Yocto Project™: Partition Configuration for UEFI featuring the Kontron M2M Platform

By Sean D. Liming and John R. Malin
Annabooks

August 2013

Booting without BIOS: UEFI

The traditional BIOS that started with the original IBM PC back in 1981 is slowly being replaced with the open UEFI specification. Newer x86 boards have a combination of BIOS and UEFI, and some have UEFI without BIOS. The firmware change is happening as firmware needs to address larger memory space, improve security, and allow for some low-level hardware testing. The amount of UEFI support will vary depending on the board manufacturer. Microsoft's Windows 8 requires UEFI on OEM manufactured PCs to address security concerns around root kits, which is creating some concern in the Linux community that the open hardware platform is being closed. In retrospect, UEFI might be a little too much overhead to launch an embedded OS; nevertheless, many embedded board manufacturers are including UEFI support. Some boards have UEFI only and no BIOS. As of this writing, there is no straight forward way to support UEFI + Linux boot from ext2,3,4 file system, but there are some workarounds to support get journal file system support.

Build and Test Distribution

The Kontron M2M platform and Yocto Project 1.4.1 are going to be used for this demonstration. The Kontron M2M comes with either BIOS and UEFI support or UEFI-only support. The unit that was used for this paper has UEFI-only support.

1. Follow the Yocto Project quick start guide and download Yocto Project 1.4.1-Poky 9.0.1 from the yoctoproject.org websites.
2. Download the Intel® Atom™ E640T Processor with Intel EG20T Controller Hub Development Kit (Queens Bay) with IEMGD (Fish River Island 2) BSP.
3. Extract the BSP into the same directory as poky-dylan-9.0.1. Confirm that you have a folder called meta-fri2.

***Note**: Some of our past instructions have suggested putting the BSP in a /poky-dylan-9.0.1/meta-intel directory. The choice is up to you.*

4. Create a new project:

```
$Source poky-danny-9.0.1/oe-init-build-dev m2m
```

Or to be more generic:

```
$Source poky-<release>-<ver>/oe-init-build-dev <platform>
```

5. The next step was to modify the local.conf and bblayers.conf files to add the layers for the fri2 BSP, target the correct hardware platform: fri2-noemgd, and set up the number of processors available. Please see the Yocto Project Quick Start Guide, BSP README, and development documentation for more information.

***Note**: The fri2 BSP includes the Linux Kernel features to support UEFI. Not every Yocto Project BSP includes these features by default. If you use a different BSP, you will have to add 'efi' recipe as an image feature.*

6. Build a core-image-x11 image.

```
$ bitbake –k core-image-x11
```

7. The Kontron MSM's firmware is set to boot from a specific device order. USB will boot first and the internal microSD card will boot second. We can take advantage of this to test out the image and extract some of the boot files we will use later. Install the image to a blank USB flash disk.

```
$ sudo dd if=core-image-x11-fris-noemgd.hddimg of=/dev/sdd
```

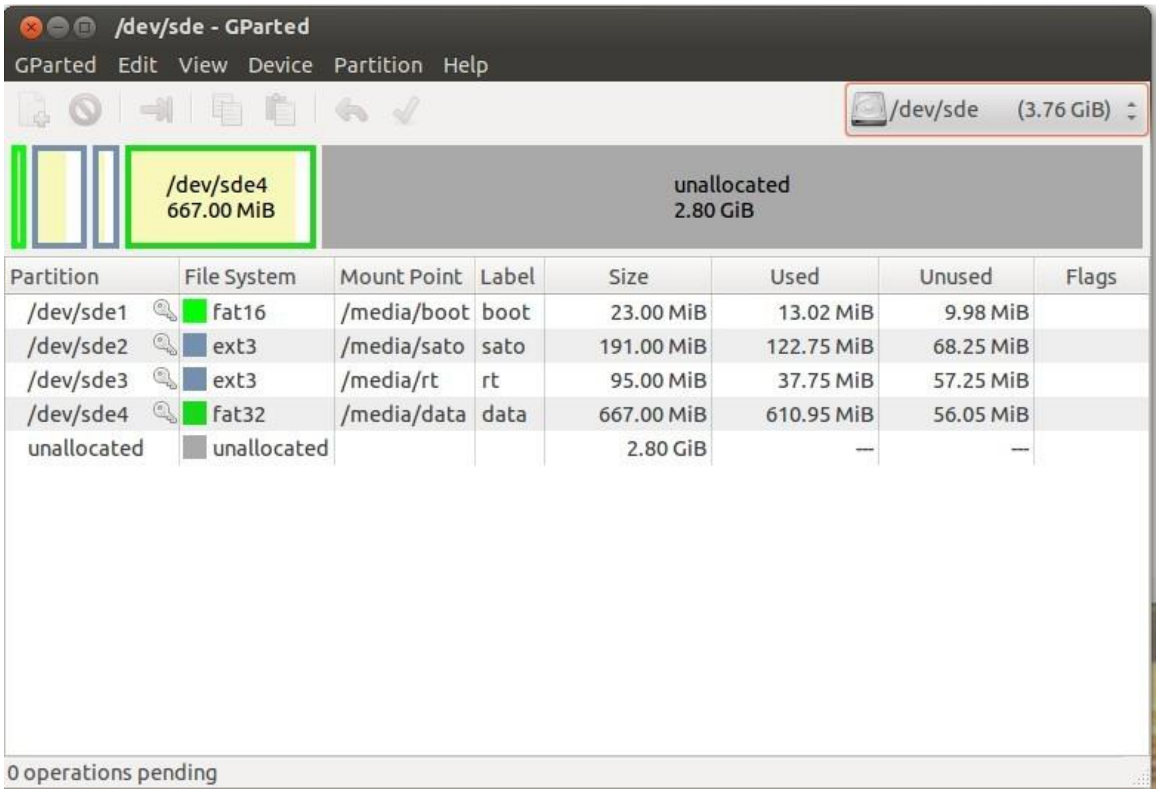
8. Modified /efi/boot/grub.cfg for the boot menu entry to add rw and loglevel

```
# Automatically created by OE
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
default=boot
timeout=10

menuentry 'boot'{
linux /vmlinuz LABEL=boot root=/dev/ram0 rw console=ttyPCH1,115200 console=tty0
video=vesafb vga=0x318 snd_hda_intel.enable_msi=0 loglevel=0
initrd /initrd
}

menuentry 'install'{
linux /vmlinuz LABEL=install-efi root=/dev/ram0   console=ttyPCH1,115200
console=tty0 video=vesafb vga=0x318 snd_hda_intel.enable_msi=0
initrd /initrd
}
```

9. Put the USB flash disk in the Kontron M2M platform and boot. The system should boot to a terminal screen.
10. Shutdown the system. The USB flash disk will be used later to copy the files over to the microSD card.

microSD Setup

The version of the Kontron M2M platform that was used came with an internal microSD drive that supported booting to 3 different Embedded Linux images built from the Yocto Project. Figure 1 shows how the partition was set up. The first partition has the EFI directory with boot files and grub configuration file. Also included are 3 vmlinuz kernel images. Two of these kernels are set up to use an associated ext3 partition. The tiny kernel uses a RAM file system.



Partition	File System	Mount Point	Label	Size	Used	Unused	Flags
/dev/sde1	🟩 fat16	/media/boot	boot	23.00 MiB	13.02 MiB	9.98 MiB	
/dev/sde2	🟦 ext3	/media/sato	sato	191.00 MiB	122.75 MiB	68.25 MiB	
/dev/sde3	🟦 ext3	/media/rt	rt	95.00 MiB	37.75 MiB	57.25 MiB	
/dev/sde4	🟩 fat32	/media/data	data	667.00 MiB	610.95 MiB	56.05 MiB	
unallocated	▨ unallocated			2.80 GiB	---	---	

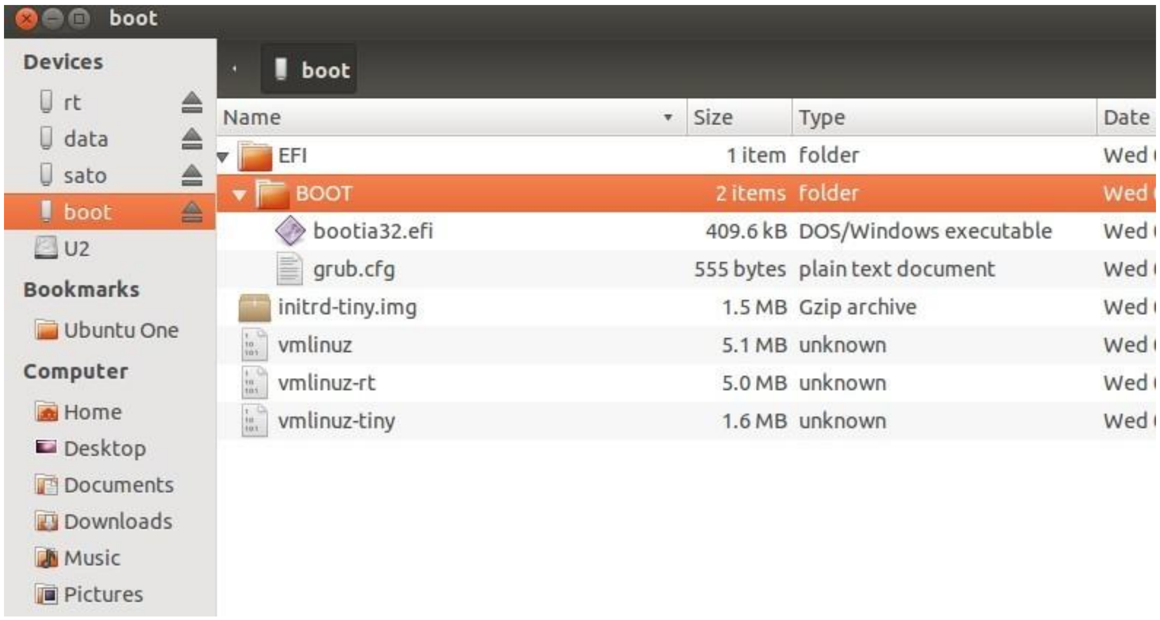
0 operations pending

Figure 1 - microSD Card Partitioning

The grub.cfg file shows the boot options for all three kernel implementations:

```
set menu_color_normal=light-gray/black
set menu_color_highlight=cyan/black

default=0
timeout=10

menuentry 'Yocto Project 1.3 - FRI2 - Poky Sato Desktop'{
linux /vmlinuz root=/dev/mmcblk0p2 rw rootwait console=ttyPCH1,115200
console=tty0
}

menuentry 'Yocto Project 1.3 - FRI2 - Poky Realtime'{
linux /vmlinuz-rt root=/dev/mmcblk0p3 rw rootwait console=ttyPCH1,115200
console=tty0
}

menuentry 'Yocto Project 1.3 - FRI2 - Poky Tiny (serial console only)'{
linux /vmlinuz-tiny root=/dev/ram0 rw console=ttyPCH1,115200 loglevel=0
initrd /initrd-tiny.img
}
```

The trick to set up UEFI support is to use a FAT partition to store the UEFI boot files and the kernel.



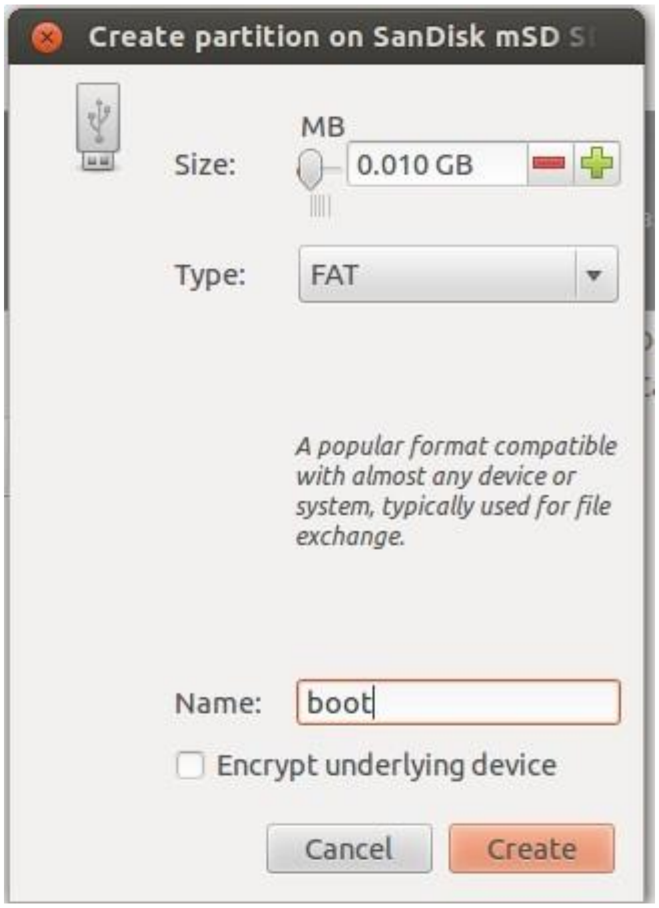
Name	Size	Type	Date
EFI	1 item	folder	Wed
BOOT	2 items	folder	Wed
bootia32.efi	409.6 kB	DOS/Windows executable	Wed
grub.cfg	555 bytes	plain text document	Wed
initrd-tiny.img	1.5 MB	Gzip archive	Wed
vmlinuz	5.1 MB	unknown	Wed
vmlinuz-rt	5.0 MB	unknown	Wed
vmlinuz-tiny	1.6 MB	unknown	Wed

Figure 2 - Boot Partition Contents

To test this solution, an 8 GB microSD card was set up using the following steps:

***Note**: Not all flash drives are created equal. Some are intended for commercial applications and are not robust enough for embedded systems. Some microSD cards will only support a single FAT partition. Attempting to partition these flash disks could result in drive failure.*

1. Insert the microSD card into the development system. In this case, Ubuntu 12.04 LTS was used.
2. Start the disk DiskUtility.
3. Umount the microSD partition.
4. Delete the FAT32 partition.
5. Select Device->Create Partition table.
6. Select Master Boot Record and click apply.
7. Create and name a primary partition FAT 0.010GB (10MB) + Name: boot



Create partition on SanDisk mSD S

Size: 0.010 GB MB

Type: FAT

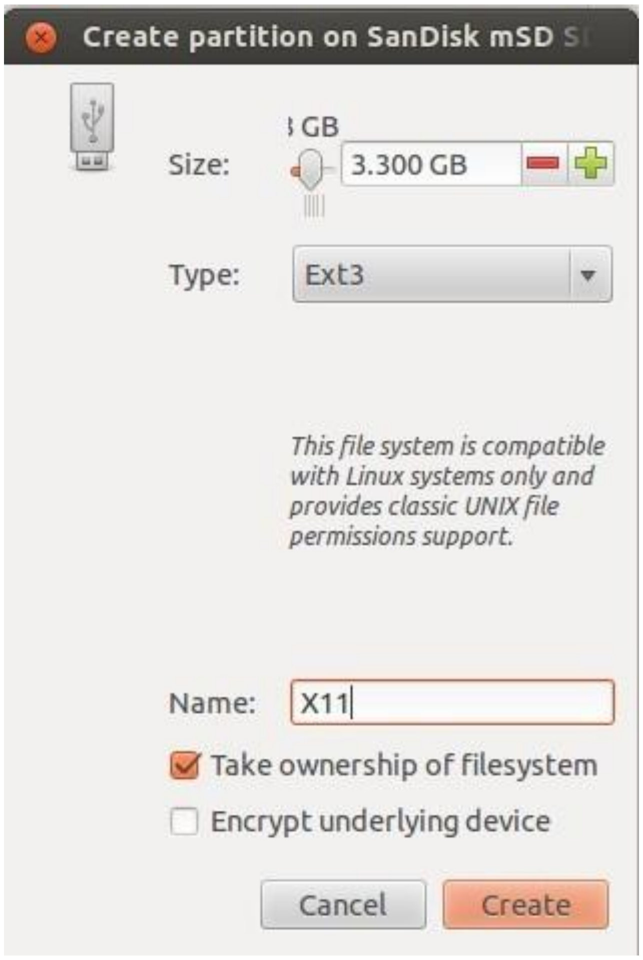
A popular format compatible with almost any device or system, typically used for file exchange.

Name: boot

☐ Encrypt underlying device

Figure 3 - FAT Boot Partition Setup

8. Create another partition ext32 – 3.3GB + Name: X11


Create partition on SanDisk mSD S

Size: 3.300 GB

Type: Ext3

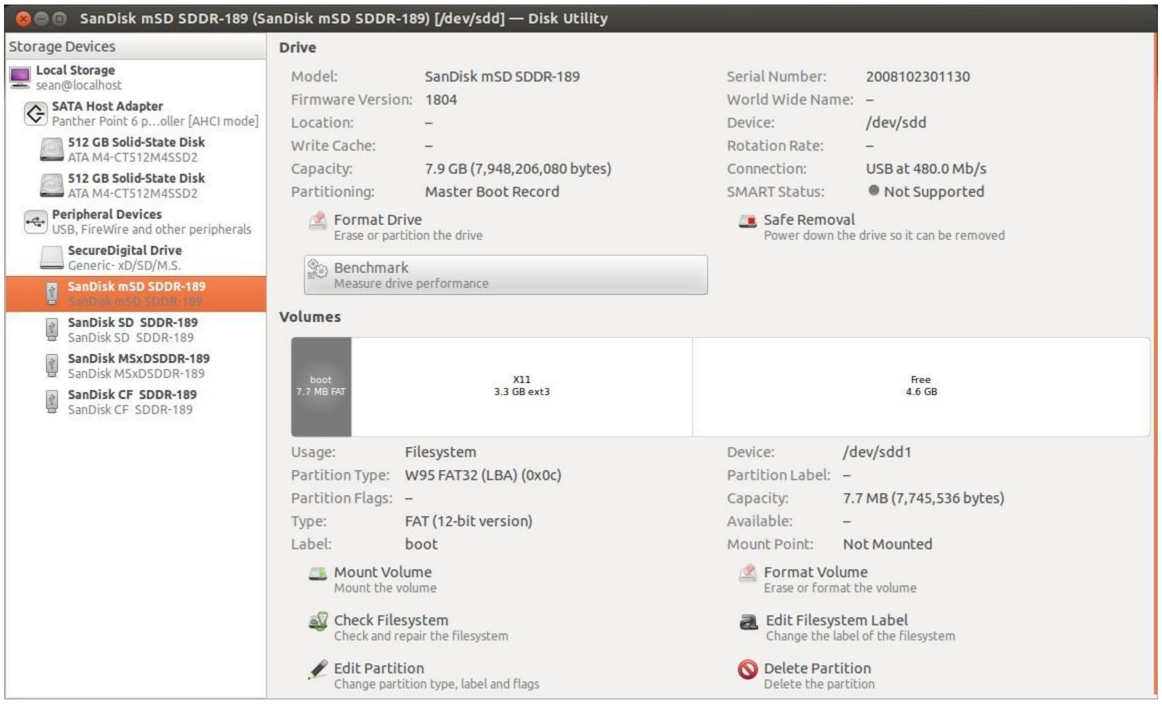
This file system is compatible with Linux systems only and provides classic UNIX file permissions support.

Name: X11

- [✓] Take ownership of filesystem
- [ ] Encrypt underlying device

Figure 4 - ext3 Partition Setup

There should be some empty space left over, which can be used for swap or other purposes.


SanDisk mSD SDDR-189 (SanDisk mSD SDDR-189) [/dev/sdd] — Disk Utility

Storage Devices
- Local Storage — sean@localhost
- SATA Host Adapter — Panther Point 6 p...oller [AHCI mode]
- 512 GB Solid-State Disk — ATA M4-CT512M4SSD2
- 512 GB Solid-State Disk — ATA M4-CT512M4SSD2
- Peripheral Devices — USB, FireWire and other peripherals
- SecureDigital Drive — Generic- xD/SD/M.S.
- SanDisk mSD SDDR-189 — SanDisk mSD SDDR-189
- SanDisk SD SDDR-189 — SanDisk SD SDDR-189
- SanDisk MSxDSDDR-189 — SanDisk MSxDSDDR-189
- SanDisk CF SDDR-189 — SanDisk CF SDDR-189

Drive

Model:	SanDisk mSD SDDR-189	Serial Number:	2008102301130
Firmware Version:	1804	World Wide Name:	–
Location:	–	Device:	/dev/sdd
Write Cache:	–	Rotation Rate:	–
Capacity:	7.9 GB (7,948,206,080 bytes)	Connection:	USB at 480.0 Mb/s
Partitioning:	Master Boot Record	SMART Status:	● Not Supported

Format Drive — Erase or partition the drive
Safe Removal — Power down the drive so it can be removed
Benchmark — Measure drive performance

Volumes

boot 7.7 MB FAT | X11 3.3 GB ext3 | Free 4.6 GB

Usage:	Filesystem	Device:	/dev/sdd1
Partition Type:	W95 FAT32 (LBA) (0x0c)	Partition Label:	–
Partition Flags:	–	Capacity:	7.7 MB (7,745,536 bytes)
Type:	FAT (12-bit version)	Available:	–
Label:	boot	Mount Point:	Not Mounted

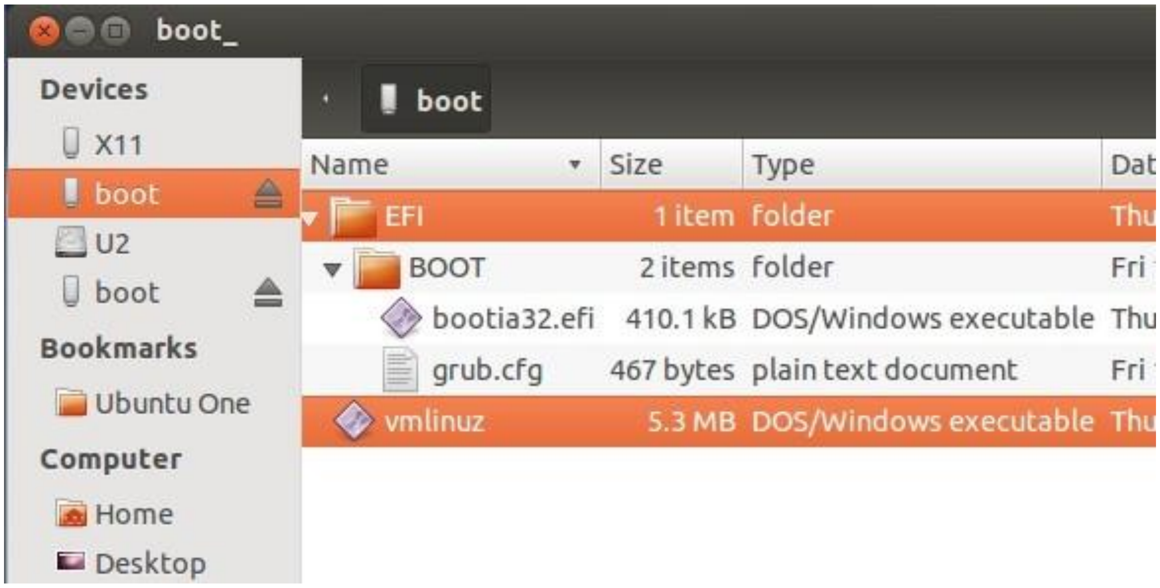
Mount Volume — Mount the volume
Format Volume — Erase or format the volume
Check Filesystem — Check and repair the filesystem
Edit Filesystem Label — Change the label of the filesystem
Edit Partition — Change partition type, label and flags
Delete Partition — Delete the partition

Figure 5 - Final Partition Setup

Install the Image on the microSD Card

With the disk partitioned, the next step is to install the image files.

1. Plug in the USB flash disk used to test the image.
2. Copy the EFI folder and vmlinuz to the microSD card's boot (FAT) partition. The RAM file system is not needed and would take up too much space.


boot_

Devices
- X11
- boot
- U2
- boot

Bookmarks
- Ubuntu One

Computer
- Home
- Desktop

boot

Name	Size	Type	Dat
EFI	1 item	folder	Thu
BOOT	2 items	folder	Fri
bootia32.efi	410.1 kB	DOS/Windows executable	Thu
grub.cfg	467 bytes	plain text document	Fri
vmlinuz	5.3 MB	DOS/Windows executable	Thu

Figure 6 - Copy only the EFI Folder and Kernel

3. Modify the grub.cfg file to boot the vmlinuz image and use the ext3 file system

```
set menu_color_normal=white/black
set menu_color_highlight=blue/black

# Automatically created by OE
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
default=boot
timeout=10

menuentry 'Yocto Project 1.4.1 M2M'{
linux /vmlinuz root=/dev/mmcblk0p2 rw rootwait console=ttyPCH1,115200
console=tty0 loglevel=0
}
```

4. Create a directory called /mnt/source
5. Mount the core-image-x11-fr2-noemgd.ext3 to the /mnt/source directory.

```
$ sudo mount -o loop core-image-x11-fr2-noemgd.ext3 ~/mnt/source
```

6. Copy the files from the /mnt/source directory to the microSD card's ext3 partition:

```
$ sudo -a ~/mnt/source/* /media/X11
```

7. Run sync to flush the disk cache.

```
$ sync
```

8. Unmount the /mnt/source directory.

```
$ sudo umount ~/mnt/source
```

9. Unmount the microSD flash disk.
10. Put the microSD card in the internal microSD slot and boot the image.

If you want to go one step further, you can load the bzimage instead of the vmlinuz. Modify the grub.cfg file as follows:

```
set menu_color_normal=white/black
set menu_color_highlight=blue/black

# Automatically created by OE
serial --unit=0 --speed=115200 --word=8 --parity=no --stop=1
timeout=10

menuentry 'Yocto Project 1.4.1 M2M'{
linux /vmlinuz root=/dev/mmcblk0p2 rw rootwait console=ttyPCH1,115200
console=tty0 loglevel=0
}

menuentry 'Yocto Project 1.4.1 M2M ext3'{
insmod part_msdos
insmod ext2
set root='(hd0,msdos2)'
linux /boot/bzImage root=/dev/mmcblk0p2 rw rootwait console=ttyPCH1,115200
console=tty0 loglevel=0
}
```

The second entry points to the kernel in the ext3 partition. If you prefer this solution, the vmlinuz kernel can be removed from the FAT partition.

Summary

Although there are solutions such as http://www.wensley.org.uk/gpt, the use of the FAT partition is a quick implementation to boot from UEFI. Time will tell if cleaner solutions will be implemented in the future.

Feedback welcome: http://www.annabooks.com/Contact.html