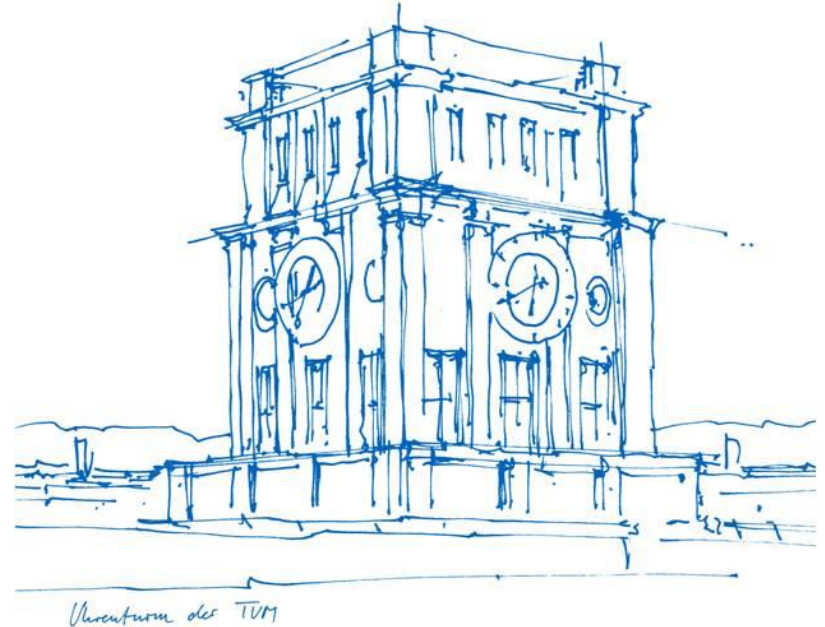


Grundlagenpraktikum: Rechnerarchitektur

SoSe 2024

~ *Danial Arbabi*

danial.arbabi@tum.de



Zulip-Gruppen

Gruppe 29

FR 12:00



<https://zulip.in.tum.de/#narrow/stream/2276-GRA-Tutorium---Gruppe-29>

Gruppe 32

FR 15:00



<https://zulip.in.tum.de/#narrow/stream/2279-GRA-Tutorium---Gruppe-32>

Tutoriums-Website



<https://home.in.tum.de/~arb/>

Disclaimer:

*Dies sind keine offiziellen
Materialien, somit besteht keine
Garantie auf Korrektheit und
Vollständigkeit.*

*Falls euch Fehler auffallen, bitte
gerne melden.*

Organisatorisches

- Noch 2 Inhaltswochen (inkl. dieser)
- Woche 9 Fragestunde
- Teamtreffen
- Praktikumsordnung (vllt. Später genaueres)
- Hausaufgaben und Übungen machen
- Fragen JETZT stellen

Kombinatorische vs. Sequentielle Schaltungen

Kombinatorisch:

- ▶ Typ von logischen Schaltungen.
- ▶ Bestehen aus Eingängen und Ausgängen.
- ▶ Ergebnisse hängen nur von aktuellen Eingängen ab!
 - ▶ Nicht von vorherigen Eingängen.

Sequentiell:

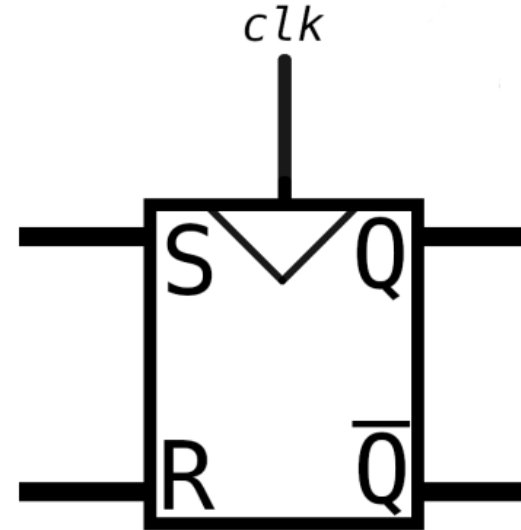
- ▶ Typ von logischen Schaltungen.
- ▶ Bestehen aus Eingängen und Ausgängen.
- ▶ Ergebnisse hängen *auch von vorherigen* Zuständen ab!

Flip-Flops

- ▶ Komponente in vielen Schaltkreisen.
- ▶ Stellt ein Speicherelement dar.
- ▶ Es gibt verschiedene Arten von Flip-Flops:
 - ▶ RS-Flip-Flop (siehe Beispiel)
 - ▶ D-Flip-Flop
 - ▶ und viele mehr...

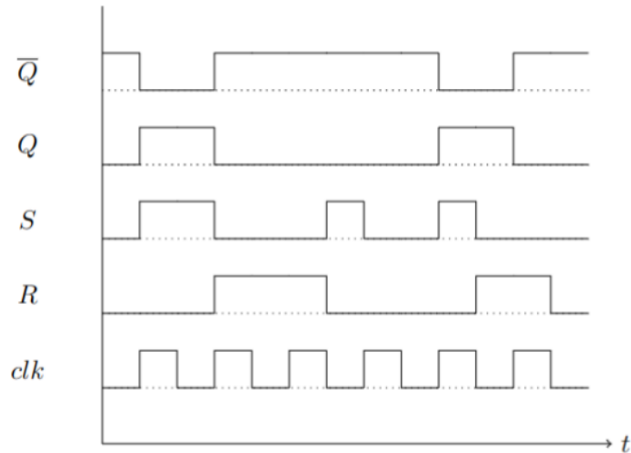
Clock:

- ▶ Bei Wechsel von 0 auf 1 \Rightarrow Flip-Flop wird aktualisiert ("steigende Flanke").
- ▶ Alternativ: Fallende Flanke beim Wechsel von 1 auf 0.



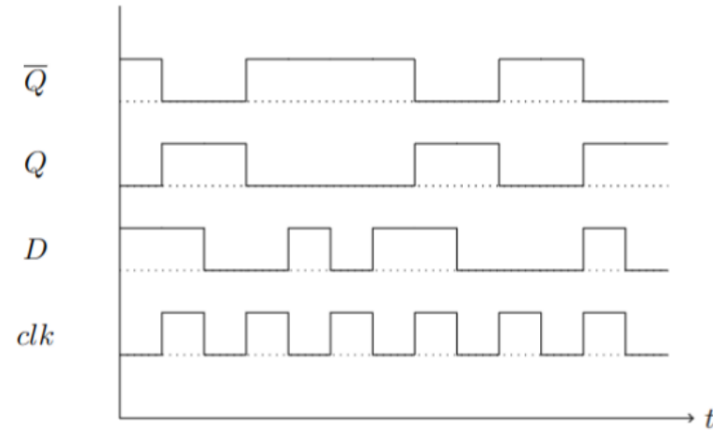
Wellenformen-Diagramm

- ▶ Diagramm zeigt Verlauf von Signalen.
- ▶ Für jeden Input und Output: eigene Zeile.



RS-Flip-Flop

- ▶ Flip-Flop mit nur einem Input: D .
- ▶ Q speichert zu jeder steigenden Clock-Flanke Wert von D ab.
- ▶ Wellenformen-Diagramm mit nur einer Input-Zeile:



D-Flip-Flop

Clocks in SystemC

- Best Practice: Wir erstellen eine Clock in `sc_main` und binden sie an Input Ports in Modulen.

```
1 SC_MODULE(MY_MODULE) {  
2     sc_in<bool> clk;  
3     SC_CTOR(MY_MODULE) { }  
4 };  
5  
6 int sc_main(int argc, char* argv[]) {  
7     sc_clock clk("clk", 2, SC_SEC);  
8  
9     MY_MODULE my_module("my_module");  
10    my_module.clk(clk);  
11  
12    sc_start(10, SC_SEC);  
13    return 0;  
14 }
```


Clocks in SystemC

SC_CTHREAD

Nutzen der Clock mit SC_CTHREAD():

```
1 SC_MODULE(MY_MODULE) {  
2     sc_in<bool> clk;  
3  
4     SC_CTOR(MY_MODULE) {  
5         SC_CTHREAD(behaviour, clk.pos());  
6     }  
7  
8     void behaviour() { ... }  
9 };
```

Clocks in SystemC

SC_CTHREAD

- ▶ SC_CTHREAD(behaviour, clk.pos())
 - ▶ sc_in.pos(): Event für steigende Flanke.
 - ▶ sc_in.neg(): Event für sinkende Flanke.
 - ▶ sc_in.value_changed(): Event für jeden Flankenwechsel.

```
1 ... // SC_CTHREAD(behaviour, clk.pos())
2 void behaviour() {
3     while(true) {
4         wait();
5         std::cout << sc_time_stamp() << std::endl;
6     }
7 }
8 // 2 s
9 // 4 s
10 // 6 s
11 // 8 s
```

Clocks in SystemC

SC_THREAD und SC_METHOD

- ▶ SC_METHOD und SC_THREAD bieten Möglichkeiten, die Prozesse bei steigender Flanke eines `sc_in<bool>` auszuführen.
- ▶ SC_METHOD:

```
1 void behaviour() {  
2     std::cout << sc_time_stamp() << std::endl;  
3     next_trigger(clk.posedge_event());  
4 }
```

- ▶ SC_THREAD:

```
1 ...  
2 SC_THREAD(behaviour);  
3 sensitive << clk.pos();
```