# INF2270 — Spring 2011

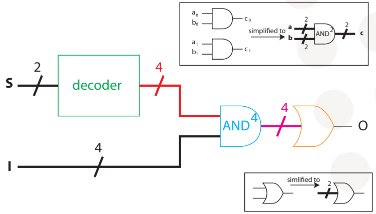
Lecture 3: Sequential Logic

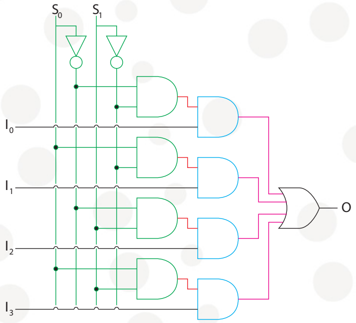
UNIVERSITETET I OSLO

## content

## content

## 2 bit Multiplexer ‘spelled out’





## content

- Brief Repetition on Circuit Notation
- Flip-Flops
- Finite State Machines
- Selected Sequential Logic
  - Counters
  - Shift Registers
- Binary Addition

## Flip-Flop

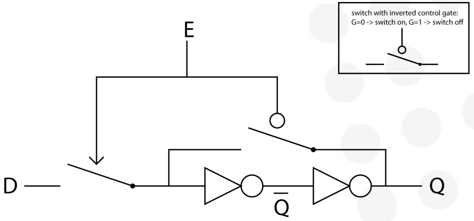
*Flip-flops* are digital circuits with two stable states that are used as storage elements for 1 bit. The term 'flip-flop' is in the more recent use of the language more specifically used for synchronous binary memory cells (e.g. D-flip-flop, JK-flip-flop, T-flip-flop), whereas the term 'latch' (e.g.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

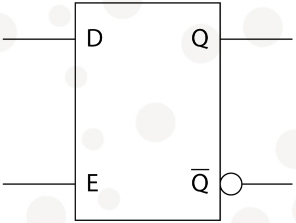
## Characteristic Table and Equation

The behaviour of a flip-flop 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 Boolean expression. Examples follow on the next slides.

## Flip-Flop Principle: Positive Feedback

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


switch with inverted control gate:
G=0 -> switch on, G=1 -> switch off
E
D
$\overline{Q}$
Q

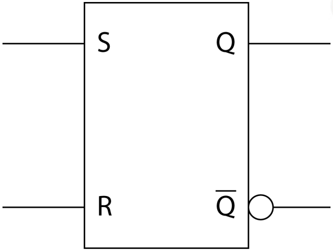


Characteristic Equation:

$Q = D \wedge E \vee \bar{E} \wedge Q$

When the feedback loop is connected the state is maintained

## SR-Latch



Characteristic Equation:
$Q = S \vee \bar{R} \wedge Q$ (Illegality of input (1,1) not covered!)

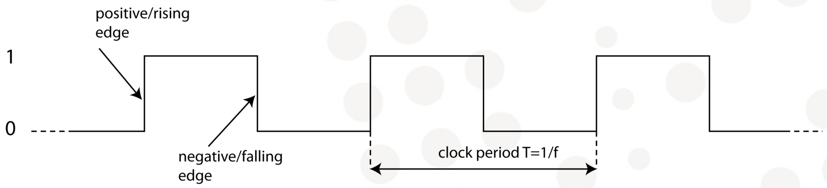
Characteristic Table

| S | R | Q | $Q_{next}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | ? |
| 1 | 1 | 1 | ? |

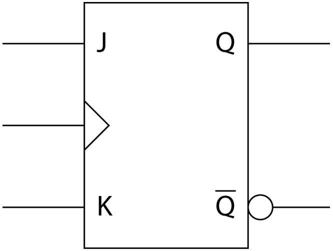
Abbreviated:

| S | R | Q |
|---|---|---|
| 0 | 0 | Q |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | ? |

## Clock Signal



## JK-Flip-Flop



Characteristic Equation:

$Q = J \wedge \overline{Q} \vee \overline{K} \wedge Q$

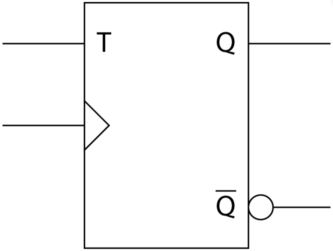
Characteristic Table

| J | K | $Q_t$ | $Q_{t+1}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

abbreviated:

| J | K | $Q_{t+1}$ |
|---|---|---|
| 0 | 0 | $Q_t$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $\overline{Q_t}$ |

## T-Flip-Flop



Characteristic Equation:
$Q = T \oplus Q$

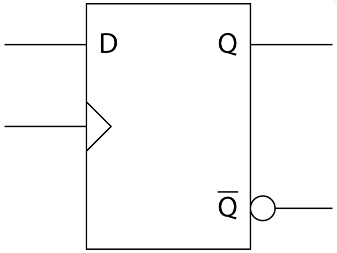
Characteristic Table

| T | $Q_t$ | $Q_{t+1}$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

abbreviated:

| T | $Q_{t+1}$ |
|---|---|
| 0 | $Q_t$ |
| 1 | $\overline{Q}_t$ |

## D-Flip-Flop



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 |

Characteristic Equation:
$Q = D$

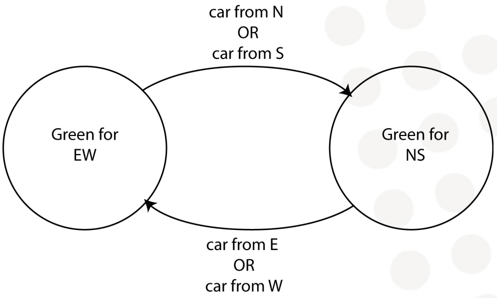
## content

## Finite State Machines

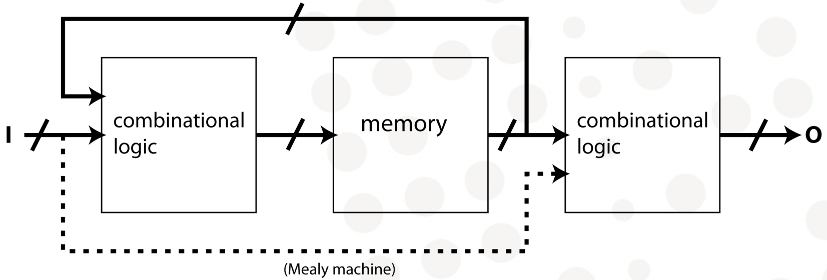
Finite State Machines (FSM) are a formal model suited to describe sequential logic, i.e. logic circuits who’s output does not only depend on the present input but on internal memory and thus, on the history or sequence of the inputs.

## Specifying a FSM with a state transition graph

Example: Traffic Light



## 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.

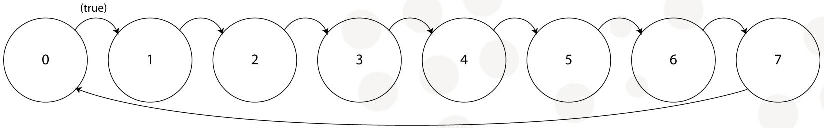
## Synchronous and Asynchronous FSM

- *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.

## content

## 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 |

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

$$= S_2 \wedge (\overline{S_1} \vee \overline{S_0}) \vee (\overline{S_2} \wedge (S_1 \wedge S_0))$$

$$= S_2 \wedge \overline{(S_1 \wedge S_0)} \vee \overline{S_2} \wedge (S_1 \wedge S_0)$$

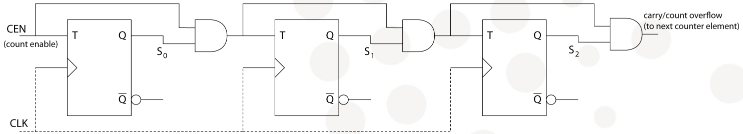
$$= S_2 \oplus (S_1 \wedge S_0)$$

## 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!)

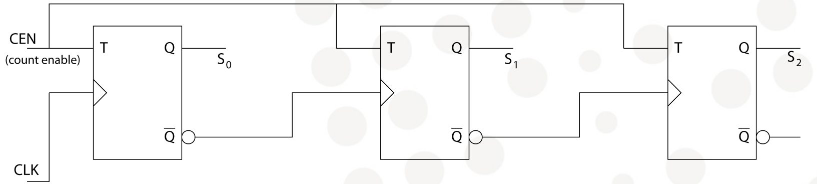
## 3 bit Synchronous Counter



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)

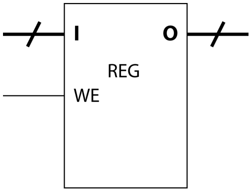
## 3 bit Ripple Counter



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.

## content

## Registers



The Clock signal is implicit and usually not drawn for most higher level synchronous logic

- Memory cells composed of an array of flip-flops that are accessed in parallel (e.g. as memory blocks in a CPU) are called registers
- Most commonly D-flip-flops and additional control combinational logic are used, e.g. a write enable (WE or LD) is 'AND'ed with the global clock for the D-flip-flop clock
- They are sized to store convenient unit of memory (Byte, Word ...)
- They are sometimes equipped with some extra functions beyond storage

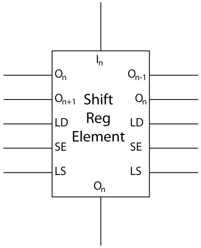
## Shift Register

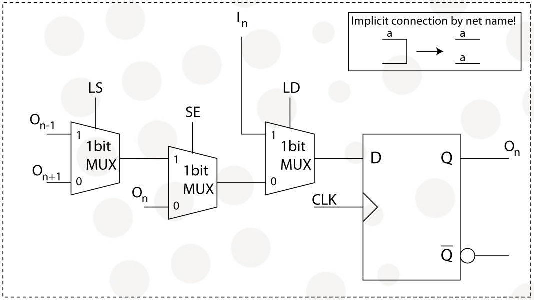
- Shift register are registers that can shift the bits by one position per clock cycle
- The last bit ’falls out’ of the register when shifting
- The first bit that is ’shifted in’ can for example:
  - be set to zero
  - be connected to the last bit (cycling/ring counter)
  - be connected to a serial input for serial loading
- Typical control signals are:
  - load (LD, for parallel loading)
  - shift enable (SE, to enable or stop the shifting)
  - left shift (LS, for controlling the direction of the shift
- A left shift of a binary number corresponds to a multiplication with 2, a right shift to a division with 2

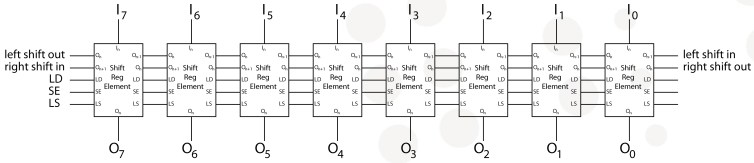
## Shift Register State Transition Table

| control | | | next | | |
|---|---|---|---|---|---|
| LD | SE | LS | $O_2$ | $O_1$ | $O_0$ |
| 1 | X | X | $I_2$ | $I_1$ | $I_0$ |
| 0 | 0 | X | $O_2$ | $O_1$ | $O_0$ |
| 0 | 1 | 0 | RSin | $O_2$ | $O_1$ |
| 0 | 1 | 1 | $O_1$ | $O_0$ | LSin |

# Shift Register Schematics







## Uses for Shift Registers

- Parallel to Serial Conversion
- Binary Multiplication
- Ring ’one-hot’ code counters/scanners
- Pseudo random number generators
- ...

## content

## Half Adder

Example:

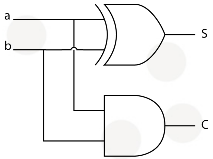
```
    0001
+   0011
--------
=   0100
```

Truth table for a 1-bit half adder:

| a | b | S | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

S is the result and C is the carry bit, i.e. a bit indicating if there is an overflow and an additional bit is necessary to represent the result.

Schematics:



## Full Adder (1/2)

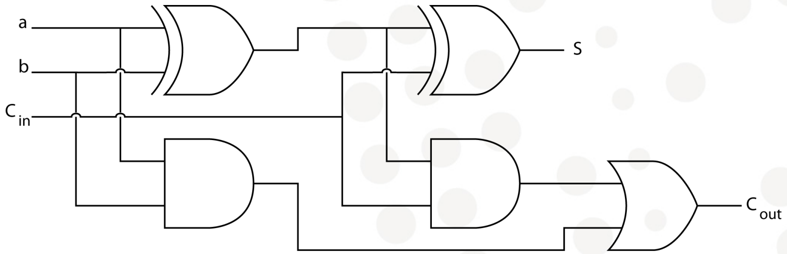
A half adder cannot be cascaded to a binary addition of an arbitrary bit-length since there is no carry input. An extension of the circuit is needed.

Full Adder truth table:

| $C_{in}$ | a | b | S | $C_{out}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

## Full Adder (2/2)

Schematics:


a
b
$C_{in}$
S
$C_{out}$