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 State Diagram, State Transition Diagram

6.1.1 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 transistions are relevant, the states are simplified to a circle, showing the state name and behaviour. The transitions are depict as arrows, where the needed condititon is written onto (See figure 9 image (2)). This diagram is called **state transition diagram**.

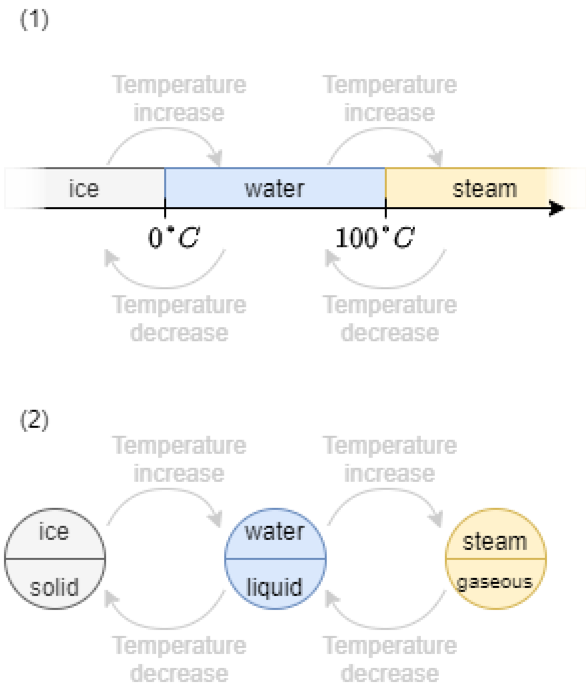


Fig. 9: States of Water

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)).

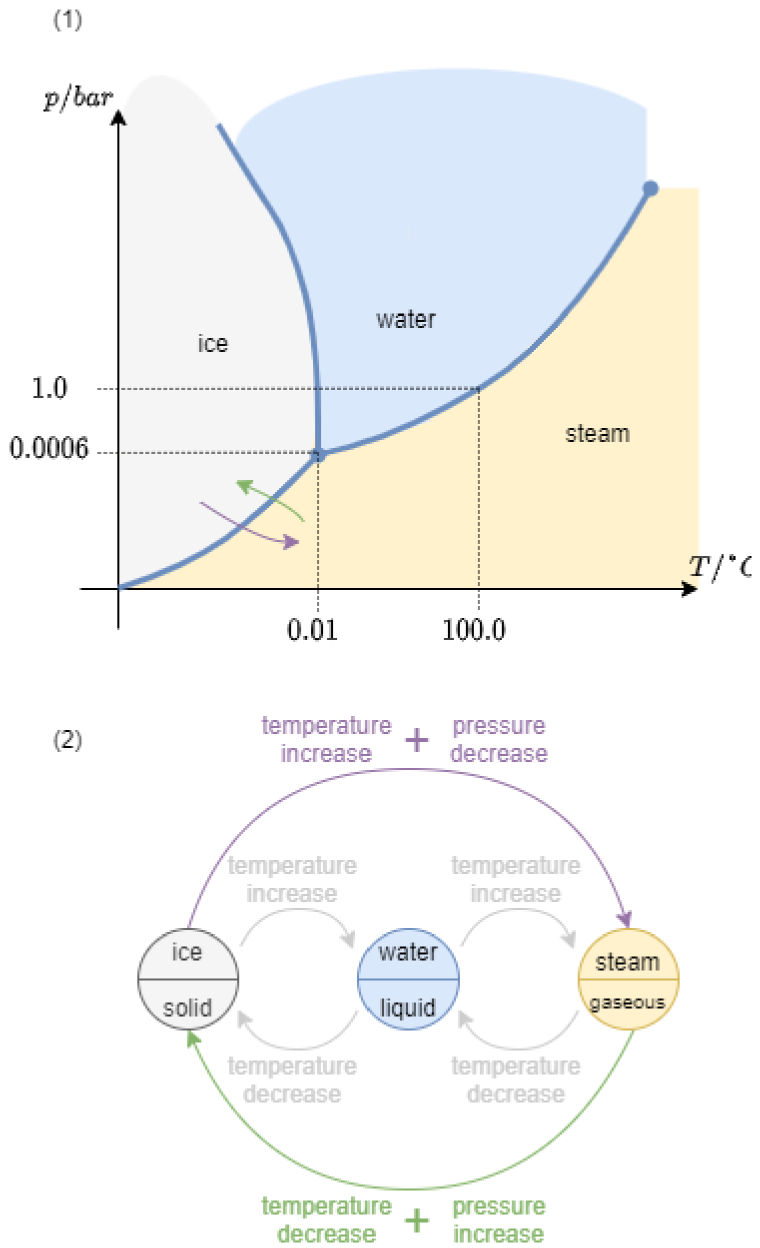


Fig. 10: States of Water

6.1.2 Simple logic Example

In German, often one has to pay for entering the toilet. An example of such a 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

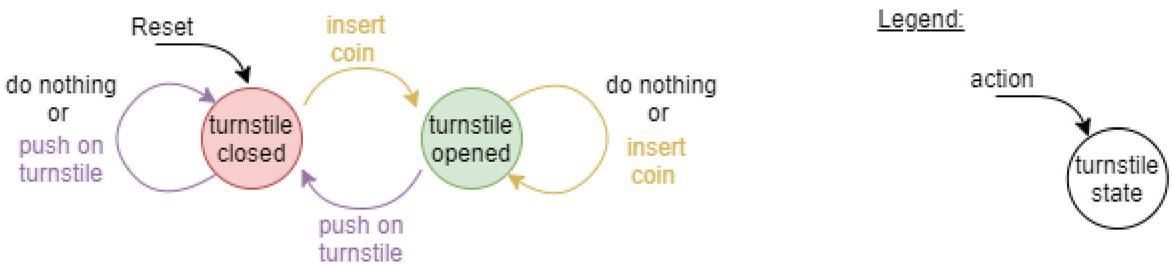
The figure 12 the state transition diagram is drawn.

- 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 Diagramm 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 Diagramm 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

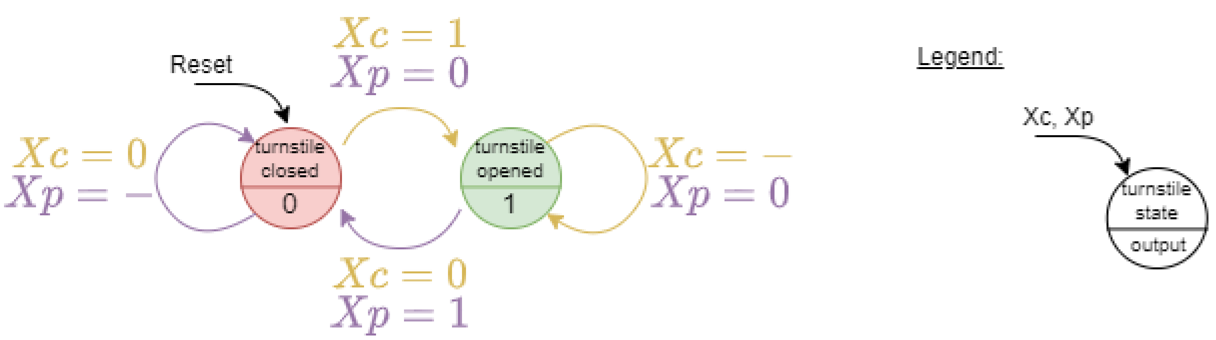
the inputs, outputs and states have to be encoded into binary, in order to investigate this table a bit more. How the binary value is connected to the outputs does not matter. We will choose the following coding:

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

This table is shown in figure 1 and is called **state transition table**.

Fig. 1: State Transition Diagramm 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 1 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.

6.1.3 First Terminology

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 $X_0 ... X_{k-1}$
- The **output vector** $\vec{Y}$ represents the l outputs $Y_0 ... Y_{l-1}$
- The **state vector** $\vec{Q}$ represents the m inputs $X_0 ... X_{m-1}$
- The sign (n) or n marking the current point in time and therefore e.g. the current state $Q_0(n)$
- The sign $(n+1)$ or $n+1$ marking the next upcomming point in time and therefore e.g. the next state $Q_0(n+1)$
- 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).

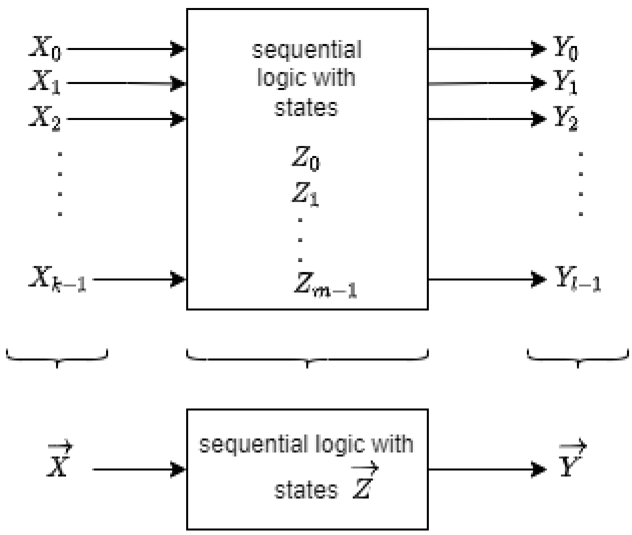
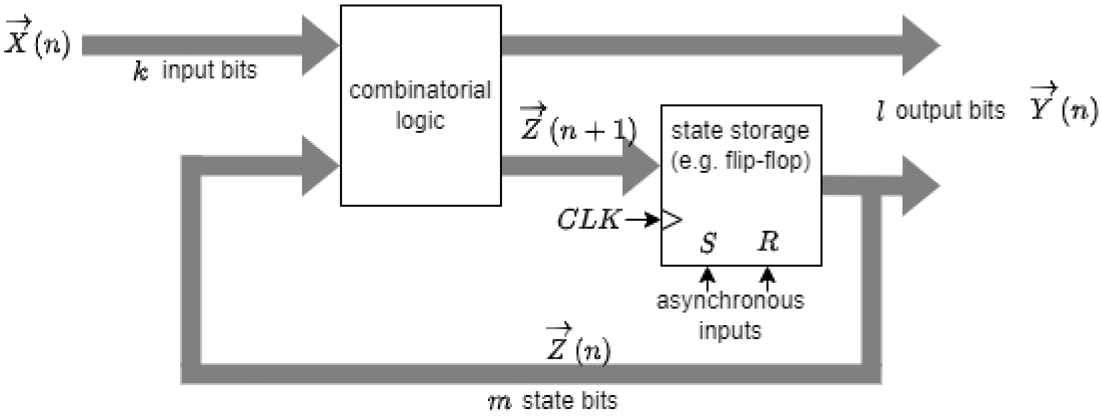


Fig. 1: Abstract View onto a Sequential Logic

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 might the state vector $\vec{Q}$ need?

Fig. 3: Example for a sequential logic

One simple example of a sequential logic is shown in figure 4. There, the combnatorial logic is explicitely 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 which will be used in the net chapters

Fig. 4: State machine of an Up-Counter

6.1.4 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 simplier situation will be investigated. So, let's look how one can create a up-counter counting $1, 2, 3, 4$.

Moore Machine

The first idea might be to use what we already have: an up-counter, which faciliate 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 $\vec{Q}$. For this the moore machine uses two combinatorial circuits:

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

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.

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 $\vec{Q}(n+1)$ 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 $\vec{Q}$. For this the mealy machine uses two combinatorial circuits:

- The input circuit, which process the input values $\vec{X}(n+1)$ and the

state values $\vec{Q}(n)$ (of the previous step) in such a way that the new states $\vec{Q}(n+1)$ are generated.
- The output circuit, which transform the state values $\vec{Q}(n)$ 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 number of states m.
- The output **not** only changes when an edge on the clock input happen: It is also dependent on the input $\vec{X}(n)$. 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 6.1.4.1 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?

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 $\vec{Q}$. For this the medvedev machine uses only one combinatorial circuit. This circuitprocess the input values $\vec{X}(n+1)$ and the state values $\vec{Q}(n)$ (of the previous step) in such a way that the new states $\vec{Q}(n+1)$ and the output values $\vec{Y}(n+1)= \vec{Q}(n+1)$ 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.

From:
https://wiki.mexle.te.hs-heilbronn.de/ - **MEXLE Wiki**

Permanent link:
https://wiki.mexle.te.hs-heilbronn.de/introduction_to_digital_systems/sequential_logic?rev=1638846149

Last update: **2021/12/07 04:02**