GÖTEBORGS MIKROVAROR

SimServer/ IO-simulator referenshandbok

Simulator för laborationsdatorer och kringenheter

Reviderad December 2024

Version 0.998

Denna referenshandbok startar med dokumentation av version:0.996.

- SimServer och IO-simulator utvecklas och förbättras fortlöpande. Nya funktioner, laborationsdatorer och IO-enheter läggs till efter hand.
- Samtliga moduler har ännu ej införts i referenshandboken, det kan därför finnas smärre avvikelser mellan program och dokumentation.
- En nyhet är ”FRISC”, en mycket enkel 16-bitars RISC-maskin, avsedd att komplettera eller ersätta FLIS-processorn i ”Grundläggande datorteknik”.
- En annan nyhet är en helt ny, RISC-V baserad laborationsdator MD307 avsedd att komplettera eller ersätta MD407 framför allt inom ”Maskinorienterad programmering”. Speciellt har laborationskorten PTB-110 (”Multi-IO”) och PTB-111 (ASCII- och TFT/Touch- displayer) utvecklats för MD307.

Innehåll

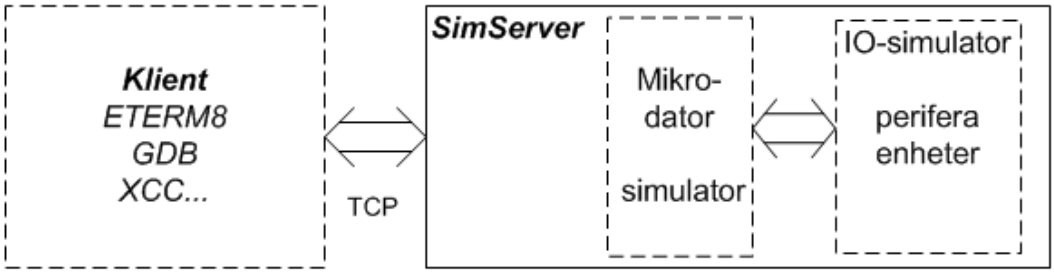
SimServer och IO-simulator

SimServer och IO-simulator är avsedd att användas framför allt i undervisning inom maskinorienterad programmering och systemprogrammering.

Programmet innehåller ett integrerat simulatorpaket och finns tillgängligt för såväl Windows- som Mac- och Linux- datorer.

SimServer kommunicerar med ett klientprogram med ett utvidgat *GDB-remote server*-protokoll via TCP.

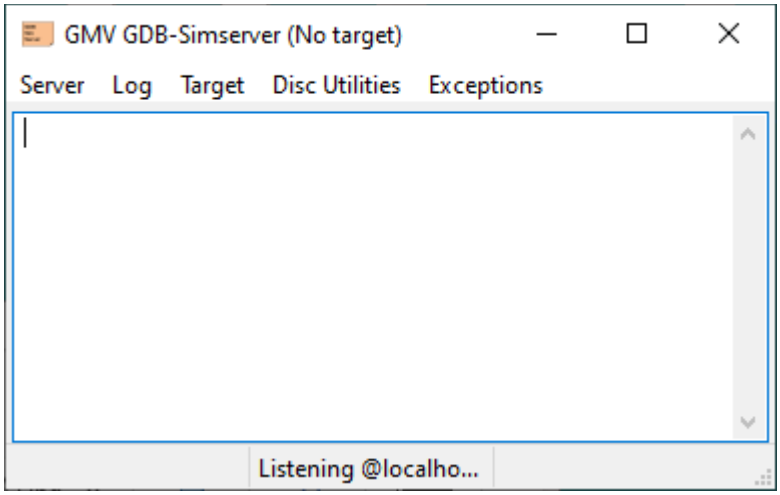
Den öppna arkitekturen och det flexibla protokollet gör det enkelt att koppla olika utvecklingssystem som klient till SimServer. Färdiga klientlösningar finns bland annat på GMV's hemsida och omfattar i dag klientprogram som ETERM8, Codelite och XCC (version3).



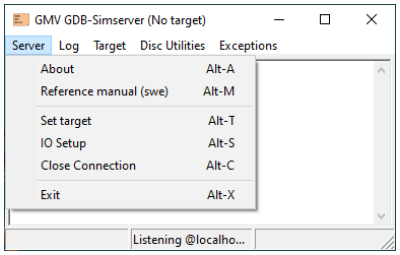
Funktioner

Simserver/IO-simulator innehåller integrerade simulatorer för en rad mikrodatorer som utvecklats för undervisningsändamål. I denna referenshandbok beskrivs funktionalitet hos dessa simulatorer.

Serverfunktionen kommunicerar via TCP med någon klient. Protokollet är ett utvidgat GDB remote-server protokoll.

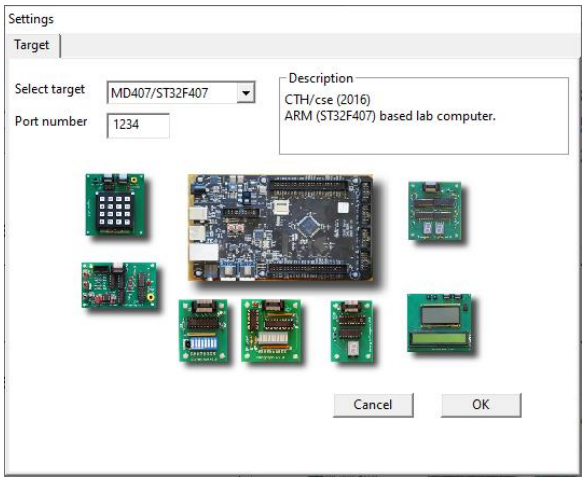


Server:



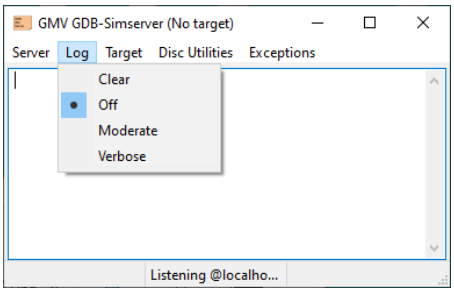
- **About**: Version och utgivningsdatum
- **Reference manual**: Öppna referensmanualen (denna) kräver Internetuppkoppling
- **Set Target**: Ange vilken måldator som ska simuleras och vilken TCP-port som ska användas, se nedan.
- **IO setup**: Koppla enheter till måldatorn, denna funktion beskrivs i separata dokument för de olika laborationsdatorerna.
- **Close Connection**: avbryt session med klient.
- **Exit**: Avsluta Simserver

Set Target:



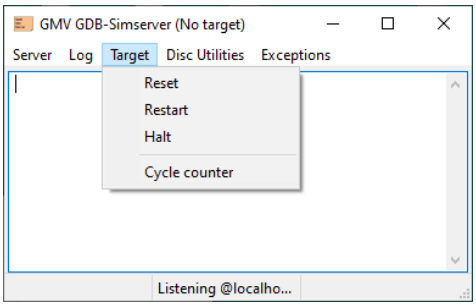
Select Target: Välj en måldator.
Port number: Ange den TCP-port Simserver ska använda för att kommunicera med klienten.
En godtycklig port kan användas men många är redan upptagna av systemets nätverkstjänster. Normalt används här port 1234 som inte bör skapa någon konflikt.
Ska man köra flera samtidiga instanser av Simserver måste dock dessa ha unika portar.
Du kan läsa mer om hur du använder TCP-portar här:
https://www.sciencedirect.com/topics/computer-science/registered-port

Menyval **Log** används för diagnostiska ändamål. Här skriver Simserver meddelande om aktiviteter såväl som felmeddelanden:



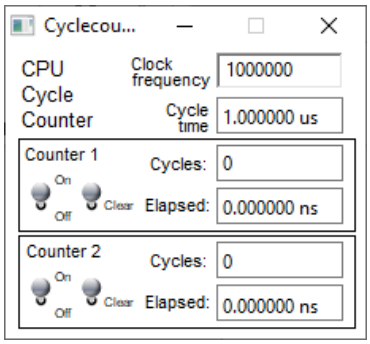
- **Clear**: Radera alla meddelande.
- **Off**: Endast meddelande om kommunikation med klient skrivs ut.
- **Moderate**: Även meddelanden om simulatorkommandon skrivs ut.
- **Verbose**: Alla meddelanden skrivs ut.

Menyval **Target** används för operationer kring den valda måldatorns programexekvering:



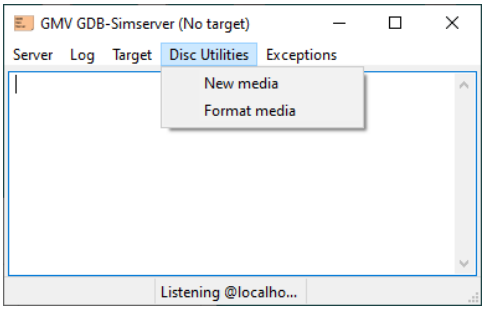
- **Reset**: Genomför en RESET-sekvens i måldatorn.
- **Restart**: Förbereder återstart av det laddade programmet.
- **Halt**: Avbryter ett exekverande program.
- **Cycle counter**: Cykelräknare för realtidssimuleringar.

Cycle counter: Cykelräknare för realtidssimuleringar.
Funktionen innehåller två cykelräknare som kan aktiveras/deaktivera (On/Off) och nollställas (Clear) oberoende av varandra.
En aktiverad räknare registrerar måldatorns antal utförda klockcykler vid programexekvering.



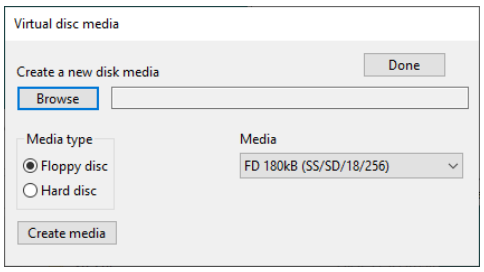
Klockfrekvensen kan ändras genom att ett nytt värde (med format `double`) matas in i fönstret (Clock frequency) följt av<Enter>.
Om värdet 0 matas in återställs klockfrekvensen till måldatorns standardinställning.

Menyval **Disc Utilities** används för skapa och formatera olika typer av virtuella media (skivminne) för skivminnesenheter (Floppy-/Hard discs)



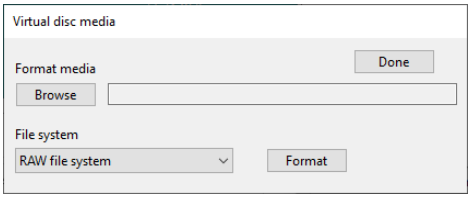
- **New media**: Skapa en ny virtuell disk.
- **Format media**: Förbered disk för användning.

New Media: används för skapa nytt skivminne som virtuellt media.



- **Browse**: Ange sökväg och namn på den fil som motsvarar disken.
- Välj **Floppy disc** för flexskiva eller **Hard disc** för hårddisk.
- **Media**: välj bland olika diskgeometrier (diskstorlek). Listan innehåller klassiska storlekar på disketter och hårddiskar.
- **Create media**, skapa skivminnesenheten.
- **Done**, avsluta och återgå.

Format Media: används för att formatera ett skivminne för skivminnesenheter (Floppy-/Hard discs)



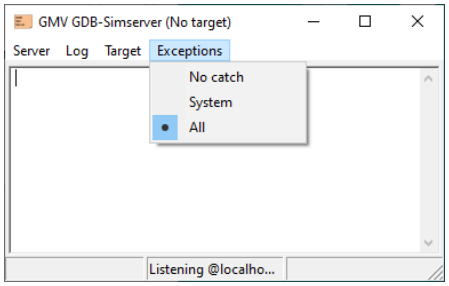
- **Browse**: Ange sökväg och namn på den fil som motsvarar disken.
- File system: förbered skivminnet för användning av något filsystem. Alternativet ”RAW” innebär inget specifikt filsystem.
- **Format**, formatera skivminnet.
- **Done**, avsluta och återgå.

Menyval **Exceptions** används då man vill studera undantagshantering med hjälp av simulatorn. Då ett undantag fångas, avbryts programexekveringen och ett meddelande om det inträffade visas. Det finns kortfattat två olika situationer då man behöver fånga undantagshantering:

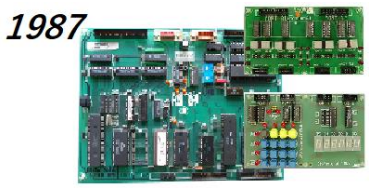
- Man förväntar sig inget undantag, men det händer ändå. Här kan funktionen hjälpa till att hitta fel i program som generar undantagsfel.
- Man förväntar sig undantag, vanligtvis i form av avbrott, och vill följa programmets undantagshantering (avbrottsrutiner). Här kontrollerar man att avbrottsrutinen faktiskt startas då undantaget (avbrottet) sker.

För vissa måldatorer skiljer simulatorn mellan undantagstyperna. Alternativet System används typiskt då programmet innefattar avbrottshantering som ska utföras men samtidigt genererar något fault som ska fångas och åtgärdas.

No catch: Undantagshantering utförs på samma sätt som i måldatorn.
System: Endast systemfel fångas.
All:Såväl systemfel som programmerad avbrottshantering fångas.



Laborationsdator MD09



MD09, "MikroDator 09" (1987) var den första av microlf's enkortsdatorer speciellt utvecklad för undervisningsändamål.
Förlagan till MD09 var ett industridatorsystem med en helt annorlunda uppbyggnad, Europa-kort i 19" racksystem, vilket inte var ett dugg ovanligt i mitten av 1980-talet. Kraven på MD09 var omfattande, den skulle integrera samtliga utmärkande funktioner hos detta industri-datorsystem på ett enda kort. Priset skulle dock maximalt bli en tiondel av förlagan! Dessutom skulle MD09 ge ytterligare funktionaliteter vilka kom att realiseras i form av speciella laborationskort (ML1, ML2 och ML3).

Minnesanvändning:

Typ	startadress	slutadress
RWM	0	0xDFFF
ROM	0xE200	0xFFFF
IO	0xE000	0xE1FF
Port: ACIA6850	0xE024	0xE025
Port: ACIA6850	0xE028	0xE029
Port: PIA6821	0xE048	0xE04B
Port: PIA6821	0xE050	0xE053
Port: FDC	0xE060	0xE06B
Port: PTM6840	0xE0E0	0xE0E7

SimServer implementerar följande funktioner hos MD09:

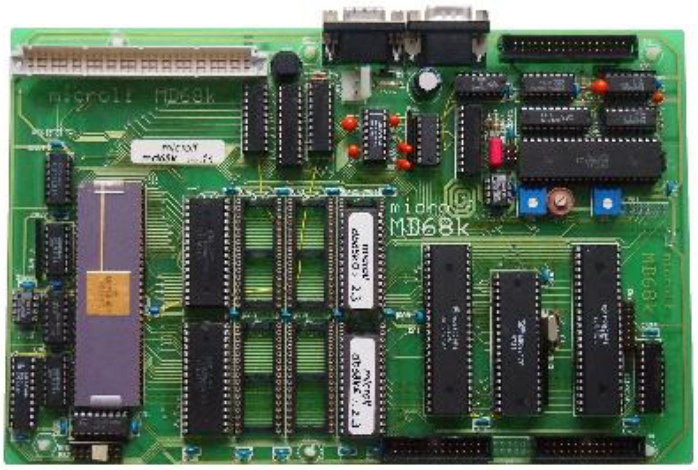
- Anslutning av parallella enheter sker via kretsarna PIA6821.
- Seriekommunikationskretsarna, ACIA6850, kan anslutas till IO-simulator 'Console'.
- Flexskiveenheten ansluts via ett förenklat simulatorgränssnitt (FDC).
- Undantagshantering (*swi och interrupts*) har implementerats i simulatorn för PTM6840, ACIA6850 och FDC.

Anm:
PTM6840: Endast kontinuerlig mod, 16 bitars nedräknare är implementerad.
PIA6821: CA1/2, CB1/2 (avbrott) är ej implementerade.
FDC: Ej kompatibel med WD2797, men möjliggör att simulatorn kan användas med de diskoperativsystem som anpassades för MD09.

Laborationsdator MD68k

MD68k introducerades 1990 som en efterföljare till MD09. Det var en, för denna tid, anmärkningsvärd laborationsdator.
Bestyckad med MC68000, dubbla seriekommunikationskretsar, parallellportar, option för 1,5 MByte primärminne och styrkretsar för flexskiveenhet. Bakåtkompatibiliteten med MD09 var fullständig, dvs allt man kunde göra med MD09 kunde man också göra med MD68k, men naturligtvis, mycket mer därtill.

Laborationsdatorn MD68k



Minnesanvändning:

Typ	startadress	slutadress
RWM	0	0x3FFFF
ROM	0x300000	0x30FFFF
IO	0x3E0000	0x41FFFF
Port: PIT68230	0x3E0001	0x3E000F
Port: PIT68230	0x3E0401	0x3E040F
Port: DUART68681	0x3E0801	0x3E081F
Port: ML18-ML5/ML2	0x402001	0x402003
Port: ML18-ML5/ML3	0x402005	0x402007
Port: ML18-ML15/ML2	0x413801	0x413805
Port: ML18-ML15/ML3	0x413805	0x413807

SimServer implementerar följande funktioner hos MD68k:

- Anslutning av parallella enheter sker via kretsarna PIT68230.
- Laborationskort med ”piggyback” ansluts via anpassning ML18.
- Seriekommunikationskrets, DUART68681, kan anslutas till IO-simulator ’Console’.

Anm: IO-adressering av byte-register görs på udda adress.
Undantagshantering är avstängd tills fullständiga tester genomförts.

Laborationsdator MC68

MC68 introducerades 1994 och kom att bli en av de mest spridda och använda laborationsdatorerna i Sverige. Konstruktionen av MC68 baserades delvis på föregångarna MD09 och MD68k, men med MC68 introducerades också ett helt nytt koncept för anslutning av olika laborationskort, *piggy-back* som tillåter att laborationskort placeras ovanpå (eller under) MC68 och samtidigt inkluderar alla nödvändiga elektriska anslutningar. MC68 har också en direkt anslutning till multifunktionskortet ML4.

Laborationsdatorn MC68

Minnesanvändning:

Typ	startadress	slutadress
RWM	0	0x7FFFF
ROM	0x100000	0x11FFFF
IO	0x80000	0x8FFFF
Port: ML5/ML2	0x8C000	0x8C001
Port: ML5/ML3	0x8C002	0x8C003
Port: ML15/ML2	0x89C00	0x89C01
Port: ML15/ML3	0x89C02	0x89C03
Port: SIM40/PORTA	0xFFFFF010	0xFFFFF017
Port: SIM40/PORTB	0xFFFFF018	0xFFFFF01F
Port: SIM40/DUART68681	0xFFFFF700	0xFFFFF721

SimServer implementerar följande funktioner hos MC68:

- Anslutning av parallella enheter kan göras via SIM40's portmoduler.
- Laborationskort med "piggyback" kan anslutas.
- Den inbyggda seriekommunikationskretsen, DUART68681, kan anslutas till IO-simulator 'Console'.

Anm: Undantagshantering är avstängd tills fullständiga tester genomförts.

Laborationsdator MC11

MC11 (1998), en kompakt och flexibel enkortsdator baserad på en MC68HC11 mikro-kontroller från Motorola. MC11 hade såväl piggy-back anslutning till laborationskort i ML5x-serien men dessutom en direkt anslutning till ML4.
Externa minnet bestod av såväl statiskt RAM (RWM) som FLASH-ROM.

Minnesanvändning:

Typ	startadress	slutadress
RWM	0	0xFF
RWM	0x2000	0x3FFF
ROM	0x4000	0xFFFF
IO Externt	0x1000	0x1FFF
Port: OUTPORT	0x1400	0x1400
Port: INPORT	0x1600	0x1600
Port: ML5/ML2	0x1C00	0x1C01
Port: ML5/ML3	0x1C02	0x1C03
Port: ML15/ML2	0x19C0	0x19C1
Port: ML15/ML3	0x19C2	0x19C3
IO Internt:		
Port: Serie (SCI0)	0x102B	0x102F
Port: Timer (RTI)	0x100B	0x1027

SimServer implementerar följande funktioner hos MC11:

- Anslutning av parallella enheter sker via portarna IO Externt.
- Speciella laborationskort ur ML-serien ansluts via specifika adresser.
- Seriekommunikationskrets SCI0 kan anslutas till IO-simulator ’Console’.
- Undantagshantering har implementerats i simulatorn
  - Software interrupt (swi)
  - SCI0, RTI och ML5/23

Laborationsdator MC12

MC12, den sista i raden av laborationsdatorer från microlf, är byggd kring en Freescale (fd. Motorola) MC9S12DG256-microcontroller. DBG12 är standard monitor/debugger. Även en "minimalvariant" MC12s, avsedd för projektinriktade kurser, producerades.

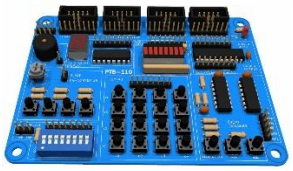
Minnesanvändning:

Typ	startadress	slutadress
RWM	0x1000	0x3FFF
ROM	0x4000	0x7FFF
BANKED ROM: page 0x30	0x8000	0xBFFF
ROM	0xC000	0xFFFF
IO: Internt	0x0	0x03FF
Port serie: SCI0	0xC8	0xCF
Port serie: SCI1	0xD0	0xD7
Timer (CRG)	0x34	0x3F
IO: Externt	0x0400	0x0FFF
Port: OUTPORT	0x0400	0x0400
Port: INPORT	0x0600	0x0600
Port: ML5/ML2	0x0C00	0x0C01
Port: ML5/ML3	0x0C02	0x0C03
Port: ML15/ML2	0x09C0	0x09C1
Port: ML15/ML3	0x09C2	0x09C3

SimServer implementerar följande funktioner hos MC12:

- Anslutning av parallella enheter sker via portarna IO Externt.
- Speciella laborationskort ur ML-serien ansluts via specifika adresser.
- Seriekommunikationskretsarna SCI0 och SCI, kan anslutas till IO-simulator 'Console'.
- Undantagshantering har implementerats i simulatorn
    - Software interrupt (swi)
    - SCI0, SCI1, CRG och ML5/23

Laborationsdator MD307



MD307, "MikroDator 307" (PTB-1000) är en RISC-V (RISC-fem) baserad laborationsdator för undervisningsändamål.

Till MD307 (och MD407) har nya, kompakta laborationskort utvecklats men MD307 kan också enkelt anslutas till laborationskort som tidigare utvecklats för MD407.

MD307 är uppbyggd kring enchipsdatorn CH32V307 med RISC-V. Simulatorn understödjer instruktionsuppsättningarna RV32imafc.

En lång rad IO-enheter kan anslutas till MD307 som därför används i tillämpningar med:

- Parallell in-/ut- matning
- Seriell in-/ut- matning
- Enkla datorspel (av ”reterokaraktär”) med användarinteraktion.
- Realtidskärnor
- Diskoperativsystem

Och naturligtvis kombinationer av dessa.
Minnesanvändning:

Typ	startadress	slutadress
RAM	0x20000000	0x2001FFFF
ROM	0x00000000	0x002FFFFF
System	0xE0000000	0xE00FFFFF
IO	0x40000000	0x500503FF
Port: AFIO/EXTI	0x40010000	0x4001001F
Port: GPIOD	0x40011400	0x40011027
Port: GPIOE	0x40011800	0x40011827
Port: USART1	0x40013800	0x40013817
Port: USART2	0x40004400	0x40004417
Port: PFIC	0xE000E000	0xE000E4FF
Port: EXTI	0x40010400	0x40010417
Port: AFIO	0x40010000	0x4001001B
Port: SYSTICK	0xE000F000	0xE000E027
Port: TIMER6	0x40001000	0x400013FF
Port: TIMER7	0x40001400	0x400017FF
Port: GAME DISPLAY	0xF0000000	0xF00000FF
Port: FDC	0xF0000100	0xF00001FF

Anm: BETA-version, Undantagshantering är bara delvis implementerad. Avbrottsprioriteter är inte implementerat.

Laborationsdator MD407

MD407, "MikroDator 407" utvecklades med syfte att användas för undervisningsändamål under åren 2013/2014 i ett examensarbete på Chalmers/CSE. Den slutliga versionen tillverkades, levererades och togs i bruk första gången 2016.
MD407 är både kraftfull och mångsidig vilket gör den användbar från en grundläggande nivå till mycket avancerade tillämpningar.

MD407 är uppbyggd kring enchipsdatorn ST32F407 med ARM Cortex-M arkitektur. Simulatorn understödjer instruktionsuppsättningarna v6, v7 och flyttalsenheten.

En lång rad IO-enheter kan anslutas till MD407 som därför används i tillämpningar med:

- Parallell in-/ut- matning
- Seriell in-/ut- matning
- Enkla datorspel (av ”reterokaraktär”) med användarinteraktion.
- Realtidskärnor
- Diskoperativsystem

Och naturligtvis kombinationer av dessa.

Minnesanvändning:

Typ	startadress	slutadress
RAM	0x20000000	0x2001FFFF
ROM	0x08000000	0x080FFFFF
System	0xE0000000	0xFFFFFFFF
IO	0x40000000	0x5FFFFFFF
Port: GPIOD	0x40020C00	0x40020FFF
Port: GPIOE	0x40021000	0x400213FF
Port: SCB	0xE000E000	0xE000E000
Port: SCB	0xE000ED00	0xE000ED3F
Port: NVIC	0xE000EF00	0xE000EF00
Port: NVIC	0xE000E100	0xE000E503
Port: SYSCFG	0x40013800	0x40013823
Port: EXTI	0x40013C00	0x40013C17
Port: USART1	0x40011000	0x40011017
Port: USART2	0x40004400	0x40004417
Port: SYSTICK	0xE000E010	0xE000E01F
Port: TIMER2	0x40000000	0x400003FF
Port: TIMER3	0x40000400	0x400007FF
Port: TIMER4	0x40000800	0x40000BFF
Port: TIMER5	0x40000C00	0x40000FFF
Port: TIMER6	0x40001000	0x400013FF
Port: TIMER7	0x40001400	0x400017FF
Port: GAME DISPLAY	0xF0000000	0xF00000FF
Port: FDC	0xF0000100	0xF00001FF

Laborationsdator FRISC

Laborationsdatorn ”FRISC” (Flisp RISC) finns endast i simulatorutförande. Syftet med RiscFlisp är att användas i grundläggande utbildningar i datorteknik.

RiscFlisp är en mycket enkel konstruktion med utgångspunkt i ”Load/Store”-arkitektur och introducerar framför allt till den öppna instruktionsuppsättningen ”RISC-V”.

Minnesanvändning:

Typ	startadress	slutadress
RWM	0	0xFFEF
RWM	0xFFE0	0xFFFF
IO Internt:	0xFF00	0xFFDF
PARIO1	0xFF00	0xFF01
PARIO2	0xFF02	0xFF03
PARIO3	0xFF04	0xFF05
PARIO4	0xFF06	0xFF07
SERIO1	0xFF30	0xFF32
SERIO1	0xFF34	0xFF36

Anm: ALFA-version, undantagshantering är avstängd tills fullständiga tester genomförts.

Strömställare 8-bitars (Dipswitch)

Namnet ”dipswitch” kommer från benämningen ”dual in line package” som syftar till ”fotavtrycket”, dvs. monteringshålen till den standardiserade strömställarkomponenten som finns i åtskilliga utföranden. ML4- och CSE-varianterna har implementerats med en 8-vägs linjeförstärkare som gör att den anslutna porten drivs antingen hög eller låg med CMOS-nivåer.

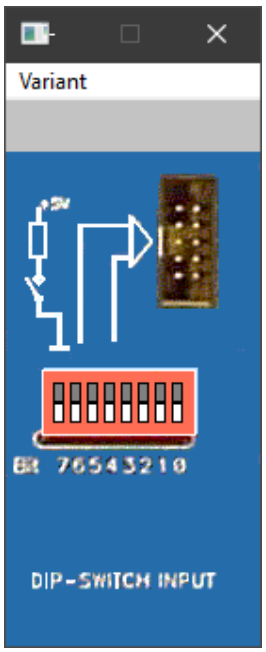
Tillgänglighet:							
MD09	MC11	MC12	MC68	MD68K	MD407	FRISC	MD307
PIA1 A,B PIA2 A,B	0x1600	0x600	PPA	PIT1 A,B PIT2 A,B	GPIO D GPIO E	PARIO1 PARIO2 PARIO3 PARIO4	GPIO D GPIO E

Den enkla formen av inmatningsenhet är en 8-polig strömställare som finns i tre olika varianter.

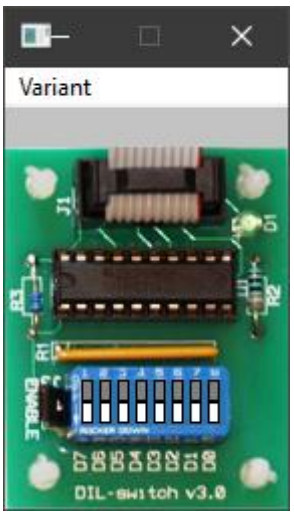
- ML4: den första varianten finns som del av multifunktionskortet ML4, i en konstruktion från *microlf*.
- CSE: den andra varianten konstruerades som ett enkelt tillbehörskort för laborationsmateriel vid Chalmers/CSE.
- PTB-110: den tredje varianten är en öppen källkod-konstruktion från GMV.

I figurena visas strömställarna i öppna lägen, 'off'.

ML4



CSE



PTB-110

Anm: läge 'on' är fast kopplad till VDD i simulatorn.

Typisk användning:
Strömställaren kopplas till en 8-bitars inport hos måldatorn. Då porten avläses ges strömställarens inställda värde.

EXEMPEL:

```
#define PORT_ADDRESS  xxxxxx
value = *( (volatile unsigned char *) PORT_ADDRESS );
```

Diodramp 8 segment (Bargraph)

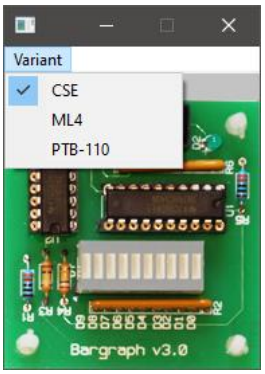
Diodrampen används ofta som nivåindikator (i vertikalt läge) tillsammans med en snabb AD-omvandlare.

Här har vi hittat en betydligt enklare användning, som indikator för en *byte* i binär form. Den verkliga indikatorn hos två av varianterna (ML4 och CSE) har visserligen 10 dioder men vi använder bara 8 utav dom. Komponenten används med en 8-vägs linjeförstärkare som gör att den anslutna portens nivåer driver diodrampen antingen hög eller låg med CMOS-nivåer.

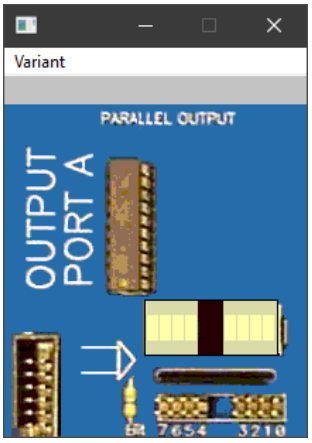
Tillgänglighet:							
MD09	MC11	MC12	MC68	MD68K	MD407	FRISC	MD307
PIA1 A,B PIA2 A,B	0x1400	0x400	PPB	PIT1 A,B PIT2 A,B	GPIO D GPIO E	PARIO1 PARIO2 PARIO3 PARIO4	GPIO D GPIO E

Diodrampen finns i tre olika *varianter*:

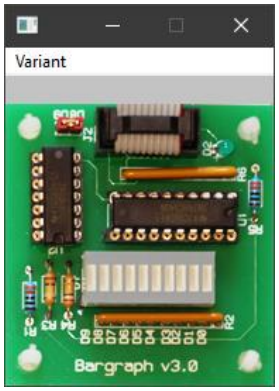
- ML4: den första varianten finns som del av multifunktionskortet ML4, i en konstruktion från *microlf*.
- CSE: den andra varianten konstruerades som ett enkelt tillbehörskort för laborationsmateriel vid Chalmers/CSE.
- PTB-110: den tredje varianten är en öppen källkod-konstruktion från GMV.



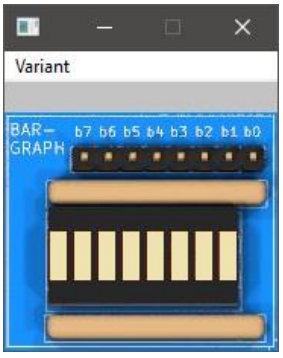
ML4



CSE



PTB-110



Typisk användning:
Diodrampen kopplas till en 8-bitars utport hos måldatorn. Diodrampen visar därefter de senaste värdet som skrevs till porten som ett binärt mönster.

EXEMPEL:

```
#define PORT_ADDRESS  xxxxxx
*( (volatile unsigned char *) PORT_ADDRESS ) = value;
```

Sju-sifferindikator (7-segment display)

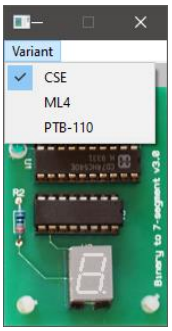
För att underlätta avläsning av digitala resultat kan man använda en 7-sifferindikator. Namnet syftar till att ett antal segment hos indikatorn kan tändas eller släckas, och tillsammans illustrera någon siffersymbol. Antalet segment för detta är då sju med ett åttonde segment som används för att tända en decimalpunkt.

Tillgänglighet:

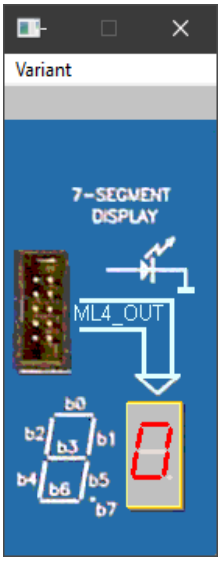
MD09	MC11	MC12	MC68	MD68K	MD407	FRISC	MD307
PIA1 A,B PIA2 A,B	0x1400	0x400	PPB	PIT1 A,B PIT2 A,B	GPIO D GPIO E	PARIO1 PARIO2 PARIO3 PARIO4	GPIO D GPIO E

7-sifferindikatorn finns i tre olika varianter:

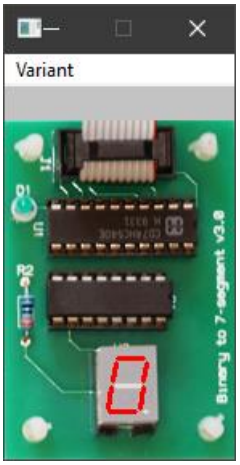
- ML4: den första varianten finns som del av multifunktionskortet ML4, i en konstruktion från *microlf*.
- CSE: den andra varianten konstruerades som ett enkelt tillbehörskort för laborationsmateriel vid Chalmers/CSE.
- PTB-110: den tredje varianten är en öppen källkod-konstruktion från GMV.



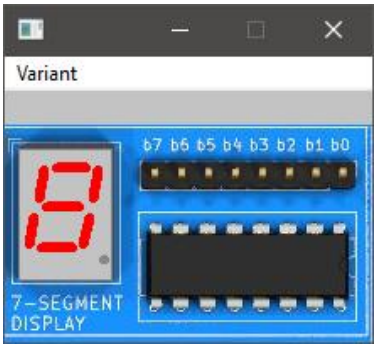
ML4



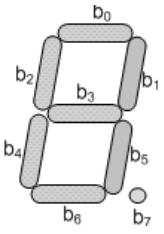
CSE

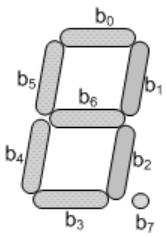


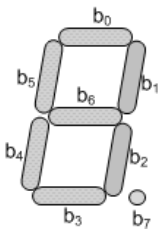
PTB-110



Segmentkodning







Sifferindikatorn kopplas till en 8-bitars utport hos måldatorn. Sifferindikatorn visar därefter ett mönster baserat från det senaste värdet som skrevs till porten. Observera att segmentavkodningen skiljer sig mellan den ursprungliga (ML4) och de andra varianterna.

EXEMPEL:

```
#define PORT_ADDRESS  xxxxxx
*( (volatile unsigned char *) PORT_ADDRESS ) = segment_pattern;
```

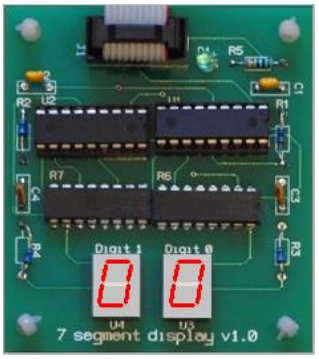
Dubbel sifferindikator

En dubbel sifferindikator (Double hexadecimal display) som visar en *byte* i form av två hexadecimala siffror. (*Most Significant Nybble*, de 4 mest signifikanta bitarna och *Least Significant Nybble*, de 4 minst signifikanta bitarna).

Tillgänglighet:

MD09	MC11	MC12	MC68	MD68K	MD407	FRISC	MD307
PIA1 A,B PIA2 A,B	0x1400	0x400	PPB	PIT1 A,B PIT2 A,B	GPIO D GPIO E	PARIO1 PARIO2 PARIO3 PARIO4	GPIO D GPIO E

Denna modul tillverkades som tillbehörskort vid CSE och finns bara som denna variant.



Dubbel sifferindikator kopplas till en 8-bitars utport hos måldatorn. Sifferindikatorn visar därefter det senaste värdet som skrevs till porten som två hexadecimala siffror.

EXEMPEL:

```
#define PORT_ADDRESS  xxxxxx
*( (volatile unsigned char *) PORT_ADDRESS ) = value;
```

Keypad

Laborationskortet *Keypad*, kan användas som ett mycket enkelt tangentbord med 16 olika tangenter. Den enkla konstruktionen gör att tangentavkodningen måste göras programvarumässigt. För detta används en teknik som kallas *koincidensavsökning*, dvs. en rad aktiveras och kolumninformationen avläses.
Via menyvalet 'Mode' väljs typ av aktivering. Aktivering kan göras som *push/pull*, dvs. en etta aktiverar en rad. Som alternativ kan *open drain* användas, i detta fall är det logikvärdet noll som indikerar en aktiverad rad. Med koincidensavsökning kan man alltså avgöra om en eller flera av radens tangenter är nedtryckta.

Tillgänglighet:

MD09	MC11	MC12	MC68	MD68K	MD407	FRISC	MD307
PIA1 A,B PIA2 A,B				PIT1 A,B PIT2 A,B	GPIO D GPIO E	PARIO1 PARIO2 PARIO3 PARIO4	GPIO D GPIO E

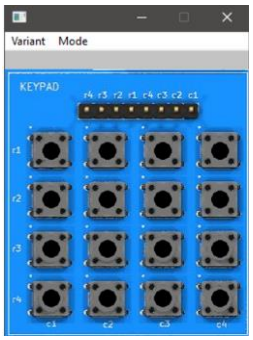
Keypad finns i två varianter:

- CSE: varianten konstruerades som ett enkelt tillbehörskort för laborationsmateriel vid Chalmers/CSE.
- PTB-110: den andra varianten är en öppen källkod-konstruktion från GMV.

CSE

Vänsterklick låser tangent i nedtryckt läge.
Nästa vänsterklick släpper upp låst tangent.

PTB-110

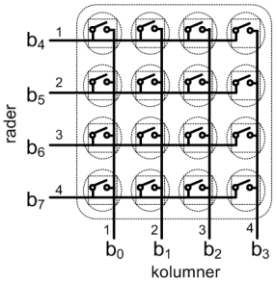


Vänsterklick håller ned tangent tills musen släpps.
Dubbelklick vänster låser tangent i nedtryckt läge.
Nästa vänsterklick släpper då upp låst tangent.

Anslutningar

16 återfjädrande strömställare har kopplats i en matris via en 8-bitars port enligt figuren.

Bitarna b4-b7 är här utsignaler från porten medan bitarna b0-b3 är insignaler till porten.

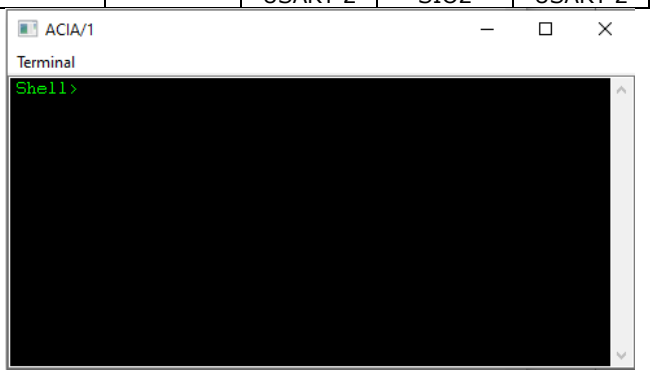


7	6	5	4	3	2	1	0	Register
w	w	w	w	r	r	r	r	IO
rad 4	rad 3	rad 2	rad 1	kol 4	kol 3	kol 2	kol 1	

Terminalemulator (Console)

Console är en klassisk terminalemulator av den allra enklaste typen.

Tillgänglighet:							
MD09	MC11	MC12	MC68	MD68K	MD407	FRISC	MD307
ACIA 1 ACIA 2	SCI 0	SCI 0	DUART A	DUART A	USART 1 USART 2	SIO1 SIO2	USART 1 USART 2



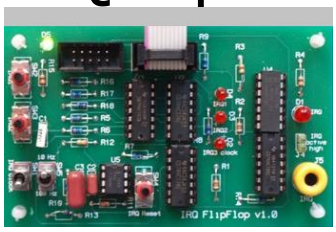
Utöver de skrivbara ASCII-tecknen tolkas enbart 0xD (*Carriage Return*), 0xA (*Line feed*) och 0x8 (*Backspace*).

IO-enheten kan bara anslutas till serieportar hos måldatorn.

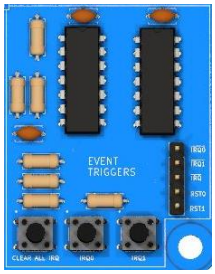
Då *Console* anslutits till en enhet för seriekommunikation kommer tecken som skrivs till denna att visas som ASCII-representation. Samtidigt kommer tecken som matats in via detta fönster att skickas till samma enhet.

Seriekommunikationskretsar av olika typer, och i olika antal, finns tillgängliga hos samtliga laborationsdatorer i *SimServer*. För programmering av dessa kretsar se repektive laborationsdatordokumentation.

IRQ flip flop ("Event triggers")



Laborationskortet *IRQ Flip Flop* används för att generera asynkrona signaler speciellt avsedda att illustrera externa avbrott. Enheten består av tre återfjädrande strömställare och två stycken tvålägesväljare.
Med en bygelfunktion väljs aktiv hög eller låg nivå för triggersignalen.
Två olika avbrott kan aktiveras manuellt via återfjädrande strömställare, en tredje typ av avbrott kommer från en pulsgenerator och ger periodiska avbrott med valbar frekvens 10Hz eller 100 Hz. Aktiverade avbrott kan återställas via strömställare, eller programmerbart.



Motsvarande modul hos PTB-110 *Event triggers* är en enklare variant med samma funktioner bortsett från periodiska avbrott och indikatorer.
Utsignalerna (IRQ0, IRQ1 och IRQ) är här alltid aktivt låga.

Tillgänglighet:							
MD09	MC11	MC12	MC68	MD68K	MD407	FRISC	MD307
					GPIO D GPIO E		GPIO D GPIO E

Modulens funktioner framgår av följande:

För system med separat avbrottsingång kopplas signalen IRQ från den gula banankontakten på kortet.

Programmerarens bild av laborationskortet *IRQ Flip Flop* framgår av följande:

7	6	5	4	3	2	1	0	Register
	w	w	w	r	r	r	r	
	RST2	RST1	RST0	IRQ	IRQ2	IRQ1	IRQ0	

Ett avbrott från Signal IRQ0 ettställer PE0 och kan återställas genom att '1' skrivs till biten PE4 (RST0). På motsvarande sätt kan RST1 och RST2 användas för att återställa avbrott från IRQ1 och IRQ2.
Utöver detta finns signalen IRQ bildad i form av ett ELLER-villkor, denna aktiveras alltså då en (eller flera) av IRQ0, IRQ1 eller IRQ2 aktiveras.

Programmerarens bild av *Event triggers* framgår av:

7	6	5	4	3	2	1	0	Register
			w	w	r	r	r	
			RST1	RST0	IRQ	IRQ1	IRQ0	

Enkla LC displayer (LC Display)

Laborationskortet utvecklades på CSE för att möjliggöra laborationer i form av enklare datorspel av ”retrokaraktär”.

Den övre displayen är pixel-orienterad och används för själva spelplanen (”grafisk”).

Den undre displayen accepterar bara ASCII-tecken och kan användas för att ge instruktioner för spelet.

Tillgänglighet:							
MD09	MC11	MC12	MC68	MD68K	MD407	RFLISP	MD307
					GPIO E		GPIO E

De olika displayerna delar gränssnitt. Funktionerna beskrivs i datablad för respektive enhet. Full funktionalitet, dvs. samtliga kommandon för respektive display, har inte implementerats i SimServer. Inte heller har korrekta tidsegenskaper hos de verkliga enheterna simulerats, däremot de tillståndsmaskiner som implementerar utmatningen. Därför krävs åtminstone korrekta algoritmer för att simulatorn ska ge resultat på displayerna.
Ett ”fungerande” program i SimServer är därför en lämplig utgångspunkt för test och implementering i hårdvaran, men ger inga garantier.
Det omvända gäller däremot i de flesta fall, ett program som fungerar med hårdvaran kommer också att fungera i SimServer.

Registeruppsättning

offset	7	6	5	4	3	2	1	0	Register
0									CTRL
0									STATUS
1									DATA

7	6	5	4	3	2	1	0	Register
	w				w	w	w	CTRL
	E				SEL	RW	RS	

Bit 6: E
Klocksignal för synkronisering med LCD-modulen.
Bit 2: SEL
Välj display, 0=ASCII, 1=grafisk
Bit 1: RW
0=skriv till LCD-modul, 1=läs från LCD-modul.
Bit 0: RS
0=skriv kommando eller läs status. 1=skriv/läs data

7	6	5	4	3	2	1	0	Register
r								STATUS
BSY								

Bit 7: BSY
0=klar att ta emot kommando. 1=upptagen

7	6	5	4	3	2	1	0	Register
rw	rw	rw	rw	rw	rw	rw	rw	DATA
DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	

Dataöverföring till/från LCD-modul

Gränssnitt ASCII-display

Kommando	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Beskrivning
Clear display	0	0	0	0	0	0	0	0	0	1	Skriver 0x20 till DDRAM och sätter adressregistret till 0.
Return home	0	0	0	0	0	0	0	0	1	X	Sätt adressregister till 0 och återställ markör ("cursor")
Entry mode set	0	0	0	0	0	0	0	1	ID	SH	Ange markörens riktning och aktivera skift.
	ID	Increment/Decrement mode, 0: markören flyttas till vänster, 1: markören flyttas till höger									
	SH	Helt Bildminne skift, 0:skift av, 1: skift på									
Display control	0	0	0	0	0	0	1	D	C	B	Sätt displayens funktion
	D	Display av/på, 0:av, 1:på.									
	C	Markör av/på, 0:av, 1:på									
	B	Blinkande markör av/på, 0:av, 1:på									
Function set	0	0	0	0	1	1	N	F	X	X	
	N	Antal rader, 0:1 rad, 1: 2 rader									
	F	Teckenstorlek, 0: 5x8 punkter, 1: 5x11 punkter									
Adress	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Sätt adress för nästa "Skriv data" i teckenminnet
Läs statusbit och adress	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	BF är 1 om displayen är upptagen med att utföra något kommando, 0 annars. Med detta kommando läser man samtidigt adressen för nästa insättning i teckenminnet
Skriv data	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Skriv data till teckenminne
Läs data	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Läs data från teckenminne

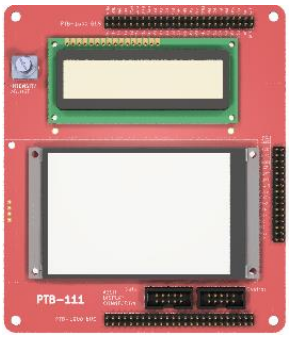
Gränssnitt grafisk display

Gäller endast MD407.

Gränssnittet mot den grafiska displayen på LCD-modulen är mycket likartat ASCII-displayen. Här kan man därför använda drivrutiner som redan är inbyggda hos laborationsdatorer med gränssnitt mot LCD-modulen. De färdiga drivrutinerna är:

```
void graphic_initalize(void);
void graphic_clear_screen (void);
void graphic_pixel_set (int x, int y);
void graphic_pixel_clear (int x, int y);
```

PTB-111 Displaymodul



Den övre displayen (ASCII-display) accepterar bara ASCII-tecken i två rader och 16 positioner.

Den undre displayen (TFT-display) är pixel-orienterad med touch-funktion.

Simulatorn har goda realtidsegenskaper och lämpar sig därför väl för avancerad applikationsutveckling.

Tillgänglighet: MD09	MC11	MC12	MC68	MD68K	MD407	RFLISP	MD307
							GPIO E SW-TRAP

Displayerna har helt olika gränssnitt. Medan ASCII-displayen ansluts via två portar (jämför LC Display) kommunicerar laborationsdatorn med TFT-displayen via inbyggda drivrutiner (se nedan). ASCII-displayen har praktiskt taget identiskt gänssnitt med LC Display.

Gränssnitt ASCII-display

Registeruppsättning

offset	7	6	5	4	3	2	1	0	Register
0									CTRL
0									STATUS
1									DATA

7	6	5	4	3	2	1	0	Register
	w					w	w	CTRL
	E					RW	RS	

Bit 6: E
Klocksignal för synkronisering med LCD-modulen.
Bit 1: RW
0=skriv till LCD-modul, 1=läs från LCD-modul.
Bit 0: RS
0=skriv kommando eller läs status. 1=skriv/läs data

7	6	5	4	3	2	1	0	Register
r								STATUS
BSY								

Bit 7: BSY
0=klar att ta emot kommando. 1=upptagen

7	6	5	4	3	2	1	0	Register
rw	rw	rw	rw	rw	rw	rw	rw	DATA
DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	

Dataöverföring till/från ASCII-display.

Kommando	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Beskrivning
Clear display	0	0	0	0	0	0	0	0	0	1	Skriver 0x20 till DDRAM och sätter adressregistret till 0.
Return home	0	0	0	0	0	0	0	0	1	X	Sätt adressregister till 0 och återställ markör ("cursor")
Entry mode set	0	0	0	0	0	0	0	1	ID	SH	Ange markörens riktning och aktivera skift.
	ID	Increment/Decrement mode, 0: markören flyttas till vänster, 1: markören flyttas till höger									
	SH	Helt Bildminne skift, 0:skift av, 1: skift på									
Display control	0	0	0	0	0	0	1	D	C	B	Sätt displayens funktion
	D	Display av/på, 0:av, 1:på.									
	C	Markör av/på, 0:av, 1:på									
	B	Blinkande markör av/på, 0:av, 1:på									
Function set	0	0	0	0	1	1	N	F	X	X	
	N	Antal rader, 0:1 rad, 1: 2 rader									
	F	Teckenstorlek, 0: 5x8 punkter, 1: 5x11 punkter									
Adress	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Sätt adress för nästa "Skriv data" i teckenminnet
Läs statusbit och adress	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	BF är 1 om displayen är upptagen med att utföra något kommando, 0 annars. Med detta kommando läser man samtidigt adressen för nästa insättning i teckenminnet
Skriv data	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Skriv data till teckenminne
Läs data	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Läs data från teckenminne

Gränssnitt TFT-display

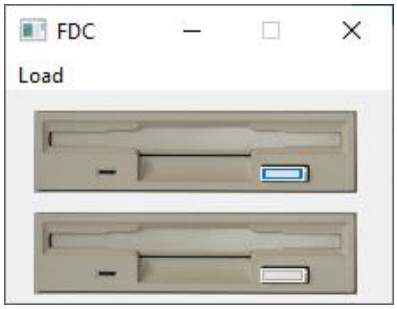
Gränssnittet mot den grafiska displayen utgörs av en uppsättning färdiga drivrutiner som är inbyggda hos såväl laborationsdatorer som *SimServer*.

De inbyggda drivrutinerna är:

```
int tft_init(int);
void tft_lcd_pixel(int x,int y, int colour);
void tft_lcd_line(int  x1, int y1, int x2, int y2, int colour);
void tft_lcd_rect(int x1, int y1, int x2, int y2, uint16_t colour, int fill);
void tft_lcd_ellipse(int xc, int yc, int rx, int ry, int colour, int fill);
void tft_lcd_crosshair(int x,int y,int colour);
void delay_ms(unsigned int ms);
int tft_tp_scan(int tp);
int tft_tp_getpos( int *x, int *y );
```

Drivrutinerna är implementerade som ”trap”-funktioner och beskrivs detaljerat i arbetsbok och laborationsanvisningar.

Floppy disc controller (FDC)



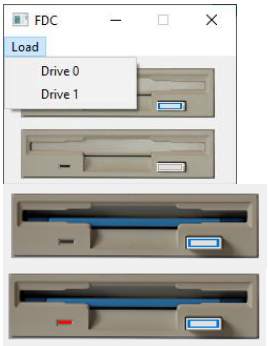
FDC (Floppy Disc Controller, ”flexskiveenhet” är en partiell implementering av klassiska styrkretsar för flexskiveenheter.

Minnesmedia (”virtuella diskar”) skapas och formateras under SimServer’s huvudmeny för att sedan läsas/skrivas via IO-simulatorn.

Styrkretsen hanterar samtidigt två olika flexskiveenheter (0 och 1).

Tillgänglighet:							
MD09	MC11	MC12	MC68	MD68K	MD407	FRISC	MD307
FDC					FDC		FDC

För att kunna avändas måste först en diskett placeras i diskettenheten. Det görs via menyvalet Load.



En diskettenhet med diskett visas enligt figuren till höger.

För att kunna läsa/skriva diskett måste dess diskettenhet vara vald, vilket startar den motor som roterar disketten. Detta illustreras av den lilla röda dioden på diskettenheten.

Registeruppsättning

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0																	CTRL
1																	SEC
2																	S_TRACK
3																	STAT
4																	ADDR_LOW
6																	ADDR_HIGH

Åtkomst av register CTRL, SEC, S_TRACK och STAT måste vara *byte*-format

Åtkomst av register ADDR_LOW och ADDR_HIGH måste vara *halfword*-format

CTRL (*Control*) Styrregister, samtliga bitar är läs- och skrivbara.

offset	7	6	5	4	3	2	1	0	Register
0x00				IR	EX	CMD			CTRL

CMD: (*Command*), kommando till styrkretsen, se tabell ”Kommando” nedan.
EX: (*Execute*), starta utförande av aktuellt kommando.
IR: (*Interrupt request*), styrkretsen genererar avbrott då aktuellt kommando slutförts.

Kommando	CMD			Beskrivning
RESET	0	0	0	Återställning av styrkretsen till initialtillstånd.
RESTORE	0	0	1	Positionera läs/skrivhuvud över spår 0.
SEEK	0	1	0	Positionera läs/skrivhuvud.
RDSEC	0	1	1	Läs sektor. Fälten DS/S/SEC och TRACK måste ha giltiga värden för aktuellt media och registren ADDR_HIGH och ADDR_LOW ska ha en giltig startadress för blocket som ska överföras till minnet.
WRSEC	1	0	0	Skriv sektor. Fälten DS/S/SEC och TRACK måste ha giltiga värden för aktuellt media och registren ADDR_HIGH och ADDR_LOW ska ha en giltig startadress för blockett som ska överföras till disketten.
SEL0	1	0	1	Välj enhet 0
SEL1	1	1	0	Välj enhet 1

SEC (*Sector*) , samtliga bitar är läs- och skrivbara.

offset	7	6	5	4	3	2	1	0	Register
0x01			SEC						SEC

SEC: (Sector), sektor nummer 0-63

S_TRACK (Side/Track), , samtliga bitar är läs- och skrivbara.

offset	7	6	5	4	3	2	1	0	Register
0x02	S	TRACK							S_TRACK

S: (*Side*), för dubbelsidiga media, 0 eller 1.
TRACK: Spår 0-127.

STAT (Status) , samtliga bitar är läsbara.

offset	7	6	5	4	3	2	1	0	Register
0x03		IRQ	BUSY	ERR	EIND				STAT

IRQ: (Interrupt Request) Kommando har utförts och IR (i CTRL) är aktiv.
BUSY: Styrkretsen är upptagen med att utföra kommando.
ERR: (Error) biten ettställs av styrkretsen då något fel uppstår.
EIND: (Error indicator) felindikator, se tabell ”Felindikator” nedan. Felbiten och felindikatorn återställs med RESET-kommando till styrkretsen.

Felindikator	EIND				Beskrivning
E_INVALID	0	0	0	1	Ogiltigt kommando eller operation.
E_NODRV	0	0	1	0	Kommando kan inte utföras eftersom ingen diskett finns i flexskiveenheten.
E_SURFACE	0	0	1	1	Försök att läsa/skriva sida 1 med en enkelsidig diskett.
E_TRACK	0	1	0	0	Försök att positionera läs/skrivhuvud över icke existerande spår.
E_SECTOR	0	1	0	1	Försök att läsa/skriva icke existerande sektor.
E_DMA	0	1	1	0	Fel vid direktminnesöverföring, minne kan inte läsas/skrivas.
E_BUSY	0	1	1	1	Nytt kommando startades då styrkretsen var upptagen. Pågående kommando avbröts.

ADDR_LOW (Address Low)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x04	A15	A15	A15	A15	A15	A15	A15	A15	A15	A15	A15	A15	A15	A15	A15	A15	ADDR_LOW

Registret håller de 16 minst signifikanta bitarna (A15..A0) i den 32-bitars adressen för DMA-överföring till eller från minnet.

ADDR_HIGH (Address High)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x06	A15																ADDR_HIGH

Registret håller de 16 mest signifikanta bitarna (A31-A16) i den 32-bitars adressen för DMA-överföring till eller från minnet.

Enkla drivrutiner

Exemplen på nästa sida illustrerar hur FDC’n initieras och hur en godtycklig sektor kan skrivas respektive läsas. Under exemplen används två vituella disketter som skapats och formaterats (se ”Funktioner/Disc Utilities”).

EXEMPEL 1:

Programmet illustrerar hur FDC'n initieras, därefter skrivs en kort textsträng till sektor 8, spår 5, disk 0 (sida 0).
I vårt första exempel förutsätts att en virtuell diskett laddats i diskettenhet 0.
Följande deklaration och definitioner används i exemplen:

```
typedef struct tag_fdc{
  volatile unsigned char ctrl;
  volatile char sec;
  volatile char s_track;
  volatile char stat;
  volatile short addr_low;
  volatile short addr_high;
} FDCDEV, *PFDCDEV;
```

```
#define FDC ((PFDCDEV) 0xXXXX )
#define FC_RESET      0x8
#define FC_RESTORE    0x9
#define FC_SEEK              0xA
#define FC_READ              0xB
#define FC_WRITE      0xC
#define FC_SEL0              0xD
#define FC_SEL1              0xE
```

```
void FdcInit( void )
{
  FDC->ctrl = FC_RESET;
}

void FdcFail( void )
{
  while(1);
}

int FdcSelDrv( unsigned int drv )
{
  if ( drv > 1 )
    return;
  if( drv )
    FDC->ctrl = FC_SEL1;
  else
    FDC->ctrl = FC_SEL0;
  return (int) (FDC->stat & 0xf);

}
```

```
int FdcWriteSec( int side, int track, int sec, unsigned int add ) {
  FDC->s_track = (unsigned char) (side<<7 | track);
  FDC->ctrl = FC_SEEK;
  if( FDC->stat & 0x10 )
    return (FDC->stat & 0xf);
  FDC->sec = (unsigned char) sec;
  FDC->addr_low = (unsigned short) add;
  FDC->addr_high = (unsigned short) (add>>16);
  FDC->ctrl = FC_WRITE;
  return (int) (FDC->stat & 0xf);
}

void main(void){
  unsigned char buffer[256];
  FdcInit();
  strcpy( buffer, "\nText string, to be written to disc...");
  if( FdcSelDrv(0) )
    FdcFail ();
  if( FdcWriteSec( 0, 5, 8, (unsigned int) &buffer ) )
    FdcFail ();
}
```

EXEMPEL 2:

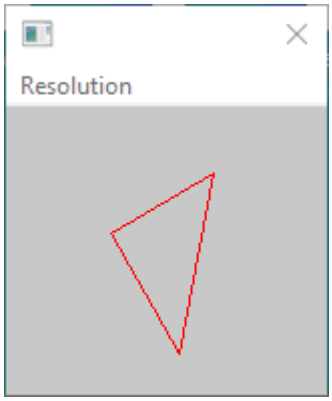
I detta exempel förutsätts att ytterligare en virtuell diskett laddats i diskettenhet 1.

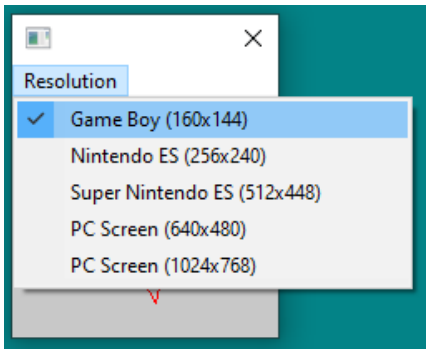
Programmet illustrerar hur FDC'n initieras, därefter kopieras sektor 8, spår 5 från disk 0 (sida 0), till sektor 0, spår 0, disk 1 (sida 0).

```
int FdcReadSec( int side, int track, int sec,
   unsigned int add )
{
  FDC->s_track = (unsigned char)(side<<7|track);
  FDC->ctrl = FC_SEEK;
  if( FDC->stat & 0x10 )
    return (FDC->stat & 0xf);
  FDC->sec = (unsigned char) sec;
  FDC->addr_low = (unsigned short) add;
  FDC->addr_high = (unsigned short) (add>>16);
  FDC->ctrl = FC_READ;
  return (int) (FDC->stat & 0xf);
}
```

```
void main(void)
{
  FdcInit();
  if( FdcSelDrv(0) )
    FdcFail ();
  if( FdcReadSec(0,5,8,(unsigned int) &buffer ) )
    fail();
  if( FdcSelDrv(1) )
    FdcFail ();
  if( FdcWriteSec(0,0,0,(unsigned int) &buffer ) )
    FdcFail ();
}
```

Grafisk display (GLCD)





Detta är en typ av display som enbart finns i IO-simulatorn, dvs. ingen motsvarande hårdvara. Displayen är av en komplex typ med såväl god beräkningskapacitet som ett stort bildminne. Den kan startas med olika ”klassiska” upplösningar.

Olika geometriska objekt; pixel, linje, rektangel, ellips och triangel kan ritas via enkla kommandon till displayen och detta gör den väldigt enkel att programmera.

Tillgänglighet:							
MD09	MC11	MC12	MC68	MD68K	MD407	FRISC	MD307
					GLCD		GLCD

Följande register utgör displayens programmerargränssnitt. Samtliga är 32-bitars register som även kan läsas/skrivas delvis på byte-format.

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0																																	CTRL
4																																	RGB
8																																	COORD1
0x0C																																	COORD2
0x10																																	COORD3

CTRL (Control)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x00	ROT									CMD						EX	CTRL

ROT: De geometriska objekten roteras i steg om 1º. Värden 0-359 är därför giltiga här. Rotationen sker kring den punkt som anges av COORD3.
EX: Execute, skrivbar, vid övergång 0→1 hos denna bit startas det kommando som finns i CMD. Övergången 1→0 har ingen effekt.
CMD: Command, läs- och skrivbar, anger vilket kommando som ska utföras vid nästa Execute. De olika kommandona framgår av tabellen nedan.

RGB (Red Green Blue)

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
4	BGINT								RED								GREEN								BLUE								RGB

BGINT: Background Intensity, displayens bakgrund, gråskala från svart (0) till vit (0xFF).
RED, GREEN, BLUE: Palett, intensitet (24-bit RGB) hos förgrundsfärgen.

COORD1 (Coordinate 1)

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
8							YCOORD																XCOORD										COORD1

YCOORD: Y-kordinat för punkt 1
XCOORD: X-koordinat för punkt 1

COORD2 (Coordinate 2)

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0xC							YCOORD																XCOORD										COORD2

YCOORD: Y-kordinat för punkt 2
XCOORD: X-koordinat för punkt 2

COORD2 (Coordinate 3)

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x10							YCOORD																XCOORD										COORD2

YCOORD: Y-kordinat för punkt 3
XCOORD: X-koordinat för punkt 3

Koordinatregistrens användning framgår av följande tabell.
För kommandon 2-17 gäller att geometrierna kan roteras, se not 1. För geometrin triangel gäller att denna alltid är rätvinklig, se not 2 nedan.

Kommando	CMD					Beskrivning
Clear display	0	0	0	0	0	Displayen raderas och tänds baserat på bakgrundsintensitet.
Fill display	0	0	0	0	1	Displayen raderas och tänds baserat på förgrundsfärg.
Draw pixel	0	0	0	1	0	Rita pixel med färg RGB i COORD1
Clear pixel	0	0	0	1	1	Rita pixel med bakgrundsintensitet i COORD1
Draw line	0	0	1	0	0	Rita linje med färg RGB från COORD1 till COORD2
Clear line	0	0	1	0	1	Rita linje med bakgrundsintensitet från COORD1 till COORD2
Draw rectangle	0	0	1	1	0	Rita rektangel med färg RGB och motstående hörnen COORD1, COORD2
Clear rectangle	0	0	1	1	1	Rita rektangel med bakgrundsintensitet och motstående hörnen COORD1, COORD2
Draw filled rectangle	0	1	0	0	0	Rita fylld rektangel med färg RGB och motstående hörnen COORD1, COORD2
Clear filled rectangle	0	1	0	0	1	Rita fylld rektangel med bakgrundsintensitet och motstående hörnen COORD1, COORD2
Draw ellipse	0	1	0	1	0	Rita ellips med färg RGB, centrum i COORD1, bredd i COORD2(x) och höjd i COORD2(y)
Clear ellipse	0	1	0	1	1	Rita ellips med bakgrundsintensitet, centrum i COORD1, bredd i COORD2(x) och höjd i COORD2(y)
Draw filled ellipse	0	1	1	0	0	Rita fylld ellips med färg RGB, centrum i COORD1, bredd i COORD2(x) och höjd i COORD2(y)
Clear filled ellipse	0	1	1	0	1	Rita fylld ellips med bakgrundsintensitet, centrum i COORD1, bredd i COORD2(x) och höjd i COORD2(y)
Draw triangle	0	1	1	1	0	Rita rätvinklig triangel med färg RGB, se not 2 nedan
Clear triangle	0	1	1	1	1	Rita rätvinklig triangel med bakgrundsintensitet, se not 2 nedan
Draw filled triangle	1	0	0	0	0	Rita fylld rätvinklig triangel med färg RGB, se not 2 nedan
Clear filled triangle	1	0	0	0	1	Rita fylld rätvinklig triangel med bakgrundsintensitet, se not 2 nedan

Not 1: Geometrierna kan roteras kring en rotationspunkt bestämd av COORD3. Den kan vara godtyckligt placerad med undantag av ellipse, där den måste vara placerad utanför ellipsen.
Not 2: Triangeln specificeras med de två koordinaterna COORD1 och COORD2. Endast rätvinkliga trianglar kan därför konstrueras där det gäller för triangelns tre punkter att basen bildas av x1,y1 tillsammans med x2,y1, höjden är linjen mellan x1,y2 och x1,y3, dvs: y2=y1 och x3=x1. För en triangel ska därför COORD1 vara x1,y1 medan COORD2 ska vara x2,y3.

Typisk användning:
I en initieringsfas sätts bakgrundsnyansen till lämplig intensitet.

En geometri som man tidigare ritat kan ”raderas” genom att man ritar exakt samma geometri en gång till, men med samma färg som bakgrunden.

EXEMPEL:

Detta exempel visar först hur displayens register kan illustreras med bitfält. Därefter funktioner för att ställa bakgrund respektive förgrundsfärg och slutligen hur en roterande triangel kan skapas.

```
typedef struct gamedisplay
{
  unsigned int ex:1;
  unsigned int :1;
  unsigned int cmd:5;
  unsigned int rot:9;

  unsigned int :0;
  unsigned int blue:8;
  unsigned int green:8;
  unsigned int red:8;
  unsigned int bgint:8;

  unsigned int :0;
  unsigned int xcoord1:10;
  unsigned int :6;
  unsigned int ycoord1:10;

  unsigned int :0;
  unsigned int xcoord2:10;
  unsigned int :6;
  unsigned int ycoord2:10;

  unsigned int :0;
  unsigned int xcoord3:10;
  unsigned int :6;
  unsigned int ycoord3:10;

} GAMEDISPLAY;

#define  GLCD (( GAMEDISPLAY *) 0xXXXX)
```

```
void setbgnd( int intensity )
{
  GLCD->bgint = intensity;
  GLCD ->cmd = 0;
  GLCD ->ex = 0;
  GLCD ->ex = 1;
}
void setcolour( int r, int g, int b )
{
  GLCD ->red = r;
  GLCD ->green = g;
  GLCD ->blue = b;
}
```

```
void triangle(int rot, int x,int y, int xwidth,
    int ywidth, int draw , int filled )
{
  GLCD->xcoord1 = x;
  GLCD->ycoord1 = y;
  GLCD->xcoord2 = xwidth;
  GLCD->ycoord2 = ywidth;
  if( filled )
  {
    if( draw )
      GLCD->cmd = 16;
    else
      GLCD->cmd = 17;

  }else{
    if( draw )
      GLCD->cmd = 14;
    else
      GLCD->cmd = 15;
  }
  GLCD->rot = rot;
  GLCD->ex = 0;
  GLCD->ex = 1;
}
```

```
void main(void)
{
/* Roterande triangel.
   Rotationscentrum i triangelns tyngdpunkt */

  setbgnd( 200 );
  GLCD->xcoord3 = 80;
  GLCD->ycoord3 = 72;
  setcolour( 255, 0, 0 );
  while(1)
  {
    for( int i = 0; i<360;i = i + 5)
    {
      triangle(i, 60,50, 120, 120, 1 , 0);
      delay();
      triangle(i, 60,50, 120, 120, 0 , 0 );
    }
  }
}
```