Examensarbete D, 20p.
Civilingenjörsprogrammet i Teknisk Datavetenskap
Institutionen för datavetenskap
Umeå Universitet

Utvärdering av realtidsoperativsystem för motorstyrning

Håkan Andersson
Tfn: 0707-139144
c96han@cs.umu.se

22 December 2005

Jan-Erik Moström,
handledare
Umeå Universitet

Adam Klotblixt, handledare
Atlas Copco Tools and
Assembly Systems AB

Abstract

A control system of an electric motor at Atlas Copco Tools AB is implemented with two microprocessors. One processor handles the real-time critical code and the secondary processor manages the rest of the application like networking, GUI and application programs. This thesis investigates if it is possible to implement the system with one single processor under a real-time operating system. The extraordinary challenge is the desired sample rate of the control loop, 10 kHz. Tests are performed with Windows CE as RTOS and a 166MHz ARM processor. The evaluation shows that it is possible to implement the system. Furthermore, the possibilities to use embedded Linux as an operating system are investigated. The authors experiences and personal opinion of Windows CE and Linux is also presented in the report. The reason for investigating Windows CE and Linux is the large amount of available applications and communication protocols.

Sammanfattning

Ett befintligt system för styrning av en elektrisk motor är idag implementerat med två mikroprocessorer. En undre processor hanterar den realtidskritiska motorstyrningen och den övre processorn applikationen i övrigt. Rapporten undersöker om det är möjligt att implementera systemet med en mikroprocessor under ett realtidsoperativsystem. En särskild omständighet är den höga samplingsfrekvensen för motorstyrningen, 10 kHz. Mätningar utförs med Windows CE som realtidsoperativsystem och en 166 MHz ARM920T processor. Tester visar att det är möjligt att realisera systemet med en processor, dock med vissa förbehåll. Vidare undersöks möjligheterna att använda någon form av Linux som RTOS.
I rapporten sammanställs också författarens synpunkter på Windows CE och Linux. Anledningen till att företaget är intresserat av Windows CE och Linux är den mängd färdiga applikationer och kommunikationsprotokoll som finns tillgängliga.

Innehållsförteckning

1 Introduktion

1.1 Bakgrund

Atlas Copco Tools AB utvecklar elektriska verktyg för momentåtdragning. Verktygen används huvudsakligen inom bilindustrin vid montage av skruvar, muttrar och bultar.

Förutom huvuduppgiften att utföra en momentdragning med korrekt moment finns en mängd andra funktioner i verktygen. Varje momentdragning lagras i en databas för att möjliggöra spårbarhet. Kommunikation sker med TCP/IP, fältbussar[1] och seriekommunikation. Funktionskraven ändras ständigt och varierar mycket mellan olika kunder.

Idag är verktyget uppbyggt med två mikroprocessorer, en undre för den realtidskritiska motorstyrningen samt en övre processor för övriga applikationer. Den undre processorn är oftast en digital signalprocessor, DSP med särskild utvecklingsmiljö medan den övre processorn är av mer generell typ.

Den snabba utvecklingen av processorer och realtidsoperativsystem skulle kunna göra det möjligt att implementera systemet med en processor och ett realtidsoperativsystem. Fördelarna skulle vara många:

- Endast en utvecklingsmiljö krävs.
- Utvecklingsmiljöerna för standardprocessorer är oftast bättre än de för DSP.
- Endast en mikroprocessor behövs.

Rapporten har till syfte att undersöka om detta är möjligt. Realtidsoperativsystemet Windows CE väljs för implementering och test. Dessutom undersöks möjligheten att använda någon form av Linux med realtidstillägg dock utan implementering och test. Anledningen till valet av Windows CE och Linux är den mängd applikationer och kommunikationsprotokoll som finns tillgängliga för dessa operativsystem.

En ordlista finns tillgänglig i Appendix B

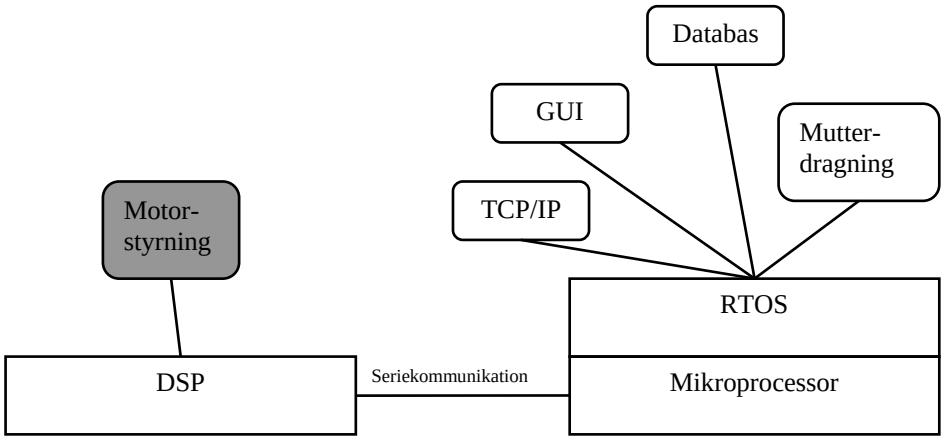
1.2 Rapportens uppbyggnad

Efter problembeskrivning och teori redovisas mätresultat med Windows CE som RTOS. Vidare utreds olika alternativ med Linux som RTOS. En litteraturuppgift redovisas där målsättningen är att hitta exempel på RTOS-baserade inbyggda system med hög samplingshastighet. Slutligen presenteras en sammanställning av författarens erfarenheter och åsikter om Windows CE och Linux.

[1] En fältbuss är ett kommunikationsprotokoll avsett för kommunikation direkt med givare och ställdon av olika slag, s.k. distribuerad I/O. Exempel på fältbussar är Profibus, Can och Interbus-S.

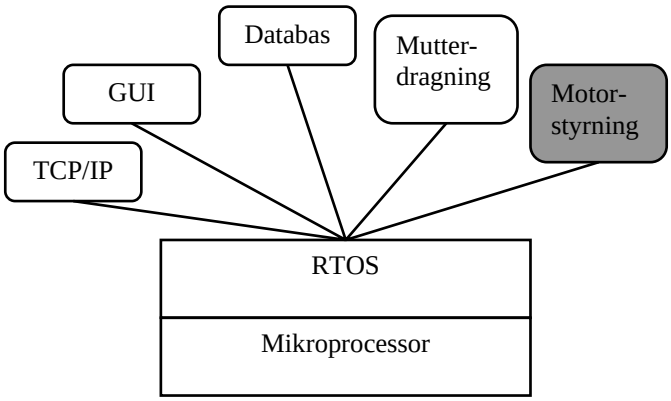
2 Problembeskrivning

Idag är systemet uppbyggt enligt Figur 1. Motorstyrningen hanteras av en separat DSP utan RTOS. Applikationen i övrigt ligger på en annan processor med ett RTOS. På detta sätt garanteras att regleralgoritmerna för motorstyrningen alltid får exekvera vid önskade tidsintervall.



Figur 1: Befintligt system

I rapporten undersöks om det är möjligt att implementera hela systemet med en mikroprocessor och ett RTOS enligt Figur 2. Algoritmerna för motorstyrningen skall exekvera med 10 kHz frekvens, dvs var 100:e mikrosekund. Den händelse som startar exekveringen av motorstyrkoden kommer att vara något slags avbrott från hårdvara, t.ex. en hårdvarutimer. Då ett avbrott genereras måste realtidsoperativsystemet snabbt avbryta nuvarande exekvering och börja köra motorstyrkoden i avbrottsrutinen.

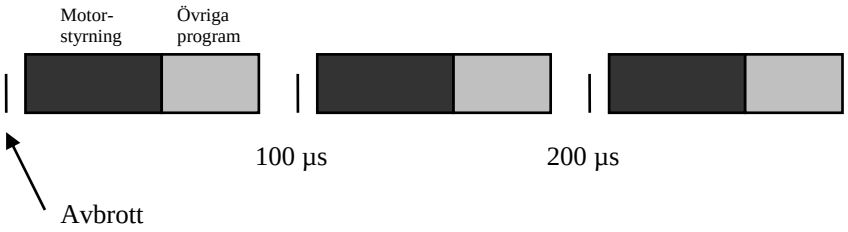


Figur 2: Vision

Det ideala operativsystemet har mycket bra realtidsprestanda för att kunna hantera motorstyrningen samt kraftfullt stöd och goda utbyggnadsmöjligheter på applikationsnivå för GUI, kommunikationsprotokoll, databaser, m.m.

2.1 Huvudfrågeställning

Kan vi garantera att realtidskritisk motorstyrkod alltid kommer att kunna exekvera inom önskat tidsintervall (var 100:e µs) enligt Figur 3, oavsett systemlast ?



Figur 3: Exekvering av algoritmer för motorstyrning.

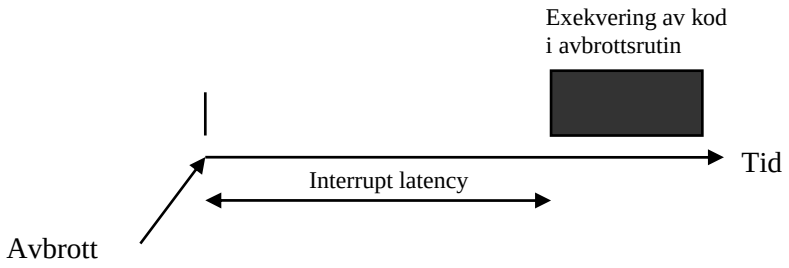
Avgränsningar

Rapporten fokuserar på mätningar relaterade till motorstyrningen. Prestanda på övriga applikationer i systemet studeras inte ingående.

3 Teori och begrepp

3.1 Interrupt latency

Ett viktigt prestandamått på ett RTOS är tidsfördröjningen från det ett hårdvaruavbrott genereras tills programkoden i avbrottsrutinen börjar exekvera. Den engelska benämningen är *interrupt latency*. Särskilt intressant är *worst case interrupt latency*, maxvärdet på interrupt latency enligt Figur 4.



Figur 4: Interrupt latency

Interrupt latency för ett RTOS beror först och främst på vilken mikroprocessor och klockfrekvens som används. Få RTOS-leverantörer anger *worst case interrupt latency* för en viss processor. Ofta uppges ett medelvärde för någon standardprocessor. Behövs mer uppgifter måste egna mätningar utföras genom att generera en mängd avbrott och mäta interrupt latency för varje avbrott. Mätningarna blir dock inte exakta då det är praktiskt omöjligt att testa alla möjliga konfigurationer av operativsystemet.

Vad orsakar interrupt latency i ett RTOS ?

- Utan ett RTOS anropas avbrottsrutinen direkt från avbrotts-vektortabellen. Med ett RTOS däremot anropas kärnan i realtidsoperativsystemet först. Kärnan anropar därefter respektive avbrottsrutin vilket tar längre tid.

- För att åstadkomma exklusiv åtkomst av delad data s.k. ömsesidig uteslutning stängs avbrottshanteringen av på flera ställen i programkoden. Då avbrottshanteringen stängs av kan inga andra avbrottsrutiner, processer och trådar börja exekvera. Detta innebär att koden som stängde av avbrottshanteringen garanterat får exekvera tills avbrotten slås på igen. Exempel på kod som stänger av avbrott är kärnan i realtidsoperativsystemet och drivrutiner. Nackdelen med att stänga av avbrotten är att interrupt latency ökar ju längre tid avbrotten är avstängda.

ISR-latency
Benämning i Windows CE. Med ISR-latency, **I**nterrupt **S**ervice **R**outine latency avses fördröjningen från hårdvaruavbrott tills kod i en avbrottsrutin börjar exekvera (se Figur 5).

IST-latency
Benämning i Windows CE. Med IST-latency, **I**nterrupt **S**ervice **T**hread latency avses fördröjningen från hårdvaruavbrott tills en tråd startar att exekvera. Tråden startas från en avbrottsrutin (ISR). Denna fördröjning är alltid större än ISR-latency (se Figur 5).

3.2 Placering av reglerkod

I Windows CE finns två alternativ för placering av reglerkoden:

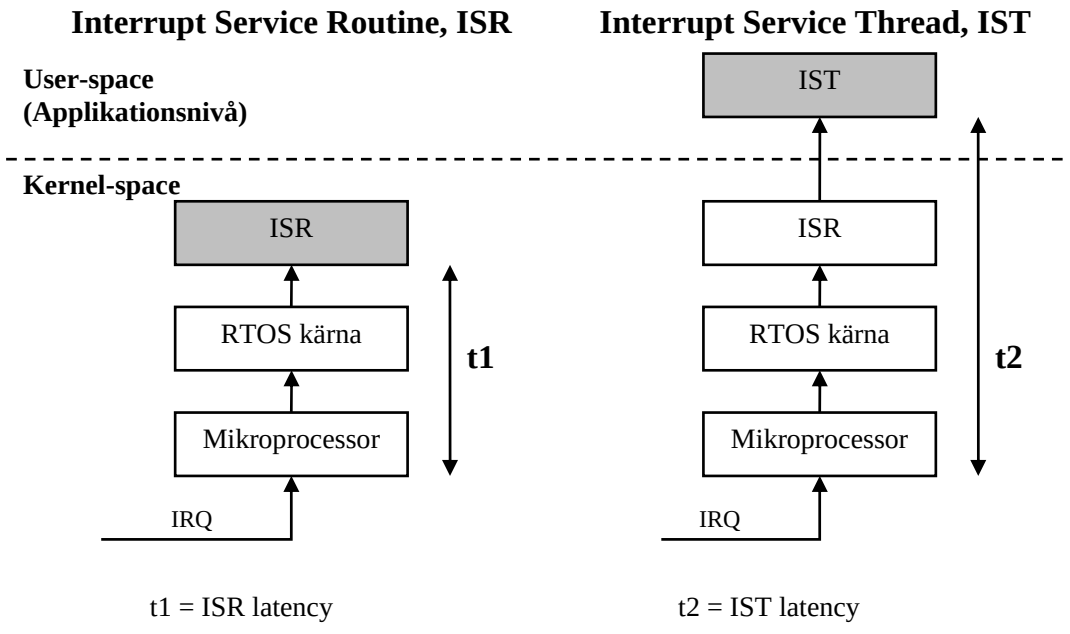
- Interrupt Service Routine (ISR) eller avbrottsrutin på svenska.
- Interrupt Service Thread (IST) eller avbrottstråd på svenska.

I dokumentationen för Windows CE rekommenderas att bara ha ett fåtal programrader i en Interrupt Service Routine, ISR och därifrån starta en Interrupt Service Thread, IST i user-space där huvuddelen av koden skall ligga. Detta förfarande är också det rekommenderade för andra RTOS. Anledningen är att avbrotten oftast är avslagna då en ISR exekverar och risken finns att ett nytt inkommande avbrott missas då avbrottsrutinen exekverar. Genom att minimera programkoden i avbrottsrutinen minimeras denna risk.

En IST exekverar i user-space som en normal tråd med tilldelad prioritet och påslaget minnesskydd via Memory Management Unit, MMU. Minnesskyddet innebär att en feladresserad pekare inte kan orsaka programvarufel utanför det egna minnesområdet.

I kernel-space är inte minnesskyddet aktiverat vilket innebär att felaktig kod kan skriva i hela systemets minnesarea. Detta är en säkerhetsrisk och kan resultera i svårupptäckta fel.

Att placera reglerkoden i user-space vore alltså fördelaktigt. Problemet är att tidsfördröjningen, Interrupt Service Thread latency är hög i user-space. Om samplingshastigheten i regleringen är för hög kan reglerkoden inte placeras i user-space pga att regleralgoritmerna inte hinner exekvera klart innan nästa regler-avbrott inkommer.



Figur 5: Placering av reglerkod

Reglerkod i Interrupt Service Routine, ISR

+ Låg tidsfördröjning från avbrott.
- Standardfunktioner i C kan inte anropas, t.ex. sin(x), printf().
- Funktioner på applikationsnivå (User-space) kan inte anropas.
  Data måste överföras via minnesbuffert eller DMA.
- Svårt att debugga.
- Inget minnesskydd vilket innebär säkerhetsrisk.
- Då reglerkoden exekverar måste övriga interrupt vara avstängda. Detta kan innebära att avbrott missas från andra enheter som ethernet, serieportar m.m.

Reglerkod i Interrupt Service Thread, IST

+ Minnesskydd aktiverat.
+ Lätt att programmera då standardfunktioner kan anropas.
+ Lätt att debugga
- Lång tidsfördröjning från avbrott.

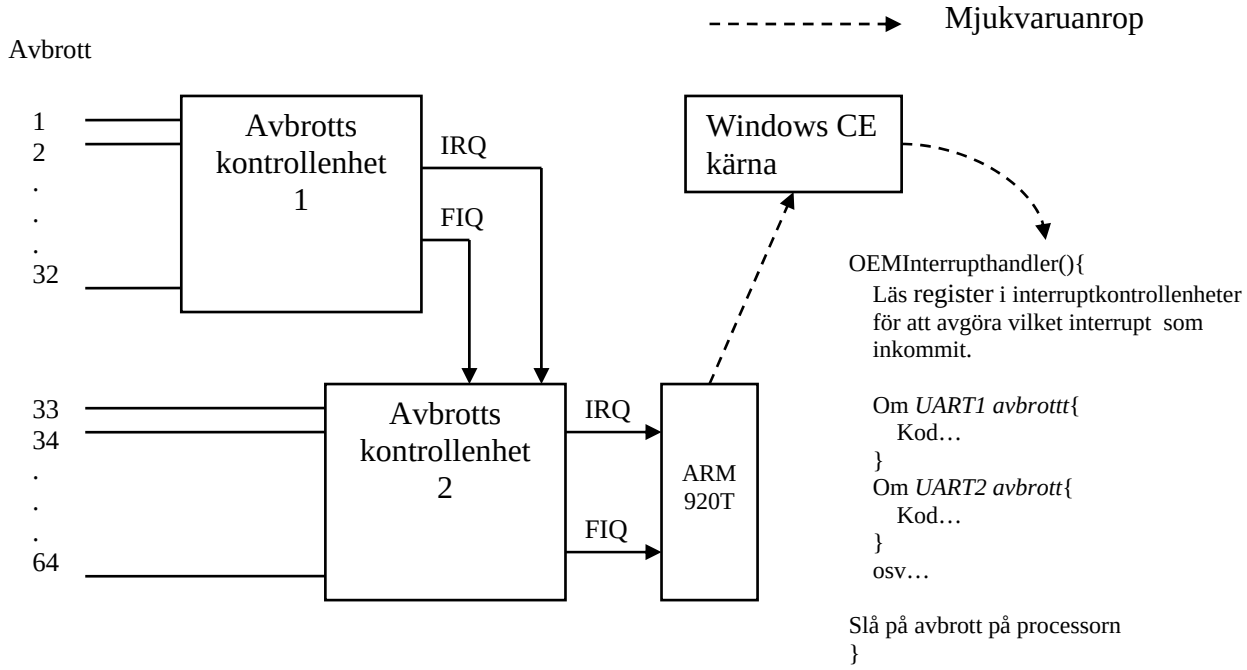
3.3 Avbrottshantering med Windows CE och ARM920T-processor

Kärnan i ARM-processorer har endast två ingångar för hårdvaruavbrott:

- IRQ, Interrupt Request. Detta är det avbrott som normalt används.
- FIQ, Fast Interrupt Request. Ett FIQ hanteras snabbare av processorn än ett IRQ. Prioriteten för ett FIQ är högre än för ett IRQ vilket innebär att ett FIQ avbryter ett IRQ. Med Windows CE kan ett FIQ inte starta en Interrupt Service Thread, IST utan man är hänvisad till at ha kod i en Interrupt Service Routine. Ett FIQ kan inte heller vara ett nästlat avbrott.

För att åstadkomma exklusiv åtkomst av data och register stänger realtidsoperativsystemet och drivrutiner av IRQ ibland. Detta är anledningen till att man normalt sett skall använda IRQ. Används FIQ måste man vara mycket noggrann med vilka data och register som accessas. FIQ används normalt sett bara vid allvarliga fel och för drivrutiner med mycket höga realtidskrav.

Avbrottshanteringen i systemet är uppbyggt enligt Figur 6. 64 olika avbrott är anslutna till två seriekopplade kontrollenheter. Då ett avbrott inkommer exekveras kod i Windows CE kärnan och en avbrottsrutin anropas. All programkod som exekverar i någon tråd avbryts oavsett trådprioritet. I avbrottsrutinen kontrolleras vilket av de 64 avbrott som genererats genom att läsa register i de två kontrollenheterna.



Figur 6: Avbrottshantering

3.3.1 Nästlade avbrott

Avbrottsrutinen (ISR) kan implementeras som icke nästlad eller nästlad. Då ett avbrott inkommer stänger ARM-processorn automatiskt av avbrottshanteringen så att nya inkommande avbrott ignoreras.

Icke nästlad innebär att alla avbrott är avstängda då kod i avbrottsrutinen exekveras. Då avbrottsrutinen exekverat klart slås avbrottshanteringen på i mjukvaran. Fördelen är att endast ett avbrott i taget kan exekvera och det finns då ingen risk att stacken med lagrade avbrottsrutiner blir för stor.

Om det finns ett viktigt avbrott i systemet kan det vara en fördel om detta avbrott får avbryta ett annat mindre viktigt avbrott som redan exekverar. Avbrottsrutinen för det mindre viktiga avbrottet sparas då tillfälligt undan av kärnan medan det högprioriterade avbrottet exekverar. Detta förfarande benämns ***nästlade avbrott*** och stöds av kärnan i Windows CE från version 3.0. Denna metod används i projektet då reglerkoden för motorstyrningen har högsta prioritet och skall tillåtas att avbryta övriga avbrott
enligt Figur 7.

```
OEMInterrupthandler(){
   Läs register i interruptkontrollenheter
   för att avgöra vilket avbrott som inkommit.

   Om Motorstyrnings-avbrott{
      Reglerkod…
   }
   Om Mindre viktigt avbrottt{
      Stäng av alla avbrott utom motorstyrnings-
      interruptet genom att skriva till avbrotts-
      kontrollenheternas maskregister.

      Slå på processorns avbrotts bit.

      Kod…
   }
   osv…

Slå på avbrott på processorn.
}
```

Figur 7: Pseudokod nästlade avbrott

3.4 Hård och mjuk realtid.

Med hård realtid avses ett system där alla deadlines garanterat hålls. Ett exempel på ett hårt realtidssystem skulle kunna vara ett reglersystem på ett flygplan där varje deadline måste hållas för att säkerställa funktionen. Med mjuk realtid avses ett system där de flesta deadlines hålls.

Det finns två aspekter på hård och mjuk realtid:

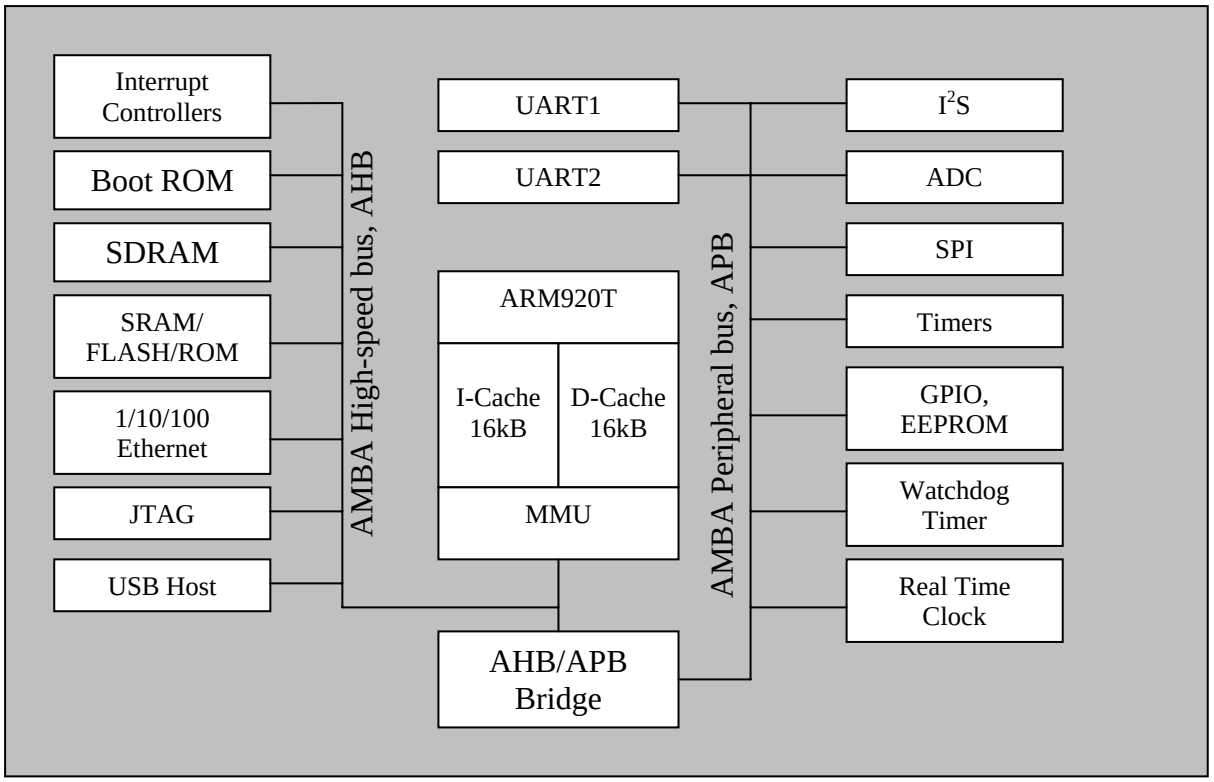
- Kräver reglerprocessen hård eller mjuk realtid ?
- Vilken typ av realtid erbjuder det RTOS som skall användas ?

Kräver regleringen av elmotorn i vår applikation hård eller mjuk realtid ?
Detta behandlas i ett parallellt pågående examensarbete [1] och slutsatsen är att hård realtid inte krävs. Det går att missa enstaka deadlines. Att ge en exakt siffra på hur mycket som kan missas är svårt. Motorregleringen havererar dock inte om enstaka deadlines missas.

En intressant artikel i Elektroniktidningen behandlar valet mellan hård och mjuk realtid [2]. Professor Hans Hansson vid Mälardalen Real-Time Research Centre säger bl.a :
”*Reglersystem är tåligare än datavetare tror, de klarar oftast missade samplingar*”.

4 Hårdvara

Utvecklingskortet EDB9301 från Cirrus Logic används i projektet. Kortet är baserat på en 166 MHz ARM920T processor. Processorn är en 32 bitars RISC med MMU vilket krävs för att köra Windows CE och Linux. 16 Mb flash och 32 Mb SDRAM finns på kortet.



Figur 8: Blockschema utvecklingskort EDB9301

4.1 Boot ROM

Utvecklingskortet är försett med ett s.k. Boot Rom. Kortet kan byglas så att processorn börjar exekvera kod från en UART via RS232 serieporten vid uppstart. På detta sätt laddas mjukvara (bootloader) ned från utvecklingsmaskinens serieport till målsystemets flashminne. Vid reset av processorn startar bootloadern upp målsystemet och initierar systemet så att programvara därefter kan laddas ned från utvecklingsmaskinen via ethernet.

5 Windows CE.net 4.2

Windows CE är ett realtidsoperativsystem för inbyggda system från Microsoft. Arkitekturen i operativsystemet har ingenting gemensamt med Microsofts övriga operativsystem för PC. Förkortningen .net anspelar på att applikationsprogram kan programmeras i Microsoft .net miljö. Denna förkortning har dock försvunnit i nyare versioner även om .net funktionaliteten finns kvar. Senaste versionen är Windows CE 5.0 som släpptes sommaren 2005. Att version 4.2 används i utvärderingen beror på att det Board Support Package som medföljde utvecklingskortet var skrivet för version 4.2.

De processorer som stöds är:

- ARM
- MIPS
- PowerPC
- Hitachi SH
- x86

Windows CE levereras med en mängd applikationer och kommunikationsprotokoll. Priset är relativt lågt. Två olika varianter av operativsystemet finns, core och professional. Professional har kraftfullare stöd för multimedia och grafik.

Utvecklingsmiljö	$995
Core licens	$3
Professional licens	$16

Tabell 1: Licenskostnader

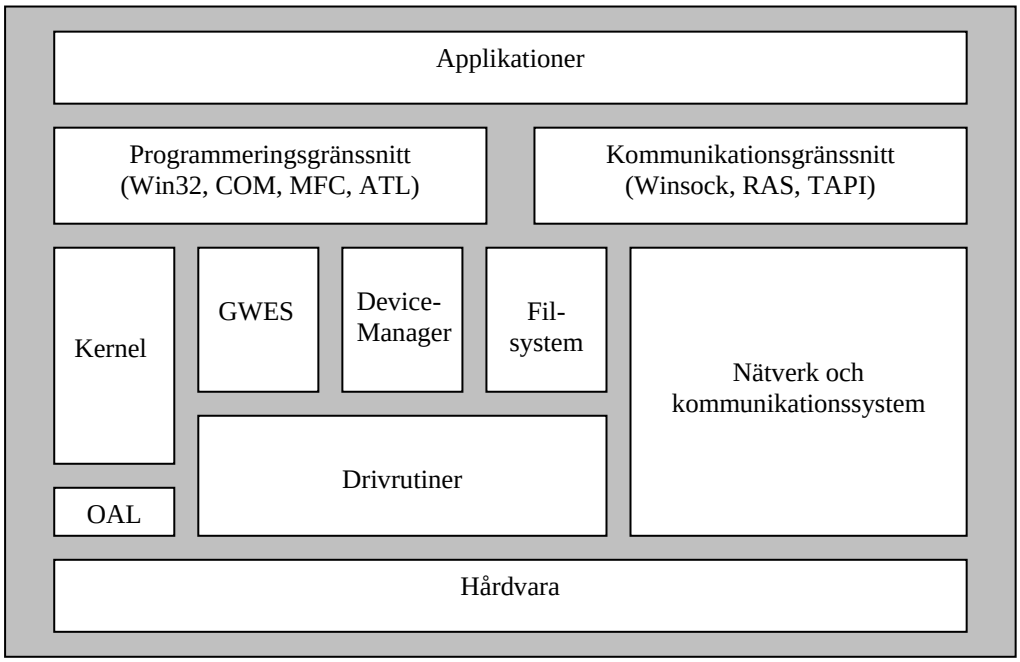
Operativsystemet levereras med två miljoner rader källkod för att underlätta felsökning. Det är dock inte all källkod som medföljer. I version 5.0 får man även modifiera källkoden efter överenskommelse med Microsoft.

Som operativsystemets styrka framhävs ofta dess kraftfulla stöd för grafik och kommunikationsmöjligheter. Operativsystemet används ofta i inbyggda system med grafisk display t.ex. pocket-PC och operatörsterminaler för industriautomation.

Operativsystemet har ett footprint från ca: 300 Kb. Med web ftp och telnetserver krävdes ca: 1Mb flashminne. Ett litet footprint är viktigt då fysiskt lagringsmedia (flashminne) fördyrar produkten.

5.1 Systemarkitektur

I Figur 9 visas operativsystemets uppbyggnad. Object Adaption Layer, OAL är implementerat av Cirrus Logic som tillhandahåller ett Board Support Package, BSP för vårt utvecklingskort. Ett Board Support Package innehåller funktioner som gör det enkelt att använda utvecklingskortets hårdvara. Drivrutiner i Windows CE implementeras som DLL:er, dvs dynamiskt laddbar programkod. Det går även att accessa hårdvara direkt från en applikation utan att skriva en drivrutin. Denna metod har använts i projektet.



Figur 9: Arkitektur

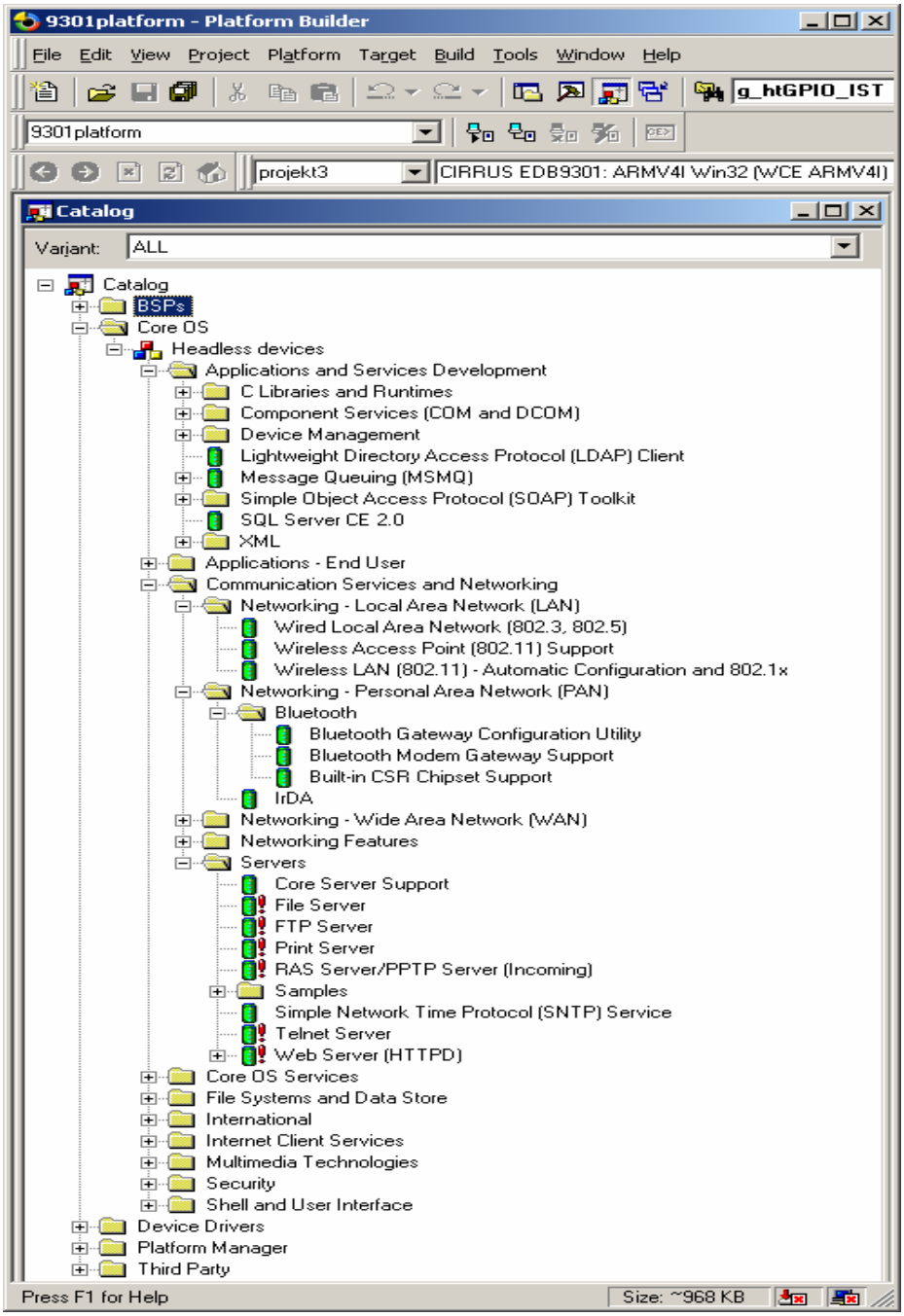
5.2 Utvecklingsmiljö

Utvecklingen är tänkt att delas upp i två delar:

- Plattformsutveckling med Platform Builder
- Applikationsutveckling med Embedded Visual C/C++, C# eller VB.net

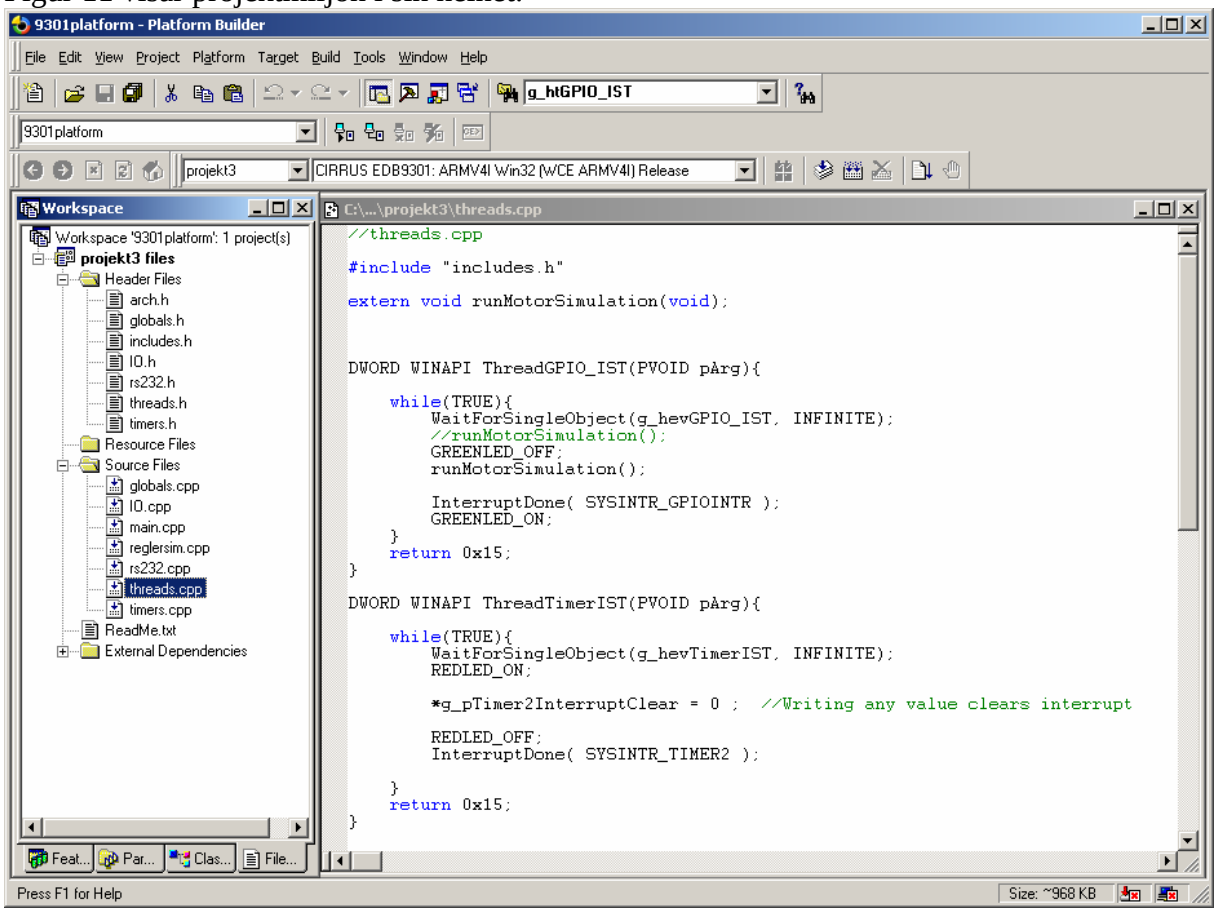
5.2.1 Plattformsutveckling

Platform Builder används för konfigurering och programmering av operativsystemet i C/C++. Det finns också ett flertal verktyg för felsökning och systeminformation. I projektet har Platform Builder använts för all programmering. De färdiga protokoll och applikationer man vill använda i plattformen hämtas enkelt med ”drag and drop”. I Figur 10 visas exempel på tillgängliga protokoll och applikationer. Figur 11 och Figur 12 visar projektmiljön.



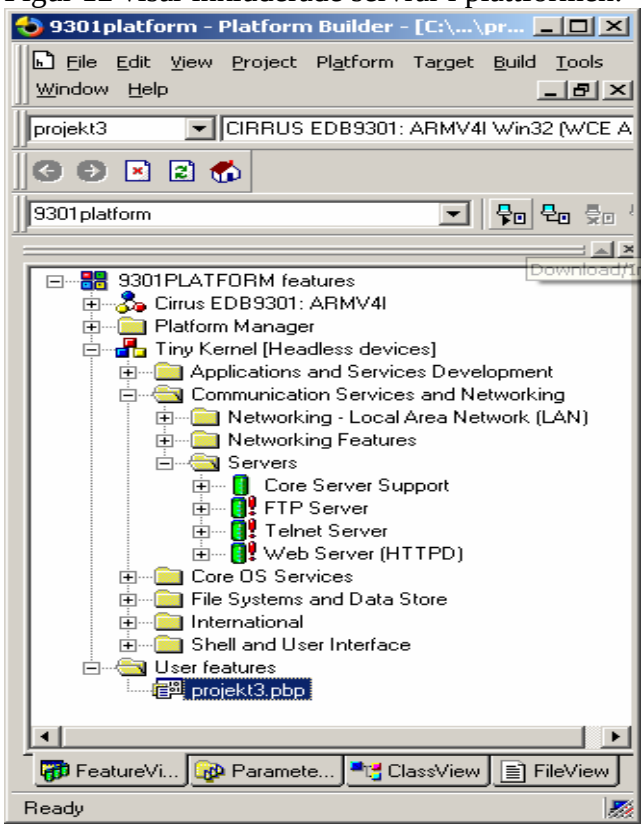
Figur 10: Tillgängliga protokoll och applikationer

Figur 11 visar projektmiljön i sin helhet:



Figur 11: Projektmiljö

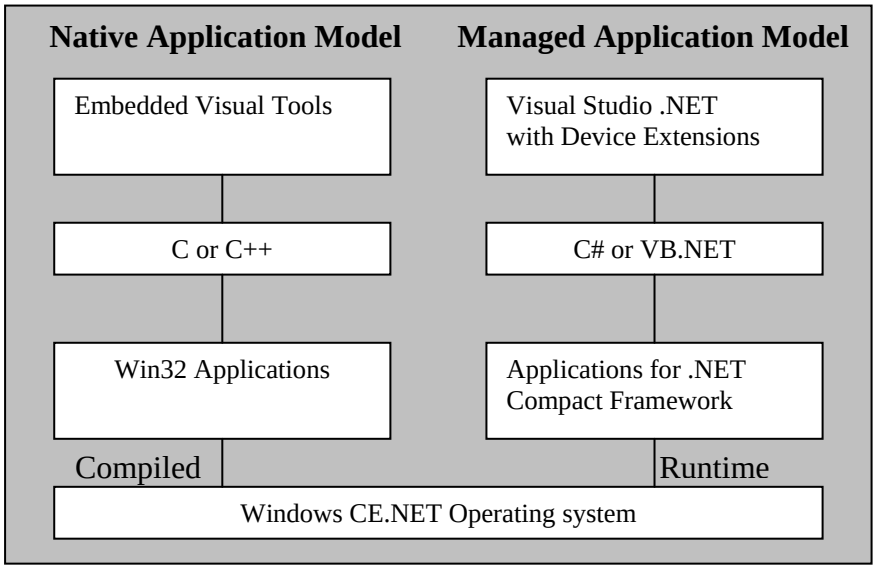
Figur 12 visar inkluderade servrar i plattformen:



Figur 12: Konfiguration av plattform

5.2.2 Applikationsprogrammering

Applikationsprogrammering kan utföras i C/C++, C# eller VB.net enligt Figur 13:

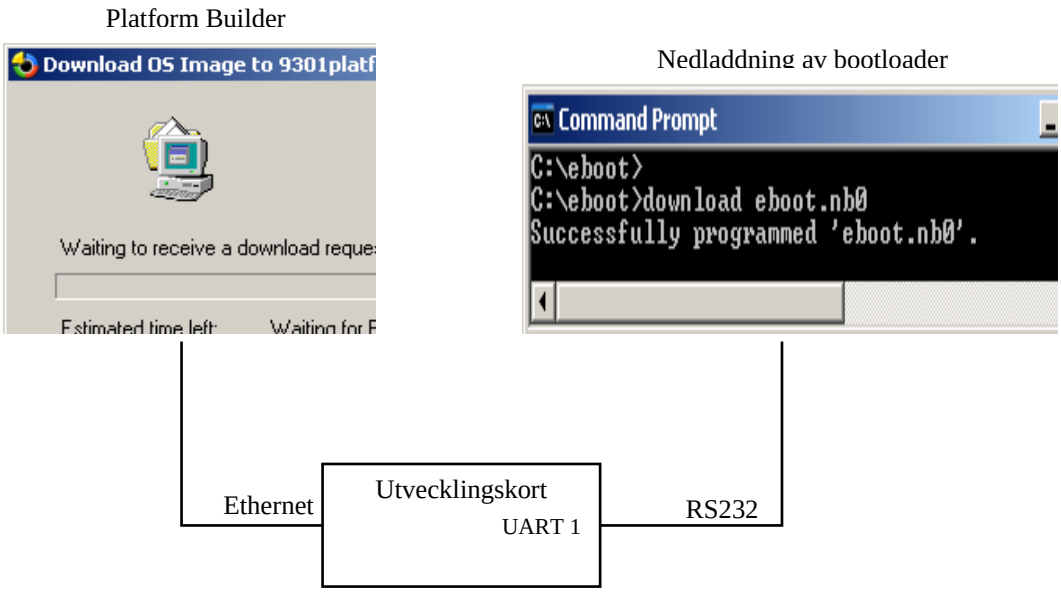


Figur 13: Programmeringsmodell

Embedded Visual C++ medföljer utvecklingsmiljön och genererar kompilerad C/C++ kod. Stöd finns för MFC, ATL och Win32 API.

5.2.3 Nedladdning av mjukvara

En bootloader för Windows CE överförs via serieport till målsystemets flashminne enligt Figur 14. Nedladdningen är känslig för överföringsfel och helst skall en stationär dator med inbyggd seriell port användas. Att använda en bärbar dator med serieport på PCMCIA-kort fungerade inte. En bygel på utvecklingskortet sätts i läge för seriell inläsning av bootkod via en UART-krets. Därefter laddas mjukvara ned från Platform Builder via ethernet. Det finns också en mängd verktyg för felsökning och systeminformation av målsystemet som använder Ethernet.



Figur 14: Nedladdning av mjukvara

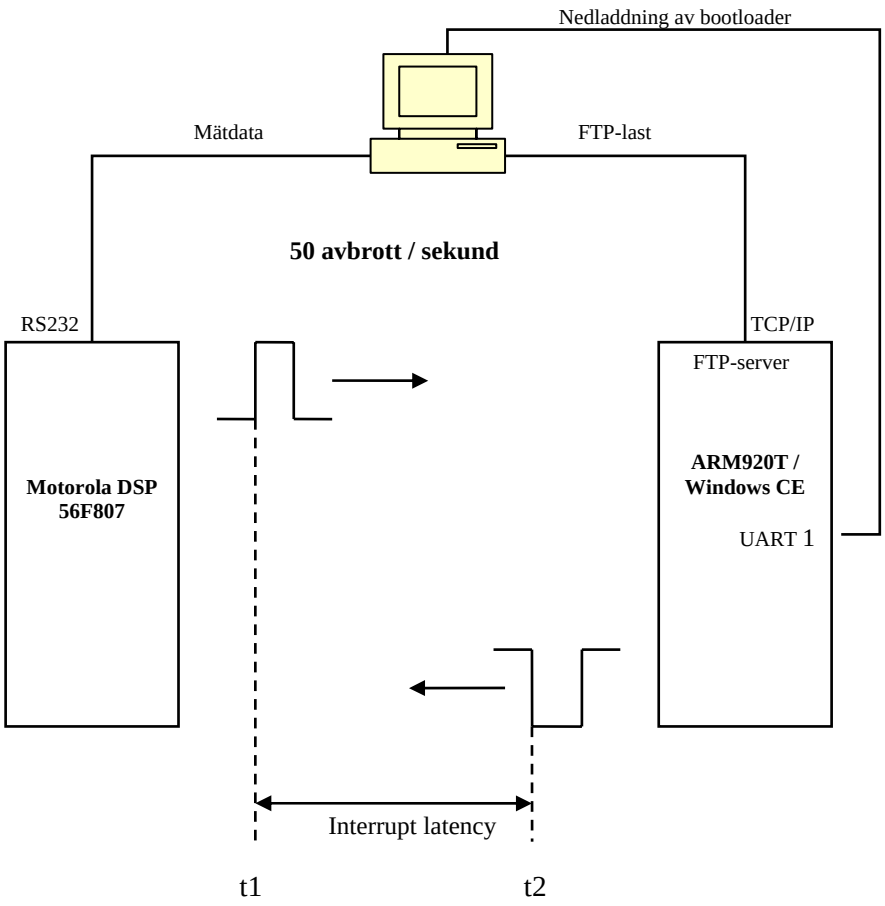
6 Mätning av Windows CE realtidsprestanda.

Interrupt latency mäts genom att skicka in en puls på en digital ingång och generera ett IO-avbrott enligt Figur 15. Då avbrottsrutinen börjar exekvera skickas ett svar ut på en digital utgång. Tiden från generering av pulsen tills pulssvaret inkommer mäts på ett utvecklingskort med en Motorola 56F807 DSP. DSP:n har hårdvarutimers som tidsstämplar det inkommande pulssvaret automatiskt utan hantering av processorn vilket ger bra noggrannhet. Resultatet skickas via RS232 till en PC och lagras i en textfil. Textfilen behandlas i Matlab där min, max och medelvärde på tidsfördröjningen beräknas.

På grund av begränsningar i överföringshastighet av mätdata skickas endast 50 pulser per sekund. För att belasta systemet används kontinuerlig FTP-överföring via ethernet. Mätningar utförs också med simulerad motorstyrningskod i systemet. Slutligen utförs ett test med 10 kHz avbrottsström genererade från en frekvensgenerator.

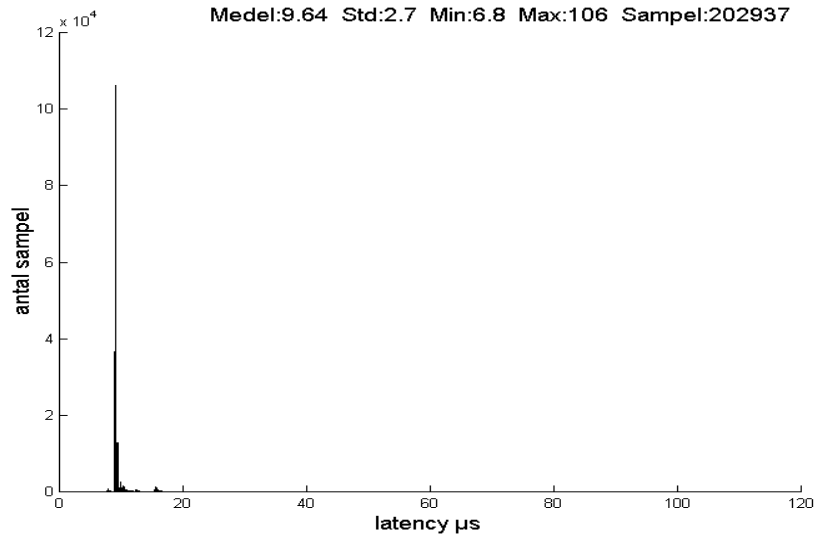
6.1 Mätning med 50 Hz avbrottsström

6.1.1 Systemuppbyggnad 50 Hz avbrottsström

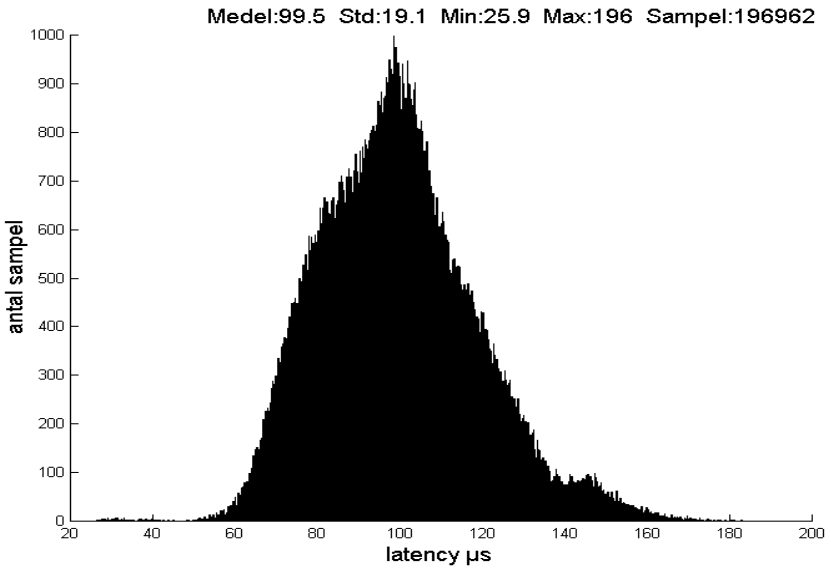


Figur 15: Mätsystem 50 Hz avbrottsström

6.1.2 Mätresultat IST latency, 50Hz avbrottsström



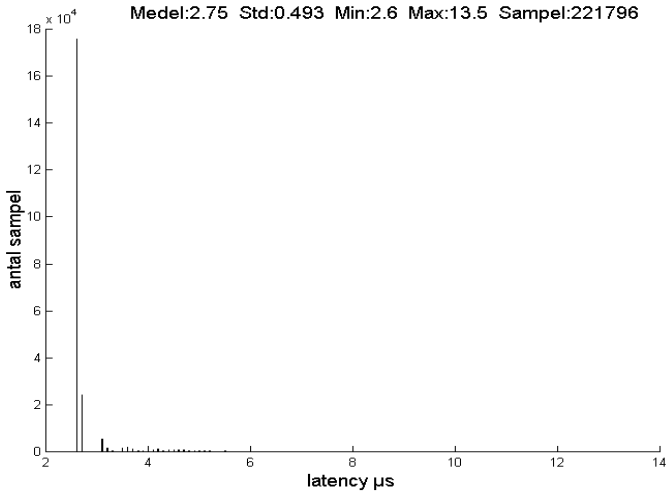
Figur 16: IST latency utan last. 50 Hz



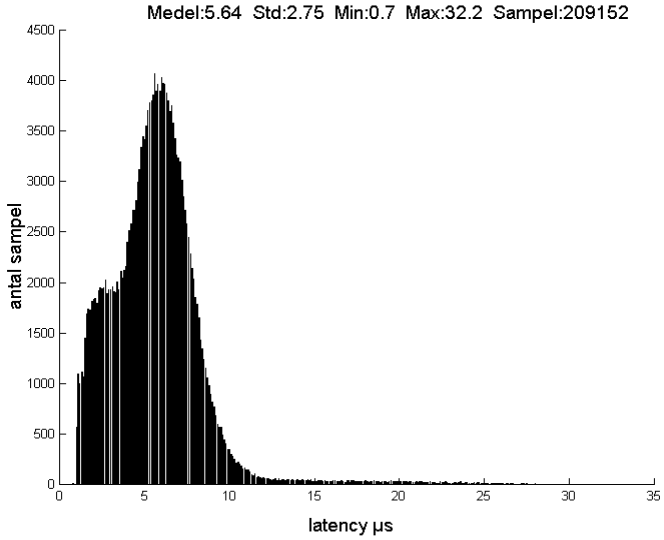
Figur 17: IST latency med last. 50 Hz

Det intressantaste mätvärdet är max latency vid belastat system på 196 µs. Målsättningen är att regleralgoritmerna ska exekvera var 100:e µs, dvs ett avbrott inkommer var 100:e µs. Då avbrottet inkommer skall det behandlas i en regleralgoritm innan nästa avbrott inkommer. Det går alltså inte att placera regleralgoritmerna i en IST pga för lång Interrupt latency.

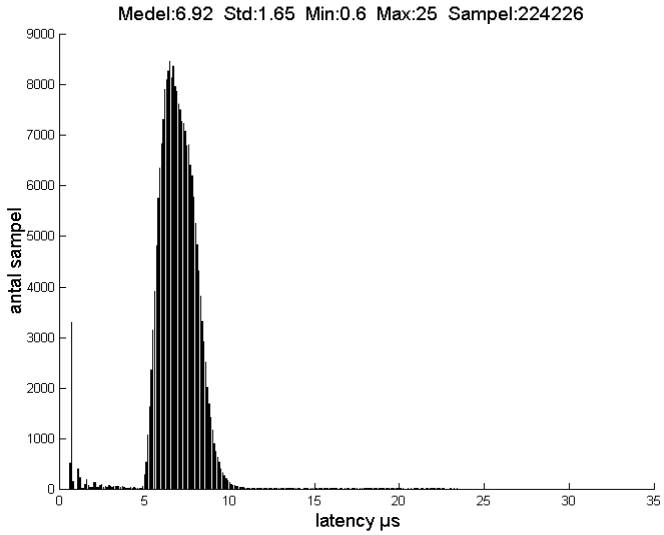
6.1.3 Mätresultat ISR latency, 50Hz avbrottsström



Figur 18: ISR latency utan last. 50 Hz



Figur 19: ISR latency med last. 50 Hz

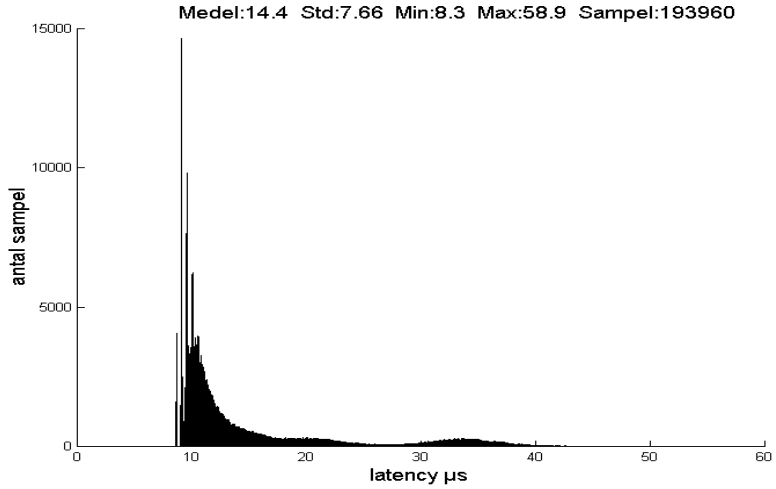


Figur 20: FIQ latency med last. 50 Hz

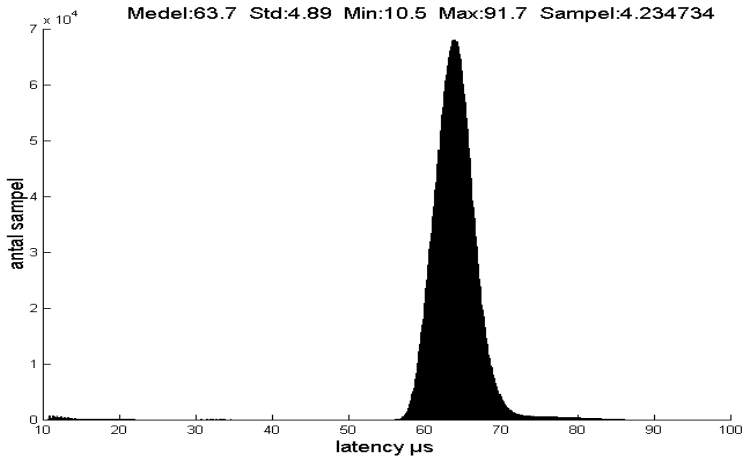
Det intressantaste mätvärdet är *max latency* vid belastat system på 32,2 µs i Figur 19. Beroende på hur lång tid regleralgoritmerna tar för exekvering så skulle det kunna fungera att placera reglerkoden i en ISR. Ett test utförs också med ARM processorns snabbare FIQ avbrott. Figur 20 visar resultatet som är marginellt bättre jämfört med standard IRQ.

6.2 Mätning med algoritmer för motorstyrning

Programkod för motorstyrning läggs nu in i en ISR. Simuleringen är mycket enkelt uppbyggd och syftar till att ge en ungefärlig bild av systemet. Ingen hårdvara accessas i simuleringen. Nya ärvärden till regleralgoritmerna skapas i mjukvara varje gång avbrottsrutinen exekverar. Tid från inkommande avbrott tills det motorstyrkoden har exekverat klart redovisas i Figur 21 och Figur 22 .



Figur 21: Interrupt latency + exekveringstid reglerkod 50 Hz utan last.



Figur 22: Interrupt latency + exekveringstid reglerkod 50 Hz med last.

Resultatet i Figur 22 är lite förvånande. Vi har tidigare uppmätt ett medelvärde på ISR-latency till 2,7 µs olastat (Figur 18) och i Figur 21 fås ett olastat medelvärde på 14,4 µs med reglerkoden inkluderad. Reglerkodens exekveringstid borde alltså vara 14,4 - 2,7 = 11,7 µs.

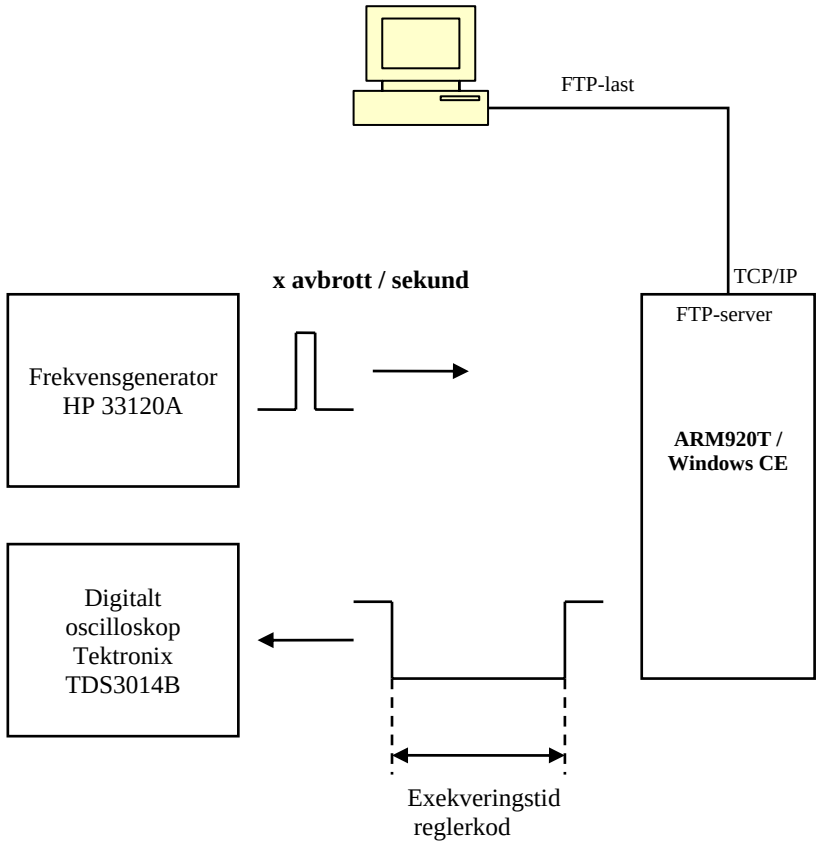
Om vi adderar denna exekveringstid och uppmätt ISR-latency utan reglerkod i Figur 19 fås 11,7 + 5,6 = 17,3 µs i medel. Medelvärdet i Figur 22 borde alltså vara 17,3 µs, istället fås

medelvärdet 63,7 µs. Orsaken är att exekveringstiden för reglerkoden varierar starkt beroende på cacheinnehåll. Detta utreds närmare i kapitel 6.3.

Ett intressant mätvärde är maxvärdet *interrupt latency* + *exekveringstid* = 91,7 µs i Figur 22. Motorstyrningen kommer alltid att hinna exekvera klart inom 100 µs. Viktigt att tänka på är att dessa data gäller med avbrottsfrekvensen 50 Hz. Målsättningen är att köra systemet med 10 kHz vilket testas i kapitel 6.4.

6.3 Mätning av exekveringstid för reglerkod

I föregående kapitel konstaterades att exekveringstiden för reglerkoden varierade starkt. Exekveringstiden skall nu mätas vid olika frekvenser på avbrottsströmmen. En frekvensgenerator genererar avbrott med olika frekvenser enligt Figur 23. Precis innan reglerkoden börjar exekvera dras en utport låg och direkt efter att exekveringen är klar dras en utport hög. Pulstiden motsvarar då exekveringstiden och mäts av ett digitalt oscilloskop. Oscilloskopet har inbyggda statistikfunktioner som bildar medelvärde och standardavvikelse baserat på 1000 st mätvärden.



Figur 23: Mätning av exekveringstid för regleralgoritmer.

6.3.1 Mätresultat

Frekvens Hz	Ex.tid medel µs	Ex.tid stdavv. µs	last
50	61	4,0	ftp
500	56	7,2	ftp
1000	49	4,6	ftp
5000	23	4,4	ftp
10000	12,8	7,5	ftp
15000	9,9	6,4	ftp

Tabell 2: Exekveringstid reglerkod med last

Frekvens Hz	Ex.tid medel µs	stdavv. µs	last
50	10,8	5,5	olastat
500	8,9	2,2	olastat
1000	8,5	1,0	olastat
5000	8,5	0,6	olastat
10000	8,0	1,87	olastat
15000	7,8	2,2	olastat

Tabell 3: Exekveringstid reglerkod utan last

Ur Tabell 2 och Tabell 3 kan två slutsatser dras:

- Exekveringstiden sjunker med ökande frekvens.
- Exekveringstiden sjunker vid minskande last.

Vid ökande frekvens kommer reglerkoden att exekvera oftare och innehållet i cachar kommer mer sällan att skrivas över av annan exekverande programvara. Motorstyrkoden kommer nästan alltid att ligga i cache, därför sjunker exekveringstiden. Vid minskande last uppnås samma effekt, annan programvara exekverar mer sällan och skriver inte i cache lika mycket.

För att verifiera denna teori töms innehållet i cache innan exekvering av reglerkod med flush-funktioner enligt följande kod:

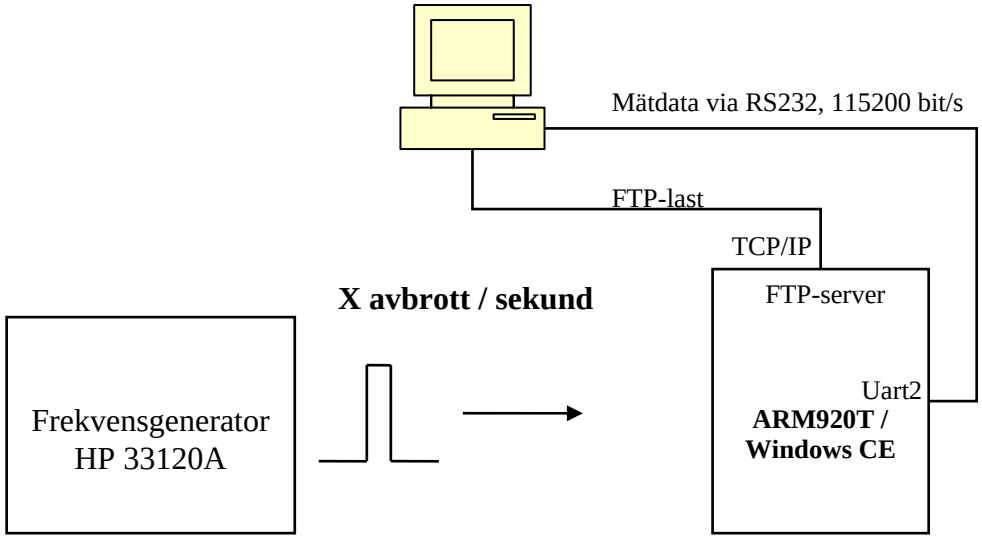
```
int OEMInterruptHandler() {
…
        FlushICache();  //Flush av instruktionscache
        FlushDCache(); // Flush av datacache
        TLBClear();  // Flush av Transfer Lookaside Buffer

        GREENLED_OFF; // Låg signal ut
        runMotorSimulation(); // Reglerkod
        GREENLED_ON; // Hög signal ut
…
```

Resultatet blir att exekveringstiden ligger konstant på **60,9 µs** oavsett last och frekvens. Detta resultat stämmer mycket bra med mätvärdet 61 µs vid 50 Hz i Tabell 2: Exekveringstid reglerkod med last
Med 50 Hz och ftp-last kommer motorstyrkoden nästan aldrig att ligga i cache, vi uppnår samma effekt som med flushad cache.

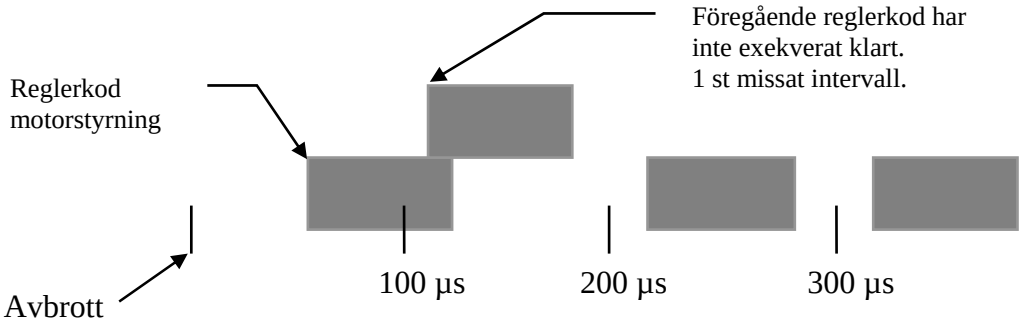
6.4 Mätning med 10 kHz avbrottsström

Tidigare mätningar har utförts med 50 avbrott per sekund pga begränsningar i mätutrustningen. Målsättningen är att motorstyrningen skall exekvera med en avbrottsström på 10 kHz. Hur fungerar systemet vid denna frekvens ? Tidigare mätuppkoppling kan inte användas p.g.a. den höga frekvensen. En frekvensgenerator genererar avbrott till systemet med hög frekvens enligt Figur 24.



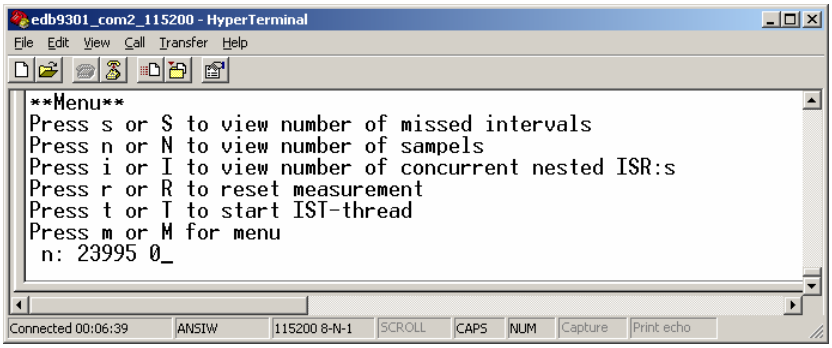
Figur 24: Mätning med 10 kHz avbrottsström

Interrupt latency kan inte mätas för varje enskilt avbrott utan istället räknas antal missade intervall enligt Figur 25.



Figur 25: Missad deadline

Innan en avbrottsrutin med reglerkod börjar exekvera kontrolleras om föregående reglerkod hunnit exekvera klart. Om så inte är fallet har ett intervall missats och en räknare i mjukvaran ökas. Antal missade intervall avläses via RS232 med ett terminalprogram enligt Figur 26.



Figur 26: Mätdata via RS232. 115200 bit/s

I tidigare mätningar har alla avbrott varit avstängda då reglerkoden har exekverat. I denna mätning måste ett nytt motorstyrningsavbrott tillåtas då reglerkoden exekverar. Ibland kommer flera instanser av reglerkoden att stackas. Pseudo-koden för avbrottshanteringen framgår i Figur 27.

```
OEMInterrupthandler(){
   Läs register i avbrottskontrollenheter
   för att avgöra vilket avbrott  som inkommit.

   Om Motorstyrnings-avbrott{
      Stäng av alla avbrott utom motorstyrnings-
      avbrottet genom att skriva till avbrotts-
      kontrollenheternas maskregister.

      Slå på processorns avbrottsbit.

      Reglerkod…
   }
   Om Mindre viktigt avbrott{
      Stäng av alla avbrott utom motorstyrnings-
      avbrottet genom att skriva till avbrotts-
      kontrollenheternas maskregister.

      Slå på processorns avbrottsbit.

      Kod…
   }
```

Figur 27: Pseudo-kod: Mätning av missade intervall

6.4.1 Mätresultat med ftp last

Frekvens Hz	% missade intervall	n missade intervall	Antal sampel
10k	0,000009	33	382.789.989
15k	0,000182	69	37.971.129
20k	0,000571	243	42.543.586
25k	0,032147	2454	7.633.677
30k	0,24	25296	10.377.102
40k	0,99	211369	21.454.605
50k	Systemlåsning		

Tabell 4: Missade intervall med ftp last

Resultatet är mycket bra. Upp till 20 kHz är antalet missade intervall försumbart litet för applikationen. Reglerkoden kommer nästan alltid att få exekvera vid önskad tidpunkt trots att systemet belastas med ständig filöverföring via ftp.

Då mätningarna utfördes noterades vid några tillfällen väsentligt högre antal missade intervall. För att avläsa antal missade intervall skickas bokstaven *s* från PC:n via RS232 till serieporten på utvecklingskortet. Då tecknet inkommer triggas ett serieportsavbrott (UART) och antalet missade intervall sänds tillbaka till PC:n via RS232. Då många avläsningar utfördes i en följd steg antalet missade intervall drastiskt.

Systemet består nu av två avbrottsrutiner med relativt mycket programkod, reglerkods-avbrottet och serieports-avbrottet enligt pseudokod i Figur 28. Kärnan i Windows CE avbryter ett exekverande serieportsavbrott om det samtidigt kommer in ett avbrott för motorstyrningen. Programkoden för serieportsavbrottet sparas då undan på avbrottsstacken och motorstyrkoden börjar exekvera. Därefter återupptas exekveringen av det avbrutna serieportsavbrottet. Detta skifte fungerar, men tar ibland för lång tid. Andelen missade intervall ökar drastiskt då avbrottet på serieporten är aktivt.

Den FIFO buffer på 16 tecken som kan användas både vid mottagning och vid sändning används inte. Varje inkommande tecken genererar ett avbrott och endast ett tecken kan skickas i taget.

```
OEMInterrupthandler(){

  Om Motorstyrnings-avbrott{
    Stäng av alla avbrott utom motorstyrnings-
    avbrottet genom att skriva till avbrotts-
    kontrollenheternas maskregister.
    Slå på processorns avbrottsbit.

    Reglerkod…
  }
  Om UART-avbrott{
    Stäng av alla avbrott utom motorstyrnings-
    avbrottet genom att skriva till avbrotts-
    kontrollenheternas maskregister.
    Slå på processorns avbrottsbit.

    Om s-tecken {
        Skicka antal missade intervall på serieport.
    }
    Om n-tecken {
        Skicka antal sampel på serieport.
    }
    osv…
```

Figur 28: Pseudo-kod: Interrupt på serieport (UART2)

6.4.2 Mätresultat med ftp och RS232 last.

I tidigare mätningar konstaterades att avbrottet på serieporten hade en negativ inverkan på antal missade intervall. I mätningarna adderas nu en RS232-last genom att en fil med ca 7000 s-tecken skapas. I terminalprogrammet på PC:n anges filen som källa och alla tecknen kommer då att skickas i en följd via RS232 till utvecklingskortet.

Frekvens Hz	% missade intervall	n missade intervall	Antal sampel
1k	0,04%	6	15586
5k	1,12%	962	85858
10 k	2,32%	3342	143932
20 k	3,19%	10445	327503

Tabell 5: Missade intervall med ftp och serielast 115 kbit/s.

Frekvens Hz	% missade intervall	n missade intervall	Antal sampel
1k	0,03%	7	26985
5k	10,15%	26278	258772
10 k	14,96%	78898	527261
15 k	20,35%	163215	802190

Tabell 6: Missade intervall med ftp och serielast 9,6 kbit/s.

I Tabell 5 ser vi att vid 10 kHz missas 2,32 % av alla intervall, vilket är alldeles för mycket. Andelen missade intervall ökar kraftigt om hastigheten sänks till 9600 bit/s. Förklaringen är att vid 9600 bit/s kommer mer tid att tillbringas i programkoden för serieportsavbrottet framförallt pga ökade väntetider vid skrivning till UART-kretsen.

16-bitars FIFO-bufferten aktiveras nu både vid läsning och skrivning till UART-kretsen. Framförallt skrivning av data till UART-kretsen går nu mycket fortare. Om exempelvis fem tecken skall sändas så skrivs tecknen direkt till FIFO:n i UART-kretsen som sedan sänder iväg tecknen utan att behöva vänta på sändningen i mjukvaran. Detta under förutsättning att FIFO-bufferten inte är full för då måste mjukvaran vänta på att det blir plats över i FIFO:n innan skrivning kan ske. Vid mottagning av ett tecken genereras ett avbrott antingen då mottagningsbufferten är halvfull eller då ett tecken mottagits och inga fler tecken inkommit under en kortare tidsperiod.

Frekvens Hz	% missade intervall	n missade intervall	Antal sampel
5k	0,01 %	4	61856
10k	0,00 %	5	131905
15k	1,42 %	2525	177882
20k	4,23 %	9151	216351

Tabell 7: Missade intervall med ftp och serielast 115 kbit/s. FIFO buffert aktiverad.

En klar förbättring noteras jämfört med värdena i Tabell 5. Vid 10 kHz missas nästan inga avbrott alls jämfört med 2,32 % i Tabell 5. Vid högre frekvenser återkommer problemet, detta då FIFO bufferten nu är full oftare och det uppstår väntetider i mjukvaran för serieports-avbrottet.

Enligt dokumentationen för Windows CE skall kod i avbrottsrutiner undvikas så mycket som möjligt. Rekommendationen är istället att starta en Interrupt Service Thread, IST från avbrottsrutinerna. Att ha koden för motorstyrningen i en IST har tidigare konstaterats olämpligt pga för lång interrupt latency.

Däremot skulle vi kunna lägga koden för serieportsavbrottet i en IST. Avbrottsrutinen skriver då tecken till en buffert utan att behöva vänta. En tråd i user-space läser bufferten och skriver sedan i FIFO-registret. Att detta inte utfördes i projektet beror på att det hade inneburit för mycket programmeringsarbete. Detta pga att avbrottsrutinen för serieporten behöver data från motorstyrnings avbrottsrutinen. Om motorstyrningens avbrottsrutin ligger i en ISR i kernel-space och serieports avbrottsrutinen i en IST i user-space måste dataöverföring mellan dem ske via separata buffrar.

Ett enkelt test utfördes istället där tecken skickades kontinuerligt till UART-kretsen från en IST utan att andelen missade intervall ökade. I detta fall klarar Windows CE av att skifta från IST-tråden till avbrottsrutinen för motorstyrningen utan problem. Så länge endast programkoden för motorstyrningen ligger i en ISR och all annan kod för avbrottsrutiner ligger i IST-trådar fungerar systemet utmärkt.

Även om det går att komma runt problemet visar försöket på en svaghet i systemet som man skall vara medveten om. Realtidskärnan tar för lång tid på sig att avbryta en lågprioriterad avbrottsrutin då ett högre prioriterat avbrott anländer. Fallet är också intressant att undersöka på andra RTOS.

6.5 Mätning av prestanda på övriga applikationer

Hittills har mätningar endast utförts på den realtidskritiska motorstyrningen. Det är också av intresse att studera hur mycket kapacitet som återstår för övriga applikationer i systemet. Dock finns det inte utrymme för noggrannare tester av detta i projektet. Vi nöjer oss med att kontrollera hur TCP/IP-flödet påverkas vid olika frekvenser på avbrottsströmmen för motorstyrningen.

Frekvens Hz	TCP/IP kbit/s
0	830
5k	658
10k	605
15k	560
20k	524

Tabell 8: TCP/IP-flöde vid olika frekvenser på motorstyrningen

Vid 10kHz har flödet sjunkit 27 % till 605 kbit/s jämfört med olastat tillstånd vilket får betraktas som bra. Övriga applikationer i systemet som Telnet och webservern fungerar också klanderfritt vid de ovan testade frekvenserna.

Först vid frekvenser upp mot 50 kHz hänger sig systemet oåterkalleligt och måste återstartas. Ett test utfördes där avbrottshanteringen konfigurerades om så att inga nästlade avbrott tilläts och således kunde inte systemresurserna överskridas. Resultatet blev att vid 50 kHz slutade ftp , telnet och webservern att fungera som tidigare men då frekvensen åter sänktes började systemet att fungera igen utan återstart.

7 Linux

Linux är benämningen på kärnan i operativsystemet GNU/Linux. Ofta används endast benämningen Linux för hela operativsystemet. Kärnan utvecklades av Linus Torvalds och första versionen släpptes 1991. Linux är ett UNIX-liknande operativsystem och uppfyller den s.k. POSIX-standarden för operativsystem till 100 %. Förutom kärnan består operativsystemet av en mängd fria programvaror från det fria mjukvaruprojektet GNU (**G**nu is **N**ot **U**nix), t.ex. utvecklingsverktyg, kompilatorer och debuggers.

Källkoden till Linux kan hämtas kostnadsfritt eller köpas paketerad i någon av de färdiga distributionerna på marknaden. Det är också tillåtet att modifiera källkoden för Linux men då måste modifieringarna publiceras då produkten distribueras.

7.1 GPL

GNU General Public Licens, oftast känd som bara GPL är den licens som används för linuxkärnan. Licensen har sitt ursprung i GNU-projektet och är den mest använda licensen för fri programvara.

För en GPL-licensierad programvara gäller följande:

- Det är tillåtet att modifiera källkoden och att använda hela eller delar av den i andra program.
- Det är tillåtet att sälja den nya programvaran vidare men då skall källkoden också distribueras om köparen av programvaran önskar detta. Det krävs dock att en köpare av programvaran hör av sig och vill ha källkoden. Man behöver inte ha källkoden offentligt publicerad utan det räcker att man skickar den till kunder som vill ha den. I praktiken levereras ofta en lapp med produkten där en ftp-adress anges och källkoden kan hämtas.
- Programvara som helt eller delvis innehåller eller härrör ifrån ett GPL-licensierat program skall licensieras under GPL i sin helhet. Den engelska ursprungstexten lyder: “*You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License”*.

Sista punkten innebär ett problem om man har en mjukvara bestående av både GPL-kod och icke-fri kod som man inte vill publicera källkoden till. Problemet är att betydelsen av orden *contains* och *derived* kan tolkas olika.

Linuxkärnan har dock fått ett viktiga tillägg till GPL-licensen av Linus Torvalds: Användarprogram som använder normala systemanrop skall inte anses som ”derived work” och behöver alltså inte licensieras under GPL.

Applikationsprogram med Linux som operativsystem behöver alltså inte publicera sin källkod. Problemet är att tillägget inte nämner något om drivrutiner, s.k. moduler i Linux. Moduler kan länkas till kärnan både statiskt och dynamiskt. Statisk länkning av en modul innebär helt klart att linuxkärnan har modifierats så att ett ”derived work” har åstadkommits och att programkoden i modulen faller under GPL.

Dynamisk länkning innebär att modulen laddas först då kärnan redan exekverar. Modulen kompileras separat, dock mot en viss version på kärnan och laddas av kärnan i binärform. Modulen kan anropa ett antal funktioner som exporteras av kärnan i ett särskilt gränssnitt. En del anser att de funktioner som exporteras av kärnan skall klassas som normala systemanrop enligt tillägget till Linuxkärnan och således inte falla under GPL. Många leverantörer som inte vill publicera källkoden till sina drivrutiner levererar idag moduler i binärform utan källkod.

Karim Yaghmour skriver i boken *Building Embedded Linux Systems* [14] att ämnet är fullständigt oklart och användandet av binärmoduler troligen kommer att fortsätta vara legalt tveksamt den närmaste framtiden. I boken finns ett appendix med utdrag från diskussioner i nyhetsgrupper där Linus Torvalds uttalar sig. Linus har dock inte copyright på all kod i Linuxkärnan och vissa menar att han inte kan uttala sig ensam om ämnet. Nedan följer några intressanta och i vissa fall motsägelsefulla utdrag från diskussionerna :

1999-02-05:
I allow binary-only modules. I allow them because I think that sometimes I cannot morally require people to make sources available to projects like AFS where those sources existed before Linux.

1999-02-07:
I allow binary-only modules, but I want people to know that they are only ever expected to work on the one version of the kernel that they were compiled for.

2002-10-17:
The only thing that allows for non-GPL modules is copyright law, and in particular the "derived work" issue. A vendor who distributes non-GPL modules is not protected by the module interface per se, and should feel very confident that they can show in a court of law that the code is not derived.

…The module interface has NEVER been documented or meant to be a GPL barrier…

The original binary-only modules were for things that were pre-existing works of code, ie drivers and filesystems ported from other operating systems, which thus could clearly be argued to not be derived works, and the original limited export table also acted somewhat as a barrier to show a level of distance.

2001-10-19:
What is left in the gray area tends to be clearly separate modules: code that had a life outside Linux from the beginning, and that do something self-contained that doesn't really have any impact on the rest of the kernel. A device driver that was originally written for something else, and that doesn't need any but the standard UNIX read/write kind of interfaces, for example.

Uppenbarligen har ett antal drivrutiner i binärform från andra UNIX-varianter tidigare accepterats. En anledning till motståndet mot binärmoduler är att de bara är garanterade att fungera mot exakt den Linuxkärna som de kompilerats för. Uppgraderas kärnan kan drivrutinen sluta att fungera och utan källkod kan inte användaren kompilera om drivrutinen för rätt version på kärnan. Utredningar av anmälda programvarufel i linuxkärnan försvåras också om inte all källkod finns att tillgå för de ansvariga i Linux-projektet.

Tilläggas bör att ingen någonsin har fällts för GPL-brott vid dynamisk länkning av en modul.

Då läget är juridiskt oklart kan man tänka sig några olika lösningar:

- Publicera källkoden till modulen enligt GPL och bli kvitt alla eventuella problem. Denna metod används bl.a. av Axis Communications i Lund[2].

- Att leverera binärkod till moduler för dynamisk laddning utan källkod är ett tillvägagångssätt som används av ett flertal företag idag. Det anses av många vara en ”de facto standard”. Intel och Nvidia[3] använder denna metod för många Linux-drivrutiner där man vill hålla källkoden hemlig.

- Montavista, den största leverantören av inbyggd Linux erbjuder advokathjälp i händelse av tvist. Företaget har ett antal egna advokater specialiserade på GPL-problematiken och tar över fallet i händelse av en stämning. Många stora företag som t.ex. Nokia och Ericsson använder Montavista Linux.

- Det finns några kommersiella mikrokärnor utan GPL-licens där Linux och motorstyrningsalgoritmerna exekverar direkt under mikrokärnan som separata trådar.

7.2 Linux för inbyggda system med realtidskrav.

Linux utvecklades från början för att användas på persondatorer med Intel x86-processorer utan realtidskrav och har därför inte goda realtidsegenskaper i grundutförandet. Det finns anpassningar av Linux för ett flertal mikroprocessorer för inbyggda system. En intressant aspekt på utvecklingsarbetet är att samma version av kärnan kan användas både på utvecklings PC:n och i det inbyggda systemet.

För att öka realtidsprestandan finns i huvudsak två metoder :

- Kärnan uppdateras med programvarutillägg som förbättrar realtidsprestandan, s.k. patchar.
- En ny separat realtidskärna införs. Standard Linux exekveras som lågprioriterad tråd och realtidskritisk kod som en högprioriterad tråd.

[2] www.axis.com
[3] www.nvidia.com

7.3 Linux med realtidstillägg

Genom att modifiera kärnan med olika patchar kan realtidsprestandan förbättras. De viktigaste realtidspatcharna har accepterats av Linus Torvalds och lagts in som standard i Linux version 2.6 . På utvecklingskortet finns en Linux 2.4 kärna och patcharna måste då hämtas och installeras manuellt om så önskas. Det är framförallt Andrew Mortons low-latency patch [15] som anses ha gynnsam inverkan på interrupt latency.

Genom att förbättra kärnan kan den göras mer preemptive, men då kärnan är stor och komplicerad är det svårt att garantera ”hård realtid” i alla lägen. Kärnan uppdateras dessutom ofta och det kan vara svårt att förutse vilken inverkan en uppdatering har på interrupt latency.

Om det inte är önskvärt att ladda hem patcharna i egen regi kan man köpa en kommersiell distribution där detta är gjort. Den största kommersiella leverantören av Linux för inbyggda system är Montavista.

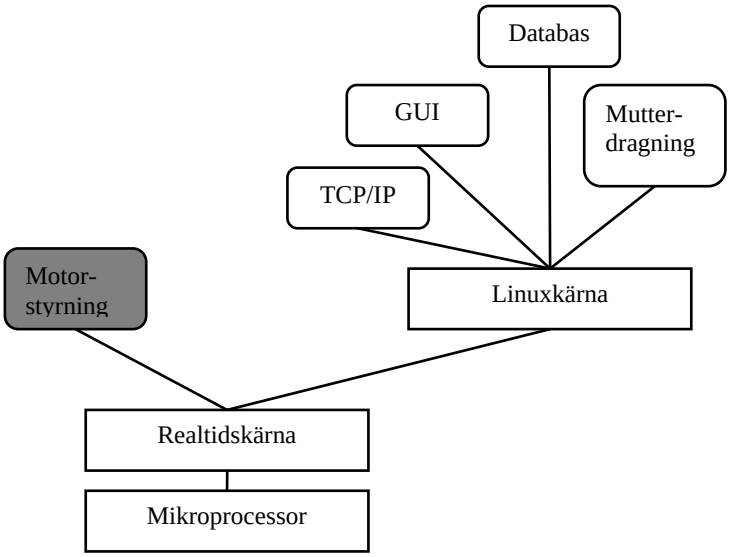
7.3.1 Montavista

Montavista ökar realtidsprestandan genom uppdateringar till kärnan. Montavista är royalty-fritt, en årlig avgift debiteras för support och man tar också betalt för utvecklingslicenser. En Eclipse-baserad utvecklingsmiljö medföljer. En fördel med Montavista är att kostnadsfri advokathjälp erbjuds vid eventuell GPL-tvist. Företaget har ett antal egna advokater specialiserade på GPL. Montavista används bl.a. av Ericsson och Nokia. Svensk distributör är Nohau i Malmö.

I Oktober 2004 startades ett open-source projekt för att åstadkomma markant bättre realtidsegenskaper, dvs interrupt latency långt under 100 µs. Detta kommer dock troligen inte att uppnås för en IST i user-space utan det är för en ISR i kernel-space som siffran skall relateras. Första versionen beräknas utkomma sommaren 2005. Resultatet av projektet är intressant att studera och även huruvida Linus Torvalds godkänner patcharna så att de läggs in som standard i Linux. Montavista har sagt att projektet kommer att genomföras även om patcharna inte läggs in som standard. Om de inte läggs in som standard kan dock framtida kompatibilitetsproblem uppstå. En nackdel med projektet är att befintliga standard Linux drivrutiner inte kan användas direkt utan måste skrivas om. Tilläggas bör att projektet är sponsrat av Montavista men programvaruförbättringarna är licensierade under GPL och kan hämtas hem kostnadsfritt.

7.4 Linux med separat realtidskärna

Ett flertal system har en liten realtidskärna i botten enligt Figur 29. Realtidskritisk programkod exekveras som en högprioriterad tråd och standard Linux som en lågprioriterad tråd.



Figur 29: Linux under separat realtidskärna

Fördelen med att införa en ny realtidskärna är att s.k. ”hård realtid” kan garanteras. Nackdelen är att realtidskod i kernel-space inte kan kommunicera direkt med applikationskoden i standard Linux utan interprocess kommunikation måste användas. Realtidskoden kan inte heller anropa Linux API utan ett särskilt begränsat API används ofta. Detta är dock generella nackdelar som inte skall appliceras på motorstyrningen i detta projekt. Om alternativet att förbättra kärnan med realtidstillägg valts så kommer motorstyrningen ändå inte att kunna läggas som en tråd i user-space, utan den hamnar som en modul i kernel-space och det kommer då också att finnas liknande begränsningar. Den mest kända icke kommersiella varianten är RTAI, ett open-source projekt under GPL.

7.4.1 RTAI

RTAI startade som ett utvecklingsprojekt vid DIAPM[4][12] med RTLinux som bas och är icke kommersiellt. RTAI är mycket likt RTLinux men är något mer utbyggt med funktioner för kommunikation mellan processer, bl.a. mailboxar. Största skillnaden är att RTAI inte är kommersiellt och licensierat under GPL.

RTAI finns för följande arkitekturer:

- x86
- PowerPC
- MIPS
- CRIS
- StrongARM
- ARM7(clps711xfamily, Cirrus Logic EP7xxx, CS89712, PXA25x)

Någon version för ARM9 finns inte i dagsläget. På hemsidan uppges att RTAI stöder Cirrus Logic EP7xxx. Detta är huvudkretsen i ett utvecklingskort liknande det vi har fast med en ARM7-processor.

[4] **D**ipartimento di **I**ngegneria **A**erospaziale del **P**olitecnico di **M**ilano

I litteraturundersökningen för RTAI som redovisas i kapitel 8 hittades ett stort antal applikationer utförda med RTAI. Ursprungligen fanns en tanke att testa RTAI i examensarbetet men då ingen portning existerade för vår ARM9-processor utfördes detta inte.

I ett examensarbete på LTH/Axis Communications AB [11] utförs en portering av RTAI till en 100 MHz processor [5]. Max interrupt latency med last uppmäts till 62,9 µs , vilket är lovande.

RTAI-LXRT är en variant av RTAI där all programmering utförs i user-space, även realtidskod. Fördelen är att programmeringen blir mycket enklare jämfört med standard RTAI där realtidsprogrammering utförs i kernel-space. Nackdelen är att systemet blir något långsammare. RTAI-LXRT finns dock inte för ARM-processorer i dagsläget. RTAI kan också köpas packeterad i en kommersiell distribution, ElinOS från SYSGO.[6]

7.4.2 RTLinux

RTLinux började som ett forskningsprojekt 1995 vid New Mexico Institute of Technology. 1998 grundades företaget FSMLabs Inc. som tog över projektet. FSMLabs marknadsför idag RTLinuxPro som kommersiell produkt. RTLinuxPro finns med stöd för ARM920T-kärnan men inte för vårt utvecklingskort[7] som helhet. Det är möjligt att hämta hem en 20 dagars utvärderingslicens.

FSMLabs har också en icke kommersiell variant, RTLinuxFree. Stöd saknas dock för ARM9 i denna variant. Det är också osäkert hur mycket den fria varianten utvecklas. RTLinuxFree jämförs med RTAI i en utvärdering av Daniel Aarno på KTH [3]. Aarno noterade avsevärt högre värde på Max Scheduling Jitter för RTLinuxFree version 3.1 jämfört med RTAI då systemen utsattes för kraftig last.

7.4.3 Green Hills

Green Hills Software, USA har nyligen släppt en variant[8] av sitt realtidsoperativsystem där Linux körs som separat lågprioriterad tråd. Den realtidskärna som används är intressant då Green Hills hävdar att man inte stänger av avbrottshanteringen i realtidskärnan och därför lämnar garanterade värden på Max Interrupt latency för ett antal processorer, dvs man garanterar ”hård realtid”. För en 233 MHz PowerPC 750 anges Max Interrupt Latency till 0,19 mikrosekunder, vilket är mycket lågt. Detta är det enda undersökta realtidsoperativsystem i examensarbetet som verkligen lämnar riktiga garantier på Max Interrupt Latency. Green Hills har kontor i Lund och Stockholm.

7.4.4 Quadros

Nohau AB, Malmö levererar realtidsoperativsystemet Quadros/RTXC som också finns i en variant där Linux körs som separat lågprioriterad tråd.

7.4.5 Sysgo

Det tyska företaget Sysgo[9] har nyligen lanserat PikeOS där Linux kan exekvera som separat lågprioriterad tråd. Svensk partner för Sysgo är Qivalue Technologies AB[10], Sollentuna.

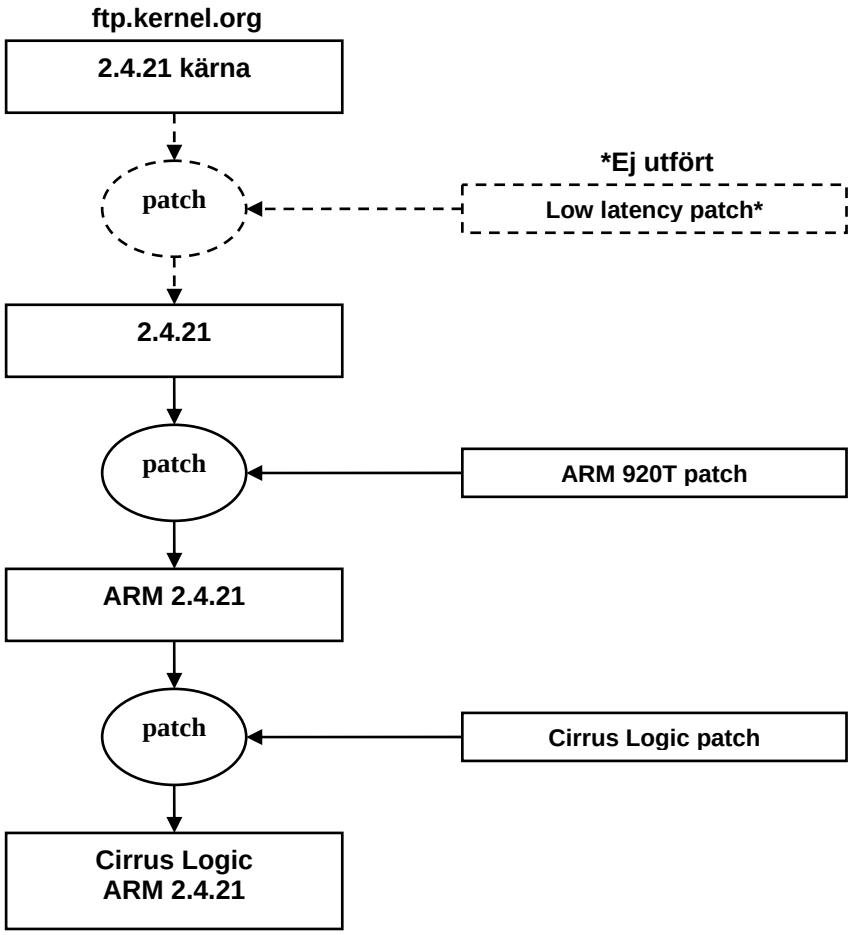
---

[5] ETRAX. Processor utvecklad av Axis Communications AB med inbyggt ethernet.
[6] http://www.sysgo.com
[7] Cirrus Logic EDB9301
[8] Integrity PC
[9] www.sysgo.com
[10] www.qivalue.se

7.5 Linux i projektet

Med utvecklingskortet levererades en Linux 2.4.21 kärna anpassad för utvecklingskortet från Cirrus-Logic. Versionen är en ARM Linux version 2.4.21-rmk1[11] kompletterad med Cirrus Logics hårdvaruanpassning[12] till utvecklingskortet. ARM Linux[13] är ett projekt för portning av Linux till ARM-processorer.

Genom att använda GNU-programmet patch modifieras källkoden till den ursprungliga Linuxkärnan. Figur 30 visar hur kärnan modifierats för att passa vårt utvecklingskort. Den ursprungliga tanken i projektet var att addera en low-latency patch för att öka realtidsegenskaperna. Detta genomfördes inte pga tidsbrist och patchen är därför ritad streckad i figuren.

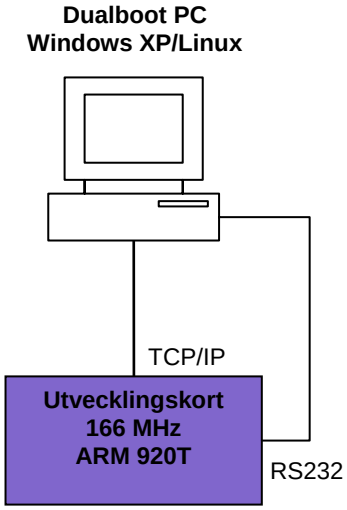


Figur 30: Uppbyggnad av Linuxkärna med patchar

[11] rmk1 = Russel King, huvudansvarig för ARM Linux projektet.
[12] cirrus-arm-linux-1-4-2
[3] http://www.arm.linux.org.uk/

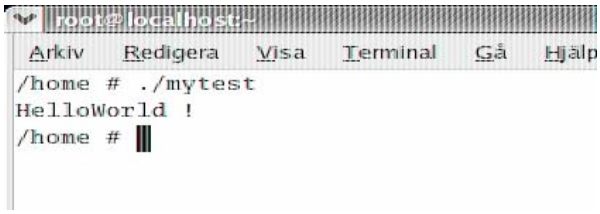
7.5.1 Utvecklingsmiljö

Med utvecklingskortet följer ett antal script och verktyg för att bygga kärnan. För att använda dessa krävs Linux som utvecklingsmiljö. Konfigurerings-scripten som medföljer utvecklingskortet är anpassat för distributionen Red Hat. Red Hat 9 installeras på en persondator som dualboot, dvs vid uppstart väljs om Windows XP eller Linux ska startas.



Figur 31: Utvecklingsmiljö Linux

En Bootloader (Redboot) laddas ned till utvecklingskortet via RS232. Därefter överförs Linuxkärnan med ftp. Tända/släcka en lysdiod och utskrift av Helloworld ! avslutar de praktiska övningarna.



Figur 32: HelloWorld !

7.5.2 Interrupt latency

Att utföra mätningar av interrupt latency som med Windows CE kunde inte genomföras pga tidsbrist. En nyligen publicerad undersökning [16] med Linux 2.6.8.1 kärnan, vilken har de flesta realtidspatchar inbyggda gav dock följande resultat:

- Max Interrupt Service Latency med last = 49,5 µs med en Intel Celeron 650 MHz processor.

Det är svårt att översätta en uppmätt interrupt latency för en specifik processor/klockfrekvens till en processor av annat fabrikat och klockfrekvens, men ARM processorn i projektet är nästan 4 gånger långsammare än den testade Intel processorn. Vi kommer med all säkerhet att erhålla interrupt latency över 100µs vilket är alldeles för långsamt för motorstyrningen. En konsult med tio års erfarenhet av inbyggd Linux [13] anser också att enbart realtidspatchar inte kommer att räcka till för Atlas Copco:s realtidskrav.

8 Litteraturuppgift

Uppgiften är att hitta exempel på realtidssystem med samplingsfrekvenser över 2 kHz implementerade med en mikroprocessor under ett RTOS.

Utvärdering av realtidslinux för robotstyrning, KTH

Evaluation of Real-Time Linux Derivatives for use in robotic control [3].

Utvärdering av tre Linuxvarianter för realtidsapplikationer, RTAI, RTLinux och KURT.

Kansas University Real-Time Linux, KURT bedöms fortfarande vara på försöksstadiet. Författaren konstaterar att han inte får någonting att fungera exakt som han vill . RTAI och RTLinux befinns vara tekniskt mycket lika.

Dock noterades avsevärt högre värde på Max Scheduling Jitter för RTLinuxFree version 3.1 jämfört med RTAI då systemen utsattes för kraftig last. Med Scheduling Jitter avses tidsavvikelsen för en periodisk tråd programmerad att exekvera med periodtiden ***t***. I det ideala fallet exekverar tråden alltid exakt vid perioden ***t*** men den kommer i praktiken att starta lite tidigare eller senare, med en statistisk spridning kring den korrekta tidpunkten. Då scheduleraren styrs av en hårdvarutimer som genererar ett periodiskt interrupt beror scheduling jitter till stor del på variationer i interrupt latency.

Författaren väljer RTAI för att implementera en robotstyrning [12].

Robotstyrning, KTH

Control of a PUMA 560 using Linux Real-Time Application interface (RTAI) [4].

I en tidigare utvärdering [3] valdes RTAI som lämpligast av tre testade RTOS för att implementera en industrirobotstyrning på KTH. RTAI implementeras på en 400 Mhz Pentium II persondator. Sampling sker med 2 kHz. Resultatet blir lyckat och författaren drar bl.a följande slutsatser:

- Scheduling Jitter ligger under 10µs, dvs mycket lågt.
- Systemet är stabilt och hänger sig aldrig under testen.
- Systemet klarar av att hantera en reglerapplikation med samplingsfrekvensen 100Hz samtidigt som robotstyrningen fungerar klanderfritt.

Examensarbete, Chalmers Tekniska Högskola/Volvo Cars.

A real-time platform for control of combustion phasing in a SI-engine [5].

Ett reglersystem för styrning av tändvinklar i en bensinmotor implementeras med realtidsoperativsystemet RTAI på en standard PC. Systemet testas på en Volvo V70-motor och fungerar enligt önskemål. Prestanda på persondatorn anges inte. Samplingshastigheten är svår att härleda i rapporten men är troligen minst 2kHz.

Examensarbete, Linköpings Tekniska Högskola

Hard Realtime Rapid Prototyping Development Platform [6].

En plattform för test av regleralgoritmer genererade av Simulink implementeras på en persondator. Simulink genererar kod antingen för RTAI eller för RTLinux. RTAI eller RTLinux körs sedan på Pc:n och ett standard datainsamlingskort används för in och utsignaler. Samplingshastigheter upp till 50 kHz testas. Processorn är en Ahtlon 1900+.

Översikt, National Institute of Standards and Technology.

Introduction to Linux for Real-Time Control.
Introductory Guidelines and Reference for Control Engineers and Managers [7].

En omfattande utredning om Linux för realtidsapplikationer. Även icke Linux RTOS behandlas. Ett flertal applikationer med RTAI och RTLinux beskrivs kortfattat. Applikationen med högst samplingshastighet var ett högupplösande mikroskop på ett universitet i Tyskland. Samplingshastigheten var 5 KHz och RTLinux användes som RTOS. Systemet implementerades på en 233 MHz Pentium.

Robotstyrning, Lunds Tekniska Högskola

Institutionen för reglerteknik vid LTH har ett robotlaboratorium där man testat RTAI för styrning av en industrirobot. Framgångsrika tester har utförts med samplingshastigheter över 10 kHz. Processorn är en 450 MHz PowerPC [8].

E. Bianchi, L. Dozio, DIAPM Milano

Some experiences in fast hard real-time control in user space with RTAI-LXRT [9].

RTAI-LXRT är en variant av RTAI där all programmering utförs i user-space, även realtidskod. Fördelen är att programmeringen blir mycket enklare jämfört med standard RTAI där realtidsprogrammering utförs i kernel-space. Nackdelen är att systemet blir något långsammare. RTAI-LXRT finns dock inte för ARM-processorer i dagsläget. I undersökningen ges exempel på applikationer med samplingshastighet upp till 12 kHz.

Arcticus

Företaget Arcticus i Stockholm uppgav att deras RTOS, Rubus använts vid ett projekt med hög samplingshastighet (upp mot 10kHz). De kunde dock inte ange exakta siffror. Implementationen var en styrning av tändvinkeln för en motorcykel. Det speciella med Rubus är att det består av 3 st. realtidskärnor med olika prioritet. Högst prioritet har kärnan som endast exekverar periodiska avbrott. En kärna exekverar övriga yttre avbrott med näst högst prioritet och den tredje kärnan exekverar mjukvaruavbrott med lägst prioritet. Det är valfritt att implementera de kärnor man behöver. Arcticus anser att genom att bara ta emot periodiska avbrott i kärnan med högst prioritet hanteras dessa avbrott snabbare än med traditionella RTOS med bara en kärna. Bland kunderna nämndes Haldex, Hägglunds och Volvo Lastvagnar i trollhättan [10].

Examensarbete, Lunds Tekniska Högskola/Axis.

Real-Time Linux in an Embedded Environment.
A port and evaluation of RTAI on the CRIS Architecture [11].

RTAI portas till Axis Communications egenutvecklade ETRAX processor. Processorn är en 100 MHz 32-bitars RISC med inbyggd nätverkshantering. Ett flertal mätningar på Interrupt latency med och utan last utförs. Mätningarna utförs inte med hög samplingshastighet, det intressanta är jämförelserna med en vanlig Linuxkärna och påverkan med last. I undersökningen noteras följande värden på Interrupt Latency med last:

µs	Min	Max	Medel	Median	Std. Av.
RTAI	4,4	64,9	17,9	18,5	5,7
Linux	5,2	162,9	14,3	11,3	6,4

Tabell 9

Intressant är att maxvärdet (Worst Case Interrupt Latency) för RTAI på 64,9 µs är avsevärt lägre än för standard Linux (162,9 µs). Att medelvärdet för RTAI är några µs högre än för vanlig Linux beror på införandet av HAL (Hardware abstraction layer) i RTAI.

8.1 Sammanfattning litteraturuppgift

De källor som identifierades var nästan uteslutande från akademiskt håll med RTAI och RTLinux Free som RTOS. Det akademiska intresset beror troligen på att RTAI och RTLinuxFree är icke-kommersiella produkter utan licenskostnad. En intressant aspekt på RTLinux är att den finns som kommersiell produkt, *RTLinuxPro* vilket kan vara intressant för företag som vill ha kommersiell support. Att hitta information från privata företag visade sig vara mycket svårt.

Av undersökningarna dras slutsatsen att RTAI har något bättre teknisk prestanda (Scheduling Jitter) än RTLinux och skulle vara intressant för implementation och test. Tyvärr fanns ingen portning av RTAI för den ARM9-processor som utvecklingskortet i projektet var bestyckat med. Detta alternativ blev därför inte aktuellt.

9 Slutsatser

9.1 Windows CE 4.2

Prestandamässigt går det att realisera systemet med endast en processor och Windows CE som realtidsoperativsystem. Windows CE är ett realtidsoperativsystem med bra realtidsegenskaper. En intressant svaghet i samband med nästlade avbrott identifierades dock i undersökningen. Realtidskärnan tar för lång tid på sig att avbryta en lågprioriterad avbrottsrutin då ett högre prioriterat avbrott anländer. Det går att komma runt problemet genom att inte ha för mycket programkod i avbrottsrutinerna, precis som dokumentationen för Windows CE föreskriver.

Erfarenheter och viktiga fakta om Windows CE redovisas här:

- Goda realtidsegenskaper, Max Interrupt Latency uppmättes till 32,2 µs
- All arkitekturberoende kod ligger samlad i ett Hardware Abstraction Layer (HAL) vilket ger en bra struktur på operativsystemet.
- Relativt lågt foot-print (flashminne). 300kb med minimal kärna.
  1 Mbyte med Web-, Telnet- och Ftp-server.
- Ett stort antal kommunikationsprotokoll och applikationer medföljer.
- Relativt låg licenskostnad.
- Stöd för många av Microsofts teknologier som MFC, ATL, COM, DCOM och .NET Compact Framework
- Microsoft uppger att 2 miljoner rader källkod bifogas operativsystemet. Det är tyvärr inte all källkod som skickas med. I WinCE 4.x och bakåt är källkoden till för debugging och granskning. I WinCE 5.0 finns det möjlighet att modifiera källkoden , dock kräver detta ett särskilt licensavtal med Microsoft.
- Att lägga till protokoll och applikationer sker med ”drag and drop”. Detta gick visserligen snabbt men då en komponent skulle tas bort uppstod svårigheter. Andra komponenter hade beroenden till denna komponent och det var problematiskt att få bort den. Windows CE talade inte om vilka komponenter det var som hade beroenden, bara att de fanns. Det fanns också en ”custom wizard” där man kunde välja vilka komponenter som skulle ingå då projektet startades upp. Denna ”wizard” brydde sig i praktiken inte så mycket om utförda val utan lade upp komponenter efter någon egen intern mall. Nu gick det att komma runt problemet genom att starta med minsta möjliga kärna och sedan lägga till komponenter efter att projektet skapats. Dock har systemet med ”wizards” och ”drag and drop” klara nackdelar. Det som egentligen sker under ytan döljs, vilket skapar osäkerhet och problem då komplikationer uppstår. Kontrollen på systemet minskar ju mer som döljs.
- Medföljande online-dokumentation är omfattande men ostrukturerad och ytlig. Vad som egentligen sker längre ner i systemet beskrivs inte.

- Tillgången till litteratur är minimal. Ett fåtal böcker finns på marknaden.

- Support i form av konsulter och andra personer som arbetar med Windows CE i Sverige är begränsad då relativt få företag använder Windows CE i Sverige. Prevas i Västerås är den konsultfirma som är specialiserad på operativsystemet. Nyhetsgruppen på Microsofts hemsida fungerade dock bra. Denna nyhetsgrupp övervakas av personal knuten till Microsoft och jag fick alltid respons på mina frågor. För OEM-kunder finns möjlighet att få support mot betalning.

- Komplicerad konfigurering med väldigt mycket inställningar och finesser gör att det tar lång tid för en nybörjare att komma igång med Windows CE. Jag upplevde systemet som svårt och komplicerat att arbeta med.

- Utvecklingsmiljön (kompilering, länkning) är komplicerad och ”buggig”. Ett fullständigt bygge tog 18 minuter. Enligt Microsoft skall den långa byggtiden vara åtgärdad i version 5.0.

- Ett flertal applikationer för att debugga målsystemet finns att tillgå. Många av dem fungerade dock inte alltid. Varför utreddes inte, men andra hade liknande problem att döma av korrespondensen på nyhetsgruppen.

Branschtidningen Dedicated Systems utförde år 2002 en test[14] på Windows CE.Net 4.0. På förstasidan av rapporten summerades följande 3 punkter:

- Bra stöd för många olika plattformar.
- Generellt sett bra realtidsprestanda.
- Otillräcklig dokumentation för ett så komplext system.

[14] www.omimo.be/encyc/BuyersGuide/RTOS/Evaluations/downloaddocpreview.asp?DOC=sample_ce400_252

9.2 Linux

- Dåliga realtidsegenskaper i grundutförandet. Detta pga att Linux från början inte är designat som ett RTOS. Olika lösningar på problemet finns dock.

- Strukturen i Linux är inte lika klar som i Windows CE. Något klart definierat hårdvarugränssnitt finns inte. Arkitekturberoende kod finns inlagd på många olika ställen i kärnan.

- Footprint i projektet blev 2Mb flashminne med Telnet och ftp-server, dvs avsevärt mer än för Windows CE. Genom att optimera linuxkärnan skulle man kunna minska footprint något. Detta utfördes inte i projektet.

- GPL-licensen gör att man inte kan länka drivrutiner statiskt till kärnan utan att publicera källkoden. Detta går dock alltid att komma ifrån genom att länka in kod dynamiskt efter att kärnan startats. Att länka in kod dynamiskt har blivit en defakto-standard som används av många stora företag idag.

- Tack vare GPL-licensen finns en mängd kostnadsfria applikationer och kommunikationsprotokoll med tillhörande källkod på marknaden. Utbudet av kommersiella produkter är också stort.

- Ingen licenskostnad. Standardverktyg för editering, kompilering och länkning finns också att tillgå utan kostnad.

- Bra kontroll över kompilering och länkning jämfört med Windows CE. All konfigurering ligger i textfiler vilket ger bra kontroll över systemet.

- All källkod till operativsystemet finns tillgänglig.

- Ingen online dokumentation som i fallet med Windows CE. En katalog med dokumentation finns med en hel del information. Källkoden är generellt sett dåligt kommenterad.

- God tillgång till litteratur.

- Samma kärna på en persondator som för ett inbyggt system. Detta att jämföra med Windows CE och Windows XP/NT/2000. Detta gör det snabbt och enkelt att utveckla och testa delar av ett program för ett inbyggt system på en persondator.

- God tillgång på linuxkompetenta utvecklare då Linux är vanligt både i inbyggda system och på persondatorer. Ett antal konsultfirmor med kompetens inom Linux finns också.

- Lång livscykel. Då all källkod finns att tillgå upphör inte supportmöjligheterna för att en viss leverantör bestämmer detta.

- Leverantörsoberoende. Ett antal kommersiella distributioner för inbyggda system finns att tillgå. Genom att själv hämta hem linuxkärna, editorer, kompilatorer, debuggers m.m. uppnås totalt leverantörsoberoende.

- Bra möjligheter till fjärradministration. Operativsystemet styrs med textkommandon vilket ger mycket goda möjligheter till enkel och stabil fjärradministration. Ett stort utbud av programvaror finns att tillgå för detta.

9.3 Rekommendation

Rapporten visar att prestandamässigt skulle Windows CE kunna användas för att implementera systemet på en processor. Personligen anser jag dock att Windows CE är svårhanterbart och inte lämpligt för Atlas Copco. Den svaghet i samband med nästlade avbrott som identifierades i undersökningen talar också emot Windows CE.

Jag anser att det intressantaste Linux-alternativet är någon variant med mikrokärna, där Linux exekverar som en lågprioriterad tråd. Med detta upplägg blir de realtidskritiska delarna tydligt separerade från övrig mjukvara. Om en kommersiell mikrokärna väljs kommer man ifrån alla eventuella GPL-tveksamheter.

9.4 Tack

Förutom mina handledare på Atlas Copco och Umeå Universitet har följande personer bidragit med hjälp i examensarbetet:

Fredrik Söderberg, Ericsson, Älvsjö. *Linux*.
Anders Törnqvist, Qivalue Technologies, Sollentuna. *Linux*.
Claes Lundholm, ESTC, Västerås. *Linux.*
Ulrik Sjölin, Joakim Nilsson Nohau, Malmö. *Linux*.
Klas Nilsson, LTH, Lund. *RTAI*.
Per-Olof Stendahl, Prevas, Västerås. *Windows CE*.
Magnus Olofstam, Beijer Electronics, Malmö. *Windows CE.*
Martin Sigrand, Michael Jons Atlas Copco Tools.

Jag har säkerligen missat några personer men det är i så fall utan ont uppsåt.

Slutligen vill jag tacka Umeå Universitet för att de ordnade med fjärredovisning via Internet så att jag slapp åka hela vägen till Umeå.

Referenser

[1] Michael Jons , KTH
Master of Science Thesis, 2005.
Motorstyrning med realtidsoperativsystem.

[2] "Spindeln i realtidsforskarnätet".
Artikel i Elektroniktidningen 2003-08-12.
Författare Jan Tångring.

[3] Daniel Aarno, KTH
Academic paper, 2004
Evaluation of Real-Time Linux Derivatives for use in robotic control.
http://www.e.kth.se/~e98_aar/resources/rtos.pdf Besökt Mars 2005.

[4] Daniel Aarno, KTH
Academic paper, 2004
Control of a PUMA 560 using Linux Real-Time Application interface (RTAI).
http://www.e.kth.se/~e98_aar/resources/rtcontrol.pdf Besökt Mars 2005.

[5] Lars Bergström, Chalmers Tekniska Högskola.
Master of Science Thesis, 2004.
A real-time platform for control of combustion phasing in a SI-engine.
http://db.s2.chalmers.se/download/masters/master_036_2004.pdf Besökt Mars 2005.

[6] Christer Rosenquist, Linköpings Tekniska Högskola.
Master of Science Thesis, 2003.
Hard Realtime Rapid Prototyping Development Platform.
http://www.vehicular.isy.liu.se/Publications/MSc/03_EX_3377_CR.pdf
Besökt Mars 2005.

[7] National Institute Of Standards & Technology.
Intelligent systems division, 2002.
Introduction to Linux for Real-Time Control.
Introductory Guidelines and Reference for Control Engineers and Managers.
http://www.aeolean.com/html/RealTimeLinux/RealTimeLinuxReport-2.0.0.pdf
Besökt Mars 2005.

[8] Klas Nilsson, LTH.
Institutionen för Datavetenskap.
http://www.cs.lth.se Besökt Mars 2005.

[9] E. Bianchi, L. Dozio.
Some experiences in fast hard real-time control in user space with RTAI-LXRT.
http://www.rtai.org/documentation/conferences/App_Orlando00.pdf Besökt Mars 2005.
DIAPM, Milano.

[10] Mats Lindberg, Arcticus.
http://www.arcticus.se/ Besökt Mars 2005.

[11] Martin P Andersson och Jens-Henrik Lindskov.
Master of Science Thesis, 2003. Lunds Tekniska Högskola/Axis Communications.
Real-Time Linux in an Embedded Environment.
A port and Evaluation of RTAI on the CRIS Architecture.
http://www.control.lth.se/~mandersson/rtai/ Besökt Augusti 2005.

[12] DIAPM RTAI project.
http://www.rtai.org Besökt Augusti 2005.

[13] Anders Tornquist, QiValue TechnologiesAB.
http://www.qivalue.se Besökt Augusti 2005.

[14] Karim Yaghmour, Building Embedded Linux Systems, O`Reilly First Edition, 2003

[15] Clark Williams, ”Which is better -- the preempt patch, or the low-latency patch ? ”
20 Mars, 2002.
http://www.linuxdevices.com/articles/AT8906594941.html . Besökt Augusti 2005

[16] Peter Laurich, ”A comparison of hard real-time Linux alternatives”
19 Nov, 2004.
http://linuxdevices.com/articles/AT3479098230.html . Besökt Augusti 2005

Appendix A

Testprogram

Uart2test.txt

```
rsssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssn
```

Ftpload.bat

På varje rad anropas filen ftpcommands.txt där ftp-kommandona utförs för att hämta en fil. Då iterativ loop saknas i MSDOS sker anropen ett antal gånger då vi vill ha en filöverföring som tar några timmar.

```
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
ftp -s:ftpcommands.txt
```

Ftpcommands.txt

Filen *ep9301_user_guide.pdf* måste överföras manuellt till målsystemet med ftp innan testen kan startas.

```
open edb9301
user hej
1234
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
get ep9301_user_guide.pdf
quit
```

Appendix B

Ordlista

ATL	Active template Library. En samling C++ template klasser för att skapa objekt som vektorer, stackar och köer.
BSP	Board Support Package. Mjukvarupaket med funktioner anpassade för specifik hårdvara.
DLL	Dynamic Link Library. Programkod som laddas in dynamiskt vid behov.
DSP	Digital Signal Processor.
footprint	Antal byte ett operativsystem upptar på fysiskt lagringsmedia.
FIFO	First In First Out. Benämning på buffert där först inmatade data skickas ut först.
IRQ	Interrupt Request. Avbrottsbegäran från hårdvaruenhet till processorn.
ISR	Interrupt Service Routine. Exekverar i kernel-space utan minnesskydd. Anropas av kärnan i realtidsoperativsystemet direkt efter ett inkommande avbrott.
IST	Interrupt Service Thread. Exekverar i user-space som tråd med minnesskydd. Triggas av en ISR. Schemaläggs som en normal tråd av kärnan i realtidsoperativsystemet.
Kernel-space	Programkod som exekverar utan tilldelad prioritet och minnesskydd sägs exekvera i kernel-space. Exempel på kod är avbrottsrutiner, drivrutiner och realtidsoperativsystemets egen programkod.
MFC	Microsoft Foundation Classes. Bibliotek med C++ klasser från Microsoft.
MMU	Memory Management Unit. Hårdvarukrets som översätter fysiska adresser till virtuella adresser.
POSIX	Portable Operating System Interface. Standardiserat gränssnitt för Unix och andra operativsystem. Tillämpningar som skrivits för ett Posix-godkänt operativsystem ska kunna köras utan omkompilering på andra Posix-godkända operativsystem. De flesta Unix-varianter är Posix-godkända, liksom Linux.
RTOS	Real Time Operating System. Realtidsoperativsystem
UART	Universal Asynchronous Receiver/Transmitter. Hårdvarukrets för att skicka och ta emot data via RS232 seriekommunikation.
User-space	Programkod som exekveras av ett RTOS med tilldelad prioritet och minnesskydd sägs exekvera i user-space. Oftast exekverar all applikationskod i user-space.