UNIVERSITAT OBERTA DE CATALUNYA
INGENIERÍA TÉCNICA INFORMÁTICA DE SISTEMAS

TFC

EMULADOR DE UN MICROCONTROLADOR PIC 16F84 BASADO EN UNA FPGA

AUTOR: ISIDRO BAS GAGO
CURSO: 2º SEMESTRE 2004-2005

1 INTRODUCCIÓN

1.1 OBJETIVOS

Los programadores de PC disponen de potentes entornos de programación que les permiten programar y depurar sus aplicaciones en el mismo sistema en que se ejecutarán. Entre las características que estos entornos de programación ofrecen a los desarrolladores de software encontramos la posibilidad de situar puntos de interrupción, la ejecución paso a paso y la consulta de los valores de las variables, de los registros del procesador, de la memoria, etc. Estas herramientas son de una gran ayuda para el programador y les permite depurar sus aplicaciones y aumentar su productividad.

En cambio los programadores de microcontroladores de sistemas electrónicos empotrados se encuentran con más inconvenientes a la hora de depurar sus aplicaciones. Esto es debido principalmente a dos características de la programación de microcontroladores:

- la mayoría de los microcontroladores no disponen de la arquitectura necesaria para que el microcontrolador trabaje en modo de depuración.

- La programación del software se hace en un sistema totalmente diferente a aquel en el que será ejecutado (generalmente un PC).

Para facilitar su labor de desarrollo los programadores de microcontroladores tienen las siguientes herramientas:

- simuladores. Son programas para PC que simulan el funcionamiento de un microcontrolador. Tienen el inconveniente de que no trabajan en tiempo real ni con las señales físicas reales que condicionarán su funcionamiento.

- Emuladores. Son sistemas electrónicos diseñados de tal forma que simulan el funcionamiento de un microcontrolador y que incorporan la posibilidad de realizar la depuración del software en unas condiciones más parecidas a las reales que las que podemos encontrar en un simulador de PC.

El objetivo de este proyecto es investigar la viabilidad de realización de emuladores de microcontroladores basados en circuitos electrónicos de lógica programable mediante un anteproyecto que analice las técnicas y herramientas necesarias.

En la ilustración 1 podemos observar un esquema de bloques los diferentes elementos que participarán en la realización de este emulador. En verde están los componentes a desarrollar.

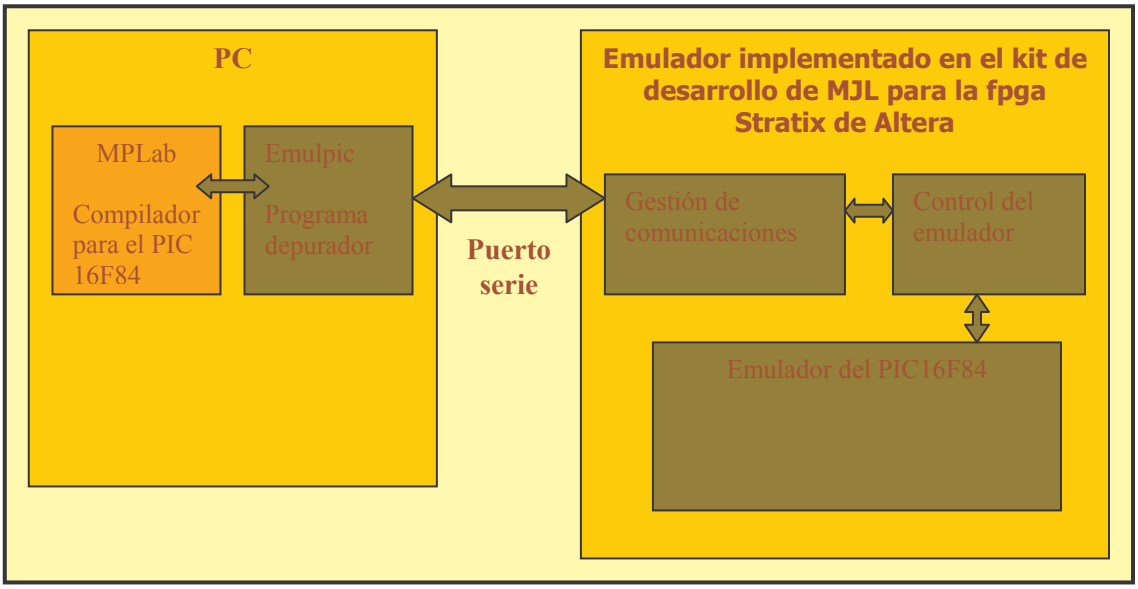


Ilustración 1. Esquema del emulador

En el proyecto podemos distinguir los siguientes componentes:

- MPlab. Es un compilador desarrollado por Microchip para su familia de microcontroladores. Mplab es un programa del fabricante de microcontroladores PIC Microchip-

- Emulpic. Programa para el PC que se deberá desarrollar en el proyecto. Emulpic es el software que desarrollaré para comunicar el PC con el emulador.

- Kit de desarrollo de MJL. Esta tarjeta de desarrollo dispone de una FPGA Altera Stratix sobre la que se programará el emulador del microcontrolador y la circuitería auxiliar necesaria (bloque de comunicaciones y gestor del emulador). La FPGA del kit de desarrollo la programaré con el emulador del microcontrolador que desarrollaré.

La función de cada uno de estos componentes es:

- MPLab. Con este programa escribiré y compilaré los programas que más adelante depuraré con el emulador. Los programas de pruebas los realizaré en el lenguaje ensamblador del Microcontrolador.

- Programa depurador Emulpic. Su función será leer los programas realizados con el compilador MPLab y presentarlos al usuario. Desde Emulpic se podrán poner puntos de interrupción, realizar ejecuciones paso a paso, consultar el estado de la memoria y de los registros del emulador, etc. Este programa se desarrollará con el entorno de desarrollo Microsoft C++ .NET.

- En la fpga Stratix en la que se programará el emulador se distinguen tres grandes bloques:

    - Gestión de comunicaciones. Este bloque funcional será el encargado de recibir los datos del PC y transmitirlos al gestor del emulador. También enviará al PC los datos que le sean transmitidos por el gestor del emulador.

    - Gestor del emulador. A partir de los comandos recibidos por el PC se encargará de realizar las acciones pertinentes para gestionar el funcionamiento del emulador.

    - Emulador. Su diseño estará realizado para obtener un funcionamiento idéntico al microcontrolador PIC 16F84.

La programación de la FPGA se realizará con el lenguaje VHDL (VHSIC Hardware Description Language) y se programará en el entorno de desarrollo ModelSim 5.7

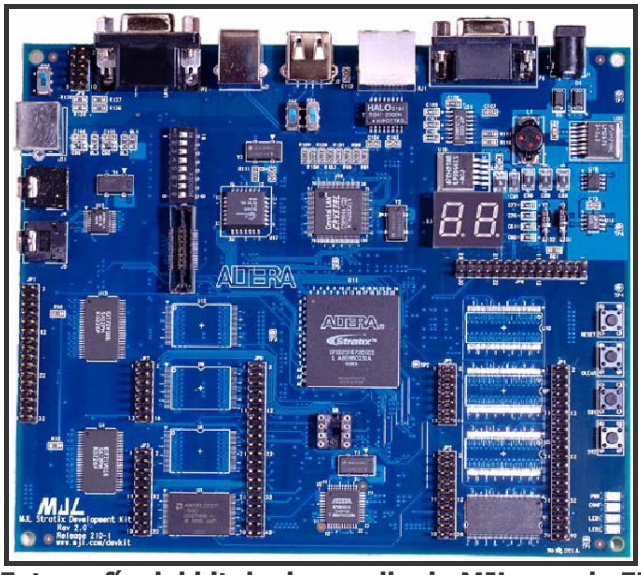


Ilustración 2. Fotografía del kit de desarrollo de MJL para la FPGA Altera Stratix

1.2 DESCRIPCIÓN DEL COMPILADOR MPLAB 7.0

El entorno de desarrollo Mplab IDE para sistemas operativos de Microsoft es una herramienta de Microchip que permite el desarrollo de software para los microcontroladores PIC.

Sus características son:

- soporte para los diferentes procesadores de Microchip. A la hora de desarrollar una aplicación el programador debe seleccionar el dispositivo de destino. Para realizar el proyecto seleccionaremos el microcontrolador PIC16F84.

- Posibilidad de desarrollo de aplicaciones en lenguaje C y en lenguaje ensamblador. El proyecto estará limitado a aplicaciones escritas en lenguaje ensamblador.

- Posibilidad de utilización de diferentes compiladores y ensambladores. En función del lenguaje utilizado, del microcontrolador escogido y de los compiladores instalados, el desarrollador podrá escoger el compilador que desea o debe utilizar.

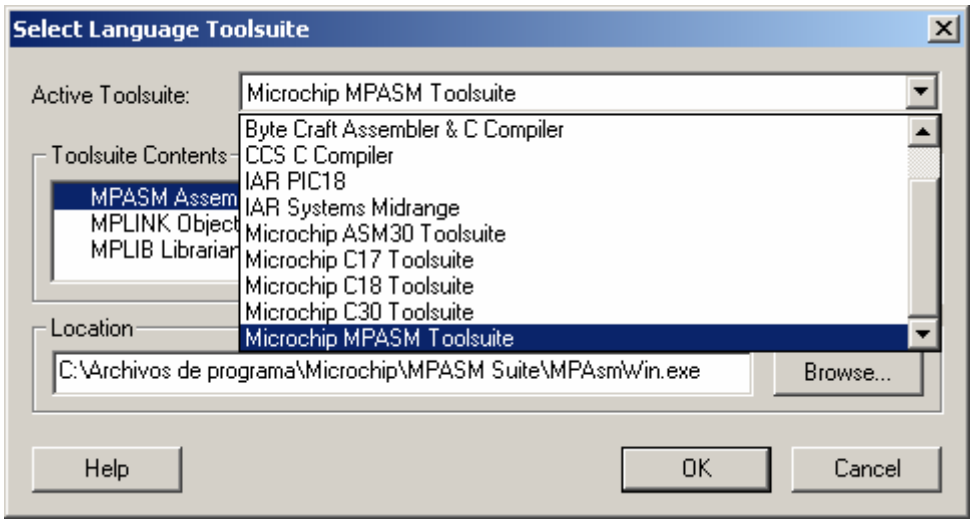


Ilustración 3. MPLab permite escoger el compilador que se desea utilizar.

- Simulador. El entorno de desarrollo permite realizar una simulación en el PC del software desarrollado. En función de las herramientas instaladas el programador puede seleccionar el depurador que desea utilizar. Mediante puntos de interrupción y ejecución paso a paso el desarrollador puede ir avanzando por las diferentes líneas del programa y puede ir observando el

estado de la memoria, de los registros del microcontrolador, de la pila, etc. Esta simulación no se hace en tiempo real, por lo que sólo es una ayuda para el desarrollo del programa, no asegura su correcto funcionamiento en el microcontrolador bajo condiciones reales.

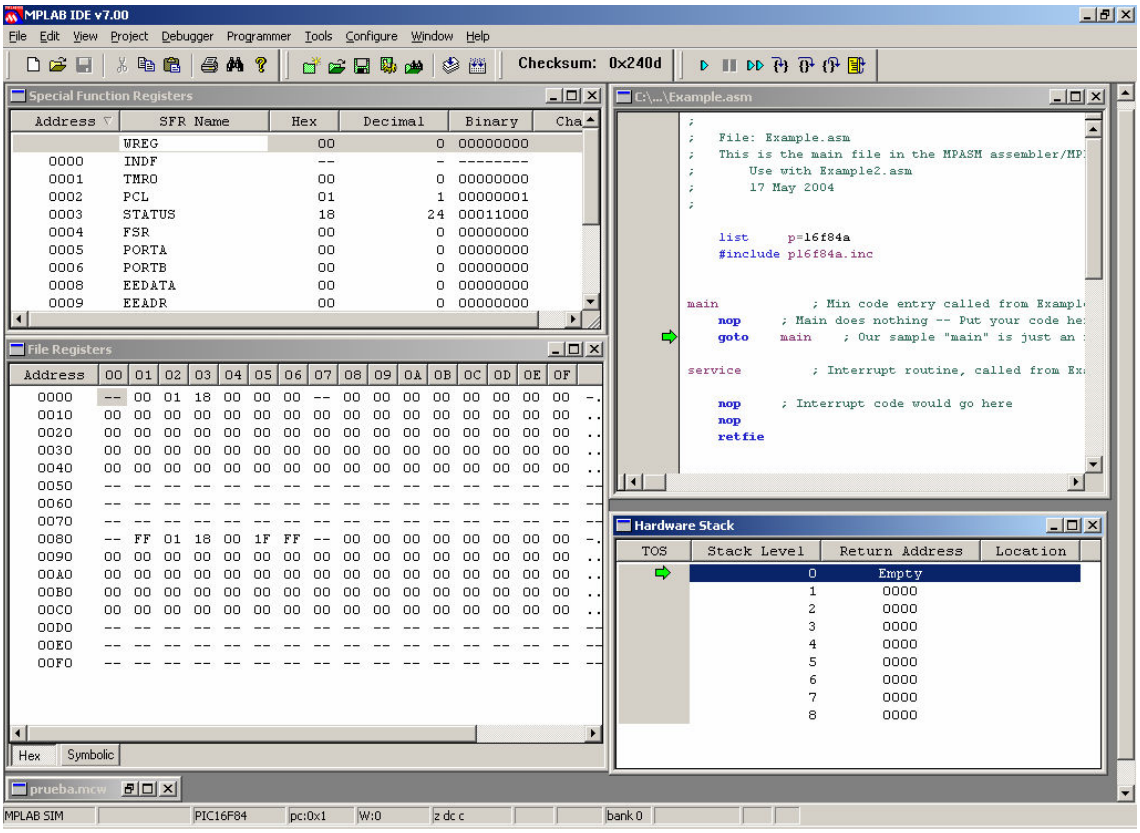


Ilustración 4. Entorno de desarrollo MPLab

El programa Emulpic leerá los archivos de los proyectos de MPLab:

- el archivo de código fuente lo presentará al usuario en pantalla.
- el archivo ensamblado lo enviará al emulador para que lo ejecute.
- Con los archivos generados por el compilador de Mplab y los puntos de interrupción que sitúe el usuario, el programa Emulpic generará la información necesaria para que el emulador pueda realizar adecuadamente la gestión de los puntos de interrupción y la ejecución paso a paso.

1.3 ELECCIÓN DE LA HERRAMIENTA DE DESARROLLO PARA EMULPIC

Debido a la complejidad del proyecto había considerado necesario minimizar el tiempo necesario para desarrollar EmulPic. Por este motivo había escogido inicialmente Microsoft Visual Basic 6.0 como herramienta de desarrollo, ya que de todos los entornos de desarrollo que conozco era el que me ofrecía una mayor rapidez a la hora de realizar esta aplicación.

Las ventajas que me ofrecía Visual Basic 6.0 era:

- un desarrollo rápido del interface gráfico de la aplicación.

- El control MSComm de Visual Basic 6.0 es una herramienta que permite la implementación de las comunicaciones serie con el emulador de manera sencilla.

- Tengo una amplia experiencia en el desarrollo de aplicaciones con VB.

Tras un primer prototipo de la aplicación los resultados obtenidos no eran los deseados. Mi intención era realizar una aplicación con una apariencia profesional igual que la de los programas de Microsoft, que incluyera barras de herramientas flotantes, menús y barras de herramientas con aspecto de Windows XP, etc. Para conseguir estas características me vería obligado a utilizar componentes para Visual Basic de los que no dispongo, realizar complejas rutinas utilizando llamadas a la API de Windows o programas mis propios controles que me dieran el aspecto deseado. Por este motivo y aunque a priori parecía que Visual Basic 6.0 fuera la herramienta de desarrollo adecuada la he descartado y he utilizado ***Microsoft C++ .NET*** con las librería de clases ***MFC*** de Microsoft que permiten conseguir las características deseadas para Emulpic.

1.4 BREVE INTRODUCCIÓN AL DISEÑO DE SISTEMAS DIGITALES CON VHDL

1.4.1 DISPOSITIVOS DE LÓGICA PROGRAMABLE

Entre los dispositivos de que disponen actualmente los diseñadores de sistemas electrónicos digitales encontramos:

- Micropocesadores, microcontroladores, procesadores digitales de señal.
- FPGA (Field Programmable Gate Array).
- ASIC (Application Specific Integrated Circuit).
- SOC (System On Chip)

Los microprocesadores ofrecen una elevada flexibilidad ya que su arquitectura y la posibilidad de programación les hace muy versátiles a la hora de afrontar el diseño de casi cualquier aplicación. No obstante, a pesar de que sean fáciles de programar y tengan una velocidad media alta, la realización de determinadas tareas puede requerir programas complejos que requieren ejecutar un gran número de instrucciones que les hace perder prestaciones.

La posibilidad que tienen las FPGA de ser configuradas con sistemas secuenciales síncronos diseñados específicamente para resolver un determinado problema las hace tener más prestaciones que los microcontroladores pero no tienen la flexibilidad que ofrece el microcontrolador.

Los ASIC son los circuitos integrados que ofrecen mayores prestaciones de velocidad pero por la elevada complejidad de su diseño sólo son rentables para grandes lotes de producción.

Los SOCs (System On Chip) son unos circuitos integrados híbridos que disponen de un microcontrolador, una FPGA, memoria interna e incluso sistemas para el tratamiento analógico de la señal. Estos circuitos integrados combinan las ventajas de las FPGA y los microcontroladores.

El emulador del microcontrolador PIC 16F84 se realizará en una FPGA. Veamos con más detalle las características de estos circuitos integrados.

El diseño interno de las FPGA es diferente según el modelo y el fabricante, pero todas tienen una estructura similar que se puede ilustrar con el esquema siguiente:

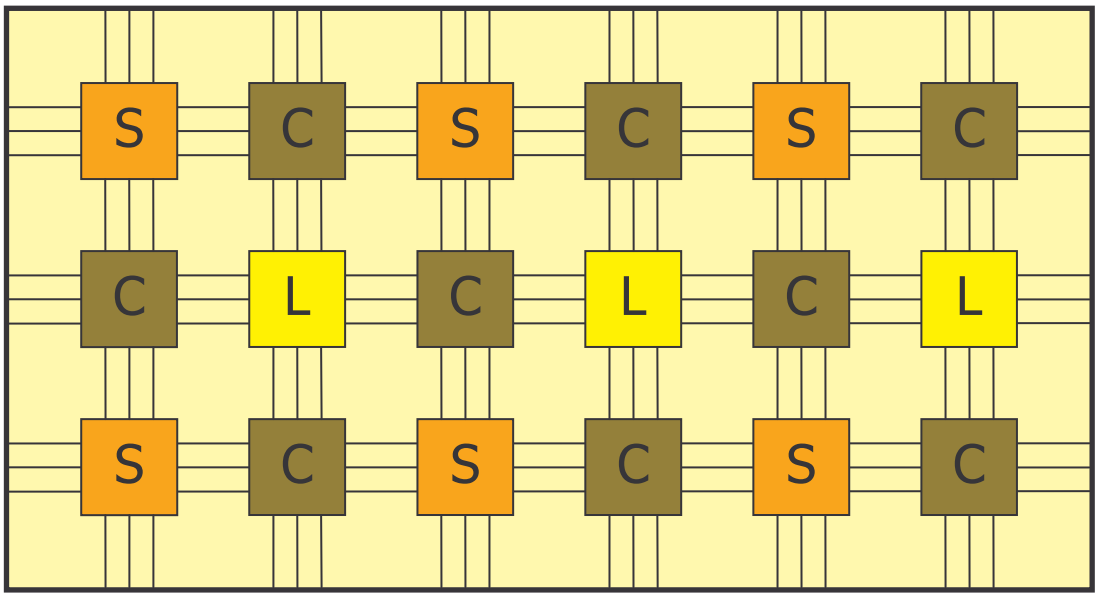


Ilustración 5. Esquema de una FPGA

En la ilustración 5 podemos ver como una FPGA está compuesta de una serie de celdas interconectadas. Las celdas C son recursos compuestos de elementos secuenciales y elementos combinacionales:

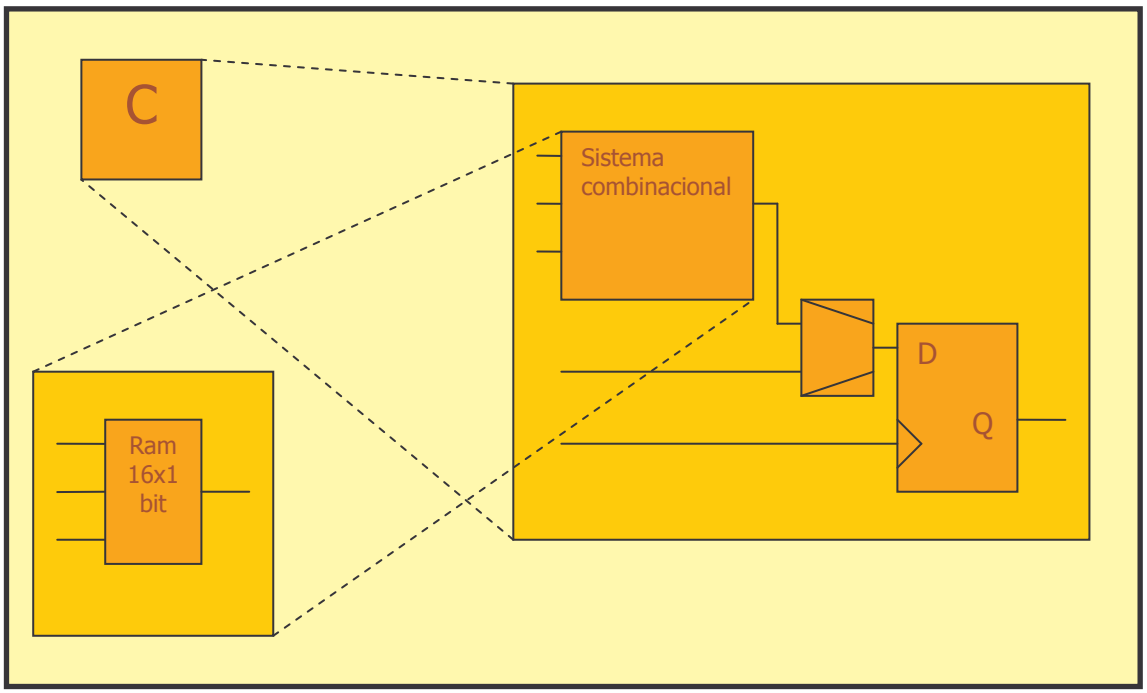


Ilustración 6. Esquema de una célda C

Los elementos secuenciales de las celdas C son registros cuyo estado cambia con la señal de reloj. Como elementos combinacionales podemos encontrar pequeñas

memorias programadas para dar una salida determinada frente a unas determinadas entradas. Con estas memorias podríamos simular, por ejemplo, combinaciones de puertas lógicas. Además de los elementos combinacionales simulados con memorias podemos encontrar otros elementos combinacionales que en la ilustración se presenta como un multiplexor.

Las celdas S y C son recursos de interconexión que permiten comunicarse a las celdas L:

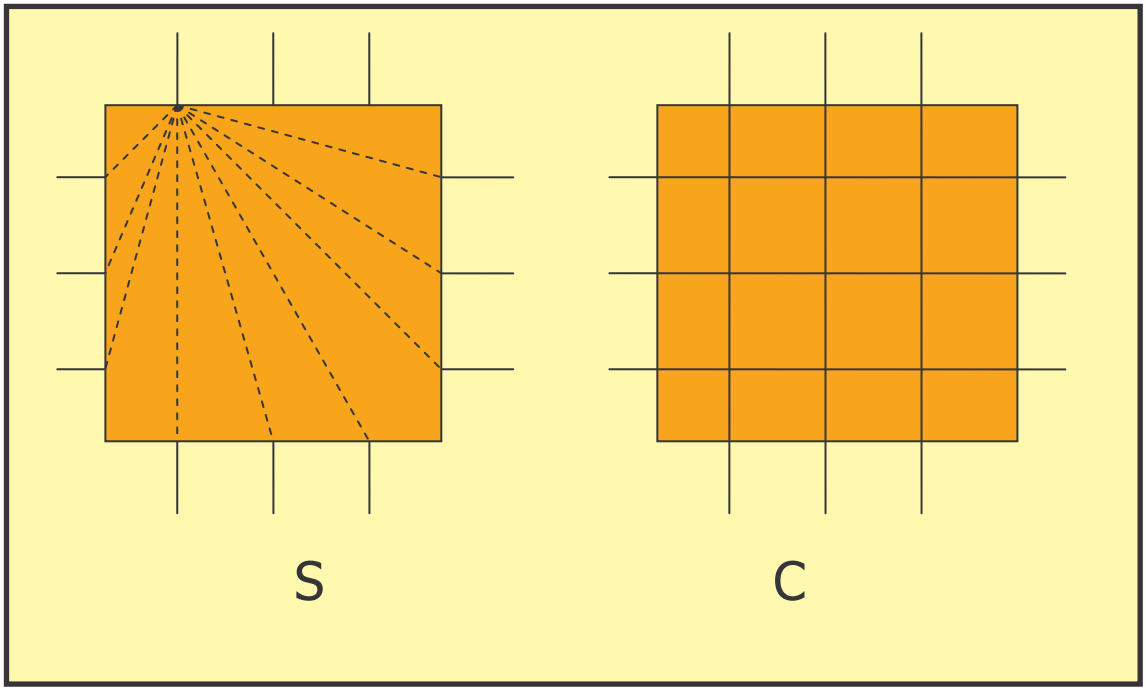


Ilustración 7. Celdas S y C

La interconexión de los recursos internos de las celdas L, la programación de las memorias y la interconexión de los recursos C y S se puede programar para obtener el diseño deseado. Esta programación puede ser única con técnicas de fusible/antifusible o puede ser reprogramable, guardando la configuración en memorias y cargándola al inicializarse el sistema.

1.4.2 CICLO DE DISEÑO CON VHDL

VHDL son las siglas de VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. Es un lenguaje que permite describir el comportamiento, estructura e implementación de circuitos electrónicos.

En el proceso de diseño de un componente electrónico con VHDL podemos distinguir las siguientes fases:

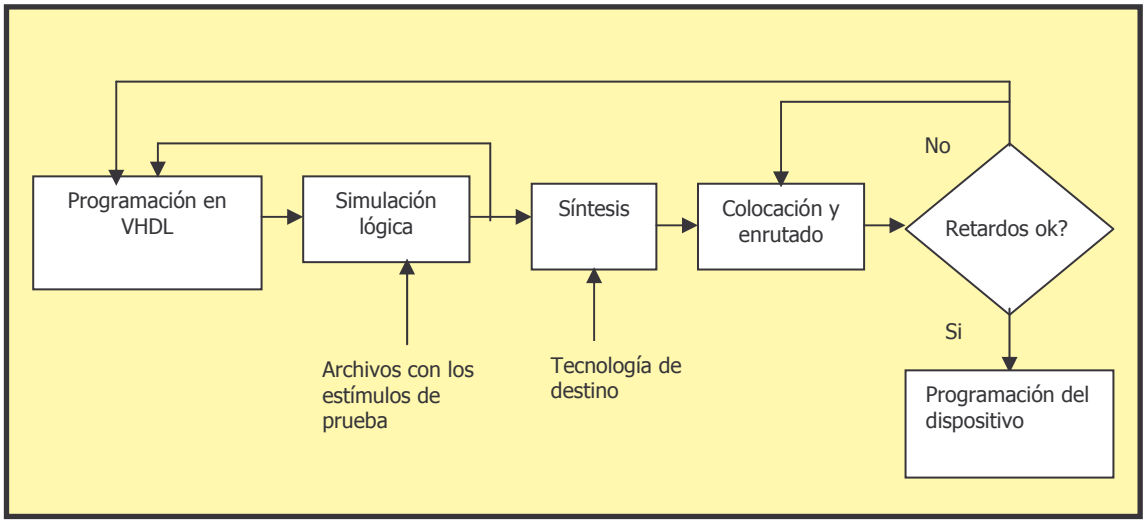


Ilustración 8. Ciclo de diseño con VHDL

- Programación. Desde los entornos de desarrollo para VHDL podemos realizar el programa en VHDL y compilarlo.

- Simulación lógica. Los simuladores de los entornos de desarrollo permiten simular el diseño realizado. El diseñador del circuito puede definir los archivos de estímulos (señales de entrada al dispositivo) y visualizar la evolución temporal del sistema para comprobar si el funcionamiento se ajusta al deseado. En esta simulación no se tienen en cuenta los retardos físicos que puede tener la señal eléctrica entre los diferentes componentes del sistema. Este retardo puede ser crítico para el correcto funcionamiento del sistema y deberá ser comprobado en una fase posterior. Si los resultados obtenidos en la simulación no se ajustan a las especificaciones se debe modificar el diseño.

- Síntesis. Una vez que el diseño del sistema funciona correctamente a nivel lógico se procede a realizar la etapa de síntesis que consiste en traducir el diseño anterior para adaptarlo a la arquitectura del circuito de lógica programable que se desea utilizar. En esta fase le indicaremos al programa que

realiza la síntesis el CI integrado que deseamos utilizar. Como resultado se obtiene una lista de los recursos que se usaran en el circuito integrado (puertas, registros, etc.) y como se deberán conectar estos componentes entre sí.

- Colocación y enrutado. Además de dar un listado de los recursos que se han de usar, el proceso de síntesis ofrece una posible ubicación y conexionado de los diferentes componentes dentro del circuito integrado. Esta ubicación se puede optimizar para obtener una mayor velocidad, para ocupar un área menor en el circuito integrado, etc. El diseñador puede modificar la ubicación de los diferentes componentes en función de sus necesidades específicas.

- Comprobación de los retardos. Una vez que se dispone de la ubicación física real de los componentes dentro del circuito de lógica programable se puede proceder a obtener los retardos de la señal entre los diferentes componentes. Con estos retardos podemos comprobar si el comportamiento que hemos simulado a nivel lógico funcionará también en la implantación física del sistema en el circuito integrado. Supongamos por ejemplo que queremos que nuestro diseño funcione a 40 MHz. En la lista de retardos hemos visto que tenemos un retardo máximo de 50ns. Esto quiere decir que entre dos componentes del circuito la señal tarda en llegar 50ns lo que significaría que el sistema podría trabajar como máximo a una frecuencia de $1/(50 \cdot 10^{-9}s)=20MHz$. En conclusión, el diseño que habíamos realizado y comprobado con una simulación lógica no puede llegar a funcionar a la frecuencia dada en las especificaciones. Para solucionar este problema se puede modificar la colocación y enrutado de estos dos componentes para hacer que queden más próximos o volver a la etapa de programación y modificar el diseño.

1.4.3 INTRODUCCIÓN AL LENGUAJE VHDL

Para realizar en pocas líneas una descripción del lenguaje VHDL veremos el ejemplo de cómo se define un multiplexor de 4 entradas con VHDL.

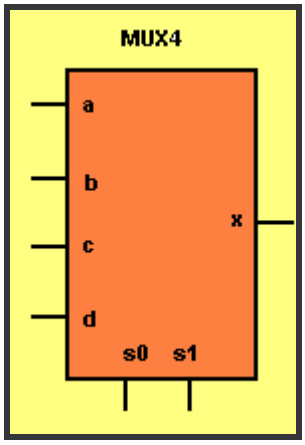


Ilustración 9. Multiplexor de cuatro entradas

A la hora de definir un componente en VHDL primero debemos describir como se relaciona dicho componente con otros componentes, es decir debemos definir su interface, que líneas de entrada y salida tiene. En VHDL a esta definición se le llama ENTITY:

```
ENTITY mux4 IS
PORT ( a, b, c, d : IN BIT;
s0, s1 : IN BIT;
x, : OUT BIT);
END mux;
```

Como se puede ver en el código VHDL hemos definido la entidad mux4 que será un multiplexor de 4 entradas. Mediante el comando PORT hemos definido las líneas de entrada y salida que posee el multiplexor. Las señales a, b, c, d, s0 y s1 son señales de entrada de un bit y la señal x es una señal de salida de 1 bit.

Una vez definida la interface del componente se procede a definir su estructura interna, o lo que es lo mismo, su comportamiento. A esta definición en VHDL se le llama ARCHITECTURE. Existen varias formas de definir la arquitectura de un componente. A continuación se expondrán cada una de ellas para ilustrar los distintos modos de trabajo que podemos utilizar para VHDL. Todas las arquitecturas que se definen a continuación cumplen la siguiente tabla de la verdad:

s1	s0	X
0	0	a
0	1	b
1	0	c
1	1	d

Arquitectura de flujo de datos

```
ARCHITECTURE mux4a OF mux4 IS
SIGNAL sel: INTEGER;
BEGIN
WITH sel SELECT
q <= a AFTER 10 ns WHEN 0,
       b AFTER 10 ns WHEN 1,
       c AFTER 10 ns WHEN 2,
       d AFTER 10 ns WHEN 3,
       'X' AFTER 10 ns WHEN OTHERS;
sel <= 0 WHEN s0 = '0' AND s1 = '0' ELSE
1 WHEN s0 = '1' AND s1 = '0' ELSE
2 WHEN s0 = '0' AND s1 = '1' ELSE
3 WHEN s0 = '1' AND s1 = '1' ELSE
4 ;
END mux4;
```

En este modo de definición de la arquitectura se ha definido una señal interna, sel, que podrá tomar valores enteros. Las señales de VHDL se diferencian de las variables en que tienen asociada una lista con los valores que han tomado o tomarán en el transcurso del tiempo.

En función del valor que tome sel se asignará a la salida q el valor de la entrada que corresponda (i0, i1, i2, i3). A la salida q se le añade un retardo de 10 ns respecto a la señal sel. El símbolo <= en VHDL no es menor o igual, significa asignar una señal a otra.

La señal sel tomará un valor u otro en función de los valores que tengamos en s0 y s1. Si s0 vale 0 y s1 vale 0 sel valdrá 0, si s0 vale 1 y s1 vale 0 sel valdrá1, etc.

Una característica muy importante de VHDL es que todas las sentencias anteriores se ejecutan en paralelo, o sea, simultáneamente, ya que esta es la manera como funciona un circuito electrónico (si no tenemos en cuenta los retardos de la señal).

Arquitectura estructural

```
ARCHITECTURE mux4b OF mux4 IS
COMPONENT andgate
PORT(a, b, c : IN bit; c : OUT BIT);
END COMPONENT;
```

```
COMPONENT inverter
PORT(in1 : IN BIT; x : OUT BIT);
END COMPONENT;
COMPONENT orgate
PORT(a, b, c, d : IN bit; x : OUT BIT);
END COMPONENT;
SIGNAL s0_inv, s1_inv, x1, x2, x3, x4 : BIT;
BEGIN
U1 : inverter(s0, s0_inv);
U2 : inverter(s1, s1_inv);
U3 : andgate(a, s0_inv, s1_inv, x1);
U4 : andgate(b, s0, s1_inv, x2);
U5 : andgate(c, s0_inv, s1, x3);
U6 : andgate(d, s0, s1, x4);
U7 : orgate(x2 => b, x1 => a, x4 => d, x3 => c, x => x);
END netlist;
```

Este modo de definición recurre a describir el comportamiento de un componente a partir de la utilización de otros componentes ya definidos. Los componentes que se usan en esta definición son:

- puertas and de tres entradas de 1 bit
- inversores
- puertas or de 4 entradas

La definición de estos componentes se realiza con la instrucción COMPONENT. Una vez declarados los componentes que se van a utilizar se procede a definir cuantos componentes de los que hemos definido se van a usar y como se van a conectar. El diseño de puertas lógicas que se ha realizado con esta especificación VHDL es:

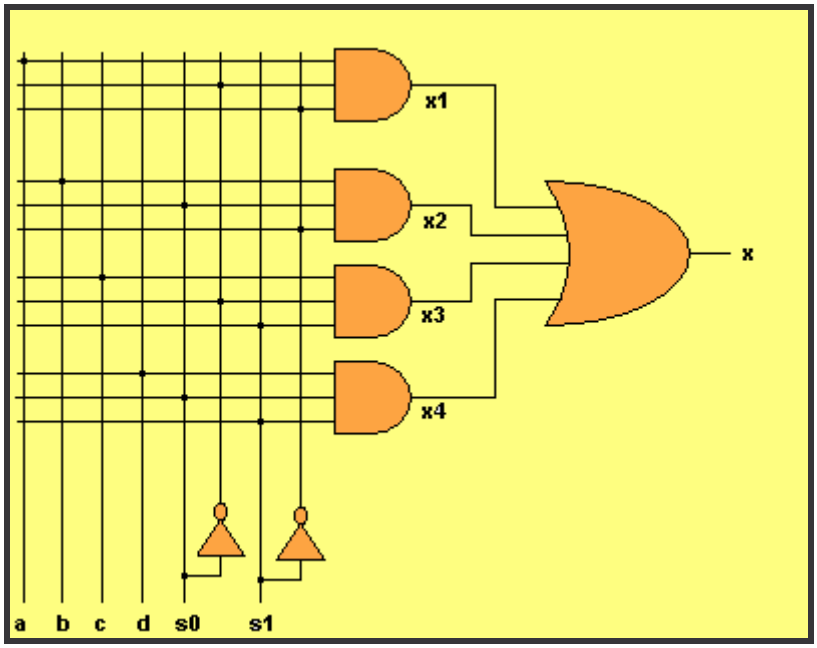


Ilustración 10.Multiplexor de 4 entradas realizado con puertas lógicas

Al igual que en el ejemplo anterior todas las sentencias de este ejemplo se ejecutan de manera concurrente.

Arquitectura comportamiento

```
ARCHITECTURE mux4c OF mux4 IS
(a, b, c, d, s0, s1 )
VARIABLE sel : INTEGER;
BEGIN
IF s0 = '0' and s1 = '0' THEN
sel := 0;
ELSIF s0 = '1' and s1 = '0' THEN
sel := 1;
ELSIF s0 = '0' and s1 = '0' THEN
sel := 2;
ELSE
sel := 3;
END IF;
CASE sel IS
WHEN 0 =>
x <= a;
WHEN 1 =>
x <= b;
WHEN 2 =>
x <= c;
WHEN OTHERS =>
x <= d;
END CASE;
END PROCESS;
END sequential;
```

Finalmente podemos definir la arquitectura del multiplexor con una arquitectura de comportamiento. Este sistema es el más parecido a la programación clásica. Este caso se comporta de manera ligeramente diferente a los anteriores en lo que respecta a la simultaneidad de ejecución de las sentencias. Todo el conjunto de instrucciones que existen entre BEGIN y END PROCESS se evalúan de forma secuencial tal y como estamos acostumbrados. Una vez que se han evaluado todas las líneas de este proceso y de otros que puedan existir se ejecutan simultáneamente todos los procesos y sentencias contenidas en la definición de esta entidad y de todas las demás que existan en el proyecto.

1.5 DESCRIPCIÓN DEL ENTORNO DE DESARROLLO PARA VHDL MODELSIM 5.7

ModelSim es un entorno de desarrollo de Mentor Graphics para diseño de componentes con VHDL o Verilog. El entorno de desarrollo dispone de:

- Librerías de componentes. Son librerías de componentes estándar que podemos incluir en nuestro diseño.
- Editor de código fuente.
- Depurador de código fuente.
- Simulador. Permite hacer una simulación lógica de nuestro diseño.

1.6 DESCRIPCIÓN DEL KIT DE DESARROLLO DE MJL

El kit de desarrollo de MJL para la FPGA Stratix 1S25 de Altera hace posible la realización del proyecto sin la necesidad de realizar el diseño de un hardware especifico.

En la siguiente ilustración podemos observar los elementos que componen la tarjeta de desarrollo de MJL.

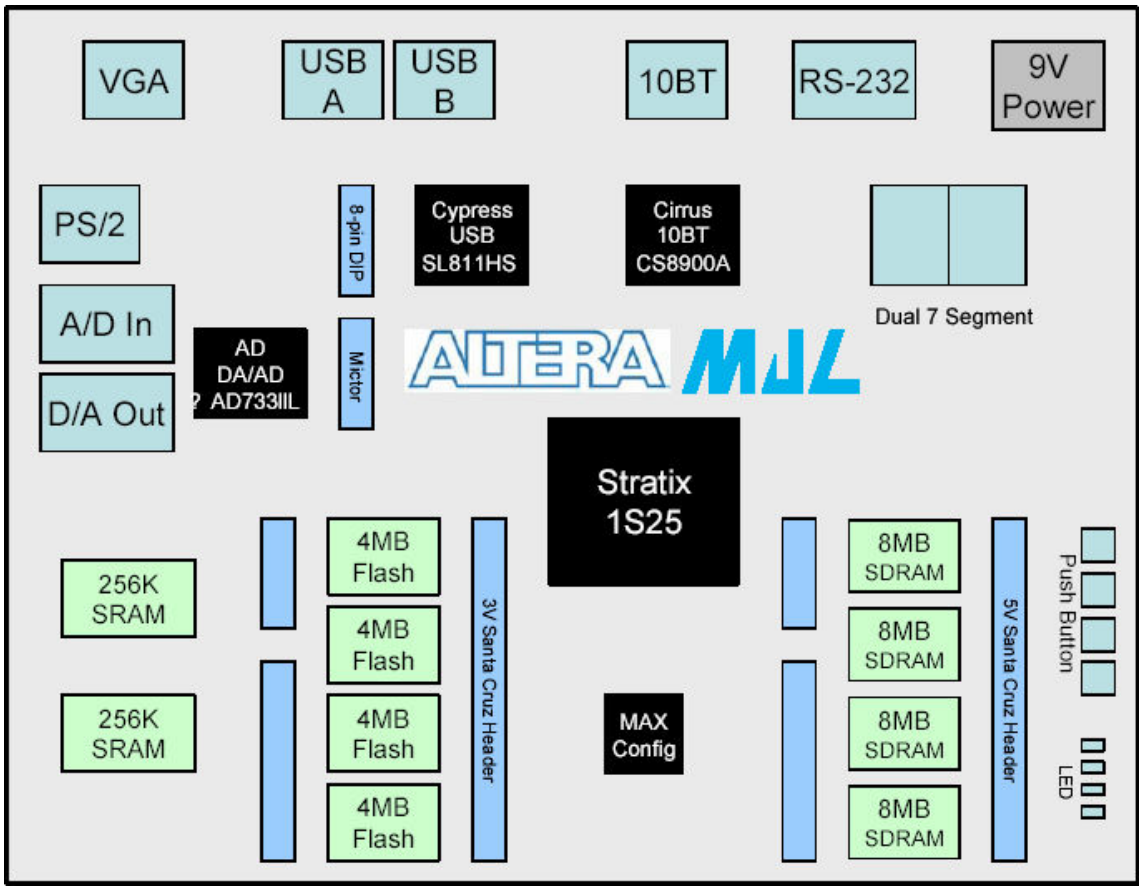


Ilustración 11. Elementos del kit de desarrollo de MJL

- FPGA Altera Stratix 1S25. Es el corazón del kit de desarrollo. Sus características principales son:

Características Stratix 1S25	
Puertas lógicas	650,000
RAM bits	1,944,576
IO pins de usuario	469

- 1 chip de 4MByte de memoria flash. Con posibilidad de ampliación hasta 16 Mbyte.

- 2 chips SRAM de 256Kbyte (128K x 16 bit) configurables como una memoria de 256K x 32 bit

- 1 chip de 64Mbit de memoria SDRAM. Con posibilidad de ampliación hasta 256Mbit.

- 2 puertos /USB

- 1 puerto VGA
- Un puerto PS/2
- Un convertidor analógico digital / digital analógico de 16 bits
- Un puerto Ethernet 10Base-T
- Un puerto RS-232
- Puertos de expansión
- Pulsadores, leds y displays 7 segmentos.

De todas estas características sólo será necesario utilizar la FPGA y el puerto serie. La memoria interna de la FPGA debería ser suficiente para emular la memoria del microcontrolador por lo que no está previsto utilizar la memoria externa (SRAM,SDRAM o FLASH).

2 PROGRAMA EMULPIC

Tal y como se ha introducido anteriormente una de las partes del proyecto es la realización de un programa que comunique el PC con el emulador. Este programa servirá para:

- Abrir programas realizados para el microcontrolador PIC 16F84 y enviarlos al emulador. Los programas se obtendrán de los archivos *.lst que genera al compilar el entorno de desarrollo MPLab de Microchip. Emulpic se limitará a abrir programas que consten de un único archivo realizado en lenguaje ensamblador y sin macros.

- Una vez hayamos cargado un programa en Emulpic el usuario podrá situar puntos de interrupción para depurar el programa.

- Con los puntos de interrupción situados el usuario podrá enviar el programa al emulador a través de la RS232

- Cuando el emulador llegue a un punto de interrupción o el usuario desee pausar la ejecución Emulpic leerá el contenido de los registros y la memoria del emulador para poder comprobar el funcionamiento del programa del PIC.

Para ofrecer a la aplicación de unas características totalmente profesionales realizaré Emulpic con el entorno de desarrollo Visual Studio C++ .NET y utilizaré la librería de clases MFC de Microsoft. De esta manera, tal como detallaré en el apartado 2.3.4, interface gráfico de la aplicación, Emulpic tendrá un aspecto como el de las aplicaciones de Windows.

Para realizar este programa he necesitado estudiar el funcionamiento de las librerías de clases MFC.

2.1 FASES DEL DISEÑO

Las fases que se han realizado en el diseño de la aplicación son las siguientes:

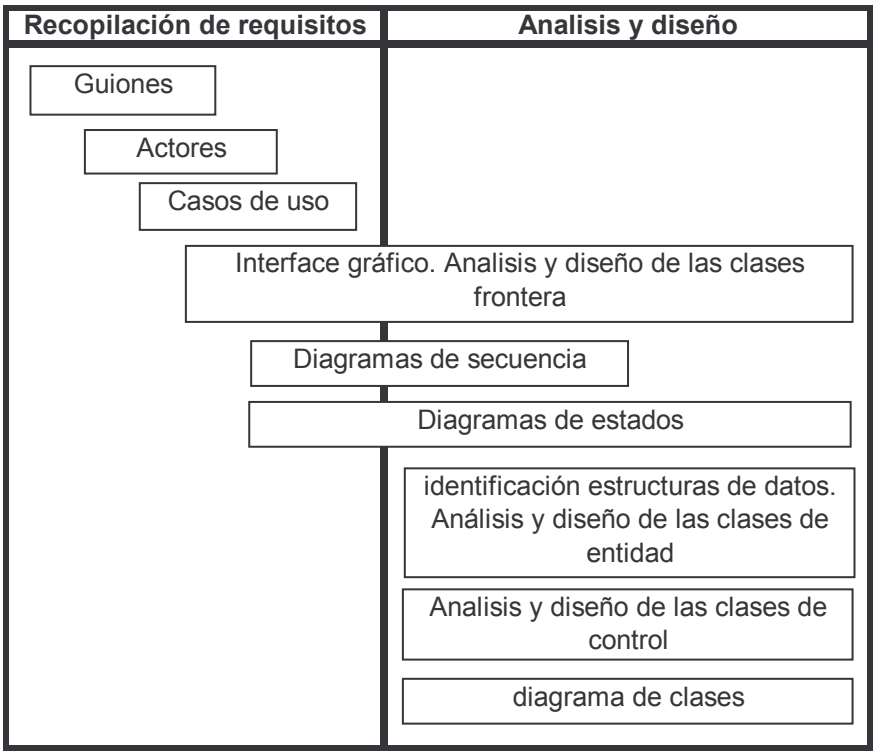


Ilustración 12. Fases del diseño

Dentro de la etapa de recopilación de requisitos me he basado en la realización de guiones, la localización de actores y de casos de uso, he recopilado información sobre el interface gráfico de la aplicación y he realizado diagramas de secuencia y un diagrama de estados del funcionamiento general que debería tener la aplicación.

Dentro de la etapa de análisis y diseño he hecho el análisis y diseño del interface gráfico (clases de frontera), he identificado estructuras de datos y he hecho su análisis y diseño (clases de entidad), he hecho el análisis y diseño de las clases de control y el diagrama de clases.

2.2 RECOPILACIÓN DE REQUISITOS

2.2.1 GUIONES

Importar archivos de MPLab

El programa Emulpic leerá los archivos *.lst que se generarán con el compilador MPLab de Microchip y mostrará el código fuente al usuario para que pueda situar puntos de interrupción. Para que sea factible la realización del proyecto en un cuatrimestre el programa se limitará a leer programas realizados en ensamblador y con un único archivo de código fuente.

El siguiente ejemplo es un sencillo programa escrito en lenguaje ensamblador para el PIC16F84:

```
;
;       File: Example.asm
;       ejemplo que saca por el puerto A la secuencia 0,3,6,9,12,...
;

        list            p=16f84a
        #include        p16f84a.inc


main                                    ; Main
        movlw           0
        addlw           3
        goto            subrutina       ; salta a una subrutina de ejemplo
        goto            main            ; hace un bucle infinito

subrutina                               ; subrutina que escribe en el puerto

        movwf           PORTA
        return


        end
```

El archivo Example.lst que corresponde al archivo anterior es el siguiente:

```
MPASM 03.90 Released                EXAMPLE.ASM   4-7-2005  16:01:23        PAGE  1

LOC  OBJECT CODE     LINE SOURCE TEXT
```

```
  VALUE

                      00001 ;
                      00002 ;       File: Example.asm
                      00003 ;       ejemplo que saca por el puerto A la secuencia 3,6,9,12,...
                      00004 ;
                      00005
Warning[215]: Processor superseded by command line.  Verify processor symbol.
                      00006         list      p=16f84a
                      00007         #include p16f84a.inc
                      00001         LIST
                      00002 ; P16F84A.INC  Standard Header File, Version 2.00    Microchip
Technology, Inc.
Message[301]: MESSAGE: (Processor-header file mismatch.  Verify selected processor.)
                      00134         LIST
                      00008
                      00009
                      00010
0000                  00011 main                        ; Main
0000  3000            00012         movlw   0
0001  3E03            00013         addlw   3
0002  2804            00014         goto    subrutina   ; salta a una subrutina de ejemplo
0003  2800            00015         goto    main        ; hace un bucle infinito
                      00016
0004                  00017 subrutina                   ; subrutina que escribe en el puerto
                      00018
0004  0085            00019         movwf   PORTA
0005  0008            00020         return
                      00021
                      00022
                      00023         end


MPASM 03.90 Released                  EXAMPLE.ASM   4-7-2005  16:01:23         PAGE  2


SYMBOL TABLE
  LABEL                           VALUE

C                                 00000000
DC                                00000001
EEADR                             00000009
EECON1                            00000088
EECON2                            00000089
EEDATA                            00000008
EEIE                              00000006
EEIF                              00000004
F                                 00000001
FSR                               00000004
GIE                               00000007
INDF                              00000000
INTCON                            0000000B
INTE                              00000004
```

```
INTEDG                00000006
INTF                  00000001
IRP                   00000007
NOT_PD                00000003
NOT_RBPU              00000007
NOT_TO                00000004
OPTION_REG            00000081
PCL                   00000002
PCLATH                0000000A
PORTA                 00000005
PORTB                 00000006
PS0                   00000000
PS1                   00000001
PS2                   00000002
PSA                   00000003
RBIE                  00000003
RBIF                  00000000
RD                    00000000
RP0                   00000005
RP1                   00000006
STATUS                00000003
T0CS                  00000005
T0IE                  00000005
T0IF                  00000002
T0SE                  00000004
TMR0                  00000001
TRISA                 00000085
TRISB                 00000086
W                     00000000
WR                    00000001
WREN                  00000002
WRERR                 00000003
Z                     00000002
_CP_OFF               00003FFF
_CP_ON                0000000F
_HS_OSC               00003FFE
_LP_OSC               00003FFC
_PWRTE_OFF            00003FFF
_PWRTE_ON             00003FF7

MPASM 03.90 Released                EXAMPLE.ASM   4-7-2005  16:01:23         PAGE  3


SYMBOL TABLE
  LABEL                             VALUE

_RC_OSC               00003FFF
_WDT_OFF              00003FFB
_WDT_ON               00003FFF
_XT_OSC               00003FFD
__16F84               00000001
main                  00000000
subrutina             00000004
```

```
MEMORY USAGE MAP ('X' = Used,  '-' = Unused)

0000 : XXXXXX---------- ---------------- ---------------- ----------------

All other memory blocks unused.

Program Memory Words Used:     6
Program Memory Words Free:  1018


Errors   :     0
Warnings :     1 reported,     0 suppressed
Messages :     1 reported,     0 suppressed
```

Lo primero que hará el programa será eliminar las líneas del fichero *.lst que no correspondan al programa. El archivo de ejemplo quedaría de la siguiente manera:

```
                        00001 ;
                        00002 ;       File: Example.asm
                        00003 ;       ejemplo que saca por el puerto A la secuencia 3,6,9,12,...
                        00004 ;
                        00005
                        00006         list     p=16f84a
                        00007         #include p16f84a.inc
                        00001         LIST
                        00002 ; P16F84A.INC  Standard Header File, Version 2.00    Microchip
Technology, Inc.
                        00134         LIST
                        00008
                        00009
                        00010
0000                    00011 main                      ; Main
0000  3000              00012         movlw   0
0001  3E03              00013         addlw   3
0002  2804              00014         goto    subrutina   ; salta a una subrutina de ejemplo
0003  2800              00015         goto    main        ; hace un bucle infinito
                        00016
0004                    00017 subrutina                 ; subrutina que escribe en el puerto
                        00018
0004  0085              00019         movwf   PORTA
0005  0008              00020         return
                        00021
                        00022
                        00023         end
```

La información de que se dispone en este paso se puede dividir en cuatro columnas, posición de memoria, código de instrucción, número de línea y línea.

La columna línea será la que se mostrará al usuario para que pueda saber en que líneas sitúa los puntos de interrupción.

Posición de memoria	Código de instrucción	Nº de línea	Línea
		00001	;
		00002	; File: Example.asm
		00003	; ejemplo que saca por el puerto A la secuencia 3,6,9,12,...
		00004	;
		00005	
		00006	
		00007	#include p16f84a.inc
		00001	LIST
		00002	; P16F84A.INC Standard Header File, Version 2.00 Microchip Technology, Inc.
		00134	LIST
		00008	
		00009	
		00010	
0000		00011	main ; Main
0000	3000	00012	**movlw** 0
0001	3E03	00013	**addlw** 3
0002	2804	00014	**goto** subrutina ; salta a una subrutina de ejemplo
0003	2800	00015	**goto** main ; hace un bucle infinito
		00016	
0004		00017	subrutina ; subrutina que escribe en el puerto
		00018	
0004	0085	00019	**movwf** PORTA
0005	0008	00020	**return**
		00021	
		00022	
		00023	**end**

Tabla 1. Contenido de los archivos *.lst de MPLab

En función de los datos que hay en las columnas podemos observar diferentes tipos de líneas:

- Líneas en blanco. Estas líneas tienen sólo comentarios, macros del compilador, etc. Ninguna de estas líneas corresponde a código ejecutable por el microcontrolador. El compilador no les asigna ni posición de memoria ni código de instrucción. En estas líneas no se podrán poner puntos de interrupción

- Etiquetas. En el ejemplo tenemos las etiquetas main y subrutina. Para estas líneas el compilador asigna una posición de memoria pero no les asigna ningún código de instrucción. En la primera línea con código ejecutable que aparezca después de la etiqueta podemos comprobar que aparece la misma posición de memoria que la que se ha asignado a la etiqueta. Por ejemplo subrutina tiene la posición de memoria 0004H, y en la posición de memoria 0004H tendremos

la instrucción 0085 (movwf PORTA). En estas líneas no se podrán poner puntos de interrupción.

- Instrucciones. El compilador les asigna una posición de memoria y un código de instrucción. En estas líneas se podrán poner puntos de interrupción.

Una vez que el usuario haya abierto el archivo *.lst y lo tenga visible en pantalla deberá tener la posibilidad de realizar las acciones que se detallarán a continuación.

Poner y quitar puntos de interrupción

El programa que se ha leído de MPLab se mostrará en pantalla y haciendo clic en las líneas deseadas se podrá poner y quitar puntos de interrupción

Guardar el documento

Desde el menú o la barra de herramientas se podrá guardar el programa en el formato de Emulpic (extensión emp) que contendrá además del código los puntos de interrupción situados por el usuario.

Enviar el programa al emulador

Las comunicaciones se realizarán a través del puerto serie configurado a 9600 baudios, protocolo RTS-CTS, 8 bits de datos, 1 bit de start y 1 bit de stop.

Iniciar la ejecución del programa

Desde Emulpic se podrá solicitar al emulador que inicie la ejecución del programa que se ha enviado previamente.

Detener la ejecución del programa

En todo momento se podrá detener la ejecución del programa en el emulador.

Pausar la ejecución del programa

Se podrá pausar la ejecución del programa para poder ver el contenido del emulador (memoria, registros,...)

Avanzar una instrucción
Permite avanzar la ejecución de una instrucción en el emulador y pausar su ejecución.

Visualizar el contenido de la memoria del emulador

Una vez se ha leído el contenido del emulador se puede visualizar el contenido de la memoria.

Visualizar el contenido de los registros del emulador

Cuando ya se dispone del contenido actualizado del emulador en Emulpic se puede visualizar el contenido de los registros.

Por motivos de limitación de tiempo no se incluirán las siguientes opciones:

- modificar el contenido de los registros
- modificar el contenido de la memoria
- modificar el contador de programa

2.2.2 ACTORES

Los actores que se han identificado son:

- usuario. Perteneciente al grupo de actores que se agrupan bajo el concepto de usuario final, o sea, personas que interactúan directamente con el sistema.

- Emulador. Perteneciente a entorno informático de la aplicación.

2.2.3 CASOS DE USO

2.2.3.1 Diagrama de casos de uso

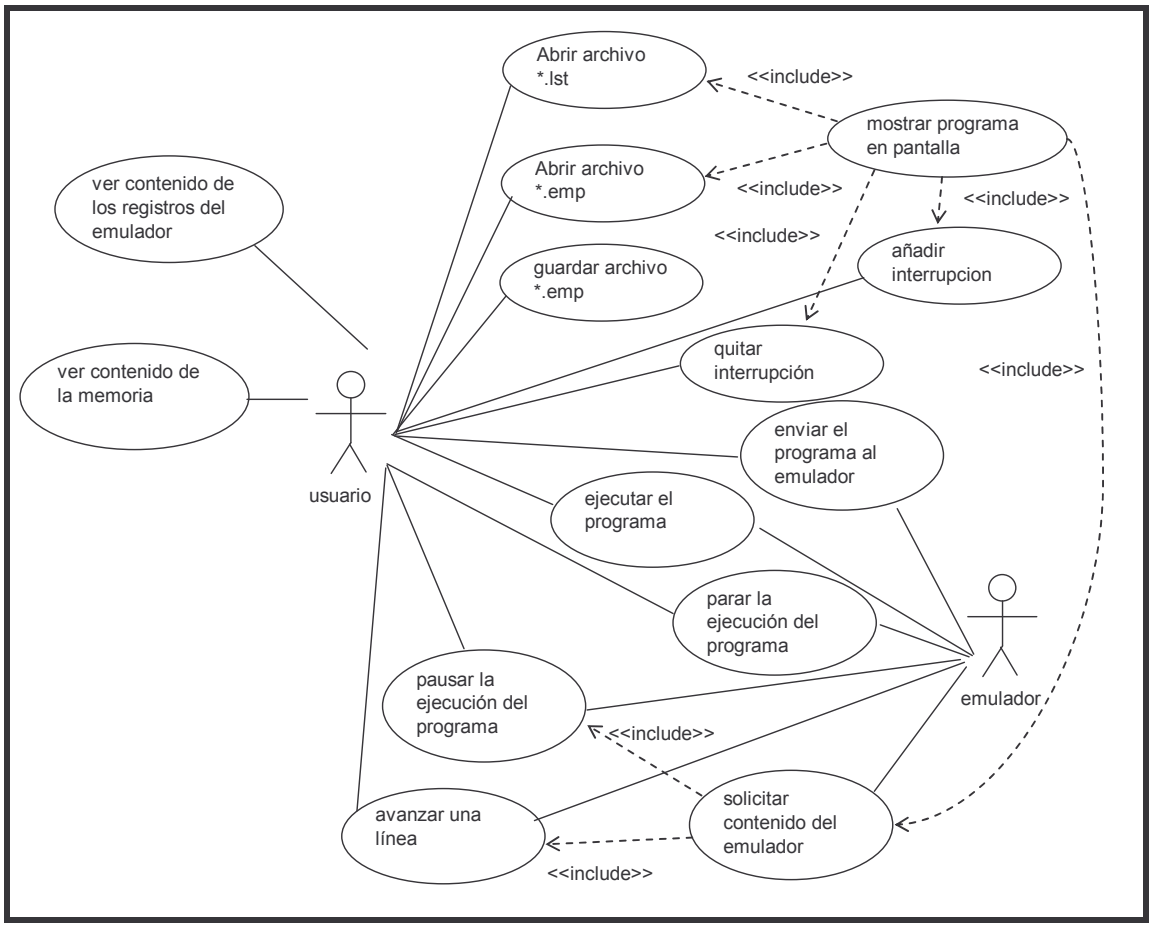


Ilustración 13. Diagrama de casos de uso

2.2.3.2 Documentación textual de los casos de uso

Abrir archivo *.lst

Resumen de la funcionalidad: abre los archivos lst que genera el compilador de Mplab.

Papel dentro del trabajo del usuario: para empezar a trabajar con el emulador el usuario debe abrir el archivo generado con el MPLab.

Actores: Usuario

Precondición: Se ha creado un programa en ensamblador con el MPLab y se ha compilado para obtener el archivo *.lst.

Postcondición: El programa se ha leído y guardado en las estructuras de datos correspondientes y está listo para ser mostrado en la pantalla.

Abrir archivo *.emp

Resumen de la funcionalidad: Abre archivos de proyectos creados y guardados anteriormente con el emulador.

Papel dentro del trabajo del usuario: Permite al usuario recuperar proyectos creados anteriormente.

Actores: Usuario

Precondición: se ha abierto anteriormente un nuevo proyecto, se han puesto puntos de interrupción y se ha guardado en el formato de Emulpic.

Postcondición: se ha cargado el programa en las estructuras de datos, incluyendo los puntos de interrupción, y está listo para ser mostrado en pantalla.

Mostrar programa en pantalla

Resumen de la funcionalidad: muestra el programa en el interface gráfico de la aplicación para que el usuario pueda ver en que líneas poner los puntos de interrupción.

Papel dentro del trabajo del usuario: este caso de uso está incluido dentro de otros casos de uso y por lo tanto se ejecuta cada vez que el usuario lanza uno de los casos indicados en el diagrama en los que este caso está incluido.

Actores: no tiene actor primario. Es un caso de uso incluido dentro de otros casos de uso tal como se muestra en el diagrama.

Precondición: se ha cargado el archivo *.lst o *.emp en las estructuras de datos

Postcondición: el programa está visible en el interface gráfico del usuario.

Guardar archivo *.emp

Resumen de la funcionalidad: guarda el programa leído de MPLab en el formato de Emulpic, que incluye los puntos de interrupción.

Papel dentro del trabajo del usuario: cuando el usuario ha abierto un archivo lst y ha trabajado con el poniendo puntos de interrupción y ejecutándolo en el emulador, puede guardar el programa junto con los puntos de interrupción.

Actores: Usuario

Precondición: se está trabajando en Emulpic con un programa importado de MPLab o guardado anteriormente con Emulpic.

Postcondición: se ha guardado el programa con el que se está trabajando incluyendo los puntos de interrupción.

Añadir interrupción

Resumen de funcionalidad: cuando el usuario hace clic sobre una línea de código fuente que no tiene punto de interrupción, se pone un punto de interrupción.

Papel dentro del trabajo del usuario: permite al usuario decidir en que líneas del programa se va a detener la ejecución del emulador para poder consultar el estado de sus registros, el contenido de memoria,...

Actor: Usuario

Precondición: hay un programa cargado y mostrado en el interface gráfico de la aplicación y se va a hacer clic sobre una línea sin punto de interrupción.

Postcondición: se ha puesto una nueva interrupción, modificándose la estructura de datos con ese nuevo punto de interrupción y visualizándolo en pantalla.

Quitar interrupción

Resumen de funcionalidad: cuando el usuario hace clic sobre una línea de código fuente que tiene punto de interrupción, se quita el punto de interrupción existente.

Papel dentro del trabajo del usuario: permite al usuario quitar un punto de interrupción que había puesto anteriormente en una línea para verificar el funcionamiento del programa.

Actor: Usuario.

Precondición: hay un programa cargado y mostrado en el interface gráfico de la aplicación y se va a hacer clic sobre una línea con punto de interrupción.

Postcondición: se ha quitado un punto de interrupción, modificándose la estructura de datos correspondiente y quitando el indicador del interface gráfico del programa.

Enviar el programa al emulador

Resumen de funcionalidad: envía el programa cargado en Emulpic al emulador.

Papel dentro del trabajo del usuario: para que el usuario pueda depurar el programa que ha creado con MPLab para el Pic 16F84 en el emulador antes debe enviarlo desde el programa Emulpic a través del puerto serie. El actor que lanza el caso de uso es el usuario que solicita enviar el programa al emulador. Si el emulador está preparado para establecer la comunicación responde al PC y se inicia el envío.

Actor: Usuario, Emulador

Precondición: Existe un programa cargado en Emulpic

Postcondición: El programa que hay en Emulpic se ha enviado al emulador.

Ejecutar el programa

Resumen de funcionalidad: una vez que se ha cargado el programa en el emulador el usuario debe iniciar su ejecución para poder depurar su funcionamiento. Si el programa está pausado se continúa la ejecución del programa donde se había detenido.

Papel dentro del trabajo del usuario: el usuario ejecuta la orden que envía este comando al emulador. El PC intenta establecer las comunicaciones serie y el emulador responde afirmativamente si está preparado para recibir la información en cuyo caso se envía la orden.

Actor: usuario, emulador

Precondición: existe un programa cargado en el emulador

Postcondición: si el emulador ha recibido el comando se ha iniciado la ejecución del programa

Parar la ejecución del programa

Resumen de funcionalidad: finaliza la ejecución del programa en el emulador. El emulador queda preparado para recibir otro programa o para volver a ejecutarlo.

Papel dentro del trabajo del usuario: el usuario ejecuta la orden que envía este comando al emulador. El PC intenta establecer las comunicaciones serie y el emulador responde afirmativamente si está preparado para recibir la información en cuyo caso se envía la orden.

Actor: usuario, emulador

Precondición: existe un programa en funcionamiento en el emulador.

Postcondición: si el emulador ha recibido el comando finaliza la ejecución del programa.

Pausar la ejecución del programa

Resumen de funcionalidad: hace una pausa en la ejecución del programa del emulador. El emulador queda preparado para continuar la ejecución donde lo había dejado o para avanzar una línea en la ejecución. En la ventana de Emulpic se indica la línea donde se ha pausado la ejecución del programa.

Papel dentro del trabajo del usuario: el usuario ejecuta la orden que envía este comando al emulador. El PC intenta establecer las comunicaciones serie y el emulador responde afirmativamente si está preparado para recibir la información en cuyo caso se envía la orden.

Actor: usuario, emulador

Precondición: existe un programa ejecutándose en el emulador.

Postcondición: si el emulador ha recibido la orden se pausa la ejecución del programa.

Avanzar una línea

Resumen de funcionalidad: con el emulador pausado el usuario puede solicitar que ejecute la siguiente línea de código y que vuelva a ponerse en pausa.

Papel dentro del trabajo del usuario: el usuario ejecuta la orden que envía este comando al emulador. El PC intenta establecer las comunicaciones serie y el emulador responde afirmativamente si está preparado para recibir la información en cuyo caso se envía la orden.

Actor: usuario, emulador

Precondición: existe un programa ejecutándose en el emulador y está en estado de pausa.

Postcondición: si el emulador ha recibido el comando ha ejecutado la siguiente línea del programa y ha vuelto a situarse en estado de pausa.

Solicitar el contenido del emulador

Resumen de funcionalidad: cuando el emulador se pone en estado de pausa inmediatamente después Emulpic le solicita el contenido de sus registros y memoria para poder dar esta información al usuario y para saber en que línea se ha interrumpido la comunicación y poder indicarlo en el código fuente.

Papel dentro del trabajo del usuario: cuando el usuario ha solicitado pausar el emulador o avanzar una línea en la ejecución del programa el usuario necesitará saber

en que línea se ha detenido el programa y consultará el contenido de los registros o la memoria. Para poder obtener esta información se debe solicitar al emulador que envíe toda la información sobre su estado.

Actor: este caso no tiene un actor principal. Es un caso de uso incluido dentro de los casos de uso pausar el emulador y avanzar una línea la ejecución del programa. El emulador es un actor no principal en este caso.

Precondición: existe un programa ejecutándose en el emulador y está en estado de pausa.

Postcondición: si el emulador ha recibido la orden ha enviado la información solicitada al PC y se mantiene en estado de pausa.

Ver el contenido de los registros del emulador

Resumen de funcionalidad: muestra el contenido de los registros del emulador

Papel dentro del trabajo del usuario: para poder comprobar el funcionamiento del programa es imprescindible para el usuario poder ver que valores toman los registros del microcontrolador.

Actor: Usuario

Precondición: Existe un programa cargado en el emulador, el usuario ha pausado la ejecución y ha solicitado que se le envíe la información del emulador.

Postcondición: se ha mostrado en la pantalla el contenido de los registros del microcontrolador.

Ver el contenido de la memoria del programa

Resumen de funcionalidad: muestra el contenido de la memoria del microcontrolador.

Papel dentro del trabajo del usuario: permite al usuario ver el contenido de la memoria para poder consultar el contenido de las variables.

Actor: Usuario

Precondición: existe un programa cargado en el emulador, el usuario ha pausado la ejecución y ha solicitado que se le envíe la información del emulador.

Postcondición: se ha mostrado en la pantalla el contenido de la memoria del microcontrolador.

2.2.4 INTERFACE GRÁFICO DE LA APLICACIÓN. CROQUIS

En la ilustración siguiente podemos ver un croquis del programa Emulpic.

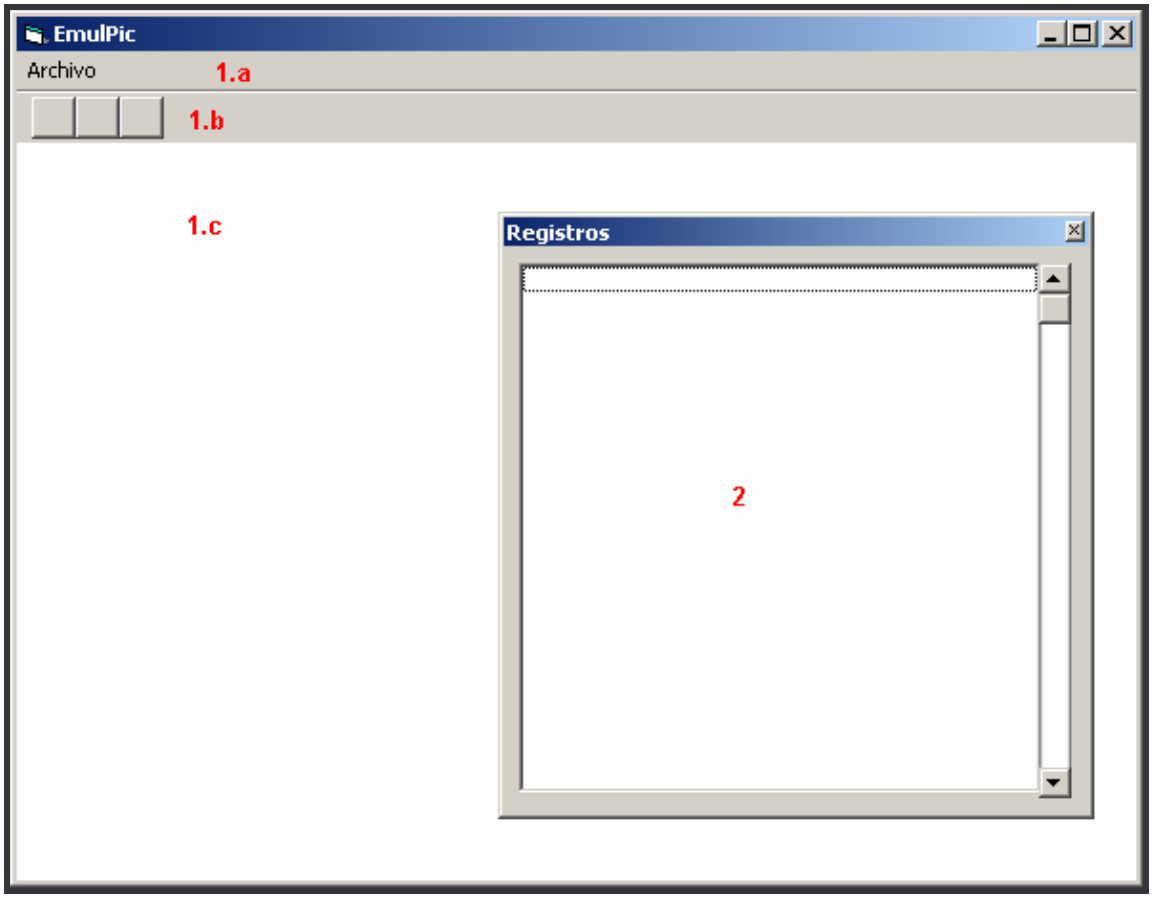


Ilustración 14. Croquis de la aplicación

En el se pueden distinguir los siguientes componentes:

1. Ventana principal de la aplicación. Será la ***clase frontera CMainFrame***. A priori se identifican los siguientes componentes:

    a. Menú
    b. Barra de herramientas
    c. Presentación del código fuente. En esta zona de la ventana se mostrará toda la información de la columna "Línea" de todas las líneas del programa. Cuando el usuario haga doble clic en una línea del programa se activarán o desactivarán los puntos de interrupción. Las líneas con puntos de interrupción aparecerán resaltadas. El programa sólo permitirá poner puntos de interrupción sobre las líneas que

correspondan a código ejecutable, o sea aquellas que disponen de código de instrucción.

Esta clase, además de ser la ventana principal de la aplicación, también será la clase principal de la aplicación, desde la que se gestionará todo su funcionamiento.

2. Ventanas para mostrar el estado del emulador del microcontrolador (memoria y registros). Cuando la ejecución del programa en el emulador este en pausa porque se ha encontrado un punto de interrupción o se está realizando una ejecución "paso a paso", el usuario podrá abrir diversas ventanas en las que se mostrara el contenido de los registros, de la memoria, etc. Se dispondrán de las siguientes ventanas:

   a. Visualización del contenido de la memoria. Será la ***clase frontera FrmMemoria***. Esta ventana presentará la información de la memoria de la siguiente manera

00000	00	00	00	00	00	00	00	00
00008	0A	0A	01	11	FF	00	00	00
00010	00	00	00	00	00	00	00	00
...								

   En cada fila se presentarán 8 posiciones de memoria consecutivas. La primera celda de la fila indicará en que posición de memoria empieza la segunda celda de la fila. En el ejemplo anterior tenemos que las posiciones de memoria que se muestran en la segunda fila corresponden a:

Posición de memoria	contenido
00008	0A
00009	0A
0000A	01
0000B	11
0000C	FF
0000D	00
0000E	00
0000F	00

   Se podrá elegir si se desea ver el contenido de la memoria en formato binario, hexadecimal o decimal.

   b. Visualización del contenido de los registros. Será la ***clase frontera FrmRegistros***. Esta clase permitirá ver los registros dentro de un grid similar a la siguiente tabla:

Nombre del registro	Valor
TMR0	00000000
PCL	00100010
STATUS	01010010
FSR	00000111
...	...

El valor de los registros se podrá visualizar en formato binario, decimal y hexadecimal.

2.2.5 DIAGRAMAS DE SECUENCIA.

Los diagramas de secuencia que se presentan a continuación tienen como objetivo analizar los diferentes casos de usos definidos desde un punto de vista de las ventanas de la aplicación identificadas.

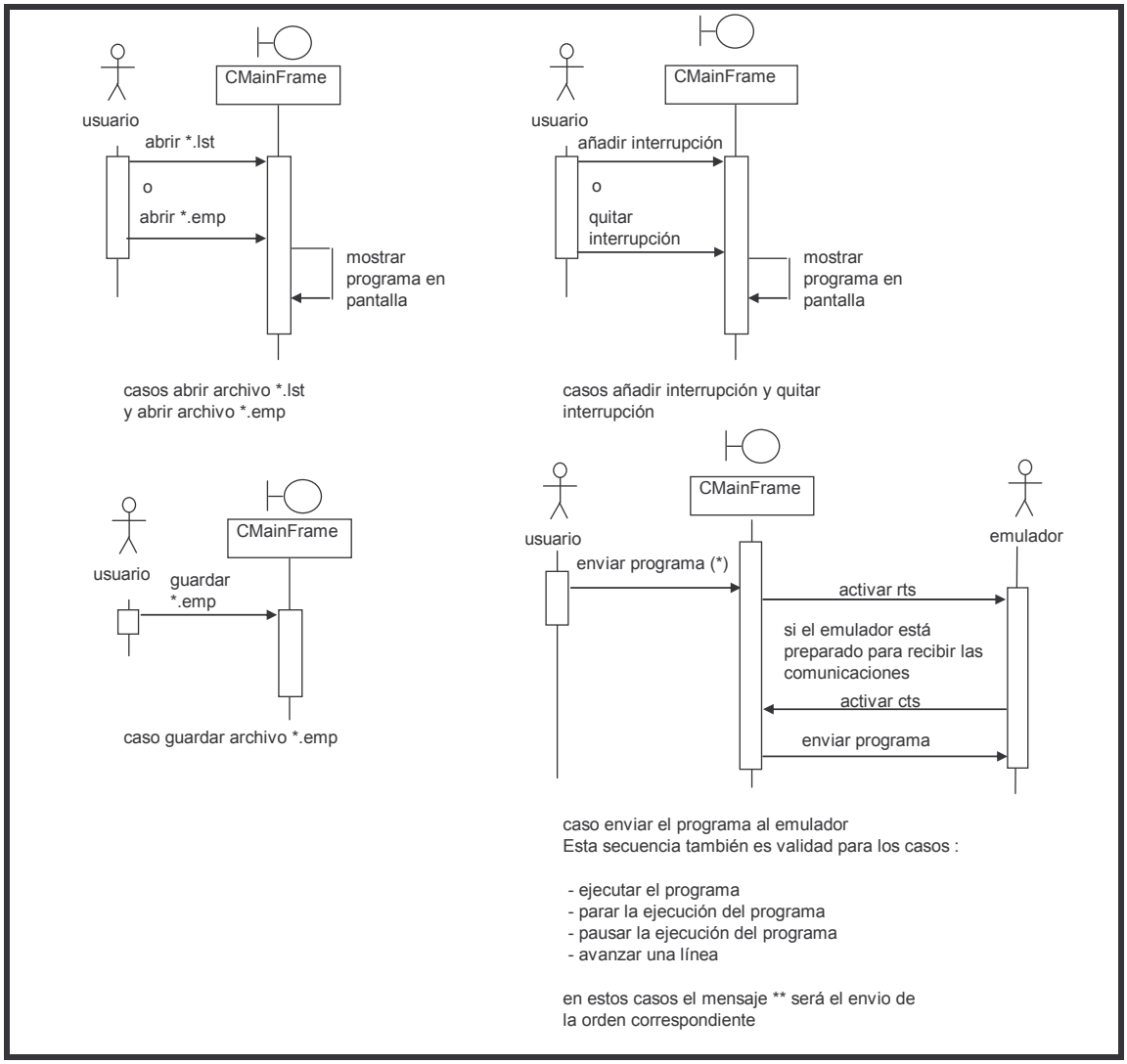


Ilustración 15. Diagramas de secuencia. Parte 1

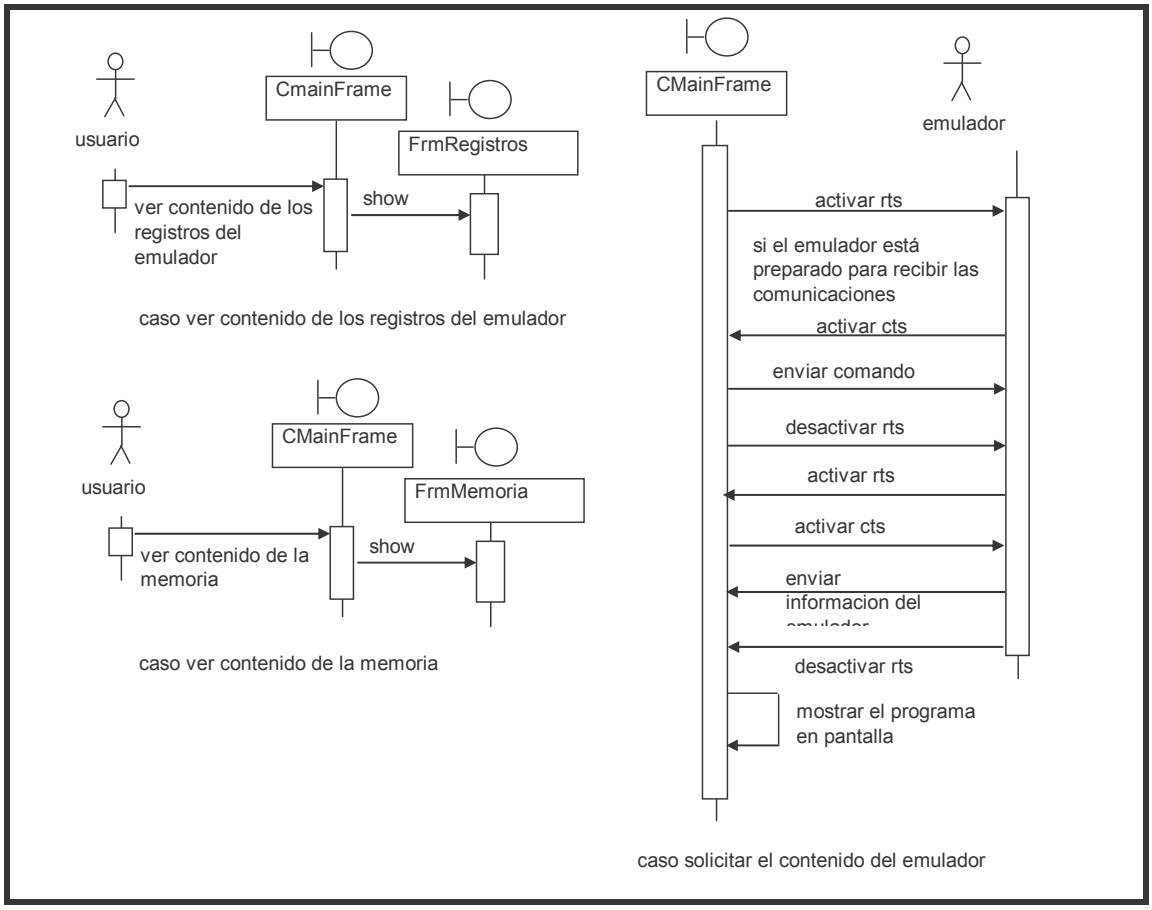


Ilustración 16. Diagramas de secuencia. Parte 2

2.2.6 DIAGRAMA DE ESTADOS DEL FUNCIONAMIENTO DE LA APLICACIÓN

El siguiente diagrama de estados se ha realizado para ilustrar los diferentes estados en que se encontrará la aplicación según las acciones que realice el usuario. Servirá de ayuda a la hora de diseñar los sistemas de menús y de barras de herramientas ya que nos ayudará a saber que comandos del menú y que botones pueden estar activos en cada momento.

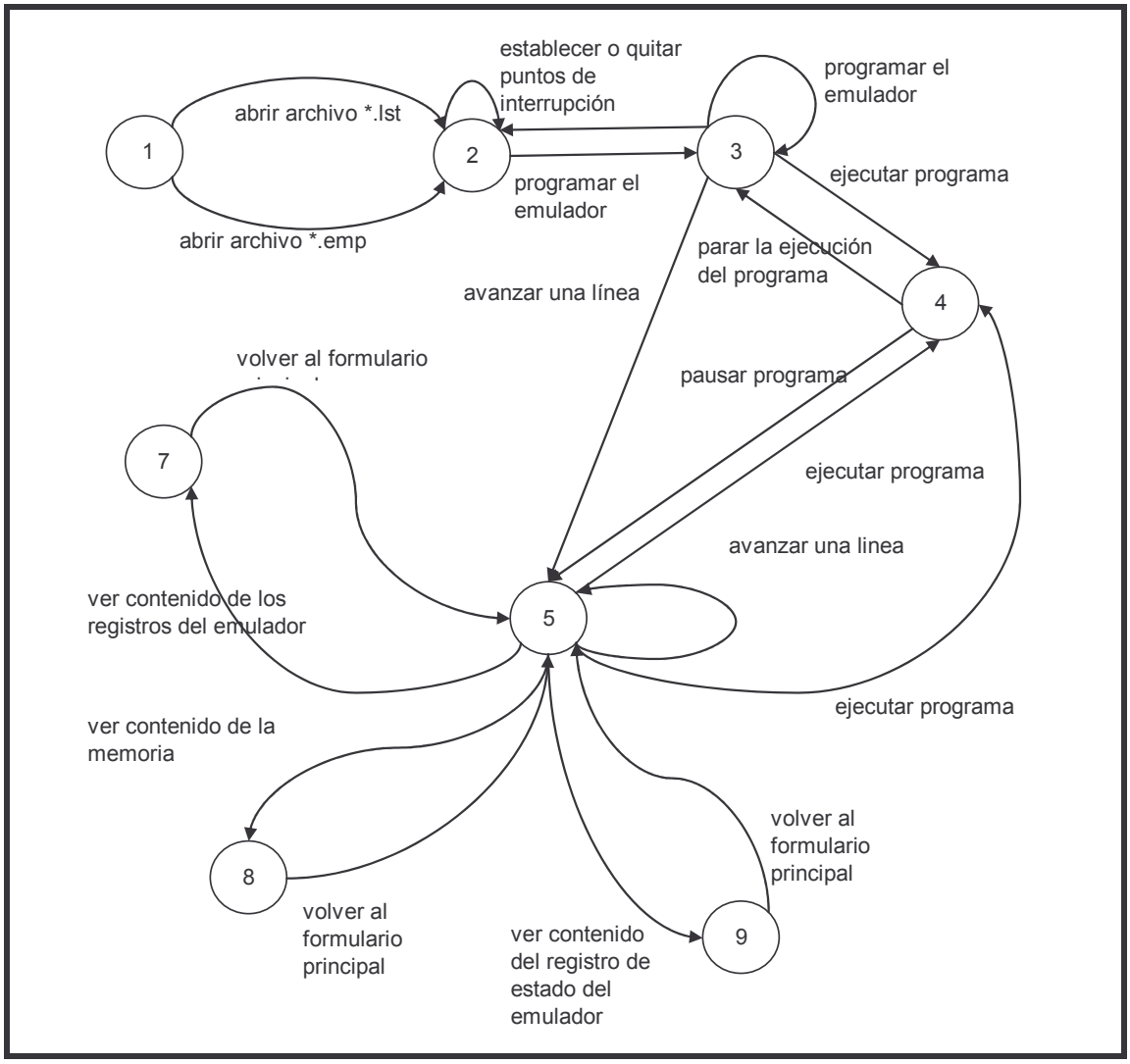


Ilustración 17. Diagrama de estados de la aplicación

El estado inicial de la aplicación es el estado 1. En el momento en que el usuario abre un archivo la aplicación pasa al estado, desde el cual se podrán poner o quitar puntos de interrupción y programar el emulador. Cuando se haya programado el emulador la aplicación estará en el estado 3. Desde este estado se podrá iniciar la ejecución continua del programa o avanzar una línea en la ejecución (ejecución paso a paso). Si se pone un punto de interrupción mientras el programa está en el estado 3 se pasa al estado2.

Si se ha ejecutado el programa en el emulador Emulpic pasa al estado 4. Desde este estado podremos parar la ejecución del programa, lo que nos devolverá al estado 3, o podremos pausar el programa, lo que nos llevará al estado 5.

Desde el estado 5 podremos consultar el contenido de los registros y la memoria del emulador, ya que al pausar la ejecución el emulador habrá enviado a Emulpic su contenido.

2.3 ANÁLISIS Y DISEÑO

2.3.1 DIAGRAMA DE CLASES

A continuación se muestra el diagrama de las clases de la aplicación dentro del marco de clases de la MFC utilizadas. Para ver correctamente el diagrama se recomienda un zoom del 150%.

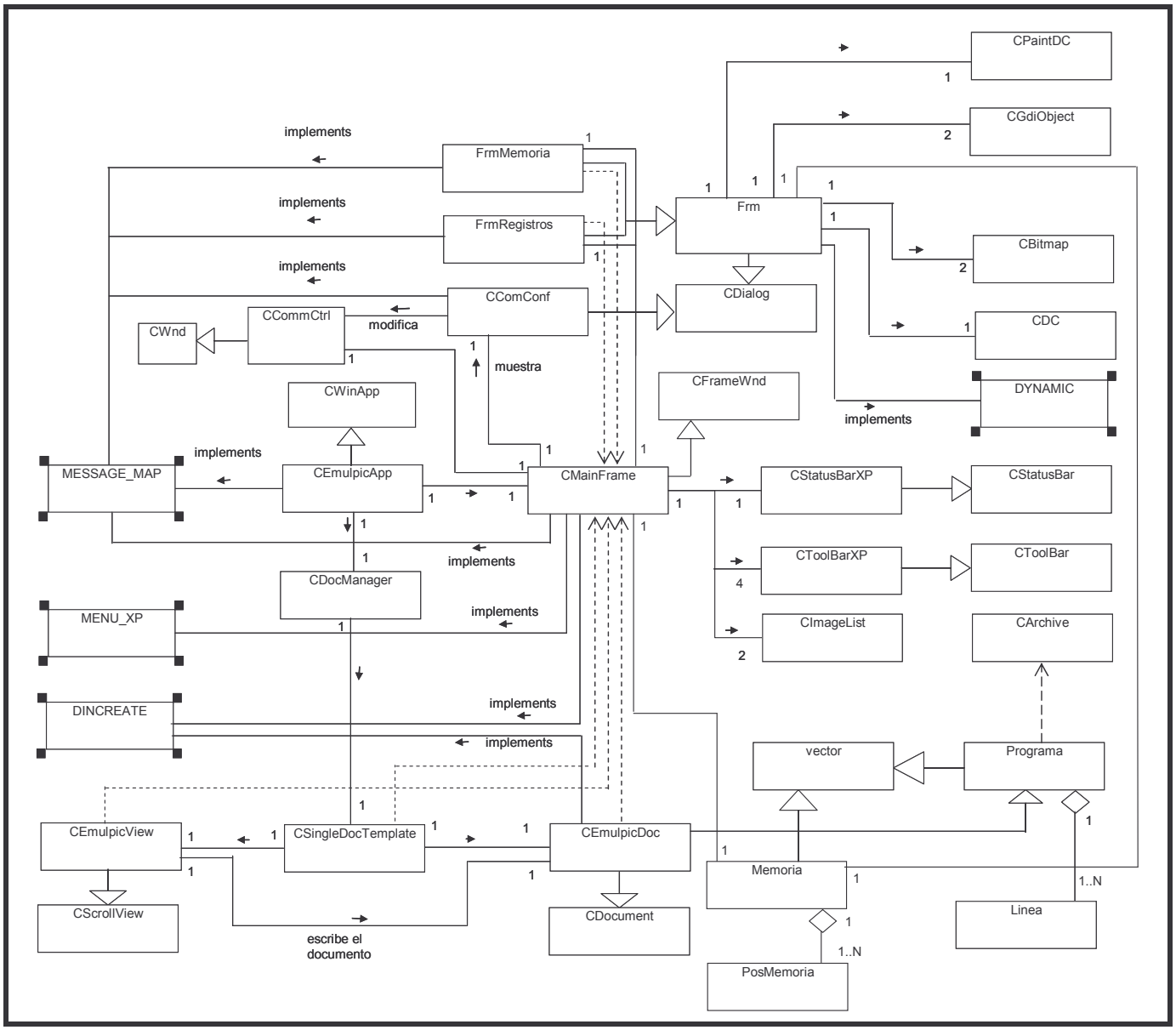


Ilustración 18. Diagrama de clases

La clase principal de la aplicación, desde la que se inicia el programa, es la clase CEmulpicApp. Esta clase debe heredar de la clase CWinApp de la biblioteca de clases de Microsoft MFC (Microsoft Foundation Clases) y debe implementar el interface MESSAGE_MAP para poder procesar mensajes de Windows (eventos).

CEmulpicApp, a través de su superclase CWinApp dispone de un atributo de la clase MFC CSingleDocTemplate que es la que permitirá implementar la arquitectura documento-vista de la aplicación. También dispone de un atributo de la clase CMainFrame, ventana principal de la aplicación.

La clase MFC CSingleDocTemplate vincula la clase de documento CEmulpicDoc, la clase vista CEmulpicView y la ventana donde se mostrará la vista CMainFrame.

La clase CMainFrame será la ventana principal de la aplicación y debe heredar de la clase MFC CFrameWnd e implementar los interfaces MESSAGE_MAP para procesar mensajes de Windows, MENU_XP para dar un aspecto de XP a los menús de la ventana y DINCREATE para que la arquitectura documento-vista pueda crear dinámicamente la ventana. La clase CMainFrame tiene:

- cuatro instancias de la clase CToolBarXP para las barras de herramientas
- una instancia de la clase CStatusBar para la barra de estado
- dos instancias de la clase CImageList para los iconos que se mostrarán en los botones de la barra de estado, una para los botones activados y una para los botones desactivados
- una instancia de la clase CComCtrl que será la encargada de las comunicaciones serie
- Una instancia de la clase FrmRegistros, ventana donde se mostrarán los registros del procesador
- Una instancia de la clase FrmMemoria, ventana donde se mostrará el contenido de las memorias del procesador
- Cinco instancias de la clase Memoria. Estos cinco objetos contendrán los datos de los registros de los bancos de registros y de las memorias del emulador.

Los programas que se enviarán al emulador estarán encapsulados por la clase Programa. Esta clase será un contenedor de objetos de la clase Linea. Cada objeto de la clase Linea corresponderá a una línea del programa que se enviará al emulador. Para actuar como tal contenedor Programa heredará de la clase estándar de la biblioteca de C++ Vector. Programa también se encargará de leer los archivos con el programa con la colaboración de la clase de la MFC CArchive

Para poder implementar la arquitectura documento-vista y permitir a la aplicación mostrar el programa guardado en Programa se ha definido la clase CEmulpicDoc que hereda de Programa y de CDocument.

Las clases CStatusBarXP y CToolbarXP son clases de una librería descargada de Internet para ofrecer un aspecto de XP a la aplicación.

Las clases FrmMemoria y FrmRegistros tienen características comunes por lo que he creado, por generalización, una clase Frm que posee los atributos y métodos comunes a ambas clases. Frm heredará de la clase MFC CDialog, ya que FrmMemoria y FrmRegistros serán dos ventanas tipo cuadro de dialogo.

Los registros y la memoria se presentarán en FrmMemoria y FrmRegistros sobre un bitmap y las operaciones gráficas necesarias para pintar sobre estos bitmap se realizarán con ayuda de las clases MFC CPainDC, CGDiObject, CDC y CBitmap de Frm. Para poder mostrar los datos de los registros o de la memoria Frm dispondrá de un puntero a la clase Memoria que contiene los datos que debe mostrar.

La clase Memoria es un contenedor de objetos de la clase PosMemoria y hereda de la clase vector.

La clase PosMemoria representa una posición de memoria del emulador, ya sea de un registro o de una posición de memoria y contiene su posición de memoria y su valor.

2.3.2 IDENTIFICACIÓN DE ESTRUCTURAS DE DATOS. ANÁLISIS DE LAS CLASES DE ENTIDAD

El programa trabajará con la siguiente información:

- líneas del archivo *.lst que se lee de MPLab
- Datos de la memoria y los registros del emulador

2.3.2.1 Clase Linea

En la descripción que se ha dado en el apartado de explicación textual de las especificaciones del programa se ha ilustrado la estructura de una línea del archivo *.lst. Hemos podido ver que esta líneas poseen la siguiente información:

- posición de memoria
- código de instrucción
- número de línea
- texto del código fuente

De esta información el número de línea no nos interesa ya que no aporta ninguna información útil al emulador.

Además de estos datos el usuario puede poner y quitar puntos de interrupción en las líneas del programa.

Con esta información podemos definir la clase Linea, que tendrá los siguientes atributos y métodos:

Linea
-PosMem: string -CodInst: string -Line: string -Interr: bolean
+ Linea (pm:string, ci: string,l:string) +~Linea() +GetPosMem( ):string +GetCodInst( ):string +GetLinea( ):string +GetInt( ):booleano +SetInt( ):booleano

El atributo PosMem contendrá el dato correspondiente a Posición de memoria. Si la línea no contiene una instrucción PosMem valdrá "".

El atributo CodInst contendrá el dato correspondiente a la columna Código de instrucción. Si la línea no es una instrucción (p.e. un comentario) CodInst deberá valer ""

El atributo Line contendrá el texto de la línea. Este dato se mostrará en la pantalla al usuario.

El atributo Interr permite saber si el usuario ha situado un punto de interrupción en esta línea.

En el constructor de clase recibirá como parámetros los que corresponde a PosMem, CodInst y Linea. El atributo Interr estará inicialmente a false.

El destructor de clase ~Linea liberará el espacio asignado dinámicamente a los diferentes atributos del tipo string de la clase.

Se han creado accesores de lectura para los atributos PosMem, CodInst y Linea. Estos atributos no necesitan accesores de lectura ya que estos atributos sólo deben ser asignados una vez en el momento de crear la clase.

Para el atributo Interr se creará un accesor de lectura, GetInt, y uno de escritura, SetInt. Como el usuario solo podrá poner puntos de interrupción sobre líneas que tengan código ejecutable, o sea aquellas que tengan código de instrucción, el funcionamiento de SetInt será el siguiente:

SetInt

{Pre: la instancia de la clase Linea está correctamente inicializada}

```
parámetros:
ninguno

si CodInst <> "" entonces
    Interr = ¬Interr
    return true
sino
    return false
fsi
```

{Post: si la instancia de la clase línea correspondía a una línea con código ejecutable y no había un punto de interrupción lo pone y devuelve true, si había punto de interrupción lo quita y devuelve true. Si la instancia de la clase Linea no correspondía a una línea con código ejecutable devuelve false}

2.3.2.2 Clase Programa

Como un programa es un conjunto de líneas de código fuente necesitamos una colección de objetos Linea para almacenar el programa. Esta colección la tendremos en la clase Programa.

Para elegir el tipo de contenedor del que heredará la clase Programa debemos saber que:

- los objetos Linea se insertarán secuencialmente y se mantendrán en el mismo orden en que se han insertado. Deberá ser un contenedor no ordenado. Solo necesita que se inserten elementos al final del contenedor.

- para presentar el programa en la ventana de EmulPic se accederá secuencialmente a todas las posiciones del contenedor. Deberá ser un contenedor recorrible.

- Para poner y quitar puntos de interrupción se accederá a un elemento concreto a partir de su posición en el contenedor. El contenedor deberá permitir acceso aleatorio a sus elementos.

- No son necesarios algoritmos de ordenación.

Como el programa se está realizando en C++ .NET dispongo de la librería estándar de C++. De todos los tipos de contenedores incluidos en esta librería la clase Vector es la que mejor se adapta a las necesidades del programa:

- inserción de elementos al final del contenedor con tiempo constante.

- permite acceder de forma rápida a cualquier elemento del contenedor

Así pues, diseñaré la clase Programa para que herede de Vector.

Programa <<extends Vector>>
+Programa( ) +LoadLst(ar:CArchive):integer +Load(ar:CArchive):integer +Save(ar:CArchive):integer
Linea

A los atributos y métodos que hereda de Vector le añadiremos los métodos LoadLst, Load y Save.

El método LoadLst leerá el archivo *.lst que genera el MPLab. Recibe como parámetro un objeto de la clase CArchive que corresponde a la clase que manipula el archivo que se va a leer. Si la lectura del archivo ha sido correcta devuelve 1. Si la lectura del archivo ha sido incorrecta devuelve un número de error negativo.

```
LoadLst
{Pre:la instancia de la clase Programa está creada}
parámetros
archivo:string
variables:
cadena:string
f:fichero

si existe el fichero cuyo nombre está en el parámetro archivo
        borrar el contenido del contenedor
        f.open()
        mientras ¬f.eof()
                cadena = f.leerLinea()
                si cadena tiene un número de línea en la columna esperada
                        leer de cadena la posición de memoria, el código de instrucción y
                        la instrucción
                        crear una nueva instancia de la clase Linea y añadirla al
                        contenedor
                fsi
                f.avanza()
        fmientras
        f.close()
        return 1;
sino
        return -1;
```

```
fsi

{Post:
si no existía el fichero devuelve –1
si el fichero se ha leído correctamente el contenedor se ha llenado con las líneas del programa y devuelve 1
}
```

Los métodos Load y Save permiten abrir guardar los archivos abiertos en el formato de EmulPic. Reciben como parámetro un objeto de la clase CArchive que es el objeto que manipula el archivo con el que se va a trabajar.

El tipo de archivo de emulpic contendrá información sobre los puntos de interrupción que ha situado el usuario. Los archivos de EmulPic tendrán la extensión emp y su formato será:

<interrupción>,<posición de memoria>,<código de instrucción>,<línea 1>
<interrupción>,<posición de memoria>,<código de instrucción>,<línea 2>
...
<interrupción>,<posición de memoria>,<código de instrucción>,<línea n>

El métod Load leerá los archivos *.emp. Si la lectura del archivo ha sido correcta devolverá 1, si ha sido incorrecta devolverá un número de error negativo.

```
Load
{Pre:la instancia de la clase Programa está creada}
parámetros
archivo:string

variables:
cadena:string
f:fichero

si existe el fichero cuyo nombre está en el parámetro archivo
        borrar el contenido del contenedor
        f.open()
        mientras ¬f.eof()
                cadena = f.leeLinea()
                leer de cadena la posición de memoria, el código de instrucción, la instrucción y el punto de interrupción
                crear una nueva instancia de la clase Linea y añadirla al contenedor
                f.avanza()
        fmientras
        f.close()
        return 1;
sino
        return -1;
fsi
```

{Post:
si no existía el fichero devuelve –1
si el fichero se ha leído correctamente el contenedor se ha llenado con las líneas del programa y devuelve 1
}
El método Save guardará los archivos *.emp. Si la escritura del archivo ha sido correcta devolverá 1, si ha sido incorrecta devolverá un número de error negativo.

```
Save
{Pre:la instancia de la clase Programa está creada}
parámetros
archivo:string

variables:
cadena:string
cadena1:string
cadena2:string
cadena3:string
cadena4:string
ln:Linea
f:fichero

si existe el fichero cuyo nombre está en el parámetro archivo
        preguntar si se desea borrar el fichero
        si se desea borrar
                borrar el fichero
        sino
                devolver –1 y salir
        fsi
fsi

f.open()
posicionar el contenedor en el primer elemento
ln=f.leeLinea()
mientras ¬EOF(contenedor)
        cadena1:ln.getPosmem()
        cadena2:ln.getCodInst()
        cadena3:ln.getLinea()
        cadena4:ln.getInt()
        cadena=cadena1+","+cadena2+","+cadena3+","+cadena4
        f.escribe(cadena)
        avanzar el punto de interés del contenedor
fmientras
cerrar el fichero

return 1;
```

{Post:
si no se deseea sobrescribir el fichero existente devuelve devuelve –1 y sale del método
si el fichero se ha creado correctamente devuelve 1
}

2.3.2.3 Clase PosMemoria

Las clases Linea y Programa eran las clases para manipular los datos que Emulpic utiliza para enviar el programa al emulador.

Las clases PosMemoria y Memoria serán las que Emulpic utilizará para manipular la información que obtiene del emulador y que debe presentar en pantalla.

La clase PosMemoria encapsulará el par de valores (posición de memoria, contenido de la posición de memoria) que corresponderán a una posición de memoria del emulador

PosMemoria
-posicion:int -valor:byte
+PosMemoria(pos:int,val:byte) +getPos( ):int +getVal( ):byte

Los atributos de la clase son:

- posicion. Contiene la posición de memoria y es de tipo entero.
- Valor. Contiene el dato contenido en la posición de memoria. Valor será de tipo byte ya que es el tamaño de las posiciones de memoria del PIC 16F84

Los métodos de la clase son:

- PosMemoria(pos:int, val:byte). Es el constructor de la clase y recibe los parámetros correspondientes a la dirección de memoria y el valor contenido en ella.
- GetDireccion. Accesor de lectura al atributo posicion. Devuelve un tipo int.
- GetValor. Accesor de lectura al atributo valor. Devuelve un tipo byte.

2.3.2.4 Clase Memoria

La clase Memoria será el contenedor de objetos de la clase PosMemoria y se encargará de almacenar en la memoria del programa Emulpic el contenido de la memoria del emulador y de los registros. Las características que debe tener esta clase son prácticamente las mismas que las de la clase Programa:

- los objetos Posmemoria se insertarán secuencialmente y se mantendrán en el mismo orden en que se han insertado. Deberá ser un contenedor no ordenado. Solo necesita que se inserten elementos al final del contenedor.

- para presentar el contenido en las ventanas del programa se accederá secuencialmente a todas las posiciones del contenedor. Deberá ser un contenedor recorrible.

- No son necesarios algoritmos de ordenación.

Elijo el contenedor vector de la librería estándar de C++ porqué:

- la inserción de elementos al final del contenedor es de tiempo constante.

- permite acceder de forma rápida a cualquier elemento del contenedor

La clase memoria no añadirá ningún método ni atributo a la clase vector, pero crearé la clase para poder especializarla en el almacenamiento de objetos de la clase PosMemoria.

Memoria
PosMemoria

2.3.3 CLASES DE CONTROL

2.3.3.1 Clase principal de la aplicación. Clase CEmulPicApp

CEmulpicApp es la clase principal de la aplicación y su punto de entrada. Heredará de CWinApp y su diagrama UML es:

CEmulpicApp <<extends CWinApp>>
+EmulpicApp() +InitInstance():booleano +ExitInstance():int +OnFileOpen() +OnAppAbout()

Metodos

El método EmulpicApp es el constructor de clase,

Los métodos InitInstance y ExitInstance son reemplazos por especialización de los métodos de la superclase.

En el método InitInstance se realizan inicializaciones estándar de las aplicaciones de Windows. La mayoría de las inicializaciones las realiza automáticamente el entorno de programación .NET cuando se crea un nuevo proyecto. Entre estas inicializaciones encontramos la inicialización de bibliotecas OLE, de la librería de controles Commctl32 y el procesamiento de la línea de comandos. Además de las inicializaciones estándar añadiré las siguientes acciones:

- activar la visualización del estilo XP de los menús. Para ello se debe llamar al método InitializeHook de la clase CMenuXP.
- Crear una entrada en el registro para la aplicación
- Cargar la lista de los cuatro últimos archivos abiertos
- Inicializar la arquitectura documento/vista de la aplicación

InitInstance devuelve true si la inicialización ha sido correcta y false si ha fallado

En el método ExitInstance sólo se deberá desactivar la presentación de los menús con el estilo XP y llamar a ExitInstance de la superclase.

El método OnFileOpen es un reemplazo por especialización del método de la superclase. Este método se ejecuta cuando el usuario selecciona el comando Abrir. En este método personalizaré el cuadro de dialogo de abrir para que permita seleccionar archivos con la extensión *.emu, *.lst y *.* .Cuando el usuario haya seleccionado el archivo que quiere abrir se debe llamar al OpenDocumentFile de la superclase.

El métod OnAppAbout es un reemplazo por especialización del método de la superclase. En el mostraré el cuadro de diálogo acerca de, que esta definido por la clase AboutDlg.

2.3.3.2 Gestión de las comunicaciones serie. Clase CComCtrl

Para implementar las comunicaciones serie se dispone de las siguientes posibilidades

- crear una clase que gestione las comunicaciones serie desde el principio.

- buscar una librería de clases que disponga de la clase que necesitamos.

- utilizar el control OCX de comunicaciones serie MSComm.

De estas tres opciones he elegido la tercera ya que es la que me ofrece unos mejores resultados con una mínima inversión de tiempo. La implementación de la clase la realiza automáticamente el entorno de desarrollo .NET. Para implementar esta clase sólo es necesario seleccionar el menú Project → AddClass y seleccionar "MFC Class From ActiveX Control":

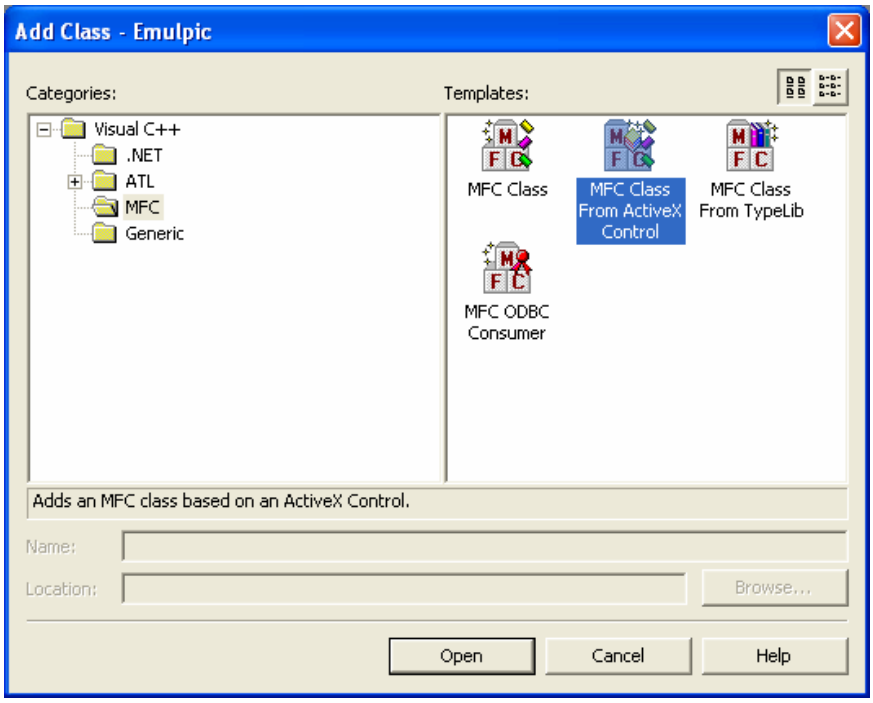


Ilustración 19. Creación de la clase de comunicaciones

Una vez que .NET haya creado la clase para utilizarla sólo se tiene realizar los siguientes pasos:

- crear una instancia de esta clase en una clase que herede de CWnd.
- Añadir los métodos necesarios para la gestión de los eventos del control Active X de comunicaciones

La clase de comunicaciones que crearé a partir del control Active X MSComm se llamará CComctrl. Los métodos que genera .NET para esta clase son:

CComctrl <<CWnd>>
+GetClsid():CLSID +Create(LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID, CCreateContext* pContext):booleano +Create(LPCTSTR lpszWindowName, DWORD dwStyle, const RECT& rect, CWnd* pParentWnd, UINT nID, CFile* pPersist,

```
        BOOL bStorage = FALSE,
        BSTR bstrLicKey):booelano
+GetCDHolding():boolenao
+SetCDHolding(booleano):void
+GetCDTimeout():long
+SetCDTimeout(long):long
+GetCommID():short
+SetCommID(short):void
+GetCommPort():short
+SetCommPort(short):void
+GetCTSHolding():booleano
+SetCTSHolding(booleano):void
+GetCTSTimeout():long
+SetCTSTimeout(long):void
+GetDSRHolding():booleano
+SetDSRHolding(booleano):void
+GetDSRTimeout():long
+SetDSRTimeout(long):void
+GetDTREnable():booleano
+SetDTREnable(booleano):void
+GetHandshaking():long
+SetHandshaking(long):void
+GetInBufferSize():short
+SetInBufferSize(short):void
+GetInBufferCount():short
+SetInBufferCount(short):void
+GetBreak():booleano
+SetBreak(booleano):void
+GetInputLen():short
+SetInputLen(short):void
+GetInterval():long
+SetInterval(long):void
+GetNullDiscard()booleano
```

Las funciones de estos métodos son:

- Métodos Create. La clase proporciona dos métodos sobrecargados para la creación de la clase. Estos métodos crean el control active X y lo vinculan a esta clase y crean la clase vinculándola a la ventana en la que se incrustara.
- Métodos Get y Set. Son métodos que acceden a las propiedades que corresponden en el control Active X MSComm.

Para ilustrar el funcionamiento de la clase podemos ver la implementación que hace .NET del método GetCommPort que obtiene el valor de la propiedad port del control MSComm:

```
short CCommCtrl::GetCommPort()
{
      short result;
      GetProperty(0x4, VT_I2, (void*)&result);
      return result;
}
```

Como se puede ver lo único que hace el método es consultar el valor de la propiedad mediante el método GetProperty que hereda de CWnd. Los parámetros que se le pasan a GetProperty son, en el mismo orden en que aparecen:

- identificador de la propiedad a recuperar
- identificador del tipo de datos de la propiedad
- puntero a variable en la que se devolverá el valor de la propiedad.

El resto de los métodos de la clase que se comunican con el control Active X funcionan igual que este método. Más información sobre esta clase la podemos encontrar consultado en la ayuda de Microsoft.

2.3.4 INTERFACE GRÁFICO DE LA APLICACIÓN. ANÁLISIS DE LAS CLASES FRONTERA

2.3.4.1 Generalidades

El diseño del interface gráfico de la aplicación se va a realizar con la intención de darle una apariencia totalmente profesional por lo que se va a prestar una gran atención en los detalles típicos de aplicaciones de Windows. Se pretende conseguir:

- Un aspecto de la aplicación actual. Para ello se dotará al programa de un sistema de menús y de barras de herramientas como el estándar de Windows XP

- Un sistema de menús con los elementos más típicos de las aplicaciones Windows (Archivo, Edición, Ver…)

- Presentar en el programa la lista de documentos más recientes

- El formato de archivos de Emulpic quedará registrado en el S.O. y permitirá que, haciendo doble clic sobre un archivo *.emu, se abra Emulpic y el documento seleccionado

- La presentación de los programas del emulador tendrá un aspecto profesional incluyendo comentarios en color verde, líneas con puntos de interrupción en color rojo, …

- El sistema de menús y barras de herramientas sólo activará en cada momento los comandos que se puedan utilizar.

2.3.4.2 Clase CMainFrame

La clase CMainFrame es la ventana principal de la aplicación. Desde ella el usuario podrá acceder a los menús, barras de herramientas y ver el código fuente que desea enviar al emulador. Tendrá las siguientes características:

Menús

- Archivo. El menú archivo dispondrá de los siguientes comandos:
    - Abrir. Será el comando que se utilizará para que el usuario pueda iniciar los casos de uso abrir archivo *.lst y abrir archivo *.emp
    - Guardar y Guardar como. Serán los comandos que se utilizarán para que el usuario pueda iniciar el caso de uso guardar archivo *.emp
    - Configurar las comunicaciones. Este comando permitirá seleccionar el puerto serie que se desea utilizar. Para simplificar el diseño del emulador no se ha previsto la posibilidad de modificar otras características de las comunicaciones serie.
    - Programar el emulador. Será el comando que se utilizará para que el usuario inicie el caso de uso Enviar el programa al emulador.
    - Salir. Finalizará el programa
    - Lista de ficheros recientes. En el menú aparecerá una lista con los últimos 4 ficheros abiertos.
- Edición. El menú edición dispondrá de los siguientes comandos:
    - Alternar punto de interrupción. Este comando se utilizará para iniciar los casos de uso añadir interrupción y quitar interrupción.
    - Borrar puntos de interrupción. Este comando se utilizará para quitar todos los puntos de interrupción. Iniciará tantas veces como sea necesario el caso de uso quitar interrupción.
- Ver. El menú ver tendrá los siguientes comandos:
    - Registros. Iniciará el caso de uso ver contenido de los registros del emulador.
    - Memoria. Iniciará el caso de uso ver contenido de la memoria

  - Barras de herramientas. Este será un menú desplegable que permitirá acceder a los comandos que mostrarán ó ocultarán las diferentes barras de herramientas que aparecen en el programa.

- Depuración. En este menú se encontrarán los siguientes comandos:
  - Ejecutar. Iniciará el caso de uso ejecutar el programa.
  - Ejecutar paso a paso. Iniciará el caso de uso avanzar una línea.
  - Interrumpir. Iniciará el caso de uso pausar la ejecución del programa.
  - Terminar. Iniciará el caso de uso parar la ejecución del programa.
- Ayuda. En este menú se incluirán elementos típicos en los menús de ayuda.

Los menús se diseñarán con el aspecto que tienen los menús de Windows XP y incluirán teclas de acceso rápido y descripción del comando en la barra de estado. Para darles el aspecto de XP se han utilizado unas librerías de clases descargadas de Internet. Las clases incluidas en esta librería que utilizaré en el programa son:

- CToolbarXP. Para dar aspecto de XP a las barras de herramientas
- CMenuXP. Para dar aspecto de XP a los menús de la aplicación
- CStatusbarXP. Para dar aspecto de XP a la barra de estado

Barras de herramientas

Se incluirán cuatro barras de herramientas que corresponderán a comandos del menú Archivo, menú Edición, menú Ver y menú depuración.

Las barras de estado se podrán ver ó ocultar, podrán ser flotantes o ajustarse a los cuatro extremos de la ventana de la aplicación. Al igual que los menús las barras de herramientas también tendrán una apariencia de Windows XP. La aplicación guardará en el registro el estado en que se encontraban los menús antes de cerrarse, de esta manera la siguiente vez que se abra mantendrá el aspecto que tenía la última vez que se ejecutó.

Barra de estado

Se incluirá una barra de estado en la que aparecerán mensajes aclaratorios de los comandos de la aplicación. La barra de estado tendrá una apariencia de Windows XP

Área cliente de la aplicación

En el área cliente de la aplicación se mostrará una ventana con el código fuente del programa que queremos ejecutar en el emulador.

Diagrama UML de CMainFrame

La clase CMainFrame hereda de la clase CframeWnd que está incluida en la librería de clases MFC de Microsoft. Su diagrama UML es el siguiente:

CMainFrame <<extends:CframeWnd>>
-CommandState:int -AlternarPintStatus:booleano -FileToolBar:CToolBarXP -EditToolBar:CToolBarXP -VerToolBar:CToolBarXP -DepuracionToolBar: CToolBarXP -StatusBar:CStatusBarXP -WndMemoria:FrmMemoria -WndRegistros:FrmRegistros -imgList:CImageList -imgListDis:CImageList -Commctrl:CCommCtrl -bancoReg0:Memoria -bancoReg1:Memoria -bancoRam0:Memoria -bancoRam1:Memoria -bancoEEprom:Memoria
+CMainFrame() +SetCommandState(cs:int) +GetCommandState():int +ActivarComandoInt(v:booleano) +GetMemoria(tipo:int):Memoria* #LoadFrame(nIDResource:UINT dwDefaultStyle:DWORD, pParentWnd: CWnd*, pContext:CCreateContext*) # OnCreate(lpCreateStruct: LPCREATESTRUCT):int #OnUpdateSave(pCmdUI:*CCmdUI) #OnUpdateProgemul(pCmdUI:*CCmdUI) #OnUpdatePlay(pCmdUI:*CCmdUI) #OnUpdateViewregisters(pCmdUI:*CCmdUI) #OnUpdateViewmemory(pCmdUI:*CCmdUI) #OnUpdatePlaystep(pCmdUI:*CCmdUI) #OnUpdatePlaypause(pCmdUI:*CCmdUI) #OnUpdatePlayend(pCmdUI:*CCmdUI) #OnUpdateAbrir(pCmdUI:*CCmdUI) #OnUpdateAlternarpi(pCmdUI:*CCmdUI) #OnUpdateBorrarpi(pCmdUI:*CCmdUI) #OnBarrasdeherramientasArchivo() #OnBarrasdeherramientasEdici() #OnBarrasdeherramientasVer() #OnBarrasdeherramientasDepuraci() #OnViewregisters() #OnViewmemory()

#OnArchivoConfigurarlascomunicaciones() #OnProgramEmulator() #OnPlay() #OnPlayStep() #OnPause() #OnPlayEnd() #OnCommEvent() -DoEvents() -EnviaInstruccion(inst:byte)

Descripción de los atributos

CommandState. Este atributo de tipo entero se utiliza para implementar una máquina de estados finita que permite activar o desactivar los menús y comandos de las barras de herramientas necesarios.

AlternarPintStatus- Este atributo de tipo booleano permite saber si se debe activar o desactivar el comando de alternar puntos de interrupción.

StatusBar. Este atributo es de la clase CstatusBarXP y es la barra de estado del formulario principal de la aplicación

FileToolBar. Este atributo es de la clase CtoolBarXP y es la barra de herramientas con algunos comandos del menú Archivo.

EditToolBar. Este atributo es de la clase CtoolBarXP. Es la barra de herramientas con los comandos del menú de edición.

VerToolBar. Este atributo es de la clase CtoolBarXP y contiene algunos comandos del menú Ver.

DepuracionToolBarXP. Este atributo es de la clase CtoolBarXP y en el se situarán los mismos comandos que en el menú Depuración.

WndMemoria. Este atributo de la clase FrmMemoria se utilizará para que la aplicación muestre la ventana con el contenido de la memoria del emulador

WndRegistros. Este atributo de la clase FrmRegistros se utilizará para que la aplicación muestre la ventana con el contenido de los registros del emulador.

ImgList. Este atributo es de la clase CimageList, una clase de la MFC que permite guardar una lista de imágenes. Esta lista de imágenes será necesaria para los iconos de las barras de herramientas.

ImgListDis. Este atributo de la clase CimageList guardará los iconos que aparecerán en las barras de herramientas cuando los comandos estén desactivados.

Commctrl. Este atributo de la clase CCommCtrl es el encargado de las comunicaciones a través del puerto serie

Los atributos de la clase, bancoReg0, bancoReg1,bancoRam0, bancoRam1 y bancoEEprom son objetos de la clase memoria que representarán los bancos de registros del emulador y las diferentes memorias del emulador.

Descripción de los métodos

CMainFrame(). Es el constructor de la clase. En este método se inicializará el atributo CommandState a 1 y AlternarPintStatus a 0.

SetCommandState(cs:int). Accesor de escritura al atributo CommandState

GetCommandState(). Accesor de lectura al atributo CommandState.

ActivarComandoInt(v:booleano). Accesor de escritura al atributo AlternarPintStatus.

GetMemoria(tipo:int). Este método será el accesor de lectura a los atributos bancoReg0, bancoReg1, bancoRam0, bancoRam1 y bancoEEprom. El parámetro tipo permitirá seleccionar el atributo que se desea leer:

- 1 → bancoReg0
- 2 → bancoReg1
- 3 → bancoRam0
- 4 → bancoRam1
- 5 → bancoEEprom

LoadFrame(…). Carga el Frame de la ventana. Los parámetros que recibe son:

- nIDResource:UINT. Es el identificador de los recursos asociados a la ventana
- dwDefaultStyle:DWORD. Define el estilo de la ventana.
- pParentWnd: CWnd*. Define la ventana padre
- pContext:CCreateContext*. Se utiliza para asociar un documento, su vista y la ventana principal donde se mostrará la vista.

En este método se deberán cargar el icono asociado a la aplicación y llamar al método UpdateMenuBar de CMenuXP para que cargue los menús con formato de XP. El resto de operaciones las realizará la superclase.

OnCreate(). Crea la ventana. Recibe como parámetro un puntero a la estructura CREATESTRUCT que define los diferentes parámetros necesarios para la creación de la ventana. En este método se deberán hacer las siguientes operaciones:

- Crear las barras de herramientas

- Crear el objeto de comunicaciones

El resto de operaciones los realizará la superclase

OnCreate devuelve 0 si se ha creado correctamente y -1 si ha habido un error

Los siguientes métodos OnUpdate* (pCmdUI:*CCmdUI) se invocan por la aplicación cuando necesita actualizar el estado del menú y/o comando de la barra de herramientas asociado al recurso definido por pCmdUI. En estos métodos consultaremos el valor del atributo CommandState y en función de su valor se indicará a la aplicación si debe activar o desactivar la aplicación. Los comandos se activarán según indica el diagrama de estados de la aplicación de la ilustración 17.

Para los métodos OnUpdateAlternarpi y OnUpdateBorrarpi se consultará el atributo AlternarPintStatus. Si el usuario se ha situado sobre una línea donde se puede poner un punto de interrupción AlternarPintStatus valdrá true y por lo tanto se podrá activar el menú y el comando de la barra de herramientas que permite poner un punto de interrupción.

OnBarrasdeherramientasArchivo. Este método se ejecutará cuando el usuario seleccione el comando de menú que permite mostrar o ocultar la barra de herramientas de comandos de archivo. Si la barra de herramientas está visible la mostrará y si no la ocultará.

Los métodos OnBarrasdeherramientasEdici(), OnBarrasdeherramientasVer() y OnBarrasdeherramientasDepuraci() funcionan igual que OnBarrasdeherramientasArchivo.

Los métodos OnViewregisters y OnViewmemory mostrarán las ventanas que presentan el contenido de los registros y el contenido de la memoria respectivamente.

El método OnArchivoConfigurarlascomunicaciones mostrará la ventana que permite seleccionar el puerto de comunicaciones serie.

OnProgramEmulator. Este método se ejecutará cuando el usuario pulse el comando para programar el emulador. Las acciones que se deberán realizar en este método son:

- abrir el puerto de comunicaciones serie
- enviar al emulador la instrucción de inicio de comunicaciones
- Llamar al método GetActiveDocument que hereda de la superclase para obtener el documento asociado a la ventana
- Recorrer todas las líneas del programa que contiene el documento y enviar para las instrucciones al emulador
- Enviar la instrucción de fin de envío del programa
- Esperar a que el buffer de salida se vacíe
- Cerrar el puerto serie

Dado que el buffer de salida del puerto de comunicaciones tiene un tamaño máximo se deberá comprobar antes del envío de cada instrucción si hay sitio en el buffer de salida. En caso de que el buffer esté lleno se deberá esperar a que se vacíe sin bloquear la aplicación. Para realizar esto realizaré un bucle que consulte el estado del buffer (commctrl.GetOutBufferCount() y dentro del bucle llamaré al método DoEvents que permite procesar los mensajes del sistema que le llegan a la aplicación, evitando de esta manera que el bucle la bloquee.

```
//cada instrucción son cuatro bytes
Si commctrl.GetOutBufferCount()  > commctrl.GetOutBufferSize() – 4
        Mientras que commctrl.GetOutBufferCount()  >
         commctrl.GetOutBufferSize() – 4
                Doevents()
        Fmientras
fsi
```

La información sobre el formato de instrucciones que se envían al emulador se encuentra en el capítulo ***Bloque de control del emulador***.

OnPlay. Envía la instrucción de ejecutar el programa al emulador.

OnPlayStep. Envía la instrucción de avanzar una instrucción al emulador.

OnPause. Envía la instrucción de pausar la ejecución.

OnPlayEnd. Envía la instrucción de detener la ejecución.

OnCommEvent. Este método se debe mapear a los eventos del control de comunicaciones y se utilizará para recibir los datos del emulador y los posibles errores de comunicaciones. Cuando se reciban los datos del emulador se llenarán los vectores bancoReg0, bancoReg1, bancoRam0, bancoRam1 y bancoEEprom con los datos recibidos del emulador. Cuando se reciba un mensaje de error se mostrará el mensaje correspondiente.

DoEvents. Este método permitirá a la ventana procesar los mensajes del sistema mientras espera a que haya sitio en el buffer de salida del puerto serie durante el envío del programa al procesador. El procesamiento de los mensajes se realizará de la siguiente manera:

```
// Procesa los mensajes existentes en la cola de la aplicación
while (::PeekMessage(&msg,NULL,0,0,PM_NOREMOVE))
{
      /* Cuando la cola se vacia hace una limpieza y sale del while*/
      if (!AfxGetThread()->PumpMessage())
            return;
}
```

EnviaInstruccion. Método que envía una instrucción al puerto serie. Recibe como parámetro el byte inst, que es la instrucción que se va a enviar. Las acciones que se deberán realizar en este método son:

- abrir el puerto serie
- enviar el byte
- esperar a que el buffer de salida se vacíe
- cerrar el puerto

Interfaces implementados por CMainFrame

Para poder obtener el aspecto de menú de Windows XP la clase CMainFrame debe implementar el interface MenuXP.

Para poder dar respuesta a las acciones del usuario cuando pulsa algún comando la clase CMainFrame deberá implementar el interna
ce Message_Map.

Para que la clase CMainFrame pueda procesar los eventos generados con el objeto CommCtrl debe implementar el interface EventSink_Map.

2.3.4.3 Clase Frm

La clase Frm la defino por generalización de atributos y métodos comunes de las clases FrmMemoria y FrmRegistros que se verán más adelante. Frm será una clase virtual ya que no existirá ningún objeto de esa clase.

Su diagrama UML es:

Frm <<extends:CDialog>>
-memdc:CDC -bmp:CBitmap -r:RECT -poldbmp:*CBitmap -pOldPen:*CGdiObject -pOldBrush:*CGdiObject -dc:*CPaintDC -viewMode:integer -memory:*Memoria
+Frm(nIDTemplate: UINT,pParent:CWnd*)

Todos los atributos menos viewMode y memory serán utilizados para realizar las operaciones gráficas necesarias para presentar los datos al usuario.

Antes de explicar estos atributos deseo exponer el funcionamiento que quiero que tengan las clases que heredan de Frm. Cuando el emulador encuentre un punto de

interrupción o el usuario pulse pause se volcará el contenido de los registros y la memoria del Emulador al PC. En ese momento el usuario podrá acceder a la ventana de visualización de registros o de memoria. Estas dos ventanas heredarán de Frm y sobre ellas dibujaré una imagen con aspecto de tabla y dentro de ella escribiré el contenido de los registros o de la memoria. Para realizar las operaciones gráficas necesitaré una serie de objetos que son los atributos de la clase Frm.

Según la documentación de Microsoft los pasos que necesitaré para realizar las operaciones gráficas deseadas son:

- Crear un objeto de la clase CpaintDC a partir de la clase Frm, que deriva de Cdialog. El atributo dc será un puntero a este objeto.

- Crear un objeto CDC a partir del objeto dc. Este objeto será el atributo memdc.

- Poner en memdc el bitmap sobre el que quiero escribir los registros o el contenido de la memoria. Esta operación se realiza con el método SelectObject de la clase memdc. El bitmap que pondré en memdc corresponde al atributo bmp.

- Cuando se pone en uno de los objetos gráficos de Microsoft otro objeto, como puede ser un bitmap, un lápiz, una brocha,... hay que guardar el anterior para restaurarlo después de realizar las operaciones gráficas. Para guardar el bitmap que hay en memdc antes de poner el mío utilizaré el atributo poldbmp.

- Poner un lápiz negro en memdc y guardar el anterior. Lo guardaré en el atributo pOldPen.

- Poner una brocha blanca en memdc para que cuando escriba el fondo de las letras quede en blanco. La brocha que exista anteriormente en memdc la guardaré en el atributo pOldBrush.

- Escribir los contenidos de los registros o de la memoria. Para saber en todo momento las coordenadas del rectángulo en el que escribiré el texto utilizaré el atributo r, del tipo RECT (rectángulo).

- El usuario podrá escoger el sistema de numeración con el que desea que se le presente la información, hexadecimal, decimal o binario. La elección del usuario quedará guardada en el atributo viewMode.

- Una vez finalizadas las operaciones gráficas restituiré todos los objetos originales.

El atributo memory es un puntero a un objeto de la clase Memoria. Las subclases FrmMemoria y FrmRegistros utilizarán este atributo para tener el vector con los registros o las posiciones de memoria que desean presentar en pantalla.

El único método que presenta esta clase es el constructor que recibe como parámetros el identificador del recurso correspondiente al cuadro de diálogo y la ventana madre del cuadro de diálogo, o sea, la ventana principal de la aplicación. Lo único que se deberá realizar en el constructor es inicializar viewMode a 0 para seleccionar por defecto la presentación en hexadecimal.

2.3.4.4 Clase FrmMemoria

La clase FrmMemoria hereda de la clase Frm y será la ventana en la que se presentará el contenido de la memoria del emulador. En esta ventana el usuario podrá seleccionar el código numérico en el que desea ver el contenido del emulador y el banco de memoria del emulador que desea ver, banco 0 , banco 1 o EEPROM. El funcionamiento de la clase en lo que se refiere a la presentación de la información en pantalla ya se ha descrito en la clase Frm. El banco de memoria que se desea ver y el código numérico de visualización (viewMode de la clase FRM) se escogerán con botones tipo RadioButton. El intercambio de información entre estos botones de radio y sus variables asociadas se realizará según las técnicas definidas por Microsoft para sus cuadros de diálogo.

FrmMemoria <<extends:Frm>>
-memoryTipe:int
+FrmMemoria(parent:CWnd) +OnBnClicked() #DoDataExchange(pDX:*CdataExchange) #OnPaint() -printRegister(value:BYTE)

Atributos

MemoryTipe. Indica el banco de memoria que ha escogido el usuario, 0 → banco de memoria 0, 1 → banco de memoria 1, 2 → EEPROM

Métodos

FrmMemoria. Este constructor que añado a la clase FrmMemoria sólo recibe como parámetro su ventana padre. Dentro del constructor deberé inicializar el atributo memoryTipe, llamar al constructor de la superclase indicándole el recurso que debe utilizar para crear el cuadro de diálogo y la ventana padre y cargar el bitmap que deseo utilizar para presentar los valores de las celdas de memoria.

OnBnClicked. Este método lo defino para que se ejecute cuando el usuario pulse el botón de salir que pondré en la ventana. En este método se cerrará la ventana.

DoDataExchange. Este método se define por especialización del método DoDataExchange de la superclase. En el se obtendrá el valor de memoryTipe y de vieMode (definido en Frm) cuando el usuario pulse los RadioButton.

OnPaint. Se define por especialización del método OnPaint de Cdialog, clase de la que hereda a través de Frm. El sistema llama a este método cuando necesita redibujar la ventana. En el se realizarán las operaciones gráficas que se han explicado en la clase Frm, y se utilizarán los atributos que hereda de Frm. Para escribir el contenido de la memoria se recorrerá el atributo memory, que hereda de Frm, e irá llamando sucesivamente al método PrintRegister.

PrintRegister. Este método escribirá en pantalla el valor del registro o de la celda de memoria que se le pasa en el parámetro value, de tipo byte. El valor se escribirá en las coordenadas de pantalla que indica el atributo r, del tipo Rectangle, que se hereda de la clase Frm. Una vez escrito el valor deberá actualizar r para que apunte a las coordenadas del bitmap que corresponden a la siguiente celda de memoria que se debe escribir.

2.3.4.5 Clase FrmRegistros

La clase FrmRegistros corresponde a la ventana que presentará el contenido de los registros del Emulador al usuario. En esta ventana se dibujará un bitmap sobre el que se escribirán los registros, habrá unos botones RadioButton que permitirán al usuario escoger el sistema númerico en el que desea ver los registros y un botón Aceptar para cerrar la ventana. FrmRegistros heredará de Frm (y de Cdialog a través de Frm).

FrmRegistros <<extends:Frm>>
+FrmRegistros(pParent:CWnd) +OnBnClicked() #DoDataExchange(pDX:*CdataExchange) #OnPaint() -selectBank(n:int) -printRegister(value:BYTE)

Esta clase no añade ningún nuevo atributo a los que hereda de Frm.

Los métodos de los que dispondrá son:

FrmRegistros. Es el constructor de la clase. Sus funciones serán las de llamar al constructor de la superclase, indicándole que recurso debe cargar para crear la ventana de diálogo y cual será su ventana padre. También deberá cargar el bitmap sobre el que dibujaré el contenido de los registros.

OnBnClicked. Se ejecutará cuando el usuario pulse el botón de Aceptar. En este método se cerrará la ventana.

DoDataExchange. Este método se define por especialización del método DoDataExchange de la superclase. En el se obtendrá el valor de vieMode (definido en Frm) cuando el usuario pulse los RadioButton. Tal como se ha indicado en la clase FrmMemoria, el intercambio de información entre los RadioButton y el atributo viewMode se realizará siguiendo las especificaciones de Microsoft.

OnPaint. Su funcionamiento es idéntico al del método OnPaint de la clase FrmMemoria, con las diferencias que implica que la información a escribir es algo diferente.

PrintRegister. Este método escribirá en pantalla el valor del registro o de la celda de memoria que se le pasa en el parámetro value, de tipo byte. El valor se escribirá en las coordenadas de pantalla que indica el atributo r, del tipo Rectangle, que se hereda de la clase Frm. Una vez escrito el valor deberá actualizar r para que apunte a las coordenadas del bitmap que corresponden a la siguiente celda de memoria que se debe escribir.

2.3.4.6 Arquitectura documento-vista

El programa Emulpic lo realizaré utilizando la arquitectura documento vista de Microsoft. Esta arquitectura consiste en un conjunto de clases que permiten asociar la clase principal de la aplicación, el documento que queremos visualizar, la vista donde la visualizaremos y la ventana en la que se pondrá esa vista. Esta asociación se realiza a través de las clases CdocManager y CsingleDocTemplate de la MFC de Microsoft. La clase principal de la aplicación, en el caso de Emulpic es CemulpicApp, tiene un objeto de la clase Cdocanager. Este objeto tiene un objeto de la clase CsingleDocTemplate, que guarda el objeto correspondiente al documento, la vista relacionada y los asocia. En el diagrama de clases de la aplicación podemos ver la interacción de estas clases entre sí. Para realizar Emulpic deberé definir la clase vista, CemulpicView y la clase documento, CemulpicDoc. CemulpicView, CemulpicDoc y CsingleDocTemplate utilizan la ventana principal de la aplicación CmainFrame. CsingleDocTemplate la crea, y CemulpicView y CemulpicDoc presentan en ella el documento, en este caso el programa que se enviará al emulador.

CemulpicDoc

Esta clase corresponde al documento que la aplicación utilizará en la arquitectura documento vista. El documento será el programa que se enviará al emulador. Esta clase la obtendré a partir de herencia múltiple de la clase Programa y de la clase Cdocument de Microsoft. Al heredar de Programa convierto a la clase en un contenedor del tipo vector, (Programa hereda de la clase Vector) que contiene cada una de las líneas del programa (encapsuladas por el objeto Linea). Al heredar de la clase Cdocument doto a la clase de la funcionalidad necesaria para su funcionamiento dentro de la arquitectura documento-vista de la aplicación.

CemulpicDoc <<extends:Cdocument,Programa>>
+CemulpicDoc( ) +Serialize(ar:Carchive) +SaveModified():booleano

La clase CemulpicDoc no incorpora ningún atributo Nuevo.

Los métodos que se definen en esta clase son:

CemulpicDoc. Es el constructor de la clase. No requiere ninguna inicialización especial.

Serialize. Se declara por especialización del método Serialize de la superclase Cdocument. Este método se llamará desde el framework cuando el usuario abra o cierre un documento. Las operaciones de archivo del documento se realizarán a través del parámetro ar, un objeto de la clase Carchive que manipula el archivo correspondiente al documento.

El siguiente pseudocódigo ilustra las operaciones que se deberán realizar dentro de Serialize:

```
Si el archivo ar está abierto para escritura
        Llamar al método save de la superclase Programa
Sino (está abierto para lectura)
        Si el archivo abierto tiene extensión lst (es un archivo de Emulpic)
                Llamar al método LoadLst de la superclase Programa
        Sino
                intenta abrirlo como un archivo de formato emulpic llamando al
                método Load de la superclase Programa
        fsi
fsi
```

SaveModified. Este método lo declaro por especialización del método SaveModified de Cdocument. Este método se llama automáticamente desde el framework cuando detecta que el documento tiene el atributo de modificado a true. Como Emulpic abre archivos del tipo lst (que se crean con Mplab) y emu (archivo de emulpic) pero sólo guarda archivos en formato emu, debo especializar el método SaveModified de la superclase Cdocument. Este método comprobará si la extensión del documento es emu, y en caso afirmativo lo guardará.

CEmulpicView

Esta clase es la vista del documento, la que gestionará la presentación en pantalla de los programas que se enviarán al emulador. CEmulpicView se encargará de escribir en la pantalla el programa, poniendo en verde los comentarios y en rojo las líneas del

programa sobre las que se sitúa un punto de interrupción. También controlará el scroll de la ventana, para lo que heredará de la clase MFC CScrollView.

CEmulpicView <<extendí:CScrollView>>
-numLines:int -CurrLine:int -CurrLineIsCode:booleano
+CEmulpicView() +OnDraw(pdc:*CDC) +OnInitialUpdate() +OnContextMenu(pWnd:CWnd,point:CPoint) #OnAlternarpi() #OnBorrarpi() #OnLButtonDown(nFlags:UINT,point:CPoint) #OnRButtonDown(nFlags:UINT,point:CPoint) #OnLButtonDblClk(nFlags:UINT,point:CPoint) #OnKeyDown(nChar:UINT,nRepCnt:UINT,nFlags:UINT) -CalculaLinea(point:CPoint) -GetDocument():*CEmulpicDoc

El atributo numLines guarda el número de líneas que tiene el documento.

El atributo CurrLine guarda el número de línea sobre la que ha situado el ratón el usuario.

CurrLineIsCode indica si la línea actual corresponde a una línea con código ejecutable.

El método GEmulpicView es el constructor de la clase.

El método OnDraw modifica el método OnDraw de la superclase ya que es necesario especializarlo para esta clase. El pseudocódigo que ilustra las acciones que se deberán realizar en este método es:

```
OnDraw
{pre:existe un documento que se va a presentar en la pantalla}

        maxLineWidth:int  //ancho de la línea más grande
        lineWidth: int  //Ancho de la línea
        altoFuente: int //altura de la fuente
        anchoDocumento:  int  //guarda  el  ancho  del  documento  en  unidades  de  la
        pantalla
        altoDocumento:int //guarda la altura del documento en unidades de la pantalla

        altoFuente = Obtener el tamaño de la fuente de la ventana
```

```
Inicializar el rectángulo que guarda las dimensiones en las que se escribirá la
línea
Para cada línea del programa
        Si la línea es la que tiene seleccionada el usuario
                Dibujar el rectángulo de la línea con fondo azul
        Sino
                Si la línea tiene una interrupción
                        Dibujar el rectángulo de la línea con fondo rojo
                Sino
                        Dibujar el rectángulo de la línea con el color de fondo de
                        la ventana
                Fsi
        Fsi
        Escribir el código de la línea en negro
        Escribir el código de la línea en verde
        lineWidth = calcular el ancho de la línea escrita
        si lineWidth > maxLineWidth
                maxLineWidth = lineWidth
        fsi
        Actualizar el rectángulo con las coordenadas de la siguiente línea.
fpara

anchoDocumento = MaxLineWidth
altoDocumento = Numero de líneas * altoFuente
{Post:se ha escrito el programa en la pantalla. En anchoDocumento y altoDocumento
se han guardado las dimensiones del documento en unidades de pantalla para
utilizarlas en los métodos de scroll}
```

El método OnInitialUpdate substituye el método del mismo nombre de la superclase para especializarlo. En este método se inicializará el tamaño de las barras de scroll y se pondrán a cero los atributos de la clase.

El método OnContextMenu se mostrará cuando el usuario pulse el botón derecho del ratón sobre la ventana. En esté método se ejecutarán las acciones necesarias para mostrar un menú contextual.

El método OnAlternarPi se ejecuta cuando el usuario pulsa el comando para poner puntos de interrupción. En este método se deberá comprobar si la línea actual tiene código ejecutable (CurrLineIsCode), y en caso afirmativo se pondrá o quitará el punto de interrupción y se activará el flag de modificado del documento. Si la línea tenía punto de interrupción lo quitará y si no tenía punto de interrupción lo pondrá.

OnBorrarpi. Este método borrará todos los puntos de interrupción del programa.

Los métodos OnLButtonDown y OnRButtonDown se ejecutan cuando el usuario pulsa el botón izquierdo o derecho del ratón sobre la ventana. En estos métodos se calculará el número de línea sobre la que se ha pulsado, se buscará en el documento ese número de línea y se comprobará si se puede poner una interrupción en ella (si la línea

tiene código ejecutable se podrá poner una interrupción). En caso de que se pueda poner una interrupción se activarán los comandos de la barra de herramientas y los menús que permiten poner un punto de interrupción.

El método OnLButtonDblclk se ejecuta cuando el usuario hace doble clic sobre la ventana. En este método se calcula el número de línea correspondiente a las coordenadas sobre las que se ha situado el ratón, se comprueba si la línea puede tener un punto de interrupción y en caso afirmativo se pone un punto de interrupción, o se quita si ya existía uno.

El método OnKeyDown se utilizará para permitir que el usuario pueda desplazar las barras de desplazamiento con las teclas del cursor, avance de página y retroceso de página. Este método lo invoca el sistema operativo cuando el usuario pulsa una tecla. En función de la tecla pulsada, dato que se recibe como parámetro, se llamará al método de scroll correspondiente.

El método CalculaLinea será utilizado por otros métodos para calcular el número de línea del programa que corresponde a las coordenadas sobre las que se ha pulsado con el ratón. Para obtener el número de línea se aplicarán las siguientes fórmulas:

N = Número de línea
Ps = Posición de la barra de scroll vertical
p.y = coordenada y sobre la que se ha pulsado el ratón
f.h = alto de la fuente

N = (Ps+p.y)/f.h

Como la línea tiene varios pixels de alto, si tenemos decimales se debe incrementar el número de línea:

Resto = (Ps+p.y) % f.h

```
Si Resto > 0
        N=N+
fsi
```

El método GetDocument devuelve un puntero al documento asociado a la vista.

2.4 IMPLEMENTACIÓN

A continuación se presentan unas capturas de pantalla para ilustrar el aspecto con el que ha quedado Emulpic.

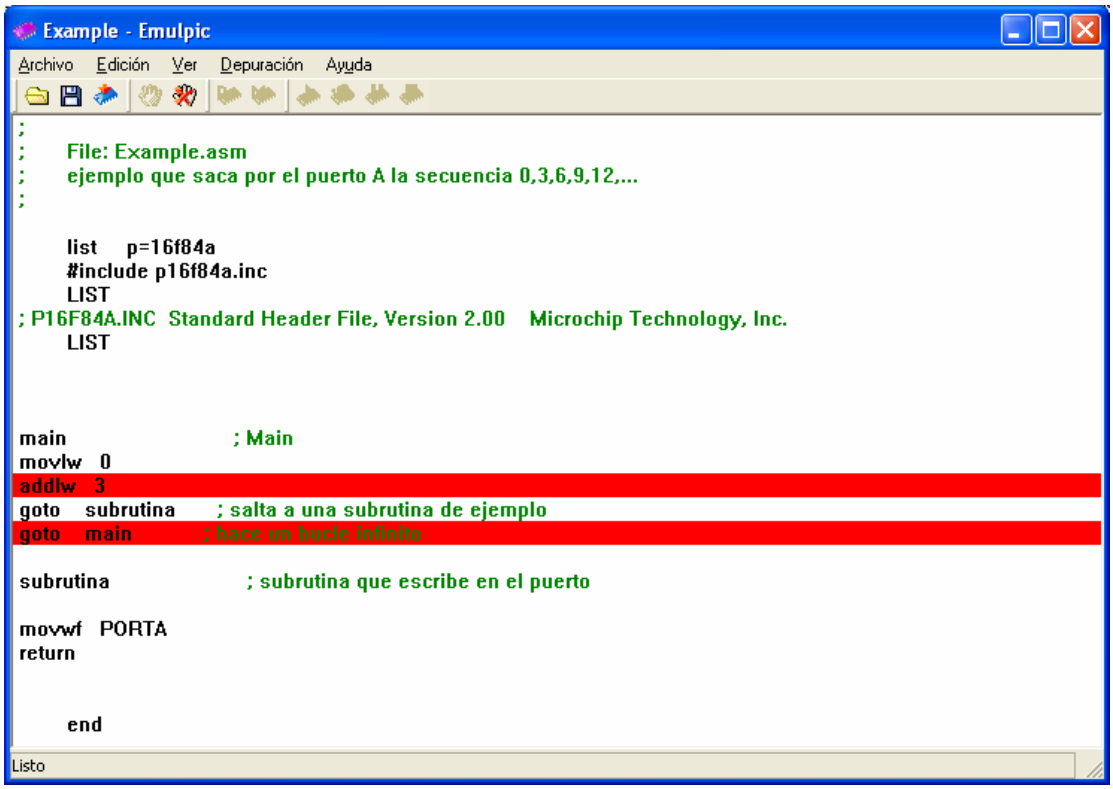
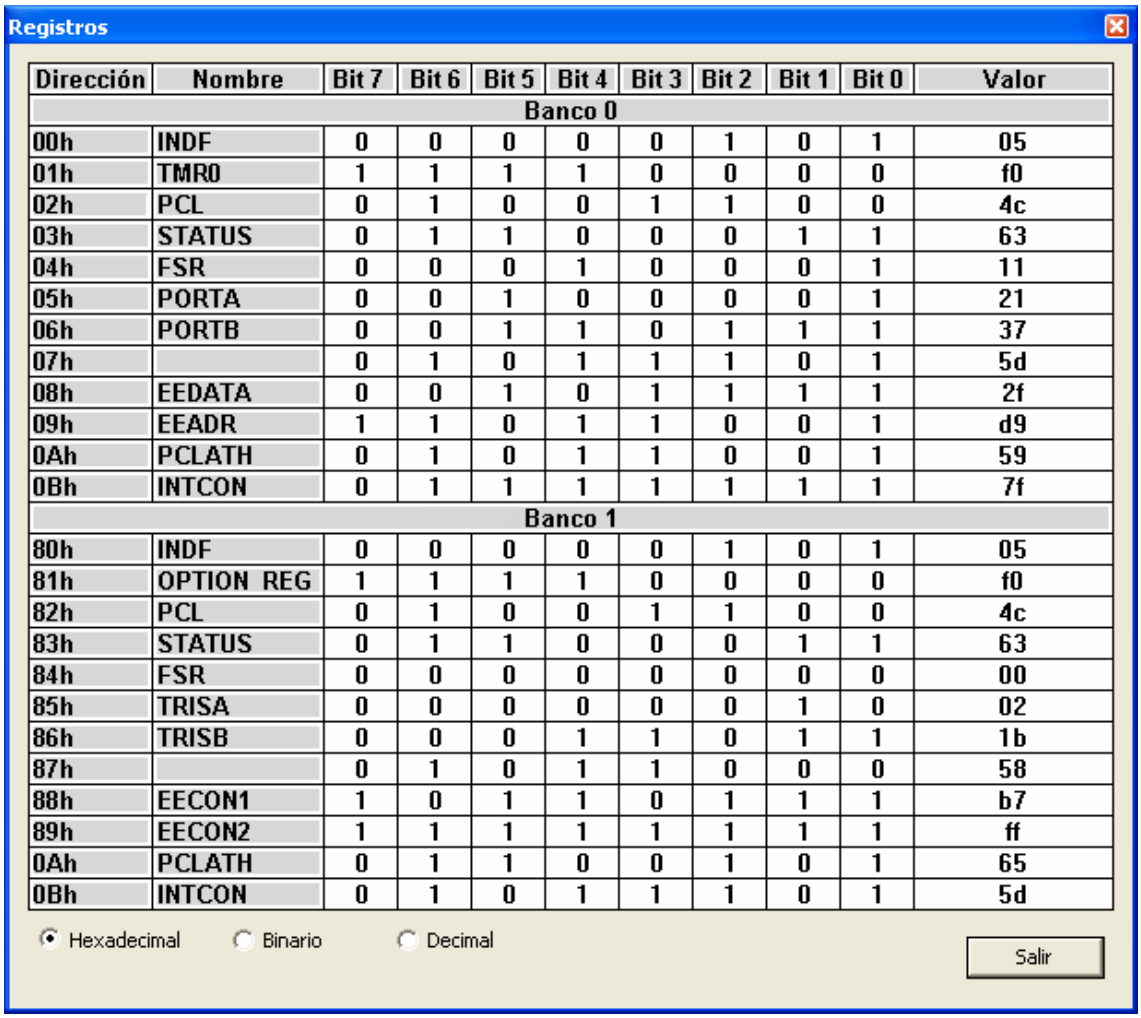


Ilustración 20. Emulpic con un programa cargado.

En la ilustración 20 se puede observar la ventana principal de la aplicación con un pequeño programa de ejemplo. Se puede observar las líneas con los puntos de interrupción, en rojo, y los comentarios del programa, en verde.



Dirección	Nombre	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Valor
Banco 0										
00h	INDF	0	0	0	0	0	1	0	1	05
01h	TMR0	1	1	1	1	0	0	0	0	f0
02h	PCL	0	1	0	0	1	1	0	0	4c
03h	STATUS	0	1	1	0	0	0	1	1	63
04h	FSR	0	0	0	1	0	0	0	1	11
05h	PORTA	0	0	1	0	0	0	0	1	21
06h	PORTB	0	0	1	1	0	1	1	1	37
07h		0	1	0	1	1	1	0	1	5d
08h	EEDATA	0	0	1	0	1	1	1	1	2f
09h	EEADR	1	1	0	1	1	0	0	1	d9
0Ah	PCLATH	0	1	0	1	1	0	0	1	59
0Bh	INTCON	0	1	1	1	1	1	1	1	7f
Banco 1										
80h	INDF	0	0	0	0	0	1	0	1	05
81h	OPTION_REG	1	1	1	1	0	0	0	0	f0
82h	PCL	0	1	0	0	1	1	0	0	4c
83h	STATUS	0	1	1	0	0	0	1	1	63
84h	FSR	0	0	0	0	0	0	0	0	00
85h	TRISA	0	0	0	0	0	0	1	0	02
86h	TRISB	0	0	0	1	1	0	1	1	1b
87h		0	1	0	1	1	0	0	0	58
88h	EECON1	1	0	1	1	0	1	1	1	b7
89h	EECON2	1	1	1	1	1	1	1	1	ff
0Ah	PCLATH	0	1	1	0	0	1	0	1	65
0Bh	INTCON	0	1	0	1	1	1	0	1	5d

Ilustración 21. Ventana con el contenido de los registros del emulador.

La ilustración 21 muestra el aspecto que tiene la ventana que presenta el contenido de los registros del emulador. Se puede observar como el usuario puede reconocer fácilmente los registros por su nombre y dirección, y ver su contenido, ya sea en bits o el valor del byte. También puede seleccionar el código numérico que se utilizará para visualizar el valor del registro.

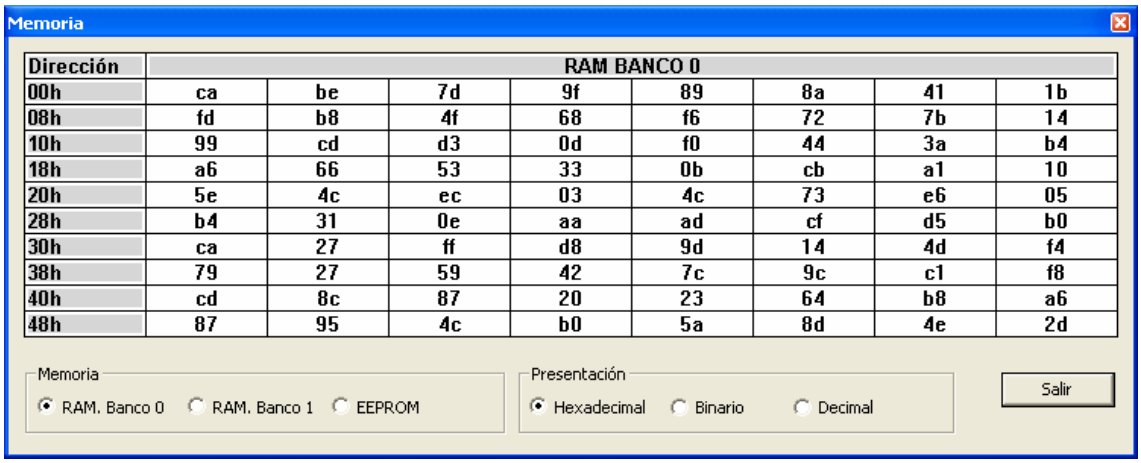


Ilustración 22. Ventana con el contenido de la memoria del Emulador

La ilustración 22 muestra como se ve el contenido de la memoria del emulador. Se puede observar que el usuario puede escoger el banco de memoria deseado y el código de visualización.

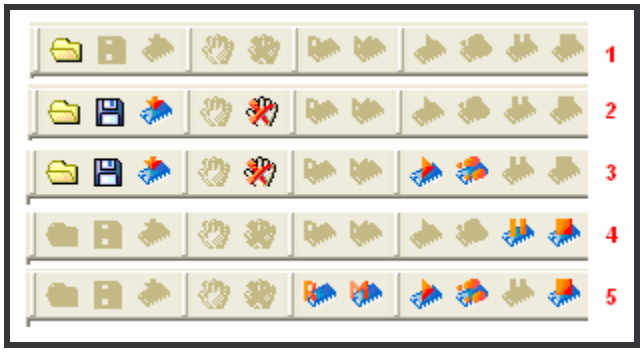


Ilustración 23. Estados de la aplicación

En la ilustración 23 se puede observar como la barra de estados activa sólo los comandos necesarios según el estado de la aplicación (ver diagrama de estados de la aplicación en el punto 2.2.6)

Otras características de la aplicación son:

- Lista de archivos más recientes (ilustración 24)

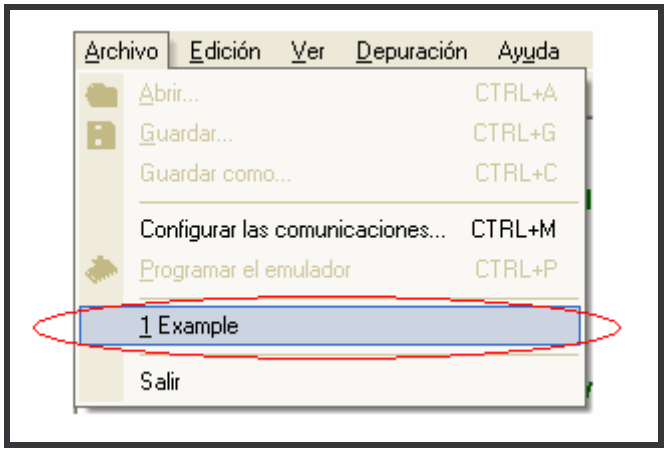


Ilustración 24. Lista de archivos más recientes

- Los archivos de Emulpic quedan registrados, apareciendo en el explorador con su icono. Si se hace doble clic sobre el archivo se ejecuta Emulpic y se abre el archivo seleccionado

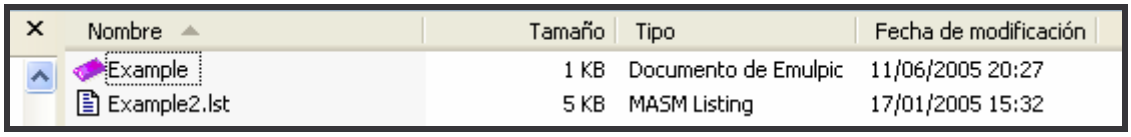


Ilustración 25. Archivos de emulpic

- Las barras de herramientas se pueden mostrar y ocultar y se pueden aparcar a cualquiera de los cuatro lados de la ventana y ser flotantes:

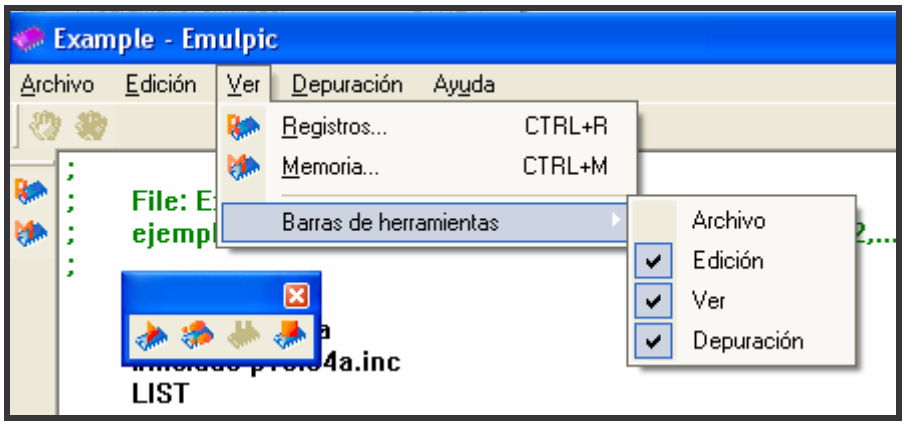


Ilustración 26. Barras de herramientas dinámicas

El funcionamiento de las comunicaciones serie salientes se han comprobado mediante el programa Software Serial Monitor Lite, que monitoriza el tráfico que circula a través de los puertos serie:

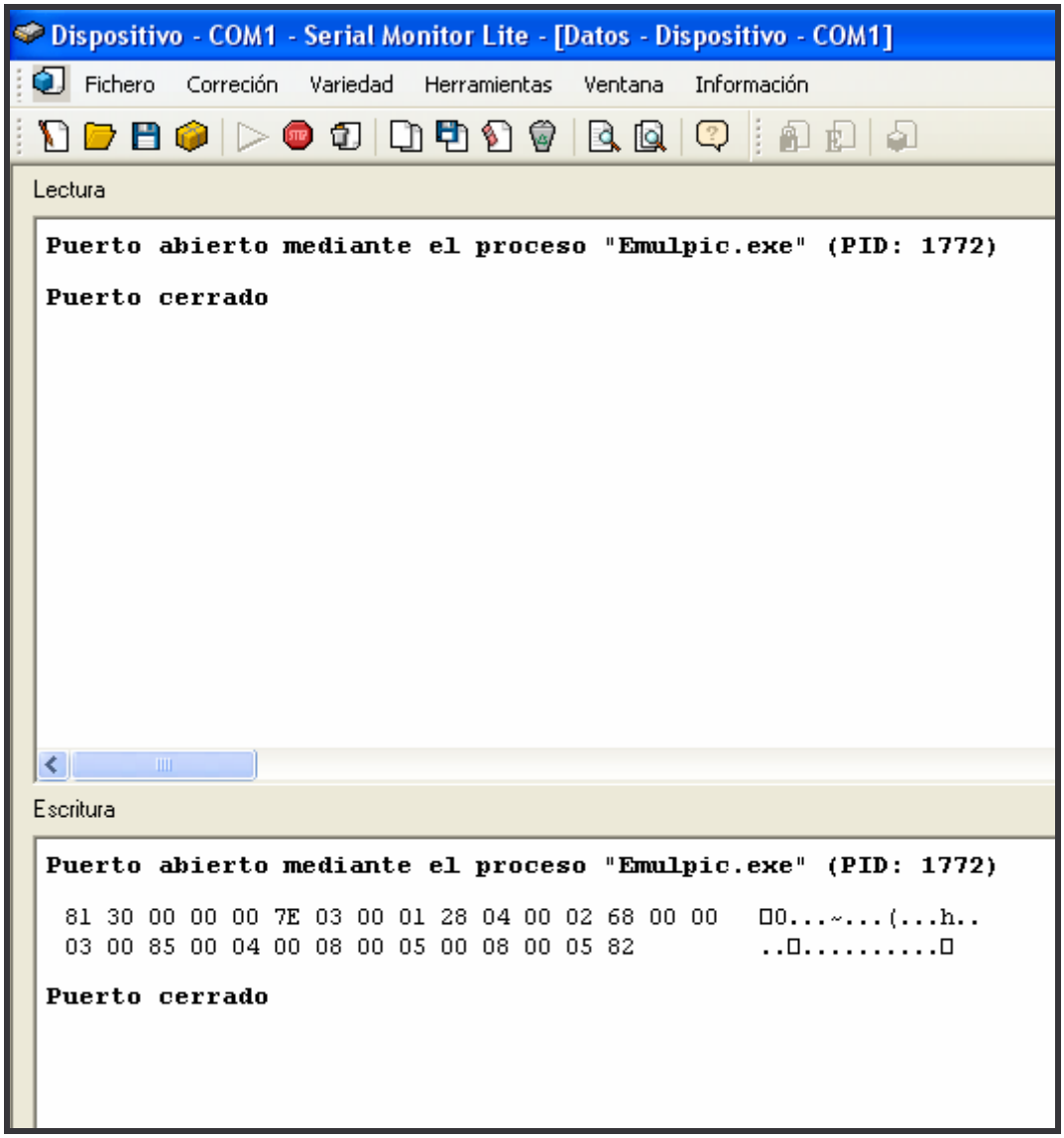


Ilustración 27. Comprobación de las comunicaciones serie salientes.

Para comprobar el funcionamiento de las comunicaciones serie entrantes se ha realizado un pequeño programa en Visual Basic 6.0 que envía un conjunto de datos aleatorios del mismo tamaño que los que enviaría el emulador:

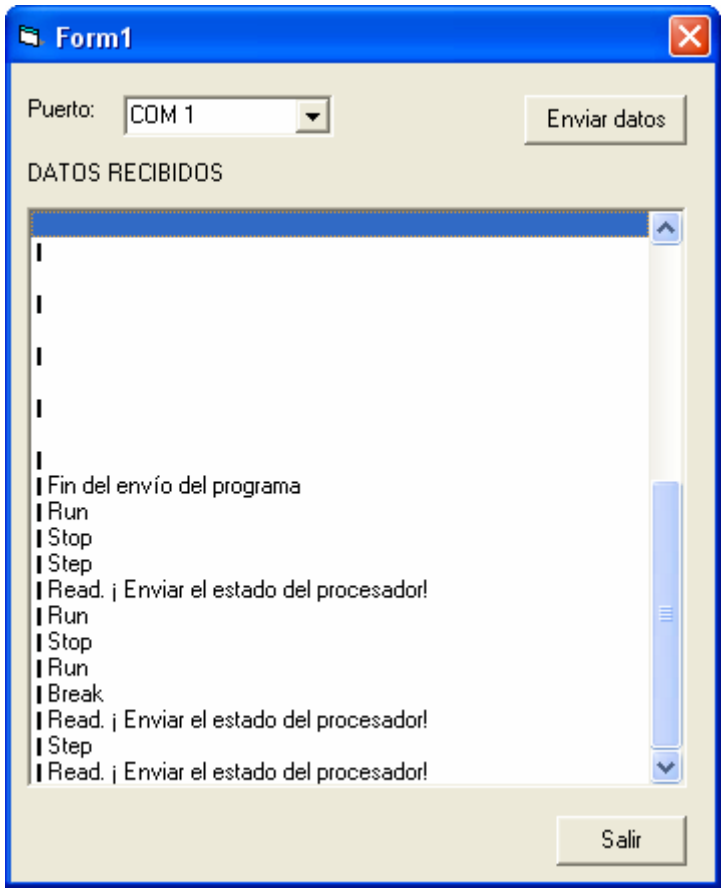


Ilustración 28. Programa para probar las comunicaciones de Emulpic

El programa va recibiendo las ordenes de Emulpic y cuando recibe la Instrucción READ se debe pulsar Enviar datos para simular que el emulador ha enviado el contenido de sus registros y memoria.

3 DISEÑO DEL EMULADOR

La emulación del microcontrolador PIC se realizará en el emulador. La creación del emulador consistirá en diseñar un sistema electrónico que permita simular el microcontrolador PIC y que permita comunicar el emulador con el PC. La programación del emulador se realizará con VHDL y se grabará en la FPGA que incorpora la tarjeta de desarrollo comprada a MJL y que ha sido descrita en la introducción del proyecto.

El diseño del emulador a programar consistirá en tres grandes bloques:

- el bloque de gestión de comunicaciones. Se encargará de recibir datos del PC a través del puerto serie y de enviar datos al PC. Los datos recibidos serán el programa que debe ejecutar el emulador y las ordenes (play, pause, step) que envía el PC. Los datos a enviar serán el contenido de los registros y de la memoria.

- El bloque de control del emulador recibirá los comandos y datos del PC y actuará sobre el bloque de emulación para que este ejecute los comandos enviados.

- El bloque de emulación serán los circuitos secuenciales y combinacionales necesarios para emular el microcontrolador, registros, ALU, memoria, puertos E/S,…

Cuando el PC envié el programa este será recibido por el bloque de gestión de comunicaciones que lo enviará al bloque de control del emulador. Este accederá a la memoria de programa del bloque de emulación y lo irá grabando. Cuando el programa se haya enviado y se reciba la orden de ejecución en el bloque de gestión de comunicaciones el bloque de control le indicará al bloque de emulación que debe iniciar la ejecución. El programa se ejecutará en el bloque de emulación hasta que se encuentre una instrucción con punto de interrupción o se reciba la orden de pausar la ejecución. En estos momentos el bloque de control accederá a los registros y a la memoria del bloque de emulación e irá enviando todos los bytes al bloque de gestión de comunicaciones para que los transmita al PC.

3.1 GESTIÓN DE COMUNICACIONES ENTRE EL EMULADOR Y EL PC

Las comunicaciones entre el PC y el emulador se realizarán a través del puerto serie. La conexión del puerto RS-232C estará configurada como módem nulo. Para esta conexión se utilizarán las líneas TxD, RxD, RTS y CTS:

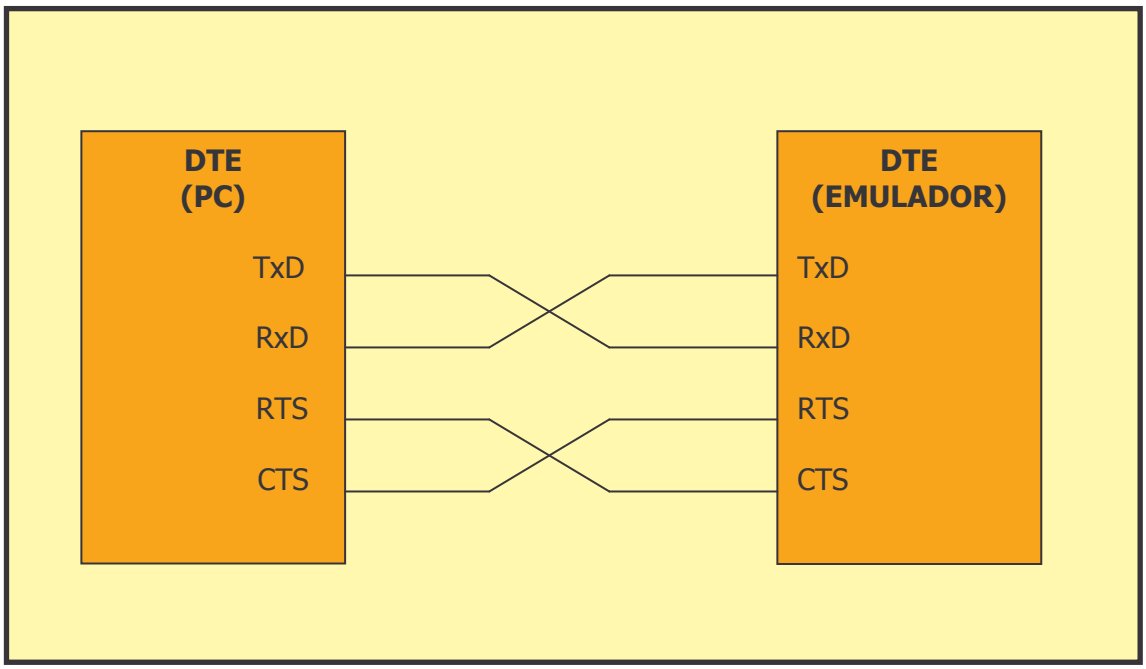


Ilustración 29. Conexión RS-232C en configuración de módem nulo

Las comunicaciones estarán configuradas de la siguiente manera:

- Velocidad: 9600 baudios
- Control de flujo: RTS-CTS
- 8 bits de datos + 1 bit de start + 1 bit de stop

En la siguiente ilustración podemos ver como evolucionan las diferentes líneas durante el proceso de comunicación:

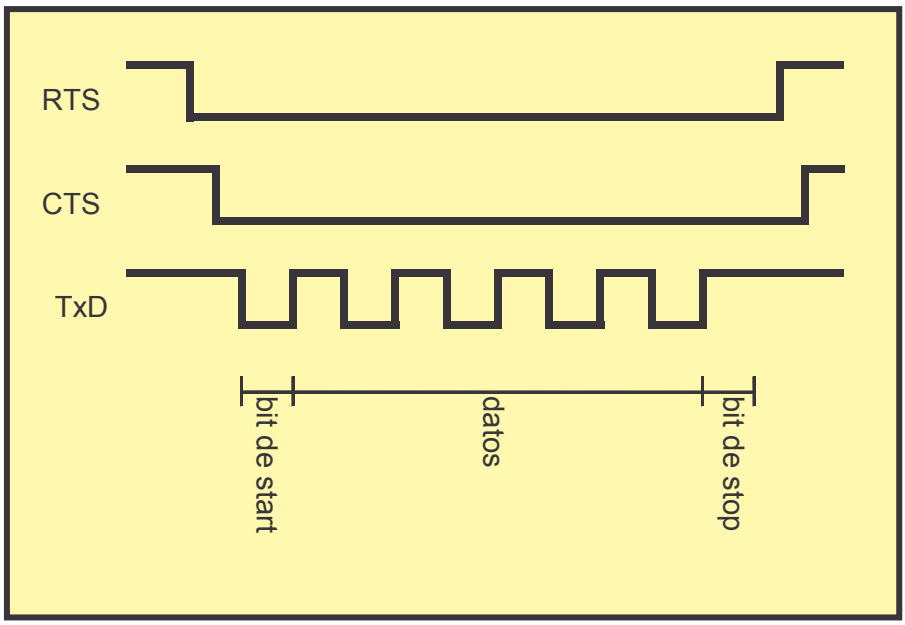


Ilustración 30. Cronograma de las señales que intervienen en la comunicación serie.

El equipo que quiere emitir los datos activa la señal RTS (activa por 0). Cuando el equipo que debe recibir los datos detecta la petición del emisor le informa de que está preparado para recibirlos activando la línea CTS. Cuando el emisor recibe la activación de la línea CTS procede a enviar los datos. Como he escogido que los datos se enviarán en paquetes de 8 bits + 1 bit de start + 1 bit de stop el primer bit que se enviará es el bit de start (activo por 0). Después seguirán los 8 bits de datos y finalmente el bit de stop (activo por 1). Cuando el emisor termine de enviar la información desactivará la señal RTS y el receptor desactivará la señal CTS.

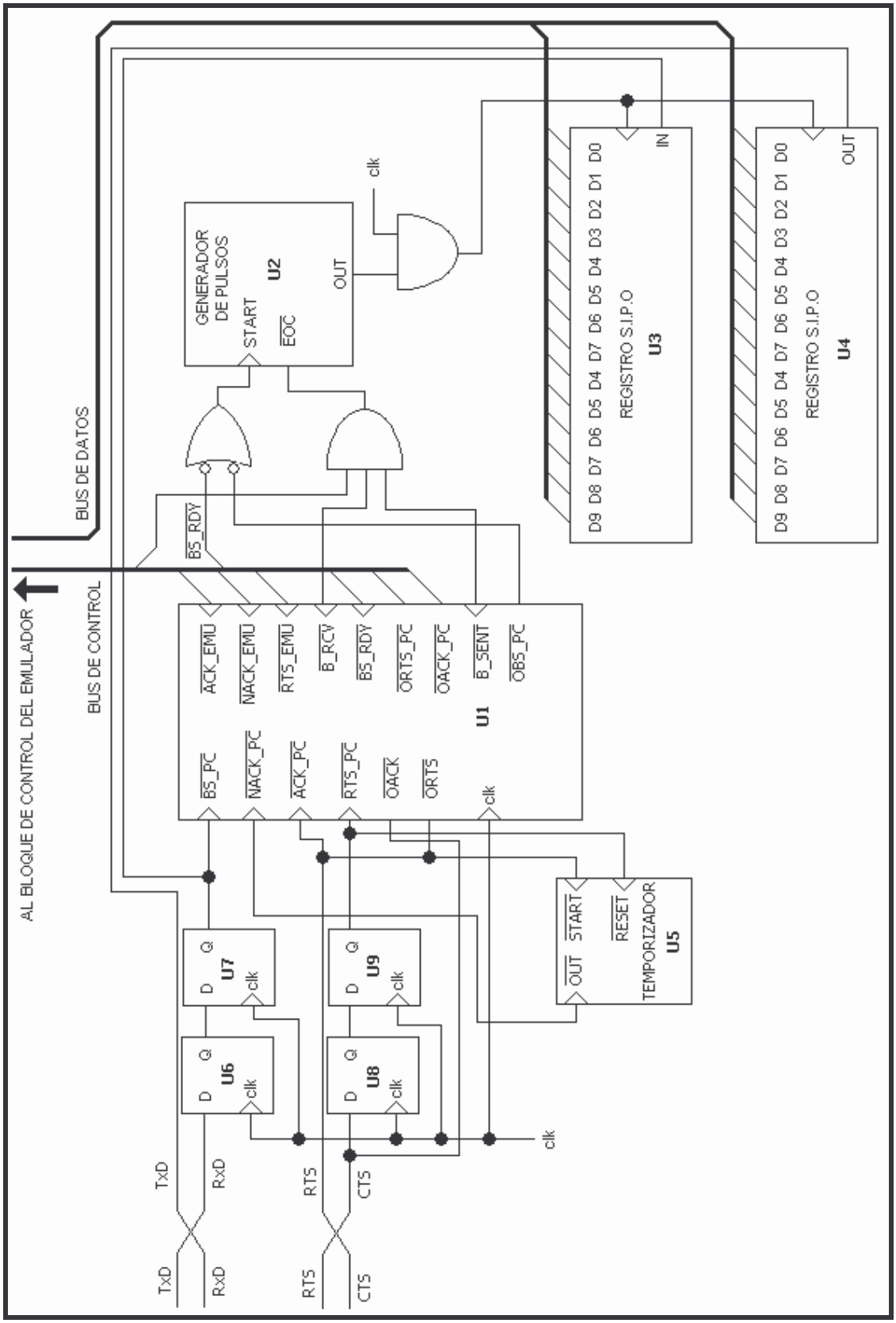


Ilustración 31. Bloque de gestión de comunicaciones

El bloque de gestión de comunicaciones está compuesto de:

- **U1**. Es una máquina de estados finita que recibe las señales del PC y del emulador. Es el cerebro del gestor de comunicaciones. Controla el estado en que se encuentran las comunicaciones, envía las señales de control que se envían del PC al emulador y del emulador al PC. Genera señales auxiliares que facilitan la tarea de pasar los bits que se reciben en serie a bytes y los bytes que se han de enviar a bits.

- Generador de pulsos **U2**. Está programado para generar pulsos sincronizados con la velocidad del puerto serie, 9600 baudios. Para cada bit que se reciba o que se deba enviar se generará un pulso del ancho adecuado. Se activa al recibir en su entrada de start la señal OBS_PC (activa por cero) que genera U1 al detectar un flanco descendente en TxD correspondiente al bit de start. Cuando se han generado los pulsos para leer los 8 bits de datos, el bit de start y el bit de stop se activa la señal EOC (activa por 0) que avisará a la máquina de estados que puede pasar a esperar otro byte.

- Registro de desplazamiento con entrada serie y salida paralelo **U3**. Este registro recibirá los bits recibidos por el PC y facilitará al emulador el byte correspondiente.

- Registro de desplazamiento con entrada paralelo y salida serie **U4**. El emulador pondrá en este registro cada byte que desea enviar y mediante la salida serie de se enviarán los bits a través de la RS-232C.

- Temporizador **U5**. Su función será establecer un tiempo de espera para recibir la señal CTS del PC después de que el emulador envíe la señal RTS. Si después del tiempo establecido el PC no responde que está dispuesto a recibir las comunicaciones del emulador este temporizador generará una señal que avisará al emulador de que no se puede comunicar y que hará que U1 pase a su estado inicial.

- Sincronización de señales de entrada **U6**, **U7**, **U8** y **U9**. Las señales de entrada al emulador deben estar sincronizadas con el reloj interno del emulador para evitar metaestabilidades. Esta sincronización de las señales de entrada es imprescindible para el correcto funcionamiento de un sistema secuencial síncrono.

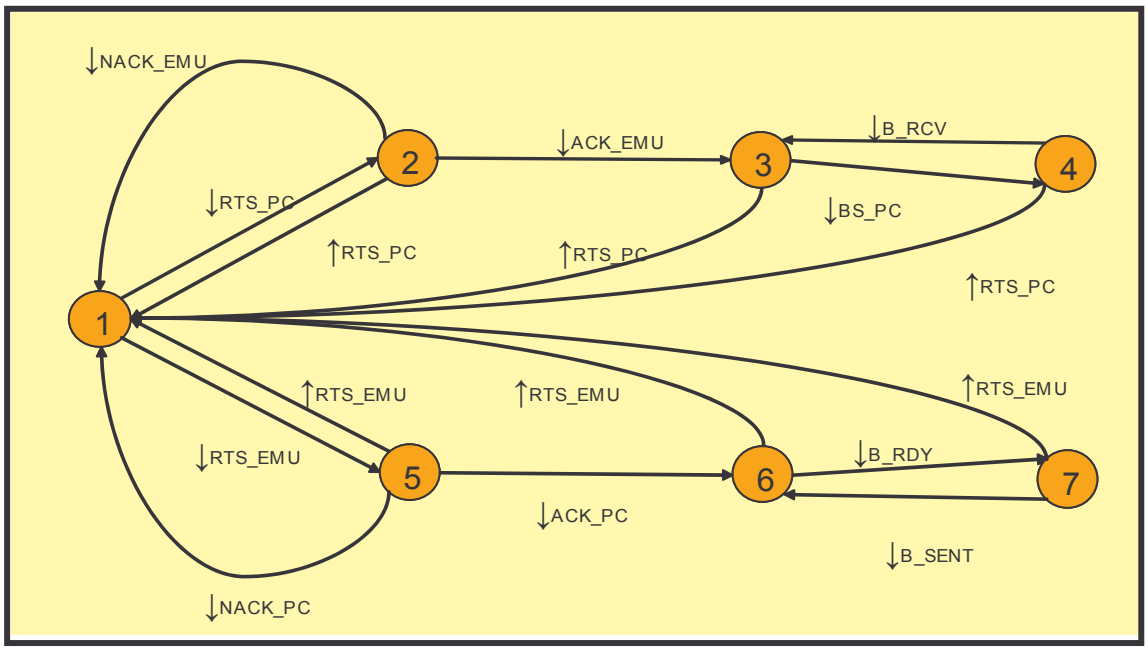


Ilustración 32. Diagrama de estados del bloque de comunicaciones.

Las señales que intervienen en el cambio de estado de U1 son:

↓RTS_PC. Esta señal corresponde a la activación de la línea RTS por parte del PC. Indica al gestor de comunicaciones que el PC desea enviar datos. Es activa por flanco descendente.

↑RTS_PC. Esta señal corresponde a la desactivación de la línea RTS por el PC. Es activa por flanco ascendente y indica que el PC ya ha terminado de enviar datos.

↓RTS_EMU. Cuando el emulador desea iniciar una comunicación solicita al gestor de comunicaciones que active la línea RTS. Es activa por flanco descendente.

↑RTS_EMU. Cuando el emulador desea finalizar una comunicación solicita al gestor de comunicaciones que desactive la línea RTS.

↓ACK_EMU. Cuando el emulador recibe la señal RTS del PC le indica al gestor de comunicaciones que active la línea CTS si puede atender la solicitud del PC. Es activa por flanco descendente.

↓NACK_EMU. El emulador le indica al gestor de comunicaciones que no puede atender la solicitud del PC. Activa por flanco descendente.

↓ACK_PC. Cuando el PC recibe una solicitud de comunicaciones del emulador (RTS) activa la línea CTS si puede recibir la transmisión. Activa por flanco descendente.

↓NACK_PC. Si después de enviar una petición de comunicaciones desde el emulador al PC no se recibe respuesta del PC (activación de la línea CTS) se activa

esta señal para que el gestor de comunicaciones vuelva al estado inicial. Activa por flanco descendente

↓BS_PC. Corresponde a la recepción del bit de start que se envía desde el PC. Activa por flanco descendente.

↓B_RCV. Cuando se reciben los 8 bits de datos y el bit de stop se activa esta señal que indica que el gestor de comunicaciones puede pasar a esperar otro byte. Activa por flanco descendente.

↓BS_RDY. Cuando el emulador tiene disponible un byte para que el gestor de comunicaciones lo envíe al PC se lo indica al gestor activando esta línea. Activa por flanco descendente.

↓B_SENT. El gestor de comunicaciones activa esta señal cuando ha enviado un byte al PC (8 bits de datos + 1 bit de start + 1 bit de stop). Activa por flanco descendente.

ESTADO INICIAL	SEÑAL	ESTADO FINAL	ACCIONES
1	↓RTS_PC	2	Activa la señal ORTS_PC (activa por 0) que indicará al bloque de gestión del emulador que el PC desea enviar datos
	↓RTS_EMU	5	Activa la señal ORTS (activa por 0) que indicará al PC que el emulador desea enviar datos
2	↓ACK_EMU	3	Activa la señal OACK (activa por 0) para que se active la línea ACK que indicará al PC que el emulador está preparado para recibir datos
	↑RTS_PC	1	Desactiva la señal ORTS_PC para indicar al bloque de gestión del emulador que el PC ha finalizado de enviar datos
	↑NACK_EMU	1	Desactiva la señal OACK para indicar al PC que el emulador no puede recibir datos
3	↓BS_PC	4	Activa la señal OBS_PC (activa por 0) de la máquina de estados U1 para indicar al generador de pulsos que empieze a funcionar
	↑RTS_PC	1	Desactiva la señal ORTS_PC para indicar al bloque de gestión que el PC ha finalizado de enviar datos
4	↓B_RCV	3	---
	↑RTS_PC	1	Desactiva la señal ORTS_PC para indicar al bloque de gestión del emulador que el PC ha finalizado de enviar datos
5	↓ACK_PC	6	Activa la señal OACK_PC (activa por 0) para indicar al gestor del emulador que el PC puede recibir datos
	↑RTS_EMU	1	Desactiva la señal ORTS para indicar al PC que el emulador a terminado de enviar datos
	↑NACK_PC	1	Desactiva la señal OACK_PC para indicar al gestor del emulador que el PC no puede recibir datos
6	↓BS_RDY	7	---

	↑RTS_EMU	1	Desactiva la señal ORTS para indicar al PC que el emulador ha terminado de enviar datos
7	↓B_SENT	6	---
	↑RTS_EMU	1	Desactiva la señal ORTS para indicar al PC que el emulador ha terminado de enviar datos

3.2 BLOQUE DE CONTROL DEL EMULADOR

El bloque de control del se encargará de procesar los bytes que el bloque de gestión de comunicaciones ha recibido y realizará las tareas oportunas.

Los bytes recibidos por el emulador se dividen en dos categorías:

- Comandos. Son órdenes que se envían desde el PC.
- Datos. Corresponden a los datos del programa que deberá ejecutar el emulador.

3.2.1 Datos

Los datos corresponderán al programa que deberá ejecutar el emulador.

Tendrán el siguiente formato:

Orden	Primer byte enviado								Segundo byte enviado							
Nº de bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
descripción	Id 0	Bit de control de puntos de interrupción	Instrucción													

Orden	Tercer byte enviado								Cuarto byte enviado							
Nº de bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
descripción	Dirección de memoria															

El bit 31 (id) puesto a cero indicará al emulador que va a recibir un dato del programa

El bit 30, bit de control de punto de interrupción, se utilizará para saber si antes de ejecutarse esa instrucción se debe detener el emulador

Los bits 16 a 29 son la instrucción del programa

Los bits 0 a 15 son la dirección de memoria donde se debe cargar la instrucción

3.2.2 Comandos

Los comandos que se enviarán al emulador tendrán el siguiente formato:

Nº de bit	7	6	5	4	3	2	1	0

descripción	Id 1	Comando

El bit id puesto a uno indicará al emulador que va a recibir un comando

Los bits 0 a 6 corresponderán al código del comando

Los bytes que se enviarán al emulador son:

Descripción	Tipo	Abreviación	Valor
Comprobar la presencia del emulador	Comando	TEST	10000000
Inicio de envío del programa	Comando	BP (Begin Program)	10000001
Fin del envío del programa	Comando	EP (End Program)	10000010
Ejecutar el programa	Comando	RUN	10000011
Finalizar la ejecución del programa	Comando	STOP	10000100
Detener la ejecución del programa	Comando	BREAK	10000101
Leer el estado del emulador (lee los registros, la memoria,...)	comando	READ	10000110
Avanzar una instrucción en la ejecución del programa	Comando	STEP	10000111
Primer byte de datos de programa	dato	DATA1	0XXXXXXX
Segundo byte de datos de programa	Dato	DATA2	XXXXXXXX
Tercer byte de datos de programa	Dato	DATA3	XXXXXXXX
Cuarto byte de datos de programa	Dato	DATA4	XXXXXXXX

Los bytes DATA2, DATA3 y DATA4 pueden tomar cualquier valor lo que haría que se pudieran confundir con cualquier comando enviado por el PC. Para evitar este problema se implementará una máquina de estados que interpretará los datos recibidos. Al establecer la máquina de estados que información puede recibir en cada momento se evita cualquier ambigüedad en los datos recibidos.

3.3 BLOQUE DE EMULACIÓN DEL MICROCONTROLADOR PIC 16F84

Este bloque será el conjunto de elementos de lógica secuencial y combinacional que emularán el funcionamiento del PIC 16F84. Implementaré una versión reducida de este microcontrolador y dispondrá de la ALU, los registros, la memoria de datos RAM y EEPROM, el contador de programa, la memoria de programa, el registro de trabajo W, el registro de estado STATUS, la pila y varios multiplexores que permitirán seleccionar los datos de entrada a la ALU, al bus de direcciones de la memoria. Para simplificar el proyecto no se incluirán los puertos de E/S ni los recursos de temporización de que dispone el microcontrolador.

La base del diseño del emulador será el diagrama de bloques proporcionado por Microchip en su dataSheet. A este diagrama de bloques le eliminaré los componentes que no se prevén implementar. Además de los elementos que se eliminarán serán necesarias las siguientes modificaciones:

- añadir un conjunto de multiplexores y ampliar los buses para poder tener acceso al contenido de la memoria y de los registros desde el bloque de control del emulador.

- La memoria de programa deberá tener un bit más (15) que la memoria de programa del PIC16F84. Este bit permitirá saber si existe un punto de interrupción en una instrucción. La interrupción se debe producir antes de que se ejecute la instrucción.

- También se deberá añadir la lógica necesaria en el bloque de decodificación de instrucciones del emulador para detectar los puntos de interrupción. Cuando el bloque de decodificación de instrucciones detecte un punto de interrupción deshabilitará las entradas de reloj de todos los componentes del emulador hasta que se reciba la orden de continuar desde el PC.

En la ilustración 33 se muestra el diagrama de bloques del emulador del microcontrolador PIC y como interactúa con el bloque de control del emulador.

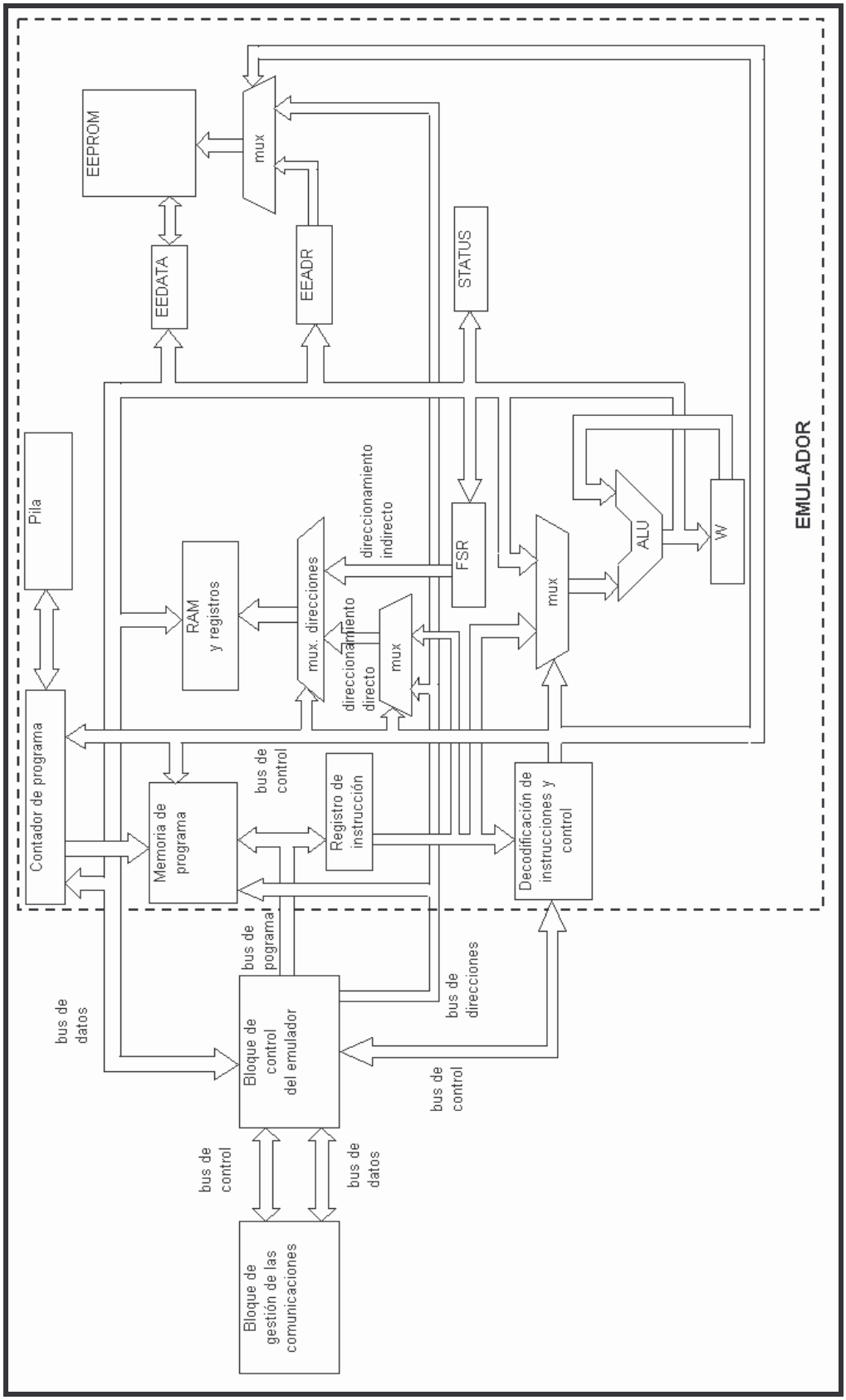


Ilustración 33. Diagrama de bloques del emulador

Cuando se recibe el programa desde el PC el bloque de gestión de comunicaciones va recibiendo los bits y pasa la información a bytes que posteriormente envía al bloque de control del emulador.

Cuando el bloque de control del emulador recibe el comando que le indica que a continuación va a recibir el programa (comando Begin Program) se prepara para empezar a guardar la información en la memoria de programa del Emulador, inicializando el contador que direccionará la memoria y indicando al bloque de decodificación de instrucciones del emulador que va a tomar el control del bus de programa.

Cuando empieza a recibir el programa va direccionando la memoria del programa y poniendo las instrucciones en el bus de programa.

Al recibirse la instrucción de que se ha finalizado el envío del programa (End Program) el bloque de control del emulador le indica al emulador que ha finalizado el envío del programa para que inicialice el bloque del emulador para la inminente ejecución del programa (inicializar el contador de programa, tomar el control de los buses y activar las señales que controlan los multiplexores).

La recepción de la instrucción RUN inicia la ejecución del programa. El bloque de decodificación de instrucciones y de control del microprocesador irá incrementado el valor del contador de programa. Con el dato del contador de programa se direcciona la memoria de programa y la instrucción que se obtiene se carga en el registro de instrucción para que la decodifique el bloque de decodificación de instrucciones y gestione la ejecución del programa.

En la memoria de programa se ha puesto un bit más en cada palabra para informar al bloque de decodificación de instrucciones de las líneas que tienen interrupción. Cuando el bloque de decodificación de instrucciones detecta que está activo el bit de interrupción desactiva las señales de reloj que llegan a todos los componentes del microcontrolador y avisa al bloque de control del emulador de que ha encontrado una interrupción. En ese momento el bloque de control del emulador toma el control de los buses y empieza a leer la información de los registros y de la memoria y la va enviando al PC. Lo mismo ocurre cuando se recibe la orden de pausa (BREAK) desde el PC.

En esos momentos el sistema permanece parado hasta que no se recibe la orden de ejecución (RUN) desde el PC. De esta manera el usuario puede consultar tranquilamente el contenido de la memoria y del emulador.

Las características del hardware del emulador que permitirán realizar este proceso son:

- La memoria de programa dispondrá de dos buses de direcciones, uno para el contador de programa y otro para el bloque de control del emulador. El bloque de decodificación de instrucciones dará las señales necesarias para la activación del bus adecuado.

- Previo al multiplexor de direcciones que hay antes de la memoria RAM se situará otro multiplexor en el bus correspondiente al direccionamiento directo. Este multiplexor permitirá escoger entre el direccionamiento que realiza el microcontrolador o el que realiza el bloque de control del emulador cuando desea enviar el contenido de la memoria al PC. El bloque de decodificación de instrucciones del microprocesador proporcionará las señales de control de este multiplexor.

- En el bus de direcciones, antes de la memoria EEPROM, se situará otro multiplexor que permitirá seleccionar el direccionamiento por parte del microcontrolador a través del registro EEADR (funcionamiento normal del microcontrolador) o el direccionamiento a través del bloque de control del emulador cuando esté leyendo el contenido de la memoria y de los registros para enviar la información al PC.

Como se puede observar he previsto utilizar dos técnicas diferentes para leer las memorias del microcontrolador una mediante multiplexores y un único bus de datos en la RAM y la EEPROM, y otra mediante dos buses de datos en la memoria del programa. He escogido probar estos dos métodos para analizar cual sería el más conveniente.

4 CONCLUSIONES

4.1 ESTIMACIÓN DE REQUISITOS PARA EL DESARROLLO DE UN EMULADOR PROFESIONAL

El estudio realizado ha demostrado la viabilidad técnica de un emulador de microcontroladores PIC basado en FPGA, explorando las técnicas necesarias para su realización y ofreciendo un orden de magnitud de su dificultad.

No obstante, en un producto profesional y competitivo que una empresa quisiera sacar al mercado se deberían ampliar las especificaciones y contemplar los siguientes puntos:

- En lugar de utilizar el kit de desarrollo de FPGA de MJL que se ha utilizado en el proyecto, la empresa debería construir su propia placa con FPGA para el emulador. En adelante me referiré a esta hipotética placa como ***EmulBoard***.

- Debe emular varios microcontroladores de Microchip y/o otros microcontroladores del mercado.

- El cliente debería tener la posibilidad de ampliar su emulador con nuevos microcontroladores que se vendan por separado.

- ***Emulpic*** debería permitir la configuración de la lógica programable de la FPGA según el tipo de microcontrolador que se desea utilizar.

- En el proyecto realizado, ***Emulpic*** sólo abría programas de MPLab que tuvieran un único archivo fuente escrito en assembler. La versión comercial debería ofrecer la posibilidad de cargar programas con más de un archivo de código fuente, programados en C, con macros,...

- El proyecto sólo se ha realizado para leer archivos de código fuente realizados con Mplab. Habría que evaluar la posibilidad de compatibilidad con programas realizados en otros entornos de desarrollo para microcontroladores PIC y/o otros microcontroladores.

- Tanto ***Emulpic*** como la placa tendrían que permitir modificar el contenido de los registros y la memoria del emulador en tiempo de depuración

- Las comunicaciones se tendrían que realizar a través del puerto USB en lugar del puerto serie y con un protocolo de transmisión que sea detector de errores.

- se debería evaluar la posibilidad de desarrollar el software Emulpic para plataformas Linux.

Como se puede ver las especificaciones de un producto profesional son mucho más amplias y requieren de un equipo de profesionales para su realización. A continuación se realiza un breve estudio de los medios necesarios, que podría servir como base para la realización de un análisis económico del proyecto. El estudio sólo contempla el personal y los medios técnicos que considero que serían imprescindibles para la realización de un producto comercial, no he entrado a valorar el tiempo necesario para su consecución.

Personal necesario para el desarrollo de la placa EmulBoard

Alternativa 1. Realizar el diseño y la programación del hardware

- Un ingeniero electrónico o de telecomunicaciones. Sus competencias serían el diseño y la coordinación de la realización de la placa del emulador. Tendría que aportar experiencia en diseño de hardware basado en FPGA y programación en VHDL.

- Un ingeniero técnico electrónico o de telecomunicaciones para diseño de Hardware. Sus competencias serían de colaboración con el diseñador y coordinador del proyecto de hardware, realización de planos, de documentación, construcción de prototipos, ensayos,... Debería tener experiencia en el desarrollo de hardware.

- Dos ingenieros técnicos electrónicos o de telecomunicaciones para diseño de software. Sus competencias serían la programación en VHDL de los emuladores de microcontroladores PIC. Debería aportar experiencia en programación con VHDL y en programación de microcontroladores PIC

- La fabricación de la placa del emulador se contrataría a otra empresa.

Alternativa 2. Contratar el diseño del hardware y realizar su programación

- Un ingeniero electrónico o de telecomunicaciones. Sería el coordinador del proyecto y el interlocutor con la empresa que diseñe la placa para el emulador. Debería aportar experiencia en diseño de hardware basado en FPGA y programación en VHDL.

- Dos ingenieros técnicos electrónicos o de telecomunicaciones para diseño de software. Sus competencias serían la programación en VHDL de los emuladores de microcontroladores PIC y/o otros microcontroladores. Tendría que tener experiencia en programación con VHDL y experiencia en programación de microcontroladores.

- La fabricación de la placa del emulador se contrataría a otra empresa.

Equipamiento necesario para la realización de la placa Emulboard

- Un kit de desarrollo para la FPGA que se fuera a utilizar en el proyecto. Sería utilizado por los programadores en VHDL mientras no estuviera desarrollada la placa Emulboard

- Equipamiento básico de laboratorio de electrónica, osciloscopio, fuente de alimentación, generador de señales, componentes electrónicos,... Sería utilizado para realizar montajes de prueba para verificar el funcionamiento del emulador.

Software necesario para la elaboración de la placa Emulboard

- Licencias de Windows y Office para los desarrolladores de hardware y software

- 2 licencias del entorno de desarrollo ModelSim para VHDL.

- 1 licencia de un programa de CAD de electrónica tipo ORCAD

- 1 licencia de un programa de simulación electrónica tipo PSPICE

- 1 licencia de cada uno de los entornos de desarrollo para PIC que serían compatibles con Emulpic.

Personal necesario para la elaboración del programa Emulpic

- 1 Ingeniero informático. Se encargaría del análisis y la coordinación del desarrollo de Emulpic. Debería tener experiencia como analista y programación en C++ y MFC.

- 1 Ingeniero técnico informático, con programación en C++ y MFC para el interface gráfico de la aplicación.

- 1 Ingeniero técnico informático programador en C++ para la programación no relacionada con el interface gráfico.

Todos deberían conocer el entorno de desarrollo Visual Studio .NET.

Software necesario para la elaboración del programa Emulpic

- 3 Licencias de Windows y Office

- 3 Licencias de Visual Studio .NET

Infraestructura informática

Las necesidades de la infraestructura informática podrían variar mucho en función de la infraestructura existente en la hipotética empresa, a la que se le debería añadir

- Un PC para cada trabajador.
- Incorporación de los equipos a la LAN de la empresa ya sea dentro de alguna subred existente o en nuevas subredes.
- Un servidor para central compartido por los electrónicos e informáticos, con una base de datos documental que permita gestionar y compartir los documentos entre los desarrolladores. Esta base de datos nos permitiría controlar las versiones de los documentos, compartir correctamente los documentos entre distintas personas evitando modificaciones indeseadas, ...
- Un sistema de copias de seguridad en el servidor

La infraestructura de la red debería asegurar que:

- Las estaciones de trabajo de los electrónicos e informáticos tienen un acceso seguro a Internet (sistemas cortafuegos, antivirus,...)
- Los trabajadores disponen de correo electrónico
- Los nuevos PC's tienen acceso a impresoras en red

4.2 CONSIDERACIONES ECONÓMICAS

Desde el punto de vista técnico las conclusiones que he obtenido son:

- El proyecto es técnicamente viable.
- Sería necesario un equipo mínimo de entre 6 y 7 personas. Para asegurar la consecución del proyecto sería imprescindible que estas personas tuvieran amplía experiencia en las funciones a desempeñar. Así mismo considero que el perfil de alguno de los trabajadores no seria fácil de encontrar. Por estos motivos los sueldos podrían resultar elevados.
- Es necesario un software y hardware de desarrollo que no es precisamente barato.

- Independientemente de que se asignaran al proyecto más personas considero imposible su realización en menos de 18 meses.

Desde el punto de vista económico parece que:

- nos encontramos ante un proyecto que requeriría una inversión inicial que podría alcanzar fácilmente los 0,6 millones de €, entre sueldos, licencias de software y equipos.

- El dinero invertido no se empezaría a recuperar, como mínimo, hasta 18 meses después.

- No es un producto de consumo general, su posible mercado es bastante pequeño.

Para poder conocer la viabilidad económica del proyecto y valorar su rentabilidad sería imprescindible realizar:

a) un planning fiable de los plazos de ejecución del proyecto

b) calcular detalladamente la inversión inicial

c) un detallado análisis económico de los costes del producto y un estudio del precio de los productos de la competencia para obtener el margen de beneficios.

d) un estudio de mercado que junto con el margen de beneficios nos podría dar una idea del plazo de amortización de la inversión inicial.

5 BIBLIOGRAFIA

PROGRAMACIÓN CON VHDL

VHDL. Programming by example. Cuarta edición, 2002. Autor: Douglas L. Perry. Editorial: McGraw Hill.

IEEE 1076. VHDL standard. Disponible como documento electrónico en http://www.ieee.org/.

VHDL. A logic synthesis approach. Primera edición, 1997. Autor: David Naylor. Editorial Chapman & Hall.

VHDL, Lenguaje estándar de diseño electrónico. Primera edición, 1998. Autores: Lluís Terés, Yago Torroja, Serafín Olcoz, Eugenio Villar. Editorial Mac Graw Hill.

Apuntes del curso de postgrado de la UPC "Tecnicas de diseño digital y sistemas configurables". Fundació Politècnica de Catalunya, 2004.

PROGRAMACIÓN CON .NET

MSDN Library for Visual Studio .NET 2003. Archivos de documentación electrónica de Microsoft para el entorno de desarrollo .NET. Se encuentra disponible en versión de CD o en versión on-line.

BIBLIOGRAFIA DE LA UOC

Fonaments de programacio I. Modulos didácticos. EDIUOC, 2001. Autores: Pere Botella López, Miquel Bofill Arasa.

Fonaments de programacio II. Módulos didácticos. EDIUOC, 2003. Autores: Julià Minguillón i Alfonso, David Cabanillas Barbacil.

Enginyeria de programari. Módulos didácticos. Fundació per a la Universitat Oberta de Catalunya. Autores: Benet Campderrich Falgueras.

Fonaments de computadors I. Módulos didácticos. Fundació per a la Universitat Oberta de Catalunya, 1999. Autores: Pedro de Miguel Anasagasti, Santiago Rodríguez de la Fuente.

Fonaments de computadors II. Módulos didácticos. Fundació per a la Universitat Oberta de Catalunya, 1998. Autores: Pedro de Miguel Anasagasti, Julio Gutiérrez Ríos, Luis Pastor Pérez, José L. Pedraza Domínguez.

Estructura de la informacio. Módulos didácticos. Fundació per a la Universitat Oberta de Catalunya, 2001. Autores: Xavier Franch Gutiérrez, Xavier Burgués i Illa,Fatos Xhafa

Teoría de autómatas y lenguajes formales I. Módulos didácticos. Fundació per a la Universitat Oberta de Catalunya, 2003. Autores: Joan Vancells i Flotats, Enric Sesa Nogueras.

BIBLIOGRAFÍA DEL MICROCONTROLADOR PIC16F84

Microcontroladores Pic. Diseño práctico de aplicaciones. Segunda edición, 1999. Autores: José Mª Angulo Usategui, Ignacio Angulo Martínez.

Microcontroladores Pic. Diseño práctico de aplicaciones, segunda parte. Año 2000. Autores: José Mª Angulo Usategui, Susana Romero Yesa, Ignacio Angulo Martínez.

Especificaciones técnicas del microcontrolador PIC 16f84. Disponible como documento electrónico en formato pdf en la web de Microchip: http://www.microchip.com