Laborationsmaterial

Realtidssystem - CT3200

Inledning

Laborationsserien består av en laboration och en projektuppgift. Uppgifterna ska lösas i grupper om max 3 personer där varje person måste kunna redogöra för de lösningar som ni tar fram.

Laborationsmaterial:

Laborationsmaterialet består av Lego Mindstorm ©, en utvcklingsmiljö och dokumentation

LEGO Mindstorm, en låda med Lego Mindstorm kan kvitteras ut hos ansvarig lärare eller labassistent. För att få kvittera ut Lego Mindstorm paketet måste alla gruppdeltagare vara registrerade på kursen. Vid utkvittering sker en inventering av material i lådan, vid återlämning av lådan ska det återlämnade materialet motsvara det utkvitterade. Lådan får utkvitteras och behållas av gruppen under den tid som det finns schemalagda kurs och/eller laborationstillfällen. Lådan lämnas tillbaka senast en vecka efter tentamenstillfället vid varje kurstillfälle, t.ex. om en grupp av studenter påbörjar kursen period 1 år 2004 så måste lådan lämnas tillbaka senast en vecka efter första tentamenstillfället för period 1 år 2004. Lådan får alltså inte behållas till gruppen har avslutat kursen. I de fall där det finns grupper som inte är klara med laborationer och projekt före inlämnaning av legolåda finns det möjlighet att vid ett senare tillfälle, under en begränsad tid (max 1 dag), få kvittera ut en legolåda. För att hinna provköra och redovisa laborationer och/eller projekt under denna dag ställs det krav på godkänd rapport och uppvisande av implementation innan gruppen tillåts kvitter ut en låda.
I de fall där det förekommer att material saknas i lådor blir gruppen skyldig att ersätta det saknade. I de fall där gruppen lämnar in lådan försent kommer inte laborationspoäng att rapporteras in innan Lego Mindstorm lådan är inlämnad.

Utvecklingsmiljön, finns att hämta på kursens hemsida. Det är en komprimerad fil som innehåller all dokumentation och verktyg som behövs för laborationerna. Filen är komprimerad med ett ZIP program så att det lätt skall kunna packas upp. Utförlig beskrivning av utvecklingsmiljön finns i kapitlet ***Ställa in utvecklingsmiljön.***

I dessa dokument finns information som behövs för att kunna lösa laborationerna,
Laborationsanvisningar: finns i filen Laborationsmaterial.doc,
Asterixmanual: finns i filerna asterix.ppt och asterix.ps dessa dokument uppdateras inte, om laborationsanvisningarna och asterixmanualen inte stämmer överens är det laborationsanvisningarna som gäller.
Librarymanual för Lego Mindstorm: finns i filerna librarymanual.doc.

För den som vill läsa mer eller behöver mer information för att lösa laborationerna finns det tekniska rapporter om Asterix och Obelix att läsa.

Laboration

Målsystemet för laborationen och projektet är LEGO Mindstorm. För att få önskat beteende hos målsystemet kommer ett realtidsoperativsystem att användas. Utförliga instruktioner hur man programmerar och vilka funktionaliteter som ni kan använda finns i laborationsanvisningarna och i manualerna.

Det ni behöver för att utföra laborationerna är LEGO Mindstorm och en utvecklingsmiljö. Lego Mindstorm kommer att finnas för utkvittering hos någon av laborationsassistenterna. För att ni ska kunna göra laborationerna även utanför schemalagd tid så återlämnas hårdvaran vid kursens slut.

Den första laborationen är en introduktion till Asterix och LEGO Mindstorm. Syftet med denna laboration är att bekanta sig med Asterix samt hårdvaran. Laborationen går även ut på att lära sig utvecklingsmiljön och göra en nedladdning av systemet till målsystemet. Ta tillfället att bekanta er med systemet ordentligt på laborationen så ni kommer att spendera mindre tid på projektet.

I laborationen ska ni även använda de teoretiska kunskaper som ni lärt er. Ni ska designa och implementera task som kan användas i ett realtidssystem. Tasken ska inte designas för ett specifikt system utan tasken ska kunna användas i ett godtyckligt realtidssystem designat för Asterix. För att lösa uppgiften ska ni använda realtidsoperativsystemet Asterix och Lego Mindstorm med de olika sensorerna som följer med.

Projekt

I projektet ska ni bygga en bil som ska klara av att hitta och följa en bana som markeras av en svart linje på en vit bana. Samtidigt, ett antal krav måste uppfyllas om projektet ska bli godkänt. Byggbeskrivning för bilen finns i manualen för Lego Mindstorm.

OBS GLÖM INTE ATT LÄSA FAQ PÅ KURSENS HEMSIDA OBS

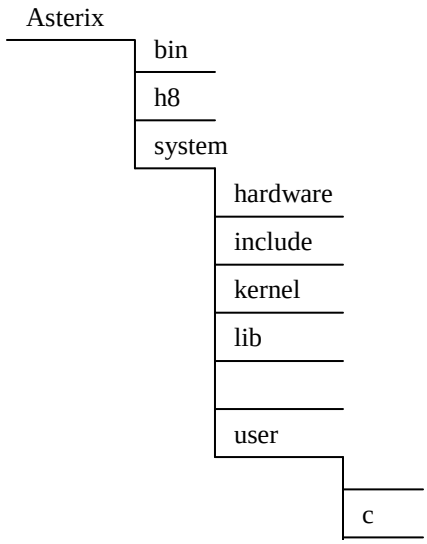
Innan ni frågar laborationsassistenter läs detta dokument noggrant och läs FAQ för laborationerna på kursens hemsida.

Grunder i Asterix

Schemat bestämmer vilka tasks och resurser som finns i ett specifikt system. Det är här som, t.ex., prioriteter till tasks sätts, eller kommunikation mellan tasks specificeras, t.ex., semaforer etc.

Katalogstruktur

Hela konfigureringen av Asterix systemet är uppbyggt runt en katalog- och fil-struktur.



Config.obx
Denna fil beskriver vilka task och resurser som kommer att användas i Asterix. Filen **måste** ligga i ***enhet:\asterix\system\user*** - katalogen annars kommer ingen kompilering av kärnan att ske (Asterix kompilerar ihop task och resurser med övriga delar av kärnan).

Schemaläggningsstrategi
OBS: Endast Fixed Priority Scheduling (FPS) får användas som schemaläggningsstrategi i laborationen och projektet (d.v.s ingen statisk schemaläggning, även kallad off-line schemaläggning, får användas).

Definition av resurser i konfigurationsfilen och användning av resurser
Syntax för hur man definierar resurser man vill använda för sitt system finns i filen **asterix.ppt**.
Nedan finns en utförlig beskrivning för hur man använder allokerade resurser.

För att läsa innehållet i en wait&lockfree kanal används:

```
buffertype_t    *buf=getReadPointerWF( wf_channel );
if(*buf > 42)
...
```

För att skriva till en wait&lockfree kanal används:

```
buffertype_t    *ut=getWritePointerWF( wf_channel );
int value = 42;
...
writeChannel( ut, value, wf_channel );
```

För att ta en semafor används följande anrop:

```
getSemaphore(sem_1);
```

För att släppa en semafor används följande:

```
releaseSemaphore(sem_1);
```

För att sända en signal används följande:

```
raiseSignal(sig_2);
```

Tänk på att det inte går att skicka med något meddelande till det task(s) som väcks av signalen.

Task

Asterix stöder endast task som är definierade med en angiven deadline som måste garanteras, d.v.s. tasks med *hård deadline*. Alla task i systemet definieras i en konfigurationsfil: *system/user/config.obx*.
Funktionaliteten till ett task kodas i en taskrutin och måste ha följande inparameter:

```
void routine_A(void *param)
{

...

}
```

Dessa taskrutiner läggs enklast i filen *user\c\user.c*. Det går även att lägga till nya filer i *user\c*. Dock måste filen os_tasks.h inkluderas i varje fil och varje ny fil måste läggas till i make-filen så att den nya filen kompileras och länkas korrekt. Lab-assistenterna rekommenderar att ni har alla taskrutiner i *user.c*.

Med andra ord, dom enda filerna ni behöver jobba med är user.c (för taskrutinerna) och config.obx (för beskrivning av task och resurser)

Datatyper som stöds

Eftersom de datatyper som anges i C-standard ej är lika stora på en Intel-processor och en Hitachi-processor (som det sitter i Lego Mindstorm), är följande datatyper fördefinierade:

Namn	Typ	Stöds av H8300
int8	8-bit signed	Ja
uint8	8-bit unsigned	Ja
int16	16-bit signed	Ja
uint16	16-bit unsigned	Ja
int32	32-bit signed	Nej
uint32	32-bit unsigned	Nej

Dock går det bra att använda vanliga c-typer. Tänk bara på att storleken kanske inte är densamma som ni är vana vid.

Felhantering

När kärnan upptäcker att en resurs används av ett annat task än det/de som har tillåtelse att använda resursen, eller om ett task missar sin deadline kommer en felrutinen för det tasket som orsakade felet, att starta. Om ingen rutin angivits i taskspecifikationen kommer default-hanteraren att gå igång.

Default-hanteraren kommer att skriva ut felkoden (se filen system\kernel\include\os_errno.h) vilket task-id som orsakade felet samt vid vilken tidpunkt. Därefter kommer processorn att resettas. För att hinna läsa av felkoderna innan processorn resettas stannas exekveringen,

för varje nytt värde som man vill avläsa måste man trycka på *VIEW* knappen på RCX-enheten.

Klockupplösning

Upplösningen på klockan anges med RESOLUTION och sätts innan tasken. Värdet som anges är i mikrosekunder (min=2 µs , max=130ms). Även om det minsta giltiga värdet är 2µs så måste man tänka på vad som är lämpligt för den processor man använder, för Hitachi H8/300 är det lämpligt att använda 5 ms som minsta upplösning. En klockupplösning på 10 millisekunder skulle då skrivas:

```
RESOLUTION = 10000;
```

Upplösningen på klockan är det som driver realtidssystemet framåt. Genom att ange en *resolution* på 10 ms så kommer systemet drivas framåt med ett tick var 10:de ms.

Hårdvaran

För att komma åt de olika funktionerna som anges i RCX Library Manual, måste följande header-filer inkluderas där funktionerna används:

rcx_display.h	-	Funktionalitet för att kunna skriva ut tecken på LCD-displayen.
rcx_button.h	-	Access av knappar på RCX-enheten (view, prgm, run).
rcx_motor.h	-	Funktionalitet för aktuering av motorer.
rcx_sensor.h	-	Funktionalitet för hantering (läsning/aktivering) av sensorer.

Grundschemat (config.obx)

Grundschemat för ett asterix system är en mall som ska användas i laborationen och projektet, denna fil följer med i utvecklings miljön och har sökvägen:
asterix/system/user/config.obx

```
SYSTEMMODE = NORMAL;
RAM = 1000;

MODE mode_1{

RESOLUTION = 10000;

/* System tasks, do not edit or remove */
HARD_TASK APERIODIC IRQ0{
        ACTIVATOR = IRQSIG0;
        OFFSET = 0;
        DEADLINE = 50;
        PRIORITY = 100;
        STACK = 50;
        ROUTINE = irq0_isr;
        ERR_ROUTINE = irqerror;
};
HARD_TASK APERIODIC IRQ1{
        ACTIVATOR = IRQSIG1;
        OFFSET = 0;
        DEADLINE = 50;
        PRIORITY = 99;
        STACK = 50;
        ROUTINE = irq1_isr;
        ERR_ROUTINE = irqerror;
};

/* Put user-tasks here */
HARD_TASK APERIODIC idle{
        ACTIVATOR = 0;
        OFFSET = 0;
        DEADLINE = 0;
        PRIORITY = 0;
        STACK = 50;
        ROUTINE = idletask;
};
/* Put waitfree-communication here */
/* default signals for irq, do not edit */
        SIGNAL IRQSIG0{
                USER = IRQ0;
        };

        SIGNAL IRQSIG1{
                USER = IRQ1;
        };
/* Put signals here */
/* Put semaphores here */
};
```

Ställa in utvecklingsmiljön

Ni är inte bundna till en någon editor eller IDE när ni ska göra laborationerna, utan ni har stor frihet när det gäller val av editorer och kommandofönster. Nedan kommer en beskrivning hur man utföra laborationerna med hjälp av en enkel text-editor och en dos-prompt i Windows. Det som labassistenterna rekommenderar är att välja en enkel texteditor för att editera källkodsfilerna och ett kommandofönster för att kompilera och länka systemet. **Glöm ej** att läsa avsnittet *Troubleshooting* och *FAQ* på kursens hemsida.

Hämta filer

1. Hämta Asterix-systemet. (Finns att hämta på *http://www.idt.mdh.se/kurser/ct3200*).
2. Packa upp zip-filen (asterix.zip) på ***d:***
3. Kontrollera att det finns en katalog med namnet ***d:\asterix***

Ställ in utvecklingsmiljön

1. Öppna en kommandotolk (DOS-prompt).
2. Ändra katalog till ***d:\asterix***
3. Ge kommandot ***bin\init***
4. Ge kommandot ***compile***
5. Om kompileringen går bra ska en fil ha producerats i asterix\system-katalogen, kernel.srec. Det är denna fil som laddas ner till RCX-enheten.
6. Make-programmet som används terminerar inte alltid. Ta därför för vana att titta i aktivitetshanteraren och avsluta gamla MAKE-program.

För er som inte vill använda compile.bat går det bra att köra make ifrån asterix\system-katalogen.
Alla filer i alla underkataloger till asterix\system måste kompileras och länkas ihop (se ***makefile*** och ***makefile.mk*** i asterix\system för detaljer).

Nedladdning

Följ punkt 1-3 i **Ställ in utvecklingsmiljön**

4. Koppla in IR-tornet i COM1. Om COM1 är upptagen, måste dl.bat modifieras.
5. Kontrollera att det finns en fil kernel.srec i d:\asterix\system. (Om filen saknas ge kommandot ***compile***)
6. Starta LEGO Mindstorm RCX.
7. Kör kommandot ***dl***

Troubleshooting

eller : 'vad gör man när ingenting fungerar ?'

1. Det går inte att kompilera!
   - Ligger filerna i d:\asterix ? Om inte måste makefiler ändras.
   - Är alla miljövariabler (asterix och path) korrekta? (Se filen asterix\bin\init.bat
   - Är kompilatorn rätt inställd? (se ”Ställa in utvecklingsmiljön”)
   - Ligger config.obx i rätt katalog? (asterix\system\user\ )
   - Windows skapar ibland filer med stora/små bokstäver! Kontrollera att filerna asterix\system\kernel\c\tasks.c och asterix\system\kernel\include\tasks.h är skapade som små!

2. Det går inte att ladda ner!
   - Är kabeln rätt inkopplad (IR-tornet har en liten lampa som lyser när dl.bat körs)?
   - Urladdade batterier?
   - Störningar ifrån grannen ?
     ”Göm” torn och RCX i Lego-lådan när ni laddar ner.
     Under tornet finns en liten knapp. Se till att den är satt åt vänster.

3. Beteendet på RCX'en är inte det tänkta!
   - Rita upp det tänkta beteendet och jämför med config.obx-filen.
   - Har ni utgått ifrån grundschemat?
     Om inte kan avbrotten ställa till problem.
     Signaler måste specificeras efter avbrottsignalerna (irqsig)!!!

4. RCX'en har låst sig!
   - Ta ur batterierna! Det är enda sättet att starta om.
   - Kontrollera ev. minnesöverskrivningar (stackstorlekar, pekarfel etc.).

5. Våran kod funkade vid förra laborationstillfället varför funkar det inte idag!
   - Spara undan de filer ni har ändrat, oftast är det bara *user.c* och *config.obx*.
   - Hämta *asterix.zip* från kursens hemsida och packa upp programvaran i rekommenderad katalog. Flytta sen över era sparade filer till asterixkatalogen.

Att tänka på:

- Make-programmet som används är lite instabilt. Det kan hända att ingen nedladdningsbar fil skapas. Kör ***compile*** igen bara. Make skapar även processer som ibland inte tas bort. Ta därför för vana att titta i processlistan och ta bort gamla make-processer.
- Det kan vara svårt att lokalisera programmeringsfel. Om ingen nedladdningsbar fil har skapats (kernel.srec), kontrollera så att inga fel inträffat vid kompileringen av era filer. Ev. fel syns vid stderr med ert filnamn som orsak.
- Ibland kan det vara nödvändigt att kompilera om hela systemet. Genom att gå till katalogen asterix\system och ge kommandot ***make clean*** eller ***make realclean*** så kompileras **hela** systemet om vid nästa kompilering.

OBS:

- **Efter varje arbetspass spara de filer ni har ändrat till eran hemkatalog.**
- **Vid början av varje nytt arbetspass bör ni ladda ner *asterix.zip* från kursens hemsida och packa upp programvaran i rekommenderad katalog. Flytta sen över era sparade filer från hemkatalogen till asterixkatalogen.**

Fråga laborationsassistenten endast om ni inte lyckas lösa problemen med ovanstående förslag!!!

Laboration

Inledning

Syftet med denna uppgift är att få en inblick i hur de olika delarna i Asterix kan användas samt hur de konfigureras.

Redovisning

Laborationen redovisas för laborationsassistent genom att alla gruppdeltagare är med i en diskussion om gruppens implementation samt en demonstration för laborationsassistent. Implementationen godkänds endast om gruppen kan övertyga laborationsassistenten att implementationen följer grundläggande regler för c-programmering och realtidsprogrammering och alla gruppdeltagare klart och tydligt kan redogöra för gruppens lösningar.

Laborationer kommer inte att godkännas där:

- Kod är svårläst och ostrukturerad, d.v.s ”spaggetikod” godkänns inte.
- Kod är onödigt komplicerad, villkor är ett bra verktyg i c-programmering men kan göra koden svårläst och ostrukturerad. Ställ er själva frågan: kan vi utöka vårt system med ett eller flera task med enklare kod som löser problemet.
- Sensorer läses av eller motorer aktueras i fler än ett task.
- Taskattribut skiljer sig från de i laborationsanvisningarna angivna taskattribut.
- Lösningar inte kan motiveras.

Förberedelser

Läs igenom dokumentationen om Asterix och Obelix, speciellt de delar som tar upp den syntax som används i konfigurationsfilen och tjänster som tillhandahålls av operativsystemet.
Läs igenom ”Grunder i Asterix”.
Ställ in utvecklingsmiljön, kompilera samt ladda ner testprogrammet.

Utgå alltid ifrån grundschemat!

Finns i filen: ***asterix/system/user/config.obx***

A) Schema

Skapa ett schema som består av fyra task.

Task A ska skriva 111 på displayen.
Task B ska skriva 222 på displayen.
Task C ska skriva 333 på displayen.
Idle-tasket ska inte utför något arbete.

Task	Period	Deadline	Prioritet
A	10	10	30
B	20	20	20
C	30	30	10
Idle	0	0	0

Upplösning på hårdvaruklockan ska vara 10 millisekunder.

1. Skapa de task-rutiner som behövs och gör lämplig kod för dessa.
2. Konfigurera Asterix för uppgiften genom att utgå ifrån grundschemat och lägga till det som behövs. Lämplig stackstorlek kan vara 30-50. Tänk på vilket namn ni skall ange som task-rutin när ni konfigurerar schemat.
3. Glöm inte att inkludera de ”h” filer som behövs.
4. Kompilera systemet.
5. Ladda ner systemet på RCX-enheten.

B) Deadlinemiss

Lägg in en oändlig loop sist i Task C.

- Vad händer?

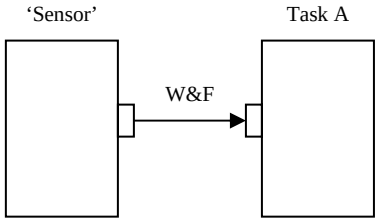
- Vilken sifferkombination skrivs ut (Glöm inte *VIEW*-knappen)?

- Vad betyder 1001? (Tips, i vilken fil finns fel-koder definierade.)

C) Wait&lockfree kanaler

Det finns olika sätt att låta task kommunicera med varandra. Alltifrån delade variabler till avancerande algoritmer för att garantera att task alltid läser senast skrivna värdet. Asterix stöder wait&lockfree-kanaler som är en bufferbaserad lösning. Vid period start av varje task, tilldelas tasket en buffer som innehåller det senast skrivna värdet (om tasket vill läsa datat) eller en ledig buffer om tasket är en skrivare.

Skapa två task (+ idletask) varav det skrivande tasket ska simulera en sensor. Det räcker om själva sensorvärdet endast är en statisk lokal variabel som räknas upp varje period.
Innehållet i 'sensor' ska skickas till det läsande tasket via en wait&lockfree kanal. För att se resultatet av sensorn bör det läsande tasket skriva ut det på displayen.



Task	Period	Deadline	Offset	Prioritet
Sensor	20	20	0	30
Task A	20	20	0	20
Idle	0	0	0	0

- Finns det något problem med att låta tasken ha samma periodicitet och samma offset?

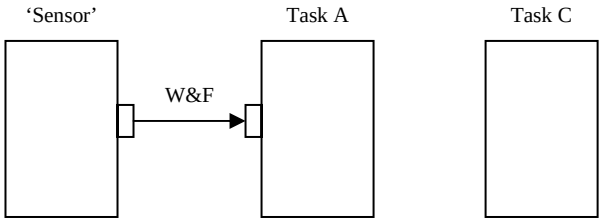
- Hur löser man det?

(Tips, hur sker tilldelningen av bufferplatser i en kanal)

D) Synkronisering

När tasks försöker lösa en gemensam uppgift, måste ibland tasken synkroniseras. Precis som i fallet med wait&lockfree-kanaler finns det olika tekniker för detta.

Anta samma scenario som i wait&lockfree-uppgiften ovan, d.v.s. en 'sensor' som läses av ett task A. Värdet som läses kommer att användas av ytterligare ett task C. Eftersom 'sensor' är en simulerad sensor och inte tillåter mer än en läsare, kommer all kommunikation med sensorn att ske genom task A.
För att göra uppgiften än mer verklighetstrogen är bara sensor-värden jämnt delbara med 10 korrekta och det är endast de värden som task C ska få tillgång till. Task A blir på så sätt en enkel device-driver för sensorn.



Task	Period	Deadline	Offset	Prioritet
Sensor	20	20	0	30
Task A	20	20	0	20
Task C	20	20	0	10
Idle	0	0	0	0

Synkronisera task C med avseende på när ett giltigt värde på sensorn har lästs av task A, med hjälp av olika tekniker.

1. Delad variabel skyddad med en semafor.

2. Wait&lockfree-kanal

3. Aperiodiska signaler (Task C är ett aperiodiskt task).

E) Design och implementation av task

Denna uppgift går ut på att skapa olika hjälpmedel samt kan användas godtycklig tillämpning i Asterix.
Tillämpningen kommer att vara det projekt ni kommer att göra, men de hjälpmedel ni konstruerar ska även kunna användas i andra applikationer.

E.1) Klocka

I Asterix finns det ingen extern klockhantering. Eran uppgift blir att implementera en klocka. Uppdatering av klockan ska ske så att en avläsning vid godtycklig tidpunkt av klockan ska returna rätt sekund. Ingen klocka som implementeras i mjukvara kan ge exakt tid, motivera vid redovisningen hur stor avvikelse eran implementation av klockan tillåter och om det är en rimlig avvikelse.
Vid redovisning måste ni också motivera val av prioritetsnivå och periodtid. Svara också på följande fråga: Om en klocka implementeras som ett periodiskt task med en periodtid på 1000 ms, vid varje ny period kommer tasket att exekvera och då räknas en sekundvariabel upp med 1. Denna klocka kommer att visa rätt tid men med hur stor avvikelse kommer klockan att uppdatera sekunderna (i värsta fallet) jämfört med en exakt klocka?

Minimikravet är att klockan ska kunna hantera timmar, minuter och sekunder, d.v.s. för att implementera en 24 timmars klocka behöver ni tre heltalsvariabler. Observera dock att endast 4 siffror kan skrivas ut på LCD'n (ex: minuter och sekunder). Dessutom, i implementationen av klockan, ni bör använda systemanropet getTicks() som returnerar många ticks som inträffat sen systemet startades (dvs inte bara en variabel som räknas upp periodisk). Avläsning av tiden måste vara ***atomär***.

Att tänka på:

- Delade resurser måste skyddas.
- Ni kommer aldrig att kunna få en klocka som går exakt. Realtidsklockan i en PC uppdateras c:a 18 ggr i sekunden.
- Största datatypen är 16-bitars heltal (uint16, int16) även om kompilatorn har stöd för datatyper med fler antal bitar, t.ex 32-bitars heltal.
- Systemanropet getTicks() talar om hur många ticks som inträffat sen systemet startades, dock är det endast en 16-bitars räknare, d.v.s. den slår runt vid 65535.
- En läsning från en 16-bitars variabel är inte atomär, även om det bara är en c-instruktion kommer det att genereras ett flertal assemblerinstruktioner vid kompilering.
- I projektet kan ni behöva en klocka med högre noggranhet, d.v.s ni kanske också behöver ett 16-bitars heltal för att lagra millisekunder.

E.2) Device drivers för displayen och klockan

Ett problem som finns med displayen är att bara en task kan skriva till den. Detta medför att displayen måste skyddas på något sätt. På samma sätt, måste uppdateringen av klockan skyddas från dom task som vill läsa den. Skapa därför en device driver som tar hand om utskrifter och uppdaterar LCD-displayen på ett korrekt och säkert sätt, och en device driver för klockan.
Det räcker om ni implementerar era device drivers som funktioner som ska vara gränssnittet mellan devices (klocka eller display) och användaren.

E.3) Sensorer och motorer

Ljussensorn är designad att användas i flera syften men det ni först och främst ska använda den till är att detektera nyanser i färger. Tänk på att innan ni läser av sensorn måste den

göras passiv, för att efter läsningen göras aktiv igen (setSensorPassive(), readSensor(), setSensorActive()).
För att kunna detektera sammanstötningar kan en trycksensor användas. Till skillnad från ljussensorn behöver inte trycksensorn göras aktiv/passiv utan kan läsas av direkt med readSensor().
Sensorerna kopplas in på portarna 1,2 eller 3.

De två motorerna kopplas in på portarna A, B eller C och styrs med kommandot setMotor().

Prova att använda sensorerna så att ni kan detektera en aktiverad trycksensor samt skilja på ljust och mörkt med ljussensorn. Använd banan som följer med Lego Mindstorm för att kalibrera ljussensorn. Dock kan ljussensorn vara svåravläst. Börja därför med att försöka hitta något utgångsvärde som ni sedan kan finjustera.

Styr motorerna framåt, bakåt eller stå still, beroende på de olika sensorvärdena. Försök att tänka på att göra ett modulärt program och inte lösa allt i ett enda task (se krav nedan). Det kan t.ex. vara viktigare att undvika en krock än att följa en bana.

Ni ska nu designa task för sensorerna och motorerna, designen ska klara följande **krav**:

- Aktuering av motorer måste ske kontinuerligt med maximum 5ggr/sek.
- Sensorer får aj avläsas i samma task där aktuering av motorer sker.
- Endast en sensor får avläsas i en task.
- Ingen logik eller beräkningar får implementeras i motor- och sensor-tasken. Dock får konvertering av indata till passande format göras.
- Alla lösningar måste ske med hjälp av schemaläggning och/eller med hjälp av klockan ni implementerade i E.1., dvs, lösningar som ”*sväng under 3 perioder*” kommer inte att godkännas.

För att testa tasken ni skapat bör ett eller flera temporära task läggas till systemet som på ett övertygande sätt verifierar funktionalitet och tidskrav på motor- och sensor-tasken.

Projekt

I projektet ska ni konstruera en bil enligt byggbeskrivningen i manualen för Lego Mindstorms eller enligt era egna design idéer. Er konstruktion måste uppfylla ett antal krav för att bli godkänd.

- **För att få godkänt måste projektrapporten vara skriven enligt Thomas Larsson anvisningar för rapportskrivning, se rapportmall.**
- **Ni måste kunna motivera varför ert system fungerar, både när det gäller det funktionella beteendet och det temporala beteendet.**
- **Ni måste kunna redogöra (vid redovisning och i rapport) för vad som gör just ert system till ett realtidssystem.**

Bilprojektet:

I detta projekt ska ni bygga en bil som klarar av att hitta en bana som markeras av en svart linje och sedan följa den svarta linjen. Byggbeskrivning för bilen finns i manualen för Lego Mindstorm.

Grundkrav för bilprojektet:

- De hjälpmedel som ni implementerade i uppgift E i laborationen måste användas där det är möjligt. T.ex. om ni ska uppföra en uppgift under en bestämd tid, t.ex. svänga med bilen i 50 ms, så måste ni använda klockan från laborationen, det är alltså inte tillåtet att använda en räknare för att svänga i *n* antal perioder.
- Aktuering av motorer: **max 5 gånger** per sekund.
- Vid start måste bilen klara av att hitta den svarta linjen från ett avstånd på **upp till 1 meter.**
- Banan måste klaras av på **max 30 sekunder** efter det att den svarta linjen hittats första gången. (naturligtvis är tiden satt m.a.p. att det inte finns hinder på banan)
- Bilen måste klara av att **köra runt ett hinder** som placeras på den svarta linjen, det är alltså inte tillåtet att vända bilen 180º och sedan fortsätta åt motsatt håll.

I enskilda fall, beroende på valet av den mekaniska konstruktionen, ni kan upptäcka att kraven är orimliga (p.g.a., till exempel, hårdvarurelaterade begränsningar). I sådana fall måste ni ha en väldigt bra motivering för varför dom angivna kraven inte kan uppfyllas, samt definiera era egna krav.

Tips: Eftersom det finns krav på maximal varvtid för ovalbanan och hur mycket bilen får avvika från svarta linjen så gäller det att välja rätt utväxling och välja rätt däckstyp.

Vid redovisning kommer bilen att placeras ut av laborationsassistenten någonstans på eller intill den medföljande ovalbanan, så ni kan inte anta att ni börjar en körning med bilen på den svarta linjen. När bilen har hittat den svarta linjen kommer laborationsassistenten påbörja tidmätning för att se om bilen klara av att köra runt banan under angiven maxtid. När bilen har klarat av tidmätningen kommer laborationsassistenten att placera ut ett hinder på banan. Klarar bilen att hantera hindret enligt kraven är demonstrationen godkänd.

OBS. Om ni har andra förslag till projektet, t.ex. robotarmen eller liften, så får ni själva specificera kraven som konstruktionen måste uppfylla. Tänk dock på att svårighet/komplexitetsnivån på kraven måste motsvara kraven för bilprojektet. Kontakta labbassen för vidare information.

Rapportering av projektuppgift

Följande fyra moment (**A**, **B, C** och **D**) måste uppfyllas för att rapport och implementation av projektuppgiften skall godkännas. Tänk på att redan från och med del A så ska det dokumentet vara utformat som en rapport.

A.
Specifikation :

- En översiktlig beskrivning .
- Funktionalitet (beskrivning av vad konstruktionen kan utföra) .
- Hårdvarubeskrivning .
- Testfall för funktionellt och temporalt beteende.

Labassistenterna kommer inte att godkänna del B för grupper där del A inte är godkänd. Givetvis får ni jobba vidare med del B men var beredda på att ni kan få komplettering på del A.

B.

- Kravanalys (mjukvara), tidskrav i systemet.
- Design, enligt RTT-modellen.
- Punkterna I-VII, sidan 166 i kurskompendiet. **OBS** Tänk på att ni endast får använda Fixed Priority Scheduling. Detta kommer att påverka valet av strategi för test av schemaläggningsbarhet.
- Temporal analys baserat på uppskattade exekveringstidsbudgetar.
- Fler/utökade testfall för funktionellt och temporalt beteende.

Labassistenterna kommer inte att godkänna del C för grupper där del B inte är godkänd. Givetvis får ni jobba vidare med del C men var beredda på att ni kan få komplettering på del B.

C

- Punkterna VII-IX, sidan 167 i kurskompendiet.

Dessutom måste nedanstående ingå i projektrapporten:

- Temporal analys baserat på uppmätta exekveringstider.
- Funktionell och temporal testning, ni måste motivera varför systemet fungerar och inte havererar.
- Slutsats innehållande bland annat:
    - Problem
    - Lärdomar

D

- Demonstration av projektet.

Labassistenterna kommer inte att godkänna del D för grupper där del C inte är godkänd. Givetvis får ni jobba vidare med del D men var beredda på att ni kan få komplettering på del C.

Obs! Om man får komplettering på rapporten så måste både en ny version och den gamla versionen lämnas in. Den gamla versionen måste vara den som innehåller lab-assens rättningsanteckningar.

Flyttalsberäkningar:
Tänk på att flyttal ej stöds av Hitachi H8/300. Om ni vill använda flyttalsberäkningar måste ni lösa det själva (Tips! Fix Point Arithmetic).

Analys av exekveringstider:
Ni kommer att analys exekveringstider med hjälp av ett WCET-analysverktyg. Mer information om detta verktyg finns i laborationsmaterialet om temporalanalys av realtidssystem.