snickerdoodle

LINUX DEVELOPMENT

SEPTEMBER 9, 2016

How to Read this Document

This document makes extensive use of links, references and notices in the page margins to detail additional information that can be useful while following the guide.

WARNING A warning notice indicates a potential hazard. If care is not taken to adhere to the safety precautions, damage may be done to snickerdoodle.

Warnings and cautions will be clearly visible in either the body of the text or in the margin and must be paid close attention while following the guided steps.

CAUTION A caution indication denotes a process that requires special attention. If the caution is not exercised and the process not adhered to, failure may result and/or potential damage to snickerdoodle.

Warning, caution and informational notices, such as this one, may also be found in the margin.

Keywords

Keywords and important terms are shown in *italicized* type. Additional important information can be found in the margins of text with superscript notation[1].

Navigation of menus and directories are shown using ***bold italicized*** type. Any hierarchical navigation is shown using an arrow to denote a ***Parent*** → ***child*** relationship.

`Teletype` text is used to highlight inputs, variables and system files within the host environment.

[1] Margin notes, such as this one, reference the body content and highlight technical details or references for further information.

Download Sources

U-Boot Source

U-Boot is the second stage boot loader used to load the the Linux kernel and root filesystem. For U-Boot to load the kernel, the kernel image needs to be wrapped with a U-Boot header. U-Boot is built with a `mkimage` utility which is used to wrap the kernel image. `mkimage` is also used to wrap file system images with U-Boot headers for loading. To build Linux images, U-Boot source needs to be downloaded and compiled to provide the `mkimage` utility binary.

To download the U-Boot source needed to compile snickerdoodle Linux images, the following `git` command can be executed from the command line:

```
git clone https://github.com/krtkl/snickerdoodle-u-boot.git
```

Linux Source

The source required to configure and compile the Linux kernel for snickerdoodle can be downloaded using the following `git` command:

```
git clone https://github.com/krtkl/snickerdoodle-linux.git
```

Checkout Stable Branch

At the time of this document's publication, the current stable and supported kernel release for snickerdoodle is 3.14. The `sd-linux/3.14` branch of the snickerdoodle Linux kernel has been updated to include the necessary driver source to run the snickerdoodle peripherals (wireless, etc.). To switch the local repository to this branch, the following `checkout` command can be used:

```
git checkout sd-linux/3.14
```

snickerdoodle Default kernel Configuration

Before the kernel or any of the executables that are build

```
make ARCH=arm zynq_snickerdoodle_defconfig
```

Build U-Boot

Generate Build Script Utilities

U-Boot requires a utility that is normally built with the kernel. The device tree compiler (`dtc`) is used to compile device tree blobs from source and is called during the build process for U-Boot. Interestingly, building U-Boot requires a utility that is normally built with the kernel while the Linux kernel requires a utility that is normally built with U-Boot (`mkimage`). To avoid building a kernel image (can take quite a long time depending on host computer performance) just to access the `dtc` and still being required to recompile the kernel after U-Boot has been built, the utilities in the `scripts` directory (including `dtc`) can be compiled with the following invocation:

```
make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- scripts
```

After building the scripts within the Linux source repository, the path to the device tree compiler must be added to the `$PATH` variable so that it is available to `make` when building U-Boot. After navigating to ***snickerdoodle-linux*** → ***scripts*** → ***dtc***, the `$PATH` variable can be updated as shown below:

```
cd scripts/dtc
export PATH=$PATH:$(pwd)
```

Build U-Boot and `mkimage`

After the device tree compiler has been built, U-Boot can be built from the it's top-level directory by invoking the following command:

```
make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-
```

Just as with the device tree compiler, for `mkimage` to be callable from `make` during the kernel build process, it's parent directory must be added to `$PATH`. The `mkimage` executable is built in the `tools` directory within the U-Boot source di-

rectory. By navigating to ***snickerdoodle-u-boot*** → ***tools***, the $PATH variable can be appended with the location of the mkimage utility.

```
cd tools
export PATH=$PATH:$(pwd)
```

To test that mkimage has been made available to be called by the system (and to debug possible conflicting installations of mkimage), which can be used to output the parent directory of the mkimage binary:

```
/home/snickerdoodle/snickerdoodle-u-boot/tools/mkimage
```

The above output shows the mkimage executable located in the directory that was added to $PATH in the previous step.

Configure Linux

After the mkimage utility has been generated, the Linux kernel can be built for U-Boot. Before building, a configuration file (.config) must be produced to define the kernel build options. The default snickerdoodle configuration can be specified using make:

```
make ARCH=arm zynq_snickerdoodle_defconfig
```

If additional configuration options are needed, the configuration can be edited using a variety of interfaces. make can be called to initiate the configuration interface using the following command:

```
make ARCH=arm <config_interface>
```

Options for <config_interface> are:

config - Text-based interface

menuconfig - Text-based interface with hierarchical menus, radiolists and in-

> *Necessary libraries can be installed using sudo apt-get install commands along with the required libraries specified for the interface*

teractive assistance. This option allows incremental saving of changes. (ncurses must be installed: `libncurses5-dev`)

`nconfig` - Text-based menus (curses must be installed: `libcdk5-dev`)

`xconfig` - QT/X-windows interface (QT is required: `qt4-dev-tools`)

`gconfig` - Gtk/X-windows interface (GTK is required)

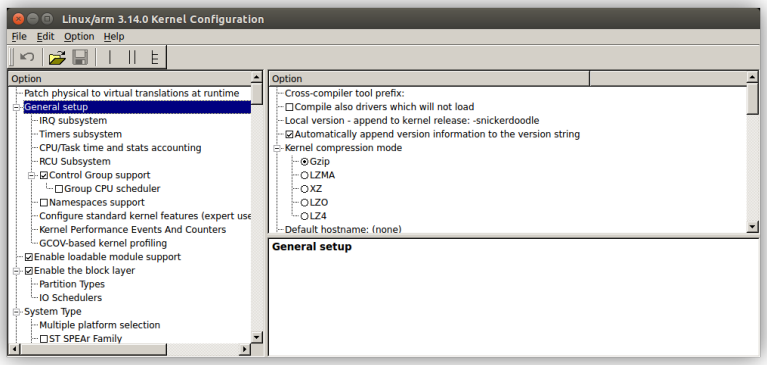

Figure 1: QT/X-Windows Configuration Interface (`xconfig`)

Figure **??** shows the QT based configuration interface that can be used to configure Linux build options.

Building Linux

The next and final step for building the Linux kernel is the build process itself. While building, the build components will be output to the console. A clean build can take quite a while. To build the Linux kernel with the necessary U-Boot header, the `uImage` argument should be specified. The following command will begin the build process:

```
make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-
    LOADADDR=0x8000 uImage
```

The build process will output messages for each compilation step. After the final build step is complete, a set of messages similar to the following will appear (note the output image path on the last line):

```
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-3.14.0-snickerdoodle-00004
Created:      Fri Dec 11 16:34:56 2015
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    4145568 Bytes = 4048.41 kB = 3.95 MB
Load Address: 00008000
Entry Point:  00008000
  Image arch/arm/boot/uImage is ready
```

Build and Install kernel Modules

The kernel modules, if any have been specified by the Linux configuration, can be built by invoking:

```
make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- modules
```

After building the kernel modules, they can be installed into a root file system by calling:

```
make ARCH=arm INSTALL_MOD_PATH=<path_to_rootfs> modules_install
```