IN3170/4170, Spring 2023

Philipp Häfliger
hafliger@ifi.uio.no

Excerpt of formar INF2270 script Chapter 6: Sequential Logic Circuits

Content

Combinational Logic

Sequential Logic

Finite State Machines

Content

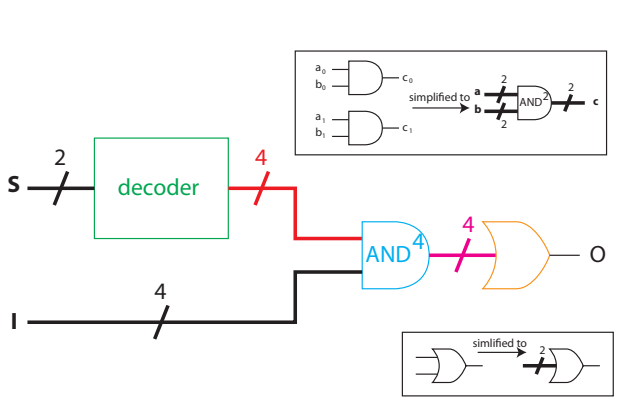
Combinational Logic

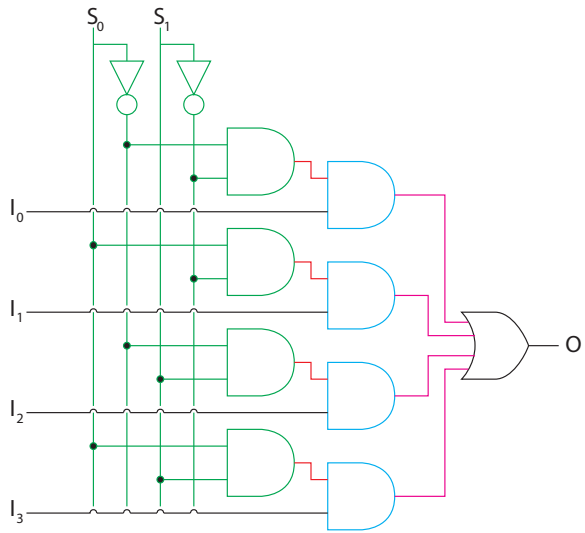
Sequential Logic

Finite State Machines

On Circuit Notation in INF2270 Script

2 bit Multiplexer ‘spelled out’





Combinational Logic

Combinational logic circuits are feed-forward logic/digital circuits with no memory that can be described by Boolean functions.

Note what is implied here: logic gates can *also* be connected in ways that include feed-back connections that implement/include *memory* that *cannot* be described as Boolean functions! This is then not 'combinational logic', but 'sequential logic', of which we will talk later.

Truth Table

Combinational logic circuits can be described by logic gate circuit schematics, Boolean functions, and truth tables. A truth table lists the input/output relationship of a combinational logic circuit.

Example:

a	b	$\overline{a \wedge b}$
0	0	1
0	1	1
1	0	1
1	1	0

a	b	$\overline{a \wedge b}$
0	X	1
X	0	1
1	1	0

Note: while different circuit schematics and Boolean expressions can describe the *same* function, there is only ONE complete i.e. listing ALL possible inputs with their output(s)) truth table for one specific Boolean function. So complete truth tables are one way to proof equivalence of different schematics and Boolean expressions.

Design and Analysis of Digital Circuits

Design of a digital circuit is the process of assembling circuit blocks to form a bigger digital circuit.

Analysis of a digital circuit is the process of finding out what it is doing, e.g. (in the case of combinational logic!) by finding an equivalent Boolean function or a complete truth table.

A complete analysis of a digital circuit is quite trivial for small digital circuits but neigh impossible for circuits of the complexity of a modern CPU. Hierarchical approaches in design and analysis provide some help.

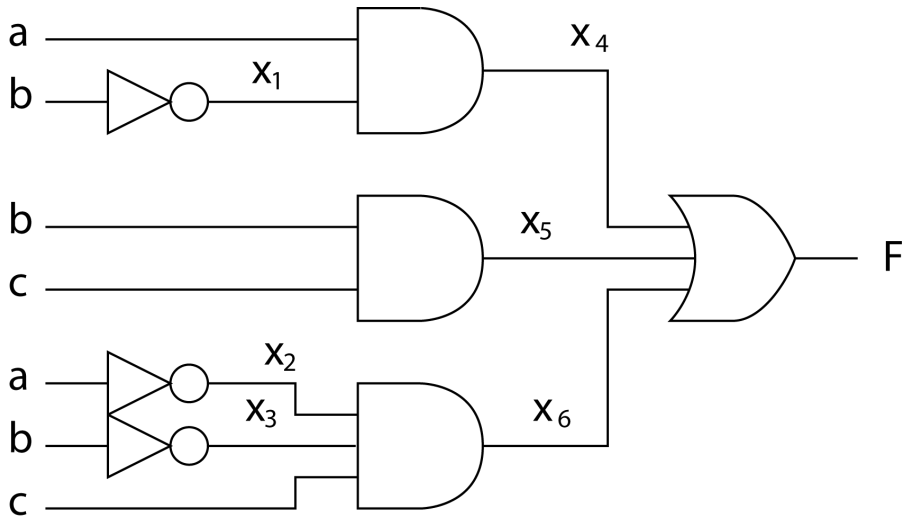
The first Pentium on the market had a mistake in its floating point unit.

> *After the Intel 286 there was the 386 and then the 486, but the 585.764529 was then dubbed 'Pentium' for simplicity sake.*

Logic Gate Signals

A certain range of input voltage is defined as 'high' or logic '1' and another range is defined as 'low' or '0'. Note: the range is usually defined bigger/with more tolerance at a logic gate *input*, making a sequence of gates robust against a certain level of noise.
E.g. in a digital circuit with a 1.8V supply one can, for instance, guarantee an input voltage of 0V to 0.5V to be recognised as '0' and 1.2V to 1.8V as '1' by a logic gate.
On the output side the gate can guarantee to deliver a voltage of either $>$1.75V or $<$0.05V.
The assymetric ranges correct 'errors' in a cascade of gates, adding a safety margin between input and output, makeing (correctly designed!) digital circuits very robust (which is necessary with millions of logic gates in a CPU, where a single error might impair the global function!)

Analysis Example



$$\equiv \underbrace{a \wedge \bar{b}}_{x_4} \vee \underbrace{b \wedge c}_{x_5} \vee \underbrace{\bar{a} \wedge \bar{b} \wedge c}_{x_6}$$

Standard Combinational Logic

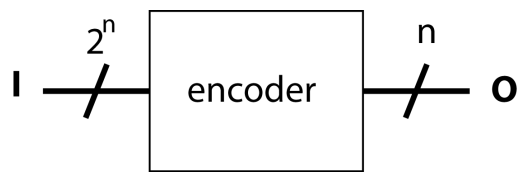
Some combinational logic (and of course also sequential logic → later) is often used in computational devices and are usually provided as 'black boxes' guaranteeing a defined function.
Examples:

- encoder/decoder
- multiplexer/demultiplexer
- adders/multipliers

There are actually variations on how those functions are implemented, resulting in different processing speeds and/or power consumption and/or scalability (i.e. how easy it is to adapt the same function for more inputs).

3-bit Encoder Specification

An encoder in digital electronics refers to a circuit that converts 2^n inputs (one-hot code) into n outputs (binary number), as specified by the following truth table.

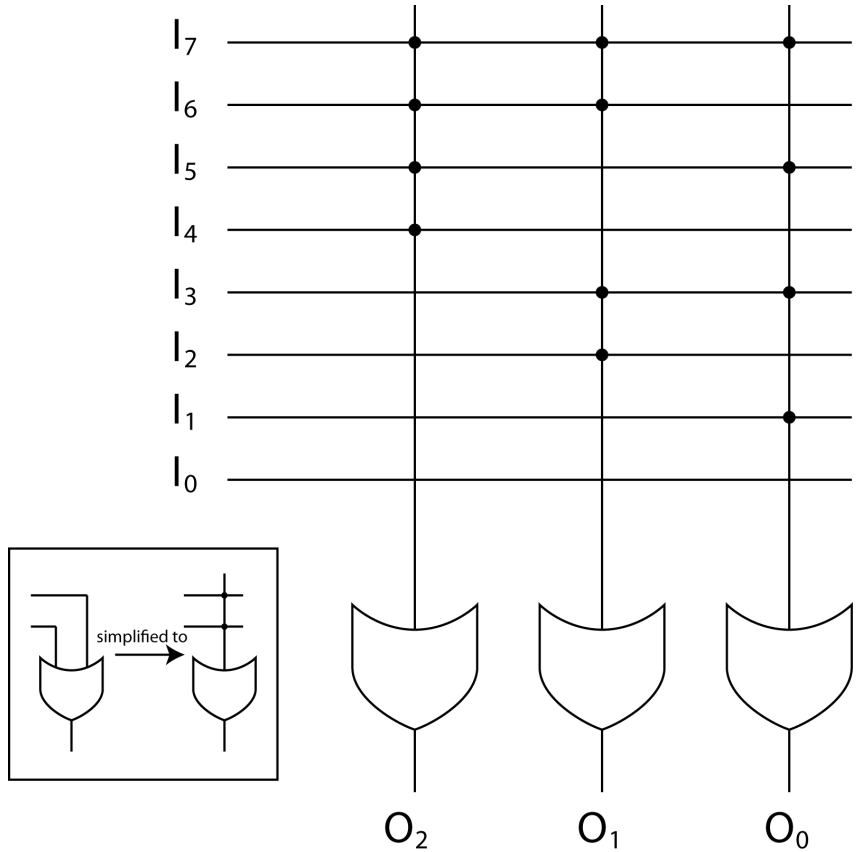


Simple 3-bit Encoder Truth Table

I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	O_2	O_1	O_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

NOTE: this table is not complete!

3-bit Encoder Implementation Variant



3-bit Encoder Remarks

The truth table that was given is not complete: some inputs are 'illegal' (not one-hot codes). Circuitry that produces the input should ensure to only produce legal states. In our specific digital circuit implementation we can deduct what the output in each illegal case would be, but other implementation may provide different outputs in those non-defined cases, and still be valid encoders!

The following truth table is a deterministic specification of an encoder, without 'illegal' inputs, where the 'highest' active input bit determines the output. 'X' in the table means 'do not care', or 'for any state' and allows to abbreviate the truth table. This would require a different implementation, but we will not present it here.

3-bit Priority Encoder Truth Table

I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	O_2	O_1	O_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

NOTE: this table IS complete.

Content

Combinational Logic

Sequential Logic

Finite State Machines

Flip-Flop

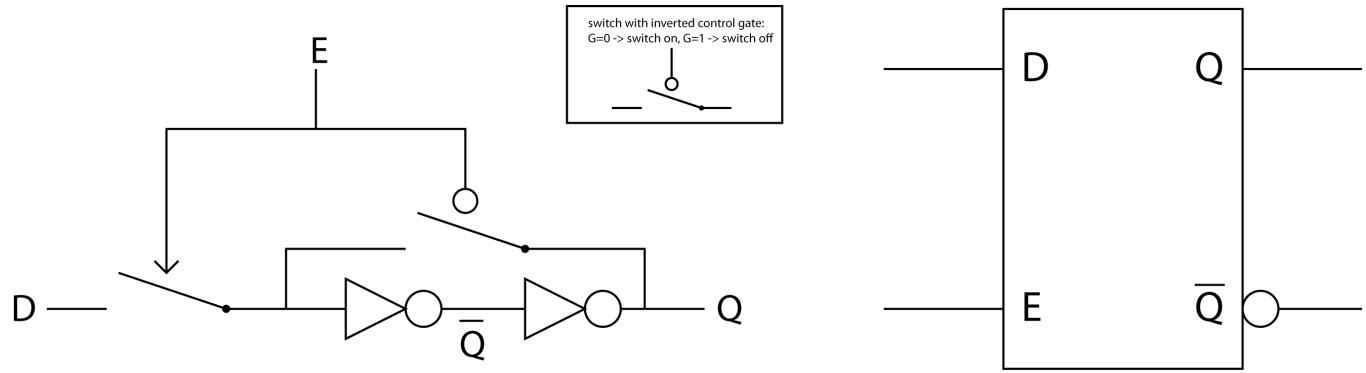
Flip-flops are digital circuits with two stable states that are used as storage elements for 1 bit. The term 'flip-flop' is in the more recent use of the language more specifically used for synchronous binary memory cells (e.g. D-flip-flop, JK-flip-flop, T-flip-flop), whereas the term 'latch' (e.g. SR-latch, D-latch) is used for the simpler more basic asynchronous 'transparent' storage elements, but this is not necessarily consequently applied throughout the literature.

Characteristic Table and Equation

The behaviour of a sequential logic circuit can be expressed with a *characteristic table*: a truth table expressing the relation between the input and the present state, and the next state. An alternative is the *characteristic equation* which defines the dependency of the next state on the input and present state as a *recursive* Boolean function. Examples follow on the next slides.

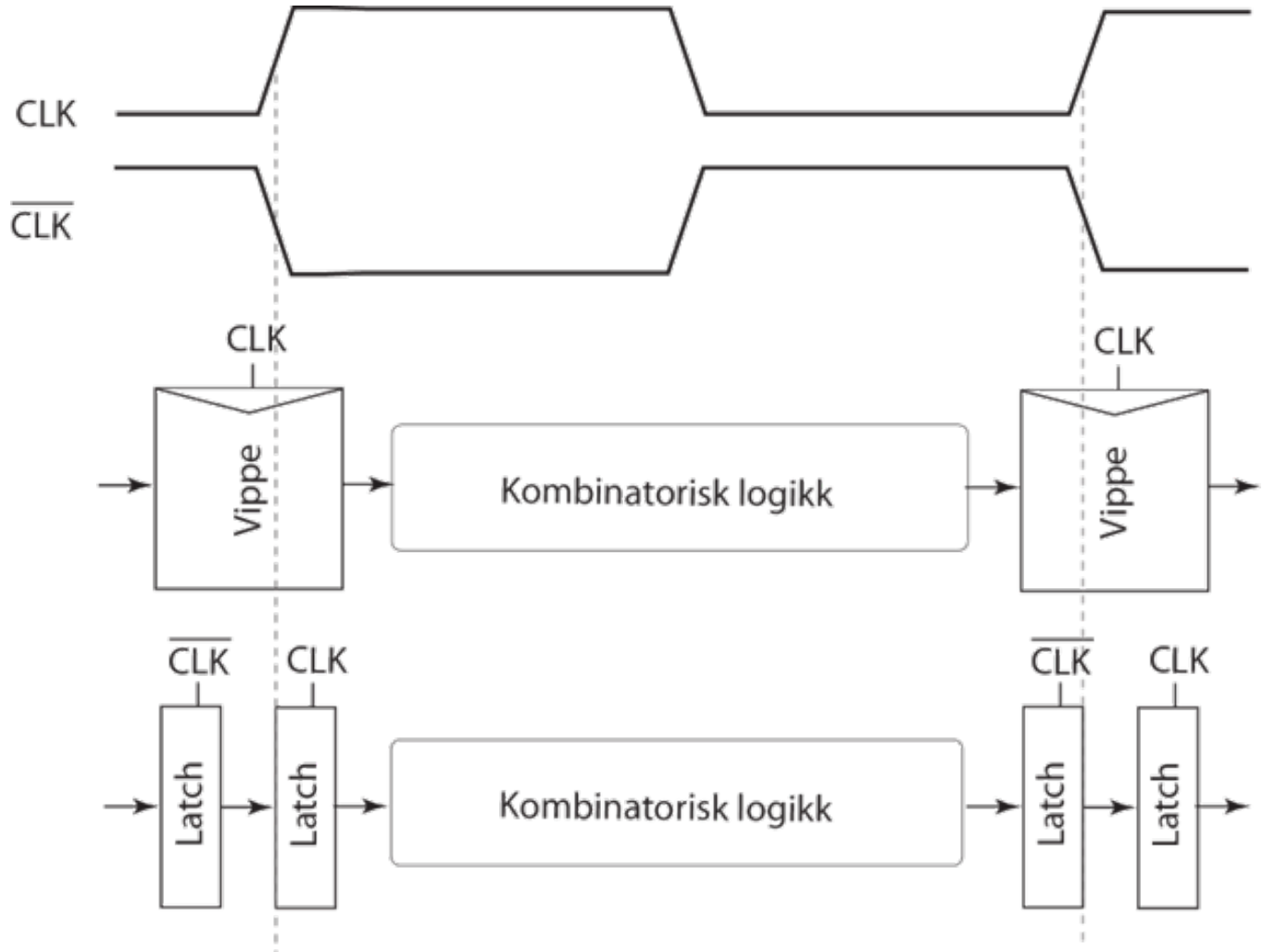
Flip-Flop Principle: Positive Feedback

The simplest Flip Flop: Gated D-Latch/Transparent Latch



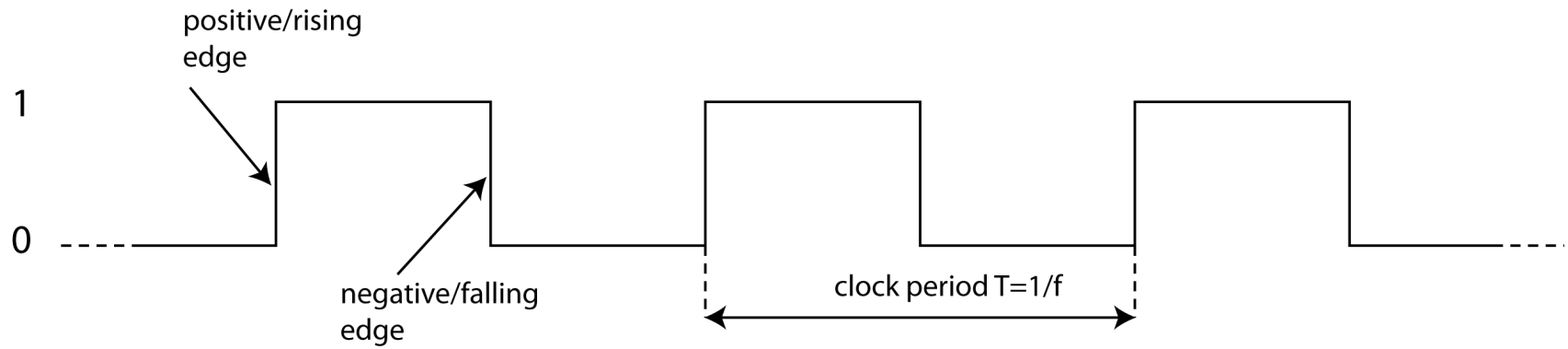
Characteristic Equation:
$Q = D \wedge E \vee \bar{E} \wedge Q$
When the feedback loop is connected the state is maintained
This is an example of an ***asynchronous*** characteristic equation simply describing a logic circuit with a feedback loop. Input signals may change at any time.
NOTE: asynchronous logic can (un)intentionally turn into an oscillator. Consider for example the characteristic equation $A = \overline{A}$

D-Flip-Flop from D-Latch

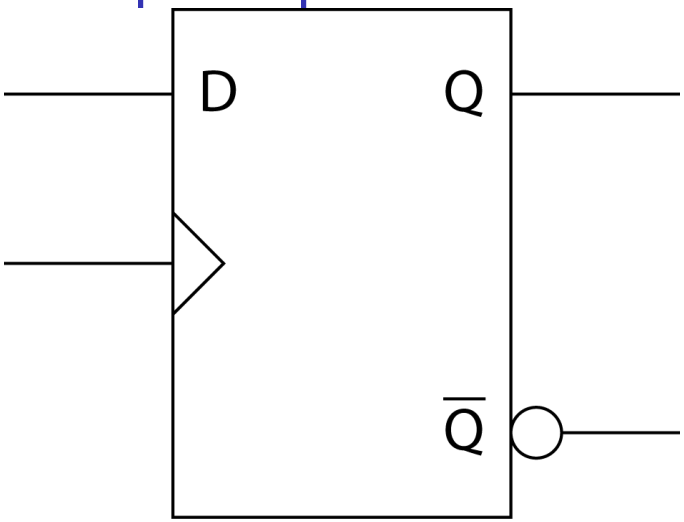


D-flip-flops used for *synchronous* sequential logic: their content only changes with a clock signal flank.

Clock Signal



D-Flip-Flop



Characteristic Equation:
$Q = D$
This is an example of an ***synchronous*** characteristic equation.
The CLK signal is not shown but implicit. Signals are time discreet, only changing with the clock flank and assumed stable before the next.

Characteristic Table

D	Q_t	Q_{t+1}
0	0	0
0	1	0
1	0	1
1	1	1

Abbreviated:

D	Q_t	Q_{t+1}
0	X	0
1	X	1

or simply:

D	Q_{t+1}
0	0
1	1

Content

Combinational Logic

Sequential Logic

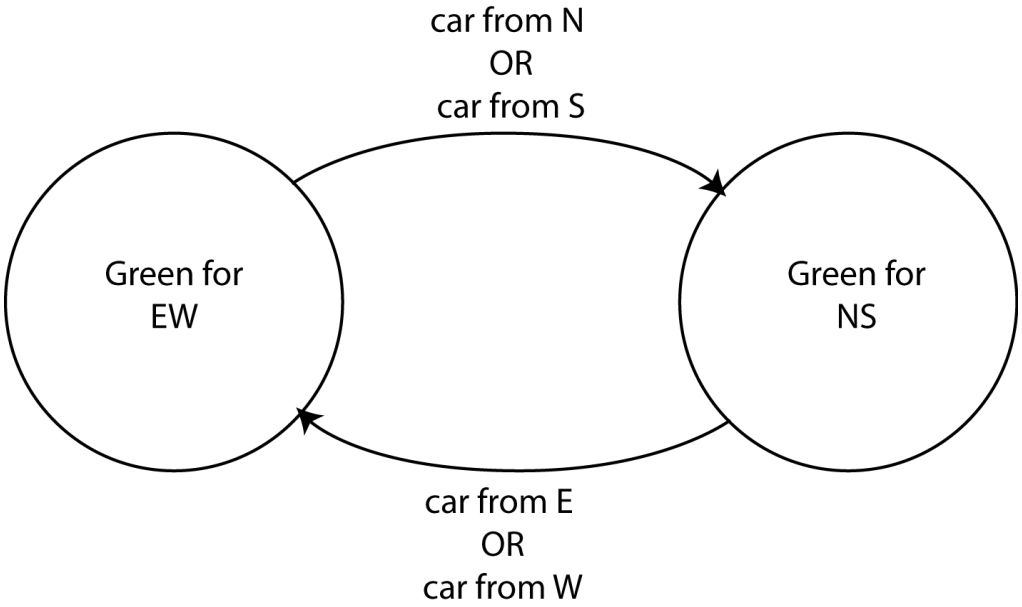
Finite State Machines

Finite State Machines

Finite State Machines (FSM) are a formal model suited to describe sequential logic, i.e. logic circuits who's output does not only depend on the present input but on internal memory and thus, on the history or sequence of the inputs.

Specifying a FSM with a state transition graph

Example: Traffic Light

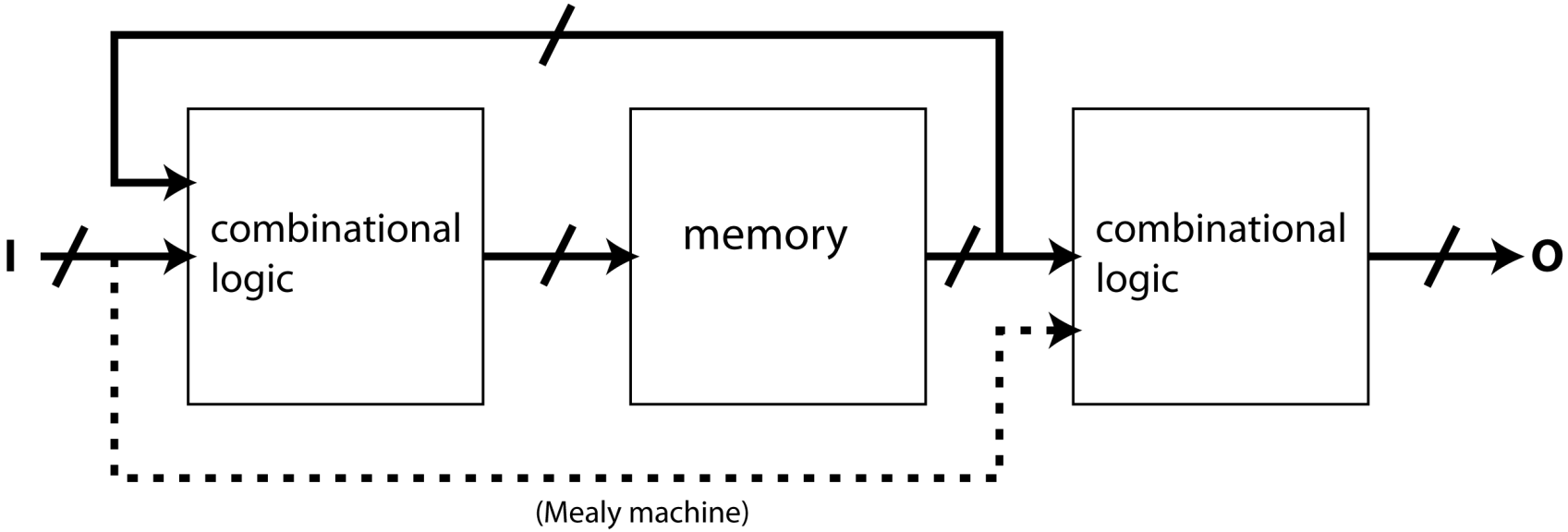


Note: this is a synchronous FSM: the clock signal is not explicitly shown, but implicitly all state transitions happen only with a specific clock flank. (Thus, you cannot end up with an asynchronous oscillator.)

Synchronous and Asynchronous FSM

- *Synchronous* FSMs: transition of a global *clock* signal as implicit transition condition for all state changes. Realized as sequential logic circuits with synchronous flip-flops, e.g. D-flip-flops. Generally simpler to implement and easier to verify and test. The clock frequency needs to be slow enough to allow the slowest combinational transition condition to be computed.
- *Asynchronous* FSMs change state at once if the explicit transition condition is met (buzzword: 'ripple'). They can be very fast but are much harder to design and verify.

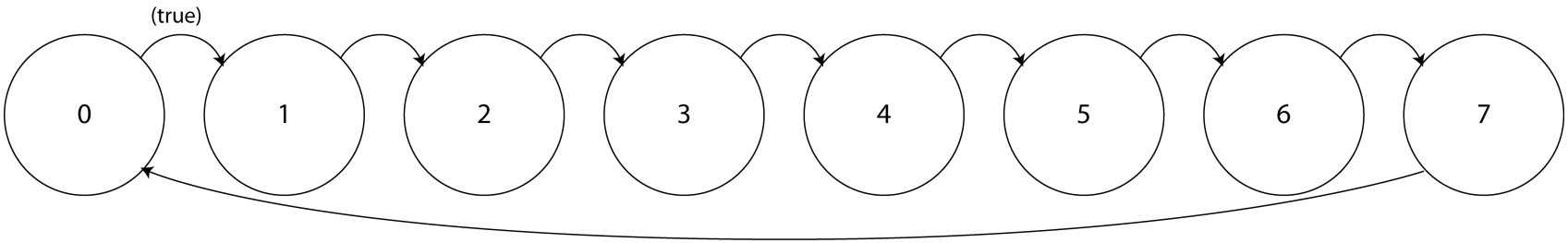
Moore and Mealy Machine



In a *Moore* FSM, the output depends solely on the state/memory, in contrast to the *Mealy model*, where outputs also depend on the inputs directly. The later sometimes allows to reduce the number of states, but is more demanding to design.

Deducing a Synchronous 3-bit Counter from a FSM

Counters are a frequently used building block in digital electronics. A counter increases a binary number with each clock edge.



For the design of this simple counter variant, state transitions are unconditional, i.e. they happen in every case at every positive clock edge.

State Transition Table

We chose to represent the 8 states in D-flip-flops with the corresponding binary numbers. Thus, there is in a first instance no output combinational logic necessary. (Note that output combinational logic will be necessary if the numbers should, for example, appear on an LED-display)

present			in	next		
S_2	S_1	S_0	NA	S_2	S_1	S_0
0	0	0		0	0	1
0	0	1		0	1	0
0	1	0		0	1	1
0	1	1		1	0	0
1	0	0		1	0	1
1	0	1		1	1	0
1	1	0		1	1	1
1	1	1		0	0	0

Karnaugh Maps

$S_{2_{next}}$

S_2S_1 \ S_0	0	1
00	0	0
01	0	1
11	1	0
10	1	1

$S_{1_{next}}$

S_2S_1 \ S_0	0	1
00	0	1
01	1	0
11	1	0
10	0	1

$S_{0_{next}}$

S_2S_1 \ S_0	0	1
00	1	0
01	1	0
11	1	0
10	1	0

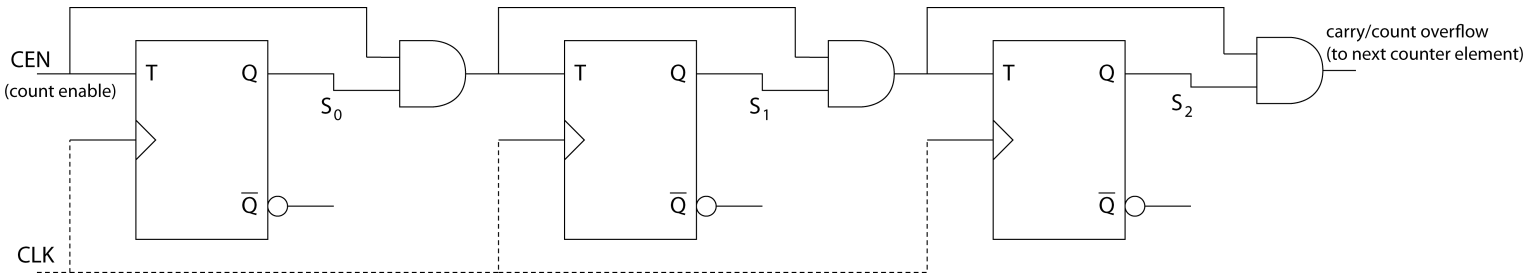
$$S_{2_{next}} = (S_2 \wedge \overline{S_1}) \vee (S_2 \wedge \overline{S_0}) \vee (\overline{S_2} \wedge S_1 \wedge S_0)$$
$$= S_2 \wedge (\overline{S_1} \vee \overline{S_0}) \vee (\overline{S_2} \wedge (S_1 \wedge S_0))$$
$$= S_2 \wedge \overline{(S_1 \wedge S_0)} \vee \overline{S_2} \wedge (S_1 \wedge S_0)$$
$$= S_2 \oplus (S_1 \wedge S_0)$$

General Synchronous Counter Element

$$S_{n_{next}} = S_n \oplus \left(\bigwedge_{k=0}^{n-1} S_k \right) \tag{1}$$

$\Rightarrow$ T-flip-flop where $T = \bigwedge_{k=0}^{n-1} S_k$
(Compare characteristic equation of the T-flip-flop!)

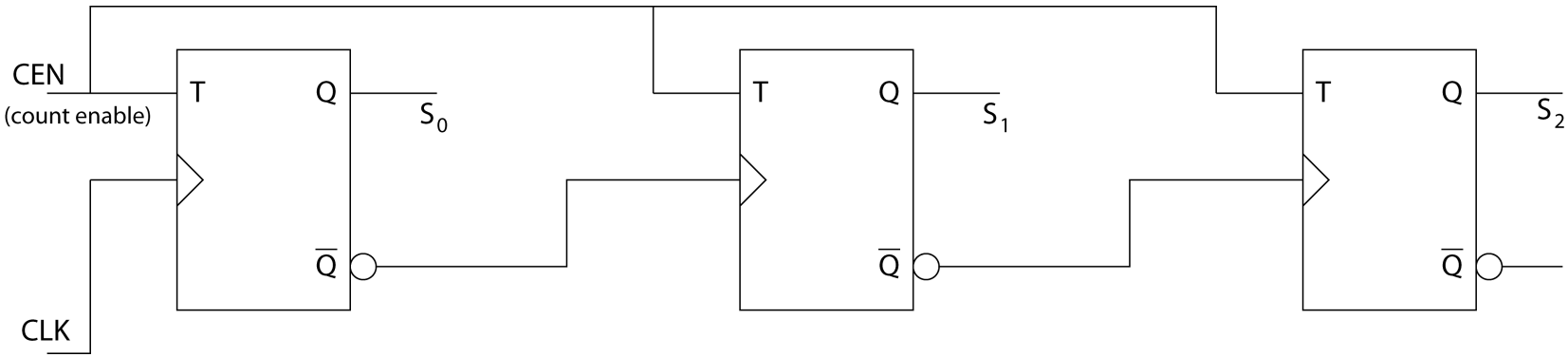
3 bit Synchronous Counter



Useful extensions:

- Possibility for loading an initial number (control signal LD and an input bus)
- Reset to zero (control signal RES)
- Switching between up and down counting (control signal UpDown)

3 bit Ripple Counter



A simple and popular asynchronous variant (only the first T-flip-flop is clocked with the global clock) of a counter. A possible disadvantage is that the output signal 'ripples' from the lowest to the highest bit, i.e. the highest bits are updated with a delay. This must be taken into account, if this kind of counter is used.