6 Sequential Logic

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

6 Sequential Logic

“I Know What You Did Last Cycle”

6.1 First Terminology

The most important term for the upcoming topics 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 are 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 hard drive) is in a different state compared to a filled one.
- A traffic light showing green has an output distinguishable from red, or yellow.

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

The following terminology is used in the upcoming explanations:

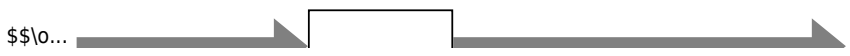
- The **input vector** $\vec{X}$ represents the k inputs $X_0 ... X_{k-1}$.
- The **output vector** $\vec{Y}$ represents the l outputs $Y_0 ... Y_{l-1}$.
- The **state vector** $\vec{Z}$ represents the m inputs $Z_0 ... Z_{m-1}$.
- The sign (n) or n marking the current point in time and therefore e.g. the current state $Z_0(n)$.
- The sign $(n+1)$ or $n+1$ marking the next upcoming 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 machines”.

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 black box in figure 12 was already shown in one practical application in the previous chapter: We saw, that we need the combination of 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? (click onto the 0 on the left)
  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 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 $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow ...$. This is a simple state machine that 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 to count differently, like $ 7, 6, 1, 5$? To understand this, a simpler situation will be investigated. So, let's look at how one can create an 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 facilitates 2 flip-flops to result in 2-bit output. The wanted new state machine needs 3 bits for the output, since the binary representation of our outputs is 001_2,010_2,011_2,100_2.

A simple idea is to take the 2-bit up-counter and add a combinatorial logic in behind it. This logic shall convert the 2-bit up-counter output 00_2 into 001_2, the 01_2 into 010_2, 10_2 into 011_2 and 11_2 into 101_2. This can be logic can be created by:

- writing down the truth table
- putting the values into a Karnaugh map
- extracting the formula with a 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 $\vec{Y}$ depend only on the state values $\vec{Z}$. For this the Moore machine uses two combinatorial circuits:
>
> - The input circuit, which processes the input values $\vec{X}(n+1)$ and the state values $\vec{Z}(n)$ (of the previous step) in such a way that the new states $\vec{Z}(n+1)$ are generated.
> - The output circuit, which transform the state values $\vec{Z}(n)$ into the output values $\vec{Y}(n)$.

The properties of a Moore machine are:

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

6.2.2 Mealy Machine

When looking at figure 5 a bit more in detail, one can see, that the outputs Y_0 and Y_1 just equals the output of the first combinatorial logic circuit. This is not surprising: the input logic circuit shows the $\vec{Z}(n+1)$ 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 $\vec{Y}$ depend not only on the state values $\vec{Z}$. For this, the mealy machine uses two combinatorial circuits:

- The input circuit, which processes the input values $\vec{X}(n+1)$ and the state values $\vec{Z}(n)$ (of the previous step) in such a way that the new states $\vec{Z}(n+1)$ are generated.
- The output circuit, which transforms the state values $\vec{Z}(n)$ and some inputs from ahead of the flip-flops into the output values $\vec{Y}(n)$.

The properties of a mealy machine are:

- The number of flip-flops is only given by the number of states m.
- The output **not** only changes when an edge on the clock input happens: It also depends on the input $\vec{X}(n)$. The mealy machine is an **asynchronous state machine**.
- The mealy machine usually needs fewer logic gates.
- The mealy machine must be designed properly 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 $\vec{Y}$ are directly given by the state values $\vec{Z}$. For this, the Medvedev machine uses only one combinatorial circuit. This circuit process the input values $\vec{X}(n+1)$ and the state values $\vec{Z}(n)$ (of the previous step) in such a way that the new states $\vec{Z}(n+1)$ and the output values $\vec{Y}(n+1)= \vec{Z}(n+1)$ are generated.

The properties of a Medvedev machine are:

- The number of flip-flops is given by the 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 needs more logic gates.

The figure 8 shows the principle differences in the architecture of the state machines.

Fig. 8: States of Water

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 behavior. 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 transitions. 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 Germany, often one has to pay for entering the toilet. An example of such an entrance control system is shown in figure 11. In 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

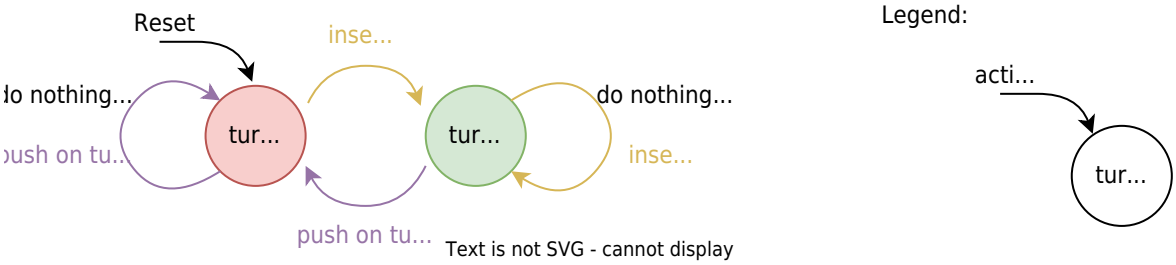
The figure 12 the state transition diagram is drawn.

- The two states are that (1) the turnstile is opened and one can 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 take 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 on 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: "non-binary" State Transition Table 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

Text is not SVG - cannot display

the inputs, outputs, and states have to be encoded into binary, 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 Table of the Entrance Control for Toilets

Interestingly, the logic circuit for this state transition table was already part of the course: it is the RS flip-flop! When looking deeper into 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 truth table of the RS flip flop.

6.3.3 First Adaption for Up-Counter

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

The answer is quite simple: there are 4 states, so 4 “circles” are needed. These states need to have circular connections to get the wanted increase from $00_2=0_{10}$, to $01_2=1_{10}$, to $10_2=2_{10}$, to $11_2=3_{10}$ and then back to $00_2=0_{10}$. The first state shall be $00_2=0_{10}$, 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 at it: 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 an up-counter 0..3 as Moore Machine

As Moore 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 - 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` » `Circuit`
    3. Now, the circuit can be started via the start button. The present state in the state transition diagram will also get highlighted.

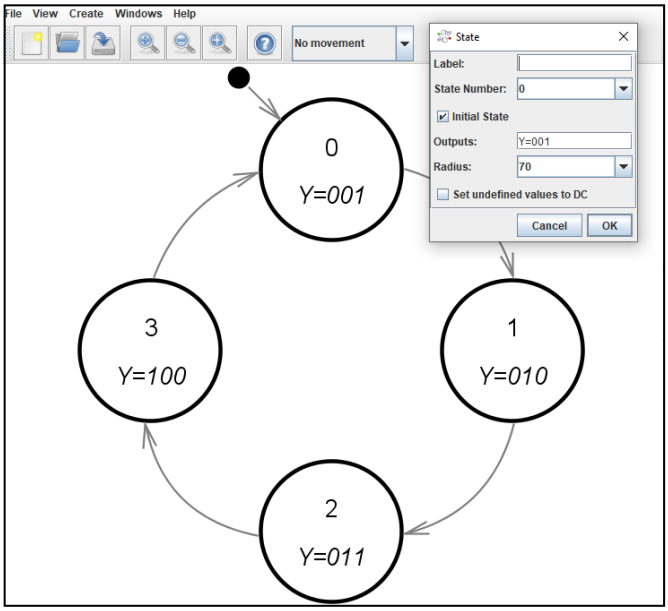
The next step is to transform this into an up counter from $1...4$. For this simply the output has to be changed. This means the numbers in the “lower half of the circles” have to be adapted (see in <imfref pic16>).

Fig. 16: State Transition Diagram of an up-counter 1..4 as Moore Machine

Exercise 6.3.3.2. State Transition Diagram II in Digital.exe

This exercise is directly attached to 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 state beginning with state 0 and add for `Outputs` $Y=001$ and up-counting (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?

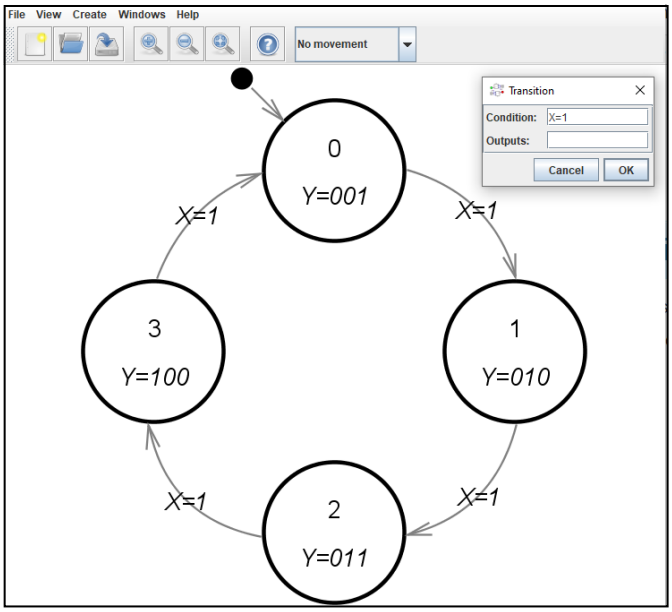
Looking at what we got, there is one part missing: the state machine does not have any input... So the input vectors have to be added to the transition (arrows). For an activatable counter, there only has to be one input X, which acts as an enable input.

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

Exercise 6.3.3.3. State Transition Diagram III in Digital.exe

This exercise is directly attached to 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:

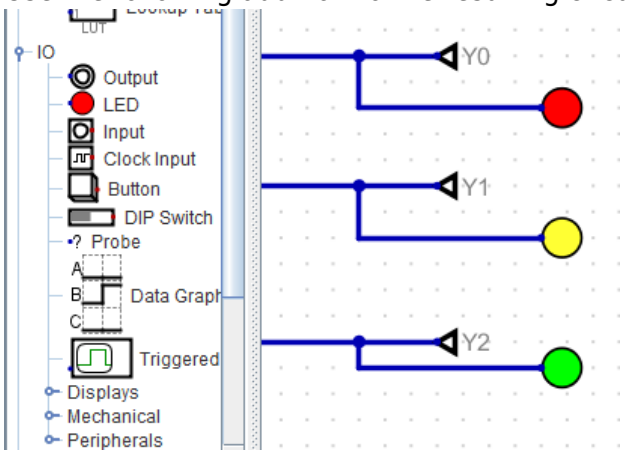
1. Make this state machine again get alive

Exercise 6.3.3.4. State Transition Diagram III in Digital.exe

This exercise is directly attached to exercise 6.3.3.3

With the state machine in 6.3.3.3, it is also possible to create a state machine that 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 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 - fill the gaps

The following state transition diagram shall be given:

- There are two transitions marked with A and B.
  What values do the inputs need to have to show all transitions explicitly?

  Strategy

  - Find the transitions wanted
  - Look at which state these transitions start.
  - Which other transitions start there?
  - Which transition conditions are missing?

  Solution

  - transition A
    - Starts at state 000
    - Also transition with 11, 00 starts here
    - 01, 10 are missing
  - transition B
    - Starts at state 010
    - Also transition with $0-$ starts here
    - $1-$ are missing

  Result

  - A: 01, 10
  - B: $1-$

- How many flip-flops are necessary for such a Moore Machine?

  Strategy

  Each flipflop can store one Bit. Each stored bit can be used to address states. So, check the number of bits i of states Z_i (“size of the state vector”).
  Be aware, that one bit can address a maximum of 2 states, two bits a maximum of 4 states, three bits a maximum of 8 states, and so on.

  Solution

  - Number of bits i of states Z_i is given in the legend. The number of bits i has also to fit the number of states.
  - Here the legend shows Z_2, Z_1, Z_0, so there are 3 bits.
  - Also the number of states in the diagram is 5. This can only be numbered with at least 3 bits.

  Result

  3

- 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							

  Strategy

  Check for each line:

  - What is the start/present state (given by the bits $\color{red}{Z_2(n), Z_1(n), Z_0(n)}$ in the line)?
  - Which transition is shown as start/present state (given by $\color{brown}{X_1, X_0}$)?

  Out of this orientation, the next columns can be derived:

  - At the end of the transition, the next state can be found (necessary for columns $\color{green}{Z_2(n+1), Z_1(n+1), Z_0(n+1)}$)

- Within the state symbol of the present state, the output values can be found (necessary for columns $\color{violet}{Y_2(n), Y_1(n), Y_0(n)}$)

Result

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

Exercise 6.1.2. State Transition Diagram II - Sequence

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

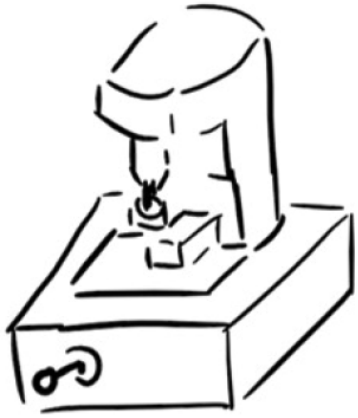
- The numbers shall repeat cyclic and without any other input than the clock CLK
- Draw the state transition diagram and the state transition table
- Use alternatively the structure of a 3bit up-counter
- Draw the structure of this Moore machine with the up-counter as a black box, the input and output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

Exercise 6.1.3. State Transition Diagram III - Milling Machine

Design a state transition diagram for an automatic milling machine as a Moore machine, under the following conditions:

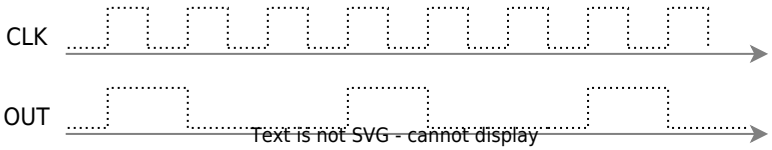
1. The milling machine has 4 states: Initialisation I, Component Change C, Running R, Error E
2. The milling machine starts at the state I.
3. Once a limit switch L is read as activated, the state E shall be entered, independent of which state the machine was in.
4. Only in the state E an alarm shall 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 $F=1$.

Explicitly draw all possible transitions.



Exercise 6.1.4. State Transition Diagram IV - Find a sequence

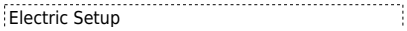
Develop a sequential circuit, which creates the following output



- Draw the state transition diagram of the Moore machine of the synchronous sequential circuit.
- Create the digital circuit.

Exercise 6.1.5. State Transition Diagram V - LED Fun

Develop a sequential circuit, which allows driving the following LED sequence



- Draw the state transition diagram of the Moore machine of the synchronous sequential circuit.
- Create the digital circuit.

Exercise 6.1.6. State Transition Diagram VI - Roll the dice

Develop a sequential circuit, which generates a clock-driven up-counter from $1..6$. The not required states shall lead after one clock cycle to the state of number 1.

- Draw the state transition diagram of the Moore machine of the synchronous sequential circuit.
- Create the digital circuit.