UPPSALA
UNIVERSITET

IT 12 003
Examensarbete 15 hp
Januari 2012

Programmering av mikrokontroller för styrning av komponenter i ett biokemiskt analysinstrument

Simon Engqvist

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Programming of microcontroller for control of components in a biochemical analysis instrument

Simon Engqvist

The Uppsala based company Q-linea develops procedures, instruments and systems for protein and nucleic acid analysis. The components in such an instrument are controlled by microcontrollers. Microcontrollers are computers in one singular chip that can be used in a wide range of applications such as cars, toys or in this case an analysis instrument.

When developing a new instrument for biochemical analysis, Q-linea needed new software for controlling and communicating between the components in the instrument. The process of developing this software is the subject of this thesis.

The project included research of microcontrollers and the components in the instrument. The result was a set of example programs for the microcontroller and a design and implementation of the control software for the new analysis instrument.

Handledare: Johan Sternberg
Ämnesgranskare: Karl Marklund
Examinator: Olle Gällmo
IT 12 003

Innehållsförteckning

Sammanfattning

En mikrokontroller är en enkel dator, komplett med processor, arbetsminne, programminnen och stödfunktioner integrerat till samma chip som är optimerad för att styra, arbeta och samarbeta med andra elektroniska enheter. Största användningsområdet är inbyggda system som till exempel bilar, mikrovågsugnar, leksaker etc.

Ett steg i utvecklingen av ett analysinstrument var att förflytta styrningen av flera elektroniska komponenter, som tidigare styrdes av olika datorer och kretskort med mikrokontrollers, till en ny prototyp av kretskort med mikrokontrollers med mer avancerade funktioner för kommunikation. Syftet med detta examensarbete var att utveckla och implementera mjukvaran för ett sådant nytt styrkort.

Arbetet med utvecklingen och implementationen till styrkortet har inneburit inläsning på den processortyp som mikrokontrollern på kortet är av och test av enklare program på kortet för att testa att olika funktioner på kortet fungerar. Slutligen utveckling av mjukvara för kommunikation och styrning till de olika komponenter som kortet ska styra samt utveckling av mjukvaran i kortet för att ta emot olika kommandon och utföra dem.

Bakgrund

Mikrokontrollers

Marknaden för mikrokontrollers växer snabbt. Enligt marknadsforskningsföretaget electronics.ca[1] förutspås att marknadsomsättningen av mikrokontrollers går över $16 miljarder under 2011 och att marknadstillväxten är 9% årligen under kommande 5 år. En mikrokontroller är en komplett dator i en och samma kiselkrets. Processorn är oftast av det enklare slaget och har ofta en registerstorlek på 8 eller 16 bitar även om mikrokontrollers finns med register om 24 eller 32 bitars processorer för mer avancerade tillämpningar. Mikrokontrollers används vanligtvis som kontroll- och styrenheter för regler- och styrsystem inom industrin men används också till exempel i bilar och leksaker. Eftersom mikrokontrollers är mycket små och är lika mångsidiga som persondatorer finns närmast oräkneligt många användningsområden.

Q-linea

Q-linea är ett företag stationerat i Science Park i Uppsala som utvecklar procedurer, instrument och system för analys av proteiner och nuklidsyror. Speciellt med avseende på mikroorganismdetektion, identifikation och *in vitro* diagnostik[2].

Under 2011 utvecklade Q-linea ett nytt instrument i vilket det finns ett antal komponenter som ska styras av mikrokontrollers. Komponenterna består av stegmotorer, ventiler och pumpar.

Ett steg i utvecklingen av analysinstrument var att förflytta styrningen av flera elektroniska komponenter, som tidigare styrdes av olika datorer eller kretskort med mikrokontrollers, till nya prototyper av kretskort med mikrokontrollers med mer avancerade funktioner för kommunikation. Syftet med detta examensarbete är att utveckla och implementera mjukvaran för ett sådant nytt styrkort.

[1] Electronics.ca Research Network,(2011) http://www.electronics.ca/presscenter/articles/1364/1/Microcontroller-Market-Forecasted-to-Reach-Over-16-billion-worldwide-In-2011-/Page1.html (2011-11-15)

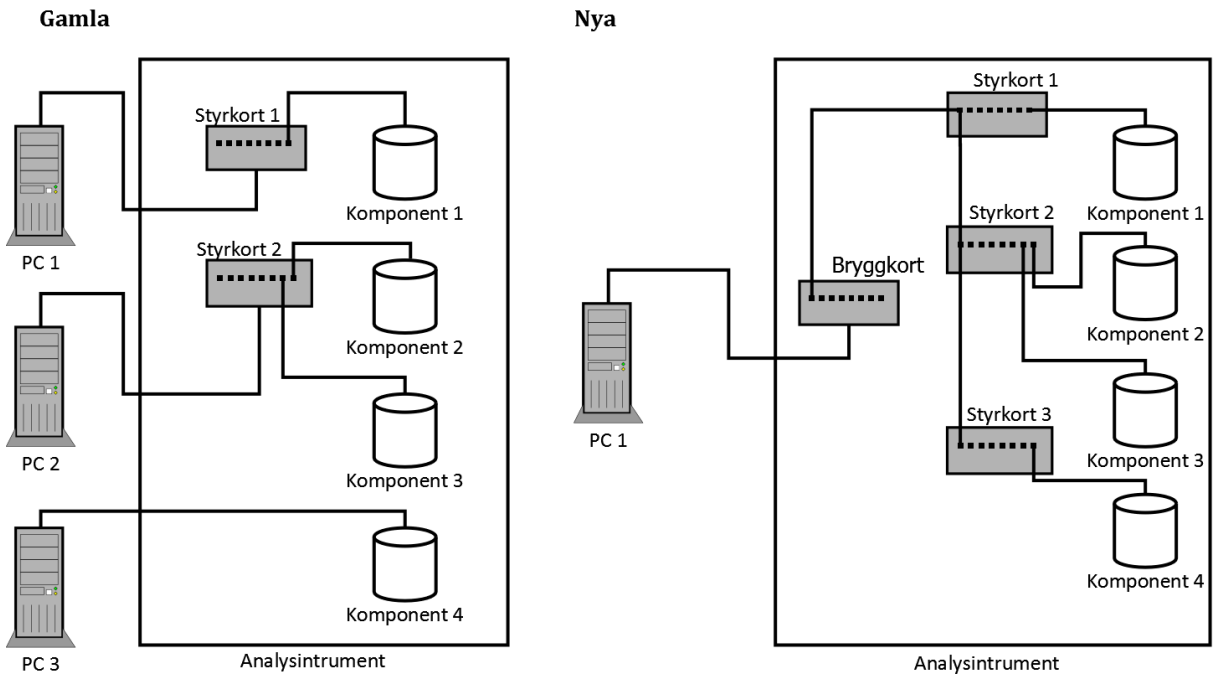
[2] *In vitro:* från latinets ”i glas”, vilket betyder att göra diagnostik utanför en levande organism i en kontrollerad miljö.

Uppgiftsbeskrivning

Målet med detta examensarbete var att vidareutveckla mjukvara för ett kretskort med en mikrokontroller som skall styra komponenter i ett biokemiskt analysinstrument som Q-linea håller på att utveckla. Q-linea har tidigare haft en implementering av ett styrsystem som varit spritt mellan många olika plattformar, PC-maskiner och olika typer av kretskort, men vill ta fram ett nytt system som är mer uniformt med en gemensam struktur för styrning, kommunikation och arkitektur för styrkorten, se figur 1.

Mikrocontollerntyp som ska programmeras är en Atmel AVR processor av modell AT90CAN128 som tillverkas av Atmel. Programmeringsspråket som skall användas är C i Atmels egna utvecklingsmiljö som heter AVR Studio 4.

Funktionerna som skall finnas med i styrsystemet är att kontrollera två stegmotorer, en ventil-switch samt en pump. Styrsystemet skall även kunna ta emot kommandon via seriell kommunikation, USART och via CAN-bus. Q-linea har tidigare implementationer av ett antal olika delar av systemet till en tidigare version av styrkort och en av uppgifterna blir att återanvända och anpassa den koden till det nya kortet.



Figur 1: En abstrakt bild av det gamla systemet respektive det nya. I det gamla systemet styrdes olika komponenter av både styrkort och PC-maskiner. Styrkorten var inte nödvändigtvis kopplade till samma PC. I det nya systemet är alla syrkort kopplade med bus till ett bryggkort som sedan kommunicerar med en PC.

Avgränsningar

I uppgiften ingår inte att utveckla styrfunktioner till någon del som inte tillkommer det aktuella styrkortet.

Metod

Arbetet delades huvudsakligen in i tre på varandra följande delar: inläsning, implementering och rapportskrivning.

Inläsning: Nödvändiga kunskaper och färdigheter införskaffades genom studier av mikrokontrollers och CAN-nätverk. Ett antal testprogram implementerades i en test- och utvecklingsmiljö.

Implementering: Utveckling och implementering av mjukvaran till de komponenter som skall styras samt mjukvaran till styrkortet och kommunikationen. Testning och verifiering av utvecklad mjukvara.

Rapportskrivning: Författandet av denna rapport.

Inläsning

Inläsningen på ämnet och om uppgiften skedde främst i två steg. Först litteraturstudier och sedan utveckling av testprogram på ett utvecklingskort.

Litteraturstudier

Eftersom uppgiften i fråga skulle vara att programmera en mikrokontroller av modellen Atmel AVR AT90CAN128 och jag tidigare saknade både kunskap och erfarenhet om att programmera mikrokontrollers började jag projektet med att studera och läsa litteratur kring mikrokontrollers. Främst kring just den typen av processorer uppgiften handlade om att programmera. Nedan listas den litteratur som användes tillsammans med kommentarer.

Mikrokontrollers: från assembler till RTOS[3] var till hjälp att förstå grunderna kring mikrokontrollers. Den var också till hjälp med att förstå ett CAN-nätverks struktur och hur det fungerar. Någon djupare hjälp att utveckla mjukvaran bidrog inte denna bok med.

AVR: An Introductory Course[4] hjälpte genom att bygga upp en förståelse kring hur just denna typ av processorer fungerar. Den hjälpte även till med att förklara strukturen av register och tanken bakom hur de skulle användas. Däremot förutsatte boken att läsaren utvecklade i en miljö med assemblerspråk, vilket jag inte skulle göra, därför var det enbart de första introduktionskapitlen som var till användning.

Embedded C Programming & the Atmel AVR 2nd Edition[5] var till störst nytta genom projektet. Den innehåller en introduktion till programmeringsspråket C och introduktion till dess användning inom mikrokontrollers. Den förklarar djupgående de vitala delarna i Atmel AVR-processorer och hur de används. Däremot förutsatte alla exempel i boken att jag använde en kompilator som jag inte använde mig av. Därför kunde jag enbart använda mig av exemplen som referenser till de tankemönster som måste användas för att programmera till en AVR-processor, men inte som referenser till exempelkod.

Internet

Internet var till stor hjälp under projektets gång. Det är lätt att hitta och få information. Däremot finns det många källor på Internet som kan vara färgade av personers åsikter vilket jag försökt

[3] Bengtsson Lars, Mikrocontrollers: från assembler till RTOS (Lund: Studentlitteratur, 2009)
[4] Morton John, AVR: An Introductory Cours, (Oxford: Newnes 2002)
[5] H. Barnett Richard H, Embedded C Programming & the Atmel AVR 2nd Edition (Stamford: Delmar Cengage Learning, 2006)

undvika att använda mig av. Däremot kan jag inte förneka olika guider och steg-för-steg-instruktioner som jag använt mig av under inläsningen var till stor hjälp.

Den flitigast använda hemsidan är http://www.avrfreaks.net/[6] som är ett ideellt ”community” och forum för utvecklare till AVR-processorer. På denna hemsida finns många av de guider och instruktioner jag har tagit till hjälp för att förstå och utveckla mjukvara till AVR-processorerna. Däremot måste läsaren vara kritisk då det i denna typ av miljö tenderas att medlemmarna på forumet färgar sina guider med personliga åsikter om hur problem bör lösas. Men det bidrog också till att jag fått se flera lösningar på samma problem vilket hjälpte mig att utveckla min egen metod för problemlösning till AVR-processorer.

Utveckling i testmiljö

Att utveckla testprogram på ett utvecklingskort med en AVR AT90CAN128-processor skedde parallellt med inläsning av litteratur och guider på internet. Det var till stor hjälp att stegvis gå från väldigt enkla program, som att få en lysdiod att lysa, till att göra mer avancerade program, med kommunikation och användandet av processorns timerfunktioner och liknande komponenter. Det bidrog till att bygga upp en förståelse kring hur processorn fungerar och hur den programmeras. Utan att få utveckla till detta utvecklingskort och få gradvis mer kunskap och erfarenhet skulle detta projekt varit mycket svårt för mig att sätta mig in i och förstå. Denna del av inläsningen var till störst hjälp.

[6] AVR freaks, http://www.avrfreaks.net (2011-11-15)

Utvecklingsverktyg

Programmeringsspråket C

Utvecklingen av mjukvaran till styrkortet skedde i programmeringsspråket C. Detta var en fördel då C är ett högnivåspråk som väldigt lätt kan anpassas för olika typer av processorer och mikrokontrollers. Utvecklaren behöver då till exempel inte tänka på registerallokering i processorn eftersom kompilatorn tar hand om det problemet automatiskt.

Utvecklingsmiljön AVR Studio 4, 5 och Notepad++.

Utvecklingen av mjukvaran skedde till en början med hjälp av Atmels egna utvecklingsmiljö AVR Studio 4[7]. AVR studio 4 är en gratis utvecklingsmiljö, utvecklad och distribuerad av Atmel som har stöd för alla Atmels olika mikrokontrollers. Utvecklingsmiljön kan till exempel automatiskt hantera de olika bibliotek som krävs för de olika processorerna. Det enda programmeraren behöver göra är att inkludera ett bibliotek i sin kod och sedan i AVR studio ställa in vilken processor som används. AVR studio tar sedan hand om specifikt vilka bibliotek som skall inkluderas vilket leder till att detta blir transparent för programmeraren. Därför kan kod flyttas mellan olika typer av mikrokontrollers utan att koden behöver ändras nämnvärt. Detta var till stor hjälp då en del av mitt arbete var att anpassa kod från en tidigare styrkrets till en ny.

AVR studio 4 har dock enbart väldigt enkelt stöd för hjälpmedel för utvecklingen av själva koden. I utvecklingsmiljön finns till exempel enbart syntaxmarkering med en färg vilket i stort sätt inte hjälper utvecklaren nämnvärt. Därför började jag efter en tid använda programmet Notepad++[8] som har bättre stöd för sådana funktioner som syntaxmarkering för olika programmeringsspråk och radinskjutning. Notepad++ användes för att utveckla koden för mjukvaran och sedan användes AVR studio 4 för att kompilera och felsöka koden. Notepad++ är ett gratis opensource-program för redigering av kod och textdokument.

Mot slutet av projektet släppte Atmel sin nästa version av AVR studio, AVR studio 5[9]. Detta program använder sig av Microsofts Visual Studio vilket ger bättre stöd för bland annat syntaxmarkering och automatisk komplettering för funktions- och variabelnamn med mera. För övrigt fungerar programmet liknande som AVR studio 4 vilket AVR studio 5 bygger vidare på. Jag använde mig av AVR studio 5 efter att det släppts både i utveckling av koden, kompileringen och felsökning av mjukvaran.

Datablad

Under de olika delarna skedde utvecklingen på olika kort, först på ett test- och utvecklingskort och sedan på det färdiga kortet som skall användas. Dock programmerade jag till samma typ av mikrokontroller, AVR AT90CAN128, till de båda korten. För att kunna utveckla mjukvara måste utvecklaren ha information om hur mikrokontrollern fungerar och hur mikrokontrollerns olika funktioner används, till exempel vad olika register har för namn i AVR Studio eller hur proceduren för

[7] Atmel Corporation, AVR Studio 4, (2011) http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725 (2011-11-15)
[8] Ho, Don, Notepad++ (2011), http://notepad-plus-plus.org/ (2011-11-15)
[9] Atmel Corporation (2011) , AVR Studio 5, http://www.atmel.com/microsite/avr_studio_5/default.asp?source=redirect (2011-11-15)

att initialisera kommunikation ser ut. Den information finns att tillgå i de datablad som hör till processorn och vilka Atmel tillhandahåller gratis på sin hemsida för utvecklare[10].

Databladen och instruktions- och dokumentationsdokumenten till pumpen och ventilen användes för att utveckla mjukvaran för de styrfunktioner tillhörande dessa båda komponenter. Dessa datablad var till stor hjälp i utvecklingen av styrprogrammen.

[10] Atmel Corporation (2005), AT90CAN128 Datasheet, http://www.atmel.com/dyn/resources/prod_documents/doc4250.pdf (2011-11-15)

Design och implementation

Implementationen av styrprogrammet bestod i stort sett av tre stora delar: mjukvara för att styra pumpen, ventilen, motorerna och kommunikationen.

Pump

Pumpen är av modell Cavro XCalibur tillverkad av Tecan. Kommunikation med denna sker via Universal Asynchronous Receiver/Transmitter (UART) med ett av två protokoll. Det ena protokollet är anpassat för att koppla ihop pumpen med en PC och skicka kommandon genom en terminal på PCn. Det andra är ett tillverkarutvecklat protokoll, utvecklat specifik till pumpen, som ger ett visst stöd för felkontroll och sekvensnummer. Uppgiften blev därför att implementera styrfunktioner för detta protokoll till mikrokontrollern.

Kommunikationsprotokoll

Protokollet bygger i enkelhet på att controllern skickar en startbyte följt av ett sekvensnummer, längden på meddelandet följt av en checksumma och sedan en slutbyte. På samma sätt kan pumpen sedan skicka svar till mikrokontrollern. Om checksumman inte stämmer antas att meddelandet är korrupt och kommandot förkastas.

I sekvensnumret finns en flagga som sätts om meddelandet är ett omskick av ett tidigare meddelande.

När pumpen får ett kommandomeddelande sparas vilket sekvensnummer som meddelandet har. Pumpen svarar sedan med ett meddelande och utför sedan kommandot. Varje gång pumpen får ett meddelande med omskick-flaggan satt kontrolleras om meddelandet tagits emot tidigare och med hjälp av det senast sparade sekvensnumret. Om meddelandet tagits emot tidigare skickas ett svarsmeddelande, om inte utförs kommandot varefter ett svarsmeddelande skickas.

Det finns nu tre möjligheter: att all kommunikation kommer fram utan att korrumperas, kommunikationen till pumpen korrumperas eller kommunikationen till mikrokontrollern korrumperas. I det första fallet fungerar allt som det är tänkt. I de andra fallen förkastar ena eller andra sidan det mottagna meddelandet.

Om ventilen förkastar ett korrumperat meddelande skall mikrokontrollern skicka om meddelandet till dess att ett korrekt svar erhålls. Mikrokontrollern skickar då om meddelandet om denna inte fått något giltigt svar efter en i protokollet fastställd tid om 10ms.

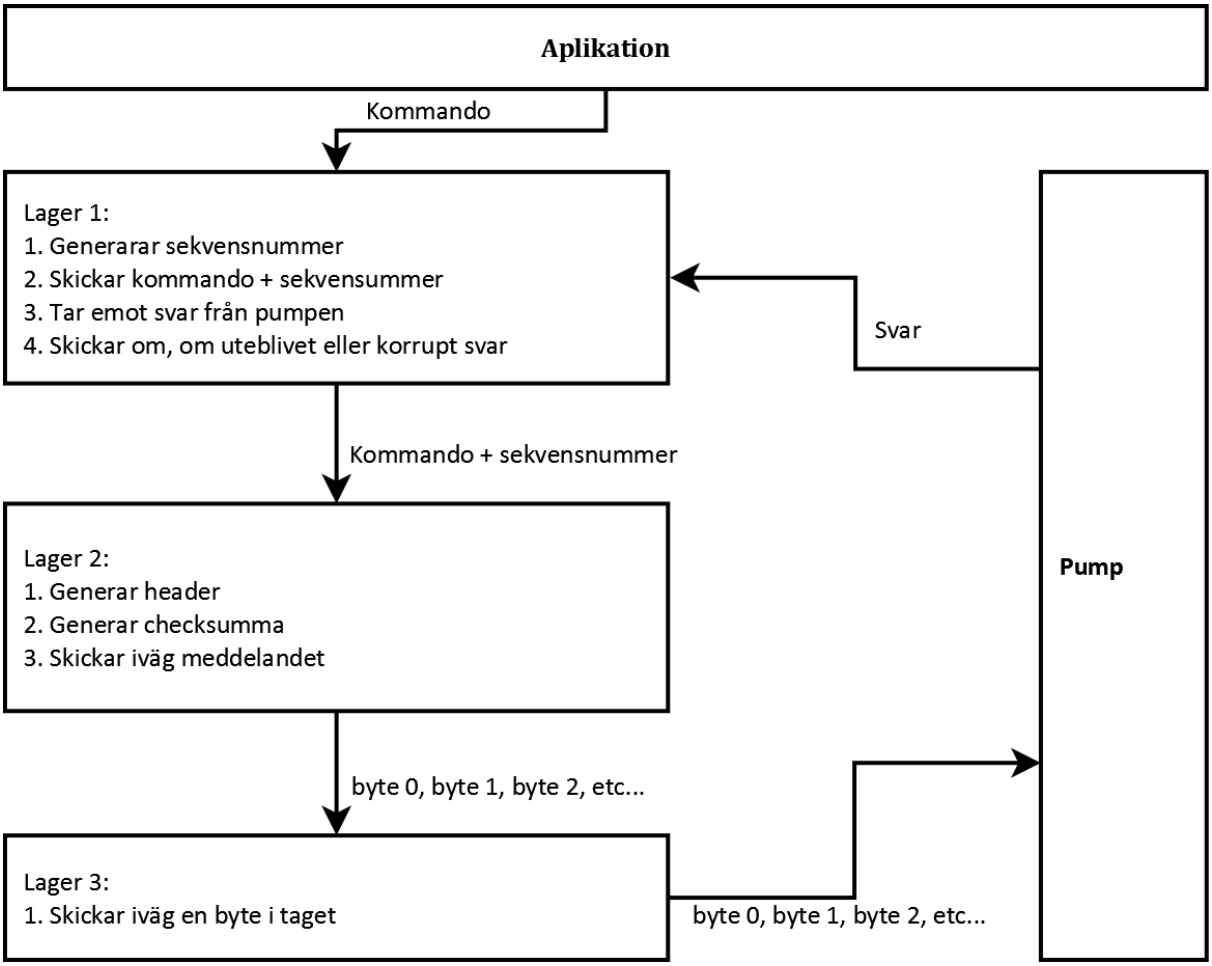
Om mikrokontrollern får ett korrumperat meddelande förkastas detta och mikrokontrollern fortsätter vänta till dess att timeout inträffar. Vid timeout skickas meddelandet på nytt till pumpen, pumpen ser att den tidigare tagit emot meddelandet och skickar om svaret till mikrokontrollern.

Design

Min design för att implementera protokollet som beskrivits ovan består av två delar, att ta emot meddelanden och att skicka meddelanden.

Att ta emot meddelanden är tämligen enkelt. Varje gång mikrokontrollern får en byte på den seriella kommunikationen sparas denna i en buffer till dess att den fått ett helt meddelande. När ett helt meddelande mottagits och checksumman är korrekt sätts en flagga att ett meddelande mottagits.

Sändfunktionen är uppdelad i tre lager, se figur 2. Det översta lagret tar hand om det överliggande protokollet med att ange rätt sekvensnummer samt att ta emot svar och att eventuellt skicka om meddelanden om kommunikationen skulle bli korrupt. I det underliggande mellersta laget finns den funktion som strukturerar upp meddelandet som skall skickas med headern och annan data som behövs för kommunikationen. Denna funktion tar emot en sträng och ett sekvensnummer och skickar detta meddelande med headern till pumpen med rätt pumpadress, checksumma, angiven sträng och sekvensnummer. Det undre lagrets funktion tar emot en byte och skickar denna till pumpen via UART. Genom en pollning, dvs. att mikrokontrollern läser en flagga eller ett register om och om igen till dess att ett önskat värde har skrivits till den, upptäcks när nästa byte är redo att sändas.



Figur 2: Sändfunktionen är uppdelad i tre lager Det översta lagret anropar det mellersta lagret med kommandot och sekvensnumret. Det mellersta lagret omstrukturerar upp meddelandet med headern och checksumma och anropar sedan det tredje lagret med en byte i taget som skall skickas till pumpen. Det tredje lagret skickar varje byte till pumpen. Det översta lagret tar emot svaren från pumpen och ansvarar för att skicka om meddelandet om svaret uteblir eller är korrupt.

När sändfunktionen skickat hela meddelandet väntar den på svar, får controllern inte svar skickas meddelandet igen. Meddelandet försöker att sändas upp till fem gånger utan giltigt svar, efter det antas att kopplingen till pumpen är bruten.

Eftersom pumpen får ett meddelande, svarar och sedan utför kommandot kan det inte antas att pumpen ständigt är redo att ta emot nya kommandon. Därför måste en förfrågan skickas till pumpen om den är klar att ta emot ett kommando innan ett nytt kan skickas.

Implementationen innehåller två funktioner för kommunikationen med pumpen. En funktion som skickar kommandot asynkront[11]. Det vill säga skickar kommandot och väntar på pumpens svar och sedan returnerar utan att titta på om pumpen är klar för nästa kommando. Den andra funktionen är synkron[12] och skickar förfrågan om pumpen är klar fram till dess att den är det. Pumpen är därför klar för nästa kommando när funktionen returnerar.

Ventil

Ventilen är av modell Rheodyne TitanEX. Den har ett antal olika positioner och den styrs genom att sätta spänning på fyra pinnar.

Styrning

Spänningen, hög eller låg, på de fyra styrpinnarna ger ett binärt tal som anger vilken position ventilen skall ta. Ventilen har upp till 10 positioner. I min implementation skall enbart 7 av dessa positioner användas och därför används enbart tre av de fyra styrpinnarna till att ange position. Den fjärde pinnen är jordad och ger därför alltid låg spänning (tolkas som nolla). När en position har angivits skall spänningen vara konstant i minst 10 ms[13] för att ventilen skall registrera den nya positionen.

Output från ventilen ges av två pinnar, ”Done feedback” och ”Error feedback”. ”Done feedback” är hög om ventilen är klar med en rörelse och låg om en rörelse är pågående. När en position sätts på styrpinnarna dras ”Done feedback” ned och ventilen börjar ändra position. ”Done feedback” dras högt igen när ventilen uppnått den angivna positionen. ”Error feedback” indikerar om ett fel har inträffat. Om den pinnen är hög när ventilen är klar med en rörelse har ett fel inträffat.

Design

Det finns två viktiga funktioner för styrning av ventilen. Den första är initieringsfunktionen[14] som förbereder för användning av ventilen genom att tre pinnar på processorn sätts till output och två sätts till input. Dessa pinnar kommer sedan att styra och ta emot feedback från pumpen.

Den andra funktionen är styrfunktionen[15] som givet en byte sätter ventilens utsignal till de tre minst signifikanta bitarna i den byten. Sedan väntar den till ventilens ”Done feedback”-pinne dras låg av ventilen för att avslutningsvis vänta på att dras upp igen. Inbyggd i detta finns en timeout-funktion som hindrar att mikrokontrollern hänger sig medan den väntar om kopplingen mellan mikrokontrollern och ventilen till exempel skulle vara bruten. Om timeouten aktiveras svarar funktionen genom att returnera ett booleskt falskt. När ventilen gör en rörelse skall ”Done feedback”-pinnen först dras låg och sedan hög igen när rörelsen är klar. Om detta sker kollar

[11] Se bilaga tecan.h, rad 83
[12] Se billaga tecan.h rad 87
[13] IDEX Health & Science LLC (2009) , Rheodyne Titan dokumentation, http://www.idex-hs.com/support/rheodyne/downloads/operating_instr/Titan_Driver_Development_Pkg_2321380D.pdf (2011-11-15)
[14] Se bilaga rheodyne.h rad 33
[15] Se bilaga rheodyne.h rad 62

mikrokontrollern om ”Error Feedback”-pinnen är hög. Om den är det returnerar funktionen ett booleskt falskt. Om inte har ventilen mottagit den nya positionen och rört sig till den positionen utan fel. Funktionen returnerar då ett booleskt sant.

Motorer

Mjukvaran bakom motorstyrningen fanns seden tidigare och det jag skulle göra var att konvertera denna kod till den nya processorn. Det visade sig dock att inget större arbete behövdes för att denna kod skulle fungera till den nya mikrokontrollern. Detta tack vare att Atmel använder sig av samma struktur för alla sina AT-controllers samt att AVR studio automatiskt hittar rätt bibliotek om rätt mikrokontroller är angiven.

Kommunikation

Kommunikationen mellan olika styrkort i systemet skall ske med CAN-protokollet. CAN står för ”Controller Area Network”. Kommunikationen i systemet är uppbyggt genom att en PC kommunicerar via USART till ett styrkort som agerar som en brygga och skickar vidare meddelandet till de andra korten på CAN-busen. I projektet ingick att utveckla mjukvaran för kommunikationen på CAN-busen mellan bryggstyrkorten och de andra styrkorten.

Kort om CAN-protokollet

CAN-protokollet är ett protokoll som utvecklades för kommunikation mellan komponenter i bilar. Tekniken är därför utvecklad med avseende på bilindustrin. Eftersom ett eventuellt fel vid kommunikation i en bil kan få ödesdigra konsekvenser jämfört med i till exempel en stereo, som oftast använder sig av I2C protokollet, har CAN utvecklats med mer avancerade funktioner till exempel för felkontroll än andra tekniker och protokoll. Protokollet har inbyggt stöd för att upptäcka fel samt att hantera om flera noder försöker skicka något på busen samtidigt.

I ett CAN-nätverk kopplas varje nod till en bus. Varje enskild nod har ingen specifik adress, istället använder mjukvaran identifierare och masker för att avgöra vilka meddelanden en nod skall ta emot. Varje nod har en lista med identifierare med tillhörande masker. Varje meddelande har en identifierare som givits av den nod som skickade meddelandet. Alla noder ser alla meddelanden men använder sig av de identifierare och masker den har för att avgöra om ett meddelande är aktuellt att ta emot.

När en nod ser ett meddelande appliceras den första masken som noden har i sin identifierarlista på meddelandets identifierare. Om resultatet matchar den första identifierare den har i sin lista tar noden emot meddelandet. Om inte, går noden vidare till nästa mask och respektive identifierare i listan till dess att en matchning uppstår eller att listan tar slut då den ignorerar meddelandet. Varje meddelande är upp till 8 byte stort.

Varje nod har 15 stycken ”brevlådor” vilka benämns som meddelandeobjekt (MObs) som används både till att ta emot och skicka meddelanden. Utvecklaren får i mjukvaran ställa in vilka meddelandeobjekt som skall användas för att ta emot respektive skicka meddelanden. Ett meddelandeobjekt kan inte användas till att både skicka och ta emot meddelanden samtidigt.

Mikrokontrollern AT90CAN har inbyggt stöd för CAN-protokollet genom en inbyggd CAN-controller. Utvecklaren behöver därför inte implementera kontrollen av CAN-nätverket i mikrokontrollern utan

det styrs genom en separat enhet på den aktuella kretsen. Denna enhet styrs genom att register skrivs till eller läses ifrån som med allt annat på en mikrokontroller.

Kodskelett

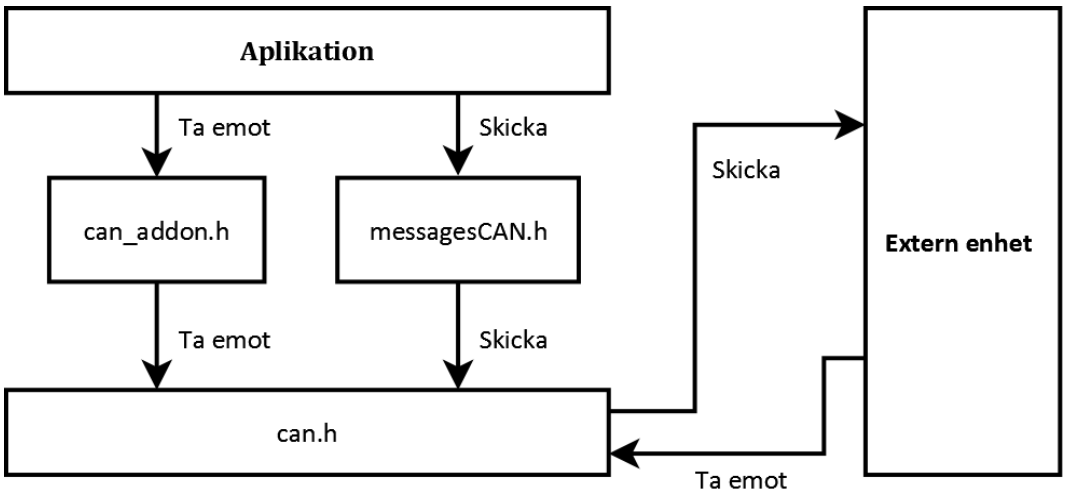
Kommunikationen via CAN-busen är uppbyggd med hjälp av ett kodskelett som tillhandahålles av Dr. Klaus Schaefer från Hochschule Darmstadt University of Applied Sciences i Tyskland[16]. Kodskelettet har funktioner för initiering av CAN-busen samt funktioner för att ta emot och skicka meddelanden på CAN-busen. Det fungerar genom att en funktion anges som ska exekveras när controllern får ett visst meddelande. Kodskelettet har en datastruktur där pekare till alla dessa funktioner sparas. För att spara funktionspekare i denna datastruktur används en funktion som anropas med den aktuella funktionspekaren, identifierare, mask och i vilket meddelandeobjekt som skall användas. När controllern får ett meddelande sker ett interrupt på controllern. I interruptet exekveras den funktion som angavs tillhöra det meddelandeobjekt där meddelandet togs emot. Som argument får funktionen meddelandet. Att skicka meddelanden är implementerat synkront med hjälp av pollning då funktionen väntar till dess att meddelandet är skickat.

Kodskelettet faller under den generella och publika GNU-licensen som innebär fri distribution och användande av koden så länge som användaren själv inte kräver rätt till den. Jag använde detta kodskelett för att det implementerade de funktioner som jag behövde. Kodskelettet var väldigt lätt att bygga vidare på. Funktioner gavs till datastrukturen och de exekverades när controllern får ett matchande meddelande. På detta vis kunde jag koncentrera mig på att skriva de funktioner som skulle användas när controllern får ett meddelande utan att jag behövde återimplementera något som redan fanns under en publik licens.

Kommunikation

Kommunikationen bygger på ovan nämnda kodskelett för att skicka och ta emot meddelanden. Ovanför det finns ett lager, se figur 3, för att andra funktioner skall kunna läsa[17] de mottagna meddelanden och för att skicka[18] meddelanden. När ett meddelande tas emot läggs meddelandet i en buffer och en flagga[19] sätts till sant. Det finns sedan en annan funktion för att läsa meddelandena[20]. Den funktionen kopierar över meddelandet från buffern till en angiven plats och sätter sedan mottagsflaggan till falsk. Flaggan är då sann när ett meddelande finns att läsa och falsk annars. På detta vis kan ett program polla flaggan till dess att ett meddelande tagits emot och kan anropa läsfunktionen för att få meddelandet.

[16] Dr. Schaefe, Klaus , Dr. Klaus Schaefer hemsida, http://kschaefer.eit.h-da.de/ATMEL/CAN/ (2011-11-15)
[17] Se bilaga Can_addon.h
[18] Se bilaga messagesCAN.h
[19] Se bilaga can_addon.h rad 28
[20] Se bilaga can_addon.h rad 26



Figur 3: Kodskelett ansvarar för att ta emot och skicka meddelandet. Applikationer använder funktioner i ett mellanlager för att skicka och ta emot meddelanden.

När kortet skall skicka meddelanden är det för att skicka svar att det mottagit ett kommando och sedan skicka ett till meddelande när kommandot har exekverat klart med dess returvärde. Dessa funktioner[21] skapar meddelanden och packar dem i rätt ordning beroende på typ och skickar sedan iväg dessa genom kodskelettets funktion för detta ändamål. Det finns olika funktioner för att skicka meddelanden beroende på vilken meddelandetyp som används.

Kommandohanteraren

Större delen av kommandohanteraren utvecklades parallellt till flera kort inom företaget av mig och andra anställda tillsammans. Min uppgift var att till kommandohanteraren skriva de ovan nämnda funktionerna. De funktioner som tar emot och skickar meddelanden, tolkar ett inkommet meddelande till ett kommando samt de funktioner som tolkar med vilka argument eller parametrar ett kommando skall exekveras med.

Vilka kommandon som finns tillgängliga är sparade i en tabell. I tabellen finns kommandots nummer, dess namn, funktionspekaren till den funktion som utgör kommandot samt en lista med de typer kommandots argument har.

Alla kommandomeddelande som skickas över CAN är strukturerade enligt specifikation från projektgivaren Q-linea. Den första byten anger kommandots nummer och resten av meddelandet är kommandots argument. Tolkingsfunktionen (parsern) är uppdelat i två steg. I steg ett kontrolleras att kommandonumret i meddelandet verkligen finns i tabellen på kortet[22]. I steg två kontrolleras om resten av meddelandet har den storlek som matchar de förväntade argument som funktionen skall ha[23].

En datastruktur[24] finns för kommandon som skall exekveras. Denna datastruktur innehåller en funktionspekare samt argumenten som funktionen skall exekveras med. Parsningen av ett meddelande med ett kommando och dess argument fyller denna datastruktur med

[21] Se bilaga messagesCAN.h rad 9, 12, 15, 18, 21, 24 & 27
[22] Se parserCAN.h rad 24
[23] Se parserCAN.h rad 34
[24] commandStructRAM

funktionspekaren och argumenten. Om parsningnen lyckades plockas funktionspekaren ur datastrukturen och exekveras med dess argument.

Den del som kollar om kommandonumret finns fungerar genom att tabellen gås igenom uppifrån och ned till dess att den har hittat kommandot eller listan är slut. Om kommandot finns kopieras funktionspekaren till en specifik plats i datastrukturen och värdet sant returneras. Om kommandot inte finns i tabellen returneras falskt. Den andra delen av parsern kollar om det i det inkomna meddelandet finns lika mycket data som motsvarar kommandots argument. Parsern följer den pekare som tidigare lagts in i kommandodatastrukturen från föregående del. Denna pekade följer funtkionen till tabellen och kollar sedan under de två kolumnerna om kommandots argument för att utröna argumentens storlek. I den första kolumnen anges hur många argument funktionen har och den andra anger vilka typer dessa argument har. Utifrån det räknar funktionen ut hur många bytes samtliga argument är sammanlagt och jämför det med de antal byte som finns i meddelandet. Om det inte stämmer returneras falsk.

Timer Countdown

Under utvecklingen av de funktioner som styr ventilen och pumpen framkom att stöd för timeouts behövde implementeras. När controllern skickat ett kommando till någon av dessa komponenter skall denna vänta till dess att ett svar erhålls. För att controllern inte skall hänga sig om något fel har inträffat och ett svar inte skickas, och därför vänta för evigt, behövs en funktion som tillåter mikrokontrollern att sluta vänta efter en viss tid. Därför beslöt jag mig för att skapa funktioner som ger denna funktionalitet.

Timerfunktionerna är två stycken. En start-[25] och en stoppfunktion[26]. Funktionerna är designade för att de skall kunna användas när som helst utan att behöva initialiseras med en annan funktion eller liknande för användande. När startfunktionen skall anropas anges en tid i millisekunder samt en pekare till en flagga. Efter den specificerade tiden sätts flaggan till sann. Funktionen sätter ingång en timer som får ett avbrott vid varje millisekund där den tickar upp en räknare. När räknaren är lika med det antal millisekunder som angetts slår flaggan om och timern stängs av. Stoppfunktionen slår av timern och skall användas om controller fått svar från respektive komponent och timern ej behöver användas mer.

[25] Se timer_countdown.h rad 68
[26] Se timer_countdown.h rad 77

Testning

Testning har skett kontinuerligt genom utvecklingen av mjukvaran. Testningen har skett modulvis genom att varje funktion har testats separat samt att varje modul har testats som en helhet. Slutligen har kortets styrprogram testats i sin helhet.

Testningen har skett genom att koden har körts på en mikrokontroller och att det i koden under utvecklingen har funnits punkter i koden där mikrokontrollern skickat information till en PC. Denna information har bestått av till exempel returvärden från funktioner som testas eller enbart ett meddelande att mikrokontrollern har exekverat en viss bit kod i mjukvaran. Ett alternativ hade varit att använda sig av de emulatorer för debugging som Atmel erbjuder eller använda sig av en av deras debuggingplattformar. Dessa alternativ visade sig dock inte behövas då de antingen krävde mer tid att använda sig av eller var mer tidskrävande än den metod jag använde mig av.

Det som var relativt lätt att testa var de funktioner och moduler som enbart använde sig av de komponenter som fanns i mikrokontrollern. De som var svårare att testa var de funktioner och moduler som kommunicerar med externa enheter som exempelvis pumpen eller ventilen. I dessa fall har testningen varit svårare eftersom mjukvaran för kommunikationen måste testas och sedan måste styrningen av enheterna testas med hjälp av kommunikationen. När ett fel upptäcks måste det först utrönas om felet ligger i styrningen eller i kommunikationen. Därför har det i vissa fall varit problematiskt att hitta var buggen finns.

Överlämning

Överlämning till Q-linea har skett kontinuerligt under utvecklingen genom att jag har lagt upp och uppdaterat filer på en SVN-server enligt instruktioner från projektgivaren.

Utvärdering

Problem

Under projektet har en del små problem uppkommit men enbart ett större.

Kretskortet

Under utvecklingen av styrfunktionerna till ventilen uppstod ett problem då ventilen inte kunde styras fullt ut. Efter felsökande framkom att ventilen kunde upptäcka att pinnar från mikrokontrollern sattes men att den inte kunde upptäcka när de drogs till låg spänning. Efter konsulterande med den ansvariga för styrkorten på Q-linea uppdagades att detta var ett fel eller ett misstag i hårdvaran.

För att skydda mikrokontrollern från att skadas när spänning eventuellt av misstag sattes på pinnarna var kortet designat så att det mellan pinnarna på mikrokontrollern och utgångspinnarna på kortet fanns motstånd. Dessa motstånd hindrade ventilen att registrera att en viss pinne hade satts till jord.

Lösningen blev att avlägsna dessa motstånd för hand genom att löda om styrkortet. En notering om detta gjordes och i framtida versioner av kortet skall dessa inte ha de aktuella motstånden.

Övrigt

Det har varit en intressant och lärorik upplevelse att arbete med programmering och mjukvaruutveckling som skall användas och vars främsta mål inte är att lämnas in för rättning eller granskning som med en laboration eller en inlämningsuppgift. Det har också varit intressant att arbeta i ett arbetslag där medlemmarna använder sig av varandras kod och funktioner. I en sådan miljö måste koden vara lättförstådd och väldokumenterad för att kunna läsas och användas av andra. En viss problematik har uppstått då en medlem inte har kunnat fortsätta innan någon annan i arbetslaget har skrivit klart en annan kritisk del i projektet.

Tidplanen höll någorlunda bra och projektet blev endast en vecka försenat utifrån den ursprungliga planeringen.

Referenser

Litteratur

Bengtsson Lars, Mikrokontrollers: från assembler till RTOS (Lund: Studentlitteratur, 2009)
ISBN: 978-914-407-362-0

Morton John, AVR: An Introductory Cours, (Oxford: Newnes 2002)
ISBN: 978-075-065-635-1

H. Barnett Richard H, Embedded C Programming & the Atmel AVR 2nd Edition (Stamford: Delmar Cengage Learning, 2006)
ISBN: 978-141-803-959-2

IDEX Health & Science LLC (2009), Rheodyna Driver/controller Development Assistance Pakage For Rheodyne Titan and MV Driver Boards (RoHS Compliant)
http://www.idex-hs.com/support/rheodyne/downloads/operating_instr/Titan_Driver_Development_Pkg_2321380D.pdf (2011-11-15)

Tecan Systems, Inc. (2005), Tecan Operating Manual Cavro™ XCalibur Modular Digital Pump
http://www.laball.co.kr/uploads/XC%20Manual.pdf (2011-07-28)

Atmel Corporation (2005), AT90CAN128 Datasheet,
http://www.atmel.com/dyn/resources/prod_documents/doc4250.pdf (2011-11-15)

Mjukvara

Atmel Corporation, (2011) AVR Studio 4,
http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725 (2011-11-15)

Atmel Corporation (2011), AVR Studio 5,
http://www.atmel.com/microsite/avr_studio_5/default.asp?source=redirect (2011-11-15)

Ho, Don, Notepad++ (2011), http://notepad-plus-plus.org/ (2011-11-15)
http://notepad-plus-plus.org/

Internetsidor

AVR freaks
http://www.avrfreaks.net (2011-11-15)
Inloggning krävs för att kunna ta del av material.

Dr. Klaus Schaefer hemsida
Dr. Schaefer, Klaus (2008),
http://kschaefer.eit.h-da.de/ATMEL/CAN/ (2011-11-15)
http://kschaefer.eit.h-da.de/ATMEL/CAN/CANdriver.zip (2011-11-15)
Plats för distribution av CAN-kodskelettet.

electronics.ca
Electronics.ca Research Network,(2011)
http://www.electronics.ca/presscenter/articles/1364/1/Microcontroller-Market-Forecasted-to-Reach-Over-16-billion-worldwide-In-2011-/Page1.html (2011-11-15)
Marknadsforskningsföretag specialiserade inom elektronik- och teknikmarknaden.

Bilagor

.h-filer för de bibliotek och styrfunktioner som ingick i projektet.

`can_addon.h`
Funktioner för kommunikation med CAN-busen

`messagesCAN.h`
Funktioner för att skicka meddelanden, implementeras genom ett messages.h bibliotek som väljer kommunikation via CAN eller USART beroende på vad som används för tillfället.

`parserCAN.h`
Funktioner för att tolka, parsa, inkommna meddelanden till kommandon och dens argument.

`rheodyne.h`
Funktioner för att styra ventilen.

`tecan.h`
Funktioner för att styra pumpen.

`timer_countdown.h`
Funktion för att få en flagga att sättas efter en viss tid.

can_addon.h

```
1  /**
2  Can_addon functions for recieve and fetch functions.
3  *
4  June 2011
5  *
6  Author: Simon Engqvist
7  **/
8  #ifndef CAN_ADDON_H
9  #define CAN_ADDON_H
10
11 #include "global.h"
12 /*
13 PRE:        the global variable cardAddress must have a valid value before use of this function.
14 POST:       The can-comunicationj have been initialized and functions are ready to use.
15 RETURN:     --
16 */
17 void can_addonInit();
18
19 /*
20 Function for reading a mesage from the CAN-bus. Bloks ubntil a message is recieved.
21
22 PRE:        can_addon_init have been called before use of this funciton.
23 POST:       A message from the CAN buffer have been but in bufffer.
24 RETURN:     --
25 */
26 void can_addon_read(char *buffer, int *size);
27
28 volatile BOOL CAN_recieved;
29
30 #endif
```

messagesCAN.h

```
1   #ifndef MESSAGES_CAN_H
2   #define MESSAGES_CAN_H
3
4   #include <avr/pgmspace.h>
5
6   #define ERRORCODE_INVALID_OK 0x01;
7
8   // Send ack
9   void ackCAN();
10
11  // Send ok
12  void okCAN();
13
14  // Send ok followed by one character
15  void ok1CharCAN(char c);
16
17  // Send ok followed by one integer
18  void ok1IntCAN(int i);
19
20  // Send ok followed by one integer
21  void ok2IntCAN(int i1, int i2);
22
23  // Send ok followed by one long
24  void ok1LongCAN(long l);
25
26  //Sends ok followed by one string
27  void ok1StringCAN(char* string);
28
29  // Send OK with a flash string
30  void ok1StringFlashCAN(const prog_char* flashString);
31
32  //Sends ok followed by one int
33  void ok2StringsCAN(char* string1, char* string2);
34
35  //Sends ok followed by three strings
36  void ok3StringsCAN(char* string1, char* string2, char* string3);
37
38  //Sends ok followed by one string and a int
39  void ok1String1IntCAN(char* string, int i);
40
41  //Sends ok followed by two strings and one int
42  void ok1String2IntCAN(char* string, int i1, int i2);
43
44  //Sends ok followed by two strings and two int
45  void ok2String2IntCAN(char* string1, char* string2, int i1, int i2);
46
47  // Send ERROR with an errorcode
48  void errorCAN(const unsigned char errorCode);
49
50  // Send FAULT with an faultcode
```

```
51      void faultCAN(const unsigned char falutCode);
52
53      #endif
```

parserCAN.h

```
1  /**
2  Parse function for can-messages in the command handler.
3  *
4  June 2011
5  *
6  Author: Simon Engqvist
7  **/
8  #ifndef PARSERCAN_H
9  #define PARSERCAN_H
10
11 #include "global.h"
12 #include "commandSyntax.h"
13
14 /* Function for parsing a can-comand string into a comand.
15
16 PRE:        commandStructFlash points at the command array in flash.
17          nrCommands is the current number of commans in the array.
18          commandString have a lenght of 8 elements.
19 POST:       command have been updated with the pointer in flash to the command if the
   command was valid.
20 RETRUN:     TRUE if command was found, FALSE otherwise.
21
22 May 2011 Simon Engqvist
23 */
24 BOOL parseCANCommand(char CAN_msg[], int CAN_msg_size, commandStructRAM* command,
   commandStructFlash* avilableCommands, int nrCommands);
25
26 /* Function for parsing out the arguments of the command.
27
28 PRE:        command contains the pointer to flash for the command.
29 POST:       The arguments have been placed in command.
30 RETURN:     TRUE if the arguments was valid, otherwise FALSE
31
32 May 2011 Simon Engqvist
33 */
34 BOOL parseCANArgs(char CAN_msg[], int CAN_msg_size, commandStructRAM* command,
   commandStructFlash* avilableCommands, int nrCommands);
35
36 #endif
```

rheodyne.h

```
1   /**
2   *Simple iplementation of BCD Control for the Rheodyne TitanEX.
3   * For specifications of the BCD control with Rheodyne TitanEX see
4   * Driver/Controller Development Assistance Package for Rheodyne Titan and  MV Driver Boards
    Chapter 6.1
5   *
6   * May 2011
7   *
8   * Author: Simon Engqvist
9   *
10  **/
11  #ifndef RHEODYNE_H
12  #define RHEODYNE_H
13
14  #include "global.h"
15  /*
16  Resources useages:
17                  Digital I/O: pins:
18                                  PE2
19                                  PE3
20                                  PF4
21                                  PF5
22                                  PF6
23                  Timer 2 via timer.h
24  */
25
26  /*
27  Function for initialization of the rheodyne valve.
28
29  PRE:            --
30  POST:           the correct setting for the valve have been set and rehodyne_set_pos and other
    related function can be used.
31  RETURN:         --
32  */
33  void rheodyne_init();
34
35  /*
36  Function for getting the status of the valve.
37
38  PRE:            rheodyne_init have been called at least once before this function.
39  POST:           --
40  RETURN:         the Status of the valve. bit 0 is set if the valve is done. bit 1 if Error have uccurd, bit 1
    only valid if bit 1 is set.
41  */
42  char rheodyne_get_status();
43
44  /*
45  Function for getting the current position of the valve. Software based!
46
47  PRE:             rheodyne_init have been called at least once before this function.
```

```
48  POST:           -
49  RETURN: the position of the valve that the software last placed the valve.
50                  If valve have nopt been set at least once or an Error have ucurd the return value is 0.
51  */
52  char rehodyne_get_pos();
53
54  /*
55  Function for setting the valve in the new position
56
57  PRE:            rheodyne_init have been called at least once before this function.
58                  pos is a corect valve position: 1<= poss<=6
59  POST:           The valve have recieved the comand and have completed it's movement.
60  RETURN:         TRUE if the Valve is in the new position with no error, otherwise false. Also false if pos
    is an invalid comand.
61  */
62  BOOL rehodyne_set_pos(char pos);
63
64  #endif
```

tecan.h

```
1  /**
2  * Simple iplementation of the OEM protocol for TECAN Cavro XCalibur Modular Digital Pump
3  * Description of the OEM protocol existin in the TECAN Cavro Operating Manual chapter 3.2.1.
4  *
5  * May 2011
6  *
7  * Author: Simon Engqvist
8  **/
9  #ifndef TECAN_H
10 #define TECAN_H
11
12 #define BAUDRATE 9600
13 #define BAUDREGISTER (F_CPU/(BAUDRATE*16UL)-1)
14
15 /*
16 Mechanism to easy switch the code from Using USART0 or 1. Easy to ad another USART interface
   also.
17 One of USART0 or USART1 may be true, not both.
18 */
19 #define USART0 FALSE
20 #define USART1 TRUE
21
22                 #if USART0
23                 #define USARTN_RX_VECT USART0_RX_vect
24                 #define UDRN UDR0
25                 #define UCSRNA UCSR0A
26                 #define UCSRNB UCSR0B
27                 #define UCSRNC UCSR0C
28                 #define RXCIEN RXCIE0
29                 #define UDREN UDRE0
30                 #define RXENN RXEN0
31                 #define TXENN TXEN0
32                 #define UCSZN1 UCSZ01
33                 #define UCSZN0 UCSZ00
34                 #define UBRRNL UBRR0L
35                 #define UBRRNH UBRR0H
36                 #endif
37
38                 #if USART1
39                 #define USARTN_RX_VECT USART1_RX_vect
40                 #define UDRN UDR1
41                 #define UCSRNA UCSR1A
42                 #define UCSRNB UCSR1B
43                 #define UCSRNC UCSR1C
44                 #define RXCIEN RXCIE1
45                 #define UDREN UDRE1
46                 #define RXENN RXEN1
47                 #define TXENN TXEN1
48                 #define UCSZN1 UCSZ11
49                 #define UCSZN0 UCSZ10
```

```
50                  #define UBRRNL UBRR1L
51                  #define UBRRNH UBRR1H
52                  #endif
53
54  /*
55  Resources useages:
56                  USART 0 or 1
57                  timer2 via timer.h
58  */
59
60  /*
61  initialization function for the tecan pump.
62
63  PRE:            --
64  POST:           The correct setting for use of the tecan pump have been set.
65  RETURN: --
66  */
67  void tecan_init();
68
69  unsigned int pump_rec_string(char* rec_buffer);
70
71  /*
72  Send functions to the TECAN Carvo XCalibur pump.
73
74  PRE:            comand_string is a string, in scence that it is a null terminated char array.
75  POST:           If not conection to the pump was lost (return value is 0) the comandstring was sent to
    the pump and ack was recieved.
76  RETURN: the status byte of the ack from the pump or 0 if conection to the pump was lost.
77
78  if ack is lost or corupted the functions tries 5 times before givning up.
79  */
80
81  // Asynchronous version. When return the pump may be executing.
82  // Return byte is the first valid recieved ack from the pump.
83  char tecan_send_async(char* comand_string);
84
85  // Synchronous version. The pump have finished with the comand and is ready for new comand.
86  // Return byte is the first valid ack from the pump when the pump is ready.
87  char tecan_send_sync(char* comand_string);
88
89  /*
90  Change the adress of the pump
91  PRE:            adress must be a char 0x31 <= new_adress <= 0x3F
92  POST:           the functions will now send to the new pump adress.
93  RETRUN: --
94  */
95  void tecan_change_adress(char new_adress);
96
97  #endif
```

timer_countdown.h

```
1  /**
2  * Timer functions for geting a flag set after a amount of specified time.
3  *
4  *May 2011
5  *
6  * Author: Simon Engqvist
7  *
8  **/
9  #ifndef TIMER_COUNTDOWN_H
10 #define TIMER_COUNTDOWN_H
11
12 #include <avr/io.h>
13 #include <string.h>
14 #include <stdio.h>
15 #include <avr/pgmspace.h>
16 #include <avr/interrupt.h>
17
18 #include "global.h"
19
20 /*
21 Mechanism to easy switch the code from Using TIMER0 or 2.
22 One of TIMER0 or TIMER1 may be true, not both.
23 NOTE: in timer_start(..,..) when the prescaler for the timer is set, there are two diffrent setting for
   prescaler for the diffrent timmers.
24 */
25 #define TIMER0 FALSE
26 #define TIMER2 TRUE
27
28                 #if TIMER0
29                 #define TIMERN_COMP_VECT TIMER0_COMP_vect
30                 #define TIMSKN TIMSK0
31                 #define TCCRNA TCCR0A
32                 #define WGMN0 WGM00
33                 #define WGMN1 WGM01
34                 #define CSN2 CS02
35                 #define CSN1 CS01
36                 #define CSN0 CS00
37                 #define OCRNA OCR0A
38                 #define OCIENA OCIE0A
39                 #endif
40
41                 #if TIMER2
42                 #define TIMERN_COMP_VECT TIMER2_COMP_vect
43                 #define TIMSKN TIMSK2
44                 #define TCCRNA TCCR2A
45                 #define WGMN0 WGM20
46                 #define WGMN1 WGM21
47                 #define CSN2 CS22
48                 #define CSN1 CS21
49                 #define CSN0 CS20
```

```
50                  #define OCRNA OCR2A
51                  #define OCIENA OCIE2A
52                  #endif
53
54  /*
55  Resources useages:
56                  Timer 0 or 2, now 2.
57  */
58
59  /*
60  Function for starting the timer.
61
62  flag MUST be a volatile variable. otherwise the usages of the flag may not notice the flag being
    changed.
63
64  PRE:            time is set in ms.
65  POST:           a timer is started and will set the flad to true after the sopecified amount of time;
    time.
66  RETURN:         --
67  */
68  void timer_countdown_start(unsigned int time, volatile BOOL* flag);
69
70  /*
71  Function to turn of the timer.
72
73  PRE:            the timer is running
74  POST:           the timer will be turned of.
75  RETURN:         --
76  */
77  void timer_countdown_stop();
78
79  #endif
```