Instalación y configuración de FreeRTOS en la placa Zybo

 |

Realizado por:

Pablo Ráez Sánchez y Adrián Ripoll Casas

PSETR

Contenido

[Introducción 2](#_Toc421635219)

[Materiales necesarios 2](#_Toc421635220)

[Hardware 2](#_Toc421635221)

[Zybo 2](#_Toc421635222)

[Software 3](#_Toc421635223)

[Vivado 3](#_Toc421635224)

[FreeRTOS 3](#_Toc421635225)

[Caso de estudio 5](#_Toc421635226)

[Nombre del sistema 5](#_Toc421635227)

[Requisitos Hardware-Software 5](#_Toc421635228)

[Configuración del sistema 5](#_Toc421635229)

[Implementación 6](#_Toc421635230)

[Observación 7](#_Toc421635231)

[Conclusiones 7](#_Toc421635232)

[Bibliografía 7](#_Toc421635233)

# Introducción

Zybo (ZYnq BOard) es una placa desarrollada por Digillent, cuya principal característica es que incluye el AP SoC (All Programmable System-on-Chip) Zynq 7010 que incluye un procesador Cortex A9 de doble núcleo y una FPGA Artex-7.

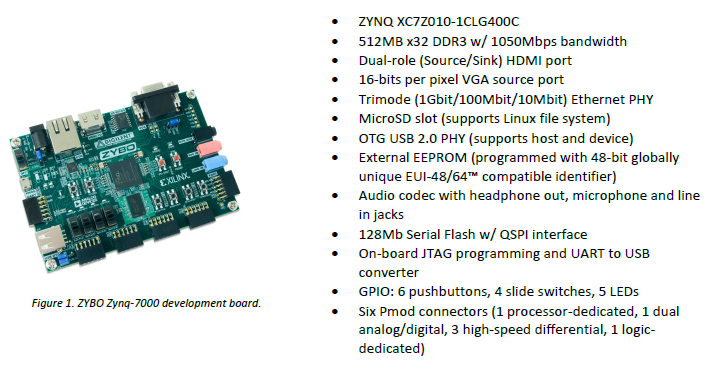
El objetivo del presente proyecto es la instalación de un sistema operativo en tiempo real, concretamente FreeRTOS. La razón por la que instalar el sistema operativo es que sea este el encargado de gestionar la memoria y planificar las distintas tareas que se programen, satisfaciendo los requisitos temporales que se impongan.

# Materiales necesarios

## Hardware

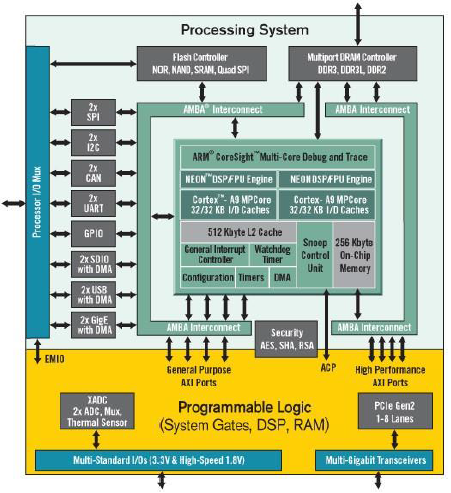
### Zybo

La placa dispone de las siguientes características:



Hay que tener en cuenta que para que los distintos componentes de la placa funcionen, debe de elaborarse un sistema en el que se incluyan los distintos cores-IP que van a utilizarse y las correspondientes interconexiones. Todo esto se debe realizar con el software Vivado.

Por otro lado, en la siguiente imagen se puede apreciar la arquitectura del sistema:



Adicionalmente, es necesario un cable USB para poder interactuar con la placa.

## Software

### Vivado

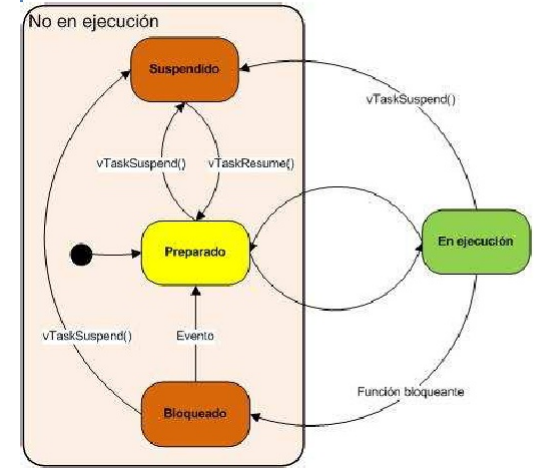
Vivado es una suite que incluye distintos programas que ayudan a programar la placa Zybo. Entre otros, incluye los siguientes programas:

* Vivado: Es el software principal, que se encarga de configurar la FPGA y la placa, permitiendo definir el sistema con los cores-IP que se necesiten. Además, también permite la creación de nuevos cores-IP.
* Vivado SDK: Basado en *Eclipse*, nos permite crear proyectos de software que luego correrá la placa, generando el correspondiente BSP.
* Vivado HLS: Programa orientado a codiseño. Permite generar código HDL a partir de código en C, lo cual es útil si alguna función software que consume mucha CPU quiere pasarse a hardware.

### FreeRTOS

Este es el sistema operativo que se va a instalar. Se necesitan los distintos archivos fuente, para incluirlos en el BSP.

Se trata de un Kernel de Tiempo real totalmente gratuito de código abierto diseñado para ocupar poca memoria y sencillo de utilizar. Entre sus principales características destacan:

* Permite el uso de procesos y corrutinas por separado o a la vez.
* Se puede considerar un microkernel, puesto que solo ofrece un planificador y una API de comunicación de procesos.
* Cada proceso es considerado un programa independiente, además, cada proceso no debe salir de su función de implementación nunca.
* En un sistema con un solo procesador, solamente se podrá ejecutar una tarea cada vez. Dichas tareas podrán estar en dos estados: en ejecución o no en ejecución. A su vez, éstas últimas pueden estar suspendidas, bloqueadas o preparadas. 
* En cuanto a la gestión de memoria, se ofrecen dos esquemas principales y un tercer esquema consistente en un esqueleto para crear esquemas propios:
  + Esquema 1: Es el más simple. No permite liberar una zona de memoria una vez se haya asignado algo allí. Este esquema es adecuado para RTOS que cumplan con la única condición de que todas las tareas y colas se creen antes de que se inicie el kernel.
  + Esquema 2: Utiliza un mejor algoritmo de ajuste y permite liberar bloques de memoria ya asignados. Sin embargo, no ofrece reordenación de memoria, lo que puede producir fragmentación en la RAM. Este esquema es adecuado para RTOS más pequeños que requieran crear dinámicamente las tareas.
* FreeRTOS permite configurar un conjunto de parámetros para optimizar el sistema operativo. Dicha parametrización se realiza en “FreeRTOSConfig.h”

# Caso de estudio

## Nombre del sistema

Zybo con FreeRTOS

## Requisitos Hardware-Software

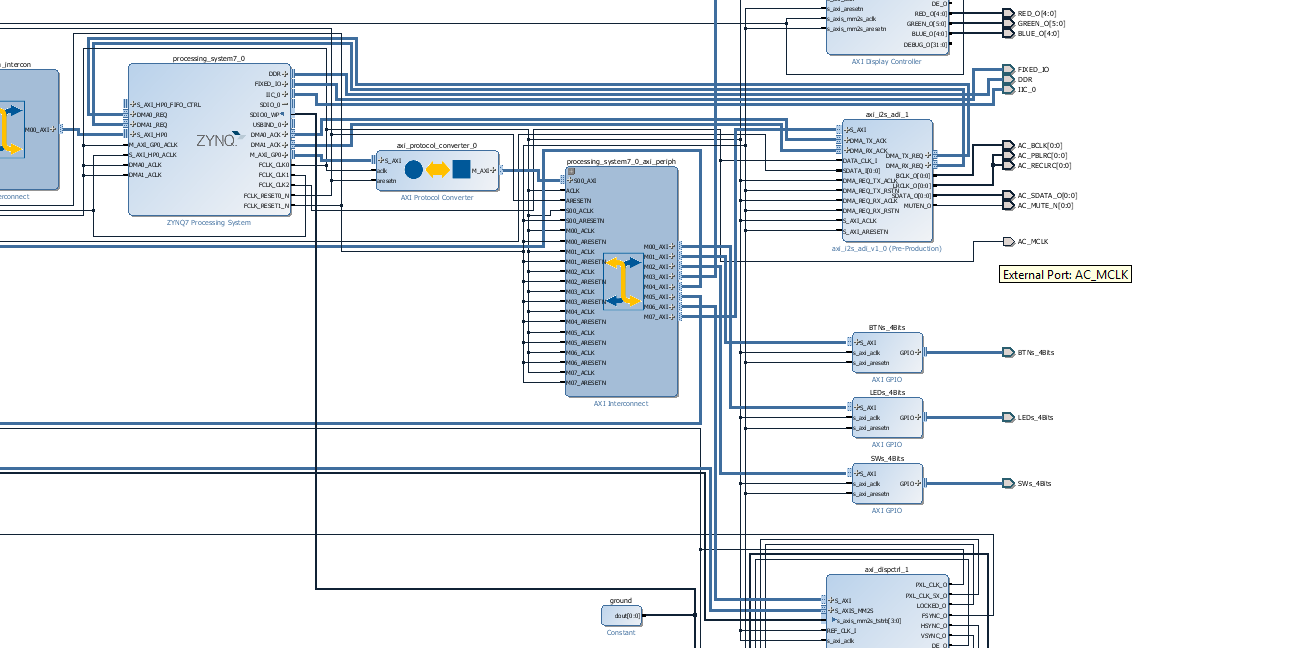
Disponer de un proyecto base que incluya toda la configuración hardware de la placa, es decir, con todos los periféricos integrados. En nuestro caso:

<https://github.com/kharenzze/Zybo/tree/master/zybo_base_system>

Por supuesto, es necesaria una placa Zybo, el software Vivado y el código fuente de FreeRTOS.

## Configuración del sistema

En primer lugar, hay que configurar el hardware. Para ello, utilizando Vivado, se importa un proyecto que incluye todos los elementos de los que dispone la placa. A continuación, se muestra una imagen del diagrama de bloques parcial del proyecto:



A partir de este proyecto, se genera un bitstream para configurar la FPGA, y este se exporta al SDK para comenzar a trabajar la parte software. Cabe destacar que los distintos cores-IP que aparecen en el diagrama de bloques del proyecto tienen asignada una dirección en el bus AXI, la cual se utiliza para poder comunicarse con estos.

Algo interesante que hace la herramienta es crear DEFINEs haciendo referencia a estas direcciones. Para ello, se basa en el nombre del periférico correspondiente. Un ejemplo para la dirección de los LEDs:

#define XPAR\_LEDS\_4BITS\_BASEADDR 0x41210000

## Implementación

Una vez dentro del SDK, hay que crear un proyecto nuevo, indicando que además que se debe generar un BSP al compilar el proyecto.

En este nuevo proyecto, se deben importar los ficheros fuente de FreeRTOS. Una vez importados, el siguiente paso es editar el archivo FreeRTOSConfig.h para definir los parámetros del sistema operativo en función del hardware del que dispone la placa Zybo.

Finalmente, se crea un main.c, en el cual se incluyen varias tareas. En la función main, se inician las tareas utilizando primitivas de FreeRTOS, donde se indica como asignar la memoria RAM y que prioridad tiene cada tarea. Por último, se ejecuta el planificador de tareas.

int main( void )

{

/\* Configure the hardware ready to run the demo. \*/

prvSetupHardware();

/\* The mainSELECTED\_APPLICATION setting is described at the top

of this file. \*/

#if( mainSELECTED\_APPLICATION == 0 )

{

xTaskCreate(AXILedBlink,"AXI\_LED", configMINIMAL\_STACK\_SIZE, NULL, (tskIDLE\_PRIORITY + 2), NULL);

xTaskCreate(tarea2, "btn\_sw", configMINIMAL\_STACK\_SIZE, NULL, (tskIDLE\_PRIORITY + 3), NULL);

vTaskStartScheduler();

}

#elif( mainSELECTED\_APPLICATION == 1 )

{

main\_full();

}

#else

{

main\_lwIP();

}

#endif

/\* Don't expect to reach here. \*/

return 0;

}

Como se puede ver, las funciones del sistema operativo utilizadas son vTaskCreate() y VTaskScheduler().

En nuestro caso, hay 2 tareas:

* Una que hace que los LEDs de la placa parpadeen continuamente
* Otra, que envía por UART un texto indicando el estado de los botones y switches de los que dispone la placa.

Una vez terminado el código, se compila. Para hacer funcionar el sistema, primero hay que programar la FPGA y acto seguido, correr el programa. Cabe destacar que el estado de la placa es volátil: es decir, si se apaga, la placa pierde toda la configuración. Si se quiere que la placa inicie siempre con el BSP creado, hay que crear una imagen e insertarla en una tarjeta microSD. Además, habría que cambiar un jumper para indicar que se inicie con la microSD.

## Observación

Si se siguen los pasos anteriormente mostrados, se puede observar que el funcionamiento es el correcto, y se obtienen los resultados esperados.

# Conclusiones

Por un lado la placa Zybo, aun siendo la placa *“low-cost”* de Xilinx tiene capacidad suficiente para realizar todo tipo de aplicaciones, por otro, la inclusión de FreeRTOS a la misma proporciona un Sistema Operativo muy ligero y personalizable, lo que permite optimizar el trabajo que se desee realizar.

Una vez que se ha trabajado con estas herramientas, se puede ver que estamos ante un sistema muy potente que nos permite realizar desde tareas muy simples como el encendido de Leds en determinados momentos, hasta montar un sistema complejo con diversas aplicaciones corriendo a la vez, como podrían ser varias aplicaciones servidor funcionando en tándem.

# Bibliografía

Repositorio del proyecto

<https://github.com/kharenzze/Zybo/tree/master/FreeRTOS>

API

<http://www.freertos.org/a00106.html>

Tutorial seguido

<http://rishifranklin.blogspot.in/2015/04/freertos-on-xilinx-zynq-zybo-single-core.html?m=1>