embed_linux_edu Documentation

mehmet.alinbay

Feb 01, 2020

Giriş

CHAPTER 1

Gömülü Linux Nedir?

> Gömülü sistem, bilgisayarın kendisini kontrol eden cihaz tarafından içerildiği özel amaçlı bir sistemdir. Genel maksatlı, örneğin kişisel bilgisayar gibi bir bilgisayardan farklı olarak, gömülü bir sistem kendisi için önceden özel olarak tanımlanmış görevleri yerine getirir. Sistem belirli bir amaca yönelik olduğu için tasarım mühendisleri ürünün boyutunu ve maliyetini azaltarak sistemi uygunlaştırabilirler. Gömülü sistemler genellikle büyük miktarlarda üretildiği için maliyetin düşürülmesinden elde edilecek kazanç, milyonlarca ürünün katları olarak elde edilebilir.

Kaynak

Yukarıdaki paragrafa bağlı olarak Gömülü Linux ise belli amaca yönelik program(lar) koşturan elektronik bir sistemdir. Bu nedenle Gömülü Linux sistemi normal PC dağıtımlarından(Ubuntu, Fedora vs.) farklı olarak sadece amacına yönelik programları içermelidir, örneğin bir fabrikada kayıt işlemlerini takip eden bir sistemde torrent istemcisine gerek yoktur.

Aynı zamanda tekrar yukarıda bulunan paragrafa atıfla sistemin olabildiğince ucuz olması gereklidir bu nedenle *işi yapabilecek kadar* sistemin hafif olması büyük olasılıkla maliyetleri azaltacaktır.

Gömlü Linux ile masaüstü linux sistemlerin karşılaştırması aşağıdaki linklerden incelenebilir.

http://embeddedcraft.org/embedlinuxdesktoplinux.html

http://embeddedcraft.org/embeddedlinux.html

CHAPTER 2

Gömülü Linux Mimarisi

Linux bir sistem temelde 4 parçadan oluşur:

- Bootloader
- Kernel
- Dosya Sistemi (RootFS)
- Kullanıcı Alanı (userspace) Dosyaları

Kullanıcı alanı dosyaları RootFS içerisinde bulunur ancak bunlar proje kapsamında geliştirdiğimiz programlar, scriptler olduğu için ayrıca bildirmek istedim.

//TODO # Neden Gömülü Linux Kullanılmalı?

CHAPTER 3

Motivasyon ve Amaç

Gömülü Linux dünyası şuan çeşitli **_PI** boardların işgali altında. Bu durum donanımsal açıdan baktığınız zaman mükemmel, S3C2416'lı boardların hayal olduğu günlerden Kadıköy'den RasPI alabildiğiniz günlere gelebilmek.

Ancak bu boardlar hazır dağıtımlarla geliyor ve işin sadece uygulama geliştirme kısmını insanlara bırakıyor. Yanlış anlaşılmasın bu durum birçok kişinin gömülü linux dünyasına geçişini kolaylaştırıyor ancak endüstriyelleşme aşamasında maalesef ciddi bir israf söz konusu. Aynı zamanda bu durum oluşan hatalara karşıda geliştiricileri savunmasız bırakıyor, bir nevi enerjiyi verdiğinizden uygulamaya kadar olan kısım tam bir **kapalı kutu**.

Tabii bu kutunun huyunu suyunu bilmeden yapılan geliştirme ne kadar sağlıklı olur, takdiri size bırakıyorum.

Son olarak hazır dağıtımlara olan bağımlılık can sıkıcı olabilir örneğin armbian mükemmel işler çıkarıyor ancak Allwinner A20 bazı kartlara desteği kesmiş ve TFT sürücü desteği yok (yada ben bulamadım).

Bu kitabın amacı ise; bu kapalı kutu hakkında bilgi sağlamak ve farkındalık sağlamak, hazır dağıtımlardan sizi kurtarmak :)

CHAPTER 4

Yöntem ve Araçlar

Bu kitapta Gömülü Linux sistemini oluşturmak için sadece Linux komut satırı (shell, kabuk) kullanılacaktır. Bootloader ve Kernel imajları için özel bir araç kullanılmayacaktır ancak RootFS (kök dosya sistemi) oluşturmak için Buildroot kullanılacaktır.

Kernel ve Bootloader için yaptığımız değişiklikler gömülü linux sistemlerinin de-facto versiyon kontrol standardı olan git yaması (patch) olarak saklanacaktır. Bunun nedenlerini ve sonuçlarını kitapta ilerleyince daha iyi anlayacaksınız.

Geliştirme için kullanılacak sistem **Ubuntu 16.04 LTS** olacaktır. Hazır bir sanal makinayı ileriki günlerde yüklemeyi düşünüyorum. Tavsiyem ise daha hafif ve Windows tip arayüzü olması nedeniyle Linux Mint 18.3. Bu sistemler Debian'dan türetildiği için aralarında bizim komut satırında yapacağımız işlemler için fark bulunmamaktadır.

Geliştirme yapacağımız kart ise Beaglebone Black (BBB) olacaktır. Her ne kadar yapacağımız işler elektronik kart bağımsız olarak hedeflenmiş olsa da kartlarda çıkan ufak farklılıklar (örneğin RasPI'de ki bootloader) başlangıç aşamasında canınızı sıkmaması için BBB tavsiyemdir.

CHAPTER 5

Öneriler

- Komut satırından her işinizi yapmaya çalışın, eliniz mouse'a gitmesin :)
- Komut satırına hakim olmayanlar için Bootlin'in dökümanına göz atın.
- Yaptığınız tüm aşamaları çıktıları ile birlikte **anlaşılır** bir biçimde not edin.
- Sanal Makina yerine sabit bir sistem kurulması performans ve olası hataları engelleyeceği için tavsiye edilmektedir.
- İlk seferde zaten çalışmayacaktır, bu nedenle code hard...

CHAPTER 6

Kaynaklar

İnternette bir dolu örnek ve kaynak bulunmaktadır ancak benim yararlandığım ve fayda gördüğüm kaynakların bir kısmı şu şekildedir:

- Nazım Koç Hoca'nın websitesi ve kitabı
- İngilizce bilenler için Bootlin

CHAPTER 7

Lisans

Bu websitesinde yayınlanan herşey ile ilgili lisans : Creative Commons Attribution - Share Alike 3.0

Lisansla ilgili kurallara göz atarsanız sevinirim.

CHAPTER 8

Klasör Organizasyonu

Bu kitapta takip edilecek bir klasör organizasyonu şu şekildedir:

- workspace
  - doc
  - uboot
  - linux
  - buildroot
  - download
  - sdk

`sdk` klasörü toolchain(ler)i koyacağımız, `doc` klasörü internetten indirdiğimiz dökümanları koymak için kullanacağımız klasörlerdir. Diğer klasörler isimlerinden zaten anlaşılıyor.

Bu klasör grubu `/opt` klasörü altındadır. `/opt` klasörünün genelde yetkisi `root` kullanıcısındadır. Bunu değiştirmemiz gereklidir.

```
sudo mkdir /opt/workspace/
sudo chown $USER:$USER /opt/workspace/
cd /opt/workspace
```

sudo apt install sed make binutils gcc g++ bash patch git gzip bzip2 perl tar cpio python unzip rsync wget libncurses-dev

CHAPTER 9

Toolchain ve Çapraz Derleyici(Cross-Compile) Kavramları

> Çapraz Derleyici üzerinde çalıştığı platformdan farklı olan platformlar için yürütülebilen kodlar üreten derleyicidir. Böyle bir araç, erişiminizde olmayan bir platform için kod derlenmesi gerektiğinde ya da böyle bir platform üzerinde kod derleme işleminin yapılmasının imkânsız olduğu (gömülü sistemlerde olduğu gibi, mikrokontrolörler minimum bellek ile çalıştığı için derleme imkânsız olur) durumlarda faydalı olabilir.

Kaynak: Wikipedia

Cross-Compile (CC) aslında mikrokontrolcülerle uğraşan insanların hergün yaptığı bir işlem. Temelde x86 tabanlı makinalarda çalıştığımızdan ve gömülü sistemler farklı tabanlarda (arm, MIPS, risc-V) olduğundan CC işlemine gerek duyarız. Eğer hedef makina ile aynı tabanda uygulama geliştiriyorsak çapraz derleme değil sadece derleme işlemi yaparız.

İnternette direk olarak gömülü sistem üzerinde derleme yapan örnekler mevcut ancak hem gömülü sistemlerin daha zayıf makinalar olması hem de gömülü sistemlerde çalışma durumunda kullanılmayacak gereksiz paketlerin kurulması nedeniyle ben böyle yöntemlere sıcak bakmıyorum.

Çapraz derleme ile ilgili daha başka yöntemlerde mevcut ancak bu kitapta bunları işlemeyi düşünmüyorum.

Peki Toolchain nedir?

> Toolchain, bilgisayar yazılımı dilinde Araç Zinciri, bir ürün (genellikle başka bir bilgisayar programı veya sistem programları) oluşturmak için kullanılan programlama araçlarının setidir. Araçlar bir zincir içerisinde kullanılabilir, böylece her bir aracın çıktısı bir sonrakinin girdisi olmuş olur. Fakat bu terim yaygın bir şekilde, bağlı geliştirim araçlarının herhangi bir setine işaret eder.

Kaynak: Wikipedia

Özetle bizim **cross-compile** yapmak için **toolchain'e** ihtiyacımız var.

CHAPTER 10

Toolchain Kurulumu

Günümüzde ARM mimari için toolchain genellikle Linaro'nun sağladığı toolchain kullanılmaktadır. Linaro ARM mimarisi için açık-kaynak yazılımlar (Linux Kernel, GCC Toolchain vs.) üreten bir şirkettir. Linaro ST, Samsung, Quallcomm, TI, Alibaba vs. gibi onlarca firma tarafından desteklenmektedir. Bu nedenle bir bakıma ARM için gerekli olan toolchain konusunda endüstri standardı durumundadır.

Geliştirme yapacağımız sisteme Linaro toolchain kurmak için birkaç seçeneğimiz var. Benim önerim eğer başlangıç olarak `Adım-2` ile yürümek şimdilik yeterlidir.

10.1 1. Ubuntu Repo'dan kurmak

```
sudo apt-get install gcc-arm-linux-gnueabihf
```

Açıkçası bu yöntemi kurduğu toolchain versiyonun farklı tarihlerde farklı olacağı nedeniyle pek tercih etmiyorum.

10.2 2. Linaro websitesinden indirmek.

Linaro derlenmiş haldeki toolchain'i https://releases.linaro.org/components/toolchain/binaries/ adresinde yayınlamaktadır. Burada istediğiniz bir versiyonu indirip kullanabiliriz.

Örneğin 7.3'ü kurmak için öncelikle indirip `toolchain` klasörüne açalım.

```
cd workspace/download
wget https://releases.linaro.org/components/toolchain/binaries/7.3-2018.05/arm-linux-
↪gnueabihf/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf.tar.xz
tar xvf gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf.tar.xz -C /opt/workspace/
↪sdk/
```

Bu noktadan sonra yapılması gereken toolchain'i `PATH` içine eklemek. Bunun için aşağıdaki komutu çalıştırmamız gerekir.

```
export CC=/opt/workspace/sdk/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin/
↪arm-linux-gnueabihf-
```

Kontrol etmek için aşağıdaki komut kullanılır.

```
{CC}gcc --version
```

Çıktı şu şekilde ise işler yolunda demektir.

> arm-linux-gnueabihf-gcc (Linaro GCC 7.3-2018.05) 7.3.1 20180425 [linaro-7.3-2018.05 revision d29120a424ecfbc167ef90065c0eeb7f91977701]Copyright (C) 2017 Free Software Foundation, Inc.This is free software; see the source for copying conditions.There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Bu yöntemde her yeni terminal açtığımızda komutu tekrarlamamız gerekmektedir. Bunun yerine toolchain'i `PATH` içine otomatik olarak ekleyebiliriz. Ancak bu yöntemde farklı toolchainler (örneğin aynı makinada hem Linaro hem de CodeSourcery gerekiyorsa) kullanmanız gerektiğinde işler biraz karışabiliyor, bu nedenle ben her terminal açılışında `PATH` içine manuel olarak ekliyorum.

Eğer yine de otomatik olarak yapmak isterseniz.

```
nano ~/.bashrc
```

son satıra aşağıdaki bölüm eklenir ve dosya kaydedilir.

```
PATH=/opt/workspace/sdk/arm/bin:$PATH
export PATH
```

Doğru ayarlandığını kontrol etmek için mevcut terminal kapatılır ve yeni terminal açılır, aşağıdaki komut ile kontrol edilir.

```
which arm-linux-gnueabihf-gcc
```

10.3 3. TI veya üreticinin verdiği toolchaini kurmak

TI Processor-SDK adı altında belirli aralıklarla U-Boot, Linux, Toolchain ve çeşitli yazılımları yayınlamaktadır.

AM335x serisi için Processor SDK

Bu kitabı yazarken en son versiyon 5.03'tü. Download Linki

TI SDK'nin avantajı üretici tarafından yayınlandığından TI forumlarında 8zaten ilk sordukalrı soru SDK versiyonu) daha rahat destek bulabiliyorsunuz ve kısmen TI destekli bir platformda geliştirme yapıyorsunuz.

SDK kurulumu şu şekildedir:

```
cd /opt/workspace/download
wget http://software-dl.ti.com/processor-sdk-linux/esd/AM335X/latest/exports/ti-
↪processor-sdk-linux-am335x-evm-05.03.00.07-Linux-x86-Install.bin
chmod +x ti-processor-sdk-linux-am335x-evm-05.03.00.07-Linux-x86-Install.bin
./ti-processor-sdk-linux-am335x-evm-05.03.00.07-Linux-x86-Install.bin
```

SDK'yi `/opt/workspace/ti-processor-sdk-linux-am335x-evm-05.03.00.07` klasörüne kuracağız.

Processor SDK içindeki Toolchain `ti-processor-sdk-linux-am335x-evm-05.03.00.07/linux-devkit/sysroots/x86_64-arago-linux/usr/bin` klasörü altındadır. Bir önceki adımda `PATH` içine toolchain ekleme açıklandığı için tekrarlanmayacaktır.

10.4 4. Buildroot Toolchain'i kullanmak.

Açıkçası benim tercihim bu yöntem. Çünkü RootFS'i ve geliştireceğimiz uygulamaları Buildroot'un toolchain ile derleyeceğiz. Bu nedenle Linux ve U-Boot'un da bu toolchain ile derlenmesini tercih ediyorum.

Şu aşamada Buildroot'u bilmediğimiz varsayarsak; sadece basit bir derleme yapacağız ve onun çıktısı olan SDK'yı kullanacağız.

Buildroot ile Basit Derleme başlığında bu konu anlatılmaktadır.

CHAPTER 11

SD-Kart Hazırlama

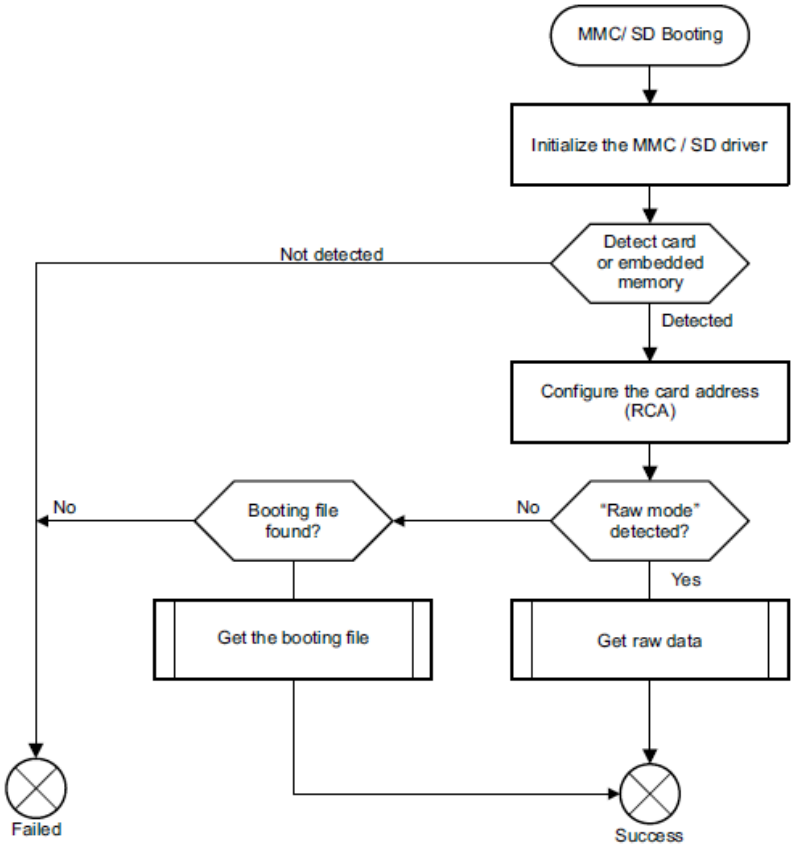
AM335x için SD-Kartın belirli formatta hazırlanması gerekmektedir. AM335x RBL aşamasında SD-Kart içerisinde FAT bir bölüm arar ve bu bölüm içinde SPL dosyasını arar.

AM335x'in MMC/SD Kart Boot prosesi şu şekildedir:

26.1.8.5.3 Booting Procedure

The high level flowchart of the eMMC / eSD and MMC/SD booting procedure is depicted in Figure 26-22.

Figure 26-22. MMC/SD Booting



alt text

SPL Dosyası U-Boot'u, U-Boot Linux Kerneli yükleyecektir. Bu dosyaların hepsi genellikle FAT bölümde olur.

RootFS ise SD-Kart üzerinde başka bir bölümde (genellikle ext4 formatında) olacaktır.

Özetle SD_Kart üzerinde FAT ve EXT4 formatında olan iki ayrı bölüm olacaktır, gereken adımlar şu şekildedir:

SD-Kart takıldığı zaman Ubuntu tarafında hangi `/dev/` altında yeni bir cihaz olarak belirecektir. Bu isim genelde `sdb`, `sdc` gibi isimlerle ortaya çıkar. Yanlış bir isim ile işlem yaparsanız mevcut diskinizi uçurabilirsiniz.

Benim sistemimde SD-Kart `/dev/sdb` olarak gözükmektedir.

Öncelikle SD-Kartın mevcut MBR silinir.

```
sudo dd if=/dev/zero of=/dev/sdb bs=1M count=16
```

SD-Kart bölümlendirilmesi için `fdisk` kullanacağız.

```
sudo fdisk /dev/sdb
```

1. `o` ile yeni bir DOS tablosu oluşturulur.
2. `n` ile yeni bir bölüm oluşturulur. Sırasıyla `p` , `1` seçilir. Başlangıç sektörünü değiştirmeyeceğimiz için `enter` ile kabul edilir. Bundan sonraki aşamada birinci bölümün boyutunu belirlemek için değer girilir, biz `+32M` girelim.

3. Birinci bölümün tipini W95 FAT16 (LBA) yapmak için sırasıyla `t` ve `e` seçilir.
4. Birinci bölümün boot edilebilir flagini setlemek için `a` ve `1` seçilir.
5. RootFS için gerekli olan ikinci bölüm için `n` ile yeni bir bölüm oluşturulur. Sırasıyla `p` , `2` seçilir. Başlangıç sektörünü değiştirmeyeceğimiz için `enter` ile kabul edilir. Bundan sonraki aşamada ikinci bölümün boyutunu belirlemek için değer girilir, biz `+256M` girelim.
6. Son olarak yaptıklarımızı `p` ile kontrol edelim. Şöyle bir tablo görmemiz gereklidir.

```
Command (m for help): p
Disk /dev/sdb: 14.6 GiB, 15634268160 bytes, 30535680 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x63db9b68

Device     Boot Start    End Sectors  Size Id Type
/dev/sdb1  *     2048  67583   65536   32M  c W95 FAT32 (LBA)
/dev/sdb2       67584 591871  524288  256M 83 Linux
```

1. `w` ile yaptığımız işler kaydedilir ve çıkılır.

Oluşturduğumuz SD-kart bölümlerine dosya sistemleri kurulur.

```
sudo mkfs.vfat /dev/sdb1 -n "boot"
sudo mkfs.ext4 /dev/sdb2 -L "rootfs"
```

Yukarıdaki SD-Kart işlemlerini otomatik olarak yapmak için hazırladığım scripti aşağıdaki komutlar ile kullanabilirsiniz.

```
chmod 755 mksdcard.sh
sudo ./mksdcard.sh /dev/sdb
```

CHAPTER 12

BBB için UART Konsol Bağlantısı

BBB UART0 portundan gelecek olan verileri izlemek için bir adet USB-TTL UART çevirici ve `picocom` programını kullanacağız. Programın kurulumu ve kullanıcının `dialout` grubuna katılması (bu sayede her picocom başlangıcında sudo yazmaktan kurtulacağız) için gereken komutlar şu şekildedir.

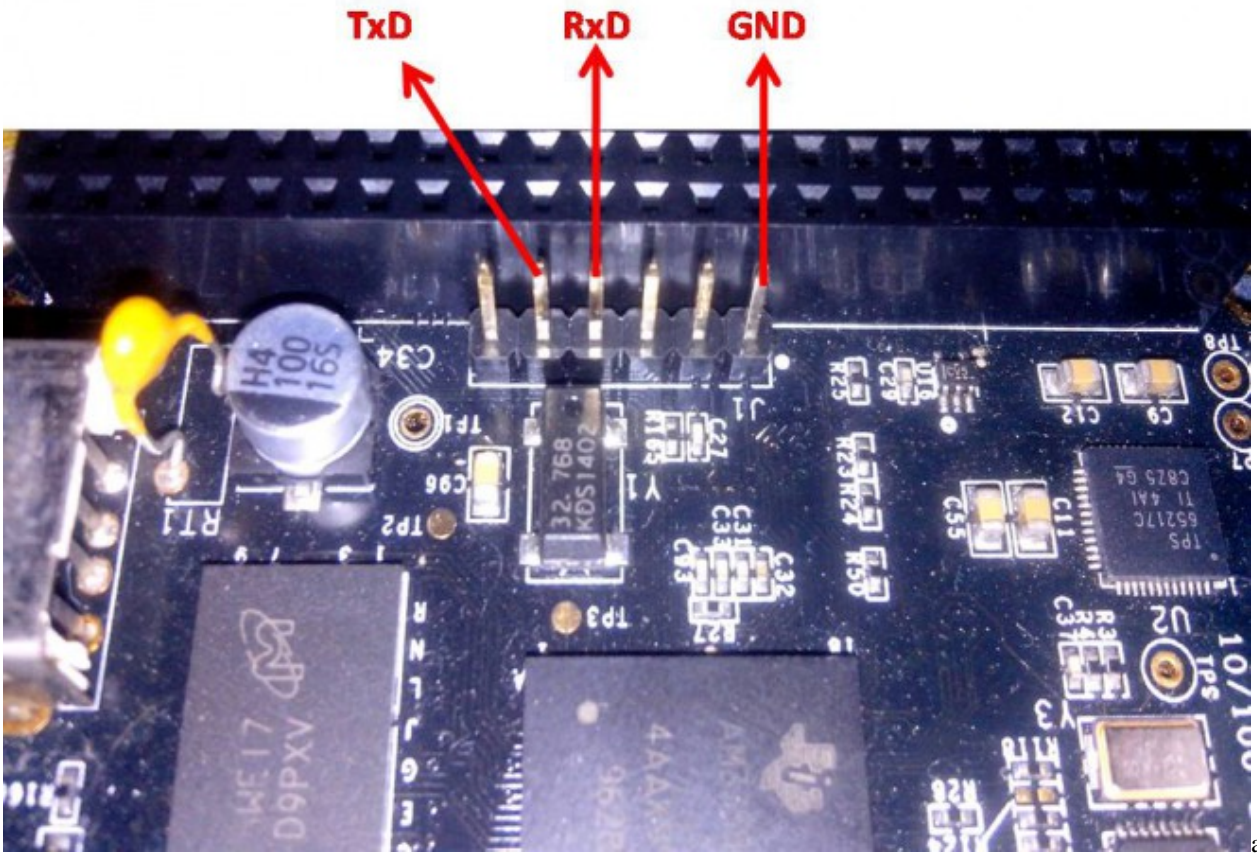
```
sudo apt install picocom
sudo adduser $USER dialout
```

Host makina restart edilir. `picocom` kullanmak ise oldukça basittir

```
picocom -b 115200 /dev/ttyUSB0
```

picocom'dan çıkmak için `CTRL+C` kullanılmalıdır.

USB-TTL UART çeviriciyi karta aşağıdaki şekilde bağlayacağız.



alt text

CHAPTER 13

Bootloader ve U-Boot

Bootloader, bilgisayar yada gömülü bir sistem ilk enerji verildiğinde çalışan uygulamalardır. Bu programların temel amacı işletim sistemi için CPU ve çevrebirimlerin hazırlanmasıdır.

PC(x86) makinalar için kullanılan bootloader bilindiği üzere BIOS'tur. Gömülü Linux sistemlerini genelde ARM mikroişlemciler kullanılmaktadır ve en yaygın olarak kullanılan bootloader U-Boot'tur.

U-Boot Denx firması tarafından geliştirilen ücretsiz genel amaçlı bir bootloaderdır. U-Boot bir çok mimariyi desteklemektedirler(ARM, MIPS, x86 vs.) U-Boot günümüzde özellikle gömülü linux sistemleri için *de-facto* standart haline gelmiştir.

U-Boot dökümantasyonu şu adreste bulunabilir. http://www.denx.de/wiki/U-Boot/Documentation

CHAPTER 14

Boot Prosesi

Bu bölümde kitapta kullanacağımız Beagleboard Black kartının işlemcisi olan AM335x serisinin boot prosesi incelenecektir. Örnek olarak QSPI Flash'tan Kernel imajı yüklenecektir, boot prosesi ise şu şekildedir:

- CPU içerisinde ilk açılışta herhangi bir yazılım yoktur, bu nedenle CPU içerisine donanımsal olarak gömülmüş olan ROM Bootloader(RBL) CPU tarafından yürütülmeye başlanır.
- RBL çevrebirimleri tarar ve bunların içerisinde İkinci Bootloader'ı (SPL) yükler.
- SPL RAM, QSPI vs. gibi en temel çevrebirimleri kurar. SPL çeşitli isimlerde UBL, XLoader, MLO vs. gibi adlandırılmaktadır.
- SPL QSPI'dan U-Boot bootloader dosyasını RAM'e yükler.
- U-Boot diğer çevrebirimleri kurar(Ethernet, USB vs.)
- U-Boot Kernel imajını RAM'e yükler.

Peki yukarıdaki süreçte neden SPL'e ihtiyaç vardır? Bunun cevabı için AM335x datasheetine bakmak gereklidir. AM335x'te dahili olarak bulunan RAM'in boyutu U-Boot'u karşılayacak seviyede değildir, bu nedenle harici RAM'in kurulması gereklidir ancak harici RAM karttan karta farklılık göstereceğinden RBL tarafından kurulamaz.

Bu nedenle ara bir çözüm olarak dahili RAM'den koşabilecek ve U-Boot'u harici RAM'e yükleyecek daha küçük bir bootloader olarak SPL kullanılmaktadır. SPL oluşturmak için ayrı bir derleme sürecine gerek yoktur, U-Boot bize SPL seçeneğini bize sunmaktadır.

U-Boot'un yüklenmesi yerine SPL'den direk olarak Kernel imajını yüklemekte mümkündür.

> TI işlemciler için boot prosesi ile ilgili detaylı kaynak

CHAPTER 15

U-Boot Giriş

U-Boot Linux çekirdeği gibi kendine özel bir konsolu ve komutları vardır. Bu komutlar ve belirli parametreleri kullanarak U-Boot'u yönlendiririz.(çekirdeği nerede arayacak, çekirdeğe hangi parametreleri verecek vs. gibi)

15.1 U-Boot Komutları

U-Boot komutlarının bir kısmı için link

Önemli olan U-Boot komutları şunlardır(kullanımları aşağıda verilmemiştir) :

- `bootz` , `bootm` Belli bir RAM adresindeki zImage formatındaki kerneli yürütür.
- `fatload` DOS dosya sistemine sahip cihazdan dosyayı RAM adresine yükler.
- `ext2load` EXT2 dosya sistemine sahip cihazdan dosyasını RAM adresine yükler.
- `ping` ağ testi için kullanılır.
- `tftp` ağdan dosya çekip RAM'e yüklemek için kullanılır.
- `dhcp` ağda bulunan DHCP serverdan IP almak için kullanılır.
- `mmc` eMMC, SD-Kart donanımlarına yönelik komutları içerir.
- `help` Varolan komutları listeler.

U-Boot komut seti bizim yapacağımız derlemeye göre değişkenlik gösterebilir. Örneğin FAT dosya sistemi ile ilgili komutları derleme esnasında yapacağımız konfigürasyona göre kapatıp/açabiliriz.

15.2 U-Boot Ortam Değişkenleri

U-Boot parametre ve komut bütününe ortam değişkenleri (environment variables) denir.

U-Boot ortam değişkenlerini derleme öncesinde, derleme sonrasında harici bir dosya ile veya komut satırından ayarlayabiliriz. Komut satırından yapılan değişiklikler eğer hedef cihazda U-Boot üzerinden kaydetme yapılıyorsa varsa kalıcı olarak saklanabilir. Ancak bu durum üretim için pek pratik değildir.

Ortam değişkenlerinin derleme öncesinde değiştirilmesi daha sonra işlenecektir.

Ortam değişkenleri ile ilgili U-Boot komutları şunlardır:

- `printenv` tüm ortam değişkenlerini listeler
- `setenv` RAM'de ortam değişkeni varsa değiştirir yoksa ekler.
- `saveenv` RAM'de olan ortam değişkenlerini kaydeder.

Harici ortam değişkeni dosyası olarak çeşitli formatlar var ancak bizim sistemimiz için `uEnv.txt` dosyası kullanılacaktır. U-Boot `uEnv.txt` dosyasını bulursa mevcut ortam değişkenlerini ezer.

Bazı önemli ortam değişkenleri şunlardır:

- `bootcmd` U-Boot'un boot amaçlı çalıştıracağı komuttur.
- `bootargs` U-Boot'un kernele ileteceği parametreleri tutar.
- `ipaddr` Cihazın IP Adresi
- `serverip` tftpboot komutu için gerekli olan server IP adresi

Ortam değişkenleri ufak scriptler olabilirler, bu sayede çeşitli U-Boot komutları tek bir ortam değişkeni ile tanımlanabilir.

15.3 Örnek : uEnv.txt

```
loadaddr=0x82000000
fdtaddr=0x88000000
arg=setenv bootargs console=ttyO0,115200n8 root=/dev/mmcblk0p2 rw rootfstype=ext4␣
↪rootwait
image=load mmc 0:1 ${loadaddr} uImage ;
fdt=load mmc 0:1 ${fdtaddr} am335x-bonegreen.dtb ;
uenvcmd=run arg;load image;load fdt;bootm ${loadaddr} - ${fdtaddr};
```

`loadaddr` –> RAM adresi.

`fdtaddr` –> RAM adresi.

`arg` –> `setenv` komutu ile `bootargs` değişkeni; konsol ttyO0 (UART0) ve rootfs'in mmcblk0p2 içinde ext4 biçiminde olacak şekilde ayarlanır. `bootargs` içindeki parametreler kernele bildirilir.

`image` –> mmc0:1 (sd-kart boot bölümü) bölümünden uImage dosyasını okur ve loadaddr adresine yükler.

`fdt` –> mmc0:1 (sd-kart boot bölümü) bölümünden am335x-bonegreen.dtb dosyasını okur ve fdtaddr adresine yükler.

`uenvcmd` –> U-Boot'un yürüteceği ilk komuttur. Örneğimizde görüldüğü üzere U-Boot sırasıyla `arg` , `image` , `fdt` komutlarını ve son olarak `bootm` komutunu yürütecektir.

CHAPTER 16

U-Boot Derleme

16.1 U-Boot Edinme

U-Boot indirmek için iki adet kaynağımız var. Denx veya BBB işlemcisinin üreticisi olan Texas Instruments (TI) websitelerinden.

TI; tekrar etmeke gerekirse Processor SDK adı altında U-Boot ve Linux için kaynaklarından aldığı snapshotların üzerine kendi işlemcileri için ek geliştirmeler koyarak dağıtımını yapmaktadır. Esas hedefimiz **endüstriyel** olduğu için ben TI'nin sağladığı U-Boot ve Linux kaynak kodlarını kullanmayı tercih ediyorum.

Biz yine de Denx'den indirmenin komutunu verelim.

```
git clone git://git.denx.de/u-boot.git
```

Peki TI'dan nasıl indireceğiz? Bunun için iki yol var.

1. TI Processor SDK (Toolchain kurma bölümünde anlatıldı)
2. git.ti.com 'dan indirme

TI GIT üzerinden indirerek işlemlerimizi yapalım.

```
cd /opt/workspace
git clone git://git.ti.com/ti-u-boot/ti-u-boot.git uboot
cd uboot/
```

GIT ile indirdiğimiz için birçok branch bulunacaktır. Önce branchlere bakalım

```
git branch -a
```

Biz çalışmalarımızı **Processor SDK 5.03'e** paralellik göstermesi için **2018.01** branch ile yapalım.

```
git checkout ti-u-boot-2018.01
```

Bu noktada işlerin karışmaması için kendi branchimizi oluşturmamız gerekiyor. Branch ismi **beagle_dev** olsun.

```
git checkout -b beagle_dev
```

Yaptığımız kontrol edelim.

```
git status
```

On branch beagle_dev nothing to commit, working directory clean

Artık U-Boot üzerinde yapacağımız tüm değişiklikler git tarafından takip edilecek.

16.2 U-Boot Derleme

Temelde Linux, U-Boot, Buildroot derlemek için en gerekli dosya **config** dosyasıdır. Eğer bir boarda ilk defa derleme yapıyorsanız başlangıç için çalışan bir `config` dosyası bulunmaz bir nimettir (özellikle Çin tabanlı işlemcilerle olan kartlarda)

`config` dosyaları U-Boot içerisinde `configs/` altında bulunur. Bu klasörün içinde bazı kartlar için `config` dosyaları bulunmaktadır.

Eğer sizin kartınız ile ilgili `config` dosyası bulamazsanız internetten aramanız gerekiyor yada gözünüzün kestirdiği bir `config` ile şansınızı deneyebilirsiniz.

```
ls /opt/workspace/uboot/configs
```

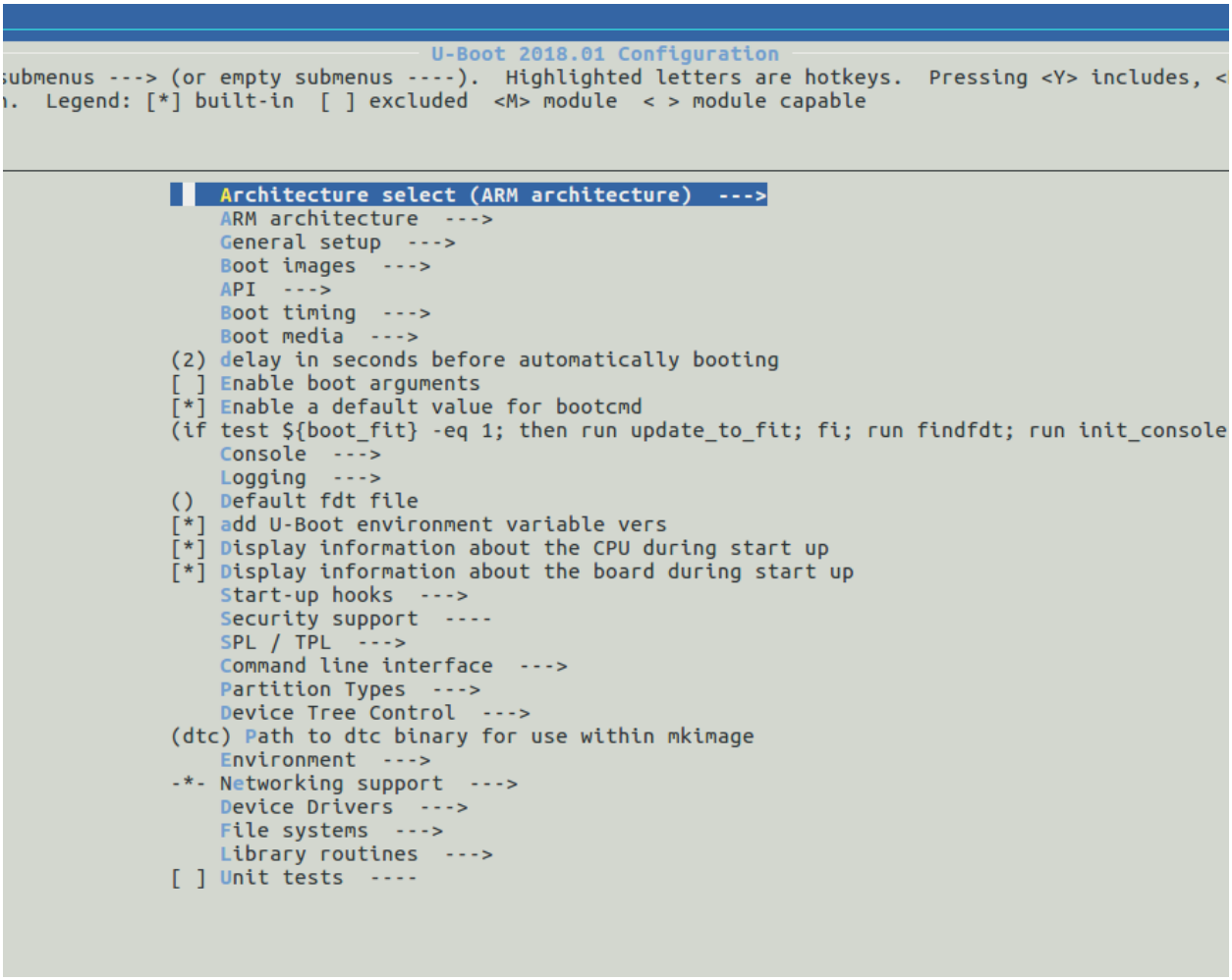
Biz derlememizi Processor SDK dökümanında yazdığı üzere BBB uyumlu olan `am335x_evm_config` dosyası ile yapacağız.

```
cd /opt/workspace/uboot
make am335x_evm_config
```

Yukarıdaki komutu yürüttüğümüzde U-Boot `am335x_evm_config` dosyasına göre ayarlanır. Aslında yapılan işlem U-Boot ana klasöründe bulunan .config dosyasını güncellemektir.

Kontrol edelim.

```
make menuconfig
```



Resimde görüldüğü üzere U-Boot 2018.01 versiyonda ve ARM çekirdek için derleme yapacak.

Peki bir derleme yapalım.

```
export CC=/opt/workspace/sdk/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin/
↪arm-linux-gnueabihf-
make ARCH=arm CROSS_COMPILE=$CC -j4
```

Toolchain olarak Linaro'dan indirdiğimiz toolchain'i kullandık ve `CROSS_COMPILE=$CC` ile toolchain'i bildiriyoruz.

`ARCH=arm` parametresi ile derlememizi ARM çekirdek ile yapacağımızı bildiriyoruz.

`-j4` parametresi ile derleme için kaç çekirdek kullanılacağını bildiriyoruz. Eğer ilk defa derleme yapıyorsanız bu parametreyi kullanmayın. Bu sayede derleme hatalarını görmemiz daha kolay olacaktır.

Derlememiz başarılı ise çıktılar uboot klasörü altında olacaktır. Bizim işimize yarayacak dosyalar `MLO` ve `u-boot.img` dosyalarıdır.

`MLO` Rom Bootloader tarafından çağrılacak olan SPL yani ön U-Boot yükleyicisidir.

`u-boot.img` ise isminden de çıkarabileceğimiz üzere U-Boot'un kendisidir.

`MLO` ile `u-boot.img` dosyalarının boyutlarına bakmanızı tavsiye ederim.

SD-Kartı hazırlama ile kartımızı hazırlayalım. `MLO` ve `u-boot.img` dosyalarını diskin `boot` bölümüne kopyalayalım.

Eğer işleri doğru yürüttüysek U-Boot UART üzerinden bize mesajlar gönderecektir. SD-Kartı takalım, *UART konsol bağlantısını* yapalım ve picocom'dan verileri izleyelim.

Eğer UART konsoldan mesajlar geliyorsa, herhangi bir tuş ile sistemin bootunu durduralım.Şöyle birşeyler görmemiz gerekiyor.

```
U-Boot SPL 2018.01-00557-g313dcd6 (Jun 16 2019 - 05:08:47)
Trying to boot from MMC1
*** Warning - bad CRC, using default environment



U-Boot 2018.01-00557-g313dcd6 (Jun 16 2019 - 05:08:47 -0700)

CPU  : AM335X-GP rev 2.0
Model: TI AM335x BeagleBone Black
DRAM:  512 MiB
NAND:  0 MiB
MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
** Bad device mmc 0 **
Using default environment

<ethaddr> not set. Validating first E-fuse MAC
Net:   cpsw, usb_ether
Hit any key to stop autoboot:  0
=> ◆◆@
```

Mesajlarda görüldüğü üzere öncelikle SPL yükleniyor daha sonra U-Boot yüklenmekte.

Mesajlarda derleme tarih ve saati yazmaktadır. Özellikle geliştirme aşamasında işe yarar bir bilgi olabiliyor.

Eğer U-Boot'un otomatik boot prosesini durdurmamış olsaydık U-Boot dahili ortam değişkenlerine bağlı olarak çeşitli yerlerde (MMC0, SPI, Ethernet vs.) Linux çekirdeği arayacaktı, bulamayacağı için ekrana bir tomar mesaj basacaktı ve en sonunda kendi konsoluna düşecekti.

Bu aşamada yapmamız gereken son iş ise bir adet uEnv.txt hazırlamak ve onu kopyalamak.

16.3 Örnek-1: U-Boot Komut Satırı Değiştirme

İndirdiğimiz U-Boot içerisinde gelen config içinde basit bir değişiklik yapalım, kaydedelim ve bunu patch olarak kaydedelim.

Yapacağımız değişiklik komut satırının başında bulunan => ibaresini değiştirelim. Bu oldukça basit bir değişiklik olup sadece görünüşte olan bir değişikliktir, U-Boot'un çalışmasını etkilemez.

Öncelikle Cross-Compiler tanıtılır ve menuconfig ile menüye girilir.

```
export CC=/opt/workspace/sdk/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin/
↪arm-linux-gnueabihf-
make ARCH=arm menuconfig
```

Menüde aşağıdaki şekilde değişiklik yapılır.

> Command line interface —>(BEAGLE=>) Shell prompt

Tekrar derleyelim ve SD-Karta yükleyeyelim. Ekrana gelen mesajlar şu şekilde olmalı.

```
make ARCH=arm CROSS_COMPILE=$CC -j4
```

```
U-Boot SPL 2018.01-00557-g313dcd6 (Jun 17 2019 - 11:44:06)
Trying to boot from MMC1
*** Warning - bad CRC, using default environment



U-Boot 2018.01-00557-g313dcd6 (Jun 17 2019 - 11:44:06 -0700)

CPU  : AM335X-GP rev 2.0
Model: TI AM335x BeagleBone Black
DRAM:  512 MiB
NAND:  0 MiB
MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
** Bad device mmc 0 **
Using default environment

<ethaddr> not set. Validating first E-fuse MAC
Net:   cpsw, usb_ether
Hit any key to stop autoboot:  0
BEAGLE=>
```

Yaptığımız değişiklik çalışmış gözüküyor. Peki bu değişikliği nasıl kaydedeceğiz ve patch haline getireceğiz?

Öncelikle kaydetmek için aşağıdaki komutu kullanırız.

```
make savedefconfig
```

Bu komut ile üzerinde çalışmış olduğumuz `.config` dosyası `defconfig` dosyası olarak U-Boot ana klasörü altında kaydedilir. `defconfig` dosyasını `configs/` klasörü altında istediğimiz bir isimle kopyalayalım. Genellikle xxx_defconfig formatı takip edilmekte bizde bbb_defconfig diyelim.

```
mv defconfig configs/bbb_defconfig
```

Peki git tarafından yapılan değişikliklere bakalım

```
git status
```

```
mehmet@ubuntu:/opt/workspace/uboot$ git status
On branch beagle_dev
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        configs/bbb_defconfig

nothing added to commit but untracked files present (use "git add" to track)
mehmet@ubuntu:/opt/workspace/uboot$
```

Resimde görüldüğü üzere kopyaladığımız bbb_defconfig dosyası yeni bir dosya olarak gözükmekte ancak `defconfig` dosyası U-Boot git ayarları nedeniyle görmezden gelinmektedir. Bunun nedeni her git reposu içerisinde bir adet `.gitignore` dosyası bulunur ve bu dosya hangi dosya/klasörlerin görmezden gelineceğini bildirir. Bu dosyayı açıp incelemenizi tavsiye ederim.

Bu yeni dosyayı git'e ekleyelim ve ilk commit'imizi yapalım.

```
git add -A
git status
```

```
mehmet@ubuntu:/opt/workspace/uboot$ git add -A
mehmet@ubuntu:/opt/workspace/uboot$ git status
On branch beagle_dev
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   configs/bbb_defconfig

mehmet@ubuntu:/opt/workspace/uboot$
```

Dosyamız artık git akışı içinde yani bu dosyada ileride yapılacak değişiklikler de artık git tarafından takip edilecek.

```
git commit -m "BBB defconfig olusturuldu. Komut satiri degisitirildi."
```

İlk commitimizi yaptık. Commitlerin genelde yeteri kadar açıklayıcı olmasına dikkat edin. Commitlerde genelde İngilizce karakter kullanıyorum, tabii tercih meselesi.

Yaptığımızı kontrol edelim.

```
git log
```

```
commit 35be6b247c80153361b2739aecbf3ab715066a9e
Author: Mehmet <mehmetalinbay@gmail.com>
Date:   Mon Jun 17 12:39:17 2019 -0700

    BBB defconfig olusturuldu. Komut satiri degisitirildi.

commit 313dcd69c2b32648266f91bcf223f9e539bc4201
Author: Andrew F. Davis <afd@ti.com>
Date:   Fri Nov 30 11:38:31 2018 -0600

    ARM: armv7: Add early stack for erratum workarounds

    commit a0106c82d677c611e479a6243f7a08252284e0f0 upstream.

    Some erratum workarounds call into C code before the stack
    is setup, this can lead to values pushed onto the stack
    being lost, firewall exceptions, and other undefined behavior.

    Setup a temporary stack to allow these functions to work
    correctly.

    Signed-off-by: Andrew F. Davis <afd@ti.com>
    Acked-by: Andreas Dannenberg <dannenberg@ti.com>
    Reviewed-by: Nishanth Menon <nm@ti.com>
```

Resimde görüldüğü üzere yaptığımız değişiklik commit olarak geçmişte yapılan değişikliklerden sonra gözükmektedir.

Son olarak yama oluşturalım.

```
git format-patch --binary -o patches/ ti-u-boot-2018.01
```

Yama oluşturmak için birkaç komut var ancak ben bunu tercih ediyorum. Bu komut ile `ti-u-boot-2018.01` noktasından itibaren gelen tüm değişiklikleri (commitleri), yama olarak `patches/` klasörü altına koyacağız. `patches/` klasörü görmezden gelinen klasörler arasındadır bu sayede sonsuz bir döngüden kurtulmuş oluruz.

`git format-patch` komutu her biri commiti ayrı bir patch dosyası olarak saklar.

16.4 Örnek-2: MMC Driver Model Özelliğinin Kapatılması

SD kart içindeki dosyalara U-Boot komut satırından bakmaya çalıştığımda SD kartın bulamadığını gördüm. Buradaki en büyük saçmalık SPL U-Boot'u SD karttan yüklüyordu ancak *nedense* U-Boot SD kartı bulamıyordu.

Sorunun çözümü (https://e2e.ti.com/support/processors/f/791/t/662382) için önerilen ise MMC Driver Model özelliğinin kapatılmasıydı. Bana **2 güne** mal olan bu çözümü uygulayalım.

```
export CC=/opt/workspace/sdk/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin/
↪arm-linux-gnueabihf-
make ARCH=arm menuconfig
```

Menüde `> Device Drivers > MMC Host controller Support` altında bulunan `Enable MMC controllers using Driver Model` seçeneğini kapatalım ve derleme yapalım.

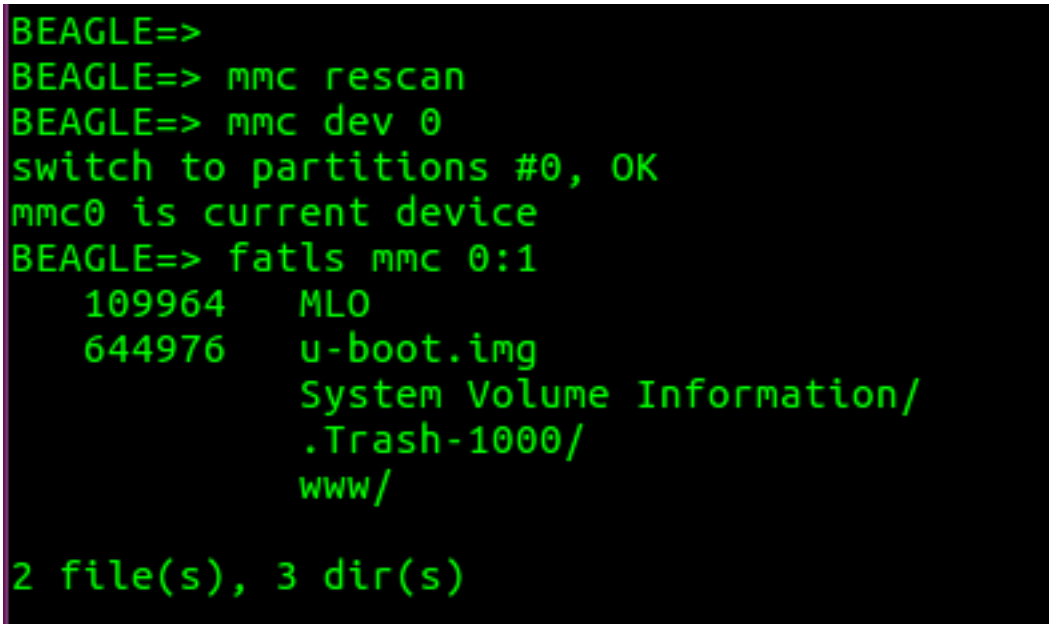
```
make ARCH=arm CROSS_COMPILE=$CC -j4
```

MLO ve u-boot.img dosyalarını SD Karta kopyalayalım, kartı çalıştıralım. U-Boot başladığında ise herhangi bir tuşa basarak konsoldan durduralım.

U-Boot komut satırında sırasıyla şu komutları çalıştıralım.

    mmc rescanmmc dev 0fatls mmc 0:1

U-Boot çıktısı şuna benze birşeyler olmalı.



`mmc rescan` komutu işlemcinin MMC cihazların taraması için kullanılır.`mmc dev 0` komutu ile 0 nolu (SD Kart) MMC cihazına geçilir. 1 nolu cihaz kart üstünde bulunan eMMC'dir. `fatls mmc 0:1` komutu ile 0 nolu MMC cihazın 1. bölümünün içindekiler listelenir. 1. bölüm FAT32, 2.nci bölüm EXT4 olarak formatladığımız bölümdü.

Son olarak bu değişikliği de yama haline getirelim.

```
make savedefconfig
mv defconfig configs/bbb_defconfig
git add -A
git commit -m "SD Kart okuma sorunu nedeniyle MMC Driver Model kapatildi."
git format-patch --binary -o patches/ ti-u-boot-2018.01
```

//TODO patchler nerede saklanacak.

CHAPTER 17

Linux Giriş

Linux içerisinde kullanıcı programları bulunmayan C dilinde yazılmış bir işletim sistemi çekirdeğidir. Linux çekirdeğinin temel görevleri şunlardır:

- Donanım kaynakları yönetmek (işlemci, RAM vs.)
- Kullanıcı programları için donanım bağımsız, taşınabilir API (sistem çağrıları) ve kütüphaneler sağlamak
- Dosya sistemleri için katman ve sürücüler
- Uygulamaların yönetimi

Linux çekirdeği birden çok işlemci mimarisini desteklemektedir. Ölçeklenebilirdir, hatta STM32 mikrokontrolcülerinde bile koşturulabilir.

Linux 1991 yılında Linus Torvalds tarafından "eğlence" amaçlı başlatılmış bir projedir. Bugün kendisininde dahil olduğu binden fazla katılımcı ile geliştirilmektedir. Linux bugün geldiği noktada çok geniş bir kullanım alanına sahiptir: Masaüstü bilgisayarlar, cep telefonları, akıllı TV, süper bilgisayarlar vs. ve tabii ki gömülü sistemler.

Burada sık sık karıştırılan bir kavram olan Linux Dağıtımlarından da bahsetmek istiyorum. Linux dağıtımları; çekirdek ve ek yazılımlarla (ofis, masaüstü yöneticisi, paket yöneticisi vs.) bir araya getirilmesiyle oluşturulmuş tam bir işletim sistemidir. En bilinen dağıtımlar Debian, Ubuntu, Fedora, Pardus vs. olarak sayılabilir.

Gömülü sistemler için de hazır Linux dağıtımlar bulunmakla beraber kaynak israfı gibi nedenlerle bizim tarafımızdan kullanılmayacaktır.

17.1 Device Tree Kavramı

`Device Tree` donanımı tanımlayan özel bir binary dosyasıdır. Device tree kaynak dosyaları (dts) ve include dosyaları (dtsi) C diline benzer sözdizimine sahip bir dosyalardır. Kaynak dosyaları bir derleyici (DTC) ile derlenir ve binary (dtb) dosyaları oluşturulur. Binary dosyaları kernele U-Boot tarafından bildirilir. Kernel bu donanım dosyasına göre kurulum yapar ve çalışır.

Device Tree sistemi özellikle yüzlerce ARM cihazın ortaya çıkmasıyla gereksinim haline gelmiştir.

Device Tree hakkında detaylı bilgi için şu döküman oldukça faydalıdır.

Kernel derleme ile aynı zamanda dtb dosyalarının da derlemesi yapılır.

CHAPTER 18

Kernel Derleme

18.1 Kernel Edinme

U-Boot'ta olduğu gibi kernel içinde iki adet temel kaynağımız vardır: Mainline(ana akım) kernel ve TI kernel. U-Boot konusunda olduğu gibi ben TI'nin sağladığı Linux kerneli kullanacağım.

Mainline kernel kernel.org websitesinden indirilebilir.

Kernel.org'dan git kullanarak indirmek için şu komut kullanılır:

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
```

Git kullanmadan `tarball` olarak indirmek için https://kernel.org/pub/linux/kernel websitesinden istenilen versiyon indirilebilir.

TI kernel indirmek için iki yol vardır:

1. TI Processor SDK (Toolchain kurma bölümünde anlatıldı)
2. git.ti.com 'dan indirme

TI GIT üzerinden indirerek işlemlerimizi yapalım.

```
cd /opt/workspace
git clone git://git.ti.com/ti-linux-kernel/ti-linux-kernel.git linux
cd linux/
```

GIT ile indirdiğimiz için birçok branch bulunacaktır. Önce branchlere bakalım

```
git branch -a
```

Biz çalışmalarımızı **Processor SDK 5.03'e** paralellik göstermesi için **ti-linux-4.14.y** branch ile yapalım.

```
git checkout ti-linux-4.14.y
```

Bu noktada işlerin karışmaması için kendi branchimizi oluşturmamız gerekiyor. Branch ismi **beagle_dev** olsun.

```
git checkout -b beagle_dev
```

Yaptığımız kontrol edelim.

```
git status
```

On branch beagle_dev nothing to commit, working directory clean

Artık Linux üzerinde yapacağımız tüm değişiklikler git tarafından takip edilecek.

18.2 Kernel Derleme

U-Boot'ta olduğu gibi derleme için bize bir adet çalışan `config` dosyası lazım. ARM sistemler için `config` dosyaları Linux içerisinde `arch/arm/configs/` altında bulunur. Bu klasörün içinde çeşitli kartlar için `config` dosyaları bulunmaktadır, aşağıdaki komut ile mevcut config dosyaları listelenir.

```
ls /opt/workspace/linux/arch/arm/configs
```

BBB kartlar için `omap2plus_defconfig` dosyası kullanılacaktır. İndirdiğimiz SDK içerisinde ayrıca config dosyaları da var ancak şimdilik `omap2plus_defconfig` ile ilerleyelim.

`omap2plus_defconfig` dosyası aynı zamanda bazı DTS dosyalarını da derleyecektir. Bize lazım olan am335x-boneblack.dts dosyasıdır ve `omap2plus_defconfig` kapsamında derlenmektedir. Hangi dosyaların derleneceğinin belirlenmesi, kendi DTS dosyamızı sisteme tanıtmak gibi konulara ileride değerlendireceğiz. Device Tree dosyaları `arch/arm/boot/dts` klasörü altındadır, aşağıdaki komut ile mevcut DTS dosyaları listelenir.

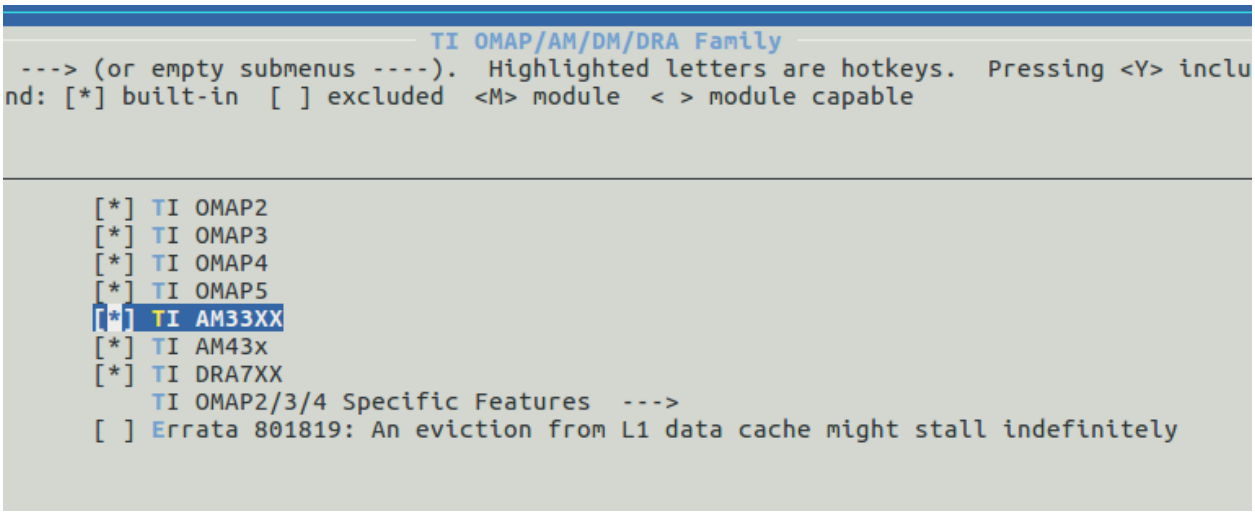
```
ls /opt/workspace/linux/arch/arm/boot/dts
```

Şimdi `omap2plus_defconfig` dosyasına göre derleme yapalım.

```
cd /opt/workspace/linux
make ARCH=arm omap2plus_defconfig
```

Isterseniz şu aşamada menuconfig ile konfigürasyona bakabilirsiniz. İşlemci seçiminin doğru olduğunu görmek için menüden System Type / TI OMAP/AM/DM/DRA Family altındaki seçeneklere bakabilirsiniz.

```
make ARCH=arm menuconfig
```



Derleme yapalım. İlk yapacağımız derleme de `-j` ile parametre vermeden tek çekirdek ile derleme yapmak daha iyi olacaktır. Bu sayede hata ayıklama işi daha kolay olmaktadır.

```
export CC=/opt/workspace/sdk/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin/
↪arm-linux-gnueabihf-
make ARCH=arm CROSS_COMPILE=$CC -j4
```

Toolchain olarak Linaro'dan indirdiğimiz toolchain'i kullandık ve `CROSS_COMPILE=$CC` ile toolchain'i bildiriyoruz.

`ARCH=arm` parametresi ile derlememizi ARM çekirdek ile yapacağımızı bildiriyoruz.

`-j4` parametresi ile derleme için kaç çekirdek kullanılacağını bildiriyoruz. Eğer ilk defa derleme yapıyorsanız bu parametreyi kullanmayın. Bu sayede derleme hatalarını görmemiz daha kolay olacaktır.

Derlememiz başarılı ise derlenmiş kernel **zImage** `arch/arm/boot` klasörü altında, **am335x-boneblack.dtb** dosyası `arch/arm/boot/dts` klasörü altında olacaktır.

Hazırladığımız SD-Karta bu iki dosyayı kopyalayalım. U-Boot başladığında ise herhangi bir tuşa basarak konsoldan durduralım.

U-Boot konsoldan aşağıdaki komutları girelim.

> mmc rescanmmc dev 0setenv kerneladdr 0x82000000setenv dtbaddr 0x88000000fatload mmc 0:1 ${dtbaddr} am335x-boneblack.dtbfatload mmc 0:1 ${kerneladdr} zImage setenv bootargs console=ttyO0,115200n8 root=/dev/mmcblk0p1 rw rootfstype=ext4 rootwaitbootz ${kerneladdr} - ${dtbaddr}

Yukarıdaki komutlara baktığımızda MMC cihazdan *setenv* tanımladığımız `kerneladdr` ve `dtbaddr` RAM adreslerine sırasıyla kernel ve device tree dosyası yüklenir. `bootargs` U-Boot ortamı için özel bir değişkendir, kernele boot esnasında verilecek parametreleri tutar. Verdiğimiz parametreler sırasıyla kernel konsolunun 115200 8n1 formatında ttyO0 (UART0) olacağı, rootfs'in /dev/mmcblk0p1 altında ext4 formatında ve yazılabilir olduğunu bildiriyoruz. En son satırda bootz komutuyla `kerneladdr` ve `dtbaddr` RAM adreslerinden makinanın boot prosesini başlatıyoruz.

Bu aşamada kernel boot işlemi başlayacak ancak rootfs olmadığı için işlem yarım kalacaktır.Eğer buraya kadar sorunsuz geldiysek işler gayet iyi gitmiş demektir.

Tabii her boot işleminde komutları tekrar tekrar girmeyeceğiz, bu komutları uEnv.txt dosyasında saklayacağız.

Örnek uEnv.txt dosyasını buradan indirebilirsiniz. Yaptığımız basitçe; yukarıda yazmış olduğumuz komutları özel bir U-Boot ortam değişkeni olan `uenvcmd` içine kaydetmek. U-Boot bu değişkeni yürüteceği için sırasıyla bizim komutları script gibi çalıştıracak. Tabii rootfs olmadığı için "Kernel panic" mesajıyla boot prosesi bitecek.

```
U-Boot 2018.01-00558-g35be6b2 (Jul 13 2019 - 05:34:55 -0700)

CPU  : AM335X-GP rev 2.0
Model: TI AM335x BeagleBone Black
DRAM:  512 MiB
NAND:  0 MiB
MMC:   OMAP SD/MMC: 0, OMAP SD/MMC: 1
*** Warning - bad CRC, using default environment

<ethaddr> not set. Validating first E-fuse MAC
Net:   cpsw, usb_ether
Hit any key to stop autoboot:  0
switch to partitions #0, OK
mmc0 is current device
SD/MMC found on device 0
** Unable to read file boot.scr **
330 bytes read in 2 ms (161.1 KiB/s)
Loaded env from uEnv.txt
Importing environment from mmc0 ...
Running uenvcmd ...
Boot Prosesi Basliyor...
switch to partitions #0, OK
mmc0 is current device
36841 bytes read in 5 ms (7 MiB/s)
4108064 bytes read in 295 ms (13.3 MiB/s)
## Flattened Device Tree blob at 88000000
   Booting using the fdt blob at 0x88000000
   Loading Device Tree to 8fff4000, end 8fffffe8 ... OK

Starting kernel ...
```

Bu aşamadan sonra ilk rootfs'mizi oluşturalım ve bir adet sistemi tamamlayalım.

CHAPTER 19

Kernel Örnekler

19.1 Örnek: Kernel Modül Kullanımı

Not: Bu örneği yürütebilmeniz için RootFS'in oluşturulmuş ve SSH üzerinden bağlanmanız gereklidir.

Kernel modülleri isteğe bağlı olarak çalışma durumunda yüklenip/çıkartılabilen kod parçalarıdır. Bu sayede sistemi yeniden başlatma gereği olmadan sistemin işlevselliği artırılabilir (kaynak).

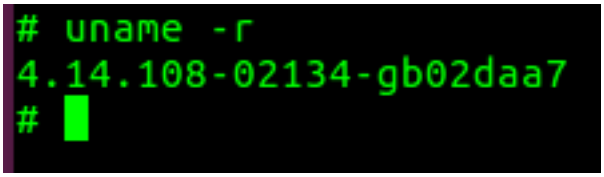
Kernel konfigürasyonunda bir girdinin yanında `M` yazıyorsa bu girdi modül olarak derlenecek demektir. Örnek olarak kernel derlemede kullandığımız `omap2plus_defconfig` içerisinde **435** (!) kernel modülü vardır. Konfigürasyon dosyanın içerisine bakmanızı tavsiye ederim.

Kernel modüllerini derleyelim:

```
export CC=/opt/workspace/sdk/gcc-linaro-7.3.1-2018.05-x86_64_arm-linux-gnueabihf/bin/
↪arm-linux-gnueabihf-
cd /opt/workspace/linux
make ARCH=arm CROSS_COMPILE=$CC -j4 modules_install INSTALL_MOD_PATH=out
```

Yukarıdaki komutlar sonrasında modüller *out* klasörüne kopyalanacaktır.

Kernel modülleri RootFS içerisinde `/lib/modules/kernel_release` içerisinde bulunurlar. *kernel_release* değişkendir bu sayede kernel kendisine ait modülleri yükler(Bu durum sahadaki cihazın kernelini güncellediğinizde modüllerin yüklenmemesi sorununa da yol açabilir). Mevcut kernelin dağıtım ismini `uname -r` komutu ile öğrenebiliriz.



Modülleri oluşturduğumuz out klasörünün içerisindeki klasör ağacı /lib/modules/4.14.108-02134-gb02daa7/ şeklindedir. Tüm bu modülleri sistemimize kopyalayalım.

```
cd out/lib/
tar cf modules.tar *
scp modules.tar root@192.168.2.100:/lib/
```

Kopyaladığımız modules.tar dosyasını boardda açalım.

```
cd /lib/
tar xf modules.tar
sync
```

Modülleri yüklemek için iki adet komut vardır: `insmod` ve `modprobe`. insmod ile modül dosyasının tam yolunu vermeniz gerekir. modprobe komutu ise `/lib/modules/$(uname -r)/modules.dep.bin` dosyasından modülleri okur ve eğer modüle ve bağlı olduğu modüller varsa onları yükler, kısaca insmod'dan daha yetenekli bir komuttur.

Burada dikkat edilmesi gereken modprobe, kernel versiyonu ile birebir aynı isimdeki klasörün altındaki modüllere bakar. 4.14.79-gvea klasörü altındaki modüller kernel 4.14.79-aabbcc tarafından modprobe ile **yüklenemeyecektir.**

Board'a kopyaladığımız modülleri listelemek için aşağıdaki komut koşturulur.

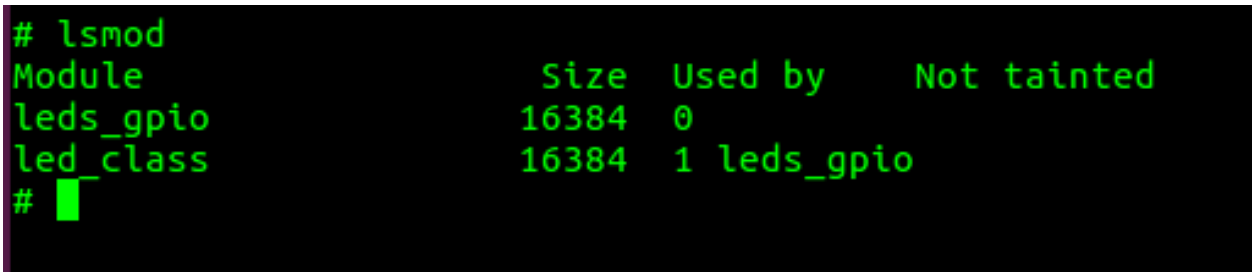
```
modprobe -l
```

Board üstünde `leds-gpio.ko` modülünü yükleyelim.

```
modprobe leds-gpio
```

Modülü yükleyince ethernet konnektörü tarafındaki LEDlerden D4'ün seyrek olarak yanmaya başladığını görebiliriz.

`lsmod` ile yüklenmiş olan modüllere bakalım.



Görüldüğü üzere biz `leds-gpio.ko` modülünü yükleme komutunu vermiştik ancak modprobe bu modülün ihtiyaç duyduğu `led_class` modülünü de yüklemiştir.

Son olarak ek bir modül daha yükleyelim.

```
modprobe ledtrig-heartbeat
```

`ledtrig-heartbeat` modülünü yüklediğimizde ise D2 LEDinin de yanmaya başladığını göreceksiniz. Heartbeat LEDi Linux çekirdeğinin çalıştığını gösteren bir modüldür.

Son olarak `ledtrig-heartbeat` modülünü kaldıralım ve D2 LEDini kapalı konuma alalım.

```
modprobe -r ledtrig-heartbeat
```

CHAPTER 20

RootFS Nedir?

Unix-benzer işletim sistemlerinde tüm dosyaların olduğu dosya sistemine kök dosya sistemi (rootfs) denilir. MS-DOS tabanlı işletim sistemlerinde tek bir kök-dosya sistemi yoktur, her bir depolama birimi veya bölümünün kendine ait kök klasörlere ayrılmıştır. Gözlemime göre bu fark Windows tabanlı kullanıcıların geçiş aşamasında yaşadığı en büyük uyum sorunlarından birisidir.

RootFS (kök-dosya sistemi) / ile gösterilir ve en üst kademedir. Bu seviyenin altında standart olarak kabul edebileceğimiz klasörler bulunmaktadır. (/bin , /dev , /mnt , /home gibi)

RootFS içerisinde çalışma anında kullanacağımız programlar, açılış scriptleri, kütüphaneler vs. belli bir hiyerarşide bulunmalıdır. Bir RootFS sistemini yüzlerce belki binlerce dosya oluşturabilir.

Örneğin sisteminizde bir adet rar dosya açıcı ve müzik programı olduğunu ve bu iki programın kullanması gereken ortak bir kütüphanenin olduğunu düşünelim. Bu durumda iki programın kütüphanenin farklı versiyonları ile çalışması/çalışmaması gibi uyumluluk sorunları ortaya çıkabilir.

Bu nedenlerle RootFS oluşturmak oldukça karmaşık bir işlem haline gelebilmektedir.

Kaynak

20.1 RootFS Oluşturma Yöntemleri

RootFS oluşturmak için temelde 3 adet yöntem vardır:

1. Manuel Oluşturulmuş Dağıtımlar
2. Hazır Dağıtımlar
3. Derleme Tabanlı Dağıtımlar

20.1.1 Manuel Yöntem

Manuel yöntem işin inceliklerini öğrenmek için en iyi yöntemdir. Bilgisayarınızda boş bir klasör açarak adım adım içerisinde ilerleyebilirsiniz. Qemu emülatörü ile de hızlı denemeler yapabilirsiniz. Bu konuda Türkçe kaynak olarak şu kaynakları öneririm:

1. http://www.ucanlinux.com/eski-blog-yaz%C4%B1lar%C4%B1
2. Gömülü Linux Sistemleri Kitabı
3. https://demirten.gitbooks.io/gomulu-linux/content/

Şuan için bu kaynakta Manuel dosya sistemi oluşturma konusu incelenmeyecektir.

20.1.2 Hazır Dağıtımlar

ARM tabanlı işlemciler için Ubuntu gibi bazı hazır dağıtımlar mevcuttur. Örneğin Debian periyodik güncellenen dağıtımlar yayınlamaktadır. Bu dağıtımların avantajı ARM için hazır bulunan repoları sayesinde program bulma/derleme/uyumluluk sorunları giderme işlerinden sizi kurtarmaktadır. Bazı geliştirme kart üreticileri kendi hazır dağıtımlarını (genellikle Debian'dan türetilmiş) yayınlamaktadır. Ancak bu sistemler genellikle oldukça şişkindir ve kaynak kullanımını yüksektir. Bu nedenle gömülü sistemler için pek ideal değildir.

20.1.3 Derleme Tabanlı Dağıtımlar

Derleme tabanlı sistemler ise belirlediğimiz yönergeler çerçevesinde bizim için RootFS oluştururlar. Bu sayede oldukça küçük veya karmaşık sistemler oluşturabiliriz.

Derleme tabanlı sistemler temelde bize kendi ekosistemlerinde bulunan paketleri sunarlar. Eğer bir paketin sistemimizde olmasını istiyorsak onu aktif ederiz ve seçtiğimiz paket internetten indirilir, işlemcimize yönelik derlemesini yapılır sistemin oluşturduğu RootFS içerisine gerekli yan dosyaları ile birlikte kopyalanır. Derleyici sistem ilgili paketin sorun çıkarmayacak versiyonu ile çalışır bu sayede paketler arası uyumluluk gözetilmiş olur.

Gömlülü Linux için en popüler iki adet sistem şunlardır: Yocto ve Buildroot

Yocto Projesi

Yocto Linux Foundatation başta olmak üzere Texas Instruments, NXP gibi büyük oyuncularında destek verdiği gömülü linux sistemleri üretmeyi amaçlayan açık kaynak kod bir projedir.

Yocto ile gömülü linux sistemindeki tüm parçalar (bootloader, kernel, rootfs) üretilebilir, konfigüre edilebilir.

Ancak Yocto projesi *nedense* oldukça hantaldır ayrıca konfigüre etmek için *nedense* bir arayüzü yoktur, üstelik karmaşık diyebileceğimiz bir yapısı vardır. Hantallığına örnek vermek için Beaglebone-Black için kurmak istediğim en basit sistemin derlenmesi i7+SSD diske sahip masaüstü bir makina da yaklaşık 4 saat sürmüştür.

Bu nedenlerle biraz Yocto projesine mesafeliyim.

Buildroot Projesi

Buildroot için kısaca Yocto'nun basitleştirilmiş hali diyebiliriz. Elbette bu iki sistemin işleyişleri birbirinden oldukça farklıdır. Buildroot açık kaynak bir projedir, topluluk tabanlı destek ile yürütülmektedir.

Buildroot projesinin başında Peter Korsgaard vardır ancak bu aşamada Bootlin firmasından bahsetmek gerekir. Bootlin Linux üzerine eğitim/danışmanlık veren ve Linux dünyasında hatırı sayılır geliştirmede bulunan bir firmadır ve websitelerinde oldukça faydalı eğitim dökümanları bulunmaktadır.

Buildroot Yocto gibi gömülü linux sistemindeki tüm parçaları (bootloader, kernel, rootfs) üretilebilir, konfigüre edilebilir.

Buildroot konfigürasyon için kernel ve u-boot kullanımında alıştığımız `menuconfig` yapısını kullanır. Bu sayede oldukça basit bir yönetim sunmaktadır.

Yocto ile hız açısından karşılaştırmak istersek Beaglebone-Black için tüm derleme 1 saatin altında olabilmektedir.

Bundan sonraki aşamalarda Buildroot ile RootFS oluşturma anlatılacaktır.

CHAPTER 21

Buildroot Giriş

21.1 Buildroot Edinme

Buildroot'un **2019.02** versiyonunu kullanacağız. Bunun için ya GIT ile indirip kendi branchimizi yada direk olarak **2019.02** paketini (tarball) indireceğiz. Bu kitapta buildroot için tarball üzerinden yürüyeceğiz.

GIT

```
cd /opt/workspace
git clone git://git.busybox.net/buildroot
cd buildroot
git checkout -b beagle_dev 2019.02
```

TARBALL

```
cd /opt/workspace
wget https://buildroot.org/downloads/buildroot-2019.02.5.tar.bz2
tar xf buildroot-2019.02.5.tar.bz2
mv buildroot-2019.02.5 buildroot
cd buildroot
```

21.2 Buildroot ile Basit Bir RootFS Oluşturma

Buildroot içerisinde kernel ve u-boot'ta olduğu gibi hazır defconfig dosyaları bulunmkatadır. Kullandığımız Beaglebone-Black içinde hazır bir config dosyası (beaglebone_defconfig) vardır ancak Buildroot ile sıfırdan başlamak kolay olduğu için biz hazır gelen defconfig yerine kendi config dosyamızı oluşturacağız.

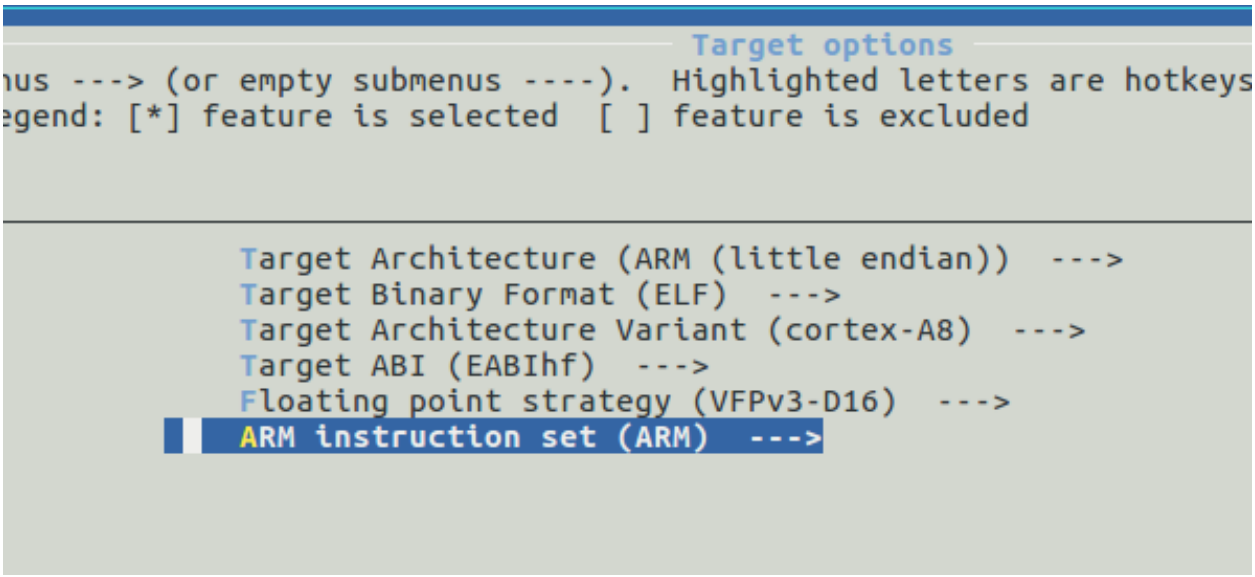
Öncelikle basit bir konfigürasyon yapalım:

```
cd buildroot
make menuconfig
```

`Target Options` bölümünden

- Mimariyi **ARM (little endian)**
- Mimariyi varyantını **cortex-A8**

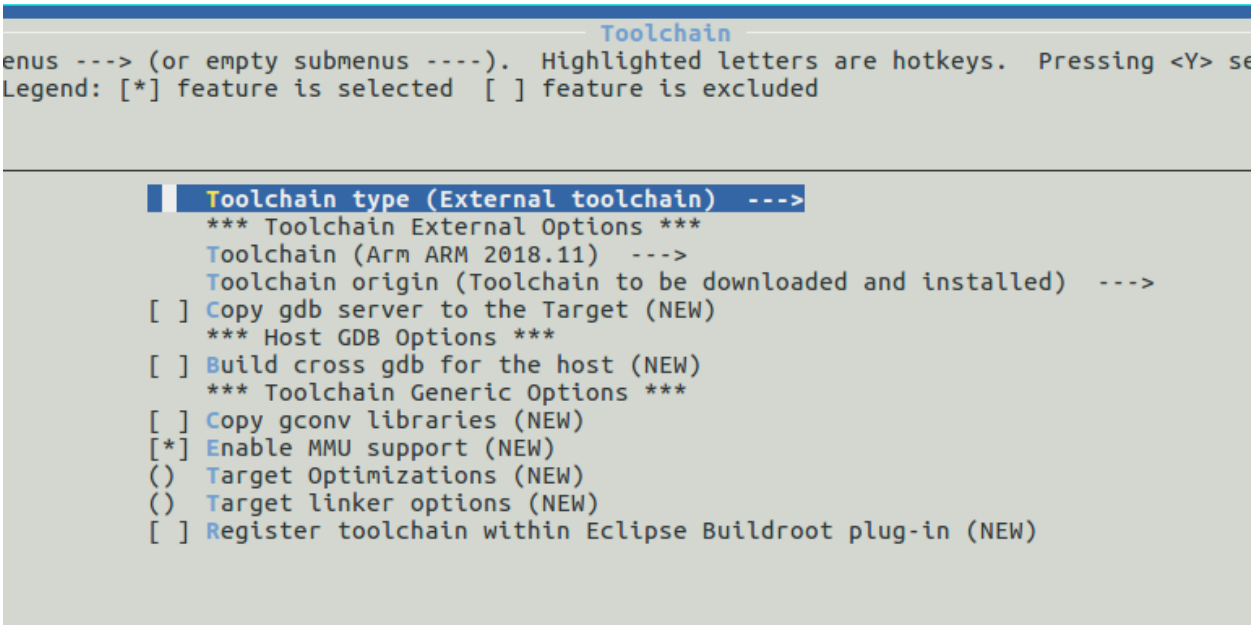
olarak seçelim. Bu verileri AM335x datasheetinden elde ettik.



Bir üst menüye dönelim ve `Toolchain` bölümünden

- Toolchain tipini **External Toolchain**

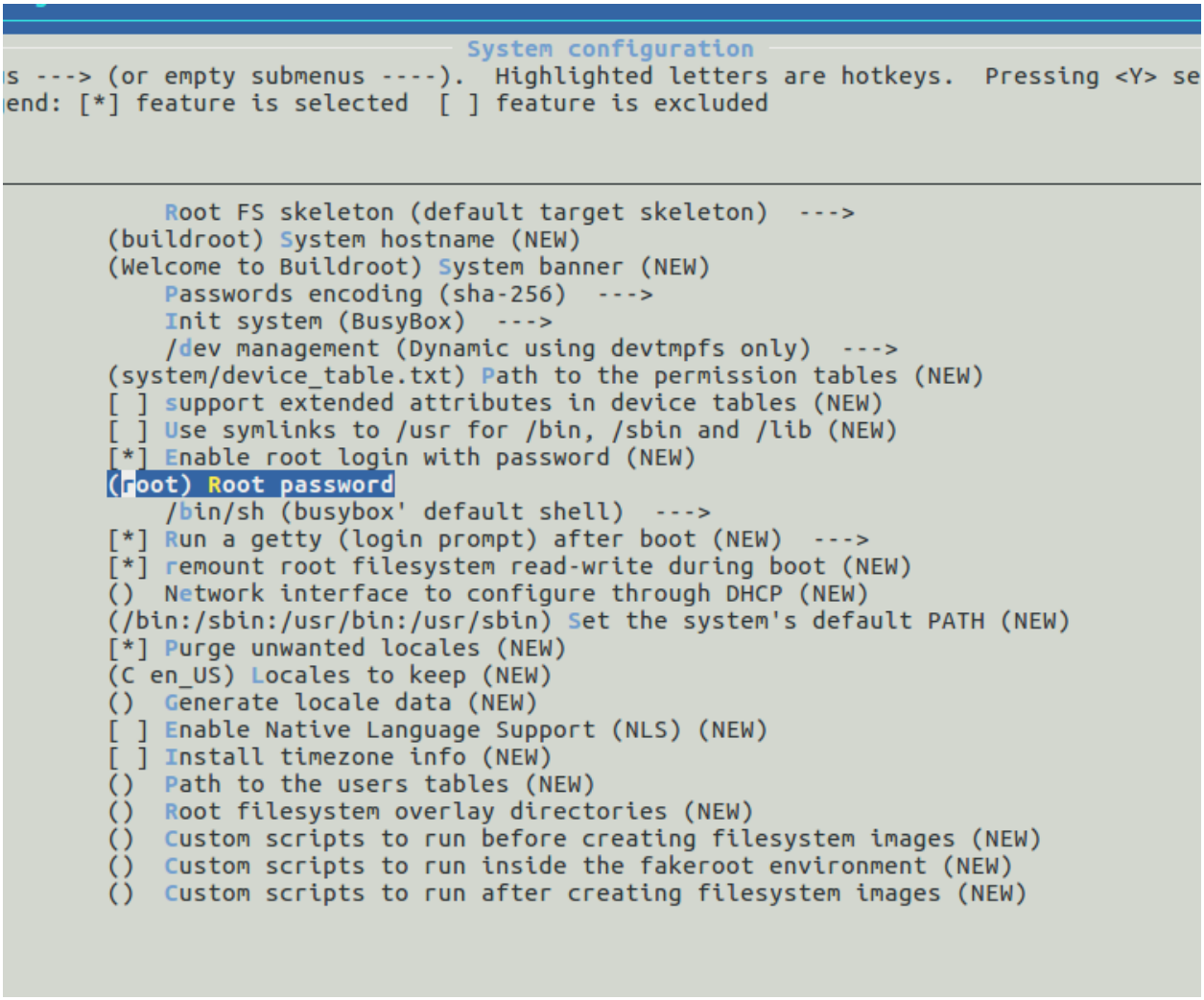
olarak seçelim. Harici toolchain olarak Linaro 2018.11 toolchain kullanılıyor. buildroot'un kendi toolchaini ben hiç kullanmadım ve oldukça zaman aldığı kaynaklarda söyleniyor.



Bir üst menüye dönelim ve `System configuration` bölümünden

- Root şifresini **root**

olarak ayarlayalım.



Ana menüde görebileceğiniz üzere **Kernel** ve **Bootloader** bölümleri adlarından da kolayca anlaşılacağı gibi Buildroot aracılığı ile kernel ve u-boot derlemek için kullanılırlar.

Target packages bölümü sistem üzerindeki varolacak paketleri belirlemede kullanılır. Buildroot içinde yaklaşık olarak 2500 paket bulunmaktadır. Bu menü içerisinde gezmenizi tavsiye ederim.

Son olarak ana menüden `Filesystem images` bölümüne göz atalım. Bu menüde oluşturacağımız RootFS'in tipini belirliyoruz. Şuan sizde sistem tar formatında çıktı üretecektir.

Peki şimdi kaydedip çıkalım ve ilk derlememizi alalım.

```
make
```

Ekranda bir süre Buildroot'un yaptığı işlemler kayacak ve en sonunda ilk RootFS imajımız oluşacak.

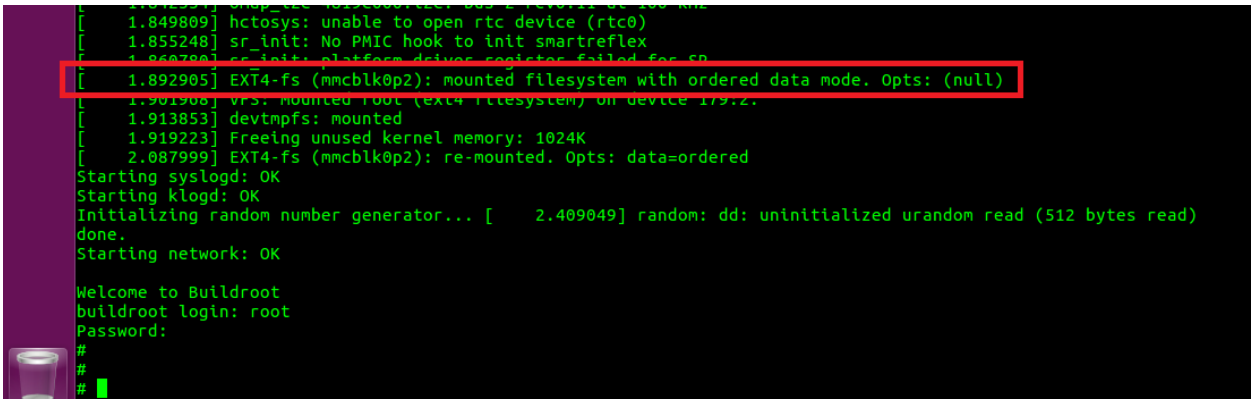
Buildroot'un oluşturduğu çıktılar `output/images` klasörü altındadır. Şuan için sadece **rootfs.tar** bu klasörün altında olmalıdır.

```
ls output/images/
```

SD-Kartımızı takalım ve oluşturduğumuz RootFS imajını atalım.

```
sudo tar xvf output/images/rootfs.tar -C /media/$USER/rootfs
sync
```

Kartı boot ettiğimiz zaman U-Boot kerneli yükleyecek, kernel ise rootfs'e bağlanarak login aşamasına gelecektir.



Tebrikler :) İlk defa tüm parçaları ile bir Gömülü Linux Sistemini oluşturdunuz.

Şimdi Buildroot'ta hazırladığımız config dosyasını kaydedelim.

```
make savedefconfig
mv defconfig configs/beagle_basic_defconfig
```

CHAPTER 22

Buildroot Örnek: SSH Server Eklemek

İlk yaptığımız derlemeye kartımıza SSH bağlantısını sağlamak için gerekli olan Dropbear paketini ekleyelim.

```
cd /opt/workspace/buildroot
make menuconfig
```

`Target packages` / `Networking applications` bölümünden `dropbear` seçeneğini işaretleyip derleme yapalım.

Bu noktada dikkatinizi çekmenizi istediğim nokta `dropbear` paketinin ihtiyaç duyduğu paket otomatik olarak seçilir ve derleme sırasında bu ek paket(ler) kurulur.

```
BR2_PACKAGE_DROPBEAR:

A small SSH 2 server designed for small memory environments.

Note that dropbear requires a per-device unique host key. The
key will be generated when dropbear starts, but it is not
persistent over reboot (if you have a read-only rootfs) or
upgrade (if you have a read-write rootfs). To make the key
persistent, replace /etc/dropbear with a symlink to a
directory on a persistent, writeable filesystem.
Alternatively, mount a persistent unionfs over your root
filesystem.

https://matt.ucc.asn.au/dropbear/dropbear.html

Symbol: BR2_PACKAGE_DROPBEAR [=y]
Type  : bool
Prompt: dropbear
  Location:
    -> Target packages
      -> Networking applications
  Defined at package/dropbear/Config.in:1
  Selects: BR2_PACKAGE_ZLIB [=y] && BR2_PACKAGE_LIBTOMCRYPT [=n]
```

```
make
```

Yeni RootFS'i SD-Karta yükleyelim. Açılış loglarında dropbear'in başladığını göreceğiz.

```
Starting syslogd: OK
Starting klogd: OK
Initializing random number generator... [    2.416019] random: dd: uninitialized urandom read (512 bytes read)
done.
Starting network: OK
Starting dropbear sshd: [    2.679076] random: dropbear: uninitialized urandom read (32 bytes read)
OK

Welcome to Buildroot
buildroot login: root
Password:
#
```

PC ile Beaglebone-Black arasında ethernet kablosu ile bağlantı kuralım. Daha sonra PC ethernet portuna sabit bir IP (örn: 192.168.2.35) verelim (Windows'ta denetim masasından IP vermeniz sanal makina için yetiyor). Kartımıza da aynı subnette sabit bir IP vermemiz gerekiyor, komut satırından şu şekilde halledebiliriz:

```
ifconfig -a
ifconfig eth0 192.168.2.100
```

```
# ifconfig eth0 192.168.2.100
[  937.371332] net eth0: initializing cpsw version 1.12 (0)
[  937.476379] SMSC LAN8710/LAN8720 4a101000.mdio:00: attached PHY driver [SMSC LAN8710/LAN8720] (mii_bus:phy_addr=4a101000.mdio:00, irq=POLL)
[  937.500767] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
# [  940.646401] cpsw 4a100000.ethernet eth0: Link is Up - 100Mbps/Full - flow control rx/tx
[  940.655402] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
```

Bu durumda kartımızın **eth0** portunu sabit bir IP vererek ayağa kaldırdık. Yaptıklarımızı doğrulamak için sanal makinanın komut satırından karta ping atalım.

```
ping 192.168.2.100
```

Eğer tüm ayarlar doğru ise ekranda ping mesajlarını göreceksiniz. SSH bağlantısını *root* kullanıcı olarak kurmak için aşağıdaki komut bilgisayardan yürütülür.

```
ssh root@192.168.2.100
```

İlk defa SSH bağlantısı kurulacağı için bilgisayarınız bu makinayı listeye eklemeyi size soracak, onaylamanız durumunda şifre bölümüne geçeceksiniz.

```
mehmet@ubuntu:/opt/workspace/buildroot$ ssh root@192.168.2.100
The authenticity of host '192.168.2.100 (192.168.2.100)' can't be established.
ECDSA key fingerprint is SHA256:7zR0L7eJDJo7PNiub7r4ilMGvMN7kbU+7JRe+/I57pM.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.2.100' (ECDSA) to the list of known hosts.
root@192.168.2.100's password:
#
#
#
```

Böylelikle ilk defa Buildroot kullanarak yeni bir paket sisteminize eklemiş olduk. Son olarak config dosyamızı kaydedelim, ek olarak içine göz atalım. OPENSSL ve DROPBEAR paketleri de yeni config dosyamızda gözüküyor olmalı.

```
make savedefconfig
mv defconfig configs/beagle_basic_defconfig
cat configs/beagle_basic_defconfig
```

CHAPTER 23

Buildroot Örnek: RootFS Overlay Özelliği

SSH bağlantısı için Dropbear paketini eklememiz tek başına yeterli olmadı, komut satırından kartımıza sabit bir IP adresi de vermemiz gerekti. Ancak sistemi yeniden başlattığımızda kartımıza verdiğimiz IP adresi kaybolacaktır. Her açılışta kartımızın sabit bir IP ile başlamasını istiyorsak `/etc/network/interfaces` dosyasına müdahale etmemiz gerekmektedir.

Sistemimizde bulunan vi editorü ile interfaces dosyasına aşağıdaki satırları ekleyelim.(vi denen eziyet ile tanışmadan olmaz!)

```
auto eth0
iface eth0 inet static
        address 192.168.2.100
        netmask 255.255.255.0
```

Dosyayı kaydedelim ve kartı yeniden başlatalım. Kartımız tekrar ayağa kalktığında `/etc/network/interfaces` içinde belirttiğimiz 192.168.2.100 IP adresini otomatik alarak başlayacaktır. İşlemleri kontrol etmek için SSH bağlantısını tekrar kurabilirsiniz.

Ancak bu durum yeni bir sorunu ortaya çıkarmaktadır. Üreteceğimiz her cihaza tek tek müdahale etmemiz gerekmektedir.

Buildroot'ta yaptığımız rootfs derlemesinin içerisine belirlediğimiz dosyaları kopyalayan bir mekanizma vardır: `rootfs overlay`

Buildroot'ta overlay yapabilmemiz için menüden bir klasörü göstereceğiz. Bu klasör derleme sonrası olduğu gibi üretilen rootfs'in üzerine kopyalanır. Bu nedenle kopyalanmasını istediğimiz dosyalar rootfs'te bulunacağı klasör ağacında olmalıdır.

Buildroot overlay klasörü gibi projeye özel dosya/klasörlerin `board` klasörü altında olmasını önerir. Bizde `board` klasörü altında kendimize bir klasör açalım. Mevcutta bulunan `board` altındaki diğer proje klasörlerini incelemenizi öneririm.

```
cd /opt/workspace/buildroot
mkdir board/beagle_edu
```

`beagle_edu` klasörü altında ileride başka dosyalarda bulunacaktır bu nedenle rootfs_overlay klasörünü oluşturup onun altında çalışacağız.

```
cd board/beagle_edu
mkdir rootfs_overlay
mkdir etc/
mkdir etc/network
nano etc/network/interfaces
```

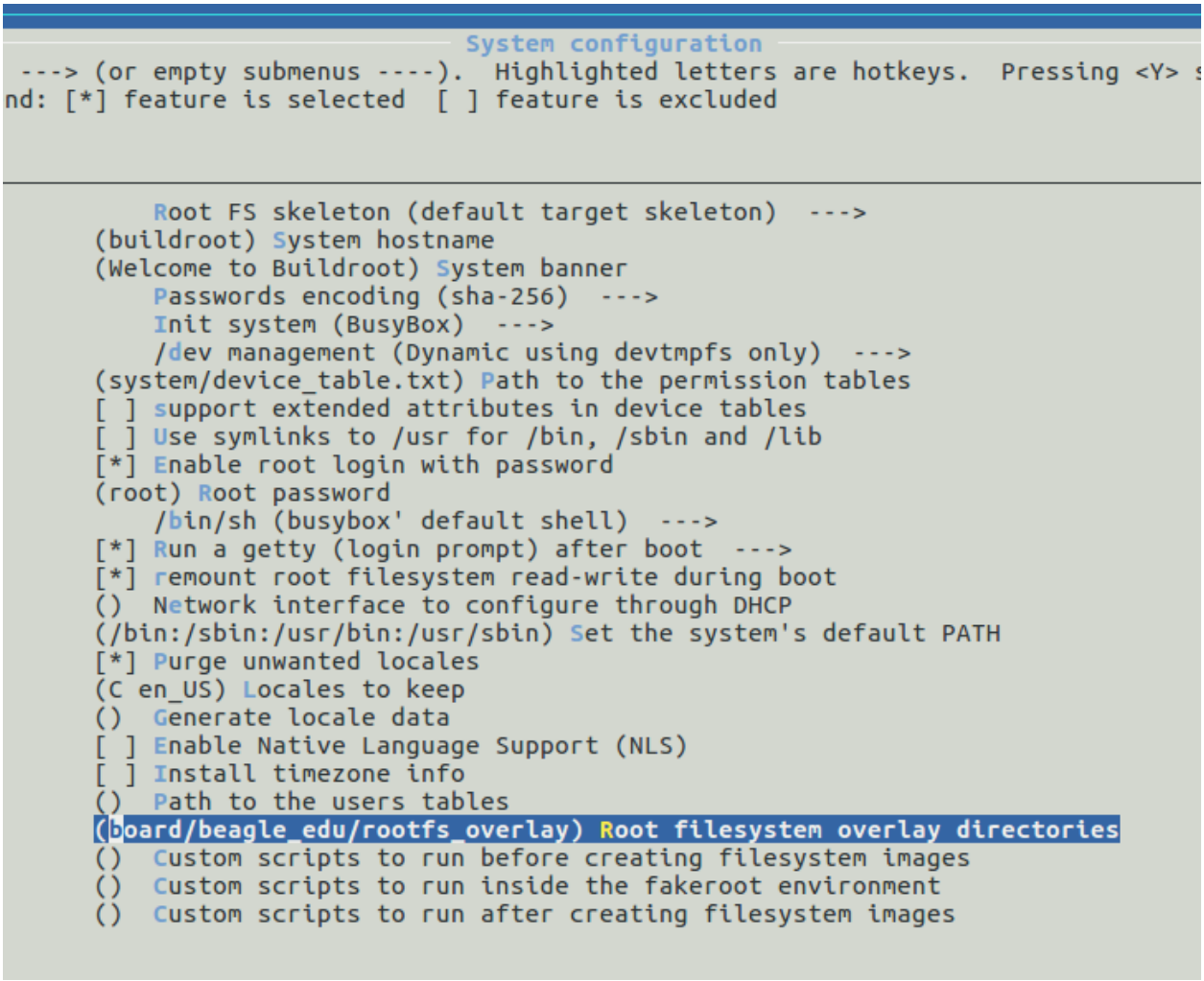
Dosya içeriği şu şekildedir:

```
# interface file auto-generated by buildroot

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
        address 192.168.2.100
        netmask 255.255.255.0
```

Overlay klasörünü buildroot'a tanıtalım: Derleme menüsünden `System Configuration` altındaki overlay bölümüne `board/beagle_edu/rootfs_overlay` yazalım.



Temiz bir build alalım.

```
make clean
make
```

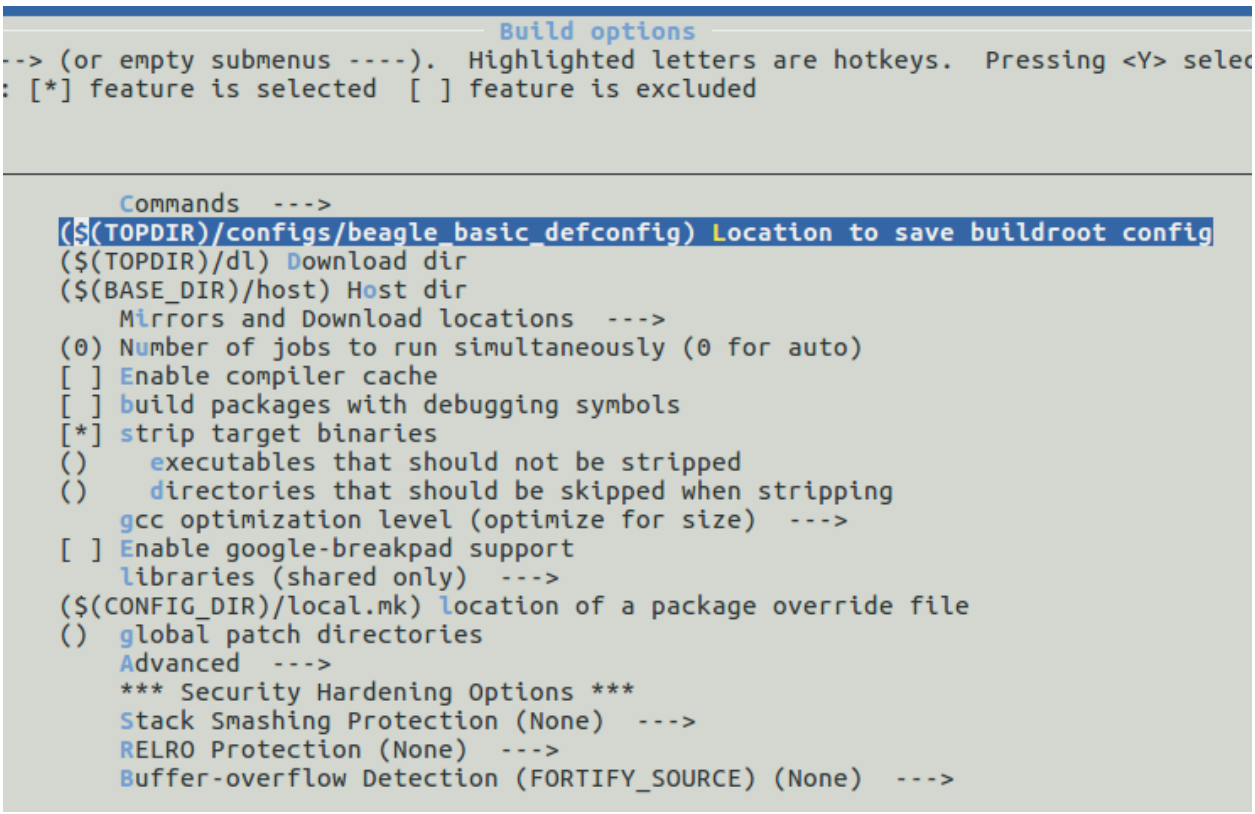
Derlemeyi hızlıca kontrol etmek için output klasörü altındaki target klasörüne bakabiliriz. Son olarak SD-Kartımıza yeni imajı yükleyelim ve kontrol edelim.

```
cat output/target/etc/network/interfaces
```

23.1 Defconfig Dosyasının Kaydedilme Yerinin Ayarlanması

Bu ana kadar yaptığımız örneklerde defconfig dosyasını `make savedefconfig` komutu ile buildroot ana klasörü içine kaydediliyor ve elle `config` klasörü altına taşınıyordu. Linux ve U-Boot'ta bu hedef dosya değiştirilemezken Buildroot'ta değiştirilebilir.

Buildroot menüsünde `Build options` altında bulunan config dosyasının kaydedileceği yer `$(TOPDIR)/configs/beagle_basic_defconfig` olarak ayarlanır.



Bundan sonra yapacağımız kaydetmeler belirttiğimiz dosyanın içerisine olacaktır.

Son yaptığımız değişiklikleri de kaydedelim ve bu bölümü sonlandıralım.

```
make savedefconfig
cat configs/beagle_basic_defconfig
```

CHAPTER 24

Teşekkürler

- SpeedyX
- OptimusPrime