Automatisierungssysteme

Thierry Prud'homme
thierry.prudhomme@hslu.ch

Aufgabenserie: **#2** Themen: **IEC 61131 - Einführung Teil 2**

[Aufgabe 1] (*Funktion*) Erstellen Sie ein neues Projekt und fügen Sie unter Bausteine (POUs) eine neue Funktion hinzu und benennen Sie den Baustein `AdditionFunction`. Konfigurieren Sie Ihre Funktion so, dass Sie zwei Eingangsparameter vom Typ `REAL` aufnimmt, diese addiert und die Summe zurückgibt. Rufen Sie Ihre Funktion `Addition` mit verschiedenen Eingangswerten von ihrem `MAIN` Programm auf und überprüfen Sie die Ergebnisse.

Gehen Sie gleichermassen vor für eine Subtraktion, eine Multiplikation und eine Division. Vergessen Sie nicht den Sonderfall: „Division durch 0" abzufangen. Rufen Sie alle vier Funktionen von Ihrem Hauptprogramm auf und speichern Sie die Rückgabewerte in lokalen Variablen.

[Aufgabe 2] (*Debugger*) Setzten Sie einen Breakpoint (mit Online – Breakpoint an/aus oder F9) innerhalb einer Ihrer Funktion. Verfolgen Sie schrittweise mit F8 und F10 wie sich Ihre Funktion verhält. Hat Ihre Funktion ein Gedächtnis?

[Aufgabe 3] (*Programm*) Fügen Sie Ihrem Projekt ein Programm namens `DoSomeMath` hinzu. Das Programm erwartet zwei Eingabeparameter. Innerhalb Ihres Programms rufen Sie die vier Funktionen `Addition`, `Subtraction`, `Multiplication` und `Division` auf und speichern die Rückgabewerte in lokalen Variablen innerhalb des Programms ab. Implementieren sie eine Zählervariable, die abspeichert wie viel Mal die Funktion aufgerufen wurde. Rufen Sie Ihr Programm von ihrem `MAIN` Programm aus auf. Überprüfen Sie die Abläufe mit dem Debugger. Was ist der Unterschied zwischen einer Funktion und einem Programm? Wie können Sie auf Variablen eines Programms zugreifen?

[Aufgabe 4] (*Bit Shifting*) Erstellen Sie ein neues Projekt. Definieren Sie eine Variable `myWord` des Typs `WORD` und initialisieren sie die Variable mit dem hexadezimalen Wert 5D7A. Verwenden Sie den Befehl `SHR` (bitwise right-shift) um eine Division durch 8 zu erreichen. Welches Ergebnis (hexadezimal und dezimal) erhalten Sie? Überprüen Sie Ihr Ergebnis von Hand oder mit MATLAB. Tipp: Sie können unter Projekt – Optionen.. – Editor die Anzeigeart von Variablen anpassen.

Definieren Sie eine neue Variable `myDWord` des Typs `DWORD`. Führen Sie ein `SHL` (bitwise left-shift) aus um `myVar` mit 8 zu multiplizieren und in `myDWord` abzuspeichern.

[Aufgabe 5] (Bool*'sche Operationen*) Setzten Sie die höchsten vier bit von `myDWord` mit einem `OR`-Befehl auf 1.

[Aufgabe 6] (*Status-Wort*) Jetzt sind Sie bereit ein Status-Wort zu verarbeiten, das für eine RS232-Kommunikation verwendent wird. Das Status-Wort stellt sich aus den folgenden bits zusammen:

Bit	SW.15	SW.14	SW.13	SW.12	SW.11	SW.10	SW.9	SW.8
Name	IL7	IL6	IL5	IL4	IL3	IL2	IL1	IL0

Bit	SW.7	SW.6	SW.5	SW.4	SW.3	SW.2	SW.1	SW.0
Name	-	OR_ERR	FR_ERR	PA_ERR	BUF_F	IA	RR	TA

- IL7 .. IL0 (InLength): Anzahl zu übertragende Bytes
- OR_ERR (Overrun Error): Empfangene Daten werden nicht in das FIFO geschrieben
- FR_ERR (Framing Error): Empfangene Daten werden nicht in das FIFO geschrieben
- PA_ERR (Parity Error): Empfangene Daten werden nicht in das FIFO geschrieben
- BUF_F (BufferFull): Empfangs-FIFO ist voll
- IA (InitAccepted-Bit): Initialisierung ist ausgeführt
- RR (ReceiveRequest): DataIn-Bits liegen vor
- TA (TrasmitAccepted): Quittierung der Entgegennahme von Daten

Schreiben Sie eine Funktion, die als Eingangsparameter ein Status-Wort (Typ `WORD`) erwartet. Die Funktion soll mit Hilfe von Bitshifting und Bool'schen Operationen folgende Rückgabeparameter extrahieren:

- `inLenght` in `INT`
- `overrunError` als `BOOL`
- `framingError` als `BOOL`
- `parityError` als `BOOL`
- `bufferFull` als `BOOL`
- `initAccepted` als `BOOL`
- `receiveRequest` als `BOOL`
- `transmitAccepted` als `BOOL`

Überprüfen Sie die Funktionalität Ihrer Funktion mit den Folgenden Status-Wörtern:

- 0x1D06
- 0x534C
- 0xC705

[Aufgabe 7] (*Array*) Definieren Sie ein lokales Array des Typs `REAL` und initialisieren Sie es mit zehn beliebigen Zahlen.

[Aufgabe 8] (`FOR`-*Schleife*) Programmieren Sie in `MAIN` eine `FOR` Schleife die mit Hilfe des Programms `DoSomeMath` die Summe, sowie das Produkt aller Werte Ihres Arrays.

[Aufgabe 9] (`WHILE`-*Schleife*) Schreiben Sie Ihr Beispiel um und verwenden Sie eine `WHILE` -Schleife. Warum sind `WHILE`-Schleifen nicht sehr geeignet für ein zyklisch ausgeführtes Programm.

[Aufgabe 10] (`REPEAT`-*Schleife*) Wie unterscheidet sich die `REPEAT`-Schleife von der `WHILE`-Schleife? Für was wird das Keyword `EXIT` verwendet?

[Aufgabe 11] (*Funktionsgenerator*) Erstellen Sie eine Funktion `SinusFunction`, die ein Sinussignal $y(t) = A\sin(\omega t) + y_{off}$ generiert. Die Eingangsparameter sind die Amplitude A, der Offset y_{off}, die Kreisfrequenz ω in (rad/s) und die Zeit t. Der Rückgabewert ist $y(t)$. Die Zeit t sollten Sie im aufrufendem Programm inkrementieren ($t = t + T$), wobei T die Zykluszeit des Tasks ist.

Erstellen Sie eine weitere Funktion (`SquareSignal`), die ein Rechtecksignal generiert. Die Funktion erwartet die Parameter Minimalwert y_{min}, Maximalwert y_{max}, die Kreisfrequenz ω in (rad/s) und einem Offset y_{off} Tipp: Sie können einen grossen Teil Ihrer Sinusfunktion übernehmen.

Erstellen Sie noch eine dritte Funktion (`PwmSignal`), die ein pulsweitennmoduliertes Signal zurückgibt. Welche Parameter benötigen Sie?

[Aufgabe 12] (*Scope*) Beckhoff stellt ein sogenanntes TwinCAT Scope View zur Verfügung. Mit diesem Werkzeug können Daten online visualisiert werden. Öffnen Sie TwinCAT Scope View und fügen Sie ihrem Scope eine neue Scope View hinzu, dort hängen Sie dann zwei Kanäle an. Konfigurieren Sie die Kanäle so, dass sie die Verläufe Ihres Sinus- und Rechtecksignals visualisieren können. Exportieren Sie die Verläufe in eine ASCII Datei und importieren Sie die Datei in Matlab. Plotten Sie die Verläufe.

[Aufgabe 13] (*Funktionsblock*) Fügen Sie Ihrem Projekt einen Funktionsblock mit der Bezeichnung `EdgeDetector` an. Die Eingangsvariable heisst `inputValue` und die Ausgangsvariablen `risingEdge` und `fallingEdge`. Das Ziel dieses Funktionsblocks ist es, die fallenden und steigenden Flanken eines Bool'schen Signals zu erkennen. Für die Verwendung ihres Funktionsblocks müssen Sie diesen instanziieren. Kreieren Sie eine Instanz für den gelben und eine für den blauen Taster und überprüfen Sie die Funktionalität.

Erweitern Sie Ihren `EdgeDetector` für Doppelbetätigungen. Tipp: Definieren Sie eine zusätzliche Eingangsvariable, die die Zeitspanne für die Erkennung einer Doppelbetätigung festlegt. Auf diese Weise können Sie die Zeitspanne anschliessend optimal anpassen.

[Aufgabe 14] (*Toggling*) Ergänzen Sie ihr Programm so, dass wenn der blaue Taster gedruckt wird (`risingEdge = TRUE`), die beiden Lämpchen regelmässig ein- und wieder ausgeschaltet werden. Wählen Sie eine Frequenz von 3 Hz und stellen Sie sicher, dass bei erneutem Tastendruck die Lämpchen wieder erlöschen. Verwenden Sie das Keyword `NOT`.

Für die gelbe Taste sollen die Lämpchen abwechslungsweise leuchten. Die blaue Taste soll jedoch Vorrang haben.

Verdoppeln Sie die Frequenz für Doppelbetätigungen der Taster.

[Aufgabe 15] (*Datentypen Konvertierungen*) An der Klemme EL3004 auf Ihrem Test Rig ist der Ultraschallsensor UB120-12GM von PEPPERL+FUCHS angeschlossen. Welche Betriebsspannung weist der Sensor auf? Was bedeutet „single-ended“?

Definieren Sie eine Eingangsvariable `intUltrasonicSensor`, die Sie mit dieser Hardware verknüpfen. Definieren Sie eine Variable `voltageUltrasonicSensor` vom Typ `REAL` die die Spannung in (V) beinhaltet. Programmieren Sie die Konvertierung von `intUltrasonicSensor` zu `voltageUltrasonicSensor`.

Geben Sie das Ergebnis anschliessend wieder auf dem Analog Output aus. Die Anzeige soll 0 V anzeigen im Ruhezustand. Testen Sie Ihre Anwendung, indem Sie einen Gegenstand dem Ultrassonic Sensor langsam nähern.

[Aufgabe 16] (*Buffer*) Deklarieren Sie ein `bufferArray` mit 100 Werten des Types `REAL` für die Werte Ihrer `voltageUltrasonicSensor` Variable.

Schreiben Sie eine Funktion `shiftLeft()`, die Ihr `bufferArray` bei einem Aufruf um eine Position nach links verschiebt:

`bufferArray` vorher:

k	$k+1$	$k+2$	$\ldots$	$k+98$	$k+99$	$k+100$

`bufferArray` nachher:

$k+1$	$k+2$	$k+3$	$\ldots$	$k+99$	$k+100$	$k+101$

[Aufgabe 17] (*Mittelwert Funktion*) Schreiben Sie eine weitere Funktion genannt `averageCalculator`, die den Mittelwert Ihres Arrays berechnet nach folgender Formel berechnet:

$$M(k) = \frac{1}{N} \sum_{j=0}^{N} y(j)$$

Das Signal $y(j)$ ist die einzige Eingangsvariable und in dieser Anwendung Ihr `bufferArray`. $M(k)$ ist der zurückgegebene Mittelwert.

Implementieren Sie das `bufferArray` und den `meanCalculator` für die Ausgabe der des Ultrasonic Sensor auf dem Analog Output.

[Aufgabe 18] (*Forgetting Factor*) Die rekursive Schreibweise der Mittelwertsberechnung lautet:

$$M(k) = \frac{y(k) + y(k-1) + y(k-2) + \ldots + y(k-N)}{N}$$

Eine Möglichkeit einen Mittelwert der letzten N Eingabewerte zu berechnen und auf einen Datenbuffer zu verzichten ist mithilfe einer Vereinfachung und der Einführung eines Forgetting Factors λ möglich. Wir schreiben den Mittelwert neu so:

$$M(k) = \frac{y(k) + \lambda^1 y(k-1) + \lambda^2 y(k-2) + \ldots + \lambda^N y(k-N)}{1 + \lambda^1 + \lambda^2 + \ldots + \lambda^N}$$

Zeigen Sie, dass unter der Berücksichtigung von $\lambda < 1$ und $N \to \infty$ folgende Beziehung gilt:

$$M(k-1) = \lambda M(k) + (1-\lambda) y(k-1)$$

Verwenden Sie diese Formel für die Programmierung eines Funktionsblocks der anhand eines Forgetting Factors (`forgettingFactor`) und einer Eingangsgrösse (`inputValue`) der Mittelwert eines Signals berechnet. Verwenden Sie eine lokale Variable für den Mittelwert (`meanValue`).

Instanzieren Sie Ihr Funktionsblock und benutzen Sie Ihn für die Ausgabe der des Ultrasonic Sensor auf dem Analog Output. Benutzen Sie TwinCAT Scope View um das gemittelte Signal mit dem Signal der vorangehenden Aufgabe zu vergleichen.

[Aufgabe 19] (*Struktur*) ST bietet die Möglichkeit Strukturen zu definieren. Unter dem Reiter Datentypen können Sie mit einem Rechtsklick ein Objekt einfügen. Legen Sie eine neue Struktur an mit der Bezeichnung `CurrentStatus`. Die Felder der Struktur sollen alle Eingabe- sowie Ausgabe-Geräte Ihres Test Rigs abbilden.

Name	Datentyp	Kommentar
`buttonYellow`	`BOOL`	Taster Gelb
`buttonBlue`	`BOOL`	Taster Blau
`lightGreen`	`BOOL`	Lampe Grün
`lightRed`	`BOOL`	Lampe Rot
`inductiveSensor`	`BOOL`	Induktiv Sensor
`intUltrasonicSensor`	`INT`	Ultrasonic Sensor
`voltageUltrasonicSensor`	`REAL`	Ultrasonic Sensor in Volt
`intAnalogOutput`	`INT`	Analog Anzeige
`voltageAnalogOutput`	`REAL`	Analog Anzeige in Volt

Legen Sie ein Array des Typs `CurrentStatus` an und ersetzen Sie damit Ihr `bufferArray`, damit die letzten 100 Werte aller Komponenten des Test Rigs zur Verfügung stehen. Verifizieren Sie Ihr neuer Buffer online.

[Aufgabe 20] (*Rudimentäre Visualisierung*) Es besteht die Möglichkeit eine rudimentäre Visualierung innerhalb von TwinCAT PLC Control zu entwickeln. Klicken Sie unten auf den dritten Tab Visualisierungen und fügen Sie ein neues Objekt ein durch einen Rechtsklick. Entwickelen Sie eine Visualisierung Ihrer `CurrentStatus`-Struktur. Benutzten Sie ein Zeigerinstrument für den Analog Output, sowie eine Balkenanzeige für den Ultrasonic Sensor. Die Bool'schen Variablen können Sie mit Rechtecken und Ellipsen visualisieren. Testen Sie Ihre Visualisierung im Betrieb.

[Aufgabe 21] (*Remanente Variablen*) Erkundigen Sie sich über die Bedeutung des Begriffs „Remanenz“. Lesen Sie dazu das Dokument BeckhoffApplicationNoteRemanenz.pdf und prüfen Sie diesen Link.

Es gibt zwei Arte von remanenten Variablen: die `RETAIN`- und die `PERSISTENT`-Variablen. Erkundigen Sie sich über die Unterschiede dieser zwei Typen von remanenten Variablen. Was ist ein UPS (uninterruptible power supply, USV auf Deutsch)? Was ist NOVRAM? Ist ein UPS oder NOVRAM auf Ihrer SPS vorhanden?

Kreieren Sie zwei Instanzen von Ihrer Buffer Struktur. Deklarieren Sie eine als `RETAIN` und eine als `PERSISTENT` und führen Sie Versuche durch um das Verhalten der Variablen zu untersuchen.