GlobalLogic

A Hitachi Group Company

EDUCATION

Smart Start: Linux/Networking OpenWRT buildroot

Sergii Kudriavtsev

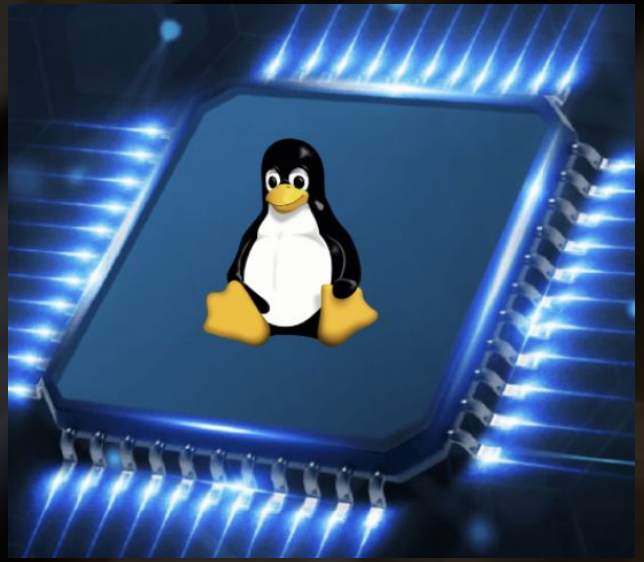
Agenda

- Introduction, HW archs
- Usage, application
- UCI (Configuration)
- Buildroot (Compilation)
- Packages Structure
- Licenses

Introduction, HW arch

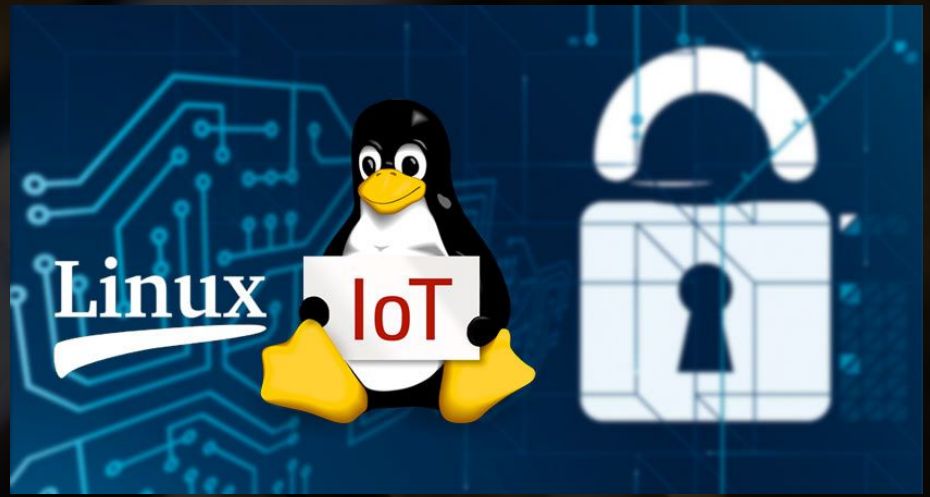
Introduction to Embedded Linux

What is Embedded Linux? - Embedded Linux utilizes the Linux Kernel, libraries and utilities in resource-constrained embedded systems. It's Open Source nature, flexibility and robustness make it a preferred choice in telecommunications, automotive, multimedia (set top boxes) and aerospace applications.



Introduction to Embedded Linux

IoT Role - As the Internet of Things (IoT) expands, embedded Linux plays a vital role in providing a customizable platform, ensuring interoperability, security and efficient resources utilization in diverse embedded applications.



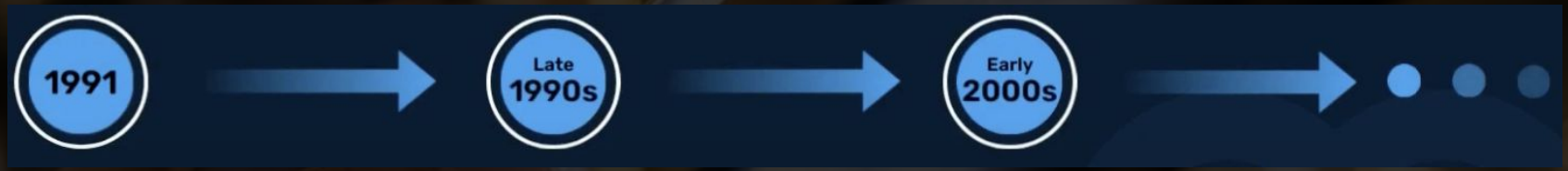
History of Embedded Linux

Linux Kernel Development Begins

The development of the Linux kernel lays the foundation for embedded Linux.

Growth in Embedded Linux Adoption

Embedded Linux experiences significant adoption across industries



Embedded Linux Emerges

Embedded Linux gains popularity for its open-source nature and customization capabilities

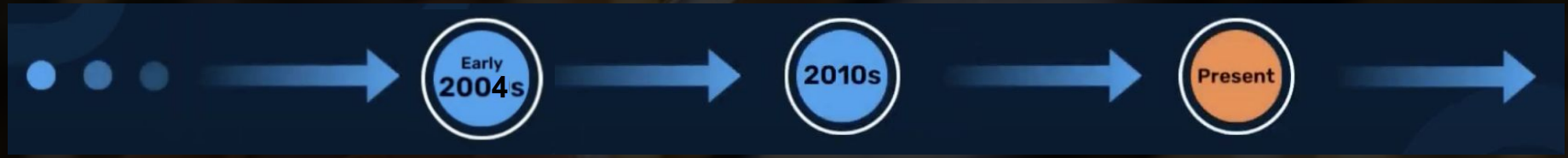
History of Embedded Linux

Inception of the OpenWRT project

Demonstrated the power of Linux in network-focused embedded devices

Continued Expansion in IoT and Edge Computing

Embedded Linux plays a crucial role in the expanding IoT and edge computing landscape



Linux Foundation and Yocto Project

The Linux Foundation and Yocto Project support the development and standardization of embedded Linux

Fundamental Concepts to Understand Embedded Linux

Understanding the Linux Kernel

Basics of Embedded Systems

Linux Distribution for Embedded Systems

Real-Time Systems

Bootloaders

Cross Compilation

Device Drivers

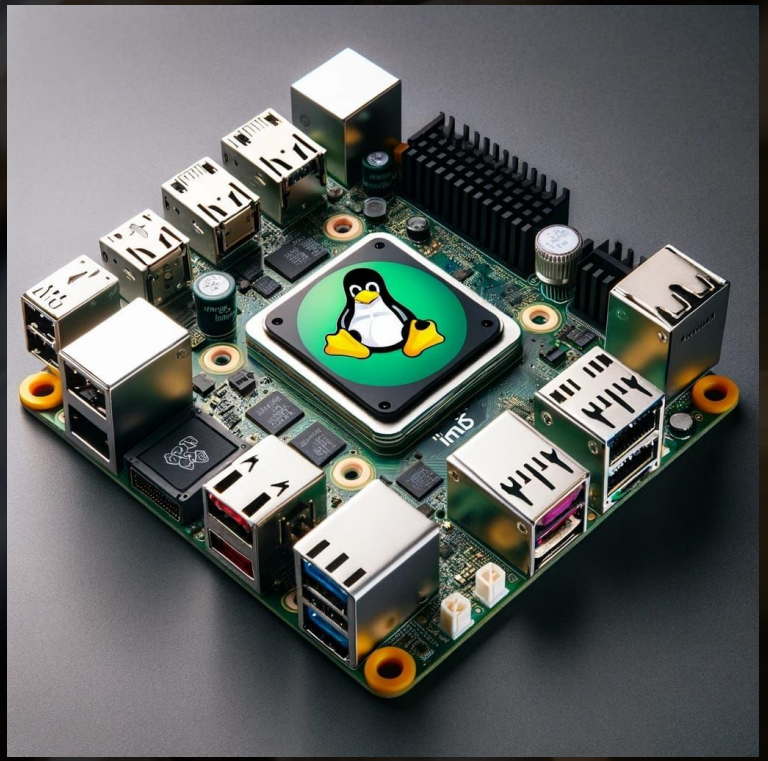
Architecture of Embedded Linux

Hardware Considerations

- Processor
- Memory
- Peripherals

Software Aspects

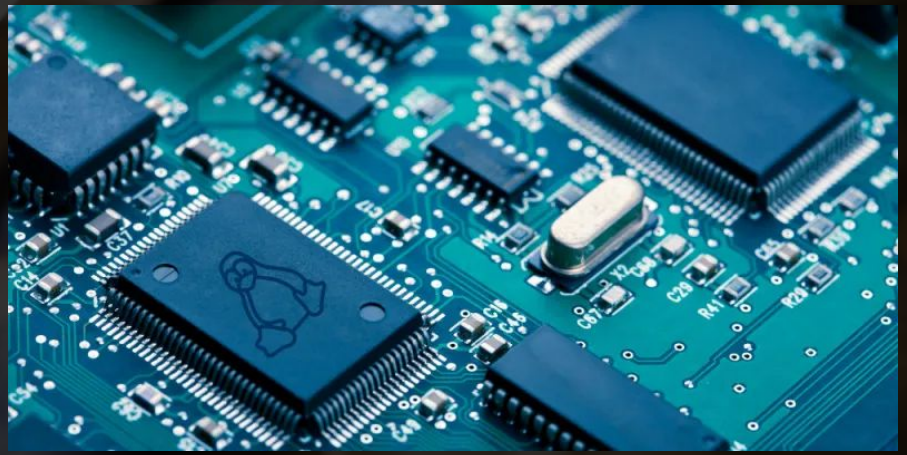
- Bootloader
- Linux Kernel
- Root File System (Rootfs)
- User Space Applications
- The Linux Kernel and Device Drivers



Building an Embedded Linux System

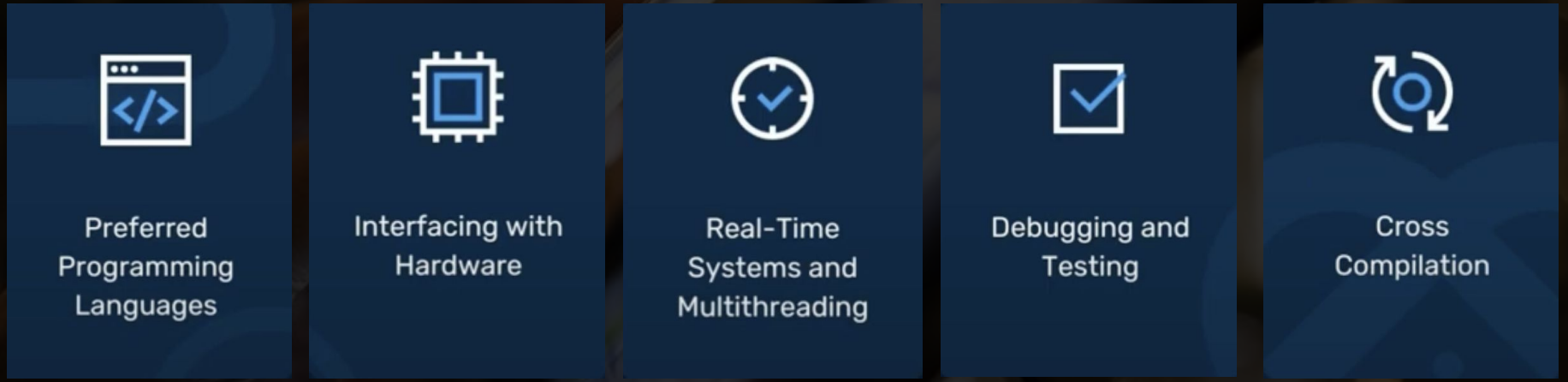
What you need to consider:

- Necessary Tools and Equipment
- Choosing the Right Linux Distribution
- Cross-Compilation and Configuration
- Flashing and Bootloading
- Testing and Debugging

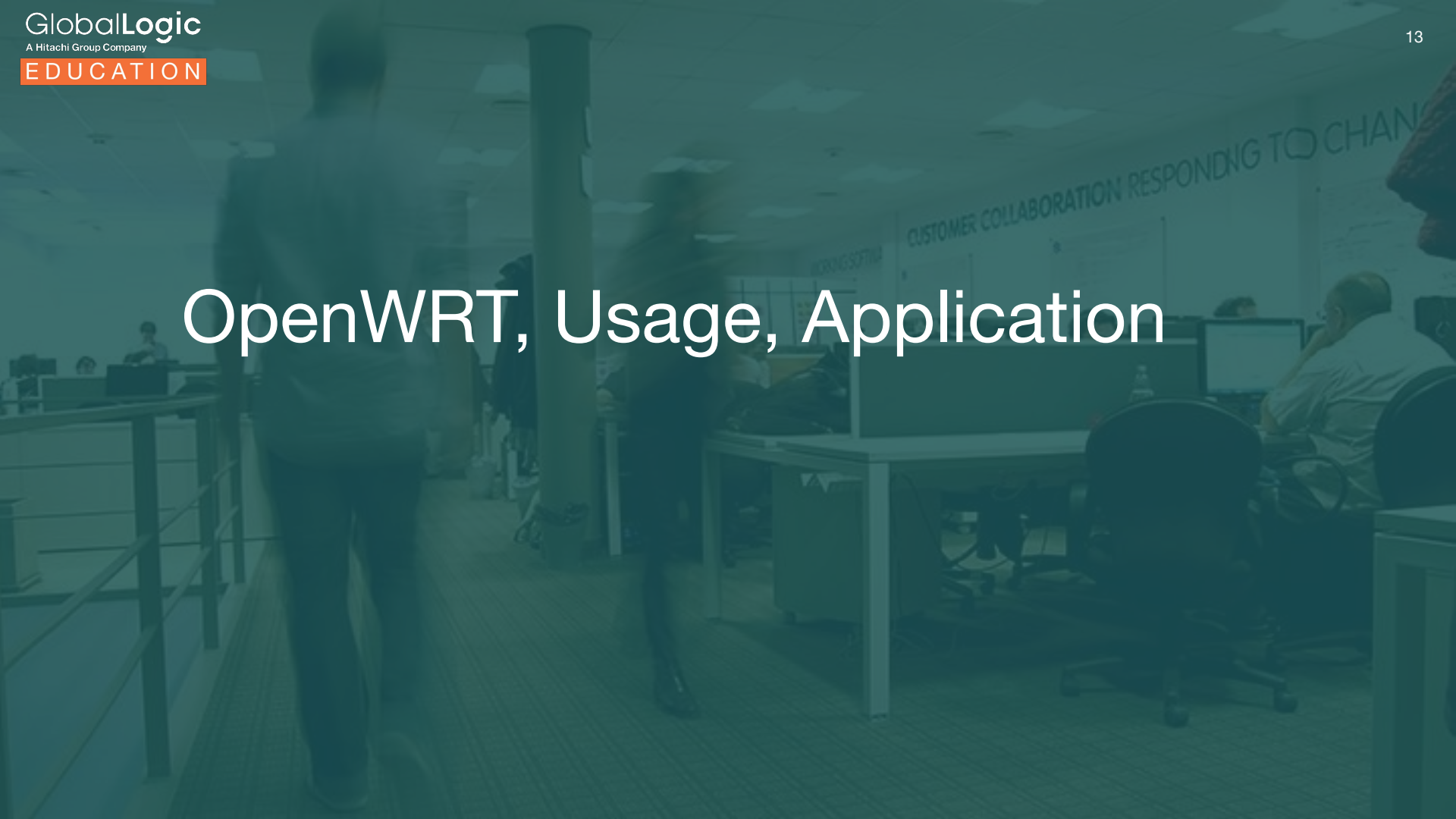


Programming in Embedded Linux

Programming for embedded Linux systems involves several unique aspects that distinguish it from programming for general-purpose computers. This is what you should take into account:



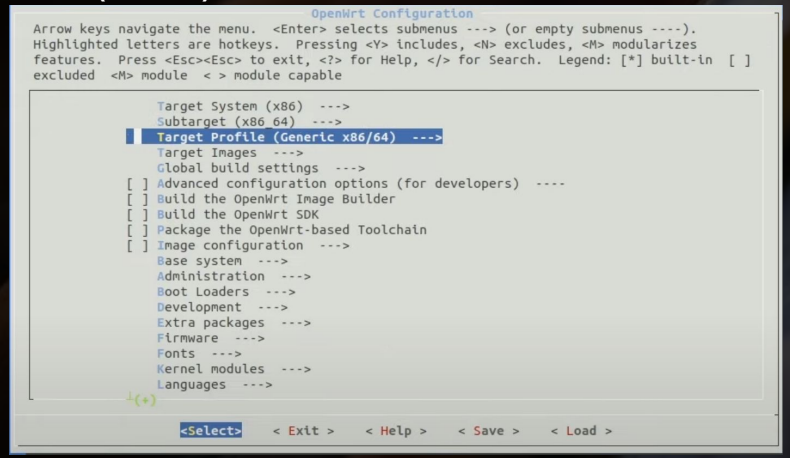
Advantages of Embedded Linux in IoT



OpenWRT, Usage, Application

What is OpenWRT?

- Open Source build system
- Linux Distribution
- Configuration using CLI or Web Interface (LuCi)



Why OpenWRT?

- To have full root access of the router
- Home routers are not up-to date with security patches
- OpenWRT Software components are kept up-to date
- Extend functionality of the router
- Strong community support

OpenWRT Installation

- Select HW (Router to be migrated to OpenWRT) https://openwrt.org/toh/start
- Download appropriate image
- Install image to selected router according to installation procedure described at HW Support page.
- Configure networking using CLI(UCI) or GUI(LuCi)
- Install auxiliary software via opkg tool.

UCI (Configuration)

OpenWRT Configuration

The abbreviation UCI stands for Unified Configuration Interface, and is a system to centralize the configuration of OpenWrt services.

Common principles:

- Configuration is split into several files located in the /etc/config/ directory.
- Each file relates roughly to the part of the system it configures.
- Configuration files can be edited via text editor or #uci tool
- Modifiable through various programming APIs (like Shell, Lua and C)
- Upon changing a UCI configuration file restart the services by an init.d call

OpenWRT Configuration files

File	Description
Basic	
/etc/config/dhcp	Dnsmasq and odhcpd settings: DNS, DHCP, DHCPv6
/etc/config/dropbear	SSH server options
/etc/config/firewall	NAT, packet filter, port forwarding, etc.
/etc/config/network	Switch, interface and route configuration: General, IPv4, IPv6, Routes, Rules, WAN, Aliases, Switches, VLAN, IPv4/IPv6 transitioning, Tunneling
/etc/config/system	Misc. system settings, NTP, RNG, Watchcat
/etc/config/wireless	Wireless settings and wifi network definition

UCI Configuration File syntax

```
package 'example'

config 'example' 'test'
        option  'string'      'some value'
        option  'boolean'     '1'
        list    'collection'  'first item'
        list    'collection'  'second item'
```

- The `config 'example' 'test'` statement defines the start of a section with the type `example` and the name `test`.
- The `option 'string' 'some value'` and `option 'boolean' '1'` lines define simple values within the section.
- `list` keyword an option with multiple values
- `option` and `list` statements is a convention to improve the readability
- default value is assumed if an option is absent and not required,

Valid UCI syntax:

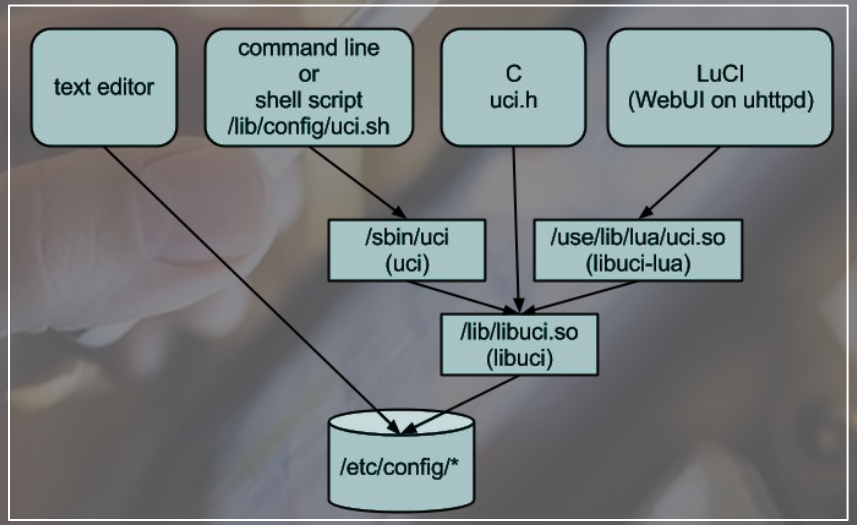
```
option  example   value
option  example  "value"
option 'example'  value
option 'example' "value"
option "example" 'value'
```

Invalid UCI syntax:

```
# missing quotes around the value
option  example   v_a l u-e
# unbalanced quotes
 option 'example" "value'
```

UCI configuration modification

- Direct config files modification using text EDITOR (vi, nano, e.t.c.)
- **uci** command line utility.
- Set of standard shell procedures (/lib/functions.sh), config_*, uci_*
- libuci (C library for the Unified Configuration Interface (UCI))



UCI command line utility

Usage:

```
# uci
Usage: uci [<options>] <command> [<arguments>]

Commands:
        commit     [<config>]
        add        <config> <section-type>
        show       [<config>[.<section>[.<option>]]]
        get        <config>.<section>[.<option>]
        set        <config>.<section>[.<option>]=<value>
        delete     <config>.<section>[.<option>]
```

Options:

```
-c <path>  set the search path for config files (default:
/etc/config)
-d <str>   set the delimiter for list values in uci show
-f <file>  use <file> as input instead of stdin
-m         when importing, merge data into an existing package
-n         name unnamed sections on export (default)
-N         don't name unnamed sections
-P <path>  add a search path for config change files and use
as default
-q         quiet mode (don't print error messages)
```

UCI data/object model

Elements:

- **config**: main configuration groups like **network**, **system**, **firewall**. Each configuration group has it's own file in **/etc/config**
- **sections**: config is divided into sections. A section can either be **named** or **unnamed**.
- **types**: a section can have a type.
- **options**: each section have some options where you set your configuration values
- **values**: value of option

Example of anonymous-name:

```
# uci show network
...
network.@switch[0]=switch
network.@switch[0].name='switch0'
network.@switch[0].reset='1'
network.@switch[0].enable_vlan='1'
...
```

Example of **autogenerated ID/CFGID**:

```
# uci show network.@switch[0]
network.cfg073777=switch
network.cfg073777.name='switch0'
network.cfg073777.reset='1'
network.cfg073777.enable_vlan='1'
```

Example of named config:

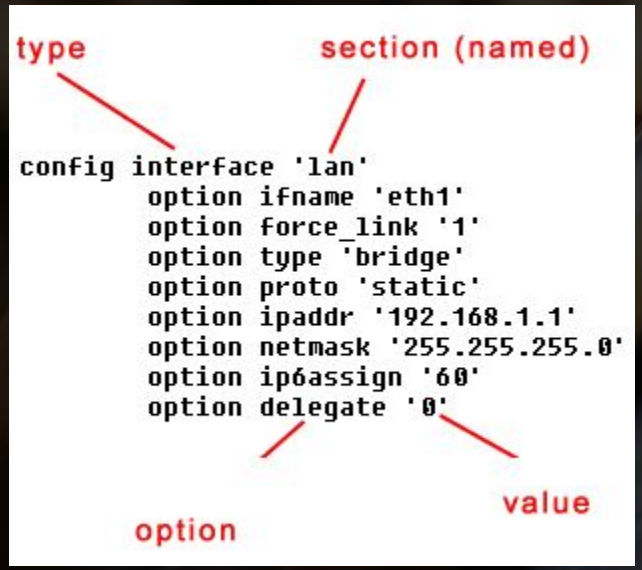
```
network.loopback=interface
network.loopback.ifname=lo
network.loopback.proto=static
network.loopback.ipaddr=127.0.0.1
network.loopback.netmask=255.0.0.0
network.loopback.ip6addr=::1/128
network.lan=interface
network.lan.ifname=eth0
network.lan.netmask=255.255.255.0
network.lan.ipaddr=192.168.0.5
network.lan.proto=static
network.lan.ipv6_proto=dhcp
network.lan.mtu=1500
network.lan.type=bridge
```

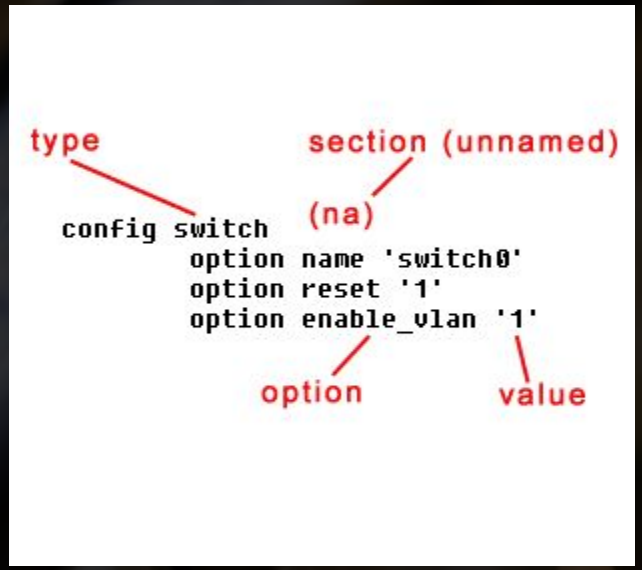
UCI config Different presentation

The same config section can be presented in different ways:

- Human-friendly: as presented in the config files or with the command “uci export <config>”
- Programmable: as presented with the command “uci show <config>”

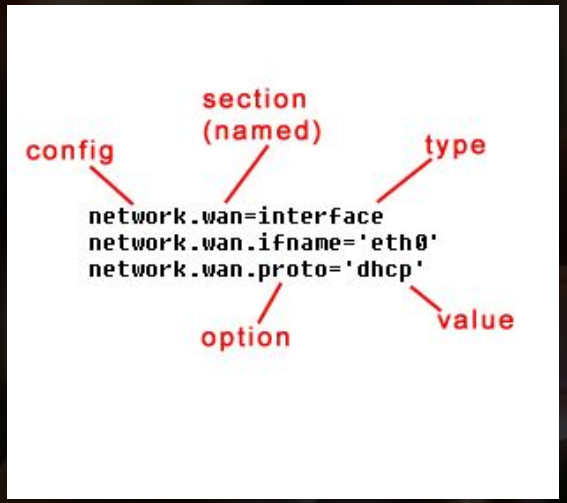
Human-friendly presentation:

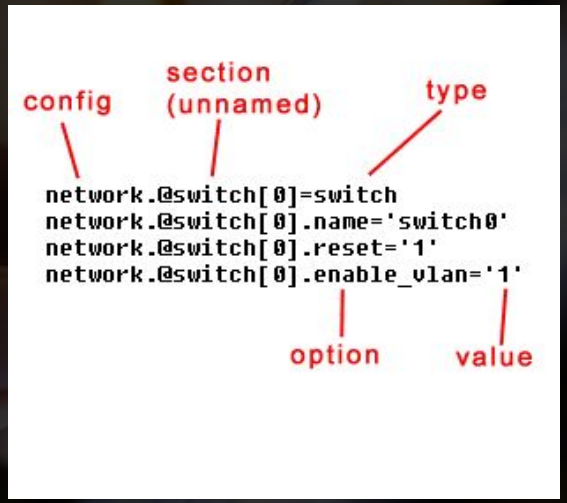


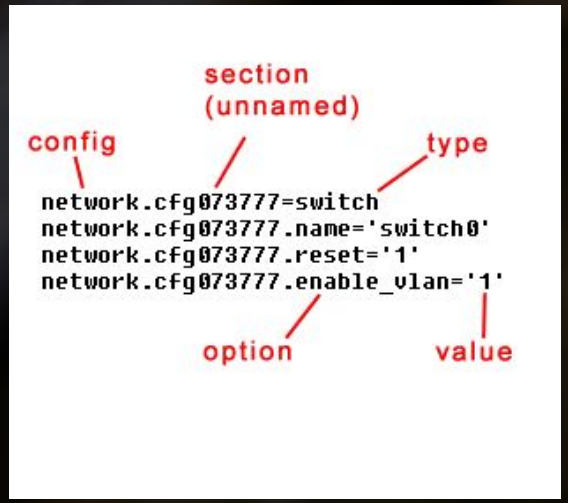


UCI config Different presentation

Programmable presentation:







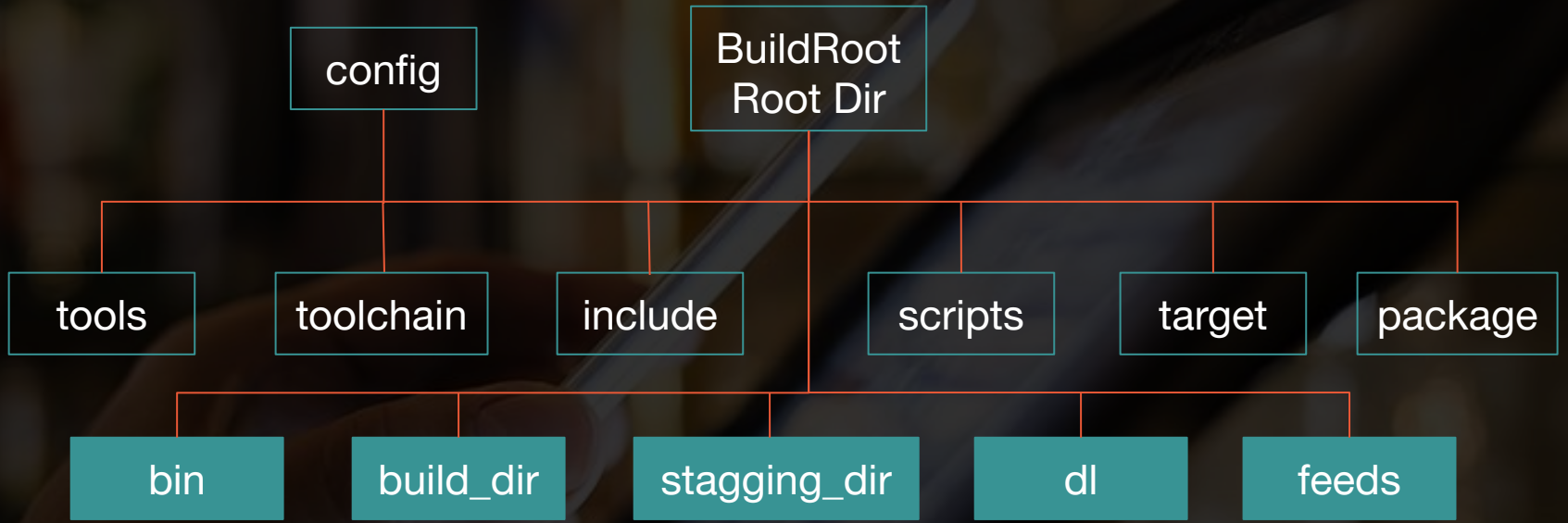
Buildroot (Compilation)

OpenWrt BuildRoot Environment

OpenWrt Buildroot environment is a collection of Makefiles, patches and scripts, which generates the cross-compilation toolchain, downloads Linux kernel, generates a root file system, manages 3rd party packages, etc.

The collection of Makefiles determines the version of Linux kernel to download, and the version of the package tarball to download and compiled in to the image.

OpenWrt BuildRoot source tree



OpenWrt BuildRoot General source structure

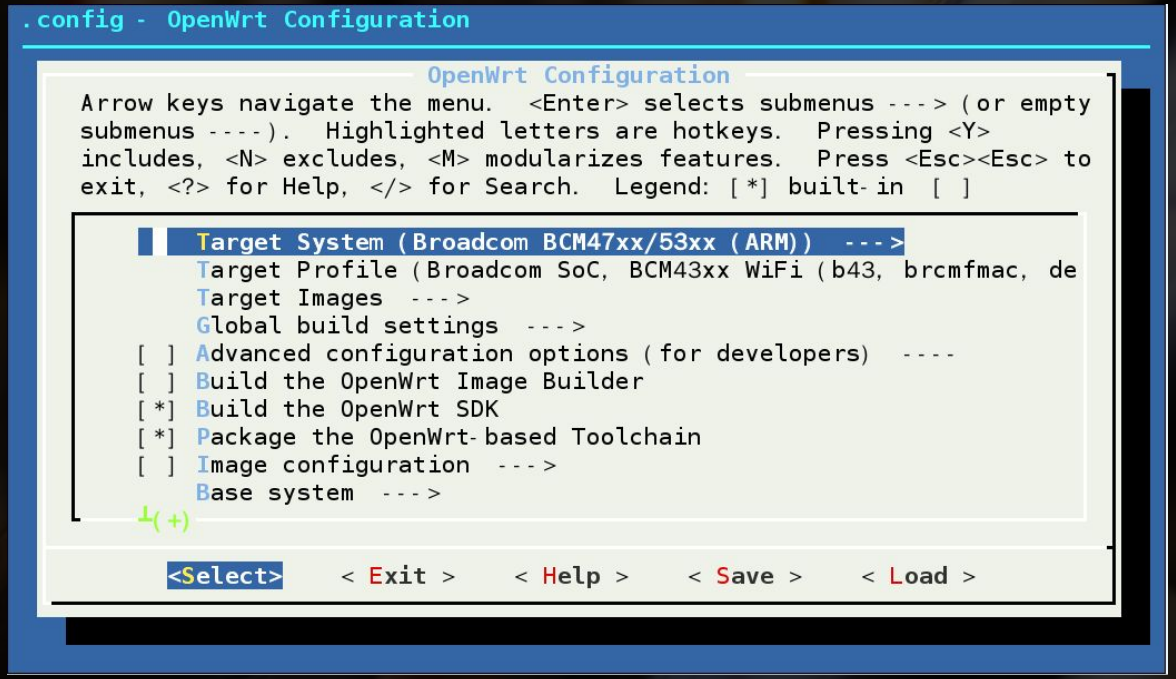
- **/config** : configuration files for menuconfig
- **/include** : makefile configuration files
- **/package** : packages makefile and configuration
- **/scripts** : miscellaneous scripts used throughout the build process
- **/target** : makefile and configuration for building imagebuilder, kernel, sdk and the toolchain built by buildroot.
- **/toolchain** : makefile and configuration for building the toolchain
- **/tools** : miscellaneous tools used throughout the build process

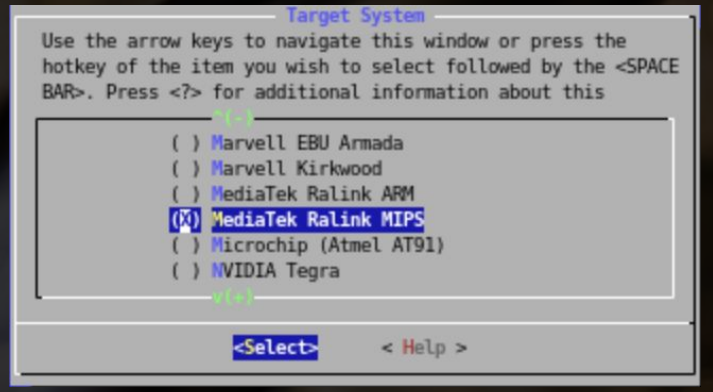
OpenWrt Build Configuration

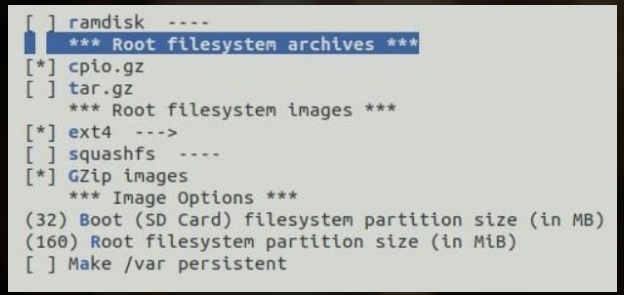
1. Check out OpenWrt Buildroot source tree from the GIT server (SVN Server).
   $ git clone https://git.openwrt.org/openwrt/openwrt.git
2. Update package list by running:
   $ ./scripts/feeds update
3. Install all package info to config file for later make operation:
   $ make package/symlinks
4. Customize your build configuration, also this will check the dependencies and availability of required tools:
   $ make menuconfig.
5. Start building process:
   $ make

OpenWrt Image Configuration

$ make menuconfig.







OpenWrt Building Process

1. Download the cross-compilation tools, kernel headers, etc.
2. Set up the staging directory (staging_dir /). This is where the cross-compilation toolchain will be installed.
   If you want to use the same cross-compilation toolchain for other purposes, such as compiling third-party applications, you can find the cross-compiler tools in this directory, and then use arch-linux-gcc to compile your application.
3. Create the download directory (dl/ by default) - This is where the tarballs will be downloaded.
4. Create the build directory (build_dir/) - This is where all user-space tools while be compiled.
5. Create the target directory (build_dir/target-arch/root by default) and the target filesystem skeleton. This directory will contain the final root filesystem.
6. Install the user-space packages to the root file system and compress the whole root file system with proper format.
7. Generate the result firmware image in bin/ directory

Packages Structure

OpenWrt Package

The term *OpenWrt package* may either refer to one of two things:

- an OpenWrt *source package* which essentially is a directory consisting of:
  - an *OpenWrt package Makefile* describing the acquisition, building and packaging procedures for a piece of software (required)
  - a supplemental directory with *OpenWrt package patches* which modify the acquired source code (optional)
  - other static files that go with the package, such as init script files, default configurations, scripts or other support files (optional)

- an OpenWrt *binary package*, which is a GNU tar compatible archive containing binary executable software artifacts and the accompanying *package control files* for installation on a running system, similar to the `.deb` or `.rpm` files used in other package managers

OpenWrt Source Package Structure

A source package is a subdirectory within its corresponding package feed containing at least one Openwrt **Makefile** and optionally **src**, **files** or **patches** directories.

Makefile - contains a series of header variable assignments, action recipes, one or multiple OpenWrt specific signature footer lines

The files directory - Package related files, .conf, .init, README, e.t.c.

The patches directory - patch files used to modify the source code.

The src directory - additional code to the compilation process.

```
./tcpdump
./tcpdump/patches
./tcpdump/patches/100-tcpdump_mini.patch
./tcpdump/patches/101-CVE-2020-8037.patch
./tcpdump/patches/001-remove_pcap_debug.patch
./tcpdump/patches/102-CVE-2018-16301.patch
./tcpdump/Makefile
```

Packages Feature Considerations

- Do not ship man pages
- Do not ship documentation
- Minimize external dependencies
- Modularize packages
- Try to rely on standard facilities (procd as context switcher)

Licenses

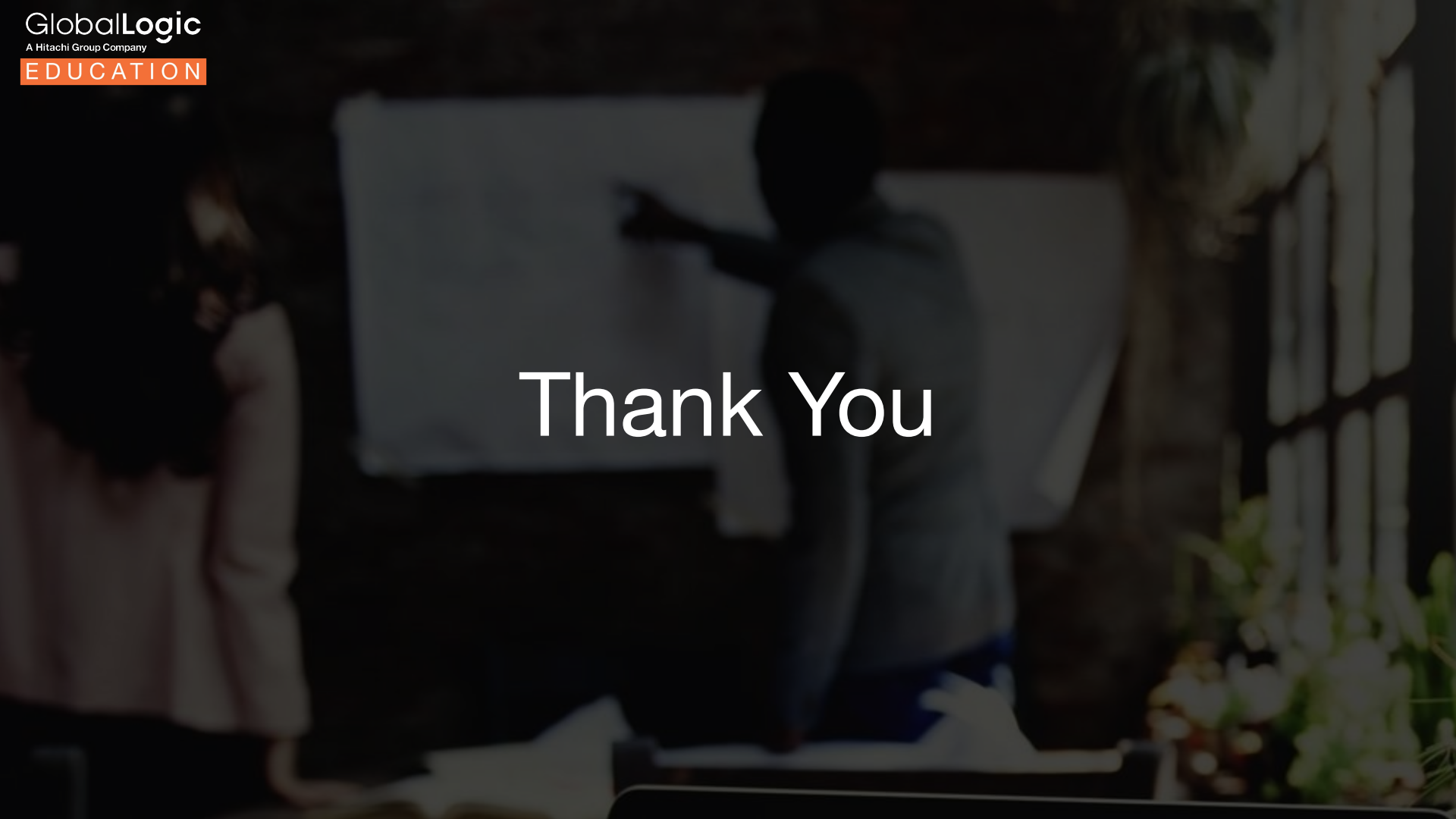
License

- OpenWrt is free software, provided AS-IS and without any warranty. https://openwrt.org/license
- If not otherwise stated in the source files, the OpenWrt build environment is provided under the terms of the GNU General Public License Version 2.
- The OpenWrt distribution (precompiled images etc.) bundles a lot of third party applications and modules which are available under various other Open Source licenses or Public Domain.

Copyright statements

Copyright notice at the top of the Makefile:

Copyright (C) 2007-2010 OpenWrt.org
This is free software, licensed under the GNU General Public License v2.
See /LICENSE for more information.


GlobalLogic
A Hitachi Group Company
EDUCATION
Thank You