Arquitectura Hardware de Comunicaciones

PROYECTO PICOBLAZE

ÁNGEL TRUQUE CONTRERAS

Universidad Politécnica de Cartagena

Contenido

1. INTRODUCCIÓN 3
2. DIAGRAMA PICOBLAZE 3
3. PERIFÉRICOS 4
3. INSTRUCCIÓN 11
   a. Modificación herramienta ensambladora (asm_custom.cpp) 11
   b. Modificaciones en el código VHDL 13
4. CÓDIGO ASM 16
5. PRUEBAS REALIZADAS: 22

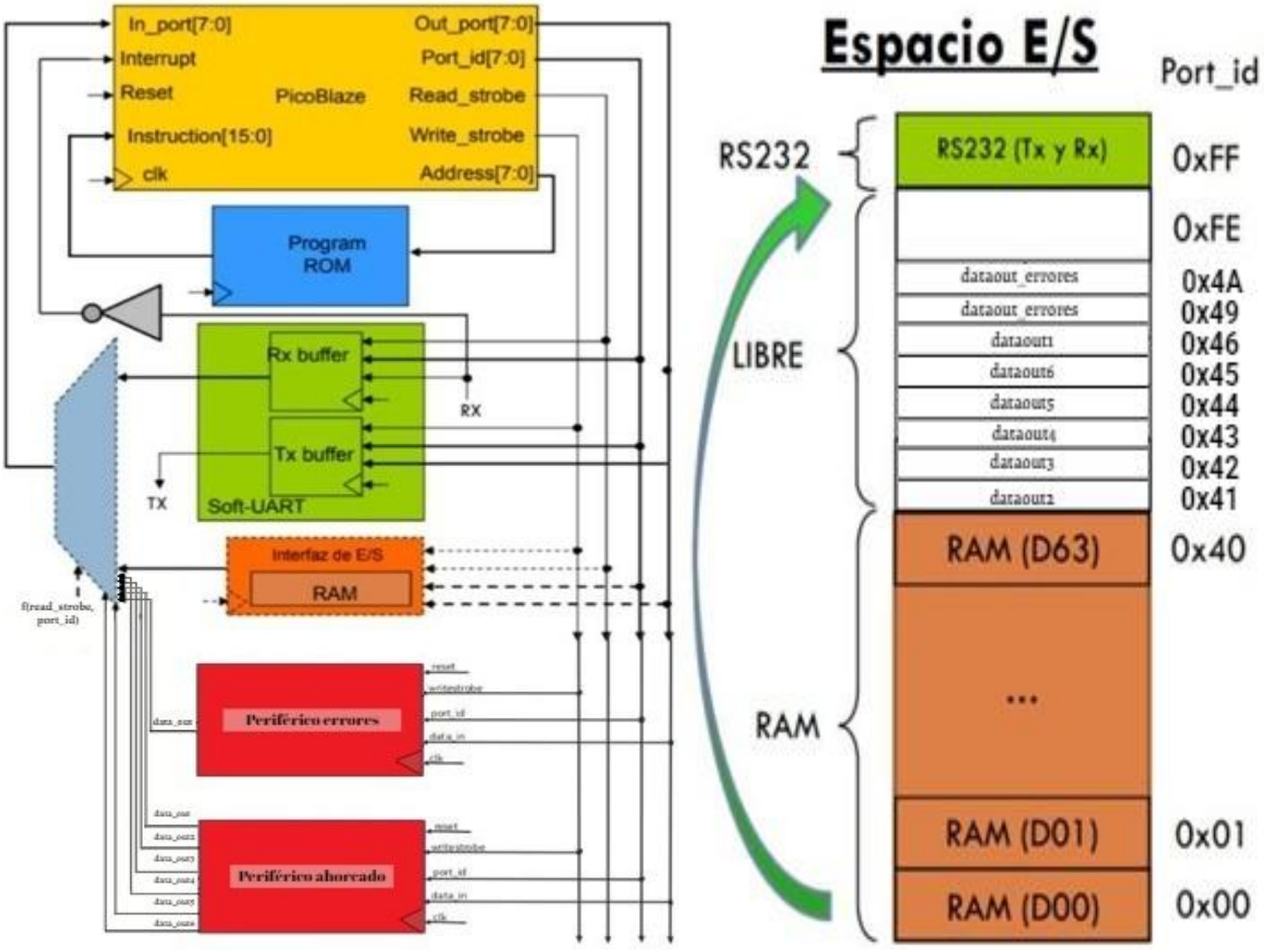
1. INTRODUCCIÓN

En esta memoria se plantea implementar un sistema de computación basado en PicoBlaze, que utilice una conexión RS-232 con un PC convencional para intercambiar datos de forma bidireccional con el usuario. Más específicamente se trata del juego convencional de ahorcado.

2. DIAGRAMA PICOBLAZE

Se han desarrollado dos periféricos a los que hemos llamado "***peri_ahorcado***" y "***peri_contador_errores***", juntos conforman el juego del ahorcado. Además, he incorporado una nueva instrucción "**TR**" que hace de traductora de los resultados, de la que entraremos más en detalle en futuros apartados.

A continuación, el diagrama de la arquitectura actualizada del **PicoBlaze** con estos periféricos. Mencionar que por cuestiones de diseño (se hicieron uso de ciertos puertos como "debug" y otros que se iban a usar como reset pero se descartaron durante el proceso) los puertos **0x47**, **0x48** están disponibles y no son usados.



3. PERIFÉRICOS

1. Periférico peri_ahorcado

Propósito

Este periférico gestiona la lógica principal del juego del ahorcado, incluyendo el almacenamiento de las letras de la palabra secreta y la comparación con las letras introducidas por el jugador.

Arquitectura y Funcionamiento

Puertos de Entrada

data_in: Datos de entrada desde el procesador
port_id: Identificador del puerto para direccionamiento
writestrobe: Señal de escritura del procesador
readstrobe: Señal de lectura del procesador
clk: Señal de reloj del sistema
reset: Señal de reset asíncrono

Puertos de Salida

data_out: Puerto principal de salida con el resultado de comparaciones
data_out2-data_out6: Puertos con las letras (por si fuera necesaria lectura externa)

Registros Internos

letra1-letra4: Almacenan las 4 letras de la palabra secreta
letra5: Almacena la letra introducida por el jugador para comparar
aux_out: Vector de 4 bits que indica qué posiciones han sido adivinadas
resultado: Vector de 8 bits que contiene el estado final de las comparaciones

Mapa de Puertos

Dirección	Función	Tipo
0x41	Escritura de letra1	Escritura
0x42	Escritura de letra2	Escritura
0x43	Escritura de letra3	Escritura
0x44	Escritura de letra4	Escritura
0x45	Escritura de letra a comparar	Escritura
0x46	Captura del resultado	Escritura/Lectura

Lógica de Comparación

Cuando se **escribe una letra en el puerto 0x45**, el periférico automáticamente:

1. **Compara la letra con cada una de las 4 letras almacenadas**
2. **Actualiza** el registro **aux_out** usando lógica OR persistente:

- **Bit 3**: indica si la letra coincide con letra1
- **Bit 2**: indica si la letra coincide con letra2
- **Bit 1**: indica si la letra coincide con letra3
- **Bit 0**: indica si la letra coincide con letra4

3. Una vez que un bit se pone a '1', **permanece así hasta el reset** (que hacemos por **placa**)

2. Periférico peri_contador_errores

Propósito

Este periférico complementa al anterior gestionando el contador de errores del juego, determinando cuándo un intento debe contarse como error basándose en el progreso del jugador.

Arquitectura y Funcionamiento

Puertos

Utiliza la misma interfaz estándar que el anterior, pero con un único puerto de salida **data_out**.

Registros Internos

resultado_actual: Almacena el resultado actual recibido del periférico principal
resultado_anterior: Mantiene el resultado del intento anterior para comparación
contador_errores: Contador de 8 bits que acumula los errores

Mapa de Puertos

Dirección	Función	Tipo
0x41	Recepción del resultado actual	Escritura
0x42	Lectura del contador de errores	Lectura
0x43	Lectura del resultado actual	Lectura

Lógica de Conteo de Errores

El periférico incrementa el contador cuando:

1. **No se adivinó nada**: El resultado es 0000 (ninguna letra coincide)
2. **No hay progreso**: El resultado es igual al anterior (sin nuevas coincidencias)

El contador **NO** se incrementa cuando:

- El resultado es diferente al anterior Y no es 0000 (hay progreso)

Para crear estos periféricos se han seguido los siguientes pasos:

- En el archivo toplevel declaramos ambos periféricos y sus respectivas señales de salida:

```
44  ----------------------------------------------------------------
45  -- Declaracion del periforico ahorcado
46  ----------------------------------------------------------------
47      component peri_ahorcado
48          Port( data_in: in std_logic_vector(7 downto 0);
49              data_out : out std_logic_vector(7 downto 0);
50                  data_out2 : out std_logic_vector(7 downto 0);
51                  data_out3 : out std_logic_vector(7 downto 0);
52                  data_out4 : out std_logic_vector(7 downto 0);
53                  data_out5 : out std_logic_vector(7 downto 0);
54                  data_out6 : out std_logic_vector(7 downto 0);
55              clk : in std_logic;
56                  port_id : in std_logic_vector(7 downto 0);
57                  writestrobe : in std_logic;
58                  readstrobe: std_logic;
59                  reset : in std_logic
60                  );
61      end component;
62
63  ----------------------------------------------------------------
64  -- Declaracion del periferico contador de errores
65  ----------------------------------------------------------------
66      component peri_contador_errores
67          Port( data_in: in std_logic_vector(7 downto 0);
68              data_out : out std_logic_vector(7 downto 0);
69              clk : in std_logic;
70                  port_id : in std_logic_vector(7 downto 0);
71                  writestrobe : in std_logic;
72                  readstrobe: in std_logic;
73                  reset : in std_logic
74                  );
75      end component;
76
77  ----------------------------------------------------------------
78  -- Señales de salida del periferico ahorcado
79  signal dataout1, dataout2, dataout3, dataout4, dataout5 :  std_logic_vector(7 downto 0);
80  signal dataout6 : std_logic_vector(7 downto 0);
81
82  -- Señales de salida del periferico contador de errores
83  signal dataout_errores : std_logic_vector(7 downto 0);
84
```

- A continuación, realizamos el correspondiente mapeo de puertos:

```
138
139   periferico: peri_ahorcado
140     port map(     data_in => outport,
141                   port_id => portid,
142                              writestrobe => writestrobe,
143                              readstrobe => readstrobe,
144                              data_out => dataout1,
145                              data_out2 => dataout2,
146                              data_out3 => dataout3,
147                              data_out4 => dataout4,
148                              data_out5 => dataout5,
149                              data_out6 => dataout6,
150                              reset => reset,
151                    clk => clk);
152
153   periferico_errores: peri_contador_errores
154     port map(     data_in => outport,
155                   port_id => portid,
156                              writestrobe => writestrobe,
157                              readstrobe => readstrobe,
158                              data_out => dataout_errores,
159                              reset => reset,
160                    clk => clk);
161
162   program: programa_ahorcado
163     port map(     address => address,
164                    dout => instruction,
165                    clk => clk);
166
```

- Posteriormente añadimos las salidas de nuestros periféricos al multiplexor.

```
203   -- Multiplexor inport
204   inport <= RAM_out when (readstrobe = '1' and portid<=x"40") else
205             rxbuff_out when (readstrobe = '1' and portid=x"FF") else
206             dataout2 when (readstrobe = '1' and portid=x"41") else --letra1
207             dataout3 when (readstrobe = '1' and portid=x"42") else --letra2
208             dataout4 when (readstrobe = '1' and portid=x"43") else --letra3
209             dataout5 when (readstrobe = '1' and portid=x"44") else --letra4
210             dataout6 when (readstrobe = '1' and portid=x"45") else --letra5
211             dataout1 when (readstrobe = '1' and portid=x"46") else --resultado_comparaciones
212             dataout_errores when (readstrobe = '1' and portid=x"49") else --contador_errores
213             dataout_errores when (readstrobe = '1' and portid=x"4A") else --resultado_actual
214             x"00";
```

- Finalmente creamos las entidades de nuestros periféricos, con los puertos de entrada y salida que se han descrito y mostrado en el diagrama del picoblaze anteriormente, y la arquitectura que describe su funcionamiento, que ha sido explicado anteriormente.

peri_ahorcado (comparador de letras):

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity peri_ahorcado is
  port (
    data_in     : in  std_logic_vector(7 downto 0);
    port_id     : in  std_logic_vector(7 downto 0);
    writestrobe : in  std_logic;
    readstrobe  : in  std_logic;                    --
    clk         : in  std_logic;
    reset       : in  std_logic;
    data_out    : out std_logic_vector(7 downto 0); -- Puerto principal de salida
    data_out2   : out std_logic_vector(7 downto 0); -- letra1
    data_out3   : out std_logic_vector(7 downto 0); -- letra2
    data_out4   : out std_logic_vector(7 downto 0); -- letra3
    data_out5   : out std_logic_vector(7 downto 0); -- letra4
    data_out6   : out std_logic_vector(7 downto 0)  -- resultado comparacion
  );
end entity;

architecture Behavioral of peri_ahorcado is
  signal letra1     : std_logic_vector(7 downto 0) := (others => '0');
  signal letra2     : std_logic_vector(7 downto 0) := (others => '0');
  signal letra3     : std_logic_vector(7 downto 0) := (others => '0');
  signal letra4     : std_logic_vector(7 downto 0) := (others => '0');
  signal letra5     : std_logic_vector(7 downto 0) := (others => '0');
  signal aux_out    : std_logic_vector(3 downto 0) := (others => '0');
  signal resultado  : std_logic_vector(7 downto 0) := (others => '0');

begin

  -- Proceso unificado de escritura
  write_process: process(reset, clk)
  begin
    if reset = '1' then
      letra1 <= (others => '0');
      letra2 <= (others => '0');
      letra3 <= (others => '0');
      letra4 <= (others => '0');
      letra5 <= (others => '0');
      aux_out <= (others => '0');
      resultado <= (others => '0');
    elsif rising_edge(clk) then
      if writestrobe = '1' then
        case port_id is
          when x"41" => letra1 <= data_in;
          when x"42" => letra2 <= data_in;
          when x"43" => letra3 <= data_in;
          when x"44" => letra4 <= data_in;
          when x"45" =>
            letra5 <= data_in;
            -- Realizar comparacion inmediatamente despues de recibir letra5
            -- Bit 0: letra1 coincide
            if (letra1 = data_in) or (aux_out(3) = '1') then
              aux_out(3) <= '1';
            end if;
            -- Bit 1: letra2 coincide
            if (letra2 = data_in) or (aux_out(2) = '1') then
              aux_out(2) <= '1';
            end if;
            -- Bit 2: letra3 coincide
            if (letra3 = data_in) or (aux_out(1) = '1') then
              aux_out(1) <= '1';
            end if;
            -- Bit 3: letra4 coincide
            if (letra4 = data_in) or (aux_out(0) = '1') then
              aux_out(0) <= '1';
            end if;
          when x"46" =>
            resultado <= "0000" & aux_out;  -- Captura resultado
          when others => null;
        end case;
      end if;
    end if;
  end process;

  -- Asignacion de salidas (siempre disponibles)
  data_out <= resultado;      -- Puerto 0x46: resultado de comparaciones
  data_out2 <= letra1;        -- Puerto 0x41: letra1
  data_out3 <= letra2;        -- Puerto 0x42: letra2
  data_out4 <= letra3;        -- Puerto 0x43: letra3
  data_out5 <= letra4;        -- Puerto 0x44: letra4
  data_out6 <= letra5;        -- Puerto 0x45: letra a comparar

end architecture Behavioral;
```

peri_contador_errores (comparador resultados):

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity peri_contador_errores is
  port (
    data_in     : in  std_logic_vector(7 downto 0);
    port_id     : in  std_logic_vector(7 downto 0);
    writestrobe : in  std_logic;
    readstrobe  : in  std_logic;
    clk         : in  std_logic;
    reset       : in  std_logic;
    data_out    : out std_logic_vector(7 downto 0)  -- Puerto principal de salida
  );
end entity;

architecture Behavioral of peri_contador_errores is
  signal resultado_actual   : std_logic_vector(3 downto 0) := (others => '0');
  signal resultado_anterior : std_logic_vector(3 downto 0) := (others => '0');
  signal contador_errores   : unsigned(7 downto 0) := (others => '0');
  -- signal reset_contador     : std_logic := '0';

begin

  -- Proceso principal de control de errores
  error_process: process(reset, clk)
  begin
    if reset = '1' then
      resultado_actual <= (others => '0');
      resultado_anterior <= (others => '0');
      contador_errores <= (others => '0');
      -- reset_contador <= '0';
    elsif rising_edge(clk) then
      if writestrobe = '1' then
        case port_id is
          -- Puerto 0x47: recibe resultado actual del periferico ahorcado
          when x"47" =>
            resultado_actual <= data_in(3 downto 0);

            -- L◆gica de incremento de errores:
            -- 1. Si resultado es 0000 (no adivino nada) -> incrementar
            -- 2. Si resultado es igual al anterior (no progreso) -> incrementar
            -- 3. Si resultado es diferente y no es 0000 -> no incrementar

            if (data_in(3 downto 0) = "0000") or
               (data_in(3 downto 0) = resultado_anterior and resultado_anterior /= "0000") then
              -- Incrementar contador solo si no ha llegado al maximo
              if contador_errores < 255 then
                contador_errores <= contador_errores + 1;
              end if;
            end if;

            -- Actualizar resultado anterior para proxima comparacion
            resultado_anterior <= data_in(3 downto 0);

          -- Puerto 0x48: reset contador de errores (nueva palabra)
          -- when x"48" =>
            -- contador_errores <= (others => '0');
            -- resultado_anterior <= (others => '0');
            -- resultado_actual <= (others => '0');

          when others => null;
        end case;
      end if;
    end if;
  end process;

  -- Proceso de lectura
  read_process: process(port_id, readstrobe, contador_errores, resultado_actual
  begin
    data_out <= (others => '0');  -- Valor por defecto

    if readstrobe = '1' then
      case port_id is
        -- Puerto 0x49: lee contador de errores
        when x"49" =>
          data_out <= std_logic_vector(contador_errores);

        -- Puerto 0x4A: lee resultado actual
        when x"4A" => -- debug
          data_out <= "0000" & resultado_actual;

        when others =>
          data_out <= (others => '0');
      end case;
    end if;
  end process;

end architecture Behavioral;
```

3.INSTRUCCIÓN

He añadido una nueva instrucción, llamada “TR”, al juego de instrucciones del Picoblaze. Esta nueva instrucción actúa como un “traductor” de los resultados recibidos del periférico “peri_ahorcado”. Básicamente le asigna un valor binario al registro sobre el que se aplique dependiendo de sus cuatro bits menos significativos. Estos valores binarios corresponden a un ASCII específico que hace más fácil el interpretar que estamos obteniendo. Sin ella seguiríamos obteniendo valores ASCII pero estos serían más difíciles de interpretar.

Esto es una lista de valores que obtendríamos / los que obtenemos:

0	0	Null
1	1	Start of heading
2	2	Start of text
3	3	End of text
4	4	End of xmit
5	5	Enquiry
6	6	Acknowledge
7	7	Bell
8	8	Backspace
9	9	Horizontal tab
10	0A	Line feed
11	0B	Vertical tab
12	0C	Form feed
13	0D	Carriage feed
14	0E	Shift out
15	0F	Shift in

VS

48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9

65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F

Para añadir esta instrucción he realizado los siguientes cambios:

a. Modificación herramienta ensambladora (asm_custom.cpp)

i. Asignamos el código de operación:

```
68
69   /* tr */ /* added new instruction */
70   char *tr_id = "11111";
71
```

ii. Añadimos la instrucción al juego de instrucciones:

```
135
136 char *instruction_set[] = {
137     "JUMP",     /* 0 */
138     "CALL",     /* 1 */
139     "RETURN",   /* 2 */
140     "LOAD",     /* 3 */
141     "AND",      /* 4 */
142     "OR",       /* 5 */
143     "XOR",      /* 6 */
144     "ADD",      /* 7 */
145     "ADDCY",    /* 8 */
146     "SUB",      /* 9 */
147     "SUBCY",    /* 10 */
148     "SR0",      /* 11 */
149     "SR1",      /* 12 */
150     "SRX",      /* 13 */
151     "SRA",      /* 14 */
152     "RR",       /* 15 */
153     "SL0",      /* 16 */
154     "SL1",      /* 17 */
155     "SLX",      /* 18 */
156     "SLA",      /* 19 */
157     "RL",       /* 20 */
158     "INPUT",    /* 21 */
159     "OUTPUT",   /* 22 */
160     "RETURNI",  /* 23 */
161     "ENABLE",   /* 24 */
162     "DISABLE",  /* 25 */
163     "CONSTANT", /* 26 */
164     "NAMEREG",  /* 27 */
165     "ADDRESS",  /* 28 */
166     "TR"};  /* 29 */ /* added new instruction */
167
```

iii. En mi caso no modifico el número de instrucciones reconocibles, ya que he eliminado la instrucción FLIP

```
92
93  /* increase instruction_count for added new instruction */
94  #define instruction_count 30/* total instruction set */            // Numero m◆ximo de instrucciones
95
```

vi. Añadimos la instrucción al parser del chequeo de sintaxis:

```
433                 case 29: /* TR */ /* added new instruction, same syntax with shift/rotate */
434                     if(op[i].op2 != NULL){
435                         printf("ERROR - Too many Operands for %s on line %d\n", op[i].instruction, i+1);
436                         fprintf(ofp,"ERROR - Too many Operands for %s on line %d\n", op[i].instruction, i+1);
437                         error++;
438                     } else if(op[i].op1 == NULL){
439                         printf("ERROR - Missing operand for %s on line %d\n", op[i].instruction, i+1);
440                         fprintf(ofp,"ERROR - Missing operand for %s on line %d\n", op[i].instruction, i+1);
441                         error++;
442                     }
443                     break;
```

Tras realizar estas modificaciones, compilamos y ejecutamos para crear el archivo .exe que utilizaremos más adelante.

b. Modificaciones en el código VHDL

- Añadimos el código de operación:

```
87    --
88    -- tr
89    constant tr_id : std_logic_vector(4 downto 0) := "11111";
90    --
```

- A continuación, declaramos e instanciamos el componente y definimos el comportamiento de la instrucción.

    - Declaramos el componente:

```
155   --
156   -- Definition of translator process ******************
157   --
158   component tr
159       Port (operand : in std_logic_vector(7 downto 0);
160             Y : out std_logic_vector(7 downto 0);
161             clk : in std_logic);
162     end component;
163   --
```

    - Lo instanciamos:

```
462
463     tr_group: tr
464     port map (operand => sX_register,
465               Y => tr_result,
466               clk => clk);
467
```

- Definimos el comportamiento de la instrucción, para ello creamos el archivo “tr.vhd”:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity tr is
    Port (operand : in std_logic_vector(7 downto 0);
          Y : out std_logic_vector(7 downto 0);
          clk : in std_logic);
end tr;

architecture low_level_definition of tr is
begin
    process (clk)
    begin
        if (clk'event and clk = '1') then
            -- Conversion directa usando los 4 bits inferiores
            case operand(3 downto 0) is
                when "0000" => Y <= "00110000"; -- '0' (ASCII 0x30)
                when "0001" => Y <= "00110001"; -- '1' (ASCII 0x31)
                when "0010" => Y <= "00110010"; -- '2' (ASCII 0x32)
                when "0011" => Y <= "00110011"; -- '3' (ASCII 0x33)
                when "0100" => Y <= "00110100"; -- '4' (ASCII 0x34)
                when "0101" => Y <= "00110101"; -- '5' (ASCII 0x35)
                when "0110" => Y <= "00110110"; -- '6' (ASCII 0x36)
                when "0111" => Y <= "00110111"; -- '7' (ASCII 0x37)
                when "1000" => Y <= "00111000"; -- '8' (ASCII 0x38)
                when "1001" => Y <= "00111001"; -- '9' (ASCII 0x39)
                when "1010" => Y <= "01000001"; -- 'A' (ASCII 0x41)
                when "1011" => Y <= "01000010"; -- 'B' (ASCII 0x42)
                when "1100" => Y <= "01000011"; -- 'C' (ASCII 0x43)
                when "1101" => Y <= "01000100"; -- 'D' (ASCII 0x44)
                when "1110" => Y <= "01000101"; -- 'E' (ASCII 0x45)
                when "1111" => Y <= "01000110"; -- 'F' (ASCII 0x46)
                when others => Y <= "00110000"; -- '0' por defecto
            end case;
        end if;
    end process;
end low_level_definition;
```

- El siguiente paso es declarar las nuevas señales de control:

```
355
356     signal i_tr : std_logic;
357
```

```
390    signal tr_result                : std_logic_vector(7 downto 0);
```

- A continuación realizaremos los cambios necesarios para “register_and_flag_enable.vhd”

  - Habilitamos las señales de habilitación de flags del banco de registros (en picobaze.vhd):

```
187    component register_and_flag_enable
188        Port (i_logical: in std_logic;
189              i_arithmetic: in std_logic;
190              i_shift_rotate: in std_logic;
191              i_tr: in std_logic; --**********
192              i_returni: in std_logic;
193              i_input: in std_logic;
194            active_interrupt : in std_logic;
195            T_state : in std_logic;
196            register_enable : out std_logic;
197            flag_enable : out std_logic;
198            clk : in std_logic);
199    end component;
200    --
```

- Añadimos la correspondiente línea al portmap:

```
484     --
485     reg_and_flag_enables: register_and_flag_enable
486     Port map (i_logical => i_logical,
487                 i_arithmetic => i_arithmetic,
488                 i_shift_rotate => i_shift_rotate,
489                 i_tr => i_tr, --**********************
490                 i_returni => i_returni,
491                 i_input => i_input,
492               active_interrupt => active_interrupt,
493               T_state => T_state,
494               register_enable => register_write_enable,
495               flag_enable => flag_clock_enable,
496               clk => clk);
```

- Añadimos también la instrucción a la entidad (ahora si, dentro del .vhd):

```
16  --
17  entity register_and_flag_enable is
18      Port (i_logical: in std_logic;
19            i_arithmetic: in std_logic;
20            i_shift_rotate: in std_logic;
21            i_tr: in std_logic; -- ***************
22            i_returni: in std_logic;
23            i_input: in std_logic;
24              active_interrupt : in std_logic;
25              T_state : in std_logic;
26              register_enable : out std_logic;
27              flag_enable : out std_logic;
28              clk : in std_logic);
29      end register_and_flag_enable;
30  --
```

- A continuación realizamos el siguiente cambio para que se habilite el registro de instrucciones cuando se ejecute nuestra instrucción:

```
46  -- added new instruction, uncomment this instruction and comment next instruction
47  -- to enable the new instruction
48   reg_instruction_decode <= (i_logical or i_arithmetic or i_shift_rotate or i_input or i_tr) -- *********** NUEVA INSTRUCCION
49                             and (not active_interrupt);
50
```

- Volvemos al archivo picoblaze.vhd y añadimos al decodificador de instrucciones de la UC la nueva instrucción:

```
649
650   -- Añadimos la nueva ****************************************
651
652      i_tr <= '1' when instruction(15 downto 11) = tr_id else '0';
653
```

- Finalmente, añadimos a la ALU la salida de la nueva instrucción:

```
669     ALU_loop: for i in 0 to 7 generate
670     begin
671         ALU_result(i) <= (shift_and_rotate_result(i) and i_shift_rotate)
672                          or (in_port(i) and i_input)
673                          or (arithmetic_result(i) and i_arithmetic)
674                          or (tr_result(i) and i_tr) -- ***************
675                          or (logical_result(i) and i_logical);
676     end generate ALU_loop;
```

4.CÓDIGO ASM

Este código implementa un juego del ahorcado en ensamblador para un procesador PicoBlaze. El programa utiliza comunicación serie RS-232 a 115200 bps para interactuar con un terminal (HyperTerminal) y periféricos hardware para gestionar la lógica del juego.

- **Declaración de constantes y registros**

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;declaracion de constantes y variables
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    CONSTANT        rs232, FF               ; puerto comunicacion serie es el FF
                                            ; rx es el bit 0 del puerto FF(entrada)
                                            ; tx es el bit 7 del puerto FF(salida), esto es porque
    ;el hyperterminal envia primero el LSB, por eso vamos desplazando a la
    ;izquierda al recibir, y al enviar tambien, con lo que enviamos de nuevo
    ;el LSB primero como corresponde para que lo entienda el hyperterminal
    NAMEREG         s1, txreg               ;buffer de transmision
    NAMEREG         s2, rxreg               ;buffer de recepcion
    NAMEREG         s3, contbit             ;contador de los 8 bits de datos
    NAMEREG         s4, cont1               ;contador de retardo1
    NAMEREG         s5, cont2               ;contador de retardo2
    NAMEREG         s7, cont_errores        ;contador de errores (ahora leído del periférico)
    ;
    ADDRESS         00                      ;el programa se cargara a partir de la dir 00
```

- **Programa bucle**

```
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;Inicio del programa
                ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                DISABLE INTERRUPT
start:          CALL            recibe
                ENABLE INTERRUPT
                JUMP start
```

-

- Llama a `recibe` para esperar datos del puerto serie
- Habilita interrupciones
- Repite el ciclo indefinidamente

- **Rutinas de visualización de mensajes**

```
envia_letra:
            LOAD        txreg,0A                ; Carriage Return
            CALL        transmite
            LOAD        txreg,0D                ; Line Feed
            CALL        transmite
            LOAD        txreg,4C                ; "L"
            CALL        transmite
            LOAD        txreg,45                ; "E"
            CALL        transmite
            LOAD        txreg,54                ; "T"
            CALL        transmite
            LOAD        txreg,52                ; "R"
            CALL        transmite
            LOAD        txreg,41                ; "A"
            CALL        transmite
            LOAD        txreg,3A                ; ":"
            CALL        transmite
            LOAD        txreg,20                ; Espacio
            CALL        transmite
            RETURN
```

- Función: Envía el prompt "LETRA: " al terminal
- Secuencia: Carriage Return (0x0A) + Line Feed (0x0D) + "LETRA: "

```
resultado:
            LOAD        txreg,0A                ; Carriage Return
            CALL        transmite
            LOAD        txreg,0D                ; Line Feed
            CALL        transmite
            LOAD        txreg,52                ; "R"
            CALL        transmite
            LOAD        txreg,45                ; "E"
            CALL        transmite
            LOAD        txreg,3A                ; ":"
            CALL        transmite
            LOAD        txreg,20                ; Espacio
            CALL        transmite
            RETURN
```

- Función: Envía "RE: " al terminal
- Uso: Precede la visualización del resultado de comparación

```
; Nueva rutina para mostrar mensaje de muchos errores
demasiados_errores:
            LOAD        txreg,0A                ; Carriage Return
            CALL        transmite
            LOAD        txreg,0D                ; Line Feed
            CALL        transmite
            LOAD        txreg,4D                ; "M"
            CALL        transmite
            LOAD        txreg,55                ; "U"
            CALL        transmite
            LOAD        txreg,43                ; "C"
            CALL        transmite
            LOAD        txreg,48                ; "H"
            CALL        transmite
            LOAD        txreg,4F                ; "O"
            CALL        transmite
            LOAD        txreg,53                ; "S"
            CALL        transmite
            LOAD        txreg,20                ; Espacio
            CALL        transmite
            LOAD        txreg,45                ; "E"
            CALL        transmite
            LOAD        txreg,52                ; "R"
            CALL        transmite
            LOAD        txreg,52                ; "R"
            CALL        transmite
            LOAD        txreg,4F                ; "O"
            CALL        transmite
            LOAD        txreg,52                ; "R"
            CALL        transmite
            LOAD        txreg,45                ; "E"
            CALL        transmite
            LOAD        txreg,53                ; "S"
            CALL        transmite
            LOAD        txreg,21                ; "!"
            CALL        transmite
            JUMP        start                   ; Reiniciar con nueva palabra
```

- Función: Muestra "MUCHOS ERRORES!" cuando se alcanza el límite
- Comportamiento: Después del mensaje, ~~reinicia el juego con `JUMP start`~~ (idea previa) --> usamos reset al llegar al hacer el salto a inicio

```
victoria:
            LOAD        txreg,0A        ; Carriage Return
            CALL        transmite
            LOAD        txreg,0D        ; Line Feed
            CALL        transmite
            LOAD        txreg,57        ; ASCII 'W' = 0x57
            CALL        transmite
            LOAD        txreg,49        ; ASCII 'I' = 0x49
            CALL        transmite
            LOAD        txreg,4E        ; ASCII 'N' = 0x4E
            CALL        transmite
            LOAD        txreg,21        ; ASCII '!' = 0x21
            CALL        transmite
            JUMP        start           ; Reiniciar con nueva palabra
```

- Función: Muestra "WIN!" cuando se adivina la palabra completa
- Comportamiento: Reinicia el juego después de mostrar el mensaje

```
; Nueva rutina para mostrar contador de errores (lee del periférico)
mostrar_errores:
            LOAD        txreg,0A                ; Carriage Return
            CALL        transmite
            LOAD        txreg,0D                ; Line Feed
            CALL        transmite
            LOAD        txreg,45                ; "E"
            CALL        transmite
            LOAD        txreg,52                ; "R"
            CALL        transmite
            LOAD        txreg,52                ; "R"
            CALL        transmite
            LOAD        txreg,4F                ; "O"
            CALL        transmite
            LOAD        txreg,52                ; "R"
            CALL        transmite
            LOAD        txreg,45                ; "E"
            CALL        transmite
            LOAD        txreg,53                ; "S"
            CALL        transmite
            LOAD        txreg,3A                ; ":"
            CALL        transmite
            LOAD        txreg,20                ; Espacio
            CALL        transmite
            ; Leer contador de errores del periférico (puerto 0x49)
            INPUT       cont_errores, 49
            LOAD        txreg, cont_errores
            ADD         txreg, 30               ; Convertir a ASCII (0='0', 1='1', etc.)
            CALL        transmite
            RETURN
```

- Función: Muestra el contador actual de errores
- Funcionamiento:
1. Muestra "ERRORES: "
2. Lee el contador del periférico (puerto 0x49)
3. Convierte a ASCII sumando 0x30
4. Transmite el dígito

- **Lógica principal del juego**

```
check_errores:
            ; Verificar contador de errores del periférico
            INPUT       cont_errores, 49    ; Leer contador actual del periférico
            CALL        mostrar_errores     ; Mostrar contador actual

            ; Verificar si hay demasiados errores (9 o más)
            LOAD        s6, cont_errores
            SUB         s6, 09          ; Comparar con 9
            JUMP        NC, demasiados_errores  ; Si contador >= 9, demasiados errores
            JUMP            bucle_intentos      ; Si no son 9 errores, continuar
```

-Función: Verifica si se ha alcanzado el límite de errores
- Algoritmo:
1. Lee contador de errores del periférico (puerto 0x49)
2. Muestra el contador actual
3. Si errores ≥ 9, va a `demasiados_errores`
4. Si no, continúa con `bucle_intentos`

```
bucle_intentos:
            ; Ahora probar 4 letras
            CALL        envia_letra
            CALL        recibe          ; Recibe letra a probar
            LOAD        txreg,rxreg     ; Muestra la letra recibida
            CALL        transmite
            OUTPUT      rxreg,45        ; Envía letra al comparador
            CALL        wait_1bit       ; Espera
            OUTPUT      rxreg,46        ; Captura el resultado
            CALL        resultado
            INPUT       txreg,46        ; Lee resultado de comparaciones

            ; Enviar resultado al periférico contador de errores (puerto 0x47)
            OUTPUT      txreg, 47       ; El periférico evaluará si incrementar errores

            TR          txreg           ; Mostrar resultado en formato hexadecimal
            LOAD        s6,txreg
            CALL        transmite       ; Muestra resultado
            XOR         s6,46           ; valor hex de "F"
            JUMP        Z,victoria      ; Si es F (resultado XOR = 0), ir a victoria
            JUMP        check_errores   ; Verificar solo contador de errores
```

- Función: Bucle principal de cada intento de adivinanza
- Secuencia:
1. Muestra "LETRA: "
2. Recibe letra del usuario
3. Muestra la letra recibida
4. Envía letra al comparador (puerto 0x45)
5. Captura resultado de comparación (puerto 0x46)
6. Envía resultado al contador de errores (puerto 0x47)
7. Convierte resultado a ASCII hexadecimal con instrucción `TR`
8. Muestra resultado
9. Si resultado = 0xF (todas las letras adivinadas), va a `victoria`
10. Si no, regresa a `check_errores`

- **Rutina de interrupción**

```
interrup:   DISABLE     INTERRUPT
            LOAD        txreg,00
            CALL        inserta_palabra

            ; Reset contador de errores en el periférico para nueva palabra (esto era debug, lo hacemos por placa)
            ; LOAD        s0, 00
            ; OUTPUT      s0, 48          ; Reset contador de errores (puerto 0x48)

            CALL        wait_1bit
            CALL        wait_1bit       ; Espera adicional para estabilizar

            ; Limpiar posible ruido en el buffer de recepción
            ; INPUT       s0, rs232       ; Leer y descartar posible ruido

            ; Recibir las 4 letras de la palabra
            CALL        recibe
            OUTPUT      rxreg,41        ; Guarda letra1
            CALL        wait_1bit

            CALL        recibe
            OUTPUT      rxreg,42        ; Guarda letra2
            CALL        wait_1bit

            CALL        recibe
            OUTPUT      rxreg,43        ; Guarda letra3
            CALL        wait_1bit

            CALL        recibe
            OUTPUT      rxreg,44        ; Guarda letra4
            CALL        wait_1bit

            JUMP        bucle_intentos

            RETURNI     ENABLE
            ADDRESS     FF
            JUMP        interrup
```

- Función: Maneja el inicio de una nueva palabra (rutina de interrupción)
- Algoritmo:
1. Deshabilita interrupciones
2. Llama a `inserta_palabra`
3. Recibe 4 caracteres consecutivos (la palabra secreta)
4. Guarda cada letra en los puertos 0x41-0x44
5. Salta a `bucle_intentos` para comenzar el juego
6. La dirección FF indica el vector de interrupción

```
inserta_palabra:
            LOAD    s6, 00          ;  indice para la RAM
loop_mensaje:
            INPUT   txreg, s6       ; Lee desde RAM[s6]
            ADD     txreg, 00       ; Verifica si es 0
            JUMP    Z, fin_mensaje  ; Si es 0, termina el mensaje
            CALL    transmite       ; Transmite el car cter
            ADD     s6, 01          ; Incrementa  ndice
            JUMP    loop_mensaje    ; Repite el bucle
fin_mensaje:
            RETURN
```

- Función: Transmite contenido de la memoria RAM hasta encontrar un 0
- Funcionamiento: Lee secuencialmente desde RAM[0] → (“índice para la RAM”) y transmite cada byte hasta encontrar 0x00

- **Interacción con periféricos**

El código interactúa con tres periféricos hardware:

1. Periférico comparador (puertos 0x41-0x46):

- 0x41-0x44: Almacena las 4 letras de la palabra
- 0x45: Letra a comparar
- 0x46: Resultado de comparación (bits indican posiciones acertadas)

2. Periférico contador de errores (puertos 0x47-0x49):

- 0x47: Recibe resultado para evaluar si incrementar errores
- 0x49: Lee contador actual de errores

3. Instrucción TR:

- Convierte nibble (4 bits) a carácter ASCII hexadecimal
- 0-9 → '0'-'9', A-F → 'A'-'F'

- **Flujo completo del juego**

1. **Inicialización**: Espera interrupción con nueva palabra

2. **Recepción de palabra**: La interrupción recibe 4 letras y las almacena

3. **Bucle de juego**:

- Solicita letra al usuario
- Compara con la palabra
- Actualiza contador de errores si es necesario
- Muestra resultado
- Verifica condiciones de victoria (resultado = 0xF) o derrota (errores ≥ 9)

4. **Final**: Muestra mensaje apropiado y reinicia

5.PRUEBAS REALIZADAS:

Se han realizado una serie de pruebas para demostrar su debido funcionamiento:

Palabra: toro (todo correcto)

```
PALABRA:
LETRA: t
RE: 8
ERRORES: 0
LETRA: o
RE: D
ERRORES: 0
LETRA: r
RE: F
WIN!
```

Palabra: loro (todo incorrecto)

```
PALABRA:
LETRA: p
RE: 0
ERRORES: 1
LETRA: ±
RE: 0
ERRORES: 2
LETRA: ┤
RE: 0
ERRORES: 3
LETRA: +
RE: 0
ERRORES: 4
LETRA: ß
RE: 0
ERRORES: 5
LETRA: ú
RE: 0
ERRORES: 6
LETRA: 5
RE: 0
ERRORES: 7
LETRA: n
RE: 0
ERRORES: 8
LETRA: m
RE: 0
ERRORES: 9
MUCHOS ERRORES!
```

Palabra: gato (partida normal, con aciertos y errores)

```
PALABRA:
LETRA: p
RE: 0
ERRORES: 1
LETRA:
PALABRA:
LETRA:
PALABRA:
LETRA: g
RE: 8
ERRORES: 0
LETRA: A
RE: 8
ERRORES: 1
LETRA: a
RE: C
ERRORES: 1
LETRA: p
RE: C
ERRORES: 2
LETRA: t
RE: E
ERRORES: 2
LETRA: o
RE: F
WIN!
```

Tabla para los resultados:

Valor real Binario	Valor real Hexadecimal	Valor traducido Hexadecimal	Valor traducido ASCII
0000 0000	0x00	0x30	0
0000 0001	0x01	0x31	1
0000 0010	0x02	0x32	2
0000 0011	0x03	0x33	3
0000 0100	0x04	0x34	4
0000 0101	0x05	0x35	5
0000 0110	0x06	0x36	6
0000 0111	0x07	0x37	7
0000 1000	0x08	0x38	8
0000 1001	0x09	0x39	9
0000 1010	0x0A	0x41	A
0000 1011	0x0B	0x42	B
0000 1100	0x0C	0x43	C
0000 1101	0x0D	0x44	D
0000 1110	0x0E	0x45	E
0000 1111	0x0F	0x46	F