UNIVERSIDAD MICHOACANA
DE
SAN NICOLÁS DE HIDALGO

FACULTAD DE INGENIERÍA
ELÉCTRICA

Implementación de un sistema operativo de tiempo real en un procesador embebido NIOS II

Tesis

Que para obtener el título de:

INGENIERA EN ELECTRÓNICA

Presenta:

Imelda Hernández García

Asesor:

M.C. Alberto Carlos Salas Mier

Enero, 2018
Morelia, Michoacán

Agradecimientos

- Primeramente quiero agradecer a Dios por permitirme llegar a esta etapa de mi vida y darme la oportunidad de lograr esta meta.

- A mis padres porque sin ellos esto no hubiera sido posible, se que han hecho muchos sacrificios para que yo pudiera llegar hasta donde estoy, y eso se los agradezco infinitamente, gracias papá y mamá por regalarme un futuro diferente a lo previsto, a mis hermanos que siempre estuvieron ahí para pelear y para echarme porras cuando era necesario. A mi hermano René por haberme brindado apoyo para poder seguir estudiando, gracias por haber estado en los momentos difíciles, gracias por cargarme para que yo pudiera ir a la escuela y gracias por ser mi compañero de aventuras desde niños

- A todos mis familiares que estuvieron ahí para tenderme una mano en mis momentos de dificultad, a mis tíos, primos y demás familiares que me han apoyado y que han creído en mí.

- A mi familia postiza, les agradezco todos sus consejos y su agradable compañía, fueron tantos momentos vividos a su lado, gracias porque me hicieron sentir en familia, porque estar lejos de la familia no es nada fácil sin embargo siempre estuvieron ahí cuando necesite un abrazo o cuando me merecía un regaño. Y también agradecer a todos mis amigos y compañeros con los que tuve el honor de convivir a lo largo de este caminar.

- A los profesores que tuve el placer de conocer durante toda la carrera unos buenos, otros malos, en fin de todo tipo de profesores conocí en esta Facultad, sin embargo unos aportaron más que otros, y otros más me motivaron a seguir adelante, no bastaría una sola hoja para nombrarlos a todos, pero muchas gracias por su paciencia y dedicación, me llevo lo mejor de cada uno de ustedes.

- Muy en especial agradezco a mi asesor de tesis al Maestro en Ciencias Alberto Carlos Salas Mier, por toda la paciencia que me tuvo a lo largo del desarrollo de este trabajo, gracias por motivarme en cada una de mis depresiones, por soportar mis berrinches, por darme ánimos, por creer en mis capacidades y por hacer que yo me las crea, gracias por todo, es usted una gran persona y un gran profesor, me llevo cada una de sus enseñanzas y sus consejos.

- Agradezco a la Facultad de Ingeniería Eléctrica por brindarme la oportunidad de permitirme hacer una carrera universitaria.

Índice general

Índice de figuras

Índice de tablas

Resumen

En el presente trabajo se documenta la implementación de un sistema operativo de tiempo real en un procesador embebido; se documentan también los elementos fundamentales para tal desarrollo: sistemas digitales y sistemas operativos.

Se carga un procesador embebido NIOS II en una FPGA Cyclone IV, y sobre él, el sistema operativo de tiempo real FreeRTOS, obteniéndose de esta manera la plataforma objetivo de la tesis.

Finalmente se realiza la comparación de la facilidad de diseño utilizando la estrategia tradicional con un microcontrolador y usando el esquema descrito previamente.

PALABRAS CLAVE: Sistemas Operativos de Tiempo Real, Microcontroladores, Sistemas Digitales, Arquitectura de Computadoras, FPGA.

Abstract

In the present work, the implementation of a real-time operative system in a embeded processor is documented; fundamental elements of desarrollate also documented: digital system and operative system. A NIOS II processor embeded into FPGA Cyclone IV, and over it, the real-time operative system, thus obtaing the target platform of thesis.

Finally, the comparison of the design facility is made using the traditional strategy with a microcontroller and using the previously described scheme.

En memoria de mi abuelo Enrique, gracias por la niñez tan feliz que viví a tu lado, por tus enseñanzas de vida y por inculcarme el respeto hacia la madre tierra. Siempre vivirás en mi corazón abuelito.
A mi papá Odón y a mi madre hermosa Carmela, por el gran sacrificio que han hecho para darme un futuro diferente. Los amo, gracias por todo.

CAPÍTULO 1

Introducción

En este capítulo se describen los objetivos del proyecto, incluye la justificación, la motivación, una introducción de los sistemas operativos, y por último la descripción de los capítulos posteriores.

1.1. Objetivos

El presente trabajo tiene como objetivos reunir en un solo documento los principios básicos de la implementación de sistemas embebidos, describir las particularidades de la arquitectura específica NIOS II en la FPGA Cyclone IV, así como el funcionamiento y características de Qsys. Se documentarán los principios de los sistemas operativos de tiempo real y a partir de esto se seleccionará alguno adecuado para la arquitectura de hardware propuesta. Se describirán las especificaciones técnicas del sistema operativo de tiempo real elegido. se realizará la implementación del sistema operativo en el procesador Nios II. Por último se llevará a cabo una comparación para determinar las ventajas de diseño bajo el esquema propuesto (NIOS II + RTOS) contra la estrategia tradicional que usa el microcontrolador MSP430F5529.

1.2. Justificación

Los sistemas operativos de tiempo real tienen una aplicación muy extensa en el área de control, por ejemplo en el procesamiento de señales digitales de TV, el control de despegue de un avión, etc. El campo de aplicación de estos sistemas es muy amplio, por lo tanto, es indispensable entender su funcionamiento y conocer sus características para saber la diferencia que existe entre un RTOS y un sistema operativo convencional. Cabe mencionar que existen sistemas

operativos de tiempo real diseñados para sistemas embebidos, y tienen la facilidad de que ya se cuenta con el código de configuración para diferentes arquitecturas.

1.3. Motivación

A los alumnos del programa de Ingeniería Electrónica de la Facultad de Ingeniería Eléctrica, hasta el momento no se da una introducción a los sistemas operativos de tiempo real, este material debería de ser conocida por los alumnos de este programa, por la amplia aplicación de este tipo de sistemas en el área de control y de los sistemas embebidos. Aparte, algunos sistemas de control son críticos, y cuando no se tiene una respuesta en el tiempo establecido, el sistema de control tiende a fallar.

1.4. Reseña histórica

En los años 40 hizo su aparición la primera computadora, una computadora de grandes dimensiones y con gastos de energía muy elevados,conforme pasaron los años ha ido mejorando hasta llegar a la computadora que conocemos actualmente, en esta evolución de la computadora se introduce el concepto de sistema operativo.
Un sistema operativo se puede definir de dos formas: como una máquina extendida con el fin de facilitar la tarea de los usuarios o programadores de la computadora, ocultando todas las complicaciones de cada uno de los componentes internos de esta, y también funciona como organizador de procesos y recursos de la computadora.[1]

1.4.0.1. Evolución de los sistemas operativos

A finales de los 70 y principios de los 80 se introducen los Sistemas BATCH o por lotes que solo realizaban una tarea a la vez.
A finales de los 80 aparecen los sistemas de tiempo compartido, estos eran multiusuarios.
Durante la década de los 80 también aparecen los sistemas para computadores personales decrementando el costo del hardware, diseñado para ser usado por solamente un usuario y se mejoró la interacción con el usuario haciendo una interfaz más agradable con las ventanas de trabajo que conocemos actualmente.
A principios de los 90 se introducen los sistemas paralelos, sistema en donde se tiene más de un procesador permitiendo la ejecución simultanea de procesos.
Con el tiempo este concepto ha ido evolucionando hasta dar paso a los sistemas operativos de

tiempo real, estos sistemas deben de dar un resultado en cierto tiempo, de lo contrario se dice que el sistema ha fallado.[1]

1.5. Organización de la tesis

En el segundo capítulo se describen los circuitos combinacionales y secuenciales, así como flip-flops y los tipos de registros.
Más adelante se tiene la descripción de la arquitectura y y componentes de una computadora básica; también se abordan los conceptos, antecedentes y características de los sistemas operativos. Para finalizar este capítulo, se da una breve descripción de los sistemas operativos de tiempo real.
En el tercer capítulo, se da una introducción al software utilizado, es decir FreeRTOS sus características, así como la descripción de las tareas y la forma de crear una tarea usando este sistema operativo.
También se describe Qsys, una herramienta de Quartus utilizada durante el desarrollo de este trabajo, las características de esta, así como las ventajas y la facilidad de manejo de esta herramienta.

En el capítulo 4 se describe paso a paso el desarrollo de pruebas realizados durante este trabajo , se abordan también los resultados y se anexan los códigos utilizados para cada una de las pruebas, así como la explicación de cada parte de estas. En el capítulo 5 se describen las conclusiones de este trabajo y se abordan los trabajos futuros

CAPÍTULO 2

Marco teórico

2.1. Lógica digital

Cuando hablamos de lógica digital, básicamente hablamos de circuitos lógicos, dentro de estos se encuentran los circuitos lógicos combinacionales y los circuitos lógicos secuenciales.

2.1.1. Lógica combinacional

Los circuitos combinacionales son compuertas lógicas, su salida depende solamente de los estados en las que se encuentran sus entradas en ese momento, sin tomar en cuenta el estado anterior. Los circuitos combinacionales procesan la información específica de forma completamente lógica o por medio de funciones Booleanas. [2]

2.1.2. Lógica secuencial

Los circuitos secuenciales funcionan de una forma muy diferente a los circuitos combinacionales, las entradas de este tipo de circuitos son entradas independientes una con un estado X y la otra se toma de la salida o estado anterior, es decir, su estado actual depende del estado anterior. Su comportamiento se determina en el proceso de diseño mediante una máquina de estados. Existen dos tipos de circuitos secuenciales, los síncronos y los asíncronos. La salida de un circuito secuencial síncrono cambia dependiendo de un ciclo de reloj, ya no depende solamente de las entradas y del estado anterior. Un circuito secuencial asíncrono se puede ver como un circuito combinacional con retroalimentación. En estas no se necesita de la presencia de un reloj que controle el comportamiento del circuito, el cambio en la salida se presenta cuando se modifica el estado de las entradas. Debido a la inestabilidad que se presenta al haber una retroalimentación, estos circuitos no son muy utilizados como los circuitos secuenciales síncronos. [2]

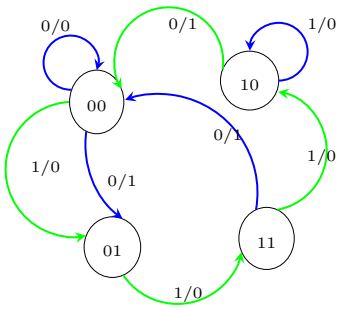

FIGURA 2.1: Diagrama de estado de un circuito secuencial

Q	S	R	$Q(t+1)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	Indeterminado
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	Indeterminado

TABLA 2.1: Tabla de verdad del flip-flop tipo RS

2.1.2.1. Diagrama de estado

Durante el proceso de diseño se dispone de una tabla de estados, las cuales se pueden representar en un diagrama de estado. En este tipo de diagramas el estado es representado por un círculo y la transición mediante líneas que conectan a los círculos. En la figura 2.1 se muestra un ejemplo, la línea que conecta del estado 00 al 01 se identifica con 1/0, es decir, que cuando el circuito secuencial esta en el estado presente 00 y la entrada es 1, la salida es 0. Después de una transición de reloj, el circuito pasa al estado siguiente 01. [3]

2.1.3. Flip-flops

Los flip-flops son circuitos compuestos por celdas capaces de almacenar un bit de información, elementos de memoria utilizados en los circuitos secuenciales síncronos. Un flip-flop consta de dos entradas y son capaces de almacenar un bit de información de manera indefinida; al cambiar la señal de entrada, cambia el valor del bit almacenado, esto siempre y cuando el circuito se encuentre energizado, una vez que se quita la alimentación, se pierde la información.[2] Existen varios tipos de flip-flops: flip-flop tipo RS, flip-flop tipo D, flip-flop tipo JK y flip-flop tipo T. El flip-flop tipo RS consiste en un flip-flop básico NOR y dos compuertas AND, ver figura 2.2.

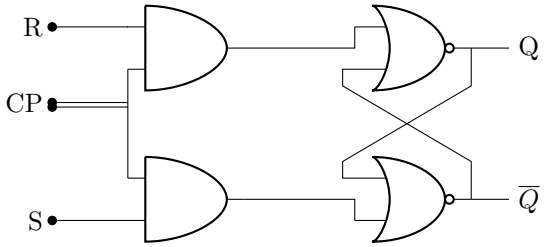

FIGURA 2.2: Diagrama lógico del flip-flop tipo RS

Q	D	$Q(t+1)$
0	0	0
0	1	1
1	0	0
1	1	1

TABLA 2.2: Tabla de verdad del flip-flop tipo D

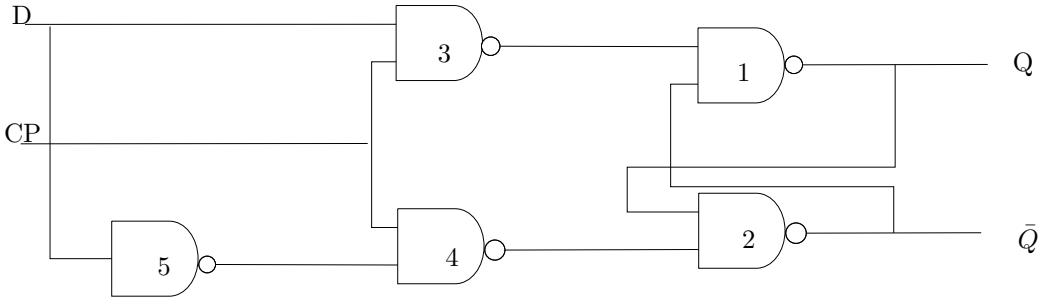

FIGURA 2.3: Diagrama lógico del flip-flop tipo D

Cuando el pulso de reloj está en 0, las salidas de las compuertas AND permanecen en bajo o 0. Cuando el pulso de reloj conmuta al estado alto o 1, se permite el paso de la información al flip-flop básico. El comportamiento característico se puede observar en la tabla 2.1. El flip-flop tipo D es una modificación del flip-flop tipo RS, el diagrama lógico se muestra en la figura 2.3. Este tipo de flip-flop se llama algunas veces bloqueador D con compuertas. La tabla característica se muestra en la tabla2.2.

El flip-flop tipo JK, es básicamente un flip-flop RS con retroalimentación, en el tipo RS se tenía un estado indeterminado, ese estado se define en el flip-flop tipo JK,la tabla característica de ésta se muestra en la tabla 2.3. El flip-flop tipo T, es el flip-flop tipo JK con la diferencia de que está es de una sola entrada. Su nombre deriva de su capacidad de variar su estado ("Toggle", conmutar cada determinado tiempo). La tabla característica se muestra en la tabla 2.4.

Q	J	K	$Q(t+1)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

TABLA 2.3: Tabla de verdad del flip-flop tipo JK

Q	T	$Q(t+1)$
0	0	0
0	1	1
1	0	1
1	1	0

TABLA 2.4: Tabla de verdad del flip-flop tipo T

2.1.4. Registros

Un registro es un grupo de flip-flops, cada uno de estos flip-flops tiene capacidad de almacenar un bit de información, por lo tanto un registro de n bits es un grupo de n flip-flops capaz de almacenar n bits de información. Un registro está compuesto por un grupo de flip-flops y compuertas que controlan su transición. Los flip-flops retienen la información binaria y las compuertas controlan cuándo y cómo se transfiere la información al registro.

Existen diversos tipos de registros en el mercado. El registro más simple es el que se compone solamente de flip-flops. En la figura 2.4 se muestra el registro compuesto de 4 flip-flops tipo D. La entrada común de reloj dispara todos los flip-flops en el flanco de subida de cada pulso y los datos binarios de las entradas. La información binaria se puede consultar en cada una de las 4 salidas, para consultar lo que contiene el registro. La entrada para borrar va a una terminal especial en cada flip-flop. Cuando esta entrada va a 0, todos los flip-flops se borran de manera asíncrona. La entrada para borrar es útil para borrar el registro a ceros antes del funcionamiento sincrónizado por reloj. La entrada para borrar debe mantenerse en 1 lógico durante la operación sincronizada por reloj normal. La transferencia de nueva información a un registro se llama carga de registro. [3]

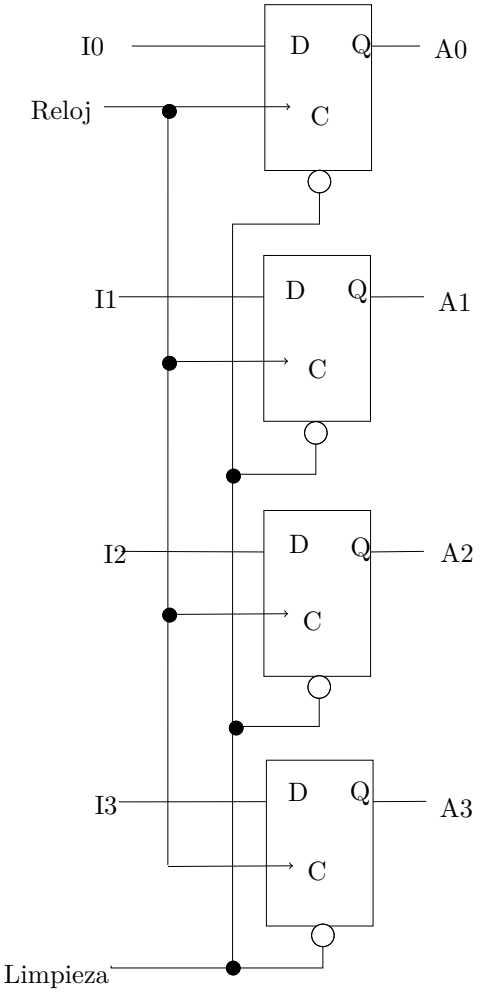

FIGURA 2.4: Registro de 4 bits

2.1.5. Tipos de registros

2.1.5.1. Registro con carga paralela

En la figura 2.5, se muestra un registro paralelo de 4 bits con una entrada de control de carga que se dirige a través de compuertas y hacia las entradas D. Las entradas C reciben pulsos de reloj en todo momento. La compuerta de acoplamiento en la entrada de reloj reduce el requisito de potencia del generador de reloj. Se requiere menos potencia cuando el reloj se conecta a una sola compuerta de entrada en lugar del consumo de energía que se requeriría para las cuatro entradas si no se usara el acoplador. La entrada de carga en el registro determina la acción que va a tomarse con cada pulso de reloj. Cuando la entrada de carga es 1, los datos en las 4 entradas se transfieren al registro con la siguiente transición positiva del pulso de reloj. Cuando la entrada de carga es 0, las entradas de datos se inhiben y las entradas D de los flip-flops se

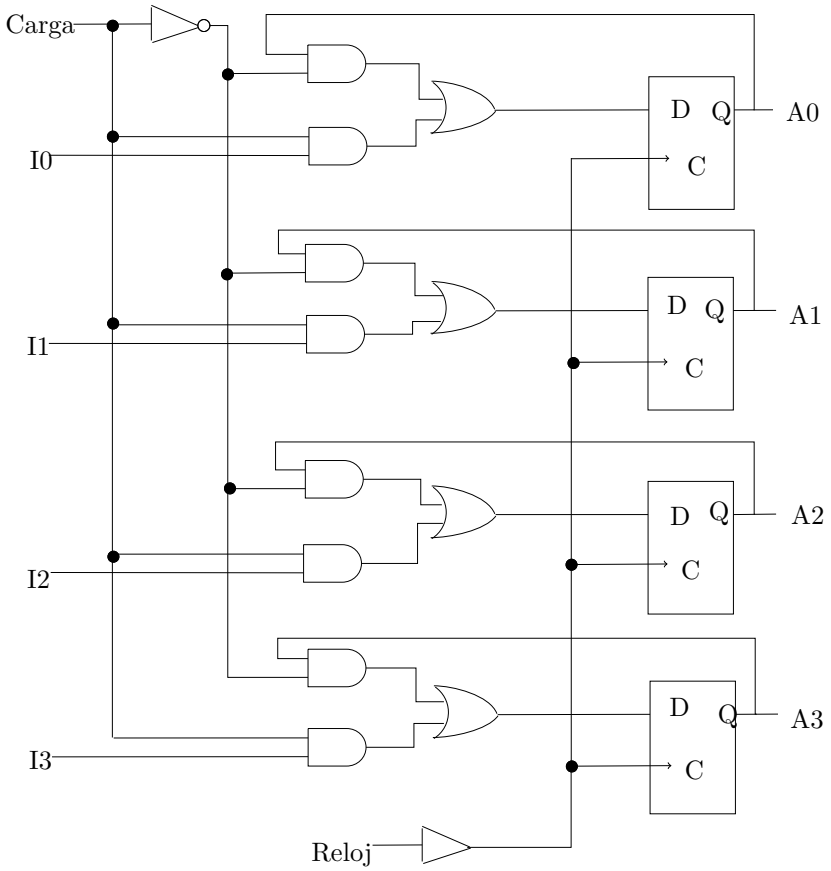

FIGURA 2.5: Registro de 4 bits con carga paralela

conectan a sus salidas. La conexión de retroalimentación de la salida a la entrada es necesaria porque los flip-flops D no tienen una condición "sin cambio". Con cada pulso de reloj, la entrada D determina el estado siguiente de la salida. Para dejar la salida sin cambio, es necesario hacer la entrada D igual al valor presente de su salida. [3]

2.1.5.2. Registros de corrimiento

Un registro capaz de desplazar su información binaria en una o en ambas direcciones se llama registro de corrimiento.
El registro de corrimiento más simple es aquel que usa sólo flip-flops como se muestra en la figura 2.6. La salida de un flip-flop dado se conecta a la entrada D del flip-flop que está a su derecha, el reloj es común a todos los flip-flops. La entrada serial determina qué va en la posición de la extrema izquierda durante el corrimiento. La salida serial se toma de la salida del flip-flop de la extrema derecha. A veces es necesario controlar el corrimiento para que ocurra en ciertos pulsos

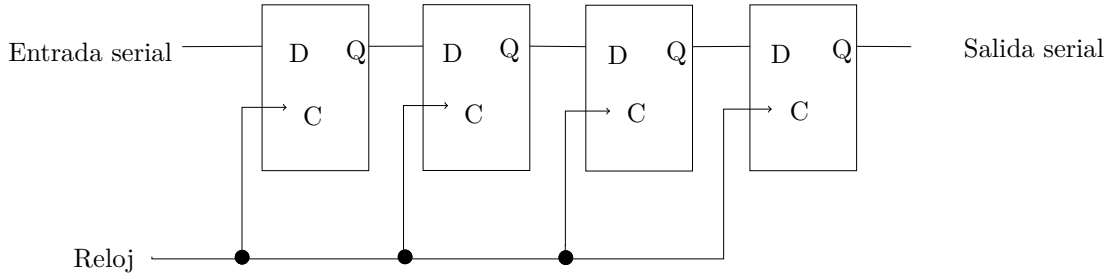

FIGURA 2.6: Registro de desplazamiento de 4 bits

de reloj, pero no en otros. Esto puede hacerse inhibiendo el reloj de la entrada del registro si no queremos el corrimiento.[3]

2.1.5.3. Registro de corrimiento bidireccional con carga paralela

Un registro capaz de tener corrimientos en una sola dirección se llama registro de corrimiento unidireccional. Un registro que puede tener corrimientos en ambas direcciones se llama registro de desplazamiento bidireccional. El registro de corrimiento más general tiene todas las capacidades listadas a continuación.

- Una entrada para pulsos de reloj para sincronizar todas las operaciones.
- Una operación de corrimiento a la derecha y una línea de entrada serial asociada con el corrimiento a la derecha.
- Una operación de corrimiento a la izquierda y una línea de entrada serial asociada con el corrimiento a la izquierda.
- Una operación de carga en paralelo y n líneas de entrada asociadas con la transferencia en paralelo.
- n líneas de salida en paralelo
- Un estado de control que deje la información en el registro sin cambio aun cuando los pulsos de reloj se apliquen continuamente.

Un registro de corrimiento bidireccional de 4 bits con carga paralela se muestra en la figura 2.7. Cada etapa consta de un flip-flop D y un multiplexor de 4x1. Las dos entradas de selección S1 y S0 seleccionan una de las entradas de datos del multiplexor para el flip-flop D. Las líneas de selección controlan el modo de operación del registro de acuerdo con la tabla de función que se muestra en la tabla 2.5.

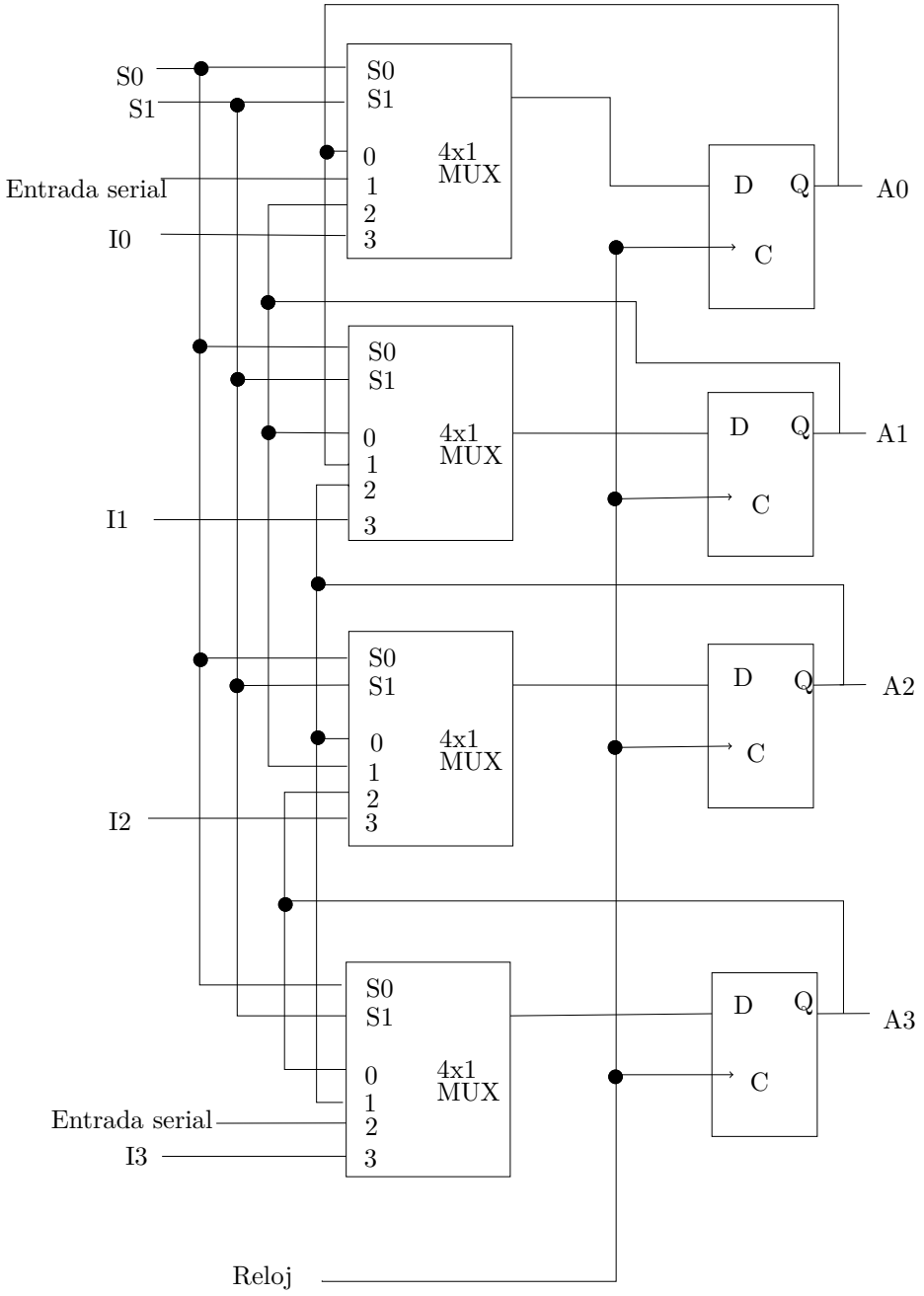

FIGURA 2.7: Registro de desplazamiento bidireccional con carga paralela

Los registros de corrimiento se utilizan comúnmente para la interfaz de sistemas digitales situados remotamennte unos de otros. [3]

Control demodo		
S1	S2	Operación de registro
0	0	Sin cambio
0	1	Desplazamiento a la derecha(abajo)
1	0	Desplazamiento a la izquierda(arriba)
1	1	Carga paralela

TABLA 2.5: Tabla de funciones para los registros de la figura 2.7

2.2. Arquitectura de computadoras

2.2.1. Organización y diseño básico de computadoras

2.2.1.1. La máquina de Von Neuman

El prototipo de una computadora general se introduce en 1952. Esta estructura general de la computadora se muestra en la figura 2.8. [4]

La figura 2.8 muestra que tanto la unidad de control como la ALU contienen posiciones de almacenamiento, llamadas registros, definidos como sigue: [4]

- Registro temporal de memoria "buffer"(MBR): Contiene una palabra que debe ser almacenada en la memoria, o para recibir una palabra procedente de la memoria.
- Registro de dirección de memoria (MAR): Especifica la dirección en memoria de la palabra que va a ser escrita o leída en MBR.
- Registro de Instrucción (IR):Contiene los 8 bits del código de operación de la instrucción que se va a ejecutar.
- Registro temporal de Instrucción (IBR): Empleado para almacenar temporalmente la instrucción contenida en la parte derecha de una palabra en memoria.
- Contador de Programa (PC): Contiene la dirección de la próxima pareja de instrucciones que van a ser captadas de la memoria.
- Acumulador (AC) y Multiplicador cociente (MQ): Se emplean para almacenar temporalmente operandos y resultados de operaciones de la ALU.

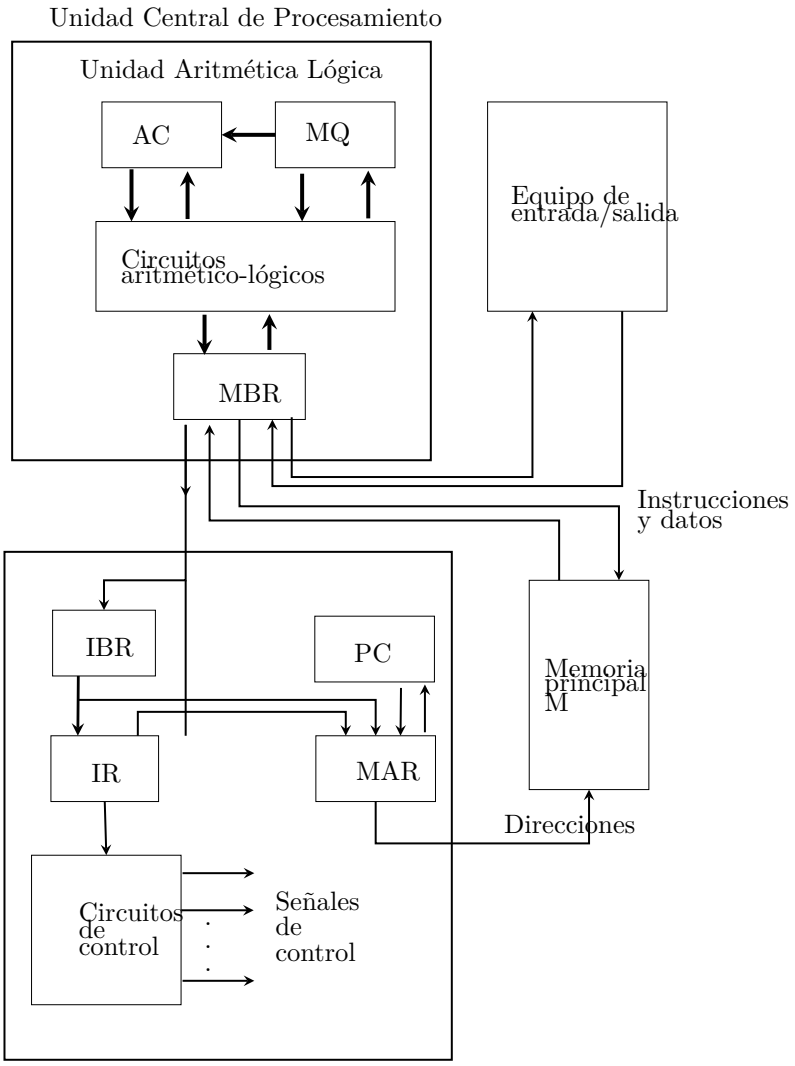

FIGURA 2.8: Estructura general de la máquina de Von Neuman

2.2.1.2. Códigos de instrucción

En esta parte se abordará el funcionamiento de una computadora básica, muy pequeña en comparación de las computadoras comerciales, pero lo suficientemente simple para explicar el proceso de diseño, de una manera más sencilla y entendible. La organización de una computadora está basada en los registros internos de la misma, la estructura de temporización y control, así como las instrucciones que usa.

Un programa es un conjunto de instrucciones que definen las operaciones, operandos y secuencia del procesamiento.

Las instrucciones de computadora son códigos binarios que definen una secuencia de microoperaciones para la computadora, esto se puede observar en el esquema de la figura 2.9. Los códigos de instrucciones y los datos se almacenan en la memoria. Lo que hace la computadora es leer

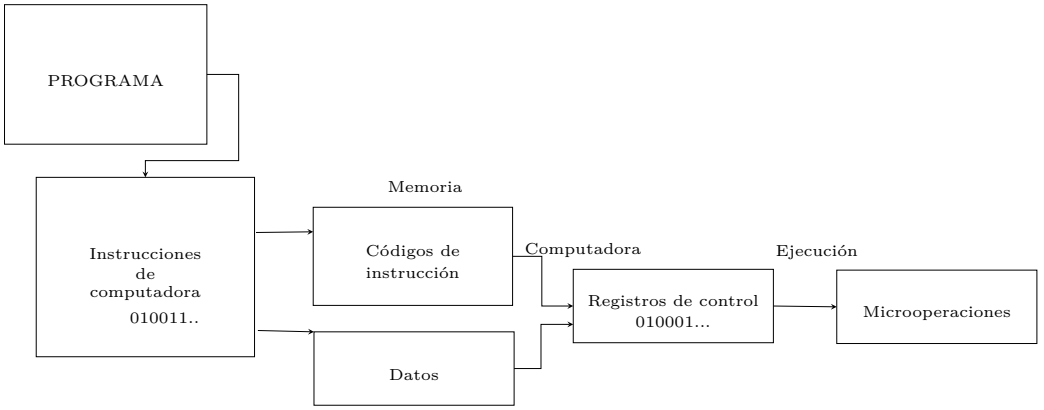

FIGURA 2.9: Partes de un programa

cada una de las instrucciones almacenadas en la memoria y moverlas a un registro de control. El control interpreta el código binario de la instrucción y procede a ejecutarlo mediante una secuencia de microoperaciones. Cada computadora tiene un conjunto de instrucciones único. [3] Un código de instrucción es un grupo de bits que instruye a la computadora sobre cómo ejecutar una operación específica. Por lo general, se divide en partes y cada una tiene una interpretación propia. La parte más básica de un código de instrucción es su parte de operación. El código de operación de una instrucción es un grupo de bits que define operaciones como sumar, restar, multiplicar, desplazar y complementar. El número de bits requerido para el código de operación de una instrucción depende de la cantidad total de operaciones disponibles en la computadora. [3]

2.2.1.3. Organización de un programa almacenado

La manera más simple de organizar una computadora es tener un registro de procesador y un formato de código de instrucción en dos partes. La primera parte especifica la operación que se va a ejecutar y la segunda especifica una dirección. La dirección de memoria le dice al control dónde encontrar un operando en la memoria. Este operando se lee en la memoria y se utiliza como los datos que se van a operar junto con los datos almacenados en el registro del procesador. Las computadoras que tienen un registro de procesador único por lo general lo nombran acumulador y lo etiquetan AC. Si una operación de un código de instrucción no necesita un operando de la memoria, puede usarse el resto de los bits de la instrucción para otros propósitos.[3]

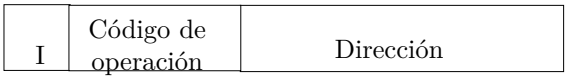

FIGURA 2.10: Direccionamiento con operando inmediato

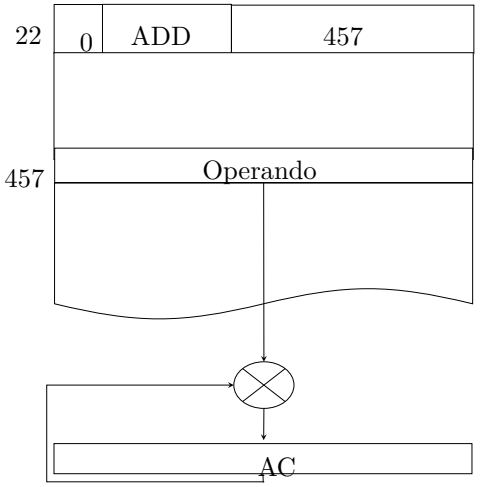

FIGURA 2.11: Direccionamiento directo

2.2.1.4. Direccionamiento

La segunda parte de un código de instrucción se usa para definir la dirección del operando, pero en ocasiones esta no se usa de esta manera, sino que en esta parte se pone el operando directamente, esto se le llama una instrucción con operando inmediato, ver figura 2.10. Otra forma de usar esta sección de código de instrucción es usarla para poner la dirección del operando, esto es llamado direccionamiento directo, ver figura 2.11. Una tercera forma de usarlo es escribiendo la dirección de la sección de memoria en donde se encuentra localizado el operando, a esta forma de direccionamiento se le llama direccionamiento indirecto, ver figura 2.12. El direccionamiento usado nos debe de conducir a la dirección efectiva, la dirección efectiva no es más que la dirección del operando en una instrucción de computadora.

2.2.1.5. Registros de computadora

Las instrucciones de computadora se almacenan en posiciones de memoria consecutivas y se ejecutan de manera secuencial, una a la vez. El control lee una instrucción de una dirección específica de la memoria y la ejecuta. Después continúa leyendo la siguiente instrucción en secuencia y la ejecuta, y así sucesivamente. Este tipo de secuencia de instrucciones necesita un contador para calcular la dirección de la siguiente instrucción después de que se termina la ejecución de la instrucción presente. La computadora necesita los registros del procesador para

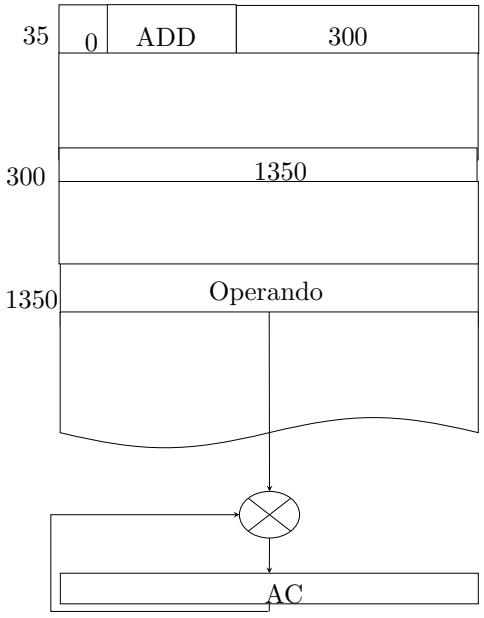

FIGURA 2.12: Direccionamiento indirecto

manipular datos y un registro para contener una dirección de memoria. [3]
Debido a lo anterior, en la computadora básica se cuenta con varios registros; entre ellos, el registro de datos DR, que contiene el operando que se lee en la memoria, registro acumulador AC es un registro de propósito general, registro de instrucción IR, en ella se coloca la instrucción que se lee en la memoria, registro temporal TR, contiene los datos temporales durante el procesamiento, contador de programa PC, contiene la dirección de la siguiente instrucción que se va a leer en la memoria, después de que se ejecute la instrucción presente. Se tienen también otros dos registros, uno de entrada INPR, este recibe un carácter de un dispositivo de entrada, y otro de salida OUTR que contiene un carácter para un dispositivo de salida. [3]
El PC recorre una secuencia de conteo y hace que la computadora lea instrucciones secuenciales almacenadas previamente en la memoria. Las palabras de instrucción se leen y se ejecutan en secuencia, a menos que se encuentre una instrucción de transferencia del programa. Una instrucción de transferencia del programa solicita una transferencia a una instrucción no consecutiva del programa. La parte de dirección de una instrucción de transferencia del programa se transfiere al PC para convertirse en la dirección de la siguiente instrucción. Para leer una instrucción, se toma el contenido del PC como la dirección para la memoria y se inicia un ciclo de lectura de memoria. Después el PC se incrementa en uno, para que contenga la dirección de la siguiente instrucción en secuencia. [3]

2.2.1.6. Bus común del sistema

La computadora básica tiene ocho registros, una unidad de memoria y una unidad de control. Deben proporcionarse trayectorias para transferir información de un registro a otro y entre la memoria y el registro. La cantidad de líneas será excesiva si se hacen conexiones entre la salida de cada registro y las entradas de los otros registros. Un esquema más eficiente para transferir información en un sistema con muchos registros es usar un bus común, en la figura 2.13 se muestra cómo queda la configuración.
Las salidas de siete registros y de la memoria están conectadas al bus común. La salida específica que se selecciona para las líneas del bus en cualquier momento dado, está determinada por el valor binario de las variables de selección S2, S1 Y S0.
Las 16 entradas de AC provienen de un circuito lógico y sumador. Este circuito tiene 3 conjuntos de entradas. Un conjunto de entradas de 16 bits viene de las salidas de AC. Se utiliza para efectuar microoperaciones de registro como el complemento a AC y el corrimiento a AC. Otro conjunto de entrada de 16 bits viene del registro de datos DR. Las entradas de DR y AC se utilizan para microoperaciones aritméticas y lógicas. Un tercer conjunto de entradas de 8 bits viene del registro de entrada INPR. [3]

2.2.1.7. Instrucciones de computadora

La computadora básica tiene 3 formatos de códigos de instrucción. Una instrucción de referencia a memoria utiliza 12 bits para especificar una dirección y 1 bit para especificar el modo de direccionamiento I, 0 cuando es directa y 1 cuando es indirecta, figura 2.14 .
Las instrucciones de referencia a registros se denotan mediante el código 111, con un 0 en el bit más significativo, no se necesita un operando de la memoria por lo tanto los bits restantes se usan para definir la operación o prueba a ejecutar. figura 2.15

Una instrucción de entrada-salida no ocupa una referencia a memoria y se identifica por el código de operación 111 con un 1 en el bit más significativo. Los bits restantes especifican la operación de entrada-salida o la prueba que se va a ejecutar. Figura 2.16
Un formato para las instrucciones es el formato hexadecimal, lo que se hace es agrupar los 16 bits en grupos de cuatro, y de cada uno de los grupos se obtiene un dígito reduciendo así los 16 bits a 4 dígitos.
Si los tres bits del código de operación en las posiciones de la 12 a la 14 no son iguales a 111, entonces la instrucción es del tipo de referencia a memoria y el bit 15 se toma como el modo de direccionamiento.

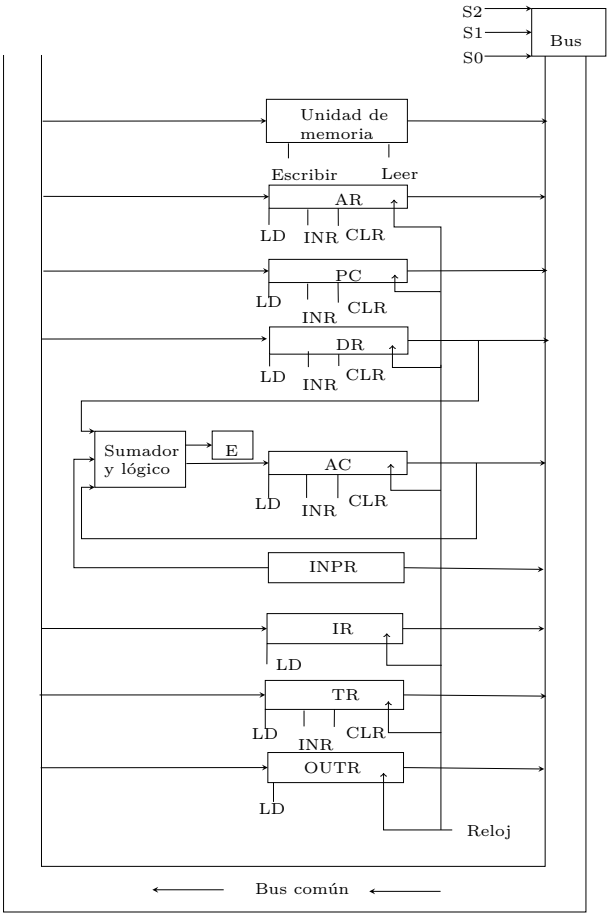

FIGURA 2.13: Registros de la computadora básica conectados a un bus común

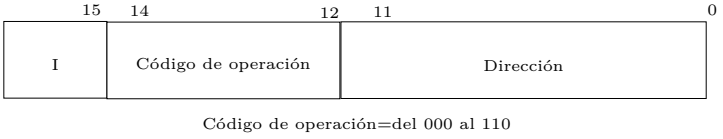

FIGURA 2.14: Formato de instrucción de referencia a memoria

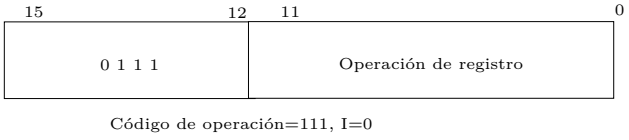

FIGURA 2.15: Formato de instrucción de referencia a registro

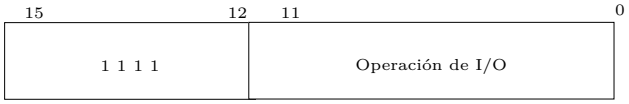

FIGURA 2.16: Formato de instrucción de entrada-salida

2.2.1.8. Versatilidad del conjunto de instrucciones

Todas las computadoras deben de tener una determinada cantidad de instrucciones para que el usuario pueda construir programas de lenguaje de computadora. Estas instrucciones deben de incluir: [3]

- Instrucciones aritméticas, lógicas y de corrimiento, estas ayudan al usuario a procesar los tipos de datos que desea emplear.
- Instrucciones para mover información hacia y desde la memoria y los registros del procesador, esto debido a que la mayor parte de la información binaria se encuentra en la memoria, sin embargo, todos los cálculos que se realizan, se llevan a cabo en los diferentes registros, de ahí la necesidad de la interacción de memoria y registros.
- Instrucciones de control del programa, junto con instrucciones que verifiquen las condiciones de estado, estas se ocupan para cambiar la secuencia de ejecución del programa.
- Instrucciones de entrada y salida, estas se ocupan para la comunicación entre computadora-usuario. Ya que los programas y los datos deben transferirse a la memoria y los resultados de los cálculos se tienen que transferir de vuelta a la interfaz de usuario.

2.2.1.9. Temporización y control

La temporización de los registros de la computadora se controla mediante un generador de reloj maestro. Los pulsos se aplican a cada uno de los flip-flops y registros del sistema, incluyendo la unidad de control, esto sin alterar el estado del registro, a menos que esta esté habilitada por una señal de control. Las señales de control se generan en la unidad de control y proporcionan entradas de control a los multiplexores del bus común, en los registros del procesador, y micro-operaciones para el acumulador. [3]
Hay dos tipos de organizaciones de control, control por cableado y control microprogramado. El primero se hace mediante compuertas flip-flop, decodificadores y otros circuitos digitales, se puede utilizar para producir un modo de operación rápido. Cuando se modifica o se cambia el

diseño se necesita hacer cambios en la instalación de cableado de los diferentes componentes. En el segundo, la información de control esta almacenada en una memoria de control. La memoria de control está programada para iniciar la secuencia de microoperaciones requerida. Cualquier cambio o modificación requeridas, se hace actualizando el programa en la memoria de control. [3]
Un ciclo de lectura o de escritura se empieza justo en el flanco de subida de la señal de temporización.

2.2.1.10. Ciclo de instrucción

Todo programa almacenado en la memoria de la computadora, está constituido por una secuencia de instrucciones. El programa se ejecuta recorriendo un ciclo para cada instrucción. Este ciclo a su vez, está dividido en subciclos o fases, cada uno de los ciclos de instrucción está constituido por las siguientes fases: [3]

1. La búsqueda de una instrucción en la memoria
2. Decodificación de la instrucción
3. Lectura de la dirección efectiva de la memoria si se cuenta con una dirección indirecta
4. Por último, llevar a cabo la ejecución de la instrucción
5. Una vez terminado el ciclo anterior regresar al paso 1 y así sucesivamente hasta ejecutar el programa en su totalidad.

2.2.2. FPGA (Field Programmable Gate Array)

Los CPLDs pueden tener más de 200 pines en el empaquetado, estos pines son frágiles y se rompen fácilmente, para programar la unidad de programación, un socket requiere mantener el chip. Los sockets para empaquetados grandes de QFP son demasiado caros. Para implementar circuitos grandes, es conveniente usar diferentes tipos de chips que tienen una gran capacidad lógica. Un arreglo de compuertas programables(FPGA) es un dispositivo lógico programable que admite la implementación de circuitos lógicos relativamente grandes.[5] Un FPGA es muy diferente a los SPLDs y los CPLDs porque un FPGA no contiene compuertas AND y OR. Un FPGA es un arreglo de compuertas programables, que permiten implementar cualquier circuito digital de alguna aplicación específica. Un FPGA consta de:

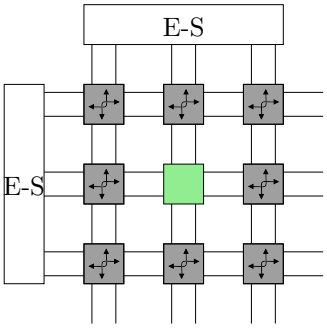

FIGURA 2.17: Arquitectura de una FPGA

- Bloques Lógicos Complejos (CLB).
- Interconexiones encargados de enviar información entre CLB´s y las entradas/salidas.
- Pines que pueden ser configurados como entradas o salidas.
- Memoria RAM.

En la figura 2.17 se muestra la arquitectura de una FPGA. Esta contiene tres tipos principales de recursos:

- Bloques lógicos.
- Bloques de entradas/salidas para conectar a los pines del empaquetado.
- Cables de interconexión y switches.

Los bloques lógicos están acomodados en matrices bidimensionales. El enrutamiento de canales contiene alambres y switches programables que permite que los bloques lógicos se puedan conectar por varios caminos. [5] Con un FPGA se puede implementar un circuito lógico de más de 1 millón de compuertas. Algunos fabricantes de dispositivos FPGA son Altera y Xilinx, estos chips están disponibles en varios empaquetados. Cuando un circuito lógico es implementado en un FPGA, los bloques lógicos son programados para realizar las funciones necesarias y el enrutamiento de canales es programado para realizar las conexiones necesarias entre los bloques. Las celdas de almacenamiento de los LUTs dentro de la FPGA son volátiles, es decir, el valor almacenado en ellos se pierde cuando se apaga la fuente de alimentación, por lo tanto se tiene que programar cada que se le suministra una fuente. [5]

2.2.3. Microcontrolador

Un microcontrolador consiste en un sencillo pero completo computador contenido en el corazón (chip) de un circuito integrado. Un microcontrolador es un circuito integrado de alta escala de integración que incorpora la mayor parte de los elementos que configuran un controlador. [6] Un microcontrolador dispone normalmente de los siguientes componentes:

- Procesador o CPU (Unidad Central de Proceso).
- Memoria RAM para Contener los datos.
- Memoria para el programa tipo ROM/PROM/EPROM.
- Líneas de E/S para comunicarse con el exterior.
- Diversos módulos para el control de periféricos (temporizadores, Puertas Serie y Paralelo, ADC: Convertidores Analógico/Digital, DAC: Convertidores Digital/Analógico, etc.).
- Generador de pulsos de reloj que sincronizan el funcionamiento de todo el sistema.

Los productos que para su regulación incorporan un microcontrolador disponen de las siguientes ventajas [6]:

- Aumento de prestaciones: mayor control sobre un determinado elemento.
- Aumento de la fiabilidad: al reemplazar el microcontrolador por un elevado número de elementos disminuye el riesgo de averías y se precisan menos ajustes.
- Reducción del tamaño en el producto acabado: La integración del microcontrolador en un chip disminuye el volumen.
- Mayor flexibilidad: las características de control están programadas por lo que su modificación sólo necesita cambios en el programa de instrucciones.

2.3. Sistema Operativo

Una computadora no se puede definir solamente por lo que podemos ver y tocar(hardware). Si sólo tuviéramos esta parte, no podemos decir que tenemos una computadora. Una computadora se compone de hardware y software, y precisamente el software es una parte esencial para el funcionamiento de esta, ya que nos permite procesar, almacenar y recuperar información.

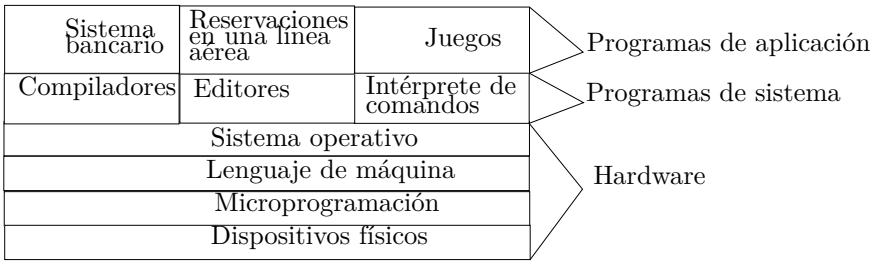

FIGURA 2.18: Sistema de cómputo

El software para computadoras se puede clasificar en dos: programas de sistema, las cuales controlan la operación de la computadora en sí y los programas de aplicación que se encargan de resolver problemas para los usuarios. [7]
El sistema operativo, es un programa fundamental de todos los programas de sistema, el cual controla todos los recursos de la computadora y nos proporciona una base para escribir los programas de aplicación.
Un sistema de computación moderno consta de uno o más procesadores, memoria central, relojes, terminales, discos, interfaces de red y otros dispositivos de entrada-salida, lo que lo vuelve un sistema muy complejo. Si el programador tuviera que preocuparse por el funcionamiento de las unidades de disco entre otros procesos, sería poco probable que pudiera desarrollar programas que permitan resolver problemas para los usuarios. Debido a esto, lo que se hace es almacenar los programas de sistema de forma en que el programador no pueda modificar ni tenga que entender esta parte, esto vuelve más fácil su tarea de hacer programas de aplicación. [7]
Como ya se dijo, la computadora es una máquina muy compleja, debido a esto se hace necesario una interfaz o una forma de representación de esto, y la forma en que se nos presenta es como se muestra en la figura 2.18.

Un microprograma es un software primitivo que controla los dispositivos y proporciona una interfaz con la siguiente capa, este programa se encuentra en la memoria exclusiva para lectura. Es un intérprete, busca las instrucciones de lenguaje de máquina para llevarlas a cabo como una serie de pasos, por ejemplo, para llevar a cabo una instrucción ADD, el microprograma debe determinar la ubicación de los números a sumar, buscarlos, sumarlos y almacenar el resultado en alguna parte.
El lenguaje máquina tiene de 50 a 300 instrucciones, la mayoría de ellas sirve para desplazar datos mediante los registros a través de la máquina, hacer operaciones aritméticas y comparar valores. En esta capa, los dispositivos de entrada-salida se controlan al cargar valores en registros del dispositivo especiales. [3]
Una de las funciones principales del sistema operativo es ocultar la complejidad y proporcionar al programador un conjunto más conveniente de instrucciones con el cual trabajar. El sistema

operativo es esa parte del software que se ejecuta en modo central o modo de supervisión. Está protegido contra la alteración del hardware por parte del usuario. Los compiladores y editores se ejecutan en modo usuario, si un usuario no está conforme con algún compilador, está en su derecho de escribir su propio controlador de interrupciones. [7]

2.3.1. Definición de sistema operativo

Es difícil dar una definición exacta de sistema operativo, debido a que llevan a cabo dos funciones: como máquina extendida y como controlador de recursos.

2.3.1.1. El sistema operativo como una máquina extendida

A nivel de lenguaje de máquina, la arquitectura de una computadora es primitiva y difícil de programar, en especial las entradas y salidas.
Para poder ejecutar una instrucción en la computadora se llevan a cabo muchos procesos complejos en los discos flexibles, las cuales son de difícil entendimiento para el programador y no desea meterse en ellas. Es ahí en donde el sistema operativo actúa, ya que le presenta al programador una interfaz más simple y de fácil manejo, sin tener que lidiar y entender todos y cada uno de los procesos internos.[7]

2.3.1.2. El sistema operativo como controlador de recursos

El sistema operativo juega el papel de proporcionar una asignación ordenada y controlar los procesadores, memorias y dispositivos de entrada-salida para los diferentes programas que compiten por ellos. [7]
Supongamos que se están ejecutando 3 programas en una computadora que desean imprimir su salida de manera simultánea, en la misma impresora, sin el sistema operativo, los primeros renglones serían del programa 1, la segunda del programa 2 y el tercero del tercer programa, etc. El resultado de esto sería desastroso. El sistema operativo puede poner orden, mandando los archivos destinados a la impresora al disco, y desde el disco copiar uno de los programas, hacer la impresión, finalizada la primera impresión, llamar el segundo programa e imprimirlo y asi sucesivamente.[7]

2.3.2. Conceptos de los sistemas operativos

Existe una interfaz entre sistema operativo y los programas de usuario, esta se define como un conjunto de instrucciones ampliadas proporcionadas por el sistema operativo, estas instrucciones

se conocen como "llamadas al sistema", tienen la capacidad de crear, eliminar y utilizar varios objetos del software, controlados por el sistema operativo. [7]

2.3.2.1. Procesos

Un proceso es básicamente un programa en ejecución. Está constituido por el programa ejecutable, sus datos, contador y otros registros, además contiene todo lo necesario para ejecutar el programa.
Si tenemos varios procesos en un mismo sistema operativo, lo que procede es almacenar cada uno de estos con sus respectivas características Y recursos en una tabla llamada tabla de procesos. Cada uno de estos procesos tiene la capacidad de crear uno o más procesos, estos procesos son llamados, procesos "hijo".
Los procesos pueden comunicarse entre si ya sea en la misma computadora o en una computadora distinta al que se está utilizando, esta comunicación entre computadoras se hace mediante el envío de mensajes a través de una red. [7]

2.3.2.2. Archivos

El directorio es un concepto que maneja la mayoría de sistemas operativos, como forma de agrupar los archivos.
El proceso y las jerarquías de archivos, se organizan como árboles. Las jerarquÍas de los procesos no son muy profundas, mientras que las jerarquías de archivos tienen por lo general cuatro, cinco o hasta más niveles. Las jerarquías de los procesos son de corta duración, de unos cuantos minutos, mientras que la jerarquía de directorio puede durar años. La propiedad y la protección son distintas entre procesos y archivos, por lo regular un proceso padre puede controlar y tener acceso a un proceso hijo. [7]
Cada archivo dentro de la jerarquía del directorio se puede determinar mediante el nombre de la ruta de acceso, desde el directorio raíz. Los nombres absolutos de la ruta de acceso constan de la lista de directorios que hay que recorrer desde el directorio raíz hasta llegar al archivo deseado, poniendo diagonales entre los componentes.
Si varias personas tienen acceso a la misma computadora, es muy importante tener un medio de protección para la privacidad de los archivos de cada uno de los usuarios. [7]

2.3.2.3. Llamadas al sistema

Los programas de usuario se comunican con el sistema operativo y le solicitan servicio mediante las llamadas al sistema. La cantidad y tipo de llamadas al sistema varía de un sistema operativo

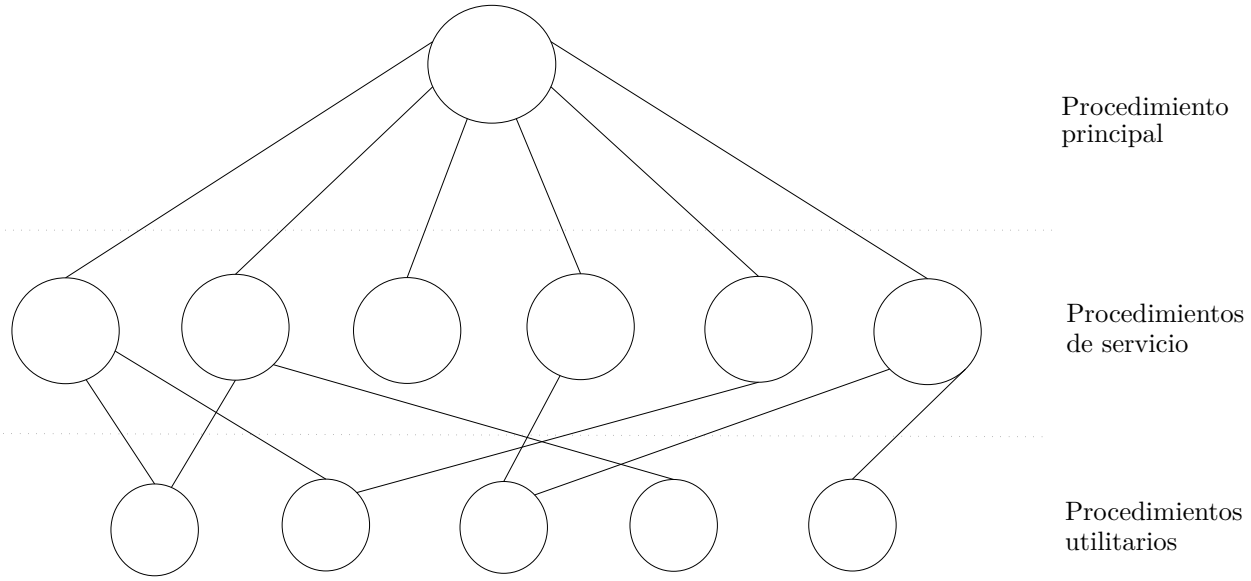

FIGURA 2.19: Un modelo de estructura simple para un sistema monolítico

a otro. Por lo general, hay llamadas al sistema para crear procesos, controlar la memoria, leer y escribir y hacer labores de entrada-salida. [7]

2.3.3. Estructura de los sistemas operativos

En esta sección se examinarán 4 diferentes estructuras, para tener una idea de las opciones que se tienen.

2.3.3.1. Sistemas monolíticos

No existe estructura alguna. El sistema operativo se escribe como una serie de procedimientos, los cuales pueden llamar a los demás en el momento que se requiera. Cada procedimiento del sistema tiene una interfaz bien definida en términos de parámetros y resultados y cada uno de ellos es libre de llamar otro, si este le es útil para cierto cálculo. [7]
En este modelo, para cada llamada al sistema existe un procedimiento de servicio que se encarga de él. [7]. La división de los procedimientos en tres capas se muestra en la figura 2.19

2.3.3.2. Sistemas con capas

Una generalización del punto de vista de la figura 2.19 consiste en organizar el sistema operativo como una jerarquía de capas, cada una construida sobre la inmediata inferior. El primer sistema

5	El operador
4	Programas del usuario
3	Control de entrada/salida
2	Comunicación operador-proceso
1	Administración de la memoria y del disco
0	Asignación del procesador y multiprogramación

Tabla 2.6: Estructura del sistema operativo THE

construido de esta manera fue el sistema THE(Techinische Hogeschool Eindhoven), desarrollado en Holanda por E. W. Dijkstra(1968) y sus estudiantes. El sistema THE era un sistema sencillo de procesamiento por lotes para una computadora holandesa, la Electrológica X8, con 32K de palabras de 27 bits.
El sistema tenía 6 capas, como se muestra en la tabla 2.6. La capa 0 trabaja con la asignación del procesador y alterna entre los procesos cuando ocurren las interrupciones o expiran los cronómetros. Sobre la capa 0, el sistema consta de procesos secuenciales, cada uno de los cuales se podía programar sin tener que preocuparse por el hecho de que varios procesos estuvieran en ejecución en el mismo procesador, es decir, la capa 0 proporcionaba l multiprogramación básica de la CPU.
La capa 1 realizaba la administración de la memoria. Asignaba el espacio de memoria principal para los procesos y un recipiente de palabras de 512K se utilizaba para almacenar partes de los procesos(páginas) para las que no existía lugar en la memoria principal. El software de la capa 1 se encargaba de garantizar que las páginas llegaran a la memoria cuando fueran necesarias. [7]

La capa 2 se encargaba de la comunicación entre cada proceso y la consola del operador. Por encima de esta etapa, cada proceso tiene su propia consola de operador. La capa 3 controla los dispositivos de E/S y guarda en almacenes(buffers) los flujos de información entre ellos. Por encima de la capa 3, cada proceso puede trabajar con dispositivos abstractos de E/S con propiedades adecuadas, en vez de dispositivos reales con muchas peculiaridades. La capa 4 es donde estaban los programas del usuario. Estos no tenían que preocuparse por el proceso, memoria, consola o control de E/S. El proceso operador del sistema se localiza en la capa 5.

2.3.3.3. Máquinas virtuales

VM/370 se basó en una observación: un sistema de tiempo compartido proporciona 1.- multiprogramación y 2.- una máquina extendida con una interfaz más conveniente que el mero hardware. La esencia de VM/370 es la separación total entre estas dos funciones.

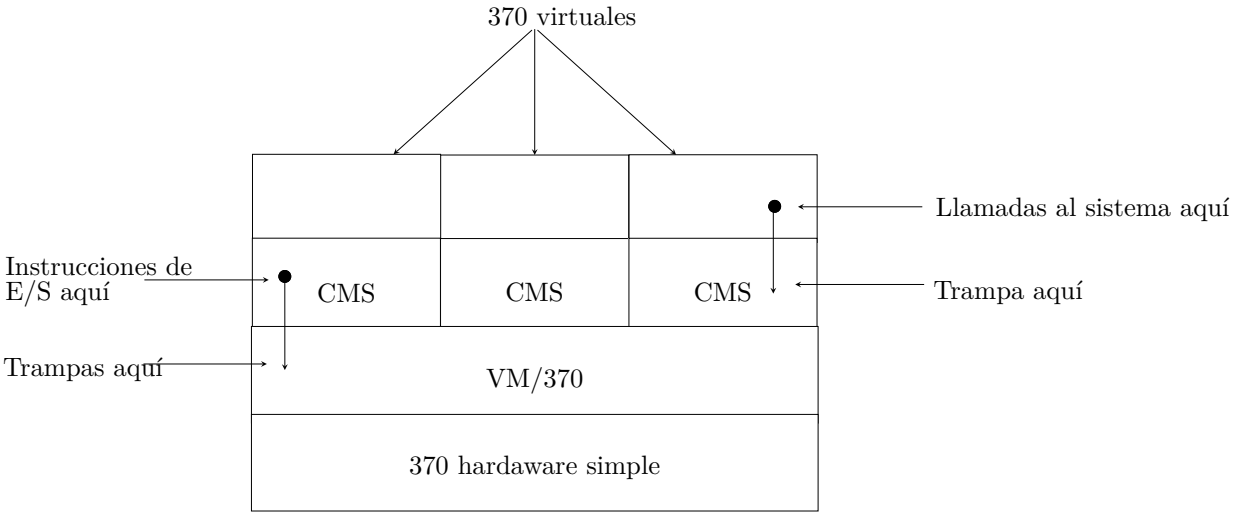

FIGURA 2.20: La estructura de VM/370 con CMS

El corazón del sistema, llamado monitor de la máquina virtual, se ejecuta en el hardware simple y realiza la multiprogramación, proporcionando no una, sino varias máquinas virtuales a la siguiente capa superior, como lo muestra la figura 2.20. [7]

2.3.3.4. Modelo cliente-servidor

Una tendencia de los sistemas operativos modernos es la de explotar más la idea de mover el código a capas superiores y eliminar la mayor parte posible del sistema operativo para mantener un núcleo mínimo. El punto de vista usual es el de implantar la mayoría de las funciones del sistema operativo en los procesos del usuario. Para solicitar un servicio, como la lectura de un bloque de cierto archivo, un proceso del usuario(proceso cliente) envía la solicitud a un proceso servidor, que realiza el trabajo y regresa la respuesta.
En este modelo, que se muestra en la figura 2.21, lo único que hace el núcleo es controlar la comunicación entre los clientes y los servidores. Al separar el sistema operativo en partes, cada una de ellas controla una faceta del sistema, como el servicio a archivos, servicio a terminales o servicio a la memoria, cada parte es pequeña y controlable. Puesto que todos los servidores se ejecutan como procesos en modo usuario y no en modo núcleo, no tienen acceso directo al hardware. Si hay un error en el servidor de archivos, éste puede fallar, pero esto no afectará a toda la máquina. [7]
Otra de las ventajas del modelo cliente-servidor es su capacidad de adaptación para su uso en los sistemas distribuidos. [7]

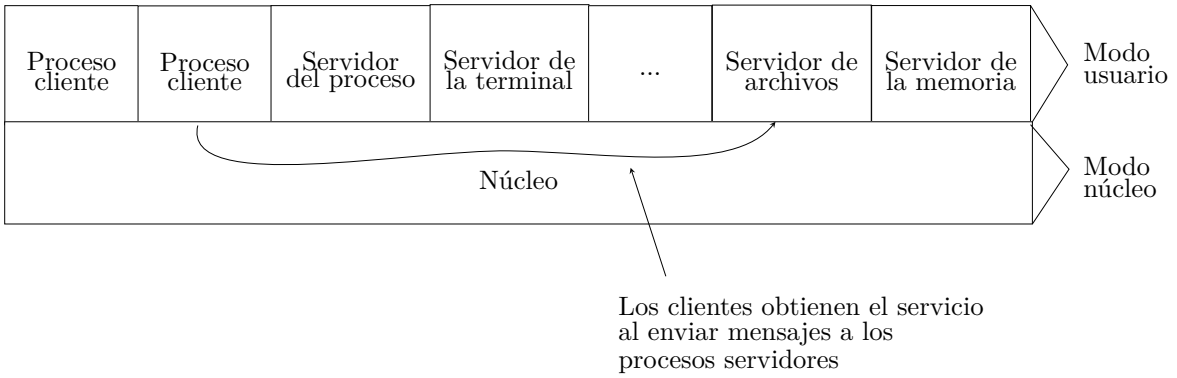

FIGURA 2.21: El modelo cliente-servidor

2.3.4. Sistemas operativos de tiempo real

Para empezar a hablar de sistemas operativos de tiempo real es conveniente definir "Sistema de Tiempo Real". El Oxford Dictionary of Computing (Diccionario Oxford de computación) "Cualquier sistema en el que el tiempo en el que se produce la salida es significativo. Esto generalmente es porque la entrada corresponde a algún movimiento en el mundo físico, y la salida está relacionada con dicho movimiento. El intervalo entre el tiempo de entrada y el de salida debe ser lo suficientemente pequeño para una temporalidad aceptable." [8]

Además Young(1982) define un sistema de tiempo real como:
"cualquier actividad o sistema de proceso de información que tiene que responder a un estímulo de entrada generado externamente en un periodo finito y especificado." [8]

Como ya se mencionó en secciones anteriores, uno de los objetivos de un sistema operativo es el de administrar los recursos de la computadora ante las diversas actividades a realizar. En un sistema operativo de tiempo real este proceso es más complicado, debido a que algunas actividades son críticas en términos del tiempo y unas tienen mayor prioridad que otras. Debido a lo anterior se debe de tener un medio para asignar prioridades a las tareas para que el planificador las pueda ejecutar dependiendo de esto. Un sistema operativo de tiempo real es muy útil en situaciones donde el tiempo es crítico y que algún retraso en las ejecuciones de tareas puede causar un riesgo de seguridad. [9]

Para que un sistema operativo se considere un sistema operativo de tiempo real debe de cumplir con los siguientes requisitos:

- Determinismo: Se dice que un sistema operativo es de tiempo real cuando se puede calcular el tiempo que puede tardar una llamada al sistema. [10] Cuando existen varios procesos

compitiendo por recursos, incluido el procesador, ningún sistema será por completo determinista. [11]

- Sensibilidad: Hace referencia a cuanto tiempo consume un sistema operativo en reconocer una interrupción, es el tiempo preciso para dar servicio a la interrupción después de haberla reconocido, esto depende del tiempo necesario para iniciar la gestión de la interrupción y empezar la ejecución de la rutina de tratamiento (ISR Interrupt Service Routine). Si la ejecución de la ISR requiere de un cambio de proceso ese tiempo será mayor. [11]
- Control de usuario: este es mayor en un sistema operativo en tiempo real que en uno de tiempo compartido. En un sistema de tiempo compartido el usuario no puede asignar prioridades a sus procesos, decidir sobre el algoritmo de planificación,etc.
- Fiabilidad: Un sistema de tiempo real controla sucesos que están en el entorno de tiempo y en su propia escala de tiempos, las pérdidas o degradaciones en el sistema que los controla puede tener consecuencias catastróficas. [11]
- Tolerancia a fallos: Un sistema operativo de tiempo real debe ser diseñado para responder incluso ante varios tipos de fallas. Un sistema operativo de tiempo real intentará corregir el problema o minimizar sus efectos antes de continuar con la ejecución.

Si una tarea debe ser completada en un momento dado, se dice que esa tarea debe ser ejecutada en tiempo real. Un sistema operativo de tiempo real gestiona los tiempos en un microcontrolador o microprocesador. Un sistema operativo de tiempo real tiene las siguientes características:

- Es Multi-tarea.
- El planificador soporta tareas con prioridades.
- Sincroniza el acceso de las tareas a recursos que sean compartidos.
- Soporta comunicación entre tareas.
- Tiempos predecibles.
- Gestión de interrupciones

CAPÍTULO 3

Descripción de herramientas

3.1. FreeRTOS

Los sistemas en tiempo real, están presentes en nuestra vida cotidiana, desde la cocina hasta las industrias productivas más desarrolladas y en empresas dedicadas a los servicios. A veces se nos hace difícil identificar qué aparatos electrónicos tienen un sistema operativo de tiempo real en su implementación; si hablamos de estos equipos, tenemos desde una lavadora, el horno de microondas, el control de vuelo un avión, etc. Los sistemas de tiempo real se han vuelto muy importantes en nuestra vida diaria, gracias a ellos tenemos acceso a la energía eléctrica, tenemos medios de transporte más eficientes actualmente, los cuales nos permiten tener una vida más cómoda. [10]

En este trabajo se abordará uno en particular, FreeRTOS, este software tiene muchas ventajas; para empezar es libre, ahorrando así una inversión en la adquisición de software, su código es libre para los programadores. Facilita el interfaz con el usuario, mediante las funciones incluidas, es decir que el programador lo único que tiene que hacer es llamar estas funciones, sin tener que internarse en una programación de alta abstracción.

FreeRTOS puede ser construido en aproximadamente 20 compiladores diferentes, y puede ejecutarse en más de 30 arquitecturas de procesadores diferentes. FreeRTOS se puede concebir como una biblioteca multitarea.

FreeRTOS se configura mediante el archivo de encabezado FreeRTOSConfig.h, esta se utiliza para adaptar FreeRTOS para su uso en alguna aplicación específica. El programador no tiene la necesidad de crear un archivo FreeRTOSConfig.h ya que cada aplicación contiene este archivo, para la configuración. [12]

Existen dos tipos de sistemas de tiempo real:

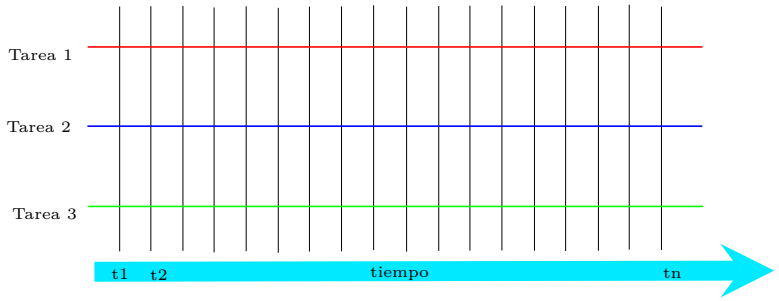

FIGURA 3.1: Ejecución de tareas en freeRTOS

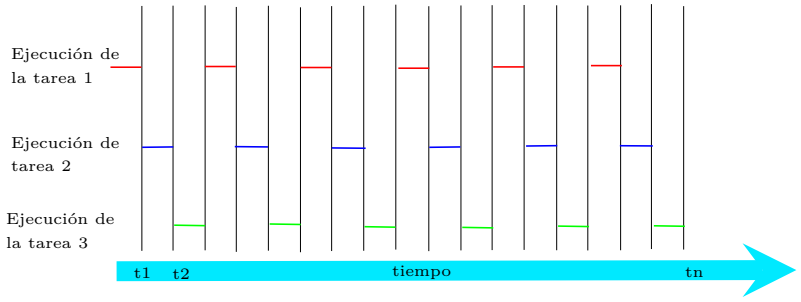

FIGURA 3.2: Verdadera ejecución de las tareas en freeRTOS

- Soft real-time: se tiene este tipo de sistema cuando se establece un tiempo determinado para que el sistema nos dé una respuesta ante algún evento, pero al no cumplir este tiempo establecido no hace que nuestro sistema sea inservible.[8]
- Hard real-time: cuando se establece un tiempo de respuesta y el sistema no responde en ese determinado tiempo, nuestro sistema en cuestión ya no funciona correctamente. [8]

3.1.1. Planificador de freeRTOS

La función del planificador es la de asignarle un determinado tiempo a cada una de las tareas a ejecutarse, freeRTOS cuenta con un planificador que aparentemente ejecuta varias tareas a la vez, esto se observa en la figura 3.1 pero en realidad, lo que sucede es, se ejecuta la tarea en un pequeño lapso y solo una parte de la tarea, en el siguiente tiempo una tarea 2, y en el próximo aumento de tiempo se ejecuta la tarea 1, y así sucesivamente hasta terminar de procesar todas las tareas, este proceso se muestra en la figura 3.2

3.1.2. Kernel de tiempo real

Existen técnicas para escribir un buen software embebido sin el uso de un kernel, cuando el sistema a desarrollar es simple estas técnicas proporcionan soluciones adecuadas. Pero cuando

el sistema es más complejo, es preferible usar un kernel. La priorización de tareas puede ayudar a asegurar que una aplicación cumpla con los plazos de procesamiento de datos, pero un núcleo trae más beneficios [12]. Se presentan algunas de ellas:

- El kernel es el responsable de la sincronización y ejecución de la aplicación. Esto permite que la estructura del código de la aplicación sea más simple, además reduce el tamaño del código general.
- La definición de los detalles de sincronización a distancia produce menos interdependencia entre los módulos de sincronización y permite que el software evolucione de una manera controlada y predecible. Además, el kernel es el responsable de la temporización, por lo que el rendimiento de la aplicación es menos susceptible a los cambios de software.
- Las tareas son módulos independientes, cada uno de ellos tiene un propósito bien definido.
- Las tareas deben de tener interfaces bien definidos, lo que facilita el desarrollo de los equipos.
- Si las tareas son módulos independientes con interfaces limpias, se pueden probar de manera aislada.
- Mayor modularidad y menos interdependencias resultan en un código reutilizable.
- El uso del kernel permite que el software sea completamente controlado por eventos, por lo que no se pierde tiempo en el procesamiento para eventos que no se han producido. El código se ejecuta solo cuando hay algo que debe hacerse.
- Se desactiva la tarea cuando se inicia el planificador.

3.1.3. Características de FreeRTOS

FreeRTOS tiene las siguientes características estándar[12]

- Operaciones preventivas o cooperativas.
- La asignación de prioridades de las tareas es muy flexible.
- El mecanismo de notificación de tareas es flexible, rápido y ligero.
- Queues (colas de datos).
- Semáforos binarios.

- Mútex
- Temporizadores de software
- Grupos de eventos
- Comprobación de desbordamiento de stack
- Recopilación de estadísticas de la ejecución de tareas.
- Soporte y licencia comercial, opcionales
- Modelo completo de anidamiento de interrupciones (para algunas arquitecturas)
- Stack de interrupción administrada por software cuando sea necesario (esto puede ahorrar RAM).

A continuación, se presentan las características y algunas funciones básicas que se requieren para la programación, utilizando este software:

3.1.4. Funciones de las tareas

Las tareas son implementadas como funciones de C. Lo único que las hace especiales es su prototipo, el cual debe de regresar vacío (void) y tomar como parámetro un apuntador vacío (void). A continuación, se muestra el prototipo. [13]

```
void ATaskFunction( void *pvParameters );
```

La mayoría de las tareas se implementan dentro de un ciclo infinito.

3.1.5. Estados de las tareas

Una aplicación consiste en muchas tareas. Si el microcontrolador que está ejecutando el programa tiene un solo núcleo, se puede ejecutar una tarea a la vez, esto significa que una tarea existe en uno o dos estados, en ejecución y No ejecución. Cuando una tarea está en estado ejecución, el procesador está ejecutando esta tarea actualmente. Cuando una tarea está en estado No ejecución, la tarea esta inactiva ver figura 3.3 .

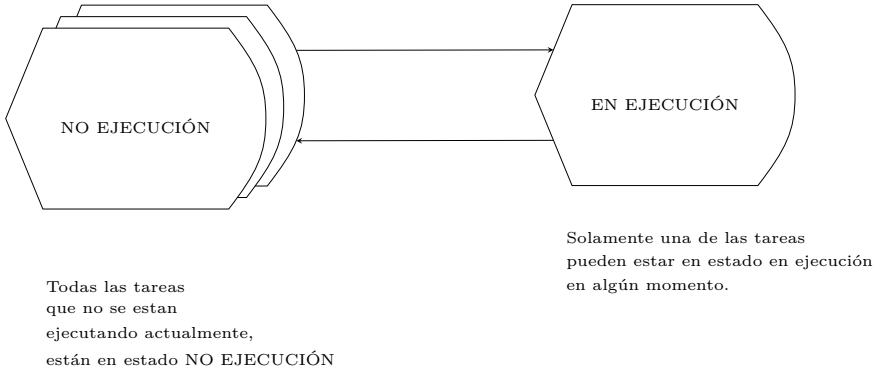

FIGURA 3.3: Estados de una tarea

3.1.6. Crear tareas

La función xTaskCreate() incluye la función xTaskCreateStatic(), la cual asigna la memoria necesaria para crear una tarea en forma estática al momento de compilar. Las tareas se crean utilizando la función de FreeRTOS xTaskCreate (). Esta quizás sea la más compleja de todas las funciones, pero se debe de entender esta función para poder dominar la creación de tareas. [13]

Las tareas se crean utilizando la función xTaskCreate() de FreeRTOS, en la siguiente tabla se muestran los parámetros se esta función.

Parámetro	**Descripción**
pvTaskCode	Las tareas son funciones de C que nunca terminan, y normalmente son implementadas en ciclos infinitos. El parámetro de pvTaskCode, simplemente apunta a la función que implementa la tarea.
pcName	Es un nombre descriptivo de la tarea. Esta no es usada por FreeRTOS en ningún momento. Es solamente para ayudar en la depuración.
	Continúa en la siguiente página

Tabla 3.1 – continuación de la página anterior

Parámetro	Descripción
usStackDepth	Cada una de las tareas tiene un estado único que es asignado por el kernel cuando se crea la tarea. El valor de usStackDepth indica al kernel el tamaño del stack. No es fácil determinar el espacio que ocupa una tarea dentro del stack. Es posible calcular, pero la mayoría de los usuarios simplemente asignan un valor que creen razonable. A continuación, en base a las características proporcionadas por FreeRTOS, se debe de asegurar que el espacio asignado es el adecuado, y que la memoria RAM no se esta desperdiciando.
pvParameters	Las funciones de tareas aceptan un parámetro de tipo apuntador void. El valor de pvParameters es el valor que será pasado a la tarea.
uxPriority	Define la prioridad con la cual se ejecuta la tarea. Las prioridades pueden ser asignadas desde 0, que es la prioridad mínima, para la más alta prioridad configMAX_PRIORITIES_1. configMAX_PRIORITIES es una constante definida por el usuario. No existen límites para la asignación de prioridades (aparte de los límites de tipos de datos y la memoria RAM disponible en el microcontrolador), pero es recomendable no usar más allá de los realmente necesarios para evitar el desperdicio de memoria RAM.
pxCreatedTask	PxCreatedTask se puede utilizar para poner un identificador en la tarea que se está creando. Este identificador se puede utilizar para hacer referencia a la tarea dentro de las llamadas, por ejemplo, pueden modificar la prioridad de una tarea o eliminarla. Si su aplicación no tiene un identificador para las tareas entonces pxCreatedTask se pone NULL

Continúa en la siguiente página

Tabla 3.1 – continuación de la página anterior

Parámetro	Descripción
Returned Value	Se tienen dos posibles valores de retorno: ▪ pdTRUE indica que la tarea se creo correctamente. ▪ errCOULD_NOT_ALLOCATE_REQUIRED _MEMORY: Esto indica que la tarea no se pudo crear correctamente debido a la falta de memoria RAM disponible de FreeRTOS para asignar a la estructura de datos de la tarea y del stack.

3.1.7. Estado bloqueado

Una tarea que está esperando a que ocurra un evento es una tarea que está en estado "Bloqueado" el cual es un sub-estado del estado No ejecución. [13] Las tareas pueden entrar al estado bloqueado por dos razones:

- Evento temporal: este evento ocurre ya sea porque un periodo de retardo expira o el tiempo absoluto se ha alcanzado. Por ejemplo, una tarea puede entrar al estado "Bloqueado" para esperar que pasen 10 milisegundos.
- Sincronización de eventos: Cuando los eventos se originan de otra tarea o de una interrupción. Por ejemplo, una tarea puede entrar al estado "Bloqueado" para esperar a que lleguen datos a una cola de datos

Es posible bloquear una tarea en un evento de sincronización mediante un tiempo de espera. Por ejemplo, una tarea puede esperar un tiempo máximo de 10 milisegundos para que los datos lleguen a la cola, pasado este tiempo, si no llega ningún dato la tarea se bloquea. [13]

3.1.8. Estado suspendido

El estado Suspendido es también un sub-estado de No ejecución. Las tareas entran al estado Suspendido si el planificador (scheduler) no está disponible. La única manera de entrar al estado suspendido es mediante la función vTaskSuspend(), mientras que la forma de salir de este estado es llamando la función vTaskSuspend (), o la función xTaskResumeFromISR (). Cabe señalar que la mayoría de las aplicaciones no utilizan el estado "suspendido". [13]

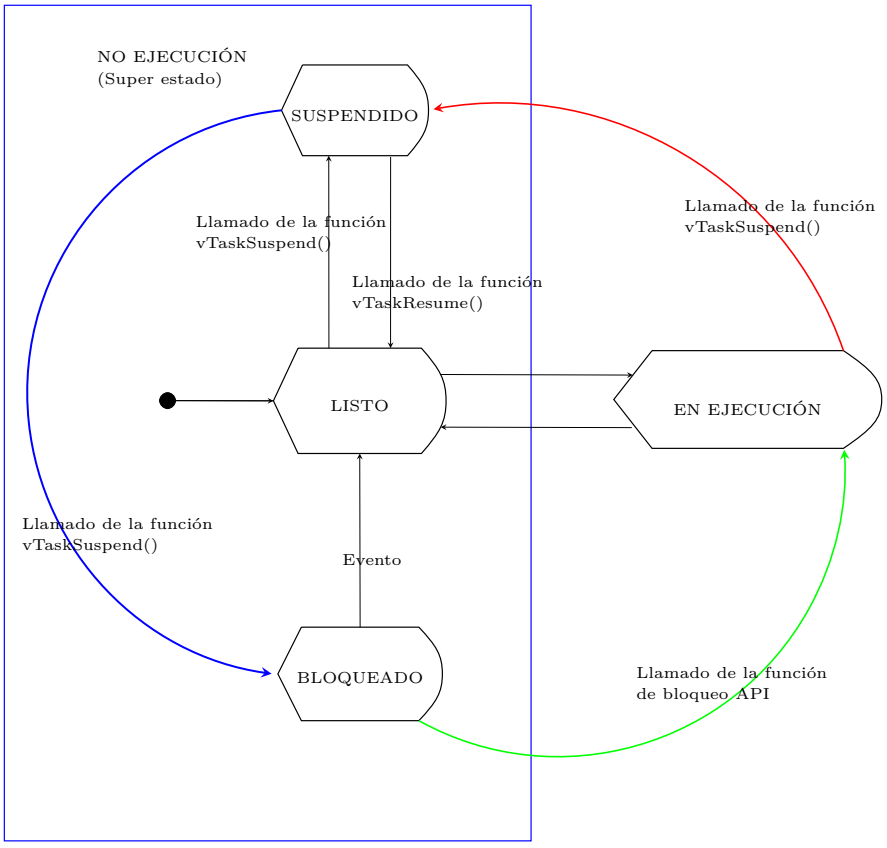

FIGURA 3.4: Diagrama completo del estado de transición de una tarea

3.1.9. Estado: listo

Cuando una tarea no está en estado "Ejecución", pero tampoco en estado "Bloqueado" o "Suspendido", se dice que está en estado Listo. Ya están "Listos" para ejecutarse, pero no están en ejecución. [13]

3.1.10. Diagrama completo del estado de transición de una tarea

La figura 3.4 muestra el diagrama de estado de una tarea, incluyendo los sub-estados del estado No ejecución.

3.1.11. Delay

La función vTaskDelay (), solo está disponible cuando INCLUDE_vTaskDelay se pone en 1, esta coloca la tarea en estado Bloqueado durante un determinado tiempo, es decir que la tarea no consume recursos de procesamiento mientras se encuentra en este estado. Por lo tanto, solo

consume los recursos de procesamiento necesarios. A continuación se muestra el prototipo de esta función. [13]

```
#include "FreeRTOS.h"
#include "task.h"
void vTaskDelay(TickType_t xTicksToDelay);
```

En la siguiente tabla 3.2 se muestran los parámetros necesarios para hacer uso de la función:

Parámetro	**Descripción**
xTicksToDelay	La tarea permanecerá en estado Bloqueado hasta que pasen los ciclos de reloj, llamados dentro de la tarea, después estará en estado Listo. La macro pdMS_TO_TICKS() convierte un tiempo en milisegundos a pulsos de reloj. Por ejemplo, al llamar vTaskDelay (pdMS_TO_TICKS (100)) dará como resultado una tarea Bloqueada durante 100 milisegundos.

Tabla 3.2: Parámetros de un Delay

El tipo de sistema utilizado para este trabajo es un hard real-time, ya que el sistema tiene que recibir varias frecuencias a la vez y no se permite que alguna de ellas se deje de aplicar en algún momento.

3.2. Qsys

Qsys es una herramienta que viene incluida en el software de Quartus II. Con esta herramienta podemos implementar nuestros diseños de hardware a nivel de sistema con un nivel alto de abstracción, y ésta automatizará las tareas de diseño e integración de los componentes HDL (núcleos IP, verificación IP, etc.). Si se quiere reutilizar el diseño, Qsys nos da facilidades mediante el empaquetamiento de los componentes personalizados con Altera (R). Qsys crea automáticamente una interconexión lógica de alto nivel, eliminando una tarea propensa a errores y ahorrando tiempo de escritura de HDL para especificar una conexión a nivel sistema.

Qsys se vuelve más potente cuando uno mismo diseña componentes personalizados usando interfaces estándar. [14]
Además, Qsys proporciona las siguientes ventajas:

- Automatiza los procesos de personalización e integración de componentes.
- Soporta direcciones de 64 bits.
- Soporta diseño modular del sistema (diseñar tareas pequeñas o dividir un todo por partes para facilitar el proceso de mantenimiento en caso de alguna falla del sistema).
- Soporta la visualización de los sistemas.
- Soporta la optimización y canalización de interconexión dentro del sistema.
- Completamente integrado con el software Quartus II.

3.2.1. Soporte de interfaces de componentes

Los componentes pueden tener cualquier número de interfaces en cualquier combinación. Cada interfaz representa un conjunto de señales que se pueden conectar dentro de un sistema Qsys o exportar fuera de un sistema Qsys.[14]
Qsys incluye los siguientes tipos de interfaces:

- Mapeo de memoria: implementa una estructura de interconexiones de arreglo matricial que incluye el maestro y el esclavo. Las interconexiones consisten en un enrutamiento y lógica síncrona dentro de la FPGA y su implementación se basa en una arquitectura en el chip.
- Transmisión: Conecta fuentes y canales de Avalon Streaming (Avalon_ST) que transmiten datos unidireccionales, así como componentes IP de banda ancha y baja latencia. Crea rutas de datos para el tráfico unidireccional y multicanal, así como paquetes de datos y datos DSP (Procesador de Señal Digital). La interconexión Avalon_ST es flexible y puede implementar interfaces en el chip para núcleos de telecomunicaciones y comunicaciones de datos estándar de la industria, como Ethernet, Interlaken y video.
- Interruptores: Conecta interruptores remitentes con interruptores receptores. Qsys es compatible con solicitudes de interrupción (IRQs) de un solo bit. Cuando varios remitentes mandan una IRQ al mismo tiempo, el receptor debe determinar cual tiene la más alta prioridad y responder de manera adecuada.

- Relojes: conecta interfaces de reloj de salida con interfaces de reloj de entrada. El reloj de salida puede desplegarse sin el uso de un puente. Se requiere un puente solamente cuando la fuente de reloj es externa.
- Resets: Conecta las fuentes de reset con las interfaces de reset de entrada. Si su sistema requiere un reset sincronizado de flanco positivo o flanco negativo, Qsys inserta un controlador de reset para crear la señal apropiada. Si diseña un sistema con varias entradas de reset, el reset del controlador OR pone todas las entradas de reset y genera una sola salida.
- Conductos: conecta las interfaces de conducto de punto a punto, o representa señales que son exportadas desde el sistema de Qsys. Qsys utiliza conductos para las señales de entrada/salida de los componentes que no son parte de alguna interfaz estándar soportada. Se puede conectar dos conductos dentro del sistema Qsys como una conexión punto a punto, o las interfaces de conducto se pueden exportar y se puede llevar al nivel superior como E / S del nivel superior del sistema. Se pueden usar conductos para conectar dispositivos externos, por ejemplo, una memoria externa DDR SDRAM, y la lógica de la FPGA se define fuera del sistema Qsys.

La implementación del procesador se realiza utilizando la herramienta Qsys de Quartus II, como primer paso se crea una nueva plantilla Deo Nano SystemBuilder 3.5.

En Quartus II se abre el archivo con extensión .V creado con la plantilla Deo Nano SystemBuilder, se muestra el código resultante en el apéndice 8.

3.3. FPGA Cyclone IV EP4CE22F17C6N

La familia de dispositivos FPGA Cyclone IV de Altera, proporciona las FPGA´s de menor costo en el mercado y de menor potencia. La serie Cyclone IV ofrece dos dispositivos:

- Cyclone IV E: potencia baja y bajo costo, con alta funcionalidad.
- Cyclone IV GX: FPGA de menor potencia y menor costo con 3.125 Gbps de transceptores.

En este trabajo se utiliza una FPGA Cyclone IV E mostrado en la figura 3.6, dicho dispositivo tiene las siguientes características:

- 153 pines de entrada/salida.

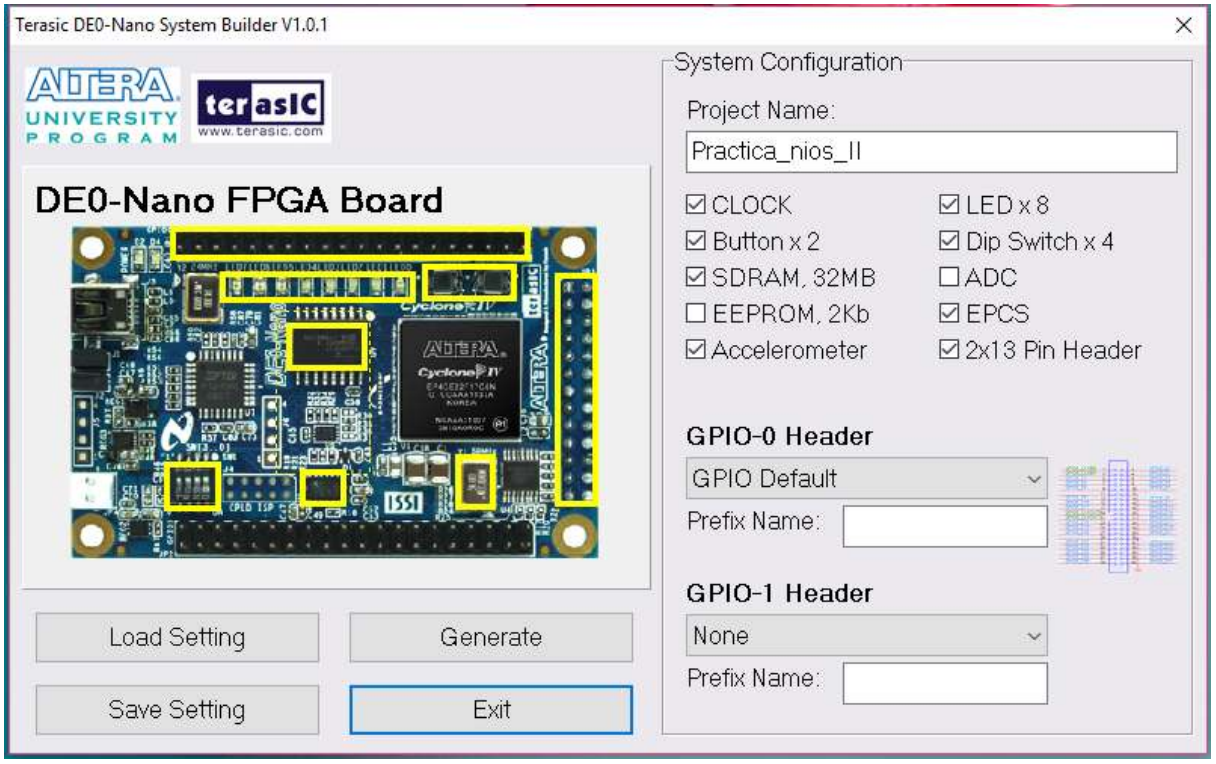

FIGURA 3.5: Ventana emergente para agregar los componentes a utilizar

- Circuito USB-Blaster incorporado para la programación.
- 2 conectores de 40 pines(GPIOs) cada uno, contiene 72 pines de entrada/salida, 4 pines de tierra, 5 pines de 5V, 2 pines de 3.3V.
- 32MB de memoria SDRAM.
- 2Kb de memoria EEPROM I2C.
- 8 Leds de color verde.
- 2 botones antirrebote.
- 4 switches.
- ADI ADXL345, acelerómetro de 3 ejes, de alta resolución(13 bits).
- NS ADC128S022, convertidor analógico-digital de 12 bits de 8 canales.
- De 50 a 200 Kmps.
- Reloj de 50MHz.
- Puerto USB tipo mini-AB (5V).

FIGURA 3.6: FPGA Cyclone IV EP4CE22F17C6N

- 2 pines de alimentación externos(3.6-5.7V).

3.4. CCS Code Composer Studio

Code Composer Studio es un entorno de desarrollo integrado (IDE) que admite la variedad de microcontroladores y procesadores integrados de Texas Instruments. Code Composer Studio cuenta con:

- Un conjunto de herramientas utilizadas para desarrollar y depurar aplicaciones integradas.
- Incluye un compilador de C / C ++.
- Editor de código fuente.
- Entorno de compilación de proyectos.
- Depurador, y muchas otras funciones.

En la figura 3.7 se muestra el entorno de trabajo de Code Composer Studio.

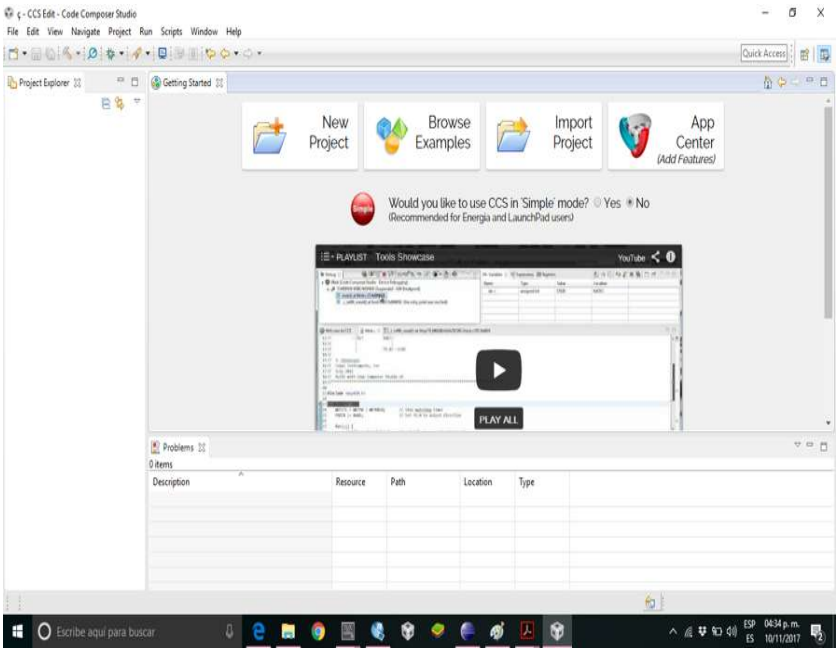

FIGURA 3.7: Entorno de trabajo de Code Composer Studio

3.5. MSP430F5529

El microcontrolador MSP430F5529 es un dispositivo de la familia MSP430, fabricado por Texas Instruments. Ver figura 3.8. Este microcontrolador cuenta con:

- Un puerto USB.
- Tiene 4 temporizadores de 16 bits.
- Un convertidor analógico-digital de alto rendimiento de 12 bits.
- Dos interfaces de comunicación serial universal (USCI).
- Módulo de reloj de tiempo real con capacidad de alarma.
- Cuenta con 63 pines de E/S.

La arquitectura de este microcontrolador se observa en la figura 3.9

3.6. Procesador NIOS II

La arquitectura de NIOS II es una arquitectura de conjunto de instrucciones (ISA, Instruction Set Architecture), ver figura 3.10. A su vez la arquitectura de conjunto de instrucciones necesita

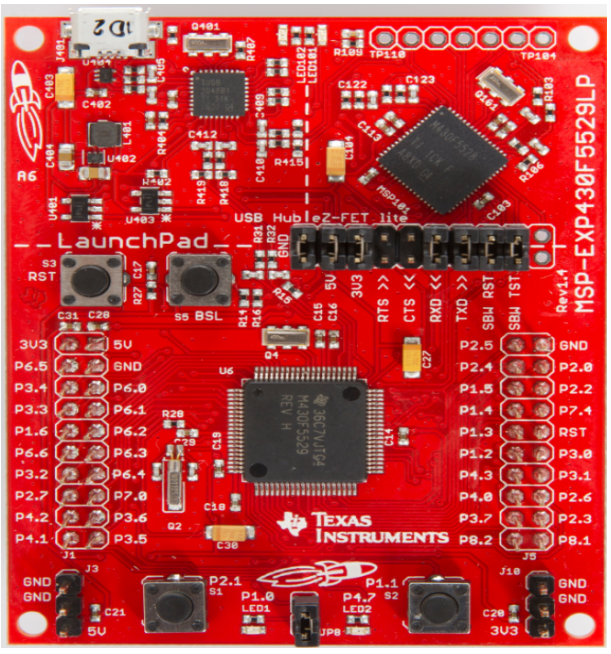

FIGURA 3.8: Microcontrolador MSP430F5529

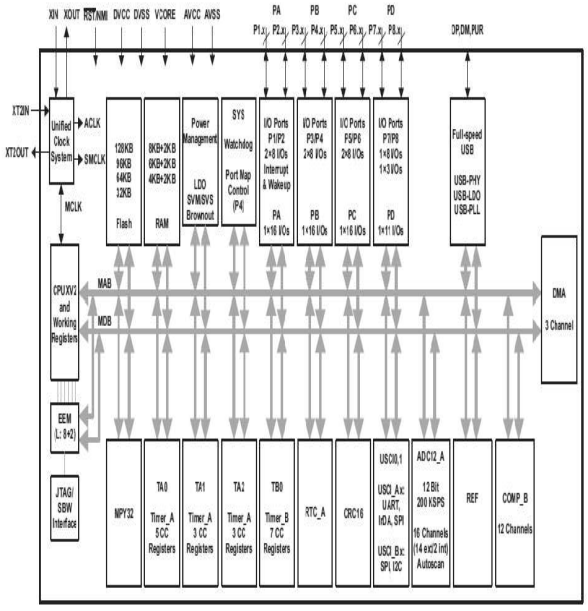

FIGURA 3.9: Arquitectura del microcontrolador MSP430F5529

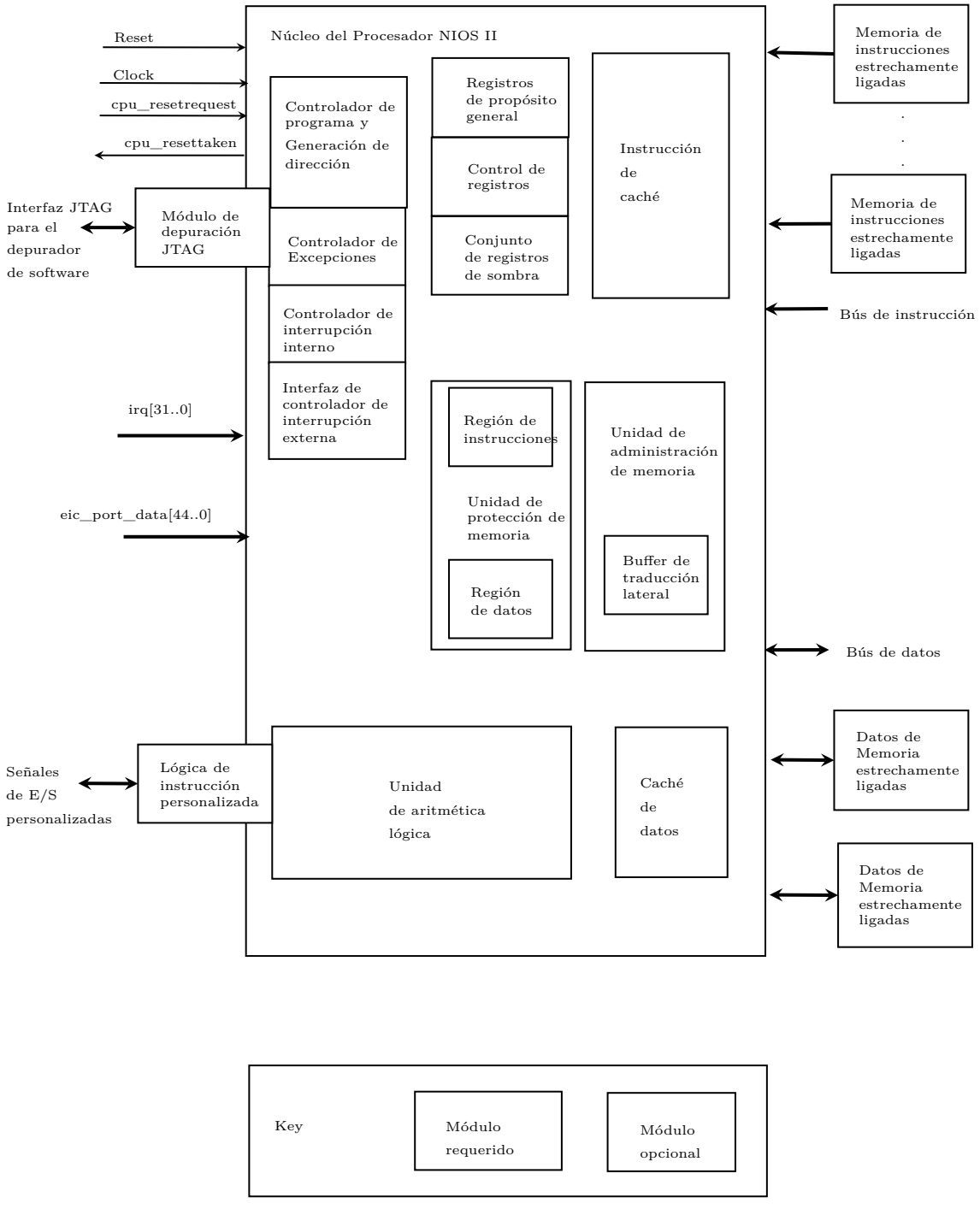

FIGURA 3.10: Arquitectura del procesador NIOS II

un conjunto de unidades funcionales para implementar las instrucciones.

La arquitectura de NIOS II define las siguientes unidades funcionales:

- Archivo de registro.

- Unidad Aritmética Lógica (ALU).
- Interfaz de instrucciones lógicos personalizado.
- Controlador de excepciones.
- Controlador de interrupción interna o externa.
- Bus de instrucciones.
- Bus de datos.
- Unidad de administración de memoria(MMU, Memory Management Unit).
- Unidad de protección de memoria (MPU, Memory Protection Unit).
- Instrucciones y datos de la memoria caché.
- Interfaces de memoria acopladas para instrucciones y datos.
- Módulo de depuración JTAG.

El procesador Nios II es un núcleo de procesador RISC (Se entiende por procesador RISC aquel que tiene un conjunto de instrucciones con unas características determinadas. Una instrucción no es más que una indicación de lo que debe de hacer el micro con los datos.) de uso general con las siguientes características:

- Conjunto completo de instrucciones de 32 bits, ruta de datos y espacio de direcciones.
- 32 registros de propósito general.
- Conjunto de registros de sombra opcionales.
- 32 fuentes de interrupción.
- Interfaz de controlador de interrupción externa para más fuentes de interrupción.
- Módulo de depuración asistida por hardware que permite el inicio, la detención, el paso y el seguimiento del procesador bajo el control de las herramientas de desarrollo de software Nios II.
- Entorno de desarrollo basado en la cadena de herramientas GNU C/C++ y herramientas de desarrollo de software de NIOS II para Eclipse.

En la práctica, la mayoría de los diseños de FPGA implementan una lógica adicional además del sistema de procesador. Los FPGA de Altera proporcionan flexibilidad para agregar funciones y mejorar el rendimiento del sistema de procesador Nios II. También puede eliminar las características y los periféricos innecesarios del procesador para adaptarlos al diseño en un dispositivo más pequeño y de menor costo.

3.6.1. Archivo de registros

La arquitectura Nios II admite un archivo de registro, que consta de 32 registros de 32 bits de propósito general y hasta 32 registros de control de 32 bits. La arquitectura admite modo supervisor/usuario que permiten que el código del sistema proteja los registros de control de las aplicaciones erróneas. El procesador Nios II puede tener uno o más conjuntos de registro de sombras. La utilidad de estos registros es acelerar el cambio de contexto.

3.6.2. Unidad Aritmética Lógica (ALU)

La unidad aritmética lógica de la arquitectura NIOS II trabaja con datos almacenados en registros de propósito general. Las operaciones toman dos entradas de dichos registros y los almacena en otro. Las operaciones que soporta esta arquitectura se describen en la siguiente tabla 3.3:

Categoría	**Descripción**
Aritmética	La ALU admite operaciones de suma, resta, multiplicación y división, en operandos sin signo y con signo
Relacional	La ALU admite operaciones relacionales de: igual, diferente de, mayor igual que, y menor que(==,!=, >=, <) en operandos con y sin signo.
Lógica	La ALU admite operaciones lógicas: AND, OR, NOR Y XOR
Desplazar y rotar	La ALU admite operaciones de desplazamiento a la izquierda/derecha y rotación a la izquierda/derecha

Tabla 3.3: Operadores de la ALU del procesador NIOS II

CAPÍTULO 4

Pruebas del sistema operativo FreeRTOS sobre el procesador NIOS II y comparación con otra arquitectura

En este apartado se describe el procedimiento que se lleva a cabo para implementar tanto el procesador NIOS II, así como la implementación de FreeRTOS.

4.0.0.1. Implementación del procesador NIOS II en la FPGA en el entorno de trabajo de Qsys

Para implementar el procesador NIOS II se realiza el siguiente procedimiento:

1. Mediante la plantilla System_Builder, se agregan los elementos a utilizar:
   - Clock.
   - Button X2.
   - LEDX8.
   - Dip Switch X4.
   - EPCS.
   - 2x13Pin Header.
2. Y se guarda el archivo creado.

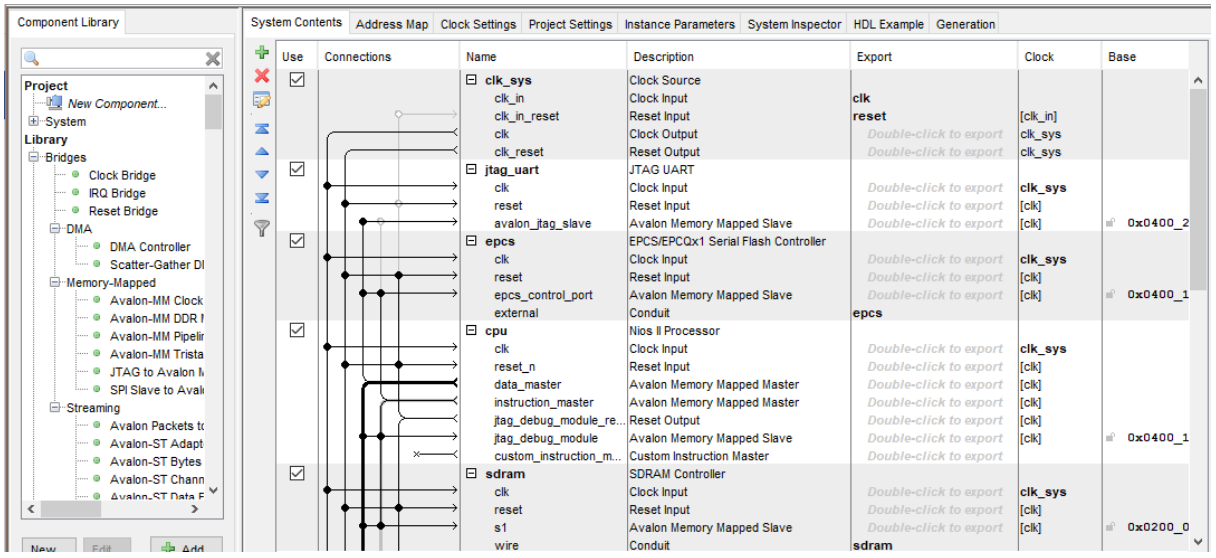

FIGURA 4.1: Elementos necesarios para la implementación del procesador NIOS II

3. Posteriormente abrir el proyecto creado con extensión .V en Quartus.

4. Mediante Qsys se agregan los siguientes componentes:

   - Clock source configurado a 50MHz.
   - NIOS II Processor.
   - System ID Peripheral.
   - SDRAM_controller.
   - EPCS/EPCQx1 Serial Flash Controller.
   - Clock source configurado a 100MHz.
   - Clock sorce configurado a 100MHz.
   - Avalon ALTPLL.
   - Altera Avalon LCD 16207.
   - JTAG_UART.
   - UART (RS-232 Serial Port).

5. Ver figuras 4.1 y 4.2.

6. Después de agregar todos los elementos mencionados anteriormente, renombrar los elementos:

   - Clock source se renombra clk_50.
   - NIOS II Processor se renombra CPU.

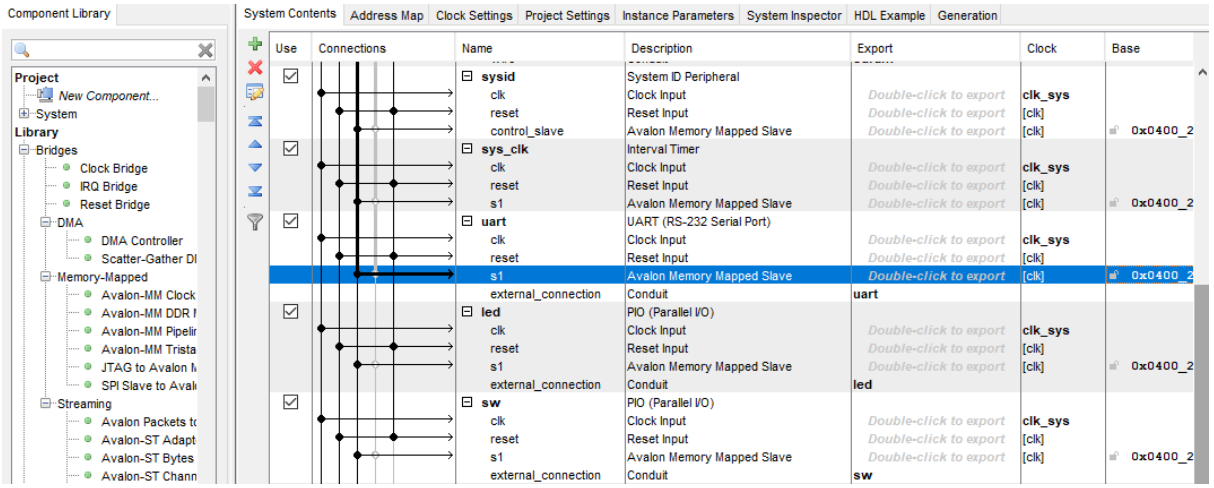

FIGURA 4.2: Elementos necesarios para la implementación del procesador NIOS II

- System ID Peripheral se renombra sysid.
- SDRAM_controller se renombra sdram.
- EPCS/EPCQx1 Serial Flash Controller se renombra como epcs.
- Interval Timer se renombra sys_clk.
- PIO(Parallel I/O) se renombra led.
- PIO(Parallel I/O) se renombra sw.
- Clock source se renombra clk_sys.
- Clock source se renombra ram_clk.
- Avalon ALTPLL se renombra sys_pll.
- Altera Avalon LCD 16207 se renombra lcd.
- JTAG_UART se renombra como jtag_uart.
- UART (RS-232 Serial Port) se renombra como uart.....

7. Después de esto se procede a hacer la conexión de cada uno de los componentes de la forma que sigue:

   - Conectar clk de clk_sys a clk de cada uno de los componentes.
   - Conectar clk_reset a reset de todos los demás elementos.
   - Conectar avalon_jtag_slave de jtag_uart a epcs_control_port de epcs, data_master del CPU, jtag_debug_module del CPU, s1 del sdram, control_slave de sysid, s1 de sys_clk, s1 de uart, s1 de led y s1 de sw.
   - Reset de epcs se conecta a reset_n y jtag_debug_module_reset de CPU, reset de sdram, reset de sysid, reset de sys_clk, reset de uart, reset de led y reset de sw.

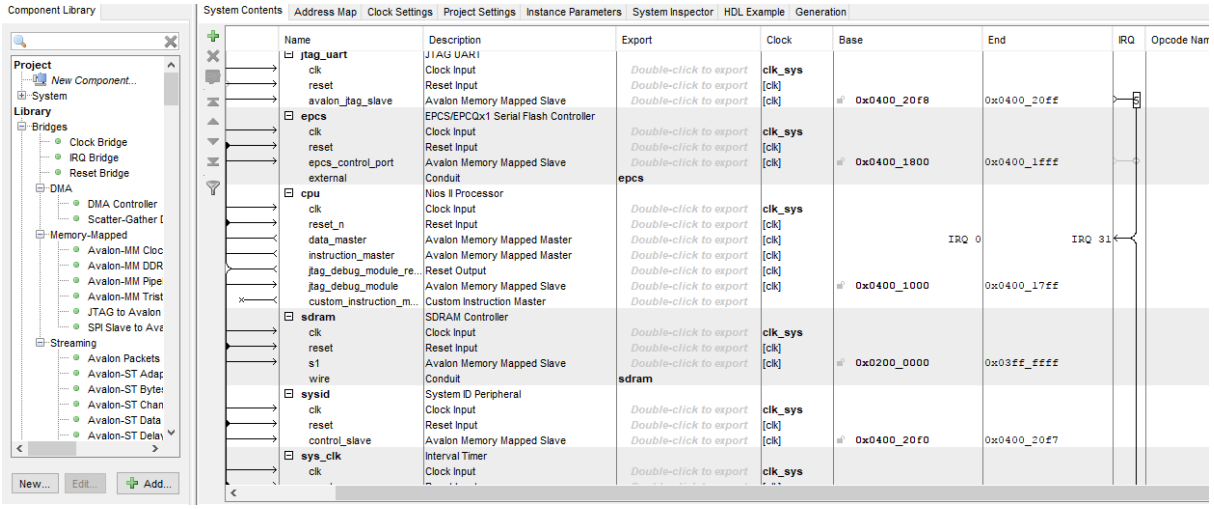

FIGURA 4.3: Conexión de elementos necesarios para la implementación del procesador NIOS II

- epcs_control_port se conecta con instruction_master de CPU, jtag_debug_module de CPU, y s1 de sdram.

8. También se realiza la conexión de IRQ0 de jtag_uart al IRQ31 de CPU, al igual que el IRQ0 de sys_clk y al IRQ2 de uart, ver figuras 4.3 y 4.5. Con estas conexiones se guarda el diseño con extensión .qsys.

9. Después de realizar todas las conexiones mencionadas anteriormente, click en el menú "System" y "Assign Base Addresses" ver figura 4.4 ir a "Assign Interrupt Number", con esto se deben de eliminar los errores.

10. Una vez generado el archivo con extensión .qsys se procede a hacer la instanciación en el entorno de trabajo de Quartus II, la forma de hacer esto es como sigue:

    - ubicarse en la parte superior izquierda de la ventana de Qsys click en el menú "Generate", se despliega otro menú, click en "Generate.." ver figura. 4.6.
    - Con esto se despliega otra ventana ver figura 4.7, dar click en "Generate" para generar el HDL correspondiente.
    - después ubicarse en la esquina superior izquierda del entorno de trabajo de Qsys dar click en el menú "Generation" con esto se desplegara otro menú, dar click en "HDL Example" con esto se genera el módulo a instanciar en Quartus II. Ver figura 4.8
    - copiar la instancia y pegar en el documento con extensión .v generado en el primer paso.

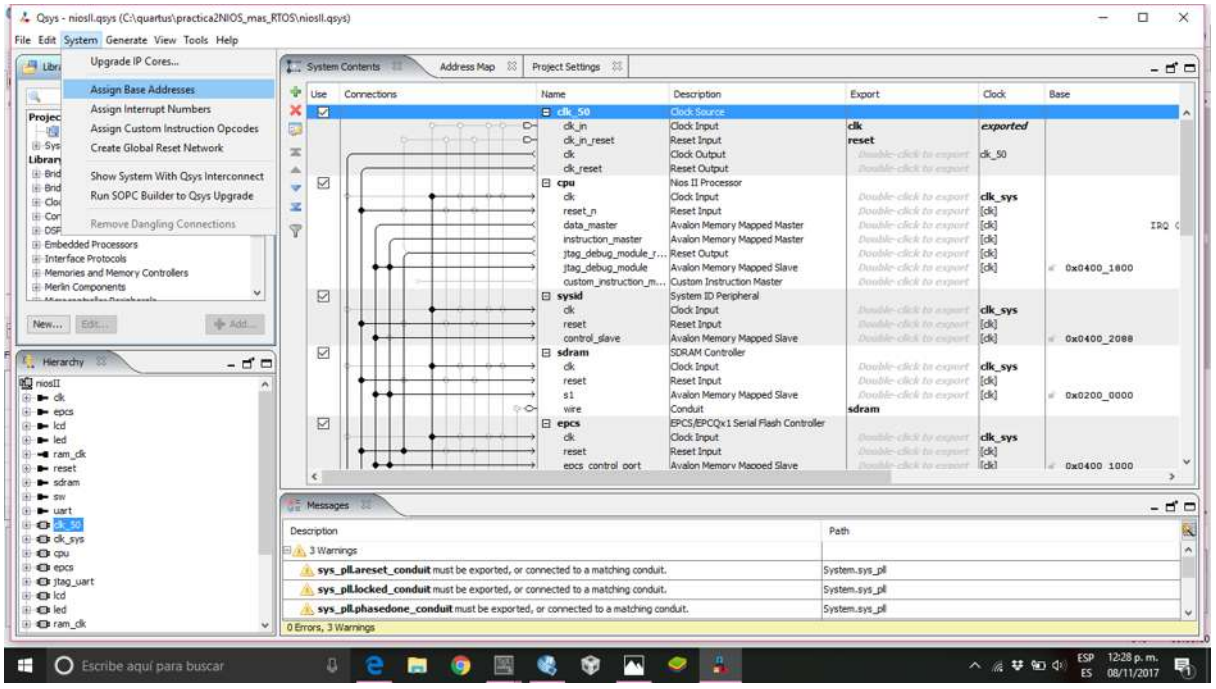

FIGURA 4.4: Menú para corrección de errores

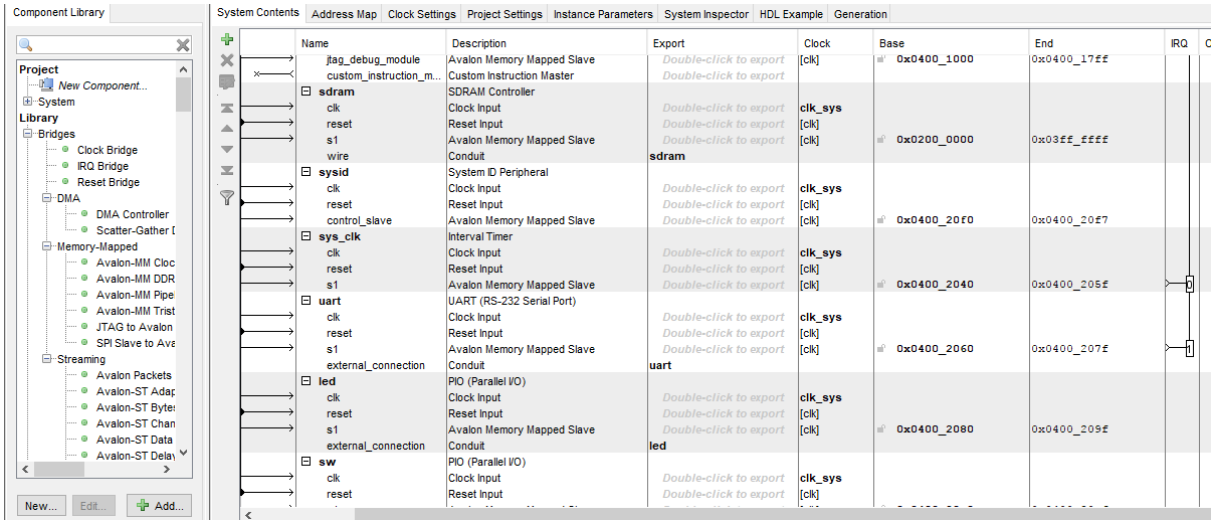

FIGURA 4.5: Conexión de elementos necesarios para la implementación del procesador NIOS II

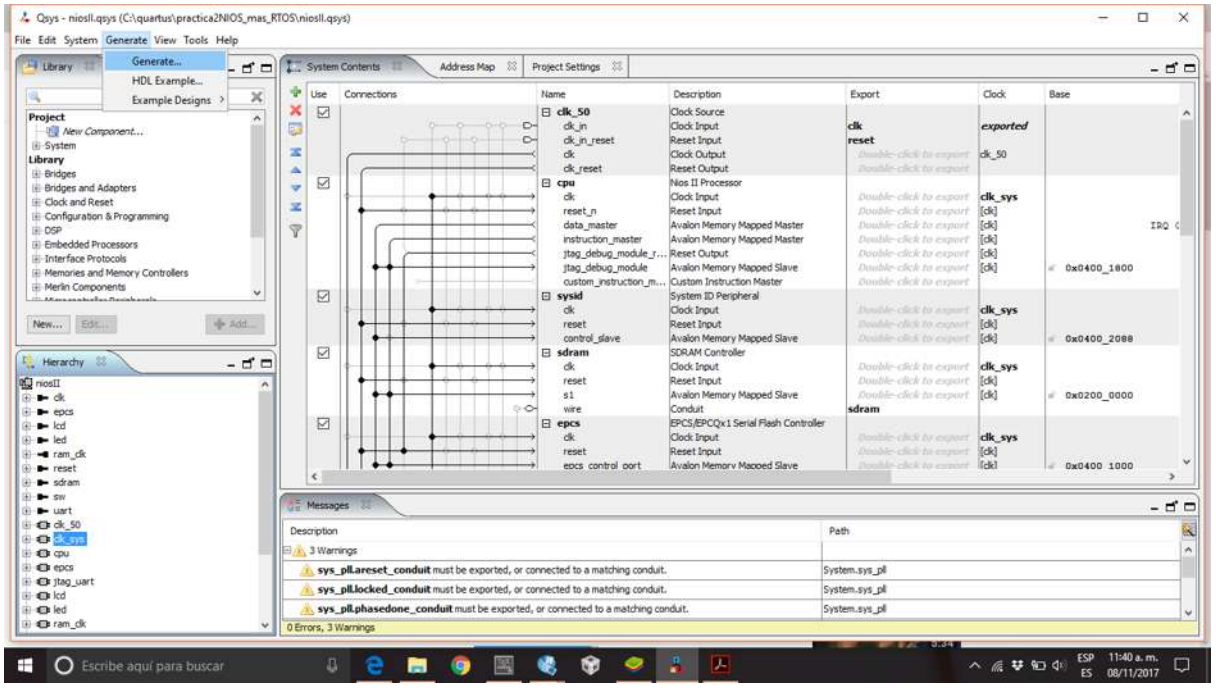

FIGURA 4.6: Menú para generar el HDL

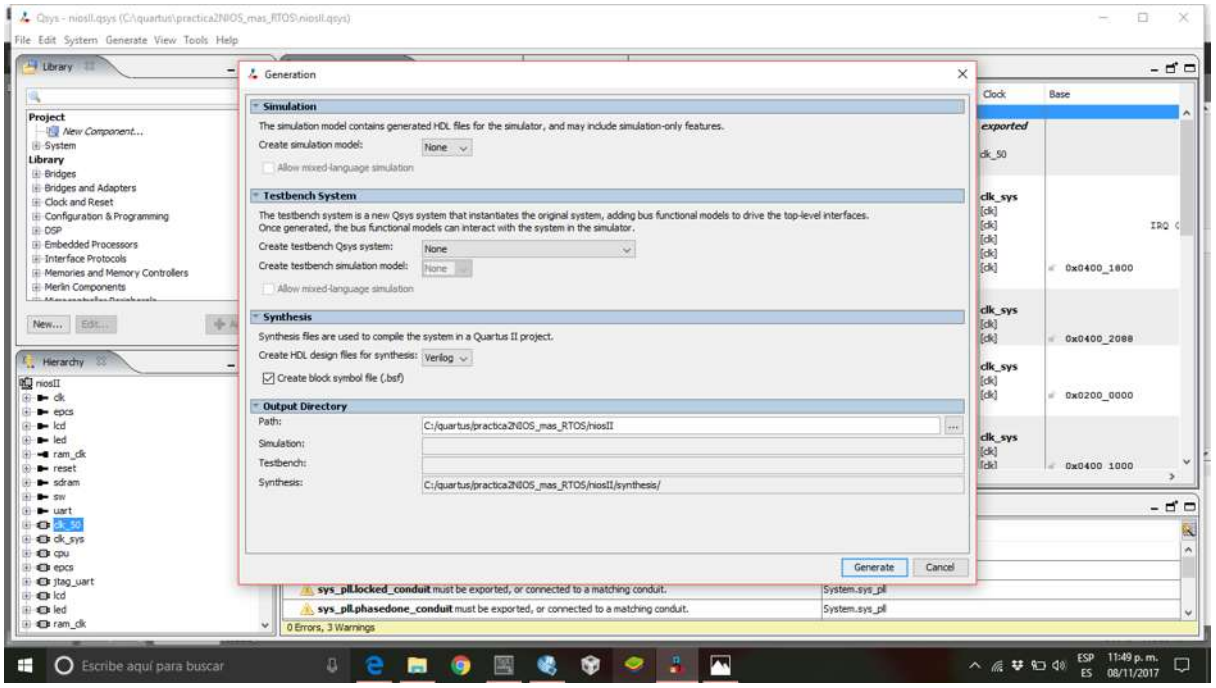

FIGURA 4.7: Menú para generar el HDL

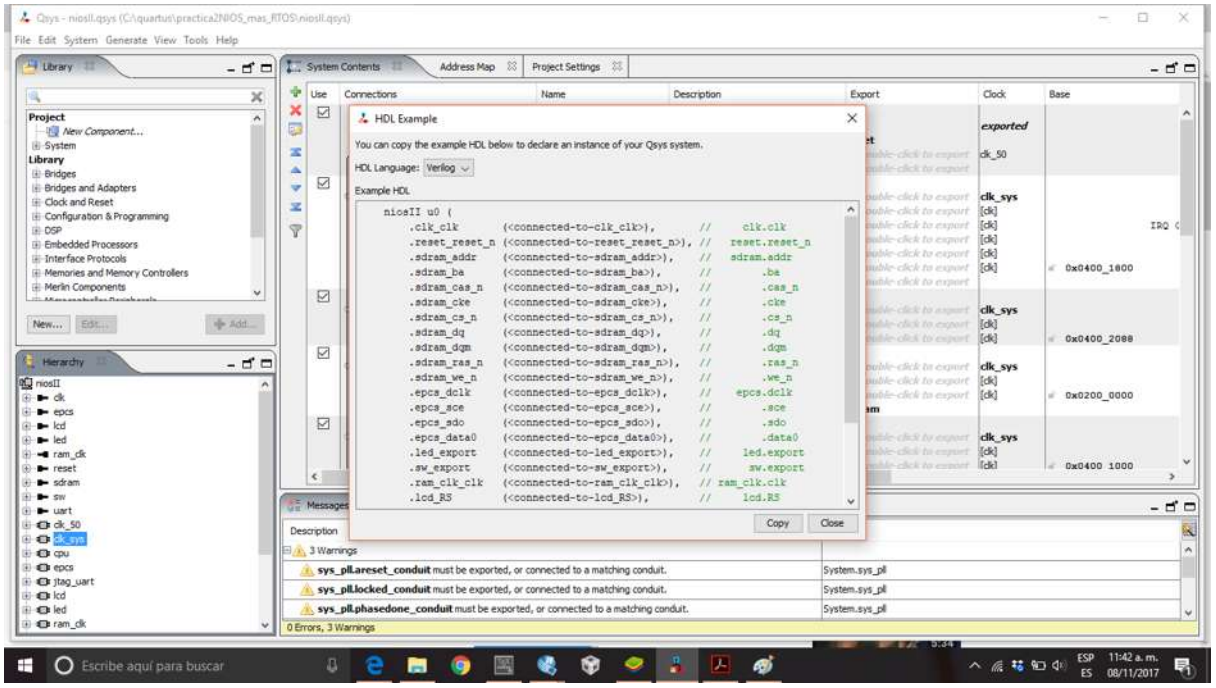

FIGURA 4.8: Archivo generado para hacer la instancia

4.0.0.2. Implementación del procesador NIOS II en el entorno de trabajo de Quartus II

1. Dentro del entorno de trabajo de Quartus, se debe añadir los archivos con extension .sdc, .v y .qsys, la forma de hacer esto es ubicarse en el menú "File", click derecho sobre la carpeta del proyecto y "Add File" y se despliega una lista de todos los archivos, solo se seleccionaran los archivos con las extensiones antes mencionadas, esto con el fin de hacer el análisis y la síntesis.

2. Después de esto se hace el análisis y la síntesis con "Start Analysis & Synthesis". Ver figura 4.9

3. Para poder ver el diagrama general ir al menú "Tools", "Netlist Viewers" y "RTL Viewer". Ver figura 4.10

4. Se tiene que generar el archivo con extensión .sof, este se genera con "Start Compilation",ver figura 4.11 este archivo servirá posteriormente para programar la FPGA Cyclone IV.

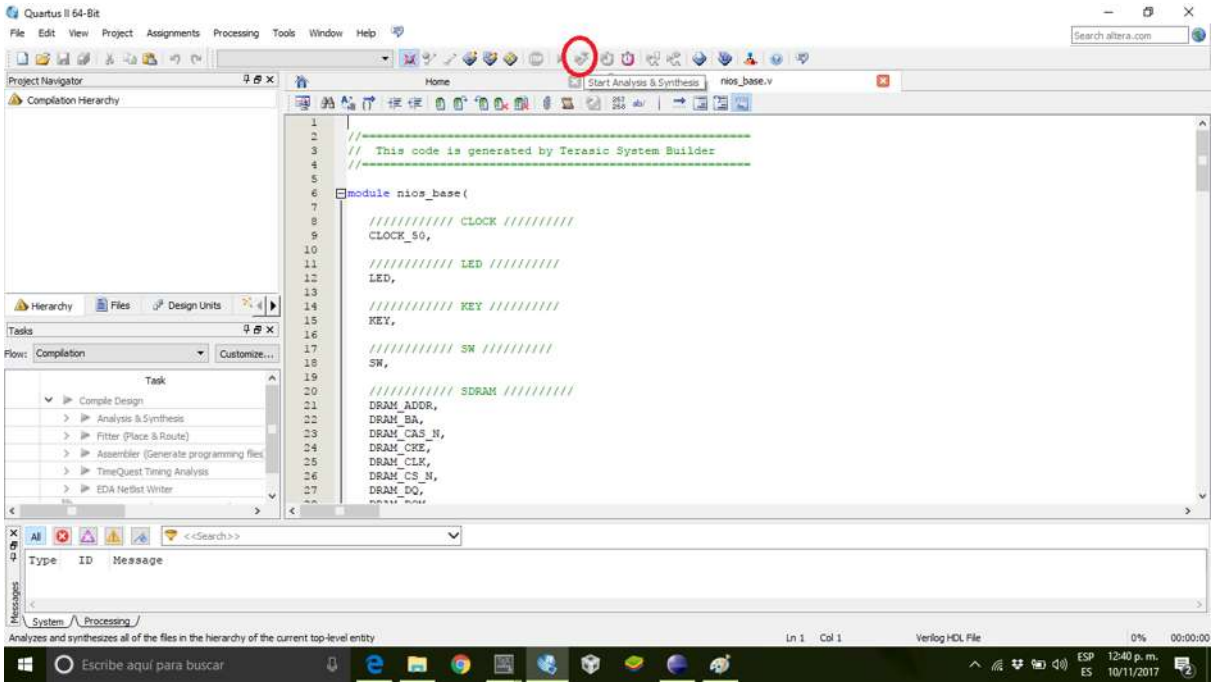

FIGURA 4.9: Menú para hacer el análisis y la síntesis del proyecto

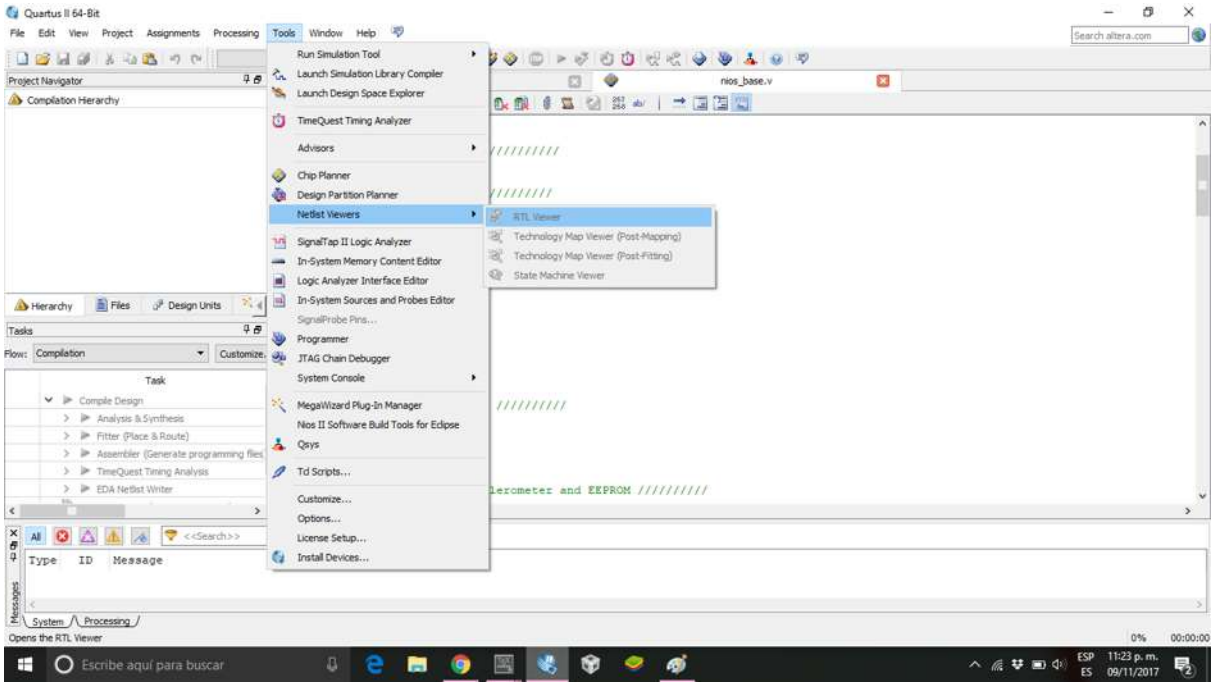

FIGURA 4.10: Menú para ver el diagrama general del procesador

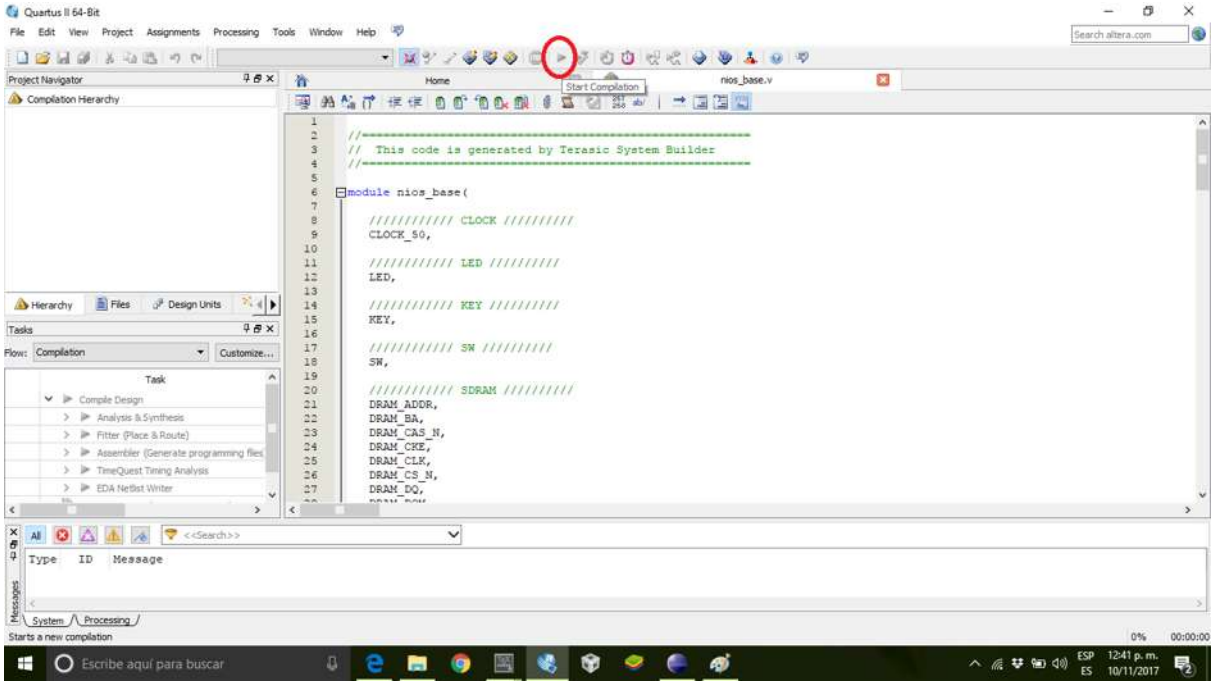

FIGURA 4.11: Menú para generar el archivo con extensión .sof

4.0.0.3. Plantilla de prueba, dentro del entorno de trabajo de Eclipse

Después de realizar todos los pasos anteriores, lo siguiente es escribir el programa para realizar las pruebas necesarias para esta investigación, y para esto se debe de ejecutar Eclipse con privilegios de administrador, esto es indispensable para evitar problemas posteriores a la hora de programar la FPGA.

1. Estando dentro de Eclipse, se crea un nuevo proyecto, para esto ir al menú "File", click en "News", "Nios Aplication and BSP From Template", ver figura 4.12, se despliega una ventana con la opción de agregar un archivo con extensión .sopc, se agrega el que se generó en pasos anteriores, se escoge la plantilla "Hola mundo" con el propósito de comprobar si todo lo realizado hasta ahora, funciona correctamente, se asigna un nombre al proyecto y click en "Next" y "Finish". Ver figura 4.13

2. Cuando ya tenemos nuestra plantilla de prueba, generamos el BSP, click derecho sobre la carpeta _bsp, se despliega un menú, click en la opción "Nios II", con esto se despliega otro menú, click en "Generate BSP", ver figura 4.14

3. Después click derecho sobre la carpeta del proyecto generado y en "Build Proyect"

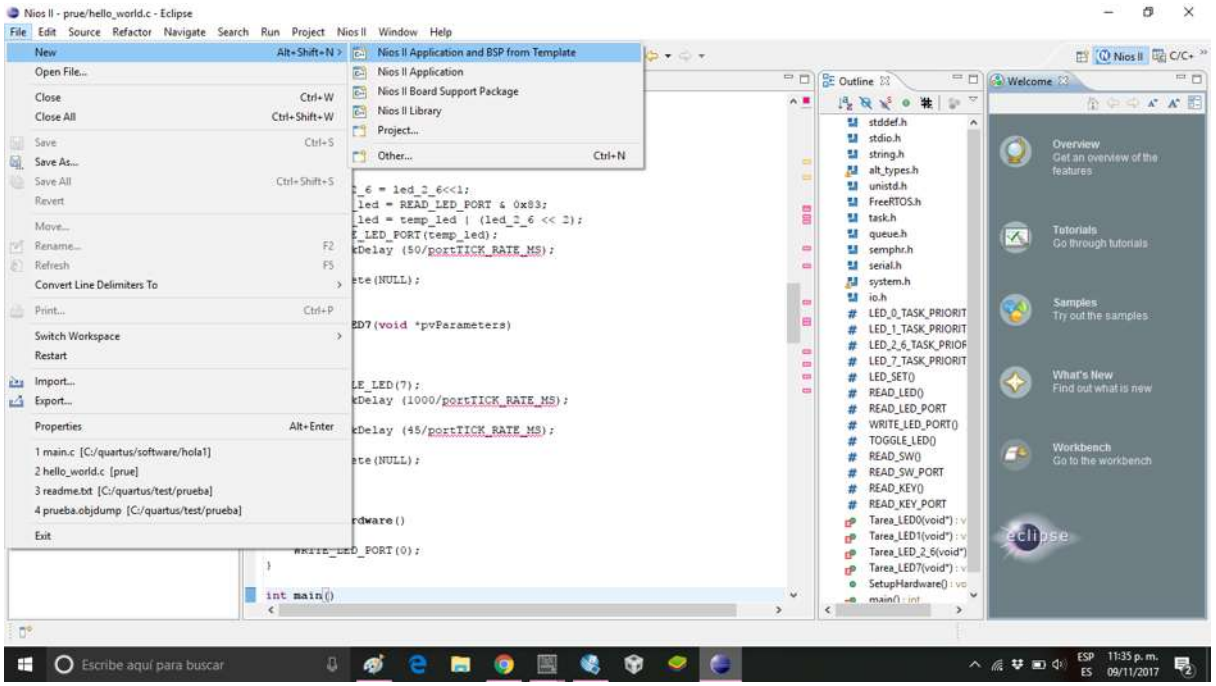

FIGURA 4.12: Menú para crear una nueva plantilla

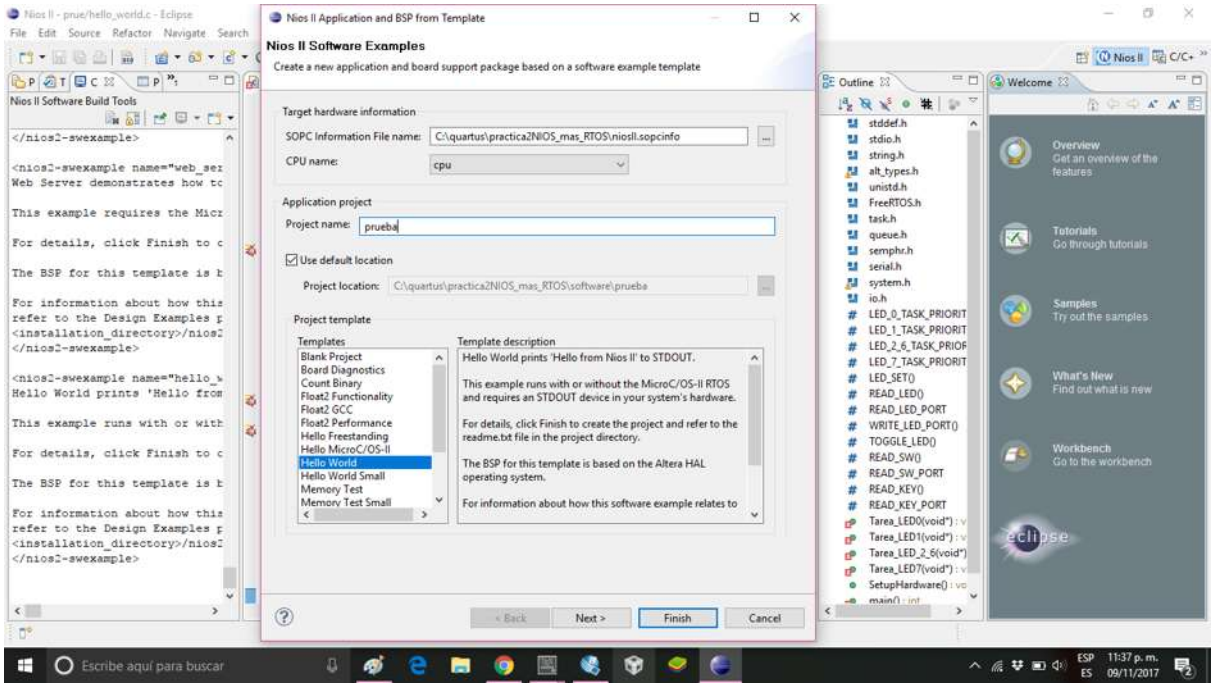

FIGURA 4.13: Crear la plantilla de prueba: "Hola mundo"

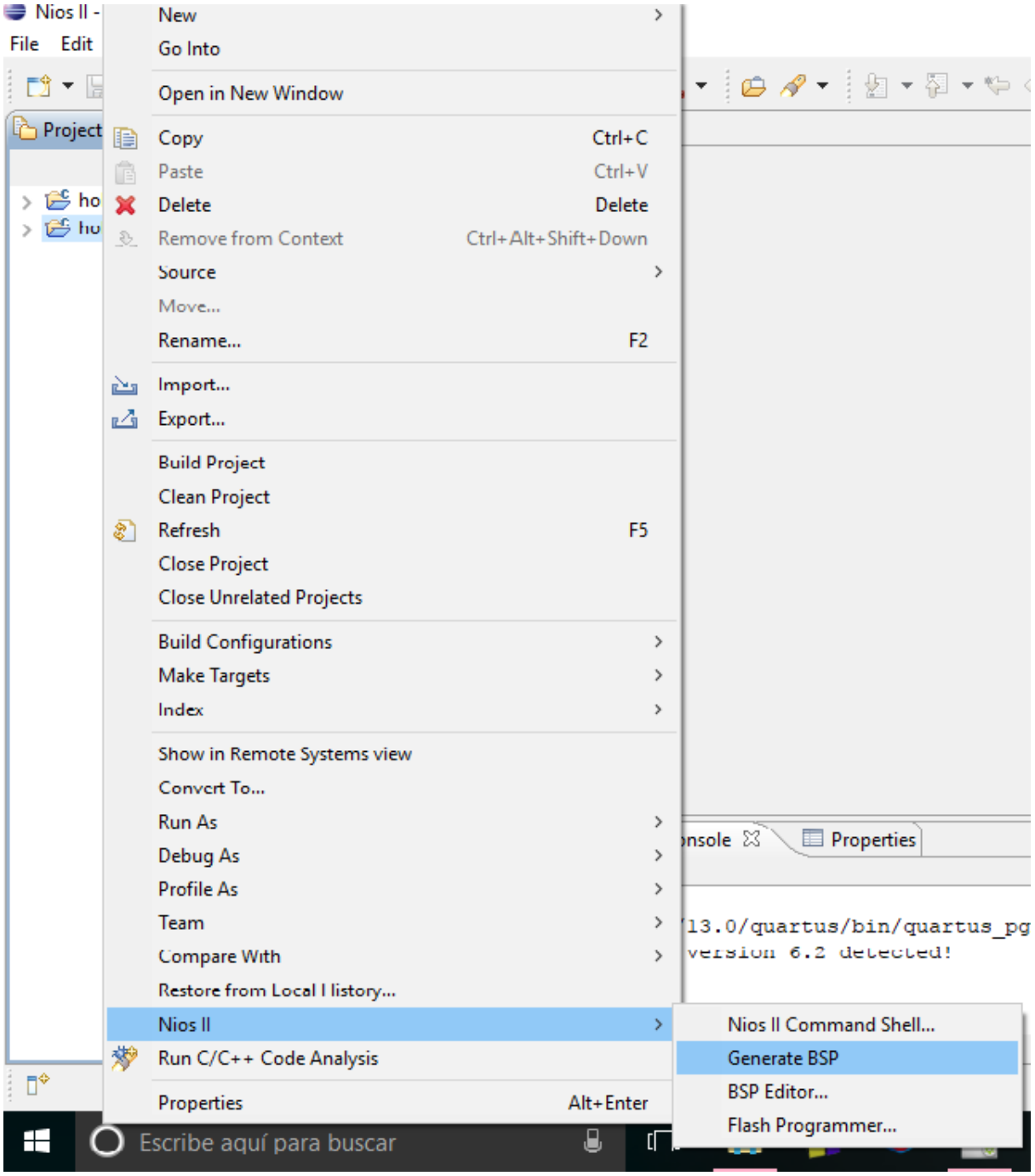

FIGURA 4.14: Menú desplegado para generar BSP

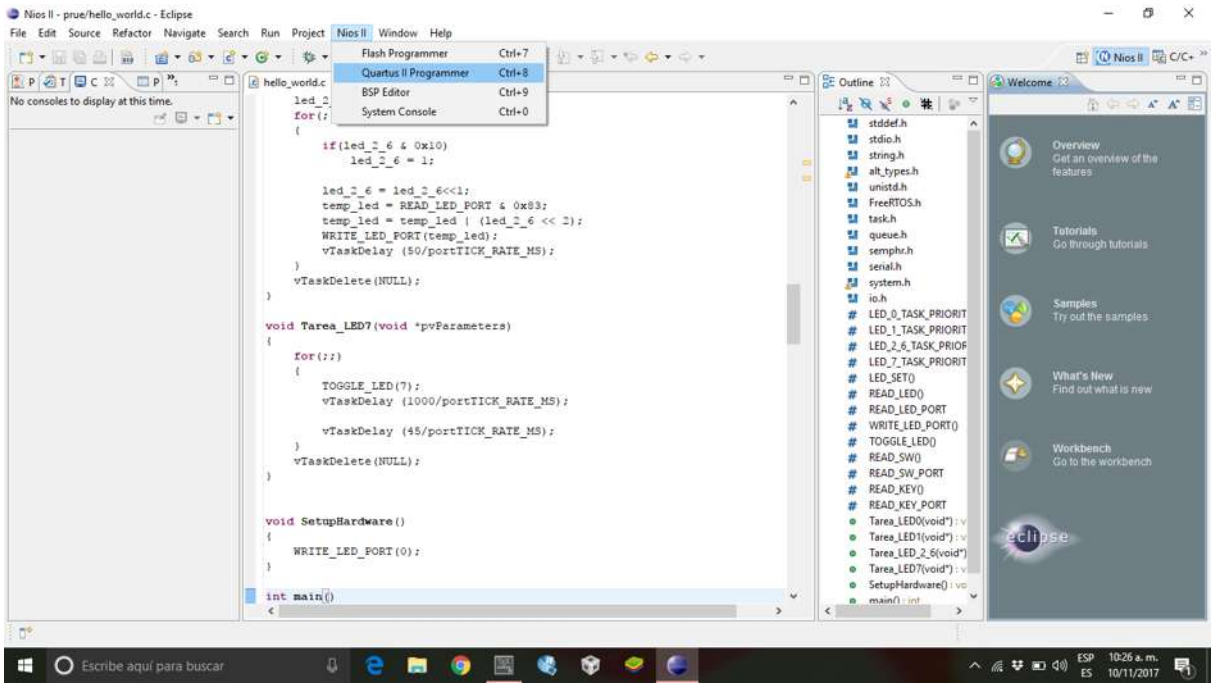

FIGURA 4.15: Menú para programar la FPGA

4.0.0.4. Programación de la FPGA

Después de realizar todos los pasos anteriores, lo que sigue es programar el dispositivo, la FPGA Cyclone IV, para eso se tiene que seguir el siguiente procedimiento:

1. En eclipse, ir al menú "Nios II" y dar click en "Quartus II Programmer", ver figura 4.15
2. con esto se despliega una ventana con varias opciones, buscamos la opción "Auto Detect" y seleccionamos el dispositivo EP4CE22, "OK" y con esto aparece nuestro dispositivo, figura 4.16
3. clik derecho sobre el dispositivo que aparece dentro de la ventana y se despliega un menú clik en "Edit", se despliega otro menú, ver figura 4.17,
4. clik en "Change File" y con esto se despliega otra ventana en donde se tiene que agregar el archivo con extensión .sof,
5. una vez terminado este proceso clik en "Start", y en la esquina superior derecha se debe de leer "100 %(successful)", ver figura 4.18 con esto la configuración se termina. Cerrar la ventana y seguir.

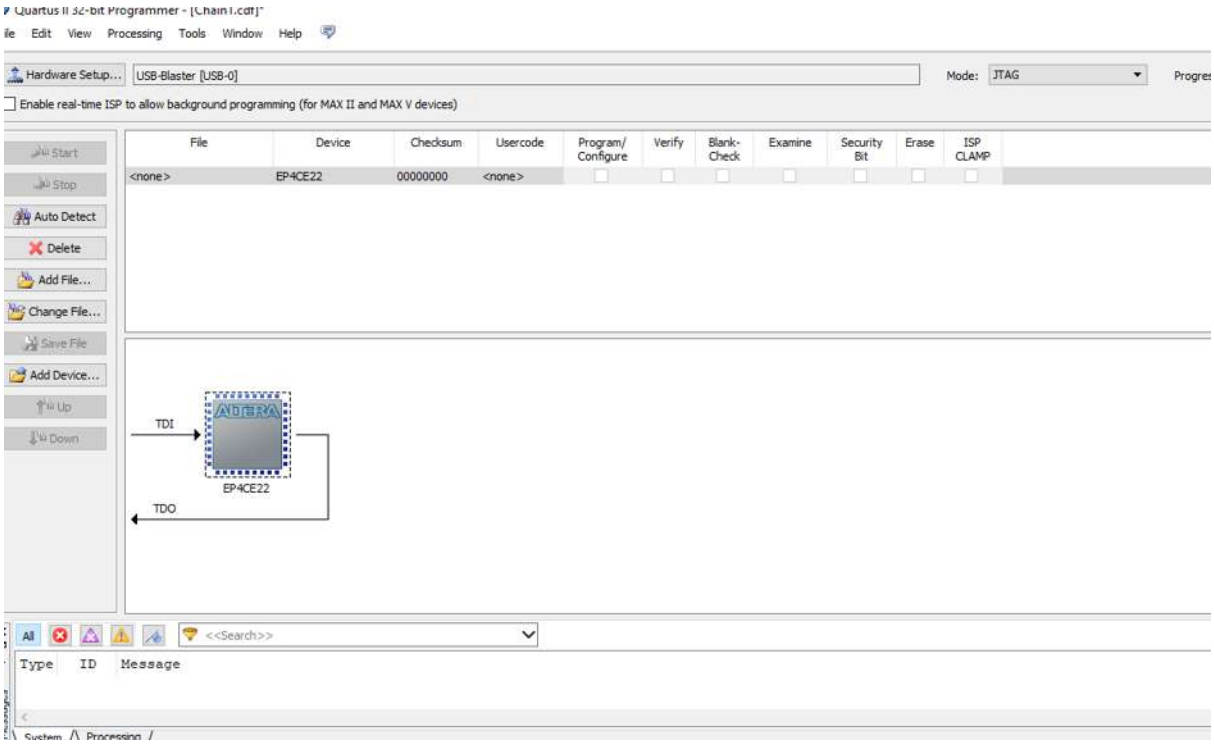

FIGURA 4.16: Procedimiento para agregar el dispositivo, FPGA Cyclone IV EP4CE22F17C6N

4.0.1. Pruebas realizadas y resultados

4.0.1.1. Pruebas realizadas con el microcontrolador MSP430F5529

En este trabajo se va a comparar las ventajas de implementar un sistema de control en un RTOS implementado en el procesador NIOS II contra la estrategia tradicional que usa el microcontrolador MSP430F5529.

En esta primera parte se muestran las pruebas realizadas con el MSP430F5529, en primera instancia se ejecuta el código mostrado a continuación. Con este código el led 0 parpadea a una frecuencia de 1Hz ver figura 4.26

```
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    P6DIR|=(BIT0+BIT1+BIT2+BIT3+BIT4+BIT5+BIT6);
    // Colocar los pines del puerto 6 como salidas
        P6OUT&=~( BIT0 + BIT1 + BIT2 + BIT3 + BIT4 + BIT5 + BIT6 );
        // Poner todas las salidas en bajo
```

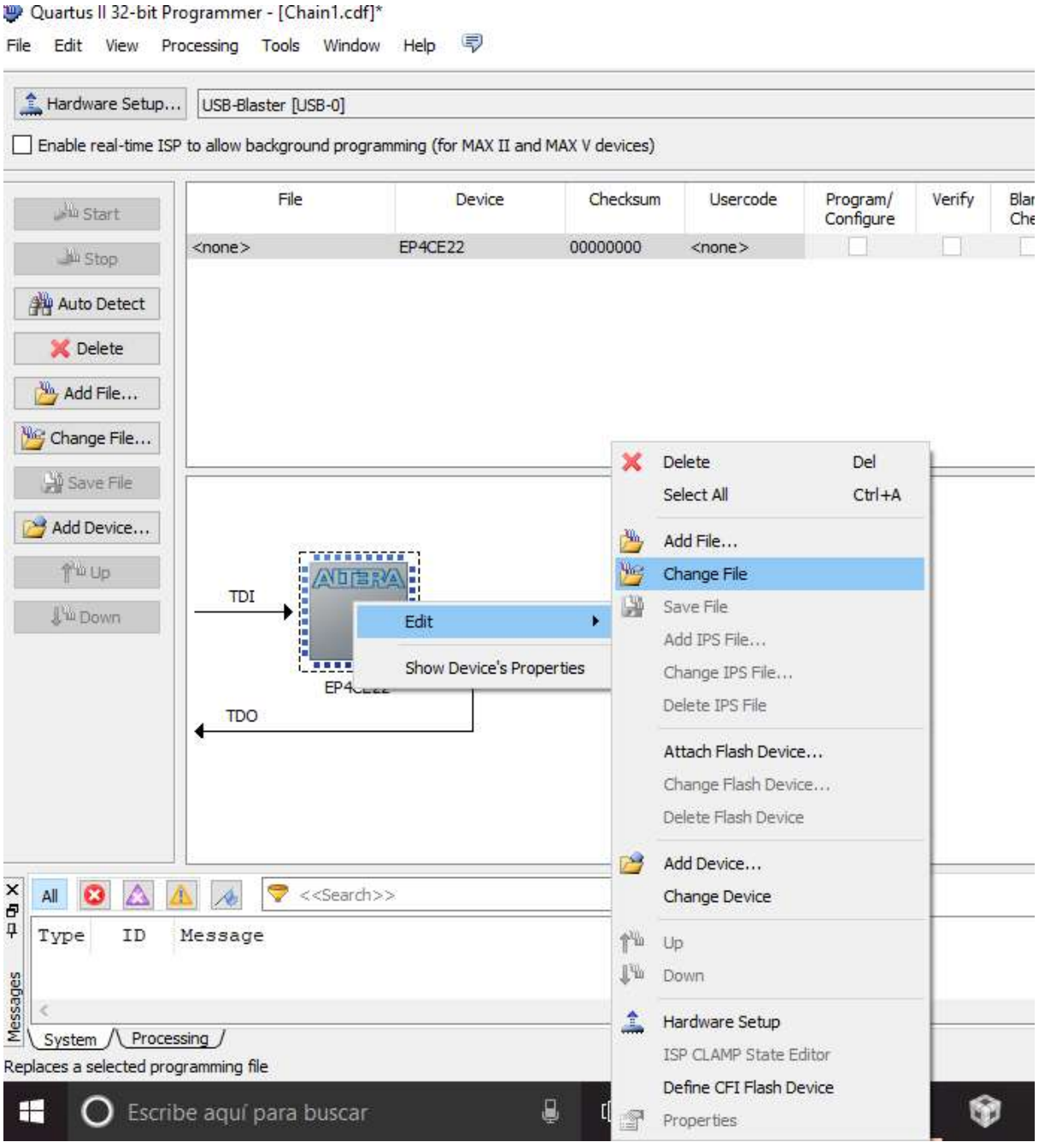

FIGURA 4.17: Opciones para agregar un archivo

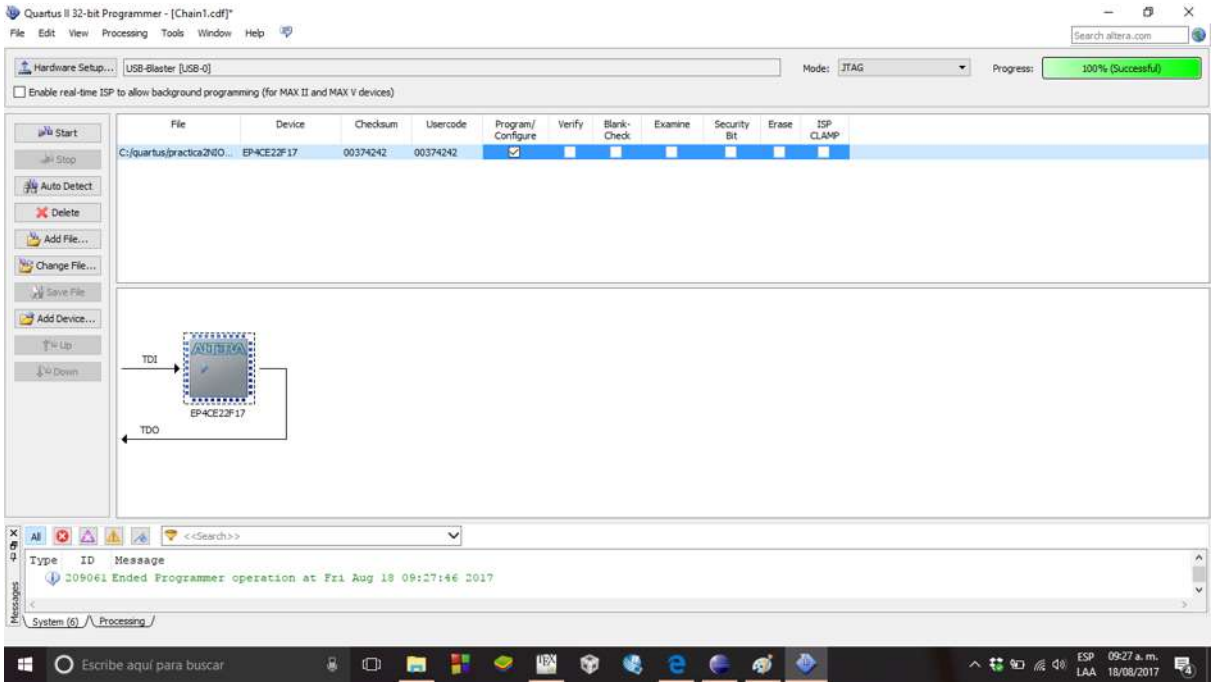

FIGURA 4.18: Ventana emergente para la programación de la FPGA

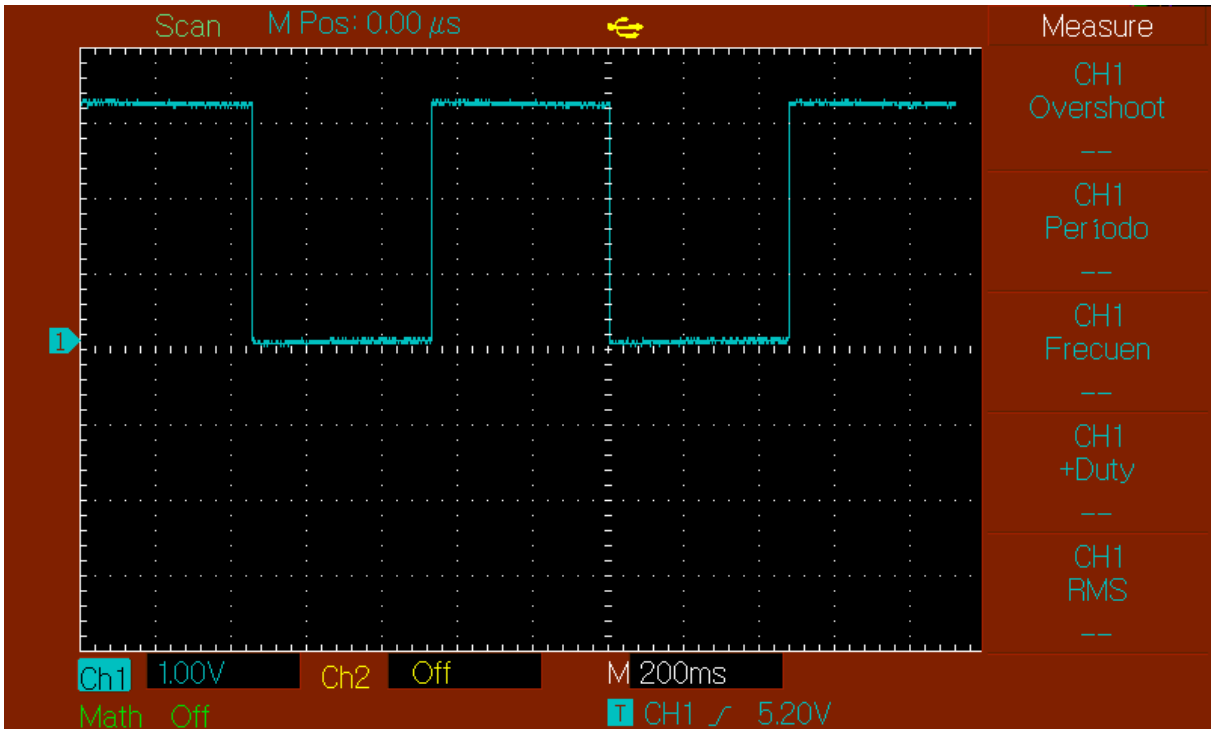

FIGURA 4.19: led 0

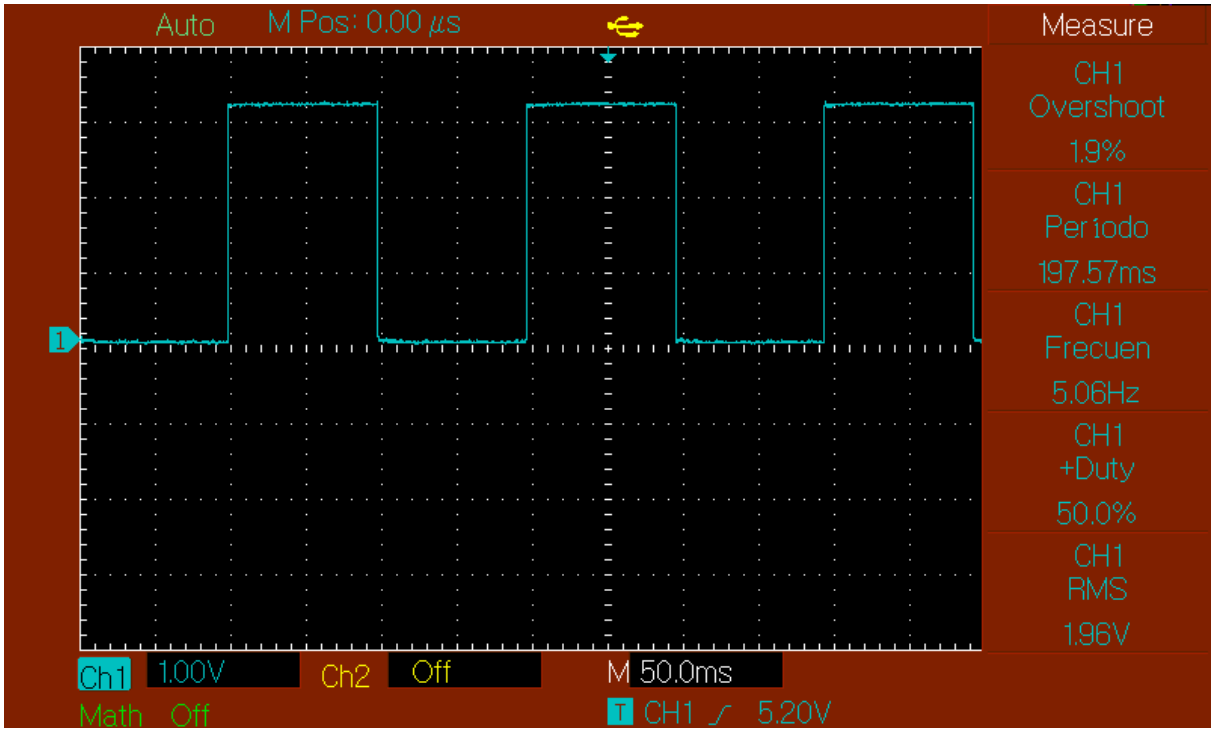

FIGURA 4.20: led 1

```
unsigned int contador_led0 =500;
// Declaraci ón de los contadores

while (1){
// Esta parte del có digo se encarga de que el led 0
// parpadee a una frec . de 1Hz
contador_led0 --;
if( contador_led0 <=0){
P6OUT ^= BIT0 ;
contador_led0 =500;
}
__delay_cycles (1000);
}
}
```

Con el código siguiente el led 1 parpadea a una frecuencia de 5Hz ver figura 4.27

```
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;    // stop watchdog timer
```

1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

TABLA 4.1: Corrimiento de los led's

```
    P6DIR|=(BIT0+BIT1+BIT2+BIT3+BIT4+BIT5+BIT6);
    // Colocar los pines del puerto 6 como salidas
        P6OUT&=~( BIT0 + BIT1 + BIT2 + BIT3 + BIT4 + BIT5 + BIT6 );
        // Poner todas las salidas en bajo
        unsigned int contador_led1 =100;
        // Declaración de los contadores

        while (1){

            // Esta parte del có digo se encarga de que el led 1
                // parpadee a una frec . de 5Hz
                contador_led1 --;
                if( contador_led0 <=0){
                P6OUT ^= BIT0 ;
                contador_led0 =500;
                }

                __delay_cycles (1000);
        }
        }
```

El siguiente código hace el corrimiento de los leds del 2 al 5 que se muestra en la tabla 4.1, haciendo que cada led esté prendido 10Hz. Ver figura 4.28

```
#include <msp430.h>


int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    P6DIR|=(BIT0+BIT1+BIT2+BIT3+BIT4+BIT5+BIT6);
    // Colocar los pines del puerto 6 como salidas
        P6OUT&=~( BIT0 + BIT1 + BIT2 + BIT3 + BIT4 + BIT5 + BIT6 );
        // Poner todas las salidas en bajo
```

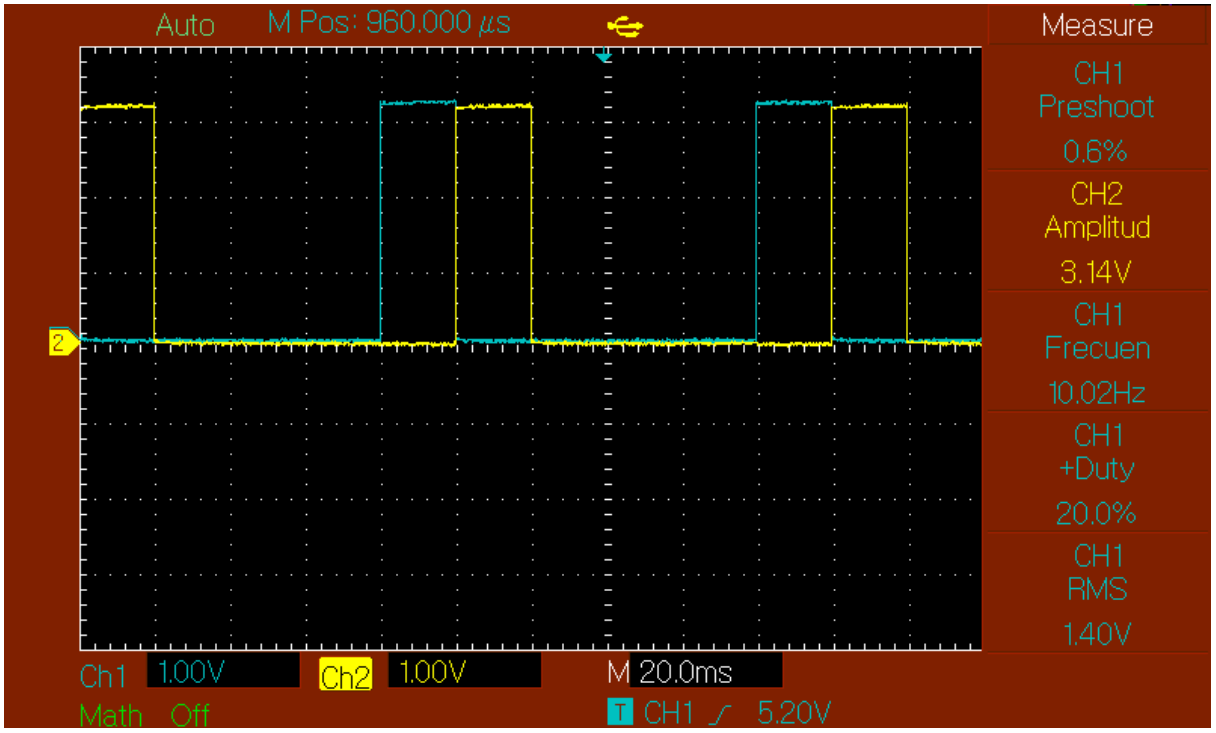

FIGURA 4.21: leds 2 y 3

```
contador_led2_6=1;
// Declaración de los contadores

while (1){

 //Esta parte del có digo se encarga de que los leds 1-5
 //hagan el corrimiento a una frec de 10 Hz.

    contador_led2_6--;
if( contador_led2_6 <= 0){
  if( P6OUT & 0x40 ){
 P6OUT &=~ BIT6 ;
P6OUT=P6OUT|BIT2;
 }
  else if(P6OUT==0x00){
       P6OUT |=BIT2;
  }
 else {
 P6OUT = P6OUT <<1;
 }
 contador_led2_6 =21;
```

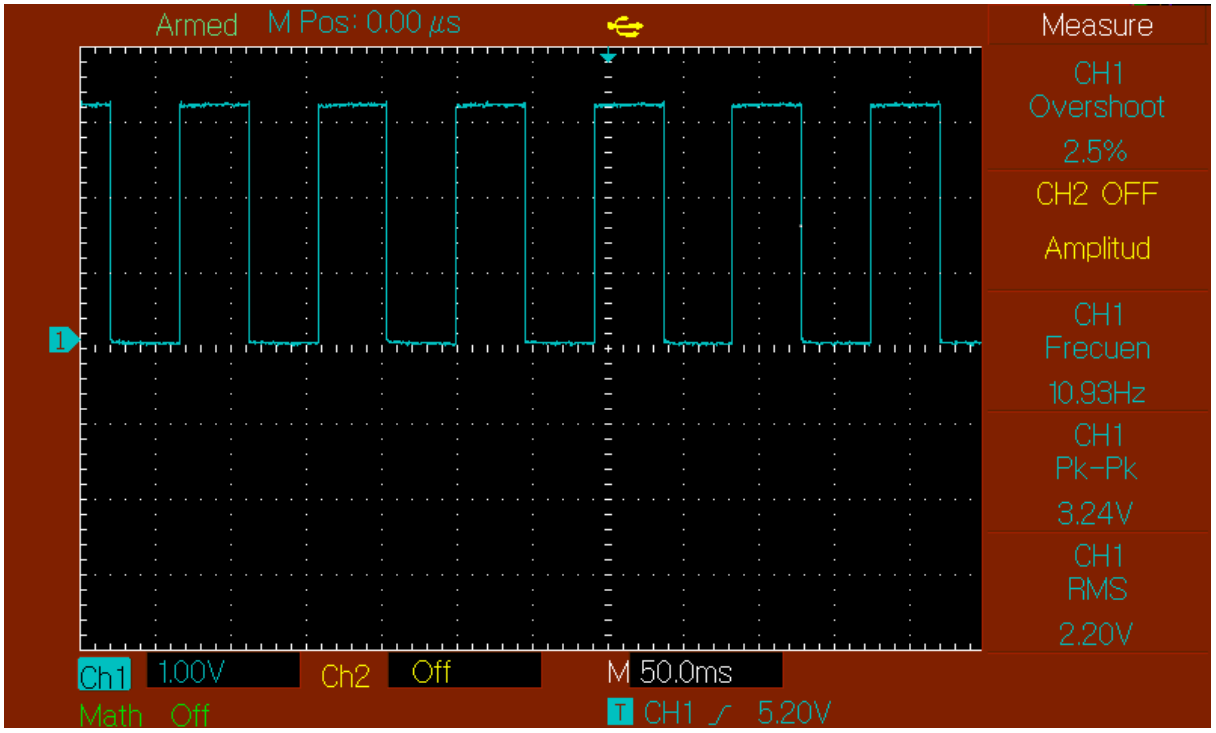

FIGURA 4.22: led 7

```
        }
        __delay_cycles (1000);
        }
        }
```

Este último código hace que el led 7 parpadee a una frecuencia de 11Hz. Ver figura 4.29.

```
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;    // stop watchdog timer
    P3DIR|=BIT2;
    // Colocar los pines del puerto 6 como salidas
        P3OUT&=~BIT2;
        // Poner todas las salidas en bajo
        unsigned int contador_ledex =45;
        // Declaraci ón de los contadores

        while (1){
        // Esta parte del có digo se encarga de que el
        // led 6 parpadee a una frec . de 11 Hz
```

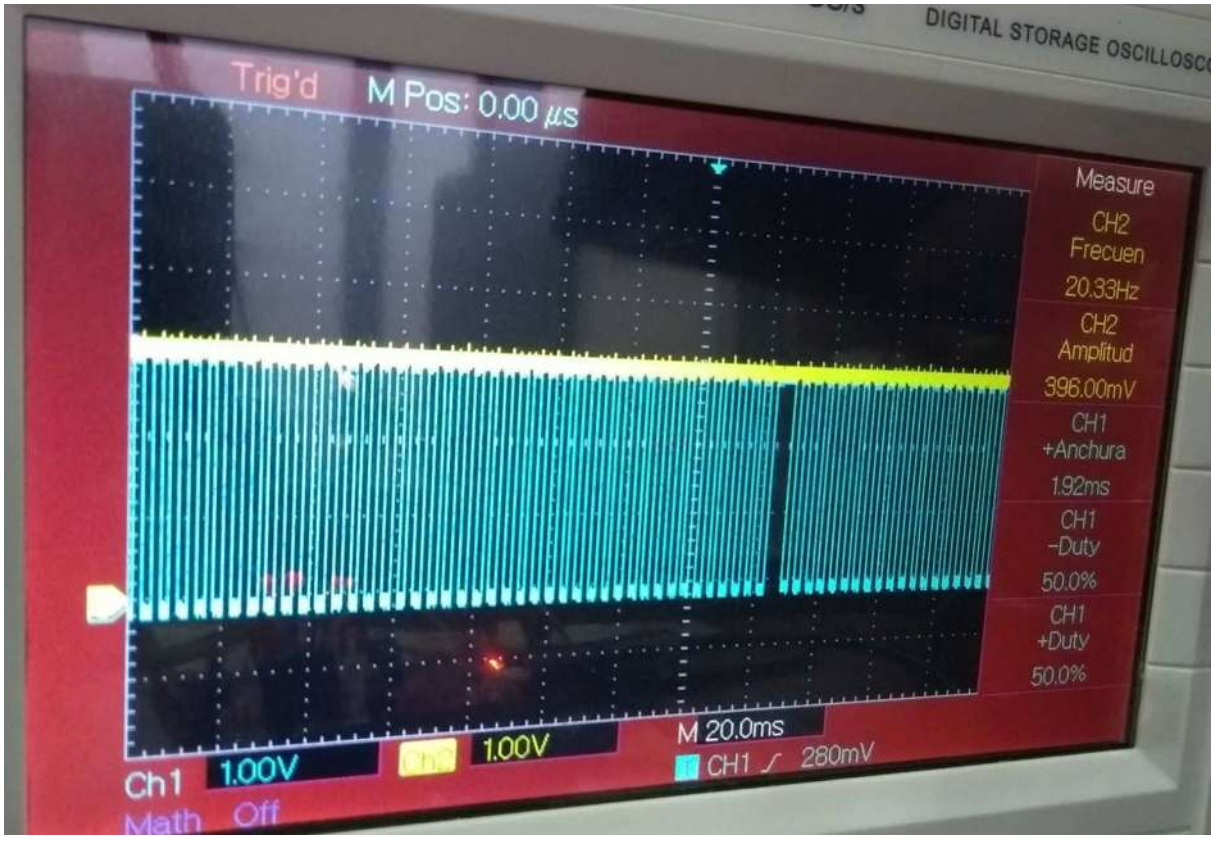

FIGURA 4.23: Captura de la lectura del osciloscopio: vista de los leds 2 y 3

```
contador_ledex --;
if( contador_ledex <=0){
P3OUT ^= BIT2 ;
contador_ledex =45;
}
__delay_cycles (1000);
}
}
```

Todos los códigos anteriores se juntaron en un sólo proyecto para su posterior ejecución y observar el resultado, el código final se muestra en el apéndice 6. Al momento de la ejecución los leds no siguen la secuencia establecida en el código, la figura 4.23 se toma de la lectura del osciloscopio medida en los leds 2 y 3.

4.0.1.2. Prueba realizada con FreeRTOS implementado en el procesador NIOS II

La siguiente prueba se realiza en la FPGA con el sistema operativo de tiempo real(FreeRTOS) implementado en el procesador NIOS II. Primero se tiene que programar la FPGA con el archivo que contiene el procesador NIOS II. Tal como se explica en la sección 'Programación de la FPGA'
Una vez programada la FPGA, se tienen que anexar la carpeta FreeRTOS, los archivos FreeRTOScheck.c, FreeRTOSConfig.h, serial.c y serial.h, descargados de la página oficial de FreeRTOS [15]; es necesario anexar estos documentos a nuestro proyecto para la instalación de FreeRTOS. Estos archivos se anexan desde el directorio, para que aparezcan en la carpeta del proyecto dentro del entorno de trabajo Eclipse, se da click derecho sobre el proyecto y "Refresh" con esto, todos los archivos anexados deben de aparecer en la carpeta del proyecto. Ver figura 4.24
Una vez terminado el proceso anterior, generar un nuevo proyecto, asignarle un nombre y escribir el código en lenguaje C. Una vez escrito el código, click derecho sobre el proyecto y click en "Build proyect", también se debe generar el "BSP", una vez realizado todos estos pasos clik en "Run", corroborar que las direcciones de Timestamp coincidan para poder llevar a cabo la ejecución del programa ver figura 4.25, una vez que estas coincidan click en "Apply" y por último click en "Run Configuration" y con esto se ejecuta el programa.

El código implementado se muestra en el apéndice 7

En este código, el led0 parpadea a una frecuencia de 1 Hz ver figura 4.26. El led1 parpadea a una frecuencia de 5Hz, esto se muestra en la figura 4.27. Los leds del 2 al 6 hacen el corrimiento mostrado en la tabla 4.1, cada led tiene una duración de 10Hz en alto(prendido). Este resultado se puede observar en la figura 4.28

Por último el led7 parpadea a una frecuencia de 11Hz, ver figura 4.29. Todas estas tareas se realizan de manera simultanea, sin que ninguno de los procesos interfiera con otra, ver figura 4.30.

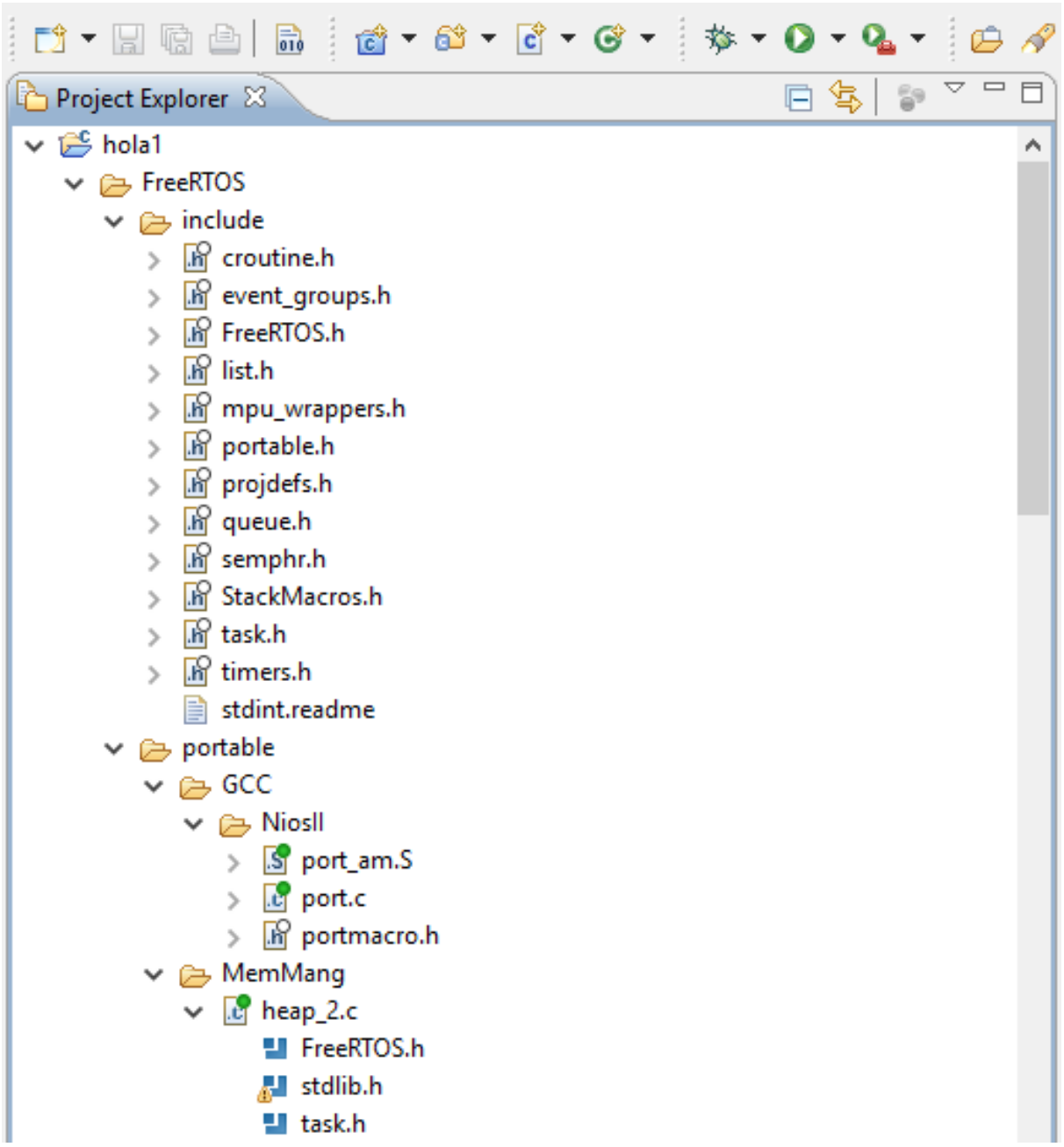

FIGURA 4.24: Vista de todos los archivos anexados al proyecto

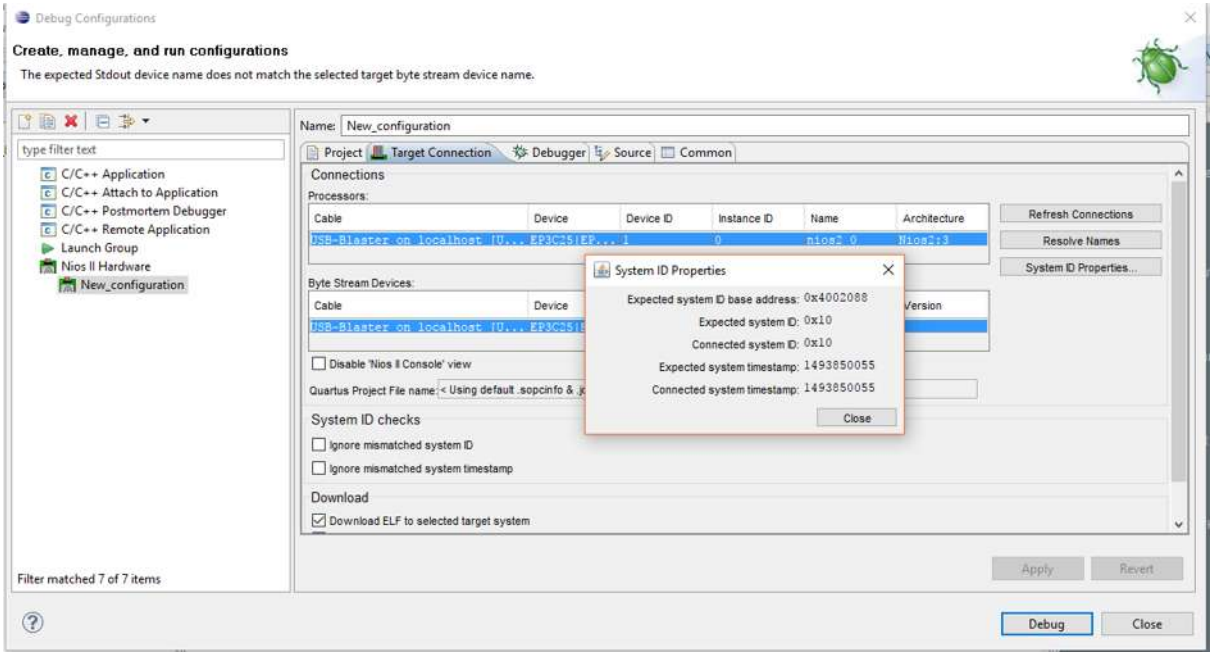

FIGURA 4.25: Ventana para configurar y verificar la dirección de memoria del programa

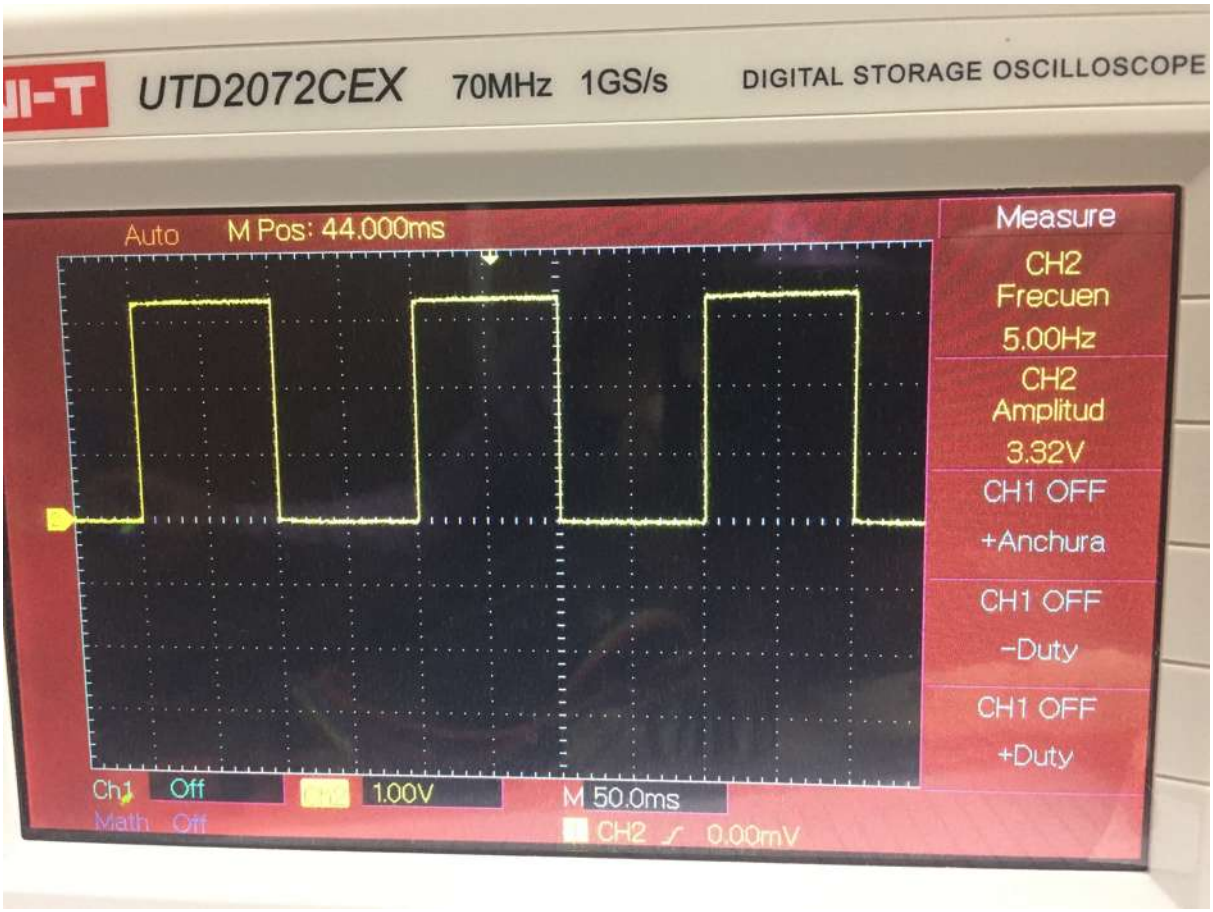

FIGURA 4.26: led 0

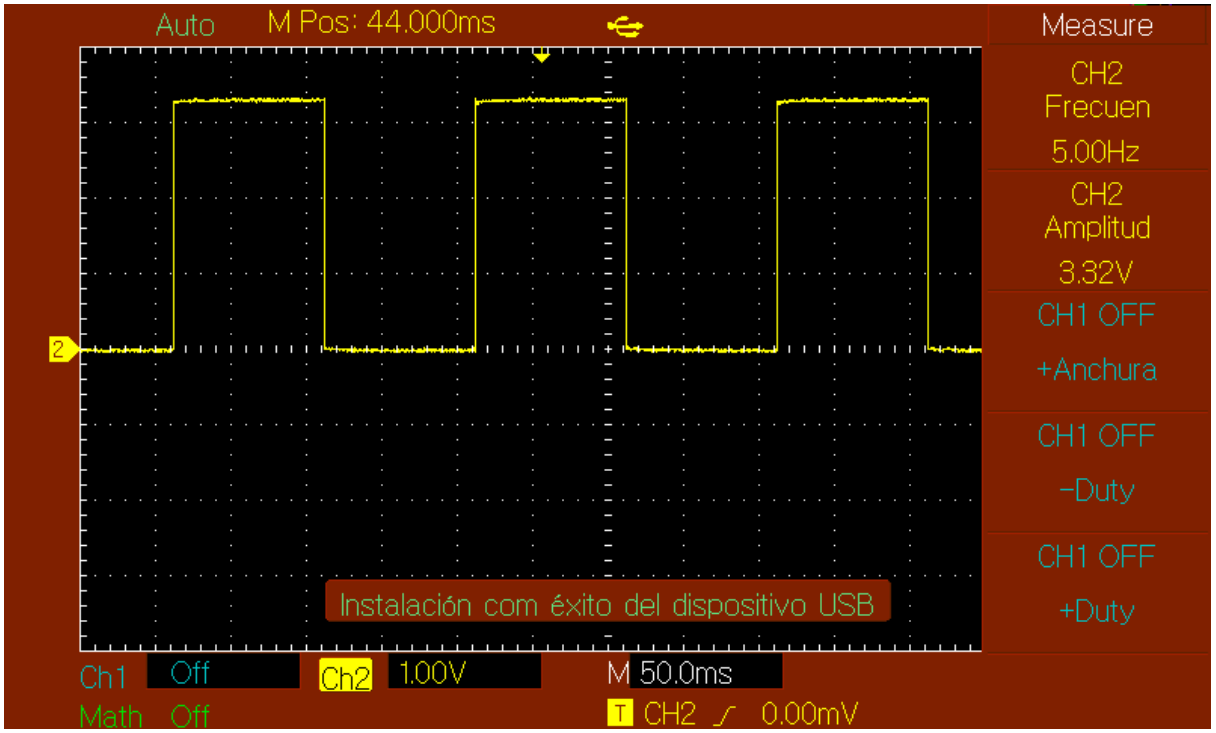

FIGURA 4.27: Led 1

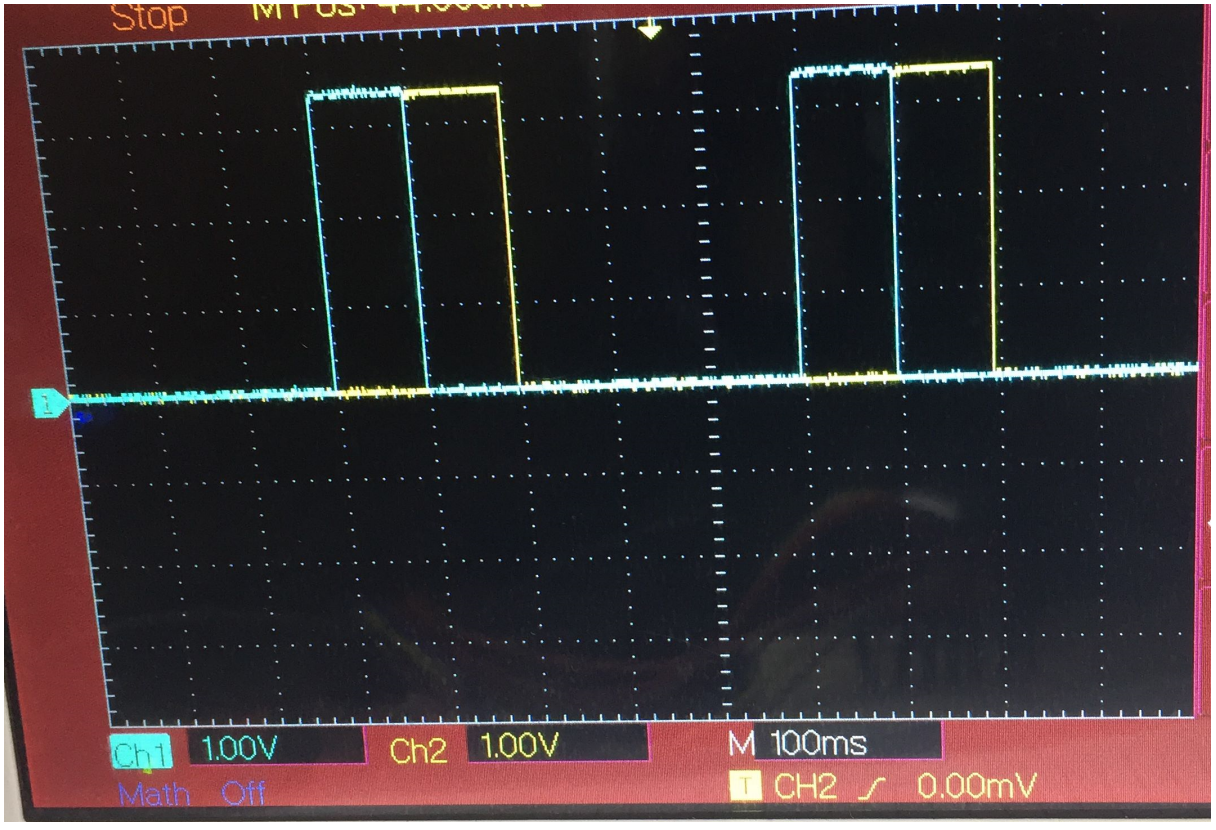

FIGURA 4.28: leds 2 al 6

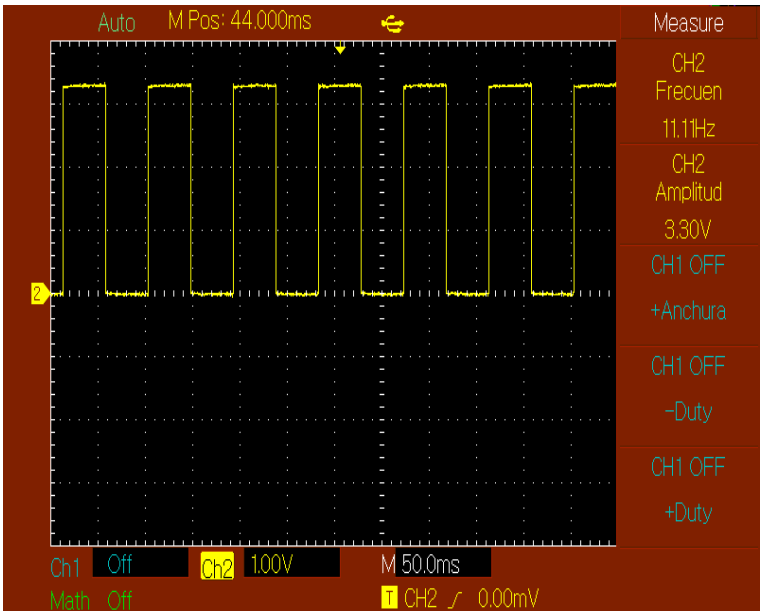

FIGURA 4.29: led 7

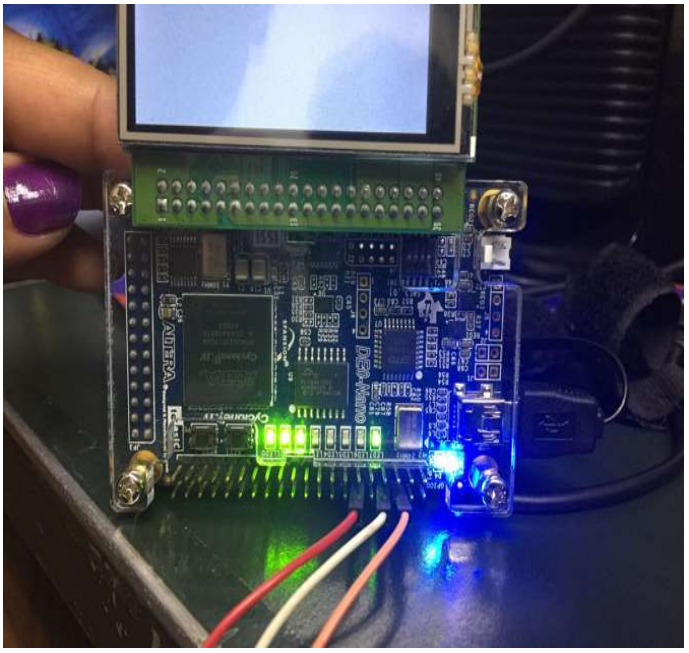

FIGURA 4.30: Vista de todos los leds

CAPÍTULO 5

Conclusiones

Después de todo el proceso de investigación y de pruebas se tienen las siguientes conclusiones:

- Se describieron las características del procesador NIOS II.
- Se documentaron las características y el funcionamiento de Qsys, al igual que el procedimiento para llevar a cabo la implementación de NIOS II en la FPGA Cyclone IV.
- Se documentaron los principios de los sistemas operativos de tiempo real, y después de investigar los tipos de RTOS existentes, se llega a la conclusión de usar FreeRTOS por sus características y por las facilidades que ofrece.
- Se implementó el sistema operativo de tiempo real FreeRTOS sobre el procesador NIOS II en la FPGA Cyclone IV.
- Se realizó la comparación para determinar las ventajas de diseño usando el esquema planteado contra la estrategia convencional que usa el microcontrolador MSP430F5529. En base a las pruebas realizadas se comprueba que realizar diseño usando FreeRTOS es más fácil y se tiene una mejor respuesta en el sistema.

5.1. Trabajos futuros

En este trabajo, lo que se hizo fue mediante frecuencias diferentes, prender y apagar leds, y de ahí ver la importancia de la priorización de tareas en un sistema muy crítico en cuestiones de tiempo y en el que ya no interviene la acción humana si no que estos dependen únicamente de la variable del tiempo, y de ahí observamos ventajas que esta nos ofrece en comparación con el microcontrolador MSP430F5529, esto debe ser un parteaguas para empezar a trabajar con este tipo de sistemas operativos en la Facultad de Ingeniería Eléctrica y más para los alumnos del

programa de Ingeniería electrónica ya que este campo no se ha explorado, aun falta mucho por hacer en este ámbito y se propone incluir prácticas con el marco de trabajo hardware-software aquí presentado dentro de las materias Electrónica Digital II, Microcontroladores I e incluso en el laboratorio de Control analógico, para desarrollar un sistema de control aplicando esta herramienta.

CAPÍTULO 6

Código implementado en CCS

```
#include <msp430.h>

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    P6DIR|=(BIT0+BIT1+BIT2+BIT3+BIT4+BIT5+BIT6);
    P3DIR|=BIT2;
    P3OUT&=~BIT2;
    // Colocar los pines del puerto 6 como salidas
        P6OUT&=~( BIT0 + BIT1 + BIT2 + BIT3 + BIT4 + BIT5 + BIT6 );
        // Poner todas las salidas en bajo
        unsigned int contador_led0 =500;
        unsigned int contador_led1 =100;
        // Declaraci ón de los contadores
        unsigned int contador_led2_6;
        unsigned int contador_ledex =45;
        contador_led2_6=1;
        while (1){
        // Esta parte del có digo se encarga de que el led 0
        // parpadee a una frec . de 1Hz
        contador_led0 --;
        if( contador_led0 <=0){
        P6OUT ^= BIT0 ;
        contador_led0 =500;
        }
            // Esta parte del có digo se encarga de que el led 0
                // parpadee a una frec . de 5Hz
                contador_led1 --;
                if( contador_led0 <=0){
                P6OUT ^= BIT0 ;
```

```
            contador_led0 =500;
            }
   //Esta parte del có digo se encarga de que los leds 1-5
   //hagan el corrimiento a una frec de 10 Hz
      contador_led2_6--;
  if( contador_led2_6 <= 0){
   if( P6OUT & 0x40 ){
   P6OUT &=~ BIT6 ;
 P6OUT=P6OUT|BIT2;
  }
    else if(P6OUT==0x00){
        P6OUT |=BIT2;
    }
  else {
  P6OUT = P6OUT <<1;
  }
  contador_led2_6 =2;
  }
  // Esta parte del có digo se encarga de que el
  // led 6 parpadee a una frec . de 11 Hz
  contador_ledex --;
  if( contador_ledex <=0){
  P3OUT |= BIT2 ;
  contador_ledex =45;
  }
  __delay_cycles (1000);
  }
  }
```

CAPÍTULO 7

Código implementado en Eclipse

```
#include<stddef.h>
#include<stdio.h>
#include<string.h>
#include<alt_types.h>
#include<unistd.h>

#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"

#include "serial.h"
#include <system.h>
#include <io.h>

#define LED_0_TASK_PRIORITY (tskIDLE_PRIORITY +1)
#define LED_1_TASK_PRIORITY (tskIDLE_PRIORITY +2)
#define LED_2_6_TASK_PRIORITY (tskIDLE_PRIORITY +3)
#define LED_7_TASK_PRIORITY (tskIDLE_PRIORITY +4)

#define LED_SET(x)\
{\
        unsigned char in_led = IORD(LED_BASE,0);\
        unsigned char out_tmp = 1<<x;\
        unsigned char out = in_led|out_tmp;\
         IOWR(LED_BASE,0,out);\
}

#define READ_LED(x)((IORD(LED_BASE,0)&(1<<x)) >> x)
#define READ_LED_PORT (IORD(LED_BASE,0))
```

```
#define WRITE_LED_PORT(x) (IOWR(LED_BASE,0,x))

#define TOGGLE_LED(x)\
{\
        unsigned char in_led = IORD(LED_BASE,0);\
        unsigned char out_tmp = 1<<x;\
        unsigned char out = in_led ^ out_tmp;\
        IOWR(LED_BASE,0,out);\
}

#define READ_SW(x) ((IORD(SW_BASE,0)&(1<<x))>>x)
#define READ_SW_PORT (IORD(SW_BASE,0))

#define READ_KEY(x) ((IORD(KEY_BASE,0)&(1<<x))>>x)
#define READ_KEY_PORT (IORD(KEY_BASE,0))


void  Tarea_LED0(void *pvParameters)
{
        for(;;)
        {
                TOGGLE_LED(0);
                vTaskDelay (500/portTICK_RATE_MS);
        }
        vTaskDelete(NULL);
}

void Tarea_LED1(void *pvParameters)
{
        for(;;)
        {
                TOGGLE_LED(1);
                vTaskDelay (100/portTICK_RATE_MS);
        }
        vTaskDelete(NULL);
}

void Tarea_LED_2_6(void *pvParameters)
{
        unsigned char led_2_6;
        unsigned char temp_led;
```

```
        led_2_6 = 1;
        for(;;)
        {

                if(led_2_6 & 0x10){
                        led_2_6 = 1;
                temp_led = READ_LED_PORT & 0x83;
                        temp_led = temp_led | (led_2_6 << 2);
                        WRITE_LED_PORT(temp_led);
                        vTaskDelay (100/portTICK_RATE_MS);
                }
                led_2_6 = led_2_6<<1;

                temp_led = READ_LED_PORT & 0x83;
                temp_led = temp_led | (led_2_6 << 2);
                WRITE_LED_PORT(temp_led);
                vTaskDelay (100/portTICK_RATE_MS);

        }
        vTaskDelete(NULL);
}

void Tarea_LED7(void *pvParameters)
{
        for(;;)
        {
                TOGGLE_LED(7);
                vTaskDelay (45/portTICK_RATE_MS);
        }
        vTaskDelete(NULL);
}


void SetupHardware()
{
        WRITE_LED_PORT(0);
}

int main()
{

        SetupHardware();
```

```
        (void) xTaskCreate(
                        Tarea_LED0,
                         (signed portCHAR *) "Tarea_LED0",
                         configMINIMAL_STACK_SIZE,
                         (void *) 0,
                         LED_0_TASK_PRIORITY,
                         (xTaskHandle *) NULL
                                 );
        (void) xTaskCreate(
                        Tarea_LED1,
                        (signed portCHAR *) "Tarea_LED1",
                         configMINIMAL_STACK_SIZE,
                         (void *) 0,
                         LED_1_TASK_PRIORITY,
                         (xTaskHandle *) NULL
                                 );
        (void) xTaskCreate(
                        Tarea_LED_2_6,
                         (signed portCHAR *) "Tarea_LED2_6",
                         configMINIMAL_STACK_SIZE,
                         (void *) 0,
                         LED_2_6_TASK_PRIORITY,
                         (xTaskHandle *) NULL
                                 );

        (void) xTaskCreate(
                        Tarea_LED7,
                        (signed portCHAR *) "Tarea_LED7",
                         configMINIMAL_STACK_SIZE,
                         (void *) 0,
                         LED_7_TASK_PRIORITY,
                         (xTaskHandle *) NULL
                                 );
        vTaskStartScheduler();

        return 0;
}
```

CAPÍTULO 8

Código resultante después de agregar los elementos necesarios en la plantilla Deo Nano SystemBuilder

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module nios_base(

        ////////////// CLOCK //////////
        CLOCK_50,

        ////////////// LED //////////
        LED,

        ////////////// KEY //////////
        KEY,

        ////////////// SW //////////
        SW,

        ////////////// SDRAM //////////
        DRAM_ADDR,
        DRAM_BA,
        DRAM_CAS_N,
        DRAM_CKE,
        DRAM_CLK,
```

```
        DRAM_CS_N,
        DRAM_DQ,
        DRAM_DQM,
        DRAM_RAS_N,
        DRAM_WE_N,

        ////////////// EPCS //////////
        EPCS_ASDO,
        EPCS_DATA0,
        EPCS_DCLK,
        EPCS_NCSO,

        ////////////// Accelerometer and EEPROM //////////
        G_SENSOR_CS_N,
        G_SENSOR_INT,
        I2C_SCLK,
        I2C_SDAT,

        ////////////// ADC //////////
        ADC_CS_N,
        ADC_SADDR,
        ADC_SCLK,
        ADC_SDAT,

        ////////////// 2x13 GPIO Header //////////
        GPIO_2,
        GPIO_2_IN,

        ////////////// GPIO_0, GPIO_0 connect to GPIO Default //////////
        GPIO,
        GPIO_IN
);

//=======================================================
//  PARAMETER declarations
//=======================================================


//=======================================================
//  PORT declarations
//=======================================================
```

```
////////////// CLOCK //////////
input                                   CLOCK_50;

////////////// LED //////////
output                [7:0]             LED;

////////////// KEY //////////
input                 [1:0]             KEY;

////////////// SW //////////
input                 [3:0]             SW;

////////////// SDRAM //////////
output               [12:0]             DRAM_ADDR;
output                [1:0]             DRAM_BA;
output                                  DRAM_CAS_N;
output                                  DRAM_CKE;
output                                  DRAM_CLK;
output                                  DRAM_CS_N;
inout                [15:0]             DRAM_DQ;
output                [1:0]             DRAM_DQM;
output                                  DRAM_RAS_N;
output                                  DRAM_WE_N;

////////////// EPCS //////////
output                                  EPCS_ASDO;
input                                   EPCS_DATA0;
output                                  EPCS_DCLK;
output                                  EPCS_NCSO;

////////////// Accelerometer and EEPROM //////////
output                                  G_SENSOR_CS_N;
input                                   G_SENSOR_INT;
output                                  I2C_SCLK;
inout                                   I2C_SDAT;

////////////// ADC //////////
output                                  ADC_CS_N;
output                                  ADC_SADDR;
output                                  ADC_SCLK;
input                                   ADC_SDAT;
```

```
///////////// 2x13 GPIO Header //////////
inout               [12:0]              GPIO_2;
input                [2:0]              GPIO_2_IN;

///////////// GPIO_0, GPIO_0 connect to GPIO Default //////////
inout               [33:0]              GPIO;
input                [1:0]              GPIO_IN;


//=======================================================
//  REG/WIRE declarations
//=======================================================




//=======================================================
//  Structural coding
//=======================================================


    niosII u0 (
        .clk_clk       (CLOCK_50),        //     clk.clk
        .reset_reset_n (1'b1), //   reset.reset_n
        .sdram_addr    (DRAM_ADDR),    //   sdram.addr
        .sdram_ba      (DRAM_BA),      //         .ba
        .sdram_cas_n   (DRAM_CAS_N),   //         .cas_n
        .sdram_cke     (DRAM_CKE),     //         .cke
        .sdram_cs_n    (DRAM_CS_N),    //         .cs_n
        .sdram_dq      (DRAM_DQ),      //         .dq
        .sdram_dqm     (DRAM_DQM),     //         .dqm
        .sdram_ras_n   (DRAM_RAS_N),   //         .ras_n
        .sdram_we_n    (DRAM_WE_N),    //         .we_n
        .epcs_dclk     (EPCS_DCLK),     //    epcs.dclk
        .epcs_sce      (EPCS_NCSO),      //        .sce
        .epcs_sdo      (EPCS_ASDO),      //        .sdo
        .epcs_data0    (EPCS_DATA),    //        .data0
        .led_export    (LED),    //     led.export
        .sw_export     (SW),     //      sw.export
        .ram_clk_clk   (DRAM_CLK),   // ram_clk.clk
        .lcd_RS        (GPIO[10]),        //     lcd.RS
        .lcd_RW        (GPIO[9]),        //        .RW
```

```
        .lcd_data       (GPIO[8:1]),       //          .data
        .lcd_E          (GPIO[0])           //          .E
    );

endmodule
```

Bibliografía

[1] Introducción a los sistemas operativos. https://www.fing.edu.uy/tecnoinf/mvd/cursos/so/material/teo/so01-introduccion.pdf. consultado:20-09-2017.

[2] M. Morris Mano. *Lógica digital y diseño de computadoras.* Prentice Hall, 1982.

[3] M. Morris Mano. *Arquitectura de Computadoras.* Prentice Hall, 1994.

[4] William Stallings. *Organización y arquitectura de computadores.* Prenticee Hall, 1997.

[5] Zvonko Vranesic Stephen Brown. *Fundamentals of Digital Logic with Verilog Dessign.* McGraw-Hill.

[6] Introdución a los microcontroladores. http://www.cec.uchile.cl/ mcarter/EL54B/Informe consultado:10-10-2017.

[7] Andrew S. Tanenbaum. *Sistemas Operativos Modernos.* Pearson, 1993.

[8] Andy Wellings Alan Burns. *Sistemas de tiempo real y lenguajes de programación.* Pearson educacion, 2003.

[9] Sistemas operativos de tiempo real. http://jeuazarru.com/wp-content/ uploads/2015/11/RTOS1.pdf. consultado:08-05-2017.

[10] Sistemas operativos de tiempo real (rtos). http://sistemas.unla.edu.ar/sistemas/sls/ ls-4-sistemas-operativos/pdf/SO-L-RTOS.pdf, . consultado:15-05-2017.

[11] Sistemas operativos en tiempo real. http://isa.uniovi.es/docencia/ TiempoReal/Recursos/temas/sotr.pdf. consultado:02-05-2017.

[12] Mastering the freertos real time kernel. http://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf, . consultado:02-05-2017.

[13] Richard Barry. *Using the freeRTOS real time kernel.* Virtual, 2009.

[14] Creating a system with qsys. https://www.altera.com/en_US/pdfs/ literature/hb/qts/qsys_intro.pdf. consultado:02-05-2017.

[15] Download. http://www.freertos.org/FreeRTOS-Labs/RTOS _labs_download.html. consultado:20-09-2017.