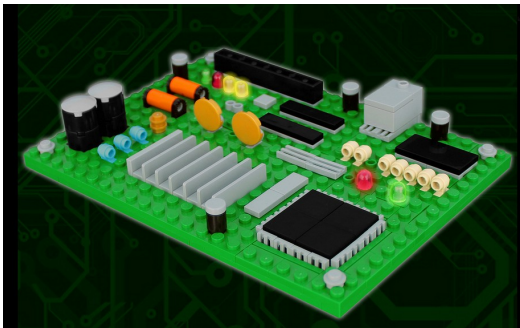
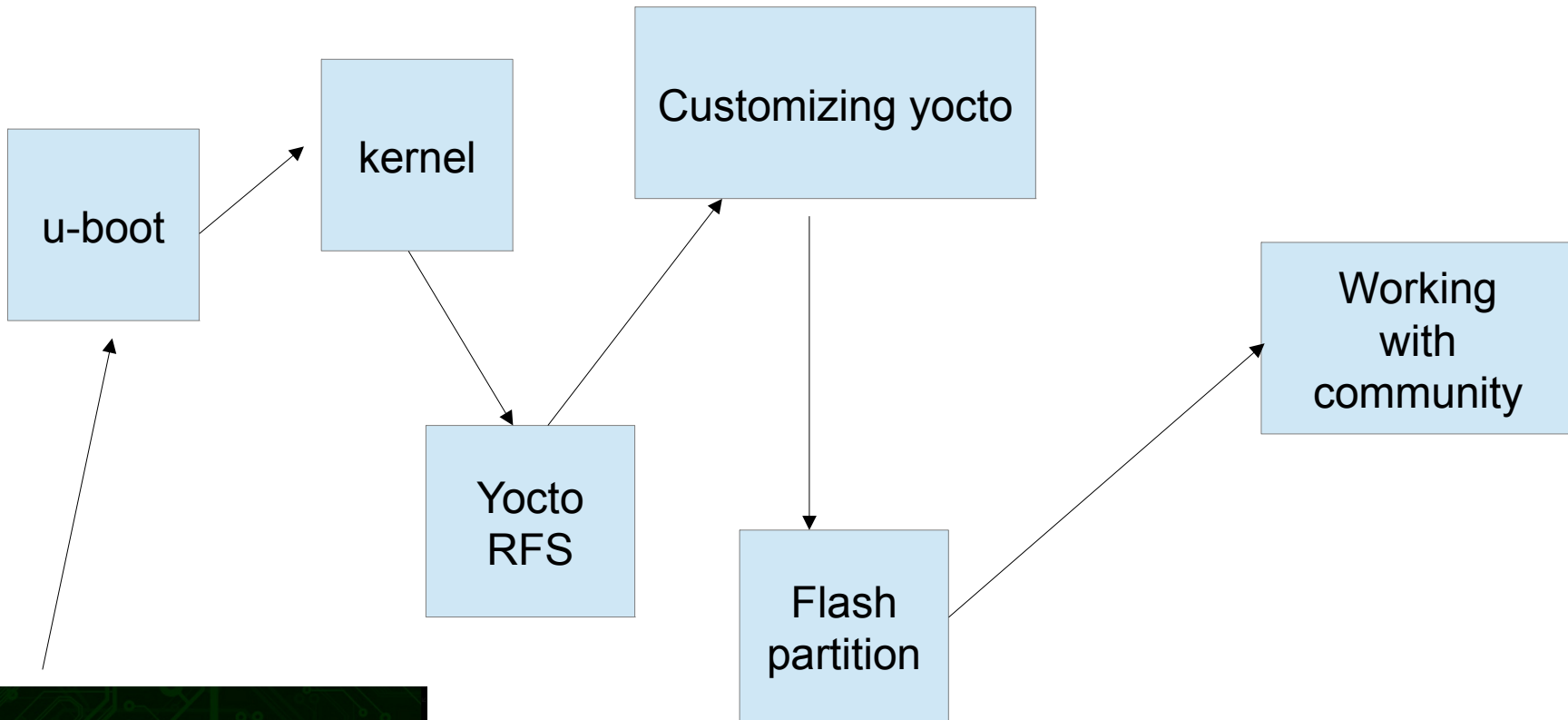


Can a board bring-up be less painful,
if with Yocto and Linux?

Embedded Linux Conference 2014
San Jose, CA

Insop Song
Gainspeed, Inc.

Agenda



U-Boot ?

- primary boot loader for embedded system
 - DDR memory config
 - Network config
 - Flash access
 - Peripheral (gpio, i2c, spi..) drivers
 - Load OS

How to customize U-Boot?

- Start with similar boards' config

- Edit your own config files

- [include/configs](#)

```
insop@sputnik ~/Projects/fsl-u-boot/include/configs(develop) $ ls
A3000.h          ELPT860.h      MPC837XEMDS.h  r7780mp.h
a320evb.h       enbw_cmc.h    MPC837XERDB.h  Rattler.h
a3m071.h        eNET.h        MPC8536DS.h    RBC823.h
a4m072.h        ep8248.h      MPC8540ADS.h   rd6281a.h
...
```

- Add your board info

- [boards.cfg](#)

- Add board specific code to:

- [board/YOUR_BOARD](#)

```
insop@sputnik ~/Projects/fsl-u-boot(develop) $ tree board/xilinx/
board/xilinx/
├── common
│   └── xbasic_types.c
├── ..
├── ml507
│   ├── Makefile
│   ├── ml507.c
│   └── xparameters.h
└── ..
```

- How to build

- [export PATH=/sysroots/fslsdk-linux/usr/bin/ppce-fsl-linux:\\$PATH](#)

- [make CROSS_COMPILE=powerpc-fsl-linux- ARCH=powerpc P1010RDB_config](#)

Add a device handler to u-boot

- Add new device handler to:
 - `board/YOUR_BOARD/my_io_handler.c`
- Add new commands to test/configure devices
 - `U_BOOT_CMD`
- Example
 - `board/amcc/taishan/lcd.c`
 - `U_BOOT_CMD(lcd_test, 1, 1, do_lcd_test, "lcd test display", "");`

U-boot environment variable

- **mkenvimage**
 - a handy tool to generate a U-Boot environment binary image from free-electrons.com
 - <http://free-electrons.com/blog/mkenvimage-uboot-binary-env-generator>
- Prepare variables in txt format
 - `./mkenvimage -s 0x4200 -o uboot-env.bin uboot-env.txt`
 - Program flash with output bin file

Kernel

- Config example
 - `make menuconfig //xconfig, nconfig`
- Keep your `.config` at board specific defconfig
 - `arch/powerpc/configs/40x/walnut_defconfig`
- Build example
 - `export PATH=/sysroots/fslsdk-linux/usr/bin/ppc64e-fsl-linux:$PATH`
 - `make 40x/walnut_defconfig`
 - `make -j4 // -j # of threads`
 - `make help // is your friend`
- Avoid kernel and module version mismatch
 - During development, this temporary patch might be handy to skip '+' in kernel version
 - https://raw.githubusercontent.com/insop/presentation/master/elc_14/scripts/dirty_skip_version.patch

Kernel image

- ulmage: u-boot readable kernel image
 - <http://blog.harrylau.com/2009/08/generate-u-boot-uimage.html> or
 - make ulmage // or use the following helper script
- Helper script:
 - https://raw.githubusercontent.com/insop/presentation/master/elc_14/scripts/mkuimage.sh
 - Helps to keep config file with time stamps

```
uImage.2014-04-24-23-06-insop.u-boot-329
uImage.u-boot
config.uImage.2014-04-24-23-06-insop.u-boot-329
vmlinux.2014-04-24-23-06-insop-329*
```

- On running kernel: [/proc/config.gz](#)

Kernel module build

- Loadable kernel module build
 - `make modules_install INSTALL_MOD_PATH=~/.tmp/linux`
- Move modules to the target board's RFS (root file system)
 - `cd ~/.tmp/linux/lib/modules/3.0.51-rt75+/`
 - `rm -rf build source`
 - `scp -r * root@target3:/lib/modules/3.0.51-rt75+/`
- Module info
 - `modinfo module.ko`

```
filename: /lib/modules/3.8.13-rt9-GS-SDK-V1.4.5j/kernel/drivers/staging/gs_fpgaboot/gs_fpga.ko
license: GPL
description: Xilinx FPGA firmware download
author: Insop Song
depends:
staging: Y
intree: Y
vermagic: 3.8.13-rt9-GS-SDK-V1.4.5j SMP preempt mod_unload
parm: file:Xilinx FPGA firmware file. (charp)
```

Device tree

- Flattened Device Tree (FDT)
 - The operating system uses the FDT data to find and register the devices in the system [1]
 - Board specific information is stored in `.dts` file (text)
 - compiled to `.dtb` (binary) and used during kernel initialization
 - Any new addition should update dt binding doc
 - <http://lxr.free-electrons.com/source/Documentation/devicetree/bindings/>
 - Example dts file
 - https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/ex-dt-fs.dts

[1] http://elinux.org/Device_Tree

[2] <http://events.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf>

Device tree

- Helper scripts

- [\\$mkdtb.sh](#)

- to compile dtb file and generate a single .dts file
 - https://raw.githubusercontent.com/insop/presentation/master/elc_14/scripts/mkdtb.sh

- [\\$mkdts.sh](#)

- to generate .dts file from .dtb file
 - https://raw.githubusercontent.com/insop/presentation/master/elc_14/scripts/mkdts.sh

Yocto project

- Yocto: allow you to create embedded linux distribution, based on OE, bitbake
 - Kernel build
 - **Root file system** generation
 - Package management
 - Yocto quick start
 - <https://www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.html>

Yocto quick start

- Setup development machines
 - Native linux or VM
- Configure `local.conf` file
 - `BB_NUMBER_THREADS = "8"`
 - `PARALLEL_MAKE = "-j 8"`
 - `MACHINE ?= "beagleboard"`
 - `DL_DIR`: for downloading tarballs and source fetch
 - `SSTATE_DIR`: shared state dir
- Run bitbake commands
 - `$ bitbake core-image-minimal`
 - c : command // fetchall, compile, build, configure, clean, cleansstate, cleanall, install, patch
http://www.openembedded.org/wiki/List_of_Executable_tasks
 - f : force the specified task
- Generates file systems
 - ext3, tar.bz2, jffs2
- Generates packages
 - rpm, deb, ipkg

Yocto 101

- Yocto is consist of
 - `meta-*` layers
 - Sample: `meta-skeleton`
 - `meta-*` layers is consist of 'recipes'
 - Recipes: `.bb` and `.conf` files
 - Samples: `meta-skeleton/recipes-skeleton/`
 - Can add custom meta layers to append rules

```
~/yocto/meta-skeleton/recipes-skeleton/service(develop) $ tree
.
├── service
│   ├── COPYRIGHT
│   ├── skeleton
│   └── skeleton_test.c
└── service_0.1.bb
```

Customize image, yocto

- Modify the image to select packages that you need
- Start with known minimal reference image(s)
 - `$bitbake fsl-image-core`
 - <http://git.freescale.com/git/cgit.cgi/ppc/sdk/meta-fsl-networking.git/plain/images/fsl-image-core.bb>
 - https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networking/images/fsl-ima
- Machine type
 - `$ conf/machine/p1020mbg.conf`
 - <http://git.freescale.com/git/cgit.cgi/ppc/sdk/meta-fsl-ppc.git/plain/conf/machine/p1020mbg.conf>

That's great, now what?

- What to follow after creating a minimal image (RFS) for your board?
 - Add/config existing programs/servers
 - Customize init process (init.d)
 - Add your own lib/programs

Add/config existing programs

- Example 1

- logrotate: rotate/compress log files

- Add program to the image first

- https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networking/images/fsl-image-ex.bb

- logrotate recipe

- <http://git.freescale.com/git/cgit.cgi/ppc/sdk/poky.git/tree/meta/recipes-extended/logrotate>

- Test configuration manually

- Create a patch file, and add to the patch to the yocto rule

- https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networking/recipes-append/logrotate/logrotate_3.8.1.bbappend

- https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networking/recipes-append/logrotate/files/logrotate-default.patch

- How to test logrotate

- `logrotate -d --force /etc/logrotate.conf`

- Note: With -d nothing is done (only shown what will be done). It's just for debugging. Remove -d to force logrotate

Adding cron job

- Example2

- crond came with yocto doesn't work for us

- So we decided to use [cronie](#), and works

- How? the same way as logrotate

- add the program in image definition

- add the config to [recipes-append](#)

- https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networkin

- https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networkin

Update init.d

- Sys V init

- Init scripts are at [/etc/init.d/](#)

- Runlevel: 0 - 6, normally 5

- <http://en.wikipedia.org/wiki/Runlevel>

- `$ runlevel` // tells you what runlevel to be booted

- init scripts are sym linked to [/etc/rc*.d \(0-6,S\)](#)

- **INITSCRIPT_PARAMS** configures sym link to [/etc/rc*](#)

- Example

- https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networking/recipes-tools/gsmo

- https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networking/recipes-tools/gsmo

Build and install programs

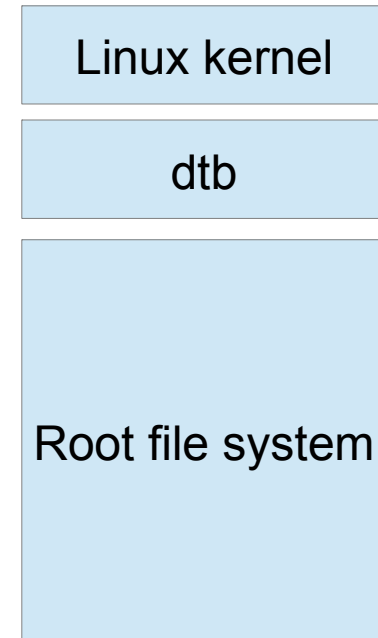
- Programs, not part of yocto
 - Build
 - Add `'virtual/kernel'` if it requires kernel headers
 - Use `_prepend` to add custom rules
 - Example
 - https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networking/recipes-bsp/programs/programs_1.0.bb
- Install script to root file system
 - https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networking/recipes-bsp/programs/programs_1.0.bb

Yocto example 3

- 1st, build your library
 - Install to the yocto's **STAGING_DIR_TARGET**
 - https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networking/rece
- 2nd program link against to the 1st program's library
 - https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta-fsl-networking/rece

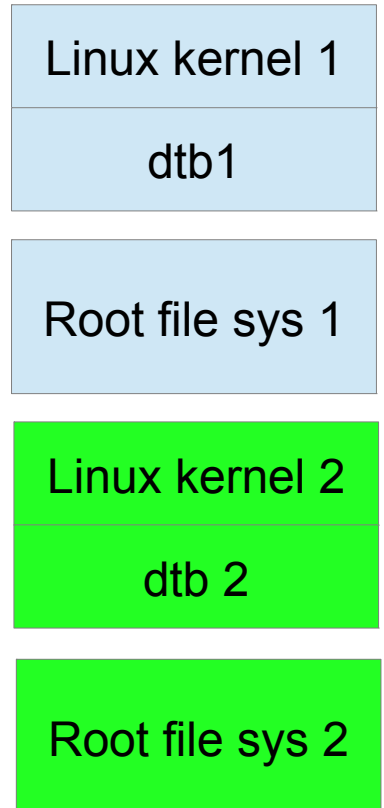
Flash partitions

- Flash storage
 - Block or flash file system
 - Stores kernel, dtb RFS
- Flash file system types
 - Jffs2, ubifs
- Single bootable partition
 - Simple and fully utilize storage
 - brick if upgrade is failed
 - Non trivial to revert upgrade



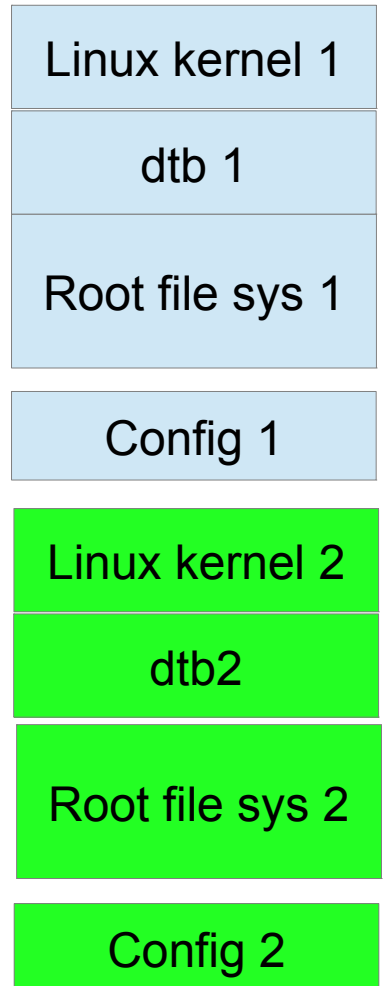
Multiple bootable partitions

- two bootable partitions
 - Storage overhead
 - Can avoid upgrade failure
 - Can revert to previous version



Extra partition for keeping config

- Add config partition
 - To keep configuration across upgrade



Working together with the community

- Applications

- Github hosted projects

- ex> `linenoise`: simple cli (command line interface) library

- <https://github.com/antirez/linenoise/pull/53/files>

- Individually hosted projects

- ex> `sysklogd`: two in one system log daemon

- Create a patch and send it to maintainer & mailing list

- https://raw.githubusercontent.com/insop/presentation/master/elc_14/example/fsl-yocto/meta/recipes-extended/sysklogd/files

- Yocto

- Use mailing list, very active and helpful

- ex> `meta-realtime`: to test sched-deadline

- <http://git.yoctoproject.org/cgi/cgit.cgi/meta-realtime/>

Working with kernel developers

- Existing driver fixes
 - <https://lkml.org/lkml/2014/4/16/635>
 - <http://www.spinics.net/lists/kernel/msg1732810.html>
- Add a new driver
 - [drivers/staging](#) is the place to start
 - https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/staging/gs_fpgaboot?id=refs/tags/v3.15-rc1

Chip manufacturer provided kernel

- You will work with kernel that chip manufacturer provided
 - Had tried to work with them by send them patches
 - Mostly ignored
 - Wanted process as customer support cases
 - Issues of lack of ownership

Conclusion

- With open source projects, board bringin up can be less painful
- Work together with the community

- Thank you
- Question?