Embedded System Design

A Unified Hardware/Software Introduction

Solution Manual

Frank Vahid
Department of Computer Science and Engineering
University of California, Riverside

Tony Givargis
Department of Information and Computer Science
University of California, Irvine

John Wiley & Sons, Inc.

Instructors: Please do not post this or any portion of this manual in electronic format.

CHAPTER 1: *Introduction*

1.1 What is an embedded system? Why is it so hard to define?

An embedded system is nearly any computing system other than a desktop computer. Embedded systems are hard to define because they cover such a board range of electronic devices.

1.2 List and define the three main characteristics of embedded systems that distinguish such systems from other computing systems.

1. single functioned : executes a specific program repeatedly
2. tightly constrained : the specified cost, performance, and power must be met
3. reactive and real-time: system must quickly and constantly react to outside stimuli

1.3 What is a design metric?

A design metric is a measure of an implementation's features such as cost, size, performance, and power.

1.4 List a pair of design metrics that may compete with one another, providing an intuitive explanation of the reason behind the competition.

(*Note: answers will vary*)
Size and performance compete with one another. Reducing the size of an implementation may cause performance to suffer and vice versa.

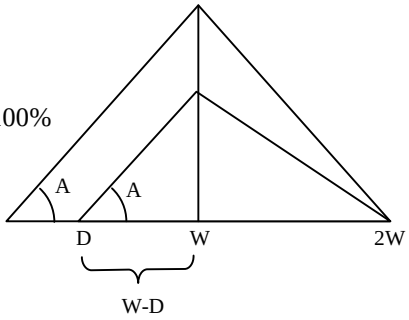
1.5 What is a "market window" and why is it so important for products to reach the market early in this window?

A market window is the time period in which a given product is in demand, specifically the time in which this product would yield the highest sales. Missing the window could mean significant loss in sales.

1.6 Using the revenue model of Figure 1.4(b), derive the percentage revenue loss equation for any rise angle, rather than just for 45 degrees (*Hint*: you should get the same equation).

tan A = opposite / adjacent
opposite = tan A * adjacent

Revenue loss = ((on time - delayed) / on time) * 100%



Area of on time = 1/2 * base * height
= 1/2 * 2W * tan A * W
= tan A * W^2

Area of delay = 1/2 * base * height
= 1/2 * (W-D+W) * (tan A * (W-D))
= 1/2 * (2W-D) * (tan A * (W-D))

Revenue Loss = [((tan A * W^2) - (1/2 * (2W-D) * tan A * (W-D))) / (tan A * W^2)] * 100%
= [(W^2 - (1/2 * (2W-D) * (W-D))) / W^2] * 100%
= [(W^2 - 1/2*(2W^2 - 2WD - WD - D^2)) / W^2] * 100%
= [(W^2 - 1/2*(2W^2 - 3WD - D^2)) / W^2] * 100%
= [(2W^2 - 2W^2 + 3WD + D^2) / 2W^2] * 100%
= [(3WD + D^2) / 2W^2] * 100%

1.7 Using the revenue model of Figure 1.4(b), compute the percentage revenue loss if D = 5 and W = 10. If the company whose product entered the market on time earned a total revenue of $25 million, how much revenue did the company that entered the market 5 months late lose?

percentage revenue loss = (D * (3W - D) / 2W^2) * 100%
= (5 * (3 * 10 - 5) / 2 * 10^2) * 100%
= (5 * 25 / 200) * 100%
= 62.5%

revenue loss = $25,000,000 * 0.625
= $15,625,000

1.8 What is NRE cost?

NRE cost is the nonrecurring engineering cost. It is the one time monetary cost of designing the system.

1.9 The design of a particular disk drive has an NRE cost of $100,000 and a unit cost of $20. How much will we have to add to the cost of each product to cover our NRE cost, assuming we sell: (a) 100 units, and (b) 10,000 units?

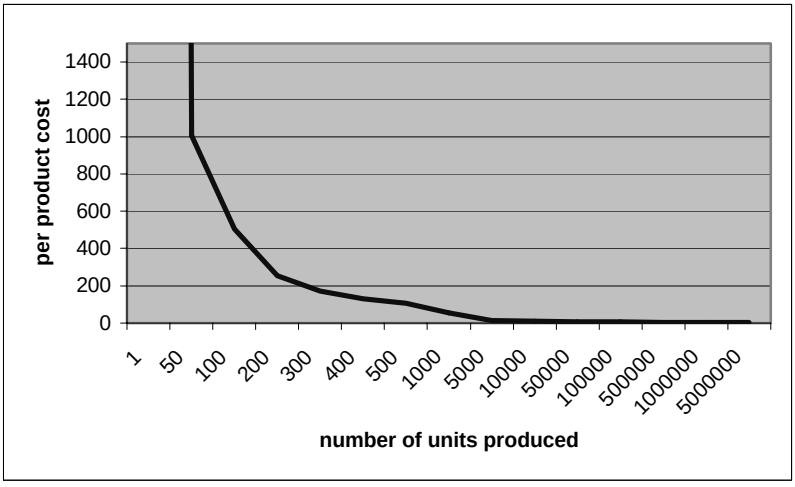
(a) added cost = NRE / # units produced
= $100,000 / 100
= $1,000

(b) added cost = NRE / # units produced
= $100,000 / 10,000
= $10

1.10 Create a graph with the *x*-axis the number of units and the *y*-axis the product cost. Plot the per-product cost function for an NRE of $50,000 and a unit cost of $5.

Number of Units	Per-product Cost
1	50005
50	1005
100	505
200	255
300	171.6666667
400	130
500	105
1000	55

Number of Units	Per-product Cost
5000	15
10000	10
50000	6
100000	5.5
500000	5.1
1000000	5.05
5000000	5.01



1.11 For a particular product, you determine the NRE cost and unit cost to be the following for the three listed IC technologies: FPGA: ($10,000, $50); ASIC: ($50,000, $10); VLSI: ($200,000, $5). Determine precise volumes for which each technology yields the lowest total cost.

Total cost = NRE cost + (unit cost * # units produced)

FPGA = 10,000 + 50x
ASIC = 50,000 + 10x
VLSI = 200,000 + 5x

10,000 + 50x = 50,000 + 10x
40x = 40,000
x = 1,000

50,000 + 10x = 200,000 + 5x
5x = 150,000
x = 30,000

tech ology	amoun
FPGA	< 1,000 units
ASIC	1,000 - 30,000 units
VLSI	> 30,000 units

1.12 Give an example of a recent consumer product whose prime market window was only about one year.

Note: Answers will vary.

1.13 Create an equation for total profit that combines time-to-market and NRE/unit cost considerations. Use the revenue model of Figure 1.4(b). Assume a 100-month product lifetime, with peak revenue of $100,000 month. Compare use of a general-purpose processor having an NRE cost of $5,000, a unit cost of $30, and a time-to-market of 12 months (so only 88 months of the product's lifetime remain), with use of a single-purpose processor having an NRE cost of $20,000, a unit cost of $10, and a time-to-market of 24 months. Assume the amount added to each unit for profit is $5.

$$\text{total profit} = \left[\frac{\left[\text{revenue} * (1 - \text{fraction revenue loss})\right] - \text{NRE}}{\text{unit cost} + 5} \right] * 5$$

Product A: General Purpose Processor
NRE = $5,000
unit cost = $30
time to market = 12 months (88 months of market window remaining)

Revenue = 1/2 base * heigth
= 1/2 * 100 months * $100,000 per month
= $5,000,000

Fraction Revenue Loss = $(D * (3W-D)) / 2W^2$
$= (12*(3*50-12)) / (2*50^2)$
= 0.3312

Total Profit = [((5,000,000 * (1-0.3312)) - 5,000) / (30 + 5)] * 5
= [(3,344,000 - 5,000) / 35] * 5
= 95400 * 5
= $477,000

Product B: Single Purpose Processor
NRE = $20,000
unit cost = $10
time to market = 24 months (76 months of market window remaining)

Revenue = $5,000,000

Fraction Revenue Loss = $(D * (3W-D)) / 2W^2$
$= (24*(3*50-24)) / (2*50^2)$
= 0.6048

Total Profit = [((5,000,000 * (1-0.6048) - 20,000) / (10 + 5)] * 5
= 130,400 * 5
= $652,000

1.14 Using a spreadsheet, develop a tool that allows us to plug in any numbers for problem 1.13 and generates a revenue comparison of the two technologies.

Note: This answer is currently unavailable as it requires the use of a spreadsheet program.

1.15 List and define the three main processor technologies. What are the benefits of using each of the three different processor technologies?

1. General Purpose Processor
A general-purpose processor consists of a controller, general datapath, and a program memory. Designers can load a variety of programs into memory. Since a general--purpose processor is flexible a large number of these devices can be sold for a variety of applications. Using a general-purpose processor, the time to market and NRE cost is low.

2. Single Purpose Processor
A digital circuit designed to execute exactly one program. Since a single purpose processor is designed for only one task it will have high performance, small size, and low power consumption.

3. Application Specific Instruction Set Processor (ASIP)
This is a compromise between a general purpose processor and a single purpose processor. It is a programmable processor optimized for a particular class of applications. An ASIP allows flexibility while still achieving good performance, size, and power consumption.

1.16 List and define the three main IC technologies. What are the benefits of using each of the three different IC technologies?

1. Full-Custom / VLSI
All layers of this IC is optimized and will therefore yield excellent performance, small size, and low power consumption.

2. Semi-custom ASIC (Gate Array and Standard Cell)
The lower layers are fully or partially built, leaving the upper layers to be finished by the designer. This yields good performance, size, and a lower NRE cost than that of a full-custom IC.

3. Programmable Logic Device (PLD)
All layers already exist so the designer can purchase an IC. PLDs offer low NRE costs and almost instant IC availability. PLDs are well suited for rapid prototyping.

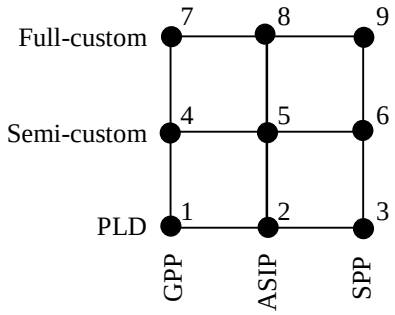
1.17 List and define the three main design technologies. How are each of the three different design technologies helpful to designers?

1. Compilation/Synthesis
This design technology allows a designer to define functionality in an abstract manner and the technology will automatically generate the lower-level implementation details. Designers benefit because they do not have to be aware of the lower-level implementation details, which will increase their productivity.

2. Libraries/IP
Libraries and IP are essentially catalog of pre-existing implementations. This benefits the designers because they don't have to "re-invent the wheel".

3. Test/Verification
This design technology ensures that the functionality of a system is correct. This saves the designer time by preventing debugging at a low level.

1.18 Create a 3*3 grid with the three processor technologies along the *x*-axis, and the three IC technologies along the *y*-axis. For each axis, put the most programmable form closest to the origin, and the most customized form at the end of the axis. Explain features and possible occasions for using each of the combinations of the two technologies.



1. GPP + PLD
The device is flexible since it can run a variety of applications. Since the designers are using a PLD they can easily change the features of the GPP. This situation may occur in the development and testing of a GPP.

2. AISP + PLD
This device will run a class of applications. Since the designers are using a PLD they can easily change the features of the ASIP. This situation may be used when developing and testing an ASIP.

3. SPP + PLD
This device will run a single program. However, since the designers are using a PLD they can easily modify the program or make corrections. This situation may occur when developing and testing an SPP.

4. GPP + Semi-Custom

This device will execute a variety of applications. Since we are using semi-custom technology, once programming/connection are done the designers cannot change them. A situation where this may occur is when the GPO is already developed and tested and a designers desire an IC but not the cost and time of making a full-custom IC.

5. ASIP + Semi-Custom
This device will execute a class of applications well since it was optimized with those applications in mind. Once the programming/connections are done, changes cannot be made. This situation arises after an ASIP is fully developed and tested and an IC is desired but the time and cost of full-custom IC is not.

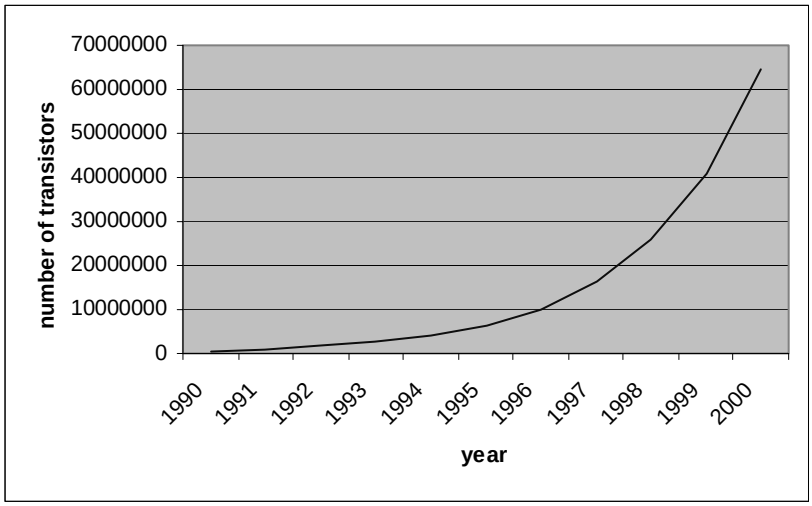
6. SPP + Semi-Custom
This device will execute a single program. Once programming/connections are done the program will not change. This situation may occur once the final development and testing is done and an IC is desired but the time and cost of full-custom IC is not.

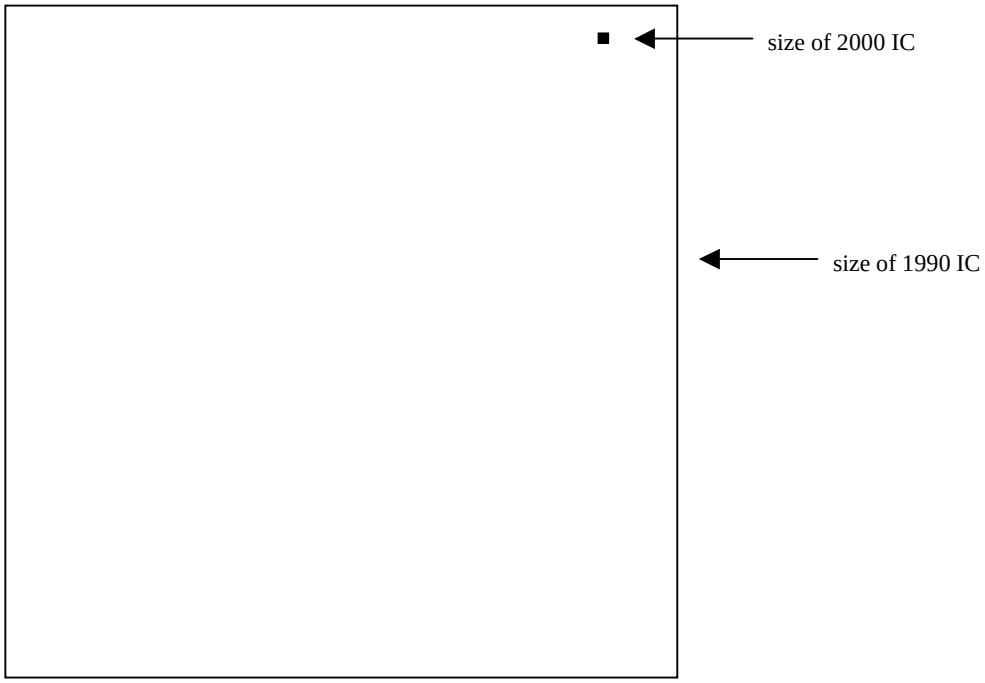
7. GPP + Full-Custom
This device will execute a variety of programs. Each layer has been optimized so that the device will yield excellent performance and power consumption. This option is one of the most expensive and is desired when the performance, size, and power trade offs are of greater concern then cost and time. This situation may arise after final development and testing. Moreover, a large quantity of ICs will be produced thereby reducing the NRE per unit.

8. ASIP + Full-Custom
This device will execute a domain specific application well. Each layer has been optimized so the device will yield excellent performance and power consumption. This option is one of the most expensive and is desired when the performance, size, and power trade offs are of greater concern then cost and time. This situation may occur after final development and testing. Moreover, a large quantity of ICs will be produced thereby reducing the NRE per unit.

9. SPP + Full-Custom
This device will execute a single program. Each layer has been optimized such that the device will yield excellent performance and power consumption. This option is one of the most expensive and is desired when the performance, size, and power trade offs are of greater concern then cost and time. This situation may occur after final development and testing. Moreover, a large quantity of ICs will be produced thereby reducing the NRE per unit.

1.19 Redraw Figure 1.9 to show the transistors per IC from 1990 to 2000 on a linear, not logarithmic, scale. Draw a square representing a 1990 IC and another representing a 2000 IC, with correct relative proportions.





1.20 Provide a definition of Moore's law.

IC transistor capacity doubles every 18 months.

1.21 Compute the annual growth rate of (a) IC capacity, and (b) designer productivity.

(a) Annual growth rate of IC capacity

X = IC capacity of current year
Y = IC capacity of current year + 3 years
r = growth rate

every 18 months chip capacity doubles therefore in 3 years chips size is 4x as large

$Y = X * r * r * r = 4X$
$X * r^3 = 4X$
$r = \sqrt[3]{4}$
$r = 1.587$

(b) Annual growth rate of designer productivity

1981 productivity = 100 transistors / designer month
2002 productivity = 5,000 transistors / designer month
X = productivity in 1981
2002 - 1981 = 21 years

$X * r21 = (5{,}000/100) X$
$r^{21} = 50$
$r = \sqrt[21]{50}$
$r = 1.205$

1.22 If Moore's law continues to hold, predict the approximate number of transistors per leading edge IC in (a) 2030, (b) 2050.

transistors in year B = # transistors in year A * annual growth rate$^{(\text{year B - year A})}$

(a) # transistors in 2030 = # transistors in 2002 * annual growth rate$^{(2030-2002)}$
= 150,000,000 * 1.587^{28}
= 61,900,000,000,000

(b) # transistors in 2050 = # transistors in 2002 * annual growth rate$^{(2050-2002)}$

$$= 150{,}000{,}000 * 1.587^{48}$$
$$= 6{,}360{,}000{,}000{,}000$$

1.23 Explain why single-purpose processors (hardware) and general-purpose processors are essentially the same, and then describe how they differ in terms of design metrics.

Single purpose processor and general-purpose processors are essentially the same in terms of a designer's job. With the maturation of design tools, both designers start with writing sequential programs. Depending on whether the designer's goal is a single purpose processor or a general-purpose processor, the synthesis tool would then produce gates or machine instructions. Hardware excels in performance, size and power consumption. Software enables flexibility, low NRE cost, and rapid product development.

1.24 What is a "renaissance engineer," and why is it so important in the current market?

It is a designer who is comfortable and familiar with various technologies. Specifically, it is a designer who knows both the software and hardware design aspects involved in product development. For the best optimization of systems a designer must be able to move functionality between hardware and software at any stage in development.

1.25 What is the design gap?

The design gap is the inability for designer productivity to keep pace with chip capacity growth.

1.26 Compute the rate at which the design productivity gap is growing per year. What is the implication of this growing gap?

Based on question 1.21 we have determined that the annual IC growth rate is 1.587 and the annual designer productivity growth rate is 1.205.

y_o = start year
y = end year

design gap(y) = $1.587^{(y-yo)} - 1.205^{(y-yo)}$

The design gap is not a simple linear function. The gap increases as each year goes by. This means that each year the designers are able to use a smaller and smaller percentage of transistors available.

1.27 Define what is meant by the "mythical man-month."

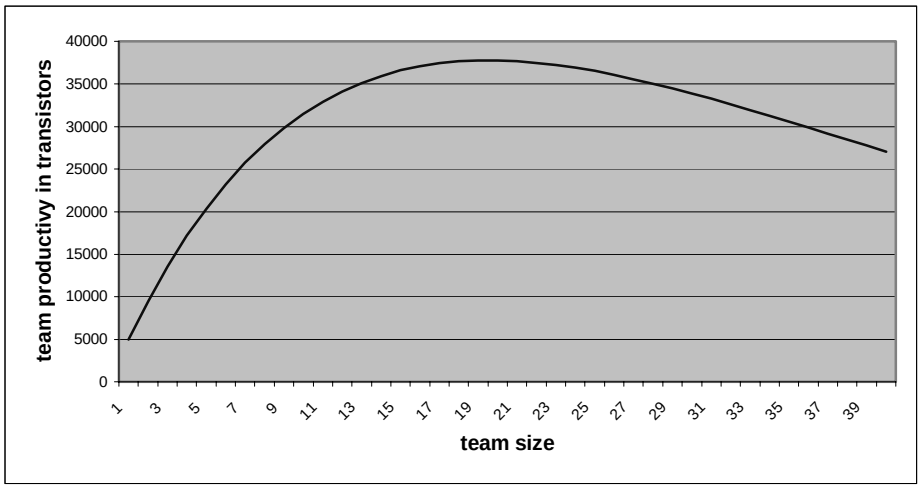
The "mythical man-month" refers to the decrease in productivity as designers are added to a project. This is due to the fact that as team sizes increase the amount of time needed for organization and communication start to exceed the time a designer is able to get actual work accomplished.

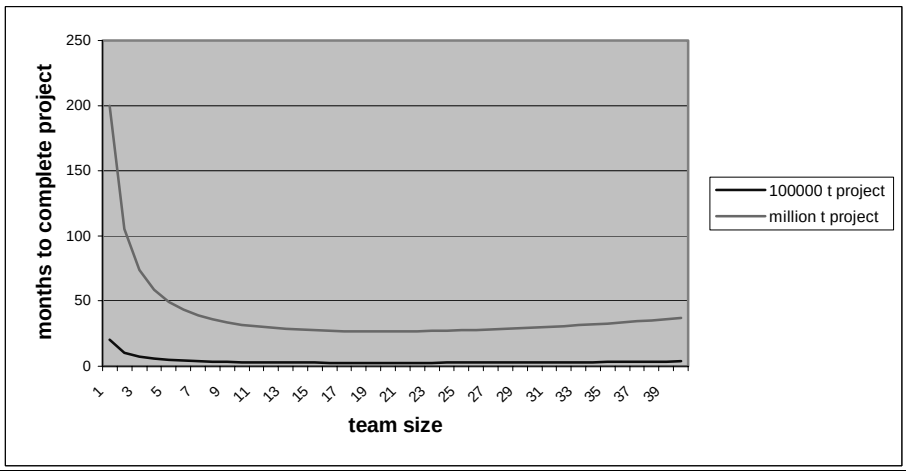
1.28 Assume a designer's productivity when working alone on a project is 5,000 transistors per month. Assume that each additional designer reduces productivity by 5%. (And keep in mind this is an extremely simplified model of designer productivity!) (a) Plot team monthly productivity versus team size for team sizes ranging from 1 to 40 designers. (b) Plot on the same graph the project completion time versus team size for projects of sizes 100,000 and 1,000,000 transistors. (c) Provide the "optimal" number of designers for each of the two projects, indicating the number of months required in each case.

team size	transistors per designer per month	team monthly productivity	completion of 100,000	completion of 1,000,000
1	5000	5000	20	200
2	4750	9500	10.52631579	105.2631579
3	4512.5	13537.5	7.386888273	73.86888273
4	4286.875	17147.5	5.8317539	58.317539
5	4072.53125	20362.65625	4.910950653	49.10950653
6	3868.904688	23213.42813	4.30785145	43.0785145
7	3675.459453	25728.21617	3.886783263	38.86783263
8	3491.68648	27933.49184	3.579931953	35.79931953
9	3317.102156	29853.91941	3.349643932	33.49643932
10	3151.247049	31512.47049	3.173346883	31.73346883
11	2993.684696	32930.53166	3.036695582	30.36695582
12	2844.000461	34128.00554	2.93014486	29.3014486
13	2701.800438	35123.4057	2.847104317	28.47104317
14	2566.710416	35933.94583	2.782883919	27.82883919
15	2438.374896	36575.62343	2.734061394	27.34061394
16	2316.456151	37063.29841	2.698086902	26.98086902
17	2200.633343	37410.76684	2.673027272	26.73027272
18	2090.601676	37630.83017	2.657395533	26.57395533
19	1986.071592	37735.36025	2.650034327	26.50034327
20	1886.768013	37735.36025	2.650034327	26.50034327
21	1792.429612	37641.02185	2.656676017	26.56676017
22	1702.808131	37461.77889	2.669387385	26.69387385
23	1617.667725	37206.35767	2.687712699	26.87712699
24	1536.784339	36882.82413	2.711289126	27.11289126
25	1459.945122	36498.62804	2.739829012	27.39829012
26	1386.947866	36060.64451	2.773106287	27.73106287
27	1317.600472	35575.21275	2.810945944	28.10945944

28	1251.720449	35048.17256	2.853215808	28.53215808
29	1189.134426	34484.89836	2.899820059	28.99820059
30	1129.677705	33890.33115	2.950694095	29.50694095
31	1073.19382	33269.00841	3.005800436	30.05800436
32	1019.534129	32625.09212	3.065125445	30.65125445
33	968.5574223	31962.39494	3.12867669	31.2867669
34	920.1295512	31284.40474	3.196480829	31.96480829
35	874.1230736	30594.30758	3.2685819	32.685819
36	830.4169199	29895.00912	3.345039957	33.45039957
37	788.8960739	29189.15474	3.42592997	34.2592997
38	749.4512702	28479.14827	3.511340966	35.11340966
39	711.9787067	27767.16956	3.60137535	36.0137535
40	676.3797714	27055.19086	3.696148385	36.96148385

(a), (b)





(c)

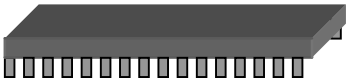
100,000 transistors = 19 or 20 members = 2.65 months

However, 19 is the correct answer because you will be paying one less designer for the same productivity and product completion time.

1,000,000 transistors = 19 or 20 members = 26.5 months

Again, for the same reason as above, 19 is the correct answer.

CHAPTER 2: *Custom Single-Purpose Processors: Hardware*



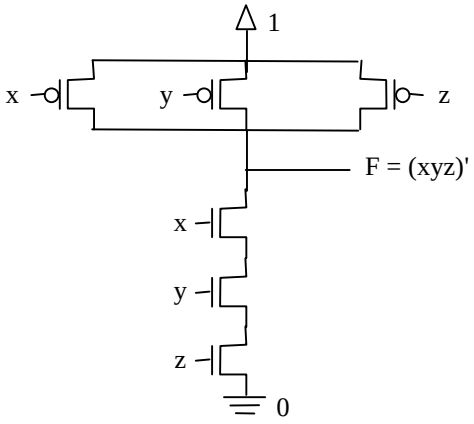
2.1 What is a single-purpose processor? What are the benefits of choosing a single-purpose processor over a general-purpose processor?

A single-purpose processor is designed specifically to carry out a particular computational task. The benefits of using a single-purpose processor is that the performance will be faster, the size may be smaller, and the power consumption may be less.

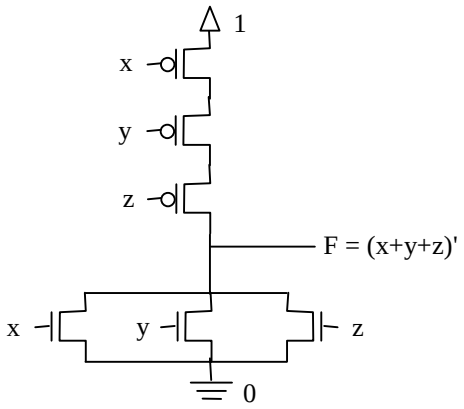
2.2 How do nMOS and pMOS transistors differ?

A nMOS transitor conducts if the gate is high (i.e. 1). A pMOS transistor conducts if the gate is low (i.e. 0).

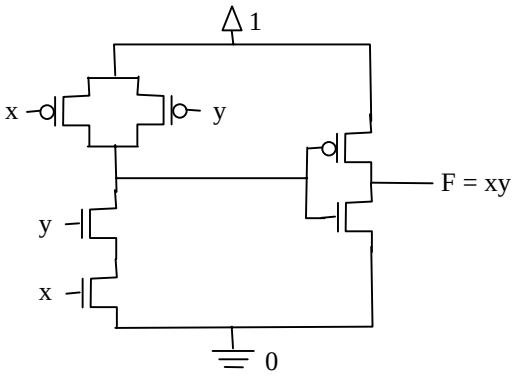
2.3 Build a 3-input NAND gate using a minimum number of CMOS transistors.



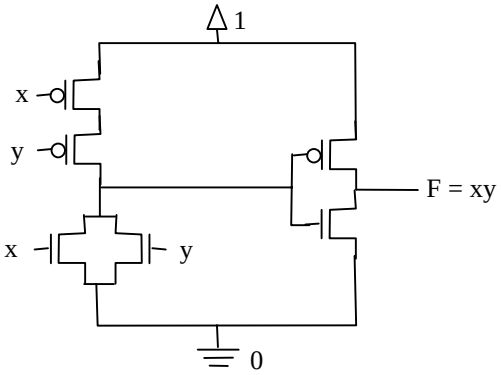
2.4 Build a 3-input NOR gate using a minimum number of CMOS transistors.



2.5 Build a 2-input AND gate using a minimum number of CMOS transistors.



2.6 Build a 2-input OR gate using a minimum number of CMOS transistors.



2.7 Explain why NAND and NOR gates are more common than AND and OR gates.

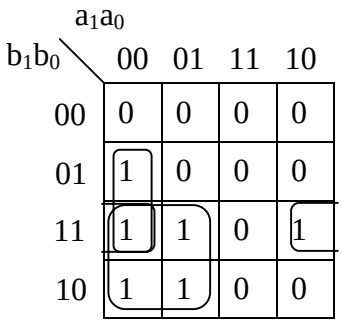
Because pMOS transistors don't conduct zeros very well and nMOS transistors don't conduct ones very well, AND and OR gates are designed by using NAND and NOR gates, respectively, along with an invertor. Thus, NAND and NOR gates are more common due to their reduced size.

2.8 Distinguish between a combinational circuit and a sequential circuit.

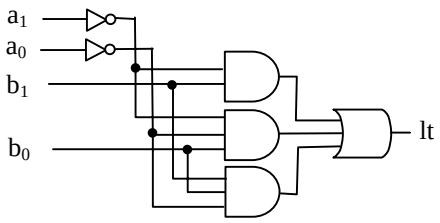
A combinational circuit is a digital circuit whose output is purely a function of its present inputs. A sequential circuit is a digital circuit whose outputs are a function of the present as well as the previous input values.

2.9 Design a 2-bit comparator (compares two 2-bit words) with a single output "less-than," using the combinational design technique described in the chapter. Start from a truth table, use K-maps to minimize logic, and draw the final circuit.

a1	a0	b1	b0	lt (a < b)
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	0
0	1	1	0	1
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	0
1	1	1	0	0
1	1	1	1	0



$lt = b_1a_1' + b_0a_1'a_0' + b_1b_0a_0'$

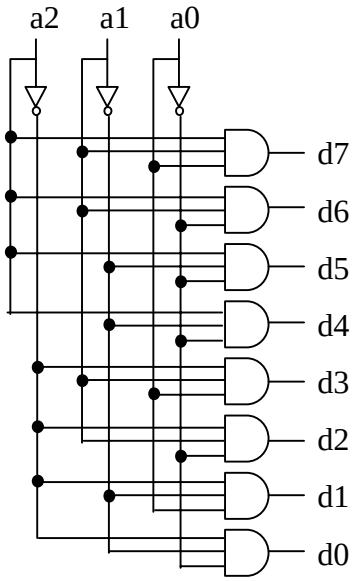


2.10 Design a 3×8 decoder. Start from a truth table, use K-maps to minimize logic and draw the final circuit.

a2	a1	a0	d7	d6	d5	d4	d3	d2	d1	d0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

K-maps are not essential for determining the log for each output since only a single one appears in each table

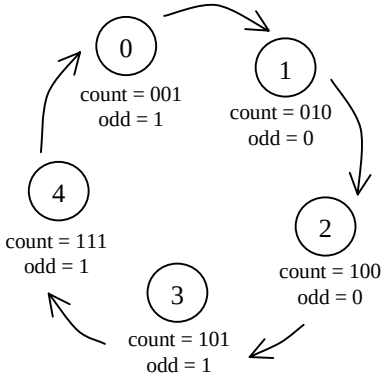
d7 = a2a1a0
d6 = a2a1a0'
d5 = a2a1'a0
d4 = a2a1'a0'
d3 = a2'a1a0
d2 = a2'a1a0'
d1 = a2'a1'a0
d0 = a2'a1'a0'

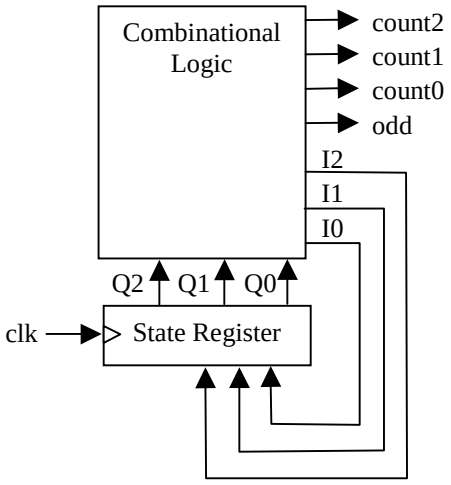


2.11 Describe what is meant by edge-triggered and explain why it is used.

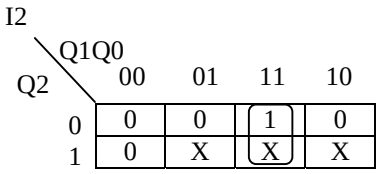
Edge-triggered describes logic that examines their non-clock triggered input when the clock signal rises from zero to one, or alternatively when the clock signal falls from one to zero. This is used because it prevents unexpected behavior from signal glitches.

2.12 Design a 3-bit counter that counts the following sequence: 1, 2, 4, 5, 7, 1, 2, etc. This counter has an output "odd" whose value is 1 when the current count value is odd. Use the sequential design technique of the chapter. Start from a state diagram, draw the state table, minimize the logic, and draw the final circuit.

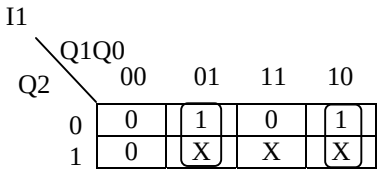




Inputs						Outputs			
Q2	Q1	Q0	I2	I1	I0	count2	count1	count0	add
0	0	0	0	0	1	0	0	1	1
0	0	1	0	1	0	0	1	0	0
0	1	0	0	1	1	1	0	0	0
0	1	1	1	0	0	1	0	1	1
1	0	0	0	0	0	1	1	1	1
1	0	1	x	x	x	x	x	x	x
1	1	0	x	x	x	x	x	x	x
1	1	1	x	x	x	x	x	x	x



I2 = Q1Q0



I1 = Q1'Q0 + Q1Q0'

I0

Q2 \ Q1Q0	00	01	11	10
0	1	0	0	1
1	0	X	X	X

I0 = Q2'Q0'

count2

Q2 \ Q1Q0	00	01	11	10
0	0	0	1	1
1	1	X	X	X

count2 = Q2 + Q1

count1

Q2 \ Q1Q0	00	01	11	10
0	0	1	0	0
1	1	X	X	X

count1 = Q2 + Q1'Q0

count0

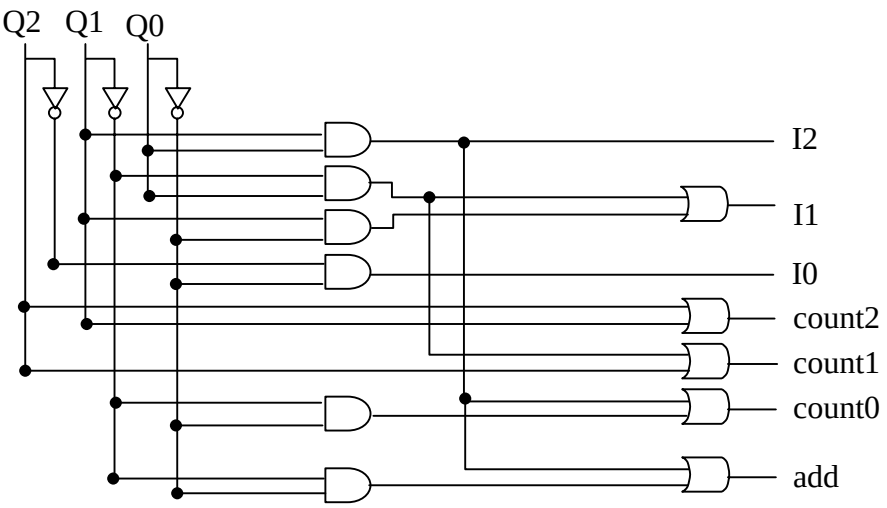
Q2 \ Q1Q0	00	01	11	10
0	1	0	1	0
1	1	X	X	X

count0 = Q1'Q0' + Q1Q0

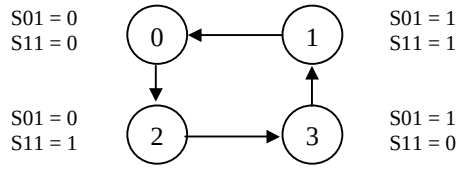
add

Q2 \ Q1Q0	00	01	11	10
0	1	0	1	0
1	1	X	X	X

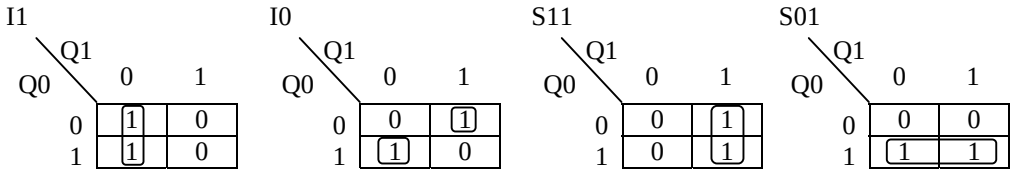
add = Q1'Q0' + Q1Q0



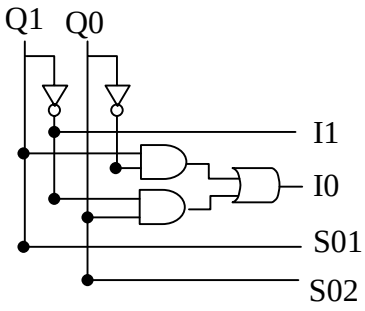
2.13 Four lights are connected to a decoder. Build a circuit that will blink the lights in the following order: 0, 2, 1, 3, 0, 2, ... Start from a state diagram, draw the state table, minimize the logic, and draw the final circuit.



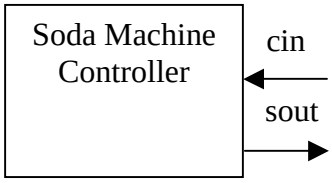
Q1	Q0	I1	I0	S11	S01
0	0	1	0	0	0
0	1	1	1	0	1
1	0	0	1	1	1
1	1	0	0	1	1

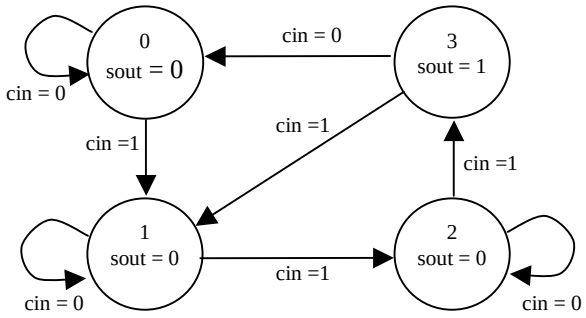


I1 = Q1' I0 = Q1Q0' + Q1'Q0 S11 = Q1 S01 = Q0



2.14 Design a soda machine controller, given that a soda costs 75 cents and your machine accepts quarters only. Draw a black-box view, come up with a state diagram and state table, minimize the logic, and then draw the final circuit.





I1

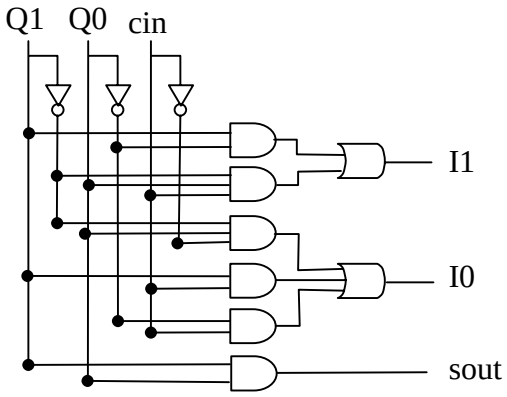
cin \ Q1Q0	00	01	11	10
0	0	0	0	1
1	0	1	0	1

I1 = Q1Q0' + Q1'Q0Cin

I0

cin \ Q1Q0	00	01	11	10
0	0	1	0	0
1	1	0	1	1

I0 = Q1'Q0Cin' + Q1Cin + Q0'Cin



sout

cin \ Q1Q0	00	01	11	10
0	0	0	1	0
1	0	0	1	0

sout = Q1Q0

2.15 What is the difference between a synchronous and an asynchronous circuit?

A synchronous circuit's input value only has an effect during a clock cycle edge. An asynchronous circuit's input value affect the circuit independent of the clock.

2.16 Determine whether the following are synchronous or asynchronous: (a) multiplexor, (b) register, (c) decoder.

(a) asynchronous
(b) synchronous
(c) asynchronous

2.17 What is the purpose of the datapath? of the controller?

The datapath stores and manipulates a systm's data. The controller sets the datapath control inputs and monitors external control inputs as well as datapath control outputs.

2.18 Compare the GCD custom-processor implementation to a software implementation (a) Compare the performance. Assume a 100-ns clock for the microcontroller, and a 20-ns clock for the custom processor. Assume the microcontroller uses two operand

instructions, and each instruction requires four clock cycles. Estimates for the microcontroller are fine. (b) Estimate the number of gates for the custom design, and compare this to 10,000 gates for a simple 8-bit microcontroller. (c) Compare the custom GCD with the GCD running on a 300-MHz processor with 2-operand instructions and one clock cycle per instruction (advanced processors use parallelism to meet or exceed one cycle per instruction). (d) Compare the estimated gates with 200,000 gates, a typical number of gates for a modern 32-bit processor.

(a) possible GCD implementation on a microprocessor:

```
mainloop:  ld go, go_i
           bz go, mainloop
           ld x, z_i
           ld y, y_i
gcdloop:   be x, y, done
           blt x, y, if
           sub x, x, y
           jmp endif
if:        sub y, y, x
endif:     jmp gcdloop
done:      st d_o, x
           jmp mainloop
```

Example execution when x=39 and y=27

clock cycle	x	y
0	39	27
1	12	27
2	12	15
3	12	3
4	9	3
5	6	3
6	3	3

microcontroller: 32 instructions = 32 * 4 cycles/instr * 100 ns = 12,800 ns
custom GCD: 38 instrucions = 38 * 20 ns = 760 ns

(b) Assume 4-bit words

2 * 4-bit 2x1 mux = 2 * 16 gates = 32 gates
3 * 4-bit register = 3 * 24 gates = 72 gates
2 * 4-bit subtractor = 2 * 29 gates = 58 gates
< comparator = 15 gates
!= comparator = 24 gates
1 4-bit state register = 24 gates
FSM combinational logic = est 45 gates

Total = 252 gates

(c) GCD on 300-MHZ processor

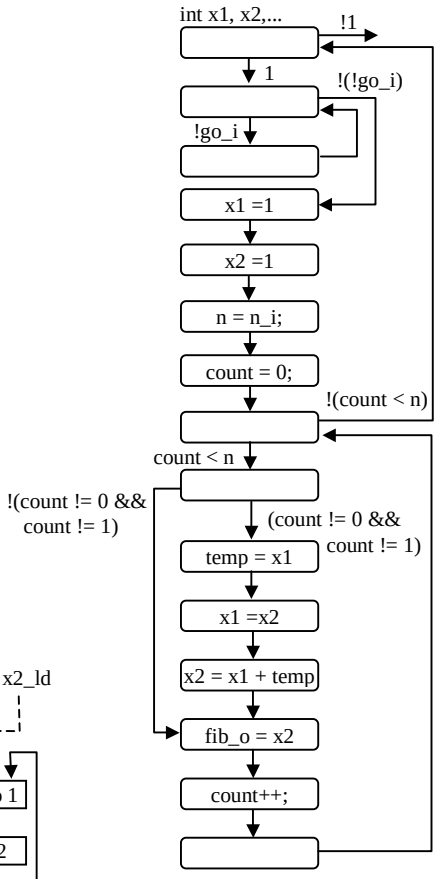
38 instruction => 38 * 1 cycle/second * 3.33 ns/cycle = 126.54 ns

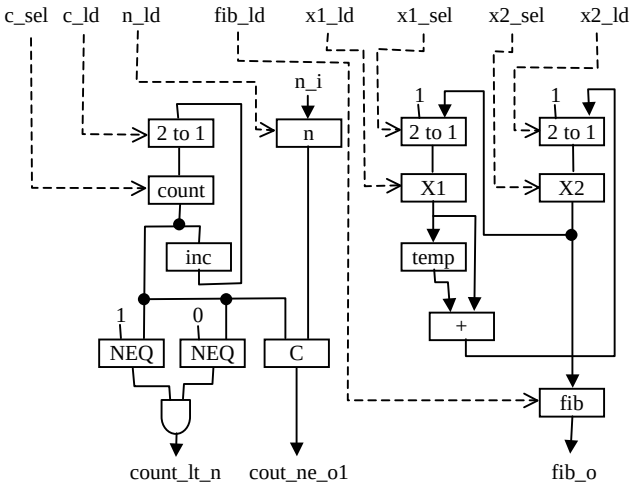
(d) compare estimated gates with 200,000

Estimated gates are 252 which is quite a bit smaller than the 200,000 gate processor.

2.19 Design a single-purpose processor that outputs Fibonacci numbers up to *n* places. Start with a function computing the desired result, translate it into a state diagram, and sketch a probable datapath.

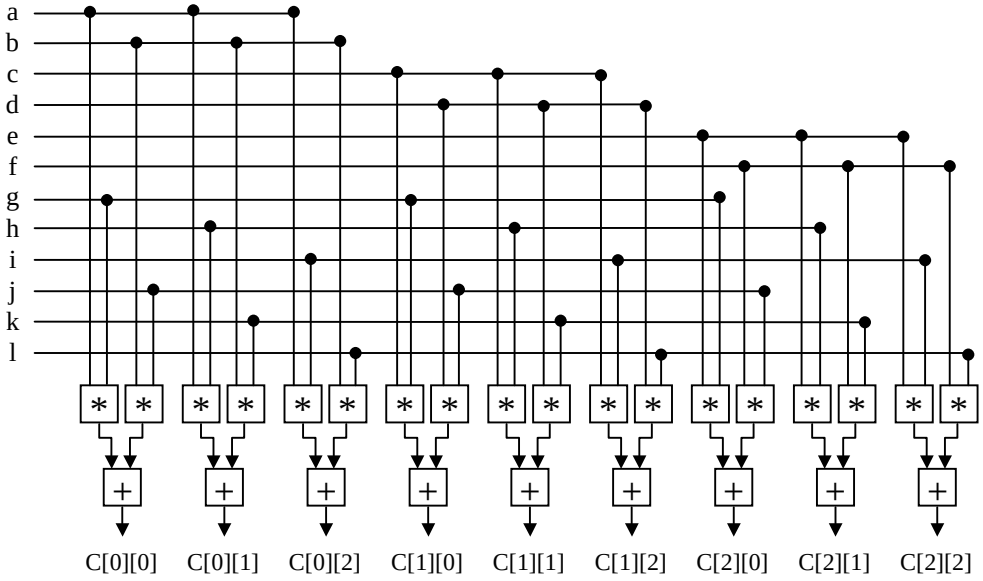
```
int x1, x2, temp, count, n;
while(1){
  while(!go_i);
  x1 = 1;
  x2 = 1;
  n = n_i;
  count = 0;
  while(count < n){
    if(count != 0 && count !=1){
      temp = x1;
      x1 = x2;
      x2 = x1 + temp;
    }
    fib_o = x2;
    count ++;
  }
}
```





2.20 Design a circuit that does the matrix multiplication of matrices *A* and *B*. Matrix *A* is 3×2 and matrix *B* is 2×3. The multiplication works as follows:

$$\underset{A}{\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}} \bullet \underset{B}{\begin{bmatrix} g & h & i \\ j & k & l \end{bmatrix}} = \underset{C}{\begin{bmatrix} a*g + b*j & a*h + b*k & a*i + b*l \\ c*g + d*j & c*h + d*k & c*i + d*l \\ e*g + f*j & e*h + f*k & e*i + f*l \end{bmatrix}}$$

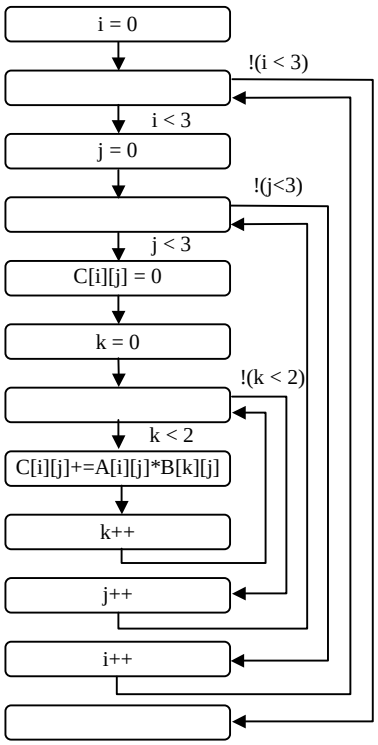


2.21 An algorithm for matrix multiplication, assuming that we have one adder and one multiplier, follows. (a) Convert the matrix multiplication algorithm into a state diagram using the template provided in Fig1.10. (b) Rewrite the matrix multiplication algorithm given the assumption that we have three adders and six multipliers. (c) If each multiplication takes two cycles to compute and each addition takes one cycle compute, how many cycles does it take to complete the matrix multiplication given one adder and one multiplier? Three adders and six multipliers? Nine adders and 18 multipliers? (d) If each an adder requires 10 transistors to implement and each multiplier requires 100 transistors to implement, what is the total number of transistor needed to implement the matrix multiplication circuit using one adder and one multiplier? Three adders and six multipliers? Nine adders and 18 multipliers? (e) Plot your results from parts (c) and (d) into a graph with latency along the *x*-axis and size along the *y*-axis.

```
main( ) {
        int A[3][2] = { {1, 2}, {3,4}, {5,6} };
        int B[2][3] = { {7, 8, 9}, {10, 11, 12} };
        int C[3][3];
        int i, j, k;

for (i=0; i < 3; i++){
        for ( j=0; j < 3; j++){
                C[i][j]=0;
                for (k=0; k < 2; k++){
                        C[i][j] += A[i][k] * B[k][j];
                }
        }
}
```

(a) state diagram for matrix multiply



(b) rewrite using 3 adders and 6 multipliers

```
main( ) {
   int A[3][2] = { {1, 2}, {3,4}, {5,6} };
   int B[2][3] = { {7, 8, 9}, {10, 11, 12} };
   int C[3][3];
   int i, k;

for (i=0; i < 3; i++){
   C[i][0]= A[i][0]*B[0][0]+A[i][1]*B[1][0];
   C[i][1]= A[i][0]*B[0][1]+A[i][1]*B[1][1];
   C[i][2]= A[i][0]*B[0][2]+A[i][1]*B[1][2];
}
```

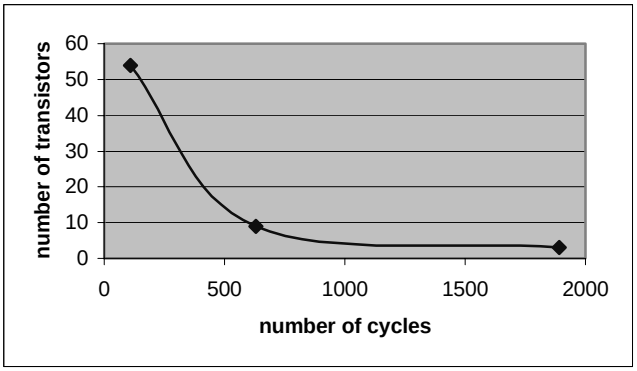
(c) cycles to complete matrix multiply

1 adder + 1 multiplier = 54 cycles
3 adders + 6 multipliers = 9 cycles
9 adders + 18 multipliers = 3 cycles

(d) number of transistors

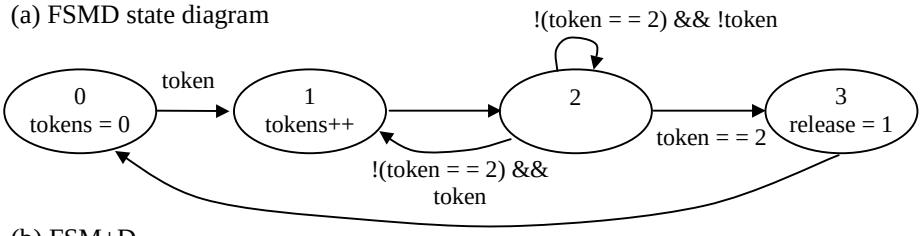
1 adder + 1 multiplier = 110 transistors
3 adders + 6 multipliers = 630 transistors
9 adders + 18 multipliers = 1890 transistors

(e) plot cycles verses number of transistors

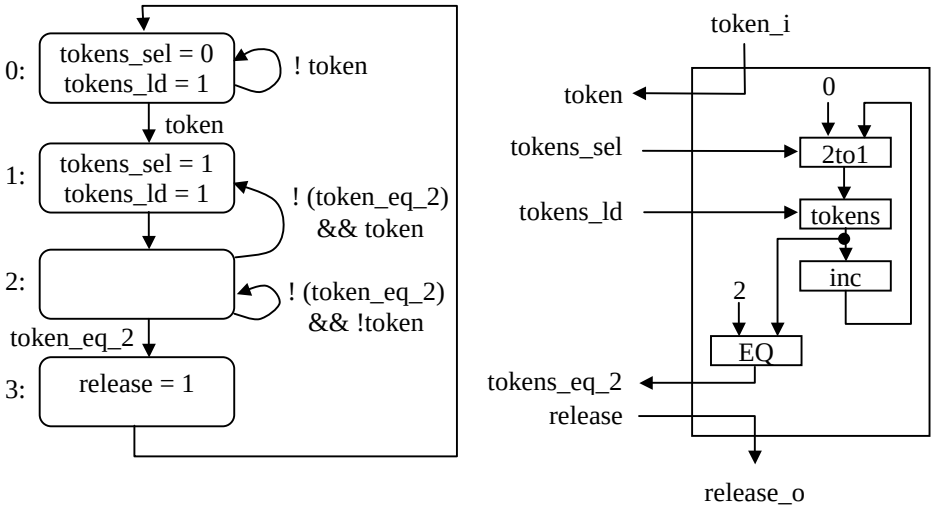


2.22 A subway has an embedded system controlling the turnstile, which releases when two tokens are deposited. (a) Draw the FSMD state diagram for this system. (b) Separate the FSMD into an FSM+D. (c) Derive the FSM logic using truth tables and K-maps to minimize logic. (d) Draw your FSM and datapath connections.

(a) FSMD state diagram

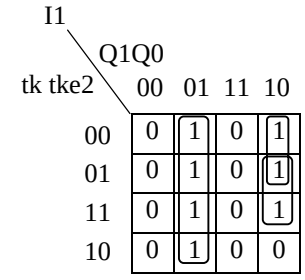


(b) FSM+D

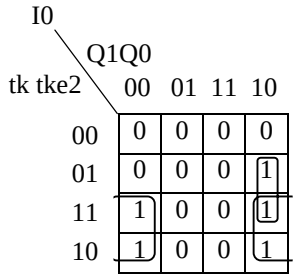


(c) FSM logic using truth tables and K-maps to minimize logic

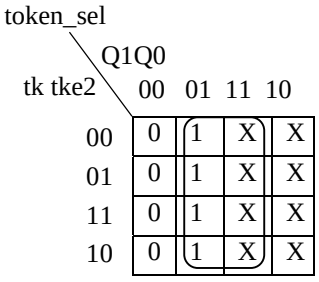
Q0	Q1	token	tokens_eq_2	I1	I0	tokens_sel	tokens_ld	release
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	1	0
0	0	1	0	0	1	0	1	0
0	0	1	1	0	1	0	1	0
0	1	0	0	1	0	1	1	0
0	1	0	1	1	0	1	1	0
0	1	1	0	1	0	1	1	0
0	1	1	1	1	0	1	1	0
1	0	0	0	1	0	X	0	0
1	0	0	1	1	1	X	0	0
1	0	1	0	0	1	X	0	0
1	0	1	1	1	1	X	0	0
1	1	0	0	0	0	X	0	1
1	1	0	1	0	0	X	0	1
1	1	1	0	0	0	X	0	1
1	1	1	1	0	0	X	0	1



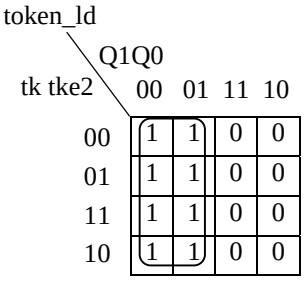
I1 = Q1'Q0 + Q1Q0'token_eq_2 + Q1Q0'token'



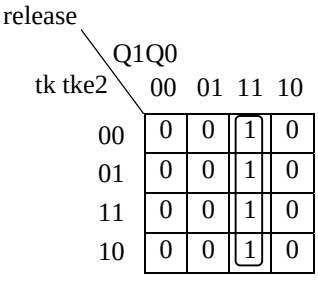
I0 = Q0'token + Q1Q0'token_eq_2



token_sel = Q0

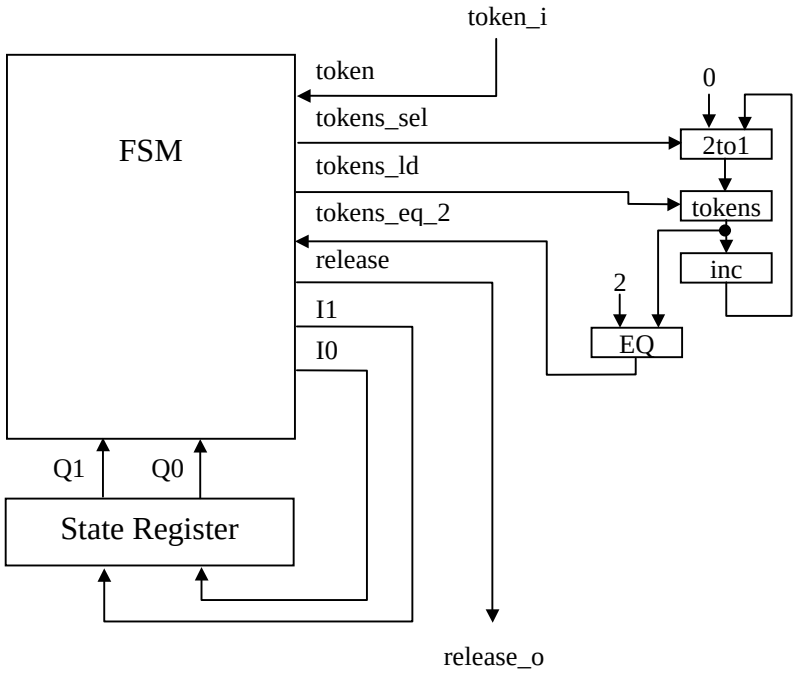


token_ld = Q1'

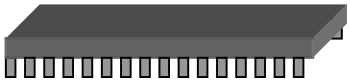


release = Q1Q0

(d) FSM and datapath connections



CHAPTER 3: *General-Purpose Processors: Software*



3.1 Describe why a general-purpose processor could cost less than a single-purpose processor you design yourself.

Processor manufactures can spread NRE cost for the processor's design over the large number of units sold, often in the millions or billions.

3.2 Detail the stages of executing the MOV instructions of Figure 3.7, assuming an 8-bit processor and a 16-bit IR and program memory following the model of Figure 3.1. For example, the stages for the ADD instruction are (1) fetch M[PC] into IR, (2) read Rn and Rm from register file through ALU configured for ADD, storing results back in Rn.

MOV Rn, direct
1. fetch M[PC] into IR
2. move contents of register Rn into M[direct]

MOV direct, Rn
1. fetch M[PC] into IR
2. move contents of register Rn into M[direct]

MOV @Rn, Rm
1. fetch M[PC] into IR
2. move contents of register Rn into M[Rm]

MOV Rn, #imm
1. fetch M[PC] into IR
2. move immediate value into register Rn

3.3 Add one instruction to the instruction set of Figure 3.7 that would reduce the size of our summing assembly program by 1 instruction. *Hint*: add a new branch instruction. Show the reduced program.

```
// add instruction JNZ Rn, relative
0:      MOV R0, #0
1:      MOV R1, #10
2:      MOV R2, #1
Loop:   ADD R0, R1
4:      SUB R1, R2
5:      JNZ R1, Loop
```

Note: Answers may vary.

3.4 Create a table listing the address spaces for the following address sizes: (a) 8-bit, (b) 16-bit, (c) 24-bit, (d) 32-bit, (e) 64-bit.

Address Size (bits)	Address Space
8	2^8-1 = 255
16	2^{16}-1 = 65,535
24	2^{24}-1 = 16,777,215
32	2^{32}-1 = 4,294,967,295
64	2^{64}-1 ≈ 1.84 x 10^{19}

3.5 Illustrate how program and data memory fetches can be overlapped in a Harvard architecture.

Suppose we have the following 2 instructions in a pipelined machine:
MOV R1, #10
ADD R3, R4

Cycle 1: Fetch PM[PC] into IR where PC=0

Cycle 2: Fetch DM[10] and store in R1
Fetch PM[PC] into IR where PC=1

In cycle 2, we are simultaneously fetching from program and data memory.

Note: Answers will vary.

3.6 Read the entire problem before beginning. (a) Write a C program that clears an array "short int M[256]." In other words, the program sets every location to 0. *Hint*: your program should only be a couple lines long. (b) Assuming M starts at location 256 (and thus ends at location 511), write the same program in assembly language using the earlier instruction set. (c) Measure the time it takes you to perform parts a and b, and report those times.

(a) C program

```
for ( i=0; i<256; i++){
        M[i] = 0;
}
```

(b) Assembly program, assume M begins at location 256

```
        MOV R1, #256    // i = 256 start location of M
        MOV R2, #1      // R2 = constant value of 1
        MOV R3, #256    // R3 = constant value of 256
        MOV R4, #0      // R4 = constant value of 0

Loop:   MOV @R1, R4     // M[R1]=0
        ADD R1, R2      // R1 ++
        SUB R3, R2      // R3 --
        JNZ R3, Loop    // if counter at zero, all locations initialized
```

(c) time to perform part(a) and part(b)

Note: Answers will vary.

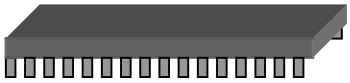
3.7 Acquire a databook for a microcontroller. List the features of the basic version of that microcontroller, including key characteristics of the instruction set (number of instructions of each type, length per instruction, etc.), memory architecture and available memory, general-purpose registers, special-function registers, I/O facilities, interrupt facilities, and other salient features.

Note: Answers will vary.

3.8 For the microcontroller in the previous exercise, create a table listing five existing variations of that microcontroller, stressing the features that differ from the basic version.

Note: Answers will vary.

CHAPTER 4: *Standard Single-Purpose Processors: Peripherals*



4.1 Given a timer structured as in Figure 4.1 (c) and a clock frequency of 10 MHz: (a) Determine its range and resolution. (b) Calculate the terminal count value needed to measure 3 ms intervals. (c) If a prescaler is added, what is the minimum division needed to measure an interval of 100 ms? (Divisions should be in powers of 2.) Determine this design's range and resolution. (d) If instead of a prescaler a second 16-bit up-counter is cascaded as in Figure 4.1 (d), what is the range and resolution of this design?

(a) resolution = period = 1 / frequency = 1 / (10 MHz) = 1e-7 s
range = 2^16 * resolution = 65536 * 1e-7s = .0065536 s
= 0 to 6.5536 ms

(b) terminal count value = desired time interval / clock period
3e-3 s / 1e-7 s = 30,000

(c) The prescaler should be set to output a frequency 1/(2^4) or 1/16 the original frequency of 10 MHz.

resolution = 16 * original resolution = 16 * 1e-7 s = 1.6e-6 s
range = 16 * original range = 16 * 6.5536 ms
= 0 to 104.8576 ms

(d) resolution = 1e-7 s (does not change)
range = 2^32 * resolution = 0 to 429.4967296 s

4.2 A watchdog timer that uses two cascaded 16-bit up-counters as in Figure 4.1 (d) is connected to an 11.981 MHz oscillator. A timeout should occur if the function *watchdog_reset* is not called within 5 minutes. What value should be loaded into the up-counter pair when the function is called?

period = 1 / (11.981 MHz) = 8.346548702e-8 s
range = period * 2^32 = 8.346548702e-8 s * 2^32 = 358.4815371 s
"time-out" = 5 min. * (60 s / 1 min.) = 300 s
X = range – "time-out" = 358.4815371 s - 300 s = 58.4815371 s
value loaded = X / period = 58.4815371 s / 8.346548702e-8 s = 700667296

4.3 Given a controller with two built-in timers designed as in Figure 4.1 (b), write C code for a function "double *RPM*" that returns the revolutions per minute of some device or –1 if a timer overflows. Assume all inputs to the timers have been initialized and the timers have been started before entering *RPM*. *Timer*1*'s cnt_in* is connected to the device and is pulsed once for each revolution. *Timer*2*'s clk* input is connected to a 10 MHz oscillator. The timers have the outputs *cnt*1, *cnt*2, *top*1, and *top*2, which were initialized to 0 when their respective timer began. What is the minimum (other than 0) and maximum revolutions per minute that can be measured if *top* is not used?

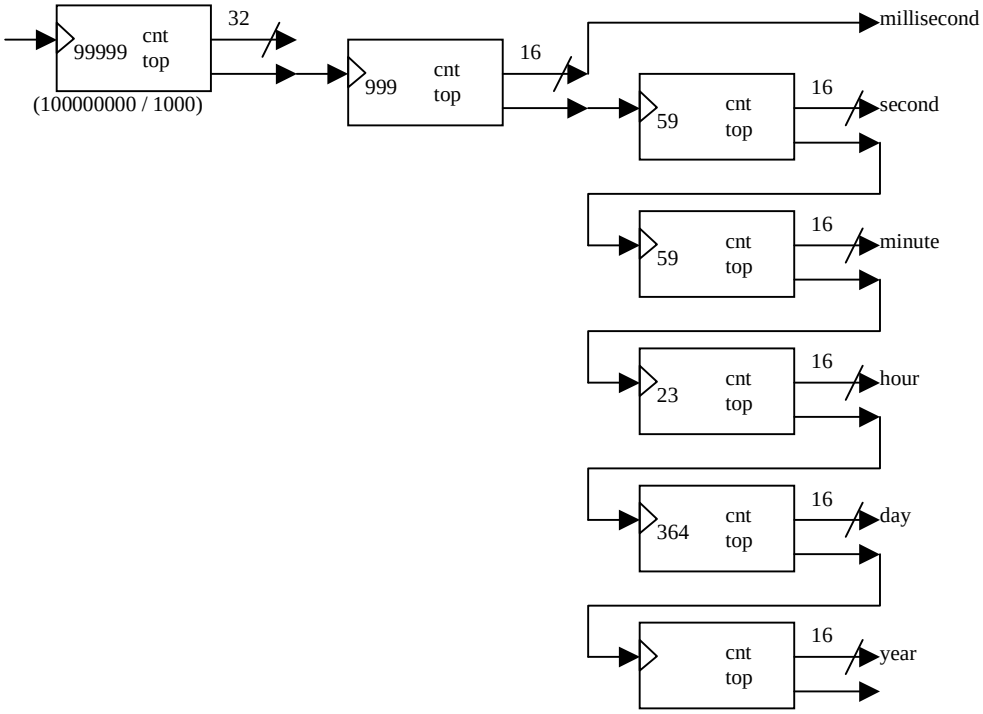
```
double RPM(){
      int rev, ticks = 0;
      double time = 0;
      rev = cnt1;
      ticks = cnt2;

      // when top1 or top2 are 1 than a timer has overflowed
      if( top1 == 0 && top2 == 0){
            // ticks of clock * period gives seconds
            time = ticks * (1/10000000);

            // convert seconds to minutes
            time = time * (1/60);
            return (rev/time);
      }
      return( -1 );
}
```

resolution = period = 1 / frequency = 1 / (10 MHz) = 1e-7 s * (1 min. / 60 s)
= 1.6667e-7 min.
max range = 2^16 * resolution = 65536 * 1e-7s
= .0065536 s * (1 min. / 60 s)
= 1.092267e-4 min.
min rpm = 1 rev / max range = 1 / 1.092267e-4 min. = 9155.273437
max rpm = max revs / (1 resolution) = 2^16 revs / 1.6667e-7 min.
= 3.932081358e11

4.4 Given a 100 MHz crystal-controlled oscillator and a 32-bit and any number of 16-bit terminal-count timers, design a real-time clock that outputs the date and time down to milliseconds. You can ignore leap years. Draw a diagram and indicate terminal-count values for all timers.



4.5 Determine the values for *smod* and *TH*1 to generate a baud rate of 9,600 for the 8051 baud rate equation in the chapter, assuming an 11.981 MHz oscillator. Remember that *smod* is 2 bits and *TH*1 is 8 bits. There is more than one correct answer.

Baudrate = (2^smod / 32) * oscfreq / (12 * (256 – TH1))

9600 = (2^3 / 32) * 11,981,000 / (12 * (256 – TH1))
9600 = 2995250 / (3072 – 12 * TH1)
29491200 – 115200 * TH1 = 2995250
26495950 = 115200 * TH1
TH1 = 230 = 11100110
smod = 3 = 11

-or-

9600 = (2^2 / 32) * 11,981,000 / (12 * (256 – TH1))
9600 = 1497625 / (3072 – 12 * TH1)
29491200 – 115200 * TH1 = 1497625
27993575 = 115200 * TH1
TH1 = 243 = 11110011
smod = 2 = 10

4.6 A particular motor operates at 10 revolutions per second when its controlling input voltage is 3.7 V. Assume that you are using a microcontroller with a PWM whose output port can be set high (5 V) or low (0 V). (a) Compute the duty cycle necessary to obtain 10 revolutions per second. (b) Provide values for a pulse width and period that achieve this duty cycle. You do not need to consider whether the frequency is too high or too low although the values should be reasonable. There is no one correct answer.

(a) 3.7V / 5V = .74 = 74% duty cycle

(b) There are infinitely many answers.

Example:
period = 100 ns (pick any reasonable value)
pulse width = .74 * period = .74 * 100 ns = 74 ns

4.7 Using the PWM described in Figure 4.6 compute the value assigned to PWM1 to achieve an RPM of 8,050 assuming the input voltage needed is 4.375 V.

4.375 / 5 = .875 = 87.5 % duty cycle
.875 * 254 (counter reset value) = 222
PWM1 = 222 = CDh

4.8 Write a function in pseudocode that initializes the LCD described in Figure 4.7. After initialization, the display should be clear with a blinking cursor. The initialization should set the following data to shift to the left, have a data length of 8-bits and a font of 5 × 10 dots, and be displayed on one line.

```
LCDinit(){
        RS = 0;  // indicate control words
        CONTROL_WORD = 0x01;  // clear display, return cursor home
        EnableLCD(45);  // toggle enable and delay
        CONTROL_WORD = 0x07;  // shift left (I/D=1, S=1) (00000111)
        EnableLCD(45); // toggle enable and delay
        CONTROL_WORD = 0x0F;  // display on, cursor on and blinking
                                 // (00001111)
        EnableLCD(45); // toggle enable and delay
```

```
        CONTROL_WORK = 0x1B;  // 8-bit, one line, and 5x10 font
                                    // (00011011)
        EnableLCD(45); // toggle enable and delay
}
```

4.9 Given a 120-step stepper motor with its own controller, write a C function *Rotate (int degrees)*, which, given the desired rotation amount in degrees (between 0 and 360), pulses a microcontroller's output port the correct number of times to achieve the desired rotation.

The motor takes 120 steps to rotate 360 degrees so the degrees per step = 360/120 = 3.

```
Rotate( int degrees ){
        const int DPS = 3    // degrees per step
        int i
        for( i = 0; i < (degrees/DPS); i++){
                outport = 0
                delay()
                outport = 1
        }
}
```

4.10 Modify only the *main* function in Figure 4.12 to cause a 240-step stepper motor to rotate forward 60 degrees followed by a backward rotation of 33 degrees. This stepper motor uses the same input sequence as the example for each step. In other words, do not change the lookup table.

```
// 360 / 240 = 1.5 degrees per step
while(1){
        //  60 degrees / 1.5 degrees per step = 40 steps
        move(1,40);  // 1 = forward
        //  33 / 1.5 = 22 steps
        move(0,22);  // 0 = backward
}
```

4.11 Extend the ratio and resolution equations of analog-to-digital conversion to any voltage range between V_{min} to V_{max} rather than 0 to V_{max}.

ratio: $\frac{(e - V_{min})}{(V_{max} - V_{min})} = \frac{d}{(2^n - 1)}$
resolution: $\frac{(V_{max} - V_{min})}{(2^n - 1)}$

4.12 Given an analog output signal whose voltage should range from 0 to 10 V, and an 8-bit digital encoding, provide the encodings for the following desired voltages: (a) 0 V, (b) 1 V, (c) 5.33 V, (d) 10 V, (e) What is the resolution of our conversion?

$n = 8$, $2^n - 1 = 255$
$V_{min} = 0$, $V_{max} = 10$

(a) 0V

0 / 10 = d / 255
d = 0
00000000

(b) 1V

1 / 10 = d / 255
d = 25.5 ≅ 25
00011001

(c) 5.33V

5.33 / 10 = d / 255
d = 135.9 ≅ 136
10001000

(d) 10V

10 / 10 = d / 255
d = 255
11111111

(e) What is the resolution of our conversion?

10 – 0 / 255 = 0.039V

4.13 Given an analog input signal whose voltage ranges from 0 to 5 V, and an 8-bit digital encoding, calculate the correct encoding for 3.5 V, and then trace the successive-approximation approach (i.e., list all the guessed encodings in the correct order) to find the correct encoding.

3.5 / 5 = d / 255
d = 178.5 ≅ 179
10110011

3.5 > ½(5 + 0)	10000000
3.5 < ½(5 + 2.5)	10000000
3.5 > ½(3.75 + 2.5)	10100000
3.5 > ½(3.75 + 3.125)	10110000
3.5 < ½(3.75 + 3.437)	10110000
3.5 < ½(3.594 + 3.437)	10110000
3.5 > ½(3.515 + 3.437)	10110010
3.5 > ½(3.515 + 3.476)	10110011

4.14 Given an analog input signal whose voltage ranges from –5 to 5 V, and a 8-bit digital encoding, calculate the correct encoding 1.2 V, and then trace the successive-approximation approach to find the correct encoding.

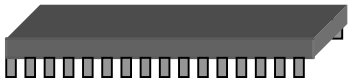
(1.2 – (-5)) / (5 – (-5)) = d / 255
.62 = d / 255
d = 158.1 ≅ 158
10011110

1.2 > ½(5 + -5)	10000000
1.2 < ½(5 + 0)	10000000
1.2 < ½(2.5 + 0)	10000000
1.2 > ½(1.25 + 0)	10010000
1.2 > ½(1.25 + 0.625)	10011000
1.2 > ½(1.25 + 0.9375)	10011100
1.2 > ½(1.25 + 1.09375)	10011110
1.2 < ½(1.25 + 1.171875)	10011110

4.15 Compute the memory needed in bytes to store a 4-bit digital encoding of a 3-second analog audio signal sampled every 10 milliseconds.

3 s * (1000 ms / 1s) = 3000 ms
3000ms / (10 ms / sample) = 300 samples
300 samples * 4 bits = 1200 bits * (1 byte / 8 bits) = 150 bytes

CHAPTER 5: *Memory*



5.1 Briefly define each of the following: mask-programmed ROM, PROM, EPROM, EEPROM, flash EEPROM, RAM, SRAM, DRAM, PSRAM, and NVRAM.

1. Mask-programmed ROM:
Mask-programmed ROM is the read only memory where the "programming" of its memory is done at fabrication time with masks. It has the lowest write ability and the highest storage permanence of the different types of memory.

2. PROM:
User programmable ROM is memory that can be written after fabrication but before normal operation. PROM's can be one-time-programmable, typically by blowing fuses, or erasable-programmable. In either case, the programming is accomplished with a device, not software.

3. EPROM:
Erasable PROM can be programmed by injecting electrons into "floating gates". They can be erased using ultra-violet light and then reprogrammed. EPROM's have higher write ability than mask-programmed ROM, but they have lower storage permanence.

4. EEPROM:
Electrically-erasable programmable ROM's are programmed and erased electronically, thus speeding up the process compared to the EPROM. Also, the EEPROM erases single words rather than the whole memory. Because of these features, EEPROM's can be written to by the embedded system during its operation, although at a much slower rate than a typical memory read. However, the written data would be kept even after the power is turned off. EEPROM's have better write ablility than the EPROM and their storage permanence is similar.

5. Flash EEPROM:
Flash EEPROM's are EEPROM's that are able to erase large blocks of memory rather than just a word at a time although they can be slower at writing single words.

6. RAM:

Random Access Memory is the traditional read-write memory. Reads and writes take approximately the same amount of time. RAM's do not have any data programmed into them before operation. All reads and writes are done during execution time.

7. SRAM:
Static RAM holds data for as long as there is power supplied to it. It is typically implemented on the IC using flip-flops to store bits.

8. DRAM:
Dynamic RAM is smaller than SRAM because it uses only a MOS transistor and a capacitor to store a bit. Because it uses a capacitor, though, it loses its charge eventually. Thus, DRAM must be "refreshed" periodically. DRAM's are typically implemented off the IC and are slower to access than SRAM's.

9. PSRAM:
Pseudo-static RAM is a DRAM that refreshes itself, thus simulating an SRAM while retaining the DRAM's compactness.

10. NVRAM:
Non-volatile RAM works much like ROM's in that its data is kept after the power is off. One type of NVRAM uses an internal battery while another actually stores its data on an EEPROM or flash memory before the power is turned off and then reloads it after the power is turned back on.

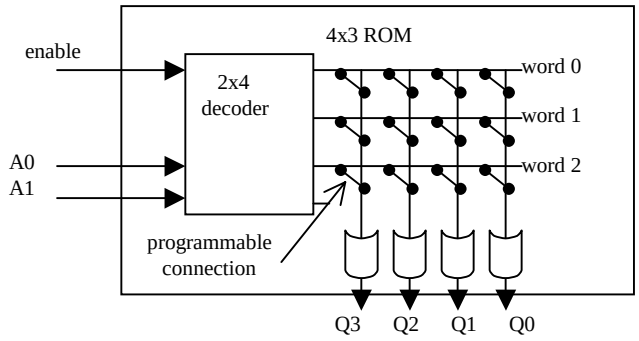
5.2 Define the two main characteristics of memories as discussed in this chapter. From the types of memory mentioned in Exercise 5.1, list the worst choice for each characteristic. Explain.

1. Write ability:
Write ability refers to the manner and speed that the memory can be written. Memories can be written by a processor, a "programmer", or only at fabrication time. Memories written by a processor vary in the speed to which they are written.

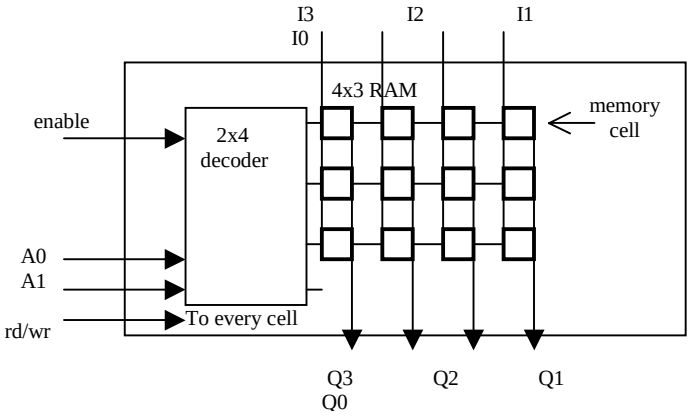
2. Storage performance:
Storage permanence refers to the ability to store the bits once they are written to the memory. The permanence ranges from near zero to essentially infinite.

Worst Choices:
The worst choice in terms of write ability is the mask-programmed ROM. This memory is written at fabrication time and can never be written again.
The worst choice in terms of storage permanence is the basic DRAM. This memory has to be "refreshed" often and loses all data when power is turned off.

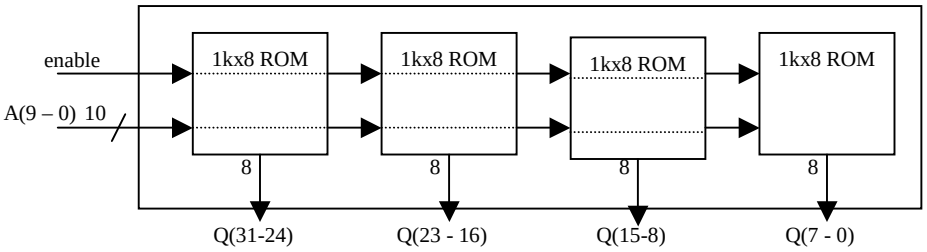
5.3 Sketch the internal design of a 4 × 3 ROM.



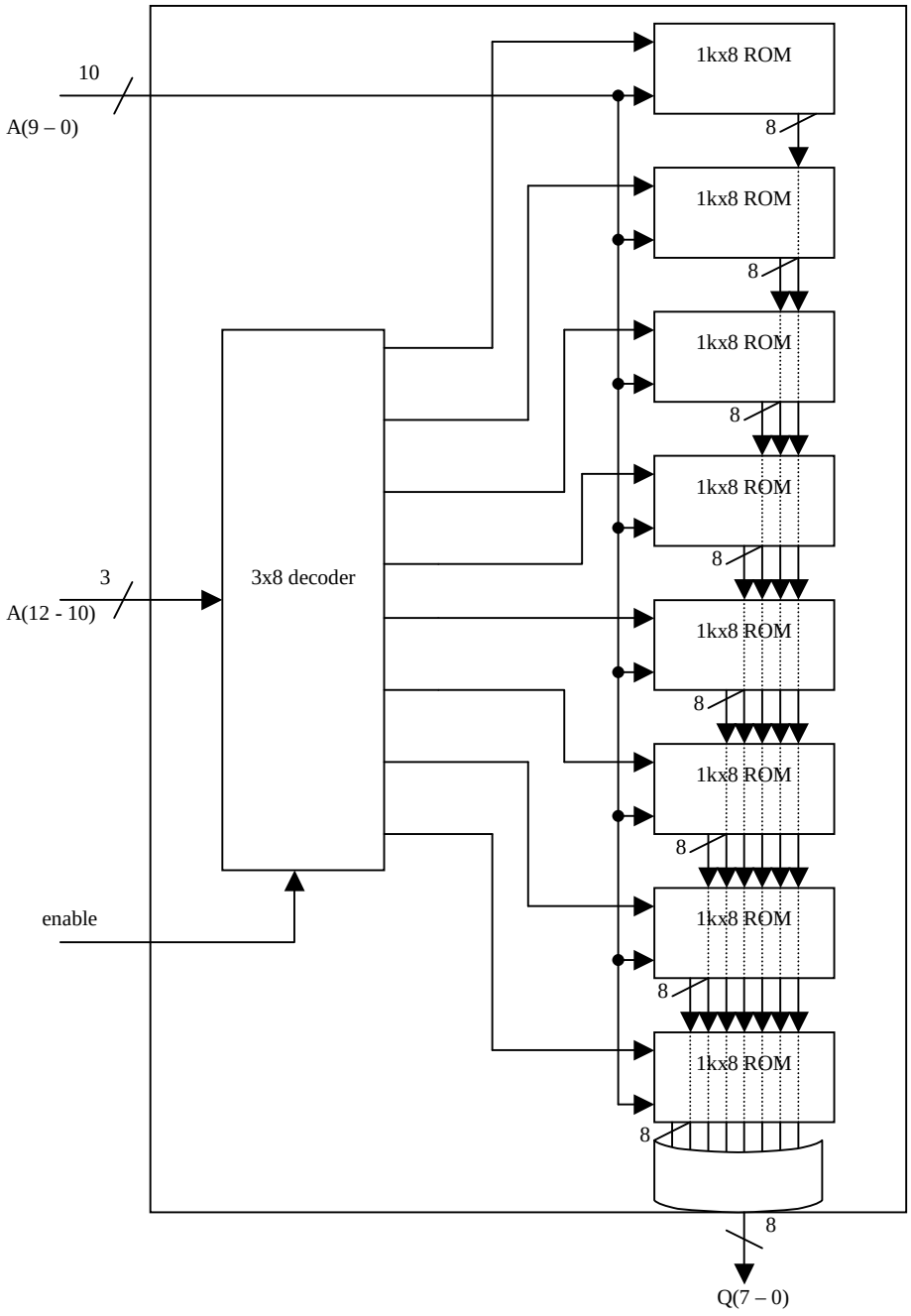
5.4 Sketch the internal design of a 4 × 3 RAM.



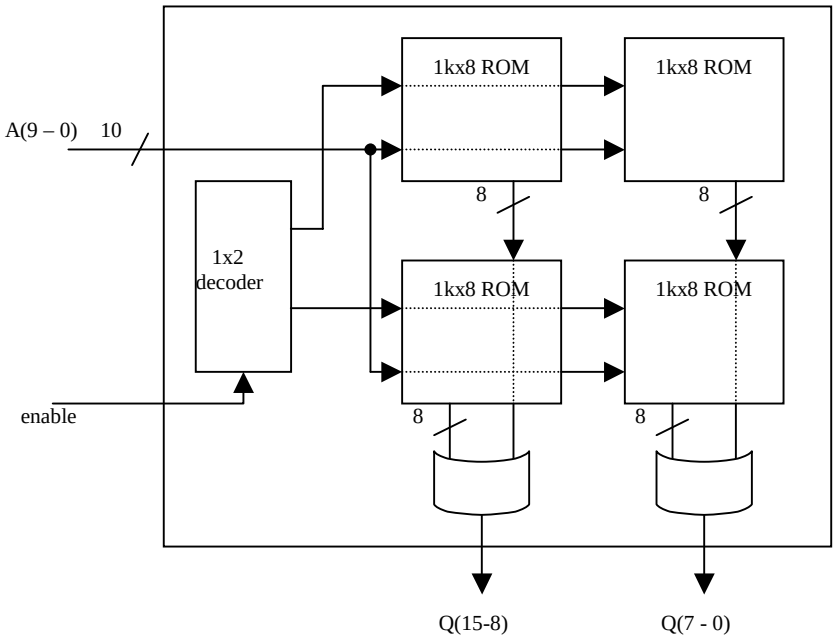
5.5 Compose 1K x 8 ROMs into a 1K × 32 ROM (Note: 1K actually means 1,024 words).



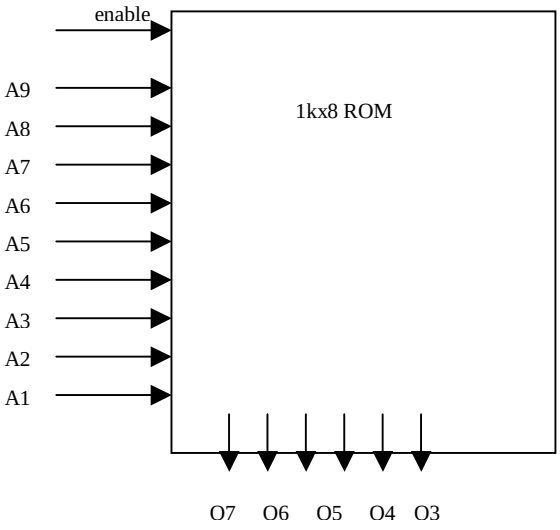
5.6 Compose 1K x 8 ROMs into an 8K × 8 ROM.


10
A(9 – 0)
1kx8 ROM
8
1kx8 ROM
8
1kx8 ROM
8
1kx8 ROM
8
3
A(12 - 10)
3x8 decoder
1kx8 ROM
8
1kx8 ROM
8
enable
1kx8 ROM
8
1kx8 ROM
8
8
Q(7 – 0)

5.7 Compose 1K x 8 ROMs into a 2K × 16 ROM.



5.8 Show how to use a 1K × 8 ROM to implement a 512 × 6 ROM.



5.9 Given the following three cache designs, find the one with the best performance by calculating the average cost of access. Show all calculations. (a) 4 Kbyte, 8-way set-associative cache with a 6% miss rate; cache hit costs one cycle, cache miss costs 12 cycles. (b) 8 Kbyte, 4-way set-associative cache with a 4% miss rate; cache hit costs two cycles, cache miss costs 12 cycles. (c) 16 Kbyte, 2-way set-associative cache with a 2% miss rate; cache hit costs three cycles, cache miss costs 12 cycles.

a.) 4 Kb, 8-way set-associative cache with a 6% miss rate; cache hit costs 1 cycle, cache miss costs 12 cycles.

miss rate = .06
hit rate = 1- miss rate = .94
.94 * 1cycle (hit) + .06 * 12 cycles (miss) = .94 + .72 = 1.66 cycles avg.

b.) 8 Kb, 4-way set-associative cache with a 4% miss rate; cache hit costs 2 cycles, cache miss costs 12 cycles.

miss rate = .04
hit rate = 1 – miss rate = .96
.96 * 2 cycles (hit) + .04 * 12 cycles (miss) = 1.92 + .48 = 2.4 cycles avg.

c.) 16 Kb, 2-way set-associative cache with a 2% miss rate; cache hit costs 3 cycles, cache miss costs 12 cycles.

miss rate = .02
hit rate = 1 – miss rate = .98
.98 * 3 cycles (hit) + .02 * 12 cycles (miss) = 2.94 + .24 = 3.18 cycles avg.

BEST PERFORMANCE: a) 1.66 cycles avg.

5.10 Given a 2-level cache design where the hit rates are 88% for the smaller cache and 97% for the larger cache, the access costs for a miss are 12 cycles and 20 cycles, respectively, and the access cost for a hit is one cycle, calculate the average cost of access.

hit rate = .88
L1 miss/L2 hit rate = .12 * .97
L1miss/L2 miss rate = .12 * .03

Avg. cost = (.88 * 1) + (.12 * .97 * 12) + (.12 * .03 * 20)
= .88 + 1.3968 + .072
= 2.3488 cycles

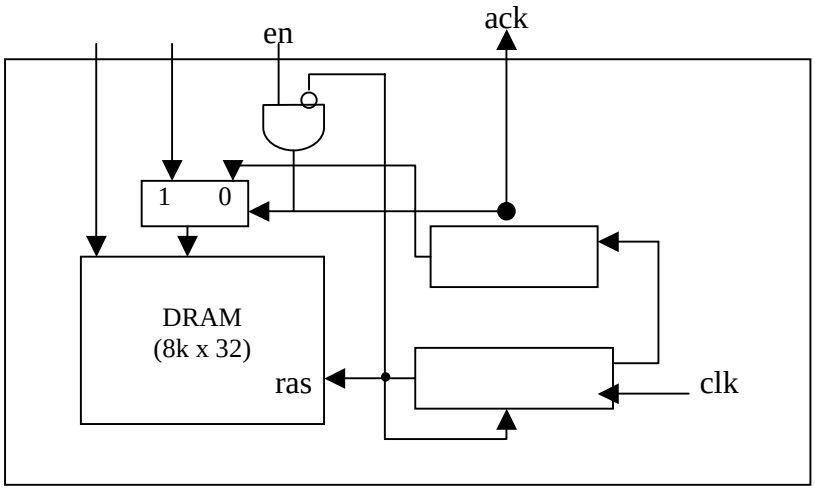
5.11 A given design with cache implemented has a main memory access cost of 20 cycles on a miss and two cycles on a hit. The same design without the cache has a main memory access cost of 16 cycles. Calculate the minimum hit rate of the cache to make the cache implementation worthwhile.

H = hit rate
miss rate = 1-hit rate = 1-H

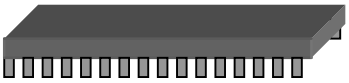
avg. memory access cost (cache) < memory access cost (no cache)
2 cycles * H + 20 cycles * (1-H) < 16 cycles
2H + 20 – 20H < 16
– 18H < – 4
H > (4/18) = .22 = 22% hit rate minimum

5.12 Design your own 8K × 32 PSRAM using an 8K × 32 DRAM, by designing a refresh controller. The refresh controller should guarantee refresh of each word every 15.625 microseconds. Because the PSRAM may be busy refreshing itself when a read or write access request occurs (i.e., the enable input is set), it should have an output signal ack indicating that an access request has been completed. Make use of a timer. Design the system down to complete structure. Indicate at what frequency your clock must operate.

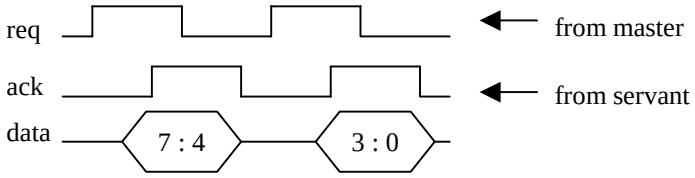
Assumptions:
- 512 rows of 16 words each
- Refreshes one row at a time (strobe *ras* with incremented address)
- *clk* must be 32.768 MHz (period = 15.625x10-6/512)
- *en* is AND'ed with *ras'*, so that it selects requested address only if *ras* is not selected (no refreshing going on). It also sets *ack* to high



CHAPTER 6: *Interfacing*



6.1 Draw the timing diagram for a bus protocol that is handshaked, nonaddressed, and transfers 8 bits of data over a 4-bit data bus.



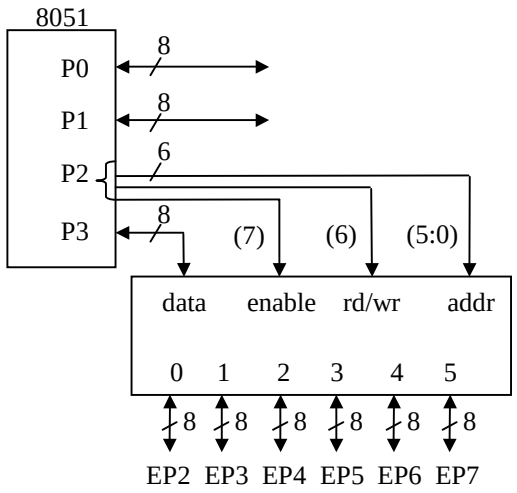
Servant puts data on the bus then asserts ack.

6.2 Explain the difference between port-based I/O and bus-based I/O.

With port-based I/O, the processor can read from or write to a port directly just as it would a register. Sometimes the processor can even access just one bit of the port. So, a peripheral can be accessed in the same way a register is accessed. Bus-based I/O, however, must use a bus protocol implemented in hardware to access any peripherals. The bus protocol writes to control, address, and data lines to communicate with the peripherals.

6.3 Show how to extend the number of ports on a 4-port 8051 to 8 by using extended parallel I/O. (a) Using block diagrams for the 8051 and the extended parallel I/O device, draw and label all interconnections and I/O ports. Clearly indicate the names and widths of all connections. (b) Give C code for a function that could be used to write to the extended ports.

(a) Using block diagrams for the 8051 and the extended parallel I/O device, draw and label all interconnections and I/O ports. Clearly indicate the names and widths of all connections



(b) Give C code for a function that could be used to write to the extended ports.

```
Ext_Port( unsigned char data, int EP, int rd_wr ) {
        unsigned char mask;
        P3 = data;
        if( rd_wr == 0 )        // sets enable to 1 and rd/wr to 0 or 1
                mask = 0x80;    // write (sets P2^6 = 0)
        else
                mask = 0xC0;    // read (sets P2^6 = 1)

        switch( EP ) {          // sets coreect Ext_Port to be enabled

                case 0:
                        P2 = mask | 0x01;
                        break;
                case 1:
                        P2 = mask | 0x02;
                        break;
                case 2:
                        P2 = mask | 0x04;
                        break;
                case 3:
                        P2 = mask | 0x08;
                        break;
                case 4:
                        P2 = mask | 0x10;
                        break;
                case 5:
                        P2 = mask | 0x20;
                        break;
                case 6:
                        P2 = mask | 0x40;
                        break;
                default:
                                // disable if incorrect EP given
```

```
                        P2 = mask & 0x00;
                        break;
                }
        }
```

6.4 Discuss the advantages and disadvantages of using memory-mapped I/O versus standard I/O.

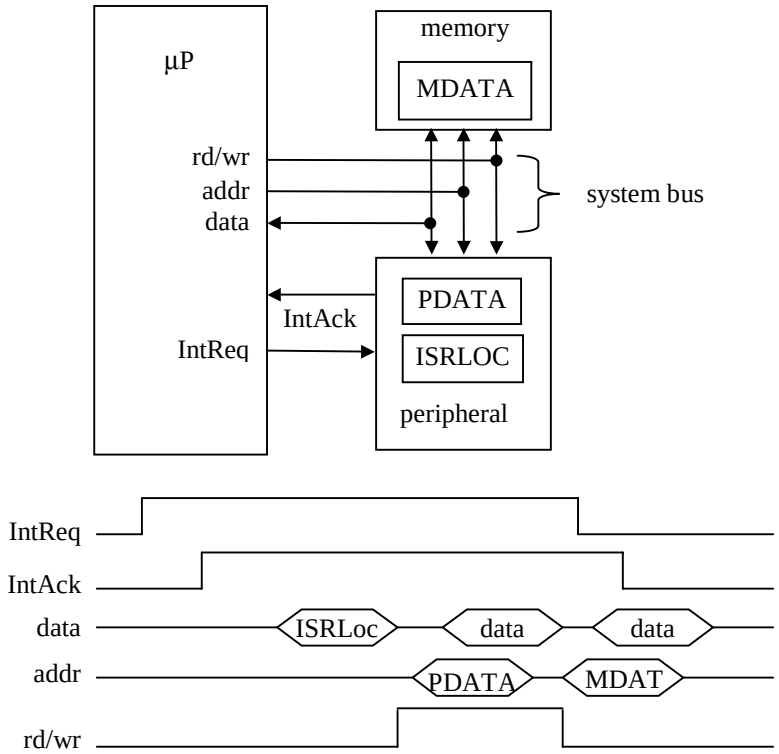
Memory-mapped I/O has the advantage of not needing special assembly instructions to access the peripherals. Any memory access instruction would also work with peripherals. The disadvantage to this method is that the peripheral addresses would decrease the number of memory addresses available with a particular address bus size, thus limiting the memory to a smaller size.

The standard I/O has a separate pin indicating an access to either memory or a peripheral. Therefore, the entire range of a particular address bus size can be used to access the memory. Another advantage to using standard I/O is that the decoding logic in a peripheral can be reduced since the bits needed to address only the peripheral will be much less.

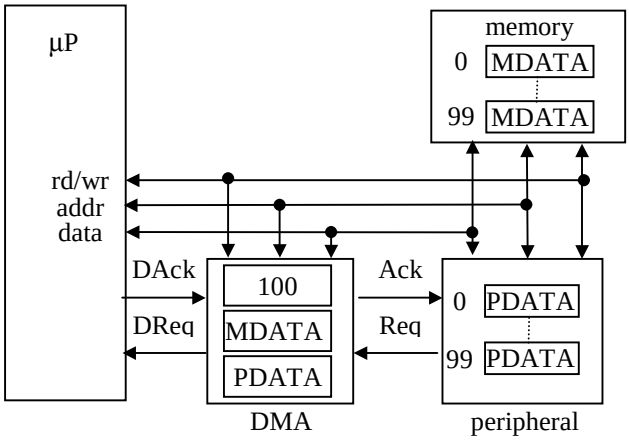
6.5 Explain the benefits that an interrupt address table has over fixed and vectored interrupt methods.

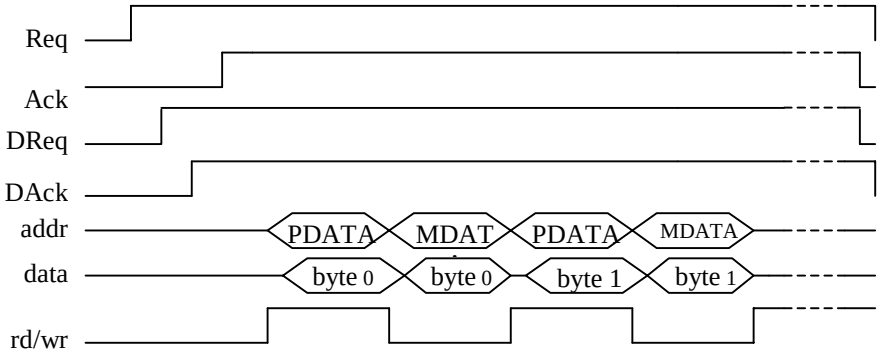
A benefit that the interrupt address table has over the fixed interrupt method is that an ISR location can be changed without having to change anything in the peripheral. Only the table will need to be updated. A benefit over the vectored interrupt method is that the table is much smaller than memory. The peripheral need only send over the entry number in the table, which will be much smaller than the address of the ISR. This is especially beneficial with a narrow data bus.

6.6 Draw a block diagram of a processor, memory, and peripheral connected with a system bus, in which the peripheral gets serviced using vectored interrupt. Assume servicing moves data from the peripheral to the memory. Show all relevant control and data lines of the bus, and label component inputs/outputs clearly. Use symbolic values for addresses. Provide a timing diagram illustrating what happens over the system bus during the interrupt.



6.7 Draw a block diagram of a processor, memory, peripheral, and DMA controller connected with a system bus, in which the peripheral transfers 100 bytes of data to the memory using DMA. Show all relevant control and data lines of the bus, and label component inputs/outputs clearly. Draw a timing diagram showing what happens during the transfer; skip the 2nd through 99th bytes.



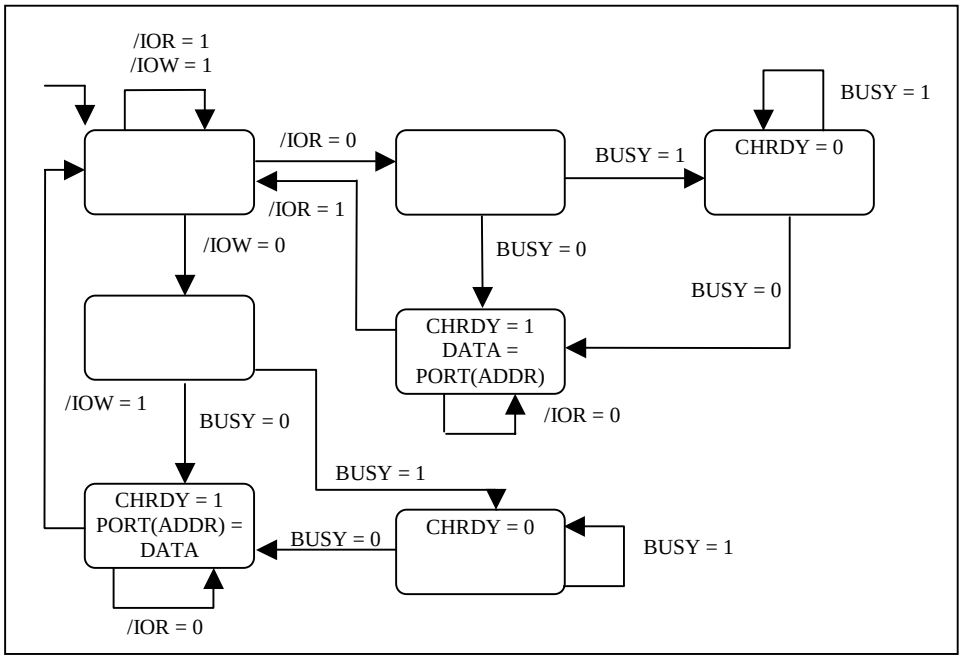


6.8 Repeat problem 6.7 for a daisy-chain configuration.

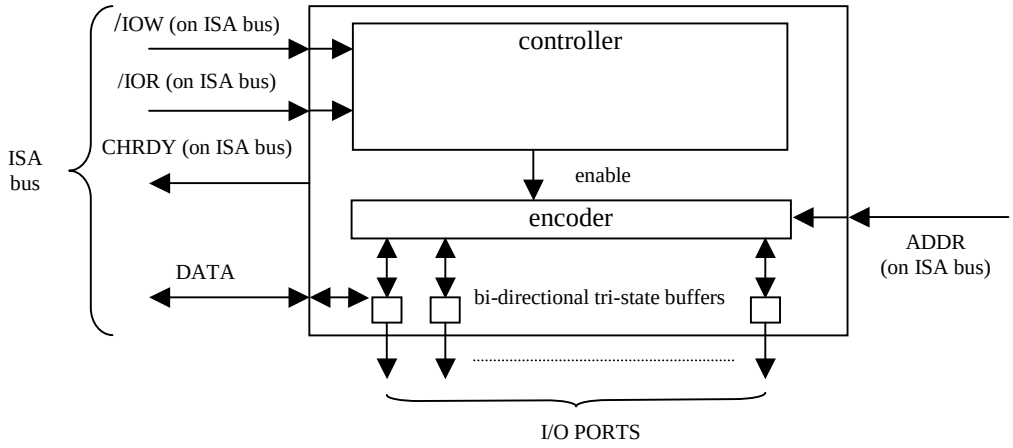
Error: Problem 6.7 is not the correct problem. There is no need for a daisy-chain configuration in Problem 6.7.

6.9 Design a parallel I/O peripheral for the ISA bus. Provide: (a) a state-machine description and (b) a structural description.

(a) state machine description

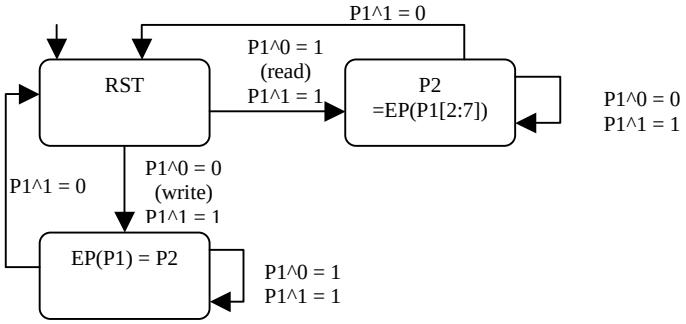


(b) structural description

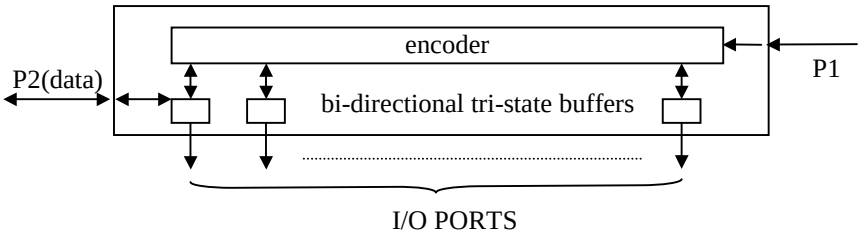


6.10 Design an extended parallel I/O peripheral. Provide: (a) a state-machine description and (b) a structural description.

(a) state-machine description



(b) structural description



6.11 List the three main transmission mediums described in the chapter. Give two common applications for each.

Parallel communication involves the simultaneous transfer of data words between two devices. It is usually only used between devices on the same IC or circuit board. Common parallel protocol applications are the PCI bus and the ARM bus.

Serial communication uses a single wire capable of sending only one bit of data at a time. I^2C actually has two wires with one wire used for control purposes. Another common serial protocol is the Universal Serial Bus, or USB.

Wireless communication typically uses infrared or radio frequencies to transfer data between two devices without a physical connection. A protocol that uses infrared is the IrDA protocol. Bluetooth is a new protocol based on radio frequencies.

6.12 Assume an 8051 is used as a master device on an I2C bus with pin P1.0 corresponding to I2C_Data and pin P1.1 corresponding to I2C_Clock. Write a set of C routines that encapsulate the details of the I2C protocol. Specifically, write the routines called StartI2C/StopI2C, that send the appropriate start/stop signal to slave devices. Likewise, write the routines ReadByte and WriteByte, each taking a device Id as input and performing the appropriate I/O actions.

```
I2C_start{
  P1.1 = 1;  // SCL held high
  P1.0 = 1;  // high to low on SDA
  delay();
  P1.0 = 0;
}

I2C_stop{
  P1.1 = 1;  // SCL held high
  P1.0 = 0;  // low to high on SDA
  delay();
  P1.0 = 1;
}

WriteByte(byte deviceID){
  I2C_start();  // start transfer

  for( int i = 7; i < 0; i-- ){  // send device ID
         P1.1 = 0;   // hold SCL low for each bit
         P1.0 = DeviceID^i;
         delay();
         P1.1 = 1;
  }

  P1.1 = 0;   // hold SCL low
  P1.0 = 0;  // write signal
  delay();
  P1.1 = 1;
```

```
    P1.1 = 0;  // get ACK from device
    delay();
    if( P1.0 == 1 ){  // did not get ack
           return(0);  // exit with error
    }

    for( int i = 7; i < 0; i-- ){  // send data byte

           P1.1 = 0;  // hold SCL low for each bit
           P1.0 = DATA^i;
           Delay();
           P1.1 = 1;
    }

    P1.1 = 0;  // get ACK from device
    delay();
    if( P1.0 == 1 ){  // did not get ack
           return(0);  // exit with error
    }
    I2C_stop();  // stop transfer
}

ReadByte(byte deviceID){
    I2C_start();  // start transfer
    for( int i = 7; i < 0; i-- ){  // send device ID
           P1.1 = 0;   // hold SCL low for each bit
           P1.0 = DeviceID^i;
           delay();
           P1.1 = 1;
    }
    P1.1 = 0;   // hold SCL low
    P1.0 = 1;  // read signal
    delay();
    P1.1 = 1;

    P1.1 = 0;  // get ACK from device
    delay();
    if( P1.0 == 1 ){  // did not get ack
           return(0);  // exit with error
    }

    for( int i = 7; i < 0; i-- ){  // receive data byte

           P1.1 = 0;  // hold SCL low for each bit
           DATA^i = P1.0;
           Delay();
           P1.1 = 1;
    }

    P1.1 = 0;  // get ACK from device
    delay();
    if( P1.0 == 1 ){  // did not get ack
           return(0);  // exit with error
    }
    I2C_stop();  // stop transfer
}
```

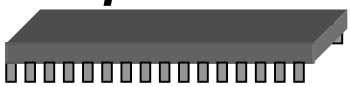
6.13 Select one of the following serial bus protocols, then perform an Internet search for information on transfer rate, addressing, error correction (if applicable), and plug-and-play capability (if applicable). Then give timing diagrams for a typical transfer of data (e.g., a write operation). The protocols are USB, I2O, Fibre Channel, SMBus, IrDA, or any other serial bus in use by the industry and not described in this book.

NOTE: Answers will vary.

6.14 Select one of the following parallel bus protocols, then, perform an Internet search for information on transfer rate, addressing , DMA and interrupt control (if applicable), and plug-and-play capability (if applicable). Then give timing diagrams for a typical transfer of data (e.g., a write operation). The protocols are STD 32, VME, SCSI, ATAPI, Micro Channel, or any other parallel bus in use by the industry and not described in this book.

NOTE: Answers will vary.

CHAPTER 7: *Digital Camera Example*



7.1 Using any programming language of choice, (a) implement the FDCT and IDCT equations presented in section 7.2 using double precision and floating-point arithmetic. (b) Use the block of 8 × 8 pixel given in Figure 1.2 (b) as input to your FDCT and obtain the encoded block. (c) Use the output of part (b) as input to your IDCT to obtain the original block. (e) Compute the percent error between your decoder's output and the original block.

a.) The C code for the FDCT and IDCT follows:

```
double C_d(int h)
{
  if (h == 0)
    return M_SQRT1_2;
  else
    return 1.0;
}

float C_f(int h)
{
  if (h == 0)
    return (float)(M_SQRT1_2);
  else
    return 1.0;
}

double FDCT_d(double pixels[8][8], int u, int v)
{
  double value = 0.25*C_d(u)*C_d(v);
  double totalSum = 0;
  for(int i =0; i<8; ++i)
  {
    for(int j = 0; j<8; ++j)
    {
      totalSum += pixels[i][j] * cos(M_PI*(2*i+1)*u/16) *
  cos(M_PI*(2*j+1)*v/16);
    }
  }
  return value*totalSum;
}
```

```
double IDCT_d(double values[8][8], int x, int y)
{
  double totalSum = 0;
  for (int u = 0; u < 8; ++u)
  {
    for (int v = 0; v < 8; ++v)
    {
      totalSum += C_d(u) * C_d(v) * values[u][v] *
  cos((double)M_PI*(2.0*(double)x+1.0)*(double)u/16.0) *
  cos((double)M_PI*(2.0*(double)y+1.0)*(double)v/16.0);
    }
  }
  return totalSum * 0.25;
}

float FDCT_f(float pixels[8][8], int u, int v)
{
  float value = 0.25*C_f(u)*C_f(v);
  float totalSum = 0;
  for(int i =0; i<8; ++i)
  {
    for(int j = 0; j<8; ++j)
    {
      totalSum += pixels[i][j] *
  ((float)(cos(M_PI*(2*i+1)*u/16))) *
  ((float)(cos(M_PI*(2*j+1)*v/16)));
    }
  }
  return value*totalSum;
}

float IDCT_f(float values[8][8], int x, int y)
{
  float totalSum = 0;
  for (int u = 0; u < 8; ++u)
  {
    for (int v = 0; v < 8; ++v)
    {
      totalSum += C_f(u) * C_f(v) * values[u][v] *
  ((float)(cos(M_PI*(2*x+1)*u/16))) *
  ((float)(cos(M_PI*(2*y+1)*v/16)));
    }
  }
  return totalSum * (float)(0.25);
}
```

(b) The encoded block from the C code is:

123	157	142	127	131	102	99	235
134	135	157	112	109	106	108	136
135	144	159	108	112	118	109	126
176	183	161	111	186	130	132	133
137	149	154	126	185	146	131	132
121	130	127	146	205	150	130	126
117	151	160	181	250	161	134	125
168	170	171	178	183	179	112	124

(c) The IDCT C code from above will result in the following block:

1150	39	–43	–10	26	–83	11	41
–81	–3	115	–73	–6	–2	22	–5
14	–11	1	–42	26	–3	17	–38
2	–61	–13	–12	36	–23	–18	5
44	13	37	–4	10	–21	7	–8
36	–11	–9	–4	20	–28	–21	14
–19	–7	21	–6	3	3	12	–21
–5	–13	–11	–17	–4	–1	7	–4

(e) The percent error between the decoder's output and the original block was 2.15 x 10^{-13} %.

7.2 Assuming 8 bits per each pixel value, calculate the length, in bits, of the block given in Figure 1.3 (b).

512 bits

7.3 Using the Huffman codes given in Figure 1.5, encode the block given in Figure 1.3 (b). (a) What is the length, in bits? (b) How much compression did we achieve be using Huffman encoding? Use the results of the last question to calculate this.

The encoded block is:

111111011010110001010011100100110
011101000111101111000100101000
111000100011010101001110101110
100101111110110011011110110010

011111111001100001011110010000
011000000010101011101111011110
110001010001001001110101000
00110001100010001000

(a) 226 bits
(b) The size has been reduced by 55.9%. (The encoded string is only 44.1% as big as the original)

7.4 Convert 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, and 1.9 to fixed-point representation using (a) two bits for the fractional part, (b) three bits for the fractional part, and (c) three bits for the fractional part.

(a) 2 bits for the fractional part

1.0 = 00000100
1.1 = 00000100
1.2 = 00000101
1.3 = 00000101
1.4 = 00000110
1.5 = 00000110
1.6 = 00000110
1.7 = 00000111
1.8 = 00000111
1.9 = 00001000

(b) 3 bits for the fractional part

1.0 = 00001000
1.1 = 00001001
1.2 = 00001010
1.3 = 00001010
1.4 = 00001011
1.5 = 00001100
1.6 = 00001101
1.7 = 00001110
1.8 = 00001110
1.9 = 00001111

(c) 4 bits for the fractional part
1.0 = 00010000
1.1 = 00010010
1.2 = 00010011

1.3 = 00010101
1.4 = 00010110
1.5 = 00011000
1.6 = 00011010
1.7 = 00011011
1.8 = 00011101
1.9 = 00011110

7.5 Write two C routines that, each, take as input two 32-bit fixed-point numbers and perform addition and multiplication using 4 bits for the fractional part and the remaining bits for the whole part.

The C code for the fixed-point addition and multiplication:

```
unsigned long FixedAdd(unsigned long a, unsigned long b)
{
  return a + b;
}
unsigned long FixedMul(unsigned long a, unsigned long b)
{
  unsigned long result = a * b;
  result >>= 4;
  return result;
}
```

7.6 Using any programming language of choice to (a) implement the FDCT and IDCT equations presented in Section 7.2 using fixed-point arithmetic with 4 bits used for the fractional part and the remaining bits used for the whole part, (b) use the block of 8 × 8 pixels given in Figure 1.2 (b) as input to your FDCT and obtain the encoded block, (c) use the output of part (b) as input to your IDCT to obtain the original block, and (d) compute the percent error between your decoder's output and the original block.

(a) C code for the fixed-point FDCT and IDCT:

```
int COSTABLE[8][8] = {
  { 16,  16,  15,  13,  11,   9,   6,   3 },
  { 16,  13,   6,  -3, -11, -16, -15,  -9 },
  { 16,   9,  -6, -16, -11,   3,  15,  13 },
  { 16,   3, -15,  -9,  11,  13,  -6, -16 },
  { 16,  -3, -15,   9,  11, -13,  -6,  16 },
  { 16,  -9,  -6,  16, -11,  -3,  15, -13 },
  { 16, -13,   6,   3, -11,  16, -15,   9 },
  { 16, -16,  15, -13,  11,  -9,   6,  -3 }
};

// 1/sqrt(2) in this fixed point representation is 11
// 1 in this fixed point representation is 16
int C(int h)
```

```
{
  if (h==0)
    return 11;
  else
    return 16;
}

// PI in this fixed point representation is 50

int FDCT_fixed(int array[8][8], int u, int v)
{
  int total = 0;
  int subtotal = 0;

  for(int x = 0; x < 8; ++x)
  {
    for(int y = 0; y < 8; ++y)
    {
      subtotal = (array[x][y] * COSTABLE[x][u]) >> 4;
      subtotal = (subtotal * COSTABLE[y][v]) >> 4;
      total += subtotal ;
    }
  }



  // .25 in this fixed point is 4

  total = (total * C(u)) >> 4;
  total = (total * C(v)) >> 4;
  total = (total * 4) >> 4;

  return total;
}

int IDCT_fixed(int array[8][8], int x, int y)
{
  int total = 0;
  int subtotal = 0;

  for(int u = 0; u < 8; ++u)
  {
    for(int v=0; v < 8; ++v)
    {
      subtotal = (C(u) * C(v)) >> 4;
      subtotal = (subtotal * array[u][v]) >> 4;
      subtotal = (subtotal * COSTABLE[x][u]) >> 4;
      subtotal = (subtotal * COSTABLE[y][v]) >> 4;
      total += subtotal;
    }
  }

  total = (total * 4) >> 4;

  return total;
}
```

(b) Depending on the actual implementation of fixed-point FDCT the results will vary. The following is a sample generated from the above code.

17384	593	-675	-156	380	-1303	175	636
-1260	-79	1843	-1191	-98	-65	310	-109
208	-208	6	-694	387	-71	267	-644
28	-989	-250	-171	551	-376	-296	77
663	190	565	-77	131	-348	87	-132
566	-213	-180	-73	310	-470	-351	213
-306	-133	337	-92	31	26	166	-354
-71	-240	-215	-270	-77	-24	101	-78

(c) Depending on the actual implementation of fixed-point IDCT the results will vary. The following is a sample of the restored original block generated from the above code.

1512	2129	1856	1633	1644	1251	1211	3360
1791	1771	2125	1410	1327	1312	1353	1805
1790	1916	2145	1342	1371	1498	1365	1652
2410	2499	2153	1361	2539	1666	1729	1700
1772	1957	2045	1619	2549	1940	1718	1685
1532	1683	1624	1939	2862	2020	1706	1609
1441	2004	2145	2494	3577	2179	1753	1579
2253	2282	2314	2448	2484	2460	1387	1549

(d) The percent error between the decoder's output and the original block using the above C code was 23.9106%.

7.7 List the modifications made in implementations 2 and 3 and discuss why each was beneficial in terms of performance.

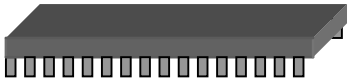
Implementation 2 modifications:

Addition of a CCDPP co-processor.- The addition of this custom logic frees up over 400,000 instructions on the 8051 as the CCDPP co-processor executes only the CCDPP very efficiently.

Implementation 3 modifications:

Changed the algorithm to use fixed point representation - This eliminated millions of instructions generated by the compiler to emulate floating point operations on an architecture that did not support them.

CHAPTER 8: *State Machine and Concurrent Process Models*



8.1 Define the following terms: finite-state machines, concurrent processes, real-time systems, and real-time operating system.

Finite State Machine : A model to describe system behavior as a set of possible states with transistors between states depending on input values.

Concurrent Processes: An independent sub-behavior of a system executing simultaneously while possibly sharing common data.

Real-time System: Systems that are fundamentally composed of two or more concurrent processes that execute with stringent timing requirements and cooperate with each other in order to accomplish a common goal.

Real-time Operating System: The underlying implementations or systems that support real-time systems.

8.2 Briefly describe three computation models commonly used to describe embedded systems and/or their peripherals. For each model list two languages that can be used to capture it.

1. State Machines
These describe behavior as a series of states and trnsitiors between states.
Possbile Languages: C, VHDL

2. Sequential Porgram
A sequential program describes a series of statements iterated and conditionally executed.
Possible Languages: Java, C++

3. Dataflow Model

A dataflow model describes a system using a set of nodes which represent transformation and a set of directed edges representing the flow from one node to another.
Possible Languages: C++, Java

8.3 Describe the elevator UnitControl state machine in Figure 1.4 using the FSMD model definition <S, I, O, V, F, H, s0> given in this chapter. In other words, list the set of states (S), set of inputs (I), and so on.

S is a set of states { going up, idle, going down, door open }
I is a set of inputs { floor, req }
O is a set of ouputs { up, down, open }
V is a set of variables { timer }
F is the next-state function, mapping states and inputs and variable to states
{ going up X req > floor → going up,
going up X !(req > floor) → door open,
idle X req > floor → going up,
idle X req == floor → idle,
idle X !(req > floor) → going down,
going down X req < floor → going down,
going down X !(req < floor) → door open,
door open X timer < 10 → door open,
door open X !(timer < 10) → idle }
H is the action function mapping current states to outputs and variables
{ going up → up=1 down=0 open=0 + time=0,
idle → up=0 down=0 open=1 + time=0,
going down → up=0 down=1 open=0 + time=0,
door open → up=0 down=0 open=1 + time=1 }
So is the initial state {idle}

8.4 Show how using the process create and join semantics one can emulate the procedure call semantics of a sequential programming model.

```
functionA( ){
        ....
}

void main( ){
        ...
        process_create(functionA);
        join(functionA);
}
```

8.5 List three requirements of real-time systems and briefly describe each. Give examples of actual real-time systems to support your arguments.

1. provide means for communication
Within concurrent processes, it is essential for them to communicate with one antoher.

2. synchronization
Concurrent processes may read and write data at the same location in memory, and therefore synchronization is required to ensure correctness.

3. stringent timing requirements
Processes must produce and respond to data within strict timing specified by the designer.

8.6 Show why, in addition to ordered locking, two-phase locking is necessary in order to avoid deadlocks. Give an example and execution trace that results in deadlock if two-phase locking is not used.

Error**:** Ordered locking couldn't lead to deadlock.

8.7 Give pseudo-code for a pair of functions implementing the send and receive communication constructs. You may assume that mutex and condition variables are provided.

```
recieve( S, &data){
    msg.wait();
    process data
}
```

```
send( D, &data){
    write the message to data
    msg.signal();
}
```

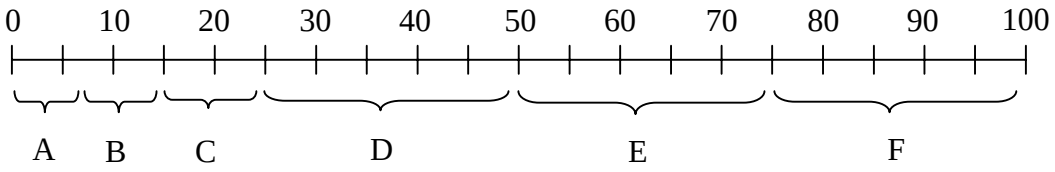
8.8 Show that the buffer size in the consumer-producer problem implemented in Figure 1.9 will never exceed one. Re-implement this problem, using monitors, to allow the size of the buffer to reach its maximum.

Error: FIG 1.9 DOES NOT REFER TO THE CORRECT FIGURE
FIG 1.19 IMPLEMENTS THE CONSUMER PRODUCER PROBLEM W/MONITORS

8.9 Given the processes A through F in Figure 1.21 (a) where their deadline equals their period, determine whether they can be scheduled on time using a non-preemptive scheduler. Assume all processes begin at the same time and their execution times are as follows: A: 8 ms, B: 25 ms, C: 6 ms, D: 25 ms, E: 10 ms, F: 25 ms. Explain your answer by showing that each process either meets or misses its deadline.

Process	execution time (ms)	deadline (ms)	start time (ms)	end time (ms)	deadline met

A	8	25	6	14	yes
B	25	50	24	49	yes
C	6	12	0	6	yes
D	25	100	74	99	yes
E	10	40	14	24	yes
F	25	75	49	74	yes



CHAPTER 9: *Control Systems*

9.1 Explain the difference between open-looped and closed-looped control systems. Why are we more concerned with closed-looped systems?

In an open-looped control system the controller reads in the reference input then computes a setting for the actuator. In a closed-looped control system the controller monitors the error between the plant output and the reference inputs then sets the plant input accordingly.

9.2 List and describe the eight parts of the closed-loop system. Give a real-life example of each (other than those mentioned in the book).

a. plant : the physical system to be controlled. An airplane is an example of a plant.
b. output: the particular physical system aspect we are interested in controlling. The altitude of an airplane is an example of this.
c. reference input: the desired value we want as output. The altitude set by the pilot is and example of this.
d. actuator: a device used to control input to the plant. A motor used to control the angle of an airplane's flaps is an example of this.
e. controller: a device used to control input to the plant. A microcontroller is an example of this.
f. disturbance: and additional undesirable input to the plant imposed by the environment that may cause the plant output to differ from what we would have expected based on the plant input. Wind is an example of this.
g. sensor: measures the plant output.
h. error detector: determines the difference between the plant output and the reference input.

9.3 Using a spreadsheet program, create a simulation of the cruise-control systems given in this chapter, using PI control only. Show simulations (graphs and data tables) for the

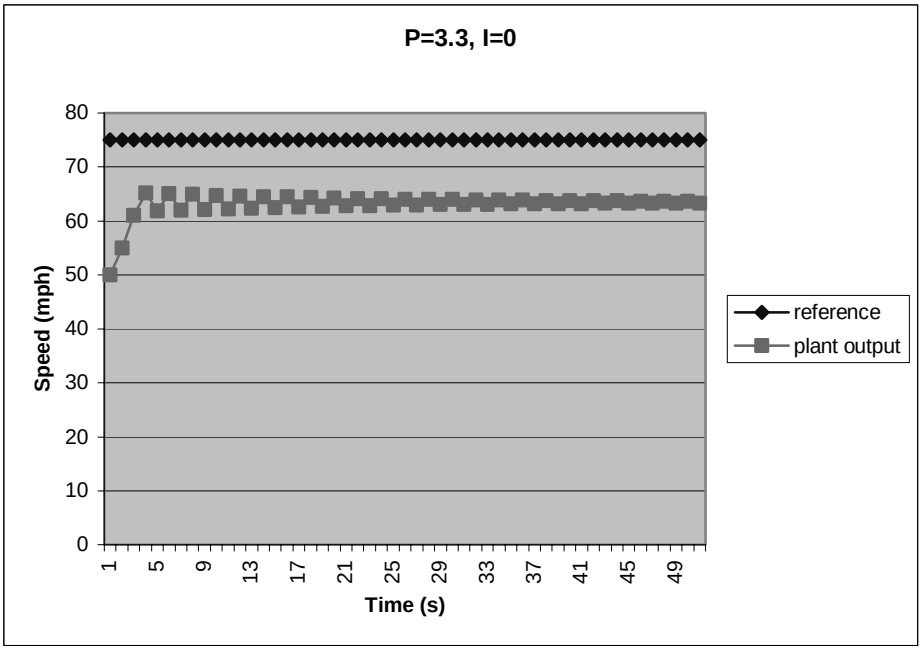
following *P* and *I* values. Remember to include throttle saturation in your equations. You can ignore disturbance. (a) *P* = 3.3, *I* = 0. (b) *P* = 4.0, *I* = 0 (How do the results differ from part (a)? Explain!) (c) *P* = 3.3, *I* = *X*. (d) *P* = 3.3, *I* = *Y*. (Choose *X* and *Y* to achieve a meaningful trade-off. Explain that trade-off.

(a) P=3.3, I=0

Car Simulation P=3.3, I=0

time	rt	et = rt - vt	st = st-1 + et	ut = Pet + Ist	Vt+1= 0.7Vt + 0.5ut	vt
0	75	25.00	25.00	40.00	55.00	50.00
1	75	20.00	45.00	45.00	61.00	55.00
2	75	14.00	59.00	45.00	65.20	61.00
3	75	9.80	68.80	32.34	61.81	65.20
4	75	13.19	81.99	43.53	65.03	61.81
5	75	9.97	91.96	32.90	61.97	65.03
6	75	13.03	104.99	43.00	64.88	61.97
7	75	10.12	115.11	33.40	62.12	64.88
8	75	12.88	127.99	42.52	64.74	62.12
9	75	10.26	138.26	33.86	62.25	64.74
10	75	12.75	151.01	42.08	64.61	62.25
11	75	10.39	161.39	34.27	62.37	64.61
12	75	12.63	174.03	41.69	64.50	62.37
13	75	10.50	184.52	34.64	62.47	64.50
14	75	12.53	197.05	41.34	64.40	62.47
15	75	10.60	207.65	34.98	62.57	64.40
16	75	12.43	220.08	41.02	64.31	62.57
17	75	10.69	230.77	35.28	62.66	64.31
18	75	12.34	243.12	40.73	64.23	62.66
19	75	10.77	253.89	35.55	62.73	64.23
20	75	12.27	266.15	40.48	64.15	62.73
21	75	10.85	277.00	35.80	62.81	64.15
22	75	12.19	289.20	40.24	64.08	62.81
23	75	10.92	300.11	36.02	62.87	64.08
24	75	12.13	312.24	40.03	64.02	62.87
25	75	10.98	323.22	36.22	62.93	64.02
26	75	12.07	335.29	39.84	63.97	62.93
27	75	11.03	346.32	36.40	62.98	63.97
28	75	12.02	358.34	39.67	63.92	62.98
29	75	11.08	369.42	36.57	63.03	63.92
30	75	11.97	381.40	39.51	63.87	63.03
31	75	11.13	392.52	36.71	63.07	63.87
32	75	11.93	404.45	39.37	63.83	63.07
33	75	11.17	415.62	36.85	63.11	63.83
34	75	11.89	427.51	39.25	63.80	63.11

35	75	11.20	438.71	36.97	63.14	63.80
36	75	11.86	450.57	39.13	63.77	63.14
37	75	11.23	461.81	37.07	63.17	63.77
38	75	11.83	473.63	39.03	63.74	63.17
39	75	11.26	484.90	37.17	63.20	63.74
40	75	11.80	496.70	38.94	63.71	63.20
41	75	11.29	507.99	37.26	63.23	63.71
42	75	11.77	519.76	38.85	63.68	63.23
43	75	11.32	531.08	37.34	63.25	63.68
44	75	11.75	542.83	38.78	63.66	63.25
45	75	11.34	554.16	37.41	63.27	63.66
46	75	11.73	565.89	38.71	63.64	63.27
47	75	11.36	577.25	37.48	63.29	63.64
48	75	11.71	588.96	38.65	63.63	63.29
49	75	11.37	600.34	37.53	63.31	63.63
50	75	11.69	612.03	38.59	63.61	63.31

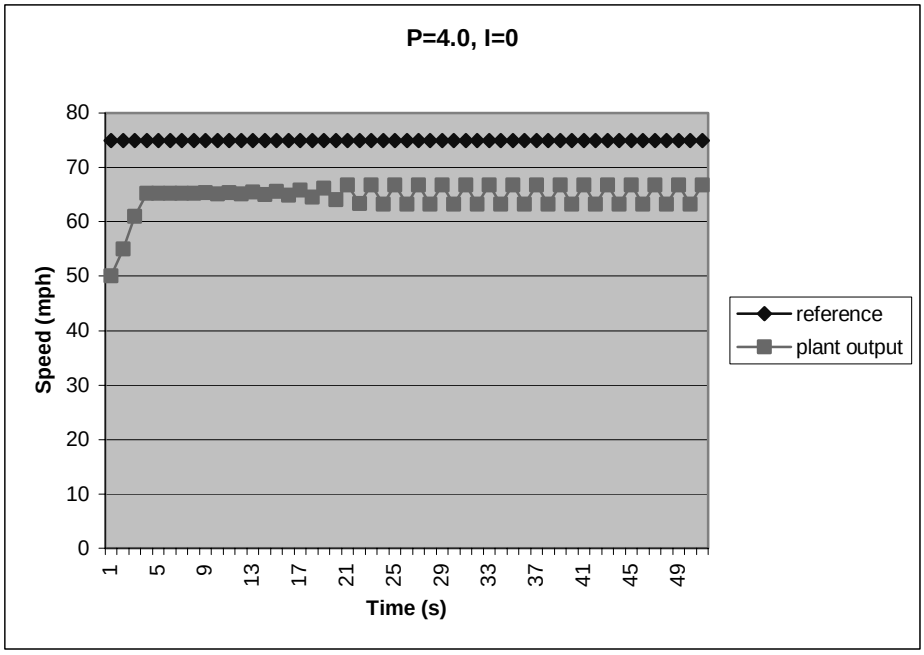


(b) P=4.0, I=0

Car Simulation P=4.0, I=0

time	rt	et = rt - vt	st = st-1 + et	ut = Pet + Ist	Vt+1= 0.7Vt + 0.5ut	vt
0	75	25.00	25.00	40.00	55.00	50.00
1	75	20.00	45.00	45.00	61.00	55.00
2	75	14.00	59.00	45.00	65.20	61.00
3	75	9.80	68.80	39.20	65.24	65.20
4	75	9.76	78.56	39.04	65.19	65.24
5	75	9.81	88.37	39.25	65.26	65.19
6	75	9.74	98.12	38.98	65.17	65.26
7	75	9.83	107.95	39.33	65.28	65.17
8	75	9.72	117.67	38.87	65.13	65.28
9	75	9.87	127.53	39.47	65.33	65.13
10	75	9.67	137.21	38.69	65.08	65.33
11	75	9.92	147.13	39.70	65.40	65.08
12	75	9.60	156.73	38.39	64.98	65.40
13	75	10.02	166.75	40.09	65.53	64.98
14	75	9.47	176.22	37.88	64.81	65.53
15	75	10.19	186.41	40.75	65.74	64.81
16	75	9.26	195.67	37.02	64.53	65.74
17	75	10.47	206.13	41.87	66.11	64.53
18	75	8.89	215.03	35.57	64.06	66.11
19	75	10.94	225.97	43.76	66.72	64.06
20	75	8.28	234.24	33.11	63.26	66.72
21	75	11.74	245.98	45.00	66.78	63.26
22	75	8.22	254.20	32.87	63.18	66.78
23	75	11.82	266.02	45.00	66.73	63.18
24	75	8.27	274.29	33.09	63.25	66.73
25	75	11.75	286.04	45.00	66.78	63.25
26	75	8.22	294.26	32.89	63.19	66.78
27	75	11.81	306.07	45.00	66.73	63.19
28	75	8.27	314.34	33.07	63.25	66.73
29	75	11.75	326.09	45.00	66.77	63.25
30	75	8.23	334.31	32.90	63.19	66.77
31	75	11.81	346.12	45.00	66.74	63.19
32	75	8.26	354.39	33.06	63.24	66.74
33	75	11.76	366.14	45.00	66.77	63.24
34	75	8.23	374.37	32.92	63.20	66.77
35	75	11.80	386.17	45.00	66.74	63.20
36	75	8.26	394.43	33.04	63.24	66.74
37	75	11.76	406.19	45.00	66.77	63.24
38	75	8.23	414.43	32.93	63.20	66.77

39	75	11.80	426.22	45.00	66.74	63.20
40	75	8.26	434.48	33.03	63.24	66.74
41	75	11.76	446.25	45.00	66.77	63.24
42	75	8.23	454.48	32.94	63.21	66.77
43	75	11.79	466.28	45.00	66.74	63.21
44	75	8.26	474.53	33.03	63.23	66.74
45	75	11.77	486.30	45.00	66.76	63.23
46	75	8.24	494.54	32.95	63.21	66.76
47	75	11.79	506.33	45.00	66.75	63.21
48	75	8.25	514.58	33.02	63.23	66.75
49	75	11.77	526.35	45.00	66.76	63.23
50	75	8.24	534.59	32.95	63.21	66.76



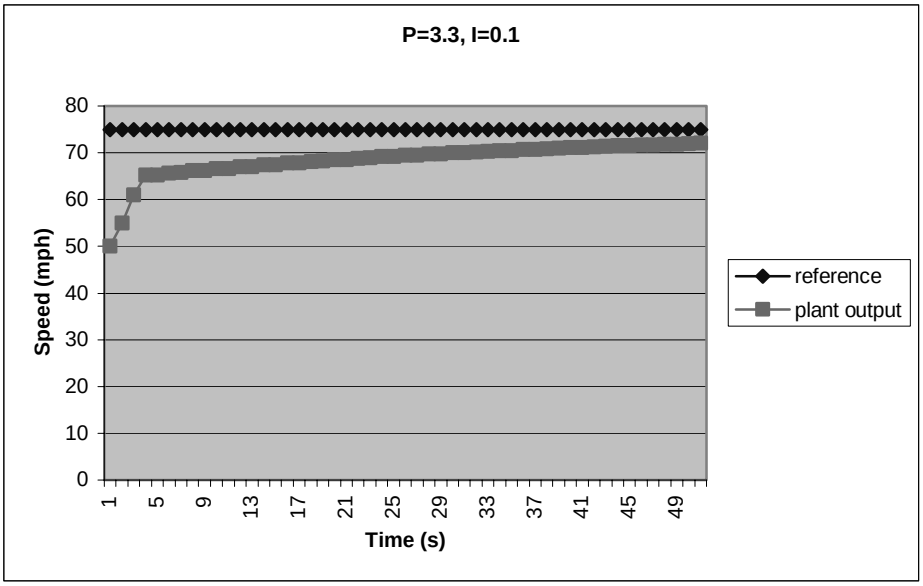
The value of P has a stability constraint. In part a, the value of P was low enough that the system eventually stabilized. In part b, we see that we have exceeded this stability constraint and therefore will have an unstable system. This is shown by the oscillation in the graph above.

(c) P = 3.3, I=X where x is 0.1

Car Siumulation P=3.3, I=0.1

time	rt	et = rt - vt	st = st-1 + et	ut = Pet + Ist	Vt+1= 0.7Vt + 0.5ut	vt
0	75	25.00	25.00	40.00	55.00	50.00
1	75	20.00	45.00	45.00	61.00	55.00
2	75	14.00	59.00	45.00	65.20	61.00
3	75	9.80	68.80	39.22	65.25	65.20
4	75	9.75	78.55	40.03	65.69	65.25
5	75	9.31	87.86	39.51	65.74	65.69
6	75	9.26	97.12	40.28	66.16	65.74
7	75	8.84	105.97	39.78	66.20	66.16
8	75	8.80	114.77	40.51	66.60	66.20
9	75	8.40	123.17	40.04	66.64	66.60
10	75	8.36	131.53	40.74	67.02	66.64
11	75	7.98	139.51	40.29	67.06	67.02
12	75	7.94	147.45	40.95	67.42	67.06
13	75	7.58	155.03	40.53	67.46	67.42
14	75	7.54	162.58	41.15	67.80	67.46
15	75	7.20	169.78	40.75	67.83	67.80
16	75	7.17	176.95	41.35	68.16	67.83
17	75	6.84	183.79	40.96	68.19	68.16
18	75	6.81	190.60	41.53	68.50	68.19
19	75	6.50	197.10	41.17	68.53	68.50
20	75	6.47	203.57	41.70	68.82	68.53
21	75	6.18	209.75	41.36	68.86	68.82
22	75	6.14	215.89	41.87	69.13	68.86
23	75	5.87	221.76	41.54	69.16	69.13
24	75	5.84	227.60	42.02	69.43	69.16
25	75	5.57	233.17	41.71	69.45	69.43
26	75	5.55	238.72	42.17	69.70	69.45
27	75	5.30	244.01	41.88	69.73	69.70
28	75	5.27	249.28	42.31	69.97	69.73
29	75	5.03	254.31	42.03	69.99	69.97
30	75	5.01	259.32	42.45	70.22	69.99
31	75	4.78	264.10	42.18	70.25	70.22
32	75	4.75	268.85	42.58	70.46	70.25
33	75	4.54	273.39	42.32	70.48	70.46
34	75	4.52	277.91	42.70	70.69	70.48
35	75	4.31	282.22	42.46	70.71	70.69
36	75	4.29	286.51	42.81	70.90	70.71
37	75	4.10	290.61	42.58	70.92	70.90
38	75	4.08	294.69	42.92	71.11	70.92

39	75	3.89	298.58	42.70	71.13	71.11
40	75	3.87	302.45	43.03	71.30	71.13
41	75	3.70	306.15	42.82	71.32	71.30
42	75	3.68	309.83	43.12	71.49	71.32
43	75	3.51	313.35	42.93	71.50	71.49
44	75	3.50	316.84	43.22	71.66	71.50
45	75	3.34	320.18	43.03	71.68	71.66
46	75	3.32	323.50	43.31	71.83	71.68
47	75	3.17	326.67	43.13	71.85	71.83
48	75	3.15	329.82	43.39	71.99	71.85
49	75	3.01	332.84	43.22	72.00	71.99
50	75	3.00	335.83	43.47	72.14	72.00

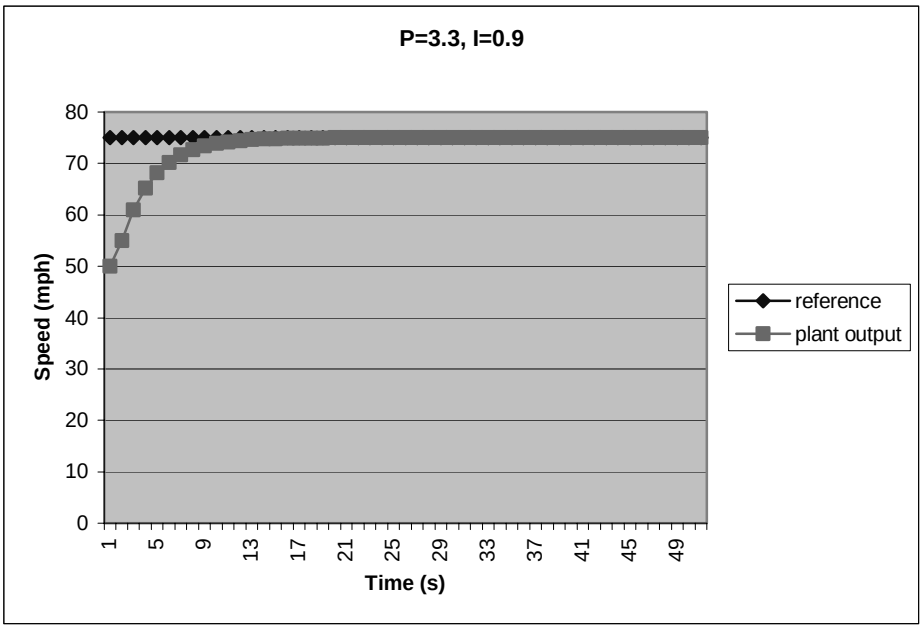


(d) P= 3.3, I=Y where y is 0.9

Car Simulation P=3.3, I=1.0

time	rt	et = rt - vt	st = st-1 + et	ut = Pet + Ist	Vt+1= 0.7Vt + 0.5ut	vt
0	75	25.00	25.00	40.00	55.00	50.00
1	75	20.00	45.00	45.00	61.00	55.00
2	75	14.00	59.00	45.00	65.20	61.00
3	75	9.80	68.80	45.00	68.14	65.20
4	75	6.86	75.66	45.00	70.20	68.14
5	75	4.80	80.46	45.00	71.64	70.20

6	75	3.36	83.82	45.00	72.65	71.64
7	75	2.35	86.18	45.00	73.35	72.65
8	75	1.65	87.82	45.00	73.85	73.35
9	75	1.15	88.98	45.00	74.19	73.85
10	75	0.81	89.78	45.00	74.44	74.19
11	75	0.56	90.35	45.00	74.60	74.44
12	75	0.40	90.74	45.00	74.72	74.60
13	75	0.28	91.02	45.00	74.81	74.72
14	75	0.19	91.21	45.00	74.86	74.81
15	75	0.14	91.35	45.00	74.91	74.86
16	75	0.09	91.45	45.00	74.93	74.91
17	75	0.07	91.51	45.00	74.95	74.93
18	75	0.05	91.56	45.00	74.97	74.95
19	75	0.03	91.59	45.00	74.98	74.97
20	75	0.02	91.61	45.00	74.98	74.98
21	75	0.02	91.63	45.00	74.99	74.98
22	75	0.01	91.64	45.00	74.99	74.99
23	75	0.01	91.65	45.00	74.99	74.99
24	75	0.01	91.65	45.00	75.00	74.99
25	75	0.00	91.66	45.00	75.00	75.00
26	75	0.00	91.66	45.00	75.00	75.00
27	75	0.00	91.66	45.00	75.00	75.00
28	75	0.00	91.66	45.00	75.00	75.00
29	75	0.00	91.66	45.00	75.00	75.00
30	75	0.00	91.67	45.00	75.00	75.00
31	75	0.00	91.67	45.00	75.00	75.00
32	75	0.00	91.67	45.00	75.00	75.00
33	75	0.00	91.67	45.00	75.00	75.00
34	75	0.00	91.67	45.00	75.00	75.00
35	75	0.00	91.67	45.00	75.00	75.00
36	75	0.00	91.67	45.00	75.00	75.00
37	75	0.00	91.67	45.00	75.00	75.00
38	75	0.00	91.67	45.00	75.00	75.00
39	75	0.00	91.67	45.00	75.00	75.00
40	75	0.00	91.67	45.00	75.00	75.00
41	75	0.00	91.67	45.00	75.00	75.00
42	75	0.00	91.67	45.00	75.00	75.00
43	75	0.00	91.67	45.00	75.00	75.00
44	75	0.00	91.67	45.00	75.00	75.00
45	75	0.00	91.67	45.00	75.00	75.00
46	75	0.00	91.67	45.00	75.00	75.00
47	75	0.00	91.67	45.00	75.00	75.00
48	75	0.00	91.67	45.00	75.00	75.00
49	75	0.00	91.67	45.00	75.00	75.00



50	75	0.00	91.67	45.00	75.00	75.00

The I value in problem (c) and (d) was chosen to show steady state error elimination. One problem associated with this is if I was increased too much the response result became unstable or oscilated.

9.4 Write a generic PID controller in C.

```
typedef struct PID_DATA {
    double Pgain, Dgain, Igain;
    double sensor_value_previous; // find the derivative
    double error_sum; // cumulative error
}

double PidUpdate(PID_DATA *pid_data, double sensor_value, double
reference_value)
{
    double Pterm, Iterm, Dterm;
    double error, difference;
    error = reference_value – sensor_value;
    Pterm = pid_data->Pgain * error; /* proportional term*/
    pid_data->error_sum += error; /* current + cumulative*/
    // the integral term
    Iterm = pid_data->Igain * pid_data->error_sum;
    difference = pid_data->sensor_value_previous –
    sensor_value;
    // update for next iteration
```

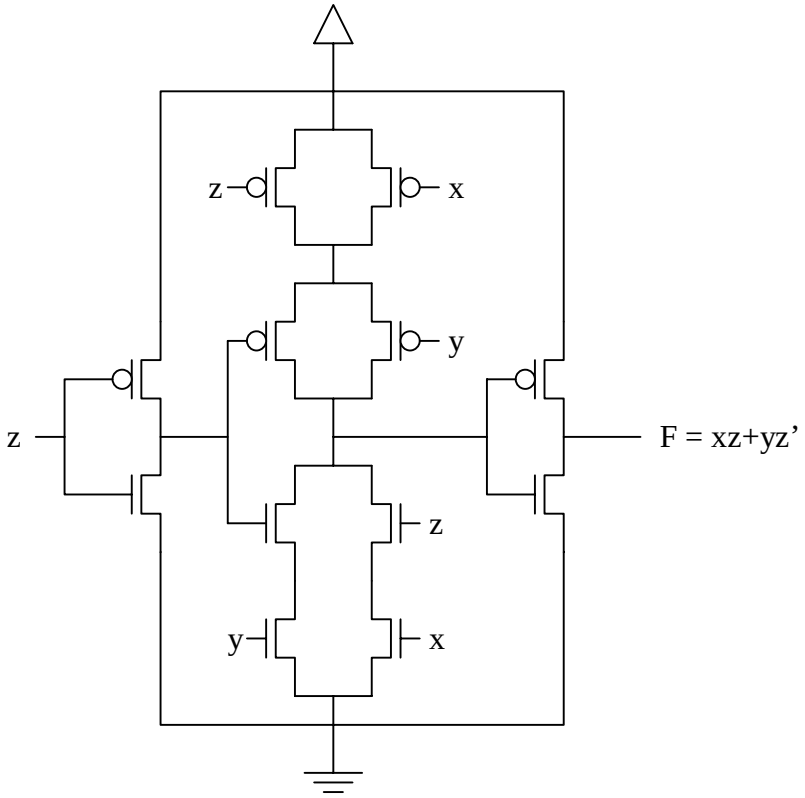
```
    pid_data->sensor_value_previous = sensor_value;
    // the derivative term
    Dterm = pid_data->Dgain * difference;
    return (Pterm + Iterm + Dterm);
}

void main()
{
    double sensor_value, actuator_value, error_current;
    PID_DATA pid_data;
    PidInitialize(&pid_data);
    while (1) {
        sensor_value = SensorGetValue();
        reference_value = ReferenceGetValue();
        actuator_value                                                    =
              PidUpdate(&pid_data,sensor_value,reference_value);
        ActuatorSetValue(actuator_value);
    }
}
```

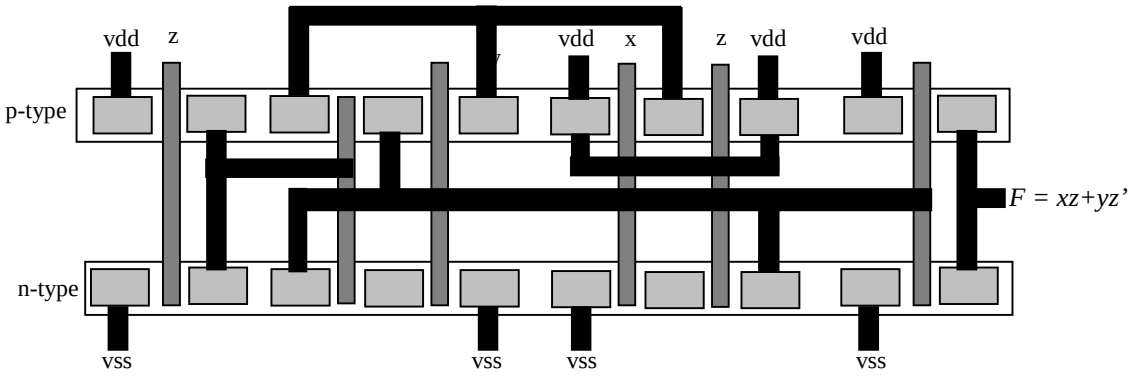
CHAPTER 10: *IC Technology*

10.1 Using the NAND gate (shown in Figure 1.1) as building block, (a) draw the circuit schematic for the function $F = xz + yz'$, and (b) draw the top-down view of the circuit on an IC (make your layout as compact as possible.)

(a) Transistor level circuit schematic for $F = xz + yz'$



(b) Layout for $F = xz + yz'$

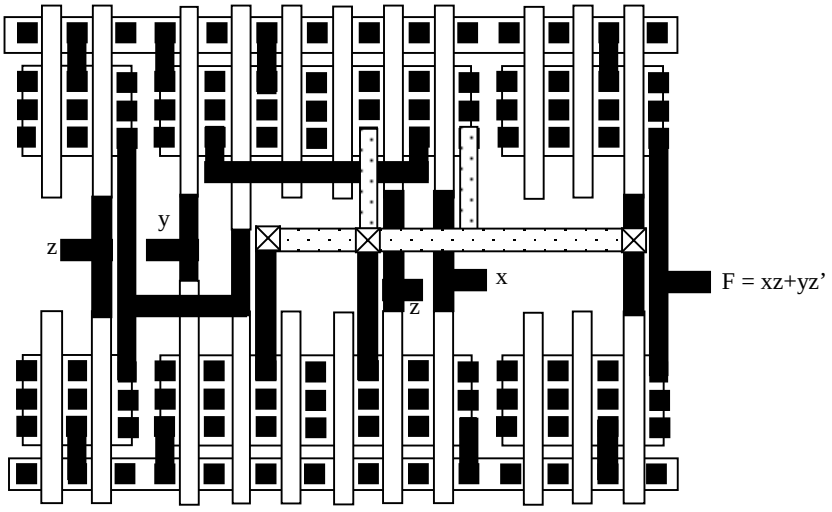


10.2 Draw (a) the transistor-level circuit schematic for a two-input multiplexor, and (b) the top-down view of the circuit on an IC (make your layout as compact as possible.)

The function for a two-input multiplexor with inputs x and y and select input z, is $F = xz+yz'$. Therefore, the answer for this question is identical to question 10.1.

10.3 Implement the function $F = xz + yz'$ using the gate array structure given in Figure 1.6 (a).

The gate array structure for the function $F = xz + yz'$



CHAPTER 11: *Design Technology*

11.1 List and describe three general approaches to improving designer productivity.

1. Automation: The task of using a computer program to replace manual design effort.

2. Reuse: The process of using pre-designed components rather than designing those components again.

3. Verification: The task of ensuring the correctness and completeness at each design step.

11.2 Describe each tool that has enabled the elevation of software design and hardware design to higher abstraction levels.

1. Assemblers and Linkers: Tools that automatically translate assembly instructions, consisting of instructions written using letters and numbers to represent symbols, into equivalent machine instructions.

2. Compilers: Compilers automatically translate sequential programs, written in high-level languages like C, into equivalent assembly instructions.

3. Logic Synthesis Tools: Automatically converts logic equations or FSM into logic gates.

4. RT Synthesis Tools: Automatically convert FSMD's into FSMs, logic equations, and pre-designed RT components like registers and adders.

5. Behavioral Synthesis Tools: Tools that convert sequential programs into FSMDs.

11.3 Show behavior and structure (at the same abstraction level) for a design that finds minimum of three input integers, by showing the following descriptions: a sequential program behavior, a processor/memory structure, a register–transfer behavior, a register/FU/MUX structure, a logic equation/FSM behavior, and finally a gate/flip-flop

structure. Label each description and associate each label with a point on Gajski's Y-chart.

NOTE: Answer will vary.

11.4 Develop an example of a Boolean function that can be implemented with fewer gates when implemented in more than two levels (your designs should have roughly 10 gates, not hundreds!). Assuming two transistors per gate input, and gate delay of 10 nanoseconds, create a single plot showing size versus delay for both designs.

NOTE: Answer will vary.

11.5 Show how to partition a single finite-state machine into two smaller finite-state machines, which might be necessary to achieve acceptable synthesis tool run time. *Hint*: you'll need to introduce a couple new signals and states.

NOTE: Answer will vary.

11.6 Define hardware/software codesign.

Hardware/software codesign is the joint consideration of general purpose and single-purpose processors by the same automatic tools.

11.7 Write a small program that reads a file of integers and outputs their sum. Write another program that does not add the integers using the built-in addition operator of a programming language, but instead "simulates" addition by using an addition function that converts each integer to a string of 0s and 1s, adds the strings by mimicking binary addition, and converts the binary result back to an integer. Compare the performance of the native program to the performance of the simulation program on a large file.

C++ program using built-in add:

```
#include <iostream>

int main()
{
    long a;
    long total = 0;

    while( cin >> a ) {
         total += a;
    }

    cout << endl << "total: " << total << endl;
    return 0;
}
```

C++ program using simulated add:

```
#include <iostream>

void add(long &total, long a)
{
    char total_b[33] = "00000000000000000000000000000000";
    char a_b[33]     = "00000000000000000000000000000000";
    char x, y, cin;
    int i;

    // conver to char string represntation
    for(i=0;i<32;i++) {
  if( total & (0x0001 << i) ) {
      total_b[i] = '1';
  }
  if( a & (0x0001 << i) ) {
      a_b[i] = '1';
  }
    }

    cin = '0';
    for(i=0; i<32; i++) {
  x = total_b[i];
  y = a_b[i];
  if( x == '0' && y == '0' && cin == '0' ) {
      total_b[i] = '0';
      cin = '0';
  }
  else if( x == '0' && y == '0' && cin == '1' ) {
      total_b[i] = '1';
      cin = '0';
  }
  else if( x == '0' && y == '1' && cin == '0' ) {
      total_b[i] = '1';
      cin = '0';
  }
  else if( x == '0' && y == '1' && cin == '1' ) {
      total_b[i] = '0';
      cin = '1';
  }
  else if( x == '1' && y == '0' && cin == '0' ) {
      total_b[i] = '1';
      cin = '0';
  }
  else if( x == '1' && y == '0' && cin == '1' ) {
      total_b[i] = '0';
      cin = '1';
  }
  else if( x == '1' && y == '1' && cin == '0' ) {
      total_b[i] = '0';
      cin = '1';
  }
  else if( x == '1' && y == '1' && cin == '1' ) {
      total_b[i] = '1';
```

```
      cin = '1';
  }
    }

    // convert back to integer representation
    total = 0;
    for(i=0;i<32;i++) {
  if( total_b[i] == '1' ) {
      total |= (0x0001 << i);
  }
    }
}

int main()
{
    long a;
    long total = 0;

    while( cin >> a ) {
  add(total, a);
    }

    cout << endl << "total: " << total << endl;
    return 0;
}
```

11.8 If a simulation environment can simulate 1,000 instructions per second, estimate how long the environment would take to simulate the boot sequence of Windows running on a modern PC. Even a very rough estimate is acceptable, since we are interested in the order of magnitude of such simulation.

A 200MHz Pentium processor required 16.75s to boot Windows ME.

200MHz = $2.00 * 10^8$ instruction/second
16.75 s = $3.35 * 10^9$ instructions
Simulation time = $3.35 * 10^6$ seconds

Thus, the simulation environment would require 930.56 hours to boot Windows ME.

NOTE: Answers will vary.

11.9 What is hardware/software co-simulation? What is a key method for speeding up such simulation?

Hardware/software co-simulation: Simulation that is designed to hide the details of the integration of an ISS and HDL simulator. In order to minimize the communication between the ISS and the HDL simulator, we can model memory in the both the ISS and the HDL simulator. Each simulator can use its own copy of the memory without communicating with other simulator most of the time.

11.10 Show the correspondence of the three types of cores with Gajski's Y-Chart.

Soft core: Soft cores relate to the behavior axis of the Y-chart.
Firm core: Firm cores relate to the structural axis of the Y-chart.
Hard core: Hard cores relate to the physical axis of the Y-chart.

11.11 Describe the new challenges created by cores for processor developers as well as users.

For core processors, pricing models and IP protection have become major challenges. Pricing for IP cores can be done using a fixed price or a royalty-based model. Each prcingin model that is employed comes with accompanying challenges of enforcing those models, and extensive licensing agreements must often be create. IP protection has become a challenge because cores are sold in an electronic format, so copying of the design has become much easier. For users of cores, they will face the challenges of dealing with licensing agreements, which often require legal assistance due to the complexity of them. In additions, users will have to spend extra design efforts, especially with soft cores. Soft core must be synthesized and tested and even minor differences in synthesis tools can lead to problems.

11.12 List major design steps for building the digital camera example of Chapter 7 assuming: (a) a waterfall process model, (b) a spiral-like Y-chart process model.

a.) Using the waterfall design process model, the major design steps for the digital camera would are as follows. First, the designer would create an extensive requirements document that would be used to create the behavioral model for the digital camera. This model would likely be create in high-level language such as C or C++. Once the algorithms of the behavioral model are completed, the designer would convert that description to a structural HDL description. Once the HDL description is completed, the designer's next task would be to synthesize that design down to gates and further verify it works correctly through gate-level simulation. Finally, the design could map the get-level design to an physical layout and test the layout using SPICE simulations. The final step would be to actually test the digital camera's prototype IC.

(b) In the spiral model, instead of describing the complete function of the digital camera in the beginning, the design will start by describing the basic operation of it. This basic operation may be to only provide the ability to capture, encode and decode a single image. With this basic requirement, the designer will spend time creating the behavioral model. Once the basic behavioral model is completed, it can be converted to a structural HDL implementation. Furthermore, the structural design will be converted to a get-level description and ultimately a physical prototype. This prototype will be used to test the functionality of the basic design and get a better understanding of the other features that should be added. For example, running the actual prototype will allow the designer to test the timing of

the image capture, encoding, and decoding. Using this information, the designer can modify the requirements of the design to meet the new timing requirements.

NOTE: Answer will vary depending on what the student chose as the major design steps.