# Digital kredsløb - Exercises

*Af Jesper Bertelsen.*

Indholdsfortegnelse

[Exercise 3 - Sandsynlighed for at være korrekt. 90 - 100 % 3](#_Toc122444597)

[1. Simplify the following Boolean functions, using four-variable maps: 3](#_Toc122444598)

[1. 3](#_Toc122444599)

[2. F(w,x,y,z) = ∑(1,4,5,6,12,14,15) 3](#_Toc122444600)

[3. F(w,x,y,z) = ∑(2,3,6,7,12,13,14) 3](#_Toc122444601)

[4. F(w,x,y,z) = ∑(1,3,4,5,6,7,9,11,13,15) 4](#_Toc122444602)

[2. Simplify the following Boolean expressions, using four-variable maps: 4](#_Toc122444603)

[1. A′B′C′D′+AC′D′+B′CD′+A′BCD+BC′D 4](#_Toc122444604)

[2. A′B′C′D+AB′D+A′BC′+ABCD+AB′C 5](#_Toc122444605)

[3. Simplify the following Boolean function F, together with the don’t care conditions d, and then express the simplified function in sum-of-minterms form: 5](#_Toc122444606)

[1. F(A,B,C,D) = ∑(2,4,7,10,12 , d(A,B,C,D) = ∑(0,6,8,15) 5](#_Toc122444607)

[2. F(A,B,C,D) = ∑(0,6,8,13,14), d(A,B,C,D) = ∑(2,4,10) 6](#_Toc122444608)

[Exercise 4 - Sandsynlighed for at være korrekt. 90 - 100% 6](#_Toc122444609)

[1. Design a combinational circuit with three inputs and one output: 6](#_Toc122444610)

[1. The output is 1 when the binary value of the inputs is more than 2. The output is 0 otherwise. 6](#_Toc122444611)

[1. The output is 1 when the binary value of the inputs is not divisable by 3. 7](#_Toc122444612)

[2. The output is 1 when the binary value of the inputs is an even number. 8](#_Toc122444613)

[2. Design a combinational circuit that converts a four-bit Gray code to a four-bit binary number. Implement the circuit with exclusive-OR gates. 8](#_Toc122444614)

[3. Construct a 16 × 1 multiplexer with two 8 × 1 and one 2 × 1 multiplexers. Use block diagrams. 10](#_Toc122444615)

[4. Using a truth table and Karnaugh maps, design the BCD-to-seven-segment decoder using a minimum number of gates. The six invalid combinations should result in a blank display : 11](#_Toc122444616)

[5. Implement the following Boolean function with a 8 × 1 multiplexer. F (A, B, C, D) = ∑ (0, 2, 5, 8, 10, 14) 12](#_Toc122444617)

[6. Implement the following Boolean function with a 4 × 1 multiplexer and external gates. F (A, B, C, D) = ∑ (1, 3, 4, 11, 12, 13, 14, 15) 12](#_Toc122444618)

[Exercise 5 - Sandsynlighed for at være korrekt 90 - 100 % udover opgave 1 som er ca. 60 %. 13](#_Toc122444619)

[1. The D latch of Figure 1 is constructed with four NAND gates and an inverter. Consider the following three other ways for obtaining a D latch. In each case, draw the logic diagram and verify the circuit operation: 13](#_Toc122444620)

[1. Use NOR gates for the SR latch part and AND gates for the other two. An inverter may be needed. 13](#_Toc122444621)

[2. Use NOR gates for all four gates. Inverters may be needed. 15](#_Toc122444622)

[3. Use four NAND gates only (without an inverter). This can be done by connecting the output of the upper gate in Figure 1 (the gate that goes to the SR latch) to the input of the lower gate (instead of the inverter output) 16](#_Toc122444623)

[2. For the following state table 16](#_Toc122444624)

[1. Tabulate the reduced state table 16](#_Toc122444625)

[2. Draw the state diagram corresponding to the reduced state table 16](#_Toc122444626)

[16](#_Toc122444627)

[3. Deteremine the output sequence, starting from state a, and input sequence 01110010011. 17](#_Toc122444628)

[3. Design a sequential circuit with two D flip-flops A and B, and one input . 17](#_Toc122444629)

[1. When , the state of the circuit remains the same. When , the circuit goes through the state transitions from 00, to 01, to 11, to 10, back to 00, and repeats. 17](#_Toc122444630)

[2. When , the state of the circuit remains the same. When , the circuit goes through the state transitions from 00, to 11, to 01, to 10, back to 00, and repeats. 18](#_Toc122444631)

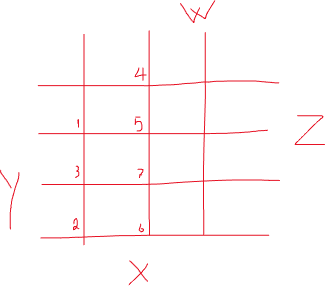
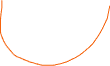
[4. Design an one-input, one output serial 2’s complementer. The circuit accepts a string of bits from the input and generates the 2’s complement at the output. 18](#_Toc122444632)

## Exercise 3 - Sandsynlighed for at være korrekt. 90 - 100 %

### Simplify the following Boolean functions, using four-variable maps:

#### 

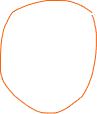
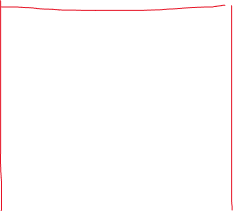
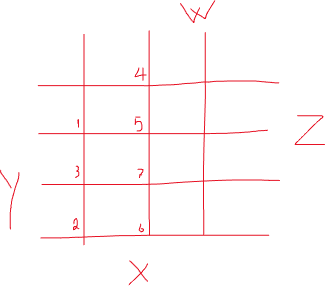
#### F(w,x,y,z) = ∑(1,4,5,6,12,14,15)



=================================

=================================

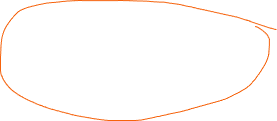
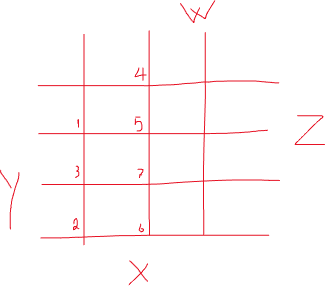
#### F(w,x,y,z) = ∑(2,3,6,7,12,13,14)



===========================

===========================

#### F(w,x,y,z) = ∑(1,3,4,5,6,7,9,11,13,15)

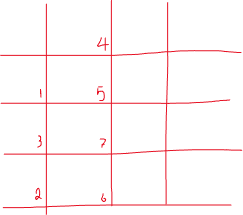


===================

===================

### Simplify the following Boolean expressions, using four-variable maps:

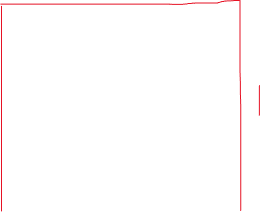
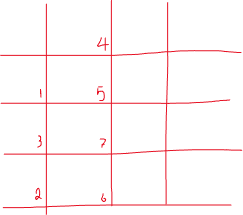
#### A′B′C′D′+AC′D′+B′CD′+A′BCD+BC′D



==============================

==============================

#### A′B′C′D+AB′D+A′BC′+ABCD+AB′C

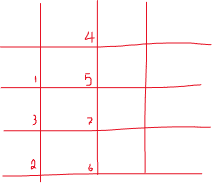


===========================================

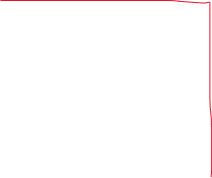
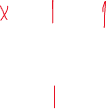
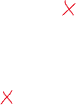
===========================================

### Simplify the following Boolean function F, together with the don’t care conditions d, and then express the simplified function in sum-of-minterms form:

Sum of minterms er de værdier med logisk 1.



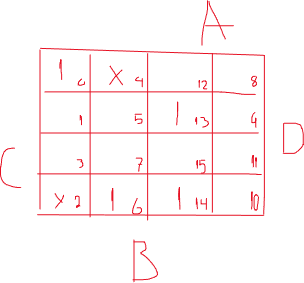
#### F(A,B,C,D) = ∑(2,4,7,10,12 , d(A,B,C,D) = ∑(0,6,8,15)



============================

============================

#### F(A,B,C,D) = ∑(0,6,8,13,14), d(A,B,C,D) = ∑(2,4,10)



=============================

=============================

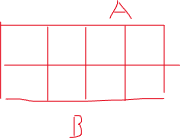
## Exercise 4 - Sandsynlighed for at være korrekt. 90 - 100%

### Design a combinational circuit with three inputs and one output:

#### The output is 1 when the binary value of the inputs is more than 2. The output is 0 otherwise.

Med 3 bits er den største værdi 7(111)

ABC



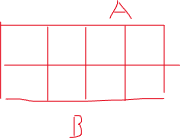
================================================================================



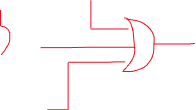
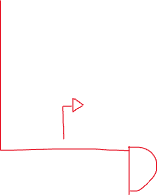
================================================================================

#### The output is 1 when the binary value of the inputs is not divisable by 3.

Divisable by 3 is interperated as the value not giving a whole number. 0 is interperated as divisable by 3, as .

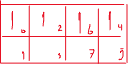


================================================================================



================================================================================

#### The output is 1 when the binary value of the inputs is an even number.



=============================



=============================

### Design a combinational circuit that converts a four-bit Gray code to a four-bit binary number. Implement the circuit with exclusive-OR gates.

The organic chemistry has made a video on this, which explains both Gray -> Binary and binary -> Gray very well!!

<https://www.youtube.com/watch?v=cbmh1DPPQyI>

When going binary to gray you are rewriting the msb and checking the binary value in pairs of two.   
If the pair has the same value, the gray value of this is 0, if it’s different then the gray value will be 1.

A demonstration of this can be seen below.



1. Step: Rewrite MSB: 1
2. Check the values of the binary number in pairs of two as seen above.

1 , 111110

Together it will be:

1111110

This can be turned back again using a different but kind of similar method.

You check the 2 index of the Gray code with the 1 index of its binary. If the same next binary value is 0, if it’s different, then the next binary number will be 1.

1. Step: Rewrite MSB: 1
2. Compare:



And look what a coincidence, this is our value that we started with, how odd is that wink, wink…

Okay so we now know of a method for bringing gray code to binary, but how are we gonna implement that.

1. The MSB will be kept
2. Grouping gray bit index i+1 with binary value index i, if different return 1, but how using xor gates?



And we know this as an XOR Gate

This means that for our 4 bits can be described as followed.

Now it’s time to build the circuit!!

==========================================================



===========================================================

Voila, its done!!

### Construct a 16 × 1 multiplexer with two 8 × 1 and one 2 × 1 multiplexers. Use block diagrams.

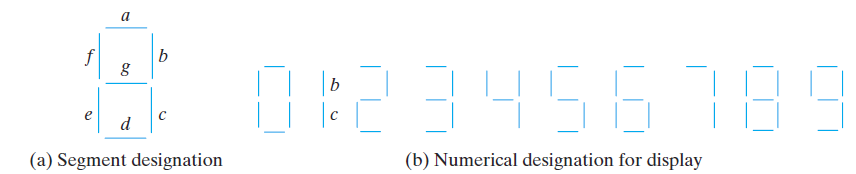
=======================================



=======================================

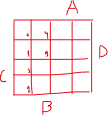
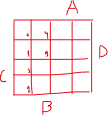
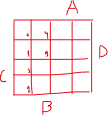
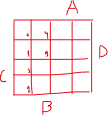
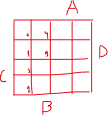
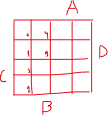
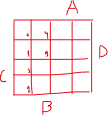
are selectors choosing values from binary value x000 to x111.

selects whether the outputs should be from 0(000) to 7(111) or 8(1000) to 15(1111).

An ABCD-to-seven-segment decoder is a combinational circuit that converts a decimal digit in BCD to an appropriate code for the selection of segments in an indicator used to display the decimal digit in a familiar form. The seven outputs of the decoder (a, b, c, d, e, f, g) select the corresponding segments in the display, as shown in Figure 1. 

### Using a truth table and Karnaugh maps, design the BCD-to-seven-segment decoder using a minimum number of gates. The six invalid combinations should result in a blank display :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D |  | Numbers |  | a | b | c | d | e | f | g |
| 0 | 0 | 0 | 0 |  | 0 |  | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 |  | 1 |  | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |  | 2 |  | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |  | 3 |  | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |  | 4 |  | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |  | 5 |  | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |  | 6 |  | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 |  | 7 |  | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |  | 8 |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |  | 9 |  | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| . | . | . | . |  | 0 |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



This is the least possible amount of gates used made by using karnaugh map for all 7 outputs. I could also draw the circuit, but that would take just as long if not longer to make, than these karnaugh maps, and I simply don’t think it’s necessary. If one were to draw this circuit, one just has to implement the functions with inverter-, AND- and OR gates as these functions are described and wire it all up. GOOD LUCK WITH THAT, it’s gonna take some time.

### Implement the following Boolean function with a 8 × 1 multiplexer. F (A, B, C, D) = ∑ (0, 2, 5, 8, 10, 14)

A 8 x 1 mux takes three selector inputs. These will in this design be the 3 most significant bits, A, B & C.



|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |
| 0 | 0 | 1 | 0 |  |  |  |  |  |  |  |  |
| 0 | 1 | 0 | 1 |  |  |  |  |  |  |  |  |
| 1 | 0 | 0 | 0 |  |  |  |  |  |  |  |  |
| 1 | 0 | 1 | 0 |  |  |  |  |  |  |  |  |
| 1 | 1 | 1 | 0 |  |  |  |  |  |  |  |  |

Now let’s draw the circuit:

=============================



=============================

### Implement the following Boolean function with a 4 × 1 multiplexer and external gates. F (A, B, C, D) = ∑ (1, 3, 4, 11, 12, 13, 14, 15)

Same procedure as last time, this time we have one 2 selectors, A & B. This is gonna require some more gates.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D |  |  |  |  |  |
| 0 | 0 | 0 | 1 |  |  |  |  |  |
| 0 | 0 | 1 | 1 |  |  |  |  |
| 0 | 1 | 0 | 0 |  |  |  |  |  |
| 1 | 0 | 1 | 1 |  |  |  |  |  |
| 1 | 1 | 0 | 0 |  |  |  |  |  |
| 1 | 1 | 0 | 1 |  |  |  |  |
| 1 | 1 | 1 | 0 |  |  |  |  |
| 1 | 1 | 1 | 1 |  |  |  |  |



Now let’s build the circuit!

==========================



==========================

## Exercise 5 - Sandsynlighed for at være korrekt 90 - 100 % udover opgave 1 som er ca. 60 %.

### The D latch of Figure 1 is constructed with four NAND gates and an inverter. Consider the following three other ways for obtaining a D latch. In each case, draw the logic diagram and verify the circuit operation:

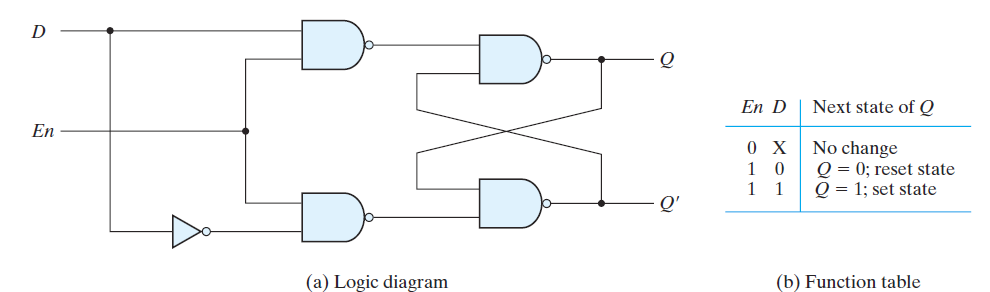


Figure 1

#### Use NOR gates for the SR latch part and AND gates for the other two. An inverter may be needed.

Alright let’s look at the latch. When D = 1 and En = 1, the upper nand wil result in 0.   
The bottom nand will result in 1.   
Moving into the latch, the upper part ( I think it’s S ), will at time = 0 be 0 not and 0, which will result in Q = 0. For the bottom part of the latch: Q not and 1 is 1.   
At time = 1, the upper part of the latch will then be 0 not and 1 which results in 1. The bottom part will then be 1 not and 1 which results in 0. At time = 1 we then have the function table as mentioned in figure 1.

Now let’s try to make it with nor gates for the latch.



De Morgan on or makes an nor gate:

Our inputs *D* should if 1 result in Q = 1.

At time = 1, the bottom part should be 0.

We then need to invert our *D coming into our S* input. results in 1.

Let’s now build the circuit:



Let’s now check

:

Which is like a no change?? Not quite sure of that. This might be what’s called the forbidden state.

Which were desired.

=======================

The circuit then checks out.

=======================

#### Use NOR gates for all four gates. Inverters may be needed.

We need to ensure this again, this time using only NOR gates.

If both signals coming into the NOR gate is 0. The enabler needs to be inverted!

If the enabler is inverted and if the input *D* is inverted before coming into the nor.

Let’s draw this circuit!



Now let’s check its functionality.

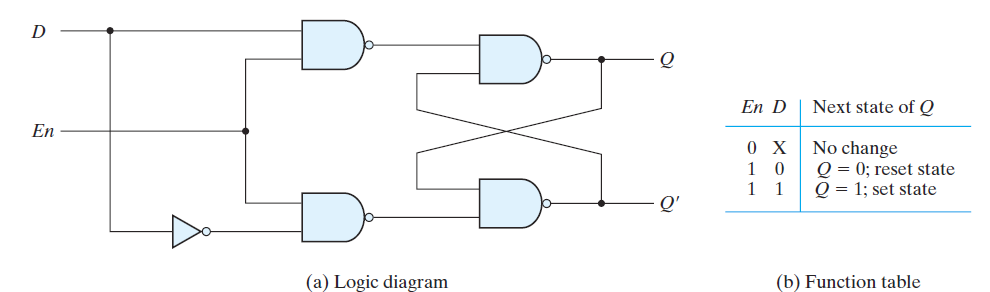
========================

This circuit checks out as well

========================

#### Use four NAND gates only (without an inverter). This can be done by connecting the output of the upper gate in Figure 1 (the gate that goes to the SR latch) to the input of the lower gate (instead of the inverter output)

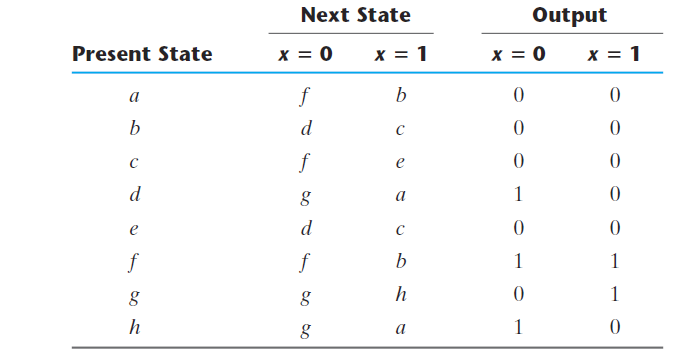
I will draw the explanation to clarify it further.





But am I finished now then? I could draw it again, but the circuit is up there ^, but with the modification that the task explained. I think this is sufficient.

### For the following state table





#### Tabulate the reduced state table

By reduced they ask us if any of these states can be removed in case it’s identical to another state.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Present state | Next state | | Output | |
|  |  |  |  |  |
| *a* | *f* | *b* | *0* | *0* |
| *b* | *d* | *a* | *0* | *0* |
| *d* | *g* | *a* | *1* | *0* |
| *f* | *f* | *b* | *1* | *1* |
| *g* | *g* | *d* | *0* | *1* |

Each state is now unique!

#### Draw the state diagram corresponding to the reduced state table

#### 



#### Deteremine the output sequence, starting from state a, and input sequence 01110010011.

The sequence moves between these states:

Does this check out?

The sequence 01110010011, has a length of 11.

, has as length of 11 if we were to ignore the state that it started from, but 12 if we don’t.

has a length of 11, which equals to 11 switches.

If I haven’t made a writing error while checking output from the different states, which is possible that I did, as I am only human, this should then be right.

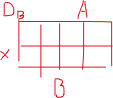
The output is then:

===========================

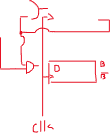
===========================

### Design a sequential circuit with two D flip-flops A and B, and one input .

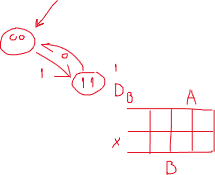
#### When , the state of the circuit remains the same. When , the circuit goes through the state transitions from 00, to 01, to 11, to 10, back to 00, and repeats.



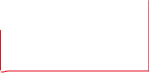
Now let’s draw the circuit!



#### When , the state of the circuit remains the same. When , the circuit goes through the state transitions from 00, to 11, to 01, to 10, back to 00, and repeats.



Now it’s drawing time!



### Design an one-input, one output serial 2’s complementer. The circuit accepts a string of bits from the input and generates the 2’s complement at the output.

I have made this before, now let’s think about it.

2’s complementer is a 1’s complimenter but with 1 added to it.

But wait, we now have a carry.

Let’s imagine we are shifting these values to the right.



We should have two states for the least significant bit.

If the LSB is 1, then let’s go to next state with its output being 0.

If the LSB is 0, then let’s keep the zero, go to a state looking for the input to be 1 before going to next state and until then, stay in this state and output 0.

Let’s see how this looks in a state diagram.



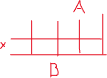
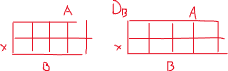
Does this make sense?   
When taking the first input, check if the value is 0 or 1. If 0, we know that this will be 1 after the 1’s compliment of this, but the 2’s complimenter adds 1 to this, so this will be 0, and ADDS A CARRY! LET’S GO TO STATE 01

While we now have a carry, the only possible combination that doesn’t add a carry is when our input is 1, as this will be complimented to 0 and then adds the carry to become 1. If the value is 0 then just return 0, as this will be the result after complimenting and adding the carry.

We can now go to state 10, were the rest of the bits acts as it would be in a 1’s complimenter.

And then enter the 1’s complimenter in state 10.

, which is the 2’s complimenter of .



Now let’s draw the final circuit!



Voila!