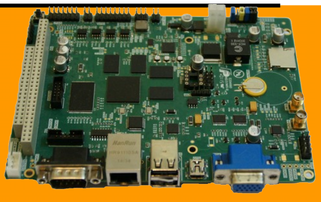
Embedded Linux From Scratch

Embedded Linux From Scratch On Apex-V210

Nasser Afshin
Afshin.Nasser@gmail.com
Samane Sanat Taha Co.

A work originated from **Free Electrons**
http://free-electrons.com/

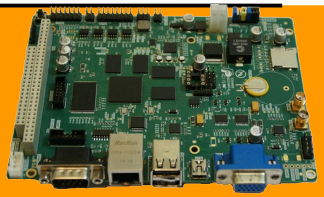
Created with LibreOffice.org 4.4.x



Rights to copy

Document sources, updates and translations:
http://free-electrons.com/docs/elfs

Corrections, suggestions, contributions and translations are welcome!

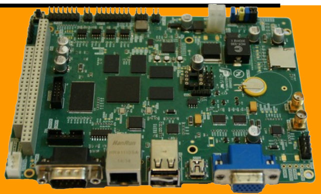


Workshop goals

Build a tiny embedded system entirely from scratch, in 90 minutes

- U-boot configuration and cross-compiling
- Linux kernel configuring and cross-compiling
- Busybox cross-compiling and installation
- Root filesystem creation
- Device file creation
- System initialization scripts: virtual filesystems, networking
- Setup of a simple HTTP interface to the system

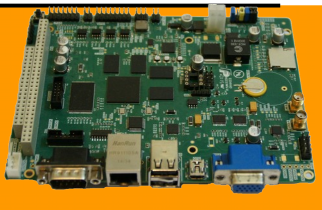
Show you how simple this can be!



Top-down approach

Top-down approach to building an embedded system

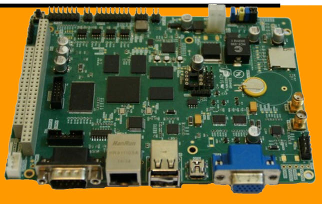
- Starting from a complete desktop GNU/Linux distribution (Debian, Fedora...) and removing unneeded stuff.
- Very tedious job: need to go through a huge number of files and packages. Need to understand what each file and package is about before removing it.
- Keeping unnecessarily complex scripts and configuration files.
- The end result is still quite big, as standard desktop toolsets and libraries are used. Lots of shared libraries still needed too.



Bottom-up approach

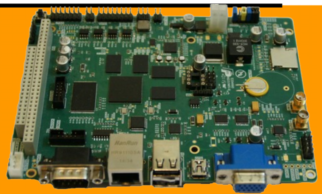
Bottom-up approach to building embedded systems

- Starting with an empty or minimalistic root filesystem, adding only things that you need.
- Much easier to do! You just spend time on things you need.
- Much easier to control and maintain: you build an understanding about the tools you use.
- You only need very simple configuration scripts.
- The end result can be extremely small, all the more as you use lightweight toolsets instead.



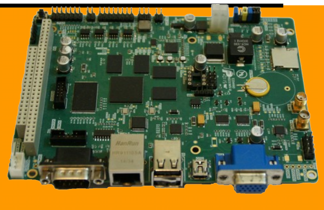
Embedded Linux From Scratch

Tools used in this workshop



Shopping list: hardware for this workshop

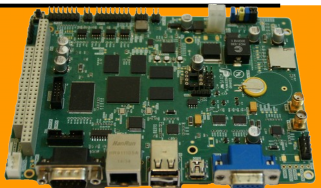
- Samaneh Sanat Taha Apex-V210 board - Available from Samaneh Sanaat Taha Co.
- USB Serial Cable - Male end: Gearmo: https://www.gearmo.com/shop/usb-to-rs-232-serial-adapter-db9-8inch/
- An SD card with at least 128 MB of capacity



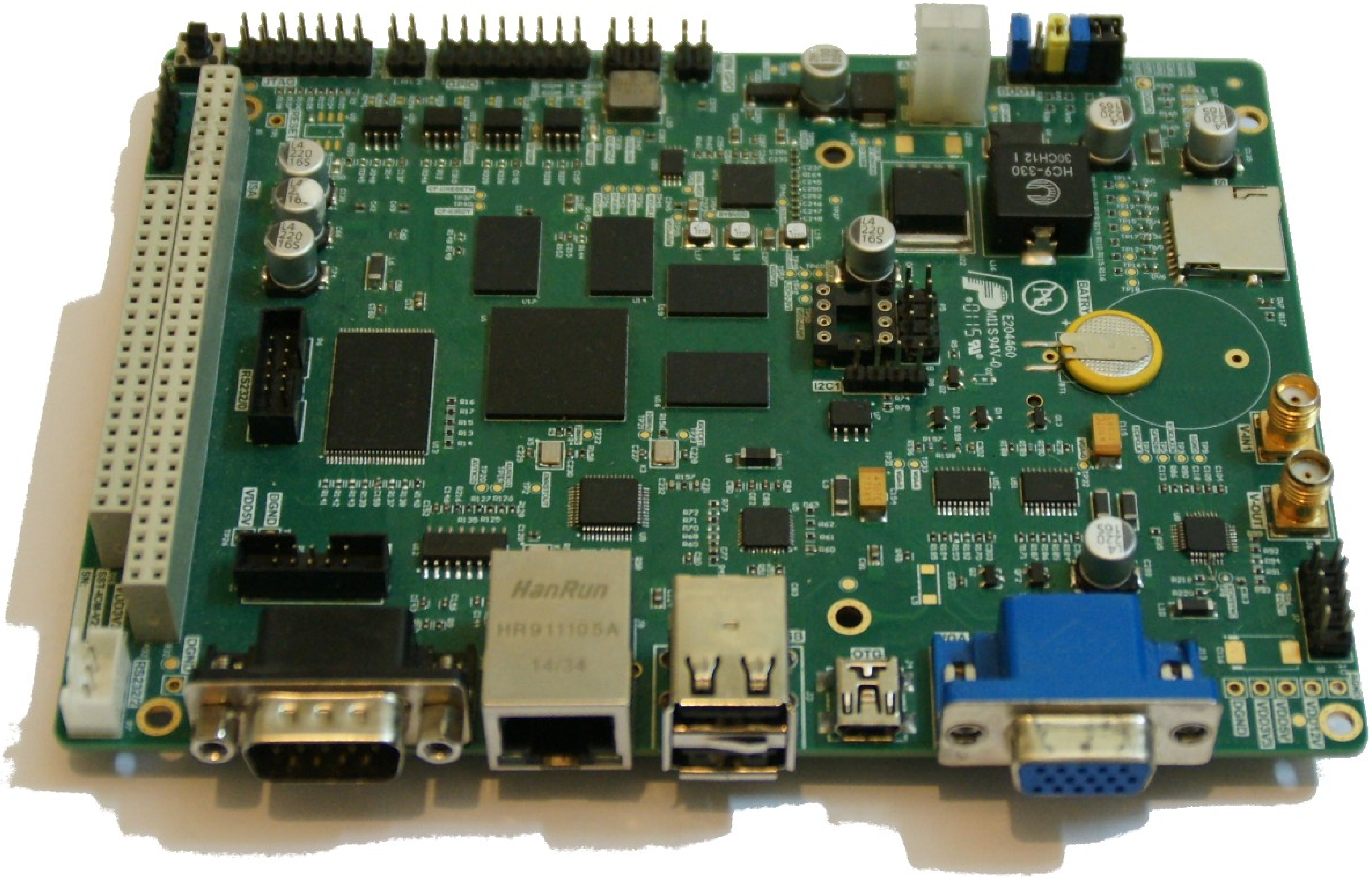
Apex-V210 Hardware

- S5PV210 (Cortex A8) CPU from Samsung
- Up to 1GB DDR2 RAM, Up to 1GB NAND flash
- 1 Ethernet port (10/100 Mbit)
- 2 USB 2.0 host, 1 USB OTG
- 1 MMC/SD slot, 1 Compact Flash slot
- 4 serial ports (RS-422 / RS-232)
- Standard ISA port
- AV input/output
- VGA/LCD ports
- Misc: JTAG, LEDs, GPIOs (Configurable),I2C, SPI, Power Management
- Designed and developed by ”Samane Sanat Taha Co.”

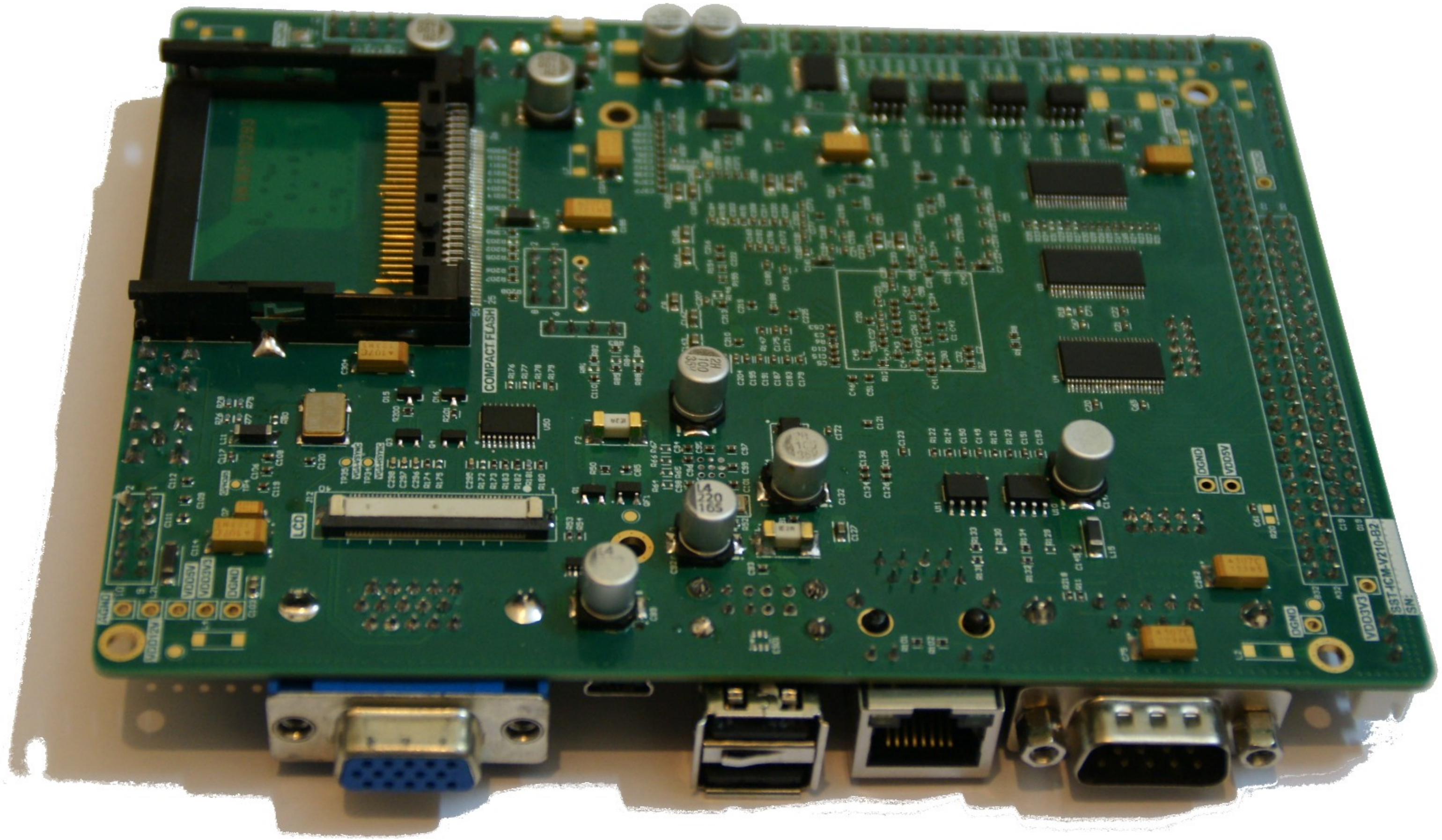
SAMANE SANAT TAHA

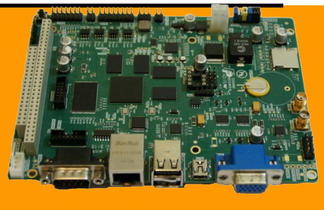


Apex-V210 Top-View



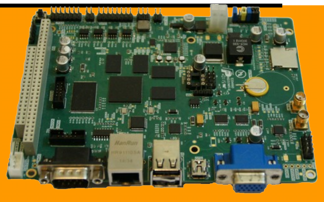
Apex-V210 Bot-View





Software Components

- Cross-compilation toolchain
  - Compiler that runs on the development machine, but generates code for the target
- Bootloader
  - Started by the hardware, responsible for basic initialization, loading and executing the kernel
- Linux Kernel
  - Contains the process and memory management, network stack, device drivers and provides services to user space applications
- C library
  - The interface between the kernel and the user space applications



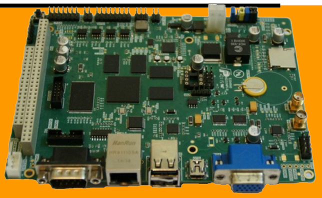
General purpose toolbox: busybox

http://www.busybox.net/ from Codepoet Consulting

- Most Unix command line utilities within a single executable!
  Even includes a web server!
- Sizes less than 1 MB (statically compiled with glibc)
  less than 500 KB (statically compiled with uClibc)
- Easy to configure which features to include
- The best choice for
  - Initrds with complex scripts
  - Any embedded system!

Busybox commands!

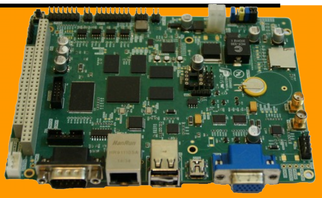
addgroup, adduser, adjtimex, ar, arping, ash, awk, basename, bunzip2, bzcat, cal, cat, chgrp, chmod, chown, chroot, chvt, clear, cmp, cp, cpio, crond, crontab, cut, date, dc, dd, deallocvt, delgroup, deluser, devfsd, df, dirname, dmesg, dos2unix, dpkg, dpkg-deb, du, dumpkmap, dumpleases, echo, egrep, env, expr, false, fbset, fdflush, fdformat, fdisk, fgrep, find, fold, free, freeramdisk, fsck.minix, ftpget, ftpput, getopt, getty, grep, gunzip, gzip, halt, hdparm, head, hexdump, hostid, hostname, **httpd**, hush, hwclock, id, ifconfig, ifdown, ifup, inetd, init, insmod, install, ip, ipaddr, ipcalc, iplink, iproute, iptunnel, kill, killall, klogd, lash, last, length, linuxrc, ln, loadfont, loadkmap, logger, login, logname, logread, losetup, ls, lsmod, makedevs, md5sum, mesg, mkdir, mkfifo, mkfs.minix, mknod, mkswap, mktemp, modprobe, more, mount, msh, mt, mv, nameif, nc, netstat, nslookup, od, openvt, passwd, patch, pidof, ping, ping6, pipe_progress, pivot_root, poweroff, printf, ps, pwd, rdate, readlink, realpath, reboot, renice, reset, rm, rmdir, rmmod, route, rpm, rpm2cpio, run-parts, rx, sed, seq, setkeycodes, sha1sum, sleep, sort, start-stop-daemon, strings, stty, su, sulogin, swapoff, swapon, sync, sysctl, syslogd, tail, tar, tee, telnet, telnetd, test, tftp, time, top, touch, tr, traceroute, true, tty, udhcpc, udhcpd, umount, uname, uncompress, uniq, unix2dos, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, watch, watchdog, wc, wget, which, who, whoami, xargs, yes, zcat



glibc

http://www.gnu.org/software/libc/

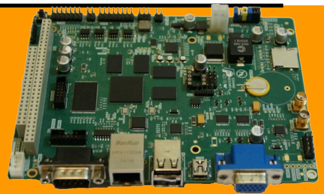
- C library from the GNU project
- Designed for performance, standards compliance and portability
- Found on all GNU / Linux host systems
- Quite big for small embedded systems: about ~1.7MB on `arm` (Familiar Linux iPAQs - `libc`: 1.2 MB, `libm`: 500 KB)
- Example "hello world" program size: 12 KB (dynamically linked), 350 KB (statically linked).



uClibc

http://www.uclibc.org/ from CodePoet Consulting

- Lightweight C library for small embedded systems, with most features though.
- The whole Debian Woody was recently ported to it... You can assume it satisfied most needs!
- Example size (`arm`): approx. 400KB (`libuClibc`: 300 KB, `libm`: 55KB)
- Example “hello world” program size: 2 KB (dynamically linked), 18 KB (statically linked).

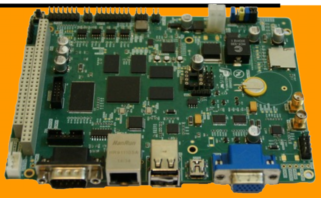


Kernel userspace interface

A few examples:

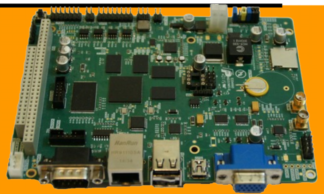
- `/proc/cpuinfo`: processor information
- `/proc/meminfo`: memory status
- `/proc/version`: version and build information
- `/proc/cmdline`: kernel command line
- `/proc/<pid>/environ`: calling environment
- `/proc/<pid>/cmdline`: process command line

... and many more! Complete details in the kernel sources:
`Documentation/filesystems/proc.txt`



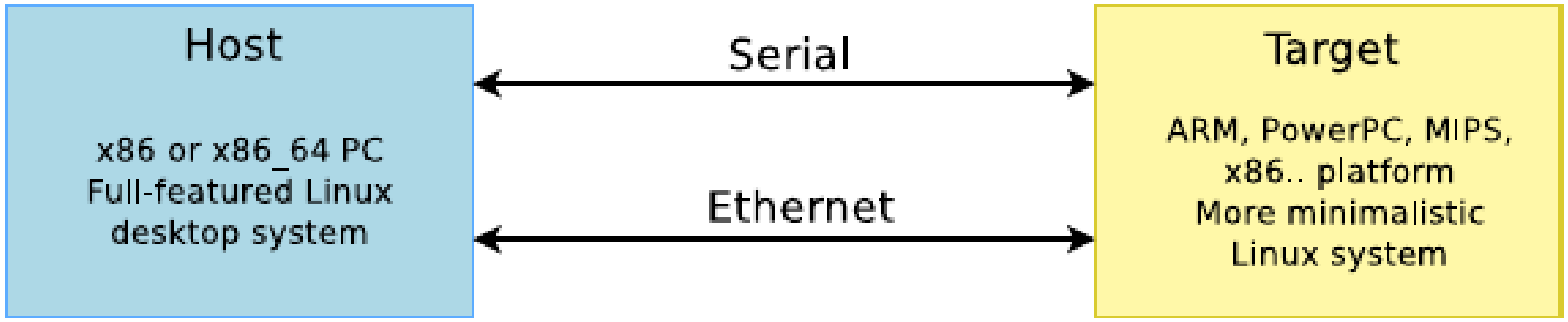
Embedded Linux From Scratch

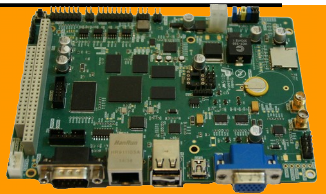
Now let's begin...



Setup the hardware connections

- Connect Apex-V210 to the Host





Compiling the u-boot bootloader

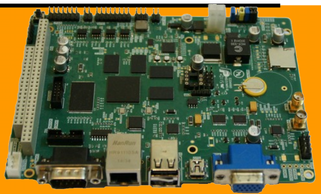
- Getting the u-boot sources from http://www.denx.de/
- Apply the Apex-V210 board support package patch
- Use the default configuration

```
make apex-v210_config
```

- Cross-compiling:

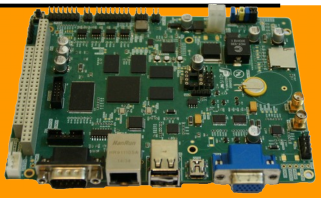
```
make
```

- Result: u-boot bootloader image `u-boot.bin`



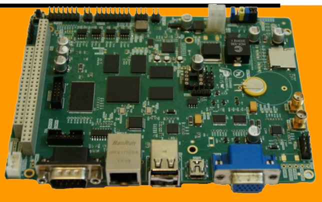
Compiling the Linux kernel

- Getting the Linux sources from http://kernel.org
- Apply the Apex-V210 board support package patch
- Adding settings specific to the Apex-V210 embedded system:
  ```
  make menuconfig
  ```
- You may simply use the default configuration
  ```
  make apex-v210_defconfig
  ```
- Cross-compiling:
  ```
  make
  ```
- Result: compressed kernel image `arch/arm/boot/zImage`



Compiling busybox

- Getting the sources from http://busybox.net
- Configuring BusyBox:
  `make menuconfig`
  Choosing to build a statically, Cross-compiled executable.
- Compiling busybox:
  `make`
- Pre-installing busybox (in the `_install/` subdirectory):
  `make install`
- Make an empty /dev directory in the installation subdirectory
- Result: a small executable implementing all the commands that we need!



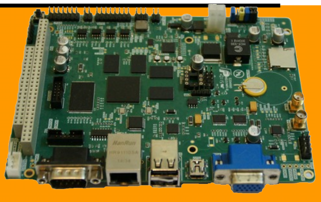
Build the root filesystem image

Use yaffs-utils package from
https://code.google.com/p/yaffs2utils/

- Compile it!
  - Result: mkyaffs2 binary to make yaffs images
- Use it to build the image

```
./mkyaffs2 --yaffs-ecclayout --all-root <rootfs_dir> <image>
```

  - Result: root file system image



Mounting virtual filesystems

Making `/proc` and `/sys` available
(required by several command line tools such as `ps`)

- Mounting `/proc`:

```
mount -t proc none /proc
```

- Mounting `/sys`: (Not used!)

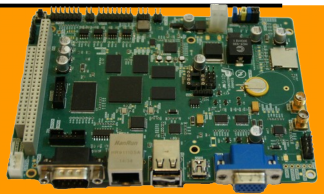
```
mount -t sysfs none /sys
```

Filesystem type

Raw device
or filesystem image
In the case of virtual
filesystems, any string is fine

Mount point

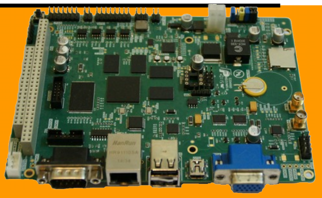
SAMANE SANAT TAHA



/etc/inittab file for busybox init

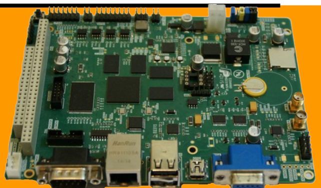
Creating the `/etc/inittab` file required by busybox `init`
Getting an example from busybox documentation
(not from the GNU/Linux host... missing features!)

```
# This is run first script
::sysinit:/etc/init.d/rcS
# Start an "askfirst" shell on the console
::askfirst:-/bin/sh
# Stuff to do when restarting the init process
::restart:/sbin/init
# Stuff to do before rebooting
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
```



Setting up networking

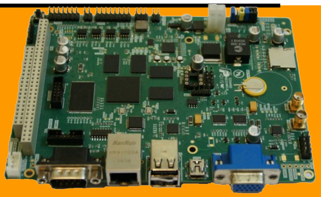
- Adding TCP/IP and network card driver to the kernel
- Bringing up the network interface:
  `ifconfig eth0 192.168.2.20`
- Testing networking:
  `ping -c 3 192.168.2.20`
  `-c 3:` useful when `[Ctrl][C]` doesn't work (missing tty settings)



Starting up a http server

- Copying HTML pages on `/www` (for example)
- Creating CGI scripts in `/www/cgi-bin/`
- Starting the busybox http server:

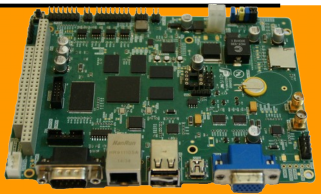
```
/usr/sbin/httpd -h /www/ &
```



/etc/init.d/rcS startup script

```
#!/bin/sh
mount -t proc none /proc
ifconfig eth0 192.168.2.20
/usr/sbin/httpd -h /www/ &
/bin/sh
```

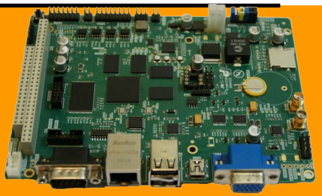
See how simple this can be!



/etc/init.d/rcS common mistakes

- Do not forget `#!/bin/sh` at the beginning of shell scripts! Without the leading `#!` characters, the Linux kernel has no way to know it is a shell script and will try to execute it as a binary file!
- In our example, do not forget to start a shell at the end of the script. Otherwise, execution will just stop without letting you type new commands!
- Do not forget to get it execution permission

```
chmod +x /etc/init.d/rcS
```



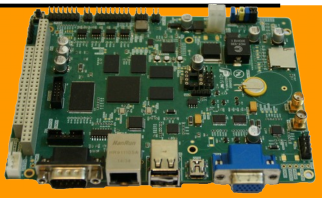
A simplistic CGI script

/www/cgi-bin/uptime:

```
#!/bin/sh
echo "Content-type: text/html"
echo ""
echo "<html><header></header><body>"
echo "<h1>Uptime information</h1>"
echo "Your Apex-V210 has been running
for:<pre><font color=Blue>"
echo `uptime`
echo "</font></pre></u>"
echo "</body></html>"
```

- Do not forget to make it executable

```
chmod +x /www/cgi-bin/uptime
```

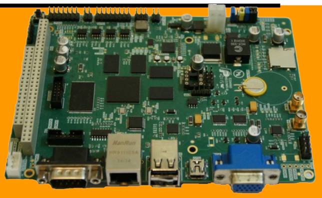


Limitations

A few minor limitations

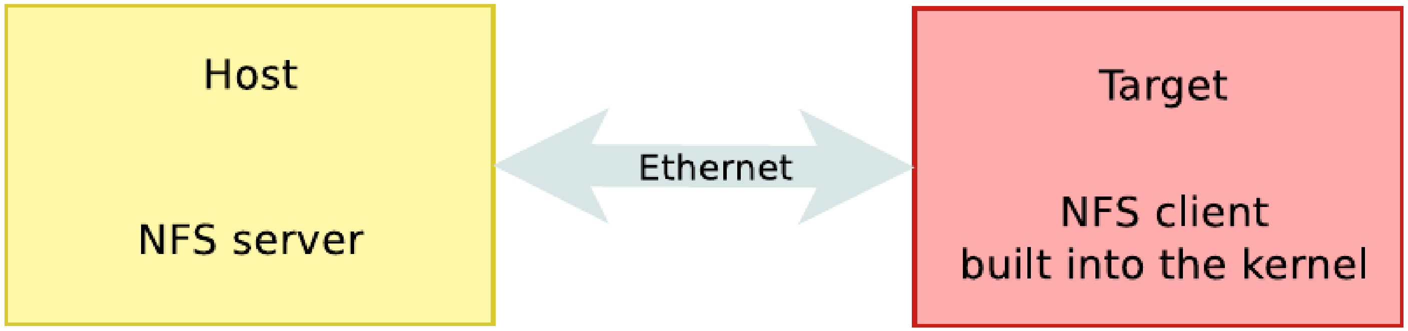
- CGI scripts: can't implement non-trivial scripts
  Need to code in C to support posting and URL parsing.
- System specific software: can't be part of busybox.
  Need more C executables. As a consequence, need to include the uClibc library and compile the executables with shared library support.

They are easy and cheap to overcome!



Real-world embedded system development

- May need to have more tools on your embedded device (Qt, sqlite, ...)
- Need to have shared libraries to save space for repetitive tasks
- Need to transfer kernel and root filesystem images to the target. An efficient way is to make the target boot on a NFS exported directory on the GNU/Linux host.
- Many more to do!



Thanks