CHALMERS

Institutionen för data och informationsteknik.
Roger Johansson, Ulf Assarsson

2019-11-20

Maskinorienterad programmering

Laborationer

Detta laborations-PM innehåller anvisningar om förberedelser inför genomförande av laborationerna, såväl som de uppgifter du ska utföra och redovisa under laborationerna.

Laborationsserien omfattar totalt fem laborationsmoment som utförs i tur och ordning under fem laborationstillfällen

Inför varje laborationstillfälle ska du förbereda dig genom att noggrant läsa igenom anvisningarna för laborationsmomentet i detta PM.

Bortsett ifrån det första laborationstillfället förutsätts det att du inför varje laboration har genomfört ***omfattande*** laborationsförberedande hemuppgifter. Vilka dessa uppgifter är, framgår av detta PM. Du ska vara beredd att visa upp och redogöra för dina förberedda lösningar inför laborationstillfället. ***Bristfälliga förberedelser kan medföra avvisning från bokad laborationstid.***

Underskrifterna på detta försättsblad är ***ditt*** kvitto på att du är godkänd på respektive laborationsmoment. Spara det, för säkerhets skull, tills slutbetyg på kursen rapporterats. **Börja med att skriva ditt namn och personnummer med bläck.**

_____ _____

Personnummer Namn (textat)

Följande tabell fylls i av laborationshandledare efter godkänd laboration.

Laboration	Godkännande av laboration	
	Datum	Laborationshandledares underskrift
1		
2		
3		
4		
5		

Godkännande – hel laborationsserie:

_____ _____

Datum Laborationshandledares underskrift

Översikt av laborationsserien

Under laborationsserien utför du uppgifter där du konstruerar och testar ett enklare dataspel av lite äldre modell.

Kompletterande material

För laborationernas genomförande behöver du, utöver kurslitteraturen, program och diverse tekniska beskrivningar. I kursen används följande program som finns att ladda hem bland annat från kursens resurssida (se kurs-PM):

- *ETERM8* - Integrerad miljö för programutveckling i assemblerspråk
- *GDB* - Gnu Debugger för ARM-processorer
- *CodeLite* – Integrerad utvecklingsmiljö
- *GCC för ARM* – Korsutvecklingsverktyg C, C++ för ARM-processorer
- *GDB SimServer* – *MD407*-simulator för användning med *GDB* (*CodeLite*)

Utöver dessa används på laborationsplatsen (du behöver normalt sett inte installera USBDM på din egen maskin) även:

- *USBDM GDB-Server* - för JTAG-gränssnitt USBDM.

Dessutom finns följande beskrivningar tillgängliga på elektronisk form via kursens resurssida. Läsanvisningar som ges i detta lab-PM avser dessa dokument.

För laborationsdator MD407:

- *MD407* beskrivning
- STM32F4xx Cortex M4 programming manual
- STM32F407 datasheet
- STM32F407 reference manual

För laborationskort som används under laborationsserien:

- Lysdiodstapel ("bargraph")
- Binär till 7-segment
- DIL-switch
- LCD-modul
- Hexadecimal till 7-segment
- Keypad
- IRQ Flip-Flop
- Datablad LCD - ASCII
- Datablad LCD - Grafisk

Laboration 1

Terminalövning: test och felavhjälpning i assemblerprogram

Under denna laboration:

- bekantar du dig med laborationssystemet
- lär dig grunderna för test och felavhjälpning med hjälp av monitor/debugger respektive JTAG-baserade verktyg.

Du kommer att undersöka och prova hårdvara:

- laborationssystemet *MD407* med monitor/debugger *dbgARM*
- microcontroller *ST32F407* med debugger *GDB*
- I/O-enheter *DIL-switch, 7-segmentsdisplay*.

Enklast förbereder du dig genom att:

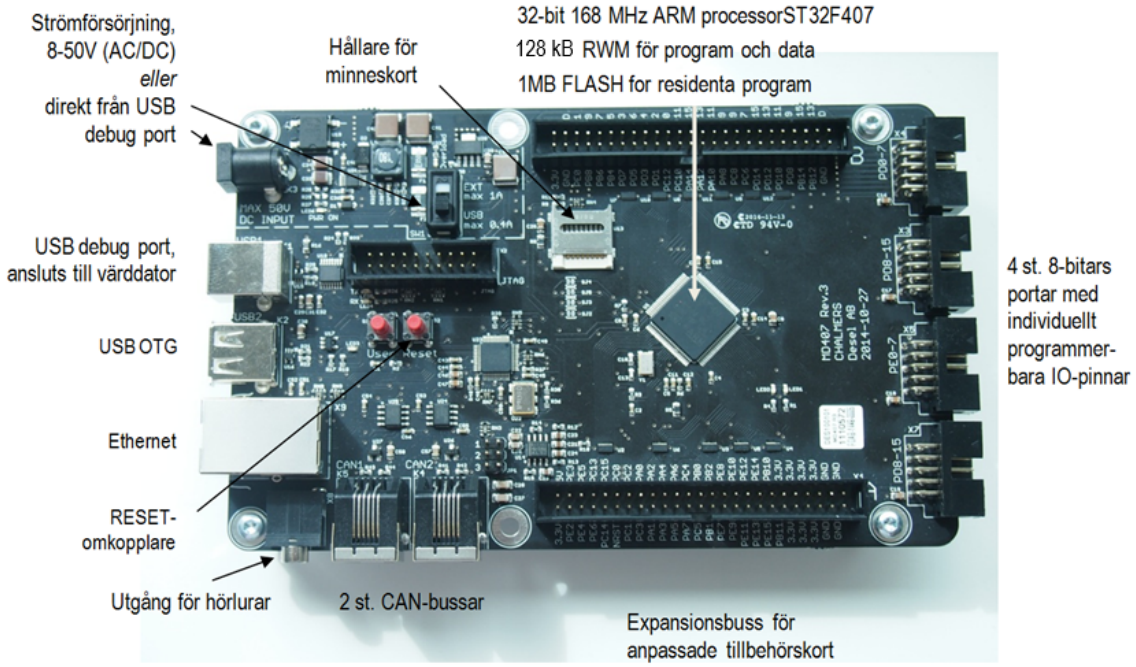
1. Läs först översiktligt igenom detta laborationsavsnitt.
   Orientera dig om de läsanvisningar som ges
2. Arbeta igenom kapitel 1 t.o.m 3 i *Arbetsbok för MD407.* Det rekommenderas att du försöker hinna med att göra samtliga övningsuppgifter i kapitlen. Några av uppgifterna är dessutom direkt förberedande för laborationen.
3. Kontrollera att du utfört de uppgifter som ska vara lösta före laborationstillfället. Detta sparar ansenligt med tid under laborationen och är nödvändigt för att du ska hinna utföra laborationen under den tillgängliga laborationstiden.

Följande laborationsuppgifter ur denna del av laborations-PM skall redovisas för en handledare för godkännande under laborationen. Då alla moment är godkända signeras också försättsbladets laboration 1.

Uppgift i arbetsbok	1.2	3.1	3.2	3.3	-
Laborations-uppgift	1.2	1.4	1.6	1.7	1.8
Sign.					

Översikt av laborationssystemets hårdvara

Laborationsdator *MD407*



Anslutningar

Laborationsuppsättningen ska vara korrekt ansluten då du kommer till laborationsplatsen, men kontrollera för säkerhets skull anslutningarna.

- Laborationssystemet *MD407* ansluts till terminalfunktionen i *ETERM* via anvisad USB-port.
- Laborationskort anslutes till MD407:s portar enligt

  DIL-switch – PE0-7

  Bargraph – PD0-7
- Strömförsörjning sker direkt från USB-anslutningen, det finns därför ingen anslutning till DC-jacket på MD407. Kontrollera att omkopplaren för val av strömförsörjning står i läge USB.

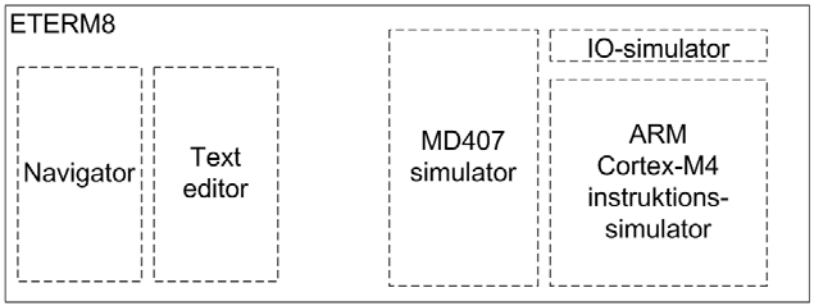
Utvecklingsmiljö

Under denna laboration behandlas uteslutande program skrivna i assemblerspråk. Du får användning för:

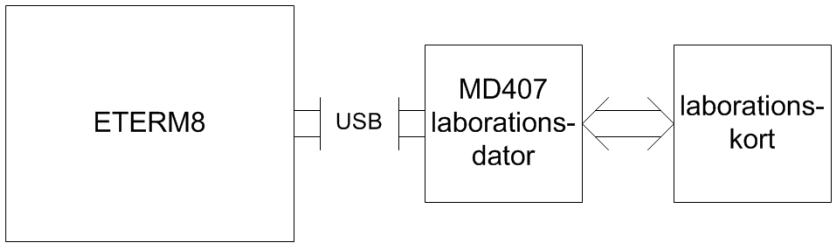
- *ETERM8* - Integrerad miljö för programutveckling i assemblerspråk
- *GDB* - Gnu Debugger för ARM-processorer
- *USBDM GDB-Server* - för JTAG-gränssnitt USBDM.

Du kan förbereda dig för användning av GDB med hjälp av *SimServer*. Denna applikation beter sig på samma sätt som en *USBDM-GDB-Server* men simulerar i stället laborationssystemet. Se arbetsbokens kapitel 3.

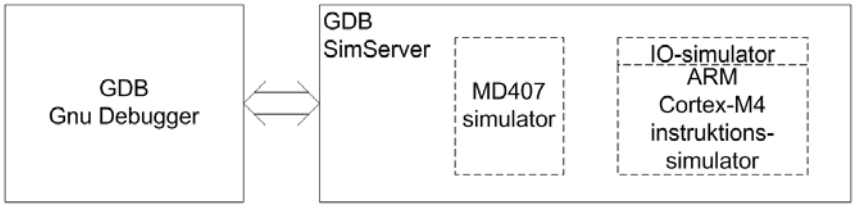
Simulatormiljö:



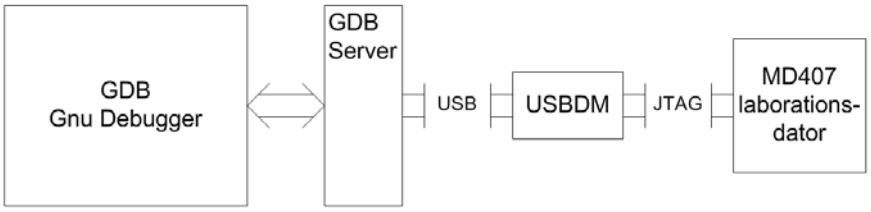
Laborationsmiljö:



Simulatormiljö:

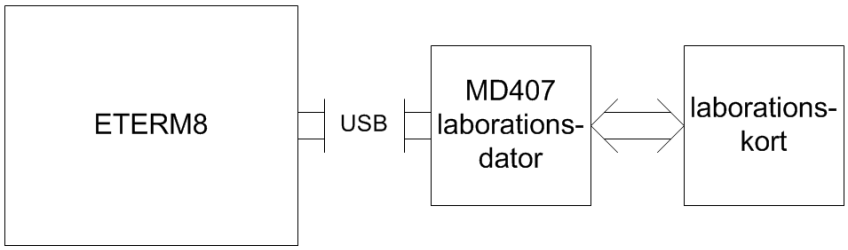


Laborationsmiljö:



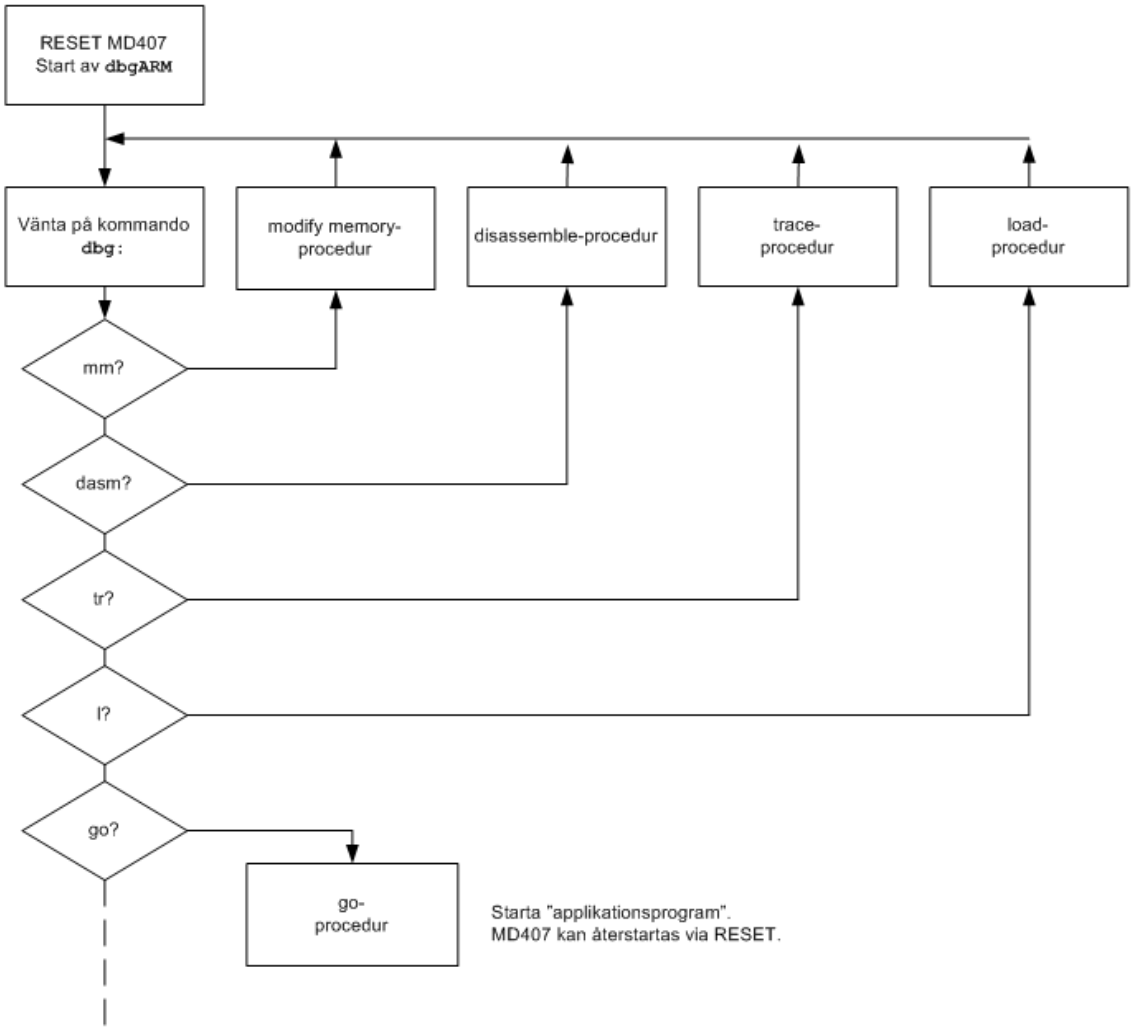
Monitor/debugger *dbgARM*

När vi ansluter *MD407:s* debugport till en USB-port på värdsystemet och startar ett terminalfönster i *ETERM* fungerar värdsystemet som en ”dum terminal” bestående av tangentbord och bildskärm. Vi använder då följande konfiguration:



Allt som skrivs på tangentbordet skickas från *ETERM* direkt ner till *MD407*. I *MD407* finns ett enkelt program som kallas ”monitor/debugger” (*dbgARM*). Programmet är, som namnet antyder avsett för *övervakning* och *test*.

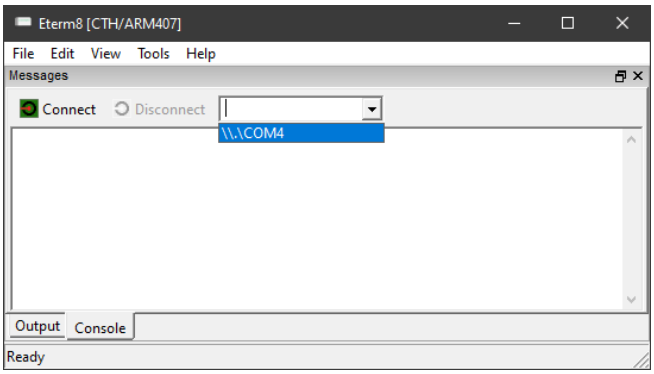
dbgARM ”lyssnar” hela tiden på kommandon från tangentbordet bortsett från när *MD407* är upptaget av något applikationsprogram. För att återgå till *dbgARM* i detta fall krävs RESET (omstart) av *MD407*. Huvudprogrammet i *dbgARM* är utformat som en enkel kommandotolk, där en rad olika kommandon accepteras.



Laborationsuppgift 1.1:

Starta *ETERM* och öppna därefter ett fönster till laborationsdatorn:

- Kontrollera att laborationsdatorn är ansluten till en USB-port.
- Maximera sektionen Messages.
- Välj Console-fliken, öppna combo-boxen. De möjliga COM-portarna visas nu. Vilken port du ska använda kan variera mellan olika miljöer. Fråga laborationshandledaren om du är osäker.
  I detta fall finns bara ett möjligt alternativ, COM4.



- Efter att ha valt COM-port, klicka på 'Connect' för att starta kommunikationsprogrammet:

Om du använder LINUX: Då laborationsdatorn anslutits skapas en ny enhet, typiskt `/dev/ttyUSBx` (x=0,1,2 osv.) Den är dock bara tillgänglig för ROOT. Ändra behörigheten enligt:

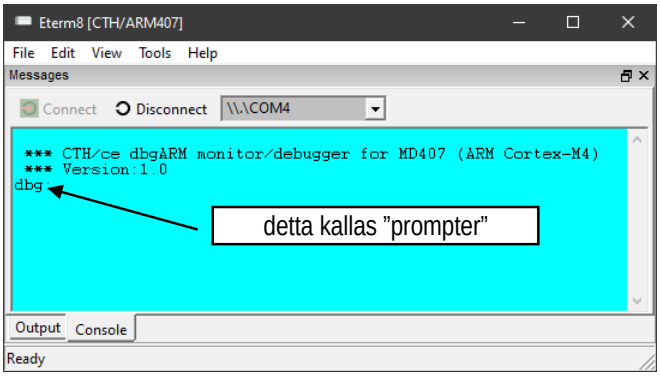
```
sudo chmod 666 /dev/ttyUSBx
```

Skriv enhetens namn, dvs: /dev/ttyUSBx i det lilla fönstret och anslut.

Då *ETERM* anslutit via porten färgläggs fönstrets bakgrund blått.

- Tryck *RESET* på laborationssystemet*:*

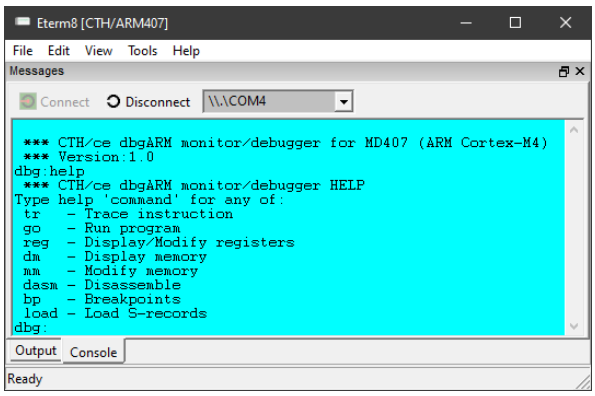
och kontrollera att *debuggern* identifierar sig genom att text skrivs till terminalfönstret. Konsollfönster bör då se ut ungefär som följande figur.
Om ingen text skrivs till skärmen eller om texten är oläslig, kontakta en handledare.



Den sista raden kallas för monitorns *prompter* och innebär att *dbgARM* är redo att ta emot kommandon från dig.

Varje gång du trycker ↵ ska *dbgARM* skriva en ny prompter till terminalen. Om så inte är fallet betyder det att *dbgARM* är upptagen och kanske rentav ”hängt sig” eller ”spårat ur”. Lösningen på ett sådant problem är att göra *RESET* på *MD407*.

- Du kan få hjälp med dbgARM:s kommandon. Skriv: **`help`**↵

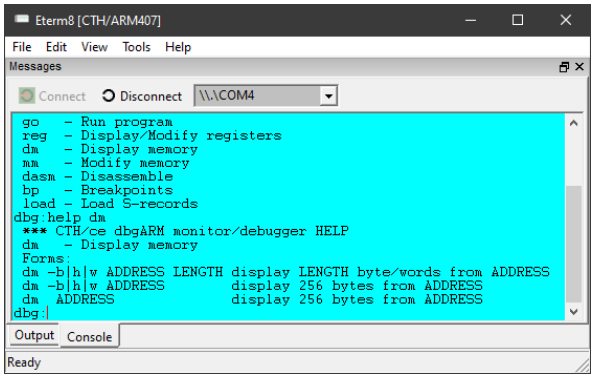


Under denna laboration får du tillfälle att prova *samtliga* dessa kommandon.

Kommandot ” help *kommando"*

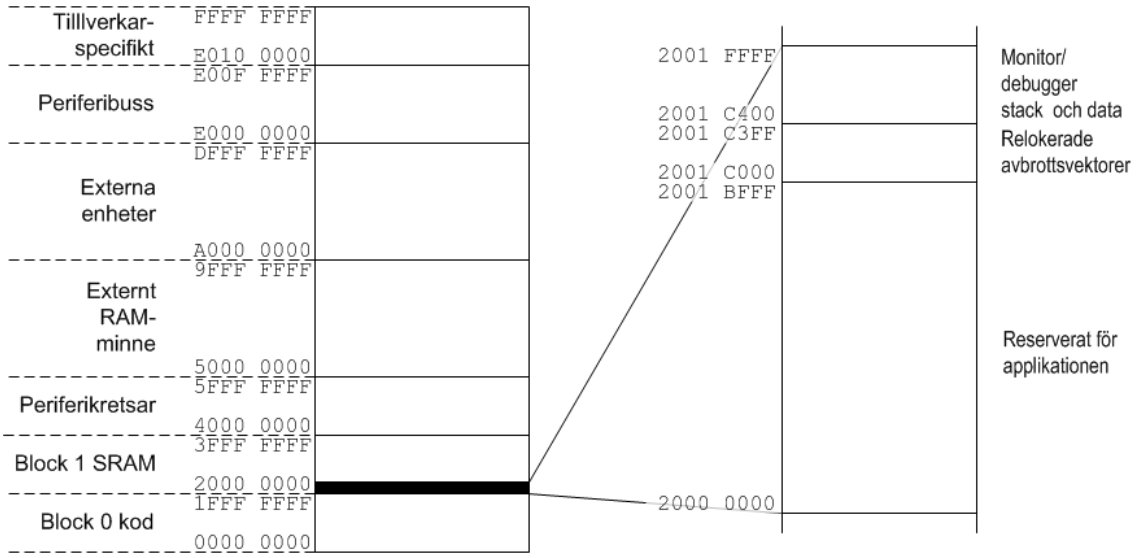
Du kan undersöka ett specifikt kommando genom att skriva: **`help`** ***`kommando`***↵.

Prova exempelvis **`help dm`**↵



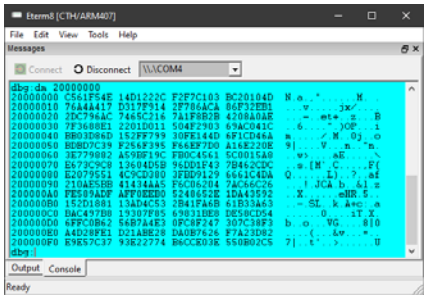
Kommandot ”display memory” (dm)

Du ska nu undersöka minnesinnehållen i några olika minnesområden i systemet. Adressrymden hos *MD407* används enligt följande:



I adressrymden Block 0 kod (FLASH-minne) finns *dbgARM*. Adressrymden Block 1 SRAM används till en liten del för *dbgARM*, men större delen är tillgänglig för att ladda ner och testa program för MD407 (Reserverat för applikation).

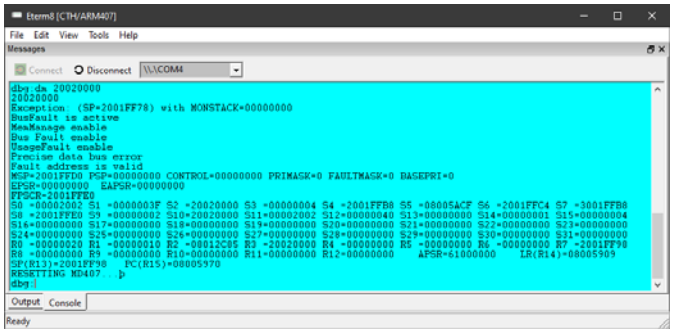
- Skriv: **`dm 20000000`**↵ för att visa innehållet i minnet.



Innehållet är ”nonsens” eftersom det bestäms slumvis då *MD407* spänningssätts. Adressangivelsen anges till vänster, minnesinnehållet i mitten och till höger visas tolkningen i form av ASCII-koder.

Om man försöker använda minne utanför det tillgängliga området kommer processorns minnes-hanteringsenhet att upptäcka detta och avbryta. I *dbgARM* tas detta upp som ett Bus Fault Exception. Prova nu detta, exempelvis med

- Skriv: **`dm 20020000`**↵ för att lösa ut minnesfelshanteringen.

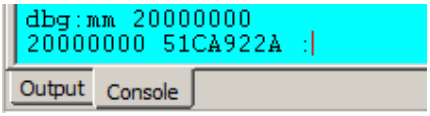


Kommandot ”modify memory” (mm)

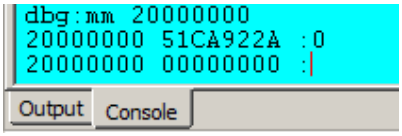
Vi ska nu använda kommandot **mm** för att försöka ändra minnesinnehåll på några olika ställen i *MD407:*s adressrum.

Ge kommandot: **`mm 20000000`**↵för att visa och ha möjlighet att ändra minnesinnehållet på adressen.

Har du samma minnesinnehåll som i figuren till höger?

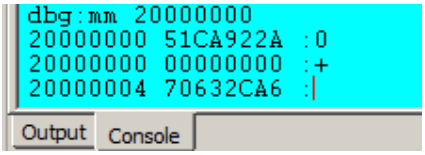


Skriv in ett nytt värde **`0`**↵ för att ändra innehållet på adressen. *dbgARM* svarar med att skriva ut raden på nytt med det nya värdet.



Tryck på **+** (plus) för att visa minnesinnehållet på *nästa* adress.

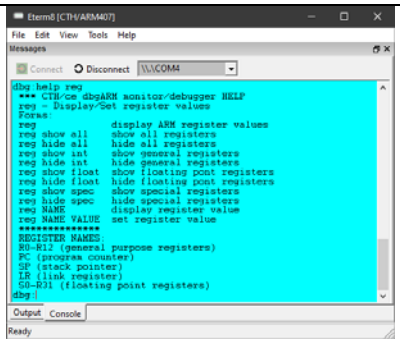
Tryck på – (minus) för att visa minnesinnehållet på föregående adress.



Försök ändra minnesinnehållet på adress 0. Varför går inte det?

Kommandot ”show/change register” (reg)

Kommandot används för att visa eller ändra innehåll i processorns register. Det finns åtskilliga varianter av kommandot, ge kommandot **`help reg`**↵ för en översikt.



`reg`	visa registerinnehåll
`reg` ***`register`***	visa innehållet i ett speciellt register
`reg` ***`register value`***	placera värdet *value* (hexadecimal form) i *register*

Efterson *reg*-kommandot utförs automatiskt vid *trace*-kommandon eller vid stopp på brytpunkt (beskrivs nedan) kan det ibland vara praktiskt att stänga av utskrifterna. Detta gör man genom att ge kommandot:

`reg hide all`↵ : dölj utskrift av alla register

Man kan också ange om man vill visa eller dölja grupper av register:

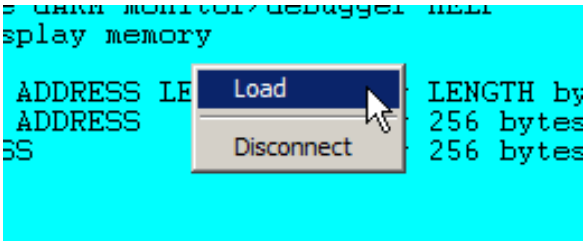
`int`: generella register, `R0` t.o.m. `R12`
`float`: register för flyttal, `S0` t.o.m `S31`
`spec`: speciella register, bland andra: `PC`, `SP`, `LR` och `APSR`.

Prova några kommando för att visa generella och speciella register. Prova också med att ändra registerinnehåll.

Kommandot “load application program” (load)

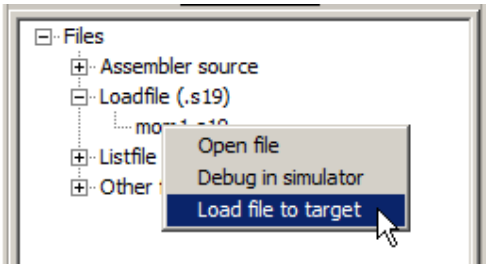
Program och data i form av S-formatfiler, kan överföras till laborationsdatorn med hjälp av load-kommandot. Det finns tre behändiga sätt att ladda en fil till laborationsdatorn:

Högerklicka i det blå fönstret



Nu öppnas en dialog som ger dig möjlighet att välja den fil du vill ladda till laborationsdatorn.

Ett annat sätt är att använda navigatorfönstret, högerklicka på filnamnet under Loadfile:



Om du använder någon annan terminalfunktion än *ETERM8* måste du ge kommandot manuellt, dvs. **`load`**↵, därefter starta en överföring med terminalprogrammet.

Kommandot "disassemble" (dasm)

Disassembleringskommandot översätter minnets innehåll till assemblerinstruktioner. Det kan ges i tre former:

`tr`	utför en instruktion från PC
`tr` ***`adress`***	utför en instruktion från *adress*
`tr` ***`adress`*** **`antal`**	utför *antal* instruktion från *adress*
`tr +`	aktivera *hot trace*

`dasm` ***`adress antal_instruktioner`***

`dasm adress`	disassemblera 16 instruktioner från *adress*
`dasm`	fortsätt disassemblering från där den sist avslutades

Laborationsuppgift 1.2:

I Arbetsbokens uppgift 1.2 har du skapat filen `mom1.asm`. Assemblera denna och ladda (`mom1.s19`) till *MD407*. Kontrollera och ange vad följande register innehåller:

PC	
SP	
LR	

Disassemblera minnesinnehållet fr.o.m. programmets startadress, komplettera följande tabell, identifiera de symboler som definierats och de konstanter som assemblatorn skapat i kolumnen längst till höger:

20000000	4803	LDR	R0,[PC,12#]	start
20000002	4904			
20000004	6008			
20000006	4904			
20000008	4A04			
2000000A	6810			
2000000C	6008			
2000000E	E7FC	B	0x2000000A	
20000010	5555			
20000012	5555			
20000014	0C00			
20000016	4002			
20000018	0C14			
2000001A	4002			
2000001C	1010			
2000001E	4002			

Jämför nu med den listfil (`mom1.lst`) som genererades vid assembleringen, vilka skillnader finner du bortsett från att typsnitt och versaler/gemena kan skilja?

Då programmet laddats till laborationsdatorn kan man börja testa funktionen. Detta sker huvudsakligen på två olika sätt,

- instruktionsvis exekvering, *trace*, vilket innebär att *dbgARM* utför en instruktion. Därefter skrivs en ny prompter och *dbgARM* är beredd att ta emot ett nytt kommando.
- programmet utförs, startas med go-kommandot. Ofta används då också så kallade brytpunkter för att stoppa exekveringen vid någon speciell instruktion.

Kommandot “step (trace) application program” (tr)

trace-kommandot kan ges i flera former:

tr	utför en instruktion från PC
tr *adress*	utför en instruktion från *adress*
tr *adress* antal	utför *antal* instruktion från *adress*
tr +	aktivera *hot trace*
tr –	deaktivera *hot trace*

Då *hot trace* är aktiverad räcker det med att ge kommandot **.** (punkt) vid promptern. Detta tolkas då som kommandot `tr`.

Kommandot “run application program” (go)

go-kommandot används för att starta ett applikationsprogram.

go *adress*	Formen används för att starta ett tillämpningsprogram som börjar på *adress*.
go	Samma som föregående men programmet startas från adress som anges av *användarregistret PC*.
go -n	Utför program *till nästa instruktion*. Speciellt används formen då man vill utföra en hel subrutin.

Kommandot ”breakpoints” (bp)

bp-kommandot används för att *visa, sätta ut* och *ta bort* brytpunkter i tillämpningsprogrammet. En brytpunkt är antingen *aktiv* eller *inaktiv*. Om brytpunkten är aktiv, kommer *dbgARM* att avbryta utförandet av tillämpningsprogrammet då processorn *skall utföra* den instruktion som ersatts med brytpunkt. Brytpunkterna placeras (internt) av *dbgARM* i en *brytpunktstabell*. Maximalt 10 brytpunkter åt gången kan hanteras av *dbgARM*. Brytpunkterna numreras ("namnges") 0 t.o.m.9.

bp	Hela brytpunktstabellen skrivs till skärmen. För varje brytpunkt skrivs, om den används för tillfället, om den är aktiv och dess adress.
bp *num* set *adress*	Används för att sätta ut en ny brytpunkt. *num* skall vara brytpunktens nummer och tolkas av DBG som ett decimalt tal. *adress* är brytpunktens adress. Om en annan brytpunkt tidigare definierats med samma nummer modifieras denna brytpunkt. Brytpunkten är aktiv.
bp *num* dis	Inaktiverar brytpunkt *num* om denna är aktiv. Brytpunkten behålls i tabellen men orsakar inget programavbrott.
bp *num* en	Aktiverar brytpunkt *num* om denna tidigare var inaktiv.
bp *num* rem	Tar bort brytpunkt *num* ur brytpunktstabellen.
bp clear	Tar bort *samtliga* brytpunkter ur brytpunktstabellen.

Laborationsuppgift 1.3:

Kontrollera att adressen i `PC` är 20000000 och ge kommandot `tr`
eller
ge kommandot `tr 20000000` för att utföra den första instruktionen.

Beskriv kortfattat vad som då händer, vad ser du i konsollfönstret? Vilket innehåll har register R0 efter den första instruktionen? Vad innebär den sista radens utskrift?

Identifiera nu adressen till instruktionen `STR R0,[R1,#0]` efter symbolen `main` och sätt en brytpunkt på denna adress. Ange kommandot för detta här:

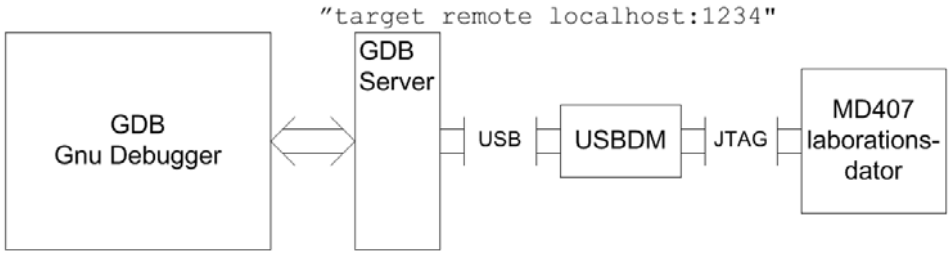
Starta därefter programmet med *go*-kommandot. Beskriv vad som då händer, vad ser du i konsollfönstret?

Ta bort brytpunkten och starta på nytt programmet. Kan du ge kommandon till *dbgARM* nu?

- Tillkalla nu en handledare för godkännande av denna del av laborationen.
- Därefter får du hjälp av handledaren med att koppla om laborationssystemet för resten av laborationen.

Användning av GDB

Vi ska nu övergå till att använda följande konfiguration:



Laborationsuppgift 1.4:

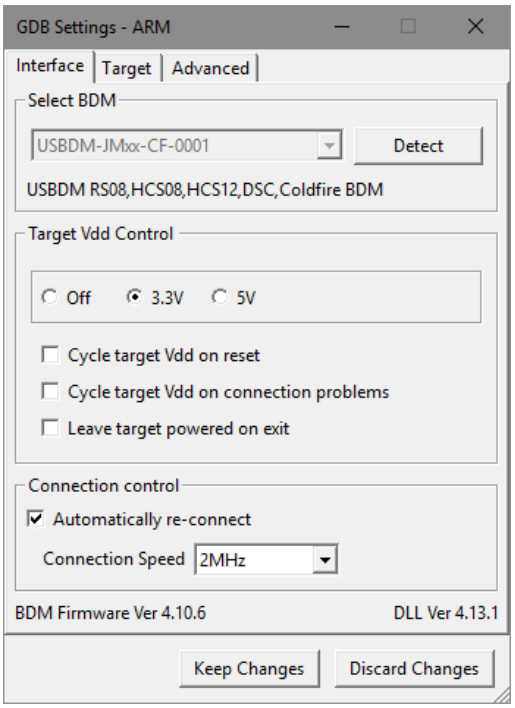
Redigera en källtext enligt följande och spara som fil `gdbtest1.asm`.

```
@  Laborationsuppgift
@  gdbtest1.asm
@

start:
      MOV   R0,#0
main:
      ADD   R0,R0,#1
      B     main
```

Assemblera filen, rätta eventuella fel.

Starta programmet ARM-GDB-Server:



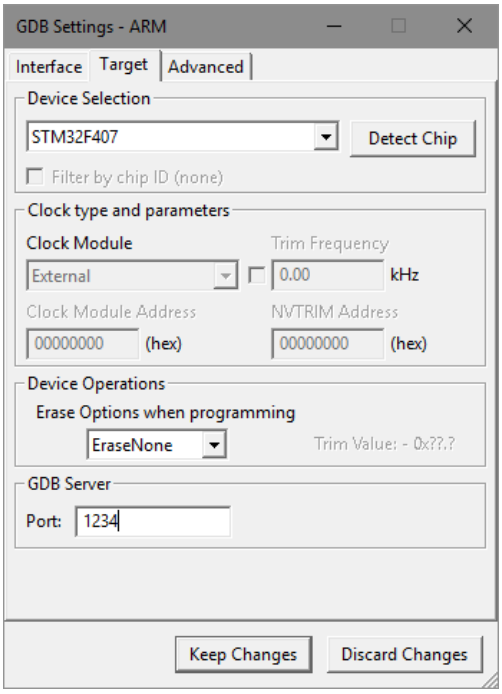
Kontrollera att GDB-server fått kontakt med USBDM.

Du ser det i rutan Select BDM.

Om det står No devices found här, kontrollera att USBDM är ansluten och att den gröna dioden lyser.

I Target Vdd control väljer du 3,3 Volt vilket innebär att man nu strömförsörjer *MD407* från USBDM via JTAG-gränssnittet.

Växla till fliken Target:

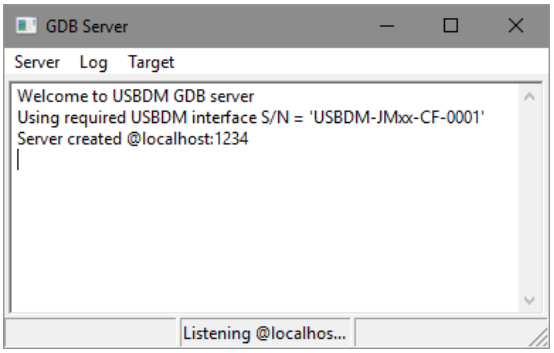


Kontrollera i Device Selection att USBDM ställs in för rätt microcontroller. Det ska stå STM32F407 här.

Om det står något annat, klicka på Detect Chip.

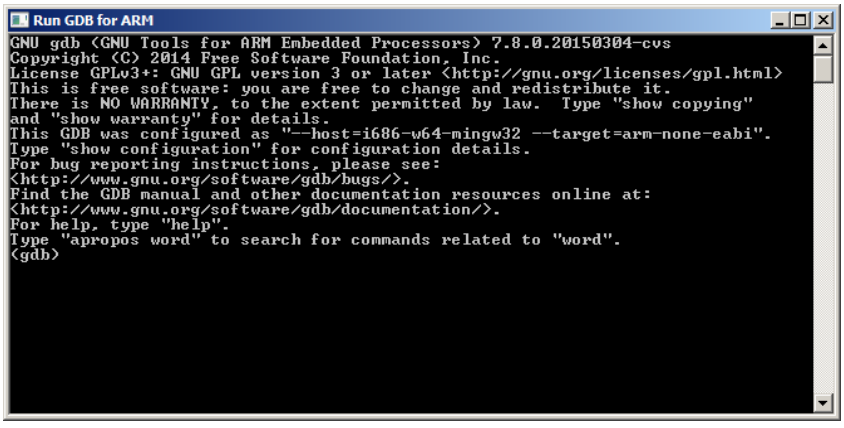
I GDB Server Port skriver du portnumret 1234.

Då detta är korrekt startar du *GDB-server* genom att klicka på Keep Changes.



GDB-Server väntar nu på att debuggern GDB ska starta kommunikationen via port 1234.

Starta *GDB för ARM*:



GDB identifierar sig nu med en rad utskrifter, slutligen skrivs GDB:s prompter till terminalen:

```
(gdb)
```

Det kan vara praktiskt att sätta aktuellt bibliotek, det gör du med cd-kommandot.

```
(gdb) cd <komplett sökväg>↵
```

Ange nu den fil du vill arbeta med med *file*-kommandot:

```
(gdb) file gdbtest1.elf↵
```

GDB läser in objektfilen med binär kod och symbolinformationen.

För att koppla ihop GDB med vår GDB-server, ger du kommandot:

```
(gdb)  target extended-remote localhost:1234↵
```

GDB gör nu RESET på *MD407*, däremot startas inte den inbyggda debuggern. Vilken adress visar *GDB*? Vad kan detta vara för adress?

Med load-kommandot, laddas nu kod och data till MD407:

```
(gdb) load↵
```

Med *list*-kommandot kan du studera programmets källtext, exempelvis med

```
(gdb) list 1,10↵
```

Ett annat användbart kommando är `info`, för att studera processorns registerinnehåll:

```
(gdb) info registers↵
```

Instruktioner för programexekvering

ge kommandot **`stepi`**↵(step instruction)

GDB låter *MD407* utföra en assemblerinstruktion och disassemblerar nästa. Vilken instruktion står nu i tur att utföras?

stega ytterligare 5 instruktioner med kommandot:

```
(gdb) stepi 5↵
```

Upprepa föregående kommando (`stepi 5`) genom att bara trycka ↵

Undersök värdet i `R0` med kommandot:

```
(gdb) info register r0↵
```

Laborationsuppgift 1.5:

- Starta nu om programmet genom att ge load-kommandot på nytt.
- Ge kommandot **`stepi 100`**↵observera dioderna på *USBDM*. Den gröna dioden blinkar då *USBDM* är upptagen.
- Vänta tills kommandot utförts, vad innehåller nu `R0`?

Laborationsuppgift 1.6:

Prova nu programmet `gdbtest2.elf`, som utför in- och utmatning. Ge kommandon.

```
(gdb) file gdbtest2.elf↵
(gdb) load↵
```

Starta programmet:

```
(gdb) c↵
```

Portarna fungerar inte denna gången. Förklaringen ligger i att GPIO-portarna måste aktiveras, dvs. deras respektive klockor måste aktiveras, innan de kan användas. Detta, bland annat, görs av dbgARM då *MD407* startas på vanligt sätt från RESET. Då *GDB* startar *MD407* startas emellertid inte dbgARM och vi måste därför själva göra detta i vår initieringsrutin för att kunna felsöka med hjälp av *USBDM*.
(Se: RCC AHB1 peripheral clock register (RCC_AHB1ENR) på sidan 178 i STM32F407-Reference manual)

Utgå från `gdbtest2.asm`, spara denna som `gdbtest3.asm`. Aktivering av portarna ska göras omdelbart, dvs. före den sekvens som konfigurerar port D. Följande tilldelning, i C, utför detta:

```
/* starta klockor port D och E */
* ( (unsigned long *) 0x40023830) = 0x18;
```

Visa här den instruktionssekvens, i assembler, som krävs för att aktivera portar D och E.

```
@ gdbtest3.elf

start:
                                    @ aktivera portar D och E
LDR     R0,=0x55555555

LDR     R1,=0x40020C00

…
```

- Assemblera `gdbtest3.asm` och rätta eventuella fel.

Prova nu programmet `gdbtest3.elf`. Ge kommandon.

```
(gdb) file gdbtest3.elf↵
(gdb) load↵
```

Starta programmet:

```
(gdb) c↵
```

- Kontrollera att programmet nu fungerar som det ska..

CodeLite och GDB

I föregående laborationsuppgift har du sett hur man kan använda GDB för att testa program i MD407 utan att använda den inbyggda monitor/debuggern dbgARM. Under nästa laborationsuppgift ska du

nu se hur man kan koppla sammand utvecklingsmiljön (CodeLite) med GDB och testa program på källtextnivå.

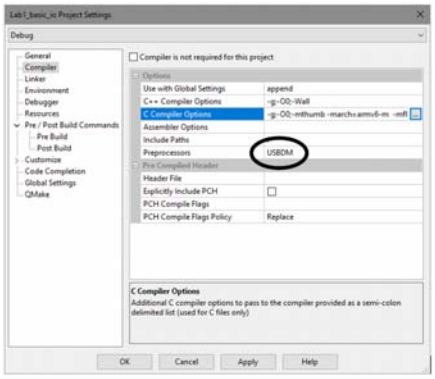
Laborationsuppgift 1.7:

Avsluta nu programmet *GDB för ARM* och starta i stället *CodeLite*.

Utgå från projekt `basic_io` (uppgift 3.3) i Arbetsboken

- Komplettera funktionen `app_init` med kod som aktiverar portar D och E

```
#ifdef USBDM
   /* starta klockor port D och E */
   * ( (unsigned long *) 0x40023830) = 0x18;
#endif
```



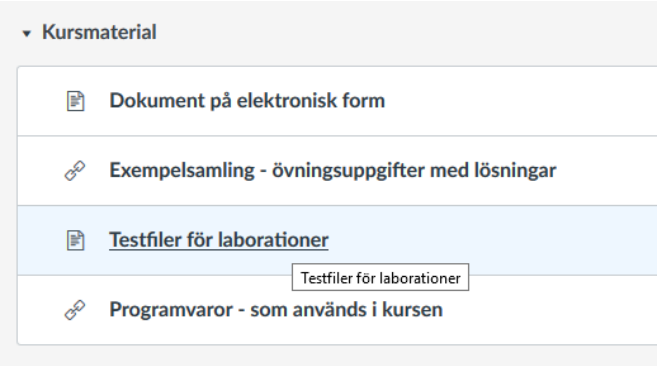
- Välj nu Project | Settings, fliken Compiler; på raden Preprocessors kan du definiera symboler som används vid kompilering av programmet. Definiera symbolen `USBDM` så att den kod du lagt till (villkorligt) också kompileras och ingår i ditt program:

- Kontrollera att *ARM GDB Server* är startad (*SimServer* ska INTE användas nu). Starta *GDB* från *CodeLite* på samma sätt som då du använde *SimServer* när du gjorde uppgiften i Arbetsboken.
- Kontrollera programmets funktion med GDB och USBDM i CodeLite. Exekvera programmet (rad för rad) och kontrollera funktionen.
- Använd därefter terminalfunktionen i *Eterm8* (jämför den inledande uppgiften i laboration 1), gör RESET på MD407, kontrollera att dbgARM startas, ladda ned programmet på nytt och starta det med kommandot go 20000000.

Användbar verktygslist för *GDB* i *CodeLite*:	
	Starta debugger – starta exekvering av programmet
	Stoppa debugger
	Avbryt exekverande program
	Återstarta (används ej med ARM)
	Visa aktuell rad, (om ingen exekveringspunkt visas i debuggern)
	Utför nästa sats i programmet
	Utför nästa rad i programmet
	Exekvera färdigt aktuell funktion

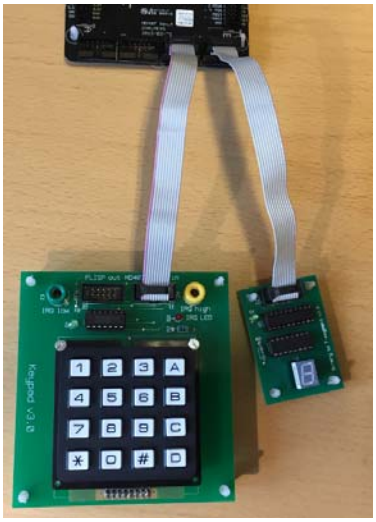
Laborationsuppgift 1.8:

Avslutningsvis ska du nu provköra ett program du själv ska konstruera inför nästa laboration.
Filen: `lab2_1.s19` hämtar du från kursens hemsida (Kursmaterial/Testfiler för laborationer):



Laborationsuppsättningen ska nu kopplas upp på samma sätt som inför laboration 2:

- Laborationskort anslutes till *MD407*:s portar enligt
  Keypad – PD8-15
  (Binär till) 7-segments display – PD0-7



- Laborationssystemet *MD407* ansluts till terminalfunktionen i *ETERM8* (jämför med laborationsuppgift 1.1).
- Ladda ned filen med hjälp av terminalfunktionen.
- Starta programmet med `go 20000000`.
- Tryck ned tangenter på keypaden, observera displayen, kontrollera att tangentnedtryckningen detekteras och att motsvarande tangent tänds på displayen.

Laboration nr 2 behandlar

Introduktion till maskinnära C,
synkronisering och tidmätning

Under denna laboration:

- praktiserar du kunskaper i maskinnära C-programmering
- får du också se hur ett program måste kunna synkroniseras med hårdvara, i detta fall i form av en ASCII-display.

Du kommer att undersöka och prova hårdvara:

- I/O-enheter *Keypad, 7-segments display* och en *LCD-ASCII display*.
- *SysTick*, en räknarkrets med hög noggrannhet.

Enklast förbereder du dig genom att:

1. Läs först översiktligt igenom detta laborationsavsnitt.
   Orientera dig om de läsanvisningar som ges
2. Arbeta igenom kapitel 4 och 5 (fram till "Drivrutin för grafisk display") i *Arbetsbok för MD407*.
   Det rekommenderas att du försöker hinna med att göra samtliga övningsuppgifter i kapitlen.
   Några av uppgifterna är dessutom direkt förberedande för laborationen.
3. Kontrollera att du utfört de uppgifter som ska vara lösta före laborationstillfället. Observera att detta i praktiken innebär att du måste lösa en sekvens av tidigare uppgifter i respektive kapitel.

Uppgifter i arbetsboken ger dig huvudsakliga lösningar på laborationsuppgifterna och ska vara utförda innan laborationen påbörjas. Observera att simulatorer sällan beter sig *exakt* på samma sätt som hårdvaran då det gäller realtidsegenskaper. Varje simulator är en avvägning mellan noggrannhet och exekveringstid (fördröjningar i simulatorn). Även om ditt program fungerar i simulatorn är det därför inte garanterat att det fungerar i hårdvara.

Följande laborationsuppgifter ur denna del av laborations-PM skall redovisas för en handledare för godkännande under laborationen. Då alla moment är godkända signeras också försättsbladets laboration 2.

Uppgift i arbetsbok	4.2	5.3	5.9	-
Laborations-uppgift	2.1	2.2	2.3	2.4
Sign.				

Läsanvisningar:

- **Arbetsbok kapitel 4**
- **Beskrivning av 7-segmentsdisplay**
- **Beskrivning av Keypad**

Anslutningar

Laborationsuppsättningen ska vara korrekt ansluten då du kommer till laborationsplatsen, men kontrollera för säkerhets skull anslutningarna.

- Laborationssystemet *MD407* ansluts till terminalfunktionen i *CodeLite* via anvisad USB-port.
- Laborationskort anslutes till MD407:s portar enligt
  Keypad – PD8-15
  (Binär till) 7-segments display – PD0-7

- Strömförsörjning sker nu från DC-jacket på *MD407*. Kontrollera att omkopplaren för val av strömförsörjning på *MD407* står i läge EXT.

Laborationsuppgift 2.1:

- Börja med att kontrollera att utrustningen är rätt konfigurerad och fungerar korrekt genom att ladda testfilen `lab2_1.s19` till *MD407* och testa programmets funktion. Tillkalla handledare om programmet *inte* beter sig som det ska.

Skriv en applikation som kontinuerligt läser av tangentbordet. Om någon tangent är nedtryckt, ska dess hexadecimala tangentkod skrivas till 7-segmentsdisplayen. Om ingen tangent är nedtryckt ska displayen släckas. Applikationen ska vara komplett med `startup`, `main`, initieringsfunktion och portdefinitioner, etc. Den ska dessutom utformas så att du enkelt kan testa den både med *USBDM* och med dbgARM:

```
void init_app( void )
{
#ifdef USBDM
```

Tänk på att klockan för port D måste aktiveras för att du ska kunna felsöka med USBDM (Jämför med laborationsuppgift 1.7).

```
   /* starta klockor port D och E */
   * ( (unsigned long *) 0x40023830) = 0x18;
#endif
   /* initiera port D */
}
/* main */
void main(void)
{
 unsigned char c;
    init_app();

   while( 1 )
   {
      c = keyb();
      out7seg( c );
   }
}
```

- Kontrollera programmets funktion med *GDB* och *USBDM* i *CodeLite*.
- Använd därefter terminalfunktionen i *CodeLite* (Output View | Eterm Console), ange den COM-port som *MD407* är ansluten till (jämför den inledande uppgiften i laboration 1), gör RESET på *MD407*, kontrollera att dbgARM startas, ladda ned programmet på nytt och starta det med kommandot `go 20000000`.

Anslutningar

Inför nästa uppgift ska ytterligare en laborationsmodul användas.

- Laborationskort anslutes till *MD407*:s portar enligt
  Bargraph – PE7-0

Låt anslutningarna från föregående uppgift vara kvar.

Laborationsuppgift 2.2:

- Börja med att kontrollera att utrustningen är rätt konfigurerad och fungerar korrekt genom att ladda testfilen `lab2_2.s19` till *MD407* och testa programmets funktion. Tillkalla handledare om programmet *inte* beter sig som det ska.

I denna uppgift ska du konstruera tre fördröjningsrutiner.

```
void  delay_250ns( void );
void  delay_mikro(unsigned int us);
void  delay_milli(unsigned int ms);
```

Rutinerna fördröjer de anropande funktionerna.

- `delay_250ns:` Fördröjer anropande funktion minst 500 ns.
- `delay_mikro`: Fördröjer anropande funktion minst `us` mikrosekunder.
- `delay_milli:` Fördröjer anropande funktion minst `ms` millisekunder.

Applikationen ska vara komplett med `startup`, `main` och portdefinitioner, etc. `SysTick`-funktionen ska användas. Den ska dessutom utformas så att du enkelt kan testa den både med *USBDM* och med dbgARM:

```
void init_app( void )
{
```

Även här måste klockan för port D aktiveras men dessutom måste processorns basklocka konfigureras för rätt arbetsfrekvens (168 MHz) för att realtidsfördröjningen ska bli rätt.Vi gör det här enklast genom att använda en funktion som finns inbyggd i dbgARM på adress 0x08000208. Följande kod kan då användas för att utföra de nödvändiga initieringarna:

```
#ifdef USBDM
  *((unsigned long *)0x40023830) = 0x18; /* starta klockor port D och E */
  __asm volatile( " LDR R0,=0x08000209\n BLX R0 \n") /* initiera PLL */;
#endif
   ... initiera portar
}
void main( void )
{
   init_app();

   while(1)
   {
      Bargraph = 0;
      delay_milli(500);
      Bargraph = 0xFF;
      delay_milli(500);
   }
}
```

Dvs. ljusdioderna ska tändas varje sekund och vara tända i en halv sekund.

- Kontrollera programmets funktion med *GDB* och *USBDM* i *CodeLite*.
- Använd därefter terminalfunktionen i *CodeLite*, gör RESET på *MD407*, kontrollera att dbgARM startas, ladda ned programmet på nytt och starta det med kommandot `go 20000000`.

Läsanvisning:

- **Arbetsbok kapitel 5**
- **Datablad ASCII-display, sidor 12-16, 25, 28,29,33,34**

Anslutningar

Inför nästa uppgift ska LCD-modulen kopplas till laborationssystemet. Eftersom LCD-modulen kan dra mer ström än den som levereras från USB-matningen måste vi nu använda extern strömförsörjning.

- Koppla bort Bargraph-modulen och 7-segments-displayen.
- Laborationskort anslutes till *MD407*:s portar enligt
  LCD-modul DATA – PE8-15
  LCD-modul CTRL – PE0-7

Laborationsuppgift 2.3:

- Börja med att kontrollera att utrustningen är rätt konfigurerad och fungerar korrekt genom att ladda testfilen `lab2_3.s19` till *MD407* och testa programmets funktion. Tillkalla handledare om programmet *inte* beter sig som det ska.

Skriv en applikation som som skriver text till ASCII-displayen.

Applikationen ska vara komplett med `startup`, `main`, initieringsfunktion och portdefinitioner, etc. Den ska dessutom utformas så att du enkelt kan testa den både med *USBDM* och med dbgARM:

```
void init_app( void )
{
#ifdef USBDM
  *((unsigned long *)0x40023830) = 0x18; /* starta klockor port D och E */
  __asm volatile( " LDR R0,=0x08000209\n BLX R0 \n") /* initiera PLL */;
#endif
   ... initiera portar
}
int main(int argc, char **argv)
{
   char *s ;
   char test1[] = "Alfanumerisk ";
   char test2[] = "Display - test";

   init_app();
   ascii_init();
   ascii_gotoxy(1,1);
   s = test1;
   while( *s )
      ascii_write_char( *s++ );
   ascii_gotoxy(1,2);
   s = test2;
   while( *s )
      ascii_write_char( *s++ );
   return 0;
}
```

- Kontrollera programmets funktion med *GDB* och *USBDM* i *CodeLite*.
- Använd därefter terminalfunktionen i *CodeLite*, gör RESET på *MD407*, kontrollera att dbgARM startas, ladda ned programmet på nytt och starta det med kommandot go 20000000.

OBSERVERA: Tänk på att displayens bakgrundsbelysning kan behöva justeras för att texten ska synas på displayen.

Laborationsuppgift 2.4:

När vi stöter på fel i våra program behöver vi först kunna hitta och isolera felet innan vi kan åtgärda det. När vi skriver program i ett sammanhang där vi har möjlighet att skriva ut text på en skärm kan vi använda den för att markera att vi kommit till en viss punkt i programmet och/eller att skriva ut värdet på variabler etc. När vi arbetar maskinnära, utan ett operativsystem med den funktionalitet som ett sådant tillhandahåller, behöver vi använda andra metoder.

Om vi använder *GDB/USBDM* för att kommunicera med *MD407* har vi möjlighet att undersöka programmets tillstånd under körning. Brytpunkter erbjuder ett sätt att stoppa exekveringen där vi misstänker att ett fel kan ha uppstått och sedan stegvis fortsätta programmet till felet identifieras. Därtill finns möjligheten att inspektera och ändra variablers värde och minnet, disassemblering med mera (se *Arbetsbok för MD407*, avsnitt 3.3 *Test och avlusning med Codelite*).

Om programmet istället körs via *ETERM8* är möjligheterna att interaktivt undersöka programmet mindre. Den fil som laddas till *MD407* (.s19) innehåller ingen symbolinformation, vilket innebär att all kommunikation med hårdvaran sker på en lägre nivå. Här är man hänvisad till att använda den inbyggda debuggern (`dbgARM`). Man måste då själv identifiera symbolers minnesadresser för att kunna visa värden, sätta brytpunkter etc.

- Vi arbetar nu åter igen med filerna från laborationsuppgift 2.1. Då projektet byggs skapas också en fil med ändelsen `.dass` (*disassembly*) . Öppna denna fil med en textredigerare.
- Leta rätt på koden för huvudprogrammet `main`, identifiera adressen till anropet av funktionen `keyb`.
- Adressen till instruktionen omedelbart efter anropet är:_____
- Ladda ned programkoden (.s19) via *ETERM8*:s terminalfunktion. Sätt ut en brytpunkt på adressen omedelbart efter anropet till `keyb`. Starta därefter programmet.
- Programmet stoppas vid brytpunkten, vilket värde finns nu i register `R0`?

- Håll ned tangent 'A' på tangentbordet, starta på nytt programmet som omedelbart stoppas på nytt vid brytpunkten. Vilket värde finns nu i `R0`?

- Fortsätt och prova ytterligare några tangentkoder, övertyga dig om att rätt värden returneras från funktionen.
- Ta nu bort brytpunkten före anropet till `keyb` och sätt i stället ut en brytpunkt på adressen där anropet till funktionen `out7seg` sker.
- Vad är giltiga värden för `c` i `out7seg`? Hur hanterar funktionen ogiltiga värden? Det måste du testa på annat sätt än att använda returvärdet från `keyb`.
- Kontrollera att `out7seg` hanterar ogiltiga värden korrekt (släcker displayen). Ledning, parametern `c` skickas till `out7seg` i register `R0`.

Laboration nr 3 behandlar

Grafisk display och keypad

Under denna laboration:

- färdigställer du drivrutiner för en grafisk display.
- beskriver du dataobjekt (geometrier, dvs. utseende respektive förmågor som hastighet och rörelseriktning) som inkapslade datastrukturer i C.
- kommer du också att använda keypad-rutiner från förra laborationen för att styra ett objekts rörelser på den grafiska displayen.

Du kommer att undersöka och prova hårdvara:

- En *LCD grafisk display*.

Enklast förbereder du dig genom att:

1. Läs först översiktligt igenom detta laborationsavsnitt.
   Orientera dig om de läsanvisningar som ges
2. Arbeta igenom resterande delar av avsnitt 5 i *Arbetsbok för MD407* (från "Drivrutin för grafisk Display"). Det rekommenderas att du försöker hinna med att göra samtliga övningsuppgifter i kapitlen. Några av uppgifterna är dessutom direkt förberedande för laborationen.
3. Kontrollera att du utfört de uppgifter som ska vara lösta före laborationstillfället. Observera att detta i praktiken innebär att du måste lösa en sekvens av tidigare uppgifter i respektive kapitel.

Följande uppgifter i arbetsboken utgör huvudsakliga lösningar på laborationsuppgifterna och ska vara utförda innan laborationen påbörjas. Observera att simulatorer sällan beter sig *exakt* på samma sätt som hårdvaran då det gäller realtidsegenskaper. Varje simulator är en avvägning mellan noggrannhet och exekveringstid (fördröjningar i simulatorn). Även om ditt program fungerar i simulatorn är det därför inte garanterat att det fungerar i hårdvara.

Följande laborationsuppgifter ur denna del av laborations-PM skall utföras/redovisas för en handledare för godkännande under laborationen. Då alla moment är godkända signeras också försättsbladets laboration 3.

Uppgift i arbetsbok	5.15	5.16	-
Laborations-uppgift	3.1	3.2	3.3
Sign.			

Läsanvisning:

- **Arbetsbok kapitel 5**
- **Datablad grafisk display, sidor 14-23**

Anslutningar

Inför denna uppgift ska LCD-modulen kopplas till laborationssystemet. Eftersom LCD-modulen kan dra mer ström än den som levereras från USB-matningen måste vi nu använda extern strömförsörjning.

- Laborationskort anslutes till MD407:s portar enligt
  LCD-modul DATA – PE8-15
  LCD-modul CTRL – PE0-7
- Strömförsörjning sker nu från DC-jacket på *MD407*. Kontrollera att omkopplaren för val av strömförsörjning står i läge EXT.

Laborationsuppgift 3.1:

I denna uppgift ska du skapa en applikation som matar ut ett horisontellt och ett vertikalt streck till displayen som då ska se ut som bilden till höger.

Efter att ha visats i en halv sekund, ska sedan strecken suddas ut så att displayen åter igen blir blank.

- Börja med att kontrollera att utrustningen är rätt konfigurerad och fungerar korrekt genom att ladda testfilen `lab3_1.s19` till *MD407* och testa programmets funktion. Tillkalla handledare om programmet *inte* beter sig som det ska.
- Kontrollera programmets funktion med *GDB* och *USBDM* i *CodeLite*.
- Använd därefter terminalfunktionen i *CodeLite*, gör RESET på *MD407*, kontrollera att dbgARM startas, ladda ned programmet på nytt och starta det med kommandot go 20000000.

Laborationsuppgift 3.2: Automatiskt PONG-spel

Skapa en applikation med en liten boll som rör sig autonomt över den grafiska displayen huvudsakligen i horisontalled. Bollen ska också ha en vertikal rörelsekomponent.

Anm: Tänk på att om uppdateringsfrekvens och hastighet är för höga kan det vara svårt att se bollen på displayen.

- Börja med att kontrollera att utrustningen är rätt konfigurerad och fungerar korrekt genom att ladda testfilen `lab3_2.s19` till *MD407* och testa programmets funktion. Tillkalla handledare om programmet *inte* beter sig som det ska.
- Kontrollera programmets funktion med *GDB* och *USBDM* i *CodeLite*.
- Använd därefter terminalfunktionen i *CodeLite*, gör RESET på *MD407*, kontrollera att dbgARM startas, ladda ned programmet på nytt och starta det med kommandot go 20000000.

Laborationsuppgift 3.3: Styr bollens bollens rörelseriktning

Lägg till keypad-rutiner från laboration 2 så att du i stället kan styra bollens rörelse över skärmen, ditt huvudprogram ska nu se ut som följer:

```
int main(int argc, char **argv)
{
   char  c;
   POBJECT p = &ball;
   init_app();
   graphic_initalize();
   graphic_clearScreen();

   while( 1 )
   {
      p->move( p );
      delay_milli(40);
      c = keyb();
      switch( c )
      {
         case 6:  p->set_speed( p, 2, 0); break;
         case 4:  p->set_speed( p, -2, 0); break;
         case 2:  p->set_speed( p, 0, -2); break;
         case 8:  p->set_speed( p, 0, 2); break;
      }
   }
}
```

- Börja med att kontrollera att utrustningen är rätt konfigurerad och fungerar korrekt genom att ladda testfilen `lab3_3.s19` till *MD407* och testa programmets funktion. Tillkalla handledare om programmet *inte* beter sig som det ska.
- Kontrollera programmets funktion med *GDB* och *USBDM* i *CodeLite*.
- Använd därefter terminalfunktionen i *CodeLite*, gör RESET på *MD407*, kontrollera att `dbgARM` startas, ladda ned programmet på nytt och starta det med kommandot go 20000000.
- I mån av tid kan du nu prova ytterligare interaktion och prova andra sätt att påverka objektet.

Laboration nr 4 behandlar

Undantagshantering

Under denna laboration:

- undersöker du processorns avbrottsmekanismer.
- konstruerar du en enkel applikation med *interna* avbrott.
- konstruerar du en enkel applikation med *externa* avbrott.

Du kommer att undersöka och prova hårdvara:

- ett laborationskort med flera avbrottskällor.

Enklast förbereder du dig genom att:

1. Läs först översiktligt igenom detta laborationsavsnitt.
   Orientera dig om, dvs. läs också översiktligt igenom, de hänvisningar som ges i avsnittet.
2. Arbeta igenom kapitel 6 i *Arbetsbok för MD407*. Det rekommenderas att du försöker hinna med att göra samtliga övningsuppgifter i kapitlen. Några av uppgifterna är dessutom direkt förberedande för laborationen.
3. Kontrollera att du utfört de uppgifter som ska vara lösta före laborationstillfället.

Följande uppgifter i arbetsboken utgör huvudsakliga lösningar på laborationsuppgifterna och ska vara utförda innan laborationen påbörjas. Observera att dessa uppgifter, i sin tur, kräver att du löst ytterligare uppgifter som föregått dessa. Observera också att simulatorer sällan beter sig *exakt* på samma sätt som hårdvaran då det gäller realtidsegenskaper. Även om ditt program fungerar i simulatorn är det därför inte garanterat att det fungerar i hårdvara.
Följande laborationsuppgifter ur denna del av laborations-PM skall utföras/redovisas för en handledare för godkännande under laborationen. Då momenten är godkända signeras också försättsbladets laboration 4.

Uppgift i arbetsbok	6.3	6.6	6.7	6.8
Laborations-uppgift	4.1	4.2	4.3	4.4
Sign.				

Läsanvisningar:

- **Arbetsboken, kapitel 6**
- **Beskrivning av "IRQ Flip Flop – Kort för avbrottsgenerering"**

Anslutningar

- Laborationskortet anslutes till *MD407*:s port enligt:
  Diodramp-modul - PD0-7
- Strömförsörjning sker från DC-jacket på *MD407*. Kontrollera att omkopplaren för val av strömförsörjning står i läge EXT.

Laborationsuppgift 4.1: Meddelandeskickning, internt avbrott med SysTick

I denna uppgift ska en fördröjningsrutin med avbrottsdriven SysTick-räknare konstrueras. Laborationsuppgiften baseras på en uppgift i arbetsboken där du kunnat testa ditt program med *SimServer*. För att kunna gå vidare och testa programmet i laborationssystemet med *USBDM* måste ytterligare initieringar göras (jämför med tidigare laboration). Dessa placeras lämpligen först i funktionen `init_app`.

- Börja med att kontrollera att utrustningen är rätt konfigurerad och fungerar korrekt genom att ladda testfilen `lab4_1.s19` till *MD407* och testa programmets funktion. Tillkalla handledare om programmet *inte* beter sig som det ska.
- Kontrollera ditt eget programs funktion.

Att tänka på då du testar program med avbrott:

För att kunna testa avbrott i program med *USBDM* krävs att du denna gång kompletterar `init_app` med följande initieringar som normalt görs av dbgARM:

```
#ifdef USBDM
   /* starta klockor port D och E */
   * ( (unsigned long *) 0x40023830) = 0x18;
   /* starta klockor för SYSCFG */
   * ((unsigned long *)0x40023844) |= 0x4000;
   /* Relokera vektortabellen */
   * ((unsigned long *)0xE000ED08) = 0x2001C000;
#endif
```

Debug-mekanismen hos *ST407* stänger av SysTick:s klocka vid instruktionsvis exekvering.

I en simulator kan du exekvera programmet instruktionsvis, generera ett avbrott och fortsätta instruktionsvis exekvering för att studera programmets uppträdande.

Med *GDB* och *USBDM* kan du *inte* stega dig igenom programmet instruktionsvis och förvänta dig att avbrottshantering sker. Detta beror på att *GDB* maskerar avbrott i dessa fall. I stället använder du brytpunkter i avbrottsrutinerna och startar exekvering av programmet från *GDB*.

Anslutningar

Inför nästa uppgift ska *FlipFlop*-modulen och en sifferdisplay kopplas till laborationssystemet.

- Laborationskortet anslutes till *MD407*:s port enligt:
  FlipFlop -modul (höger kontakt) – PE0-7
  Hexadecimal-sifferdisplay - PD0-7
- Strömförsörjning sker från DC-jacket på *MD407*. Kontrollera att omkopplaren för val av strömförsörjning står i läge EXT.

Laborationsuppgift 4.2: Laborationskort IRQ FlipFlop

Skriv en enkel applikation som använder PE3 hos MD407 som avbrottsingång och som registrerar varje avbrott, med en enkel räknare, och i huvudprogrammet skriver ut antalet avbrott till en visningsenhet. Applikationen ska vara komplett med `startup`, `main`, initieringsfunktioner och avbrottshantering:

```
void init_app( void ) { … }
void flipflop_interrupt_handler( void ) { … }
void main(void)
{
   init_app();
   while(1){
      utport = count;
   }
}
```

- Börja med att kontrollera att utrustningen är rätt konfigurerad och fungerar korrekt genom att ladda testfilen `lab4_2.s19` till *MD407* och testa programmets funktion. Tillkalla handledare om programmet *inte* beter sig som det ska.
- Kontrollera ditt eget programs funktion, tänk på att vipporna måste återställas manuellt mellan varje avbrott.

Laborationsuppgift 4.3: Selektiv avbrottskvittens

Utöka avbrottshanteringen från uppgift 4.2 så att:

- om IRQ0 aktiveras ökas räknaren med 1
- om IRQ1 aktiveras nollställs räknaren
- om IRQ2 (periodiska avbrott) aktiveras så tänds och släcks omväxlande samtliga dioder på diodrampen.
- alla avbrott återställs i avbrottsrutinen.

Prova speciellt om IRQ0 alltid ökar räknaren med 1 eller om en aktivering i stället gör att räknaren ökas med 2 eller mer. Här är det möjligt att kontaktstudsar hos strömbrytaren kan ge ett sådant resultat.

Laborationsuppgift 4.4: Selektiv avbrottskvittens

I den avslutande laborationsuppgiften ska du utnyttja flera IO-pinnar och ansluta tre olika avbrottskällor till tre olika IO-pinnar. Utgå från din lösning av uppgift 4.3. Avbrottsrutinen ska nu delas upp i tre olika avbrottsrutiner, en för varje avbrottskälla, IRQ0, IRQ1 och IRQ2. Funktionen ska vara den samma som tidigare, men du ska nu konfigurera så att:

- IRQ0 kopplas till EXTI0
- IRQ1 kopplas till EXTI1
- IRQ2 kopplas till EXTI2.

Laboration nr 5

Applikationsprogrammering

Under denna laboration:

- konstruerar du en komplett applikation med flera olika typer av IO-enheter.

Du väljer själv vad du vill göra för applikation men ett förslag kan vara ett enkelt datorspel.
Du kan söka inspiration till sådana spel på Internet, några klassiska varianter är:

- Pong
- Breakout
- Space invaders
- Tetris

Se exempelvis ”www.coolaspel.se”, där kan också du prova spelen. På kursens hemsida finns ytterligare "coola spel", på laddformat, som konstruerats av tidigare kursdeltagare. Du kan förstås hämta inspiration även från dessa!

Följande laborationsuppgifter ur denna del av laborations-PM skall redovisas för en handledare för godkännande under laborationen.

Laborations-uppgift	5.1
Sign.	

Laborationsuppgift 5.1:

Konstruera, implementera och demonstrera en komplett applikation.

Applikationen ska

- Använda såväl grafisk som alfanumerisk display
- Använda någon form av inmatningsenhet, *keypad* och eller *USART*