Sistemas Críticos

Tema 2:
FreeRTOS

Lección 6:
Desarrollo de aplicaciones de tiempo real con *FreeRTOS* en la *Zybo*

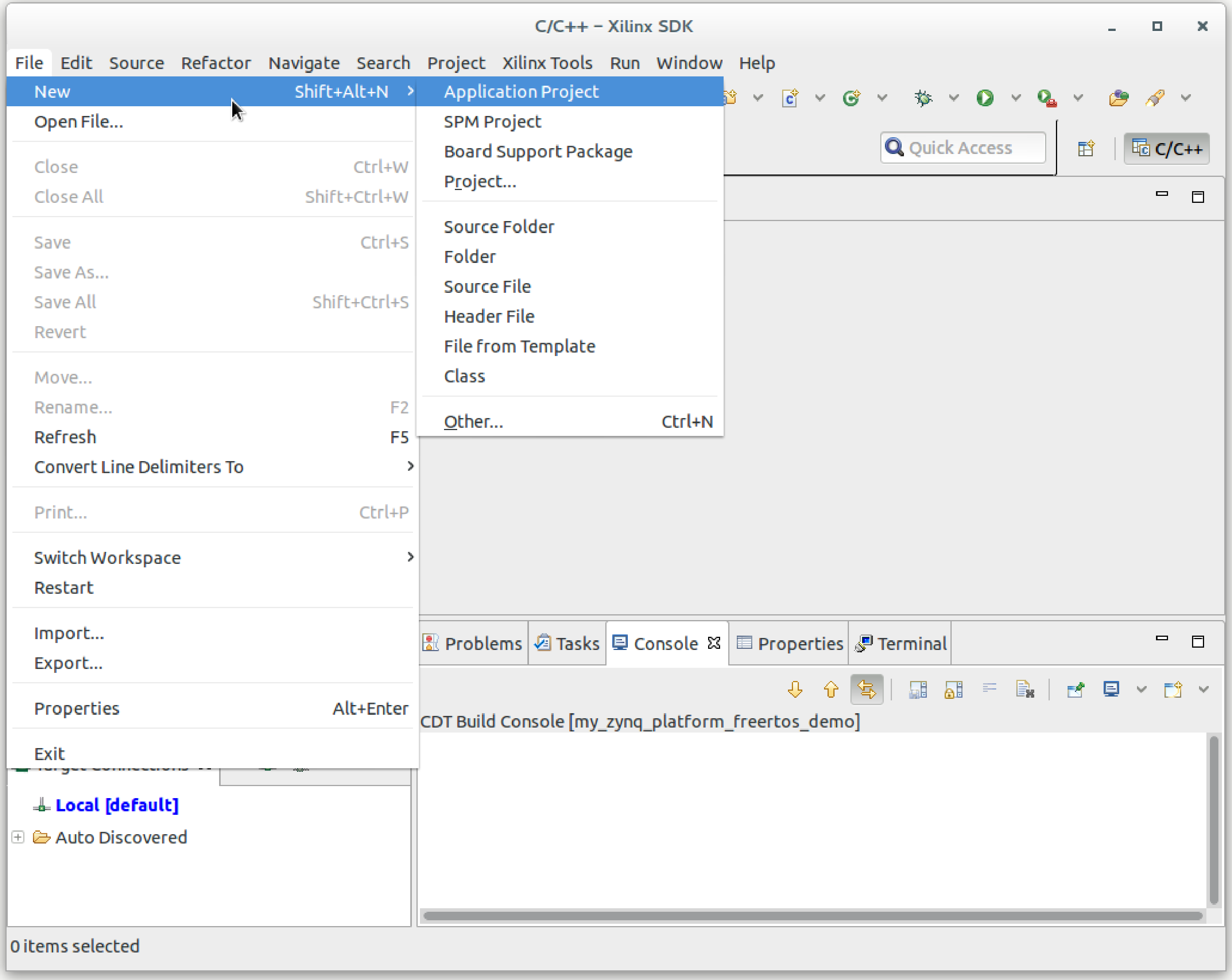
Jesús González Peñalver

Contenidos

Tema 2: *FreeRTOS*

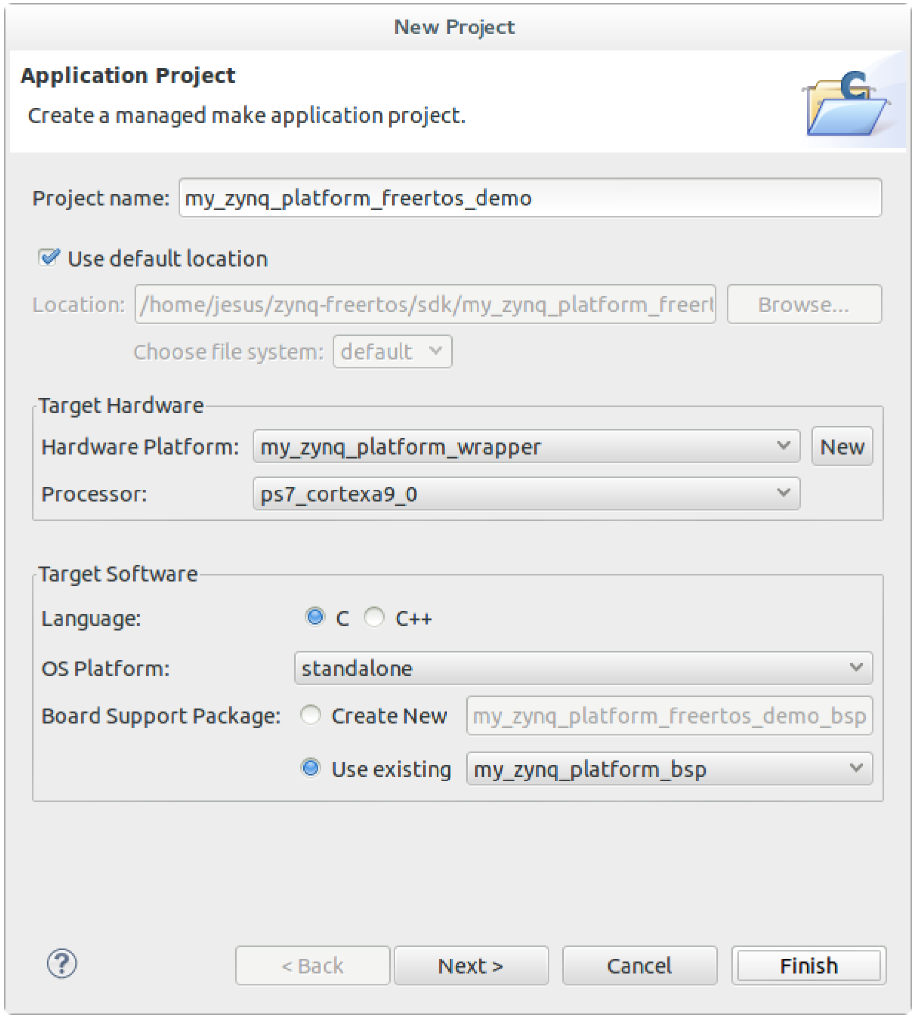
- Creación de un BSP para nuestra plataforma
- *Port* de *FreeRTOS* al *Zynq Processing System* de la *Zybo*
- Uso del *port* de *FreeRTOS*
- Generación del *First Stage Boot Loader*
- Preparación de la imagen de arranque

Creamos un proyecto para nuestra aplicación



CDT Build Console [my_zynq_platform_freertos_demo]

Creamos un proyecto para nuestra aplicación



New Project

Application Project

Create a managed make application project.

Project name: my_zynq_platform_freertos_demo

☑ Use default location

Location: /home/jesus/zynq-freertos/sdk/my_zynq_platform_freert Browse...

Choose file system: default

Target Hardware

Hardware Platform: my_zynq_platform_wrapper New

Processor: ps7_cortexa9_0

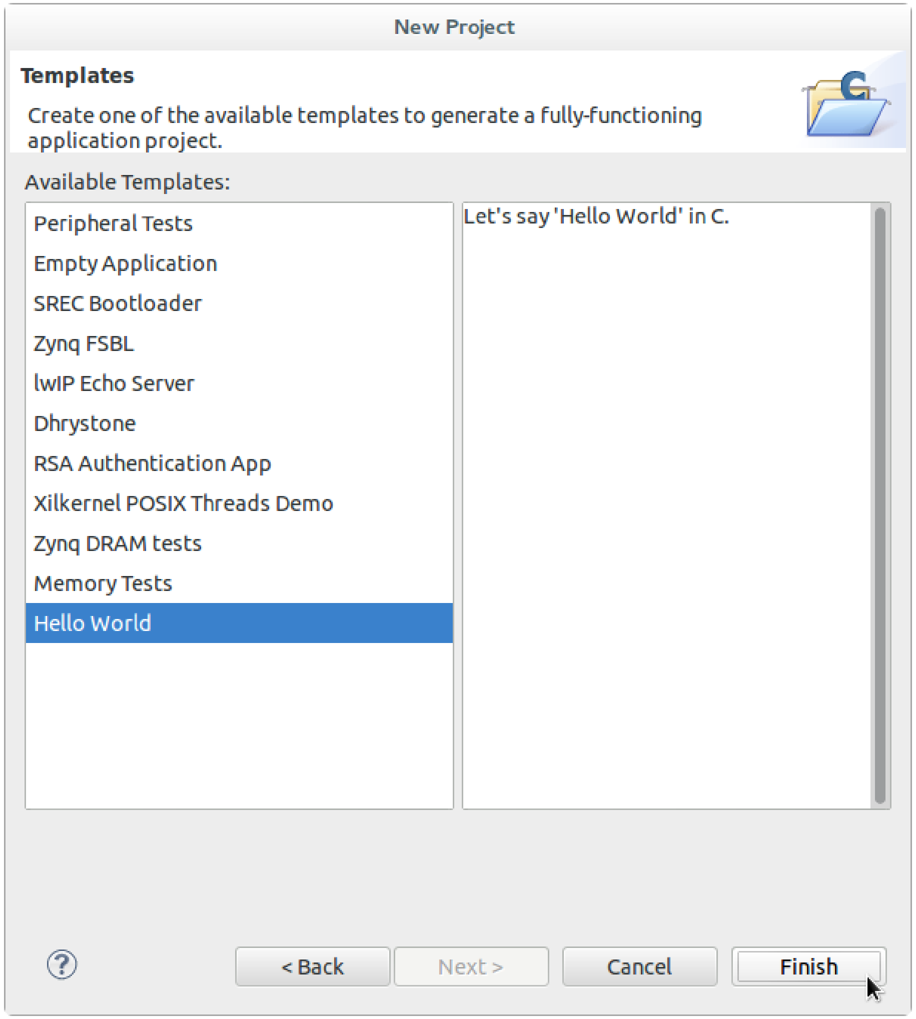
Target Software

Language: ● C ○ C++

OS Platform: standalone

Board Support Package: ○ Create New my_zynq_platform_freertos_demo_bsp

● Use existing my_zynq_platform_bsp



New Project

Templates

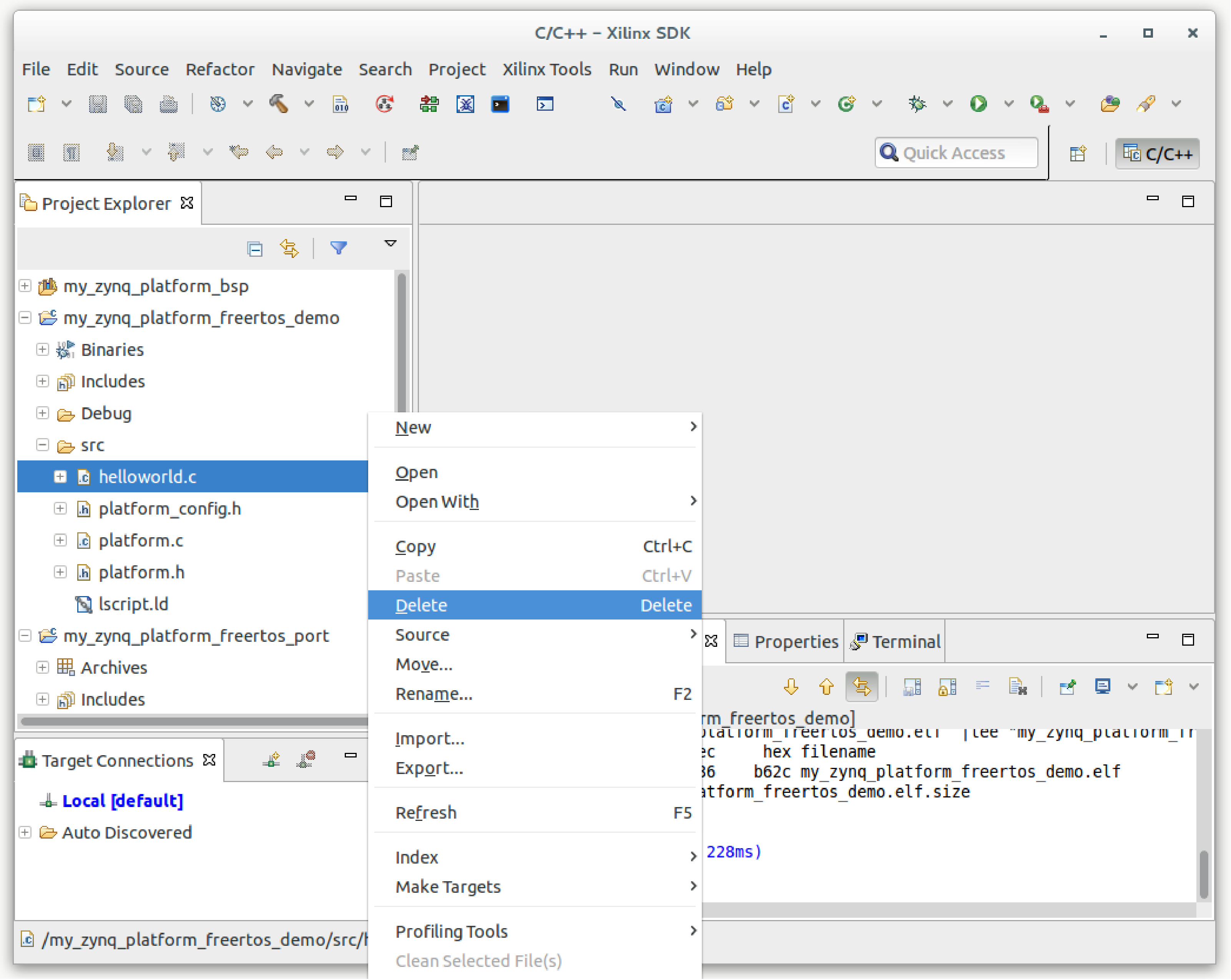
Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Peripheral Tests
- Empty Application
- SREC Bootloader
- Zynq FSBL
- lwIP Echo Server
- Dhrystone
- RSA Authentication App
- Xilkernel POSIX Threads Demo
- Zynq DRAM tests
- Memory Tests
- Hello World

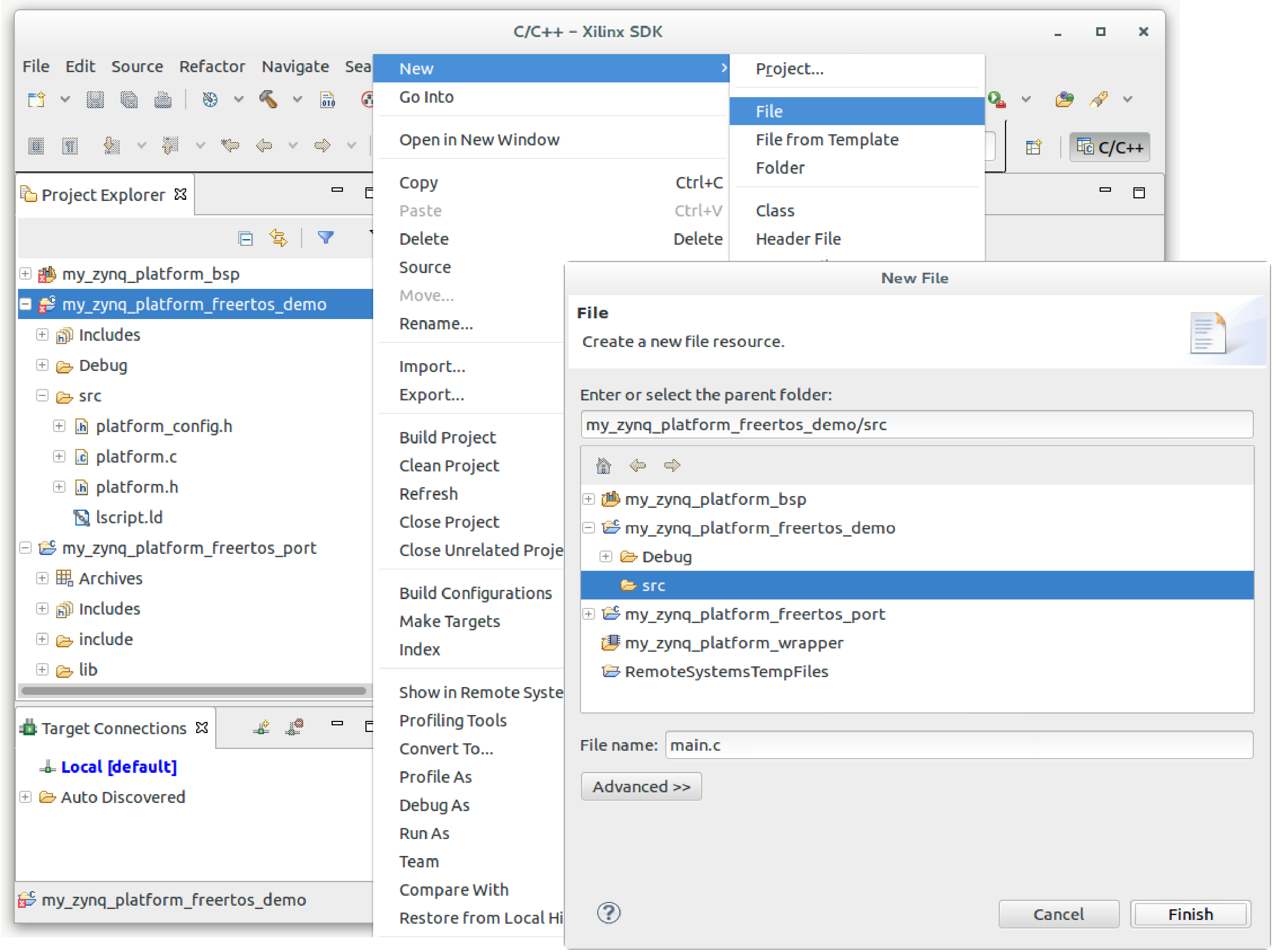
Let's say 'Hello World' in C.

Borramos el código C de la aplicación generada



```
rm_freertos_demo]
platform_freertos_demo.elf  |tee "my_zynq_platform_fr
ec      hex filename
36     b62c my_zynq_platform_freertos_demo.elf
atform_freertos_demo.elf.size

 228ms)
```

Creamos nuestro fichero `main.c`



New File

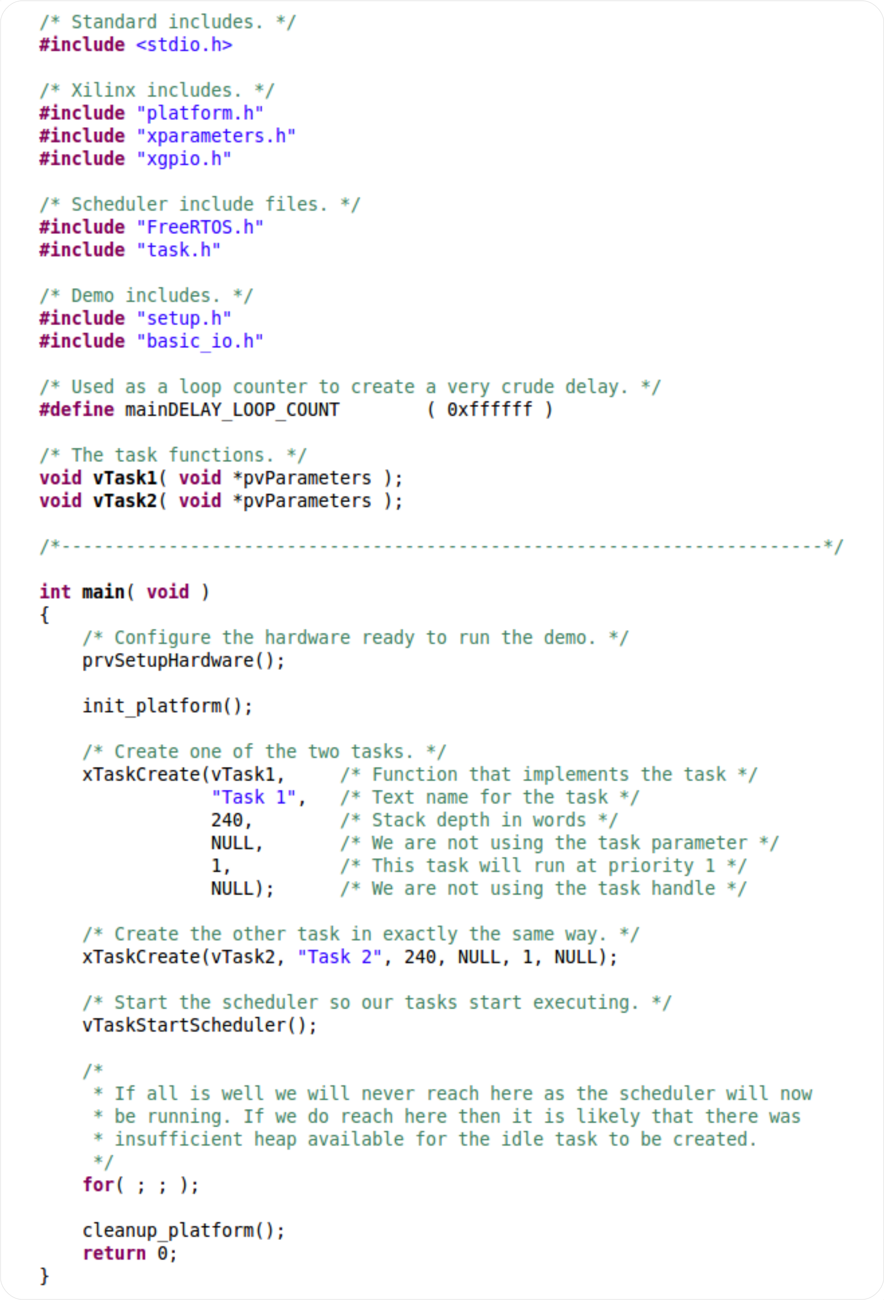
File

Create a new file resource.

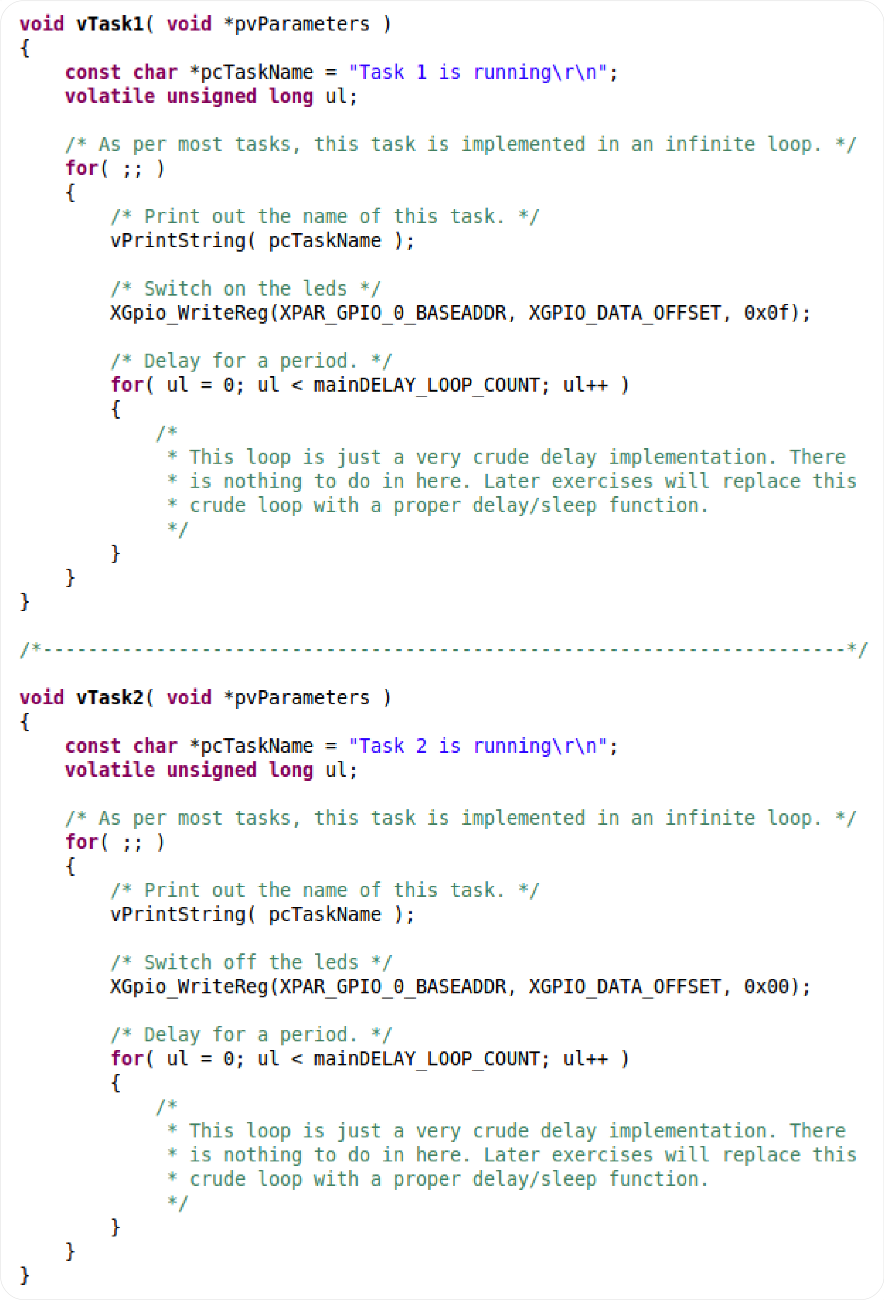
Enter or select the parent folder: my_zynq_platform_freertos_demo/src

File name: main.c

Código de `main.c`

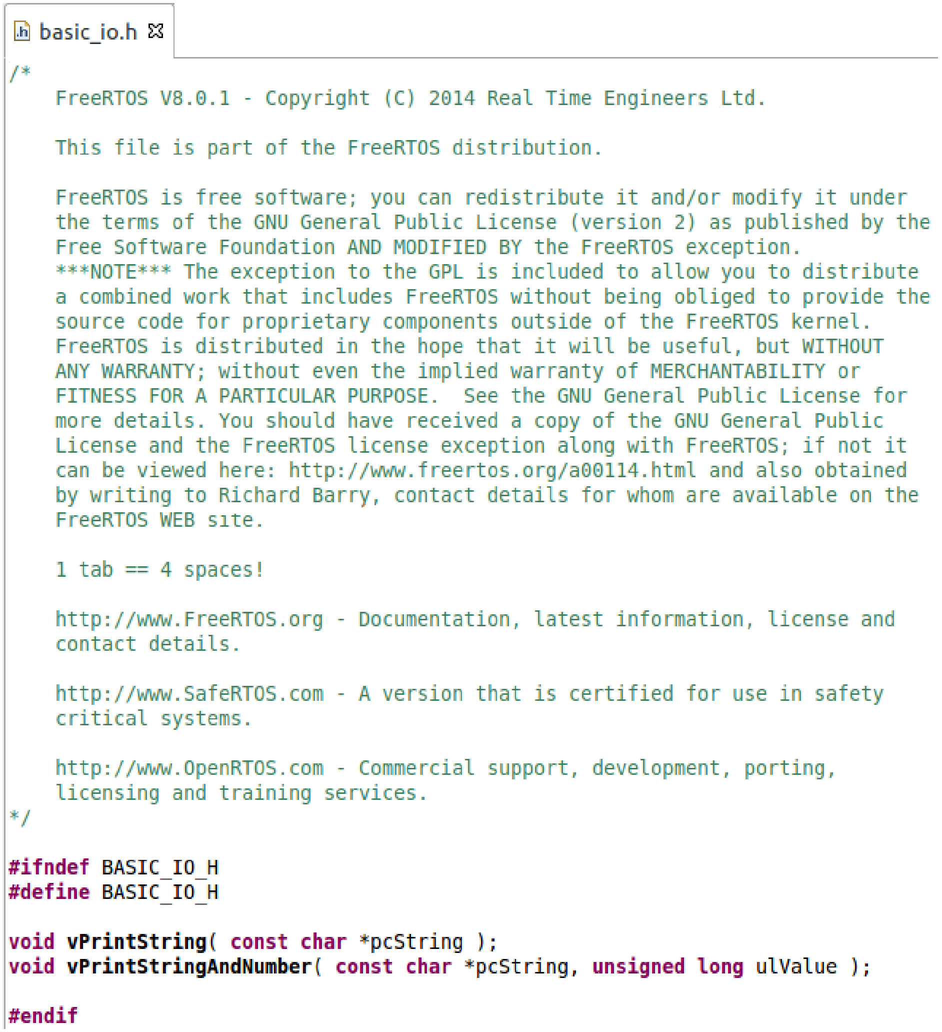


```c
/* Standard includes. */
#include <stdio.h>

/* Xilinx includes. */
#include "platform.h"
#include "xparameters.h"
#include "xgpio.h"

/* Scheduler include files. */
#include "FreeRTOS.h"
#include "task.h"

/* Demo includes. */
#include "setup.h"
#include "basic_io.h"

/* Used as a loop counter to create a very crude delay. */
#define mainDELAY_LOOP_COUNT        ( 0xffffff )

/* The task functions. */
void vTask1( void *pvParameters );
void vTask2( void *pvParameters );

/*-----------------------------------------------------------------------*/

int main( void )
{
    /* Configure the hardware ready to run the demo. */
    prvSetupHardware();

    init_platform();

    /* Create one of the two tasks. */
    xTaskCreate(vTask1,     /* Function that implements the task */
                "Task 1",   /* Text name for the task */
                240,        /* Stack depth in words */
                NULL,       /* We are not using the task parameter */
                1,          /* This task will run at priority 1 */
                NULL);      /* We are not using the task handle */

    /* Create the other task in exactly the same way. */
    xTaskCreate(vTask2, "Task 2", 240, NULL, 1, NULL);

    /* Start the scheduler so our tasks start executing. */
    vTaskStartScheduler();

    /*
     * If all is well we will never reach here as the scheduler will now
     * be running. If we do reach here then it is likely that there was
     * insufficient heap available for the idle task to be created.
     */
    for( ; ; );

    cleanup_platform();
    return 0;
}
```

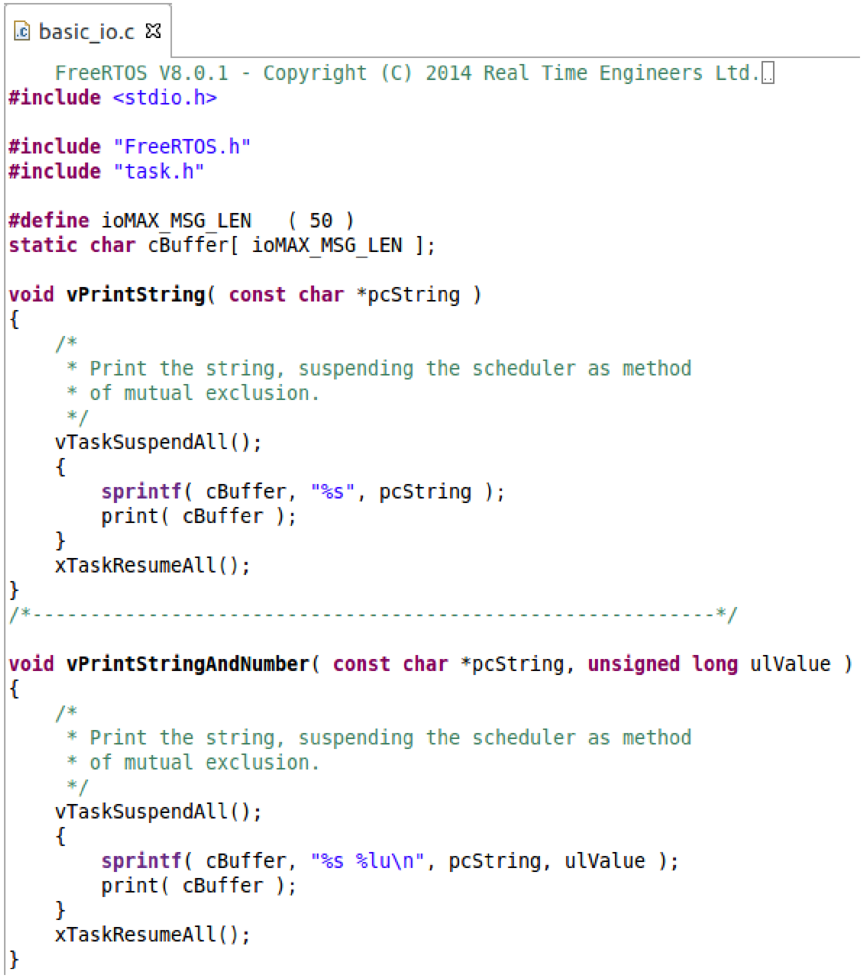


```c
void vTask1( void *pvParameters )
{
    const char *pcTaskName = "Task 1 is running\r\n";
    volatile unsigned long ul;

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* Switch on the leds */
        XGpio_WriteReg(XPAR_GPIO_0_BASEADDR, XGPIO_DATA_OFFSET, 0x0f);

        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
            /*
             * This loop is just a very crude delay implementation. There
             * is nothing to do in here. Later exercises will replace this
             * crude loop with a proper delay/sleep function.
             */
        }
    }
}

/*-----------------------------------------------------------------------*/

void vTask2( void *pvParameters )
{
    const char *pcTaskName = "Task 2 is running\r\n";
    volatile unsigned long ul;

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* Switch off the leds */
        XGpio_WriteReg(XPAR_GPIO_0_BASEADDR, XGPIO_DATA_OFFSET, 0x00);

        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
            /*
             * This loop is just a very crude delay implementation. There
             * is nothing to do in here. Later exercises will replace this
             * crude loop with a proper delay/sleep function.
             */
        }
    }
}
```

Acceso a la consola en exclusión mutua

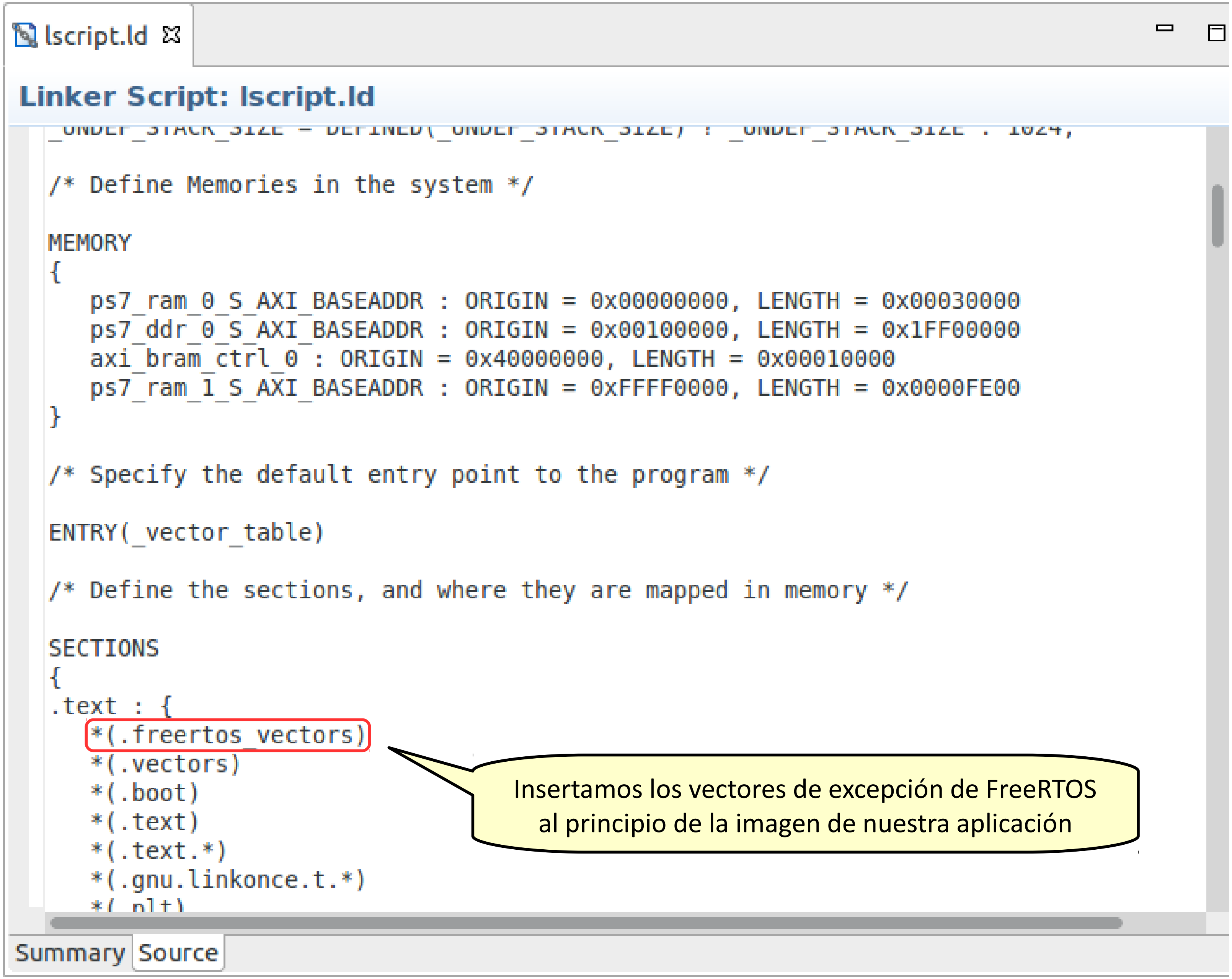


```
/*
    FreeRTOS V8.0.1 - Copyright (C) 2014 Real Time Engineers Ltd.

    This file is part of the FreeRTOS distribution.

    FreeRTOS is free software; you can redistribute it and/or modify it under
    the terms of the GNU General Public License (version 2) as published by the
    Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
    ***NOTE*** The exception to the GPL is included to allow you to distribute
    a combined work that includes FreeRTOS without being obliged to provide the
    source code for proprietary components outside of the FreeRTOS kernel.
    FreeRTOS is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
    more details. You should have received a copy of the GNU General Public
    License and the FreeRTOS license exception along with FreeRTOS; if not it
    can be viewed here: http://www.freertos.org/a00114.html and also obtained
    by writing to Richard Barry, contact details for whom are available on the
    FreeRTOS WEB site.

    1 tab == 4 spaces!

    http://www.FreeRTOS.org - Documentation, latest information, license and
    contact details.

    http://www.SafeRTOS.com - A version that is certified for use in safety
    critical systems.

    http://www.OpenRTOS.com - Commercial support, development, porting,
    licensing and training services.
*/

#ifndef BASIC_IO_H
#define BASIC_IO_H

void vPrintString( const char *pcString );
void vPrintStringAndNumber( const char *pcString, unsigned long ulValue );

#endif
```



```
    FreeRTOS V8.0.1 - Copyright (C) 2014 Real Time Engineers Ltd.
#include <stdio.h>

#include "FreeRTOS.h"
#include "task.h"

#define ioMAX_MSG_LEN   ( 50 )
static char cBuffer[ ioMAX_MSG_LEN ];

void vPrintString( const char *pcString )
{
    /*
     * Print the string, suspending the scheduler as method
     * of mutual exclusion.
     */
    vTaskSuspendAll();
    {
        sprintf( cBuffer, "%s", pcString );
        print( cBuffer );
    }
    xTaskResumeAll();
}
/*-----------------------------------------------------------*/

void vPrintStringAndNumber( const char *pcString, unsigned long ulValue )
{
    /*
     * Print the string, suspending the scheduler as method
     * of mutual exclusion.
     */
    vTaskSuspendAll();
    {
        sprintf( cBuffer, "%s %lu\n", pcString, ulValue );
        print( cBuffer );
    }
    xTaskResumeAll();
}
```

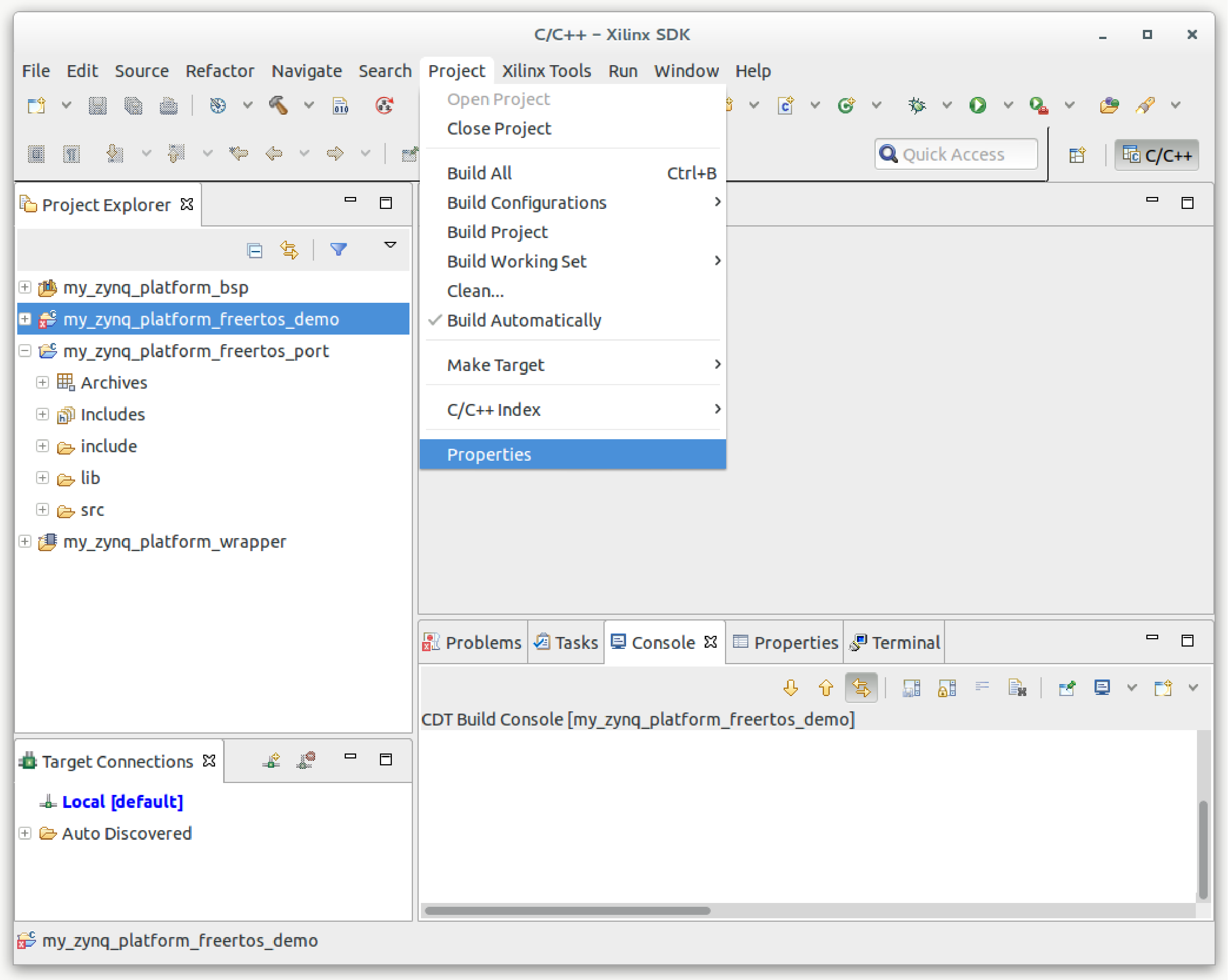
Modificamos el *linker script* para insertar los vectores de *FreeRTOS*



```
_UNDEF_STACK_SIZE = DEFINED(_UNDEF_STACK_SIZE) ? _UNDEF_STACK_SIZE : 1024;

/* Define Memories in the system */

MEMORY
{
   ps7_ram_0_S_AXI_BASEADDR : ORIGIN = 0x00000000, LENGTH = 0x00030000
   ps7_ddr_0_S_AXI_BASEADDR : ORIGIN = 0x00100000, LENGTH = 0x1FF00000
   axi_bram_ctrl_0 : ORIGIN = 0x40000000, LENGTH = 0x00010000
   ps7_ram_1_S_AXI_BASEADDR : ORIGIN = 0xFFFF0000, LENGTH = 0x0000FE00
}

/* Specify the default entry point to the program */

ENTRY(_vector_table)

/* Define the sections, and where they are mapped in memory */

SECTIONS
{
.text : {
   *(.freertos_vectors)
   *(.vectors)
   *(.boot)
   *(.text)
   *(.text.*)
   *(.gnu.linkonce.t.*)
   *(.plt)
```

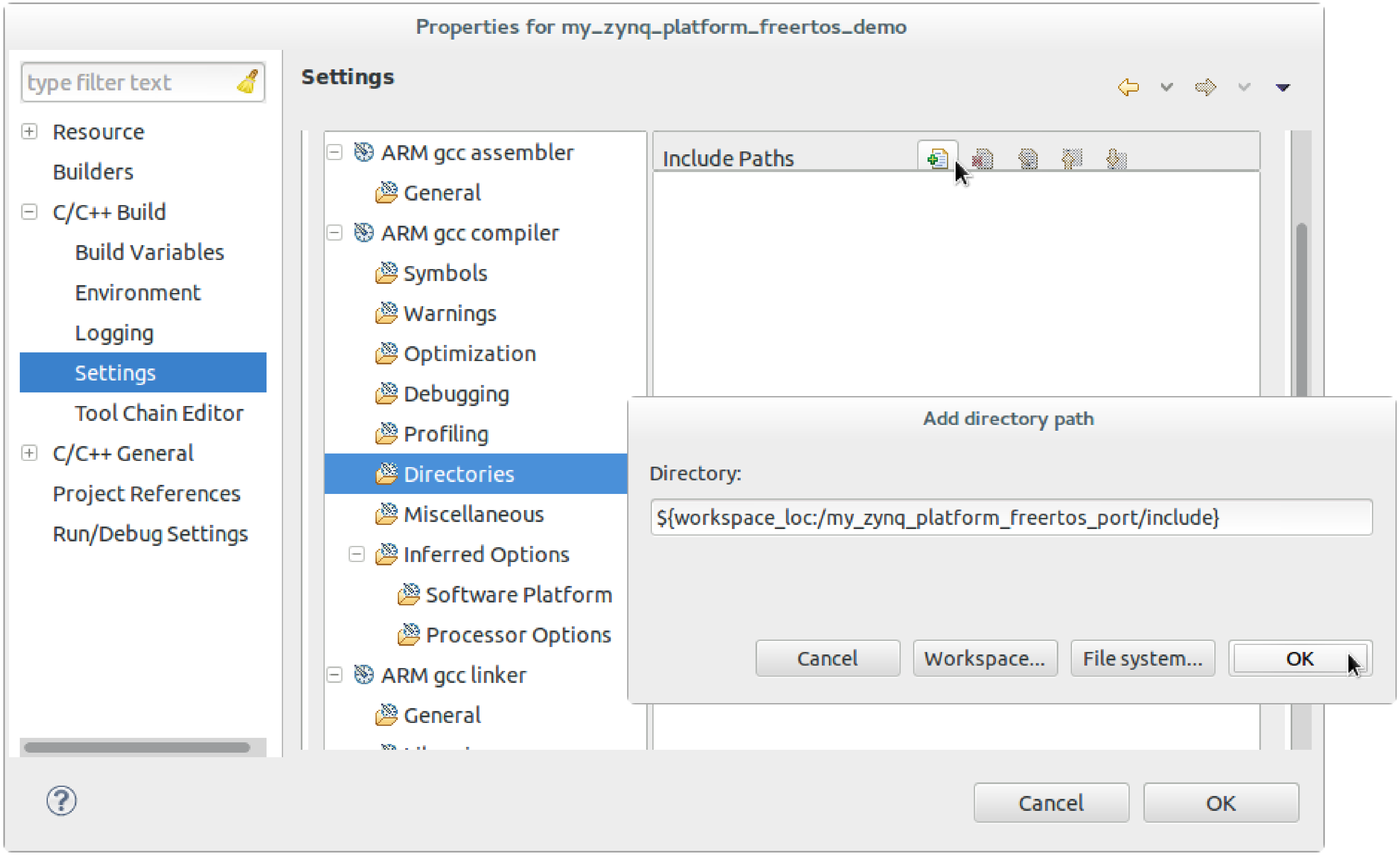
Insertamos los vectores de excepción de FreeRTOS al principio de la imagen de nuestra aplicación

Configuramos el proyecto



CDT Build Console [my_zynq_platform_freertos_demo]

Añadimos el directorio con las cabeceras de las funciones de *FreeRTOS*



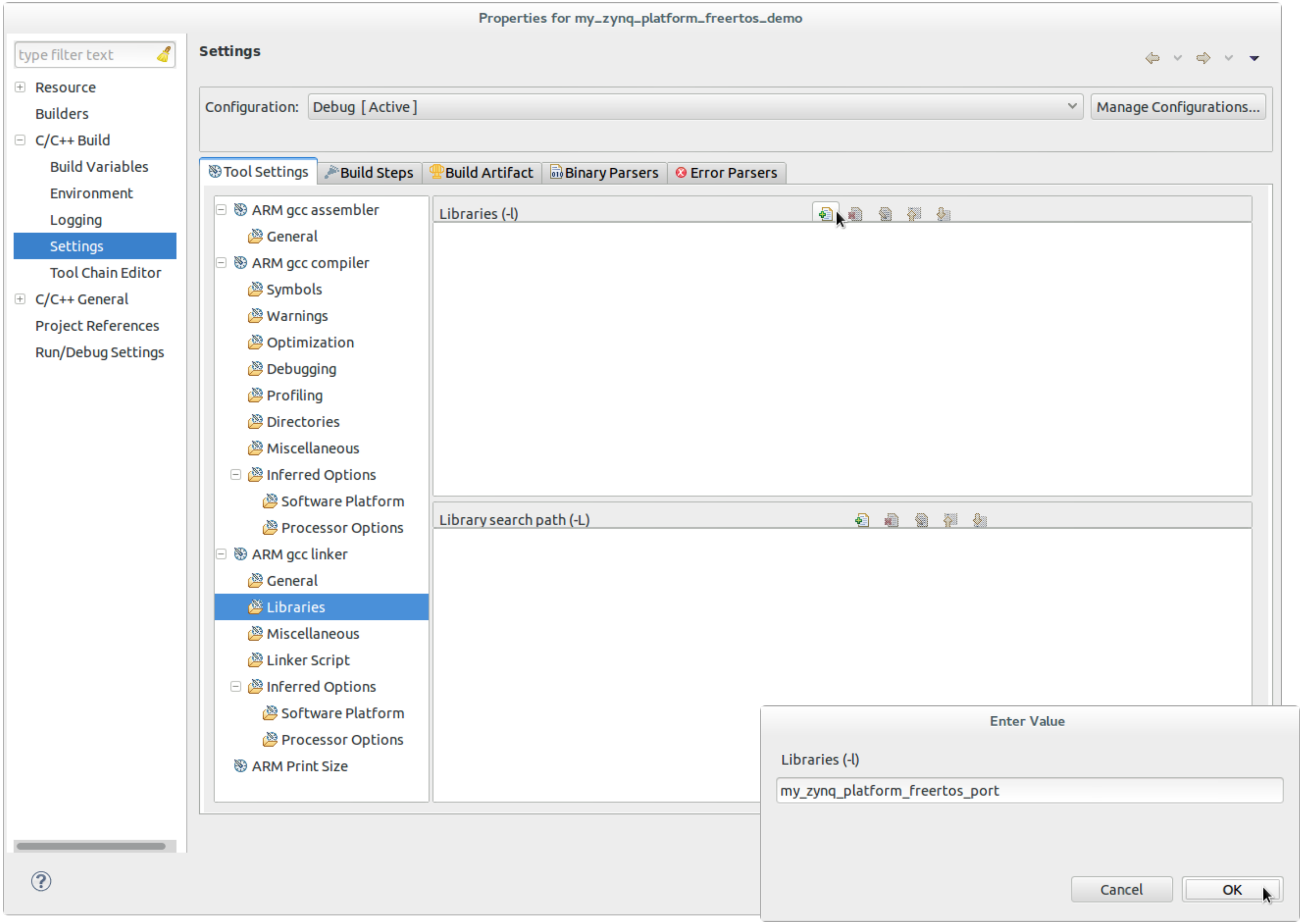
Properties for my_zynq_platform_freertos_demo

Settings

Add directory path

Directory: ${workspace_loc:/my_zynq_platform_freertos_port/include}

Enlazamos con la biblioteca del *port* de *FreeRTOS*



Properties for my_zynq_platform_freertos_demo

Settings

Configuration: Debug [Active]

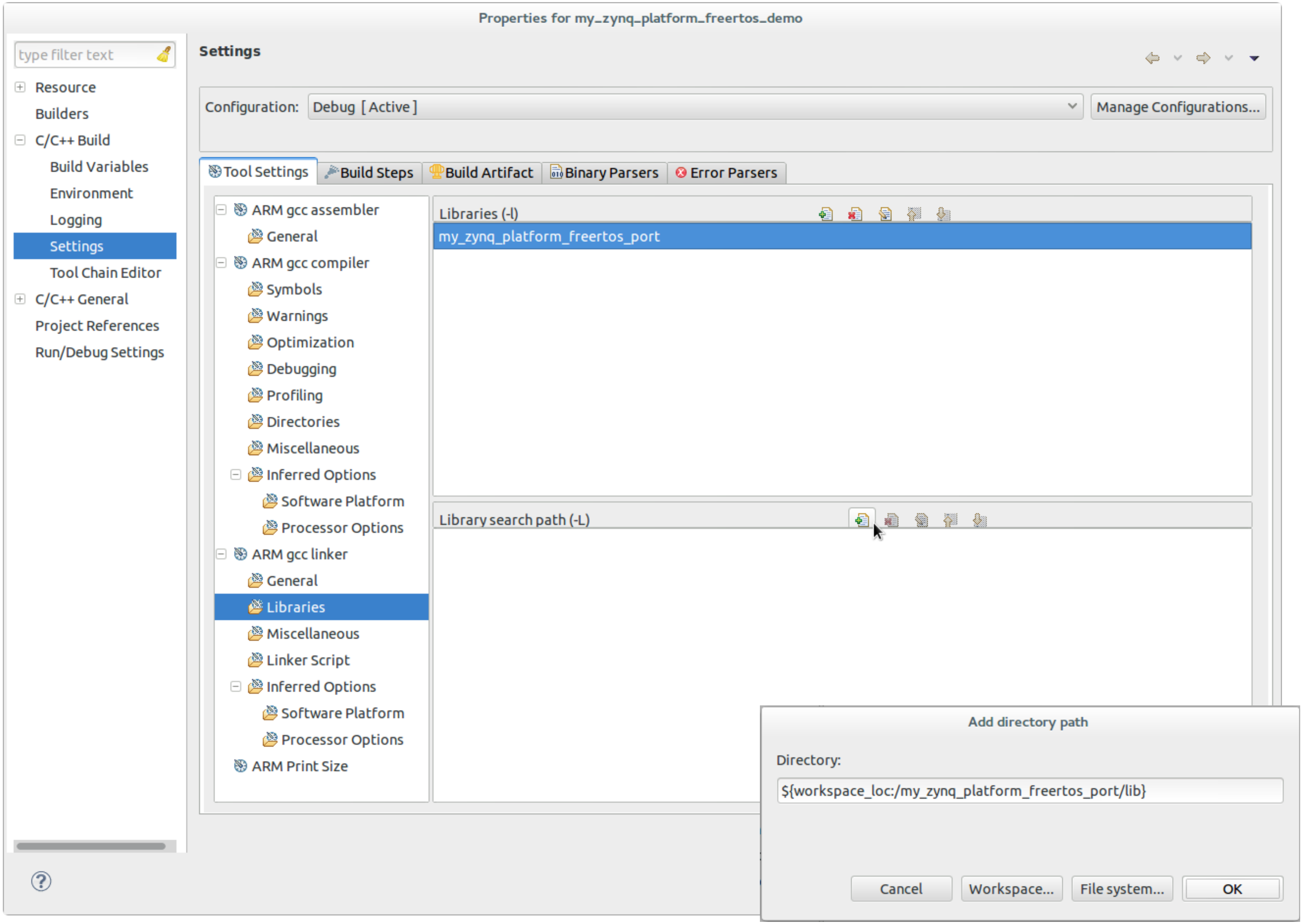
Libraries (-l)

Enter Value

Libraries (-l)

my_zynq_platform_freertos_port

Enlazamos con la biblioteca del *port* de *FreeRTOS*



Properties for my_zynq_platform_freertos_demo

Settings

Configuration: Debug [Active]

ARM gcc linker > Libraries

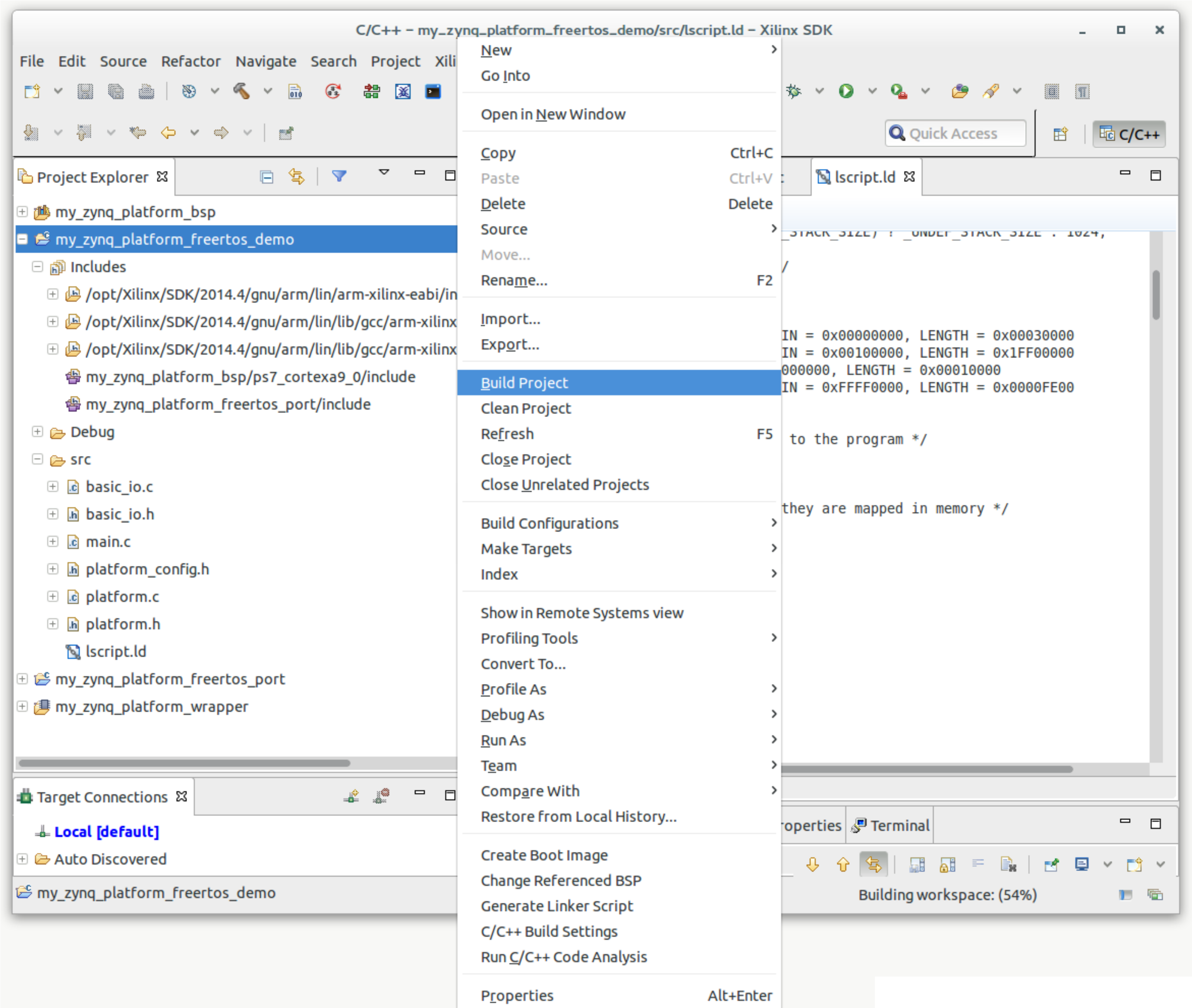
Libraries (-l): my_zynq_platform_freertos_port

Library search path (-L)

Add directory path

Directory: ${workspace_loc:/my_zynq_platform_freertos_port/lib}

Construimos la aplicación



```
_STACK_SIZE) ? _UNDEF_STACK_SIZE : 1024;
/
IN = 0x00000000, LENGTH = 0x00030000
IN = 0x00100000, LENGTH = 0x1FF00000
000000, LENGTH = 0x00010000
IN = 0xFFFF0000, LENGTH = 0x0000FE00
 to the program */
they are mapped in memory */
```

Contenidos

Tema 2: *FreeRTOS*

Creación de un BSP para nuestra plataforma

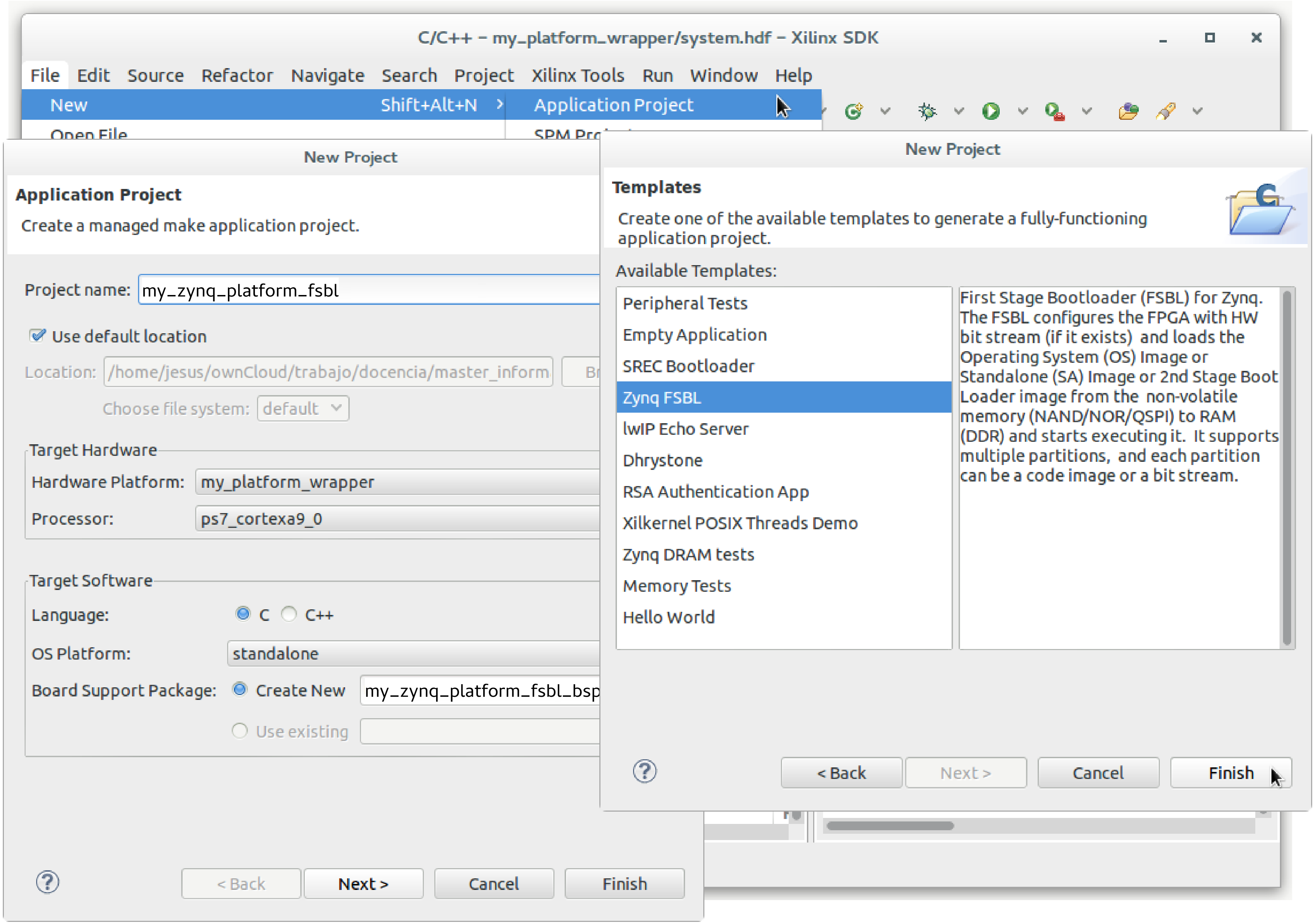
Port de *FreeRTOS* al *Zynq Processing System* de la *Zybo*

Uso del *port* de *FreeRTOS*

Generación del *First Stage Boot Loader*

Preparación de la imagen de arranque

Creación del *First Stage Boot Loader* mediante el *SDK*



C/C++ – my_platform_wrapper/system.hdf – Xilinx SDK
File Edit Source Refactor Navigate Search Project Xilinx Tools Run Window Help
New Shift+Alt+N
Application Project
Open File
SPM Pro

New Project

Application Project

Create a managed make application project.

Project name: my_zynq_platform_fsbl

☑ Use default location

Location: /home/jesus/ownCloud/trabajo/docencia/master_inform

Choose file system: default

Target Hardware

Hardware Platform: my_platform_wrapper

Processor: ps7_cortexa9_0

Target Software

Language: ● C ○ C++

OS Platform: standalone

Board Support Package: ● Create New my_zynq_platform_fsbl_bsp

○ Use existing

New Project

Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Peripheral Tests
- Empty Application
- SREC Bootloader
- Zynq FSBL
- lwIP Echo Server
- Dhrystone
- RSA Authentication App
- Xilkernel POSIX Threads Demo
- Zynq DRAM tests
- Memory Tests
- Hello World

First Stage Bootloader (FSBL) for Zynq. The FSBL configures the FPGA with HW bit stream (if it exists) and loads the Operating System (OS) Image or Standalone (SA) Image or 2nd Stage Boot Loader image from the non-volatile memory (NAND/NOR/QSPI) to RAM (DDR) and starts executing it. It supports multiple partitions, and each partition can be a code image or a bit stream.

Creación del *First Stage Boot Loader* mediante *scripts*

Variables de entorno

```
export PLATFORM="my_zynq_platform"
export PLATFORM_DIR="${PRJ_ROOT}/${PLATFORM}"
export SDK_DIR="${PRJ_ROOT}/sdk"
export PLATFORM_WRAPPER="${PLATFORM}_wrapper"
export FSBL="${PLATFORM}_fsbl"
export FSBL_DIR="${SDK_DIR}/${FSBL}"
```

Creación del FSBL

```
mkdir -p ${SDK_DIR}/${PLATFORM_WRAPPER}

cp ${PLATFORM_DIR}/${PLATFORM}.runs/impl_1/${PLATFORM_WRAPPER}.sysdef \
   ${SDK_DIR}/${PLATFORM_WRAPPER}/system.hdf

hsi -mode batch -source fsbl.tcl
mkdir -p ${PRJ_ROOT}/images
cp ${FSBL_DIR}/executable.elf ${PRJ_ROOT}/images/${FSBL}.elf
```

Fichero `fsbl.tcl`

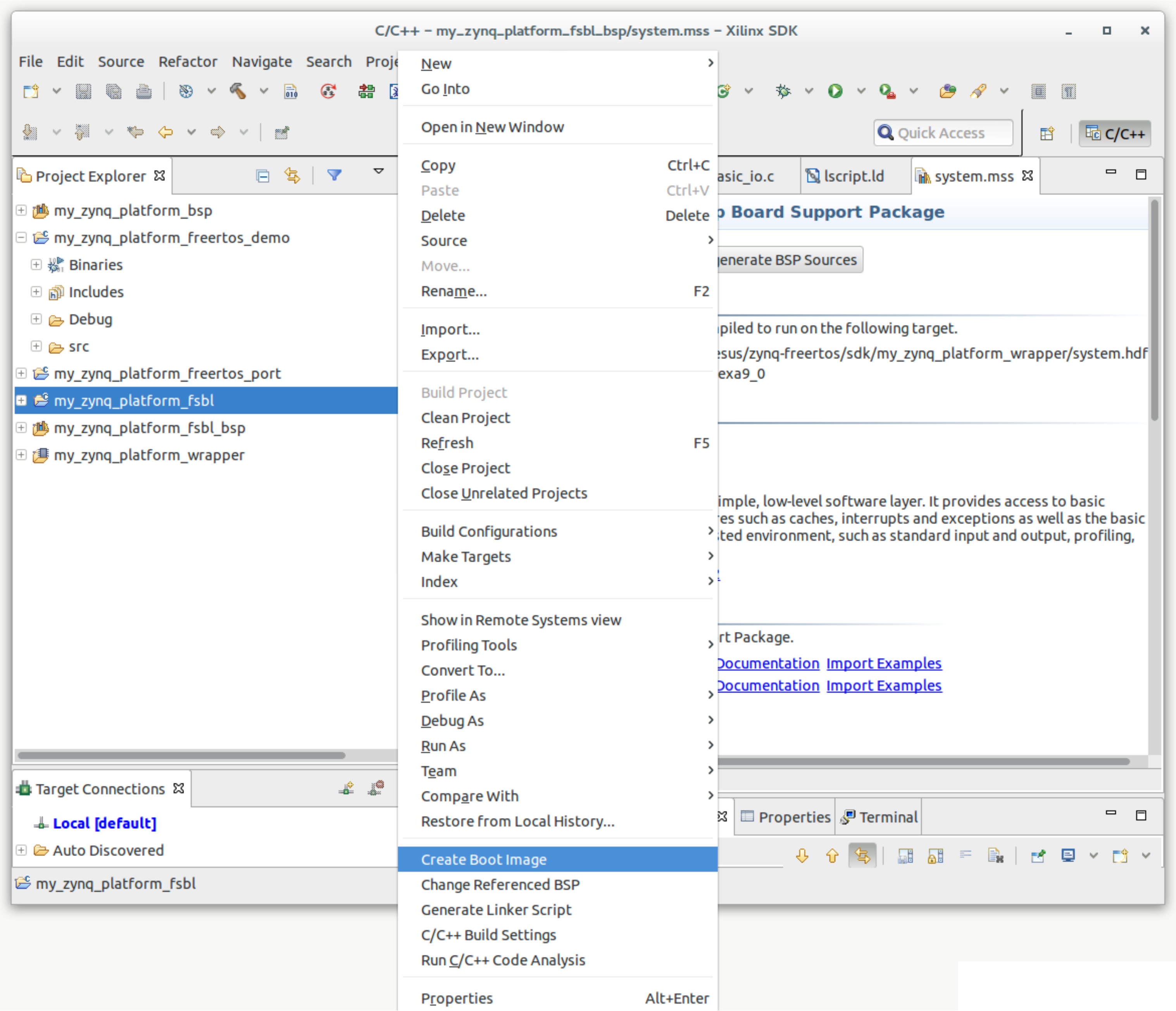
```
open_hw_design $env(SDK_DIR)/$env(PLATFORM_WRAPPER)/system.hdf

generate_app -hw $env(PLATFORM)_imp -os standalone -proc ps7_cortexa9_0 \
             -app zynq_fsbl -compile -sw $env(FSBL) -dir $env(FSBL_DIR)
```

Contenidos

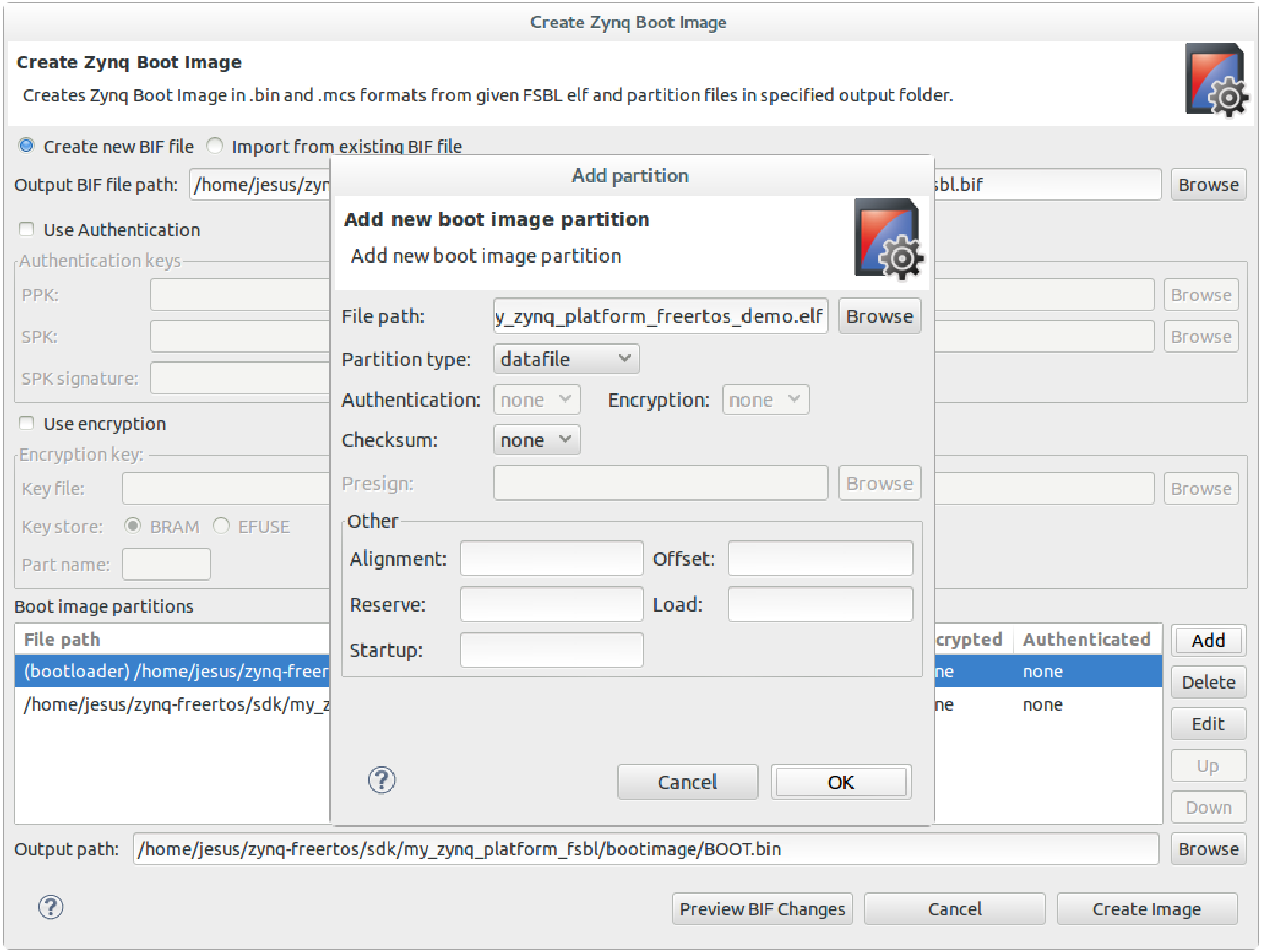
Tema 2: *FreeRTOS*

- Creación de un BSP para nuestra plataforma
- *Port* de *FreeRTOS* al *Zynq Processing System* de la *Zybo*
- Uso del *port* de *FreeRTOS*
- Generación del *First Stage Boot Loader*
- Preparación de la imagen de arranque

Creamos la imagen de arranque



Añadimos nuestra aplicación a la imagen



Create Zynq Boot Image

Create Zynq Boot Image

Creates Zynq Boot Image in .bin and .mcs formats from given FSBL elf and partition files in specified output folder.

● Create new BIF file ○ Import from existing BIF file

Output BIF file path: /home/jesus/zyn… …sbl.bif

□ Use Authentication

Authentication keys

PPK:

SPK:

SPK signature:

□ Use encryption

Encryption key:

Key file:

Key store: ● BRAM ○ EFUSE

Part name:

Boot image partitions

File path	… crypted	Authenticated
(bootloader) /home/jesus/zynq-freer…	…ne	none
/home/jesus/zynq-freertos/sdk/my_z…	…ne	none

Output path: /home/jesus/zynq-freertos/sdk/my_zynq_platform_fsbl/bootimage/BOOT.bin

Add partition

Add new boot image partition

Add new boot image partition

File path: y_zynq_platform_freertos_demo.elf

Partition type: datafile

Authentication: none

Encryption: none

Checksum: none

Presign:

Other

Alignment:

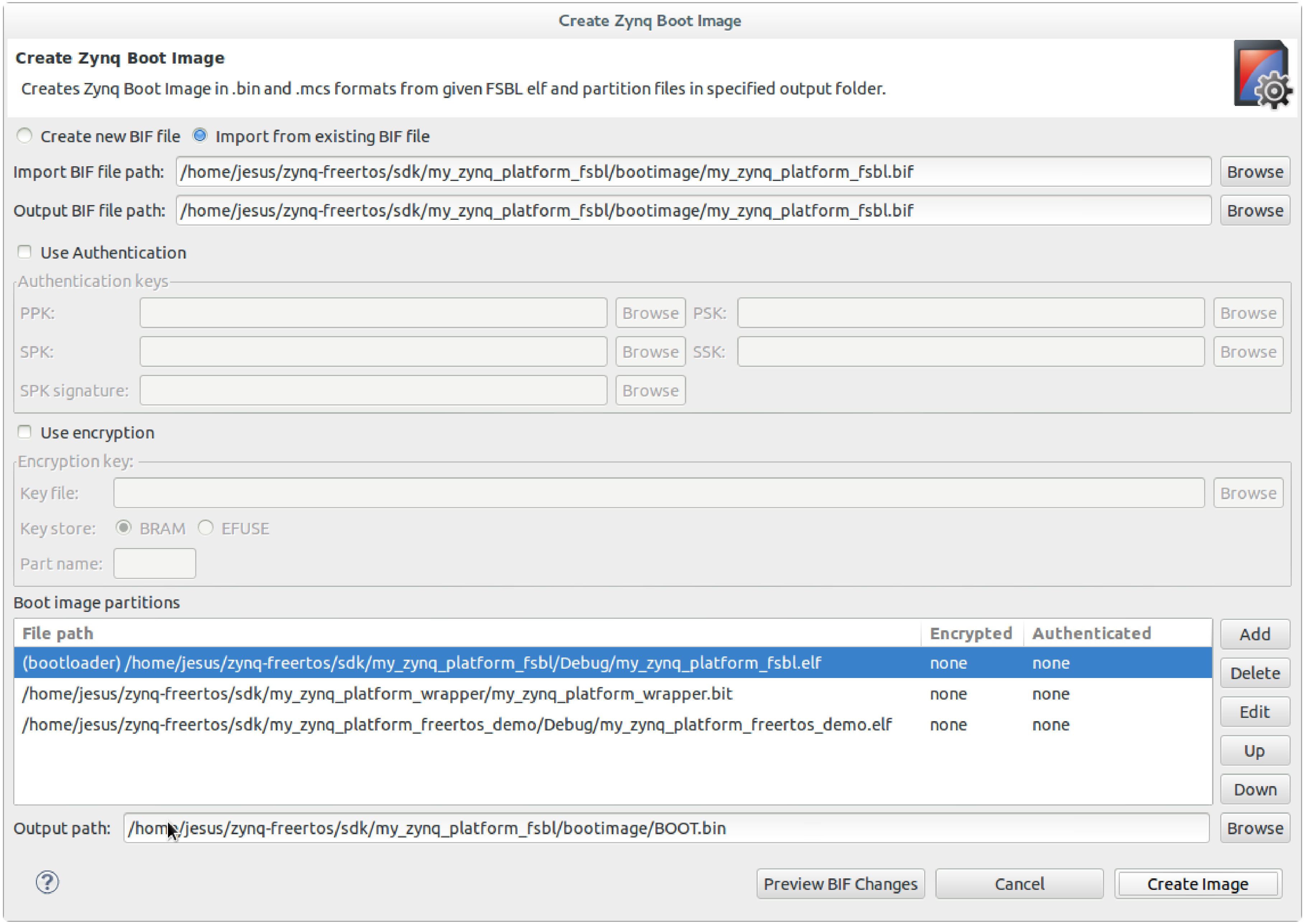
Offset:

Reserve:

Load:

Startup:

Generamos la imagen de arranque



Create Zynq Boot Image

Create Zynq Boot Image

Creates Zynq Boot Image in .bin and .mcs formats from given FSBL elf and partition files in specified output folder.

○ Create new BIF file ● Import from existing BIF file

Import BIF file path: /home/jesus/zynq-freertos/sdk/my_zynq_platform_fsbl/bootimage/my_zynq_platform_fsbl.bif

Output BIF file path: /home/jesus/zynq-freertos/sdk/my_zynq_platform_fsbl/bootimage/my_zynq_platform_fsbl.bif

□ Use Authentication

Authentication keys

PPK: SPK: SPK signature: PSK: SSK:

□ Use encryption

Encryption key:

Key file:

Key store: ● BRAM ○ EFUSE

Part name:

Boot image partitions

File path	Encrypted	Authenticated
(bootloader) /home/jesus/zynq-freertos/sdk/my_zynq_platform_fsbl/Debug/my_zynq_platform_fsbl.elf	none	none
/home/jesus/zynq-freertos/sdk/my_zynq_platform_wrapper/my_zynq_platform_wrapper.bit	none	none
/home/jesus/zynq-freertos/sdk/my_zynq_platform_freertos_demo/Debug/my_zynq_platform_freertos_demo.elf	none	none

Output path: /home/jesus/zynq-freertos/sdk/my_zynq_platform_fsbl/bootimage/BOOT.bin

Generamos la imagen de arranque de la placa mediante *scripts*

Variables de entorno

```
export PLATFORM="my_zynq_platform"
export SDK_DIR="${PRJ_ROOT}/sdk"
export PLATFORM_WRAPPER="${PLATFORM}_wrapper"
export PLATFORM_WRAPPER_DIR="${SDK_DIR}/${PLATFORM_WRAPPER}"
export FREERTOS_APP=${PLATFORM}_freertos_demo
export FREERTOS_APP_DIR=${SDK_DIR}/${FREERTOS_APP}
```

Copiamos el *bitfile* de la plataforma al directorio de las imágenes

```
cp ${PLATFORM_WRAPPER_DIR}/${PLATFORM_WRAPPER}.bit ${PRJ_ROOT}/images
```

Copiamos la demo al directorio de las imágenes

```
cp ${FREERTOS_APP_DIR}/Debug/${FREERTOS_APP}.elf ${PRJ_ROOT}/images
```

Generamos la imagen de arranque

```
cd ${PRJ_ROOT}/images
bootgen -image boot.bif -o boot.bin
```

Fichero `boot.bif` (define el formato de la imagen de arranque de la placa)

```
image :
{
        [bootloader]my_zynq_platform_fsbl.elf
        my_zynq_platform_wrapper.bit
        my_zynq_platform_freertos_demo.elf
}
```

Preparamos la placa

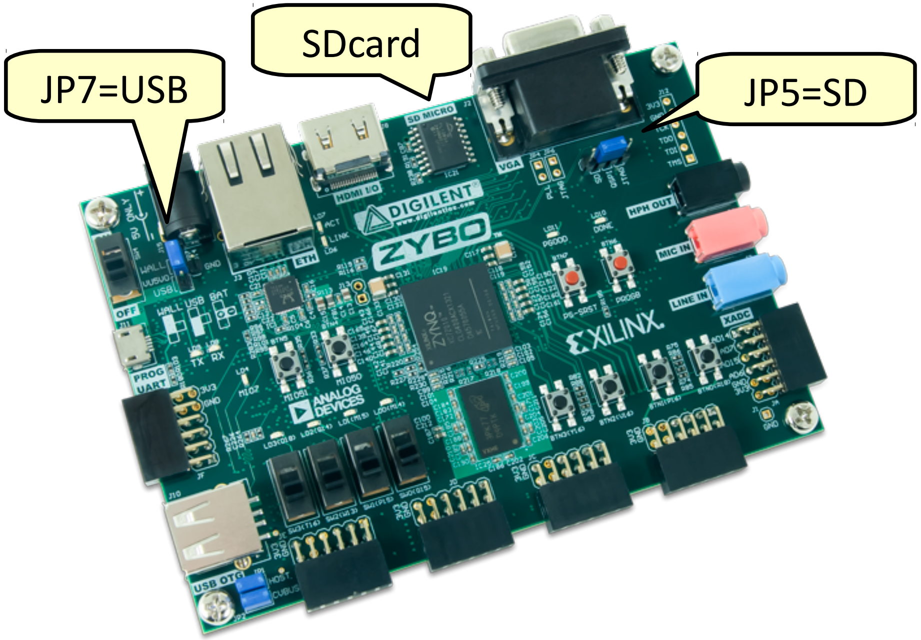
Variables de entorno

```
SDCARD_DIR="/media/jesus/9016-4EF8"
```

Configuramos la tarjeta microSD (copiamos a la primera partición)

```
cd ${PRJ_ROOT}/images
cp boot.bin ${SDCARD_DIR}
```

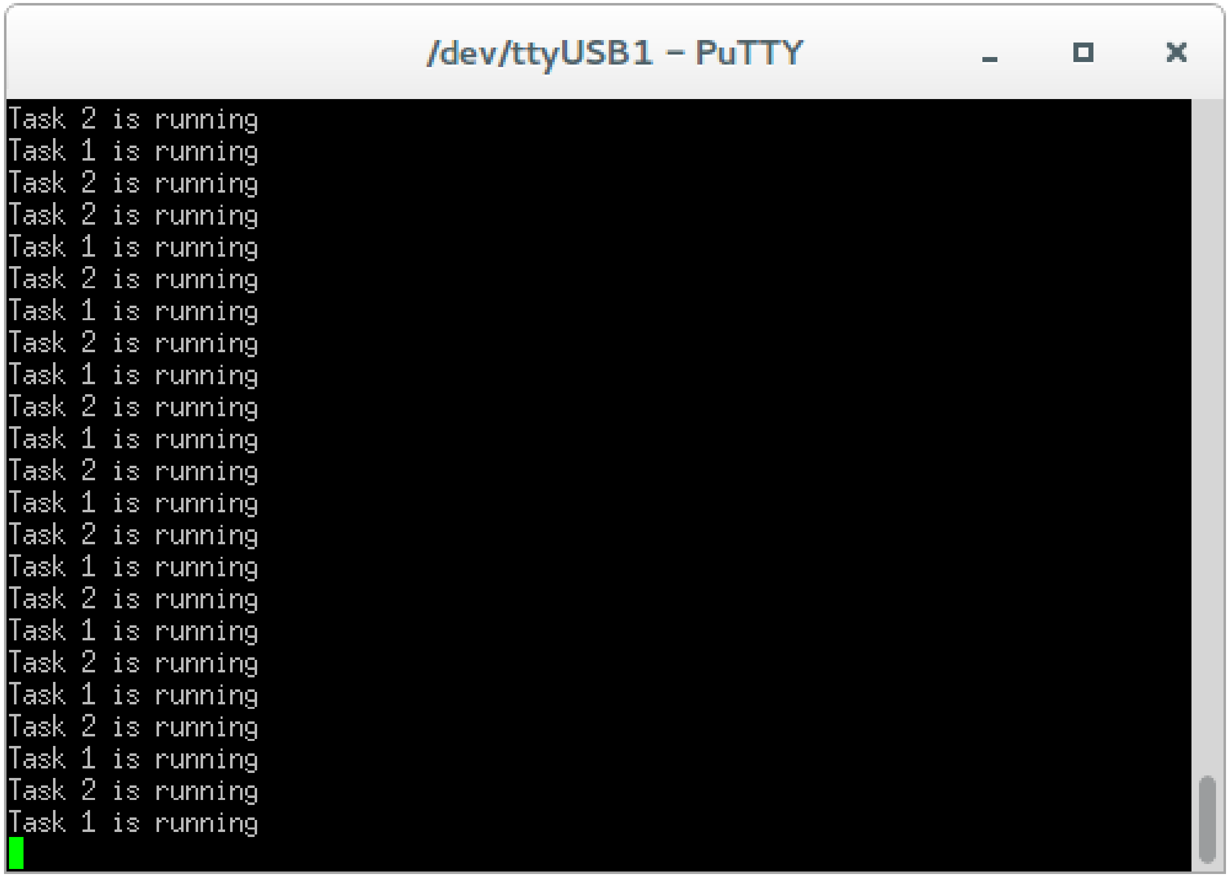
Preparamos la placa



Encendemos la placa y nos conectamos vía serie

Conexión a la placa

```
putty -serial -sercfg 115200 /dev/ttyUSB1
```



```
Task 2 is running
Task 1 is running
Task 2 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
Task 2 is running
Task 1 is running
```

Lecturas recomendadas

Documentación oficial de *FreeRTOS*:

Real Time Engineers Ltd. *FreeRTOS Quick Start Guide*. http://www.freertos.org/FreeRTOS-quick-start-guide.html

Richard Barry. *Using the FreeRTOS Real Time Kernel: A Practical Guide*. Real Time Engineers, 2010. Disponible en la biblioteca

Real Time Engineers Ltd. *Book Companion Source Code*. http://www.freertos.org/Documentation/code/

Real Time Engineers Ltd. *Real Time Application Design Tutorial. Using FreeRTOS in small embedded systems*. http://www.freertos.org/tutorial/

Cursos de FreeRTOS:

Amr Ali. *FreeRTOS Course – Introduction to FreeRTOS*. http://embedded-tips.blogspot.com.es/2010/06/free-freertos-course-introduction-to.html

Amr Ali. *FreeRTOS Course - Task Management*. http://es.slideshare.net/amraldo/free-freertos-coursetask-management

Amr Ali. *FreeRTOS Course - Queue Management*. http://es.slideshare.net/amraldo/m3-introduction-to-free-rtos-v605

Amr Ali. *FreeRTOS Course - Semaphore/Mutex Management*. http://es.slideshare.net/amraldo/freertos-course-semaphoremutex-management

Lecturas recomendadas

FreeRTOS porting:

Real Time Engineers Ltd. *Official FreeRTOS Ports*.
http://www.freertos.org/RTOS_ports.html

Real Time Engineers Ltd. *Modifying a FreeRTOS Demo*.
http://www.freertos.org/porting-a-freertos-demo-to-different-hardware.html

Real Time Engineers Ltd. *Creating a New FreeRTOS Port*.
http://www.freertos.org/FreeRTOS-porting-guide.html

Port de *FreeRTOS* para el *SoC Zynq*:

Real Time Engineers Ltd. *Xilinx Zynq-7000 (dual core ARM Cortex-A9) SoC Port*.
http://www.freertos.org/RTOS-Xilinx-Zynq.html

Circuit Sense. *FreeRTOS on Xilinx Zynq Zybo [Single Core]*, abril 2015.
http://rishifranklin.blogspot.com.es/2015/04/freertos-on-xilinx-zynq-zybo-single-core.html