Özgun Mirtchev & Rojan Dinc
ÖREBRO UNIVERSITET
NATURVETENSKAP OCH TEKNIK

Controller Area Network (CAN)

Denna laboration gjordes i syfte i att få en introduktion till Controller Area Network(CAN), med användning av två CAN-noder och en CAN-buss för två laborationsgrupper.

Kurs: Datorkommunikation och nät (DT2017-0200/DT2022-0222)

Härmed försäkrar jag/vi att jag/vi utan att ha erhållit eller lämnat någon hjälp utfört detta arbete.

Datum: 2016-01-22

Underskrift:

Özgun Mirtchev

Namn: *Özgun Mirtchev*
Personnr: *920321-2379*
E-post: *ozzieee@gmail.com*
Program: *Dataingenjörsprogrammet*

Rojan Dinc

Namn: *Rojan Dinc*
Personnr: *940421 - 8451*
E-post: *rojand94@gmail.com*
Program: *Dataingenjörsprogrammet*

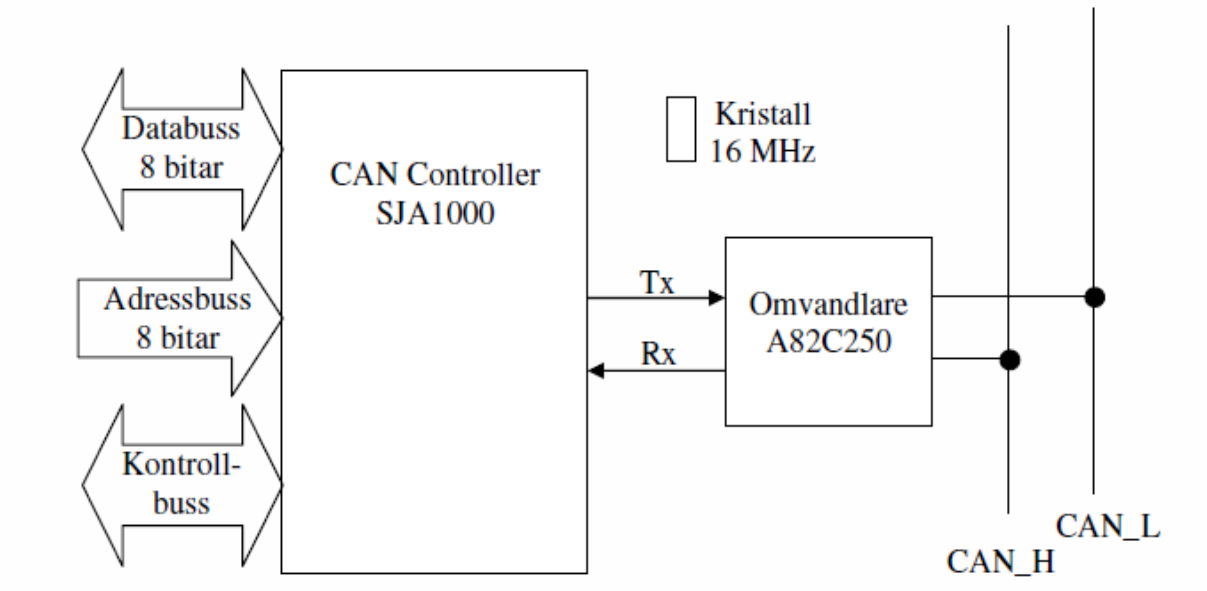
Lärarens anteckningar

Innehållsförteckning

Bakgrund

I denna laboration skulle ett applikationsprotokoll/program skrivas för kommunicerande noder via en CAN-buss. Varje nod skulle tilldelas till vardera två grupper för att samarbeta mellan varandra. Applikationsprogram är för inmatning och utmatning på terminalfönster och för applikationsprotokoll är det överföring av meddelanden på CAN-bussen mellan noderna.

Samtliga av den använda utrustningen är från Philips. Varje nod har en CAN-controller, den som användes för denna laboration är av typen **SJA1000**, och en omvandlare av typen **A82C250**. Tillståndet i SJA1000 kommer att vara Basic CAN för denna laboration, förutom det andra tillståndet PeliCan. Dessutom användes en MicroController Unit (MCU) av typen AVR ATmega8515, med ett tillhörande flashminne för att ladda ned programmet genom en kabel med en programmerare, STK200. Nedan syns en bild på en CAN-Controller.

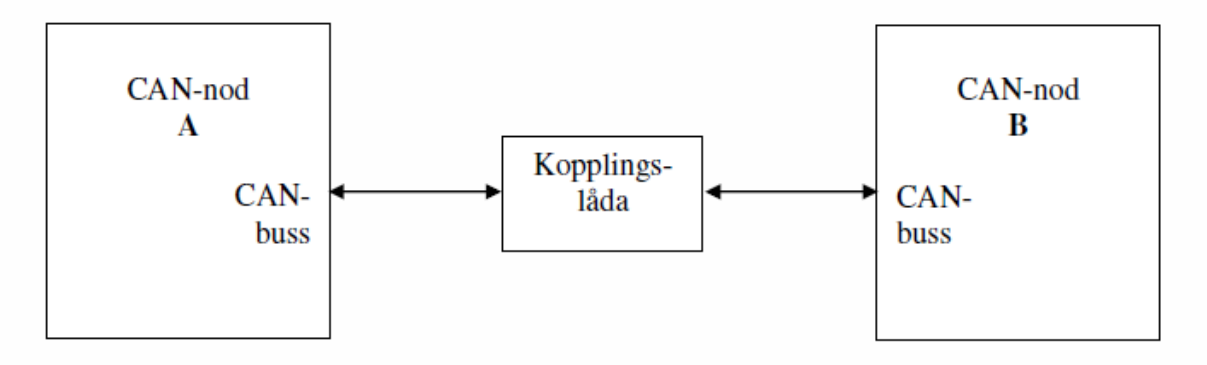


Programmet Eclipse användes för kodredigering och nedladdning av programmet till MCU:n. Koderna för programmet hittas i Bilaga-sektionen med diverse kommentarer, längst ned i detta dokument.

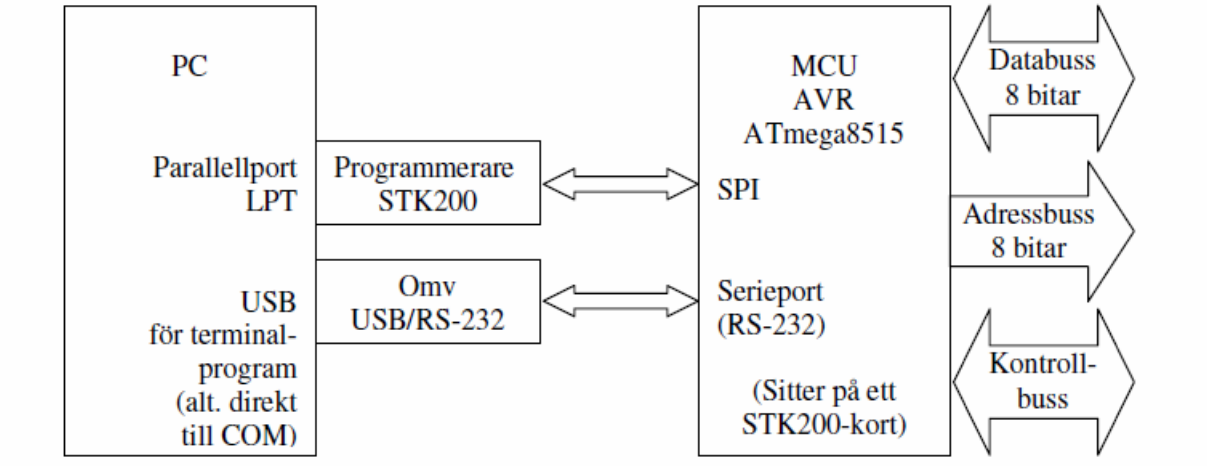
Resultat

Uppgift A

CAN-noderna kopplades ihop enligt instruktioner med hjälp av D-sub-kontakter (9-poliga) med en kopplingslåda emellan för att senare kunna kontrollera strömtillförsel till de olika noderna. Nod A är den nod som användes av laboranterna och Nod B är den nod som användes av medlaboranterna under laborationen.



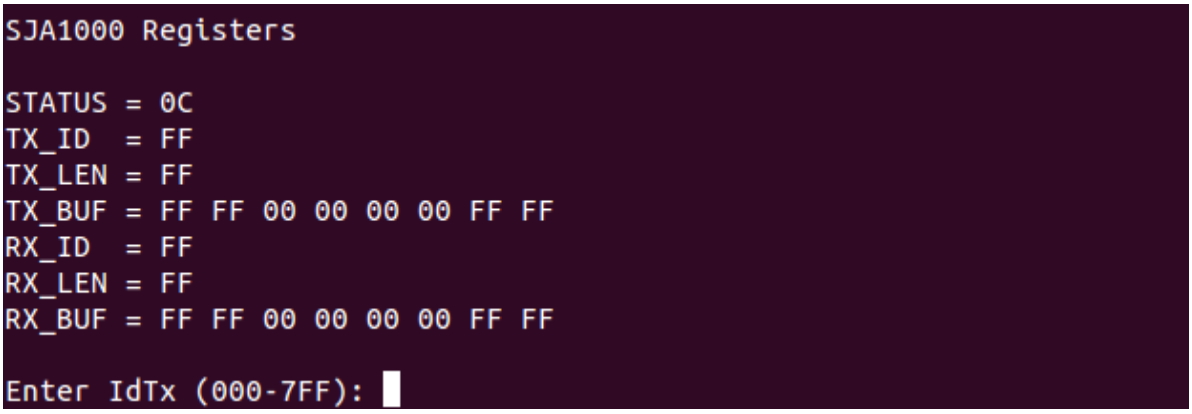
En nod består av en MCU och en CAN – Controller (SJA1000) som är anslutna tillsammans med bandkablar till STK-portarna.



Programmet Kermit användes för att utföra kommunikationen mellan noderna och skicka meddelanden. RS-232 kopplades till USB-port 0 och därför användes respektive kommandon i terminalen för att koppla upp sig till Kermit:

```
~$ kermit
C-kermit> set line /dev/ttyUSB0
C-kermit> set carrier-watch off
C-kermit> set speed 9600
C-kermit> c
```

Vid uppkoppling visades meddelandet på bilden nedan med status 0C, som indikerar att uppkoppling till STK-kortet är OK.



Figur 1 – Status 0C

Uppgift B

Programmet CANJSA användes för skriva ut meddelanden på terminalen och visa meddelanden på LCD. För att öppna en port att kommunicera med användes identifieraren 005, vilket skrevs in i ”Enter IdTx (000-7FF): ”, som skrev ut vilka alternativ en hade. I detta fall kan man se i bilden nedan, att det enda alternativet är att skicka meddelanden till den kopplade noden.

```
Enter IdTx (000-7FF): 005

Using IdTx = 0005

From now and on you have two choices:
1. Press [Enter] to write a message
2. Don't press any key to wait for an incoming message
```

Figur 2 – Meddelandeinskrivning i kermit

Efter att grannoden (Nod B) hade kopplat upp sig med identifieraren 123, syntes det på Nod A STK-kortets display med meddelandet ”RX 0123 ON-LINE”. RX står för Receive, alltså den identifierare som har ankommit med ett meddelande.

Under en provkörning, syns det i bilden nedan att den identifierare som hade 123 (Nod B), skickade ett meddelande ”Hej”, till Nod A. Detta syntes i Nod A:s LCD-skärm:

Figur 3 – LCD-meddelande på STK-kortet

Uppgift C

Vid ett test mellan noderna med CAN-bussen, för att kontrollera att den även fungerar vid kortslutning, kom gruppen fram till dessa resultat:

Kortslutningsprov	**Pol: Matningspänning (5V)**	**Pol: Signaljord**	**Med varandra**
CAN_L	OK	OK	FEL
CAN_H	OK	FEL	
Båda signalerna till samma pol	FEL	FEL	
Båda signalerna till olika poler	CAN_L till matningsspänning CAN_H till signaljord	FEL	
Båda signalerna till olika poler	CAN_L till signaljord CAN_H till matningsspänning	FEL	

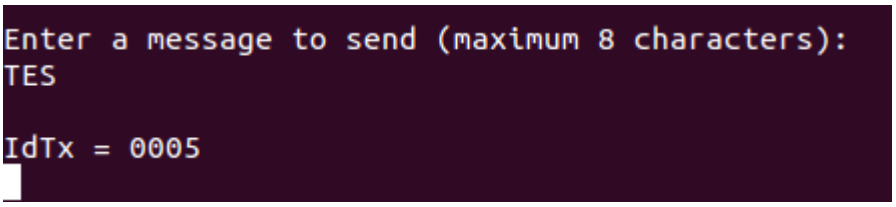
De som är markerade med OK, klarade av att skicka meddelanden även fast det var kortslutning till polerna. Dessvärre gick det inte så bra med de andra. En sak som blev tydligt var att efter en kortslutning var aktiv och ett meddelande skickades, kom det inte fram direkt men eftersom det lagrades på bufferten, skickades den senare när kortslutningen blev inaktiv. Detta orsakade att meddelandet inte syntes på LCD-skärmen förrän kortslutningen slutade.

Uppgift D

D1.

För att skriva ut Nod A:s identifierare (laboranternas egen nod) i terminalen, behövdes förändringar ske i canjsa.c programmet. Kod som lades till i main-funktionen av programmet befinns i bilaga 1.

För att kunna skriva ut identifieraren användes prints för strängar och printh för hexadecimala tal. IdTx håller ett heltal, som deklarerats tidigare i main-funktionen. Eftersom printh bara kan skriva ut två tecken per heltal, användes två variabler för att hålla hexadecimal-talen. Printh skrevs därför ut två gånger för varje variabel.

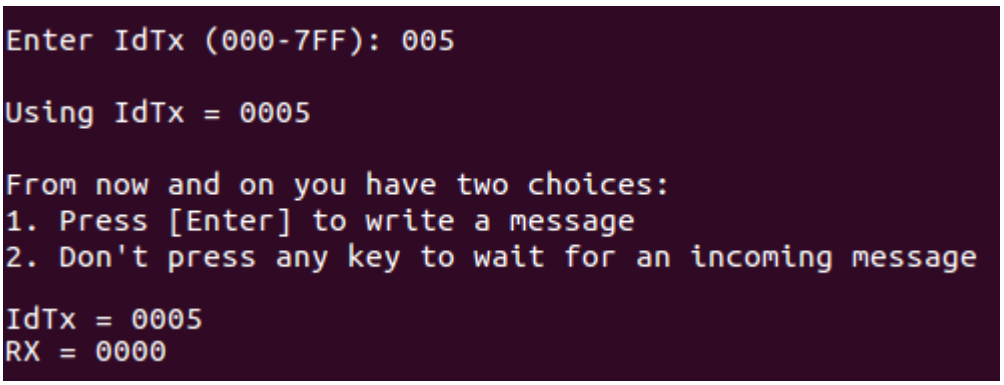


Figur 4 – IdTx-utskrift

I bilden ovan ser man meddelande som har skickats till motpartens nod och utskriften av identifieraren 005, när ett meddelande skickades tillbaka.

D2.

För att skriva ut IdRx också, som är avsändarens identifierare, behövdes den först beräknas från registervärdena, och sedan skrivas in i IdRx. När den väl skrevs in i IdRx, utfördes samma steg som föregående uppgift där variabelvärdet delades upp i två variabler för utskrift. Koden kan hittas i bilaga 2.



Figur 5 – IdTx- och IdRx-utskrift

I bilden ovan ser man att IdTx (0005) och IdRx (0000) skrivs ut efter mottaget meddelande. Däremot syns meddelandet inte i terminalen då koden för den funktionen inte har implementerats i programmet ännu.

D3.

För att skriva ut meddelanden i terminalen måste bufferten läsas in i en predefinierad array, `char R[9]`, från eventuella tecken. Då det maximalt kan finns visas åtta tecken så läses bara bufferten upp till plats nummer 7 in i arrayen med hjälp av `SJA_Read(CAN_RX_BUF0 + i)` som sker inuti i en for-loop. Eftersom meddelanden kommer att skrivas ut med prints() måste det sista tecknet vara en `'\0'` för att indikera att det är slutet på strängen. Detta åtgärdas simpelt med en `R[8] = '\0';` efter for-loopen. Se bilaga 3 för mer detalj av koden. Nedan visas en provkörning av det uppdaterade programmet med utskrifter av mottagen meddelade (Notera att motpartens nod har identifierare 002, istället för 123, från och med nu):

```
Enter IdTx (000-7FF): 005

Using IdTx = 0005

From now and on you have two choices:
1. Press [Enter] to write a message
2. Don't press any key to wait for an incoming message

Medd: ASDASD**
IdTx = 0005
RX = 0002
```

Figur 6 – Utskrift av mottagen meddelande

D4.

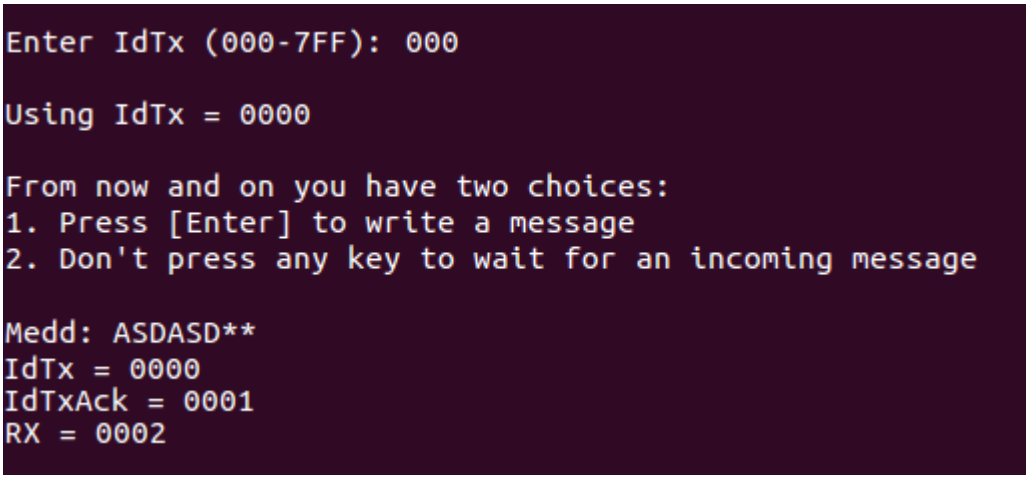
För att bygga ett acceptansfilter, gäller det att man skriver kod med villkor (if-sats). I detta fall skrevs ett acceptansfilter för IdRx = 002, motpartens identifierare. I fallet då villkoret är sant, ska meddelandet skrivas ut, annars skrivs ett ”Fel ID” meddelande ut i terminalen (mest för test ändamål). I bilden nedan kan man se ett exempel på körning. Kodens primitiva funktion befinns i bilaga 4, men för att se helheten av koden se istället bilaga 7.

```
Enter IdTx (000-7FF): 005

Using IdTx = 0005

From now and on you have two choices:
1. Press [Enter] to write a message
2. Don't press any key to wait for an incoming message

Medd: ASDASD**
IdTx = 0005
RX = 0002

Fel ID
IdTx = 0005
RX = 0123
```

Figur 7 – Exempel på acceptansfilter

D5.

Ett ACK-värde ska skrivas ut tillsammans med de andra utskrifterna som har implementerats tills nu. Koden är inte mycket märkbart från koden som användes vid uppgift D1 och D2. Enda skillnaden är att IdTx adderas med 1 för att öka värdet. Om IdTx = 000, blir ACK = 001. Se bilaga 5 för kod och bilden nedan för ett exempel där motsvarande nod (B) skickar till ett meddelande till nod A. IdTxAck skrivs ut under IdTx, med ett ökat värde av ett.



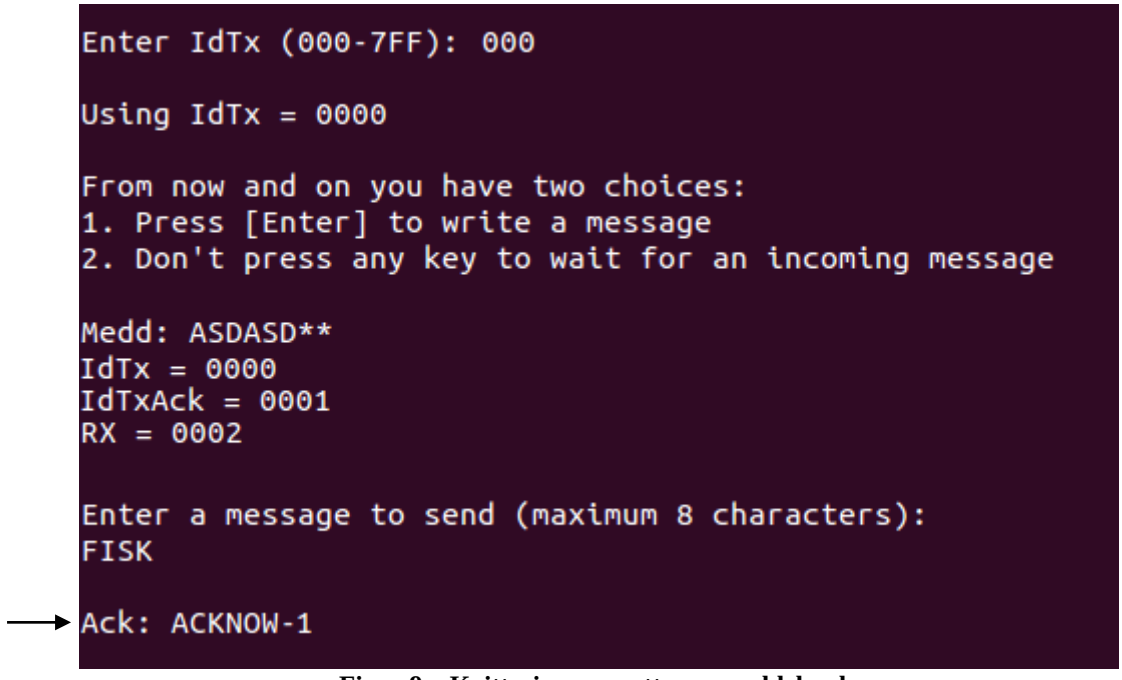
Figur 8 - IdTxAck

D6.

I denna uppgift skulle varje mottagen meddelande från en nod kvitteras med ett ACK-meddelande till den andra noden. Som tidigare nämnt tillhör Nod A laboranterna och Nod B tillhör den andra gruppen av laboranter. Meddelandet som ska kvitteras är ”ACKNOW –n” där n är ett tal som inkrementeras för varje nytt meddelande, men roteras inom 0-9.

Kvitteringsmeddelandet skrevs in i transmit-buffern av CAN Controllern, genom en for-loop (Går även att skriva in varje tecken i varje plats av bufferten manuellt). För att utföra detta användes funktionen `SJA_Write()`, med första argumentet `CAN_TX_BUF0` och ett andra argument som är ett char-tecken. Som t.ex. `SJA_Write(CAN_TX_BUF0, ”A”);` för att skriva in tecknet ”A” i första platsen av bufferten. När allt hade skrivits in, inklusive det inkrementerande talet n, skulle allting skickas till nod B för utskrift i motsvarande terminal. Detta skulle ske efter att ett meddelande hade inkommit från motsvarande nod, i detta fall nod B, för att meddela att det skickade meddelandet har inkommit till respektive nod. Vice versa ifall nod A skickar ett meddelande till nod B, så får nod A en kvittering från nod B. Genom filtret som kollar ifall inkommande IdRx är en ack-identifierare eller inte så skrevs antingen ett meddelande ut eller ack-meddelandet ut.

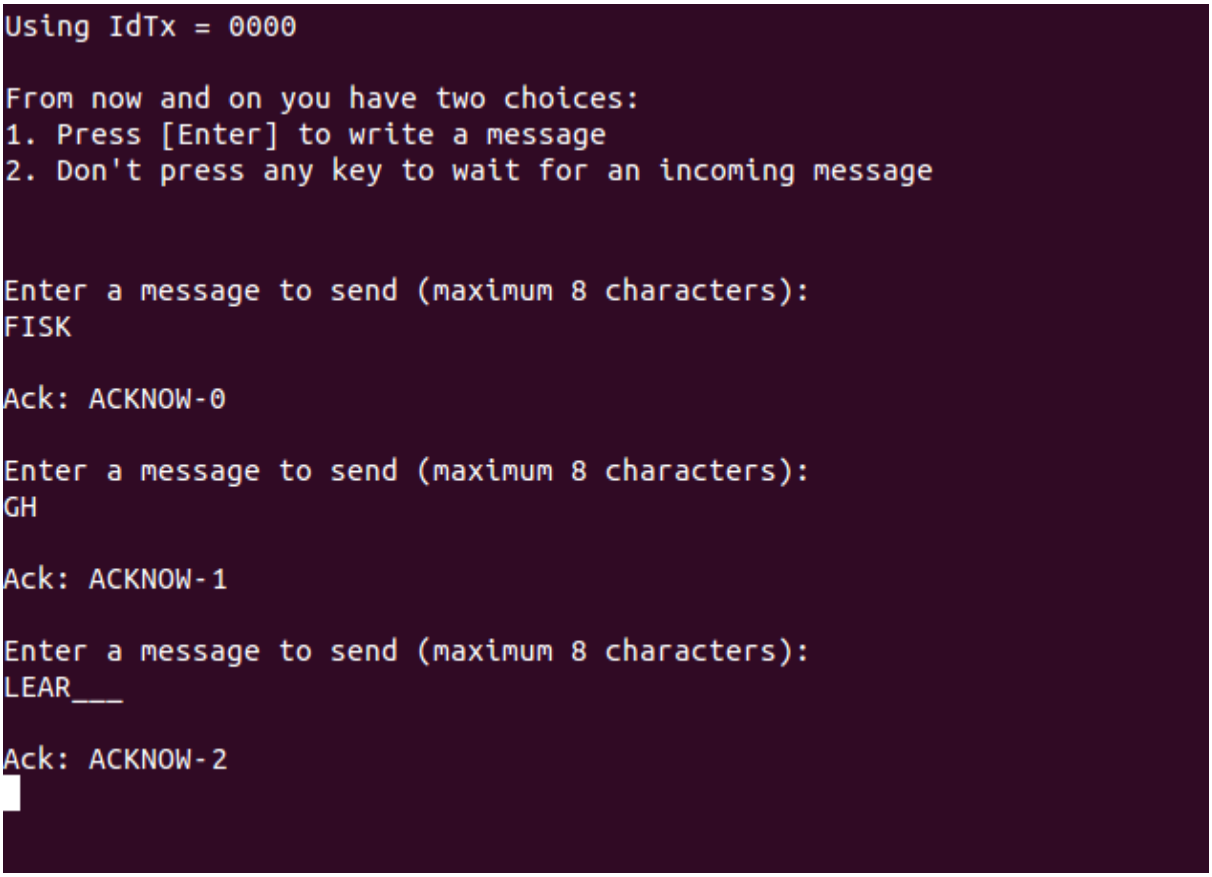
Nedanstående bild ger en demonstration av processen. Först skickas ett meddelande från nod B till nod A. Inkommande IdRx är 002 och enligt if-satsen så accepteras det och meddelandet skrivs ut tillsammans med identifierar-värden och ett ACK skickas som kommer upp på nod B:s terminal-skärm. I andra steget, skickas ett meddelande från nod A till nod B, i det ögonblicket får nod B meddelandet och skickar tillbaka ACK-meddelandet med IdRx-identifierare 003, för att indikera en kvittering och if-satsen kollar om det är en meddelandeidentifierare eller om det är en kvitteringsidentifierare och skriver ut ACK-meddelandet eftersom villkoret för kvitteringsidentifieraren har uppfyllts.
Koden för denna uppgift hittas i bilaga 6.



Figur 9 – Kvittering av mottagen meddelande

D7.

Slutligen kördes en genomgående testning av programmet som visade att kvitteringen endast räknades upp till 9 och sedan började om vid 0 och att filtret fungerade som det ska. Vid felaktig identifierare skedde bara en buffer-rensning då all kod var inkapslad i en if-sats som ej exekverades då villkoret inte var uppfylld. Dessvärre finns inga större bildbevis för detta förutom nedanstående bild och därför får läsaren antingen tro på författarens ord, gå igenom hela koden i bilaga 7 eller vid anmodan få en provkörning av programmet.



Figur 10 – Kvitteringstester

Özgun Mirtchev & Rojan Dinc
ÖREBRO UNIVERSITET
NATURVETENSKAP OCH TEKNIK

Bilagor

Bilaga 1 – Uppgift D1

```
unsigned int x1, x2;

x1 = IdTx / 256;
x2 = IdTx - x1 * 256;

prints("IdTx = ");
printh(x1);
printh(x2);
prints("\n");
```

Bilaga 2 – Uppgift D2

```
unsigned int IdRx = 0;
unsigned char tmp1 = 0, tmp2 = 0;

tmp1 = SJA_Read(CAN_RX_ID);
tmp2 = SJA_Read(CAN_RX_LEN);

if (SJA_Read(CAN_STATUS) & RBS)
{
            IdRx = ((unsigned int)tmp1 << 3) + ((tmp2 >> 5) & 0x07);
}

x1 = IdRx / 256;
x2 = IdRx - x1 * 256;

prints("RX = ");
printh(x1);
printh(x2);
prints("\n");
```

Bilaga 3 – Uppgift D3

```
int i = 0;
char R[9];

for(i; i < 8; i++)
{
            R[i] = SJA_Read(CAN_RX_BUF0 + i);
}

R[8] = '\0';

prints("Medd: ");
prints(R);
prints("\n");
```

Bilaga 4 – Uppgift D4

Notera att denna kod inte är komplett då det mesta av koden har satts ihop i senare uppgifter. För att se en uppdaterad version se istället bilaga 7 för uppgift D7 där all tidigare kod har skrivits in i if-satsen.

```
if (IdRx == 0x002)
{
            // Kod från bilaga 1,2,3,5,6,7 sker här
}
else
{
             prints("Fel ID");

             //Kod från bilaga 1,2

}
```

Bilaga 5 – Uppgift D5

```
unsigned int IdTxAck = IdTx + 1; //Nod A

x1 = IdTxAck / 256;
x2 = IdTxAck - x1 * 256;

prints("IdTxAck = ");
printh(x1);
printh(x2);
prints("\n");
```

Bilaga 6 – Uppgift D6

```
SJA_Write(CAN_TX_ID, IdTxAck >> 3);
SJA_Write(CAN_TX_LEN, (IdTxAck << 5) | 8);

int i;
char ackmessage = ['ACKNOW-'];

for (i = 0; i <= 6; i++)
{
SJA_Write(CAN_TX_BUF0 + i, ackmessage[i]);
}

// Skriver ut ACKNOW nummer i slutet av bufferten
// j har deklarerats innan while-loopen
SJA_Write(CAN_TX_BUF7, j + 48);
j = (j + 1) % 10;

SJA_Write(CAN_CMD, TXREQ);
```

Özgun Mirtchev & Rojan Dinc
ÖREBRO UNIVERSITET
NATURVETENSKAP OCH TEKNIK

Bilaga 7 – Uppgift D7 (Hela koden för både Nod A och Nod B)

```
int main(void)
{
		unsigned char btr0 = 0x03, btr1 = 0x1C;
		unsigned int IdTx;
		int j = 0;

		/* Initieringar och registeravläsning */
		USART_Init();			// Initierar USART
		USART_Flush();			// Rensar RS232-bufferten
		LCD_Init();			// Initierar LCD
		DDRA = 0xFF;			// Sätter Port A på STK200-kortet för bussar till/från
CAN-kontroller
		DDRB = 0xFF;			// Sätter Port B på STK200-kortet för LED:s (data output,
obs flera olika kontakttyper)

				// Sätter Port C på STK200-kortet för LCD (data output)
		DDRD = 0x00;			// Sätter Port D på STK200-kortet för bussar till/från
CAN-kontroller

				// Sätter Port E på STK200-kortet för NC
		SJA_Init(&btr0, &btr1);		// Initierar CAN-kontrollerna SJA1000 till 125 kbit
		SJA_Dump();				// Visar CAN-registren på terminalen via RS232-
porten

		// Status bör vara 0C, om inte starta om med Powerknappen

		// (Observera att några SJA1000-register inte kan avläsas i körläget)


		/* Visar text på LCD */
		LCD_Clean(LINE1);		// Rensar LCD-rad 1 och gå till början av den
		LCD_StrOut(MSG1);		// Matar ut titeln på LCD-rad 1
		LCD_Clean(LINE2);		// Rensar LCD-rad 2 och gå till början av den


				/* Läser in IdTx och skickar en ram på CAN-bussen */
		IdTx = USART_ReceiveIdTx();	// Läser in Tx-identifieraren från terminalens
tangentbord
		CAN_StrOut(IdTx, &MSG2[0]);	// Skickar dataram på CAN med texten "xxx ON-LINE ", där
xxx = IdTx


				/* Evighetsloop */
		while (1)
		{
			if (SJA_Read(CAN_STATUS) & RBS)
			// Kontrollerar om ram har kommit in till CAN-kontrollerns Rx
			{
					CAN_Check();
			// Hämtar in IdRx och meddelande från CAN-kontroller och visar på LCD


					/* Övning D:2 - Beräknar IdRx utifrån registervärden */

					unsigned int IdRx = 0;
					unsigned char tmp1 = 0, tmp2 = 0;

					tmp1 = SJA_Read(CAN_RX_ID);
					tmp2 = SJA_Read(CAN_RX_LEN);

					if (SJA_Read(CAN_STATUS) & RBS)
					{
					IdRx = ((unsigned int)tmp1 << 3) + ((tmp2 >> 5) & 0x07);
					}
```

```
                    /* Övning D:4 - Filter för inkommande identifierare.
                    Alla andra uppgifter är inuti denna if-sats */

                    /* IdRx-filter för Nod A står först, Nod B under */
                    if(IdRx == 0x002 || IdRx == 0x007 || IdRx == 0x00C)
                    //if(IdRx == 0x000 || IdRx == 0x005 || IdRx == 0x00A)
                    {

                    /* Övning D:3 - Utskrift av inkommande meddelanden */

                                int i = 0;
                                char R[9];

                                for(i; i < 8; i++)
                                {
                                R[i] = SJA_Read(CAN_RX_BUF0 + i);
                                }

                                R[8] = '\0';

                                prints("Medd: ");
                                prints(R);
                                prints("\n");


            /* Övning D:1 - Beräknar IdTx och skriver ut det på terminalen */

                                unsigned int x1, x2;

                                x1 = IdTx / 256;
                                x2 = IdTx - x1 * 256;

                                prints("IdTx = ");
                                printh(x1);
                                printh(x2);
                                prints("\n");


            /* Övning D:5 - Beräknar IdTxAck(IdTx+1) och skriver ut på terminalen */

                                /* IdTx = 0000 för Nod A
                                IdTxAck blir 0001 och skickas till Nod B */
                                /* IdTx = 0001 för Nod B
                                IdTxAck blir 0002 och skickas till Nod A*/
                                unsigned int IdTxAck = IdTx + 1;

                                x1 = IdTxAck / 256;
                                x2 = IdTxAck - x1 * 256;

                                prints("IdTxAck = ");
                                printh(x1);
                                printh(x2);
                                prints("\n");


            /* Övningen D:2 - Beräknar IdRx och skriver ut på terminalen */

                                x1 = IdRx / 256;
                                x2 = IdRx - x1 * 256;

                                prints("RX = ");
                                printh(x1);
                                printh(x2);
                                prints("\n");
```

```
                                        /* Övning D:6 - Skickar över en ACKNOW till motsvarande
                                        nod för att bekräfta mottagen meddelande */

/* Mottagen IdTxAck för Nod B från Nod A visas i deras display som RX 0001 och blir IdRx för att
skicka tillbaka deras Ack-meddelande som koden i else-if satsen visar, vice versa för mottagen IdTxAck
för Nod A från Nod B visas i Nod A:s display som RX 0003 och går in i else-if satsen längre ned */
                                                SJA_Write(CAN_TX_ID, IdTxAck >> 3);
                                                SJA_Write(CAN_TX_LEN, (IdTxAck << 5) | 8);

                                                int i;
                                                char ackmessage = ['ACKNOW-'];

                                                for (i = 0; i <= 6; i++)
                                                {
                                                SJA_Write(CAN_TX_BUF0 + i, ackmessage[i]);
                                                }

                                        // Skriver ut ACKNOW nummer i slutet av bufferten
                                        // j har deklarerats innan while-loopen
                                                SJA_Write(CAN_TX_BUF7, j + 48);
                                                j = (j + 1) % 10;

                                                SJA_Write(CAN_CMD, TXREQ);

                                        }
                                        else if (IdRx == 0x003) //Nod A
                                        //else if (IdRx == 0x001) //Nod B
                                        //ACKNOW identifierare. Skriver ut ACK-medd
                                        {
                                                int i = 0;
                                                char R[9];

                                                for (i; i < 8; i++)
                                                {
                                                R[i] = SJA_Read(CAN_RX_BUF0 + i);
                                                }

                                                R[8] = '\0';

                                                prints("Ack: ");
                                                prints(R);
                                                prints("\n");

                                        }

                                        SJA_Write(CAN_CMD, RRB); // Frigör Rx-bufferten

                                }

                                if (USART_KbHit()) CAN_MessOut(IdTx);
                                // Om det finns data att skicka, hämtar detta från terminalen
                                // flyttar över data till CAN-kontrollern (SJA1000) och
                                // skickar ut det på CAN-bussen
                        }
}
```