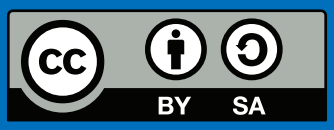
A Practical Introduction to Computer Architecture

Daniel Page ⟨dan@phoo.org⟩

git # b4055dd3 @ 2019-05-13



EXAMPLE EXAM-STYLE SOLUTIONS

1 Chapter 1

S1.

a
- i $|A| = 3$.
- ii $A \cup B = \{1, 2, 3, 4, 5\}$.
- iii $A \cap B = \{3\}$.
- iv $A - B = \{1, 2\}$.
- v $\overline{A} = \{4, 5, 6, 7, 8\}$.
- vi $\{x \mid 2 \cdot x \in \mathcal{U}\} = \{1, 2, 3, 4\}$.

b
- i +0 in sign-magnitude is 00000000, in two's-complement is 00000000.
- ii −0 in sign-magnitude is 10000000, in two's-complement is 00000000.
- iii +72 in sign-magnitude is 01001000, in two's-complement is 01001000.
- iv −34 in sign-magnitude is 10100010, in two's-complement is 11011110.
- v −8 in sign-magnitude is 10001000, in two's-complement is 11111000.
- vi This is a trick question: one cannot represent 240 in 8-bit sign-magnitude or two's-complement; the incorrect guess of 11111000 in two's-complement for example is actually −8.

S2. 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. Using an unsigned 32-bit integer x for example, an implementation might be written as follows:

```
int H( uint32_t x ) {
  int t = 0;

  for( int i = 0; i < 32; i++ ) {
    if( ( x >> i ) & 1 ) {
      t = t + 1;
    }
  }

  return t;
}
```

S3. a The truth table for this function is as follows

a	b	c	$(a \wedge b \wedge \neg c)$	$(a \wedge \neg b \wedge c)$	$(\neg a \wedge \neg b \wedge c)$	$f(a, b, c)$
0	0	0	0	0	0	0
0	0	1	0	0	1	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	0	1	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	0

Since there are $n = 3$ input variables, there are clearly $2^n = 2^3 = 8$ input combinations; three of these produce 1 as an the output from the function.

b The truth table for this function is as follows

a	b	c	d	$f(a,b,c,d)$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	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	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

so since there is only one case where $f(a,b,c,d) = 1$, the only assignment given which matches the criteria is $a = 0$, $b = 1$, $c = 0$ and $d = 1$.

This hints at a general principle: when we have an expression like this, a term such as $\neg x$ can be read as "x should be 0" and x as "x should be 1". So the expression as a whole is read as "a should be 0 and b should be 1 and c should be 0 and d should be 1". Since we have basically fixed all four inputs, only *one* entry of the truth table matches. On the other hand, if we instead had

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

for example, we would be saying "a should be 0 and b should be 1 and c should be 0, and d can be anything" which gives *two* possible assignments (i.e., $a = 0$, $b = 1$, $c = 0$ and either $d = 0$ or $d = 1$).

c Informally, SoP form means there are say n terms in the expression: each term is the conjunction of some variables (or their complement), and the expression is the disjunction of the terms. As conjunction and disjunction basically means the AND and OR operators, and AND and OR act sort of like multiplication and addition, the SoP name should make some sense: the expression is sort of like the sum of terms which are themselves each a product of variables. The second option is correct as a result; the first and last violate the form described above somehow (e.g., the first case is in the opposite, PoS form).

d One can easily make a comparison using a truth table such as

a	b	$a \vee 1$	$a \oplus 1$	$\neg a$	$a \wedge 1$	$\neg(a \wedge b)$	$\neg a \vee \neg b$
0	0	1	1	1	0	1	1
0	1	1	1	1	0	1	1
1	0	1	0	0	1	1	1
1	1	1	0	0	1	0	0

from which it should be clear that all the equations are correct except for the first one. That is, $a \vee 1 \neq a$ but rather $a \vee 1 = 1$.

e i Inspecting the following truth table

a	$\neg a$	$\neg\neg a$
0	1	0
0	1	0
1	0	1
1	0	1

shows this equivalence is correct (this is the involution axiom).

ii Inspecting the following truth table

a	b	$\neg a$	$\neg b$	$a \wedge b$	$\neg(a \wedge b)$	$\neg a \vee \neg b$
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0

shows this equivalence is correct (this is the de Morgan axiom).

iii Inspecting the following truth table

a	b	$\neg a$	$\neg b$	$\neg a \wedge b$	$a \wedge \neg b$
0	0	1	1	0	0
0	1	1	0	1	0
1	0	0	1	0	1
1	1	0	0	0	0

shows this equivalence is incorrect.

iv Inspecting the following truth table

a	$\neg a$	$a \oplus a$
0	1	0
1	0	0

shows this equivalence is incorrect.

S4. a The dual of any expression is constructed by using the principle of duality, which informally means swapping each AND with OR (and vice versa) and each 0 with 1 (and vice versa); this means, for example, we can take the OR form of each axiom and produce the AND form (and vice versa).

So in this case, we start with an OR form: this means the dual will the corresponding AND form. Making the swaps required means we end up with

$$x \wedge 0 \equiv 0$$

so the second option is correct.

b This question is basically asking for the complement of f, since the options each have $\neg f$ on the left-hand side: this means using the principle of complements, a generalisation of the de Morgan axiom, by swapping each variable with the complement (and vice versa), each AND with OR (and vice versa), and each 0 with 1 (and vice versa). If we apply these rules (taking care with the parenthesis) to

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

we end up with

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

which matches the last option.

c The de Morgan axiom, which can be generalised using by the principle of complements, says that

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

or conversely that

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

You can think of either form as "pushing" the NOT operator on the left-hand side into the parentheses: this acts to complement each variable, and swap the AND to an OR (or vice versa). We know that

$$\begin{aligned} x \mathbin{\overline{\wedge}} y &\equiv \neg(x \wedge y) \\ x \mathbin{\overline{\vee}} y &\equiv \neg(x \vee y) \end{aligned}$$

So pattern matching against the options, it is clear the first one is correct, for example, because

$$x \mathbin{\overline{\vee}} y \quad \equiv \quad \neg(x \vee y) \quad \equiv \quad \neg x \wedge \neg y$$

where the right-hand side matches the description of an AND whose two inputs are complemented. Likewise, the second one is correct because

$$x \mathbin{\overline{\wedge}} y \quad \equiv \quad \neg(x \wedge y) \quad \equiv \quad \neg x \vee \neg y.$$

S5. a The third option, i.e., $\neg a \wedge \neg b$ is the correct one; the three simplification steps, via two axioms, are as follows:

$$\begin{array}{rlllllll} & \neg & (a \vee b) & \wedge & \neg & (c \vee d \vee e) & \vee & \neg & (a \vee b) & \\ = & & (\neg a \wedge \neg b) & \wedge & \neg & (c \vee d \vee e) & \vee & & (\neg a \wedge \neg b) & \text{(de Morgan)} \\ = & & (\neg a \wedge \neg b) & \wedge & & (\neg c \wedge \neg d \wedge \neg e) & \vee & & (\neg a \wedge \neg b) & \text{(de Morgan)} \\ = & & \neg a \wedge \neg b & & & & & & & \text{(absorption)} \end{array}$$

b We can clearly see that

$$\begin{array}{rll} & (a \vee b \vee c) \wedge \neg(d \vee e) \vee (a \vee b \vee c) \wedge (d \vee e) & \\ = & (a \vee b \vee c) \wedge (\neg(d \vee e) \vee (d \vee e)) & \text{(distribution)} \\ = & (a \vee b \vee c) \wedge ((d \vee e) \vee \neg(d \vee e)) & \text{(commutativity)} \\ = & (a \vee b \vee c) \wedge 1 & \text{(inverse)} \\ = & a \vee b \vee c & \text{(identity)} \end{array}$$

meaning the first option is the correct one.

c We can clearly see that

$$\begin{array}{rll} & a \wedge c \vee c \wedge (\neg a \vee a \wedge b) & \\ = & (a \wedge c) \vee (c \wedge (\neg a \vee (a \wedge b))) & \text{(precedence)} \\ = & (c \wedge a) \vee (c \wedge (\neg a \vee (a \wedge b))) & \text{(commutativity)} \\ = & c \wedge (a \vee \neg a \vee (a \wedge b)) & \text{(distribution)} \\ = & c \wedge (1 \vee (a \wedge b)) & \text{(inverse)} \\ = & c \wedge ((a \wedge b) \vee 1) & \text{(commutativity)} \\ = & c \wedge 1 & \text{(null)} \\ = & c & \text{(identity)} \end{array}$$

meaning the last option is the correct one: *none* of the above is correct, since the correct simplification is actually just c.

d The fourth option, i.e., $a \wedge b$ is correct. This basically stems from repeated application of the absorption axiom, the AND form of which states

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

Applying it from left-to-right, we find that

$$\begin{array}{rll} & a \wedge b \vee a \wedge b \wedge c \vee a \wedge b \wedge c \wedge d \vee a \wedge b \wedge c \wedge d \wedge e \vee a \wedge b \wedge c \wedge d \wedge e \wedge f & \\ = & (a \wedge b) \vee (a \wedge b) \wedge (c) \vee (a \wedge b) \wedge (c \wedge d) \vee (a \wedge b) \wedge (c \wedge d \wedge e) \vee (a \wedge b) \wedge (c \wedge d \wedge e \wedge f) & \text{(precedence)} \\ = & (a \wedge b) \vee (a \wedge b) \wedge (c \wedge d) \vee (a \wedge b) \wedge (c \wedge d \wedge e) \vee (a \wedge b) \wedge (c \wedge d \wedge e \wedge f) & \text{(absorption)} \\ = & (a \wedge b) \vee (a \wedge b) \wedge (c \wedge d \wedge e) \vee (a \wedge b) \wedge (c \wedge d \wedge e \wedge f) & \text{(absorption)} \\ = & (a \wedge b) \vee (a \wedge b) \wedge (c \wedge d \wedge e \wedge f) & \text{(absorption)} \\ = & (a \wedge b) & \text{(absorption)} \end{array}$$

e We can simplify this function as follows

$$\begin{array}{rll} f(a,b,c) & = & (a \wedge b) \vee a \wedge (a \vee c) \vee b \wedge (a \vee c) \\ & = & (a \wedge b) \vee a \vee b \wedge (a \vee c) \qquad \text{(absorbtion)} \\ & = & a \vee (a \wedge b) \vee b \wedge (a \vee c) \qquad \text{(commutitivity)} \\ & = & a \vee b \wedge (a \vee c) \qquad \text{(absorbtion)} \\ & = & a \vee (b \wedge a) \vee (b \wedge c) \qquad \text{(distribution)} \\ & = & a \vee (a \wedge b) \vee (b \wedge c) \qquad \text{(commutitivity)} \\ & = & a \vee (b \wedge c) \qquad \text{(commutitivity)} \end{array}$$

at which point there is nothing else that can be done: we end up with 2 operators (and AND and an OR), so the second option is correct.

f Working from the right-hand side toward the left, we have that

$$\begin{array}{rll} & \neg x \vee \neg y & \\ = & (\neg x \wedge 1) \vee \neg y & \text{(identity)} \\ = & (\neg x \wedge 1) \vee (\neg y \wedge 1) & \text{(identity)} \\ = & (\neg x \wedge (y \vee \neg y)) \vee (\neg y \wedge 1) & \text{(inverse)} \\ = & (\neg x \wedge (y \vee \neg y)) \vee (\neg y \wedge (x \vee \neg x)) & \text{(inverse)} \\ = & (\neg x \wedge y) \vee (\neg x \wedge \neg y) \vee (\neg y \wedge (x \vee \neg x)) & \text{(distribution)} \\ = & (\neg x \wedge y) \vee (\neg x \wedge \neg y) \vee (\neg y \wedge x) \vee (\neg y \wedge \neg x) & \text{(distribution)} \\ = & (\neg x \wedge y) \vee (\neg y \wedge x) \vee (\neg x \wedge \neg y) \vee (\neg x \wedge \neg y) & \text{(commutativity)} \\ = & (\neg x \wedge y) \vee (\neg y \wedge x) \vee (\neg x \wedge \neg y) & \text{(idempotency)} \end{array}$$

g By writing

$$\begin{array}{lcl} t_0 & = & x \wedge y \\ t_1 & = & y \wedge z \\ t_2 & = & y \vee z \\ t_3 & = & x \vee z \\ t_4 & = & t_1 \wedge t_2 \end{array}$$

we can shorten the LHS and RHS to

$$\begin{array}{lcl} f & = & t_0 \vee t_4 \\ g & = & y \wedge t_3 \end{array}$$

and then perform a brute-force enumeration

x	y	z	t_0	t_1	t_2	t_3	t_4	f	g
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	0	0	0
0	1	0	0	0	1	0	0	0	0
0	1	1	0	1	1	1	1	1	1
1	0	0	0	0	0	1	0	0	0
1	0	1	0	0	1	1	0	0	0
1	1	0	1	0	1	1	0	1	1
1	1	1	1	1	1	1	1	1	1

to demonstrate that $f = g$, i.e., the equivalence holds. Note that this approach is not as robust if the intermediate steps are not shown; simply including f and g in the truth table does not give much more confidence that simply writing the equivalence!

To prove the equivalence using an axiomatic approach, the following steps can be applied:

$$\begin{array}{lll} & (x \wedge y) \vee (y \wedge z \wedge (y \vee z)) & \\ = & (x \wedge y) \vee (y \wedge z \wedge y) \vee (y \wedge z \wedge z) & \text{(distribution)} \\ = & (x \wedge y) \vee (y \wedge y \wedge z) \vee (y \wedge z \wedge z) & \text{(commutativity)} \\ = & (x \wedge y) \vee (y \wedge z) \vee (y \wedge z) & \text{(idempotency)} \\ = & (x \wedge y) \vee (y \wedge z) & \text{(idempotency)} \\ = & (y \wedge x) \vee (y \wedge z) & \text{(commutativity)} \\ = & y \wedge (x \vee z) & \text{(distribution)} \end{array}$$

2 Chapter 2

S6. Writing

$$\begin{array}{lcl} t_0 & = & (x \wedge y) \oplus z \\ t_1 & = & (\neg x \vee y) \oplus z \\ t_2 & = & (x \vee \neg y) \oplus z \\ t_3 & = & \neg(x \vee y) \oplus z \\ t_4 & = & \neg\neg(x \vee y) \oplus z \end{array}$$

for brevity, we can write the following truth table:

x	y	z	t_0	t_1	t_2	t_3	t_4
0	0	0	0	1	1	1	0
0	0	1	1	0	0	0	1
0	1	0	0	1	0	0	1
0	1	1	1	0	1	1	0
1	0	0	0	0	1	0	1
1	0	1	1	1	0	1	0
1	1	0	1	1	1	0	1
1	1	1	0	0	0	1	0

Looking at the row where $x = 0$, $y = 0$ and $z = 1$, it is clear that $t_0 = 1$ and $t_4 = 1$, so $(x \wedge y) \oplus z$ and $\neg\neg(x \vee y) \oplus z$ are the correct answers.

S7. In the same way as NAND, we know NOR is functionally complete: this can be shown via

$$\begin{array}{lcl} \neg x & \equiv & x \mathbin{\overline{\vee}} x \\ x \wedge y \equiv \neg x \mathbin{\overline{\vee}} \neg y & \equiv & (x \mathbin{\overline{\vee}} x) \mathbin{\overline{\vee}} (y \mathbin{\overline{\vee}} y) \\ x \vee y & \equiv & (x \mathbin{\overline{\vee}} y) \mathbin{\overline{\vee}} (x \mathbin{\overline{\vee}} y) \end{array}$$

Then, since $x \mathbin{\overline{\vee}} y \equiv \neg(x \vee y)$, clearly $\{\vee, \neg\}$ is functionally complete: this set can be rewritten directly as $\{ \mathbin{\overline{\vee}} \}$. We can harness these facts to show that in fact *all* other options are functionally complete.

- Given the truth table

x	y	$\oplus$
0	0	0
0	1	1
1	0	1
1	1	0

  we have $\neg x \equiv x \oplus 1$, or, put another way, we can construct $\neg$ using $\oplus$. Overall then,

  $$\{\oplus, \vee\} \quad \rightsquigarrow \quad \{\neg, \vee\},$$

  i.e., we have constructed the RHS from the LHS; we know the RHS is functionally complete, so the LHS is as well.

- This option is somewhat more difficult: using the same strategy as above, we now need to construct both $\neg$ *and* $\vee$.

  – Given the truth table

x	y	$\equiv$	$\not\equiv$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0

    it should be clear that since $x \not\equiv y \equiv x \oplus y$, we have $\neg x \equiv x \not\equiv 1$. Alternatively, given the truth table

x	y	$\Rightarrow$
0	0	1
0	1	1
1	0	0
1	1	1

    we have $x \Rightarrow 0 \equiv \neg x$.

  – Given the axiom $x \Rightarrow y \equiv \neg x \vee y$, we could write $\neg x \Rightarrow y \equiv \neg(\neg x) \vee y \equiv x \vee y$. That is, based on constructions for $\neg$ and $\Rightarrow$ (the LHS) we can construct $\vee$ (the RHS).

  Overall then,

  $$\{\Rightarrow, \not\equiv\} \quad \rightsquigarrow \quad \{\neg, \vee\},$$

  i.e., we have constructed the RHS from the LHS; we know the RHS is functionally complete, so the LHS is as well.

- Based on the above, it should be clear that

  $$\{\Rightarrow\} \quad \rightsquigarrow \quad \{\neg, \vee\}$$

  because we can construct both $\neg$ and $\vee$ using it alone; in the above $\not\equiv$ was potentially redundant in fact, which is also true of $\not\Rightarrow$ here. Alternatively, given the truth table

x	y	$\Rightarrow$	$\not\Rightarrow$
0	0	1	0
0	1	1	0
1	0	0	1
1	1	1	0

  we could write $1 \not\Rightarrow y \equiv \neg y$, and use $\not\Rightarrow$ as a replacement for $\not\equiv$ and so $\neg$ as required in the above. Either way, it is clearly the case that

  $$\{\Rightarrow, \not\Rightarrow\} \quad \rightsquigarrow \quad \{\neg, \vee\}$$

  i.e., we have constructed the RHS from the LHS; we know the RHS is functionally complete, so the LHS is as well.

S8. You may be able to just spot which one is incorrect, but looking at each case exhaustively (via a truth table for the LHS and RHS of the supposed equivalence), we see that

x	y	z	$(x \wedge y) \wedge z$	$x \wedge (y \wedge z)$	$x \vee 1$	x	$x \vee \neg x$	1	$\neg(x \vee y)$	$\neg x \wedge \neg y$	$\neg\neg x$	x
0	0	0	0	0	1	0	1	1	1	1	0	0
0	0	1	0	0	1	0	1	1	1	1	0	0
0	1	0	0	0	1	0	1	1	0	0	0	0
0	1	1	0	0	1	0	1	1	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	1	1
1	0	1	0	0	1	1	1	1	0	0	1	1
1	1	0	0	0	1	1	1	1	0	0	1	1
1	1	1	1	1	1	1	1	1	0	0	1	1

and identify $x \vee 1 \equiv x$ as the incorrect case: it probably should be $x \vee 1 \equiv 1$, or maybe $x \wedge 1 \equiv x$.

S9. By writing

$$\begin{aligned} t_0 &= x \vee (z \vee y) \\ t_1 &= \neg(\neg y \wedge \neg z) \end{aligned}$$

we can shorten the expression to

$$t_2 = t_0 \wedge t_1.$$

Then, you can see either by enumeration, i.e.,

x	y	z	t_0	t_1	t_2	$y \vee z$
0	0	0	0	0	0	0
0	0	1	1	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	0	0
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

or via the derivation

$$\begin{array}{lll} & (x \vee (z \vee y)) \wedge \neg(\neg y \wedge \neg z) & \\ = & (x \vee (z \vee y)) \wedge (y \vee z) & \text{(de Morgan)} \\ = & (x \vee (y \vee z)) \wedge (y \vee z) & \text{(commutativity)} \\ = & (x \vee (y \vee z)) \wedge ((y \vee z) \vee 0) & \text{(identity)} \\ = & ((y \vee z) \vee x) \wedge ((y \vee z) \vee 0) & \text{(commutativity)} \\ = & (y \vee z) \vee (x \wedge 0) & \text{(distribution)} \\ = & (y \vee z) \vee (0) & \text{(null)} \\ = & (y \vee z) & \text{(identity)} \end{array}$$

that the correct answer is $y \vee z$.

S10. By writing

$$\begin{aligned} t_0 &= x \vee y \\ t_1 &= x \wedge z \end{aligned}$$

we can shorten the expression to

$$t_2 = t_0 \vee t_1.$$

Then, you can see either by enumeration, i.e.,

x	y	z	t_0	t_1	t_2	$x \vee y$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	1	0	1	1
0	1	1	1	0	1	1
1	0	0	1	0	1	1
1	0	1	1	1	1	1
1	1	0	1	0	1	1
1	1	1	1	1	1	1

or via the derivation

$$\begin{array}{lll} & (x \vee y) \vee (x \wedge z) & \\ = & (y \vee x) \vee (x \wedge z) & \text{(commutativity)} \\ = & y \vee (x \vee (x \wedge z)) & \text{(association)} \\ = & y \vee x & \text{(absorption)} \\ = & x \vee y & \text{(commutativity)} \end{array}$$

that the correct answer is $x \vee y$.

S11. First, note that each input can obviously be assigned one of two values, namely 0 or 1, so there are 2^n possible assignments to n inputs. For example, if we have 1 input, say x, there are $2^1 = 2$ possible assignments because x can either be 0 or 1. In the same way, for 2 inputs, say x and y, there are $2^2 = 4$ possible assignments: we can have

$$\begin{array}{ll} x = 0 & y = 0 \\ x = 0 & y = 1 \\ x = 1 & y = 0 \\ x = 1 & y = 1 \end{array}$$

This is why a truth table for n inputs will have 2^n rows: each row details one assignment to the inputs, and the associated output.

So how many *functions* are there? A function with n-inputs means a truth table with 2^n rows; each row includes an output that can either be 0 or 1 (depending on exactly *which* function the truth table describes). So to count how many functions there are, we can just count how many possible assignments there are to the 2^n outputs. The correct answer is 2^{2^n}.

S12. The question essentially just demands application of

$$\hat{x} \mapsto \sum_{i=0}^{i<n} x_i \cdot b^i,$$

which defines a mapping between the representation (on the LHS) and value (on the RHS) of x. In this case, setting $b = 3$ allows computation of a (decimal) value for *each* representation (i.e., literal); we need to select the one whose value turns out to be $123_{(10)}$. As such, it should be clear that

$$\begin{array}{lll} \hat{x} = 11120 & \mapsto & \langle 0, 2, 1, 1, 1 \rangle_{(3)} \\ & \mapsto & \sum_{i=0}^{i<n} x_i \cdot 3^i \\ & \mapsto & 0 \cdot 3^0 + 2 \cdot 3^1 + 1 \cdot 3^2 + 1 \cdot 3^3 + 1 \cdot 3^4 \\ & \mapsto & 0 \cdot 1 + 2 \cdot 3 + 1 \cdot 9 + 1 \cdot 27 + 1 \cdot 81 \\ & \mapsto & 123_{(10)} \end{array}$$

st. 11120 is the correct answer.

S13. It should be clear that

$$\begin{array}{lll} \hat{x} & = & \langle x_0, x_1, \ldots, x_{n-1} \rangle \\ & = & \langle 1, 1, 0, 0, 1, 1, 0, 0 \rangle \\ & \mapsto & -x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i \\ & \mapsto & 2^0 + 2^1 + 2^4 + 2^5 \\ & \mapsto & 1 + 2 + 16 + 32 \\ & \mapsto & 51_{(10)} \end{array}$$

$$\begin{array}{lll} \hat{y} & = & \langle y_0, y_1, \ldots, y_{n-1} \rangle \\ & = & \langle 1, 1, 0, 0, 1, 1, 0, 0 \rangle \\ & \mapsto & -1^{y_{n-1}} \cdot \sum_{i=0}^{n-2} y_i \cdot 2^i \\ & \mapsto & 1 \cdot (2^0 + 2^1 + 2^4 + 2^5) \\ & \mapsto & 1 + 2 + 16 + 32 \\ & \mapsto & 51_{(10)} \end{array}$$

i.e., both x and y are represented by the binary literal 00110011, which yields the decimal value $51_{(10)}$ in both cases. However, if we set the MSB of $\hat{x}$ and $\hat{y}$ to 1, then we get

$$\begin{aligned}
\hat{x}' &= \langle x'_0, x'_1, \ldots, x'_{n-1} \rangle \\
&= \langle 1, 1, 0, 0, 1, 1, 0, 1 \rangle \\
&\mapsto -x'_{n-1} \cdot 2^{n-1} + \textstyle\sum_{i=0}^{n-2} x'_i \cdot 2^i \\
&\mapsto 2^0 + 2^1 + 2^4 + 2^5 - 2^7 \\
&\mapsto 1 + 2 + 16 + 32 - 128 \\
&\mapsto -77_{(10)}
\end{aligned}$$

$$\begin{aligned}
\hat{y}' &= \langle y'_0, y'_1, \ldots, y'_{n-1} \rangle \\
&= \langle 1, 1, 0, 0, 1, 1, 0, 1 \rangle \\
&\mapsto -1^{y'_{n-1}} \cdot \textstyle\sum_{i=0}^{n-2} y'_i \cdot 2^i \\
&\mapsto -1 \cdot (2^0 + 2^1 + 2^4 + 2^5) \\
&\mapsto -1 - 2 - 16 - 32 \\
&\mapsto -51_{(10)}
\end{aligned}$$

so −77 and −51 is the correct answer.

S14. In 16 bits, the largest possible positive value we can represent using two's-complement is

$$2^{n-1} - 1 = 2^{16-1} - 1 = 32767$$

and the largest possible negative value is

$$-2^{n-1} = -2^{16-1} = -32768$$

Using this, we can deduce the following:

a The largest possible positive product is given by

$$x \cdot y = -32768 \cdot -32768 = 1073741824$$

which is somewhat counter-intuitive given both operands are negative, but stems from the fact that in two's-complement more negative values can be represented than positive.

b The largest possible negative product is given by

$$x \cdot y = -32768 \cdot 32767 = -1073709056$$

or

$$x \cdot y = 32767 \cdot -32768 = -1073709056.$$

S15. First, note that

$$\begin{array}{lcll}
\texttt{x} & = & 256_{(10)} & = 0000000100000000_{(2)} \\
\texttt{y} & = & 4852_{(10)} & = 0001001011110100_{(2)}
\end{array}$$

Casting from `short` to `char` basically truncates the value from 16 to 8 bits (i.e., leaves the 8 LSBs only), meaning

$$\begin{array}{lcll}
\texttt{(char)(x)} & = & 00000000_{(2)} & = 0_{(10)} \\
\texttt{(char)(y)} & = & 11110100_{(2)} & = -12_{(10)}
\end{array}$$

st. 0 and −12 is the correct answer.

S16. Given

$$\begin{array}{lcl}
\texttt{x} & = & 9_{(10)} \\
 & = & 00001001_{(2)} \\
\texttt{0x97} & = & 97_{(16)} \\
 & = & 10010111_{(2)}
\end{array}$$

the expression `( ~x << 4 ) | 0x97` evaluates to give

$$\begin{array}{lcl}
\texttt{(~x)} & = & 11110110_{(2)} \\
\texttt{(~x << 4)} & = & 01100000_{(2)} \\
\texttt{(~x << 4) | 0x97} & = & 11110111_{(2)} \\
 & = & -9_{(10)}
\end{array}$$

st. `r` = −9 is the correct answer.

S17. For a signed, n-bit integers represented using two's-complement we know that

$$-2^{n-1} \leq \mathtt{x} \leq 2^{n-1} - 1$$

so for the 8-bit examples in this case

$$-128 \leq \mathtt{x} \leq 127.$$

Of the 256 possible values of `x`, 128 are negative (i.e., < 0) and 128 are positive (i.e., ≥ 0). Clearly *all* positive values are fixed points: for these, the `if` statement causes the assignment `r = x` to be executed. So the question is whether or not *any* of the negative values are fixed points?

At first glance, this would seem impossible so none is an understandable response. However, recall that negation in two's-complement is achieved via

$$\mathtt{r} = -\mathtt{x} = \neg\mathtt{x} + 1.$$

Now consider what happens if we negate $\mathtt{x} = -128$:

$$\begin{aligned} -\mathtt{x} &= \neg\,\mathtt{x} + 1 \\ &= \neg\,10000000_{(2)} + 1 \\ &= 01111111_{(2)} + 1 \\ &= 10000000_{(2)} \\ &= \mathtt{x} \end{aligned}$$

This means $\mathtt{x} = -128$ is a fixed point, i.e., that `abs` fails to work correctly for this value (since we cannot represent 128 in 8 bits using two's-complement). As such, 129 is the correct answer: the 128 positive values, plus the 1 negative value from above.

S18. We can deal with the first two statements in one go: an N-MOSFET (or N-type MOSFET) has N-type semiconductor terminals and a P-type body. If the type of semiconductors were swapped, we have a P-MOSFET (or P-type MOSFET) not an N-MOSFET.

A CMOS cell *is* a pairing of two transistors. However, it depends on use of complementary types, namely one N-type and one P-type, rather than the same type as suggested. As such, this statement is incorrect (although subtly so).

A given N-MOSFET is deemed active (resp. inactive) when current is allowed (resp. disallowed) to flow. The flow is, in some sense, controlled by the voltage level applied: it acts to widen or narrow the associated depletion region. At some threshold, "enough" voltage is applied for there to be "enough" current flowing for source and drain to be deemed connected and hence the MOSFET active. So this statement is true, although arguably some detail is glossed over (e.g., the fact a leakage current will *always* exist, so a transistor is *always* a little bit active).

S19. This is a NAND gate, so clearly there *are* two inputs and one output: by inspecting the circuit we can see them labelled x, y and r, plus identify the two power rails labelled V_{dd} and V_{ss}. The output r is "pulled-up" to the V_{dd} voltage level iff. a connection is made via one or other of the top two transistors. Their parallel arrangement gives a hint at their type, even if the system is not recognisable. If we look at the truth table for NAND, i.e.,

NAND		
x	y	r
0	0	1
0	1	1
1	0	1
1	1	0

then if either $x = 0$ or $y = 0$ then $r = 1$, where $V_{ss} \equiv 0$ and $V_{dd} \equiv 1$: these must be P-MOSFETS, therefore, because we want a connection to be formed between r and V_{dd} if $x = V_{ss}$ or $y = V_{ss}$. There is, of course, a companion pull-down network allowing r to be "pulled-down" to the V_{ss} voltage level. However, this is constructed from a sequential arrangement of N-MOSFETS: we did not study BJT transistors, which represent a different technology from MOSFETs.

Finally, note that it is highly unlikely you will see a flux capacitor in a circuit other than during Back To The Future!

S20. Provided you know what the behaviour of the pull-up network (top, consisting of P-MOSFETs) and pull-down network (bottom, consisting of N-MOSFETs) is, it is *reasonably* easy (if long winded) to answer the question by looking at a case-by-case analysis. A more efficient way is to spot the sequential and parallel organisation of MOSFETs:

- If $x = V_{ss}$, y and z are irrelevant because a connection between r and V_{dd} is formed (via the bottom-left P-MOSFET), while a connection between r and V_{dd} is impossible (due to the top N-MOSFET).
- If $x = V_{dd}$, y and z are relevant:
  - If $y = V_{ss}$, $z = V_{ss}$ then a path between r and V_{ss} is impossible; in this case, however, a path between r and V_{dd} is formed (via the top P-MOSFETs).
  - If $y = V_{ss}$, $z = V_{dd}$ then a path between r and V_{ss} is formed (via the top and bottom-right N-MOSFETs).
  - If $y = V_{dd}$, $z = V_{ss}$ then a path between r and V_{ss} is formed (via the top and bottom-left N-MOSFETs).
  - If $y = V_{dd}$, $z = V_{dd}$ then a path between r and V_{dd} is impossible; in this case, however, a path between r and V_{ss} is formed (via the bottom N-MOSFETs).

So given $V_{ss} \equiv 0$ and $V_{dd} \equiv 1$, we can write

x	y	z	$r = f(x, y, z)$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

which, translated into an expression, is $r = \neg(x \wedge (y \vee z)$ or $\neg x \vee (\neg y \wedge \neg z)$ if you prefer: basically $r = 1$ if $x = 0$ or both $y = 0$ and $z = 0$, otherwise $r = 0$.

S21. This question is tricky, in the sense there are *lots* of ways an XOR gate can be constructed using logic gate instances. One can show, for example, that $x \oplus y$ is equivalent to

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

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

c $t_0 \mathbin{\overline{\wedge}} t_1$ where $t_0 = (x \mathbin{\overline{\wedge}} x) \mathbin{\overline{\wedge}} y$ and $t_1 = (y \mathbin{\overline{\wedge}} y) \mathbin{\overline{\wedge}} x$, or

d $t_1 \mathbin{\overline{\wedge}} t_2$ where $t_0 = x \mathbin{\overline{\wedge}} y$, $t_1 = x \mathbin{\overline{\wedge}} t_0$, and $t_2 = y \mathbin{\overline{\wedge}} t_0$,

and potentially more besides: note that the question rules out the otherwise viable option of directly using transistors, for example.

To compare the options, the starting point is how each logic gate we *could* use will be realised using transistors. In reality this may increase the range of possible answers even further, but to provide *an* answer imagine that we only consider the four options above, and assume that

$$\begin{array}{lcl} \text{NOT} & \leadsto & 2 \text{ transistors} \\ \text{NAND} & \leadsto & 4 \text{ transistors} \\ \text{NOR} & \leadsto & 4 \text{ transistors} \\ \text{AND} & \leadsto & 6 \text{ transistors} \\ \text{OR} & \leadsto & 6 \text{ transistors} \end{array}$$

i.e., since we *know* we can construct NOT, NAND and NOR using the stated number of MOSFET-based transistors, the best way to form AND and OR is simply to append a NOT to a NAND or NOR. This means we can just count the logic gate instances, and translate:

$$\begin{array}{lcl} 2 \cdot \text{NOT} + 2 \cdot \text{AND} + 1 \cdot \text{OR} & \leadsto & 22 \text{ transistors} \\ 1 \cdot \text{NOT} + 2 \cdot \text{AND} + 1 \cdot \text{OR} & \leadsto & 20 \text{ transistors} \\ 5 \cdot \text{NAND} & \leadsto & 20 \text{ transistors} \\ 4 \cdot \text{NAND} & \leadsto & 16 \text{ transistors} \end{array}$$

Based on this 16 is the correct answer, although keep in mind it *might* be possible to do better based on using a different set of options (for XOR) and assumptions.

S22. This is quite a tricky question, in the sense there are several plausible answers: selecting between them really needs some justification, which of course is impossible with a multiple choice question! It is important to keep in mind that the question focuses on design of some behaviour based on transistors, *not* precision wrt. manufacture of the design. Put another way, the question is intentionally pitched at a high-level, with the various caveats attempting to limit the possible answers:

a It might *seem* possible to use 0 transistors, in that one can just connect x directly to r. This may realise the pass through behaviour required, but does not impose a delay (and more generally does not satisfy use-cases for current or voltage buffers) so is not really valid.

b One could implement a buffer using 1 N-MOSFET transistor, say, in series with a pull-down resistor: the idea is that when $x = V_{dd}$ the transistor connects r to V_{dd}, otherwise the resistor pulls r down to V_{ss}.

However, the material provided does not cover use of resistors: the question caters for this by asking for an implementation using only transistors, and not listing 1 as an answer! This is justified further by the fact by placing an emphasis on CMOS, using only an N-MOSFET might seem confusing therefore (given the argument that N- and P-MOSFETs occur in pairs as part of a pull-down and pull-up network). So although this is arguably the best answer, it is not the expected one!

c One could start by considering a NOT gate implementation, which will clearly invert x. It does so via a P-MOSFET that connects V_{dd} to r, and an N-MOSFET that connects V_{ss} to r. As such, when $x = V_{ss}$ (resp. $x = V_{dd}$) the P-MOSFET will be connected and N-MOSFET disconnected (resp. vice versa) meaning that $r = V_{dd} \simeq \neg x$ (resp. $r = V_{ss} \simeq \neg x$). A somewhat simple observation is that if we swap the P- and N-MOSFETs, we end up with a buffer. That is, if the P-MOSFET connects V_{ss} to r and the N-MOSFET connects V_{dd} to r, then the behaviour swaps st. if $x = V_{dd}$ (resp. $x = V_{ss}$) then $r = V_{dd} \simeq x$ (resp. $r = V_{ss} \simeq x$).

So 2 transistors is a reasonable answer *if* we assume it is possible to organise them this way. In reality, this is debatable: it is the *opposite* of normal pull-down and pull-up networks connected to V_{dd} and V_{ss}, so may not be reasonable under the constraints of a given manufacturing process. However, the question is careful to ask for an unconstrained organisation for transistors, so the assumption is allowed.

d Finally, one could simply use two NOT gate implementations in series with each other: this inverts x twice, computing $r = \neg\neg x = x$ using 4 transistors. This approach needs no assumptions, but on the other hand is obviously not very efficient wrt. the number of transistors used.

In summary then, 2 transistors is a correct (or the expected) answer in this case (although you could quite reasonably argue for other answers).

S23. CMOS combines *complementary* transistor types: one N-MOSFET, and one P-MOSFET. Since these transistor behave in a complementary manner, it is always that case that one will be active and one inactive. This produces an attractive feature wrt. power, in that static consumption (i.e., when there is no chance in state) is very low. The dynamic power consumption (i.e., when there is a change of state) is of course higher, but occurs only when the inputs cause switching activity.

So, at a high-level at least, we *could* argue the highest consumption is likely when the initial value differs the most from stored value. That is, we argue that the highest switching activity occurs when the largest number of bits stored by the register change. We can measure this using the Hamming distance between initial and stored value: given

$$\begin{array}{rcrcl} t_0 &=& DEAD_{(16)} &=& 1101111010101101_{(2)} \\ t_1 &=& BEEF_{(16)} &=& 1011111011101111_{(2)} \\ t_2 &=& F00D_{(16)} &=& 1111000000001101_{(2)} \\ t_3 &=& 1234_{(16)} &=& 0001001000110100_{(2)} \\ t_4 &=& FFFF_{(16)} &=& 1111111111111111_{(2)} \\ t_5 &=& 0000_{(16)} &=& 0000000000000000_{(2)} \end{array}$$

and recalling that

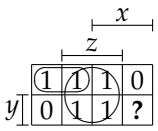
$$\mathcal{D}(x, y) = \sum_{i=0}^{i<16} x_i \oplus y_i,$$

we can compute

$$\begin{array}{rcr} \mathcal{D}(t_0, t_1) &=& 4 \\ \mathcal{D}(t_0, t_2) &=& 6 \\ \mathcal{D}(t_0, t_3) &=& 8 \\ \mathcal{D}(t_0, t_4) &=& 5 \\ \mathcal{D}(t_0, t_5) &=& 11 \end{array}$$

This means that based on our argument above, the correct answer would be $0000_{(16)}$. A more precise answer requires a much more detailed analysis of the component: we would need to assess the implementation in terms of individual transistors, and compute the so-called toggle count, i.e., switching activity, at that level (rather than via the assumption about toggling the bits stored).

S24. From the truth table, one can form a Karnaugh map



which includes two groups: unlike some other examples, the don't care in this case is *uncovered* (i.e., we treat it as 0) since we cannot make fewer or larger groups by covering it (i.e., treating it as 1). Even so, translating each group into a term yields the SoP expression

$$r = f(x, y, z) = (\neg x \wedge \neg y) \vee z$$

which is the correct answer: the fact this is the only one in SoP form at least *hints* at the correct answer even without going through the above.

S25. Given an l-bit control signal c, the demultiplexer can select between at most 2^l outputs: we treat c as an unsigned, l-bit integer which will clearly range in value between 0 and $2^l - 1$. In general, we want an l st. $2^l \geq m$ so each output can be specified; typically m is m a power-of-two, since this matches the maximum number of outputs that can be specified. However, in this case we have $m = 5$.

Since $2^2 = 4 < 5$ and $2^3 = 8 > 5$ we know a 2-bit control signal is not enough (it cannot select r_4 as $0 \leq c < 4$), but a 3-bit control signal is (although it *could* cope with upto $m = 8$, and since $0 \leq c < 8$ select r_5, r_6 and r_7 if they existed). In summary then, $l = 3$ is the correct answer.

S26. To answer this question, the idea is that we

- use a 3-level cascade to produce an 8-input, 1-bit multiplexer, then
- replicate this 8 times to produce an 8-input, 8-bit multiplexer

using a generic 2-input, 1-bit multiplexer whose behaviour can be described as

$$r = \begin{cases} x & \text{if } c = 0 \\ y & \text{otherwise} \end{cases}$$

i.e., it selects the input x (i.e., connects the output r to x) if the control signal $c = 0$, and selects the input y (i.e., connects the output r to y) if the control signal $c = 1$.

Imagine the 8, 1-bit inputs are named s, t, u, v, w, x, y and z and there is a 3-bit control signal c (to select between $2^3 = 8$ inputs). The cascade of 2-input, 1-bit multiplexers is constructed as follows

$$t_0 = \begin{cases} s & \text{if } c_0 = 0 \\ t & \text{otherwise} \end{cases}$$

$$t_1 = \begin{cases} u & \text{if } c_0 = 0 \\ v & \text{otherwise} \end{cases}$$

$$t_2 = \begin{cases} w & \text{if } c_0 = 0 \\ x & \text{otherwise} \end{cases}$$

$$t_3 = \begin{cases} y & \text{if } c_0 = 0 \\ z & \text{otherwise} \end{cases}$$

$$t_4 = \begin{cases} t_0 & \text{if } c_1 = 0 \\ t_1 & \text{otherwise} \end{cases}$$

$$t_5 = \begin{cases} t_2 & \text{if } c_1 = 0 \\ t_3 & \text{otherwise} \end{cases}$$

$$r = \begin{cases} t_4 & \text{if } c_2 = 0 \\ t_5 & \text{otherwise} \end{cases}$$

noting there are 3 layers (i.e., they form a tree of depth 3). Using a table such as

c	c_2	c_1	c_0	t_0	t_1	t_2	t_3	t_4	t_5	r
0	0	0	0	s	u	w	y	s	w	s
1	0	0	1	t	v	x	z	t	x	t
2	0	1	0	s	u	w	y	u	y	u
3	0	1	1	t	v	x	z	v	z	v
4	1	0	0	s	u	w	y	s	w	w
5	1	0	1	t	v	x	z	t	x	x
6	1	1	0	s	u	w	y	u	y	y
7	1	1	1	t	v	x	z	v	z	z

makes it clear the c-th input is selected as the output r. Given such a component, we then just replicate it 8 times: the same control signal is used for each i-th replication, which then produces the i-th bit of r by selecting between the i-th bits of the 8-bit inputs s through to z.

We use 7 instances of the 2-input, 1-bit multiplexer for each of the cascades; there are 8 replicated cascades, so the correct answer is that we need $7 \cdot 8 = 56$ instances.

S27. First, notice that the critical path (or longest sequential path) runs through the 4 full-adder instances (as a result of the carry chain): the 3-rd (most-significant) instance cannot produce either $co = c_4$ or r_3 until the carry propagates from the 0-th (least-significant) instance, and so is dependent on $ci = c_0$, x_0, and y_0).

Next, we need some detail about each full-adder instance: the i-th such instance will compute the sum and carry-out

$$\begin{array}{lcl} r_i & = & x_i \oplus y_i \oplus c_i \\ c_{i+1} & = & (x_i \wedge y_i) \vee (x_i \wedge c_i) \vee (y_i \wedge c_i) \\ & = & (x_i \wedge y_i) \vee ((x_i \oplus y_i) \wedge c_i) \end{array}$$

from summands x_i and y_i, plus the carry-in c_i (where $c_0 = ci$ and $co = c_4$). There are two different options for c_{i+1}, so, for the quoted gate delays, we find the critical paths from input to output can be described as:

	Input(s)	Output(s)	Critical path	
Option 1	x_i, y_i	r_i	$T_{XOR} + T_{XOR}$	= 120ns
	c_i	r_i	T_{XOR}	= 60ns
	x_i, y_i	c_{i+1}	$T_{AND} + T_{IOR} + T_{IOR}$	= 60ns
	c_i	c_{i+1}	$T_{AND} + T_{IOR} + T_{IOR}$	= 60ns
Option 2	x_i, y_i	c_{i+1}	$T_{XOR} + T_{AND} + T_{IOR}$	= 100ns
	c_i	c_{i+1}	$T_{AND} + T_{IOR}$	= 40ns

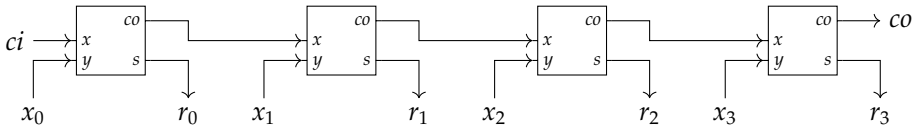
The correct answer will clearly differ depending on the option we select for c_{i+1}, but imagine we select the latter: irrespective of whether it is better or worse, it matches the lecture slide(s). As such, we can deduce the following:

- For the 0-th full-adder instance, it takes 100ns to generate c_1 from c_0, x_0, and y_0.
- For the 1-st full-adder instance, it takes 40ns to generate c_2 from c_1, The reason it is *not* 100ns, as you might expect, is because the gate computing $x_1 \oplus y_1$ can produce an output *before* c_1 is available; this means it does not contribute to the critical path.
- For the 2-nd full-adder instance, it takes 40ns to generate c_3 from c_2, x_2, and y_2. The reason it is *not* 100ns, as you might expect, is because the gate computing $x_2 \oplus y_2$ can produce an output *before* c_2 is available; this means it does not contribute to the critical path.
- For the 3-rd instance, it takes 40ns to generate c_4 from c_3, The reason it is *not* 100ns, as you might expect, is because the gate computing $x_3 \oplus y_3$ can produce an output *before* c_3 is available; this means it does not contribute to the critical path. Likewise, it takes 60ns to generate r_3 from c_3, x_3, and y_3; for the same reason as above, this is *not* 120ns as you might expect.

So the critical path is 220ns wrt. c_4 and 240ns wrt. r_3, and therefore 240ns overall. It turns out applying similar reasoning to the former option yields a slightly longer critical path of 280ns overall, because

- For the 0-th full-adder instance, it takes 100ns to generate c_1 from c_0, x_0, and y_0.
- For the 1-st full-adder instance, it takes 60ns to generate c_2 from c_1, x_1, and y_1.
- For the 2-nd full-adder instance, it takes 60ns to generate c_3 from c_2, x_2, and y_2.
- For the 3-rd instance, it takes 60ns to generate c_4 from c_3, x_3, and y_3, and 60ns to generate r_3 from c_3, x_3, and y_3. The reason it is *not* 120ns, as you might expect, is because the gate computing $x_3 \oplus y_3$ can produce an output *before* c_3 is available; this means it does not contribute to the critical path.

S28. The optimisation they are suggested is captured by the following:



Put another way, their idea implies the half-adders used no longer have a carry-in. This is not a problem in terms of computing *an* addition: we can still use the two available inputs (vs. three for a full-adder) to provide one operand (i.e., x_i, the i-th bit of x) plus a carry-in from the previous half-adder instance. However, clearly this is not the *same* addition as previously, since the input we *would* use to provide the other operand is no longer available. That is, we can select x as normal, but can no longer provide a y input (the half-adders use that for to propagate the carry).

Other than x, the only other input we can control is ci which acts as the overall carry-in to the addition. Since $ci \in \{0, 1\}$, this means $y = 0$ and $y = 1$ are the correct answers.

S29. 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

Crucially, this demonstrates that

$$x \oplus x \equiv 0$$

and

$$x \oplus 0 \equiv x,$$

and hence

$$x \oplus x \oplus y \equiv y$$

for any x and y.

Using these axioms, we can write expressions to describe the output of each XOR gate, and hence each overall output r_i. Inspecting the circuit from left-to-right and top-to-bottom, we can show

$$\begin{array}{lllllllll}
 & & t_0 & = & x_0 \oplus x_2 & & & & \\
 & & t_1 & = & x_1 \oplus x_3 & & & & \\
r_0 & = & t_2 & = & x_0 \oplus t_0 & = & x_0 \oplus x_0 \oplus x_2 & = & x_2 \\
r_1 & = & t_3 & = & x_1 \oplus t_1 & = & x_1 \oplus x_1 \oplus x_3 & = & x_3 \\
r_2 & = & t_4 & = & t_0 \oplus t_2 & = & x_0 \oplus x_2 \oplus x_2 & = & x_0 \\
r_3 & = & t_5 & = & t_1 \oplus t_3 & = & x_1 \oplus x_3 \oplus x_3 & = & x_1
\end{array}$$

st. it becomes clear $r_0 = x_2$, $r_1 = x_3$, $r_2 = x_0$, and $r_3 = x_1$, i.e., the most- and least-significant 2-bit halves of x are swapped over to produce r.

S30. We can deal with the statements one-by-one:

- DRAM cells are based on use of a capacitor, so only use one transistor (to access the capacitor), while an SRAM cells uses only transistors: a 6T SRAM cell design requires six for example, so certainly more than one.
- Both SRAM and DRAM cells store one bit of information: larger memories are constructed by replicating the cells, but it is not true that one or other *cell* can store more information.
- Their transistor-based design makes the access latency of SRAM cells low, i.e., they can be accessed quickly. In contrast, the need to (dis)charge the capacitor limits a DRAM cell in this respect; it is a fair assumption that DRAM cell access latency will be greater as a result.
- Their need to retain charge in the capacitor means DRAM cells need to be refreshed, since over time that charge will naturally leak (st. the stored value will "degrade" in some sense).

S31. For SR-type latch and flip-flop components, we expect at *least* S, R and *en* inputs and a Q output; in contrast, for the D-type (resp. T-type) latch and flip-flop components we expect D (resp. T) and *en* inputs and a Q output which match. Even based on the argument that we might have Q and $\neg Q$ from one or other, this at least makes the former less likely. Additionally, we might expect S and R to be used in a controlled way so as to avoid a problematic meta-stable state; there are instances where $x = y = z = 1$ and $x = y = z = 0$, so this control is not clearly being applied.

Narrowing down the choices further requires interpretation of the signal labelling and behaviour. While somewhat tricky, it should be clear that the value of z changes to match y while $x = 1$ and is unchanged when $x = 0$. Crucially, it is not the case that z changes to match y *only* at the point where x transitions from 0 to 1 (or 1 to 0): as shown in the right-hand portion of the waveform, z changes at *any* point that $x = 1$. As such, we infer the component is likely to be a level-triggered latch not an edge triggered flip-flop. Additionally, z does not toggle between 0 and 1 while $x = 1$; it matches whatever y is.

In summary therefore, i.e., having ruled out a) SR-type components, b) flip-flop components, and c) a T-type flip-flop, we conclude that the correct answer is a D-type latch: x represents the enable signal *en*, y represents the input D and z represents the output Q.

S32. First, recall that the truth table for NAND is

NAND		
x	y	r
0	0	1
0	1	1
1	0	1
1	1	0

If one or other (or both) of x and y is equal to 0 then the output r *must* be 1: only when x *and* y equal 1 do we get r equal to 0. This gives a useful way to ascertain the behaviour of the circuit, in the sense we can enumerate the set of consistent states it can be in:

a If $S = 0$ then the top NAND gate must output 1, and if $R = 0$ then the bottom NAND gate must output 1. That is, $S = 0$ and $R = 0$ means we have $Q = 1$ and $\neg Q = 1$.

b If $R = 0$ then the bottom NAND gate must output 1, so if $S = 1$ then the top NAND gate outputs $1 \overline{\wedge} 1 = 0$. That is, $S = 1$ and $R = 0$ means we have $Q = 0$ and $\neg Q = 1$.

c If $S = 0$ then the top NAND gate must output 1, so if $R = 1$ then the bottom NAND gate outputs $1 \overline{\wedge} 1 = 0$. That is, $S = 0$ and $R = 1$ means we have $Q = 1$ and $\neg Q = 0$.

d If $S = 1$ and $R = 1$ then one of *two* possibilities can apply: either

   i the top NAND gate outputs 0, meaning the bottom NAND gate outputs $0 \overline{\wedge} 1 = 1$ (which is consistent with the top gate computing $1 \overline{\wedge} 1 = 0$)

   ii the bottom NAND gate outputs 0, meaning the top NAND gate outputs $0 \overline{\wedge} 1 = 1$ (which is consistent with the bottom gate computing $1 \overline{\wedge} 1 = 0$).

Put another way, $S = R = 0$ is the meta-stable state (which is inconsistent in the sense wrt. $Q = \neg Q$: they should differ) and $S = R = 1$ is the storage state (which retains whatever values Q and $\neg Q$ have already); $S = 0$ and $R = 1$ sets $Q = 1$, while $S = 1$ and $R = 0$ resets $Q = 0$, *whatever* the currently value of Q is. As such, the second excitation table is correct (the first one is for a NOR-based SR-latch), noting this is sort of the *inverse* of a NOR-based SR-latch wrt. the meaning of S and R.

S33. There are 2^{16} addressable bytes, meaning a 16-bit address needs to be supplied. However, in contrast to an SRAM memory, a DRAM memory will normally use a 2-step (or more, potentially) approach: half the address is supplied by each of the steps (under control of row and column address strobe signals), which requires only half the number of address pins.

The memory stores bytes, i.e., 8-bit elements, so we expect there to be 8 duplicated arrays each consisting of 65536 cells. Overall, there will be $8 \cdot 65536 = 524288$ cells. So, in summary, an answer of 8-bit address pins, and 524288 cells is correct; the alternative of 16-bit address pins, and 524288 cells is not *wrong* per se, but certainly less common in practice.

S34. Various clues should (in combination) be strong enough to hint that

$$\begin{aligned} \alpha &\mapsto \text{row address buffer} \\ \beta &\mapsto \text{column address buffer} \\ \gamma &\mapsto \text{row address decoder} \\ \delta &\mapsto \text{column address decoder} \end{aligned}$$

is the correct answer. For example, note that:

- α is provided input from A_i (the address pins), and is controlled (indirectly) by RAS: this is the row address strobe. As such, this is likely to be the row address buffer.
- β is provided input from A_i (the address pins), and is controlled (indirectly) by CAS: this is the column address strobe. As such, this is likely to be the column address buffer.
- γ is taking the content of α and controlling signals on the left-hand side (horizontal orientation) of the memory array, suggesting it computes the row address: it is likely to be the row address decoder.
- δ is taking the content of β and controlling signals on the top side (vertical orientation) of the memory array, suggesting it computes the column address: it is likely to be the column address decoder.

S35. This is quite a tricky question, but the central feature to note, for both multiplexers in the circuit, is the loop between output and (an) input:

- in the left-hand case the loop connects the multiplexer output, say t_0, to the y input, whereas
- in the right-hand case the loop connects the multiplexer output, say t_1, to the x input.

In both cases, this allows a 1-bit value to be *stored* by "holding" it in the loop. Put another way, when $b = 0$ then $t_0 = a$ since the x input is selected; when $b = 1$, however, whatever value t_0 has is fed back into the multiplexer. This is conceptually similar to how SRAM cells work, for example. In that case we had a loop through two NOT gates that acted to refresh (or reinforce) the stored value, plus extra access transistors. More formally, in each case the loop results in bistability. Focusing on the left-hand multiplexer as an example, if $b = 1$ then either of the states $t_0 = 0$ or $t_0 = 1$ (meaning $y = 0$ and $y = 1$) is stable (meaning it will not transition into the other state without a stimulus), so the value is retained until updated (or lost if the power supply is removed).

In this case, the left-hand and right-hand multiplexers are arranged in a standard manner; they are termed the master and slave respectively. The idea is basically that b acts as an enable signal (typically it will be a clock), operating both the master and slave in one of two modes. Per the above,

- when $b = 0$ the master passes a through to t_0 whereas the slave is in storage mode,
- when $b = 1$ the master is in storage mode whereas the slave passes t_0 through to t_1

To understand their combined behaviour, focus on the instant in time when a positive *edge* occurs on b and imagine that $a = \alpha$ for a value $\alpha \in \{0, 1\}$. Before the edge, the master and slave will be in pass-through and storage mode respectively since $b = 0$. This means $t_0 = a = \alpha$ and $c = t_1$. At the instance the edge occurs on b, $t_0 = \alpha$. Since the master flips into storage mode, this value is retained (i.e., fed back around the loop into the y input) because $b = 1$: changes to a are irrelevant. Simultaneously, the slave flips into pass-through mode st. $c = \alpha$. Then, at some point, there is a negative edge on b meaning both master and slave flip back to the opposite mode. Given $c = t_1 = \alpha$ at this instant, the fact the slave is now in storage mode (again) means it retains the value of α. Likewise, since the master is in pass-through mode, any change to a is reflected in t_0 (but since the slave is in storage mode, this is irrelevant to the value, namely α, it retains). Diagrammatically, this can be viewed as in Figure 1.

A somewhat reasonable analogy is that the master and slave act as latches (each being level triggered), but their *combination* forms a flip-flop that passes α along a 2-step "conveyor belt". First the master (on the positive edge) stores whatever α us passed as input by the user; this is subsequently passed to the slave (on the negative edge) which stores and (in some sense) "protects" it from subsequent changes to a (until another positive edge on b). So the correct answer is that the circuit represents a flip-flop, i.e., an edge triggered storage cell: a is the flip-flop input, c is the output (i.e., the value stored) and b is the control signal.

Throughout the following, keep in mind that three main component groups can be identified in the implementation: read from bottom-to-top, these are

- an input register (bottom),
- some combinatorial logic (middle), and
- an output register (top).

We know this is an FSM, so we expect the input register to hold the current state and the combinatorial logic to compute both the transition and output functions. Specifically, the input register and x are provided as input to combinatorial logic (in the middle-left and -center) that represents the transition function. It computes the next state, then stored in the output register; the rest of this logic (in the middle-right) clearly represents the output function, since it computes r.

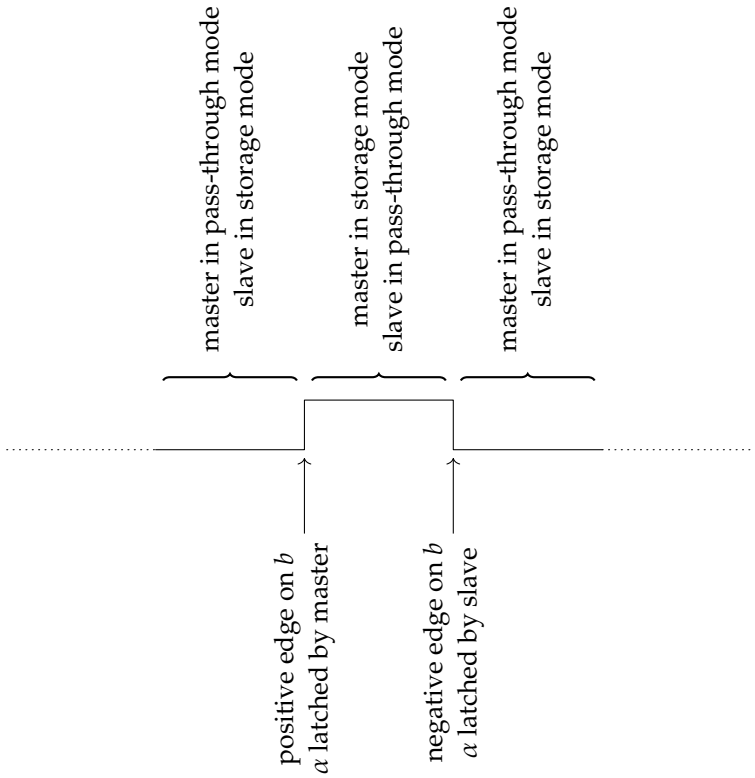


Figure 1: *A time line illustrating behaviour of a multiplexer-based, master-slave flip-flop.*

S36. a Saying a signal is digital is the same as saying it takes the values 0 and 1 only; for a clock signal, this is the same as saying it the form is a perfect square wave.

In practice this is difficult to achieve since transitions between 0 and 1 cannot be perfectly instantaneous. This implies each edge has a slope, however shallow this is. Even so, the same issue is true for all digital logic components: if the inputs to an AND gate are neither 0 or 1 (or their associated voltage levels), the output is undefined. So it is also fair to say this is a requirement for Φ_1 and Φ_2, at least as far as is practical.

b Φ_1 and Φ_2 are said to be non-overlapping in the sense a positive level on one always occurs at the same time as a negative level on the other.

If this were *not* true, e.g., $\Phi_1 = \Phi_2$, then a "loop" would form during the overlap: the output register would be updated with whatever was computed by the combinatorial logic, which is fed by the input register *also* being updated at the same time by the output register. This would likely result in a malfunction of some sort: the input register could not settle into a stable state, for example.

c To gate *any* signal, Φ_1 and Φ_2 included, means to (conditionally) disable them. This is typically realised by adding extra logic, e.g., an AND gate, so the clock signal can be forced to 0.

This might be useful; it allows one to disable latch updates and so "pause" the FSM (e.g., to save power when idle). However, doing so is not a requirement and not evident in this implementation.

d In general, the concept of skew describes a situation where the clock signal arrives at two components at different times; with a 2-phase clock, we might *also* find cases where Φ_1 and Φ_2 can arrive at the same component at different times.

This is clearly undesirable, since the clock is meant to synchronise the component. If they were *not* synchronised then malfunction is the likely result: one latch might be updated at a different time, and hence with an unrelated value, than another, for example.

e The duty cycle of a clock (signal) is the percentage of each clock period in which it has a positive level. For example, saying that Φ_1 has a duty cycle of 33% is the same as saying $\Phi_1 = 1$ for a third of the time (and hence $\Phi_1 = 0$ for two thirds of the time).

Here, there is no reason the duty cycle of Φ_1 and Φ_2 must be 33%. If is was 40%, for example, the implementation would still function correctly. Assuming Φ_1 and Φ_2 have the same form, then of course it must be true their duty cycles are less than 50% otherwise they would have to overlap. Other than that, however, getting closer to 50% just means less separation between their positive levels.

S37. This is not a *trick* question per se, but the correct answer is that the register might hold any 2-bit value when powered-on. Although the register must settle into *some* state, it is not clear how we could predict what this will be: the stored value is basically random, or more precisely determined by physics of the underlying implementation and fabrication process.

As an aside, this FSM has an related, unattractive design feature: there is no reset input. This means there is no way to enforce a start state, i.e., whatever value is held in the register at power-on is used as the start state: the only way to alter this, and hence make the FSM function as required, is to power-cycle the implementation and hope the initial stored value is the required start state!

S38. Partly as a result of the multiple-choice format, this question might seem odd. Exactly the same *concepts* are involved, but the *steps* are the opposite way around: rather than derive an implementation from a given specification, it asks you to reverse engineer a specification from a given implementation.

Each of the registers constitutes 2 D-type latches, so the FSM can be in at most $2^2 = 4$ possible states. Denoting the current (resp. next) state $Q = \langle Q_0, Q_1 \rangle$ (resp. $Q' = \langle Q'_0, Q'_1 \rangle$), we can write an expression for the next state and output in terms of the current state and input: this basically just means translating the logic gate symbols into a Mathematical form. That is, we can write

$$\begin{array}{lcllll} Q'_1 & = & (Q_1 \wedge Q_0) & \vee & (x \wedge Q_1) & \vee \quad (x \wedge Q_0) \\ Q'_0 & = & (Q_1 \wedge Q_0) & & & \vee \quad (x \wedge \neg Q_0) \\ r & = & (Q_1 \wedge Q_0) & \vee & (x \wedge Q_1) & \end{array}$$

By enumerating the possible values of x, Q_0 and Q_1 we find

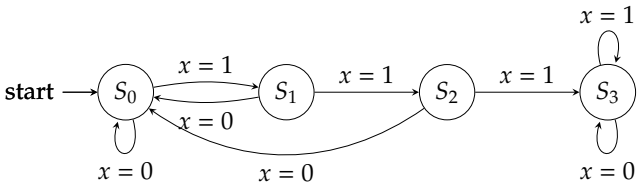
x	Q_1	Q_0	Q'_1	Q'_0	r
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	1	1

i.e., we reverse engineer the transition and output functions, expressed as a truth table. For example, if the current state is $Q = \langle 0, 0 \rangle$ and the input is $x = 1$ then we can see the next state will be $Q' = \langle 1, 0 \rangle$ and the output will be $r = 0$.

The truth table encodes the same information as a diagrammatic alternative. The only difference is the use of a concrete representation, rather than an abstract label for each state. We *need* the former, because of course the implementation stores and computes Boolean values: it cannot deal with a label such as S_0 or S_3 means unless we give that label a value. So imagine we make such an (reverse) assignment, namely

$$\begin{array}{lcl} \langle 0, 0 \rangle & \mapsto & S_0 \\ \langle 1, 0 \rangle & \mapsto & S_1 \\ \langle 0, 1 \rangle & \mapsto & S_2 \\ \langle 1, 1 \rangle & \mapsto & S_3 \end{array}$$

Now we can say, for example that if the current state is S_0 and the input is $x = 1$ then the next state will be S_1 and the output will be $r = 0$. This makes drawing the diagrammatic alternative a little easier: if we draw 4 nodes for the four states, we join them with edges based on rows of the truth table. The end result, and correct answer, is



S39. The central difference between Mealy- and Moore-type FSMs stems from how the output function is defined. In the former, the output is a function of the current state *and* input; in the latter, *only* the current state is relevant. For a set of states S and input and output alphabets Σ and Γ, this means for a Mealy-type FSM we have

$$\omega : S \times \Sigma \rightarrow \Gamma$$

whereas for a Moore-type FSM we have

$$\omega : S \rightarrow \Gamma.$$

For this FSM, we already know from the previous question that the output is described by

$$r = (Q_1 \wedge Q_0) \vee (x \wedge Q_1).$$

As such, it should be obvious $r = \omega(Q, x)$ is a function of both Q (the current state) and x (the input): this is therefore a Mealy-type FSM.

S40. The behaviour of this FSM can be described as repeated iteration over two steps under control of the clock. That is, it repeatedly does

- Step #1:
  - the combinatorial logic compute the next state $Q' = \delta(Q, x)$ and output $r = \omega(Q, x)$, and
  - the output register latches Q'.

  Note that the critical path of this step is that from the the Q output of the input register to the Q output of the output register.
- Step #2:
  - the input register latches Q' as Q.

  Note that the critical path of this step is that from the the Q output of the output register to the Q output of the input register.

For example, the first step occurs during the period when $\Phi_1 = 1$ and the second when $\Phi_2 = 1$. Within every clock period, i.e., within the "time limit" represented by ρ, *both* steps must be completed. Therefore, we can say

$$\rho \geq (T_{\text{logic}} + T_{\text{latch}}) + (T_{\text{latch}})$$

where T_{latch} and T_{logic} are the critical paths associated with a D-type latch and the combinatorial logic respectively.

Note the critical path runs through the middle-left *or* middle-center of the combinatorial logic: although the former includes a NOT gate, the latter includes a 3- versus 2-input OR gate. Either way the delay is 50ns, so we can write

$$\begin{aligned} \rho &\geq 50 + 60 + 60\text{ns} \\ &\geq 170\text{ns} \end{aligned}$$

then compute the maximum clock frequency as

$$\begin{aligned} f\text{max} &= 1/\rho \\ &= 1/170\text{ns} \\ &\simeq 5.9\text{MHz} \end{aligned}$$

S41. From the definition of the transition function (above), it *should* be clear the FSM will progress from left-to-right as consecutive inputs $x = 1$ are encountered. Moreover, encountering an input of $x = 0$ means restarting in state S_0, and when eventually the FSM reaches state S_3 it stays in that state (whether $x = 1$ or $x = 0$). So it basically counts the number of consecutive times $x = 1$ until that count is 3 (if it is in state S_i, then the count is i). This already provides the correct answer, but is further confirmed by inspecting the output function: $r = 1$ when the FSM is in state S_3, i.e., when the count is 3.

S42. a Using a Karnaugh map, for example, one can produce the result

$$r = f(x, y, z) = y \vee (z \wedge \neg x) \vee (x \wedge \neg z)$$

which, by inspection, gives

			f			
x	y	z	y	$z \wedge \neg x$	$x \wedge \neg z$	r
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	1	0	0	1
0	1	1	1	1	0	1
1	0	0	0	0	1	1
1	0	1	0	0	0	0
1	1	0	1	0	1	1
1	1	1	1	0	0	1

However, notice that the sub-expression

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

can be simplified to

$$z \oplus x$$

so per the question, the simplest implementation of f is

$$r = f(x, y, z) = y \vee (z \oplus x).$$

b First, note that the following identities can be applied:

$$\begin{array}{lcl} \neg x & \equiv & x \mathbin{\overline{\wedge}} x \\ x \vee y & \equiv & (x \mathbin{\overline{\wedge}} x) \mathbin{\overline{\wedge}} (y \mathbin{\overline{\wedge}} y) \\ x \wedge y & \equiv & (x \mathbin{\overline{\wedge}} y) \mathbin{\overline{\wedge}} (x \mathbin{\overline{\wedge}} y) \end{array}$$

As such, we can write

$$\neg(x \vee y) \equiv ((x \mathbin{\overline{\wedge}} x) \mathbin{\overline{\wedge}} (y \mathbin{\overline{\wedge}} y)) \mathbin{\overline{\wedge}} ((x \mathbin{\overline{\wedge}} x) \mathbin{\overline{\wedge}} (y \mathbin{\overline{\wedge}} y)).$$

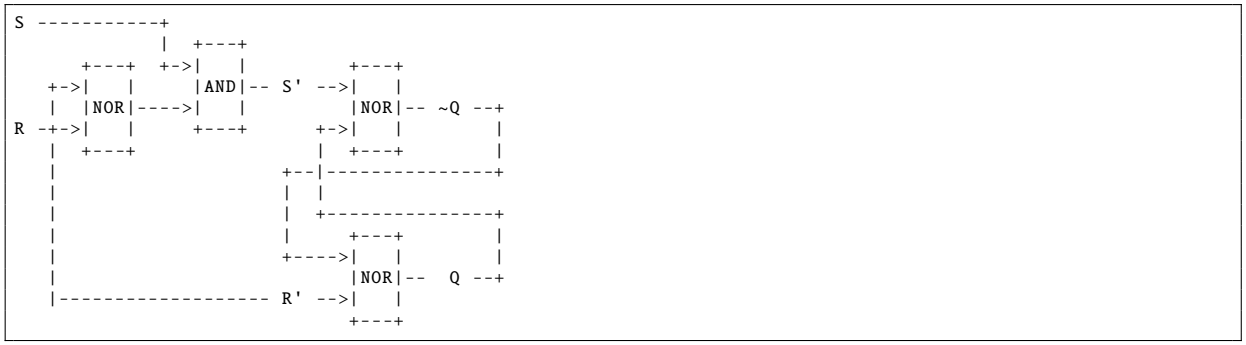
c The excitation table of a standard SR latch is

		Current		*Next*	
S	R	Q	$\neg Q$	Q'	$\neg Q'$
0	0	0	1	0	1
0	0	1	0	1	0
0	1	**?**	**?**	0	1
1	0	**?**	**?**	1	0
1	1	**?**	**?**	**?**	**?**

meaning the reset-dominate alteration gives

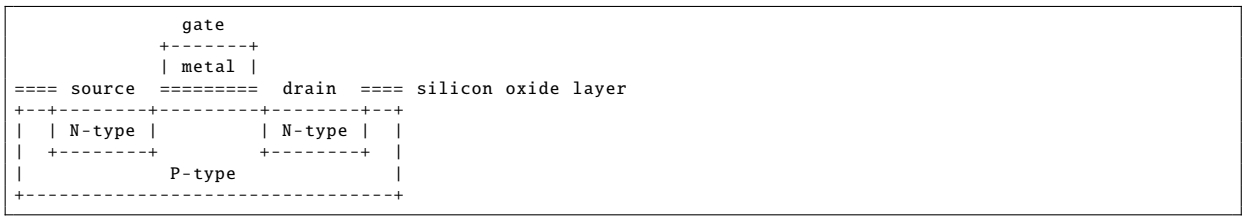
		Current		*Next*	
S	R	Q	$\neg Q$	Q'	$\neg Q'$
0	0	0	1	0	1
0	0	1	0	1	0
0	1	**?**	**?**	0	1
1	0	**?**	**?**	1	0
1	1	**?**	**?**	0	1

Given the inputs S and R, the circuit can be constructed by using a standard SR latch with two cross-coupled NOR gates whose inputs are S' and R'. We simply set $R' = R$ and $S' = \neg R \wedge S$ so S' is only 1 when $R = 0$ and $S = 1$: if $R = 1$ (including the case when both $R = 1$ and $S = 1$), the latch it reset since $S = 0$. The circuit is as follows:



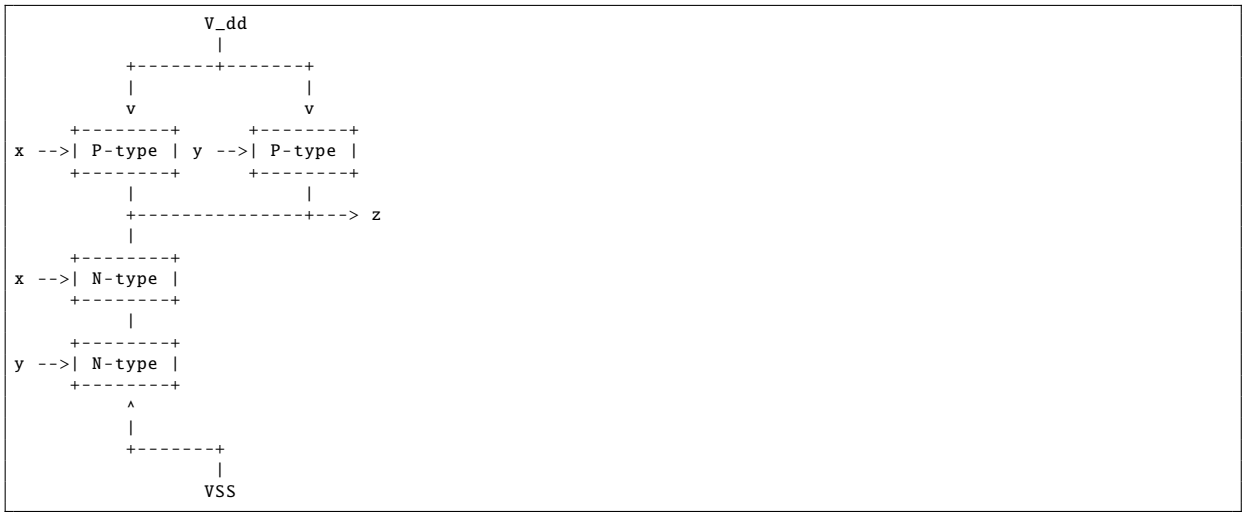
d A wide range of answers are clearly possible. Obvious examples include physical size, and power consumption or heat dissipation. Other variants include worst-case versus average-case versions of each metric, for example in the case of efficiency.

S43. a MOSFET transistors work by sandwiching together N-type and P-type semiconductor layers. The different types of layer are doped with different substances to create more holes or more electrons. For example, in an N-type MOSFET the layers are constructed as follows



with additional layers of silicon oxide and metal. There are three terminals on the transistor. Roughly speaking, applying a voltage to the gate creates a channel between the source and drain through which charge can flow. Thus the device acts like a switch: when the gate voltage is high, there is a flow of charge but when it is low there is little flow of charge. A P-type MOSFET swaps the roles of N-type and P-type semiconductor and hence implements the opposite switching behaviour.

b One can construct an NAND to compute $z = x \overline{\wedge} y$ gate from such transistors as follows:

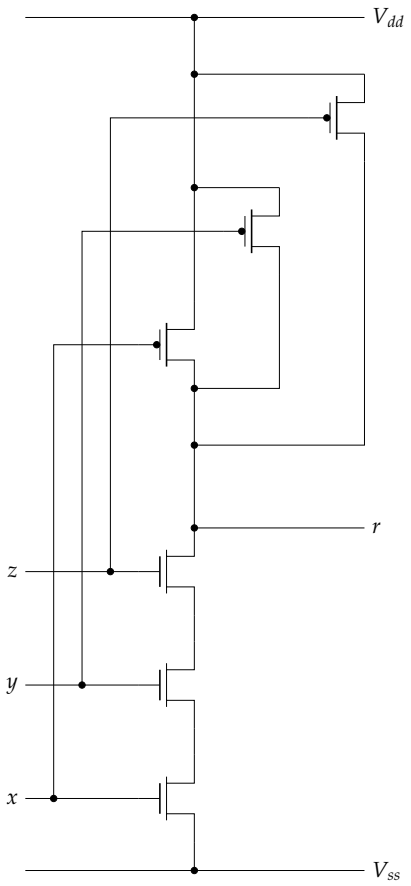


If x and y are connected to V_{ss} then both top P-type transistors will be connected, and both bottom N-type transistors will be disconnected; r will be connected to V_{dd}. If x and y are connected to V_{dd} and V_{ss} respectively then the right-most P-type transistor will be connected, and both lower-most N-type transistor will be disconnected; r will be connected to V_{dd}. If x and y are connected to V_{ss} and V_{dd} respectively then the left-most P-type transistor will be connected, and both upper-most N-type transistor will be disconnected; r will be connected to V_{dd}. If x and y are connected to V_{dd} then both top P-type transistors will be disconnected, and both bottom N-type transistors will be connected; r will be connected to V_{ss}. In short, the behaviour we get is described by

x	y	r
V_{ss}	V_{ss}	V_{dd}
V_{ss}	V_{dd}	V_{dd}
V_{dd}	V_{ss}	V_{dd}
V_{dd}	V_{dd}	V_{ss}

which, if we substitute 0 and 1 for V_{ss} and V_{dd}, is matches that of the NAND operation.

S44. This question is a lot easier than it sounds; basically we just add two extra transistors (one P-MOSFET and one N-MOSFET) to implement a similar high-level approach. That is, we want r connected to V_{ss} only when each of x, y *and* z are connected to V_{dd}; this means the bottom, N-MOSFETs are in series. If *any* of x, y or z are connected to V_{ss}, we want r connected to V_{dd}; this means the top, P-MOSFETs are in parallel. Diagrammatically, the result is as follows:



S45. This is quite an open-ended question, but basically it asks for high-level explanations only. As such, some example answers include the following:

a CMOS transistors are constructed from atomic-level understanding and manipulation; the immutable size of atoms therefore acts as a fundamental limit on the size of any CMOS-based transistor.

b Feature scaling improves the operational efficiency of transistors, simply because smaller features reduce delay. Beyond this however, one must utilise the extra transistors to achieve some useful task if computational efficiency is to scale as well: improvements to an architecture or design are often required, for instance, to exploit parallelism and so on.

c Even assuming the transistors available can be harnessed to improve computational efficiency, this has implications: more transistors within a fixed size area will increase power consumption and also heat dissipation for example, both of which act as limits even if managed (e.g., via aggressive forms of cooling).

d On one hand, smaller transistors mean less cost per-transistor: with a fixed number of transistors, their area and manufacturing cost will decrease. With a fixed sized area and hence more transistors in it however, this probably means increase defect rate during manufacture. The resulting cost implication could act as an economic limit to transistor size.

S46. a The most basic interpretation (i.e., not really doing any grouping using Karnaugh maps but just picking out each cell with a 1 in it) generates the following SoP equations

$$\begin{aligned} e &= (\neg a \wedge \neg b \wedge c \wedge \neg d) \vee (a \wedge \neg b \wedge \neg c \wedge \neg d) \vee (\neg a \wedge b \wedge \neg c \wedge d) \\ f &= (\neg a \wedge \neg b \wedge \neg c \wedge d) \vee (\neg a \wedge b \wedge \neg c \wedge \neg d) \vee (a \wedge \neg b \wedge c \wedge \neg d) \end{aligned}$$

b From the basic SoP equations, we can use the don't care states to eliminate some of the terms to get

$$\begin{aligned} e &= (\neg a \wedge \neg b \wedge c) \vee (a \wedge \neg c \wedge \neg d) \vee (b \wedge d) \\ f &= (\neg a \wedge \neg b \wedge d) \vee (b \wedge \neg c \wedge \neg d) \vee (a \wedge c) \end{aligned}$$

then, we can share both the terms $\neg a \wedge \neg b$ and $\neg c \wedge \neg d$ since they occur in e and f.

S47. Simply transcribing the truth table into a suitable Karnaugh map gives

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

from which we can derive the SoP expression

$$r = (\neg y \wedge z) \vee (\neg x \wedge \neg z).$$

S48. Define $\overline{\wedge}$ as the NAND operation with the truth table:

x	y	$x \mathbin{\overline{\wedge}} y$
0	0	1
0	1	1
1	0	1
1	1	0

Using NAND, we can implement NOT, AND and OR as follows:

$$\begin{aligned} \neg x &= x \mathbin{\overline{\wedge}} x \\ x \wedge y &= (x \mathbin{\overline{\wedge}} y) \mathbin{\overline{\wedge}} (x \mathbin{\overline{\wedge}} y) \\ x \vee y &= (x \mathbin{\overline{\wedge}} x) \mathbin{\overline{\wedge}} (y \mathbin{\overline{\wedge}} y) \end{aligned}$$

To prove this works, we can construct truth tables for the expressions and compare the results with what we would expect; for NOT we have:

x	$x \mathbin{\overline{\wedge}} x$	$\neg x$
0	1	1
1	1	0

while for AND we have:

x	y	$x \mathbin{\overline{\wedge}} y$	$(x \mathbin{\overline{\wedge}} y) \mathbin{\overline{\wedge}} (x \mathbin{\overline{\wedge}} y)$	$x \wedge y$
0	0	1	0	0
0	1	1	0	0
1	0	1	0	0
1	1	0	1	1

and finally for OR we have:

x	y	$x \mathbin{\overline{\wedge}} x$	$y \mathbin{\overline{\wedge}} y$	$(x \mathbin{\overline{\wedge}} y) \mathbin{\overline{\wedge}} (x \mathbin{\overline{\wedge}} y)$	$x \vee y$
0	0	1	1	0	0
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	1	1

such that it should be clear all three are correct.

S49. Conventionally a 4-input, 1-bit multiplexer might be described using a truth table such as the following:

c_1	c_0	w	x	y	z	r
0	0	?	?	?	0	0
0	0	?	?	?	1	1
0	1	?	?	0	?	0
0	1	?	?	1	?	1
1	0	?	0	?	?	0
1	0	?	1	?	?	1
1	1	0	?	?	?	0
1	1	1	?	?	?	1

This assumes that there are four inputs, namely w, x, y and z, with two further control signals c_1 and c_0 deciding which of them provides the output r. However, another valid way to write the same thing would be

c_1	c_0	r
0	0	w
0	1	x
1	0	y
1	1	z

This reformulation describes a 2-input, 1-output Boolean function whose behaviour is selected by fixing w, x, y and z, i.e., connecting each of them directly to either 0 or 1. For instance, if $w = x = y = 0$ and $z = 1$ then the truth table becomes

c_1	c_0	r
0	0	$w = 0$
0	1	$x = 0$
1	0	$y = 0$
1	1	$z = 1$

which is of course the same as AND. So depending on how w, x, y and z are fixed (on a per-instance basis) we can form *any* 2-input, 1-output Boolean function; this includes NAND and NOR, which we know are universal, meaning the multiplexer is also universal.

S50. a The expression for this circuit can be written as

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

which yields the Karnaugh map

c \ d \ a \ b	00	01	11	10
00	1	0	0	1
01	1	1	1	1
11	0	1	1	0
10	1	1	1	1

and from which we can derive a simplified SoP form for e, namely

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

b The advantages of this expression over the original are that is is simpler, i.e., contains less terms and hence needs less gates for implementation, and shows that the input a is essentially redundant. We have probably also reduced the critical path through the circuit since it is more shallow. The disadvantages are that we still potentially have some glitching due to the differing delays through paths in the circuit, although these existed before as well, and the large propagation delay.

c The longest sequential path through the circuit goes through a NOT gate, two AND gates and two OR gates; the critical path is thus 90ns long. This time bounds how fast we can used it in a clocked system since the clock period must be at least 90ns. So the shortest clock period would be 90ns, meaning the clock ticks about 11111111 times a second (or at about 11MHz).

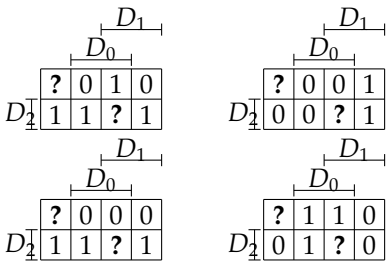
S51. a Examining the behaviour required, we can construct the following truth table:

D_2	D_1	D_0	L_8	L_7	L_6	L_5	L_4	L_3	L_2	L_1	L_0
0	0	0	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0	0	1	0	0	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0	0	1	0
0	1	1	1	0	0	0	1	0	0	0	1
1	0	0	1	0	1	0	0	0	1	0	1
1	0	1	1	0	1	0	1	0	1	0	1
1	1	0	1	1	1	0	0	0	1	1	1
1	1	1	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**

Note that

$$\begin{aligned} L_3 &= 0 \\ L_5 &= 0 \\ L_6 &= L_2 \\ L_7 &= L_1 \\ L_8 &= L_0 \end{aligned}$$

so actually we only need expressions for $L_{0...2}$ and L_4, and that don't care states are used to capture the idea that $D = 0$ and $D = 7$ never occur. The resulting four Karnaugh maps



can be translated into the expressions:

$$\begin{aligned} L_0 &= D_2 \vee (D_1 \wedge D_0) \\ L_1 &= (D_1 \wedge \neg D_0) \\ L_2 &= D_2 \\ L_4 &= D_0 \end{aligned}$$

b All the LEDs can be driven in parallel, i.e., the critical path relates to the single expression whose critical path is the most. $L_{2...6}$ have no logic involved, so we can discount them immediately. Of the two remaining LEDs, we find

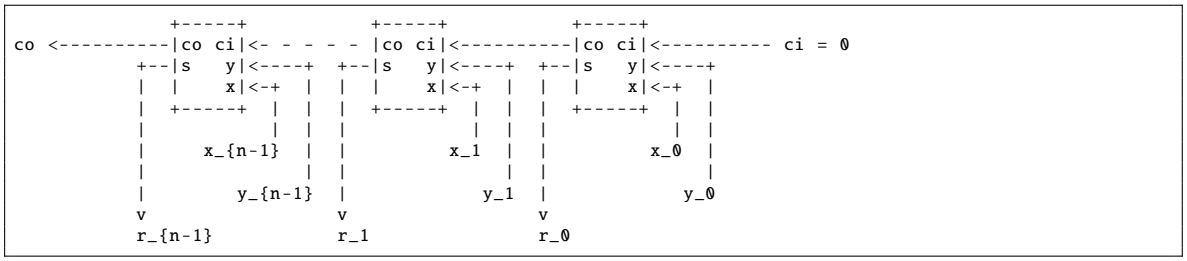
$$\begin{aligned} L_0 &\rightsquigarrow 20\text{ns} + 20\text{ns} \\ L_1 &\rightsquigarrow 10\text{ns} + 20\text{ns} \end{aligned}$$

hence L_0 represents the critical path of 40ns. Thus if one throw takes 40ns, we can perform

$$\frac{1\text{s}}{40\text{ns}} = \frac{1 \cdot 10^9\text{ns}}{40\text{ns}} = 25000000$$

throws per-second. Which is quite a lot, and certainly too many to actually see with the human eye!

c i A rough block diagram would resemble



ii If we sum 8 values $1 \leq x_i \leq 6$, where x_i is the i-th throw (or i-th value of D supplied), then the maximum total is $8 \cdot 6 = 48$. We can represent this in 6 bits, hence $n = 6$.

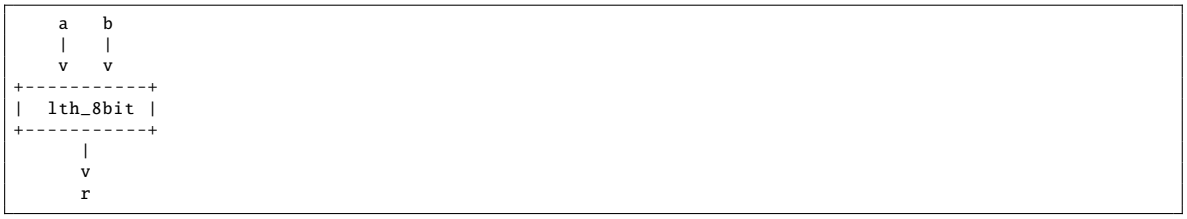
iii Using the left-shift method, we compute $D' = 2 \cdot D$ by simply relabelling the bits in D. That is, $D'_0 = 0$ and $D'_{i+1} = D_i$ for $0 \leq i < 3$. For example, given $D = 6_{(10)} = 110_{(2)}$ we have

$$\begin{aligned} D'_0 &= 0 \\ D'_1 = D_0 &= 0 \\ D'_2 = D_1 &= 1 \\ D'_3 = D_2 &= 1 \end{aligned}$$

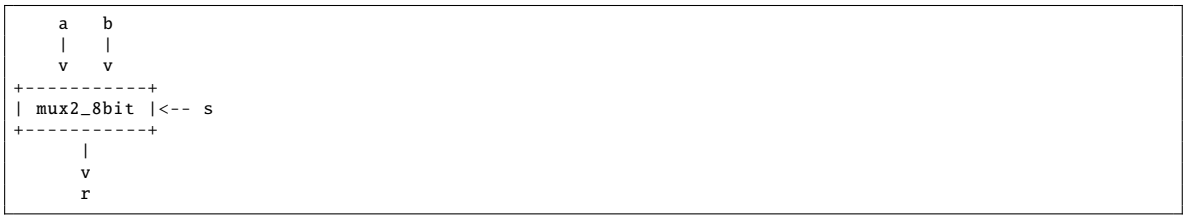
and hence $D' = 1100_{(2)} = 12_{(10)}$. Since there is no need for any logic gates to implement this method, the critical path is essentially nil: the only propagation delay relates to (small) wire delays. In comparison to the larger critical path of a suitable n-bit adder, this clearly means the left-shift approach is preferable.

S52. a A basic design would use two building blocks:

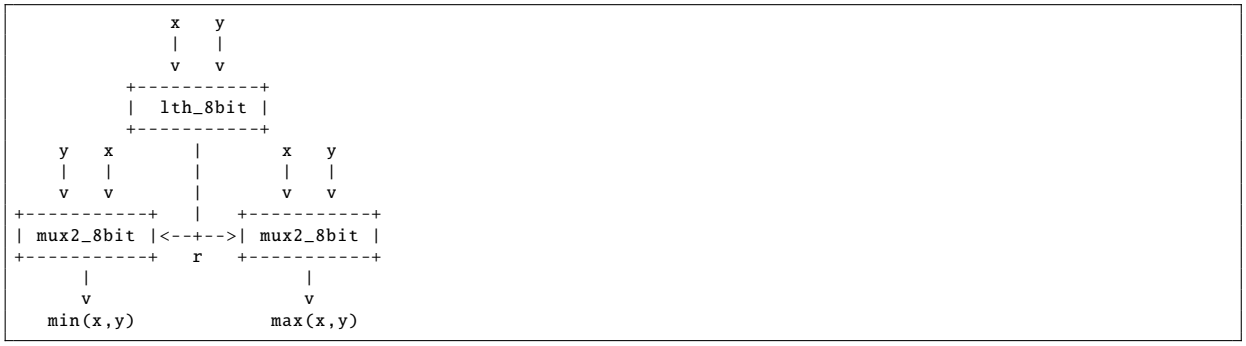
- `lth_8bit` compares two 8-bit inputs a and b and produces a 1-bit result r, where $r = 1$ if $a < b$ and $r = 0$ if $a \geq b$:



- `mux2_8bit` selects between two 8-bit inputs; if the inputs are a and b, the output $r = a$ if the control signal $s = 0$, or $r = b$ if $s = 1$:



Based on these building blocks, one can describe the component C as follows:



From a functional perspective, C compares x and y using an instance of the `lth_8bit` building block, and then uses the result r as a control signal for two instances of `mux2_8bit`. The left-hand instance selects y as the output if $r = 0$ and x if $r = 1$; that is, if $x < y$ then the output is $x = \min(x, y)$ otherwise the output is $y = \min(x, y)$. The right-hand instance swaps the inputs so it selects x as the output if $r = 0$ and y if $r = 1$; that is, if $x < y$ then the output is $y = \max(x, y)$ otherwise the output is $x = \max(x, y)$.

b The short answer (which gets about half the marks) is that the longest path through the mesh will go through $2n - 1$ of the C components: this is the path from the top-left corner down along one edge to the bottom-left and then along another edge to the bottom-right. So in a sense, if we write the propagation delay associated with each instance of C as T_C then the overall critical path is

$$(2n - 1) \cdot T_C.$$

In a bit more detail, the critical path through C is through one instance of `lth_8bit` and one instance of `mux2_8bit`. So we could write the overall critical path is

$$(2n - 1) \cdot (T_{\texttt{lth\_8bit}} + T_{\texttt{mux2\_8bit}}).$$

To be more detailed than this, we need to think about individual logic gates. Imagine we assume $T_{AND} = 50$ns, $T_{AND} = 20$ns, $T_{OR} = 20$ns and $T_{NOT} = 10$ns.

- `mux2_8bit` is simply eight `mux2_1bit` instances placed in parallel with each other; that is, the i-th such instance produces the i-th bit of the output based on the i-th bit of the inputs (but all using the same control signal). Assuming that the propagation delay of AND and OR gates dominates that of a NOT gate, the critical path through `mux2_1bit` will be $T_{AND} + T_{OR}$.
- `lth_8bit` is a combination of eight sub-components:

Each of these sub-components is placed in series so that t_{i-1} is an input from the previous sub-component and t_i is an output provided to the next.

Based on simple circuits derived from their truth tables, the critical paths for `lth_1bit` and `equ_1bit` are $T_{AND} + T_{NOT}$ and $T_{XOR} + T_{NOT}$ respectively. Thus the critical path of the whole sub-component is $T_{XOR} + T_{NOT} + T_{AND} + T_{OR}$ (since the critical path of `equ_1bit` is longer). Overall, the critical path of `lth_8bit` is

$$8 \cdot (T_{XOR} + T_{NOT} + T_{AND} + T_{OR}),$$

or more exactly

$$7 \cdot (T_{XOR} + T_{NOT} + T_{AND} + T_{OR}) + T_{AND} + T_{NOT}$$

because the 0-th sub-component is "special": there is no input from the previous sub-component.

Using this we can write the overall critical path for the mesh as

$$(2n - 1) \cdot (7 \cdot T_{XOR} + 8 \cdot T_{NOT} + 9 \cdot T_{AND} + 8 \cdot T_{OR})$$

or roughly $(2n - 1) \cdot 770$ns if we plug in the assumed delays.

c One problem is that the mesh does not always give the right result! If you were to build a 4×4 mesh and feed $5, 6, 7, 8$ into the top and $4, 3, 2, 1$ into the left-hand side, the bottom reads $5, 6, 7, 8$ and the right-hand side reads $4, 3, 2, 1$: numbers cannot move from bottom to top or from right to left, so there are some inputs a mesh cannot sort.

Beyond this trick question, the main idea is that the mesh is special-purpose, the processor is general-purpose: this implies a number of trade-offs in either direction that could be viewed as advantages or disadvantages in certain cases. For example, depending on n, one might argue that the processor will be require more logic to realise it (since it will include features extraneous to the task of sorting). Since it operates a fetch-decode-execute cycle to complete each instruction, there is an overhead (i.e., the fetch and decode at least) which means it potentially performs the task of sorting less quickly. On the other hand, once constructed the mesh is specialised to one task: it cannot be used to sort strings for example, and the size of input (i.e., n) is fixed. The processor makes the opposite trade-off; it should be clear that while it might be slower and potentially larger, it is vastly more flexible.

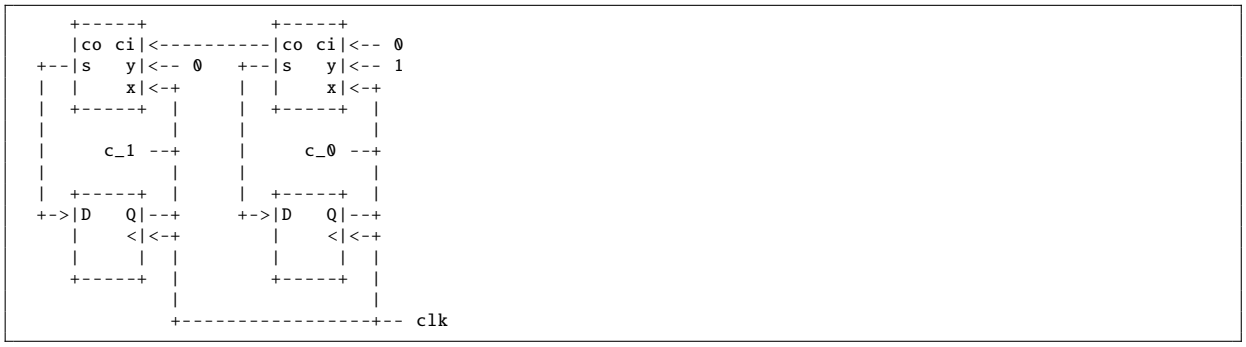
S53. a Imagine a component which is enabled (i.e., "turned on") using the input *en*:

- The idea of the component being level triggered is that the value of *en* is important, not a change in *en*: the component is enabled when *en* has a particular value, rather than at an edge when the value changes.
- The fact *en* is active high means that the component is enabled when $en = 1$ (rather than $en = 0$ which would make it active low). Though active high might seem the more logical choice, this is just part of the component specification: as long as everything is consistent, i.e., uses the right semantics to "turn on" the component, there is sometimes no major benefit of one approach over the other.

b Assume that M is a 4-state switch represented by a 2-bit value $M = \langle M_0, M_1 \rangle$: $\langle 0, 0 \rangle$ means off, $\langle 1, 0 \rangle$ means slow, $\langle 0, 1 \rangle$ means fast and $\langle 1, 1 \rangle$ means very fast. Also assume there is a clock signal called *clk* available, for example supplied by an oscillator of some form.

One approach would basically be to take *clk* and divide it to create two new clock signals c_0 and c_1 which have a longer period: each of the clock signals could then satisfy the criteria of toggling the fire button on and off at various speeds. A clock divider is fairly simple: the idea is to have a counter c clocked by *clk* and to sample the $(i-1)$-th bit of the counter: this behaves like *clk* divided by 2^i. For example the 0-th bit acts like *clk* but with twice the period.

A circuit to do this is fairly simple: we need some D-type flip-flops to hold the counter state, and some full-adders to increment the counter:



Given such a component which runs freely as long as it is driven by *clk*, we want to feed the original fire button F_0 through to form the new fire button input F'_0 when $M = 0$, and c_1, c_0 or *clk* through when $M = 1$, $M = 2$ or $M = 3$ (meaning a slow, fast or very fast toggling behaviour). We can describe this as the following truth table:

M_1	M_0	F'_0
0	0	F_0
0	1	c_1
1	0	c_0
1	1	*clk*

This is essentially a multiplexer controlled by M, and permits us to write

$$\begin{aligned} F'_0 = & (\neg M_0 \wedge \neg M_1 \wedge F_0) \\ & (M_0 \wedge \neg M_1 \wedge c_1) \\ & (\neg M_0 \wedge M_1 \wedge c_0) \\ & (M_0 \wedge M_1 \wedge clk) \end{aligned}$$

c i A synchronous protocol demands that the console and controller share a clock signal which acts to synchronise their activity, e.g., ensures each one sends and receives data at the right time. The problem with this is ensuring that the clock is not skewed for either component: since they are physically separate, this might be hard and hence this is not such a good option.

An asynchronous protocol relies on extra connections between the components, e.g., "request" and "acknowledge", that allow them to engage in a form of transaction: the extra connections essentially signal when data has been sent or received on the associated bus. This is more suitable given the scenario: the extra connections could potentially be shared with those that already exist (e.g., F_0, F_1, F_2 and D) thereby reducing the overhead, plus performance is not a big issue here (the protocol will presumably only be executed once when the components are turned on or plugged in).

Both approaches have an issue in that

- once the protocol is run someone could just plug in another, fake controller, or
- or simply intercept c and $T(c)$ pairs until it recovers the whole look-up table and then "imitate" it using a fake controller

so neither is particularly robust from a security point of view!

ii The temptation here is to say that the use of a 3-bit memory (or register) is the right way to go. Although this allows some degree of flexibility which is not required since the function is fixed, the main disadvantage is retention of the content when the controller or console is turned off: some form of non-volatile memory is therefore needed.

However, we can easily construct some dedicated logic to do the same thing. If we say that $y = T(x)$, the we can describe the behaviour of T using the following truth table:

x_2	x_1	x_0	y_2	y_1	y_0
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	0	1	1

This can be transformed into the following Karnaugh maps for y_0, y_1 and y_2

	x_1			
	x_0			
x_2	0	0	1	1
	0	0	1	1

	x_1			
	x_0			
x_2	1	1	0	1
	0	0	1	0

	x_1			
	x_0			
x_2	0	1	0	1
	1	0	0	1

which in turn can be transformed into the following equations

$$\begin{array}{rcl}
y_0 & = & (\quad x_1 \quad) \\
y_1 & = & (\quad \neg x_2 \wedge \neg x_1 \quad) \vee \\
 & & (\quad \neg x_2 \wedge \neg x_0 \quad) \vee \\
 & & (\quad x_2 \wedge x_1 \wedge x_0 \quad) \\
y_2 & = & (\quad x_1 \wedge \neg x_0 \quad) \vee \\
 & & (\quad x_2 \wedge \neg x_0 \quad) \vee \\
 & & (\quad \neg x_2 \wedge \neg x_1 \wedge x_0 \quad)
\end{array}$$

which are enough to implement the look-up table: we pass x as input, and it produces the right y (for this fixed T) as output.

S54. This is a classic "puzzle" question in digital logic. There are a few ways to describe the strategy, but the one used here is based on counting the number of inputs which are 1. In short, we start by computing

$$\begin{array}{rcl}
t_1 & = & \neg(x \wedge y \vee y \wedge z \vee x \wedge z) \\
t_2 & = & \neg((x \wedge y \wedge z) \vee t_1 \wedge (x \vee y \vee z))
\end{array}$$

which use our quota of NOT gates. The idea is that $t_1 = 1$ iff. one or zero of x, y and z are 1, and in the same way $t_2 = 1$ iff. two or zero of x, y and z are 1. This can be hard to see, so consider a truth table

x	y	z	$x \wedge y$	$y \wedge z$	$x \wedge z$	$x \wedge y \wedge z$	$x \vee y \vee z$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1
0	1	0	0	0	0	0	1
0	1	1	0	1	0	0	1
1	0	0	0	0	0	0	1
1	0	1	0	0	1	0	1
1	1	0	1	0	0	0	1
1	1	1	1	1	1	1	1

meaning

x	y	z	$x \wedge y \vee y \wedge z \vee x \wedge z$	t_1	$(x \wedge y \wedge z) \vee t_1 \wedge (x \vee y \vee z)$	t_2
0	0	0	0	1	0	1
0	0	1	0	1	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	1	1	0
1	0	1	1	0	0	1
1	1	0	1	0	0	1
1	1	1	1	0	1	0

and hence t_1 and t_2 are as required. Now, we can generate the three results as

$$\begin{aligned}
\neg x &= (t_1 \wedge t_2) \vee (t_1 \wedge (y \vee z)) \vee (t_2 \wedge y \wedge z) \\
&= (t_1 \wedge t_2) \vee (t_1 \wedge y) \vee (t_1 \wedge z) \vee (t_2 \wedge y \wedge z) \\
\neg y &= (t_1 \wedge t_2) \vee (t_1 \wedge (x \vee z)) \vee (t_2 \wedge x \wedge z) \\
&= (t_1 \wedge t_2) \vee (t_1 \wedge x) \vee (t_1 \wedge z) \vee (t_2 \wedge x \wedge z) \\
\neg z &= (t_1 \wedge t_2) \vee (t_1 \wedge (x \vee y)) \vee (t_2 \wedge x \wedge y) \\
&= (t_1 \wedge t_2) \vee (t_1 \wedge x) \vee (t_1 \wedge y) \vee (t_2 \wedge x \wedge y)
\end{aligned}$$

S55. Imagine that for some n-bit input x, we let $y_i = C_i(x)$ denote the evaluation of C_i to get an output y_i. As such, the equivalence of C_1 and C_2 can be stated as a test whether $y_1 = y_2$ for all values of x; another way to say the same thing is to test whether an x exists such that $y_1 \neq y_2$ which will distinguish the circuits, i.e., imply they are not equivalent.

Using the second formulation, we can write the test as $y_1 \oplus y_2$ since the XOR will produce 1 when y_1 differs from y_2 and 0 otherwise. As such, we have n Boolean variables (the bits of x) and want an assignment that implies the expression $C_1(x) \oplus C_2(x)$ will evaluate to 1. This is the same as described in the description of SAT, so if we can solve the SAT instance we prove the circuits are (not) equivalent.

S56. a The latency of the circuit is the time taken to perform the computation, i.e., to compute some r given x. For this circuit, the latency is simply the sum of the critical paths.

b The throughput is the number of operations performed per unit time period. This is essentially the number of operations we can start (resp. that finish) within that time period.

By pipelining the circuit, using say 3 stages, one might expect the latency to increase slightly (by virtue of having to add pipeline registers between each stage) but the throughput to increase (by virtue of decreasing the overall critical path to the longest stage, and hence increasing the maximum clock frequency). The trade-off is strongly influenced by the number of and balance between stages, meaning careful analysis of the circuit before applying the optimisation is important.

S57. a The latency of a circuit is the time elapsed between when a given operation starts and when it finishes. The throughput of a circuit is the number of operations that can be started in each time period; that is, how long it takes between when two subsequent operations can be started.

b The latency of the circuit is the sum of all the latencies of the parts,i.e.,

$$40\text{ns} + 10\text{ns} + 30\text{ns} + 10\text{ns} + 50\text{ns} + 10\text{ns} + 10\text{ns} = 160\text{ns}.$$

The throughput relates to the length of the longest pipeline stage; the circuit is not pipelined, so more specifically we can say it is $\frac{1}{160 \cdot 10^{-9}}$.

c The new latency is still the sum of all the parts, but now includes the extra pipeline register:

$$40\text{ns} + 10\text{ns} + 30\text{ns} + 10\text{ns} + 10\text{ns} + 50\text{ns} + 10\text{ns} + 10\text{ns} = 170\text{ns}.$$

However, the throughput is now more because the longest pipeline stage only has a latency of 100ns (including the extra register). Specifically, the throughput increases to $\frac{1}{100 \cdot 10^{-9}}$ which essentially means we can start new operations more often than before.

d To maximise the throughput we need to minimise the latency of the longest pipeline stage (i.e., the one whose individual latency is the largest) since this will act as a limit. The latency of part E is largest (at 50ns) and hence represents said limit: the longest pipeline stage cannot have a latency of less than 60ns (i.e., the latency of part E plus the latency of a pipeline register).

We can achieve this by creating a 4-stage pipeline: adding two more pipeline registers, between parts B and C and parts E and F, ensures the stages have latencies of

$$\begin{array}{lcll}
A + B + REG & \leadsto & 40\text{ns} + 10\text{ns} + 10\text{ns} & = 60\text{ns} \\
C + D + REG & \leadsto & 30\text{ns} + 10\text{ns} + 10\text{ns} & = 50\text{ns} \\
E + REG & \leadsto & 50\text{ns} + 10\text{ns} & = 60\text{ns} \\
F + REG & \leadsto & 10\text{ns} + 10\text{ns} & = 20\text{ns}
\end{array}$$

Overall, the latency is increased to 190ns but the throughput is $\frac{1}{60 \cdot 10^{-9}}$.

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

S58. a Reference implementation:

$$\begin{array}{rcccccc} r & = & (& \neg y & \wedge & \neg z &) \vee \\ & & (& y & \wedge & \neg z &) \vee \\ & & (& y & \wedge & z &) \end{array}$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$\begin{array}{rcccccc} r & = & (& & & \neg z &) \vee \\ & & (& y & & &) \end{array}$$

S59. a Reference implementation:

$$\begin{array}{rcccccc} r & = & (& \neg y & \wedge & \neg z &) \vee \\ & & (& \neg y & \wedge & z &) \vee \\ & & (& y & \wedge & z &) \end{array}$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$\begin{array}{rcccccc} r & = & (& \neg y & & &) \vee \\ & & (& & & z &) \end{array}$$

S60. a Reference implementation:

$$r = (\neg y \wedge \neg z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\neg y \wedge \neg z)$$

S61. a Reference implementation:

$$\begin{array}{rcccccc} r & = & (& \neg y & \wedge & z &) \vee \\ & & (& y & \wedge & \neg z &) \end{array}$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$\begin{array}{rcccccc} r & = & (& y & \wedge & \neg z &) \vee \\ & & (& \neg y & \wedge & z &) \end{array}$$

S62. a Reference implementation:

$$r = (\ \neg y \ \wedge \ \neg z\ ) \ \vee \ (\ y \ \wedge \ \neg z\ )$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\ \neg z\ )$$

S63. a Reference implementation:

$$r = (\ y \ \wedge \ z\ )$$

b Annotated Karnaugh map

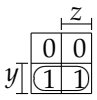
and associated, optimised implementation:

$$r = (\ y \ \wedge \ z\ )$$

S64. a Reference implementation:

$$r = (\ y \ \wedge \ \neg z\ ) \ \vee \ (\ y \ \wedge \ z\ )$$

b Annotated Karnaugh map



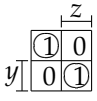
and associated, optimised implementation:

$$r = (\ y\ )$$

S65. a Reference implementation:

$$r = (\ \neg y \ \wedge \ \neg z\ ) \ \vee \ (\ y \ \wedge \ z\ )$$

b Annotated Karnaugh map



and associated, optimised implementation:

$$r = (\ y \ \wedge \ z\ ) \ \vee \ (\ \neg y \ \wedge \ \neg z\ )$$

S66. a Reference implementation:

$$r = (\ \neg y \ \wedge \ z\ )$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\neg y \wedge z)$$

S67. a Reference implementation:

$$r = (y \wedge \neg z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (y \wedge \neg z)$$

S68. a Reference implementation:

$$\begin{aligned} r = {} & (\neg y \wedge z) \vee \\ & (y \wedge z) \end{aligned}$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (z)$$

S69. a Reference implementation:

$$\begin{aligned} r = {} & (\neg y \wedge \neg z) \vee \\ & (\neg y \wedge z) \vee \\ & (y \wedge \neg z) \end{aligned}$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$\begin{aligned} r = {} & (\neg y) \vee \\ & (\neg z) \end{aligned}$$

S70. a Reference implementation:

$$\begin{aligned} r = {} & (\neg y \wedge z) \vee \\ & (y \wedge \neg z) \vee \\ & (y \wedge z) \end{aligned}$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$\begin{array}{lcl} r & = & (\quad\quad z\quad) \vee \\ & & (\ y\quad\quad\ ) \end{array}$$

S71. a Reference implementation:

$$\begin{array}{lcl} r & = & (\ \neg y \wedge \neg z\ ) \vee \\ & & (\ \neg y \wedge z\ ) \end{array}$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\ \neg y\quad\quad)$$

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

S72. a Reference implementation:

$$r = (\ y \wedge z\ )$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\quad\quad z\quad)$$

S73. a Reference implementation:

$$\begin{array}{lcl} r & = & (\ \neg y \wedge z\ ) \vee \\ & & (\ y \wedge \neg z\ ) \end{array}$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$\begin{array}{lcl} r & = & (\quad\quad z\quad) \vee \\ & & (\ y\quad\quad\ ) \end{array}$$

S74. a Reference implementation:

$$r = (\ \neg y \wedge \neg z\ )$$

b Annotated Karnaugh map

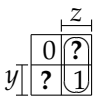
and associated, optimised implementation:

$$r = (\ \neg y\quad\quad)$$

S75. a Reference implementation:

$$r = (y \wedge z)$$

b Annotated Karnaugh map



and associated, optimised implementation:

$$r = (z)$$

S76. a Reference implementation:

$$r = (\neg y \wedge z) \vee (y \wedge z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (z)$$

S77. a Reference implementation:

$$r = (\neg y \wedge \neg z) \vee (y \wedge \neg z)$$

b Annotated Karnaugh map

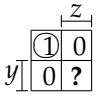
and associated, optimised implementation:

$$r = (\neg z)$$

S78. a Reference implementation:

$$r = (\neg y \wedge \neg z)$$

b Annotated Karnaugh map



and associated, optimised implementation:

$$r = (\neg y \wedge \neg z)$$

S79. a Reference implementation:

$$r = (\neg y \wedge z) \vee (y \wedge z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\quad z \quad)$$

S80. a Reference implementation:

$$r = (\; y \; \wedge \; \neg z \;)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\; y \quad)$$

S81. a Reference implementation:

$$r = (\; y \; \wedge \; \neg z \;)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\; y \; \wedge \; \neg z \;)$$

S82. a Reference implementation:

$$\begin{array}{rcl} r & = & (\; \neg y \; \wedge \; z \;) \; \vee \\ & & (\; y \; \wedge \; \neg z \;) \; \vee \\ & & (\; y \; \wedge \; z \;) \end{array}$$

b Annotated Karnaugh map

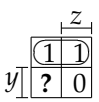
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\quad z \quad) \; \vee \\ & & (\; y \quad) \end{array}$$

S83. a Reference implementation:

$$\begin{array}{rcl} r & = & (\; \neg y \; \wedge \; \neg z \;) \; \vee \\ & & (\; \neg y \; \wedge \; z \;) \end{array}$$

b Annotated Karnaugh map



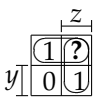
and associated, optimised implementation:

$$r = (\neg y)$$

S84. a Reference implementation:

$$r = (\neg y \wedge \neg z) \vee (y \wedge z)$$

b Annotated Karnaugh map



and associated, optimised implementation:

$$r = (\neg y) \vee (z)$$

S85. a Reference implementation:

$$r = (\neg y \wedge \neg z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\neg y)$$

S86. a Reference implementation:

$$r = (y \wedge z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (y \wedge z)$$

S87. a Reference implementation:

$$r = (y \wedge z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (y)$$

S88. a Reference implementation:

$$r = (\neg y \wedge z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\quad z \quad)$$

S89. a Reference implementation:

$$r = (\neg y \wedge z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\neg y \wedge z)$$

S90. a Reference implementation:

$$r = (y \wedge z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\quad z \quad)$$

S91. a Reference implementation:

$$r = (\neg y \wedge \neg z) \vee (y \wedge \neg z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\quad \neg z \quad)$$

S92. a Reference implementation:

$$r = (\neg y \wedge z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\neg y \wedge z)$$

S93. a Reference implementation:

$$\begin{array}{rcccccc} r & = & (& \neg y & \wedge & z &) \quad \vee \\ & & (& y & \wedge & \neg z &) \end{array}$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$\begin{array}{rcccccc} r & = & (& y & \wedge & \neg z &) \quad \vee \\ & & (& \neg y & \wedge & z &) \end{array}$$

S94. a Reference implementation:

$$r = (\; y \; \wedge \; \neg z \;)$$

b Annotated Karnaugh map

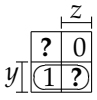
and associated, optimised implementation:

$$r = (\quad \neg z \quad)$$

S95. a Reference implementation:

$$r = (\; y \; \wedge \; \neg z \;)$$

b Annotated Karnaugh map



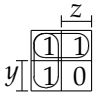
and associated, optimised implementation:

$$r = (\; y \quad)$$

S96. a Reference implementation:

$$\begin{array}{rcccccc} r & = & (& \neg y & \wedge & \neg z &) \quad \vee \\ & & (& \neg y & \wedge & z &) \quad \vee \\ & & (& y & \wedge & \neg z &) \end{array}$$

b Annotated Karnaugh map



and associated, optimised implementation:

$$\begin{array}{rcccccc} r & = & (& \neg y & & &) \quad \vee \\ & & (& & & \neg z &) \end{array}$$

S97. a Reference implementation:

$$\begin{array}{rcccccc} r & = & (& \neg y & \wedge & \neg z &) \quad \vee \\ & & (& \neg y & \wedge & z &) \end{array}$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (\neg y)$$

S98. a Reference implementation:

$$r = (\neg y \wedge \neg z) \vee (y \wedge z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (y \wedge z) \vee (\neg y \wedge \neg z)$$

S99. a Reference implementation:

$$r = (y \wedge \neg z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

$$r = (y \wedge \neg z)$$

S100. a Reference implementation:

$$r = (y \wedge \neg z)$$

b Annotated Karnaugh map

and associated, optimised implementation:

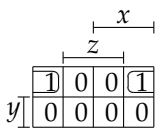
$$r = (y)$$

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

S101. a Reference implementation:

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

b Annotated Karnaugh map



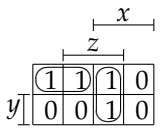
and associated, optimised implementation:

$$r = (\quad \neg y \wedge \neg z \quad)$$

S102. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg x \wedge \neg y \wedge z) \vee \\ & & (x \wedge \neg y \wedge z) \vee \\ & & (x \wedge y \wedge z) \end{array}$$

b Annotated Karnaugh map



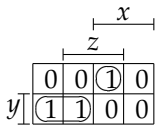
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\neg x \wedge \neg y) \vee \\ & & (x \wedge z) \end{array}$$

S103. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg x \wedge y \wedge \neg z) \vee \\ & & (\neg x \wedge y \wedge z) \vee \\ & & (x \wedge \neg y \wedge z) \end{array}$$

b Annotated Karnaugh map



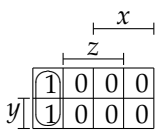
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\neg x \wedge y) \vee \\ & & (x \wedge \neg y \wedge z) \end{array}$$

S104. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg x \wedge y \wedge \neg z) \end{array}$$

b Annotated Karnaugh map



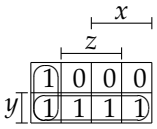
and associated, optimised implementation:

$$r = (\neg x \quad \wedge \neg z)$$

S105. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



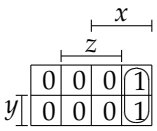
and associated, optimised implementation:

$$\begin{array}{rcccccc} r & = & (& & y & &) & \vee \\ & & (& \neg x & \wedge & \neg z &) & \end{array}$$

S106. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & y & \wedge & \neg z &) & \end{array}$$

b Annotated Karnaugh map



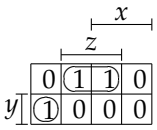
and associated, optimised implementation:

$$r = (x \wedge \neg z)$$

S107. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



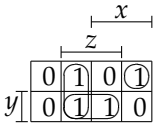
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& & & \neg y & \wedge & z &) & \end{array}$$

S108. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



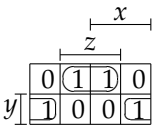
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \qquad \wedge \quad z\ ) \vee \\ & & (\ \qquad y \wedge \quad z\ ) \vee \\ & & (\ \ x \wedge \neg y \wedge \neg z\ ) \end{array}$$

S109. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \wedge \neg y \wedge \quad z\ ) \vee \\ & & (\ \neg x \wedge \quad y \wedge \neg z\ ) \vee \\ & & (\ \ x \wedge \neg y \wedge \quad z\ ) \vee \\ & & (\ \ x \wedge \quad y \wedge \neg z\ ) \end{array}$$

b Annotated Karnaugh map



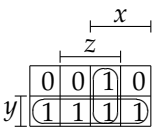
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \qquad y \wedge \neg z\ ) \vee \\ & & (\ \qquad \neg y \wedge \quad z\ ) \end{array}$$

S110. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \wedge \quad y \wedge \neg z\ ) \vee \\ & & (\ \neg x \wedge \quad y \wedge \quad z\ ) \vee \\ & & (\ \ x \wedge \neg y \wedge \quad z\ ) \vee \\ & & (\ \ x \wedge \quad y \wedge \neg z\ ) \vee \\ & & (\ \ x \wedge \quad y \wedge \quad z\ ) \end{array}$$

b Annotated Karnaugh map



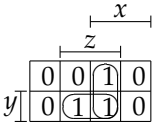
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \qquad y \qquad\ ) \vee \\ & & (\ x \qquad \wedge \ z\ ) \end{array}$$

S111. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \wedge \quad y \wedge z\ ) \vee \\ & & (\ \ x \wedge \neg y \wedge z\ ) \vee \\ & & (\ \ x \wedge \quad y \wedge z\ ) \end{array}$$

b Annotated Karnaugh map



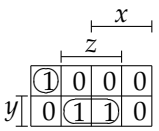
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \qquad y \wedge z\ ) \vee \\ & & (\ x \qquad \wedge \ z\ ) \end{array}$$

S112. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



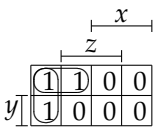
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& & & y & \wedge & z &) & \end{array}$$

S113. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \end{array}$$

b Annotated Karnaugh map



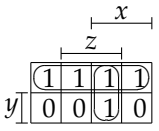
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & & &) & \vee \\ & & (& \neg x & & & \wedge & \neg z &) & \end{array}$$

S114. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



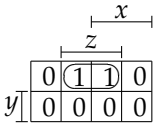
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& & \neg y & & & &) & \vee \\ & & (& x & & & \wedge & z &) & \end{array}$$

S115. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



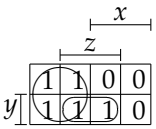
and associated, optimised implementation:

$$r = (\quad \neg y \wedge z \quad)$$

S116. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg x \wedge \neg y \wedge z) \vee \\ & & (\neg x \wedge y \wedge \neg z) \vee \\ & & (\neg x \wedge y \wedge z) \vee \\ & & (x \wedge y \wedge z) \end{array}$$

b Annotated Karnaugh map



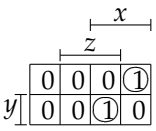
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\neg x \quad) \vee \\ & & (\quad y \wedge z) \end{array}$$

S117. a Reference implementation:

$$\begin{array}{rcl} r & = & (x \wedge \neg y \wedge \neg z) \vee \\ & & (x \wedge y \wedge z) \end{array}$$

b Annotated Karnaugh map



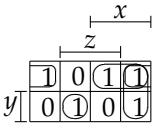
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (x \wedge y \wedge z) \vee \\ & & (x \wedge \neg y \wedge \neg z) \end{array}$$

S118. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg x \wedge y \wedge z) \vee \\ & & (x \wedge \neg y \wedge \neg z) \vee \\ & & (x \wedge \neg y \wedge z) \vee \\ & & (x \wedge y \wedge \neg z) \end{array}$$

b Annotated Karnaugh map



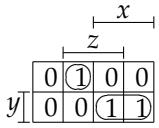
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\neg x \wedge y \wedge z) \vee \\ & & (x \quad \wedge \neg z) \vee \\ & & (x \wedge \neg y \quad) \vee \\ & & (\quad \neg y \wedge \neg z) \end{array}$$

S119. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



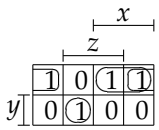
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & & &) & \end{array}$$

S120. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



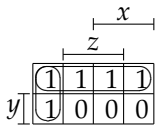
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & & &) & \vee \\ & & (& & & \neg y & \wedge & \neg z &) & \end{array}$$

S121. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



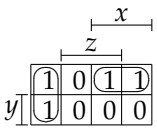
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& & & \neg y & & &) & \vee \\ & & (& \neg x & & & \wedge & \neg z &) & \end{array}$$

S122. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



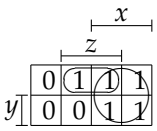
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ x \wedge \neg y\ ) \vee \\ & & (\ \neg x \wedge \neg z\ ) \end{array}$$

S123. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \wedge \neg y \wedge z\ ) \vee \\ & & (\ x \wedge \neg y \wedge \neg z\ ) \vee \\ & & (\ x \wedge \neg y \wedge z\ ) \vee \\ & & (\ x \wedge y \wedge \neg z\ ) \vee \\ & & (\ x \wedge y \wedge z\ ) \end{array}$$

b Annotated Karnaugh map



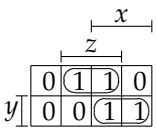
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ x\ ) \vee \\ & & (\ \neg y \wedge z\ ) \end{array}$$

S124. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \wedge \neg y \wedge z\ ) \vee \\ & & (\ x \wedge \neg y \wedge z\ ) \vee \\ & & (\ x \wedge y \wedge \neg z\ ) \vee \\ & & (\ x \wedge y \wedge z\ ) \end{array}$$

b Annotated Karnaugh map



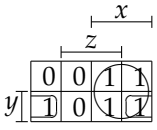
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ x \wedge y\ ) \vee \\ & & (\ \neg y \wedge z\ ) \end{array}$$

S125. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \wedge y \wedge \neg z\ ) \vee \\ & & (\ x \wedge \neg y \wedge \neg z\ ) \vee \\ & & (\ x \wedge \neg y \wedge z\ ) \vee \\ & & (\ x \wedge y \wedge \neg z\ ) \vee \\ & & (\ x \wedge y \wedge z\ ) \end{array}$$

b Annotated Karnaugh map



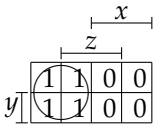
and associated, optimised implementation:

$$\begin{array}{lcl} r & = & (\quad y \wedge \neg z \quad) \vee \\ & & (\; x \qquad\quad) \end{array}$$

S126. a Reference implementation:

$$\begin{array}{lcl} r & = & (\neg x \wedge \neg y \wedge \neg z \;) \vee \\ & & (\neg x \wedge \neg y \wedge z \;) \vee \\ & & (\neg x \wedge y \wedge \neg z \;) \vee \\ & & (\neg x \wedge y \wedge z \;) \end{array}$$

b Annotated Karnaugh map



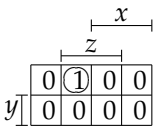
and associated, optimised implementation:

$$r = (\neg x \qquad)$$

S127. a Reference implementation:

$$r = (\neg x \wedge \neg y \wedge z \;)$$

b Annotated Karnaugh map



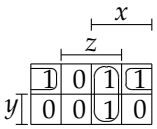
and associated, optimised implementation:

$$r = (\neg x \wedge \neg y \wedge z \;)$$

S128. a Reference implementation:

$$\begin{array}{lcl} r & = & (\neg x \wedge \neg y \wedge \neg z \;) \vee \\ & & (\; x \wedge \neg y \wedge \neg z \;) \vee \\ & & (\; x \wedge \neg y \wedge z \;) \vee \\ & & (\; x \wedge y \wedge z \;) \end{array}$$

b Annotated Karnaugh map



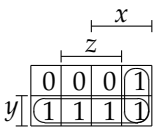
and associated, optimised implementation:

$$\begin{array}{lcl} r & = & (\; x \qquad \wedge \; z \;) \vee \\ & & (\qquad \neg y \wedge \neg z \;) \end{array}$$

S129. a Reference implementation:

$$\begin{array}{lcl} r & = & (\neg x \wedge y \wedge \neg z \;) \vee \\ & & (\neg x \wedge y \wedge z \;) \vee \\ & & (\; x \wedge \neg y \wedge \neg z \;) \vee \\ & & (\; x \wedge y \wedge \neg z \;) \vee \\ & & (\; x \wedge y \wedge z \;) \end{array}$$

b Annotated Karnaugh map



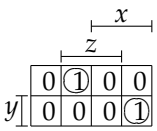
and associated, optimised implementation:

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

S130. a Reference implementation:

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

b Annotated Karnaugh map



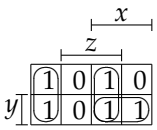
and associated, optimised implementation:

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

S131. a Reference implementation:

$$\begin{aligned} r = &(\ \neg x \wedge \neg y \wedge \neg z\ ) \vee \\ &(\ \neg x \wedge y \wedge \neg z\ ) \vee \\ &(\ x \wedge \neg y \wedge z\ ) \vee \\ &(\ x \wedge y \wedge \neg z\ ) \vee \\ &(\ x \wedge y \wedge z\ ) \end{aligned}$$

b Annotated Karnaugh map



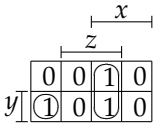
and associated, optimised implementation:

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

S132. a Reference implementation:

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

b Annotated Karnaugh map



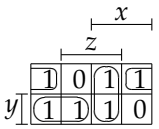
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & & & \wedge & z &) & \end{array}$$

S133. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



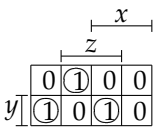
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& & & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & & & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & & &) & \end{array}$$

S134. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



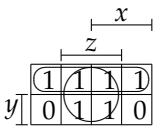
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

S135. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



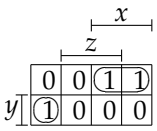
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& & & & & z &) & \vee \\ & & (& & \neg y & & & &) & \end{array}$$

S136. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



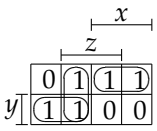
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & & &) & \end{array}$$

S137. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



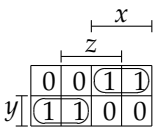
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & & & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & & &) & \vee \\ & & (& \neg x & \wedge & y & & &) & \end{array}$$

S138. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



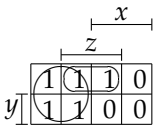
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& x & \wedge & \neg y & & &) & \vee \\ & & (& \neg x & \wedge & y & & &) & \end{array}$$

S139. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



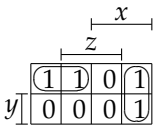
and associated, optimised implementation:

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

S140. a Reference implementation:

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

b Annotated Karnaugh map



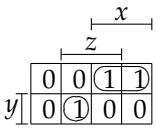
and associated, optimised implementation:

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

S141. a Reference implementation:

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

b Annotated Karnaugh map



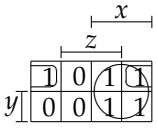
and associated, optimised implementation:

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

S142. a Reference implementation:

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

b Annotated Karnaugh map



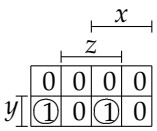
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ x \qquad\qquad\quad\ \ )\ \vee \\ & & (\ \quad \neg y \ \wedge \ \neg z\ )\end{array}$$

S143. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ y \ \wedge \ \neg z\ )\ \vee \\ & & (\ \ x \ \wedge \ y \ \wedge \ \ z\ )\end{array}$$

b Annotated Karnaugh map



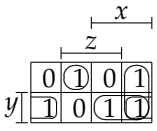
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ y \ \wedge \ \neg z\ )\ \vee \\ & & (\ \ x \ \wedge \ y \ \wedge \ \ z\ )\end{array}$$

S144. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ \neg y \ \wedge \ \ z\ )\ \vee \\ & & (\ \neg x \ \wedge \ \ y \ \wedge \ \neg z\ )\ \vee \\ & & (\ \ x \ \wedge \ \neg y \ \wedge \ \neg z\ )\ \vee \\ & & (\ \ x \ \wedge \ \ y \ \wedge \ \neg z\ )\ \vee \\ & & (\ \ x \ \wedge \ \ y \ \wedge \ \ z\ )\end{array}$$

b Annotated Karnaugh map



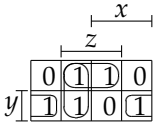
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \ x \qquad\ \ \wedge \ \neg z\ )\ \vee \\ & & (\ \neg x \ \wedge \ \neg y \ \wedge \ \ z\ )\ \vee \\ & & (\ \ x \ \wedge \ \ y \qquad\ \ )\ \vee \\ & & (\ \qquad\ \ y \ \wedge \ \neg z\ )\end{array}$$

S145. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ \neg y \ \wedge \ \ z\ )\ \vee \\ & & (\ \neg x \ \wedge \ \ y \ \wedge \ \neg z\ )\ \vee \\ & & (\ \neg x \ \wedge \ \ y \ \wedge \ \ z\ )\ \vee \\ & & (\ \ x \ \wedge \ \neg y \ \wedge \ \ z\ )\ \vee \\ & & (\ \ x \ \wedge \ \ y \ \wedge \ \neg z\ )\end{array}$$

b Annotated Karnaugh map



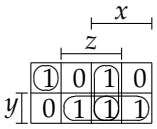
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \qquad\ \ \wedge \ \ z\ )\ \vee \\ & & (\ \qquad\ \ y \ \wedge \ \neg z\ )\ \vee \\ & & (\ \qquad \neg y \ \wedge \ \ z\ )\end{array}$$

S146. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



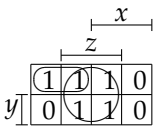
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& & & y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & & &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & & & \wedge & z &) & \end{array}$$

S147. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



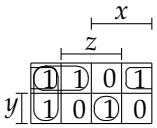
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & & &) & \vee \\ & & (& & & & & z &) & \end{array}$$

S148. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



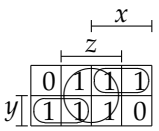
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & & &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \vee \\ & & (& \neg x & & & \wedge & \neg z &) & \vee \\ & & (& & & \neg y & \wedge & \neg z &) & \end{array}$$

S149. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



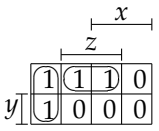
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& & & & & z &) & \vee \\ & & (& x & \wedge & \neg y & & &) & \vee \\ & & (& \neg x & \wedge & y & & &) & \end{array}$$

S150. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



and associated, optimised implementation:

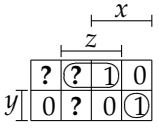
$$\begin{array}{rcccccccc} r & = & (& \neg x & & & \wedge & \neg z &) & \vee \\ & & (& & & \neg y & \wedge & z &) & \end{array}$$

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

S151. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & \neg z &) & \end{array}$$

b Annotated Karnaugh map



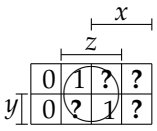
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& & & \neg y & \wedge & z &) & \end{array}$$

S152. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



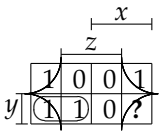
and associated, optimised implementation:

$$r = (\quad z \quad)$$

S153. a Reference implementation:

$$\begin{aligned} r = & (\neg x \wedge \neg y \wedge \neg z) \vee \\ & (\neg x \wedge y \wedge \neg z) \vee \\ & (\neg x \wedge y \wedge z) \vee \\ & (x \wedge \neg y \wedge \neg z) \end{aligned}$$

b Annotated Karnaugh map



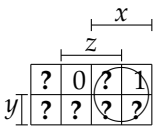
and associated, optimised implementation:

$$\begin{aligned} r = & (\quad \neg z \quad) \vee \\ & (\neg x \wedge y \quad) \end{aligned}$$

S154. a Reference implementation:

$$r = (x \wedge \neg y \wedge \neg z)$$

b Annotated Karnaugh map



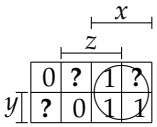
and associated, optimised implementation:

$$r = (x \quad)$$

S155. a Reference implementation:

$$\begin{aligned} r = & (x \wedge \neg y \wedge z) \vee \\ & (x \wedge y \wedge \neg z) \vee \\ & (x \wedge y \wedge z) \end{aligned}$$

b Annotated Karnaugh map



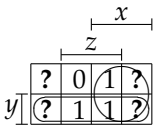
and associated, optimised implementation:

$$r = (x \quad)$$

S156. a Reference implementation:

$$\begin{aligned} r = & (\neg x \wedge y \wedge z) \vee \\ & (x \wedge \neg y \wedge z) \vee \\ & (x \wedge y \wedge z) \end{aligned}$$

b Annotated Karnaugh map



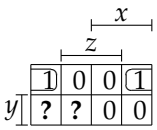
and associated, optimised implementation:

$$r = (\ y\ ) \vee (\ x\ )$$

S157. a Reference implementation:

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

b Annotated Karnaugh map



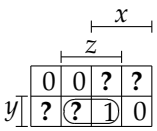
and associated, optimised implementation:

$$r = (\ \neg y \wedge \neg z\ )$$

S158. a Reference implementation:

$$r = (\ x \wedge y \wedge z\ )$$

b Annotated Karnaugh map



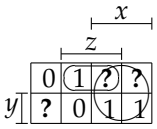
and associated, optimised implementation:

$$r = (\ y \wedge z\ )$$

S159. a Reference implementation:

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

b Annotated Karnaugh map



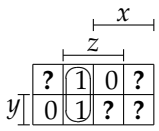
and associated, optimised implementation:

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

S160. a Reference implementation:

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

b Annotated Karnaugh map



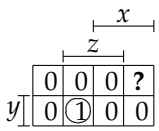
and associated, optimised implementation:

$$r = (\neg x \quad \wedge \quad z)$$

S161. a Reference implementation:

$$r = (\neg x \wedge y \wedge z)$$

b Annotated Karnaugh map



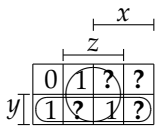
and associated, optimised implementation:

$$r = (\neg x \wedge y \wedge z)$$

S162. a Reference implementation:

$$\begin{aligned} r = & (\neg x \wedge \neg y \wedge z) \vee \\ & (\neg x \wedge y \wedge \neg z) \vee \\ & (x \wedge y \wedge z) \end{aligned}$$

b Annotated Karnaugh map



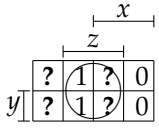
and associated, optimised implementation:

$$\begin{aligned} r = & (y) \vee \\ & (z) \end{aligned}$$

S163. a Reference implementation:

$$\begin{aligned} r = & (\neg x \wedge \neg y \wedge z) \vee \\ & (\neg x \wedge y \wedge z) \end{aligned}$$

b Annotated Karnaugh map



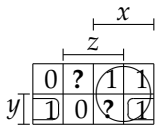
and associated, optimised implementation:

$$r = (z)$$

S164. a Reference implementation:

$$\begin{aligned} r = & (\neg x \wedge y \wedge \neg z) \vee \\ & (x \wedge \neg y \wedge \neg z) \vee \\ & (x \wedge \neg y \wedge z) \vee \\ & (x \wedge y \wedge \neg z) \end{aligned}$$

b Annotated Karnaugh map



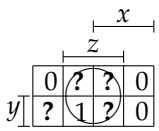
and associated, optimised implementation:

$$r = (\ x \qquad\qquad\ )\ \vee$$
$$(\ \ \ y \ \wedge\ \neg z\ )$$

S165. a Reference implementation:

$$r = (\ \neg x \ \wedge\ y \ \wedge\ z\ )$$

b Annotated Karnaugh map



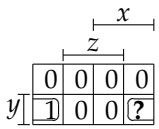
and associated, optimised implementation:

$$r = (\ \ \ z\ )$$

S166. a Reference implementation:

$$r = (\ \neg x \ \wedge\ y \ \wedge\ \neg z\ )$$

b Annotated Karnaugh map



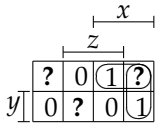
and associated, optimised implementation:

$$r = (\ \ \ y \ \wedge\ \neg z\ )$$

S167. a Reference implementation:

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

b Annotated Karnaugh map



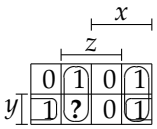
and associated, optimised implementation:

$$r = (\ x \qquad\ \wedge\ \neg z\ )\ \vee$$
$$(\ x \ \wedge\ \neg y \qquad\ )$$

S168. a Reference implementation:

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

b Annotated Karnaugh map



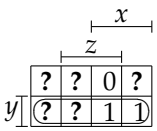
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & & \wedge & z &) & \vee \\ & & (& x & & \wedge & \neg z &) & \vee \\ & & (& & y & \wedge & \neg z &) & \end{array}$$

S169. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



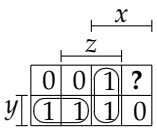
and associated, optimised implementation:

$$r = (\quad y \quad)$$

S170. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



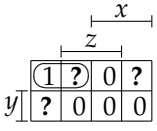
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& x & & & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & & &) & \end{array}$$

S171. a Reference implementation:

$$r = (\quad \neg x \quad \wedge \quad \neg y \quad \wedge \quad \neg z \quad)$$

b Annotated Karnaugh map



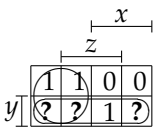
and associated, optimised implementation:

$$r = (\quad \neg x \quad \wedge \quad \neg y \quad)$$

S172. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



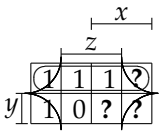
and associated, optimised implementation:

$$\begin{array}{rcccccc} r & = & (& \neg x & &) & \vee \\ & & (& & y &) & \end{array}$$

S173. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



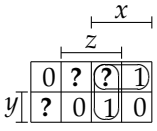
and associated, optimised implementation:

$$\begin{array}{rcccccc} r & = & (& & & \neg z &) & \vee \\ & & (& & \neg y & &) & \end{array}$$

S174. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



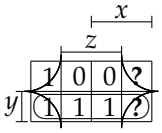
and associated, optimised implementation:

$$\begin{array}{rcccccccc} r & = & (& x & \wedge & \neg y & & &) & \vee \\ & & (& x & & & \wedge & z &) & \end{array}$$

S175. a Reference implementation:

$$\begin{array}{rcccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



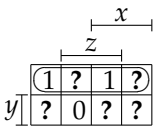
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\quad\quad\quad\quad \neg z \quad) \quad \vee \\ & & (\quad\quad y \quad\quad\quad) \end{array}$$

S176. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ \neg y \ \wedge \ \neg z \ ) \ \vee \\ & & (\ \ x \ \wedge \ \neg y \ \wedge \ \ z \ ) \end{array}$$

b Annotated Karnaugh map



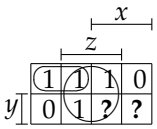
and associated, optimised implementation:

$$r = (\quad\quad \neg y \quad\quad)$$

S177. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ \neg y \ \wedge \ \neg z \ ) \ \vee \\ & & (\ \neg x \ \wedge \ \neg y \ \wedge \ \ z \ ) \ \vee \\ & & (\ \neg x \ \wedge \ \ y \ \wedge \ \ z \ ) \ \vee \\ & & (\ \ x \ \wedge \ \neg y \ \wedge \ \ z \ ) \end{array}$$

b Annotated Karnaugh map



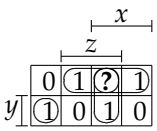
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ \neg y \quad\quad) \ \vee \\ & & (\quad\quad\quad\quad z \ ) \end{array}$$

S178. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ \neg y \ \wedge \ \ z \ ) \ \vee \\ & & (\ \neg x \ \wedge \ \ y \ \wedge \ \neg z \ ) \ \vee \\ & & (\ \ x \ \wedge \ \neg y \ \wedge \ \neg z \ ) \ \vee \\ & & (\ \ x \ \wedge \ \ y \ \wedge \ \ z \ ) \end{array}$$

b Annotated Karnaugh map



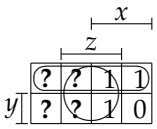
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ \ y \ \wedge \ \neg z \ ) \ \vee \\ & & (\ \ x \ \wedge \ \neg y \quad\quad) \ \vee \\ & & (\ \ x \quad\quad\quad \wedge \ \ z \ ) \ \vee \\ & & (\quad\quad \neg y \ \wedge \ \ z \ ) \end{array}$$

S179. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ x \ \wedge \ \neg y \ \wedge \ \neg z \ ) \ \vee \\ & & (\ x \ \wedge \ \neg y \ \wedge \ \ z \ ) \ \vee \\ & & (\ x \ \wedge \ \ y \ \wedge \ \ z \ ) \end{array}$$

b Annotated Karnaugh map



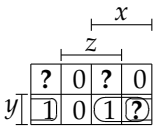
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\quad\quad\quad\quad\quad z\quad) \vee \\ & & (\quad\quad \neg y \quad\quad\quad) \end{array}$$

S180. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ y \ \wedge \ \neg z\ ) \vee \\ & & (\ \ x \ \wedge \ y \ \wedge \ \ z\ ) \end{array}$$

b Annotated Karnaugh map



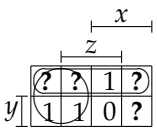
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ x \ \wedge \ y\quad\quad\quad) \vee \\ & & (\quad\quad y \ \wedge \ \neg z\ ) \end{array}$$

S181. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ \ y \ \wedge \ \neg z\ ) \vee \\ & & (\ \neg x \ \wedge \ \ y \ \wedge \ \ z\ ) \vee \\ & & (\ \ x \ \wedge \ \neg y \ \wedge \ \ z\ ) \end{array}$$

b Annotated Karnaugh map



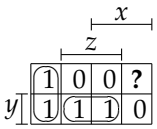
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \neg x\quad\quad\quad\quad) \vee \\ & & (\quad\quad \neg y\quad\quad) \end{array}$$

S182. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg x \ \wedge \ \neg y \ \wedge \ \neg z\ ) \vee \\ & & (\ \neg x \ \wedge \ \ y \ \wedge \ \neg z\ ) \vee \\ & & (\ \neg x \ \wedge \ \ y \ \wedge \ \ z\ ) \vee \\ & & (\ \ x \ \wedge \ \ y \ \wedge \ \ z\ ) \end{array}$$

b Annotated Karnaugh map



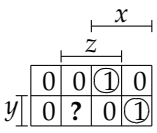
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\quad\quad\ y \ \wedge \ \ z\ ) \vee \\ & & (\ \neg x\quad\quad \wedge \ \neg z\ ) \end{array}$$

S183. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & \neg z &) & \end{array}$$

b Annotated Karnaugh map



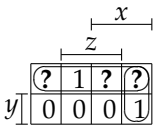
and associated, optimised implementation:

$$\begin{array}{rcccccccccc} r & = & (& x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \end{array}$$

S184. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & \neg z &) & \end{array}$$

b Annotated Karnaugh map



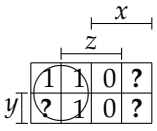
and associated, optimised implementation:

$$\begin{array}{rcccccccccc} r & = & (& x & & & \wedge & \neg z &) & \vee \\ & & (& & & \neg y & & &) & \end{array}$$

S185. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



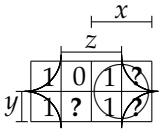
and associated, optimised implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg x & & & & &) & \end{array}$$

S186. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



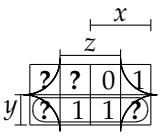
and associated, optimised implementation:

$$\begin{array}{lcl} r & = & (\quad \quad \neg z \quad) \vee \\ & & (\quad x \quad \quad \quad) \end{array}$$

S187. a Reference implementation:

$$\begin{array}{lcl} r & = & (\quad \neg x \wedge \quad y \wedge \quad z \quad) \vee \\ & & (\quad x \wedge \neg y \wedge \neg z \quad) \vee \\ & & (\quad x \wedge \quad y \wedge \quad z \quad) \end{array}$$

b Annotated Karnaugh map



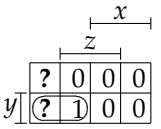
and associated, optimised implementation:

$$\begin{array}{lcl} r & = & (\quad \quad \neg z \quad) \vee \\ & & (\quad y \quad \quad) \end{array}$$

S188. a Reference implementation:

$$r = (\quad \neg x \wedge \quad y \wedge \quad z \quad)$$

b Annotated Karnaugh map



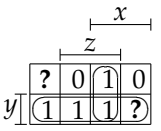
and associated, optimised implementation:

$$r = (\quad \neg x \wedge \quad y \quad)$$

S189. a Reference implementation:

$$\begin{array}{lcl} r & = & (\quad \neg x \wedge \quad y \wedge \neg z \quad) \vee \\ & & (\quad \neg x \wedge \quad y \wedge \quad z \quad) \vee \\ & & (\quad x \wedge \neg y \wedge \quad z \quad) \vee \\ & & (\quad x \wedge \quad y \wedge \quad z \quad) \end{array}$$

b Annotated Karnaugh map



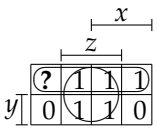
and associated, optimised implementation:

$$\begin{array}{lcl} r & = & (\quad \quad y \quad \quad) \vee \\ & & (\quad x \quad \quad \wedge \quad z \quad) \end{array}$$

S190. a Reference implementation:

$$\begin{array}{lcl} r & = & (\quad \neg x \wedge \neg y \wedge \quad z \quad) \vee \\ & & (\quad \neg x \wedge \quad y \wedge \quad z \quad) \vee \\ & & (\quad x \wedge \neg y \wedge \neg z \quad) \vee \\ & & (\quad x \wedge \neg y \wedge \quad z \quad) \vee \\ & & (\quad x \wedge \quad y \wedge \quad z \quad) \end{array}$$

b Annotated Karnaugh map



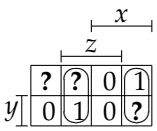
and associated, optimised implementation:

$$r = (\ z\ ) \vee (\ \neg y\ )$$

S191. a Reference implementation:

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

b Annotated Karnaugh map



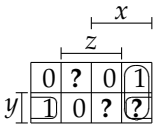
and associated, optimised implementation:

$$r = (\ \neg x \wedge z\ ) \vee (\ x \wedge \neg z\ )$$

S192. a Reference implementation:

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

b Annotated Karnaugh map



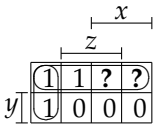
and associated, optimised implementation:

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

S193. a Reference implementation:

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

b Annotated Karnaugh map



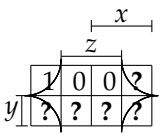
and associated, optimised implementation:

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

S194. a Reference implementation:

$$r = (\neg x \wedge \neg y \wedge \neg z)$$

b Annotated Karnaugh map



and associated, optimised implementation:

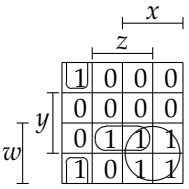
$$r = (\neg z)$$

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

S195. a Reference implementation:

$$\begin{aligned} r = {} & (\neg w \wedge \neg x \wedge \neg y \wedge \neg z) \vee \\ & (w \wedge \neg x \wedge \neg y \wedge \neg z) \vee \\ & (w \wedge \neg x \wedge y \wedge z) \vee \\ & (w \wedge x \wedge \neg y \wedge \neg z) \vee \\ & (w \wedge x \wedge \neg y \wedge z) \vee \\ & (w \wedge x \wedge y \wedge \neg z) \vee \\ & (w \wedge x \wedge y \wedge z) \end{aligned}$$

b Annotated Karnaugh map



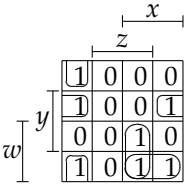
and associated, optimised implementation:

$$\begin{aligned} r = {} & (\neg x \wedge \neg y \wedge \neg z) \vee \\ & (w \wedge y \wedge z) \vee \\ & (w \wedge x) \end{aligned}$$

S196. a Reference implementation:

$$\begin{aligned} r = {} & (\neg w \wedge \neg x \wedge \neg y \wedge \neg z) \vee \\ & (\neg w \wedge \neg x \wedge y \wedge \neg z) \vee \\ & (\neg w \wedge x \wedge y \wedge \neg z) \vee \\ & (w \wedge \neg x \wedge \neg y \wedge \neg z) \vee \\ & (w \wedge x \wedge \neg y \wedge \neg z) \vee \\ & (w \wedge x \wedge \neg y \wedge z) \vee \\ & (w \wedge x \wedge y \wedge z) \end{aligned}$$

b Annotated Karnaugh map



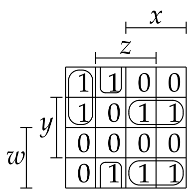
and associated, optimised implementation:

$$\begin{array}{lcl} r & = & (w \wedge x \wedge \neg y) \vee \\ & & (w \wedge x \wedge z) \vee \\ & & (\neg x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg w \wedge y \wedge \neg z) \end{array}$$

S197. a Reference implementation:

$$\begin{array}{lcl} r & = & (\neg w \wedge \neg x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg w \wedge \neg x \wedge \neg y \wedge z) \vee \\ & & (\neg w \wedge \neg x \wedge y \wedge \neg z) \vee \\ & & (\neg w \wedge x \wedge y \wedge \neg z) \vee \\ & & (\neg w \wedge x \wedge y \wedge z) \vee \\ & & (w \wedge \neg x \wedge \neg y \wedge z) \vee \\ & & (w \wedge x \wedge \neg y \wedge \neg z) \vee \\ & & (w \wedge x \wedge \neg y \wedge z) \end{array}$$

b Annotated Karnaugh map



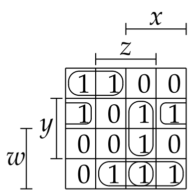
and associated, optimised implementation:

$$\begin{array}{lcl} r & = & (\neg x \wedge \neg y \wedge z) \vee \\ & & (w \wedge x \wedge \neg y) \vee \\ & & (\neg w \wedge x \wedge y) \vee \\ & & (\neg w \wedge \neg x \wedge \neg z) \end{array}$$

S198. a Reference implementation:

$$\begin{array}{lcl} r & = & (\neg w \wedge \neg x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg w \wedge \neg x \wedge \neg y \wedge z) \vee \\ & & (\neg w \wedge \neg x \wedge y \wedge \neg z) \vee \\ & & (\neg w \wedge x \wedge y \wedge \neg z) \vee \\ & & (\neg w \wedge x \wedge y \wedge z) \vee \\ & & (w \wedge \neg x \wedge \neg y \wedge z) \vee \\ & & (w \wedge x \wedge \neg y \wedge \neg z) \vee \\ & & (w \wedge x \wedge \neg y \wedge z) \vee \\ & & (w \wedge x \wedge y \wedge z) \end{array}$$

b Annotated Karnaugh map



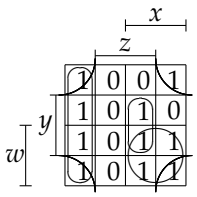
and associated, optimised implementation:

$$\begin{array}{lcl} r & = & (\neg w \wedge \neg x \wedge \neg y) \vee \\ & & (w \wedge x \wedge \neg y) \vee \\ & & (\neg w \wedge y \wedge \neg z) \vee \\ & & (w \wedge \neg y \wedge z) \vee \\ & & (x \wedge y \wedge z) \end{array}$$

S199. a Reference implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}
$$

b Annotated Karnaugh map



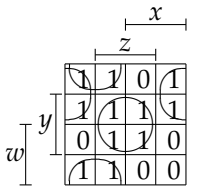
and associated, optimised implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& & & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& & & \neg x & & & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & & & & &) & \vee \\
 & & (& & & & & \neg y & \wedge & \neg z &) &
\end{array}
$$

S200. a Reference implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}
$$

b Annotated Karnaugh map



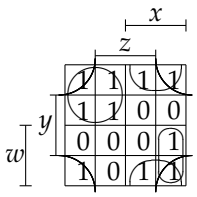
and associated, optimised implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& \neg w & & & & & \wedge & \neg z &) & \vee \\
 & & (& & & & & y & \wedge & z &) & \vee \\
 & & (& & & \neg x & \wedge & \neg y & & &) &
\end{array}
$$

S201. a Reference implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) &
\end{array}
$$

b Annotated Karnaugh map



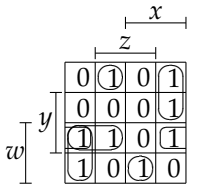
and associated, optimised implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& w & \wedge & x & & & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & & & & &) & \vee \\
 & & (& & & x & \wedge & \neg y & & &) & \vee \\
 & & (& & & & & \neg y & \wedge & \neg z &)
\end{array}
$$

S202. a Reference implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) &
\end{array}
$$

b Annotated Karnaugh map



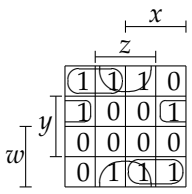
and associated, optimised implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& w & & & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & & &) & \vee \\
 & & (& \neg w & \wedge & x & & & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & & & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) &
\end{array}
$$

S203. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



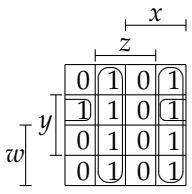
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y\ ) \vee \\
 & & (\ \neg w \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg y \wedge z\ )
\end{array}$$

S204. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge \neg z\ )
\end{array}$$

b Annotated Karnaugh map



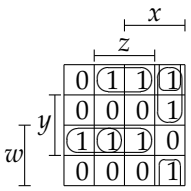
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \neg x \wedge z\ ) \vee \\
 & & (\ x \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge y \wedge \neg z\ )
\end{array}$$

S205. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



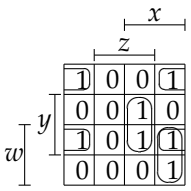
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ w \wedge y \wedge z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y\ ) \vee \\
 & & (\ x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge \neg y \wedge z\ )
\end{array}$$

S206. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



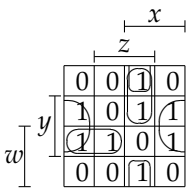
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ w \wedge x \wedge \neg z\ ) \vee \\
 & & (\ w \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ x \wedge y \wedge z\ )
\end{array}$$

S207. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge \neg z\ )
\end{array}$$

b Annotated Karnaugh map



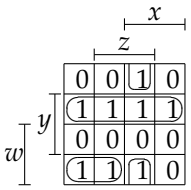
and associated, optimised implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & \wedge & x & & \wedge & z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & y & &) & \vee \\ & & (& & & x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& & & & & y & \wedge & \neg z &) \end{array}$$

S208. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) \end{array}$$

b Annotated Karnaugh map



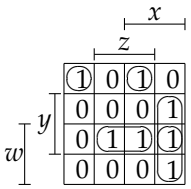
and associated, optimised implementation:

$$\begin{array}{rcccccccccc} r & = & (& w & \wedge & \neg x & \wedge & \neg y & & &) & \vee \\ & & (& \neg w & & & \wedge & y & & &) & \vee \\ & & (& & & x & \wedge & \neg y & \wedge & z &) & \end{array}$$

S209. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & x & \wedge & y & \wedge & z &) \end{array}$$

b Annotated Karnaugh map



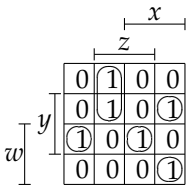
and associated, optimised implementation:

$$\begin{array}{rcccccccccc} r & = & (& w & \wedge & x & & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & & & \wedge & y & \wedge & z &) & \vee \\ & & (& & & x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \end{array}$$

S210. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



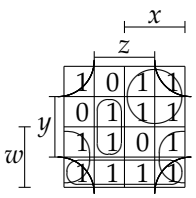
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & & & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) &
\end{array}$$

S211. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) &
\end{array}$$

b Annotated Karnaugh map



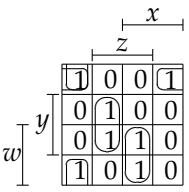
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& w & & & \wedge & \neg y & & &) & \vee \\
 & & (& & & & & \neg y & \wedge & \neg z &) & \vee \\
 & & (& & & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & & & & &) & \vee \\
 & & (& w & & & & & \wedge & \neg z &) &
\end{array}$$

S212. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



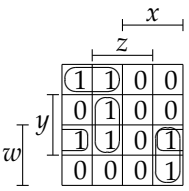
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& & & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& & & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & & & \wedge & z &) & \vee \\
 & & (& \neg w & & & \wedge & \neg y & \wedge & \neg z &) &
\end{array}$$

S213. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) &
\end{array}$$

b Annotated Karnaugh map



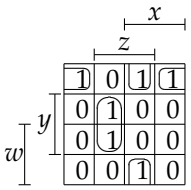
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & & &) & \vee \\
 & & (& w & & & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & & & \wedge & \neg z &) & \vee \\
 & & (& & & \neg x & \wedge & y & \wedge & z &) &
\end{array}$$

S214. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



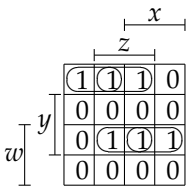
and associated, optimised implementation:

$$\begin{array}{rcllllll} r & = & (& & \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& & x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg w & & \wedge & \neg y & \wedge & \neg z &) & \end{array}$$

S215. a Reference implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



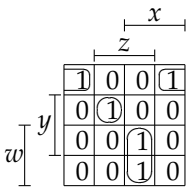
and associated, optimised implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & & &) & \vee \\ & & (& w & \wedge & x & \wedge & y & & &) & \vee \\ & & (& w & & & \wedge & y & \wedge & z &) & \vee \\ & & (& \neg w & & & \wedge & \neg y & \wedge & z &) & \end{array}$$

S216. a Reference implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& w & \wedge & x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



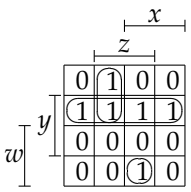
and associated, optimised implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& w & \wedge & x & & & \wedge & z &) & \vee \\ & & (& \neg w & & & \wedge & \neg y & \wedge & \neg z &) & \end{array}$$

S217. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



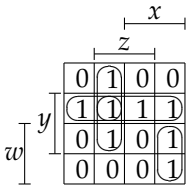
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & & & \wedge & z &) & \vee \\
 & & (& \neg w & & & \wedge & y & & &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) &
\end{array}$$

S218. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) &
\end{array}$$

b Annotated Karnaugh map



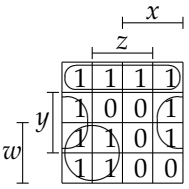
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & & & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & & & \wedge & \neg z &) & \vee \\
 & & (& \neg w & & & \wedge & y & & &) & \vee \\
 & & (& & & \neg x & \wedge & y & \wedge & z &) &
\end{array}$$

S219. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) &
\end{array}$$

b Annotated Karnaugh map



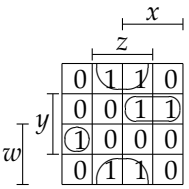
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ w \wedge \neg x \qquad\qquad\ )\ \vee \\ & & (\ \neg w \qquad \wedge \neg y \qquad\ )\ \vee \\ & & (\qquad\qquad\ y \wedge \neg z\ ) \end{array}$$

S220. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge z\ )\ \vee \\ & & (\ \neg w \wedge x \wedge \neg y \wedge z\ )\ \vee \\ & & (\ \neg w \wedge x \wedge y \wedge \neg z\ )\ \vee \\ & & (\ \neg w \wedge x \wedge y \wedge z\ )\ \vee \\ & & (\ w \wedge \neg x \wedge \neg y \wedge z\ )\ \vee \\ & & (\ w \wedge \neg x \wedge y \wedge \neg z\ )\ \vee \\ & & (\ w \wedge x \wedge \neg y \wedge z\ ) \end{array}$$

b Annotated Karnaugh map



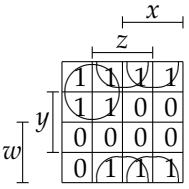
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ w \wedge \neg x \wedge y \wedge \neg z\ )\ \vee \\ & & (\ \neg w \wedge x \wedge y \qquad\ )\ \vee \\ & & (\qquad\qquad\ \neg y \wedge z\ ) \end{array}$$

S221. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge \neg z\ )\ \vee \\ & & (\ \neg w \wedge \neg x \wedge \neg y \wedge z\ )\ \vee \\ & & (\ \neg w \wedge \neg x \wedge y \wedge \neg z\ )\ \vee \\ & & (\ \neg w \wedge \neg x \wedge y \wedge z\ )\ \vee \\ & & (\ \neg w \wedge x \wedge \neg y \wedge \neg z\ )\ \vee \\ & & (\ \neg w \wedge x \wedge \neg y \wedge z\ )\ \vee \\ & & (\ w \wedge \neg x \wedge \neg y \wedge z\ )\ \vee \\ & & (\ w \wedge x \wedge \neg y \wedge \neg z\ )\ \vee \\ & & (\ w \wedge x \wedge \neg y \wedge z\ ) \end{array}$$

b Annotated Karnaugh map



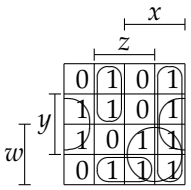
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \quad\quad\quad\ x \wedge \neg y \quad\quad\ )\ \vee \\
 & & (\ \neg w \wedge \neg x \quad\quad\quad\ )\ \vee \\
 & & (\ \quad\quad\quad\ \neg y \wedge z\ )
\end{array}$$

S222. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge z\ )\ \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge z\ )\ \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge \neg z\ )\ \vee \\
 & & (\ w \wedge \neg x \wedge \neg y \wedge z\ )\ \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge \neg z\ )\ \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge z\ )\ \vee \\
 & & (\ w \wedge x \wedge y \wedge \neg z\ )\ \vee \\
 & & (\ w \wedge x \wedge y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



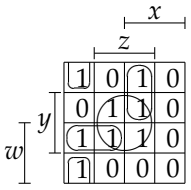
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \quad\quad\ \wedge z\ )\ \vee \\
 & & (\ \quad\quad\ x \quad\quad\ \wedge \neg z\ )\ \vee \\
 & & (\ w \wedge x \quad\quad\quad\ )\ \vee \\
 & & (\ \quad\quad\quad\ y \wedge \neg z\ )\ \vee \\
 & & (\ w \quad\quad\ \wedge \neg y \wedge z\ )
\end{array}$$

S223. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge z\ )\ \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge z\ )\ \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge z\ )\ \vee \\
 & & (\ w \wedge \neg x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge \neg z\ )\ \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge z\ )\ \vee \\
 & & (\ w \wedge x \wedge y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



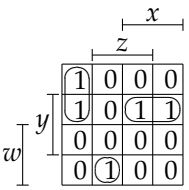
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \quad\quad\quad\ y \wedge z\ )\ \vee \\
 & & (\ \quad\quad \neg x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge x \quad\quad\ \wedge z\ )\ \vee \\
 & & (\ w \wedge \neg x \wedge y \quad\quad\ )
\end{array}$$

S224. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



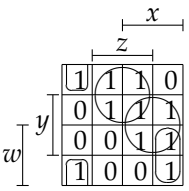
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & & &) & \vee \\
 & & (& \neg w & \wedge & \neg x & & & \wedge & \neg z &) &
\end{array}$$

S225. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



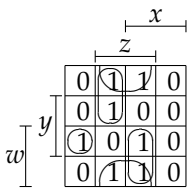
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& w & \wedge & x & & & \wedge & \neg z &) & \vee \\
 & & (& & & x & \wedge & y & & &) & \vee \\
 & & (& \neg w & & & & & \wedge & z &) & \vee \\
 & & (& & & \neg x & \wedge & \neg y & \wedge & \neg z &) &
\end{array}$$

S226. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



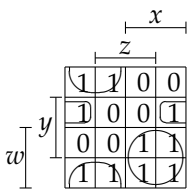
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge z\ ) \vee \\
 & & (\ \neg y \wedge z\ )
\end{array}$$

S227. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



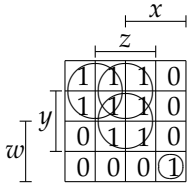
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \neg x \wedge \neg y\ ) \vee \\
 & & (\ \neg w \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x\ )
\end{array}$$

S228. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



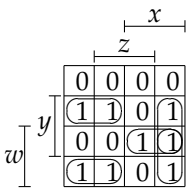
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& & & & & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & & & & &) & \vee \\
 & & (& \neg w & & & & & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) &
\end{array}$$

S229. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



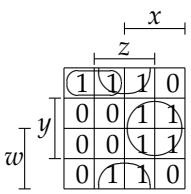
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& w & \wedge & x & & & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & & &) & \vee \\
 & & (& & & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & & &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & & &) &
\end{array}$$

S230. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



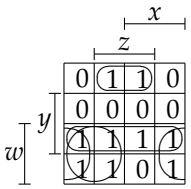
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & & &) & \vee \\
 & & (& & & x & \wedge & y & & &) & \vee \\
 & & (& & & & & \neg y & \wedge & z &) &
\end{array}$$

S231. a Reference implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}
$$

b Annotated Karnaugh map



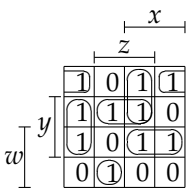
and associated, optimised implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& w & & & \wedge & y & & &) & \vee \\
 & & (& w & & & & & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & & & & &) & \vee \\
 & & (& \neg w & & & \wedge & \neg y & \wedge & z &) &
\end{array}
$$

S232. a Reference implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}
$$

b Annotated Karnaugh map



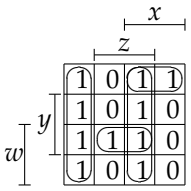
and associated, optimised implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & & & \wedge & z &) & \vee \\
 & & (& & & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & & &) & \vee \\
 & & (& \neg w & & & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & & & \wedge & y & \wedge & z &) &
\end{array}
$$

S233. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



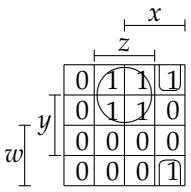
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& w & & & \wedge & y & \wedge & z &) & \vee \\
 & & (& & & \neg x & & & \wedge & \neg z &) & \vee \\
 & & (& & & x & & & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & & &) &
\end{array}$$

S234. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) &
\end{array}$$

b Annotated Karnaugh map



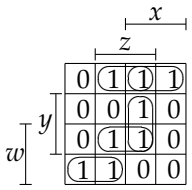
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & & & & & \wedge & z &) & \vee \\
 & & (& & & x & \wedge & \neg y & \wedge & \neg z &) &
\end{array}$$

S235. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



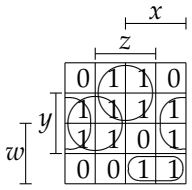
and associated, optimised implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& & & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & & & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & & &) & \vee \\
 & & (& \neg w & & & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & & &) &
\end{array}
$$

S236. a Reference implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) &
\end{array}
$$

b Annotated Karnaugh map



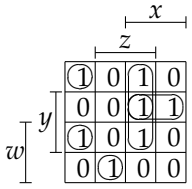
and associated, optimised implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& w & \wedge & x & \wedge & \neg y & & &) & \vee \\
 & & (& & & & & y & \wedge & \neg z &) & \vee \\
 & & (& & & \neg x & \wedge & y & & &) & \vee \\
 & & (& \neg w & & & & & \wedge & z &) &
\end{array}
$$

S237. a Reference implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}
$$

b Annotated Karnaugh map



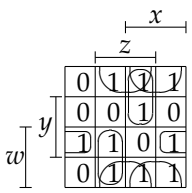
and associated, optimised implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& & & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & & &) & \vee \\
 & & (& \neg w & \wedge & x & & & \wedge & z &) &
\end{array}
$$

S238. a Reference implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) &
\end{array}
$$

b Annotated Karnaugh map



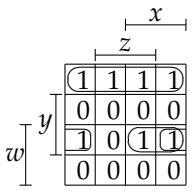
and associated, optimised implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& & & x & \wedge & \neg y & & &) & \vee \\
 & & (& w & & & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & & & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & & & \wedge & z &) & \vee \\
 & & (& & & & & \neg y & \wedge & z &) &
\end{array}
$$

S239. a Reference implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}
$$

b Annotated Karnaugh map



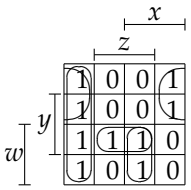
and associated, optimised implementation:

$$
\begin{array}{rcccccccccc}
r & = & (& w & \wedge & x & \wedge & y & & &) & \vee \\
 & & (& \neg w & & & \wedge & \neg y & & &) & \vee \\
 & & (& w & & & \wedge & y & \wedge & \neg z &) &
\end{array}
$$

S240. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



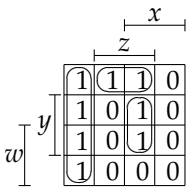
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& w & \wedge & x & & & \wedge & z &) & \vee \\
 & & (& w & & & \wedge & y & \wedge & z &) & \vee \\
 & & (& & & \neg x & & & \wedge & \neg z &) & \vee \\
 & & (& \neg w & & & & & \wedge & \neg z &) &
\end{array}$$

S241. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



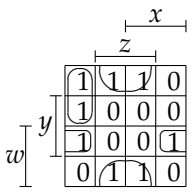
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& & & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& & & \neg x & & & \wedge & \neg z &) & \vee \\
 & & (& \neg w & & & \wedge & \neg y & \wedge & z &) &
\end{array}$$

S242. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) &
\end{array}$$

b Annotated Karnaugh map



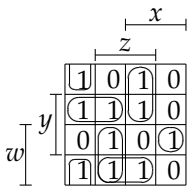
and associated, optimised implementation:

$$\begin{array}{rcllllllll} r & = & (& w & & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& & & & \neg y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge \neg x & & & \wedge & \neg z &) & \end{array}$$

S243. a Reference implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge x & \wedge & y & \wedge & z &) & \vee \\ & & (& w & \wedge \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& w & \wedge \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& w & \wedge x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& w & \wedge x & \wedge & y & \wedge & \neg z &) & \end{array}$$

b Annotated Karnaugh map



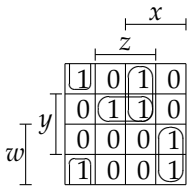
and associated, optimised implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge \neg x & \wedge & y & & &) & \vee \\ & & (& w & \wedge \neg x & & & \wedge & z &) & \vee \\ & & (& w & \wedge x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& w & & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge x & & & \wedge & z &) & \end{array}$$

S244. a Reference implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge x & \wedge & y & \wedge & z &) & \vee \\ & & (& w & \wedge \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge x & \wedge & y & \wedge & \neg z &) & \end{array}$$

b Annotated Karnaugh map



and associated, optimised implementation:

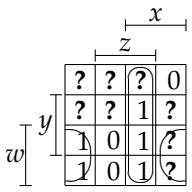
$$\begin{array}{rcl}
r & = & (\ w \wedge x \wedge z\ ) \vee \\
 & & (\ \neg x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge z\ )
\end{array}$$

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

S245. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge x \wedge y \wedge z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



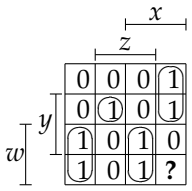
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ x \wedge z\ ) \vee \\
 & & (\ w \wedge \neg z\ )
\end{array}$$

S246. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



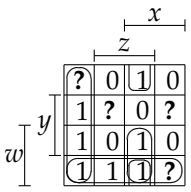
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ w \wedge \neg x \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge z\ )
\end{array}$$

S247. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



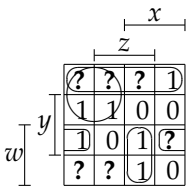
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ w \wedge x \wedge z\ ) \vee \\
 & & (\ x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge \neg y\ ) \vee \\
 & & (\ \neg x \wedge \neg z\ )
\end{array}$$

S248. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



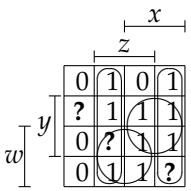
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ w \wedge x \wedge z\ ) \vee \\
 & & (\ w \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge \neg y\ ) \vee \\
 & & (\ \neg w \wedge \neg x\ )
\end{array}$$

S249. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge \neg x \wedge y \wedge z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge \neg y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ \neg w \wedge x \wedge y \wedge z\ ) \vee \\
 & & (\ w \wedge \neg x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge \neg y \wedge z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge \neg z\ ) \vee \\
 & & (\ w \wedge x \wedge y \wedge z\ )
\end{array}$$

b Annotated Karnaugh map



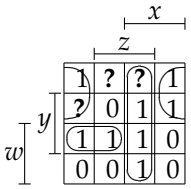
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \ \ \ x \ \ \ \ \wedge\ \neg z\ )\ \vee \\
 & & (\ \ \ \ x \ \wedge\ y \ \ \ \ )\ \vee \\
 & & (\ \ \ \neg x \ \ \ \ \wedge\ \ z\ )\ \vee \\
 & & (\ w \ \ \ \ \ \ \ \ \wedge\ \ z\ )
\end{array}$$

S250. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge\ \ x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge\ \ x \wedge\ \ y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge\ \ x \wedge\ \ y \wedge\ \ z\ )\ \vee \\
 & & (\ \ \ w \wedge \neg x \wedge\ \ y \wedge \neg z\ )\ \vee \\
 & & (\ \ \ w \wedge \neg x \wedge\ \ y \wedge\ \ z\ )\ \vee \\
 & & (\ \ \ w \wedge\ \ x \wedge \neg y \wedge\ \ z\ )\ \vee \\
 & & (\ \ \ w \wedge\ \ x \wedge\ \ y \wedge\ \ z\ )
\end{array}$$

b Annotated Karnaugh map



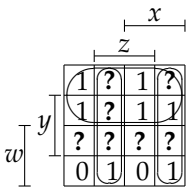
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \ \ \ \ \ \ \ \ \wedge\ \neg z\ )\ \vee \\
 & & (\ \ \ \ \ \ x \ \ \ \ \wedge\ \ z\ )\ \vee \\
 & & (\ \ \ w \wedge \neg x \wedge\ y \ \ \ \ )
\end{array}$$

S251. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge \neg x \wedge\ \ y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge\ \ x \wedge \neg y \wedge\ \ z\ )\ \vee \\
 & & (\ \neg w \wedge\ \ x \wedge\ \ y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge\ \ x \wedge\ \ y \wedge\ \ z\ )\ \vee \\
 & & (\ \ \ w \wedge \neg x \wedge \neg y \wedge\ \ z\ )\ \vee \\
 & & (\ \ \ w \wedge\ \ x \wedge \neg y \wedge \neg z\ )
\end{array}$$

b Annotated Karnaugh map



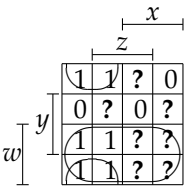
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \ \ \ \neg x \ \ \ \ \wedge\ \ z\ )\ \vee \\
 & & (\ \ \ \ \ \ x \ \ \ \ \wedge\ \neg z\ )\ \vee \\
 & & (\ \neg w \ \ \ \ \ \ \ \ \ \ \ \ )
\end{array}$$

S252. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



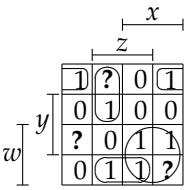
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& & & \neg x & \wedge & \neg y & & &) & \vee \\
 & & (& w & & & & & & &) &
\end{array}$$

S253. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



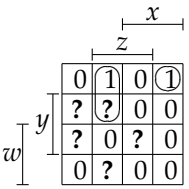
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & & & \wedge & z &) & \vee \\
 & & (& w & & & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & & & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & & & & &) &
\end{array}$$

S254. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) &
\end{array}$$

b Annotated Karnaugh map



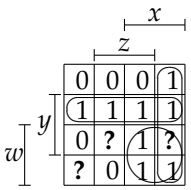
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\neg w \wedge \neg x \wedge z) \vee \\ & & (\neg w \wedge x \wedge \neg y \wedge \neg z) \end{array}$$

S255. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg w \wedge \neg x \wedge y \wedge \neg z) \vee \\ & & (\neg w \wedge \neg x \wedge y \wedge z) \vee \\ & & (\neg w \wedge x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg w \wedge x \wedge y \wedge \neg z) \vee \\ & & (\neg w \wedge x \wedge y \wedge z) \vee \\ & & (w \wedge x \wedge \neg y \wedge \neg z) \vee \\ & & (w \wedge x \wedge \neg y \wedge z) \vee \\ & & (w \wedge x \wedge y \wedge z) \end{array}$$

b Annotated Karnaugh map



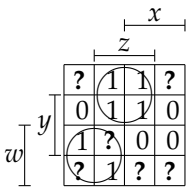
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (x \wedge \neg z) \vee \\ & & (\neg w \wedge y) \vee \\ & & (w \wedge x) \end{array}$$

S256. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg w \wedge \neg x \wedge \neg y \wedge z) \vee \\ & & (\neg w \wedge \neg x \wedge y \wedge z) \vee \\ & & (\neg w \wedge x \wedge \neg y \wedge z) \vee \\ & & (\neg w \wedge x \wedge y \wedge z) \vee \\ & & (w \wedge \neg x \wedge \neg y \wedge z) \vee \\ & & (w \wedge \neg x \wedge y \wedge \neg z) \end{array}$$

b Annotated Karnaugh map



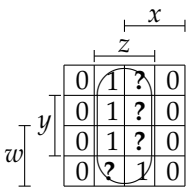
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (w \wedge \neg x) \vee \\ & & (\neg w \wedge z) \end{array}$$

S257. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg w \wedge \neg x \wedge \neg y \wedge z) \vee \\ & & (\neg w \wedge \neg x \wedge y \wedge z) \vee \\ & & (w \wedge \neg x \wedge y \wedge z) \vee \\ & & (w \wedge x \wedge \neg y \wedge z) \end{array}$$

b Annotated Karnaugh map



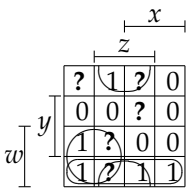
and associated, optimised implementation:

$$r = (\quad z \quad)$$

S258. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg w \wedge \neg x \wedge \neg y \wedge z) \vee \\ & & (w \wedge \neg x \wedge \neg y \wedge \neg z) \vee \\ & & (w \wedge \neg x \wedge y \wedge \neg z) \vee \\ & & (w \wedge x \wedge \neg y \wedge \neg z) \vee \\ & & (w \wedge x \wedge \neg y \wedge z) \end{array}$$

b Annotated Karnaugh map



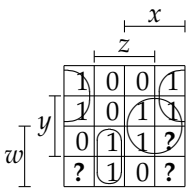
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (w \wedge \neg y) \vee \\ & & (w \wedge \neg x) \vee \\ & & (\neg y \wedge z) \end{array}$$

S259. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg w \wedge \neg x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg w \wedge \neg x \wedge y \wedge \neg z) \vee \\ & & (\neg w \wedge x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg w \wedge x \wedge y \wedge \neg z) \vee \\ & & (\neg w \wedge x \wedge y \wedge z) \vee \\ & & (w \wedge \neg x \wedge \neg y \wedge z) \vee \\ & & (w \wedge \neg x \wedge y \wedge z) \vee \\ & & (w \wedge x \wedge y \wedge z) \end{array}$$

b Annotated Karnaugh map



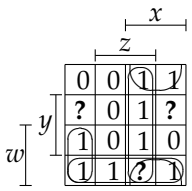
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (w \wedge \neg x \wedge z) \vee \\ & & (x \wedge y) \vee \\ & & (\neg w \wedge \neg z) \end{array}$$

S260. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



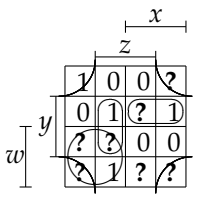
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& & & x & \wedge & \neg y & & &) & \vee \\
 & & (& w & \wedge & \neg x & & & \wedge & \neg z &) & \vee \\
 & & (& w & & & \wedge & \neg y & & &) & \vee \\
 & & (& & & x & & & \wedge & z &) &
\end{array}$$

S261. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



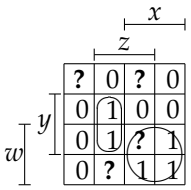
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& & & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & & & & &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & & &) & \vee \\
 & & (& & & & & \neg y & \wedge & \neg z &) &
\end{array}$$

S262. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) &
\end{array}$$

b Annotated Karnaugh map



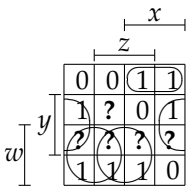
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\quad \neg x \wedge y \wedge z \quad) \vee \\ & & (w \wedge x \quad) \end{array}$$

S263. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg w \wedge \neg x \wedge y \wedge \neg z) \vee \\ & & (\neg w \wedge x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg w \wedge x \wedge \neg y \wedge z) \vee \\ & & (\neg w \wedge x \wedge y \wedge \neg z) \vee \\ & & (w \wedge \neg x \wedge \neg y \wedge \neg z) \vee \\ & & (w \wedge \neg x \wedge \neg y \wedge z) \vee \\ & & (w \wedge x \wedge \neg y \wedge z) \end{array}$$

b Annotated Karnaugh map



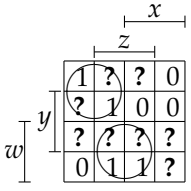
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\quad w \quad\quad\quad\quad \wedge \quad z \quad) \vee \\ & & (\quad w \wedge \neg x \quad\quad\quad\quad\quad) \vee \\ & & (\quad\quad\quad\quad\quad y \wedge \neg z \quad) \vee \\ & & (\neg w \wedge x \wedge \neg y \quad\quad\quad) \end{array}$$

S264. a Reference implementation:

$$\begin{array}{rcl} r & = & (\neg w \wedge \neg x \wedge \neg y \wedge \neg z) \vee \\ & & (\neg w \wedge \neg x \wedge y \wedge z) \vee \\ & & (w \wedge \neg x \wedge \neg y \wedge z) \vee \\ & & (w \wedge x \wedge \neg y \wedge z) \end{array}$$

b Annotated Karnaugh map



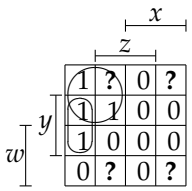
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\neg w \wedge \neg x \quad\quad\quad) \vee \\ & & (\quad w \quad\quad\quad \wedge z) \end{array}$$

S265. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \end{array}$$

b Annotated Karnaugh map



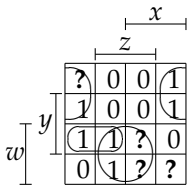
and associated, optimised implementation:

$$\begin{array}{rcccccccccc} r & = & (& & & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & \neg x & & & & &) & \end{array}$$

S266. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



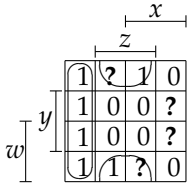
and associated, optimised implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & & & & & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & y & & &) & \vee \\ & & (& w & & & & & \wedge & z &) & \end{array}$$

S267. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \end{array}$$

b Annotated Karnaugh map



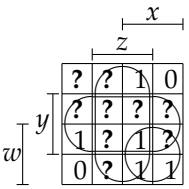
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\quad \neg x \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad \neg y \quad \wedge \quad z \quad) \end{array}$$

S268. a Reference implementation:

$$\begin{array}{rcl} r & = & (\quad \neg w \quad \wedge \quad x \quad \wedge \quad \neg y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad \neg x \quad \wedge \quad y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad x \quad \wedge \quad \neg y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad x \quad \wedge \quad \neg y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad x \quad \wedge \quad y \quad \wedge \quad z \quad) \end{array}$$

b Annotated Karnaugh map



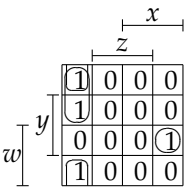
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\quad \quad \quad \quad z \quad) \quad \vee \\ & & (\quad \quad y \quad \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad x \quad \quad) \end{array}$$

S269. a Reference implementation:

$$\begin{array}{rcl} r & = & (\quad \neg w \quad \wedge \quad \neg x \quad \wedge \quad \neg y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad \neg x \quad \wedge \quad y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad \neg x \quad \wedge \quad \neg y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad x \quad \wedge \quad y \quad \wedge \quad \neg z \quad) \end{array}$$

b Annotated Karnaugh map



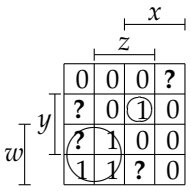
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\quad \quad \neg x \quad \wedge \quad \neg y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad x \quad \wedge \quad y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad \neg x \quad \quad \wedge \quad \neg z \quad) \end{array}$$

S270. a Reference implementation:

$$\begin{array}{rcl} r & = & (\quad \neg w \quad \wedge \quad x \quad \wedge \quad y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad \neg x \quad \wedge \quad \neg y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad \neg x \quad \wedge \quad \neg y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad \neg x \quad \wedge \quad y \quad \wedge \quad z \quad) \end{array}$$

b Annotated Karnaugh map



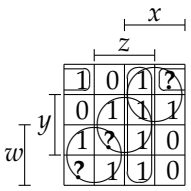
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\quad w \quad \wedge \quad \neg x \qquad\qquad\qquad\quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad x \quad \wedge \quad y \quad \wedge \quad z \quad) \end{array}$$

S271. a Reference implementation:

$$\begin{array}{rcl} r & = & (\quad \neg w \quad \wedge \quad \neg x \quad \wedge \quad \neg y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad \neg x \quad \wedge \quad y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad x \quad \wedge \quad \neg y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad x \quad \wedge \quad y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad x \quad \wedge \quad y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad \neg x \quad \wedge \quad \neg y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad \neg x \quad \wedge \quad y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad x \quad \wedge \quad \neg y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad x \quad \wedge \quad y \quad \wedge \quad z \quad) \end{array}$$

b Annotated Karnaugh map



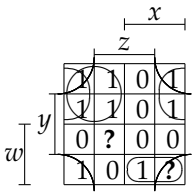
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\quad w \quad \wedge \quad \neg x \qquad\qquad\qquad\quad) \quad \vee \\ & & (\qquad\qquad\qquad\quad y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad x \qquad\qquad\qquad\quad) \quad \vee \\ & & (\quad \neg w \qquad\quad \wedge \quad \neg y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\qquad\quad x \qquad\quad \wedge \quad z \quad) \end{array}$$

S272. a Reference implementation:

$$\begin{array}{rcl} r & = & (\quad \neg w \quad \wedge \quad \neg x \quad \wedge \quad \neg y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad \neg x \quad \wedge \quad \neg y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad \neg x \quad \wedge \quad y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad \neg x \quad \wedge \quad y \quad \wedge \quad z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad x \quad \wedge \quad \neg y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad \neg w \quad \wedge \quad x \quad \wedge \quad y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad \neg x \quad \wedge \quad \neg y \quad \wedge \quad \neg z \quad) \quad \vee \\ & & (\quad w \quad \wedge \quad x \quad \wedge \quad \neg y \quad \wedge \quad z \quad) \end{array}$$

b Annotated Karnaugh map



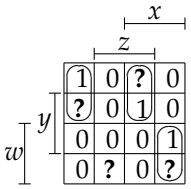
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \qquad\qquad\quad \wedge\ \neg z\ )\ \vee \\
 & & (\ \neg w \wedge \neg x \qquad\quad\ \ )\ \vee \\
 & & (\ \ w \wedge\ x \wedge \neg y \qquad\ )\ \vee \\
 & & (\qquad\quad\ \ \neg y \wedge \neg z\ )
\end{array}$$

S273. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge\ x \wedge\ y \wedge\ z\ )\ \vee \\
 & & (\ \ w \wedge\ x \wedge\ y \wedge \neg z\ )
\end{array}$$

b Annotated Karnaugh map



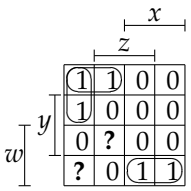
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \ w \wedge\ x \qquad\ \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge\ x \qquad\ \wedge\ z\ )\ \vee \\
 & & (\ \neg w \wedge \neg x \qquad\ \wedge \neg z\ )
\end{array}$$

S274. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge \neg x \wedge \neg y \wedge\ z\ )\ \vee \\
 & & (\ \neg w \wedge \neg x \wedge\ y \wedge \neg z\ )\ \vee \\
 & & (\ \ w \wedge\ x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ \ w \wedge\ x \wedge \neg y \wedge\ z\ )
\end{array}$$

b Annotated Karnaugh map



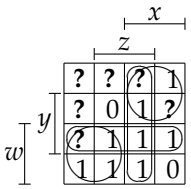
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge \neg x \wedge \neg y \qquad\ )\ \vee \\
 & & (\ \ w \wedge\ x \wedge \neg y \qquad\ )\ \vee \\
 & & (\ \neg w \wedge \neg x \qquad\ \wedge \neg z\ )
\end{array}$$

S275. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \wedge\ x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ \neg w \wedge\ x \wedge\ y \wedge\ z\ )\ \vee \\
 & & (\ \ w \wedge \neg x \wedge \neg y \wedge \neg z\ )\ \vee \\
 & & (\ \ w \wedge \neg x \wedge \neg y \wedge\ z\ )\ \vee \\
 & & (\ \ w \wedge \neg x \wedge\ y \wedge\ z\ )\ \vee \\
 & & (\ \ w \wedge\ x \wedge \neg y \wedge\ z\ )\ \vee \\
 & & (\ \ w \wedge\ x \wedge\ y \wedge \neg z\ )\ \vee \\
 & & (\ \ w \wedge\ x \wedge\ y \wedge\ z\ )
\end{array}$$

b Annotated Karnaugh map



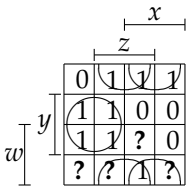
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ w \wedge \neg x \qquad\qquad\ \ )\ \vee \\ & & (\ \qquad\ x \qquad\ \wedge\ z\ )\ \vee \\ & & (\ w \qquad\ \wedge\ y \qquad\ )\ \vee \\ & & (\ \neg w \wedge\ x \qquad\qquad\ )\end{array}$$

S276. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge\ z\ )\ \vee \\ & & (\ \neg w \wedge \neg x \wedge\ y \wedge \neg z\ )\ \vee \\ & & (\ \neg w \wedge \neg x \wedge\ y \wedge\ z\ )\ \vee \\ & & (\ \neg w \wedge\ x \wedge \neg y \wedge \neg z\ )\ \vee \\ & & (\ \neg w \wedge\ x \wedge \neg y \wedge\ z\ )\ \vee \\ & & (\ w \wedge \neg x \wedge\ y \wedge \neg z\ )\ \vee \\ & & (\ w \wedge \neg x \wedge\ y \wedge\ z\ )\ \vee \\ & & (\ w \wedge\ x \wedge \neg y \wedge\ z\ )\end{array}$$

b Annotated Karnaugh map



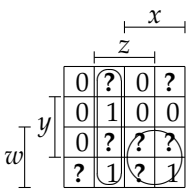
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \qquad x \wedge \neg y \qquad\ )\ \vee \\ & & (\ \qquad\ \neg y \wedge\ z\ )\ \vee \\ & & (\ \neg x \wedge\ y \qquad\ )\end{array}$$

S277. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg w \wedge \neg x \wedge\ y \wedge\ z\ )\ \vee \\ & & (\ w \wedge \neg x \wedge \neg y \wedge\ z\ )\ \vee \\ & & (\ w \wedge\ x \wedge \neg y \wedge \neg z\ )\end{array}$$

b Annotated Karnaugh map



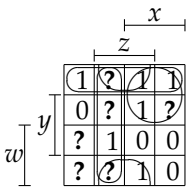
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \qquad\ \neg x \qquad\ \wedge\ z\ )\ \vee \\ & & (\ w \wedge\ x \qquad\ )\end{array}$$

S278. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



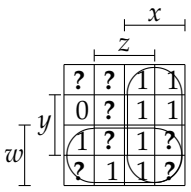
and associated, optimised implementation:

$$\begin{array}{rcccccccccc}
r & = & (& & & \neg x & & & \wedge & z &) & \vee \\
 & & (& \neg w & & & \wedge & \neg y & & &) & \vee \\
 & & (& \neg w & \wedge & x & & & & &) & \vee \\
 & & (& & & & & \neg y & \wedge & z &) &
\end{array}$$

S279. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



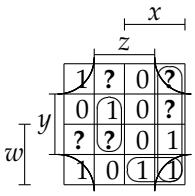
and associated, optimised implementation:

$$\begin{array}{rcccccc}
r & = & (& & x & &) & \vee \\
 & & (& w & & &) &
\end{array}$$

S280. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) &
\end{array}$$

b Annotated Karnaugh map



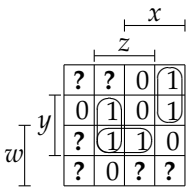
and associated, optimised implementation:

$$\begin{array}{rcccccccccc} r & = & (& w & \wedge & x & \wedge & \neg y & & &) & \vee \\ & & (& & & x & & & \wedge & \neg z &) & \vee \\ & & (& & & & & \neg y & \wedge & \neg z &) & \vee \\ & & (& & \neg x & \wedge & y & \wedge & z & &) & \end{array}$$

S281. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& w & \wedge & x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



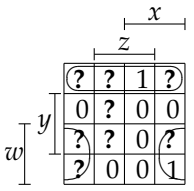
and associated, optimised implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & \wedge & x & & & \wedge & \neg z &) & \vee \\ & & (& & & \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& w & & & \wedge & y & \wedge & z &) & \end{array}$$

S282. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \end{array}$$

b Annotated Karnaugh map



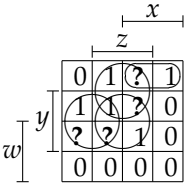
and associated, optimised implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & & & \wedge & \neg y & & &) & \vee \\ & & (& w & & & & & \wedge & \neg z &) & \end{array}$$

S283. a Reference implementation:

$$\begin{array}{rcccccccccc} r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



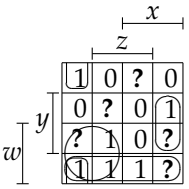
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w \qquad\qquad\qquad\qquad \wedge\ z\ )\ \vee \\
 & & (\ \qquad\qquad\qquad\ y\ \wedge\ z\ )\ \vee \\
 & & (\ \neg w\ \wedge\ x\ \wedge\ \neg y\ \qquad\ )\ \vee \\
 & & (\ \qquad\ \neg x\ \wedge\ y\ \qquad\ )
\end{array}$$

S284. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w\ \wedge\ \neg x\ \wedge\ \neg y\ \wedge\ \neg z\ )\ \vee \\
 & & (\ \neg w\ \wedge\ x\ \wedge\ y\ \wedge\ \neg z\ )\ \vee \\
 & & (\ w\ \wedge\ \neg x\ \wedge\ \neg y\ \wedge\ \neg z\ )\ \vee \\
 & & (\ w\ \wedge\ \neg x\ \wedge\ \neg y\ \wedge\ z\ )\ \vee \\
 & & (\ w\ \wedge\ \neg x\ \wedge\ y\ \wedge\ z\ )\ \vee \\
 & & (\ w\ \wedge\ x\ \wedge\ \neg y\ \wedge\ z\ )
\end{array}$$

b Annotated Karnaugh map



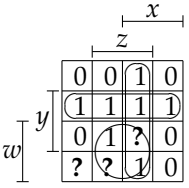
and associated, optimised implementation:

$$\begin{array}{rcl}
r & = & (\ w\ \wedge\ \neg x\ \qquad\qquad\qquad\ )\ \vee \\
 & & (\ \qquad\ x\ \wedge\ y\ \wedge\ \neg z\ )\ \vee \\
 & & (\ \qquad\ \neg x\ \wedge\ \neg y\ \wedge\ \neg z\ )\ \vee \\
 & & (\ w\ \qquad\ \wedge\ \neg y\ \qquad\ )
\end{array}$$

S285. a Reference implementation:

$$\begin{array}{rcl}
r & = & (\ \neg w\ \wedge\ \neg x\ \wedge\ y\ \wedge\ \neg z\ )\ \vee \\
 & & (\ \neg w\ \wedge\ \neg x\ \wedge\ y\ \wedge\ z\ )\ \vee \\
 & & (\ \neg w\ \wedge\ x\ \wedge\ \neg y\ \wedge\ z\ )\ \vee \\
 & & (\ \neg w\ \wedge\ x\ \wedge\ y\ \wedge\ \neg z\ )\ \vee \\
 & & (\ \neg w\ \wedge\ x\ \wedge\ y\ \wedge\ z\ )\ \vee \\
 & & (\ w\ \wedge\ \neg x\ \wedge\ y\ \wedge\ z\ )\ \vee \\
 & & (\ w\ \wedge\ x\ \wedge\ \neg y\ \wedge\ z\ )
\end{array}$$

b Annotated Karnaugh map



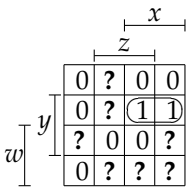
and associated, optimised implementation:

$$\begin{array}{rcllllllll} r & = & (& w & & & \wedge & z &) & \vee \\ & & (& \neg w & \wedge & y & & &) & \vee \\ & & (& & x & & \wedge & z &) & \end{array}$$

S286. a Reference implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



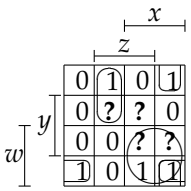
and associated, optimised implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge & x & \wedge & y & & &) \end{array}$$

S287. a Reference implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & x & \wedge & \neg y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



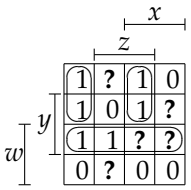
and associated, optimised implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge & \neg x & & & \wedge & z &) & \vee \\ & & (& & & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & x & & & & &) & \vee \\ & & (& w & & & \wedge & \neg y & \wedge & \neg z &) & \end{array}$$

S288. a Reference implementation:

$$\begin{array}{rcllllllll} r & = & (& \neg w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & \neg y & \wedge & z &) & \vee \\ & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\ & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \end{array}$$

b Annotated Karnaugh map



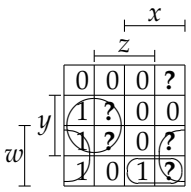
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ w \wedge y\ ) \vee \\ & & (\ \neg w \wedge x \wedge z\ ) \vee \\ & & (\ \neg w \wedge \neg x \wedge \neg z\ ) \end{array}$$

S289. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\ & & (\ w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\ & & (\ w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\ & & (\ w \wedge x \wedge \neg y \wedge z\ ) \end{array}$$

b Annotated Karnaugh map



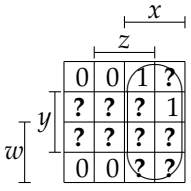
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ w \wedge \neg z\ ) \vee \\ & & (\ w \wedge x \wedge \neg y\ ) \vee \\ & & (\ \neg x \wedge y\ ) \end{array}$$

S290. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg w \wedge x \wedge \neg y \wedge z\ ) \vee \\ & & (\ \neg w \wedge x \wedge y \wedge \neg z\ ) \end{array}$$

b Annotated Karnaugh map



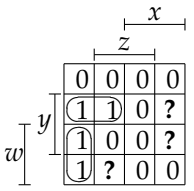
and associated, optimised implementation:

$$r = (\ x\ )$$

S291. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg w \wedge \neg x \wedge y \wedge \neg z\ ) \vee \\ & & (\ \neg w \wedge \neg x \wedge y \wedge z\ ) \vee \\ & & (\ w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\ & & (\ w \wedge \neg x \wedge y \wedge \neg z\ ) \end{array}$$

b Annotated Karnaugh map



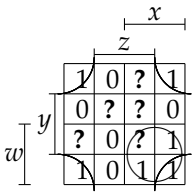
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \ w \wedge \neg x \qquad \wedge \neg z\ ) \vee \\ & & (\ \neg w \wedge \neg x \wedge y \qquad\ \ )\end{array}$$

S292. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\ & & (\ \neg w \wedge\ \ x \wedge \neg y \wedge \neg z\ ) \vee \\ & & (\ \ w \wedge \neg x \wedge \neg y \wedge \neg z\ ) \vee \\ & & (\ \ w \wedge\ \ x \wedge \neg y \wedge \neg z\ ) \vee \\ & & (\ \ w \wedge\ \ x \wedge \neg y \wedge\ \ z\ ) \vee \\ & & (\ \ w \wedge\ \ x \wedge\ \ y \wedge \neg z\ )\end{array}$$

b Annotated Karnaugh map



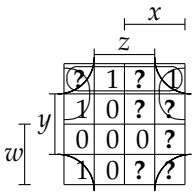
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \ w \wedge\ \ x \qquad\qquad\ \ ) \vee \\ & & (\qquad\qquad \neg y \wedge \neg z\ )\end{array}$$

S293. a Reference implementation:

$$\begin{array}{rcl} r & = & (\ \neg w \wedge \neg x \wedge \neg y \wedge\ \ z\ ) \vee \\ & & (\ \neg w \wedge \neg x \wedge\ \ y \wedge \neg z\ ) \vee \\ & & (\ \neg w \wedge\ \ x \wedge \neg y \wedge \neg z\ ) \vee \\ & & (\ \ w \wedge \neg x \wedge \neg y \wedge \neg z\ )\end{array}$$

b Annotated Karnaugh map



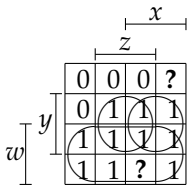
and associated, optimised implementation:

$$\begin{array}{rcl} r & = & (\ \neg w \qquad\qquad \wedge \neg z\ ) \vee \\ & & (\ \neg w \qquad \wedge \neg y \qquad\ \ ) \vee \\ & & (\qquad\qquad \neg y \wedge \neg z\ )\end{array}$$

S294. a Reference implementation:

$$\begin{array}{rcccccccccc}
r & = & (& \neg w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& \neg w & \wedge & x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & \neg y & \wedge & z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & \neg x & \wedge & y & \wedge & z &) & \vee \\
 & & (& w & \wedge & x & \wedge & \neg y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & \neg z &) & \vee \\
 & & (& w & \wedge & x & \wedge & y & \wedge & z &) &
\end{array}$$

b Annotated Karnaugh map



and associated, optimised implementation:

$$\begin{array}{rcccccccc}
r & = & (& & x & \wedge & y & &) & \vee \\
 & & (& & & & y & \wedge \; z &) & \vee \\
 & & (& w & & & & &) &
\end{array}$$

3 Chapter 3

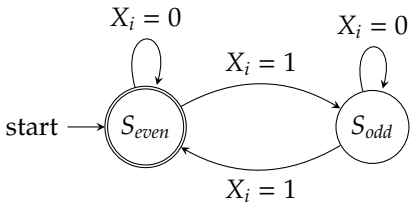
S295. A generic, block diagram style framework for FSMs is as follows:

st.

- An n-bit register (middle component) holds Q, the current state of the FSM.
- Within a given clock period, the current state is provided as input to δ, the transition function: based on Q and any input, this computes the next state Q'.
- At the same time that δ is computing the next state, the output function ω computes any output from the FSM; depending on the type of FSM, this might be based on Q only, or on Q and any input.
- A positive edge of the clock signal causes the state to be updated with the output from δ. That is, the FSM advances from the current to next state; computation by δ and ω is performed in the same way during the subsequent clock period, once Q has been updated with Q'.

Note that this framework is assumed in *any* of the following questions that ask for it.

This FSM can be in one of two states: either the bits of X processed so far have an even or odd number of elements equal to 1; we give each of the states a label, so in this case S_{even} and S_{odd} for example. Next we can describe how the FSM can transitions from some current state to a next state, i.e., how the transition function δ works: based on an input X_i provided at each step, we might draw



or equivalently say

	δ	
Q	Q'	
	$X_i = 0$	$X_i = 1$
S_{even}	S_{even}	S_{odd}
S_{odd}	S_{odd}	S_{even}

where Q is the current state and Q' is the next state.

Given the FSM has two states only, we can store the current state using a 1-bit register. Based on a natural mapping of the abstract to concrete state labels (i.e., $S_{even} \mapsto 0$ and $S_{odd} \mapsto 1$), we can rewrite the transition function as a truth table:

X_i	Q	Q'
0	0	0
0	1	1
1	0	1
1	1	0

and see clearly that $Q' = Q \oplus X_i$. Inspecting Q directly provides the output: if $Q = 0$ we have (so far) even parity, and in contrast if $Q = 1$ we have odd parity. So in a sense the output function ω is simply the identity function. In short, the low-level detail filled into the high-level design is very simple (in this case at least) once the question has been digested.

One obvious addition would be some form of mechanism to reset the FSM: as stated above, we assume it starts in the state S_{even} when powered-on but clearly this may not be true (the content in Q will essentially be random initially).

S296. a There are several approaches to solving this problem. Possibly the easiest, but perhaps not the most obvious, is to simply build a shift-register: the register stores the last three inputs, when a new input is available the register shifts the content along by one which means the oldest input drops off one end and the new input is inserted into the other end. One can then build a simple circuit to test the current state of the shift-register to see if the last three inputs match what is required..

Alternatively, one can take a more heavy-weight approach and formulate the solution as a state machine. First we need to decide on an encoding for our state; when searching though the input we can have matched zero through three correct tokens we denote this by the integer S stored in two bits using Q_1 and Q_0 as the most-significant and least-significant respectively. We also need an encoding of the actual input tokens I which are being passed to the matching circuit. Arbitrarily we might select $A = 0$, $C = 1$, $G = 2$ and $T = 3$ although other encodings are valid and might actually simplify things; we use I_1 and I_0 to denote the most and least-significant bits of the input token I. From this we can now create a table describing the mapping between current state S and input I to next state S' which can be roughly written as

I_1	I_0	Q_1	Q_0	Q'_1	Q'_0
0	0	0	0	0	1
0	1	0	0	0	0
1	0	0	0	0	0
1	1	0	0	0	0
0	0	0	1	0	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	0	1	0	0
0	0	1	0	0	1
0	1	1	0	0	0
1	0	1	0	0	0
1	1	1	0	1	1
0	0	1	1	0	1
0	1	1	1	0	0
1	0	1	1	0	0
1	1	1	1	0	0

Thus, if we are in state $(Q_1, Q_0) = (0, 0) = 0$ and see an A input, we move to state $(Q'_1, Q'_0) = (0, 1) = 1$ otherwise we stay in state $(Q'_1, Q'_0) = (0, 0) = 0$. Now we can define the transition function from current state to next state as

$$\begin{aligned} Q_0 &= (\neg Q_1 \wedge \neg Q_0 \wedge \neg I_1 \wedge \neg I_0) \vee \\ &\quad (\neg Q_1 \wedge Q_0 \wedge \neg I_1 \wedge \neg I_0) \vee \\ &\quad (Q_1 \wedge \neg Q_0 \wedge \neg I_1 \wedge \neg I_0) \vee \\ &\quad (Q_1 \wedge Q_0 \wedge \neg I_1 \wedge \neg I_0) \vee \\ &\quad (Q_1 \wedge \neg Q_0 \wedge I_1 \wedge I_0) \\ Q_1 &= (\neg Q_1 \wedge Q_0 \wedge \neg I_1 \wedge I_0) \vee \\ &\quad (Q_1 \wedge \neg Q_0 \wedge I_1 \wedge I_0) \end{aligned}$$

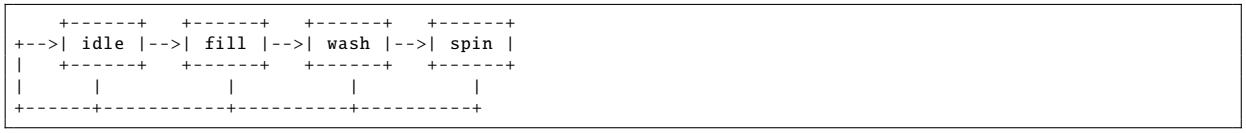
with simplifications as appropriate. Finally, the output flag F will be set only according to

$$F = Q_1 \wedge Q_0$$

to signal when we have matched three characters. As such, we can realise the FSM framework described in S295. by filling each component with the associated implementation above.

b Making a general-purpose matching circuit will probably use less logic than having three separate circuits; this will reduce the space required. As an extension one might consider implementing the transition and output functions as a look-up table instead of hard-wiring them; this will mean the circuit could be used to match any sequence providing the tables were correctly initialised. Introducing a more complex circuit design could have the disadvantage of increasing the critical path (the longest sequential path though the entire circuit). If the critical path is longer, the design will have to be clocked slower and hence will not perform the matching function as quickly.

S297. a A basic diagram should show the four states and transitions between them which relate the movement from one to the other as a result of the washing cycle, and movement as a result of input from the buttons; for example a (very) basic diagram would be:

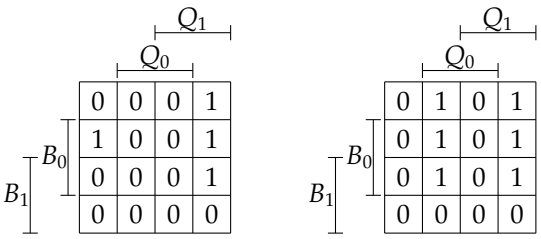


b Since there are four states, we can encode them using one two bits; we assign the following encoding *idle* = 00, *fill* = 01, *wash* = 10 and *spin* = 11. We use Q_1 and Q_0 to represent the current state, and Q'_1 and Q'_0 to represent the next state; B_1 and B_0 are the input buttons. Using this notation, we can construct the following state transition table which encodes the state machine diagram:

B_1	B_0	Q_1	Q_0	Q'_1	Q'_0
0	0	0	0	0	0
0	1	0	0	0	1
1	0	0	0	0	0
1	1	0	0	0	0
0	0	0	1	1	0
0	1	0	1	1	0
1	0	0	1	0	0
1	1	0	1	1	0
0	0	1	0	1	1
0	1	1	0	1	1
1	0	1	0	0	0
1	1	1	0	1	1
0	0	1	1	0	0
0	1	1	1	0	0
1	0	1	1	0	0
1	1	1	1	0	0

so that if, for example, the machine is in the *wash* state (i.e., $Q_1 = 1$ and $Q_0 = 0$) and no buttons are pressed then the next state is *spin* (i.e., $Q'_1 = 1$ and $Q'_0 = 1$); however if button B_1 is pressed to cancel the cycle, the next state is *idle* (i.e., $Q'_1 = 0$ and $Q'_0 = 0$).

c From the state transition table, we can easily extract the two Karnaugh maps:



Basic expressions can be extracted from the tables as follows:

$$\begin{aligned} Q'_0 &= (Q_1 \wedge \neg Q_0 \wedge B_0) \vee (Q_1 \wedge \neg Q_0 \wedge \neg B_1) \vee (\neg Q_0 \wedge B_0 \wedge \neg B_1) \\ Q'_1 &= (\neg Q_1 \wedge Q_0 \wedge B_0) \vee (\neg Q_1 \wedge Q_0 \wedge \neg B_1) \vee (Q_1 \wedge \neg Q_0 \wedge B_0) \vee (Q_1 \wedge \neg Q_0 \wedge \neg B_1) \end{aligned}$$

and through sharing, i.e., by computing

$$\begin{aligned} t_0 &= \neg B_1 \\ t_1 &= \neg B_0 \\ t_2 &= \neg Q_1 \\ t_3 &= \neg Q_0 \\ t_4 &= t_2 \wedge Q_0 \\ t_5 &= t_3 \wedge Q_1 \end{aligned}$$

we can simplify these to

$$\begin{aligned} Q'_0 &= (t_5 \wedge B_0) \vee (t_5 \wedge t_0) \vee (t_3 \wedge B_0 \wedge t_0) \\ Q'_1 &= (t_4 \wedge B_0) \vee (t_4 \wedge t_0) \vee (t_5 \wedge B_0) \vee (t_5 \wedge t_0) \end{aligned}$$

S298. a The two properties are defined as follows:

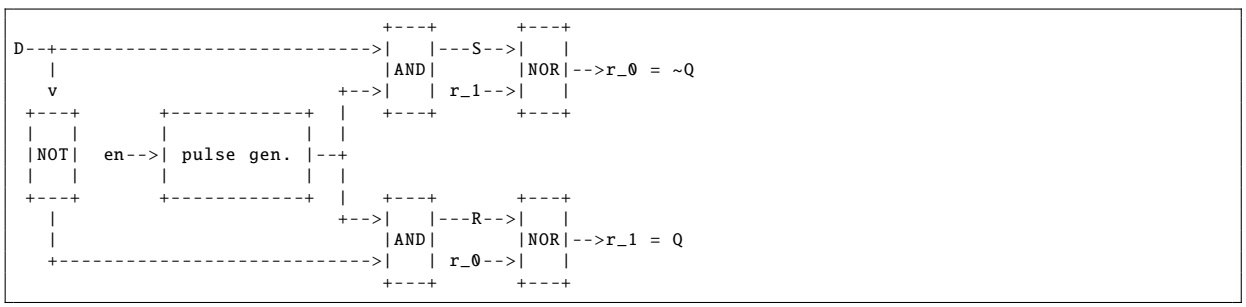
i The Hamming weight of X is the number of bits in X that are equal to 1, i.e., the number of times $X_i = 1$. This can be computed as

$$\mathcal{H}(X) = \sum_{i=0}^{n-1} X_i.$$

ii The Hamming distance between X and Y is the number of bits in X that differ from the corresponding bit in Y, i.e., the number of times $X_i \neq Y_i$:

$$\mathcal{D}(X, Y) = \sum_{i=0}^{n-1} X_i \oplus Y_i.$$

b There are two main approaches to constructing a flip-flop of this type; since both start with an SR-latch, the difference is mainly in how the edge-triggered behaviour is realised. Use of a master-slave arrangement is probably the more complete answer, but a simpler alternative would be to use a pulse generator. The overall design can be described roughly as follows:



There are basically four features to note:

i An SR-latch has two inputs S and R, and two outputs Q and $\neg Q$. When

- $S = 0$, $R = 0$ the component retains Q,
- $S = 1$, $R = 0$ the component updates to $Q = 1$,

- $S = 0, R = 1$ the component updates to $Q = 0$,
- $S = 1, R = 1$ the component is meta-stable.

The component is level-triggered in the sense that Q is updated within the period of time when $S = 1$ or $R = 1$ (rather than when they transition to said values).

ii To provide more fine-grained control over the component, the two inputs are typically gated using (i.e., AND'ed with) an enable signal *en*: when $en = 0$, the latch inputs are always zero and hence it retains the same state, when $en = 1$ it can be updated as normal.

iii In order to change from the current level-triggered behaviour into an edge-triggered alternative, one approach is to use a pulse generator. The idea here is to intentionally create a mismatch in propagation delay into the inputs of an AND gate: each time *en* changes, the result is that we see a small pulse on the output of the AND gate. Provided this is small enough, one can argue it acts like an edge rather than a level.

iv Finally, the gated S and R inputs are tied together and controlled by one input D meaning $S = D$ and $R = \neg D$. This prevents the component being used erroneously: it can *only* retain or update the state.

c The power consumed by CMOS transistors can be decomposed into two parts: the static part (which relates to leakage) and the dynamic part (which relates to power consumed when the transistor switches). In short, a value switching (i.e., changing from one value to another) consumes much more power than staying the same. In this case, clearly we have an advantage in the all but one of the n bits in the register will stay the same; hence in terms of power consumption, storing elements of the the Gray code (versus some other sequence for example) is an advantage.

d See S295..

e As an aside, a potentially neat approach here is to use a Johnson counter. This is basically an n-bit register (initialised to zero) whose content is shifted by one place on each clock edge. The new incoming, 0-th bit is computed as the NOT of the outgoing, $(n-1)$-th bit and every other bit is shifted up by one place (e.g., each i-th bit for $1 \leq i < n-1$ becomes the $(i+1)$-th bit). For $n = 3$, this produces the sequence

$$\begin{array}{c} \langle 0,0,0 \rangle \\ \langle 1,0,0 \rangle \\ \langle 1,1,0 \rangle \\ \langle 1,1,1 \rangle \\ \langle 0,1,1 \rangle \\ \langle 0,0,1 \rangle \\ \vdots \end{array}$$

which satisfies the Hamming distances property, but does not include all possible values: for example, $\langle 1,0,1 \rangle$ is not included. So this does not really answer the question in the sense that we require a component that cycles through the full 2^n-element sequence, an example of which is

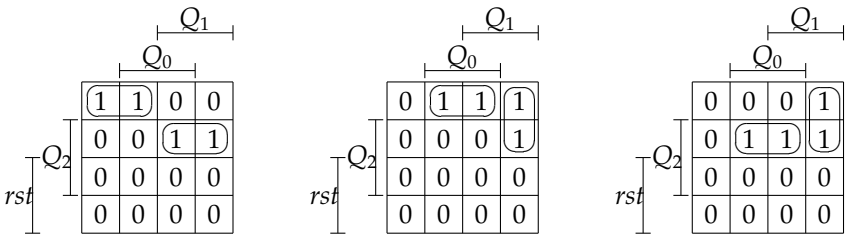
$$\begin{array}{c} \langle 0,0,0 \rangle \\ \langle 1,0,0 \rangle \\ \langle 1,1,0 \rangle \\ \langle 0,1,0 \rangle \\ \langle 0,1,1 \rangle \\ \langle 1,1,1 \rangle \\ \langle 1,0,1 \rangle \\ \langle 0,0,1 \rangle \end{array}$$

As a result, we can use an FSM-based approach based on the framework in the question above. For $n = 3$ there are $2^3 = 8$ elements in the Gray code, and so a 3-bit state $Q = \langle Q_0, Q_1, Q_2 \rangle$ is enough to store the current element. The output function ω is basically free: we simply provide the current state Q as output, which is also the current element in the Gray code sequence. Based on the inputs Q and *rst*, the state

transition function δ can be described as follows:

rst	Q_2	Q_1	Q_0	Q'_2	Q'_1	Q'_0
0	0	0	0	0	0	1
0	0	0	1	0	1	1
0	0	1	1	0	1	0
0	0	1	0	1	1	0
0	1	1	0	1	1	1
0	1	1	1	1	0	1
0	1	0	1	1	0	0
0	1	0	0	0	0	0
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	1	0	0	0
1	0	1	0	0	0	0
1	1	1	0	0	0	0
1	1	1	1	0	0	0
1	1	0	1	0	0	0
1	1	0	0	0	0	0

From this truth table we can (more easily that usual perhaps) extract Karnaugh maps for each bit of the next state Q'



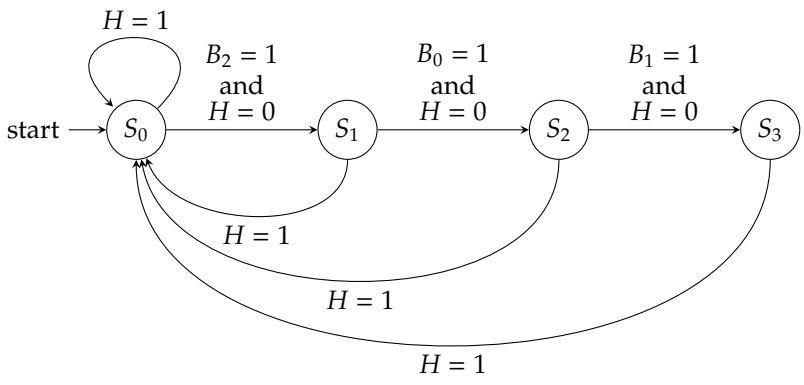
and hence Boolean expressions

$$\begin{aligned}
Q'_0 &= (\ \neg rst \wedge \neg Q_2 \wedge \neg Q_1\ ) \vee \\
&\quad (\ \neg rst \wedge Q_2 \wedge Q_1\ ) \\
Q'_1 &= (\ \neg rst \wedge \neg Q_2 \wedge Q_0\ ) \vee \\
&\quad (\ \neg rst \wedge Q_1 \wedge \neg Q_0\ ) \\
Q'_2 &= (\ \neg rst \wedge Q_2 \wedge Q_0\ ) \vee \\
&\quad (\ \neg rst \wedge Q_1 \wedge \neg Q_0\ )
\end{aligned}$$

Placing the associated combinatorial logic and a 3-bit, D-type flip-flop based register to store Q into the generic framework, we end up with a component that cycles through our 3-bit Gray code sequence under control of a clock signal.

S299. a See S295..

b There are a few different ways to interpret some parts of the problem definition, but one reasonable approach is as follows:



Essentially, the idea is that by pressing buttons we advance from the stating state S_0 toward the final state S_3 (as long as the handle is not turned, which means we go back to the start): when in S_3 the door is unlocked, otherwise it remains locked. In particular, if the buttons are pressed in the wrong order we get "stuck" half way along the sequence and never reach S_3. For example if B_1 is pressed while in state S_1, the FSM does not (and cannot ever) transition into S_2 since the button stays pressed: the only way to "unstick" the FSM is to turn the handle, reset the mechanism and start again.

There are four states in total; since $2^2 = 4$ we can represent the current state Q as a 2-bit integer, making the concrete assignment

$$\begin{array}{lcl} S_0 & \mapsto & \langle 0, 0 \rangle \\ S_1 & \mapsto & \langle 1, 0 \rangle \\ S_2 & \mapsto & \langle 0, 1 \rangle \\ S_3 & \mapsto & \langle 1, 1 \rangle \end{array}$$

The FSM diagram can be expressed as a truth table, particular to this P, which captures the various transitions:

H	B_2	B_1	B_0	Q_1	Q_0	Q'_1	Q'_0
0	0	?	?	0	0	0	0
0	1	?	?	0	0	0	1
1	?	?	?	0	0	0	0
0	?	?	0	0	1	0	1
0	?	?	1	0	1	1	0
1	?	?	?	0	1	0	0
0	?	0	?	1	0	1	0
0	?	1	?	1	0	1	1
1	?	?	?	1	0	0	0
0	?	?	?	1	1	1	1
1	?	?	?	1	1	0	0

Implementing this truth table via a 6-input Karnaugh map is a little more tricky than with fewer inputs; instead, we simply derive the expressions by inspection (i.e., by forming a term for each 1 entry in a given output) to yield

$$\begin{array}{lcl}
Q'_0 & = & (\ \neg H \wedge B_2 \wedge \neg Q_1 \wedge \neg Q_0\ ) \vee \\
 & & (\ \neg H \wedge \neg B_0 \wedge \neg Q_1 \wedge Q_0\ ) \vee \\
 & & (\ \neg H \wedge B_1 \wedge Q_1 \wedge \neg Q_0\ ) \vee \\
 & & (\ \neg H \wedge Q_1 \wedge Q_0\ ) \\
 & & \\
Q'_1 & = & (\ \neg H \wedge B_0 \wedge \neg Q_1 \wedge Q_0\ ) \vee \\
 & & (\ \neg H \wedge \neg B_1 \wedge Q_1 \wedge \neg Q_0\ ) \vee \\
 & & (\ \neg H \wedge B_1 \wedge Q_1 \wedge \neg Q_0\ ) \vee \\
 & & (\ \neg H \wedge Q_1 \wedge Q_0\ )
\end{array}$$

with minor optimisation possible thereafter. Returning to the framework, the idea is then that we

i instantiate the middle box with a 2-bit register, using D-type flip-flops for example, to store Q,

ii instantiate the top box to implement δ using the equations above,

iii instantiate the bottom box to implement ω using the equation

$$L = \neg(Q_1 \wedge Q_0)$$

so the door is locked unless the FSM is in state S_3.

c The purpose of a clock signal is to control the FSM, advancing it through steps (i.e., transitions) with all components synchronised. However, the only updates of state occur on positive transistors of B_i or H. That is, the FSM only chances state when one of the buttons is pressed, or the handle turned: in each case, this means the associated value transitions from 0 to 1. As a result, one *could* argue the expression

$$H \vee B_0 \vee B_1 \vee B_1 \vee B_2$$

can be used to advance the FSM (i.e., latch the next state produced by the transition function), rather than "polling" the buttons and handle at each clock edge to see if their value has changed.

d Among various valid answers, the following are clear:

i The content stored in an SRAM memory is lost if the power supply is removed: such devices depend on a power supply so transistors used to maintain the stored content can operate. In the context of the proposed approach, this means if a power cut occurs, for example, then the password will be "forgotten" by the lock.

ii When the power supply comes back online the password might be essentially random due to the way SRAMs work. If this is not true however, and the SRAM is initialised into a predictable value (e.g., all zero), this could offer an attractive way to bypass the security offered!

iii Given physical access to the lock, one might simply read the password out of the SRAM. With an FSM hard-wired to a single password, the analogue is arguably harder: one would need to (invasively) reverse engineer the gate layout and connectivity, then the FSM design.

Less attractive answers include degradation of performance (e.g., as a result of SRAM access latency) or increase in cost: given constraints of the application, neither seems particular important. For example the access latency of SRAM memory is measured in small fractions of a second; although arguably true in general, from the perspective of a human user of the door lock the delay will be imperceptible.

e This is quite open-ended, but one reasonable approach would be as follows:

i This is a slightly loaded question in that it implies some alteration *is* needed; as such, marks might typically be given for identifying the underlying reason, and explaining each aspect of the proposed alteration.

The crucial point to realise is testing implementations of δ and ω, for example, depends on being able to set (and possibly inspect) the state Q which acts as input to both. An example technology to allow this would be JTAG, which requires an additional interface (inc. TDI, TDO, TCLK, TMS and TRST pins) and also injection of a scan chain to access all flip-flops. This allows the test process to scan a value into Q one bit at a time, run the system normally, then scan out Q to test it.

ii The idea would be to place each system under the control of a test stimulus that automates a series of tests: the test stimulus has access to all inputs (i.e., the JTAG interface, each button and the handle) and outputs (e.g., the JTAG interface, and the lock mechanism), and is tasked with making sure the overall behaviour matches some reference.

In this context, the number of states, inputs and outputs is small enough that a brute force approach is reasonable; this is also motivated by the fact there are no obvious boundary cases and so on. The strategy would therefore be: for each entry in the truth table

- put the device in test mode,
- scan-in the state Q and drive each B_i with the associated values,
- put the device in normal mode, and force an update of the FSM using the clock signal,
- put the device in test mode,
- check the value of L matches that expected,
- scan-out and check the value of Q matches that expected.

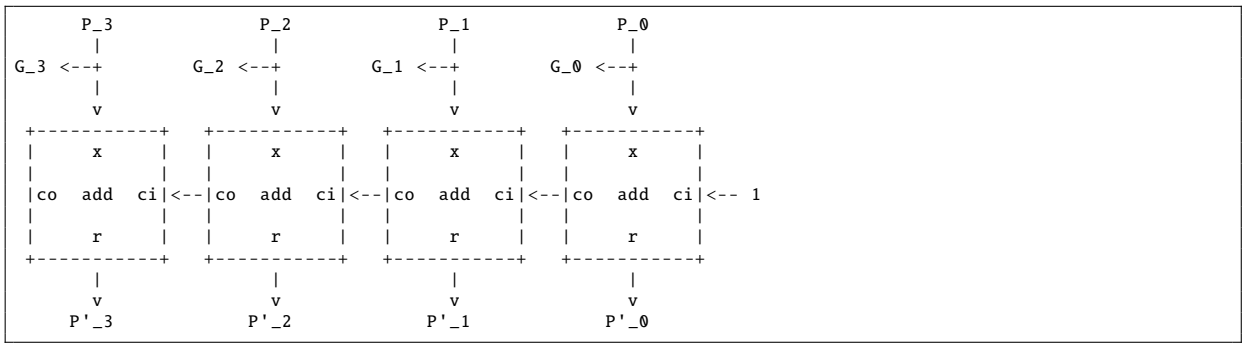
An alternative answer might focus on some form of BIST, but in essence this just places all the above *inside* the system rather than viewing it as something done externally.

S300. a At least three advantages (or disadvantages, depending on which way around you view the options) are evident:

- With option one, extracting each digit of the current PIN to form a guess is trivial; with option two this is much harder, in that we need to take the integer P and decompose it into a decimal representation (through repeated division and modular reduction).
- With option one, incrementing the current PIN is harder (since the addition is in decimal); with option two this is much easier, in that we can simply use a standard integer adder.
- With option one, the total storage requirement is $4 \cdot 4 = 16$ bits; with option two this is only 14 bits, since $2^{14} = 16384 > 9999$.

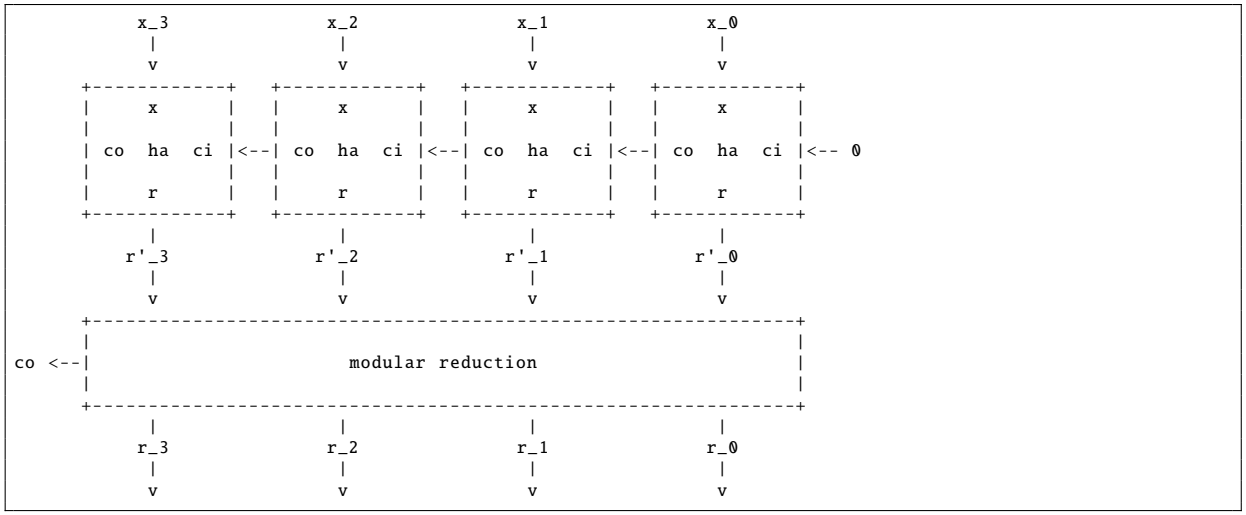
Based on this, and reading ahead to the next question, the decimal representation seems more attractive: designing a decimal adder is significantly easier than a binary divider.

b Given the choice, and although both options are viable, we focus on a design for the second, decimal representation: this is simpler by some way, so the expected answer. At a high-level, the component can be described as follows:



$P_i = G_i$ so production of the guess is trivial; the other output is a little harder. The basic idea is to use something similar to a ripple-carry adder. Each i-th cell takes a decimal digit P_i and a carry-in from the previous, $(i-1)$-th cell; it produces a decimal digit P'_i and a carry-out into the next $(i+1)$-th cell. The difference from a binary ripple-carry adder then is that it only accepts one digit rather than two as input (since it increments P rather than computes a general-purpose addition), plus it obviously works with decimal rather than binary digits.

There are various ways to approach the design of each decimal adder cell, but perhaps the most straightforward uses two stages:

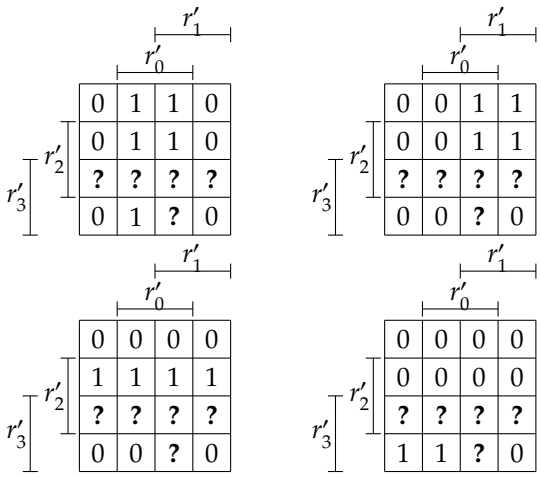


The first stage computes an integer sum $r' = x + ci$. Although this could be realised using a standard ripple-carry adder, we can make a more problem-specific improvement: a ripple-carry adder normally uses full-adder cells that compute $x + y + ci$, but we lack the second input y. Thus we can use half-adder cells instead, which use half the number of gates; we assume such a half-adder is available as a standard component. The second stage takes $r' = x + ci$ as input, and produces the outputs r and co, implementing the modular reduction. The range of each input means $0 \leq r' < 11$, or equivalently that cases where

$r' > 10$ are impossible. We can describe the behaviour of the stage using the following truth table:

r'_3	r'_3	r'_3	r'_3	co	r_3	r_2	r_1	r_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	?	?	?	?	?
1	1	0	0	?	?	?	?	?
1	1	0	1	?	?	?	?	?
1	1	1	0	?	?	?	?	?
1	1	1	1	?	?	?	?	?

As such, we can produce a set of Karnaugh maps



which translate fairly easily into Boolean expressions

$$\begin{aligned} r_0 &= r'_0 \\ r_1 &= \neg r'_3 \wedge r'_1 \\ r_2 &= \neg r'_3 \wedge r'_2 \\ r_3 &= r'_3 \wedge \neg r'_2 \wedge \neg r'_1 \end{aligned}$$

that allow implementation.

c The FSM maintains a current state Q. Given there are five states, we can represent the current state as

$$Q = \langle Q_0, Q_1, Q_2 \rangle$$

i.e., three bits (since $2^3 = 8 > 5$), so the device could store it in a register comprised of three D-type flip-flops; doing so accepts there are three unused state representations.

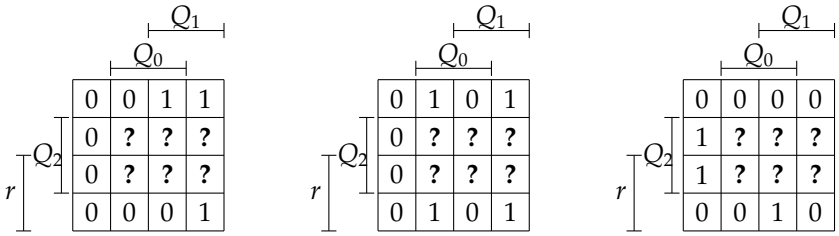
We can represent the states as follows

$$\begin{aligned} S_0 &= \langle 0, 0, 0 \rangle \\ S_1 &= \langle 1, 0, 0 \rangle \\ S_2 &= \langle 0, 1, 0 \rangle \\ S_3 &= \langle 1, 1, 0 \rangle \\ S_4 &= \langle 0, 0, 1 \rangle \end{aligned}$$

and therefore formulate a tabular transition function δ:

b	r	Q_2	Q_1	Q_0	Q'_2	Q'_1	Q'_0
0	?	0	0	0	0	0	0
1	?	0	0	0	0	0	1
?	?	0	0	1	0	1	0
?	?	0	1	0	0	1	1
?	0	0	1	1	0	0	1
?	1	0	1	1	1	0	0
?	?	1	0	0	1	0	0

Turning these into Karnaugh maps and then Boolean expressions is a little tricky due to the need for five inputs. To cope, we assume there are no transitions from S_0 and ignore b, then patch the equation for Q'_0 (the only bit of the next state influenced by moving out of S_0) appropriately. That is, we get the following



which then translate into

$$
\begin{array}{rcl}
Q'_0 & = & (\quad b \wedge \neg\, Q_2 \wedge \neg\, Q_1 \wedge \neg\, Q_0 \quad) \vee \\
 & & (\neg\, r \qquad \wedge \quad Q_1 \qquad\quad) \vee \\
 & & (\qquad\qquad\quad Q_1 \wedge \neg\, Q_0 \quad) \\
Q'_1 & = & (\qquad\qquad \neg\, Q_1 \wedge \quad Q_0 \quad) \vee \\
 & & (\qquad\qquad\quad Q_1 \wedge \neg\, Q_0 \quad) \\
Q'_2 & = & (\quad r \wedge \qquad\quad Q_1 \wedge \quad Q_0 \quad) \vee \\
 & & (\qquad\quad Q_2 \qquad\qquad\quad) \\
\end{array}
$$

4 Chapter 4

S301. a $O(n)$ implies the critical path is proportional to the number of bits (including some constant factor) required to represent each of the operands. The reason is the carry chain which runs through all n full-adders in the design: each i-th full-adder produces a carry-out used as a carry-into the $(i+1)$-th full-adder. This means each i-th bit of the result depends on, and cannot be computed before, all j-th bits for $0 \leq j < i$.

An alternative, carry look-ahead design separates computation of carries from the full-adder cells themselves; this allows an organisation whose critical path can be described as $O(\log n)$, although the number of logic gates required is less attractive.

b The relationship

$$x \text{ is a power-of-two} \equiv (x \wedge (x-1)) = 0$$

performs the test which can be written as the C expression

```
( x & ( x - 1 ) ) == 0
```

This works because if x is an exact power-of-two then $x-1$ sets all bits less-significant that the n-th to one; when this is AND'ed with x (which only has the n-th bit set to one) the result is zero. If x is not an exact power-of-two then there will be bits in x other than the n-th set to one; in this case $x-1$ only sets bits less-significant than the least-significant and hence there are others left over which, when AND'ed with x, result in a non-zero result. Note that the expression fails, i.e., it is non-zero, for $x = 0$ but this is allowed since the question says $x \neq 0$.

c `x` is of type `char`, so is therefore represented using two's-complement in 8 bits; values for such a representation range between $2^{n-1} - 1 = 2^{8-1} - 1 = 127$ and $-2^{n-1} = -2^{8-1} = -128$ inclusive. This means that by

i decrementing `x` we get the value before 127, which is 126, or

ii incrementing `x` we get the value after 127, which is −128: the reason for this is that the representation of 127 is $01111111_{(2)}$, but the next value $10000000_{(2)}$ is the largest negative value possible. That is, there has been an overflow with the result "wrapping around".

d The expression computes the comparison $0 < x$. This is because if $x < 0$ then $x_3 = 1$, and if $x = 0$ then $x_3 = x_2 = x_1 = x_0 = 0$. Therefore, $x > 0$ if both $x_3 = 0$ and one of $x_i \neq 0$ for $i \in \{2, 1, 0\}$. Strictly speaking, it tests whether $0 < x \leq 7$ but the upper bound is implied by the representation of x: it cannot take a value greater than 7 by definition.

e The initial temptation is to use six adder components to compute

$$r = 7 \cdot x = x + x + x + x + x + x + x$$

where the size of inputs and outputs increases as one progresses through the computation; a considered approach might utilise carry save adders to reduce the critical path associated with the multiple summands, but here we consider ripple-carry designs only.

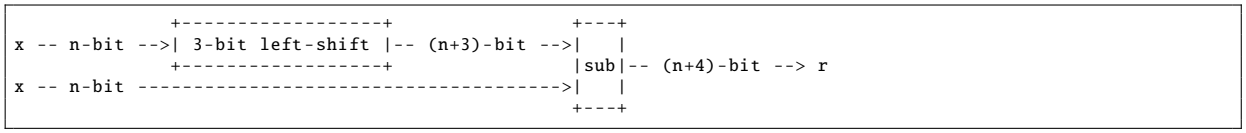
A more efficient alternative would use three adders to compute

$$r = 7 \cdot x = 4 \cdot x + 2 \cdot x + 1 \cdot x = 2^2 \cdot x + 2^1 \cdot x + 2^0 \cdot x$$

noting that the multiplications by powers-of-two are "free" since they can be achieved by simply relabelling bits rather than computation. This approach can be further refined to compute

$$r = 7 \cdot x = 8 \cdot x - 1 \cdot x = 2^3 \cdot x - 2^0 \cdot x$$

using just one adder (assuming addition and subtraction can be realised using the same component). Clearly this will produce the shortest critical path, and relates to the following diagram:



S302. a Clearly we can implement $\neq$ by negating the result of $=$ and likewise for $<$ and $\geq$, and $>$ and $\leq$. Furthermore, we can build $\geq$ from $>$ and $=$, and $\leq$ from $<$ and $=$. So essentially we only need two comparisons, say $=$ and $<$ to be able to compute the rest so long as we have the logic operations as well. The choice of which three is simply a matter of which ones you want to go faster: the ones built from a combination of other comparison and logic instructions will take longer to execute. One might take the approach of looking at C programs and selecting the set most used. For example $=$ and $<$ are used a lot to program typical loops; one might select them for this reason.

b You can be as fancy as you want with any optimisations or special cases, for example checking for multiplication by zero, one or a power-of-two might be a good idea. But basically, the easiest way to do this is as follows:

```
uint16_t mul( uint16_t x, uint16_t y ) {
  switch( x ) {
    case  0 : return 0;
    case  1 : return y;
    case  2 : return y << 1;
    case  4 : return y << 2;
    case  8 : return y << 3;
    case 16 : return y << 4;
  }
  switch( y ) {
    case  0 : return 0;
    case  1 : return x;
    case  2 : return x << 1;
    case  4 : return x << 2;
    case  8 : return x << 3;
    case 16 : return x << 4;
  }

  uint16_t t = 0;

  for( int i = 15; i >= 0; i-- ) {
    t = t << 1;

    if( ( y >> i ) & 1 ) {
      t = t + x;
    }
  }

  return t;
}
```

c A basic implementation might look like the following:

```
int H( uint16_t x ) {
  int t = 0;

  for( int i = 0; i < 16; i++ ) {
    if( ( x >> i ) & 1 ) {
      t = t + 1;
    }
  }

  return t;
}
```

but this has a number of drawbacks. First, the overhead of of operating the loop quite high in comparison to the content; for example the loop body needs only a few instructions, while it takes nearly as many again to test and increment `i` during each iteration. Second, the number of branches in the code means that pipelined processors might not execute them efficiently at all. An improvement is to use some form of divide-and-conquer approach where we split the problem into 2-bit then 4-bit chunks and so on. The result might look like:

```
int H( uint16_t x ) {
  x = ( x & 0x5555 ) + ( ( x >> 1 ) & 0x5555 );
  x = ( x & 0x3333 ) + ( ( x >> 2 ) & 0x3333 );
  x = ( x & 0x0F0F ) + ( ( x >> 4 ) & 0x0F0F );
  x = ( x & 0x00FF ) + ( ( x >> 8 ) & 0x00FF );

  return ( int )( x );
}
```

S303. First, note that the result via a naive method would be

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

However, we can write down three intermediate values using only three multiplications as

$$\begin{aligned} t_2 &= x_1 \cdot y_1 \\ t_1 &= (x_0 + x_1) \cdot (y_0 + y_1) \\ t_0 &= x_0 \cdot y_0. \end{aligned}$$

The original result can then be expressed in terms of these intermediate values via

$$\begin{aligned} r_2 &= t_2 &&= x_1 \cdot y_1 \\ r_1 &= t_1 - t_0 - t_2 &&= x_0 \cdot y_0 + x_0 \cdot y_1 + x_1 \cdot y_0 + x_1 \cdot y_1 - x_0 \cdot y_0 - x_1 \cdot y_1 \\ &&&= x_1 \cdot y_0 + x_0 \cdot y_1 \\ r_0 &= t_0 &&= x_0 \cdot y_0. \end{aligned}$$

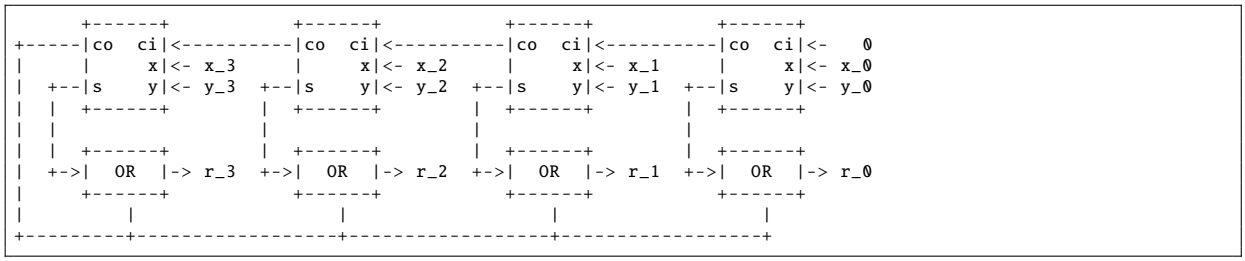
So roughly speaking, over all we use three $(n/2)$-bit multiplications and four $(n/2)$-bit additions.

S304. a In binary, the addition we are looking at is

$$\begin{aligned} 10_{(10)} &= 1010_{(2)} \\ 12_{(10)} &= \underline{1100_{(2)}} \quad + \\ &\quad 10110_{(2)} \end{aligned}$$

where $10110_{(2)} = 22_{(10)}$. In 4 bits this value is $0110_{(2)} = 6_{(10)}$ however, which is wrong.

b A design for the 4-bit clamped adder looks like this:

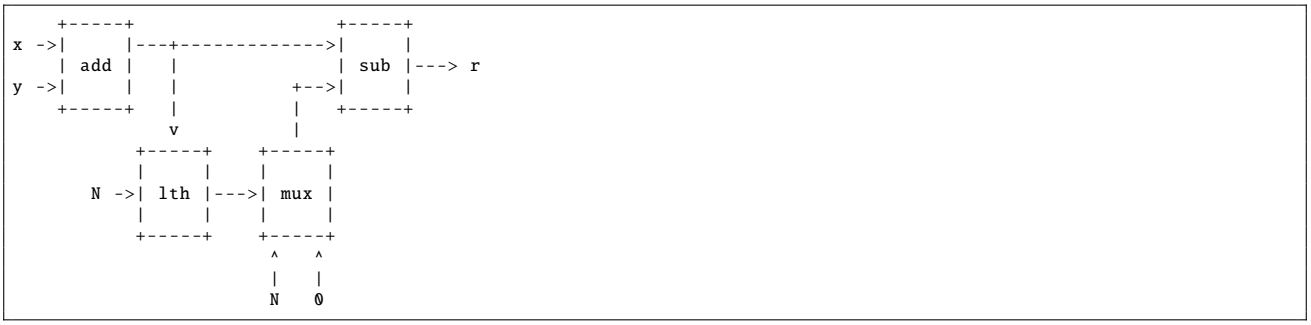


Essentially the idea is that if a carry-out occurs from the most-significant adder, this turns all the output bits to 1 via the additional OR gates. That is, if the carry-out occurs then we get $1111_{(2)} = 15_{(10)}$ as the result, i.e., the largest 4-bit result possible.

S305. Since we know nothing about N, there is no obvious short-cut to performing the modular reduction after the multiplication. Instead, the most simple way to approach the design is to recall that

$$x \cdot y = \underbrace{x + x + \cdots + x + x}_{y \text{ copies}}.$$

So to compute $x \cdot y \pmod N$, we just have to make sure that each of the additions is modulo N; then we can use whatever method we want. A circuit for modular addition is actually quite simple:



In short, we add x and y together, and then compare the result t with N: if t is smaller, we select 0 as the output from the multiplexer otherwise we select N. Then, we subtract the value we selected from t. The end result is that we get $x + y - 0 = x + y \pmod N$ if $x + y < N$, and $x + y - N = x + y \pmod N$ if $x + y \geq N$.

Recall that an 8-bit, bit-serial multiplier would compute the product $x \cdot y$ as follows:

Input: An 8-bit multiplicand x, and 8-bit multiplier y
Output: The product $x \cdot y$

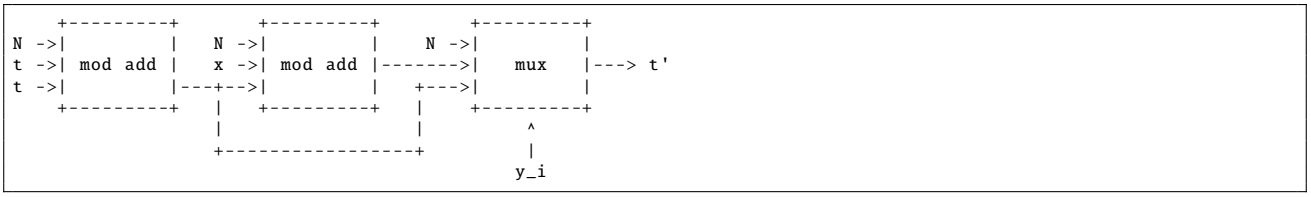
```
1 t ← 0
2 for i = 7 downto 0 do
3 |   t ← t + t
4 |   if y_i = 1 then
5 |   |   t ← t + x
6 |   end
7 end
8 return t
```

Armed with our modular adder circuit, we can rewrite this as

Input: An 8-bit modulus N, a multiplier $0 \leq x < N$ and multiplicand $0 \leq y < N$
Output: The product $x \cdot y \pmod N$

```
1 t ← 0
2 for i = 7 downto 0 do
3 |   t ← t + t (mod N)
4 |   if y_i = 1 then
5 |   |   t ← t + x (mod N)
6 |   end
7 end
8 return t
```

which then simply demand eight iterations, under control of a clock, over the circuit



Notice that we first perform the operation $t + t \pmod{N}$, then use a multiplexer to decide if we take $t + t \pmod{N}$ or $t + t + x \pmod{N}$ as the next value of t. So each iterated use of the circuit represents an iteration of the algorithm loop. Of course, one could construct a combinatorial multiplier using the same approach, i.e., replacing any standard adder circuits with modular alternatives.