Práctica 5

Implementación del microprocesador ICAI-RiSC-16

José Javier González Ortiz Lucía Montero Sanchis

30 de enero de 2015

Índice

1. Enunciado

La práctica ha consistido en la implantación del microprocesador ICAI-RiSC-16 que se describe en el documento "Arquitectura del microprocesador ICAI-RiSC-16", haciendo uso de la Unidad Aritmético-Lógica descrita en la práctica anterior.

A continuación se ha procedido a la implementación de los periféricos del micro: Un timer, el transmisor-receptor serie que se había diseñado en la Práctica 2, y un display alfanumérico.

2. Microprocesador ICAI-RiSC-16

La descripción estructural del microprocesador queda reflejada en el Código A.1. En el diseño se ha separado la Unidad de Control de la Ruta de Datos.

2.1. Unidad de Control

Es la máquina de estados del documento de la Arquitectura del ICAI-RiSC-16, reflejada en la Figura 2.1. Se implementa por medio del código A.2.

2.2. Ruta de Datos

Además de los relativos a los periféricos, los componentes que se han implementado son los siguientes:

- *Unidad Aritmético Lógica* - Se ha implementado la ALU diseñada en la práctica anterior con 16 bits. Queda definida en el código A.3.
- *Memoria RAM* - Se ha configurado parte de la memoria interna de la FPGA como memoria RAM para el almacenamiento de los datos. Se muestra en el código A.4.
- *Memoria ROM* - Parte de la memoria interna de la FPGA configurada como ROM para almacenar el programa. La ROM utilizada para probar el correcto funcionamiento del microprocesador se muestra en el código A.5.
- *Banco de Registros de Trabajo* - Banco de 8 registros de propósito general, de escritura síncrona y lectura asíncrona. El registro R0 contiene la constante cero. Se define en el código A.6
- *Registros* - Para la Ruta de Datos ha sido necesaria la implementación de varios registros de 16 bits. Para ello se ha hecho uso del registro de un número genérico de bits con enable, que se define en el código A.7.
  - Registro de Contador de Programa
  - Registro de Instrucción
  - Registro de la ALU
- Multiplexores - Se han implementado multiplexores, definidos con un número génerico de bits:
  - Multiplexor de 2 a 1, en el código A.8.
  - Multiplexor de 4 a 1, en el código A.9.

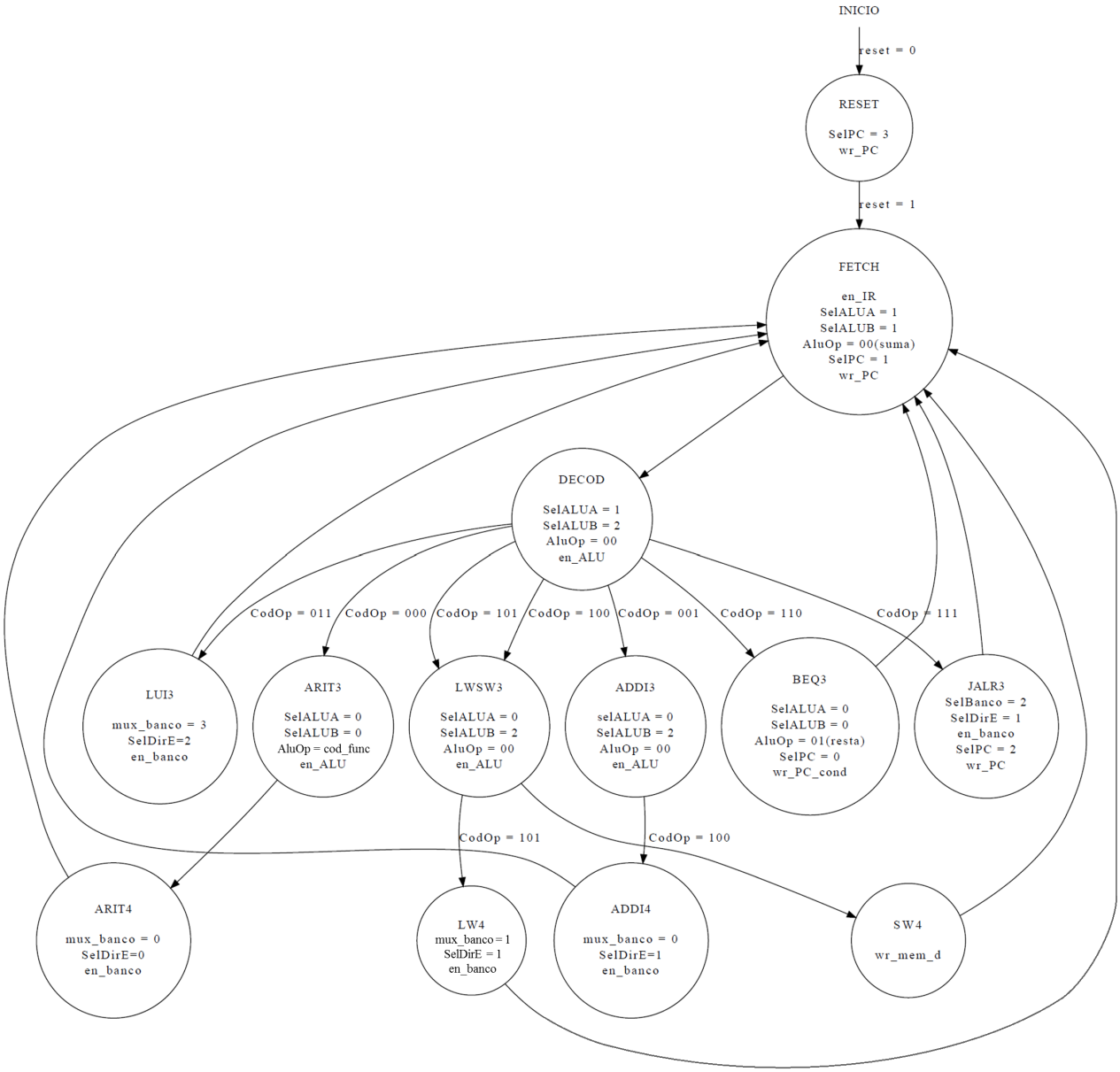

Figura 2.1: Unidad de Control del ICAI-RiSC-16

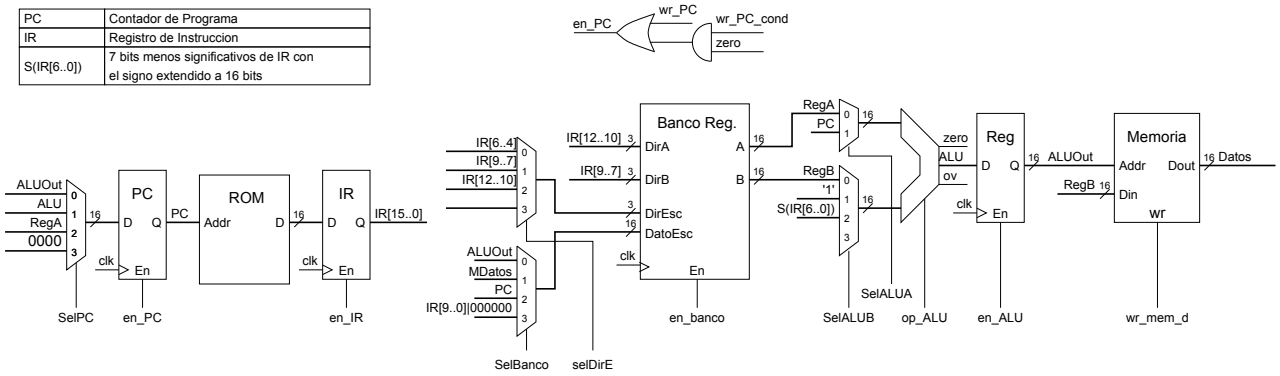

Figura 2.2: Diagrama de bloques de la ruta de datos del microprocesador, sin periféricos

2.3. Mapa de memoria

En la Figura 2.3 se muestra el mapa con la distrubución de memoria empleada en el diseño. Las zonas de memoria en color gris se califican como *shadow*, se deben a la forma de mapeo de memoria, y contienen copias de las *no shadow*. El criterio de diseño que se empleó fue el siguiente:

$$\text{Memoria RAM} \rightarrow \underbrace{A_{15}}_{0}\ \underbrace{A_{14}\text{–}A_8}_{X}\ \underbrace{A_7\text{–}A_0}_{\text{Dirección}}$$

$$\text{Periféricos} \rightarrow \underbrace{A_{15}}_{1}\ \underbrace{A_{14}\text{–}A_7}_{X}\ \underbrace{A_6\text{–}A_4}_{\text{Periférico}}\ \underbrace{A_3\text{–}A_0}_{\text{Registro}}$$

El mapa de memoria final que se ha implementado con los periféricos mostrados más adelante es el de la Tabla 2.1

Nombre	A_{15}	$A_{14} \cdots A_8$	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
RAM	0	x	DIRECCIÓN							
TIMER	1	x	x	1	0	0	x	x	DIR	
TXRX	1	x	x	1	0	1	x	x	DIR	
DISPLAY	1	x	x	1	1	0	x	DIR		
INT/LED	1	x	x	1	1	1	x	x	x	x

Tabla 2.1: Mapa de Memoria del ICAI-RiSC-16

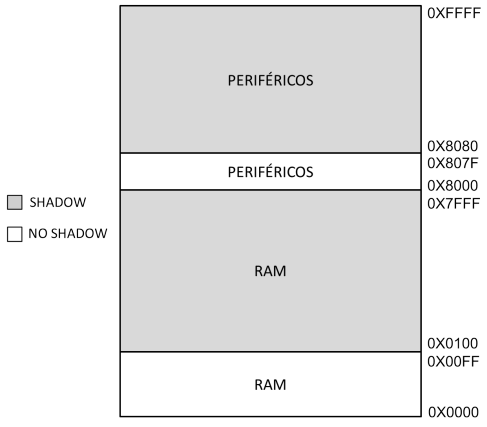

Figura 2.3: Mapa de memoria

3. Periféricos

Los periféricos que se han añadido al diseño del microprocesador son, como ya se mencionaba anteriormente:

- Timer
- Transmisor-Receptor Serie
- Display Alfanumérico
- Interruptores/LEDs

Por defecto, todos los bits utilizados de los registros de los periféricos que se indican a continuación son de *lectura y escritura*. Si en algún caso no es así, se indica junto al bit que corresponda.

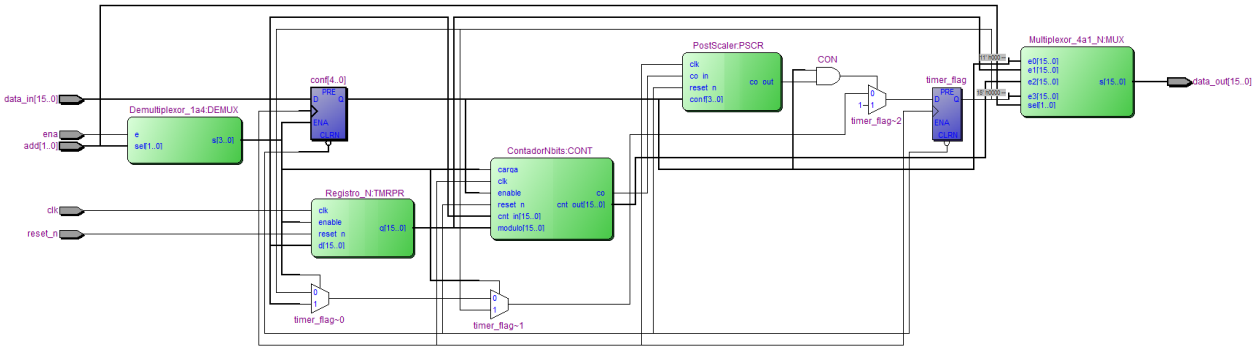

Figura 3.1: Diagrama de bloques del Timer

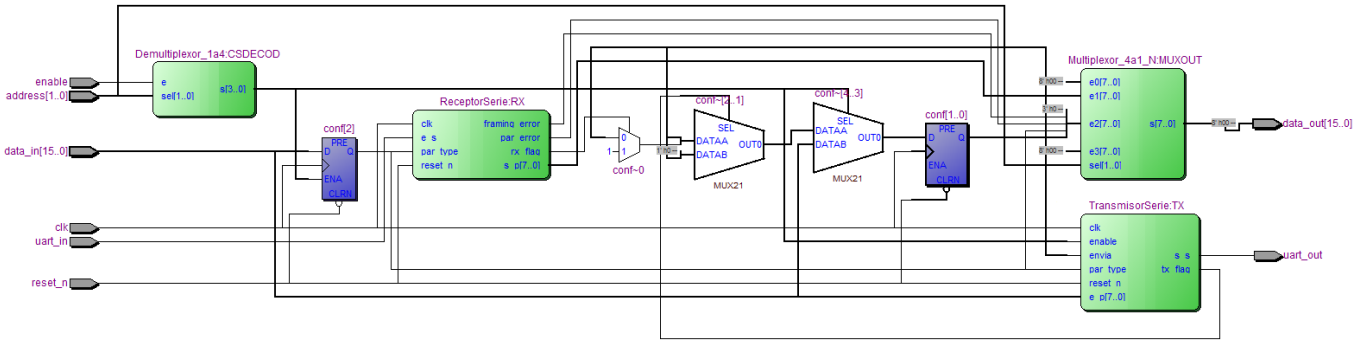

Figura 3.2: Diagrama de bloques del Transmisor-Receptor Serie

3.1. Timer

Nombre	Addr	Bit 15	Bit 14	...	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TMRCON	0	–	–	–	–	TPSCON				TMRON
TMRPR	1	Configuración del módulo del Contador del Timer								
TMRCNT	2	Valor del Contador del Timer								
TMRFL	3	–	–	–	–	–	–	–	–	TMRF

Tabla 3.1: Mapa de Registros del Timer

REGISTRO TMRCON

bits 15-5 **Sin implementar**: Leídos como 0

bits 4-1 **TPSCON**: Configuración del Post Scaler
`1111` = 1 : 32768 de Post Scaler
`1110` = 1 : 16384 de Post Scaler
...
`0010` = 1 : 4 de Post Scaler
`0001` = 1 : 2 de Post Scaler
`0000` = 1 : 1 de Post Scaler

bit 0 **TMRON**: Enable del Contador del Timer
`1` = El contador está activado
`0` = El contador está desactivado

REGISTRO TMRPR

bits 15-0 **TMRPR**: Módulo del Contador del Timer
Valor entre 0x0000 y 0xFFFF del módulo del contador del Timer

REGISTRO TMRCNT

bits 15-0 **TMRCNT**: Valor del Contador del Timer
Cuenta del Timer. Puede escribirse sobre este registro para que el contador descendente comience la cuenta desde un valor menor al módulo.

REGISTRO TMRFL

bits 15-1 **Sin implementar**: Leídos como 0

bit 0 **TMRF**: Flag del Timer
`1` = El timer ha terminado de contar
`0` = El timer aún no ha terminado de contar

3.2. Transmisor-Receptor Serie

Nombre	Addr	Bit 15	...	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TXREG	0	–	–	TXDAT							
RXREG	1	–	–	RXDAT							
TRREG	2	–	–	–	–	–	TRFE	TRPE	TRPAR	TXF	RXF

Tabla 3.2: Mapa de Registros del Transmisor-Receptor Serie

REGISTRO TXREG

bits 15-8 **Sin implementar**: Leídos como 0

bits 7-0 **TXDAT**: Entrada paralelo del transmisor serie (sólo escritura)
Bits a enviar

REGISTRO RXREG

bits 15-8 **Sin implementar**: Leídos como 0

bits 7-0 **RXDAT**: Salida paralelo del receptor serie (sólo lectura)
Bits recibidos

REGISTRO TRREG

bits 15-5 **Sin implementar**: Leídos como 0

bit 4 **TRFE**: Bit de error de Framing
1 = Ha habido un error de Framing
0 = No ha habido un error de Framing

bit 3 **TRPE**: Bit de error de Paridad
1 = Ha habido un error de Paridad
0 = No ha habido un error de Paridad

bit 2 **TRPAR**: Tipo de Paridad
1 = Paridad impar
0 = Paridad par

bit 1 **TXF**: Bit de estado de la transmisión
1 = Está transmitiendo, no puede iniciarse un nuevo envío
0 = Ya ha terminado de transmitir

bit 0 **RXF**: Bit de estado de la recepción
1 = Se ha recibido un bit
0 = No se ha recibido nada

3.3. Display Alfanumérico

Nombre	Addr	Bit 15	...	Bit 8	Bit 7	Bit 6	...	Bit 2	Bit 1	Bit 0
DS0	0	–	–	–	Contenido del Registro 0 del Display					
DS1	1	–	–	–	Contenido del Registro 1 del Display					
DS2	2	–	–	–	Contenido del Registro 2 del Display					
DS3	3	–	–	–	Contenido del Registro 3 del Display					
DS4	4	–	–	–	Contenido del Registro 4 del Display					
DS5	5	–	–	–	Contenido del Registro 5 del Display					
DS6	6	–	–	–	Contenido del Registro 6 del Display					
DS7	7	–	–	–	Contenido del Registro 7 del Display					

Tabla 3.3: Mapa de Registros del Display

REGISTRO DSx

bits 15-8 **Sin implementar**: Leídos como 0

bits 7-0 **DSx**: Contenido del registro x del display
Codificación en 16 segmentos de la constante a representarse en el Display x

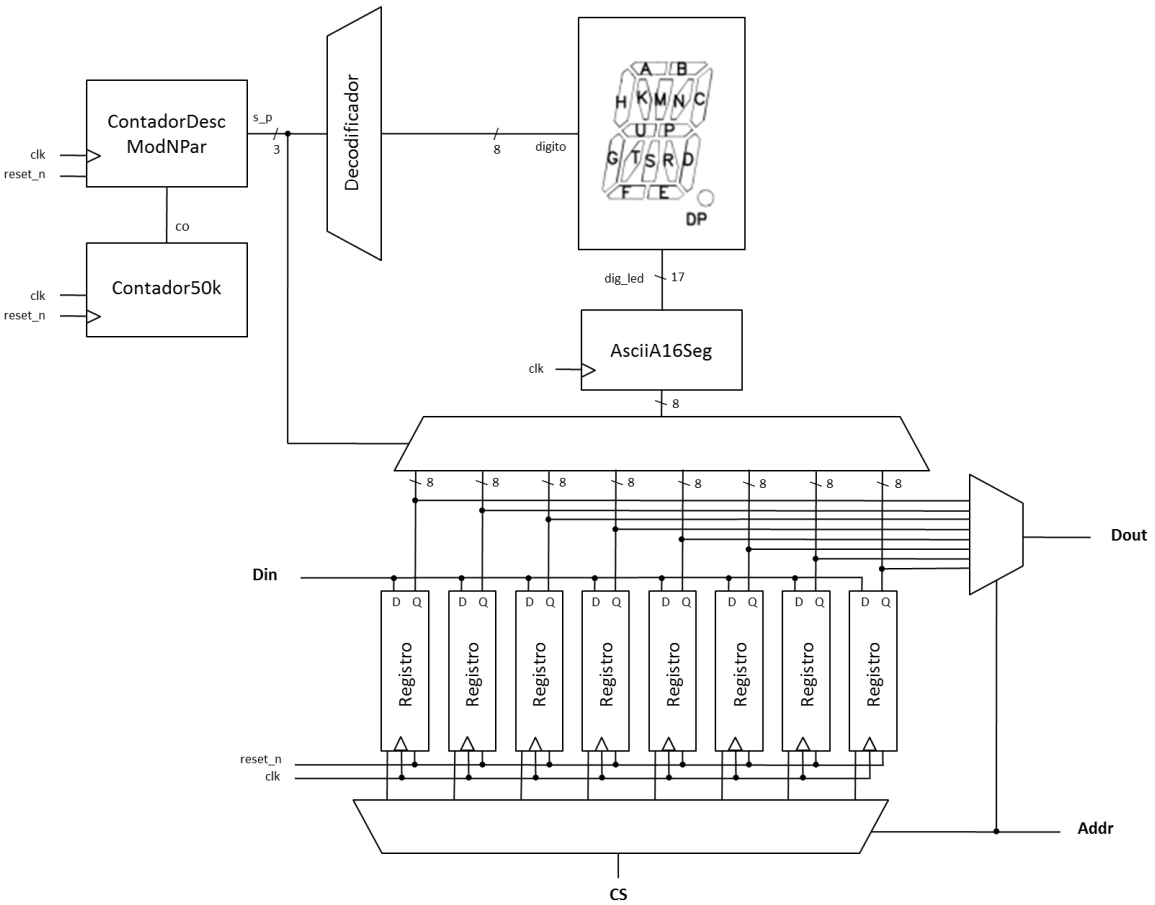

Figura 3.3: Diagrama de bloques del Display Alfanumérico

3.4. Interruptores/LEDs

Nombre	Bit 15	...	Bit 10	Bit 9	Bit 8	Bit 7	...	Bit 2	Bit 1	Bit 0
RLED	–	–	–	LEDDATA						
RINT	–	–	–	INTDATA						

Tabla 3.4: Mapa de Registros del periférico LEDs/Interruptores

REGISTRO RLED

bits 15-10 **Sin implementar**: Leídos como 0

bits 9-0 **LEDDATA**: Contenido del registro de los LEDs (sólo escritura)
Bits que se están mostrando en los LEDs

REGISTRO RINT

bits 15-10 **Sin implementar**: Leídos como 0

bits 9-0 **INTDATA**: Bits de los interruptores (sólo lectura)
Bits de los interruptores, que pueden cargarse para mostrarse en los LEDs

4. Simulación y Volcado

Para poder comprobar el correcto funcionamiento, se simularon los bloques de forma independiente, algunos habiendo sido ya simulados en prácticas anteriores. Para poder comprobar el correcto funcionamiento e interconexión de los periféricos y de cara a obtener un resultado visible del microprocesador, se elaboró un programa.

4.1. Programa Cronómetro

El programa, que se encuentra implementado en el Código C.1 define un cronómetro funcional con resolución de centésimas de segundo y de hasta una hora de longitud. Entre sus funcionalidades encontramos:

- El interruptor más bajo es empleado como RUN/STOP del cronómetro.
- Si se envía un carácter ASCII `t` por el puerto serie, el dispostivo lo reconoce, y responde envíando por el puerto serie una cadena de la forma MM:SS,CC indicando minutos, segundos y centésimas.
- Esta misma cadena se muestra continuamente en el display LED de 16 Segmentos y punto decimal.

El programa almacena las variables del reloj en representación ASCII en las primeras direcciones de la RAM. Continuamente comrpueba el estado del interruptor y lo copia al bit de estado del timer. El refresco del multiplexado es por hardware así que el software solamente comprueba los flags del Receptor Serie y del Timer y en caso de que estén activos los borra y actúa. Se ha definido una función para el envío serie para poder enviar todo el mensaje ASCII.

4.2. Simulación

Para poder comprobar el correcto funcionamiento del programa, se ha diseñado el *testbench* del Código D.1. Con dicho código se comprueban la práctica totalidad de los periféricos y de las instrucciones del ICAI-RiSC-16.

Como la simulación es muy compleja y se necesita mucho tiempo para poder mostrar todas las funcionalidades sólo se han capturado elementos importantes y relevantes de la misma. Como es de esperar se ha tenido que reducir los contadores del Display y del Transmisor receptor así como el periodo del Timer ya que debido a los largos tiempos no sería factible realizar una simulación.

Entre las capturas tomadas encontramos:

- Arranque desde Reset del ICAI-RiSC-16 (Figura 4.1)
- Inicialización de las Variables ASCII de la memoria RAM (Figura 4.2)
- Incremento de la variable de centesimas por un flag del Timer (Figura 4.3)
- Pregunta y respuesta por puerto serie del *timestamp* actual (Figura 4.4)
- Simulación global del programa del cronómetro (Figura 4.5)

4.3. Volcado

Tras un volcado inicial en el que se corría un código muy simple para comprobar la validez del microprocesador y obtener una simulación satisfactoria del código del cronómetro, se volcó dicho programa en la FPGA pudiendo comprobar el correcto funcionamiento del cronómetro.

Figura 4.1: Arranque desde Reset del ICAI-RiSC-16

En esta figura podemos apreciar el estado inicial del microprocesador. Como vemos prácticamente todo empieza a cero por el reset inicial, a excepción de la RAM que no tiene un valor inicial predefinido.
Se puede observar como avanzan el *Program Counter* y el *Instruction Register*, así como los tres/cuatro ciclos de operación por cada instrucción.
Como gran parte del código mostrado son inicializaciones de registros y variables vemos como va cambiando el banco de registros.
Hacer notar por último que el ánodo va cambiando para lograr la multiplexación del valor en los registros del display que empieza como todo apagado ya que para la LUT el 0 ASCII no tiene nada iluminado

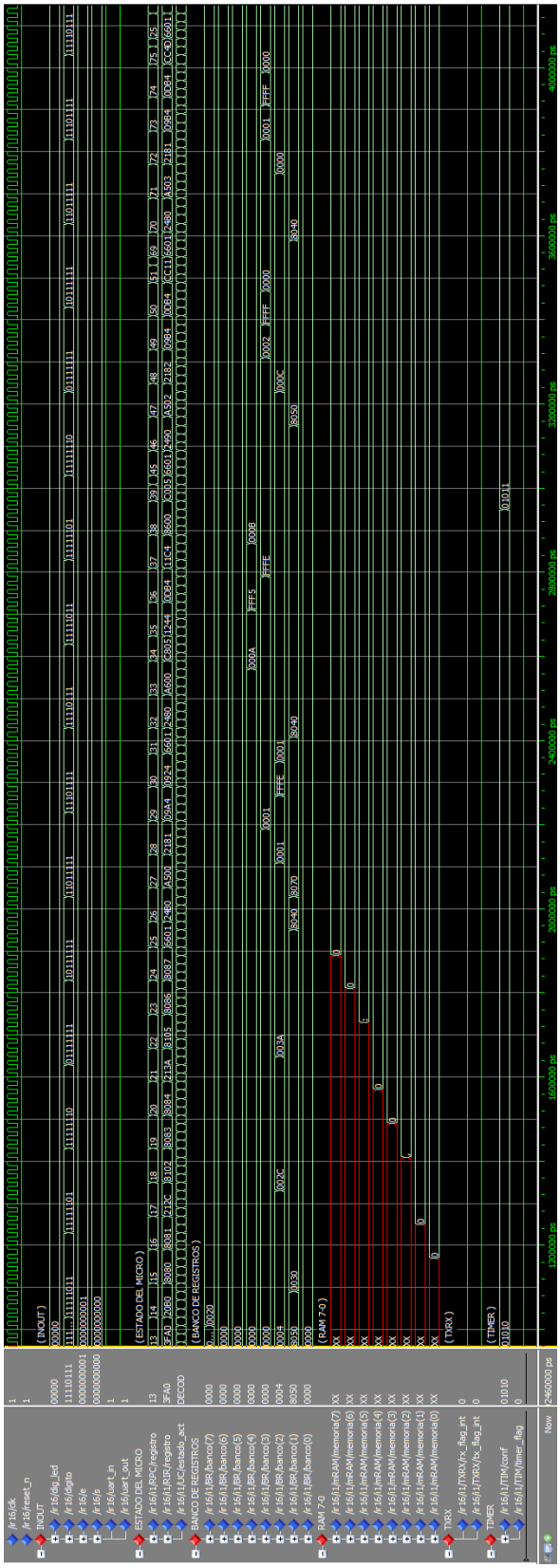

Figura 4.2: Inicialización de las Variables ASCII de la memoria RAM

En esta captura podemos ver el progresivo proceso de carga en la RAM de los valores ASCII que serán empleados para representar los valores de minutos, ssegundos y centésimas.

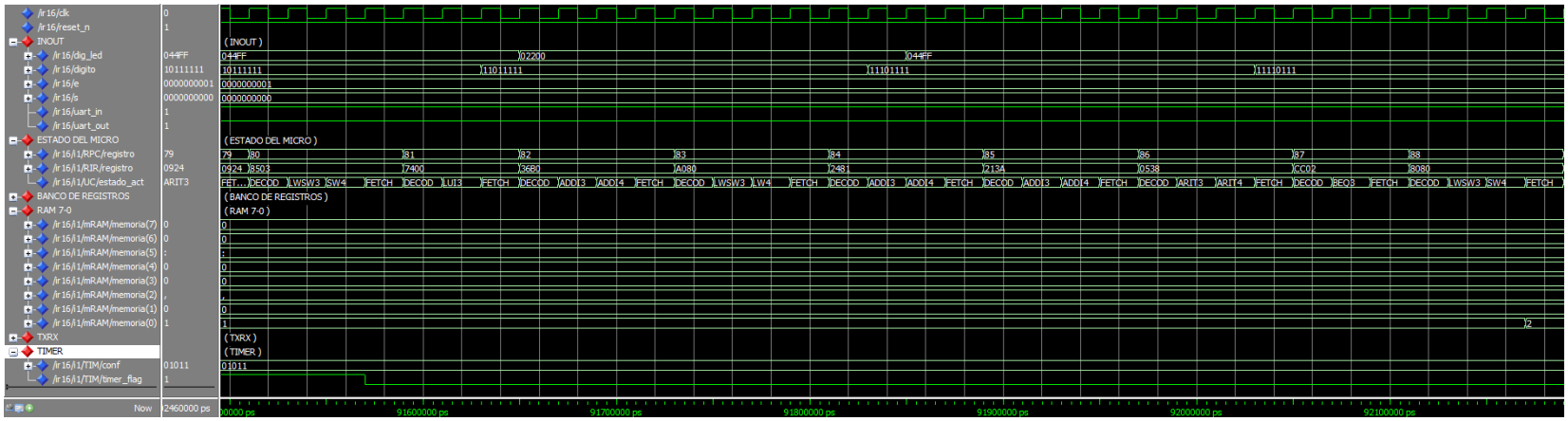

Figura 4.3: Incremento de la variable de centesimas por un flag del Timer

En este código vemos cómo una vez ha saltado el flag del timer y se ha detectado que está a 1, el programa lo borra e incrementa las centésimas y mira si hay desborde para unidades y decenas, en caso de que lo haya incrementará de igual forma segundos y minutos.

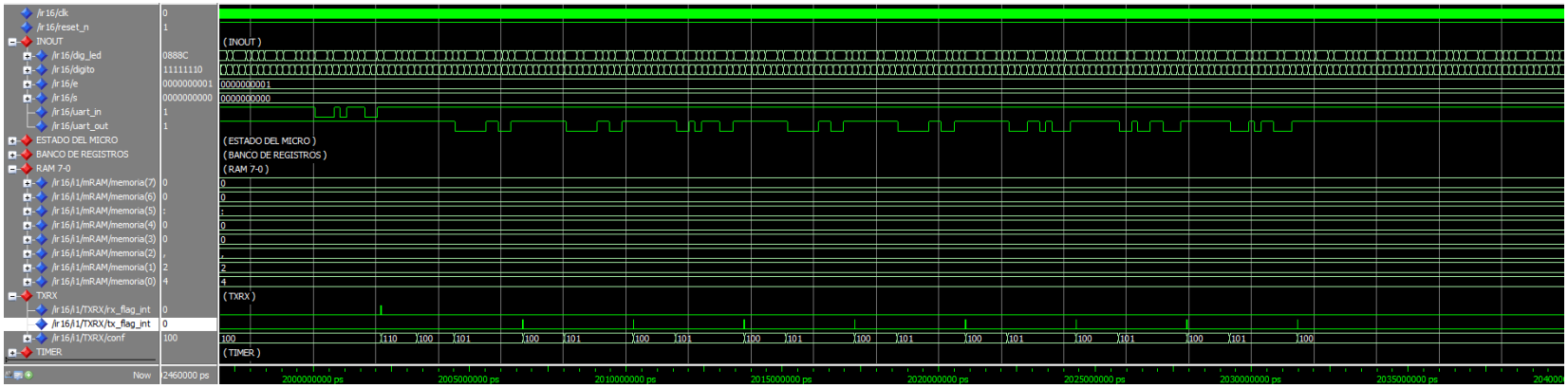

Figura 4.4: Pregunta y respuesta por puerto serie del *timestamp* actual

En esta figura se aprecia cómo al recibir una '`t`' ASCII se envía por el puerto serie el valor actual del cronómetro en formato MM:SS,CC

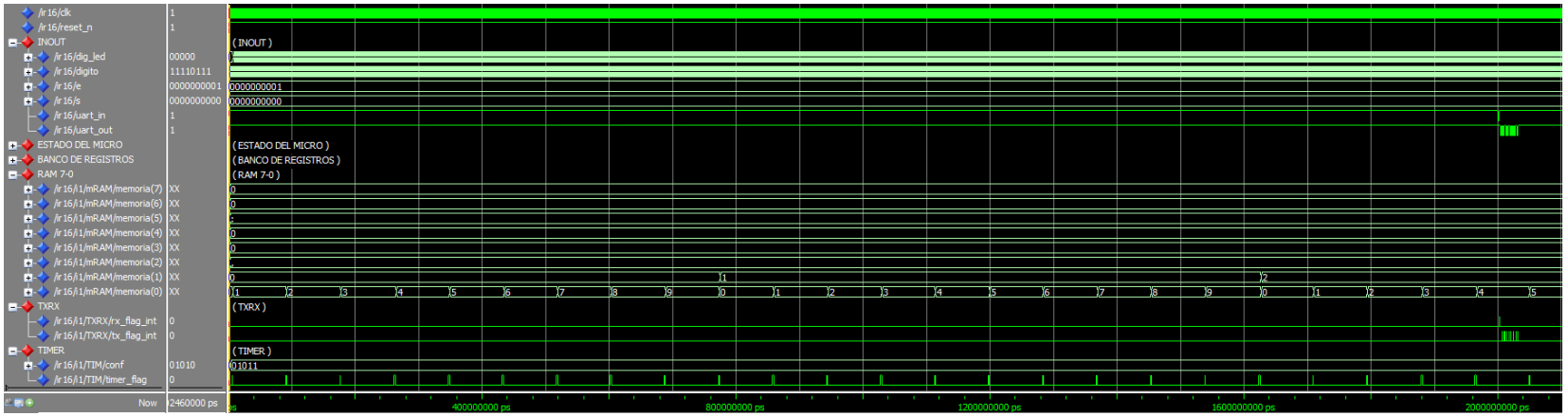

Figura 4.5: Simulación global del programa del cronómetro

Por último, esta figura da una visión global del programa elaborado pudiendo apreciar como cada vez que el flag timer se activa (cuando se alcanza el valor del post scaler), se incrementan las centésimas y se comprueban el resto. Como se puede comprobar a pesar de los 2 ms que se han empleado, no ha sido suficiente para comprobar el incremento de todas las posiciones.
Simulaciones con periodos más cortos permitían comprobar los cambios en minutos y segundos, pero no permitían simular correctamente ya que interferían con el transmisor y los displays.

5. Conclusiones

Como se ha podido comprobar el desarrollo de un microprocesador y más generalmente de un microcontrolador es una tarea compleja que se apoya en la evidente máxima de "*divide y vencerás*". A partir de descripciones de elementos tanto combinacionales (multiplexores, Sumadores, Multiplicadores...) como secuenciales (biestables, registros, memorias...) y la correcta interconexión de los mismos hemos obtenido un microprocesador completamente funcional.

Resulta muy importante pararnos a analizar el diseño, ya realizado para la arquitectura del microprocesador y ejercicio para la parte de los periféricos, dado que las decisiones sobre el diseño repercutirán tanto en la funcionalidad del sistema así como en las posibles implementaciones del mismo. Muchas veces durante el diseño fue necesario volver a atrás para poder reformular ciertas características debido a problemas encontrados.

Con todo ello hemos podido apreciar la complejidad de un sistema como el elaborado y la imperante necesidad de tener una simulación lo suficientemente exhaustiva que cubra todos los casos. En múltiples partes del desarrollo la simulación nos permitió detectar errores y fallas que en el volcado hubiesen sido imposibles de detectar.

Por último recalcar lo interesante y desafiante de la práctica que nos ha permitido obtener una profundidad en lo referente a un microprocesador RiSC lo suficientemente sencillo como para ser implementado en el tiempo dado, pero lo suficientemente complejo como para poder apreciar las diferentes características que lo componen.

A. Código del Microprocesador

A continuación se incluyen los códigos de implementación de los distintos bloques del microprocesador, así como el código estructural del mismo.

A.1. Código Estructural del Microprocesador ICAI-RiSC-16

Código A.1: Código estructural del Microprocesador

```
1
2  -- Jose Javier Gonzalez Ortiz
3  -- Lucia Montero Sanchis
4  -- 2014-11-02
5  -- ICAI_RiSC_16
6
7  --  DISPLAY ALFANUMERICO   --
8  --       MULTIPLEXADO      --
9
10 library IEEE;
11 use IEEE.std_logic_1164.all;
12 ---------------INPUT/OUTPUT------------------
13
14 entity ICAI_RiSC_16 is
15     port(
16         clk             : in    std_logic;
17         reset_n         : in    std_logic;
18         e               : in    std_logic_vector(9 downto 0);
19         s               : out   std_logic_vector(9 downto 0);
20         uart_in         : in    std_logic;
21         uart_out        : out   std_logic;
22         digito          : out   std_logic_vector(7  downto 0);
23         dig_led         : out   std_logic_vector(16 downto 0)
24         );
25
26 end ICAI_RiSC_16;
27
28 ---------------ARQUITECTURA------------------
29
30 architecture structural of ICAI_RiSC_16 is
31
32 -----------------------------------
33 --Declaracion de senales intermedias
34 -----------------------------------
35
36 --Senales
37     type stdlv16_vector is array (integer range <>) of std_logic_vector(15 downto 0);
38     --Auxiliares
39     signal zero_int         : std_logic;
40     signal ena_PC_int       : std_logic;
41
42     signal high_int         : std_logic_vector(15 downto 0);
43     signal sign_inm_int     : std_logic_vector(15 downto 0);
44
45     --Registros
46     signal IR_int           : std_logic_vector(15 downto 0);
47     signal PC_int           : std_logic_vector(15 downto 0);
48     signal data_IR_int      : std_logic_vector(15 downto 0);
49     signal data_PC_int      : std_logic_vector(15 downto 0);
```

```
50      signal ALU_A_int        : std_logic_vector(15 downto 0);
51      signal ALU_B_int        : std_logic_vector(15 downto 0);
52      signal ALU_res_int      : std_logic_vector(15 downto 0);
53      signal ALU_out_int      : std_logic_vector(15 downto 0);
54      signal regA_int         : std_logic_vector(15 downto 0);
55      signal regB_int         : std_logic_vector(15 downto 0);
56      signal dirEsc_int       : std_logic_vector(2  downto 0);
57      signal datoEsc_int      : std_logic_vector(15 downto 0);
58      signal op_alu_int       : std_logic_vector(3  downto 0);
59      signal MDatos_int       : std_logic_vector(15 downto 0);
60      signal Datos_int        : std_logic_vector(15 downto 0);
61
62      --Enables
63      signal ena_ir_int       : std_logic;
64      signal wr_PC_int        : std_logic;
65      signal wr_PC_cond_int   : std_logic;
66      signal ena_banco_int    : std_logic;
67      signal ena_ALU_int      : std_logic;
68      signal wr_d_int         : std_logic;
69      signal wr_mem_d_int     : std_logic;
70
71      --Sel multiplexor
72      signal sel_PC_int       : std_logic_vector(1 downto 0);
73      signal mux_banco_int    : std_logic_vector(1 downto 0);
74      signal sel_dirE_int     : std_logic_vector(1 downto 0);
75      signal sel_ALU_a_int    : std_logic;
76      signal sel_ALU_b_int    : std_logic_vector(1 downto 0);
77
78      signal e_int            : std_logic_vector(15 downto 0);
79      signal s_int            : std_logic_vector(15 downto 0);
80
81      --Perifericos
82      signal wr_per_d_int     : std_logic;
83      signal cs_int           : std_logic_vector(7 downto 0);
84      signal per_out_int      : stdlv16_vector  (6 downto 0);
85      signal PDatos_int       : std_logic_vector(15 downto 0);
86
87
88  ----------------------------
89  --Declaracion de componentes
90  ----------------------------
91
92  --Unidad de Control
93      component UnidadControl is
94          port(
95              --Externas
96              reset_n         : in    std_logic;
97              clk             : in    std_logic;
98              cod_op          : in    std_logic_vector(2 downto 0);
99              cod_func        : in    std_logic_vector(3 downto 0);
100
101             --Enables
102             ena_ir          : out   std_logic;
103             wr_PC           : out   std_logic;
104             wr_PC_cond      : out   std_logic;
105             ena_banco       : out   std_logic;
106             ena_ALU         : out   std_logic;
107             wr_d            : out   std_logic;
108
```

```
109             --Sel multiplexores
110             sel_PC          : out   std_logic_vector(1 downto 0);
111             mux_banco       : out   std_logic_vector(1 downto 0);
112             sel_dirE        : out   std_logic_vector(1 downto 0);
113             sel_ALU_a       : out   std_logic;
114             sel_ALU_b       : out   std_logic_vector(1 downto 0);
115
116             op_alu          : out   std_logic_vector(3 downto 0));
117
118     end component;
119
120 --Unidad Aritmetico Logica
121     component ALU is
122         generic(
123             n_bits   :  integer := 16);
124         port(
125             a       :   in      std_logic_vector(n_bits-1 downto 0);
126             b       :   in      std_logic_vector(n_bits-1 downto 0);
127             op_alu  :   in      std_logic_vector(3 downto 0);
128             res     :   out     std_logic_vector(n_bits-1 downto 0);
129             co      :   out     std_logic;
130             z       :   out     std_logic;
131             ov      :   out     std_logic);
132     end component;
133
134
135 --Memoria RAM
136     component RAM is
137         generic(
138             n_bits : integer := 16;
139             n_dir  : integer := 64;
140             n_add  : integer := 6); --Debe ser mayor o igual a log(2,n_dir)
141         port(
142             clk         : in    std_logic;
143             wr          : in    std_logic;
144             addr        : in    std_logic_vector(n_add-1 downto 0);
145             d_in        : in    std_logic_vector(n_bits-1 downto 0);
146             d_out       : out   std_logic_vector(n_bits-1 downto 0));
147
148     end component;
149
150 --Memoria ROM
151     component ROM is
152       port(
153         clk: in std_logic; -- La ROM es sincrona
154         en_pc: in std_logic; -- Y tiene un enable
155         dir: in std_logic_vector(7 downto 0); -- Bus de direcciones
156         dat: out std_logic_vector(15 downto 0) ); -- Salida de datos
157     end component;
158
159 --Banco de Registros de Trabajo
160     component BancoReg is
161         generic(
162             n_bits : integer := 16);
163         port(
164             reset_n     : in    std_logic;
165             clk         : in    std_logic;
166             enable      : in    std_logic;
167             dirA        : in    std_logic_vector(2 downto 0);
```

```
168             dirB        : in    std_logic_vector(2 downto 0);
169             dirEsc      : in    std_logic_vector(2 downto 0);
170             datoEsc     : in    std_logic_vector(n_bits-1 downto 0);
171             a           : out   std_logic_vector(n_bits-1 downto 0);
172             b           : out   std_logic_vector(n_bits-1 downto 0));
173
174     end component;
175
176 --Registro Generico
177     component Registro_N is
178         generic(
179             n_bits : integer := 16);
180         port(
181             reset_n     : in    std_logic;
182             clk         : in    std_logic;
183             enable      : in    std_logic;
184             d           : in    std_logic_vector(n_bits-1 downto 0);
185             q           : out   std_logic_vector(n_bits-1 downto 0));
186
187     end component;
188
189 --Multiplexor de 8 entradas
190     component Multiplexor_8a1_N is
191         generic(
192             n_bits      : integer := 16);
193         port(
194             e0  : in    std_logic_vector(n_bits-1 downto 0);
195             e1  : in    std_logic_vector(n_bits-1 downto 0);
196             e2  : in    std_logic_vector(n_bits-1 downto 0);
197             e3  : in    std_logic_vector(n_bits-1 downto 0);
198             e4  : in    std_logic_vector(n_bits-1 downto 0);
199             e5  : in    std_logic_vector(n_bits-1 downto 0);
200             e6  : in    std_logic_vector(n_bits-1 downto 0);
201             e7  : in    std_logic_vector(n_bits-1 downto 0);
202             sel : in    std_logic_vector(2 downto 0);
203             s   : out   std_logic_vector(n_bits-1 downto 0));
204
205     end component;
206
207 --Multiplexor de 4 entradas
208     component Multiplexor_4a1_N is
209         generic(
210             n_bits      : integer := 16);
211         port(
212             e0  : in    std_logic_vector(n_bits-1 downto 0);
213             e1  : in    std_logic_vector(n_bits-1 downto 0);
214             e2  : in    std_logic_vector(n_bits-1 downto 0);
215             e3  : in    std_logic_vector(n_bits-1 downto 0);
216             sel : in    std_logic_vector(1 downto 0);
217             s   : out   std_logic_vector(n_bits-1 downto 0));
218
219     end component;
220
221 --Multiplexor de 2 entradas
222     component Multiplexor_2a1_N is
223         generic(
224             n_bits      : integer := 16);
225         port(
226             e0  : in    std_logic_vector(n_bits-1 downto 0);
```

```
227             e1   : in    std_logic_vector(n_bits-1 downto 0);
228             sel  : in    std_logic;
229             s    : out   std_logic_vector(n_bits-1 downto 0));
230 
231     end component;
232 
233 --Demultiplexor de 8 Salidas individual para el descodificador
234     component Demultiplexor_1a8 is
235         port(
236             e   : in     std_logic;
237             sel : in     std_logic_vector(2 downto 0);
238             s   : out    std_logic_vector(7 downto 0));
239 
240     end component;
241 
242 ---------------------PERIFERICOS-------------------------
243 --Display Alfanumerico
244     component DisplayAlfanumerico is
245         port(
246             clk             : in    std_logic;
247             reset_n         : in    std_logic;
248             enable          : in    std_logic;
249             address         : in    std_logic_vector(2  downto 0);
250             data_in         : in    std_logic_vector(15 downto 0);
251             data_out        : out   std_logic_vector(15 downto 0);
252             digito          : out   std_logic_vector(7  downto 0);
253             dig_led         : out   std_logic_vector(16 downto 0)
254             );
255     end component;
256 
257 --Transmisor Receptor Serie
258     component TransmisorReceptorSerie is
259         port(
260             clk             : in    std_logic;
261             reset_n         : in    std_logic;
262             enable          : in    std_logic;
263             address         : in    std_logic_vector(1  downto 0);
264             data_in         : in    std_logic_vector(15 downto 0);
265             data_out        : out   std_logic_vector(15 downto 0);
266             uart_in         : in    std_logic;
267             uart_out        : out   std_logic);
268 
269     end component;
270 
271 --Timer de 16 Bits con POSTscaler de 4 bits
272     component Timer is
273         port(
274             clk             : in    std_logic;
275             reset_n         : in    std_logic;
276             ena             : in    std_logic;
277             add             : in    std_logic_vector(1 downto 0); -- 0-TMRCON, 1-TMRPR
    , 2-TMRINI, 3-TMRCNT
278             data_in         : in    std_logic_vector(15 downto 0);
279             data_out        : out   std_logic_vector(15 downto 0));
280 
281     end component;
282 
283 
284 ----------DESCRIPCION ESTRUCTURAL-----------------
```

```
285 begin -- structural
286
287 --Enable del PC condicional
288 ena_PC_int   <= wr_PC_int or (wr_PC_cond_int and zero_int);
289
290 --Extension de signo del literal del codigo de instrucion
291 sign_inm_int(15 downto 7) <= (others => IR_int(6));
292 sign_inm_int(6 downto 0)  <= IR_int(6 downto 0);
293 --Literal de LUI
294 high_int     <= IR_int(9 downto 0) & "000000";
295
296 --Multiplexor del CS de RAM/Perifericos
297 wr_mem_d_int    <= wr_d_int and (not ALU_out_int(15));
298 wr_per_d_int    <= wr_d_int and ALU_out_int(15);
299
300 per_out_int(3 downto 0) <= (others => (others =>'0'));
301
302 --Limite de IO a 10 bits (FPGA)
303 e_int <= "000000"&e;
304 s <= s_int(9 downto 0);
305
306
307 --- Instancias ---
308
309 --Unidad de Control
310     UC :  UnidadControl
311         port map(
312             --Externas
313             reset_n         => reset_n,
314             clk             => clk,
315             cod_op          => IR_int(15 downto 13),
316             cod_func        => IR_int(3 downto 0),
317
318             --Enables=>
319             ena_ir          => ena_ir_int,
320             wr_PC           => wr_PC_int,
321             wr_PC_cond      => wr_PC_cond_int,
322             ena_banco       => ena_banco_int,
323             ena_ALU         => ena_ALU_int,
324             wr_d            => wr_d_int,
325
326             --Sel mux=>
327             sel_PC          => sel_PC_int,
328             mux_banco       => mux_banco_int,
329             sel_dirE        => sel_dirE_int,
330             sel_ALU_a       => sel_ALU_a_int,
331             sel_ALU_b       => sel_ALU_b_int,
332             op_alu          => op_alu_int);
333
334 --Unidad Aritmetico Logica
335     ALU16 : ALU
336         port map(
337             a       => ALU_A_int,
338             b       => ALU_B_int,
339             op_alu  => op_alu_int,
340             res     => ALU_res_int,
341             co      => open,            --Y el status?
342             z       => zero_int,
343             ov      => open);           --Y el status?
```

```
344
345
346   --Memoria RAM
347       mRAM : RAM
348           port map(
349               clk          => clk,
350               wr           => wr_mem_d_int,
351               addr         => ALU_out_int(5 downto 0),
352               d_in         => regB_int,
353               d_out        => MDatos_int);
354
355   --Memoria ROM
356       mROM : ROM
357         port map(
358           clk              => clk,
359           en_pc            => ena_PC_int,
360           dir              => data_PC_int(7 downto 0),
361           dat              => data_IR_int);
362
363   --Banco de Registros de Trabajo
364       BR : BancoReg
365           port map(
366               reset_n      => reset_n,
367               clk          => clk,
368               enable       => ena_banco_int,
369               dirA         => IR_int(12 downto 10),
370               dirB         => IR_int(9 downto 7),
371               dirEsc       => dirEsc_int,
372               datoEsc      => datoEsc_int,
373               a            => regA_int,
374               b            => regB_int);
375
376   --Registros
377       --Program Counter
378           RPC : Registro_N
379               port map(
380                   reset_n      => reset_n,
381                   clk          => clk,
382                   enable       => ena_PC_int,
383                   d            => data_PC_int,
384                   q            => PC_int);
385
386       --Instruction Register
387           RIR : Registro_N
388               port map(
389                   reset_n      => reset_n,
390                   clk          => clk,
391                   enable       => ena_ir_int,
392                   d            => data_IR_int,
393                   q            => IR_int);
394
395       --ALU Register
396           RALU : Registro_N
397               port map(
398                   reset_n      => reset_n,
399                   clk          => clk,
400                   enable       => ena_ALU_int,
401                   d            => ALU_res_int,
402                   q            => ALU_out_int);
```

```
403
404 --Multiplexor para las salidas de datos de los perifericos
405     MUXOUT: Multiplexor_8a1_N
406         generic map(
407             n_bits      => 16)
408         port map(
409             e0          => per_out_int(0), --SFR
410             e1          => per_out_int(1), --
411             e2          => per_out_int(2), --
412             e3          => per_out_int(3), --
413             e4          => per_out_int(4), --TIMER
414             e5          => per_out_int(5), --TXRX
415             e6          => per_out_int(6), --DISPLAY
416             e7          => e_int,          --INTERRUPTORES
417             sel         => ALU_out_int(6 downto 4),
418             s           => PDatos_int);
419
420 --Descodificador
421     DESC : Demultiplexor_1a8
422         port map(
423             e           => wr_per_d_int,
424             sel         => ALU_out_int(6 downto 4),
425             s           => cs_int);
426
427 --Multiplexores de 4 entradas
428
429     --Multiplexor de PC
430         MPC : Multiplexor_4a1_N
431             port map(
432                 e0          => ALU_out_int,
433                 e1          => ALU_res_int,
434                 e2          => regA_int,
435                 e3          => X"0000",
436                 sel         => sel_PC_int,
437                 s           => data_PC_int);
438
439     --Multiplexor de DirEsc
440         MDI : Multiplexor_4a1_N
441             generic map(
442                 n_bits      => 3)
443             port map(
444                 e0          => IR_int(6 downto 4),
445                 e1          => IR_int(9 downto 7),
446                 e2          => IR_int(12 downto 10),
447                 e3          => "000",
448                 sel         => sel_dirE_int,
449                 s           => dirEsc_int);
450
451     --Multiplexor de DatoEsc
452         MDA : Multiplexor_4a1_N
453             port map(
454                 e0          => ALU_out_int,
455                 e1          => Datos_int,
456                 e2          => PC_int,
457                 e3          => high_int,
458                 sel         => mux_banco_int,
459                 s           => datoEsc_int);
460
461     --Multiplexor de RegB
```

```
462         MRB : Multiplexor_4a1_N
463             port map(
464                 e0          => regB_int,
465                 e1          => X"0001",
466                 e2          => sign_inm_int,
467                 e3          => X"0000",
468                 sel         => sel_ALU_b_int,
469                 s           => ALU_B_int);
470 
471 --Multiplexores de 2 entradas RegA
472     --Multiplexor de PC
473         MRA : Multiplexor_2a1_N
474             port map(
475                 e0          => regA_int,
476                 e1          => PC_int,
477                 sel         => sel_ALU_a_int,
478                 s           => ALU_A_int);
479 
480     --Multiplexor de RAM/Perifericos
481         MRD : Multiplexor_2a1_N
482             port map(
483                 e0          => MDatos_int,
484                 e1          => PDatos_int,
485                 sel         => ALU_out_int(15),
486                 s           => Datos_int);
487 
488 ------- PERIFERICOS ---------
489 
490 --LED Register
491     RLED : Registro_N
492         port map(
493             reset_n     => reset_n,
494             clk         => clk,
495             enable      => cs_int(7),
496             d           => regB_int,
497             q           => s_int);
498 
499 --Display Alfanumerico
500     DISP : DisplayAlfanumerico
501         port map(
502             clk         => clk,
503             reset_n     => reset_n,
504             enable      => cs_int(6),
505             address     => ALU_out_int(2 downto 0),
506             data_in     => regB_int,
507             data_out    => per_out_int(6),
508             digito      => digito,
509             dig_led     => dig_led);
510 
511 --TransmisorReceptorSerie
512     TXRX : TransmisorReceptorSerie
513         port map(
514             clk         => clk,
515             reset_n     => reset_n,
516             enable      => cs_int(5),
517             address     => ALU_out_int(1 downto 0),
518             data_in     => regB_int,
519             data_out    => per_out_int(5),
520             uart_in     => uart_in,
```

```
521             uart_out    => uart_out);
522 --Timer 16 Bits
523     TIM : Timer
524         port map(
525             clk         => clk,
526             reset_n     => reset_n,
527             ena         => cs_int(4),
528             add         => ALU_out_int(1 downto 0),
529             data_in     => regB_int,
530             data_out    => per_out_int(4));
531
532 end structural;
```

A.2. Unidad de Control

Código A.2: Código que implementa la Unidad de Control del ICAI-RiSC-16

```
1  -- Jose Javier Gonzalez Ortiz
2  -- Lucia Montero Sanchis
3  -- 2014-11-02
4  -- UnidadControl
5
6  library IEEE;
7  use IEEE.std_logic_1164.all;
8
9  entity UnidadControl is
10   port(
11     --Externas
12     reset_n      : in  std_logic;
13     clk        : in  std_logic;
14     cod_op       : in  std_logic_vector(2 downto 0);
15     cod_func     : in  std_logic_vector(3 downto 0);
16
17     --Enables
18     ena_ir       : out   std_logic;
19     wr_PC      : out   std_logic;
20     wr_PC_cond     : out   std_logic;
21     ena_banco      : out   std_logic;
22     ena_ALU      : out   std_logic;
23     wr_d       : out   std_logic;
24
25     --Sel multiplexores
26     sel_PC       : out   std_logic_vector(1 downto 0);
27     mux_banco    : out   std_logic_vector(1 downto 0);
28     sel_dirE     : out   std_logic_vector(1 downto 0);
29     sel_ALU_a      : out   std_logic;
30     sel_ALU_b      : out   std_logic_vector(1 downto 0);
31
32     op_alu       : out   std_logic_vector(3 downto 0));
33
34 end UnidadControl;
35
36 architecture behavioral of UnidadControl is
37
38   type t_estados is (RESET,FETCH,DECOD,LUI3,ARIT3,ARIT4,BEQ3,ADDI3,ADDI4,LWSW3,LW4,SW4
       ,JALR3);
39   signal estado_act, estado_sig : t_estados;
40
41 begin -- behavioral
42
43   VarEstado : process(clk, reset_n)
44   begin
45
46     if reset_n = '0' then
47       estado_act <= RESET;
48     elsif  clk'event and clk = '1' then
49       estado_act <= estado_sig;
50     end if;
51   end process VarEstado;
52
53   TransicionEstado : process(estado_act,reset_n,cod_op,cod_func)
```

```
54    begin
55
56      estado_sig <= estado_act;
57
58      case estado_act is
59        when RESET =>
60          if reset_n = '1' then
61            estado_sig <= FETCH;
62          end if;
63
64        when FETCH =>
65          estado_sig <= DECOD;
66
67        when DECOD =>
68          case cod_op is
69            when "000" =>
70              estado_sig <= ARIT3;
71            when "001" =>
72              estado_sig <= ADDI3;
73            when "011" =>
74              estado_sig <= LUI3;
75            when "100" =>
76              estado_sig <= LWSW3;
77            when "101" =>
78              estado_sig <= LWSW3;
79            when "110" =>
80              estado_sig <= BEQ3;
81            when "111" =>
82              estado_sig <= JALR3;
83            when others =>
84              estado_sig <= RESET;
85          end case;
86
87        when LUI3 =>
88          estado_sig <= FETCH;
89
90        when ARIT3 =>
91          estado_sig <= ARIT4;
92
93        when ARIT4 =>
94          estado_sig <= FETCH;
95
96        when BEQ3 =>
97          estado_sig <= FETCH;
98
99        when ADDI3 =>
100         estado_sig <= ADDI4;
101
102       when ADDI4 =>
103         estado_sig <= FETCH;
104
105       when LWSW3 =>
106         if cod_op = "101" then
107           estado_sig <= LW4;
108         elsif cod_op = "100" then
109           estado_sig <= SW4;
110         else
111           estado_sig <= RESET;
112         end if;
```

```
113
114         when LW4 =>
115           estado_sig <= FETCH;
116
117         when SW4 =>
118           estado_sig <= FETCH;
119
120         when JALR3 =>
121           estado_sig <= FETCH;
122
123         when others =>
124           estado_sig <= RESET;
125
126       end case;
127     end process TransicionEstado;
128
129     Salidas : process(estado_act,cod_func)
130     begin
131       --Enables
132       ena_ir      <= '0';
133       wr_PC     <= '0';
134       wr_PC_cond    <= '0';
135       ena_banco     <= '0';
136       ena_ALU     <= '0';
137       wr_d      <= '0';
138
139       --Sel multiplex
140       sel_PC      <= "00";
141       mux_banco   <= "00";
142       sel_dirE    <= "00";
143       sel_ALU_a     <= '0';
144       sel_ALU_b     <= "00";
145
146       op_alu      <= "0000";
147
148       case estado_act is
149         when RESET =>
150           sel_PC    <= "11";
151           wr_PC     <= '1';
152
153         when FETCH =>
154           ena_ir    <= '1';
155           sel_ALU_a   <= '1';
156           sel_ALU_b   <= "01";
157           op_alu    <= "0000";
158           sel_PC    <= "01";
159           wr_PC     <= '1';
160
161         when DECOD =>
162           sel_ALU_a   <= '1';
163           sel_ALU_b   <= "10";
164           op_alu    <= "0000";
165           ena_ALU   <= '1';
166
167         when LUI3 =>
168           mux_banco   <= "11";
169           sel_dirE  <= "10";
170           ena_banco   <= '1';
171
```

```
172         when ARIT3 =>
173           sel_ALU_a   <= '0';
174           sel_ALU_b   <= "00";
175           op_alu    <= cod_func;
176           ena_ALU   <= '1';
177 
178         when ARIT4 =>
179           mux_banco   <= "00";
180           sel_dirE    <= "00";
181           ena_banco   <= '1';
182 
183         when BEQ3 =>
184           sel_ALU_a   <= '0';
185           sel_ALU_b   <= "00";
186           op_alu    <= "0001";
187           sel_PC    <= "00";
188           wr_PC_cond  <= '1';
189 
190         when ADDI3 =>
191           sel_ALU_a   <= '0';
192           sel_ALU_b   <= "10";
193           op_alu    <= "0000";
194           ena_ALU   <= '1';
195 
196         when ADDI4 =>
197           mux_banco   <= "00";
198           sel_dirE  <= "01";
199           ena_banco   <= '1';
200 
201         when LWSW3 =>
202           sel_ALU_a   <= '0';
203           sel_ALU_b   <= "10";
204           op_alu    <= "0000";
205           ena_ALU   <= '1';
206 
207         when LW4 =>
208           mux_banco   <= "01";
209           sel_dirE  <= "01";
210           ena_banco   <= '1';
211 
212         when SW4 =>
213           wr_d  <= '1';
214 
215         when JALR3 =>
216           mux_banco   <= "10";
217           sel_dirE  <= "01";
218           ena_banco   <= '1';
219           sel_PC    <= "10";
220           wr_PC     <= '1';
221 
222         when others =>
223           null;
224 
225       end case;
226     end process Salidas;
227 
228 end behavioral;
```

A.3. Unidad Aritmético Lógica

Código A.3: Código que define la ALU de N bits

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-09-22
-- ALU

-- UNIDAD ARITMETICO-LOGICA --
--          DE N BITS       --

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


---------------INPUT/OUTPUT------------------

entity ALU is
    generic(
        n_bits   :  integer := 16);
    port(
        a       :   in      std_logic_vector(n_bits-1 downto 0);
        b       :   in      std_logic_vector(n_bits-1 downto 0);
        op_alu  :   in      std_logic_vector(3 downto 0);
        res     :   out     std_logic_vector(n_bits-1 downto 0);
        co      :   out     std_logic;
        z       :   out     std_logic;
        ov      :   out     std_logic);
end ALU;

---------------ARQUITECTURA------------------

architecture structural of ALU is

-----------------------------------
--Declaracion de senales intermedias
-----------------------------------

constant zero : std_logic_vector(n_bits-1 downto 0) := (others => '0');

signal s_int : std_logic_vector(n_bits-1 downto 0);
signal p_int : std_logic_vector(n_bits-1 downto 0);
signal ov_sr_int : std_logic;
signal ov_p_int  : std_logic;
signal z_int : std_logic;
signal ULT_int : std_logic;

----------------------------
--Declaracion de componentes
----------------------------

--SumadorRestadorNBits
    component SumadorRestadorNBits is
        generic(
            n_bits : integer := 16);
        port (
```

```
55              s_r      :  in  std_logic;
56              a        :  in  std_logic_vector(n_bits-1 downto 0);
57              b        :  in  std_logic_vector(n_bits-1 downto 0);
58              s        :  out std_logic_vector(n_bits-1 downto 0);
59              overflow :  out std_logic;
60              co       :  out std_logic);
61
62      end component;
63  --MultiplicadorNBits
64      component MultiplicadorNBits is
65          generic(
66              n_bits : integer := 16);
67          port (
68              s_u      :  in  std_logic;
69              a        :  in  std_logic_vector(n_bits-1 downto 0);
70              b        :  in  std_logic_vector(n_bits-1 downto 0);
71              p        :  out std_logic_vector(n_bits-1 downto 0);
72              overflow :  out std_logic);
73
74      end component;
75
76
77  ----------DESCRIPCION ESTRUCTURAL------------------
78  begin -- structural
79
80      ULT_int <=  '1' when unsigned(a) < unsigned(b) else
81                  '0';
82
83      res <=  s_int                                  when op_alu = "0000" else
84              s_int                                  when op_alu = "0001" else
85              p_int                                  when op_alu = "0010" else
86              p_int                                  when op_alu = "0011" else
87              a nand b                               when op_alu = "0100" else
88              a(n_bits-2 downto 0)&'0'               when op_alu = "0101" else
89              a(n_bits-1)&a(n_bits-1 downto 1)       when op_alu = "0110" else
90              '0'&a(n_bits-1 downto 1)               when op_alu = "0111" else
91              (0 => ULT_int, others=>'0')            when op_alu = "1000" else
92              (others => '-');
93
94      ov <=   ov_sr_int when op_alu = "0000" else
95              ov_sr_int when op_alu = "0001" else
96              ov_p_int  when op_alu = "0010" else
97              ov_p_int  when op_alu = "0011" else
98              '0';
99
100     z   <= '1' when (s_int = zero) and (op_alu(3 downto 1) = "000") else
101            '0';
102
103
104 --Instancias
105     SRN : SumadorRestadorNBits
106         generic map(
107             n_bits => n_bits)
108         port map(
109             s_r         => op_alu(0),
110             a           => a,
111             b           => b,
112             s           => s_int,
113             overflow    => ov_sr_int,
```

```
114             co          => co);
115
116     MUL : MultiplicadorNBits
117         generic map(
118             n_bits => n_bits)
119         port map(
120             s_u         => op_alu(0),
121             a           => a,
122             b           => b,
123             p           => p_int,
124             overflow    => ov_p_int);
125
126 end structural;
```

A.4. Memoria RAM

Código A.4: Código que implementa la Memoria RAM

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-11-02
-- RAM.vhd

-- MEmoeia RAM de D direcciones de memoria de N bits cada una
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
------ Entidad ------

entity RAM is
    generic(
        n_bits : integer := 16;
        n_dir  : integer := 64;
        n_add  : integer := 6); --Debe ser mayor o igual a log(2,n_dir)
    port(
        clk         : in    std_logic;
        wr          : in    std_logic;
        addr        : in    std_logic_vector(n_add-1 downto 0);
        d_in        : in    std_logic_vector(n_bits-1 downto 0);
        d_out       : out   std_logic_vector(n_bits-1 downto 0));

end RAM;

---- Arquitectura ----

architecture behavioral of RAM is
    type t_mem is array (n_dir-1 downto 0) of std_logic_vector(n_bits-1 downto 0);
    signal  memoria : t_mem;

begin --behavioral

    mRAM : process(clk)
    begin

        if clk'event and clk = '1' then
            if wr = '1' then
                memoria(to_integer(unsigned(addr))) <= d_in;
            end if;
        end if;
    end process mRAM;


    d_out <= memoria(to_integer(unsigned(addr)));
end behavioral;
```

A.5. Memoria ROM

Código A.5: Código que implementa la Memoria ROM

```
-- ICAI-RiSC-16 code.
-- Archivo fuente: .\cronos.asm.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity ROM is
  port(
    clk: in std_logic; -- La ROM es sincrona
    en_pc: in std_logic; -- Y tiene un enable
    dir: in std_logic_vector(7 downto 0); -- Bus de direcciones
    dat: out std_logic_vector(15 downto 0) ); -- Salida de datos
end ROM;

architecture Behavioural of ROM is
  -- Se declara un tipo de datos para albergar la memoria de programa
  type mem_t is array (0 to 255) of std_logic_vector(15 downto 0);
  signal memoria : mem_t:= (-- Se crea la senal memoria con el contenido del programa.
    16#0000# => X"6601",
    16#0001# => X"2480", -- reset:  la   r1, 0x8040     #Apunto al TCON
    16#0002# => X"210a", --     addi r2, r0, 0x0A   #POSTscaler a 5
    16#0003# => X"8500", --     sw   r2, r1, 0
    16#0004# => X"6802",
    16#0005# => X"2900", --     la   r2, 0x0080         #Simulacion 256 cuentas
    16#0006# => X"8501", --     sw   r2, r1, 1      # cuentas 2*10^6
    16#0007# => X"6601",
    16#0008# => X"2490", --     la   r1, 0x8050         #Apunto a TXRXCON
    16#0009# => X"2104", --     addi r2, r0, 4      #Paridad par
    16#000a# => X"8502", --     sw   r2, r1, 2
    16#000b# => X"7c00",
    16#000c# => X"3fa0", --     la   r7, 0x0020     #Inicializo el puntero a pila en
   mitad de memoria
    16#000d# => X"20b0", --     addi r1, r0, 0x30   #Inicializamos con ceros
    16#000e# => X"8080", --     sw   r1, r0, 0      #Unidades centesimas
    16#000f# => X"8081", --     sw   r1, r0, 1      #Decenas    centesimas
    16#0010# => X"212c", --     addi r2, r0, 0x2c     # Coma
    16#0011# => X"8102", --     sw   r2, r0, 2
    16#0012# => X"8083", --     sw   r1, r0, 3      #Unidades segundos
    16#0013# => X"8084", --     sw   r1, r0, 4      #Decenas    segundos
    16#0014# => X"213a", --     addi r2, r0, 0x3a     # Dos puntos
    16#0015# => X"8105", --     sw   r2, r0, 5
    16#0016# => X"8086", --     sw   r1, r0, 6      #Unidades minutos
    16#0017# => X"8087", --     sw   r1, r0, 7      #Decenas    minutos
    16#0018# => X"6601",
    16#0019# => X"24b0", -- main: la   r1, 0x8070     #Leo los interruptores
    16#001a# => X"a500", --     lw   r2, r1, 0
    16#001b# => X"2181", --     addi r3, r0, 1      # Compruebo el mas bajo
    16#001c# => X"09a4", --     nand r2, r2, r3       # Las dos nand equivalen a una
   and
    16#001d# => X"0924", --     nand r2, r2, r2
    16#001e# => X"6601",
    16#001f# => X"2480", --     la   r1, 0x8040       #Apunto al timer
    16#0020# => X"a600", --     lw   r4, r1, 0      #Traigo la configuracion
```

```
53      16#0021# => X"c805", --     beq  r2, r0, toff     #Miro si lo pongo a 0 o a 1
54      16#0022# => X"1244", -- ton:  nand r4, r4, r4     #r4 != 1
55      16#0023# => X"0db4", --     nand r3, r3, r3
56      16#0024# => X"11c4", --     nand r4, r4, r3
57      16#0025# => X"8600", --     sw   r4, r1, 0
58      16#0026# => X"c005", --     beq  r0, r0, uart
59      16#0027# => X"6fff",
60      16#0028# => X"2dbe", -- toff:   la   r3, 0xFFFE         #r4 &= FFFE
61      16#0029# => X"0e44", --     nand r4, r3, r4
62      16#002a# => X"1244", --     nand r4, r4, r4
63      16#002b# => X"8600", --     sw   r4, r1, 0
64      16#002c# => X"6601",
65      16#002d# => X"2490", -- uart: la   r1, 0x8050         #Leemos la UART
66      16#002e# => X"a502", --     lw   r2, r1, 2
67      16#002f# => X"2182", --     addi r3, r0, 2          #Comprobamos si se ha recibido
        algo
68      16#0030# => X"09b4", --     nand r3, r2, r3     #AND
69      16#0031# => X"0db4", --     nand r3, r3, r3
70      16#0032# => X"cc11", --     beq  r3, r0, timer
71      16#0033# => X"6fff",
72      16#0034# => X"2dbd", --     la   r3, 0xFFFD     #Borramos el flag
73      16#0035# => X"09a4", --     nand r2, r2, r3
74      16#0036# => X"0924", --     nand r2, r2, r2
75      16#0037# => X"8502", --     sw   r2, r1, 2
76      16#0038# => X"a501", --     lw   r2, r1, 1      #Traemos lo recibido
77      16#0039# => X"6c01",
78      16#003a# => X"2db4", --     la   r3, 0x0074     #Una t devuelve el tiempo
79      16#003b# => X"c981", --     beq  r2, r3, est
80      16#003c# => X"c007", --     beq  r0, r0, timer
81      16#003d# => X"2107", -- est:  addi r2, r0, 7      #Apuntamos a las variables
82      16#003e# => X"6c02",
83      16#003f# => X"2d8f", --     la   r3, send
84      16#0040# => X"ef00", -- looptx: jalr r3, r6
85      16#0041# => X"c802", --     beq  r2, r0, timer
86      16#0042# => X"297f", --     addi r2, r2, -1
87      16#0043# => X"c07c", --     beq  r0, r0, looptx
88      16#0044# => X"6601",
89      16#0045# => X"2480", -- timer:  la   r1, 0x8040     #Apuntamos al timer
90      16#0046# => X"a503", --     lw   r2, r1, 3      #Traemos el registro de flags
91      16#0047# => X"2181", --     addi r3, r0, 1      #Miramos si ha terminado de contar
92      16#0048# => X"09b4", --     nand r3, r2, r3     # &= 1
93      16#0049# => X"0db4", --     nand r3, r3, r3
94      16#004a# => X"cc4d", --     beq  r3, r0, main
95      16#004b# => X"6fff",
96      16#004c# => X"2dbe", --     la   r3, 0xFFFE     #Borramos el flag
97      16#004d# => X"09a4", --     nand r2, r2, r3
98      16#004e# => X"0924", --     nand r2, r2, r2
99      16#004f# => X"8503", --     sw   r2, r1, 3
100     16#0050# => X"7400",
101     16#0051# => X"36b0", -- uncen:  la   r5, 0x0030     #Para poner los ceros
102     16#0052# => X"a080", --     lw   r1, r0, 0      #Leemos unidades de centesimas
103     16#0053# => X"2481", --     addi r1, r1, 1
104     16#0054# => X"213a", --     addi r2, r0, 0x3A
105     16#0055# => X"0538", --     sltu r3, r1, r2
106     16#0056# => X"cc02", --     beq  r3, r0, deccen
107     16#0057# => X"8080", --     sw   r1, r0, 0
108     16#0058# => X"c028", --     beq  r0, r0, disp
109     16#0059# => X"8280", -- deccen: sw   r5, r0, 0
110     16#005a# => X"a081", --     lw   r1, r0, 1      #leemos decenas de centesima
```

```
111     16#005b# => X"2481", --     addi r1, r1, 1
112     16#005c# => X"213a", --     addi r2, r0, 0x3A
113     16#005d# => X"0538", --     sltu r3, r1, r2
114     16#005e# => X"cc02", --     beq  r3, r0, unseg
115     16#005f# => X"8081", --     sw   r1, r0, 1
116     16#0060# => X"c020", --     beq  r0, r0, disp
117     16#0061# => X"8281", -- unseg:  sw   r5, r0, 1
118     16#0062# => X"a083", --     lw   r1, r0, 3      #leemos unidades de segundo
119     16#0063# => X"2481", --     addi r1, r1, 1
120     16#0064# => X"213a", --     addi r2, r0, 0x3A
121     16#0065# => X"0538", --     sltu r3, r1, r2
122     16#0066# => X"cc02", --     beq  r3, r0, decseg
123     16#0067# => X"8083", --     sw   r1, r0, 3
124     16#0068# => X"c018", --     beq  r0, r0, disp
125     16#0069# => X"8283", -- decseg: sw   r5, r0, 3
126     16#006a# => X"a084", --     lw   r1, r0, 4      #leemos decenas de segundo
127     16#006b# => X"2481", --     addi r1, r1, 1
128     16#006c# => X"2136", --     addi r2, r0, 0x36
129     16#006d# => X"0538", --     sltu r3, r1, r2
130     16#006e# => X"cc02", --     beq  r3, r0, unmin
131     16#006f# => X"8084", --     sw   r1, r0, 4
132     16#0070# => X"c010", --     beq  r0, r0, disp
133     16#0071# => X"8284", -- unmin:  sw   r5, r0, 4
134     16#0072# => X"a086", --     lw   r1, r0, 6      #leemos unidades de minuto
135     16#0073# => X"2481", --     addi r1, r1, 1
136     16#0074# => X"213a", --     addi r2, r0, 0x3A
137     16#0075# => X"0538", --     sltu r3, r1, r2
138     16#0076# => X"cc02", --     beq  r3, r0, decmin
139     16#0077# => X"8086", --     sw   r1, r0, 6
140     16#0078# => X"c008", --     beq  r0, r0, disp
141     16#0079# => X"8286", -- decmin: sw   r5, r0, 6
142     16#007a# => X"a087", --     lw   r1, r0, 7      #leemos decenas de minuto
143     16#007b# => X"2481", --     addi r1, r1, 1
144     16#007c# => X"2136", --     addi r2, r0, 0x36
145     16#007d# => X"0538", --     sltu r3, r1, r2
146     16#007e# => X"cc02", --     beq  r3, r0, disp
147     16#007f# => X"8284", --     sw   r5, r0, 4
148     16#0080# => X"c000", --         beq  r0, r0, disp
149     16#0081# => X"6601",
150     16#0082# => X"24a0", -- disp: la   r1, 0x8060     # Apunto a los displays
151     16#0083# => X"6e01",
152     16#0084# => X"2da8", --     la   r3, 0x8068
153     16#0085# => X"0020", --     add  r2, r0, r0
154     16#0086# => X"aa00", -- loopd:  lw   r4, r2, 0          #Leemos la RAM
155     16#0087# => X"8600", --     sw   r4, r1, 0      #Escribimos en el display
156     16#0088# => X"2481", --     addi r1, r1, 1      #Movemos el puntero de display
157     16#0089# => X"2901", --     addi r2, r2, 1      #Movemos el puntero de variable
158     16#008a# => X"c581", --     beq  r1, r3, fin
159     16#008b# => X"c07a", --     beq  r0, r0, loopd
160     16#008c# => X"6400",
161     16#008d# => X"2498", -- fin:  la   r1, main
162     16#008e# => X"e400", --     jalr r1, r0
163     16#008f# => X"aa00", -- send:   lw   r4, r2, 0          #En r1 ya apuntamos al
    modulo UART, en r2 nos pasan el puntero de RAM, no tocamos r3, el tx esta libre
164     16#0090# => X"8600", --     sw   r4, r1, 0      #Escribimos en TXREG
165     16#0091# => X"a602", --     lw   r4, r1, 2      #Leemos la configuracion
166     16#0092# => X"2281", --     addi r5, r0, 1      # Literal 1
167     16#0093# => X"1244", --     nand r4, r4, r4     #OREQUAL
168     16#0094# => X"16d4", --     nand r5, r5, r5
```

```
169         16#0095# => X"12c4", --     nand r4, r4, r5
170         16#0096# => X"8602", --     sw   r4, r1, 2      #Lo ponemos en marcha la
        transmision
171         16#0097# => X"a602", -- notf: lw   r4, r1, 2
172         16#0098# => X"2281", --     addi r5, r0, 1      # Literal 1
173         16#0099# => X"12c4", --     nand r4, r4, r5     #ANDEQUAL
174         16#009a# => X"1244", --     nand r4, r4, r4
175         16#009b# => X"d001", --     beq  r4, r0, finsd    #Si se pone a cero hemos acabado
176         16#009c# => X"c07a", --     beq  r0, r0, notf
177         16#009d# => X"f800", -- finsd:  jalr r6,r0
178         16#009e# => X"0000",
179         16#009f# => X"0000",
180         16#00a0# => X"0000",
181         16#00a1# => X"0000",
182         16#00a2# => X"0000",
183         16#00a3# => X"0000",
184         16#00a4# => X"0000",
185         16#00a5# => X"0000",
186         16#00a6# => X"0000",
187         16#00a7# => X"0000",
188         16#00a8# => X"0000",
189         16#00a9# => X"0000",
190         16#00aa# => X"0000",
191         16#00ab# => X"0000",
192         16#00ac# => X"0000",
193         16#00ad# => X"0000",
194         16#00ae# => X"0000",
195         16#00af# => X"0000",
196         16#00b0# => X"0000",
197         16#00b1# => X"0000",
198         16#00b2# => X"0000",
199         16#00b3# => X"0000",
200         16#00b4# => X"0000",
201         16#00b5# => X"0000",
202         16#00b6# => X"0000",
203         16#00b7# => X"0000",
204         16#00b8# => X"0000",
205         16#00b9# => X"0000",
206         16#00ba# => X"0000",
207         16#00bb# => X"0000",
208         16#00bc# => X"0000",
209         16#00bd# => X"0000",
210         16#00be# => X"0000",
211         16#00bf# => X"0000",
212         16#00c0# => X"0000",
213         16#00c1# => X"0000",
214         16#00c2# => X"0000",
215         16#00c3# => X"0000",
216         16#00c4# => X"0000",
217         16#00c5# => X"0000",
218         16#00c6# => X"0000",
219         16#00c7# => X"0000",
220         16#00c8# => X"0000",
221         16#00c9# => X"0000",
222         16#00ca# => X"0000",
223         16#00cb# => X"0000",
224         16#00cc# => X"0000",
225         16#00cd# => X"0000",
226         16#00ce# => X"0000",
```

```
227       16#00cf# => X"0000",
228       16#00d0# => X"0000",
229       16#00d1# => X"0000",
230       16#00d2# => X"0000",
231       16#00d3# => X"0000",
232       16#00d4# => X"0000",
233       16#00d5# => X"0000",
234       16#00d6# => X"0000",
235       16#00d7# => X"0000",
236       16#00d8# => X"0000",
237       16#00d9# => X"0000",
238       16#00da# => X"0000",
239       16#00db# => X"0000",
240       16#00dc# => X"0000",
241       16#00dd# => X"0000",
242       16#00de# => X"0000",
243       16#00df# => X"0000",
244       16#00e0# => X"0000",
245       16#00e1# => X"0000",
246       16#00e2# => X"0000",
247       16#00e3# => X"0000",
248       16#00e4# => X"0000",
249       16#00e5# => X"0000",
250       16#00e6# => X"0000",
251       16#00e7# => X"0000",
252       16#00e8# => X"0000",
253       16#00e9# => X"0000",
254       16#00ea# => X"0000",
255       16#00eb# => X"0000",
256       16#00ec# => X"0000",
257       16#00ed# => X"0000",
258       16#00ee# => X"0000",
259       16#00ef# => X"0000",
260       16#00f0# => X"0000",
261       16#00f1# => X"0000",
262       16#00f2# => X"0000",
263       16#00f3# => X"0000",
264       16#00f4# => X"0000",
265       16#00f5# => X"0000",
266       16#00f6# => X"0000",
267       16#00f7# => X"0000",
268       16#00f8# => X"0000",
269       16#00f9# => X"0000",
270       16#00fa# => X"0000",
271       16#00fb# => X"0000",
272       16#00fc# => X"0000",
273       16#00fd# => X"c07f", -- error:  beq r0,r0,error
274       16#00fe# => X"0000", --     add r0, r0, r0
275       others => X"0000"); -- Para las posiciones sin inicializar
276   begin
277     mem_rom: process(clk)
278     begin
279       if clk'event and clk = '1' then
280         if en_pc = '1' then          dat <= memoria(to_integer(unsigned(dir)));
281         end if;
282       end if;
283     end process mem_rom;
284   end architecture Behavioural;
```

A.6. Banco de Registros de Trabajo

Código A.6: Código que implementa el Banco de Registros

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-11-02
-- BancoReg.vhd

-- Banco de 8 Registros para una RiSC
-- El R0 vale la constante 0
-- El Banco es asincrono en la lectura
-- sincrono en la escritura
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
------ Entidad ------

entity BancoReg is
    generic(
        n_bits : integer := 16);
    port(
        reset_n     : in    std_logic;
        clk         : in    std_logic;
        enable      : in    std_logic;
        dirA        : in    std_logic_vector(2 downto 0);
        dirB        : in    std_logic_vector(2 downto 0);
        dirEsc      : in    std_logic_vector(2 downto 0);
        datoEsc     : in    std_logic_vector(n_bits-1 downto 0);
        a           : out   std_logic_vector(n_bits-1 downto 0);
        b           : out   std_logic_vector(n_bits-1 downto 0));

end BancoReg;

---- Arquitectura ----

architecture behavioral of BancoReg is
    type stdlv_vec is array (7 downto 0) of std_logic_vector(n_bits-1 downto 0);
    signal banco : stdlv_vec;

begin --behavioral

    BR : process(clk, reset_n)
    begin

        if reset_n = '0' then
                banco <= (others => (others=> '0'));

        elsif clk'event and clk = '1' then
            if enable = '1' then
                if dirEsc /= "000" then
                    banco(to_integer(unsigned(dirEsc))) <= datoEsc;
                end if;
            end if;
        end if;
    end process BR;

    a <=    (others => '0') when dirA = "000" else
```

```
55              banco(to_integer(unsigned(dirA)));
56
57      b <=    (others => '0') when dirB = "000" else
58              banco(to_integer(unsigned(dirB)));
59
60  end behavioral;
```

A.7. Registro de ancho N

Código A.7: Código que implementa un registro de ancho genérico

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-11-02
-- Registro_N.vhd

-- Registro de Ancho N
library IEEE;
use IEEE.std_logic_1164.all;

------ Entidad ------

entity Registro_N is
    generic(
        n_bits : integer := 16);
    port(
        reset_n     : in    std_logic;
        clk         : in    std_logic;
        enable      : in    std_logic;
        d           : in    std_logic_vector(n_bits-1 downto 0);
        q           : out   std_logic_vector(n_bits-1 downto 0));

end Registro_N;

---- Arquitectura ----

architecture behavioral of Registro_N is
    signal registro : std_logic_vector(n_bits-1 downto 0);

begin --behavioral
    q <= registro;

    Reg : process(clk, reset_n)
    begin

        if reset_n = '0' then
            registro <= (others => '0');

        elsif clk'event and clk = '1' then
            if enable = '1' then
                registro <= d;
            end if;
        end if;
    end process;

end behavioral;
```

A.8. Código del multiplexor 2 a 1 de N bits

Código A.8: Multiplexor de 2 entradas y una salida con longitud de palabra genérica

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-11-02
-- Multiplexor_2a1_N.vhd

-- Multiplexor de 2 entradas y 1 salida con longitud de palabra generica

library IEEE;
use IEEE.std_logic_1164.all;

------ Entidad ------

entity Multiplexor_2a1_N is
    generic(
        n_bits      : integer := 16);
    port(
        e0  : in    std_logic_vector(n_bits-1 downto 0);
        e1  : in    std_logic_vector(n_bits-1 downto 0);
        sel : in    std_logic;
        s   : out   std_logic_vector(n_bits-1 downto 0));

end Multiplexor_2a1_N;

---- Arquitectura ----

architecture behavioral of Multiplexor_2a1_N is
--Senales

begin --behavioral

    with sel select
        s <=
        e0 when '0',
        e1 when '1',
        (others => '0') when others;

end behavioral;
```

A.9. Código del multiplexor 4 a 1 de N bits

Código A.9: Multiplexor de 4 entradas y una salida con longitud de palabra genérica

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-11-02
-- Multiplexor_4a1_N.vhd

-- Multiplexor de 4 entradas y 1 salida con longitud de palabra generica

library IEEE;
use IEEE.std_logic_1164.all;

------ Entidad ------

entity Multiplexor_4a1_N is
    generic(
        n_bits      : integer := 18);
    port(
        e0  : in    std_logic_vector(n_bits-1 downto 0);
        e1  : in    std_logic_vector(n_bits-1 downto 0);
        e2  : in    std_logic_vector(n_bits-1 downto 0);
        e3  : in    std_logic_vector(n_bits-1 downto 0);
        sel : in    std_logic_vector(1 downto 0);
        s   : out   std_logic_vector(n_bits-1 downto 0));

end Multiplexor_4a1_N;

---- Arquitectura ----

architecture behavioral of Multiplexor_4a1_N is
--Senales

begin --behavioral

    with sel select
        s <=
        e0 when "00",
        e1 when "01",
        e2 when "10",
        e3 when "11",
        (others => '0') when others;

end behavioral;
```

B. Código de los Periféricos

A continuación se incluyen los códigos de los bloques de cada uno de los periféricos del microprocesador.

B.1. Timer

B.1.1. Código estructural del Timer

Código B.1: Código estructural del Timer

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-11-29
-- Timer

-- TIMER PARA EL MICRO ICAI-RISC-16 --
--       CODIGO ESTRUCTURAL         --


library IEEE;
use IEEE.std_logic_1164.all;

---------------INPUT/OUTPUT------------------

entity Timer is
    port(
        clk             : in    std_logic;
        reset_n         : in    std_logic;
        ena             : in    std_logic;
        add             : in    std_logic_vector(1 downto 0); -- 0-TMRCON, 1-TMRPR, 2-
    TMRCNT, 3-XXXX
        data_in         : in    std_logic_vector(15 downto 0);
        data_out        : out   std_logic_vector(15 downto 0));

end Timer;



---------------ARQUITECTURA------------------

architecture structural of Timer is

-----------------------------------
--Declaracion de senales intermedias
-----------------------------------

signal s_mod_int        : std_logic_vector(15 downto 0);
signal s_cnt_int        : std_logic_vector(15 downto 0);
signal enable_int       : std_logic_vector(3 downto 0);
signal co_cont_int      : std_logic;
signal co_glob_int      : std_logic;
signal conf             : std_logic_vector(4 downto 0);
signal conf_int         : std_logic_vector(15 downto 0);
signal timer_flag       : std_logic;
signal timer_flag_int   : std_logic_vector(15 downto 0);
---------------------------
--Declaracion de componentes
---------------------------
```

```
48
49  -- Contador
50  component ContadorNbits is
51      generic(
52          n_bits : integer := 16);
53      port(
54          reset_n     : in    std_logic;
55          clk         : in    std_logic;
56          carga       : in    std_logic;
57          cnt_in      : in    std_logic_vector(n_bits-1 downto 0);
58          enable      : in    std_logic;
59          modulo      : in    std_logic_vector(n_bits-1 downto 0);
60          cnt_out     : out   std_logic_vector(n_bits-1 downto 0);
61          co          : out   std_logic);
62  end component;
63
64  -- Post Scaler
65  component PostScaler
66      port(
67          reset_n     : in    std_logic;
68          clk         : in    std_logic;
69          conf        : in    std_logic_vector(3 downto 0);
70          co_in       : in    std_logic;
71          co_out      : out   std_logic);
72  end component;
73
74  -- Registro Paralelo Paralelo
75  component Registro_N is
76      generic(
77          n_bits : integer := 16);
78      port(
79          reset_n     : in    std_logic;
80          clk         : in    std_logic;
81          enable      : in    std_logic;
82          d           : in    std_logic_vector(n_bits-1 downto 0);
83          q           : out   std_logic_vector(n_bits-1 downto 0));
84  end component;
85
86  -- Multiplexor 4 a 1 de ancho generico
87  component Multiplexor_4a1_N
88      generic(
89          n_bits      : integer := 16);
90      port(
91          e0  : in    std_logic_vector(n_bits-1 downto 0);
92          e1  : in    std_logic_vector(n_bits-1 downto 0);
93          e2  : in    std_logic_vector(n_bits-1 downto 0);
94          e3  : in    std_logic_vector(n_bits-1 downto 0);
95          sel : in    std_logic_vector(1 downto 0);
96          s   : out   std_logic_vector(n_bits-1 downto 0));
97  end component;
98
99  -- Demultiplexor 1 a 4
100 component Demultiplexor_1a4
101     port(
102         e   : in     std_logic;
103         sel : in     std_logic_vector(1 downto 0);
104         s   : out    std_logic_vector(3 downto 0));
105 end component;
106
```

```
107 ----------DESCRIPCION ESTRUCTURAL------------------
108 begin -- structural;
109
110 conf_int(15 downto 5) <= (others => '0');
111 conf_int(4 downto 0) <= conf;
112 timer_flag_int(15 downto 1) <= (others => '0');
113 timer_flag_int(0) <= timer_flag;
114
115 -- Modulo del contador
116 TMRPR : Registro_N
117     port map(
118         reset_n => reset_n,
119         clk     => clk,
120         enable  => enable_int(1),
121         d       => data_in,
122         q       => s_mod_int);
123
124 CON : process(clk,reset_n,co_glob_int, enable_int)
125     begin
126         if reset_n = '0' then
127             conf <= (others => '0');
128             timer_flag <= '0';
129
130         elsif clk'event and clk = '1' then
131             if enable_int(0) = '1' then
132                 conf <= data_in(4 downto 0);
133             elsif enable_int(3) = '1' then
134                 timer_flag <= data_in(0);
135             end if;
136             if co_glob_int = '1' and conf(0) = '1' then --Si se ha terminado de contar
     activa el flag
137                 timer_flag <= '1';
138             end if;
139
140         end if;
141     end process CON;
142
143 -- Contador
144 CONT : ContadorNbits
145     port map(
146         reset_n => reset_n,
147         clk     => clk,
148         carga   => enable_int(2),
149         cnt_in  => data_in,
150         enable  => conf(0),
151         modulo  => s_mod_int,
152         cnt_out => s_cnt_int,
153         co      => co_cont_int);
154
155 -- PostScaler
156 PSCR : PostScaler
157     port map(
158         reset_n => reset_n,
159         clk     => clk,
160         conf    => conf(4 downto 1),
161         co_in   => co_cont_int,
162         co_out  => co_glob_int);
163
164 -- Multiplexor
```

```
165 MUX : Multiplexor_4a1_N
166     port map(
167         e0      => conf_int,
168         e1      => s_mod_int,
169         e2      => s_cnt_int,
170         e3      => timer_flag_int,
171         sel     => add,
172         s       => data_out);
173 
174 -- Demultiplexor
175 DEMUX : Demultiplexor_1a4
176     port map(
177         e       => ena,
178         sel     => add,
179         s       => enable_int);
180 
181 end structural;
```

B.1.2. Contador de N bits

Código B.2: Contador de N bits para el Timer

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-11-29
-- ContadorNbits.vhd

-- Contador para el Timer del microprocesador ICAI-RiSC-16

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

------ Entidad ------
entity ContadorNbits is
    generic(
        n_bits : integer := 16);
    port(
        reset_n     : in    std_logic;
        clk         : in    std_logic;
        carga       : in    std_logic;
        cnt_in      : in    std_logic_vector(n_bits-1 downto 0);
        enable      : in    std_logic;
        modulo      : in    std_logic_vector(n_bits-1 downto 0);
        cnt_out     : out   std_logic_vector(n_bits-1 downto 0);
        co          : out   std_logic);
end ContadorNbits;

---- Arquitectura ----
architecture behavioral of ContadorNbits is
    signal contador : std_logic_vector(n_bits-1 downto 0);
    constant cero   : std_logic_vector(n_bits-1 downto 0) := (others => '0');

begin --behavioral
    cnt_out <= contador;
    co  <= '1' when contador = cero and enable = '1' else
           '0';

    Cont : process(clk, reset_n)
    begin
        if reset_n = '0' then
            contador <= (others => '0');
        elsif clk'event and clk = '1' then
            if carga = '1' then
                contador <= cnt_in;
            elsif enable = '1' then
                if contador = cero then
                    contador <= modulo;
                else
                    contador <= std_logic_vector(unsigned(contador)-1);
                end if;
            end if;
        end if;
    end process;

end behavioral;
```

B.1.3. Post Scaler

Código B.3: Post Scaler para el Timer

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-11-29
-- PostScaler.vhd

-- Post Scaler para Timer
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

------ Entidad ------

entity PostScaler is
    port(
        reset_n     : in    std_logic;
        clk         : in    std_logic;
        conf        : in    std_logic_vector(3 downto 0);
        co_in       : in    std_logic;
        co_out      : out   std_logic);

end PostScaler;

---- Arquitectura ----

architecture behavioral of PostScaler is
    signal contador : std_logic_vector(15 downto 0);
    signal modulo   : std_logic_vector(15 downto 0);
    signal cero     : std_logic_vector(15 downto 0) := (others => '0');

    -- Demultiplexor para el modulo
    component Demultiplexor_1a16
        port(
            e   : in     std_logic;
            sel : in     std_logic_vector(3 downto 0);
            s   : out    std_logic_vector(15 downto 0));
    end component;

begin --behavioral
    co_out <= '1' when contador = modulo and co_in = '1' else
              '0';

    Cont : process(clk, reset_n)
    begin
        if reset_n = '0' then
            contador <= (others => '0');

        elsif clk'event and clk = '1' then
            if co_in = '1' then
                if contador = cero then
                    contador <= modulo;
                else
                    contador <= std_logic_vector(unsigned(contador)-1);
                end if;
            -- else -- co_in = '0' Resetea la cuenta
```

```
55              --      contador <= modulo;
56              end if;
57          end if;
58      end process;
59
60  -- Demultiplexor
61  DEMUX : Demultiplexor_1a16
62      port map(
63          e       => '1',
64          sel     => conf,
65          s       => modulo);
66
67
68  end behavioral;
```

B.2. Transmisor Receptor Serie

B.2.1. Código Estructural del Transmisor Receptor

Código B.4: Código estructural del Transmisor Receptor Serie

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-09-26
-- TransmisorReceptorSerie

-- TRANSMISOR RECEPTOR SERIE RS 232 --
--     CODIGO ESTRUCTURAL         --
--      Compatible ICAI-RiSC-16     --

-- MAPA DE DIRECCIONES
--      0 TX        W
--      1 RX        R
--      2 TRXCON    RW
--      3 -


library IEEE;
use IEEE.std_logic_1164.all;

---------------INPUT/OUTPUT-----------------

entity TransmisorReceptorSerie is
  port(
    clk       : in  std_logic;
    reset_n          : in    std_logic;

        enable           : in    std_logic;
        address          : in    std_logic_vector(1  downto 0);
        data_in          : in    std_logic_vector(15 downto 0);
        data_out         : out   std_logic_vector(15 downto 0);

        uart_in          : in    std_logic;
        uart_out         : out   std_logic);

end TransmisorReceptorSerie;



---------------ARQUITECTURA-----------------

architecture structural of TransmisorReceptorSerie is

-----------------------------------
--Declaracion de senales intermedias
-----------------------------------

signal rx_flag_int          : std_logic;
signal tx_flag_int          : std_logic;
signal framing_error_int    : std_logic;
signal par_error_int        : std_logic;
signal enable_int           : std_logic_vector(3 downto 0);
signal s_p_int              : std_logic_vector(7 downto 0);
```

```
53  signal conf                 : std_logic_vector(2 downto 0); --ParType & Received &
        Send
54  signal conf_int             : std_logic_vector(7 downto 0);
55  signal data_out_int         : std_logic_vector(7 downto 0);
56  ----------------------------
57  --Declaracion de componentes
58  ----------------------------
59
60  -- Transmisor RS232
61      component TransmisorSerie
62          port(
63              clk             : in    std_logic;
64              reset_n         : in    std_logic;
65              envia           : in    std_logic;
66              par_type        : in    std_logic;
67              enable          : in    std_logic;
68              e_p             : in    std_logic_vector(7 downto 0);
69              s_s             : out   std_logic;
70              tx_flag         : out   std_logic);
71      end component;
72
73  -- Receptor RS232
74      component ReceptorSerie is
75          port(
76              clk             : in    std_logic;
77              reset_n         : in    std_logic;
78              par_type        : in    std_logic;
79              e_s             : in    std_logic;
80              s_p             : out   std_logic_vector(7 downto 0);
81              framing_error   : out   std_logic;
82              par_error       : out   std_logic;
83              rx_flag         : out   std_logic);
84      end component;
85
86  --Multiplexor del dato de salida
87      component Multiplexor_4a1_N is
88          generic(
89              n_bits      : integer := 8);
90          port(
91              e0  : in    std_logic_vector(n_bits-1 downto 0);
92              e1  : in    std_logic_vector(n_bits-1 downto 0);
93              e2  : in    std_logic_vector(n_bits-1 downto 0);
94              e3  : in    std_logic_vector(n_bits-1 downto 0);
95              sel : in    std_logic_vector(1 downto 0);
96              s   : out   std_logic_vector(n_bits-1 downto 0));
97      end component;
98
99  --Demultiplexor para el descodificador de enable
100     component Demultiplexor_1a4 is
101         port(
102             e   : in     std_logic;
103             sel : in     std_logic_vector(1 downto 0);
104             s   : out    std_logic_vector(3 downto 0));
105     end component;
106
107 ----------DESCRIPCION ESTRUCTURAL-----------------
108 begin -- structural;
109
110 data_out(15 downto 8) <= (others=> '0');
```

```
111 data_out( 7 downto 0) <= data_out_int;
112
113 conf_int <= "000"&framing_error_int&par_error_int&conf;
114
115 --REGISTRO DE CONFIGURACION
116     CON : process(clk,reset_n,tx_flag_int)
117     begin
118         if reset_n = '0' then
119             conf <= (others => '0');
120
121         elsif clk'event and clk = '1' then
122             if enable_int(2) = '1' then
123                 conf <= data_in(2 downto 0);
124             else
125                 if tx_flag_int = '1' then --Si se ha terminado de transmitir ponlo a 0
126                     conf(0) <= '0';
127                 elsif rx_flag_int = '1' then --Si se ha recibido algo avisa
128                     conf(1) <= '1';
129                 end if;
130             end if;
131         end if;
132     end process CON;
133
134 -- Transmisor RS232
135     TX : TransmisorSerie
136         port map(
137             clk             => clk,
138             reset_n         => reset_n,
139             envia           => conf(0),
140             par_type        => conf(2),
141             enable          => enable_int(0),
142             e_p             => data_in(7 downto 0),
143             s_s             => uart_out,
144             tx_flag         => tx_flag_int);
145
146 -- Receptor RS232
147     RX : ReceptorSerie
148         port map(
149             clk             => clk,
150             reset_n         => reset_n,
151             par_type        => conf(2),
152             e_s             => uart_in,
153             s_p             => s_p_int,
154             framing_error   => framing_error_int,
155             par_error       => par_error_int,
156             rx_flag         => rx_flag_int);
157
158
159 --Multiplexor del dato de salida
160     MUXOUT : Multiplexor_4a1_N
161         generic map(
162             n_bits          => 8)
163         port map(
164             e0              =>  (others =>'0'),
165             e1              =>  s_p_int,
166             e2              =>  conf_int,
167             e3              =>  (others =>'0'),
168             sel             =>  address,
169             s               =>  data_out_int);
```

```
170
171 --Demultiplexor para el descodificador de enable
172     CSDECOD : Demultiplexor_1a4
173         port map(
174             e               =>  enable,
175             sel             =>  address,
176             s               =>  enable_int);
177
178 end structural;
```

B.2.2. Código Estructural del Transmisor Serie RS-232

Código B.5: Código estructural del Transmisor Serie

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-09-22
-- TransmisorSerie

-- TRANSMISOR SERIE RS 232 --
--   CODIGO ESTRUCTURAL  --

library IEEE;
use IEEE.std_logic_1164.all;

---------------INPUT/OUTPUT------------------

entity TransmisorSerie is
    port(
        clk             : in    std_logic;
        reset_n         : in    std_logic;
        envia           : in    std_logic;
        par_type        : in    std_logic;
        enable          : in    std_logic;
        e_p             : in    std_logic_vector(7 downto 0);
        s_s             : out   std_logic;
        tx_flag         : out   std_logic);

end TransmisorSerie;

---------------ARQUITECTURA------------------

architecture structural of TransmisorSerie is

-----------------------------------
--Declaracion de senales intermedias
-----------------------------------

--Senales de enable de los componentes
    signal despl_int            : std_logic;
    signal enable_par_int       : std_logic;
    signal enable_cntbit_int    : std_logic;
    signal enable_cnt8_int      : std_logic;

--Senales de carry out de los contadores
    signal co_bit_int        : std_logic;
    signal co_int            : std_logic;

    signal paridad_int       : std_logic;
    signal reset_s_par_int   : std_logic;

    signal sel_int           : std_logic_vector(1 downto 0);
    signal carga_int         : std_logic;
    signal s_s_int           : std_logic;

    signal paridad           : std_logic;

----------------------------
```

```
55  --Declaracion de componentes
56  ----------------------------
57
58  --Unidad de Control
59      component UnidadControl_TX is
60          port(
61              --Externas
62              reset_n         : in    std_logic;
63              clk             : in    std_logic;
64              envia           : in    std_logic;
65              co              : in    std_logic;      --Carry ouy de 8
66              co_bit          : in    std_logic;      --Carry out de bit
67              sel             : out   std_logic_vector(1 downto 0);
68              despl           : out   std_logic;
69              enable_par      : out   std_logic;
70              enable_cntbit   : out   std_logic;
71              enable_cnt8     : out   std_logic;
72              reset_s_par     : out   std_logic;
73              tx_flag         : out   std_logic);
74
75      end component;
76
77  --Detector de paridad del byte recibido
78      component DetectorParidad
79          port(
80              e_s     : in  std_logic;
81              reset_n : in  std_logic;
82              clk     : in  std_logic;
83              enable  : in  std_logic;
84              reset_s : in  std_logic;
85              par     : out std_logic);
86
87      end component;
88
89  --Registro Paralelo Serie
90      component RegistroParaleloSerie is
91          generic(
92              n_bits : integer := 8);
93          port(
94              reset_n     : in    std_logic;
95              clk         : in    std_logic;
96              s_s         : out   std_logic;
97              e_p         : in    std_logic_vector(n_bits-1 downto 0);
98              despl       : in    std_logic;
99              carga       : in    std_logic);
100
101     end component;
102
103 --Contador ascendente de 19200 baudios y mitad
104     component ContadorAscMod
105         port(
106             clk, reset_n    : in    std_logic;
107             enable          : in    std_logic;
108             co              : out   std_logic;
109             co_half         : out   std_logic);
110
111     end component;
112
113 --Contador Descendente Modulo 8
```

```
114     component ContadorDescModN
115         generic(
116             n_bits      :       integer := 3;
117             modulo      :       integer := 8);
118         port(
119             clk         : in    std_logic;
120             reset_n     : in    std_logic;
121             enable      : in    std_logic;
122             co          : out   std_logic);
123 
124     end component;
125 
126 --Multiplexor 4 a 1
127     component Multiplexor_4a1 is
128 
129         port(
130             e0  : in    std_logic;
131             e1  : in    std_logic;
132             e2  : in    std_logic;
133             e3  : in    std_logic;
134             sel : in    std_logic_vector(1 downto 0);
135             s   : out   std_logic);
136 
137     end component;
138 
139 --Detector de Flanco
140 
141   component DetectorFlanco
142     port(
143       reset_n   : in  std_logic;
144       e         : in  std_logic;
145       clk       : in  std_logic;
146       s         : out   std_logic);
147   end component;
148 
149 
150 ----------DESCRIPCION ESTRUCTURAL------------------
151 begin -- structural
152 
153 paridad <= paridad_int xor par_type; --Dependiendo del tipo de paridad
154                                      -- 0 = Par     1 = Impar
155 
156 --Unidad de Control
157     i1 : UnidadControl_TX
158         port map(
159             reset_n         =>  reset_n,
160             clk             =>  clk,
161             envia           =>  envia,
162             co              =>  co_int,
163             co_bit          =>  co_bit_int,
164             sel             =>  sel_int,
165             despl           =>  despl_int,
166             enable_par      =>  enable_par_int,
167             enable_cntbit   =>  enable_cntbit_int,
168             enable_cnt8     =>  enable_cnt8_int,
169             reset_s_par     =>  reset_s_par_int,
170             tx_flag         =>  tx_flag);
171 
172 
```

```
173 --Detector de paridad del byte recibido
174     i2 : DetectorParidad
175         port map(
176             e_s             =>  s_s_int,
177             reset_n         =>  reset_n,
178             clk             =>  clk,
179             enable          =>  enable_par_int,
180             reset_s         =>  reset_s_par_int,
181             par             =>  paridad_int);
182 
183 --Registro Serie Paralelo
184     i3 : RegistroParaleloSerie
185         port map(
186             reset_n         => reset_n,
187             clk             => clk,
188             s_s             => s_s_int,
189             e_p             => e_p,
190             despl           => despl_int,
191             carga           => enable);
192 
193 --Contador ascendente de 19200 baudios
194     i4 : ContadorAscMod
195         port map(
196             clk         =>  clk,
197             reset_n     =>  reset_n,
198             enable      =>  enable_cntbit_int,
199             co          =>  co_bit_int,
200             co_half     =>  open);
201 
202 --Contador Descendente Modulo 8
203     i5 : ContadorDescModN
204         port map(
205             clk         =>  clk,
206             reset_n     =>  reset_n,
207             enable      =>  enable_cnt8_int,
208             co          =>  co_int);
209 
210 --Multiplexor 4 a 1
211     i6 : Multiplexor_4a1
212         port map(
213             e0  => '1',
214             e1  => '0',
215             e2  => s_s_int,
216             e3  => paridad,
217             sel => sel_int,
218             s   => s_s);
219 
220 
221 end structural;
```

B.2.3. Código de la Unidad de Control del Transmisor Serie

Código B.6: Código de la Unidad de Control del Transmisor Serie

```
1  -- Jose Javier Gonzalez Ortiz
2  -- Lucia Montero Sanchis
3  -- 2014-09-22
4  -- UnidadControl_TX
5
6  library IEEE;
7  use IEEE.std_logic_1164.all;
8
9  entity UnidadControl_TX is
10   port(
11     --Externas
12     reset_n      : in  std_logic;
13     clk          : in  std_logic;
14     envia        : in  std_logic;
15     co           : in    std_logic;     --Carry ouy de 8
16     co_bit       : in    std_logic;     --Carry out de bit
17     sel          : out   std_logic_vector(1 downto 0);
18     despl        : out   std_logic;
19     enable_par     : out   std_logic;
20     enable_cntbit    : out   std_logic;
21     enable_cnt8    : out   std_logic;
22     reset_s_par    : out   std_logic;
23     tx_flag      : out   std_logic);
24
25 end UnidadControl_TX;
26
27
28 architecture behavioral of UnidadControl_TX is
29
30   type t_estados is (Reposo,CargaByte,EnviaStart,EnviaBit,Desplaza,EnviaParidad,
       EnviaStop,EnviaFin);
31   signal estado_act, estado_sig : t_estados;
32
33 begin -- behavioral
34
35   VarEstado : process(clk, reset_n)
36   begin
37
38     if reset_n = '0' then
39       estado_act <= Reposo;
40     elsif  clk'event and clk = '1' then
41       estado_act <= estado_sig;
42     end if;
43   end process VarEstado;
44
45   TransicionEstado : process(estado_act,envia,co,co_bit)
46   begin
47
48     estado_sig <= estado_act;
49
50     case estado_act is
51
52       when Reposo=>
53         if envia = '1' then
```

```
54            estado_sig <= CargaByte;
55          end if;
56
57        when CargaByte =>
58          estado_sig <= EnviaStart;
59
60        when EnviaStart =>
61          if co_bit = '1' then
62            estado_sig <= EnviaBit;
63          end if;
64
65        when EnviaBit =>
66          if co_bit = '1' then
67            estado_sig <= Desplaza;
68          end if;
69
70        when Desplaza =>
71          if co = '1' then
72            estado_sig <= EnviaParidad;
73          else
74            estado_sig <= EnviaBit;
75          end if;
76
77        when EnviaParidad =>
78          if co_bit = '1' then
79            estado_sig <= EnviaStop;
80          end if;
81
82        when EnviaStop =>
83          if co_bit = '1' then
84            estado_sig <= EnviaFin;
85          end if;
86
87
88        when EnviaFin =>
89          estado_sig <= Reposo;
90
91        when others =>
92          estado_sig <= Reposo;
93
94      end case;
95
96    end process TransicionEstado;
97
98    Salidas : process(estado_act)
99    begin
100
101     sel       <= "00";
102     despl       <= '0';
103     enable_par    <= '0';
104     enable_cntbit   <= '0';
105     enable_cnt8   <= '0';
106     reset_s_par     <= '0';
107     tx_flag     <= '0';
108
109     case estado_act is
110
111       when Reposo=>
112         null;
```

```
113
114         when CargaByte=>
115           reset_s_par <= '1';
116
117         when EnviaStart =>
118           sel <= "01";
119           enable_cntbit <= '1';
120
121         when EnviaBit =>
122           sel <= "10";
123           enable_cntbit <= '1';
124
125         when Desplaza =>
126           sel <= "10";
127           despl <= '1';
128           enable_par <= '1';
129           enable_cnt8 <= '1';
130
131         when EnviaParidad =>
132           sel <= "11";
133           enable_cntbit <= '1';
134
135         when EnviaStop =>
136           sel <= "00";
137           enable_cntbit <= '1';
138
139         when EnviaFin =>
140           tx_flag <= '1';
141
142         when others =>
143           null;
144       end case;
145     end process Salidas;
146
147 end behavioral;
```

B.2.4. Código Estructural del Receptor Serie RS-232

Código B.7: Código estructural del Receptor Serie

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-09-22
-- ReceptorSerie

-- TRANSMISOR SERIE RS 232 --
--   CODIGO ESTRUCTURAL    --

library IEEE;
use IEEE.std_logic_1164.all;

---------------INPUT/OUTPUT------------------

entity ReceptorSerie is
    port(
        clk             : in    std_logic;
        reset_n         : in    std_logic;
        par_type        : in    std_logic;
        e_s             : in    std_logic;
        s_p             : out   std_logic_vector(7 downto 0);
        framing_error   : out   std_logic;
        par_error       : out   std_logic;
        rx_flag         : out   std_logic);

end ReceptorSerie;

---------------ARQUITECTURA------------------

architecture structural of ReceptorSerie is

-----------------------------------
--Declaracion de senales intermedias
-----------------------------------

--Senales de enable de los componentes
    signal enable_despl_int     : std_logic;
    signal enable_par_int       : std_logic;
    signal enable_cntbit_int    : std_logic;
    signal enable_cnt8_int      : std_logic;

--Senales de carry out de los contadores
    signal co_bit_int        : std_logic;
    signal co_hbit_int       : std_logic;
    signal co_int            : std_logic;

    signal paridad_int       : std_logic;
    signal reset_s_par_int   : std_logic;

    signal par_error_int     : std_logic;


----------------------------
--Declaracion de componentes
----------------------------
```

```
55
56  --Unidad de Control
57      component UnidadControl_RX
58          port(
59              --Externas
60              reset_n         : in    std_logic;
61              clk             : in    std_logic;
62              e_s             : in    std_logic;
63              co              : in    std_logic;
64              co_bit          : in    std_logic;
65              co_hbit         : in    std_logic;
66              par             : in    std_logic;
67              enable_despl    : out   std_logic;
68              enable_par      : out   std_logic;
69              enable_cntbit   : out   std_logic;
70              enable_cnt8     : out   std_logic;
71              par_error       : out   std_logic;
72              framing_error   : out   std_logic;
73              reset_s_par     : out   std_logic;
74              rx_flag         : out   std_logic);
75
76      end component;
77
78  --Detector de paridad del byte recibido
79      component DetectorParidad
80          port(
81              e_s     : in  std_logic;
82              reset_n : in  std_logic;
83              clk     : in  std_logic;
84              enable  : in  std_logic;
85              reset_s : in  std_logic;
86              par     : out std_logic);
87
88      end component;
89
90  --Registro Serie Paralelo
91      component RegistroSerieParalelo
92          generic(
93              n_bits  : integer := 8);
94          port(
95              e_s     : in  std_logic;
96              s_p     : out std_logic_vector(n_bits-1 downto 0);
97              reset_n : in  std_logic;
98              clk     : in  std_logic;
99              enable  : in  std_logic);
100
101     end component;
102
103 --Contador ascendente de 19200 baudios y mitad
104     component ContadorAscMod
105         port(
106             clk, reset_n    : in    std_logic;
107             enable          : in    std_logic;
108             co              : out   std_logic;
109             co_half         : out   std_logic);
110
111     end component;
112
113 --Contador Descendente Modulo 8
```

```
114     component ContadorDescModN
115         generic(
116             n_bits      :       integer := 3;
117             modulo      :       integer := 8);
118         port(
119             clk         : in    std_logic;
120             reset_n     : in    std_logic;
121             enable      : in    std_logic;
122             co          : out   std_logic);
123 
124     end component;
125 
126 ----------DESCRIPCION ESTRUCTURAL------------------
127 begin -- structural
128 
129 par_error <= par_error_int xor par_type;
130 
131 --Unidad de Control
132     i1 : UnidadControl_RX
133     port map(
134         reset_n         =>  reset_n,
135         clk             =>  clk,
136         e_s             =>  e_s,
137         co              =>  co_int,
138         co_bit          =>  co_bit_int,
139         co_hbit         =>  co_hbit_int,
140         par             =>  paridad_int,
141         enable_despl    =>  enable_despl_int,
142         enable_par      =>  enable_par_int,
143         enable_cntbit   =>  enable_cntbit_int,
144         enable_cnt8     =>  enable_cnt8_int,
145         par_error       =>  par_error_int,
146         framing_error   =>  framing_error,
147         reset_s_par     =>  reset_s_par_int,
148         rx_flag         =>  rx_flag);
149 
150 --Detector de paridad del byte recibido
151     i2 : DetectorParidad
152     port map(
153         e_s             =>  e_s,
154         reset_n         =>  reset_n,
155         clk             =>  clk,
156         enable          =>  enable_par_int,
157         reset_s         =>  reset_s_par_int,
158         par             =>  paridad_int);
159 
160 --Registro Serie Paralelo
161     i3 : RegistroSerieParalelo
162     port map(
163         e_s         =>  e_s,
164         s_p         =>  s_p,
165         reset_n     =>  reset_n,
166         clk         =>  clk,
167         enable      =>  enable_despl_int);
168 
169 --Contador ascendente de 19200 baudios y mitad
170     i4 : ContadorAscMod
171     port map(
172         clk         =>  clk,
```

```
173         reset_n     =>  reset_n,
174         enable      =>  enable_cntbit_int,
175         co          =>  co_bit_int,
176         co_half     =>  co_hbit_int);
177
178 --Contador Descendente Modulo 8
179     i5 : ContadorDescModN
180     port map(
181         clk         =>  clk,
182         reset_n     =>  reset_n,
183         enable      =>  enable_cnt8_int,
184         co          =>  co_int);
185
186 end structural;
```

B.2.5. Código de la Unidad de Control del Receptor Serie

Código B.8: Código de la Unidad de Control del Receptor Serie

```
1  -- Jose Javier Gonzalez Ortiz
2  -- Lucia Montero Sanchis
3  -- 2014-09-08
4  -- UnidadControl_RX
5
6  library IEEE;
7  use IEEE.std_logic_1164.all;
8
9  entity UnidadControl_RX is
10   port(
11     --Externas
12     reset_n      : in  std_logic;
13     clk          : in  std_logic;
14     e_s          : in    std_logic;
15     co           : in    std_logic;     --Carry ouy de 8
16     co_bit       : in    std_logic;     --Carry out de bit
17     co_hbit      : in    std_logic;     --carry out de medio bit
18     par          : in  std_logic;
19     enable_despl   : out   std_logic;
20     enable_par     : out   std_logic;
21     enable_cntbit    : out   std_logic;
22     enable_cnt8    : out   std_logic;
23     par_error      : out   std_logic;
24     framing_error    : out   std_logic;
25     reset_s_par    : out   std_logic;
26     rx_flag      : out   std_logic);
27
28 end UnidadControl_RX;
29
30
31 architecture behavioral of UnidadControl_RX is
32
33   type t_estados is (Reposo,EspStart,Framing0,EspBit,Desplaza,EspParidad,CheckParidad,
       EspStop,Framing1,FramingError,Esp1);
34   signal estado_act, estado_sig : t_estados;
35   signal set_framing    : std_logic;
36   signal reset_framing  : std_logic;
37   signal get_paridad    : std_logic;
38   signal reset_paridad  : std_logic;
39
40 begin -- behavioral
41
42   VarEstado : process(clk, reset_n)
43   begin
44
45     if reset_n = '0' then
46       estado_act <= Reposo;
47     elsif  clk'event and clk = '1' then
48       estado_act <= estado_sig;
49     end if;
50   end process VarEstado;
51
52   TransicionEstado : process(estado_act,e_s,co,co_bit,co_hbit,par)
53   begin
```

```
54
55      estado_sig <= estado_act;
56
57      case estado_act is
58
59        when Reposo=>
60          if e_s = '0' then
61            estado_sig <= EspStart;
62          end if;
63
64        when EspStart=>
65          if co_hbit = '1' then
66            estado_sig <= Framing0;
67          end if;
68
69        when Framing0=>
70          if e_s = '0' then
71            estado_sig <= EspBit;
72          else
73            estado_sig <= FramingError;
74          end if;
75
76        when EspBit=>
77          if co_bit = '1' then
78            estado_sig <= Desplaza;
79          end if;
80
81        when Desplaza =>
82          if co = '1' then
83            estado_sig <= EspParidad;
84          else
85            estado_sig <= EspBit;
86          end if;
87
88        when EspParidad =>
89          if co_bit = '1' then
90            estado_sig <= CheckParidad;
91          end if;
92
93        when CheckParidad =>
94          estado_sig <= EspStop;
95
96        when EspStop =>
97          if co_bit = '1' then
98            estado_sig <= Framing1;
99          end if;
100
101       when Framing1 =>
102         if e_s = '1' then
103           estado_sig <= Esp1;
104         else
105           estado_sig <= FramingError;
106         end if;
107
108       when FramingError =>
109         estado_sig <= Esp1;
110
111       when Esp1 =>
112         if e_s = '1' then
```

```
113             estado_sig <= Reposo;
114           end if;
115
116         when others =>
117           estado_sig <= Reposo;
118
119       end case;
120
121     end process TransicionEstado;
122
123     Salidas : process(estado_act)
124     begin
125
126       enable_despl  <= '0';
127       enable_par    <= '0';
128       enable_cntbit   <= '0';
129       enable_cnt8   <= '0';
130       reset_s_par   <= '0';
131       set_framing   <= '0';
132       reset_framing <= '0';
133       get_paridad   <= '0';
134       reset_paridad <= '0';
135       rx_flag     <= '0';
136
137       case estado_act is
138
139         when Reposo =>
140           null;
141         when EspStart   =>
142           enable_cntbit <= '1';
143           reset_s_par   <= '1';
144           reset_framing     <= '1';
145           reset_paridad     <= '1';
146
147         when Framing0   =>
148           null;
149
150         when EspBit   =>
151           enable_cntbit <= '1';
152
153         when Desplaza   =>
154           enable_despl <= '1';
155           enable_par  <= '1';
156           enable_cnt8 <= '1';
157
158         when EspParidad   =>
159           enable_cntbit <= '1';
160
161         when CheckParidad   =>
162           enable_par  <= '1';
163
164         when EspStop  =>
165           enable_cntbit <= '1';
166
167         when Framing1   =>
168           get_paridad <= '1';
169           rx_flag   <= '1';
170
171         when FramingError   =>
```

```
172           set_framing   <= '1';
173
174         when Esp1 =>
175           null;
176
177         when others =>
178           null;
179       end case;
180     end process Salidas;
181
182     Paridad : process(reset_n,get_paridad,reset_paridad,par)
183     begin
184       if reset_n = '0' then
185         par_error <= '0';
186       elsif get_paridad = '1' then
187         par_error <= par;
188       elsif reset_paridad = '1' then
189         par_error <= '0';
190       end if;
191     end process Paridad;
192
193     Framing : process(reset_n,set_framing,reset_framing)
194     begin
195       if reset_n = '0' then
196         framing_error <= '0';
197       elsif set_framing = '1' then
198         framing_error <= '1';
199       elsif reset_framing = '1' then
200         framing_error <= '0';
201       end if;
202     end process Framing;
203
204   end behavioral;
```

B.2.6. Código del Detector de Paridad del Transmisor Receptor Serie

Código B.9: Código del Detector de Paridad del Transmisor Receptor Serie

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-09-22
-- DetectorParidad

library IEEE;
use IEEE.std_logic_1164.all;

entity DetectorParidad is
    port(
        e_s     : in  std_logic;
        reset_n : in  std_logic;
        clk     : in  std_logic;
        enable  : in  std_logic;
        reset_s : in  std_logic;
        par     : out std_logic);

end DetectorParidad;

architecture behavioral of DetectorParidad is

    signal paridad : std_logic;

begin --behavioral

    Parity : process(reset_n, clk, enable, e_s, reset_s)
    begin

        if reset_n = '0' then
            paridad <= '0';

        elsif clk'event and clk = '1' and enable = '1' then
            paridad <= paridad xor e_s ;

        elsif clk'event and clk = '1' and reset_s = '1' then
            paridad <= '0';

        end if;
    end process Parity;

    par <= paridad;

end behavioral;
```

B.3. Display Alfanumérico

B.3.1. Código Estructural del Display

Código B.10: Código Estructural del Display alfanumérico

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-10-20
-- DisplayAlfanumerico

--  DISPLAY ALFANUMERICO   --
--       MULTIPLEXADO      --
-- Compatible ICAI-RiSC-16 --

library IEEE;
use IEEE.std_logic_1164.all;

---------------INPUT/OUTPUT------------------

entity DisplayAlfanumerico is
    port(
        clk             : in    std_logic;
        reset_n         : in    std_logic;

        enable          : in    std_logic;
        address         : in    std_logic_vector(2  downto 0);
        data_in         : in    std_logic_vector(15 downto 0);
        data_out        : out   std_logic_vector(15 downto 0);

        digito          : out   std_logic_vector(7  downto 0);
        dig_led         : out   std_logic_vector(16 downto 0)
        );
end DisplayAlfanumerico;

---------------ARQUITECTURA------------------

architecture structural of DisplayAlfanumerico is

-----------------------------------
--Declaracion de senales intermedias
-----------------------------------

--Senales
   type stdlv8_vector is array (integer range <>) of std_logic_vector(7 downto 0);
    signal s_p_int          : stdlv8_vector(7 downto 0);
    signal s_mux_int        : std_logic_vector(7 downto 0);
    signal enable_int       : std_logic_vector(7 downto 0);
    signal co_50k_int       : std_logic;
    signal contador_int     : std_logic_vector(2 downto 0);
    signal digito_int       : std_logic_vector(7 downto 0);
    signal data_out_int     : std_logic_vector(7 downto 0);
----------------------------
--Declaracion de componentes
----------------------------

--Unidad de Control

--RegistroParalelo
```

```
54      component Registro_N is
55          generic(
56              n_bits : integer := 16);
57          port(
58              reset_n      : in    std_logic;
59              clk          : in    std_logic;
60              enable       : in    std_logic;
61              d            : in    std_logic_vector(n_bits-1 downto 0);
62              q            : out   std_logic_vector(n_bits-1 downto 0));
63
64      end component;
65
66  --Look Up Table Ascii 16Seg
67      component AsciiA16Seg is
68        port (
69          clk : in  std_logic;
70          e   : in  std_logic_vector(7 downto 0);     -- Entrada en ASCII
71          s   : out std_logic_vector(16 downto 0));   -- Salida (16 segmentos)
72                                                      -- el bit 0 es el segmento A
73      end component;
74
75  --Contador 50K para conseguir 1kHz
76      component Contador50K is
77          port(
78              clk             : in    std_logic;
79              reset_n         : in    std_logic;
80              enable          : in    std_logic;
81              co              : out   std_logic);
82      end component;
83
84  --Contador generico modulo 8 para multiplexar a 125Hz
85      component ContadorDescModNPar is
86          generic(
87              n_bits      :       integer := 3;
88              modulo      :       integer := 8);
89          port(
90              clk         : in    std_logic;
91              reset_n     : in    std_logic;
92              enable      : in    std_logic;
93              s_p         : out   std_logic_vector(n_bits-1 downto 0));
94      end component;
95
96  --Multiplexor 1 a 8 para los registros visibles
97      component Multiplexor_8a1_N is
98          generic(
99              n_bits      : integer := 16);
100         port(
101             e0  : in    std_logic_vector(n_bits-1 downto 0);
102             e1  : in    std_logic_vector(n_bits-1 downto 0);
103             e2  : in    std_logic_vector(n_bits-1 downto 0);
104             e3  : in    std_logic_vector(n_bits-1 downto 0);
105             e4  : in    std_logic_vector(n_bits-1 downto 0);
106             e5  : in    std_logic_vector(n_bits-1 downto 0);
107             e6  : in    std_logic_vector(n_bits-1 downto 0);
108             e7  : in    std_logic_vector(n_bits-1 downto 0);
109             sel : in    std_logic_vector(2 downto 0);
110             s   : out   std_logic_vector(n_bits-1 downto 0));
111
112     end component;
```

```
113
114 --Demultiplexor para controlar el display activo
115     component Demultiplexor_1a8 is
116         port(
117             e   : in     std_logic;
118             sel : in     std_logic_vector(2 downto 0);
119             s   : out    std_logic_vector(7 downto 0));
120
121     end component;
122
123 ----------DESCRIPCION ESTRUCTURAL-----------------
124 begin -- structural
125
126 data_out(15 downto 8) <= (others => '0');
127 data_out( 7 downto 0) <= data_out_int;
128
129 --- Instancias ---
130
131     --Registros
132         GenReg : for i in 0 to 7 generate
133             REGi : Registro_N
134            generic map(
135               n_bits      => 8)
136                 port map(
137                     reset_n     => reset_n,
138                     clk         => clk,
139                     enable      => enable_int(i),
140                     d           => data_in(7 downto 0),
141                     q           => s_p_int(i));
142         end generate GenReg;
143
144     --Look Up Table Ascii 16Seg
145
146         LUT : AsciiA16Seg
147             port map(
148                 clk => clk,
149                 e   => s_mux_int,
150                 s   => dig_led);
151
152     --Contador 50K para conseguir 1kHz
153         C50 : Contador50K
154             port map(
155                 clk             => clk,
156                 reset_n         => reset_n,
157                 enable          => '1',
158                 co              => co_50k_int);
159
160     --Contador generico modulo 8 para multiplexar a 125Hz
161         C8 : ContadorDescModNPar
162             port map(
163                 clk             => clk,
164                 reset_n         => reset_n,
165                 enable          => co_50k_int,
166                 s_p             => contador_int);
167
168     --Multiplexor 8 a 1 para los registros visibles
169         MUXLUT : Multiplexor_8a1_N
170             generic map(
171                 n_bits          => 8)
```

```
172             port map(
173                 e0              => s_p_int(0),
174                 e1              => s_p_int(1),
175                 e2              => s_p_int(2),
176                 e3              => s_p_int(3),
177                 e4              => s_p_int(4),
178                 e5              => s_p_int(5),
179                 e6              => s_p_int(6),
180                 e7              => s_p_int(7),
181                 sel             => contador_int,
182                 s               => s_mux_int);
183 
184     --Multiplexor 8 a 1 para los registros visibles
185         MUXOUT : Multiplexor_8a1_N
186             generic map(
187                 n_bits          => 8)
188             port map(
189                 e0              => s_p_int(0),
190                 e1              => s_p_int(1),
191                 e2              => s_p_int(2),
192                 e3              => s_p_int(3),
193                 e4              => s_p_int(4),
194                 e5              => s_p_int(5),
195                 e6              => s_p_int(6),
196                 e7              => s_p_int(7),
197                 sel             => address,
198                 s               => data_out_int);
199 
200     --Demultiplexor para controlar el display activo
201         DMX1_8 : Demultiplexor_1a8
202             port map(
203                 e               => '1',
204                 sel             => contador_int,
205                 s               => digito_int);
206 
207     --Demultiplexor de direccion para escribir en el registro
208         ADDRDEM : Demultiplexor_1a8
209             port map(
210                 e               => enable,
211                 sel             => address,
212                 s               => enable_int);
213 
214 digito <= not digito_int;
215 
216 end structural;
```

B.3.2. Contador de 50k

Código B.11: Contador de 50k para el Display

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-10-20
-- Contador50K

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity Contador50K is
  port(
        clk             : in    std_logic;
        reset_n         : in    std_logic;
        enable          : in    std_logic;
        co              : out   std_logic);

end Contador50K;

architecture behavioral of Contador50K is

  signal contador   : std_logic_vector(15 downto 0);

  constant modulo   : unsigned(15 downto 0) := to_unsigned(400000-1, 16);
  --constant modulo   : unsigned(15 downto 0) := to_unsigned(10-1, 16); --Simulacion

begin

  Counter : process(clk,reset_n)
  begin

    if reset_n = '0' then
      contador <= (others => '0');
    elsif clk'event and clk = '1' then
      if enable = '1' then
        if contador  = std_logic_vector(modulo) then
          contador <= (others => '0');
        else
          contador <= std_logic_vector(unsigned(contador)+1);
        end if;
      else
        contador <= (others => '0');
      end if;
    end if;
  end process Counter;

  co    <=  '1' when enable = '1' and contador = std_logic_vector(modulo) else
        '0';

end behavioral;
```

B.3.3. Contador Descendente Genérico

Código B.12: Código del Contador descendente para el Display

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-10-20
-- ContadorDescModNPar

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity ContadorDescModNPar is
  generic(
    n_bits    :     integer := 3;
    modulo    :   integer := 8);
  port(
    clk     : in  std_logic;
    reset_n   : in  std_logic;
    enable    : in  std_logic;
    s_p     : out   std_logic_vector(n_bits-1 downto 0));

end ContadorDescModNPar;

architecture behavioral of ContadorDescModNPar is

  constant maximo : unsigned(n_bits-1 downto 0) := to_unsigned(modulo-1,n_bits);
  constant cero   : unsigned(n_bits-1 downto 0) := to_unsigned(0,n_bits);

  signal contador : unsigned(n_bits-1 downto 0);

begin --behavioral

  s_p <= std_logic_vector(contador);

  Counter : process(clk,reset_n)
  begin

    if reset_n = '0' then
      contador <= maximo;
    elsif clk'event and clk = '1' then

      if  enable = '1' then
        if contador = cero then
          contador <= maximo;
        else
          contador <= contador-1;
        end if;
      end if;


    end if;

  end process Counter;

end behavioral;
```

B.3.4. Decodificador de ASCII a 16 segmentos y punto decimal

Código B.13: Decodificador de ASCII a 16 Segmentos

```
-- Jose Javier Gonzalez Ortiz
-- Lucia Montero Sanchis
-- 2014-10-20
-- AsciiA16Seg

-- Decodificador para un display alfanumerico de 16 segmentos + punto decimal.
-- Se codifica la tabla de conversion ASCII a 16 segmentos + punto decimal
-- Mediante una memoria ROM.

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity AsciiA16Seg is

  port (
    clk : in  std_logic;
    e   : in  std_logic_vector(7 downto 0);     -- Entrada en ASCII
    s   : out std_logic_vector(16 downto 0));   -- Salida
                                                -- el bit 0 es el segmento A
end AsciiA16Seg;

architecture behavioural of AsciiA16Seg is
  type mem_t is array (0 to 255) of std_logic_vector(16 downto 0);
  signal memoria : mem_t := (
    16#01# => X"FFFF"&'1',              -- Todos encendidos
    16#20# => X"0000"&'0',              -- Espacio
    16#21# => X"3000"&'1',              -- !
    16#22# => X"2040"&'0',              -- "
    16#23# => X"0355"&'0',              -- #
    16#24# => X"DD55"&'0',              -- $
    16#25# => X"9977"&'0',              -- %
    16#26# => X"8EC9"&'0',              -- &
    16#27# => X"0040"&'0',              -- '
    16#28# => X"0028"&'0',              -- (
    16#29# => X"0082"&'0',              -- )
    16#2A# => X"00FF"&'0',              -- *
    16#2B# => X"0055"&'0',              -- +
    16#2C# => X"0002"&'0',              -- ,
    16#2D# => X"0011"&'0',              -- -
    16#2E# => X"0000"&'1',              -- .
    16#2F# => X"0022"&'0',              -- /

    16#30# => X"FF22"&'0',              -- 0
    16#31# => X"3020"&'0',              -- 1
    16#32# => X"EE11"&'0',              -- 2
    16#33# => X"FC10"&'0',              -- 3
    16#34# => X"3111"&'0',              -- 4
    16#35# => X"DD11"&'0',              -- 5
    16#36# => X"9F11"&'0',              -- 6
    16#37# => X"F000"&'0',              -- 7
    16#38# => X"FF11"&'0',              -- 8
    16#39# => X"F911"&'0',              -- 9
    16#3A# => X"0044"&'0',              -- :
```

```
55      16#3B# => X"0042"&'0',              -- ;
56      16#3C# => X"0029"&'0',              -- <
57      16#3D# => X"0C11"&'0',              -- =
58      16#3E# => X"0092"&'0',              -- >
59      16#3F# => X"E014"&'1',              -- ?
60
61      16#40# => X"EF30"&'0',              -- @
62      16#41# => X"F311"&'0',              -- A
63      16#42# => X"FC54"&'0',              -- B
64      16#43# => X"CF00"&'0',              -- C
65      16#44# => X"FC44"&'0',              -- D
66      16#45# => X"CF01"&'0',              -- E
67      16#46# => X"C301"&'0',              -- F
68      16#47# => X"DF10"&'0',              -- G
69      16#48# => X"3311"&'0',              -- H
70      16#49# => X"CC44"&'0',              -- I
71      16#4A# => X"3E00"&'0',              -- J
72      16#4B# => X"0329"&'0',              -- K
73      16#4C# => X"0F00"&'0',              -- L
74      16#4D# => X"33A0"&'0',              -- M
75      16#4E# => X"3388"&'0',              -- N
76      16#4F# => X"FF00"&'0',              -- O
77
78      16#50# => X"E311"&'0',              -- P
79      16#51# => X"FF08"&'0',              -- Q
80      16#52# => X"E319"&'0',              -- R
81      16#53# => X"DD11"&'0',              -- S
82      16#54# => X"C044"&'0',              -- T
83      16#55# => X"3F00"&'0',              -- U
84      16#56# => X"0322"&'0',              -- V
85      16#57# => X"330A"&'0',              -- W
86      16#58# => X"00AA"&'0',              -- X
87      16#59# => X"2115"&'0',              -- Y
88      16#5A# => X"CC22"&'0',              -- Z
89      16#5B# => X"4844"&'0',              -- [
90      16#5C# => X"0088"&'0',              -- \
91      16#5D# => X"8444"&'0',              -- ]
92      16#5E# => X"000A"&'0',              -- ^
93      16#5F# => X"0C00"&'0',              -- _
94
95      16#60# => X"0080"&'0',              -- `
96      16#61# => X"0E05"&'0',              -- a (He cambiado un segmento)
97      16#62# => X"0705"&'0',              -- b
98      16#63# => X"0601"&'0',              -- c
99      16#64# => X"0645"&'0',              -- d
100     16#65# => X"0E03"&'0',              -- e
101     16#66# => X"4055"&'0',              -- f
102     16#67# => X"8545"&'0',              -- g
103     16#68# => X"0305"&'0',              -- h
104     16#69# => X"0004"&'0',              -- i
105     16#6A# => X"0644"&'0',              -- j
106     16#6B# => X"006C"&'0',              -- k
107     16#6C# => X"0844"&'0',              -- l
108     16#6D# => X"1215"&'0',              -- m
109     16#6E# => X"0205"&'0',              -- n
110     16#6F# => X"0605"&'0',              -- o
111
112     16#70# => X"8341"&'0',              -- p
113     16#71# => X"8145"&'0',              -- q
```

```
114       16#72# => X"0201"&'0',              -- r
115       16#73# => X"8505"&'0',              -- s
116       16#74# => X"0055"&'0',              -- t
117       16#75# => X"0604"&'0',              -- u
118       16#76# => X"0202"&'0',              -- v
119       16#77# => X"120A"&'0',              -- w
120       16#78# => X"00AA"&'0',              -- x
121       16#79# => X"00A4"&'0',              -- y
122       16#7A# => X"0403"&'0',              -- z
123       16#7B# => X"4845"&'0',              -- {
124       16#7C# => X"0300"&'0',              -- |
125       16#7D# => X"8454"&'0',              -- }
126       16#7E# => X"01A0"&'0',              -- ~
127       16#7F# => X"0000"&'0',              -- DEL (se deja en blanco)
128
129       others => X"0000"&'0');             -- El resto de codigos dejan el display
130                                           -- apagado.
131
132     signal s_i : std_logic_vector(16 downto 0);
133
134   begin  -- behavioural
135
136     mem_rom : process(clk)
137     begin
138       if clk'event and clk = '1' then
139         s_i <= memoria(to_integer(unsigned(e)));
140       end if;
141     end process mem_rom;
142
143     -- La tabla en la memoria ROM se ha generado con el segmento A en el bit
144     -- mas significativo, pero en el circuito el segmento A esta en el bit menos
145     -- significativo, por lo que es necesario darle la vuelta a los bits:
146     INV : for i in 0 to 16 generate
147       s(i) <= s_i(16-i);
148     end generate INV;
149
150   end behavioural;
```

C. Código ensamblador

A continuación se incluye el código ensamblador para el cronómetro:

Código C.1: Código ensamblador del cronómetro

```
#Programa de cronometro
        .org 0x00
        ##################  INIT  ##################
        #Configuro los perifericos
        #Timer
reset:  la   r1, 0x8040         #Apunto al TCON
        addi r2, r0, 0x0A       #POSTscaler a 5
        sw   r2, r1, 0
        la   r2, 0x3d09        #5^6 modulo
        #la   r2, 0x0080         #Simulacion 256 cuentas
        sw   r2, r1, 1          # cuentas 2*10^6

        #UART
        la   r1, 0x8050         #Apunto a TXRXCON
        addi r2, r0, 4          #Paridad par
        sw   r2, r1, 2

        la   r7, 0x0020         #Inicializo el puntero a pila en mitad de memoria

        #Variables de memoria almacenadas en ASCII
        addi r1, r0, 0x30       #Inicializamos con ceros
        sw   r1, r0, 0          #Unidades   centesimas
        sw   r1, r0, 1          #Decenas    centesimas
        addi r2, r0, 0x2c       # Coma
        sw   r2, r0, 2
        sw   r1, r0, 3          #Unidades   segundos
        sw   r1, r0, 4          #Decenas    segundos
        addi r2, r0, 0x3a       # Dos puntos
        sw   r2, r0, 5
        sw   r1, r0, 6          #Unidades   minutos
        sw   r1, r0, 7          #Decenas    minutos

        ################### MAIN ###################

main:   la   r1, 0x8070         #Leo los interruptores
        lw   r2, r1, 0
        addi r3, r0, 1          # Compruebo el mas bajo
        nand r2, r2, r3         # Las dos nand equivalen a una and
        nand r2, r2, r2
        la   r1, 0x8040         #Apunto al timer
        lw   r4, r1, 0          #Traigo la configuracion
        beq  r2, r0, toff       #Miro si lo pongo a 0 o a 1
ton:    nand r4, r4, r4         #r4 != 1
        nand r3, r3, r3
        nand r4, r4, r3
        sw   r4, r1, 0
        beq  r0, r0, uart
toff:   la   r3, 0xFFFE         #r4 &= FFFE
        nand r4, r3, r4
        nand r4, r4, r4
        sw   r4, r1, 0

uart:   la   r1, 0x8050         #Leemos la UART
```

```
54          lw   r2, r1, 2
55          addi r3, r0, 2          #Comprobamos si se ha recibido algo
56          nand r3, r2, r3         #AND
57          nand r3, r3, r3
58          beq  r3, r0, timer
59          #Borramos el flag
60          la   r3, 0xFFFD         #Borramos el flag
61          nand r2, r2, r3
62          nand r2, r2, r2
63          sw   r2, r1, 2
64          #Comprobacion de 't'
65          lw   r2, r1, 1          #Traemos lo recibido
66          la   r3, 0x0074         #Una t devuelve el tiempo
67          beq  r2, r3, est
68          beq  r0, r0, timer
69  est:    addi r2, r0, 7          #Apuntamos a las variables
70          la   r3, send
71  looptx: jalr r3, r6
72          beq  r2, r0, timer
73          addi r2, r2, -1
74          beq  r0, r0, looptx
75
76  timer:  la   r1, 0x8040         #Apuntamos al timer
77          lw   r2, r1, 3          #Traemos el registro de flags
78          addi r3, r0, 1          #Miramos si ha terminado de contar
79          nand r3, r2, r3         # &= 1
80          nand r3, r3, r3
81          beq  r3, r0, main
82
83          la   r3, 0xFFFE         #Borramos el flag
84          nand r2, r2, r3
85          nand r2, r2, r2
86          sw   r2, r1, 3
87
88  uncen:  la   r5, 0x0030         #Para poner los ceros
89          lw   r1, r0, 0          #Leemos unidades de centesimas
90          addi r1, r1, 1
91          addi r2, r0, 0x3A
92          sltu r3, r1, r2
93          beq  r3, r0, deccen
94          sw   r1, r0, 0
95          beq  r0, r0, disp
96
97  deccen: sw   r5, r0, 0
98          lw   r1, r0, 1          #leemos decenas de centesima
99          addi r1, r1, 1
100         addi r2, r0, 0x3A
101         sltu r3, r1, r2
102         beq  r3, r0, unseg
103         sw   r1, r0, 1
104         beq  r0, r0, disp
105
106 unseg:  sw   r5, r0, 1
107         lw   r1, r0, 3          #leemos unidades de segundo
108         addi r1, r1, 1
109         addi r2, r0, 0x3A
110         sltu r3, r1, r2
111         beq  r3, r0, decseg
112         sw   r1, r0, 3
```

```
113         beq  r0, r0, disp
114
115 decseg: sw   r5, r0, 3
116         lw   r1, r0, 4          #leemos decenas de segundo
117         addi r1, r1, 1
118         addi r2, r0, 0x36
119         sltu r3, r1, r2
120         beq  r3, r0, unmin
121         sw   r1, r0, 4
122         beq  r0, r0, disp
123
124 unmin:  sw   r5, r0, 4
125         lw   r1, r0, 6          #leemos unidades de minuto
126         addi r1, r1, 1
127         addi r2, r0, 0x3A
128         sltu r3, r1, r2
129         beq  r3, r0, decmin
130         sw   r1, r0, 6
131         beq  r0, r0, disp
132
133 decmin: sw   r5, r0, 6
134         lw   r1, r0, 7          #leemos decenas de minuto
135         addi r1, r1, 1
136         addi r2, r0, 0x36
137         sltu r3, r1, r2
138         beq  r3, r0, disp
139         sw   r5, r0, 4
140         beq  r0, r0, disp
141
142 disp:   la   r1, 0x8060         # Apunto a los displays
143         la   r3, 0x8068
144         add  r2, r0, r0
145 loopd:  lw   r4, r2, 0          #Leemos la RAM
146         sw   r4, r1, 0          #Escribimos en el display
147         addi r1, r1, 1          #Movemos el puntero de display
148         addi r2, r2, 1          #Movemos el puntero de variable
149         beq  r1, r3, fin
150         beq  r0, r0, loopd
151 fin:    la   r1, main
152         jalr r1, r0
153
154
155
156         ################ ENVIAR ##########################
157 send:   lw   r4, r2, 0          #En r1 ya apuntamos al modulo UART, en r2 nos pasan el
        puntero de RAM, no tocamos r3, el tx esta libre
158         sw   r4, r1, 0          #Escribimos en TXREG
159         lw   r4, r1, 2          #Leemos la configuracion
160         addi r5, r0, 1          # Literal 1
161         nand r4, r4, r4         #OREQUAL
162         nand r5, r5, r5
163         nand r4, r4, r5
164         sw   r4, r1, 2          #Lo ponemos en marcha la transmision
165 notf:   lw   r4, r1, 2
166         addi r5, r0, 1          # Literal 1
167         nand r4, r4, r5         #ANDEQUAL
168         nand r4, r4, r4
169         beq  r4, r0, finsd      #Si se pone a cero hemos acabado
170         beq  r0, r0, notf
```

```
171 finsd:  jalr r6,r0
172
173
174         .org 0xFD
175 error:  beq r0,r0,error
176
177         #genera sitio en la ROM (255 palabras)
178         .org 0xFE
179         add r0, r0, r0
```

D. Código de la Simulación

Código D.1: Código del testbench del cronómetro

```
-- Copyright (C) 1991-2013 Altera Corporation
-- Your use of Altera Corporation's design tools, logic functions
-- and other software and tools, and its AMPP partner logic
-- functions, and any output files from any of the foregoing
-- (including device programming or simulation files), and any
-- associated documentation or information are expressly subject
-- to the terms and conditions of the Altera Program License
-- Subscription Agreement, Altera MegaCore Function License
-- Agreement, or other applicable license agreement, including,
-- without limitation, that your use is for the sole purpose of
-- programming logic devices manufactured by Altera and sold by
-- Altera or its authorized distributors.  Please refer to the
-- applicable agreement for further details.
-- *********************************************************************
-- This file contains a Vhdl test bench template that is freely editable to
-- suit user's needs .Comments are provided in each section to help the user
-- fill out necessary details.
-- *********************************************************************
-- Generated on "11/25/2014 02:52:07"
-- Vhdl Test Bench template for design  :  ICAI_RiSC_16
--
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY IR16 IS
END IR16;
ARCHITECTURE ICAI_RiSC_16_arch OF IR16 IS
-- constants
CONSTANT mensaje : std_logic_vector(7 downto 0) := (X"74");
-- signals
SIGNAL clk : STD_LOGIC := '0';
SIGNAL dig_led : STD_LOGIC_VECTOR(16 DOWNTO 0);
SIGNAL digito : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL reset_n : STD_LOGIC;
SIGNAL s : STD_LOGIC_VECTOR(9 DOWNTO 0);
SIGNAL uart_in : STD_LOGIC;
SIGNAL uart_out : STD_LOGIC;
COMPONENT ICAI_RiSC_16
  PORT (
  clk : IN STD_LOGIC;
  dig_led : OUT STD_LOGIC_VECTOR(16 DOWNTO 0);
  digito : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
  e : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
  reset_n : IN STD_LOGIC;
  s : OUT STD_LOGIC_VECTOR(9 DOWNTO 0);
  uart_in : IN STD_LOGIC;
  uart_out : OUT STD_LOGIC
  );
END COMPONENT;
BEGIN
  i1 : ICAI_RiSC_16
  PORT MAP (
-- list connections between master ports and signals
```

```
55    clk => clk,
56    dig_led => dig_led,
57    digito => digito,
58    e => e,
59    reset_n => reset_n,
60    s => s,
61    uart_in => uart_in,
62    uart_out => uart_out
63    );
64  init : PROCESS
65  -- variable declarations
66  BEGIN
67          -- code that executes only once
68  WAIT;
69  END PROCESS init;
70  clk <= not clk after 10 ns;
71  always : PROCESS
72  -- optional sensitivity list
73  -- (        )
74  -- variable declarations
75  BEGIN
76          e <= (others => '0');
77          uart_in <= '1';
78          reset_n <= '0';
79          wait for 30 ns;
80          reset_n<='1';
81          wait for 30 ns;
82          e(0) <= '1';
83          wait for 1 ms;
84          --e(0) <= '0';
85          wait for 1 ms;
86
87           --Bit Start
88          uart_in <='0';
89          wait for 200 ns;
90
91          --Byte de mensaje
92          for j in 0 to 7 loop
93              uart_in <= mensaje(j);
94              wait for 200 ns;
95          end loop;
96
97          -- Paridad
98          uart_in <= '0';
99          wait for 200 ns;
100
101         -- Bit Stop
102         uart_in <='1';
103         wait for 400 ns;
104         wait for 100 us;
105         assert false report "Fin de la Simulacion" severity failure;
106 WAIT;
107 END PROCESS always;
108 END ICAI_RiSC_16_arch;
```