Pragmatux

Documentation

Quick-Start Guide for the Inforce Computing IFC6410

This is a brief guide to creating a Pragmatux Workstation development environment and Pragmatux Device filesystem images for the Inforce Computing IFC6410 platform.

There is a community of Snapdragon™ users at http://mydragonboard.org, with forums suitable for hardware questions. Pragmatux software questions are best directed to the Pragmatux mailing list.

Setup a Workstation

The development environment for creating programs, packages, and filesystem images is a desktop, laptop, or server running the Wheezy version of the Debian Linux distribution. Pragmatux refers to that environment as the *workstation*, as opposed to the embedded board, the *device*.

The workstation need not be a physical machine; it is common to use a virtual machine or a chroot within another operating system; however, the workstation must be of the *amd64* architecture.

1. Install Debian Wheezy

   Begin by installing the Debian Wheezy operating system on any amd64 architecture machine. The following commands are to be run on that machine, unless otherwise stated.

2. Add package sources for the Pragmatux SDK

   Configure the workstation to use the Pragmatux SDK package sources by downloading and installing the package apt-sources-ptux-sdk-ifc6410.

   To install:

   ```
   # dpkg -i apt-sources-ptux-sdk-ifc6410.deb
   ```

3. Have the package management system download the contents of the package sources

   ```
   # apt-get update
   ```

   If you see errors about access to files related to the armhf architecture, make sure the packages sources configured for your workstation offer packages for the armhf architecture; a minority do not. When in doubt, try http://ftp.us.debian.org/debian.

4. Install the SDK for the IFC6410

```
# apt-get install --no-install-recommends ptux-sdk-ifc6410
```

This will install tens of packages containing the tools and code libraries essential for cross-development targeting the IFC6410.

In the future, your installation of the SDK can be upgraded by repeating steps 3 and 4.

```
# apt-get update
# apt-get install --no-install-recommends ptux-sdk-ifc6410
```

Create device filesystem images

1. Compose a filesystem using the default set of packages

```
$ mkdir wrk
$ cd wrk
$ fakeroot mkos-ifc6410
```

This will create two files, *boot.img* containing the kernel and initramfs, and *ptux.img* containing the root filesystem. They will be written to the IFC6410's eMMC storage in subsequent steps.

When invoked without arguments, `mkos-ifc6410` composes a filesystem for the device using a predefined list of packages. Via arguments, this list can be modified and extended to define a customized filesystem for each project using the platform.

Packages can be added to and removed from a running device using the package management tools included in the base image.

Create images containing limited-distribution content

The firmware required to run several of the IFC6410's peripherals including the Wi-Fi radio is available only to licensees of Inforce Computing. In the common case, a license is conferred with the purchase of a board. Through the purchase, you also gain access to limited-distribution documents and software via the Inforce TechWeb. Among the downloads available with a TechWeb account is a key which enables mkos-ifc6410 to pull from an access-controlled repository containing packages with the limited-distribution firmware and other binaries.

1. To the workstation, download the package containing the key and install it via dpkg

```
# dpkg -i key-inforce-ifc6410-licensee-example.deb
```

2. Compose a filesystem in the same manner as above

```
# fakeroot mkos-ifc6410
```

`mkos-ifc6410` will automatically detect the installed key and add the limited-distribution content to the filesystems it builds.

Connect a serial terminal

The IFC6410's 3-pin RS-232 header is connected to the processor's primary serial line, to which bootloader and kernel messages are directed, and on which a command shell is started. The serial line parameters are 115200-8-N-1 (115200 bits per second, 8 data bits, no parity bit, and 1 stop bit) at RS-232 signaling levels. The header's location and pinout is given in the IFC6410's hardware reference manual, found on the Inforce TechWeb. You will likely need to make a custom cable to go from this header to a DCE-pinout DE9 connector for connection to the workstation.

Use whatever serial terminal software and hardware you prefer; the instructions below demonstrate using a USB-to-serial adapter on a Debian Wheezy workstation and GNU screen as a terminal emulator.

1. Connect a suitable cable between the device's RS-232 header and the USB-to-serial adapter.
2. Start screen as a terminal emulator.

   ```
   # screen /dev/ttyUSB0 115200
   ```

   For help, type *ctrl-a* followed by *?*; and to quit, type *ctrl-a k*.

Write device filesystem images to eMMC storage

1. Put the device into *fastboot* mode

   Jumper pins 30 and 26 of the Expansion Connector (pulling GPIO6 low) as shown in the photo in this blog post on the myDragonBoard.com community site and power the IFC6410 on. The bootloader will pause in fastboot mode, listening to commands from a fastboot client. Serial output from the bootloader will indicate it's listening for fastboot commands.

   ```
   Android Bootloader - UART_DM Initialized!!!
   [0] welcome to lk

   [10] platform_init()
   [10] target_init()
   [130] USB init ept @ 0x88f4e000
   [170] fastboot_init()
   [170] udc_start()
   [300] -- reset --
   [300] -- portchange --
   [470] fastboot: processing commands
   ```

2. Connect the device USB device port to the workstation, and verify connectivity

   After making the USB connection and giving workstation a few seconds to enumerate the new device, run the fastboot client to verify connectivity to the bootloader. If the client is able to talk to the bootloader, it will print the serial number of the DragonBoard.

```
# fastboot devices
b1732aaf        fastboot
```

3. Write the kernel and initramfs to the *boot* partition

   Using the fastboot client on the workstation, write the *boot.img* image containing the kernel and initramfs to the partition named *boot*.

```
# fastboot flash boot boot.img
sending 'boot' (5474 KB)...
OKAY [  1.032s]
writing 'boot'...
OKAY [  1.093s]
finished. total time: 2.125s
```

4. Write the root filesystem image to the *userdata* partition

   Using the fastboot client on the workstation, write the *ptux.img* image containing the root filesystem to the partition named *userdata*.

```
# fastboot flash userdata ptux.img
sending 'userdata' (85797 KB)...
OKAY [  8.391s]
writing 'userdata'...
OKAY [ 27.812s]
finished. total time: 36.203s
```

Now the operating system has been installed on the device and it is ready to be booted for the first time.

Boot the device for the first time

Reset the device by cycling its power. Within seconds, the newly installed kernel should boot and write considerable output to the serial console. The first time the operating system starts, it will go through a minute-long installation procedure and automatically reboot.

```
Setting up ncurses-base (5.7+20100313-5em1) ...
Setting up sensible-utils (0.0.4em1) ...
Setting up dpkg-autoconfigure (1.5~dev2) ...
Setting up devnodes-ptux (1.3) ...
Setting up sshd-run (1.0) ...
Setting up linux-db8060a (3.0.21-12374-gcae2925-1) ...
[....]
```

The installation generates much debugging, informational, and warning output due to the unusual state of the system at installation time and the inconsistent use of logging levels by several of the packages being installed. While ignoring warnings is normally a bad practice, novice users can safely ignore warnings in this output when installing a default configuration unless the system fails to behave as expected after the after the first boot.

On the second and all subsequent boots, a login prompt leading to a command shell is offered on the serial port. The only account which exists following a basic installation is *root* with the password *password*.

```
Pragmatux 3.0 ptux ttyHSL0

ptux login:
```

Configure Wi-Fi

Pragmatux uses the network manager connman, which is well suited to embedded systems. In addition to flexibly handling multiple media types (Wi-Fi, wired Ethernet, Bluetooth, tethering, etc.), connman is designed to be programatically controlled via its DBus interface, which enables tight integration with the embedded device's custom software. This differs from other network managers, most of which are designed assuming the system is administered by an operator who can edit configuration files.

A command-line client, `connmanctl`, provides an interface for configuring the network during testing and development. The always-excellent Arch Linux wiki has a page detailing the operation of `connman` and `connmanctl`.

The steps to connect to a WPA-secured network follow.

1. Launch the command line client `connmanctl` in interactive mode

```
# connmanctl
connmanctl>
```

2. Use `help` to see the set of available commands

```
connmanctl> help
state
technologies
enable          <technology>|offline
disable         <technology>|offline
tether          <technology> on|off
[....]
```

3. Enable the technology `wifi`

```
connmanctl> enable wifi
Enabled wifi
```

4. Force `wifi` to scan for access points

```
connmanctl> scan wifi
Scan completed for wifi
```

5. See the available network services

```
connmanctl> services
*A  Wired        ethernet_1a51ce4589e2_cable
    foobar       wifi_00037f20a274_6b7339406574_managed_psk
    willowroad   wifi_00037f20a274_77696c639407726f6164_managed_psk
    :)           wifi_00037f20a274_3b08_managed_psk
```

6. Turn on the interactive password-handler

```
connmanctl> agent on
Agent registered
```

7. Associate to the desired access point

```
connmanctl> connect wifi_00037f20a274_6b7339406574_managed_psk
Agent RequestInput wifi_00037f20a274_6b7339406574_managed_psk
  Passphrase = [ Type=psk, Requirement=mandatory ]
Passphrase?
Connected wifi_00037f20a274_6b7339406574_managed_psk
```

8. See that the interface is active and has an address

```
root@device:~# ip addr show dev wlan0
14: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 state UP
    link/ether 00:03:7f:20:a2:74 brd ff:ff:ff:ff:ff:ff
    inet 10.100.0.110/24 brd 10.100.0.255 scope global wlan0
    inet6 fe80::213:7fff:fe23:b274/64 scope link
       valid_lft forever preferred_lft forever
```