Gömülü(Embedded) Linux

Gömülü Sistemler

- Gömülü sistemleri özel amaçlar için tasarlanmış mikroişlemci(computing) yeteneği içeren sistemler olarak düşünebiliriz.
- PAL, CPLD, FPGA gibi programlanabilir donanımlar olabileceği gibi yazılımla programlanabilen mikrodenetleyici ve mikroişlemci de olabilir.

Mikrodenetleyici - Mikroişlemci

MCU(Micro Controller Unit):

- Hesaplama yetenekleri düşüktür
- MMU(Memory Management Unit) olmadığı için linux, android gibi işletim sistemlerini çalıştıramazlar.
- Dahili bellek(RAM) ve flash mevcuttur.
- FreeRTOS, MBed OS, Zephyr gibi işletim sistemleri kullanılabilir.
- STM32, ESP32 gibi örnekler verilebilir

MPU(Micro Processor Unit)

- Hesaplama yetenekleri yüksektir.
- MMU birimleri mevcuttur, harici bellek (DDR) kullanabilirler ve linux, android gibi işletim sistemlerini çalıştırabilirler.
- Özellikle ilklendirme(boot) aşamasında kullanılan dahili bellekleri mevcuttur.
- TI AM335x (Beaglebone),Broadcom BCM2711(Raspberry Pi 4) gibi örnekler verilebilir.

İşlemci Mimarileri

Yakın gelecekte RISC-V yaygın olacak görünse de şimdilik ARM en yaygın kullanılan mimaridir.

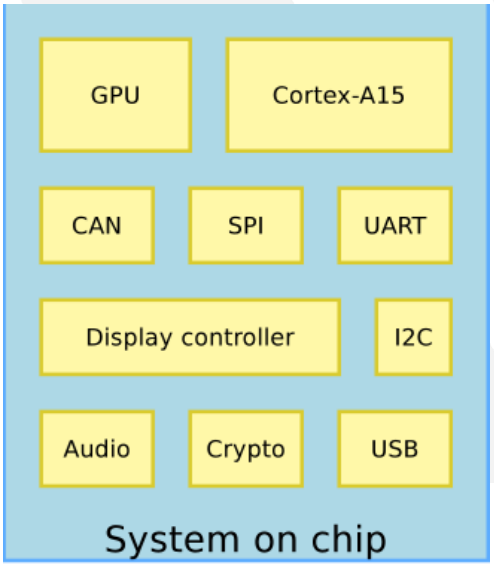
- x86 (CISC)
- ARM (RISC)
- RISC-V (RISC)
- MIPS(RISC)

ARM

E: Enhanced DSP instructions
J: Java byte code execution
M: Long multiply support
T: Thumb mode, 16 bit komut desteği

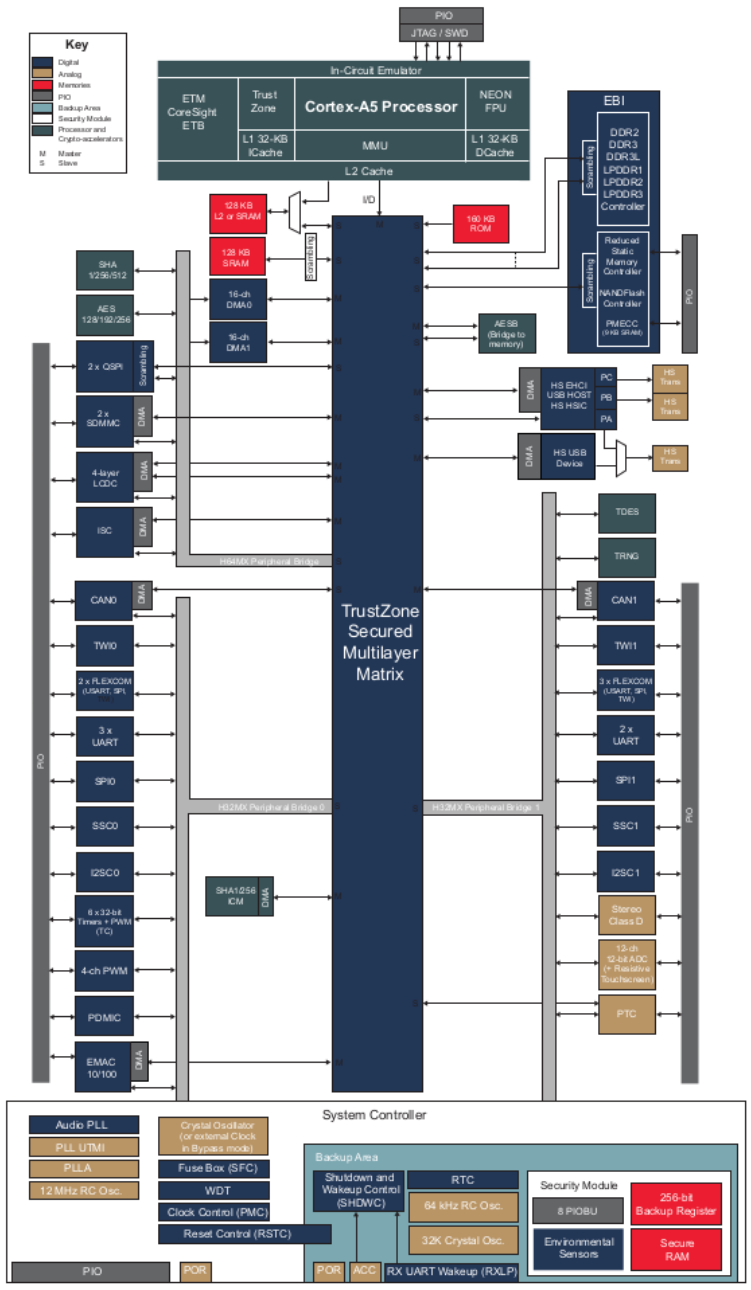
- …
- ARMv4T (ARM9x)
- ARMv5TEJ (ARM926EJ-S - iMX28)
- ARMv6 (ARM11)
- ARMv6-M (Cortex -M0, Cortex-M1)
- ARMv7-M (Cortex-M3)
- ARMv7E-M (Cortex-M4, Cortex-M7)
- ARMv7-A (Cortex-A5, Cortex-A7, Cortex-A5, Cortex-A8, Cortex-A9, Cortex-A15)
- ARMv8-A (Cortex-A34, Cortex-A35, Cortex-A57, Cortex-A72, Cortex-A73)
- …
- Her mimari kendi içinde kod uyumludur.

SoC



- ARM fiziksel bir ürün satmamaktadır.
- Çip üreticileri (Broadcom, Microchip, NXP, ….) ARM'dan istedikleri tasarımı satın alırlar
- SoC'de bulunan diğer birimleri(GPU, CAN, UART, USB, …) kendileri tasarlayabilecekleri gibi bunları da dışardan satın alabilirler.
- SoC olarak adlandırılan entegre devre üretirler.

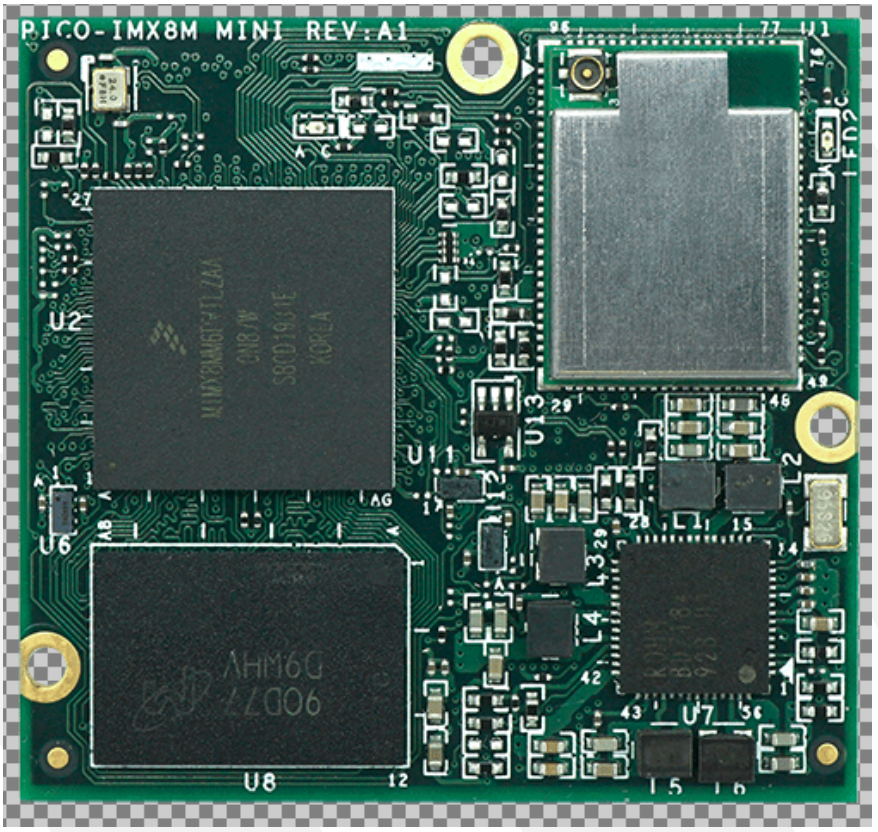
miltera

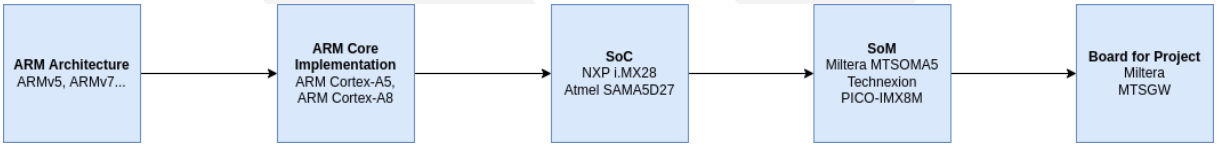


+90 (262) 320 00 20 info@miltera.com.tr www.miltera.com.tr
Bilişim Vadisi Muallim Mah. Deniz Cad.No: 143/5 A7 Blok, Kat: 1 Kapı No: 15 Gebze/Kocaeli

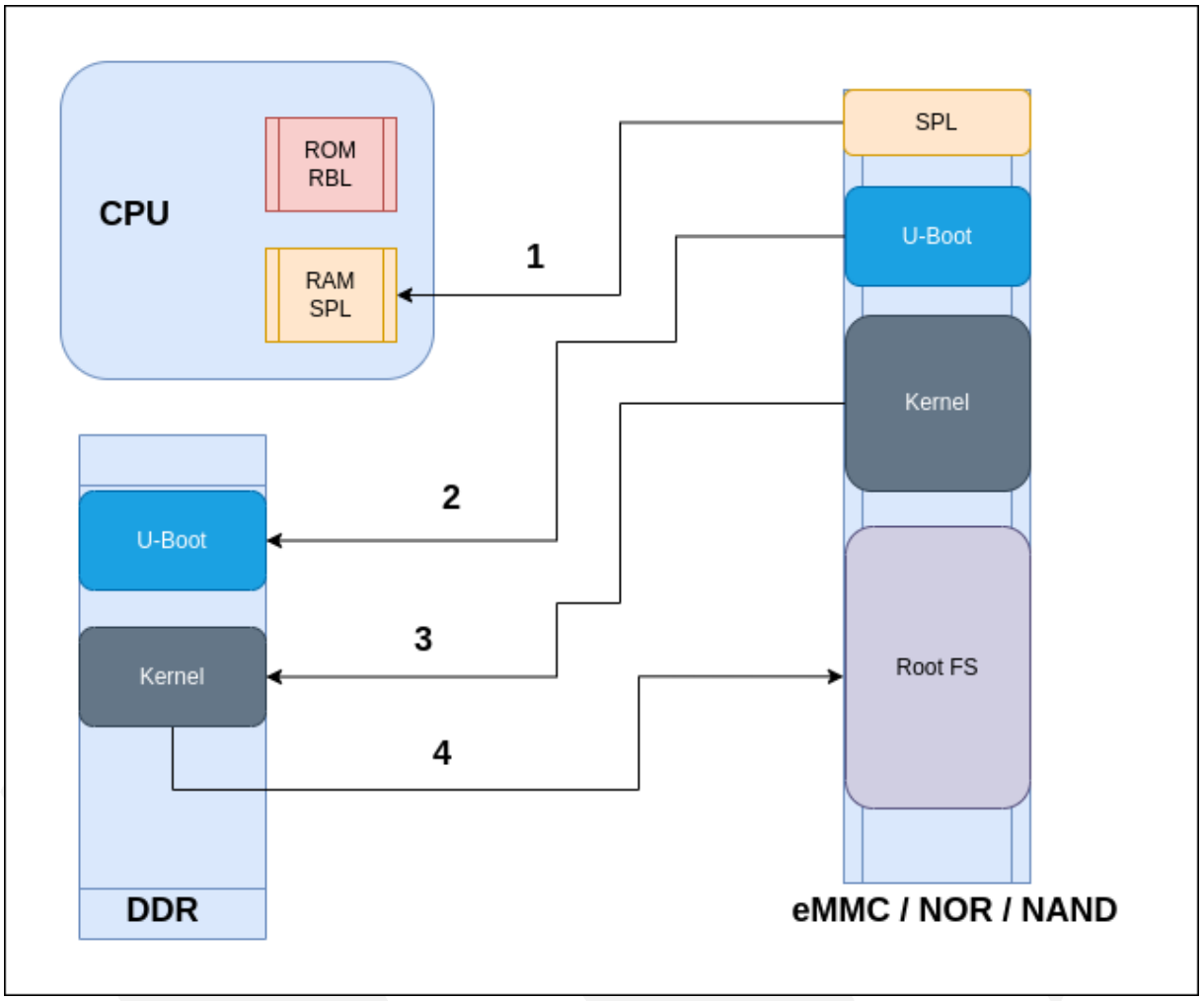
SoM

- Linux koşturan uygulama işlemcileri(MPU) DDR3/DDR4 gibi yüksek hızlı yollara ihtiyaç duymaktadırlar
- Kalıcı bellek olarak NAND, eMMC gibi donanımlar gerekmektedir.
- Ethernet arayüzü için phy kullanılmaktadır.
- Herbir projede bu tasarımları tekrar tekrar yapmak yerine modüler tasarımlar faydalı olacaktır.





Gömülü Linux için Açılış Süreci



- Her çip üreticisi ilgili SoC için özel ROM Code geliştirmektedir.
- İşlemci ilk açılışta (reset aldığında) ROM içinde bulunan reset vektöre dallanır.
- SPL kodu SoC haricinde bulunduğu için hangi ortamdan(eMMC, SDCard, NOR Flash, NAND Flash) yükleneceği ilgili pinlerle konfigüre edilmelidir. ROM Boot Loader bu pinleri kontrol ederek ilgili arayüzü ve ilgili kod bileşenlerini aktif ederek SPL kodunu dahili hafızaya yüklemektedir.
- ROM Bootloader yüklediği (dahili hafızadaki) kullanıcı koduna dallanacaktır (SPL).
- SoC içinde bulunan dahili hafıza yeteri kadar büyük olmadığı için bu kodun olabildiğince küçük olması gerekmektedir.
- Secondary Program Loader(SPL) SoC için özel olacağı için çip üreticileri tarafından sağlanmaktadır.
- Kimi çip üreticileri SPL kodu U-Boot'a entegre etmektedir kimileri ise ayrı bir uygulama olarak geliştirmektedir.

- SPL kodunda kullanılacak çevre birimler (güç, pll, …) ve harici bellek (DDR) ilklendirilmelidir.
- Harici bellek ilklendirilidikten sonra kalıcı hafızadaki kullanıcı kodu (U-Boot) harici belleğe çekilir ve bu koda dallanılır.
- Harici bellekten çalışmaya başlayan U-Boot proje gereksinimlerine uygun olarak ilgili çevre birimlerini ilklendirir (ekran, ethernet ….).
- Kalıcı hafızadaki kernel'i (linux imajı: uImage, zImage, Image) harici belleğe yükler ve sistem parametrelerini(ATAGs, devicetree) kernele aktararak kernel'e dallanır.
- Harici bellekten çalışmaya başlayan kernel proje gereksinimlerine uygun olarak ilgili sürücü ve bileşenleri aktif eder ve kullanıcı dosya sistemine bağlanır (mount).
- Kullanıcı dosya sistemine bağlanıldığında **INIT** (busybox init, sys v, systemd …) süreci başlatılır.

SPL & U-Boot

- Çoğu SoC üreticisi SPL kodu U-Boot koduna entegre etmektedir.
- Dizin yapısı linux ile benzerlik gösterir.
- İşlemci fiziksel adres uzayında çalıştırılır.
- Proje gereksinimlerine uygun olarak ilgili çevre birimleri etkinleştirir.
- Kalıcı hafızadan kerneli belleğe yükler.
- Sistem parametreleri (ATAGs) kernele aktarılır
- Kernele dallanır.

Çalışma Silsilesi

- **arch/arm/lib/vectors.S**
  exception vector tabloları ayarlanır
- **arch/arm/cpu/armv7/start.S**
  Kesmeler(IRQ, FIQ) iptal edilir.
  Cache ve MMU iptal edilir
- **arch/arm/cpu/armv7/lowlevel_init.S**
  Alt seviye ilklendirmeler
- **arch/arm/lib/crt0.S**
  _main(): stack ve GlobalData alanı ilklendirme
  board_init_f(): arch_cpu_init, board_early_init_f, timer_init, env_init, init_baud_rate, serial_init, …. (**common/board_f.c**)
  relocate_code(): U-Boot, board_init_f() tarafından belirtilen adrese kopyalanır (**arch/arm/lib/relocate.S**)
  board_init_r(): initr_env, audio_add_devices, console_init_r, interrupt_init, initr_net (**common/board_r.c**)
  main_loop(): autoboot ya da cli (**common/main.c**)

U-Boot main_loop() fonksiyonunu çalıştırdığına kullanıcıdan **bootdelay** saniye içinde herhangi bir tuşa basmasını beklemektedir. Bu süre zarfında herhangi bir tuşa basılmazsa **bootcmd** komutu çalıştırılır.

Dizin Yapısı

api	
arch	Mimari bağımlı dosyalar
board	Bord bağımlı dosyalar
cmd	Komut dosyaları
common	Mimari bağımsız fonksiyonlar
configs	Default konfigürasyon dosyaları
disk	
doc	
drivers	Aygıt sürücüleri
dts	
env	
examples	
fs	Dosya sistem kütüphaneleri
include	Başlık dosyaları
lib	Temel kütüphaneler
Licenses	
net	Network kütüphaneleri
post	
scripts	
spl	SLP çıktı dosyaları
test	
tools	Araç gereç dosyaları

Bord Ekleme

Aşağıdaki çalışma pico-imx6 som modül için yapılmıştır.

Arch Dizini

Mikroişlemci ailesi ile ilgili dosyaları barındırır. CPU ilklendirme, pinmux kontrol yazılımları, DRAM(controller) ile ilgili yazılımlar, clock yazılımları örnek olarak verilebilir. Genellikle SoC üreticisinin sağlamış olduğu hazır yapı yeterlidir.
Yeni bir bord ekleneceği zaman bordu tanımlayan temel bilgilerin **arch/arm/mach-imx/mx6/Kconfig** dosyasına eklenmesi yeterlidir.

Board Dizini

Borda özel dosyaları barındırır. Pinmux konfigürasyonları, bord ilklendirme(early, late) fonksiyonları gibi yazılımları barındıran borda özel kaynak kodlar ve bu kaynak kodun derlenebilmesi için gerekli Kconfig ve Makefile dosyalarını örnek olarak söyleyebiliriz.

Configs Dizini

Bütün boadların default konfigurasyon dosyalarını barındırır. Make defconfig aşamasında kullanılmaktadır.

Include/configs Dizini

Bütün bordların başlık dosyalarını barındırır. Bord tanımlamaları, auto boot ve önemli komut tanımlamaları burada yapılır.

- **Bord KConfig Dosyasının Oluşturulması**

board/my_vendor/ my_board/Kconfig

```
if TARGET_MY_BOARD

config SYS_BOARD
        default "my_board"

config SYS_VENDOR
        default "my_vendor"

config SYS_CONFIG_NAME
        default "myboard"

endif
```

SYS_VENDOR ve **SYS_BOARD** ifadeleri board dosyalarının konumunu göstermektedir.
board/my_vendor/ my_board
SYS_CONFIG_NAME ifadesi ise board için kullanılacak başlık dosyasının adını göstermektedir.
include/configs/myboard.h

- **Bord Kaynak Kodunun Oluşturulması**

board/my_vendor/my_board/my_board.c

```
#include <...>

DECLARE_GLOBAL_DATA_PTR;

…
int dram_init(void)
{
        gd->ram_size = imx_ddr_size();
        return 0;
}

/*
init-peripherals, uart, eth, eMMC, …
*/
```

```
…
int board_init(void)
{
        return 0;
}
```

Board dosyasında, common dizini altında yer alan __weak ifadesi ile tanımlanmış fonksiyonlar board için yeniden tanımlanır. __weak ifadesi link aşamasında kullanılmaktadır; tanımlanmış gerçek bir fonksiyon yoksa bu ifade ile tanımlanmış fonksiyonlar link edilir. U-Boot çalışmaya başladığında **common/board_f.c** dosyasında yer alan **static init_fnc_t init_sequence_f[]** dizisinde tanımlı fonksiyonlar çalıştırılır. Burada DRAM, clocks, seri port gibi birimlerin ilklendirilmeleri gerçekleştirilir.
Sonrasında ise **common/board_r.c** dosyasında yer alan **static init_fnc_t init_sequence_f[]** dizisinde tanımlı fonksiyonlar çalıştırılır. Burada ise NAND, NOR flash cihazları, MMC, MAC, ETH , PCI gibi çevre birimler ilklendirilir.
Yine board dosyasında pin konfigürasyonları ayarlanmaktadır. Mikroişlemcilerde, pin sayısının yetersizliğinden, her bir pin birden fazla fonksiyonu sağlayabilmektedir. Bord dosyasında tasarımda kullanılan pinler istenilen fonksiyonları sağlayacak şekilde konfigüre edilmelidir.

- **Bord Makefile Dosyasının Oluşturulması**

board/my_vendor/my_board/Makefile

```
obj-y := my_board.o
```

- **Bord Default Konfigürasyon Dosyasının Oluşturulması**

configs/myboard_defconfig

```
CONFIG_ARM=y
…
CONFIG_ARCH_MX6=y
…
CONFIG_TARGET_MY_BOARD=y
…
CONFIG_MXC_UART=y
…
CONFIG_BOOTDELAY=2
```

- **Bord Başlık Dosyasının Eklenmesi**

include/configs/myboard.h

```
#ifndef __MY_BOARD_CONFIG_H__

#define __MY_BOARD_CONFIG_H__


#include <....>
```

```
#define .....


#define CONFIG_EXTRAENV_SETTINGS \
        "image=zImage\0" \
        "console=/dev/ttyX\0" \
        ...
#endif
```

- **Board TARGET Kconfig Seçeneğinin Tanımlanması ve Kaynak Gösterilmesi**

arch/arm/mach-imx/mx6/Kconfig

```
choice
...
config TARGET_MY_BOARD
bool "My custom board"
select CPU_XXX
select SUPPORT_XXX
select BOARD_EARLY_INIT_F
...
endchoice


…
source "board/my_vendor/my_board/Kconfig"
….. .
```

Uygulama

- Toolchain kurulumu

Yocto ile oluşturulan toolchain **/opt/toolchains/fs-imx-wayland** dizinine kurulur.

./fsl-imx-wayland-glibc-x86_64-meta-toolchain-cortexa9t2hf-neon-pico-imx6-toolchain-5.10-hardknott.sh

- U-Boot'un build edilmesi

Örnek SOM için U-Boot kodu indirilir.

git clone git@gitlab.com:milteraopenlab/u-boot.git

Yukarıda anlatıldığı gibi bordumuz u-boot'a eklenir.

PATH çevre değişkeni ayarlanır

PATH=$PATH:/opt/toolchains/fs-imx-wayland/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi

Derleme sürecinde gerekli değişkenler ayarlanır

export CROSS_COMPILE=/opt/toolchains/fs-imx-wayland/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-

export ARCH=arm

U-Boot konfigüre edilir ve derlenir.

make mt-imx6_spl_defconfig
make

- U-Boot Imajının SD Karta Yazılması

sudo dd if=u-boot-with-spl.imx of=/dev/sdX bs=512 seek=2

IMX ailesine ait ROM boot kodu, sd karttan boot ederken 0x400 (512 x 2) adresinden başlayacak şekilde ham okuma yapar. Bu amaçla SPL link edilmiş U-Boot kodumuzu SD kartın 0x400 adresine ham olarak yazdık.
TI ve Atmel ailelerine ait ROM boot kodu FAT dosya sistemini desteklemektedir ve hem SPL kodunu(MLO, BOOT.BIN) hem de u-boot imajını sd kartta oluşturacağımız fat dosya sistemine yazabiliriz.

U-Boot Komutları

Çevre birimler(i2c, mmc, gpio, usb, phy, …), dosya sistemleri(ext4, ubi, fat, …), donanım bilgileri, bellek işlemleri gibi bir çok gereksinim için komut bulunmaktadır.
main_loop aşamasında herhangi bir tuşa basarak komut arayüzüne geçildiğinde **help** komutu ile var olan komutlar görülebilir.
print(env): Çevre değişkenlerini listeler
setenv: Çevre değişkeni değerini günceller veya yeni çevre değişkeni ekler
saveenv: Çevre değişkenlerini kaydeder.
reset: Cihazı yeniden başlatır.
bootm: Bellekte istenilen noktaya dallanır (uImage ile kullanılır).
bootz: Bellekteki kernele dallanır. (zImage ile kullanılır)
ext2(4)load: Ext2/ext4 dosya sisteminden belleğe dosya çeker.
fatload: FAT dosya sisteminden belleğe dosya çeker.

U-Boot Çevre Değişkenleri

U-Boot bir takım çevre değişkenlerine sahiptir. Bu çevre değişkenleri ile baudrate, kullanıcı bekleme süresi, belleğe yüklenecek dosya adı gibi değişkenler ayarlanabilir, proje özelinde davranışlar tanımlanabilir. U-Boot çevre değişkenlerini kalıcı hafızaya kaydeder. İstenildiğinde **env default -a** komutu ile bütün değişkenler binary imajdaki hallerine alınabilir. Kullanıcı gereksinimlerine göre yeni değişken tanımlayabilir (**setenv my_env my_val)**.

En önemli iki değişken aşağıdaki gibidir;

bootcmd: Kullanıcı her hangi bir tuşa basmadığında otomatik çalıştırılacak komutları tanımlar. Temel olarak kernel, device tree ve varsa initrd imajlarını belleğe alır ve dallanır.
Belleğe yüklenecek bu dosyalar farklı hafıza birimlerinde olabilir (NFS, NAND, NOR, eMMC) dolayısıyla ilgili hafıza birimine ve kullanılan dosya sistemine özel komutları içerir. Kullanılan/tasarlanan güncelleme mekanizmasına bağlı olarak güncel kernel'in belirlenmesi ve belleğe alınması gibi operasyonlar bu komut ile yürütülebilir.

bootargs: Kernele aktarılacak komut satırı parametrelerini tanımlar. Konsol bilgisi, dosya sistemi için bağlanacak (mount edilecek) aygıt bilgisi, bellek bilgisi, ekran bilgileri kernele aktarılabilir.

bootcmd ve **bootargs** değişkenleri **include/configs/my_board.h** dosyasında proje için özelleştirilmektedir.

Otomatik Açılış Sürecinin İncelenmesi

- U-boot main_loop aşamasında belli bir süre(bootdelay) kullanıcının u-boot komut satırına giriş yapmasını bekleyecektir.
- Giriş yapılmadığında otomatik olarak, **kernel,** kullanılıyor ise **initrd(initial ramdisk)** ve **device-tree** imajlarının belleğe alınması, kernel parametrelerinin ayarlanması ve kernel adresine dallanma sürecini başlatacaktır.
- Otomatik açılış süreci **bootcmd** komutu ile, kernel parametreleri ise **bootargs** parametresi ile sağlanmaktadır.

My_Board.h(include/configs/) Başlık Dosyası

- **bootcmd** komutu ve **bootargs** parametresi **include/configs/my_board.h** başlık dosyasında tanımlanmaktadır.

CONFIG_BOOTCOMMAND tanımlaması.

```
#define CONFIG_BOOTCOMMAND \
        "mmc dev ${mmcdev}; if mmc rescan; then " \
                "if run loadbootenv; then " \
                        "echo Loaded environment from ${bootenv};" \
                        "run importbootenv;" \
                "fi;" \
                "if test -n $uenvcmd; then " \
                        "echo Running uenvcmd ...;" \
                        "run uenvcmd;" \
                "fi;" \
                "if run loadbootscript; then " \
                        "run bootscript; " \
                "fi; " \
                "if run loadfit; then " \
                        "run fitboot; " \
                "fi; " \
                "if run loadimage; then " \
                        "run mmcboot; " \
                "else " \
                        "echo WARN: Cannot load kernel from boot media; " \
                "fi; " \
        "else run netboot; fi"
```

- Görüleceği üzere **bootcmd** komutumuz eMMC (SD Kart) üzerinden açılacak şekilde ayarlanmıştır. Öncelikle sd kartta uEnv.txt dosyası aranmaktadır. Sonrasında çevre değişkeni olarak ayarlanmış **uenvcmd** komut kümesi kontrol edilmektedir. Daha sonra sd kartta **bootsrc** dosyası aranmaktadır. Son

olarak **loadimage & run mmcboot** komutları ile kernel ve device tree belleğe alınmış ve kernel başlatılmıştır.

- Gerek başlık dosyasıyla gerekse uEnv.txt ve boot.scr dosyaları yardımıyla hangi şartlarda hangi aygıttan boot edileceği kontrol altına alınabilir.
- include/configs/my_board.h dosyası basit tutularak uEnv.txt ve/veya boot.csr dosyaları ile otomatik boot kontrol edilebilir.

uEnv.txt

- **key=value** şeklinde okunabilir dosyadır.
- Aşağıdaki gibi örnek bir dosya oluşturulur ve sd kartın boot bölümüne kopyalanır.

Uygulama

```
mmcargs=setenv bootargs console=${console},${baudrate} root=${mmcroot} ${displayinfo}
bootcmd_mmc=run loadimage;run mmcboot;
uenvcmd=run bootcmd_mmc;
```

U-Boot komut satırında **fatload mmc 0 0x12000000 uEnv.txt** komutu ile değişken dosyası belleğe alınır. **env import -t -r 0x12000000 $filesize** komutu ise dosyadaki değişkenleri sisteme ekler. **filesize** dosya sisteminden okuma işlemi yapıldığında sistem tarafından otomatik doldurulur. **printenv** komutu ile kontrol sağlanabilir.

boot.scr

- Koşul ifadeleri, döngü ifadeleri kullanarak otomatik açılışı kontrol etmemizi sağlar.

Uygulama

boot.script adında uygulama dosyası oluşturulur.

```
setenv my_value      "c"
setenv VALUES        "a b c d"

for value in "${VALUES}"; do
   if test "${value}" != "${my_value}"; then
        echo "Cannot found my_valusse: ${my_value}"
   else
        echo "Found my_value: ${my_value}, we can do anything like boting from custom
device"
   fi
done
```

Oluşturulan dosya **mkimage -c none -A arm -T script -d boot.script boot.scr** komutu ile u-boot imajı haline getirilir. Çıktı dosyası **boot.scr** sd kartın boot bölümüne kopyalanır.
U-Boot komut satırında **fatload mmc 0 0x12000000 boot.scr** komutu ile uygulama imajı belleğe alınır. **source** komutu ile uygulama çalıştırılır.

Uygulama

SD kart boot bölümünde bulunan kernel belleğe alınır;
fatload mmc 0 0x12000000 zImage

SD kart boot bölümünde bulunan device-tree belleğe alınır;
fatload mmc 0 0x18000000 imx6q-pico-nymph-qca.dtb

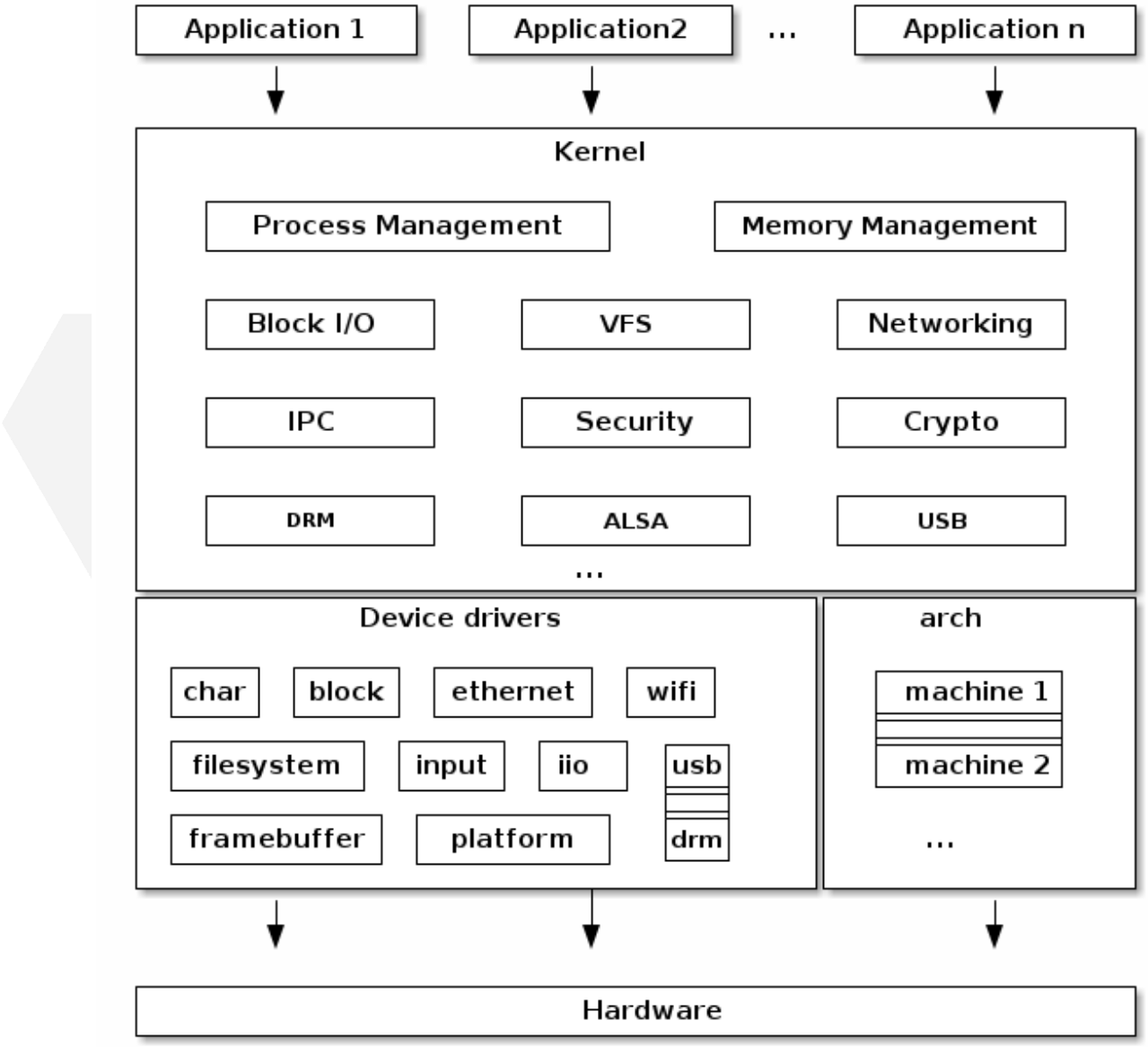
bootargs parametresi ayarlanır;
setenv bootargs console=ttymxc0,115200 root=/dev/mmcblk0p2 rootwait rw

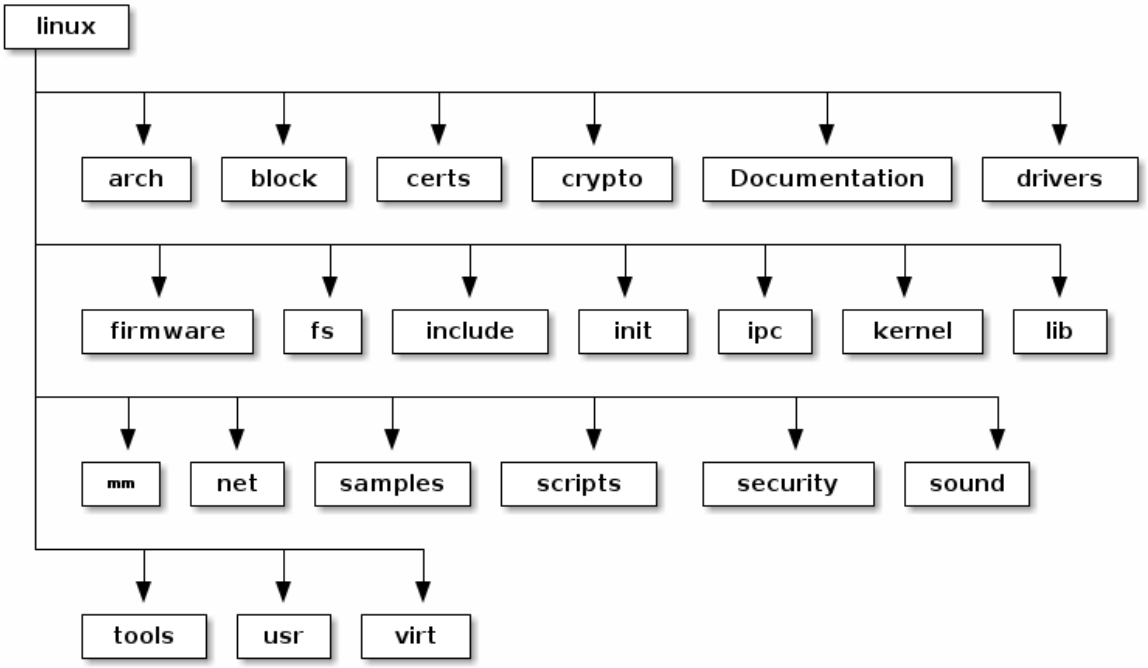
Kernele dallanılır;
bootz 0x12000000 - 0x18000000

Linux

Kaynak Kodlar

- Kernel kodları aygıt sürücüleri ve temel kernel kodları şeklinde iki gruba ayrılır.
- Aygıt sürücüleri sistemde kullanılan donanımların yönetimini sağlarken temel kernel kodları; dosya sistemi, proses yönetimi,network yönetimi gibi bileşenlerden oluşur.





- **arch**: Mimari bağımlı kodlar (Arm, mips, x86...).
- **block**: Blok türü aygıtlara okuma/yazma alt yapısını barındırır (block I/O request).
- **certs**: Sertifika kullanımı ve imza denetimi
- **crypto**: Kriptografi algoritmaları
- **documentation**: İlgili dokümanlar
- **drivers**: Aygıt sürücüleri
- **firmware**: Bazı donanımlar tarafından kullanılan binary dosyalar
- **fs**: Sanal dosya sistemi ve diğer dosya sistemleri
- **include**: Başlık dosyaları
- **init**: Mimari bağımsız ilklendirme
- **ipc**: Proses haberleşme mekanizmaları (message queue, semaphores, shared memory..)
- **kernel**: Proses yönetimi(kernel thread, workqueue), scheduler, ...
- **lib**: Genel kütüphaneler
- **mm**: Bellek yönetimi
- **net**: Network stack'leri, yönlendirme, filtreleme, ...
- **samples**: Örnek sürücüler ve kodlar
- **security**: SELinux yapısı
- **sound**: ALSA
- **tools**: Kullanıcı araç/gereçleri
- **usr**: Kernele entegre initrd desteği
- **virt**: KVM(Kernel Virtual Machine) hypervisor

Kernel Konfigürasyonu

- Kernel, birçok işlemci mimarisini (x86, PowerPC, MIPS, ARM, …) desteklemektedir.
- Kernel yüzlerce aygıt sürücüsü, onlarca dosya sistemi kütüphanesi, ağ protokolleri gibi bir çok bileşene sahiptir.
- Kernel derlenmeden önce kullanılan işlemci türü ve proje gereksinimlerine uygun olarak konfigüre edilmelidir.
- Kernel konfigürasyonu kök dizinde bulunan **.config** dosyasında tutulmaktadır (key=value).
- **KConfig** tanımlama yapısı sayesinde **make menuconfig**(text tabanlı) veya **make xconfig**(grafik tabanlı) komutları ile konfigüre edilebilir.
- Kernel konfigürasyonu ve derleme sistemi **Makefile** temellidir ve mimari konfigürasyonu **.config** dosyası ile birlikte **ARCH** çevre değişkenini de gerektirmektedir. **export ARCH=arm**
- Default konfigürasyonlar **arch/<ARCH>/configs/** dizini altında bulunmaktadır. Proje kapsamında oluşturulan **.config** dosyası bu dizine özel bir isimle kayıt edilebilir. **make my_board_defconfig**
- Default durumda **ARCH=x86** şeklindedir.
- Proje özelinde sade bir konfigürasyon oluşturmak için özellikle **Networking Support, Device Drivers** ve **File Systems** menüleri incelenebilir.

Uygulama

git clone git@gitlab.com:milteraopenlab/linux.git
export ARCH=arm
make make tn_imx_defconfig
make menuconfig

Kernel Derleme

- Çapraz derleyici yolu **PATH** değişkenine atanır.
- **ARCH** ve **CROSS_COMPILE** değişkenleri tanımlanır. Bu değişkenler make komutunda parametre olarak verilebileceği gibi **export** ile çevre değişkeni olarak da tanımlanır.
- **make** komutu ile derleme gerçekleştirilir.

Uygulama

PATH=$PATH:/opt/toolchains/fs-imx-wayland/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi
export ARCH=arm

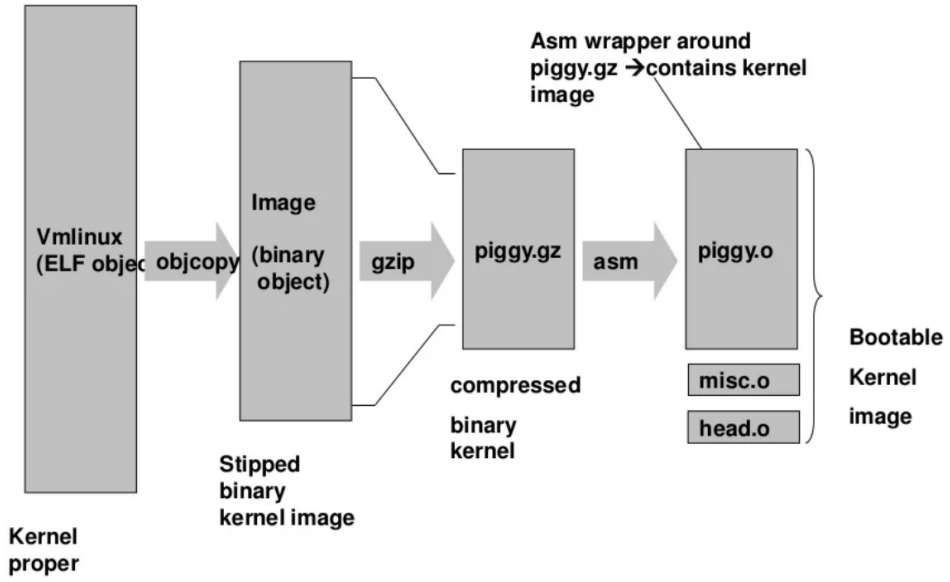
```
export CROSS_COMPILE=arm-poky-linux-gnueabi-
make tn_imx_defconfig
make
```

Çıktı Dosyaları

- **Image**: kernel imajı (arch/arm/boot)
- **zImage**: Sıkıştırılmış imaj (arch/arm/boot)
- **uImage**: mkimage ile zImage iamıjının başına 64 byte u-boot bilgisi eklenmiş imaj (arch/arm/boot)
- ***.dtb**: device tree dosyaları (arch/arm/boot/dts)

Imaj Oluşum Süreci

```
...
LD      vmlinux
SORTTAB vmlinux
SYSMAP  System.map
OBJCOPY arch/arm/boot/Image
Kernel: arch/arm/boot/Image is ready
AS      arch/arm/boot/compressed/head.o
LZO     arch/arm/boot/compressed/piggy_data
AS      arch/arm/boot/compressed/piggy.o
CC      arch/arm/boot/compressed/misc.o
LD      arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
```



- Öncelikle kök dizininde ELF(Executable File Format) formatında vmlinux dosyası oluşmaktadır. vmlinux dosyası sembol tabloları, debug bilgileri içermektedir ve boyutu 20MB üzerinde olabilir.
- **arm-poky-linux-gnueabi -O binary -R .note -R .comment -S vmlinux arch/arm/boot/Image** komutu ile ELF formatından ham binary dönüşümü yapılır. Kernel boyu 2-10MB seviyesindedir.
- **arch/arm/boot/Image** dosyası sıkıştırılarak **arch/arm/boot/compressed/piggy_data** elde edilir.
- **arch/arm/boot/compressed/piggy.S** dosyası ile **piggy_data** dosyası **piggy.o** obje dosyasına çevrilir.
- **arch/arm/boot/compressed/head.S** ve **arch/arm/boot/compressed/misc.c** dosyaları build edilir ve **piggy.o** obje dosyası ile link edilerek **arch/arm/boot/compressed/vmlinux** elde edilir.
- **arch/arm/boot/compressed/vmlinux** dosyası ELF formatındadır ve sembol tabloları, debug bilgileri içermektedir. Bu bilgiler silinerek **arch/arm/boot/zImage** ham binary dosyası oluşturulur.

  head.o: Kernel bootstrap kodudur.Her mimari kendi head.S dosyasını barındırır.
  misc.o: Sıkıştırılmış imajı açar. "**Uncompressing Linux … Done**"

Kernel Parametreleri

- Gömülü sistemlerde bellek miktarı, bağlanılacak(mount) dosya sistemi aygıt bilgileri, ramdisk bilgileri, ekran bilgileri gibi bazı bilgilerin kernele aktarılması gerekmektedir.

ARM Tag List ile Parametre Aktarımı

- Kernele parametre aktarımı linux-3.8 sürümüne kadar ARM Tag List (ATAG) ve commandline (kernel komut satırı) ile yapılmaktaydı.
- Bootloader ilgili bilgileri uygun yapıda toparlar ve bellekte bir adrese yazar (Genellikle +100).
- ATAG kullanımında ARM register kümesinden **R1** makina kodunu, **R2** ise bellekteki ATAG adresini tutar.
- Kernel komut satırı da aynı şekilde tag list ile aktarılmaktadır.

Device Tree ile Parametre Aktarımı

- Kısaca donanım tanımlama dosyası(imajı) diyebiliriz.
- Bootlader tarafından belleğe alınan device-tree imajının bellekteki başlangıç adresi ARM register kümesinden **R2** kullanılarak kernele aktarılır.

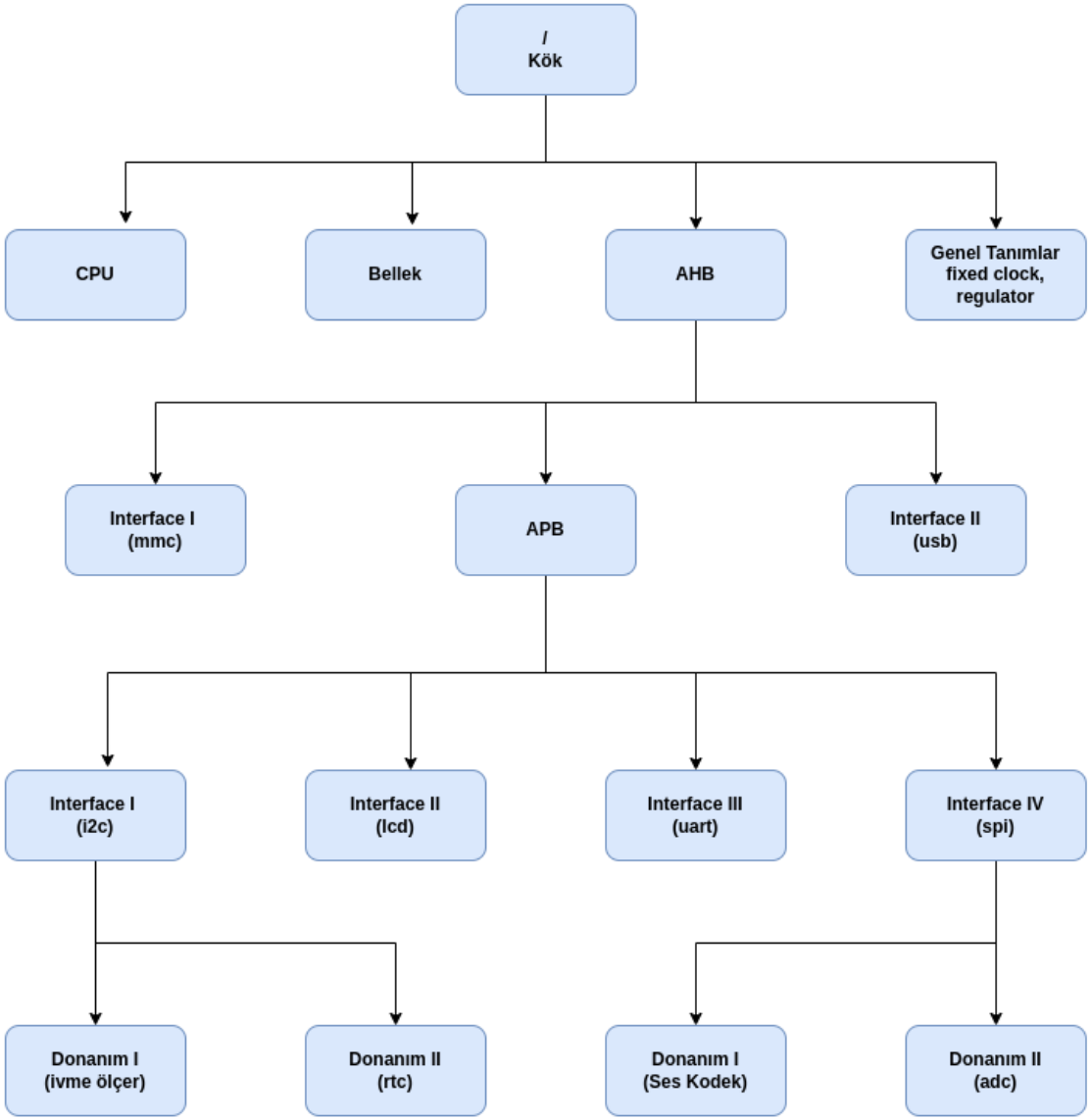
- Kernel komut satırının nasıl belirleneceği ile ilgili 3 farklı seçenek bulunmaktadır ve **Menuconfig -> Boot Options -> Kernel Command Line Type** ile konfigüre edilebilmektedir.
    - **Bootloader:** Yalnızca bootlader; ***include/configs/myboard.h veya uEnv.txt***
    - **Kernel + Bootloader:** Kernel tarafından tanımlanan komut satırına bootloader tarafından aktarılan eklenecektir.
    - **Kernel:** Yalnızca kernelde tanımlanan kullanılacaktır; ***menuconfig -> Boot Options -> Default Kernel Command String***
- Seçenek I veya Seçenek II aktif olduğu durumda bootlader belleğe aldığı device tree imajına **/chosen** düğümü altında **bootargs** parametresini ekleyecektir. **common/fdt_support.c -> fdt_chosen()**.

Donanım Tanımlama

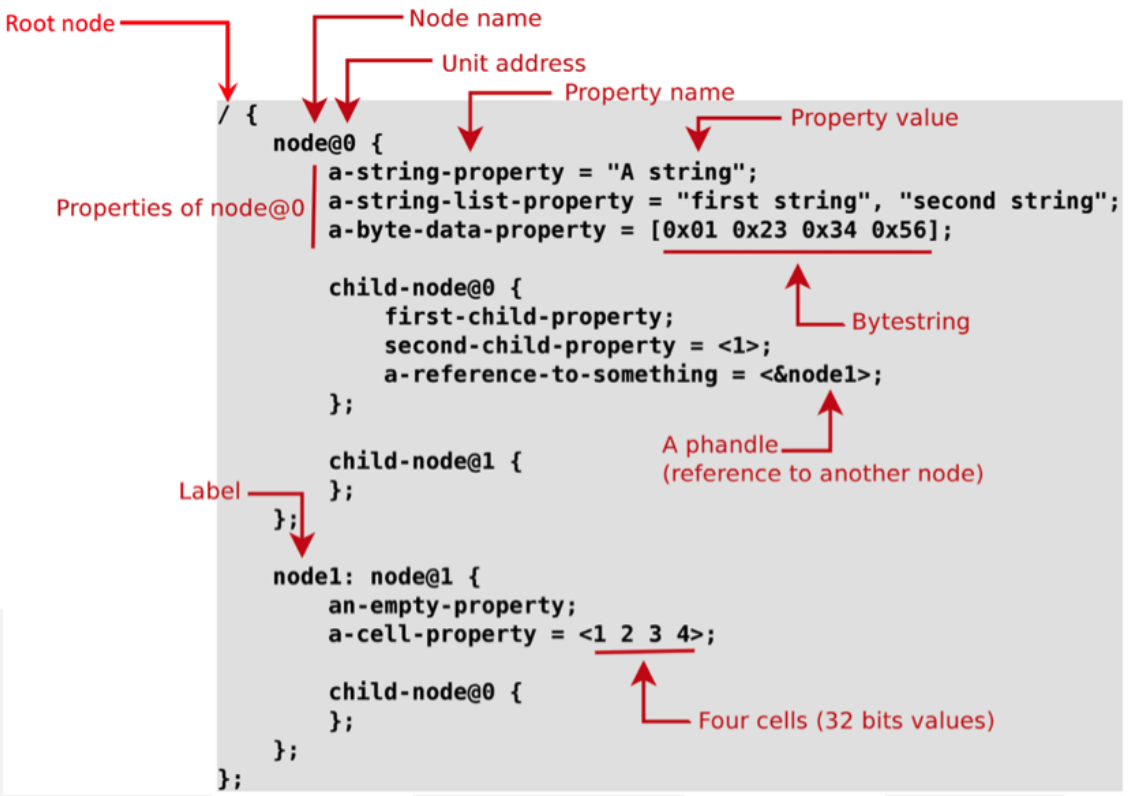
- **Aygıt** ve **Sürücü** birbirini tamamlayan iki kavramdır. Aygıt SoC içinde bulunan veya harici bağlanabilen donanımları ifade ederken sürücü bu aygıtları yöneten kernel seviyesi yazılımlardır.
- Aygıt sürücüleri, yönetecekleri donanımlarla ilgili bilgilere(arayüz bilgileri, pin bilgileri, clock, irq, bazı ön değerler, …) ihtiyaç duymaktadırlar.
- Bazı aygıtlar, arayüzleri doğası gereği çalışma zamanında otomatik tanımlanabilmektedir (PCI, USB). Bazı aygıtlar ise bu özelliğe sahip değildir (CAN, I2C, SPI, …). Bu tür aygıtlar için linux bilgilendirilmelidir.
- Genel amaçlı bilgisayarlarda, anakartta bulunan çevre birimler BIOS aracılığı ile tanımlanmaktadır.
- PCI, USB gibi arayüzlere takılan aygıtlar ise tak/çalıştır ile otomatik tanımlanabilmektedir.
- Fakat gömülü sistemlerde, borda hangi aygıt hangi arayüzden(spi, i2c, can, uart, …) bağlanmış, bu arayüz hangi pinleri kullanmaktadır, ilgili konfigürasyonları(clock, irq, …) nedir gibi bilgileri tanımlayan BIOS gibi bir mekanizma bulunmamaktadır.
- Device-tree öncesinde bu tanımlamalar platform kodu dediğimiz ve **arch/arm/mach-xxx/** ve **arch/arm/plat-xxx/** dizinleri altında yapılmaktaydı.
- Platform sayısının artmasıyla birlikte birbirinin kopyası sayılabilecek yüzlerce dizin, binlerce dosya oluşmaya başladı.
- 07.03.2011 tarihinde Linus Torvalds'ın **"Guys, this whole ARM thing is a f*cking pain in the ass."** cümlesiyle başlayan e-postası sonrası device-tree çalışmaları başlatılmış oldu.

Device Tree

- Bordun donanım bileşenlerini tanımlayan ağaç türü şeklinde veri yapısıdır.
  - ***.dts**: Bord seviyesindeki tanımlamaların olduğu kaynak dosyadır.
  - ***.dtsi**: Daha çok SoC seviyesi tanımlamaları barındıran ve *.dts dosları tarafından include edilerek kullanılan yardımcı dosyalardır.
  - ***.dtb**: device tree blob; derlenmiş device tree dosyasıdır.
- Linux açılış aşamasında **R2** register ile bellek adresi verilmiş **dtb** dosyasını yorumlar ve tanımlanan aygıtlar için bu dosyadaki bilgileri kullanarak ilgili sürücüleri aktif eder.



- Device tree düğümlerden oluşmaktadır ve bu düğümler alt düğümlere sahip olabilirler.
- Kök düğüm **/** ile ifade edilir.
- Hiyerarşik bir yapıdadır. **AHB -> APB -> I2Cx -> RTC**



- Özellikler **key = value** şeklinde tanımlanmaktadır.
- node@0 ve node@1 köke bağlı birinci derece çocuk düğümlerdir.
- node@0 için bazı özellikler tanımlıdır ve child-node@0 ve child-node@1 adında iki çocuk düğüme sahiptir.
- node1 node@1 düğümü için etiket görevi görmektedir.
- Herhangi bir özellik başka bir nodu değer olarak alabilir.
  a-reference-to-something = <&node1>

- Dahil edilen bir dosyadaki düğüm özellikleri mevcut dosyada güncellenebilir.
- SoC seviyesini tanımlayan *.dtsi dosyasında SoC kapsamlı bilgiler, SoM seviyesini tanımlayan *.dtsi dosyasında SoM kapsamında bilgiler, bord seviyesini tanımlayan *.dts dosyasında ise bord kapsamında bilgiler tanımlanabilir.

sama5d2.dtsi dosyası, ahb bus altında bulunan sdmmc çevre birimi

```
        };

        sdmmc0: sdio-host@a0000000 {
                compatible = "atmel,sama5d2-sdhci";
                reg = <0xa0000000 0x300>;
                interrupts = <31 IRQ_TYPE_LEVEL_HIGH 0>;
                clocks = <&pmc PMC_TYPE_PERIPHERAL 31>, <&pmc PMC_TYPE_GCK 31>,
                clock-names = "hclock", "multclk", "baseclk";
                assigned-clocks = <&pmc PMC_TYPE_GCK 31>;
                assigned-clock-rates = <480000000>;
                status = "disabled";
        };
```

mtsoma5_d27.dtsi dosyası, ahb altında bulunan sdmmc0 çevre birimi

```
    ahb {
        sdmmc0: sdio-host@a0000000 {
            bus-width = <8>;
            non-removable;
            pinctrl-names = "default";
            pinctrl-0 = <&pinctrl_sdmmc0_default>;
            status = "okay";
        };
```

- sama5d2.dtsi dosyası SoC seviyesini tanımlamaktadır.
- mtsoma5_d27.dtsi dosyası ise SoM seviyesini tanımlamaktadır.
- mtsoma5_d27.dtsi dosyası sama5d2.dtsi dosyasını kullanmaktadır ve **status** özelliğini **okay** olarak değiştirmiştir.
- sama5d2.dtsi dosyasında ilgili çevre birim için register kümesinin adres uzayı, çevre birime ait irq ve clock kaynakları gibi SoC seviyesi tanımlamalar yapılmıştır.
- mtsoma5_d27.dtsi dosyasında SoM üzerinde bulunan eMMC için pin tanımlamaları yapılmıştır.
- Diyelim ki taşıyıcı kartımıza sd kart koyacağız ve **sdmmc1** arayüzümüze ait pinleri SoM konnektörü ile dışarı çıkaracağız. Bu durumda sdmmc1 tanımlamasını **mtsoma5_d27.dtsi** dosyasında yaparız fakat **status** özelliğini değiştirmeyiz. SD kartı kullanacaksak bord seviyesinde **status** özelliğini **okay** yaparız.

```
<------>apb {
<------><------>qspi1: spi@f0024000 {
<------><------><------>status = "okay";
<------><------>};

<------><------>i2c0: i2c@f8028000 {
<------><------><------>pinctrl-names = "default";
<------><------><------>pinctrl-0 = <&pinctrl_i2c0_default>;
<------><------><------>dmas = <0>, <0>;
<------><------><------>status = "okay";

<------><------><------>eeprom@50 {
<------><------><------><------>compatible = "at24,24c02";
<------><------><------><------>reg = <0x50>;
<------><------><------>};

<------><------><------>pcf8563: rtc@51 {
<------><------><------><------>compatible = "nxp,pcf85363mt";
<------><------><------><------>reg = <0x51>;
<------><------><------>};
```

- mtsoma5_mtsgw.dts (smart gateway donanımı) dosyasında apb bus altında bulunan i2c arayüzüne eeprom ve rtc entegreleri bağlanmıştır.
- i2c0 düğümünde yer alan pinmux özellikleri SoM seviyesi tanımlama dosyasında yer almaktadır.

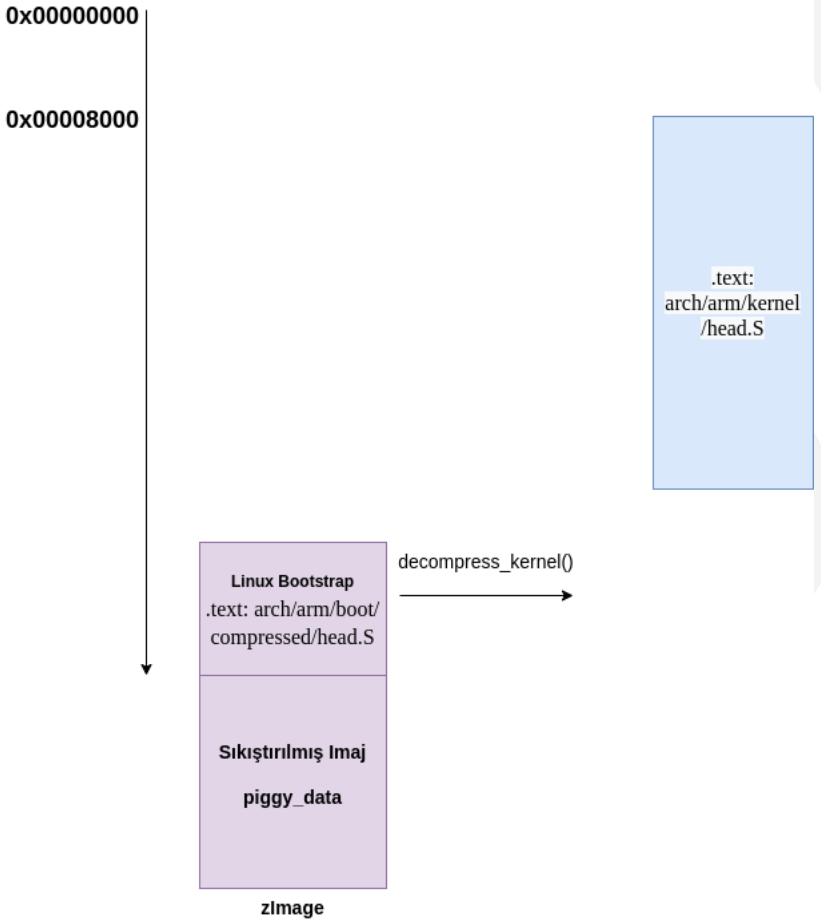
Aygıt - Sürücü Eşleştirme

- Her bir arayüz(örnek:i2c) veya aygıt(örnek:rtc) düğümünde **compatible** özelliği mevcuttur.
- Device tree uyumlu arayüz/aygıt sürücüleri de **compatible** özelliği sağlamaktadırlar.
- Kernel açılış aşamasında device tree dosyasını yorumlar ve eklenen aygıta ait compatible parametresini sürücü listesinde arar. Eşleşme sağlanırsa aygıta ait diğer özellikleri de kullanarak sürücünün **probe()** fonksiyonu çağrılır.

```
/*-----------------------------------------------------------*/

static const struct of_device_id dev_ids[] = {
    { .compatible = "nxp,pcf85363mt" },
    {}
};
MODULE_DEVICE_TABLE(of, dev_ids);

static struct i2c_driver pcf85363mt_driver = {
    .driver = {
        .name   = "pcf85363mt",
        .of_match_table = of_match_ptr(dev_ids),
    },
    .probe  = pcf85363mt_probe,
};

module_i2c_driver(pcf85363mt_driver);

MODULE_AUTHOR("Ridvan Portakal <ridvan.portakal@miltera.com.tr>");
MODULE_DESCRIPTION("PCF85363 / PCF8563  I2C RTC driver");
MODULE_LICENSE("GPL");
```

Açılış Süreci

- U-Boot, sıkıştırılmış kernel imajımızı(zImage) ve device-tree dosyamızı belleğe yükler, device-tree dosyasının bellek başlangıç adresini **R2** register'ına kaydeder ve **arch/arm/boot/compressed/head.S** dosyasında yer alan **start** alanına dallanır.
- **arch/arm/boot/compressed/misc.c** dosyasında yeralan **decompress_kernel()** fonksiyonu ile kernel, fiziksel belleğin **başlangıç adresi + 0x8000** konumuna açılır. "**Uncompressing Linux … Done**"
- Belleğin 0x100 adresinde ARM Tag List yer almaktadır bu ve bunun gibi sebepler dolayısıyla belleğin 0x8000 adresine kadar olan alanına dokunulmamaktadır.
- Sıkıştırılmış imajı açtığımızda, belleğin 0x8000 ofsetli konumunda kernelin **entry point** olarak adlandırabileceğimiz **.text** alanı yer almaktadır (**arch/arm/kernel/head.S)**.
- Burada açılan kernel belleğin ileri bir alanında yer alan sıkıştırılmış imajı ezmemelidir, bu durumda sıkıştırılmış imaj daha ileri bir noktaya kopyalanır.

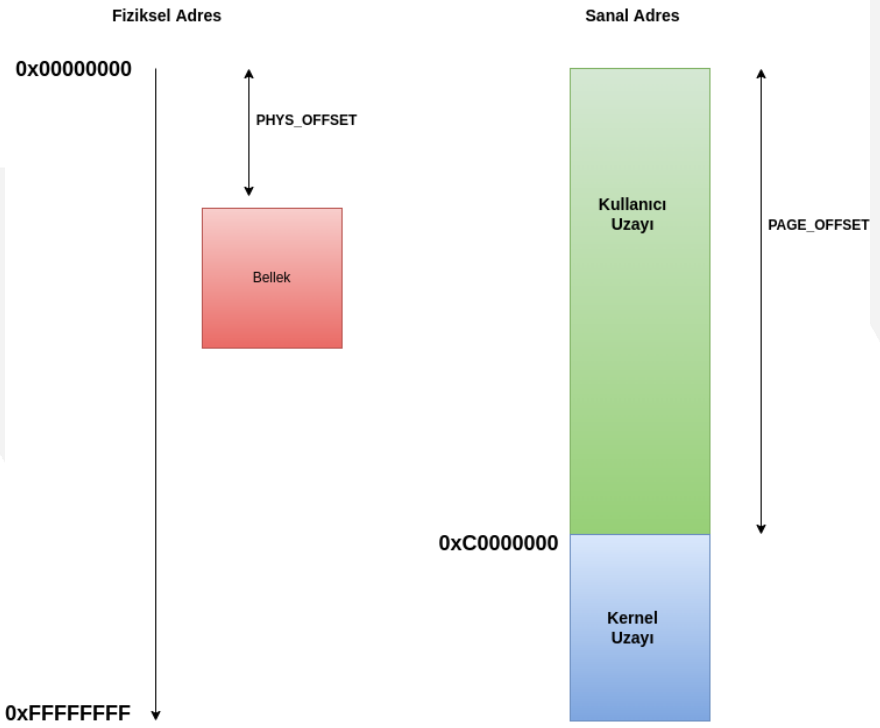


- Sıkıştırılmamış **Imaj** (vmlinux imajından elde edilen ham binary imaj**)** dosyası ile çalışmak yerine neden bu zahmete katlandık? Kalıcı bellekte daha az yer kaplıyor ve belleğe yükleme daha hızlı gerçekleşiyor.
- Linux bootstrap kodu sıkıştırılmış imajı açtıktan sonra **arch/arm/kernel/head.S** dosyasında yer alan **stext(start of text segment)** fonksiyonuna dallanacaktır.
- **arch/arm/kernel/head.S** ile mimari ve makine türü tespit edilir, MMU ve sanal bellek mekanizması aktif edilir, PAGE tabloları oluşturulur ve **init/main.c** dosyasında yer alan **start_kernel()** fonksiyonu çağrılır.
- **start_kernel()** fonksiyonu kernele ait birçok alt sistemi etkinleştirir.
    - **arch/arm/kernel/setup.c** dosyasında bulunan **setup_arch((&command_line)** fonksiyonunu çağırır. U-Boot tarafından **bootargs** değişkeni ile aktarılan kernel parametreleri yorumlanır.
    - Konsol ilklendirilir.
- **start_kernel()** gerekli ilklendirilmelerden sonra **rest_init()** fonksiyonunu çağırır.
- **rest_init()** fonksiyonu öncelikle init prosesinin 1 numaralı proses olmasını garanti altına almak için **kernel_init()** fonksiyonu için **kernel_thread** başlatır.
- **rest_init()** kernele bağlanmış veya belleğe yüklenmiş initramfs/initrd kontrolleri yapar bulamaz ise **init/do_mount.c** dosyasındaki **prepare_namespace()** fonksiyonu ile **kök dosya sistemini(rootfs)** mount eder.
- **init** prosesi için başlatılmış olan **kernel_init()** thread'i dosya sisteminde bulunan **init** prosesini çalıştırır.

Sanal Bellek ve Bellek Bölümlendirme

Yukarıda MMU, sanal bellek gibi kavramlar geçtiği için bir kaç kısa açıklama yapalım.

- 32 bit işlemciler toplamda 4GB alan adresleyebileceklerdir.
- İşlemcinin adres haritasında(memory map) fiziksel belleğin hangi aralıkta olacağı değişkenlik göstermektedir.
- Linux, sanal bellek başlangıç adresi için PAGE_OFFSET sembolünü kullanmaktadır.
- PAGE_OFFSET için 4 farklı seçenek vardır. **menuconfig -> Kernel Features -> Memory Split**
- Bu konfigürasyon ile sanal bellek yapısında kernel uzayının ne kadar, kullanıcı uzayının ne kadar adresleme yapabileceği ayarlanmaktadır. Ve genellikle **3G/1G user/kernel split** tercih edilmektedir.
- Bu seçenek ile PAGE_OFFSET 0xC0000000 değerini almaktadır ve kernel uzayı 0xC0000000 - 0xFFFFFFFF aralığını, kullanıcı uzayı ise 0x00000000 - 0xBFFFFFFF (PAGE_OFFSET - 1) aralığını kullanmaktadır.



Özet

- İşlemcimiz açılışta boot konfigürasyon pinlerini kontrol ederek önyükleyicimizin bulunduğu kalıcı hafızaya erişim sağladı.
- Kendisine bağlı bellek konfigürasyonunu bilmediği için dahili belleğine sığabilecek kadar küçük boyutlu önyükleyiciyi dahili hafızasına alarak harici belleği, ihtiyaç duyulan birkaç çevre birimi ayağa kaldırdı ve U-Boot'u harici belleğe alarak dallandı.
- U-Boot ihtiyaç duyulan çevre birimleri ayağa kaldırdı, linux ve device-tree imajlarını belleğe aldı, kernel komut satırı için **bootargs** parametresi ile device-tree dosyasında **choosen** alanını güncelledi, device-tree imajının bellek başlangıç alanını **R2** register'ına kopyladı ve sıkıştırılmış kernel imajını(zImage) kopyaladığı bellek adresine dallandı.
- Linux bootstrap kodumuz kerneli açtı ve **init/main.c** dosyasındaki **start_kernel()** fonksiyonunu çalıştırdı. Kernelimiz çalışmaya başladı.
- **start_kernel()** fonksiyonu kernel bileşenlerini hazırladı, bootloader tarafından aktarılan dosya sistemini mount etti ve **init** prosesini çalıştırdı.

Kök Dosya Sistemi - Rootfs

- Dosya sistemi, dizin ve dosyaları hiyerarşik şekilde tutan bir yapıdır, kök dosya sistemi ise linux işletim sisteminde kernel tarafından **/ (kök)** noktasına bağlanan(mount) birincil seviye dosya sistemidir.
- Kök dosya sistemi bellekte, ağda, NOR flash'da NAND flash'da, eMMC / SD kartda veya harddiskde bulunabilir.
- Kök dosya sisteminde herhangi bir dizine başka dosya sistemleri bağlanabilir.
- Kök dosya sistemi, dinamik kütüphaneler, uygulamalar, konfigürasyon dosyaları, kernel modülleri, aygıt erişim dosyaları gibi sistemin ihtiyaç duyacağı herşeyi barındırmaktadır.

```
/              Kök
├── bin        Kullanıcı uygulamaları (/bin/cat, /bin/ls, /bin/cp...)
├── dev        Aygıt erişim dosyaları (/dev/tty0, /dev/video0...)
├── etc        Konfigürasyon dosyaları
├── home       Kullanıcılar dizini
├── lib        Dinamik kütüphaneler
│   ├── modules
│   │   └── 5.10.xxx
│   │       ├── kernel        Kernel modülleri
├── proc       Sözde dosya sistemi (/proc/cpuinfo, /proc/version...)
├── root       root dizini
├── run        Çalışma zamanı bilgi dosyaları
├── sbin       Sistem uygulamaları (/sbin/fsck, /sbin/init)
├── sys        Sözde dosya sistemi
├── tmp        Geçici dosyalar
├── usr        Kullanıcı uygulama/kütüphane/… dizini
│   ├── bin    Kullanıcı uygulamaları (/usr/bin/lsusb, …)
│   ├── lib    Kullanıcı kütüphaneleri
│   ├── sbin   Sistem uygulamaları(/usr/sbin/alsactl, …)
├── var        Değişken dosyalar (/var/log/messages)
```

Dosya Sistemi Oluşturma

- Dosya sisteminde gerek bizim yazacağımız gerekse linux işletim sistemiyle gelen uygulamalara ihtiyaç vardır.
    - /bin/cp, /bin/ls, /sbin/ifconfig, /sbin/insmod, /usr/bin/find, /usr/sbin/udhcpd, …..
- Bu uygulamalar kütüphanelere ihtiyaç duymaktadır.
    - ld, libc, libdl, libm, libpthread, libresolv, libstdc++, ….
- Uygulamalar için konfigürasyon dosyalarına ihtiyaç vardır.
    - fstab, group, passwd, profile, httpd.conf, …..
- Temel kütüphaneler kullanılan toolchain ile gelmektedir, ihtiyaç duyduğumuz kütüphaneleri buradan edinebiliriz. Toolchain'de bulunmayan kütüphanelerin kaynak kodlarını edinip derlememiz gerekmektedir.
- Bazı konfigürasyon dosyaları uygulamalar tarafında otomatik oluşturulmaktadır diğerlerini kendimiz oluşturmak zorundayız.
- Uygulamaları tek tek derleyerek kullanabiliriz fakat gömülü sistemler için yeterli ve çok kolay diğer bir yol vardır, **busybox**

Busybox

- Busybox, gömülü linux projelerinde ihtiyaç duyulabilecek bir çok uygulamayı tek bir binary dosyada barındıran oldukça kullanışlı bir uygulamadır.
- Busybox tarafından sağlanan uygulamalar gnu sürümlerine göre daha az özellik barındırmaktadır ama gömülü sistemler için yeterlidir.
- Busybox uygulamalarla birlikte init yönetimi de sağlamaktadır. Bunun için **/sbin/init** uygulaması ve bu uygulamanın okuyacağı **inittab** dosyaları bulunmaktadır.

Uygulama

- Busybox inceleme
- Busybox ile basit bir dosya sistemi oluşturma
- SD kart oluşturma

Dosya Sistemi Türleri

Sözde Dosya Sistemleri

- Kernel tarafından çalışma zamanında oluşturulan, kalıcı hafızada yer almayan, kernelle haberleşme sağlayan dosya sistemleridir.
- procfs -> /proc
    - **menuconfig -> File Systems -> Pseudo filesystems -> proc file system support**
    - **mount -t proc proc /proc**
    - /proc/cmdline
    - /proc/cpuinfo
    - /proc/devices
    - /proc/meminfo
- sysfs -> /sys
    - **menuconfig -> File Systems -> Pseudo filesystems -> sysfs file system support**
    - **mount -t sysfs sysfs /sys**
    - /sys/block/
    - /sys/bus/devices
    - /sys/bus/drives
    - /sys/class
    - /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
- debugfs
- configfs

RAM Dosya Tipi

- ramdisk
    - RAM üzerinde yapay bir blok aygıt oluşturulur.
    - Boyutu oluşturma aşamasında belirlenir ve sabittir.
    - Blok aygıt okuma/yazma işlemleri söz konusudur.
    - ext2/ext3/ext4 olarak biçimlendirilebilir.
- ramfs
    - Linux'ın disk önbellekleme mekanizmasını kullanır.
    - Boyutu limitlendirilemez.
    - Mevcut boyutunu öğrenmek için **free** komutu kullanılabilir, **df** komutu işe yaramaz.
    - **mount -t ramfs ramfs /mount_dir**
- tempfs
    - ramfs eksikleri giderilmiştir.
    - Boyutu limitlendirilebilir.
    - **df** komutu ile mevcut boyutu ve limiti görülebilir.

- Kök dosya sisteminde bir çok dizin tempfs ile bağlanmıştır.
  - /dev/shm, /run, /tmp/, /run/user/<id>
- **mount -t tmpfs -o size=512m tmpfs /mount_dir**

Kalıcı Hafıza Dosya Tipi

- JFFS2 - Journalling Flash FileSystem Version 2
  - NOR ve NAND gibi flash belleklerde çalışmaktadır.
- UBI - Unsorted Block Image File System
  - Özellikle NAND flash bellekler için uygundur.
- EXT2 / EXT3 / EXT4
  - SD kart, eMMC, harddisk gibi blok aygıtlarda çalışır.

INITRD - INITRAMFS

- Kernel belleğe yüklenen kök dosya sistemini bağlayarak açılabilir.
- Genellikle kalıcı hafızadaki asıl kök sistemini bağlamadan önce sistemi hazırlamak için kullanılır ama bazı projelerde nihai kök dosya sistemi olabilir.
- initrd, ramdisk tabanlıdır ve blok aygıt olarak çalışır.
- initramfs tar benzeri basit bir arşiv dosyasıdır (cpio) ve tempfs olarak kullanılır.
- Her ikisi de kernele link edilebileceği gibi ayrı bir imaj olarak da belleğe yüklenebilir.

Açılış Yönetimi

- Kernel dosya sistemini bağladıktan sonra kullanıcı uygulamalarının başlatılması gerekmektedir.
- Kernel tarafından başlatılan ilk uygulamadır ve PID'si 1'dir.
- Kernel **init** uygulamasını bulamazsa **panic** oluşmaktadır.
- **init** uygulaması ile sistem konfigürasyonları yapılır, kullanıcı servis ve uygulamaları başlatılır.
- Kullanıcı için doğrudan veya getty ile giriş kontrollü shell başlatabilir
- **initramfs /init** sağlarken diğerleri **/sbin/init** sağlar.

Busybox

- **System V Init**'in basitleştirilmiş halidir.
- Uygulama yönetimi için **/etc/inittab** konfigürasyon dosyasını kullanmaktadır.
- Genellikle **/etc/init.d** dizininde yer alan betiklerle süreç yürütülür.

/etc/inittab

id::action:process

- **id:** Komutun çalıştırılacağı terminal
- **action**: Uygulamanın çalışma kontrolü
  - sysinit: sistem seviyesi uygulamaları ifade eder, diğerlerinden önce başlatılır.
  - respawn: Uygulama sonlandığında tekrar başlatılır.
  - askfirst: Uygulama başlatılmadan önce kullanıcı onayı istenir genellikle shell için kullanılır.
  - once: Uygulama bir kez çalıştırılır, öldüğünde tekrar başlatılmaz.
  - wait: uygulama sonlanana kadar beklenir.
  - restart: **init** uygulamasının kendisi yeniden başlatıldığında çalışacak uygulamayı tanımlar.
  - ctrlaltdel:init uygulaması SIGINT sinyali aldığında çalıştırılacak uygulamayı tanımlar.
  - shutdown: halt/reboot/poweroff komutlarında çalıştırılacak uygulamayı tanımlar.

System V Init

- **Unix System V**(AT&T tarafından geliştirilen Unix sürümü - 1983) örnek alınmıştır.
- Sistem konfigürasyonu için **/etc/default/init** dosyasını kullanır.
- Uygulama yönetimi için **/etc/inittab** konfigürasyon dosyasını kullanmaktadır.

/etc/inittab

id:runlevels:action:process

    - id: Tekil ID (max. 4 karakter)
    - runlevel: 0, 1, 2, 3, 4, 5, 6
- Runlevel özelliği kullanılmaktadır.
- **runlevel** komutu ile mevcut durum öğrenilebilir.
- inittab dosyasındaki **initdefault** satırı ile açılış runlevel değeri belirlenmektedir.
- **telinit** komutu ile runlevel geçişleri yapılabilir.

runlevel

- 0: Halt the system(shutdown)
- 1: Single user
- 2: Single user without network
- 3: Single user with network
- 4: Kullanılmıyor
- 5: Multiuser with graphic
- 6 Reboot

Açılış Betikleri

- /etc/init.d/rc dizininde bulunurlar.
- /etc/init.d/rcN.d dizini altında **Sxx** ve **Kxx** öneki ile sembolik link oluşturulur. **S**: Start, **K**:Kill, **xx**:Açılış önceliğini tanımlayan nümerik değer
- **start** ve **stop** parametrelerini yorumlayacak şekilde yazılırlar.

```
#! /bin/sh
case "$1" in
  start)
    echo "Starting simpelserver"
    start-stop-daemon -S -n simpleserver -a /usr/bin/simpleserver
    ;;
  stop)
    echo "Stopping simpleserver"
    start-stop-daemon -K -n simpleserver
    ;;
  *)
    echo "Usage: $0 {start|stop}"
  exit 1
esac
exit 0
```

Systemd(System and Service Manager)

- Servis kümesinden oluşur ve servisler arasında bağımlılık söz konusudur.

- Servisler paralel başlatılmaktadır.
- systemd servislerin durumunu kontrol eder ve gerektiğinde yeniden başlatabilir.
- Servisler **systemctl** komutu ile yönetilebilmektedir.
- Systemd **unit** adını verdiğimiz dosyalarla konfigüre edilmektedirler.
- **target,** System V runlevel eşleniği sayılabilecek servisler grubunu ifade etmektedir.
- Servis konfigürasyon dosyaları aşağıdaki dizinlerde yer almaktadır.
    - **/lib/systemd/system:** Sistemde bulunmasını istediğimiz servisler.
    - **/etc/systemd/system:** Kullanmak istediğimiz servisler. **/lib/systemd/system** dizininde bulunan dosyaya sembolik link olabileceği gibi burada da oluşturulabilir.
    - **/run/systemd/system:** Çalışma zamanı konfigürasyonları

Konfigürasyon Dosyası

- Konfigürasyon dosyası **[Unit], [Service]** ve **[Install]** alanlarından oluşmaktadır.
- **Unit**
    - **Description:** Servis tanımlayıcısı
    - **Requires:** Servisle birlikte başlatılan, ihtiyaç duyulan diğer servisler.
    - **Wants:** Requires gibi fakat ihtiyaç olan servislerde sıkıntı olursa servisimiz çalışmaya devam edecektir.
    - **Before:** Öncesinde başlamamız gereken servisler.
    - **After:** Sonrasında başlamamız gereken servisler.
- **Service**
    - ExecStart: Servisin başlatması gereken uygulamaya ait çalıştırma komut satırı.
- **Install**
    - Hangi target grubuna(runlevel) ait olduğunu tanımlar. Sistem bu target grubuna geçtiğinde servis çalıştırılacaktır.

```
[Unit]
Description=Servis for my_application

[Service]
ExecStart=/usr/sbin/my_application

[Install]
WantedBy=multi-user.target
```