TSEA56 - Kandidatprojekt i elektronik
Förstudie: Kommunikation

Version 1.1

Grupp 4

Gustafsson, Lovisa `lovgu777`

Tronje, Elena `eletr654`

2 juni 2016

Status

Granskad	-	-
Godkänd	-	-

PROJEKTIDENTITET

2016/VT, Undsättningsrobot Gr. 4

Linköpings tekniska högskola, ISY

Namn	**Ansvar**	**Telefon**	**E-post**
Isak Strömberg (IS)	Projektledare	073-980 38 50	isast763@student.liu.se
Olle Hynén Ulfsjöö (OHU)	Dokumentansvarig	070-072 91 84	ollul666@student.liu.se
Emil Wasteson (EW)	Hårdvaruansvarig	076-836 61 66	emiwa068@student.liu.se
Elena Tronje (ET)	Mjukvaruansvarig	072-276 92 93	eletr654@student.liu.se
Zimon Inge (ZI)	Testansvarig	070-171 35 18	zimin415@student.liu.se
Lovisa Gustafsson (LG)	Leveransansvarig	070-210 32 53	lovgu777@student.liu.se

E-postlista för hela gruppen: isast763@student.liu.se

Kund: ISY, Linköpings universitet
tel: 013-28 10 00, fax: 013-13 92 82
Kontaktperson hos kund: Mattias Krysander
tel: 013-28 21 98, e-post: matkr@isy.liu.se

Kursansvarig: Tomas Svensson
tel: 013-28 13 68, e-post: tomass@isy.liu.se
Handledare: Peter Johansson
tel: 013-28 13 45, e-post: peter.a.johansson@liu.se

Sammanfattning

Denna rapport undersöker dels hur intern kommunikation i en modulär robot kan ske, dels vad som är viktigt att tänka på i designen av modulerna och testerna av dem. I ett projket med begränsat antal moduler är I^2C ett alternativ för kommunikation. För att minska hårdvaruberoendet är C ett användbart programmeringsspråk. När tester ska designas är det viktigt att tänka på hela systemets funktionalitet. I modulbaserade projekt finns det en rad faktorer att ha i åtanke.

e-post: lovgu777@student.liu.se
eletr654@student.liu.se

Innehåll

Dokumenthistorik

Version	**Datum**	**Utförda förändringar**	**Utförda av**	**Granskad**
0.1	2016-03-03	Första utkastet	Grupp 4	ET
1.0	2016-04-07	Andra utkastet	Grupp 4	LG
1.1	2016-04-13	Tredje utkastet	Grupp 4	ET

1 Inledning

Vid utveckling av en robot som består av olika moduler med olika funktion är det viktigt att undersöka hur dessa moduler ska designas på bästa sätt. Därför är det intressant att titta på hur de ska kommunicera med varandra och tillsammans funktionera som en robot, inte som delmoduler.

Syftet med denna studie är att undersöka hur intern kommunikation och design av moduler kan ske vid utveckling av en modulär robot samt analysera vad som skulle vara lämpligt att använda i projektet som ska genomföras.

2 Problemformulering

För att besvara syftet med studien avses följande frågeställningar undersökas.

- Vilken busstyp är lämpligast, SPI eller I^2C, och hur ska den konfigureras?
- Vilka för- och nackdelar finns med C repektive Assembler?
- Hur ska tester designas?
- Vad är viktiga faktorer att ha i åtanke vid design av modulbaserade projekt?

e-post: lovgu777@student.liu.se
eletr654@student.liu.se

3 Kunskapsbas

I detta avsnitt behandlas den bakgrundsinformation som krävs för att besvara frågeställningarna.

3.1 I²C

I²C-bussen är en tvåtrådad buss som sköter överföringen mellan två eller fler enheter. Varje enhet har en unik adress och kan konfigureras till att agera både mottagare och sändare. Utöver detta kan enheter kopplade till samma buss konfigureras till två olika roller, *slav* eller *master*. De som kan initiera överföringar via bussen är de som innehar rollen *master*. På varje buss kan det finnas flera masterenheter, likväl som flera slavenheter. Hastigheten med vilken data kan skickas är i standardläge 100 kbit/s. Vid High speed-mode kan den komma upp i 3,4 Mbit/s. Både SDA och SCL ligger höga när bussen är fri.

Överföringsprocessen består av fem steg:

1. **Start:** Triggas av negativ flank på SDA då SCL ligger hög.
2. **Adress till slav:** En sju bitar lång sekvens som bildar ett unikt ID till en annan enhet på bussen. När en masterenhet triggat initiering anses alla andra enheter vara slavar.
3. **Datariktining:** En bit som reglerar om masterenheten vill skicka eller ta emot data. Tillsammans med ovanstående bildar de den första byten som skickas.
4. **Information:** Åtta bitar långa sekvenser med den mest signifikanta biten (MSB) först. Antalet bytes per överföring är inte reglerat.
5. **Stop:** Triggas av positiv flank på SDA då SCL ligger hög.

Ett alternativ till att trigga avslut är att trigga en ny start, det vill säga att masterenheten har fortsatt kommunikation över bussen. Är inte slaven redo för överföring när den blir kallad på kan den sätta masterenheten i vänteläge. Efter att varje byte är överförd skickas en bit för att försäkra sändaren om att mottagaren lyckats ta emot den sända informationen, en acknowledge-signal. [1]

3.2 SPI

SPI är ett överföringsprotokoll som kan användas för dataöverföring mellan två eller fler enheter. I konfigurationen finns det minst en masterenhet, vilken initierar all kommunikation, och resten är slavenheter. Som standard sker överföringen genom ett skifte där en bit i taget flyttas från slaven till mastern och vice versa. Hårdvarumässigt krävs minst fyra portar:

e-post: lovgu777@student.liu.se
eletr654@student.liu.se

1. **MISO:** Master input, slave output
2. **MOSI:** Master output, slave input
3. **SCK:** Klockpuls
4. **SS:** Slavenheternas koppling till masterenheten

Masterenheten initierar kontakt genom att, på rätt SS-port, dra aktuell slav ur sitt idle-state. För att starta överföringen skrivs en byte till *dataregistret*. Då startas klockgeneratorn och skiftet börjar. Beroende på konfiguration kan samplingen och iordninggörandet, *setup*, ske på positiv respektive negativ flank eller vice versa. När överföringen är färdig sätts en flagga. För att fortsätta överföring kan en ny byte skickas till dataregistret, annars dras aktuell SS-lina tillbaka till idle. Synkronisering av överföringen kan ske genom att skifta SS-linan fram och tillbaka. Det finns en buffert som lagrar den senaste byten för framtida användning.[2]

3.3 Testning av system

För testning av mikroprocessorer finns verktyg att ta till, till exempel JTAG och logikanalysator.

3.3.1 JTAG ICE

JTAG är ett verktyg för att avlusa alla AVR 8-bitars mikrokontrollers som har ett JTAG-gränssnitt. Istället för att efterlikna enhetens beteende likt en emulator använder JTAG ett inbyggt chip som finns på de enheter som har ett JTAG-gränssnitt. Detta innebär att den kör koden på den fysiska enheten. Användaren får feedback genom programmet AVR Studio. [3]

3.3.2 Logikanalysator

Logikanalysatorn är ett oscilloskop som har möjlighet att analysera flertalet signaler samtidigt. Det finns 16 logiska kanaler och två analoga kanaler. Analysatorn har en rad inställnings- och beräkningsfunktioner för avlusning. [4]

4 Diskussion

I detta avsnitt diskuteras möjliga lösningar till frågeställningarna.

4.1 Konfiguration av I²C och SPI

Vid användning av ATMega16 tillsammans med I²C eller SPI är det inte möjligt att använda parallella bussar, då processorn enbart har en uppsättning utgångar för att hantera I²C respektive SPI. Med parallella bussar menas i detta fall att det finns två eller flera bussar av samma typ på samma processor men som är helt särskilda från varandra. Önskas denna funktionalitet, två separerade bussar, behöver båda typerna användas. För repektive busstyp finns en rad konfigurationsmöjligheter. Det sätt som passar bäst beror både av behovet av timing och vilket beroende varje enhet har av data från andra. Nedan listas tre sätt:

1. Flera enheter kan konfigureras till att inneha en masterroll. Dessa kan, när bussen är fri, initiera kontakt med vilken annan enhet den önskar. På så vis kan de i stor utsträckning styra själva när de vill skicka respektive ta emot data. Vid användning av I²C används arbitrering för att hantera situationer då flera masterenheter vill använda bussen samtidigt för att säkerställa att enbart en når ut på bussen och att det meddelandet inte förstörs. [5] Motsvarande system för att hantera flera masterenheter saknas för SPI.
2. En eller flera slavenheter kan kopplas till en enda masterenheten med avbrott. När slaven vill ha kontakt över bussen triggar den en avbrottssignal, vilket i sin tur får masterenheten att initiera kontakt. Detta medför att data enbart kan gå mellan masterenheten och respektive slav, inte slavarna emellan.
3. Bussens enda masterenhet pollar slavenheterna med ett jämt intervall. Då kan masterenheten själv styra när den vill ha respektive data, med risk för att slavenheten inte har något att skicka och antingen sätter mastern i vänteläge eller skickar den vidare.

Det är även möjligt att kombinera alternativ 2 och 3, det vill säga att några slavenheter kopplas med avbrott för att säga till när det finns information att hämta och några lämnas åt masterenheten att polla.

Då initieringen på I²C-bussen fungerar som ett avbrott behöver inte enheterna kopplade till bussen ligga och lyssna efter en startsignal. De blir istället avbrutna i det de håller på med.[5] Över SPI sker istället ett informationsutbyte mellan master och slav, vilket gör att dataregistret kopplat till SPI:n kan uppdateras med ny information så länge det inte pågår en överföring. [2]

Generellt innebär avbrottsstyrning att CPU:n hålls fri eftersom enheten slipper ligga och lyssna efter en signal om att skicka eller ta emot data. Det ligger istället på varje del i systemet att tillkalla uppmärksamhet. Detta är fördelaktigt då det är kritiskt att något tas om hand precis när det sker. I polling är det upp till huvudmodulen att se till att tillfredsställa alla delar i systemet. Fördelen med detta är att den styrande enheten säger till när den har möjlighet att processa en förfrågan. [6]

När projektet är beroende av bra tajming är avbrottsstyd buss att föredra då förfrågan kan hanteras på en gång. Eftersom det beroende som finns mellan projektets olika moduler kan styras via en och samma modul är det möjligt att enbart använda en masterenhet.

4.2 Jämförelse mellan SPI och I²C

Både SPI och I²C kommer med sina för- och nackdelar och har några väsentliga skillnader:

1. På SPI behöver slavarna inte ha unika adresser, som de behöver ha på I²C. Detta gör att för I²C finns det ett begränsat antal slavadresser som kan användas och därmed ett begränsat antal enheter som kan kopplas till bussen. Däremot behöver SPI unika kopplingar mellan masterenheterna och slavenheterna, vilket kräver fler utgångar.
2. I²C använder sig av en acknowledge-signal för att försäkra sändaren om att mottagaren fått informationen, vilken saknas hos SPI. Detta gör det möjligt för masterenheten att skicka information till ingenting och inte veta om det.
3. SPI använder fler antal portar än I²C i grundutförandet.
4. I²C har ett inbyggt system för att hantera flera masterenheter, vilket SPI inte har.

Det som talar för SPI är kopplingen mellan de olika enheterna då den sker hårdvarumässigt och inte kräver programmering av adresser. Dock tas fler portar i anspråk, vilket kan vara ett problem i vissa tillämpningar. Acknowledge-signalen som I²C använder talar för användningen av den bussen då det går att försäkra sig om att information går fram. Vid mer komplexa system med behov av flera masterenheter är I²C att föredra tack vare det inbyggda systemet.

I ett projekt med ett tydligt begränsat antal moduler och där portarna är en begränsning faller valet i första hand på I²C. Behövs det ytterligare en buss för informationsöverföring kan SPI användas för det syftet.

4.3 Assembler eller C

Enligt [7] finns det några faktorer att tänka på vid val av programmeringsspråk för inbyggda system. Dessa inkluderar bland annat hur väl det går att uttrycka händelser, speciellt vid avbrott, och hur hårdvara med speciell funktion hanteras, så som statusregister och tillhörande avbrott.

I [8] anses Assembler vara ett snabbare alternativ än högnivåspråk som C i enklare konstruktioner av realtidssystem. En nackdel med Assembler är dock att koden blir svårare att läsa för framtida användare då dessa måste sätta sig in i hur processorn fungerar. Assembler ser olika ut för olika processorer och att lära sig nytt språk till varje ny processor är inte optimalt. Detta gör Assembler mindre attraktivt. Även [9] säger att Assembler blir för knutet till den specifika hårdvaran som används medan C kan fungera på olika processorer. Koden i C blir även mer lättförståelig och visar koncepten av vad processorn ska göra på ett logiskt sätt.

Det finns alltså fördelar och nackdelar med båda språken. Även om Assembler ger snabbare utfall gör dess hårdvaruberoende användningen onödigt krånglig. En sådan detaljinsikt i hårdvaran är i många fall inte nödvändig. Sett till att projektet som ska utföras är så pass litet är det tveksamt om den snabbhet Assembler ger är av någon direkt betydelse. Dessutom finns det inte tid till att på detaljnivå förstå hårdvaran för att kunna skriva allt i Assembler. I det avseendet skulle C på grund av dess smidighet vara ett mer fördelaktigt språk att programmera i. Är någon funktion tidskritisk kan det ändå vara en idé att överväga att skriva detta i Assembler.

Det finns möjlighet att kombinera båda språken. I [10] beskrivs hur Assembler kan användas i ett C-projekt i Atmel Studio 6. En assemblerrutin kan bli synlig för en C-kodad fil och de kan dela globala variabler. Detta kan vara användbart om vissa delar blir enklare eller tydligare att koda i Assembler.

4.4 Design av tester

En simulator är ett enkelt val för testning av kod då de flesta mikroprocesorerna har en egen sådan. Simulatorer brister dock i att det är svårt att förutse allt som kommer hända i verkligheten, vilket gör att den som programmerar måste vara extremt medveten om alla situationer som kan uppstå annars missas saker. Avbrott försvårar detta.[8]

För testning av system föreslår [8] att testning av hårdvara och mjukvara sker var för sig för att sedan testas tillsammans. Hårdvarutestning kan exempelvis ske genom att enklare kod skrivas till en viss port för att på ett oscilloskop se vad utslaget blir. Mjukvarutestning görs bäst genom att dela upp koden och testa en liten bit i taget lämpligtvis med hjälp av breakpoints. Vad gäller test i realtidssystem blir det svårare då avbrott sker kontinuerligt vilket gör att breakpoints i koden inte är optimalt. För att underlätta då kan alla avbrott som inte är kritiska stängas av för att se om felet kvarstår och sedan aktiveras ett avbrott i taget för att hitta var felet är.

4.4.1 Olika typer av test

Nedan listas olika testfaser för att jobba sig igenom systemets olika delarna på ett systematiskt sätt:

1. **Enhetstest:** Testar en enskild enhet. Denna typ av test är viktigt då det är lättare att hitta fel när det är tydligt från vilken enhet felet kommer.
2. **Integrationstest:** Testar att interaktionen mellan olika enheter fungerar, med syfte att samanfoga enheter för att testa deras helhetsbeteende. Här ligger fokuset på att testa gränssnitt.
3. **Systemtest:** Testar hela systemet, och fokuserar på de egenskaper som enbart finns när systemet är komplett.
4. **Acceptanstest:** Test för att kunden ska godkänna systemet.

I varje specifik fas är det viktigt att ta fram testfall på ett sådant sätt att om testet godkänns går det att anta att programmet är korrekt då testning enbart kan påvisa att fel existerar, aldrig att de inte gör det. Detta kan göras genom att dela upp indata i olika ekvivaleskategorier, och testa så att programmet svarar på önskvärt sätt. Här är det viktigt att räkna med undantagsfall också.[11] Med hjälp av en bra kravspecifikation kan testerna lätt definieras då det tydligt framgår vad varje del, samt det totala systemet, ska klara av.

4.5 Modulbaserade projekt

Nedan beskrivs fördelar med modulära system samt faktorer att tänka på vid utveckling av modulära system.

4.5.1 Fördelar med modulär design

Vid arbete med modulär design av system kan de olika modulerna göras så de kan produceras oberoende av varandra och användas i olika system. Detta gör att kostnader minskar då modulerna ej behöver specialgöras vilket även medför flexibilitet vad gäller design och att systemet enkelt kan utökas med ytterligare moduler.[12] I case-studien [13] bekräftas att modulär design bidrar till ökad produktvariation då ett fåtal moduler kan kombineras på många olika sätt och på så sätt skapa olika produkter. Även en förbättrad kvalitet och ledtid kan fås genom modulär design. Enligt [14] är fördelarna med modulär design att tiden för designarbetet blir kortare, lägre priser och likt ovanstående ett ökat antal möjliga konfigurationer.

4.5.2 Att tänka på vid design av modulära system

I design av ett modulärt system är det viktigt att både bryta ned de funktionella kraven och modellera systemet. Genom att enbart titta på funktionerna går systemets dynamiska beteende förlorat. När det kommer till stora system blir kraven lätt många, och då är det viktigt att titta på helheten, annars finns det en risk att vissa delar optimeras på bekostnad av någon annan. Begränsande krav på systemets helhet minskar möjligheten för modulär design, vilket kan skapa intresse av att minska eller eliminera dessa. [15] I [13] uppmärksammas även vikten av att systemintegration tas i beaktning vid modulär design, något som inte alla tillverkare tillämpar.

[16] beskriver bland annat följande faktorer att tänka på vid design av en ”open-architecture target environment”. Det måste finnas en synkronisering av uppgifter då det exempelvis inte är garanterat att alla sensorer jobbar i samma hastighet och därför behöver olika frekvens för olika uppgifter. Det kan även bli problem med förvrängning av signalen när två processorers systemklockor jobbar i något olika hastighet och två uppgifter har samma frekvens. Detta medför att integrationen av moduler ej bör bero på frekvens eller systemklockor vid synkronisering. Dessutom bör ett helt set av data skickas vid kommunikation mellan två moduler, inte en del från den föregående cykeln och en del från den senaste. Det är även viktigt att kommunikationen mellan moduler är förutsägbar så att det kan förutspås hur lång

tid utförandet som sämst kommer ta så att hänsyn kan tas till det. Detta är särskilt viktigt i realtidssystem. När alla moduler delar databuss är det fördelaktigt att designa kommunikationen mellan dessa så att den tar så lite tid som möjligt för att inte trafikera bussen för länge. På så sätt är det mer sannolikt att händelser sker i realtid.

e-post: lovgu777@student.liu.se
eletr654@student.liu.se

5 Resultat och slutsatser

Nedan presenteras studiens resultat och slutsatser.

5.1 Val av databuss och hur den ska konfigureras

I projektet är antalet moduler begränsat till tre, vilket innebär att de utan problem får plats på I^2C-bussen. Eftersom den största begränsningen inte är ett problem faller valet på I^2C för huvudbussen.

I projketet går det att konfigurera bussen med enbart en masterenhet då beroendet mellan dessa moduler kan styras med en central modul. Detta skapar ett behov av att även implementera en SPI-buss om några av slavarna behöver viss information från andra enheter än de andra modulerna.

Projektet har ett stort behov av tajming, att uppgifter utförs direkt när de behövs. Därför faller valet på avbrottsstyrning över pollning.

5.2 Assembler eller C

För programmering av processorer i ett litet realtidssystemsprojekt är C att föredra då det är smidigt, lättläst och enkelt kan appliceras på en annan processor än den som används. Förståelsen för hårdvaran behöver inte vara lika stor, vilket underlättar för programmeraren. Vid tidskritiska moment kan det vara en god idé att överväga att skriva koden i Assembler då det då utförs snabbare. Eftersom det går att blanda språken relativt enkelt är det inget större problem att göra det. I projektet kommer därför C att användas i första hand.

5.3 Design av tester

I framtagningen av en testplan är det viktigt att se till att täcka alla fall av indata för att upptäcka alla eventuella fel så tidigt som möjligt i processen. Ju tidigare de upptäcks, desto lättare är det att identifiera dem och göra något åt dem. Därför kan en framgångsrik strategi vara att designa test på olika nivåer för att stegvis lägga till mer och mer funktionalitet.

I avlusningen av koden finns tillgång till JTAG ICE, vilken är bättre än en simulator då den kör koden på den faktiska enheten. För att testa hårdvaran är logikanalysatorn ett lämpligt verktyg.

5.4 Att tänka på i modulbaserade projekt

Modulbaserade system kan ge flertalet fördelar, bland annat en ökad produktvariation då moduler kan kombineras på flertalet olika sätt och minskade kostnader.Vid utveckling av dess system är det dock viktigt att inte bara dela upp efter funktion utan även tänka på systemet som helhet, särkilt vad gäller stora system. Faktorer att tänka på vid utveckling av

moduler är också hur de ska fungera tillsammans. Integrationen av moduler bör ej bero på frekvens eller systemklockor vid synkronisering. Det är även viktigt att ha tänkt igenom kommunikationen mellan moduler så att inte bussen blir uppehållen för länge och processer tar för lång tid samt att det som kommuniceras är så uppdaterat som möjligt.

e-post: lovgu777@student.liu.se
eletr654@student.liu.se

Referenser

[1] Philips Semiconductors, *The I^2C-bus specification*, January 2000. Version 2.1.

[2] Atmel Corporation, *ATmega16, ATmega16L*, 2010.

[3] Atmel Corporation, *AVR® JTAG ICE, User Guide*, 2001.

[4] Agilent Technologies, *Agilent 6000 Series Ascilloscope, User's Guide*, June 2006. Fourth edition.

[5] NXP Semiconductors, *I^2C-bus specification and user manual*, April 2014. Sixth edition.

[6] A. M. Richard Jones, *Computer Science Java Enabled*. Victoria, Australia: IBID Press, second ed., 2004.

[7] D. O. Williams, "Software and languages for microprocessors," *Computer Physics Communications*, vol. 41, no. 2-3, pp. 295 – 326, 1986.

[8] S. R. Ball, *Embedded Microprocessor Systems: Real World Design*. Burlington, MA: Newnes, second ed., 2000.

[9] J. Pardue, *C Programming for Microcontrollers: Featuring ATMEL's AVR Butterfly and the Free WinAVR Compiler*. Knoxville, TN: Smiley Micros, 2005.

[10] Atmel Corporation, *Atmel AT1886: Mixing Assembly and C with AVRGCC*. November 2012.

[11] Chalmers - Göteborg University, *Testning*, 2009.

[12] M. Karimi, A. Ahmadi, N. K. Korghond, E. Babaians, and S. S. Ghidary, "Remoro; a mobile robot platform based on distributed i/o modules for research and education," in *Robotics and Mechatronics (ICROM), 2015 3rd RSI International Conference on*, pp. 657–662, Oct 2015.

[13] A. K. W. Lau, "Managing modular product design: Critical factors and a managerial guide," in *Management of Engineering Technology, 2009. PICMET 2009. Portland International Conference on*, pp. 2045–2057, Aug 2009.

[14] C. Pozna, "Modular robots design concepts and research directions," in *Intelligent Systems and Informatics, 2007. SISY 2007. 5th International Symposium on*, pp. 113–118, Aug 2007.

[15] G. J. R. Armen Zakarian, "Development of modular electrical systems," *IEEE/ASME TRANSACTIONS ON MECHATRONICS*, vol. 6, no. 4, pp. 507–520, 2001.

[16] D. B. Stewart, R. A. Volpe, and P. K. Khosla, "Integration of real-time software modules for reconfigurable sensor-based control systems," in *Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on*, vol. 1, pp. 325–332, Jul 1992.