

Contenido

REPOSITORIO GIT CON LOS CÓDIGOS FUENTES	2
▪ REPOSITORIO GIT	2
▪ ESTRUCTURA DEL REPOSITORIO GIT	2
CONFIGURACIÓN DE LA IMAGEN DE DOCKER	4
▪ ARCHIVO DOCKERFILE.....	4
▪ PASOS PARA INSTALAR Y CONFIGURAR EL CONTENEDOR DE DOCKER.....	5
▪ SALIR DEL CONTENEDOR CREADO.....	6
▪ PASOS PARA VOLVER A PONER EN EJECUCION EL CONTENEDOR CREADO.....	7
▪ SIEMPRE INICIAR SSH EN DOCKER.....	7
▪ CONEXIÓN DE ECLIPSE AL CONTENDOR DE DOCKER ATRAVES DE SSH	8
EJECUCIÓN DE LOS EJEMPLOS	9
▪ LISTADO DE EJEMPLOS QUE SE PUEDEN EJECUTAR EN CADA PLACA	9
▪ FORMA PARA EJECUTAR LOS EJEMPLOS.....	9
A. PROYECTO BUTTON	9
B. PROYECTO BUTTON_INT.....	10
C. PROYECTO BLINKLED	11
D. PROYECTO ADC_SINGLE.....	12
E. PROYECTO FREERTOS_SINGLETHREAD	14
F. PROYECTO FREERTOS_MULTITHREAD	15
G. PROYECTO SOFTWARE_INT	16
H. PROYECTO SYSTICK	17
I. PROYECTO TIMER.....	18
J. PROYECTO UART_ECHO	20
K. PROYECTO UART_ECHO_INT	20
L. PROYECTO PRINTF_DEMO	21

REPOSITORIO GIT CON LOS CÓDIGOS FUENTES

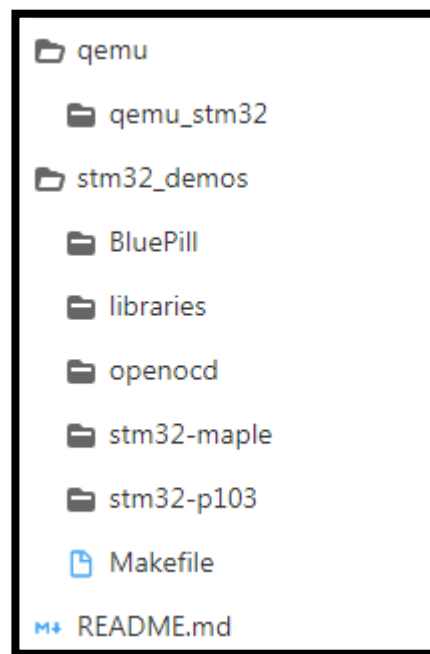
▪ REPOSITORIO GIT

Se creó un repositorio GIT, que contiene el código fuente de qemu creado por Beckus, que fue modificado para poder ejecutar algunos ejemplos correctamente. Además se agregaron ejemplos para ser ejecutados en las placas STM32-p103, STM32-Maple y Bluepill al ser simuladas en qemu y también en el HW físico. En este sentido, la siguiente es la URL del repositorio GIT creado.

<https://github.com/soaunlam2021/soa-entorno-integracion>

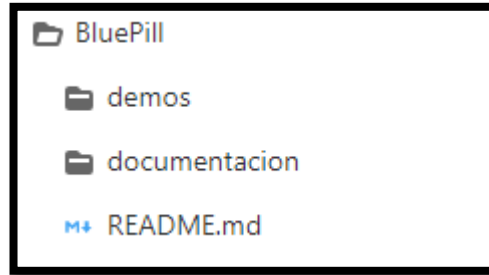
▪ ESTRUCTURA DEL REPOSITORIO GIT

El contenido del repositorio GIT se encuentra estructurado en los siguientes directorios y subdirectorios

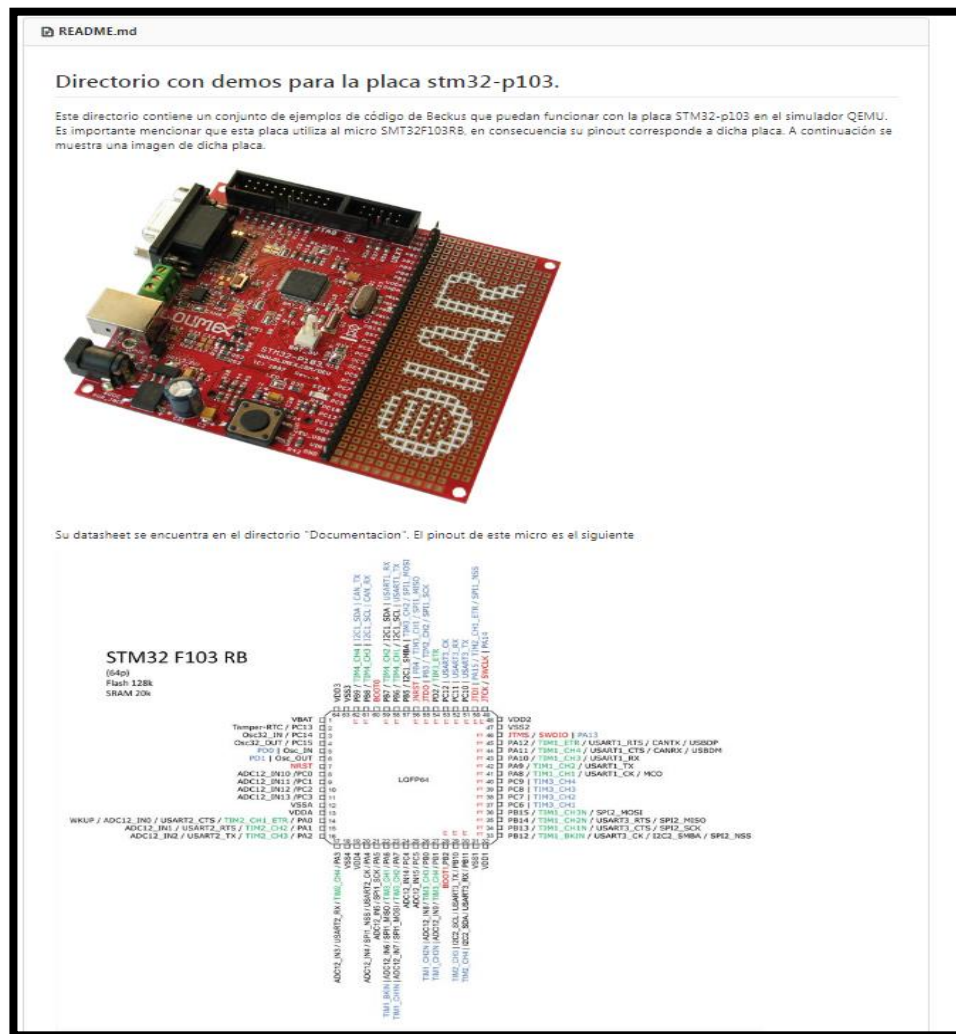


A continuación se describe el contenido de cada directorio

- ✚ **QEMU:** Este directorio contiene el código fuente de qemu que fue creado por Beckus, el cual fue modificado para poder ejecutar los ejemplos de mencionados en este tutorial.
- ✚ **STM32_DEMOS:** Este directorio contiene los subdirectorios en donde se encuentran los códigos de los ejemplos de las distintas placas para poder ser ejecutados en qemu. Cada uno de los subdirectorios (stm32-p103, stm32-maple y Bluepill) están estructurados internamente de la siguiente manera.



El subdirectorio **demos** contiene todos los códigos fuentes de los ejemplos para esa placa. Por otra parte el subdirectorio **documentación** contiene su datasheet y los archivos imágenes que emplea el archivo README.md para esa placa. En este archivo se detalla el PINOUT y las características del HW. Por ejemplo en la siguiente imagen se muestra el contenido del archivo README.md del subdirectorio stm32-p103



CONFIGURACIÓN DE LA IMAGEN DE DOCKER

■ ARCHIVO DOCKERFILE

Se creó un archivo DockerFile que permite generar en forma automatizada, un contenedor de Docker que contenga instalado todo lo necesario para poder ejecutar en qemu los distintos ejemplos para las placas mencionadas. En el siguiente link se encuentra el archivo DockerFile mencionado

<https://github.com/soaunlam2021/soa-entorno-integracion/blob/master/Dockerfile>

Dentro del archivo DockerFile se realizan las siguientes acciones dentro de una imagen de Docker.

- Se instala la versión de Ubuntu 20.04
- Se descargan todos los paquetes necesarios para poder ejecutar qemu sin problemas.
- Se descarga el repositorio de GIT que fue mencionado en el apartado anterior
- Se compila el código fuente de qemu, que fue descargado.
- Se compilan todos los códigos fuentes de los ejemplos de las 3 placas mencionadas (stm32-p103, stm32-maple y Bluepill)
- Se realiza la configuración necesaria para poder acceder vía ssh al contenedor de Docker creado.

En la siguiente imagen se muestra con mayor detalle su contenido

```
#se descarga la version de Ubuntu 20.04
from ubuntu:20.04

ARG DEBIAN_FRONTEND='noninteractive'

#instalo los paquetes necesario de Ubuntu
RUN apt-get update && apt-get install -y --no-install-recommends \
    apt-utils \
    gcc-arm-none-eabi \
    libnewlib-arm-none-eabi \
    findutils \
    gcc \
    git \
    python2.7 \
    pkgconf \
    libglib2.0-dev \
    libbfd-dev \
    libpixman-1-dev \
    zlib1g-dev \
    make \
    pkgconf \
    openssh-client \
    psmisc \
    net-tools \
    iputils-ping \
    ca-certificates \
    ssh \
    && rm -rf /var/lib/apt/lists/* #se borra la cache de apt para liberar espacio

#descargo del repositorio de SOA la imagen de QEMU que hizo beckus, que fue modificada para poder ejecutar
#eficientemente en las placas Bluepill, STM32-p103 y STM32-Maple
RUN cd / && \
    git clone https://github.com/soaunlam2021/soa-entorno-integracion.git

#Compile and create image of qemu
RUN cd /soa-entorno-integracion/qemu/qemu_stm32 && \
    chmod 555 configure && \
    ./configure --extra-cflags="-w" --enable-debug --target-list="arm-softmmu" --python=/usr/bin/python2.7 && \
    make

#compilo todos los ejemplo para las ejecutar en las disintas placas
RUN cd /soa-entorno-integracion/stm32_demos && \
    make

#se modifica el archivo sshd_config, para poder acceder por sshs al Docker
RUN sed -i 's/#PermitRootLogin prohibit-password/PermitRootLogin yes/g' /etc/ssh/sshd_config && \
    sed -i 's/#PasswordAuthentication/PasswordAuthentication/g' /etc/ssh/sshd_config

#se establece el password root
RUN echo 'root:1234' | chpasswd

#agrego comandos que se ejecutaran cada vez que se inicia el contenedor
RUN echo alias qemu-system-arm="/soa-entorno-integracion/qemu/qemu_stm32/arm-softmmu/qemu-system-arm" >> /root/.bashrc && \
    echo echo "Comando qemu-system-arm creado." >> /root/.bashrc &&
```

■ PASOS PARA INSTALAR Y CONFIGURAR EL CONTENEDOR DE DOCKER

1º. Habilitar Internet dentro de Docker

El primer paso necesario es habilitar conexión a internet dentro de las imágenes de Docker, para de esta forma poder descargar los paquetes necesarios de Ubuntu dentro de ellas. Para eso se deben los siguientes subpasos. Estos pasos los saque del siguiente foro

<https://github.com/kubernetes-sigs/kubespray/issues/213>

a) En el Host de Linux abrir el siguiente archivo

/usr/lib/systemd/system/docker.service

b) Agregar las siguientes líneas dentro de ese archivo

- ExecStartPre=/usr/bin/ip link set dev docker0 down
- ExecStartPre=/usr/sbin/brctl delbr docker0

Ejemplo de cómo agregar esas líneas en el archivo docker.services:

```
[Unit]
Description=Docker Application Container Engine
Documentation=http://docs.docker.com
After=network.target
Wants=docker-storage-setup.service

[Service]
Type=notify
EnvironmentFile=/etc/default/docker
ExecStartPre=/usr/bin/ip link set dev docker0 down
ExecStartPre=/usr/sbin/brctl delbr docker0
Environment=GOTRACEBACK=crash
ExecStart=/usr/bin/docker daemon \
    $OPTIONS \
    $DOCKER_STORAGE_OPTIONS \
    $DOCKER_NETWORK_OPTIONS \
    $INSECURE_REGISTRY
LimitNOFILE=1048576
LimitNPROC=1048576
LimitCORE=infinity
MountFlags=slave
TimeoutStartSec=1min

[Install]
WantedBy=multi-user.target
```

c) Guardar los cambios en el archivo y luego desde la línea de comandos de bash reiniciar el servicio de docker con los siguientes comandos:

- systemctl daemon-reload
- systemctl restart docker

2º. Crear una imagen de ejecutando el archivo DockerFile

Se debe crear la imagen del Docker con todo lo necesario para poder ejecutar los ejemplos. Para ello se debe utilizar el archivo DockerFile, mencionado previamente, en la siguiente sentencia en el Host de Linux.

docker build --tag soaqemu:v2 --file DockerFile .

NOTA: El proceso de creación de imagen tarda aproximadamente unos 20 minutos.

3º. Verificar si la imagen se creo correctamente.

Para ello se debe utilizar el siguiente comando:

`docker images`

```
esteban@ubuntu:~$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
soaqemu	v2	e2201fea1792	4 days ago	1.52GB
ubuntu	20.04	f643c72bc252	4 weeks ago	72.9MB
soaqemu	v1	6ed6bdd53fa5	2 months ago	2.83GB
ubuntu	latest	9140108b62dc	3 months ago	72.9MB

```
esteban@ubuntu:~$
```

4º. Asociarle un contenedor a la imagen creada

Para ello se debe ejecutar el siguiente comando:

`docker run -it soaqemu:v2`

Tras su ejecución se abrirá el contenedor y se mostrará una consola de linux que contendrá todo lo instalado para poder ejecutar los ejemplos. Este contendrá los siguientes directorios instalados en su raíz.

```
root@6736d6f2ac3f:/# ls -l
total 52
lrwxrwxrwx 1 root root 7 Nov 6 01:21 bin -> usr/bin
drwxr-xr-x 2 root root 4096 Apr 15 2020 boot
drwxr-xr-x 5 root root 360 Dec 28 22:51 dev
drwxr-xr-x 1 root root 4096 Dec 28 22:51 etc
drwxr-xr-x 2 root root 4096 Apr 15 2020 home
lrwxrwxrwx 1 root root 7 Nov 6 01:21 lib -> usr/lib
lrwxrwxrwx 1 root root 9 Nov 6 01:21 lib32 -> usr/lib32
lrwxrwxrwx 1 root root 9 Nov 6 01:21 lib64 -> usr/lib64
lrwxrwxrwx 1 root root 10 Nov 6 01:21 libx32 -> usr/libx32
drwxr-xr-x 2 root root 4096 Nov 6 01:21 media
drwxr-xr-x 2 root root 4096 Nov 6 01:21 mnt
drwxr-xr-x 2 root root 4096 Nov 6 01:21 opt
dr-xr-xr-x 299 root root 0 Dec 28 22:51 proc
drwx----- 2 root root 4096 Nov 6 01:25 root
drwxr-xr-x 1 root root 4096 Dec 23 22:26 run
lrwxrwxrwx 1 root root 8 Nov 6 01:21/sbin -> usr/sbin
drwxr-xr-x 1 root root 4096 Dec 23 22:23 soa-entorno-integracion
drwxr-xr-x 2 root root 4096 Nov 6 01:21 srv
dr-xr-xr-x 13 root root 0 Dec 28 22:51 sys
drwxrwxrwt 1 root root 4096 Dec 23 22:26 tmp
drwxr-xr-x 1 root root 4096 Nov 6 01:21 usr
drwxr-xr-x 1 root root 4096 Nov 6 01:25 var
root@6736d6f2ac3f:/#
```

Como se puede observar el contenedor ya tiene descargado e instalado el contenido del repositorio GIT, en el directorio **so-entorno-integración**

5º. Iniciar el servicio de SSH

Cada vez que se inicia el contenedor, es necesario iniciar manualmente el servicio de ssh. Para ello es necesario ejecutar el siguiente comando

`service ssh start`

■ SALIR DEL CONTENEDOR CREADO

Se debe ejecutar el siguiente comando Cuando deseamos salir del contenedor creado

`Exit`

```
root@6736d6f2ac3f:/# exit
exit
esteban@ubuntu:~$
```

Nota: Cuando se sale del contenedor con exit, este detiene su ejecución.

■ PASOS PARA VOLVER A PONER EN EJECUCION EL CONTENEDOR CREADO.

Cuando se sale del contenedor con exit, este detiene su ejecución. Con lo cual, para poder ponerlo en ejecución nuevamente se debe realizar los siguientes pasos

1º. Saber cual es el contenedor al que se asoció la imagen creada

Para ello debemos saber su ID. Esto se hace ejecutando el siguiente comando en el Host de Linux

```
docker ps -a
```

Una vez ejecutado sabremos cual es el ID del contenedor, como se muestra en la siguiente imagen

```
esteban@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
5736d6f2ac3f   soademu:v2    "/bin/bash"            28 minutes ago Exited (0)    15 minutes ago happy_tharp
9c91ee1051de   6ed6bdd53fa5  "/bin/bash"            6 days ago    Exited (0)    28 minutes ago relaxed_burnell
9acc6643e80f   6ed           "/bin/bash"            2 months ago  Exited (0)    5 days ago    jolly_khorana
esteban@ubuntu:~$
```

2º. Poner en ejecución nuevamente el contenedor creado

Para ello se debe ejecutar el siguiente comando en el Host de Linux

```
Docker start ID_contenedor
```

3º. Ingresar a la consola del contenedor

Para ello se debe ejecutar el siguiente comando en el Host de Linux

```
Docker attach ID_contenedor
```

OJO si ejecuto run, en lugar de attach

Si ejecuto docker run -it nuevamente, se va a crear un CONTENEDOR NUEVO con la imagen de cero. Con lo cual los cambios que hayamos realizado anteriormente en el contenedor no se verán, ya que estaremos trabajando en un contenedor nuevo. No obstante los cambios no se perderán.

■ SIEMPRE INICIAR SSH EN DOCKER

Es necesario siempre iniciar manualmente el servicio de ssh cada vez que se ingresa a la consola del contenedor. De esta forma se podrá acceder en forma remota al Docker y se podrá editar sus archivos con Eclipse. Para ellos se deben seguir los siguientes pasos.

1º. Iniciar el servicio de ssh

Para ello se debe ejecutar el siguiente comando dentro de Docker

```
service ssh start
```

2º. Comprobar la conexión ssh

Una vez iniciado el servicio de ssh en Docker, desde host de Linux se debe comprobar la conexión con ssh de la siguiente forma

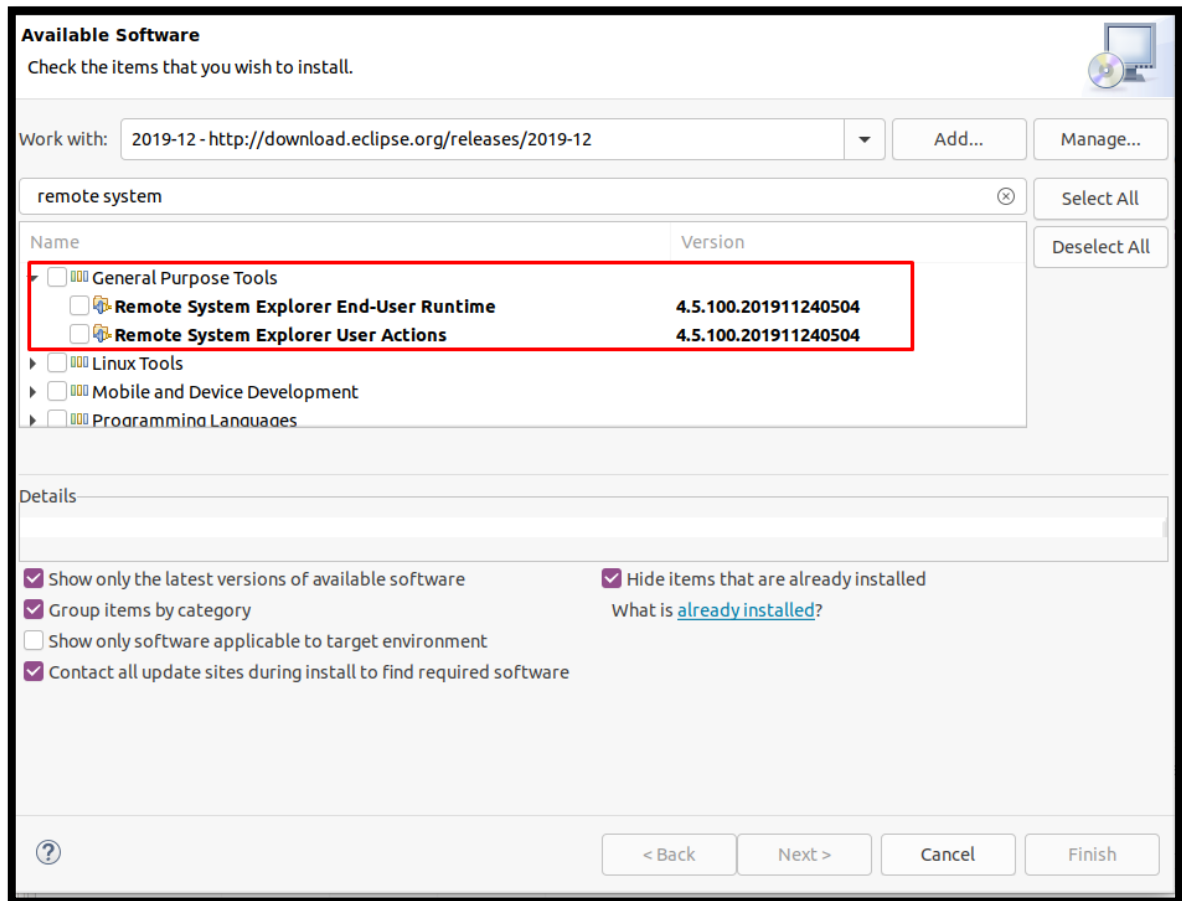
ssh root@172.17.0.3

NOTA1: La contraseña de root del Docker es "1234"

NOTA2: 172.17.0.3 es la ip de Docker

■ CONEXIÓN DE ECLIPSE AL CONTENEDOR DE DOCKER ATRAVÉS DE SSH

Para que sea más cómoda la edición de los archivos que hay dentro de un contenedor de Docker, se puede conectar Eclipse a este a través de ssh. Para ello es necesario instalar el plugin de Eclipse llamado **Remote System Explorer**.






Una vez instalado el Plugin se podrá editar los archivos de Docker en forma remota a través de ssh. Para ello se deber a seguir los pasos mostrados en el siguiente video

<https://www.youtube.com/watch?v=1f-HUvODMb0>

EJECUCIÓN DE LOS EJEMPLOS

■ LISTADO DE EJEMPLOS QUE SE PUEDEN EJECUTAR EN CADA PLACA

Se debió modificar los códigos fuentes de los ejemplos que fueron desarrollados por Beckus para poder ejecutarse correctamente en forma simulada en qemu. De esta forma, estos ejemplos se pudieron ejecutar hasta el momento en las siguientes tres placas de desarrollo

-  STM32-P103
-  STM32-MAPLE (SMT32F103RB)
-  BLUEPILL (STM32F103C8T6)

En consecuencia, no todos los ejemplos que fueron desarrollados por Beckus, funcionan correctamente en todas las placas. Por ese motivo, en la siguiente tabla se muestran los ejemplos que funcionan satisfactoriamente dependiendo del embebido utilizado

PLACAS		
STM32-P103	STM32-MAPLE	BLUEPILL
1. adc_single 2. Blink_flash 3. DAC (No entiendo como funciona) 4. FreeRTOS_Streambuffer 5. FreeRTOS_SingleThread 6. FreeRTOS_multithread 7. printf_demo 8. software_int 9. SysTick 10. timer 11. uart_echo 12. uart_echo_int (funciona pero pierde caracteres) 13. uart_repeat_write 14. button 15. button_int	1. adc_single 2. Blink_flash 3. DAC (No entiendo como funciona) 4. FreeRTOS_Streambuffer 5. FreeRTOS_SingleThread 6. FreeRTOS_multithread 7. printf_demo 8. software_int 9. SysTick 10. timer 11. uart_echo 12. uart_echo_int (funciona pero pierde caracteres) 13. uart_repeat_write 14. button 15. button_int	1. adc_single 2. Blink_flash 3. FreeRTOS_Streambuffer 4. FreeRTOS_SingleThread 5. FreeRTOS_multithread 6. printf_demo 7. software_int 8. SysTick 9. timer 10. uart_echo 11. uart_echo_int (funciona pero pierde caracteres) 12. uart_repeat_write 13. button 14. button_int

■ FORMA PARA EJECUTAR LOS EJEMPLOS

A continuación se detalla en forma la forma para ejecutar los ejemplos de beckus en las distintas placas.

A. PROYECTO BUTTON

Este ejemplo lee los botón es utilizando pooling

1. **Dentro de docker ejecutar el siguiente comando**

```
cd /soa-entorno-integracion/stm32_demos
```

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

2. **Después ejecutar el binario del ejemplo en qemu**

Dependiendo de la placa que se desee simular en qemu se debe utilizar el siguiente comando

Para STM32-p103:

```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/button/main.bin -serial tcp::7777,server -nographic
```

Para STM32-maple:

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/button/main.bin -serial tcp::7777,server -nographic
```

Para Bluepill:

```
qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/button/main.bin -serial tcp::7777,server -nographic
```

3. **Luego desde Ubuntu ejecutar en putty con los siguientes parámetros**
 - **Telnet**
 - **IP:** ip de docker
 - **Puerto:** 7777
4. **Ingresar al Monitor de Qemu de la siguiente forma :Mientras se esté ejecutando qemu dentro de docker presionar la siguiente secuencia de teclas para ingresar al monitor de qemu**
 - a. Presionar al mismo tiempo las teclas CTRL+A y soltarla
 - b. Luego presionar la tecla C y soltarlo
 - c. Luego presionar la tecla ENTER
 - d. Se mostrara el prompt del monitor de qemu de la siguiente forma
(qemu)
5. **Desde la consola de Qemu simular que se presiona el botón a través del siguiente comando**
sendkey b
6. **A medida que se enviando el comando anterior se ira prendiendo y apagando el LED**

```
root@6736d6f2ac3f:/soa-entorno-integracion/stm32_demos# ./soa-entorno-integracion/qemu/qemu_stm32-arm-softmmu/qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/button/main.bin -serial tcp::7777,server -nographic
QEMU 2.1.3 monitor - type 'help' for more information
(qemu) QEMU waiting for connection on: tcp:0.0.0.0:7777,server
button -1765964752 LED Off
main-loop: WARNING: I/O thread spun for 1000 iterations

(qemu) sendkey b
(qemu) LED Off
sendkey b
(qemu) LED On
sendkey b
(qemu) LED Off
sendkey b
(qemu) LED On
q
```

B. PROYECTO BUTTON INT

Este ejemplo lee los botones utilizando interrupciones

1. **Dentro de docker ejecutar el siguiente comando**

```
cd /soa-entorno-integracion/stm32_demos
```

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

7. **Después ejecutar el binario del ejemplo en qemu**

Dependiendo de la placa que se desee simular en qemu, se debe utilizar el siguiente comando

Para STM32-p103:

```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/button_int/main.bin -serial tcp::7777,server -nographic
```

Para STM32-maple:

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/button_int/main.bin -serial tcp::7777,server -nographic
```

Para Bluepill:

```
qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/button_int/main.bin -serial tcp::7777,server -nographic
```

8. **Luego desde Ubuntu ejecutar en putty con los siguientes parámetros**

- **Telnet**
- **IP:** ip de docker
- **Puerto:** 7777

9. **Ingresar al Monitor de Qemu de la siguiente forma :Mientras se este ejecutando qemu dentro de docker presionar la siguiente secuencia de teclas para ingresar al monitor de qemu**

- a. Presionar al mismo tiempo las teclas CTRL+A y soltarla
- b. Luego presionar la tecla C y soltarlo
- c. Luego presionar la tecla ENTER
- d. Se mostrara el prompt del monitor de qemu de la siguiente forma

(qemu)

10. **Desde la consola de Qemu simular que se presiona el botón a través del siguiente comando**

```
sendkey b
```

11. **A medida que se enviando el comando anterior se ira prendiendo y apagando el LED**

```
root@6736d6f2ac3f:/soa-entorno-integracion/stm32_demos# /soa-entorno-integracion/qemu/qemu_stm32/arm-softmmu/qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/button/main.bin -serial tcp::7777,server -nographic
QEMU 2.1.3 monitor - type 'help' for more information
(qemu) QEMU waiting for connection on: tcp:0.0.0.0:7777,server
button -1765964752 LED Off
main-loop: WARNING: I/O thread spun for 1000 iterations

(qemu) sendkey b
(qemu) LED Off
sendkey b
(qemu) LED On
sendkey b
(qemu) LED Off
sendkey b
(qemu) LED On
q
```

C. PROYECTO BLINKLED

1. **Dentro de docker ejecutar el siguiente comando**

```
cd /soa-entorno-integracion/stm32_demos
```

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

2. **Después ejecutar el binario del ejemplo en qemu**

Dependiendo de la placa que se desee simular en qemu se debe utilizar el siguiente comando

Para STM32-p103:

```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/blink_flash/main.bin -serial tcp::7777,server -nographic
```

Para STM32-maple:

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/blink_flash/main.bin -serial tcp::7777,server -nographic
```

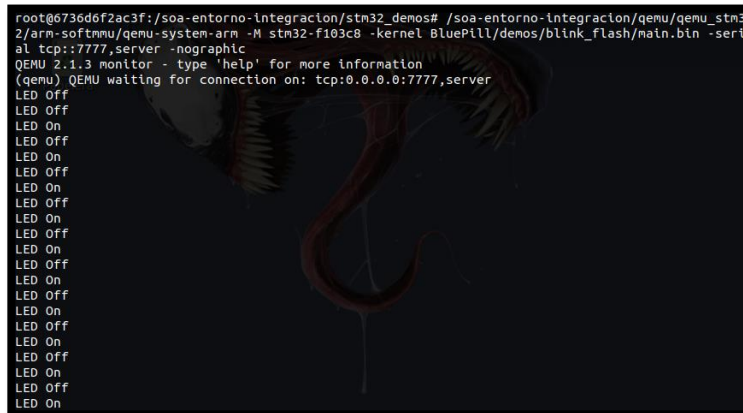
Para Bluepill:

```
qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/blink_flash/main.bin -serial tcp::7777,server -nographic
```

3. **Luego desde Ubuntu ejecutar en putty con los siguientes parámetros**



- a. **Telnet**
- b. **IP:** ip de docker
- c. **Puerto:** 7777

4. **Una vez realizada la conexión con putty, en la pantalla de docker se ira prendiendo y apagando el LED**



D. PROYECTO ADC SINGLE

Este ejemplo permite simular sensores analógicos. En este sentido el programa puede ser configurado para funcionar en los distintos modos. Estos modos pueden ser cambiados de dos maneras distintas:

-  **Desde el código fuente:** Para ello se debe cambiar el valor de la variable **mode**, que se encuentra dentro del archivo main.c.
-  **Durante la ejecución:** Se puede cambiar de modo durante la ejecución mediante la pulsación del botón. En Qemu eso se puede realizar ejecutando desde su consola el comando **“sendkey b”**, esto se debe realizar de la misma manera de cómo se detalló en el ejemplo del botón.

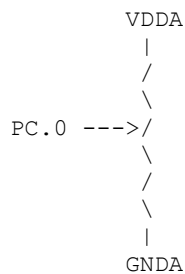
A continuación se detallan los distintos modos de simulación en que se pueden configurar los sensores analógicos.

There are several modes. To scroll through the modes, press the button on the development board:

- Mode 1** - Print the value of the internal temperature sensor.
Also attempts to calculate the actual temperature using the formula in the reference manual. On my board, this gives a high temperature which I assume is wrong. Further investigation is needed.
- Mode 2** - Print value of the internal voltage reference.
- Mode 3** - Print value on the external pin PC.0
- Mode 4** - A special mode that does not use the ADC, but prints the clock configuration. This is useful for debugging.

All numbers are hexadecimal except for the degrees Celsius temperature.

One way to drive pin PC.0 with an analog signal is to use a potentiometer as follows:



PASOS DE EJECUCIÓN

1. **Dentro de docker configurar el modo de ejecución en el archivo main.c. Por defecto el mode está en 1.**
2. **Dentro de docker ejecutar el siguiente comando**

```
cd /soa-entorno-integracion/stm32_demos
```

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

3. **Luego recompilar el código ejecutando el comando**

```
Make
```

De esta forma se generará el binario con los cambios realizados en el archivo fuente.

4. **Después ejecutar el binario del ejemplo en qemu**

Dependiendo de la placa que se desee simular en qemu se debe utilizar el siguiente comando

Para STM32-p103:

```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/adc_single/main.bin -serial tcp::7777,server -nographic
```

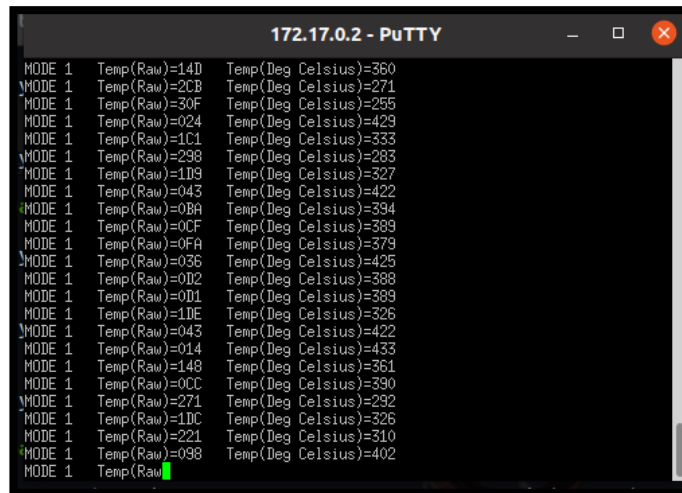
Para STM32-maple:

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/adc_single/main.bin -serial tcp::7777,server -nographic
```

Para Bluepill:

```
qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/adc_single/main.bin -serial tcp::7777,server -nographic
```

5. **Luego desde Ubuntu ejecutar en putty con los siguientes parámetros**
 - a. **Telnet**
 - b. **IP:** ip de docker
 - c. **Puerto:** 7777
6. **Una vez realizada la conexión con putty, en la putty se empezarán a ver los valores de la simulación del sensor analógico**



```
MODE 1 Temp(Raw)=14D Temp(Deg Celsius)=360
MODE 1 Temp(Raw)=2CB Temp(Deg Celsius)=271
MODE 1 Temp(Raw)=30F Temp(Deg Celsius)=255
MODE 1 Temp(Raw)=024 Temp(Deg Celsius)=429
MODE 1 Temp(Raw)=1C1 Temp(Deg Celsius)=333
MODE 1 Temp(Raw)=298 Temp(Deg Celsius)=285
MODE 1 Temp(Raw)=1D9 Temp(Deg Celsius)=327
MODE 1 Temp(Raw)=043 Temp(Deg Celsius)=422
MODE 1 Temp(Raw)=0BA Temp(Deg Celsius)=394
MODE 1 Temp(Raw)=0CF Temp(Deg Celsius)=389
MODE 1 Temp(Raw)=0FA Temp(Deg Celsius)=379
MODE 1 Temp(Raw)=036 Temp(Deg Celsius)=425
MODE 1 Temp(Raw)=0D2 Temp(Deg Celsius)=388
MODE 1 Temp(Raw)=0D1 Temp(Deg Celsius)=389
MODE 1 Temp(Raw)=1DE Temp(Deg Celsius)=326
MODE 1 Temp(Raw)=043 Temp(Deg Celsius)=422
MODE 1 Temp(Raw)=014 Temp(Deg Celsius)=433
MODE 1 Temp(Raw)=148 Temp(Deg Celsius)=361
MODE 1 Temp(Raw)=0CC Temp(Deg Celsius)=390
MODE 1 Temp(Raw)=271 Temp(Deg Celsius)=292
MODE 1 Temp(Raw)=1DC Temp(Deg Celsius)=326
MODE 1 Temp(Raw)=221 Temp(Deg Celsius)=310
MODE 1 Temp(Raw)=098 Temp(Deg Celsius)=402
```

E. PROYECTO FREERTOS SINGLETHREAD

1. **Dentro de docker ejecutar el siguiente comando**

```
cd /soa-entorno-integracion/stm32_demos
```

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

2. **Después ejecutar el binario del ejemplo en qemu**

Dependiendo de la placa que se desee simular en qemu se debe utilizar el siguiente comando

Para STM32-p103:

```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/freertos_singlethread/main.bin -serial tcp::7777,server -nographic
```

Para STM32-maple:

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/freertos_singlethread/main.bin -serial tcp::7777,server -nographic
```

Para Bluepill:

```
qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/freertos_singlethread/main.bin -serial tcp::7777,server -nographic
```

3. **Luego desde Ubuntu ejecutar en putty con los siguientes parámetros**
 - a. **Telnet**

- b. **IP:** ip de docker
 - c. **Puerto:** 7777
4. **Una vez realizada la conexión con putty, en la pantalla de docker se ira prendiendo y apagando el LED**

F. PROYECTO FREERTOS_MULTITHREAD

- **Hilo 1:** Este thread prende y apaga el led y se ve en la pantalla de Docker
- **Hilo 2:** Este thread muestra en la pantalla de Putty **"Hello 1"**
- **Hilo 3:** Este thread muestra en la pantalla de Putty **"Hello 2"**
- **Hilo 2:** Este thread realiza un eco de lo que se ingresa en la pantalla de putty mostrando el dato en la misma pantalla

1. Dentro de docker ejecutar el siguiente comando

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

Dependiendo de la placa que se desee simular en qemu se debe utilizar el siguiente comando


```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/freertos_multithread/main.bin -serial tcp::7777,server -nographic
```

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/freertos_multithread/main.bin -serial tcp::7777,server -nographic
```

```
qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/freertos_multithread/main.bin -serial
tcp::7777,server -nographic
```

- a. **Telnet**
- b. **IP:** ip de docker
- c. **Puerto:** 7777

- Una vez realizada la conexión con putty, en la pantalla de docker se ira prendiendo y apagando el LED. Mientras que en la pantalla de putty se ira mostrando las impresiones de los hilos 1 y 2, además del eco de lo que se ingrese por pantalla



G. PROYECTO SOFTWARE INT

Este ejemplo genera cada determinado tiempo una interrupción por software que prende y apaga el led

PASOS DE EJECUCIÓN

- Dentro de docker ejecutar el siguiente comando

```
cd /soa-entorno-integracion/stm32_demos
```

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

- Después ejecutar el binario del ejemplo en qemu

Dependiendo de la placa que se desee simular en qemu se debe utilizar el siguiente comando

Para STM32-p103:

```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/software_int/main.bin -serial tcp::7777,server -nographic
```

Para STM32-maple:

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/software_int/main.bin -serial tcp::7777,server -nographic
```

Para Bluepill:

```
qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/software_int/main.bin -serial tcp::7777,server -nographic
```

- Luego desde Ubuntu ejecutar en putty con los siguientes parámetros

- Telnet
- IP: ip de docker
- Puerto: 7777

- Una vez realizada la conexión con putty, en la pantalla de docker se ira prendiendo y apagando el LED


```
root@6736d6f2ac3f:/soa-entorno-integracion/stm32_demos# ./soa-entorno-integracion/qemu/qemu_stm32/arm-softmmu/qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/blink_flash/main.bin -serial tcp::7777,server -nographic
QEMU 2.1.3 monitor - type 'help' for more information
(qemu) QEMU waiting for connection on: tcp:0.0.0.0:7777,server
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
```

H. PROYECTO SYSTICK

Este ejemplo genera cada determinado tiempo una interrupción por hardware del timer que prende y apaga el led

PASOS DE EJECUCIÓN

1. **Dentro de docker ejecutar el siguiente comando**

```
cd /soa-entorno-integracion/stm32_demos
```

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

2. **Después ejecutar el binario del ejemplo en qemu**

Dependiendo de la placa que se desee simular en qemu se debe utilizar el siguiente comando

Para STM32-p103:

```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/systick/main.bin -serial tcp::7777,server -nographic
```

Para STM32-maple:

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/systick/main.bin -serial tcp::7777,server -nographic
```

Para Bluepill:

```
qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/systick/main.bin -serial tcp::7777,server -nographic
```

3. **Luego desde Ubuntu ejecutar en putty con los siguientes parámetros**
 - a. **Telnet**
 - b. **IP:** ip de docker
 - c. **Puerto:** 7777
4. **Una vez realizada la conexión con putty, en la pantalla de docker se ira prendiendo y apagando el LED**

```
root@6736d6f2ac3f:/soa-entorno-integracion/stm32_demos# ./soa-entorno-integracion/qemu/qemu_stm32/arm-softmmu/qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/blink_flash/main.bin -serial tcp::7777,server -nographic
QEMU 2.1.3 monitor - type 'help' for more information
(qemu) QEMU waiting for connection on: tcp:0.0.0.0:7777,server
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
LED Off
LED On
```

I. PROYECTO TIMER

Este ejemplo consiste en un contador de tiempo que puede variar su comportamiento dependiendo del modo en que este configurado durante su ejecución. De esta forma puede comportarse de manera incremental o decremental, entre otras.

Se puede cambiar de modo durante la ejecución mediante la pulsación del botón. En Qemu eso se puede realizar ejecutando desde su consola el comando ***"sendkey b"***, esto se debe realizar de la misma manera de cómo se detalló en el ejemplo del botón.

A continuación se detallan los distintos modos de simulación en que se pueden configurar el contador de tiempo.

```
Uses the timer to count up and down.

There are several modes. To scroll through the modes, press the button on
the development board:
Mode 1 - Count up for five seconds. Once five seconds have elapsed, the
        LED is toggled and the count resets to 1.
Mode 2 - Similar to Mode 1, but counts down instead of up.
Mode 3 - Similar to Modes 1 and 2, but alternately counts up and then down.
Mode 4 - Counts in one-shot mode. Each time the count is finished, the
        timer will be re-enabled to do another one shot count.
Mode 4 - Counts up, but when the count reaches 3, the program interrupts the
        count using the UG flag. This resets the count back to 1.
Mode 6 - A special mode that does not use the timer, but prints
        the clock configuration. This is useful for debugging.

All numbers are in decimal.

* Note that the timer in QEMU STM32 does not behave exactly like real
  hardware. In particular, the counting does not always go in the right
  direction, and the one shot mode does not appear to work properly.
```

NOTA: Por defecto se encuentra configurado en Modo 1

PASOS DE EJECUCIÓN

1. Dentro de docker ejecutar el siguiente comando

```
cd /soa-entorno-integracion/stm32_demos
```

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

2. Después ejecutar el binario del ejemplo en qemu

Dependiendo de la placa que se desee simular en qemu se debe utilizar el siguiente comando

Para STM32-p103:

```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/timer/main.bin -serial tcp::7777,server -nographic
```

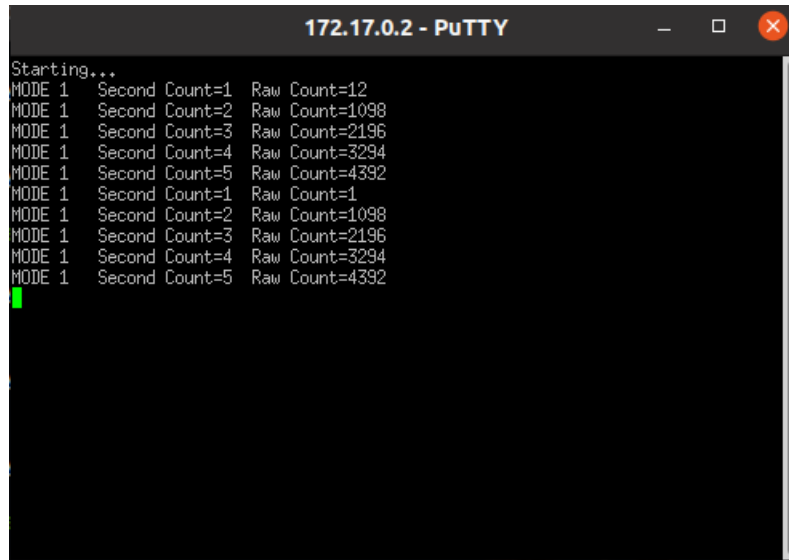
Para STM32-maple:

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/timer/main.bin -serial tcp::7777,server -nographic
```

Para Bluepill:

```
qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/timer/main.bin -serial tcp::7777,server -nographic
```

3. **Luego desde Ubuntu ejecutar en putty con los siguientes parámetros**
 - a. **Telnet**
 - b. **IP:** ip de docker
 - c. **Puerto:** 7777
4. **Una vez realizada la conexión con putty, en la pantalla de putty se mostrará cómo se incrementa el contador por tiempo en el Modo 1 de ejecución**



```
Starting...
MODE 1 Second Count=1 Raw Count=12
MODE 1 Second Count=2 Raw Count=1098
MODE 1 Second Count=3 Raw Count=2196
MODE 1 Second Count=4 Raw Count=3294
MODE 1 Second Count=5 Raw Count=4392
MODE 1 Second Count=1 Raw Count=1
MODE 1 Second Count=2 Raw Count=1098
MODE 1 Second Count=3 Raw Count=2196
MODE 1 Second Count=4 Raw Count=3294
MODE 1 Second Count=5 Raw Count=4392
```

12. **Si se desea cambiar de modo durante la ejecución, se deberá ingresar al Monitor de Qemu de la siguiente forma :**

Mientras se esté ejecutando qemu dentro de docker presionar la siguiente secuencia de teclas para ingresar al monitor de qemu

 - a. Presionar al mismo tiempo las teclas CTRL+A y soltarla
 - b. Luego presionar la tecla C y soltarlo
 - c. Luego presionar la tecla ENTER
 - d. Se mostrara el prompt del monitor de qemu de la siguiente forma

(qemu)
13. **Desde la consola de Qemu simular que se presiona el botón a través del siguiente comando**

sendkey b

J. PROYECTO UART ECHO

Este ejemplo recibe los caracteres enviados desde putty, a través de pooling y los reenvía nuevamente a putty utilizando otro pooling

PASOS DE EJECUCIÓN

1. **Dentro de docker ejecutar el siguiente comando**

```
cd /soa-entorno-integracion/stm32_demos
```

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

2. **Después ejecutar el binario del ejemplo en qemu**

Dependiendo de la placa que se desee simular en qemu se debe utilizar el siguiente comando

Para STM32-p103:

```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/uart_echo/main.bin -serial tcp::7777,server -nographic
```

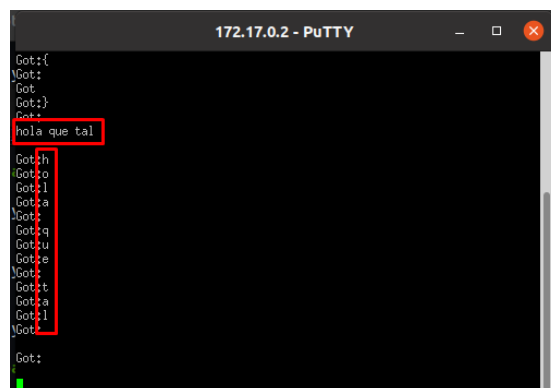
Para STM32-maple:

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/uart_echo/main.bin -serial tcp::7777,server -nographic
```

Para Bluepill:

```
qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/uart_echo/main.bin -serial tcp::7777,server -nographic
```

3. **Luego desde Ubuntu ejecutar en putty con los siguientes parámetros**
 - a. **Telnet**
 - b. **IP:** ip de docker
 - c. **Puerto:** 7777
4. **Una vez realizada la conexión con putty, en la pantalla de putty se podrán ingresar caracteres que se reimprimirán en la misma pantalla. De esta forma se estará haciendo un eco a través de pooling**



K. PROYECTO UART ECHO INT

Este ejemplo recibe los caracteres enviados desde putty, a través de interrupciones y los reenvía nuevamente a putty utilizando otras interrupciones.

NOTA: Cuidado con este ejemplo que funciona pero en el echo se pierden algunos caracteres

PASOS DE EJECUCIÓN

1. **Dentro de docker ejecutar el siguiente comando**

```
cd /soa-entorno-integracion/stm32_demos
```

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

2. **Después ejecutar el binario del ejemplo en qemu**

Dependiendo de la placa que se desee simular en qemu se debe utilizar el siguiente comando

Para STM32-p103:

```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/uart_echo_int/main.bin -serial tcp::7777,server -nographic
```

Para STM32-maple:

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/uart_echo_int/main.bin -serial tcp::7777,server -nographic
```

3. **Luego desde Ubuntu ejecutar en putty con los siguientes parámetros**

- a. **Telnet**
- b. **IP:** ip de docker
- c. **Puerto:** 7777

4. **Una vez realizada la conexión con putty, en la pantalla se podrán ingresar caracteres que se reimprimirán en la misma pantalla. De esta forma se estará haciendo un eco a través de interrupciones**

L. PROYECTO PRINTF DEMO

Este ejemplo utiliza una función similar a printf, la cual se encuentra desarrollada dentro del archivo myprintf.h. Esta función, envía una extensa cadena de string vía UART, para que posteriormente sea impresa en la pantalla de putty

PASOS DE EJECUCIÓN

1. **Dentro de docker ejecutar el siguiente comando**

```
cd /soa-entorno-integracion/stm32_demos
```

De esta forma nos posicionaremos en el directorio adecuado para ejecutar el ejemplo

2. **Después ejecutar el binario del ejemplo en qemu**

Dependiendo de la placa que se desee simular en qemu se debe utilizar el siguiente comando

Para STM32-p103:

```
qemu-system-arm -M stm32-p103 -kernel stm32-p103/demos/printf_demo/main.bin -serial tcp::7777,server -nographic
```

Para STM32-maple:

```
qemu-system-arm -M stm32-maple -kernel stm32-maple/demos/printf_demo/main.bin -serial tcp::7777,server -nographic
```

Para Bluepill:

```
qemu-system-arm -M stm32-f103c8 -kernel BluePill/demos/printf_demo/main.bin -serial tcp::7777,server -nographic
```

3. **Luego desde Ubuntu ejecutar en putty con los siguientes parámetros**
 - a. **Telnet**
 - b. **IP:** ip de docker
 - c. **Puerto:** 7777
4. **Una vez realizada la conexión con putty, en la pantalla de putty se mostrará varias veces la misma cadena de string**



```
test num 4660=0x1234 str Strings ch Z
test num 4660=0x1234 str Strings ch Z
test num 4660=0x1234 str Strings ch Z
test num 4660=0x1234 str Strings ch Z
test num 4660=0x1234 str Strings ch Z
test num 4660=0x1234 str Strings ch Z
test num 4660=0x1234 str Strings ch Z
test num 4660=0x1234 str Strings ch Z
test num 4660=0x1234 str Strings ch Z
test num 4660=0x1234 str Strings ch Z
test num 4660=0x1234 str Strings ch Z
test num 4660=0x1234 str Strings ch Z
```