IEC 61499 FUNCTION BLOCKS FÜR DEN ENTWURF VON EINGEBETTETEN UND VERTEILTEN SYSTEMEN

Dritte Ausgabe

Valeriy Vyatkin

Luleå Tekniska Universitet, Sweden

Übersetzung: Ewald Matull

Hochschule Emden/Leer,

Deutschland

67 Alexander Drive

P.O. Box 12277

Research Triangle Park, NC 27709

Printed in the United States of America.

10 9 8 7 6 5 4 3 2

ISBN: 978-0-876640-44-9

Hinweis

Die Informationen, die in dieser Veröffentlich präsentiert werden, dienen der allgemeinen Unterweisung des Lesers. Da weder der Autor noch der Verlag einen Einfluss auf die Benutzung der Informationen durch den Leser haben, lehnen sowohl der Autor als auch der Verlag jede Haftung irgendeiner Art ab, die sich aus der Benutzung dieses Werkes ergeben sollte. Vom Leser wird eine gründliche professionelle Beurteilung erwartet hinsichtlich der Nutzung aller Informationen in einer spezifischen Applikation. Weiterhin haben weder der Autor noch der Verlag untersucht oder betrachtet, ob die Benutzung der im Werk enthaltenen Informationen in einer speziellen Anwendung durch den Leser irgendein Patent verletzt. Der Leser ist selbst dafür verantwortlich, alle denkbaren Patente daraufhin zu überprüfen, ob sie eine spezielle Nutzung der angebotenen Information beeinträchtigen.

Sämtliche Hinweise auf kommerzielle Produkte in diesem Werk sind lediglich als Beispiele zitiert. Weder der Autor noch der Verlag unterstützen eines der erwähnten kommerziellen Produkte. Sämtliche referenzierten Markenzeichen und Markennamen gehören dem jeweiligen Eigentümer des Markenzeichen oder Markennamens. Weder der Autor noch der Verlag übernehmen zu irgendeiner Zeit irgendeine Verantwortung hinsichtlich der Verfügbarkeit eines im Werk referenzierten kommerziellen Produkts. Die Anweisungen des jeweiligen Herstellers hinsichtlich der Benutzung irgendeines kommerziellen Produkts müssen zu jeder Zeit befolgt werden, auch wenn sie im Konflikt mit den Inhalten dieses Werks stehen.

1 Inhalt

Einführung zur dritten (deutschen) Ausgabe

In der dritten Ausgabe dieses Buches wurde das Tutorial-Material neu implementiert mit dem kommerziellen Entwicklungswerkzeug nxtStudio der Firma nxtControl GmbH (Österreich). Dieses Tool unterstützt einen breiten Bereich von Hardwareplattformen und Feldbussen und zeigt, dass die IEC 61499 eine ernstzunehmende Technologie im Industriebereich darstellt. Der Übergang von FBDK zu nxtStudio hat zu einem bestimmten Wechsel in der Materialauswahl geführt. So präsentiert Kapitel 12 nun nxtStudio-spezifische Wege zur Adaption der Hardware-Peripherie. Kapitel 14 zeigt vollständig neues Material über die Implementation von Mensch-Maschine-Schnittstellen und Kapitel 16 bietet eine neu entwickelte Fallstudie und ein Tutorial, das die Entwicklung eines Pick-and-Place-Roboters behandelt.

Wie sollte man dieses Buch benutzen?

Dieses Buch ist gedacht zur Benutzung parallel zur Vorlesung, am Arbeitsplatz oder zum Selbststudium und soll einen systematischen Lernprozess unterstützen.

Der Lernablauf wurde in Industrie-Schulungen sowie in Vorlesungskursen an der University of Auckland und an der Martin Luther-Universität in Halle–Wittenberg erprobt.

In der Erarbeitung von zahlreichen Beispielen mit dem Softwaretool FBDK sollen die Studierenden Verständnis durch praktisches Handeln aufbauen. Das Buch möchte dieses Toolkit nicht besonders hervorheben gegenüber anderen, die ebenfalls mit dem 61499-Standard konform sind. Dennoch zeigt die Erfahrung, dass der Lernerfolg steigt, wenn die Studierenden mit dem FBDK-Softwaretool Beispiele erproben und modifizieren können.

Die Tutorial-Anleitung im Kapitel 7 und Beispiele in den Kapiteln 8 und 11 geben dem Leser eine Schritt-für-Schritt-Einführung in die Welt der Function Blocks und Systeme.

Das Buch ist so aufgebaut, dass jedes Kapitel eine abgeschlossene Lerneinheit bildet.

Kapitel 1 gibt eine schnelle Einführung in das Function Block-Konzept anhand des einfachen Blinkgeber-Beispiels FLASHER.

Kapitel 2 betrachtet, wie ein solcher Blinkgeber in den verschiedenen Generationen der Automatisierungstechnologie realisiert worden wäre, beginnend mit festverdrahteten Schützen der Vergangenheit bis zu heutigen verteilten Embedded Systems. Diese Kapitel betrachtet die Idee der Function Blocks aus der historischen Sicht der Industrieautomation.

Kapitel 3 führt diese Diskussion fort mit Blick auf die Herausforderungen heutiger Produktionssysteme, die neue Ansprüche an die Entwicklung automatisierter System stellen.

Kapitel 4 zeigt neue technische Möglichkeiten auf, die durch die Einführung der 61499 in den Kontext der aufkommenden intelligenten Automatisierungssysteme entstehen. Kapitel 3 und 4 wurden aufgenommen für Spezialisten, die in einem breiteren Rahmen an Aspekten der Industrieautomation interessiert sind. Diese Kapitel können überflogen oder in mehr technisch orientierten Kursen sogar ausgelassen werden.

Kapitel 5 führt in die Grundlagen des neuen Function Block-Konzepts ein. Die meisten der nachfolgenden Kapitel entfernen sich von den Beispielen dieses Kapitels und beziehen sich auf die Benutzung des Software-Tools nxtStudio.

Kapitel 6 bietet einen Crash-Kurs für die nxtStudio-Entwicklungsumgebung. Die Informationen in diesem Kapitel helfen dem Leser, die Grundlagen der nxtStudio-Architektur zu verstehen Das vorliegende Buch ist nicht als Ersatz für die nxtStudio-Dokumentation gedacht, sondern als ergänzender Lernführer.

Kapitel 7 erklärt das Konzept des *Basic Function Blocks*.

Kapitel 8 führt das Konzept des *Composite* Function Block ein.

Kapitel 9 zeigt, wie man Applikationen aus Function Blocks aufbaut. (In Verbindung mit den Kapiteln 7 und 8 liefert das Kapitel 9 ausreichendes Material für das Verständnis der abstrakten Definition des gewünschten Systemverhaltens bei FBs.)

Kapitel 10 diskutiert ein weiteres wesentliches Konzept verteilter automatisierter Systeme – das Modell des Device. Nach diesem Kapitel sollte der Leser den inneren Aufbau eines Automatisierungssystems verstehen können.

Kapitel 11 führt in das Konfigurationsmodell eines verteilten Systems ein. Das FLASHER-Beispiel wird hier überarbeitet und illustriert die Prinzipien der verteilten Automation.

Auf dieser Stufe möchte der Leser evtl. mehr über Aspekte der unteren Systemebenen lernen, wie z.B. die Verbindung zu eingebetteter Hardware oder zu einer Visualisierung.

Kapitel 12 beschreibt daher die *Service Interface* Function Blocks formal (obwohl diese bereits in den Beispielen der vorhergehenden Kapitel benutzt wurden). Insbesondere zeigt es, wie man Eingangs- und Ausgangssignale von Steuerungen ankoppelt und über Netzwerke mittels Service Interface Function Blocks kommuniziert.

Kapitel 13 gibt ein Beispiel eines vollständigen, verteilten Steuerungsentwurfs für ein einfaches Mechatronik-System.

Kapitel 14 stellt zusammenfassend den Entwurf eines Human-Machine-Interfaces mittels Function Blocks vor.

Kapitel 15 führt in die Methodik des objektorientierte Entwurfs für Automatisierungssysteme ein, basierend auf dem Model/View/Control-Entwurfsmuster (MVC). Dies wird anhand des gleichen Beispiels eines Mechatroniksystems wie in Kapitel 13 gezeigt.

Kapitel 16 bietet eine erweiterte Fallstudie und ein Tutorial, das den objektorientierten Entwurf anhand des Beispiels eines Mechatroniksystems erläutert.

Kapitel 17 zeigt, wie die Einführung der IEC 61499 die Geschäftsmodelle von Unternehmen aus der Industrieautomation positiv beeinflussen können.

Kapitel 18 enthält einen Überblick über andere verfügbare Tools, die mit der IEC 61499 konform sind. Insbesondere behandelt es detailliert die integrierten Entwicklungsumgebungen ISaGRAF und FBDK und gibt einen knappen Überblick über die 4DIAC-Entwicklungsumgebung.

Kapitel 19 beschließt das Buch.

Anhang A zeigt die Details der standardisierten Event-Verarbeitung bei Function Bocks.

Anhang B weist wichtige Auszüge aus dem Compliance-Profil des Standards auf.

Anhang C listet die in der zweiten Fassung der IEC 61499 zu erwartenden Add-Ons und Korrekturen auf.

Website zu diesem Buch

Wenn Sie an weiteren Informationen zu diesem Buch interessiert sind, folgen Sie bitte dem Link *www.fb61499.com*. Diese Webseite enthält weiteres Lehrmaterial, Textkorrekturen zu allen mitgeteilten Fehlern sowie ein Forum zur Klärung weiterer Fragen.

1

Quick Start

Dieses Kapitel ist ein Einstieg in die grundlegenden Konzepte der Systementwicklung mit Function Blocks nach IEC 61499. Ein Beispiel eines verteilten Automatisierungssystems (FLASHER) wird benutzt, um die Schlüsseleigenschaften des neuen Standards zu beleuchten. Besondere Vorkenntnisse sind zum Verständnis dieses Kapitels nicht erforderlich.

Modellierung eines Systems mit Function Blocks

Dieses Kapitel zeigt einige Konzepte der IEC 61499 Baustein-Architektur am Beispiel eines Systems, das mit der IEC 61499 spezifiziert, modelliert und implementiert wurde. Dieses Beispiel wird in den nachfolgenden Kapiteln ausgebaut. Obwohl das System auf den ersten Blick sehr simpel wirkt, enthält es genügend Details und kann dem Leser mit seinem einfachen Aufbau einige wichtige Funktionen verdeutlichen.

Das System besteht aus vier Lampen, die in einer gewünschten Betriebsart blinken, wie in Abbildung 1-1 dargestellt. Die Betriebsart ist einstellbar und ermöglicht Befehle wie „alle Lampen blinken zusammen“, „Lauflicht Links“ oder „Lauflicht Rechts“, und so weiter. Die Lampen werden „gesteuert“ von einem „Black Box“-Controller mit vier Ausgangssignalen, die jeweils mit einer Lampe verbunden sind und diese ein-und ausschalten.

Wir stellen uns nun ein Function Block-Modell vor, das dieses System mit zugehöriger automatischer Steuerung simuliert. Später werden wir sehen, dass das gleiche Modell mit nur kleinen Änderungen benutzt werden kann, um das physische Gerät zu steuern.

Schritt 1 Öffnen Sie das “Solution File” FLASHER.sln im nxtStudio wie in Abbildung 1-2[1] gezeigt. Dafür wählen Sie den Menübefehl File->Open->Project/Solution. Als nächstes klicken Sie das APP1-Symbol unter dem System-Symbol im Navigationsbaum (obere linke Seite des Fensters). Sie sehen ein Fenster ähnlich Abbildung 1-2. Das Editor-Fenster ist in drei Zonen aufgeteilt:

- Ein *Navigationsbaum* links zeigt die Systemstruktur
- Der *Arbeitsbereich* rechts (der größte Bereich) zeigt den „Inhalt” des gewählten Systemelements. Daher zeigt er in Abbildung 1-2 den Inhalt der Ressource[2] RES1 im Device FLASHER.
- Der *Servicebereich* unten enthält verschiedene Registerkarten mit Serviceinformationen wie z.B. Fehlermeldungen, Warnungen usw.

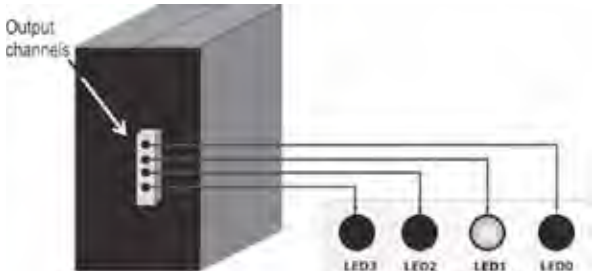


Abbildung 1-1: Das FLASHER-System, durch eine “Black Box” gesteuert.

Schritt 2 Um das System zu starten, muss man zunächst eine Steuerung erzeugen. Da wir keine Hardware zur Verfügung haben, erschaffen wir ein „Soft-Device“. Dies geschieht durch Start der Windows-Anwendung „nxtRT61499F-1“, die zusammen mit nxtStudio installiert wurde. Der Start erfolgt durch Doppel-Klick des Icons im nxtControl-Verzeichnis. Anschließend muss das FLASHER-System ins Device geladen werden („Deploy“). Dazu die Registerkarte „Devices” (links oben, siehe Pfeil) klicken und System.Dev0 auswählen, dann „Deploy” wählen.

- Wenn die Applikation heruntergeladen ist, wird das Visualisierungsfenster geöffnet. Dies geschieht durch das Rechtsklicken der “Canvas” (Leinwand) “1280 X 980” im Navigationsbaum und Auswahl des Punktes „Test HMI Runtime on the local computer”.
- Anschließend erscheint das HMI-Fenster (HMI=Human Machine Interface) wie in Abbildung 1-3.

Nun können Sie mit dem Modell “spielen”, indem Sie die Lichter mit den Radio Buttons starten oder stoppen, die verschiedenen Betriebsmodi im Pull-Down-Menü wählen (z.B. „FLASH_ALL“) und anschließend den Startknopf betätigen.

Folgendes leistet diese Systemkonfiguration: Blinken der vier Lampen nach einem aus fünf Mustern (alle blinken, aufwärts zählen, abwärts zählen, Lauflicht links, Lauflicht rechts) und mit variabler Frequenz, die durch den Drehknopf bestimmt wird. Das Blinkintervall ist auf 200 ms voreingestellt und kann bis auf 1000 ms erhöht werden.

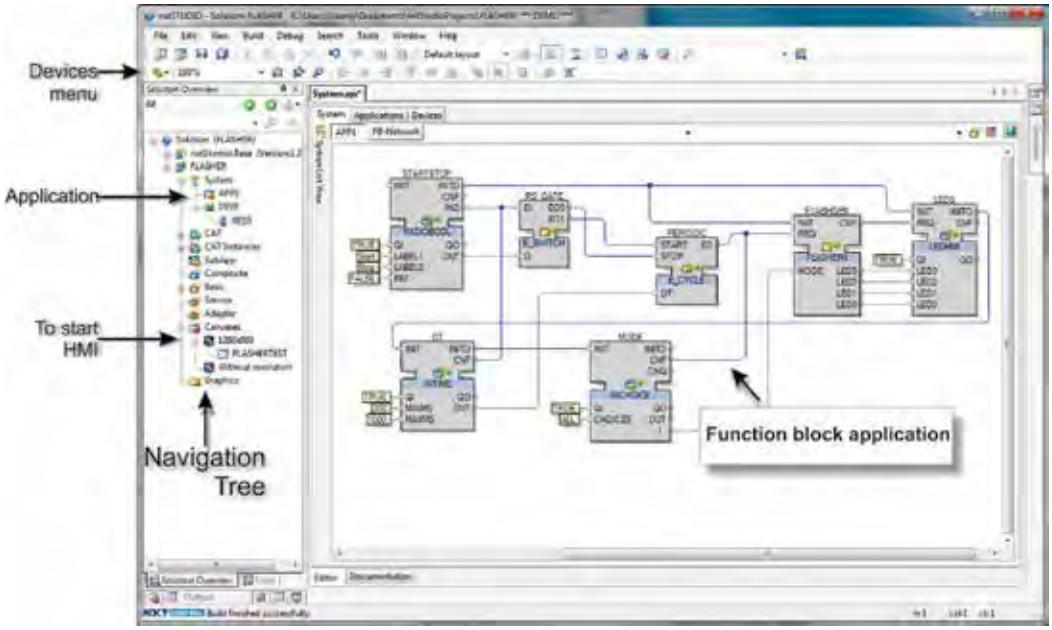


Abbildung 1-2 FLASHER TEST-Systemkonfiguration im Function Block-Editor geöffnet.

Lassen Sie uns den inneren Aufbau des Systems betrachten. Eine Systemkonfiguration ist mehr als nur rein Programm. Sie beschreibt ein *verteiltes System*, das aus mehreren kommunizierenden *Devices* und *Ressourcen* zusammengesetzt ist. In diesem Fall besteht das System aus einem Device (FLASHER), das genau eine Ressource besitzt (RES1).

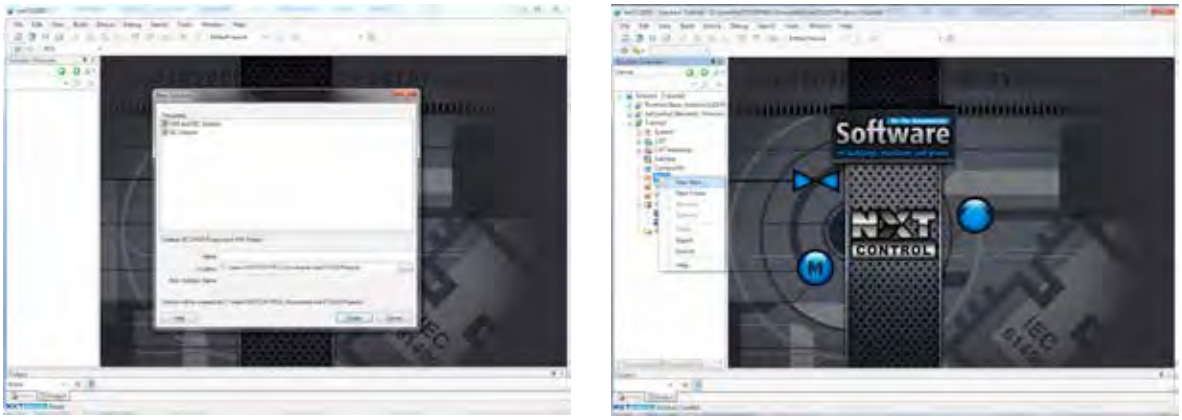
Die Ressource weist ein Netzwerk aus *Function Blocks* auf: Software-Komponenten, die sich gegenseitig aufrufen und einander Daten übermitteln. In unserem Fall implementiert das FB-Netzwerk das Modell des FLASHER-Systems. Im Kontext der IEC 61499 wird ein solches Netz eine *Applikation* genannt.

Tutorial 1: Einen neuen Basic Function Block-Typ erzeugen

Ziel diese Tutorials ist es, praktische Erfahrungen mit der Entwicklung von Function Blocks in nxtStudio zu sammeln.

In nxtStudio kann man einen Function Block-Typ nicht losgelöst erzeugen, sondern nur innerhalb einer "Solution", also einer Struktur, die analog zu einem Projekt in anderen Programmierumgebungen zu sehen ist. Diese Struktur wird später in Kap. 11 detaillierter diskutiert.

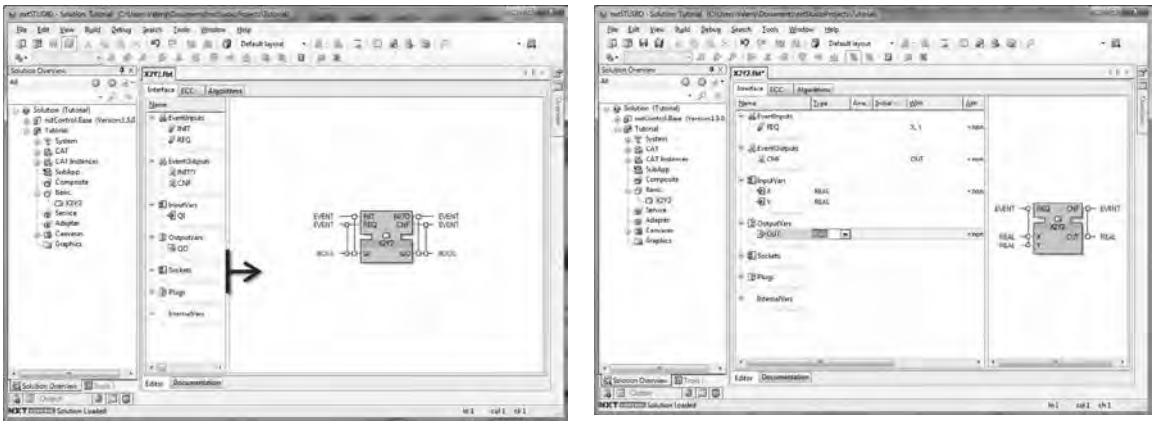
Lassen Sie uns eine neue Solution namens "Tutorial" erzeugen, wie in Abbildung 7-9 (a) gezeigt; dann den Cursor auf den Basic-Eintrag[2] im Solution-Baum setzen, die rechte Maustaste drücken und New Item im Pull-Down-Menü auswählen. Im erscheinenden Dialog wird diesem neuen Function Block-Typ der Name X2Y2 zugewiesen.



a) b)

Abbildung 7-9 (a) Erzeugen einer neuen Solution; (b) Erzeugen eines neuen Basic Function Block-Typs in der Solution.

Man sieht nun einen neu erzeugten FB-Typ mit einem Standard-Interface wie in Abbildung 7-10 (a) gezeigt. Wir ziehen die Grenze des Interface-Ausschnitts nach rechts und editieren die Interface-Elemente in der Tabelle entsprechend Abbildung 7-10 (b). Dazu werden die Events INIT und INITO gelöscht und die Daten QI und QO umbenannt in X und OUT, ihr Datentyp in REAL geändert. Danach fügen wir einen weiteren Eingang Y ebenfalls vom Typ REAL hinzu.



a) b)

Abbildung 7-10 (a) Öffnen des neu geschaffenen FB-Typs; (b) Ändern des FB-Typ-Interfaces.

Um eine Verbindung zwischen einem Event-Eingang / Ausgang und Datenein-und Ausgängen herzustellen,

Prozessinterface: Eingänge lesen und Ausgänge schreiben

Das Prozessinterface eines Devices ist definiert durch Service Interface Function Blocks, die die Werte von Sensoren, die mit den Eingangsports verbunden sind, lesen können und die Werte von Aktoren über Ausgangsports schreiben. Abbildung 12-8 zeigt beispielhaft die Benutzung solcher Service Interface Function Blocks in einer Steuerungsanwendung. In der Abbildung sind die Eingänge und Ausgänge des Steuerungs-Function Blocks FEEDER_CTL mit den Signalen des Controllers ADAM 6650 über den zugehörigen Service Interface Function Block des Typs A6650 verbunden. Dieser FB-Typ wird vom Hersteller zusammen mit der Hardware ausgeliefert. Die Anzahl seiner Daten-Ein- und Ausgänge korrespondiert mit der Anzahl der Eingangs- und Ausgangskontakte in der Controller-Hardware. Der IO-Function Block scannt die Eingänge des Controllers im Zeitraster, das durch die Eingangsvariable DT festgelegt wird (50 ms in diesem Beispiel). Für den Fall, dass eine Änderung gegenüber dem vorausgegangenen Scan eingetreten ist, wird das Event IND ausgesandt.

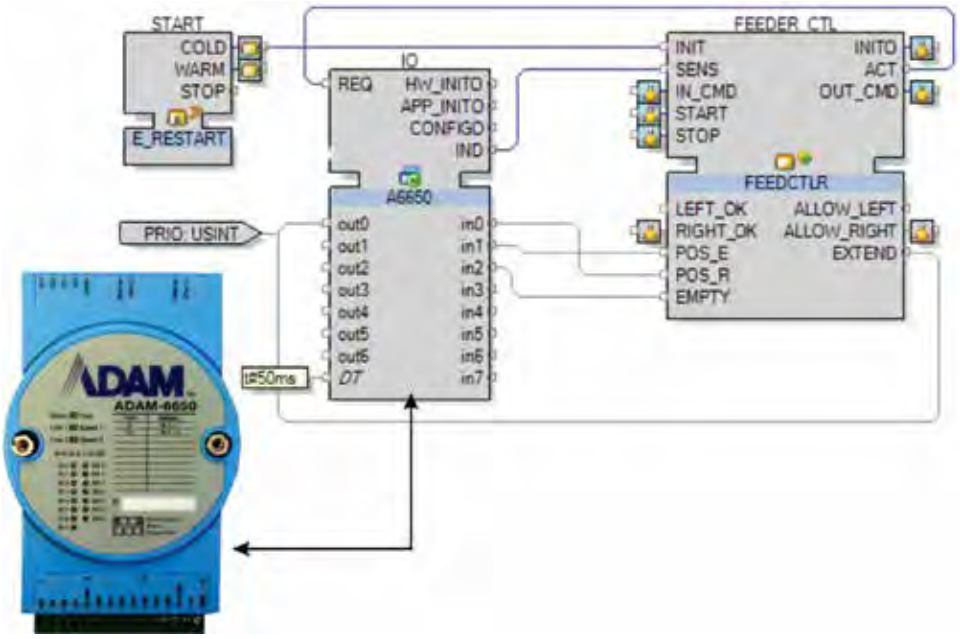


Abbildung 12-8 Diskrete Eingänge und Ausgänge des Moduls ADAM 6650.

Symbolic Links

Eine weitere Methode, Peripheriemodule zu adaptieren, wird bei Controllern mit modularen Eingangs/Ausgangsmodulen angewandt. Sie basiert auf dem Konzept der „Symbolic Links“. Die Hauptvorteile dieser Methode im Vergleich zur vorher betrachteten Lösung sind Flexibilität und ein klareres Design, indem ein „Spaghetti-Gewirr“ von Verbindungsbögen vermieden wird.

Symbolic Links erlauben es, Ausgänge und Eingänge von Function Blocks über die Hierarchie geschachtelter Konstrukte hinweg zu verbinden, wie in Abbildung 12-9 a) dargestellt. Ein Symbolic Link kann definiert sein in Form eines Publishers oder Subscribers, der tief innerhalb eines Composite Function Blocks angesiedelt ist. Dann kann er zu einem Eingang oder Ausgang verbunden werden, der sich innerhalb weiterer geschachtelter Strukturen befindet, wie in Abbildung 12-9 b) gezeigt.

bunden werden. Die Function Blocks sind mit einem Adapter-Link verbunden, der dem KBUS-Interface im Device entspricht.

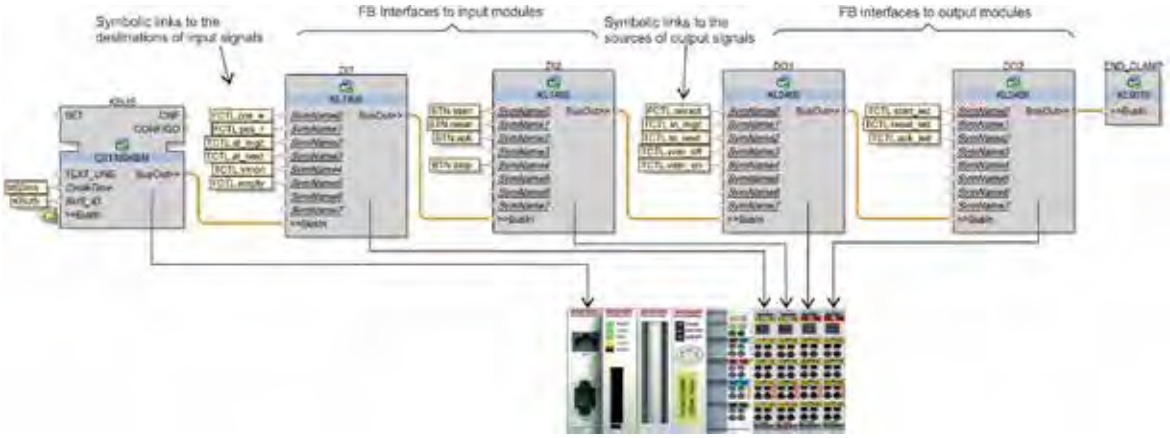


Abbildung 12-11 Symbolic Links zur Verbindung von Function Blocks einer Application mit Dienst-Interfaces, die die I/O-Module eines Beckhoff-IPC repräsentieren.

Zusammenfassung

Service Interface Function Blocks bilden das Interface zwischen einer Applikation und der spezifischen Funktionalität, die durch die Hardware und die Systemsoftware von eingebetteten Devices bereitgestellt wird. Der Code eines Service Interface Function Blocks kann verborgen sein, die verfügbaren Mittel werden jedoch grafisch in Form von Dienstsequenzen dargestellt, um ihre Funktionalität zu spezifizieren und zu dokumentieren.

Übungen für Kapitel 12

In der FLASHERD Systemkonfiguration:

1. Benutzen Sie PUBLISH- und SUBSCRIBE-Function Blocks anstelle der automatisch eingefügten Kommunikationsmechanismen.
2. Ersetzen Sie die PUBLISH/SUBSCRIBE-Kommunikation durch die CLIENT/SERVER-Kommunikation.
3. Spezifizieren Sie einen Function Block für das rein eventgetriebene Lesen von Booleschen Eingängen.

BRES in ähnlicher Weise hinzu.

4) APP2 verteilt auf neun Devices: jedes Ventil und jeder Sensor wird auf ein separates Device gemappt. Diese Konfiguration korrespondiert mit dem Extremfall der drahtlos gekoppelten intelligenten Sensoren und Aktoren.

Gang durch den Entwicklungsprozess

Lassen Sie uns zum Beispiel das Verteilungsszenario 2 betrachten, bei dem APP1 auf drei Devices gemappt, wie dies Abbildung 16-16 zeigt.

Um Function Blocks auf ein Device zu mappen (dieses muss mindestens eine Ressource vom Typ EMB_RES aufweisen), wählen wir die gewünschten Function Blocks in der Applikation aus, rechts-klicken diese Blöcke und gehen im Kontextmenü „Mapping“ zum Zieldevice, wie in Abbildung 16-17 zu sehen.

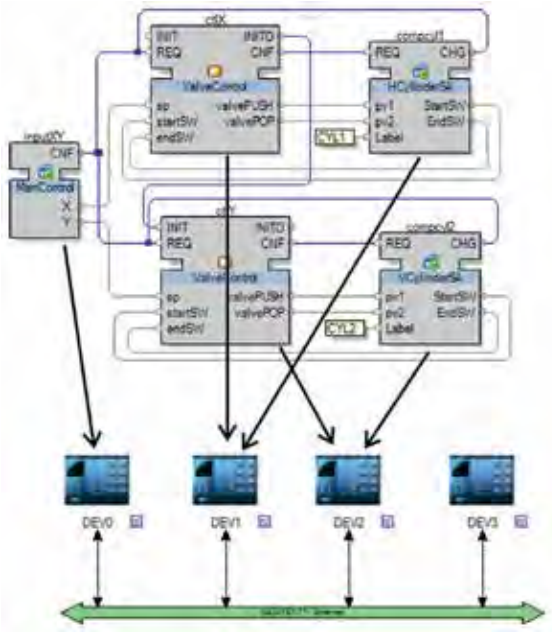


Abbildung 16-16 Verteilung von APP1.

Anschließend besuchen wir die Devices nacheinander, um die Initialisierung korrekt zu konfigurieren. CAT-Instanzen werden von innen heraus durch die eingebetteten E_RESTART - FB-Instanzen initialisiert, Basic– und Composite-Function Blocks benötigen jedoch eine Initialisierung. Wie Abbildung 16-18 zeigt, muss z.B. im Device DEV1 der INIT-Eventeingang von ctlX mit den COLD- und WARM-Ausgängen des START-FB verbunden werden. Eine ähnliche Konfiguration muss im Device DEV2 vorgenommen werden.

18

Weitere Werkzeuge für den Function Block-Entwurf

Dieses Kapitel präsentiert verschiedene weitere Softwaretools, die mit dem IEC 61499-Standard kompatibel sind: die integrierten Entwicklungsumgebungen ISaGRAF, FBDK und 4DIAC.

ISaGRAF

Übersicht

ISaGRAF ist ein Produkt der Rockwell Automation, Canada, und dient zur Entwicklung von Automatisierungssoftware. Beginnend mit der Version 5, die 2005 erschien, wurde die Software um die Unterstützung der IEC 61499 erweitert. Es ist damit möglich, verteilte Steuerungsapplikationen nach IEC 61499 zusammen mit Anwendungsbestandteilen in IEC 61131 zu entwickeln.

ISaGRAF ist das erste vollwertige Automatisierungsprodukt, das die komplette Entwicklungskette auf Basis der IEC 61499 unterstützt. Diese Lösung, die auf dem gut etablierten Standard IEC 61131 basiert, zeigt, wie der neue Standard eingebettet werden kann. Wie in Abbildung 18-1 gezeigt, besteht ISaGRAF aus einer Entwicklungsumgebung und portabler Firmware.

Abbildung 18-1 Struktur von ISaGRAF: Entwicklungsumgebung und Firmware für die IEC 61499- und IEC 61131-3- Systementwicklung.

Nicht alle Konzepte der IEC 61499 wurden implementiert. Die Kommunikation zwischen den Teilen einer Applikation wird über Netzwerkvariable anstelle von Service Interface Function Blocks erreicht. Die äußere Erscheinungsform der Function Blocks in ISaGRAF folgt der IEC 61499, allerdings sehen ihre Interna etwas unterschiedlich aus. Das Execution Control Chart eines Basic Function Blocks ist z.B. in einer Weise definiert, die der Sprache SFC der IEC 61131 ähnelt. Die Abarbeitung der ereignisgetriebenen IEC 61499 - Function Blocks ist auf einem zyklisch gescannten IEC 61131-Laufzeitsystem aufgesetzt.

ISaGRAF erlaubt die Benutzung der traditionellen SPS-Programmiersprachen des IEC 61131-Standards in Function Blocks; z.B. ist die Kontaktplansprache (wie in Abbildung 18-2 gezeigt) möglich.

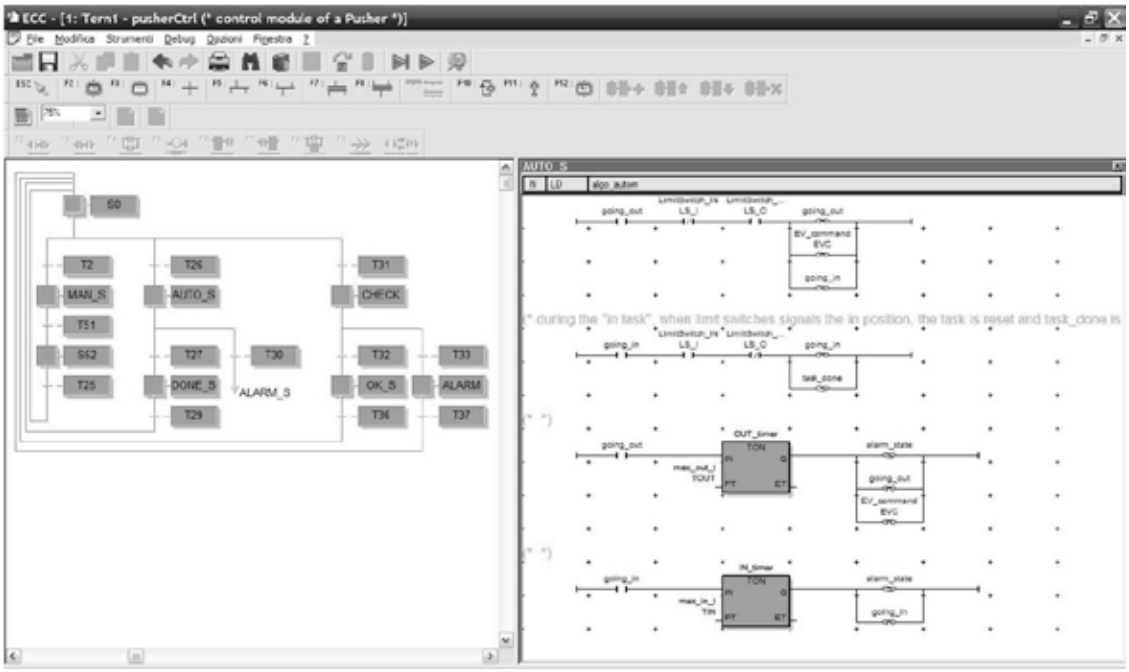


Abbildung 18-2 ISaGRAF-Werkbank nutzt die Kontaktplandarstellung, um Algorithmen in Basic Function Blocks zu programmieren.

Referenzbeispiel

Das LED-Lauflicht - Referenzbeispiel („LEDChaser“), das mit dem ISaGRAF-Demokit (Abbildung 18-3) ausgeliefert wird, soll genutzt werden, um die Unterschiede dieser Implementation zu anderen Tools wie nxt-Studio und FBDK zu zeigen. Das Demokit besteht aus drei identischen Mikrocontroller-Devices mit einigen Eingangs- und Ausgangsports, die mit zwei Tastern, zwei Schaltern und vier LEDs verbunden sind.

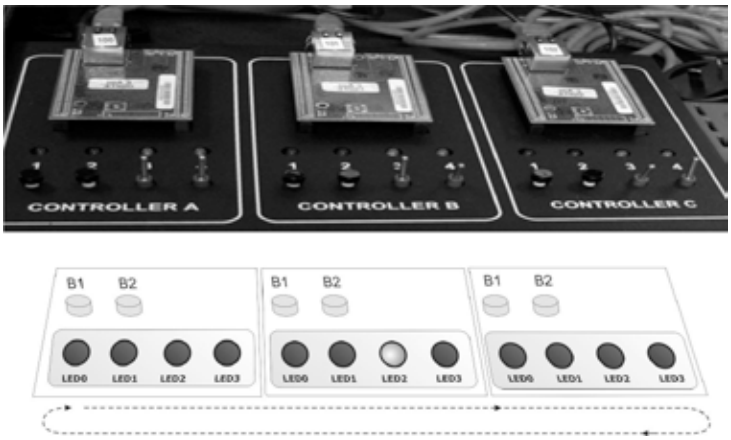


Abbildung 18-3 LED-Lauflichtsystem (LEDChaser).

Die LEDChaser-Applikation arbeitet wie folgt. Beim Hochlauf „läuft“ das Licht von der linken LED (LED3) nach rechts. Die Laufrichtung wechselt, wenn das Licht die ganz rechts befindliche LED erreicht usw. Drückt man den Taster B1 auf einem der Devices, erhöht sich die Laufgeschwindigkeit, und das Drücken von B2 vermindert diese. Zu jedem Zeitpunkt ist nur eine LED aller drei Devices erleuchtet. Das Function Block-Netzwerk dieser Steuerung in der ISaGRAF-Version mit IEC 61499-Function Blocks ist in Abbildung 18-4 gezeigt.

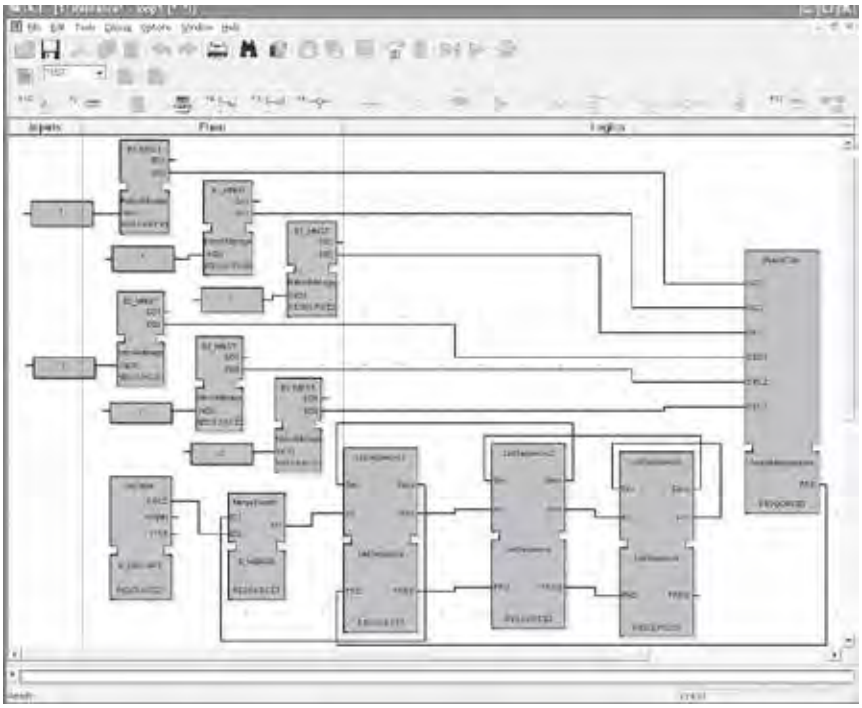


Abbildung 18-4 LedChaser-Programm in ISaGRAF.

Die Steuerung ist dezentralisiert; das Drücken eines Tasters in einem beliebigen Device beeinflusst sofort die Geschwindigkeit des Lauflichts, sogar wenn das Licht sich gerade in einem anderen Device „läuft“.

Wen man einen Taster am Device 3 drückt, während das Licht sich z.B. in Device 1 befindet, dann trifft dieses Ereignis am PeriodManagement-Block ein und eine neue Periodendauer wird an LedSequence1 übertragen. Als Ergebnis ändert sich die Geschwindigkeit (des sich im Device „befindlichen“ Lichts) sofort.

Das Programm umfasst sechs Instanzen des „ButtonManagement“-Function Blocks, eine für jeden Taster. Dieser FB erfasst das Loslassen eines Tasters und erzeugt ein Eventsignal, das zum FB „PeriodicCalc“ gesendet wird; dieser wiederum berechnet die Dauer des Zeitintervalls zwischen den Aktivierungen zweier benachbarter LEDs. Dieser Wert wird an die FBs gesendet, die für die Aussendung der Signale an die LEDs innerhalb eines Devices verantwortlich sind. Dies sind die drei Instanzen des Function Block-Typs „LedSequence“. Dieser FB hat zwei Eingangsevents: „Inc“ und „Dec“. Wenn „Inc“ empfangen wird, zeigt der FB nacheinander alle fünf möglichen Binär-Muster an: ‘0000’, ‘1000’, ‘0100’, ‘0010’, ‘0001’ und wieder ‘0000’, wobei zwischen jedem Wechsel die eingestellte Verzögerungszeit abläuft. (Bei „Dec“ ergibt sich die invertierte Reihenfolge.)

Die „LedSequence“ ist ein Basic Function Block, dessen Logik durch ein ECC definiert ist. In ISaGRAF ist ein ECC in Form eines Sequential Function Charts (SFC) implementiert. Augenscheinlich wollten die ISaGRAF-Entwickler den bestehenden SFC-Editor weiterbenutzen, um der Vertrautheit der Steuerungsingenieure mit diesem Tool Rechnung zu tragen.

Trotz des ähnlichen Aussehens ist die ECC-Semantik in ISaGRAF abweichend von der reinen SFC-Semantik in der IEC 61131 (ebenfalls ISaGRAF). Im „reinen“ SFC kann nicht mehr als eine Transition per SPS-Zyklus auftreten. Nach einer Transition ist das SFC „auf Halt gesetzt“.

In der IEC 61499 wird die ECC-Ausführung fortgesetzt, bis es keine weitere Transition im Chart gibt, die zum Wert TRUE evaluiert. Wenn die Auswertung einer Transition zwischen zwei Zuständen den Wert FALSE

4DIAC IDE und FORTE

Das 4DIAC-Konsortium wurde 2007 von mehreren akademischen und Industrieorganisationen mit dem Ziel gegründet, Open Source-Entwurfstools und Laufzeit-Plattformen für IEC 61499 zur Verfügung zu stellen (4DIAC, 2009). Das Konsortium bildete sich um das Odo Strugger-Labor an der Technischen Universität Wien und das Forschungsunternehmen PROFACTOR (Österreich).

4DIAC hat eine Open Source-IDE unter Eclipse entwickelt (siehe Abbildung 18-18). Diese benutzt die Plug-In-Mechanismen von Eclipse, um Werkzeugflexibilität und –Erweiterbarkeit zu erreichen. Das Werkzeug kann Code für Function Block-Applikationen generieren und auf zwei Laufzeitumgebungen ausführen: Forte und FBRT.

Forte ist eine weitere lieferbare Leistung des 4DIAC-Konsortiums. Es handelt sich hier um eine Laufzeitumgebung, die in C++ geschrieben ist. Dies macht sie anerkanntermaßen effizienter als FBRT (obwohl einige Tests auf einem PC überraschenderweise das gegenteilige Ergebnis zeigen). Forte wurde für einen großen Umfang von Hardwarecontrollern portiert. Es besitzt nachgewiesenermaßen Echtzeitfähigkeiten, wie in Zoitl (2009) näher ausgeführt. Abbildung 18-19 zeigt ein Beispiel einer solchen Anwendung: ein Roboter mit sechs Achsen und einer verteilten Function Block-Steuerung wurde von PROFACTOR (Österreich) aufgebaut. Das Steuerungssystem ist mithilfe von drei vernetzten Embedded Controllern implementiert, von denen jeder für zwei Achsen zuständig ist.

Forte wird in der kommerziellen Werkzeugkette von nxtControl eingesetzt. Ergänzend stellen die 4DIAC-IDE und Forte eine gute Alternative zu FBDK/FBRT dar, speziell für Forschungs-, Ausbildungs- und Trainingszwecke.

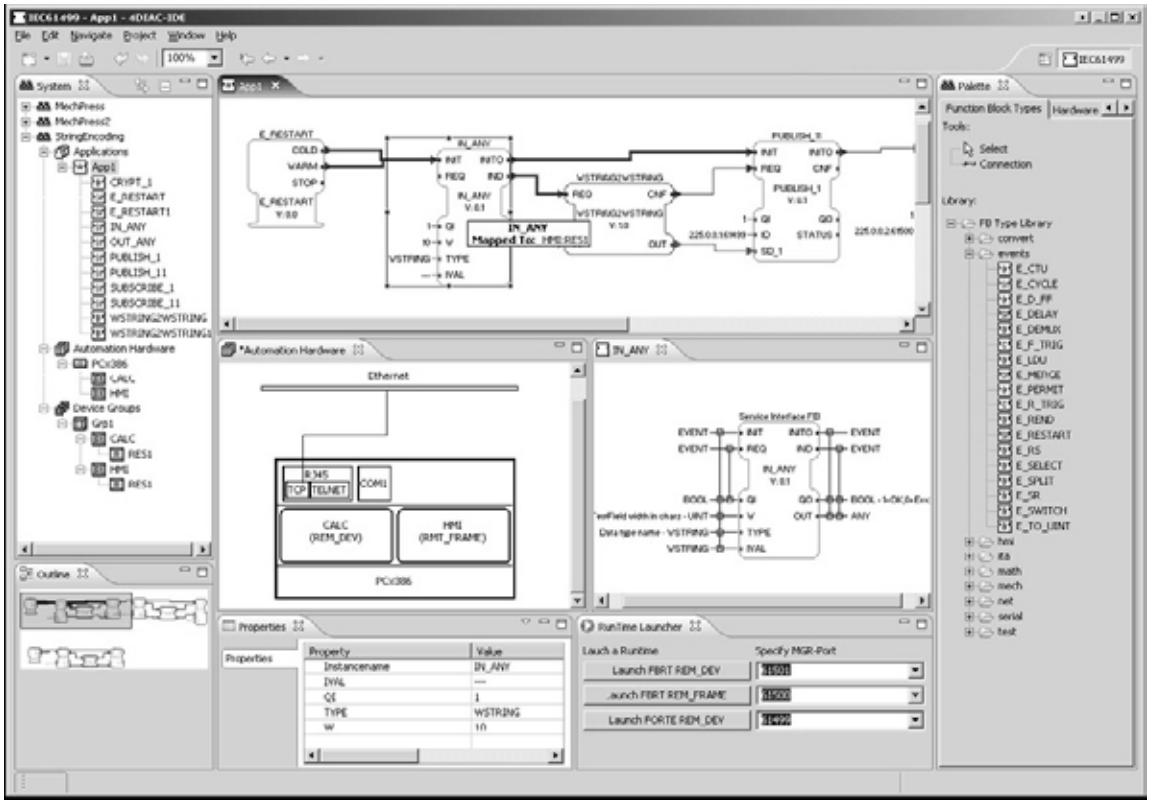


Abbildung 18-18 Screenshot der 4DIAC-IDE.

ergibt, endet die Ausführung des Blocks in diesem Scan. Im nächsten Scan wird an gleicher Stelle mit einer neuen Auswertung dieser Transition fortgesetzt. Betrachten wir z.B. das SFC „LEDSequence“ in Abbildung 18-5.

Abbildung 18-5 ECC des ‘LEDSequence’-Function Blocks.

Die Semantik im „LEDSequence“-FB erlaubt es, die Ausgabe aller fünf Bitsequenzen in einem der Zweige T2-T7 (für das Lauflicht nach rechts) und T8-T13 (für das Lauflicht nach links) zu programmieren. Jede Bitkombination benötigt eine beträchtliche Zeitspanne (Sekunden), verglichen mit dem Zeitmaßstab des Controllers (Milli- oder Mikrosekunden). Daher wird der Function Block viele Male während eines aktiven Zustands (z.B. S6) betreten und verlassen, solange die folgende Transition T6 FALSE ist, nur um die Transitionsbedingung zu checken.

Vergleich des Ausführungsmodells von ISaGRAF mit dem von nxtStudio

ISaGRAF implementiert das *zyklische* Ausführungsmodell: innerhalb einer einzelnen Ressource werden die FBs in einer festen Reihenfolge aufgerufen, die vor der Ausführung festgelegt wird. Die Werte der FB-Ausgänge (Events und Daten) sind den „nachfolgenden“ (unterhalb angeordneten) Function Blocks sofort im gleichen Scan verfügbar, jedoch den vorhergehend (oberhalb) angeordneten FBs erst im nächsten Scan. Das zyklische Modell wurde gewählt, um die Kompatibilität zu ISaGRAFs IEC 61131-Laufzeit sicherzustellen.

Im Folgenden werden die Details von ISaGRAFs Ausführungsmodell diskutiert und mit den Implementationen verglichen, bei denen Function Blocks strikt durch Events aufgerufen werden (z.B. nxtStudio, FBDK usw.). Diese Implementationen werden als zugehörig zum *Event Driven Invocation* - Paradigma – auch EDI - bezeichnet.

Syntax und Daten

ISaGRAF unterstützt das Standard-XML-Format der IEC 61499 nicht, daher ist es nicht möglich, Function Blocks aus oder nach FBDK zu importieren oder zu exportieren. ISaGRAF erlaubt globale Variable, so dass eine strenge Datenkapselung innerhalb von Function Blocks nicht möglich ist.

Event—Daten-Zuordnung

Die IEC 61499 definiert die Zuordnungsmechanismen zwischen Event- und Daten- Ein- und Ausgängen eines Function Blocks. Die Semantikfestlegungen schreiben vor, dass nur assoziierte Daten (z.B. Eingänge) übernommen werden, wenn die zugeordneten Events eintreffen. Daher wird im Function Block in Abbildung 18-6 (links) beim Eintreffen des Events REQ nur der Dateneingang V1 aktualisiert/übernommen.

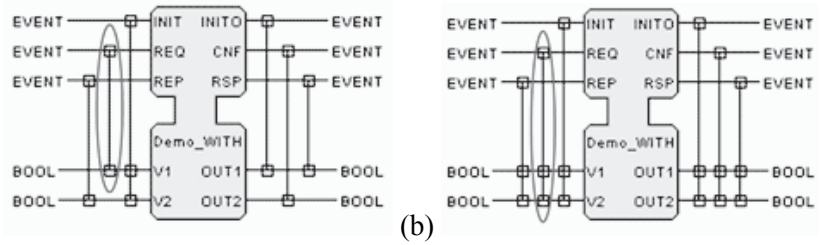


Abbildung 18-6 Event-/Datenzuordnung (links) kann in ISaGRAF nicht implementiert werden (rechts).

In ISaGRAF wird keine explizite Zuordnung zwischen Events und Dateneingängen bzw. Ausgängen unterstützt. Dies kann man als gleichbedeutend zu dem Fall ansehen, in dem jedes Events mit jedem Datenelement assoziiert ist. In FBDK wäre das äquivalent zum Function Block in Abbildung 18-6 (rechts).

Eventerkennung und Eventlebensdauer

In ISaGRAF beeinflusst das Eintreffen oder Fehlen von Events den Aufruf von Function Blocks nicht. Mit dem Aufruf muss das Vorkommen eines Events im ECC etabliert werden. Das Event kann dann in anderen Operationen im ECC genutzt werden, z.B. als Transitionsbedingung.

Betrachten wir das Beispiel eines Function Blocks in nxtStudio, der mit verschiedenen Ausgangsevents auf unterschiedliche Eingangsdaten reagieren soll (Abbildung 18-7).

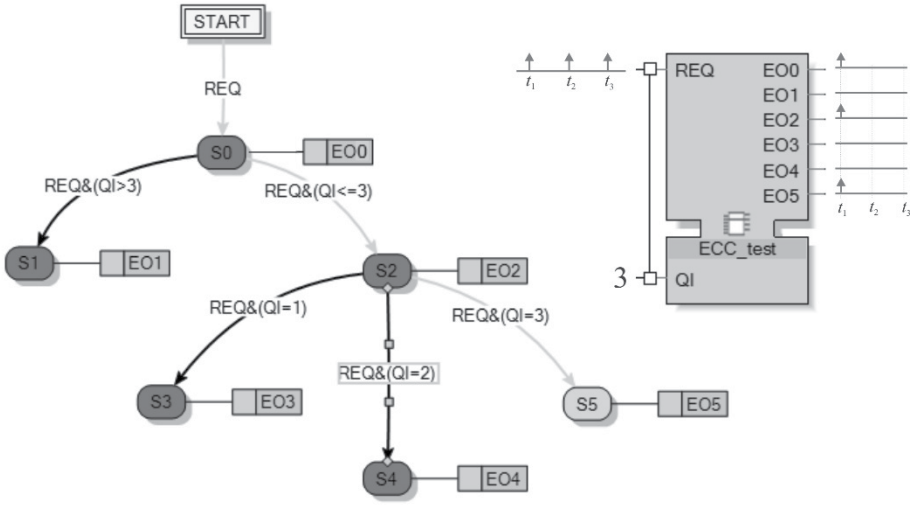


Abbildung 18-7 Ausführung eines Function Blocks in nxtStudio.

Hier gibt es drei „wartende“ Zustände START, S0 und S2, in denen der FB auf das Eintreffen des Events REQ wartet, um in einen neuen Zustand zu springen. Beim ersten Eintreffen des REQ-Events im START-Zustand erfolgt zunächst der Übergang in den Zustand S0 und sodann sofort der Übergang in den State S2 und schließlich in den S5, da das REQ-Event solange „lebt“, bis keine Transition mehr zu TRUE evaluiert oder eine

Anhang C: Korrekturen, die in der zweiten Ausgabe des Standards implementiert wurden

Überblick über die Korrekturen

1. Temporäre Variablen in Algorithmen
2. Syntax und Semantik des ECC
3. Nutzung von Adaptern
4. Subapplications
5. Netzwerk-Segmente
6. Hinzugefügte FBs zur Kommunikation mit SPS

Temporäre Variablen in Algorithmen

Die Syntax erlaubt nun, dass in den Algorithmen von Basic-FBs temporäre Variablen deklariert werden.

```
fb_algorithm_declaration ::=
    ,ALGORITHM‘ algorithm_name ,IN‘ language_type ,:‘
      [temp_var_decls]
      algorithm_body
    ,END_ALGORITHM‘

temp_var_decls ::= <as defined in IEC 61131-3>
```

Syntax und Semantik des ECC

Die Einstellungen des ECC zielen darauf ab, eindeutige und intuitiv erkennbare ECC-Ausführungsregeln zu etablieren. Zusätzlich wurde die Syntax erweitert, um die Benutzung von Plug/Socket-Ein/Ausgaben in ECC-Übergangsbedingungen zu erlauben (Regeln 2, 3, 5). Die Korrekturen in der Syntax von Übergangsbedingungen betonen die Tatsache, dass ein FB durch ein einzelnes Event aktiviert wird; dadurch ist die Situation mit mehreren Eventeingängen, die gleichzeitig TRUE werden, unmöglich.

1.	`ec_action ::= algorithm_name |`
	`(,->‘ ec_action_output) |`
	`(algorithm_name ,->‘ ec_action_output)`

2.	`ec_action_output ::=`
	`([plug_name ,.‘] event_output_name)`
	`| (socket_name ,.‘ event_input_name)`

3.	`ec_transition_event ::=`
	`([plug_name ,.‘] event_input_name) |`
	`(socket_name ,.‘ event_output_name)`