Proyecto Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Implementación del microcontrolador 8051 mediante FPGA

Autor: Esteban Castro Polvillo
Tutor: Hipólito Guzmán Miranda

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Grado
Grado en Ingeniería Electrónica,
Robótica y Mecatrónica

Implementación del microcontrolador 8051 mediante FPGA

Autor:
Esteban Castro Polvillo

Tutor:
Hipólito Guzmán Miranda
Profesor Contratado Doctor

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Grado: Implementación del microcontrolador 8051 mediante FPGA

Autor: Esteban Castro Polvillo

Tutor: Hipólito Guzmán Miranda

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mis padres y hermana

A mis amigos y familia

Agradecimientos

En primer lugar, agradecer al doctor Hipólito Guzmán Miranda su ayuda durante este proyecto desde principio a fin. Darle las gracias por su paciencia, dedicación y enseñanza durante todos estos años, especialmente en la recta final.

También quería dar las gracias a los trabajadores de la oficina técnica de la empresa AND&OR por su confianza, paciencia y formación.

No podía haber logrado todos mis triunfos sin el gran apoyo de todos mis amigos, especialmente los de David, Diego y Manuel. Aquellos que supieron darme su apoyo en los momentos más duros.

Agraceder a mi padre Esteban, mi madre María del Carmen y mi hermana Beatriz por su enorme apoyo durante todos estos años. Son incontables los momentos en que quería tirar la toalla y estaban ahí para levantarme y hacer que me esforzara cada día más por ellos.

Por último, agradecer a mi familia: a los que están y los que se fueron por haberme hecho llegar donde estoy ahora.

A todos aquellos que han contribuido de una manera u otra a llegar mi meta.

Gracias.

Esteban Castro Polvillo

Sevilla, 2018

Resumen

En este proyecto se ha desarrollado la implementación de un microcontrolador 8051 mediante el uso de FPGA (Field-Programmable Gate Array) en lenguaje VHDL. Mediante esta metodología se podrá utilizar este dispositivo, el cual ya ha cesado su fabricación, mediante el lenguaje VHDL. De esta forma se ofrece una alternativa a la obsolescencia del microcontrolador.

El primer objetivo del proyecto es el de encontrar y configurar un IP Core válido para este microcontrolador entre las alternativas que existen. El segundo objetivo es el de crear programas basados en lenguaje C y ensamblador de modo que el microcontrolador lo interprete de manera idónea. Para verificar que el proyecto se ha realizado correctamente, se inicializará la memoria interna ROM del microcontrolador con esos programas creados. Por último, se tomará un ejemplo de programa real para comprobar si la solución elegida es la correcta.

Abstract

In this project the implementation of an 8051 microcontroller has been developed through the use of FPGA (Field-Programmable Gate Array) in VHDL language. With this methodology this device can be used through the VHDL language. In this way, an alternative to the obsolescence of the microcontroller is offered.

The first objective of the project is to find and configure a valid IP Core for this microcontroller among the alternatives that exist. The second goal is to create programs in C and assembler language so that the microcontroller interprets it in an ideal way. To verify that the project has been carried out correctly, the ROM internal memory of the microcontroller will be initialized with those created programs. Finally, an example of a real program is taken to verify if the chosen solution is the correct one.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Cuando las cosas están en peligro alguien tiene que renunciar
a ellas, perderlas para que otros las conserven.

- Frodo Bolsón -

En la actualidad, el auge de la industria electrónica crea la necesidad de diseñar circuitos con mejores características, centrándose en la miniaturización de los componentes y con especial énfasis en la duplicidad del sistema diseñado. Al definir el propio usuario la lógica, se permiten individualizar a los sistemas y centrarse más en las especificaciones del destinatario.

Debido a este mercado, los fabricantes se han encontrado con la necesidad de satisfacer esa demanda mediante la fabricación de circuitos cada vez más potentes y, a la vez más pequeños, que pueda desempeñar todas las funcionalidades requeridas. En numerosas aplicaciones la adquisición, tratamiento y transmisión de datos, junto con la complejidad de los circuitos o el número de componentes hace que sea necesario un sistema más complejo. Es ahí donde aparecen los microcontroladores.

Un microcontrolador es un circuito integrado compuesto por una CPU, memoria(s) y puertos de entrada y de salida principalmente. Suele ir acompañado de un puerto serie, contadores, convertidores analógico-digital, timers, etc. Supone el componente principal de todo sistema embebido en el que controla los elementos de entrada y salida. Su principal función es la de procesar información y automatizar procesos.

Este proyecto se va a centrar en el microcontrolador 8051 debido al problema de obsolescencia que posee este dispositivo. Fue creado en el año 1980 por la empresa *Intel* y pronto se convirtió en un dispositivo muy popular. La fabricación de este componente cesó en marzo del año 2007.

Fue entonces cuando la empresa *AND&OR* me presentó el problema con el que se encontraron hace años. Esta empresa se dedica a la automatización de todo tipo de envasado y este microcontrolador es el principal componente de numerosas máquinas que fueron creadas desde los años 80. Tras encontrar ese vacío en el mercado, decidieron solventar ese problema mediante el uso de PLCs, pero aun existían un gran número de máquinas automatizadas que necesitaban ser reemplazadas o reconfiguradas con el microcontrolador 8051.

El problema se encuentra en la actualidad, ya que esas máquinas necesitan ser reemplazadas debido al paso del tiempo que afecta a estos micros y, al no haber mercado que ofrezca una solución alternativa, deben sustituir gran parte de la máquina y con el elevado coste que supone.

Es en este momento donde aparece en escena la FPGA (Field-Programmable Gate Array). Este dispositivo programable es capaz de reproducir cualquier circuito usando la lógica programable. Sus características

principales son las de configurabilidad y reprogramabilidad. Esto supone un coste de desarrollo y adquisición mucho menor para pequeñas cantidades de dispositivos, además del tiempo de desarrollo necesario.

De esta manera se va a proponer el uso de FPGAs como solución a este problema de obsolescencia del microcontrolador 8051. Para ello se va a establecer una metodología con la que sintetizar un diseño de este dispositivo mediante la herramienta de la empresa *Xilinx* llamada *ISE Design Suite*. También se comprobará su funcionamiento a través de diferentes herramientas de programación. Finalmente se adaptará a las especificaciones necesarias para que pueda ser una solución al problema actual de muchos fabricantes que empleaban este microcontrolador.

2 ESTADO DEL ARTE

Sólo tú puedes decidir qué hacer con el tiempo que se te ha dado.

- Dama Galadriel -

2.1. Obsolescencia electrónica.

En la actualidad, la electrónica está presente en el día a día de la sociedad, desde los ya conocidos smartphones hasta los automóviles. Debido a este gran auge de la tecnología, los fabricantes se han encontrado con la necesidad de saciar esa demanda con menores costes y dispositivos más potentes. Esto trae consigo la sustitución de una tecnología por una mejor, generando a su vez un elevado volumen de basura electrónica.

Este comportamiento se puede dar también por la imposibilidad de que el propio consumidor lo arregle en caso de avería o bien por la decisión de la propia industria que acaba imponiendo la limitación de las acciones del dispositivo. Por ejemplo, los agricultores no pueden arreglar la electrónica de su maquinaria debido a los fabricantes han blindado estos dispositivos contra la piratería, provocando a su vez que los propios compradores no sean los dueños.

Es por ello por lo que en Estados Unidos se están planteando obligar a las empresas a publicar manuales de reparación y a facilitar herramientas para diagnosticar problemas. O el caso de Francia, donde una ley obliga a las empresas a avisar de cuanto tardarían potenciales reparaciones del producto en cuestión, para evitar situaciones como las que obligan a comprar un nuevo producto debido a que las piezas de recambio tardan un tiempo excesivo en llegar.

Ilustración 1. obsolescencia programada

2.2. Historia del microcontrolador 8051.

A pesar de su desuso hoy en día, el **8051** o **MCS-51** ha sido uno de los microcontroladores más usados en los sistemas de control industrial. Fue desarrollado por la empresa *Intel* en 1980 para uso en productos embebidos y está basado en la Arquitectura Harvard.

Esta arquitectura se caracteriza por tener pistas de señal y de almacenamiento (buses) físicamente separadas para las instrucciones y los datos. De esta manera se puede direccionar tanto la memoria RAM como la ROM mediante líneas separadas para programa y para datos.

Otra caracterísitca que ha hecho destacar a este dispositivo es que tiene incluido una unidad de proceso booleano para que operaciones de nivel de bit lógica booleana se ejecuten directamente en registros internos de manera eficiente.

Su predecesor fue el **8048**, empleado en el teclado del primer PC de la empresa *IBM*. Convertía la pulsación de las teclas del teclado en una secuencia de datos en serie que se enviaba al ordenador. Hoy en día, se usan versiones modernas de este microcontrolador para los teclados de modelo básico.

Tras el éxito del 8051, Intel desarrolló el **8052**. Éste era una versión mejorada de su antecesor con el doble de memoria RAM, ROM y con un tercer temporizador de 16 bits.

A partir de este último microcontrolador, se lanzaron al mercado otros muchos derivados de estas versiones. Es el caso del **8751**, el cual era una versión mejorada del 8051 que incluía una EPROM. Esta versión ofrecía otra versión en un encapsulado de cerámica con una ventana de cuarzo transparente sobre la parte superior de la matriz para que la luz ultravioleta borrara la memoria EPROM. De esta forma se convirtió en uno de los primeros microcontroladores con la posibilidad de ser programado varias veces.

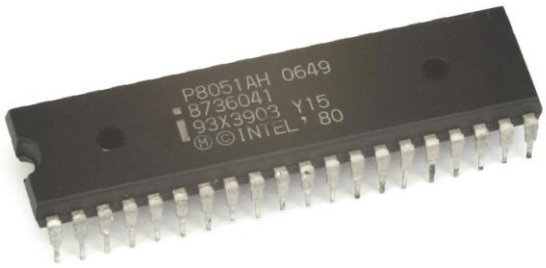


Ilustración 2. Intel 8051. Recuperado de https://es.wikipedia.org/wiki/Intel_8051

2.3. Microcontroladores del siglo XXI.

La mayoría de los productos que se fabricarán en un futuro poseerán un computador integrado que le darán "inteligencia" y dará lugar a mejor intercomunicación y control global. Los microcontroladores han hecho posible esta idea mediante la inclusión de una computadora en el circuito integrado.

Dentro de éstos podemos encontrar: procesador, memoria de datos, memoria de programa, puertos de entrada y de salida, convertidores analógico-digital y digital-analógico, canales de comunicación, y un conjunto de recursos que hace posible controlar correctamente los periféricos.

Los microcontroladores están en contínuo desarrollo. Actualmente, ya encontramos en el mercado microchips con micros integrados que posean numerosos periféricos y un reducido número de pines. Éste es el caso de la familia **PIC16(L)F161X**, microcontroladores que amplían la oferta de periféricos independientes del núcleo y que están diseñados para reducir la latencia de interrupción y consumo de energía además de aumentar la eficiencia del sistema.

Ilustración 3. PIC16F161X. Recuperado de https://www.electronicspecifier.com/micros/microchip-exhibit-peripheral-rich-mcus

En un futuro no muy lejano, podríamos encontrarnos con dispositivos con capacidad de computación cuántica. Éstos, a diferencia del uso de bits, usaría los cúbits (o *quantum bits*) con el que se revolucionaría la tecnología tal y como la conocemos hoy día.

La principal característica que tienen los cúbits es que todas las partículas atómicas pueden estar en varios estados a la vez. Si en un bit encontramos 2 valores (0 ó 1), en un cúbit hay 4 valores ([0 ó 1] y [0 ó 1]). Según un artículo de la revista FayerWayer asegura que "*con estas características se cree que un computador cuántico con 600 qubits podría realizar cálculos que contengan la información de* ***todos los átomos del universo en segundos*** *(podría procesar 2600 resultados)"* (p. 6).

> Gutierrez, Gary. *Qubit, la unidad fundamental del futuro informático y tecnológico*: FayerWayer (2013).

Pero hay una segunda característica más interesante para los microcontroladores y es que cada proceso es independiente. Esto quiere decir que, mientras en la computación clásica la resolución de problemas es lineal, la computación cuántica puede resolver más de una operación al mismo tiempo.

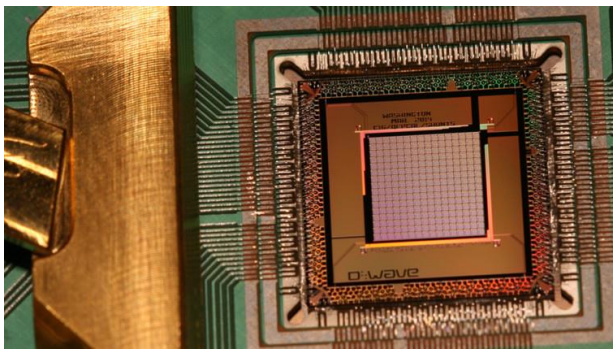


Ilustración 4. Procesador cuántico. Recuperado de http://www.redusers.com/noticias/google-y-la-nasa-trabajan-en-una-nueva-computadora-cuantica

2.4. Cómo afecta el uso de FPGA hoy en día.

Debido al estancamiento de la ley de Moore, las empresas no pueden mejorar la velocidad de sus algoritmos mediante el contínuo incremento de potencia de los procesadores. Es lo que ha llevado a empresas como *Microsoft* por apostar por esta tecnología, incluyendo una FPGA en todos sus servidores. Esto ha provocado una mejora de velocidad del 40% de sus algoritmos de búsqueda.

Hoy en día, se está popularizando su uso en la industria electrónica. El problema reside en que hay pocas FPGAs disponibles que sean compatibles con herramientas *Open Source* y las que están disponibles son

pequeñas y poco potentes.

Es posible que, si se siguen desarrollando su uso y no se apuesta por la dirección hacia el *Open Source*, hay gran riesgo de que se quede en una burbuja tecnológica.

La pregunta es: ¿seguirá teniendo utilidad o son una moda pasajera? Frecuentemente se usan como dispositivos para facilitar el diseño y prototipo de *ASIC* (prototipo integrado para aplicaciones específicas), aunque se está ampliando su uso desde aplicaciones con cálculos pesados a versiones ligeras para dispositivos móviles.

En cuanto al estado actual de las FPGAs, existen modelos como la serie *Zynq* de *Xilinx*, la cual integra la programación de software de un procesador con la programabilidad hardware de una FPGA. Estos dispositivos habilitan una aceleración de hardware a fin de obtener un software con el que reconfigurar el hardware, creando o deshaciendo componentes (procesadores, memorias, etc) dependiendo de las necesidades.

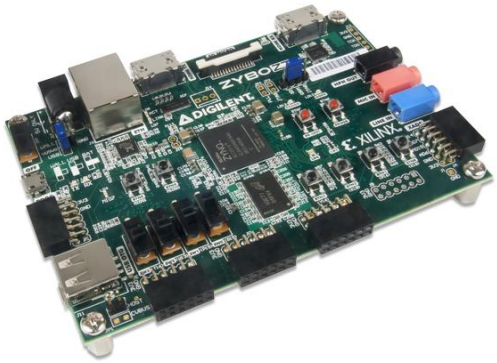


Ilustración 5. Zybo Z7 Zynq-7000. Recuperado de: https://store.digilentinc.com/zybo-z7-zynq-7000-arm-fpga-soc-development-board/

3 ALCANDE DEL TRABAJO

Los héroes desempeñan siempre un papel pequeño en las grandes hazañas.

- Gandalf -

3.1. Requisitos de funcionalidad

1. Adaptación del IP Core elegido para el microcontrolador 8051.
2. Elegir un toolchain que se adapte a las necesidades del proyecto.
3. Inicializar la memoria del microcontrolador a partir de un programa.
4. Correcta síntesis del proyecto en Xilinx.

3.2. Requisitos de prestaciones

1. Crear un programa en ensamblador con el que inicializar la memoria ROM del microcontrolador.
2. Crear un programa en lenguaje C con el que inicializar la memoria ROM del microcontrolador.
3. Transformar el programa generado en un archivo coe.
4. Simular en ISIM el programa en ensamblador para que realice las operaciones para las que ha sido programado.
5. Simular en ISIM el programa en lenguaje C para que realice las operaciones para las que ha sido programado.

3.3. Requisitos de diseño

1. Implementación del microcontrolador mediante un FPGA
2. Elegir un IP Core en el que estén implementadas las interrupciones del microcontrolador 8051.

3.4. Requisitos de operación

1. Utilizar un programa real de la empresa AND&OR para comprobar si se ha solucionado su problema de fin de ciclo de vida.
2. Prueba de un programa en una FPGA real.
3. Construcción de un pequeño prototipo para exponer la FPGA a problemas reales a los que estaría sometida.
4. Integración de la FPGA en el circuito que posea la máquina.
5. Comunicación con el microcontrolador mediante Ethernet o puerto serie.

3.5. Restricciones

1. Presupuesto lo más reducido posible.
2. El tiempo del proyecto no debe llegar hasta finales del año 2018 por la inexistencia de micros 8051 en el mercado.

4 PREPARACIÓN DEL DISEÑO

No dejéis que vuestras cabezas se vuelvan más grandes que vuestros sombreros.

- Bilbo Bolsón -

El objetivo de este apartado es el de establecer la metodología de trabajo que se va a desarrollar para la realización del proyecto. También se incluyen los tipos de *IP Cores* que podemos encontrar hoy día como alternativa al microcontrolador 8051. Por último, se expondrán los *toolchain* disponibles necesarios para su programabilidad.

4.1. Metodología de trabajo

1. Búsqueda exhaustiva de las distintas versiones de *IP Cores* disponibles para el 8051.
2. Elección de una versión que cumpla los requisitos propuestos.
3. Configuración de las memorias del microcontrolador usando *Core Generator*.
4. Implementación correcta del diseño del proyecto.
5. Creación de un programa mediante el *toolchain* elegido.
6. Generación del programa en formato ".HEX".
7. Generación del programa en formato ".coe".
8. Inicialización de la memoria del microcontrolador con el archivo creado para comprobar su correcto funcionamiento.
9. Comprobar los resultados obtenidos mediante simulaciones.

4.2. IP Cores

4.1.1 Oregano 8051

Es el *IP Core* más común, desarrollado en cooperación con la Universidad Tecnológica de Viena. Es compatible con el procesador de *Intel* 8051 y su circuito sintetizable está disponible en VHDL.

Ofrece una ejecución de programa más rápida que el original debido a la arquitectura mejorada del procesador. Posee licencia **LGPL**, lo que proporciona un uso sin coste y no es necesario liberar el código de los periféricos extra que se puedan añadir.

Ilustración 6. Logotipo Oregano Systems

4.1.2 Open cores T51

Core compatible con el microcontrolador 8051 y 8052. Incluye una utilidad para crear ROM en VHDL. La característica de esta versión es que usa el bus **Wishbone**, utilizado en numerosos cores de opencores y brinda la posibilidad a incluir periféricos si fuera necesario.

Ilustración 7. Logotipo Open cores

4.1.3 Open cores 8051 core

IP Core que incluye temporizadores, contadores y 128 bytes de memoria de datos y hasta 4 Kbytes de memoria de programa. Es quizás uno de los menos usados debido a su estado "*Alpha*" y desactualizado.

4.1.4 Modelo sintetizable de la Universidad de Dalton

Esta versión desarrollada por la Universidad de Dalton (California) posee un diseño implementado y sintetizable en VHDL del modelo 8051. Es 100% compatible con las instrucciones del microcontrolador.

Sin embargo, se encuentra con algunas limitaciones. No es compatible "cycle-accurate", no tiene implementado interrupciones y no es posible actualmente la conexión con periféricos.

Ilustración 8. Logotipo Universidad de Dalton

4.1.5 Otros IP Cores

Como alternativa a los anteriores cores, podemos encontrar otros tantos:

- Evatronix 8051 microcontroller IP cores.
- E8051.
- Cast R8051XC.
- Digital Core Design DP8051 CPU.
- CorePool FHG8051.

Todas éstas son algunas de las opciones que podemos encontrar, pero todas ellas no poseen licencia libre. Por ello es necesario valorar si el coste de tener esa licencia es mayor que el de buscar otra alternativa a la obsolescencia del 8051 o si es posible, realizar un buen diseño partiendo de uno de los proyectos LGPL.

4.3. Toolchain

La elección de un buen software *toolchain* para desarrollar el micro es tan importante como la elección del *IP Core*. Debido a que el 8051 fue un dispositivo muy popular, encontramos varias opciones para desarrollar un buen programa con el que comprobar el funcionamiento del proyecto.

4.3.1 Keil uVision

Keil es uno de los *toolchain* más populares para el desarrollo del firmware de la arquitectura del 8051. Proporciona un compilador, depurador y un *IDE* completo llamado C51 para su programación.

El toochain es un software con licencia que se puede comprar desde su página web. La versión de evaluación limitada se puede descargar libremente de su web.

Este toolchain admite el desarrollo de software utilizando

- Embedded C.
- Lenguaje ensamblador 8051.

Ilustración 9. Logotipo Keil

4.3.2 Small Device C Compiler (SDCC)

El compilador **Small Device C** es un compilador C de código abierto (GPL) gratuito para microcontroladores basados en 8051. Consta de un *linker*, ensamblador, simulador y depurador para desarrollar software para la arquitectura 8051.

SDCC es completamente gratuito y no hay restricciones en el tamaño del código y admite el desarrollo de firmware utilizando *Embedded C*.

Consta de herramientas usando línea de comandos que luego puede integrar con IDE abiertos como "**Eclipse**" o "**Code :: Blocks**".

Ilustración 10. Logotipo SDCC

4.3.3 IAR Embedded Workbench for 8051

IAR embedded workbench es otro IDE de desarrollo popular para este microcontrolador. Proporciona una versión de prueba completa de 30 días o una versión de evaluación limitada para 8051.

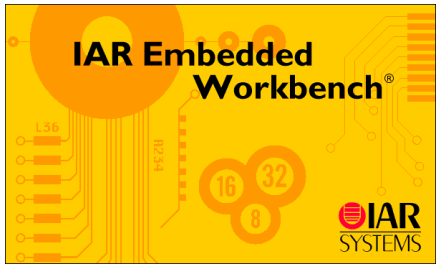


Ilustración 11. Logotipo IAR Embedded Workbench

4.3.4 MikroElectronica Compiler Collection

MikroElectronika es una compañía que proporciona depuradores y desarrollo de software IDE para 8051. Tienen una gama de compiladores para:

- BASIC.
- Pascal.
- Lenguaje C.

Ilustración 12. Logotipo MikroElectronica

4.3.5 MIDE-51

Software para herramientas de desarrollo para el 8051. Lenguaje soportado:

- Lenguaje C.
- Lenguaje ensamblador.

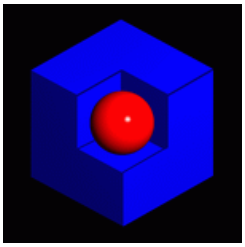


Ilustración 13. Logotipo MIDE-51

5 SOLUCIÓN PROPUESTA

No os diré no lloréis, pues no todas las lágrimas son amargas.

- Gandalf -

En este apartado se explica detalladamente el proceso desarrollado en este proyecto. Abarca desde la eleción del *IP Core* que mejor se adapte a la solución del problema de obsolescencia del microcontrolador 8051, hasta la correcta implementación en VHDL.

5.1. Características del 8051

La familia del micro **8051** es muy diversa. En ella encontramos el **8751**, microcontrolador con una EPROM interna con la capacidad de ser programada por el usuario, o el **8031** que se programa sin PROM ni EPROM.

En este caso, el dispositivo elegido es el 8051. Posee una ROM interna pre-programada por el fabricante basado en los microprocesadores de 8 bits.

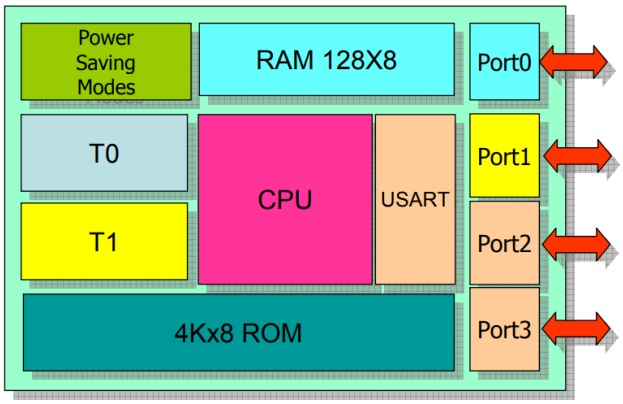


Ilustración 14. Diagrama de bloques 8051

Características del microcontrolador 8051: una CPU de 8 bits, 3 puertos de entrada/salida paralelos, un puerto serie, dos entradas para timer o contador de 16 bits, dos entradas para interrupciones externas, señales de RD (Read) y WR (Write) para la lectura o almacenamiento de datos externos contenidos en la RAM y una señal de PSEN para la toma de instrucciones localizadas en una EPROM externa.

Debido a estas últimas tres señales, el 8051 puede direccionar 64K de memoria de programa y 64K de datos separadamente, obteniendo un total de 128K.

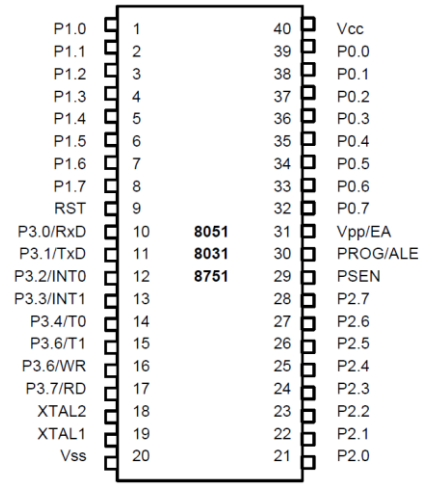


Ilustración 15. Puertos 8051/8031/8751

Características de las conexiones:

- **P0.0-P0.7**: Puerto bidireccional con salidas a colector abierto. Cuando el pin toma un valor lógico de "1" escritos, las salidas pueden servir como entradas en alta impedancia. También es usado para obtener el dato y la parte baja de la dirección.
- **P1.0-1.8**: Es un puerto bidireccional, cuando se escribe "1's" en el puerto, puede ser utilizado como entrada.
- **P2.-P2.7**: Pin bidireccional con fijadores de nivel internos. Cuando se activan todas las señales del puerto, las líneas pueden ser utilizadas como entradas o salidas. Como entradas, las líneas que son externamente colocadas en la posición baja proporcionaran una corriente hacia el exterior. El puerto 2 es utilizado además para direccionar memoria externa. Este puerto, emite el byte más alto de la dirección durante la búsqueda de datos en la memoria del programa externo y durante el acceso a memorias de datos externos que usan direccionamientos de 16 bits. Durante el acceso a una memoria de dato externa, que usa direcciones de 8 bits, el puerto dos emite el contenido del registro del correspondiente a este puerto, que se encuentra en el espacio de funciones especiales.
- **P3.0-3.7**: similar al puerto anterior, pero en éste las líneas externas colocadas en las líneas más bajas proporcionan corriente. Además, posee características únicas como son:
    - **RxD** (P3.0): Puerto serie de entrada.
    - **TxD** (P3.1): Puerto serie de salida.
    - **INT0** (P3.2): Interrupción externa.
    - **INT1** (P3.3): Interrupción externa.
    - **T0** (P3.4): Entrada externa timer0.
    - **T1** (P3.5): Entrada externa timer1.

  - **WR** (P3.6): Habilitador de escritura para memoria externa de datos.
  - **RD** (P3.7): Habilitador de lectura para memoria externa de datos.
- **RST**: Reset con entrada de alta impedancia que dura 2 ciclos de máquina que detiene el microcontrolador.
- **ALE** (*Address Latch Enable*): permite fijar el byte najo de la dirección durante el acceso a una memoria externa usando para elo un pulso positive de salida. Es emitido en un rango constante de 1/6 de frecuencia de reloj, por lo que suele tener uso de cronómetro.
- **PSEN** (*Program Store Enable*): Habilitador de lectura para memoria de programas externos. Se active 2 veces cada ciclo de reloj a excepción de cuando se desea accede a la memoria de datos externos.
- **EA** (*External Access Enable*): señal que debe mantenerse externamente en una posición baja para habilitar la elección del código de las localizaciones de la memoria de programas externos. Si son "unos" los que se escriben, se ejecutan programas contenidos en la ROM.
- **XTAL1**: Entrada del cristal para el generador de reloj interno. Amplifica e invierte la señal.
- **XTAL2**: Salida del amplificador oscilador inversor.

5.2. Elección del IP Core

En el apartado 4: preparación del diseño, se presentaron las posibles alternativas para diseñar el 8051. Entre todos las versiones disponibles, se rechazaron aquellas que no poseían licencia libre para ahorrar. Esto redujo bastante los IP Cores con los que obtener nuestro microprocesador, por lo que se decidió realizar la siguiente tabla comparativa con el fin de poder elegir uno de los proyectos que mejor se adaptara a los requisitos.

	Oregano 8051	**Open Cores T51**	**Open Cores 8051 Core**	**Universidad de Dalton**
RAM	256 bytes interna 64Kbytes externa	256 bytes interna	128 Kbytes	64 Kbytes externa
ROM	64Kbytes	64 Kbytes	64 Kbytes	64 Kbytes
Interrupciones	Sí	Sí	Sí	No
Periféricos implementados	Todos	Todos	Todos	No
Última actualización	Marzo 2010	Marzo 2009	Mayo 2009	Febrero 2001
Otros	Muy usado y recomendado en foros	Compatible con el 8052 y usa el bus Wishbone	No está "Proven"	No es cycle/timing compatible

Tabla 1. Tabla comparativa IP Cores

Tras la comparativa de todas las opciones posibles, se optó por la elección del *IP Core* "Oregano 8051", debido a que es la versión más reciente y que era más recomendado su uso como alternativa al antiguo microcontrolador.

El siguiente en ser un gran candidato era "Open Cores T51" debido a su compatibilidad con el 8052, en este caso no necesaria, y el uso del Wishbone, bus muy usado en cores de licencia libre para integrar parte de un circuito integrado para comunicar con otras dentro del mismo.

5.3. Oregano 8051

El **IP Core Oregano 8051** ofrece una descripción parametizable y sintetizable en VHDL del procesador 8051 de la empresa *Intel*. Esta versión además permite una ejecución de programa mucho más rápida que la del microcontrolador original. Se encuentra disponible de forma gratuita, incluso para aplicaciones industriales, bajo la licencia libre *LGPL*.

Ilustración 16. Logotipo Oregano Systems

Características:

- Diseño totalmente síncrono.
- Conjunto de instrucciones compatibles con el microcontrolador estándar 8051.
- La arquitectura optimizada permite de 1 a 4 ciclos de reloj por operación.
- 10 veces más rápido debido a su arquitectura.
- Número de temporizadores/contadores seleccionables por el usuario.
- Implementación opcional de MUL (*multiply command*) usando multiplicador con arquitectura paralela.
- Implementación opcional de DIV (*divide command*) usando divisor con arquitectura paralela.
- Implementación opcional del comando de ajuste decimal (DA).
- No hay puertos de entrada/salida multiplexado.
- 256 bytes de RAM interna.
- Más de 64 Kbytes de ROM y más de 64 Kbytes de ROM.
- Código disponible gratis bajo GNU GPL.
- Tecnología independiente, claramente estructurado, código VHDL bien comentado.
- Fácilmente expandible adaptando/cambiando el código VHDL.
- Parametizable via constantes de VHDL.

5.4. Síntesis del microcontrolador 8051

La implentación del microcontrolador 8051 se realiza mediante el software "ISE Design Suite 14.7" de la empresa *Xilinx*.

Ilustración 17. Logotipo ISE Design Suite

La última versión disponible de Oregano 8051 es la V1.6 y en ella encontramos la carpeta "vhdl" con los archivos necesarios para crear el proyecto.

A continuación, se enumeran los pasos seguidos para la implementación del circuito:

1) Creación de un nuevo proyecto: "**Proyecto_8051**".
2) Selección de dispositivo y diseño: se ha elegido el modelo **LX9 Microboard** de la familia **Spartan6** de FPGA.
3) Añadir los archivos "vhd" contenidos en el archivo del proyecto descargado que se incluyen en la carpeta con el mismo nombre.
4) Esos archivos se encuentran bajo una jerarquía que se muestra a continuación.

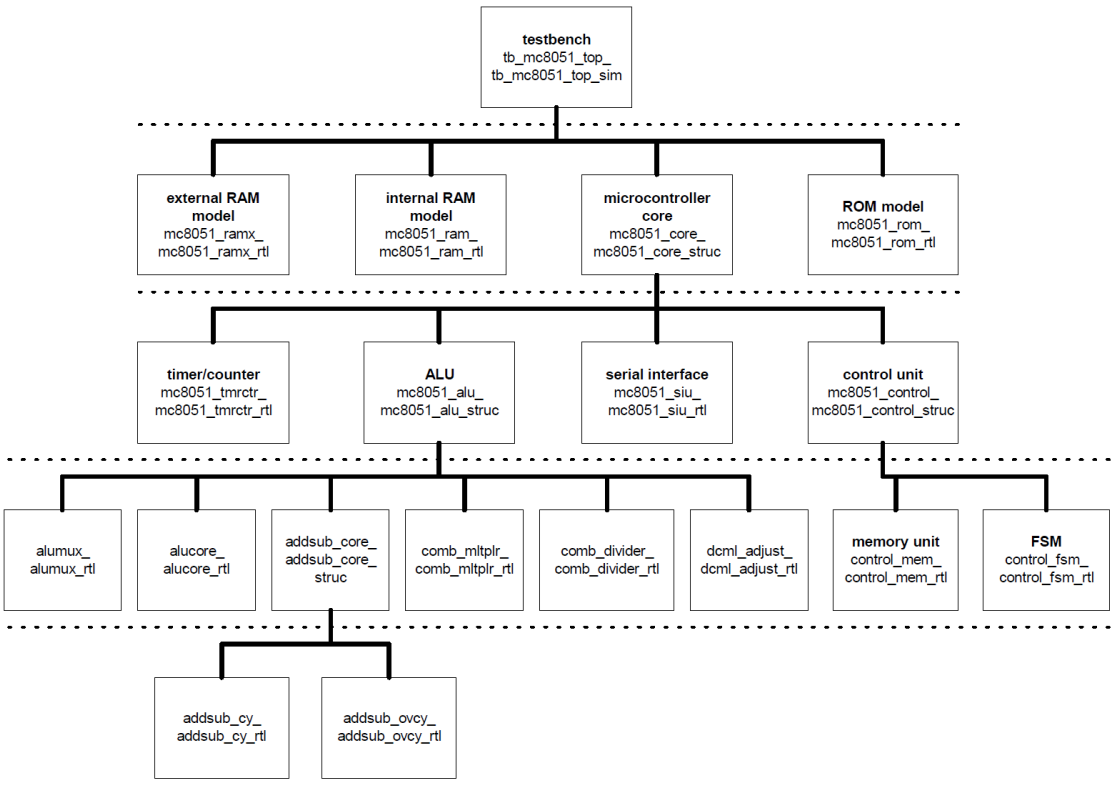


Ilustración 18. Jerarquía Oregano 8051. Recuperado de https://www.oreganosystems.at/

5) Generación de la memoria ROM mediante *CORE Generator*:
   a) Se crea la memoria ROM llamada "**mc8051_rom**".
   b) Se configura como tipo de memoria "*Single Port ROM*".
   c) La memoria del microcontrolador, aunque como máximo tiene un ancho de escritura de 16, se configura con 8 por problemas de memoria que se produce al sintetizar el diseño. En consecuencia, la profundidad es de 16384.
   d) La inicialización de memoria se realizará cuando se tenga un archivo coe ya simulado en ".asm".
   e) Activar el pin de Reset.

6) Generación de la memoria RAM mediante CORE Generator:
   a) Se crea la memoria RAM llamada "**mc8051_ram**".
   b) Se configura como tipo de memoria "*Single Port RAM*".
   c) La memoria tiene un ancho de escritura de 8 y una profundidad de 128.
   d) Activar el pin de Enable.
   e) Activar el pin de Reset.

7) Generación de la memoria RAM externa mediante CORE Generator:
   a) Se crea la memoria RAM externa llamada "**mc8051_ramx**".
   b) Se configura como tipo de memoria "*Single Port RAM*".
   c) La memoria del microcontrolador, aunque como máximo tiene un ancho de escritura de 16, se configura con 8 por problemas de memoria que se produce al sintetizar el diseño. En consecuencia, la profundidad es de 1024.
   d) Activar el pin de Reset.

8) Creación de un "*wrapper*": si se quiere sintetizar el proyecto, aparece un error de instancias entre las memorias creadas. Esto es debido a que no coinciden los puertos entre la identidad, el componente y la instancia. El 8051 espera que tenga unos puertos diferentes como "addr y "data", pero en su lugar se encuentra con que en la memoria de Xilinx aparecen puertos llamados "addra" y "dataina".

Para solucionar estos errores, se procede a la creación de un "wrapper". Con este componente se instancia a la memoria de CORE Generator, de forma que se entiendan entre sí.

```
COMPONENT mc8051_ram
  PORT (
    clka : IN STD_LOGIC;
    rsta : IN STD_LOGIC;
    ena : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
          --wea : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(6 DOWNTO 0);
    dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END COMPONENT;


COMPONENT mc8051_rom
  PORT (
    clka : IN STD_LOGIC;
    rsta : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END COMPONENT;


COMPONENT mc8051_ramx
  PORT (
    clka : IN STD_LOGIC;
    rsta : IN STD_LOGIC;
    wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
    addra : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    dina : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
  );
END COMPONENT;
```

Tabla 2. Archivo vhdl para instanciar a las memorias

9) Sintetizar e implementar el diseño.

5.5. Generación del archivo coe

Teniendo ya el microcontrolador correctamente configurado, es necesario crear un archivo para inicializar la memoria ROM y comprobar el correcto funcionamiento del diseño implementado.

Para ello, se crea un programa en ensamblador en lenguaje c, se genera un archivo en hexadecimal, se transforma a coe y se inicializa la memoria con el último archivo.

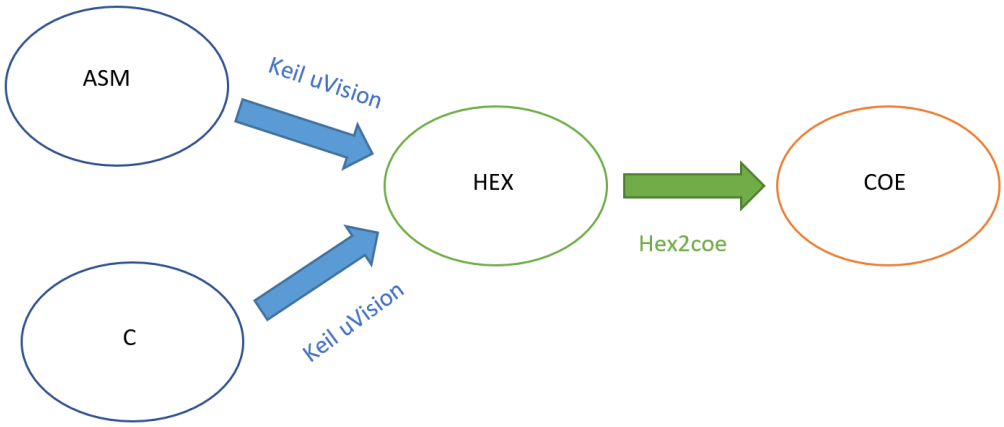


Ilustración 19. Diagrama generación archivo coe

5.5.1 Creación de un programa en ensamblador mediante Keil uVision

Para generar un archivo ensamblador que sea capaz de comunicarse con el 8051 es necesario un compilador/ensambaldor. Entre todos los toolchain encontramos varias opciones posibles, entre ellas: MikroElectronica Compiler Collection, IAR Embedded Workbench for 8051, Small Device C Compiler (SDCC), MIDE-51 y Keil uVision. Este último es el elegido para generar el archivo ensamblador.

Keil uVision Proporciona un compilador, depurador y un IDE completo llamado C51 para su programación. Este toochain es un software con licencia que se puede comprar desde su página web, aunque este proyecto se ha desarrollado usando la versión de evaluación limitada se puede descargar libremente. Para la creación de los programas, admite el desarrollo de software utilizando lenguaje y ensamblador 8051.

Para generar un programa mediante Keil uVision se debe empezar creando un nuevo proyecto y seleccionando el componente que queremos programar. En este caso buscamos uno con las mismas características que hemos diseñado. Seleccionar "AT89C51ED2" y crear el proyecto a realizar.

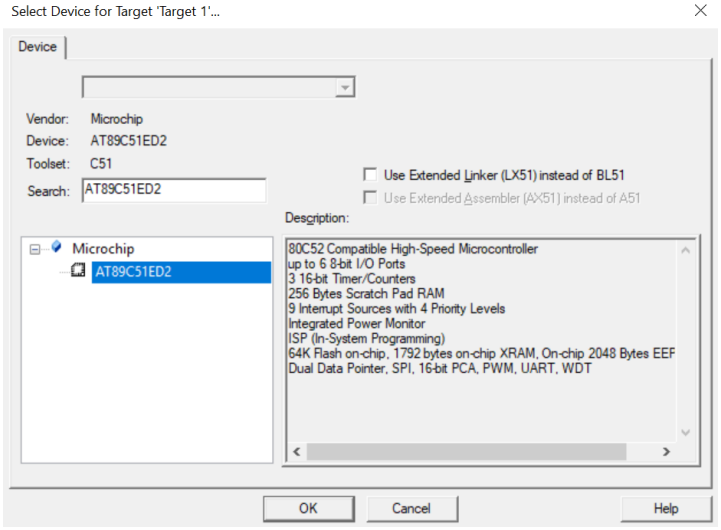


Ilustración 20. Selección de MC en Keil uVision

Realizar un programa con el lenguaje deseado. En este caso encontramos 2 opciones:

a. Generación de un archivo en ensamblador:

i. Crear el código necesario para la aplicación que se desea.

```
1  ORG 0000h ;indica el principio del programa y pone todos los registros que empiecen en la dirección 0000h
2
3  MOV A,#20h ;Introduce 20h en el Acumulador A
4  MOV R0,#15 ;Introduce 15 en el registro R0
5  MOV 0x80,#07h ;Introduce 7 en el P1
6  MOV 20h,#05h ;Introduce 05h en el registro de la dirección 20h
7
8  END
```

Ilustración 21. Programa sin guardar en Keil uVision

ii. Guardar en la carpeta del proyecto creado, añadiendo la extensión de programa ".asm". Se puede comprobar fácilmente porque cambia la visualización del código.

```
1 ORG 0000h ;indica el principio del programa y pone todos los registros que empiecen en la dirección 0000h
2
3 MOV A,#20h ;Introduce 20h en el Acumulador A
4 MOV R0,#15 ;Introduce 15 en el registro R0
5 MOV 0x80,#07h ;Introduce 7 en el P1
6 MOV 20h,#05h ;Introduce 05h en el registro de la dirección 20h
7
8 END
```

Ilustración 22. Programa guardado en Keil uVision

iii. Añadir archivo al proyecto y seleccionar *Asm Source file* en el tipo de archivo para elegir el que se ha creado.

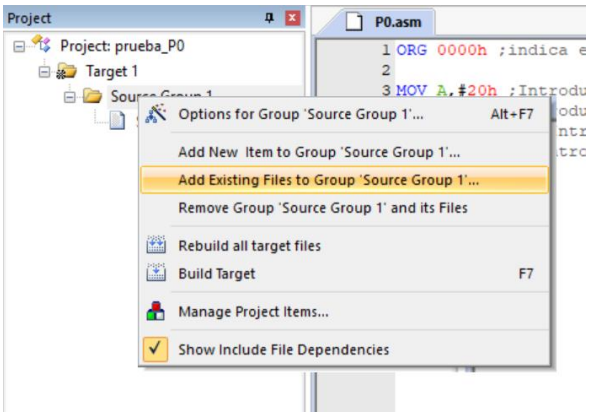


Ilustración 23. Adición de programa al proyecto

iv. Cambiar las opciones del archivo para generar el archivo ".HEX" generado una vez que se haya compilado correctamente. Para ello seleccionar *Options for File* 'prueba.asm' dentro de la ventana *Proyect*. Una vez dentro activar la opción *Create HEX file* en la pestaña Output.

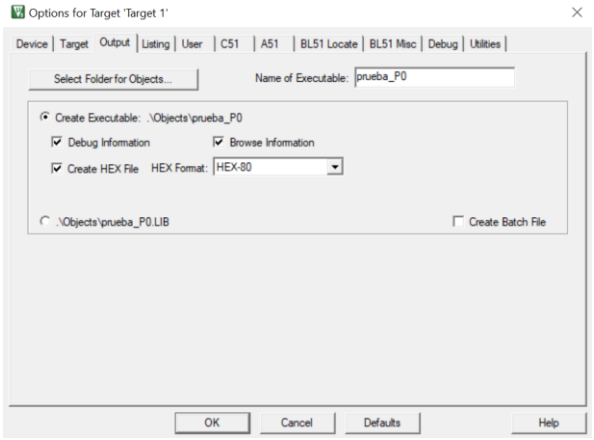


Ilustración 24. Selección de generación .HEX

v. Comprobar los errores y compilar usando los iconos de la parte superior .

vi. Iniciar el *Debuger* para comprobar el correcto funcionamiento del programa .

vii. Pulsando F5 podemos iniciar el programa. En el caso en el que queramos probar la información contenida en un periférico: abrir la venta *Peripherals* y seleccionar el deseado. Eligiendo el Puerto 0 del microcontrolador se nos abrirá una nueva ventana.

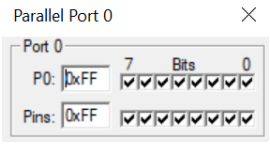


Ilustración 25. Ejemplo activación puerto en Keil uVision

b. Generación de un archivo en lenguaje C:

Seguir los mismos pasos que para la generación en ensamblador, cambiando los siguientes pasos:

i. Crear el código y guardar el programa con la extensión ".c".

ii. Añadir archivo al proyecto y seleccionar "*C Source file*" en el tipo de archivo para elegir el que se ha creado.

iii. Añadir la librería **REG51.h** la cual contiene las direcciones de memoria

```
#ifndef __REG51_H__
#define __REG51_H__

/*  BYTE Register  */
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
sfr B    = 0xF0;
sfr SP   = 0x81;
sfr DPL  = 0x82;
sfr DPH  = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0  = 0x8A;
sfr TL1  = 0x8B;
sfr TH0  = 0x8C;
sfr TH1  = 0x8D;
sfr IE   = 0xA8;
sfr IP   = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;


/*  BIT Register  */
/*  PSW   */
sbit CY   = 0xD7;
sbit AC   = 0xD6;
sbit F0   = 0xD5;
sbit RS1  = 0xD4;
sbit RS0  = 0xD3;
sbit OV   = 0xD2;
sbit P    = 0xD0;

/*  TCON  */
sbit TF1  = 0x8F;
sbit TR1  = 0x8E;
sbit TF0  = 0x8D;
sbit TR0  = 0x8C;
sbit IE1  = 0x8B;
sbit IT1  = 0x8A;
sbit IE0  = 0x89;
sbit IT0  = 0x88;
```

```
/*  IE   */
sbit EA   = 0xAF;
sbit ES   = 0xAC;
sbit ET1  = 0xAB;
sbit EX1  = 0xAA;
sbit ET0  = 0xA9;
sbit EX0  = 0xA8;

/*  IP   */
sbit PS   = 0xBC;
sbit PT1  = 0xBB;
sbit PX1  = 0xBA;
sbit PT0  = 0xB9;
sbit PX0  = 0xB8;

/*  P3  */
sbit RD   = 0xB7;
sbit WR   = 0xB6;
sbit T1   = 0xB5;
sbit T0   = 0xB4;
sbit INT1 = 0xB3;
sbit INT0 = 0xB2;
sbit TXD  = 0xB1;
sbit RXD  = 0xB0;

/*  SCON  */
sbit SM0  = 0x9F;
sbit SM1  = 0x9E;
sbit SM2  = 0x9D;
sbit REN  = 0x9C;
sbit TB8  = 0x9B;
sbit RB8  = 0x9A;
sbit TI   = 0x99;
sbit RI   = 0x98;

#endif
```

Tabla 3. Librería REG51.H

5.5.2 Generación de un archivo coe

Una vez que se ha generado el programa y se ha comprobado que realiza las operaciones por las que se le ha programado, debe aparecer un archivo con extensión ".HEX".

Este formato fue diseñado por *Intel* para la programación de microcontroladores, EPROMs y otros circuitos integrados. Se trata de un archivo de texto donde aparecen valores hexadecimales que codifican datos junto con su dirección de memoria.

Cada línea consta de los siguientes elementos:

:03000000020800F3

:0C080000787FE4F6D8FD75810702000047

:0A000000E7420780F7580077520054 5

:00000001FF

Código de inicio: símbolo ':'.

Longitud del registro: dos dígitos hexadecimales con la cantidad de bytes del campo de datos. Usualmente son 16 o 32 bytes.

Dirección: cuatro dígitos hexadecimales con la dirección de inicio de los datos. Para direcciones mayores a 0xFFFF se emplean otros tipos de registro.

Tipo de registro: dos dígitos hexadecimales (de 00 a 05) que definen el tipo del campo de datos.

Datos: dígitos hexadecimales que contienen los datos del programa.

Checksum: dos dígitos hexadecimales con el complemento a dos de la suma de todos los campos anteriores, salvo el código de inicio.

También hay 6 tipos de registros:

00: Datos. Contiene una dirección de 16 bits y los datos correspondientes

01: Fin de archivo. No contiene datos y debe estar al final del archivo.

02: Dirección extendida de segmento. Dirección para acceder a direcciones con más de 16 bits. Este valor se desplaza 4 bits a la izquierda y se suma a la dirección proporcionada por los registros de datos.

03: Dirección de comienzo de segmento. Especifica los valores iniciales de los registros CS:IP, para procesadores 80x86. los datos contienen dos bytes para el segmento de código y otros dos para el instruction pointer

04: Dirección lineal extendida. Permite dirigirse a 32 bits de memoria al contener los 16 bits superiores de la dirección.

05: Comienzo de dirección lineal. Contiene 4 bytes que se cargan en el registro EIP de los procesadores 80386 y superiores.

Ahora es necesario inicializar la memoria ROM del microcontrolador 8051. En concreto se necesita un archivo coe que es necesario generar a partir del archivo hexadecimal.

Para obtener este archivo es necesario usar un programa llamado **hex2coe**, el cual se encarga de obtener la inicialización de memoria ofreciéndole un archivo con el nombre **filename.hex**. Tambíen se puede modificar el programa para que tome el archivo con el nombre con el que se haya creado. Ambos deben estar en la misma carpeta para su generación.

```
#include <iostream>

#include <iostream>
#include <fstream>
#include <assert.h>
#include <cstdlib>
#include <string>


using namespace std;
int main()
{
     ifstream fin;     // declare stream variable name
     ofstream fout;
     string line;
     char output[3];         // 2 chars + termination
     /* These could be inputs */

          long max_address = 0xFFFF;
          unsigned char default_value = 0;

     unsigned char buffer[65536];
     unsigned char byte_count;

     char * p;

     long i;
     int j;
     long address;
     long address_offset = 0;

     for (i=0; i <= max_address; i++)
          {
                    buffer[i] = default_value;
          }
fin.open("filename.hex",ios::in);    // open file
assert (!fin.fail( ));
          getline(fin, line);
while (!fin.eof( ))      //if not at end of file, continue reading numbers
{
          cout << line << endl;
          switch (line[8])
          {
                    case '0':
                    {
                              /* This is data */
                              /* First two bytes = number of bytes in hex */
                              byte_count = strtol(line.substr(1,2).c_str(), &p, 16);
                              /* check address */
                              address = address_offset + strtol(line.substr(3,4).c_str(), &p,
16);
                              if (address <= max_address)
                              {
                                        for (i = 0; i < byte_count; i++)
                                        {
                                                  buffer[address + i] =
strtol(line.substr(9+i*2,2).c_str(), &p, 16);
                                        }
                              }
                              else
                              {
                                        cout << "Invalid Address" << endl;
                                        assert(1);
                              }
                              break;
                    }
```

```
                    case '1':
                    {
                              cout << "File complete." << endl;
                              break;
                    }
                    case '4':
                    {
                              address_offset = strtol(line.substr(9,4).c_str(), &p, 16);
                              cout << "New high address = " << line.substr(9,4) << endl;
                              break;
                    }
                    default:
                    {
                              cout << "Invalid line in hex file." << endl;
                              assert(1);
                              break;
                    }
          }
          getline(fin, line);
     }
    }
     fin.close( );        //close file
     /* Now genereate coe file from buffer */
     fout.open("filename.coe",ios::out);
     fout << "; COE File Generated from filename.hex" << endl;
     fout << "memory_initialization_radix=16;" << endl;
     fout << "memory_initialization_vector=" << endl;
           i = 0;
           while (i <= max_address)
           {
                    for (j= 0; j < 32; j++)
                    {
                              sprintf((char*) &output, "%02X", buffer[i+j]);
                              fout << output << " ";
                    }
                    fout << endl;
                    i = i + 32;
           }

     fout.close();
}
```

Tabla 4. Programa para generar archivos coe a partir de hex

El archivo que se genera con el nombre **filename.coe**, contiene los datos necesarios para inicializar la memoria. Si se abre con un editor de texto se puede visualizar su contenido.

```
; COE File Generated from filename.hex
memory_initialization_radix=16;
memory_initialization_vector=
74 20 78 0F 75 80 07 75 20 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Tabla 5. Ejemplo del contenido de un archivo coe generado

En el proyecto anteriormente creado mediante *ISE Design*, se vuelve a configurar la memoria ROM. Para ello se abre **mc8051_rom** y se selecciona el archivo para inicializar la memoria, activando *Load Init File* y eligiendo el archivo coe.

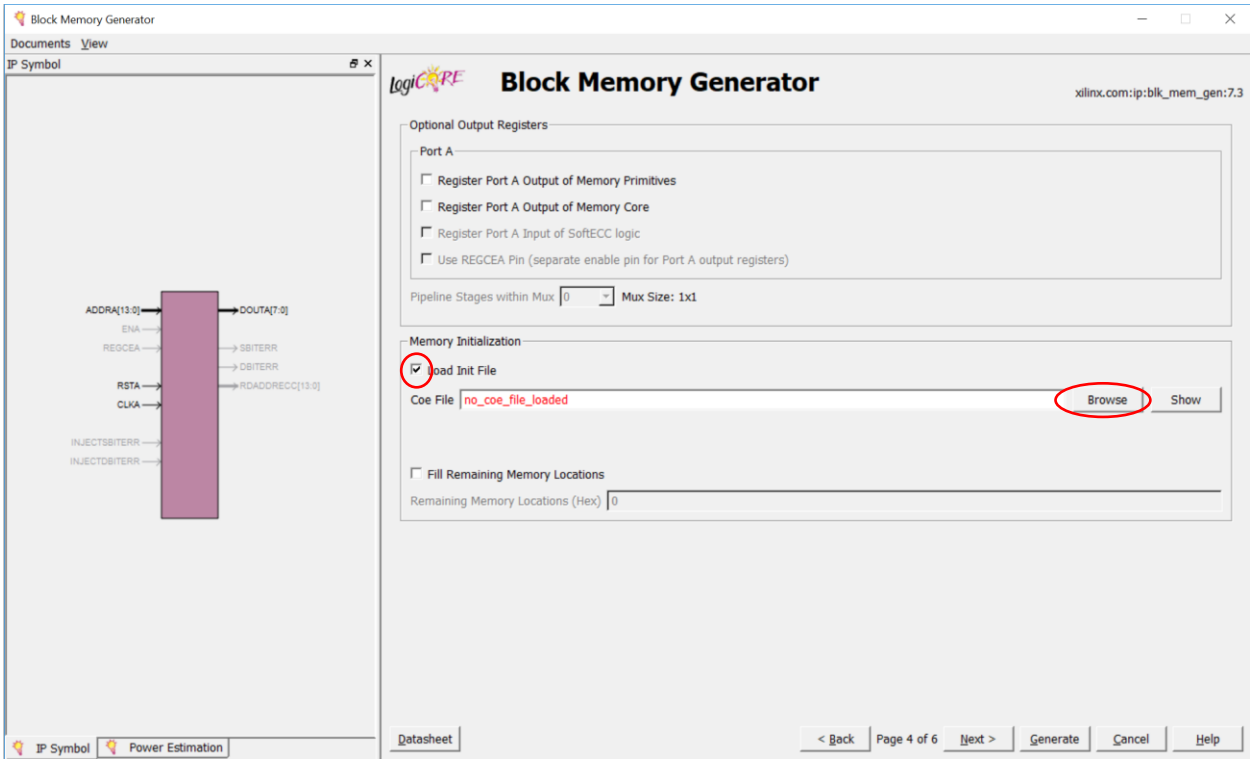


Ilustración 26. Inicialización de memoria

Utilizando la FPGA **LX90 Microborad**, se produce un error de tamaño máximo de memoria. Esto es debido a que anteriormente se establece el tamaño de memoria menor que en el del proyecto original.

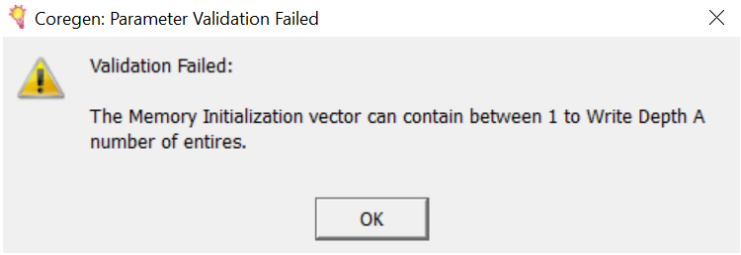


Ilustración 27. Error en la inicialización de memoria

Si el archivo coe no ocupa toda su totalidad, se puede borrar tantas filas de ceros como sea necesario para que no se produzca este error. De esta forma ya se tiene el proyecto totalmente configurado y preparado para desempeñar las distintas funcionalidades para las que se vaya a programar.

5.6. Creación de un programa en ensamblador

El lenguaje ensamblador es un lenguaje de programación de bajo nivel utilizado para escribir código de

programa en términos de mnemónicos. A pesar de que hay muchos idiomas de alto nivel que actualmente están en demanda, el lenguaje de programación de ensamblador se usa popularmente en muchas aplicaciones. Se puede usar para manipulaciones directas de hardware. También se usa para escribir el código de programación 8051 de manera eficiente con menos ciclos de reloj, al consumir menos memoria en comparación con otros lenguajes de alto nivel.

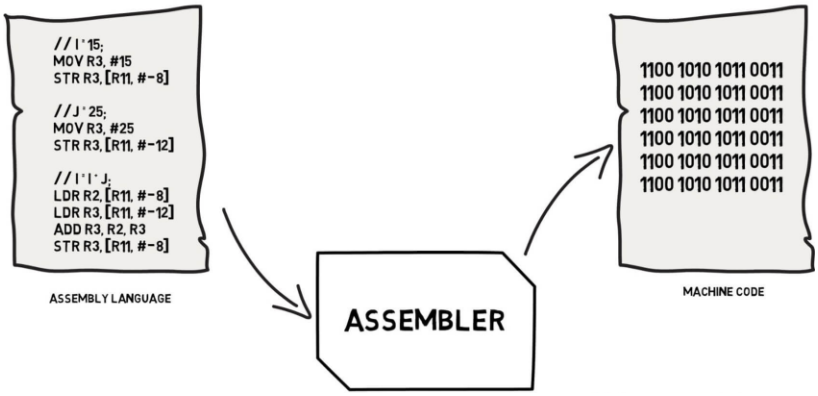


Ilustración 28. Ensamblador. Recuperado de https://www.androidauthority.com/assembly-language-and-machine-code-678230/

Este lenguaje de programación está totalmente relacionado con el hardware. Los programadores deben tener el suficiente conocimiento sobre el hardware de un procesador antes de escribir el programa. El lenguaje ensamblador está desarrollado por mnemónicos; por lo tanto, los usuarios no pueden entenderlo fácilmente para modificar el programa.

6 RESULTADOS

No todo lo que es de oro reluce, ni toda la gente errante anda perdida.

- Aragorn -

En este capítulo se expondrán los resultados obtenidos en **Keil uVision** y en **ISIM** para comprobar el correcto funcionamiento del proyecto. En la primera prueba se genera un programa en ensamblador mediante *Keil uVision* y después se añade el archivo coe generado para comprobar si los resultados son los mismos en ISIM.

6.1. Prueba programa en lenguaje ensamblador

El programa P0P1P2P3.asm está creado en lenguaje ensamblador para probar el correcto funcionamiento del microcontrolador 8051 desarrollado. Se va a escribir diferentes números en formato hexadecimal en los distintos puertos del micro. A continuación, se presenta el programa descrito.

```
ORG 0000h

MOV 0x80,#06h
MOV 0x90,#07h
MOV 0xA0,#08h
MOV 0xB0,#09h

END
```

Tabla 6. Programa en ensamblador

Se procede a explicar cómo funciona este sencillo programa:

- **ORG 0000h**: Indica el comienzo del programa y es usado para establecer la dirección de los registros durante su ensamblado. En esta instrucción en concreto, se le dice al compilador que el código empieza en la dirección 0.

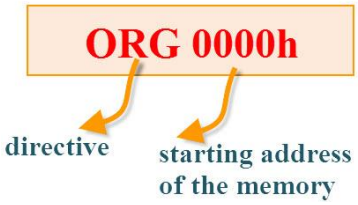


Ilustración 29. Instrución ORG ensamblador

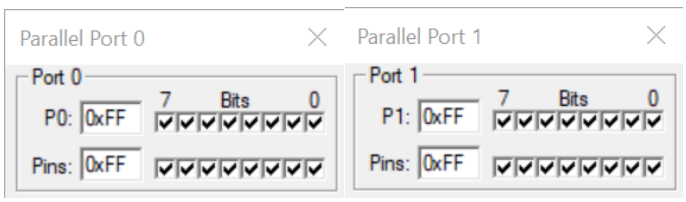
- **MOV 0x80, #06h**: la instrucción MOV indica modo de direccionamiento a una dirección de memoria. El destino es un registro del microcontrolador que debe estar con el '#' por delante. Se usa para almacenar inmediatamente el valor en los registros de memoria. En este caso se introduce el valor hexadecimal **6** (#06h) en la dirección del **Pin 0** (0x80).

Ilustración 30. Instrución MOV ensamblador

- **MOV 0x90, #07h**: se introduce el valor hexadecimal **7** (#07h) en la dirección del **Pin 1** (0x90).
- **MOV 0xA0, #08h**: se introduce el valor hexadecimal **8** (#08h) en la dirección del **Pin 2** (0xA0).
- **MOV 0xB0, #09h**: se introduce el valor hexadecimal **9** (#09h) en la dirección del **Pin 3** (0xB0).

A continuación, hay que seguir los pasos explicados en el punto **4.5.1**. Podemos comprobar su funcionamiento activando la visibilidad de todos los puertos en la pestaña *Periphreals*.

Antes de iniciar el programa se encuentran todos los bits de los puertos activos.



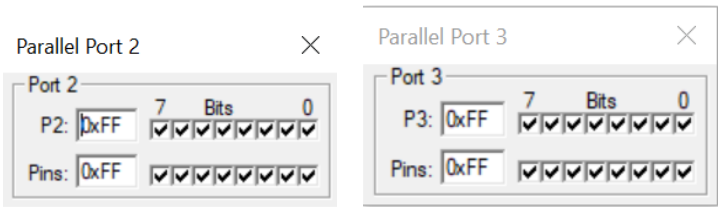


Ilustración 31. Estado 0 pines prueba 1

Una vez que se inicia el programa se activan los distintos pines de todos los puertos del microcontrolador, de forma que escriben los números escritos en forma binaria.

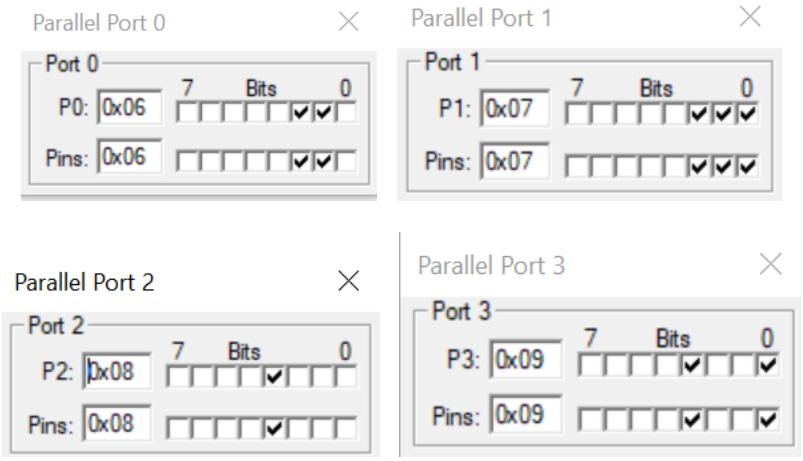


Ilustración 32. Estado 1 pines prueba 1

Observando por ejemplo el puerto 2, podemos comprobar cómo escribe correctamente el valor en binario "00000100" (8) introducido.

Ahora que se ha comprobado el correcto funcionamiento de este programa, es necesario generar el archivo coe para inicializar la memoria. Luego, en el proyecto de *ISE Design*, hay que añadir ese archivo en la memoria ROM para inicializarla. Tras ello, generar el core (y regenerarlo si se ha modificado el archivo) y sintetizar el diseño.

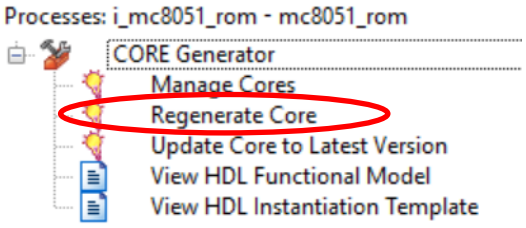


Ilustración 33. Regeneración de core

Es necesario comprobar el programa en nuestro microcontrolador y para ello se va a usar la herramienta que ofrece *Xilinx* para simularlo. Hay que crear un archivo de pruebas tipo *testbench* y elegir qué vamos a simular. En este caso elegimos **mc8051_top**.

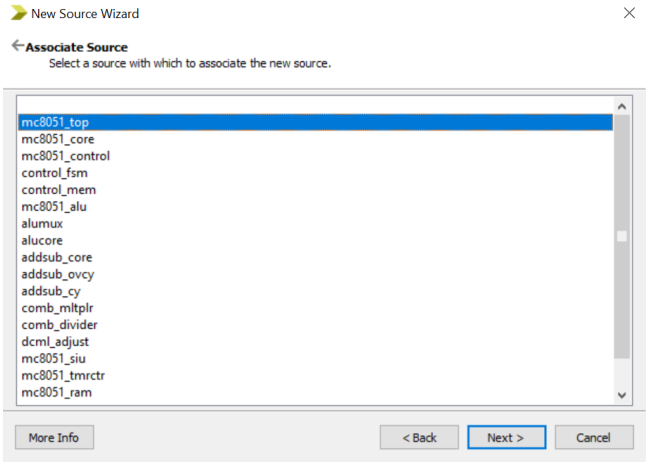


Ilustración 34. Creación de test bench

De esta forma podemos simular los estímulos a los que estará sometido el micro en la vida real. Para esta prueba sólo se va a activar el **Reset** durante 10 ciclos de reloj y después desactivarlo.

```
stim_proc: process
begin
            reset <= '1';
   wait for clk_period*10;
            reset <= '0';
   wait;
end process;
```

Tabla 7. Test bench prueba 1

Se comprueba que no haya ningún error de sintaxis y observamos la simulación que se genera.

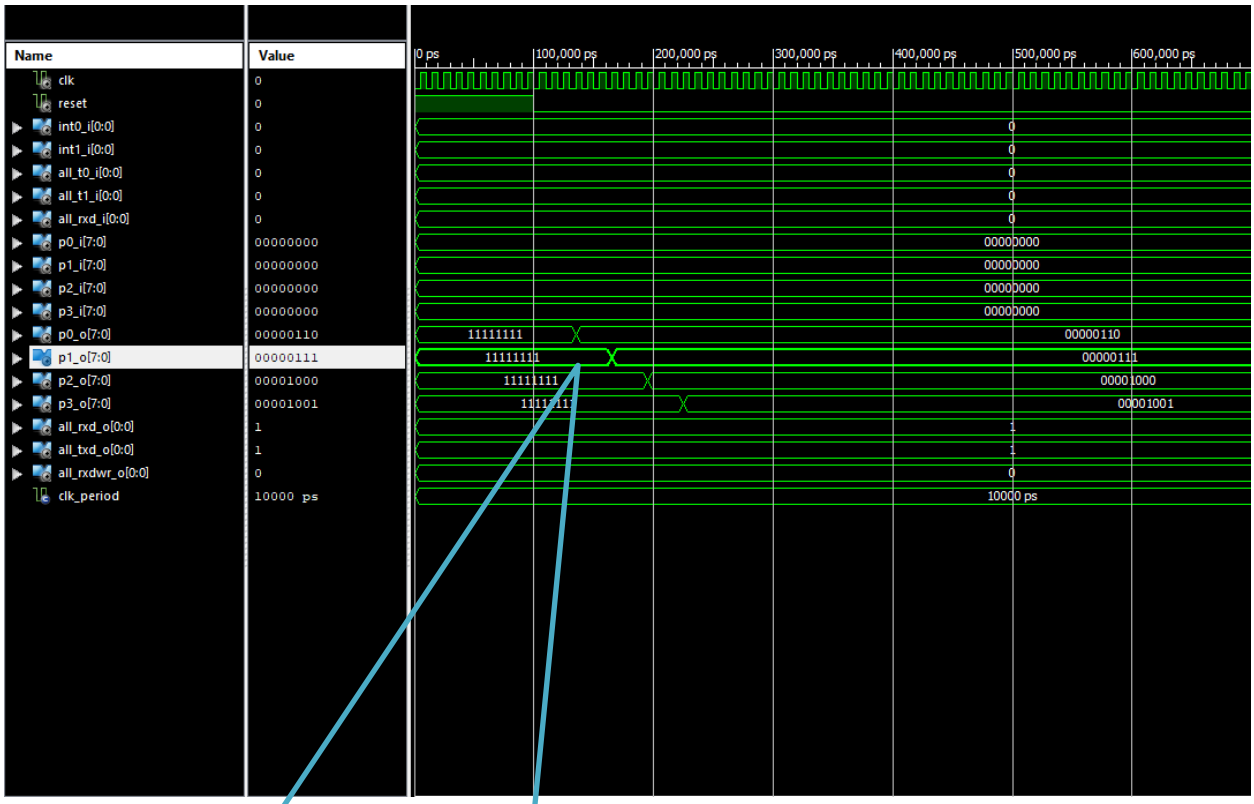


Ilustración 35. Prueba 1 resultado ISIM

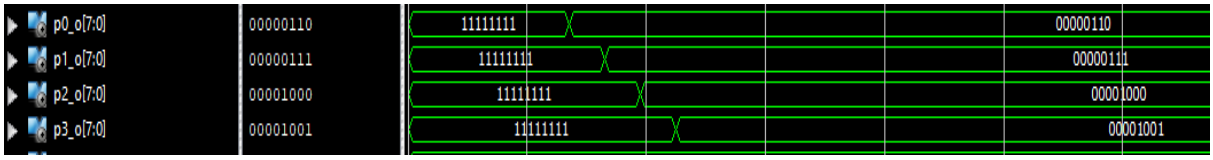


Ilustración 36. Prueba 1 aumentado resultado ISIM

Tomando como ejemplo el puerto 2, debe escribir un 8 en hexadecimal; es decir el valor en binario: “00000100”.

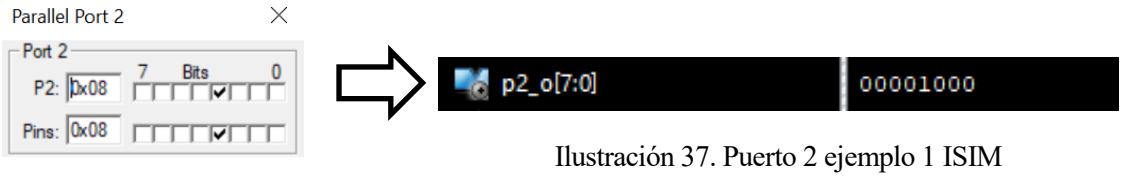


Ilustración 37. Puerto 2 ejemplo 1 ISIM

Ilustración 38. Puerto 2 ejemplo 1 Keil uVision

Viendo el resultado de la simulación, podemos concluir que el programa creado en lenguaje ensamblador funciona correctamente en el microcontrolador implementado.

6.2. Prueba de programa en lenguaje C

El programa **PRUEBAEN_C** realiza varias operaciones con el fin de comprobar si es posible la inicialización de la memoria ROM a partir de un programa realizado en lenguaje C. Para este caso, se necesita la librería **REG51.h.**

El funcionamiento de este ejemplo es sencillo: se usan 2 puertos: P3 para encender o apagar botones y P2 para encender o apagar los LEDs. Al apagar un bit del puerto 3 se enciende un LED del mismo bit, pero del puerto 2 durante un segundo y luego se apaga.

```
#include<reg51.h>

//LEDs
sbit Led1 = P2^0;
sbit Led2 = P2^1;
sbit Led3 = P2^2;
sbit Led4 = P2^3;

//Botones
sbit Boton1 = P3^0;
sbit Boton2 = P3^1;
sbit Boton3 = P3^2;
sbit Boton4 = P3^3;

void Delay(int);

int main ()

{
  P2 = 0x00;       //output port
  P3 = 0xFF;       //input port
do
{
if(Boton1 == 0 )
    {
      Led1 = 1;
                              Delay(1000);
      Led1 = 0;
    }
```

```
if(Boton2 == 0 )
          {
       Led2 = 1;
                                Delay(1000);
       Led2 = 0;
    }
if(Boton3 == 0 )
     {
       Led3 = 1;
                                Delay(1000);
       Led3 = 0;
     }
if(Boton4 == 0 )
     {
       Led4 = 1;
                                Delay(1000);
       Led4 = 0;
     }
}
       while(1);
}


void Delay(int k)
{
          int j;
          int i;
          for(i=0;i<k;i++)
  {
                    for(j=0;j<100;j++)
    {
    }
  }
}
```

Tabla 8. Programa en lenguaje C

Al iniciar el programa todos los bits del puerto 2 se encuentran desactivados y todos los del puerto 3 activos.

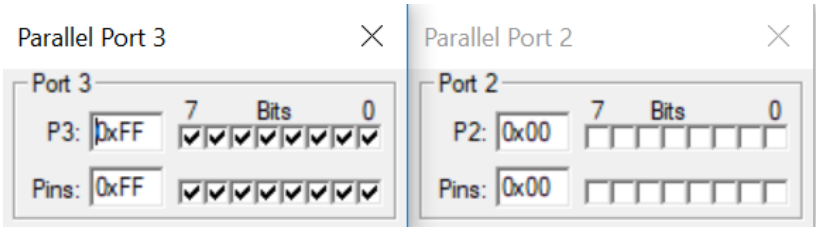


Ilustración 39. Estado 0 pines prueba 2

Se activan todos los bits del puerto 3 para ver la secuencia completa que realiza el puerto 2.

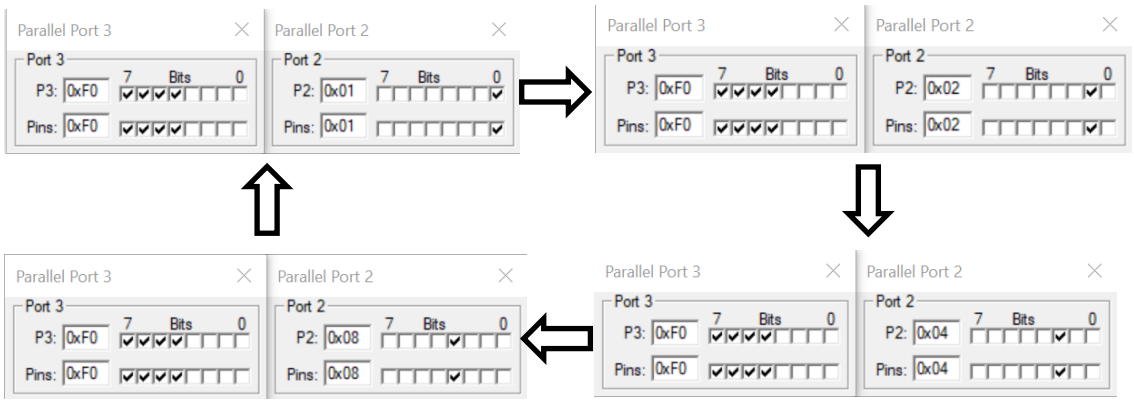


Ilustración 40. Secuencia prueba 2

De nuevo acudimos a la herramienta de Xilinx para la simulación del microcontrolador 8051. Una vez se ha inicializado la memoria con el programa y se ha regenerado el core, pasamos a configurar el test bench.

En este caso además de activar y desactivar el reset, es necesario activar los bits de entrada del pin 3 necesarios para que funcione el programa.

```
stim_proc: process
begin
                reset <= '1';
   wait for clk_period*10;
                reset <= '0';
                p3_i <= "11110000";
   wait;
end process;
```

Tabla 9. Test bench prueba 2

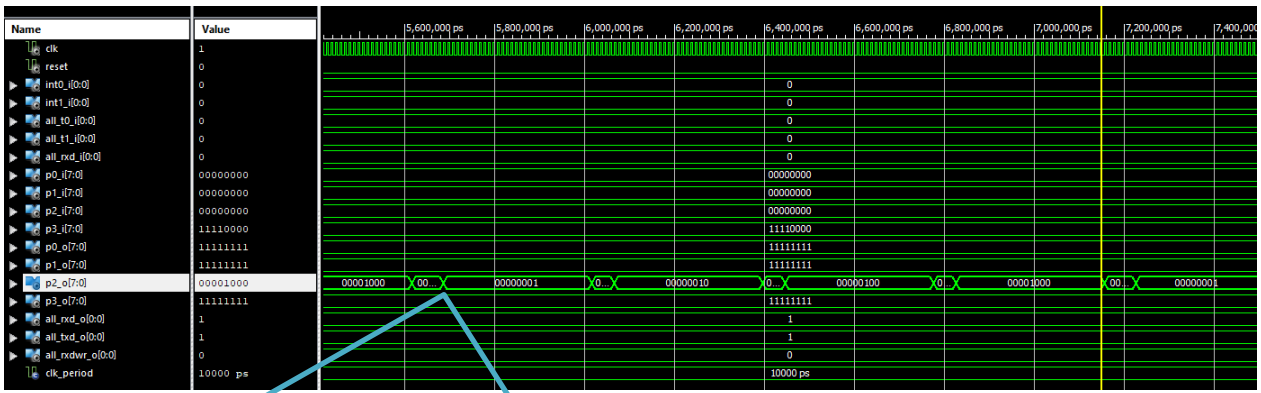


Ilustración 42. Prueba 2 resultado ISIM

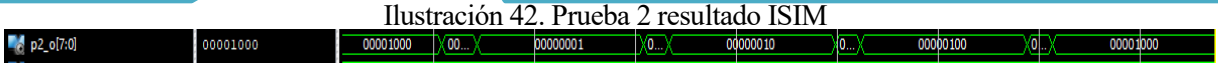


Ilustración 41. Prueba 2 aumentado resultado ISIM

En ISIM comprobamos cómo funciona correctamente la secuencia del programa. Se puede observar cómo durante varios ciclos de reloj el puerto se encuentra con todos los bits a cero. Esto es lo que tarda el programa en pasar de una instrucción a otra.

6.3. Prueba de un ejemplo real

En esta prueba se va a inicializar la memoria ROM del microcontrolador con un programa creado en la empresa *AND&OR*. Debido a que este programa forma parte de los recursos de la empresa y, por lo tanto es de uso privado, solo se va a exponer una parte de su contenido y prueba para comprobar su funcionamiento.

El programa fue creado en septiembre de 1990 por el programador Jaime Lobato y en su programa se encuentran 3 programas distintos para máquinas diferentes: un comprobador de poros, el control de una cinta de transporte y el pesaje de cajas. Para este ejemplo, vamos a elegir un comprobador de poros. Su funcionamiento es el siguiente: las botellas entran a través de un trasnsportador, se detienen ante un cabezal que baja hasta el cuello de la botella e insufla aire para medir su presión. Si es correcta continuará hasta la siguiente fase de producción y si hay alguna anomalía en su medida se descartará. Tiene las siguientes características:

1. Programación de parámetros del equipo a través del teclado.
2. Almacenamiento de dichos parámetros en EEPROM.
3. Posee 3 niveles de programación:
    a. Nivel 3: Accesible mediante la pulsación de la letra **E**.
        i. Presión en el interior del envase.
        ii. Tiempo de pruebas.
        iii. Tiempo de retardo a la expulsion.
        iv. Preselección para el llenado de cajas.
    b. Nivel 2: Accesible al pulsar el número **2** al encender la máquina.
        i. Detección de fugas.
        ii. Expulsión de buena o mala.
        iii. Detección de rebabas.
        iv. Tiempo de retardo tras la lectura de la fotocélula de entrada.
        v. Número de identificación.
        vi. Tipo de envase.
        vii. Autotest S/N
    c. Nivel 1: Accesible al pulsar ***** y una clave.
        i. Tipo de control.
        ii. Control local o remoto.

```
...
INICIO              CALL      IPORT
                    BAJADACAB
                    MOV       R5,#7             ;TEMP DE 1sg.
I2                  MOV       R4,#0
I1                  MOV       R3,#0
                    DJNZ      R3,$
                    DJNZ      R4,I1
                    DJNZ      R5,I2
                    MOV       SP,#30H
                    CALL      DISPI             ;INICIALIZA DISPLAY
                    CALL      BMEMOR
                    CALL      CINI1A            ;COMPROBACION DE DATOS DE EEPROM
                    MOV       A,R7
                    JZ        INI1              ;SALTA SI NO HUBO ERROR
          ;
                    CALL      ESP8
                    MOV       R3,A
                    CALL      BINASC            ;PASA A ASCII EL ERROR
                    MOV       2FH,R3
                    MOV       DPTR,#MENS21      ;'ERROR No.'
                    CALL      TMENSAL
                    MOV       AUX1,#0FFH        ;FLAG DE ERROR
          ;
INI1                MOV       TECLA,#0FFH       ;BORRA TECLA
                    MOV       FINT,#00000111B   ;HABILITA PITIDO, TECLADO Y, TEMPORIZ.
                                                ;DESABILITA DISPLAY
                    MOV       CINT1,#1
                    MOV       CINT2,#90H        ;TEMP. DE 1 sg
                    CALL      ACTIN2            ;ACTIVA INTERRUPCIONES
INI04               MOV       A,TECLA
                    CJNE      A,#0FFH,INI03
                    MOV       A,FINT
                    JB        ACC.2,INI04       ;NO SALE HASTA TERMINAR LA TEMP
INI05               MOV       A,AUX1
                    JZ        INI10             ;SALTA AL PROGRAMA PRINCIPAL SI NO HUBO ERROR
                    JMP       $                 ;!! RUPTURA DEL PROGRAMA !!
                                                ;NO SALE SI DETECTO UN ERROR.
          ;
INI03               CJNE      A,#TENTER,INI02
                    CALL      PNIVEL3           ;SI PULSO LA 'E' HACE PROGRAMACION 2
                    CALL      GRABN3            ;GRABACION DE DATOS
                    JMP       INI10
...
```

Tabla 10. Programa real en ensamblador

Para esta simulación se va a activar el rechazo de la botella como si lo hubiera detectado como una botella que no cumple las especificaciones y se va a activar una alarma que debe quedarse enclavada hasta que sea reseteado manualmente. Para ello usamos los bits **0** y **3** del puerto **P1**.

```
stim_proc: process
   begin
                    reset <= '1';
                    wait for clk_period*10;
                    reset <= '0';
                    --p3_i <= "11110000";
                    p1_i <= "00000001";
                    wait for clk_period*10;
                    p1_i <= "00000000";
                    wait for clk_period*10;
                    p1_i <= "00000100";
                    wait for clk_period*10;
                    p1_i <= "00000000";
      wait;
   end process;
```

Tabla 11. Test bench prueba 3

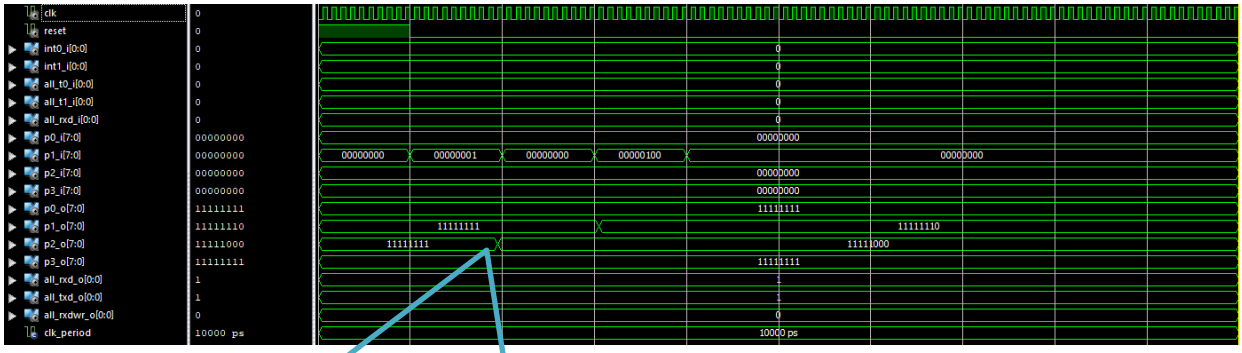


Ilustración 43. Prueba 3 resultado ISIM

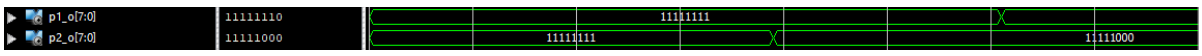


Ilustración 44. Prueba 3 aumentado resultado ISIM

Tras esta simulación se puede comprobar cómo cumple las funciones para las que se le ha programado. Por un lado, se activa la alarma, la cual tarda varios ciclos de reloj en activarse a través del puerto **P2**. Tras 10 ciclos de reloj se activa también la señal de rechazo, la cual aparece por el puerto **P1**.

Esta prueba demuestra que todo el trabajo hecho anteriormente ha sido un éxito, ya que se ha podido probar un programa real que existe en muchas máquinas y que ahora se puede realizar por medio de una FPGA.

7 CONCLUSIONES Y TRABAJOS FUTUROS

Que las estrellas brillen para ti hasta el final del camino.

- Gildor -

7.1. Conclusiones

En este proyecto se ha desarrollado una metodología para implementar un microcrontrolador 8051 en una FPGA y así solventar el problema de obsolescencia que posee actualmente. La utilización de FPGA como solución a este modelo de microcontrolador está justificado por diversas razones.

La familiarización con el lenguaje VHDL es la primera de estas razones. Este lenguaje de descripción hardware usado para describir circuitos digitales ha sido anteriormente utilizado en asignaturas de este grado como *Sistemas Electrónicos* y *Sistemas Electrónicos para la Automatización* del *Grado en Ingeniería Electrónica, Robótica y Mecatrónica*. Por ello, el IP Core *Oregano 8051* no ha sido un hándicap añadido ya que, mediante la recopilación de documentación necesaria, se ha podido sintetizar el circuito sin ser necesaria la gran curva de aprendizaje que es el entorno de programación de Xilinx.

La segunda razón es por el uso de las FPGAs en el entorno industrial es debido a sus prestaciones, versatilidad y capacidad de respuesta. Este dispositivo programable ofrece una configurabilidad que atrae cada vez más a las empresas por el hecho de poder actualizar sus productos o instalaciones sin tener que desarrollar una alternativa muy costosa en recursos.

En tercer lugar, otra razón por la que se ha elegido esta alternativa es la comunidad que respalda el uso de los *softcores*. Ha ofrecido de manera gratuita una solución al problema de obsolescencia de este dispositivo electrónico y, a cambio se ha generado la documentación necesaria para hacer funcionar el microcontrolador 8051 de una forma eficaz. También hay que destacar que este proyecto se ha realizado sin tener coste económico debido a la licencia *LGPL* que posee.

Por otro lado, en la realización de este proyecto fin de grado se ha aprendido las buenas prácticas en el desarrollo de proyectos. En la asignatura *Proyectos Integrados* ya se introdujo la gran importancia que cobra el uso de los repositorios como *Git.* Mediante su empleo se previene la sobre-escritura de versiones y la recuperación de anteriores versiones de proyectos o archivos. No se tiene en cuenta la importancia de estos repositorios hasta que ocurre un problema, ya que ofrece una protección de todos nuestros archivos. Tambíen es destacable su uso para organizar, preservar y contribuir a difundir información en su modo de acceso libre. De esta forma se ayuda a la libre difusión de información a todo el mundo para que pueda llevar a cabo un proyecto desde la misma u otra perspectiva, pero teniendo presente toda esa información integrada.

En conclusión, en este proyecto se ha volcado todos los conocimientos adquiridos durante estos años de aprendizaje. Sobre todo se ha ofrecido una alternativa a un problema real que posee una empresa y que hubiera conllevado una enorme inversión en actualizar todas esas máquinas junto con el tiempo que eso hubiera requerido. De esta forma se ha establecido una solución mediante el uso de FPGAs para que incluso en un futuro se pueda reconfigurar.

7.2. Trabajos futuros

Para este punto se van a exponer las posibles ideas que se podrían realizar de cara al futuro. También se incluyen algunas propuestas que han quedado pendientes debido a la amplia extensión del trabajo que excedía los límites de un trabajo fin de grado. Estas ideas son:

1. Elección de una FPGA que se adapte a las necesidades de la empresa.
2. Integración de este proyecto en una FPGA real.
3. Verificación de distintos programas realizados anteriormente usando esta alternativa.
4. Construcción de un pequeño prototipo para exponer la FPGA a problemas reales a los que estaría sometida.
5. Integración de la FPGA en el circuito real que posee la máquina en cuestión, mediante la construcción de un adaptador que implique los menores cambios posibles en el circuito.
6. Aumentar la velocidad del microcontrolador desde los 30MHz que posee actualmente hasta los 100MHz de las FPGAs. Esto supondría el posible no funcionamiento del micro a 100MHz debido a que si el programa está realizado para que funcione a 30MHz y se hace funcionar a más alta frecuencia, habría problemas en el timing del software. Por ejemplo si hace X *NOPs* durante un ciclo normal en una máquina, con la nueva frecuencia deberá esperar 3.3*X NOPs.
7. Comunicación mediante puerto serie o Ethernet con la FPGA para conocer el estado real de cada uno de los puertos mientras la persona se encuentra conectada en remoto.

REFERENCIAS

[1] *Oregano 8051 IP Core*. [en línea] https://www.oreganosystems.at/products/ip-cores/8051-ip-core

[2] (2018) *Programming 8051 on a FPGA board*. http://www.eecs.ucf.edu/~jinyier/courses/EEE4932/lab3_8051/source/8051_Guide.pdf

[3] (2018) *8051 IP Cores*. [en línea] http://turbo51.com/documentation/8051-ip-cores

[4] (2018) *Open Cores T51*. [en línea] https://opencores.org/project/t51

[5] Guzmán Miranda, Hipólito. (2018). *Tutorial de Git en modo gráfico*. [en línea] https://hipolitoguzman.net/tutorial-git-modo-grafico/

[6] *Xilinx*. [en línea] https://www.xilinx.com

[7] (2018) *ARM Keil*. [en línea] http://www.keil.com/dd/chip/3509.htm

[8] Bascón Mallado, José María. (2017). *Desarrollo de un flujo de preparación de diseño para inyección de fallos sobre el microcontrolador openMSP430* (Trabajo fin de grado). Universidad de Sevilla, Sevilla, España.

[9] (2018) *Ensambladores 8051*. [en línea] http://www.8051projects.info/resources/8051-assemblers-and-ides.22

[10] (2018) *Manual del microcontrolador 8051*. [en línea] http://lacova.upc.es/~jesusv/uc8051_web/man_51_cast.pdf

[11] (2018) *Accessing code ROM space in 8051 C*. [en línea] http://what-when-how.com/8051-microcontroller/accessing-code-rom-space-in-8051-c/

[12] (2018) *CoolRunner-II CPLD 8051 Microcontroller Interface. [en línea] https://www.xilinx.com/support/documentation/application_notes/xapp393.pdf*

[13] (2018) *Introduction to 8051 Programming in Assembly Language*. [en línea] https://www.elprocus.com/8051-assembly-language-programming/

[14] (2018) *Sinthesizable VHDL Model of 8051*.[en línea] http://www.cs.ucr.edu/~dalton/i8051/i8051syn/

[15] (2018) *La familia del microcontrolador 8051* .[en línea] http://galia.fc.uaslp.mx/~cantocar/microcontroladores/SLIDES_8051_PDF/4__EL8051.PDF

[16] (2018) *The Unoffical History of 8051* .[en línea] http://www.cpushack.com/Historyofthe8051.html

[17] Puro Marketing. (2017). *Obsolescencia programada y reparaciones imposibles: ¿prácticas llamadas a morir?*. [en línea] https://www.puromarketing.com/13/29064/obsolescencia-programada-reparaciones-imposibles-practicas-llamadas-morir.html

[18] Gary Gutierrez. (2013). *Qubit, la unidad* fundamental del futuro infromático y tecnológico . [en línea] https://www.fayerwayer.com/2013/09/qubit-la-unidad-fundamental-del-futuro-informatico-y-tecnologico/

[19] (2018) *SDCC*. [en línea] http://sdcc.sourceforge.net/

[20] *IAR Embedded Workbench*. [en línea] https://www.iar.com/iar-embedded-workbench/

[21] *MikroElectronika*. [en línea] https://www.mikroe.com/

[22] (2018) *MIDE-51*. [en línea] http://www.opcube.com/home.html