Factory Acceptance Test ICS

MTCA CRATE

<REPLACE_ITEM_NAME> <REPLACE_ITEM_SERIAL_N^o> <REPLACE_ITEM_TAG>

	Name (Role/Title)
Operator	¡replace_name_person_entitled_to_tests(role/title)¿
Quality Manager	¡replace_name_person_entitled_to_quality(role/title)¿

Operator Signature

Quality Manager Signature

Date: January 31, 2017

Description	Factory Acceptance Test ICS - MTCA crate
Document number	ESS-xxxxxxx
Date	January 31, 2017

1 Hardware Components

The Device Under Test (from now on simply referred by DUT) is a ¡replace_item_name¿ MTCA crate comprising the following components :

Name	ICS-TAG	Vendor Documentation
¡replace_item_name¿	¡replace_item_tag¿	¡replace_item_vendor_documentation¿
MCH	MCH	MCH
MCH	MCH	MCH
MCH	MCH	MCH
MCH	MCH	MCH
MCH	MCH	MCH
MCH	MCH	MCH
MCH	MCH	MCH
MCH	MCH	MCH
MCH	MCH	MCH

1.1 Scope

- This document provides a brief overview of the IFC1211 main features and describes the necessary steps to boot-up a basic system.
- This document provides the information to create and compile a U-boot script to be used at startup to load configuration settings from a server.
- The purpose of this document is to describe the engineering procedure and troubleshooting about how the IFC1211 board will be booted up and how to import the ESS EPICS Environment (EEE).

Note that this is a very early draft document and should be updated as development progresses.

1.2 Target Audience

This document is targeted to ICS engineers and technical stakeholders of the ESS IOCs.

2 System Description

The IOxOS IFC1211 is a VME64x compliant board that is currently being used to develop EPICS IOCs because its hardware mimics that of the microTCA.4 AMC cards currently under development. The IFC1211 embeds a T2081 QorIQ 64bit PowerPC manufactured by Freescale, a 32NT24 PCIexpress switch and 3 FPGAs: a XC7A75 Artix 7 (PON), a XC7A35 Artix 7 (IO) and a XCKU040 Kintex Ultrascale.

The PON FPGA is preconfigured and is supposed to manage both the programming procedure of the other two FPGAs and the reset signal for the T2081. The "IFC_1211 Hardware Technical User's Guide"[1] describes the board capabilities more in depth and also the different configurations that might be set and loaded by the board at bootstrap, it also describes some added functionalities to the common U-boot distribution that allow to check and modify bitstream images for the programmable logics so that the PON FPGA can program them at startup, all of these informations are provided in the pages from 77 to 80.

This document describes how to set up an IFC1211 board to be able to boot from a NFS mounted file system.

2.1 IFC1211

Figure 1 shows the rough physical dimensions 233×160 mm^2 of the VME IFC1211 card.

Figure 1 IFC1211-A1 VME64x board with FMC and XMC mezzanine cards support.

Figure 2 describes the position of the various components on the board, it is important to take note of the position of the dip switches (SWxxx) as they need to be configured before providing power to the board.

Figure 2 Dip Switches and Serial interface locations on the IFC1211.

The block diagram shown in Figure 3 represents the interconnection scheme amongst the different components.

As can be seen the T2081 has direct access to all the 3 RJ45 connectors on front panel, two of those are actual gigabit Ethernet connections while the remaining one is a serial link. 2GB of ECC DDR3-2133 memory are provided to the T2081 PowerPC processor and one additional GB is made available to the Kintex Ultrascale FPGA.

The board is available in two different configurations depending on the mezzanine cards that it is supposed to host, the IFC1211-A0 version is configured to support up to two FMC HPC cards while the -A1 version is tailored for a FMC and a XMC cards. The differential connections of both the FMCs are routed to the Xilinx Ultrascale FPGA. Additional 4 XCVR lanes per slot are provided and routed to the same FPGA MGT blocks.

Description	Factory Acceptance Test ICS - MTCA crate
Document number	ESS-xxxxxxx
Date	January 31, 2017

Figure 3 IFC1211-Ax VME64x Block Diagram.

Operator ¡replace_name_person_entitled_to_tests(role/title)¿	Operator Signature	Quality Manager Signature
Quality Manager ¡replace_name_person_entitled_to_quality(role/title)¿		

3 System Environment

Before describing the engineering procedure necessary to boot up the IFC1211 VME board, it is necessary to have the following list of hardware and software components. This chapter will cover the hardware and software requirements and will explain how to access the infrastructure already available in the ICS lab at ESS. Additional informations on such infrastructure can be found in the following ESS wiki documents:

- General description[2];
- How to setup lab infrastructure[3];
- Set-up of ESS boot machine[4].

3.1 Hardware

Table 1 shows the list of hardware necessary to boot up the board.

Hardware	**Info**
IFC1211	IOxOS VME64x board
VME Crate	6U size B capable crate
Serial interface	
Serial cable	see Table 4 and [1] for the pinout of IFC1211 serial interface
Desktop or Server PC	

Table 1 Hardware List

It is assumed that a computer providing NFS and TFTP server functionalities is available and is configured properly to allow incoming connections from the IFC1211 board, if this is not the case an overview of the procedure to set up such a machine is defined in the "How to setup lab infrastructure"[3] and "Set-up of ESS boot machine" [4].

3.1.1 Hardware at ICS lab

The list in Table 2 shows the hardware available in the ICS lab, the user is not bound to use the same components but he should be aware that, despite many devices provide similar capabilities, it might be necessary to modify the instructions contained in this Engineering Manual according to the informations provided in the manufacturers datasheet or to the software revision number.

Hardware	Info	Serial Number
IFC1211	`ICS TAG-382` or `ICS TAG-383`	16370107 or 16370109
VME Crate	Hostname : icsb-vmeifc1210-528 or icsb-vmeifc1210-529	
Moxa Serial Device Server	Hostname : ics-moxabox-01	
Serial cable	see Figure 4 and Table 4	
EEE NFS Server		

Table 2 Hardware List

3.1.2 IFC1211 Dip Switches Setting

Before powering up the board it is necessary to ensure that the Dip switches whose value is read at bootstrap are configured in the right way. The location of these switches on the board is depicted in Figure 2. Their position should be set as follows:

IFC1210 Switch setting								
Switch Designator	Switch Lever n°							
	1	2	3	4	5	6	7	8
SW100 :	0	0	0	0				
SW101 :	1	0	0	0	0	0	0	0
SW102 :	0	1	1	0	0	0	0	0

Table 3 Dip Switches Setting

3.1.3 Serial Connection: Pinout and Cable Wiring

In order to access the boot-loader shell the user shall have access to the serial interface. The Serial link to the IFC1211 T2081 processor is routed through a standard RJ45 connector (see [1] sections 2.1.2 and 2.6.7) so the user shall take care of creating a cable that matches the board pinout to his equipment.

The ICS group uses a Moxa serial device server[5] to access RS232 interfaces but there is no off-the-shelf cable that matches its pinout to that of the IFC1211 board so the user must create his own cable according to the following instructions.

Figure 4 shows the usual cable enumeration and Table 4 shows the required connections at both ends assuming that the cable end at the Moxa NPort device follows the T568B coloring scheme shown.

The white and orange stripes cable is not routed to any pin on the IFC1211 side and is actually left dangling, in addition one of the GND pins is not connected but so far no problem was detected related to this.

Figure 4 Ethernet T568B cables enumeration[6].

	Moxa NPort 5610 (ics-moxabox-01) see [5] section A-3		**IFC1211** see [1] section 2.6.7	
Pin n°	Color	Description	Color	Description
1	white/orange stripe	DSR (in)	green solid	Not used (DCD)
2	orange solid	RTS (out)	orange solid	RTS
3	white/green stripe	GND	white/green stripe	GNDC
4	blue solid	TxD (out)	white/blue stripe	TXD
5	white/blue stripe	RxD (in)	blue solid	RXD
6	green solid	DCD (in)	none (do not connect to white/orange cable)	GNDC
7	white/brown stripe	CTS (in)	white/brown stripe	CTS
8	brown solid	DTR (out)	brown solid	Not used (DTR)

Table 4 Serial cable connections

3.2 Software

Table 5 shows the list of software and services required. It is not necessary to have the same TFTP or NFS server versions to be able to boot the IFC1211 board, but there might be a few differences in terms of their configuration.

Item	Version and Info.
TFTP server	`tftp-hpa 5.2`
NFS server	NFS protocol 3 required
uImage_2080	currently version `3.12`
t2080ifc.dtb	N.A.
root file system .tar.gz	fsl-image-full-t2080rdb-64b.tar.gz

Table 5 Software and its version information.

The file `uImage_2080` is a Linux kernel image binary file with the U-boot boot-loader wrapper. The `t2080ifc.dtb` is the device-tree binary file that provides informations about the available devices, their access address and the compatible drivers. The root file system is provided as a tarball and needs to be uncompressed before it can be used, further instructions are provided later on. An additional file named `u-boot-t2080.bin` might be available, this file contains the U-boot boot-loader image which should already be loaded into the flash memory of the board and as such it is not required for the tasks described in this document. If the boot-loader is not already installed or if it needs to be updated please refer to the relevant documentation.

3.2.1 Software at ICS lab

Currently at ICS both the TFTP and NFS services are provided by the `ics-boot-01.esss.lu.se` server located in the ICS lab. This machine has the following setup:

Item	Version and Info.
CentOS Linux	`7.1.1503`
Kernel	`3.10.0-229.7.2.rt56.141.6.el7.centos.x86_64`
TFTP server	`tftp-hpa 5.2`
NFS server	NFS protocol 3 and 4

Table 6 Software and its version information.

The files required to boot up the IFC1211 are the same as described in the previous section. The procedure to set up a machine with the same characteristics as defined in Table 6 can be found in [3] and [4].

4 Engineering Procedure

This chapter provides the information necessary to boot the IFC1211 board. The modifications required to the U-boot environment in order to automatically load a script will be described together with the steps necessary to compile such a script. All the procedures will assume that the user have access to the `ics-boot-01.esss.lu.se` machine, available in the ics-lab network.

4.1 U-boot Script File

The U-boot script file is used to modify the boot loader environment variables in order to define the booting procedure of the system.

4.1.1 Default Environment and System Access

It might be useful to take a look at the default environment settings of a system before editing them. To be able to access the U-boot shell we need to connect to the serial interface of the IFC1211.

Figure 2 shows the location of the serial interface on the IFC1211 board.

The user must configure an available serial device port according to the values in Table 7.

Parameter	Value
Baud rate	155200
Data bits	8
Stop bits	1
Parity	None
Flow control	None
FIFO	Enable
Interface	RS-232 Only

Table 7 Serial Device port configuration

There are many options available to connect to a serial device depending on the hardware available, for example the PC could provide a serial interface or the user can purchase a serial to USB converter. In any case a cable should be available to interface between the two components and the IFC1211 end connection should match the one described in Table 4.

The following subsection will cover the configuration used at ICS.

Moxa Serial Device Configuration

The user at ICS can setup a serial connection to the IFC1211 board using the Moxa Serial Device Server (Hostname : `ics-moxabox-01` in the ics-lab network) and the cable previously defined in this document.

To configure such a connection the user should open a web browser and access the Moxa configuration web page (`http://10.4.3.90/`), in the "Serial Settings" menù an available port number amongst the set of available ones should be selected and its settings shall be modified to reflect the values previously described. When the device is properly configured the appropriate end of the Ethernet cable can be connected to the IFC1211 and to the selected Moxa Serial Device Server port.

It is possible to access the serial device in many different ways, all of which are selected from the "Operating Settings" section of every port. The most common settings are:

- **Real COM Mode:** The driver required to interface to the MoxaBox is already installed on the `ics-workstation (10.4.3.18)`, if you need to setup the communication to this device on your pc the relevant informations are described in the ESS wiki in the section "How To Setup Serial Communication to Moxa NPort 5610 on Your Machine" of the "Accessing Serial Devices Located in the Lab"[7] document. To communicate with the serial interface of the device the user shall `ssh` into the `ics-workstation` and use a terminal emulation program like the picocom tool. Port enumeration in Linux starts from 0 so the `ttyr` number to connect to is the port number written on the Moxa chassis minus one.

  ```
  [srvr-machine@localhost ~]$ picocom -b 115200 /dev/ttyr1
  ```

  The `-b` option allows to specify the baudrate of the connection if it needs be.

- **TCP Server Mode:** Allows the user to connect to the device using a telnet client connecting to the Moxa IP address and to the port number specified in the Local TCP Port field.

The U-boot shell should now be displayed and the `printenv` command will return the environment variables used at boot time. Here follows the default environment setting for the IFC1211 connected on port number 9 of the MoxaBox.

```
[srvr-machine@localhost ~]$ picocom -b 115200 /dev/ttyr8
picocom v1.7

port is        : /dev/ttyr8
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv
```

```
imap is        :
omap is        :
emap is        : crcrlf,delbs,

Terminal ready

=>
=> printenv
baudrate=115200
bdev=sda3
bootargs=root=/dev/ram rw console=ttyS0,115200
bootcmd=setenv bootargs root=/dev/ram rw console=$consoledev,$baudrate $othbootargs;setenv
     ramdiskaddr 0x02000000;setenv fdtaddr 0x00c00000;setenv loadaddr 0x1000000;bootm $loadaddr
     $ramdiskaddr $fdtaddr
bootdelay=-1
bootfile=uImage_2080_3.12
consoledev=ttyS0
eth1addr=7c:dd:20:10:01:19
ethact=FM1@DTSEC4
ethaddr=7c:dd:20:10:01:18
ethprime=FM1@DTSEC4
fdtaddr=2000000
fdtfile=t2080ifc_new.dtb
fileaddr=2000000
filesize=8baf
fman_ucode=eff00000
gatewayip=192.168.1.1
hostname=IFC1211_101
hwconfig=fsl_ddr:ctlr_intlv=cacheline,bank_intlv=auto;usb1:dr_mode=host,phy_type=utmi
ipaddr=192.168.1.138
loadaddr=1000000
netdev=eth1
netmask=255.255.255.0
nfsboot=setenv bootargs root=/dev/nfs rw nfsroot=$serverip:$rootpath ip=$ipaddr:$serverip:
     $gatewayip:$netmask:$hostname:$netdev:off console=$consoledev,$baudrate $othbootargs;tftp
     $loadaddr $bootfile;tftp $fdtaddr $fdtfile;bootm $loadaddr - $fdtaddr
ramboot=setenv bootargs root=/dev/ram rw console=$consoledev,$baudrate $othbootargs;tftp
     $ramdiskaddr $ramdiskfile;tftp $loadaddr $bootfile;tftp $fdtaddr $fdtfile;bootm $loadaddr
     $ramdiskaddr $fdtaddr
ramdiskaddr=5000000
ramdiskfile=ramdisk.uboot
rootpath=/data/T2081/rootfs_2080_1.9
serverip=192.168.1.57
stderr=serial
stdin=serial
stdout=serial
tftpflash=tftpboot $loadaddr $uboot && protect off $ubootaddr +$filesize && erase $ubootaddr +
     $filesize && cp.b $loadaddr $ubootaddr $filesize && protect on $ubootaddr +$filesize && cmp.b
      $loadaddr $ubootaddr $filesize
uboot=u-boot-t2080.bin
ubootaddr=0xeff40000

Environment size: 1648/8188 bytes
=>
```

Currently the default environment that can be found on any pristine board does not allow the user to load a U-boot script at startup, in order to be able to load one the user needs to modify it. The next section will show how to create the script file and the following one will describe how to configure the boot loader to load a script file.

4.1.2 U-boot Script Creation and Compilation

The U-boot script is just a plain text file that will afterwards be compiled in order to be recognized by the boot loader, as such it can be created using any text editor and does not need to have any specific file extension. To avoid any misunderstanding we will append the extension .scr on the compiled script and .script on the regular text file. The script can contain any command that could also be issued from the boot loader shell, to view the list of available commands type either `help` or `?`.

```
=>
=> ?
?       - alias for 'help'
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
bootelf - Boot from an ELF image in memory
bootm   - boot application image from memory
bootp   - boot image via network using BOOTP/TFTP protocol
bootvx  - Boot vxWorks from an ELF image
chpart  - change active partition

...(omissis)...

nfs     - boot image via network using NFS protocol
nm      - memory modify (constant address)
pci     - list and access PCI Configuration Space
ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
prom    - PROM handling
protect - enable or disable FLASH write protection
reginfo - print register information
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
sata    - SATA sub system
saveenv - save environment variables to persistent storage
setenv  - set environment variables
setexpr - set environment variable as the result of eval expression
sf      - SPI flash sub-system
showvar - print local hushshell variables
sleep   - delay execution for some time
source  - run script from memory
test    - minimal test like /bin/sh
tftpboot- boot image via network using TFTP protocol
true    - do nothing, successfully
usb     - USB sub-system
usbboot - boot from USB device
vdd_override- override VDD
vdd_read- read VDD
version - print monitor, compiler and linker version
=>
```

In order to be able to boot up the IFC1211 we need to define its network settings, specify the necessary files locations and also provide some basic boot arguments. The files required, as previously defined, are:

- uImage_2080, the linux kernel image;
- t2080ifc.dtb, the compiled device tree binary;

- fsl-image-full-t2080rdb-64b.tar.gz, the root file system structure (this file needs to be extracted).

At the time of this document these files are provided by IOxOS/PSI. In addition the following FPGA bitstream files can be loaded at startup and should be also provided:

- io.bit, the IO FPGA bitstream configuration file;
- central.bit, the CENTRAL FPGA bitstream configuration file.

At the time of this document these files are not yet available.

All of these files need to be copied into the export directory of an available server providing TFTP and NFS functionalities, it is not necessary to create a folder structure identical to the one available at ICS but the user must take care of changing all the paths in the scripts and in the U-boot environment to match his own configuration. The following sections will assume that the server has the same directory structure as described hereafter.

In the ICS lab the `ics-boot-01.esss.lu.se` machine provides all the required services, its export folder is `/export` and its directory structure is as follows

```
[username@ics-boot-01 export]$ tree -d -L 2
.
        boot
                EFI
                images
                pxelinux.cfg
        epics
                bases
                modules
        images
                ifc1210-default
                ifc1210-maurizio
                ifc1210-rflps
                ifc1210-scope
                ifc1211-default
                mtcacpu-default
        modules
                centos7
        nfsroots
                centos7
                ifc1210
                ifc1211
        sandbox
                bases
                modules
        sb-startup
                ioc
        sdk
                eldk-5.6
        startup
        boot
        ioc
```

In order to keep the files organized create a folder named `ifc1211-default` in the `/export/images` directory and copy the `uImage_2080` and `t2080ifc.dtb` files into it, then create a folder called `ifc1211` into `/export/nfsroots` and untar the root file system

into it. If the user has the io.bit and central.bit files available he should copy them to the same folder of the kernel image and the device tree (`/export/images/ifc1211-default`).

We are now ready to define the content of the U-boot script which should be as follows:

```
setenv fpgaIO /images/ifc1211-default/io.bit
setenv fpgaCE /images/ifc1211-default/central.bit
setenv bootfile images/ifc1211-default/uImage_2080
setenv fdtfile images/ifc1211-default/t2080ifc.dtb
setenv serverip 194.47.240.7
setenv rootpath /export/nfsroots/ifc1211/rootfs
setenv nfsver nfsvers=3
setenv ipaddr 10.4.3.28
setenv gatewayip 10.4.0.1
setenv netmask 255.255.252.0
setenv hostname IFC1211_101
setenv netdev eth0
setenv consoledev ttyS0
setenv baudrate 115200
setenv othbootargs rootdelay=1 earlyprintk
setenv bootargs root=/dev/nfs nfsroot=$serverip:$rootpath,$nfsver ip=$ipaddr:
    $serverip:$gatewayip:$netmask:$hostname:$netdev:off console=$consoledev,
    $baudrate $othbootargs ramdisk=60000 pci_scan_all pci=realloc
setenv nfsboot 'setenv bootargs $bootargs;tftp $loadaddr $bootfile;tftp $fdtaddr
    $fdtfile;bootm $loadaddr - $fdtaddr'
setenv loadIO 'tftp $loadaddr $fpgaIO;fpga load io $loadaddr'
setenv loadCE 'tftp $loadaddr $fpgaCE;fpga load central $loadaddr'
setenv fpgaload 'run loadIO; run loadCE; fpga reset 11b01'
```

The first two and the last three lines are optional and should be included only if the bitstream files have been provided. The ethernet related variables (`serverip`, `ipaddr`, `gatewayip` and `netmask`) should be set to proper values according to the network configuration and IP availability. This file will not be recognized by U-boot unless we translate it into a proper script, in order to do so copy it to a folder in the `ics-boot-01.esss.lu.se` machine where the user has writing privileges, here it is assumed to be the user home directory, and issue the following command:

```
[username@ics-boot-01 ~]$ /home/shared/SVN/Software/IOxOS/u-boot/setenv/mkimage -T script -C none
    -n 'Boot Script' -d uboot_ifc1211_text.script uboot_ifc1211.scr
```

where `uboot_ifc1211_text` is supposed to be the name of text file created in the previous step and `uboot_ifc1211` is the desired name for the compiled script.

If the `mkimage` executable is not available in the location shown above a way to find it is to run the following command.

```
[username@ics-boot-01 ~]$ find / ! -readable -prune -o -path *u-boot* -name mkimage -print
```

This will display all available files named `mkimage` that are located in a folder with `u-boot` in its path.

The generated `.scr` file has to be copied in the `/export` folder in order to be retrieved by U-boot using the TFTP service, in the remainder of this document it will be assumed that this file has been copied in `/export/boot`.

4.1.3 U-boot environment setup to load script

The generated script is now ready to be retrieved and loaded from the server. The user must instruct U-boot to load the file and this is done by modifying the default Environment. There are two different procedures available depending if the system relies on a DHCP server to acquire the network informations or not.

Procedure without DHCP

When the system does not rely on DHCP service to gain network informations all of the `serverip`, `ipaddr`, `gatewayip` and `netmask` have to be provided by the user in order to boot up the board properly. These values need to be set according to the network setup and IP availability, in the following section the proper values for the ICS lab network are provided. The `ipaddr` can be changed to any available value and the list of IP addresses currently in use can be found in [8].

Connect to the IFC1211 board serial interface, as described previously, and type the following commands:

```
[srvr-machine@localhost ~]$ picocom -b 115200 /dev/ttyr8
picocom v1.7

port is        : /dev/ttyr8
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv
imap is        :
omap is        :
emap is        : crcrlf,delbs,

Terminal ready

=> setenv serverip 194.47.240.7
=> setenv ipaddr 10.4.3.28
=> setenv gatewayip 10.4.0.1
=> setenv netmask 255.255.252.0
=> setenv netdev eth0
=> setenv bootcmd 'run envload'
=> setenv envload 'if tftp $loadaddr $serverip:boot/env-$eth0addr.scr; then source $loadaddr &&
     run fpgaload && run nfsboot; elif tftp $loadaddr $serverip:boot/uboot_ifc1211.scr; then
     source $loadaddr && run fpgaload && run nfsboot; else false; fi'
=> saveenv
```

The next section describes the procedure to boot the board using the configuration provided by a DHCP server.

Procedure with DHCP

Connect to the IFC1211 board serial interface, as described previously, and type the following commands:

```
[srvr-machine@localhost ~]$ picocom -b 115200 /dev/ttyr8
picocom v1.7

port is        : /dev/ttyr8
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv
imap is        :
omap is        :
emap is        : crcrlf,delbs,

Terminal ready

=> setenv netdev eth0
=> setenv bootcmd 'dhcp; run envload'
=> setenv envload 'if tftp $loadaddr $serverip:boot/env-$eth0addr.scr; then source $loadaddr &&
     run fpgaload && run nfsboot; elif tftp $loadaddr $serverip:boot/uboot_ifc1211.scr; then
     source $loadaddr && run fpgaload && run nfsboot; else false; fi'
=> saveenv
```

The user still needs to specify the Ethernet interface that is connected to the LAN, here is the `eth0` device. If the same settings provided by the DHCP server should be retained by the board after boot up an additional modification has to be made on the script itself, the `bootargs` must be modified to look like this

```
setenv bootargs root=/dev/nfs nfsroot=$serverip:$rootpath,$nfsver ip=dhcp console=
    $consoledev,$baudrate $othbootargs ramdisk=60000 pci_scan_all pci=realloc
```

and the new script must be recompiled.

The `envload` instruction translates to the following list of actions:

- if a file named env-$eth0addr.scr is found on the server and can be loaded at the $loadaddr address in memory then:
  - source the address where the file was copied, which means to import and to execute the contained instructions;
  - run the instruction called `fpgaload`;
  - run the instruction called `nfsboot`;

- otherwise if a file named uboot_ifc1211.scr is found on the server and can be loaded at the $loadaddr address in memory then:
  - source the address where the file was copied, which means to import and to execute the contained instructions;
  - run the instruction called `fpgaload`;
  - run the instruction called `nfsboot`;
- otherwise if none of the previous two conditions apply U-boot will stop the boot procedure and will wait for the user to provide instructions via the serial interface.

The `$loadaddr` variable is defined in the default U-boot environment and translates to a memory address. The `env-$eth0addr.scr` is introduced to reflect a possible approach for a default naming scheme and translates to a file name that includes the MAC address of the network interface `eth0`, in this way every script file will be associated to a single board.

If the bitstream files are not available the instruction `&& run fpgaload` shall be removed from the previous command. The IFC1211 can now be booted up by issuing the `boot` command in the U-boot shell at every startup, to avoid the necessity to interact with the board every time it resets or power cycles the following additional modification can be made to the environment:

```
[srvr-machine@localhost ~]$ picocom -b 115200 /dev/ttyr8
picocom v1.7

port is        : /dev/ttyr8
flowcontrol    : none
baudrate is    : 115200
parity is      : none
databits are   : 8
escape is      : C-a
local echo is  : no
noinit is      : no
noreset is     : no
nolock is      : no
send_cmd is    : sz -vv
receive_cmd is : rz -vv
imap is        :
omap is        :
emap is        : crcrlf,delbs,

Terminal ready

=> setenv bootdelay 5
=> saveenv
```

This will make U-boot wait for 5 seconds at every startup to see if the user wants to stop the booting procedure otherwise it will run the boot command.

Bibliography

[1] Joël Bovier. *IFC_1211 Hardware Technical User's Guide*, September 29, 2016. URL ./IFC1211_HW_UG_A0_20160928.pdf.

[2] Niklas Claesson. General description, 2015. URL https://ess-ics.atlassian.net/wiki/display/HAR/General+description.

[3] Niklas Claesson. How to setup lab infrastructure, 2016. URL https://ess-ics.atlassian.net/wiki/display/HAR/How+to+setup+lab+infrastructure#Howtosetuplabinfrastructure-PrepareforIFC1210boot.

[4] Jeong Han Lee, Simone Farina. *N.A.*, N.A.

[5] MOXA. Nport 5000 series users manual, 2016. URL http://www.moxa.com/doc/man/NPort_5000_Series_UM_e2.0.pdf.

[6] Tia/eia-568. URL https://en.wikipedia.org/wiki/TIA/EIA-568#T568A_and_T568B_termination.

[7] Niklas Claesson, Jeong Han Lee. Accessing serial devices located in the lab, 2016. https://ess-ics.atlassian.net/wiki/display/HAR/Accessing+Serial+Devices+Located+in+the+Lab#AccessingSerialDevicesLocatedintheLab-HowToConnecttoaSerial

[8] Ursa Rojec. Development hardware: Ips and availability, 2016. URL https://ess-ics.atlassian.net/wiki/display/HAR/Development+Hardware%3A+IPs+and+availability#.

Operator ¡replace_name_person_entitled_to_tests(role/title)¿	Operator Signature	Quality Manager Signature
Quality Manager ¡replace_name_person_entitled_to_quality(role/title)¿		