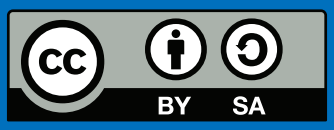
A Practical Introduction to Computer Architecture

Daniel Page ⟨dan@phoo.org⟩

git # 5ac601b @ 2018-07-13



EXAMPLE EXAM-STYLE QUESTIONS

1 Chapter 1

Q1. a For the sets $A = \{1, 2, 3\}$, $B = \{3, 4, 5\}$ and $\mathcal{U} = \{1, 2, 3, 4, 5, 6, 7, 8\}$, compute the following:

i $|A|$.
ii $A \cup B$.
iii $A \cap B$.
iv $A - B$.
v $\overline{A}$.
vi $\{x \mid 2 \cdot x \in \mathcal{U}\}$.

b For each of the following decimal integers, write down the 8-bit binary representation in sign-magnitude **and** two's-complement:

i $+0$.
ii -0.
iii $+72$.
iv -34.
v -8.
vi 240.

Q2. For some 32-bit integer x, explain what is meant by the Hamming weight of x; write a short C function to compute the Hamming weight of a given 32-bit input.

Q3. a Write out a truth table for the Boolean function

$$f(a, b, c) = (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge c) \vee (\neg a \wedge \neg b \wedge c),$$

then decide how many

i input combinations, and
ii outputs where $f(a, b, c) = 1$

exist in it.

b Consider the Boolean function

$$f(a, b, c, d) = \neg a \wedge b \wedge \neg c \wedge d.$$

Which of the following assignments

i $a = 0, b = 0, c = 0$ and $d = 1$,
ii $a = 0, b = 1, c = 0$ and $d = 1$,
iii $a = 1, b = 1, c = 1$ and $d = 1$,
iv $a = 0, b = 0, c = 1$ and $d = 0$.

produces the output $f(a, b, c, d) = 1$?

c Which of the following Boolean expressions

i $(a \vee b \vee d) \wedge (\neg c \vee d)$,
ii $(a \wedge b \wedge d) \vee (\neg c \wedge d)$,
iii $(a \vee b \vee d) \vee (\neg c \vee d)$.

is in Sum-of-Products (SoP) standard form?

d Identify **each** equivalence that is correct:

i $a \vee 1 \equiv a$.
ii $a \oplus 1 \equiv \neg a$.
iii $a \wedge 1 \equiv a$.
iv $\neg(a \wedge b) \equiv \neg a \vee \neg b$.

e Identify **each** equivalence that is correct:

i $\neg\neg a \equiv a$.
ii $\neg(a \wedge b) \equiv \neg a \vee \neg b$.
iii $\neg a \wedge b \equiv a \wedge \neg b$.
iv $\neg a \equiv a \oplus a$.

Q4. a The OR form of the null axiom is $x \vee 1 \equiv 1$. Which of the following options

i $x \wedge 1 \equiv 1$,
ii $x \wedge 0 \equiv 0$,
iii $x \vee 0 \equiv 0$,
iv $x \wedge x \equiv x$,

is the dual of this axiom?

b Given the Boolean equation

$$f = \neg a \wedge \neg b \vee \neg c \vee \neg d \vee \neg e,$$

which of the following

i $\neg f = a \vee b \vee c \vee d \vee e$,
ii $\neg f = a \wedge b \wedge c \wedge d \wedge e$,
iii $\neg f = a \wedge b \wedge (c \vee d \vee e)$,
iv $\neg f = a \wedge b \vee \neg c \vee \neg d \vee \neg e$,
v $\neg f = (a \vee b) \wedge c \wedge d \wedge e$

is correct?

c If we write the de Morgan axiom in English, which of the following

i NOR is equivalent to AND if each input to AND is complemented,
ii NAND is equivalent to OR if each input to OR is complemented,
iii AND is equivalent to NOR if each input to NOR is complemented, or
iv NOR is equivalent to NAND if each input to NAND is complemented.

describes the correct equivalence?

Q5. a Identify which **one** of these Boolean expressions

i $c \vee d \vee e$
ii $\neg c \wedge \neg d \wedge \neg e$
iii $\neg a \wedge \neg b$
iv $\neg a \wedge \neg b \wedge \neg c \wedge \neg d \wedge \neg e$

is the correct result of simplifying

$$(\neg(a \vee b) \wedge \neg(c \vee d \vee e)) \vee \neg(a \vee b).$$

b If you simplify the Boolean expression

$$(a \vee b \vee c) \wedge \neg(d \vee e) \vee (a \vee b \vee c) \wedge (d \vee e)$$

into a form that contains the fewest operators possible, which of the following options

i $a \vee b \vee c$,
ii $\neg a \wedge \neg b \wedge \neg c$,
iii $d \vee e$,
iv $\neg d \wedge \neg e$,
v none of the above

do you end up with and why?

c If you simplify the Boolean expression

$$a \wedge c \vee c \wedge (\neg a \vee a \wedge b)$$

into a form that contains the fewest operators possible, which of the following options

i $(b \wedge c) \vee c$,
ii $c \vee (a \wedge b \wedge c)$,
iii $a \wedge c$,
iv $a \vee (b \wedge c)$,
v none of the above

do you end up with and why?

d Consider the Boolean expression

$$a \wedge b \vee a \wedge b \wedge c \vee a \wedge b \wedge c \wedge d \vee a \wedge b \wedge c \wedge d \wedge e \vee a \wedge b \wedge c \wedge d \wedge e \wedge f.$$

Which of the following simplifications

i $a \wedge b \wedge c \wedge d \wedge e \wedge f$,
ii $a \wedge b \vee c \wedge d \vee e \wedge f$,
iii $a \vee b \vee c \vee d \vee e \vee f$,
iv $a \wedge b$,
v $c \wedge d$,
vi $e \wedge f$,
vii $a \vee b \wedge (c \vee d \wedge (e \vee f))$
viii $((a \vee b) \wedge c) \vee d \wedge e \vee f$

is correct?

e Given the options

i 1,
ii 2,

iii 3,

iv 4,

decide which is the least number of operator required to compute the same result as

$$f(a,b,c) = (a \wedge b) \vee a \wedge (a \vee c) \vee b \wedge (a \vee c).$$

Show how you arrived at your decision.

f Prove that

$$(\neg x \wedge y) \vee (\neg y \wedge x) \vee (\neg x \wedge \neg y) \equiv \neg x \vee \neg y.$$

g Prove that

$$(x \wedge y) \vee (y \wedge z \wedge (y \vee z)) \equiv y \wedge (x \vee z).$$

2 Chapter 2

Q6. From the following list

A: $(x \wedge y) \oplus z$
B: $(\neg x \vee y) \oplus z$
C: $(x \vee \neg y) \oplus z$
D: $\neg(x \vee y) \oplus z$
E: $\neg\neg(x \vee y) \oplus z$

identify **each** Boolean expression that evaluates to 1 given the assignment $x = 0$, $y = 0$ and $z = 1$.

Q7. A given set of Boolean operators may be termed functionally complete (or universal): this means *any* Boolean function can be expressed using a Boolean expression involving elements of the set alone. For example, because we know the NAND operator is functionally complete, we can also term the sets $\{ \overline{\wedge} \}$ and $\{\wedge, \neg\}$ functionally complete. Noting that $\not\equiv$ and $\not\Rightarrow$ denote the inverse of equivalence and implication respectively (i.e., not equivalent, and does not imply), which of the following sets

A: $\{\vee, \oplus\}$
B: $\{\Rightarrow, \not\equiv\}$
C: $\{\Rightarrow, \not\Rightarrow\}$
D: all of the above
E: none of the above

is/are functionally complete?

Q8. **One** of the following equivalences

A: $(x \wedge y) \wedge z \equiv x \wedge (y \wedge z)$
B: $x \vee 1 \equiv x$
C: $x \vee \neg x \equiv 1$
D: $\neg(x \vee y) \equiv \neg x \wedge \neg y$
E: $\neg\neg x \equiv x$

is incorrect: identify which.

Q9. The Boolean expression

$$(x \vee (z \vee y)) \wedge \neg(\neg y \wedge \neg z)$$

is equivalent to which of the following alternatives?

A: $y \vee z$
B: $((x \vee z) \vee y)) \wedge (x \vee z)$
C: $(x \wedge y) \vee (x \wedge z)$
D: $(x \vee y) \wedge \neg(x \vee z)$
E: $(x \wedge z) \vee (x \wedge y)$

Q10. The Boolean expression

$$(x \vee y) \vee (x \wedge z)$$

is equivalent to which of the following alternatives?

`A:` $(x \vee y) \wedge (x \vee z)$
`B:` $(x \vee y) \wedge z$
`C:` $(x \vee y) \wedge (x \wedge z)$
`D:` $x \vee y$
`E:` $(x \wedge y) \vee x$

Q11. How many n-input, 1-output Boolean functions are there?

`A:` 1
`B:` n
`C:` 2^n
`D:` 2^{2^n}
`E:` $2^{2^{2^n}}$

Q12. We studied representation of unsigned integers using a base-b positional number system. Which of the following literals

`A:` 10101
`B:` 11111
`C:` 11120
`D:` 12200
`E:` 12345

represents the unsigned decimal integer $123_{(10)}$ in base-3 (or ternary, digits in which are termed trits).

Q13. Imagine that two signed, 8-bit integers x and y are represented using two's-complement and sign-magnitude respectively, and both of which have the decimal value $51_{(10)}$. If the most-significant bit of both x and y is set to 1, what are their new (decimal) values?

`A:` $-77_{(10)}$ and $179_{(10)}$
`B:` $-77_{(10)}$ and $-51_{(10)}$
`C:` $-51_{(10)}$ and $-77_{(10)}$
`D:` $179_{(10)}$ and $179_{(10)}$
`E:` $179_{(10)}$ and $-51_{(10)}$

Q14. Imagine that two signed, 16-bit integers x and y are represented using two's-complement; their product $r = x \cdot y$ is a signed, 32-bit integer also represented using two's-complement. What is the largest (i.e., whose magnitude is greatest) negative value of r possible?

`A:` -0
`B:` -32768
`C:` -65535
`D:` -1073709056
`E:` -2147483648

Q15. Imagine you write a C program that defines signed, 16-bit integer variables `x` and `y` (of type `short`) and then assigns them the decimal values $256_{(10)}$ and $4852_{(10)}$ respectively. If `x` and `y` are then cast into signed, 8-bit integers (of type `char`), which of the following

`A:` 0 and 12
`B:` 0 and -12
`C:` -1 and 256
`D:` -1 and -52
`E:` 0 and 52

identifies their decimal value? Or, put another way, which are the result of evaluating the two expressions `( char )( x )` and `( char )( y )`?

Q16. Consider two signed, 8-bit integer variables `x` and `r` (of type `char`) used in a C program. If `x` has the decimal value $9_{(10)}$ and an assignment

```
r = ( ~x << 4 ) | 0x97
```

is executed, what is the decimal value of `r` afterwards?

`A:` $-9_{(10)}$
`B:` $-1_{(10)}$
`C:` $0_{(10)}$
`D:` $1_{(10)}$
`E:` $9_{(10)}$

Q17. In general, some x is a fixed point of a function f if $f(x)$ equals x, i.e., if f maps x to itself. Consider the following function

```
int8_t abs( int8_t x ) {
  int8_t r;

  if( x >= 0 ) {
    r =  x;
  }
  else {
    r = -x;
  }

  return r;
}
```

implemented in C: `abs` was written in an attempt to compute the absolute value of `x`, a signed, 8-bit integer representing using two's-complement. How many of the $2^8 = 256$ possible values of `x` are fixed points of `abs`?

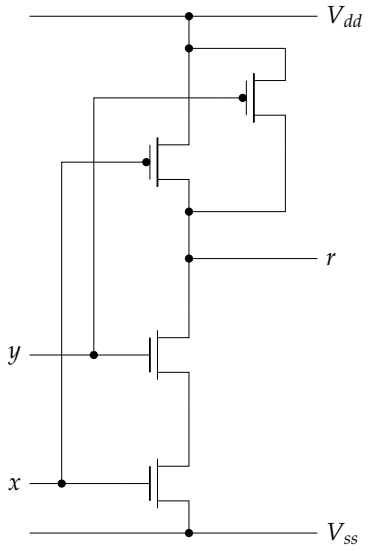
`A:` 0
`B:` 127
`C:` 128
`D:` 129
`E:` 256

Q18. From the following list

`A:` has N-type semiconductor terminals and P-type body
`B:` has P-type semiconductor terminals and N-type body
`C:` is paired with another N-MOSFET to form a CMOS cell
`D:` has a threshold voltage above which the transistor is deemed active

identify **each** statement that correctly describes an N-MOSFET.

Q19. Consider the following implementation of a 2-input NAND gate:

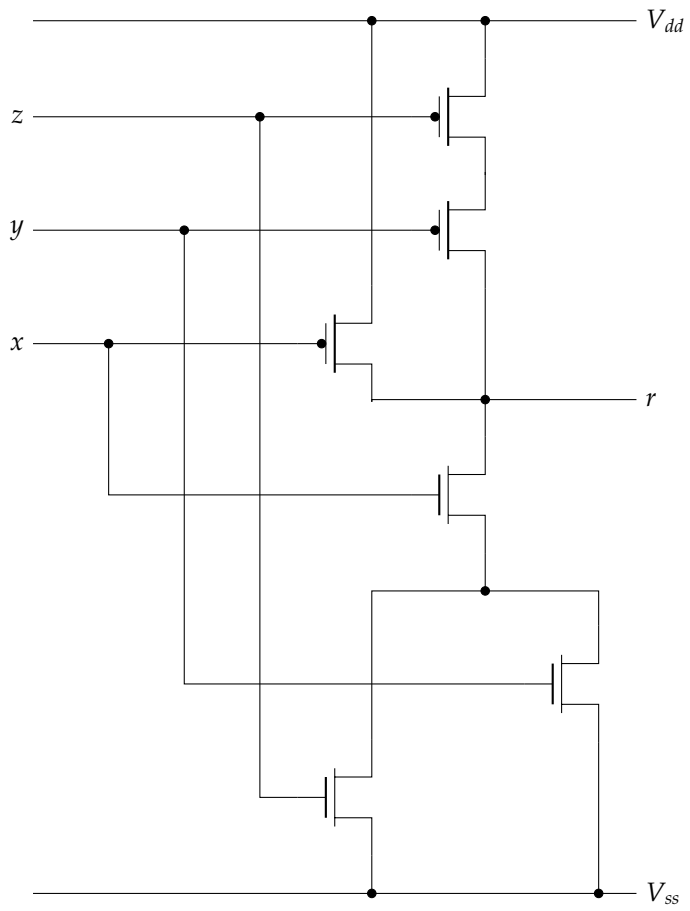


From the following list

`A:` two inputs x and y, and one output r
`B:` a pull-up network of P-MOSFET transistors
`C:` a pull-down network of BJT transistors
`D:` two power rails supplying different voltage levels
`E:` a flux capacitor

identify **each** component evident in the implementation?

Q20. Consider the following organisation of MOSFET transistors



which implements a 3-input Boolean function $r = f(x, y, z)$. *Which* function, from the following, do you think it matches?

`A:` $r = x \wedge y \wedge z$
`B:` $r = x$
`C:` $r = \neg(x \wedge (y \vee z))$
`D:` $r = x \wedge (y \vee z)$
`E:` $r = x \vee y \vee z$

Q21. Recall that a 2-input XOR operator can be described via the following truth table:

XOR		
x	y	r
0	0	0
0	1	1
1	0	1
1	1	0

An implementation of this operator is realised by combining logic gate instances, e.g., for NOT, NAND, AND, NOR, and OR, while attempting to minimise the total number of underlying MOSFET-based transistors. How many such transistors do you think it uses?

A: 14
B: 16
C: 18
D: 20
E: 22

Q22. A buffer can be described as a "pass through" logic gate: although it performs no computation (i.e., the output r matches the input x, so $r = x$), it does impose a delay (often roughly the same as a NOT gate). It may be termed a non-inverting buffer (cf. an *inverting* buffer, or NOT gate) because of this.

You are asked to implement a buffer, using an unconstrained organisation of N- and P-MOSFET transistors alone. Assuming you attempt to minimise the number used, how many transistors do you need?

A: 0
B: 2
C: 4
D: 6
E: 8

Q23. Consider a 16-bit register, constructed from CMOS-based D-type latches. Based on high-level reasoning about this component alone, if the initial value stored is $DEAD_{(16)}$ then overwriting it with which of the following

A: $BEEF_{(16)}$
B: $F00D_{(16)}$
C: $1234_{(16)}$
D: $FFFF_{(16)}$
E: $0000_{(16)}$

might you expect to consume more power?

Q24. Recalling that **?** denotes don't-care, the following truth table

f			
x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	**?**
1	1	1	1

describes a 3-input, 1-output Boolean function f st. $r = f(x, y, z)$. Which of the following Boolean expressions

A: $(\neg x \oplus \neg y) \wedge z$
B: $(\neg x \oplus \neg y) \vee z$
C: $(\neg x \wedge \neg y) \wedge z$
D: $(\neg x \wedge \neg y) \vee z$
E: $(\neg x \vee \neg y) \wedge z$

correctly realises f?

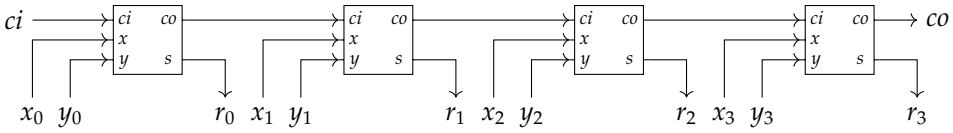
Q25. A m-output, 1-bit demultiplexer connects a 1-bit input x to one of m separate 1-bit outputs (say r_i for $0 \leq i < m$). The output is selected using an l-bit control signal c (or, equivalently, c is a collection of l separate 1-bit control signals). If $m = 5$, what value of l is required?

A: 0
B: 1
C: 2
D: 3
E: 4

Q26. Imagine you want to design an 8-input, 8-bit multiplexer. Rather than do so from scratch, you intend to form the design using multiple instances of an existing 2-input, 1-bit multiplexer component. How many do you need?

A: 1
B: 8
C: 24
D: 40
E: 56

Q27. The following diagram



illustrates a 4-bit ripple-carry adder circuit, constructed using 4 full-adder instances: it computes the sum $r = x + y + ci$, given two operands x and y and a carry-in ci, and an associated carry-out co. Given the propagation delay of NOT, AND, OR and XOR gates is 10ns, 20ns, 20ns and 60ns respectively, which of the following

A: 120ns
B: 180ns
C: 240ns
D: 280ns
E: 480ns

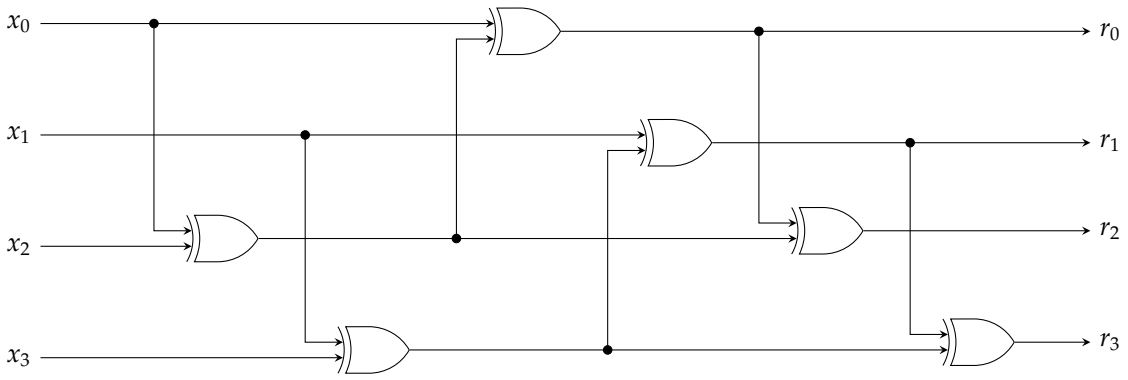
most accurately reflects the critical path of the entire circuit?

Q28. Imagine you use the ripple-carry adder in the previous question to compute an unsigned addition within some larger circuit. Having seen your design, your friend suggests they can optimise it: they claim that replacing each full-adder instance with a half-adder instance will halve the total number of logic gates required. However, they admit the optimisation does have a disadvantage. Specifically, although any value of x can be accommodated the optimised circuit can only produce the correct output for *some* values of y. Which of the following values of y

A: -1
B: 0
C: 1
D: any $2 \leq y < 8$
E: any $8 \leq y < 16$

will produce the correct output?

Q29. Consider the following combinatorial circuit



with a 4-bit input x and a 4-bit output r. Which of the following best describes the purpose of this circuit?

A: it computes the Hamming weight of x

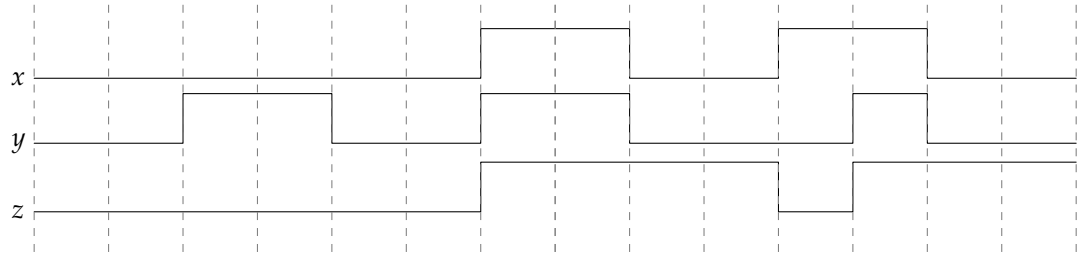
`B:` it computes the parity of x
`C:` it swaps the most-significant 2-bit half of x with the least-significant 2-bit half of x
`D:` it adds the most-significant 2-bit half of x to the least-significant 2-bit half of x (treating it as an unsigned, 4-bit integer)
`E:` it negates x (treating it as a signed, 4-bit integer represented using two's-complement)

Q30. From the following list

`A:` the design of a DRAM cell includes more transistors than an SRAM cell
`B:` an SRAM cell can store more information than a DRAM cell
`C:` SRAM cells can be accessed more quickly than DRAM cells
`D:` DRAM cells require a mechanism to refresh their content

identify each statement that correctly describes SRAM and DRAM cells.

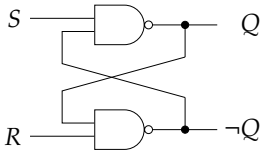
Q31. Consider the following waveform



which details the behaviour of three signals labelled x, y and z. Which of the following components *could* the behaviour illustrated relate to?

`A:` an SR-type flip-flop
`B:` an SR-type latch
`C:` a D-type flip-flop
`D:` a D-type latch
`E:` a T-type flip-flop

Q32. The following diagram



illustrates a preliminary NAND-based SR-latch design, in the sense it currently lacks an enable signal. If Q and Q' denote the current and next state respectively, which of the following excitation tables

			Current		*Next*	
	S	R	Q	$\neg Q$	Q'	$\neg Q'$
`A:`	0	0	0	1	0	1
	0	0	1	0	1	0
	0	1	?	?	0	1
	1	0	?	?	1	0
	1	1	?	?	0	0
`B:`	0	0	?	?	1	1
	0	1	?	?	1	0
	1	0	?	?	0	1
	1	1	0	1	0	1
	1	1	1	0	1	0
`C:`	0	0	?	?	0	1
	1	1	?	?	1	0
`D:`	0	?	?	?	0	1
	1	?	?	?	1	0
`E:`	?	0	?	?	0	1
	?	1	?	?	1	0

correctly captures the behaviour of this circuit?

Q33. Consider a DRAM memory device with a capacity of 65536 addressable bytes. Of the following options

`A:` 8 address pins, 65536 cells
`B:` 16 address pins, 65536 cells
`C:` 8 address pins, 524288 cells
`D:` 16 address pins, 524288 cells

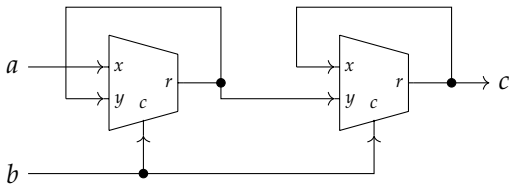
which offers the most likely description of said device?

Q34. Figure 1 illustrates the design of a DRAM memory. The labels on four components in the block diagram have been blanked-out, then replaced with the symbols α, β, γ and δ: which of the following mappings

`A:`	α	$\mapsto$	row address buffer
	β	$\mapsto$	row address decoder
	γ	$\mapsto$	column address buffer
	δ	$\mapsto$	column address decoder
`B:`	α	$\mapsto$	row address buffer
	β	$\mapsto$	column address buffer
	γ	$\mapsto$	row address decoder
	δ	$\mapsto$	column address decoder
`C:`	α	$\mapsto$	column address buffer
	β	$\mapsto$	row address buffer
	γ	$\mapsto$	column address decoder
	δ	$\mapsto$	row address decoder
`D:`	α	$\mapsto$	row address decoder
	β	$\mapsto$	column address decoder
	γ	$\mapsto$	row address buffer
	δ	$\mapsto$	column address buffer
`E:`	α	$\mapsto$	column address decoder
	β	$\mapsto$	row address decoder
	γ	$\mapsto$	column address buffer
	δ	$\mapsto$	row address buffer

do you think is correct?

Q35. Although perhaps unusual, the following diagram

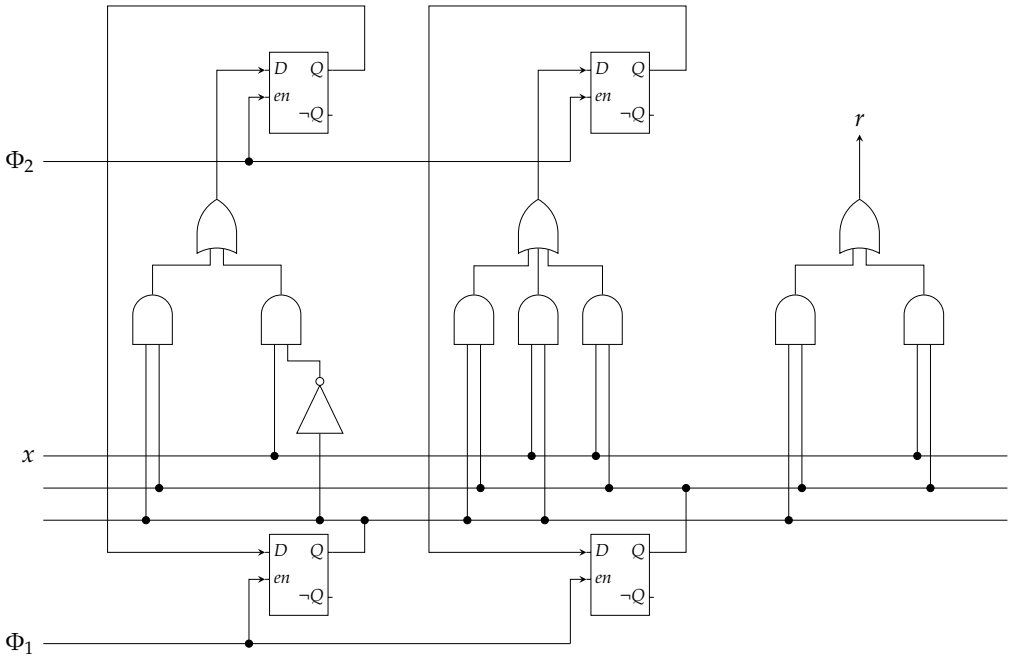


illustrates a circuit with well defined behaviour. Based on analysis of this behaviour, which of the following components

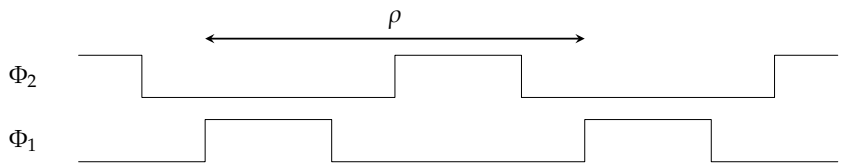
`A:` a flip-flop
`B:` a latch
`C:` a RAM cell
`D:` a ROM cell
`E:` a clock multiplier

does the circuit implement?

All questions in this Section relate to a Finite State Machine (FSM) whose concrete implementation is as follows:



Notice that the implementation is based on use of four D-type latches, and a 2-phase clock supplied via Φ_1 and Φ_2; one additional input x plus one output r are also evident. To function correctly, a clock generator ensures Φ_1 and Φ_2 are driven as follows:



Q36. From the following list

`A:` Φ_1 and Φ_2 are digital signals
`B:` Φ_1 and Φ_2 are non-overlapping
`C:` Φ_1 and Φ_2 can be gated
`D:` Φ_1 and Φ_2 are unskewed
`E:` Φ_1 and Φ_2 each have a duty cycle of 33%

identify **each** property the clock generator *must* guarantee is true for the implementation to function correctly.

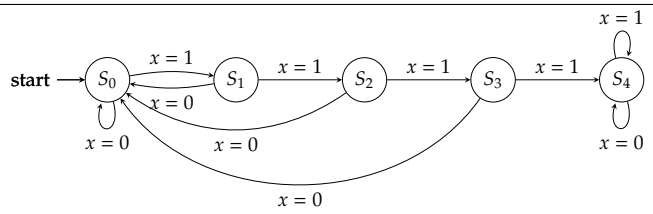
Q37. Consider the two D-type latches at the bottom of the diagram, which form a 2-bit register. Imagine the value stored in this register is expressed as a 2-bit integer: when the implementation is initially powered-on, is this value equal to

`A:` $00_{(2)}$
`B:` $01_{(2)}$
`C:` $10_{(2)}$
`D:` $11_{(2)}$
`E:` any of the above

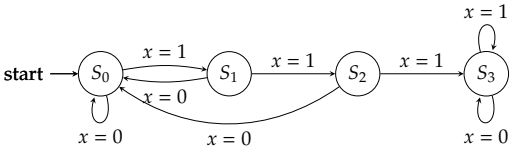
Q38. Any FSM specification will include a transition function, often denoted δ, which can be described in either

tabular or diagrammatic form. Of the following options

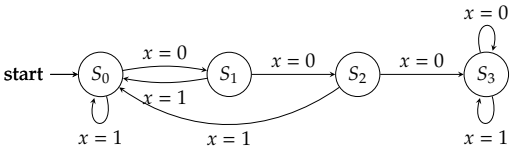
A :



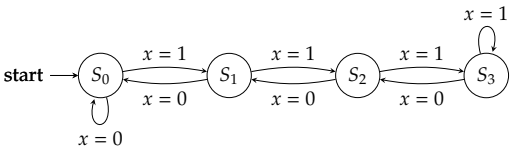
B :



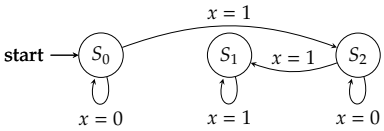
C :



D :



E :



which captures the transition function of this FSM?

Q39. Which of the following FSM types, namely

`A:` Mealy
`B:` Moore

does this implementation represent?

Q40. In the 2-phase clock waveform above, ρ illustrates the clock period: recall this is inversely proportional to the clock frequency. Imagine the gate delay for NOT, AND, and 2- and 3-input OR gates are 10ns 20ns 20ns, 30ns respectively, and the critical path associated for a D-type latch is 60ns. Which of the following best matches the maximum possible clock frequency of this implementation?

`A:` 1.0kHz
`B:` 5.9MHz
`C:` 9.0MHz
`D:` 9.5MHz
`E:` 1.0GHz

Q41. Which of the following best describes the purpose of this FSM?

`A:` set $r = 1$ iff. the current value of x is different from the previous value of x,
`B:` act as a modulo 4 counter that is incremented by the value of x, and set $r = 1$ the current counter value is zero,
`C:` compute the Hamming weight of a sequence fed as input bit-by-bit via x, and set $r = 1$ once this is equal to 3

`D:` count the number of consecutive times $x = 1$, and set $r = 1$ once this is equal to 3
`E:` inspect the sequence fed as input bit-by-bit via x, and set $r = 1$ iff. this sequence, when interpreted as an unsigned integer, is odd

Q42. a Write the simplest (i.e., with **fewest** operators) possible Boolean expression that implements the Boolean function

$$r = f(x, y, z)$$

described by

f			
x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	**?**
1	0	0	1
1	0	1	0
1	1	0	**?**
1	1	1	1

where **?** denotes don't care.

b Take the Boolean expression

$$\neg(x \vee y)$$

and draw a gate-level **circuit diagram** that computes an equivalent resulting using only 2-input NAND gates.

c Recall that an SR latch has two inputs S (or set) and R (or reset); if $S = R = 1$, the two outputs Q and $\neg Q$ are undefined. This issue can be resolved by using a reset-dominate latch: the alternative design has the same inputs and outputs, but resets the latch (i.e., has $Q = 0$ and $\neg Q = 1$) whenever $S = R = 1$.

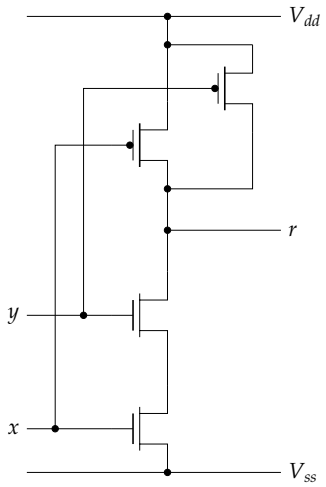
Using a gate-level **circuit diagram**, describe how a reset-dominate latch can be implemented using **only** NOR gates and at most **one** AND gate.

d The quality of the design for some hardware component is often judged by measuring efficiency, for example how quickly it can produce output on average. Name **two** other metrics that might be considered.

Q43. a Describe how N-type and P-type MOSFET transistors are constructed using silicon and how they operate as switches.

b Draw a diagram to show how N-type and P-type MOSFET transistors can be used to implement a NAND gate. Show your design works by describing the transistor states for each input combination.

Q44. The following diagram



details a 2-input NAND gate comprised of two P-MOSFET transistors (top) and two N-MOSFET transistors (bottom). Draw a similar diagram for a 3-input NAND gate.

Q45. Moore's Law predicts the number of CMOS-based transistors we can manufacture within a fixed sized area will double roughly every two years; this is often *interpreted* as doubling computational efficiency over the same period. Briefly explain **two** limits which mean this trend cannot be sustained indefinitely.

Q46. Given that **?** is the don't care state, consider the following truth table which describes a function p with four inputs (a, b, c and d) and two outputs (e and f):

p					
a	b	c	d	e	f
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	**?**	**?**
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	**?**	**?**
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	**?**	**?**
1	1	0	0	**?**	**?**
1	1	0	1	**?**	**?**
1	1	1	0	**?**	**?**
1	1	1	1	**?**	**?**

a From the truth table above, write down the corresponding Sum of Products (SoP) equations for e and f.

b Simplify the two SoP equations so that they use the minimum number of logic gates possible. You can assume the two equations can share logic.

Q47. Using a Karnaugh map, derive a Boolean expression for the function

$$r = f(x, y, z)$$

described by the truth table

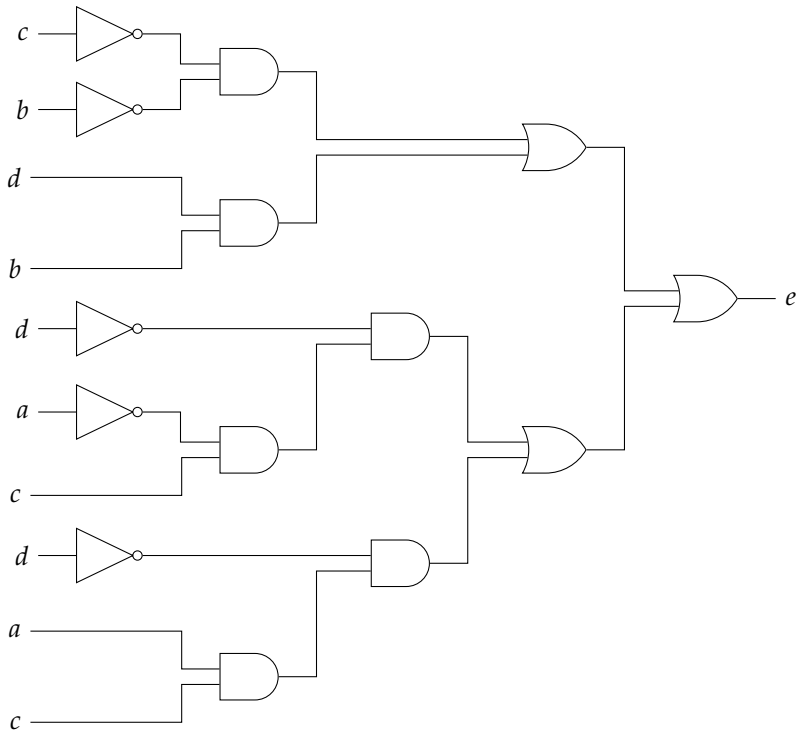
f			
x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	**?**

where **?** denotes don't care.

Q48. NAND is a universal logic gate in the sense that the behaviour of NOT, AND and OR gates can be implemented using only NAND. Show how this is possible using a truth table to demonstrate your solution.

Q49. Both NAND and NOR gates are described as universal because *any* other Boolean gate (i.e., AND, OR, NOT) can be constructed using them. Imagine your friend suggests a 4-input, 1-bit multiplexer (that selects between four 1-bit inputs using two 1-bit control signals to produce a 1-bit output) is also universal: state whether or not you believe them, and explain why.

Q50. Consider the following circuit where the propagation delay of logic gates in the circuit are 10ns for NOT, 20ns for AND, 20ns for OR and 60ns for XOR:



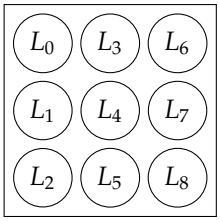
a Draw a Karnaugh map for this circuit and derive a Sum of Products (SoP) expression for the result.

b Describe advantages and disadvantages of your SoP expression and the dynamic behaviour it produces.

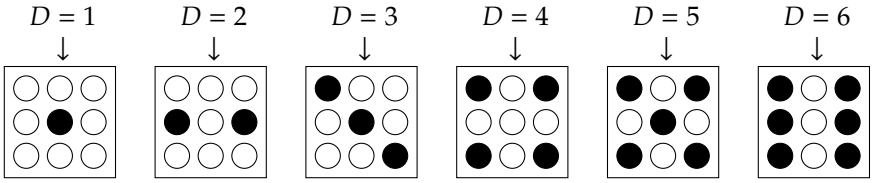
c If the circuit is used as combinatorial logic within a clocked system, what is the maximum clock speed of the system?

Q51. A game uses nine LEDs to display the result of rolling a six-sided dice; the i-th LED, say L_i for $0 \leq i < 9$, is driven with 1 or 0 to turn it on or off respectively. A 3-bit register D represents the dice as an unsigned integer.

a The LEDs are arranged as follows,



and the required mapping between dice and LEDs, given a filled dot means an LED is on, is



Using Karnaugh maps as appropriate, write a simplified Boolean expression for **each** LED (i.e., for **each** L_i in terms of D).

b The 2-input XOR, AND, OR and NOT gates used to implement your expressions have propagation delays of 40, 20, 20 and 10 nanoseconds respectively. Calculate how many times per-second the dice can be rolled, i.e., D can be updated, if the LEDs are to provide the correct output.

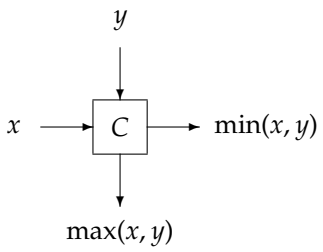
c The results of individual dice throws will be summed using a ripple-carry adder circuit, to give a total; each 3-bit output D will be added to and stored in an n-bit accumulator register A.

i Using a high-level **block diagram**, show how an n-bit ripple-carry adder circuit is constructed from full-adder cells.

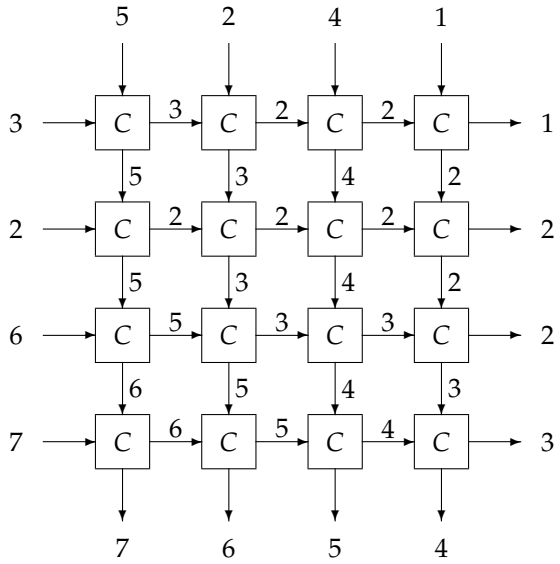
ii If $m = 8$ throws of the dice are to be summed, what value for n should be selected?

iii Imagine that instead of D, we want to add $2 \cdot D$ to A. Doubling D can be achieved by computing either $D + D$ or $D \ll 1$ (i.e., a left-shift of D by 1 bit). Carefully state which method is preferable, and why.

Q52. Consider a simple component called C that compares two inputs x and y (both are unsigned 8-bit integers) in order to produce their maximum and minimum as two outputs:



Instances of C can be connected in a mesh to sort integers: the input is fed into the top and left-hand edges of the mesh, the sorted output appears on the bottom and right-hand edges. An example is given below:



a Using standard building blocks (e.g., adder, multiplexer etc.) rather than individual logic gates, draw a **block diagram** that implements the component C.

b Imagine that an $n \times n$ mesh of components is created. Based on your design for C and clearly stating any **assumptions** you need to make, write down an expression for the critical path of such a mesh.

c Algorithms for sorting integers can clearly be implemented on a general-purpose processor. Explain **two** advantages and **two** disadvantages of using such a processor versus using a mesh like that above.

Q53. Imagine you are working for a company developing the "Pee", a portable games console. The user interface is a fancy controller that has

- three fire buttons represented by the 1-bit inputs F_0, F_1 and F_2, and
- a 8-direction D-pad represented by the 3-bit input D

and you are charged with designing some aspects of it.

a The fire button inputs are described as level triggered and active high; explain what this means (in comparison to the alternatives in each case).

b Some customers want an "autofire" feature that will automatically and repeatedly press the F_0 fire button for them. The autofire can operate in four modes, selected by a switch called M: off (where the fire button F_0 works as normal), slow, fast or very fast (where the fire button F_0 is turned on and off repeatedly at the selected speed). Stating any assumptions and showing your working where appropriate, design a circuit that implements such a feature.

c In an attempt to prevent counterfeiting, each controller can only be used with the console it was sold with. This protocol is used:

$\mathcal{P}$		C
$c \xleftarrow{\$} \{0,1\}^3$		
	$\xrightarrow{c}$	
		$r = T(c)$
	$\xleftarrow{r}$	
$r \stackrel{?}{=} T(c)$		

which, in words, means that

- the console generates a random 3-bit number c and sends it to the controller,
- the controller computes a 3-bit result $r = T(c)$ and sends it to the console,
- the console checks that r matches $T(c)$ and assumes the controller is valid if so.

i There is some debate as to whether the protocol should be synchronous or asynchronous; explain what your recommendation would be and why.

ii The function T is simply a look-up table. For example

$$T(x) = \begin{cases} 2 & \text{if } x = 0 \\ 6 & \text{if } x = 1 \\ 7 & \text{if } x = 2 \\ 1 & \text{if } x = 3 \\ 4 & \text{if } x = 4 \\ 0 & \text{if } x = 5 \\ 5 & \text{if } x = 6 \\ 3 & \text{if } x = 7 \end{cases}$$

Each pair of console and controller has such a T fixed inside them during the manufacturing process. Stating any assumptions and showing your working where appropriate, explain how **this** T might be implemented as a circuit.

Q54. Imagine you have three Boolean values x, y, and z. Given access to **as many** AND and OR gates as you want but **only two** NOT gates, write a set of Boolean expressions to compute all three results $\neg x$, $\neg y$ and $\neg z$.

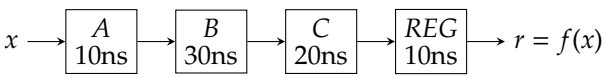
Q55. SAT is the problem of finding an assignment to n Boolean variables which means a given Boolean expression is satisfied, i.e., evaluates to 1. For example, given $n = 3$ and the expression

$$(x \wedge y) \vee \neg z,$$

$x = 1, y = 1, z = 0$ is one assignment (amongst several) which solves the associated SAT problem.

The ability to solve SAT can be used to test whether or not two n-input, 1-output combinatorial circuits C_1 and C_2 are equivalent. Show how this is possible.

Q56. Consider the following combinatorial circuit, which is the composition of four parts (labelled A, B, C and REG): each part is annotated with a name and an associated critical path. The circuit computes an output $r = f(x)$ from the corresponding input x.

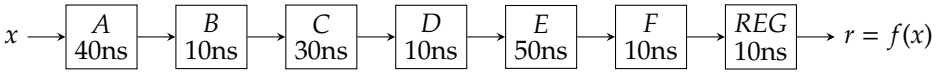


With respect to this circuit,

a first define the terms latency and throughput, then

b explain how and why you would expect use of pipelining to influence both metrics.

Q57. The figure below shows a block of combinatorial logic built from seven parts; the name and latency of each part is displayed inside it. Note that the last part is a register which stores the result:



It is proposed to pipeline the block of logic using two stages such that there is a pipeline register in between parts D and E:

a Explain the terms latency and throughput in relation to the idea of pipelining.

b Calculate the overall latency and throughput of the initial circuit described above.

c Calculate the overall latency and throughput of the circuit after the proposed change.

d Calculate the number of extra pipeline registers required to maximise the circuit throughput; state this new throughput and the associated latency. Explain the advantages and disadvantages of this change.

This is a (large) set of example Boolean minimisation questions: each asks you to transform some truth table describing an n-input Boolean function into a Boolean expression. Each solution includes

1. a reference implementation (produced by forming a SoP expression with a full term for *each* minterm, i.e., row where $r = 1$), and
2. a Karnaugh map annotated with sensible groups, and an optimised implementation based on those groups.

The goal is to focus on producing the latter, since the former is somewhat easier. Keep in mind and take care wrt. the following:

- There are 2^{2^n} Boolean functions with n inputs (or 3^{2^n} if you include don't-care as a valid output); whereas for small n a complete set of functions is included, but for large n there is only a random sub-set.
- No real effort is made to order the questions, and only minor effort to avoid duplicates. That said, there should be no trivial (in the sense $r = 1$ or $r = 0$ for all inputs, e.g., tautological) cases.
- The questions and solutions are generated automatically, meaning a small but real chance of bugs in the associated implementation!

2-input problems, without don't-care outputs

Q58.

y	z	r
0	0	1
0	1	0
1	0	1
1	1	1

Q59.

y	z	r
0	0	1
0	1	1
1	0	0
1	1	1

Q60.

y	z	r
0	0	1
0	1	0
1	0	0
1	1	0

Q61.

y	z	r
0	0	0
0	1	1
1	0	1
1	1	0

Q62.

y	z	r
0	0	1
0	1	0
1	0	1
1	1	0

Q63.

y	z	r
0	0	0
0	1	0
1	0	0
1	1	1

Q64.

y	z	r
0	0	0
0	1	0
1	0	1
1	1	1

Q65.

y	z	r
0	0	1
0	1	0
1	0	0
1	1	1

Q66.

y	z	r
0	0	0
0	1	1
1	0	0
1	1	0

Q67.

y	z	r
0	0	0
0	1	0
1	0	1
1	1	0

Q68.

y	z	r
0	0	0
0	1	1
1	0	0
1	1	1

Q69.

y	z	r
0	0	1
0	1	1
1	0	1
1	1	0

Q70.

y	z	r
0	0	0
0	1	1
1	0	1
1	1	1

Q71.

y	z	r
0	0	1
0	1	1
1	0	0
1	1	0

2-input problems, with don't-care outputs

Q72.

y	z	r
0	0	0
0	1	**?**
1	0	0
1	1	1

Q73.

y	z	r
0	0	0
0	1	1
1	0	1
1	1	**?**

Q74.

y	z	r
0	0	1
0	1	**?**
1	0	**?**
1	1	0

Q75.

y	z	r
0	0	0
0	1	**?**
1	0	**?**
1	1	1

Q76.

y	z	r
0	0	0
0	1	1
1	0	**?**
1	1	1

Q77.

y	z	r
0	0	1
0	1	**?**
1	0	1
1	1	0

Q78.

y	z	r
0	0	1
0	1	0
1	0	0
1	1	**?**

Q79.

y	z	r
0	0	**?**
0	1	1
1	0	0
1	1	1

Q80.

y	z	r
0	0	0
0	1	0
1	0	1
1	1	**?**

Q81.

y	z	r
0	0	0
0	1	**?**
1	0	1
1	1	0

Q82.

y	z	r
0	0	0
0	1	1
1	0	1
1	1	1

Q83.

y	z	r
0	0	1
0	1	1
1	0	**?**
1	1	0

Q84.

y	z	r
0	0	1
0	1	**?**
1	0	0
1	1	1

Q85.

y	z	r
0	0	1
0	1	**?**
1	0	0
1	1	**?**

Q86.

y	z	r
0	0	0
0	1	0
1	0	0
1	1	1

Q87.

y	z	r
0	0	0
0	1	0
1	0	**?**
1	1	1

Q88.

y	z	r
0	0	**?**
0	1	1
1	0	0
1	1	**?**

Q89.

y	z	r
0	0	0
0	1	1
1	0	**?**
1	1	0

Q90.

y	z	r
0	0	**?**
0	1	**?**
1	0	0
1	1	1

Q91.

y	z	r
0	0	1
0	1	0
1	0	1
1	1	0

Q92.

y	z	r
0	0	0
0	1	1
1	0	0
1	1	0

Q93.

y	z	r
0	0	0
0	1	1
1	0	1
1	1	0

Q94.

y	z	r
0	0	**?**
0	1	**?**
1	0	1
1	1	0

Q95.

y	z	r
0	0	**?**
0	1	0
1	0	1
1	1	**?**

Q96.

y	z	r
0	0	1
0	1	1
1	0	1
1	1	0

Q97.

y	z	r
0	0	1
0	1	1
1	0	0
1	1	0

Q98.

y	z	r
0	0	1
0	1	0
1	0	0
1	1	1

Q99.

y	z	r
0	0	0
0	1	0
1	0	1
1	1	0

Q100.

y	z	r
0	0	0
0	1	**?**
1	0	1
1	1	**?**

3-input problems, without don't-care outputs

Q101.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Q102.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Q103.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Q104.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Q105.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Q106.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Q107.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Q108.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Q109.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Q110.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Q111.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Q112.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Q113.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Q114.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Q115.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Q116.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Q117.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Q118.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Q119.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Q120.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Q121.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Q122.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Q123.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Q124.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Q125.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Q126.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Q127.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Q128.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Q129.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Q130.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Q131.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Q132.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Q133.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Q134.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Q135.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Q136.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Q137.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Q138.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Q139.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Q140.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Q141.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Q142.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Q143.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Q144.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Q145.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Q146.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Q147.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Q148.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Q149.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Q150.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

3-input problems, with don't-care outputs

Q151.

x	y	z	r
0	0	0	**?**
0	0	1	**?**
0	1	0	0
0	1	1	**?**
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Q152.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	**?**
1	0	0	**?**
1	0	1	**?**
1	1	0	**?**
1	1	1	1

Q153.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	**?**
1	1	1	0

Q154.

x	y	z	r
0	0	0	**?**
0	0	1	0
0	1	0	**?**
0	1	1	**?**
1	0	0	1
1	0	1	**?**
1	1	0	**?**
1	1	1	**?**

Q155.

x	y	z	r
0	0	0	0
0	0	1	**?**
0	1	0	**?**
0	1	1	0
1	0	0	**?**
1	0	1	1
1	1	0	1
1	1	1	1

Q156.

x	y	z	r
0	0	0	**?**
0	0	1	0
0	1	0	**?**
0	1	1	1
1	0	0	**?**
1	0	1	1
1	1	0	**?**
1	1	1	1

Q157.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	**?**
0	1	1	**?**
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Q158.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	**?**
0	1	1	**?**
1	0	0	**?**
1	0	1	**?**
1	1	0	0
1	1	1	1

Q159.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	**?**
0	1	1	0
1	0	0	**?**
1	0	1	**?**
1	1	0	1
1	1	1	1

Q160.

x	y	z	r
0	0	0	**?**
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	**?**
1	0	1	0
1	1	0	**?**
1	1	1	**?**

Q161.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	**?**
1	0	1	0
1	1	0	0
1	1	1	0

Q162.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	**?**
1	0	0	**?**
1	0	1	**?**
1	1	0	**?**
1	1	1	1

Q163.

x	y	z	r
0	0	0	**?**
0	0	1	1
0	1	0	**?**
0	1	1	1
1	0	0	0
1	0	1	**?**
1	1	0	0
1	1	1	**?**

Q164.

x	y	z	r
0	0	0	0
0	0	1	**?**
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	**?**

Q165.

x	y	z	r
0	0	0	0
0	0	1	**?**
0	1	0	**?**
0	1	1	1
1	0	0	0
1	0	1	**?**
1	1	0	0
1	1	1	**?**

Q166.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	**?**
1	1	1	0

Q167.

x	y	z	r
0	0	0	**?**
0	0	1	0
0	1	0	0
0	1	1	**?**
1	0	0	**?**
1	0	1	1
1	1	0	1
1	1	1	0

Q168.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	**?**
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Q169.

x	y	z	r
0	0	0	**?**
0	0	1	**?**
0	1	0	**?**
0	1	1	**?**
1	0	0	**?**
1	0	1	0
1	1	0	1
1	1	1	1

Q170.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	**?**
1	0	1	1
1	1	0	0
1	1	1	1

Q171.

x	y	z	r
0	0	0	1
0	0	1	**?**
0	1	0	**?**
0	1	1	0
1	0	0	**?**
1	0	1	0
1	1	0	0
1	1	1	0

Q172.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	**?**
0	1	1	**?**
1	0	0	0
1	0	1	0
1	1	0	**?**
1	1	1	1

Q173.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	**?**
1	0	1	1
1	1	0	**?**
1	1	1	**?**

Q174.

x	y	z	r
0	0	0	0
0	0	1	**?**
0	1	0	**?**
0	1	1	0
1	0	0	1
1	0	1	**?**
1	1	0	0
1	1	1	1

Q175.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	**?**
1	0	1	0
1	1	0	**?**
1	1	1	1

Q176.

x	y	z	r
0	0	0	1
0	0	1	**?**
0	1	0	**?**
0	1	1	0
1	0	0	**?**
1	0	1	1
1	1	0	**?**
1	1	1	**?**

Q177.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	**?**
1	1	1	**?**

Q178.

x	y	z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	**?**
1	1	0	0
1	1	1	1

Q179.

x	y	z	r
0	0	0	**?**
0	0	1	**?**
0	1	0	**?**
0	1	1	**?**
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Q180.

x	y	z	r
0	0	0	**?**
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	**?**
1	1	0	**?**
1	1	1	1

Q181.

x	y	z	r
0	0	0	**?**
0	0	1	**?**
0	1	0	1
0	1	1	1
1	0	0	**?**
1	0	1	1
1	1	0	**?**
1	1	1	0

Q182.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	**?**
1	0	1	0
1	1	0	0
1	1	1	1

Q183.

x	y	z	r
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	**?**
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Q184.

x	y	z	r
0	0	0	**?**
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	**?**
1	0	1	**?**
1	1	0	1
1	1	1	0

Q185.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	**?**
0	1	1	1
1	0	0	**?**
1	0	1	0
1	1	0	**?**
1	1	1	0

Q186.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	**?**
1	0	0	**?**
1	0	1	1
1	1	0	**?**
1	1	1	1

Q187.

x	y	z	r
0	0	0	**?**
0	0	1	**?**
0	1	0	**?**
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	**?**
1	1	1	1

Q188.

x	y	z	r
0	0	0	**?**
0	0	1	0
0	1	0	**?**
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Q189.

x	y	z	r
0	0	0	**?**
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	**?**
1	1	1	1

Q190.

x	y	z	r
0	0	0	**?**
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Q191.

x	y	z	r
0	0	0	**?**
0	0	1	**?**
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	**?**
1	1	1	0

Q192.

x	y	z	r
0	0	0	0
0	0	1	**?**
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	**?**
1	1	1	**?**

Q193.

x	y	z	r
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	**?**
1	0	1	**?**
1	1	0	0
1	1	1	0

Q194.

x	y	z	r
0	0	0	1
0	0	1	0
0	1	0	**?**
0	1	1	**?**
1	0	0	**?**
1	0	1	0
1	1	0	**?**
1	1	1	**?**

4-input problems, without don't-care outputs

Q195.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Q196.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Q197.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Q198.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Q199.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Q200.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Q201.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Q202.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Q203.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Q204.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Q205.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Q206.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Q207.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Q208.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Q209.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Q210.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Q211.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Q212.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Q213.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Q214.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Q215.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	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	1
1	1	1	1	1

Q216.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Q217.

w	x	y	z	r
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	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Q218.

w	x	y	z	r
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	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Q219.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Q220.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Q221.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Q222.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Q223.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Q224.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Q225.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Q226.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Q227.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Q228.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Q229.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
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	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Q230.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Q231.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Q232.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Q233.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Q234.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Q235.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
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	1

Q236.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Q237.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Q238.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Q239.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Q240.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Q241.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Q242.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Q243.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Q244.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

4-input problems, with don't-care outputs

Q245.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	**?**
0	0	1	0	**?**
0	0	1	1	**?**
0	1	0	0	0
0	1	0	1	**?**
0	1	1	0	**?**
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	**?**
1	1	0	1	1
1	1	1	0	**?**
1	1	1	1	1

Q246.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	**?**
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Q247.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	0
0	0	1	0	1
0	0	1	1	**?**
0	1	0	0	0
0	1	0	1	1
0	1	1	0	**?**
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	**?**
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Q248.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	**?**
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	**?**
0	1	1	0	0
0	1	1	1	0
1	0	0	0	**?**
1	0	0	1	**?**
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	**?**
1	1	1	1	1

Q249.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	**?**
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	**?**
1	1	0	0	**?**
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Q250.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	**?**
0	0	1	0	**?**
0	0	1	1	0
0	1	0	0	1
0	1	0	1	**?**
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Q251.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	**?**
0	0	1	0	1
0	0	1	1	**?**
0	1	0	0	**?**
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	**?**
1	0	1	1	**?**
1	1	0	0	1
1	1	0	1	0
1	1	1	0	**?**
1	1	1	1	**?**

Q252.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	0
0	0	1	1	**?**
0	1	0	0	0
0	1	0	1	**?**
0	1	1	0	**?**
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	**?**
1	1	0	1	**?**
1	1	1	0	**?**
1	1	1	1	**?**

Q253.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	**?**
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	**?**
1	0	1	1	0
1	1	0	0	**?**
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Q254.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	**?**
0	0	1	1	**?**
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	**?**
1	0	1	0	**?**
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	**?**

Q255.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	**?**
1	0	0	1	0
1	0	1	0	0
1	0	1	1	**?**
1	1	0	0	1
1	1	0	1	1
1	1	1	0	**?**
1	1	1	1	1

Q256.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	**?**
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	**?**
1	0	0	1	1
1	0	1	0	1
1	0	1	1	**?**
1	1	0	0	**?**
1	1	0	1	**?**
1	1	1	0	0
1	1	1	1	0

Q257.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	**?**
0	1	1	0	0
0	1	1	1	**?**
1	0	0	0	0
1	0	0	1	**?**
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	**?**

Q258.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	**?**
0	1	1	0	0
0	1	1	1	**?**
1	0	0	0	1
1	0	0	1	**?**
1	0	1	0	1
1	0	1	1	**?**
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Q259.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	**?**
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	**?**
1	1	0	1	0
1	1	1	0	**?**
1	1	1	1	1

Q260.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	**?**
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	**?**
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	**?**
1	1	1	0	0
1	1	1	1	1

Q261.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	**?**
0	1	0	1	0
0	1	1	0	1
0	1	1	1	**?**
1	0	0	0	**?**
1	0	0	1	1
1	0	1	0	**?**
1	0	1	1	**?**
1	1	0	0	**?**
1	1	0	1	**?**
1	1	1	0	0
1	1	1	1	0

Q262.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	**?**
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	**?**
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	**?**

Q263.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	**?**
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	**?**
1	0	1	1	**?**
1	1	0	0	0
1	1	0	1	1
1	1	1	0	**?**
1	1	1	1	**?**

Q264.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	**?**
0	0	1	0	**?**
0	0	1	1	1
0	1	0	0	0
0	1	0	1	**?**
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	**?**
1	0	1	1	**?**
1	1	0	0	**?**
1	1	0	1	1
1	1	1	0	**?**
1	1	1	1	**?**

Q265.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	**?**
0	0	1	0	1
0	0	1	1	1
0	1	0	0	**?**
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	**?**
1	0	1	0	1
1	0	1	1	0
1	1	0	0	**?**
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Q266.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	**?**
1	1	0	1	**?**
1	1	1	0	0
1	1	1	1	**?**

Q267.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	**?**
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	**?**
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	**?**
1	1	1	0	**?**
1	1	1	1	0

Q268.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	**?**
0	0	1	0	**?**
0	0	1	1	**?**
0	1	0	0	0
0	1	0	1	1
0	1	1	0	**?**
0	1	1	1	**?**
1	0	0	0	0
1	0	0	1	**?**
1	0	1	0	1
1	0	1	1	**?**
1	1	0	0	1
1	1	0	1	1
1	1	1	0	**?**
1	1	1	1	1

Q269.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Q270.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	**?**
0	0	1	1	0
0	1	0	0	**?**
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	**?**
1	0	1	1	1
1	1	0	0	0
1	1	0	1	**?**
1	1	1	0	0
1	1	1	1	0

Q271.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	**?**
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	**?**
1	0	0	1	1
1	0	1	0	1
1	0	1	1	**?**
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

Q272.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	**?**
1	1	0	0	**?**
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Q273.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	**?**
0	0	1	1	0
0	1	0	0	0
0	1	0	1	**?**
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	**?**
1	0	1	0	0
1	0	1	1	0
1	1	0	0	**?**
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Q274.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	**?**
1	0	0	1	0
1	0	1	0	0
1	0	1	1	**?**
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Q275.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	**?**
0	0	1	0	**?**
0	0	1	1	0
0	1	0	0	1
0	1	0	1	**?**
0	1	1	0	**?**
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	**?**
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Q276.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	**?**
1	0	0	1	**?**
1	0	1	0	1
1	0	1	1	1
1	1	0	0	**?**
1	1	0	1	1
1	1	1	0	0
1	1	1	1	**?**

Q277.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	**?**
0	0	1	0	0
0	0	1	1	1
0	1	0	0	**?**
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	**?**
1	0	0	1	1
1	0	1	0	0
1	0	1	1	**?**
1	1	0	0	1
1	1	0	1	**?**
1	1	1	0	**?**
1	1	1	1	**?**

Q278.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	**?**
0	0	1	0	0
0	0	1	1	**?**
0	1	0	0	1
0	1	0	1	1
0	1	1	0	**?**
0	1	1	1	1
1	0	0	0	**?**
1	0	0	1	**?**
1	0	1	0	**?**
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Q279.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	**?**
0	0	1	0	0
0	0	1	1	**?**
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	**?**
1	0	0	1	1
1	0	1	0	1
1	0	1	1	**?**
1	1	0	0	**?**
1	1	0	1	1
1	1	1	0	**?**
1	1	1	1	1

Q280.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	**?**
0	0	1	0	0
0	0	1	1	1
0	1	0	0	**?**
0	1	0	1	0
0	1	1	0	**?**
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	**?**
1	0	1	1	**?**
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Q281.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	**?**
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	**?**
1	0	0	1	0
1	0	1	0	**?**
1	0	1	1	1
1	1	0	0	**?**
1	1	0	1	**?**
1	1	1	0	0
1	1	1	1	1

Q282.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	**?**
0	0	1	0	0
0	0	1	1	**?**
0	1	0	0	**?**
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	**?**
1	0	0	1	0
1	0	1	0	**?**
1	0	1	1	**?**
1	1	0	0	1
1	1	0	1	0
1	1	1	0	**?**
1	1	1	1	0

Q283.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	**?**
0	1	1	0	0
0	1	1	1	**?**
1	0	0	0	0
1	0	0	1	0
1	0	1	0	**?**
1	0	1	1	**?**
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Q284.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	**?**
0	1	0	0	0
0	1	0	1	**?**
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	**?**
1	0	1	1	1
1	1	0	0	**?**
1	1	0	1	1
1	1	1	0	**?**
1	1	1	1	0

Q285.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	**?**
1	0	0	1	**?**
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	**?**

Q286.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	**?**
0	0	1	0	0
0	0	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	**?**
1	0	1	0	**?**
1	0	1	1	0
1	1	0	0	**?**
1	1	0	1	**?**
1	1	1	0	**?**
1	1	1	1	0

Q287.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	**?**
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	**?**
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	**?**
1	1	1	1	**?**

Q288.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	**?**
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	**?**
0	1	1	1	1
1	0	0	0	0
1	0	0	1	**?**
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	**?**
1	1	1	1	**?**

Q289.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	**?**
0	1	0	0	**?**
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	**?**
1	1	0	0	**?**
1	1	0	1	1
1	1	1	0	**?**
1	1	1	1	0

Q290.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	**?**
0	0	1	1	**?**
0	1	0	0	**?**
0	1	0	1	1
0	1	1	0	1
0	1	1	1	**?**
1	0	0	0	0
1	0	0	1	0
1	0	1	0	**?**
1	0	1	1	**?**
1	1	0	0	**?**
1	1	0	1	**?**
1	1	1	0	**?**
1	1	1	1	**?**

Q291.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	**?**
0	1	1	1	0
1	0	0	0	1
1	0	0	1	**?**
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	**?**
1	1	1	1	0

Q292.

w	x	y	z	r
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	**?**
0	1	0	0	1
0	1	0	1	**?**
0	1	1	0	0
0	1	1	1	**?**
1	0	0	0	1
1	0	0	1	0
1	0	1	0	**?**
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	**?**

Q293.

w	x	y	z	r
0	0	0	0	**?**
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	**?**
0	1	1	0	**?**
0	1	1	1	**?**
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	**?**
1	1	0	1	**?**
1	1	1	0	**?**
1	1	1	1	0

Q294.

w	x	y	z	r
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	**?**
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	**?**
1	1	1	0	1
1	1	1	1	1

3 Chapter 3

Q295. The **parity function** f accepts an n-bit sequence X as input, and yields $f(X) = 1$ iff. X has an odd number of elements equal to 1. If $f(X) = 1$ (resp. $f(X) = 0$), we say the parity of X is odd (resp. even). Using a combinatorial circuit, one can compute this as

$$f(X) = X_0 \oplus X_1 \oplus \cdots \oplus X_{n-1}$$

since XOR can be thought of as addition modulo two. However, how could we design a Finite State Machine (FSM) to compute $f(X)$ when supplied with X one element at a time? Explain step-by-step how you would solve this challenge: start with a high-level design for any FSM then fill in detail required for *this* FSM. Are there any features or requirements you can add to this basic description so the FSM is deemed "better" somehow?

Q296. Imagine you are asked to to build a simple DNA matching hardware circuit as part of a research project. The circuit will be given DNA strings which are sequences of tokens that represent chemical building blocks. The goal is to search a large input sequence of DNA tokens for a small sequence indicative of some feature.

The circuit will receive one token per clock cycle as input; the possible tokens are adenine (A), cytosine (C), guanine (G) and thymine (T). The circuit should, given the input sequence, set an output flag to 1 when the matching sequence ACT is found somewhere in the input or 0 otherwise. You can assume the inputs are infinitely long, i.e., the circuit should just keep searching forever and set the flag when the match is a success.

a Design a circuit to perform the required task, show all your working and explain any design decisions you make.

b Now imagine you are asked to build two new matching circuits which should detect the sequences CAG and TTT respectively. It is proposed that instead of having three separate circuits, they are combined into a single circuit that matches the input sequence against one matching sequence selected with an additional input. Describe **one** advantage and **one** disadvantage you can think of for the two implementation options.

Q297. A revolutionary, ecologically sound washing machine is under development by your company. When turned on, the machine starts in the *idle* state awaiting input. The washing cycle consists of the three stages: *fill* (when it fills with water), *wash* (when the wash occurs), *spin* (when spin dying occurs); the machine then returns to *idle* when it is finished. Two buttons control the machine: pressing B_0 starts the washing cycle, pressing B_1 cancels the washing cycle at any stage and returns the machine to *idle*; if both buttons are pressed at the same time, the machine continues as normal as if neither were pressed.

a You are asked to design a circuit to control the washing machine. Draw a diagram illustrating states the washing machine can be in, and valid transitions between them.

b Translate your diagram from above into a corresponding, tabular description of the transition function.

c Using an appropriate technique, derive Boolean expressions which allow computation of the transition function; note that because the washing machine is ecologically sound, minimising the overall gate count is important.

Q298. Recall that an n-bit Gray code is a cyclic, 2^n-element sequence S where each i-th element S_i is itself an n-element binary sequence, and the Hamming distance between adjacent elements is one, i.e.,

$$\mathcal{D}(S_i, S_{i-1 \pmod{2^n}}) \;=\; \mathcal{D}(S_i, S_{i+1 \pmod{2^n}}) \;=\; 1.$$

a Using an expression (rather than words), define

i $\mathcal{H}(X)$, the Hamming weight of a binary sequence X, and

ii $\mathcal{D}(X, Y)$, the Hamming distance between binary sequences X and Y.

b Consider a D-type flip-flop, capable of storing a 1-bit value, realised using CMOS-based transistors arranged into logic gates. Using a gate-level **circuit diagram**, describe the design of such a component (clearly explaining the purpose of each part).

c Imagine successive elements of a 3-bit Gray code sequence are stored, one after another, in a register realised using flip-flops of the type described above. The fact only one bit changes each time the register is updated could be viewed as advantageous: explain why.

d Using a **block diagram**, draw a generic Finite State Machine (FSMs) framework, including for example δ, ω and any input and output; clearly explain the purpose of each component in the framework.

e Using the framework outlined above, design a concrete FSM which has

- two 1-bit inputs *rst* and *clk*, and
- one 3-bit output *r*.

and whose behaviour is as follows: at each positive edge of the clock signal *clk*, if $rst = 0$ then r should be updated with the next element of a 3-bit Gray code, otherwise r should be reset to the first element.

Note that your answer should provide enough detail to fully specify each component in the framework (e.g., Boolean expressions for δ).

Q299. An electronic security system, designed to prevent unauthorised use of a door, is attached to a mains electricity supply. The system has the following components:

- Three buttons, say B_i for $0 \leq i < 3$, whose value is initially 0; when pressed, a button remains pressed and the value changes to 1.
- A door handle modelled by

$$H = \begin{cases} 1 & \text{when the handle is turned} \\ 0 & \text{when the handle is unturned} \end{cases}$$

- A lock mechanism modelled by

$$L = \begin{cases} 1 & \text{when the door is locked} \\ 0 & \text{when the door is unlocked} \end{cases}$$

If the door handle is turned after the order of button presses matches a 3-element password sequence P, the door should be unlocked; if there is a mismatch, it should remain locked. The mechanism is reset (and all buttons released) whenever the handle is turned (whether or not the door is unlocked). If $P = \langle B_1, B_0, B_2 \rangle$, then for example

- B_1 then B_0 then B_2 is pressed, then the handle is turned, the door is unlocked, i.e., L is set to 0, and the mechanism is reset,
- B_0 then B_1 then B_2 is pressed, then the handle is turned, the door remains locked, i.e., L is set to 1, and the mechanism is reset,
- B_1 then B_0 is pressed, then the handle is turned, the door remains locked, i.e., L is set to 1, and the mechanism is reset.

a Using a **block diagram**, draw a generic Finite State Machine (FSMs) framework, including for example the transision and output functions (i.e., δ and ω) and any input and output; clearly explain the purpose of each component in the framework.

b Imagine the password is fixed to $P = \langle B_2, B_0, B_1 \rangle$. Using the framework outlined above, design a concrete FSM which can be used to control the security system as required.

Note that your answer should provide enough detail to fully specify each component in the framework (e.g., Boolean expressions for the transision function).

c After inspecting your design, someone claims they can avoid the need for a clock signal: explain how this is possible.

d The same person suggests an alternative approach whereby P is not fixed, but rather stored in an SRAM memory device. Although this approach could be more useful, explain **one** reason it could be viewed as disadvantageous.

e Before being sold, each physical system needs to be tested to ensure it functions as advertised. Explain a suitable testing strategy for your design, **and** any alterations required to facilitate it.

Q300. Imagine you are John Connor in the film Terminator II: your aim is to design a device that guesses ATM (or cash machine) Personal Identification Numbers (PINs) using brute-force search. The ATM uses 4-digit decimal PINs, examples being 1234 and 9876. The device stores a current PIN denoted P: it performs each guess in sequence by first checking whether P is correct, then incrementing P ready for the next step. The process concludes when P is deemed correct.

a Two potential representations for the PIN are suggested:

a decimal representation in which the PIN is stored as a sequence of four unsigned integers, i.e., $P = \langle P_0, P_1, P_2, P_3 \rangle$, with each $0 \leq P_i < 10$, or

a binary representation in which the PIN is stored as a single unsigned integer, i.e., P, with $0 \leq P < 10000$.

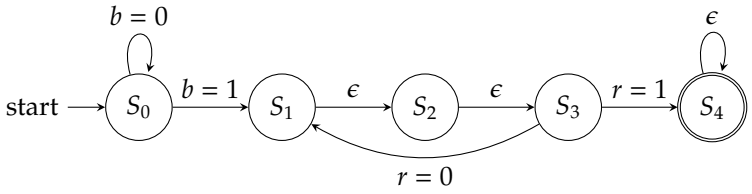
State **one** advantage of **each** option, and explain which you think is more appropriate.

b A combinatorial component within the device should take the current PIN P as input, and produce two outputs:

- the guess sent to the ATM, i.e., $G = \langle G_0, G_1, G_2, G_3 \rangle$, where each $0 \leq G_i < 10$ is the i-th decimal digit of the current PIN, and
- the incremented PIN P' ready for the next guess.

Produce a design for this component; include a **block diagram** and enough detail to fully specify how a gate-level implementation could be performed.

c The device is controlled by a simple Finite State Machine (FSM) which can be described diagrammatically:



In a more explanatory form, the idea is as follows:

- The device starts in state S_0, in which P is initialised; once the start button b is pressed, it moves into state S_1.
- In state S_1, P is driven as input into combinatorial component and the device moves into state S_2.
- In state S_2, G is sent to the ATM and P' is latched to form the new value of P; the device moves into state S_3.
- In state S_3 the device checks the ATM response r. If $r = 1$ then G was the correct guess and the device moves into state S_4 where it halts (i.e., remains in S_4); otherwise, the device moves into state S_1 and the process repeats.

Focusing on the diagram above only, produce a design for the FSM; include a **block diagram**, and enough detail to fully specify how a gate-level implementation could be performed.

4 Chapter 4

Q301.

a An n-bit ripple-carry adder has a critical path that can be described as $O(n)$ gate delays. Explain intuitively why this is the case, and name an alternative whose critical path is shorter.

b Give a single-line C expression to test if a non-zero integer x is an exact power-of-two; i.e., if $x = 2^n$ for some n then the expression should evaluate to a non-zero value, otherwise it evaluates to zero.

c Imagine you are writing a C program that includes a variable called `x`. If `x` has the type `char` and a current value of 127, what is the new value after

i decrementing (i.e., subtracting 1 from it), or

ii incrementing (i.e., adding 1 to it)

the variable?

d Imagine x represents a two's-complement, signed integer using 4 bits; x_i denotes the i-th bit of x. Write a human-readable description (i.e., the meaning) of what the Boolean function

$$f(x) = \neg x_3 \wedge (x_2 \vee x_1 \vee x_0)$$

computes arithmetically.

e Given an n-bit input x, draw a **block diagram** of an efficient (i.e., with a **short** critical path) combinatorial circuit that can compute $r = 7 \cdot x$ (i.e., multiply x by the constant 7). Take care to label each component, and the size (in bits) of each input and output.

Q302.

a Comparison operations for a given processor take two 16-bit operands and return zero if the comparison is false or non-zero if it is true. By constructing some of the comparisons using combinations of other operations, show that implementing all of $=, \neq, <, \leq, >$ and $\geq$ is wasteful. State the smallest set of comparisons that need dedicated hardware such that all the standard comparisons can be executed.

b The ALU in the same processor design does not include a multiply instruction. So that programmers can still multiply numbers, write an efficient C function to multiply two 16-bit inputs together and return the 16-bit lower half of the result. You can assume the inputs are always positive.

c The population count or Hamming weight of x, denoted by $\mathcal{H}(x)$ say, is the number of bits in the binary expansion of x that equal one. Some processors have a dedicated instruction to do this but the proposed one does not; write an efficient C function to compute the population count of 16-bit inputs.

Q303. Imagine we want to compute the result of multiplying two n-bit numbers x and y together, i.e., $r = x \cdot y$, where n is even. One can adopt a divide-and-conquer approach to this computation by splitting x and y into two parts each of size $n/2$ bits

$$\begin{aligned} x &= x_1 \cdot 2^{n/2} + x_0 \\ y &= y_1 \cdot 2^{n/2} + y_0 \end{aligned}$$

and then computing the full result

$$r = r_2 \cdot 2^n + r_1 \cdot 2^{n/2} + r_0$$

via the parts

$$\begin{aligned} r_2 &= x_1 \cdot y_1 \\ r_1 &= x_1 \cdot y_0 + x_0 \cdot y_1 \\ r_0 &= x_0 \cdot y_0. \end{aligned}$$

The naive approach above uses four multiplications of $(n/2)$-bit values. The Karatsuba-Ofman method reduces this to three multiplications (and some extra low-cost operations); show how this is achieved.

Q304. Assume that unsigned integers are represented in 4 bits.

a What is the result of using a normal 4-bit adder circuit to compute the sum 10 + 12?

b A saturating (or clamped) adder is such that if an overflow occurs, i.e., the result does not fit into 4 bits, the highest possible result is returned instead. With a clamped 4-bit addition denoted by $\uplus$, we have that $10 \uplus 12 = 15$ for example. In general, for an n-bit clamped adder

$$x \uplus y = \begin{cases} x + y & \text{if } x + y < 2^n \\ 2^n - 1 & \text{otherwise} \end{cases}$$

Design a circuit that implements a 4-bit adder of this type.

Q305. A software application needs 8-bit, unsigned modular multiplication, i.e., it needs to compute

$$x \cdot y \pmod{N}$$

which is the same as

$$t - (N \cdot \lfloor t/N \rfloor)$$

where $t = x \cdot y$. You have been asked to extend an existing ALU to support this operation. The high cost of a dedicated circuit for division rules out that option; using standard building blocks (e.g., adder, multiplexer) rather than individual logic gates, draw a **block diagram** of an alternative solution.

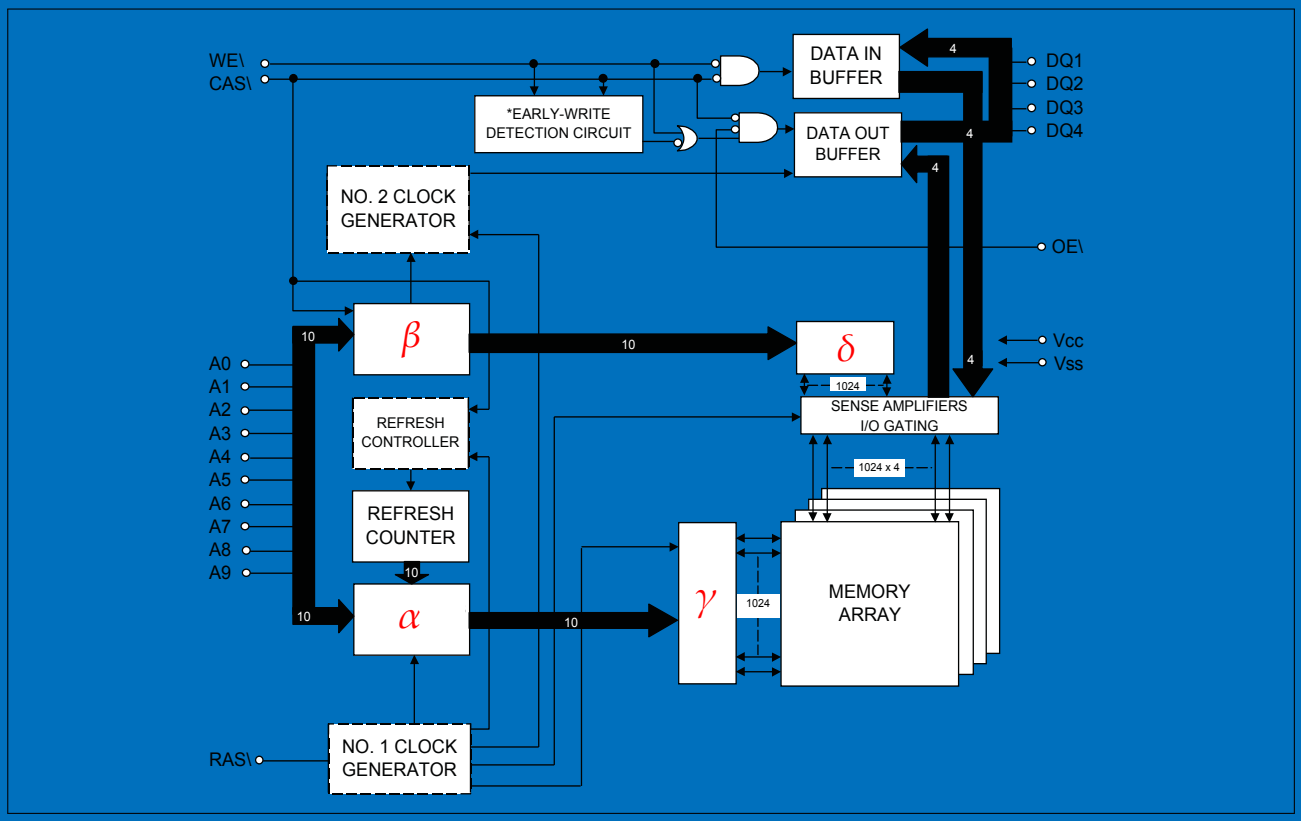


Figure 1: *A* 4Mbit *DRAM block diagram (source:* `http://www.micross.com/pdf/MT4C4001J.pdf`*).*