5. Binäre Steuerungen und Boolesche Algebra

-> hierzu auch Folgen 1 bis 5 der Reihe Praktische Einführung in CoDeSys

5.1 Übersicht

Binärische Steuerungen zeichnen sich durch ihre Beschränkung auf **Bitoperationen** aus. Sie verarbeiten ausschließlich Bitsignale. Alle Variablen sind vom Typ BOOL.

Systeme der binärischen Signalverarbeitung heißen auch **logische Netzwerke.** Diese enthalten **logische Funktionen**.

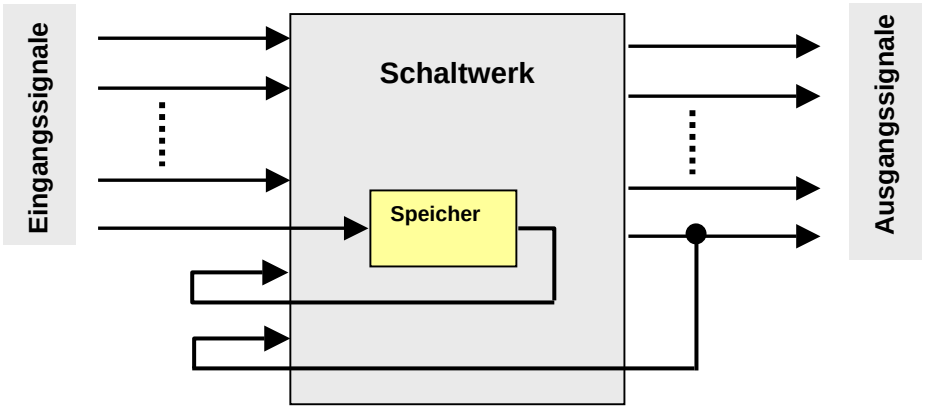
Je nachdem ob die logischen Netzwerke innere Signalspeicher enthalten oder nicht, werden unterschieden:

- **Kombinatorische Netzwerke = Schaltnetze = Verknüpfungssteuerungen ohne Speicher**

Sie enthalten keine inneren Speicher und keine Rückführungen von Ausgangssignalen auf Eingänge. Als Folge sind die Ausgangssignale eindeutig eine logische Funktion (Kombination) allein der Eingangssignale.

- **Sequentielle Netzwerke = Schaltwerke = Verknüpfungssteuerungen mit Speicher**

Sie enthalten innere Speicher und / oder Rückführungen von Ausgangssignalen auf Eingänge. Als Folge sind die Ausgangssignale nicht mehr eindeutig nur eine logische Funktion (Kombination) der Eingangssignale, sondern hängen auch noch vom Zustand der Ausgangssignale bzw. der inneren Speicher ab.



Binäre Steuerungen wurden über Jahrzente mit unterschiedlicher Technik realisiert:

- Fluidisch mit pneumatischen und hydraulischen Schaltgliedern
- Elektrisch mit Kontakt-Schaltelementen, Logikbaugruppen und schließlich mit programmierbaren Baugruppen.

Grundsätzlich haben fluidische Steuerungen deutlich an Bedeutung verloren, sie finden sich nur noch in Nischen. Dagegen behalten **fluidische Aktoren und Stellglieder** weiterhin große Bedeutung.

Durchgesetzt haben sich auch in der binären Steuerungstechnik **programmierbare Steuerungen**, da sie bereits für kleinere Aufgaben mit nur wenigen Verknüpfungen wirtschaftlich sind.
(Beispiele: System LOGO! der Siemens AG, System Easy von Moeller)

Binäre Steuerungstechnik auf Basis von **Relais und Schützen** wird seit Beginn der Elektrotechnik praktiziert. Bereits hier wurden die logischen Grundfunktionen definiert. Da die Realisierung einer logische Funktion wie z.B. eines UND-Gliedes durch Relais nicht zu vernachlässigende Kosten verursacht, wurde durch Anwendung der **Boolschen Algebra und Minimierungsverfahren** versucht, die Logik von jedweder Redundanz zu befreien. Jedes eingesparte Relais bedeutete geringere Kosten.

Die **Boolesche Algebra** (Schaltalgebra) ist eine mathematische Beschreibung für die Beziehungen zwischen logischen Variablen, die nur zwei Zustände annehmen können. Ähnlich wie in der Algebra existieren ein Kommutativ-, Assoziativ- und Distributiv-Gesetz für das Ausklammern und Tauschen von Termen mit Variablen.

Bei programmierbarer Technik hat die Einsparung einer Anweisung nur in Sonderfällen Auswirkungen! Der Speicherplatz für einige Anweisungen mehr oder weniger fällt im allgemeinen nicht ins Gewicht. Deshalb haben heute die Minimierungsverfahren wie auch die Boolesche Algebra **an Bedeutung verloren**. Es gibt aber weiterhin Sonderprobleme, wo Logik entwickelt und minimiert werden muss.

Bekannte Verfahren zur Minimierung der Schaltgleichung von Schaltnetzen sind der **Karnaugh-Plan** und das Verfahren von **Quine-McCluskey**. Bei Bedarf wird hierzu auf die Literatur verwiesen.

➔ ***hierzu Dateien der Disk2:***
Karnaugh_Diagramme.pdf sowie Karnaugh_Erläuterungen.pdf

Während der historische Entwicklung der binären Steuerungstechnik haben sich die Zeichen der Schaltalgebra deutlich gewandelt:

UND: E1 ∩ E2 -> E1 ∧ E2 -> E1 & E2 , heute E1 . E2 oder E1 * E2

ODER: E1 ∪ E2 -> E1 ∨ E2 , heute E1 + E2

Negation: /E1 oder $\overline{E1}$

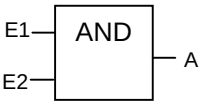
5.2 Logische Grundfunktionen

Für die Verknüpfung von Signalen in Schaltnetzen reichen wenige logische Grundfunktionen aus, so wie sie mit der Relaistechnik realisierbar waren.

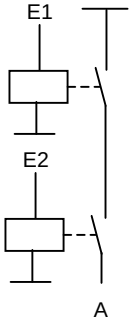
Nachfolgende Übersicht zeigt die möglichen Darstellungen der logischen Verknüpfungen.
Die AWL oben wurde mit Variablen geschrieben gemäß IEC 1131-3, unten jeweils die Schreibweise von Step7 mit Symbolen.

Operation	FUP-Operator KOP-Operator	AWL	Kontaktschaltung	Schalttabelle Boolsche Gleichung

UND (Konjunktion)



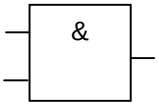

```
LD   Eingang_1
AND  Eingang_2
ST   Ausgang
```



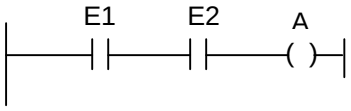
E1	E2	A
0	0	0
0	1	0
1	0	0
1	1	1

A = E1 * E2

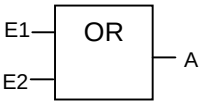
Step7



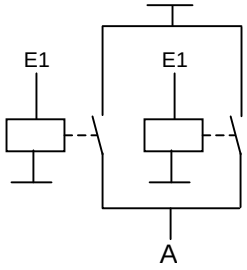

```
U   "Eingang_1"
U   "Eingang_2"
=   "Ausgang"
```



ODER (Disjunktion)



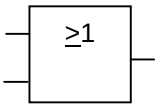

```
LD   Eingang_1
OR   Eingang_2
ST   Ausgang
```



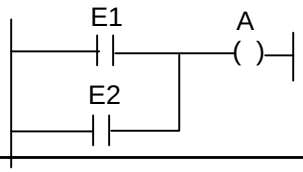
E1	E2	A
0	0	0
0	1	1
1	0	1
1	1	1

A = E1 + E2

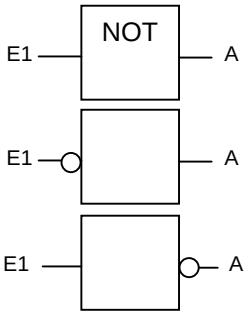
Step7



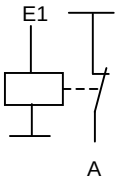

```
U   "Eingang_1"
O   "Eingang_2"
=   "Ausgang"
```



Nicht (NEGATION)



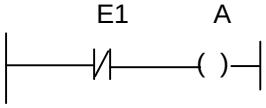

```
LD   Eingang_1
NOT
ST   Ausgang
oder
LDN  E1
ST   A
```



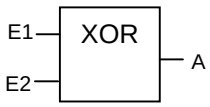
E1	A
0	1
1	0

$A = \overline{E1}$

```
Step7
U   "Eingang_1"
NOT
=   "Ausgang"
oder
UN  "Eingang_1"
=   "Ausgang"
```

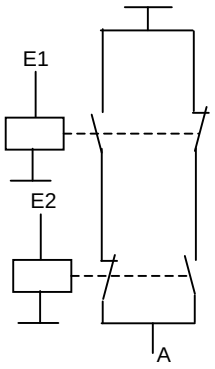


EXKLUSIV-ODER



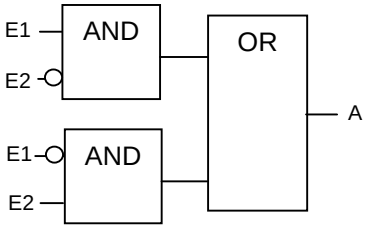

```
LD    Eingang_1
XOR Eingang_2
ST    Ausgang
```

```
X  "Eingang_1"
X  "Eingang_2"
=  "Ausgang"
```



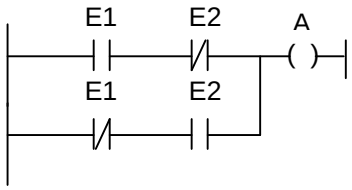
E1	E2	A
0	0	0
0	1	1
1	0	1
1	1	0

XOR mit Grundoperationen ausgeführt




```
LD     Eingang_1
ANDN Eingang_2
OR (
LDN    Eingang_1
AND    Eingang_2
)
ST     Ausgang
```

$$A = E1 * \overline{E2} + \overline{E1} * E2$$

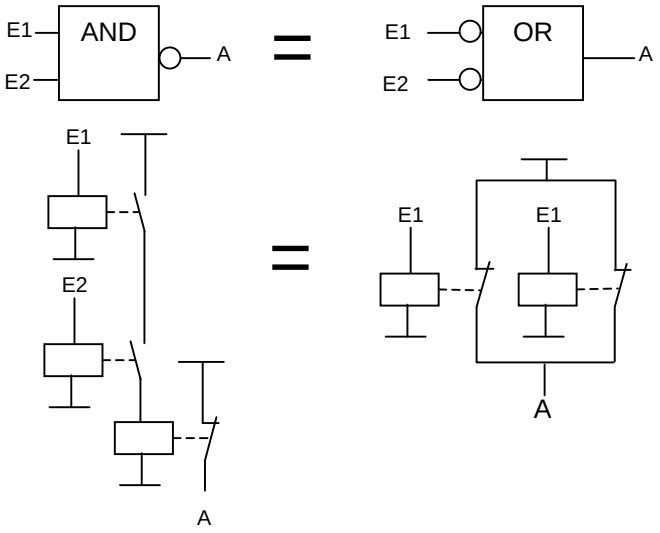


Ein Beispiel der **Booleschen Algebra** von besonderem Interesse ist das **DeMorgan-Theorem**: Danach liefert die Negation eines Terms das gleiche Ergebnis wie die alternative Verknüpfung der einzeln negierten Signale. Mit dem Theorem kann man beispielsweise eine UND-Verknüpfung gegen eine ODER-Verknüpfung austauschen.

DeMorgan-Theorem:

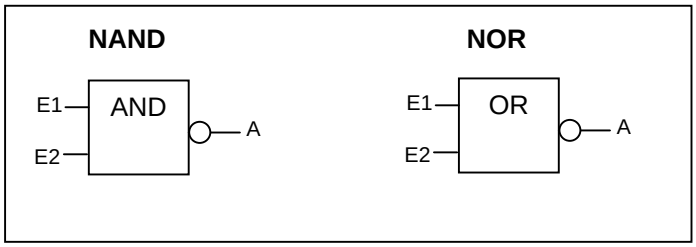
Schaltgleichung

$$A = \overline{E1 \cdot E2} = \overline{E1} + \overline{E2}$$



Das Beispiel des DeMogan Theorems macht deutlich, dass Boolesche Algebra durchaus zur Einsparung von Schaltgliedern führen kann und in bestimmten Fällen Schließerkontakte gegen Öffnerkontakte getauscht werden können.

Die hierbei benutzten Glieder **NAND und NOR** waren Basis verschiedener Transistor-Logik-Familien und galten deshalb historisch ebenfalls als logische Grundfunktionen



Nachfolgender Auszug der AWL Referenzliste von CoDeSys zeigt die IEC-Operatoren für Logische Verknüpfungen:

LD	Lädt den Operanden in den Akkumulator.
LDN	Lädt den negierten Wert des Operanden in den Akkumulator.
ST	Speichert den Inhalt des Akkumulators in die Variable, die als Operand verwendet wird.
STN	Speichert den negierten Inhalt des Akkumulators in die Variable, die als Operand verwendet wird.
AND	Bitweises AND von Akkumulator und Operand
ANDN	Bitweises AND von Akkumulator und negiertem Operand
OR	Bitweises OR von Akkumulator und Operand
ORN	Bitweises OR von Akkumulator und negiertem Operand
XOR	Bitweises exklusives OR von Akkumulator und Operand
XORN	Bitweises exklusives OR von Akkumulator und negiertem Operand
NOT	Bitweise Negation des Akkumulatorinhalts

Das Programmieren binärer Logik erfolgt nach folgendem Grundschema:

1. Laden der ersten Variablen (des ersten Operanden) in den Akkumulator mit den Operationen LD oder LDN (bei Step7 mit U oder UN). Dieser Schritt wird auch als Erstabfrage bezeichnet.

2. Nach jeder logischen Verknüpfung steht das Ergebnis wieder im Akkumulator.

3. Ausgabe des Akkumulatorinhaltes in die gewünschte Variable (den gewünschten Operanden) mit den Operationen ST oder STN (bei Step7 mit Zuweisung (=)).

Achtung! Die IEC-Operation STORE (ST) gilt sowohl für Variable vom Typ BOOL als auch für BYTE, WORD oder DWORD.

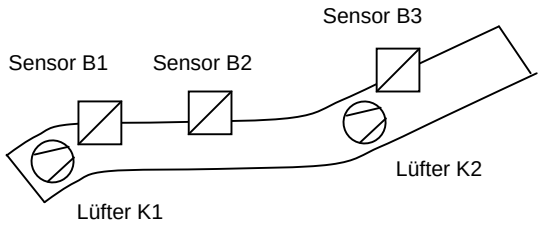
ST =

Dagegen unterscheidet Step7 die Zuweisung (=) für Operanden vom Typ BOOL und Transferieren (T) für Operanden vom Typ BYTE, WORD oder DWORD bzw. INT und DINT

5.3 Lösungsverfahren für Schaltnetze

Ein bekanntes Verfahren zur Lösung von Aufgaben mit Schaltnetzen ist die empirische Aufstellung der Schaltgleichung aus einer Schalttabelle. Dies demonstriert nachfolgendes Standardbeispiel:

Ein Tunnel wird über zwei Lüfter versorgt, welche von drei Sensoren gesteuert werden.
Diese messen die Luftqualität und sprechen bei Überschreitung von Grenzwerten mit Signal FALSE (!) an



Theoretisch müssen bei Aufstellung der Schalttabelle die möglichen 8 Kombinationen der drei Sensoren betrachtet werden. Mitunter kann man aber empirisch die wenigen Kombinationen finden, bei denen überhaupt Lüfter laufen müssen.

B3	B2	B1	K1	K2	Zeile
0	0	0	1	1	0
0	0	1	0	1	1
0	1	0	1	1	2
0	1	1	1	0	3
1	0	0	0	1	4
1	0	1	1	1	5
1	1	0	1	0	6
1	1	1	0	0	7

Schalttabelle für Lüfter K1 und K2

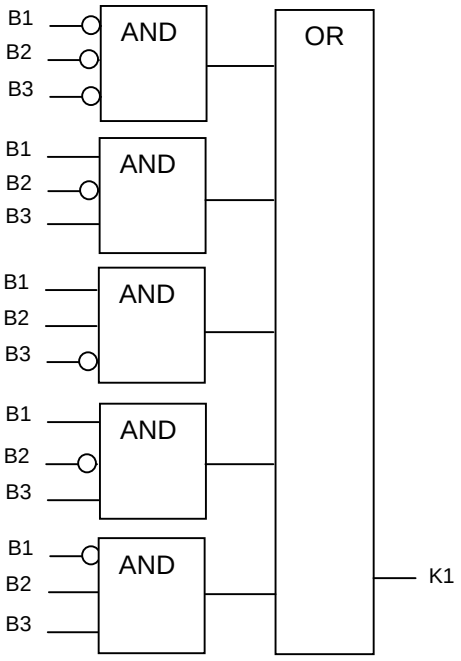
Liegt die Belegung der Ausgangssignale in der Schalttabelle fest, wird z.B. die **ODER-Normalform (Disjunktive Normalform)** der Ausgangssignale aufgeschrieben.

Dazu sind alle Zeilen, in denen das Ausgangssignal den Wert 1 (TRUE) aufweist, über ODER-Logik verbunden. An die einzelnen ODER-Glieder werden die Eingangssignale mit ihren tatsächlichen Belegungen über UND-Verknüpfungen angeschaltet.

Daneben könnte man auch eine UND-Normalforn (Konjunktive Normalform) nutzen.

An dieser Stelle nun könnte ein Minimierungsverfahren angesetzt werden, was sich aber bei Realisierung mit einem SPS-Programm kaum mehr lohnt.

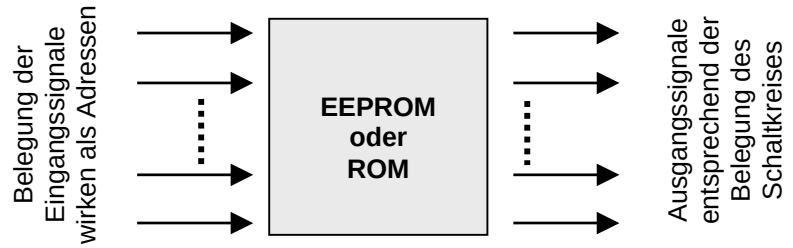
Das Ergebnis für Lüfter K1 ohne Minimierung ist die Schaltgleichung bzw. der nebenstehende Funktionsplan.



Schaltgleichung und FUP für Lüfter K1 in Disjunktiver Normalform:

K1=/B1./B2./B3 + B1./B2.B3 + B1.B2./B3 + B1./B2.B3 + /B1.B2.B3

Größere **Schaltnetze können mit Speicherschaltkreisen realisiert** werden.
Die Kombination der möglichen Eingangssignale wird als Adresse interpretiert, und der Schaltkreis gibt die gewünschte Kombination von Ausgangssignalen aus.



Beispiel: Die Belegung der Sensoren 2#1 0 0 von Zeile 4 der Schalttabelle müsste als Adresse für eine Speicherzelle dienen, von der dann die Belegung 2# 0 1 für die Lüfter K1 und K2 ausgegeben wird. Es ist ersichtlich, dass sich dieses Verfahren nur bei Schaltnetzen mit einer Vielzahl von Eingangs- und Ausgangssignalen loht.

5.4 Speicherfunktionen

Um Schaltwerke (sequentielle Netzwerke) zu realisieren, benötigt man zusätzlich zu den logischen Funktionen auch Speicherelemente. In der Technik werden dazu **R-S-Flip-Flop** verwendet. Diese werden auch als Bistabile Funktionsblöcke bezeichnet.

Das Ein- und Ausschalten des Speicherelementes erfolgt durch Setzen und Rücksetzen. Unabhängig vom Setzsignal bleibt ein einmal gesetzter Ausgang solange auf Wert TRUE, bis er zurückgesetzt wird.

Auch Speicher wurden ursprünglich durch Kontaktschaltungen realisiert. Das entscheidende Element ist hierbei der **Selbsthaltekontakt**, welcher eine **Selbsthaltung** der Relaisspule ermöglicht.

Wenn des Setz- und das Rücksetzsignal gleichzeitig anstehen, entscheidet die Art der Schaltung, ob **dominierend gesetzt** oder **dominierend zurückgesetzt** wird. Mehr als 95% aller Anwendungsfälle der Automatisierungstechnik verlangen ein dominierendes AUS (Rücksetzen).

In der aktuellen SPS-Technik gibt es für die Realisierung von Speicherfunktionen die Operationen Setzen (S) und Rücksetzen (R).

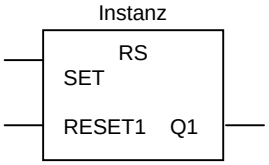
Auszug der CoDeSys AWL Referenzliste :

S	Setzt den Operanden (Typ BOOL) auf TRUE, wenn der Akkumulatorinhalt TRUE ist.
R	Setzt den Operanden (Typ BOOL) auf FALSE, wenn der Akkumulatorinhalt FALSE ist.

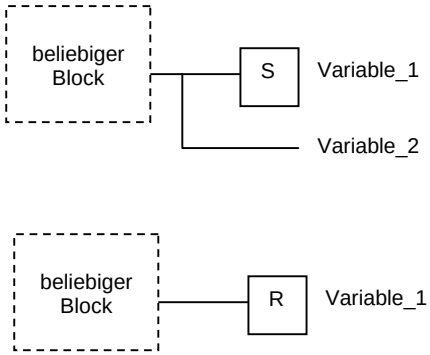
Je nachdem, ob Setzen vor oder nach dem Rücksetzen programmiert wird, entstehen durch Wirkung der zyklischen und seriellen Programmbearbeitung die dominierenden Eigenschaften.

Die Setz- und Rücksetzoperationen können graphisch in FUP und KOP kombiniert in einem R-S-FlipFlop oder aber als Einzeloperationen genutzt werden.

Beispiel CoDeSys:



Setz- und Rücksetzbefehl für Variable_1 und alternativ Zuweisung an Variable_2



In IEC-Programmiersystemen sind Flip-Flops als bistabile Funktionsblocks der Standardbibliothek zu entnehmen und zu instanzieren. Im Detail wird dieses Thema im Abschnitt 9 „Funktionsbausteine“ behandelt.

Die graphischen Setz- und Rücksetzbefehle werden in Step7 sehr ähnlich genutzt. In AWL-Darstellung bestehen erhebliche Unterschiede.

Hinweis: **Drahtbruchsichere Programmierung:**
(hierzu Übersicht Realisierung von Speicherelementen auf nachfolgender Seite)

„Drahtbruchsichere Programmierung“ bedeutet Vorkehrungen zu schaffen, dass

- ein unterbrochener Leiter zwischen Aus-Befehlsgeber und SPS einen Aus-Befehl bewirkt
- ein unterbrochener Leiter zwischen Ein-Befehlsgeber und SPS einen Ein-Befehl verhindert.

Nach VDE-Vorschriften muss deshalb

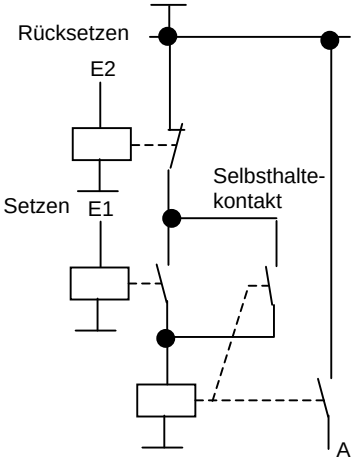
- ein Einschaltbefehl durch **Schalten eines Arbeitsstromes** gegeben werden
- ein **Ausschaltbefehl durch Unterbrechung eines Ruhestroms** gegeben werden, d.h. durch FALSE-Signal. In der Praxis erfolgt dies überwiegend durch **Betätigung eines Tasters mit Öffnerkontakt.**

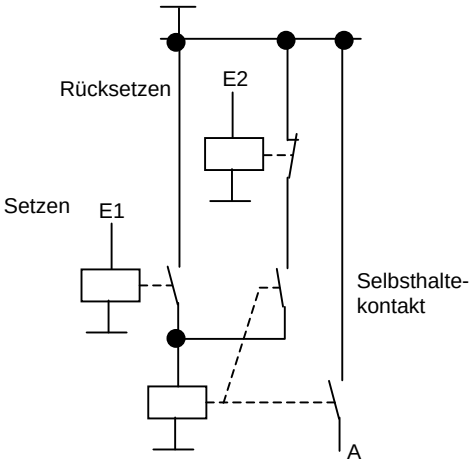
  Das Rücksetzen erfolgt nur bei Wert TRUE am Eingang RESET (bzw. bei Step7 mit Verknüpfungsergebnis VKE=1 am Eingang R. Deshalb muss in den allermeisten Fällen das **Rücksetzsignal über eine Negation angeschaltet** werden!

Realisierung von Speicherelementen (Flip-Flop):

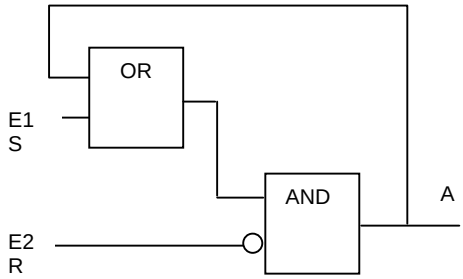
Dominierend Rücksetzen (Dominierend AUS) **Dominierend Setzen (Dominierend EIN)**

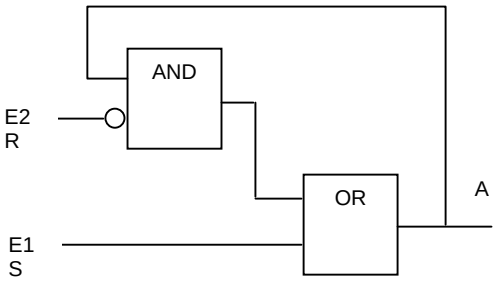
Kontaktschaltungen



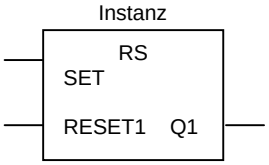


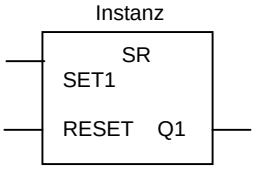
Speicherelemente mit Logik und Rückführungen





Darstellung der Flip-Flops nach IEC: Die Zuordnung der Ziffer 1 bestimmt dominierendes Setzen oder Rücksetzen

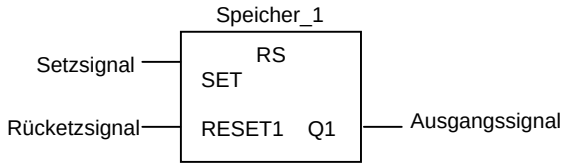




Darstellung der Flip-Flops in Step7: Die Reihenfolge bei der gedachten Abarbeitung des Zyklus bestimmt dominierendes Setzen oder Rücksetzen

AWL eines R-S-Flip-Flops: Der Name der Instanz des RS-Speichers sei Speicher_1.

VAR
Speicher_1:RS;
Setzsignal:BOOL;
Ruecksetzsignal:BOOL;
Ausgangssignal:BOOL;
END_VAR




```
CAL Speicher_1(SET:=Setzsignal, RESET1:=Ruecksetzsignal)
LD Speicher_1.Q1
STAusganssignal
```

zum Vergleich Step7 mit symbolischen Operanden:
U “Setzsignal”
S “Ausgangssignal”
U “Ruecksetzsignal”
R “Ausgangssignal”

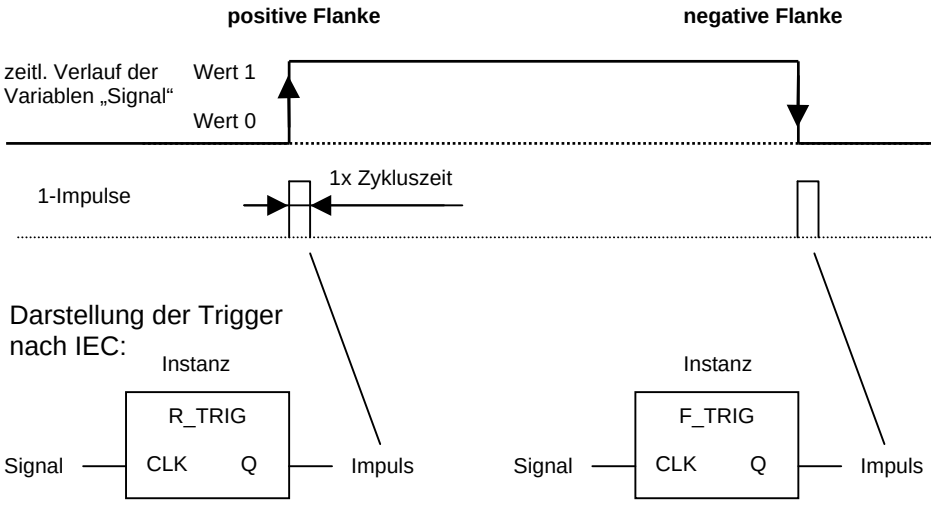
Beim Aufruf der Instanz mit dem Befehl CAL können INPUT- Parameter übergeben werden.
OUTPUT - Parameter müssen dagegen im Programm unabhängig vom Aufruf behandelt werden.
Es ist auch möglich, einzelne oder alle INPUT-Parameter vor dem Aufruf zuzuweisen.

5.5 Triggerfunktionen

Zu den Bitoperationen gehören weiter die Triggerfunktionen, auch als **Flankenauswertung** bezeichnet. Sie werden dann benötigt, wenn nicht das Signal an sich, sondern nur die **Signaländerung** eine Operation auslösen soll. Sie werden weiter eingesetzt, wenn dauerhaft anstehende Signale die Funktion eines Programms stören.

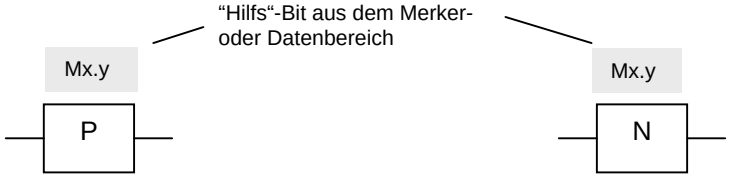
Das Ausgangssignal der Trigger ist jeweils ein **Impuls von Zykluszeit**. Er kann genutzt werden, um z.B. Variablen zu setzen oder zurückzusetzen. Der Impuls wird mitunter auch als **1-Impuls** bezeichnet, weil er nur einen Programmzyklus lang zur Verfügung steht.

Realisierung von Triggern:



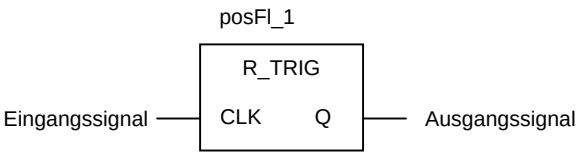
Darstellung der Trigger nach IEC:

Darstellung in Step7:



Die Trigger werden wiederum als Funktionsbausteine der Standardbibliothek entnommen und instanziert. Untenstehendes Beispiel zeigt die Vorgehensweise in AWL. Die Instanz des Triggers für steigende Flanke sei posFl_1.

```
VAR
posFl_1:R_TRIG;
Eingangssignal:BOOL;
Ausgangssignal:BOOL;
END_VAR
```




```
CAL posFl_1(CLK:=Eingangssignal)
LD posFl_1.Q
ST Ausganssignal
```

zum Vergleich Step7:
U “Eingangssignal”
FP “Hilfsmerker” (Hilfsbit aus dem Merker oder Datenbereich)
= “Ausgangssignal”

Einige Hinweise zum eigenständigen Programmieren von 1-Impulsen:

Fall 1: 1-Impuls von Zykluszeit bei Übergang der Steuerung von STOP auf RUN

```
LDN Variable_1      (*Variable mit Anfangswert FALSE*)
ST  Variable_2      (*1-Impuls*)
S   Variable_1
```

Fall 2: 1-Impuls von Zykluszeit bei positiver Signalflanke der Variablen mit Bezeichner Eingangssignal (kann Trigger ersetzen, analog auch für negative Signalflanke)

```
LD    Eingangssignal
ANDN Variable_1     (*Variable mit Anfangswert FALSE*)
ST    Variable_2    (*1-Impuls bei positiver Signalflanke*)
S     Variable_1
LDN  Eingangssignal
R     Variable_1
```

5.6 Hinweise

Während es für Schaltnetze brauchbare Entwurfsverfahren gibt, ist man bei Schaltwerken häufig auf Intuition und Erfahrung angewiesen.
Einige Empfehlungen zum Entwurf sollen hier angeführt werden.

Es kann erfolgreich sein, den Entwurf von den Ausgangssignalen hin zu den Eingangssignalen (de facto „rückwärts denkend“) zu gestalten:

- Welche Ausgangssignale sind zu steuern?
- Müssen – da die Eingangssignale nur kurzzeitig anstehen - RS-Speicher eingesetzt werden oder stehen die Signale z.B. durch rastende Schalter dauerhaft zur Verfügung. Im letzteren Fall genügen logische Verknüpfungen (führt zu Schaltnetzen).
- Erforderliche Anzahl Speicher anlegen und Ausgangssignale anschalten
- Welche Signale setzen einen Speicher? Sind dies mehrere Signale, dann sind diese zumeist UND-verknüpft anzuschalten!
- Welche Signale setzen Speicher zurück? Sind dies mehrere Signale, dann sind diese zumeist ODER-verknüpft anzuschalten!