CHALMERS

Institutionen för data och informationsteknik.
Roger Johansson, Ulf Assarsson

2016-10-08

Maskinorienterad programmering

Laborationer

Detta laborations-PM innehåller anvisningar om förberedelser inför genomförande av laborationerna, såväl som de uppgifter du ska utföra och redovisa för, 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 steg för steg 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 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 avsnitt 1 och 2 i *Arbetsbok för MD407*
3. Kontrollera att du utfört de uppgifter som ska vara lösta före laborationstillfället.

Följande uppgift ur avsnittet ska vara utförd innan laborationen påbörjas. Du ska på begäran av laborationshandledare redovisa denna.

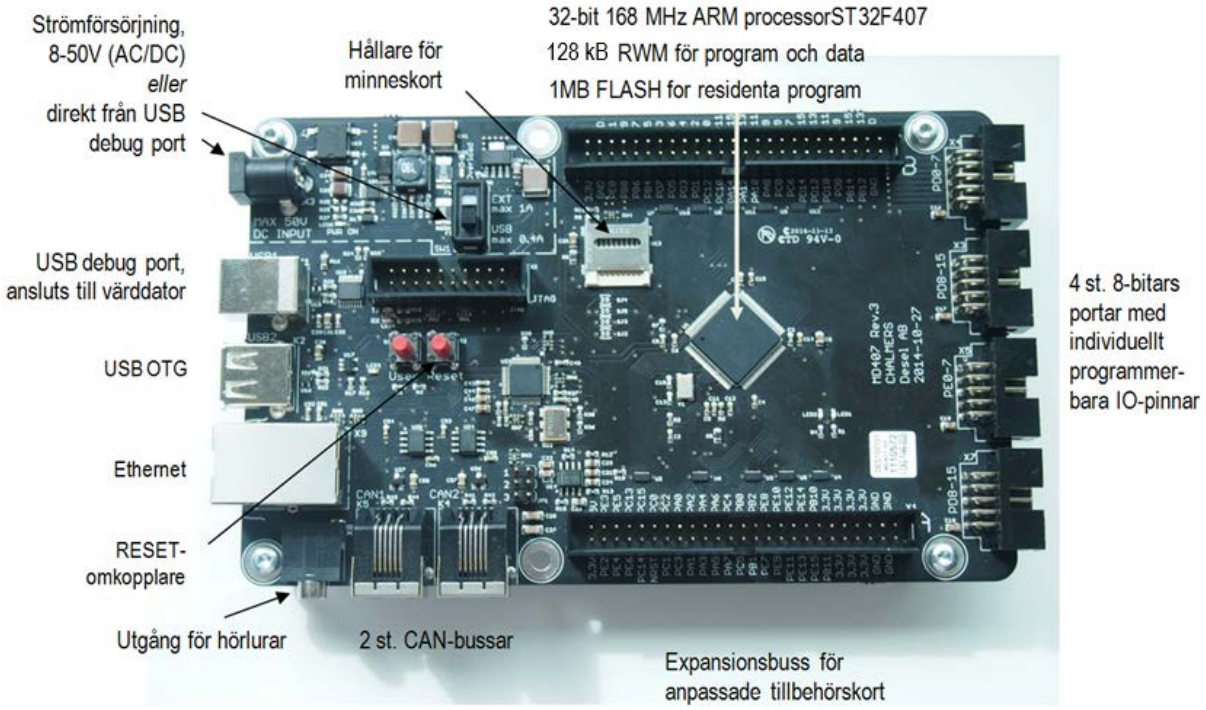
Uppg.	2	14	16
Sign.			

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.

Laborations-uppgift	1.2	1.4	1.6
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.

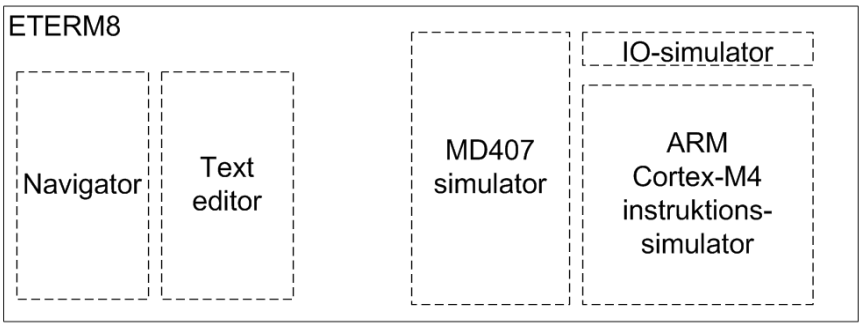
Utvecklingsmiljö

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

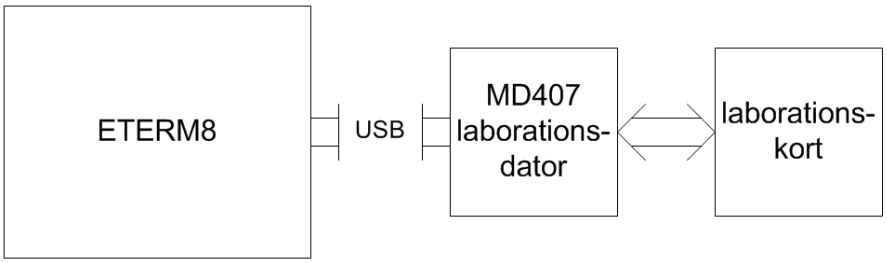
- *ETERM8* - Integrerad miljö för programutveckling i assemblerspråk
- *GDB* - Gnu Debugger för ARM-processorer
- *USBDM GDB-Server* - för JTAG-gränssnitt USBDM.

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 avsnitt 2.

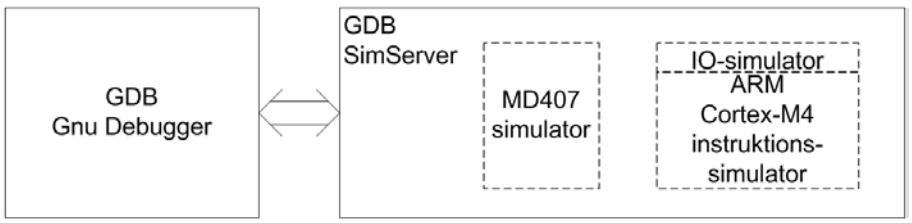
Simulatormiljö:



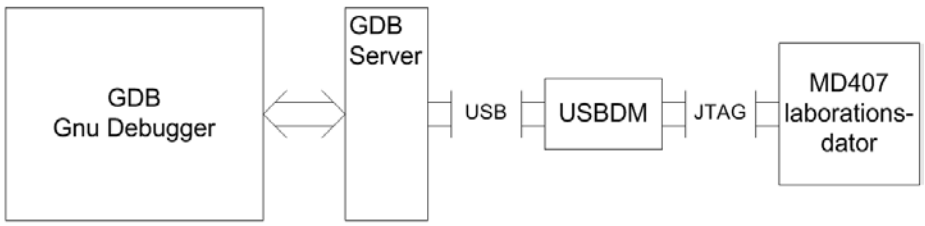
Laborationsmiljö:



Simulatormiljö:

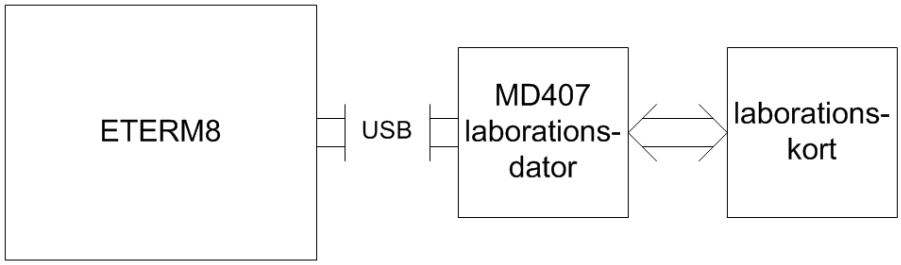


Laborationsmiljö:



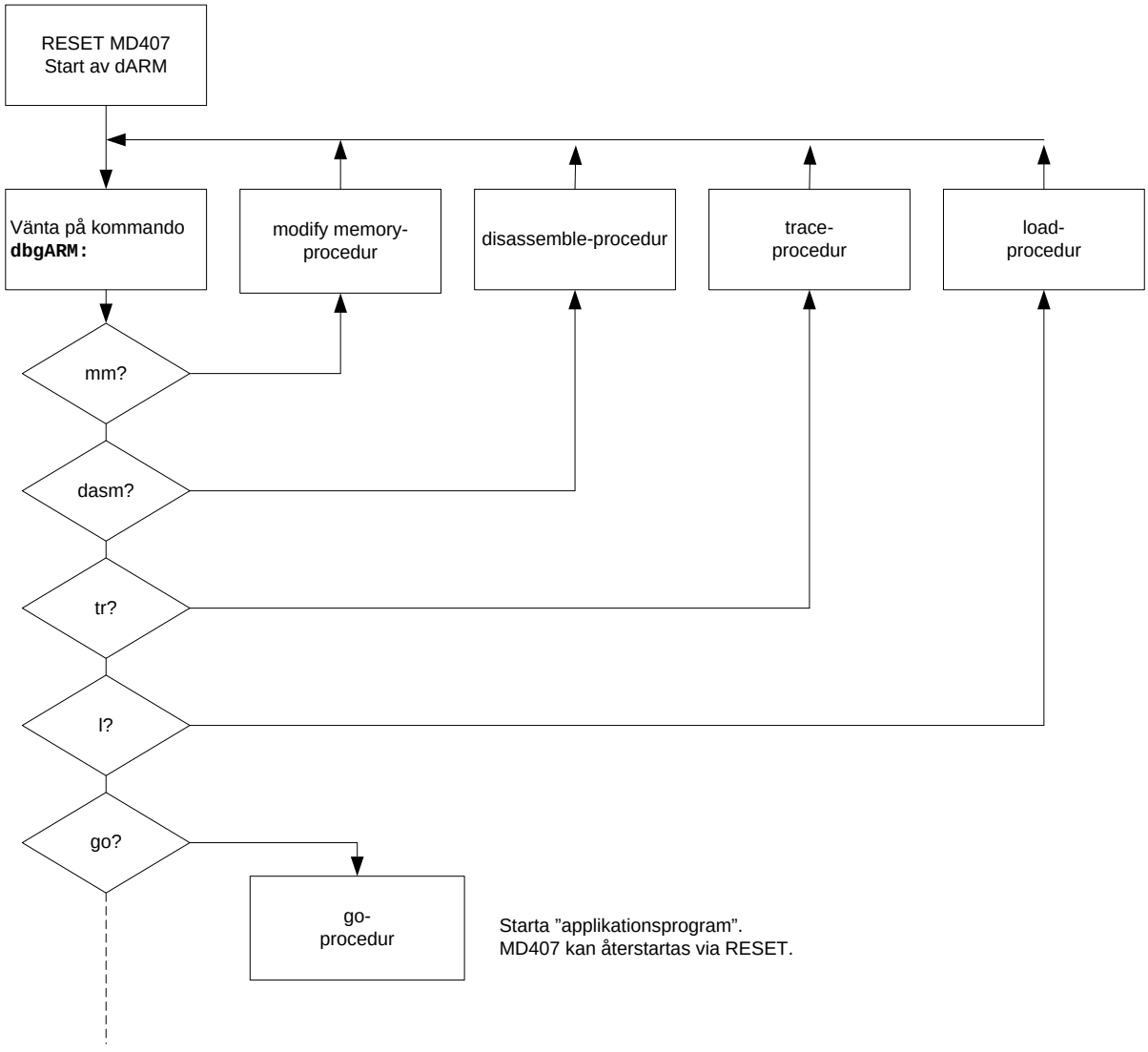
Monitor/debugger *dbgARM*

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*.

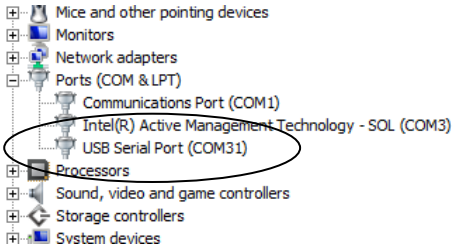
dbgARM ”lyssnar” hela tiden på kommandon från tangentbordet bortsett från när *MD407* är upptaget av något applikationsprogram. För att återgå till *dbgARM* i detta fall krävs RESET (omstart) av *MD407*. Huvudprogrammet i *dbgARM* är utformat som en enkel kommandotolk, där en rad olika kommandon accepteras.



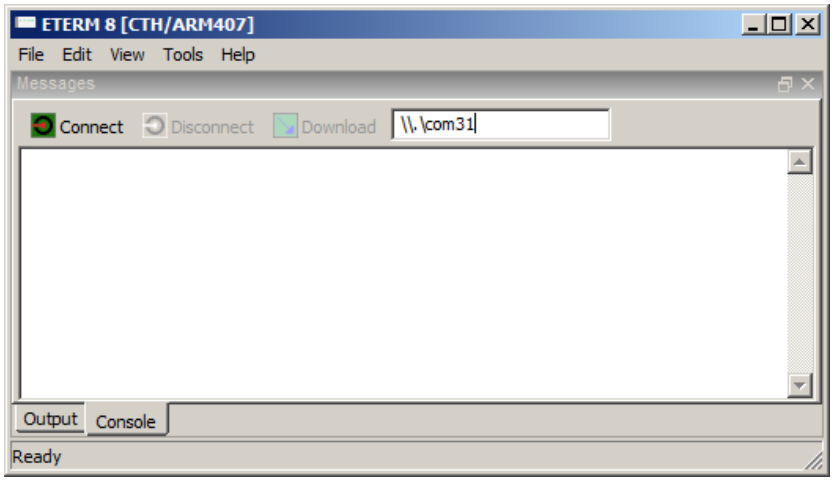
Laborationsuppgift 1.1:

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

- Kontrollera att laborationsdatorn är ansluten till en USB-port.
- Maximera sektionen Messages.
- Välj Console-fliken och ange den port laborationsdatorn är ansluten till, vilken port det är kan variera mellan olika miljöer, fråga laborations-handledaren om du är osäker, skriv porten i det lilla fönstret.
  I detta fallet är serieporten ansluten till COM31 vilket man också kan undersöka via Windows Device Manager.



OBSERVERA: Senare Windows kräver att portarna skrivs med den utökade syntaxen "`\\.\comX`".

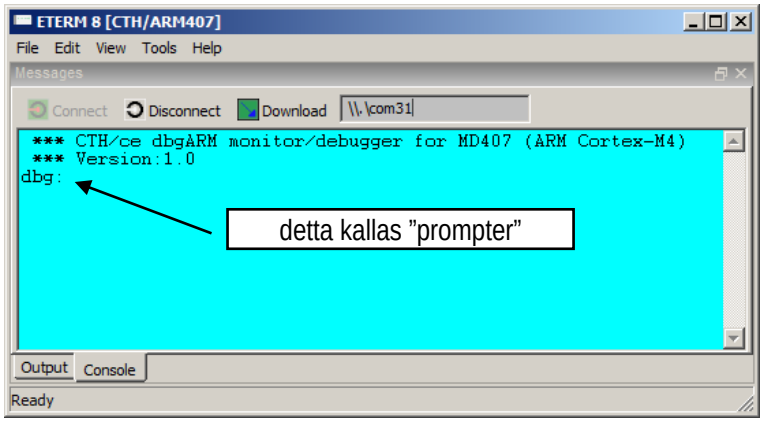


- Välj 'Connect' för att starta kommunikationsprogrammet:

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

- Tryck *RESET* på laborationssystemet*:*

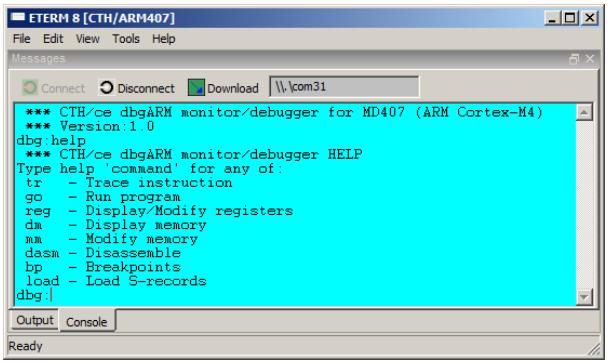
och kontrollera att *debuggern* identifierar sig genom att text skrivs till terminalfönstret. Konsollfönster bör då se ut ungefär som följande figur.
Om ingen text skrivs till skärmen eller om texten är oläslig, kontakta en handledare.



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`**↵

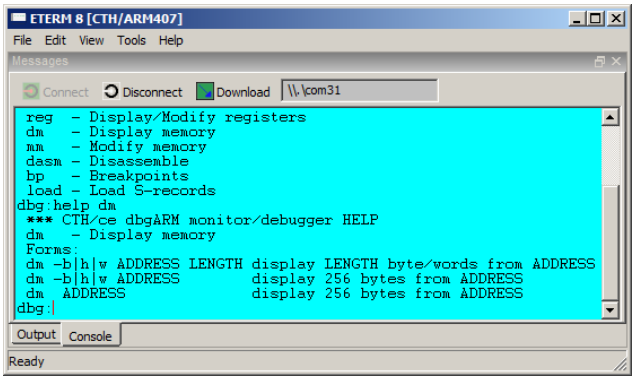


Under denna laboration får du tillfälle att prova *samtliga* dessa kommandon.

Kommandot ” help *kommando*”

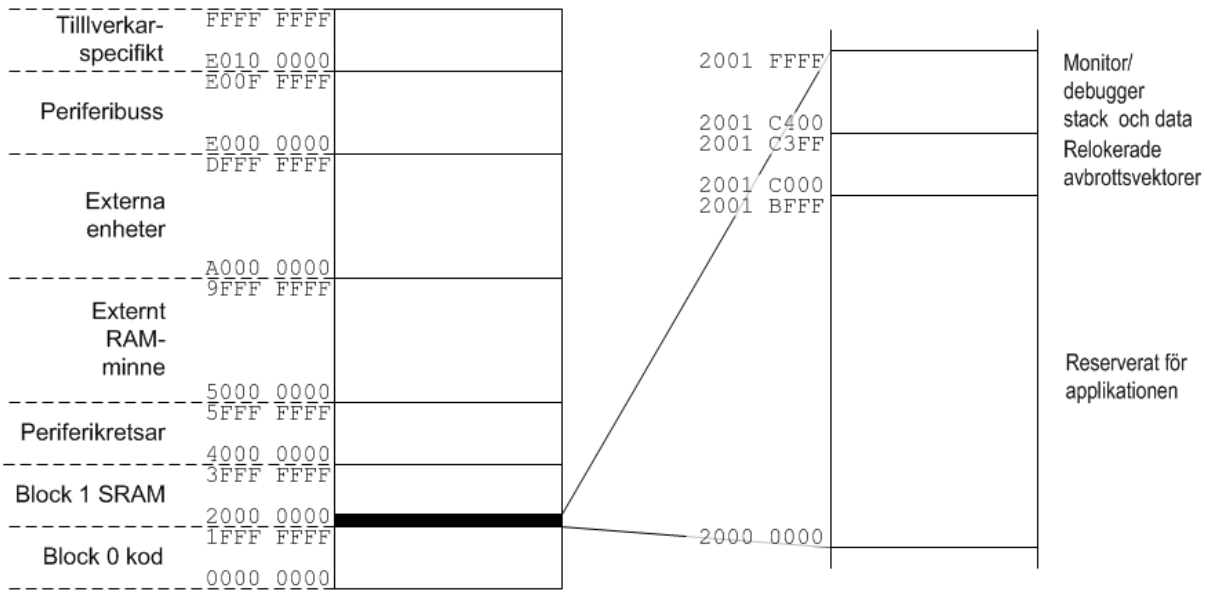
Du kan undersöka ett specifikt kommando genom att skriva: **`help`** ***`kommando`***↵ .

Prova exempelvis **`help dm`**↵



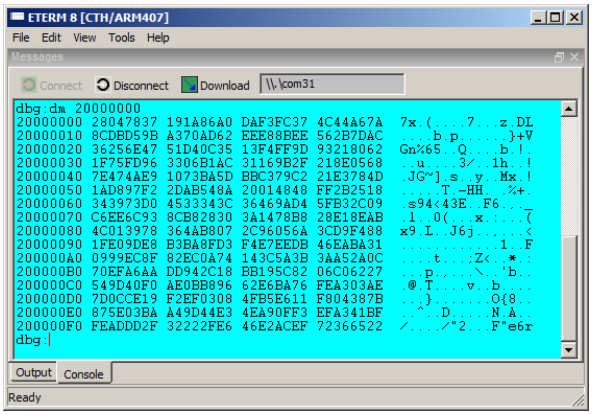
Kommandot ”display memory” (dm)

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).

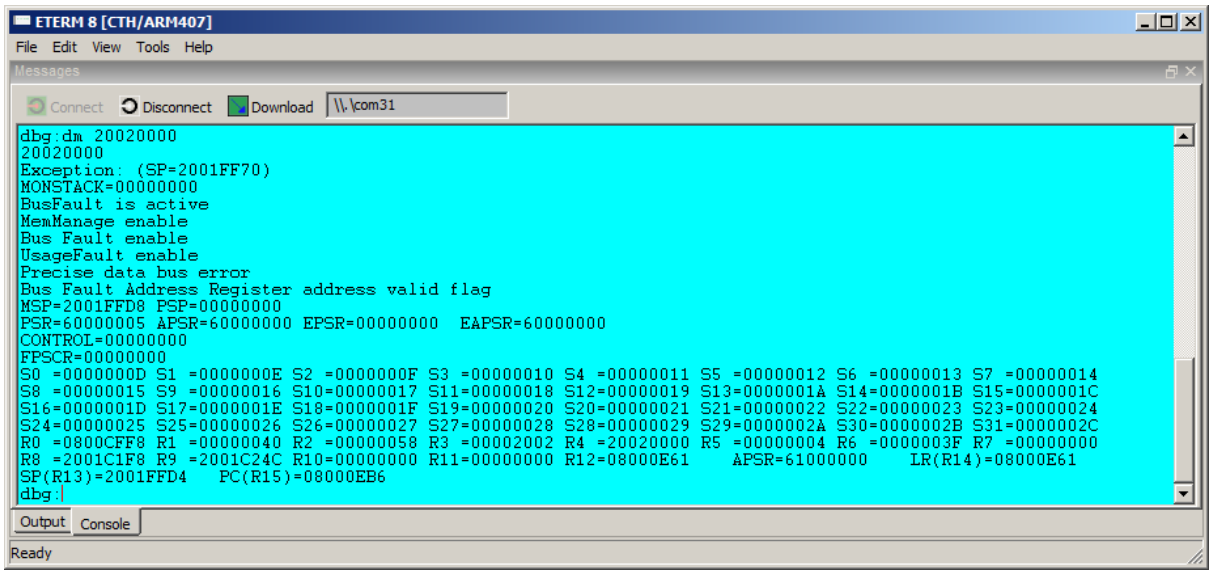
- Skriv: **dm 20000000↵** för att visa innehållet i minnet.



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 minneshanteringsenhet 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.

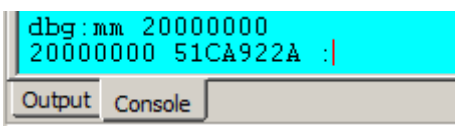


Kommandot ”modify memory” (mm)

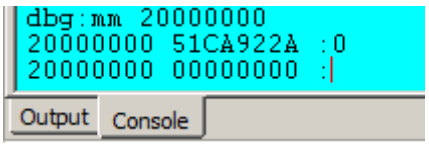
Vi ska nu använda kommandot **mm** för att försöka ändra minnesinnehåll på några olika ställen i *MD407:*s adressrum.

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

Har du samma minnesinnehåll som i figuren till höger?

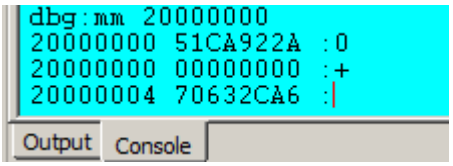


Skriv in ett nytt värde **0↵**, för att ändra innehållet på adressen. *dbgARM* svarar med att skriva ut raden på nytt med det nya värdet.



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

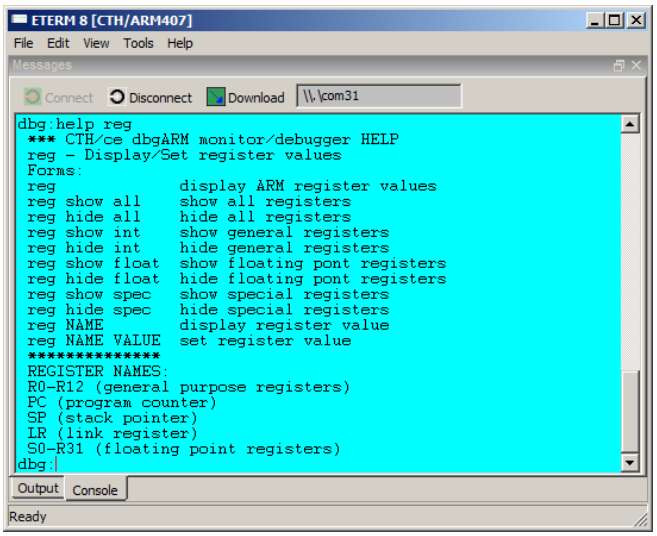
Tryck på – (minus) för att visa minnesinnehållet på föregående adress.



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`.

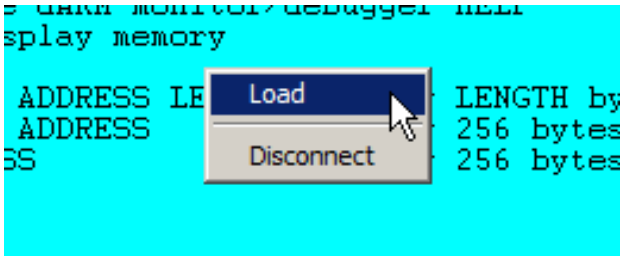
Prova några kommando för att visa generella och speciella register. Prova också med att ändra registerinnehåll.

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:

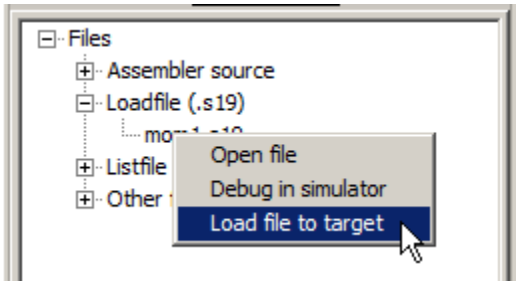
Klicka på Download i verktygslisten,.

eller högerklicka i det blå fönstret



Då du använder något av dessa alternativ öppnas en dialog som ger dig möjlighet att välja den fil du vill ladda till laborationsdatorn.

Ett tredje 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 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#]	;(0x20000010)	start
20000002	4904				
20000004	6008				
20000006	4904				
20000008	4902				
2000000A	4A03				
2000000C	6810				
2000001E	E7FC	B.N	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:

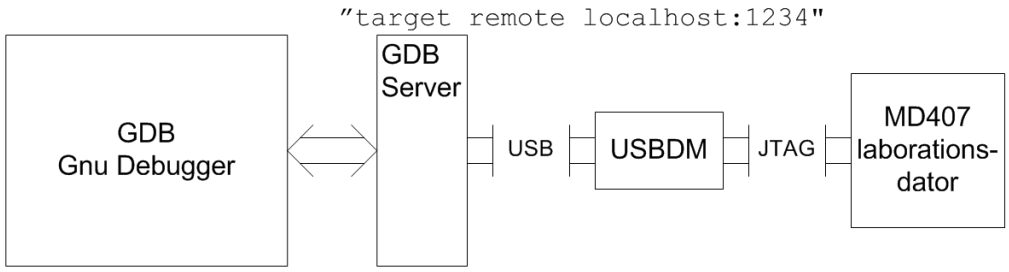
Starta därefter programmet med *go*-kommandot. Beskriv vad som då händer, vad ser du i konsollfönstret?

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: (Uppgift 15 i arbetsboken)

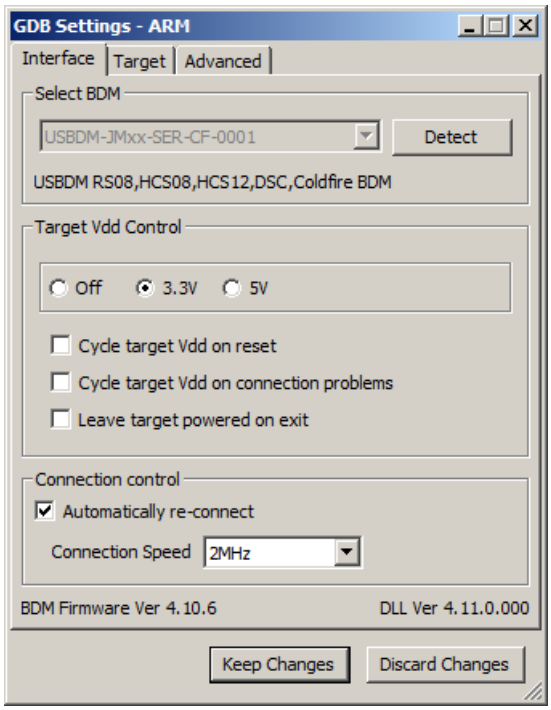
Redigera en källtext enligt följande, spara som fil `gdbtest1.asm`.

```
@  Laborationsuppgift
@  gdbtest1.asm
@

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

Assemblera filen, rätta eventuella fel.

Starta programmet ARM-GDB-Server:



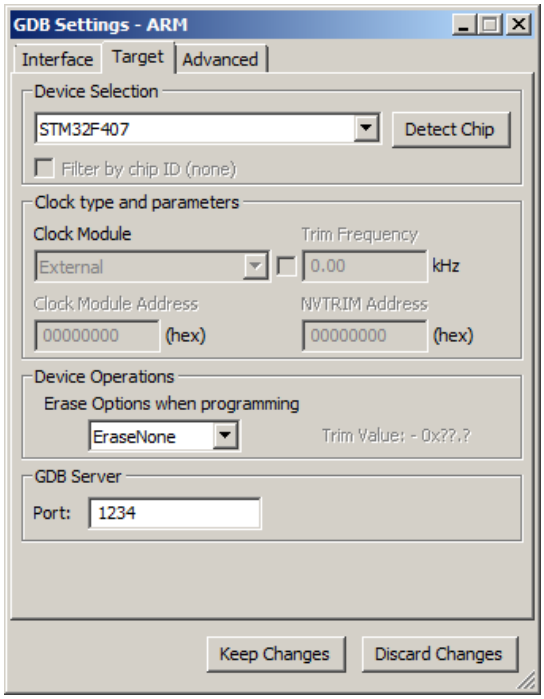
Kontrollera att GDB-server fått kontakt med USBDM.

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:

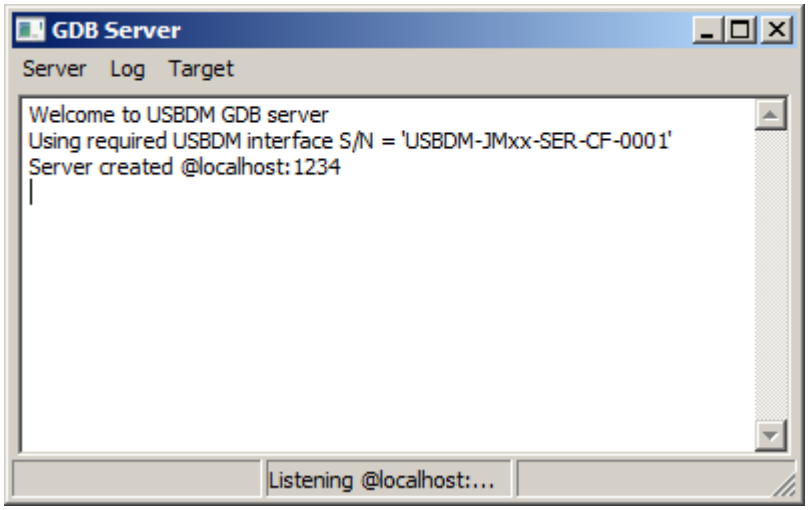


Kontrollera i Device Selection att USBDM ställs in för rätt microcontroller, det ska stå STM32F407 här.

Om det står något annat, klicka på Detect Chip.

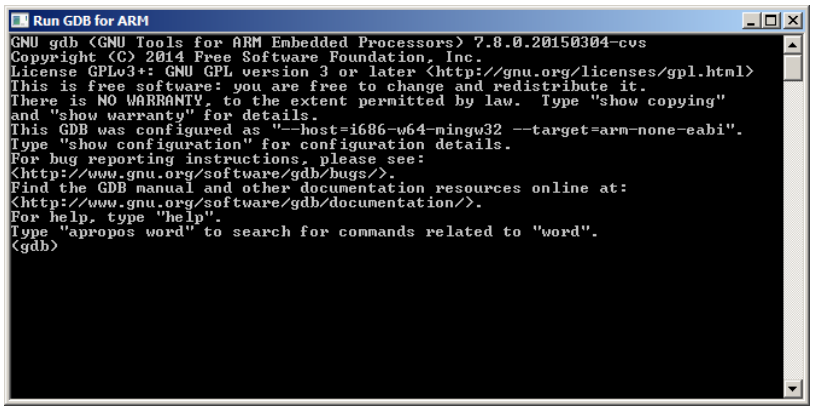
I GDB Server Port, skriver du portnumret 1234.

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 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 innan de kan användas.

Utgå från `gdbtest2.asm`, spara denna som `gdbtest3.asm`. Aktivering av portarna ska göras omdelbart, dvs. före den sekvens som konfigurerar port D.

Se:

RCC AHB1 peripheral clock register (RCC_AHB1ENR)

på sidan 178 i ”STM32F407-Reference manual”

Visa här den instruktionssekvens 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, 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..

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 avsnitt 3,4 och 5 (fram till "Drivrutin för grafisk display") i *Arbetsbok för MD407*
3. Kontrollera att du utfört de uppgifter som ska vara lösta före laborationstillfället.

Följande uppgifter i arbetsboken utgör hududsakliga lösningar på laborationsuppgifterna och ska vara utförda innan laborationen påbörjas. Du ska på begäran av laborationshandledare redovisa dessa. Observera 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.

Uppg.	19	21	27
Sign.			

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

Laborations-uppgift	2.1	2.2	2.3
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 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.

Laborationsuppgift 2.1:

Skriv en applikation som 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` och portdefinitioner, etc.

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:

I denna uppgift ska du konstruera tre fördröjningsrutiner, varav 2 med parametrar.

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

Fördröjningsrutinerna blockerar de anropande funktionerna.

- `delay_250ns:` Blockerar (fördröjer) anropande funktion minst 500 ns.
- `delay_mikro`: Blockerar (fördröjer) anropande funktion minst `us` mikrosekunder.
- `delay_milli:` Blockerar (fördröjer) anropande funktion minst `ms` millisekunder.

Testa fördröjningsrutinerna med följande programsekvens i `main`:

```
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.

Applikationen ska vara komplett med `startup`, `main` och portdefinitioner, etc. `SysTick`-funktionen ska användas.

Läsanvisning:

- **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.
- 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 2.3:

Anslut Display-modulen, CTRL till port E0-E7 och DATA till E8-E15. Följande testprogram ska skriva ut korrekt textsträng på ASCII-displayen:

```
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;
}
```

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

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").
3. Kontrollera att du utfört de uppgifter som ska vara lösta före laborationstillfället.

Följande uppgifter i arbetsboken utgör *hududsakliga* 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.

Uppg.	33	34
Sign.		

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

Laborations-uppgift	3.2	3.3
Sign.		

Läsanvisning:

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

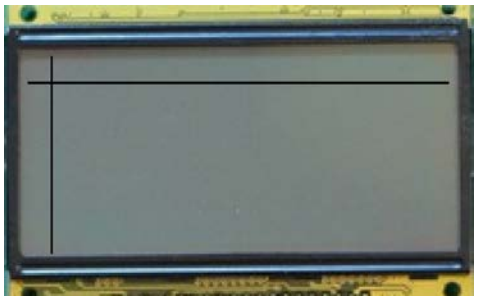
Anslutningar

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 på följande sätt:



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

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.

Laborationsuppgift 3.3: Styr bollens bollens rörelseriktning

Lägg till keypad-rutiner från laboration 2 så kan du i stället 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);  /* 25 frames/sec */
      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;

      }
   }
}
```

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 en enkel applikation med *interna* avbrott.
- konstruerar 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 avsnitt 6 i *Arbetsbok för MD407*
3. Kontrollera att du utfört de uppgifter som ska vara lösta före laborationstillfället.

Läsanvisningar:

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

Följande uppgifter i arbetsboken utgör huvudsakliga lösningar på laborationsuppgifterna och ska vara utförda innan laborationen påbörjas. Observera att dessa uppgifter, i sin tur, kan kräva att du löst betydligt fler uppgifter som föregått dessa.

Uppg.	36	39
Sign.		

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

Laborations-uppgift	4.1	4.3
Sign.		

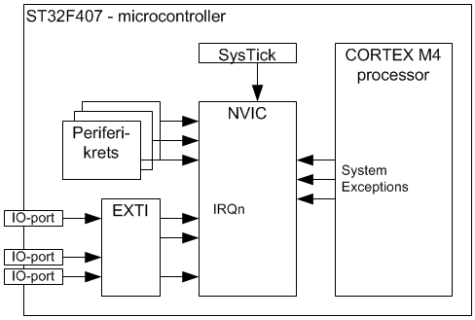
Laborationsuppgift 4.1: Meddelandeskickning, internt avbrott med SysTick

I denna uppgift ska en fördröjningsrutin med avbrottsdriven SysTick-räknare konstrueras.

Vi har tre programdelar

- Ett huvudprogram som omväxlande tänder och släcker en diodramp.
- En fördröjningsrutin `delay_250ns` som tillsammans med en avbrottshanterare åstadkommer en 250 ns fördröjning och därefter meddelar applikationen via ett *meddelande*.
- En fördröjningsfunktion `void delay(unsigned int count)` som blockerar den anropande funktionen `count` × 250 ns, och kommunicerar med avbrottsrutinen via *meddelande*.

Resten av denna laboration ägnas åt externa avbrott, en extern programvarumässigt kopplas till processorns avbrotssystem via en IO-port.

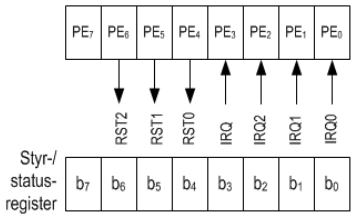


Laborationsuppgift 4.2: Laborationskort IRQ FlipFlop

Flank triggat avbrott med manuell återställning.

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:

Genom att undersöka statusregistret kan man bestämma vilken av de olika avbrottskällorna som aktiverats.



Laborationsuppgift 4.3: Selektiv avbrottskvittens

Utöka avbrottshanteringen från uppgift 4.2 så att:

- Om IRQ1 aktiveras ökas räknaren med 1
- Om IRQ2 aktiveras nollställs räknaren
- Om IRQ3 (periodiska avbrott) aktiveras så tänds och släcks omväxlande samtliga dioder på diodrampen.

Prova speciellt om IRQ1 alltid ökar räknaren med 1.

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 du också 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*