

Auf den ersten Blick sieht man, dass es zu einem Server beliebig viele Clients geben kann. Im Gegensatz zu einer Punkt-zu-Punkt-Verbindung (z.B. RS232, USB oder auch Bluetooth) können beliebig im Netz verteilte Clients (praktisch gleichzeitig) Daten mit dem Server austauschen.

Und noch ein Unterschied zu den vorgenannten Verbindungen fällt auf: Server und Client können beliebig weit voneinander entfernt sein („soweit das Netz reicht...“). Außerdem ist es leicht möglich, mit Standardkomponenten drahtlose Verbindungen herzustellen.

Die Server-Seite

Die Software auf der Server-Seite muss zwei Aufgaben erfüllen:

Zum einen natürlich (wie der Name Embedded Control schon sagt) die - in die anwendungsspezifische Hardware eingebettete - Steuerung/Regelung/Messwert-Erfassung/ -Verarbeitung.

Zum andern muss In den meisten Fällen zusätzlich ein Kommunikationsprotokoll implementiert werden, was den Datenaustausch mit anderen Geräten ermöglicht, z.B. zur Bedienung oder für die Datenspeicherung. Selten ist eine Embedded-Control-Anwendung vollständig autonom. Es gibt so gut wie immer eine Schnittstelle zur Außenwelt, und sei's nur ein Fehler-Auslese-Stecker für den Service.

Eine typische Konstellation in der Embedded-Control-Historie hat so ausgesehen: Die Steuerung kommunizierte via RS232 mit einem PC. Manchmal auch über „die Centronics“. Wenn eine Netzwerkstruktur erforderlich war, wurde RS485 oder CAN eingesetzt. RS232 und Druckerport sind mittlerweile von USB abgelöst worden.

Natürlich gab (und gibt es) unzählige Bussysteme, die im Embedded-Bereich eingesetzt wurden (und werden). Aber Ethernet war auf diesem Sektor lange Zeit „tabu“. Der Mehraufwand an Hardware (allein schon die Verkabelung) war enorm. Dazu kam komplexe Software. Bereits CAN erfordert einen relativ hohen Softwareaufwand. Ethernet erst recht. Das dafür heute benutzte Transportprotokoll ist TCP/IP (Transmission Control Protocol/Internet Protocol) und arbeitet nach dem Schichtenmodell. Der sogenannte „TCP/IP Stack“ muss für den verwendeten Controller auch zur Verfügung stehen.

Durch die Verbreitung des Internets gibt es heute Ethernet-Hardware, z.B Router und (w)LAN-Geräte „wie Sand am Meer“ und das Schlagwort „Web-enabled“ steht auch bei Embedded-Control-Anwendungen immer häufiger im Pflichtenheft. „Netzwerk-Konnektivität wird im embedded Sektor ein zunehmend ernstzunehmendes Thema“ - so Erhard Scherer, Entwickler des NET7026.

Zurück zum Thema! Mit dem W5300, dem Ethernet-Controller auf NET7026, wird der Aufwand um „ins Netz zu gehen“ minimal. Vom TCP/IP Stack bis zum PHY ist alles „On-Chip“. Den Prozessorbus auf der einen, den RJ45-Stecker auf der anderen Seite, bewältigt der WIZnet-Baustein die komplette Transportebene.

Welches Protokoll für die Kommunikation gewählt wird, ist (Geschmack-)Sache des Programmierers (und muss ja auch mit der Client-Seite „abgesprochen“ werden). Ein einfach zu realisierendes Beispiel folgt weiter unten.

Die Client-Seite

Die Software auf der Client-Seite hängt stark von der Anwendung ab. Je nachdem, was über die Schnittstelle „abgewickelt“ werden soll (z.B. Bedienung, Konfiguration, Messwert-Erfassung/ -Auswertung, Diagnose etc.). Meistens wird man nicht umhin kommen, dafür ein dediziertes Programm zu schreiben, speziell zugeschnitten auf die vorgesehene Art der Kommunikation. Das ist ja genauso, wie bei Verwendung einer traditionellen Schnittstelle. Genauso? Nicht ganz.

Bei einer Netzwerkschnittstelle kann man den Softwareaufwand auf der Client-Seite sogar komplett „auf Null fahren“, d.h. man braucht gar kein spezielles Programm mehr. Das Zauberwort heißt „Browser-basiert“.

Bei einer herkömmlichen RS232-Schnittstelle würde man dazu vielleicht „Terminal-Programm-basiert“ sagen. Das setzt natürlich voraus, dass ein Kommunikationsprotokoll benutzt wird, das von jedem Terminal-Programm beherrscht wird (fast alle Terminal-Programme können z.B. ein VT52 emulieren). Nicht nur, dass man kein spezielles Programm mehr schreiben muss, die Plattform-Unabhängigkeit (solche Terminal-Programme gibt es für jede Hardware) ist ein unschätzbare Vorteil. Allerdings wäre eine derartige Kommunikation heutzutage schon rein optisch nicht akzeptabel...

„Browser-basiert“ ist eine ganz andere Liga. Schon längst können Browser mehr als nur einfache HTML-Seiten anzeigen. Javascript, Java-Applets und spezielle Plugins machen Browser zu universellen, Multimedia-fähigen Alleskännern. Das Server-Client-Modell ist Internet-Realität. „Cloud-Computing“, „Webanwendungen“ - der Browser ist das Frontend und die Arbeit macht irgendwo ein Server. Dieses Prinzip lässt sich ebenso gut auf Embedded-Control-Anwendungen übertragen. Wie bereits erwähnt - es muss nur ein Protokoll benutzt werden, das alle Browser verstehen, z.B. HTTP (Hypertext Transfer Protocol).

Vielleicht muss man Server-seitig mehr Aufwand treiben, dafür können aber die Clients komplett auf Standard Hard- und Software zurückgreifen. Man muss sich klarmachen, was das heißt: Ob Windows- oder Linux-PC, Mac, iPhone, iPod oder iPad (sonstige Smartphones und Tablet-PCs inbegriffen) - kurz, jede Hardware, auf der ein Browser läuft, kann als „Kommunikationseinheit“ eingesetzt werden.

Spätestens jetzt wird klar, was in der Überschrift mit „bequem“ gemeint war...

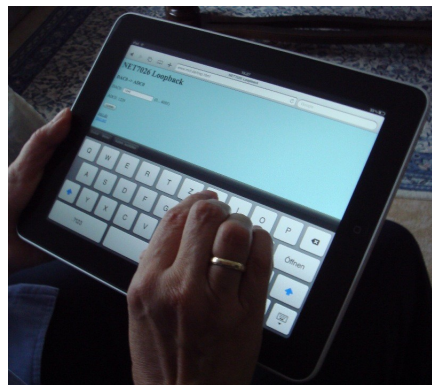
Bei aller Euphorie - eine Netzwerkschnittstelle kann auch problematisch werden. Nicht alle Browser sind immer 100-prozent kompatibel. Und es ist ein gewaltiger Unterschied, ob Daten über ein knapp zwei Meter langes USB-Kabel „laufen“, oder in ein Netzwerk „verschwinden“ und man nicht genau weiß, ob sie an der richtigen Stelle unverseht (und ohne unvorhergesehene Zwischenstationen) wieder zum Vorschein kommen.

Klar, man kann Browser-Typen und -Versionen abfragen, und mit HTTPS steht ein sicheres Protokoll zur Verfügung - alles eine Frage des Aufwands.

Ein Beispiel

Eine beliebte Methode, die Funktionsweise von Schnittstellen zu demonstrieren ist der „Loopback Mode“, wo einfach ein Ausgang auf einen Eingang rückgekoppelt wird. Ein „Kurz“-Schluss quasi, der die eigentliche Datenverarbeitung umgeht und dadurch ohne weitere Hardware auskommt.

Im Falle des NET7026 bietet sich an, einen DAC-Ausgang auf einen ADC-Eingang zu legen. Der DAC kann dann auf beliebige Werte gesetzt werden, die am ADC wieder auslesbar sind. Das ganze via Ethernet mit einem Browser nach Wahl. Das Beispiel ist auf der NET7026 Produktseite zu finden (www.mct.de/product/net7026.html). Das Programm ist (hopefully) gut dokumentiert (in Englisch), hier ein paar Anmerkungen:



iPad als Bedieneinheit

Das Programm kann mit der Demo-Version des ECO-C-arm Compilers übersetzt werden (zu finden auf www.mct.de unter Download). Vor dem Compilieren müssen die IP-Adresse (für das NET7026), die Netzwerkmaske und die Gateway-Adresse (Router) angepasst werden. Dann einfach im Adressfeld des Browsers die NET7026-IP-Adresse eingeben.

Wie funktioniert's? Die Software stellt einfache Funktionen - `net_open()`, `net_read()` und `net_write()` - ähnlich den File-Operationen zur Verfügung. Der Server liefert dem Browser eine HTML-Seite, in die der aktuelle ADC-Wert eingeblendet ist. In einem Formular-Feld kann man dann einen neuen DAC-Wert eingeben. Als Kommunikationsprotokoll wird HTTP benutzt.

Wie gesagt - ein Beispiel. Man könnte noch viele Verbesserungen vornehmen, z.B. die HTML-Seite optisch aufpeppen, Client-spezifische Besonderheiten (z.B. die Bildschirmgröße) berücksichtigen u.dergl. mehr.

Zugabe

Das hat jetzt wirklich nichts mehr mit Embedded Control zu tun, sondern soll nur die Flexibilität einer Netzwerk-fähigen Hardware wie dem NET7026 demonstrieren. Das kann nämlich nebenbei auch noch Web-Radio spielen...

„Nebenbei“ ist etwas übertrieben, aber mit „einem Handgriff“ (soll heißen mit einer anderen Firmware) wird aus dem Server ein Client, der einen mp3-Stream aus dem Netz abspielt. Die Software schickt der Autor auf Wunsch per E-Mail zu.