CSEE 4840

Embedded System Design
How Linux Boots on the SoCKit Board

Stephen A. Edwards
Columbia University

2015

This explains how Linux boots over the network on the SoCKit boards in our lab. It also explains how the server is configured.

Contents

1 Overview

The boot process proceeds as follows

1. The first stage bootloader loads from onboard serial flash, resets the processor, configures the memory controller, and loads the second stage bootloader.
2. The second stage bootloader sets the board's Ethernet address, uses DHCP to obtain the board's IP address and the address of the TFTP/NFS server.
3. The second stage bootloader uses TFTP to download *pxelinux.cfg/default*, which contains paths to kernel images, FPGA files, and device tree files.
4. The user selects one of these groups by typing on the console.
5. The second stage bootloader uses TFTP to download the kernel image, the device tree, and FPGA bitstream file, programs the FGPA, and passes control to the Linux kernel.
6. The Linux kernel enumerates and initializes devices, mounts the root directory with NFS, then runs */sbin/init*, which runs the initialization scripts in */etc/init.d/*, eventually running a *getty* instance, which prompts for a username and password.

2 The QSPI Flash Chip

We store the U-Boot bootloader in the QSPI onboard flash chip and instruct the HPS to boot from it so each board does not need an SD card.

2.1 Flashing the QSPI

There are a few ways to flash the QSPI, but the easiest is to copy the two bootloader image files onto an SD card, boot into U-Boot (e.g., from an SD card) and halt the automatic booting process by pressing a key during the initial countdown. This should give the U-Boot command line.

At the U-Boot command line, verify the contents of the FAT filesystem on the SD, verify that you can access the QSPI flash chip, load the two bootloader files into memory, then erase and flash the QSPI chip with the two files. This is done as shown below:

```
fatls mmc 0:1
sf probe
fatload mmc 0:1 0x2000000 preloader-mkpimage.bin
fatload mmc 0:1 0x2100000 u-boot-01.img
sf erase 0x0000 0x10000
sf write 0x2000000 0x0000 0x10000
sf erase 0x60000 0x40000
sf write 0x2100000 0x60000 0x40000
```

This writes the 64K first-stage bootloader to address 0 and the 256K second-stage bootloader to address 0x60000.

The first-stage bootloader is the same for every board, but we customize the second-stage bootloader by giving each a unique MAC address.

We want the HPS to boot from onboard QSPI flash memory. Three BOOTSEL jumpers on the front of the board control the source form which the HPS initially boots.

	HPS Boot Source Selection BOOTSEL[2:0]	Function
→	111	3.3V SPI Flash Memory
	101	3.3V SD/MMC Flash Card
	001	FPGA

We want the HPS to be able to configure the FPGA with data downloaded from the server. A tiny dipswitch on the back of the board, SW6 (SW6.1 – SW6.5), sets MSEL, which controls where the FPGA loads its configuration data.

	FPGA Configuration Modes MSEL[4:0]	Function
→	10000	Passive Serial (from HPS), fast POR
	10001	Passive Serial (from HPS), standard POR
	10010	Active Serial (from onboard EPCQ flash), fast POR
	10011	Active Serial (from onboard EPCQ flash), standard POR

3 U-Boot

U-Boot is a boot loader with excellent support for ARM processors as well as others. Altera distributes a version with support for the Cyclone V on the SoCKit board.

3.1 Patching and Compiling

I modified two files in the U-Boot source tree to enable PXE and configure the FPGA.

common/cmd_pxe.c Moved the call to *label_localboot()* in the *pxe* command to run *localcmd* after loading files, instead of before.

include/configs/socfpga_cyclone5.h Enabled PXE support by adding *#define* directives and completely changed the initial variable environment to use PXE and configure the FPGA.

Compilation of the second-stage bootloader images proceeds as follows:

```
/opt/altera/13.1/embedded/embedded_command_shell.sh
export ARCH=arm
export CROSS_COMPILE=arm-none-eabi-
export LOADADDR=0x8000

cd u-boot-socfpga

for num in 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21
do
  echo "#define BOARDNUM \"$num\"" > include/configs/boardnum.h
  make mrproper
  ./MAKEALL socfpga_cyclone5
  mv u-boot.img u-boot-$num.img
done
```

This generates files of the form *u-boot-02.img*, which need to be flashed to the QSPI.

The initial U-Boot environment (for the first board) is set to

```
bootargs=console=ttyS0,57600
ethaddr=1c:76:ca:48:40:01
loadaddr=0x3000000
pxefile_addr_r=0x3000000
kernel_addr_r=0x7fc0
fdt_addr_r=0x100
ramdisk_addr_r=0x2000000
fpga=0
fpgadatasize=6AEBD0
updatebootargs=setenv bootargs ${bootargs} ip=${ipaddr}:${serverip}
localcmd=fpga load ${fpga} ${ramdisk_addr_r} ${fpgadatasize} ;
   run updatebootargs ; bootm ${kernel_addr_r} - ${fdt_addr_r}
```

and the boot command is

```
dhcp ; pxe get ; pxe boot
```

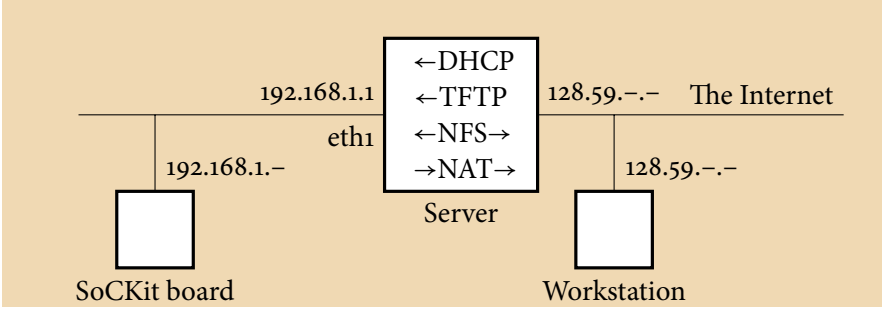
The first three bytes of each board's Ethernet are Terasic's registered prefix; the next two are the number of the class; and the last byte indicates the board number (the one thing that differs among boards) in BCD: 0x01, …, 0x09, 0x10, …, 0x19, 0x20, ….

When it gets control, the second-stage bootloader

1. Uses DHCP to get its IP address and TFTP server address.
2. Uses TFTP according to the PXE standard to fetch the *pxelinux.cfg/default* file, which specifies the pathname for the kernel, the *.rbf* file (taken from PXE's facility for the initial RAMdisk image, *initrd*), and the device tree file (PXE's fdt).
3. Prints a menu of options (PXE labels); the user selects one.
4. Fetches the kernel image, the FPGA bitstream, and the device tree blob using TFTP.
5. Configures the FPGA using the just-downloaded bitstream file.
6. Adds information about the IP address of the board and the the NFS server (assumed to be the same as the TFTP server) to the Linux boot command line.
7. Boots the Linux kernel, informing it about the serial console, its IP adress, the NFS server address, and the address of the device tree blob.

4 Network Configuration

We place a server between a local network with all the SoCKit boards and the rest of the Internet. Among other things, it acts as a network address translator that allows the SoCKit boards to reach the Internet, but not vice versa.



4.1 Static Network Configuration

Under CentOS, */etc/sysconfig/network-scripts/ifcfg-eth1*,

```
DEVICE=eth1
BOOTPROTO=none
HWADDR=00:04:23:9f:30:16
ONBOOT=yes
HOTPLUG=no
TYPE=Ethernet
NETMASK=255.255.255.0
IPADDR=192.168.1.1
```

Under Ubuntu, */etc/network/interfaces*,

```
auto eth1
iface eth1 inet static
   address 192.168.1.1
   network 192.168.1.0
   netmask 255.255.255.0
   broadcast 192.168.1.255
   gateway 192.168.1.1
```

4.2 Port Forwarding (NAT)

NAT port forwarding (out eth2) using iptables:

```
/sbin/iptables -t nat -A POSTROUTING -o eth2 -j MASQUERADE
```

NAT port forwarding using the *ufw* package:

In */etc/default/ufw*,

```
DEFAULT_FORWARD_POLICY="ACCEPT"
```

In */etc/ufw/sysctl.conf*,

```
net/ipv4/ip_forward=1
```

Add to */etc/ufw/before.rules* right after the initial comments

```
# nat Table rules
*nat
:POSTROUTING ACCEPT [0:0]

# Forward traffic from eth1 through eth0.
-A POSTROUTING -s 192.168.1.0/24 -o eth0 -j MASQUERADE

# don't delete the 'COMMIT' line or these nat table rules won't be processed
COMMIT
```

5 DHCP Server Configuration

Our U-Boot setup queries DHCP to get various network parameters, including the board's IP address and the address of the TFTP server where it downloads the PXE configuration file.

Under CentOS, install the *dhcpd* package. Run DHCP on only one of the interfaces by editing */etc/sysconfig/dhcpd*:

```
DHCPDARGS=eth1
```

Under Ubuntu, install the *isc-dhcp-server* package. Run DHCP on only one of the interfaces by editing */etc/default/isc-dhcp-server*:

```
INTERFACES="eth1"
```

The main configuration file is */etc/dhcpd.conf* under CentOS; */etc/dhcpd/dhcpd.conf* under Ubuntu. For the 192.168.1.– network, this defines the router as 192.168.1.1 (the server), DNS servers (Columbia's), and will automatically assign network addresses between 20 and 99 to devices that request them.

We assign fixed addresses to each SoCKit board based on its MAC address and tell each the address of the TFTP server.

```
subnet 192.168.1.0 netmask 255.255.255.0 {
  option routers 192.168.1.1;
  option subnet-mask 255.255.255.0;
  option domain-name-servers 128.59.1.3, 128.59.1.4;
  range 192.168.1.20 192.168.1.99;

  group {                    # SoCKit boards
    next-server 192.168.1.1; # TFTP server
    host board01 {
      hardware ethernet 1c:76:ca:48:40:01;
      fixed-address 192.168.1.101;
    }
    host board02 {
      hardware ethernet 1c:76:ca:48:40:02;
      fixed-address 192.168.1.102;
    }
    # ... etc.
  }
}
```

6 TFTP Server Configuration

TFTP, as its name suggests, is a very simple Internet file transfer protocol used mostly for system bootstrapping applications such as ours. It is insecure (e.g., no usernames or passwords), so should only be run on protected networks, and only has provisions for downloading and uploading files; it cannot list directories, for example.

Under CentOS, you need the *xinetd* and *tftp-server* packages; under Ubuntu, *xinetd* and *tftpd*.

Under CentOS,

```
/sbin/chkconfig --level 345 xinetd on
/sbin/chkconfig --level 345 tftp on
```

Under Ubuntu,

```
update-rc.d xinted enable
```

Configure it with */etc/xinitd.d/tftp*:

```
service tftp
{
  socket_type           = dgram
  protocol              = udp
  wait                  = yes
  user                  = nobody
  server                = /usr/sbin/in.tftpd
  server_args           = -s /export
  disable               = no
  bind                  = 192.168.1.1
  flags                 = IPv4
}
```

This puts the root of what files it can serve at */export*, runs it as the *nobody* user, and binds it only to the interface associated with 192.168.1.1.

7 PXE Configuration

Unlike the other servies, PXE is not a daemon by itself but is instead built on DHCP and TFTP. For our purposes, it amounts to a single configuration file accessible from TFTP as *pxelinux.cfg/default* (in the */export* directory if TFTP is so-configured). The file consists of a list of labels, each of which corresponds to a particular running environment (kernel image, FPGA configuration, device tree blob, and root directory). Here is a representative example. Note that

```
prompt 1
default lab2

label lab2
  menu label lab2
  kernel 2014_groups/lab2/uImage
  initrd 2014_groups/lab2/soc_system.rbf
  append console=ttyS0,57600 root=/dev/nfs rw
      nfsroot=192.168.1.1:/export/2014_groups/lab2/root
  fdt 2014_groups/lab2/socfpga.dtb
  localboot 1

label lab3
  menu label lab3
  kernel 2014_groups/lab3/uImage
  initrd 2014_groups/lab3/soc_system.rbf
  append console=ttyS0,57600 root=/dev/nfs rw
     nfsroot=192.168.1.1:/export/2014_groups/lab3/root
  fdt 2014_groups/lab3/socfpga.dtb
  localboot 1
```

8 NFS Server Configuration

NFS is an Internet protocol for remotely mounting filesystems. It allows the SoCKit boards to store their root directories (i.e., the entire filesystem) on the server.

Under CentOS, install the *nfs-utils* and *nfs-utils-lib* packages and

```
chkconfig nfs on
service rpcbind start
service nfs start
```

Under Ubuntu, install the *nfs-kernel-server* package and

```
service nfs-kernel-server start
```

The configuration file for NFS is */etc/exports*. We export the root filesystems to the SoCKit boards and allow the workstations to also mount the relevant directory so files, such as the FPGA configuration file, can be updated.

```
# Export root filesystems to the SoCKit boards
/export/2014_groups 192.168.1.*(rw,no_root_squash,no_subtree_check)

# Export to the workstations
/export/2014_groups micro1.ilab.columbia.edu(rw)
/export/2014_groups micro2.ilab.columbia.edu(rw)
/export/2014_groups micro3.ilab.columbia.edu(rw)
# ... etc.
```

Glossary

BCD Binary-Coded Decimal: A way to represent decimal (base 10) numbers in binary that uses four bits per decimal digit.

DHCP Dynamic Host Configuration Protocol: used to send the IP address and other network configuration details to the SoCKit board

DNS Domain Name Service: The Internet protocol that translates names like "google.com" into IP addresses like 192.168.1.5.

EPCQ Erasable Programmable Configurable Quad: a four-bit-wide serial flash device that can hold the FPGA's configuration bitstream. Not used when we have the bootloader configure the FPGA with data from the server.

FAT File Allocation Table: The MS-DOS filesystem typically used to format SD cards.

Flash Nonvolatile memory that can be reprogrammed slowly. Used to hold the bootloader code.

FPGA Field-Programmable Gate Array: e.g., the main Cyclone V chip on the board

HPS Hard Processor System: the ARM processor and associated peripherals on the Cyclone V FPGA

IP address Internet Protocol address: a 32-bit address typically written in dotted decimal form, e.g., 192.168.1.12

MAC address Media Access Control address: a 48-bit Ethernet address

NFS Network File System: a protocol for mounting a remote disk. Used to enable the Linux root filesystem on the server to appear on the board.

POR Power On Reset

PXE Preboot eXecution Environment: a protocol that provides information to the board about which kernel to download, etc. Uses DHCP and TFTP.

QSPI Quad Serial Peripheral Interface: a variant of the simple, synchronous serial peripheral interface protocol that uses four data lines (hence "quad"). The interface between the HPS and an onboard flash chip.

SD Secure Digital: a flash card format. The SoCKit board accepts mini-SD cards.

TFTP Trivial File Transfer Protocol: an insecure, very simple way to transfer files over the Internet, e.g., the Linux kernel image.

Bibliography

Boot the Arrow SoCKit from QSPI.
http://www.rocketboards.org/foswiki/Documentation/BootTheArrowSoCKitFromQSPI

SoCKit User Manual.
http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=816&PartNo=4