Upgraded Unified Board

Operative System and firmware upgrading

User and developer manual

This document is a work in progress.
The most current version is kept in the github repository:
https://github.com/auger-prime-sde/uub-integration

Written by Roberto Assiro

Manual version 2.4 – written for UUB
V3 – sys ver. 0.99.3

INFN - LECCE
Nov 2019

Standard UUB system configuration

Default system setting parameters

- MAC address 00:0A:35:00:1E:53
- Eth0 dhcp enabled
- Eth0:0 192.168.168.168 - net mask 255.255.255.0 (fixed number for the field)
- System console speed **115200** baud (u-boot and petalinux)
- Login: **root**
- Password: **root**

To change standard configuration the patch must be applied

Standard file names

Repository's files for UUB:

- uboot.bin First file to transfer in UUB's flash memory. Uboot and FSBL
- image.ub Compressed image of the Petalinux System (Kernel, device tree and root file system)
- fpga.bit bitstream file for FPGA
- uub-flash.bin Binary image of flash memory128Mb. It includes all partitions and volumes
- **system.dtb** Device tree file (optional – Also present in image.ub)

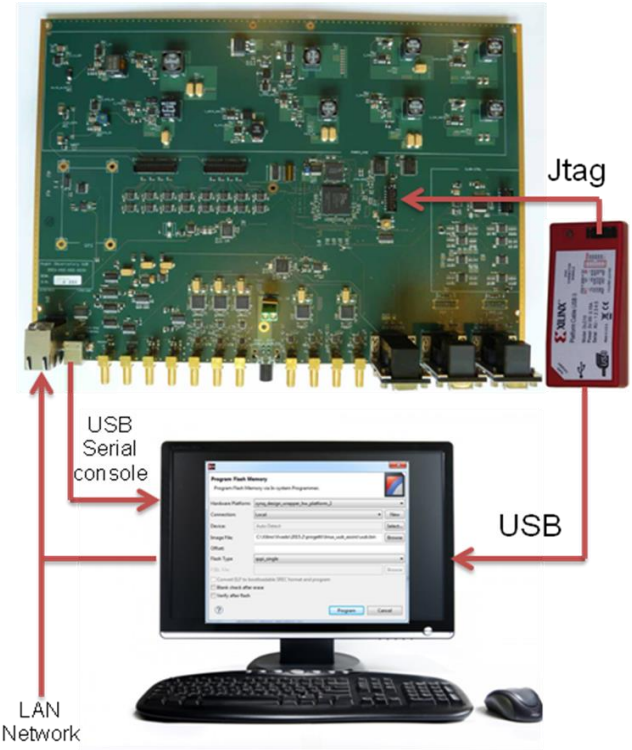
Implemented commands in Petalinux

- **mountboot** mounts itbs volume (system image folder) under **/boot**
- **umountboot** unmounts the volume
- **mountflash** mounts flash volume (64 Mb of flash memory) under **/flash**
- **umountflash** unmounts the volume
- **version (**or **ver**) shows version of system
- **recovery-boot** restore system image, bitstream and patch directory from **recovery volume to /boot**
- **recovery-update** update recovery image of system and bitstream with current into recovery volume
- **uub-update** command to update the UUB in SDECO lab only (IP 192.168.1.136)
- **makeflash** command to create the /flash volume in MTD3
- **daq on/off** command to stop (off) and run(on) the data acquisition on the UUB
- **watchdog-log** print watchdog log file from flash memory (history events)
- **image** replace petalinux image.ub from tftp server (no bitstream)
- **restart** command to restart the UUB with the watchdog disabled
- **upgrade <IP number>** command to upgrade all the system from a tftp machine on the local network with specific IP number
- **sdeco_update** command to upgrade the uub in the SDECO network from the TFTP server

How to program a brand new UUB

store uub.bin to flash memory (Jtag only)

1. Connect jtag programmer to UUB (Digilent or xilinx)
2. Remove the jumper (flash/jtag) and turn on the UUB
3. Launch SDK on computer and select "**Program flash**" from the Xilinx Tools menu
4. Select **uub-flash.bin** as image file (128 Mb)
5. Click "Program" button to store the file into the flash memory. The programming phase requires 60 minutes. Restart UUB



step 1: store uboot.bin to flash (jtag + TFTP)

6. Step 1, 2, 3.
7. Select **uboot.bin** as image file (480 Kb)
8. Click "Program" button to store the file into the flash memory. The programming phase requires few minutes
9. Turn off the UUB, insert the jumper and turn on UUB again. U-boot will start from flash memory.

step 2: Store uub-flash.bin to flash memory by TFTP

- **uub-flash.bin** is the entire image file of flash memory (128Mb)
- Turn on the UUB and wait to run u-boot . After few seconds you will get the prompt command :

```
U-boot-UUB>
```

Please set TFTP server IP if needed set serverip 192.168.1.1 (example ip number)

`run flash-all` is a custom command under u-boot to update the flash memory partitions

```
U-boot-UUB> run flash-all  (tftp server required on the network)

Load address: 0x2000000
Loading: ##################################################################
#################################################################
#################################################################
#################################################################
946.3 KiB/s
done
Bytes transferred = 33554432 (3000000 hex)

device 0 offset 0x0, size 0x8000000
18481152 bytes written, 15073280 bytes skipped in 59.551s, speed 578524 B/s
U-boot-UUB>
```

uub-flash.bin is stored into the flash memory

Sequence to program a UUB's flash memory (not empty)

- Turn on the UUB and wait to run u-boot . After few seconds you will get the prompt command :

```
U-boot-UUB>
```

Erase two partitions area of flash memory by the follow commands:

sf probe
sf erase 0x0 0x4000000

1. Connect jtag programmer to UUB (Digilent or xilinx)
2. Launch SDK on computer and select "**Program flash**" from the Xilinx Tools menu
3. Select **uboot.bin** as image file (480 Kb)
4. Click "Program" button to store the file into the flash memory. The programming phase requires few minutes

Now the first partition is programmed with u-boot

Restart the uub and wait u-boot prompt:

```
U-boot-UUB>
```

Please set TFTP server IP of your linux machine on the network: (Is needed a TFTP server service on your network)
set serverip 192.168.1.1 (example ip number)
Download and store the file uub.bin into the flash:

```
U-boot-UUB> run flash-all  (tftp server required on the network)
Load address: 0x2000000
Loading: #################################################################
#################################################################
#################################################################
#################################################################
946.3 KiB/s
done

device 0 offset 0x0, size 0x8000000
18481152 bytes written, 15073280 bytes skipped in 59.551s, speed 578524 B/s
U-boot-UUB>
```

Now flash memory partitions are updated with a complete petalinux image .

uub-flash.bin is stored into the flash memory

How to update the operative system or bitstream on UUB

To update bitstream, device tree and linux image to the latest version there are two ways:

1. **UUB update by TFTP server. This solution require ethernet connection to the network**

2. **UUB update by USB memory stick. Standalone solution (very useful in the field)**

Insert the USB memory stick to the UUB or connect to the network by LAN

1 - TFTP server:

- Connect UUB to the network by LAN and turn it on
- At command line type: **uub-update** (example for a SDECO server machine 192.168.1.136).
- Command for specific server IP is **uub-update followed by IP number** of your tftp machine
  UUB will download and overwrite the image system and bitstream files on flash memory
- Check the new version number of operative system after reboot

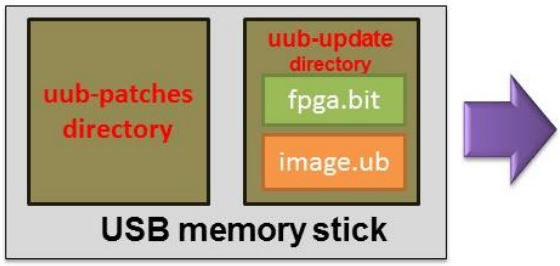
2.A - USB memory stick (very useful in the field):

- Turn off the UUB
- Download **image.ub** and **fpga.bit** from repository and copy the files in the directory **uub-update** into the usb memory stick
- Connect the memory stick to the USB plug of UUB
- Turn on the UUB
- Wait for the booting phase and the patching starts
- When upgrading will be complete a message will inform you about .
- reboot the UUB and check the new version number of bitstream and operative system

2.B - USB memory stick (manual command):

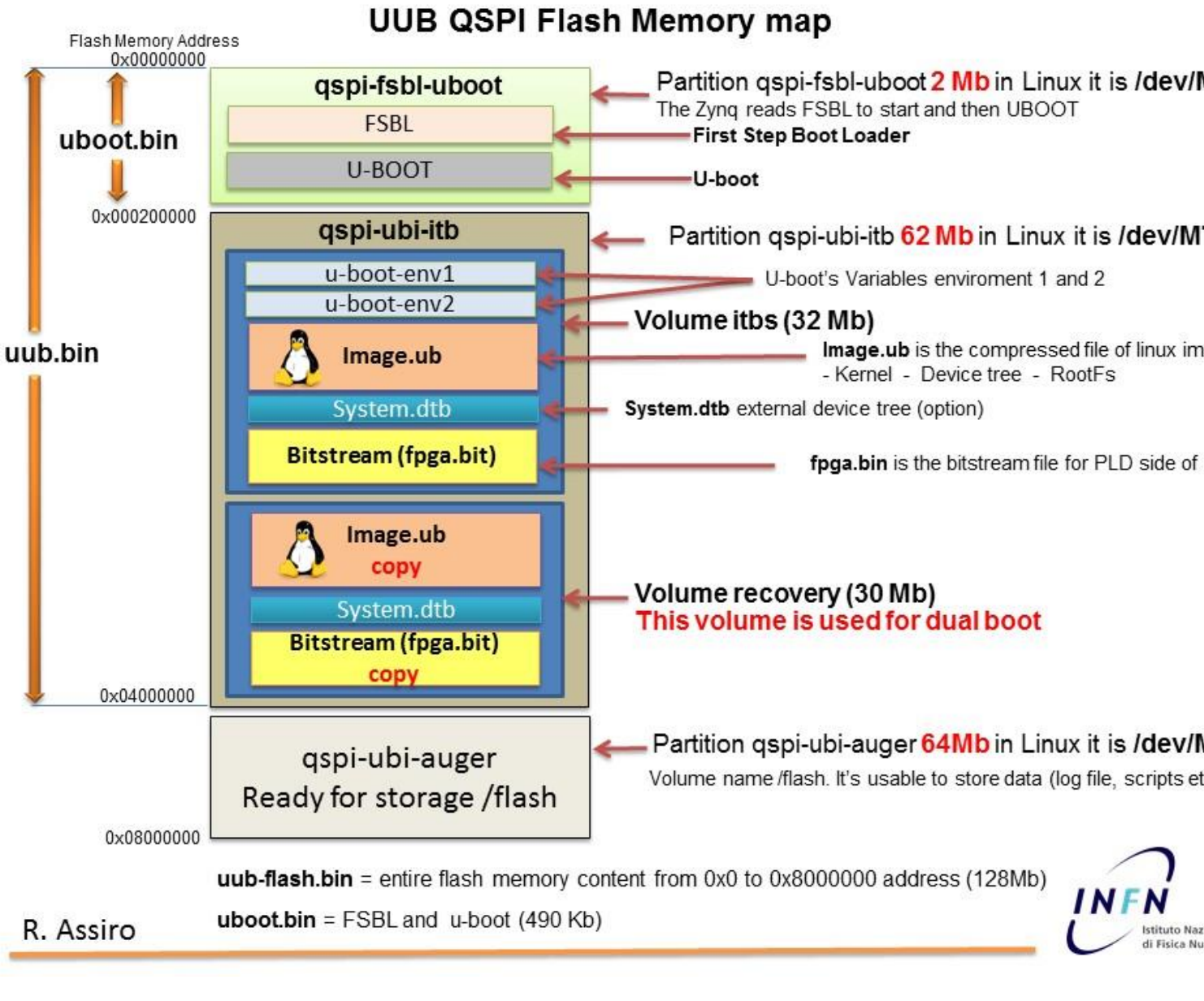
- While the UUB is running insert the USB memory stick to the plug
- At command line digit: **uub-update**
- UUB will transfer the image system and bitstream files from USB memory folder to the flash memory
- reboot the UUB and check the new version number of bitstream and operative system

Important! - The system expects to find the upgrading files (image.ub and fpga.bit) in the memory stick in the directory **uub-update** in order be installed correctly.



UUB's flash memory architecture

Petalinux system is built in a compressed file: **image.ub** The FPGA contents in file **fpga.bit** (bitstream)
Inside image.ub there are: Kernel, device tree and root file system with all programs, scripts and drivers developed
Image.ub is stored into the flash memory in the MTD2 partition in the volume **itbs.**
At any boot the image.ub is decompressed into the RAM and the system can start with all programs built inside.



How to program and upgrade the UUB's operative system and FPGA firmware

Questions

- How it's possible to store into the system image (image.ub) new programs, scripts or settings without re-build the petalinux image?
- Can we update the system by radio communication?

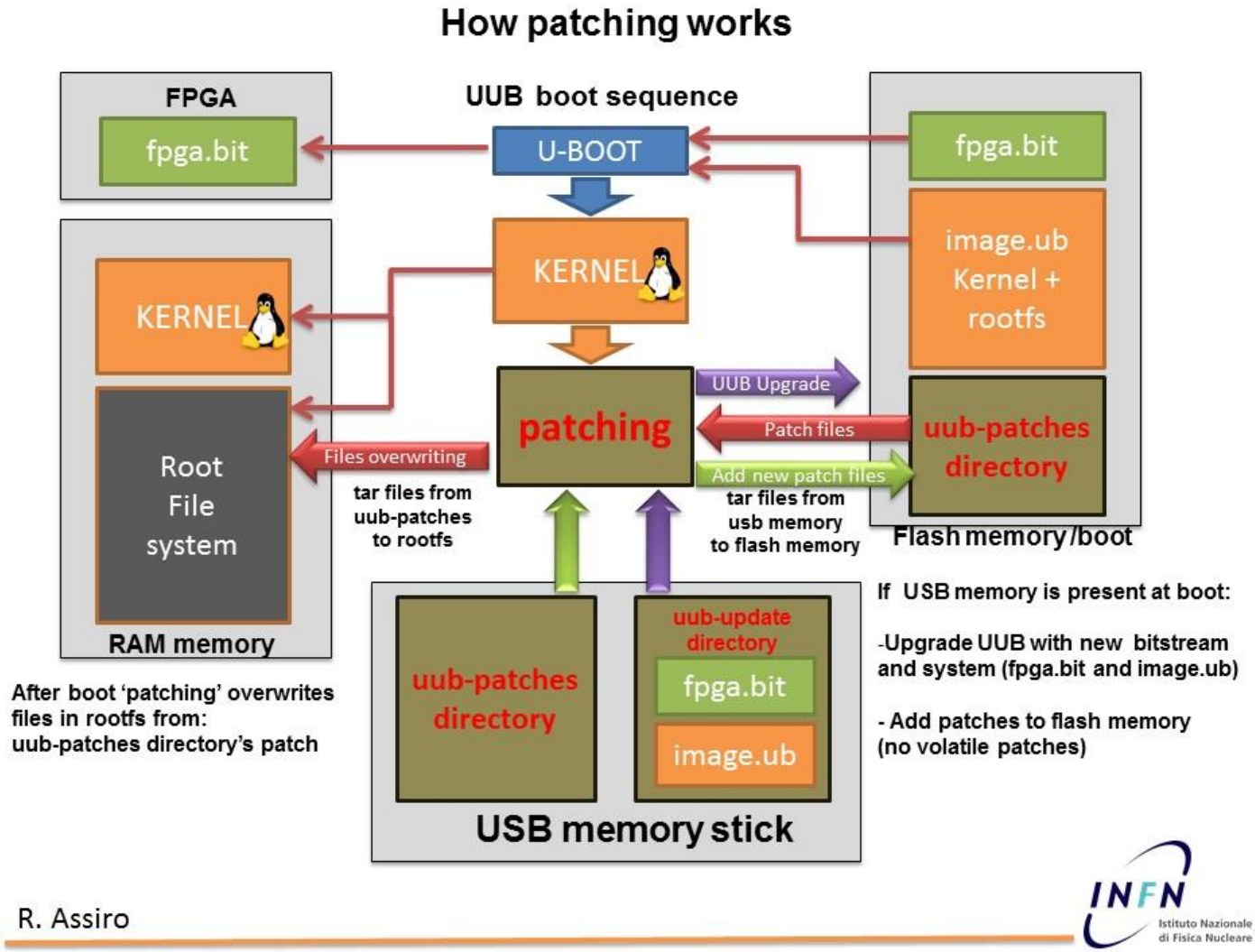
Solution:

The solution is to patching the root file system just after the kernel running.
"Patching" is the script's name which runs at any boot. It controls if into the volume /boot/uub-patches of the flash memory there are files for patching.
The patch file are compressed file type tar (.tgz) (lite to store and also to transfer by UUB's radio communication system).
To patch the system we need to create a compressed file and to save it in the flash memory in the folder **uub-pathes**

"Patching" the solution to add software and settings to the rootfs

"patching" runs after kernel start, it will:

1. write or overwrite files in rootfs from patches contents in flash memory (uub-patches folder)
2. overwrite files image.ub and bitstream from USB memory stick if connected (uub-upgrade folder)
3. if USB memory stick is connected "patching" writes or overwrites files in rootfs from patches contents (uub-patches folder). If patch's name starts with a "**P**", patch file will be stored in flash memory too (permanent patching).
4. if USB memory stick is connected "patching" write or overwrite files in rootfs from patches contents (uub-patches folder). If patch's name starts with "**_P**", patch file will be not stored to flash. (useful function to test software on UUB)

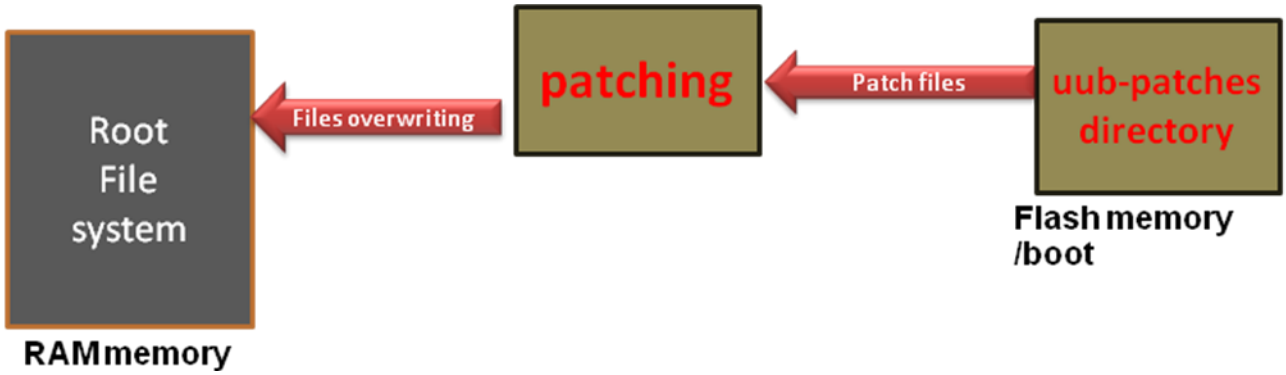


Repository's files for UUB:

- **uboot.bin** First file to transfer in UUB's flash memory. This file contents the image of u-boot only
- **uub.bin** Image of entire flash memory. (48Mb).
- **image.ub** Compressed image of operative system and RootFs
- **fpga.bit** bitstream file for FPGA

The patch

"Patch" is a compressed file which contains programs and scripts with path destination in root file system.
The patch's name must start with **"P"** followed by a sequence number from 00 to 99
In the flash memory the directory containing patch's files is **/boot/uub-patches**
In the USB memory stick the directory containing patch's files is **/usb/uub-patches**
"Patching" is the main script that runs, after Kernel starts, and it can write and overwrite files in rootfs



How build a no volatile patch into flash memory (/boot)

command to create a patch from UUB's command line:

Example:

1. Tranfer your application software in your workspace in file system (example: bin/hello.elf)

2. mount flash memory partition **/boot**:
   root@Auger-uub:~# **mountboot**

3. create tar file from /root:
   root@Auger-uub:~# **cd /**
   root@Auger-uub:~# **tar cvzf /boot/uub-patches/P01hello.tgz bin/hello.elf**

4. unmount partition /boot:
   root@Auger-uub:~# **umountboot**

How build a no volatile patch into USB memory stick

After boot, the patch file will be transferred to flash memory folder if connected
patch directory = uub-patches update directory = uub-update

Example:

1. Tranfer your application software in your workspace in file system (example: bin/hello.elf)
2. insert USB memory and mount it **:**
   root@Auger-uub:~# **mountusb**
3. create tar file from /root:
   root@Auger-uub:~# **cd /**
   root@Auger-uub:~# **tar cvzf /usb/uub-patches/P01hello.tgz bin/hello.elf**
4. unmount device:
   root@Auger-uub:~# **umountusb**

How build a volatile patch into USB memory stick

volatile patch is very useful during developing software

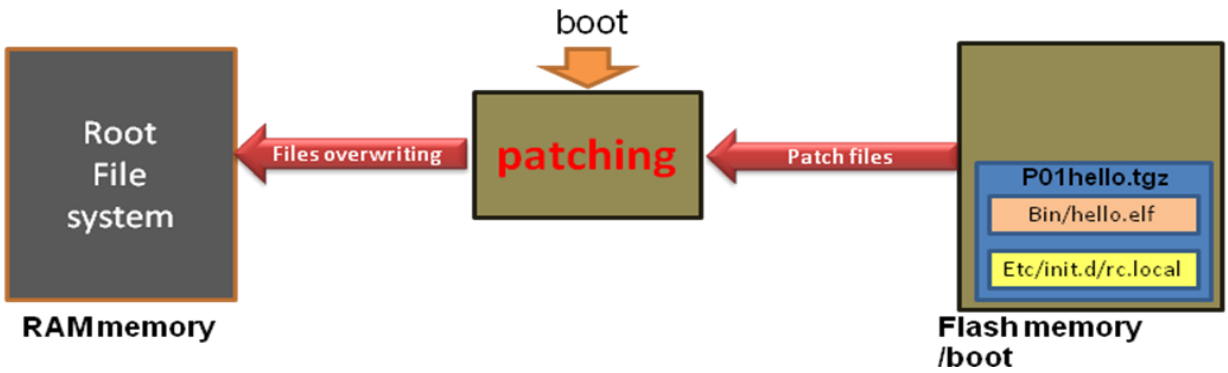
Volatile patch will not stored in flash memory

If patch's name starts with: **_P** the patch will be applied, **but not stored in flash memory**

volatile patch name : **_P01name-patch.tgz**

How to build a patch to run automatically a program at boot

rc.local is a script file to edit to set an auto start process



1. edit and modify rc.local :
   root@Auger-uub:~# **vi etc/init.d/rc.local**

2. insert command a line to run your application (example hello.elf in flash memory)

   #!/bin/bash
   #Please put follow your application programs you want run at boot
   #example: name2run > /dev/null 2>/dev/null &

   hello.elf > /dev/null 2>/dev/null &
   echo "hello is running at boot!"

   exit and save from vi

3. build the patch file :
   root@Auger-uub:~# **cd /**
   root@Auger-uub:~# **mountboot**
   root@Auger-uub:~# **tar cvzf /boot/uub-patches/P01hello.tgz bin/hello.elf etc/init.d/rc.local**
   root@Auger-uub:~# **umountboot**

Reboot the UUB, hello.elf will run automatically after boot

How to connect a laptop to the UUB by LAN in the field

Standard LAN configuration is DHCP (eth0) and alias address **192.168.168.168** (eth0:0)

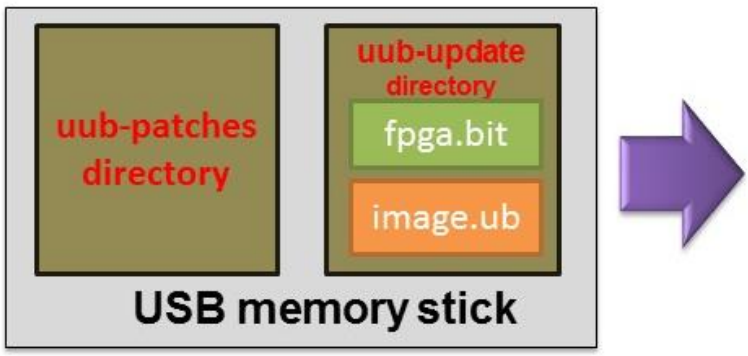
In the field is helful to use the direct connection LAN to LAN

Set your laptop LAN configuration as fixed IP number :

IP address : 192.168.168.1
IP mask : 255.255.255.0

Open your browser and connect to: 192.168.168.168

Upgrade UUB by memory stick (helpful in the field)



Directory: uub-patches and uub-update

uub-patches for tar files for patching

uub-update for image.ub (petalinux system) and fpga.bit (FPGA bitstream)

UUB update sequence:

- Turn off the uub
- Insert the usb memory stick on USB plug
- Turn on the UUB and wait the booting. The patches into the memory stick will be transferred into the flash memory and the system will be update (if update is present into the folder)

How to use the recovery volume

What is the recovery volume?
- recovery volume is the secondary boot volume for the cpu to load the system from the flash memory, in case of corruption of the primary boot image (/boot volume)

In case of system booting problem (/boot volume content compromised), we might try to boot the UUB from the recovery volume.
From u-boot prompt command:

```
U-boot-UUB>  run recovery
```

The system will start from recovery volume in the flash (recovery volume is a backup of first booting area called /boot)

Now is possible to copy bitstream and system image from **recovery** to the **/boot volume**:

```
root@Auger-uub:~#  recovery-boot
```

Restart the UUB at the end of process

The reverse process is:

recovery-update to update recovery contents from the primary /boot volume

UUB TEST - Tools and commands

Test by serial terminal or by lan on http protocol (API):

#	Name of function	Parameter (in/out)	explanation
# 1	get_sn	in: serial number, range [0001…4000] (e.g. 0102 or 1677) out: 48 bit serial number (e.g. 6B-2D-05-01-00-00-D6)	Returns the 48bit internal SN as provided by MSP. It provides as input the simple SN, which can be used to generate a **hostname** like **UUBxxxxV3** for storage in flash memory.
# 2	get_msp_version	out: MSP-software version (e.g. 2.09T or 2.09F)	Returns the MSP software version and the mode (field or test).
# 3a	get_fpga_version	out: FPGA software version (e.g. EA 0.97.8)	Returns the FPGA software version.
# 3b	get_mac_adress	out: UUBs Mac address (e.g. 00:0A:35:00:1E:53)	Returns the UUBs Mac address.
# 4	get_i_v	out: table of voltages and currents names of variable etc. are tbd A proposal will be given in attachment (later)	Requests and returns the latest measured voltages and currents by the MSP.
# 5	HV_test	in: <Max> value for HV DAC, range [1…4095] out: measured voltage change, when DAC is set to 0 and to <Max> for each PMT output 1..6. For example the values could be: 2010, 1960, 2001, 150(error), 1998, 2004	This routine sets all DACs to zero and requests (and stores) the read-back voltages for each PMT output. Then it sets the DACs to <Max> and requests the voltages again. The difference of the two values is returned for the for PMTs.
# 6	GPS_serial	out: test result, string e.g. “Ok Ok” or “failed message”	Verifies the GPS interface (serial I/O only) by sending and read back some characters.
# 7	Radio_serial	out: test result, string e.g. “Ok Ok” or “failed message”	Verifies the Radio interface (serial I/O only) by sending and read back some characters.
# 8	Trig_test	in: <N1>= # of triggers generated, range: [1…255] out: <N2>= # of triggers received The test is passed, if N1 is equal to N2.	This routine generates <N1> pulses at the trigger output and counts the number of pulses received <N2> at the trigger input. The pulse width could be e.g. 1us, pulse distance 10us

(tbd).

# 9	Digital_test	out1: result of interf. #1; I1 out2: result of interf. #2; I1 The result should be 0, if the interface is okay. Values >0 indicate which port(s) is/ are defect.	This routine sends some a bit steam through 4 ports and verifies reception on 4 others. The test is repeated with trans-mitting ports and receiving ports inter-changed for both interfaces.
# 10	DAC_ramp	in: <N> number of repetitions, range [1…1000] in: <A1> amplitude of DAC1 in:<A2> amplitude of DAC2 range[100…4095] ?	This routine generates a ramp signal with a max. amplitude <A1> and <A2> for DACs 1/2. After the ramps the signal is zero (for some ?? ms). The cycle is repeated <N> times.