Embedded Linux på Olimex SAM9-L9260

Kjell Enblom

22 Mars 2011

Presentationen

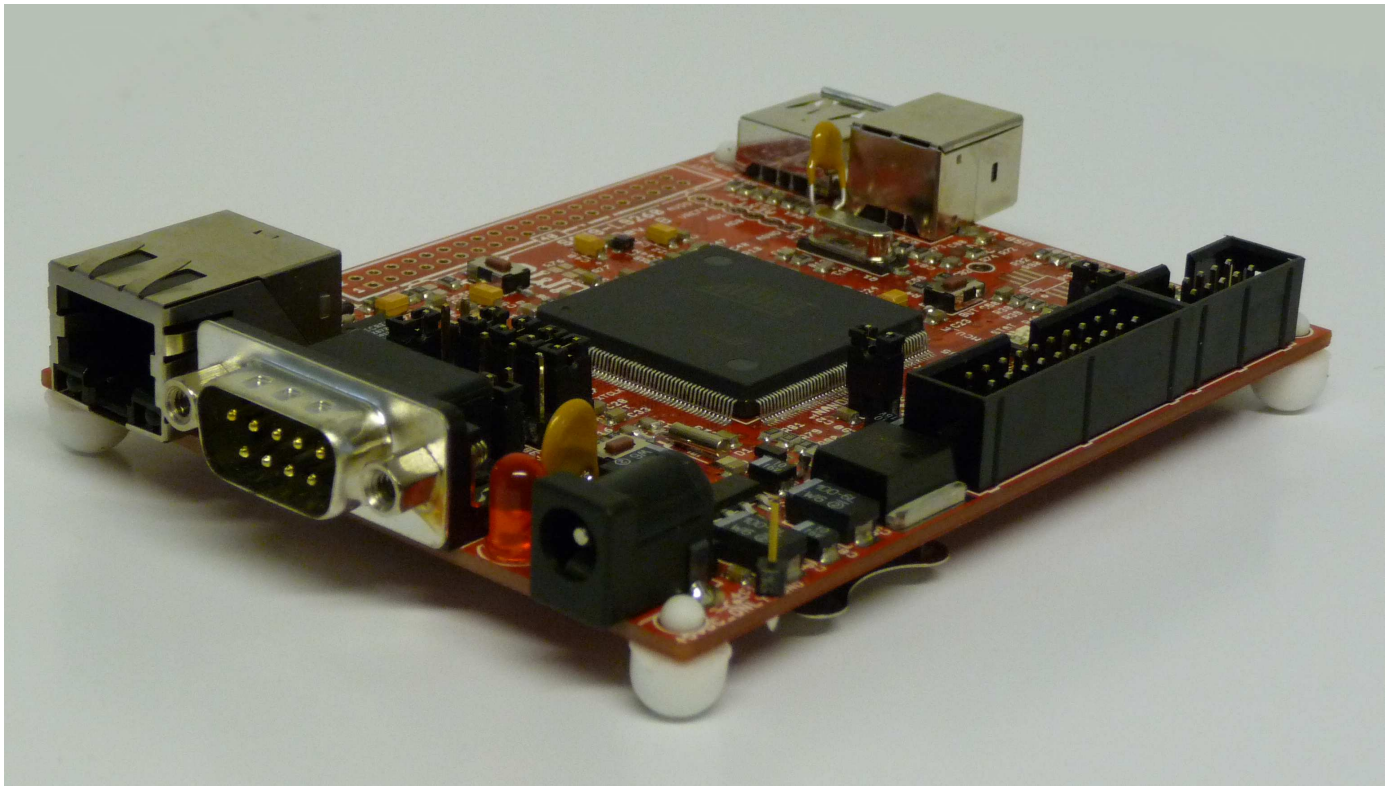
Denna presentation, dokumentation och exempelkod finns på http://www.lysator.liu.se/~kjell-e/embedded/olimex/SAM9-L9260/

Presentationen finns även på:
http://www.lysator.liu.se/~kjell-e/tekla/linux/dokument.html

Introduktion till inbyggda system

- SAM9-L9260 är ett utvecklingskort för inbyggda system.
- Inbyggda system är datorer som sitter inbyggda i apparater; tvättmaskiner, mikrovågsugnar, bilar, digitalboxar för TV, mobiltelefoner, handdatorer, varuautomater etc.
- Inbyggda system har oftast begränsade resurser som t.ex. relativt liten mängd minne, ingen hårddisk, litet flashminne, långsammare CPU än i moderna arbetsstationer och servrar etc.
- Inbyggda system kan t.ex. ha en mängd med I/O-portar, exempelvis för serieportar, I^2C-bussar, I^2S-bussar, CAN-bussar etc.
- Inbyggda system kan sakna minnesskydd.
- Många riktigt små inbyggda system saknar operativsystem.

Olimex SAM9-L9260



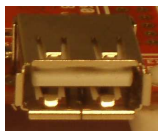
Introduktion till Olimex SAM9-L9260

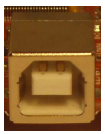
- Avsikten med denna introduktion är att lära sig så pass mycket om Olimexkortet att man snabbt kan komma igång och börja labba med detta kort.
- Innehåll:
  - Introduktion till kortet
  - Grunderna i bootloadern U-Boot
  - Korskompilera busybox
  - Bygga klart root-filsystemet
  - Korskompilera Linuxkärnan
  - Boota med NFS-rootfilsystem
  - Programutveckling

Introduktion till Olimex SAM9-L9260

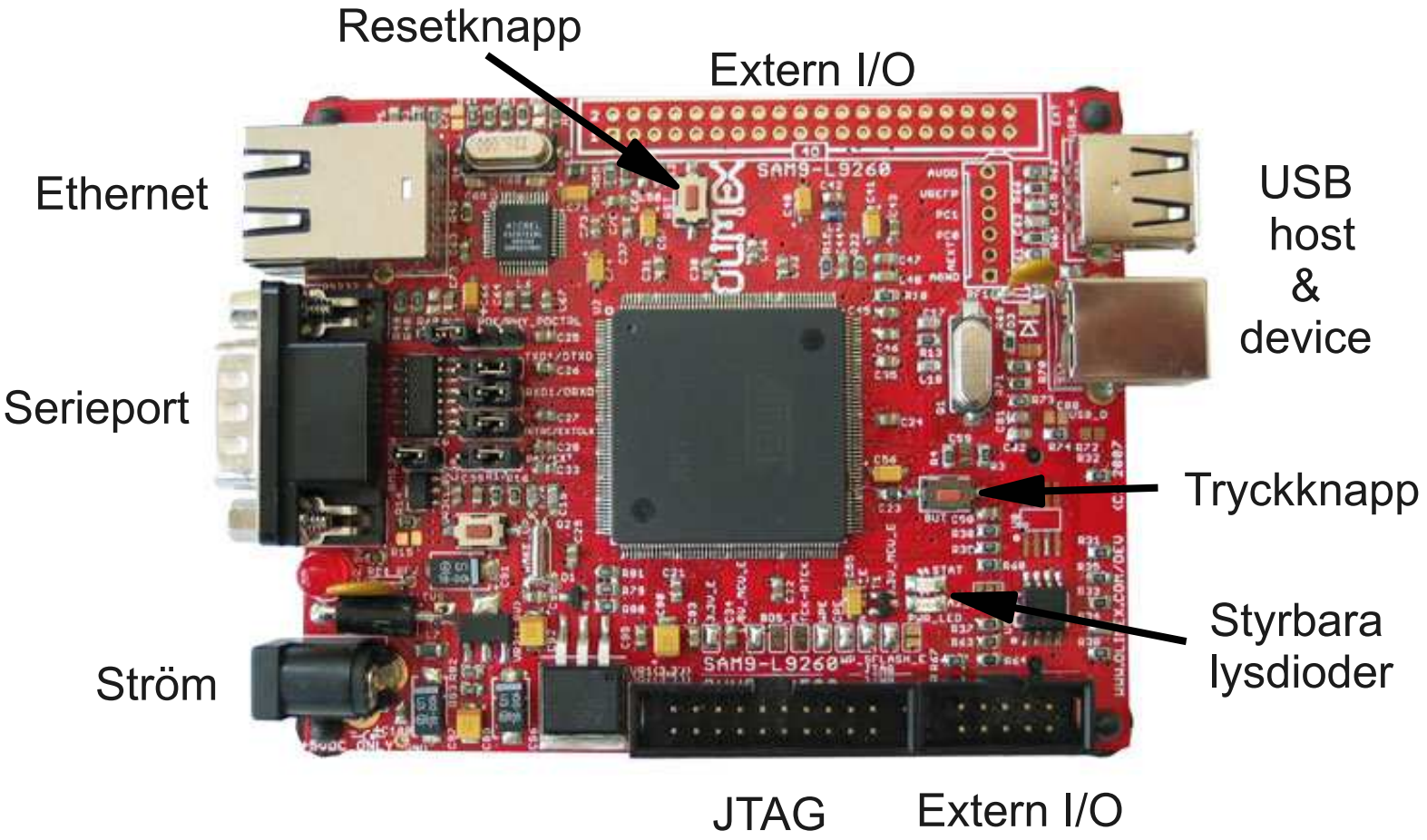
Data i korthet:

- ARM9 (Atmel AT91SAM9260 mikrokontroller med ARM926EJ-S CPU)
- 64 MB SDRAM
- 512 MB NAND-flash
- JTAG-anslutning
- Plats för SD/MMC-kort.
- Ethernetanslutning
- Serieport (9-pol DSUB)
- USB host och USB device
- Tryckknapp
- Styrbara lysdioder
- U-Boot boot-loader
- JFFS2-rootfilsystem med Debian på en del av flashminnet

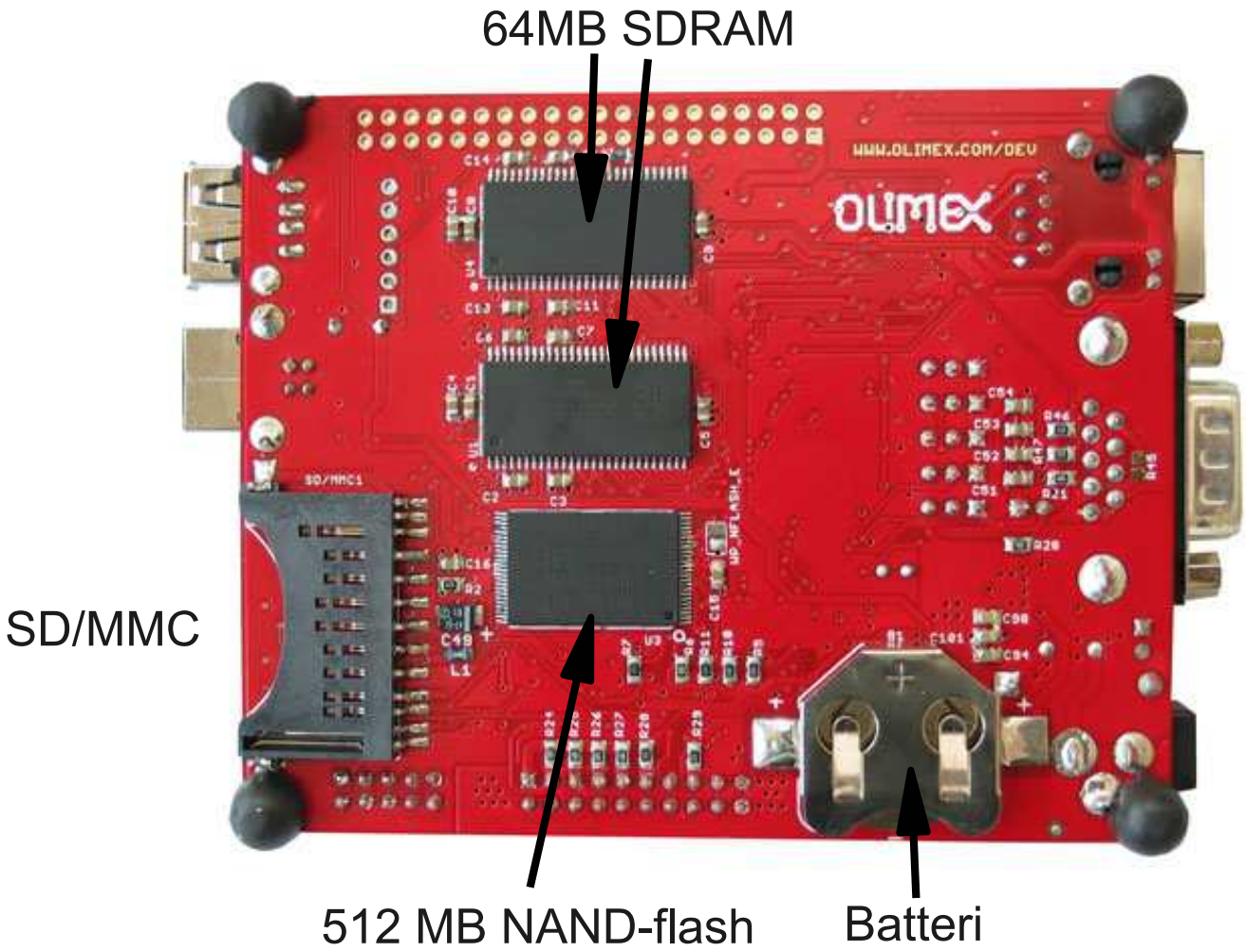




Introduktion till Olimex SAM9-L9260

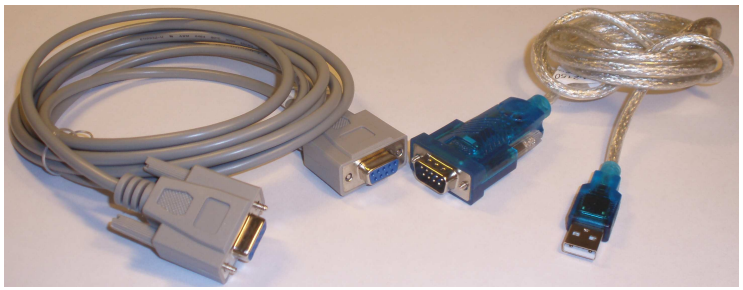


Introduktion till Olimex SAM9-L9260

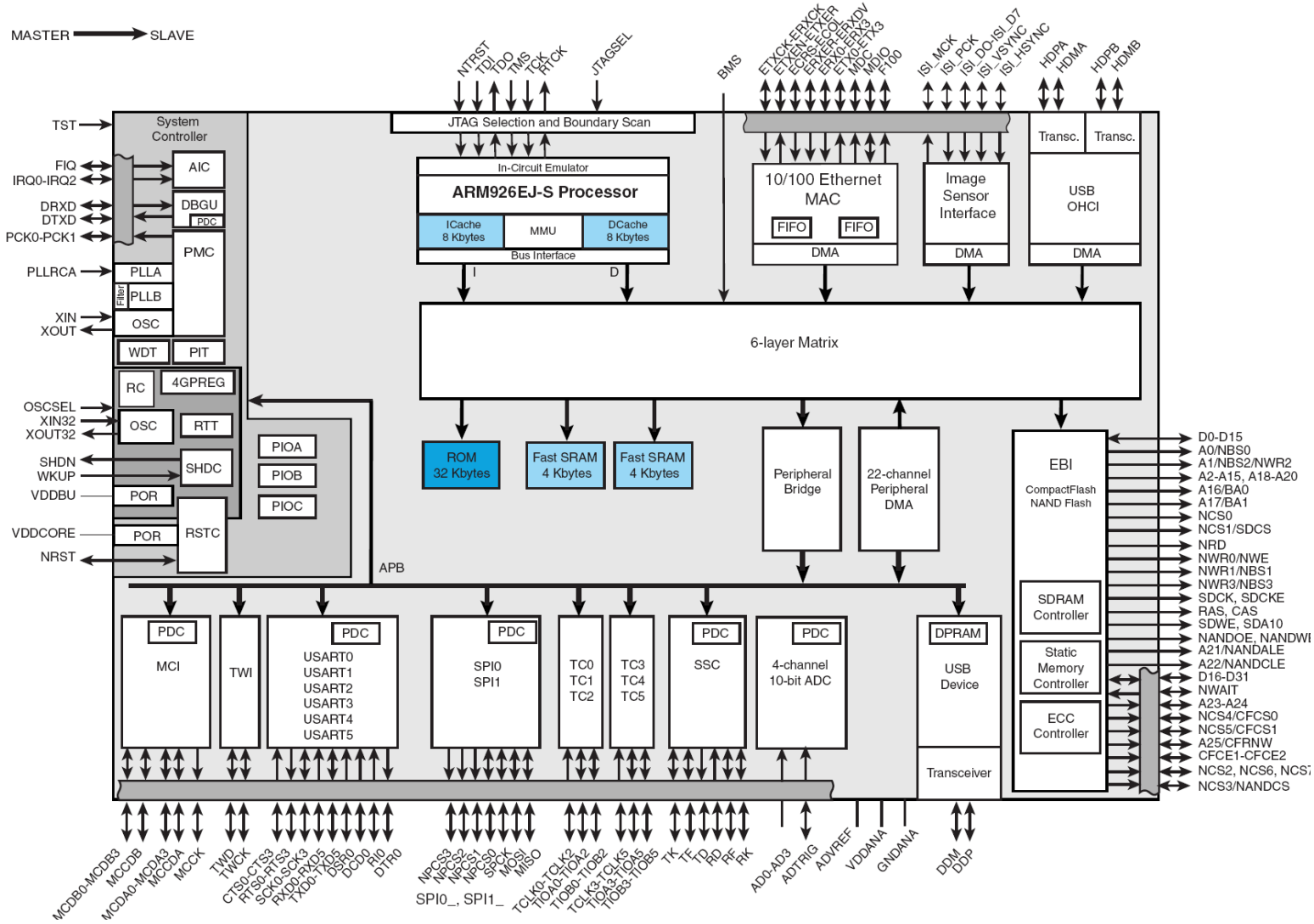


Spänningsmatning och serieportskonsol

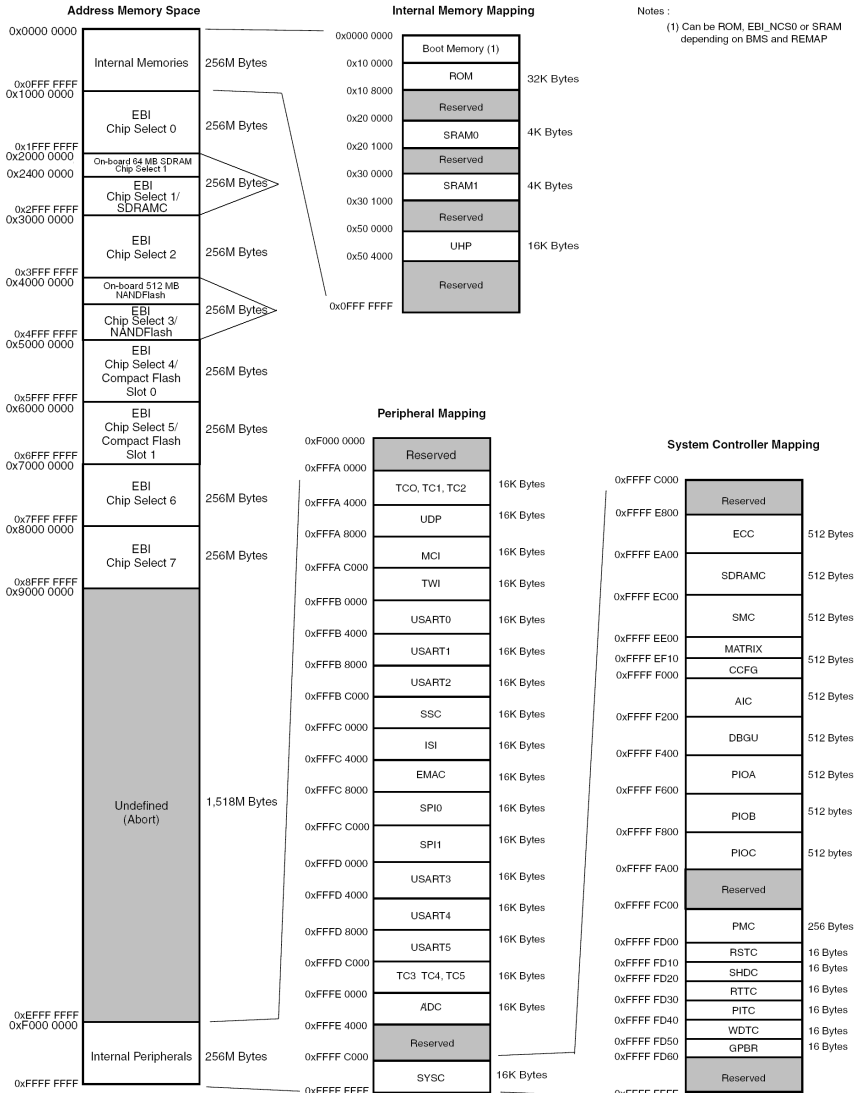
- Olimex SAM9-L9260 ska matas med 5v DC.
- Serieporten är inställd på 115200 bps 8 databitar ingen paritet.
- För att ansluta serieporten till en annan dator behövs en nollmodemkabel som korsar bland annat stiften 2 och 3.
- Nollmodemkabel (9-pol hona-hona) finns att köpa på välsorterade elektronikbutiker som t.ex. Clasohlsson.
- USB till serieport finns även det att köpa på välsorterade elektronikbutiker.



Blockschema för AT91SAM9260



Minneslayout



Grunderna i U-Boot

- Bootloadern på Olimexkortet är Das U-Boot (U-Boot).
- Olimexkortet använder serieportskonsol på vilken den kommunicerar i 115200 bps, 8 databitar, ingen paritet.
- Anslut serieporten och starta t.ex. kermit eller picocom.
- Exempel:
  picocom -b 115200 /dev/ttyUSB0
- Anslut ström till kortet.
- Innan U-Boot har hunnit börja boota tryck på någon tangent. U-Boot har en timeout på 3 sekunder som standard på detta kort.
- Alla inställningar inklusive timeout lagras i variabler i U-Boot.

Grunderna i U-Boot

- Skriv **help** eller **?** för att lista alla kommandon.
- Det går även att skriva **help kommando** för att få mer hjälp om ett specifikt kommando.
- **printenv** visar alla variabler och deras värden.

```
U-Boot> printenv
ethaddr=3a:1f:34:08:54:50
bootdelay=3
baudrate=115200
bootargs=mem=64M console=ttyS0,115200 root=/dev/mtdblock1 rw rootfstype=jffs2
bootcmd=cp.b 0xD0040000 0x22200000 0x001BC124; bootm 0x22200000
stdin=serial
stdout=serial
stderr=serial
ethact=macb0

Environment size: 250/16892 bytes
```

Grunderna i U-Boot

- Olimexkortet kan ladda filer från USB-minne, från en tftp-server, från flash etc.
- USB anges på följande sätt:
  - **usb 0** anger första USB-enheten med fat-filsystem (vfat).
- **fatls** <**interface**> <**dev[:part]**> **[directory]** - listar filer.
- **fatload** <**interface**> <**dev[:part]**> <**addr**> <**filename**> **[bytes]** - laddar en fil.
- addr är en minnessadress.
- Exempel:
  fatls usb 0
- Exempel:
  fatload usb 0 0x22600000 romfs.img

Grunderna i U-Boot

- För att kunna accessa USB-minnen måste man först aktivera USB-stödet.
- USB startas med **usb start**
- USB-stödet stoppas igen med **usb stop**
- USB-partitioner visas med **usb part**
- **help usb** ger mer hjälp om vad man kan göra med USB i U-Boot.
- Exempel på att ladda en fil:
  fatload usb 0 0x22200000 vmlinux.uImage

Grunderna i U-Boot

- **tftpboot [loadAddress] [[hostIPaddr:]bootfilename]** - laddar en fil från en tftp-server och lägger den på angiven minnesadress.
- Exempel:
  tftpboot 0x22200000
  192.168.0.1:/linux-install/olimex/vmlinux.bin
- Exempel 2 (variabeln serverip är satt i detta exempel):
  tftpboot 0x22200000
  /linux-install/olimex/vmlinux.bin
- **bootm addr [arg ...]** - bootar med det som finns på minnesadress addr.
- Exempel:
  bootm 0x22200000
  Exempel med argument:
  bootm 0x22200000 0x22600000

Grunderna i U-Boot

- **run var [...]** - Kör kommandona i variabeln var.
- Exempel:
  - Anta att variabeln tftp_boot innehåller följande:
    tftpboot 0x22600000
    192.168.0.1:/linux-install/olimex/romfs.img;
    tftpboot 0x22200000
    192.168.0.1:/linux-install/olimex/vmlinux.bin;
    bootm 0x22200000 0x22600000
  - **run tftp_boot**
- Det går att betrakta variabeln som ett shellscript som man kör.

Grunderna i U-Boot

- Ändra innehåll på en variabel görs med:
  setenv variabel värde
- Exempel:
  setenv bootargs 'mem=64M console=ttyS0,115200 root=/dev/mtdblock1 rw rootfstype=jffs2'
- Det går att spara alla variablers innehåll till flash med **saveenv**
- **ping host** skickar pingpaket till angiven host. Exempel: **ping 192.168.0.1**

Grunderna i U-Boot

- Standardalternativet för U-Boot är att köra innehållet i variabeln bootcmd, d.v.s. utföra **run bootcmd**
- För Olimex-kortet innebär det att den kopierar kärnan från flash från adress 0xD0040000 till RAM till adress 0x22200000, 0x001BC124 bytes.
- Därefter bootar den från adress 0x22200000.
- Med U-boot går det att flash om den flash som finns på kortet.

U-Boot - U-Boot-filer

- För att kunna använda filer i U-Boot måste dessa filer ha en U-Boot header.
- För att skapa filer i U-Boots format med en header använder man programmet **mkimage** som följer med U-Boot.
- Exempel för ett komprimerat rootfilsystem:
  mkimage -n 'Simple Ramdisk Image'
  -A arm -O linux -T ramdisk -C gzip
  -d ramdisk.image.gz initramfs
- För linuxkärnan är det enklast att se till att mkimage finns med i PATH och sedan kompilera kärnan med
  make ARCH=arm uImage
- Därefter kan filerna laddas enligt beskrivningarna nedan.

U-Boot - boota från USB

- Om man har ett USB-minne med två filer, en fil innehållandes en kärna och en fil med ett rootfilsystem i, då kan man boota på följande sätt.
  - Starta först USB-stödet, ladda in root-filsystemet, ladda kärnan och slutligen kör igång kärnan med adressen till rootfilsystemet som argument.
    - **usb start**
    - **fatload usb 0 0x22200000 vmlinux.bin**
    - **fatload usb 0 0x22600000 romfs.img**
    - **bootm 0x22200000 0x22600000**
- Notera att USB-minnet måste innehålla ett VFAT-filsystem.
- Det går även att göra **usbboot**. Se manualen till U-Boot för mer detaljer.

U-Boot - boota via tftp

- För att ladda filerna från en tftp-server och boota från dessa filer behövs en fungerande tftp-server med de aktuella filerna och en fungerande ethernetförbindelse.
- Nedan visas inställningarna för en tftp-server som startas från xinetd.

```
service tftp
{
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -s /tftpboot
        disable                 = no
        per_source              = 11
        cps                     = 100 2
        flags                   = IPv4
}
```

- tftp-servern ovan chrootas till katalogen /tftpboot

U-Boot - boota via tftp

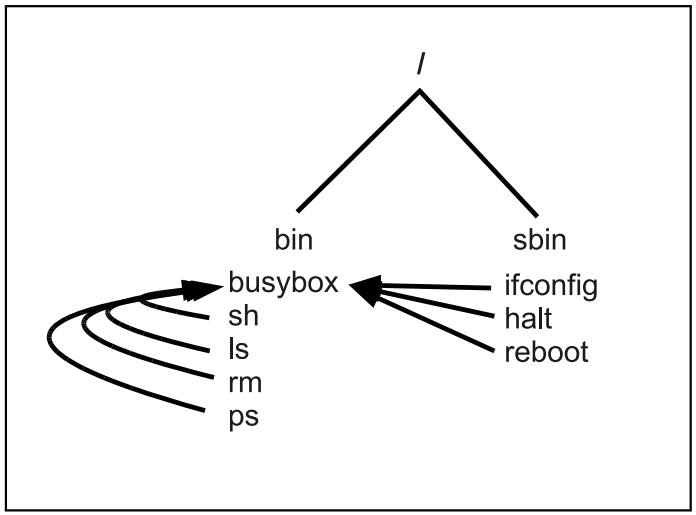
- Filerna i följande exempel är placerade i katalogen /tftpboot/linux-install/olimex/ på host-datorn.
- Tftp-servern har här IP-adress 192.168.0.1.
- Olimexkortet är target.
- Här laddas först filen med root-filsystemet, därefter laddas kärnan och slutligen startas kärnan med adressen till root-filsystemet som argument.
  - **dhcp**
  - **tftpboot 0x22200000 192.168.0.1:/linux-install/olimex/vmlinux.bin**
  - **tftpboot 0x22600000 192.168.0.1:/linux-install/olimex/romfs.img**
  - **bootm 0x22200000 0x22600000**

busybox

- busybox är den schweiziska armekniven för inbyggda system.
- busybox består av **en** programbinär som implementerar många funktioner: sh, ls, rm, mv, ps etc.

busybox

- Med busybox har man en programbinär /bin/busybox som implementerar fler funktioner.
- För varje funktion sätts sedan upp hårda eller mjuka länkar som pekar på busybox.

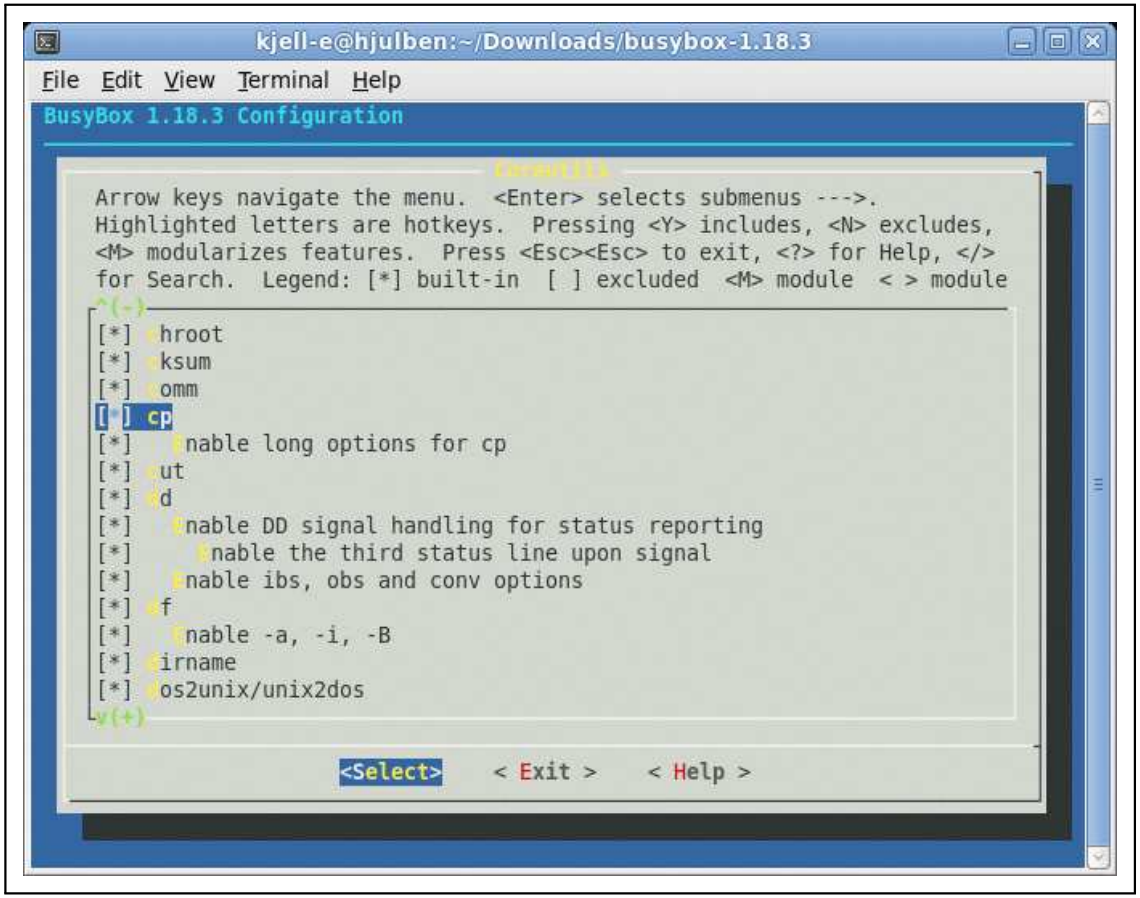


- Vilka funktioner som busybox ska innehålla bestäms vid kompilering.

Kompilera busybox

- Se till att du har ett toolchain installerat och att den finns med i din PATH.
- Ladda hem källkoden till busybox från http://www.busybox.net/.
- Packa upp källkoden och gå ner i katalogen där den hamnade. Ex:
  tar xvfj busybox-1.18.3.tar.bz2; cd busybox-1.18.3
- Starta busybox konfiguration med
  make ARCH=arm menuconfig

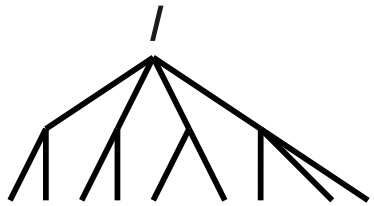
Kompilera busybox



Kompilera busybox

- Notera att man bör sätta toolchain prefix under **'Busybox Settings'→'Build Options'→'Cross Compiler prefix'**
- För t.ex. arm-none-linux-gnueabi-gcc blir prefix arm-none-linux-gnueabi-
- Om man inte har tänkt att lägga in libfiler på sitt målsystem eller vill att busybox ska vara oberoende av libfiler måste busybox länkas statiskt vid kompilering.
- För att länka busybox statiskt gå in på **'Busybox Settings'→'Build Options'** och välj **'Build BusyBox as a static binary'**.
- Konfigurera färdigt busybox och avsluta och spara.
- Kompilera med **make ARCH=arm**
- Installera slutligen med **make ARCH=arm install**

Kompilera busybox

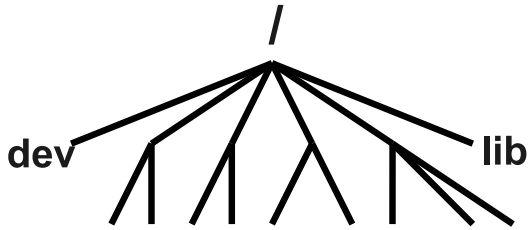


- Nu finns ett rootfilträd i katalogen _install.
- Var rootfilträdet ska installeras går att ställa in när man konfigurerar busybox.
  'Busybox Settings'→'Installation Options'→'BusyBox installation prefix'
- Ett alternativ kan vara att sätta den till /export/embedded/dinuser/root/

Kompilera busybox

- Ett alternativ till att ge ARCH=arm på kommandoraden och sätta toolchain prefix i konfigurationen är att använda kommandoskalsvariablerna ARCH och CROSS_COMPILE
- Exempel:
  export ARCH=arm
  export CROSS_COMPILE=arm-none-linux-gnueabi-
- Därefter kan man konfigurera, kompilera och installera med:
  make menuconfig
  make
  make install

bygga klart rootfilsystemet



- Det som saknas i rootfilsystemet för att det ska vara komplett är devicefiler och eventuella libfiler.
- Devicefiler skapas med **mknod** (som root). Exempel: **mknod rootfilesystem/dev/console c 5 1**
- Här skapas console som är en character device med major numer 5 och minor numer 1.
- Det är major number som knyter ihop device-filen och drivrutinen som hanterar den.

bygga klart rootfilsystemet

- De devicefiler som behövs är åtminstone följande: console, null, ttyS0
- Nedan skapas dessa och några till. Här står vi i det nya root-filsystemet när följande kommandon körs.

```
mknod dev/console c 5 1
mknod dev/null c 1 3
mknod dev/tty c 5 0
mknod dev/tty0 c 4 0
mknod dev/tty1 c 4 1
mknod dev/ttyS0 c 4 64
mknod dev/random c 1 8
mknod dev/urandom c 1 9
mknod dev/mtd1 c 90 2
mknod dev/mtdblock1 b 31 1
```

bygga klart rootfilsystemet

- För att hitta libfilerna till ett toolchain så att dessa kan installeras i katalogen lib i rootfilsystemet kan man köra kompilatorn och ge följande flagga och argument till den --print-file-name libc.so.6
- Exempel:
  arm-none-linux-gnueabi-gcc --print-file-name libc.so.6
- Sök efter libfilerna och kopiera sedan dessa. Exempel:

```
[dinusere@volac]$ arm-none-linux-gnueabi-gcc --print-file-name libc.so.6
/home/dinuser/bin/codesourcery-armgcc-2009q1/bin/..
/arm-none-linux-gnueabi/libc/lib/libc.so.6

[dinusere@volac]$ cp -a /home/dinuser/bin/codesourcery-armgcc-2009q1/bin/../
arm-none-linux-gnueabi/libc/lib /export/embedded/dinuser/root/
```

bygga klart rootfilsystemet

- När systemet bootar startas init som den första processen om man inte anger någonting annat till kärnan.
- Till den busybox init som vi använder här behövs en enkel /etc/inittab som talar om för init vad den ska göra vid boot.

```
::sysinit:/etc/init.d/rcS
ttyS0::respawn:/sbin/getty 115200 ttyS0
::restart:/sbin/init
::shutdown:/bin/umount -a -r
```

- Vid boot kör init först /etc/init.d/rcS
- Därefter startar den /sbin/getty på ttyS0 som ger en loginprompt på serieporten.
- Vid restart ska init starta om sig.
- Vid shutdown ska alla filsystem avmonteras.

bygga klart rootfilsystemet

- /etc/inittab anropar här /etc/init.d/rcS där den i sin tur monterar filsystem och startar loggning, tjänster etc.

```
#! /bin/sh

echo "mounting /proc"
mount -t proc proc /proc
echo "mounting /tmp"
mount -t tmpfs tmpfs /tmp
echo "mounting /sys"
mount -t sysfs sysfs /sys

echo "Starting system logger"
klogd
syslogd

# load module led1
modprobe led1

# Start ssh daemon dropbear
/usr/local/sbin/dropbear
hostname my-tiny-arm-system
```

bygga klart rootfilsystemet

- I exemplet på föregående bild förutsattes att katalogerna /proc /sys och /tmp existerar.
- Om man inte har skapat dessa måste det göras.
- Gör följande på host-systemet/utvecklingsdatorn för att skapa dem
  mkdir
  /export/embedded/dinuser/root/{proc,sys,tmp}
- Det kan även vara trevligt att ha katalogen /var/log för loggfiler
  mkdir -p /export/embedded/dinuser/root/var/log

bygga klart rootfilsystemet

- Om man har ett rootfilsystem som är komplett förutom att det saknar devicefiler då går det att skapa en image-fil med root-filsystem inklusive devicefiler, som icke root, med **genext2fs**.
- genext2fs finns på http://genext2fs.sourceforge.net/

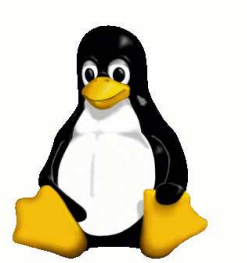
```
# File devicetable.txt
# name   type mode   user    group     major    minor    start     inc     count
/dev       d   755      0        0         -        -        -         -         -
/dev/mem   c   640      0        0         1        1        0         0         -
/dev/tty   c   666      0        0         5        0        0         0         -
/dev/tty   c   666      0        0         4        0        0         1         5
/dev/console c 666      0        0         5        1        0         0         -
/dev/null  c   666      0        0         1        3        0         0         -
/dev/random c 666      0        0         1        8        0         0         -
/dev/urandom c 666     0        0         1        9        0         0         -
/dev/ttyS  c   666      0        0         4        64       0         1         2
/dev/mtd   c   666      0        0         90       0        0         2         2
/dev/mtdblock b 666    0        0         31       0        0         1         2

# Ovan skapas bland annat katalogen /dev
# en character device /dev/mem med major number 1 och minor number 1
# en tty med major 5, minor 0,
# 5 ttyer tty0 - tty4 med major 4 med
# minor number från 0 - 4,
# två mtd, mtd0 och mtd1 med major 90 och minor 0 respektive 2,
# två block devices mtdblock, mtdblock0, mtdblock1 med major number 31
# och minor number 0 respektive 1.
```

bygga klart rootfilsystemet

- En image-fil skapas med **genext2fs -b 4096 -d rootfilesystem -D devicetable.txt -e 0 initramfs.img**
- Här skapas en fil på 4096 blocks (4MB) där innehållet i filträdet rootfilesystem kopieras in i image-filen.
- Devices skapas enligt vad som är definierat i devicetable.txt.
- Resultatet skrivs till initramfs.img
- Om innehållet i rootfilesystem tillsammans med devices inte fyller upp 4MB kommer resten av image-filen att bestå av nollor.
- För att kunna använda denna fil till U-Boot måste en U-Boot header läggas till med mkimage. Exempel:
  mkimage -n 'Ramdisk Image'
  -A arm -O linux -T ramdisk -C none
  -d initramfs.img initramfs.uImage

korskompilera linuxkärnan



- För att kunna korskompilera kärnan behöver man sätta variablerna ARCH och CROSS_COMPILE.
- Detta kan göras i Makefilen högst upp i källkodsträdet till kärnan.
- Alternativt kan man sätta dem i kommandoskalet på samma sätt som för busybox.
- För att konfigurera och kompilera kärnan blir det då:
  make menuconfig
  make uImage

korskompilera linuxkärnan

- Det går även att sätta variablerna på kommandoraden till make. Exempel:
  make ARCH=arm
  CROSS_COMPILE=arm-none-linux-gnueabi-
  menuconfig
- Respektive
  make ARCH=arm
  CROSS_COMPILE=arm-none-linux-gnueabi- uImage
- Kopiera kärnan som finns i arch/arm/boot/uImage och lägg kopian i tftp-katalogen så att den finns tillgänglig via tftp.

korskompilera linuxkärnan

- Kärnmoduler kompileras med: **make modules**
- För att installera kärnmodulerna måste variabeln INSTALL_MOD_PATH sättas till var modulerna ska installeras.
- Denna kan sättas på kommandoraden eller i Makefile högst upp i källkodsträdet till kärnan.
- Lämpligt är att peka på root-katalogen som skapades vid kompilering och installation av busybox.

korskompilera linuxkärnan

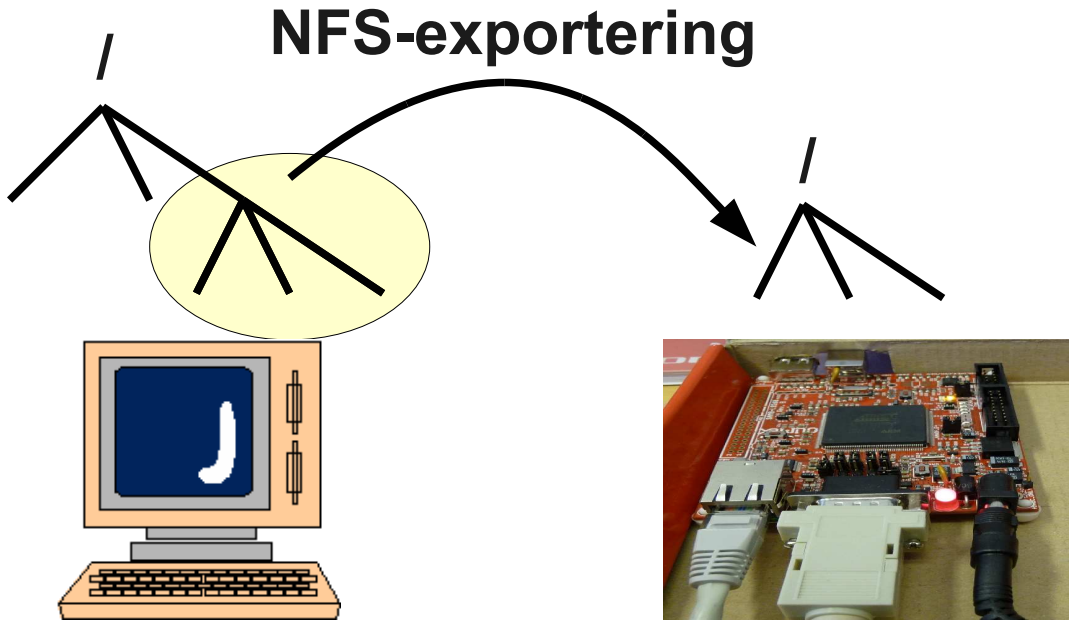
- Sätta variabeln i kommandoskalet görs med **export INSTALL_MOD_PATH=/export/embedded/dinuser/root/**
- Sätt variabeln i Makefilen görs genom att lägga till raden **INSTALL_MOD_PATH = /export/embedded/dinuser/root/** .
- Notera att det måste göras före det att variabeln MODLIB sätts i makefilen då denna använder INSTALL_MOD_PATH.

```
# INSTALL_MOD_PATH specifies a prefix to MODLIB for module directory
# relocations required by build roots.  This is not defined in the
# makefile but the argument can be passed to make if needed.
#

INSTALL_MOD_PATH = /export/embedded/dinuser/root/
MODLIB = $(INSTALL_MOD_PATH)/lib/modules/$(KERNELRELEASE)
export MODLIB
```

Boota med NFS-rootfilsystem

- NFS står för Network File System.
- Med NFS kan ett filträd ligga på en dator och monteras in i filträdet på en annan dator över nätet.

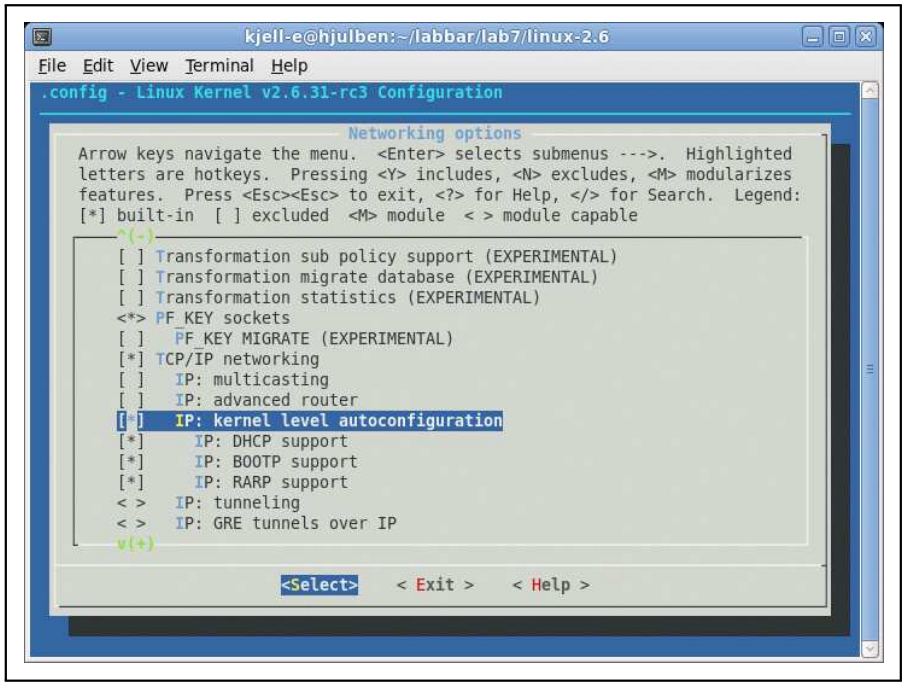


Boota med NFS-rootfilsystem

- Att montera rootfilsystemet över NFS är väldigt praktiskt vid programutveckling.
- Kortet behöver inte flashas om eller bootas om för att installera programvara.
- Program och även kärnmoduler installeras i filträdet på NFS-servern och finns omedelbart tillgängliga på target-systemet.

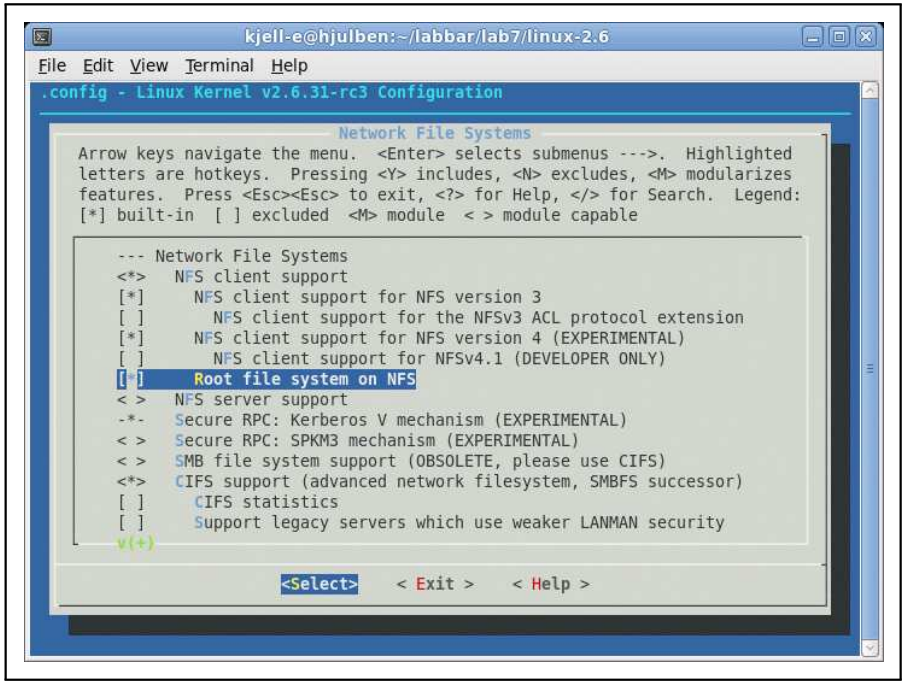
Boota med NFS-rootfilsystem

- För att kunna boota med rootfilsystemet monterat från en NFS-server behöver man göra några inställningar i Linuxkärnan.
- Starta konfigurationen av kärnan och gå till **'Networking'→'Networking options'** och aktivera **'TCP/IP-stöd'**, **'kernel level autoconfiguration'**.



Boota med NFS-rootfilsystem

- Gå till **'File systems'→'Network File Systems'** och aktivera **'NFS file system support'**, **'Provide NFSv3 client support'**, **'Root file system on NFS'**.



Boota med NFS-rootfilsystem

- Kompilera kärnan och lägg den färdiga kärnan i tftp-katalogen så att target-systemet kan ladda in kärnan via tftp.
- För att kunna boota med NFS-rootfilsystem behöver man skicka med NFS-server etc som argument till kärnan.
- Ex:

setenv bootargs 'mem=64M root=/dev/nfs rw console=ttyS0,115200N8

nfsroot=192.168.0.1:/export/embedded/dinuser/root/,proto=tcp

ip=192.168.0.62:192.168.0.1:192.168.0.1:255.255.255.0:olimex:eth0:off'

Boota med NFS-rootfilsystem

- Nedan visas en nedklippt skärmdump på en boot med nfs-rootfilsystem.
- Här får kortet först en dynamisk IP-adress och därefter laddas kärnan från en tftp-server.

```
U-Boot> dhcp

macb0: link up, 10Mbps half-duplex (lpa: 0x0020)
BOOTP broadcast 1
DHCP client bound to address 192.168.0.62
Using macb0 device
TFTP from server 192.168.0.10; our IP address is 192.168.0.62
Filename 'pxelinux.0'.
Load address: 0x22200000
Loading: #
done
Bytes transferred = 13936 (3670 hex)

U-Boot> tftpboot 0x22200000 192.168.0.1:uImage-SAM9L9260-NFS-withoutLED

macb0: link up, 10Mbps half-duplex (lpa: 0x0020)
Using macb0 device
TFTP from server 192.168.0.1; our IP address is 192.168.0.62
Filename 'uImage-SAM9L9260-NFS-withoutLED'.
Load address: 0x22200000
Loading: ##################################################################
done
Bytes transferred = 2853924 (2b8c24 hex)
```

Boota med NFS-rootfilsystem

- Fortsättning...
- Här sätts kärnparametrar och därefter startas kärnan.

```
U-Boot> setenv 'bootargs mem=64M root=/dev/nfs rw
  console=ttyS0,115200N8
  nfsroot=192.168.0.1:/export/olimex/sam9l9260/root/,proto=tcp
  ip=dhcp'

U-Boot> bootm 0x22200000

## Booting kernel from Legacy Image at 22200000 ...
   Image Name:   Linux-2.6.31-rc3-olimex-with-nfs
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2853860 Bytes =  2.7 MB
   Load Address: 20008000
   Entry Point:  20008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux.....................................................
Linux version 2.6.31-rc3-olimex-with-nfs-without-led (kjell-e@hjulben.eis.semcon
CPU: ARM926EJ-S [41069265] revision 5 (ARMv5TEJ), cr=00053177
CPU: VIVT data cache, VIVT instruction cache
Machine: Olimex SAM9-L9260
```

Boota med NFS-rootfilsystem

- Fortsättning...

```
rtc-at91sam9 at91_rtt.0: hctosys: unable to read the hardware clock
IP-Config: Complete:
     device=eth0, addr=192.168.0.62, mask=255.255.255.0, gw=192.168.110.1,
     host=olimex, domain=, nis-domain=(none),
     bootserver=192.168.0.1, rootserver=192.168.0.1, rootpath=
Looking up port of RPC 100003/2 on 192.168.0.1
Looking up port of RPC 100005/1 on 192.168.0.1
VFS: Mounted root (nfs filesystem) on device 0:13.
Freeing init memory: 128K
eth0: link up (10/Half)
mounting /proc
mounting /tmp
mounting /sys
Starting system logger

Du har kommit till min tiny-linux.
my-tiny-arm-system login: root
Password:
#
```

Programutveckling

- För att kunna utveckla program till SAM9-L9260 behövs ett toolchain.
- Några exempel på toolchains är: codesourcery-armgcc-2009q1, toolchain byggt med hjälp av crosstool-ng, Ångström toolchain, snapgear, buildroot m.fl.
- Make är praktisk att ha men inte nödvändig.
- Något standardbibliotek, glibc, eglibc, newlib, dietlibc etc. måste man ha.
- Oftast brukar standardbiblioteken följa med i det toolchain man använder.

Programutveckling

- Med make är det enklare att korskompilera sina program.
- Exempel på en enkel Makefile.

```
CROSS_COMPILE ?= arm-none-linux-gnueabi-
INSTALLDIR ?= /export/embedded/dinuser/root/

AS              = $(CROSS_COMPILE)as
LD              = $(CROSS_COMPILE)ld
CC              = $(CROSS_COMPILE)gcc
CPP             = $(CC) -E
AR              = $(CROSS_COMPILE)ar
NM              = $(CROSS_COMPILE)nm
STRIP           = $(CROSS_COMPILE)strip
OBJCOPY         = $(CROSS_COMPILE)objcopy
OBJDUMP         = $(CROSS_COMPILE)objdump

CFLAGS = -Wall -pedantic -g
LDFLAGS = -static

program: program.o
        $(CC) $^ $(LDFLAGS) -o $@

clean:
        \rm -f program *.o *~
install: program
        cp program $(INSTALLDIR)/usr/bin/
```

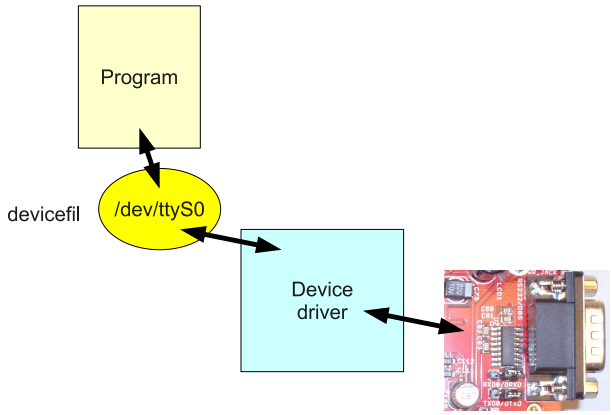
Programutveckling

- Sedan är det bara att köra **make** för att korskompilera programmet.

```
[kjell-e@hjulben testprogram]$ make
arm-none-linux-gnueabi-gcc -Wall -pedantic -g    -c -o program.o program.c
arm-none-linux-gnueabi-gcc  program.o  -static  -o program
```

- Installera programmet med **make install**
- Mer om make går att läsa på http://www.linux.se/forum/viewtopic.php?f=13&t=35266
- Mer om gcc finns att läsa på http://www.linux.se/forum/viewtopic.php?f=13&t=35265
- Mer om libfiler finns att läsa på http://www.linux.se/forum/viewtopic.php?f=13&t=35267

Programutveckling - I/O



- Accessa hårdvara och I/O måste göras från kernel space.
- Lämpligtvis skriver man en device driver som gör att hårdvaran är tillgänglig från user space via en device-fil.
- I/O är minnesmappad på Olimex SAM9-L9260.

Inköpslista

Om du tänker skaffa ett eget Olimex SAM9-L9260-kort behövs följande:

- Olimex SAM9-L9260 utvecklingskort (finns att köpa på http://www.lawicel-shop.se/).
- Spänningsaggregat/batterieliminator 5V DC med plus på mittstiftet.
- USB till serieportsadapter.
- Nollmodemkabel 9pol hona-hona.
- Nätverkssladd (TP-kabel).
- SD-kort.
- USB-minne.
- USB-sladdar, för att kunna ansluta kortet som en deviceenhet till en dator respektive för att kunna ansluta olika enheter till Olimexkortet.

Böcker

- Linux Device Drivers, 3rd edition, Alessandro Rubini, O'Reilly online-version finns på http://lwn.net/Kernel/LDD3/ (pdf)
- Building Embedded Linux Systems, Second Edition, Philippe Gerum, Karim Yaghmour, Jon Masters, Gilad Ben-Yossef, August 2008, O'Reilly.
- Programming Embedded Systems: With C and GNU Development Tools, 2nd Edition, Michael Barr och Anthony Massa, 2006, O'Reilly.
- Designing Embedded Hardware, John Catsoulis, 2005, O'Reilly.
- Embedded Linux Primer A Practical Real-World Approach 2nd ed, Christopher Hallinan, 2010, Prentice Hall.

Länkar

- Manual till U-Boot, http://www.denx.de/wiki/DULG/Manual
- Embedded Linux Developer Forum, http://www.ucdot.org/
- SparkFun Electronics, http://forum.sparkfun.com/
- Busybox, http://www.busybox.net
- Linux på SAM, http://www.at91.com/linux4sam/bin/view/Linux4SAM/
- Linux on ARM Wiki, http://www.linux-arm.com/
- Arm, http://www.arm.com/
- crosstool-ng, http://freshmeat.net/projects/crosstool-ng/
- Ångström toolchain, http://www.angstrom-distribution.org/toolchains/

Frågor

- Frågor?


EIS
semcon