

# 6 Sequential Logic

## Student Group

First Name	Surname	Matrikel Nr.

## Table of Contents

<b>6. Sequential Logic</b> .....	2
<b>6.1 First Terminology</b> .....	2
Exercise 6.1.3.1 Example of a State Machine .....	3
<b>6.2 Classical State Machine Types</b> .....	3
6.2.1 Moore Machine .....	3
Note! .....	4
6.2.2 Mealy Machine .....	4
Note! .....	5
Exercise 6.2.2.1 Invalid States of the Mealy Machine .....	5
6.2.3 Medvedev Machine .....	5
Note! .....	5
Note! .....	6
<b>6.3 State Diagram, State Transition Diagram</b> .....	6
6.3.1 Motivation .....	6
6.3.2 Simple logic Example .....	8
6.3.3 First Adaption for Up-Counter .....	11
Exercise 6.3.3.1. State Transition Diagram in Digital.exe .....	12
Exercise 6.3.3.2. State Transition Diagram II in Digital.exe .....	13
Exercise 6.3.3.3. State Transition Diagram III in Digital.exe .....	14
Exercise 6.3.3.4. State Transition Diagram III in Digital.exe .....	15
<b>Exercises</b> .....	15
Exercise 6.1.1. State Transition Diagram I .....	15
Exercise 6.1.2. State Transition Diagram II .....	16
Exercise 6.1.3. State Transition Diagram III .....	16

# 6. Sequential Logic

“I Know What You Did Last Cycle”

## 6.1 First Terminology

The most important term is the word **state** But what is a state? It is a unique situation, where the possible next steps (= possible next states), the inner behavior or the outputs are distinguishable from other situations. Here some practical examples:

- being happy or being sad, are two different states, since the inner behavior is different (this least often also to a different output). Similarly, an empty memory (or harddrive) is in a different state compared to an compared to a full one. Even a fresh deleted one is distingisable, since deletion

Sequential logic is used to describe logic cicruits which show internal states (“stateful”), and therefore have at least one memory element (= flip-flop).

The following terminology is used in the upcoming explanations:

- The **input vector**  $\vec{X}$  represents the  $k$  inputs  $X_0 \dots X_{k-1}$
- The **output vector**  $\vec{Y}$  represents the  $l$  outputs  $Y_0 \dots Y_{l-1}$
- The **state vector**  $\vec{Z}$  represents the  $m$  inputs  $Z_0 \dots Z_{m-1}$
- The sign  $(n)$  or  $n$  marking the current point in time and therefore e.g. the current state  $Z_0(n)$
- The sign  $(n+1)$  or  $n+1$  marking the next upcomming point in time and therefore e.g. the next state  $Z_0(n+1)$
- Sequential logic circuits are also called **Finite State Machines** (FSM) or sometimes also shortened to “state machine”

The [figure 12](#) shows the different terms in an abstract diagram. The “current” output values are here the values, which are shown after the delay time of the gates (about some nanoseconds).



Fig. 1: Abstract View onto a Sequential Logic

The principle interior of the blackbox in [figure 12](#) was already shown in one practical application in the

[previous chapter](#): We saw, that we need the combination of an combinatorial logic and some storage components. Additionally, both have to be connected by feeding back some of the outputs back to the combinatorial logic. The output bits  $\vec{Y}$  can result either from the combinatorial logic or the flip-flops. This is shown in [figure 13](#).

Fig. 2: One Step more into the Sequential Logic



### Exercise 6.1.3.1 Example of a State Machine

[figure 1](#) depicts a state machine.

- What happens, when  $X$  is changed? On which edge the change is triggered?
- Write down how many components each vector  $\vec{X}$  and  $\vec{Y}$  has.
- How many bits (= flip flops) might the state vector  $\vec{Z}$  need?

Fig. 3: Example for a sequential logic

One simple example of a sequential logic is shown in [figure 4](#). There, the combinatorial logic is explicitly shown. Depending on the input  $X$  the output  $\vec{Y}$  shows an up-counting 2-bit value counting  $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow \dots$ . This is a simple state machine which will be used in the net chapters

Fig. 4: State machine of an Up-Counter

## 6.2 Classical State Machine Types

The up-counter in the previous sub-chapter was able to count from  $0$  to  $3$ . But what can we do in order to count differently, like  $7, 6, 1, 5$ ? In order to understand this, a simpler situation will be investigated. So, let's look how one can create a up-counter counting  $1, 2, 3, 4$ .

### 6.2.1 Moore Machine

The first idea might be to use what we already have: an up-counter, which facilitate 2 flip-flops in order to result into 2-bit output. The wanted new state machine needs 3 bits for the output, since the binary representation of our outputs are  $\%001\$, \%010\$, \%011\$, \%100\%$ .

An simple idea is to take the 2-bit up-counter and add an combinatorial logic in behind. This logic shall convert the 2-bit up-counter output  $\$00\$$  into  $\$001\$$ , the  $\$01\$$  into  $\$010\$$ ,  $\$10\$$  into  $\$011\$$  and  $\$11\$$  into  $\$101\$$ . This can be logic can be created by:

- writing down the truth table
- putting the values into a Karnaugh map
- extracting the formula with view onto the implicants
- generating the circuit with gates

When this is done, the result looks like [figure 5](#)

Fig. 5: Up-Counter 1..4 as a Moore Machine

This resembles a so called **Moore Machine**

### Note!

A state machine is a **Moore Machine**, when the output values  $\vec{Y}$  depends only on the state values  $\vec{Z}$ . For this the moore machine uses two combinatorial circuits:

- The input circuit, which process the input values  $X^{(n+1)}$  and the state values  $Z^{(n)}$  (of the previous step) in such a way that the new states  $Z^{(n+1)}$  are generated.
- The output circuit, which transform the state values  $Z^{(n)}$  into the output values  $Y^{(n)}$ .

The properties of a moore machine are:

- The number of flip-flops is only given by number of states  $m$ .
- The output only changes when an edge on the clock input happen. The moore machine is a **synchronous state machine**.
- The moore machine usually need less logic gates. But this comes with the cost of optimizing two combinatorial logic circuits.

## 6.2.2 Mealy Machine

When looking onto [figure 5](#) a bit more in detail, one can see, that the outputs  $Y_0$  and  $Y_1$  just equals output of the first combinatorial logic circuit. This is not surprising: the input logic circuit shows the  $Z^{(n+1)}$  and this is for the counter always the stored value plus one, except when the maximum is reached.

With this information the state machine in [figure 5](#) can be simplified by using the outputs of the input circuit for  $Y_0$  and  $Y_1$ . This is shown in [figure 6](#).

Fig. 6: Up-Counter 1..4 as a Mealy Machine

**Note!**

A state machine is a **Mealy Machine**, when the output values  $\vec{Y}$  depends not only on the state values  $\vec{Z}$ . For this the mealy machine uses two combinatorial circuits:

- The input circuit, which process the input values  $X_{(n+1)}$  and the state values  $Z_{(n)}$  (of the previous step) in such a way that the new states  $Z_{(n+1)}$  are generated.
- The output circuit, which transform the state values  $Z_{(n)}$  and some inputs from ahead of the flip-flops into the output values  $Y_{(n)}$ .

The properties of a mealy machine are:

- The number of flip-flops is only given by number of states  $m$ .
- The output **not** only changes when an edge on the clock input happen: It is also dependent on the input  $X_{(n)}$ . The mealy machine is an **asynchronous state machine**.
- The mealy machine usually need less logic gates.
- The mealy machine has to be designed properly in order not to get invalid outputs.

**Exercise 6.2.2.1 Invalid States of the Mealy Machine**

The mealy machine in [figure 6](#) can show invalid outputs. Try to find these by the correct timing of the input  $X=1$  or  $X=0$ .

- Which outputs can be created?

**6.2.3 Medvedev Machine**

Fig. 7: Up-Counter 1..4 as a Medvedev Machine

**Note!**

A state machine is a **Medvedev Machine**, when the output values  $\vec{Y}$  is directly given by the state values  $\vec{Z}$ . For this the medvedev machine uses only one combinatorial circuit. This circuit process the input values  $X_{(n+1)}$  and the state values  $Z_{(n)}$  (of the previous step) in such a way that the new states  $Z_{(n+1)}$  and the output values  $Y_{(n+1)} = Z_{(n+1)}$  are generated.

The properties of a medvedev machine are:

- The number of flip-flops is given by number of outputs  $l$ .
- The output only changes when an edge on the clock input happens: The mealy machine is an **synchronous state machine**.

- The medvedev machine usually need more logic gates.

The [figure 8](#) shows the principle differences in the architecture of the state machines.

Fig. 8: States of Water  
Moore Machine



Mealy Machine



MedvedevMachine



### Note!

This chapter is only focussing on Moore machines.

## 6.3 State Diagram, State Transition Diagram

### 6.3.1 Motivation

The diagrams of different states are well known from physics for example the state diagram (or better: phase diagram) of water, where its three states are: solid ice, liquid water and gaseous steam. The possible state transitions are due to temperature increase or decrease.

In [figure 9](#) image (1) the states of water are shown on the temperature axis. When only the state transitions are relevant, the states are simplified to a circle, showing the state name and behaviour. The transitions are depicted as arrows, where the needed condition is written onto (See [figure 9](#) image

(2) ). This diagram is called **state transition diagram**.



Fig. 9: States of Water

For matter not only the dimension “temperature” is important, but also the “pressure”. The full phase diagram is shown in [figure 10 image \(1\)](#). By this, another variable is available and more transistions. These can be drawn into the state transition diagram ([figure 10 image \(2\)](#)).

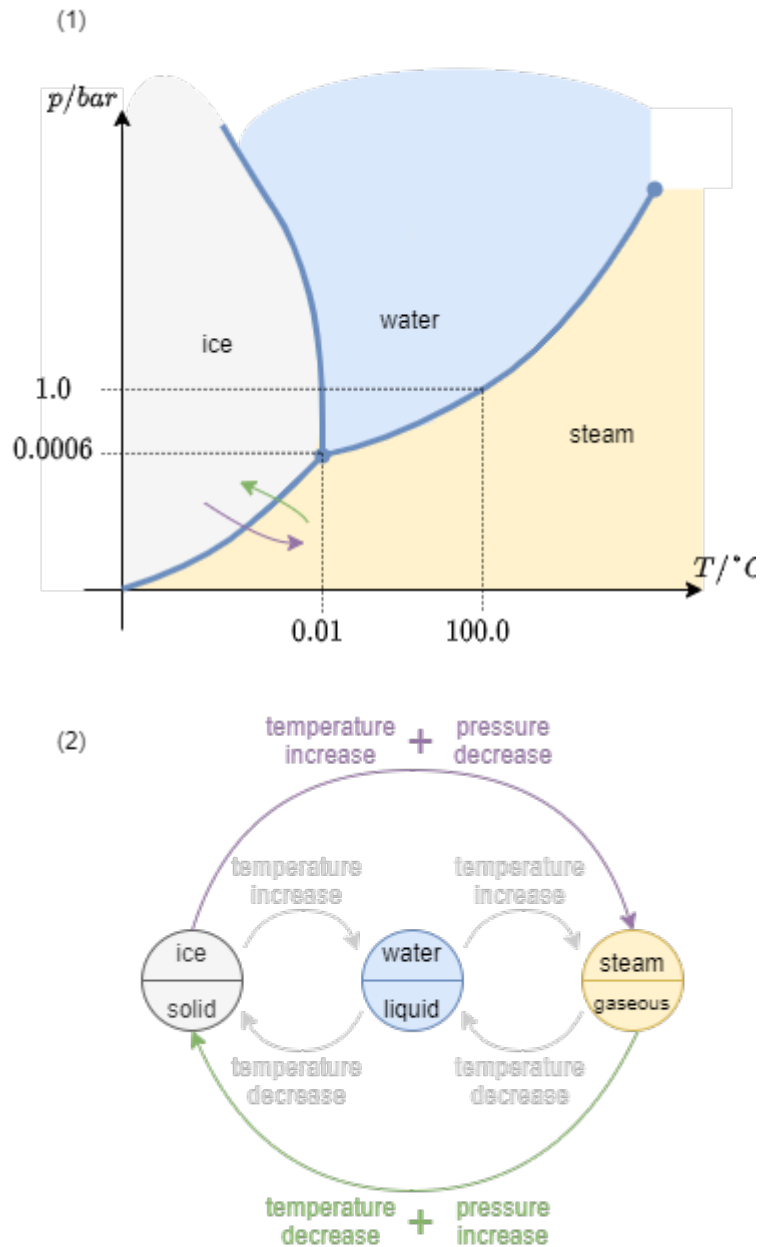


Fig. 10: States of Water

### 6.3.2 Simple logic Example

In German, often one has to pay for entering the toilet. An example of such an entrance control system is shown in figure 11. At this (artificial) example, one can pay either 50ct or 1€. Once paid, the turnstile will release and one can enter. Once the turnstile was pushed the entrance is closed again.



Fig. 11: Entrance Control for Toilets

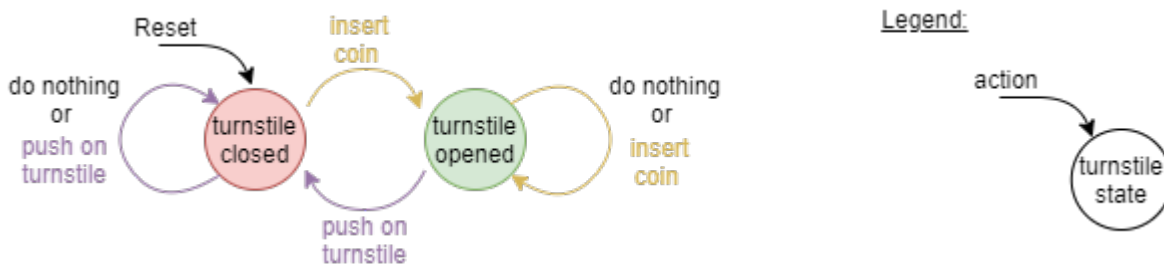
The figure 12 the state transition diagram is drawn.

- The two states are that (1) the turnstile is opened and one is able to go through and (2) the turnstile is closed and one cannot enter anymore.
- The transitions are given by the done actions: one can either insert a coin or push on the turnstile.

Important here are some additional considerations:

- For the state transition diagram one has to **look for all possible transitions**. So, also pushing a closed turnstile or inserting more coins have to taken into account.
- A state transition diagram is not complete without a **legend** and without an **beginning/reset point**. The reset point is given by an arrow with "reset" written onto it

Fig. 12: State Transition Diagram of the Entrance Control for Toilets



Out of this state transition diagram one can create a table-like representation, see figure 13.

Fig. 13: State Transition Diagram of the Entrance Control for Toilets

Toilet Entrance Control			
current state	input / event	next state	output / action
turnstile closed	push turnstile	turnstile closed	disallow entrance
turnstile closed	insert coin	turnstile opened	allow entrance
turnstile opened	push turnstile	turnstile closed	disallow entrance
turnstile opened	insert coin	turnstile opened	allow entrance

the inputs, outputs and states have to be encoded into binary, in order to investigate this table a bit more. How the binary value is connected to the outputs does not matter. We will choose the following coding:

- Encoding of the states: turnstile closed  $\triangleq Z=0$ , turnstile opened  $\triangleq Z=1$ ,
- Encoding of the inputs: no coin inserted  $\triangleq Xc=0$ , coin inserted  $\triangleq Xc=1$ , turnstile not pushed  $\triangleq Xp=0$ , turnstile pushed  $\triangleq Xp=1$ ,
- Encoding of the outputs: disallow entrance  $\triangleq Y=0$ , allow entrance  $\triangleq Y=1$ ,

This table is shown in figure 14 and is called **state transition table**.

Fig. 14: State Transition Diagram of the Entrance Control for Toilets

Toilet Entrance Control						
current state	$Z(n)$	$Xc$	$Xp$	next state	$Z(n + 1)$	$Y$
	0	0	-		0	0
	0	1	0		1	1
	1	0	1		0	0
	1	-	0		1	1



Interestingly, the logic circuit for this state transition table was already part of the course: it is the RS flip-flop! When looking deeper onto the table in figure 14 one can substitute  $Xc$  with  $S$  (as in Set) and  $Xp$  with  $R$  (as in Reset) to directly get the truthtable of the RS flip flop.

### 6.3.3 First Adaption for Up-Counter

In the previous sub-chapter (6.2) we had a look onto different implementations of an up counter and on this chapter the way to represent a state machine via a state transition diagram. So one question is: how does these up counter look like in the state transition diagram?

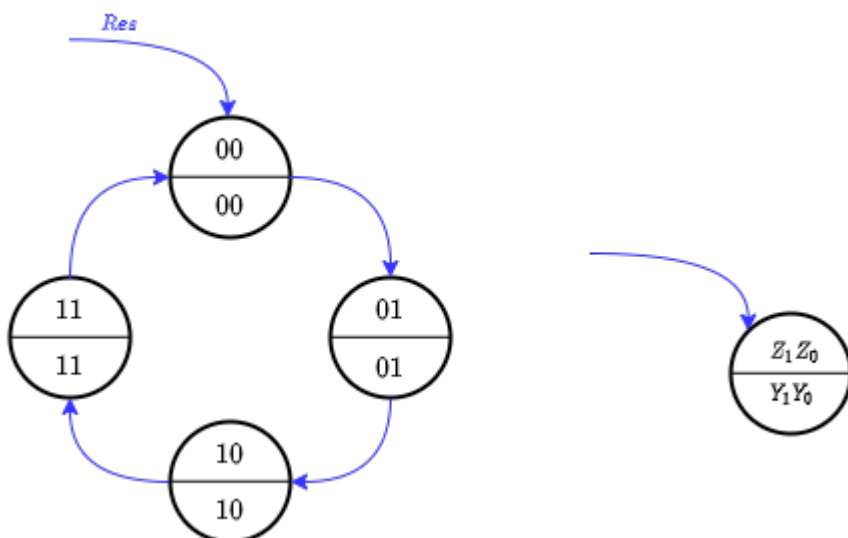
The Answer ist quite simple: there are 4 states, so 4 "circles" are needed. These states need to have circular connections in order to get the wanted increase from  $00=0$ , to  $01=1$ , to  $10=2$ , to  $11=3$  and then back to  $00=0$ . The first state shall be  $00=0$ , so after the reset the state machine will get entered there. Finally, the output vector is similar to the state vector. With a deeper look onto is: This is therefore in particular a Medvedev machine!

In [figure 15](#) both types of state machines are shown.

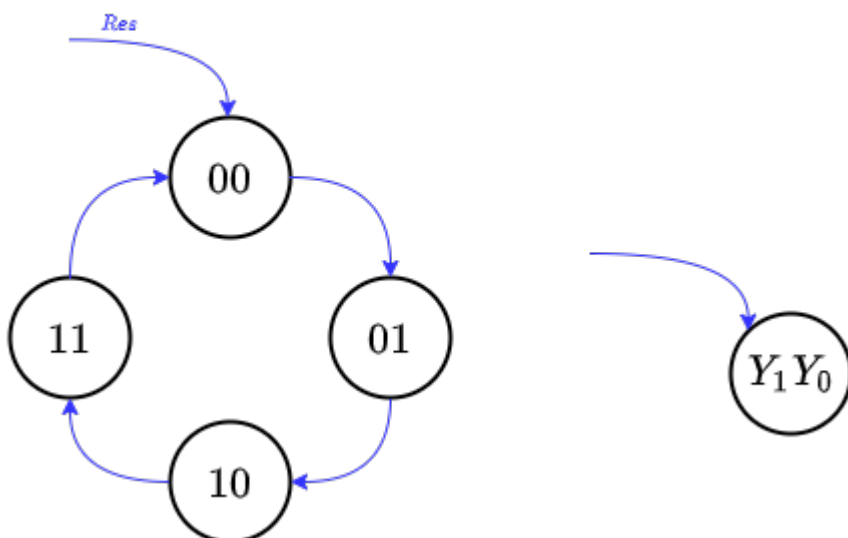
- In the Moore Machine the already seen "halving of the circle" is used to show the distinct state vector in the upper half and the output vector in the lower half.
- In the Medvedev Machine only one value is written in the circle, since here the state vector is also the output vector.

Fig. 15: State Transition Diagram of a up-Counter 0..3 as Moore Machine

**As Moore Machine**



**As Medvedev Machine**



**Exercise 6.3.3.1. State Transition Diagram in Digital.exe**

The shown state transition diagram can easily be created in the tool Digital:

1. Click on the menu Analysis » Finite State Machine
2. Here one either can add new states with a right click, or - more easier - go to the menu Create » Create Counter » 4 States
3. A (up) counter will get created

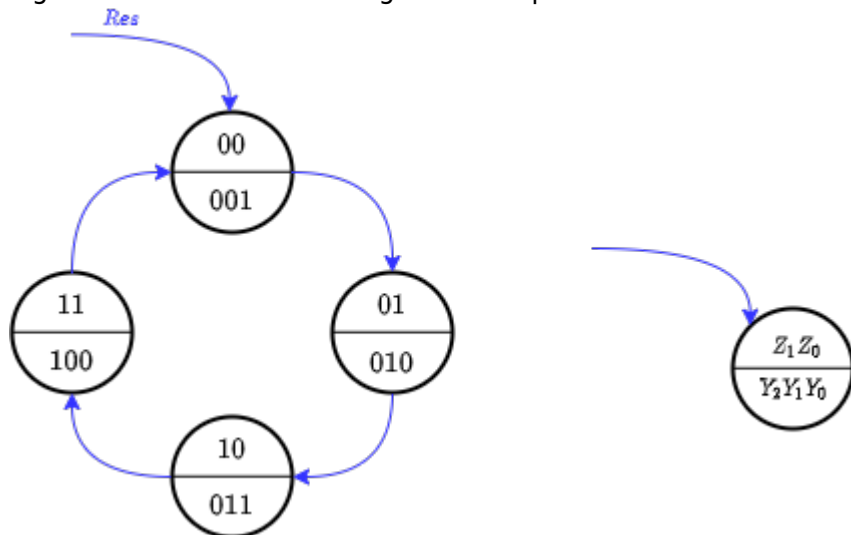
Tasks:

1. Make the state machine get alive:
  1. Click to the menu Create » State Transition Table in order to see the state transition table
  2. Here, click on the menu Create » Cicruit
  3. Now, the circuit can be started via the start button. The present state in the state

transition diagram will also get highlighted.

The next step is to transform this into a up counter from 1...4. For this simply the output has to be changed. This means the numbers in the "lower half of the circles" have to be adapted (see in <imfref pic16>).

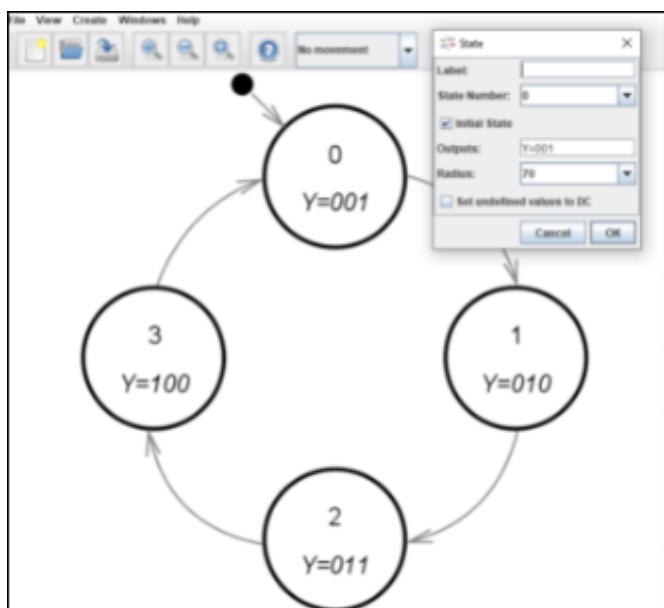
Fig. 16: State Transition Diagram of a up-Counter 1..4 as Moore Machine



### Exercise 6.3.3.2. State Transition Diagram II in Digital.exe

This exercise directly attached onto Exercise 6.3.3.2

The counter shall now be changed in such a way, that it counts 1 - 2 - 3 - 4. For this: right click on each present states beginning with state 0 and add for Outputs \$Y=001\$ and upcounting (see image).



Tasks:

1. Make this state machine again get alive

2. Look at the circuit: Is it a Moore, Mealy or a Medvedev machine?

Looking on what we got, there is one part missing: the state machine does not have any input.. So the input vectors have to be added onto the transition (arrows). For an activateable counter, there only have to be one input  $X$ , which acts as an enable input.

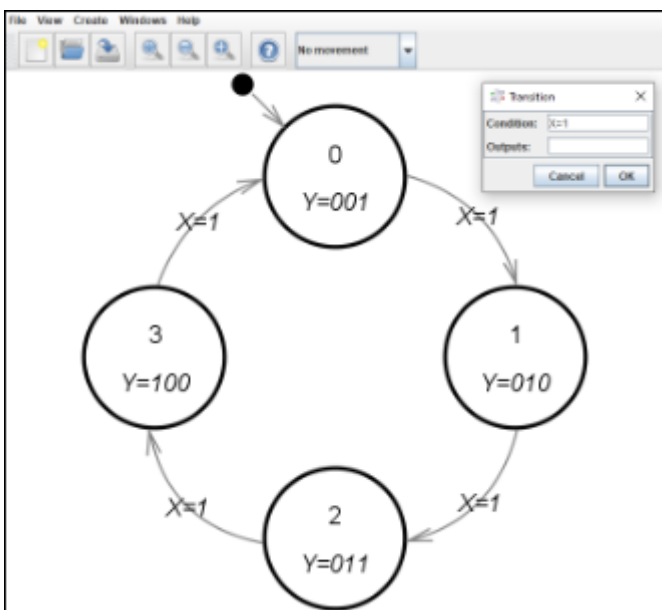
Fig. 17: State Transition Diagram of a up-Counter 1..4 as Moore Machine with enable input



**Exercise 6.3.3.3. State Transition Diagram III in Digital.exe**

This exercise directly attached onto Exercise 6.3.3.2

The last step is to add the transitions: right click on all transitions and add  $X=1$  as a Condition.



Tasks:

1. Make this state machine again get alive

### Exercise 6.3.3.4. State Transition Diagram III in Digital.exe

This exercise directly attached onto Exercise 6.3.3.3

With the state machine in 6.3.3.3 it is also possible to create a state machine which can resemble a traffic light state machine. This shall transit from red, to red-yellow, to green, to yellow and then back to red.

Use the following addition to the resulting circuit in order to get the light running:



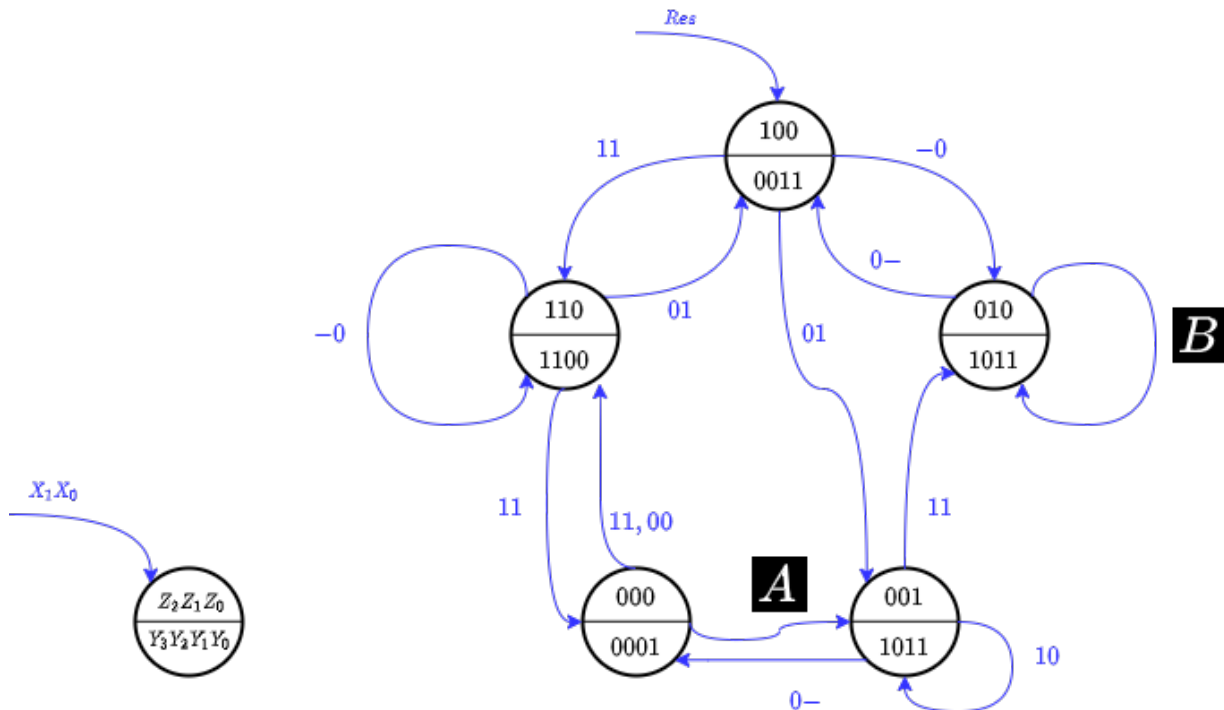
Tasks:

1. Think about the right way to change the outputs in the state machine.
2. Implement the needed correction and test the state machine.

## Exercises

### Exercise 6.1.1. State Transition Diagram I

The following state transition diagram shall be given:



- There are two transitions marked with \$A\$ and \$B\$. What values does the inputs need to have in order show all transitions explicitly?
- How many flipflops are necessary for such a Moore Machine?
- Fill in the missing cells in the following state transition table:

$Z_2(n)$	$Z_1(n)$	$Z_0(n)$	$X_1$	$X_0$	$Z_2(n+1)$	$Z_1(n+1)$	$Z_0(n+1)$	$Y_3(n)$	$Y_2(n)$	$Y_1(n)$	$Y_0(n)$
0	1	0	0	1							
1	1	0	1	1							
0	0	1	1	1							
0	0	1	0	1							

**Exercise 6.1.2. State Transition Diagram II**

Design a state machine, which result in an output of the following (decimal) numbers: \$4 - 5 - 2 - 1 - 6 - 7 - 0 - 3 - 4 - ...\$

- The numbers shall repeat cyclic and without any other input than the clock \$CLK\$
- Draw the state transition diagram and the state transition table
- Use alternatively the structure of a 3bit up-counter
- Draw the structure of this moore machine with the up-counter as a blackbox, the input and output values and - if necessary - further blackboxes
- draw and fill in the additional table.

**Exercise 6.1.3. State Transition Diagram III**

Design a state transition diagram for a automatic milling machine as Moore machine, under

the following conditions:

1. The milling machine has 4 states: Initialisation  $I$ , Component Change  $C$ , Running  $R$ , Error  $E$
2. The milling machine starts at the state  $I$ .
3. Once a limit switch  $L$  is read as activated, the state  $E$  shall be entered, independent which state the machine was in.
4. Only in the state  $E$  an alarm shall ring  $A=1$ .
5.  $E$  can only be exited with a reset.
6. In order to change from Component Change  $C$  to Running  $R$ , the user has to activate a Key  $K=1$ .
7. Only when the machine is running  $R$  and the Fixed Condition is reached  $F=1$ , the state Component Change  $C$  shall be entered.
8. Initialisation  $I$  only changes into Component Change  $C$ , when no limit switch  $L$  is activated and the Key is deactivated (input  $K=0$ ) and the Fixed Condition is reached  $F=1$ .

Explicitely draw all possible transitions.



From:

<https://first.mexle.te.hs-heilbronn.de/> - MEXLE Wiki

Permanent link:

[https://first.mexle.te.hs-heilbronn.de/introduction\\_to\\_digital\\_systems/sequential\\_logic?rev=1640121737](https://first.mexle.te.hs-heilbronn.de/introduction_to_digital_systems/sequential_logic?rev=1640121737)

Last update: 2021/12/21 22:22

