

Proyecto 1: Proceso de arranque de un sistema embebido BSP y entorno de desarrollo

Nombre: Ing. Luis Carlos Alvarez Mata
Nombre: Ing. Verny Moreles Soto

Carne: 200902144
Carne: 201116212

Durante el proyecto, aprendí cuál es el proceso de arranque común de un dispositivo embebido y el caso específico de Raspberry Pi 4 (RPI4), además, se utilizan las herramientas de compilación cruzada Linaro y utilizando una distribución del Proyecto Yocto.

1. Respuesta de preguntas

1.1. ¿Qué es un device tree, por qué se introdujo y por qué es tan importante?

Un “device tree” es una estructura de datos que describe los componentes de hardware de una computadora en particular para que el kernel del sistema operativo pueda utilizar y los pueda administrar. Alguna información que se puede encontrar en el devicetree es la cantidad de cores que tiene el sistema, cuanta memoria, cual es la dirección donde está mapeada la memoria, cantidad de buses y que dispositivos hay conectados a los buses. El OS necesita esta información para saber cuales drivers necesita cargar. Es de suma importancia para algunas arquitecturas el devicetree ya que el kernel lo necesita para saber que hardware se está utilizando [1].

1.2. ¿Qué es un device tree overlay, qué se puede controlar con ella?

Es el formato que manejan los desarrolladores para incorporar los ficheros y ser incluidos en el kernel. Cuando el compilador parsea el primer fichero irá haciéndolo de manera ordenada, uno por uno. A esto se le llama overlay, el cual habilita ingresar librerías, y ficheros de manera dinámica agregando nodos y realizando cambios al árbol existente. En este proyecto el agregar nuevas recetas de la manera en que lo hicimos con los meta layers es un ejemplo de este tipo de topología y manera de agregar datos a una imagen ya existente. Por ejemplo, la raspberry pi permite crear layers alternas con componentes que no siempre se incorporan utilizando lo que se llama como un devicetree parcial, y posteriormente se crea el devicetree master, el cual está compuesto por todas las ramas agregadas por medio de los overlays [2].

1.3. ¿Cuáles son los processor fuses y las bootstrap resistors y qué exactamente del proceso de arranque se puede controlar con ellos?

Los fuses son básicamente interconexiones que existen entre el core y todo el ensamblaje principal (cache, uncore, restos de cores, SA, entre otros). Los fuses se programan una única vez en la etapa de desarrollo, esto permite al desarrollador bloquear o activar ciertas regiones del procesador lo que permite segmentar los procesadores y dividirlos en función a sus características disponibles. Esto permite colocar diferentes precios al mismo procesador, ya que algunos van a tener mayores capacidades y atributos que otros dependiendo que fuses se encuentren habilitados [3].

Un circuito de bootstrap es aquel en el que parte de la salida de una etapa de amplificador se aplica a la entrada, para alterar la impedancia de entrada del amplificador. Cuando se aplica deliberadamente, la intención suele ser aumentar en lugar de disminuir la impedancia. En el dominio de los circuitos MOSFET, “bootstrapping” se usa comúnmente para significar levantar el punto de operación de un transistor por encima del riel de la fuente de alimentación para habilitar alguna configuración. A nivel de un procesador un bootstrap resistor se utiliza para

habilitar configuraciones de inicio o características las cuales están ligadas al hardware y son controladas por medio de una impedancia específica. [4, 5, 6].

Un ejemplo de esta implementación se muestra en el circuito de selección para la configuración de un Half-Bridge, donde un bootstrap resistor se utiliza para la configuración de arranque en configuraciones para suministrar la polarización al high-side FET. Figura ?? muestra la ruta de carga de un circuito de arranque en una configuración simplificada. Cuando el low-side FET está encendido (el high-side FET está apagado), el pin HS y el switch nodo se tiran a tierra; el suministro de polarización VDD, a través del bypass capacitor, carga el capacitor de arranque a través del diodo de arranque y la resistencia [7].

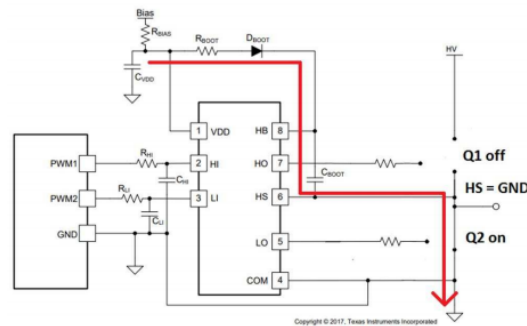


Figura 1: Bootstrap de carga. [7]

Cuando se apaga el low-side FET y se enciende el high-side, el pin HS del controlador de puerta y el switch nodo se llevan al bus de alto voltaje HV; el bootstrap capacitor descarga parte del voltaje almacenado (acumulado durante la secuencia de carga) al high-side FET a través de los pines HO y HS del controlador de puerta como se muestra en la Figura 15 [7].

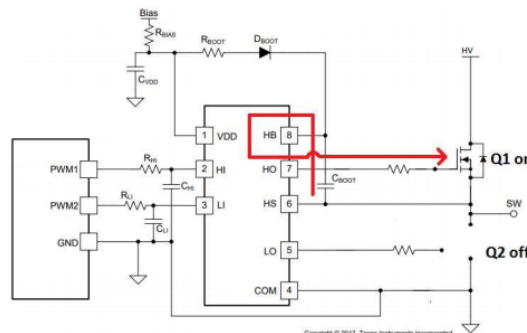


Figura 2: Bootstrap de descarga. [7]

Los procesadores ARM Cortex-M4 utilizan straps para cambiar la funcionalidad de ciertos pines donde por ejemplo por medio del pin "GPIO104.es utilizado para para determinar el nivel de voltaje para la interfaz de Flash compartido. También se utilizan straps para la configuración de las señales JTAG utilizadas para la depuración [8].

1.4. Investigar la ubicación predeterminada en la RAM del árbol de dispositivos en el momento del arranque para el RPI4 y busque información sobre las direcciones RAM reservadas que U-Boot usa para operar.

La dirección del device tree usado en nuestra configuración va desde 0x3af499b0 hasta 0x3af56dc1, siendo esta la ubicación por default

La mayoría de espacios de memoria reservados se encuentra en el archivo config.txt donde se pueden encontrar los parámetros de configuración para video y el Kernel de Linux (direcciones de carga, device tree, UART / velocidades

de transmisión de la consola, etc.).

El kernel es cargado en la dirección 0x8000 y los sistemas basados en linux tienen un tamaño alrededor de 64MB, por lo cual este fragmento de la memoria será reservado para el kernel.

Sobre NFS

¿Se carga todo el sistema de archivos en la RAM cuando monta un NFS? Si no, ¿cuál es el criterio seguido por el Kernel de Linux para cargar archivos del sistema de archivos en la RAM?

No. Una vez se monta el NFS, el cual es guardado en el SD card,

Agregar respuesta

¿Qué pasa si usamos un Initrd? ¿Se copiaría completamente en la RAM en el momento del arranque?

Agregar respuesta

¿Cuáles son los beneficios y los casos de uso de ambos?

Agregar respuesta

2. Resultados

Generar el meta-tec con la receta

1. Creación de la carpeta meta-tec.

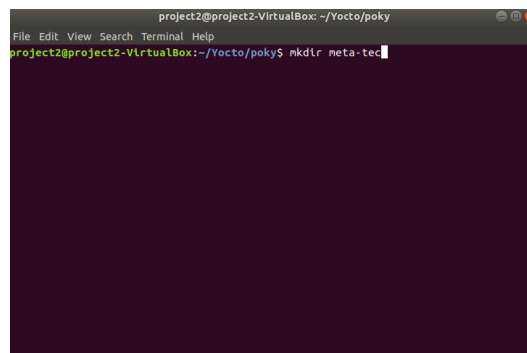


Figura 3: mkdir meta-tec.

2. Posteriormente se copia la carpeta files la cual fue dada por los profesores para utilizar el autotools. En la siguiente imagen se observa una imagen de la carpeta original, sin embargo, esta se va a modificar para ser utilizada por el código en C del trapecioide.
3. Se crean las carpetas donde estarán las recetas correspondientes para generar el código ejecutable en C.
4. Se procede a crear el esquema de archivos dentro de la carpeta trapezoidal-1.0 la cual se va a utilizar para compilar el código C. A continuación se muestran la jerarquía de carpetas, así como los archivos Makefile.am y el configure.ac, vitales en el autotool.
5. Se ejecuta el autoreconf --install.

```
project2@project2-VirtualBox: ~/Yocto/poky/meta-tec
File Edit View Search Terminal Help
project2@project2-VirtualBox:~/Yocto/poky/meta-tec$ tree
.
├── files
│   ├── autotools
│   │   ├── configure.ac
│   │   ├── Makefile.am
│   │   └── src
│   │       ├── hello.c
│   │       └── Makefile.am
│   ├── conf
│   │   ├── layer.conf
│   │   ├── COPYING.MIT
│   │   ├── layer
│   │   │   ├── layer.conf
│   │   │   └── README
│   │   └── recipe
│   │       └── hello-1.0.bb
└── 6 directories, 9 files
```

Figura 4: Carpeta File para utilizar el autotools.

```
project2@project2-VirtualBox: ~/Yocto/poky/meta-tec/recipes-trapezoidal
File Edit View Search Terminal Help
project2@project2-VirtualBox:~/Yocto/poky/meta-tec/recipes-trapezoidal$ tree
.
├── trapezoidal
│   └── trapezoidal-1.0
└── 2 directories, 0 files
```

Figura 5: Jerarquía de folders para preparar la receta

```
project2@project2-VirtualBox: ~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal-1.0
File Edit View Search Terminal Help
project2@project2-VirtualBox:~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal-1.0$ tree
.
├── configure.ac
├── Makefile.am
├── src
│   └── trapezoidal.c
└── 1 directory, 4 files
```

Figura 6: Jerarquía de archivos para compilar el código C

```
Open ▾ *Makefile.am
~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal-1.0/Makefile.am
bin_PROGRAMS = trapezoidal
trapezoidal_SOURCES = trapezoidal.c
```

Figura 7: Archivo Makefile.am

```
Open ▾ *configure.ac
~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal-1.0/configure.ac
AC_INIT([trapezoidal], [1.0], [sercr0388@address])
AM_INIT_AUTOMAKE([foreign -Wall -Werror])
AC_PROG_CC
AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile src/Makefile])
AC_OUTPUT
```

Figura 8: Archivo configure.ac

```
project2@project2-VirtualBox: ~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal-1.0$ autoreconf --install
configure.ac:3: installing './compile'
configure.ac:2: installing './install.sh'
configure.ac:2: installing './missing'
src/Makefile.am: installing './depcomp'
project2@project2-VirtualBox:~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal-1.0$
```

Figura 9: Archivo configure.ac

6. Se procede a correr el `./configure` para crear el entorno de compilación.
7. Se procede a generar el binario.
8. Generación del paquete para que Yocto entienda que esto va a ser incorporado en la imagen con el comando `make distcheck`.

```
project2@project2-VirtualBox: ~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal/trapezoidal-1.0
File Edit View Search Terminal Help

project2@project2-VirtualBox:~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal/trapezoidal-1.0$ ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking whether gcc understands -c and -o together... yes
checking for style of include used by make... GNU
checking dependency style of gcc... gcc
checking that generated files are newer than configure... done
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating config.h
config.status: executing depfiles commands
```

Figura 10: Entorno de compilación creado

```
project2@project2-VirtualBox:~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal/trapezoidal-1.0$ make
make all-recursive
make[1]: Entering directory '/home/project2/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal/trapezoidal-1.0'
Making all in src
make[2]: Entering directory '/home/project2/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal/trapezoidal-1.0/src'
gcc -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT trapezoidal.o -MD -MP -MF .deps/trapezoidal.Tpo -c -o trapezoidal.o t
rapezoidal.c
mv -f .deps/trapezoidal.Tpo .deps/trapezoidal.Po
gcc -g -O2 -o trapezoidal trapezoidal.o
make[2]: Leaving directory '/home/project2/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal/trapezoidal-1.0/src'
make[1]: Leaving directory '/home/project2/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal/trapezoidal-1.0'
make[1]: Leaving directory '/home/project2/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal/trapezoidal-1.0'
```

Figura 11: Generación del binario

```
trapezoidal-1.0 archives ready for distribution:
trapezoidal-1.0.tar.gz
project2@project2-VirtualBox:~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal/trapezoidal-1.0$
```

Figura 12: Compresión de la imagen

```
project2@project2-VirtualBox:~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal/trapezoidal-1.0$ ls
aclocal.m4  config.h  config.status  depcomp  Makefile.am  src
autom4te.cache  config.h.in  configure  install-sh  Makefile.in  stamp-h1
compile  config.log  configure.ac  Makefile  missing  trapezoidal-1.0.tar.gz
project2@project2-VirtualBox:~/Yocto/poky/meta-tec/recipes-trapezoidal/trapezoidal/trapezoidal-1.0$
```

Figura 13: Imagen en el formato tar.gz.

- Se procede a modificar el archivo layer.conf y la receta trapezoidal-1.0.bb. Posteriormente se puede observar como la jerarquía ya se encuentra como la necesitamos para generar la imagen con el autotool.

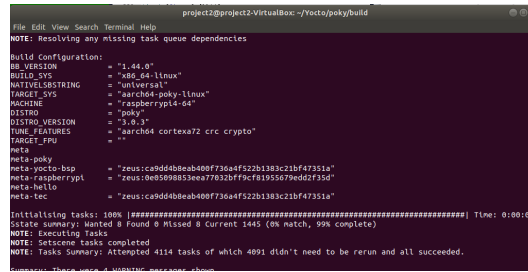
```
project2@project2-VirtualBox: ~/Yocto/poky/meta-tec
File Edit View Search Terminal Help
project2@project2-VirtualBox:~/Yocto/poky/meta-tec$ tree
.
├── conf
│   └── layer.conf
├── recipes-trapezoidal
│   └── trapezoidal
│       ├── trapezoidal-1.0
│       │   ├── trapezoidal-1.0.tar.gz
│       │   └── trapezoidal-1.0.bb
└── 4 directories, 3 files
```

Figura 14: Esquema final para generar la imagen con autotool

- Se procede a correr el comando `source oe-init-build-env`, y posteriormente el `btbake core-image-base` para generar la imagen de Yocto. Posteriormente se copia en la carpeta `/var/nfs/rootfs/` y está lista para ser cargada en la Raspberry4.

3. Analisis de resultados

Agregar respuesta



```
project2@project2-VirtualBox: ~/Yocto/poky/build
File Edit View Search Terminal Help
NOTE: Resolving any missing task queue dependencies
Build Configuration:
  BUILD_SYS      = "x86_64-linux"
  BUILD_TARGET   = "universal"
  DISTRO         = "raspberrypi4-64"
  MACHINE        = "poky"
  DISTRO_VERSION = "3.0.3"
  MACHINE_FEATURES = "armv7 cortexa72 crc crypto"
  TARGET_FPU     = ""
meta
meta-poky
meta-yocto-bsp = "zeus:ca9d64deab40ff730eaf522b1383c21bf47351a"
meta-raspberrypi = "zeus:9e5098853ee77632bffc9cf81955079ed2f31d"
meta-hello      = "zeus:ca9d64deab40ff730eaf522b1383c21bf47351a"
meta-tec
Installing tasks: 100% [#####] Time: 0:00:05
State summary: Wanted 0 Found 0 Missed 0 Current 1445 (0% match, 99% complete)
NOTE: Executing Tasks
NOTE: Setscene tasks completed
NOTE: Tasks Summary: Attempted 4114 tasks of which 4091 didn't need to be rerun and all succeeded.
Summary: 1396 tasks, 14091 tasks, 14091 tasks
```

Figura 15: Imagen de Yocto creada.

4. Problemas no resueltos

Agregar respuesta

Referencias

- [1] Raspberry pi 4 - device tree. Library Catalog: blog.stabel.family. [Online]. Available: <https://blog.stabel.family/raspberry-pi-4-device-tree/>
- [2] FrankBau/raspi-repo-manifest. Library Catalog: github.com. [Online]. Available: <https://github.com/FrankBau/raspi-repo-manifest>
- [3] Intel's management engine. Library Catalog: puri.sm. [Online]. Available: <https://puri.sm/learn/intel-me/>
- [4] "IEEE 100: the authoritative dictionary of IEEE standards terms."
- [5] M. J. M. Pelgrom, *Analog-to-digital conversion*, 2nd ed. Springer, OCLC: ocn838102317.
- [6] J. P. Uyemura, *CMOS logic circuit design*. Kluwer Academic Publishers.
- [7] M. Diallo, "Bootstrap circuitry selection for half bridge configurations," p. 10.
- [8] Microchip Technology Inc., "Cryptographic embedded controller - CEC1712." [Online]. Available: <http://ww1.microchip.com/downloads/en/DeviceDoc/CEC1712-Data-Sheet-DS00003416A.pdf>