

# Sistemas Críticos

Tema 1:

Selección y configuración de un sistema operativo

Lección 3:

Construcción de un Linux empotrado desde cero



Jesús González Peñalver

# Contenidos

## Tema 1: Selección y configuración de un sistema operativo

Introducción

Fundamentos de *Linux*

Selección de la plataforma y prerequisites del sistema

Diseño de una plataforma de ejecución mínima

Construcción del *kernel* de *Linux*

Construcción del *Device Tree Blob*

Necesidad de un *Root File System*

Construcción de un *Root File System*

Generación del *First Stage Boot Loader*

Construcción de *U-Boot*

Preparación de la imagen de arranque

# Descarga y configuración por defecto del *kernel* de *Linux*

## Obtención de las fuentes

```
KERNEL="Linux-Digilent-Dev"  
DILIGENT_GIT="https://github.com/DigilentInc"  
KERNEL_DIR="${PRJ_ROOT}/${KERNEL}"  
git -C ${PRJ_ROOT} clone -b master-next ${DILIGENT_GIT}/${KERNEL}.git
```

## Fijamos el valor de las variables de entorno necesarias

```
export ARCH="arm"  
export HOST="${ARCH}-xilinx-linux-gnueabi"  
export CROSS_COMPILE="${HOST}-"
```

## Limpiamos restos de compilaciones anteriores

```
cd ${KERNEL_DIR}  
make distclean
```

## Configuramos por defecto para la plataforma

```
make xilinx_zynq_defconfig
```

# Ajuste de la configuración y construcción del *kernel* de *Linux*

## Ajuste de la configuración

```
make menuconfig
```

```
[*] Enable loadable module support
Kernel Features
    [*] Use the ARM EABI to compile the kernel
    [*] Allow old ABI binaries to run with this kernel
```

Dado que la *toolchain* usa la interfaz ARM EABI, debemos añadir al kernel el soporte para dicha interfaz

## Construimos el kernel

```
nice make -j 4
```

5m en un Quad Core a 2.4GHz y 8GB de RAM

## Lo copiamos al directorio de las imágenes

```
mkdir -p ${PRJ_ROOT}/images
cp arch/arm/boot/zImage ${PRJ_ROOT}/images
```

# Contenidos

## Tema 1: Selección y configuración de un sistema operativo

Introducción

Fundamentos de *Linux*

Selección de la plataforma y prerequisites del sistema

Diseño de una plataforma de ejecución mínima

Construcción del *kernel* de *Linux*

Construcción del *Device Tree Blob*

Necesidad de un *Root File System*

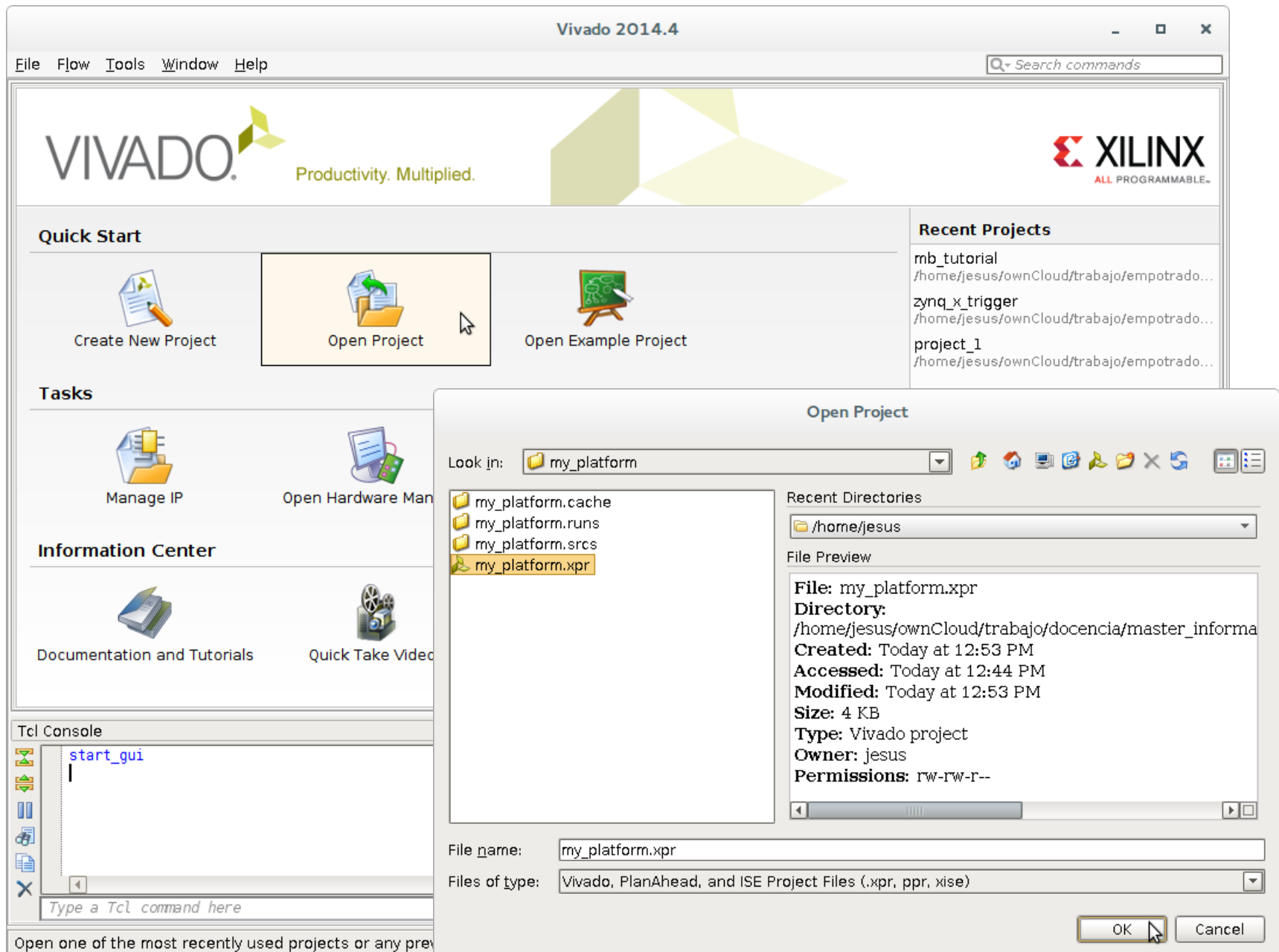
Construcción de un *Root File System*

Generación del *First Stage Boot Loader*

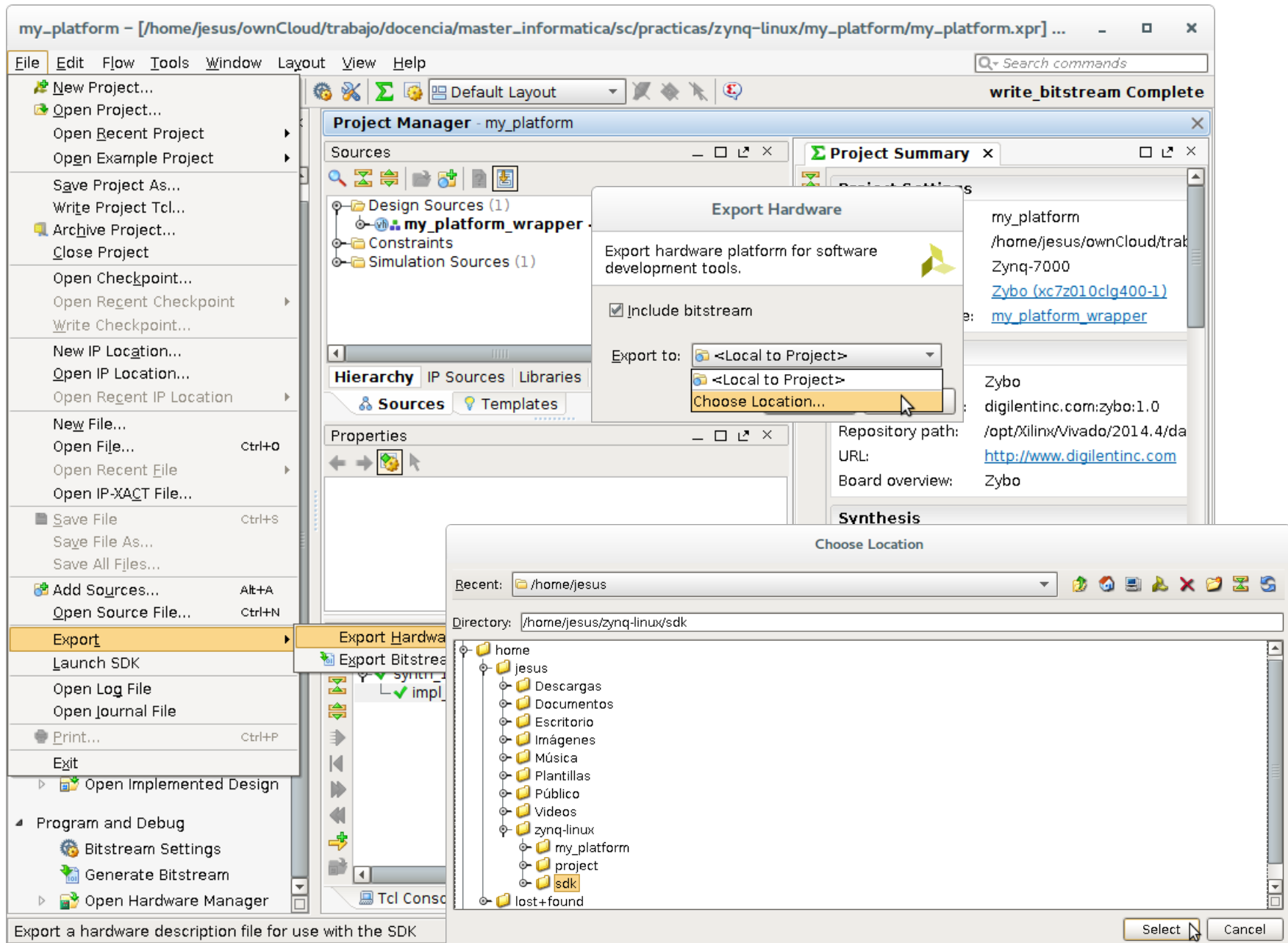
Construcción de *U-Boot*

Preparación de la imagen de arranque

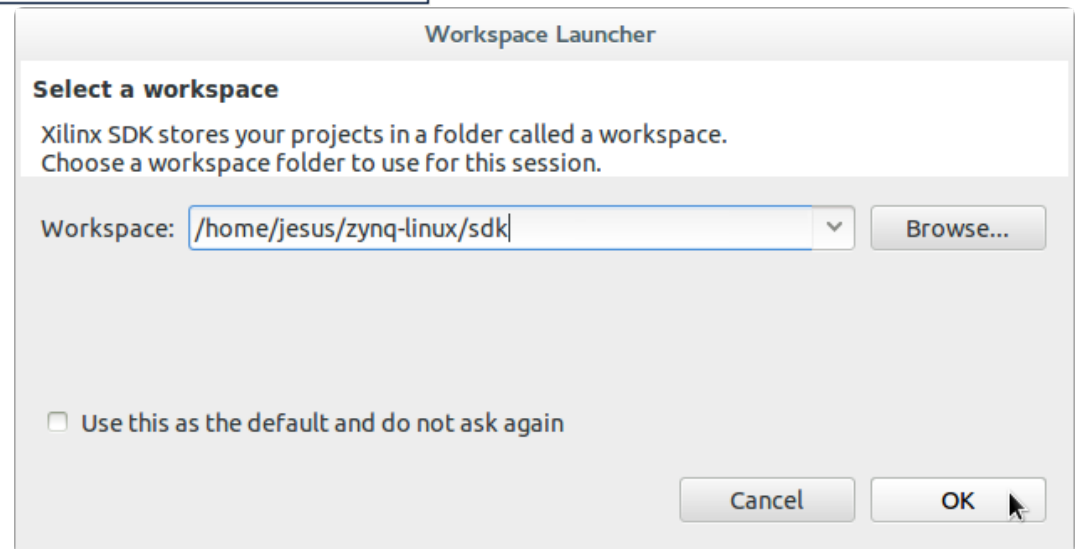
# Exportamos el diseño de la plataforma de Vivado al SDK



# Exportamos el diseño de la plataforma de Vivado al SDK

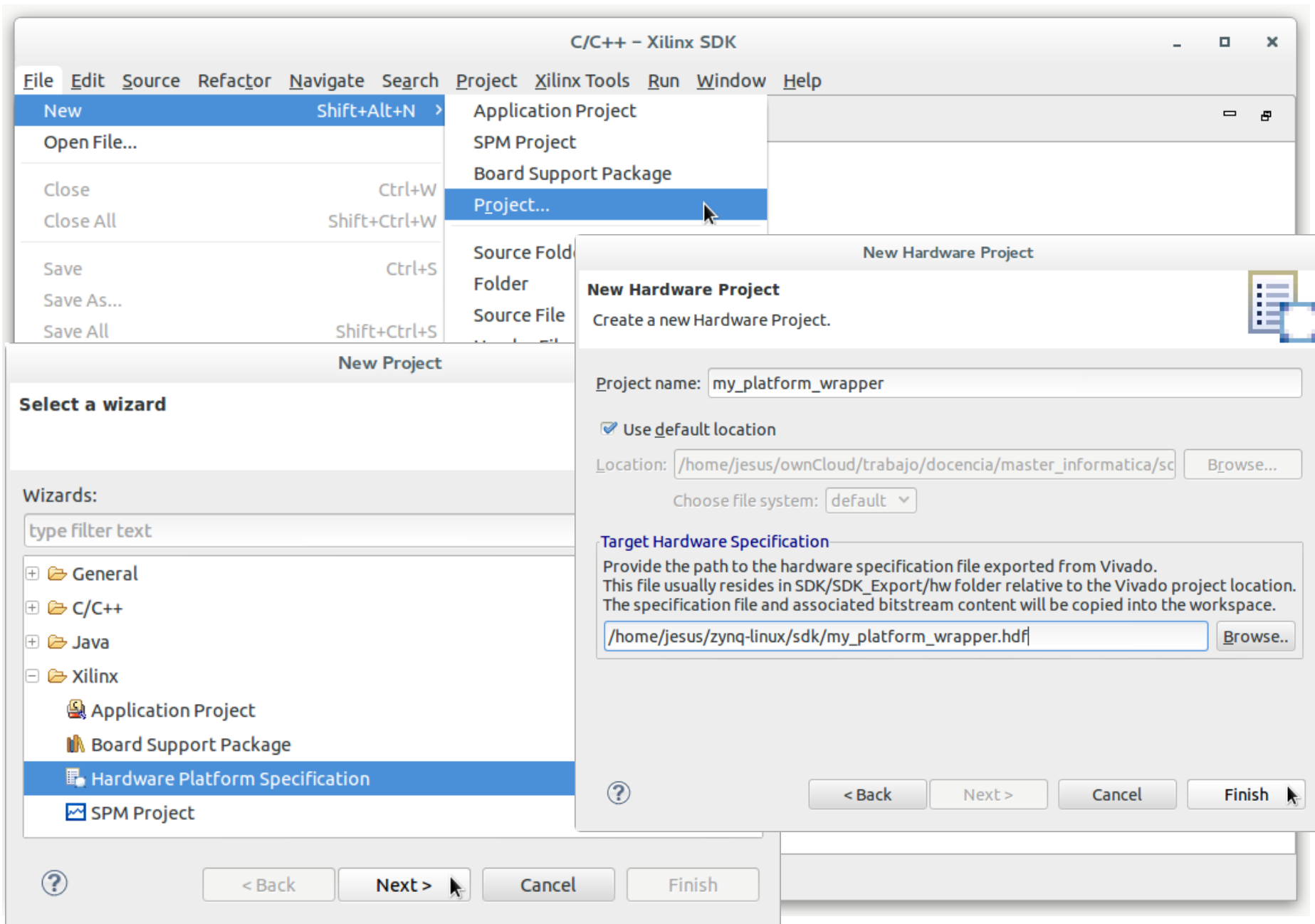


# Creación de un proyecto HW en el *SDK* para nuestra plataforma





# Creación de un proyecto HW en el SDK para nuestra plataforma



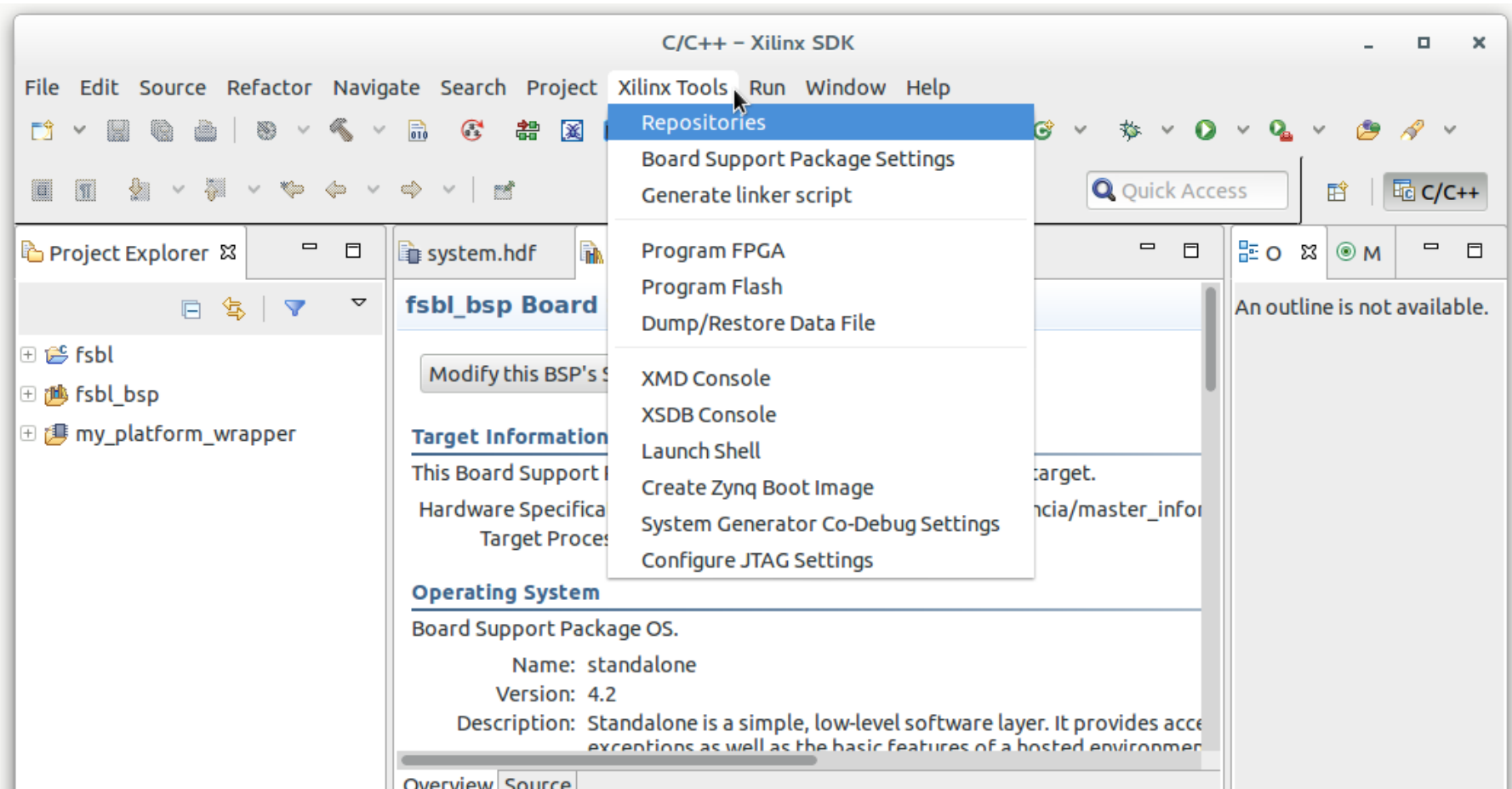
# Generación del *Device Tree Source* mediante el SDK

## Descargamos el *Device Tree Generator*

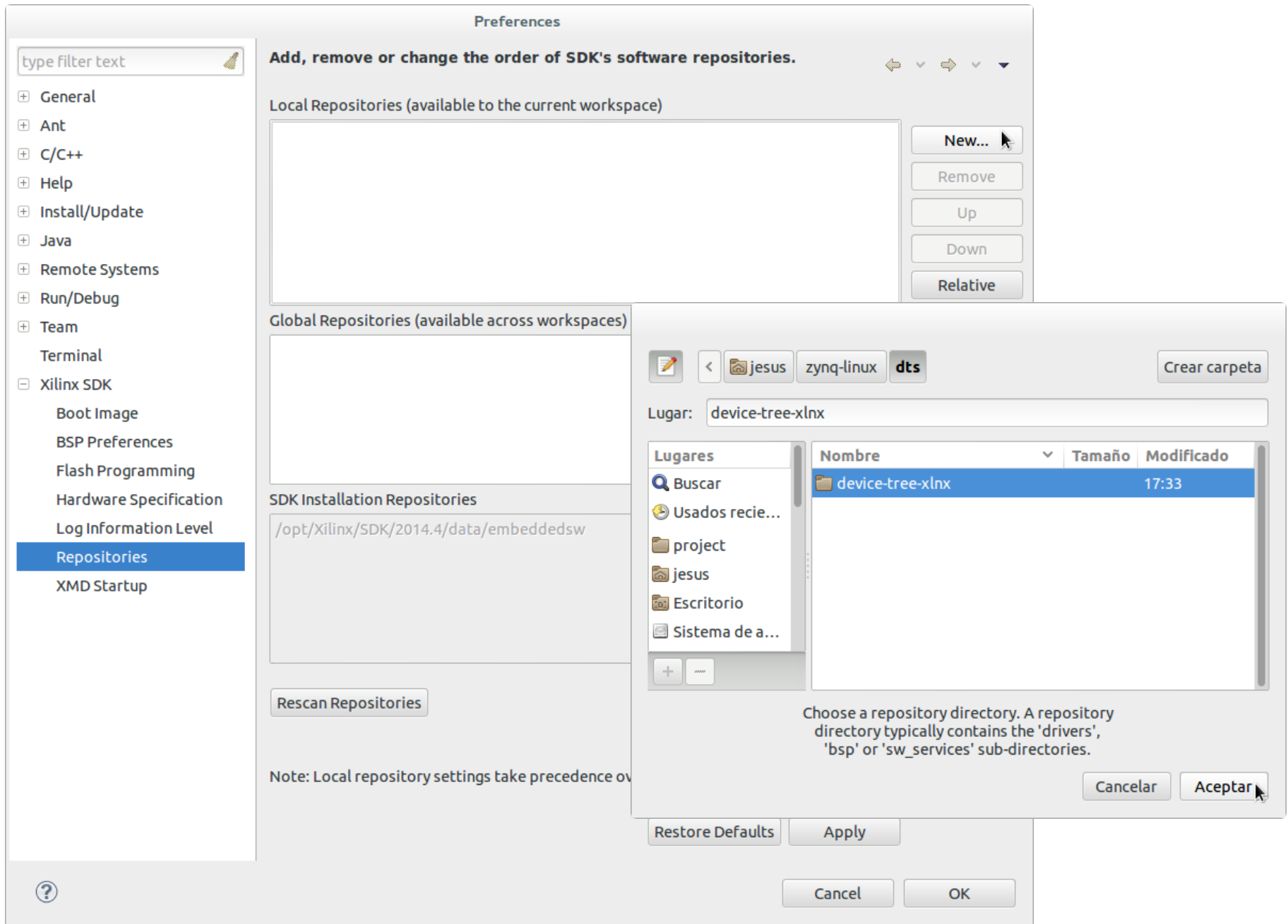
```
mkdir -p ${PRJ_ROOT}/dts
```

```
git -C ${PRJ_ROOT}/dts clone git://github.com/Xilinx/device-tree-xlnx.git
```

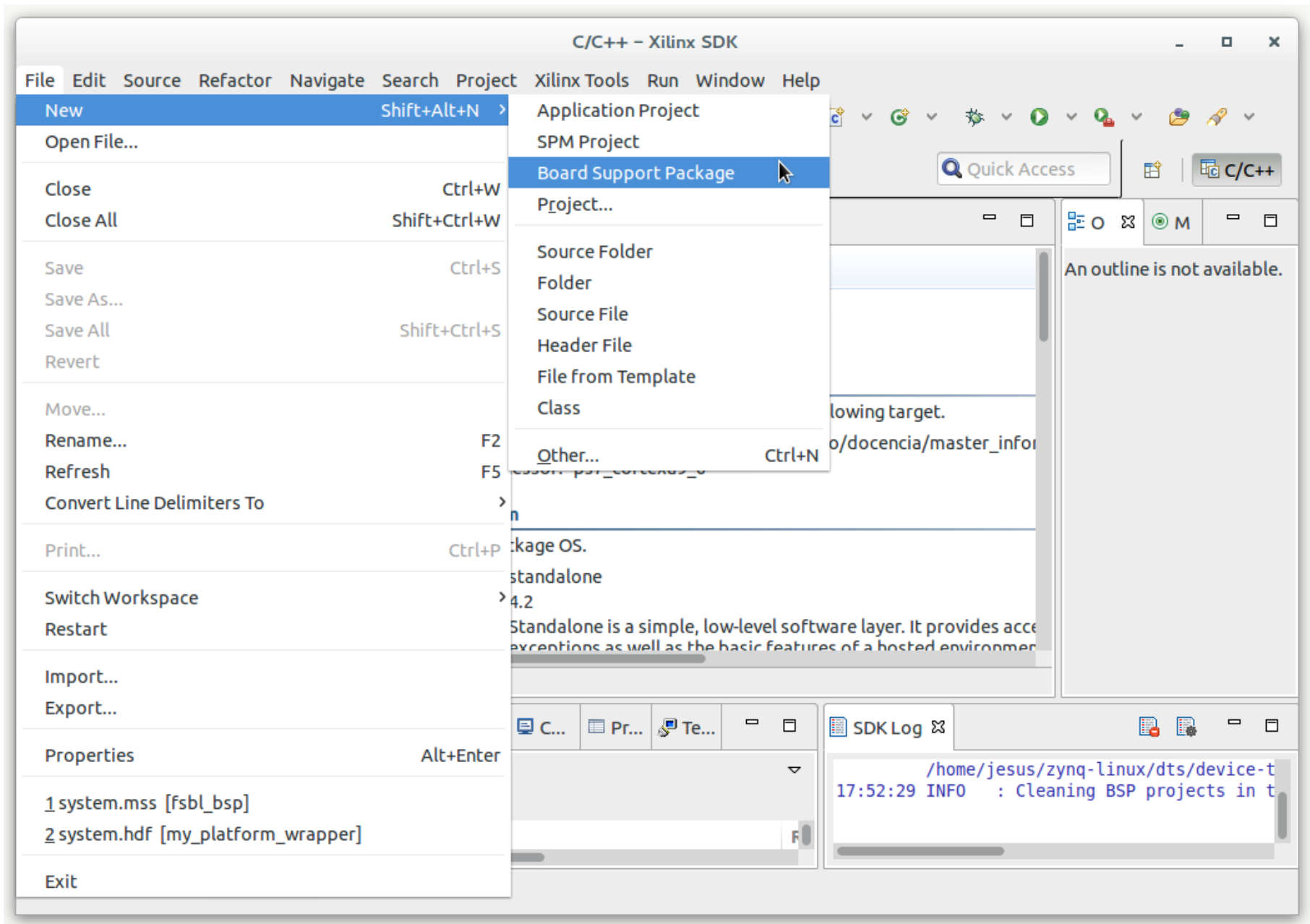
## Importamos el repositorio en el SDK



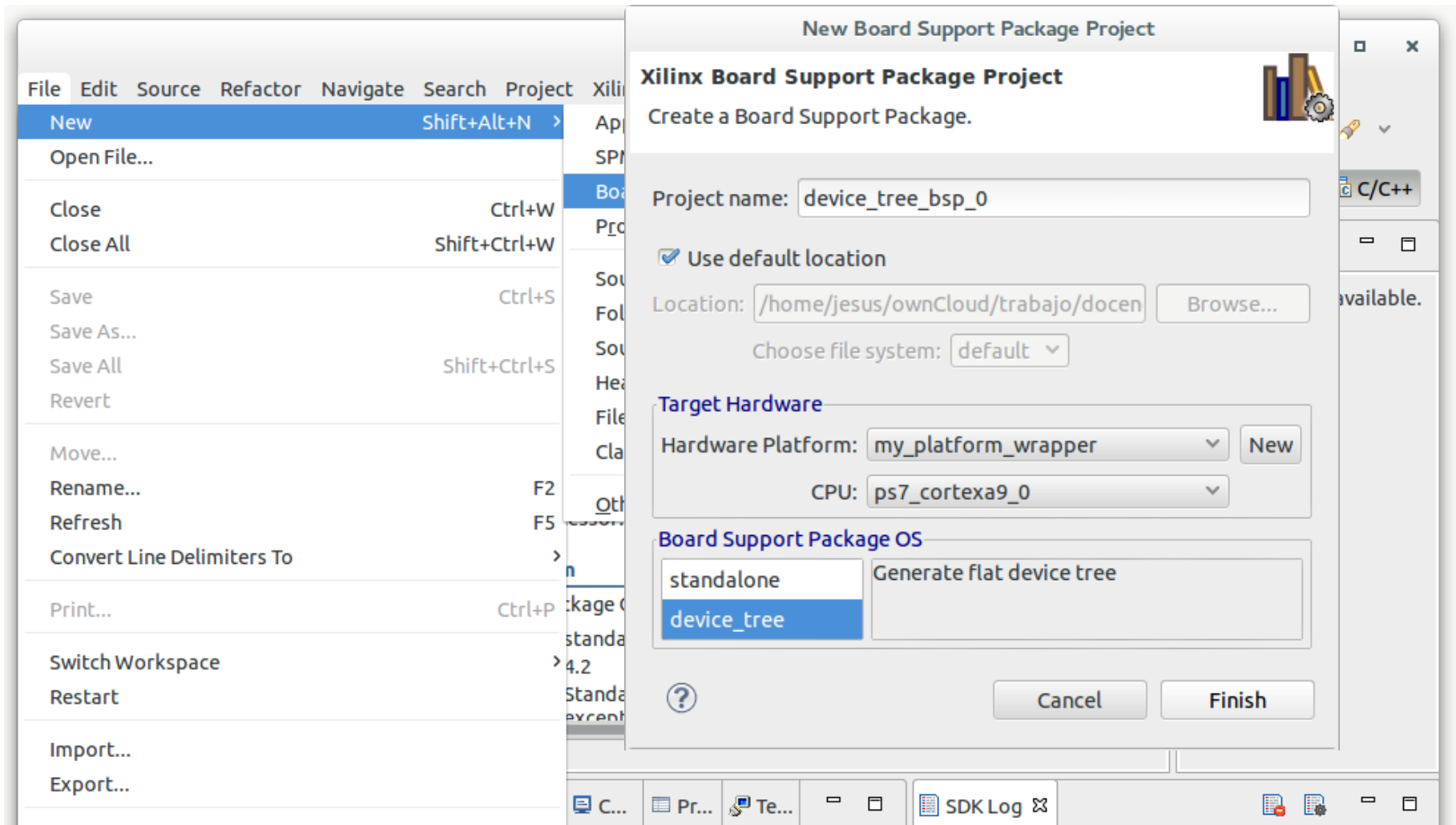
# Construcción del *Device Tree Source* mediante el SDK



# Construcción del *Device Tree Source* mediante el SDK



# Construcción del *Device Tree Source* mediante el SDK



## Construcción del *Device Tree Blob* a partir del *Device Tree Source*

```
cd ${PRJ_ROOT}/sdk/device_tree_bsp_0
${KERNEL_DIR}/scripts/dtc/dtc -I dts -O dtb -o devicetree.dtb system.dts
cp devicetree.dtb ${PRJ_ROOT}/images
```

# Construcción del *Device Tree Source* mediante *scripts*

## Variables de entorno

```
export PLATFORM="xilinx_zynq_a9"  
export PLATFORM_DIR="${PRJ_ROOT}/${PLATFORM}"  
export PLATFORM_WRAPPER="${PLATFORM}_wrapper"  
export SDK_DIR="${PRJ_ROOT}/sdk"  
export DTB_DIR="${PRJ_ROOT}/devicetree"  
export DTS_DIR="${DTB_DIR}/dts"  
KERNEL="Linux-Digilent-Dev"  
KERNEL_DIR="${PRJ_ROOT}/${KERNEL}"
```

## Exportamos el diseño de la plataforma

```
mkdir -p ${DTB_DIR}/${PLATFORM_WRAPPER}  
cp ${PLATFORM_DIR}/${PLATFORM}.runs/impl_1/${PLATFORM_WRAPPER}.sysdef \  
  ${DTB_DIR}/${PLATFORM_WRAPPER}/${PLATFORM_WRAPPER}.hdf
```

## Descargamos el *Device Tree Generator*

```
git -C ${DTB_DIR} clone git://github.com/Xilinx/device-tree-xlnx.git
```

# Construcción del *Device Tree Blob* mediante *scripts*

## Generamos el *Device Tree Source*

```
hsi -mode batch -source dts.tcl
```

## Generamos el *Device Tree Blob* a partir del *Device Tree Source*

```
cd ${DTB_DIR}
${KERNEL_DIR}/scripts/dtc/dtc -I dts -O dtb \
    -o devicetree.dtb ${DTS_DIR}/system.dts
cp devicetree.dtb ${PRJ_ROOT}/images
```

## Fichero *dts.tcl*

```
open_hw_design $env(DTB_DIR)/$env(PLATFORM_WRAPPER)/$env(PLATFORM_WRAPPER).hdf
set_repo_path $env(DTB_DIR)/device-tree-xlnx
create_sw_design device-tree -os device_tree -proc ps7_cortexa9_0
generate_target -dir $env(DTS_DIR)
```

# Contenidos

## Tema 1: Selección y configuración de un sistema operativo

Introducción

Fundamentos de *Linux*

Selección de la plataforma y prerequisites del sistema

Diseño de una plataforma de ejecución mínima

Construcción del *kernel* de *Linux*

Construcción del *Device Tree Blob*

Necesidad de un *Root File System*

Construcción de un *Root File System*

Generación del *First Stage Boot Loader*

Construcción de *U-Boot*

Preparación de la imagen de arranque

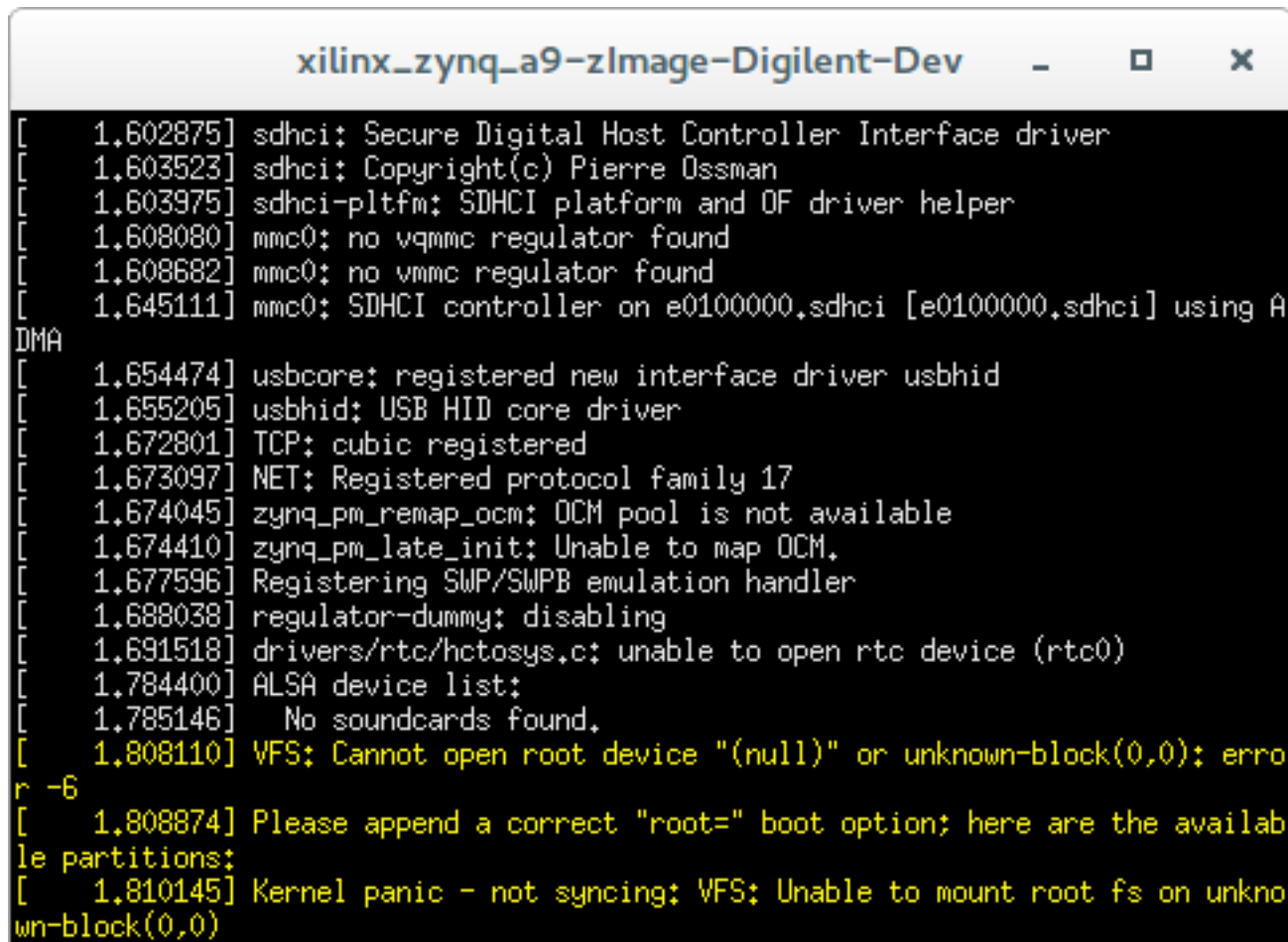


# Ejecución

## Ejecución en el simulador

```
cd ${PRJ_ROOT}/images
```

```
qemu-system-arm -M xilinx-zynq-a9 -m 1024 -serial null -serial mon:stdio \  
-nographic -dtb devicetree.dtb -kernel zImage
```



```
xilinx_zynq_a9-zImage-Digilent-Dev  -  □  x  
[ 1.602875] sdhci: Secure Digital Host Controller Interface driver  
[ 1.603523] sdhci: Copyright(c) Pierre Ossman  
[ 1.603975] sdhci-pltfm: SDHCI platform and OF driver helper  
[ 1.608080] mmc0: no vqmmc regulator found  
[ 1.608682] mmc0: no vmmc regulator found  
[ 1.645111] mmc0: SDHCI controller on e0100000.sdhci [e0100000.sdhci] using A  
DMA  
[ 1.654474] usbcore: registered new interface driver usbhid  
[ 1.655205] usbhid: USB HID core driver  
[ 1.672801] TCP: cubic registered  
[ 1.673097] NET: Registered protocol family 17  
[ 1.674045] zynq_pm_remap_ocm: OCM pool is not available  
[ 1.674410] zynq_pm_late_init: Unable to map OCM.  
[ 1.677596] Registering SWP/SWPB emulation handler  
[ 1.688038] regulator-dummy: disabling  
[ 1.691518] drivers/rtc/hctosys.c: unable to open rtc device (rtc0)  
[ 1.784400] ALSA device list:  
[ 1.785146]   No soundcards found.  
[ 1.808110] VFS: Cannot open root device \"(null)\" or unknown-block(0,0): erro  
r -6  
[ 1.808874] Please append a correct \"root=\" boot option; here are the availab  
le partitions:  
[ 1.810145] Kernel panic - not syncing: VFS: Unable to mount root fs on unkno  
wn-block(0,0)
```

## Nuestro fichero *init*

Código fuente del fichero `${PRJ_ROOT}/project/myinit.c`

```
#include <stdio.h>

int main ()
{
    /* Escribimos el mensaje de saludo */
    printf ("\n");
    printf ("Hola desde la Zybo!\n");

    /* El proceso init nunca debe terminar */
    while (1) { }

    /* Para que no proteste el compilador */
    return 0;
}
```

# Creación de un *rootfs* mínimo

## Creamos un directorio para almacenar el sistema de archivos

```
mkdir -p ${PRJ_ROOT}/rootfs  
cd ${PRJ_ROOT}/rootfs
```

## Simulamos ser *root*

```
fakeroot
```

## Compilamos nuestro proceso init (en /)

```
${CROSS_COMPILE}gcc -static ${PRJ_ROOT}/project/myinit.c -o init
```

## Creamos la consola

```
mkdir -m 0755 dev  
mknod dev/console c 5 1
```

## Creamos el fichero cpio

```
find . | cpio --quiet -o -H newc | gzip > ${PRJ_ROOT}/images/min_rootfs.cpio.gz
```

## Dejamos de simular que somos *root*

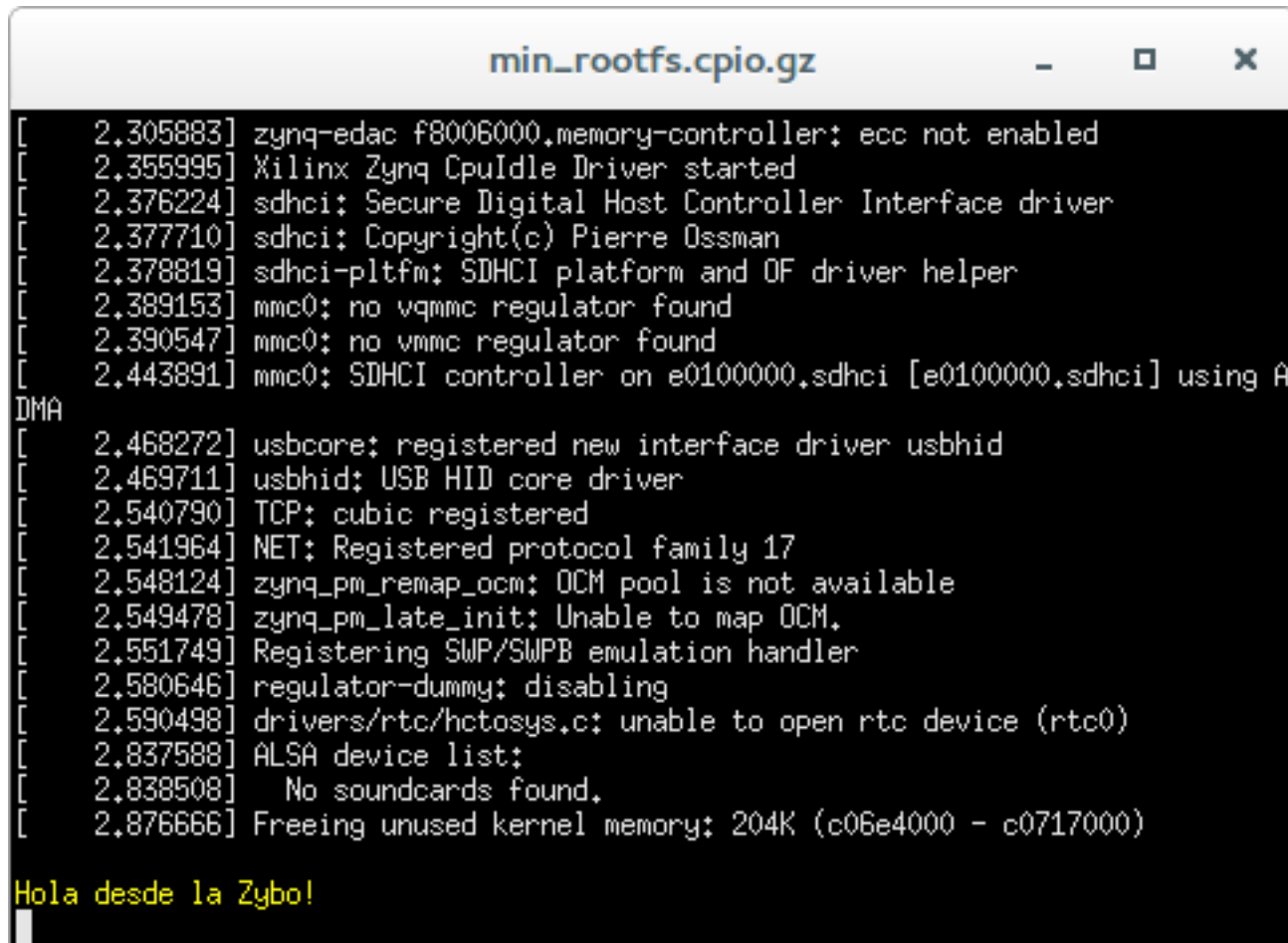
```
exit
```

# Ejecución

## Ejecución en el simulador

```
cd ${PRJ_ROOT}/images
```

```
qemu-system-arm -M xilinx-zynq-a9 -m 1024 -serial null -serial mon:stdio \  
-nographic -dtb devicetree.dtb -kernel zImage -initrd min_rootfs.cpio.gz
```



```
min_rootfs.cpio.gz  
[ 2.305883] zynq-edac f8006000.memory-controller: ecc not enabled  
[ 2.355995] Xilinx Zynq CpuIdle Driver started  
[ 2.376224] sdhci: Secure Digital Host Controller Interface driver  
[ 2.377710] sdhci: Copyright(c) Pierre Ossman  
[ 2.378819] sdhci-pltfm: SDHCI platform and OF driver helper  
[ 2.389153] mmc0: no vqmmc regulator found  
[ 2.390547] mmc0: no vmmc regulator found  
[ 2.443891] mmc0: SDHCI controller on e0100000.sdhci [e0100000.sdhci] using A  
DMA  
[ 2.468272] usbcore: registered new interface driver usbhid  
[ 2.469711] usbhid: USB HID core driver  
[ 2.540790] TCP: cubic registered  
[ 2.541964] NET: Registered protocol family 17  
[ 2.548124] zynq_pm_remap_ocm: OCM pool is not available  
[ 2.549478] zynq_pm_late_init: Unable to map OCM.  
[ 2.551749] Registering SWP/SWPB emulation handler  
[ 2.580646] regulator-dummy: disabling  
[ 2.590498] drivers/rtc/hctosys.c: unable to open rtc device (rtc0)  
[ 2.837588] ALSA device list:  
[ 2.838508]   No soundcards found.  
[ 2.876666] Freeing unused kernel memory: 204K (c06e4000 - c0717000)  
  
Hola desde la Zybo!  
█
```

# Contenidos

## Tema 1: Selección y configuración de un sistema operativo

Introducción

Fundamentos de *Linux*

Selección de la plataforma y prerequisites del sistema

Diseño de una plataforma de ejecución mínima

Construcción del *kernel* de *Linux*

Construcción del *Device Tree Blob*

Necesidad de un *Root File System*

Construcción de un *Root File System*

Generación del *First Stage Boot Loader*

Construcción de *U-Boot*

Preparación de la imagen de arranque

# Construcción de *Busybox*

## Directorio de instalación

```
mkdir -p ${PRJ_ROOT}/sysapps  
cd ${PRJ_ROOT}/sysapps
```

## Obtención de las fuentes de *busybox*

```
export BUSYBOX="busybox-1.22.1"  
wget http://busybox.net/downloads/${BUSYBOX}.tar.bz2  
tar xf ${BUSYBOX}.tar.bz2
```

## Configuración mínima y construcción de *busybox*

```
cd ${BUSYBOX}  
export ARCH="arm"  
export HOST="${ARCH}-xilinx-linux-gnueabi"  
export CROSS_COMPILE="${HOST}-"  
make defconfig  
make menuconfig
```

```
Busybox Settings ->  
  Build Options ->  
    [ ] Build BusyBox as a static binary (no shared libs)  
    [ ] Force NOMMU build  
    [ ] Build with Large File Support (for accessing files > 2 GB)  
    (${CROSS_COMPILE}) Cross Compiler prefix
```

```
make -j 4
```

# Busybox es altamente configurable

jesus@gargamel: ~/embedded\_linux/sysapps/busybox-1.22.1

Archivo Editar Ver Buscar Terminal Ayuda

## BusyBox 1.22.1 Configuration

### Busybox Configuration

Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [\*] built-in [ ] excluded <M> module < > module capable

```
Busybox Settings --->
-- Applets
Archival Utilities --->
Coreutils --->
Console Utilities --->
Debian Utilities --->
Editors --->
Finding Utilities --->
Init Utilities --->
Login/Password Management Utilities --->
Linux Ext2 FS Progs --->
Linux Module Utilities --->
Linux System Utilities --->
Miscellaneous Utilities --->
Networking Utilities --->
Print Utilities --->
Mail Utilities --->
Process Utilities --->
Runit Utilities --->
Shells --->
System Logging Utilities --->
---
Load an Alternate Configuration File
Save Configuration to an Alternate File
```

<Select> < Exit > < Help >

# Construimos el rootfs

## Creamos un directorio para almacenar el sistema de archivos

```
rm -rf ${PRJ_ROOT}/rootfs  
mkdir -p ${PRJ_ROOT}/rootfs  
cd ${PRJ_ROOT}/rootfs
```

## Simulamos ser *root*

```
fakeroot
```

## Creamos los directorios esenciales

```
mkdir -m 0700 root  
mkdir -m 0755 bin dev etc lib proc sbin sys usr var  
mkdir -m 0755 usr/bin usr/sbin usr/lib  
mkdir -m 0755 var/lib var/lock var/log var/run var/www  
mkdir -m 0755 etc/init.d
```

## Los ficheros temporales sólo podrán ser borrados por quien los haya creado

```
mkdir -m 1777 tmp var/tmp
```



# Construimos el rootfs

## Instalamos los módulos del *kernel*

```
cd ${PRJ_ROOT}/${KERNEL}
export ARCH="arm"
export HOST="${ARCH}-xilinx-linux-gnueabi"
export CROSS_COMPILE="${HOST}-"
make INSTALL_MOD_PATH=${PRJ_ROOT}/rootfs/ modules_install
```

## Copiamos las bibliotecas de *glibc*

```
cp -r ${XILINX_ROOT}/SDK/2014.4/gnu/arm/lin/${HOST}/libc/lib/* \
    ${PRJ_ROOT}/rootfs/lib
```

## Quitamos la información de depuración de las bibliotecas

```
arm-xilinx-linux-gnueabi-strip ${PRJ_ROOT}/rootfs/lib/*
```

## Instalamos *busybox* en el *Root FS*

```
cd ${PRJ_ROOT}/sysapps/${BUSYBOX}
make CONFIG_PREFIX=${PRJ_ROOT}/rootfs install
```

## Enlace para el proceso *init*

```
cd ${PRJ_ROOT}/rootfs
ln -s bin/busybox init
```

Poblamos el directorio `${PRJ_ROOT}/rootfs/etc`

`${PRJ_ROOT}/rootfs/etc/profile`

```
# Fijamos el PATH
PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH
```

`${PRJ_ROOT}/rootfs/etc/inittab`

```
# Fijamos /etc/init.d/rcS como fichero de inicializacion del sistema
::sysinit:/etc/init.d/rcS
```

*Script de inicialización del sistema*

```
# Iniciamos una sesión de login en la consola
```

```
::respawn:/sbin/getty 115200 ttyPS0
```

Proceso que será iniciado al arrancar el sistema y reiniciado cada vez que termine

```
# Indicamos que se ejecute /sbin/init si init se reinicia
```

```
::restart:/sbin/init
```

```
# Fijamos /etc/init.d/rck como fichero de apagado del sistema
```

```
::shutdown:/etc/init.d/rck
```

*Script de apagado del sistema*

# Poblamos el directorio /etc

`${PRJ_ROOT}/rootfs/etc/init.d/rcS`

```
#!/bin/sh
```

```
hostname -F /etc/hostname
```

Asignamos un nombre al sistema

```
mount -t sysfs none /sys
```

```
mount -t proc none /proc
```

```
mount -t tmpfs none /tmp
```

Montamos los  
sistemas de archivos

```
echo "/sbin/mdev" > /proc/sys/kernel/hotplug  
/sbin/mdev -s
```

Inicializamos  
los dispositivos

```
mkdir -p /dev/pts
```

```
mkdir -p /dev/i2c
```

```
mount -t devpts devpts /dev/pts
```

```
ifconfig eth0 down
```

```
ifconfig eth0 192.168.1.10 up
```

Configuramos  
una IP estática

```
telnetd -l /bin/sh
```

```
httpd -h /var/www
```

```
tcpsvd 0:21 ftpd ftpd -w /&
```

Iniciamos los  
demonios

# Poblamos el directorio /etc

```
${PRJ_ROOT}/rootfs/etc/init.d/rcK
```

```
#!/bin/sh  
umount -a -r
```

Hacemos que los *scripts* de inicialización y apagado sean ejecutable

```
chmod a+x etc/init.d/rcS etc/init.d/rcK
```

Nombre del equipo

```
echo "xilinx_zynq_a9" > etc/hostname
```

Mensaje de bienvenida

```
echo "ARM-Linux desde cero \n \l" > etc/issue
```

Fichero *passwd*

```
echo "root::0:0:root:/root:/bin/sh" > etc/passwd
```

Creamos el archivo *cpio*

```
find . | cpio --quiet -o -H newc | gzip > ${PRJ_ROOT}/images/rootfs.cpio.gz
```

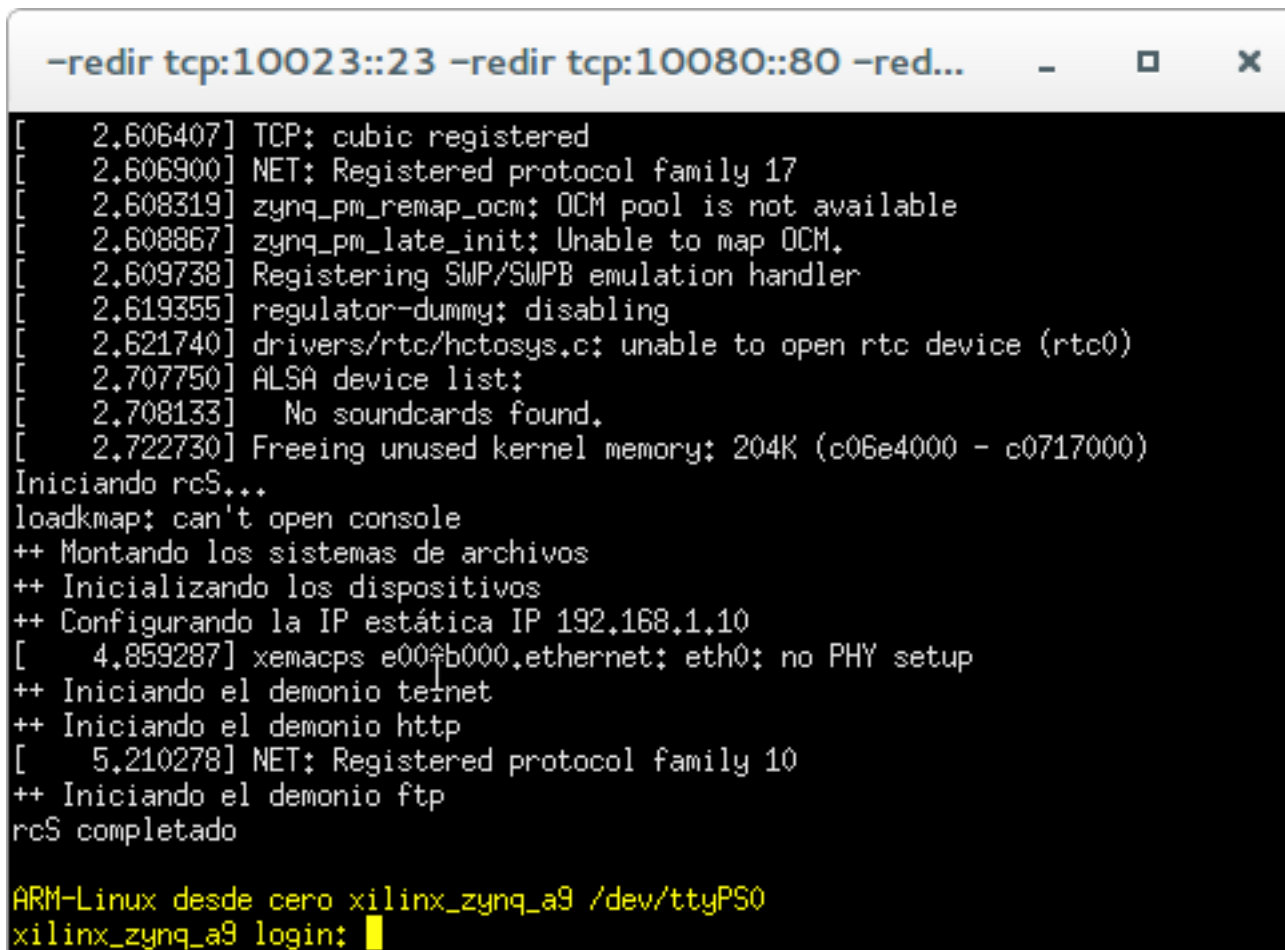
Dejamos de simular que somos *root*

```
exit
```

# Ejecución

## Ejecución en el simulador

```
qemu-system-arm -M xilinx-zynq-a9 -m 1024 -serial null -serial mon:stdio \  
-nographic -dtb devicetree.dtb -kernel zImage -initrd rootfs.cpio.gz \  
-net nic,model=cadence_gem -net user -tftp ~/ -redir tcp:10023::23 \  
-redir tcp:10080::80 -redir tcp:10022::22 -redir tcp:10021::21
```



```
-redir tcp:10023::23 -redir tcp:10080::80 -red...  
[ 2.606407] TCP: cubic registered  
[ 2.606900] NET: Registered protocol family 17  
[ 2.608319] zynq_pm_remap_ocm: OCM pool is not available  
[ 2.608867] zynq_pm_late_init: Unable to map OCM.  
[ 2.609738] Registering SLP/SWPB emulation handler  
[ 2.619355] regulator-dummy: disabling  
[ 2.621740] drivers/rtc/hctosys.c: unable to open rtc device (rtc0)  
[ 2.707750] ALSA device list:  
[ 2.708133]   No soundcards found.  
[ 2.722730] Freeing unused kernel memory: 204K (c06e4000 - c0717000)  
Iniciando rcS...  
loadkmap: can't open console  
++ Montando los sistemas de archivos  
++ Inicializando los dispositivos  
++ Configurando la IP estática IP 192.168.1.10  
[ 4.859287] xemacps e001b000.ethernet: eth0: no PHY setup  
++ Iniciando el demonio telnet  
++ Iniciando el demonio http  
[ 5.210278] NET: Registered protocol family 10  
++ Iniciando el demonio ftp  
rcS completado  
  
ARM-Linux desde cero xilinx_zynq_a9 /dev/ttyPS0  
xilinx_zynq_a9 login: █
```

# Lecturas recomendadas

## Linux en plataformas de Xilinx:

Xilinx. *Getting Started. Overview of the Xilinx Zynq AP SoC Design Flow.*

<http://www.wiki.xilinx.com/Getting+Started>

Diligent. *Embedded Linux Hands-on Tutorial for the ZYBO*, julio 2014.

[http://digilentinc.com/Data/Products/ZYBO/ZYBO-Embedded\\_Linux\\_Hands-on\\_Tutorial.pdf](http://digilentinc.com/Data/Products/ZYBO/ZYBO-Embedded_Linux_Hands-on_Tutorial.pdf)

Bankras.org. *Xilinx Vivado 2014.4 and the Diligent ZYBO tutorial.*

<http://www.bankras.net/radko/uncategorized/xilinx-vivado-2014-4-and-the-diligent-zybo-tutorial/>

## QEMU:

QEMU. *Main Page.* <http://wiki.qemu.org/>

## Device Tree:

Xillybus. *A Tutorial on the Device Tree (Zynq).*

<http://xillybus.com/tutorials/device-tree-zynq-1>

## Root filesystem:

Denys Vlasenko. *BusyBox.* <http://www.busybox.net/>

Xilinx. *Zynq Root File System Creation.* <http://xilinx.wikidot.com/zynq-rootfs>