Informática Retro: Emulación del Apple I (1976)

(IRA1)

Oskar Andrzej Stepien

Ingeniería de Computadores. 2018/19

Tutorizado por:

Vicente Á. García Alcántara / Samuel Barrado Aguirre

27/06/2019

Índice

Índice de Ilustraciones

Índice de Tablas

Objetivo

El objetivo del proyecto será recrear una máquina de los principios de la computación en un sistema más actual, combinándola con las tecnología de hoy en día. Este objetivo contempla una gran selección en ambos aspectos, y será conveniente encontrar la máquina "retro" adecuada y recrearla en un sistema adecuado.

Tras un análisis de las diferentes máquinas, desde un *Programma 101* (Contreras, s.f.) de 1962, primer ordenador personal, conocido por ser el ordenador con el cual se calculó el aterrizaje del Apollo 11 (Contreras, s.f.), hasta una *Commodore 64* (Velasco, Historia de la Tecnología: Commodore 64, 2012), el ordenador más conocido de las máquinas "retro", se va a realizar una emulación del primer ordenador de Apple (Apple Inc., 2019), el *Apple I* (The Apple-1, s.f.).

De manera similar, para el sistema de destino, donde se va a ejecutar esta recreación, se podría realizar desde un circuito impreso con todos los componentes soldados, siendo muy fiel a la realidad de la máquina, hasta simular su funcionamiento completamente por software. Derivando en ajustarse a un punto medio de ambos, y utilizar una placa FPGA (What is an FPGA?, 2019), esta es una placa de desarrollo de tecnología FPGA que está diseñada para la reprogramación de sus componentes y ofrece una buena comunidad en caso de encontrar algún error. Actualmente existen pocos puestos relacionados con las FPGAs, pero se ha analizado el mercado actual, y los mencionados circuitos parecen tener un esperanzador futuro (Streams, 2016).

Este tipo de hardware (PLD, o Dispositivo Lógico Programable) es muy versátil, pues permite replicar cualquier funcionamiento sin la necesidad de otros componentes en muchos casos. Dado que se ha optado por la realización en una FPGA por su gran interés en el futuro, se ha investigado y se ha encontrado un repositorio en GitHub, donde se realiza un emulador de está maquina en diferentes placas FPGA o de similar funcionamiento (Garfield, Moseley, & Otros, 2018).

Para alcanzar el objetivo indicado, se comenzará creando un emulador "básico" de Apple I, entendiendo sus componentes y programación de todo ello en un lenguaje de descripción hardware (HDL) como es Verilog, con su explicación necesaria. Con este emulador "básico" se propone comprobar el correcto funcionamiento del sistema antes de realizar mejoras en él.

Para modernizar el ya citado proyecto, se pretende incluir dispositivos comunes en la mayoría de los hogares, como un teclado USB o pantallas con entrada VGA. Ofreciendo como alternativa el uso de un ordenador para ambos propósitos.

Finalmente, para facilitar el sistema en comparativa con la máquina original, se incluirán un cargador de programas, sustituyendo el molesto cargador de casetes de la máquina original, y unas guías para un correcto uso de este proyecto, en forma de anexos.

Resumen

Este proyecto busca realizar una emulación del ordenador Apple I de Apple, del año 1976. Este ordenador destacaba del resto, al ser el pionero en usar circuitos integrados sobre la placa madre y ser vendida únicamente ella, reduciendo los costes de su fabricación.

Para lograr una emulación semejante al funcionamiento de la maquina original, se ha valorado su realización con distintas tecnologías, como Arduino, Raspberry o circuitos impresos, seleccionando finalmente su ejecución sobre una placa de desarrollo de FPGA, tecnología reprogramable y de fácil acceso. Con fin de conseguir este objetivo, se ha realizado un análisis de los componentes del Apple I, para replicar su trabajo usando un lenguaje de descripción de *hardware* (HDL) como es Verilog. Entre estos dispositivos se encuentra la Unidad Central de Procesamiento (CPU), la memoria del equipo (RAM y ROM) y los controladores de entrada y salida que disponía el sistema.

En este trabajo, al enfocarse en varios aspectos del *hardware* del Apple I, se ha realizado una mejora de un proyecto ya existente, y disponible en GitHub. De esta manera, se ha incorporado varios dispositivos actuales, como puede ser un teclado *USB* o una salida de video de interfaz *VGA.*

También, se ha buscado durante la realización de dichas mejoras, una implementación del casete original de la máquina original, en este caso, dentro de la memoria de la placa FPGA. Así, se indaga en el funcionamiento tanto de la máquina real como de la emulación realizada, comunicando sus aspectos más relevantes.

Todo ello se ha desarrollado teniendo en mente, los aspectos medioambientales y sociales que supone este proyecto, entre los cuales destacan, un menor consumo de energía y la facilitación de esta tecnología y sistema al entorno actual, con la finalidad de mostrar los avances de la sociedad.

Como punto final, se ha considerado de gran utilidad realizar varios anexos para entender con mayor precisión el funcionamiento de dicha máquina, y las peculiaridades del trabajo realizado, como pueden ser la instalación de un nuevo programa o una instalación sobre la placa FPGA seleccionada.

Abstract

This project seeks to perform an emulation of Apple's Apple I computer from 1976. This computer stood out from the rest, being the pioneer in using integrated circuits on the motherboard and being sold only it, reducing the costs of its manufacture.

To achieve an emulation like the operation of the original machine, its realization has been valued with different technologies, such as Arduino, Raspberry or printed circuits, finally selecting its execution on a development board of FPGA, reprogrammable and easily accessible technology. In order to achieve this goal, an analysis of the components of the Apple I has been performed, to replicate its work using a Hardware Description Language (HDL) as is Verilog. These devices include the Central Processing Unit (CPU), the computer memory (RAM and ROM) and the input and output drivers that the system had.

In this work, focusing on various aspects of Apple I hardware, has been done an upgrade of an existing project, and available in GitHub. In this way, several current devices have been incorporated, such as a USB keyboard or a VGA interface video output.

Also, during the realization of these improvements, an implementation of the original cassette of the original machine, in this case, within the memory of the FPGA board, has been sought. Thus, the operation of both the actual machine and the emulation performed are investigated, communicating its most relevant aspects.

All of this has been developed with the environmental and social aspects of this project in mind, among which are less energy consumption and the provision of this technology and system to the current environment, in order to show the progress of society.

As a final point, it has been considered very useful to make several annexes in order to understand more accurately the operation of this machine, and the peculiarities of the work carried out, such as the installation of a new program or installation on the selected FPGA board.

Motivación

La motivación que lleva a realizar un emulador de esta mítica máquina fue la innovación que la llevo a ser conocida, y por el interés del autor por las maquinas “retro”. Entre las múltiples opciones de máquinas seleccionadas, se eligió el Apple I, debido al gran cariño que le aprecia Steve Wozniak (cofundador de Apple), siendo la maquina completamente de su diseño e ingenio.

Para el autor, el Apple I es una historia de cómo enfrentarse a las grandes compañías de la década de los 70, y tener éxito. Es interesante la ambición de Steve Wozniak (Martín, 2018), cofundador de Apple Inc., el cual consiguió crear una maquina diseñada por él mismo.

Otra alternativa interesante sería realizar el Apple II, el gran éxito de la marca Apple de la década de los 70 (The Apple II, s.f.), pero se opina que esta última no supone un cambio tan influyente, con el añadido que el Apple I no disponía de S.O. como tal, haciéndolo más llamativo. También la intriga por el funcionamiento de los componentes, aunque sean por medio de una emulación, puede ser interesante y propone un desafío.

1. Marco del problema

El Apple I (The Apple-1, s.f.) fue de los primeros ordenadores personales de la historia, y el primero de la marca Apple (Apple Inc., 2019), fundada por Steve Jobs (Steve Jobs, 2019) y Steve Wozniak (Wozniak, 2019). Una de las características más icónicas de este ordenador es que sólo vendían la placa madre, tal y como se puede ver en la Ilustración 1, eliminando los costes de la carcasa, pantalla y teclado que tenían otros ordenadores. Gracias a ello, se redujo en gran parte el coste del ordenador, y al ser el primero de la marca Apple, fue muy limitado en ventas y sólo a tiendas específicas. Actualmente, se conservan pocas unidades que estén a la venta en subastas o incluso que sigan en posesión de museos o particulares, dado que es un ordenador de 1976 y fue descatalogado meses después.

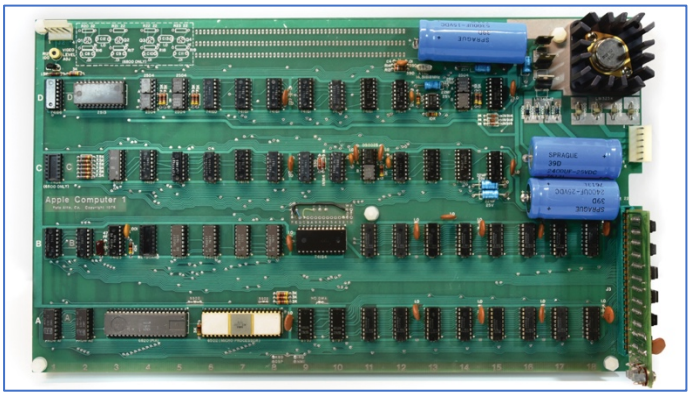


Ilustración 1: Apple I (Vilches, 2018)

El Apple I o Apple One, fue un gran cambio en los ordenadores porque fue el primero en usar un teclado y pantalla externos a la máquina, con el objetivo de reducir los costes de la máquina, guardando un gran parecido a comprar un ordenador actualmente. Aunque no sea el primer ordenador personal de la historia es el producto fundador de Apple Inc.

2. Alternativas Existentes

Para la realización de este proyecto, se ha buscado información y otros proyectos de carácter similar, estos serán la base para comenzar a realizar el trabajo, pues algunos ofrecen un punto de apoyo en términos de CPU o "controladores" de video, que facilitarán el trabajo a realizar, adaptándose los mismos, cuando sea necesario, en el desarrollo del proyecto.

2.1. Replica I

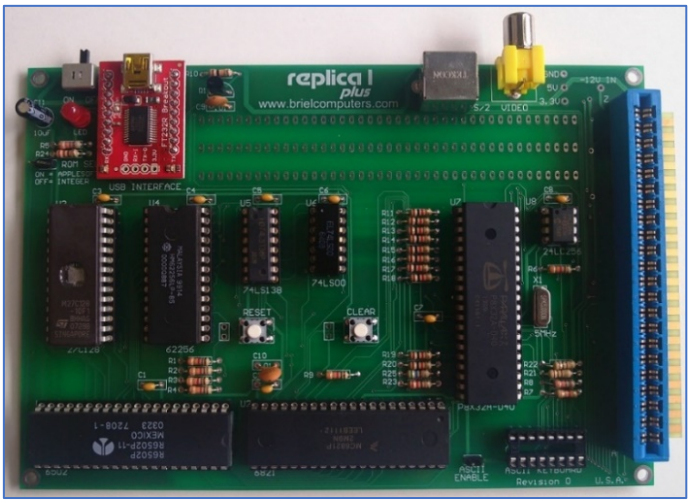


Ilustración 2: Replica I de Briel Computers (Briel, Replica 1, 2017)

El primer proyecto en el cual se desarrolla el Apple I, es un Replica 1 (Briel, Replica 1, 2017) de Briel Computers (Briel, Homepage, 2019) , esta placa es un clon "oficial" del Apple 1, que su diseñador original, Steve Wozniak ha permitido realizar. Es uno de los primeros proyectos que se encontró para obtener diferentes formas de realizarlo. En este caso, Replica 1, es un kit de montaje, para tener una placa muy cercana a la realidad del Apple 1. Esta alternativa, tiene como elementos básicos circuitos lógicos específicos (muy similar al modelo original), tal y como aparece en la Ilustración 2.

Es el proyecto más cercano al cual se puede llegar del Apple 1,pues su funcionamiento es casi igual que el citado Apple 1, teniendo en cuenta que utiliza el procesador MOS6502 (MOS6502, s.f.), microprocesador original, y otros circuitos integrados que el mencionado computador a emular utilizaba. El año de su salida al mercado fue 2003, y son kits que actualmente están muy limitados debido a la falta de distribuidores. El tamaño del Replica 1 es más reducido que el original, pero ofrece algunas mejoras en su funcionamiento, como un puerto PS/2 , para un teclado de la época, o su firmware (software contenido en la placa necesario para su funcionamiento).

2.2. Apple I en una FPGA

Ilustración 3: Emulación de Apple I en FPGA (Alangarf) (Garfield, Moseley, & Otros, 2018)

El siguiente proyecto analizado es el realizado por Alan Garfield y otros colaboradores, disponible en GitHub (Garfield, Moseley, & Otros, 2018). En esta implementación, el hardware empleado para la realización del emulador es una placa FPGA como puede observarse en la Ilustración 3, ofreciendo algunas mejoras como salida de video por VGA y conexión de teclado por PS/2. En esta alternativa, se han realizado varias versiones para distintas placas FPGA como pueden ser Spartan-3E Starter Kit (Spartan 3E Starter Kit, s.f.) o Terasic DE0 (Terasic, s.f.). Es un proyecto que ofrece varias alternativas para realizar en placas FPGA usando un lenguaje en HDL, en este caso, Verilog, aunque podría implementarse en VHDL sin más que traducir de un lenguaje en otro.

2.3. ESSPLE

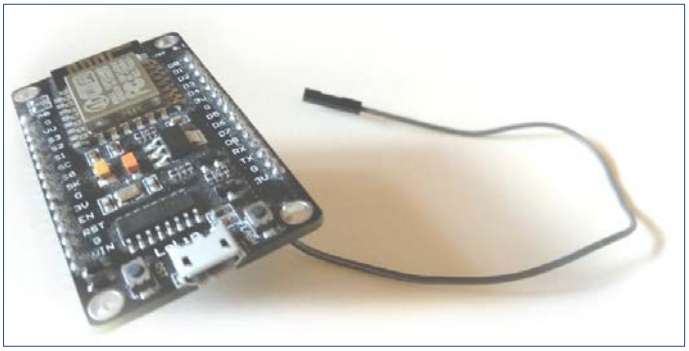


Ilustración 4: Emulación del Apple I en ESP8266 (Čavrak H. , 2018)

Otro proyecto de similares características es ESPPLE (Čavrak H. , 2018), realizado por Hrvoje Cavrak (Čavrak H. , 2019), siendo en este caso, la base para la implementación un módulo Wifi ESP8266 (ESP8266, 2019), como puede observarse en la Ilustración 4, usando su capacidad Wifi, permite una interfaz con el usuario a través de un Terminal vía WiFi mientras lo muestra en una pantalla NTSC/PAL conectada a él. Es un proyecto muy compacto, al medir la placa menos de 5cm de largo, pero más complejo en su desarrollo.

2.4. DINAAO

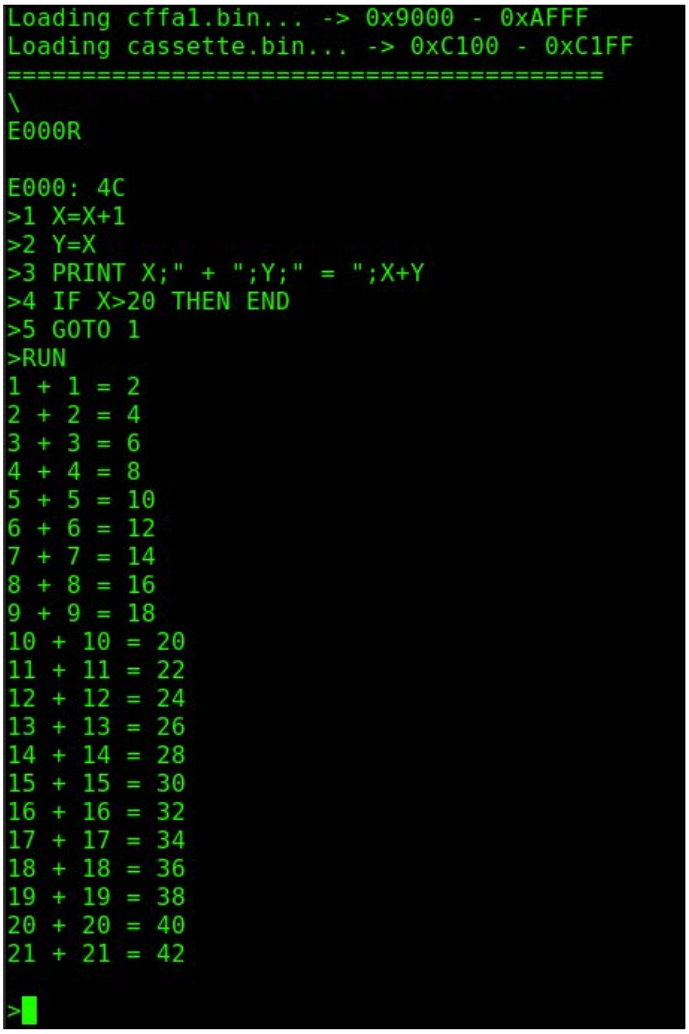


Ilustración 5: Simulación de DINAAO

Por último, se ha tenido en cuenta un simulador del Apple I, DINAAO (jgilbert, Dinaao, A new Apple1 emulator., 2008), que son las siglas de "Dinaao Is Not An Apple One", desarrollado por "jhilbert" (jgilbert, jgilbert, s.f.), que simula bastante fielmente al original, siendo más ágil, pudiendo servir para probar códigos originales del Apple 1 o ver algunas propiedades que ofrecía en la década de los 70 este último, mostrándose el resultado de la Ilustración 5.

Tanto el último como el anterior, han sido proyectos que se han probado por cuenta propia para ver su funcionamiento, ambos con un resultado correcto, pero sin respetar la velocidad real del Apple I, que es complicada de emular.

2.5. Apple II en una FPGA

Ilustración 6: Apple II (Sanchez Hidalgo, 2017)

También se ha analizado desarrollar el emulador con otra máquina como objetivo, como puede ser el Apple II (Sanchez Hidalgo, 2017), que se muestra en la Ilustración 6. Esta máquina fue el verdadero éxito de Apple Computers, siendo su primer ordenador fabricado en masa, dado que el Apple I fue un ordenador con escasa distribución. El proyecto que se ha encontrado, es un emulador del Apple II+, sucesor del Apple II, pero con las misma estructura (Edwards, 2007). Este desarrollo es muy interesante, al ser el primer ordenador Apple con AppleDOS (Apple Disk Operating System) (AppleDOS, s.f.), precursor del MacOS (MacOS, 2019), pero que no consigue reflejar tan claramente la pasión que se utilizó en el Apple I. Esta realizado en VHDL, con un funcionamiento similar al Apple I, dado que utilizaba casi los mismos componentes, CPU 6502, 4 KB de RAM iniciales, ampliables, pero con la ventaja de disponer, además del sistema operativo, de una interfaz gráfica atractiva. Se podría añadir de manera relativamente sencilla, esta funcionalidad en la emulación actual del Apple I, pero se requeriría de una modificación del código del Apple II a Verilog. De esta manera se evitaría la mezcla de códigos VHDL con Verilog, tal y como se puede apreciar en la Ilustración 7.

Ilustración 7: Emulación del Apple II en una placa FPGA (Edwards, 2007)

El rechazo de la inclusión de este sistema, no quiere indicar que no sea interesante, emular el mismo que, además, podría resultar más atractivo al disponer de interfaz gráfica y las ventajas que ello aporta pero no alcanza el carisma en la invención y desarrollo que tiene el limitado Apple I.

2.6. Videos del funcionamiento del Apple I

Al no disponer de un modelo Apple I original, para el análisis del comportamiento del Apple I, se tendrán en cuenta, varios videos del funcionamiento del mismo, que se puede ver en varios videos subidos a YouTube (YouTube, 2019) y que son citados cuando se mencionan.

Ilustración 8: Frame del video de funcionamiento del Apple I en Boston (BREKER KÖLN, 2012)

En el primer video se muestra una ejecución sobre el Apple I de un programa “Hello World” (fotograma del mismo en la Ilustración 8) publicado por la casa de subastas Breker de Boston (Breker, 2019).

En el siguiente video (fotograma en la Ilustración 9), aparece la emulación del código de un Apple I que es visualizado en el monitor del Apple III.

Ilustración 9: Emulación de un código del Apple I en video (alker33, 2012)

3. Metodología de trabajo

Para iniciar el trabajo se han valorado varias alternativas hardware con el que implementar el proyecto, eligiendo las placas FPGA gracias a su gran auge y versatilidad en proyectos similares, teniendo también en cuenta de la disposición de suficiente tamaño para almacenar los códigos referentes a la máquina.

Como un comienzo ante la metodología del trabajo, se ha considerado importante, elegir una FPGA en la cual se disponga de suficientes recursos y no existan complicaciones en el desarrollo posterior.

Para evitar esos inconvenientes, se comenzará por elegir en primer lugar, antes del desarrollo y explicación de la metodología del proyecto, la placa que mejor pueda asegurar el funcionamiento del Apple I y donde se pueda añadir nuevas funcionalidades, en caso de considerarse mejoras útiles para el proyecto desarrollado.

3.1. Placa FPGA

La primera selección que se hace es la de emplear una placa de desarrollo donde viene integrada una FPGA. Se toma esta opción puesto que dispone de varios puertos donde conectar los periféricos, además de requerir únicamente la escritura del código que se va a utilizar, estando disponible para su uso. El tutor, Vicente A. García Alcántara, propuso tres placas FPGAs que podrían ser utilizadas: Basys-3 Artix-7, Zybo Zynq-7000 y Nexys-4 DDR. Su recomendación fue utilizar la placa Zybo por sus puertos para periféricos, que disponía de una mayor cantidad que el resto de las placas propuestas. A pesar de ello, se decidió valorar todas las placas FPGA que se han nombrado previamente. Cómo se han analizado dos baterías de placas, se muestran las siguientes tablas con la información más útil que servirá para determinar cuál es la placa, de las estudiadas, que mejor se adapta al proyecto a desarrollar.

3.1.1. Primera batería de placas FPGA

Esta primera batería de placas, fueron las que se seleccionó en un principio para implementar el proyecto sobre ellas, pero por una incompatibilidad de ficheros inicial, no ha sido posible utilizarlas, aun así, se ha considerado necesario comparar estas placas, porque podrían servir en caso de una reimplementación del proyecto.

En la siguiente tabla (Tabla 1), se comparan tres de estas placas, con el objetivo de seleccionar una.

	Basys-3 Artix-7 (Basys-3 Artyx-7, s.f.)	**Zybo Zynq-7000** (Zybo Zynq-7000, s.f.)	**Nexys-4 DDR** (Nexys-4 Artyx-7, s.f.)
Reloj	450 MHz	650 MHz	450 MHz
RAM	225 KB	240 KB	600 KB

	Basys-3 Artix-7 (Basys-3 Artyx-7, s.f.)	**Zybo Zynq-7000** (Zybo Zynq-7000, s.f.)	**Nexys-4 DDR** (Nexys-4 Artyx-7, s.f.)
Input/Output	16 switches, 16 LEDs, 3 puertos PMOD y 5 botones. Además de USB-UART. También dispone de un display de 4 dígitos.	4 switches, 4 LEDs, 6 puertos PMOD y 6 botones. Además de USB-UART y lector de tarjetas SD.	16 switches, 16 LEDs, 5 puertos PMOD y 5 botones. También dispone de un USB-UART y un display de hasta 8 dígitos.
Conexión de pantalla	Puerto VGA de 12 bits	Puerto VGA de 16 bits y puerto HDMI.	Puerto VGA de 12 bits
Mejor característica	La mejor característica son la gran cantidad de puertos de Entrada/Salida, llegando a ser excesivo en este proyecto.	La mejor característica es la posibilidad de conectar tarjetas SD y varias formas de visualizar la pantalla. También permite instalar Petalinux, un Sistema Operativo para esta FPGA.	Al igual que la placa FPGA Basys-3, su mejor característica es la gran cantidad de puertos Entrada/Salida, llegando a ser molesto en el uso. También su gran capacidad RAM
Imagen	Ilustración 10: Placa FPGA Basys-3 de Xilinx	Ilustración 11: Placa FPGA de Xilinx Zybo Zynq-7000	Ilustración 12: Placa FPGA de Xilinx, Nexys-4
Selección	Zybo Zynq-7000, como se comentará en “Resolución de primera batería de placas FPGA”.		

Tabla 1: Comparativa de la primera batería de placas FPGA

3.1.1.1. Basys-3 Artix-7

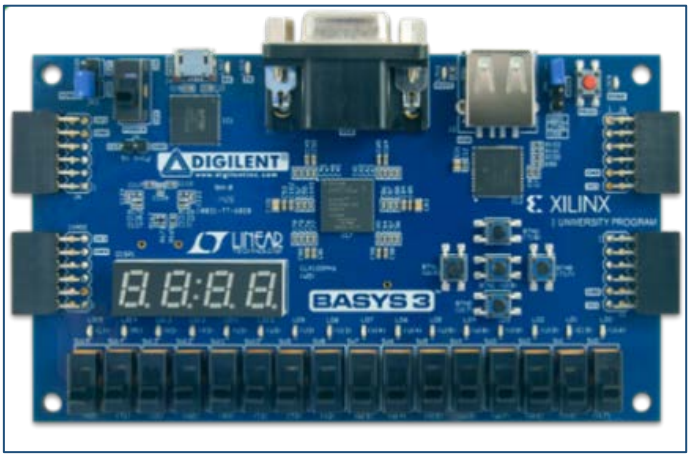


Ilustración 10: Placa FPGA Basys-3 de Xilinx (Basys-3 Artyx-7, s.f.)

3.1.1.2. Zybo Zynq-7000

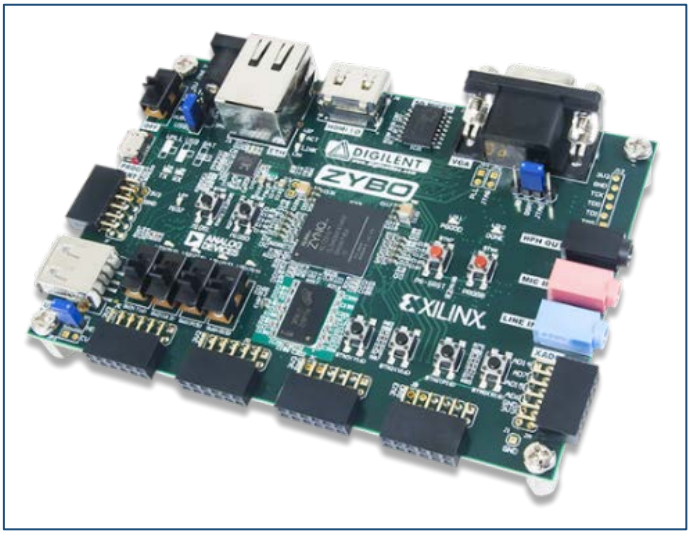


Ilustración 11: Placa FPGA de Xilinx Zybo Zynq-7000 (Zybo Zynq-7000, s.f.)

3.1.1.3. Nexys-4 DDR

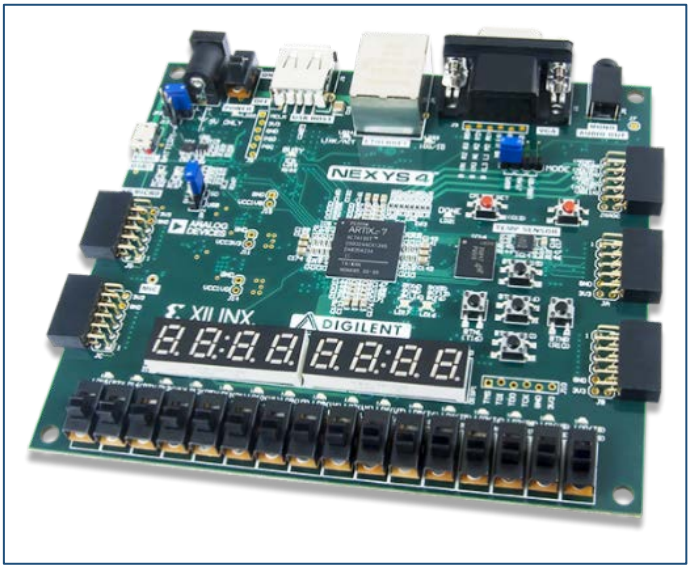


Ilustración 12: Placa FPGA de Xilinx, Nexys-4 (Nexys-4 Artyx-7, s.f.)

3.1.1.4. Resolución de primera batería de placas FPGA

Una vez comentadas estas características, la placa que ofrece más posibilidades para emular el Apple I objetivo es Zybo Zynq-7000, dado que incorpora puertos USB, HDMI, Ethernet, VGA y dispone de varios PMod para utilizarlos en caso de utilizar una carcasa para el proyecto.

Durante un primer contacto con el software que se tenía pensado utilizar era Vivado (Vivado Design Suite, 2019), esta interfaz para reconfigurar placas FPGAs de Xilinx (Xilinx, 2019), ofrece mucha ayuda para placas de la gama 7 de dicho fabricante (Artyx-7, Spartan-7, Zynq-7xxx).

Por ese motivo, en un primer contacto con el software Vivado, este no es capaz de reconocer determinados ficheros del tipo .HEX, los cuales son claves en el proyecto, al crear un fichero ficticio con la memoria del Apple I. Esto a pesar de un intento de solución del co-creador del proyecto de GitHub de *apple-one*, Niels Moseley, quien ha trabajado con una placa Spartan-3E de Xilinx en su repositorio del proyecto (Moseley, 2018), informa de la "exclusividad" que exige el software Xilinx ISE

(ISE Design Suite, 2018) de Xilinx (Xilinx, 2019). Supone que el software Vivado, es más "exclusivo" respecto a los tipos de los archivos que acepta, que en el caso de Vivado, sólo es capaz de aceptar archivos y cabeceras de formato propio, y se ha considerado evitar este software más "moderno" y optar por Xilinx (Xilinx, 2019) ISE (ISE Design Suite, 2018).

Este nuevo software, que también se va a comentar en un apartado más adelante, es menos exigente que el software Vivado (Vivado Design Suite, 2019), y permite los archivos de tipo .HEX, poniendo la condición que ocupen un tamaño múltiplo de 1024 bytes o 1 KB. Esta condición no supone un gran problema, ya que se dispone de un tamaño suficiente. Pero es necesario volver a analizar nuevas placas para el proyecto en desarrollo.

3.1.2. Segunda Batería de placas FPGA

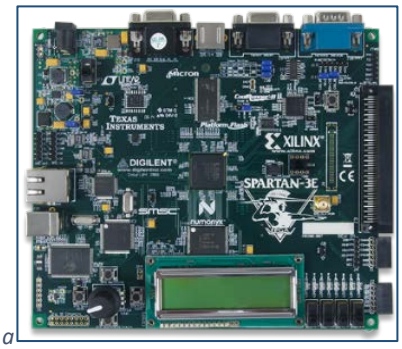
Para esta segunda batería de placas FPGA, se ha decidido utilizar placas que el departamento de Sistemas Informáticos, DSI, donde se realiza este proyecto, dispone y se pueden utilizar para la implementación del mismo, y funcionan con Xilinx ISE. Esto facilita al admitir el formato de tipo .HEX en los proyectos. Las placas para esta segunda batería de análisis son: Spartan-3E Starter Kit, Spartan-3E 1600, Atlys Spartan-6 y Nexys-3 Spartan-6. Las dos placas Spartan-3E únicamente se diferencian en el número de bloques lógicos que disponen. En la siguiente tabla (Tabla 2) se muestra una comparación de estas cuatro nuevas placas seleccionadas.

	Spartan-3E Starter Kit, Spartan-3E 1600 (Spartan 3E Starter Kit, s.f.) (Spartan 3E 1600, s.f.)	**Atlys Spartan-6** (Atlys Spartan-6, s.f.)	**Nexys-3 Spartan-6** (Nexys-3 Spartan-6, s.f.)
Reloj	100 MHz	100 MHz (Hasta 500 MHz)	100 MHz
RAM	64 MB	128 MB + 16 MB Flash	3 bloques de 16 MB
Input/Output	4 switches, 16 LEDs, 2 puertos PMOD y 5 botones. Además de USB-UART. También dispone de un display de 16 dígitos y dos filas y de un puerto PS/2.	8 switches, 8 LEDs, 1 puertos PMOD y 5 botones. Además dispone de USB-UART y soporta Petalinux.	8 switches, 8 LEDs, 4 puertos PMOD y 5 botones. También dispone de un USB-UART, un display de hasta 4 dígitos y de un puerto USB para el teclado.
Conexión de pantalla	Puerto VGA	4 puertos HDMI, de los cuales hay dos de entrada de datos vía HDMI y otros dos de salida de video.	Puerto VGA de 8 bits

	Spartan-3E Starter Kit, Spartan-3E 1600	Atlys Spartan-6	Nexys-3 Spartan-6
Mejor característica	La mejor característica es la garantía del funcionamiento del emulador creado	La mejor característica es la disposición de 2 puertos HDMI para usar de pantalla. También dispone de un almacenamiento grande y un reloj modificable.	Su mejor característica es la disposición de un puerto USB y una moderada cantidad de puertos entrada/salida.
Imagen	Ilustración 13a: Placa FPGA Spartan-3E Starter Kit	Ilustración 14: Placa FPGA Atlys Spartan-6 (Atlys Spartan-6, s.f.)	Ilustración 15: Placa FPGA Nexys-3 Spartan-6
Selección	Nexys-3 Spartan-6, como se comentará en "Resolución de la segunda batería de placas FPGA"		

Tabla 2: Comparativa de la segunda batería de placas FPGA

3.1.2.1. *Spartan-3E Starter Kit, Spartan-3E 1600*

a 

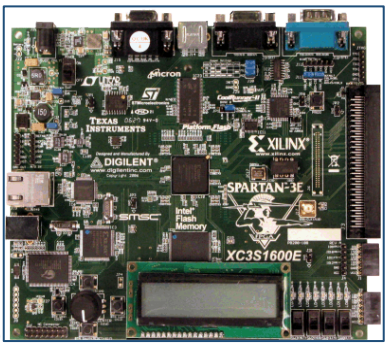
b 

Ilustración 13a: Placa FPGA Spartan-3E Starter Kit (Spartan 3E Starter Kit, s.f.)

Ilustración 13b: Placa Spartan-3E 1600 (Spartan 3E 1600, s.f.)

3.1.2.2. *Atlys Spartan-6*

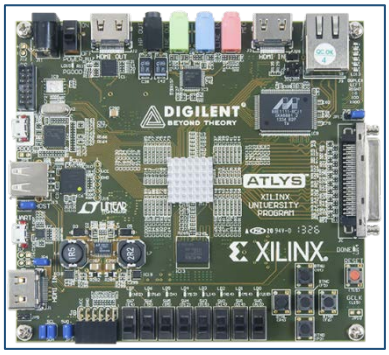


Ilustración 14: Placa FPGA Atlys Spartan-6 (Atlys Spartan-6, s.f.)

3.1.2.3. Nexys-3 Spartan-6

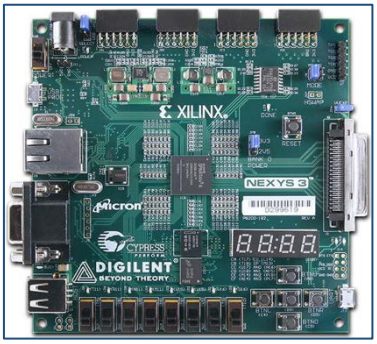


Ilustración 15: Placa FPGA Nexys-3 Spartan-6 (Nexys-3 Spartan-6, s.f.)

3.1.2.4. Resolución de la segunda batería de placas FPGA

En esta segunda batería de placas FPGA, se ha decidido seleccionar la placa Nexys-3 Spartan-6 para la realización del proyecto actual, siendo el punto intermedio entre el proyecto realizado, y un proyecto ideal, que utilice únicamente tecnologías actuales (HDMI, DisplayPort, USB 3.0, Bluetooth, Wifi).

Se ha decidido descartar la placa Atlys, por no saber cómo utilizar los puertos HDMI de entrada y la salida de video también por esta tecnología, además de la falta de puertos PMod. La placa Spartan-3E, se consideraría como la placa final, en caso de resultar completamente imposible la utilización de la placa Nexys-3, teniendo en cuenta que su programación no dista en gran escala de la realizada por Niels Moseley (Moseley, 2018).

3.1.3. Resolución Final

En este último apartado relacionado con las placas FPGA, se ha decidido comparar las mejores placas de cada batería, en este caso, Zybo Zynq-7000 y Nexys-3 Spartan-6. Aunque se haya decidido elegir la placa Nexys-3 debido a la incompatibilidad de los ficheros, este cambio ha mejorado el desarrollo del proyecto, al ofrecer más almacenamiento en la placa. En la siguiente tabla (Tabla 3), se todos los factores que se han valorado y la resolución final. Asimismo, se puede ver una ilustración de la placa seleccionada, donde se indican los componentes que se van a utilizar en el proyecto (Ilustración 16).

	Zybo Zynq-7000	Nexys-3 Spartan-6
Reloj	650 MHz	100 MHz
RAM	240 KB	3 bloques de 16 MB
Input/Output	4 switches, 4 LEDs, 6 puertos PMOD y 6 botones. Además de USB-UART y lector de tarjetas SD.	8 switches, 8 LEDs, 4 puertos PMOD y 5 botones. También dispone de un USB-UART, un display de hasta 4 dígitos y de un puerto USB para el teclado.
Conexión de pantalla	Puerto VGA de 16 bits y puerto HDMI.	Puerto VGA de 8 bits

	Zybo Zynq-7000	Nexys-3 Spartan-6
Mejor característica	La mejor característica es la posibilidad de conectar tarjetas SD y varias formas de visualizar la pantalla. También permite instalar Petalinux	Su mejor característica es la disposición de un puerto USB y una moderada cantidad de puertos entrada/salida.
Imagen	Ilustración 11: Placa FPGA de Xilinx Zybo Zynq-7000	Ilustración 15: Placa FPGA Nexys-3 Spartan-6
Resolución	Como selección final, se ha elegido la placa FPGA Nexys-3 Spartan-6, debido a la compatibilidad de archivos .HEX y para realizar una adaptación ligera de las actuales tecnologías, además de disponer de mayor memoria.	

Tabla 3: Comparativa de las mejores placas FPGA de cada batería

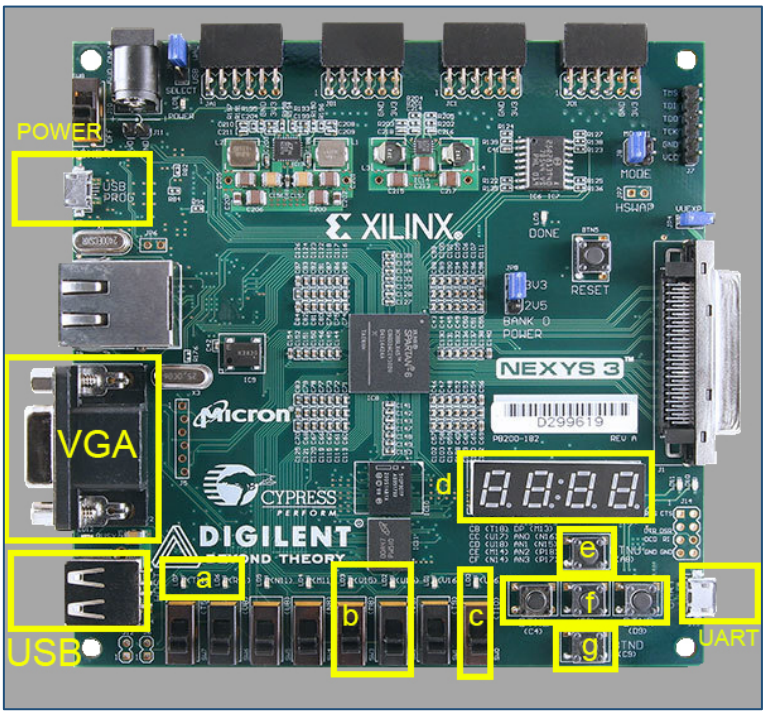


Ilustración 16: Placa FPGA utilizada, Nexys 3

3.2. Apple I

Antes de explicar el código del Apple One (Garfield, Moseley, & Otros, 2018) desarrollado por Alan Garfield y Niels Moseley, se ha considerado conveniente explicar las características de esta máquina, que fue la pionera de Apple Inc. (Apple Inc., 2019).

Ilustración 17:Apple I con carcasa y teclado (Apple I Case, s.f.)

El Apple I (Ilustración 18), diseñado por Stephen Wozniak, es un ordenador personal creado en 1976 e inició el curso actual de la compañía Apple Inc. Su principal característica, era que solo se componía de la placa base, necesitando el usuario, un monitor, una fuente, un teclado ASCII y un lector de casetes, en caso de querer cargar BASIC (BASIC, 2019) u otro software de Apple. Todos estos componentes no eran originales de la empresa, permitiendo a los usuarios añadir dichos dispositivos, incluso de bajo coste, para disfrutar de la novedad en ordenadores resultando, desde el punto de vista económico, tremendamente atractiva. En la siguiente Ilustración 17 se puede ver un Apple I con una carcasa hecha de madera, un teclado, una toma de corriente y con espacio para el monitor.

Sus características eran:

- Microprocesador MOS6502 de 8 bits, de 1 MHz. (MOS6502, s.f.), que fue una gran novedad en el mundo de la computación de la época, porque disponía de una gran funcionalidad, a un precio que ni Intel no podía ofrecer con su producto Intel 8080 (Alpern, s.f.) ni Motorola y su MC6800 (Motorola 6800, 2018), comparado con el que disponía en su diseño el Apple I de MOS Technology (Matthews, 2003).
- Funcionamiento en 8 bits.
- RAM desde 4 KB, que es el tamaño que ofrecía el Apple I inicialmente, con posibilidad de, posteriormente, añadir un módulo de 8 KB, para que pudiera ejecutar BASIC. Pero que permitía instalar módulos RAM hasta llegar a 64 KB con módulos de memoria no oficiales de Apple, aunque nunca superó los 48 KB.
- Display monocromo, conectado con un interfaz PAL/NTSC, que permitía 40x24 caracteres en pantalla como máximo.
- Memoria ROM de 256 Bytes, donde estaba grabado el firmware del Apple I.

Aunque no se considere un ordenador potente, fue el ordenador personal que comenzó la cartera de productos de Apple Inc., y sirvió de punto de partida para sus otros éxitos como el Apple II, Macintosh, entre otros, pero que también se incluyó en los fracasos de la compañía americana, como el Apple III o Apple LISA.

La gama Apple, puede ser también conocida por la variedad y diferentes prestaciones que ofrece:

- Apple I, un éxito en su época, pero con un volumen de ventas muy limitado.
- Apple II, el gran éxito de Apple Inc. de la década de los 70, que ya disponía de un sistema operativo DOS, Apple DOS (AppleDOS, s.f.).
- Apple III, un fracaso que casi arruina la compañía por su poca innovación con respecto a Apple II.
- Posteriormente comercializaron el Apple IIe, una mejora del Apple II con mejoras sobre lo que ofrecía el Apple II.
- Apple LISA, añadió muchas mejoras como un ratón y muchas funcionalidades, pero a un precio excesivo.

A pesar de ser conocido con la típica carcasa de madera (Ilustración 17), esta nunca fue oficial, al ser únicamente la placa base lo que se vendía , como ya se mencionó.

Otra información interesante sobre el Apple I es que fue uno de los primeros computadores en usar un microprocesador del fabricante MOS Technology (el mencionado MOS6502) y que fue pionero en la incorporación de circuitos integrados en la placa base del ordenador, significando un gran avance en la computación de la época. Todo ello, lleva a pensar que, posiblemente, fue uno de los mayores hitos en la historia de los ordenadores personales.

3.3. Código Apple One

Como ya se ha comentado previamente en el apartado “Placa FPGA”, para la realización del código se ha tomado como base un proyecto ya realizado y comentado en el apartado “Alternativas Existentes”.

El código del cual se parte corresponde a una modificación del Apple I original, desarrollada en una placa FPGA que aporta novedades respecto a la maquina original, entre las cuales está la utilización de un teclado con interfaz PS/2, en lugar del teclado por “*socket de 16 pines”* (Ilustración 18b) del cual disponía el ordenador real. También incorpora una salida por medio de un puerto VGA, más común que el puerto de 4 pines (Ilustración 18b) que utilizaba el modelo original, y que era utilizado con un cable de Video Compuesto, muy conocidos en la época (Ilustración 19).

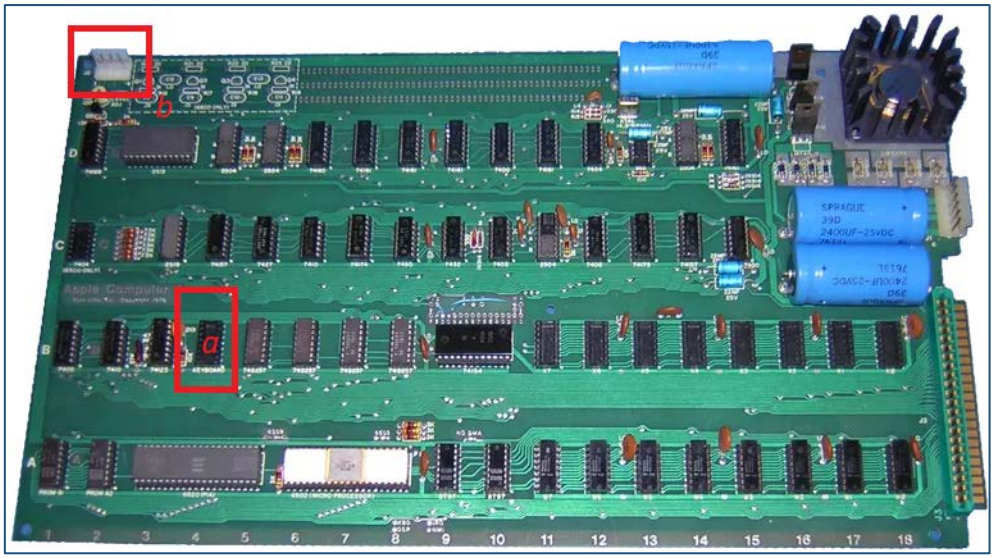


Ilustración 18: a) Conector de Teclado 16 pines;

b) Video del Apple I original, para conectar Video Compuesto (The Apple-1, s.f.)

Ilustración 19: Cable Video Compuesto (Cmple, 2019)

Este será el código de partida, que será modificado para aumentar la "personalización" y "mejora" del Apple I, sin perder su estética "retro", aprovechando también el reducido consumo eléctrico de las FPGAs, alcanzándose así, una alternativa para el desarrollo del proyecto bastante viable.

El código del Apple I que se va a utilizar está compuesto por los siguientes programas, se va a ofrecer una breve explicación sobre el contenido de cada código.

- "apple1_nexys3_top.v":
  Código exclusivo para esta placa, se encarga de ajustar el reloj del sistema a las necesidades del resto de códigos, además de establecer las entradas y salidas del sistema.
- "apple1.v":
  Coordina todos los componentes de la máquina para su correcto funcionamiento.
- "arlet_6502.v":
  Emula el funcionamiento del microprocesador MOS 6502 (MOS6502, s.f.), dispone de todas las instrucciones que este dispone, del mismo modo, ofrece de la Unidad Aritmeticológica (ALU).

- "ram.v":
  Refleja el funcionamiento de la memoria RAM del ordenador real, permitiendo introducir y recoger los valores almacenados. Disponen de un funcionamiento similar los códigos "rom_wozmon.v" y "rom_basic.v", donde los valores ya vienen definidos y únicamente permiten obtener los datos de la memoria que emulan, no son modificables.
- "uart.v":
  Contiene las funcionalidad del componente UART (UART Protocol Validation Service, s.f.) de la placa, entre ellas la de recibir y transmitir datos.
- "ps2keyboard.v":
  Controlador del teclado, ofrece la traducción a codificación ASCII de las teclas de un teclado PS2 o USB con distribución americana.
- "vga.v":
  Permite la salida por medio del dispositivo VGA de la placa, con una resolución de pantalla de 640 x 480 pixeles.

Todos estos códigos se detallan en un apartado posterior junto con las modificaciones realizadas.

3.4. Compatibilidad

Se han realizado pruebas para comprobar si la placa FPGA seleccionada es la adecuada o si requiere modificaciones.

Se ha determinado de manera objetiva que la placa que mejor se adapta para implementar las "mejoras" de la máquina Apple I, añadiendo funcionalidades para facilitar su uso en la época actual, sin perder el estilo "retro" o "vintage" que caracterizaba a la placa del Apple I es la placa de desarrollo seleccionada (como ya se vio en el capítulo correspondiente).

Las partes que se han analizado para comprobar su compatibilidad son:

- Suficiente memoria flash para almacenar el programa completo y todos sus añadidos, verificándose que de los 16 MB de memoria SPI Flash y otros tanto de memoria RAM y BPI Flash son suficientes. El programa actualmente, sin contar añadidos supera un 5-10% de la memoria total, siendo este parámetro general, sin poder especificar la ocupación real.

- Disponibilidad de un puerto USB, VGA, UART, Switches, Botones y PMODs para futuros cambios.
  La entrada USB que viene por defecto instala en la placa, sustituye la conexión PS/2 que se piensa quitar del proyecto inicial, para conectar un teclado vía USB.
  La salida VGA, para evitar realizar un código a ciegas para una salida HDMI dado la experiencia nula de su funcionamiento, aparte de sus pines y frecuencia. La salida VGA ofrece una salida más clásica, pero aún vigente en la mayoría de los monitores de ordenador.
  La interfaz UART, tiene el objetivo de permitir, en caso de falta de un teclado USB, acepte una entrada por medio de un terminal y un puerto microUSB.
  Los conmutadores y botones están previstos utilizarse para realizar modificaciones sobre el modo de interacción con la placa y Apple I, además de la incorporación de mejoras para futuras líneas.

Por último, los puertos PMOD, se requerirán en caso del añadido de una carcasa del estilo Apple I, para realizar las conexiones necesarias para su funcionamiento.

En resumen, la compatibilidad con el código ya realizado, ha facilitado los posibles añadidos y ha permitido un aprendizaje sobre la placa Nexys-3 en cuestión. Este aprendizaje incluye, que cambios deberían ser necesarios y cuáles podrían ser las modificaciones más interesantes en el proyecto final. Un posible inconveniente sobre la compatibilidad fue la idea, errónea, de suponer que la tecnología PS/2 y USB son incompatibles, y los cambios que han afectado en la tecnología VGA, que ha cambiado en un breve periodo, de manera considerable, en las placas FPGAs.

4. Desarrollo del proyecto

Una vez detallada la metodología detallada previamente, es necesario realizar un análisis de los códigos, conociendo la precisión en funcionamiento con el componente real. También se va a explicar de forma concisa cualquier añadido que se consideraba necesario y los cambios realizados.

4.1. Metodología

Se ha creído conveniente resaltar la metodología que se piensa realizar de manera más detallada. Esta metodología dependerá del tiempo aplicado y el funcionamiento del programa.

- Aprendizaje sobre el Apple I y su historia.
- Hacer una ejecución de prueba del código inicial en la placa seleccionada: Permitiendo un aprendizaje sobre la tecnología utilizada en el código desarrollado en lenguaje Verilog. Del mismo modo permitirá conocer las capacidades y funcionamiento de la placa seleccionada, antes de comenzar el desarrollo de los añadidos que se prevén.
- Analizar la efectividad y precisión en funcionamiento de las partes programadas y su circuito real.
- Añadir un controlador teclado vía USB: En caso de no ser funcional o válido el código desarrollado para el teclado PS/2, se creará un controlador para un teclado que funcione vía USB. Este funcionará de manera similar al código ya realizado, y podrá agilizar posibles pruebas futuras.
- Ofrecer la posibilitar de limpiar la pantalla y reiniciar la maquina: Dado la escasez de botones en las placas donde ya se ha desarrollado el código, se pretende ofrecer al usuario esa capacidad, con botones para hacer un “Reset” de la máquina Apple, uno del sistema y otro botón para la limpieza de la pantalla “Clear” para vaciar el contenido de la pantalla.
- Analizar la necesidad de un menú de “Ajustes” y realizarlo: En caso de requerir una pantalla para ajustar ciertos parámetros, se va a analizar la utilidad que podría ofrecer para cargar programas, modificar resolución o aceptar algunos cambios que se consideran necesarios. O su implementación en “caliente”, estando la maquina activa, realizar estos cambios, de manera similar a la realidad, ahorrando recursos.
- Introducción de una “memoria externa”: Se va a valorar el añadido de una manera de almacenar los programas externos que era capaz de ejecutar la máquina Apple I, para ello se valora el uso de la memoria interna o utilizar una memoria USB externa para este propósito. Simulando el lector de casetes de la época.

También se ha realizado una tabla de hitos, que no se pueden confundir con la metodología que se pretende seguir, donde los hitos están desordenados y corresponden a una primera aproximación sobre los objetivos del proyecto (Tabla 4).

1- Informarse sobre la placa FPGA y Apple I
1.1- Conocer las características de la placa FPGA
1.2- Conocer las necesidades del Apple I
1.3- Saber su compatibilidad y preparación
2- Instalación de Apple I en la placa FPGA
2.1- Realizar cambios necesarios en el código
2.2- Instalación en la placa FPGA
3- Configuración y prueba
3.1- Prueba de todos los dispositivos básicos

3.2- Rectificación de algún error
4- Configuración de memoria "externa"
4.1- Conocer la posibilidad de usarlo
4.2- Realizar la solución adecuada
5- Configurar botonera/display/switch
6- Resolución de algún error

Tabla 4: Hitos iniciales del proyecto (Febrero 2019)

Para una gestión del proyecto se ha realizado un Diagrama de Gantt inicial (Ilustración 20).

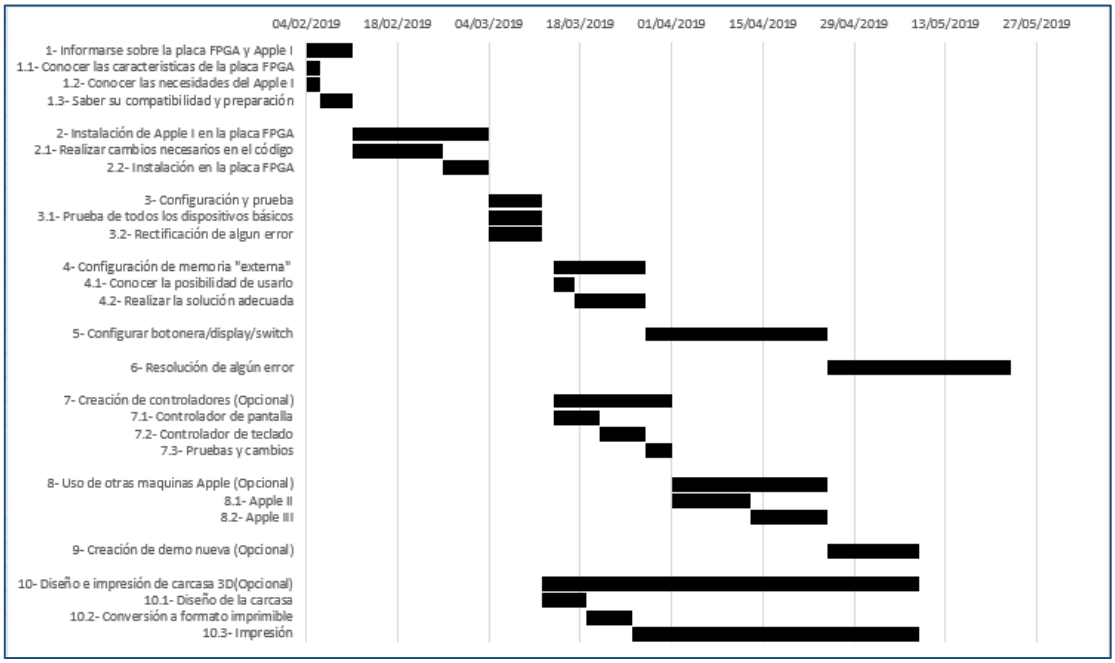


Ilustración 20:Diagrama de Gantt inicial (Enero 2019)

4.2. Software utilizado

El software que se pretendía utilizar en una primera intención era Vivado (Vivado Design Suite, 2019), herramienta para programar placa FPGA de Xilinx (Xilinx, 2019). Posteriormente, al indicar el problema de la incompatibilidad de determinados ficheros en hexadecimal, se ha tenido que hacer un cambio de entorno a uno menos especifico, este es, el ISE de Xilinx (ISE Design Suite, 2018).

Este entorno, ISE, es el acrónimo de *Integrated Synthesis Environment* (Entorno de Síntesis Integrada), es una herramienta de automatización del diseño electrónico (*EDA*) (About the EDA Industry, 2015), utilizadas para el diseño de circuitos integrados llegando hasta desarrollar placas de circuitos impresos, en este caso, no se va a llegar hasta ese nivel. El programa ofrece una manera sencilla de programar en lenguajes de descripción hardware (*HDL*) en las placas de Xilinx, además de disponer de herramientas para la síntesis de circuitos a nivel de registros y transistores (*RTL*) y la simulación del funcionamiento de circuitos y sus dispositivos de entrada y salida. Igualmente, permite la configuración de la síntesis e implementación buscando un objetivo, como reducir el coste del sistema, o reducir el tamaño del proyecto.

La utilización de este entorno menos exigente en relación con los ficheros externos, ha permitido la utilización de placas ya descatalogadas de Xilinx, pero donde ya existe una gran

comunidad en el caso de la aparición de dudas. Al ser una herramienta software propietaria, se ha tenido que disponer de una licencia para el uso. Esta ha sido cedida por el propio Xilinx, del tipo estudiante/trabajador del ámbito educativo.

Para cargar el proyecto en la placa seleccionada, se utiliza una herramienta de Digilent (Digilent, 2019), distribuidor de las placas de Xilinx, "Digilent Adept 2" (Digilent Adept 2, 2019), que permite preparar las placas FPGA que utilizan el ya indicado entorno de programación. En versiones posteriores, el trabajo realizado por este programa forma parte de las funciones predeterminadas de Vivado.

Para el proyecto se ha utilizado el lenguaje de descripción hardware Verilog (Verilog, 2012), aunque no exista una necesidad real de utilizar este lenguaje, se ha considerado el adecuado para este proyecto.

El Apple I incorporaba el lenguaje de programación BASIC (Bishop, 1982), creado en 1964, simple de utilizar, debido a que fue diseñado para el aprendizaje de lenguajes de programación en la época. En este caso, se ha utilizado una versión modificada por Wozniak, Integer BASIC (Weyhrich, 2003) que reduce el rango de instrucciones disponibles, para reducir el tamaño que ocupaba en la máquina original.

4.3. Apple I / Apple-One (FPGA)

El procedimiento que se va a seguir, es ir analizando cada uno de los componentes que se han programado y su exactitud con el componente utilizado en el Apple I. Algunos de los componentes donde se va a centrar el estudio son la CPU, Memorias y de forma imprecisa el conjunto de "Sistema", donde se valorará su funcionamiento ajeno a la ejecución del propio código. Para distinguir entre la maquina real Apple I y la réplica, Apple-One (Garfield, Moseley, & Otros, 2018), se van a utilizar estos nombres.

4.3.1. CPU

La CPU que fue utilizada en el Apple I (1976) fue un microprocesador que en su época fue una gran revolución, siendo una amenaza para Intel y Motorola, de MOS Technology (Matthews, 2003), actualmente desaparecida, con su MOS6502 (MOS6502, s.f.). Este microprocesador de 8 bits, creado en 1976 fue una gama entera que desarrollaron en la misma época, siendo los primeros de "software libre", donde cualquier programa era capaz de ejecutarse. Estos disponían de osciladores internos además de memoria interna. Una de sus mayores características, es que permitía frecuencias de entre 1MHz y 2Mhz que, aunque siendo la misma frecuencia que sus competidores, destacaban por su bajo precio, lo que le hizo un microprocesador muy versátil y útil.

Este procesador ofrecía un número menor de instrucciones en su conjunto de instrucciones comparado con la competencia, disponiendo de únicamente 56 instrucciones, mientras que el juego de instrucciones de los de Intel era de 78 instrucciones y de 72 los de Motorola. Esto fue uno de los motivos de porqué fue tan exitoso, cuando Intel o Motorola ofrecían más instrucciones y una frecuencia similar. El 6502 era el procesador más rápido de su gama y su menor coste, ofreciendo más posibilidades en muchos proyectos de estudiantes. Se trata de una segunda versión, puesto que, el primer procesador que diseñaron, el MOS6501 era muy parecido al MC68000, de Motorola, empresa de la que provenían los diseñadores, por lo que la misma interpuso una demanda por copia. Para evitar la misma, creando el 6502, que utiliza las mismas salidas que el MC6800 pero con diferentes instrucciones, evitando litigios con Motorola.

Este procesador fue utilizado en ordenadores como el Commodore PET (Velasco, Historia de la Tecnología: Commodore PET, 2013) y consolas como Nintendo Enterainment System (NES) de

Nintendo (Super Famicon/NES, s.f.). También es considerado el punto de salida de las arquitecturas SoC y ARM. Por tanto, se puede mencionar, que se trata de un procesador muy interesante en la historia de la computación.

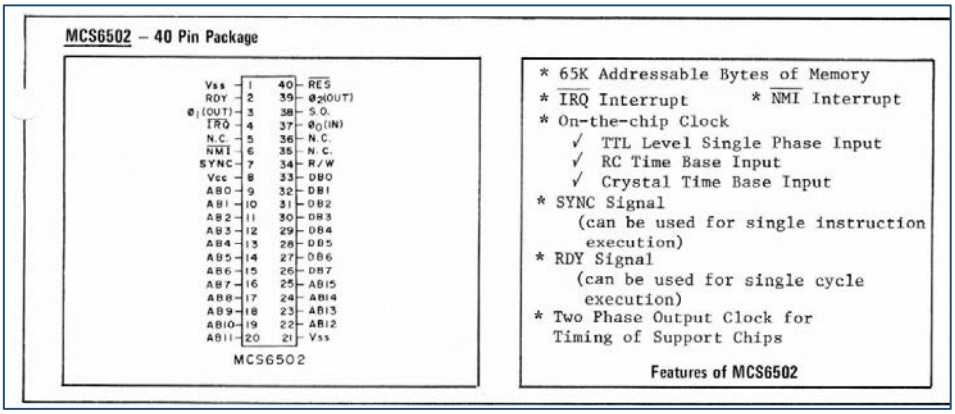


Ilustración 21: Diseño del MOS6502 y Pinout (MOS Technology, 1976)

Entre sus 40 pines (Ilustración 21), que eran lo más común en la época, similares a Intel o Motorola, se pueden ver dos salidas de frecuencia y una de entrada, permitiendo, obtener varias frecuencias para el sistema donde estaba instalado. También disponía de 16 bits de direccionamiento de memoria, que posibilitaba el uso de hasta 64 KB de memoria, aunque el Apple I, nunca superó los 48 KB. También ofrecía un reloj basado en lógica de transistores (TTL) y en cristales, que estaba integrado en el microprocesador.

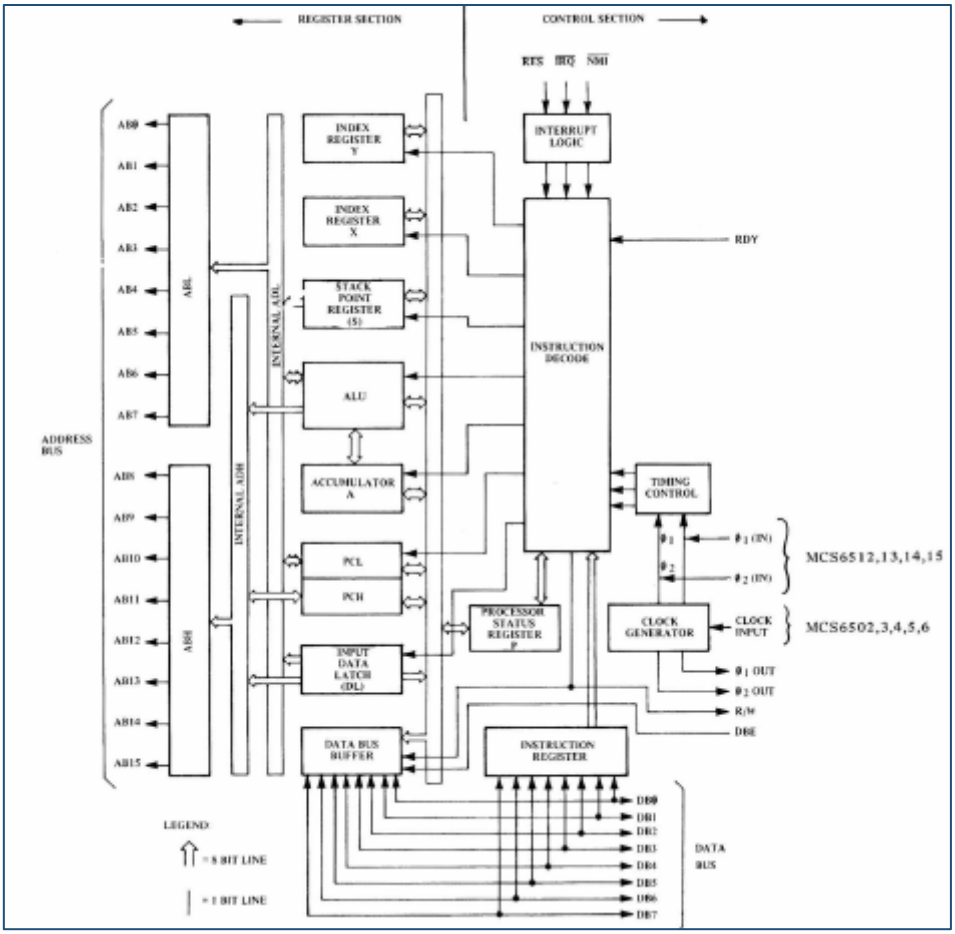


Ilustración 22: Arquitectura del MOS6502 (MOS Technology, 1976)

Su estructura interna, que compartía con otros de la gama 6500 (Ilustración 22), ofrecía varios registros como puede ser un contador de programa de direcciones bajas y altas, una ALU y varios registros de indexación, además de un registro de instrucciones.

ADC	Add Memory to Accumulator with Carry
AND	"AND" Memory with Accumulator
ASL	Shift left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result not Zero
BPL	Branch on Result Plus
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	"Exclusive-or" Memory with Accumulator
INC	Increment Memory by One
INX	Increment Index X by One
INY	Increment Index Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift One Bit Right (Memory or Accumulator)
NOP	No Operation
ORA	"OR Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Pointer to Index X
TXA	Transfer Index X to Accumulator

TXS	Transfer Index X to Stack Pointer
TYA	Transfer Index Y to Accumulator

Tabla 5: Set de instrucciones del MOS6502 (MOS Technology, 1976)

En su juego de instrucciones (Tabla 5) aparecen algunas instrucciones como operaciones de puertas lógicas AND, OR o XOR, funciones de transferencia o ramificaciones en función de una condición, disponiendo de un juego de instrucciones muy completo.

Una vez explicado de manera breve el microprocesador, se pasará a verificar su integridad o programación en Verilog. Este fue creado por Arlet Ottens y llamado "arlet-6502", siendo una aproximación que busca ser fiel al microprocesador MOS6502, pero no implementa todas las funcionalidades reales. Se realizará un análisis de la funcionalidad que implementa.

```
module cpu( clk, reset, AB, DI, DO, WE, IRQ, NMI, RDY, PC_MONITOR );

input clk;              // CPU clock
input reset;            // reset signal
output reg [15:0] AB;   // address bus
input [7:0] DI;         // data in, read bus
output [7:0] DO;        // data out, write bus
output WE;              // write enable
input IRQ;              // interrupt request
input NMI;              // non-maskable interrupt request
input RDY;              // Ready signal. Pauses CPU when RDY=0
output [15:0] PC_MONITOR;      // signal to spy / monitor the program counter for debugging
```

Ilustración 23: Código referente a la CPU (MOS6502) en ISE Design Suite

Con relación a los "pines" que dispone la réplica del 6502 (Ilustración 23) y el original (Ilustración 21), se puede ver, que no ofrece ninguna frecuencia de salida, ni tampoco una señal para el parámetro SYNC, que se utilizaba para una sincronización con el sistema. El resto de los pines, contando que el MOS6502 tenía 3 o 4 pines sin ningún uso, no son relevantes. En la parte interna, la mayoría de los registros y buses se han replicado de manera exacta al original.

En relación con las instrucciones que ofrecía el microprocesador 6502, el código de Arlet, ofrece la gran mayoría, ahorrando en algunos casos líneas de códigos, ya que agrupa las instrucciones para que no se repitan algunas operaciones por todo el código.

El procesador se ha hecho para que funcione a una frecuencia de 1MHz, como funcionaba el Apple I original, y en caso de aparecer algún error, es altamente probable que sea debido al programa que se ejecuta.

4.3.2. Memoria RAM

Otro de los componentes donde se busca más la similitud con la máquina real es la memoria RAM, que para su uso en el sistema se ha creado un fichero que simule los 8 KB que ofrecía. Este fichero inicializa la RAM a su valor por defecto, es decir, al comienzo de la ejecución este fichero se copia de manera íntegra en la memoria RAM que utiliza el sistema, procedimiento que se realiza cada vez que la máquina se resetea. Cómo es también lógico, almacena y saca valores de la memoria RAM, distinta a donde está cargado el intérprete BASIC.

El código no ha sufrido ninguna modificación, dado que es muy simple y no requiere de muchos cambios. Pero si se ha realizado, como modificación al proyecto de partida, un programa externo al Apple I para cargar la memoria RAM.

En un instante inicial del desarrollo se comentó la implementación de una memoria "externa" para añadir más capacidad de almacenamiento al sistema original, y con ello, más aplicaciones, debido a la complicación de la conexión de dispositivos USB a la placa, y los mecanismos para que actúe como

tal. Una solución sencilla podría ser el uso de una tarjeta SD para este propósito, pero se tendría el mismo problema anteriormente citado de la interfaz USB. Se ha optado por no implementar un segundo USB y guardar los programas utilizados dentro de la memoria interna de la FPGA

Se estudió la posibilidad de la carga de varios programas simultáneamente en la memoria RAM, viéndose que, en la máquina original no estaba contemplado, resultando inviable en dicho sistema. El problema se produce puesto que el software disponible para el Apple I está escrito en, lo que se podría llamar, su ensamblador, siendo específico para dicha máquina (códigos de operación, direcciones, registros…) y, de como máximo, 4 KB que, era el tamaño de la memoria RAM original del Apple I. Posteriormente, se duplicó la cantidad de memoria RAM disponible, pero al ser tan tedioso escribir software en el mencionado lenguaje, se optó por añadir in intérprete de un BASIC reducido que permitía la programación del mismo, ya de una forma más sencilla.

Para la integración de los programas originales del Apple One en el sistema, se implementó un programa que copia los ficheros de las diferentes aplicaciones a incluir en el fichero de la memoria RAM de 8 KB. De esta manera, se puede disponer del software deseado ya cargado en la RAM, no siendo necesario un lector de casetes. Esta es una solución del propio creador del código Alan Garfield en Twitter (Ilustración 24), ofreciendo además de la carga directa a la RAM el traspaso vía Terminal en ASCII. Se ha optado por realizar un “cargador” de RAM externo.

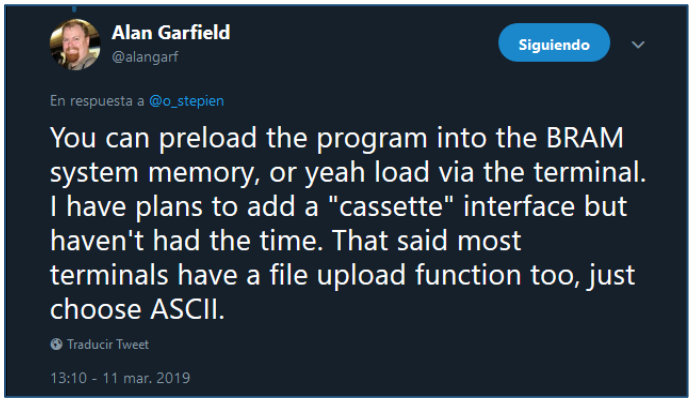


Ilustración 24: Captura tweet de respuesta del creador del código ante la forma de usar la memoria RAM

Dado que el uso inicial del código de la memoria RAM únicamente hace que se copie un fichero .HEX (en hexadecimal) a un registro que actúa como la RAM y permite la lectura y escritura a la RAM de ese fichero, se podría utilizar ese modulo para cargar el programa. Pero esta carga podría ocasionar algún problema en la ejecución, pues el sistema, aunque cada ciclo de reloj puede escribir en la memoria RAM, no es posible evitar que el usuario escriba mientras se cargue el programa y bloquee la ejecución del sistema.

Por ese motivo se ha considerado realizar un programa externo, que pueda funcionar a la máxima frecuencia de la placa, 100 MHz, para escribir en la memoria RAM el programa deseado y después se pueda ejecutar en el emulador. Cómo esto resultaba complejo dado al error (4.3.2.1.b), se ha optado por simplificar el modo de funcionamiento del sistema, que se comentará a continuación, después de detallar el método que se va a utilizar en los ficheros de programas.

Como se ha indicado, la funcionalidad pedida para la creación de este programa es la carga del programa en la RAM. Se ha especificado inicialmente un formato concreto para realizar la carga del software añadido. Como se puede ver en la imagen (Ilustración 25), la primera línea del documento indica la posición de inicio del documento, habiéndose probado con distintas posiciones y concluyendo que estos programas solo pueden funcionar en la posición indicada, y únicamente de

uno en uno. Por ello, cada vez que se realice una carga de un programa, el resto del documento (con el que se implementa la RAM) debe quedar vacío, dejándolo a 0x00.

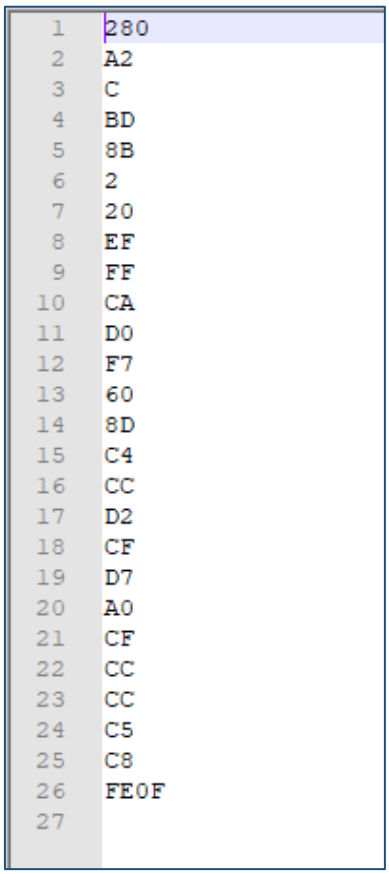


Ilustración 25: Estructura de un programa original de Apple I (hello-world)

Este mecanismo previamente explicado no era compatible con el funcionamiento deseado, por lo que, se optó por combinar todos los ficheros de programas e incorporarlos en uno único. De esta forma, se permite al usuario añadir más programas al sistema de una manera sencilla, aunque no se pueda realizar la carga "en caliente" de los programas. Este método fue el que se pensó más conveniente teniendo en cuenta que, como era lo deseado, no se pretendía modificar el funcionamiento del emulador tomado como referencia.

Cómo el Apple I permite dos formatos de programas, uno escritos para el "ensamblador" del Apple I, el cual se detalla en un apartado más adelante, y otro formato es por medio del interprete BASIC (apartado más adelante), ambos cargados originalmente en la ROM del Apple I. En el caso ideal, se implementaría una solución que permitiera cargar ambos tipos de formatos, pero siendo fiel al funcionamiento real del Apple I, por casetes se ha optado por no hacerlo, manteniendo la funcionalidad original.

Los casetes que utilizaba en la época la máquina Apple I usaban cintas de audio, que, por medio de distintos tonos grabados, el sistema era capaz de cargar los valores en hexadecimal en la memoria RAM. Por tanto, los casetes sólo contenían valores hexadecimales. El usuario era el encargado de cargarlos en la posición indicada y después ejecutarlo. Del mismo modo se cargaban los programas en BASIC, aunque estos conllevan un poco más de dificultad para su grabado. Por ese motivo, y dado algunos inconvenientes detallados en el apartado correspondiente, se ha implementado un proceso carga los programas en la memoria RAM del sistema.

Este procedimiento se activa al pulsar el botón central de la botonera, entrando en una fase donde se carga el programa. El proyecto contiene un fichero *"cassette.v"* , que es el encargado de seleccionar el programa deseado. Este programa devuelve una variable "pick", que es el puntero al programa deseado. En la RAM (*"ram.v"*), se buscan los valores del tamaño del programa, su posición inicial en el fichero de los programas y la posición inicial donde se quiere cargar en la RAM. Posteriormente entra en la fase donde copia y/o reescribe la memoria RAM usando tres fases, las cuales se pueden ver en la Ilustración 26.

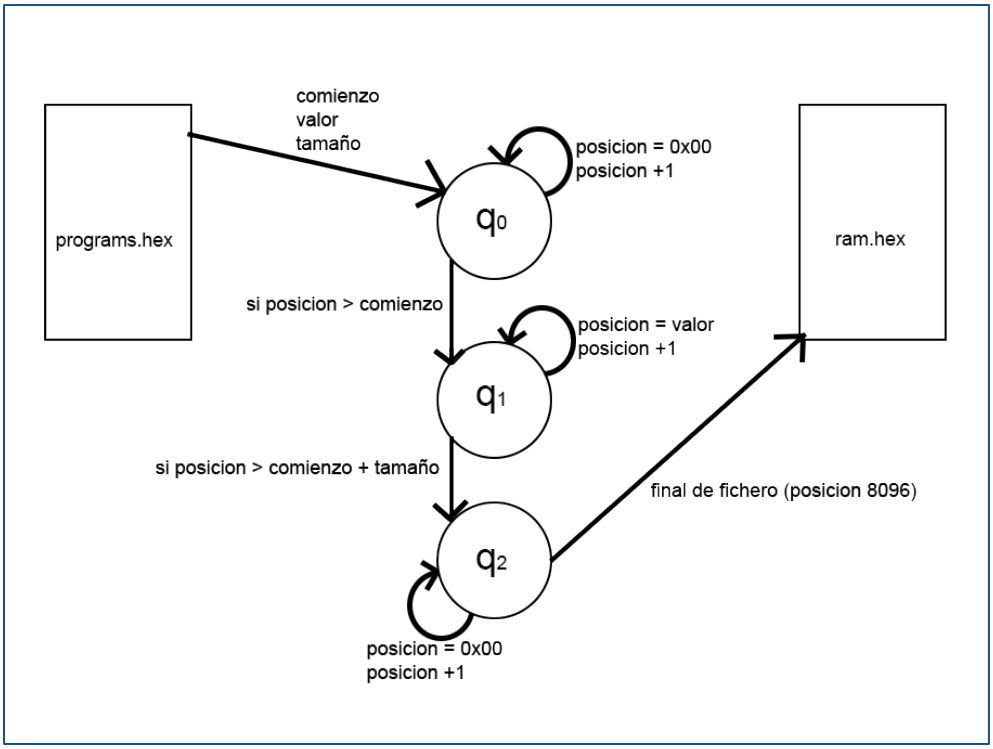


Ilustración 26: Diagrama de la carga en memoria RAM

La primera fase vacía el programa ejecutado anteriormente, y que va a ser sustituido por el nuevo, del fichero de RAM, en concreto desde la posición ocupada por la primera instrucción del programa anteriormente en ejecución (y que, si se trata del ejemplo anterior, Ilustración 25, es la dirección: 0x280) y hasta la dirección de comienzo donde el nuevo programa debe almacenarse (indicado por la variable "pick" antes comentada), escribiendo la palabra 0x00 en cada dirección. Una vez llegado a la posición deseada, indicada en una variable específica, "pick", se entra en la fase dos.

En esta fase el programa leerá una línea del fichero de entrada, "programs.hex", (del nuevo programa que se quiere ejecutar) y lo escribirá en el fichero de salida, "ram.hex" (que implementa la memoria RAM). En el momento que el proceso de copia escriba la última instrucción del nuevo programa en la RAM, se pasará a la fase tercera fase.

En ésta, el programa diseñado recorrerá todas las posiciones restantes del fichero que implementa la memoria RAM, "ram.v", estableciéndolas a 0x00, esto es, vaciando el resto del fichero. Para distinguir las distintas fases del programa, se ha utilizado dos activadores, uno indicando que el documento se puede escribir "rdy" y otro indicando si está en el inicio del fichero o en el final. Para ofrecer un indicador al usuario, se utilizan dos LEDs de la placa para indicar que el programa cargador se ha iniciado y se ha cargado. Al realizarlo en frecuencias de 25MHz, esta copia es casi instantánea y apenas se distingue el momento de encender cada LED, estos LEDS se pueden ver en el recuadro "a" de la Ilustración 16.

Al finalizar la carga en la tercera fase, se cambian el valor de los LEDS usados y algunas variables “cerrojo” para permitir volver a escribir la memoria RAM por si se hubiera cambiado de idea y se quisiera ejecutar otro programa.

Una vez explicado el procedimiento implementado se describen someramente los programas que, en este proyecto, se dejarán a modo de “demo” en el fichero “programs.hex” (fichero usado para cargar los programas a la memoria emulada “ram.hex”, por medio de los programas encargados de ello “cassette.v” y “ram.v”), siendo todos ellos originales del Apple I.

Para seleccionar el programa a ejecutar, en el display de la placa se muestra el número de orden y un pequeño acrónimo, que identifica a cada programa, facilitando así la selección del programa a ejecutar.

En la lista mostrada a continuación, aparece el mencionado acrónimo al principio, posteriormente el nombre completo del programa y finaliza, cada ítem, con una breve descripción de la aplicación:

0. nUL (Null): Programa vacío con el objetivo de vaciar toda la memoria RAM, para una instalación “limpia” en el siguiente reinicio de la máquina.
1. A30 (Apple30th) : Programa de 2006 conmemorando los éxitos de Apple en sus primeros 30 años, siendo un detalle que se haya realizado un programa para el Apple I y la gama de Apple con lector de casetes. Muestra imágenes, en caracteres ASCII, de Wozniak, Jobs, el Apple I, Apple II, Macintosh, iMac, iPod y MacBook, además de su ya conocido logo.
2. hEL (Hello-World): Programa sencillo que muestra por pantalla el clásico “HELLO WORLD” . Lo llamativo, por lo que hay que tenerlo en cuenta, es que se ha escrito para el monitor del Apple I, y no en BASIC.
3. LUn (Lunar Lander): Programa similar a un juego, diseñado para el Apple II, se controla en modo texto, un módulo lunar para su aterrizaje. El programa original, Apple II, utilizaba la pantalla en modo gráfico, pero siempre es bienvenida una implementación para el Apple I.
4. MEM (Memory Test): Programa, que consta de dos partes, y se escribe en dos posiciones de la memoria. El programa verifica la integridad de la memoria inicial hasta la posición 0xFF de la misma.
5. uCh (MicroChess): Programa para jugar al ajedrez en modo texto, no gráfico, que funciona escribiendo, en el clásico formato empleado en el ajedrez, la posición de la ficha que se quiere mover y la casilla (escaque) de destino.
6. PAS (Pasart): Programa que realiza unas tablas artísticas en base al número de sus filas, columnas, modulo y el estilo seleccionado.

Además, se han añadido otros programas como:

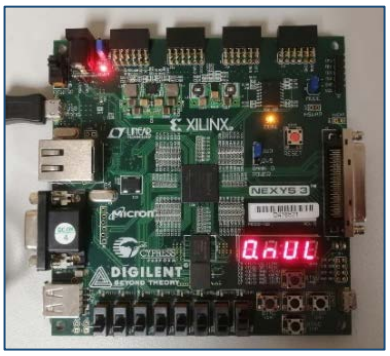
7. Star Trek 1
8. Star Trek 2
9. Little Tower
10. Blackjack 1
11. Blackjack 2
12. Checkers 1
13. Checkers 2
14. Hammurabi 1
15. Hammurabi 2
16. Matrix 1

17. Matrix 2
18. Slots 1
19. Slots 2
20. Monitor
21. Wumpus 1
22. Wumpus 2

Cuyas características, de forma más detallada, se muestra en

Lista con los programas disponibles apartado Anexo II.0, obtenidos de la página LinuxCoffee (LinuxCoffee, 2019).

La información, sobre los programas disponibles, se muestra, como se ha indicado, en el display de forma cíclica, esto es, cada vez que se acciona el botón BTNR o BTNL de la placa (visibles en el recuadro "f" de la Ilustración 16), se muestra el siguiente de la lista, y al llegar el final de la misma, se vuelve a empezar. Una muestra de algunos de estos mensajes mostrados por el display (Ilustración 27) a continuación:

A) 

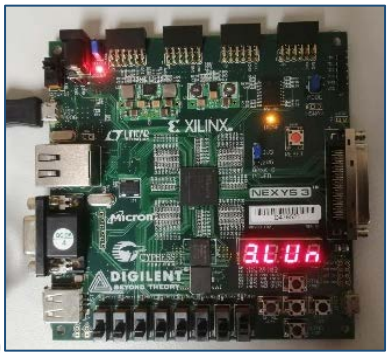
B) 

Ilustración 27: Imágenes de los distintos estados del casete realizado: a) Programa 0.NULL; b) Programa 1.LUNAR

Para desarrollar esta funcionalidad (mostrar los programas almacenados en la memoria auxiliar) se han creado tres programas: uno encargado de mostrar por el display el mensaje necesario para hacer saber al usuario el programa donde apunto actualmente "display.v". En el display, se ha establecido esos nombres, sin posibilidad de modificarlos si no se vuelve a cargar el programa en la placa FPGA. Otro programa "ram.v" es encargado de la copia del programa a la memoria RAM del Apple I. Y el tercer programa, "casette.v" es un programa que informa a los programas previos el programa a mostrar y cargar.

4.3.2.1. Problemas y resoluciones.

A diferencia de la CPU, donde se ha preferido no modificar nada del código, al ofrecer todo lo necesario para interactuar de manera correcta, en la memoria RAM, en concreto el programa "cassette.v" han surgido los siguientes inconvenientes.

a) Inicialmente, como sustituto del dispositivo de cinta (casete), se planteó la implementación de conectividad con dispositivos de almacenamiento USB, obviamente no existentes en la máquina original, para el intercambio de datos y ofrecer una memoria externa en la placa. Esta memoria seguramente permitiría una carga "en caliente" , es decir, cuando ya se está ejecutando el emulador (sin necesidad de reinicios) de los programas almacenados en un dispositivo mencionado. Por motivos de incompatibilidad en el sistema, esto no ha sido posible, ofreciendo la solución alternativa anteriormente planteada.

b) Otro problema resuelto que tiene relación con la memoria RAM, es la utilización de funciones de Verilog para interactuar con ficheros, entre las que destacan *"$fopen"*, *"$fclose"*, *"$fread"*, *"$fwrite"* y *"$fdisplay"*. Estas funciones están permitidas únicamente en simulación, no son sintetizables, motivo por el cual, son incompatibles con la implementación sobre la placa FPGA. Para este inconveniente, se plantearon varias soluciones alternativas.

Una de ellas, que ofrece una solución *sintetizable*, es utilizar la función "*$readmem",* que realiza la copia de un fichero externo en una variable dentro del sistema. De esta manera se dispone una solución compatible con la implementación sobre la placa FPGA. El uso que se le quería dar es la copia del fichero de programa directamente en la memoria RAM *"ram_data"*. Pero esta función sólo puede usarse al inicio de la ejecución teniéndose que volver a reiniciar el sistema cada vez que se quisiera un cambio de programa para ejecución.

Otra solución con la cual evitar el problema existente, consiste en utilizar una copia en tres fases, usando un fichero global con todos los programas en distintas posiciones del fichero. Esta alternativa es una modificación de la probada en simulación en un proyecto ajeno a este "casete.xise", en el cual se utilizaban las mismas funciones de *"$fopen",etc…* usando un fichero global, en lugar de uno externo, para cargar el contenido del programa en la *variable "ram_data"* que actúa de RAM en la emulación. Esta solución fue descartada, al disponer ya de un mecanismo funcional, realizado previamente, cómo códigos externos.

Estos serían los problemas ocasionados relativos a la realización de la RAM y a la carga de los programas. A pesar de la existencia de otros problemas menores con software a ejecutar, la utilización de un "macro fichero" para almacenar los programas, los ha solucionado, ofreciendo una solución viable.

c) También se ha encontrado un error con el programa "Codebreaker", el cual, a pesar de copiar perfectamente la memoria era incapaz de ejecutar las aplicaciones cargadas, bloqueando el Apple I en su ejecución. La solución a este problema ha sido, omitir este programa, puesto que, con la solución implementada, ya no era necesaria su ejecución.

4.3.3. Memoria ROM

Otro componente importante y necesario del funcionamiento del Apple I. La memoria ROM que ofrecía en su época el Apple I era un "ensamblador" programado por Wozniak, llamado originalmente "WozMon" de "Wozniak Monitor", para hacer funcionar el Apple I. A pesar de ocupar escasos 256 Bytes, es un programa lo suficientemente potente como para controlar todas las opciones del microprocesador 6502, como por ejemplo, la instrucción "NOP" o las instrucciones de transferencia que ofrecía, visibles en la Tabla 5. Es importante que estas instrucciones las entiende en el lenguaje "ensamblador" con las que funciona WozMon y no vienen escritas en codificación ASCII, para hacer funcionar estas instrucciones, se ejecutan las rutinas que tiene predefinido este "ensamblador".

Este código está almacenado en la ROM en hexadecimal, mediante instrucciones de 8 bits de longitud, esto es, dos dígitos hexadecimales, al igual que los programas que se leen desde el casete. Dado que es el encargado de iniciar el Apple I para su arranque y todas sus instrucciones, es un programa indispensable para la máquina. Se almacena en las posiciones altas FF00-FFFF de la memoria del Apple I.

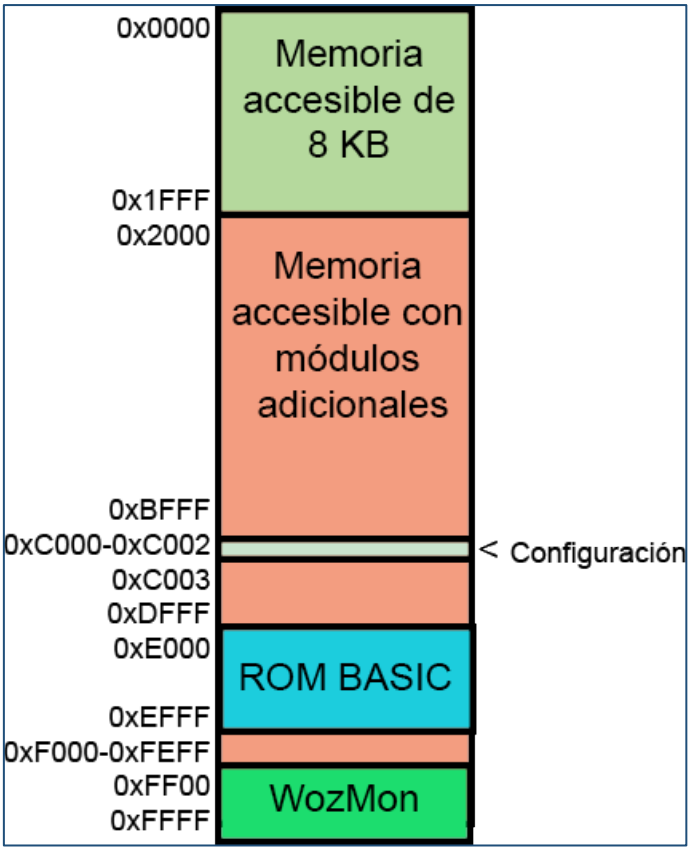


Ilustración 28: Disposición de la memoria del proyecto

El principal funcionamiento al cual se le aplica es la escritura en las direcciones bajas de la memoria usando comandos como ***"0001:1"*** escribiendo en la posición ***"0x0001"***el valor ***"0x01"***. También ofrece la opción de ejecutar con ***"R"*** en la posición indicada previamente y también ofrece la opción de consultar los valores almacenados en la dirección que se solicite.

Con estas tres funcionalidades básicas es capaz de escribir en varias direcciones seguidas, escribir en intervalos y de ejecutar cualquier programa cargado en la Memoria RAM. En la página de "SB Projects" (The Woz Monitor, 2018), se muestra un funcionamiento del ensamblador de Wozniak, el cual ofrecía una "interfaz" propia al Apple I.

Entre sus otras funcionalidades existen algunas instrucciones que un programa puede utilizar para interactuar con el microprocesador. Entre ellas destacan, indicándose incluso la dirección donde se almacenan, las siguientes:

- Dirección ***"FF1F":*** GETLINE, permite devolver el control al Monitor una vez este finalice su ejecución.
- Dirección ***"FFEF":*** ECHO, imprime el carácter del "Acumulador" por pantalla.
- Dirección ***"FFDC":*** PRBYTE, imprime el byte del "Acumulador" en hexadecimal.
- Dirección ***"FFE5":*** PRHEX, imprime el último byte del "Acumulador" en hexadecimal.

También dispone de una lista de direcciones como una funcionalidad dedicada, algunas de las cuales se muestran en la siguiente tabla (Tabla 6).

$24 -> $2B	**Pila Cero.** Almacenamiento de propósito general, cuyo contenido el usuario puede modificar sin ningún problema
$100 -> $1FF	**Almacenamiento de pila.** A pesar de que la ROM de WozMon sólo ocupa 3 bytes, su posición inicial varía entre 256 bytes de esta pila. Cualquier código escrito en estas posiciones puede resultar sobrescrito.
$200 -> $27F	**Buffer de entrada.** En estas posiciones se almacenan los valores usados por otros programas. Al igual que las anteriores posiciones, WozMon las sobrescribe, por tanto, no almacena nuevos valores en estas posiciones.
Direcciones externas a la RAM pero importantes	
$D010	Registro del teclado. Valor que se almacena cuando el teclado está activo.
$D011	Registro utilizado para saber si el teclado recibe datos o los recibe
$D012	Registro con los caracteres mostrados por pantalla
$D013	Registro especifico de WozMon

Tabla 6: Direcciones "reservadas" de WozMon (The Woz Monitor, 2018)

Como ya se ha indicado previamente, la RAM original tenía un tamaño de 8 KB, por tanto, desde la posición 0x0000 hasta la posición 0x1FFF. Pero en caso de ampliar hasta el máximo posible, 48 KB, ésta llegara hasta la posición 0xBFFF, "reservadas" para el uso de la RAM. En adelante son registros de configuración o memoria ROM, siendo accesibles, pero no se recomienda modificarlos.

De esta manera es posible acceder hasta la última dirección disponible 0xFFFF, reservada para WozMon, como ya se ha especificado previamente. En relación con la programación del mismo en Verilog, está disponible en el fichero "rom_wozmon.v" y es muy similar al código ya realizado para implementar la RAM interna. Se carga el fichero "rom_wozmon.hex" en una variable interna del programa. Durante el funcionamiento, permite únicamente obtener los valores almacenados dentro. Dado que también existe otra memoria ROM, se va a explicar esta segunda ROM.

4.3.4. Memoria ROM BASIC

BASIC (BASIC, 2019) es un lenguaje de programación creado en 1964, con el objetivo de acercar a los usuarios y aficionados al mundo de la programación. Disponía de instrucciones muy simples y fáciles de recordar, lo cual limitaba mucho en lenguaje, habiendo una gran diversidad en programas BASIC para el Apple I.

Uno de los mayores fallos es creer que el BASIC utilizado en el Apple I es el BASIC (Microsoft BASIC, 2019) de Microsoft (Microsoft, 2019) puesto que aunque pueda compartir instrucciones con él, Wozniak creó su propia versión, Integer BASIC (Weyhrich, 2003). Esta versión, era una mejora del BASIC original de Darthmouth College (Bishop, 1982), buscando una forma más simple de BASIC, en este caso por la palabra "Integer" de "Integer BASIC", se denota que sólo trabajaba con enteros.

Integer BASIC, del cual se verán con mayor detalle sus palabras reservadas y forma de programación en el 0, es un lenguaje interpretado (no compilado), conocido mayormente por Apple II, pero que fue ya utilizado en el Apple I. Como ya se ha comentado, tiene instrucciones muy limitadas, como LET, PRINT, GOTO, GOSUB, etc… y sólo permite interaccionar con enteros o cadenas de caracteres, pero consiguió estar muy difundido entre los aficionados, que han desarrollado gran cantidad de aplicaciones, algunas muy complejas, desde la generación de secuencias de caracteres aleatorias, del estilo de la película Matrix (Ilustración 29), hasta un juego basado en Star Trek (Ilustración 30).

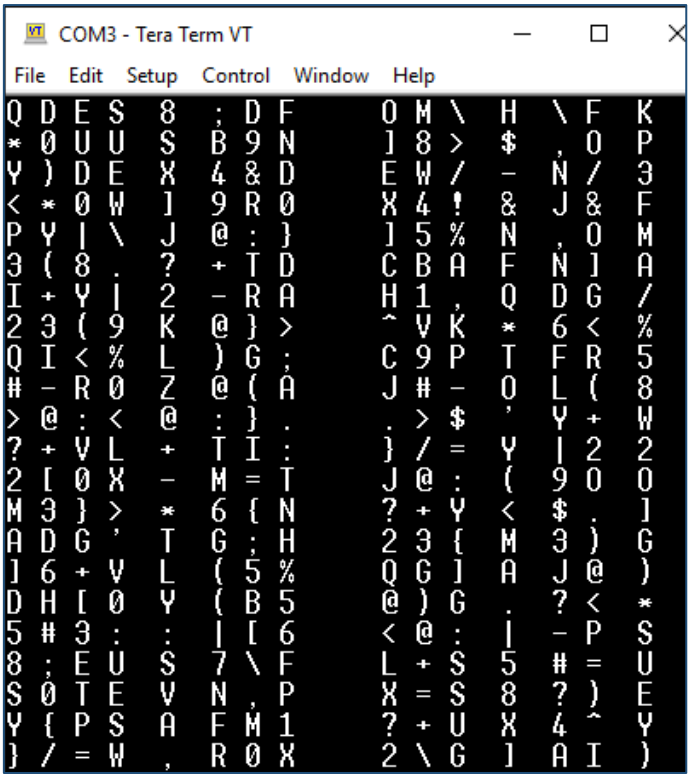


Ilustración 29: Ejecución de código estilo 'Matrix'

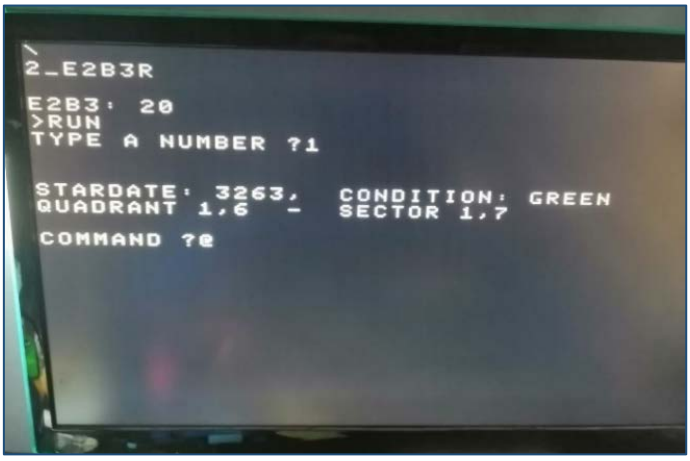


Ilustración 30: Programa cargado de Star Trek

Antes de ser cargados a un casete, los códigos se debían escribir en BASIC en la propia máquina para comprobar su integridad y su correcto funcionamiento, siendo necesaria su escritura a mano. La carga en estos casetes reducía gran parte del tiempo que supondría escribir el código nuevamente a mano por otro usuario.

La implementación, en este proyecto, de la funcionalidad de ejecución de aplicaciones en Integer Basic, disponible en “rom_basic.v”, es muy similar a la utilizada para la RAM o ROM. Realizando la carga de los valores en un momento inicial, el arranque de la máquina, permitiendo únicamente el volcado de datos durante el reinicio.

Otro fichero importante para disponer de esta funcionalidad, es el que contiene el código de Integer BASIC en hexadecimal, codificación que entiende la máquina, disponible en “/roms/basic_ise.hex”, ocupando 4096 bytes, por tanto 4 KB. Como se puede ver, ocuparía casi la

memoria completa del Apple I, por lo cual, se cargaba en una memoria externa a la RAM, con un tamaño de 4 KB, empezando en la dirección 0xE000 y llegando hasta 0xEFFF.

También es conveniente mencionar que existen archivos con los ficheros de los programas en Basic traducidas las instrucciones al Hexadecimal. Varios de estos programas se pueden ver en la dirección de LinuxCoffee (LinuxCoffee, 2019).

Retomando el tema de los casetes, existen algunos con juegos en BASIC, pero, únicamente realizan la carga de datos hexadecimales en las direcciones de memoria donde se guardan, para su ejecución, los programas en BASIC, cómo ejemplo, el fichero del programa del juego de Star Trek que, aun siendo un programa escrito en BASIC, en el fichero que se emplea para su carga en memoria sólo existen caracteres en hexadecimal, habiéndose explicado, en el apartado relacionado con la Memoria RAM, la manera de cargar programas en este formato.

En los apartados siguientes se va a explicar los elementos referentes a la Entrada y Salida del Apple I, la UART y el Teclado.

4.3.5. UART

La UART (UART Protocol Validation Service, s.f.) es el mecanismo que sustituye la pantalla y el teclado, permitiendo el uso del emulador desarrollado mediante un terminal, empleándose en este proyecto la aplicación "TeraTerm" (Teraterm, 2019), emulador de terminal gratuito para Windows (Windows, 2019) configurable para su correcto funcionamiento.

En este caso, se pretendía mejorar el funcionamiento de la UART, pero al ver, que el código del cual se iba a basar y el utilizado originalmente era el mismo (FPGA4Fun, 2019), se ha decidido no modificar el ya disponible.

El funcionamiento es sencillo, pues la placa FPGA dispone de un puerto UART con este propósito, utilizando un registro Rx y otro Tx, para recibir y transmitir, respectivamente. Para la transmisión por medio de UART, se debe configurar la frecuencia de Baudios, utilizada para la sincronización Terminal-Placa, decidiéndose usar 115200 Baudios, por ser la frecuencia por defecto del código y del terminal utilizado.

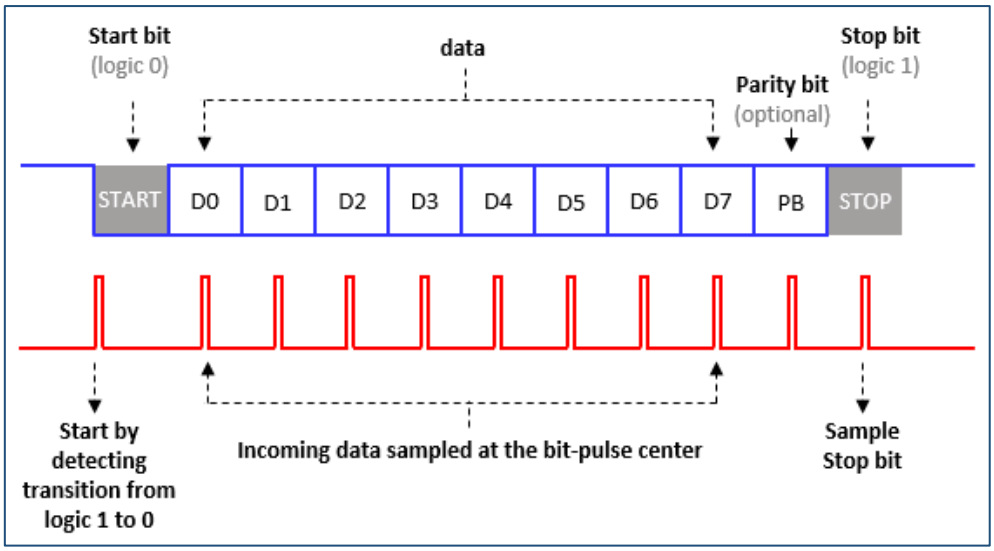


Ilustración 31: Protocolo UART (UART Protocol Validation Service, s.f.)

Como se ve en la ilustración anterior (Ilustración 31), el protocolo UART realiza el siguiente procedimiento, comienza detectando un cambio en el bit recibido (de '1' a '0'), a continuación se

envían los 8 bits de datos, opcionalmente con un bit de paridad añadido. Para cerrar el paquete recibe en su bit el valor '1', permitiendo enviar o recibir otro dato de 8 bits.

Al ser un protocolo bastante conocido, y muy utilizado, se cree innecesario comentar más el código, disponible en "uart.v". Este código, inicializa ambos, receptor y transmisor, que se activan con el Switch más a la derecha de la placa a '0' y el LED correspondiente apagado, estos están disponibles en el recuadro "c" de la placa (Ilustración 16). Dispone también de un registro "address" de 2 bits, para indicar que procedimiento debe ejecutar en ese momento, estos son "Recepción de datos (RX)", "Recepción de datos con Retorno de carro (RXCR)", "Transferencia de datos (TX)" y "Transferencia de datos con Retorno de carro (TXCR)". Permiten una comunicación usando este protocolo tal y como se ve en la Ilustración 31. Como ya se ha comentado, aunque se escriba por teclado, el terminal replica en la pantalla VGA conectada lo escrito (para verificar su corrección).

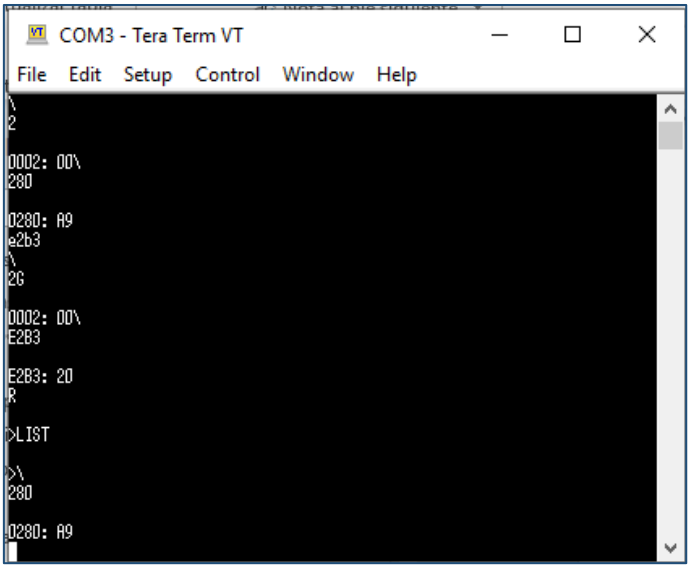


Ilustración 32: Terminal mostrando el Apple I

Como se puede ver en la imagen anterior (Ilustración 32), no limpia la pantalla, siendo un registro en caso de fallar parte del código. Dado que en el terminal se ve más pequeño, es una opción más sencilla para visualizar de forma más correcta por su resolución el contenido que se muestra por pantalla (Ilustración 33).

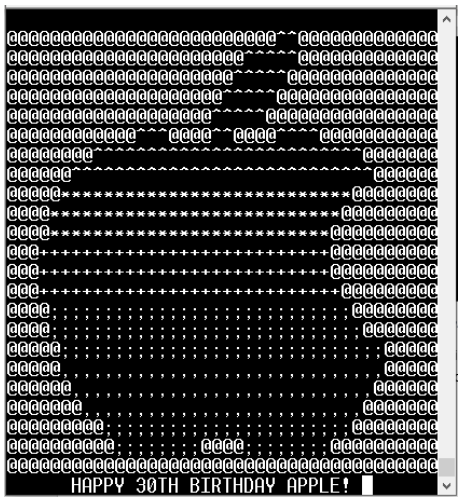


Ilustración 33: Visualización por TeraTerm del Apple I

A continuación, se verá cómo se ha resuelto la interacción del Apple I para utilizar un teclado vía USB.

4.3.6. Teclado

La placa seleccionada dispone de un puerto USB (Universal Serial Bus) (Definicion ABC, s.f.) que permite la conexión de un teclado con dicha interfaz. Inicialmente se pensó realizar una modificación del código ya realizado por Alan Garfield (Garfield, Moseley, & Otros, 2018), pero en dicho proyecto se utilizó un teclado mediante interfaz PS/2 (IBM Personal System/2) (CAVSI, s.f.), del que actualmente pocas FPGAs disponen, y con el afán de "actualizar" el Apple I, se decidió implementar la funcionalidad para emplear un teclado USB.

El puerto PS/2 utiliza 6 pines (Ilustración 34), de los cuales, 2 son reservados para los ratones que funcionan con este puerto, de los 4 restantes, uno funciona cómo Vcc, ya sea 5v o 3.3v y el otro como Gnd. Restando únicamente dos pines, Clk y Data.

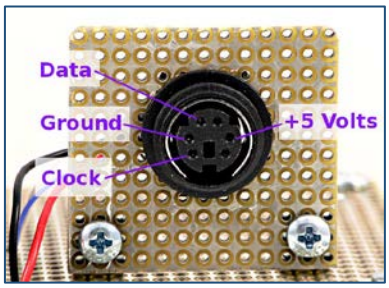


Ilustración 34: Puerto PS/2 (PS2 Keyboard Library, s.f.)

Esta distribución es muy similar a los utilizados en USB (Ilustración 35), donde se tiene, un pin de corriente, Vcc, otro de toma a tierra, Gnd o *Ground*, y dos que son Data+ y Data-, que permite una conexión entre ambos interfaces muy simple.

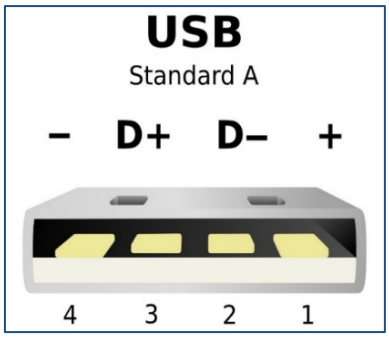


Ilustración 35: Puerto USB (alain, 2014)

Después de revisar el manual de referencia de la placa Nexys-3 (Digilent, 2016), se encontró que el funcionamiento del USB de la placa, funciona igual que el puerto PS/2, resultando que el código ya realizado es compatible con el teclado USB a pesar de haber sido diseñado para teclados PS/2.

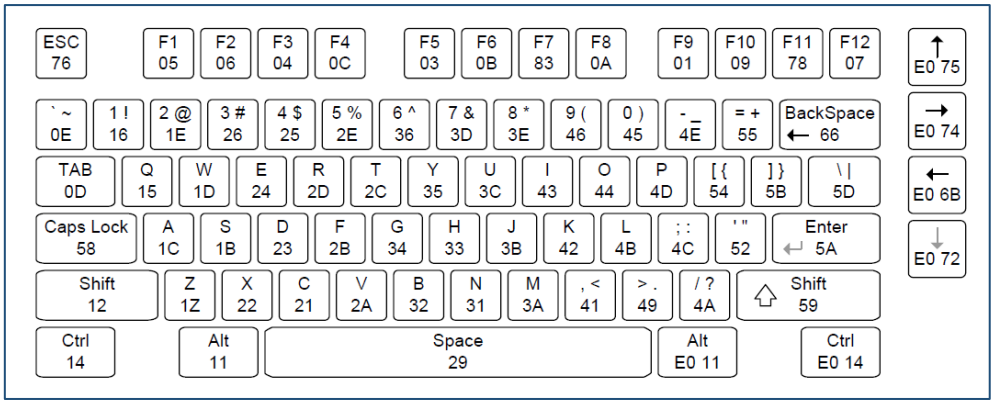


Ilustración 36: Teclado USB Nexys-3 (Digilent, 2016)

El principal inconveniente, es la necesaria conversión del código recibido por teclado a formato ASCII original (de 7 bits, pues usaba menos caracteres), pues el Apple I utilizaba teclados con esa codificación, y no la ASCII Extendida, de 8 bits, de los "modernos" de USB o PS/2. Por tanto, se requirió una conversión del ASCII Extendido, mostrándose el código hexadecimal de las teclas del teclado en la ilustración anterior (Ilustración 36) a ASCII, que pueda entender el emulador, teniendo en cuenta, además, que la tabla ASCII tiene 128 caracteres y el Apple I no utiliza minúsculas. Para realizar este propósito cada código del teclado se traduce en un carácter ASCII, independientemente si se ha pulsado o no la tecla "Mayúscula" o "*Shift*", siempre en mayúsculas. También se debe tener en cuenta que para no "perder la esencia", se ha dejado que el teclado que entiende es de distribución americana, y esta dista en muchos aspectos de la distribución española, por tanto, se "complica" la escritura por teclado.

Para conseguir un funcionamiento adecuado del teclado, el pin Data+ se utiliza para el reloj del sistema y el pin Data- para el código del teclado, teniendo un gran parecido con los teclados PS/2. La frecuencia que utiliza es de 25 MHz, la misma en la cual funciona el sistema, siendo el procesador el único que funciona a 1 MHz por su construcción y no perder matices de la recreación del Apple I. En el código del proyecto, "ps2keyboard.v", se puede ver que este omite las pulsaciones "especiales" y pulsaciones de teclas adicionales. Estas pulsaciones se "atrapan", porque indican la forma de pulsación "E0" si se mantiene pulsada la tecla, "F0" si se suelta y "E0F0" si se mantiene y suelta. Estas combinaciones, no son relevantes en el código, por tanto, no deben tenerse en cuenta y se omiten. Aceptando solo las pulsaciones relevantes o teclas del teclado usado.

4.3.7. Pantalla

Como se sabe, hasta 1987 no se comercializó la tecnología VGA (Sanchez, 2016) que, a pesar de los años, se sigue utilizando aún. Su predecesor conocido fue SCART o Euroconector (Pillou, Conector SCART, 2008), usado desde 1978 y un estándar en los televisores, que reemplazó al Video Compuesto (Pillou, Video Compuesto, 2008).

En el Apple I, la salida original empleada para conectar el monitor era una salida de cuatro pines (Ilustración 18 b)) que requería de un adaptador a Video Compuesto (Ilustración 19). Diseñado de esta forma para ahorrar gastos.

Teniendo en cuenta que son tecnologías obsoletas, para este proyecto se decidió realizar la salida de vídeo utilizando la tecnología VGA que ofrece la placa elegida, además de ser más sencillo su uso en la transmisión de la imagen, optándose, por ello, a no modificar de gran manera el código tomado como referencia, que no es más que una ligera modificación del código que ofrece Xilinx para probar el video.

A pesar de disponer exclusivamente de una resolución de 640x480 pixeles, se ha valorado disponer de otras resoluciones, pero la salida por VGA no permite un cambio así sin modificar la frecuencia interna de la máquina. Al analizar con más detenimiento, el código de partida utiliza un fichero que simula la memoria RAM de la tarjeta VGA, mediante el cual, se indica al sistema el estado actual de la pantalla. Esto dificulta el cambio de resolución dado que supondría modificar el tamaño de esta RAM de video a un tamaño adecuado a la resolución deseada.

Para poder utilizar el VGA de esta resolución se debe tener en cuenta la frecuencia de refresco de cada pantalla. Las pantallas VGA suelen funcionar a una tasa de refresco de 60Hz y una tasa de bits de 25MHz. Por ello, el sistema desarrollado tiene que utilizar esa tasa de bits para transmitir la información que se desea mostrar. También se debe saber los valores constantes para cada resolución. En la Ilustración 37 se indica el valor por la resolución estándar de 640 x 480 pixeles.

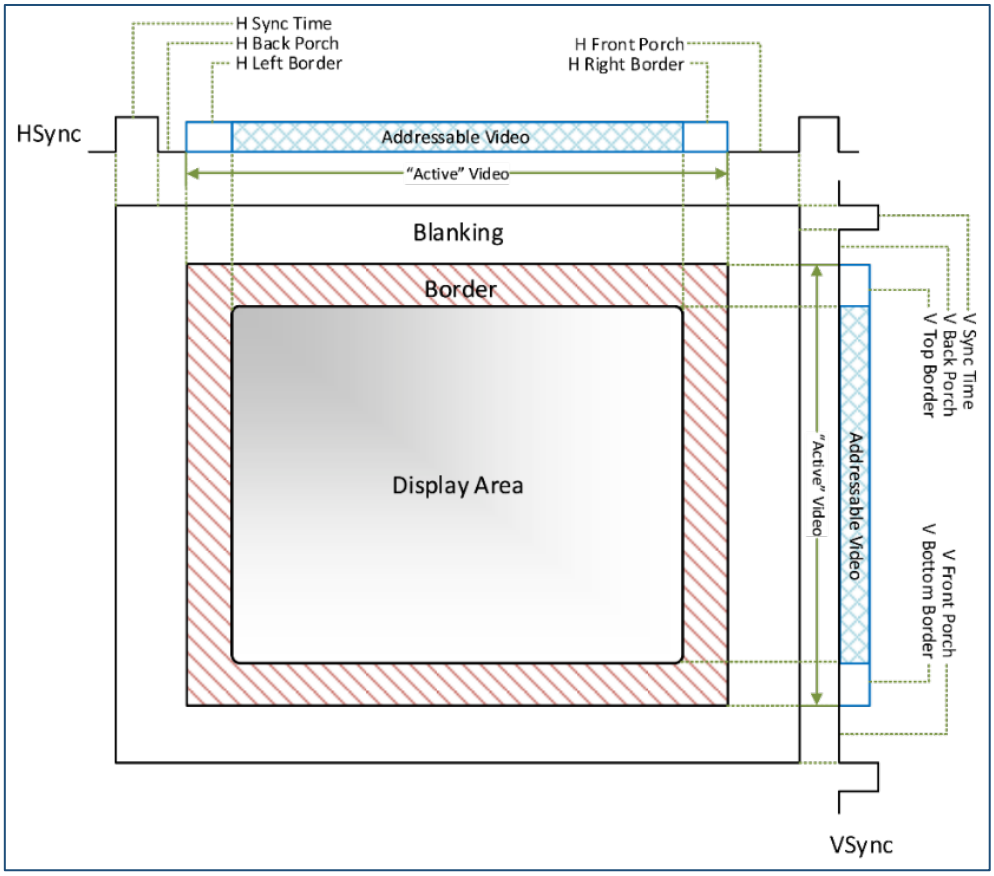


Ilustración 37: Estructura del video VGA (Digilent, 2014)

Las pantallas VGA de 640 x 480 pixeles, tienen un margen horizontal izquierdo de 16 pixeles, y otro derecho de 48, donde no se muestra información alguna, a modo de márgenes en el monitor. También dispone de un borde para la sincronización horizontal de 96 pixeles. De manera similar, en el eje vertical, se dispone de un margen superior de 10-11 pixeles y un margen inferior de 31-33 pixeles, con una sincronización vertical de 2 pixeles. Esto en su cálculo total implica que se necesitan 800 x 525 pixeles para utilizar VGA con la resolución estándar mencionada.

Mirando las contantes definidas en el fichero “vga.v” del código (Ilustración 38), se tienen los siguientes valores:

```
// video structure constants
parameter h_pixels = 800;   // horizontal pixels per line
parameter v_lines = 525;    // vertical lines per frame
parameter h_pulse = 96;     // hsync pulse length
parameter v_pulse = 2;      // vsync pulse length
parameter hbp = 144;        // end of horizontal back porch
parameter hfp = 784;        // beginning of horizontal front porch
parameter vbp = 35;         // end of vertical back porch
parameter vfp = 515;        // beginning of vertical front porch
```

Ilustración 38: Captura de los parámetros del video VGA utilizado

Estas constantes indican lo siguiente:

- H_pixels: 800 pixeles, el total de pixeles que necesita en la pantalla en horizontal.
- V_lines: 525 pixeles, total de pixeles necesarios en el eje vertical.
- H_pulse: Longitud de la sincronización horizontal (96px).
- V_pulse: Longitud de la sincronización vertical (2px).
- Hbp: Margen posterior horizontal, suma de 96 y 48. Sincronización + margen posterior.
- Hfp: Margen frontal horizontal, suma de la resolución real (640) y el margen posterior (96) y la sincronización horizontal (96). Lo mismo que la diferencia del tamaño real (800px) y el margen frontal (16).
- Vbp: De manera análoga a "hbp", suma de Sincronización (2) con el margen posterior vertical (33)
- Vfp: Análogo a "hfp", suma de resolución real (480) con sincronización (2) y margen posterior (33). O la diferencia de la resolución total (525) menos el margen frontal (10).

Para generar de manera sencilla la salida de video deseada, se establece que cada línea ocupa 20 pixeles, por tanto, con los 480 pixeles se obtienen 24 líneas, que son las que aparecen en la ejecución (Ilustración 39). También se calcula que cada carácter ocupa 16 pixeles horizontalmente y, por tanto, se obtiene que cada línea va a tener 40 caracteres.

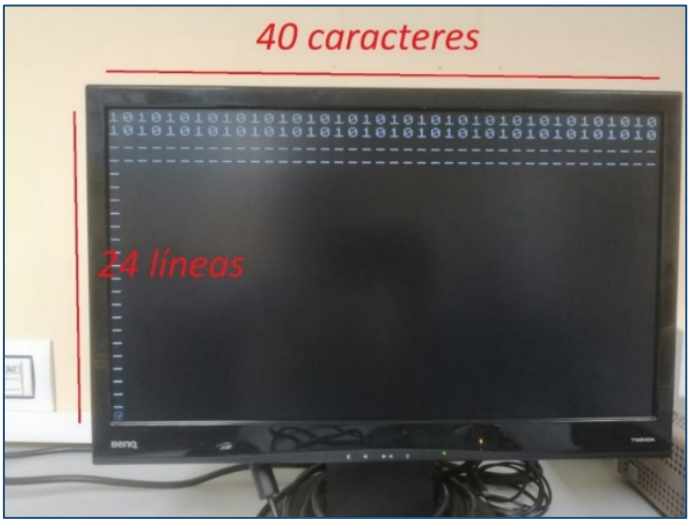


Ilustración 39: Imagen de la resolución del video VGA del Apple I

Una vez establecido el máximo de caracteres disponibles de debe establecer de donde se obtienen esos caracteres, siendo fichero, que se adjunta al código, "vga_font_bitreversed.hex", donde se dispone de todos los caracteres imprimibles. Como no todos los caracteres son imprimibles en el

Apple I, solo mayúsculas, números y símbolos de ASCII estándar, en el fichero que se utiliza se codifican los caracteres en 10 líneas, aunque utilice después el doble (20) para mostrarlo en la pantalla.

El sistema permite cambiar el formato de la fuente, pudiéndose elegir entre Normal, Líneas Verticales, Líneas Horizontales o Puntos, sin más que modificar el contenido de la dirección 0xC000, como se ve en la imagen (Ilustración 40), esta implementación no estaba permitida por la máquina original, pero se gestiona usando su propio "ensamblador".

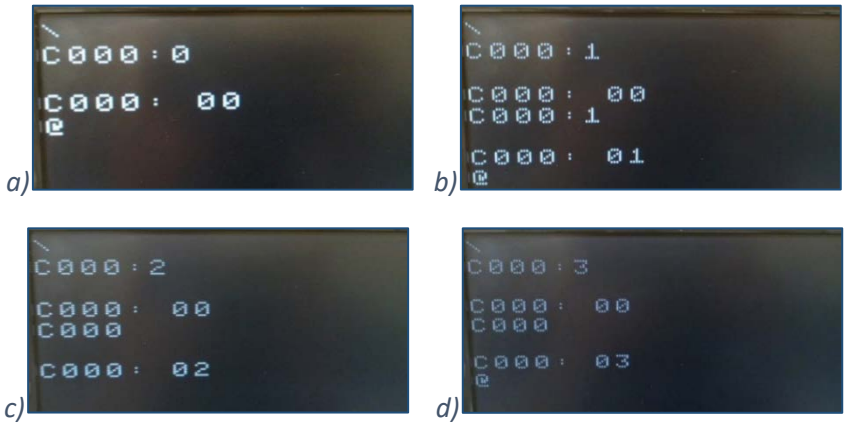


Ilustración 40: Personalización de la fuente del Apple I. a) Normal; b) Líneas Verticales; c) Líneas Horizontales; d) Puntos

En el caso de la placa que se utiliza, dispone de una salida VGA de 8 bits, debido a lo cual, para que se vea de manera correcta, se ha optado por usar únicamente 3 bits de los 8 disponibles, al igual que en el proyecto en el cual se basa, se ha utilizado el bit inicial de cada configuración de colores del RGB que ofrece esta salida. La codificación RGB permite una variación de 3 bits, por tanto 8 colores (Ilustración 41), tanto para el fondo como para la fuente, esto, al igual que el cambio del formato de fuentes, es un añadido a la maquina real.

Estos cambios de color, se consiguen modificando el contenido de la dirección 0xC001 para la fuente y en la dirección 0xC002 para el fondo.

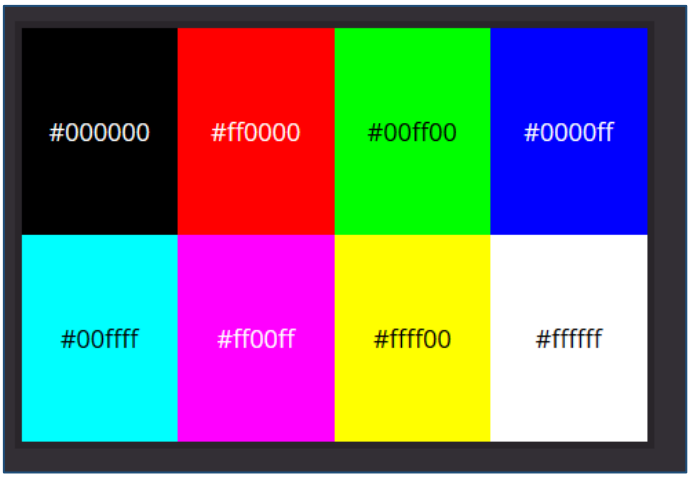


Ilustración 41: Colores posibles para personalizar el Apple I (3 bits -> 8 colores) (3-Bit RGB Palette, s.f.)

Esta funcionalidad, que permite el cambio entre estos 8 colores en el color del fondo y de las letras, añade una personalización, no disponible en el Apple I, siendo una incorporación a su emulación.

4.3.8. Display/Casete

Se ha considerado interesante añadir un apartado con el uso del display, del que dispone la placa FPGA empleada, realizado, un elemento que no se utiliza en el proyecto original ni disponía el Apple I de él. En este caso, tiene un uso informativo para el usuario al elegir el programa a cargar en memoria desde la "memoria externa" que implementa la funcionalidad del casete del que disponía el Apple I.

Para iniciar, los valores que se utilizan son únicos y están generados fuera de ejecución, por tanto, añadir un programa nuevo, no crea un nuevo valor. Está limitado a 10 elementos, desde el '0' hasta el '9', y una breve información en 3 caracteres, como se puede ver en la siguiente imagen (Ilustración 42), puesto que el display empleado está formado por cuatro displays de siete segmentos más punto decimal, dedicándose el primero al dígito numérico del orden y los tres restantes a la breve descripción del nombre del programa.

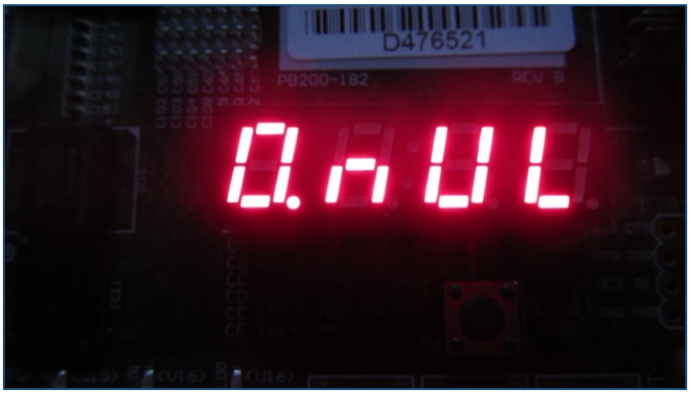


Ilustración 42:Display de 7 segmentos de la placa

El display funciona con el reloj del sistema (100 MHz) aunque podría funcionar con una frecuencia de hasta 50 MHz, habiéndose probado, incluso, con una frecuencia de 25 MHz, ocasionando parpadeos constantes y notables. Para evitar esto, además de aumentar la frecuencia, sólo es posible avanzar a la siguiente posición cuando el valor previo del registro utilizado ha sido '1' (no pulsado) y este acaba de cambiar.

Con relación al código desarrollado, se han insertado valores por defecto para cada posición del "array" de programas (Ilustración 43), dado que hacerlo automático, no podría ofrecer esta opción, sin ocupar mucho código. Estos valores de 32 bits, indican el valor que debe aparecer en el display.

```
case(pick)
   0: x = 32'b01000000101010111100000111000111; //0.nUL
   1: x = 32'b01111001100010001011000011000000; //1.A30
   2: x = 32'b00100100100010111000011011000111; //2.hEL
   3: x = 32'b00110000110001111100000110101011; //3.LUn
   4: x = 32'b00011001100010011000011010001001; //4.HEH (MEM)
   5: x = 32'b00010010111000111100011010001011; //5.uCh
   6: x = 32'b00000010100011001000100010010010; //6.PAS

   default: x = 32'b10000000100000001000000010000000; //8888

endcase
```

Ilustración 43: Display por defecto

Después, cada digito del display, recoge los valores correspondientes a su posición para mostrarlo y se realiza de manera secuencial, digito a digito. En el Anexo III, se detallará la creación de este valor de 32 bits.

4.3.9. Problemas adicionales

Con el objetivo de resolver o justificar algunos problemas durante el desarrollo, ajenos al código desarrollado, se ha pensado conveniente realizar este apartado.

Comenzando por añadir el emulador de otro ordenador en la misma placa FPGA. Esta idea se sugirió en un principio incorporar al proyecto esta funcionalidad, y el problema que se ha encontrado, es la poca capacidad de la cual dispone la placa FPGA seleccionada. Por ese motivo, a pesar de ocupar poco espacio el código, no se puede garantizar que otro sistema de la misma gama, como podría ser el Apple II, cupiese dentro de la misma placa FPGA. Debido a que del Apple II en adelante, se disponía de un Sistema Operativo, y otras funcionalidad de gran capacidad, se ha optado por no incorporar en un inicio esta funcionalidad para alternar maquinas.

La utilización del puerto Ethernet para una transmisión por SSH, se ha descartado, porque el sistema ya incorpora un mecanismo similar, la UART. El cometido que se ofrecería sería el mismo que el que ya ofrece la UART, por tanto, se consideró innecesaria su implementación..

Para no perder la esencia del Apple I o su estilo original, no se han utilizado los puertos PMod para uso similar a los switches y botones, debido a que la placa dispone de ellos, dejando un estilo más similar al Apple I.

En el apartado de objetivos, también se comentó la realización de una carcasa impresa usando una impresora 3D, que ha sido imposible realizar, por no disponer de impresora 3D en el proceso de desarrollo del proyecto.

Un problema ocurrido en el entorno de desarrollo ISE de Xilinx, es la gran cantidad de *Warnings* en la compilación del código. Una gran parte de estas advertencias se deben a una falta de inicialización de los elementos y variables, que el sistema ya inicia al ejecutar el código. Otras se deben a que el bit izquierdo de varias variables se desborda y se pierde un nuevo bit, que el sistema debería omitir, dado que no es necesario en las siguientes ejecuciones.

De estas advertencias hay una específica que si es preocupante, pero no ocasiona ningún fallo, pues el sistema ya se ajusta a lo necesario. Esta advertencia se refiere a la forma de almacenamiento en bloques de la placa, que se limita a 9 K bits, habiendo ciertos programas, en concreto la CPU del Apple I, que supera ese tamaño. Su resolución es sencilla, siendo suficiente, que el sistema, asigne los bloques de menor a mayor tamaño, pero esta configuración se debe cambiar internamente, por Xilinx o sabiendo que variables tocar. En cualquier caso, esto no supone ningún inconveniente, pues el sistema particiona o busca algún bloque donde pueda entrar de manera sencilla.

4.4. Planificación del proyecto

Tras la finalización del proyecto, se han actualizado los hitos realizados (Tabla 7) y el diagrama de Gantt finalmente cumplido.

1- Informarse sobre la placa FPGA y Apple I
1.1- Conocer las características de la placa FPGA
1.2- Conocer las necesidades del Apple I
1.3- Saber su compatibilidad y preparación

2- Instalación de Apple I en la placa FPGA
2.1- Realizar cambios necesarios en el código
2.2- Instalación en la placa FPGA
3- Configuración y prueba
3.1- Prueba de todos los dispositivos básicos
3.2- Rectificación de algún error
4- Configuración de una manera de cargar los programas
4.1- Utilizar el display para indicar el programa a cargar
4.2- Realizar una correcta copia en la memoria RAM
5- Configurar los botones/display/switch
6- Resolución de errores
7- Creación de controladores
7.1- Controlador de teclado
7.2- Pruebas y cambios
8- Creación de Anexos
8.1- Anexo de uso de BASIC
8.2- Anexo del uso inicial del emulador
8.3- Anexo para la carga de un programa nuevo

Tabla 7: Tabla de hitos final (Mayo 2019)

Como se puede ver, se han cambiado algunos hitos respecto a la tabla de hitos inicial (Tabla 4), suprimiendo algunos e incorporando otros. También se ha actualizado el Diagrama de Gantt con las desviaciones correspondientes (Ilustración 44), en comparación con el diagrama inicial que se había creado (Ilustración 20).

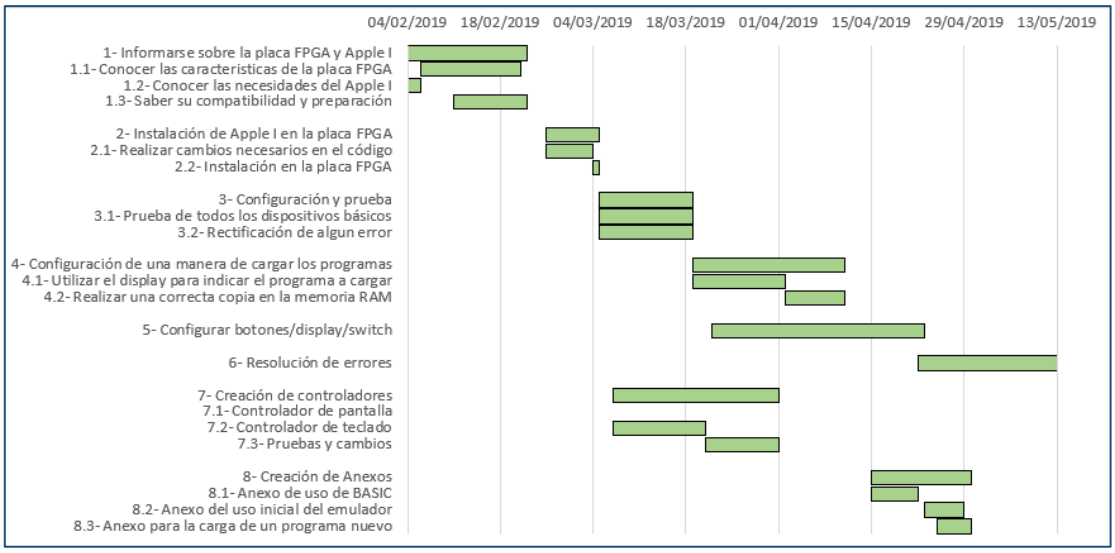


Ilustración 44:Diagrama de Gantt Final (Mayo 2019)

Entre las desviaciones que han ocurrido está la supresión de los hitos opcionales y una rectificación con respecto a la memoria RAM del sistema. También se ha incluido un hito con los anexos que se han creído convenientes para un entendimiento completo del proyecto.

En la siguiente tabla (Tabla 8), se indica de forma más clara a como se muestra en los diagramas de Gantt vistos, los tiempos que han sido esperados y el tiempo de los plazos que el proyecto ha tomado finalmente.

Hitos	Tiempo	Tiempo Real	Tiempo	Tiempo Real	Días	Días Reales
1- Informarse sobre la placa FPGA y Apple I	04/02	04/02	11/02	22/02	7	18
1.1- Conocer las características de la placa FPGA	04/02	06/02	06/02	21/02	2	15
1.2- Conocer las necesidades del Apple I	04/02	04/02	06/02	06/02	2	2
1.3- Saber su compatibilidad y preparación	06/02	11/02	11/02	22/02	5	11
2- Instalación de Apple I en la placa FPGA	11/02	25/02	04/03	05/03	21	8
2.1- Realizar cambios necesarios en el código	11/02	25/02	25/02	04/03	14	7
2.2- Instalación en la placa FPGA	25/02	04/03	04/03	05/03	7	1
3- Configuración y prueba	04/03	05/03	12/03	19/03	8	14
3.1- Prueba de todos los dispositivos básicos	04/03	05/03	12/03	19/03	8	14
3.2- Rectificación de algún error	04/03	05/03	12/03	19/03	8	14
4- Configuración de una manera de cargar los programas	14/03	19/03	28/03	11/04	14	23
4.1- Utilizar el display para indicar el programa a cargar	14/03	19/03	17/03	02/04	3	14
4.2- Realizar una correcta copia en la memoria RAM	17/03	02/04	28/03	11/04	11	9
5- Configurar botones/display/switch	28/03	22/03	25/04	23/04	28	32
6- Resolución de errores	25/04	22/04	23/05	13/05	28	21
7- Creación de controladores	14/03	07/03	01/04	01/04	18	25
7.1- Controlador de pantalla	14/03		21/03		7	0
7.2- Controlador de teclado	21/03	07/03	28/03	21/03	7	14
7.2- Pruebas y cambios	28/03	21/03	01/04	01/04	4	11
8- Uso de otras máquinas Apple (Opcional)	01/04		25/04		24	0
8.1- Apple II	01/04		13/04		12	0
8.2- Apple III	13/04		25/04		12	0
9- Creación de demo nueva (Opcional)	25/04		09/05		14	0
10- Diseño e impresión de carcasa 3D(Opcional)	12/03		09/05		58	0
10.1- Diseño de la carcasa	12/03		19/03		7	0
10.2- Conversión a formato imprimible	19/03		26/03		7	0
10.3- Impresión	26/03		09/05		44	0
8- Creación de Anexos		15/04		30/04	0	15
8.1- Anexo de uso de BASIC		15/04		22/04	0	7
8.2- Anexo del uso inicial del emulador		23/04		29/04	0	6

Hitos	Tiempo	Tiempo Real	Tiempo	Tiempo Real	Días	Días Reales
8.3- Anexo para la carga de un programa nuevo		25/04		30/04	0	5

Tabla 8: Tabla con fechas y tiempos

Como se puede ver en los diagramas de Gantt, (Ilustración 20) y (Ilustración 44), el inicio del proyecto se comenzó en las mismas fechas, pero debido a inconvenientes durante la selección de la placa y su compatibilidad, este primer hito se alargó en gran volumen. Pero en el segundo hito, donde se instaló el programa se aceleró el proceso, dado a la gran compatibilidad del proyecto ya realizado y la placa, siendo más sencilla su instalación.

También se demoró mucho durante las pruebas y la configuración de una manera de cargar los programas, siendo la parte más compleja y donde se tenía que comprobar que todos los componentes funcionaban de manera correcta. Para acelerar el desarrollo del proyecto, se comenzó a desarrollar los botones y switches al mismo tiempo, a medida que se realizaba el cargador de programas. Finalmente, se recuperó tiempo durante la resolución de errores, resolviendo el máximo de errores de Verilog o de ciertos programas, que se han preferido no especificar.

Un añadido que no se contemplaba introducir, es la escritura de los tres anexos que se han creído convenientes añadir para explicar el uso de BASIC, la utilización del sistema y la carga de un programa no listado. Estos han sido coordinados durante la búsqueda de errores, para pensar una solución a los errores o los componentes que pueden producir un error. Los hitos que no se han realizado ha sido por la falta de tiempo y la inexperiencia en el uso de la herramienta que se pretendía utilizar. Por tanto, se propone que, en líneas futuras, estos hitos se completen.

4.4.1. Tabla de costes

De manera análoga, se ha decidido realizar una tabla con los costes que ha supuesto el desarrollo de este proyecto, valorando diferentes opciones (Tabla 9). Y en verde, la opción finalmente seleccionada y su motivo.

Proyecto	Coste	Motivo
Apple One en FPGA	250 €	Seleccionado por facilidad, y conocimiento de la tecnología
Apple One en Arduino (Arduino, 2019)	60 €	Falta de experiencia con la placa
Apple One en módulo ESP8266	30 €	Bajo coste, pero falta de I/O físicos

Tabla 9: Elección de proyecto de Apple I con precios

Se ha realizado una tabla con los precios que habría supuesto la realización del proyecto con materiales inicialmente (Tabla 10), y la tabla de los precios finales (Tabla 11), como se puede ver, no se ha modificado el coste final del proyecto.

Componente	Precio	Enlace (en caso necesario)
Apple I		
Placa FPGA (Nexys-3 Spartan 6)	Descatalogada (~200€)	Departamento
^ Puede variar el modelo		
Añadidos		
Pantalla VGA	30€	Departamento
Teclado USB (Ingles)	12€	RS Components
Conversor USB a UART	14€	RS Components
Cable M-M (10 uds)	5€ (aprox.)	
^ Para conectar a UART y Terminal		

Componente	Precio	Enlace (en caso necesario)
Carcasa		
Impresora 3D (Alquiler)		
Filamento impresora 3D (PVC)	(Depende de la impresora 3D)	
COSTE FINAL:	**>200€ completo**	

Tabla 10: Tabla de costes inicial (Enero 2019)

Componente	Precio	Enlace (en caso necesario)
Apple I		
Placa FPGA (Nexys-3 Spartan 6)	Descatalogada (~200€)	Departamento
^ Puede variar el modelo		
Añadidos		
Pantalla VGA	30€	Departamento
Teclado USB	10€	Departamento
Cable USB-microUSB	5€	
^ Para conectar a UART y Terminal		
Carcasa		
^ Se ha optado por no realizar la carcasa por falta del diseño.		
COSTE FINAL:	**250€**	

Tabla 11:Tabla de costes final (Mayo 2019)

5. Aspectos Sociales, Ambientales, Éticos y Legales

En este capítulo se va a enfocar en el apartado de los Aspectos Sociales, Ambientales, Ético y Legales que atañen al proyecto realizado. Primero se identificará los aspectos significativos de él, posteriormente una selección de los más importantes a resolver y su integración. Finalizando con una valoración final sobre estos Aspectos.

5.1. Identificación de Aspectos Relevantes

Se va a analizar e identificar los aspectos relevantes que afectan al proyecto desarrollado, para este propósito se van a responder las siguientes cuestiones.

- *¿Cómo se resuelve actualmente el problema que pretendo resolver?*

  El problema que se pretende resolver es la adaptación a la actualidad de un ordenador de 1976 como es el Apple I. Hay diversas formas de resolver este problema, desde la creación de un nuevo circuito para recrear íntegramente el Apple I sobre él hasta la realización de una simulación por software usando un ordenador personal para su ejecución. En este caso, se ha optado por utilizar una placa FPGA, que es un punto intermedio, en cuanto a la implementación de los anteriormente citado. Y sobre este dispositivo hardware se realizarán modificaciones y mejoras de un proyecto ya existente similar al objetivo del presente.

  Por tanto, se ha decidido optar por una solución que en muchos casos viene a ser una opción muy interesante por permitir una fácil migración de partes del desarrollo del hardware al software y viceversa, además de su cada vez mayor uso en la industria, que es la utilización de placas FPGA.

- *¿Existe una necesidad real del proyecto?*

  Una respuesta rápida seria “No”. Porque existe multitud de ordenadores personales que son capaces de emular el Apple I por software, y una persona desinteresada, lo vería como una pérdida de tiempo, el realizar una máquina de hace 43 años porque existen ordenadores mucho más potentes.

  Desde el punto de vista de un interesado por la computación retro, un ámbito que está muy en boga actualmente, es una respuesta más elaborada, resumiéndose que es más que interesante su realización. Para explicar esto, y como ejemplo, no se vería con buenos ojos que se derribase una catedral de siglos pasados porque hay infraestructuras mucho más ingeniosas y elaboradas. En el caso de este proyecto es lo mismo, para llegar al tiempo actual, se debe haber pasado por un inicio básico de los ordenadores personales, su tecnología y diseño. La computación en estos días llamada “retro” es necesaria, es una parte vital de la historia de la computación, no se podría haber llegado a un ordenador actual sin haber pasado por circuitos integrados en algún momento, y estos a su vez, sin la existencia de válvulas de vacío que iniciaron todo este mundo.

  La computación “retro” también es una manera de enseñar a las próximas generaciones el gran avance que se ha realizado, y los puntos en su historia que han ocasionado esto. Existen muchas personas que se interesan por estos temas, y un proyecto como este puede ser de gran interés para aquellos que crean que se deben recrear estos ordenadores de décadas pasadas.

- *¿El proyecto permitirá mejorar, directa o indirectamente, la calidad de vida de las personas?*

  De cierta manera, el proyecto mejorará el conocimiento de la existencia de ordenadores de generaciones pasadas, y en concreto del Apple I que fue un ordenador muy limitado y nunca salió a la venta fuera de Estados Unidos. A pesar de parecer ahora un armatoste sin ningún uso, fue un proyecto muy ambicioso de Steve Wozniak, de sus ideas y su habilidad para conseguir lo que quería.

  Sobre el tema de mejorar la calidad de vida, es un gran apoyo para las personas que quieran emprender su futuro y no se sientan seguros de ello, de alguna manera, es un proyecto motivador a la par que lúdico para próximas generaciones. No sólo se refiere a este proyecto, cualquier proyecto que puede estar relacionados con la computación "retro" o informática "retro" viene a tener las mismas iniciativas motivadoras y lúdicas.

- *¿Hay algún colectivo que podría verse perjudicado? ¿En qué medida?*

  Dado que existen pocas empresas que se dediquen a realizar recreaciones o emulaciones de máquinas clásicas, y en caso de hacerlo tienen su ámbito concreto, como el ámbito de los videojuegos con la NES (Super Famicon/NES, s.f.) o Atari 2600 (Gómez Hinestrosa, s.f.), que decidió utilizar una modificación del MOS 6502 más económica, el MOS6507 (Durán, 2000). Existiendo muy pocas compañías dedicadas a la computación "retro" o informática "retro", existiría un colectivo muy limitado afectado, prácticamente nulo pues, entre otras cosas, estos colectivos siempre ven con buenos ojos el que se recreen máquinas históricas en algún aspecto diferente a las recreaciones pasadas.

  Pero en muchos casos este proyecto es realizado con un objetivo de entretenimiento, y no enfocado a la venta al público.

- *¿El proyecto permitirá reducir el uso de recursos, la generación de residuos, contaminación o algún otro aspecto de la huella ecológica?*

  El uso de FPGAs resuelve todos estos aspectos medioambientales. Usando una placa FPGA se permite la reprogramación de la placa, por tanto, si se decidiera que no es ya satisfactorio el sistema, se instala otro sin ningún gasto de recursos. En el proyecto en cuestión, se podría disponer de la gama de máquinas Apple de la década de los 70: Apple I, Apple II, Apple III y/o Apple IIe, ahorrando muchos recursos al instalarlos sobre el mismo dispositivo hardware en lugar de tener uno para cada sistema. En relación con la generación de residuos, el único residuo es el ocasionado por el reciclaje de una placa FPGA en caso de esta averiarse o quemarse, siendo, en cualquier caso, una placa "reciclable" en el aspecto de la programación.

  La contaminación provocada es mínima, al ser la FPGA una placa con funcionamiento eléctrico. Pero seguirá consumiendo menos energía que el Apple I original o una recreación en circuito impreso o con circuitos integrados. Siento la opción que menos afecta en la huella medioambiental de todas las opciones barajadas en un principio.

- *¿Hay normativas, leyes o regulaciones que he de tener en cuenta al desarrollar el proyecto?*

  Existe un aspecto relacionado con la ley de propiedad intelectual, que no se ha tenido en cuenta al desarrollar el proyecto. Si este se realizase buscando beneficio, la propiedad intelectual de la marca Apple podría afectar, a pesar de ser una maquina Apple de 1976 la que

se pretende recrear, sin su consentimiento, lo cual podría llegar a provocar el emprender acciones legales al recrear una maquina suya sin su consentimiento.

En cualquier otro caso, como puede ser la placa FPGA, Xilinx es una empresa que desarrolla placas por Europa y alrededor de todo el mundo, todas las regulaciones referentes a la normativa europea sobre los productos y circuitos que se venden, se pueden entender que los cumplen, al comercializarlos en territorio europeo. Aun así, no estaría de más revisar estos, antes de comenzar a desarrollar el proyecto en un futuro, si este se llega a comercializar.

5.2. Selección

Al ser un proyecto basado en emular una máquina, el principal impacto que se quiere abordar es reducir en todo lo posible su consumo y contaminación, propósitos que en los inicios de la informática no se tuvieron demasiado en cuenta. Para alcanzar esta meta se ha utilizado una placa FPGA, dado que la tecnología reprogramable que ofrece es un gran avance, reduciendo muchos gastos.

Otro impacto, en este caso beneficioso que aparece, es la necesidad , en muchas ocasiones, de recrear una tecnología de antaño. Con esto se ofrece a muchas personas, probar o visualizar el funcionamiento de una maquina primigenia de la computación, para mostrar los grandes avances que se han realizado, pero, también, las condiciones con las que trabajaban, usando maquinas a 1 MHz (en los 70) comparado con los procesadores multinúcleo a varios GHzs actuales. O retrocediendo más, la vertiginosa velocidad de un ordenador con válvulas de vacío, comparándola mismamente con ese 1 MHz del microprocesador MOS 6502. También ver el tamaño con el cual se han replicado y reducido los componentes, pero siguen funcionando igual.

Todos estos impactos, se consideran importantes en el proyecto a realizar dado que son temas en los que estar muy concienciados en estos momentos, como el impacto medioambiental. También el impacto social beneficioso que origina y, quizás, en menor detalle, el legal que podría ocurrir al recrear una máquina Apple sin su consentimiento.

5.3. Integración en el proyecto

Para resolver estos impactos seleccionados, se han propuesto los siguientes objetivos:

- Encontrar una placa FPGA de bajo consumo y de suficiente potencia: Es un requisito sencillo, cualquier placa FPGA es de bajo consumo energético, también, al ser reprogramable, ahorra gastos de reciclaje e impactos medioambientales que podrían ocurrir en el caso de desechar una máquina con sólo una finalidad en caso de generar algún inconveniente.
- Encontrar una máquina que sea poco conocida o que tenga interés en emular: Se ha elegido el Apple I, una máquina que marcó el inicio de “La Gran Manzana” o Apple Computers, que fue un gran proyecto para Wozniak, su creador y sus unidades a la venta fueron muy limitadas, apenas 200, que sólo se comercializaron en Estados Unidos. Una máquina tan extraña y limitada, necesita ser emulada en muchos casos, y aunque existan proyectos que la repliquen con circuitos integrados, o en emulación software, sigue siendo interesante hacerlo sobre recientes e innovadores dispositivos hardware.
- Sobre el impacto legal que pueda generar, no se va a llegar a comercializarlo, por tanto, es un impacto irrelevante. Es cierto que al ser un proyecto de software libre y de libre acceso no es un grave impacto legal, pero siempre conviene tenerlo en cuenta.

- Otro requisito que se ha considerado cumplir es una mejora del código ya disponible, añadiendo mayor funcionalidad o siendo un código más entendible.

5.4. Valoración Final

Como valoración final del proyecto, se considera que este proyecto va a mejorar la calidad de vida de las personas, al menos de la gente interesada en la computación "retro", dando la oportunidad de ver una versión mejorada del Apple I sin perder su característico estilo o su funcionalidad, dado que la emulación realizada, además de ser más respetuosa con el medio ambiente, combina de una manera acertada la funcionalidad original con las tecnologías modernas.

Sobre el tema de la viabilidad económica, actualmente, no es un proyecto interesante comercialmente, dado que únicamente incorpora un Apple I mejorado, pero está disponible, para introducir otras máquinas de similar funcionamiento, reprogramando la FPGA, e incluso disponiendo de ella en la misma placa FPGA simultáneamente, en ese caso, sería más rentable, disponiendo del funcionamiento de varias máquinas en una misma placa FPGA, que incluso replica su funcionamiento.

El único inconveniente personal ha sido posiblemente una mala selección de la placa FPGA, disponiendo de una placa con menores prestaciones pero que por un inconveniente de tiempo personal, se ha optado por tener ciertos componentes ya disponible, aunque eso ha costado "retroceder" en algunos conceptos del tipo software y hardware, principalmente, en la carga externa de programas, y el entorno seleccionado que ha supuesto elegir una placa FPGA de una generación anterior. Aun así, este cambio ha mejorado mi conocimiento sobre algunas tecnologías y sobre la placa FPGA seleccionada. En el caso de volver a hacer el proyecto, aunque se apreciarían pocos cambios a los realizados, seguramente, se hubiese seleccionado otro modelo de placa FPGA más moderna y añadido algunas funcionalidades adicionales.

6. Conclusiones

Tras finalizar el proyecto, o una gran parte de él, dado que es un proyecto que se puede ampliar constantemente, se ha decidido exponer las conclusiones a las que se han llegado.

En primer lugar, se considera que el proyecto se ha realizado de manera exitosa, se ha conseguido una emulación fiel a la maquina original, usando una placa FPGA. El cumplimiento de los hitos indicados y el resultado obtenido, deja ver que el objetivo se ha cumplido.

Respecto al lenguaje Verilog, este proyecto me ha servido para aumentar el conocimiento sobre él. Principalmente, comparado con otro lenguaje de programación de placas de desarrollo de FPGA como puede ser VHDL, es su gran selección de instrucciones. Pero muchas de estas sólo funcionan en simulación, requiriendo de una modificación de las mismas para su síntesis e implementación. En resumen, el lenguaje Verilog tiene una mejor comunidad a la hora de solucionar errores, pero también ofrece muchas opciones que son imposibles de programar en una FPGA, pero es un lenguaje sencillo con el que trabajar.

La siguiente conclusión a la cual he llegado, es el aumento del conocimiento sobre placas FPGA de Xilinx, en este caso, era una placa ya descatalogada, pero que fue muy usada, incluso actualmente, por lo que dispone de mucha gente que te puede ayudar en caso de encontrar algún problema.

Pasando a las conclusiones sobre la maquina desarrollada, el interés sobre la informática retro o “RetroComputing” no ha cambiado, y si ese fuese el caso, sólo lo ha aumentado, al ser un ámbito de gran atractivo. Por tanto, mi opinión no ha sido modificada tras la realización de este proyecto, sigo pensando que era un ordenador adelantado a su época, aunque después haya sido muy superado, y sobre todo, fue un proyecto creado del ingenio y habilidades de Stephen Wozniak.

Sobre el trabajo realizado, ha mejorado mi capacidad de programar y de organizar proyectos. También me ha ayudado a entender la complejidad que supone programar o emular un microprocesador conocido, como lo fue el MOS6502, además de conocer la historia del Apple I y sus componentes.

En resumen, este proyecto ha aumentado y consolidado mi aprendizaje y ha incrementado mi interés sobre el tema de la informática retro, del cual no he cambiado mi opinión sobre la importancia invertir en este tema para las próximas generaciones.

7. Líneas futuras

Se ha estimado de gran utilidad proponer mejoras y posibles líneas futuras que se pueden abordar sobre este proyecto.

En primer lugar, un completo cambio radical, sería realizarlo sobre circuitos impresos, los cuales se han rechazado dado al poco tiempo del cual se disponía. Pero también se puede ofertar en realizar en una placa FPGA de mayores prestaciones.

También se planea incorporar un mecanismo para seleccionar distintas distribuciones de teclado, para ofrecer mayor diversidad a la hora de utilizar el sistema desarrollado, como podría ser el teclado con distribución “española” o “europea”, y no únicamente la ordenación americana.

Sobre este mismo proyecto, una línea futura, sería la mejora del cargador de programas, para que este ofrezca de manera dinámica, una forma para cargar nuevas aplicaciones en memoria. Entre las formas de cargar nuevos programas “en caliente”, sería la programación de un controlador para un USB o lector de tarjetas, y disponer de una memoria externa de manera real, y no de manera ficticia con ficheros. También a la hora de mostrar el programa al cual se apunta, que se genere de manera automática, indicando qué programa es, sin valores predefinidos.

Con una ligera relación con la línea previa, también se podría mejorar para que actúe de manera automática, la carga de programas, buscando la dirección donde comenzar y finalizar. También se encontraría la forma con la cual poder cargar programas en BASIC, debido a que actualmente sólo acepta programas en hexadecimal, como se cargaban en los casetes de la época.

Por último, se propone la ampliación en este proyecto a aceptar otras máquinas con el mismo microprocesador, como podría ser el Apple II o la NES, en la misma placa.

Para finalizar, como un proyecto a largo plazo, se propone la continuación y la oferta de proyectos fin de grado relacionados con la “Informática Retro” para futuros años, donde se puedan recrear por medio de FPGAs o por medio de circuitos impresos el funcionamiento de conocidas maquinas que actualmente pocas de ellas funcionan y mostrar el funcionamiento a futuras generaciones.

8. Referencias

3-Bit RGB Palette. (s.f.). Obtenido de Lospec: https://lospec.com/palette-list/3-bit-rgb

About the EDA Industry. (2015). Obtenido de Electronic Design Automation Consortium: https://web.archive.org/web/20150802073506/http://www.edac.org/industry

alain. (9 de Mayo de 2014). *How to identify the USB to Serial wire mismatched?* Obtenido de Raspberry Exchanges: https://raspberrypi.stackexchange.com/questions/15819/how-to-identify-the-usb-to-serial-wire-mismatched

alker33. (18 de Marzo de 2012). *My working original 1976 Apple I - finally I got it to run - Apple 1 in 2012*. Obtenido de YouTube: https://www.youtube.com/watch?v=ysMfjU--h8U

Alpern, D. A. (s.f.). *El microprocesador 8080*. Obtenido de Alpertron Argentina: https://www.alpertron.com.ar/8080.HTM

Apple Computers. (1976). *Apple 1 Basic Manual.* Palo Alto, California. Recuperado el 16 de 4 de 2019

Apple I Case. (s.f.). Obtenido de Wired: https://www.wired.com/wp-content/uploads/images_blogs/gadgetlab/2011/10/apple_one_f.jpg

Apple Inc. (2019). *Apple*. Obtenido de https://www.apple.com/es/

AppleDOS. (s.f.). Obtenido de Apple2History: https://apple2history.org/history/ah14/#01

Arduino. (2019). Obtenido de https://www.arduino.cc/

Atlys Spartan-6. (s.f.). Obtenido de Digilent: https://store.digilentinc.com/atlys-spartan-6-fpga-trainer-board-limited-time-see-nexys-video/

BASIC. (2019). Obtenido de True BASIC: http://www.truebasic.com/

Basys-3 Artyx-7. (s.f.). Obtenido de Digilent: https://store.digilentinc.com/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/

Bishop, P. (1982). *Further Computer Programming in BASIC.* Anaya 1990. doi:84-7614-013-4

Breker. (2019). Obtenido de Breker: http://www.breker.com/english/index.htm

BREKER KÖLN. (24 de Octubre de 2012). *Original »Apple 1 Computer«, 1976; 24 November 2012 - Auction*. Obtenido de YouTube: https://www.youtube.com/watch?v=4l8i_xOBTPg

Briel, V. (27 de Mayo de 2017). *Replica 1*. Obtenido de Briel Computers: http://www.brielcomputers.com/wordpress/?cat=17

Briel, V. (2019). *Homepage*. Obtenido de Briel Computers: http://www.brielcomputers.com/wordpress/

Čavrak, H. (20 de Enero de 2018). *Espple*. Obtenido de GitHub: https://github.com/hrvach/espple

Čavrak, H. (2019). *hrvach*. Obtenido de GitHub: https://github.com/hrvach

CAVSI. (s.f.). *¿Qué es Puerto PS/2?* Obtenido de CAVSI: http://www.cavsi.com/preguntasrespuestas/que-es-puerto-ps2/

Cmple. (2019). *RCA Composite Video, Subwoofer S/PDIF Cable*. Obtenido de Cmple: https://www.cmple.com/rca-composite-video-subwoofer-spdif-cable-coax-75-feet

Contreras, A. E. (s.f.). *Programma 101*. Obtenido de Museu d'Informàtica: http://museo.inf.upv.es/es/programma-101-4/

Definicion ABC. (s.f.). *Definición de USB*. Obtenido de Definicion ABC: https://www.definicionabc.com/tecnologia/usb.php

Digilent. (2014). *VGA Display Controller*. Obtenido de Digilent: https://learn.digilentinc.com/Documents/269

Digilent. (11 de Abril de 2016). *Nexys 3 Reference Manual*. Obtenido de Digilent: https://reference.digilentinc.com/_media/nexys:nexys3:nexys3_rm.pdf

Digilent. (2019). Obtenido de Digilent: https://store.digilentinc.com/

Digilent Adept 2. (2019). Obtenido de Digilent: https://store.digilentinc.com/digilent-adept-2-download-only/

Durán, M. (1 de Septiembre de 2000). *MOS 6507*. Obtenido de Museo 8 Bits: http://www.museo8bits.es/chips/6507.html

Edwards, S. A. (2007). *Apple2fpga: Reconstructing an Apple II+ on an FPGA*. Obtenido de Columbia University: http://www.cs.columbia.edu/~sedwards/apple2fpga/

ESP8266. (2019). Obtenido de Espressif: https://www.espressif.com/en/products/hardware/esp8266ex/overview

FPGA4Fun. (2019). Obtenido de Serial Interface: https://www.fpga4fun.com/SerialInterface.html

Garfield, A., Moseley, N., & Otros. (10 de Mayo de 2018). *Apple-One on FPGA*. Obtenido de GitHub: https://github.com/alangarf/apple-one

Gómez Hinestrosa, J. J. (s.f.). *Atari 2600*. Obtenido de Punto de Partida: https://puntodepartida.com/retroinformatica/maquinadeltiempo/atari2600.php

ISE Design Suite. (Febrero de 2018). Obtenido de Xilinx: https://www.xilinx.com/products/design-tools/ise-design-suite.html

jgilbert. (15 de Enero de 2008). *Dinaao, A new Apple1 emulator.* Obtenido de AppleFritter: https://www.applefritter.com/node/22434

jgilbert. (s.f.). *jgilbert*. Obtenido de Sourceforge: https://sourceforge.net/u/jggilbert/profile/

LinuxCoffee. (2019). Obtenido de Apple 1 Software Notes: https://linuxcoffee.com/apple1/software/notes.html

MacOS. (2019). Obtenido de Apple: https://www.apple.com/es/macos/what-is/

Martín, A. M. (11 de Noviembre de 2018). *Qué fue de Steve Wozniak, el genio que creó el ordenador y fundó Apple*. Obtenido de El Independiente: https://www.elindependiente.com/futuro/2017/08/10/que-fue-de-steve-wozniak-el-genio-que-creo-el-ordenador-y-fundo-apple/

Matthews, I. (15 de Febrero de 2003). *The Rise of MOS Technology & The 6502*. Obtenido de Commodore Computers: https://www.commodore.ca/commodore-history/the-rise-of-mos-technology-the-6502/

Microsoft. (2019). Obtenido de Microsoft: https://www.microsoft.com/es-es

Microsoft BASIC. (17 de Mayo de 2019). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Microsoft_BASIC

MOS Technology. (1976). *Preliminary Data Sheet. MCS6500 Microprocessors.* Norristown, PA, USA. Recuperado el 3 de 2019, de http://archive.6502.org/datasheets/mos_6500_mpu_preliminary_may_1976.pdf

MOS6502. (s.f.). Obtenido de MicroChip Wiki: https://microchip.fandom.com/wiki/MOS_6502#References

Moseley, N. (9 de Mayo de 2018). *Apple-one*. Obtenido de GitHub: https://github.com/trcwm/apple-one

Motorola 6800. (3 de Abril de 2018). Obtenido de CPU World: http://www.cpu-world.com/CPUs/6800/

Nexys-3 Spartan-6. (s.f.). Obtenido de Digilent: https://store.digilentinc.com/nexys-3-spartan-6-fpga-trainer-board-limited-time-see-nexys4-ddr/

Nexys-4 Artyx-7. (s.f.). Obtenido de Digilent: https://store.digilentinc.com/nexys-4-artix-7-fpga-trainer-board-limited-time-see-nexys4-ddr/

Pillou, J.-F. (16 de Octubre de 2008). *Conector SCART*. Obtenido de CCM: https://es.ccm.net/contents/183-conector-scart

Pillou, J.-F. (16 de Octubre de 2008). *Video Compuesto*. Obtenido de CCM: https://es.ccm.net/contents/737-video-compuesto

PS2 Keyboard Library. (s.f.). Obtenido de PJRC: https://www.pjrc.com/teensy/td_libs_PS2Keyboard.html

Reed, M. (s.f.). *Hunt the Wumpus*. Obtenido de TRS 80: http://www.trs-80.org/hunt-the-wumpus/

Sanchez Hidalgo, E. (29 de Junio de 2017). *Apple II*. Obtenido de About Español: https://www.aboutespanol.com/apple-ii-841657

Sanchez, A. L. (2 de Noviembre de 2016). *¿Que es un cable VGA?* Obtenido de About Español: https://www.aboutespanol.com/que-es-un-cable-vga-4006750

Spartan 3E 1600. (s.f.). Obtenido de Digilient: https://store.digilentinc.com/spartan-3e-1600-development-board-retired/

Spartan 3E Starter Kit. (s.f.). Obtenido de Digilent: https://store.digilentinc.com/spartan-3e-starter-board-limited-time/

Steve Jobs. (2019). Obtenido de Biografías y Vidas: https://www.biografiasyvidas.com/biografia/j/jobs.htm

Streams, D. (20 de Noviembre de 2016). *La Tecnología FPGA: futuro y beneficios principales*. Obtenido de David Streams: https://www.davidstreams.com/2016/11/20/la-tecnologia-fpga-futuro-y-beneficios-principales/

Super Famicon/NES. (s.f.). Obtenido de CyberiaPC: http://www.cyberiapc.com/vgg/nintendo_nes.htm

Terasic, I. (s.f.). *Altera DE0 Board*. Obtenido de Intel: https://www.intel.com/content/www/us/en/programmable/solutions/partners/partner-profile/terasic-inc-/board/altera-de0-board.html

Teraterm. (2019). Obtenido de Teraterm: https://ttssh2.osdn.jp/index.html.en

The Apple II. (s.f.). Obtenido de Apple][History: https://apple2history.org/history/ah03/

The Apple-1. (s.f.). Obtenido de Apple][History: https://apple2history.org/history/ah02/

The Woz Monitor. (16 de Septiembre de 2018). Obtenido de SB Projects: https://www.sbprojects.net/projects/apple1/wozmon.php

UART Protocol Validation Service. (s.f.). Obtenido de Soliton Technologies: https://www.solitontech.com/uart-protocol-validation-service/

van de Loo, A. (3 de Octubre de 2007). *Remembering Hara Ra (Gregory Yob)*. Obtenido de Cryonet: http://www.cryonet.org/cgi-bin/dsp.cgi?msg=29879

Velasco, J. J. (4 de Enero de 2012). *Historia de la Tecnología: Commodore 64*. Obtenido de Hipertextual: https://hipertextual.com/2012/01/historia-de-la-tecnologia-commodore-64

Velasco, J. J. (7 de Marzo de 2013). *Historia de la Tecnología: Commodore PET*. Obtenido de Hipertextual: https://hipertextual.com/2013/03/commodore-pet-historia

Verilog. (2012). *Verilog Resources*. Obtenido de Verilog: http://www.verilog.com/

Vilches, I. (26 de Septiembre de 2018). *Apple I: el primer ordenador Apple se subasta por 375.000 dólares*. Obtenido de Expansion: http://www.expansion.com/fueradeserie/tecno/2018/09/26/5ba0da7022601dc2268b4595.html

Vivado Design Suite. (2019). Obtenido de Xilinx: https://www.xilinx.com/products/design-tools/vivado.html

Weyhrich, S. (30 de Julio de 2003). *Lenguages - Integer BASIC* . Obtenido de Apple II History Archive: https://web.archive.org/web/20071102034313/http://apple2history.org/history/ah16.html

What is an FPGA? (2019). Obtenido de Xilinx Inc.: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

Windows. (2019). Obtenido de Microsoft: https://www.microsoft.com/es-es/windows

Wozniak, S. (2019). *About*. Obtenido de Woz.org: http://www.woz.org/about/

Xilinx. (19 de 9 de 2018). *Guide to Installing and Running ISE 10.1 or 14.7 on a Windows 8.1 or Windows 10 machine*. Obtenido de XIlinx: https://www.xilinx.com/support/answers/62380.html

Xilinx. (2019). Obtenido de https://www.xilinx.com/

YouTube. (2019). Obtenido de YouTube: https://www.youtube.com/

Zybo Zynq-7000. (s.f.). Obtenido de Digilent: https://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/

Anexo I. Integer BASIC

En este anexo se pretende explicar de la manera más concisa el uso de Integer BASIC, en qué consiste y todas sus características, para que la programación resulte más sencilla. Todas las instrucciones y sentencias que se muestran en este anexo se han probado en el proyecto, por tanto, son funcionales.

1. Introducción

BASIC es un lenguaje de programación creado en Dartmouth College, Hanover, Nuevo Hampshire, en Estados Unidos entre 1963 y 1964. Su objetivo era ofrecer una forma sencilla de interactuar con el ordenador sin utilizar lenguaje ensamblador. Dado su gran facilidad y versatilidad, este fue avanzando hasta llegar a ser Visual Basic de Microsoft, pero perdiendo muchos matices en su evolución. Las siglas BASIC vienen de "**B**eginner's **A**ll-Purpose **S**ymbolic **I**nstruction **C**ode" y como dice su nombre, es un lenguaje basado en códigos de instrucción para principiantes.

Pero este lenguaje de programación le parecía demasiado complejo y pesado a Steve Wozniak, y desarrolló, 13 años más tarde Integer BASIC, un intérprete de BASIC para las maquinas Apple que trabajaba únicamente con enteros, reduciendo su complejidad. Al utilizar sólo enteros, Integer BASIC redondea hacia el entero menor, en caso de encontrarse decimales en las operaciones. También es necesario indicar que el lenguaje BASIC, sólo admite caracteres en mayúsculas, establecidas en el código ASCII estándar (128 caracteres).

Este anexo se ha realizado con la ayuda de 2 documentos, "Programación Avanzada en BASIC" (Bishop, 1982) y "Apple 1 Basic Manual" (Apple Computers, 1976).

Un comienzo sería indicar cómo se entra al interprete BASIC del Apple I, normalmente se tiene que cargar desde casete en la dirección E000, pero en el caso de esta emulación, este paso es innecesario. Es suficiente con escribir tras el prompt clásico del Apple I, "/" el comando ***E000R*** para iniciar Integer BASIC. También ofrece una forma de acceder al anterior código, almacenado en la sesión previa, sin cargar nuevos programas en caso de un *reseteo* accidental. Para acceder a la anterior sesión se debe ejecutar la instrucción ***E2B3R***. Ambas formas han sido probadas y son completamente funcionales. Del mismo modo, para la ejecución de programas escritos en BASIC desde el casete, se realiza con ***E2B3R***.

2. Constantes

Como concepto inicial es necesario conocer que en Integer BASIC, sólo se admiten constantes enteras, cualquier número con decimales es ignorado y asignado el menor valor entero disponible. Para representar valores constantes es suficiente con escribirlos junto a la sentencia deseada.

Existen 2 tipos de constantes:

- **Numéricas:** Estas se escriben directamente, sin comillas.

  p.e. ***PRINT*** *67* , devuelve *67*

  En este caso, la sentencia "***PRINT***" se comentará en otro punto, pero sirve para mostrar por pantalla variables y constantes. El número "*67*"es una constante, su valor no puede cambiar.

- **Alfanuméricas:** Este tipo de identificador, utiliza tanto números como letras, se deben escribir con comillas en su comienzo y final, admitiendo espacios.

  p.e. ***PRINT*** *"TEST LINE"* , devuelve *TEST LINE*

Al igual que en las constantes numéricas, el valor "*TEST LINE"* no puede variar, esto permite mostrar textos y números por pantalla.

3. Variables

Las variables tienen una construcción particular haciéndolas diferentes entre sí y siendo también muy limitadas. Se construyen de la siguiente manera:

- **Variables numéricas:**
  Letra + (numero)?
  De esta manera se puede distinguir las variables numéricas de las constantes alfanuméricas. Siendo el número opcional, pero permite distinguir más variables.
- **Array de variables numéricas:**
  Letra + (número)?
  Siempre y cuando, esté la variable dimensionada se pueden realizar arrays, pero no se distinguen de las variables numéricas en el nombre.
- **Variables alfanuméricas:**
  Letra + (número)? + $
  La diferencia entre las variables numéricas y estas, es el símbolo "***$***" al finalizar. En BASIC esta variable es capaz de contener largas cadenas de caracteres, pero en el caso de Integer BASIC sólo puede contener 1 carácter. Esto se puede resolver usando la sentencia "***DIM***", explicada más adelante, para ofrecer un tamaño mayor y limitado en la variable alfanumérica.

Uso de las variables, la instrucción "***LET***" utilizada en el ejemplo (Tabla 12) también se explicará más adelante, pero es una forma de instanciar variables.

```
> LET X = 25
> LET A8$ = "B"
> LET C6 = 56
> PRINT  X,A8$,C6

Devuelve: 25 B 56
```

Tabla 12: Ejemplo Variables BASIC

4. Expresiones

Las expresiones, al igual que en las matemáticas, en BASIC son combinaciones de variables con operadores para obtener un resultado. Como se está detallando una versión "simplificada" de BASIC, Integer BASIC, hay que indicar que algunos de estos operadores no están disponibles, por tanto, se va a indicar los operadores disponibles. Además de expresiones aritméticas, Integer BASIC ofrece expresiones relacionales empleando los operadores que, a continuación, se detallan:

Operadores:

Los dos primeros son operadores "unarios" (solo un operando). El resto son todos binarios, de ellos, los cinco primeros, son aritméticos y los restantes son operadores lógicos.

- **Negar una expresión:** El primer operador es el signo menos, "-", este se utiliza para obtener el valor contrario de una variable numérica o expresión.
  p.e. La sentencia "**LET** W = -X", asigna a la variable 'W', el valor contrario a 'X'

- **Operador "NOT":** Este operador es lógico, asigna el valor '0' si el resultado de la expresión es distinto a 0, y asigna el valor '1' si esta es '0'.
  p.e. "**PRINT NOT** W", devolverá 0, si W era distinto a '0' o '1' en caso contrario.
- **Multiplicación:** El símbolo para el operador aritmético de la multiplicación es "*****", y como es de sentido común, se utiliza para multiplicar.
- **División:** El símbolo que se utiliza para la división aritmética es "**/**", y te devuelve el entero menor de hacer la división entre dos expresiones.
- **Suma:** El símbolo utilizado para el operador de la suma es "**+**", devolviendo el valor de la suma de dos expresiones.
- **Resta:** De manera análoga a la suma, su símbolo es "**-**", y a diferencia de la negación de una expresión, este requiere una expresión a cada lado, aunque admite el valor '0', realizando el mismo cometido.
- **Módulo:** Por último, existe el operador modulo, utilizado para obtener el residuo de la división de dos expresiones, utilizando para ello la instrucción "**MOD**".
- **Igual:** Para realizar la operación relacional de igualdad se utiliza "**==**", comparando ambas expresiones, devolviendo un '1' si son iguales y '0' en caso contrario. Esta opción admite la comparación de una secuencia de caracteres como de caracteres numéricos, y se suele utilizar en las sentencias **IF** … **THEN** …
- **Mayor que:** El símbolo utilizado es "**>**" y devuelve '1' si la expresión izquierda es mayor que la expresión al lado derecho.
- **Menor que:** El símbolo utilizado es "**<**" y devuelve '1' si la expresión izquierda es menor que la expresión al lado derecho.
- **Mayor o igual que:** Es una combinación del anterior operador "Mayor que" con el operador "Igual". Utilizando el símbolo "**>=**", y devuelve un '1' si el operador del lado izquierdo es mayor o igual que el puesto en el lado derecho.
- **Menor o igual que:** Es una combinación del anterior operador "Menor que" con el operador "Igual". Utilizando el símbolo "**<=**", y devuelve un '1' si el operador del lado izquierdo es menor o igual que el puesto en el lado derecho.
- **Distinto:** Este operador es una combinación de los previos operadores "Mayor que" y "Menor que", utilizando dos símbolos para ello, "**<>**" y "**#**", cualquiera de los dos cumple su cometido, devolviendo '1' si los operandos no son iguales.
- **Operador "AND":** Se utiliza para realizar la operación "AND" entre dos expresiones, y para juntar dos condiciones dentro de un **IF … THEN …**. Devuelve '1' si ambas expresiones se cumplen.
- **Operador "OR":** De manera similar a la del operador "**AND"**, con la diferencia que realiza la operación lógica OR, comparando ambas expresiones, y devolviendo '0' si ninguna de las dos se cumple. Se suele utilizar en las sentencias **IF … THEN …**.

5. Funciones

El siguiente concepto que se quiere explicar y es interesante son las funciones. Estas siguen estando relacionadas con las Expresiones, pero tienen un uso ligeramente distinto, por tanto, se ha considerado no incluirlas en el apartado previo. Hay 5 funciones que son importantes y ofrecen una funcionalidad adicional a la normal.

- **ABS(**expr**):** Esta función de vuelve el valor absoluto de la expresión deseada, esta función devuelve un valor numérico, siempre y cuando la expresión devuelva un resultado numérico, siendo esto lo más común.
  p.e. "**LET** G = **ABS(**-24**)**", devolviendo por pantalla el valor 24 que es el valor absoluto.

- **SGN(**expr**):** Esta función ahorra algunas líneas del código, actuando como la función "signo del resultado" disponible en otros lenguajes.
- **PEEK(**expr**):** Esta función tiene su contraria en "**POKE**" que se detallará más adelante, en el apartado de Sentencias. Su finalidad es devolver el valor, en decimal, almacenado en la dirección en memoria RAM que devuelve la expresión. La dirección también se debe escribir en decimal, a pesar de que la memoria RAM se direcciona en hexadecimal. Cómo la memoria RAM contiene palabras de 8 bits, el máximo valor almacenado es 255.

  p.e. "**PRINT PEEK**(768)", en la posición 768 o 0x300, se encuentra en este caso el valor 0x4C o 76 en decimal. La ejecución devolverá el valor 76.
- **RND(**expr**):** Esta función devuelve números aleatorios desde 0 hasta el valor asignado con la expresión, siendo este no incluido. Dado que su ejecución no es muy relevante se va a omitir.
- **LEN(**var$**):** Es la única función, excepto algunas sentencias o comandos, que sólo acepta variables alfanuméricas, devuelve la longitud actual de la variable, siempre que esté dentro de la dimensión o limite asignado.

  p.e. "**PRINT LEN**(A$)", en el caso que la variable A$ tuviese dimensión 4 y contenga "WW5". La función devolverá el valor 3, siendo este la longitud de la variable.

Algunas funciones adicionales que están disponibles en Integer BASIC, y no aparecen aquí, se debe a que existen funciones que pueden actuar como comandos, sin necesidad de incluirse en el código, o depender de otro comando o sentencia. Más adelante se detallarán estos comandos y sentencias que se comportan como funciones también.

6. Arrays

El siguiente elemento del lenguaje Integer BASIC son los arrays, son secuencias de números o caracteres, que se pueden invocar usando las posiciones del array. A diferencia de otros lenguajes conocidos actualmente, el primer elemento del array se posiciona en la posición '1', cualquier acceso a la posición '0' o a posiciones negativas devolverá un error de rango del array.

Un fragmento importante, es la necesidad de definir previamente la dimensión del array, teniendo estos, como ya se comentó, el mismo identificador que una variable. En el caso de las variables alfanuméricas, aquellas que sirven para almacenar "strings", se consideran arrays, indicando la posición de cada carácter con su posición en el array.

Sentencia DIM:

Una vez explicada esta diferencia entre arrays en Integer BASIC y otros lenguajes, para crear o instanciar un array, se utiliza la sentencia "**DIM**", siendo un acrónimo de "DIMension" para definir la dimensión del array. Para la utilización de la sentencia "**DIM**", esta viene seguida por la variable a convertir en array y la dimensión de esta.

p.e. "**DIM** A**(**5**)**", esta sentencia asigna, y crea en caso de no estar ya definida previamente, a la variable A una dimensión de 5, como la construcción de la variable es numérica (no viene seguida por "$" no es alfanumérica) creará un array con 5 valores numéricos entre -32767 y 32767.

En caso de exceder el valor asignado, devuelve un error de rango excedido. No existe limite al dimensionar el array, y en caso de llegar al límite de la memoria, se informará con un error. En el caso de las variables alfanuméricas, la dimensión indica los caracteres que es capaz de almacenar, pero se utiliza de la misma manera. También es necesario indicar que, en el caso de las variables numéricas, estas no son inicializadas a ningún valor, por tanto, tomarán, por defecto, inicialmente el valor que

esté almacenado en la posición elegida. Esta indeterminación se resuelve inicializando correctamente las variables al definirlas, antes de ser usadas.

Strings:

Una vez indicado cómo dimensionar variables, se va a detallar cómo funcionan los Strings en Integer Basic. Un *string* es una secuencia de caracteres, letras, números y espacios (excepto comillas dobles), contenidos entre comillas dobles, es una secuencia de valores alfanuméricos. Su uso principalmente es para las sentencias **PRINT** e **INPUT**, que se van a explicar más adelante.

p.e. "**PRINT** "ESTO ES UNA PRUEBA DE TEXTO" ", devolverá por pantalla *ESTO ES UNA PRUEBA DE TEXTO*.

Pero cómo ya se ha dicho existen variables del tipo string o que puedan contener caracteres alfanuméricos, estos pueden ser asignados usando la sentencia "**LET**" o incluso "**INPUT**". Pero existen restricciones o normas que debe respetar la variable del tipo String.

- La forma de estas variables es ***(letra)+[numero]+$***, siendo el número una parte opcional de la declaración de estas.
- Usando la sentencia "**DIM**", explicada previamente, se establece su longitud máxima permitida, pudiendo tener una longitud de entre 1 hasta 255 caracteres.
- En caso de no dimensionar la variable, su longitud por defecto es '0', pero admite 1 carácter.
- Una variable string puede contener tantos caracteres como longitud máxima se ha asignado, pero si se sobrepasa, ocurre un error de desbordamiento de pila, no aceptando el valor.

Substrings:

Integer BASIC también ofrece una manera de obtener los segmentos de un string alfanumérico, esto se obtiene indicando la posición inicial y su posición final, si la tiene. Esto permite obtener una porción de una variable para comparar con una secuencia de caracteres o sólo obtener los caracteres de determinadas posiciones.

Un ejemplo de ello sería el siguiente:

"**LET** A$ = "ABCDEFG"", dimensionada hasta 8 caracteres ("**DIM** A$(8)")

Si se ejecutan las siguientes sentencias:

"**PRINT** A$(4)", se imprime *DEFG*. Al ser desde la cuarta posición en adelante hasta el final.

"**PRINT** A$(2,6)", se imprime *BCDEF*. Los caracteres desde la segunda posición hasta la sexta.

"**PRINT** A$", se imprime *ABCDEFG*. Se imprimen todos los caracteres.

"**PRINT** A$(2,2)", se imprime *B*. Se imprime sólo el carácter de la segunda posición.

Concatenación de strings:

A pesar de no ser la traducción exacta de "Destingation strings", siendo este una errata para "Destination strings". Se resume en la posibilidad de concatenar strings siempre y cuando sea lo suficientemente largo para contenerlo. En el string destino, situado a la izquierda del "=" se le concatena desde la posición indicada o substring, el string o substring de la posición derecha del "=".

p.e. "**A$(3) = B$**", concatenará el string de la variable B$ en la variable A$ desde la posición 3 de la misma, si es suficientemente grande para contenerlo.

Esta concatenación también tiene algunas restricciones o reglas:

- La variable destino (izquierda del "=") debe ser suficientemente larga para contener la variable origen o fuente (derecha del "=").
- Si no se establecen substrings ("**A$ = B$**"), se sobrescribe el contenido de la variable B$, en este caso, en la variable A$. Si la variable origen es menor al tamaño de la variable A$, se le asignan espacios en blanco hasta llenar la variable.
- Si se establece un substring en el destino, se empieza a cargar desde el punto indicado. ("**A$(5) = B$**") escribirá desde la posición 5 de la variable A$, el contenido de la variable B$.
- Integer BASIC no contempla la concatenación en un substring "especifico"[1] como destino, es decir, no es posible cargar en la variable "**A$(2,5)**", que sería el substring desde la posición 2 hasta la 5.
- En la variable origen sí pueden aparecer estos substrings "específicos" [2].

Función LEN:

A pesar de haber especificado ligeramente la funcionalidad de esta función previamente, es también útil en el caso de querer concatenar a continuación dos variables. Usando "**B$(LEN(B$)+1) = A$**", se puede concatenar dos variables una a continuación de la otra, si la primera lo permite.

Sentencia IF usando Strings:

Integer BASIC permite la utilización de strings y substrings en la comparación de la sentencia **IF,** de esta manera se puede comprar si dos substrings o strings son iguales o distintos, siendo las únicas opciones válidas. Se matiza que si una de los dos strings tiene una longitud distinta pero tiene los caracteres en la misma posición, esta comparación se considera errónea devolviendo que son distintas.

7. Comandos

En Integer BASIC existen dos tipos de instrucciones, los comandos, que se detallaran en este apartado, y las sentencias, que se detallaran en el siguiente.

Los comandos son instrucciones que no pueden ser utilizadas en el código del programa, sin embargo, tal y como se indicará posteriormente, existen sentencias que pueden ser empleadas como comandos.

```
10 LET X=X+1
20 PRINT X
30 IF X<=50 THEN END
40 GOTO 10
RUN
```

Tabla 13: Programa que escribe por pantalla números del 1 al 50 y finaliza en Integer BASIC

Como se puede ver, un programa BASIC tiene la forma mostrada en la Tabla 13, con las líneas del programa numeradas y la instrucción que se desee, y que se almacena en la zona de memoria dedicada al BASIC. El comando, es la instrucción "**RUN**", que se ve al finalizar.

[1] Aclaración: Según el manual oficial de Integer BASIC (Apple Computers, 1976), se consideran como dos substrings utilizar A$(2,5)

[2] Mismo objetivo que [1]

Los comandos que ofrece BASIC (externos a los programas) son:

- **AUTO** *value1, value2*: Es de los más útiles para la programación, pone la cabecera o línea de programa de forma automática al inicio de la línea. El valor "*value1*" indica la posición inicial desde la que comenzar, si no se introduce empieza en la posición 10. El valor "*value2*" es el incremento que se le aplicará, también es opcional, aunque es necesario que exista *value1* para su utilización, por defecto se incrementa en 10. Para salir del modo *AUTO*, se utiliza *CONTROL+D*, aunque en este proyecto sólo se puede acceder a *CONTROL+D* por medio del terminal, dado que el código del teclado implementado no admite combinaciones de teclas, y mostrará únicamente la letra "D".
- **CLR:** Utilizado para resetear todas las variables, ponerlas a "0", cancelar todos los FOR pendientes o quitar cualquier dimensión, se suele utilizar antes de ejecutar RUN, en caso de declarar las constantes fuera del programa. Aunque esto no es recomendable hacer, porque se pueden perder valores y causar muchos problemas.
- **DEL** *val1, val2*: Para borrar las líneas indicadas desde la posición "*val1*" hasta "*val2*", ambos valores son inclusive. Si se ejecuta el comando, sin el segundo argumento, *"val2"*, se borra la línea numerada con *"val1"***.**
- **LIST** *val1, val2*: Se utiliza para mostrar el programa desde la línea "*val1*" hasta "*val2*", ambos valores son opcionales, si no se escribe "*val2*", mostrará desde "*val1*" hasta el final del programa, y si también se omite "*val1*", se muestra todo el programa desde la primera línea.
- **RUN** *val1*: Ejecuta el programa desde la línea "*val1*" o desde la primera línea disponible si este argumento no es especificado. Además, realiza un **CLR**, de las variables contenidas dentro.
- **SCR**: Borra todo el programa guardado, viene de SCRatch (rayar, borrar). No se guarda nada.
- **HIMEM (**expr**)**: Elimina cualquier programa del usuario almacenado en la expresión dada en la memoria de alto nivel, desde la posición 4096.
- **LOMEM (**expr**):** Realiza lo mismo que HIMEM pero en las posiciones de bajo nivel, desde 2048. En ambos HIMEM y LOMEM, "*expr*" es la posición de memoria en decimal.

8. Sentencias

Las sentencias, como se ha comentado en el apartado anterior, son instrucciones que se utilizan dentro del programa, a diferencia de los comandos que son externos al mismo. Pero existen algunas sentencias que se utilizan también cómo comandos. A continuación, se muestra un listado de sentencias del lenguaje donde, en verde, se indican las que también pueden ser usadas como comandos:

- **LET *var = expr***, también se acepta su forma explícita "***var = expr***":

  Evalúa la expresión "*expr*" y se la asigna a la variable "*var*", las variables pueden ser tanto variables numéricas como arrays o strings.
- **INPUT *ítem*:**

  Ítem puede ser de cualquier tipo de variable. La sentencia **INPUT**, puede contener varios ítems seguidos de coma, siempre y cuando se introduzca el valor correspondiente a cada uno.

  INPUT muestra el carácter "*?*", cuando requiera de una interacción con el usuario, pero se puede preceder texto antes escribiendo el texto entre comillas. En el caso de utilizar multivariables, estas pueden ser introducidos con ',' entre ellas. Pero en el caso de ser strings, deben estar separadas con "ENTER"/"↓"(símbolo de enter). Como recomendación, utilizar ENTER en cada variable soluciona errores relacionados con las variables en **INPUT.**

- **PRINT** *ítem(s)*:

  Imprime por pantalla los ítems que se han pasado como parámetros, pudiendo ser estos variables, mensajes, o constantes. Para separar varios elementos, se pueden utilizar comas ‘,’, que incluyen un espacio a continuación o punto y coma ‘;’, indicando que viene seguido. En el caso de usar ‘;’ al final de la instrucción, se fuerza a que el siguiente comando PRINT empiece a imprimir su información justo detrás de donde acabó el anterior, mientras que dejarlo sin ‘;’ implica que, el siguiente PRINT mostrará los resultados en la siguiente línea.
- **TAB (**expr**):**

  Escribe el número de espacios que se indica en la expresión *expr*.
- **FOR** *var* = *expr1* **TO** *expr2* **STEP** *expr3*
  NEXT *var:*

  En este caso las sentencias **FOR** y **NEXT** deben ir juntas. El fragmento “**STEP** *expr3*” es opcional, e indica el incremento con el cual crece “*var*” que, por defecto se usará un incremento de ‘1’. Esta sentencia realiza tantas veces como se indique en “*expr2*” y con un incremento de “*expr3*”, las instrucciones programadas entre **FOR** y **NEXT**. Se puede establecer un valor en “*expr1*” distinto a ‘1’ si se requiere otro valor.
- **IF (***expr***) THEN** *statement/line number*:

  Si el valor de “*expr*” es ‘0’, no ejecuta nada, y pasa a la siguiente línea. En caso contrario, ejecuta “statement” o salta a la línea “*line number*”. Con esta sentencia/comando se implementan las bifurcaciones de código en el programa.

  En el caso de ser utilizado como un comando, la sentencia “*statement*” debe ser otro comando.
- **GOTO** *expr*:

  Realiza un salto a la línea indicada por la expresión “*expr*”. No permitiendo volver una vez finalizada la operación solicitada.
- **GOSUB** *expr*
  RETURN:

  GOSUB y **RETURN** son sentencias que deben usarse juntas. **GOSUB** permite hacer un salto a la línea indicada por “*expr*” y **RETURN**, permite volver a la línea siguiente al **GOSUB** que se ha ejecutado recientemente. Solucionando el problema que suponía **GOTO.**
- **DIM** *var1* **(***expr1***),** *var2* **(***expr2***),** *…*:

  Utilizado para establecer una dimensión o límite a los arrays de un tamaño especificado en “*expr*”. Se pueden dimensionar tantas variables como se quiera en una misma línea.
- **REM** text:

  Es utilizado para hacer comentarios en los programas, no aparecen en la ejecución, son sólo informativos para visualizar el código. Aunque se utilice como comando, no muestra nada.
- **END:**

  Finaliza la ejecución del programa.
- **POKE** *expr1, expr2*:

  Como ya se explicó su contraparte **PEEK**, en el apartado de

  Funciones, esta sentencia se utiliza para cargar una posición indicada, el valor deseado, borrando cualquier otro valor ya almacenado. “*expr1*” es la posición en la memoria y “*expr2*” el valor, ambos en decimal.
- **CALL** *expr*:

  Permite utilizar subrutinas en lenguaje ensamblador desde BASIC, indicando la posición (en decimal) en memoria de la subrutina. Al finalizar, devuelve el control a BASIC.

Integer BASIC permite la ejecución de varias sentencias en una misma línea de código lo que viene a ser un "pipe" en lenguajes modernos, para ello, entre cada sentencia se introduce ":".

9. Errores comunes

En este último apartado, se van a comentar los errores más comunes que aparecen en Integer BASIC, debido a una inexperiencia en la programación y pueden ayudar a entender errores que pueden aparecer.

- ***** SYNTAX**:
  Error originado de una incorrecta estructura semántica, se resuelve volviendo a escribir la línea en cuestión, comprobando que está bien sintácticamente.
- ***** > 32767 ERR**:
  Error ocasionado al emplear datos de mayor rango que el que permiten almacenar los 16 bits de las variables numéricas (-32767, 32767).
- ***** END ERR:**
  Error que ocurre cuando la última instrucción no es un "**END**", el programa finaliza, pero devuelve el error.
- ***** RANGE ERR:**
  Error que indica que se ha superado el límite de la dimensión en un substring o array, o ésta es menor que 1. También ocurre al acceder a posiciones inexistentes del array
- ***** STR OVFL ERR:**
  Error que indica que se ha superado la dimensión del string. Es un error muy común la primera vez que programas en BASIC.
- ***** STRING ERR:**
  Error que indica que se ha querido ejecutar una instrucción ilegal o inexistente.
- **RETYPE LINE:**
  Indica que se ha introducido un valor no compatible con el "**INPUT**" esperado, permite volver a escribir el valor.

Otros errores menos comunes son:

- ***** > 255 ERR:**
  En el caso de utilizar memorias o tamaños de dimensión de las variables, y salirse del rango de 0 a 255.
- ***** BAD BRANCH ERR:**
  Error que aparece al hacer un salto de línea a una línea inexistente en el programa. La solución más sencilla, es comprobar a qué línea apuntaba y modificar la sentencia o añadir la línea buscada.
- ***** BAD RETURN ERR:**
  Error que surge al utilizar más "**RETURN**" que "**GOSUB**" previamente.
- ***** BAD NEXT ERR:**
  Error que aparece al utilizar un "**NEXT**" sin disponer de su respectivo "**FOR … TO … STEP …**" antes, aunque permita realizar "**FOR**" anidados.
- ***** > 8 GOSUBS ERR:**
  Error originado al utilizar más de 8 niveles de "**GOSUB**" anidados, se llega al límite de la pila, no acepta más profundidad.
- ***** > 8 FORS ERR:**
  Error originado al utilizar más de 8 niveles de "**FOR … TO … STEP …**" anidados, se llega al límite de la pila, no acepta más profundidad.

- ***** MEM FULL ERR:**
  Error que indica que la memoria se ha llenado.
- ***** TOO LONG ERR:**
  Error que indica que se han anidado demasiados paréntesis.
- ***** DIM ERR:**
  Error que ocurre cuando se pretende dimensionar un array ya dimensionado previamente.
- En el caso de superar los 128 caracteres en una línea sin pulsar ENTER, en esta aparece un "*backslash*", '\'.

Anexo II. Inicio correcto del proyecto

En este anexo se detallará de la manera más concisa cómo realizar un inicio correcto del sistema, desde su instalación, hasta las características que lo definen para utilizarlo.

1. Requisitos

En un principio es necesario conocer qué requisitos se tienen para instalar el programa, y son los siguientes:

- **Placa FPGA Nexys-3 de Digilent**:

  Esta es la placa que se ha seleccionado para el proyecto dadas sus características, pero realizando pequeños ajustes, se puede modificar para que pueda ejecutarse en otra placa distinta, estos ajustes no son relevantes en este caso.
- **Un ordenador**:

  Aunque sea obvio, para contener los programas instalados y los ficheros con los códigos del programa es necesario el disponer de un ordenador con estos instalados, cualquier configuración del ordenador es personal del usuario.
- **Código Apple_One_Nexys3**:

  Este código está disponible en el fichero comprimido "apple_one_nexys3.zip", es necesario en el caso de querer volver a cargar el programa en la placa, y contiene multitud de ficheros, siendo los más importantes, las carpetas "progs_apple" y "roms", y el fichero del proyecto "apple_one_nexys3.xise". Todos son importantes, pero con estos 3 no se podrá verificar un buen funcionamiento.
- **Programa "Xilinx ISE Design Suite 14.7"**:

  Este programa es el utilizado durante el proyecto, es la última versión estable del software "ISE Design Suite", compatible con las placas Spartan-6 entre otros. La versión utilizada es específicamente la versión de Windows 7/8/8.1 instalada en Windows 10, que con un pequeño ajuste en la configuración, este consiste en sustituir el archivo "libPortability.dll" de las rutas "..\Xilinx\14.7\ISE_DS\ISE\lib\nt64\" y "..\Xilinx\14.7\ISE_DS\common\lib\nt64\" por su versión NOSH (NOSmartHeap), que no permite iniciar ISE 14.7 en Windows 10 (Xilinx, 2018). Este programa está disponible para descargar en la propia página de Xilinx, teniendo cuidado al elegir la versión o Sistema Operativo, para evitar imprevistos. Se va a omitir su instalación, dado que no es relevante en este anexo.
- **Programa "Digilent Adept"**:

  Programa necesario para cargar el archivo ".bit" en la placa seleccionada, es gratuito y muy intuitivo.
- **Programa terminal para comunicaciones con la placa FPGA desde el PC, vía USB**:

  El programa que se ha elegido en este caso es "Tera Term", un emulador de terminal que permite la visualización de los puertos COM del ordenador. La configuración necesaria para su utilización es:
  - Tamaño de terminal: 40x25 caracteres, la misma cantidad de caracteres que en la pantalla utilizada.
  - Configuración de nueva línea: *Receive-> CR+LF* y *Transmit->CR*. Esto realiza en el caso de *CR*, un retorno de carro, para enviar el texto solicitado. Y LF realiza un salto de línea, para evitar que las líneas se sobrescriban en la pantalla.
  - Configuración de teclado: Se debe permitir la conversión del uso de *Delete* en el caso de pulsar el botón *BACKSPACE* o *DELETE* del teclado, es decir, (<-) y *Supr*.

- Configuración del puerto: Es la configuración más importante, se debe elegir el puerto donde está conectado el cable que se usa como *UART*, además, configurar la velocidad de transmisión (*Baud Rate*) a 115200, seleccionada por el autor del código, y con 8 bits de datos, sin paridad y con 1 bit de parada (8N1). Por último, se recomienda cambiar el flujo de datos a *RTS/CTS*, pero se obtiene un resultado favorable usando un flujo de datos *Xon/Xoff*.

Con esto debería estar configurado Tera Term, en cualquier caso, esta configuración no cambia en gran parte si se elige otro software para emular el terminal.

- **Pantalla y teclado:** Estos son dos elementos opcionales, no son necesarios para un correcto funcionamiento, pero su existencia facilita la utilización del Apple I en la FPGA.

2. Preparativos

Antes de conectar la placa, es conveniente verificar algunos aspectos relevantes del código, y ciertas explicaciones.

1. Se debe verificar que se dispone de todos los ficheros necesarios.

   Algunas de las carpetas disponibles en la siguiente ilustración (Ilustración 45) no son necesarios y pueden aparecer otros distintos, pero en concreto los ".v" son muy importantes que se dispongan de todos.

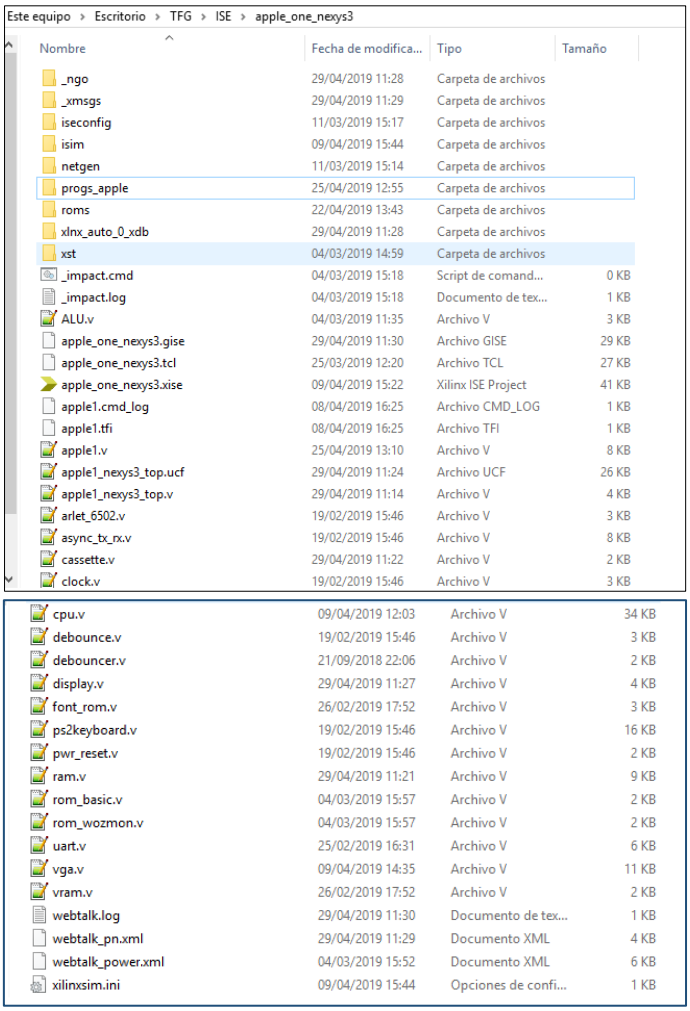


Ilustración 45: Ficheros necesarios para el funcionamiento

Una vez verificado que se dispone de todos los ficheros de la anterior ilustración (Ilustración 45), se puede pasar al siguiente paso.

2. Verificar cualquier cambio realizado en el proyecto.

   Una vez, abierta la mencionada *suite* de diseño, en caso de haber modificado algún parámetro en los ficheros, “apple_one_nexys3.v”, “cassette.v”, “display.v” y/o “ram.v”, indicados en la captura siguiente (Ilustración 46), es recomendable revisar su correcta sintaxis y que los valores introducidos son los deseados. Esto se deberá realizar siempre que se cargue un nuevo programa [0].

Ilustración 46: Ficheros a revisar en rojo

3. Guardar y ejecutar la síntesis.

   Una vez verificada la correcta sintaxis del código, se debe guardar todo y ejecutar la síntesis e implementación dentro de “ISE Design Suite”. Con el proyecto abierto, se puede buscar por la pestañas en fichero raíz “apple_one_nexys3.v”, o una vez marcado este fichero, doble clic en el proceso “Generate Programming File” (Ilustración 47), para realizar las mencionadas fases de diseño, además del mapeo y todas las operaciones necesarias. Creará el fichero “apple_one_nexys3_top.bit” con la programación del Apple I para la placa.

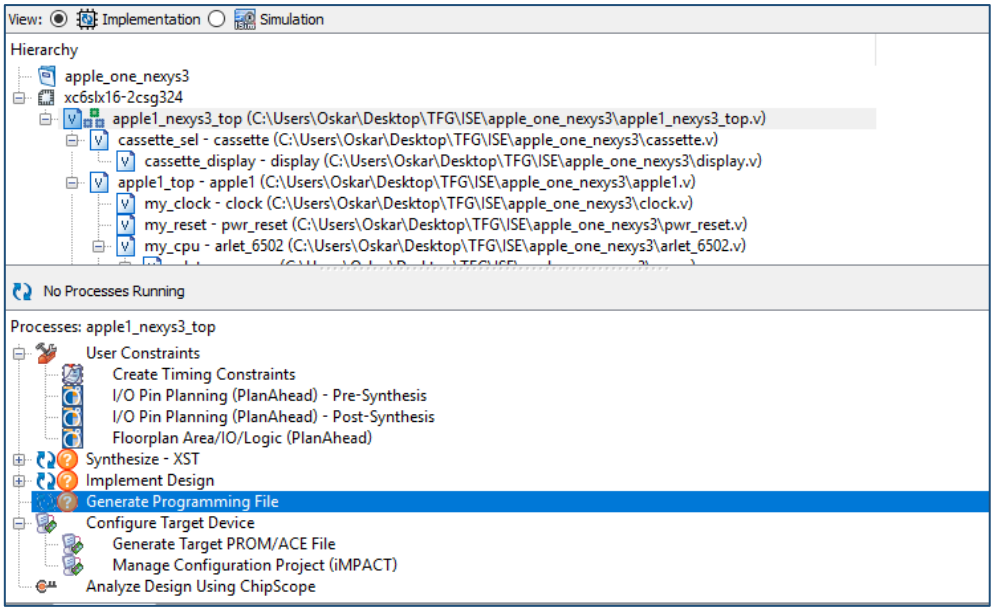


Ilustración 47: Procesos disponibles

Pueden aparecer ciertos avisos, donde una gran parte indican un error en la inicialización de los ficheros o variables sin inicializar, pero que no causan ningún error grave. También aparecerá la advertencia de que existen ficheros que superan los 9 KB de tamaño, pero no es un error pues lo soluciona automáticamente el programa. Se puede ver el estado final en la siguiente captura (Ilustración 48).

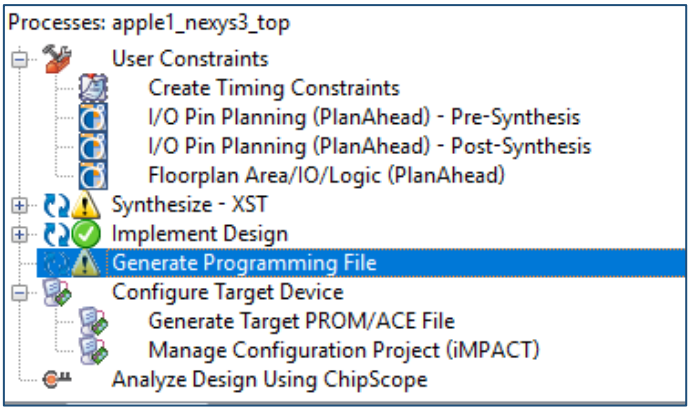


Ilustración 48: Finalización de la implementación

4. Instalación en la placa.

   Con el programa “Digilent Adept” abierto y la placa conectada al ordenador, se selecciona el fichero “.bit” deseado, que, con la ejecución de la fase anterior, se ha guardado en el directorio del proyecto, por lo que poniendo su ruta es suficiente. Se selecciona el fichero. Y se realiza clic en el botón “Program”, al finalizar debe obtener un mensaje similar al mostrado en la captura (Ilustración 49).

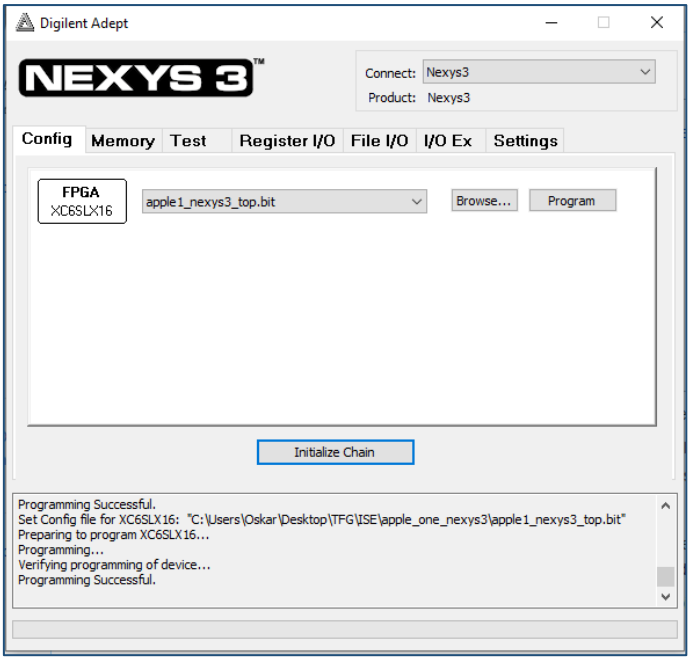


Ilustración 49: Digilent Adept con el programa instalado

Una vez, obtenido el mensaje "Programming Successful", el programa se ha cargado y se tiene la placa lista para su funcionamiento como un Apple I.

3. Placa FPGA

Se van a mostrar las funcionalidades de la placa y la utilidad de cada elemento de entra y salida. Se numeran, las partes a comentar posteriormente en la siguiente ilustración (Ilustración 50).

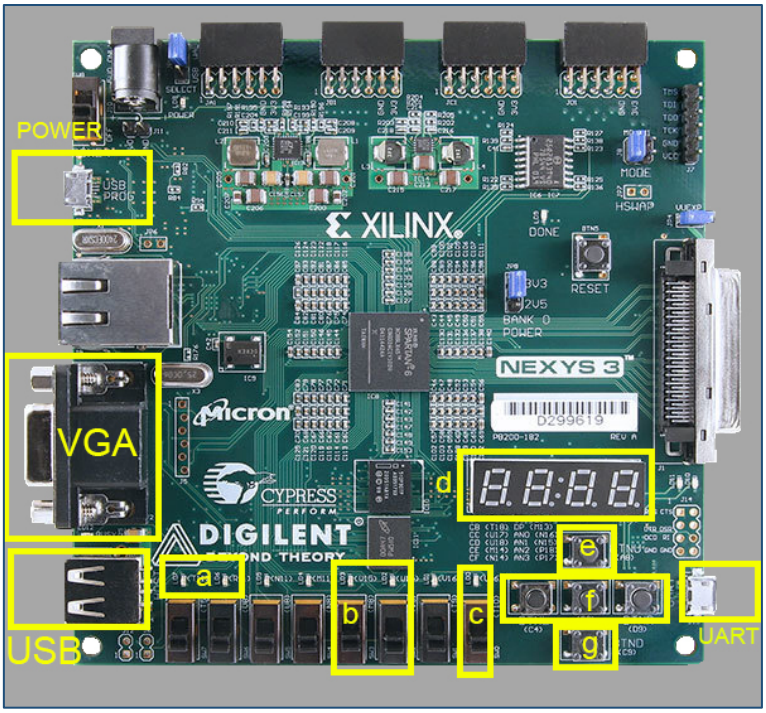


Ilustración 50: Imagen de la placa Nexys-3

En la ilustración anterior (Ilustración 50), se pueden ver 7 áreas indicadas de ‘a’ a ‘g’, siendo estos elementos importantes para el funcionamiento. Su utilidad es la siguiente:

- Área ‘a’:

  Dos *LEDs* que indican respectivamente, que el programa se ha empezado a cargar y ha finalizado de cargarse. Al ser un proceso muy rápido el utilizado para la copia de los ficheros, suelen encenderse simultáneamente.

- Área ‘b’:

  Conjunto de 2 interruptores y 2 *LEDs*, que se emplean para indicar al usuario y al sistema a qué bloque de memoria RAM se quiere acceder para ejecutar sus programas. Se utiliza en binario, por tanto: ‘00’ es el bloque 0, ‘01’ el bloque 1, ‘10’ el bloque 2 y ‘11’ el bloque 3. Al disponer de 23 programas, estos se dividen en 3 bloques de 10 aplicaciones, pues este es el número máximo posible de programas a mostrar en el *display* (d), junto con un breve título. El bloque 3 de memoria está vacío, por tanto, devuelve “8888” en el *display*.

- Área ‘c’:

  Conjunto de un *switch* y un *LED*, para permitir y mostrar si se quiere utilizar la entrada por UART o por Teclado, de manera que si el interruptor está a ‘0’, led apagado, se tendrá que usar la UART, terminal, como entrada, mientras que si está a ‘1’, por tanto, led encendido, se usará como entrada de información a la placa el teclado.

- Área ‘d’:

  Es el *display* que indica el programa al cual apunta un registro interno del sistema, varía de 0 a 9, cada uno con un pequeño título del programa, y en caso necesario, qué programa se va a cargar en la memoria del emulador para ejecutar. Si el titulo finaliza en ‘1’ indica que se carga antes de la memoria accesible (0x280/640) y en el caso de finalizar en ‘2’, en la memoria es accesible. También estos programas, indican si se inicia con BASIC o no, con la dirección donde comienza.

- Área ‘e’:

  Este botón se utiliza para realizar un *Reset* al Apple I, y muestra “\” *prompt* del sistema, en la pantalla y/o en el terminal, indicando que se acaba de iniciar el sistema. Hay que tener en cuenta que es necesario limpiar la pantalla y después reiniciar el sistema, para que muestre una pantalla en blanco. Si no, se quedan los residuos de anteriores ejecuciones, siendo este un funcionamiento que replica el uso del Apple I.

- Área ‘f’:

  En esta área se encuentran 3 botones, se han colocado en fila para ser más intuitivo, pero estar cerca del botón de *Reset* de la maquina o el de limpiar la pantalla puede suponer algún inconveniente si no se pulsa el botón correcto para realizar la acción deseada.

  Estos botones son los controles del puntero interno del programa para seleccionar el programa de Apple I que se quiere utilizar. De izquierda a derecha, sus usos son: “Programa Anterior”, “Aceptar/Cargar” y “Programa Siguiente”. “Programa Anterior” y “Programa Siguiente”, según indica su denominación, sirven, pasar a la anterior o siguiente aplicación, siendo suficiente un leve toque al botón. En el caso de “Aceptar/Cargar”, se utiliza para realizar la carga en memoria del programa apuntado y autor del este proyecto recomienda

mantener durante unos segundos pulsado el botón para garantizar que se ha copiado correctamente y no queden variables sin asignar.

- Área 'g':

  Por último, esta área contiene un botón cuya utilidad es la de limpiar la pantalla. Es recomendable pulsar este botón si se quiere resetear el sistema, para que deje una pantalla de inicio vacía y limpia.

Estos son los elementos que se utilizan de la placa, y en el siguiente apartado, se comenzará con un uso del sistema desde 0.

4. Uso del sistema

Una vez detallados los preparativos, carga del emulador en la placa, y los elementos de la misma a emplear durante la emulación, ya se puede empezar a utilizar el sistema.

Se enciende placa usando el interruptor izquierdo superior, donde pone *Power*, y con esto, el sistema se inicia y muestra por pantalla y/o terminal, por pantalla si se ha conectado a la placa y por el terminal si se encuentra conectada la misma por *USB* al ordenador. En este caso el terminal devuelve lo mostrado en la siguiente captura (Ilustración 51).

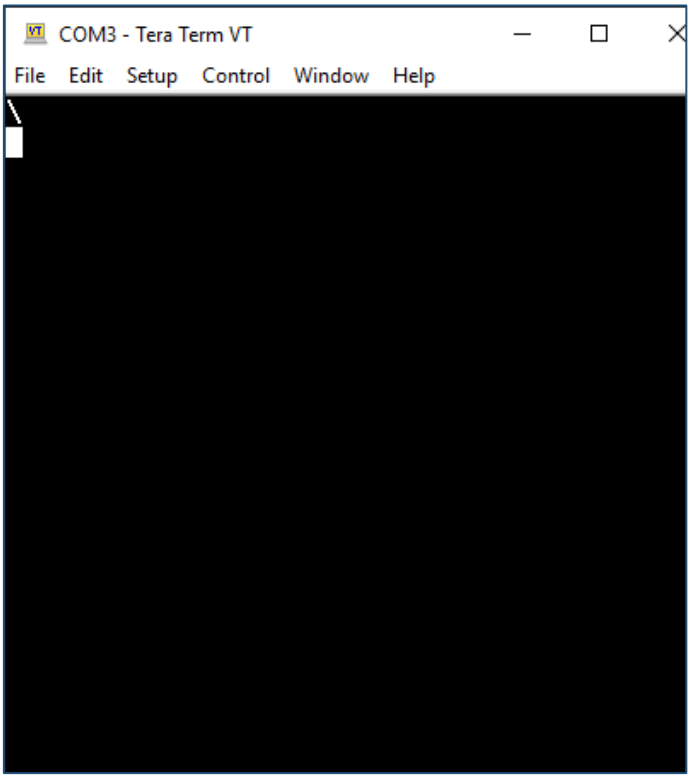


Ilustración 51: Inicio del sistema (Tera Term)

Se puede ver el indicador de que está en el sistema WozMon y espera una entrada. Al estar usando el terminal en este caso, se debe verificar que el *switch* situado en la mencionada anteriormente área 'c' de la Placa FPGA está desactivado, luego el *LED* está apagado, mientras que si se quiere usar el teclado *USB* conectado a la placa FPGA, este *switch* se deberá activar.

En este caso se tienen varias opciones, como se ha explicado en la memoria en el apartado referente al monitor, denominado WozMon, se puede consultar el contenido de la memoria RAM y, cómo el sistema se acaba de iniciar, se va a verificar que el contenido de las direcciones más

relevantes, donde suelen existir programas, está a nulo. Cómo existen algunos programas cuya primera posición tiene el valor 0x00, se consulta la siguiente ubicación del mismo para obtener el valor. En este caso: '0x4B', '0x281', '0x301', '0x801' y '0x1001'. Como se puede ver en la ilustración siguiente (Ilustración 52), estas direcciones están vacías, por tanto el sistema está a nulo.

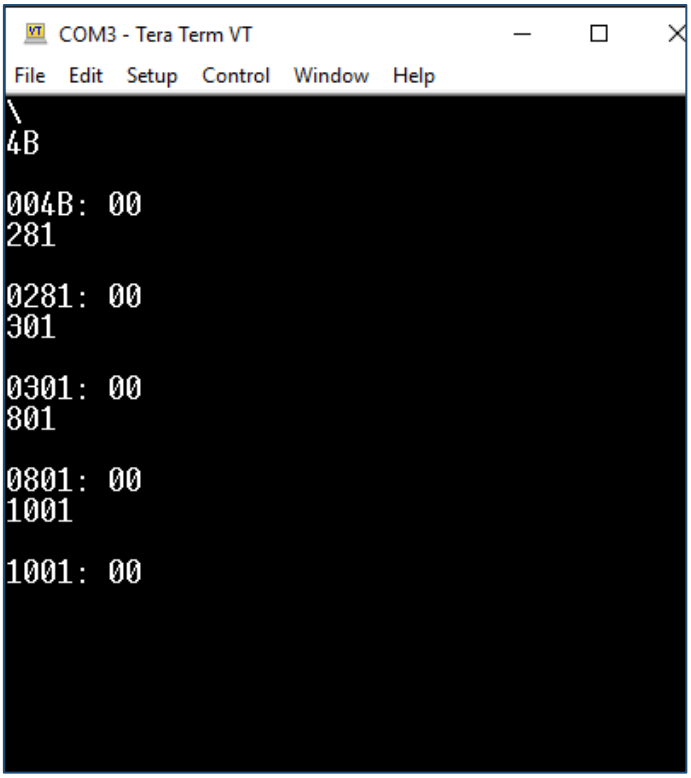


Ilustración 52: Comprobación de direcciones

WozMon también permite la escritura de un valor en una posición de memoria, por tanto, se va a escribir el programa "Hello World" (posición 2 del bloque 0 de memoria) por medio de WozMon, al ser un programa corto y sencillo. A continuación, se ejecutará el programa con '280R' y comprobará su funcionamiento correcto.

El código de "Hello World" se almacena desde la dirección 0x280 de memoria, por tanto la sentencia que se escribirá es la siguiente: "280: A2 0C BD 8B 02 20 EF FF CA D0 F7 60 8D C4 CC D2 CF D7 A0 CF CC CC C5 C8", siendo necesario escribir todos los valores, ya que si en caso de faltar un valor es posible que no se ejecute el programa. De igual forma, se recomienda no copiar y pegar los programas para evitar que el terminal se salte algunos caracteres en la transmisión de los datos. Su ejecución se ve en la siguiente captura (Ilustración 53).

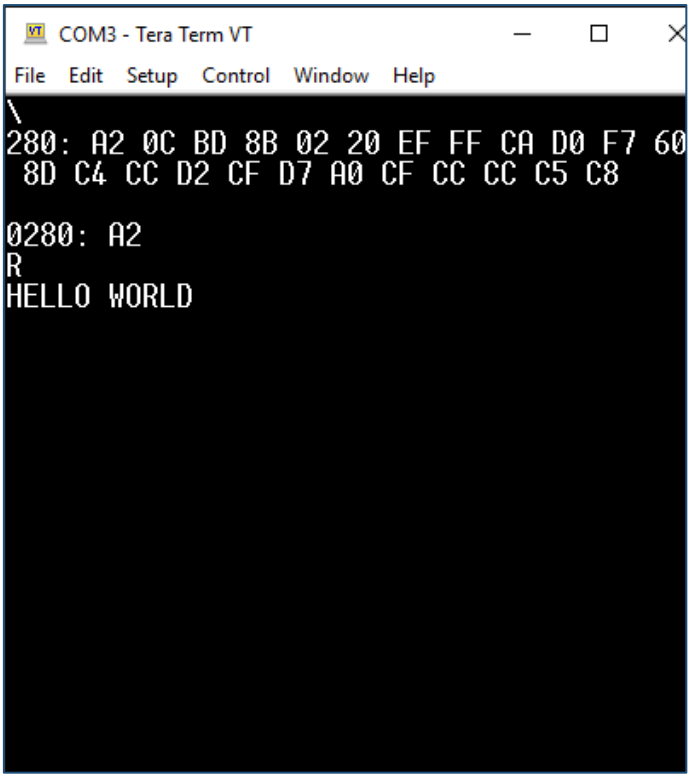


Ilustración 53: Hello World por terminal

Como no es muy cómodo escribir todo el programa en hexadecimal, y ya es molesto equivocarse en un valor y no poder recuperar la anterior línea, aunque se puede cambiar un único valor, una manera más sencilla es realizarlo mediante el cargador de programas. Con el mencionado procedimiento se puede elegir y cargar cualquier programa que contenga el fichero "\progs_apple\programs.hex", pero hay que tener cuidado, porque necesita valores específicos, tal y como se explicará con más detalle en un anexo posterior. En caso de no modificar ningún programa, la ejecución correcta de éste está garantizada. Cómo no se visualiza en el terminal la carga, es necesario fotografiar la placa, para mostrar el procedimiento.

Se selecciona el programa que se desea de una lista ya mostrada en el apartado 4.3.2 del proyecto y en un apartado siguiente de este mismo anexo [Lista con los programas disponibles]. Una vez, elegido el programa y/o programas a ejecutar, se elige el bloque de memoria donde se encuentra almacenado, con los *switch* del "área b" y el programa deseado con los botones del "área f" de la Placa FPGA. Una vez posicionado en el programa o primer programa a cargar, se pulsa el botón central del "área f" (Aceptar/Cargar). Obteniendo el resultado de la Ilustración 54. También se carga su segunda parte, como se observa en la siguiente Ilustración 55.

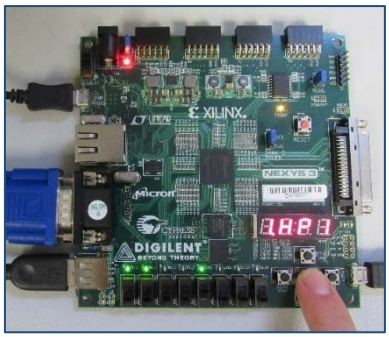


Ilustración 54: Carga del programa Wumpus(1/2)

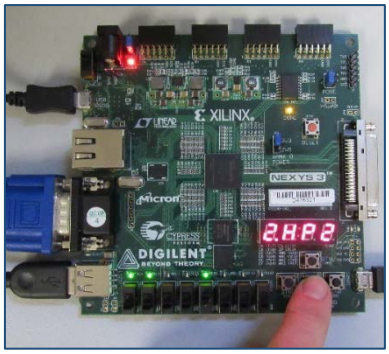


Ilustración 55: Carga del programa Wumpus (2/2)

Desde el terminal, se verifica la integridad del programa, comprobando el valor de algunas posiciones aleatorias del mismo. En este caso, se comprueba que son los correctos. También se aprecia que no se sabe desde el terminal que se han cargado los programas deseados. En la ilustración siguiente (Ilustración 56), se aprecian estos detalles.

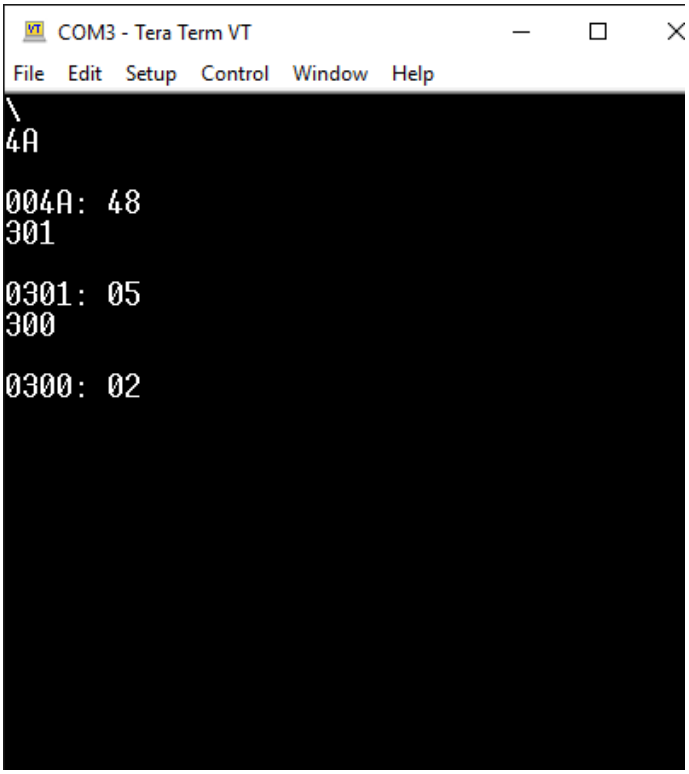


Ilustración 56:Posiciones del programa Wumpus en memoria

En este caso, el programa Wumpus, es un programa de 2 cargas o dos ficheros, y funciona desde BASIC, y que, en el apartado con la lista detallada de los programas instalados, se especifica cómo se ejecuta el programa. En este caso con 'E2B3R' se accede BASIC sin borrar su memoria. Y después con 'LIST' se vería el contenido del programa y con 'RUN' se ejecutaría. Como se puede ver en la captura siguiente (Ilustración 57), el juego llega hasta la posición 5000 en memoria, por tanto, tendrá cerca de 500 líneas de código, si se ha utilizado una separación de 10 entre líneas. También se ha ejecutado el juego y se ha probado su funcionamiento, en este caso el juego es "Hunt the Wumpus" (Reed, s.f.), un juego de aventuras en modo texto creado por Gregory Yo (van de Loo, 2007) en 1973.También se muestran otras capturas del juego en el Apple I (Ilustración 58).

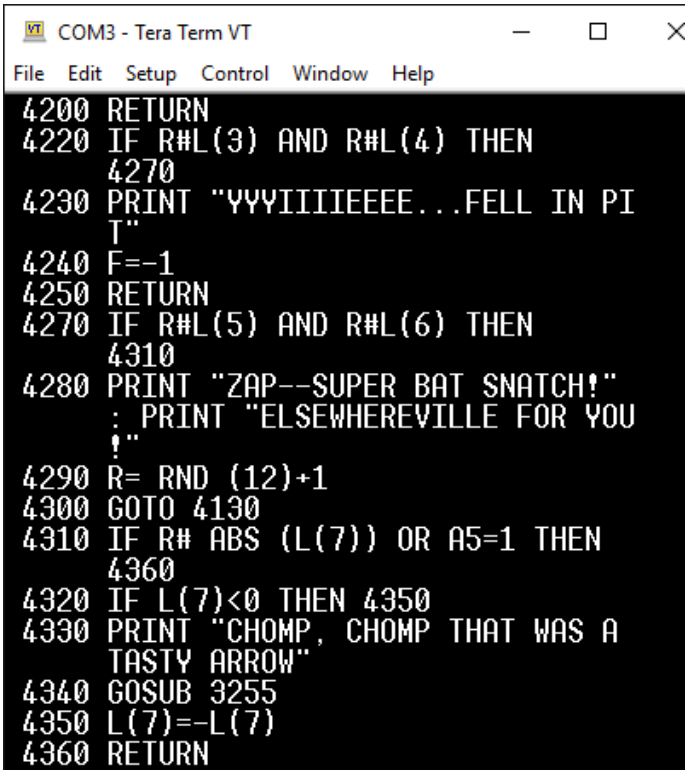
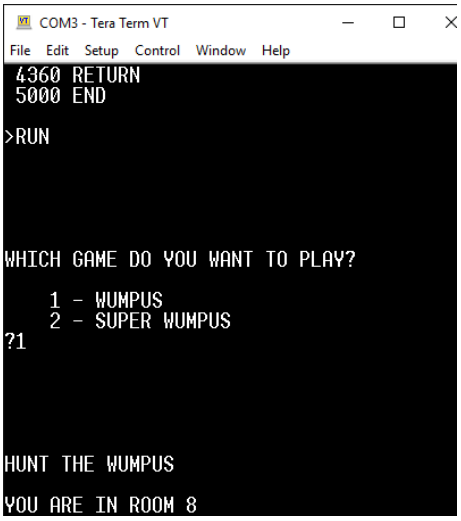


Ilustración 57: Parte del código de Wumpus, usando LIST en BASIC

a)



b)

Ilustración 58: Capturas del juego, usando RUN en BASIC. a) inicio b) más avanzado

En caso de no querer ejecutar ningún programa, desde el sistema reiniciado, se puede acceder a BASIC usando la dirección 'E000R', cualquier programa guardado se eliminará, por tanto, hay que tener cuidado con cómo entrar, recomendándose, en caso de querer usar BASIC, acceder siempre con 'E2B3R', que no borra el programa, y si no existe un programa, tiene la memoria vacía. Pero es más seguro, en caso de reiniciar la máquina sin querer, y así no perder el trabajo de codificación previo.

5. Lista con los programas disponibles

A continuación, se listan los programas con sus posiciones, que están disponibles en este proyecto para su posible ejecución, a modo de demo de funcionamiento.

Bloque y posición		Nombre	Breve descripción	Ejecutar con:
0	0	[nUL] de **Null**	Posición que limpia/vacía la memoria por completo, excepto la posición 0, que debe ser borrada manualmente.	No se ejecuta.
	1	[A30] de **Apple 30th**	Programa que ofrece una demo grafica que Apple creo para el Apple I con motivo de la celebración del 30 aniversario de la compañía. Muestra en formato ASCII a los fundadores de Apple, y sus productos más famosos hasta 2006.	Se guarda desde la dirección 280. Se ejecuta con **'280R'.**
	2	[hEL] de **Hello World**	Programa que escribe por pantalla la frase "HELLO WORLD", al ser un programa, deja ejecutarlo varias veces, pero puede bloquearse y no devolver el control a WozMon.	Se guarda desde la dirección 280. Se ejecuta con **'280R'.**
	3	[LUn] de **Lunar Lander**	Programa/juego que simula el aterrizaje de un módulo lunar. Se aplica una cantidad de combustible y el módulo sube o baja, dependiendo de la velocidad y altura. Requiriendo de 5 unidades de combustible para aterrizar.	Se guarda desde la dirección 300. Se ejecuta con **'300R'.**
	4	[HEH] de **Memory Test**	Programa que verifica y almacena valores en las primeras posiciones de la memoria RAM. Es un programa lento y sin un propósito establecido.	Se guarda desde la posición 300, y requiere de los valores '0x05' en la posición 1 y '0x10' en la posición 3. Se ejecuta con **'300R'.**
	5	[uCh] de **MicroChess**	Programa/juego que permite jugar al ajedrez, se dispone de varias formas de juego y la apertura del juego.	Se guarda desde la dirección 1000. Se ejecuta con **'1000R'.**
	6	[PAS] de **Pasart**	Programa que crea "matrices artísticas" en base al tamaño, módulo y estilo de la matriz. No se ha encontrado una descripción adecuada al resultado.	Se guarda desde la dirección 300. Se ejecuta con **'300R'.**
	7	[St1] de **Star Trek (1/2)**	Programa/juego de aventuras en modo texto de la serie homónima, donde se toma el control de una nave de la "Tropa Estelar" para derrotar a los Klingon. Se puede mostrar una matriz como mapa, disparar o realizar otras acciones.	Es un programa en BASIC. Se ejecuta con **'E2B3R'**. Y después usar **'RUN'**.
	8	[St1] de **Star Trek (2/2)**		

Bloque y posición		Nombre	Breve descripción	Ejecutar con:
	9	[Lto] de **Little Tower**	Programa/juego de aventuras en modo texto, donde explorar una torre abandona cerca de un lago.	Se guarda desde la dirección 300. Se ejecuta con **'300R'.**
1	0	[bJ1] de **BlackJack (1/2)**	Programa/juego del juego de cartas homónimo. El ordenador da varias cartas siendo el objetivo sumar 21 con las mismas, sin pasarse, pudiendo pedir más cartas si el jugador lo cree oportuno.	Es un programa en BASIC. Se ejecuta con **'E2B3R'**. Y después usar **'RUN'**.
	1	[bJ2] de **BlackJack (2/2)**		
	2	[Ck1] de **Checkers (1/2)**	Programa/juego del juego de las damas. Al igual que en el juego original, se mueven las fichas en diagonal, poniendo la posición actual y a la que se quiere mover.	Es un programa en BASIC. Se ejecuta con **'E2B3R'**. Y después usar **'RUN'**.
	3	[Ck2] de **Checkers (2/2)**		
	4	[hA1] de **Hammurabi (1/2)**	Programa/juego del juego de Hammurabi, donde se tienen que gestionar los recursos de una ciudad de manera sabia para controlar el Imperio Babilónico de Hammurabi	Es un programa en BASIC. Se ejecuta con **'E2B3R'**. Y después usar **'RUN'**.
	5	[hA2] de **Hammurabi (2/2)**		
	6	[Ht1] de **Matrix (1/2)**	Programa que simula un efecto de caracteres cayendo o escribiendo al estilo Matrix. No tiene utilidad, pero es entretenido de ver.	Es un programa en BASIC. Se ejecuta con **'E2B3R'**. Y después usar **'RUN'**.
	7	[Ht2] de **Matrix (2/2)**		
	8	[SL1] de **Slots (1/2)**	Programa/juego donde se utiliza una máquina tragaperras. Se comienza con 100 monedas, y se elige cuantas introduces en la máquina, y se muestra el resultado de la jugada.	Es un programa en BASIC. Se ejecuta con **'E2B3R'**. Y después usar **'RUN'**.
	9	[SL2] de **Slots (2/2)**		
2	0	[Hon] de **Monitor**	Programa que ofrece un monitor más útil que WozMon, pero no se ha encontrado ninguna guía de él.	Se guarda desde la dirección 800. Se ejecuta con **'F3DR'.**
	1	[HP1] de **Wumpus (1/2)**	Programa/juego "Hunt the Wumpus" de 1972. El jugador está en una mazmorra, con forma de dodecaedro, donde hay una bestia	Es un programa en BASIC. Se ejecuta

Bloque y posición		Nombre	Breve descripción	Ejecutar con:
	2	[HP2] de **Wumpus (2/2)**	llamada Wumpus, a oscuras y se pueden disparar flechas , para acabar con la bestia, estando en otra sala. El Wumpus se mueve cuando el jugador realiza cualquier acción, siendo su objetivo encontrar al jugador.	con **'E2B3R'**. Y después usar **'RUN'**.

Tabla 14:Tabla programas

En la tabla anterior (Tabla 14) se detalla qué programas contiene el Apple I realizado en el proyecto, todos estos son originales que se ejecutaban en el Apple I.

Todos los ficheros de estos programas, escritos en formato hexadecimal, están disponibles en el fichero comprimido del proyecto "apple_one_nexys3.zip". Para cualquier comprobación de la correcta posición de los valores en la memoria, es un buen recurso disponible.

Anexo III. Instalación de un nuevo programa

En este anexo se va a explicar el formato de programas que pueden crearse, así como introducir un nuevo programa dentro del proyecto del Apple I. Es necesario conocer que, al igual que en la placa original, no es posible cargar directamente programas BASIC y se utilizaba un lector de casetes con los programas escritos en hexadecimal legible para el intérprete WozMon.

1. Formato de programas

Se ha explicado previamente, en el proyecto, el formato en el cual se escribían los programas del Apple I. Cuando se comenzó a vender el mismo, en la única tienda de Palo Alto donde lo consiguieron, la placa sólo disponía del interprete Integer BASIC y WozMon.

El formato que ofrecían los casetes para los programas se basaba en el sonido que generaba cierta secuencia de bits grabadas en él. Actualmente, se han visto programas del Apple I en formato .aiff, siendo el más cercano al utilizado en la época, el cual se desconoce. Esto lleva a la pregunta de cómo es posible que existan programas en BASIC si el lector de casetes sólo transfería los sonidos y, lo cierto, es que no ha existido ningún casete con un programa BASIC, escrito como tal. Los casetes que utilizaban los programas en BASIC, constaban de dos secciones que requerían que se grabasen en posiciones concretas, una primera sección debía ser grabada desde las posiciones de memoria **'$004A'**, mientras que la segunda, dependiendo del programa, estaba en la memoria disponible para el usuario.

El formato de la memoria que disponía la maquina original se dividía en dos zonas, las direcciones desde la posición **'$0000'** de la memoria RAM hasta la posición **'$0280'** de la misma, en esta sección de la memoria es utilizada por el programa para guardar variables, y no se garantiza que tras la ejecución se mantengan esos valores. La otra sección de la memoria, que en este caso comienza en la dirección **'$0280'** y finaliza en la dirección **'$1FFF'**, es la sección no volátil de la memoria RAM, los datos almacenados, no deberían ser sobrescritos. Esta memoria puede crecer hasta la dirección **'$BFFF'** en caso de ofrecer la máxima capacidad, 48 KB de memoria RAM. Un curioso detalle, que a pesar de indicar que las direcciones **'$E000'** o **'$C000'** son accesibles, éstas se situaban antes de la dirección **'$0000'**, actuando como posiciones negativas de memoria, debido a que los dos bits más significativos, actúan para apuntar a direcciones negativas de la memoria total, y no se recomienda su modificación.

Volviendo a los programas, todos los casetes y programas que funcionan en el Apple I están escritos en el lenguaje ensamblador de WozMon que, aunque se conozca sus características, no son muy precisas, siendo un verdadero reto crear un programa funcional a partir del lenguaje de WozMon. Es cierto que existen emuladores del Apple I que aceptan lenguaje BASIC, pero en este proyecto se busca replicar el funcionamiento del Apple I y no el replicar una réplica de él.

Por tanto, para cargar un programa en memoria, actualizando la velocidad de carga y evitando los molestos casetes, se optó por crear un fichero con todos los programas funcionales para este proyecto y que se pueda cargar en cualquier momento en la memoria del sistema sin perder tiempo en su reinicio y cambio de casete.

2. Requisitos

Se van a indicar unos requisitos puntuales, que se consideran necesarios para poder cargar un programa nuevo.

- Un programa nuevo:

  Para evitar repetir un programa, sería recomendable revisar qué programas ya están cargados en memoria, evitando las repeticiones de los programas, aunque se permite. Este debe estar escrito en el lenguaje ensamblador de WozMon y es válido cualquier formato de fichero.
- Un ordenador:

  Del mismo modo que para iniciar el proyecto, se necesita un ordenador con los mismos programas, recomendándose haberse familiarizado con los mismos y el inicio del sistema.
- Ficheros del proyecto:

  Se van a modificar datos específicos del proyecto, por tanto, es conveniente tener disponibles los ficheros, así como todo el software necesario para volver a generar el fichero del proyecto con el que se debe reprogramar la FPGA.
- Un editor de texto:

  Se va a requerir un editor de texto para modificar el fichero ".hex" con los programas ya cargados. Cualquier editor es bueno, recomendándose "Notepad ++", al ser el utilizado en este caso.
- Un cuaderno o un sitio donde apuntar:

  No es estrictamente necesario, pero es útil para apuntar los valores más importantes de los ficheros.

3. Preparativos

En este caso, se va a instalar un programa que ya existe en el sistema y, por tanto, servirá como ejemplo de la carga en memoria. Será un programa en BASIC, estando formado por dos partes, aunque en el fichero descargado, los sitúa en un único fichero. El fichero descargado, viene en extensión ".txt" escrito en hexadecimal legible para WozMon, por tanto, es suficiente. Tal y como se ve en la siguiente ilustración (Ilustración 59), tiene dos direcciones donde comenzará a escribir el programa.

1- El primer paso es situar en cada línea un único valor, para que sea legible por el programa RAM.

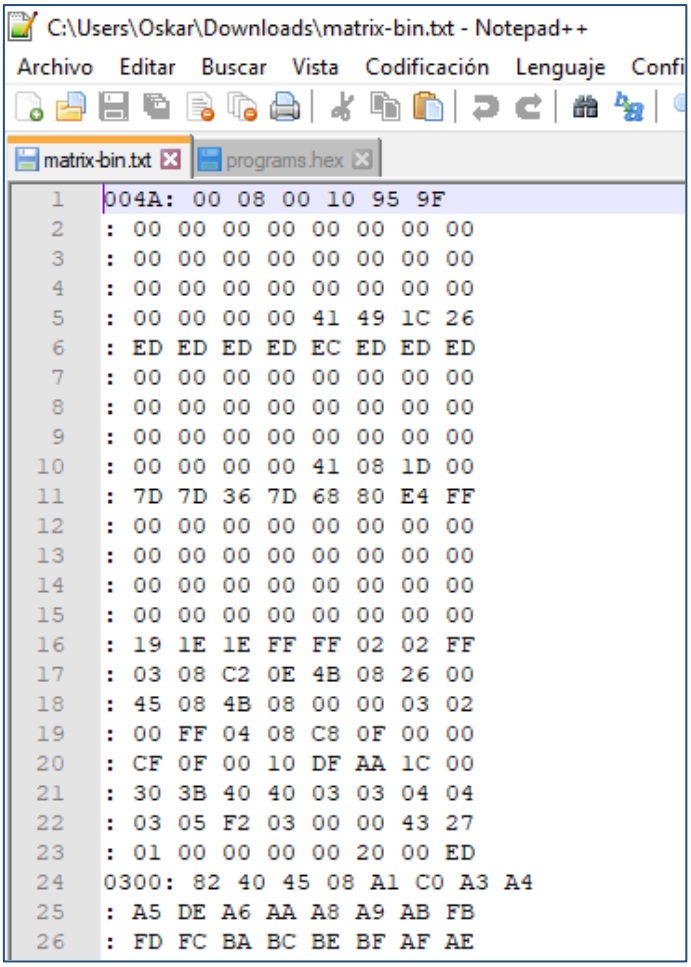
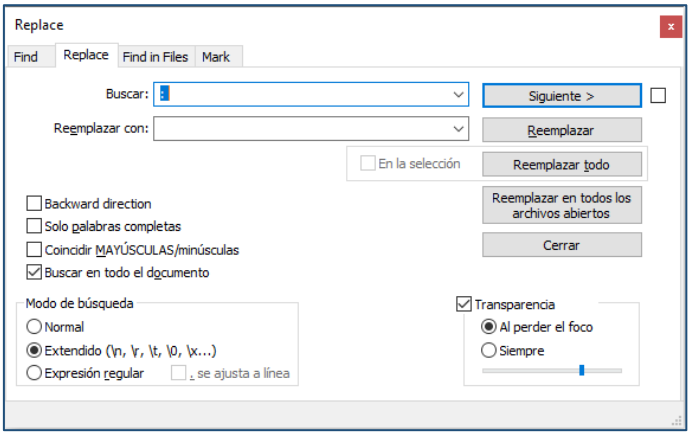


Ilustración 59: Programa Matrix en Notepad++

Para realizar el propósito de cambiar estos valores, Notepad++, al igual que otros editores de texto, ofrece la herramienta de "Buscar y Reemplazar", en Notepad, un atajo es "Ctrl+H". Una vez seleccionado esto, se tiene que reemplazar ": " por "", es decir, se reemplazará la secuencia de "dos puntos y espacio" por la secuencia vacía, evitándose así errores en la siguiente sustitución. Es recomendable separar la dirección del código, sin borrar para que sirva como indicador posteriormente. El reemplazo mostrado en la siguiente ilustración (Ilustración 60a) será suficiente para este propósito, sustituyendo después, " " por "\n", para que las líneas sólo ofrezcan 2 caracteres hexadecimales como se muestra en la ilustración posterior (Ilustración 60b).

a)



b)

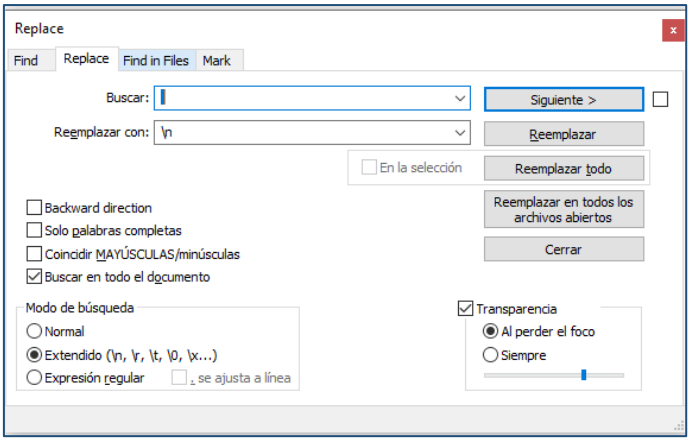


Ilustración 60: Reemplazos en el documento. a) Reemplazo ": " a "", b) Reemplazo " " a "\n"

Una vez realizado este reemplazo, el documento tendrá un aspecto similar al siguiente, visible en la siguiente ilustración (Ilustración 61).

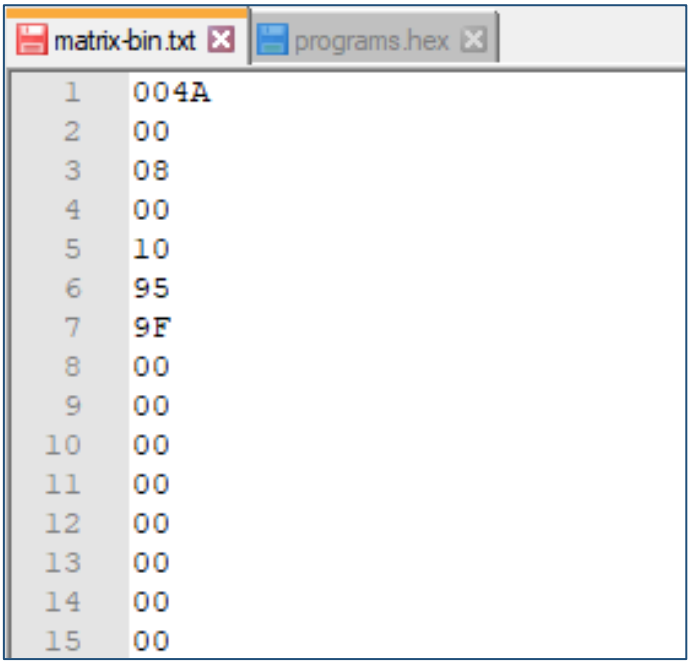


Ilustración 61: Fichero después del reemplazo

2- Apuntar las direcciones y tamaños de los documentos:

Con un lugar donde apuntar, se van a tomar nota de los valores necesarios para copiar el programa, siendo estos los siguientes:

- Se necesita saber la primera dirección útil donde empezar a escribir el nuevo programa implementado. Se busca el tamaño del actual fichero: "programs.hex", que según los ficheros dados en este proyecto es: 36979, y se le resta 1 (se recuerda que las direcciones empiezan siempre en la posición 0) para saber cuál es la mencionada primera dirección no ocupada de dicho fichero.

  El segundo segmento del código se escribirá a partir de la dirección: 37160 (la anterior + el tamaño del primer segmento de código, 182 palabras).
- Posición donde escribir el programa que, en este caso son las direcciones, **$004A** y **$0300.**

- Tamaño del programa. En este caso, esto es la cantidad de líneas necesarias en cada parte, **182** y **2048.** Es necesario realizar esto en cada sección del programa, en este caso, 2 veces.
- También es necesario saber cómo aparecerá en el *display*, acrónimo, de este nuevo programa, que será "Mt1" y ocupará el número 3 en la lista de programas disponibles. En este caso, es necesario saber el funcionamiento del *display*, cada digito está dividido en 7 segmentos y un punto, lo cual es útil dado puesto que permite escribir ***3.Mt1***. El orden que siguen estos segmentos es ***Pgfedcba,*** en cada *display*, como se puede ver en la figura siguiente (Ilustración 62). También es necesario saber qué es lo que se quiere escribir, en este caso, al no disponer de M o W, se utilizará la forma H, que actúa como ambos. El resto de los valores es sencillo de crear, obteniendo el resultado mostrado en la siguiente ilustración (Ilustración 63).

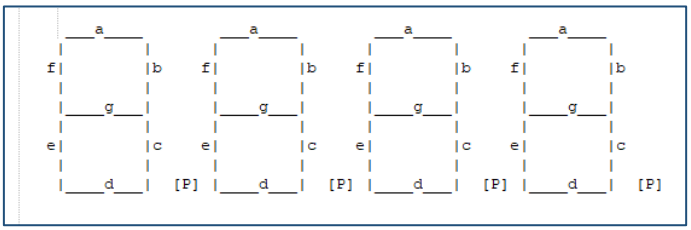


Ilustración 62: Display explicado

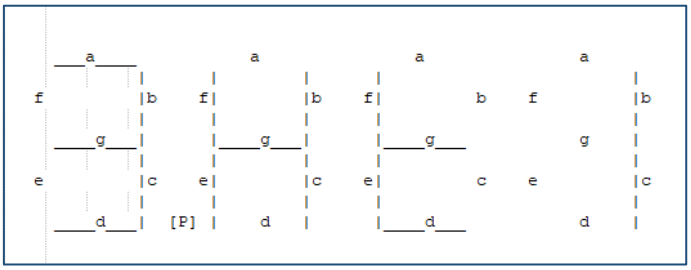


Ilustración 63:Display deseado

Para obtener el código para el *display*, este es una variable de 32 bits pues el mismo está formado por cuatro dígitos, ordenados como "**PgfedcbaPgfedcbaPgfedcbaPgfedcba**", y cada valor se aplica al dígito en esa misma posición. Para utilizar el *display*, los valores que se quieren encendidos deben dejarse a '0' y los que se quiera apagados a '1'. Con esto, el resultado deseado es: **"00110000100010011000011111111001"** para la primera parte del programa, y **"00011001100010011000011110100100"** para la segunda**.**

4. Procedimiento

Una vez obtenidos los valores anteriormente mencionados, siendo en este caso los mostrados en la siguiente ilustración (Ilustración 64).

```
Matrix

Parte 1:
Display: 00110000100010011000011111111001 (3.Mt1)
Posición RAM: 4A
Tamaño: 182
Inicio en : 36979 (36978)

Parte 2:
Display: 00011001100010011000011110100100 (4.Mt2)
Posición RAM: 800
Tamaño: 2048
Inicio en : 37160 (Anterior + 182)
```

Ilustración 64:Valores programa

Los siguientes pasos son:

1- Escribir los valores de cada parte en el fichero “programs.hex”.
2- Saber dónde se van a colocar los programas, se recomienda por detrás de los programas ya existentes en dicho fichero, en este caso: Bloque 2, Posiciones 3 y 4.
3- Actualizar el valor máximo en el fichero “cassette.v”, línea >40. En este caso, se quiere que en el bloque 2, llegue hasta 4. Se aumenta el valor de 2 a 4, como se muestra en la siguiente ilustración (Ilustración 65).

```
38    always @(posedge clk25)  //3. CHANGE MAXIMUM PICK OPTION
39    begin
40       case (block)
41          0,1: max <= 4'd9;
42
43          2: max <= 4'd4;
44
45          default: max <= 4'd0;
46       endcase;
47    end;
```

Ilustración 65: Paso 3 - Cassette.v

4- De manera análoga, en el fichero “display.v” se añaden los valores que se mostrarán en el *display* para indicar la lista de programas guardados en el fichero correspondiente en las posiciones seleccionadas, en este caso, 3 y 4 del bloque 2. El resultado final, se puede ver en la ilustración siguiente (Ilustración 66).

```
73          endcase
74          end
75       2: begin
76          case(pick)
77             0: x = 32'b01000000100010011010001110101011; //0.Mon
78             1: x = 32'b01111001100010011000110011111001; //1.Wp1 (HP2)
79             2: x = 32'b00100100100010011000110010100100; //2.Wp2 (HP2)
80             3: x = 32'b00110000100010011000011111111001; //2.Mt1 (Ht2)
81             4: x = 32'b00011001100010011000011110100100; //2.Mt2 (Ht2)
82
83             default: x = 32'b10000000100000001000000010000000; //8888
84
```

Ilustración 66:Paso 4 - Display.v

Es conveniente informar que, si se llena un bloque, con más de 10 elementos, se puede utilizar el bloque 3, que está vacío, con una estructura similar a la ya propuesta. En ambos casos:

5- Actualizar los valores del fichero que se utiliza en "ram.v", en dos casos, al abrir el fichero por medio de la instrucción *$readmemh(…)*, donde requiere actualizar el valor final al apuntado anteriormente y actualizar el tamaño de la variable "programs.hex" con el mismo valor. El valor que requiere es el valor total del fichero, en este caso 39208, es 1 menos, del valor completo del fichero (39209), siendo la última posición escrita. En la ilustración siguiente (Ilustración 67) se observa que hay que cambiar.

```
reg [7:0] ram_data[0:8191];
reg [7:0] programs[0:39208]; //5. SIZE OF THE FILE PROGS (ALL-1)

reg start;
reg [13:0] addressMem;
reg rdy;
reg fnl;
reg finish;
reg [15:0] pos;
reg [15:0] size;
reg [15:0] add_prog;
reg [15:0] end_prog;

initial
begin
   $readmemh(RAM_FILENAME, ram_data, 0, 8191);
   $readmemh(PROGS, programs, 0, 39208);  //5. SIZE OF THE FILE PROGS (ALL-1)
```

Ilustración 67: Valores que cambiar - ram.v

6- Añadir los valores necesarios para cada caso (3 y 4 del bloque 2, en este caso), para lo que se necesita el valor donde comenzará a escribir, el tamaño del fichero, la posición en "programs.hex" y en caso de ser una posición antes de la dirección 280, actualizar el valor de la variable de la memoria RAM, "addressMem" a 0, para que empiece desde el inicio de la RAM.

Se ha colocado y enumerado de tal manera, porque el primer código, además de escribir en las posiciones iniciales de la memoria, vacía el resto de la memoria. Dejando limpia la RAM para el segundo programa.

En este caso, los valores se muestran en la siguiente ilustración (Ilustración 68). Son: la dirección donde comienza en hexadecimal, el tamaño apuntado, la posición de inicio apuntada (uno menos de la posición real).

```
3: begin //PRE $280
      pos <= 16'd74;                //Pos RAM hexadecimal
      size <= 16'd182;              //Size code
      add_prog <= 16'd36978;        //Initial pos in file
      end_prog <= add_prog + size;  //Automatic
      addressMem <= 14'd0;          //Is PRE-$280
   end
4: begin //POST $280
      pos <= 16'd2048;              //Pos RAM hexadecimal
      size <= 16'd2048;             //Size code
      add_prog <= 16'd37160;        //Initial pos in file
      end_prog <= add_prog + size;  //Automatic
   end
```

Ilustración 68:Valores añadidos - ram.v

7- Por último, antes de compilar y ejecutar, asegurarse que todo está guardado, desde "programs.hex" y dentro del proyecto los archivos modificados. Una línea futura de este proyecto arreglaría este proceso, de forma automática.

5. Pruebas

Este apartado, es una prueba de que efectivamente, se ha actualizado el proyecto, con el código deseado, con el *display* y posiciones deseados. Aunque sea el mismo procedimiento que en el caso del Inicio del proyecto del anterior Anexo II.

Como se puede ver en las ilustraciones siguientes, en la primera (Ilustración 69) se puede ver el *display* en el bloque deseado de la primera sección del código. Mientras en la siguiente ilustración (Ilustración 70) se ve el correcto *display* de la segunda parte. Hay que tener en cuenta que se ha cargado en el bloque 2, teniendo el LED 4 encendido.

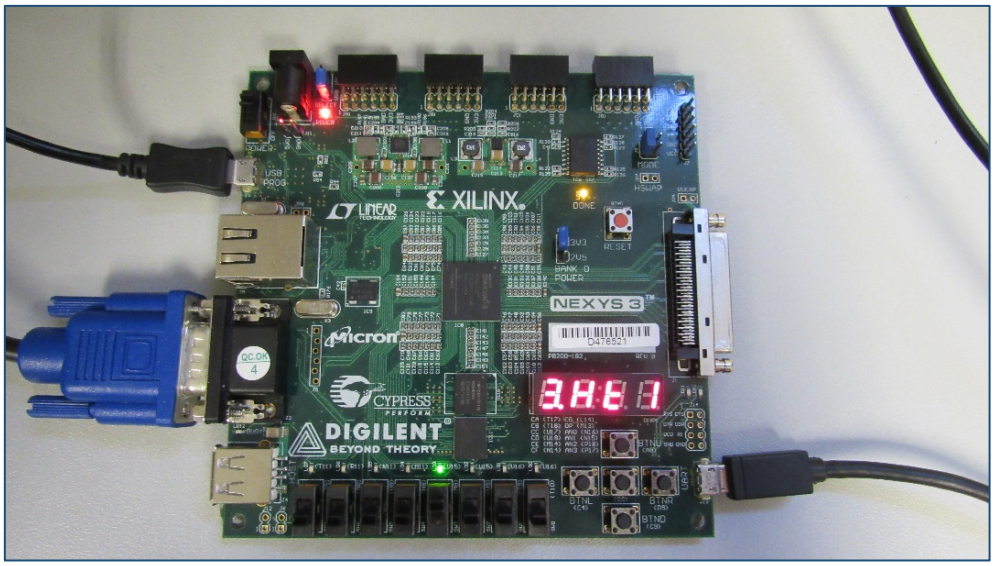


Ilustración 69: Fotografía del correcto display 1

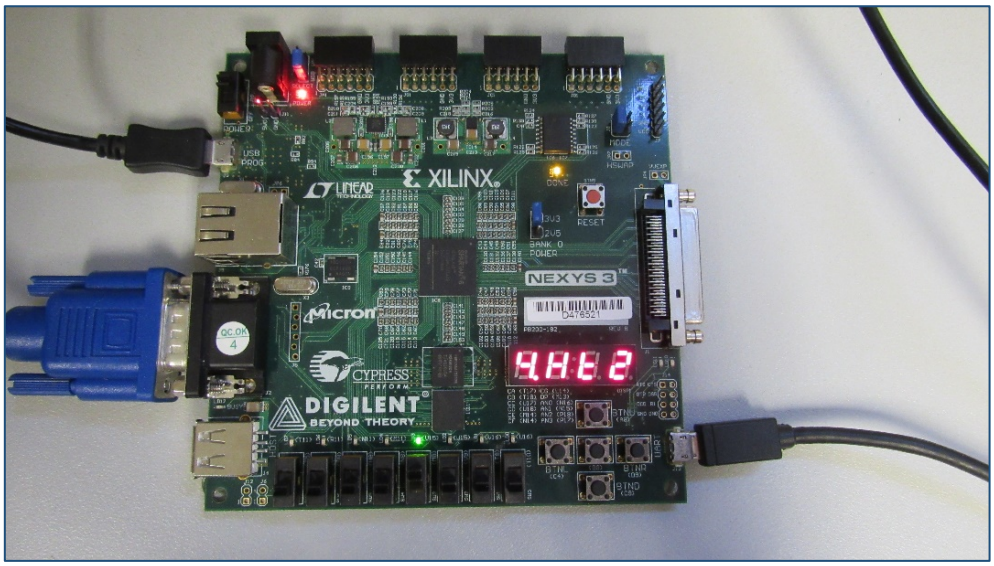
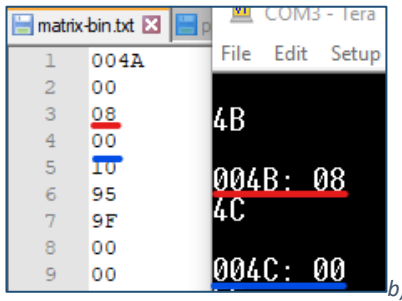


Ilustración 70: Fotografía del correcto display 2

A continuación, es necesario comprobar la correcta carga de cada uno de los códigos, para lo cual comprobará que la dirección 4B y 801 corresponden a la segunda parte de cada código. Y se verá

si la última dirección también se corresponde. En la Ilustración 71 se puede ver que esto se cumple para el primer programa, antes de $0280.

a) 

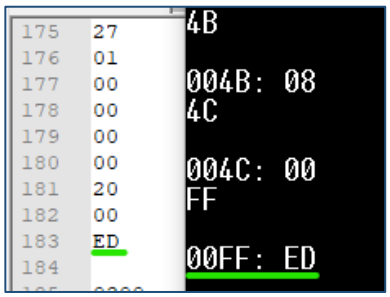
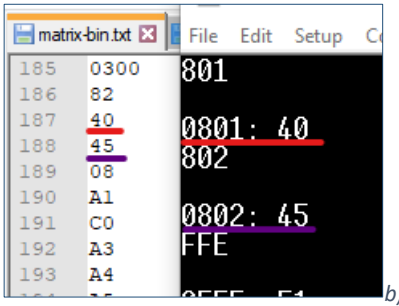
b) 

Ilustración 71: Capturas integridad primera parte del programa. a) Inicio 4B y 4C; b) Final FF

Como se comentó anteriormente, también se cumple para la segunda parte, mostrándose en la siguiente ilustración (Ilustración 72) el resultado obtenido.

a) 

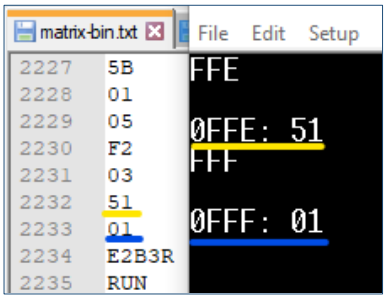
b) 

Ilustración 72: Capturas integridad segunda parte del programa. a) Inicio 801 y 802; b) Final 1FFE y 1FFF

Por último, la demostración del funcionamiento, al ser Matrix, un programa escrito en BASIC, se tiene que entrar usando **"E2B3R"**, y después se ejecuta con RUN.

En la imagen (Ilustración 73), se puede ver que el funcionamiento ha sido un éxito y se ha cargado correctamente el nuevo programa.

Ilustración 73: Ejecución programa cargado Matrix

6. Conclusión

Para finalizar este anexo, se ha intentado resolver de la manera más sencilla posible la forma de introducir un nuevo programa, facilitando y dando más uso al sistema creado del Apple I, si bien, sigue siendo un proceso no trivial. Existiendo varias maneras de introducir programas, habiéndose considerado que esta es la mejor, dada la gran potencia de la máquina y su suficiente almacenamiento. En el caso del proyecto realizado, la memoria disponible es de 8 KB, por tanto, se pueden encontrar ciertos programas que no se puedan instalar el él mismo.