

SPS verstehen und anwenden

Ein Lernheft für deinen Einstieg: vom Sensor bis zum ersten getesteten Programm.

Was bedeutet SPS?

Speicherprogrammierbare Steuerung heißt: Eine Steuerung entscheidet anhand eines gespeicherten Programms, was eine Anlage tun soll. Englisch heißt sie PLC (Programmable Logic Controller). Sie erfasst Eingangssignale, verarbeitet Bedingungen und beeinflusst über Ausgänge den Prozess.



Ein Beispiel aus der Werkstatt

Ein Sensor erkennt ein Werkstück. Eine Betriebsfreigabe ist aktiv. Nur wenn **beide** Bedingungen erfüllt sind, leuchtet die Meldelampe. Die SPS verarbeitet dazu zwei Informationen und berechnet einen Ausgangswert. Später könntest du eine Verzögerung oder einen Stückzähler ergänzen.

Woraus besteht eine SPS?

Teil	Aufgabe
CPU	Führt das Programm aus und koordiniert die Verarbeitung.
Speicher	Enthält Programm, Einstellungen und Variablen.
Ein-/Ausgänge (I/O)	Verbinden die Steuerung mit Sensoren und Aktoren.
Versorgung und Schnittstellen	Versorgen das Gerät und ermöglichen Programmierung sowie Kommunikation.

Das Ziel: Du kannst anschließend den Signalweg erklären, einfache KOP-Logik lesen und eine Start-Stopp-Funktion in einer Simulation aufbauen und systematisch prüfen.

Dein Weg: Ausstattung (2) · Signale & Bits (3) · Zyklus (4) · Kontaktplan (5) · Selbsthaltung (6) · Zeit & Zählen (7) · Erstes Projekt (8) · Prüfen & Fehlersuche (9) · Übungen (10).

Was brauche ich dafür?

Erst die Aufgabe verstehen, dann passende Software und Hardware auswählen.

Zum Lernen: PC und Simulation

Eine geeignete SPS-Entwicklungsumgebung mit Simulator oder Soft-SPS reicht für den Anfang. Du kannst Variablen schalten, ein Programm ausführen und die Ergebnisse beobachten, ohne eine Maschine anzuschließen. Nicht jede Umgebung enthält dieselben Simulationsfunktionen; Gerätetyp, Softwareversion und Lizenz müssen passen.

Für einen realen Übungsaufbau

Du brauchst	Darauf achten
SPS mit I/O	Genügend Eingänge und Ausgänge; passende Signalarten und Schnittstellen.
Passende Versorgung	Bei vielen kleinen DC-Systemen 24 V DC. Maßgeblich sind immer die Daten des konkreten Geräts.
Taster / Sensoren	Signalpegel, PNP/NPN, Bezugspotential und Anschlussbild müssen zum Eingang passen.
Signallampe / Aktor	Ausgangstyp, Laststrom, Spannung und ggf. Schutzbeschaltung prüfen.
Aufbau und Verbindung	Geeignete Leitungen, Klemmen, Absicherung, geschützter Aufbau sowie die vorgesehene Programmierschnittstelle.
Unterlagen	Hardwarehandbuch, Schaltplan, I/O-Liste und Beschreibung des gewünschten Ablaufs.

Was brauchst du zunächst nicht?

Ein HMI-Bedienpanel, ein großer Schaltschrank oder ein echter Motor sind für die erste Logikübung nicht erforderlich. Eine simulierte Lampe macht das Ergebnis bereits sichtbar. Ein HMI dient später dem Anzeigen, Eingeben und Bedienen; es ersetzt die Steuerungslogik nicht automatisch.

Last und Sicherheit: Ein Motor wird üblicherweise über ein Schütz oder einen geeigneten Antrieb angesteuert. Die Leistung fließt nicht durch einen beliebigen kleinen SPS-Ausgang. Reale Verdrahtung nur nach Herstellerunterlagen und Schaltplan durch entsprechend qualifizierte Personen.

Für Lernversuche zuerst die Simulation verwenden. Die Auswahl einer Sicherheitssteuerung und die Auslegung von Sicherheitsfunktionen sind eine eigene technische Aufgabe. Eine normale Programmbedingung ist noch keine Not-Halt-Funktion.

Signale, Adressen und Bits

Die SPS arbeitet mit Informationen. Die elektrische Signalart und die Bedeutung im Programm gehören zusammen.

Digital oder analog?

Digital: Zwei Zustände, etwa Werkstück erkannt / nicht erkannt. Im Programm meistens BOOL mit FALSE (0) und TRUE (1). Eine logische 1 ist eine Information; die dafür erforderliche Eingangsspannung ergibt sich aus dem Modulhandbuch.

Analog: Ein Wertebereich, z. B. Druck als 4–20-mA-Signal oder Position über 0–10 V. Das Eingangsmodul liefert einen Rohwert, der passend zum Messbereich in eine physikalische Größe umgerechnet wird. Beispiel: Bei idealer linearer Skalierung von 4–20 mA auf 0–10 bar entsprechen 12 mA genau 5 bar.

Mit sinnvollen Namen arbeiten

Name	Beispiel	Bedeutung
Start	I0.0 / E0.0	Eingangsbit eines Starttasters
Stop_OK	I0.1 / E0.1	1 bedeutet: Stopptaster nicht betätigt
Betrieb	Interne BOOL-Variable	Merkt den Betriebszustand
Lampe	Q0.0 / A0.0	Ausgangsbit der Meldelampe

I/E = Eingang, Q/A = Ausgang; M steht in manchen Systemen für Merker. Adresssyntax und Zuordnung sind herstellerabhängig. I0.0 bezeichnet beispielhaft Bit 0 im Eingangsbyte 0. Nach I0.7 folgt I1.0. Die Namen sollten sagen, **was 1 bedeutet**.

Warum Hex und Binär zur SPS passen

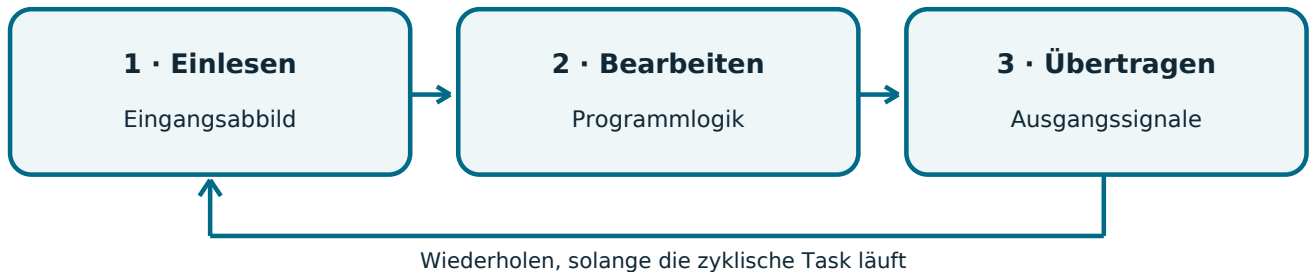
Ein Byte besteht aus 8 Bit. Bit 0 hat den Wert 1, Bit 1 den Wert 2, Bit 2 den Wert 4 usw. **0010 1010 binär = $32 + 8 + 2 = 42$ dezimal = 2A hex**. Eine Hex-Ziffer entspricht vier Bits; A bis F bedeuten 10 bis 15.

Datentyp / Deutung	Typischer Bereich
BYTE: 8-Bit-Muster	00 bis FF in Hex
UINT: 16 Bit ohne Vorzeichen	0 bis 65535
INT: 16 Bit mit Vorzeichen	-32768 bis 32767

Datentyp beachten: 1111 1111 bedeutet bei 8 Bit entweder 255 ohne Vorzeichen oder -1 im Zweierkomplement. Die Bits sind gleich, die Interpretation ist anders. Breite und Typ im jeweiligen SPS-System prüfen.

Wie arbeitet der SPS-Zyklus?

Ein übersichtliches Grundmodell: lesen, verarbeiten, übertragen, wiederholen.



1 · Eingänge einlesen

Die Steuerung übernimmt die Eingangszustände in ein **Eingangsabbild**. Stell es dir wie ein Foto der Signale vor: Das Programm verwendet diese Momentaufnahme. Eine spätere Änderung am Sensor ändert in diesem Modell nicht rückwirkend das bereits aufgenommene Bild.

2 · Programm ausführen

Das Programm verarbeitet die Anweisungen in der festgelegten Reihenfolge. Es prüft Kontakte, rechnet, aktualisiert Variablen und berechnet Ausgangswerte. Eine Variable kann innerhalb dieser Bearbeitung schon einen neuen Wert erhalten. Mehrere Zuweisungen an denselben Ausgang machen das Verhalten schnell unübersichtlich.

3 · Ausgänge übertragen

Die berechneten Ausgangswerte werden an die Ausgangshardware übergeben. Erst jetzt reagiert im Lernmodell die Lampe. Anschließend beginnt der nächste Zyklus.

Ein Sensor ändert sich mitten im Zyklus

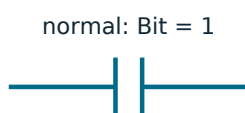
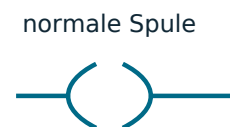
Zeitpunkt	Was geschieht?
Einlesen	A = 1 und B = 1 werden gespeichert.
Danach	Sensor A fällt auf 0. Das Eingangsabbild enthält noch A = 1.
Programm	A-Abbild UND B-Abbild ergibt weiterhin 1.
Übertragen	Die Lampe wird eingeschaltet.
Nächster Zyklus	A = 0 wird gelesen; die UND-Bedingung wird 0.

In der Praxis: Zykluszeit, Task-Aufruf und I/O-Aktualisierung hängen vom System ab. Es gibt z. B. feste zyklische Tasks, Ereignisse, Buszyklen und direkte I/O-Zugriffe. Das Dreischrittmodell erklärt die Grundidee, nicht jede interne Reihenfolge jedes Herstellers.

Ein sehr kurzer Impuls kann zwischen zwei Abtastungen auftreten und übersehen werden. Für schnelle Signale können spezielle Eingänge, Interrupts oder schnelle Zähler nötig sein. Auf LehrLynx: „SPS verstehen“ → „SPS-Zyklus“ selbst durchschalten.

Kontaktplan richtig lesen

KOP (englisch LD) ist eine grafische Programmiersprache mit logischen Kontakten und Spulen.



Element	Bedeutung im Programm
Normaler Kontakt	Prüft, ob die zugeordnete BOOL-Variable 1 ist.
Negierter Kontakt	Prüft, ob die zugeordnete BOOL-Variable 0 ist.
Kontakte in Reihe	UND: Alle Bedingungen müssen erfüllt sein.
Parallele Zweige	ODER: Mindestens ein Zweig muss erfüllt sein.
Normale Spule	Schreibt bei jeder Ausführung das Ergebnis 1 oder 0 in die Variable.

Hardwarekontakt und Programmkontakt unterscheiden

Ein **physischer Schließer** schließt beim Betätigen seinen Stromkreis. Ein **physischer Öffner** öffnet ihn. Welcher Wert am SPS-Eingang entsteht, hängt von der Verdrahtung und der Eingangsschaltung ab.

Ein **KOP-Kontakt** bewegt sich dagegen nicht. Er fragt einen gespeicherten Wert ab. Im Kontaktplan fließt kein Laststrom zur Lampe. Die Schienen symbolisieren logischen Signalfluss; sie sind kein Netzanschlussplan mit L1 und N.

Der entscheidende Unterschied: Ein als Öffner verdrahteter Stopptaster liefert in unserem Beispiel unbetätigt 1. Diese Variable heißt Stop_OK. Damit der Betrieb erlaubt ist, fragen wir Stop_OK mit einem **normalen KOP-Kontakt auf 1** ab. Ein negierter Kontakt würde auf 0 prüfen und die Bedeutung umkehren.

UND und ODER auf einen Blick

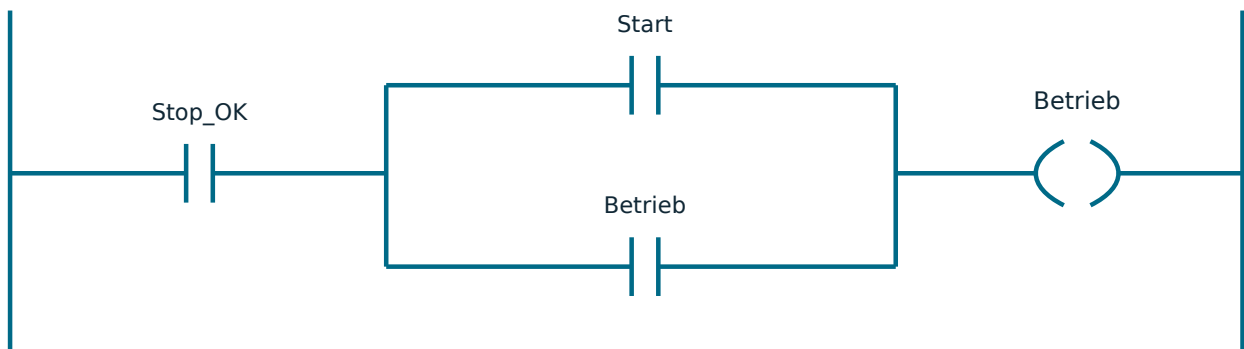
A	B	A UND B	A ODER B
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Weitere Darstellungen: FUP/FBD verknüpft Funktionsblöcke; ST schreibt die Logik als Text. SCL ist eine herstellerspezifische Bezeichnung für eine ST-nahe Sprache. Unterschiedliche Darstellungen können dieselbe Funktion ausdrücken.

Start, Selbsthaltung, Stopp

Ein vollständiges Beispiel: Eine Betriebslampe soll sich einen kurzen Startimpuls merken.

Aufgabe: Start schaltet die Lampe ein. Nach dem Loslassen bleibt sie ein. Stopp schaltet sie aus. Stopp soll Vorrang haben. Beim Neustart beginnt Betrieb in diesem Beispiel mit FALSE; der Wert wird nicht remanent gespeichert.



Logische Abfragen und Spule; kein Verdrahtungsplan

```
Betrieb := Stop_OK AND (Start OR Betrieb);  
Lampe := Betrieb;
```

So funktioniert die Selbsthaltung

Vor dem Start: Start = 0, Stop_OK = 1, Betrieb = 0. Der parallele Zweig ist falsch, die Lampe bleibt aus.

Start drücken: Start wird 1. Die Gesamtbedingung wird wahr, Betrieb wird auf 1 gesetzt.

Start loslassen: Start wird 0, aber Betrieb ist bereits 1 und hält den parallelen Zweig wahr.

Stopp drücken: Stop_OK wird 0. Die gesamte UND-Bedingung wird falsch, Betrieb und Lampe werden 0.

Was bedeutet Vorrang?

Sind Start = 1 und Stop_OK = 0 gleichzeitig vorhanden, ergibt die Formel 0. Stopp wirkt deshalb auch bei gedrücktem Start. Ein normaler KOP-Kontakt für Stop_OK ist hier fachlich richtig, obwohl der echte Stopptaster ein Öffner ist.

Grenze dieses einfachen Beispiels: Bleibt Start dauerhaft 1, kann die Lampe nach dem Loslassen von Stopp wieder einschalten. Soll zwingend eine neue Startbetätigung erforderlich sein, muss diese Bedingung zusätzlich entworfen werden, z. B. mit geeigneter Flankenbewertung und Wiederanlauflogik.

Keine Sicherheitsfunktion: Eine gewöhnliche SPS-Selbsthaltung ersetzt weder Not-Halt noch Schutztürüberwachung. Ein Öffner kann bei passender Verdrahtung auch bei einer Leitungsunterbrechung 0 liefern; das allein ist jedoch keine vollständige Sicherheitslösung.

Zeit und Stückzahl verarbeiten

Bausteine besitzen Zustand. Jede benötigte Funktion erhält eine eigene, zyklisch aufgerufene Instanz.

TON: erst nach einer Verzögerung einschalten

Beispiel: Eine Meldung erscheint erst, wenn Freigabe 3 Sekunden ununterbrochen wahr ist. **IN** ist das Startsignal, **PT** die vorgegebene Zeit, **ET** die abgelaufene Zeit und **Q** das Ergebnis.

Verhalten	Ergebnis
IN wechselt von 0 auf 1	Die Verzögerung beginnt. Q bleibt zunächst 0.
IN bleibt 1 bis $ET \geq PT$	Q wird 1.
IN wird wieder 0	Q wird 0, ET wird zurückgesetzt.

```
Verzoegerung(IN := Freigabe, PT := T#3s);
Meldung := Verzoegerung.Q;
```

Die Variable Verzoegerung ist zuvor als Instanz vom Typ TON zu deklarieren. Der Baustein wird im Zyklus aufgerufen. Eine neue Instanz bei jeder Bearbeitung würde den benötigten Zustand nicht erhalten.

CTU: ein Werkstück einmal zählen

Eine **steigende Flanke** ist der erkannte Wechsel von 0 auf 1. CTU erhöht seinen Zählerstand bei dieser Flanke. Ein dauerhaftes Signal 1 darf nicht bei jeder Programmausführung ein neues Teil ergeben.

Parameter	Bedeutung
CU	Zähleingang; steigende Flanke zählt.
RESET / R	Setzt den Zähler zurück; Bezeichnung systemabhängig.
PV / CV	Vorgabewert / aktueller Zählerstand.
Q	Wahr, wenn CV den Vorgabewert erreicht oder überschreitet.

```
Teile(CU := Sensor, RESET := Reset, PV := 3);
ZielErreicht := Teile.Q;
```

Das CTU-Beispiel verwendet CODESYS-Parameternamen. Teile ist eine CTU-Instanz. Grenz- und Sättigungsverhalten unterscheiden sich je nach Baustein. Im LehrLynx-Modell wird nur bis $PV = 3$ gezählt; Reset hat Vorrang. Ein nach Reset noch anliegendes $CU = 1$ ist keine neue Flanke.

Probiere es aus: Sensor auf 1 setzen und mehrere Zyklen ausführen. Es wird einmal gezählt. Danach 0 einlesen, wieder 1 einlesen: Jetzt wird erneut gezählt. Kontaktprellen kann zusätzliche echte Flanken verursachen.

Dein erstes SPS-Projekt

Ein Ablauf, den du auf die passende Entwicklungsumgebung übertragen kannst.

1 bis 4: vorbereiten

- 1. Aufgabe festlegen:** Verwende die Betriebslampe von Seite 6. Schreibe Start-, Stopp- und Neustartverhalten auf.
- 2. Projekt anlegen:** Passendes Zielgerät oder Simulationsziel wählen. Den benötigten zyklischen Programmaufruf vorsehen.
- 3. Variablen anlegen:** Start, Stop_OK, Betrieb und Lampe sind BOOL. Betrieb startet mit FALSE.
- 4. I/O zuordnen:** In der ersten Simulation genügen Testvariablen. An Hardware werden Variablen entsprechend der realen I/O-Konfiguration verknüpft.

Deklaration und Logik getrennt betrachten

```
VAR
Start : BOOL := FALSE;
Stop_OK : BOOL := FALSE;
Betrieb : BOOL := FALSE;
Lampe : BOOL := FALSE;
END_VAR
```

Dies ist eine beispielhafte lokale ST-Deklaration für eine Simulation, keine universelle Adressierung realer Ein-/Ausgänge. Stop_OK wird erst beim Test bewusst auf TRUE gesetzt. Ein Hardwareeingang muss seinen tatsächlichen Wert liefern: ihn nicht durch einen dauerhaft gesetzten Ersatzwert umgehen.

```
Betrieb := Stop_OK AND (Start OR Betrieb);
Lampe := Betrieb;
```

5 bis 8: ausführen und prüfen

- 5. Programm schreiben:** Entweder als KOP oder als obige ST-Logik. Beide Anweisungen in dieser Reihenfolge ausführen. Nur eine klare Schreibstelle für Betrieb und Lampe vorsehen.
- 6. Übersetzen:** Syntaxfehler, falsche Datentypen und fehlende Zuordnungen beheben.
- 7. Simulation starten:** Verbinden, Programm laden, zyklische Task prüfen und RUN aktivieren. In STOP wird die zyklische Logik nicht normal abgearbeitet.
- 8. Beobachten:** Stop_OK auf 1 setzen. Start einen Zyklus lang auf 1, dann wieder 0. Lampe muss eingeschaltet bleiben. Danach Stop_OK auf 0: Lampe muss ausgehen.

Wenn du TON oder CTU ergänzt: Zusätzlich z. B. Verzögerung : TON; oder Teile : CTU; deklarieren. Die Instanzen in der zyklischen Logik aufrufen. Namen, Bibliothekseinbindung und Einfügeort hängen von der Umgebung ab.

Beobachten liest Werte. Schreiben verändert sie. Forcen kann normale Programm- oder I/O-Werte übersteuern. Solche Eingriffe an einer realen Anlage verlangen ein kontrolliertes Vorgehen und dürfen keine Fehlersuche durch Schutzumgehung ersetzen.

Prüfen statt raten

Ein Testplan macht aus „geht irgendwie“ eine nachvollziehbare Funktion.

Diese Fälle muss die Betriebslampe bestehen

Test	Aktion	Erwartetes Ergebnis
1	Neustart; Start = 0	Betrieb = 0, Lampe aus
2	Stop_OK = 1; kurzer Startimpuls	Lampe ein
3	Start wieder 0	Lampe bleibt ein
4	Stop_OK = 0	Lampe aus
5	Stop_OK wieder 1, Start = 0	Lampe bleibt aus
6	Start = 1 und Stop_OK = 0	Lampe aus; Stopp hat Vorrang

Zusätzlich das Verhalten bei dauerhaftem Start testen und dokumentieren. Die einfache Selbsthaltung schaltet nach Rückkehr von Stop_OK wieder ein, wenn Start noch 1 ist. Ob das zulässig bzw. gewünscht ist, entscheidet die reale Aufgabenstellung einschließlich Wiederanlaufanforderungen.

Fehlersuche entlang des Signalwegs

Beobachtung	Nächster sinnvoller Prüfschritt
Sensor reagiert, Eingangsbit nicht	Versorgung, Verdrahtung, Signaltyp, Bezugspotential und Kanaluordnung.
Eingangsbit stimmt, Logik nicht	Richtige Variable? Negation vertauscht? Fehlt eine weitere Freigabe?
Logik stimmt, Ausgangswert nicht	RUN/Task-Aufruf, Reihenfolge und andere Schreibstellen prüfen.
Ausgangsbit 1, Lampe aus	I/O-Mapping, Moduldiagnose, Lastversorgung und Last.
Zähler zählt mehrfach	Flankenbewertung, Kontaktprellen und Signalqualität.
Zeitglied beginnt immer neu	IN-Unterbrechungen, Instanz und zyklischen Aufruf prüfen.

Was du dokumentierst

Notiere Aufgabe, Variablenliste, Signalbedeutung, Programmversion, Startwerte und Testergebnisse. Speichere ein nachvollziehbares Projekt. So kann jemand anderes verstehen, warum eine Ausgangsvariable 1 oder 0 wird.

Merksatz: Erst den realen Zustand prüfen, dann das Eingangssignal, die Programmbedingung, das Ausgangssignal und schließlich den Aktor. Nicht sofort die Programmlogik ändern.

Jetzt selbst erklären

Löse die kurzen Aufgaben zuerst ohne die Antworten. Übertrage die Beispiele danach in die LehrLynx-Labore.

Fünf Aufgaben

1. Ein Öffner-Stopptaster liefert unbetätigt 1. Welche KOP-Abfrage erlaubt den Betrieb?
2. Sensor und Freigabe sind beide 1. Nach dem Einlesen fällt der Sensor auf 0. Welchen Wert verwendet der aktuelle Lernzyklus?
3. TON hat $PT = 3\text{ s}$. IN bleibt 2 s auf 1 und fällt dann auf 0. Wird Q wahr?
4. CTU sieht die Folge 0, 1, 1, 1, 0, 1. Wie viele steigende Flanken gibt es?
5. Ein Byte lautet 0000 0100. Welches Bit ist gesetzt, und wie lauten Dezimal- und Hexwert?

Antworten mit Begründung

1. Ein normaler Kontakt auf Stop_OK = 1. Der Hardware-Öffner bestimmt das Eingangssignal; der Programmkontakt prüft es.
2. Im aktuellen Eingangsabbild bleibt Sensor = 1. Die Änderung wird in diesem Modell beim nächsten Einlesen sichtbar.
3. Nein. Die Zeit wurde nicht erreicht. IN = 0 setzt ET und Q zurück.
4. Zwei: jeweils beim Übergang 0 → 1. Ein gehaltenes Signal zählt nicht pro Zyklus weiter.
5. Bit 2; dezimal 4; hexadezimal 04. Die Bitnummerierung beginnt rechts bei 0.

Auf LehrLynx weiterüben

Interaktiv → SPS verstehen: Zyklus, Selbsthaltung, TON und CTU.

Interaktiv → SPS-Kontaktplan-Trainer: Kontakte selbst auswählen und alle Signalzustände prüfen.

Rechner → Zahlensysteme: Binär, Dezimal, Hex, Oktal und Zweierkomplement.

Rechner → Bit- & Hex-Rechner: Maskieren, Setzen, Löschen, Schieben und Rotieren.