Eine SPS ohne Zykluszeit

Vorteile der High-Speed FPGA Steuerungen der ZANDER ZX Serie

von Prof. Helmut Maier

in Zusammenarbeit mit der H. Zander GmbH & Co.KG

zum Autor:

Prof. Helmut Maier, Jahrgang 1938, studierte an der Technischen Universität München Elektrotechnik mit Schwerpunkt Hochfrequenztechnik. Nach dem Studium war er 17 Jahre in verschiedenen industriellen Unternehmen als Entwicklungsingenieur, Laborleiter, Projektleiter etc. tätig und konnte praktische Erfahrungen sammeln auf Gebieten wie Analog- und Hybridrechner, Datenverarbeitung in der Medizin, Digitalisierung von Radardaten, Planung und Realisierung von Flugsicherungssystemen sowie optisch-elektronische Analysesysteme im Umweltbereich.

Von 1981 bis 2002 betreute er an der Hochschule Augsburg, Fakultät für Elektrotechnik, die Lehrgebiete Mathematik, Informatik, Prozessdatenverarbeitung, Mikroprozessortechnik, Steuerungs- und Automatisierungstechnik mit Schwerpunkt Speicherprogrammierbare Steuerungen. Sein großes Interesse galt der Entwicklung von Simulationen zur Ausbildung im Bereich der Automatisierung. Im Rahmen von Diplomarbeiten entstanden in seinem Labor Simulationsprogramme für Schützsteuerungen (ELSIM), Industrielle Prozesse (PROSIM) und SPS-Steuerungen (SPS-SIM, S7-SIM), die auch von der Industrie verbreitet wurden.

Seit 1998 nahm er in Südost-Asien (Thailand, Vietnam, Sri Lanka) eine Reihe von Lehraufträgen wahr (SPS, Digitaltechnik, Robotik, Industrial Automation) und betreute Hochschulprojekte (Entwicklung von Curricula) an diversen Universitäten. Von 2002 bis 2005 arbeitete er als Leiter eines GIZ-Projektes zur Einführung neuer praxisorientierter Master-Studiengänge am KMUTNB (King Mongkut's University of Technology, North Bangkok).

Seit Beginn seiner Hochschullaufbahn veröffentlichte Prof. Maier zahlreiche Berichte zu Themen der Automatisierungstechnik und angewandten Informatik.

Prof. Maier und seine thailändische Frau, Dr. Chayanee Maier, leben und arbeiten wechselweise in Utting am Ammersee und in Thailand, Großraum Bangkok.

Email: helmai@hs-augsburg.de

Eine SPS ohne Zykluszeiten

Wie in vielen anderen technischen Bereichen, so besteht auch in der industriellen Automatisierungstechnik eine eindeutige Tendenz zu immer höheren Verarbeitungsgeschwindigkeiten. Moderne industrielle Fertigungs- und Produktionsanlagen, seien es Verpackungs- oder Abfüllanlagen, Zuschneideinrichtungen oder Klebestationen, Kunststoff-Spritzmaschinen oder sonstige Handlingssysteme wie z. B. superschnelle Nockenschaltwerke erfordern daher immer schnellere Steuereinheiten, insbesondere im Bereich der Speicherprogrammierbaren Steuerungen (SPS). Auch beim Auftreten von Fehler- oder Alarmsituationen sind sowohl schnelle Signalverarbeitung wie extrem kurze Reaktionszeit unverzichtbar. Nur so lässt sich jederzeit die erforderliche Sicherheit für Personen und Anlagen gewährleisten. Für den Anwender leitet sich daraus eine weitere Forderung ab, nämlich dass die Steuerung bei zugleich hoher Geschwindigkeit auch immer garantiert deterministisch arbeitet. Fehlende Zyklen oder unterschiedlich lange Reaktionszeiten sind hier meist nicht tolerierbar.

Das Problem konventioneller Steuerungen

Der eindeutige Nachteil jeder konventionellen SPS war und ist nach wie vor die periodisch-sequentielle Abarbeitung der Programmanweisungen durch Mikrocontroller und die sich daraus ergebende Zeit für einen Programmdurchlauf, die so genannte „Zykluszeit“. Obgleich die Steuerungshersteller versuchten und versuchen, über intelligente Ansätze und mit beträchtlichem Aufwand die Bearbeitungszeit für SPS-Anweisungen zu verkürzen (bei modernen Steuerungen bewegt sich die Ausführungszeit für Binäranweisungen bei 10 ns und darunter), so können sie, trotz immer schnellerer Mikrocontroller, nicht die Tatsache umgehen, dass die Zykluszeit nahezu linear mit der Programmlänge ansteigt. Das Hauptproblem liegt jedoch nicht allein in der Länge, sondern vor allem auch in der Inkonstanz der Zykluszeit. Während der Programmbearbeitung auftretende Interrupts oder bedingte Programmsprünge und –schleifen können die Zyklus- und damit die Reaktionszeiten der SPS nahezu beliebig verändern. Diese Fluktuationen („Jitter“-Effekt) sind für viele Applikationen kritisch oder nicht tolerierbar.

Ein neues Steuerungskonzept

Die Firma Zander in Aachen hat diese Mängel längst erkannt und verfolgt daher bereits seit vielen Jahren ein völlig anderes Steuerungskonzept. Die im Anschluss mehr detailliert beschriebenen Steuerungen arbeiten ohne jede Zykluszeit und garantieren damit eine extrem hohe und insbesondere auch konstante Verarbeitungsgeschwindigkeit, die zudem unabhängig von der Länge des jeweiligen Anwenderprogramms ist. Herzstück dieser extrem schnellen Steuerungen ist nicht mehr ein sequentiell arbeitender Mikrocontroller, sondern ein programmierbarer Digitalbaustein, ein FPGA (Field Programmable Gate Array) oder ein CPLD-Chip (Complex Programmable Logic Device). Das bedeutet, die Steuerungen arbeiten ihr Programm intern absolut parallel in Echtzeit ab. Durch diese Art der Programmbearbeitung treten keine Zykluszeiten auf. Auch die oben erwähnten Probleme mit unterschiedlichen Zyklus- und damit Reaktionszeiten an der Maschine gehören damit der Vergangenheit an. Egal wie schnell nun beispielsweise eine Verpackungsmaschine läuft: der Zuschnitt des Materials wird stets passgenau gesteuert und die Beleimung sitzt immer an den richtigen Stellen. Da diese Steuerungen mit modernen Möglichkeiten der Vernetzung (Ethernet und andere Netzwerkstandards) ausgestattet sind und somit einen kontinuierlichen Datenaustausch mit weiteren integrierten Automatisierungskomponenten ermöglichen, sind sie auf die Neuerungen und Anforderungen von „Industrie 4.0“ bestmöglich vorbereitet.

Zwei Steuerungstypen

Bisher hat man bei den Steuerungstypen stets folgende zwei Hauptrichtungen unterschieden:

- Verbindungsprogrammierte und
- Speicherprogrammierbare Steuerungen, also die klassische SPS.

Bei verbindungsprogrammierten Steuerungen wird die Schaltlogik und somit das Programm durch feste Verbindungen zwischen den einzelnen Bauelementen realisiert. Dies können elektrische sowohl Steuerungen mit Kontakten und Relais wie auch pneumatische oder hydraulische Steuerungen sein. Auch bei elektronischen Steuerungen mit Logikgattern bestehen die Verbindungen in der Regel aus der (festen) Verdrahtung der Logikbausteine bzw. sie ergeben sich aus der mit Bauelementen bestückten gedruckten Schaltung einer Leiterplatte. Im Gegensatz zur SPS zeichnet sich diese Art Steuerungen durch parallele Abarbeitung der Steuerungsprogramme aus, allerdings auf Kosten der Flexibilität.

Mit Einführung einer Steuerung auf der Basis programmierbarer Digitalbausteine entfällt diese starre Abgrenzung. Durch die Tatsache, dass die FPGA- bzw. CPLD-Bausteine generell eine reversible Speicherung der Verbindungen ermöglichen, entsteht eine neuartige hybride Form aus verbindungs- und speicherprogrammierbarer Steuerung.

Prinzipielle Arbeitsweise einer SPS

Damit besser verständlich wird, worin sich die neuartige Steuerung der Firma Zander im Wesentlichen von einer klassischen SPS unterscheidet, sollen zunächst die zyklische Arbeitweise einer konventionellen SPS und die damit verbundenen Prozessabbilder etwas näher betrachtet werden. Fast allen heute vorherrschenden Speicherprogrammierbaren Steuerungen gemeinsam ist die bereits angeführte sich permanent zyklisch wiederholende Bearbeitung des Anwenderprogramms. Ein für die Beurteilung von Geschwindigkeit, Reaktionszeit und allgemeiner Leistungsfähigkeit einer SPS wesentlicher Parameter ist daher die Zyklusdauer, also die erforderliche Zeit zur Ausführung einer kompletten Programmbearbeitung durch die CPU (Central Processing Unit). Wie später dargelegt wird, steht diese Zykluszeit in einer festen Relation zur Reaktionszeit der SPS. Darunter versteht man die zeitliche Verzögerung zwischen dem Signalwechsel eines (binären) Eingangssignales und einer dadurch bewirkten Änderung eines Ausgangssignales.

Hauptprogramm
Der permanent zyklisch ausgeführte Teil eines Anwenderprogramms bildet das Hauptprogramm und entspricht der normalen Programmbearbeitung einer SPS. Die Ausführung von anderen, nicht zyklisch ablaufenden Programmteilen kann durch bestimmte Ereignisse wie Prozessalarme, Programm- oder Hardwarefehler etc. oder durch Zeitglieder wie Laufzeitüberwachung, Weck- oder andere Alarme ausgelöst werden. Derartige Programme werden jedoch zumeist nur ergänzend zum eigentlichen Hauptprogramm ausgeführt.

Prozessabbilder der Ein-/ Ausgänge
Eng mit der zyklischen Programmbearbeitung einer SPS verbunden sind die Speicherbereiche der Prozessabbilder. Diese bestehen aus folgenden zwei Teilbereichen:

- Dem „Prozessabbild der Eingänge – PAE“ als Abbild der digitalen Eingänge
- sowie dem „Prozessabbild der Ausgänge – PAA“ als Abbild der digitalen Ausgänge,

Zu Beginn eines Programmzyklus werden alle Signalwerte der Eingangs-Peripherie in den als „Prozessabbild der Eingänge – PAE“ bezeichneten Bereich des Systemspeichers geladen, vergleiche Bild 1. Über das PAE hat die CPU während des gesamten Programmzyklus den unmittelbaren Zugriff auf jedes beliebige anliegende Eingangssignal. Eine direkte Abfrage der aktuellen Signalzustände in den Eingabebaugruppen selbst findet demnach nicht statt.

Wird während der Programmbearbeitung ein Verknüpfungsergebnis (VKE) einem Ausgang zugewiesen, so spielt sich dies zunächst nur im „Prozessabbild der Ausgänge – PAA“ ab. Der Zustand des betreffenden Ausgangs in der Signalbaugruppe bleibt vorerst unverändert. Auch bei der Abfrage von Ausgängen für weitere Verknüpfungen greift das Programm nur auf das PAA zu. Erst nach Ende der Programmbearbeitung werden die neuen Werte des PAA in die zugehörigen Ausgabebaugruppen übertragen. Mit dieser Verfahrensweise wird es z. B. einfacher, die Eingänge zu „forcen“, d. h. zu Testzwecken auf einen bestimmten Wert zu zwingen, der nicht den tatsächlichen Eingabesignalen entspricht. Außerdem kann die CPU während der Programmbearbeitung mit unveränderlichen und somit konsistenten Ein- und Ausgabesignalen arbeiten. Wie dem folgenden Abschnitt „Zyklische Programmbearbeitung“ zu entnehmen ist, werden die Prozessabbilder vom Betriebssystem zyklisch-periodisch aktualisiert.

Die zyklische Programmbearbeitung

Wie aus Bild 1 zu ersehen beginnt jeder Programmzyklus mit der Aktualisierung des Prozessabbildes der Eingänge durch Kopieren der aktuellen an den Eingabebaugruppen anliegenden Gebersignale in das PAE. Signalzuweisungen an die Ausgänge werden während der Bearbeitung des Anwenderprogramms noch nicht an die Ausgabebaugruppen ausgegeben, sondern zunächst in das Prozessabbild der Ausgänge (PAA) geschrieben. Erst nach Abschluss des Hauptprogramms, nach Bearbeiten der letzten Steueranweisung, werden die aktuell ermittelten Signalzustände für die Ausgangswerte vom Prozessabbild der Ausgänge in die Ausgabebaugruppen übertragen. Bild 1 zeigt ein vereinfachtes Schema der Vorgänge bei der zyklischen Programmbearbeitung.

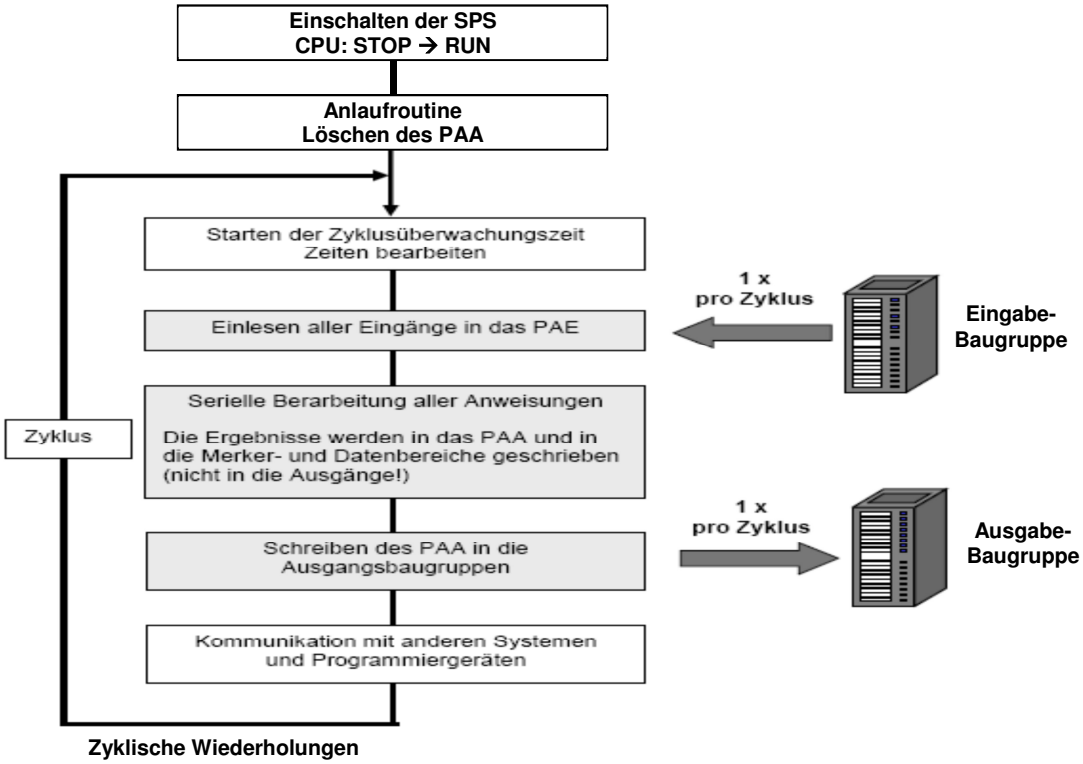

Bild 1: Vereinfachtes Schema der zyklischen Programmbearbeitung

Diese ständig sich wiederholende serielle Bearbeitung aller Anweisungen hat folgende Konsequenzen:

- Das Programm wird grundsätzlich nur mit Werten aus den Prozessabbildern der Ein- und Ausgänge bearbeitet. Andernfalls könnten die Eingangssignale während des Zyklus mit unterschiedlichen Zuständen in die Programmbearbeitung eingehen.
- Den Ausgangssignalen dürfen im Programm nicht mehrmals Werte zugeordnet werden, weil zuvor ermittelte Werte im PAA durch nachfolgende Zuweisungen überschrieben werden. Diese gefürchteten „Mehrfachzuweisungen“ sind immer Ursache von Programmfehlern!

Die Zykluszeit

Die Zeit, die der Prozessor einer SPS zur Abarbeitung eines kompletten Programmdurchlaufes, also eines OB1-Durchlaufes (siehe hierzu Bild 2: „Unterschiedliche Zykluszeiten“), einschließlich aller diesen Durchlauf unterbrechenden Programmteile und Systemtätigkeiten benötigt, bezeichnet man als “Zykluszeit”. Sie ist somit abhängig von der Anzahl und Art der Programmanweisungen und sonstigen Kommandos, die während einer vollständigen Programmbearbeitung ausgeführt werden. Zur Zykluszeit tragen aber auch diverse Laufzeiten des Betriebssystems bei, beispielsweise für die Systemsteuerung, die Aktualisierung der Prozessabbilder, die Bearbeitung von Zeitfunktionen sowie zeitliche Belastungen durch Kommunikationsaufgaben. Tasks mit höherer Priorität wie Alarm- oder Fehler-Organisationsbausteine unterbrechen den Zyklus und verlängern damit ebenfalls die Zykluszeit. Die Zykluszeit wird von der CPU überwacht. Überschreitet sie einen einstellbaren Maximalwert (typisch 150 ms), so geht die CPU in den Stopp-Zustand.

Bearbeitungszeit
Die Zykluszeit ist als wichtiger Parameter kennzeichnend für die Leistungsfähigkeit und Bearbeitungsgeschwindigkeit einer SPS. Daher geben die Hersteller üblicherweise in den Leistungsdaten Werte zur Bearbeitung von 1000 (“1 k”) AWL-Befehlen an, um die Leistungsklasse der jeweiligen CPU zu spezifizieren. Beispielsweise benötigen die schnellsten CPUs der SIMATIC S7-300 Familie zur Bearbeitung von Bitoperationen 0,004 µs, womit man eine Zykluszeit von 4 µs für 1000 Anweisungen erhält. Für die leistungsfähigsten Prozessoren der neuen SIMATIC S7-1500 gibt der Hersteller 1 ns für Bitoperationen an. Das bedeutet eine Zykluszeit von 1 µs für 1 k AWL-Befehle. Wortoperationen für Festpunkt- oder Gleitpunktarithmetik weisen jedoch erheblich höhere Bearbeitungszeiten auf.

Der ständige Zuwachs an Verarbeitungsgeschwindigkeit folgt dem gleichen Trend, der bei PCs und allen Arten von Prozessoren festzustellen ist. Sowohl der PC wie die CPU einer SPS, basieren auf modernen Mikroprozessoren, deren Leistungsmerkmale sich in großen Schritten fortentwickeln. Schnellere Schaltkreis-Technologien, höhere Taktfrequenzen und fortgeschrittene Prozessorarchitekturen treiben die Prozessorleistungen stetig in die Höhe. Auch Mehr-Prozessor Lösungen innerhalb der CPU fortgeschrittener SPSen tragen zur Leistungssteigerung bei. So kombinieren die Hersteller Standard-Mikroprozessoren mit speziell entwickelten Bit-Prozessoren. Letztere sind darauf zugeschnitten, binäre Verknüpfungen mit höchstmöglicher Geschwindigkeit auszuführen, während normale Mikroprozessoren für die Wort-Verarbeitenden Operationen wie Festkomma- und Gleitkomma-Arithmetik sowie für die vielseitigen mit dem Betriebssystem und den Systemprogrammen verbundenen Aufgaben zuständig sind.

Unterschiedliche Zykluszeiten
Die Zykluszeit schließt auch die Laufzeiten all derjenigen Programmteile ein, die von Organisationsbausteinen höherer Priorität aufgerufen werden und so das Hauptprogramm des OB1 unterbrechen können. Sie ist daher nicht für alle Zyklen gleich lang. Das folgende Bild 2 veranschaulicht die Ursachen für unterschiedliche Zykluszeiten (TZ1 und TZ2). TZ2 ist deutlich größer als TZ1, weil im Zyklus 2 der bearbeitete OB 1 durch einen Uhrzeitalarm-OB (hier: OB 10) unterbrochen wird, was eine Verlängerung der Zyklusdauer bewirkt.

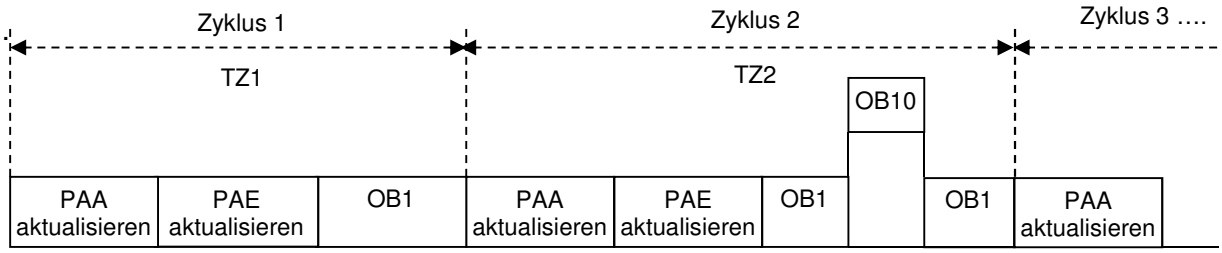

Bild 2: Ursachen unterschiedlicher Zykluszeiten (TZ1 und TZ2)

Es gibt noch weitere Ursachen für unterschiedlich lange Zyklus- und damit Bearbeitungszeiten. Die Bearbeitungszeit von Bausteinen, z. B. von OB 1, kann variieren wegen:

- bedingter Befehle, z. B. bedingter Programmsprünge, Verzweigungen,
- bedingter Bausteinaufrufe,
- unterschiedlicher Bearbeitungspfade für Programme,
- Programmschleifen etc.

Reaktionszeiten

Die Reaktionszeit einer SPS lässt sich definieren als Zeitdauer, die zwischen einem Signalwechsel im Eingabebereich und einer dadurch hervorgerufenen Reaktion mit Änderung eines Ausgangssignals vergeht. Durch die Einführung der Prozessabbilder ist die Reaktionszeit einer SPS eng mit ihrer Zykluszeit verknüpft. Diese Verzögerung ist eindeutig abhängig von der Programm-Zykluszeit und liegt, wie nachfolgend anhand von Bild 3 gezeigt, zwischen mindestens einer und im ungünstigsten Fall zwei Zykluszeiten.

Fall a
Im Fall a (Bild 3a) ändert sich ein Eingangssignal, z.B. durch Anfahren eines Grenztasters, unmittelbar vor der Aktualisierung des PAE. Somit kann dieser eingangsseitige Signalwechsel sofort in das Prozessabbild der Eingänge übernommen und in der darauf folgenden Bearbeitung des Hauptprogramms bereits im Zyklus n berücksichtigt werden. Angenommen, das Programm nutzt diesen Signalwechsel zum Rücksetzen eines Ausgangs und damit zum Abschalten eines Antriebs. Nach dem Programmende steht der geänderte Ausgangswert im PAA und wird in die zugeordnete Ausgangsbaugruppe übertragen zum Abschalten des Motors. Für diese optimale Situation (“best case”) beträgt die Reaktionszeit etwa eine Programm-Zykluszeit.

Fall b
Im Fall b (Bild 3b) war zum Zeitpunkt der Änderung des Eingangssignals die Aktualisierung des PAE gerade beendet. Der Signalwechsel wird, etwa um eine Zyklusdauer verzögert, erst im Verlauf der nächsten Aktualisierung des PAE in das Prozessabbild übernommen. Daher kann das geänderte Eingangssignal erst im Zyklus n+1 vom Programm berücksichtigt werden. Nach dem Programmende steht wieder der geänderte Ausgangswert zum Abschalten des Antriebs im PAA und wird in die entsprechende Ausgangsbaugruppe übertragen. Als Reaktionszeit ergibt sich jedoch für diese Konstellation (“worst case”) eine

Dauer von etwa zwei Programm-Zykluszeiten. Zur Zykluszeit selbst addieren sich noch die Verzögerungszeiten der Eingabebaugruppen (typisch: einige ms) sowie die Ansprechzeiten der eigentlichen Aktoren, z.B. eines Relais oder Magnetventils (typisch: 20-30 ms).

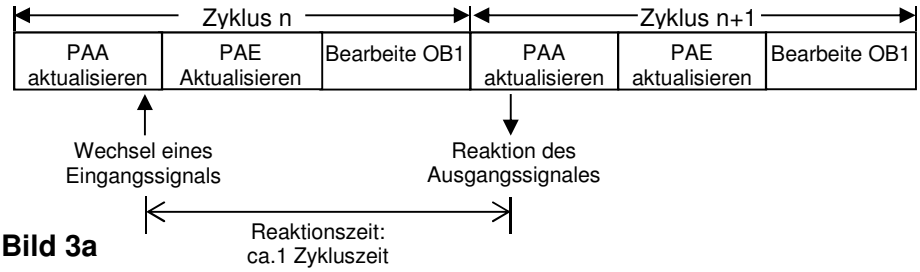

Bild 3a

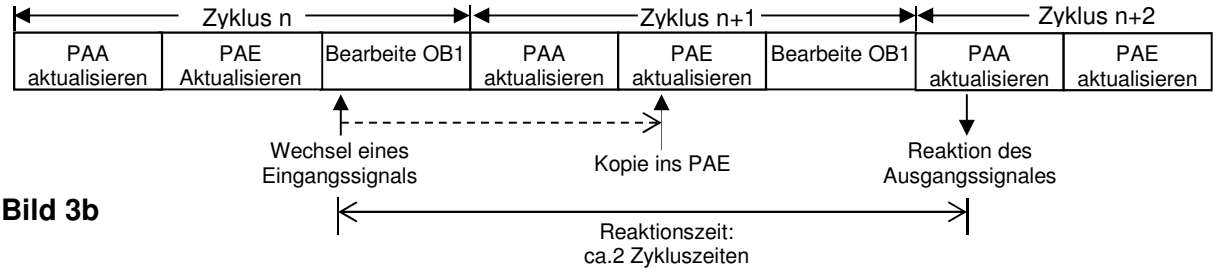

Bild 3b

Bild 3: Die Reaktionszeit bewegt sich zwischen 1 und 2 Zykluszeiten

All diese Effekte zeigen erhebliche Nachteile der zyklischen Programmbearbeitung auf. Weil die Bearbeitungs- und damit die Reaktionszeit schwanken kann, ist das System streng genommen nicht deterministisch! Zeitkritische Programmteile werden durch Änderungen im Programmablauf beeinflusst, auch wenn im Programm selbst keine Veränderungen erfolgen. Es entsteht daraus eine Art „Jitter“, worunter zufallsbedingte unterschiedliche Verzögerungen, Fluktuationen oder Schwankungen zu verstehen sind. Die Herausforderung besteht also weniger im absoluten Betrag der Reaktionszeit, sondern vielmehr im Sicherstellen exakt reproduzierbarer Schaltpunkte, die sich auch bei unterschiedlichen Maschinengeschwindigkeiten nicht verschieben. Gerade wegen dieser schwankenden Reaktionszeiten können herkömmliche SPS-Steuerungen die Echtzeitforderungen mancher Anwendungen in heutigen Automatisierungsanlagen oft nicht erfüllen.

Echtzeitverhalten

Man kann nun das Zyklusverhalten einer SPS klassifizieren als

1. nicht störend
2. störend, aber noch akzeptabel oder
3. prohibitiv, also nicht mehr tragbar.

Zur ersten Gruppe zählen Prozesse, deren Zeitkonstanten sehr lang sind gegenüber der Zykluszeit, wie es z. B. bei Temperaturregelungen sowie alle Anwendungen der Fall ist, in denen eine Maschine zeitlich nie ausgelastet und daher nicht Taktzeit - bestimmend ist.

Viele Anwendungen fallen in die 2. Gruppe. Bei Bewegungsabläufen sorgt man z. B. dafür, dass Initiatoren auf Nocken treffen, die lang genug sind, dass eine Position innerhalb der Zykluszeit sicher erkannt wird. Notfalls programmiert man eine direkte Peripherieabfrage im SPS-Programm mehrfach, um die Zykluszeit scheinbar zu verkürzen. Wenn das nicht reicht, kann man interruptfähige Eingabebaugruppen einsetzen. Doch selbst die Antwortzeit auf

Interrupts zeigt meist noch ein Zyklusverhalten. Interruptfähige Eingänge konkurrieren zudem mit Timern, Alarm-OBs, Kommunikations- und anderen Verwaltungsaufgaben.

Bei Bewegungen, beispielsweise einer Maschinenachse, mit einer relativ bescheidenen Geschwindigkeit von 1 m/s beträgt die Zeit zum Zurücklegen einer Weglänge von 1 mm eben nur 1 ms. Doch manchmal reicht selbst eine Auflösung oder Genauigkeit von 1 mm nicht mehr aus oder aber die Geschwindigkeit von 1 m/s ist für viele Vorgänge moderner Anlagen deutlich zu gering. Derartigen und höheren Anforderungen an Echtzeitbedingungen ist das Zyklus- und Reaktionsverhalten einer konventionellen SPS häufig nicht mehr gewachsen.

Programmierbare Logikbausteine

Bei einer Programmierung mit üblichen SPS-Programmiersprachen, z. B. in Anweisungsliste oder der Kontaktplandarstellung, muss man immer das zeitliche Nacheinander der Programmbearbeitung berücksichtigen. Man sollte sich jedoch alle Pfade eines Kontaktplanes so vorstellen, wie sie eigentlich gemeint waren: als eine elektrische (Relais, Schütz) oder elektronische Schaltung, in der alle Verknüpfungen parallel, also gleichzeitig ausgeführt werden. Viele Steuerungsaufgaben, die ansonst den Aufbau einer speziellen Elektronik erfordern, lassen sich wesentlich eleganter und flexibel mit einer auf Programmierbaren Logikbausteinen basierenden Steuerung lösen. Auch hierbei werden alle programmierten Verknüpfungen stets parallel und somit gleichzeitig bearbeitet, was zu extrem schnellen Reaktionszeiten führt. Zum besseren Verständnis der anschließend beschriebenen SPS ohne Zykluszeiten sollen zunächst die Prinzipien Programmierbarer Logikbausteine kurz vorgestellt werden.

Prinzipien programmierbare Logik

Anders als normale logische Bausteine wie Gatter, Flipflops oder Register, denen eine feste Funktion vorgegeben ist, erhalten Programmierbare Logikbausteine oder „Programmable Logic Devices“, kurz PLD, erst nach der Herstellung ihre funktionellen Eigenschaften durch eine entsprechende Programmierung oder „Konfiguration“. Sie bieten damit die Möglichkeit, digitale Schaltungen in nahezu beliebigem Umfang und hoher Komplexität aufzubauen. Unter Programmierbarer Logik versteht man heute weniger die programmierbaren "Urväter" wie PAL („Programmable Array Logic“) oder GAL (“Generic Array Logic“), sondern vielmehr Bausteine, die sich in einer Hochsprache wie VHDL oder Verilog beschreiben lassen, also CPLDs (Complex Programmable Logic Devices) oder FPGAs (Field Programmable Gate Arrays).

CPLDs

CPLDs, komplexe programmierbare Logikbausteine, weisen im Vergleich zu "gewöhnlichen" PALs und GALS flexibel programmierbare Makrozellen auf und sind mit mehr Funktionen sowie vielseitig nutzbaren Schaltkreisanschlüssen ausgestattet. So lassen sich Anschlüsse wahlweise als Eingänge, Ausgänge oder auch bidirektional verwenden. Manche Typen bieten zudem kombinatorische Verknüpfungsmöglichkeiten, die weit über die relativ einfachen UND-ODER-Strukturen von PALs hinausgehen. CPLDs sind daher praktisch keinen Einschränkungen bezüglich der Verwendbarkeit der UND-/ODER-Matrizen im Eingangsbereich unterworfen. Über eine zentrale Schaltmatrix werden mehrere kleinere PLD-Blöcke flexibel miteinander verbunden. Darüber hinaus verfügen sie alle über eine bestimmte Anzahl an Registern. Bild 4 zeigt die prinzipielle Architektur eines CPLD’s. Somit eignen sich diese Bausteine besonders für den Aufbau von umfangreichen Schalt- und Steuerwerken, Zustandsmaschinen etc. CPLDs lassen sich elektronisch programmieren und wieder löschen. Sie sind, wie auch FPGAs, in einheitlichen Hardware-Programmiersprachen wie beispielsweise VHDL programmierbar.

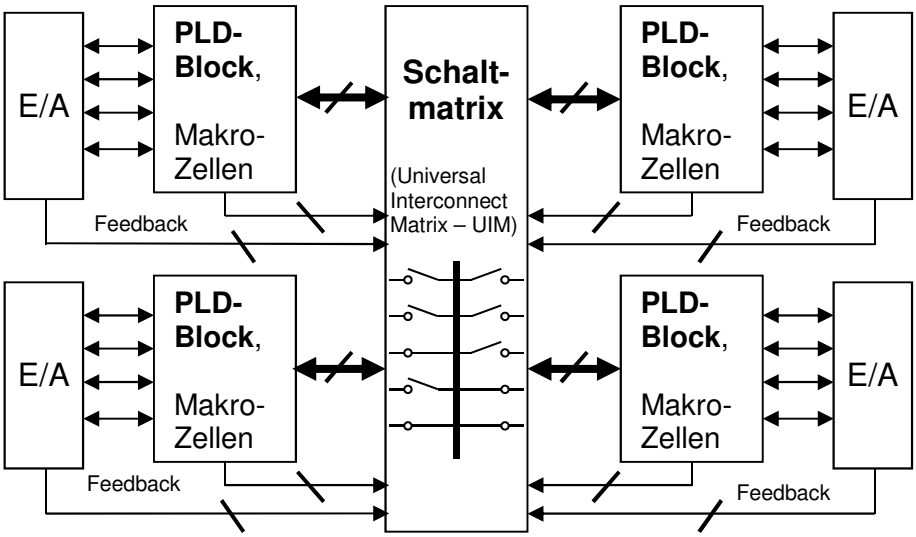

Bild 4: Prinzipielle Architektur eines CPLD's

Programmpeicher
Da CPLDs auf EEPROM-Technologie basieren, verlieren sie, im Gegensatz zu manchen FPGAs, beim Wegfall der Versorgungsspannung nicht ihren Programminhalt. So bleibt das Programm auch ohne externen Programmspeicher im CPLD erhalten. Es entfällt die bei SRAM-basierten FPGAs übliche Bootphase zum Laden der Konfiguration bei jedem Neustart, sofern der Chip nicht über einen Zwischenspeicher für das Programm verfügt.

FPGAs

Die Bezeichnung „Field Programmable Gate Array" (frei programmierbare Gattermatrix) leitet sich einerseits aus dem Aufbau als Array von Gattern und zum anderen aus der Programmierbarkeit durch den Anwender her, also eine „im Anwendungs-Feld (field) programmierbare Gatter-Anordnung". Wie bei anderen PLDs ist die Funktion eines FPGAs nicht im Voraus festgelegt und somit kann der Entwickler den Baustein seinen individuellen Bedürfnissen anpassen. Hierzu besteht ein FPGA aus vielen kleinen Funktionsblöcken, in denen sich logische Funktionen realisieren lassen. Die einzelnen Blöcke werden über ein Netzwerk von Verbindungskanälen miteinander verknüpft. Dadurch entsteht dann als Endprodukt eine personalisierte digitale Schaltung. Neben rein logischen Verknüpfungen verfügen die Chips auch über reichlich Speicherelemente wie Flipflops und Register. Somit eignen sich FPGAs auch für speicherintensive Anwendungen.

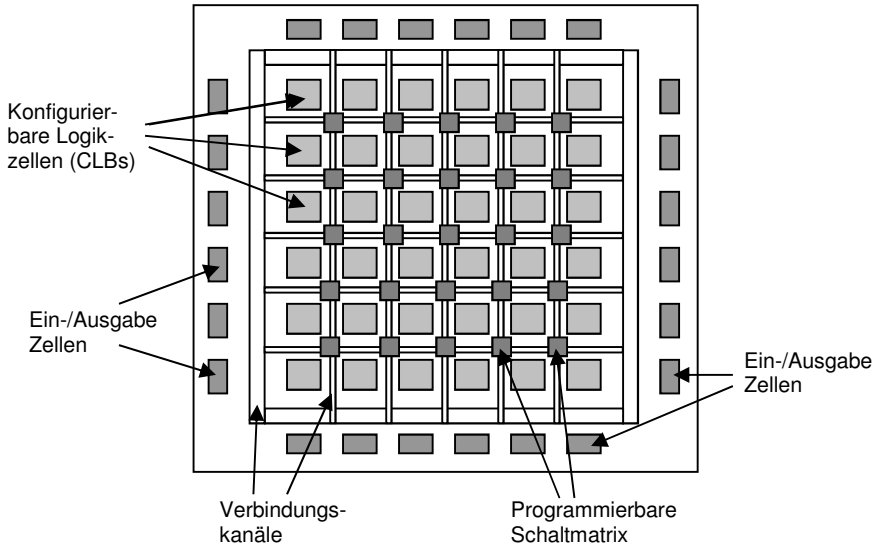

Bild 5: Vereinfachte Darstellung der Architektur eines FPGAs

Wie dem Bild 5 zu entnehmen ist, besteht ein FPGA aus einer Vielzahl vordefinierter Komponenten mit programmierbaren Verbindungsleitungen sowie Ein-/Ausgabezellen (I/O Blocks) mit Speicherelementen zur Anbindung an die Schaltungsumgebung. Damit eignet sich das FPGA insbesondere zur Entwicklung von komplexen rekonfigurierbaren digitalen Schaltungen mit Ein-/Ausgabeblöcken zur Kommunikation mit der Außenwelt. Die konfigurierbaren Logikblöcke („CLBs - Configurable Logic Blocks“) sind in eine programmierbare Verbindungsmatrix eingebettet und stellen die wesentlichen Logikkomponenten eines FPGAs dar. Sie bestehen wiederum aus zwei grundlegenden Komponenten: den Flipflops und den Look-up-Tabellen („LUT“ – look up table).

CLB und Look up table

Configurable Logic Blocks sind Funktionsblöcke in einer programmierbaren Schaltung, deren Elemente wie die Lookup-Tabellen (LUT), UND-ODER-Matrizen und Speicherelemente wie D-Flipflops oder Register sich gezielt in bestimmte Konfigurationen bringen lassen, um die beabsichtigten Verknüpfungen zu realisieren.

Eine LUT kann eine beliebige kombinatorische Funktion (NAND, XOR, Multiplexer etc.) aus den anliegenden Eingangssignalen erzeugen. Oft werden sehr einfache logische Funktionen durch eine Tabelle ersetzt. Hierzu werden die Funktionswerte wie z.B. von trigonometrischen Funktionen vorab ermittelt und dann als Tabelle gespeichert. Eine rein arithmetische Berechnung der Werte kann hingegen recht lange dauern und muss bei jedem Aufruf der Funktion wiederholt werden. Die Anzahl der Eingangssignale pro LUT ist abhängig vom FPGA und liegt meist zwischen 4 und 6. Bild 6 zeigt einen einfachen Logikblock (CLB) mit einer LUT, einem D-Flipflop und einem Multiplexer. Die getakteten Flipflops dienen als Zwischenspeicher von Signalwerten, die dann im nächsten Takt weitergeleitet werden.

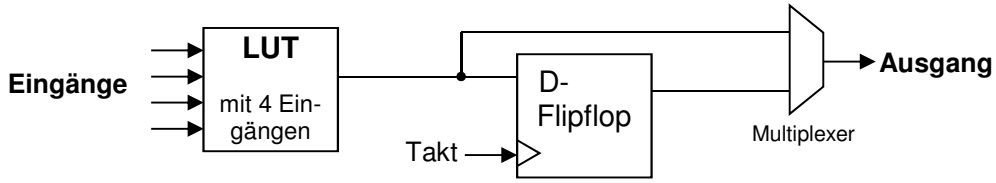

Bild 6: Einfacher Logikblock (CLB) eines FPGAs mit LUT und Flipflop

Das andere Steuerungskonzept

Die Firma ZANDER in Aachen kann bereits auf vielfältige Erfahrungen mit Steuerungen zurückblicken, die prinzipbedingt zykluszeitfrei arbeiten. All diesen High-Speed Steuerungen ist gemeinsam, dass sie auf der Basis von Programmierbaren Logikbausteinen wie CPLDs oder FPGAs anstelle von Mikroprozessoren beruhen. Bedingt durch die parallele Abarbeitung der Steuerungsprogramme arbeiten diese Steuerungen zykluszeitfrei.

Verarbeitungszeit
Diese Zykluszeit-Freiheit bedeutet zwar keineswegs, dass keine Verarbeitungszeit benötigt wird, denn auch ein CPLD oder FPGA ist mit Signallaufzeiten behaftet. Diese Laufzeiten sind jedoch sehr kurz und liegen, je nach Bausteintechnologie, in der Größenordnung von 0,01 µs. Was aber weitaus wichtiger ist: Diese Laufzeiten sind unabhängig von der Anzahl der quasi-parallel abzuarbeitenden Teilaufgaben. Dies bedeutet für den Anwender, dass die Steuerung bei zugleich hoher Geschwindigkeit auch immer garantiert deterministisch arbeitet. Schwankende Zyklus- und Reaktionszeiten sind hier ausgeschlossen. Dies ist wichtig für Anwendungen, bei denen nicht nur hohe Geschwindigkeit, sondern auch deterministische Reaktionen gefordert sind, wie es in vielen Bereichen wie z.B. bei Verpackungs- und Abfüllanlagen oder bei Anwendungen mit schnellen synchronen Antrieben vorausgesetzt wird.

Die ZX20-Familie

Auch die neue Speicherprogrammierbare Steuerung der Firma ZANDER, die zur Klasse der Mikro-SPS zählende ZX20-Familie, funktioniert auf Basis von FPGAs und arbeitet damit ohne Zykluszeiten. Sie lässt sich einfach vernetzen und wird ergänzt durch ein komplettes Entwicklungssystem auf Basis der SPS-Hochsprache „Strukturierter Text –ST“.

Bei Aufgaben mit einer hohen Ereignisdichte, also parallel anfallenden Verknüpfungen, sind die programmierbaren Logikbausteine den üblichen Mikroprozessoren weitaus überlegen. Bei der ZX20 wird der jeweilige Steuerungsalgorithmus mit Hilfe der dafür entwickelten Designsoftware ähnlich einer auf die Aufgabe zugeschnittenen Hardwareentwicklung in einen speziell für diese Applikation angefertigten Steuerungsprozessor umgesetzt. Die daraus resultierende Parallelität bei der Bearbeitung ist der eigentliche Grund für die enorme Beschleunigung gegenüber den Mikroprozessor-basierten Lösungen. Während ein Mikroprozessor durch die relativ zeitintensive sequentielle Abarbeitung einer Befehlsfolge einen Algorithmus nach dem anderen ausführt, läuft im applikationsspezifischen Prozessor alles in echter Parallelität ab.

Mikroprozessoren sind jedoch besser für das Netzwerkmanagement oder zur Speicherung größerer Datenmengen geeignet. Beides ist in vernetzten Steuerungsapplikationen notwendig, so dass die ZX20 mit einem FPGA für die schnelle binäre Steuerungsarbeit und zur Bearbeitung arithmetischer Operationen sowie einem integrierten Mikrocontroller als Kommunikationswandler für die Netzwerkkommunikation bestens gerüstet ist. Bild 7 zeigt vereinfacht die interne Struktur der ZX20.

Im Vergleich zu den bisher am Markt erhältlichen ZANDER-Steuerungen wie der EX-16 oder ZX8 konnte bei der ZX20 die Kapazität um mehr als den Faktor 50 gesteigert werden. Die interne Rechenzeit beträgt 0,1 µs, unabhängig von der Größe der Applikation. Die ZX20 ist damit die ideale Steuerung für schnelle Prozesse an Verpackungsmaschinen, Zuschneideinrichtungen, Kunststoffspritzmaschinen, Abfüllanlagen, Klebestationen etc.

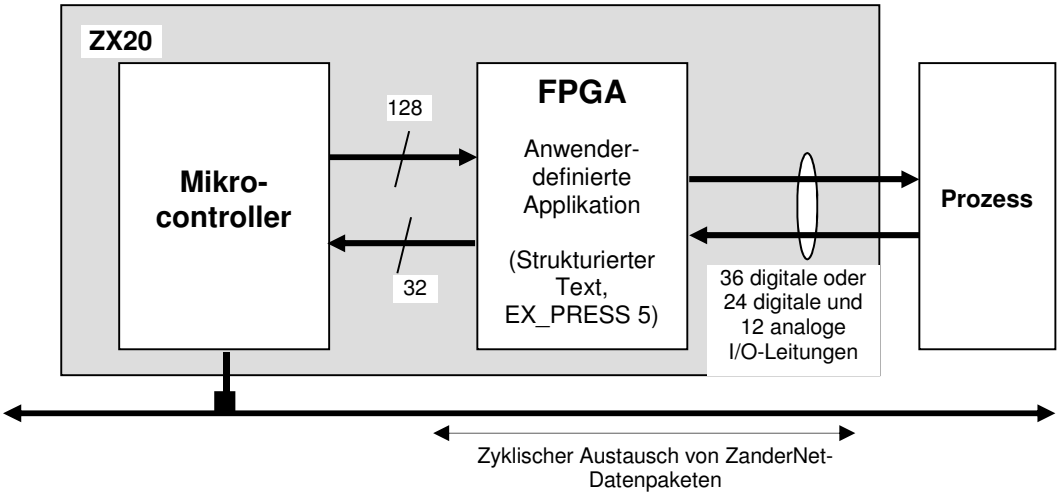

Bild 7: Vereinfachte interne Struktur der ZX20

Vernetzung
Die ZX20 ist mit einfachen Möglichkeiten der Vernetzung ausgestattet. Der integrierte Mikrocontroller übernimmt dabei in Zusammenarbeit mit einem Netzwerk-IC für Ethernet oder andere Netzwerkstandards das Management des Netzes und bildet das Interface zwischen Applikation und Netzwerk. Für einen kontinuierlichen Datenaustausch ermöglicht die ZX20 eine Vernetzung via Switched-Ethernet, um so eine verteilte Gesamtapplikation aufzubauen -siehe: interne Struktur der ZX20 wie in Bild 7 dargestellt. Zur Vernetzung kann ein bereits vorhandenes Ethernet genutzt werden, auch wenn dies bereits durch andere Kommunikationsaufgaben genutzt wird. Selbst die Programmierung der Steuerungen erfolgt über Ethernet Anschluss und ist damit auch im eingebauten Zustand möglich.

Das ZanderNet ist derzeit der zugleich einfache und effiziente Standard zur Kommunikation zwischen den vernetzten ZX20-Steuerungen. Aktuell wird die Kommunikation direkt via Ethernet und IPV4/UDP (Internet Protocol Version 4, die 4. Version des Internet Protokolls und „User Datagram Protocol“, ein minimales, verbindungsloses Netzwerkprotokoll) unterstützt. Das Protokoll dient dem Austausch von Variablen zwischen den einzelnen Steuerungen. Zusätzlich können die aktuellen Werte an den Ein- und Ausgängen per Netz ausgelesen sowie User-Definierte Nachrichten erzeugt und abgefragt werden. Weitere Protokolle sind in Vorbereitung. Besonders vorteilhaft ist jedoch die Möglichkeit, die Steuerungen auch im eingebauten Zustand zu programmieren. Das ZanderNet kann hierzu mit anderen Übertragungsstandards über das gleiche Kabel kombiniert werden.

Verteilte Applikation
Die Kommunikation innerhalb einer verteilten Applikation läuft so ab, dass jede ZX20, bei der Netz-Ausgangsvariablen definiert sind, diese zyklisch über das ZanderNet auf dem Netz verteilt. Jede ZX20, die diesen Variablen entsprechende Eingangsvariable definiert hat, filtert sich deren aktuelle Werte aus den eintreffenden Datenpaketen heraus und übergibt sie der jeweiligen Applikation. Zusätzlich wird die Verfügbarkeit des Netzwerks signalisiert. Wie Bild 7 zeigt, ist zwischen der Anwenderdefinierten Applikation, also dem Programm für die jeweilige ZX20-Steuerung, und dem globalen Netzwerk der Austausch von maximal 128 Bitvariablen auf der ZX20-Eingangsseite und 32 Bitvariablen ZX20-ausgangsseitig möglich. Hierzu deklariert der Anwender diese Variablen mit selbst gewählten symbolischen Namen, die dann der Compiler von „EX_PRESS 5“, dem Übersetzungssystem für die ZX20, auf die definierten Interfacevariablen abbildet. Während der Laufzeit filtert der Mikrocontroller wiederum „seine“ Variablen aus den zyklisch eintreffenden Meldungen aller anderen ZX20-Steuerungen heraus, einschließlich der eines PC-Programms als Master. Diese Struktur verteilter Applikationen kommt den von der IEC 61499 definierten verteilten Steuerungssystemen sehr nahe.

IEC 61499
Von der IEC (International Electrotechnical Commission, ein Normungsgremium für Elektrotechnik) wurde 2005 ein Standard für verteilte Automatisierungs- und Steuerungssysteme („Distributed Automation“) veröffentlicht, der sich der Weiterentwicklung noch offen gebliebener Punkte aus der Norm IEC 61131-3 annehmen soll. Die IEC 61499 bzw. EN 61499 definiert ein Modell für verteilte Steuerungssysteme bzw. verteilte Intelligenz. Das klassische zyklische Ausführungsmodell der IEC 61131 wird dabei durch ein ereignisorientiertes Ausführungsmodell ersetzt. Für den Anwender stellt dieser Standard die nächste Generation industrieller Steuerungen dar, denn die Architektur ermöglicht ein real verteiltes Automatisierungsdesign. Dabei kann die Intelligenz von Anfang an dezentral gestaltet und frei auf die vernetzten Geräte/Steuerungshardware verteilt werden.

Vorteile
Je mehr Intelligenz in der Anlage verteilt ist, desto modularer gestaltet sich das Gesamtkonzept und desto mehr wieder verwertbare Teileinheiten werden geschaffen. Ein modernes System der Distributed Automation zeichnet sich auch durch seine Einfachheit gegenüber einem zentralen Automatisierungsmodell aus. Das wird bewirkt durch die Aufteilung von Intelligenz, also letztlich Programmcode, auf verschiedene, räumlich verteilte Geräte und Einheiten. Dieses Prinzip wird bereits in den Geräten der ZANDER-Steuerungen bis zu einem gewissen Grad auf der Ebene verteilter intelligenter Steuerungsmodule bzw. über sehr schnelle FPGA-Lösungen realisiert.

Programmbeschreibung in Strukturiertem Text - ST

Obwohl die Baureihe ZX20 intern mit FPGA-Technologie, statt mit Mikrocontrollern, arbeitet, braucht der Anwender keine neue Programmiersprache zu erlernen, denn die Programmierung unter der Entwicklungssoftware EX_PRESS 5 erfolgt mit Strukturiertem Text (ST) nach der Norm EN 61131-3. Da jedoch das Programmieren mit ST noch nicht so allgemein verbreitet ist, wie mit Anweisungsliste oder Kontaktplan, erscheint eine knappe Einführung in diese Programmiersprache sinnvoll.

Grundlagen von ST
Die Zeiten, als man SPS-Programme ausschließlich oder überwiegend in Form relativ einfacher Anweisungslisten (AWL) oder Kontaktpläne (KOP) erstellte, sind Vergangenheit. Um einerseits den durch Komplexität und Umfang heutiger Steuerungsaufgaben gewachsenen Anforderungen zu genügen und andrerseits die kostbare Zeit zur Erstellung der Programme deutlich zu reduzieren, wird zunehmend von höheren Programmiersprachen der Steuerungstechnik Gebrauch gemacht. Dabei handelt es sich entweder um grafische Programmiersprachen wie die Ablaufsprache AS für Ablaufsteuerungen oder aber um textuelle Hochsprachen, wie die von der DIN EN 61131-3 definierte PASCAL-ähnliche Sprache “ST – Strukturierter Text”.

Es gibt gute Gründe für den Einsatz von ST, wie z. B.:

- Die einfache Portierung auf neue Programmiersysteme oder Automatisierungssysteme andere SPS-Hersteller.
- ST ist effizient und kompakt, um auch komplexe Aufgaben zu beschreiben.
- Wissen und Erfahrungen aus anderen Skriptsprachen sind intuitiv anwendbar.
- Es können meist beliebige Texteditoren verwendet werden.

Eigenschaften
Die EN 61131-3 legt neben anderen Spezifikationen auch den Sprachumfang von ST fest. Dabei lehnt sich die Syntax der Sprachelemente an die Programmiersprache PASCAL an, ist jedoch mit SPS-spezifischen Erweiterungen ausgestattet. ST bietet weit mehr Strukturierungsmöglichkeiten als AWL und löst diese daher zunehmend ab. Strukturierter

Text eignet sich besonders für Aufgaben, die sich mit mathematischen Formeln beschreiben lassen wie die Programmierung komplexer Algorithmen. Typisch für ST sind Kontrollstrukturen, die wie in Hochsprachen bedingt (IF.. THEN .. ELSE) oder als Schleifen (WHILE .. DO) ausgeführt werden. Ansonst arbeitet man in ST mit Anweisungen, Ausdrücken, Zuweisungen, selbstverständlich aber auch mit Boole'schen Operationen zur Beschreibung logischer Ausdrücke.

Aufbau von ST-Anweisungen
Eine ST-Anweisung setzt sich aus folgenden Komponenten zusammen:

- Einer (optionalen) Sprungmarke ("Label"), abgeschlossen mit einem Doppelpunkt.
- Einer Ausführungsanweisung, die beschreibt, was die CPU ausführen soll. Sie wird durch ein Semikolon abgeschlossen.
- Einem optionalen Kommentar, der mit einer öffnenden Klammer und einem Sternsymbol „(*" beginnt und wieder mit einem Sternsymbol mit schließender Klammer „*)" endet.

Innerhalb der Ausführungsanweisung muss der Operationscode von einem nachfolgenden Operanden bzw. einer Adresse durch mindestens ein Leerzeichen oder den Tabulator getrennt sein. Einige Beispiele sollen dies etwas näher veranschaulichen:

```
        Volumen   := 4/3 * PI * Radius**3;      (* Berechnung eines Volumens *)
        Leistung  := Strom * Spannung;          (* Elektrische Leistung berechnen *)
        IF  E3.7 = TRUE THEN GOTO M01;          (* Programmverzweigung *)
        …….
        …….
M01:  END_FUNCTION;                             (* Sprungmarke, Bausteinende *)
```

Entwicklungssoftware EX_PRESS 5

Die Entwicklungssoftware EX_PRESS 5 wurde speziell für die ZX20-Familie erstellt. Mit dieser Software erstellen und verwalten Sie Ihre Projekte in der Sprache Strukturierter Text und programmieren die ZX20-Steuerungen direkt im Netz über die integrierte Ethernetschnittstelle. Zu einem Projekt gehören dabei eine oder mehrere in ST verfasste Steuerungsbeschreibungen, die dann einzeln für jede Steuerung übersetzt und in diese per Ethernet übertragen werden. Im Vergleich zu den Vorgängern, den Zander-Kleinsteuerungen EX16 und ZX8, weist diese Entwicklungssoftware viele Neuerungen auf. Nunmehr lassen sich auch Integer-Variablen (INT) mit 8 und 16 Bit Datenbreite deklarieren und verarbeiten. Logische und arithmetische Operationen, Verzweigungen von nahezu beliebiger Komplexität sowie Funktionen sind weitere Bestandteile der Entwicklungssoftware. Im Verbund mit fast beliebig vielen applikations-spezifischen Timern sowie einer Entprellung von Eingängen durch einfache Softwarebefehle (bei Verzicht auf Entprellung der Hardware) wurde hiermit eine komfortable Entwicklungsumgebung geschaffen. Da die Version ZX20AT 8 analoge Eingänge und 4 analoge Ausgänge aufweist, wurde auch die direkte Integration von AD-Eingängen und DA-Ausgängen in die Software aufgenommen. Ansonst funktioniert alles wie bei einer herkömmlichen SPS. Programme können beliebig oft gelöscht und überschrieben werden. Somit lässt sich mit EX_PRESS 5 schnell eine individuelle Lösung für die jeweilige Steuerungsaufgabe erstellen. Bild 8 zeigt Beispiele für die Syntax der Variablendeklaration.

```
EXTERN VAR_INPUT
        Q0_XT; Q1_XT; Q2_XT; Q3_XT;
END_VAR;                        (* 4 Eingänge aus dem Netz, maximal 128 *)

VAR_ADC
        AD_0  AT AIn_01;  XYZ AT Ain_04;
END_VAR;

VAR_DAC
        DA_0  AT AOut_01;  DAC_OUT  AT AOut_03;
END_VAR;                        (* je 2AD- und DA-Anschlüsse,
                                   mit Definition der Anschlussklemme *)
```

Bild 8: Definition von Netz-Eingangsvariablen, A/D- und D/A-Anschlüssen in ST

Windows-Oberfläche

EX_PRESS 5 bietet eine komfortable Windows-Oberfläche mit Projektverwaltung und Netzwerkunterstützung, siehe Bild 9. Dadurch lassen sich auch komplex vernetzte Systeme übersichtlich verwalten. Als Programmierschnittstelle dient der Ethernet-Anschluss. Mit EX_PRESS 5 ist auch eine Programmierung der Steuerungen direkt im Netzwerk möglich!

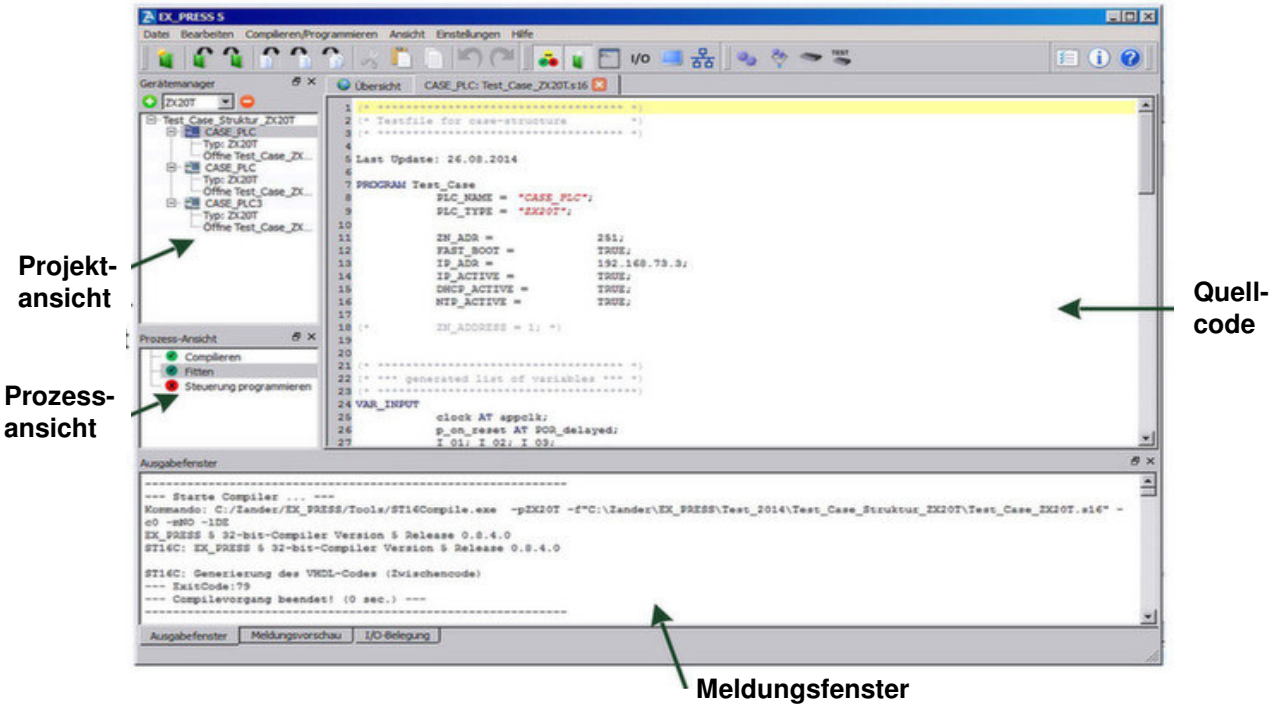

Bild 9: Die Entwicklungssoftware EX_PRESS 5 unterstützt den ZX20-Anwender

Globale Variable
Im Unterschied zur Definition interner Variabler wird für Werte von Eingangs-Variablen, die von anderen ZX-Steuerungen kommend per Netz übermittelt werden, das Schlüsselwort „EXTERN“ verwendet. Der Compiler sorgt dafür, dass diese Variablen keinen physikalischen Anschluss bekommen, sondern, wie beschrieben, aus dem Netzwerk stammen. Ansonsten können diese Variablen in der Applikation genau so verwendet werden wie alle anderen auch, jedoch mit einer Ausnahme: Ihr zeitliches Verhalten ist von der lokalen Applikation unabhängig, da sich ihre Quelle an einem anderen Ort der verteilten Applikation befindet und sich durch das Netzwerk eine gewisse zeitliche Verzögerung ergibt.

Programmiersprache ST-16

Die in EX_PRESS 5 benutzte Programmiersprache ST-16 stellt eine Untermenge der in DIN EN 61131-3 definierten Sprache des Strukturierten Textes dar. ST-16 wurde speziell definiert und zugeschnitten auf die Erfordernisse einer Kompilierung in FPGA-konforme Darstellungen.

Datentypen
Zur Deklaration von Variablen ohne Peripheriebezug sind in ST16 folgende Datentypen zugelassen:

- BIT (Defaultwert) und BOOL für eine Variable mit den zulässigen Werten ‘0‘ und ‘1‘
- INT für eine Variable mit dem Wertebereich -32768 bis +32767 (16 bit Integer)
- UINT und WORD für eine Variable mit dem Wertebereich 0 bis +65535 (16 bit Unsigned Integer)
- SINT für eine Variable mit dem Wertebereich -128 bis +127 (8 bit Short Integer)
- USINT und BYTE für eine Variable mit dem Wertebereich 0 bis +255 (8 bit Unsigned Short Integer)
- BIT_ARRAY(xx) für eine Variable als Bit-Array mit xx Bits (xx zwischen 1 und 16)
- BIT_ALIAS für eine Variable, in der mehrere bereits deklarierte Variablen vom Typ BIT bzw. BOOL zusammengefasst sind.

In Tabelle 1 finden Sie eine Zusammenstellung der in ST-16 genutzten Datentypen mit ihren wesentlichen Eigenschaften. Nahezu all diese Datentypen sind auch in der Norm DIN EN 61131-3 definiert.

Datentyp	Daten-breite	Lesender Zugriff	Schreibender Zugriff	Automatische Konvertierung in:
BIT	1	BIT	BIT	BOOL
BIT_ARRAY	1 .. 16	BIT_ARRAY	BIT_ARRAY	INT, UINT, SINT, USINT, BYTE, WORD
BIT_ALIAS	1 .. 16	BIT_ARRAY	BIT_ARRAY	INT, UINT, SINT, USINT, BYTE, WORD
INT	16	INT	INT	BIT_ARRAY, UINT, SINT, USINT, BYTE, WORD
UINT	16	UINT	UINT	BIT_ARRAY, INT, SINT, USINT, BYTE, WORD

SINT	8	SINT	SINT	BIT_ARRAY, INT, UINT, USINT, BYTE, WORD
USINT	8	USINT	USINT	BIT_ARRAY, INT, UINT, SINT, BYTE, WORD
BYTE	8	USINT	USINT	BIT_ARRAY, INT, UINT, SINT, UINT, WORD
WORD	16	UINT	UINT	BIT_ARRAY, INT, UINT, SINT, UINT, BYTE
TIMER	1	BIT	-	BIT, BOOL
COUNTER	1	BIT	-	BIT, BOOL
ADC	1 .. 16	BIT_ARRAY	-	INT, UINT, SINT, USINT, BYTE, WORD
DAC	1 .. 16		BIT_ARRAY	-

Tabelle 1: Übersicht der von ST-16 genutzten Datentypen

Zuweisungen
Zuweisungen oder „Assignments“ bewirken, dass ein Wert einer Zielvariablen zugewiesen wird. Dies bedeutet, dass eine Zielvariable, die natürlich deklariert sein muss, mit der Zuweisung auf einen neuen Wert gesetzt wird. Der zugewiesene Wert kann ein Ausdruck oder eine Konstante sein.

Operationen
Ein Ausdruck besteht aus einer Konstanten, einer Variablen oder aus einer Verknüpfung zwischen diesen Größen. Die Zuweisung hierzu besitzt die allgemeine Form:

<variable> := <constant> | <variable> | <expression>;

Einen Ausdruck <expression> wiederum erhält man durch die Verknüpfung eines oder zweier Operanden mithilfe einer Operation. In ST-16 sind, nach Prioritäten geordnet, die in Tabelle 2 aufgeführten Operationen zulässig:

Operator, Operation	Bedeutung
OR	logisches ODER
XOR	Exklusiv-ODER
AND	logisches UND
=, <>	Vergleiche auf Gleichheit und Ungleichheit
<, >, <=, >=	Vergleiche auf kleiner, größer, kleiner gleich, größer gleich
+, -	Addition, Subtraktion
*	Multiplikation
NOT	logische Invertierung, Komplement
<fkt_name>( )	Funktionsaufruf
(,)	Klammersetzung

Tabelle 2: Operatoren von ST-16, sortiert nach aufsteigender Priorität

Hinweis:
Beachten Sie bitte, dass im Hinblick auf die parallelen Verknüpfungen im Zielbaustein FPGA alle Zuweisungen innerhalb des ST-16 Hauptprogramms (zwischen „PROGRAM“ .. und „END_ PROGRAM“) zueinander parallel verlaufen. Dies bedeutet zugleich, dass einer Variablen, auf der linken Seite der Zuweisung stehend, nur an einer Stelle im Hauptprogramm ein Wert zugewiesen werden darf. Man spricht hier von einem „Single-Assignment-Konzept“, wenn eine direkte Zuweisung nur an einer Stelle erlaubt ist.

Kontrollstrukturen
Ein Steuerungsalgorithmus wird im Allgemeinen auch Verzweigungen, bedingte Anweisungen, Fallunterscheidungen usw. enthalten. Damit lassen sich Programmabschnitte nur unter bestimmten Bedingungen ausführen bzw. überspringen. Eine Verzweigung legt fest, welcher von zwei oder mehreren Programmabschnitten, abhängig von einer oder mehreren Bedingungen, bearbeitet wird. Derartige bedingte Anweisungen und Verzweigungen bilden, zusammen mit Schleifen, die Kontrollstrukturen der Programmiersprachen, die zu den wichtigsten Bestandteilen der Programmierung gehören. Für diesen Zweck stehen in ST16 zwei Auswahlanweisungen zur Verfügung: Die IF-Anweisung und die CASE-Anweisung. Diese Anweisungen besitzen folgende allgemeine Struktur:

1. **IF – THEN – ELSE Konstrukt**
   Die wohl häufigste und wichtigste Kontrollstruktur ist die „Wenn-Dann-Sonst-“ oder IF-THEN-ELSE-Anweisung. Die Bedingung (condition) muss ein Boolescher Ausdruck, also wahr oder falsch sein.

```
IF <condition> THEN
    <expression>; {<expression2>;}
{ELSIF <condition2> THEN
    <expression>; {<expression2>;}}
[ELSE <expression>; {<expression2>;}]
END_IF;
```

2. **CASE - OF**
   Die Struktur dieser Fallabfrage lautet:

```
CASE <var_name> OF
    <value_1>:
    {<value_k>:}
        <expression>; {<expression2>;}
    {<value_2>:
    {<value_m>:
        <expression>; {<expression2>;}}
[ELSE
    <expression>; {<expression2>;}]
END_CASE;
```

Beide Programmkonstrukte sind für Fallunterscheidungen bzw. Verzweigungen gleichermaßen geeignet, so dass es dem Anwender überlassen wird, welche er benutzen möchte. Im Allgemeinen wird die CASE-Funktion für Vielfachverzweigungen gewählt, während wenige Bedingungen eher mit der IF - THEN ... Verzweigung codiert werden. IF- und CASE-Anweisungen können in ST16 beliebig bis zu einer Tiefe von 10 Ebenen miteinander verschachtelt werden.

Programm-Generierung

Die einzelnen Schritte von der Erzeugung des ST-Quellcodes durch den Anwender bis zum fertigen Programm der ZX20-Steuerungen werden im Bild 10 schematisch aufgezeigt. Ziel dieser Software-Kette ist die Generierung einer für programmierbare logische Schaltungen geeigneten Programmdatei im VHDL-Code, aus der eine Netzliste erstellt wird. Die Netzliste

wird nach Konvertierung in einen geeigneten „Bitstream“ schließlich direkt in den für die Steuerung verwendeten FPGA-Baustein geladen. Auf diese Weise entsteht aus dem sequentiell orientierten ST-Quellprogramm eine funktionsgleiche, jedoch absolut parallel arbeitende Verknüpfungslogik.

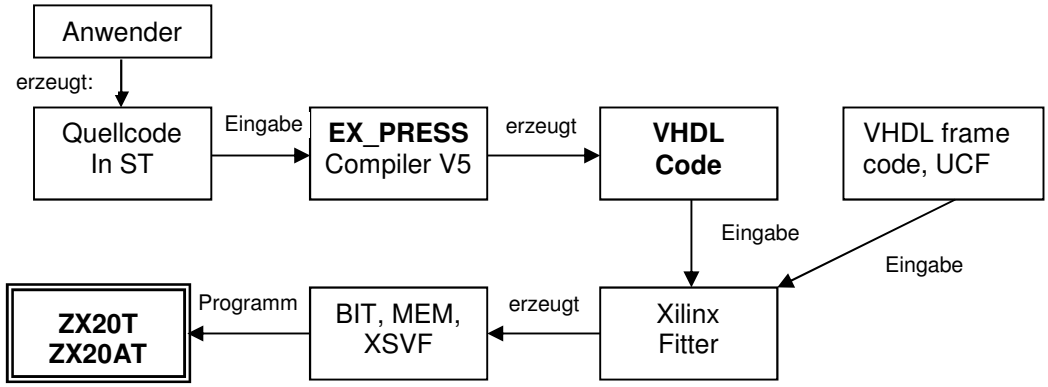

Bild 10: Die Software-Kette zur Programm-Generierung

Nachfolgende Tabelle 3 erläutert kurz einige in Bild 8 verwendete Bezeichnungen bzw. Begriffe.

Kürzel	Bezeichnung	Erläuterung
UCF	User Constraint File	Mit Hilfe von .ucf-Dateien wird die Verbindung von logischen Netzen (in- und out-Ports) mit den physikalischen Pins des (FPGA-) Bausteins hergestellt.
SVF	Serial Vector Format	Das Serial Vector Format, kurz „SVF“, ist ein Dateiformat zum Austausch von so genannten Boundary Scan-Testvektoren. Das SVF-Format ist als Programmierdatei für programmierbare logische Schaltungen üblich.
XSVF	Variante von SVF	Eine Variante ist das XSVF-Format des Unternehmens Xilinx. Im Gegensatz zum SVF ist XSVF keine ASCII- sondern eine Binärdatei und stellt eine Art komprimierter Form von SVF-Dateien dar.
BIT, MEM	Dateiformate	Spezielle zur Programmierung von FPGAs eingesetzte Dateiformate („Bitstream“ zum Laden der Konfigurationsdaten in ein FPGA)

Tabelle 3: Erläuterungen zu Bild 8

VHDL
VHDL steht für „Very High Speed Integrated Circuit Hardware Description Language“ und ist eine Hardwarebeschreibungssprache, die es ermöglicht, komplexe digitale Systeme textbasiert zu beschreiben. Dabei arbeitet man nicht mit den einzelnen elektronischen Bauteilen wie Gatter oder Register, sondern beschreibt die gesamte Schaltung hinsichtlich ihres gewünschten Verhaltens auf einer höheren Abstraktionsebene. Somit ermöglicht VHDL das schnelle Entwickeln großer und komplexer Schaltungen.

Die ZX20-Steuerungen

Abschließend soll die Hardware der ZX20-Steuerung etwas näher vorgestellt werden. Für die ZX20 – Baureihe sind derzeit zwei Varianten verfügbar. Die ZX20T ermöglicht die Verarbeitung von rein digitalen/binären Signalen, während die ZX20AT eine gemischte

Verarbeitung von analogen und digitalen Signalen gestattet. Bild 11 zeigt eine Ansicht der ZX20T mit 20 digitalen Eingängen und 16 digitalen Ausgängen.

Analogsignale
Während die Version ZX20T mit insgesamt 20 digitalen Ein- und 16 digitalen Ausgängen aufwartet, bietet die Variante ZX20AT nur je 12 digitale Anschlüsse, dafür jedoch 8 analoge Ein- sowie 4 analoge Ausgänge. Die Auflösung dieser Ein-/Ausgänge beträgt jeweils 16 Bit, wobei in den Applikationen auch geringere Bitbreiten verrechnet werden können. Zeitlich gesehen kann die Wandlung mit 200 ksps (ksps: Kilosample(s) pro Sekunde, also 200 000 Wandlungen pro Sekunde) ablaufen. Bei der Nutzung von 8 eingangsseitigen Kanälen beträgt die resultierende maximale Zykluszeit für eine Wandlung also 40 µs, bei 5 µs je belegtem Eingang/ Ausgang. Durch die Verarbeitung von Analogwerten erweitert sich das Spektrum der realisierbaren Applikationen um Aufgaben wie Temperaturerfassung und -überwachung, Druckmessung, Anfahr- und Abbremsrampen für Antriebe, Drehzahlsteuerung und –regelung von Motoren usw. Die Einbindung in die Software ist denkbar einfach: Der entsprechende Eingang wird zunächst wie in Bild 8 gezeigt definiert. Dann kann der Wert jeweils als Integer-Variable in der Applikation verarbeitet werden. Zur Ausgabe erfolgt dementsprechend eine Wertzuweisung an die deklarierte DAC-Variable. Den Rest erledigt der Compiler von EX_PRESS 5.

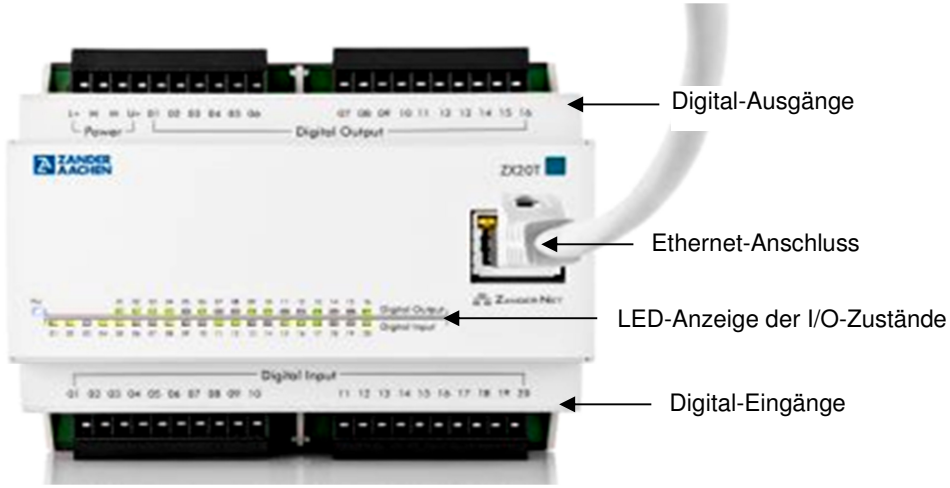

Bild 11: Ansicht der Steuerung ZX20T

Einsatzgebiete
Typische Einsatzgebiete für diese Kleinsteuerungen finden sich bei Verpackungsmaschinen, Zuschneideinrichtungen, Abfüllanlagen, Klebestationen, Kunststoff-Spritzgussmaschinen, Transportanlagen, Nockenschaltwerke etc. In Tabelle 4 finden Sie eine Zusammenstellung der wichtigsten technischen Daten für die beiden Steuerungen ZX20T und ZX20AT.

Technische Daten	ZX20T	ZX20AT
Digitale Eingänge	20 x DC 18..30 V	12 x DC 18..30 V
Digitale Ausgänge	16 x Transistor DC 10..30 V; 0,5 A	12 x Transistor DC 10..30 V; 0,5 A
Analoge Eingänge	-	8 x 16 Bit; 0..10 V
Analoge Ausgänge	-	4 x 16 Bit; 0..10 V
Verarbeitungszeit	< 20 ns	< 20 ns
Reaktionszeit	< 9 µs bei 250 mA/Ausgang	< 9 µs bei 250 mA/Ausgang
Max. Takteingangsfrequenz	500 kHz	500 kHz
Anzahl programmierbarer Zeiten	2000	2000
Programmierung	EX_PRESS 5 für Windows; Strukturierter Text (ST) nach IEC 61131-3	EX_PRESS 5 für Windows; Strukturierter Text (ST) nach IEC 61131-3
Anschlüsse	Programmierschnittstelle, Erweiterungsstecker, Ethernet	Programmierschnittstelle, Erweiterungsstecker, Ethernet

Tabelle 4: Wichtige technischen Daten der Steuerungen ZX20T und ZX20AT

Weitere Informationen zu den Steuerungen ohne Zykluszeit und zu weiteren Produkten für die Automatisierungstechnik der Firma Zander finden Sie unter der Homepage

http://www.zander-aachen.de/

oder besuchen Sie doch die Firma Zander auf der nächsten „sps ipc drives“- Messe in Nürnberg.