

Universidad de Granada



Sistemas Críticos

Port de un RTOS a la Zybo

Marvin Matías Agüero Torales

maguero@correo.ugr.es

Curso 2016-2017

Sumario

Enunciado de la práctica.....	3
Desarrollo.....	3
Creación de un BSP para nuestra plataforma.....	3
Port de FreeRTOS al Zynq Processing System de la Zybo.....	4
Uso del port de FreeRTOS.....	7
Generación del First Stage Boot Loader y preparación de la imagen de arranque.....	10
Inconvenientes.....	14
Conclusiones.....	14
Anexos.....	14

Enunciado de la práctica

En esta práctica se realizará un port del sistema operativo en tiempo real FreeRTOS para la plataforma de desarrollo Zybo. El alumno deberá:

- Diseñar un hardware básico incluyendo algunos periféricos con Vivado.
- Generar el BSP para la plataforma a partir del hardware diseñado.
- Crear el port de FreeRTOS para la plataforma.
- Desarrollar una pequeña demo.
- Generar con SDK el sistema de arranque de la tarjeta.

Además de las actividades realizadas, incluir los principales problemas resueltos así como los elementos del diseño y formación recibida más interesantes.

Se deberá adjuntar un archivo comprimido con los archivos necesarios para cargar FreeRTOS en la Zybo.

Desarrollo

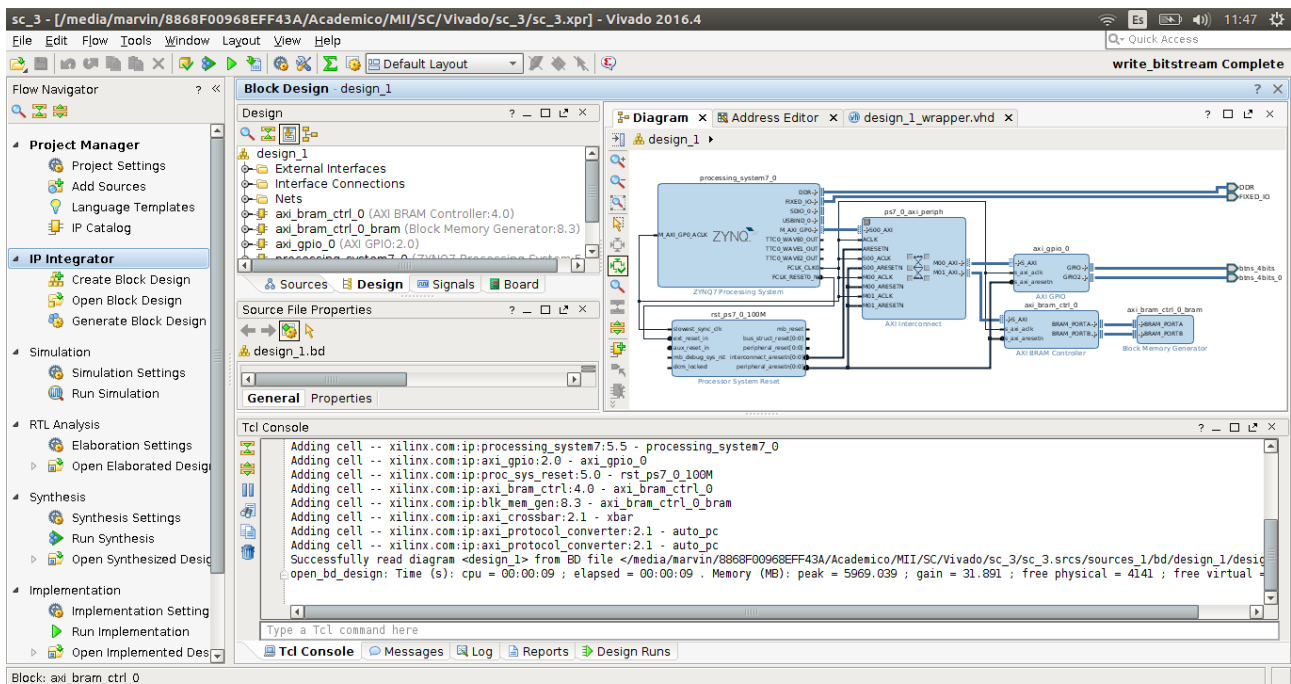
El desarrollo se hace sobre una portátil HP Envy con procesador i7 3º generación con 8 cores y 8 GB de RAM, con SO Ubuntu 16.04 LTS de 64 bits.

Se siguió el material de la asignatura para llevar acabo este trabajo (González Peñalver, 2015).

Creación de un BSP para nuestra plataforma

Usaremos el mismo diseño que usamos para la Práctica 2. Creamos un proyecto en Vivado del tipo RTL, Target Language: VHDL, Simulator Language: Mixed, la Board la Zybo de Digilent Inc. Dentro del proyecto, creamos un diseño mínimo: Create block design, añadimos el SoC Zynq de Xilinx: Add IP, automatizamos las conexiones: Run Block Automation, añadimos el controlador de memoria y el GPIO: Add IP, automatizamos las conexiones: Run Connection Automation (en GPIO, led de 4 bits), regeneramos el layout del diseño, fijamos el tamaño máximo de la BRAM a 64K: Address Editor, validamos el diseño: pulsando F6.

Una vez validado el diseño, pasamos a la generación de los ficheros HDL: IP Integrator, Generate Block Design; creamos el Bitstream: Program and Debug, Generate Bitstream.



Declaramos las variables de entorno necesarias

```
export PRJ_ROOT="/media/marvin/8868F00968EFF43A/Academico/MII/SC/Vivado/zynq-freertos"
```

```
export PLATFORM="zynq-freertos"
```

```
export PLATFORM_DIR="${PRJ_ROOT}/${PLATFORM}"
```

```
mkdir ${PRJ_ROOT}/sdk
```

Exportamos el diseño de la plataforma de Vivado al SDK, vamos al proyecto, clic derecho sobre File, Export Hardware y seleccionamos la ubicación `${PRJ_ROOT}/sdk` (o Launch SDK y seleccionamos la misma ubicación).

Pasamos a la creación de un proyecto HW en el SDK para nuestra plataforma, clic derecho sobre File, New, Project, Hardware Platform Specification, como target seleccionamos nuestro .hdf y lo nombramos como `my_zynq_platform_wrapper`. Luego en File, New, Board Support Package, como HW Platform seleccionamos nuestro `my_zynq_platform_wrapper_hw_platform`, CPU, `ps7_cortexa9_0` y OS, `standlone`, lo nombramos como `my_zynq_platform_bsp`, y con ello creamos un BSP en el SDK para nuestra plataforma, ingresando como OS Version, 4.2.

Port de FreeRTOS al Zynq Processing System de la Zybo

Primero descargamos el código fuente de FreeRTOS, antes declaramos las variables de entorno a utilizar

```
FREERTOS_VER="8.2.1"
```

```
FREERTOS="FreeRTOSV${FREERTOS_VER}"
```

```
ZIPFILE=${FREERTOS}.zip
```

```
FREERTOS_DIR="${PRJ_ROOT}/freertos"
```

```
FREERTOS_FORGE="http://sourceforge.net/projects/freertos/"
FREERTOS_SRC_DIR="${FREERTOS_DIR}/${FREERTOS}/FreeRTOS"
FREERTOS_KERNEL_DIR="${FREERTOS_SRC_DIR}/Source"
FREERTOS_PORTABLE_DIR="${FREERTOS_KERNEL_DIR}/portable"
FREERTOS_DEMO_DIR="${
{FREERTOS_SRC_DIR}/Demo/CORTEX_A9_Zynq_ZC702/RTOSDemo"
SDK_DIR="${PRJ_ROOT}/sdk"
FREERTOS_PORT_DIR="${SDK_DIR}/${FREERTOS_PORTABLE_DIR}"
```

Entonces pasamos a descargarnos FreeRTOS

```
mkdir -p ${FREERTOS_DIR}
cd ${FREERTOS_DIR}
wget ${FREERTOS_FORGE}/files/FreeRTOS/V${FREERTOS_VER}/${ZIPFILE}
unzip ${ZIPFILE}
```

Ahora copiamos los ficheros necesarios, primero creamos un directorio para el port para la Zybo dentro de workspace del SDK

```
mkdir -p ${FREERTOS_PORT_DIR}
```

Copiamos el kernel de FreeRTOS a nuestro directorio

```
cd ${FREERTOS_PORT_DIR}
mkdir ./src ./include
cp ${FREERTOS_KERNEL_DIR}/* ./src
cp ${FREERTOS_KERNEL_DIR}/include/* ./include
```

Copiamos los ficheros del port de FreeRTOS para ARM Cortex A9

```
cp ${FREERTOS_PORTABLE_DIR}/GCC/ARM_CA9/portASM.S ./src
cp ${FREERTOS_PORTABLE_DIR}/GCC/ARM_CA9/port.c ./src
cp ${FREERTOS_PORTABLE_DIR}/GCC/ARM_CA9/portmacro.h ./include
```

Copiamos la implementación de las funciones de gestión de memoria

```
cp ${FREERTOS_PORTABLE_DIR}/MemMang/heap_4.c .
```

Copiamos los ficheros imprescindibles de la demo de FreeRTOS para el Zynq

```
cp ${FREERTOS_DEMO_DIR}/src/FreeRTOSConfig.h ./include
cp ${FREERTOS_DEMO_DIR}/src/FreeRTOS_asm_vectors.S ./src
cp ${FREERTOS_DEMO_DIR}/src/FreeRTOS_tick_config.c ./src
```

Corregimos erratas (“Task.h” → “task.h” en FreeRTOS_tick_config.c)

```
sed -i 's/"Task.h"/"task.h"/g' ./src/FreeRTOS_tick_config.c
```

Ahora debemos extraer la inicialización del Zynq del fichero main.c de la demo, para ello creamos fichero `${FREERTOS_PORT_DIR}/src/setup.c`, extraemos el código de inicialización del Zynq del fichero main.c de la demo de FreeRTOS en `${FREERTOS_DEMO_DIR}/src/main.c`. Debemos agregar a los Includes, “setup.h”, y modificar las funciones así:

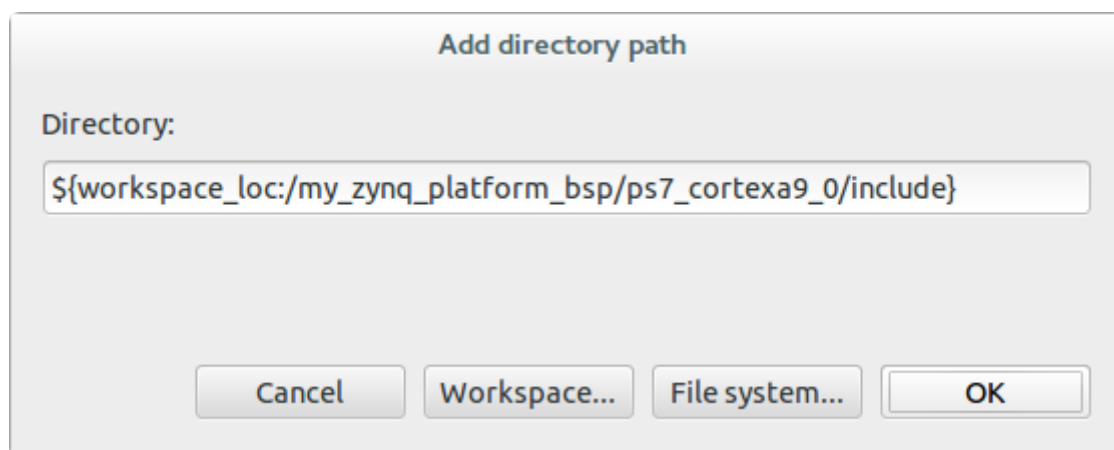
```
void prvSetupHardware( void )
```

```
void vApplicationTickHook( void )
```

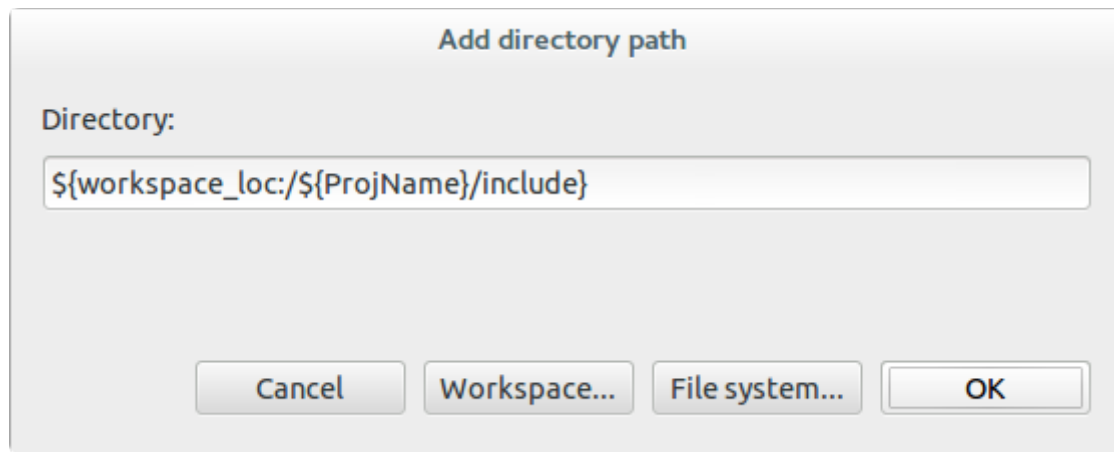
Luego creamos el fichero `${FREERTOS_PORT_DIR}/include/setup.h`, con las cabeceras de las funciones de inicialización del Zynq para incluirlas en nuestras aplicaciones.

```
#ifndef __SETUP_H
#define __SETUP_H
/* Scheduler include files. */
#include "FreeRTOS.h"
#include "task.h"
/*
 * Configure the hardware as necessary to run FreeRTOS on a Zynq SoC
 */
void prvSetupHardware( void );
/* Prototypes for the standard FreeRTOS callback/hook functions */
void vApplicationMallocFailedHook( void );
void vApplicationIdleHook( void );
void vApplicationStackOverflowHook( TaskHandle_t pxTask, char *pcTaskName );
void vApplicationTickHook( void );
#endif
```

Lo siguiente que debemos hacer es la creación de un proyecto de biblioteca para el port de FreeRTOS en SDK. En File, New, Project, C Project, debemos usar el mismo directorio al que hemos copiado los ficheros del port de FreeRTOS, lo nombramos `my_zynq_platform_freertos_port`, Project type, Xilinx ARM Static Library, ToolChains, Xilinx ARM GNU, marcamos como Debug. Luego sobre el proyecto creado, clic derecho, Project, Properties, C/C++ Build, Manage Configurations, Rename, y lo nombramos lib, y le ponemos la descripción que nos parezca, entonces la biblioteca se generará en el directorio lib. En la misma ventana, dentro C/C++ Build, en Settings, Directory, añadimos el directorio con las cabeceras de las funciones del BSP:



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



Luego, indicamos que el port de FreeRTOS depende del BSP, desde Project References, seleccionando my_zynq_platform_bsp.

Sobre el proyecto, seleccionamos Build Project. Probablemente en los archivos setup.h/setup.c, saltan errores, completar “;” donde se indique, y Build Project y listo. Con ello ya obtenemos la biblioteca con el port de FreeRTOS (libmy_zynq_platform_freertos_port.a).

Uso del port de FreeRTOS

En el SDK, en File, New, Application Project, Language, C, HW Platform, my_zynq_platform_wrapper, Processor, ps7_cortexa90, OS, standalone, en BSP, usar my_zynq_platform_bsp, usamos la localizacion default, y lo nombramos como my_zynq_platform_freertos_demo. Como templates, Hello World, luego eliminamos helloworld.c y creamos un nuevo archivo main.c con el código siguiente:

```
#include <stdio.h>

#include "platform.h"
#include "xparameters.h"
#include "xgpio.h"

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

// #include "setup.h"
#include "basic_io.h"

/* Xilinx includes. */
#include "platform.h"
#include "xparameters.h"
#include "xscutimer.h"
#include "xscugic.h"
#include "xil_exception.h"

/* Standard demo includes. */
#include "partest.h"
#include "TimerDemo.h"
#include "QueueOverwrite.h"
#include "EventGroupsDemo.h"
#include "TaskNotify.h"
#include "IntSemTest.h"

#define mainDELAY_LOOP_COUNT (0xffffffff)

XScuGic xInterruptController;

static void prvSetupHardware( void );

void vTask1 (void *pvParameters);
void vTask2 (void *pvParameters);

int main(void)
{
    prvSetupHardware();

    init_platform();

    xTaskCreate(vTask1, "Task 1", 240, NULL, 1, NULL);
    xTaskCreate(vTask2, "Task 2", 240, NULL, 1, NULL);

    vTaskStartScheduler();

    for( ; ; );
}
```

```

        cleanup_platform();
        return 0;
    }

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

        for( ;; ){
            vPrintString(pcTaskName);

            XGpio_WriteReg(XPAR_GPIO_0_BASEADDR, XGPIO_DATA_OFFSET, 0x0f);
            sleep(3);

            for (ul = 0; ul < mainDELAY_LOOP_COUNT; ul++){
                /*
                 *
                 */
            }
        }

    }

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

        for( ;; ){
            vPrintString(pcTaskName);

            XGpio_WriteReg(XPAR_GPIO_0_BASEADDR, XGPIO_DATA_OFFSET, 0x00);

            for (ul = 0; ul < mainDELAY_LOOP_COUNT; ul++){
                /*
                 *
                 */
            }
        }

    }

    static void prvSetupHardware( void )
    {
        BaseType_t xStatus;
        XScuGic_Config *pxGICConfig;

        /* Ensure no interrupts execute while the scheduler is in an inconsistent
        state. Interrupts are automatically enabled when the scheduler is
        started. */
        portDISABLE_INTERRUPTS();

        /* Obtain the configuration of the GIC. */
        pxGICConfig = XScuGic_LookupConfig( XPAR_SCUGIC_SINGLE_DEVICE_ID );

        /* Sanity check the FreeRTOSConfig.h settings are correct for the
        hardware. */
        configASSERT( pxGICConfig );
        configASSERT( pxGICConfig->CpuBaseAddress == ( configINTERRUPT_CONTROLLER_BASE_ADDRESS +
        configINTERRUPT_CONTROLLER_CPU_INTERFACE_OFFSET ) );
        configASSERT( pxGICConfig->DistBaseAddress == configINTERRUPT_CONTROLLER_BASE_ADDRESS );

        /* Install a default handler for each GIC interrupt. */
        xStatus = XScuGic_CfgInitialize( &xInterruptController, pxGICConfig, pxGICConfig->CpuBaseAddress );
        configASSERT( xStatus == XST_SUCCESS );
        ( void ) xStatus; /* Remove compiler warning if configASSERT() is not defined. */

        /* Initialise the LED port. */
        vParTestInitialise();

        /* The Xilinx projects use a BSP that do not allow the start up code to be
        altered easily. Therefore the vector table used by FreeRTOS is defined in
        FreeRTOS_asm_vectors.S, which is part of this project. Switch to use the
        FreeRTOS vector table. */
        vPortInstallFreeRTOSVectorTable();
    }

```

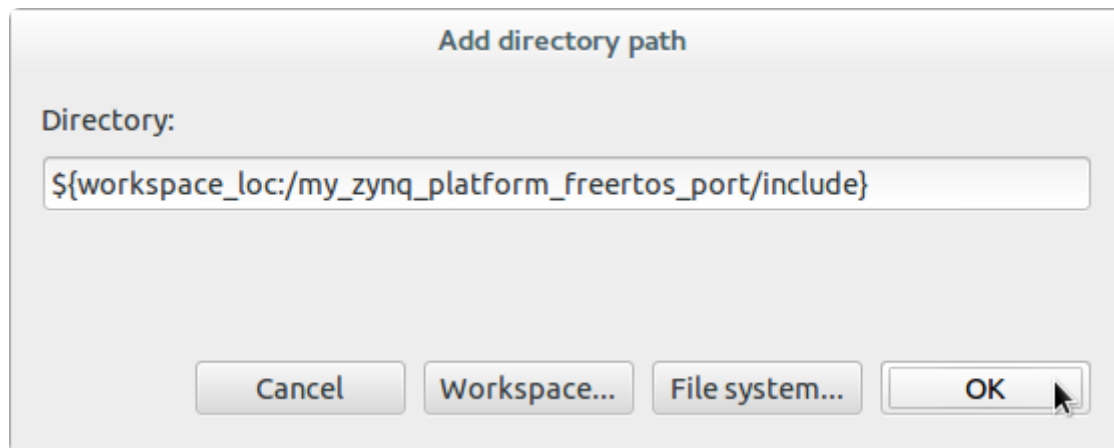
También creamos el acceso a la consola en exclusión mutua: basic_io.h/basic_io.c.

<pre> #ifndef BASIC_IO_H #define BASIC_IO_H void vPrintString(const char *pcString); void vPrintStringAndNumber(const char *pcString, unsigned long ulValue); #endif </pre>	<pre> #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){ vTaskSuspendAll(); { sprintf(cBuffer, "%s", pcString); print(cBuffer); } xTaskResumeAll(); } void vPrintStringAndNumber(const char *pcString, unsigned long ulValue){ vTaskSuspendAll(); { sprintf(cBuffer, "%s %ul\n", pcString, </pre>
---	--

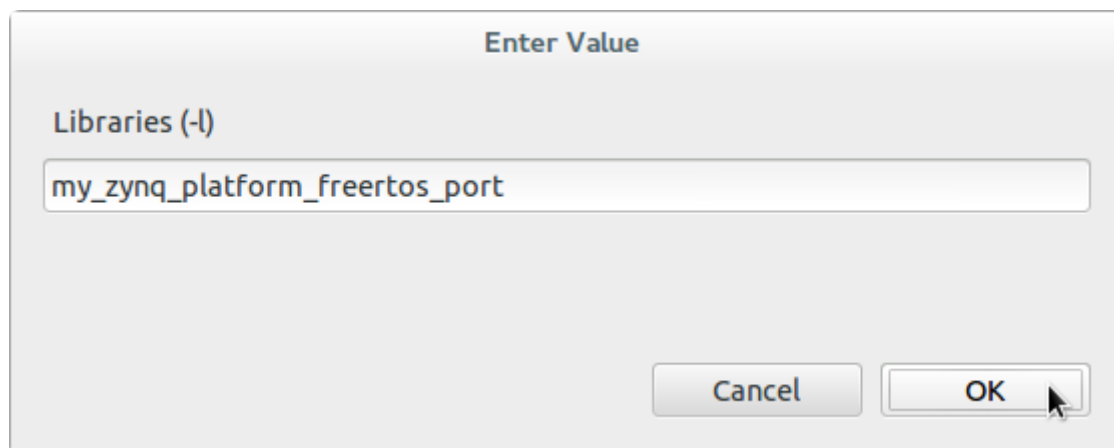
	<pre>ulValue); print(cBuffer); } xTaskResumeAll(); } }</pre>
--	---

Cambien debemos modificar el linker script (lscript.ld) para insertar los vectores de FreeRTOS, en SECTIONS, agregamos la línea: *(.freertos_vectors). Con ello insertamos los vectores de excepción de FreeRTOS al principio de la imagen de nuestra aplicación.

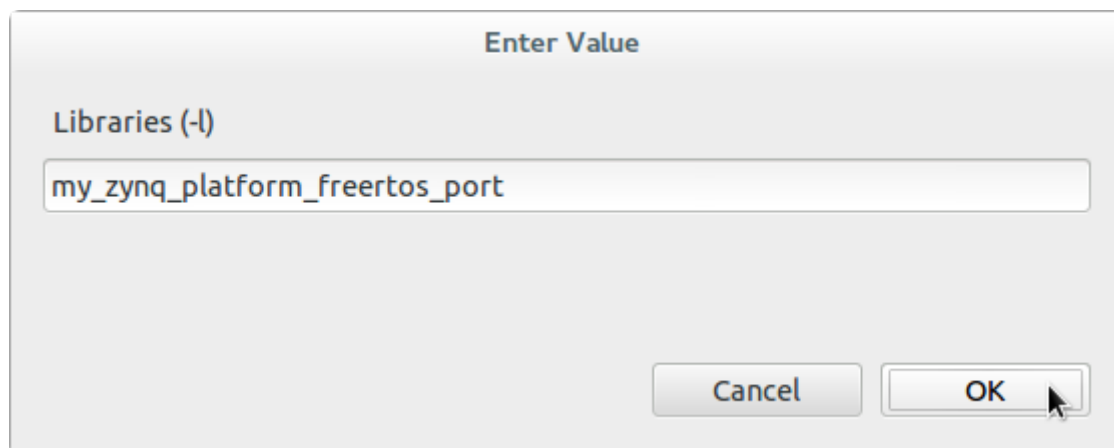
Ahora configuramos el proyecto, sobre el mismo seleccionamos Properties, C/C++ Build. Settings, Directories:



Luego en Libraries, Libraries, New



Y en Libraries search path, New



Entonces sobre el proyecto, Build Project. Si se nos presentan errores, debemos realizar el paso

anterior “Port de FreeRTOS al Zynq Processing System de la Zybo”, copiando a nuestro proyecto demo los archivos necesarios, incluyendo el main.c (renombrandolo a main2.c) en \$FREERTOS_DEMO_DIR, que podrían ser más o menos estos:

```
-rwxrwxrwx 1 marvin marvin 29345 mar 21 2015 event_groups.h
-rwxrwxrwx 1 marvin marvin 23992 mar 21 2015 event_groups.c
-rwxrwxrwx 1 marvin marvin 16850 mar 21 2015 heap_4.c
-rwxrwxrwx 1 marvin marvin 34 may 15 18:13 Xilinx.spec
-rwxrwxrwx 1 marvin marvin 10610 may 15 18:17 main_blinky.c
-rwxrwxrwx 1 marvin marvin 4853 may 15 18:30 FreeRTOS_asm_vectors.S
-rwxrwxrwx 1 marvin marvin 11636 may 15 18:30 FreeRTOSConfig.h
-rwxrwxrwx 1 marvin marvin 6552 may 15 18:31 lscript.ld
-rwxrwxrwx 1 marvin marvin 10388 may 15 18:31 lwipopts.h
-rwxrwxrwx 1 marvin marvin 5796 may 15 18:31 ParTest.c
-rwxrwxrwx 1 marvin marvin 3725 may 15 18:31 platform.c
-rwxrwxrwx 1 marvin marvin 952 may 15 18:31 platform.h
-rwxrwxrwx 1 marvin marvin 124 may 15 18:31 platform_config.h
-rwxrwxrwx 1 marvin marvin 6814 may 15 18:31 printf-stdarg.c
-rwxrwxrwx 1 marvin marvin 15767 may 15 18:32 croutine.c
-rwxrwxrwx 1 marvin marvin 10989 may 15 18:32 list.c
-rwxrwxrwx 1 marvin marvin 822 may 15 18:32 readme.txt
-rwxrwxrwx 1 marvin marvin 144430 may 15 18:32 tasks.c
-rwxrwxrwx 1 marvin marvin 34453 may 15 18:32 timers.c
-rwxrwxrwx 1 marvin marvin 22686 may 15 18:33 port.c
-rwxrwxrwx 1 marvin marvin 9393 may 15 18:33 portASM.S
-rwxrwxrwx 1 marvin marvin 11019 may 15 18:33 portmacro.h
-rwxrwxrwx 1 marvin marvin 29129 may 15 18:34 croutine.h
-rwxrwxrwx 1 marvin marvin 10164 may 15 18:34 deprecated_definitions.h
-rwxrwxrwx 1 marvin marvin 26154 may 15 18:34 FreeRTOS.h
-rwxrwxrwx 1 marvin marvin 21066 may 15 18:34 list.h
-rwxrwxrwx 1 marvin marvin 7712 may 15 18:34 mpu_wrappers.h
-rwxrwxrwx 1 marvin marvin 8490 may 15 18:34 portable.h
-rwxrwxrwx 1 marvin marvin 5249 may 15 18:34 projdefs.h
-rwxrwxrwx 1 marvin marvin 63013 may 15 18:34 queue.h
-rwxrwxrwx 1 marvin marvin 34105 may 15 18:34 semphr.h
-rwxrwxrwx 1 marvin marvin 9444 may 15 18:34 StackMacros.h
-rwxrwxrwx 1 marvin marvin 850 may 15 18:34 stdint.readme
-rwxrwxrwx 1 marvin marvin 84186 may 15 18:34 task.h
-rwxrwxrwx 1 marvin marvin 54402 may 15 18:34 timers.h
-rwxrwxrwx 1 marvin marvin 4071 may 15 18:36 AltBlckQ.h
-rwxrwxrwx 1 marvin marvin 4072 may 15 18:36 AltBlock.h
-rwxrwxrwx 1 marvin marvin 4070 may 15 18:36 AltPollQ.h
-rwxrwxrwx 1 marvin marvin 4102 may 15 18:36 AltQTest.h
-rwxrwxrwx 1 marvin marvin 4057 may 15 18:36 BlockQ.h
-rwxrwxrwx 1 marvin marvin 4056 may 15 18:36 blocktim.h
-rwxrwxrwx 1 marvin marvin 4212 may 15 18:36 comtest.h
-rwxrwxrwx 1 marvin marvin 4090 may 15 18:36 comtest2.h
-rwxrwxrwx 1 marvin marvin 4110 may 15 18:36 comtest_strings.h
-rwxrwxrwx 1 marvin marvin 4075 may 15 18:36 countsem.h
-rwxrwxrwx 1 marvin marvin 4460 may 15 18:36 crflash.h
-rwxrwxrwx 1 marvin marvin 4205 may 15 18:36 crhook.h
-rwxrwxrwx 1 marvin marvin 4058 may 15 18:36 death.h
-rwxrwxrwx 1 marvin marvin 4073 may 15 18:36 dynamic.h
-rwxrwxrwx 1 marvin marvin 4316 may 15 18:36 EventGroupsDemo.h
-rwxrwxrwx 1 marvin marvin 4148 may 15 18:36 fileIO.h
-rwxrwxrwx 1 marvin marvin 4002 may 15 18:36 flash.h
-rwxrwxrwx 1 marvin marvin 4475 may 15 18:36 flash_timer.h
-rwxrwxrwx 1 marvin marvin 4049 may 15 18:36 flop.h
-rwxrwxrwx 1 marvin marvin 4126 may 15 18:36 GenQTest.h
-rwxrwxrwx 1 marvin marvin 4069 may 15 18:36 integer.h
-rwxrwxrwx 1 marvin marvin 4168 may 15 18:36 IntQueue.h
-rwxrwxrwx 1 marvin marvin 4133 may 15 18:36 IntSemTest.h
-rwxrwxrwx 1 marvin marvin 4045 may 15 18:36 mevents.h
-rwxrwxrwx 1 marvin marvin 4149 may 15 18:36 partest.h
-rwxrwxrwx 1 marvin marvin 4056 may 15 18:36 PollQ.h
-rwxrwxrwx 1 marvin marvin 4104 may 15 18:36 print.h
-rwxrwxrwx 1 marvin marvin 4065 may 15 18:36 QPeek.h
-rwxrwxrwx 1 marvin marvin 4146 may 15 18:36 QueueOverwrite.h
-rwxrwxrwx 1 marvin marvin 4131 may 15 18:36 QueueSet.h
-rwxrwxrwx 1 marvin marvin 4136 may 15 18:36 QueueSetPolling.h
-rwxrwxrwx 1 marvin marvin 4069 may 15 18:36 recmutex.h
-rwxrwxrwx 1 marvin marvin 4065 may 15 18:36 semtest.h
-rwxrwxrwx 1 marvin marvin 5369 may 15 18:36 serial.h
-rwxrwxrwx 1 marvin marvin 4107 may 15 18:36 TaskNotify.h
-rwxrwxrwx 1 marvin marvin 4144 may 15 18:36 TimerDemo.h
-rwxrwxrwx 1 marvin marvin 7693 may 15 18:38 FreeRTOS_tick_config.c
-rwxrwxrwx 1 marvin marvin 82772 may 15 18:43 queue.c
-rwxrwxrwx 1 marvin marvin 161 may 17 17:01 basic_io.h
-rwxrwxrwx 1 marvin marvin 2877 may 17 17:58 main.c
-rwxrwxrwx 1 marvin marvin 14495 may 17 17:58 main2.c
-rwxrwxrwx 1 marvin marvin 547 may 22 13:43 basic_io.c
```

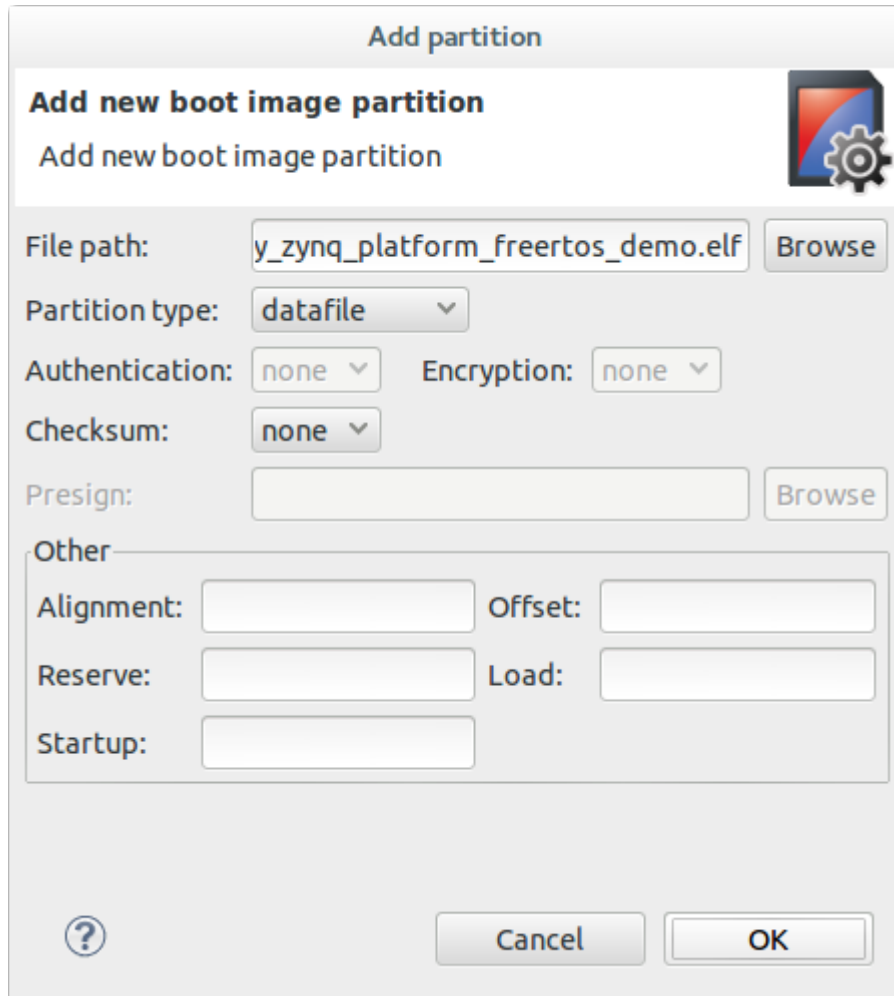
Ahora si volvemos a construir el Proyecto, ya debería no dar errores.

Generación del First Stage Boot Loader y preparación de la imagen de arranque

Para la creación del First Stage Boot Loader mediante el SDK, en File, New, Application Project, loo nombramos como my_zynq_platform_fsbl, default location, HW Platform, my_platform_wrapper, Processor, ps7_cortexa9o, Language C, OS, standalone, nuevo BSP, nombrándolo como my_zynq_platfomr_fsbl_bsp, luego Zynq FSBL. Podemos modificar

fsbl_debug.h y agregar #define FSBL_DEBUG_INFO, con esto logramos que nos de más información en consola.

Creamos la imagen de arranque, en el proyecto FSBL, Create Boot Image. Luego, añadimos nuestra aplicación a la imagen, en Boot image partitions, Add:



Add partition

Add new boot image partition

Add new boot image partition

File path:

Partition type:

Authentication: Encryption:

Checksum:

Presign:

Other

Alignment: Offset:

Reserve: Load:

Startup:

Entonces, generamos la imagen de arranque, y deberíamos quedarnos así:

Create Boot Image

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

Architecture: Zynq

☐ Create new BIF file ☒ Import from existing BIF file

Import BIF file path: /media/marvin/8868F00968EFF43A/Academico/MII/SC/Vivado/zynq-freertos/sdk/my_zynq_ Browse...

Basic Security

Output BIF file path: /media/marvin/8868F00968EFF43A/Academico/MII/SC/Vivado/zynq-freertos/sdk/my_zynq_ Browse...

UDF data: Browse...

☐ Split Output format: BIN

Output path: /media/marvin/8868F00968EFF43A/Academico/MII/SC/Vivado/zynq-freertos/sdk/my_zynq_ Browse...

Boot image partitions

File path	Encrypted	Authenticated	
(bootloader) /media/marvin/8868F00968EFF43A/A	none	none	Add Delete Edit Up
/media/marvin/8868F00968EFF43A/Academico/MI	none	none	
/media/marvin/8868F00968EFF43A/Academico/MI	none	none	

Preview BIF Changes
Cancel
Create Image

Ahora preparamos la placa

```
SDCARD_DIR="/media/marvin/E91B-0925"
```

```
IMG_ROOT="/media/marvin/8868F00968EFF43A/Academico/MII/SC/Vivado/zynq-freertos/sdk/my_zynq_platform_fsbl/bootimage"
```

```
cd ${IMG_ROOT}
```

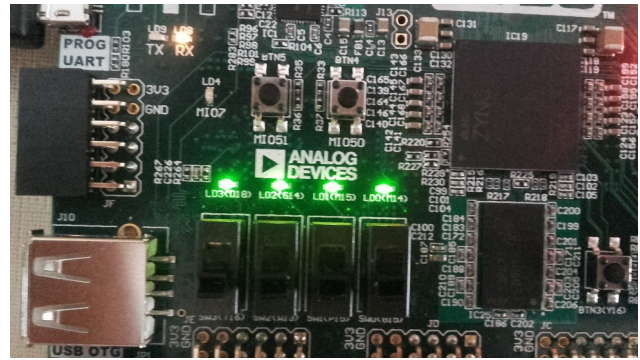
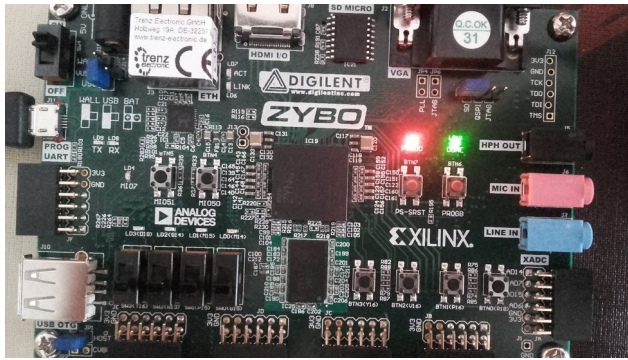
```
cp BOOT.bin ${SDCARD_DIR}
```

Y con gkterm, que lo tenemos instalado de la práctica pasada, pasamos a probarlo

```
sudo gkterm
```

En Configuration, Port, seleccionamos el puerto USB objetivo, y Baud Rate ponemos al máximo.

Si utilizamos la demo de FreeRTOS, veremos esta salida



Inconvenientes

Como se pudo ver en la sección de Desarrollo de este trabajo, los principales problemas se dieron con la librería creada, primero en el archivo *setup* saltaban errores, poniendo “;” a ciertas líneas se remediaba, haciendo build de vuelta.

En la generación de la librería, no hubieron grandes problemas, pero al querer usarla con la demo no funciona, por ello se procedió a copiar los fuentes desde el zip de FreeRTOS a nuestra demo y combinar el archivo *main.c* nuestro con el de la demo de FreeRTOS.

Conclusiones

En este trabajo he aprendido el Desarrollo de aplicaciones de tiempo real con FreeRTOS en la Zybo, antes la creación del Port de FreeRTOS al Zynq Processing System de la Zybo, que aunque al querer usarlo falle, es una grata experiencia adquirir estos conocimientos, aprendiendo todas estas tecnologías, entregando algo funcional pese a los problemas con la librería generada.

Bibliografía consultada

González Peñalver, J. (2015). FreeRTOS. Universidad de Granada.

Anexos

Archivo necesario para arrancar el sistema: *boot.bin*, y el SDK del proyecto comprimido.