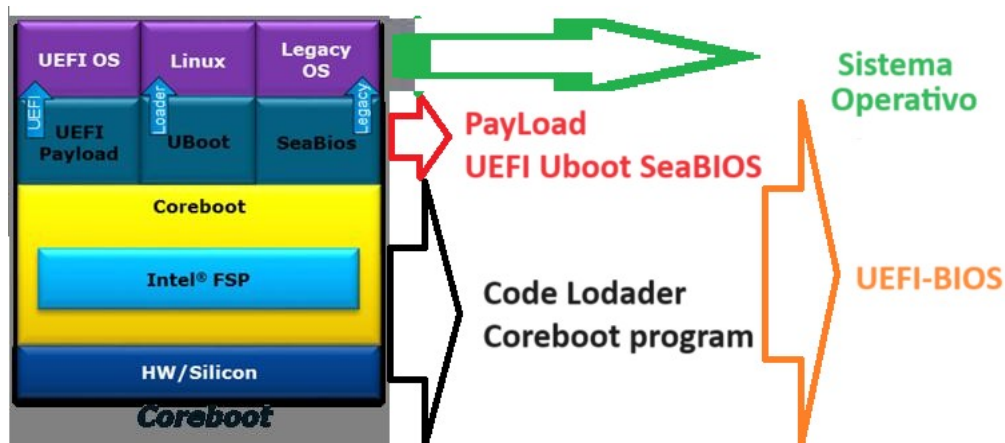


# UEFI\_BIOS

## Introduccion

EL BIOS es el programa encargado de inicializar el hardware a bajo nivel de una tarjeta madre, una vez funcional el hardware el programa comenzara a cargar al Sistema Operativo del equipo de computo, el cual hara uso de los servicios del recién inicializado Hardware.

Existen varios software de base que cumple la tarea del BIOS, entre ellos mencionamos U-Boot, Open-Bios, Libreboot, por mencionar algunos, nos centramos en alguno que permita implimentar una especificación UEFI, tenemos a SlimBoot y Coreboot como algunos candidatos, para este documento escogeremos este ultimo, el cual permite implementar como una "payload" varios tipos de especificaciones para "cargadores de sistema operativo", observe la siguiente figura:



Usando CoreBoot como base Elegiremos a UEFI como payload por lo que tendremos un esquema como la parte izquierda de la siguiente figura:

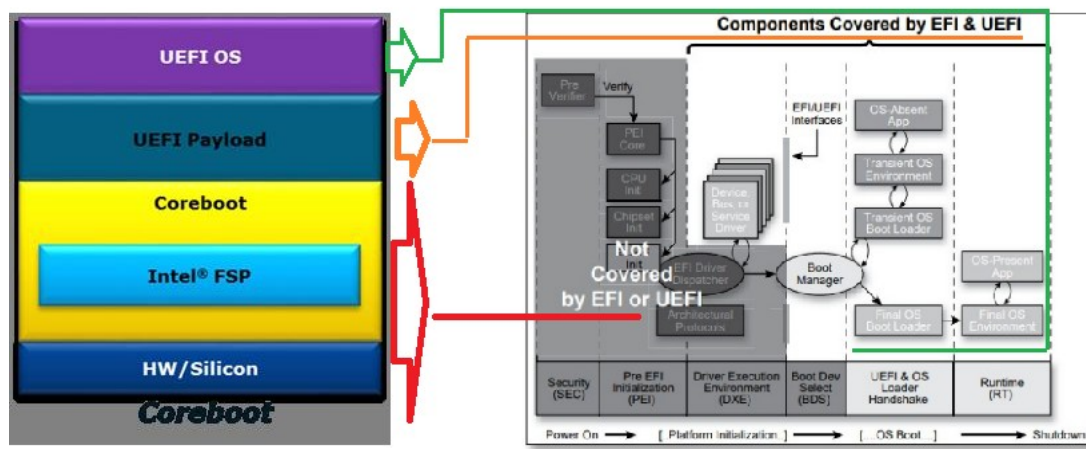


Figure 1.1 Where EFI and UEFI Fit into the Platform Boot Flow

La seccion derecha de la figura anterior muestra a detalle los servicios que "espera" la implementacion UEFI (sombreado obscuro linea roja) y los servicios que "provee"(linea naranja) dicha implementacion, podemos ver que algunos servicios de UEFI, se "traslapan" (linea verde) con la especificacion UEFI, dichos servicios no son proporcionados por la seccion UEFIPayload, sino creado a partir de esta en "tiempo de ejecución del sistema operativo" y acompañaran al sistema operativo mientras este "corriendo", esta carecteristica es una de los beneficios de la migracion del anterior BIOS PC de IBM.

Tanto la documentacion de EDK2 como de CoreBoot especifican el proceso que debe seguirse para unir los dos proyectos. Se muestra acontinuacion un ejemplo del uso del ambiente de desarrollo de EDK2 y uno para el de CoreBoot.

## Ejemplos practicos del uso de las herramientas de cada proyecto

El proposito de este apartado es mostrar como se usan los ambientes de desarrollo propios de cada proyecto, la primera practica indica como crear una aplicacion sencilla que ejecuta en EFI-Shell llamadas "modulos" en la jerga de EDKII, esta aplicacion no esta integrada al BIOS como lo hacen las *payloads*, son usadas para crear usualmente soluciones de mantenimiento que ofrecen los fabricantes de tarjetas madres (o fabricantes de equipos de computadoras personales). La segunda practica muestra como crear un BIOS completo combinando Coreboot y la Payload por defecto (SeaBIOS en este caso) en arquitectura ARM, el proposito es verificar que la instalación del ambiente esta correcta y funcional (como el proceso de compilación toma algo de tiempo se elegieron las opciones por defecto para agilizar el proceso). El tercer ejemplo es lo que nos interesa en cuestion, crear un BIOS con la especificacion implementada UEFI, es a dos partes la primera crear la payload por separado usando el ambiente de EDK2 y la segunda integrar esa payload en la compilacion de CoreBoot para formar el BIOS completo para nuestra maquina virtual.

## Ejemplos practicos del uso de las herramientas de cada proyecto

El equipo donde realizaremos nuestro desarrollo EDK2 Raspberry Pi 3, asegure de actualizar el sistema operativo primero que nada, complete los siguientes requisitos:

- Instalar un ambiente de desarrollo EDK2 (consulte el Apendice A).
- Contar con una maquina virtual, escogeremos Qemu por ser la más compatible con nuestro OS (la instalacion se indica el Apendice B).
- Poder compilar correctamente EDK2 para la arquitectura x86\_64 (esto en el apendice C).

Una vez que tenga instalado y funcional el ambiente para compilar en arquitectura x86\_64, procedemos a mostrar como compilar el paquete de ejemplo *MdeModulePkg.dsc*, el cual contine una descripcion de los archivos requeridos para construir la aplicacion UEFI:

```

rodrigo@raspberrypi:~/src/edk2 $ cat MdeModulePkg/MdeModulePkg.dsc
## @file
# EFI/PI Reference Module Package for All Architectures
#
# (C) Copyright 2014 Hewlett-Packard Development Company, L.P.<BR>
# Copyright (c) 2007 - 2021, Intel Corporation. All rights reserved.<BR>
# Copyright (c) Microsoft Corporation.
# Copyright (C) 2024 Advanced Micro Devices, Inc. All rights reserved.<BR>
#
#   SPDX-License-Identifier: BSD-2-Clause-Patent
#
##

[Defines]
  PLATFORM_NAME                = MdeModule
  PLATFORM_GUID                = 587CE499-6CBE-43cd-94E2-186218569478
  PLATFORM_VERSION             = 0.98
  DSC_SPECIFICATION            = 0x00010005
  OUTPUT_DIRECTORY             = Build/MdeModule
  SUPPORTED_ARCHITECTURES      = IA32|X64|EBC|ARM|AArch64|RISCV64|LOONGARCH64
  BUILD_TARGETS                = DEBUG|RELEASE|NOOPT
  SKUID_IDENTIFIER             = DEFAULT

!include MdePkg/MdeLibs.dsc.inc

[LibraryClasses]
#
# Entry point
#
PeiCoreEntryPoint|MdePkg/Library/PeiCoreEntryPoint/PeiCoreEntryPoint.inf
PeimEntryPoint|MdePkg/Library/PeimEntryPoint/PeimEntryPoint.inf
DxeCoreEntryPoint|MdePkg/Library/DxeCoreEntryPoint/DxeCoreEntryPoint.inf

```

Para esto usaremos el script *build* que se genera por defecto con la instalación de la paquetería EDK2, antes de comenzar a usar el script debemos asegurar que las variables de ambiente que usa el proyecto EDK2 estén configuradas correctamente, ejecutamos el script de configuración:

```

rodrigo@raspberrypi:~/src/edk2 $ . ./edksetup.sh
Loading previous configuration from /home/rodrigo/src/edk2/Conf/BuildEnv.sh
Using EDK2 in-source Basetools
WORKSPACE: /home/rodrigo/src/edk2
EDK_TOOLS_PATH: /home/rodrigo/src/edk2/BaseTools
CONF_PATH: /home/rodrigo/src/edk2/Conf
rodrigo@raspberrypi:~/src/edk2 $

```

El cual nos configura las variables a los directorios correctos, después de eso utilizaremos el script *build* con los siguientes parametros:

```

rodrigo@raspberrypi:~/src/edk2 $ build -a X64 -t GCC5 -b DEBUG -p MdeModulePkg/MdeModulePkg.dsc

```

El parametro *-a X64* indica que tipo de arquitectura estamos "apuntando" compilar nuestra aplicación (es decir al microprocesador que ejecutara la aplicación), la opción *-t GCC5* se refiere a usar el conjunto de herramientas de compilación GCC5 (el anexo C describe cual compilador es el usado), y por ultimo la orden *-p MdeModulePkg/MdeModulePkg.dsc* indica la dirección del archivo de descripción que contiene el listado de los archivos usados para la compilación de nuestra aplicación, cuando la compilación termine enviara un mensaje de éxito.

```

make: No se hace nada para 'tbuild'.
make: No se hace nada para 'tbuild'.
Building ... /home/rodrigo/src/edk2/MdeModulePkg/Universal/CapsulePei/CapsulePei.inf [X64]
Building ... /home/rodrigo/src/edk2/MdeModulePkg/Universal/Variable/MmVariablePei/MmVariablePei.inf [X64]
make: No se hace nada para 'tbuild'.
Building ... /home/rodrigo/src/edk2/MdeModulePkg/Bus/Pci/UfsPciHcPei/UfsPciHcPei.inf [X64]
make: No se hace nada para 'tbuild'.
make: No se hace nada para 'tbuild'.
make: No se hace nada para 'tbuild'.
make: No se hace nada para 'tbuild'.
- Done -
Build end time: 22:19:09, Aug.13 2024
Build total time: 00:01:12
rodrigo@raspberrypi:~/src/edk2 $

```

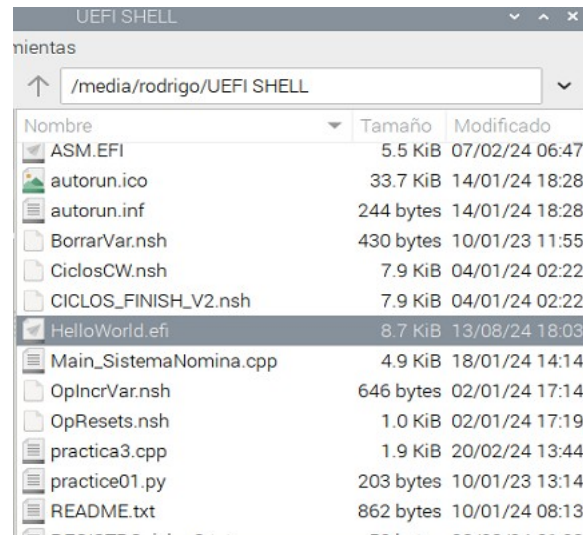
Ahora para usar nuestro paquete debemos de copiarlo o moverlo a una USB con sistema de archivo FAT32, el cual se encuentra en la direcccion siguiente:

```

rodrigo@raspberrypi:~/src/edk2/Build/MdeModule/DEBUG_GCC5/X64 $ pwd
/home/rodrigo/src/edk2/Build/MdeModule/DEBUG_GCC5/X64
rodrigo@raspberrypi:~/src/edk2/Build/MdeModule/DEBUG_GCC5/X64 $ ls HelloWorld.efi
HelloWorld.efi
rodrigo@raspberrypi:~/src/edk2/Build/MdeModule/DEBUG_GCC5/X64 $

```

El nombre del paquete construido fue *HelloWorld.efi*. Lo copiamos a nuestra USB:



Y ejecutamos nuestra maquina virtual con acceso a la unidad USB, para esto necesitamos averiguar el puerto en el que se encuentra nuestra usb:

```

rodrigo@raspberrypi:~/src/edk2 $ lsusb
Bus 001 Device 004: ID ffff:5678 Blackpcs
Bus 001 Device 003: ID 0424:ec00 Microchip Technology, Inc. (formerly SMSC) SMSC9512/9514 Fast Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Microchip Technology, Inc. (formerly SMSC) SMC9514 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
rodrigo@raspberrypi:~/src/edk2 $

```

En este caso es el puerto 4, tomamos esta informacion y la agregamos a los parametros de nuestra llamada a Qemu:

*sudo qemu-system-x86\_64 -bios OVMF.fd -m 256M -net none -nographic -usb -device usb-ehci,id=ehci -device usb-host,hostbus=<lsusb bus#>,hostaddr=<lsusb device#>*

```

rodrigo@raspberrypi:~/src/edk2 $ lsusb
Bus 001 Device 004: ID ffff:5678 Blackpcs
Bus 001 Device 003: ID 0424:ec00 Microchip Technology, Inc. (formerly SMSC) SMSC9512/9514 Fast Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Microchip Technology, Inc. (formerly SMSC) SMC9514 Hub
rodrigo@raspberrypi:~/src/edk2 $ sudo qemu-system-x86_64 -bios OVMF.fd -m 256M -net none -nographic -usb -device usb-ehci,id=ehci -device usb-host,hostbus=1,hostaddr=4

```

Observe como el numero de Bus y el numero de direccion de puerto coinciden, una vez iniciada la sesion virtual del UEFI-Shell:

```
UEFI Interactive Shell v2.2
edk2-stable202311 (https://github.com/pbatard/UEFI-Shell)
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
FS0: Alias(s):HD1a0b:;BLK2:
      PciRoot(0x0)/Pci(0x3,0x0)/USB(0x0,0x0)/HD(1,MBR,0x0674A67F,0x800,0x3A97800)
BLK0: Alias(s):
      PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
BLK1: Alias(s):
      PciRoot(0x0)/Pci(0x3,0x0)/USB(0x0,0x0)
Press ESC in 4 seconds to skip startup.nsh or any other key to continue.
Shell> fs0: (A)
FS0:\> HelloWorld.efi (B)
UEFI Hello World! (C)
FS0:\>
```

Accedemos a la memoria USB con `fs0:` indicado en (A), escribimos el nombre de nuestra aplicacion `HelloWorld.efi` (B) y observamos el resultado en (C) donde se muestra el mensaje UEFI Hello World!.

## Practica 2 para la creacion de una BIOS simple para ARM mediante CoreBoot para ejecutarse en la maquina virtual Qemu

Complete la instalacion de un ambiente de desarrollo dentro de un contenedor para CoreBoot como indican las instrucciones del Apendice D, acontinuacion compile un archivo ROM binario de prueba para inicializar el software de virtualización de Qemu, **Nota:** inicie el contenedor de Docker como se especifico al final del Apendice D, y ejecute el comando `make clean` para borrar cualquier resto de compilaciones anteriores:

```
root@13d893afc647:/home/coreboot/coreboot# make clean
root@13d893afc647:/home/coreboot/coreboot#
```

Seguido del comando `make distclean`, para regresar las opciones de compilacion a "Default".

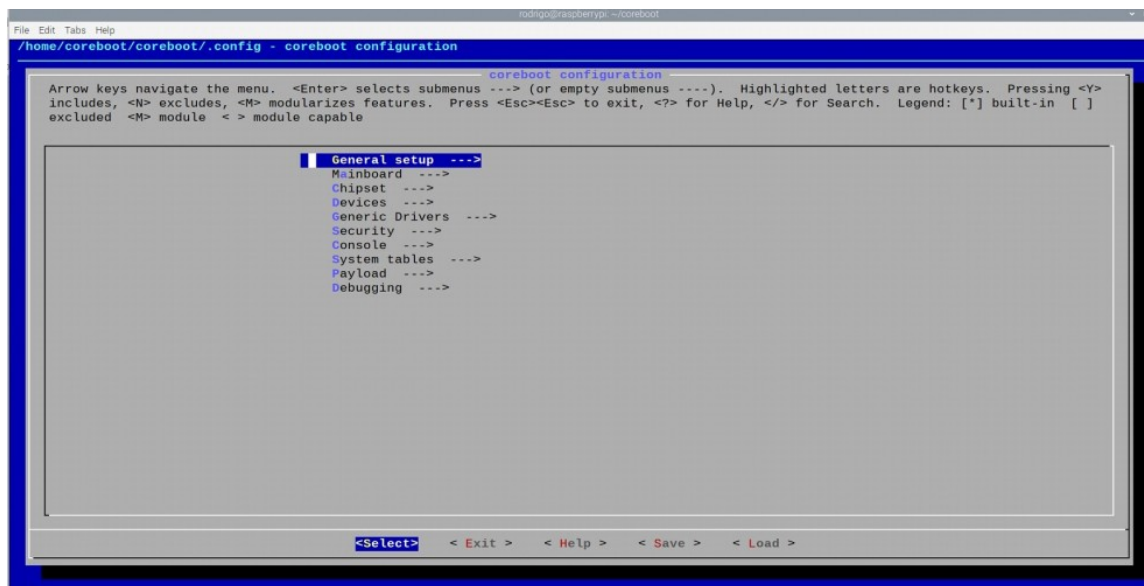
```
root@235918de96d7:/home/coreboot/coreboot# make distclean
```

De esta forma nos aseguramos iniciar con una configuracion en blanco, posteriormente haremos uso de la herramienta del menu de configuración con el comando `make menuconfig`:

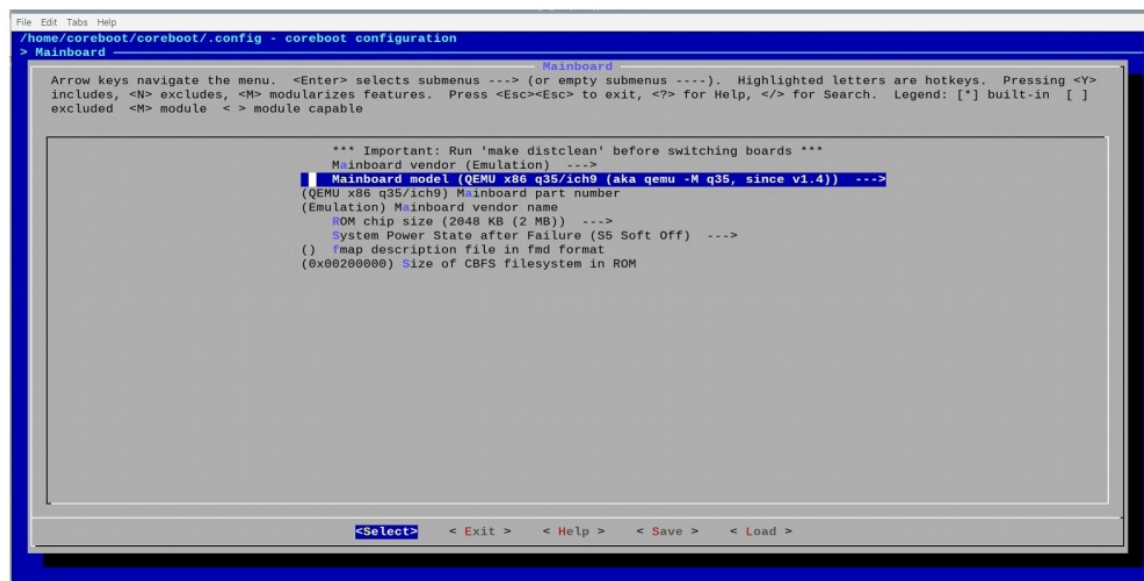
```
root@235918de96d7:/home/coreboot/coreboot# make menuconfig
```

Nos presentara en pantalla un menu visual:

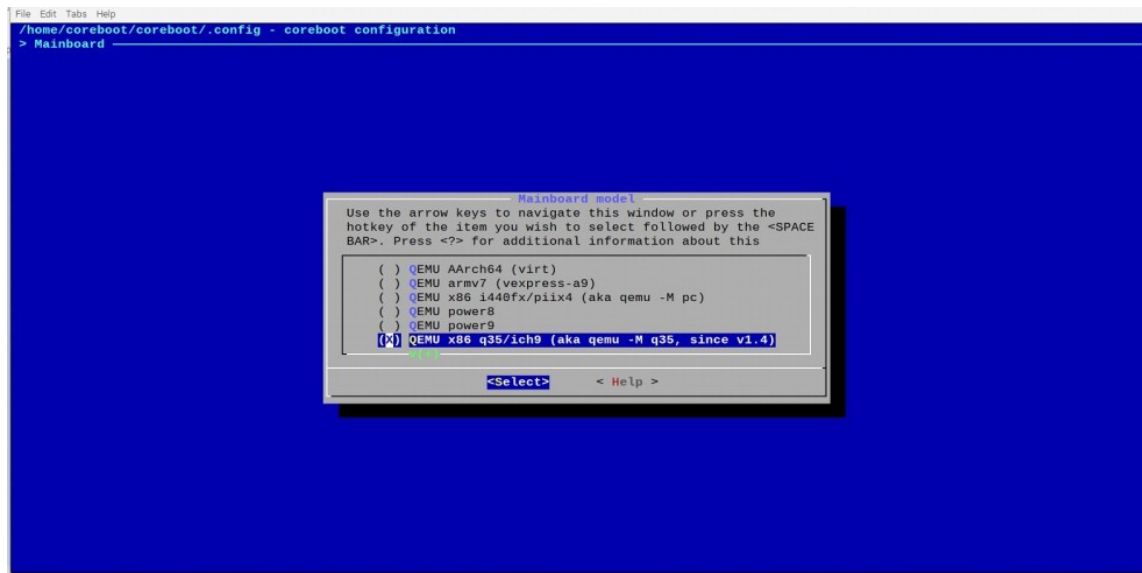




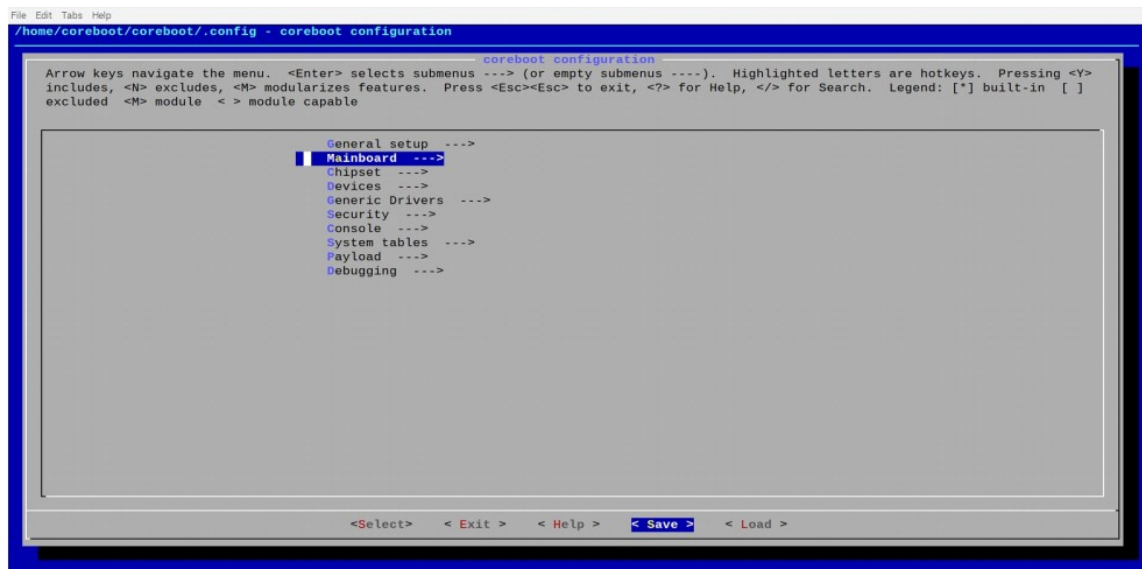
Elegiremos la opción "Main Board":



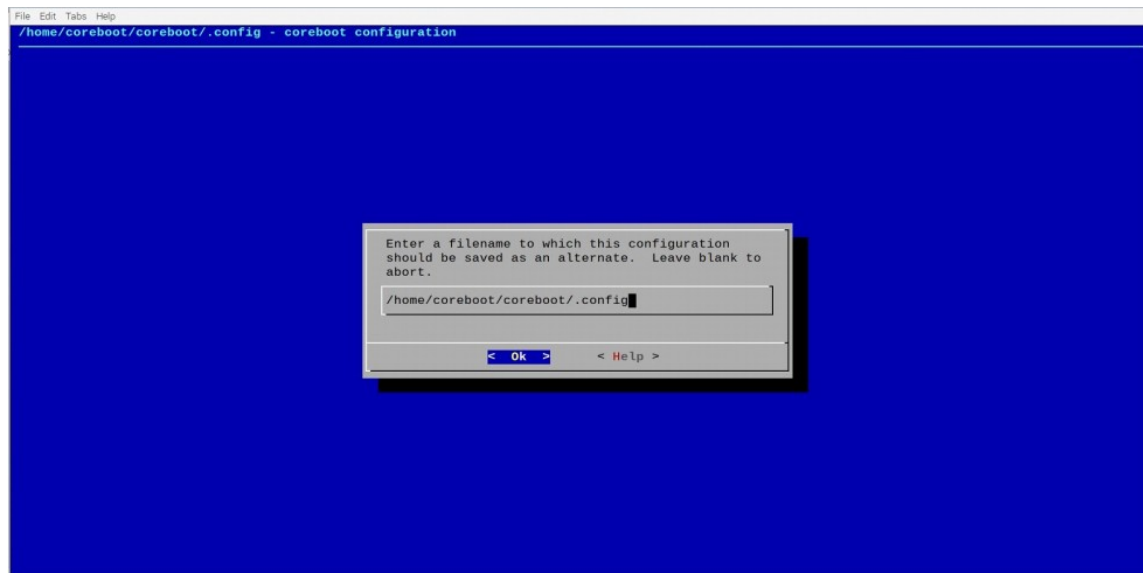
De ahí la opción MainBoard mode (Qemu..", la cual nos despliega el siguiente listado:



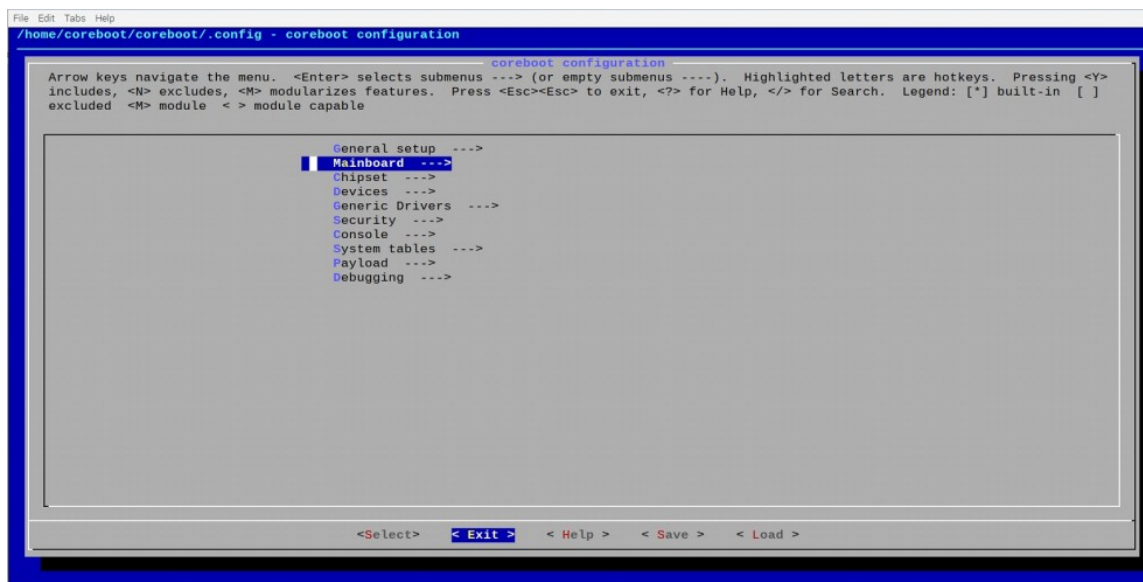
Marcamos la opcion "QEMU x86 q35/ich9 ...", la cual corresponde al modelo de configuraci3n de ROOM del software de la maquina Virtual.



Tras seleccionarlo regresaremos al men3 anterior y escogeremos la opcion "<Save>", de la parte inferior del listado.

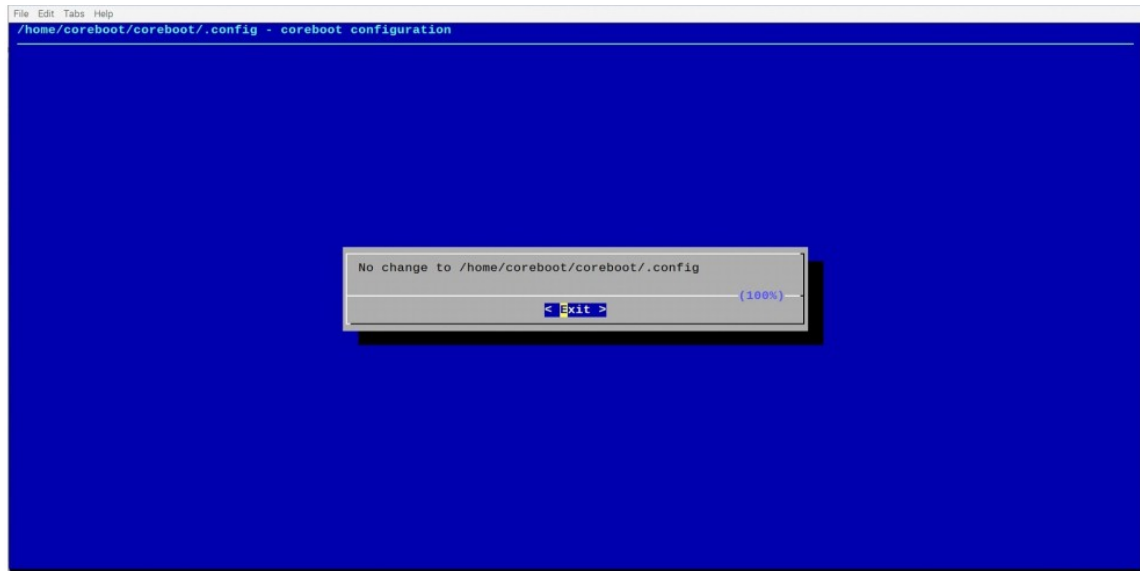


Nos mostrara la opcion del nombre del archivo y de la dirección donde se guardara el archivo con esta configuración, lo dejamos tal como esta.



Terminamos la configuracion eligiendo la opcion "<Exit>" .





Tras elegir la opción de "<Exit>" nos indicara si se realizaron cambios en el archivo, en este caso estamos usando las opciones por defecto así que puede que no veamos cambio alguno.

```
root@2235918de96d7:/home/coreboot/coreboot# make
```

Regresaremos al prompt del Terminal, donde usaremos el comando "make", el cual iniciara el proceso de compilación acorde al archivo de configuración que construimos con la herramienta Menu (guardado en `:/home/coreboot/coreboot/.config`). podemos ver que iniciara el proceso de compilacion, el cual continuara durante un buen tiempo dependiendo de las capacidades de computo del sistema que se haya elegido para ejecutar la tarea (en este caso una Raspberry Pi 3, tomara una buena cantidad considerable de tiempo). Al finalizar el sistema creara un archivo ROM que podemos usar para iniciar nuestra maquina virtual.

```
HOSTCC      cbfstool/subpart_hdr_2.o
HOSTCC      cbfstool/subpart_entry_1.o
HOSTCC      cbfstool/cse_surger (link)
FMAP REGION: COREBOOT
Name      Offset      Type      Size  Comp
cbfs master_header  0x0        cbfs header  32 none
fallback/romstage  0x80        stage      17536 none
fallback/ramstage  0x4580     stage      71749 LZMA (149844 decompressed)
config      0x15e40     raw        2392 LZMA (7426 decompressed)
revision    0x16800     raw        709 none
build_info  0x16b00     raw        104 none
fallback/dsdt.aml  0x16bc0     raw        7175 none
cmos_layout.bin  0x18800     cmos_layout  640 none
fallback/postcar  0x18ac0     stage      21536 none
fallback/payload  0x1df40     simple elf  72645 none
payload_config  0x2fb40     raw        1621 none
payload_revision  0x301c0     raw        237 none
(empty)      0x30300     null       1886884 none
bootblock    0x1fcd0     bootblock   12288 none

Built emulation/qemu-q35 (QEMU x86 q35/ich9)
root@46c68f0282d:/home/coreboot/coreboot# ls
```

La imagen anterior muestra el final de la compilacion, la cual se creo en el directorio `/build`, donde encontraremos un archivo denominado `coreboot.rom`:

```
root@46c68f0282d:/home/coreboot/coreboot/build# ls
auto.conf      cb-config.rustcfg  config.h      cse_surger    dsdt.dsl      generated      postcar      rmodules_ppc64  smm           util
auto.conf.cnd  cbfs               coreboot.pre  decompressor  fnap.desc     ifuitool       ramstage     rmodules_riscv  smmstub       verstage
bootblock      cbfstool           coreboot.rom  dsdt.aml      fnap.fmap     libgnat-x86_32  rmodtool     rmodules_x86_32  static.h       xcompile
build.h        cmos_layout.bin   cpu           dsdt.asl      fnap.fnd      mainboard      rmodules_arm  rmodules_x86_64  static_devices.h
build_info     config             cse_fpt      dsdt.d         fnap_config.h  option_table.h  rmodules_arm64  romstage        static_fw_config.h
```

El cual usaremos para inicializar nuestra maquina virtual, con el siguiente comando:

`qemu-system-x86_64 -M q35 -bios ./build/coreboot.rom -serial stdio -display none`

```
root@ec46c68f0282d:/home/coreboot/coreboot# qemu-system-x86_64 -M q35 -bios ./build/coreboot.rom -serial stdio -display none

[NOTE] coreboot-coreboot-unknown Sat Jun 08 01:51:33 UTC 2024 x86_32 bootblock starting (log level: 7)...
[DEBUG] FMAP: Found "FLASH" version 1.1 at 0x0.
[DEBUG] FMAP: base = 0xffa00000 size = 0x200000 #areas = 3
[DEBUG] FMAP: area COREBOOT found @ 200 (2096640 bytes)
[INFO] CBFS: ncache 0x00014e00 built for 13 files, used 0x2d4 of 0x4000 bytes
[INFO] CBFS: Found 'fallback/romstage' @0x80 size 0x4480 in ncache 0x00014e2c
[DEBUG] BS: bootblock times (exec / console): total (unknown) / 117 ms

[NOTE] coreboot-coreboot-unknown Sat Jun 08 01:51:33 UTC 2024 x86_32 romstage starting (log level: 7)...
[DEBUG] SMBus controller enabled
[INFO] QEMU: firmware config interface detected
[INFO] Firmware config version id: 3
[INFO] QEMU: firmware config: Found 'etc/e820'
[DEBUG] CBMEM:
[DEBUG] TMO: root @ 0x07fff000 254 entries.
[DEBUG] TMO: root @ 0x07ffae00 62 entries.
[DEBUG] FMAP: area COREBOOT found @ 200 (2096640 bytes)
```

Esto iniciara la ejecucion de la Maquina Virtual con nuestro Bios.

**Nota** Este BIOS es para uso exclusivo de la maquina virtual con QEMU, no intente usarlo en un

Hardware Fisico.

## Practica 3 para ilustrar la integracion de la payload UEFI en la compilacion de CoreBoot para formar un UEFI-BIOS completo.

Esta practica es a dos partes, la primera es compilar nuestra payload por separado, Consulte el Apendice E para realizar el proceso, y conseguir el archivo *UEFIPAYLOAD32.fd* (renombrado de *UEFIPAYLOAD.fd* originalmente). Copie ese archivo en lo mas "alto" del directorio de CoreBoot:

```
rodri@raspberrypi: /coreboot $ ls
AUTHORS  build  build.32  COPYING  core  MAINTAINERS  qemu.cbfstool_20240904-083756_67320.core  UEFIPAYLOAD.fd
gnat.adc  Makefile  Makefile.mk  README.md  toolchain.mk  UEFIPAYLOAD32.fd
```

La segunda parte es compilar el proyecto CoreBoot integrando la payload que creamos anteriormente, para esto ingrese al conenedor de docker.

```
rodri@raspberrypi: /coreboot $ sudo docker run --privileged --rm tonistiigi/binfmt --install all
{
  "supported": [
    "linux/arm64",
    "linux/amd64",
    "linux/riscv64",
    "linux/ppc64le",
    "linux/s390x",
    "linux/386",
    "linux/mips64le",
    "linux/mips64",
    "linux/arm/v7",
    "linux/arm/v8"
  ],
  "emulators": [
    "qemu-i386",
    "qemu-mips64",
    "qemu-mips64le",
    "qemu-ppc64le",
    "qemu-riscv64",
    "qemu-s390x",
    "qemu-x86_64"
  ]
}
rodri@raspberrypi: /coreboot $ sudo docker run -w /home/coreboot/coreboot -u root -it -v $PWD:/home/coreboot/coreboot --rm --platform linux/amd64 coreboot/coreboot-sdk /bin/bash
root@84e03adc3a3e3:/home/coreboot/coreboot#
```

Para configurar las reglas de compilacion siguiendo las instrucciones como lo muestra el Apendice F. Una vez realizado el proceso de configuracion ejecute el comando *make* y espere a que la compilacion termine, esto puede tomar mucho tiempo dependiendo de las capacidades del hardware:

```

root@4e03adc3a3e3:/home/coreboot/coreboot# make menuconfig

*** End of the configuration.
*** Execute 'make' to start the build or try 'make help'.

root@4e03adc3a3e3:/home/coreboot/coreboot# make
Updating git submodules.
FMAP REGION: COREBOOT


| Name               | Offset   | Type        | Size     | Comp                       |
|--------------------|----------|-------------|----------|----------------------------|
| cbfs_master_header | 0x0      | cbfs header | 32       | none                       |
| fallback/romstage  | 0x80     | stage       | 17736    | none                       |
| fallback/ramstage  | 0x4640   | stage       | 72238    | LZMA (148036 decompressed) |
| config             | 0x16100  | raw         | 2618     | LZMA (8093 decompressed)   |
| revision           | 0x18b80  | raw         | 727      | none                       |
| build_info         | 0x18e80  | raw         | 117      | none                       |
| fallback/dsdt.aml  | 0x18f40  | raw         | 7175     | none                       |
| cmos_layout.bin    | 0x18b80  | cmos_layout | 640      | none                       |
| fallback/postcar   | 0x10e40  | stage       | 21608    | none                       |
| fallback/payload   | 0x1e800  | simple elf  | 971953   | none                       |
| (empty)            | 0x10b800 | null        | 15406500 | none                       |
| bootblock          | 0xfbcd0  | bootblock   | 12288    | none                       |


Built emulation/qemu-x86 (QEMU x86 q35/ich9)
root@4e03adc3a3e3:/home/coreboot/coreboot# exit
exit
rodrigo@raspberrypi:~/coreboot $

```

Asegure que la compilacion fue exitosa, salga del contenedor usando el comando `exit`, localice la carpeta `build`, dentro debe estar un archivo con el nombre `coreboot.rom`, copie la carpeta con la instruccion `sudo cp -rf build build_32` para evitar perder los resultados de la compilación accidentalmente al volver a realizar una nueva compilación.

```

rodrigo@raspberrypi: ~/coreboot $ ls build
auto.conf      build.ht       cbfstool       coreboot.rom   decompressor  dsdt.dsl       fmap.fmd       mainboard      modtool        modules_riscv  smm            static.h
auto.conf.cmd  build_info     cmos_layout.bin  cpio           dsdt.aml      fmap_config.h  generated      option_table.h  modules_arm    modules_x86_32  smmstub        util
bootblock      cb-config.rustcfg  config         cse_fpt        dsdt.asl      fmap_desc      libtool        postcar        modules_arm64  modules_x86_64  static_devices.h  verstage
build.h         cbfs           cse_serger      config.h       dsdt.d         fmap.fmap      libnat-x86_32  ramstage      modules_ppc64  ramstage       static_fw_config.h  xcompile
rodrigo@raspberrypi: ~/coreboot $ sudo cp -rf build build_32

```

Para comprobar el funcionamiento correcto del nuevo UEFI-BIOS en la maquina virtual Qemu use la siguiente instruccion:

```
sudo qemu-system-x86_64 -M q35 -bios ./build_32/coreboot.rom -serial stdio -display none
```

```
rodrigo@raspberrypi:~/coreboot $ qemu-system-x86_64 -M q35 -bios ./build_32/coreboot.rom -serial stdio -display none
```

La maquina virtual comenzara su ejecucion y cargara el EFI-Shell

```

[Bds]UnregisterKeyNotify: 0002/0000 Success
[Bds]UnregisterKeyNotify: 0002/0000 Success
[Bds]UnregisterKeyNotify: 0000/0000 Success
PROGRESS CODE: V03051001 10
Memory Previous Current Next
Type Pages Pages Pages
=====
08 00000019 0000000A 00000013
0A 00000004 00000000 00000004
00 00000008 00000000 00000008
06 00000100 00000050 00000100
05 00000100 00000038 00000100
[Bds]Booting EFI Shell
[Bds] Expand Fv(8053C21A-8E58-4576-95CE-089E87975D23)/FvFile(7C04A583-9E3E-4F1C-A065-E052680B4D1) -> Fv(8053C21A-8E58-4576-95CE-089E87975D23)/FvFile(7C04A583-9E3E-4F1C-A065-E052680B4D1)
PROGRESS CODE: V03058000 10
InstallProtocolInterface: 5B1B3141-95B2-1102-8E39-00A0C969723B 526F240
Loading driver at 0x00004B8A00 EntryPoint=0x000044E785E Shell.efi
InstallProtocolInterface: BC82157E-9E33-4FEC-9920-2D0B86D750DF 61B2888
ProtectUefiImageCommon - 0x526F240
- 0x00000000/4B8A00 - 0x00000000/00F93C0
!!!!!!! Image Section Alignment(0x40) does not match Required Alignment (0x1000) !!!!!!!!
ProtectUefiImage failed to create image properties record
PROGRESS CODE: V03058001 10

UEFI Interactive Shell v2.287477C2-69C7-1102-8E39-00A0C969723B 4F5C8A0
EDK II IProtocolInterface: 752F3138-4E16-4FDC-A22A-ESF46812FACA 4F5B6B8
UEFI v2.70 (EDK II, 0x00010000)008-7F9B-4F3D-87AC-60C3FEF5D4AE 4545560
Mapping table
BLK0: Alias(s):
PciRoot(0x0)/Pci(0x1F,0x2)/Sata(0x2,0xFFFF,0x0)
Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
Shell> reset -s
Reset with <null string> (0 bytes)PROGRESS CODE: V0311100A 10
DXE ResetSystem2: ResetType Shutdown, Call Depth = 1.
qemu-system-x86_64: terminating on signal 2
rodrigo@raspberrypi:~/coreboot $

```

Para terminar correctamente de nuestra maquina virtual usamos la instruccion `reset -s` en el EFI-Shell (puede que requiera usar la combinacion de teclas CTRL+C adicionalmente)

```

Mapping table
BLK0: Alias(s):
PciRoot(0x0)/Pci(0x1F,0x2)/Sata(0x2,0xFFFF,0x0)
Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
Shell> reset -s
Reset with <null string> (0 bytes)PROGRESS CODE: V0311100A 10
DXE ResetSystem2: ResetType Shutdown, Call Depth = 1.
qemu-system-x86_64: terminating on signal 2
rodrigo@raspberrypi:~/coreboot $

```

Nota: este UEFI-BIOS solo funciona con la maquina virtual Qemu, no intente usarlo en Hardware Real

## Desarrollo de aplicaciones UEFI

Para probar nuestras aplicaciones UEFI tenemos dos opciones hasta ahora, nuestro homemade UEFI-BIOS o la Open Virtual Machine Firmware (OVMF), antes de explicar detalles de cada una de las dos opciones recomendamos instalar la herramienta *Tmux*, la cual permite crear "sesiones persistentes", las cuales usaremos como "envolturas" para lanzar nuestras maquinas virtuales, esto para poder terminar la maquina virtual en caso de que esta no responda a nuestros comandos (el termino coloquial es "quedarse colgada"), y asi evitar reiniciar nuestro equipo de desarrollo ("raspberry pi").

Para instalar Temux debemos asegurarnos primero que nuestro sistema esta actualizado:

*sudo apt-get update*

```
rodrigo@raspberrypi:~$ sudo apt-get update
Des:1 http://security.debian.org/debian-security bullseye-security InRelease [27.2 kB]
Obj:2 http://deb.debian.org/debian bullseye InRelease
Des:3 http://deb.debian.org/debian bullseye-updates InRelease [44.1 kB]
Obj:4 https://download.docker.com/linux/debian bullseye InRelease
Obj:5 http://archive.raspberrypi.org/debian bullseye InRelease
Descargados 71.2 kB en 3s (22.7 kB/s)
Leyendo lista de paquetes... 35%
```

*sudo apt-get upgrade*

```
rodrigo@raspberrypi:~$ sudo apt-get upgrade
Leyendo lista de paquetes... Hecho
Creando Árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Calculando la actualización... 50%
```

lo instalamos con la siguiente orden:

*sudo apt-get install tmux*

```
rodrigo@raspberrypi:~$ sudo apt-get install tmux
Leyendo lista de paquetes... Hecho
Creando Árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
```

Una vez instalado creamos una sesion de prueba usando:

*tmux new-session -s <Nombre>*

```
rodrigo@raspberrypi:~$ tmux new-session -s test
```

La terminal cambiara de forma:

```
File Edit Setup Control Window Help
rodrigo@raspberrypi:~ $ 
[test] 0:bash*
```

Observe que apareciera una barra de color verde en la parte inferior de la terminal con el nombre de nuestra sesion. para dar por terminada la sesion escribimos `exit` o presionamos las teclas 'CTRL' + 'b' y despues la tecla 'x', la barra inferior de estado cambiara a amarillo y nos preguntara si deseamos salir, escogemos 'y' para salir:

```
File Edit Setup Control Window Help
rodrigo@raspberrypi:~ $ 
kill-pane 0? (y/n)
```

Para mayor informacion sobre el uso de Temux, consulte la pagina de los desarrolladores:

<https://github.com/tmux/tmux/wiki>

Para desarrollar aplicaciones necesitamos una USB fisica con formato FAT32 (debido a que UEFI-Shell no cuenta con almacenamiento externo), la cual debe contener el archivo de ejemplo HelloWorld (creado en la practica 1 mas arriba) y lanzar dentro de nuestra sesion de Temux la maquina virtual ya sea con nuestro UEFI-BIOS o con la OVMF, la cual debe contar con acceso a la memoria USB fisica donde guardamos nuestras aplicaciones, para poder en lazar nuestra memoria USB a la maquina virtual debemos saber donde esta localizada en nuestro sistema, use el comando `lsusb` y tome nota del numero de Bus y la Direccion del dispositivo:

```
rodrigo@raspberrypi:~ $ lsusb
Bus 001 Device 004: ID ffff:5678 Blackpcs
Bus 001 Device 003: ID 0424:ec00 Microchip Technology, Inc. (formerly SMSC) SMSC9512/9514 Fast Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Microchip Technology, Inc. (formerly SMSC) SMC9514 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Los detalles adicionales sobre como ejecutar cada maquina virtual se muestran en los siguientes parrafos.

## homemade UEFI-BIOS

Dentro de nuestra sesion de Temux lanzamos una maquina virtual con nuestro BIOS usando el comando siguiente:

```
sudo qemu-system-x86_64 -M q35 -bios ./coreboot/build_32/coreboot.rom -m
```

256M -net none -nographic -usb -device usb-ehci,id=ehci -device usb-host,hostbus=<lsusb#bus>,hostaddr=<lsusb#device>

```
rodrigo@raspberrypi:~$ sudo qemu-system-x86_64 -m 335 -bios ./coreboot/build_32/coreboot.rom -m 256M -net none -nographic -usb -device usb-ehci,id=ehci -device usb-host,hostbus=1,hostaddr=4
```

Ya que este en el prompt del UEFI-Shell, escriba lo siguiente:

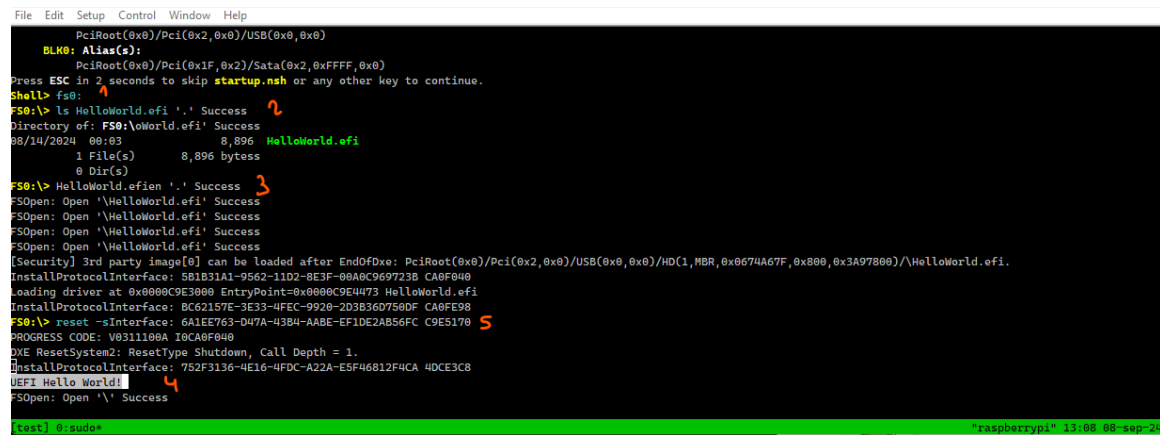
(1 en la imagen)- fs0:

(2 en la imagen)- ls HelloWorld.efi

(3 en la imagen)- HelloWorld.efi

(4 en la imagen)- Observe el resultado "UEFI Hello World"

(5 en la imagen)- reset -s



```
File Edit Setup Control Window Help
PciRoot(0x0)/Pci(0x2,0x0)/USB(0x0,0x0)
BLK0: Alias(0):
PciRoot(0x0)/Pci(0x1F,0x2)/Sata(0x2,0xFFFF,0x0)
Press ESC in 2 seconds to skip startup.nsh or any other key to continue.
Shell> fs0:
FS0:\> ls HelloWorld.efi
Directory of: FS0:\oWorld.efi
08/14/2024 00:03      8,896 HelloWorld.efi
1 File(s)      8,896 bytes
0 Dir(s)
FS0:\> HelloWorld.efi
[Security] 3rd party image[0] can be loaded after EndOfDxe: PciRoot(0x0)/Pci(0x2,0x0)/USB(0x0,0x0)/HD(1,MBR,0x0674A67F,0x800,0x3A97800)/\HelloWorld.efi.
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C9697238 CABF040
Loading driver at 0x0000C9E3000 EntryPoint=0x0000C9E4473 HelloWorld.efi
InstallProtocolInterface: BCG2157E-3E33-4FEC-992B-2D3B36D7580F CABFE98
FS0:\> reset -s
PROGRESS CODE: 0B311100A 10CABF040
DYE ResetSystem2: ResetType Shutdown, Call Depth = 1.
InstallProtocolInterface: 752F3136-4E16-4FDC-A22A-E5F46812F4CA 4DCE3C8
UEFI Hello World!
FS0:\>
```

Si el sistema se "cuelga" use la orde CTRL+B y X para salir de la sesion de Temux y terminar la ejecucion de Tmux.

## OVMF firmware para UEFI

En la sesion de Temux ejecutamos la maquina virtual con la siguiente linea:

sudo qemu-system-x86\_64 -bios OVMF.fd -m 256M -net none -nographic -usb -device usb-ehci,id=ehci -device usb-host,hostbus=<lsusb#bus>,hostaddr=<lsusb#device>

```
rodrigo@raspberrypi:~$ sudo qemu-system-x86_64 -bios OVMF.fd -m 256M -net none -nographic -usb -device usb-ehci,id=ehci -device usb-host,hostbus=1,hostaddr=4
```

una vez inicializado en el prompt del UEFI-Shell, escriba lo siguiente:

(1 en la imagen)- fs0:

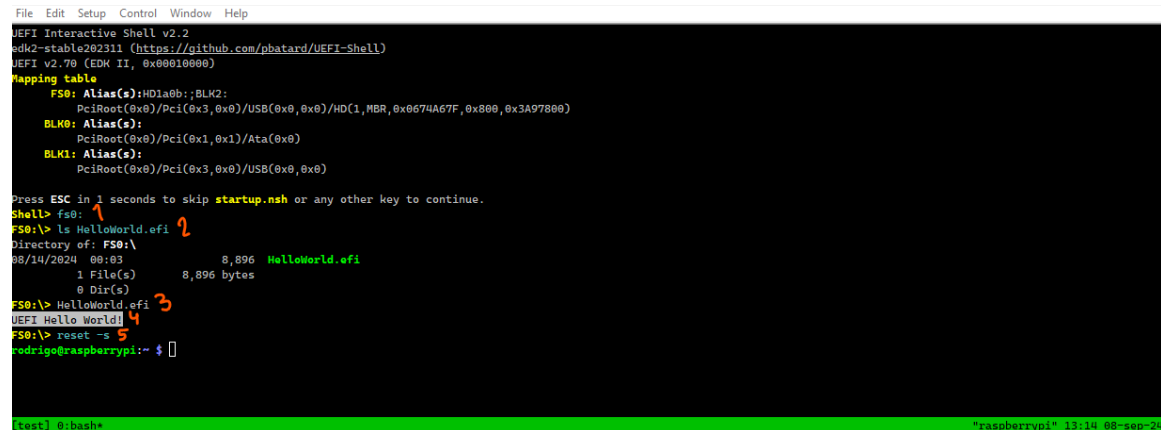


(2 en la imagen)- `ls HelloWorld.efi`

(3 en la imagen)- `HelloWorld.efi`

(4 en la imagen)- `Observe el resultado "UEFI Hello World"`

(5 en la imagen)- `reset -s`



```
File Edit Setup Control Window Help
UEFI Interactive Shell v2.2
edk2-stable202311 (https://github.com/pbatard/UEFI-Shell)
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
FS0: Alias(s): HD1a0b:;BLK2:
      PciRoot(0x0)/Pci(0x3,0x0)/USB(0x0,0x0)/HD(1,MBR,0x6674A67F,0x800,0x3A97800)
BLK0: Alias(s):
      PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
BLK1: Alias(s):
      PciRoot(0x0)/Pci(0x3,0x0)/USB(0x0,0x0)

Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
Shell> fs0:
FS0:> ls HelloWorld.efi
Directory of: FS0:\
08/14/2024  00:03                8,896  HelloWorld.efi
              1 File(s)                8,896 bytes
              0 Dir(s)
FS0:> HelloWorld.efi
UEFI Hello World!
FS0:> reset -s
rodrigo@raspberrypi:~$
```

En caso de que el sistema se "cuelgue" use la combinacion de teclas CTRL+B y X, para terminar la sesion de Temux.

## Donde probar nuestras aplicaciones UEFI

Las dos opciones OVMF y UEFI-BIOS "casero", funcionan de manera muy similar, para esta ultima es muy frecuente tener que terminar la sesion de Temux para poder terminar correctamente la ejecución de la maquina virtual, genera mucha información de diagnostico "adicional" cada que ejecutamos un comando, esto es util para entender como se hacen las llamadas en la especificación UEFI, si se desea hacer automatizacion requerrira codificacion adicional para limpiar la cadena de esos caracteres adicionales y tampoco es muy estable a la hora de terminar la ejecucion de la Maquina Virtual, OVMF es mejor en desempeño genera una salida mas parecida a lo que haria un hardware real ("mas limpia de caracteres"), lo cual facilita la automatización, rara vez se "cuelga" y en general es la opcion con soporte de la comunidad de EDK2, por lo cual sera la elegida para el desarrollo de aplicaciones UEFI, la version UEFI-BIOS se utilizara para mostrar como crear un BIOS mas "seguro" en caso de que necesitemos una solucion "completa" para nuestro Sistema "Fisico" (Hardware), al final todo Software de Base debe ser probado en Hardware Real fuera de la emulación para asegurar que funciona correctamente.

---

## APENDICE A

### Como instalar un ambiente de diseño usando el proyecto EDK2

Comenzamos creando un directorio de trabajo, para este ejemplo lo llamaremos EDK2

```
rodrigo@raspberrypi:~$ ls
rodrigo@raspberrypi:~$ sudo mkdir EDK2
rodrigo@raspberrypi:~$ ls
rodrigo@raspberrypi:~$ cd EDK2/
rodrigo@raspberrypi:~/EDK2$ ls
rodrigo@raspberrypi:~/EDK2$
```

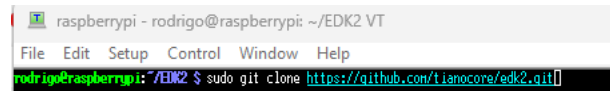
Dentro clonamos el repositorio de Git Hub, con las siguientes instrucciones:

```
sudo git clone --recurse -submodules https://github.com/tianocore/edk2.git
```

Si presenta problemas de conexión use los comandos separados:

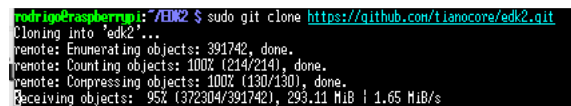
```
git clone https://github.com/tianocore/edk2.git
```

*git submodule update --init #si ocurre un problema solo siga las instrucciones en pantalla*



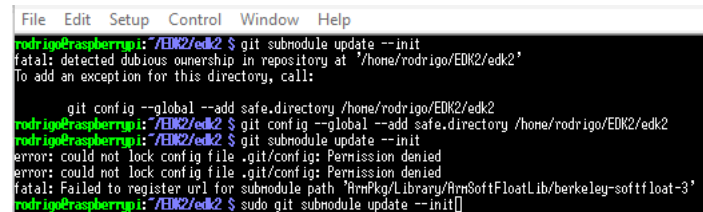
```
raspberrypi - rodrigo@raspberrypi: ~/EDK2 VT
File Edit Setup Control Window Help
rodrigo@raspberrypi:~/EDK2$ sudo git clone https://github.com/tianocore/edk2.git
```

Dependiendo de su conexión a Internet esta operacion tardara algún tiempo en completarse



```
rodrigo@raspberrypi:~/EDK2$ sudo git clone https://github.com/tianocore/edk2.git
Cloning into 'edk2'...
remote: Enumerating objects: 391742, done.
remote: Counting objects: 100% (214/214), done.
remote: Compressing objects: 100% (130/130), done.
Receiving objects: 95% (372304/391742), 293.11 MiB | 1.65 MiB/s
```

*Nota: en caso de algun problema siga las intrucciones que le indican*



```
File Edit Setup Control Window Help
rodrigo@raspberrypi:~/EDK2/edk2$ git submodule update --init
fatal: detected dubious ownership in repository at '/home/rodrigo/EDK2/edk2'
To add an exception for this directory, call:

    git config --global --add safe.directory /home/rodrigo/EDK2/edk2
rodrigo@raspberrypi:~/EDK2/edk2$ git config --global --add safe.directory /home/rodrigo/EDK2/edk2
rodrigo@raspberrypi:~/EDK2/edk2$ git submodule update --init
error: could not lock config file .git/config: Permission denied
error: could not lock config file .git/config: Permission denied
fatal: Failed to register url for submodule path 'ArmPkg/Library/ArmSoftFloatLib/berkeley-softfloat-3'
rodrigo@raspberrypi:~/EDK2/edk2$ sudo git submodule update --init
```

Esta operacion tomara cierto tiempo en completarse si falla en algún punto simplemente re-ejecute el comando.

```

root@rodigo:~/berrypie/.EDK2/edk2 $ sudo git submodule update --init
Submodule 'SoftFloat' (https://github.com/ucb-bar/berkeley-softfloat-3.git) registered for path 'ArmPkg/Library/ArmSoftFloatLib/berkeley-softfloat-3'
Submodule 'BaseTools/Source/C/BrotliCompress/brotli' (https://github.com/google/brotli) registered for path 'BaseTools/Source/C/BrotliCompress/brotli'
Submodule 'CryptoPkg/Library/HbedtlsLib/hbedtls' (https://github.com/Hackplayers/hbedtls) registered for path 'CryptoPkg/Library/HbedtlsLib/hbedtls'
Submodule 'CryptoPkg/Library/OpenSSLLib/openssl' (https://github.com/openssl/openssl) registered for path 'CryptoPkg/Library/OpenSSLLib/openssl'
Submodule 'MdeModulePkg/Library/BrotliCustomDecompressLib/brotli' (https://github.com/google/brotli) registered for path 'MdeModulePkg/Library/BrotliCustomDecompressLib/brotli'
Submodule 'MdeModulePkg/Universal/RegularExpressionDxe/ognimura' (https://github.com/kkos/ognimura) registered for path 'MdeModulePkg/Universal/RegularExpressionDxe/ognimura'
Submodule 'MdePkg/Library/BaseFdtLib/libfdt' (https://github.com/device-tree-org/libfdt.git) registered for path 'MdePkg/Library/BaseFdtLib/libfdt'
Submodule 'MdePkg/Library/MipiSysTlb/nipist' (https://github.com/NIP-II-Flance/public-mipi-sust-git) registered for path 'MdePkg/Library/MipiSysTlb/nipist'
Submodule 'RedfishPkg/Library/JsonLib/jansson' (https://github.com/akheron/jansson) registered for path 'RedfishPkg/Library/JsonLib/jansson'
Submodule 'SecurityPkg/DeviceSecurity/SpdmLib/libspdm' (https://github.com/DMTF/libspdm.git) registered for path 'SecurityPkg/DeviceSecurity/SpdmLib/libspdm'
Submodule 'UnitTestFrameworkPkg/Library/Chockalib/cnocka' (https://github.com/1ianocore/edk2-cnocka.git) registered for path 'UnitTestFrameworkPkg/Library/Chockalib/cnocka'
Submodule 'UnitTestFrameworkPkg/Library/GoogleTestLib/googletest' (https://github.com/google/googletest.git) registered for path 'UnitTestFrameworkPkg/Library/GoogleTestLib/googletest'
Submodule 'UnitTestFrameworkPkg/Library/SubhookLib/subhook' (https://github.com/xeex/subhook.git) registered for path 'UnitTestFrameworkPkg/Library/SubhookLib/subhook'
Cloning into '/home/rodigo/EDK2/edk2/ArmPkg/Library/ArmSoftFloatLib/berkeley-softfloat-3'...
Cloning into '/home/rodigo/EDK2/edk2/BaseTools/Source/C/BrotliCompress/brotli'...

```

Tras completar ejecute el segundo comando para actualizar el repositorio, sera necesario otorgar permisos completos a todos los grupos a la carpeta recién creada (edk2):

```
sudo chmod -R 777 /home/rodrigo/EDK2/edk2/
```

```
rodri@rodriaspberry:~/E0K2$ ls -l
total 4
drwxr-xr-x 36 root root 4096 Jul 12 14:13 edk2
rodri@rodriaspberry:~/E0K2$ sudo chmod -R 777 /home/rodriego/E0K2/edk2/
rodri@rodriaspberry:~/E0K2$ ls -l
total 4
drwxr-xr-x 36 root root 4096 Jul 12 14:13 edk2
rodri@rodriaspberry:~/E0K2$
```

Esto permitira que los diferentes scripts puedan terminar la configuración sin reestriccion alguna. Ingrese al directorio que se genero tras clonar el repositorio:

```
cd ./edk2
```

```
File Edit Setup Control Window Help
root@raspberrypi:~/edk2 $ cd edk2/
root@raspberrypi:~/edk2 $ ls
binutils      BaseTools      CryptlibPkg    edksetup.sh    FatPkg          IntelFsp2PkrpPkg  Maintainers.txt  Networking      PkgPkg         pip-requirements.txt  RedfishPkg      SignedCapsulePkg  UefiAppPkg
binPlatformPkg  Conf           DynamicLibrariesPkg  EmbeddedPkg    EmulatorPkg     IntelFsp2Pkg     License-History.txt  ModulePkg       OvfPkg         PkgPkg           Readme.rst       SecurityPkg       SourceLevelDebugPkg  UefiAppLoading
binUefiPkg      CONTRIBUTING.md  EdkUefiPkg     EmulatorPkg     EmulatorPkg     IntelFsp2Pkg     License.txt         ModulePkg       PdfChipsPkg    PkgPkg           Readme.rst       SecurityPkg       StandalonePkg       UnitTestFrameworkPkg
root@raspberrypi:~/edk2 $
```

Y ejecute el comando (observe que la carpeta esta llena de los archivos del proyecto):

```
sudo ./edksetup.sh
```

```
rod@rod:~/edk2$ ./edk2 $ sudo ./edksetup.sh
Using EDK2 source-base BaseTools
WORKSPACE: /home/rod/edk2/edk2
EDK_TOOLS_PATH: /home/rod/edk2/edk2/BaseTools
CONF_PATH: /home/rod/edk2/edk2/Conf
Copying SEC_TOOLS_PATH/Conf/build.rule.template
to /home/rod/edk2/edk2/Conf/build.rule.txt
Copying SEC_TOOLS_PATH/Conf/tools.def.template
to /home/rod/edk2/edk2/Conf/tools.def.txt
Copying SEC_TOOLS_PATH/Conf/target.template
to /home/rod/edk2/edk2/Conf/target.txt
rod@rod:~/edk2$
```

Para asegurar que la configuración de los directorios de trabajo es la correcta, confirmada la configuración ejecute el comando siguiente para crear las herramientas básicas de compilación:

```
sudo make -C BaseTools/
```

```

nvidia@rodrigo-vm:~/lib$ cd /etc/crontab; sudo make -C BaseTools/
make: Entering directory '/home/rodrigo/EUR2/edk2/BaseTools/'
make -C Source/C
make[1]: Entering directory '/home/rodrigo/EUR2/edk2/BaseTools/Source/C'
Attempting to detect HOST_ARCH from "uname -m": xarch64
Detected HOST_ARCH of ARMCH64 using uname.
mkdir -p lib
nvidia ~/lib
make -C Common
make[1]: Entering directory '/home/rodrigo/EUR2/edk2/BaseTools/Source/C/Common'
gcc -c -I../.. -I./Include/Common -I./Include/IndustryStandard -I./Common -I.. -I../.. -I../.. -I../MdePkg/Include/HardArch64 -I../.. -I../.. -I../MdePkg/Include -fshort-uchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wdeprecated-declarations -Wno-string-truncation -Wno-restrict -Wno-unused-result -nostdlib -x80 -O2 -BasePlatform -C BasePlatform.o
nvidia -o lib/libcommon.a ./libcommon.o
nvidia -r ../MdePkg/Include/HardArch64 -I../.. -I../.. -I../MdePkg/Include -fshort-uchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wdeprecated-declarations -Wno-string-truncation -Wno-restrict -Wno-unused-result -nostdlib -x80 -O2 -BasePlatform -C BaseFunctions.o
nvidia -o lib/libcommon.a ./libcommon.o
nvidia -r ../MdePkg/Include/HardArch64 -I../.. -I../.. -I../MdePkg/Include -fshort-uchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wdeprecated-declarations -Wno-string-truncation -Wno-restrict -Wno-unused-result -nostdlib -x80 -O2 -Ccrc32 -C crc32.o
nvidia -o lib/libcommon.a ./libcommon.o
nvidia -r ../MdePkg/Include/HardArch64 -I../.. -I../.. -I../MdePkg/Include -fshort-uchar -fno-strict-aliasing -fwrapv -fno-delete-null-pointer-checks -Wall -Werror -Wdeprecated-declarations -Wno-string-truncation -Wno-restrict -Wno-unused-result -nostdlib -x80 -BiosSupport -C BiosSupport.o

```

La operacion tomara tiempo en completarse

```
File Edit Setup Control Window Help
test_build_init (CheckPythonSyntax.Tests) ... ok
test_build_build (CheckPythonSyntax.Tests) ... ok
test_build_buildoptions (CheckPythonSyntax.Tests) ... ok
test_sitecustomize (CheckPythonSyntax.Tests) ... ok
test_tests_split_test_split (CheckPythonSyntax.Tests) ... ok
test32bitUnicodeCharInUtf8Comment (CheckUnicodeSourceFiles.Tests) ... ok
test32bitUnicodeCharInUtf8File (CheckUnicodeSourceFiles.Tests) ... ok
testSupplementaryPlaneUnicodeCharInUtf16File (CheckUnicodeSourceFiles.Tests) ... ok
testSurrogatePairUnicodeCharInUtf16File (CheckUnicodeSourceFiles.Tests) ... ok
testSurrogatePairUnicodeCharInUtf8File (CheckUnicodeSourceFiles.Tests) ... ok
testSurrogatePairUnicodeCharInUtf8FileWithBom (CheckUnicodeSourceFiles.Tests) ... ok
testUtf16InUtf8File (CheckUnicodeSourceFiles.Tests) ... ok
testValidUtf8File (CheckUnicodeSourceFiles.Tests) ... ok
testValidUtf8FileWithBom (CheckUnicodeSourceFiles.Tests) ... ok

-----
Ran 303 tests in 8.753s

OK
make[1]: Leaving directory '/home/rodrigo/EDK2/edk2/BaseTools/Tests'
make: Leaving directory '/home/rodrigo/EDK2/edk2/BaseTools'
rodrigo@raspberrypi:~/EDK2/edk2$
```

Observe que no haya habido errores durante la ejecución, encaso de existir vuelva e ejecutar el comando y preste atención a los mensajes de error.

Ahora realicemos los ajustes en el archivo de configuracion para indicar que tipo de archivo EDK2 UEFI de arranque queremos crear, utilice el comando:

`sudo vi Conf/target.txt`

```
rodrigo@raspberrypi:~/EDK2/edk2$ sudo vi Conf/target.txt
```

En el archivo encuentre las variables que se muestran en la siguiente tabla:

ACTIVE\_PLATFORM = ArmVirtPkg/ArmVirtQemu.dsc

TARGET = DEBUG

TARGET\_ARCH = AARCH64

TOOL\_CHAIN\_TAG = GCC5

Modifique los valores en caso de ser necesario:

```
# Copyright (c) 2006 - 2019, Intel Corporation. All rights reserved.<BR>
# SPDX-License-Identifier: BSD-2-Clause-Patent
#
# ALL Paths are Relative to WORKSPACE
#
# Separate multiple LIST entries with a SINGLE SPACE character, do not use comma characters.
# Un-set an option by either commenting out the line, or not setting a value.
#
# PROPERTY      Type      Use      Description
# -----
# ACTIVE_PLATFORM  Filename  Recommended Specify the WORKSPACE relative Path and Filename
#                                     of the platform description file that will be used for the
#                                     build. This line is required if and only if the current
#                                     working directory does not contain one or more description
#                                     files.
ACTIVE_PLATFORM = EmulatorPkg/EmulatorPkg.dsc ← (A)
"Conf/target.txt" [dos] 70L, 4828B
```

La figura en (A) muestra el valor de configuración de ACTIVE\_PLATFORM, en este caso debe cambiarse al valor anteriormente indicado (en este caso comentamos la línea y sustituimos con una que tiene el valor requerido).

```

rodrigo@raspberrypi:~/EDK2/edk2 $ cat Conf/target.txt
#
# Copyright (c) 2006 - 2019, Intel Corporation. All rights reserved.<BR>
#
# SPDX-License-Identifier: BSD-2-Clause-Patent
#
#
# ALL Paths are Relative to WORKSPACE
#
# Separate multiple LIST entries with a SINGLE SPACE character, do not use comma characters.
# Un-set an option by either commenting out the line, or not setting a value.
#
#-----
# PROPERTY      Type      Use      Description
#-----
# ACTIVE_PLATFORM  Filename  Recommended  Specify the WORKSPACE relative Path and Filename
#                  of the platform description file that will be used for the
#                  build. This line is required if and only if the current
#                  working directory does not contain one or more description
#                  files.
#ACTIVE_PLATFORM  = EmulatorPkg/EmulatorPkg.dsc
#ACTIVE_PLATFORM  = ArmVirtPkg/ArmVirtQemu.dsc

```

Tras realizar la misma accion en todas las lineas requeridas, volvemos a ejecutar el comando de configuracion:

```

rodrigo@raspberrypi:~/EDK2/edk2 $ ./edksetup.sh
Loading previous configuration from /home/rodriego/EDK2/edk2/Conf/BuildEnv.sh
Using EDK2 in-source Basetools
WORKSPACE: /home/rodriego/EDK2/edk2
EDK_TOOLS_PATH: /home/rodriego/EDK2/edk2/BaseTools
CONF_PATH: /home/rodriego/EDK2/edk2/Conf
rodrigo@raspberrypi:~/EDK2/edk2 $

```

Seguido de el comando "Build" (el cual es de python) para comenzar la creación de nuestro archivo de EDK2-BIOS para nuestra Maquina Virtual.

```

File Edit Setup Control Window Help
rodrigo@raspberrypi:~/EDK2/edk2 $ build
Build environment: Linux-6.1.21-v8+-aarch64-with-glibc2.31
Build start time: 15:34:26, Jul.12 2024

WORKSPACE      = /home/rodriego/EDK2/edk2
EDK_TOOLS_PATH = /home/rodriego/EDK2/edk2/BaseTools
CONF_PATH      = /home/rodriego/EDK2/edk2/Conf
PYTHON_COMMAND = python3

Processing meta-data
..Architecture(s) = AARCH64
Build target      = DEBUG
Toolchain         = GCC5

Active Platform   = /home/rodriego/EDK2/edk2/ArmVirtPkg/ArmVirtQemu.dsc
.....

```

Comenzara el proceso de compilación de nuestro archivo indicado por la "barra de progreso", este proceso puede tardar un momento el cual dependera de las capacidades de computo de nuestro hardware.

```

File Edit Setup Control Window Help
Generate Region at Offset 0x40000
Region Size = 0x40000
Region Name = DATA

Generate Region at Offset 0x80000
Region Size = 0x40000
Region Name = None

GUID cross reference file can be found at /home/rodriego/EDK2/edk2/Build/ArmVirtQemu-AARCH64/DEBUG_GCC5/FV/Guid.xref

FV Space Information
FVMAIN [992Full] 6823680 (0x681f00) total, 6823656 (0x681ee8) used, 24 (0x18) free
FVMAIN_COMPACT [572Full] 2093056 (0x1ff000) total, 1193184 (0x1234e0) used, 899872 (0xd8bb20) free

- Done -
Build end time: 15:46:21, Jul.12 2024
Build total time: 00:11:54

rodrigo@raspberrypi:~/EDK2/edk2 $ ls Build/ArmVirtQemu-AARCH64/DEBUG_GCC5/FV/QEMU_EFI.fd
Build/ArmVirtQemu-AARCH64/DEBUG_GCC5/FV/QEMU_EFI.fd
rodrigo@raspberrypi:~/EDK2/edk2 $

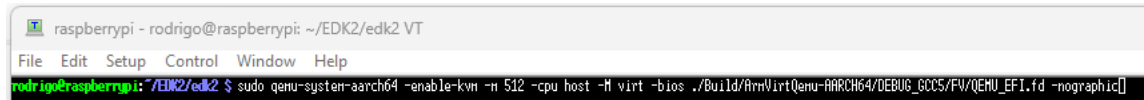
```

Cuando la operación termine pude ubicar el archivo usando:

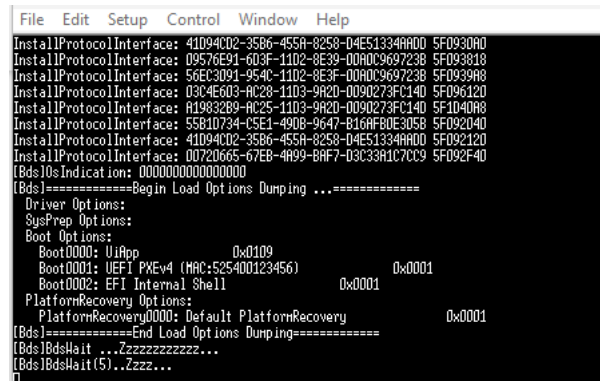
*ls Build/ArmVirtQemu-AARCH64/DEBUG\_GCC5/FV/QEMU\_EFI.fd*

Puede ejecutar el archivo en la maquina virtual de Qemu, con el siguiente comando:

```
sudo qemu-system-aarch64 -enable-kvm -m 512 -cpu host -M virt -bios
./Build/ArmVirtQemu-AARCH64/DEBUG_GCC5/FV/QEMU_EFI.fd -nographic
```



Cuando termine la ejecucion veremos la salida que nos mostraria el puerto de Debug de nuestra Maquina Virtual.

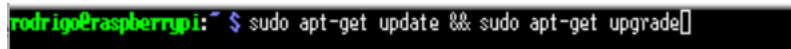


Hemos completado la creaci3n de un IFWI-BIOS para nuestra maquina virtual

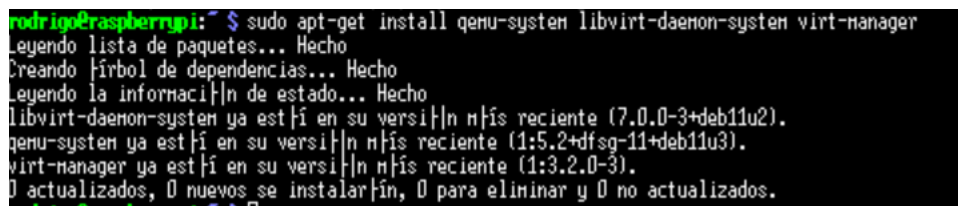
## APENDICE B

### Instalacion del software de virtualizaci3n Qemu

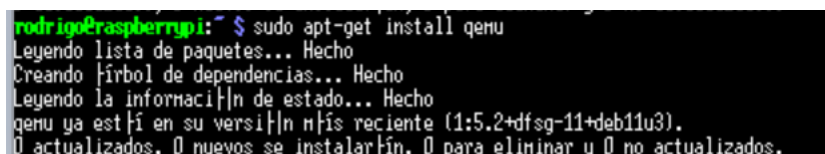
Antes de realizar acci3n alguna asegure que su sistema operativo esta actualizado:



Posteriormente descargue e instale los paquetes requeridos para el funcionamiento correcto del software de virtualizacion:



Los tres paquetes basicos son "qemu-system", "libvirt-daemon-system" y "virt-manager", asi mismo si requiere instale el paquete de Qemu:



Tras haber instalado los paquetes de de alta su usuario del sistema para que pueda acceder a los servicios de virtualizacion:



```
rodrigo@raspberrypi:~$ sudo adduser rodrigo libvirt
El usuario 'rodrigo' ya es un miembro de 'libvirt'.
```

Sustituya su nombre de usuario en la línea de comando anterior, y para que los programas de segundo plano del software virtual se activen reinicie el sistema:

```
rodrigo@raspberrypi:~$ sudo reboot
```

Cuando el sistema vuelva a iniciar, compruebe que funciona con la siguiente línea de comandos:

```
rodrigo@raspberrypi:~$ sudo qemu-system-x86_64 -bios OVMF.fd -m 256M -net none -nographic
```

Esto inicializa una sesión virtual del UEFI-Shell:

```
PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
Shell> help reset
Resets the system.

RESET [-u [string]]
RESET [-s [string]]
RESET [-c [string]]

-s - Performs a shutdown.
-u - Performs a warm boot.
-c - Performs a cold boot.
string - Describes a reason for the reset.

NOTES:
1. This command resets the system.
2. The default is to perform a cold reset unless the -u parameter is
   specified.
3. If a reset string is specified, it is passed into the Reset()
   function, and the system records the reason for the system reset.
Shell> reset -s
Reset with <null string> (0 bytes)rodrigo@raspberrypi:~$
```

Para terminar la sesión use la orden "reset -s" en el prompt del UEFI-Shell y regresará al Prompt de la raspberry pi.

## APENDICE C

### Compilación de módulos EDK2 para arquitectura X86\_64 usando una raspberry pi

Tras haber completado las instrucciones del **Apéndice A**, y verificar que contamos con un ambiente de desarrollo funcional (al menos para Arquitectura AARCH64=ARM), debemos realizar unos ajustes adicionales a las herramientas del paquete EDK2 para poder compilar para arquitectura x86\_64, lo primero es instalar la familia de compilación GCC requerida por las herramientas:

```
Archivo  Editar  Pestañas  Ayuda
rodrigo@raspberrypi:~$ sudo apt-get install gcc-x86-64-linux-gnu
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
gcc-x86-64-linux-gnu ya está en su versión más reciente (4:10.2.1-1).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
rodrigo@raspberrypi:~$
```

Tras instalar el paquete del compilador, verificamos la ruta en la cual fue instalado el compilador:

```
rodrigo@raspberrypi:~ $ which x86_64-linux-gnu-gcc
/usr/bin/x86_64-linux-gnu-gcc
rodrigo@raspberrypi:~ $
```

Copiamos esta dirección para modificar el archivo de configuración ubicado en la carpeta *Conf* de nuestro ambiente de desarrollo:

```
rodrigo@raspberrypi:~/src/edk2/Conf $ ls
BuildEnv.sh  build_rule.txt  ReadMe.txt  target.txt  tools_def.txt
rodrigo@raspberrypi:~/src/edk2/Conf $ sudo vim tools_def.txt
```

Usamos el editor de nuestra preferencia, (en este caso Vim con la opción de números de línea):

```
85 DEFINE CYGWIN_BINX64      = c:/cygwin/opt/tiano/x86_64-pc-mingw64/x86_64-pc-mingw64/bin/
86
87 DEFINE GCC48_IA32_PREFIX  = ENV(GCC48_BIN)
88 DEFINE GCC48_X64_PREFIX   = ENV(GCC48_BIN)
89
90 DEFINE GCC49_IA32_PREFIX  = ENV(GCC49_BIN)
91 DEFINE GCC49_X64_PREFIX   = ENV(GCC49_BIN)
92
93 DEFINE GCCNOLTO_IA32_PREFIX = ENV(GCCNOLTO_BIN)
94 DEFINE GCCNOLTO_X64_PREFIX = ENV(GCCNOLTO_BIN)
95 #DEFINE GCC5_IA32_PREFIX   = ENV(GCC5_BIN)
96 DEFINE GCC5_IA32_PREFIX    = /usr/bin/x86_64-linux-gnu- (B)
97 #DEFINE GCC5_X64_PREFIX    = ENV(GCC5_BIN)
98 DEFINE GCC5_X64_PREFIX     = /usr/bin/x86_64-linux-gnu- (B)
99 DEFINE GCC_IA32_PREFIX     = ENV(GCC_BIN)
100 DEFINE GCC_X64_PREFIX      = ENV(GCC_BIN)
101 DEFINE GCC_HOST_PREFIX     = ENV(GCC_HOST_BIN)
102
103 DEFINE UNIX_IASL_BIN       = ENV(IASL_PREFIX)iasl
104 DEFINE WIN_IASL_BIN        = ENV(IASL_PREFIX)iasl.exe
105
106 DEFINE IASL_FLAGS          =
-- VISUAL --
```

Copie las líneas que se muestran en (A) y las "originales" las puede borrar o comentar, sustituya las nuevas líneas con la información mostrada (B), la cual es la dirección donde está instalado el compilador.

Cuando compile para arquitectura X64 o IA32, usando el script de python *build* ajuste el parámetro *-t GCC5* (tool chain GCC5), ya que fue la variable de ambiente para el compilador que modificamos. Por ejemplo, compilar OVMF.fd (archivo de Qemu) usaremos:

```
rodrigo@raspberrypi:~/src/edk2 $ build -a X64 -t GCC5 -b DEBUG -p MdeModulePkg/MdeModulePkg.dsc
```

Tras termine el proceso de compilación deberemos ver el mensaje de compilación terminada sin errores:

```
- Done -
Build end time: 21:56:10, Aug.13 2024
Build total time: 00:01:19
rodrigo@raspberrypi:~/src/edk2 $
```

## APENDICE D

### Configuración de un ambiente de Desarrollo para el proyecto Coreboot.

Este apartado describe los pasos requeridos para la instalación de un ambiente de desarrollo de un BIOS usando Coreboot, las instrucciones originales se encuentran en la dirección:

<https://doc.coreboot.org/tutorial/part1.html>

Debido a que el sistema en el cual se "montara" la instalación es una raspberry pi modelo 3b+, sera necesario realizar algunos ajustes a las indicaciones originales,.

Actualice el sistema primero:

```
rodrigo@raspberrypi:~$ sudo apt-get update[]
```

```
rodrigo@raspberrypi:~$ sudo apt-get upgrade[]
```

Y posteriormente descarge el conjunto de herramientas necesarias para la configuración:

```
rodrigo@raspberrypi:~$ sudo apt-get install -y bison build-essential curl flex git gnat libncurses5-dev libssl-dev m4 zlib1g-dev pkg-config
```

```
sudo apt-get install -y bison build-essential curl flex git gnat libncurses5-dev \
libssl-dev m4 zlib1g-dev pkg-config
```

La linea anterior descarga las herramientas que se requeriran durante las etapas del proceso de compilación del Bios. Lo siguiente es clonar el arbol de recursos:

```
git clone --recurse -submodules https://review.coreboot.org/coreboot.git
```

```
~$ sudo git clone --recurse -submodules https://review.coreboot.org/coreboot.git
```

Acorde a las instrucciones de la pagina web del proyecto, nos pide que compilemos la "tool chain", los cuales son el grupo de complidores enlazadores y cargadores, para compilar el BIOS:

1) hay que ingresar a la carpeta de coreboot creada anteriormente:

```
cd coreboot
```

2) usar el comando make junto con el parametro crossgcc, elegiremos la arquitectra x86:

```
sudo make crossgcc-i386
```

**Este proceso es largo y propenso a errores, por lo que podrian requerirse varios intentos.**

Para evitar este inconveniente es mejor usar un contendor Docker con las herramientas ya compiladas de antemano, para eso primero debemos instalar el software en nuestra raspberry pi:

```
curl -sSL https://get.docker.com | sh
```

```
$ curl -sSL https://get.docker.com | sh
```

Terminada la instalación debemos otorgar permisos a nuestro usuario para que pueda ingresar a la ejecución del contenedor.

```
sudo usermod -aG docker <usuario>
```

```
rodrigo@raspberrypi:~ $ sudo usermod -aG docker rodrigo
```

Nota: en caso de que necesite desinstalar Docker realice lo siguiente:

-Liste los archivos instalados: `dpkg -l | grep -i docker`

-Elimine los archivos agregandolos en linea tras el siguiente comando: `sudo apt-get purge -y`

`docker-buildx-plugin docker-ce ....`

-Por ultimo eliminelos directorios que quedan tras la instalación:

```
sudo rm -rf /var/lib/docker/etc/docker
```

```
sudo rm -rf /etc/apparmor.d/docker
```

```
sudo groupdel docker
```

```
sudo rm -rf /var/run/docker.sock
```

```
sudo rm -rf /var/lib/container
```

Si la instalación de Docker ocurrio sin problema, debemos descargar el contenedor con las herramientas ya compiladas:

```
sudo docker pull coreboot/coreboot-sdk
```

```
~ $ sudo docker pull coreboot/coreboot-sdk
```

Este contenedor es descargado del repositorio:

<https://hub.docker.com/r/coreboot/coreboot-sdk>

Este contenedor solo puede ejecutarse en un ambiente x86, debido a que nuestra Raspberry Pi usa ARM, necesitamos descargar un contenedor adicional:

```
sudo docker pull tonistiigi/binfmt
```

```
~ $ sudo docker pull tonistiigi/binfmt
```

<https://hub.docker.com/r/tonistiigi/binfmt>

tras la descarga debemos ejecutar este contenedor Prior a la ejecucion del contenedor de Coreboot, nota asegura de contar con el software de virtualizacion Qemu:

```
sudo apt-get install qemu
```

```
docker run --privileged --rm tonstigi/binfmt --install all
```

```
~/coreboot $ sudo docker run --privileged --rm tonistiigi/binfmt --install all
```

Ahora podemos ejecutar el contenedor con las herramientas de compilación del proyecto Coreboot:

`cd coreboot #asegure de estar siempre en la carpeta del proyecto la cual clonamos #de git`

`sudo docker run -w /home/coreboot/coreboot -u root -it -v $PWD:/home/coreboot/coreboot --rm --platform linux/amd64 coreboot/coreboot-sdk /bin/bash`

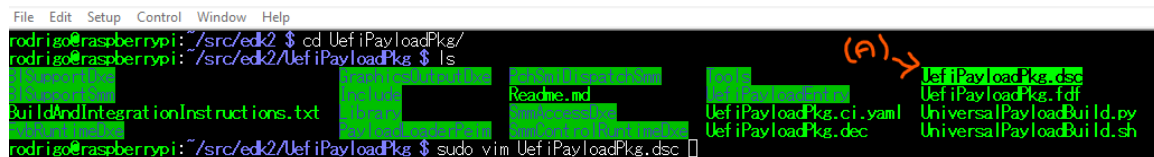
```
rodri@raspberrypi:~$ cd coreboot/
rodri@raspberrypi:~/coreboot$ sudo docker run -u /home/coreboot/coreboot -u root -it -v $PWD:/home/coreboot/coreboot --rm --platform linux/amd64 coreboot/coreboot-sdk /bin/bash
root@235918de96d7:/home/coreboot/coreboot#
```

Observe como el prompt cambia de color y de nombre, indicando que estamos dentro del contenedor.

## APENDICE E

### Proceso de compilacion de la payload EDK2.

Asegure de contar con un ambiente de desarrollo EDKII funcional antes de continuar con estas instrucciones, dentro de la carpeta del proyecto edk2 ingrese al directorio `UefiPayloadPkg`, y localice la carpeta con el nombre `UefiPayloadPkg.dsc` (B).



```
File Edit Setup Control Window Help
rodri@raspberrypi:~/src/edk2$ cd UefiPayloadPkg/
rodri@raspberrypi:~/src/edk2/UefiPayloadPkg$ ls
BuildAndIntegrationInstructions.txt  Readme.md  UefiPayloadPkg.ci.yaml  UefiPayloadPkg.dsc  UefiPayloadPkg.fdf  UniversalPayloadBuild.py  UniversalPayloadBuild.sh
rodri@raspberrypi:~/src/edk2/UefiPayloadPkg$ sudo vim UefiPayloadPkg.dsc
```

Abra el archivo y modifique la linea que se indica (numero 25) a que sea tal como se muestra en la siguiente figura:



```
1 ## @file
2 # Bootloader Payload Package
3
4 # Provides drivers and definitions to create uefi payload for bootloaders.
5 #
6 # Copyright (c) 2014 - 2023, Intel Corporation. All rights reserved.<BR>
7 # Copyright (c) Microsoft Corporation.
8 # SPDX-License-Identifier: BSD-2-Clause-Patent
9 #
10 ##
11
12 #####
13 #
14 # Defines Section - statements that will be processed to create a Makefile.
15 #
16 #####
17 [Defines]
18 PLATFORM_NAME                = UefiPayloadPkg
19 PLATFORM_GUID                = F71608AB-D63D-4491-B744-A99998C8CD96
20 PLATFORM_VERSION              = 0.1
21 DSC_SPECIFICATION             = 0x00010005
22 SUPPORTED_ARCHITECTURES      = IA32|x64
23 BUILD_TARGETS                 = DEBUG|RELEASE|NOOPT
24 SKUID_IDENTIFIER              = DEFAULT
25 OUTPUT_DIRECTORY              = Build/UefiPayloadPkg
26 FLASH_DEFINITION              = UefiPayloadPkg/UefiPayloadPkg.fdf
27 PCD_DYNAMIC_AS_DYNAMICEX     = TRUE
28
29 DEFINE SOURCE_DEBUG_ENABLE    = FALSE
30 DEFINE PS2_KEYBOARD_ENABLE    = FALSE
31 DEFINE RAM_DISK_ENABLE        = FALSE
32 DEFINE SIO_BUS_ENABLE          = FALSE
33
34 -- VISUAL --
```

El valor que debe tener es `OUTPUT_DIRECTORY=Build/UefiPayloadPkg`, guarde los cambios en el archivo al salir.

Regrese al directorio principal de EDKII , ejecute el script de configuracion `./edksetup.sh` y ejecute la

siguiente linea de compilación:

```
build -a IA32 -a X64 -p UefiPayloadPkg/UefiPayloadPkg.dsc -b DEBUG -t GCC5 -D  
BOOTLOADER=COREBOOT
```

```
rodrigo@raspberrypi: /src/edk2/UefiPayloadPkg $ cd ..  
rodrigo@raspberrypi: /src/edk2 $ ./edksetup.sh  
Loading previous configuration from /home/rodrigo/src/edk2/Conf/BuildEnv.sh  
Using EDK2 in-source Basetools  
WORKSPACE: /home/rodrigo/src/edk2  
EDK_TOOLS_PATH: /home/rodrigo/src/edk2/BaseTools  
CONF_PATH: /home/rodrigo/src/edk2/Conf  
rodrigo@raspberrypi: /src/edk2 $ build -a IA32 -a X64 -p UefiPayloadPkg/UefiPayloadPkg.dsc -b DEBUG -t GCC5 -D BOOTLOADER=COREBOOT
```

La compilacion comenzara a ejecutarse, esto tomara algo de tiempo:

```
E-4f1c-AD65-E05268D0B4D1Shell/7C04A583-9E3E-4f1c-AD65-E05268D0B4D1.ffs -oi /home/rodrigo/src/edk2/Build/UefiPayloadPkg/DEBUG_GCC5/FV/Ffs/7C04A583-9E3E-4f1c-AD65-E05268D0B4D1Shell/7C04A583-9E3E-4f1c-AD65-E05268D0B4D1SEC1.1.pe32 -oi /home/rodrigo/src/edk2/Build/UefiPayloadPkg/DEBUG_GCC5/FV/Ffs/7C04A583-9E3E-4f1c-AD65-E05268D0B4D1Shell/ShellOffset.raw -oi /home/rodrigo/src/edk2/Build/UefiPayloadPkg/DEBUG_GCC5/FV/Ffs/7C04A583-9E3E-4f1c-AD65-E05268D0B4D1Shell/7C04A583-9E3E-4f1c-AD65-E05268D0B4D1SEC2.ui -oi /home/rodrigo/src/edk2/Build/UefiPayloadPkg/DEBUG_GCC5/FV/Ffs/7C04A583-9E3E-4f1c-AD65-E05268D0B4D1Shell/7C04A583-9E3E-4f1c-AD65-E05268D0B4D1SEC3.ver  
Building ... /home/rodrigo/src/edk2/ModulePkg/Universal/ReportStatusCodeRouter/RuntimeDxe/ReportStatusCodeRouterRuntimeDxe.inf [X64]  
make: No se hace nada para 'tbuild'.  
make: No se hace nada para 'tbuild'.  
make: No se hace nada para 'tbuild'.  
Fd File Name:UEFIPAYLOAD (/home/rodrigo/src/edk2/Build/UefiPayloadPkg/DEBUG_GCC5/FV/UEFIPAYLOAD.fd)  
Generate Region at Offset 0x0  
Region Size = 0x590000  
Region Name = FV  
Generating PLDFV FV  
Generating DXEFV FV  
#####  
Generating DXEFV FV  
#####  
GUID cross reference file can be found at /home/rodrigo/src/edk2/Build/UefiPayloadPkg/DEBUG_GCC5/FV/Guid.xref  
FV Space Information  
PLDFV [60%Full] 5832704 (0x590000) total, 3531872 (0x35e460) used, 2300832 (0x231ba0) free  
DXEFV [99%Full] 3502080 (0x357000) total, 3498872 (0x356378) used, 3208 (0xc88) free  
- Done -  
Build end time: 21:40:30, Sep.07 2024  
Build total time: 00:01:04  
rodrigo@raspberrypi: /src/edk2 $
```

Asegure que al final termino correctamente y cuenta con el archivo *UEFIPAYLOAD.fd* en la dirección *~/src/edk2/Build/UefiPayloadPkg/DEBUG\_GCC5/FV/*

```
rodrigo@raspberrypi: /src/edk2 $ ls Build/UefiPayloadPkg/DEBUG_GCC5/FV/  
DXEFV.ext DXEFV.Fv.map DXEFV.inf GuidedSectionTools.txt PLDFV.ext PLDFV.Fv.map PLDFV.inf UEFIPAYLOAD.fd  
DXEFV.Fv DXEFV.Fv.txt Ffs Guid.xref PLDFV.Fv PLDFV.Fv.txt UEFIPAYLOAD32.fd
```

Para diferenciar el archivo renombralo a *UEFIPAYLOAD32.fd*.

**Nota:** Asegure tener un ambiente correctamente instalado y funcional a la hora de realizar la compilacion de la PayLoad.

## APENDICE F

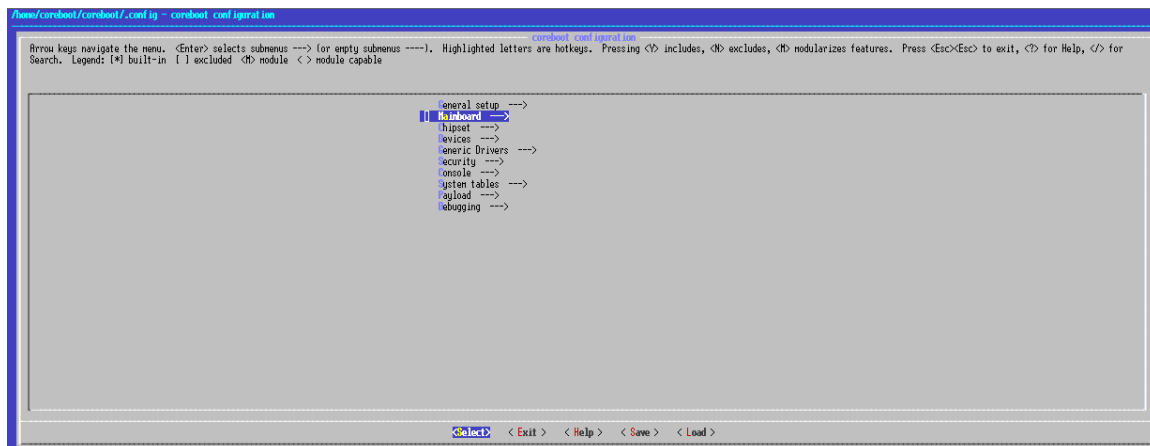
### Guia visual de configuracion de las reglas de compilacion para integrar EDK2 payload con CoreBoot.

No olvide verificar que el ambiente de desarrollo este correctamente instalado y sea funcional antes de ejecutar la compilacion (no olvide que debe ingresar al contenedor Docker), usando la instruccion *make menuconfig* ingrese al menu de reglas de compilacion:

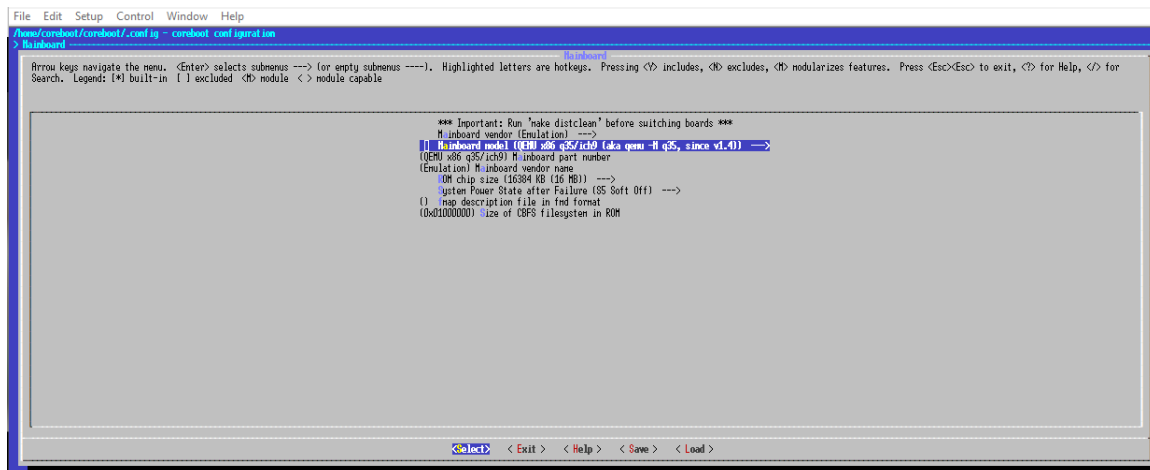
```
File Edit Setup Control Window Help  
root@235918de96d7:/home/coreboot/coreboot# make menuconfig
```



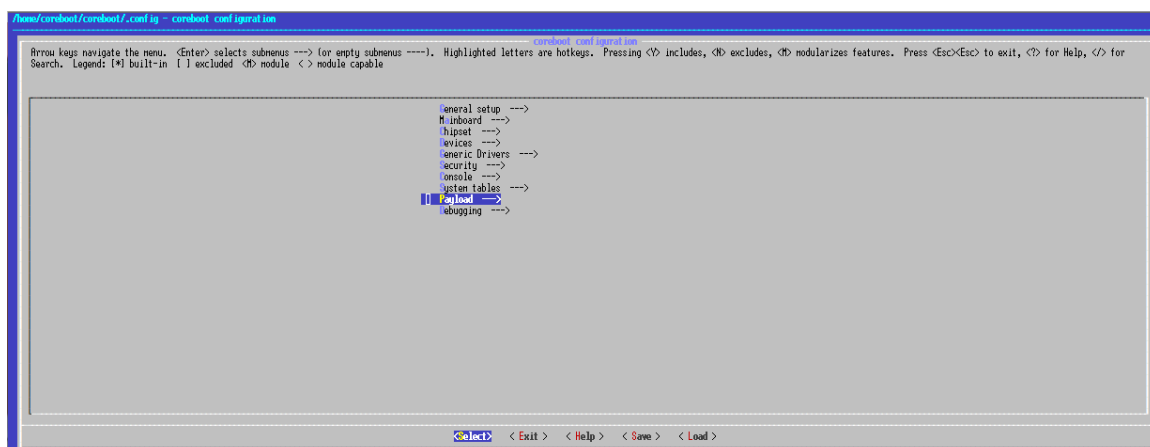
En la opción *Main Board*:



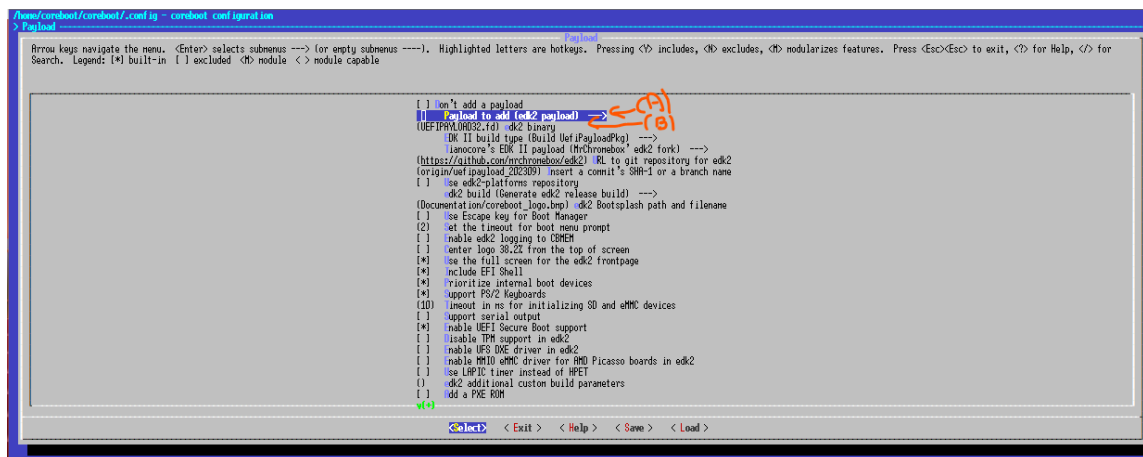
Asegure que en la opción *Mainboard Vendor* este seleccionada la opción *Emulation* y *Mainboard model* la opción entre parentesis sea *Qemu x86 q35/ich9 ... since v1.41*)



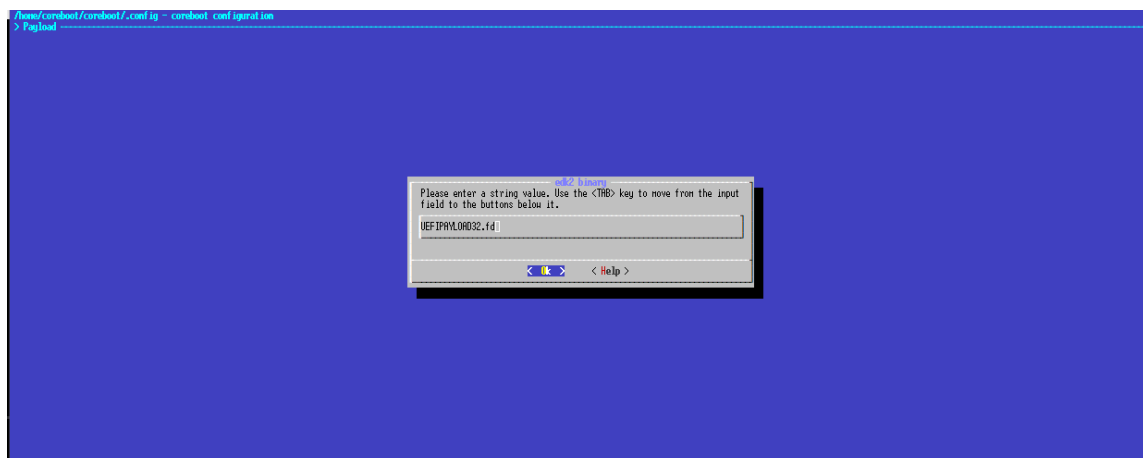
Regrese al menu principal nuevamente y vaya a la opción *Payload*



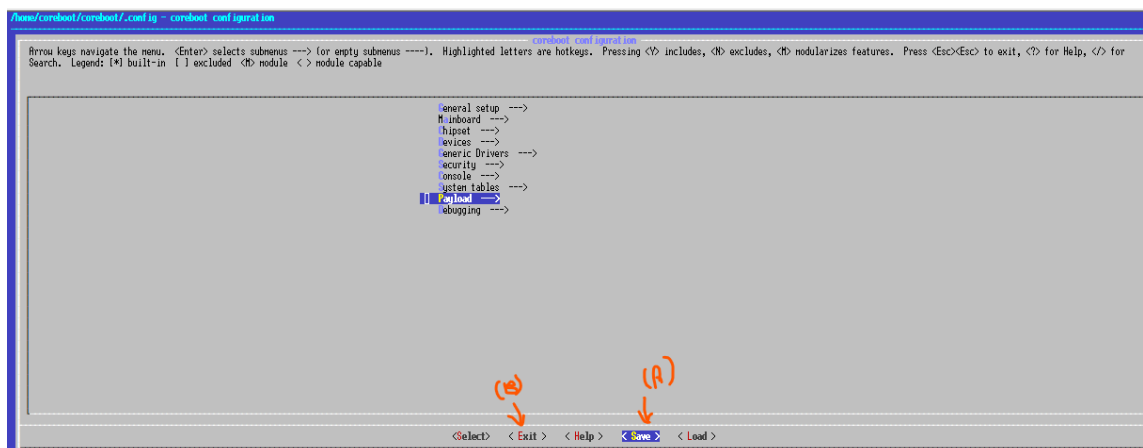
En la opción *Payload to add (edk2 payload)* este como se muestra en inciso (A)



Para la opcion (B) de la figura anterior indique en el cuadro de dialogo la ubicación del archivo de Payload (UEFI PAYLOAD32.fdt en este caso, antes UEFIPAYLOAD.fdt) que se usara, para este caso se encuentra en la parte mas "alta" del directorio del ambiente de CoreBoot, **Nota: si no desea tener el UEFI-Shell instalado desactive la casilla correspondiente.**



Todas las demas opciones se dejan tal cual, en el menu principal seleccione la opcion Save (A) y despues la opcion Exit (B):



Muy importante no modifique el nombre del archivo de configuracion donde se han guardados estos cambios a menos que sepa lo que esta haciendo. Una vez configurado puede proseguir con la compilacion del proyecto CoreBoot.