CAN

Fysiskt medium

CAN kommunicerar via två trådar kopplade i en busstopologi. Alla noder ansluts parallellt på denna buss, och i varje ände av bussen finns en resistiv terminering. Data skickas seriellt med en förutbestämd bitrate som alla noder måste hålla sig till. Beroende på vilken bitrate som väljs fås olika begränsningar på maximalt tillåten kabellängd.

Bitrate	Maximal kabellängd
1Mbit/s	25m
800kbit/s	50m
500kbit/s	100m
250kbit/s	250m
125kbit/s	500m
50kbit/s	1km
20kbit/s	2.5km
10kbis/s	5km

Tabell 1 - Kabellängder för olika bitrates.

Kommunikation

CAN är frame-baserat, d.v.s. data paketeras in i frames innan den skickas. Varje frame har en identifierare, ett antal databytes (max 8), samt en CRC-checksumma. En vanlig *standard*-frame har en identifierare på 11bitar, medan en s.k. *extended* frame har 29bitars-identifierare.

CAN-frames skickas alltid till alla noder på bussen (*broadcasting*). Identifieraren talar endast om vad en frame innehåller, inte vilken nod som skickat den eller vilken nod som skall ta emot den. Om två noder försöker skicka samtidigt på bussen sker arbitrering, och den frame med lägst identifierare blir den som kommer fram först (dvs. låga identifierare prioriteras).

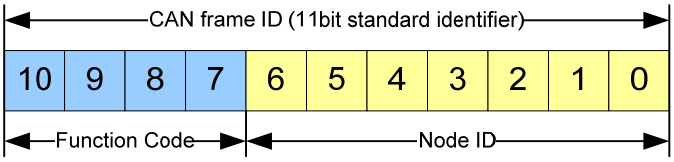
CANopen

Introduktion

CANopen är ett högnivå-protokoll för CAN-bussen. Protokollet har många användningsområden, men vanliga tillämpningar inkluderar automation i industrier och byggnader, distribuerade reglersystem för maskiner, automation av medicinsk utrustning m.m. Standarden underhålls av organisationen CiA (*CAN in Automation*) [http://www.can-cia.org/canopen/].

Kommunikation

CANopen kommunicerar via Communication OBjects (*COBs*). Dessa är i praktiken vanliga standard-CAN-frames, dvs. frames med 11bit-identifierare. De fyra mest signifikanta bitarna i detta ID (bit7-bit10) är reserverade för en funktionskod som talar om för CANopen vad det är för typ av COB som skickas. Resterande sju bitar (bit0-bit6) används som nod-ID. Figur 1 illustrerar detta.



Figur 1 – Användandet av CAN-identifierare i CANopen.

Tack vare denna uppdelning får vissa typer av COBs alltid högre prioritet än andra. Detta utnyttjar man i CANopen genom att välja låga funktionskoder till högprioriterade COB-typer. Om två COBs av samma typ (dvs. med samma funktionskod) skickas från två olika noder kommer också noden med lägst nod-ID att få högst prioritet.

En sidoeffekt av att endast ha sju bitar över till nod-ID är att man maximalt kan ha 127 olika noder i ett CANopen-system. Det är inom specen ej heller tillåtet att använda extended CAN-frames (med 29bit-identifierare) för att komma runt detta.

Object Dictionary (OD)

Kärnan i CANopen är användandet av ett s.k. *Object Dictionary* (OD). Ett OD kan ses som ett lagringsutrymme för all data tillhörande en viss nod. Sådana data kan vara allt från konfigurationsparametrar, statusflaggor eller identifierare, till processdata såsom mätvärden eller börvärden. Även funktioner eller procedurer i noden kan ses som data i OD.

Man säger att OD innehåller en mängd objekt (*entries*). Varje objekt har en ”adress” bestående av ett 16bit-index och ett 8bit-subindex. Värdet hos varje objekt kan vara av variabel längd. Exempelvis kan både 8bit-, 16bit-, och 32bit-variabler av olika typer lagras i ett objekt, men även långa strängar eller t.o.m. komplexa datastrukturer. Objekt bör alltså inte ses som minnesplatser, utan snarare som pekare till data som finns i noden.

I varje nods OD finns ett antal objekt som är fördefinierade och obligatoriska enligt CANopen-standarden. Exempel på sådana objekt är nodens typ, dess ID, dess namn, samt några parametrar relaterade till hur noden kommunicerar på CANopen. Det finns även ett s.k. ”error-register”, som är ett obligatoriskt objekt som lagrar eventuella felkoder från noden. Figur 2 visar några av de obligatoriska objekten i ett OD, tillsammans med beskrivning.

Index	SubIdx	Type	Description
1000h	0	UNSIGNED32	Device Type Information
1001h	0	UNSIGNED8	Error Register
1018h			Identity Object
	0	UNSIGNED8	= 4 (Number of sub-index entries)
	1	UNSIGNED32	Vendor ID
	2	UNSIGNED32	Product Code
	3	UNSIGNED32	Revision Number
	4	UNSIGNED32	Serial Number

Figur 2 - Beskrivning av några objekt i OD. [1]

Mekanismer

Eftersom all form av data i ett CANopen-system finns utspritt i de olika nodernas Object Dictionaries så handlar mycket av kommunikationen som sker på bussen i huvudsak om att komma åt (d.v.s. läsa från- och skriva till-) nodernas OD. För detta finns det två mekanismer; *PDO* och *SDO*. En annan viktig mekanism är *NMT*, som används för att underhålla nätverket och bevaka systemet. Mekanismerna beskrivs nedan.

Process Data Objects (PDO)

PDO-mekanismen används typiskt för att överföra processdata mellan noder, exempelvis mätdata från sensorer, eller styrsignaler till aktuatorer. Själva mekanismen kan liknas vid broadcasting eftersom noder själva skickar ut PDO:er (antingen periodiskt eller triggat av diverse händelser). Det är sedan upp till övriga noder att välja vilka PDO:er de vill lyssna på.

En PDO identifieras helt med hjälp av funktionskoden i CAN-identifieraren, så alla åtta databytes kan utnyttjas av applikationsdata. Det finns inget inbyggt stöd för segmenterade överföringar i PRO-mekanismen, så om man behöver skicka data större än 8 bytes m.h.a. PDO får man antingen sprida datan över flera olika PDO:er, eller sköta multiplexningen i sin applikation.

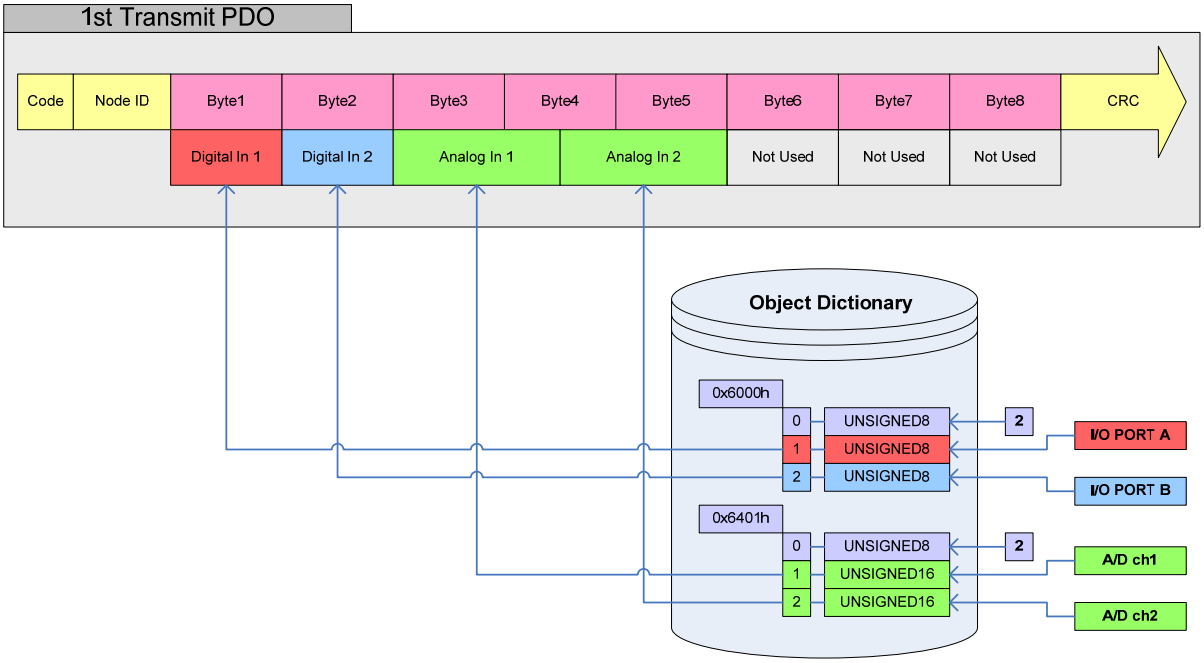
PDO-meddelanden får automatiskt hög prioritet på bussen i CANopen. De är därför väldigt lämpade till överföring av kritisk realtidsdata, speciellt om det är stora datamängder som ska överföras, eftersom mekanismen inte har någon protokolloverhead i form av reserverade databytes.

I noder baserade på enkla mikrokontrollers har man typiskt en statisk konfiguration som bestämmer vad varje PDO innehåller, d.v.s. man mappar statiskt in olika data från OD i nodens PDO:er. Flera OD-objekt kan mappas in i samma PDO i mån av utrymme. Själva mappningen sker bitvis, så man kan själv ”stycka upp” sina objekt och sprida ut dem på valfritt vis i sina PDO:er. Exempelvis kanske man använder en 16bit-variabel för att lagra ett 12bitars-värde. Då kan man enkelt packa in endast de 12 intressanta bitarna i PDO istället för att skicka hela 16bitars-objektet.

Själva PDO-mappningen är, liksom processdatan, också lagrad i OD. I system med mer komplexa noder kan man därför tillåta dynamisk mappning genom att låta noder ändra i dessa mappningsobjekt under drift. I Figur 3 och Figur 4 visas ett exempel på en PDO-mappning. Två digitala 8bits-indata och två analoga 12bits-indata mappas till transmit PDO1. Processdatan lagras i objekt 0x6000h och 0x6401h, och mappningen för TPDO1 lagras i 0x1A00h.

Index	SubIdx	Type	Description
1A00h			1st Transmit PDO - Mapping
	0	UNSIGNED8	= 4 (Number of supported map entries)
	1	UNSIGNED32	= 6000 01 08h (Idx 6000, SubIdx 1, 8 bit)
	2	UNSIGNED32	= 6000 02 08h (Idx 6000, SubIdx 2, 8 bit)
	3	UNSIGNED32	= 6401 01 0Ch (Idx 6401, SubIdx 1, 12 bit)
	4	UNSIGNED32	= 6401 02 0Ch (Idx 6401, SubIdx 2, 12 bit)
6000h			Process data, digital inputs
	0	UNSIGNED8	= 2 (Number of sub-index entries)
	1	UNSIGNED8	8-bit digital input
	2	UNSIGNED8	8-bit digital input
6401h			Process data, analog inputs
	0	UNSIGNED8	= 2 (Number of sub-index entries)
	1	UNSIGNED16	12-bit analog input
	2	UNSIGNED16	12-bit analog input

Figur 3 - Exempel på PDO-mappning i OD. [1]



Figur 4 - Illustration av PDO-mappningens effekt.

PDO:er saknar helt tillstånd, så informationen som överförs i varje PDO är väldefinierad av objekt-mappningen och är därför helt oberoende av föregående PDO:er. Detta gör att det inte behövs några komplexa tillståndsmaskiner för att sända och ta emot data med hjälp av PDO-mekanismen.

Service Data Objects (SDO)

SDO:er används till dataöverföringar som inte är tidskritiska, men som istället kräver hög tillförlitlighet. Själva mekanismen är av client/server-typ, till skillnad från PDO:er som broadcastas. Varje CANopen-nod har minst en SDO-server som man kan ansluta till från vilken annan nod som helst genom att använda en SDO-klient.

Från en SDO-klient kan man komma åt server-nodens OD genom att använda diverse tjänster. Exempelvis använder man *download*-tjänsten för att skriva till OD, och *upload*-tjänsten för att läsa från OD. All SDO-kommunikation består dels av själva klientanropen, och dels av en bekräftelse tillbaka från servern efter varje kommando. Detta gör att SDO blir mycket mer tillförlitligt än PDO eftersom servern hela tiden måste svara att allt är OK innan klienten kan fortsätta. Om servern upptäcker att något gått snett finns det ett antal olika felkoder som den istället kan skicka tillbaka till klienten för att meddela detta och avsluta aktuell server/client-session.

SDO-mekanismen har inbyggt stöd för segmenterad överföring av data. Om man vill överföra stora mängder data (exempelvis komplexa objekt i form av datastrukturer) fördelas datan automatiskt över ett flertal SDO-meddelanden.

SDO-meddelanden har låg prioritet på CAN-bussen, så man kan utan problem ha ett antal SDO-sessioner aktiva i ”bakgrunden” samtidigt som man skickar tidskritisk realtidsdata via PDO.

Det krävs viss logik i form av tillståndsmaskiner i noderna för att hantera SDO-mekanismens client/server-sessioner.

Network Management (NMT)

För att underhålla nätverket finns det dedikerade tjänster för network management. Dessa tjänster kan användas för att starta, stoppa eller starta om noder. De kan också användas för att bevaka noder. Varje nod sänder vid uppstart ut ett boot-up-meddelande, och under drift sänds heartbeat-meddelanden ut med en viss periodicitet.

Tjänsterna för att starta, stoppa och starta om noder kan endast användas från en unik NMT-master, men alla noder kan bevaka boot-up och heartbeat från övriga noder och därmed upptäcka om en nod skulle sluta fungera eller starta om vid något tillfälle.

Device Profiles

För att öka graden av kompatibilitet, samt för att effektivisera utvecklandet av nya CANopen-noder finns det fördefinierade *device profiles* som man baserar sina noder på. En device profile beskriver den grundläggande funktionaliteten hos en klass av noder. Till exempel så definierar profilen DS401 (*CANopen Device Profile for I/O Modules*) en klass av noder som har ett antal digitala- och/eller analoga in- och utgångar. Denna generella klass är lämplig som grund för alla typer av noder vars uppgifter på något sätt är I/O-relaterade. De uppenbara tillämpningarna är sensor- och aktuator-noder.

Object Dictionary har en area reserverad för alla objekt definierade av device profile. Tack vare dessa fördefinierade objekt kan många noder utvecklas utan att man behöver skapa några nya OD-objekt alls.

Electronic Data Sheets

Det finns många verktyg för att analysera och konfigurera CANopen-system. Tack vare de obligatoriska objekten i OD kan sådana verktyg enkelt leta reda på alla noder i systemet och hämta grundläggande information om dessa. Genom att dessutom ta reda på en nods device profile kan verktyget få ytterligare information om vilken typ av processdata som återfinns i OD. De nod-specifika objekten kan dock ett verktyg inte ta reda helt på egen hand utan någon form av ytterligare

information. Sådan information kan man ge i form av *Electronic Data Sheets*, s.k. EDS-filer. Till varje CANopen-nod skall det finnas en tillhörande EDS-fil, som i princip innehåller hela Object Dictionary i form av text.