Software Update on Embedded Systems

Do not brick your device

Stefano Babic

ELCE October 2014
Introduction

• Me:
  – Software Engineer at DENX, GmbH
  – U-Boot Custodian for Freescale's i.MX
  – Focus on Linux embedded with PowerPC and ARM processors.
Agenda

- Why upgrade?
- Why is it different with a Linux-PC?
- Upgrading strategies
- Swupdate
Why do we need to update an embedded system?

- It is not only hardware
- Bug fixes
- New features can be added
- Security issues: heartbleed, bad implementation...
Why is ES different?

- Power failure
- Bad firmware
- Communication errors in case of remote update
- No access to target

Target must recover from errors!
Which elements must be updated?

- Bootloader (dangerous !)
- Kernel + DT
- Root filesystem
- Application data, other filesystems..
- Customer data (migration )
- Specific software (FPGA bitstream,...)
Where is a new SW installed?
Which interface?

- Local:
  - Local storage (USB, SD,..)
  - Local peripheral (USB as device, UART,..)
- Remote:
  - HTTP / web based
  - FTP
  - Proprietary protocol
  - Many more...
Who will update?
No expertise required
System upgrade solutions

- Bootloader upgrade
- Linux upgrade
  - Package Manager
  - Rescue image or specific application
  - From the running application
Bootloader

- Limited access to peripherals (drivers, filesystems)
- Implementation in bootloader not in sync with Linux
- Limited network support (UDP, not TCP)
- Limited UI with an operator

✔ Update is simpler
✔ Smaller footprint
Linux App

- Footprint

- Availability of all drivers used by the product
- A lot of tools/libraries
package manager as distro?

- Upgrade is not atomic
- Nightmare for test engineers/support
- New firmware partially written
- More places where things can go wrong

☑ Small update image
Full update

- Size, Time to transfer
  - Atomic: it works or not
  - Single image delivery
Double copy strategy

- **BOOT LOADER**
  - Application Software Running copy
  - Application Software Standby copy
  - Databases, config, user data

- **BOOT LOADER**
  - Application Software Standby copy
  - Application Software Running copy
  - Databases, config, user data
Single copy (rescue)
swupdate: FLOSS upgrade sw

- Missing an open source upgrade software for ES
- Take care of failure mechanism
- Hardware / software compatibility
- Proof correctness images to be installed (chksum,..)
- Partitioning storage
- Local or remote install
Swupdate-2

- Scriptable (LUA), pre- and postinstall scripts
- Single image for multiple devices
- Easy for users to perform update
- Missing: signed images!
Handling hardware differences

- **HMI**
- **Gateway**
- **Type A-1**
- **Type A-2**
- **Type A-3**
- **Type A-4**
One release, multiple devices

Release XX.YY for device family

Software for A-1

Software for A-3

Software for HMI

Type A-1

Type A-3

HMI
Single image structure

CPIO Header

sw-description

Image 1

Image 2

Image 3

Image i

Image n
Swupdate architecture

- Notifier
- INSTALLER
- Handler manager
  - UBI
  - MTD
  - RAW
  - U-Boot ENV
  - Custom LUA Handler
- Local Storage
- WebServer
- Custom protocol

Default Parser (libconfig)
Custom Parser (LUA)
Handling HW differences

software =
{
    version = "0.1.0";

    target-1 = {
        images: (  
                    
        );
    };

    target-2 = {
        images: (  
                    
        );
    };
}
sw-description

software =
{
    version = "0.1.0";

    myboard = {

        hardware-compatibility: [ "1.2", "1.3", "18#010071"];

        partitions: ( /* UBI Volumes */
            {
                name = "rootfs";
                device = "mtd10";
                size = 104896512; /* in bytes */
            },
            {
                name = "kernel";
                device = "mtd9";
                size = 4194304; /* in bytes */
            }
        );
    }
}
sw-description

images: (  
    {  
        filename = "core-image-base-myboard.ubifs";  
        volume = "rootfs";  
    },  
    {  
        filename = "uboot-env";  
        type = "uboot";  
    },  
    {  
        filename = "uImage";  
        volume = "kernel";  
    },  
    {  
        filename = "fpga.bin";  
        type = "fpga";  
    }  
);
sw-description: scripts, u-boot

scripts: (
    {
        filename = "test.lua";
        type = "lua";
    },
    {
        filename = "sdcard.lua";
        type = "lua";
    },
    {
        filename = "test_shell.sh";
        type = "shellscript";
    }
);

uboot: (
    {
        name = "vram";
        value = "4M";
    }
)
Recovery from failures

- U-Boot
- Swupdate
- Set update flag
- Update flag set?
- Bootcounter?
- Load kernel
- Kernel boots
- Panic, watchdog...
- Application
- Reset bootcounter
- Success
- Fail
- Run
- Reset update flag
API for external client

Client

swupdate

REQ_INSTALL

ACK

DATA (IMAGE)

DATA (IMAGE)

DATA (IMAGE)

DATA (IMAGE)

DATA (IMAGE)

DATA (IMAGE)

DATA (IMAGE)

DATA (IMAGE)

DATA (IMAGE)

DATA (IMAGE)

DATA (IMAGE)

DATA (IMAGE)

GET STATUS

Status, Notification

GET STATUS

Status, Notification
Using with Yocto

● Meta-swupdate

● It generates a ramdisk suitable for u-boot (.uboot.gz)

● “dora” and “daisy” branches

● Footprint RAMDISK (gzipped) : 2.6 – 7 MB
  - Typical: ~4MB
require("swupdate")

fpga_handler = function(image)
    print("Install FPGA Software ")
    for k,l in pairs(image) do
        print("image[" .. tostring(k) .. "] = " .. tostring(l) )
        swupdate.notify(swupdate.RECOVERY_STATUS.RUN,0,
        "image[" .. tostring(k) .. "] = " .. tostring(l))
    end
    return 0
end

swupdate.register_handler("fpga",fpga_handler)
swupdate todo list

- Create a community around the project
- Security: add support for signed images!
- Low resources: support for full streamable image
- New handlers
Links

- Swupdate sources at https://github.com/sbabic/swupdate
- Documentation at http://sbabic.github.io/swupdate
- Mailing list: swupdate@googlegroups.com
- http://www.denx.de/
Questions ...

• It's your turn now...