Sample Solution for Summer 2014

Question 1

(a)

```
f = (A + B + C) . (A + B + C') . (A + B' + C') . (A' + B + C) . (A' + B + C')
```

(b)

Each term in the CSOP form corresponds to a row in the truth table for g in which the output is 1. Similarly, each term in the CPOS form corresponds to a row with output 0. If there are fewer terms in the CPOS form, then there are more 1s than 0s in the truth table for g.

(c)

Minterms contain each input variable only once, either inverted or not inverted. Simplified products, as they come from simplified representations of the CSOP form, do not contain all input variables – some are missing because they have been factored out as redundant terms.

Question 2

(a)

These functions can have any number of arguments. However, a majority decoder with 1 argument is equivalent to the identity function. Additionally, majority decoders with even numbers of arguments have cases where neither 1

nor 0 is in the majority among argument values. So behaviour is undefined for some argument value combinations.

(b)

Neither form should be preferred, as they will always be the same length.

Majority decoders with an odd number of arguments will output 0 in half of the cases and 1 in the other half of the cases. With an even number of arguments, where n is the number of arguments, `m = ((2^n) - 1)/3` cases will have majority 0s, the same number will have majority 1s, and `m + 1` cases will have an even number of 1s and 0s.

(c)

Yes, the simplified forms will always have a shorter representation, as long as the number of arguments is greater than 2.

Take the Karnaugh map for a majority decoder with more than 2 arguments. The case where all arguments have the value 1 will have (at least) one neighbour where the arguments are all 1 except for one of them, as all neighbours in a Karnaugh map will have 1 more or 1 fewer 1s among their arguments than each other.

So there'll be one group of 2 that can be made. Even if all other 1s have to be covered singly, there'll be 1 fewer term in the result than in the canonical representation. A similar argument can be made for the SPOS form.

Then, for 2 arguments, the simplified form will have the same number of terms, but will use one fewer gates.

Question 3

(a)

Here's the truth table for the function:

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

Here's the Karnaugh map:

AB \ CD	00	01	11	10
00	0	0	1	1
01	0	1	1	0
11	0	1	0	0
10	0	0	1	0

The SSOP can be formed by making the following groupings (as it's the sum of

products, we are aiming to cover the 1s):

1. Top-right and its left neighbour (inputs 0010 and 0011)
2. First row, third column and its upper neighbour (inputs 0011 and 1011)
3. Second row, third column and its left neighbour (inputs 0111 and 0101)
4. Second row, second column, and its lower neighbour (inputs 0101 and 1101)

I've gone with four groups of two rather than two of two and two of one, because larger groups give simpler expressions.

This gives the following terms:

1. A'B'C
2. B'CD
3. A'BD
4. BC'D

So the SSOP is:

```
A'B'C + B'CD + A'BD + BC'D
```

(b)

We now have a slightly changed Karnaugh map with don't-care entries (marked by Ns):

AB \ CD	**00**	**01**	**11**	**10**
00	0	0	1	1
01	0	1	1	0
11	N	N	N	N
10	0	0	N	N

We're still covering 1s, but now we can cover don't-care entries as well. This gives

us:

1. Top-right and its left neighbour, and their upper neighbours (inputs 0010, 0011, 1010 and 1011)
2. Second row, third column and its left neighbour, and their lower neighbours (inputs 0111, 0101, 1111 and 1101)

This gives these terms:

1. B'C
2. BD

So the SSOP is now:

```
B'C + BD
```

(c)

Convert all ANDs to ORs with DeMorgan's:

- `B'C -> (B + C')'`
- `BD -> (B' + D')'`

This gives:

```
(B + C')' + (B' + D')'
```

To draw the circuit diagram:

- use a NOR for each NOR operation
- connect an input into both inputs of a NOR for each NOT operation
- use a NOR followed by a NOT for the OR operations

My diagram has 7 NORs.

(d)

Convert all ORs to ANDs with DeMorgan's:

- `B'C + BD -> ((B'C)' . (BD)')'`

So the answer is:

```
((B'C)' . (BD)')'
```

To draw the circuit diagram:

- use a NAND for each NAND operation
- connect an input into both inputs of a NAND for each NOT operation
- use a NAND followed by a NOT for the AND operations

My diagram has 4 NANDS.

(e)

AB \ CD	**00**	**01**	**11**	**10**
00	0	0	1	1
01	0	1	1	0
11	N	N	N	N
10	0	0	N	N

To cover the 0s we can make the following groups:

1. Top-left and its right neighbour, and their upper neighbours (inputs 0000, 0001, 1001, and 1000)
2. Second row, first column and its left neighbour, and their lower neighbours (inputs 0100, 0110, 1100, and 1110)

This gives these terms (remember primes are the other way around for products

of sums – I forgot):

1. C + B
2. D + B'

Which gives a SPOS of:

```
(C + B) . (D + B')
```

Question 4

(a)

z	y
0	0
1	1
2	1
3	0
4	1
5	0
6	0
7	1

(b)

z \ x	0	1
0	0	1
1	2	3
2	4	5
3	6	7
4	0	1
5	2	3
6	4	5
7	6	7

(c)

One sequence is:

- `000111000111`…

You can see that as you step to the right, the number of 1s increases or decreases by 1, producing alternating output.

Another sequence is:

- `010101010101`…

The first set of 3 has one 1, and then the next has 2, and the next has 1, and so on, producing alternating output.

I think these are the only two, but I'm not certain. I found them by thinking about which sequences will have different numbers of 1s in successive steps, and the fact that you can only change it by 1 each time.

(d)

A circuit realisation of this machine would contain 3 flip-flops, as there are 8 states (alternatively, as each of the last three steps needs to be recorded).