Beispielsammlung (3. Beispiel)
Laborübung Sensor/Aktor-Systeme

Wolfgang Kastner, Friedrich Kastner-Masilko

SS 2007 (Vers. 1.2)

1 Allgemeines

Dieser Teil der Beispielsammlung enthält die Kollektion[1] aller Angaben zum 3. Beispiel, das Sie mit den Grundlagen der Profibus-DP-Programmierung vertraut machen soll. Jedes Beispiel besteht aus 3 Teilaufgaben, die alle Elemente der vorangegangenen Beispiele sowie Elemente der Profibus-DP-Programmierung enthalten:

a. Profibus-DP-*Programmierung mittels Polling*

- Grundlagen Profibus-DP-Interface (CP5613-Hardware und Library für VC++)
- Ansprechen von Profibus-DP-Teilnehmern (grundlegende CP5613-Operationen)
- Digital-I/O (Bit-Operationen, Sondermerker)
- Ansprechen digitaler ASi-Sensoren/Aktuatoren
- Timer-Programmierung

b. Profibus-DP-*Programmierung mittels Hardware-Events*

- Event-Handling (weiterführende CP5613-Operationen)
- Digital-I/O (Bit-Operationen, Sondermerker)
- Ansprechen digitaler ASi-Sensoren/Aktuatoren

c. Profibus-DP-*Slave-Auswertung*

- Grundlagen Encoder-Hardware
- Digital-I/O (Bit-Operationen, Sondermerker)
- Ansprechen digitaler und analoger ASi-Sensoren/Aktuatoren
- Komplexere Abläufe (Kettenprogrammierung)

Wir bitten Sie, Ihre Lösung durch ein Protokoll zu dokumentieren und während der betreuten Übungszeiten abzugeben. Im Protokoll sollte zu jeder Teilaufgabe folgendes enthalten sein:

1. Deckblatt: Das Deckblatt soll in formloser Weise folgende Informationen beinhalten: Name, Matrikelnummer und Studienkennzahl sowie Nummer und Bezeichnung des Beispiels (also z.B. Beispiel III.a.0 – Polling: Sonar auf LED).

[1] Die konkrete Beschreibung des von Ihnen zu lösenden Beispiels finden Sie unter Pkt. 2.nr mit $nr = (E + J)$ modulo 5, wobei E die **Einerstelle** Ihrer Matrikelnummer und J die **Einerstelle** des aktuellen Jahres bezeichnet.

2. Kurze Lösungsbeschreibung und Kommentare: Beschreibung eventueller Besonderheiten der Lösung und Modifikationen der Aufgabenstellung sowie Angabe von Problemen, Kritik und Anregungen[2]

   - bei der Beispiellösung
   - im Umgang mit dem Targetsystem und den Entwicklungstools
   - allgemeine Anregungen und Beschwerden bzgl. der Laborübung

3. PROFIBUS-DP-Master Programm-Sourcecode: Ausdruck des VC++-Programms mit Kommentaren. Auch hier sollte im Header der Datei zumindest Name und Matrikelnummer des Autors zu finden sein.

Wie Sie Ihre Sourcecodes und Executables benennen, ist Ihnen freigestellt. In Anbetracht der besseren Unterscheidbarkeit der Files sollte die bisherige Konvention jedoch beibehalten werden:

`Matrikelnummer_III_{a,b,c}.mwp`	...	SPS-Sourcecode
`.exe`	...	PROFIBUS-DP-Master-Executable
`.zip`	...	Archiv mit PROFIBUS-DP-Master-Sourcecodes (bitte **ohne** Debug-Daten)

Bitte vergessen Sie nicht, Ihre endgültigen Abgabedateien auch unter dem Verzeichnis `~/Bsp/` für den Praxistest abzuspeichern.

2 Beispielangaben

2.0 Positionsanzeige

(a) Implementieren Sie ein Master-Slave-System, das durch entsprechende Schalterstellungen am Targetsystem die Spannvorrichtung ein- bzw. ausschaltet. Dabei soll gelten:[3]

$ZylUp := Swt0 \oplus Swt1 \oplus Swt2 \oplus Swt3$

$ZylDo := Swt4 \oplus Swt5 \oplus Swt6 \oplus Swt7$

Diese Berechnungs- und Zuweisungsfunktion soll allerdings als Masterfunktionalität ausgeführt werden. Bei Ausfall des Masters dürfen die oben genannten Funktionen nicht mehr durchgeführt werden. Die Ausfallserkennung soll durch Zeitüberwachung eines Life-Bytes erfolgen.

Schreiben Sie dazu ein SPS-Programm, das alle relevanten Daten in den ersten PROFIBUS-DP-Block (Bank 12 am CP242-8) nach folgendem Schema schreibt bzw. von dort liest:

Bezeichnung	Bytenummer im PROFIBUS-DP-Block	Richtung
SWT-Abbild	0	Slave→Master
AS-i-Slave 7 Ausgänge	0	Master→Slave
Life-Byte	1	Master→Slave

[2] Bitte beachten Sie, daß gerade dieser Abschnitt für die Verbesserung der Laborübung sehr wichtig ist. Wir werten jeweils vor Beginn eines neuen Semesters die Protokolle der Übungsteilnehmer des vorigen Semesters aus, um z.B. jene Dinge zu identifizieren, die unnötige oder unerwartete Schwierigkeiten bereitet haben.

[3] $\oplus$ ... logische XOR-Verknüpfung

Das SPS-Programm soll auch überprüfen, ob sich der Wert des Life-Bytes innerhalb von 10s geändert hat. Bei Ablauf dieser Zeit ohne Wertänderung soll LED0 im 1-Hz-Rhythmus blinken.

Schreiben Sie weiters ein VC++-Konsolen-Programm, das die CP5613-Library zunächst nutzt, um sämtliche nötige Initialisierungsschritte (Sequenz 1 auf Seite 16) für zyklischen Polling-Betrieb durchzuführen. Anschließend soll in einer Endlosschleife zunächst auf eine Tastatureingabe gewartet werden. Entsprechend der Tastatureingabe soll dann

'q' das Master-Programm beenden, wobei auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'1' den Fortlauf für 1000 Zyklen ohne Tastatureingabe bewirken, wobei jeweils am Zyklusende eine 10ms-Pause einzuhalten ist.

'u' für genau einen Zyklus unabhängig vom Ergebnis der bool'schen Verknüpfung der Schalter das Signal ZylUp ausgeben.

'd' für genau einen Zyklus unabhängig vom Ergebnis der bool'schen Verknüpfung der Schalter das Signal ZylDo ausgeben.

In jedem Zyklus des Master-Programms muß

- das Swt-Abbild (und dabei der Status des Profibus-DP-Slaves "Targetsystem") gelesen und ausgewertet werden,
- entsprechend der Auswertung die Signale ZylUp bzw. ZylDo geschrieben werden,
- das Life-Byte um 1 erhöht und geschrieben werden,
- sämtliche Datenwerte (Swt-Abbild, Signale, Life-Byte, Slave-Status) am Bildschirm angezeigt werden (z.B. mittels `printf()`).

(b) Implementieren Sie wie unter **(a)** ein Master-Slave-System, nur sollen in diesem Falle statt zyklischem Polling Hardware-Events[4] verwendet werden. Alle Operationen sollen somit nicht innerhalb der Hauptprogrammschleife, sondern innerhalb eines eigenen, synchronisierten Threads durchgeführt werden.

Im Initialisierungsschritt muß in diesem Falle noch zusätzlich ein Semaphor sowie ein neuer Thread erzeugt werden (Sequenz 3 auf Seite 16). Im erzeugten Thread selbst soll auf den Semaphor gewartet (Befehl 4) und bei Zustandsänderungen die Schalterstellung ausgewertet werden.

In der Hauptprogrammschleife soll nur mehr entsprechend der Tastatureingabe

'q' das Master-Programm beenden, wobei auf die Entfernung des Semaphors und des Threads (Sequenz 5) sowie auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'u' den Zustand des Signals ZylUp zwischen *logisch 1* und *Verknüpfungsergebnis* "toggeln".

'd' den Zustand des Signals ZylDo zwischen *logisch 1* und *Verknüpfungsergebnis* "toggeln".

[4] nicht zu verwechseln mit Hardware-Interrupts

Der Mechanismus aus **(a)** zur Detektion eines Master-Ausfalls kann hier natürlich nicht mehr benutzt werden, da die Bearbeitung des Threads nur mehr bei Änderung von Slave-Daten stattfindet. Benutzen Sie stattdessen das Fehlerregister der S7-214 SPS (Busfehler-Bit: SM9.1) für die Ausfallserkennung.

(c) Realisieren Sie ein Master-Slave-System, das die Position der Schlitteneinheit des Targetsystems kontinuierlich und skaliert auf DVM ausgibt. Zusätzlich dazu soll die Taste TAST_0 eine Schlittenbewegung vom ENCODER weg und die Taste TAST_2 eine Schlittenbewegung zum ENCODER hin starten. Die Taste TAST_1 soll jede Schlittenbewegung stoppen. Die Skalierung soll durch zwei Punkte bestimmt werden, die durch Anfahren und "Teachen" – initiiert durch Tastaturkommandos – definiert werden. Die oben genannten Funktionen sollen größtenteils vom Master durchgeführt werden. Sollte dieser ausfallen, so ist am Slave die gleiche Ausfallsanzeige wie unter **(a)** durchzuführen.

Schreiben Sie ein SPS-Programm, das alle relevanten Daten in den ersten PROFIBUS-DP-Block (Bank 12 am CP242-8) nach folgendem Schema schreibt bzw. von dort liest:

Bezeichnung	Bytenummer im PROFIBUS-DP-Block	Richtung
AS-i-Slave 1 Eingänge	0	Slave→Master
AS-i-Slave 4 Ausgänge	0	Master→Slave
Analogwert für DVM	1-2	Master→Slave

Der Analogwert für DVM muß vom SPS-Programm an den AS-i-Slave 2 auf Kanal 1 weitergegeben werden. Das Signal MCGO in den Daten für den AS-i-Slave 4 – welche vom Master gesendet werden – darf erst dann an den entsprechenden AS-i-Slave weitergegeben werden, wenn sichergestellt ist, daß die Endpositionen MAGE_D bzw. MAGE_U nicht überschritten werden.

Schreiben Sie weiters ein VC++-Konsolen-Programm, das die oben genannten Funktionen auf Masterseite bereitstellt. Mittels Multithreading soll ein eigener Thread die Masterzyklen durchführen, während die Hauptprogrammschleife die Benutzerinteraktion abwickelt. Entsprechend der Tastatureingabe soll dann

'q' das Master-Programm beenden, wobei auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'a' die momentane Schlittenposition als **obersten** Punkt (ergibt 10.00V) der Skalierung "teachen". Als Defaultwert soll hier der maximal erreichbare Punkt (MAGE_U) einprogrammiert werden.

'y' die momentane Schlittenposition als **untersten** Punkt (ergibt 0.00V) der Skalierung „teachen". Als Defaultwert soll hier der minimal erreichbare Punkt (MAGE_D) einprogrammiert werden.

In jedem Zyklus des Bearbeitungsthreads muß

- das Prozeßabbild der beteiligten Slaves (DP-Slaves „Targetsystem" und ENCODER) gelesen und ausgewertet werden,
- der Analogwert für die DVM-Ausgabe anhand des Positionswertes des ENCODER und der internen Skalierungspunkte berechnet und geschrieben werden,

- entsprechend der Auswertung der Tasten (AS-i-Slave 1) die Signale MCDir bzw. MCGo (AS-i-Slave 4) geschrieben werden,
- sämtliche Datenwerte (Prozeßabbild, Signale, Slave-Status) am Bildschirm angezeigt werden (z.B. mittels `printf()`).

Überlegen Sie sich, ob im vorliegenden Fall ein Einsatz von Hardware-Events zielführend ist.

2.1 Geschwindigkeitsanzeige

(a) Implementieren Sie ein Master-Slave-System, das den Zustand der von Tast_0 – Tast_3 des Targetsystems abwechselnd auf LED0 – LED7 abbildet. Dabei soll gelten:

Zyklus n:	*Zyklus* n+1:
$LED0 := Tast\_0$	$LED0 := abgeschaltet$
$LED1 := Tast\_1$	$LED1 := abgeschaltet$
$LED2 := Tast\_2$	$LED2 := abgeschaltet$
$LED3 := Tast\_3$	$LED3 := abgeschaltet$
$LED4 := abgeschaltet$	$LED4 := Tast\_0$
$LED5 := abgeschaltet$	$LED5 := Tast\_1$
$LED6 := abgeschaltet$	$LED6 := Tast\_2$
$LED7 := abgeschaltet$	$LED7 := Tast\_3$

Diese Berechnungs- und Zuweisungsfunktion soll allerdings als Masterfunktionalität ausgeführt werden. Bei Ausfall des Masters dürfen die oben genannten Funktionen nicht mehr durchgeführt werden. Die Ausfallserkennung soll durch Zeitüberwachung eines Life-Bytes erfolgen.

Schreiben Sie dazu ein SPS-Programm, das alle relevanten Daten in den ersten Profibus-DP-Block (Bank 12 am CP242-8) nach folgendem Schema schreibt bzw. von dort liest:

Bezeichnung	Bytenummer im Profibus-DP-Block	Richtung
AS-i-Slave 1 Eingänge	0	Slave→Master
LED-Abbild	0	Master→Slave
Life-Byte	1	Master→Slave

Das SPS-Programm soll auch überprüfen, ob sich der Wert des Life-Bytes innerhalb von 10s geändert hat. Bei Ablauf dieser Zeit ohne Wertänderung soll Lampe_0 im 1-Hz-Rhythmus blinken.

Schreiben Sie weiters ein VC++-Konsolen-Programm, das die CP5613-Library zunächst nutzt, um sämtliche nötige Initialisierungsschritte (Sequenz 1 auf Seite 16) für zyklischen Polling-Betrieb durchzuführen. Anschließend soll in einer Endlosschleife zunächst auf eine Tastatureingabe gewartet werden. Entsprechend der Tastatureingabe soll dann

'q' das Master-Programm beenden, wobei auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'5' den Fortlauf für 1000 Zyklen ohne Tastatureingabe bewirken, wobei jeweils am Zyklusende eine 50ms-Pause einzuhalten ist.

'a' für genau einen Zyklus unabhängig vom tatsächlichen Zustand der Taste TAST_3 einen Druck auf selbige simulieren.

's' für genau einen Zyklus unabhängig vom tatsächlichen Zustand der Taste TAST_2 einen Druck auf selbige simulieren.

'd' für genau einen Zyklus unabhängig vom tatsächlichen Zustand der Taste TAST_1 einen Druck auf selbige simulieren.

'f' für genau einen Zyklus unabhängig vom tatsächlichen Zustand der Taste TAST_0 einen Druck auf selbige simulieren.

In jedem Zyklus des Master-Programms muß

- der Zustand der Tasten TAST_0 bis TAST_3 (und dabei der Status des DP-Slaves "Targetsystem") gelesen und ausgewertet werden,
- entsprechend der Auswertung das LED-Abbild geschrieben werden,
- das Life-Byte um 1 erhöht und geschrieben werden,
- sämtliche Datenwerte (LED-Abbild, Tasten, Life-Byte, Slave-Status) am Bildschirm angezeigt werden (z.B. mittels `printf()`).

(b) Implementieren Sie wie unter **(a)** ein Master-Slave-System, nur sollen in diesem Falle statt zyklischem Polling Hardware-Events[5] verwendet werden. Alle Operationen sollen somit nicht innerhalb der Hauptprogrammschleife, sondern innerhalb eines eigenen, synchronisierten Threads durchgeführt werden.

Im Initialisierungsschritt muß in diesem Falle noch zusätzlich ein Semaphor sowie ein neuer Thread erzeugt werden (Sequenz 3 auf Seite 16). Im erzeugten Thread selbst soll auf den Semaphor gewartet (Befehl 4) und bei Zustandsänderungen die Tasterstellung ausgewertet werden.

In der Hauptprogrammschleife soll nur mehr entsprechend der Tastatureingabe

'q' das Master-Programm beenden, wobei auf die Entfernung des Semaphors und des Threads (Sequenz 5) sowie auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'a' die Tastenauswertung für TAST_3 zwischen *simuliert logisch 1* und *tatsächlicher Wert* "toggeln".

's' die Tastenauswertung für TAST_2 zwischen *simuliert logisch 1* und *tatsächlicher Wert* "toggeln".

'd' die Tastenauswertung für TAST_1 zwischen *simuliert logisch 1* und *tatsächlicher Wert* "toggeln".

'f' die Tastenauswertung für TAST_0 zwischen *simuliert logisch 1* und *tatsächlicher Wert* "toggeln".

[5] nicht zu verwechseln mit Hardware-Interrupts

Der Mechanismus aus **(a)** zur Detektion eines Master-Ausfalls kann hier natürlich nicht mehr benutzt werden, da die Bearbeitung des Threads nur mehr bei Änderung von Slave-Daten stattfindet. Benutzen Sie stattdessen das Fehlerregister des S7-214 SPS (Busfehler-Bit: SM9.1) für die Ausfallserkennung.

(c) Realisieren Sie ein Master-Slave-System, das die Geschwindigkeit der Schlitteneinheit des Targetsystems errechnet und kontinuierlich skaliert auf LED0 – LED7 ausgibt. Zusätzlich dazu soll die Ansteuerung der *Steuerspannung* des MOTOR über Tastatureingabe in der Einheit Volt möglich sein. Die Skalierung auf LED0 – LED7 soll folgendermaßen erfolgen:

$LED\ i := (Geschwindigkeit[mm/s] > i[mm/s])$

Die oben genannten Funktionen sollen größtenteils vom Master durchgeführt werden. Sollte dieser ausfallen, so ist am Slave die gleiche Ausfallsanzeige wie unter **(a)** durchzuführen. Der Slave soll weiters unabhängig vom Master-Zustand die Schlitteneinheit des Targetsystems dauerhaft zwischen den Positionen MAGE_D und MAGE_U pendeln lassen.

Schreiben Sie ein SPS-Programm, das alle relevanten Daten in den ersten PROFIBUS-DP-Block (Bank 12 am CP242-8) nach folgendem Schema schreibt bzw. von dort liest:

Bezeichnung	Bytenummer im PROFIBUS-DP-Block	Richtung
LED-Abbild	0	Master→Slave
Analogwert für MCSPEED	1-2	Master→Slave

Der Analogwert für MCSPEED muß vom SPS-Programm an den AS-i-Slave 2 auf Kanal 2 weitergegeben werden.

Schreiben Sie weiters ein VC++-Konsolen-Programm, das die oben genannten Funktionen auf Masterseite bereitstellt. Mittels Multithreading soll ein eigener Thread die Masterzyklen durchführen, während die Hauptprogrammschleife die Benutzerinteraktion abwickelt. Entsprechend der Tastatureingabe soll dann

'q' das Master-Programm beenden, wobei auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

's' einen Prompt zur Eingabe der *Steuerspannung* in Volt (Default 5V) bereitstellen.

Der Bearbeitungsthread soll zur einfachen Berechnung der Geschwindigkeit zyklisches Polling mit 100ms Wartezeit betreiben. In jedem Zyklus des Bearbeitungsthreads muß

- der Positionswert des ENCODER gelesen und – durch Vergleich mit dem vorhergehenden Positionswert – die Geschwindigkeit ausgewertet werden,
- der Analogwert anhand der Vorgabewerte geschrieben werden,
- entsprechend der Skalierung das LED-Abbild geschrieben werden,
- sämtliche Datenwerte (Positions- und Geschwindigkeitswert, LED-Abbild, Slave-Status) am Bildschirm angezeigt werden (z.B. mittels `printf()`).

2.2 Linearitätsmessung

(a) Implementieren Sie ein Master-Slave-System, das durch entsprechende Schalterstellungen am Targetsystem die Lampen aufleuchten lässt. Dabei soll gelten:[6]

$Lampe\_0 := Swt0 \oplus Swt1$

$Lampe\_1 := Swt2 \oplus Swt3$

$Lampe\_2 := Swt4 \oplus Swt5$

$Lampe\_3 := Swt6 \oplus Swt7$

Diese Berechnungs- und Zuweisungsfunktion soll allerdings als Masterfunktionalität ausgeführt werden. Bei Ausfall des Masters dürfen die oben genannten Funktionen nicht mehr durchgeführt werden. Die Ausfallserkennung soll durch Zeitüberwachung eines Life-Bytes erfolgen.

Schreiben Sie dazu ein SPS-Programm, das alle relevanten Daten in den ersten PROFIBUS-DP-Block (Bank 12 am CP242-8) nach folgendem Schema schreibt bzw. von dort liest:

Bezeichnung	Bytenummer im PROFIBUS-DP-Block	Richtung
SWT-Abbild	0	Slave→Master
AS-i-Slave 1 Ausgänge	0	Master→Slave
Life-Byte	1	Master→Slave

Das SPS-Programm soll auch überprüfen, ob sich der Wert des Life-Bytes innerhalb von 10s geändert hat. Bei Ablauf dieser Zeit ohne Wertänderung soll LED0 im 1-Hz-Rhythmus blinken.

Schreiben Sie weiters ein VC++-Konsolen-Programm, das die CP5613-Library zunächst nutzt, um sämtliche nötige Initialisierungsschritte (Sequenz 1 auf Seite 16) für zyklischen Polling-Betrieb durchzuführen. Anschließend soll in einer Endlosschleife zunächst auf eine Tastatureingabe gewartet werden. Entsprechend der Tastatureingabe soll dann

'q' das Master-Programm beenden, wobei auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'0' den Fortlauf für 1000 Zyklen ohne Tastatureingabe bewirken, wobei jeweils am Zyklusende eine 100ms-Pause einzuhalten ist.

'a' für genau einen Zyklus unabhängig vom Ergebnis der bool'schen Verknüpfung der Schalter die Lampe LAMPE_3 aufleuchten lassen.

's' für genau einen Zyklus unabhängig vom Ergebnis der bool'schen Verknüpfung der Schalter die Lampe LAMPE_2 aufleuchten lassen.

'd' für genau einen Zyklus unabhängig vom Ergebnis der bool'schen Verknüpfung der Schalter die Lampe LAMPE_1 aufleuchten lassen.

'f' für genau einen Zyklus unabhängig vom Ergebnis der bool'schen Verknüpfung der Schalter die Lampe LAMPE_0 aufleuchten lassen.

In jedem Zyklus des Master-Programms muß

[6] $\oplus$ ... logische XOR-Verknüpfung

- das SWT-Abbild (und dabei der Status des DP-Slaves "Targetsystem") gelesen und ausgewertet werden,
- entsprechend der Auswertung die Lampen LAMPE_0 bis LAMPE_3 gesetzt werden,
- das Life-Byte um 1 erhöht und geschrieben werden,
- sämtliche Datenwerte (SWT-Abbild, Lampen, Life-Byte, Slave-Status) am Bildschirm angezeigt werden (z.B. mittels `printf()`).

(b) Implementieren Sie wie unter **(a)** ein Master-Slave-System, nur sollen in diesem Falle statt zyklischem Polling Hardware-Events[7] verwendet werden. Somit sollen alle Operationen nicht innerhalb der Hauptprogrammschleife, sondern innerhalb eines eigenen, synchronisierten Threads durchgeführt werden.

Im Initialisierungsschritt muß in diesem Falle noch zusätzlich ein Semaphor sowie ein neuer Thread erzeugt werden (Sequenz 3 auf Seite 16). Im erzeugten Thread selbst soll auf den Semaphor gewartet (Befehl 4) und bei Zustandsänderunge Schalterstellung ausgewertet werden.

In der Hauptprogrammschleife soll nur mehr entsprechend der Tastatureingabe

'q' das Master-Programm beenden, wobei auf die Entfernung des Semaphors und des Threads (Sequenz 5) sowie auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'a' den Zustand der Lampe LAMPE_3 zwischen *leuchten* und *Verknüpfungsergebnis* "toggeln".

's' den Zustand der Lampe LAMPE_2 zwischen *leuchten* und *Verknüpfungsergebnis* "toggeln".

'd' den Zustand der Lampe LAMPE_1 zwischen *leuchten* und *Verknüpfungsergebnis* "toggeln".

'f' den Zustand der Lampe LAMPE_0 zwischen *leuchten* und *Verknüpfungsergebnis* "toggeln".

Der Mechanismus aus **(a)** zur Detektion eines Master-Ausfalls kann hier natürlich nicht mehr benutzt werden, da die Bearbeitung des Threads nur mehr bei Änderung von Slave-Daten stattfindet. Benutzen Sie stattdessen das Fehlerregister des S7-214 SPS (Busfehler-Bit: SM9.1) für die Ausfallserkennung.

(c) Realisieren Sie ein Master-Slave-System, das die Position der Schlitteneinheit des Targetsystems kontinuierlich mit dem DIST-Wert vergleicht und das Vorzeichen sowie das Ausmaß des *Differenzwertes* auf den Lampen LAMPE_0 bis LAMPE_3 signalisiert. Dabei soll gelten:

$Differenzwert := Encoderwert[mm] - Distwert[mm]$

- LAMPE_3 leuchtet **gelb**, wenn der *Differenzwert* negatives Vorzeichen hat,
- LAMPE_2 leuchtet **grün**, wenn der *Differenzwert* $\frac{1}{3}$ des *Schwellwertes* übersteigt,
- LAMPE_1 leuchtet **weiß**, wenn der *Differenzwert* $\frac{2}{3}$ des *Schwellwertes* übersteigt,

[7] nicht zu verwechseln mit Hardware-Interrupts

- LAMPE_0 leuchtet **rot**, wenn der *Differenzwert* $\frac{3}{3}$ des *Schwellwertes* übersteigt.

Der *Schwellwert* soll mittels Tastatureingabe in Millimeter angegeben.

Die oben genannten Funktionen sollen größtenteils vom Master durchgeführt werden. Sollte dieser ausfallen, so ist am Slave die gleiche Ausfallsanzeige wie unter **(a)** durchzuführen. Der Slave soll weiters unabhängig vom Master-Zustand die Schlitteneinheit des Targetsystems dauerhaft zwischen den Positionen MAGE_D und MAGE_U pendeln lassen.

Schreiben Sie ein SPS-Programm, das alle relevanten Daten in den ersten PROFIBUS-DP-Block (Bank 12 am CP242-8) nach folgendem Schema schreibt bzw. von dort liest:

Bezeichnung	Bytenummer im PROFIBUS-DP-Block	Richtung
Analogwert von DIST	0-1	Slave→Master
AS-i-Slave 1 Ausgänge	0	Master→Slave

Der Analogwert für DIST muß vom SPS-Programm aus AS-i-Slave 5 auf Kanal 2 gelesen und weitergegeben werden.

Schreiben Sie weiters ein VC++-Konsolen-Programm, das die oben genannten Funktionen auf Masterseite bereitstellt. Mittels Multithreading soll ein eigener Thread die Masterzyklen durchführen, während die Hauptprogrammschleife die Benutzerinteraktion abwickelt. Entsprechend der Tastatureingabe soll dann

'q' das Master-Programm beenden, wobei auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

's' einen Prompt zur Eingabe des *Schwellwertes* in Millimeter (Default 10mm) bereitstellen.

In jedem Zyklus des Bearbeitungsthreads muß

- das Prozeßabbild der beteiligten Slaves (PROFIBUS-DP-Slaves "Targetsystem" und ENCODER) gelesen und ausgewertet werden,
- entsprechend der Auswertung LAMPE_0 – LAMPE_3 (AS-i-Slave 1) gesetzt werden,
- sämtliche Datenwerte (Prozeßabbild, Lampen, Slave-Status) am Bildschirm angezeigt werden (z.B. mittels `printf()`).

Überlegen Sie sich, ob im vorliegenden Fall ein Einsatz von Hardware-Events zielführend ist.

2.3 Entfernungsabhängige Geschwindigkeitsanpassung

(a) Implementieren Sie ein Master-Slave-System, das den SONAR-Zustand des Targetsystems auf LED0 – LED7 abbildet. Dabei soll gelten:

$$
\begin{aligned}
LED0 &:= D0 \vee D1 \vee D2 \\
LED1 &:= D1 \vee D2 \\
LED2 &:= D1 \vee D2 \\
LED3 &:= D2 \\
LED4 &:= D2 \\
LED5 &:= D1 \vee D2 \\
LED6 &:= D1 \vee D2 \\
LED7 &:= D0 \vee D1 \vee D2
\end{aligned}
$$

$D0$, $D1$ und $D2$ sind dabei die SONAR-Bits. Ist SONAR-Bit D3 aktiv, so sollen die entsprechend aktivierten LEDs im 1-Hz-Rhythmus blinken, um die Unzuverlässigkeit der angezeigten Daten erkennen zu können.

Diese Berechnungs- und Zuweisungsfunktion soll allerdings als Masterfunktionalität ausgeführt werden. Bei Ausfall des Masters dürfen die oben genannten Funktionen nicht mehr durchgeführt werden. Die Ausfallserkennung soll durch Zeitüberwachung eines Life-Bytes erfolgen.

Schreiben Sie dazu ein SPS-Programm, das alle relevanten Daten in den ersten PROFIBUS-DP-Block (Bank 12 am CP242-8) nach folgendem Schema schreibt bzw. von dort liest:

Bezeichnung	Bytenummer im PROFIBUS-DP-Block	Richtung
AS-i-Slave 9 Eingänge	0	Slave->Master
LED-Abbild	0	Master→Slave
Life-Byte	1	Master→>Slave

Das SPS-Programm soll auch überprüfen, ob sich der Wert des Life-Bytes innerhalb von 10s geändert hat. Bei Ablauf dieser Zeit ohne Wertänderung soll LAMPE_0 im 1-Hz-Rhythmus blinken.

Schreiben Sie weiters ein VC++-Konsolen-Programm, das die CP5613-Library zunächst nutzt, um sämtliche nötige Initialisierungsschritte (Sequenz 1 auf Seite 16) für zyklischen Polling-Betrieb durchzuführen. Anschließend soll in einer Endlosschleife zunächst auf eine Tastatureingabe gewartet werden. Entsprechend der Tastatureingabe soll dann

'q' das Master-Programm beenden, wobei auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'1' den Fortlauf für 1000 Zyklen ohne Tastatureingabe bewirken, wobei jeweils am Zyklusende eine 10ms-Pause einzuhalten ist.

'a' für genau einen Zyklus unabhängig vom tatsächlichen Zustand des SONAR-Bits D0 dieses simulieren.

's' für genau einen Zyklus unabhängig vom tatsächlichen Zustand des SONAR-Bits D1 dieses simulieren.

'd' für genau einen Zyklus unabhängig vom tatsächlichen Zustand des SONAR-Bits D2 dieses simulieren.

'e' für genau einen Zyklus unabhängig vom tatsächlichen Zustand des SONAR-Bits D3 dieses simulieren.

In jedem Zyklus des Master-Programms muß

- der Zustand der SONAR-Bits D0 bis D3 (und dabei der Status des DP-Slaves "Targetsystem") gelesen und ausgewertet werden,
- entsprechend der Auswertung das LED-Abbild geschrieben werden,
- das Life-Byte um 1 erhöht und geschrieben werden,
- sämtliche Datenwerte (LED-Abbild, SONAR-Bits, Life-Byte, Slave-Status) am Bildschirm angezeigt werden (z.B. mittels `printf()`).

(b) Implementieren Sie wie unter **(a)** ein Master-Slave-System, nur sollen in diesem Falle statt zyklischem Polling Hardware-Events[8] verwendet werden. Somit sollen alle Operationen nicht innerhalb der Hauptprogrammschleife, sondern innerhalb eines eigenen, synchronisierten Threads durchgeführt werden.

Im Initialisierungsschritt muß in diesem Falle noch zusätzlich ein Semaphor sowie ein neuer Thread erzeugt werden (Sequenz 3 auf Seite 16). Im erzeugten Thread selbst soll auf den Semaphor gewartet (Befehl 4) und SONAR-Zustandsänderungen ausgewertet werden.

In der Hauptprogrammschleife soll nur mehr entsprechend der Tastatureingabe

'q' das Master-Programm beenden, wobei auf die Entfernung des Semaphors und des Threads (Sequenz 5) sowie auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'a' die Auswertung für SONAR-Bit D0 zwischen *simuliert logisch 1* und *tatsächlicher Wert* "toggeln".

's' die Auswertung für SONAR-Bit D1 zwischen *simuliert logisch 1* und *tatsächlicher Wert* "toggeln".

'd' die Auswertung für SONAR-Bit D2 zwischen *simuliert logisch 1* und *tatsächlicher Wert* "toggeln".

'e' die Auswertung für SONAR-Bit D3 zwischen *simuliert logisch 1* und *tatsächlicher Wert* "toggeln".

Der Mechanismus aus **(a)** zur Detektion eines Master-Ausfalls kann hier natürlich nicht mehr benutzt werden, da die Bearbeitung des Threads nur mehr bei Änderung von Slave-Daten stattfindet. Benutzen Sie stattdessen das Fehlerregister des S7-214 SPS (Busfehler-Bit: SM9.1) für die Ausfallserkennung.

(c) Realisieren Sie ein Master-Slave-System, das die Geschwindigkeit der Schlitteneinheit des Targetsystems errechnet und kontinuierlich auf dem Bildschirm ausgibt. Zusätzlich dazu soll die Ansteuerung der *Steuerspannung* des MOTOR gestuft über den SONAR-Wert erfolgen:

$Aktives\ Sonar - Bit$	$Motor - Steuerspannung$
$keines$	$0.0V$
$D0$	$3.3V$
$D1$	$6.6V$
$D2$	$9.9V$
$D3$	$0.0V$

[8] nicht zu verwechseln mit Hardware-Interrupts

Die oben genannten Funktionen sollen größtenteils vom Master durchgeführt werden. Sollte dieser ausfallen, so ist am Slave die gleiche Ausfallsanzeige wie unter **(a)** durchzuführen. Der Slave soll weiters unabhängig vom Master-Zustand die Schlitteneinheit des Targetsystems dauerhaft zwischen den Positionen MAGE_D und MAGE_U pendeln lassen.

Schreiben Sie ein SPS-Programm, das alle relevanten Daten in den ersten PROFIBUS-DP-Block (Bank 12 am CP242-8) nach folgendem Schema schreibt bzw. von dort liest:

Bezeichnung	Bytenummer im PROFIBUS-DP-Block	Richtung
AS-i-Slave 9 Eingänge	0	Slave→Master
Analogwert für MCSPEED	0-1	Master→Slave

Der Analogwert für MCSPEED muß vom SPS-Programm an den AS-i-Slave 2 auf Kanal 2 weitergegeben werden.

Schreiben Sie weiters ein VC++-Konsolen-Programm, das die oben genannten Funktionen auf Masterseite bereitstellt. Mittels Multithreading soll ein eigener Thread die Masterzyklen durchführen, während die Hauptprogrammschleife die Benutzerinteraktion abwickelt. Entsprechend der Tastatureingabe soll dann

'q' das Master-Programm beenden, wobei auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'e' den Zustand des SONAR zwischen *simuliert ausgefallen* und *tatsächlicher Zustand* "toggeln".

Der Bearbeitungsthread soll zur einfachen Berechnung der Geschwindigkeit zyklisches Polling mit 100ms Wartezeit betreiben. In jedem Zyklus des Bearbeitungsthreads muß

- der Positionswert des ENCODER gelesen und – durch Vergleich mit dem vorhergehenden Positionswert – die Geschwindigkeit ausgewertet werden,
- der Analogwert anhand der SONAR-Werte geschrieben werden,
- sämtliche Datenwerte (Positions- und Geschwindigkeitswert, SONAR-Werte, Slave-Status) am Bildschirm angezeigt werden (z.B. mittels `printf()`).

2.4 Positionsfahrt

(a) Implementieren Sie ein Master-Slave-System, das durch entsprechende Schalterstellungen am Targetsystem die Schlitteneinheit bewegt. Dabei soll gelten:[9]

$$MCGo := (Swt0 \oplus Swt1 \oplus Swt2 \oplus Swt3) \vee (Swt4 \oplus Swt5 \oplus Swt6 \oplus Swt7)$$

$$MCDir := (Swt0 \oplus Swt1 \oplus Swt2 \oplus Swt3) \wedge \neg(Swt4 \oplus Swt5 \oplus Swt6 \oplus Swt7)$$

Diese Berechnungs- und Zuweisungsfunktion soll allerdings als Masterfunktionalität ausgeführt werden. Bei Ausfall des Masters dürfen die oben genannten Funktionen nicht mehr durchgeführt werden. Die Ausfallserkennung soll durch Zeitüberwachung eines Life-Bytes erfolgen.

[9] $\oplus$ ... logische XOR-Verknüpfung

Schreiben Sie dazu ein SPS-Programm, das alle relevanten Daten in den ersten PROFIBUS-DP-Block (Bank 12 am CP242-8) nach folgendem Schema schreibt bzw. von dort liest:

Bezeichnung	Bytenummer im PROFIBUS-DP-Block	Richtung
SWT-Abbild	0	Slave→Master
AS-i-Slave 4 Ausgänge	0	Master→Slave
Life-Byte	1	Master→Slave

Das SPS-Programm soll auch überprüfen, ob sich der Wert des Life-Bytes innerhalb von 10s geändert hat. Bei Ablauf dieser Zeit ohne Wertänderung soll LED0 im 1-Hz-Rhythmus blinken.

Schreiben Sie weiters ein VC++-Konsolen-Programm, das die CP5613-Library zunächst nutzt, um sämtliche nötige Initialisierungsschritte (Sequenz 1 auf Seite 16) für zyklischen Polling-Betrieb durchzuführen. Anschließend soll in einer Endlosschleife zunächst auf eine Tastatureingabe gewartet werden. Entsprechend der Tastatureingabe soll dann

'q' das Master-Programm beenden, wobei auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'0' den Fortlauf für 1000 Zyklen ohne Tastatureingabe bewirken, wobei am Ende jedes Zykluses eine 100ms-Pause einzuhalten ist.

'a' für genau einen Zyklus unabhängig vom Ergebnis der bool'schen Verknüpfungen der Schalter die Signale MCGO und MCDIR so setzen, daß die Schlitteneinheit **vom** ENCODER **weg** bewegt wird.

'y' für genau einen Zyklus unabhängig vom Ergebnis der bool'schen Verknüpfungen der Schalter die Signale MCGO und MCDIR so setzen, daß die Schlitteneinheit **zum** ENCODER **hin** bewegt wird.

In jedem Zyklus des Master-Programms muß

- das SWT-Abbild (und dabei der Status des DP-Slaves „Targetsystem“) gelesen und ausgewertet werden,
- entsprechend der Auswertung die Signale MCGO und MCDIR gesetzt werden,
- das Life-Byte um 1 erhöht und geschrieben werden,
- sämtliche Datenwerte (SWT-Abbild, Signale, Life-Byte, Slave-Status) am Bildschirm angezeigt werden (z.B. mittels `printf()`).

(b) Implementieren Sie wie unter **(a)** ein Master-Slave-System, nur sollen in diesem Falle statt zyklischem Polling Hardware-Events[10] verwendet werden. Somit sollen alle Operationen nicht innerhalb der Hauptprogrammschleife, sondern innerhalb eines eigenen, synchronisierten Threads durchgeführt werden.

Im Initialisierungsschritt muß in diesem Falle noch zusätzlich ein Semaphor sowie ein neuer Thread erzeugt werden (Sequenz 3 auf Seite 16). Im erzeugten Thread selbst soll auf den Semaphor gewartet (Befehl 4) und SWT-Zustandsänderungen ausgewertet werden.

In der Hauptprogrammschleife soll nur mehr entsprechend der Tastatureingabe

[10] nicht zu verwechseln mit Hardware-Interrupts

'q' das Master-Programm beenden, wobei auf die Entfernung des Semaphors und des Threads (Sequenz 5) sowie auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

'a' den Zustand der Signale MCGo und MCDir so "toggeln", daß einmal die Schlitteneinheit **vom** Encoder **weg** bewegt wird und einmal das Verknüpfungsergebnis aktiv wird. Eine eventuelle Bewegung, die durch **'y'** initiiert wurde, soll übergangen werden.

'y' den Zustand der Signale MCGo und MCDir so "toggeln", daß einmal die Schlitteneinheit **zum** Encoder **hin** bewegt wird und einmal das Verknüpfungsergebnis aktiv wird. Eine eventuelle Bewegung, die durch **'a'** initiiert wurde, soll übergangen werden.

Der Mechanismus aus **(a)** zur Detektion eines Master-Ausfalls kann hier natürlich nicht mehr benutzt werden, da die Bearbeitung des Threads nur mehr bei Änderung von Slave-Daten stattfindet. Benutzen Sie stattdessen das Fehlerregister des S7-214 SPS (Busfehler-Bit: SM9.1) für die Ausfallserkennung.

(c) Realisieren Sie ein Master-Slave-System, bei dem durch Vorgabe einer Joystickstellung (Achse Joy_U/D) und anschließendem Tastendruck auf Tast_0 eine beliebige Position mit der Schlitteneinheit angefahren werden kann. Die Joystickstellung soll dazu kontinuierlich am Bildschirm in Millimeter angezeigt werden, wobei die Ruhestellung des Joysticks der Mittelposition zwischen den Sensoren MagE_U und MagE_D entspricht und die maximalen Ausschläge des Joysticks der jeweiligen Endposition. Wird die Taste Tast_0 betätigt, so soll die momentan angezeigte Position gespeichert und mittels der Signale MCGo und MCDir angefahren werden, bis die Istposition innerhalb des Positionsfensters (Sollposition$\pm$*Toleranzwert*) liegt. Bei jedem Tastendruck soll die gespeicherte Position überschrieben werden. Der Schlitten muß somit nicht unbedingt im Stillstand sein, um eine neue Positionswahl treffen zu können.

Die oben genannten Funktionen sollen größtenteils vom Master durchgeführt werden, sollte dieser ausfallen, so ist am Slave die gleiche Ausfallsanzeige wie unter **(a)** durchzuführen.

Schreiben Sie ein SPS-Programm, das alle relevanten Daten in den ersten DP-Block (Bank 12 am CP242-8) nach folgendem Schema schreibt bzw. von dort liest:

Bezeichnung	Bytenummer im Profibus-DP-Block	Richtung
Analogwert von Joy_U/D	0-1	Slave→Master
AS-i-Slave 1 Eingänge	2	Slave→Master
AS-i-Slave 4 Ausgänge	0	Master→Slave

Der Analogwert für Joy_U/D muß vom SPS-Programm aus AS-i-Slave 3 auf Kanal 1 gelesen und weitergegeben werden. Das Signal MCGo in den Daten für den AS-i-Slave 4 – welche vom Master gesendet werden – darf erst dann an den entsprechenden AS-i-Slave weitergegeben werden, wenn sichergestellt ist, daß die Endpositionen MagE_D bzw. MagE_U nicht überschritten werden.

Schreiben Sie weiters ein VC++-Konsolen-Programm, das die oben genannten Funktionen auf Masterseite bereitstellt. Mittels Multithreading soll ein eigener Thread die Masterzyklen durchführen, während die Hauptprogrammschleife die Benutzerinteraktion abwickeln soll. Entsprechend der Tastatureingabe soll dann

'q' das Master-Programm beenden, wobei auf die richtige Abschlußsequenz (Sequenz 2) geachtet werden muß.

't' einen Prompt zur Eingabe des *Toleranzwertes* in Millimeter (Defaultwert 10mm) bereitstellen.

In jedem Zyklus des Bearbeitungsthreads muß

- das Prozeßabbild der beteiligten Slaves (DP-Slaves "Targetsystem" und ENCODER) gelesen und ausgewertet werden,
- entsprechend der Auswertung der Soll- und Istposition die Signale MCDIR bzw. MCGO (AS-i-Slave 4) geschrieben werden,
- sämtliche Datenwerte (Prozeßabbild, Positionen, Signale, Slave-Status) am Bildschirm angezeigt werden (z.B. mittels `printf()`).

Überlegen Sie sich, ob im vorliegenden Fall ein Einsatz von Hardware-Events zielführend ist.

3 Verwendete Befehle

Die folgenden Befehle bzw. Befehlssequenzen werden zur Bearbeitung der Beispiele unter Pkt. 2 benötigt. Die angegebenen Befehle liefern nur einen Hinweis auf ihre Verwendung. Die genaue Syntax muß aus der Beschreibung des CP5613bzw. der MSDN (http://msdn.microsoft.com) bezogen werden.

Initialisierung : (1)
`DP_start_cp`(...)
`DP_open`(..., &`cpHandle`, ...)
`DP_get_pointer`(`cpHandle`, ..., &`mempointer`, ...)
`DP_set_mode`(`cpHandle`, `DP_STOP`, ...)
`DP_set_mode`(`cpHandle`, `DP_CLEAR`, ...)
`DP_set_mode`(`cpHandle`, `DP_OPERATE`, ...)

$Shut - Down$: (2)
`DP_set_mode`(`cpHandle`, `DP_CLEAR`, ...)
`DP_set_mode`(`cpHandle`, `DP_STOP`, ...)
`DP_set_mode`(`cpHandle`, `DP_OFFLINE`, ...)
`DP_release_pointer`(`cpHandle`, ...)
`DP_close`(`cpHandle`, ...)

$Initialisierung\ Hardware - Event$: (3)
`DP_init_sema_object`(`cpHandle`, `DP_OBJECT_TYPE_INPUT_CHANGE`, &`semaHandle`, ...)
`CreateThread`(..., (`LPTHREAD_START_ROUTINE`)`cThreadProc`, ...)

Synchronisierung : (4)
`WaitForSingleObject`(`semaHandle`, `timeout_in_milliseconds`)

$Shut - Down\ Hardware - Event$: (5)
`DP_delete_sema_object`(`cpHandle`, `semaHandle`, ...)
`closeHandle`(`threadHandle`)

4 Weiterführende Beispiele

Die folgenden Beispiele müssen von Ihnen nicht zur Vorbereitung auf den Praxistest bearbeitet werden. Sie sollen nur als Anreiz dienen, um Interessierten mögliche komplexere Aufgabenstellungen näherzubringen.

4.1 Linearitätsmessung

Implementieren Sie eine Mess-Station zur Erfassung des Ausmasses der Nicht-Linearität der Distanzmesseinrichtung DIST des Targetsystems. Die Mess-Station soll einen einheitlichen Bewegungsablauf des Schlittens durchführen und die Unterschiede der gemessenen DIST-Werte und tatsächlichen ENCODER-Werte kontinuierlich am Bildschirm darstellen. Nach Abschluß des Bewegungsablaufes soll der Bediener der Meß-Station über die Kommandoelemente des Targets ein Protokoll des Tests in Form eines Files generieren können, das anschließend durch eine Tabellenkalkulation als Diagramm dargestellt werden kann. Weiters soll eine Testfunktionalität implementiert werden, die es dem Bediener ermöglicht, verschiedenen Kompensationsmethoden zu testen.

Inhalt Linearitätsmessung:

Beschreibung	Typ	Target	DP-Master
Digital-IO	Slave-Daten	Weiterleitung	Auswertung
ASi-Digital	Slave-Daten	Weiterleitung	Auswertung
ASi-Analog	Slave-Daten	Auswertung	-
Schrittketten	Funktion	Ja	Ja
ENCODER-Auswertung	Master-Funktion	-	Ja
Protokollierung	Master-Funktion	-	Ja
Kompensationen	Slave-Funktion	Ja	-

4.2 Positioniersteuerung

Implementieren Sie mithilfe des Motorcontrollers (MC_SPEED) und des Absolutwertgebers (ENCODER) des Targetsystems eine Positioniersteuerung für den Schlitten. Dabei sollen die Funktionen einer modernen "intelligenten" Achse nachgestellt werden.

Referenzfahrt: Die Achse kann durch mehrere Arten an einen Nullpunkt herangeführt werden, der letztendlich als Referenzpunkt für alle nachfolgenden Aktionen benutzt wird.

Handfahrt: Die Achse kann durch entsprechende Tasten händisch gesteuert werden.

Positionstabelle: Bis zu 8 verschiedene Positionen können angegeben werden, wobei das Fahrprofil – bestehend aus Beschleunigung, Geschwindigkeit, Verzögerung und weiteren Sonderfunktionen (Schaltbedingung, Verzögerung, etc.) – für jede Position einzeln angeben werden kann.

Positionsfahrt: Eine Position aus der Positionstabelle kann entsprechend ihres Fahrprofils angefahren werden.

Dateninterface: Die Ein- und Ausgabe von Achsdaten erfolgt an der Mensch-Maschine-Schnittstelle des DP-Masters.

Einheitenumrechnung: Die Eingabe und Anzeige von Positions-, Geschwindigkeits- und Beschleunigungswerten erfolgt in korrekten Einheiten (mm, cm, m, ft, inch, etc.), die durch Umrechnungsfaktoren ermittelt werden.

Inhalt Positioniersteuerung:			
Beschreibung	Typ	Target	DP-Master
Digital-IO	Slave-Daten	Auswertung	-
ASi-Digital	Slave-Daten	Auswertung	-
ASi-Analog	Slave-Daten	Auswertung	-
ENCODER	Slave-Daten	Auswertung	Weiterleitung
Schrittketten	Slave-Funktion	Ja	-
Motoransteuerung	Slave-Funktion	Ja	-
Positionstabelle	Master-Funktion	-	Ja
Einheitenumrechnung	Master-Funktion	-	Ja

4.3 Prozeßvisualisierung

Die Funktionsweise der Materialerkennung aus der zweiten Beispielsammlung soll hier implementiert werden, wobei die Interaktion mit dem Targetsystem ausschließlich über die Mensch-Maschine-Schnittstelle des DP-Masters stattfindet. Weiters soll der Vorgang der Materialerkennung detailliert am Bildschirm visualisiert werden. Somit sind alle relevanten Sensoren und Aktoren mit ihrem momentanen Wert in geeigneter Form darzustellen. Darüberhinaus soll das Ergebnis der Materialerkennung mit Datum und Uhrzeit in einem File protokolliert werden.

Inhalt Prozessvisualisierung:			
Beschreibung	Typ	Target	DP-Master
Digital-IO	Slave-Daten	Auswertung	Visualisierung
ASi-Digital	Slave-Daten	Auswertung	Visualisierung
ASi-Analog	Slave-Daten	Auswertung	Visualisierung
ENCODER	Slave-Daten	-	Auswertung
Schrittketten	Funktion	Ja	Ja
Materialerkennung	Slave-Funktion	Ja	-
Protokollierung	Master-Funktion	-	Ja

4.4 Kommandopultsimulation

Das Targetsystem soll als Mensch-Maschine-Schnittstelle für ein simuliertes Fahrzeug dienen, wobei der Joystick in beiden Achsen als richtungsweisendes, die Leuchtdrucktaster als befehlsgebendes und der SONAR-Sensor als umgebungserfassendes Element operieren. Der DP-Master hat hierbei die Aufgabe, die Reaktionen des Fahrzeuges auf die simulierte (oder durch den Sonar "erfaßte") Umgebung zu simulieren und beides – also Fahrzeug und Umgebung – zu visualisieren. Um die Reaktionen eines echten Systems möglichst realistisch simulieren zu können, wird eine Achse des Fahrzeuges nicht numerisch, sondern durch die Reaktionen des Ersatzsystems Schlitten-ENCODER simuliert.

Inhalt Kommandopultsimulation:			
Beschreibung	Typ	Target	DP-Master
Digital-IO	Slave-Daten	Weiterleitung	Auswertung
ASi-Digital	Slave-Daten	Weiterleitung	Auswertung
ASi-Analog	Slave-Daten	Weiterleitung	Auswertung
ENCODER	Slave-Daten	-	Auswertung
Schrittketten	Master-Funktion	-	Ja
Realachsen-Reaktion	Slave-Funktion	Ja	-
Fahrzeugsimulation	Master-Funktion	-	Ja

4.5 ASi-DP-Gateway

Um Datenwerte von Slaves aus dem ASi-Netzwerk des Targetsystems flexibel an den Master weiterreichen zu können, soll die S7-214 SPS als programmierbares Gateway dienen. Als ASi-DP-Gateway soll das Targetsystem für 3 Betriebsmodi programmiert werden:

Konfigurationsmodus: In diesem Modus soll das Targetsystem durch ein DP-Master-Konfigurationsprogramm konfiguriert werden. Anzahl, Art (digital oder analog) und Übertragungsform (skaliert/unskaliert bzw. Integer-/Realzahlen) der Slavedaten sowie deren Mapping auf den DP-Block soll über menügeführte Benutzerinteraktion eingestellt werden können.

Betriebsmodus: In dieser Betriebsart soll das Targetsystem die aktuellen Slavedaten zyklisch in der zuvor festgelegten Form zum DP-Master übertragen, wobei auf der Masterseite ein Beobachtungsprogramm die Daten am Bildschirm anzeigen soll.

Diagnosemodus: In diesem Modus soll es dem DP-Master-Konfigurationsprogramm ermöglicht werden, Fehlerzustände im ASi-DP-Gateway (z.B. bei Auswertung eines Analog-Slaves) auszulesen.

Das Konfigurationsprogramm und das Beobachtungsprogramm werden hierbei in einem Programm zusammengefaßt, das bei Aufruf die eine oder die andere Funktionalität übernimmt. Weiters soll es möglich sein, Konfigurationen des Gateways in Dateien zu speichern und wieder zu laden.

Inhalt ASi-DP-Gateway:			
Beschreibung	Typ	Target	DP-Master
Digital-IO	Slave-Daten	Auswertung	Visualisierung
ASi-Digital	Slave-Daten	Auswertung	Visualisierung
ASi-Analog	Slave-Daten	Auswertung	Visualisierung
Schrittketten	Slave-Funktion	Ja	-
Gateway	Slave-Funktion	Ja	-
Konfiguration	Master-Funktion	-	Ja