

# NILAB ST-zu-IL-Compiler - Simulator

## NILAB uPLC — Structured-Text-zu-IL-Konverter

Online-Werkzeug zur Programmierung von NILAB-Integriertlinearmotoren (NLI-Familie) mit IEC-61131-3-Structured-Text-Syntax. Das Werkzeug kompiliert ST-Quellcode in das Instruction List (IL)-Format, kompatibel mit der NILAB-μPLC-Firmware ( $\geq 5E41$ ).

**Zugriff auf das Werkzeug:** [ni-lab.online](#) → [Werkzeuge](#) → [ST-Konverter](#)

### 1. Übersicht

Die NILAB-μPLC ist eine eingebettete Soft-SPS, die in jedem NLI-Integriertlinearmotor läuft. Sie führt ein Programm in Instruction List (IL)-Format aus — eine niedrige, zeilenweise Sprache ähnlich Assembler. Das direkte Schreiben von IL ist fehleranfällig und schwer zu lesen. Der ST-zu-IL-Konverter ermöglicht es Ihnen, Programme in **Structured Text (ST)** zu schreiben, einer Hochsprache nach IEC 61131-3 ähnlich Pascal, und kompiliert sie automatisch zu IL.

| Parameter              | Wert                                         |
|------------------------|----------------------------------------------|
| Erforderliche Firmware | $\geq 5E41$                                  |
| Maximale Anweisungen   | 64                                           |
| Scan-Zyklus            | 6 ms (fest)                                  |
| Stack-Tiefe            | 1 (keine verschachtelten AND/OR in einem IF) |
| Timer                  | 2 Hardware-Timer (R8028, R8029)              |

#### Wichtige Fakten über die NILAB-μPLC:

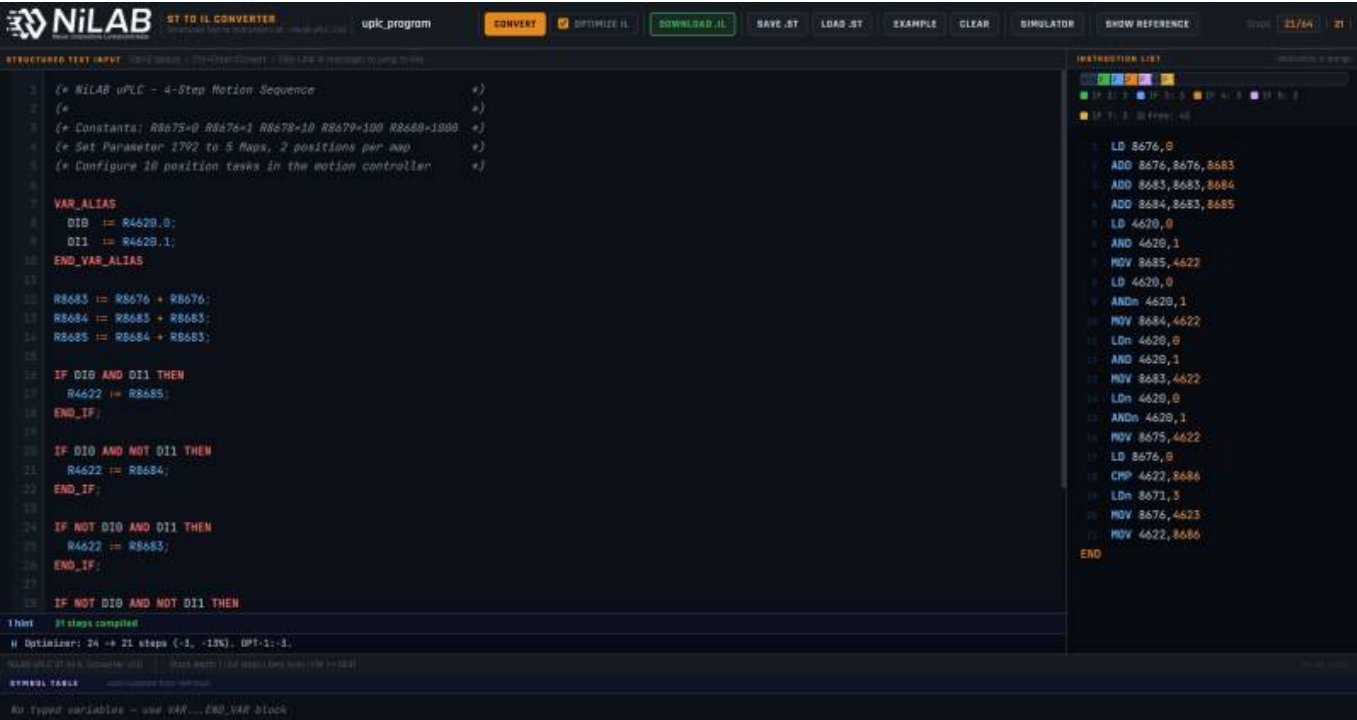


Abbildung 1: Hauptschnittstelle. Linke Spalte: ST-Editor. Rechte Spalte: erzeugter IL-Ausgang. Untere Panels: Symboltabelle und IL-Simulator (wenn aktiv).

## 2. Schnittstellen-Layout

Das Werkzeug ist in drei Hauptbereiche unterteilt:

- ① **Werkzeugleiste** — Projektname-Feld, Konvertieren-Schaltfläche, IL herunterladen, ST-Datei speichern/laden, Optimieren-Kontrollkästchen, Simulator-Umschalter, Referenzpanel zeigen/verbergen.
- ② **ST-Editor (linke Spalte, ~65 % Breite)** — Schreiben Sie Ihr Structured-Text-Programm hier.  
Merkmale:

| Element                                                                                 |
|-----------------------------------------------------------------------------------------|
| Syntaxhervorhebung (Schlüsselwörter rot, Register blau, Typnamen lila, Kommentare grau) |
| Zeilennummern mit Fehlermarkierungen im Rinnsal (rot ! = Fehler, gelb W = Warnung)      |
| Tab-Taste fügt 2 Leerzeichen ein                                                        |
| Strg+Enter löst die Konvertierung aus                                                   |
- ③ **IL-Ausgang (rechte Spalte, ~35 % Breite)** — Der kompilierte Instruction List erscheint hier nach Drücken von Konvertieren. Schrittnummern werden links grau angezeigt; das Zielregister von Mathematik-Anweisungen ist orange hervorgehoben.
- ④ **Meldungsbereich** — Unterhalb des ST-Editors. Zeigt Kompilierungsfehler (rot), Warnungen (gelb) und Optimierer-Hinweise (blau). Klicken Sie auf ein L##-Standort-Tag, um zu dieser Zeile im Editor zu springen.
- ⑤ **Symboltabelle** — Erscheint am unteren Rand der Seite, wenn ein VAR-Block vorhanden ist. Zeigt

alle deklarierten Variablen mit ihren automatisch zugewiesenen Registeradressen.

⑥ **IL-Simulator** — Erscheint unter der Symboltabelle, wenn die Simulator-Schaltfläche aktiviert ist. Ermöglicht schrittweise oder kontinuierliche Ausführung des kompilierten IL.

⑦ **Referenzpanel** — Standardmäßig ausgeblendet. Drücken Sie *Referenz zeigen* in der Werkzeugleiste, um eine Syntax-Quick-Reference-Seitenleiste zu öffnen.

## 3. ST-Sprachreferenz

### 3.1 VAR\_ALIAS-Block

Verwenden Sie VAR\_ALIAS, um lesbare Namen für Hardware-Register und Bitadressen zu vergeben. Dies sind einfache Textsubstitutionen — kein Register wird automatisch zugewiesen.

```
\
VAR_ALIAS\
DI0  := R4620.0;    (* Digital Input 0, Bit 0 von R4620 *)\
DI1  := R4620.1;    (* Digital Input 1, Bit 1 von R4620 *)\
ACTV := R8684.1;    (* Sequenzaktivitäts-Flag *)\
TIMER1 := R8028;    (* Timer-1-Register *)\
END_VAR_ALIAS\
```

### 3.2 VAR-Block (typisierte Variablen)

Verwenden Sie VAR, um typisierte Variablen zu deklarieren. Der Compiler **weist automatisch** das nächste freie Register aus dem Zuweisungspool zu — Sie müssen die Registeradressen nicht kennen.

```
\VAR
counter : INT := 0;    (* automatisch R8683 zugewiesen *)
enable  : BOOL;       (* automatisch R8681.0 zugewiesen *)
done    : BOOL := TRUE; (* automatisch R8681.1 zugewiesen, init *)
T1      : TON;        (* automatisch Timer1 R8028 zugewiesen *)
edge1   : R_TRIG;     (* automatisch 2 BOOL-Slots zugewiesen *)
\END_VAR
```

#### Unterstützte Typen:

| Typ   | IL-Registerpool                          | Max | Anfangswert         |
|-------|------------------------------------------|-----|---------------------|
| BOOL  | R8681 Bits 0-15, dann R8682 Bits 0-15    | 32  | 0/1/TRUE/FALSE      |
| INT   | R8683, R8684, R8685, R8686, R8687, R8688 | 6   | 0, 1, 10, 100, 1000 |
| DWORD | gleicher Pool wie INT                    | 6   | gleich              |
| TON   | R8028 (Timer1), R8029 (Timer2)           | 2   | -                   |
| CTD   | R8028 (Timer1), R8029 (Timer2)           | 2   | -                   |
| CTU   | ein INT-Slot für Zählerwert              | 16  | -                   |

|        |                                     |    |   |
|--------|-------------------------------------|----|---|
| R_TRIG | 2 BOOL-Slots (Speicher + Q-Ausgang) | 16 | - |
| F_TRIG | 2 BOOL-Slots (Speicher + Q-Ausgang) | 16 | - |

**Hinweis:** Konstanten R8675–R8680 sind schreibgeschützt und können nicht als Schreibziele verwendet werden. Der Compiler erzwingt dies mit einem Fehler.

### 3.3 Register-Konstanten

Die folgenden Konstantenregister sind immer ohne Deklaration verfügbar:

| Register | Wert | Verwendungsbeispiel             |
|----------|------|---------------------------------|
| R8675    | 0    | Register auf null zurücksetzen  |
| R8676    | 1    | Erhöhen, boolesches TRUE        |
| R8678    | 10   | Multiplizieren oder vergleichen |
| R8679    | 100  | Timer-Neuladewert (600 ms)      |
| R8680    | 1000 | Große Konstante                 |

Ganzzahliliterale 0, 1, 10, 100 und 1000 werden im Compiler automatisch diesen Registern zugeordnet. Jeder andere Literalwert erzeugt einen Kompilierungsfehler — laden Sie zuerst den Wert in ein Benutzerregister.

### 3.4 IF / ELSE / END\_IF

```
\
IF condition THEN\
statement(s);\
ELSE                (* optional *)\
statement(s);\
END_IF;\
```

#### Bedingungsoperatoren:

| Operator | IL-Anweisung     | Hinweise                      |
|----------|------------------|-------------------------------|
| AND      | AND / ANDn       | Serienkontakt                 |
| OR       | OR / ORn         | Parallelkontakt               |
| NOT      | LDn / ANDn / ORn | Invertiert den folgenden Term |

**Mischen von AND und OR** wird mit strikt links-nach-rechts-Auswertung unterstützt (keine Operatorpräzedenz):  $* A \text{ AND } B \text{ OR } C \rightarrow \text{IL: LD } A, \text{ AND } B, \text{ OR } C \rightarrow \text{Ergebnis: } (A \text{ AND } B) \text{ OR } C$   $* A \text{ OR } B \text{ AND } C \rightarrow \text{IL: LD } A, \text{ OR } B, \text{ AND } C \rightarrow \text{Ergebnis: } (A \text{ OR } B) \text{ AND } C$  **\* Eine Warnung** wird angezeigt, wenn AND und OR in derselben Bedingung gemischt werden.

**Stack-Tiefe = 1 Einschränkung:** Jede IF-Bedingung verwendet den einstufigen Stack. Sie können  $(A \text{ AND } B) \text{ OR } (C \text{ AND } D)$  nicht in einem IF verwenden — teilen Sie es in zwei IF-Blöcke auf oder verwenden Sie ein Zwischen-BOOL-Flag.

### 3.5 Bit-Anweisungen

```

SET(R4621.0);          (* Bit ON verriegeln *)\
RESET(R4621.0);        (* Bit OFF verriegeln *)\
R4621.0 := 1;          (* äquivalent zu SET *)\
R4621.0 := 0;          (* äquivalent zu RESET *)\

```

### 3.6 Register-Mathematik

```

(* Vier arithmetische Operationen *)
Rc := Ra + Rb;          (* IL: ADD Ra, Rb, Rc *)
Rc := Ra - Rb;          (* IL: SUB Ra, Rb, Rc *)
Rc := Ra * Rb;          (* IL: MUL Ra, Rb, Rc *)
Rc := Ra / Rb;          (* IL: DIV Ra, Rb, Rc – ganzzahlige Division *)

(* Kopieren *)
Rb := Ra;               (* IL: MOV Ra, Rb *)

(* Vergleichen – setzt Flags in R8671 *)
CMP(Ra, Rb);

```

#### CMP-Ergebnisflags in R8671:

| Bit | Flag | Bedeutung                 |
|-----|------|---------------------------|
| .0  | T1   | Timer1 (R8028) abgelaufen |
| .1  | T2   | Timer2 (R8029) abgelaufen |
| .2  | LT   | Pa < Pb                   |
| .3  | EQ   | Pa = Pb                   |
| .4  | GT   | Pa > Pb                   |

### 3.7 Funktionsbausteine

Funktionsbausteine werden beim Aufruf mit benannten Portzuweisungen aufgerufen. Sie werden inline zu IL-Anweisungen erweitert.

#### R\_TRIG — Flankenerkennung steigend

```

\
VAR\
edge1 : R_TRIG;\
END_VAR\
VAR_ALIAS\
DI1 := R4620.1;\
END_VAR_ALIAS

edge1(CLK := DI1);

(* edge1.Q ist für genau einen Scan bei steigender Flanke von DI1 TRUE *)\

```

```
IF edge1.Q AND NOT ACTV THEN\
SET(ACTV);\
END_IF;\
```

Erzeugtes IL (4 Anweisungen): LD / ANDn / OUT / LD / OUT

## F\_TRIG — Flankenerkennung fallend

Gleiche wie R\_TRIG, löst aber bei der **fallenden** Flanke aus. Port: CLK.

## TON — Timer Ansprechverzögerung

```
\VAR
T1 : TON;
END_VAR

T1(IN := ACTV, PT := 100);
(* PT=100 Scans × 6ms = 600ms Verzögerung *)
(* T1 abgelaufen, wenn R8671.0 = 1 *)

IF R8671.0 AND ACTV \THEN
(* Timer abgelaufen Aktion *)
R8028 := R8679; (* Timer neu laden *)
END_IF;
```

## CTU — Aufwärtszähler

```
\VAR
C1 : CTU;
END_VAR

C1(CU := pulse_bit, PV := 5);
(* C1.Q = TRUE, wenn Zähler >= 5 *)
```

# 4. IL-Optimierer

Aktivieren Sie das Kontrollkästchen **IL optimieren**, bevor Sie Konvertieren drücken, um den Optimierer zu aktivieren. Er führt bis zu 12 iterative Durchläufe aus und wendet vier Transformationen an:

### OPT-1 — Bedingungs-Hoisting (größte Einsparung)

Jede Anweisung in einem IF-Block wiederholt normalerweise die vollständige Bedingung. Der Optimierer erkennt aufeinanderfolgende wiederholte Bedingungen und gibt sie nur einmal aus, wobei

der Stack über mehrere Aktionen aktiv bleibt.

```

Vorher:  LD 8671.2  AND 8671,0  ADD 4622,8683,4622  (3+1 = 4 Anweisungen)
          LD 8671.2  AND 8671,0  MOV 8679,8028      (3+1 = 4 Anweisungen)

Nachher: LD 8671.2  AND 8671,0  ADD 4622,8683,4622  (3+1+1 = 5
Anweisungen)
          MOV 8679,\8028

```

Für ein IF mit N Anweisungen und einer Bedingungslänge C ist die Einsparung  $(N-1) \times C$  Anweisungen.

### OPT-2 — IF/ELSE SET/RES → OUT

IF A THEN SET(Q); ELSE RESET(Q); END\_IF; ist semantisch identisch mit OUT Q — das Bit erhält genau den Stackwert.

```

Vorher:  LD 4620.1  SET 8684,0      (2 Anweisungen)
          LDn 4620.1 RES 8684,0     (2 Anweisungen)

Nachher: LD 4620.1  OUT 8684,0      (2 Anweisungen insgesamt)

```

### OPT-3 — Totes MOV eliminiert

MOV Pa, Pa (Register auf sich selbst kopieren) wird stillschweigend entfernt.

### OPT-4 — Doppelte NOT-Eliminierung

LDn + OUTn → LD + OUT (doppelte Negation hebt sich auf).

**Typische Einsparungen:** 30–45 % bei Programmen mit mehreren IF-Blöcken. Der Meldungsbereich zeigt eine Aufschlüsselung: z. B. *Optimierer: 38 → 22 Schritte (-42 %)*. *OPT-1:-14, OPT-2:-2. Durchläufe: 2.*

## 5. Symboltabelle

Die Symboltabelle erscheint automatisch am unteren Rand der Seite, wenn das Programm einen VAR-Block enthält. Sie zeigt:

Das Zähler-Abzeichen (z. B. *3 Variablen*) zeigt, wie viele typisierte Variablen deklariert wurden.

Verwenden Sie die Symboltabelle, um genau zu wissen, welches Register jeder Variable entspricht, wenn Sie den Motor mit MODBUS oder NILAB Starter überwachen.

## 6. IL-Simulator

Der IL-Simulator führt das kompilierte IL-Programm zyklusweise direkt im Browser aus und ermöglicht eine Funktionsverifikation ohne Verbindung zur Hardware.

### Aktivieren des Simulators

Wenn Sie das ST-Programm ändern und erneut Konvertieren drücken, während der Simulator aktiv ist, wird das IL neu geladen und die Simulation automatisch zurückgesetzt.

### Simulator-Panels

- Eingänge R4620** — Drei Umschalt-Schaltflächen (IN0, IN1, IN2), die Bits 0-2 des digitaleingang-Registers R4620 darstellen. Klicken, um das Bit umzuschalten. Grüne Hervorhebung = Bit ist 1.
- Ausgänge R4621** — Vier LEDs, die Bits 0-2 von R4621 (digitalausgänge) und Bit 0 von R4096 (Antrieb aktivieren) zeigen. Orange = aktiv.
- R4622** — Zeigt den aktuellen Wert des Benutzerausgangsregisters R4622 (Bewegungsregler-Tabellenindex, Positionsbefehl usw.).
- Flags R8671** — Indikator-Pillen für jedes Flag-Bit. Grün = aktiv: \* T1, T2 — Timer abgelaufen \* LT, EQ, GT — Ergebnis der letzten CMP-Anweisung
- Steuerung:** \* **Zurücksetzen** — Lädt IL aus dem Ausgabepanel neu und initialisiert alle Register \* **Schritt** — Führt einen vollständigen Scan-Zyklus aus (oder **Leertaste** drücken, wenn pausiert) \* **Lauf / Pause** — Kontinuierliche Ausführung bei der ausgewählten Geschwindigkeit \* **Langsam / Mittel / Schnell / 6ms** — Scan-Intervall: 1000ms / 300ms / 80ms / 6ms (echte Hardwaregeschwindigkeit)
- Register** — Eine Liste aller im kompilierten IL referenzierten Register, angezeigt als editierbare Felder. Werte werden bei jedem Scan-Zyklus aktualisiert. Sie können auch einen neuen Wert eingeben, um ein Register zu überschreiben (nützlich, um einen Positionswert zu injizieren oder eine MODBUS-Schreiboperation zu simulieren).

### Timer-Simulation

Beide Hardware-Timer (R8028, R8029) werden zu Beginn jedes Scan-Zyklus um 1 dekrementiert. Die T1/T2-Flags in R8671 werden gesetzt, wenn der Timer null erreicht. Um einen 600-ms-Timer zu simulieren:

## 7. Speichern, Laden und Herunterladen

| Schaltfläche | Aktion                                               |
|--------------|------------------------------------------------------|
| KONVERTIEREN | Das ST-Programm kompilieren. Verknüpfung: Strg+Enter |

|                   |                                                                                             |
|-------------------|---------------------------------------------------------------------------------------------|
| DOWNLOAD .IL      | Den kompilierten IL als .il-Textdatei speichern. Blockiert, wenn Fehler vorhanden sind      |
| SPEICHERN .ST     | Den aktuellen ST-Quellcode als .st-Datei speichern                                          |
| LADEN .ST         | Eine zuvor gespeicherte .st-Datei in den Editor öffnen. Löst automatische Konvertierung aus |
| IL OPTIMIEREN     | Kontrollkästchen — vor Konvertieren aktivieren, um den 4-Pass-IL-Optimierer zu aktivieren   |
| SIMULATOR         | Das IL-Simulator-Panel umschalten                                                           |
| REFERENZ ZEIGEN   | Die Syntaxreferenz-Seitenleiste umschalten                                                  |
| PROJEKTNAMEN-FELD | Setzt den Dateinamen, der von Download/Save-Schaltflächen verwendet wird                    |

## 8. IL in den Motor laden

Nach dem Herunterladen der .il-Datei:

Die maximale Programmgröße beträgt **64 Anweisungen** (einschließlich des obligatorischen END). Der Schritt-Zähler in der Werkzeugleiste zeigt die aktuelle Nutzung und wird gelb über 48 Schritte und rot über 64.

## 9. Einschränkungen und bekannte Grenzen

| Einschränkung                   | Details                                                                 |
|---------------------------------|-------------------------------------------------------------------------|
| Maximale Anweisungen            | 64 Schritte (einschließlich END)                                        |
| Stack-Tiefe                     | 1 — keine verschachtelten AND/OR in einer IF-Bedingung                  |
| Keine FOR / WHILE-Schleifen     | Verwenden Sie stattdessen Timer-basierte Sequenzen oder entrollte Logik |
| Kein verschachteltes IF         | Verwenden Sie Flag-Bits (BOOL-Variablen) für komplexe Zustände          |
| Keine Arithmetik in Bedingungen | Mit CMP zuerst auswerten, dann R8671-Flagbits testen                    |
| Konstantenlitterale             | Nur 0, 1, 10, 100, 1000 werden Konstantenregistern zugeordnet           |
| MAX BOOL-Variablen              | 32 (Bits 0-15 von R8681 und R8682)                                      |
| MAX INT-Variablen               | 6 (R8683 bis R8688)                                                     |
| MAX Timer (TON/CTD)             | 2 (R8028 und R8029)                                                     |
| Bitindex-Bereich                | nur 0 bis 15                                                            |
| Schreibgeschützte Register      | R8675-R8680 können keine Schreibziele sein                              |

## 10. Vollständiges Beispiel — 4-Schritt-Bewegung mit 2 Digitaleingängen

Dieses Programm wählt einen von vier Bewegungstabelleneinträgen basierend auf dem Zustand von zwei digitaleingängen aus. Es wird typischerweise mit dem NILAB-Bewegungsregler verwendet, der für 5 Karten mit 2 Positionen pro Karte konfiguriert ist (Parameter 1792 = 5).

```

\
(* NILAB uPLC - 4-Schritt-Bewegungssequenz mit 2 Digitaleingängen *)\
(* *)\
(* Parameter 1792 = 5 Karten, 2 Positionen pro Karte *)\
(* Bewegungsregler: 10 Positionsaufgaben konfiguriert *)\

VAR_ALIAS\
DI0 := R4620.0; (* Digital Input 0 *)\
DI1 := R4620.1; (* Digital Input 1 *)\
END_VAR_ALIAS

(* Tabellenindex-Konstanten vorberechnen *)\
R8683 := R8676 + R8676; (* R8683 = 2 *)\
R8684 := R8683 + R8683; (* R8684 = 4 *)\
R8685 := R8684 + R8683; (* R8685 = 6 *)

(* Tabellenindex basierend auf DI0-und-DI1-Kombination auswählen *)\
IF DI0 AND DI1 THEN (* DI0=1, DI1=1 → Index 6 *)\
R4622 := R8685;\
END_IF;

IF DI0 AND NOT DI1 THEN (* DI0=1, DI1=0 → Index 4 *)\
R4622 := R8684;\
END_IF;

IF NOT DI0 AND DI1 THEN (* DI0=0, DI1=1 → Index 2 *)\
R4622 := R8683;\
END_IF;

IF NOT DI0 AND NOT DI1 THEN (* DI0=0, DI1=0 → Index 0 *)\
R4622 := R8675;\
END_IF;

(* Tabellenindex nur schreiben, wenn er sich ändert *)\
CMP(R4622, R8686);\
IF NOT R8671.3 THEN (* EQ-Flag = 0 bedeutet Wert geändert *)\
R4623 := R8676; (* Signal: neuer Tabellenindex aktiv *)\
R8686 := R4622; (* aktuellen Index als Referenz speichern *)\
END_IF;\

```

### Wahrheitstabelle:

| DI0 | DI1 | R4622 (Tabellenindex) - R4623 muss auf 1 gesetzt werden | Bewegungsposition        |
|-----|-----|---------------------------------------------------------|--------------------------|
| 0   | 0   | 0                                                       | Positionsaufgabe 0 und 1 |
| 0   | 1   | 2                                                       | Positionsaufgabe 2 und 3 |
| 1   | 0   | 4                                                       | Positionsaufgabe 4 und 5 |
| 1   | 1   | 6                                                       | Positionsaufgabe 6 und 7 |

**So testen Sie mit dem Simulator:** - Konvertieren drücken (mit Optimize IL aktiviert: 23 Anweisungen → ~16) - Simulator → Zurücksetzen drücken - IN0- und IN1-Schaltflächen umschalten —

beobachten Sie, wie R4622 sich im Registerpanel ändert - Das EQ-Flag (R8671.3) sollte 0 sein, wenn sich der Wert ändert, und 1, wenn er gleich bleibt - R4623 ändert sich nur im Scan auf 1, wenn sich der Tabellenindex ändert

## Siehe auch:

| Element                                       |
|-----------------------------------------------|
| <a href="#">uPLC Instruction-Set-Referenz</a> |
| <a href="#">uPLC-Beispielprogramme</a>        |

From:

<https://dokuwiki.nilab.at/> - **NiLAB GmbH**  
**Knowledgebase**

Permanent link:

[https://dokuwiki.nilab.at/doku.php?id=de:nilab\\_st\\_converter:start](https://dokuwiki.nilab.at/doku.php?id=de:nilab_st_converter:start)

Last update: **2026/09/11 18:07**

