SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

Systemarchitektur SS 2021

Präsenzblatt 5 (Lösungsvorschläge)

Hinweis: **Dieses Aufgabenblatt wurde von Tutoren erstellt. Die Aufgaben sind für die Klausur weder relevant noch irrelevant, die Lösungsvorschlage weder korrekt noch inkorrekt.**

Aufgabe 5.1: Multiplikation von Binärzahlen

Sie haben in der Vorlesung gelernt, Binärzahlen zu multiplizieren. Multiplizieren Sie 1011 und 0111 mit der

- vorzeichenlosen Multiplikation und
- Zweier-Komplement-Multiplikation.

Lösungsvorschlag:

- $1011 \cdot 0111 = 00001011 + 00010110 + 00101100 + 00000000 = 01001101$
  $\langle 01001101 \rangle = 77 = 11 \cdot 7 = \langle 1011 \rangle \cdot \langle 0111 \rangle$
- $1011 \cdot 0111 = 11111011 + 11110110 + 11101100 - 00000000 = 11011101$
  $[11011101] = -35 = -5 \cdot 7 = [1011] \cdot [0111]$

Aufgabe 5.2: Arithmetic Logic Unit

In dieser Aufgabe möchten wir eine einfache arithmetisch-logische Einheit (ALU) entwerfen, die auf zwei n-Bit-Binärzahlen A und B die folgenden Operationen anwenden kann:

- $A \wedge B$
- $A \wedge \neg B$
- $A \vee B$
- $A \vee \neg B$
- $A + B$
- $A - B$

Dabei bezeichnet $\wedge$ die bitweise Verundung, $\vee$ die bitweise Veroderung und $\neg$ die bitweise Negation. Außerdem soll die ALU jeweils einen Bitstring bestehend nur aus Einsen bzw. Nullen ausgeben können.

1. Ordnen Sie die acht Funktionen sinnvoll dem select-Eingangssignal s zu, indem Sie die folgende Tabelle vervollständigen. Warum sind drei Steuerbits ausreichend?

s_2	s_1	s_0	Funktion
0	0	0	
0	0	1	$A \vee B$
0	1	0	
0	1	1	
1	0	0	$A \wedge \neg B$
1	0	1	
1	1	0	
1	1	1	1...1

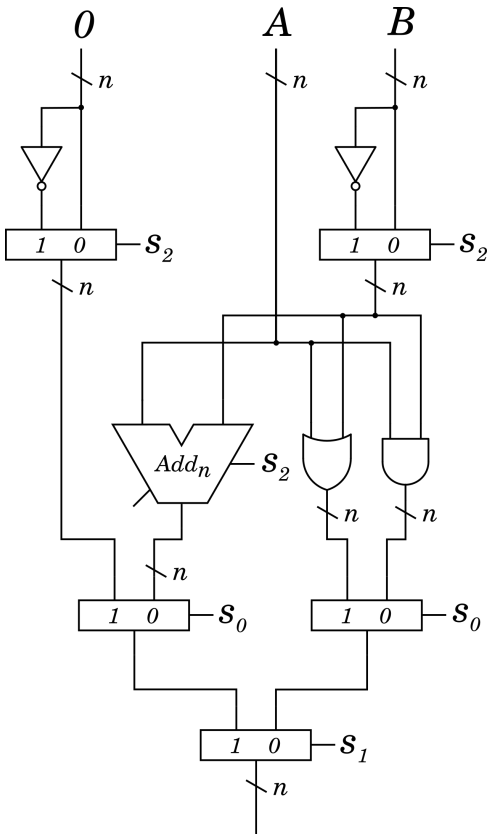
2. Realisieren Sie die ALU mithilfe eines Addierers, elementarer Gatter aus unserer Standardbibliothek sowie mehreren Multiplexern.

Lösungsvorschlag:

1.

s_2	s_1	s_0	Funktion
0	0	0	$A \wedge B$
0	0	1	$A \vee B$
0	1	0	$A + B$
0	1	1	0...0
1	0	0	$A \wedge \neg B$
1	0	1	$A \vee \neg B$
1	1	0	$A - B$
1	1	1	1...1

2.



Aufgabe 5.3: Endliche Automaten

In dieser Aufgabe werden wir den bekanntesten Mathematiker der Welt modellieren. Dafür werden wir diesen zuerst bis auf die Knochen beschreiben. Unser guter Freund...

- kann vergessen oder zählen.
- zählt von eins bis drei. Wenn er vergisst, wo er war, bleibt er bei seiner momentanen Zahl.
- startet wieder bei eins, sobald er von drei an weiter zählt.
- ruft nach jedem Zählen laut "Muhaha". Wenn er vergisst, wo er war, sagt er nichts.

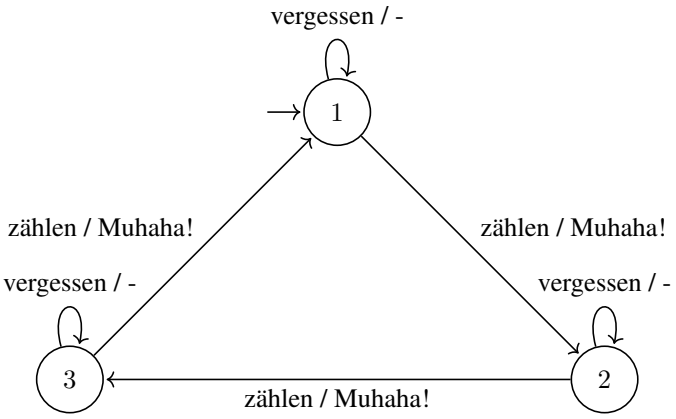
Ihre Aufgabe ist es nun dieses überaus durchdachtes Zahlengenie mit Hilfe eines

1. Mealy Automaten
2. Moore Automaten

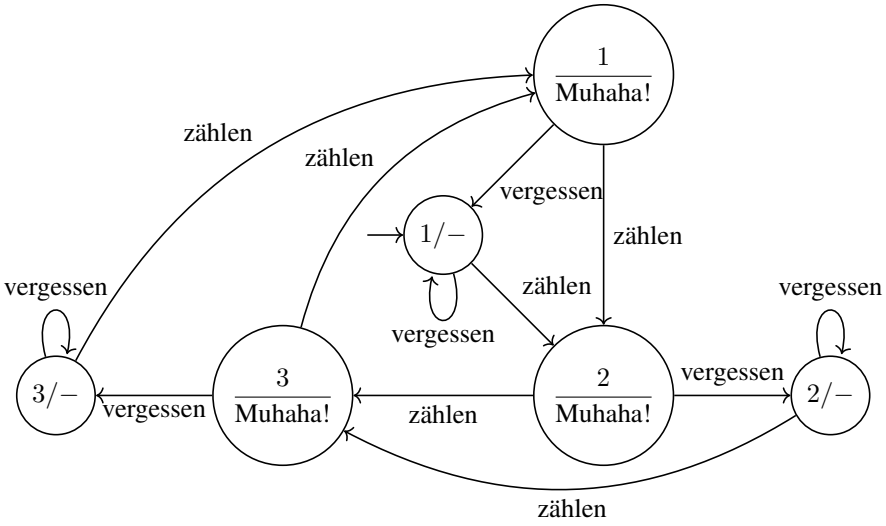
zu modellieren.

Lösungsvorschlag:

1. Der Mealy Automat:



2. Der Moore Automat:

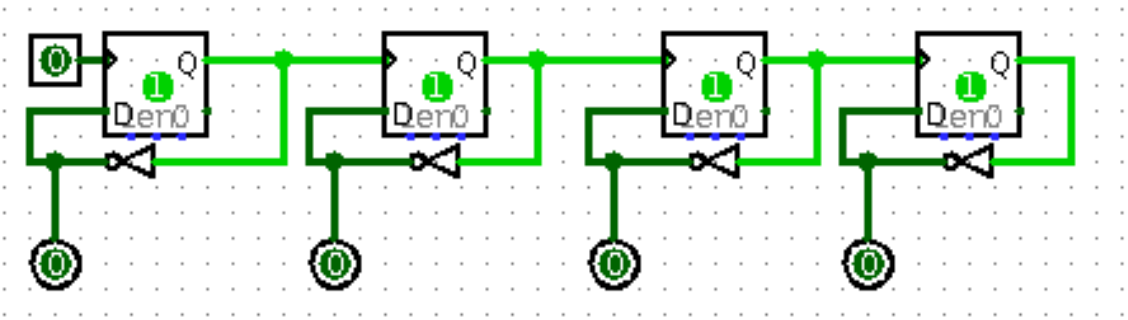


Aufgabe 5.4: Schaltwerke

1. In der Vorlesung haben Sie bereits gesehen, wie man einen Zähler mit D-Flip-Flops und Inkrementer implementiert. Geht dies auch nur mit D-Flip-Flops (und Negationen)? Falls ja, geben Sie ein entsprechendes Schaltwerk an. Begründen Sie andernfalls, warum es nicht möglich ist.
2. Realisieren Sie das Schaltwerk zur Aufgabe 5.3.1.
3. Erstellen Sie ein Schaltwerk mit einem 3 bit Zustand, der Primzahlen aufzählen kann: $0 \to 2 \to 3 \to 5 \to 7 \to 2 \to 3 \to \ldots$

Lösungsvorschlag:

1.



D-Counter

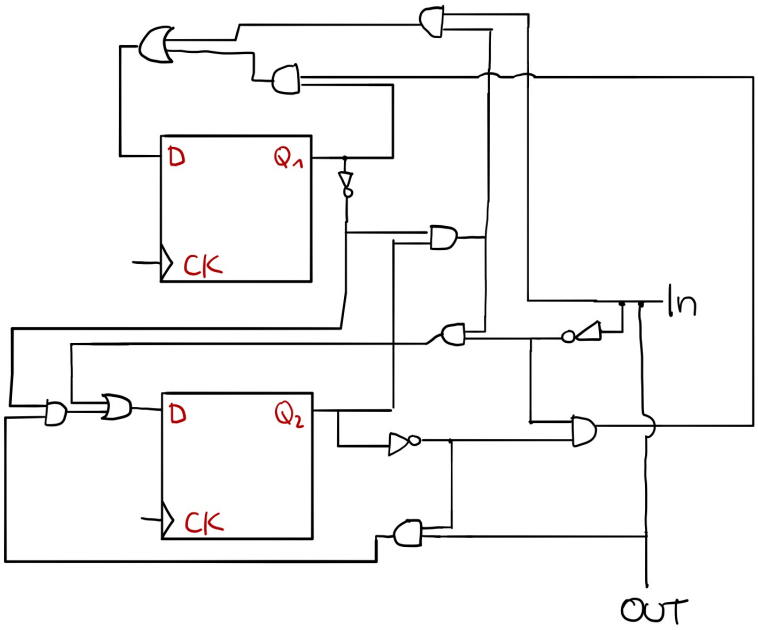
2.
- Übergangsfunktion:
  - Zustände: $1 \mapsto 00, 2 \mapsto 01, 3 \mapsto 10$
  - Eingaben: vergessen $\mapsto 0$, zählen $\mapsto 1$
  - Wertetabelle:

q_1	q_2	In	x_1	x_2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	0

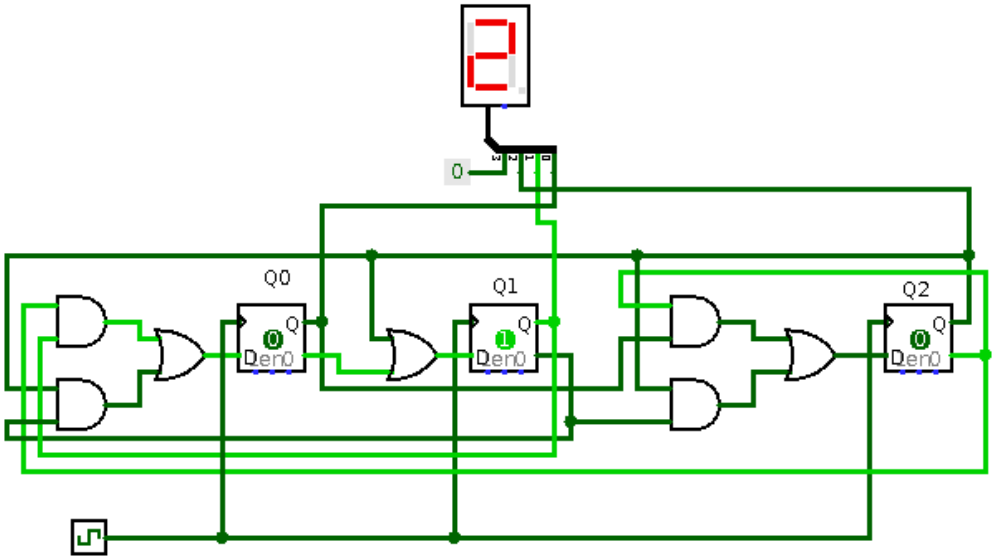
  - Minterme für x_1: $\neg q_1 \cdot q_2 \cdot In$ und $q_1 \cdot \neg q_2 \cdot \neg In$
  - Minterme für x_2: $\neg q_1 \cdot \neg q_2 \cdot In$ und $\neg q_1 \cdot q_2 \cdot \neg In$
- Ausgabefunktion:
  - Ausgabe: - $\mapsto 0$, Muhaha! $\mapsto 1$
  - Wertetabelle:

q_1	q_2	In	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1

- Schaltwerk:



3.



Aufgabe 5.5: Einführung Verilog

1. Wie kann man einen Halbaddierer in Verilog darstellen?
2. Nutzen Sie den Halbaddierer nun, um einen Volladdierer zu konstruieren.
3. Konstruieren sie den Zähler $Count_{16}$ aus Aufgabe 5.2.1 mit Verilog.
4. Kombinieren Sie Zustände und Addierer zu einem Zähler, der immer eine Eingabe auf den Zustand addiert.

Lösungsvorschlag:

1. Halbaddierer:

```
1  module half_adder (
2    input a,
3    input b,
4    output s,
5    output c
6  );
7    assign s = a ^ b;
8    assign c = a & b;
9  endmodule
```

2. Volladdierer:

```
1  module full_adder (
2    input a0,
3    input b0,
4    input c,
5    output s0,
6    output s1
7  );
8    wire ha0c, ha0s, ha1c;
9
10   half_adder ha0(.a(a0), .b(b0), .c(ha0c), .s(ha0s));
11   half_adder ha1(.a(ha0s), .b(c), .c(ha1c), .s(s0));
12   assign s1 = ha0c | ha1c;
13 endmodule
```

3. Counter:

```
1  module count (
2    input clock,
3    input R,
4    output [3:0] out
5  );
6    reg [3:0] Q;
7    initial begin
8       Q=0;
9    end
10   always @(posedge clock)
11   begin
12         if (R)
13                Q <= 4'b0;
14         else
15                Q <= Q + 1'b1;
16   end
17   assign out = Q;
18 endmodule
```

4. SumAdd:

```
1  module sumadd #(parameter W = 4)(
2    input clk,
3    input rst,
4    input en,
```

```
5     input [W-1:0] A,
6     output [2*W-1:0] O
7   );
8     reg [2*W-1:0] Q;
9     always @(posedge clk or posedge rst)
10      if (rst) Q <= 0;
11           else if (en) Q <= Q + A;
12    assign O = Q;
13  endmodule
```

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

System Architecture SS 2021

Tutorial Sheet 5 (Suggested Solutions)

Note: **This task sheet was created by tutors. The tasks are neither relevant nor irrelevant for the exam, the suggested solutions are neither correct nor incorrect.**

Problem 5.1: Multiplication of binary numbers

In the lecture we have learned how to multiply binary numbers.
Multiply 1011 and 0111 with

- unsigned multiplication and
- signed multiplication in two's complement representation.

Suggested solution:

- $1011 \cdot 0111 = 00001011 + 00010110 + 00101100 + 00000000 = 01001101$
  $\langle 01001101 \rangle = 77 = 11 \cdot 7 = \langle 1011 \rangle \cdot \langle 0111 \rangle$
- $1011 \cdot 0111 = 11111011 + 11110110 + 11101100 - 00000000 = 11011101$
  $[11011101] = -35 = -5 \cdot 7 = [1011] \cdot [0111]$

Problem 5.2: Arithmetic Logic Unit

In this exercise we will design a simple arithmetic logic unit (ALU). It should be able to do the following operations on two n-bit binary numbers A and B.

- $A \wedge B$
- $A \wedge \neg B$
- $A \vee B$
- $A \vee \neg B$
- $A + B$
- $A - B$
- output all ones
- output all zeros

$\vee, \neg$ and $\wedge$ should be applied bitwise.

1. Assign these operations to the values of the select input s. Try to make your assignment useful for the implementation of the circuit. Complete the following table. Why do we need 3 select bits?

s_2	s_1	s_0	function
0	0	0	
0	0	1	$A \vee B$
0	1	0	
0	1	1	
1	0	0	$A \wedge \neg B$
1	0	1	
1	1	0	
1	1	1	1...1

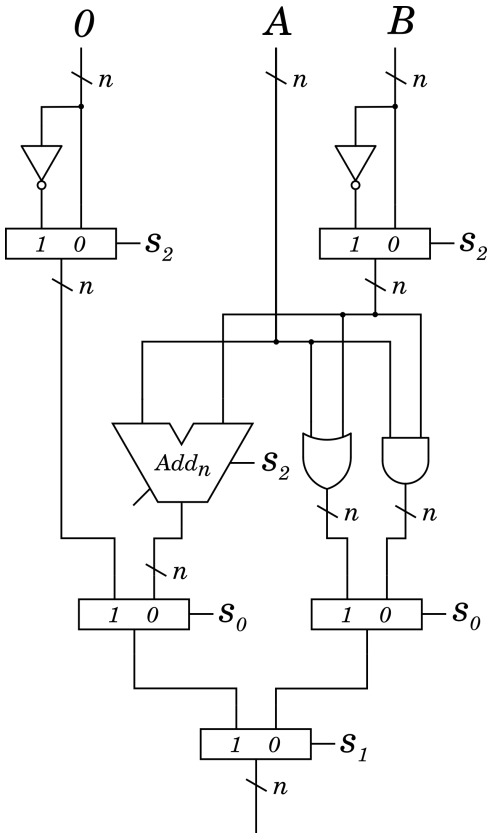
2. Implement the ALU using one adder as well as multiple multiplexers and gates from our standard library.

Suggested solution:

1.

s_2	s_1	s_0	function
0	0	0	$A \wedge B$
0	0	1	$A \vee B$
0	1	0	$A + B$
0	1	1	0...0
1	0	0	$A \wedge \neg B$
1	0	1	$A \vee \neg B$
1	1	0	$A - B$
1	1	1	1...1

2.



Problem 5.3: Finite state machine

In this exercises we will model the most famous mathematician of all time. First we need to describe him:

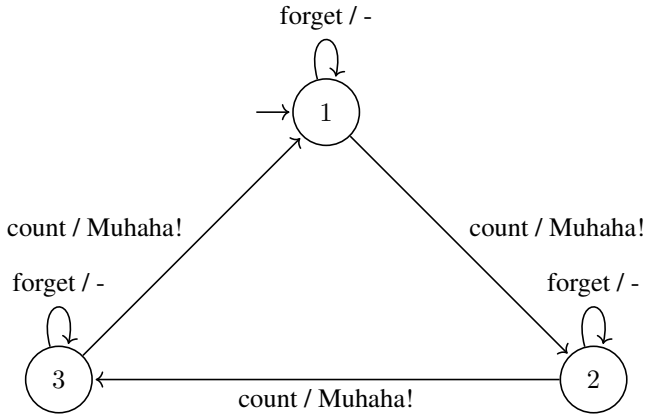
- he can forget or count.
- he counts from one to three. If he forgets he stays at his current number.
- after he has reached three he will start counting from one again.
- he shouts 'Muhaha' on every number. If he forgets he says nothing.

Your task is to model our genius by using a

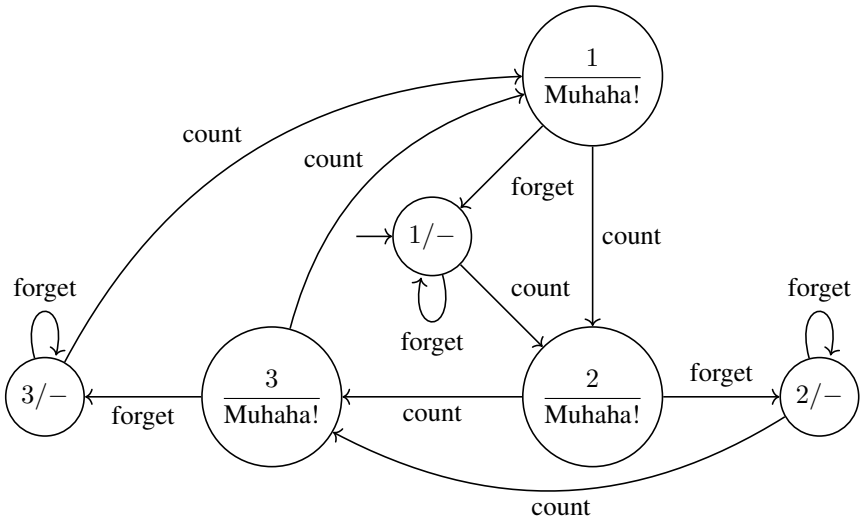
1. Mealy machine
2. Moore machine

Suggested solution:

1. Mealy machine:



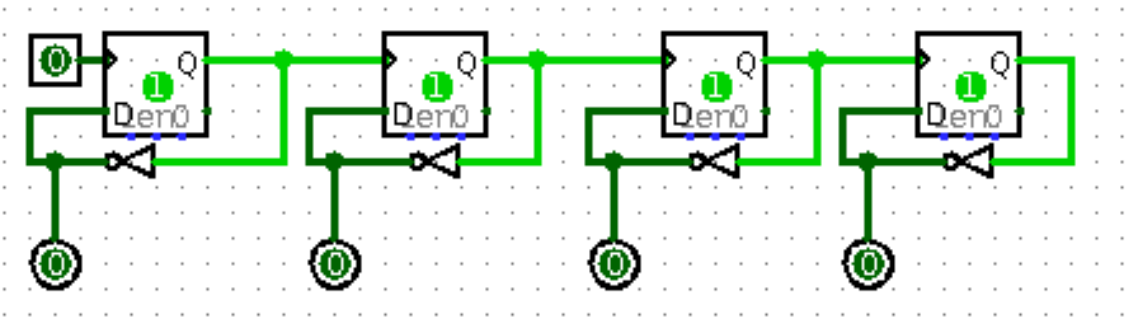
2. Moore machine:



Problem 5.4: Sequential circuits

1. We have already seen how to implement a counter using D-flip-flops and an incrementer. Can we also do that with only D-flip-flops and inverters? If yes, design the circuit. If no, explain why it is impossible.
2. Implement the sequential circuit from exercises 5.3.1.
3. Design a sequential circuit with a 3 bit state, that counts prime numbers as follows: $0 \to 2 \to 3 \to 5 \to 7 \to 2 \to 3 \to \ldots$

Suggested solution:

1. 

D-Counter

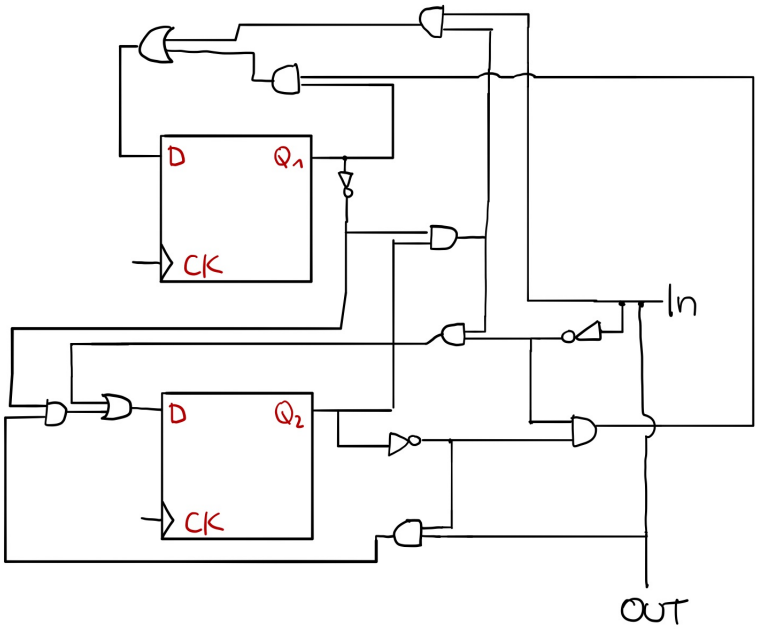
2.
   - Transition function:
     - States: $1 \mapsto 00, 2 \mapsto 01, 3 \mapsto 10$
     - Inputs: vergessen $\mapsto 0$, zählen $\mapsto 1$
     - Table of values:

q_1	q_2	In	x_1	x_2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	0	0

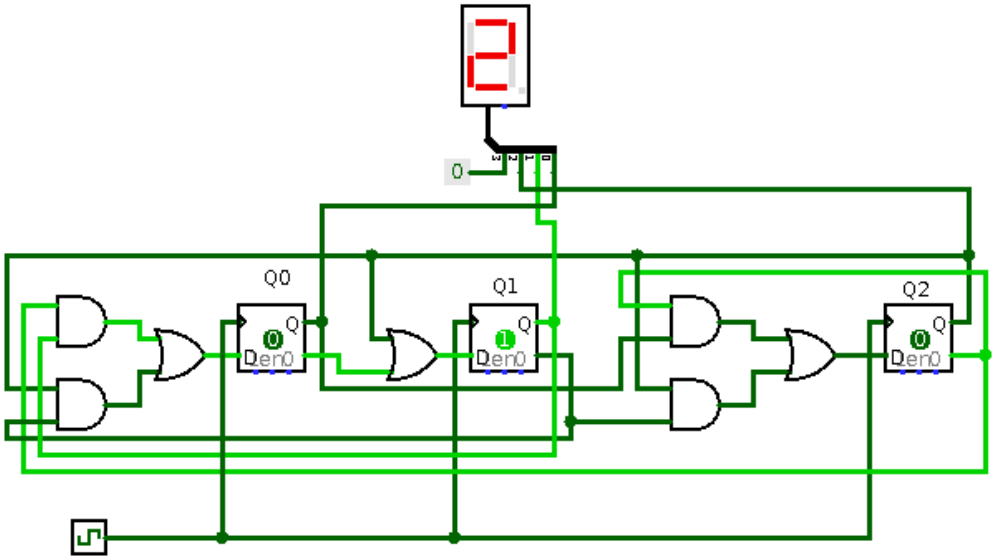
     - Minterms of x_1: $\neg q_1 \cdot q_2 \cdot In$ and $q_1 \cdot \neg q_2 \cdot \neg In$
     - Minterms of x_2: $\neg q_1 \cdot \neg q_2 \cdot In$ and $\neg q_1 \cdot q_2 \cdot \neg In$
   - Output function:
     - Output: - $\mapsto 0$, Muhaha! $\mapsto 1$
     - Table of values:

q_1	q_2	In	Out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1

   - Circuit:



3.



Problem 5.5: Introduction to Verilog

1. Write a half adder in Verilog.
2. Use the half adder to build a full adder.
3. Design a 16-bit counter in Verilog.
4. Write a circuit in Verilog that adds up inputs that come sequentially.

Suggested solution:

1. Half adder:

```
module half_adder (
  input a,
  input b,
  output s,
  output c
);
  assign s = a ^ b;
  assign c = a & b;
endmodule
```

2. Full adder:

```
module full_adder (
  input a0,
  input b0,
  input c,
  output s0,
  output s1
);
  wire ha0c, ha0s, ha1c;

  half_adder ha0(.a(a0), .b(b0), .c(ha0c), .s(ha0s));
  half_adder ha1(.a(ha0s), .b(c), .c(ha1c), .s(s0));
  assign s1 = ha0c | ha1c;
endmodule
```

3. Counter:

```
module count (
  input clock,
  input R,
  output [3:0] out
);
  reg [3:0] Q;
  initial begin
     Q=0;
  end
  always @(posedge clock)
  begin
        if (R)
                Q <= 4'b0;
        else
                Q <= Q + 1'b1;
  end
  assign out = Q;
endmodule
```

4. SumAdd:

```
module sumadd #(parameter W = 4)(
  input clk,
  input rst,
  input en,
```

```
5     input [W-1:0] A,
6     output [2*W-1:0] O
7   );
8     reg [2*W-1:0] Q;
9     always @(posedge clk or posedge rst)
10      if (rst) Q <= 0;
11           else if (en) Q <= Q + A;
12    assign O = Q;
13  endmodule
```