

Customización del kernel de linux para raspberry pi 3 mediante yocto.

Autor

Christian Yáñez

Este trabajo ha sido realizado en el marco de la asignatura Implementación de sistemas operativos en agosto de 2019.

Tabla de contenido

Registros de cambios	3
1.- Objetivo	4
2.- Elementos	4
3.- Diagrama de bloques del hardware	4
4.- Yocto-Poky	4
5.- Metadata - Input	5
6.- Interfaz gráfica para configuración de poky (Toaster)	6
7.- Qt project	7
8.- Proceso de desarrollo.	7
8.1.- Información del sistema utilizado	7
8.2.- Descarga de repositorios base	8
8.3.- Directorio de trabajo.	8
8.4.- Inicialización de yocto	9
8.5.- Personalización de los archivos de configuración.	9
8.6.- Construcción de la imagen.	10
8.6.1.- Uso de Toaster	11
8.7.- Copia de la imagen a la SD.	12
9.- Resultados.	12
10.- Fuentes de consulta.	16

Registros de cambios

Revisión	Detalle de los cambios realizados	Fecha
1.0	Creación del documento	23/08/2019
2.0	Correcciones	25/8/2019

1.- Objetivo

Implementar una distribución customizada del kernel de linux que permita el diseño de interfaces gráficas para la placa de desarrollo raspberry pi 3.

2.- Elementos

- Raspberry pi 3

<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>

- Kernel linux
- Yocto project

<https://www.yoctoproject.org/software-overview/>

- Qt project

<https://www.qt.io/developers/>

- Touch screen element14 para raspberry

<https://www.element14.com/community/docs/DOC-78156/1/raspberry-pi-7-touchscreen-display>

3.- Diagrama de bloques del hardware

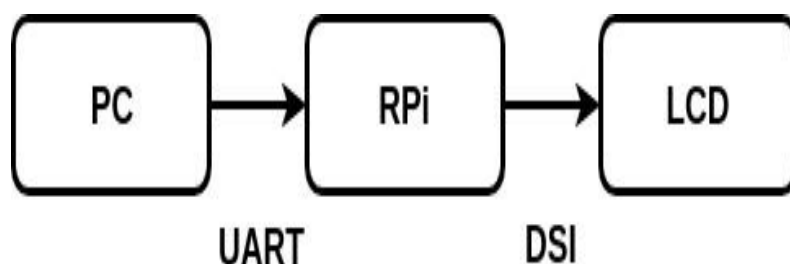


Figura 1.- Diagrama de bloques.

4.- Yocto-Poky

Yocto es un sistema para el desarrollo de distribuciones linux.

Poky es una distribución reducida de Yocto y OpenEmbedded, la figura 2 muestra la estructura y dependencia de la distribución poky.

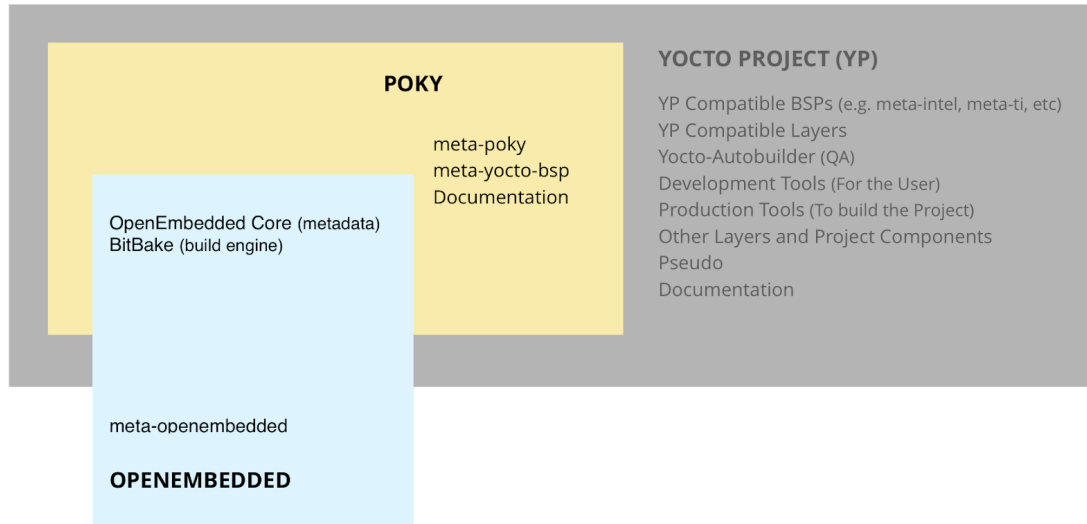


Figura 2.- Estructura general de poky.

Por otro lado la figura 3 muestra el flujo de trabajo de poky mediante la arquitectura OpenEmbedded.

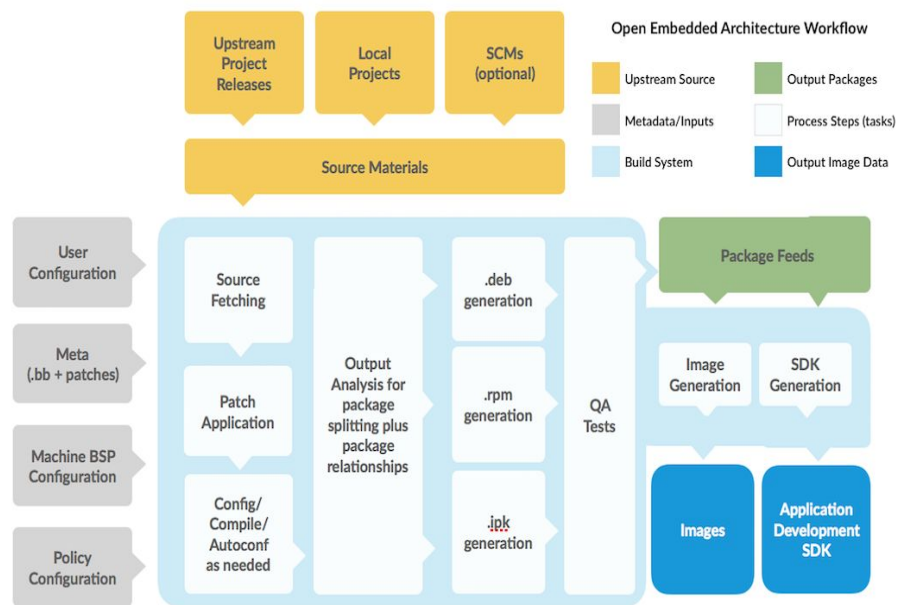


Figura 3.- Flujo de trabajo Poky-OpenEmbedded.

5.- Metadata - Input

La información necesaria para que poky construya la imagen solicitada la recoge de diferentes archivos conocidos como metadata, los principales se listan a continuación.

Recetas (.bb).- Contienen información sobre un elemento de software concreto, por ejemplo, parches, fuentes de compilación entre otros.

Clases (.bbclass).- Contienen la información compartida entre recetas.

Configuraciones (.conf).- Definen las variables de configuración que controlan el comportamiento de poky.

Capa (Meta-).- Es la agrupación de diferentes recetas que proporcionan alguna funcionalidad específica.

6.- Interfaz gráfica para configuración de poky (Toaster)

Toaster permite configurar y construir imágenes además de mostrar estadísticas del proceso.

La figura 4 muestra el ambiente de desarrollo de toaster.

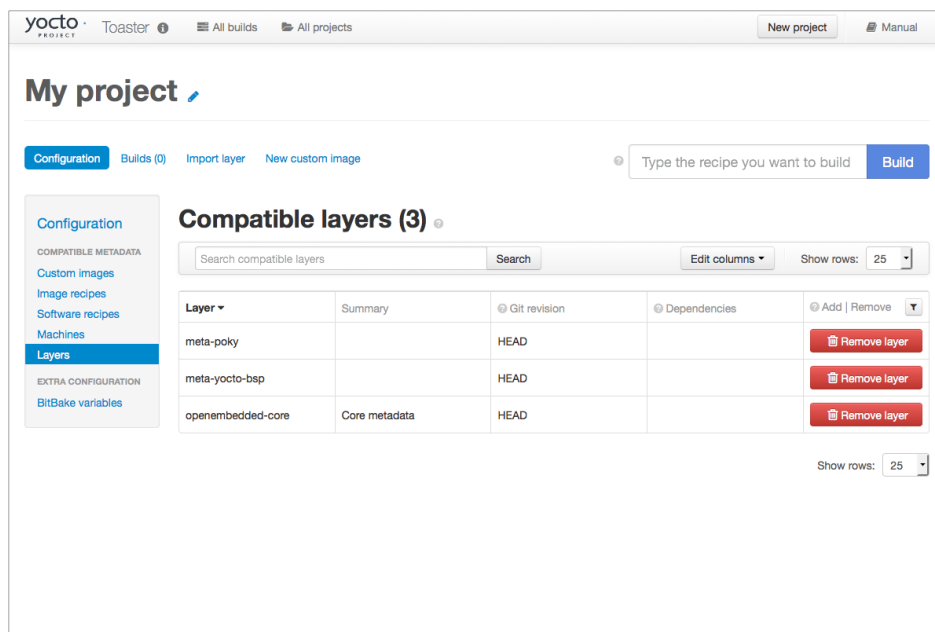


Figura 4.- Ambiente de desarrollo toaster.

Toaster permite seleccionar la placa de trabajo, luego muestra los layers y recetas compatibles con el hardware seleccionado de una gran lista de documentos disponibles desarrollados por la comunidad.

Seleccionamos los layers y recetas según las necesidades y finalmente presionamos el botón "Build" para generar la imagen.

Toaster se levanta localmente por medio de una interfaz html desde la carpeta de construcción del proyecto. (Más información en la sección “proceso de desarrollo” de éste documento o en el link siguiente).

<https://www.yoctoproject.org/software-item/toaster/>

7.- Qt project

Qt es un framework multiplataforma utilizado para el desarrollo de aplicaciones que utilizan interfaces gráficas.

Es un entorno altamente documentado y utilizado por las diferentes funcionalidades que ofrece. Por tal razón se proveerá a la imagen de linux creada la capacidad de levantar interfaces gráficas diseñadas mediante el entorno QT.

La figura 5 muestra el ambiente de desarrollo “Qt Creator” de QT.

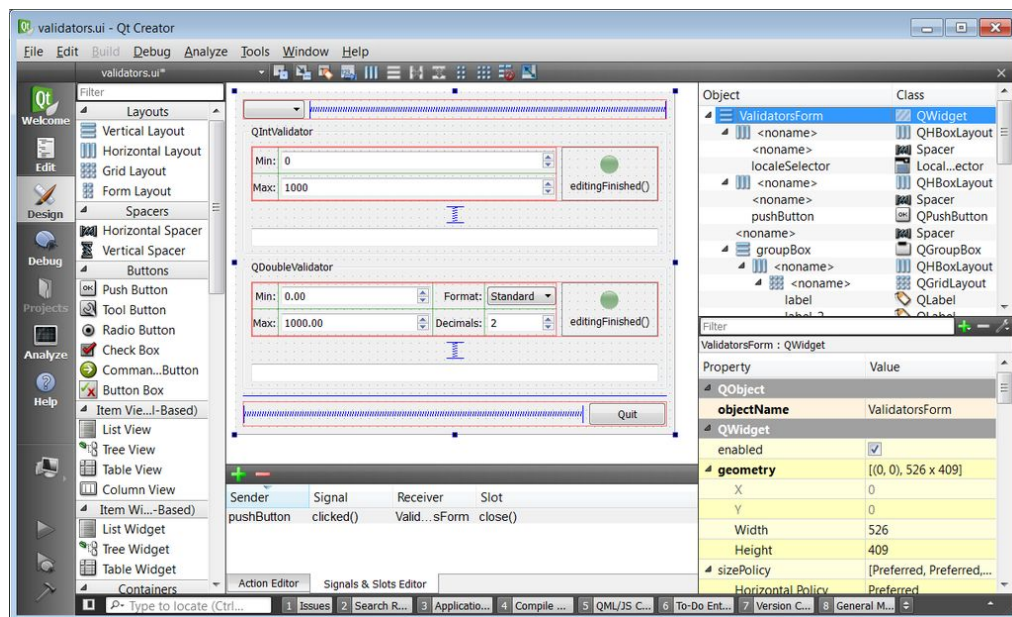


Figura 5.- Interfaz QT Creator.

8.- Proceso de desarrollo.

8.1.- Información del sistema utilizado

- Yocto - Poky versión 2.7
- Kernel linux versión 4.19
- Qt versión 5.12
- Python versión 3.7

8.2.- Descarga de repositorios base

Es necesario descargar dentro de nuestro directorio o carpeta de trabajo, las diferentes capas que poky integrará a la construcción de la imagen así como las capas necesarias para el funcionamiento de poky. A continuación las fuentes para las capas utilizadas en este proyecto.

Layer Yocto:

```
git clone git://git.yoctoproject.org/poky.git poky-warrior
```

Layer QT:

```
git clone https://github.com/meta-qt5/meta-qt5
```

Layer Openembedded:

```
git clone git://git.openembedded.org/meta-openembedded
```

Layer raspberry pi:

```
git clone git://git.yoctoproject.org/meta-raspberrypi
```

```
git clone -b warrior git://github.com/jumpnow/meta-rpi
```

Todos los layers incluidos en este proyecto son prediseñados o existentes los cuales se incluirán para simplificar el proceso, sin embargo se pueden desarrollar layers propios, para esto se sugiere referirse a :

<https://www.yoctoproject.org/docs/2.6/bsp-guide/bsp-guide.html>

8.3.- Directorio de trabajo.

La carpeta que contiene todo el desarrollo, en su interior lucirá según la figura 6.

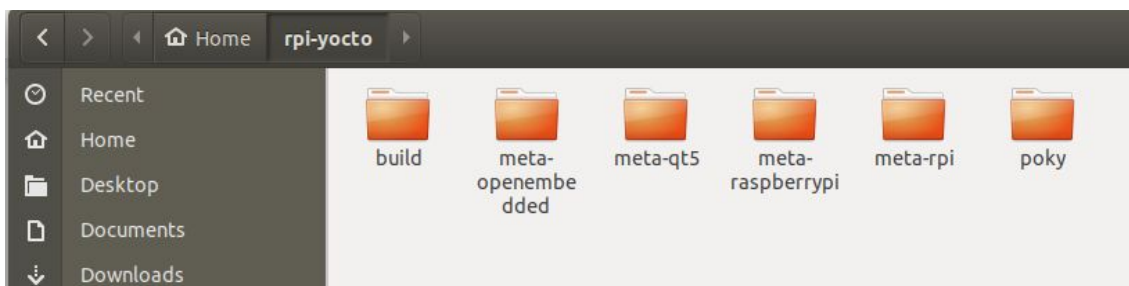


Figura 6.- Directorio de trabajo.

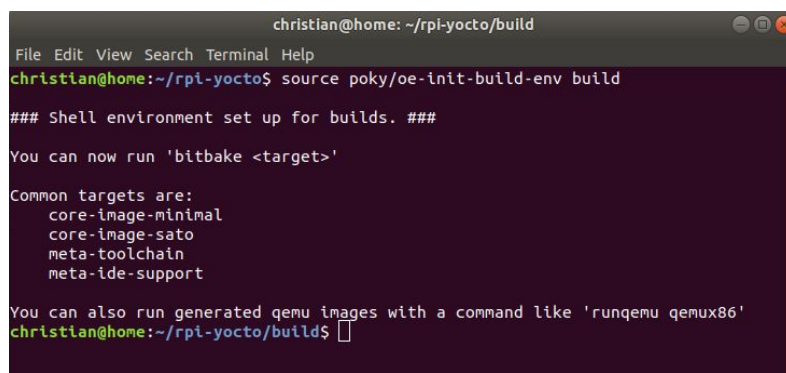
El directorio de trabajo contiene los layers descargados y la carpeta “build” en la cual poky guarda las configuraciones y resultados de su proceso.

8.4.- Inicialización de yocto

Dentro de la carpeta “yocto” se encuentra el script “oe-init-build-env” el cual es ejecutado mediante el comando source.

```
source poky/oe-init-build-env build
```

“build” corresponde al nombre de la carpeta donde se guarda la construcción del proyecto. El resultado de inicializar poky se muestra en la figura 7.



```
christian@home: ~/rpi-yocto/build
File Edit View Search Terminal Help
christian@home:~/rpi-yocto$ source poky/oe-init-build-env build

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
  core-image-minimal
  core-image-sato
  meta-toolchain
  meta-ide-support

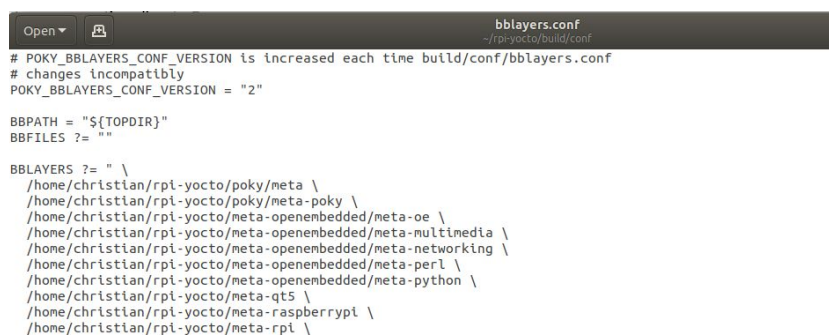
You can also run generated qemu images with a command like 'runqemu qemu86'
christian@home:~/rpi-yocto/build$
```

Figura 7.- Inicialización del ambiente poky.

8.5.- Personalización de los archivos de configuración.

Dentro del directorio build/conf, se generan los archivos bblayers.conf y local.conf.

El primer archivo contiene la dirección de los diferentes layers que poky utilizará para la construcción, en éste agregamos los necesarios según nuestras necesidades. La figura 8 muestra la configuración para el proyecto.



```
Open bblayers.conf ~/rpi-yocto/build/conf
# POKY_BBLAYERS_CONF_VERSION is increased each time build/conf/bblayers.conf
# changes incompatibly
POKY_BBLAYERS_CONF_VERSION = "2"

BBPATH = "${TOPDIR}"
BBFILES ?= ""

BBLAYERS ?= " \
/home/christian/rpi-yocto/poky/meta \
/home/christian/rpi-yocto/poky/meta-poky \
/home/christian/rpi-yocto/meta-openembedded/meta-oe \
/home/christian/rpi-yocto/meta-openembedded/meta-multimedia \
/home/christian/rpi-yocto/meta-openembedded/meta-networking \
/home/christian/rpi-yocto/meta-openembedded/meta-perl \
/home/christian/rpi-yocto/meta-openembedded/meta-python \
/home/christian/rpi-yocto/meta-qt5 \
/home/christian/rpi-yocto/meta-raspberrypi \
/home/christian/rpi-yocto/meta-rpi \
"
```

Figura 8.- archivo de configuración bblayers.conf.

El archivo `local.conf` contiene la configuración de nuestro target, por ejemplo, nombre de la placa, formato de la imagen de salida, habilitación de interfaces de comunicación, tipo de arquitectura, versión de kernel entre las más importantes.

Todas las posibles variables de configuración del archivo `local.conf` dependen de la confección específica del layer vinculado; por ejemplo, para el layer `meta-raspberrypi`, la documentación del enlace presenta las diferentes alternativas para la placa utilizada.

<https://buildmedia.readthedocs.org/media/pdf/meta-raspberrypi/latest/meta-raspberrypi.pdf>

La figura 9 muestra parte de la configuración del proyecto.

```
#MACHINE = "raspberrypi"
#MACHINE = "raspberrypi0"
#MACHINE = "raspberrypi0-wifi"
#MACHINE = "raspberrypi2"
#MACHINE = "raspberrypi3"
#MACHINE = "raspberrypi4"
#MACHINE = "raspberrypi-cm"
#MACHINE = "raspberrypi-cm3"

# Choices are Image or zImage if NOT using u-boot (no u-boot is the default)
# Choices are uImage or zImage if using u-boot, though if you choose zImage
# with u-boot you will also have to change the boot script boot command
KERNEL_IMAGETYPE = "zImage"

DISABLE_OVERSCAN = "1"
ENABLE_UART = "1"
ENABLE_RPI3_SERIAL_CONSOLE = "1"
SERIAL_CONSOLES = "115200;ttyAMA0"
# SERIAL_CONSOLES_forcevariable = ""

# comment this line if you want a 4.14 kernel
PREFERRED_VERSION_linux-raspberrypi = "4.19.%"

#DL_DIR = "${HOME}/oe-sources"
#SSTATE_DIR = "/oe4/rpi/sstate-cache"
#TMPDIR = "/oe4/rpi/tmp-warrior"

DISTRO = "poky"
PACKAGE_CLASSES = "package_ipk"

# i686 or x86_64
SDKMACHINE = "x86_64"
```

Figura 9.- Archivo de configuración `local.conf`

8.6.- Construcción de la imagen.

Una vez inicializado el ambiente `poky`, se procede a generar la imagen mediante el comando `"bitbake -nombre de la imagen"`. El nombre de la imagen corresponde a las diferentes recetas que `poky` utilizará para crear la imagen, por default `poky` muestra las recetas:

- `core-image-minimal`
- `core-image-sato`
- `meta-toolchain`
- `meta-ide-support`

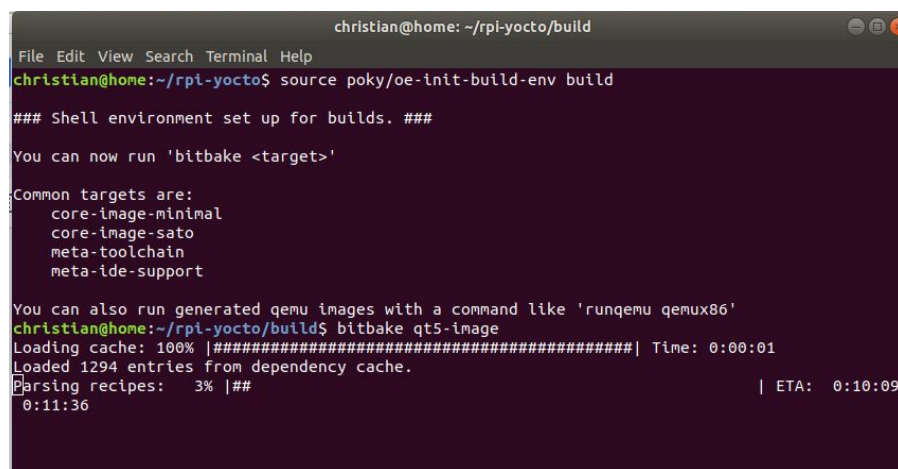
Para mayor información sobre estas y más recetas referirse a:

<https://layers.openembedded.org/layerindex/branch/master/layer/openembedded-core/>

Dichos nombres dependen también de cómo se diseñaron los layers, como usuarios podemos crear nuestras propias recetas; en el caso de este proyecto se utilizan recetas diseñadas por defecto dentro del layer meta-rpi diseñado para la implementación de linux con compatibilidad para qt mediante el comando:

```
bitbake qt5-image
```

La figura 10 muestra el proceso inicial de la construcción de la imagen del kernel customizado.



```
christian@home: ~/rpi-yocto/build
File Edit View Search Terminal Help
christian@home:~/rpi-yocto$ source poky/oe-init-build-env build
### Shell environment set up for builds. ###
You can now run 'bitbake <target>'

Common targets are:
  core-image-minimal
  core-image-sato
  meta-toolchain
  meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemu86'
christian@home:~/rpi-yocto/build$ bitbake qt5-image
Loading cache: 100% |#####| Time: 0:00:01
Loaded 1294 entries from dependency cache.
Parsing recipes: 3% |##
0:11:36 | ETA: 0:10:09
```

Figura 10.- Creación de la imagen customizada.

El proceso de construcción demora muchas horas dependiendo de los layers incluidos, las recetas utilizadas, conexión a internet y hardware del host. Para este caso particular 10 horas como promedio ya que se construyeron varias imágenes con diferentes configuraciones.

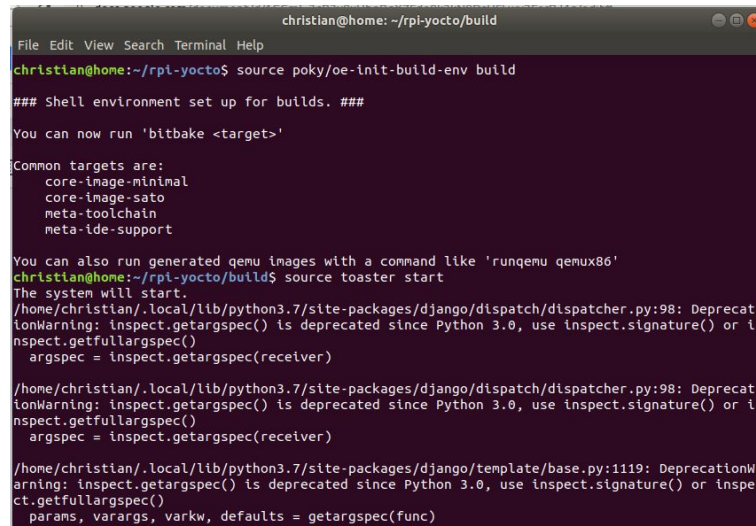
Sin embargo, una vez construida la imagen, las modificaciones se realizan en un tiempo mucho más corto.

8.6.1.- Uso de Toaster

En lugar de utilizar el comando bitbake, es posible usar la interfaz de toaster, para esto se usa el comando:

```
source toaster start
```

La figura 11 muestra la inicialización de la interfaz de toaster.



```
christian@home: ~/rpi-yocto/build
File Edit View Search Terminal Help
christian@home:~/rpi-yocto$ source poky/oe-init-build-env build

### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
  core-image-minimal
  core-image-sato
  meta-toolchain
  meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemu86'
christian@home:~/rpi-yocto/build$ source toaster start
The system will start.
/home/christian/.local/lib/python3.7/site-packages/django/dispatch/dispatcher.py:98: Deprecat
ionWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signature() or i
nspect.getfullargspec()
  argspec = inspect.getargspec(receiver)

/home/christian/.local/lib/python3.7/site-packages/django/dispatch/dispatcher.py:98: Deprecat
ionWarning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signature() or i
nspect.getfullargspec()
  argspec = inspect.getargspec(receiver)

/home/christian/.local/lib/python3.7/site-packages/django/template/base.py:1119: DeprecationW
arning: inspect.getargspec() is deprecated since Python 3.0, use inspect.signature() or inspe
ct.getfullargspec()
  params, varargs, varkw, defaults = getargspec(func)
```

Figura 11.- Inicialización del ambiente toaster.

Sin embargo, para este proyecto no fue posible utilizar toaster ya que se produjo problemas de compatibilidad entre las versiones de python y django elementos necesarios para levantar la interfaz.

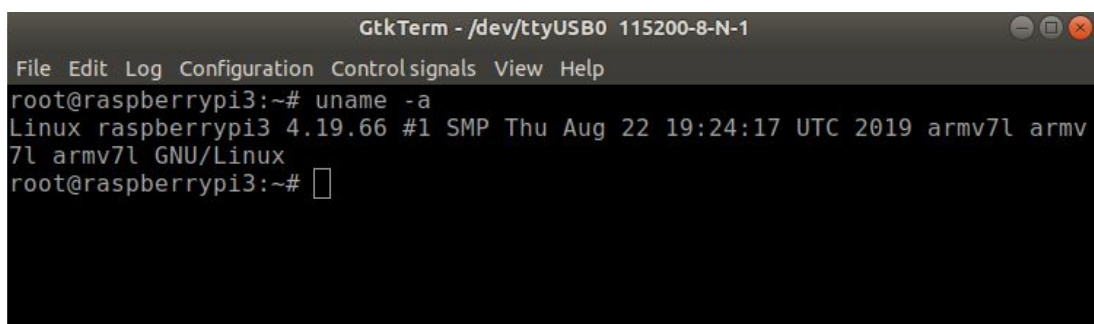
8.7.- Copia de la imagen a la SD.

Una vez terminado el proceso de construcción de la imagen, la misma se ubica dentro del directorio /home/usuario/rpi-yocto/build/tmp/deploy/images/raspberrypi3.

El archivo o la imagen “**qt5-image-raspberrypi3.rpi-sdimg**” puede ser directamente transferido mediante programas para flashear imágenes como ETCHER (el cual fue utilizado en este proyecto), puede ser copiado mediante comandos de consola o scripts según la preferencia del diseñador ya que en el directorio mencionado se almacenan todos los archivos necesarios para dicha tarea.

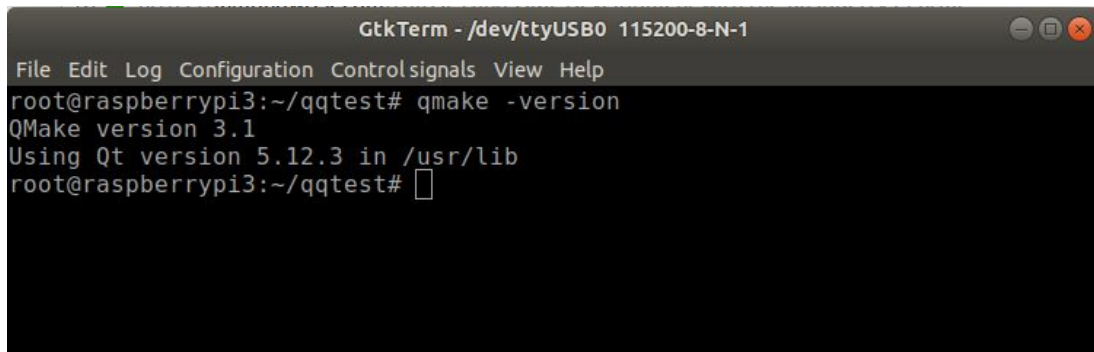
9.- Resultados.

Boot de la placa e identificación del sistema.



```
GtkTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Controls signals View Help
root@raspberrypi3:~# uname -a
Linux raspberrypi3 4.19.66 #1 SMP Thu Aug 22 19:24:17 UTC 2019 armv7l armv
7l armv7l GNU/Linux
root@raspberrypi3:~#
```

Versión de qt instalada.



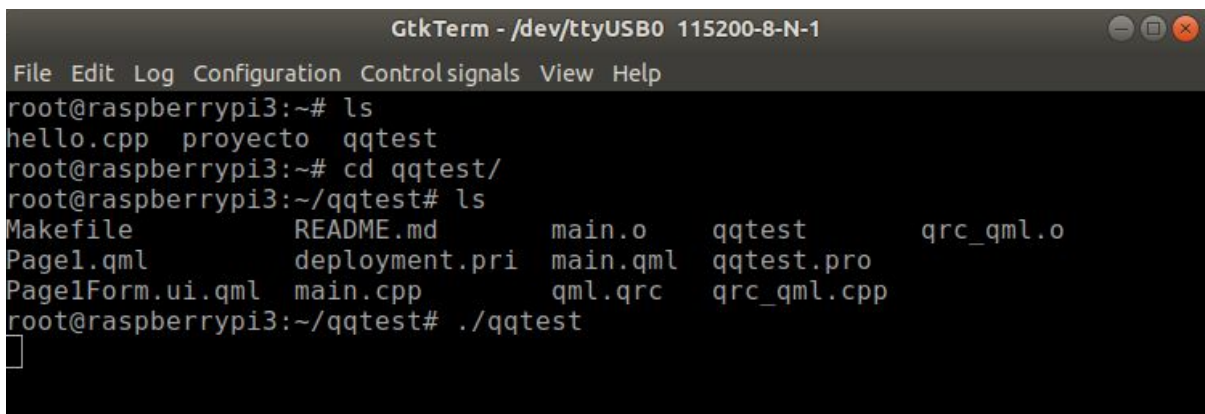
```
GtkTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Controls signals View Help
root@raspberrypi3:~/qqtest# qmake -version
QMake version 3.1
Using Qt version 5.12.3 in /usr/lib
root@raspberrypi3:~/qqtest#
```

En estas condiciones la placa está en capacidad de levantar interfaces diseñadas bajo el entorno qt.

Para esto, se copia la carpeta que contiene el proyecto de la interfaz gráfica (qqtest), dentro de esta esta el archivo main.cpp dentro del cual está el diseño de la interfaz, el proyecto debe ser levantado mediante el comando:

qmake && make

Posterior a esto se crea un ejecutable “qqtest” el cual levanta la interfaz grafica.



```
GtkTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Controls signals View Help
root@raspberrypi3:~# ls
hello.cpp  proyecto  qqtest
root@raspberrypi3:~# cd qqtest/
root@raspberrypi3:~/qqtest# ls
Makefile      README.md      main.o      qqtest      qrc_qml.o
Pagel.qml     deployment.pri main.qml     qqtest.pro
PagelForm.ui.qml main.cpp      qml.qrc     qrc_qml.cpp
root@raspberrypi3:~/qqtest# ./qqtest

```



```
GtkTerm - /dev/ttyUSB0 115200-8-N-1
File Edit Log Configuration Controlsignals View Help
root@raspberrypi3:~/qqtest# ./qqtest
Unable to query physical screen size, defaulting to 100 dpi.
To override, set QT_QPA_EGLFS_PHYSICAL_WIDTH and QT_QPA_EGLFS_PHYSICAL_HEIGHT (in millimeters).
qml: Button 1 clicked.
qml: Button 1 clicked.
qml: Button 1 clicked.
qml: Button 2 clicked.
qml: Button 2 clicked.
qml: Exit clicked
root@raspberrypi3:~/qqtest#
```

La interfaz muestra dos pestañas (First, Second) y tres botones los cuales imprimen en pantalla los mensajes:

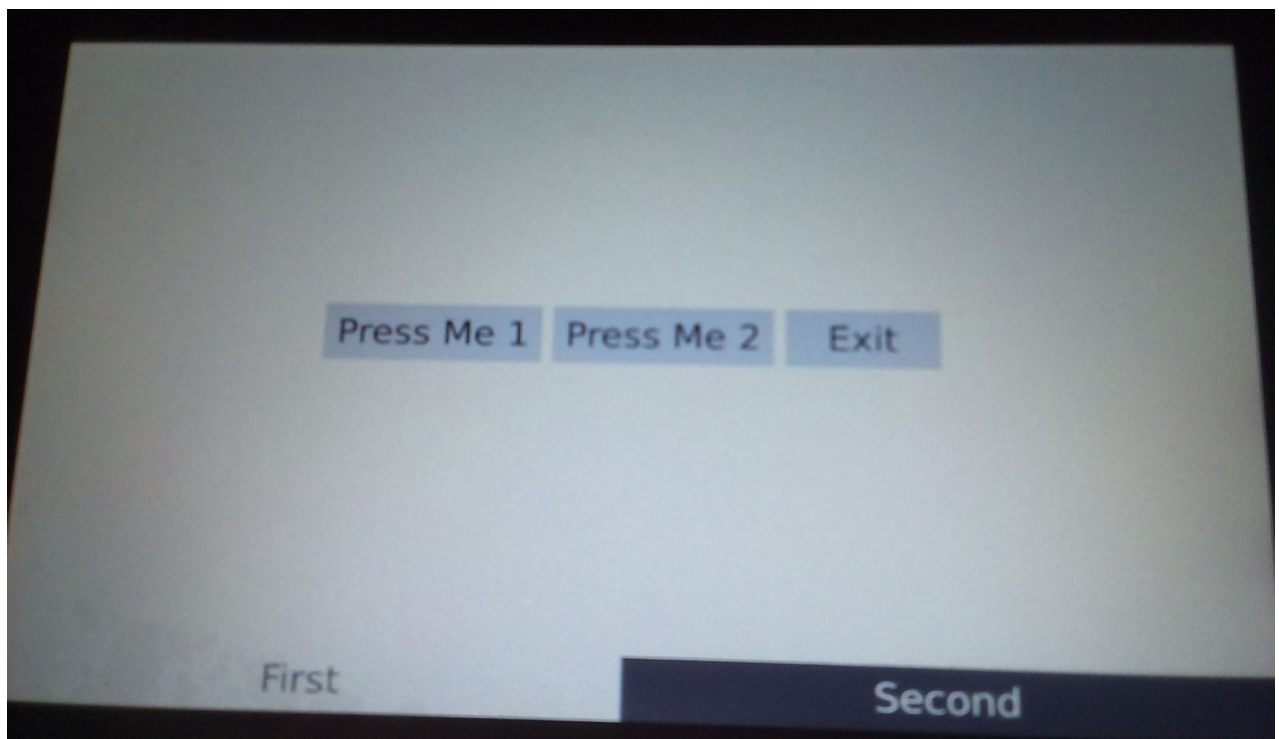
Button1 clicked

Button3 clicked

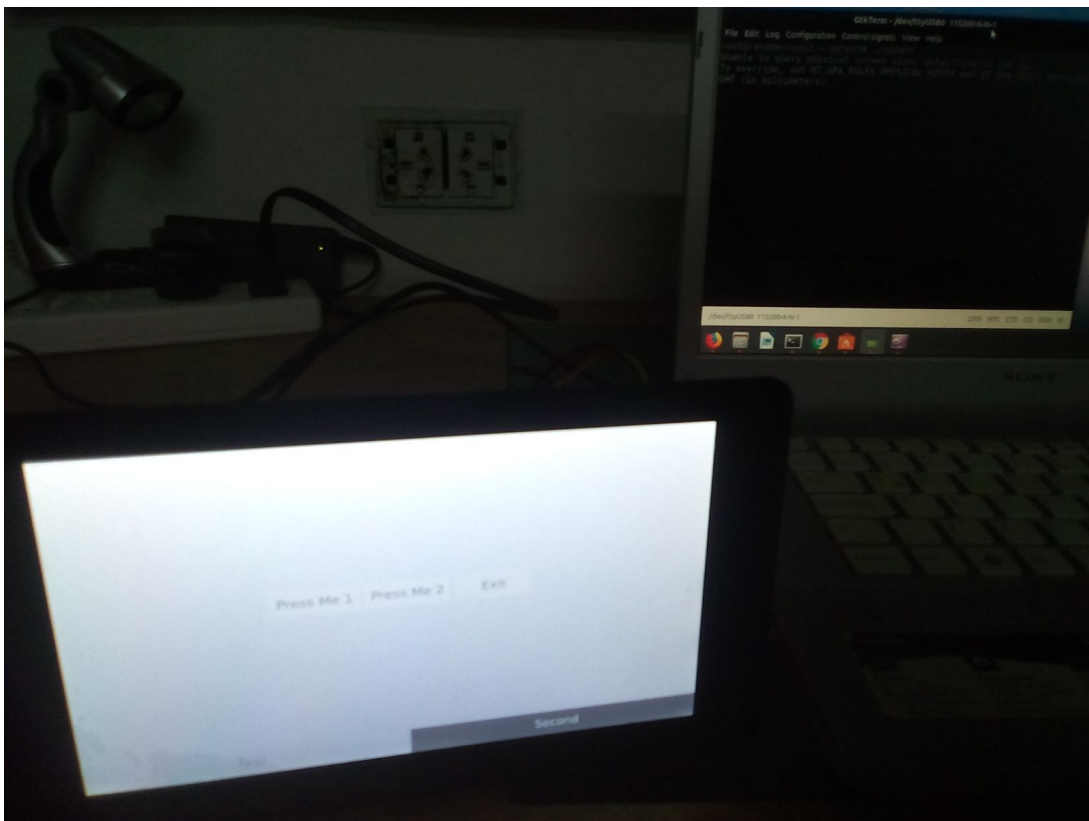
Exit clicked

respectivamente.

Aspecto de la interfaz.



Sistema final.



10.- Fuentes de consulta.

<https://jumpnowtek.com/rpi/Raspberry-Pi-Systems-with-Yocto.html>

<https://jumpnowtek.com/rpi/Qt5-and-QML-Development-with-the-Raspberry-Pi.html>

<https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>

<https://www.yoctoproject.org/software-overview/>

<https://www.qt.io/developers/>

<https://www.element14.com/community/docs/DOC-78156/1/raspberry-pi-7-touchscreen-display>

<https://www.yoctoproject.org/software-item/toaster/>

<https://www.yoctoproject.org/docs/2.6/bsp-guide/bsp-guide.html>

<https://buildmedia.readthedocs.org/media/pdf/meta-raspberrypi/latest/meta-raspberrypi.pdf>

<https://layers.openembedded.org/layerindex/branch/master/layer/openembedded-core/>