Hands on with embedded Linux using zero hardware

By

Rajesh Sola

CDAC-ACTS,Pune

v2016.03

License

- This content is released under

  Creative Commons Attribution Share Alike 3.0

  https://creativecommons.org/licenses/by-sa/3.0/

  https://creativecommons.org/licenses/by-sa/3.0/legalcode

  Under this license terms you are free

  - to copy, distribute, display, and perform the work
  - to make derivative works
  - to make commercial use of the work
- Under the following conditions
  - **Attribution:**. You must give the original author credit.
  - **Share Alike:**. If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions**

Disclaimer

- These slides are not meant for beginners as a sole reference in quick mode.
- You may use this content as a checklist once you undergone basic course of embedded Linux or please refer suggested resources along with these if you are comfortable with self learning or explore more on listed concepts, techniques.
- The documented steps are verified under specified versions only,some tuning may be required with other versions
- Certain steps are less detailed in current version, planning to elaborate in further versions.
- Please report any corrections, enhancements,additions towards this content for improvements in further versions

Objectives

- Understanding cross development, cross toolchain
- Using various tools under a typical toolchain
- Building Linux Kernel,Applications for target architecture
- ARM Versatile Express family board as a reference platform
- Preparing file system for target platform
- Emulating built kernel, applications under QEMU
- Building and working with u-boot – sdcard,network approach
- Creating and linking with libraries – static, dynamic
- Writing simple modules for target kernel – external, internal
- Adding system calls(ARM specific)

Contents

Package Dump – checklist

- Linaro toolchain from https://releases.linaro.org/14.09/components/toolchain/binaries/

  gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_linux.tar.xz
- Kernel source from https://www.kernel.org/pub/linux/kernel/v3.x/

  linux-4.1.8.tar.xz
- Qemu source from http://wiki.qemu.org/download/

  qemu-2.0.0.tar.bz2
- Prebuilt rootfs from http://downloads.yoctoproject.org/releases/yocto/yocto-2.0/machines/qemu/qemuarm/

  core-image-minimal-qemuarm.tar.bz2 (or) core-image-minimal-qemuarm.ext4
- U-boot source code from ftp://ftp.denx.de/pub/u-boot/

  u-boot-2016.01.tar.bz2

Setting up QEMU

- Extract source code and switch into

  tar -jxvf qemu-2.0.0.tar.bz2

  cd qemu-2.0.0

- Configure, build and install

  ./configure –target-list=arm-softmmu,arm-linux-user \

  --enable-sdl –prefix=/opt/qemu-2.0

  make

  make install

- Update path to Qemu binaries

  export PATH=/opt/qemu-2.0/bin:$PATH

  #you may add above line to ~/.bash_profile or ~/.bashrc

Setting up toolchain

- Linaro toolchain

  tar -xvf gcc-linaro-arm-linux-gnueabihf-4.9-2014.09_linux.tar.xz -C /opt

  export PATH=/opt/gcc-linaro-linux-gnueabihf-4.9-2014.09_linux/bin:$PATH

  #you may add above line to ~/.bash_profile or ~/.bashrc for further use

- Simple check

  arm-linux-gnueabihf-gcc #should work

Building Kernel

- Extract kernel source and switch into, call it as KSRC

  tar -jxvf linux-4.1.8.tar.bz2

  cd linux-4.1.8 #KSRC now onwards

- Configuring kernel

  make mrproper

  make ARCH=arm vexpress_defconfig

  make ARCH=arm menuconfig

  #change local version under general setup during menuconfig

  #or copy tested configuration file as .config under KSRC

- Building

  make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
  zImage modules dtbs

  #skipping modules_install for time being

Building kernel

- Copy the following files to a temp dir for quick boot

  KSRC/arch/arm/boot/zImage

  KSRC/arch/arm/boot/dts/vexpress-v2p-ca9.dtb

- Refer Device tree for dummies from free-electrons.com

  for better understanding of Flattened Device Tree(FDT) concepts.

Preparing rootfs

- Preparing rootfs image from prebuilt contents

```
qemu-img create -f raw rootfs.img 64M
mkfs.ext4 rootfs.img
mount -o loop,rw,sync rootfs.img /mnt/image
tar -jxvf core-image-minimal-qemuarm.tar.bz2 -C /mnt/image
umount /mnt/image
#or use provided rootfs initially
#or download core-image-minimal-qemuarm.ext4 from same link and rename as
#rootfs.img, but this has very less free space left out
```

Copying files to rootfs

- Copying files to home dir rootfs

  mount -o loop,rw,sync rootfs.img /mnt/image

  cp <source-files> /mnt/image/home/root

  eg:- cp test.out /mnt/image/home/root

  umount /mnt/image

- Rebuild kernel for initrd support,upto 64MB size

  make ARCH=arm menuconfig

  Device Drivers → Block Devices →

  (*) RAM Block device support

  (16) Default number of RAM disks

  (65536) Default RAM disk size

Tuning for dynamic libs

```
mount -o loop,rw,sync rootfs.img /mnt/image

mkdir /mnt/image/lib/hardfp

cp /opt/gcc-linaro-linux-gnueabihf-4.9-2014.09_linux/arm-linux-gnueabihf/libc/lib/ld-linux-armhf.so.3  /mnt/image/lib/hardfp/

cp /opt/gcc-linaro-linux-gnueabihf-4.9-2014.09_linux/arm-linux-gnueabihf/libc/lib/arm-linux-gnueabihf/libc-2.19-2014.04.so  /mnt/image/lib/hardfp/

vi /mnt/image/etc/ld.so.conf      #add the line "/lib/hardfp"

umount /mnt/image

#or use provided dynrootfs.img initially
```

- Rebuild kernel to prevent read only mounting for large rootfs

```
make ARCH=arm menuconfig

General Setup → Enable the block layer

                Support for large(2TB+) block devices and files
```

- Run "ldconfig" once in target to configure and update added libraries

```
ldconfig      (or)

ldconfig -n /lib/hardfp
```

Quick boot

- Checklist

  zImage

  vexpress-v2p-ca9.dtb

  rootfs.img

- Booting with sdcard

```
qemu-system-arm -M vexpress-a9 -m 1024 -serial stdio \
-kernel zImage -dtb vexpress-v2p-ca9.dtb -sd rootfs.img \
-append “console=ttyAMA0 root=/dev/mmcblk0 rw”
```

- Initrd approach

```
qemu-system-arm -M vexpress-a9 -m 1024 -serial stdio \
-kernel zImage -dtb vexpress-v2p-ca9.dtb -initrd rootfs.img \
-append “console=ttyAMA0 root=/dev/ram0 rw”
```

Post boot

- Try the following commands in booted system

  uname -r

  uname -v

  cat /proc/cpuinfo

  lsmod

  cat /proc/modules

  cat /proc/kallsyms

Building u-boot

- Extract and switch into

  tar -jxvf u-boot-2016.01.tar.bz2

  cd u-boot-2016.01

- Configure and build

  make ARCH=arm vexpress_ca9x4_defconfig

  make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-

  cp tools/mkimage /usr/local/bin

  #copy generated “u-boot” to tempdir

- Preparing kernel image for u-boot format

  make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \

  uImage LOADADDR=0x60008000

U-Boot approach using SD card

- Prepare SD card image

```
qemu-img create -f raw sdcard.img 128M
mkfs.vfat sdcard.img
mkdir /mnt/sdcard
mount -o loop,rw,sync sdcard.img /mnt/sdcard
#copy uImage, vexpress-v2p-ca9.dtb, rootfs.img to /mnt/sdcard
umount /mnt/sdcard
```

- Boot from u-boot using SD card image

```
qemu-system-arm -M vexpress-a9 -m 1024 -kernel u-boot \
                                        -serial stdio -sd sdcard.img
#stop auto boot or wait for u-boot prompt
```

U-Boot approach using SD card

- Enter the following commands in u-boot prompt

```
mmcinfo

fatls mmc 0:0

fatload mmc 0:0 0x82000000 rootfs.img       #note down size

fatload mmc 0:0 0x80600000 uImage

fatload mmc 0:0 0x80400000 vexpress-v2p-ca9.dtb

setenv bootargs 'console=ttyAMA0 root=/dev/ram0 rw
rootfstype=ext4 initrd=0x82000000,8388608'

bootm 0x80600000  -  0x80400000
```

U-Boot approach using tftp

- Preparing QEMU with network support

```
mkdir /dev/net                          #skip if exists already
mknod /dev/net/tun c 10 200             #skip if exists already
modprobe tun
#copy qemu-ifup,qemu-ifdown under /etc
#modify ETH0IPADDR, GATEWAY,BROADCAST in /etc/qemu-ifup
chmod +x /etc/qmu-ifup /etc/qemu-ifdown
qemu-system-arm -M vexpress-a9 -m 1024 -kernel u-boot \
    -serial stdio -net nic -net tap,ifname=tap0
```

- Enable TFTP server in your host using tempdir as server root

  YaST → Network Services → TFTP Server in case of opensuse

U-Boot approach using tftp

- Network setup in u-boot

  setenv ipaddr 192.168.0.5

  setenv serverip 192.168.0.1 #ETH0IPADDR in /etc/qemu-ifup

  ping 192.168.0.1

- Loading & Booting via tftp

  tftp 0x82000000 rootfs.img

  tftp 0x80600000 uImage

  tftp 0x80400000 vexpress-v2p-ca9.dtb

  setenv bootargs 'console=ttyAMA0 root=/dev/ram0 rw rootfstype=ext4 initrd=0x82000000,8388608'

  bootm 0x80600000 - 0x80400000

Cross compiling sample code

- Simple Program

  arm-linux-gnueabihf-gcc hello.c -c

  arm-linux-gnueabihf-gcc hello.o -o h.out

- Multifile example

  arm-linux-gnueabihf-gcc test.c -c

  arm-linux-gnueabihf-gcc sum.c -c

  arm-linux-gnueabihf-gcc sqr.c -c

  arm-linux-gnueabihf-gcc test.o sum.o sqr.o -o all.out

- Copy h.out, all.out as described in previous slide and test them

Static Linking

- Creating static library

  arm-linux-gnueabihf-gcc sum.c -c

  arm-linux-gnueabihf-gcc sqr.c -c

  arm-linux-gnueabihf-ar rc libsample.a sum.o sqr.o

- Linking with static library

  arm-linux-gnueabihf-gcc test.c -c

  arm-linux-gnueabihf-gcc -L. test.o -lsample -o p.out

  arm-linux-gnueabihf-gcc -L. test.o -lsample -o s.out -static

- Testing

  Copy p.out, s.out to rootfs and try executing them

Dynamic linking

- Creating static library

  arm-linux-gnueabihf-gcc sum.c -c

  arm-linux-gnueabihf-gcc sqr.c -c

  arm-linux-gnueabihf-gcc -shared sum.o sqr.o -o libsample.so

- Linking with static library

  arm-linux-gnueabihf-gcc test.c -c

  arm-linux-gnueabihf-gcc -L. test.o -lsample -o d.out

- Testing

  #copy d.out to home dir of rootfs

  #copy libsample.so to ~/mylibs of rootfs

  LD_LIBRARY_PATH=~/mylibs ./d.out

  #Adding ~/mylibs to /etc/ld.so.conf eliminates the need of

  LD_LIBRARY_PATH

Analysis

- Analysis

  file s.out p.out d.out

  ls -sh s.out p.out d.out

  arm-linux-gnueabihf-readelf -h s.out p.out d.out

  arm-linux-gnueabihf-ldd --root /mnt/image s.out p.out d.out

  arm-linux-gnueabihf-strings s.out p.out d.out

Writing simple modules

- Makefile

```
obj-m += hello.o
```

- hello.c

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
static int __init hello_init(void)
{
        printk("Hello World..welcome\n");
        return 0;
}
static void __exit hello_exit(void)
{
        printk("Bye,Leaving the world\n");
}
module_init(hello_init);
module_exit(hello_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your name");
MODULE_DESCRIPTION("A Hello, World Module");
```

Building and testing

- Building

```
make -C <path-of-ksrc> M=$PWD modules \
    ARCH=arm CROSS_COMPI:LE=arm-linux-gnueabihf-
```

- Copy generated hello.ko to rootfs
- Better Makefile, Now simple run “make”

```
obj-m += hello.o
all:
    make -C <path-of-ksrc> M=$PWD modules \
        ARCH=arm CROSS_COMPI:LE=arm-linux-gnueabihf-
clean:
    make -C <path-of-ksrc> M=$PWD clean \
        ARCH=arm CROSS_COMPI:LE=arm-linux-gnueabihf-
```

Testing the module

- Testing in host

  file hello.ko

  arm-linux-gnueabihf-objdump -d hello.ko

  arm-linux-gnueabihf-readelf -h hello.ko

  modinfo hello.ko

- Testing in target

  dmesg -c

  insmod hello.ko

  dmesg

  lsmod

  cat /proc/modules

  rmmod hello

  dmesg

Module parameters,dependency

- Module parameters

  int ndevices=0;

  module_param(ndevices,int,S_IRUGO);

  insmod simple.ko ndevices=3

- Exporting symbols (eg:- from simple.c)

  EXPORT_SYMBOL_GPL(ndevices);

  EXPORT_SYMBOL_GPL(sayHello);

- Accessing symbols from other modules(eg:- complex.c)

  extern int ndevices;

  extern void sayHello();

- GPL symbols are accessible from GPL modules only

  MODULE_LICENSE("GPL");

Internal modules - dynamic

- drivers/char/mtest/ ==> simple.c, complex.c
- drivers/char/mtest/Makefile

  obj-m += simple.o complex.o
- drivers/char/Makefile

  obj-m += dtest/
- Rebuild the kernel with modules_install

  make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- \

  modules_install INSTALL_MOD_PATH=/mnt/image
- Testing

  ls /lib/modules/$(uname -r)/kernel/drivers/mtest

  modprobe complex #loads simple also due to dependency

  lsmod

  cat /proc/modules

  modprobe -r complex

  modprobe -r simple

Internal modules - static

- drivers/char/mtest/ ==> simple.c, complex.c
- drivers/char/mtest/Makefile

  obj-y += simple.o complex.o
- drivers/char/Makefile

  obj-y += mtest/
- Rebuild the kernel
- Testing in host

  arm-linux-gnueabihf-nm KSRC/vmlinux #try with objdump also
- Testing in target

  ls /lib/modules/$(uname -r)/kernel/drivers/char

  cat /proc/kallsyms #check for init methods of modules

  dmesg #check for printk output in static modules

Adding Kconfig entries

- drivers/char/mtest/Kconfig

```
#menu "My Modules"
config SIMPLE
tristate "A simple module"
default m
help
  This is a simple module
config COMPLEX
tristate "A complex module"
depends on SIMPLE
default m
help
  This is a sample module dependending on simple
#endmenu
```

- drivers/char/Kconfig

```
source “drivers/char/mtest/Kconfig”
```

Testing Kconfig entries

- make ARCH=arm menuconfig

  drivers → char → custom modules
- verify .config
- driver/char/mtest/Makefile

  obj-$(CONFIG_SIMPLE)+=simple.o

  obj-$(CONFIG_COMPLEX)+=complex.o
- driver/char/Makefile

  obj-y += mtest/

Adding system calls

- Changes to kernel – adding entry

  arch/arm/include/uapi/asm/unistd.h

  arch/arm/kernel/calls.S

  arch/arm/include/asm/unistd.h

  include/linux/syscalls.h

- Changes to kernel – defining system call

  KSRC/kernel/mysyscall.c

```
asmlinkage long sys_testcall(void)
{
    printk("This is my system call\n");
}
```

  KSRC/kernel/Makefile

```
    obj-y += mysyscalls.o
```

Building & Testing system calls

- Rebuild the kernel
- Verifying system call under new kernel

```
arm-linux-gnueabihf-nm KSRC/vmlinux | grep sys_testcall #in host
cat /proc/kallsyms | grep sys_testcall                  #in target
```

- Testing system call using C code

```
#include<unistd.h>
#define SYS_mycall 388
int main()
{
    syscall(SYS_mycall);
    return 0;
}
```

- arm-linux-gnueabihf-gcc systest.c -o s1.out

Testing system calls

- Assembly code – asmtest.s

```
mov r7,#388
mov r0,#25
mov r1,#35
SWI 0
mov r7,#1
mov r0,#5
SWI 0
```

- arm-linux-gnueabihf-as asmtest.c -o asmtest.o
- arm-linux-gnueabihf-ld asmtest.o -o s2.out

```
#Copy the binaries s1.out,s2.out to rootfs and test them
```

Appendix - Environment Variables

- PATH

  Holds list of directories holding external commands

  Updating PATH(QEMU binaries as an example):-

  export PATH=/opt/qemu-2.0/bin:$PATH

  Can add above line to ~/.bash_profile, ~/.bashrc

  Or write the settings in a script and invoke script using source command

- LD_LIBRARY_PATH

  List of directories holding dependent dynamic libraries

  export LD_LIBRARY_PATH=~/mylib:$LD_LIBRARY_PATH

  (or) add ~/mylibs to /etc/ld.so.conf and run ldconfig once or reboot

Appendix - Appending dtb file to kernel image

- Configure kernel
- Append kernel image and dtb file

  cd arch/arm/boot

  cat zImage vexpress-v2p-ca9.dtb > zImage-dtb
- Convert to u-boot format

  mkimage -A arm -O linux -C none -T kernel -a 0x60008000 \

  -e 0x60008000 -n 'Linux-4.1.2-vexpress' -d zImage-dtb uImage-dtb
- Copy zImage-dtb, uImage-dtb as desired
- Reference:- Device Tree for Dummies, Free Electrons

Appendix – ELF Format, Tools

- Executable and Linkable Format, applicable for relocatable object files, executables, shared object files etc.

  https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

  http://elinux.org/Executable_and_Linkable_Format_(ELF)

- Tools for ELF files

  file

  arm-linux-gnueabihf-nm

  arm-linux-gnueabihf-objdump # -d, -t, -S

  arm-linux-gnueabihf-readelf # -h

  arm-linux-gnueabihf-ldd # --root /mnt/image

  arm-linux-gnueabihf-strings

  arm-linux-gnueabihf-strip

References, Acknowledgments

- Building Embedded Linux Systems, Karim Yaghmour, O'Reilly Media
- Device Tree for Dummies, Thomas Petazzoni, Free Electrons
- How to Cross Compile the Linux Kernel with Device Tree Support, Rajesh Sola, Open Source For You Magazine(EFY Group), September 2014
- Thanks to Mr.Babu Krishnamurthy for his valuable inputs on working with Beagle Board xM, Beagle Bone Black.
- Thanks to the following presentation template

  http://templates.libreoffice.org/template-center/university-course-material-template
- Thanks to the community,contributors of opensuse,libreoffice and other open source components listed in beginning.

  https://www.opensuse.org/

  https://www.libreoffice.org/

Thank You

rajeshsola@gmail.com