MANUFACTURING TOOLS FOR i.MX APPLICATIONS PROCESSORS

AMF-AUT-T2324

BRYAN THOMAS

FIELD APPLICATION ENGINEER

SECURE CONNECTIONS FOR A SMARTER WORLD

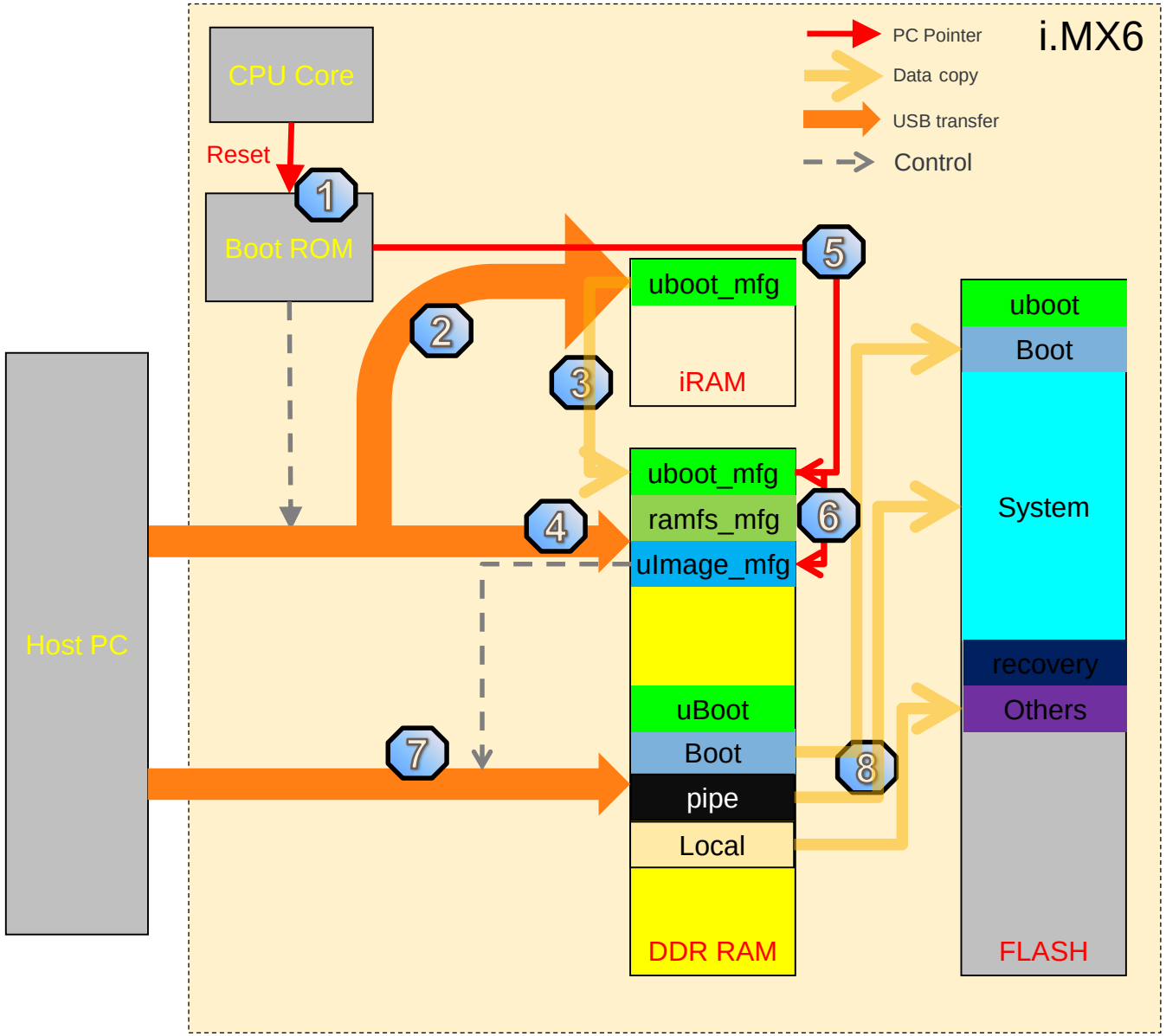
What is in the Mfgtool

- What is the tool used for
  - To burn your own firmware, demo files and other images to storage media (NOR, Nand, SD, etc.).
  - Debug tool in early stage of a project
    - download & execute uboot from RAM…
  - Recover a “brain dead” board
  - Program OTP bits/fuses
  - Etc.?

Work flow

1. Core reset. PC Jump to Boot ROM if configured as download mode or empty boot device.
2. ROM enumerate USB as HID with host, download uboot_mfg from host and store it into iRAM.
3. ROM parse IVT/DCD in uboot_mfg and init the DDR. Copy the body of uboot_mfg to DDR.
4. Boot ROM continue to download uImage_mfg and ramfs_mfg from PC to DDR.
5. PC send “jump” command to ROM. ROM then hand over execution to uboot_mfg in DDR.
6. Uboot_mfg continue to boot into uImage_mfg with ramfs_mfg configured.
7. uImage_mfg re-enumerate USB as UMS devices and download all images from host to DDR .
8. Write images from DDR to flash (dd/pipe) or do some local action with flash such as format.

(*red text is flexible user steps)



Building the “normal” code

- **Use Yocto, etc. to build final uboot, linux, dtb and rootfs images for the board.**
  - These will be the “final” images that the board will ultimately run with containing any user applications, etc.
    - uboot
    - kernel
    - dtb
    - rootfs
  - See NXP Yocto Project User's Guide, Rev. L3.14.28_1.0.0-ga, 04/2015 - *NXP_Yocto_Project_User’s_Guide.pdf* for specific instructions.

Building for mfg. tool**

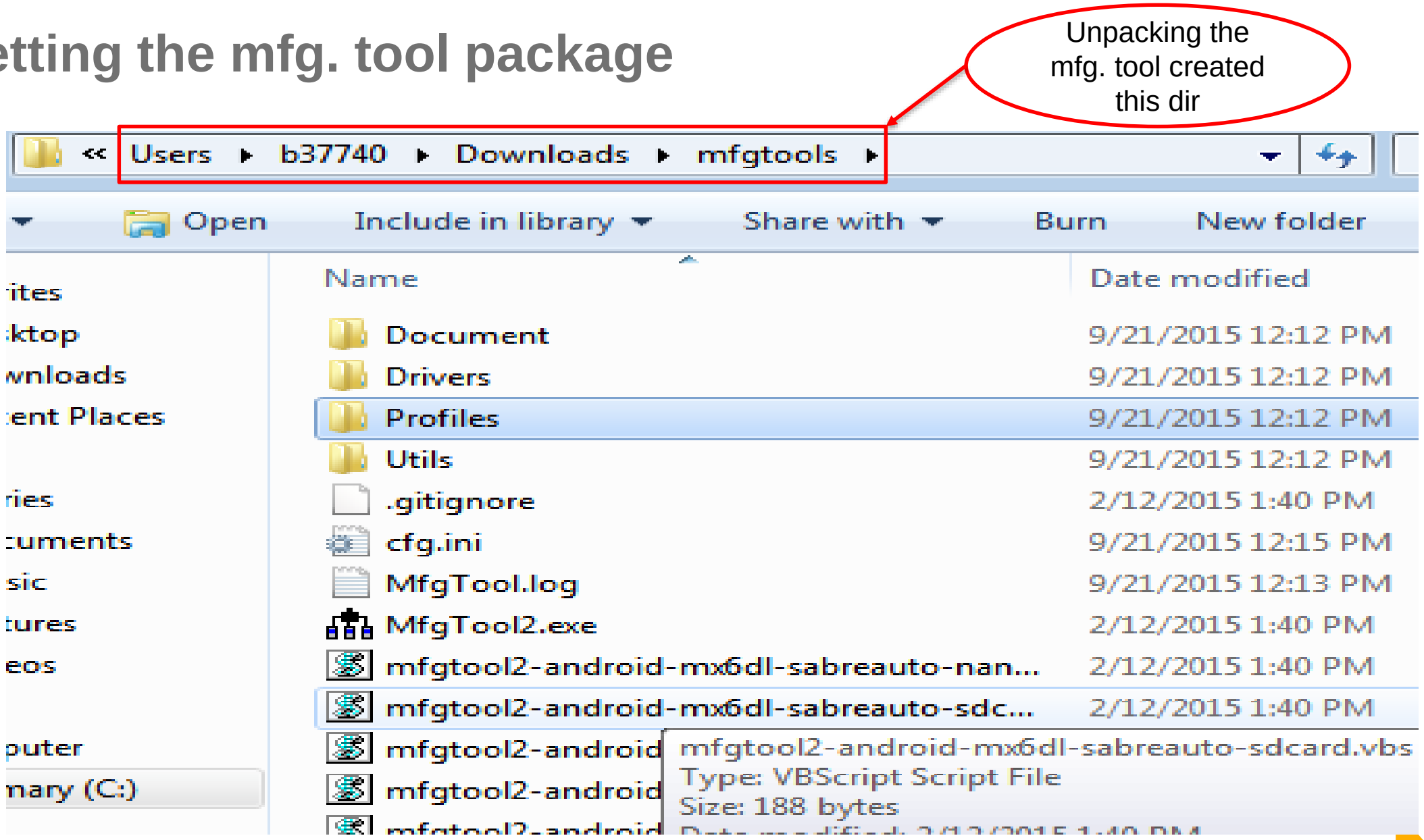
- **6.2 Manufacturing Tool, MFGTool**
  - To build a manufacturing image do the following – this linux will talk to the mfg. tool during reprogramming.
    - $ bitbake fsl-image-mfgtool-initramfs
    - A manufacturing tool kernel is built using the imx_v7_mfg_defconfig while the default kernel is built by using the imx_v7_defconfig and builds linux-imx-mfgtool and u-boot-mfgtool.
      - This is handled automatically by the MFGTool recipes listed above.
  - Creates (in tmp/deploy/images/imx6qsabreauto):
    - u-boot-imx6qsabreauto-mfgtool-2014.04-r0.imx
    - zImage_mfgtool
    - zImage-mfgtool--3.14.28-r0-imx6q-sabreauto-20150915175120.dtb (plus a few other dtb files)
    - fsl-image-mfgtool-initramfs-imx6qsabreauto-20150915175120.rootfs.cpio.gz.u-boot

****NXP Yocto Project User's Guide, Rev. L3.14.28_1.0.0-ga, 04/2015 -*NXP_Yocto_Project_User's_Guide.pdf***

Getting the mfg. tool package that runs on the PC

- Go to http://www.NXP.com/imx.
- Choose an imx6 (q, solo, etc.)
- Look in the Software & Tools tab.
- Hardware Development Tools
  - Click Programmers (Flash, etc.)
- I downloaded this one: IMX6_L3.14.28_MFG_TOOL
- I got this file after download: imx-3.14.28_1.0.0_ga-mfg-tools.tar.gz
- Unpack it to get these files
  - mfgtools-without-rootfs.tar.gz
  - mfgtools-with-rootfs.tar.gz
- Unpack mfgtools-with-rootfs.tar.gz to get directory: mfgtools/

Getting the mfg. tool package



Main files & directories used with mfg. tool.

- **Document**
  - Contains all documentation files for the installed version of the mfg. tool.
  - mfg. tool documentation files mentioned in these slides will be found in this directory.
- **MFGTool2.exe**
  - Main program. Uses all other files and provides GUI.
- **cfg.ini**
  - let MFGTool2 know where it can get the ucl2.xml
  - Defines which operation list should be used for next programming session
  - Changes to this file only take effect when the MfgTool2.exe is started again (close and re-open).
- **UICfg.ini**
  - PortMgrDlg defines the max number of devices you want to operate at the same time
- **MfgTool.log**
  - Contains log messages from last mfg. tool session.

Main files & directories used with mfg. tool. – Profiles directory

- **Profiles/${chip}/OS Firmware/ucl2.xml –** (where chip=the profile name)
  - Location determined by settings in cfg.ini
  - tell MFGTool2 what kind of SoC it works on
    - different SoCs have different PID/VID, and MFGTool2 needs this information
  - Define operation lists
    - Contain commands to reprogram, burn fuses, etc.

UICfg.ini

```
[UICfg]
PortMgrDlg=1
```

- PortMgrDlg=1
  - Tells us that one device will be programmed for the next session

cfg.ini

The cfg.ini file is used to configure the target chip profile and target operation list sections. The format of this file looks like as the following:

[profiles]
chip = Linux

Indicates the target profile name (i.e. directory) which can be found under “<MFG>/Profiles”
MFGTool2 will try to find the ucl2.xml with the path (relative path to mfgtool2.exe) Profiles**${chip}**\OS Firmware. If you take the above chip value as an example, MFGTool2 will try to find the ucl2.xml at “Profiles**Linux**\OS Firmware”.

[platform]
board = SabreSD

Reserved

[LIST]
name = SDCard

Indicates the target operation list name which can be found in the file located at “<MFG>/Profiles/${chip}/OS Firmware/ucl2.xml". The name specified here points to a list entry in XML.
For example in “Profiles\Linux\OS Firmware” there should be this based on the above cfg.ini example contents:

```
<LIST name="SDCard" desc="Choose SD Card as media">
…
</LIST>
```

[variable]
board = sabreauto
mmc = 0
sxuboot=17x17arm2
sxdtb=17x17-arm2
ldo=

Variable values used in the ucl2.xml file in command lists...

```
<CMD state="BootStrap" type="boot" body="BootStrap" file ="firmware/u-boot-imx6q%board%_sd.imx"
ifdev="MX6Q">Loading U-boot</CMD>
```

ucl2.xml

A collection of all the tasks needed to do burning work. Consists of several parts:

- Global Configuration is contained between <CFG></CFG>.

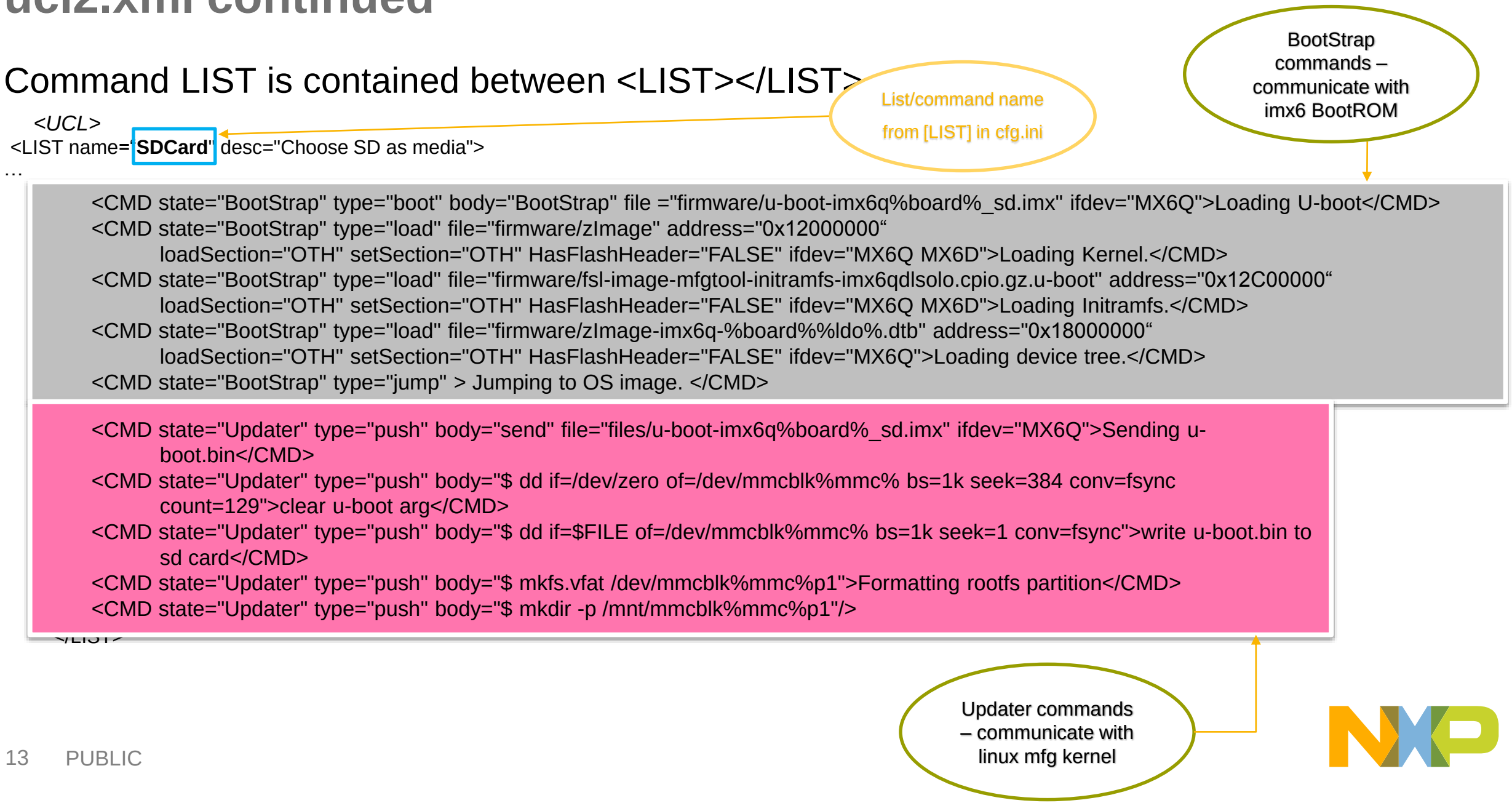
```
<UCL>
    <CFG>
      <STATE name="BootStrap" dev="MX6SL" vid="15A2" pid="0063"/>
      <STATE name="BootStrap" dev="MX6D" vid="15A2" pid="0061"/>
      <STATE name="BootStrap" dev="MX6Q" vid="15A2" pid="0054"/>
      <STATE name="BootStrap" dev="MX6SX" vid="15A2" pid="0071"/>
      <STATE name="Updater"   dev="MSC" vid="066F" pid="37FF"/>
    </CFG>
```

"<STATE name="BootStrap" dev="MX6Q" vid="15A2" pid="0054"/>" *- indicates the first phase of the burning process (communicating with the BootROM), the phase name is "BootStrap", and a device named "MX6Q" could be connected with the USB pid "0054" and vid "15A2", or MX6SL, MX6D or MX6SX. For i.MX 6 serial, in the phase "BootStrap", the valid strings for dev are: "MX6Q", "MX6D", "MX6SL" and "MX6SX"*

"<STATE name="Updater"   dev="MSC" vid="066F" pid="37FF"/>" *indicates the second phase of the burning process (communicating with the mfg. linux kernel),, the phase name is "Updater", and a device named "MSC" should be connected with the USB pid "37FF" and vid "066F".*

ucl2.xml continued

Command LIST is contained between <LIST></LIST>




```
<UCL>
<LIST name="SDCard" desc="Choose SD as media">
...
    <CMD state="BootStrap" type="boot" body="BootStrap" file ="firmware/u-boot-imx6q%board%_sd.imx" ifdev="MX6Q">Loading U-boot</CMD>
    <CMD state="BootStrap" type="load" file="firmware/zImage" address="0x12000000“
        loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" ifdev="MX6Q MX6D">Loading Kernel.</CMD>
    <CMD state="BootStrap" type="load" file="firmware/fsl-image-mfgtool-initramfs-imx6qdlsolo.cpio.gz.u-boot" address="0x12C00000“
        loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" ifdev="MX6Q MX6D">Loading Initramfs.</CMD>
    <CMD state="BootStrap" type="load" file="firmware/zImage-imx6q-%board%%ldo%.dtb" address="0x18000000“
        loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" ifdev="MX6Q">Loading device tree.</CMD>
    <CMD state="BootStrap" type="jump" > Jumping to OS image. </CMD>

    <CMD state="Updater" type="push" body="send" file="files/u-boot-imx6q%board%_sd.imx" ifdev="MX6Q">Sending u-
        boot.bin</CMD>
    <CMD state="Updater" type="push" body="$ dd if=/dev/zero of=/dev/mmcblk%mmc% bs=1k seek=384 conv=fsync
        count=129">clear u-boot arg</CMD>
    <CMD state="Updater" type="push" body="$ dd if=$FILE of=/dev/mmcblk%mmc% bs=1k seek=1 conv=fsync">write u-boot.bin to
        sd card</CMD>
    <CMD state="Updater" type="push" body="$ mkfs.vfat /dev/mmcblk%mmc%p1">Formatting rootfs partition</CMD>
    <CMD state="Updater" type="push" body="$ mkdir -p /mnt/mmcblk%mmc%p1"/>
</LIST>
```

ucl2.xml continued

BootStrap commands

<CMD state="BootStrap" type="boot" body="BootStrap" file ="firmware/u-boot-imx6q%board%_sd.imx" ifdev="MX6Q">Loading U-boot</CMD>

1. state = “BootStrap” – command communicates with the bootROM

2. type = “boot” - download u-boot-mx6q-sabresd.bin from host and store it into iRAM. ROM parse IVT/DCD in u-boot-mx6q-sabresd.bin and init the DDR. Copy the body of u-boot-mx6q-sabresd.bin to DDR.
3. body = “BootStrap” –
4. file ="firmware/u-boot-imx6q%board%_sd.imx“ – file used for this command – located in Profiles/Linux/OS Firmware/, with the %board% coming from cfg.ini, [variable] section.
5. “Loading U-boot” – comment that appears in the mfgtool GUI when this command is executed.

ucl2.xml continued

BootStrap commands continued

<CMD state="BootStrap" type="load" file="firmware/zImage" address="0x12000000“ loadSection="OTH" setSection="OTH" HasFlashHeader="FALSE" ifdev="MX6Q MX6D">Loading Kernel.</CMD>

1. state = “BootStrap” – command communicates with the bootROM

2. type = “load” – load image to memory
3. file ="firmware/zImage“ – file load use for this command – located in Profiles/Linux/OS Firmware/.
4. loadSection=“OTH” – a parameter used by ROM code, should be set to “OTH”.
5. HasFlashHeader="FALSE“ – set TRUE if the image contains a flash header, or set to FALSE.
6. ifdev="MX6Q MX6D“ – conditional describing which device this statement should be used with (see “dev=“ in <CFG></CFG> in ucl2.xml file – this parameter will be detected by the tool)
7. “Loading Kernel” – comment that appears in the mfgtool GUI when this command is executed.

ucl2.xml continued

BootStrap commands “jump” – start mfg. Linux

<CMD state="BootStrap" type="jump" > Jumping to OS image. </CMD>

1. state = “BootStrap” – command communicates with the bootROM
2. type = “jump” – Notify ROM code to jump to the RAM image to run.
3. “Jumping to OS image” – comment that appears in the mfgtool GUI when this command is executed.

ucl2.xml continued

Updater commands – now that we've jumped to the mfg uboot/kernel/ramfs, we're ready to tell it what to do…

<CMD state="Updater" type="push" body="send" file="files/u-boot-imx6q%board%_sd.imx" ifdev="MX6Q">Sending u-boot.bin</CMD>

1. state = "Updater" – command communicates with the mfg. kernel
2. type = "push" – the command is parsed and executed by the targeted device instead of host, the only thing host has to do is to send the command to the targeted device..
3. body="send" - Receive the file from the host. Subsequent shell commands can refer to the file received as $FILE.
4. file="files/u-boot-imx6q%board%_sd.imx" – file to send to the target (imx6), the %board% coming from cfg.ini, [variable] section.
5. ifdev="MX6Q MX6D" – conditional describing which device this statement should be used with (see "dev=" in <CFG></CFG> in ucl2.xml file – this parameter will be detected by the tool)
6. "Sending u-boot.bin" – comment that appears in the mfgtool GUI when this command is executed.

ucl2.xml continued

Updater commands – now that we've jumped to the mfg uboot/kernel/ramfs, we're ready to tell it what to do…

<CMD state="Updater" type="push" body="$ dd if=/dev/zero of=/dev/mmcblk%mmc% bs=1k seek=384 conv=fsync count=129">clear u-boot arg</CMD>

1. state = "Updater" – command communicates with the mfg. kernel
2. type = "push" – the command is parsed and executed by the targeted device instead of host, the only thing host has to do is to send the command to the targeted device..
3. body="$ dd if=/dev/zero of=/dev/mmcblk%mmc% bs=1k seek=384 conv=fsync count=129" – The "$" means command to run on target device. the %mmc% coming from cfg.ini, [variable] section.
4. "clear u-boot arg" – comment that appears in the mfgtool GUI when this command is executed.

ucl2.xml continued

Updater commands – now that we've jumped to the mfg uboot/kernel/ramfs, we're ready to tell it what to do…

<CMD state="Updater" type="push" body="frf">Finishing rootfs write</CMD>

1. state = "Updater" – command communicates with the mfg. kernel
2. type = "push" – the command is parsed and executed by the targeted device instead of host, the only thing host has to do is to send the command to the targeted device..
3. body = "frf" – Wait for all data transfer to be finished and processed (i.e. same as flush).
4. "Finishing rootfs write" – comment that appears in the mfgtool GUI when this command is executed.

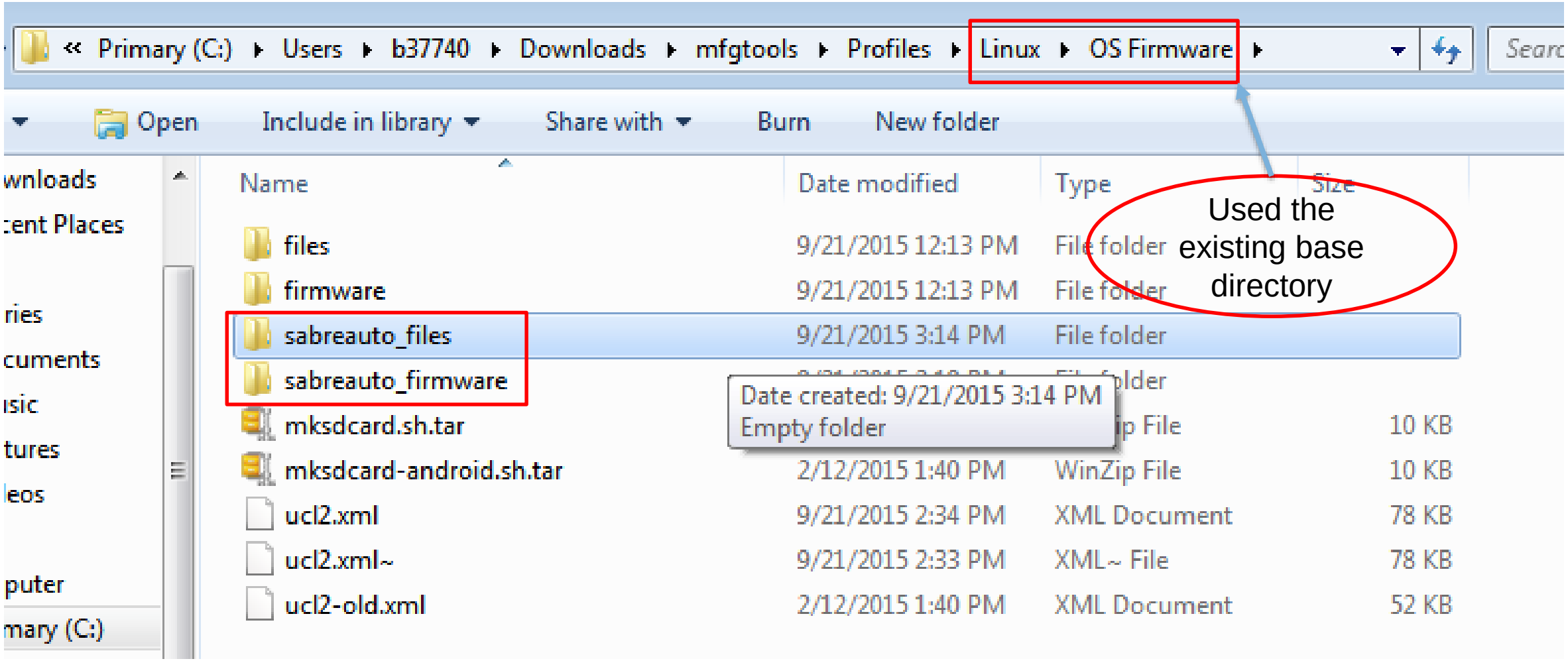
Logfile

- **MfgTool.log**
  - Log line examples:
    - DLL version: 2.3.4
    - Tuesday, September 15, 2015 17:51:14 Start new logging
    - ModuleID[2] LevelID[10]: DeviceManager::OnMsgDeviceEvent() - DEVICE_ARRIVAL_EVT(\\?\USB#VID_15A2&PID_0054#5&1604d86d&0&1#{a5dcbf10-6530-11d2-901f-00c04fb951ed})
      - VID=15a2, PID=0054 – it's imx6q!
    - ModuleID[2] LevelID[10]: ExecuteCommand--Boot[WndIndex:0], File is E:\mfg_tools\imx-3.14.28_1.0.0_ga-mfg-tools\mfgtools\Profiles\Linux\OS Firmware\firmware\u-boot-imx6qsabreauto_sd.imx
      - Loading the mfg. uboot file
    - ModuleID[2] LevelID[10]: *********MxHidDevice[00E0AC98] Jump to Ramkernel successfully!**********
      - Jumped to the mfg. builds of uboot,kernel,ramfs.
  - Can find error messages when board isn't programming.

Using the mfg. tool to program the generated files.

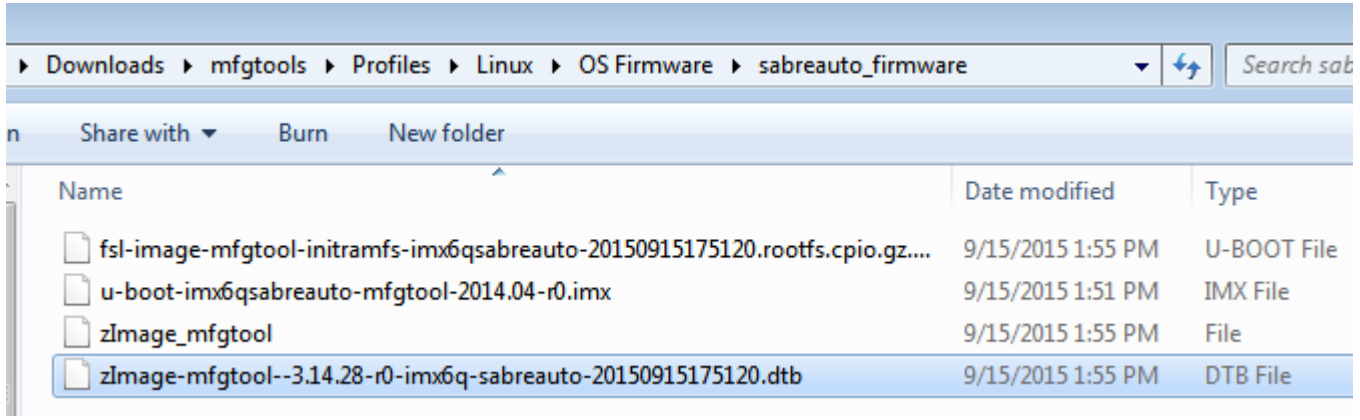
- Simple way
  - Keep using “chip = Linux” in [profiles] section in cfg.ini
  - To use most of the same subdirectories & files
  - This way we can use the existing ucl2.xml file and just add our command LIST.
- Create directories for the files we generated
  - Create “Profiles\Linux\OS Firmware\sabreauto_firmware” directory for mfg. linux images
    - mfg uboot, mfg kernel + dtb, mfg rfs
  - Create “Profiles\Linux\OS Firmware\sabreauto_files” directory for target linux images
    - uboot, kernel + dtb, rfs
- We could also have created our own profile .directory
  - mfgtools\Profiles\”mydirname”
  - We’d have to edit cfg.ini [profiles] section so the mfg tool could find our new dir and files
    - “chip=mydirname”
  - Also create an “mfgtools\Profiles\”mydirname”\OS Firmware”
  - Then create our own mfgtools\Profiles\”mydirname”\OS Firmware\ucl2.xml file (or copy/paste and modify the existing one)

Using the mfg. tool to program the generated files.



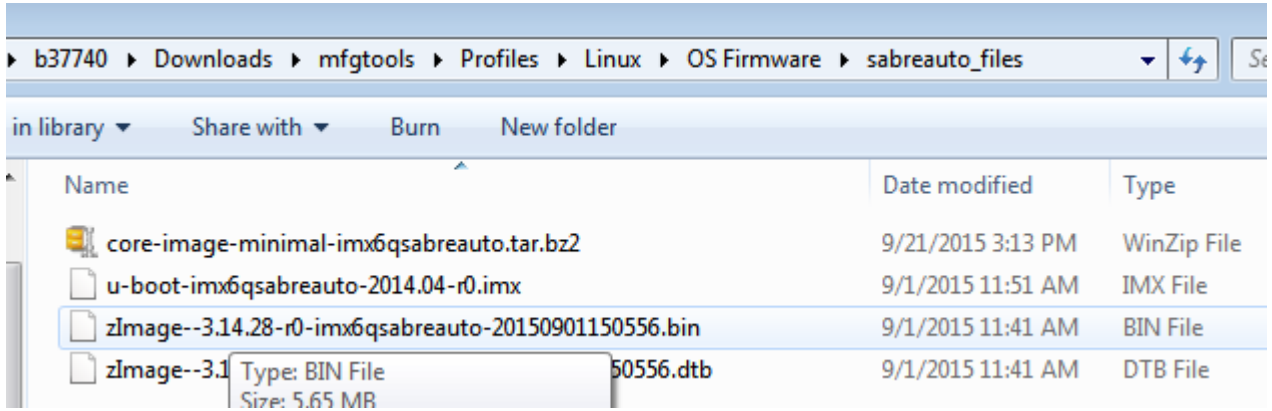
Using the mfg. tool to program the generated files.

- Copy mfg. tool linux images into the sabreauto_firmware directory.



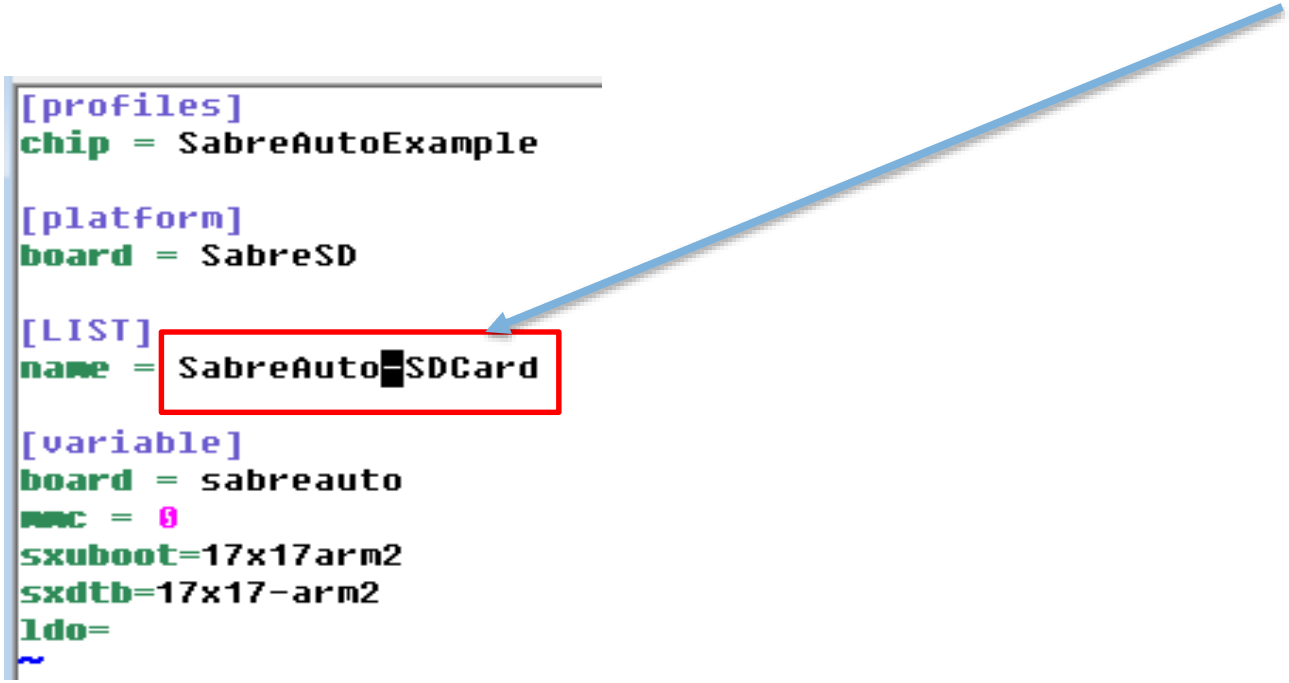
Using the mfg. tool to program the generated files.

- Copy final target images (to be “flashed”) into the sabreauto_files directory.



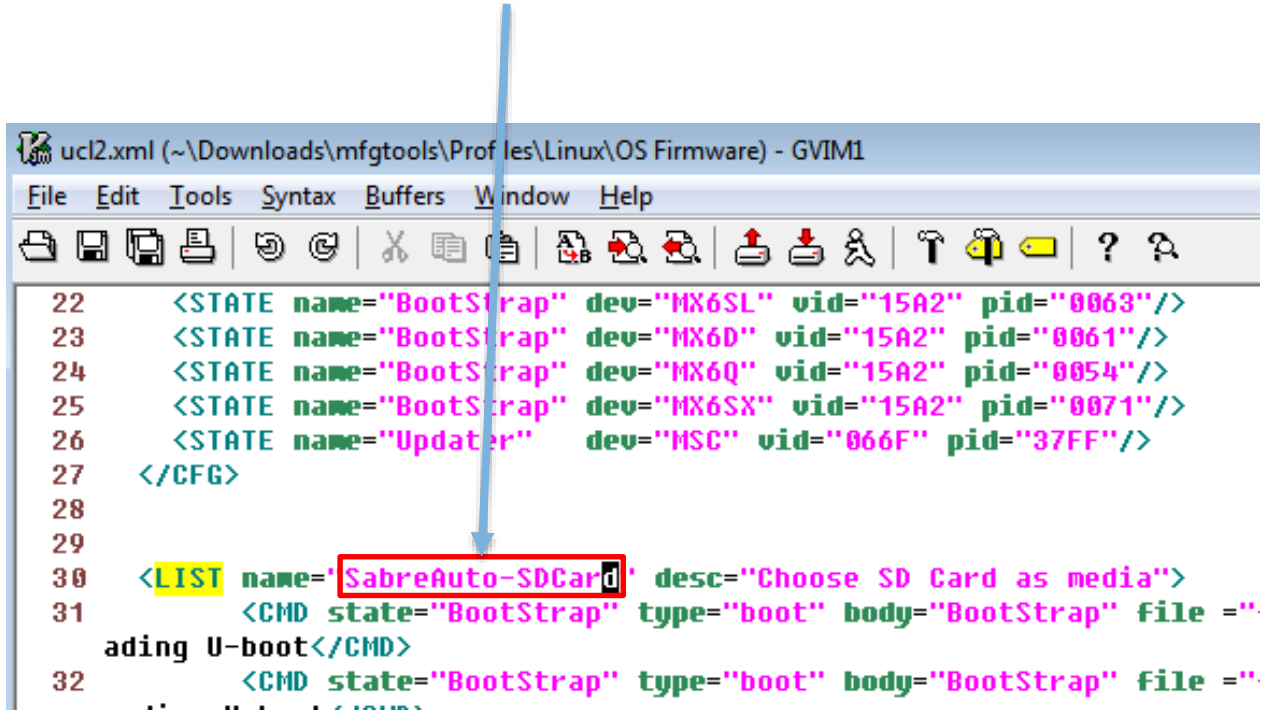
Using the mfg. tool to program the generated files.

- Since we're using the existing "chip" (directory) – Linux - and adding to the existing ucl2.xml file, we need to change the name of the LIST we want to use to program, because our command LIST will be in the same file as the old command list (SDCard), so let's name ours SabreAuto-SDCard in the cfg.ini file. We don't need to change any variables but we can if we want to use them in the ucl2.xml file..



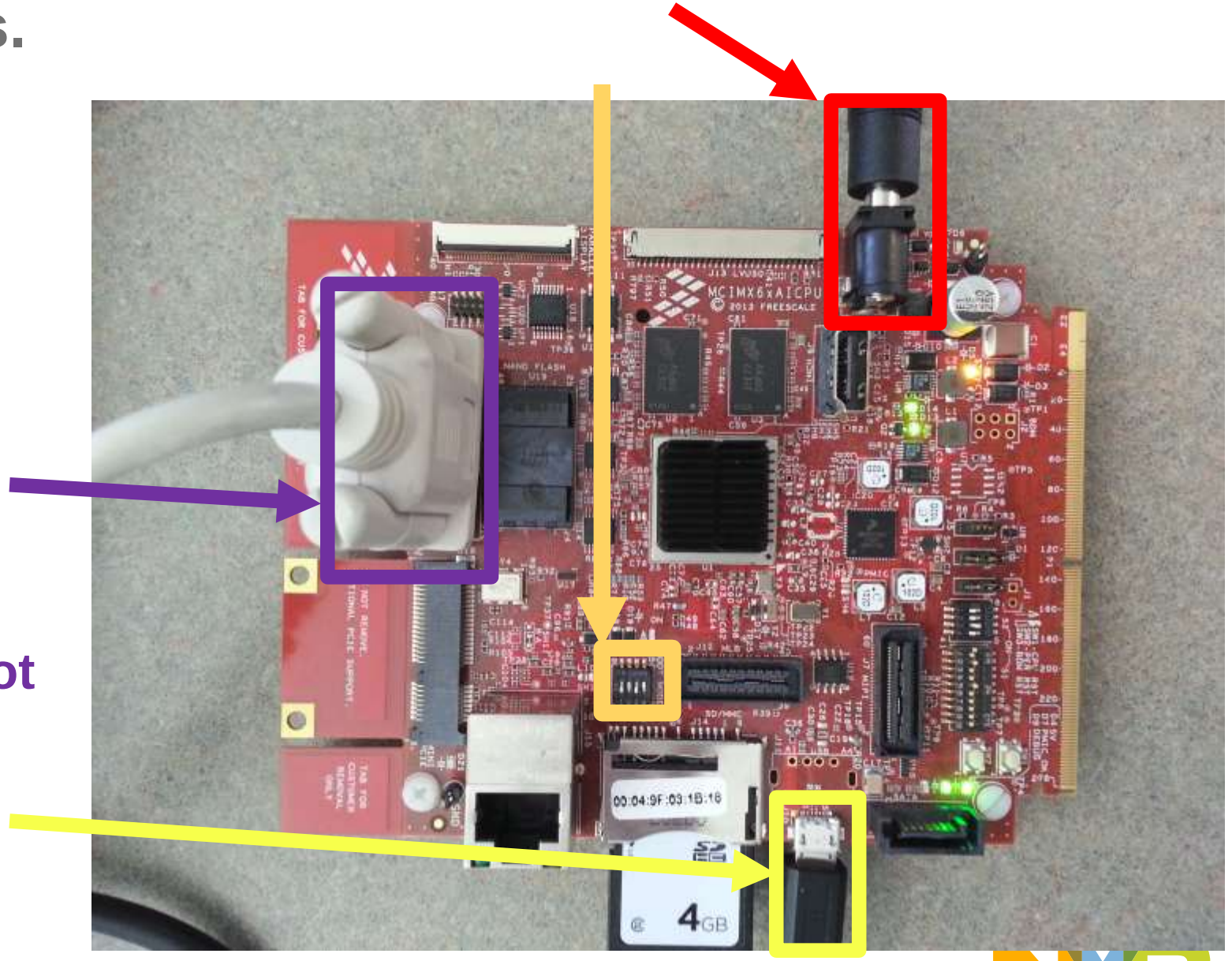
Using the mfg. tool to program the generated files.

- Copy and paste a command LIST in the ucl2.xml file to begin with.
- Change the LIST name in the ucl2.xml file to match what we used in the cfg.ini in the previous slide/step.



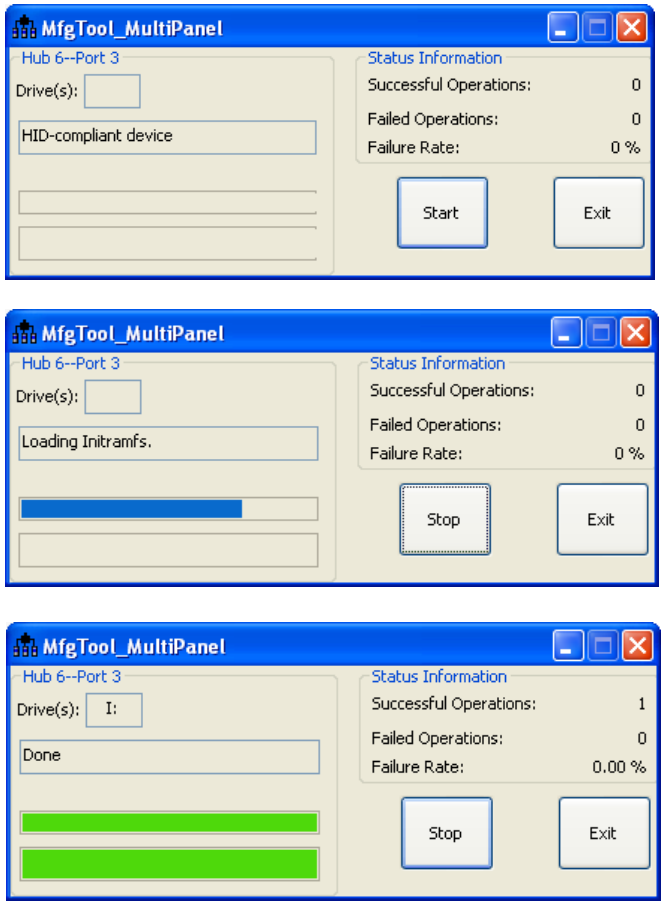
Set HW and connections.

- Set DIP switches on board
- Connect micro USB cable
- Connect other end of USB to PC
- **Connect serial cable (optional – will show linux booting, etc. mainly for debugging purpose if the board fails, although may not be very specific)**
- **Plug power to board**



Operation of Mfgtool

- Open “MfgTool.exe”. A device named “HID-compliant device” will be found if there is a valid device plugged in.
- Click “Start” button, it starts burning work.
  - Finished successfully.
- Press Stop button
  - Otherwise as soon as you connect a board or reboot one that is connected and in mfg. mode, the tool will try to reprogram it.
- Exit



EXTRA SLIDES

Errors I’ve seen

- ucl2.xml file not found
  - Error printed in Mfgtool.log?
- rootfs tar file was “empty”
  - No error in log file
  - No error on the console
  - Booted the board from the media (SD) created and noticed rfs was empty.

Console messages during reprogramming

- Console will display booting messages and UTP messages once the system is booted and running the mfg. linux to reprogram the board (Updater commands).
- Commands may not show sufficient detail to debug issues.

Building for mfg. tool

- **Build uboot, kernel, dtb and rfs specifically for the mfg. tool to communicate with.**
  - These images will allow the imx6 target hardware to run linux and communicate with the mfg. tool.
    - Initializes target HW according to any custom settings (SD port, NOR flash driver, etc.)
    - This allows linux commands to be used to program the board
  - Images for FSL EVBs can be downloaded from www.NXP.com/imx
  - Images can be built using Yocto (instructions on yocto preparation and building available from www.NXP.com/imx
  - These will be downloaded in steps 2,3 and 4 on the earlier Workflow slide
  - Then executed in steps 5,6 and 7 to perform the actual reprogramming actions

- Each board could use different linux images
- Can use variables in cfg.ini to combine boards into a single command LIST
- One operation list is dedicated to defining a specific storage on a specific board
  - Could use multiple storage media on a single board (SPINOR for uboot, SD for kernel, dtb & rfs).

ucl2.xml continued

From “*Manufacturing Tool V2 UCL User Guide.docx*”

1.1.2.1 OTP Bits Programming

<CMD state="Updater" type="push" body="$ ls /sys/fsl_otp ">Showing HW_OCOTP fuse bank</CMD>

<CMD state="Updater" type="push" body="$ echo 0x11223344 > /sys/fsl_otp/HW_OCOTP_MAC0">write 0x11223344 to HW_OCOTP_MAC0 fuse bank</CMD>

<CMD state="Updater" type="push" body="$ cat /sys/fsl_otp/HW_OCOTP_MAC0">Read value from HW_OCOTP_MAC0 fuse bank</CMD>

The fuse bank name (ex: HW_OCOTP_MAC0) should be set as needed.

*See “*Manufacturing Tool V2 UCL User Guide.docx*” for more details and examples.

Download Preparation

- Setup download environment as following:
  - A target device. Set boot option to “download” mode with DIP switches.
  - Prepare a micro-USB cable.
  - Prepare a PC with proper MFG tool installed (download from www.NXP.com/imx).
  - Prepare images to be downloaded to the target device.

SECURE CONNECTIONS
FOR A SMARTER WORLD