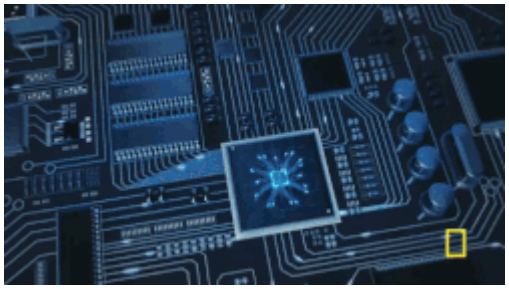
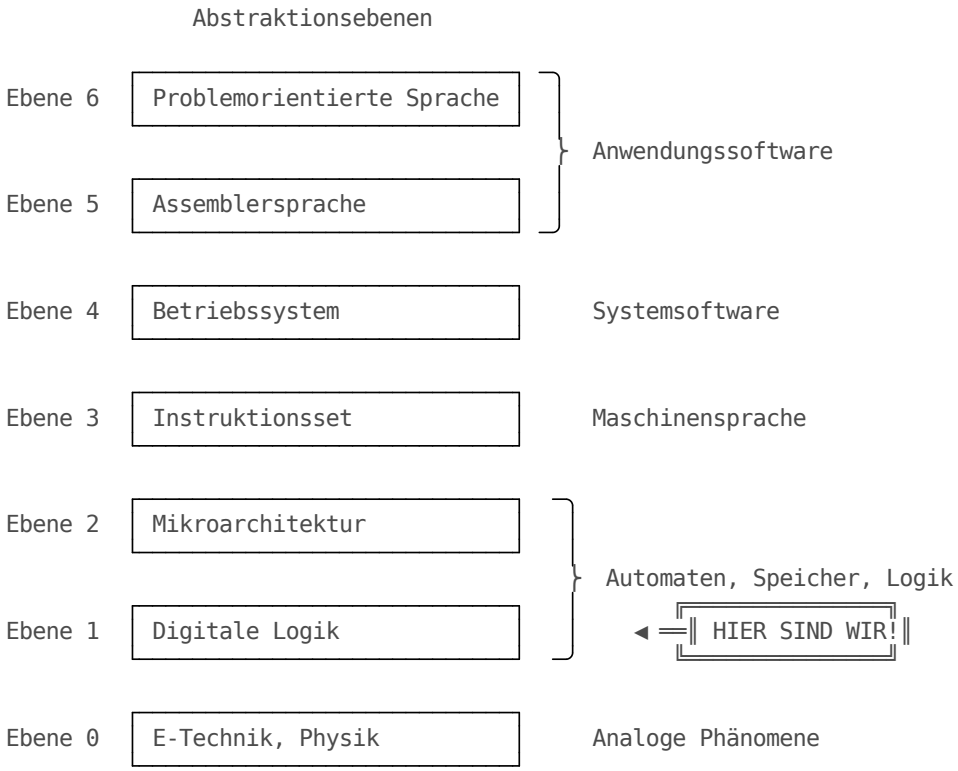
Standardschaltwerke

Parameter	Kursinformationen
Veranstaltung:	`Digitale Systeme / Eingebettete Systeme`
Semester:	`Wintersemester 2025/26`
Hochschule:	`Technische Universität Freiberg`
Inhalte:	`Register, Schieberegister, Zähler und deren Anwendungen als Rechner-Basiskomponenten`
Link auf GitHub:	https://github.com/TUBAF-IfI-LiaScript/VL_EingebetteteSysteme/blob/master/08_StandardSchaltwerke.md
Autoren:	Sebastian Zug & André Dietrich & Fabian Bär



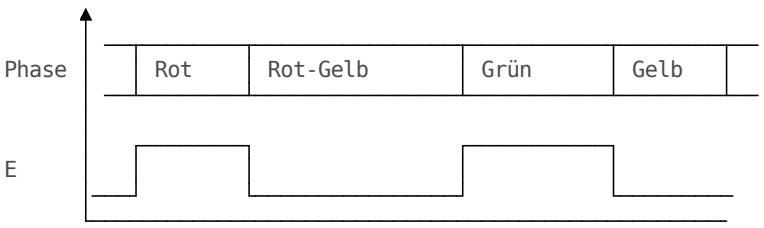
Fragen an die Veranstaltung

- Welche Funktionalität bieten Standard-Schaltwerke wie Zähler und Register?
- Wie unterscheiden sich synchrone und asynchrone Zähler?
- Welche Vorteile bietet ein Gray-Code-Zähler gegenüber einem Binärzähler?
- Was ist ein Schieberegister und welche Betriebsarten kennen Sie?
- Wie funktioniert ein FIFO-Speicher (First-In-First-Out)?
- Welche Arten von Speicherbausteinen kennen Sie (SRAM, DRAM, ROM)?
- Was verstehen Sie unter einer Pipeline in der Datenverarbeitung?
- Wie werden Multiplexer zur Datenweiterleitung in komplexen Systemen eingesetzt?



Übungsbeispiel

Wir wollen eine Ampelanlage entwerfen, die durch einen Eingang E getrieben wird, wobei der Wechsel zwischen den Zuständen von E eine Änderung der Ampelphase bewirkt. Die Umsetzung erfolgt als Moore-Automat.



1. Schritt: Aufstellen des Automatenmodels

2. Schritt: Erstellen der Zustandstabelle

Vorhergehender Zustand	Eingabe E	Nachfolgezustand
Rot	0	RotGelb
Rot	1	Rot
RotGelb	0	RotGelb
RotGelb	1	Grün
Grün	0	Gelb
Grün	1	Grün
Gelb	0	Gelb
Gelb	1	Rot

Schritt 3: Auswahl einer binären Zustandskodierung und Generierung einer binären Zustandstabelle

Zustand	G	F
Rot	0	0
RotGelb	0	1
Grün	1	0
Gelb	1	1

Aufgabe: Ergänzen Sie die Zustandsübergangstabelle!

G	F	E	G'	F'
		0		
		1		
		0		
		1		
		0		
		1		
		0		
		1		

Lösung:

Vorhergehender Zustand	G	F	Eingabe E	Nachfolgezu
Rot	0	0	0	RotGelb
Rot	0	0	1	Rot
RotGelb	0	1	0	RotGelb
RotGelb	0	1	1	Grün
Grün	1	0	0	Gelb
Grün	1	0	1	Grün
Gelb	1	1	0	Gelb
Gelb	1	1	1	Rot

Schritt 4: Auswahl eines Flip-Flop Typs und Ermittlung der für jeden Zustandsübergang benötigten Flip-Flop Ansteuerungen

Wir entscheiden uns für einen JK-Flip-Flop für die Realisierung. Die entsprechende invertierte Wahrheitstafel haben Sie zwischenzeitlich im Kopf:

$Q(t)$	$Q(t+1)$	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

Schauen wir zunächst auf den Flip-Flop G, dessen Eingangsbelegung muss also auf die erwarteten Zustandsübergänge abgebildet werden.

G	**F**	**E**	**G'**	**F'**
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Aufgabe: Ergänzen Sie die Einträge für JF und KF!

G	F	E	G'	F'
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Minimieren der Schaltung

G	F	E	JG	KG
0	0	0	0	d
0	0	1	0	d
0	1	0	0	d
0	1	1	1	d
1	0	0	d	0
1	0	1	d	0
1	1	0	d	0
1	1	1	d	1

Aufgabe: Lesen Sie die minimierten Funktionen für die Flip-Flop-Eingänge ab!

JG

	$\overline{GF}$	$\overline{G}F$	GF	$G\overline{F}$
$\overline{E}$			d	d
E		1	d	d

JF

	$\overline{GF}$	$\overline{G}F$	GF	$G\overline{F}$
$\overline{E}$	1	d	d	1
E		d	d	

KG

	$\overline{GF}$	$\overline{G}F$	GF	$G\overline{F}$
$\overline{E}$	d	d		
E	d	d	1	

KF

	$\overline{GF}$	$\overline{G}F$	GF	$G\overline{F}$
$\overline{E}$	d			d
E	d	1	1	d

.

Damit lassen sich folgende Funktionen ablesen:

$$JG = FE$$
$$KG = FE$$
$$JF = \overline{E}$$
$$KF = E$$

Ausgabelogik

Die Ansteuerung der drei Ausgabeleitungen R, Gr, Ge ergibt sich aus den Zuständen:

Zustand	**G**	**F**	R	Gr
Rot	0	0	1	0
RotGelb	0	1	1	0
Grün	1	0	0	1
Gelb	1	1	0	0

$$R = \overline{G}$$
$$Gr = G\overline{F}$$
$$Ge = F$$

Wie sehen alternative Umsetzungen aus?

Simulation time: 00:23.906 (122%)

statemachine.cpp

```
typedef struct {
    int state;
    int next;
    int A_red;
    int A_yellow;
    int A_green;
    int timer;
} tl_state_t;    // Traffic light state

tl_state_t states[4] = {
// state     A_red             timer
//  |   next  |  A_yellow       |
//  |    |    |    |    A_green  |
//-------------------------------------------
{   0,   1,   1,  0,    0,      3},
{   1,   2,   1,  1,    0,      1 },
{   2,   3,   0,  0,    1,      3},
{   3,   0,   0,  1,    0,      1,}
};

const int greenPin = 11;
const int yellowPin = 12;
const int redPin = 13;
int state = 0;

void setup() {
  pinMode(greenPin, OUTPUT);
  pinMode(yellowPin, OUTPUT);
  pinMode(redPin, OUTPUT);
}

void loop() {
  if (states[state].A_red == 1) digitalWrite(redPin, HIGH);
  else digitalWrite(redPin, LOW);
  if (states[state].A_yellow == 1) digitalWrite(yellowPin, HIGH);
  else digitalWrite(yellowPin, LOW);
  if (states[state].A_green == 1) digitalWrite(greenPin, HIGH);
  else digitalWrite(greenPin, LOW);
  delay(states[state].timer*1000);
  state =  states[state].next;
}
```

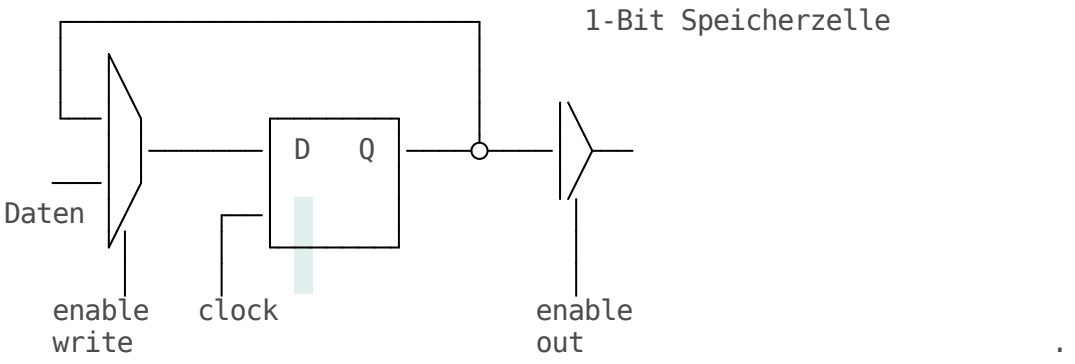
```
Sketch uses 1150 bytes (3%) of program storage space. Maximum is 32256
bytes.
Global variables use 59 bytes (2%) of dynamic memory, leaving 1989
bytes for local variables. Maximum is 2048 bytes.


Sketch uses 1150 bytes (3%) of program storage space. Maximum is 32256
bytes.
Global variables use 59 bytes (2%) of dynamic memory, leaving 1989
bytes for local variables. Maximum is 2048 bytes.
```

Methode	Bauteilkosten	Entwurf	Variabilität
Analoge Schaltung	minimal	aufwändig	gering
Digitale Logik	gering	einfach	mittel
Software (eingebetteter μC)	überschaubar	einfach	hoch
Software (PC)	hoch	sehr einfach	hoch

Register

> **Merke** Register bilden das Zusammenspiel mehrerer Flip-Flops.

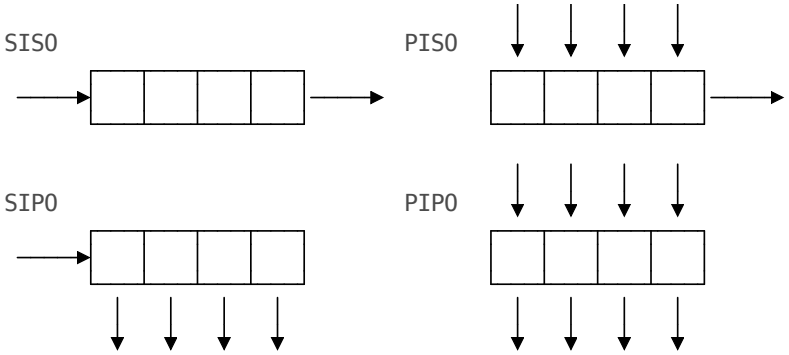


.

Aggregieren wir nun mehrere D-Flip-Flops in einem Baustein, können wir wie nachfolgend gezeigt 4 Bit damit speichern. Der Flip-Flop $1Q$ könnte dabei die niedrigste Stelle in unser Zahlenrepresentation einnehmen, $Q4$ die höchste. Um also 1011 in unserem Speicher abzulegen,

müssten wir and den Inputs D_1 bis D_4 die zugehörigen Pinbelegungen realisieren, die dann mit dem nächsten Taktflankenwechsel übernommen werden. Die Übernahme der daten wird dabei durch zwei Enable Flags definiert, die $G, \overline{G}$ konfiguriert.

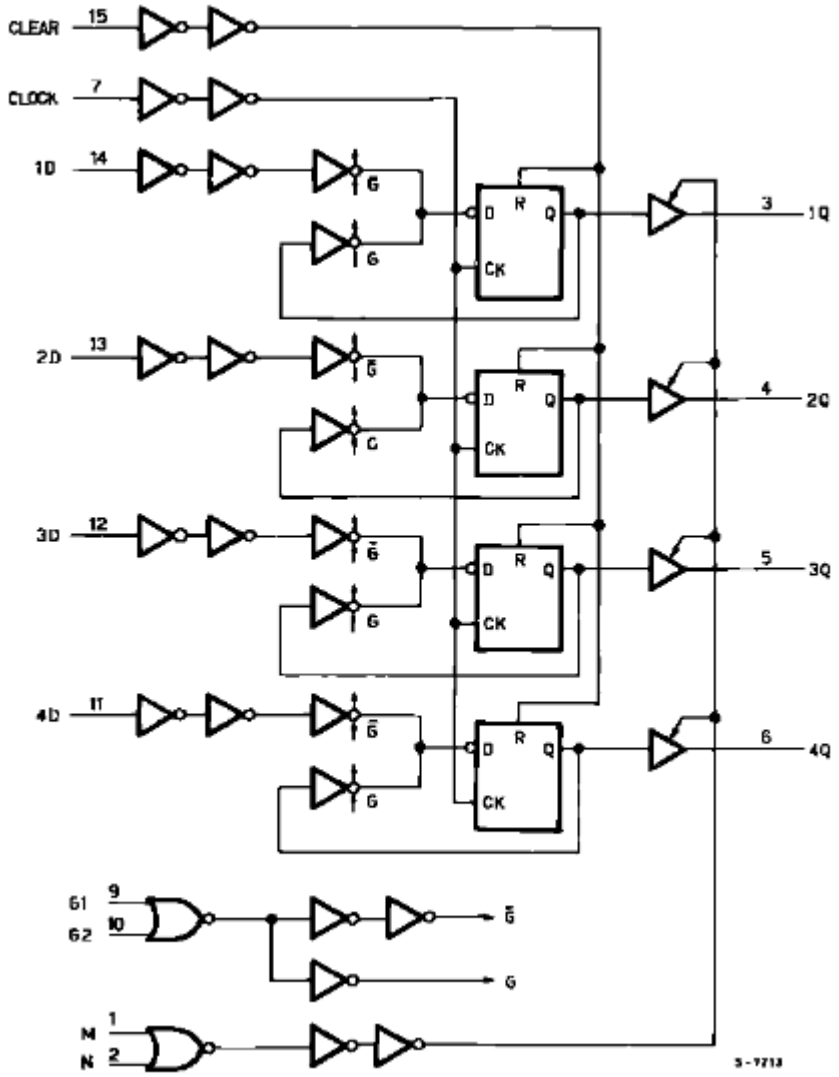
Die Eingabe der Daten kann seriell (serial input, SI) oder parallel erfolgen (parallel input, PI), ebenso die Ausgabe (serial output, SO bzw. parallel output, PO).



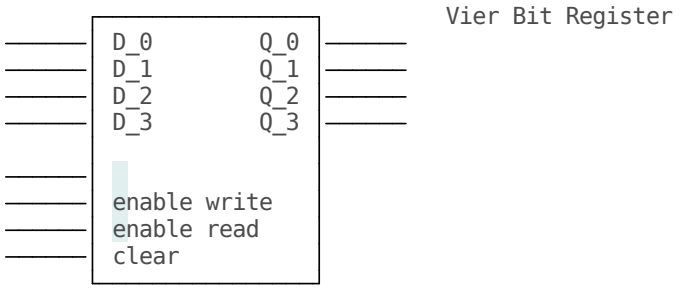
.

Prinzip	**Anwendung**
SISO	FIFO-Speicher (first-in-first-out), Multiplikation bzw. Division mit dem Faktor 2
SIPO	Umwandlung serielles Datenformat in paralleles Datenformat
PISO	Umwandlung paralleles Datenformat in serielles Datenformat
PIPO	Register zum reinen Zwischenspeichern, z. B. Akkumulator

Schauen wir uns das Ganze mal am Beispiel eines 4 Bit PIPO Registers an:

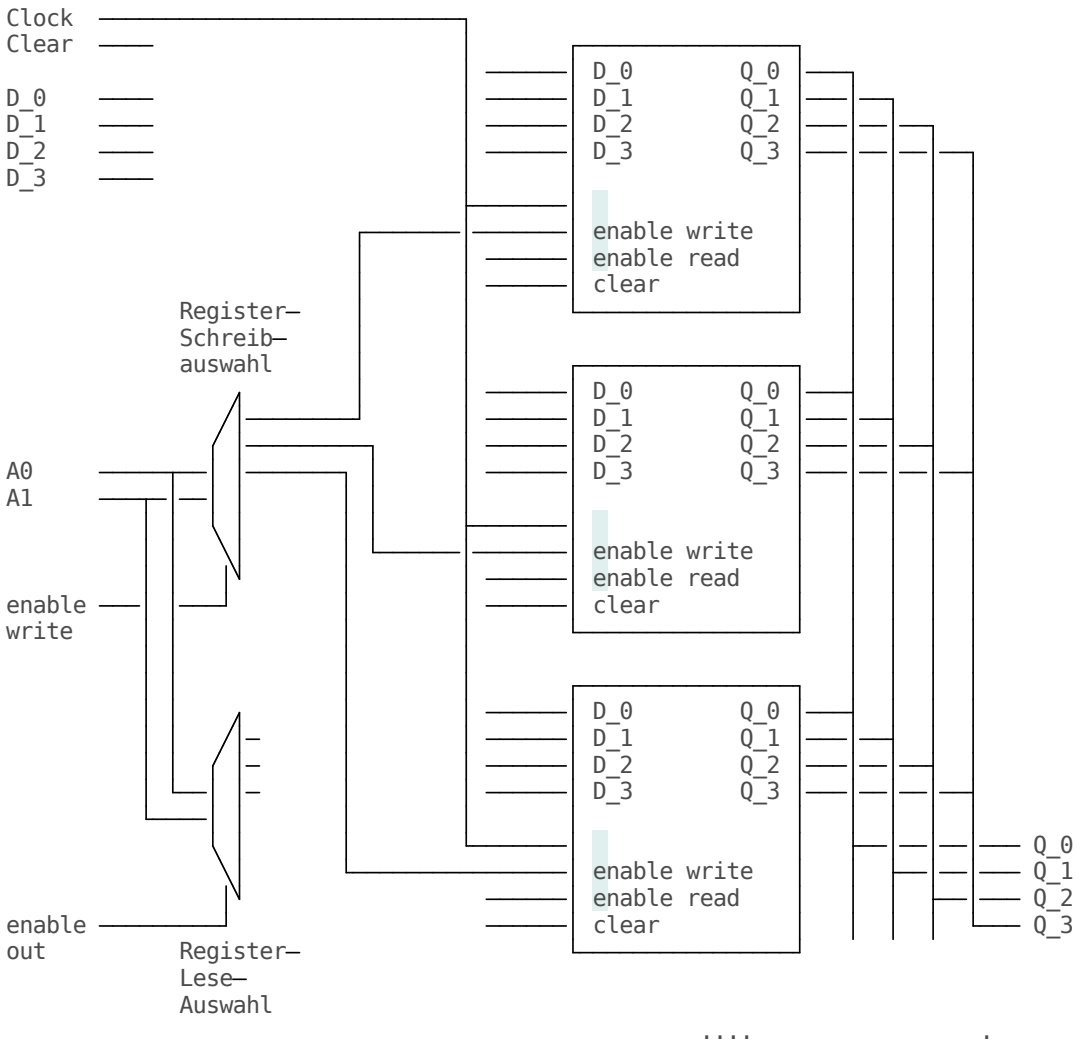


SGS Thomson, Datenblatt 74HC173, Quad D-Type Register [1]



In der Summe brauchen wir damit 4 Eingangsleitungen, 4 Ausgangsleitungen sowie diverse Steuerleitungen - für eine größere Anzahl an Registern eine zu aufwändige Realisierung - wir brauchen einen "Zuteilungsmechanismus".

Entsprechend führen wir eine Adresse ein, diese kodiert, welches Register konkret angesprochen werden soll. Die Datenbreite bleibt erhalten.



An diesem Schaubild werden zwei zentrale Größen einer Rechnerarchitektur deutlich. Die Adressbusbreite und die Wortlänge bzw.. die Datenbusbreite des Systems z. B. 8, 16, 32 oder 64 Bit.

An dieser Stelle sei noch einmal auf die Erläuterungen zum Schieberegister aus der Veranstaltung L6 verwiesen. Dieses erweitert die Funktionalität und ermöglicht neben dem Abspeichern und Lesen auch Shiftoperationen.

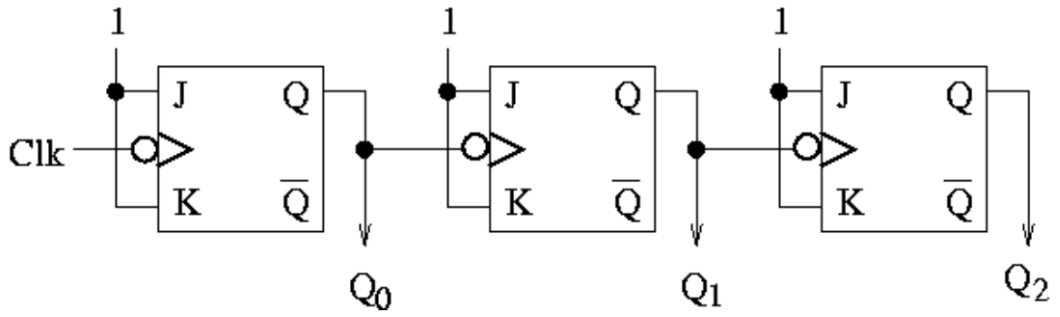
---

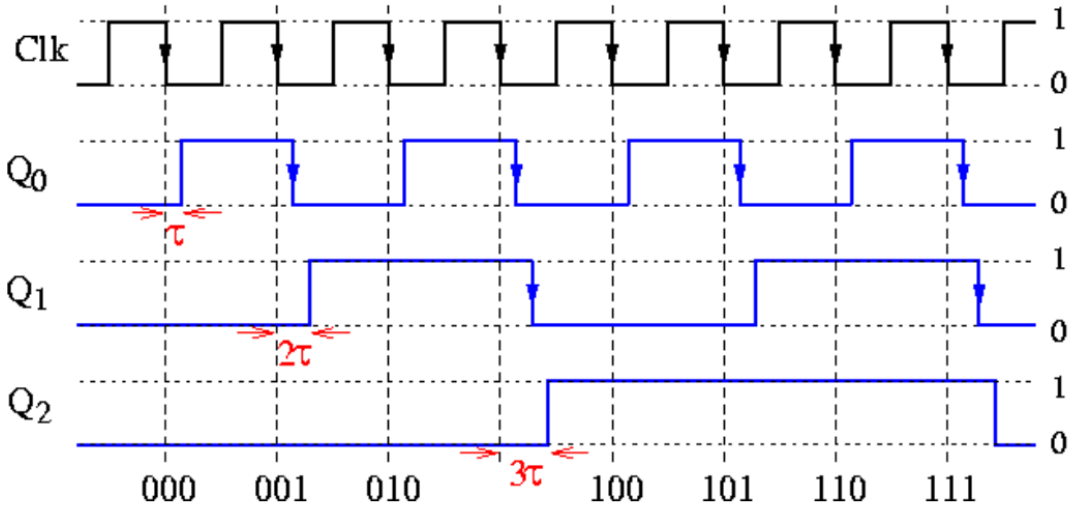
[1] SGS Thomson, Datenblatt 74HC173, "Quad D-Type Register", Link

Zähler

Eine sequentielle Schaltung, die Eingangsimpulse auf Zustände abbildet nennt man Zähler. Mit ihnen lassen sich zeitlich aufeinander folgende Ereignisse registrieren aber auch erzeugen.

- **Synchron vs. asynchron** In einem synchronen Zähler sind alle Zähler-Speicher durch eine Taktleitung getriggert. Der Zustandswechsel wird immer zugleich vollzogen. Asynchrone Zähler heben diese Trennung auf und erlauben eine vereinfachte Umsetzung. Allerdings ist die Gültigkeit der Ergebnisse dann von dem Laufzeitverhalten der Komponenten abhängig.





- **Zahlenformat** Die binäre Zahlenrepräsentation ist die intuitive Darstellung des Zahlenwertes. Alternativ lassen sich aber auch Dezimale Zähler, beruhend auf z. B. dem BCD-Code oder Zähler mit anderen Codes arbeiten, z. B. Gray-Code umsetzen.
- **Zählrichtung** Unidirektionale Zähler können ausschließlich vorwärts oder rückwärts zählen (inkrementieren/dekrementieren). Bidirektionale Zähler unterstützen beide Zählrichtungen, benötigen aber auch eine separate Steuerleitung.

Leiten wir uns einen synchronen 4 Bit Zähler zur Vertiefung des Verständnisses noch mal her:

Offenbar sind 4 Zustände notwendig, um die Zahlen von 0-15 abbilden zu können. Die Zustandsübergangstabelle ergibt sich folglich zu.

F	G	H	I	F'
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

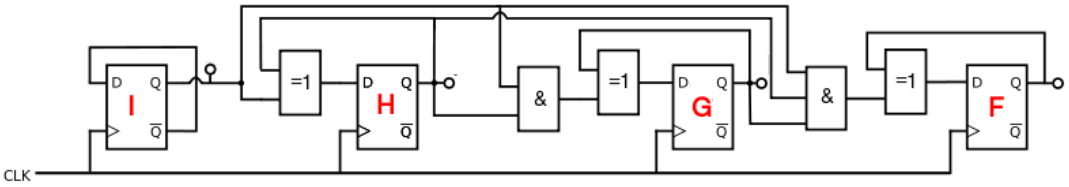
Unter der Vorgabe, dass wir D-Flip-Flops für die Umsetzung verwenden, sind die Eingänge der Speicher mit den Zuständen F' bis I' zu beschalten.

In der Vereinfachung ergeben sich daraus folgende Gleichungen:

$$\overline{\quad} \quad \overline{\quad} \quad \overline{\quad} \quad \overline{\quad}$$

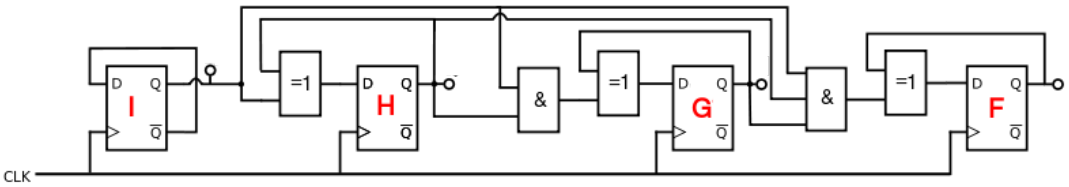
$$
\begin{aligned}
DF &= \overline{I}F + \overline{G}F + \overline{H}F + IHG\overline{F} \\
DG &= \overline{I}G + \overline{H}G + IH\overline{G} \\
DH &= I\overline{H} + \overline{I}H \\
DI &= \overline{I}
\end{aligned}
$$

Im Ergbnis lässt sich das System mit entsprechenden XOR und AND Bausteinen Umsetzen.



Aufgabe: Leiten Sie die Im Schaltwerk gezeigte Realisierung aus den oben genannten Gleichungen her.

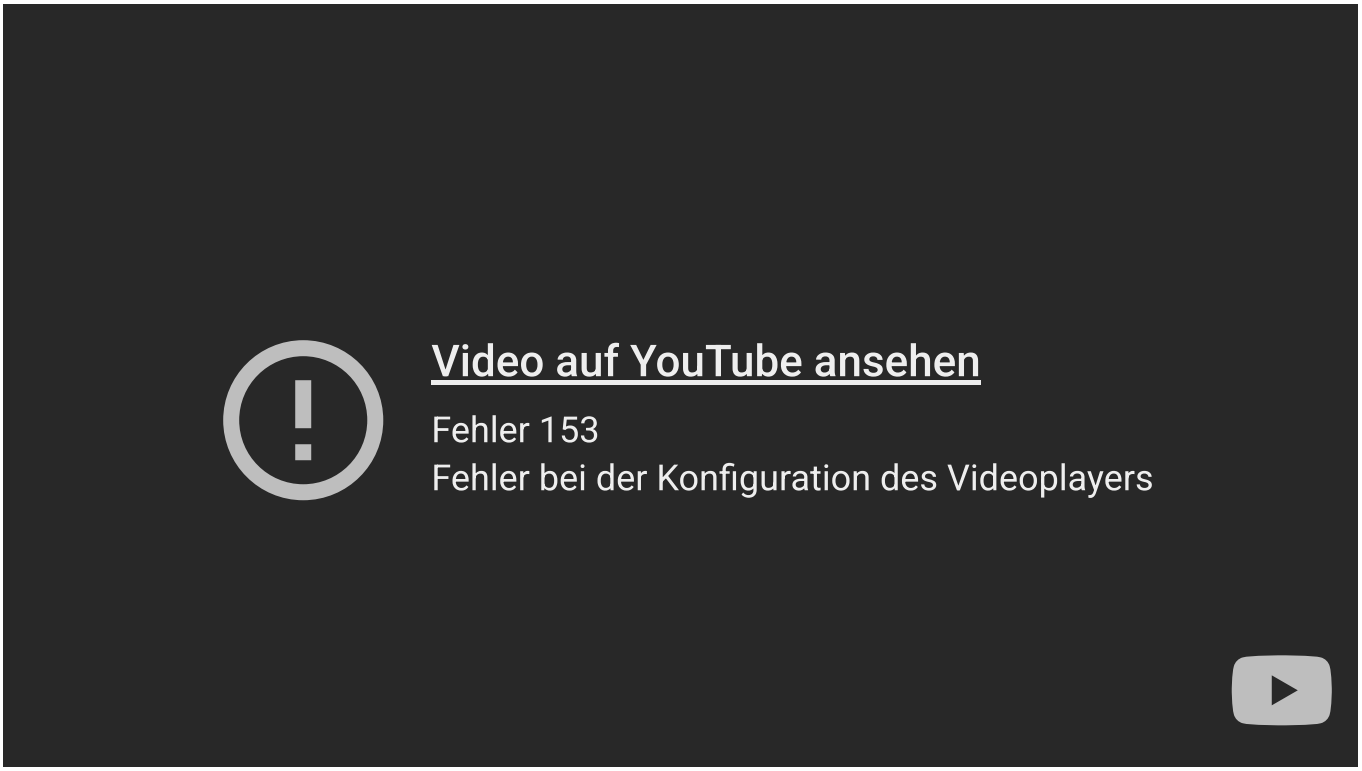
Aufgabe: Leiten Sie die Im Schaltwerk gezeigte Realisierung aus den oben genannten Gleichungen her.



Für DI und DH ist die Lösung trivial, für DG müssen wir den Term etwas umstellen. Die Schaltung drückt die Gleichung $DG = IH \oplus G$ aus.

$$
\begin{aligned}
DG &= IH \oplus G \\
&= \overline{IH}G + IH\overline{G} \\
&= (\overline{I} + \overline{H})G + IH\overline{G} \\
DG &= \overline{I}G + \overline{H}G + IH\overline{G}
\end{aligned}
$$

Merke Aus der Kombination der genannten Klassifkationsmerkmale für Zähler lassen sich spezifische Umsetzungen realisieren.



Alternative Implementierung mit erweiterter Funktionalität:

Der Zustand eines Zählerbits **wechselt**, wenn das nächst kleinere Bit **von 1 zu 0** springen wird.

a	b	c	d	a'	b'	c'	d'
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0

Das bedingte Wechseln eines Bits lässt sich mit XOR realisieren:

$$x \oplus 0 = x$$
$$x \oplus 1 = \overline{x}$$

Wenn man möchte, dass x wechselt, wenn x_{flip} high ist, dann kann $x \oplus x_{flip}$ verwendet werden.

Aus der Bedingung oben ergibt sich:

$$a_{flip} = b_{flip} \cdot b$$
$$a = a \oplus a_{flip}$$

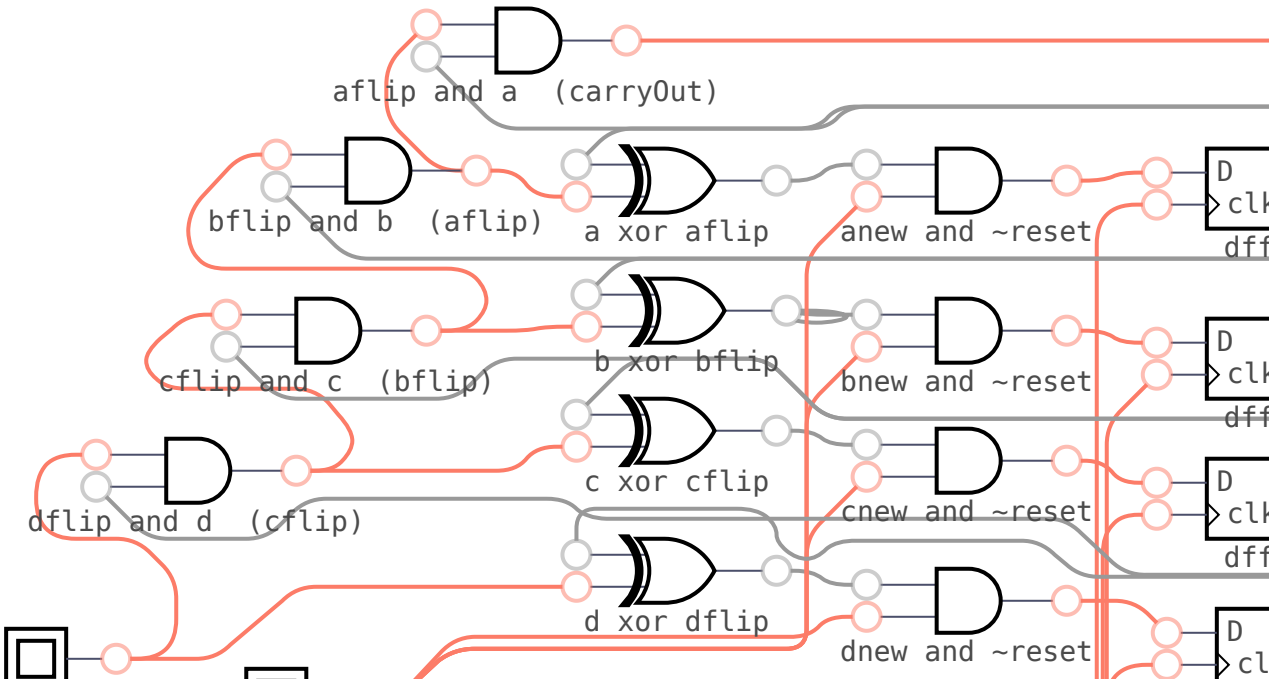
a wird wechseln, wenn b wechseln wird und $b = 1$ gilt, wobei b das nächst niederwertigste Bit von a ist. b_{flip} lässt sich wie a_{flip} berechnen, wenn es nicht das niederwertigste Bit ist. Das niederwertigste Bit wechselt dann wenn der Eingang $count$ (carryIn) auf High ist.

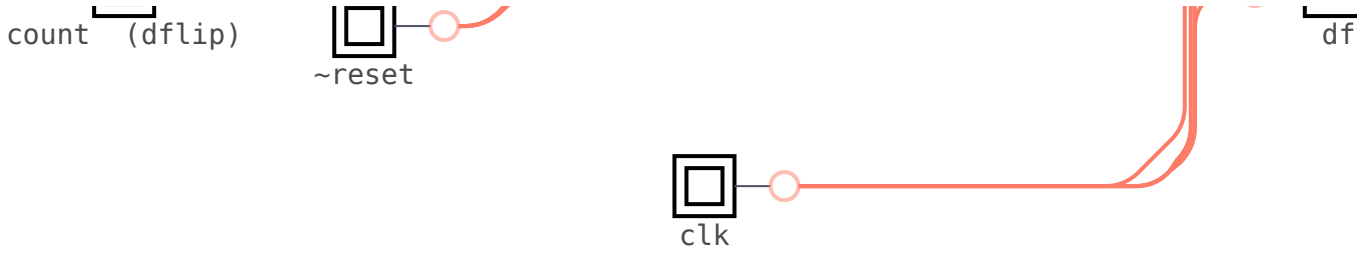
Der Ausgang $carryOut$ lässt sich berechnen mit $a_{flip} \cdot a$.

Für einen 4 Bit Zähler mit den Bits a,b,c,d (a höchstwertig, d niederwertigste) ergeben sich diese Formeln:

$$\begin{aligned} d_{flip} &= count \\ d' &= d_{flip} \oplus d \\ c_{flip} &= d_{flip} \cdot d \\ c' &= c \oplus c_{flip} \\ b_{flip} &= c_{flip} \cdot c \\ b' &= b \oplus b_{flip} \\ a_{flip} &= b_{flip} \cdot b \\ a' &= a \oplus a_{flip} \\ carryOut &= a_{flip} \cdot a \end{aligned}$$

Aus technischen Gründen hat die folgende Simulation noch eine Reset Schaltung (jeder neue Zustand wird noch mit $\overline{reset}$ UND genommen.) Am Anfang `count` und `~reset` auf LOW lassen, clock puls geben und dann `count` und `~reset` auf HIGH setzen um bei dem nächten clock Puls das Zählen zu beginnen.





Nachteil dieser Schaltung: höhere Laufzeiten

Übungsaufgaben

- Entwerfen Sie einen Zähler, der eine Zählrichtungsvorgabe ermöglicht und evaluieren Sie Ihre Lösung mit einem Simulator.

  logisim-evolution

- Recherchieren Sie die Umsetzung eines Johnson Zählers und übernehmen Sie eine Implementierung in eine Simulator.