. Values may be displayed in hexadecimal (you may assume that you have a hex to 7-segment display converter at your disposal).

Algorthm

```
1. cash = 10;
2. LeftSeven = blank;
3. RightSeven = cash;
4. while(cash != 0)
5. LEDarray = 1 0 0 0; // Pick Bet Win Loose
6. while(ROLL == 0);
7. guess = Datain;
8. while(ROLL == 1);
9.
10. LEDarray = 0 1 0 0; // Pick Bet Win Loose
11. LeftSeven = guess;
12. RightSeven = blank;
13. while(ROLL == 0);
14. bet = Datain;
15. while(ROLL == 1);
16.
17. LEDarray = 0 0 0 0; // Pick Bet Win Loose
18. RightSeven = bet;
19. while(ROLL == 0);
20. while(ROLL == 1) count = count + 1;
21.
22. RightSeven = count;
23. if (count == guess)
24. cash = cash + (bet *4);
25. LEDarray = 0 0 1 0; // Pick Bet Win Loose
26.  else
27. cash = cash - bet;
28. LEDarray = 0 0 0 1; // Pick Bet Win Loose
29.
30. while(ROLL == 0);
31. while(ROLL == 1);
32. RightSeven = cash;
33.
34.
35. LEDarray = 0 0 0 1; // The user is out of money
36. while(1); // Halt the machine
```

Datapath and Control

Table 8.1: The control word for the roulette circuit and its value for each state.

State	counter	guess	bet	cmux	bmux	cash	add/sub	ledmux	rmux	lmux
	00 hold	0 hold	0 hold	0 $10	0 bet	0 hold	0 add	00 (pick)	00 cash	00 cash
	01 load	0 load	1 load	1 add/sub	1 bet¡¡2	1 load	1 sub	01 (bet)	01 bet	01 guess
	10 up							10 (win)	10 count	10 blank
								11 (loose)	11 blank	
init										
wg1										
guess										
wg2										
wb1										
bet										
wb2										
wr1										
spin										
load 1										
comp										
win										
add										
loose										
sub										
wn										

Control Word

Light Show A light show consists of an endlessly repeating sequence of up to 16 frames. A frame is an illuminated pattern of LEDs on a LED bar graph. The user creates a light show by specifying the number of frames in the show, editing those frames, and then instructing the circuit to cycle through the frames. The input to the circuit comes from a standard PS/2 keyboard. The output of the circuit is displayed on a LED bar graph and a 7-segment display. The behavior of the Light Show circuit is given in Figure 8.2.

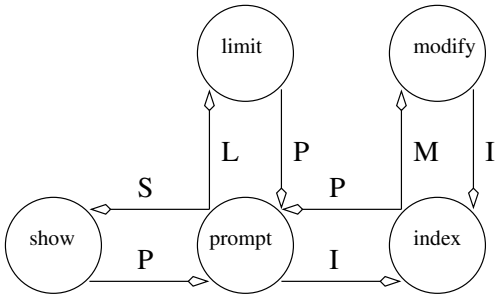


Figure 8.2: A state diagram describing the behavior of the Light Show circuit.

The Light Show circuit changes state when the users presses a key on the keyboard. For example, pressing "M" while in the **index** state causes the circuit to transition into the **modify state**. The behavior of the Light Show circuit in each of its states is described in Table 8.2.

The algorithm for the light show circuit continually scans the busy signal and the keyboard scan code. When the user presses either a "L", "I", or "S", the algorithm drops into one of the subfunctions described in Table 8.2. Assume the Light Show circuit is clocked at 16MHz. The clock rate is needed in order to create a delay of 0.25 seconds required to pace the frames during the show phase.

State	LED bar graph	7-segment	Behavior
reset	blank	blank	All sequential elements are reset
prompt	blank	"P"	Waiting for user input
limit	blank	current limit	The number of frames in the light show is called the limit. If the user enters a hex value between 0-F it is stored as the new limit.
index	current frame	current index	Each frame in the show has an index which defines its position in the show. If the user enters a hex value between 0-F, this frame will be edited in the **modify** state.
modify	current frame	current index	Each frame has eight bits which specify the state of the 8 LEDs on the bar graph. These bits are toggled by pressing the corresponding key. For example, if LED 5 is on, pressing "5", causes LED 5 to go off.
show	cycle through frames	current index	Consecutive frames are displayed on the LED bar graph at around 4Hz. After the last frame is displayed, the circuit loops back to the 0th frame.

Table 8.2: The behavior of the Light Show circuit in each of its states.

```
1.  while(1) {
2.      HexDisplay = "P"
3.      if (!busy and IsL(ScanCode)) {
4.          while (!busy' and !IsP(ScanCode)) {
5.             HexDisplay = Hex2Seven(limit);
6.             BarGraph = 0x00;
7.             if (!busy and IsHex(ScanCode)) {
8.                  limit = Scan2Hex(ScanCode);
9.                  while(!busy);
10.     }   }  }

11.     if (!busy and IsI(ScanCode)) {
12.         while (busy or !IsP(ScanCode)) {
13.             HexDisplay = Hex2Seven(index);
14.             if (!busy and IsHex(ScanCode)) {
15.                index = Scan2Hex(ScanCode);
16.                 while(!busy);
17.             }
```

```
18.             if (!busy and IsM(ScanCode)) {
19.                 while (busy or !IsI(ScanCode)) {
20.                     HexDisplay = Hex2Seven(index);
21.                     BarGraph = RAM[index];
22                      while(!busy);
23                      while(busy);
24.                     if (!busy and IsOct(ScanCode)) {
25.                         RAM[index] = Flip(IsOct(ScanCode),RAM[index]);
26.     }   }   }   }   }

27.     if (!busy and IsS(ScanCode)) {
28.     index = 0;
29.         while(!busy and !IsP(ScanCode)) {
30.             BarGraph = RAM[index];
31.             HexDisplay = Hex2Seven(index);
32.             for (timer=0; timer<2^22; timer++);
33.             index += 1;
34.             if (index == limit) index=0;
35.  }  }
```

State	bar mux	7-seg mux	hex mux	index count	delay count	limit register	bargraph register	CS	R/W'	tsb	flip
	0 0x00 1 bar	0 index 1 limit	0 7-seg 1 "P"	00 hold 01 cnt 10 load 11 reset	00 hold 01 cnt 10 load 11 reset	0 hold 1 load	0 hold 1 load	0 off 1 on	0 write 1 read	0 tri 1 pass	0 pass 1 flip
prompt											
set limit											
load limit											
waitlimit											
reset inde											
load frame											
wait frame											
inc index											
comp index											
set index											
load index											
wait index											
modify											
modify read											
modify write											
wait !busy											
wait busy											

Table 8.3: The control word for the LightShow circuit and its value for each state.

Figure 8.3: The datapath and control for the Light show circuit.