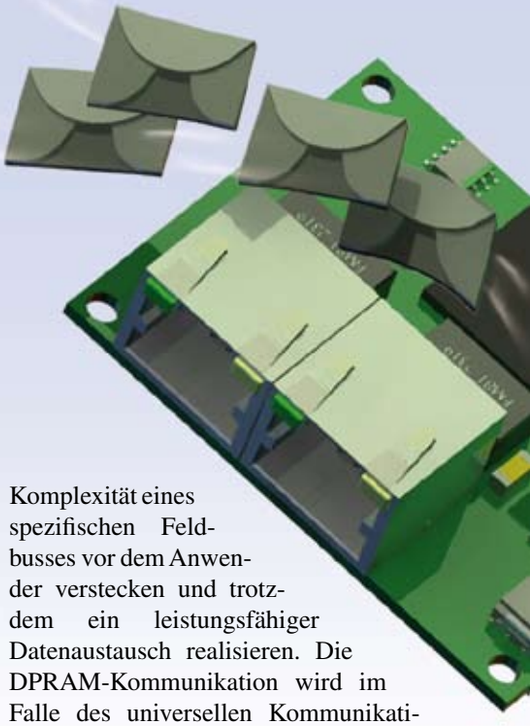
Michael Naugk

Die dynamischen Objektverzeichnisse



Komplexität eines spezifischen Feldbusses vor dem Anwender verstecken und trotzdem ein leistungsfähiger Datenaustausch realisieren. Die DPRAM-Kommunikation wird im Falle des universellen Kommunikationsmoduls „Port Unified Data Interface“ – kurz PUDIn – abgewickelt. Das PUDIn implementiert die Zwischenschicht, die das Application Programming Interface (API) des jeweiligen Protokollstacks auf eine einheitliche Applikationsschnittstelle umsetzt und die komplette DPRAM-Kommunikation abhandelt.

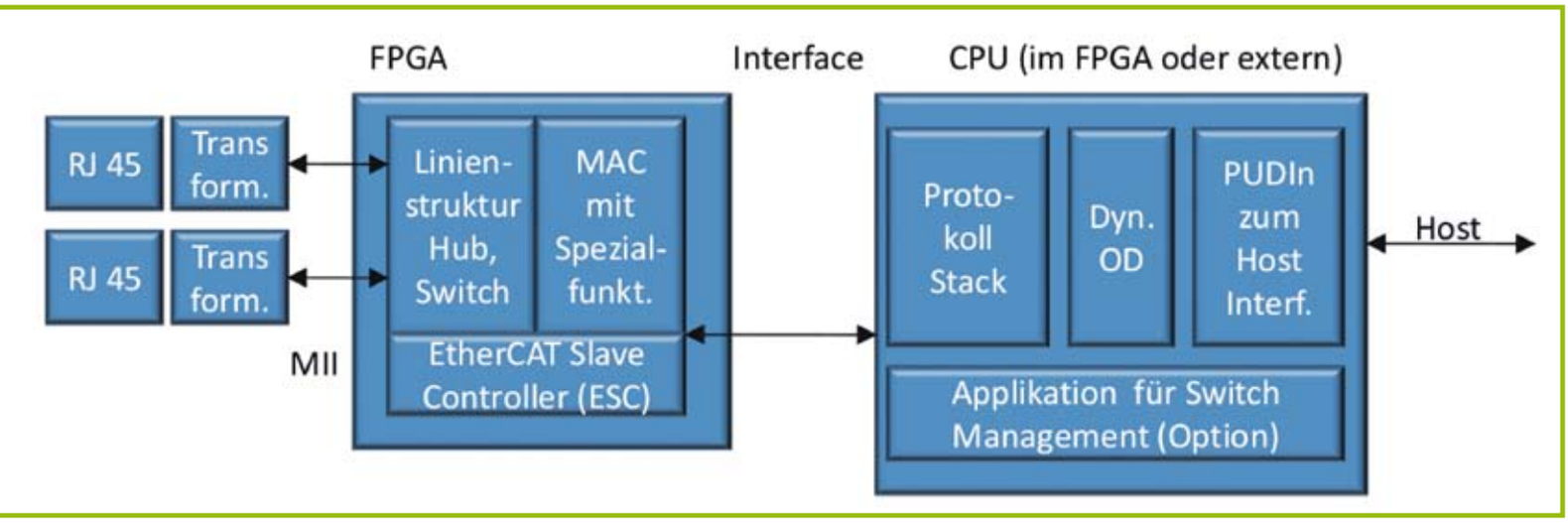
Mit FPGA-Bausteinen steht eine universelle Hardware-Plattform für die unterschiedlichsten Ethernet-Protokolle zur Verfügung. Um hierfür eine einheitliche, „pflegeleichte“ Schnittstelle zu schaffen, bieten sich dynamische Objektverzeichnisse an.

Kaum ein Hersteller kann sich spezifisch für die Unterstützung nur eines Industrial-Ethernet-Systems entscheiden. Eine entsprechende Kommunikationsschnittstelle besteht für gewöhnlich aus einem FPGA, der die Bus-Kommunikation übernimmt, und einem Applikationsprozessor, auf dem die eigentliche Anwendung läuft. Der Datenaustausch der beiden Einheiten findet über einen Dual-Ported-RAM (DP-RAM) statt. Im FPGA wird neben dem Layer-2-IP-Core, der die Bus-Anbindung implementiert, ein Softcore-Prozessor zur Ausführung des Protokollstacks integriert.

Für die jeweiligen Ethernet-Protokolle sind spezielle FPGA-Bausteine verfügbar, wie zum Beispiel der Ethercat-Slave-Controller (ESC) IP-Core. Auf Standard-Ethernet aufsetzende Lösungen lassen sich mit jedem MAC-Controller umsetzen. Für manche Feldbusse sind zudem spezielle Bausteine einsetzbar, die die Protokollspezifika abbilden. Um höhere Flexibilität zu erreichen, lohnt sich jedoch auch hier der Einsatz einer dedizierten IP-Core-Lösung. Der so genannte „Enhanced Ethernet MAC“ (PE²MAC) von Port beispielsweise ist mit speziellen Filter- und Auto-Reply-Funktionen ausgestattet, um etwa die Antwort-Zeiten in Ethernet-Powerlink-Netzwerken zu optimieren. Der Dual-Ported-RAM ist ebenfalls als IP-Core im FPGA realisierbar. Als Alternative kann der Datenaustausch zwischen Protokollstack und Applikation auch über ein serielles Protokoll wie zum Beispiel SPI erfolgen.

Die Trennung von Netzwerkzugriff und der eigentlichen Anwendung hat für den Nutzer diverse Vorteile. Zum einen wird die Kommunikation nicht durch Unterbrechungen der Anwendungssoftware gestört, was besonders bei Antriebsanwendungen relevant ist. Zum anderen lässt sich dadurch die

Die Aufteilung in mehrere Datenkanäle ermöglicht dabei die Priorisierung von Daten. Dadurch ist sichergestellt, dass Prozessdaten immer zeitnah nach dem Empfang auf dem Netzwerk an die Applikation übertragen werden beziehungsweise rechtzeitig zum Versenden vorliegen. PUD-In prüft, welche der empfangenen Daten aktualisiert wurden und meldet dies der Host-Applika-



Eine FPGA-basierte Busanbindung besteht generell aus den folgenden Komponenten: Linienstruktur-Komponente (Hub/Switch) – MAC – Protokollstack – PUDIn (DP-RAM).

(Bilder: Port)

tion. Das Kommunikationsmodul wird je nach (Ethernet-)Feldbus mit einer speziellen Firmware ausgestattet, welche den jeweiligen IP-Core und Protokollstack enthält. Die Host-Applikation ist davon unabhängig und muss beim Austausch des Feldbus-Protokolls nicht verändert werden. Das Übertragen der Modul-Firmware kann über den DPRAM mit Hilfe einer PUDIn-Funktion oder über das Netzwerk erfolgen.

Das Objektverzeichnis

Um Daten auf dem Bus auszutauschen, sind diese zunächst in Form von Kommunikationsobjekten (Objekte) zu definieren. Damit ein Feldgerät Daten mit anderen Kommunikationsteilnehmern austauschen kann, müssen diese die Objekte des Gerätes kennen. Die Gesamtheit aller Kommunikationsobjekte, welche über das Netzwerk adressiert, gelesen und geschrieben werden können, bilden das Objektverzeichnis eines Feldgerätes. Jedes Objekt hat einen Datentyp, bestimmte Zugriffsrechte, minimales und maximales Limit und einen aktuellen Wert. Welche Objekte ein Feldgerät implementiert hat, ist in dessen Gerätebeschreibung dokumentiert. Wie diese Kommunikationsobjekte konkret aussehen, definiert jedes Kommunikationssystem für sich (siehe *Kasten* auf Seite 77).

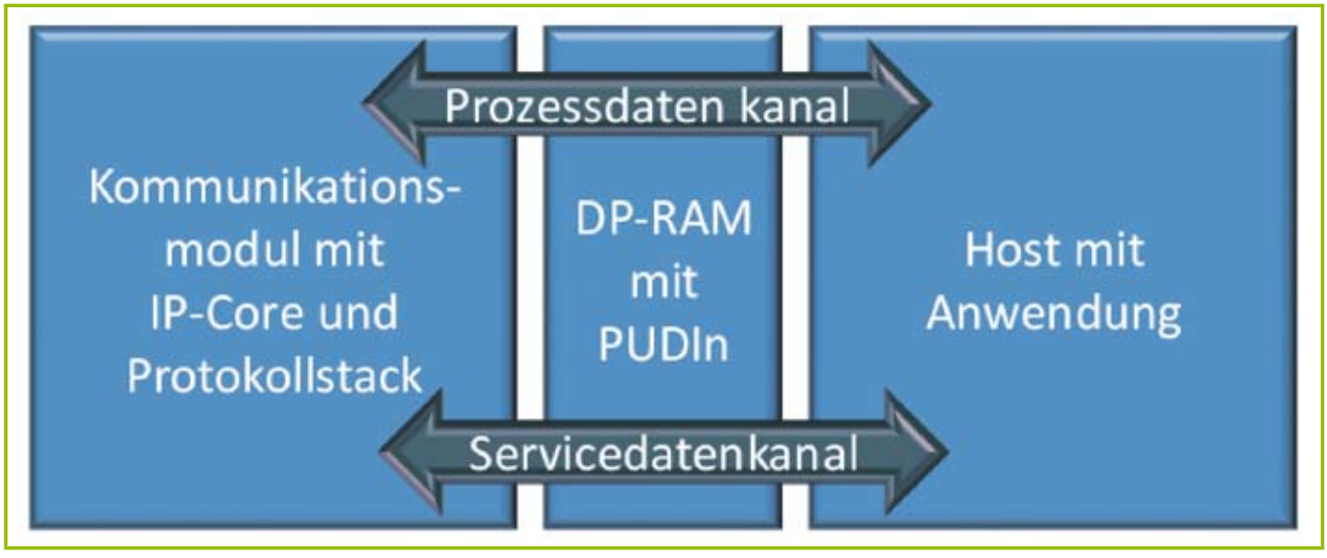
Aus Sicht des Geräte-Entwicklers muss das Objektverzeichnis in irgendeiner Form im Gerät abgelegt und organisiert werden. Die Anwendungsschicht schreibt beispielsweise die Werte der digitalen Eingänge in die Eingangsobjekte und setzt die Ausgänge abhängig vom Inhalt der Ausgangsobjekte. Der Protokollstack kopiert die Werte zwischen Objektverzeichnis und Netzwerk-Paketen.

Wie das Objektverzeichnis im Gerät zu organisieren ist, ist von Protokollstack zu Protokollstack unterschiedlich. Im einfachsten Fall wird die Verwaltung in die Applikation verlagert, so dass der Protokollstack auf dem Kommunikationsmodul Anfragen einfach über den DPRAM weiterleitet. Für den Anwendungsentwickler bedeutet das, dass er Datentypen, Zugriff-Flags und Limits sowie aktuelle Werte der Objekte aufwendig selbst verwalten muss. Deutlich einfacher wird es, wenn der Stack diese Aufgabe übernimmt. Dafür muss der Protokollstack die Kommunikationsobjekte kennen.

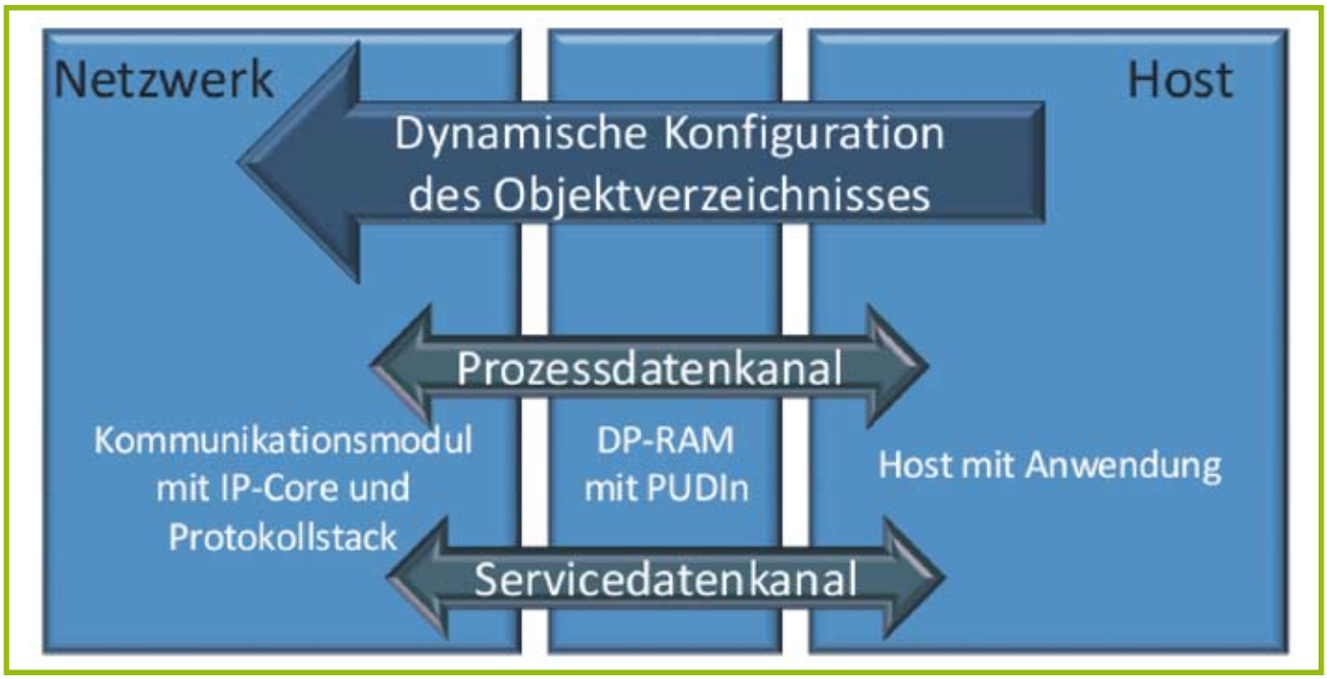
Zur Verwaltung des Objektverzeichnisses gibt es verschiedene Möglichkeiten. Eine sehr effiziente ist die Speicherung in C-Strukturen, da sie schnellen und sicheren Zugriff gewährleistet. Dafür wird das Objektverzeichnisses statisch in die Modul-Firmware eincompiliert. Das spart zwar Speicher für Daten und Code, verlangt aber ein erneutes Compilieren, sobald ein Objekt hinzugefügt, gelöscht oder verändert wird. Erfahrungsgemäß passiert das bei der Entwicklung eines Geräts sehr häufig, weshalb dieses Verfahren für den Einsatz in Kommunikationsmodulen wenig geeignet ist.

Dynamik steht im Vordergrund

Wird das Objektverzeichnis dynamisch beim Start des Kommunikationsmoduls aufgebaut, so kann die Modul-Firmware – unabhängig vom Objektverzeichnis – immer gleich sein, und der Geräte-Entwickler spart sich das erneute Compilieren. Die Host-Applikation definiert, wie das Objektverzeichnis aussieht und überträgt es zum Kommunikationsmodul. Dazu wurde das PUDIn-Protokoll erweitert, um dem Protokollstack über einen zusätzlichen DPRAM-Kanal mitzuteilen, wie das Objektverzeichnis aufgebaut ist. Diese Host-Schnittstelle ist für alle Feldbusse gleich; die Umwandlung der Objekte für den spezifischen Protokollstack erfolgt erst im Kommunikationsmodul. Die Überprüfung von Zugriff-



Der Datenaustausch zwischen Kommunikationsmodul und Host wird in verschiedene Kanäle aufgeteilt und erfolgt über einen Dual-Ported-RAM.



Dynamische Konfiguration: Die Übertragung des Objektverzeichnisses vom Host zum Modul geschieht über einen zusätzlichen DP-RAM-Kanal, welcher von PUDIn verwaltet wird.

Flags, Datentypen und Limits lässt sich dadurch schon auf dem Modul erledigen und braucht nicht von der Applikation durchgeführt werden, wodurch diese zusätzlich entlastet wird. Hierbei muss der Protokollstack Unterstützung liefern, um das Objektverzeichnis und die Zugriffe darauf zu verwalten.

Die Erstellung des C-Codes für das Objektverzeichnis ist eine aufwendige und fehlerträchtige Aufgabe. Zusätzlich muss vom Entwickler eine Gerätebeschreibung erstellt werden, was besonders durch die neuen XML-Formate Schwierigkeiten bereitet. Ideal ist es, wenn ein Tool all diese Aufgaben automatisch erledigt und für die Erzeugung von dynamischen Objektverzeichnissen ausgelegt ist. Das heißt: Der Anwender legt die Objekte

Bus-Systeme und ihre Kommunikationsobjekte

CANopen
Objekte in CANopen-Geräten werden über einen 16 Bit breiten Index adressiert. Ein Index enthält einen oder mehrere Subindizes, wodurch neben einfachen Datentypen auch Arrays und Records nutzbar sind. Das Objektverzeichnis wird in drei Bereiche unterteilt: Die Objekte im Kommunikationsbereich dienen der Parametrierung der CANopen-Kommunikation. Im herstellerspezifischen Teil sind die vom Gerät zum Datenaustausch benötigten Objekte beliebig definierbar. Im Geräteprofil-Bereich stehen dafür standardisierte Objekte je nach Gerätetyp zur Verfügung. Welche Objekte als Prozessdaten gesendet beziehungsweise empfangen werden können, legt das PDO-Mapping-Flag fest. Die Gerätebeschreibung erfolgt im Electronic Datasheet (EDS).

Ethernet Powerlink
Für Ethernet Powerlink wurde die CANopen-Anwendungsschicht und damit das Objektverzeichnis adaptiert und der Kommunikationsbereich entsprechend angepasst; Objekte im herstellerspezifischen Teil und aus Geräteprofilen sind unverändert nutzbar. Dadurch kommen die leistungsfähigen Profile des CiA auch hier zur Anwendung. Als Gerätebeschreibungssprache wurde die XML Device Description (XDD) definiert.

Ethercat
Auch für das Ethercat-Applikationsprotokoll „CAN Application Layer over Ethercat" wurde das bewährte CANopen-Objektverzeichnis übernommen und im Kommunikationsbereich angepasst. Die Gerätebeschreibung erfolgt in Ethercat-Slave-Information-Dateien (ESI).

Profinet IO
Die Adressierung von Profinet-Kommunikationsobjekten erfolgt über Slot und Subslot. Bei modularen Geräten stellt ein Slot einen Steckplatz für ein Modul dar. IO-Daten werden direkt auf Subslot-Ebene geschrieben und gelesen; für Parameterdaten gibt es eine weitere Adressierungsstufe – den Index. Geräteweite Parameter werden über Indizes im Device Access Point adressiert, der für gewöhnlich über Slot 0/Subslot 0 erreichbar ist. Zur Trennung von mehreren Anwendungen, die auf einem Gerät laufen, dient der Application Process Identifier (API). Welche APIs, Slots und Module ein Profinet-Gerät implementiert hat, wird in der General Station Description Markup Language (GSDML) dokumentiert.

Ethernet/IP
Das Ethernet/IP-Objektverzeichnis ist nach objektorientierten Prinzipien organisiert. Die Adressierung von Daten erfolgt über Klassen-ID, Instanz-ID und Attribut-ID. Das unterliegende Common Industrial Protocol definiert verschiedene Objekte, welche zur Verwaltung der Kommunikation dienen. Zur Organisation von Anwenderdaten kann der Geräte-Entwickler eigene Klassen definieren oder auf vorgefertigte Geräteprofile zugreifen. Die Gerätebeschreibung erfolgt im CIP-spezifischen Electronic Datasheet (EDS).

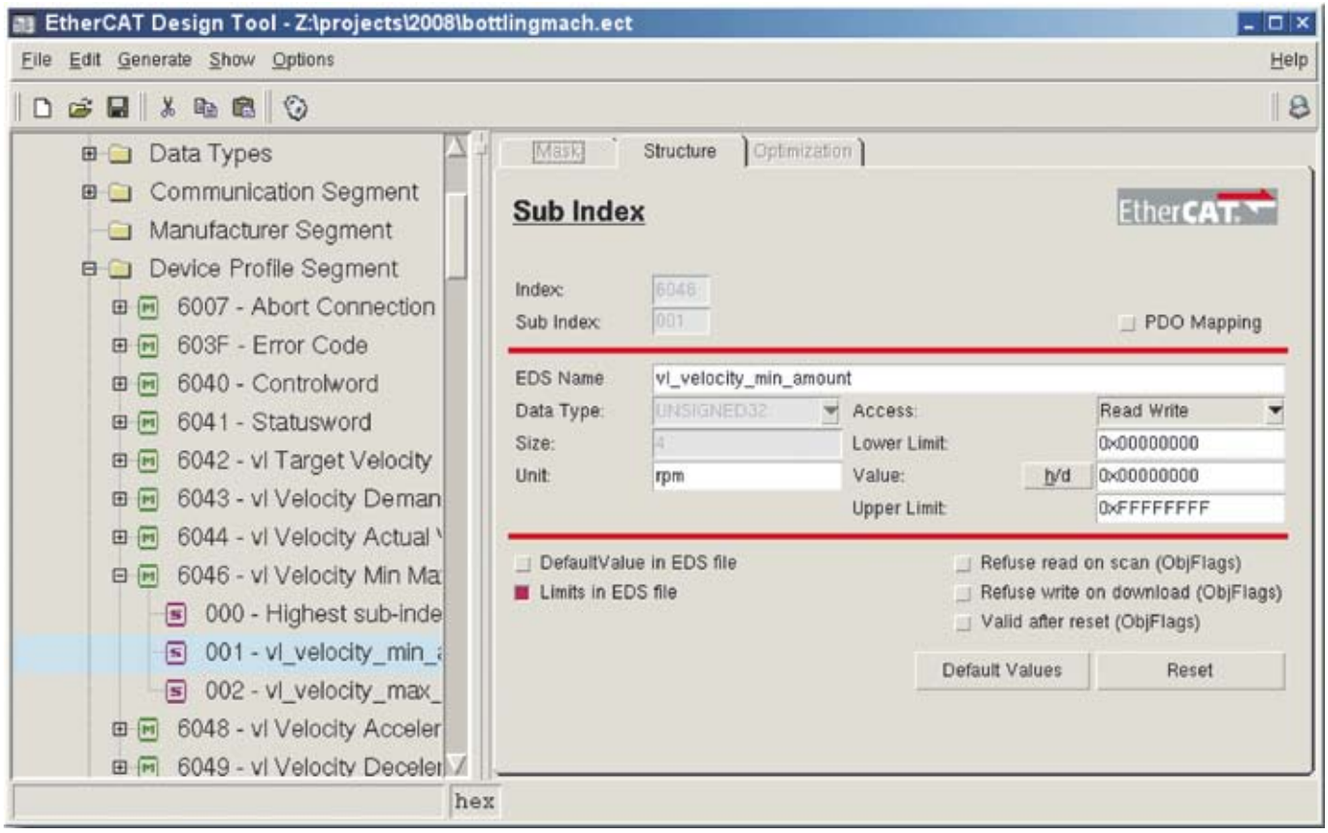
mit Datentyp, Zugriffsrechten, Limits und Default-Wert komfortabel im Tool an und dieses erzeugt das Objektverzeichnis, welches in die Applikation auf der Host-Seite eincompiliert wird. Zur Konfiguration des Kommunikationsmoduls werden zusätzlich die notwendigen PUDIn-Funktionsaufrufe generiert, um das Objektverzeichnis dynamisch über den DPRAM zum Protokollstack zu übertragen. Weiterhin wird die Gerätebeschreibung und eine Anwenderdokumentation generiert. Dabei stellt das Design-Tool die Konsistenz zwischen dem Objektverzeichnis der Host-Seite, dem Objektverzeichnis des Kommunikationsmoduls, der Gerätebeschreibungsdatei (EDS, XDD, ESI, GSDML) und der Dokumentation sicher.

Im Fall von Port steht für jeden unterstützten Standard ein eigenes Design-Tool zur Verfügung. Um dem Anwender die Umschaltung zwischen verschiedenen Feldbus-Protokollen so einfach wie möglich zu machen, ist ein Merge-Tool integriert. So kann der Anwender ein einmal erstelltes Objektverzeichnis einfach von einem Feldbus auf den anderen übertragen und muss dieses nicht mehrfach anlegen. Das Design-Tool erzeugt neben den Anwenderobjekten die spezifischen Objekte für den ausgewählten Feldbus. Diese sind nur für den Protokollstack relevant und für die Host-Applikation unsichtbar. Durch die automatische Generierung des kompletten Objektverzeichnisses und aller benötigten Funktionsaufrufe zur Konfiguration der Module, welche in die Host-Applikation eincompiliert werden, wird der Anwender von wAufgaben befreit, die die Kommunikation betreffen, und kann den Fokus auf die eigentliche Geräte-Applikation legen.

Neben dem Einsatz auf einem Kommunikationsmodul bestehen viele weitere Anwendungsfälle für den Einsatz von dynamischen Objektverzeichnissen. In modularen Geräten sind je nach gesteckten Modulen Kommunikationsobjekte während der Laufzeit zu erzeugen. Dazu muss die Applikation erkennen, welche Steckplätze mit welchen Baugruppen belegt sind. Weiterhin muss sie dem Protokollstack mitteilen, welche Objekte zur Kommunikation mit den Modulen nötig sind. Auch hinsichtlich des Produktionsprozesses nicht modularer Geräte ergeben sich Vorteile, wenn eine Firmware für eine ganze Produktreihe genutzt werden kann und die Kommunikationsobjekte im Protokollstack je nach Bestückung automatisch beim Start des Geräts erzeugt werden. Ein weiterer Anwendungsfall ist der Einsatz von dynamischen Objektverzeichnissen in Steuerungen, welche Kommunikationsobjekte in Abhängigkeit von den angeschlossenen Feldgeräten benötigen. Diese lesen beim Starten die Gerätebeschreibungen der Slaves ein und leiten daraus das Objektverzeichnis ab, welches über die dynamische Konfiguration dem Protokollstack übergeben wird. *gh*

Michael Naugk

ist Software-Entwickler bei der Firma Port, Halle/Saale.



Die Erstellung des dynamischen Objektverzeichnisses lässt sich im Ethercat-Designtool mit Hilfe einer übersichtlichen Baumstruktur bewerkstelligen.