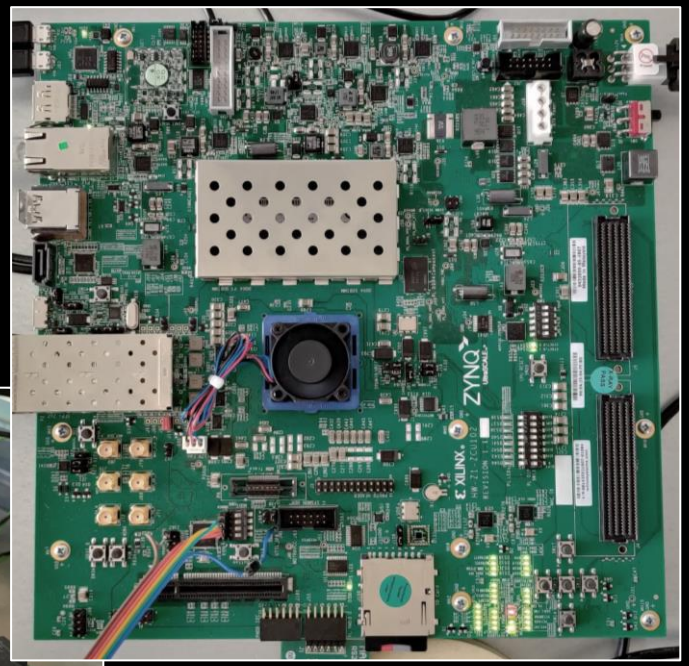
Reliable booting system for Zynq Ultrascale+ MPSoCs

Introduction to RelBoot & RelUpgrades for Zynq MPSoCs

Nekija Džemaili

CMS-DAQ team

Acknowledgements:
P. Žejdl, M. Dobson, V. Amoiridis, D. Gigi

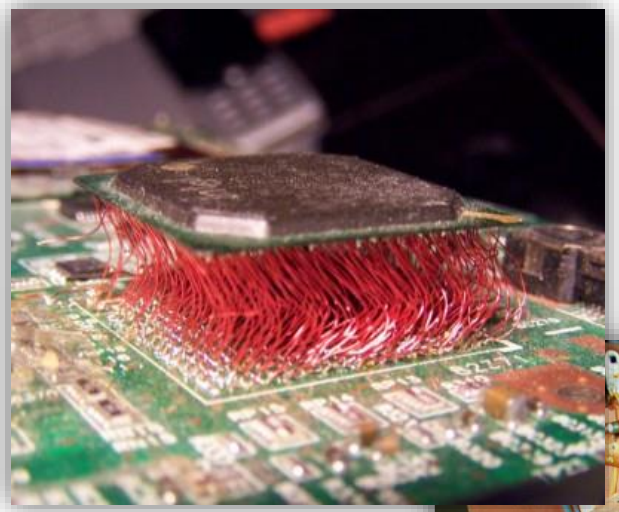


ZCU102 development board

Trenz mezzanine + carrier board (TE0803 + TEBF0808)

Contents

- Why a reliable booting system?
- Zynq MPSoC and booting process overview
- Possible failures
- Built-in failover mechanisms
- Proposed solutions
  - Reliable Booting (RelBoot)
  - Reliable Upgrades (RelUpgrade)
- Implementation
- Tutorial time!



Source: EEVBLOG

Source: hackaday

These work,
but are they reliable?

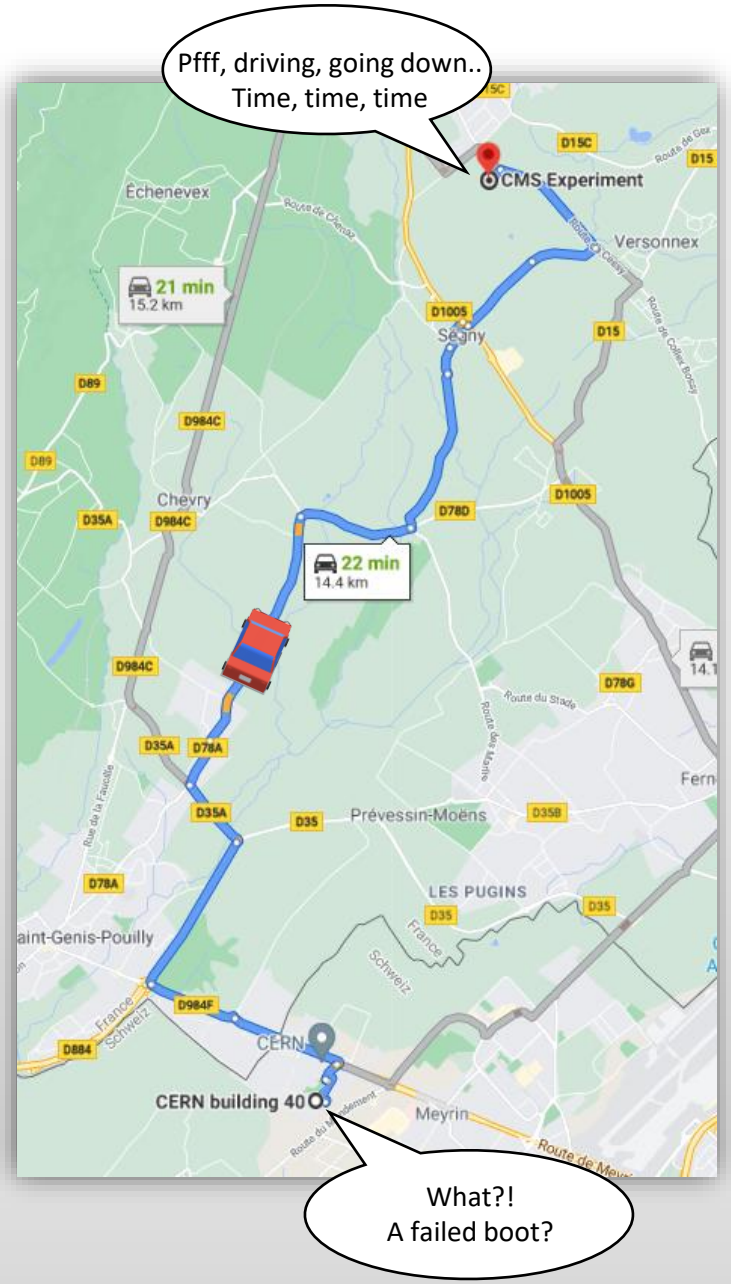
Why reliable?

- Failed Zynq Ultrascale+ MPSoC booting
  - Debugging
  - Remote connection not always possible

- Hardware in CMS service cavern ***is*** **accessible** →

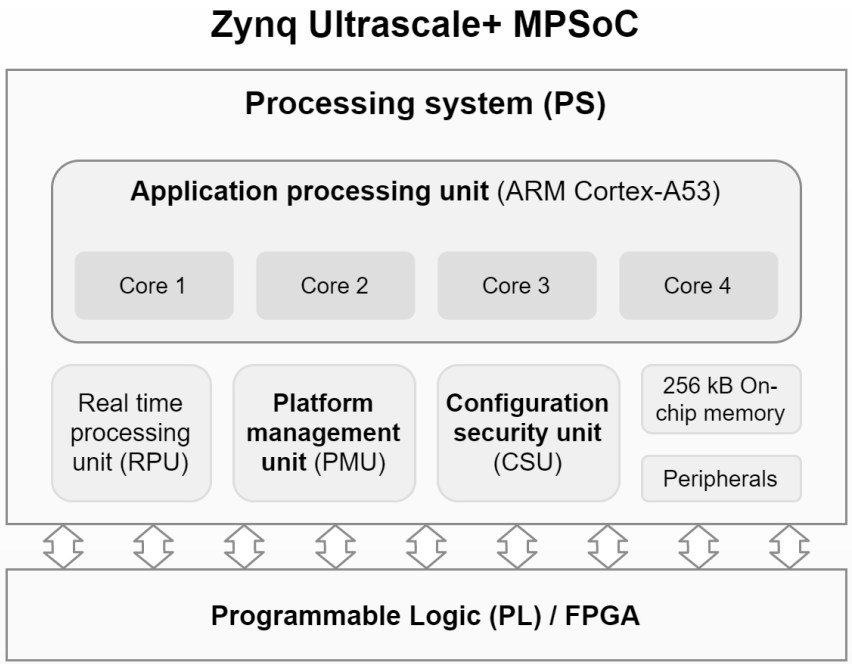
- The reliable booting system can:
  - Bring Zynq MPSoC to a **well-known state!**
    - A console that is **responding**
    - Accessible remotely

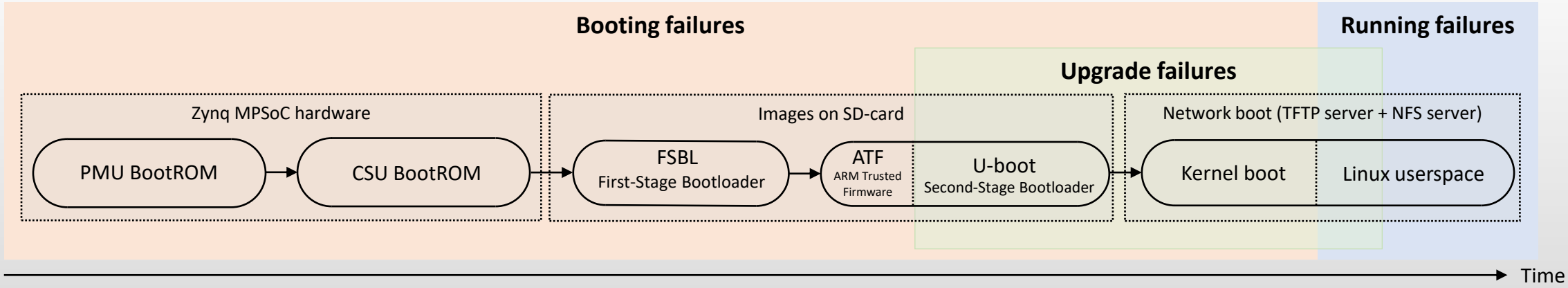
  - Recover from a failed boot, upgrade or running system
  - Store the reason of the failure
  - Report the failure over email to the user



Zynq MPSoC booting process

- **Non-trivial booting process** with multiple processing units
  - PMU and CSU run BootROMs
  - APU runs bootloaders and Linux OS
- Boot image on SD-card
- CSM DAQ has chosen to boot **through the network (using TFTP + NFS)**

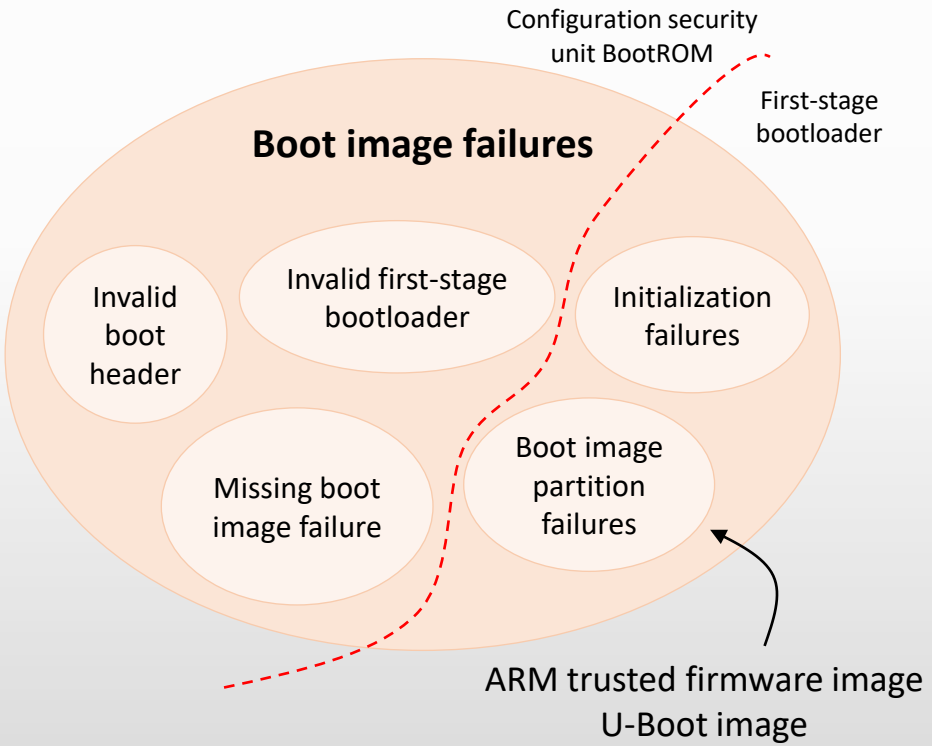




Built-in failover mechanisms

CSU BootROM and FSBL

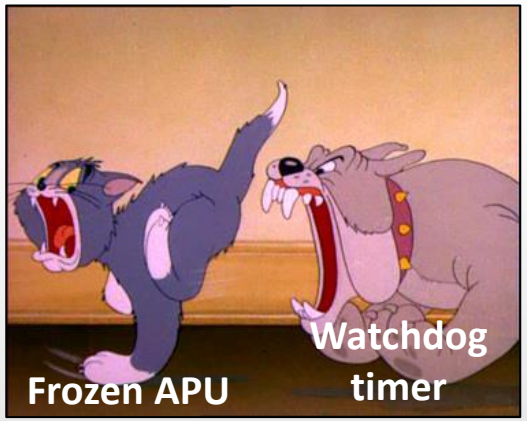
- Golden image search mechanism (CSU BootROM) & MultiBoot mechanism (FSBL)
- Mechanisms recover from failures by:
  - **Validating boot header** of the boot image
  - Performing checksums on boot image partitions
  - Using **multiple boot images** on the boot device
    - `BOOT0001.BIN`
      `BOOT0002.BIN`
      `BOOT0003.BIN`
      etc…
    - Switch to other image when image does not pass validation
- Mechanisms can detect:
  - Missing boot image
  - Invalid boot header
  - Corrupt image partitions (FSBL, ATF, U-Boot, PMU firmware)



Built-in failover mechanisms

Watchdog timers

- **Watchdog timers** in the Zynq MPSoC
  - CSU watchdog timer → **Can reset the system when PMU firmware gets stuck**
  - Full Power Domain (FPD) watchdog timer → **Can reset the APU when stuck**
  - Low Power Domain (LPD) watchdog timer → Can reset the RPU *(not used)*



- Can **recover** the Zynq MPSoC from:
  - Programmable logic freezes the APU
  - Failed kernel boot-up (Kernel panic)
  - Broken U-Boot

- FPD/LPD watchdog timeouts interrupts are handled by the **PMU firmware**
  - **60 seconds default timeout**
  - Timeout period can be changed using ~~**RECOVERY_TIMEOUT**~~ flag in the PMU firmware

We reconfigure the watchdog timer timeout through U-Boot

Built-in failover mechanisms

Watchdog timers

- Enabling **watchdog timer handling** in PMU firmware:
  - Error management module (ENABLE_EM)
  - Recovery module (ENABLE_RECOVERY)
  - CSU watchdog timer functionality (ENABLE_WDT)

Also requires some kernel, U-Boot and device-tree configuration

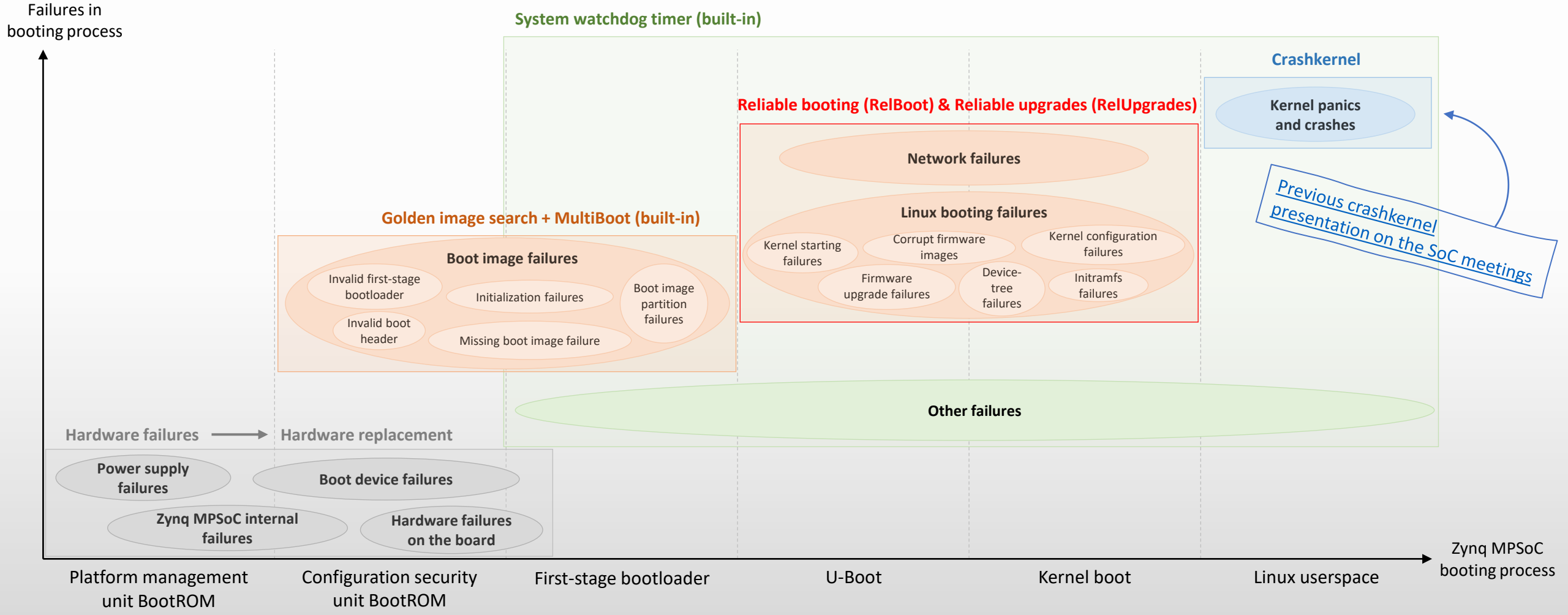
PMU firmware **compilation flags**

- Watchdog timer is initialized in the **first-stage bootloader**

- PMU firmware will **reset** the Zynq MPSoC after watchdog timer expiry
- Requires **periodic servicing** using a **heartbeat service** in Linux
  - Created a **systemd heartbeat service** in Linux

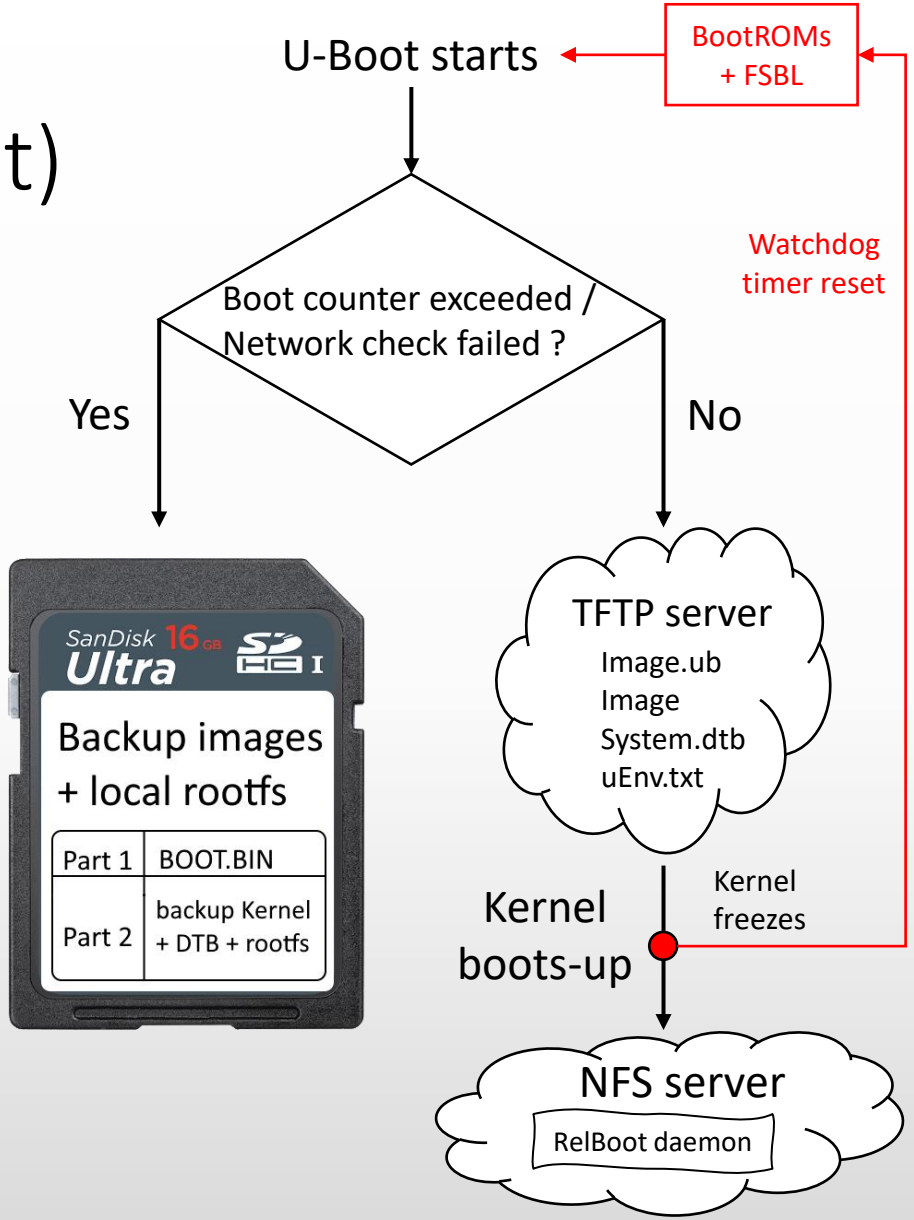
Xilinx provides an example watchdog servicing application

Zynq MPSoC failure chart and fallbacks



Reliable booting mechanism (RelBoot)

- Fallback for **booting failures**
  - A **boot counter** to detect multiple failed boot attempts
  - In case of multiple failed boot attempts, performs a fallback boot from a **backup image**
  - Network failures result in direct booting from the backup
  - Added as custom **U-Boot script**
- Uses **backup Linux images** on SD-card
  - Switches to backup Linux images after detecting a failure
  - Local root filesystem on SD-card or ramdisk
- Added custom **daemon in Linux (RelBoot & RelUpgrades daemon)**
  - Detects if system was booted using backup images
  - Notifies user through email
    - If network is available

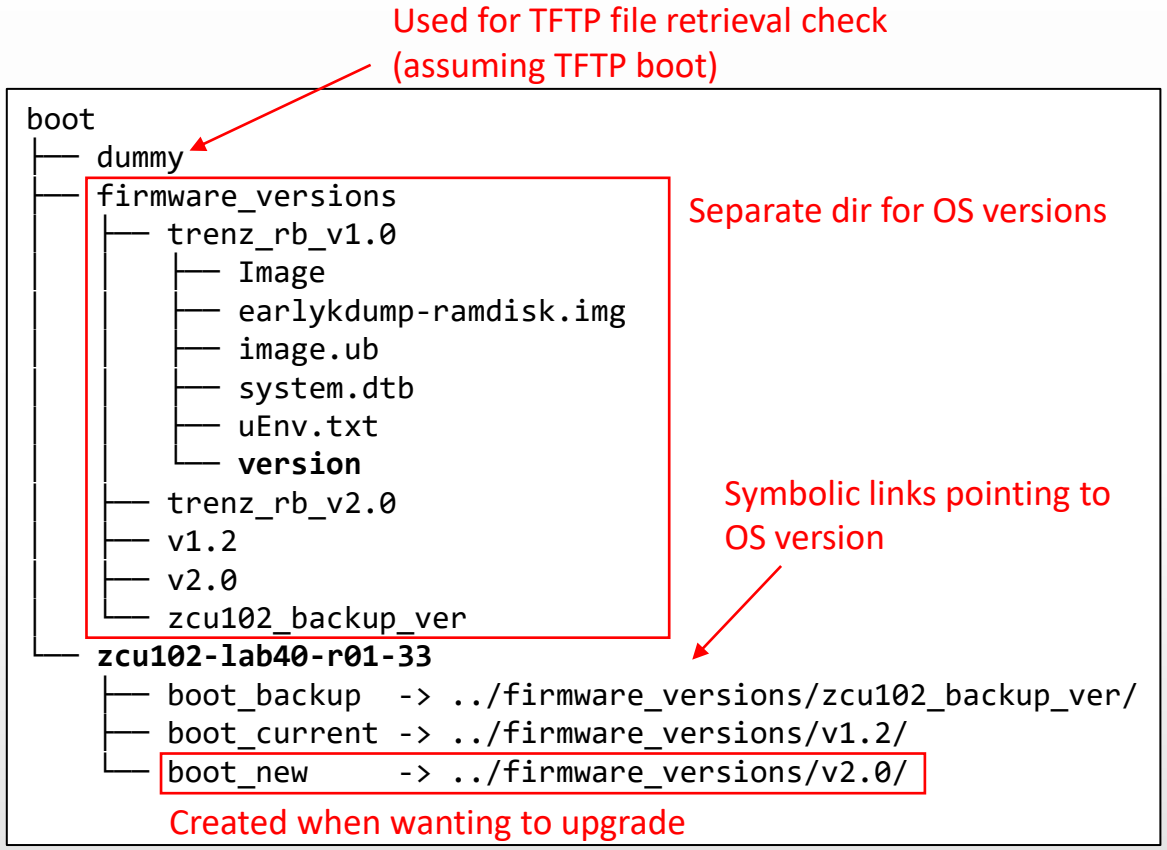


Reliable upgrade mechanism (RelUpgrades)

- Implements an **upgrade system**
  - Can protect against **upgrade failures**
  - Supports kernel, device-tree, and ramdisk upgrades
    - Support for FPGA firmware and OS upgrades are coming

- **Extension** in Reliable booting (RelBoot) U-Boot script
  - Detects **new versions** on boot device
  - Attempts upgrades **automatically**
  - **Rolls back to previous version automatically** after multiple failed upgrade attempts using a counter.

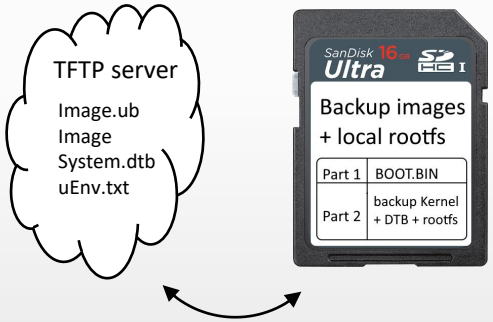
- **RelBoot & RelUpgrades Linux daemon**
  - Detects if an **upgrade was started** during booting
  - Notifies user of upgrades
  - Finishes upgrade by **updating files on boot device** automatically



RelBoot & RelUpgrade features

- RelBoot and RelUpgrades are part of U-Boot booting script
  - Created using HUSH scripting language
  - Can be enabled/disabled

- Reliable booting system supports **different bootmodes** ⟶ Ability to configure **primary** and **secondary** bootmode
  - SD boot
  - TFTP boot
  - QSPI boot support *(planned)*



- Support for 3 boot image sets
  1. **`Image.ub`** format
  2. Separate kernel and device-tree images (**`Image + system.dtb`**)
  3. Separate kernel and device-tree images + ramdisk (**`Image + system.dtb + ramdisk.img`**)

- uEnv.txt support **on TFTP server**
  - Makes booting procedure more configurable for the user
  - **File is easy to change remotely** ⟶ Even when the board is off

Proposed network booting!
Presentation by Petr Žejdl on wednesday

Implementation of RelBoot & RelUpgrades

U-Boot environment

- Script is compiled into the U-Boot binary
  - Added using the **`CONFIG_EXTRA_ENV_SETTINGS`** option
  - Script is saved in the **U-Boot environment on QSPI flash** during installation

- U-Boot environment contains **configuration values** to enable/disable features of the booting script
- System can be configured through
  - RelBoot & RelUpgrades daemon configuration file
  - uEnv.txt file on TFTP server
  - Manually
    - U-Boot CLI (console)
    - Linux terminal (using U-Boot firmware utilities)

```
$ cat /etc/relboot.d/relboot.conf
enable_relboot                  1
enable_relup                    1
boot_attempts                   3
currentver_fw_dir               boot_current
newver_fw_dir                   boot_new
backup_dir                      boot_backup
primary_bootdev                 tftp_boot
secondary_bootdev               sd_boot
mmc_blkpart                     0:2
img_type                        sep_img_ramdisk
…
```

Adding custom RelBoot and RelUpgrade U-Boot scripts in PetaLinux projects

- Currently using PetaLinux 2019.2
- Adding big U-Boot scripts to macro `CONFIG_EXTRA_ENV_SETTINGS` **can be a headache**!!
  - Very unreadable

```
#define CONFIG_EXTRA_ENV_SETTINGS \
        SERIAL_MULTI \
        CONSOLE_ARG \
        DFU_ALT_INFO_RAM \
        DFU_ALT_INFO_MMC \
        PSSERIAL0 \
        "nc=setenv stdout nc;setenv stdin nc;\0" \
        "ethaddr=\0" \
        "bootenv=uEnv.txt\0" \
        "importbootenv=echo \"Importing environment from SD ...\"; " \
                "env import -t ${loadbootenv_addr} $filesize\0" \
        "loadbootenv=load mmc $sdbootdev:$partid ${loadbootenv_addr}
${bootenv}\0" \
        "sd_uEnvtxt_existence_test=test -e mmc $sdbootdev:$partid
/uEnv.txt\0" \
        "uenvboot=" \
                "if run sd_uEnvtxt_existence_test; then " \
                        "run loadbootenv; " \
                        "echo Loaded environment from ${bootenv}; " \
                        "run importbootenv; " \
                        "fi; " \
                "if test -n $uenvcmd; then " \
                        "echo Running uenvcmd ...; " \
                        "run uenvcmd; " \
                "fi\0" \
        "autoload=no\0" \
        "sdbootdev=0\0" \
        "clobstart=0x10000000\0" \
        "netstart=0x10000000\0" \
        "dtbnetstart=0x23fff000\0" \
        "loadaddr=0x10000000\0" \
        "boot_img=BOOT.BIN\0" \
        "load_boot=tftpboot ${clobstart} ${boot_img}\0" \
        "bootenvstart=0x100000\0" \
        "eraseenv=sf probe 0 && sf erase ${bootenvstart}
${bootenvsize}\0" \
        "jffs2_img=rootfs.jffs2\0" \
        "load_jffs2=tftpboot ${clobstart} ${jffs2_img}\0" \
        "sd_update_jffs2=echo Updating jffs2 from SD; " \
        " mmcinfo && fatload mmc ${sdbootdev}:1 ${clobstart}
${jffs2_img} && run install_jfs2\0" \
        "install_jffs2=sf probe 0 && sf erase ${jffs2start}
${jffs2size} && " \
                "sf write ${clobstart} ${jffs2start} ${filesize}\0" \
        "kernel_img=image.ub\0" \
        "load_kernel=tftpboot ${clobstart} ${kernel_img}\0" \
ETC...
```

- Developed **Scriptadder tool**
  - Allows script code in a **separate file**
  - Allows you to **define "functions"** in an easy way
  - **Parses** the script code and
    - Translates "functions" into environmental variables for U-Boot
    - Adds formatting and inserts tabs for easy reading later
    - Adds it to macro `CONFIG_EXTRA_ENV_SETTINGS` in `platform-top.h`

`<petalinux-project>/project-spec/meta-user/recipes-bsp/u-boot/files/`**`platform-top.h`**

Adding U-Boot scripts to CONFIG_EXTRA_ENV_SETTINGS

Using Scriptadder and function macros

U-Boot script defined in a separate file using Scriptadder *function macro*:

```
1   @UENV_FUNC test_func()
2   {
3       # This is a comment
4       echo “Hello world!”;
5       echo “This is U-Boot with added reliability”;
6
7       flag=1;
8       if test flag = 1; then
9           echo “The flag is one”;
10      fi;
11  }
```

Output in CONFIG_EXTRA_ENV_SETTINGS:

```
1   "test_func= \n" \
2       "\techo \"Hello world!\"; \n" \
3       "\techo \"This is U-Boot with added reliability\"; \n" \
4       "\t \n" \
5       "\tflag=1; \n" \
6       "\tif test flag = 1; then \n" \
7           "\t\techo \"The flag is one\"; \n" \
8       "\tfi; \n\0" \
```

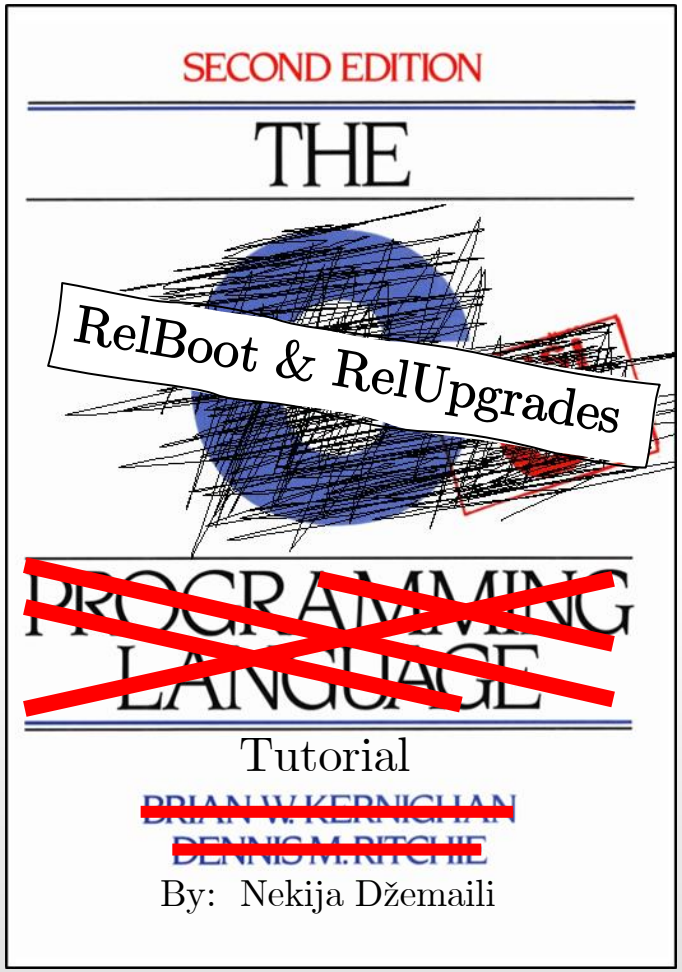
Output in U-Boot:

```
ZynqMP> printenv test_func
test_func=
    echo “Hello world!”;
    echo “This is U-Boot with added reliability”;

    flag=1;
    if test flag = 1; then
        echo “The flag is one”;
    fi;
```

Tutorial time!

1. Install Reliable Booting services on root filesystem
2. Show configuration of RelBoot and RelUpgrades
3. Add Reliable Booting system to PetaLinux project and U-Boot
4. Testing the reliable booting system
   - NFS-server not available
   - Kernel image checksum failure in U-Boot
   - System service causes kernel crash



Thanks for listening!!!

- GitLab repository + guide:
  https://gitlab.cern.ch/hardware/zynq/zynq-relboot
- Detailed documentation:
  https://cds.cern.ch/record/2763095
- Contact for questions:
  e-mail address: nekija.dzemaili@cern.ch

Backup slides

Built-in failover mechanisms

CSU BootROM and FSBL

- Golden image search mechanism (CSU BootROM)
  - Searches for "XLNX" identification string in the **boot header** of the boot image
  - Checks the validity of the boot header using checksum (requires **checksum** attribute in BIF file)
  - If nothing is found:
    - Try to search for another boot image
    - Increment the offset value in CSU_MULTI_BOOT register
    - Offset the search address by 32 kB (Using QSPI/NAND flash)
    - Convert offset value to string and concatenate with BOOT.BIN filename → BOOT0001.BIN (Using SD/eMMC)

BOOT0001.BIN
BOOT0002.BIN
BOOT0003.BIN
BOOT0004.BIN
BOOT0005.BIN
Etc…

- MultiBoot mechanism (FSBL)
  - Validates the rest of the boot image
  - Increments CSU_MULTI_BOOT register when a failure is detected
  - Performs system reset
  - CSU BootROM will try to load another image

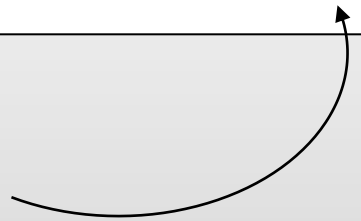
Binary Image Format

Checksum attribute

- BIF files specify the layout of the boot image
- Used by **Bootgen utility** to generate the boot image
- Enabling checksumming for FSBL and other image partitions:

```
the_ROM_image:
{
    [bootloader, destination_cpu=a53-0, checksum=SHA3] zynqmp_fsbl.elf
    [pmufw_image, checksum=SHA3] pmufw.elf
    [destination_cpu=a53-0, exception_level=el-3, trustzone, checksum=SHA3] bl31.elf
    [destination_cpu=a53-0, exception_level=el-2, checksum=SHA3] u-boot.elf
}
```

- You have to include the **checksum** attribute



Enabling watchdog timer support for Linux

Kernel and device-tree configuration

- Enabling the watchdog drivers in the **kernel menuconfig**:

```
Device Drivers --->
     [*] Watchdog Timer Support --->                          CONFIG_WATCHDOG
          [*] Disable watchdog shutdown on close              CONFIG_WATCHDOG_NOWAYOUT
          <*> Xilinx Watchdog Timer                           CONFIG_XILINX_WATCHDOG
          <*> Cadence Watchdog Timer                          CONFIG_CADENCE_WATCHDOG
```

- Watchdog node in device-tree is normally defined in `zynqmp.dtsi` provided by Xilinx.
- Changing parameters for the watchdog node using the **meta-user layer and `system-user.dtsi`**:

```
&watchdog0 {
     status = "okay";
     reset-on-timeout;
     timeout-sec = <60>;
};
```

Example boot device setup for RelBoot

Secondary boot device

SD-card partition 1

```
BOOT.BIN
BOOT0001.BIN
BOOT0002.BIN
BOOT0003.BIN
BOOT0004.BIN
```

SD-card partition 2

```
bin
boot
├── firmware_versions
│   └── zcu102_backup_ver
└── zcu102-lab40-r01-33
    └── boot_backup -> ../firmware_versions/zcu102_backup_ver/
dev
etc
home
lib
lib64
…
```

Primary boot device

TFTP server

```
boot
├── dummy
├── firmware_versions
│   ├── trenz_rb_v1.0
│   │   ├── Image
│   │   ├── earlykdump-ramdisk.img
│   │   ├── image.ub
│   │   ├── system.dtb
│   │   ├── uEnv.txt
│   │   └── version
│   ├── trenz_rb_v2.0
│   ├── v1.2
│   ├── v2.0
│   └── zcu102_backup_ver
├── trenz-lab40-r02-board01
│   └── boot_current -> ../firmware_versions/trenz_rb/v2.0/
└── zcu102-lab40-r01-33
    ├── boot_backup  -> ../firmware_versions/zcu102_backup_ver/
    └── boot_current -> ../firmware_versions/v1.2/
```